Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

Securing Heritage Spaces: A Machine Learning-Powered Dashboard for Real-time Visitor Management

Gonçalo Sousa 21874

Advisors:
Professor Doutor Renato Panda
Professor Doutor José Casimiro Pereira

2023/2024

Acknowledgments

I would like to express my deepest gratitude to everyone who, in one way or another, contributed to the completion of this project.

First and foremost, I extend my sincere thanks to my supervisors, José Casimiro Pereira and Renato Panda, for their invaluable guidance, support, and encouragement throughout every stage of this project. Their expertise and dedication were crucial to the success of this work.

I am also grateful to **Instituto Politécnico de Tomar** for providing a supportive environment that facilitated the development of this project.

This project would not have been possible without the initiative of Rui Ferreira, responsible for preserving the cultural heritage of the **Convento de Cristo** whose commitment to this task inspired the development of this system. His dedication to safeguarding cultural heritage has provided an invaluable context and motivation for this work.

Abstract

This project focuses on developing an innovative system for monitoring visitor flow in historical monuments, aiming to preserve their structural and cultural integrity. The primary subject of the study is the Convento de Cristo, a renowned monument located in Tomar, Portugal. With increasing tourist numbers, traditional manual methods of visitor counting and management have proven inadequate. The project seeks to replace these methods with a real-time automated system capable of accurately counting and tracking visitors.

To achieve this goal, advanced computer vision algorithms were integrated, namely YOLOv5 for object detection and DeepSORT for real-time object tracking. The system architecture was designed for modularity and scalability, utilizing a Raspberry Pi 5 for video capture and Docker to containerize the machine learning stack. A user-friendly interface was developed using Flask, allowing users to monitor real-time visitor counts, visualize historical data, and manage system configurations with ease.

Throughout the project, extensive testing was conducted using both pre-recorded video samples and live camera feeds to evaluate the system's performance. Sockets were employed to enable efficient communication between the Raspberry Pi and the machine learning stack, ensuring real-time data processing. The system demonstrated the capability to accurately track individuals and adapt to various monitoring scenarios, with an average processing speed of approximately 40 frames per second and a delay of under one second. These results validate the proposed solution's effectiveness for real-time monitoring.

Keywords: Real-Time Visitor Monitoring, Object Detection, YOLOv5, DeepSORT, Computer Vision, Historical Monument Preservation

Glossary

Term	Definition
YOLOv5	You Only Look Once, a real-time object detection algorithm.
DeepSORT	Deep Simple Online and Real-time Tracking, an object tracking algorithm that integrates appearance features for robust tracking.
ByteTrack	A tracking algorithm that leverages both high- and low-confidence detections for improved tracking accuracy.
Flask	A lightweight web framework for Python used to build web applications.
Sockets	A low-level communication endpoint enabling data exchange between processes over a network.
SQLite	A lightweight, serverless relational database management system.
CNN	Convolutional Neural Network, a deep learning model used for image and video recognition tasks.
Kalman Filter	An algorithm that estimates the state of a system from noisy observations, commonly used in motion prediction.
Hungarian Algorithm	An optimization algorithm used to solve the assignment problem, particularly for associating detections with tracked objects.
AFLink	Appearance-Free Link, a statistical model in StrongSORT for associating tracklets without relying on appearance features.
GSI	Gaussian-Smoothed Interpolation, a method in StrongSORT for predicting object positions in frames with missing detections.
Docker	A platform for developing, shipping, and running applications in containers, ensuring consistent environments.
Object Tracking	The process of maintaining the identity of detected objects across successive video frames.

ReID	Person Re-Identification, a technique for matching individuals across different camera views.
CRUD	Create, Read, Update, Delete; basic operations for managing data in a database.
Bounding Box	A rectangular border that highlights detected objects in an image or video.
Polygon Areas	Custom-defined areas within the camera's field of view used for determining object entry or exit.
Visitor Flow	The movement of people through a monitored area, such as a room or a historical site.
Historic Preservation	The practice of protecting and maintaining cultural heritage sites for future generations.

Table 1: Glossary of Key Terms

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The cultural and historical significance of monuments draws visitors from across the globe. However, the increasing influx of tourists poses a pressing challenge to the preservation and sustainability of these iconic structures. The manual methods currently employed for visitor control, primarily relying on human observers to estimate and manage visitor numbers, have become increasingly inadequate in the face of growing tourist footfall.

1.1 Motivation

The motivation behind this project is to address the pressing need for effective visitor management solutions to protect the integrity of culturally significant monuments while improving visitor experience and management efficiency. By integrating machine learning algorithms, this project aims to mitigate the detrimental impact of excessive visitation on the structural integrity and cultural heritage of monuments.

The landmark where all the tests and proofs of concept will take place is the Convento de Cristo (Fig.1.1), situated in the Portuguese city of Tomar.

1.2 The Problem

The existing problem lies in the inadequacies of current manual counting systems, which are often subjective and prone to human error. These methods fail to adapt to dynamic visitor patterns and cannot provide real-time data needed for proactive management. Machine learning algorithms offer an unparalleled opportunity to analyze large datasets, detect patterns, and provide real-time insights into visitor numbers, forecasting potential peak periods and enabling better management strategies.

1.3 Document Structure

This report is structured as follows:

Section 1 provides an overview of the motivation and problem definition.

Section 2 covers the State of the Art related to machine learning algorithms.

Section 3 discusses all the System Analysis done in the project.

Section 4 discusses the implementation process and the challenges faced.

Section 5 details a practical case study with the objective of testing the app while in the early stages of development.

Finally, Section 6 presents the evaluation results and discusses the potential benefits and limitations of the proposed system.

Through this project, we aspire to contribute to the sustainable management of cultural heritage sites, ensuring that future generations can continue to appreciate these enduring symbols of human history.

Figure 1.1: Convento de Cristo

Chapter 2

State of the Art

This section provides an overview of the current research and technologies related to this project. In this section, are presented a summary and analysis of key studies and technologies, an identification of trends and patterns, and an evaluation of the strengths and limitations of existing work.

2.1 Object Detection Models - A comparison

Object detection is a key task in computer vision, where the objective is to automatically detect and identify various objects within an image or video. This process involves two main steps: locating objects by predicting bounding boxes—rectangular areas that define the position and size of each object—and categorizing these detected objects into predefined classes.

Modern object detection algorithms typically rely on machine learning or deep learning techniques, enabling them to achieve high accuracy even in complex and dynamic environments. These algorithms analyse pixel patterns, extract features, and use trained models to make precise predictions about the objects present in a scene.

In the following sections, we will explore some of the most widely used object detection models (Fig.2.1), such as R-CNN (Region-based Convolutional Neural Networks) (Girshick et al., 2013), Fast R-CNN (Girshick, 2015), and YOLO (You Only Look Once) (Terven et al., 2023). Each of these approaches offers unique strengths and trade-offs, making them suitable for different applications, from real-time processing to high-accuracy requirements.

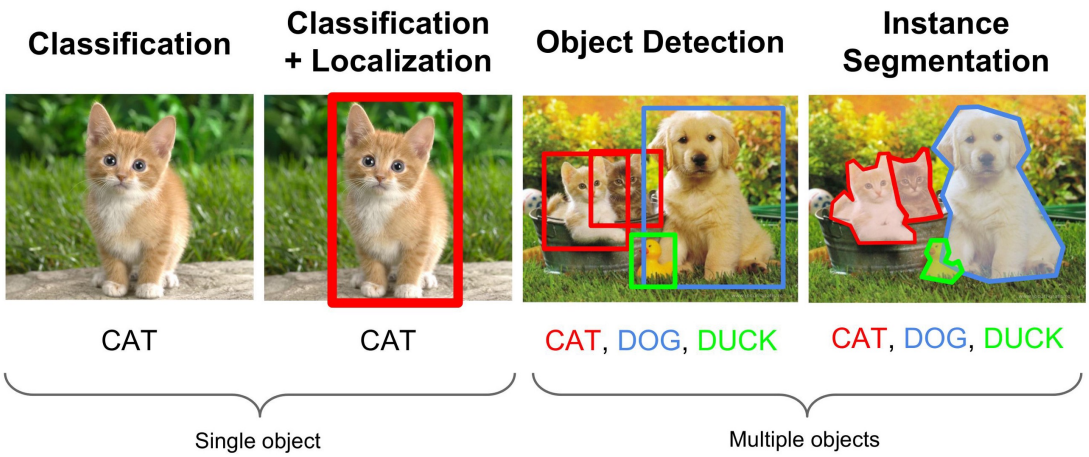

Figure 2.1: Object Detection Example. Credit to (Jaiswal et al., 2021)

2.1.1 Region-based Convolutional Neural Network - R-CNN

The Region-based Convolutional Neural Network (R-CNN) was one of the pioneering approaches to object detection using deep learning. This algorithm employs a technique known as **Selective Search** to extract approximately 2000 candidate regions from the input image, referred to as *region proposals*. These proposals are then fed into a Convolutional Neural Network (CNN) to identify and classify objects within the image (Malhotra and Garg, 2020) (Fig.2.2).

Selective Search:

1. **Generation of candidate regions:** The algorithm initially generates many small candidate regions based on image segmentation.
2. **Merging similar regions:** Recursively, similar smaller regions are combined into larger ones based on colour, texture, size, and shape similarity.
3. **Creation of region proposals:** The final region proposals are formed by merging these similar regions, reducing the number of proposals to around 2000 per image.

Challenges with R-CNN:

- **Slow training process:** The model's training is time-consuming since it must classify approximately 2000 region proposals per image, making it inefficient for large datasets.
- **Real-time performance issues:** R-CNN cannot be implemented in real-time as it takes around 49 seconds to process a single image, making it impractical for real-time applications.
- **Lack of learning in region proposal generation:** The Selective Search algorithm is fixed and does not involve any learning process. Consequently, it may generate suboptimal region proposals that can affect the detection accuracy.

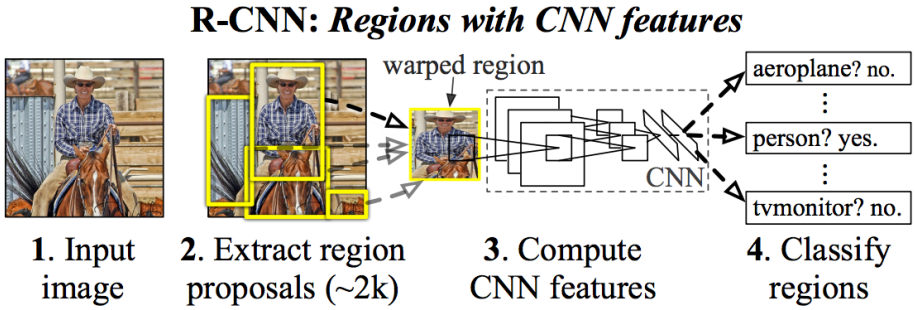

Figure 2.2: Illustration of the R-CNN architecture. Adapted from (Gandhi, 2018)

2.1.2 Fast R-CNN

The development of **Fast R-CNN** was driven by the need to address the long training times and inefficiencies of the original R-CNN. The same author of R-CNN introduced this improved algorithm, which aimed to significantly speed up the object detection process by optimizing the handling of region proposals.

Unlike R-CNN, which processes each region proposal individually through the Convolutional Neural Network (CNN), Fast R-CNN feeds the entire input image into the CNN in a single forward pass. This produces a **convolutional feature map** that captures important visual features from the entire image.

From this convolutional feature map, region proposals are identified using an external algorithm, such as **EdgeBoxes**. The identified regions are then processed by a **Region of Interest (RoI) pooling layer**, which converts the proposals into fixed-size feature vectors. This step ensures that features from each proposed region are mapped into a uniform vector size, regardless of the region's shape or size.

Finally, the output feature vectors are passed through a fully connected network, followed by a **softmax layer** to classify the objects within each region. The network simultaneously predicts the object class and refines the bounding box coordinates for each region proposal (Malhotra and Garg, 2020) (Fig.2.3).

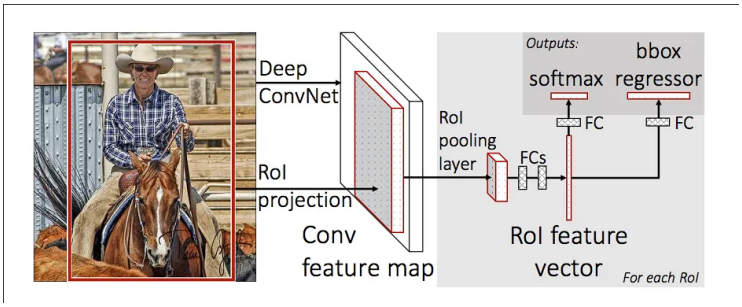

Figure 2.3: Illustration of the Fast R-CNN architecture. Adapted from (Gandhi, 2018)

The primary reason why Fast R-CNN is significantly faster than R-CNN is that it processes the entire image just once through the convolutional layers, rather than running the CNN separately on each of the 2000 region proposals. This reduces the computational overhead and makes the detection process much more efficient.

2.1.3 You Only Look Once - YOLO

The **You Only Look Once (YOLO)** algorithm revolutionized object detection by framing it as a single regression problem rather than treating it as a classification task for multiple region proposals. In YOLO, the input image is divided into an $S \times S$ grid. Each grid cell is responsible for predicting a fixed number of bounding boxes and their associated confidence scores. The confidence score reflects the probability that a bounding box contains an object and how accurate the predicted bounding box is.

For each bounding box, YOLO predicts:

- The coordinates of the bounding box (center x, y, width, and height).
- The confidence score, which indicates the likelihood of an object being present.
- Class probabilities for detecting which class the object belongs to (e.g., person, car, etc.).

During the final step, the algorithm classifies the detected bounding boxes based on the calculated class probabilities. YOLO's unique approach of predicting both the bounding boxes and class probabilities directly from the input image in a single pass allows for real-time object detection (Malhotra and Garg, 2020) (Fig.2.4).

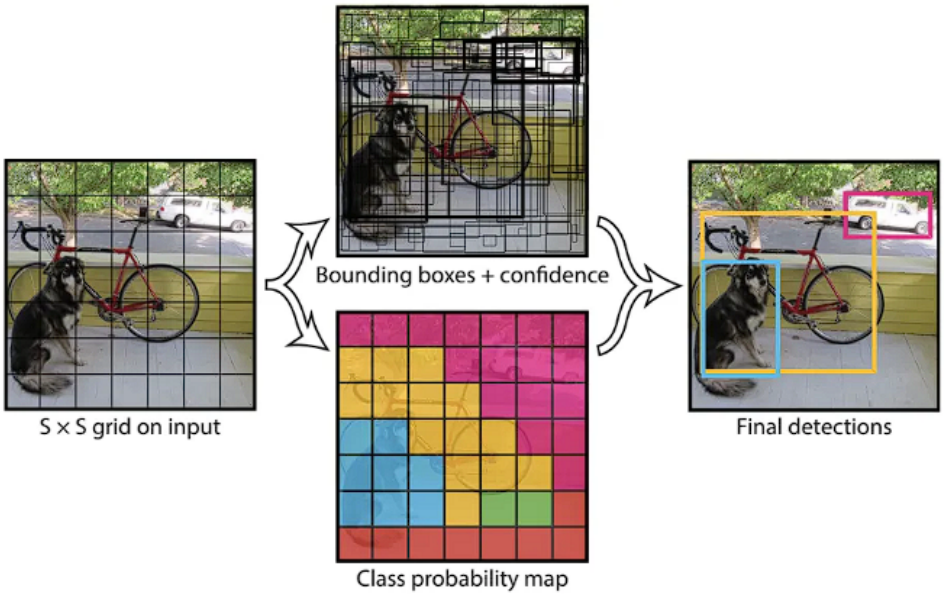

Figure 2.4: Illustration of YOLO's object detection approach. Adapted from (Gandhi, 2018)

For the context of this project, YOLO, more specifically YOLOv5, was chosen as it fulfills all the necessary requirements. The main reasons for this selection are as follows:

- The task only requires identifying people, which are not considered "small objects." YOLO generally struggles with detecting small objects, but this limitation does not impact the objectives of this project.

- The system needs to provide real-time monitoring, requiring an algorithm capable of processing video streams in real-time. YOLOv5 excels in this aspect.

- YOLO is known for its efficient resource utilization due to its streamlined process flow, making it a suitable choice for deployment on devices with limited computational power.

Testing YOLOv5

To evaluate the performance of YOLOv5, tests were conducted using two distinct MP4 video samples. These videos were selected to represent varying levels of complexity and visual characteristics, providing a comprehensive assessment of the algorithm's robustness:

- **Video 1 - Shopping Center:** A simple, colorful video depicting individuals walking through a shopping center, primarily moving in two directions. This video serves as a straightforward scenario to test YOLOv5's ability to detect people under minimal visual complexity (Fig. 2.5).

- **Video 2 - Street Scene:** A black-and-white video capturing pedestrians on a street, featuring increased complexity due to factors like varying motion directions and environmental effects such as smoke. This video challenges YOLOv5's capability to maintain detection accuracy under more intricate conditions (Fig. 2.6).

Figure 2.5: Coloured MP4 Video Sample

Figure 2.6: Black and White MP4 Video Sample

Colored Sample

Figure 2.7: Coloured MP4 Video Sample with YOLOv5 (Taken using nano PyTorch file)

Results:

Model	Elapsed Time [s]	Average FPS	Minimum FPS	Maximum FPS
YOLOv5n	8	36	12	46
YOLOv5s	9	33	15	41
YOLOv5m	9	33	12	41
YOLOv5l	9	33	16	38
YOLOv5x	9	32	14	37

Table 2.1: Results of YOLOv5 running on the Colored Sample

Note: FPS stands for frames per second which is the one of the units chosen in this project to measure the rate of how fast the video samples are being processed by the algorithm. A higher value implies a faster processing.

Conclusions:

The results obtained using the *nano* PyTorch model demonstrate faster processing times compared to its heavier and more complex counterparts, which offer higher accuracy. It is worth noting that the original video has a duration of twelve seconds, and all execution times recorded were below this threshold. Consequently, there is no compelling reason to prioritize the lighter weight models, as the performance difference compared to the heaviest model (*YOLOv5x*) is not significant. The potential increase in accuracy with the heavier model justifies its use, especially in contexts where precision is critical.

Black and White Sample

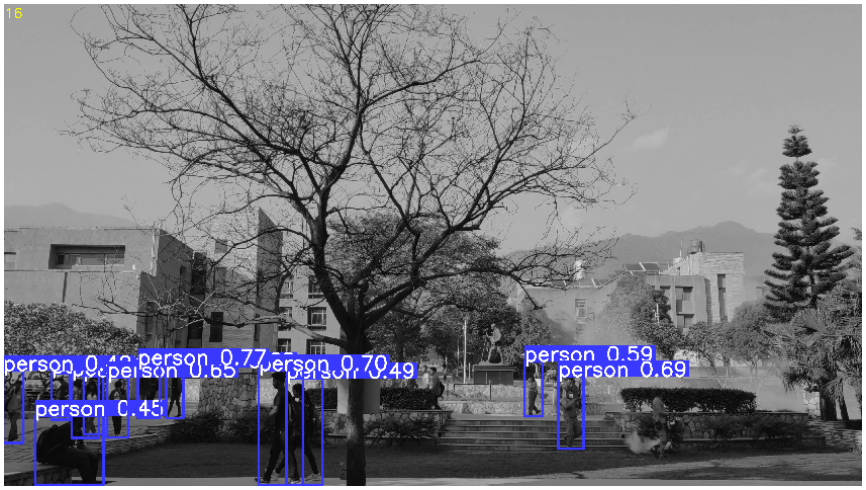

Figure 2.8: Black and White MP4 Video Sample with YOLOv5 (using the nano PyTorch model)

Results:

Model	Elapsed Time [s]	Average FPS	Minimum FPS	Maximum FPS
YOLOv5n	18	19	12	26
YOLOv5s	21	17	8	24
YOLOv5m	24	15	10	19
YOLOv5l	25	14	8	20
YOLOv5x	25	14	7	17

Table 2.2: Results of YOLOv5 running on the Black and White Sample

The results showed similar patterns to those observed in the coloured sample. However, due to the increased complexity of the video (e.g., environmental effects such as smoke and diverse pedestrian movements), the processing times were longer overall. This is reflected in higher elapsed times and lower FPS values. Additionally, the classification confidence scores were generally lower, indicating that the more intricate visual features posed a greater challenge for the algorithm.

Subsequently, the algorithm was adjusted to focus exclusively on detecting people. After re-testing this adjustment on both video samples, it was concluded that filtering classification classes to only people had no significant impact on execution times or classification confidence.

2.1.4 Comparison

Table 2.3 highlights the main differences between the discussed Object Detection Modules (R-CNN, Fast R-CNN, and YOLO), comparing their approaches, performance, and suitability for real-time applications.

Feature	**R-CNN**	**Fast R-CNN**	**YOLO**
Detection Approach	Classification-based	Classification-based	Regression-based
CNN Usage	Runs CNN on 2000 region proposals per image	Single forward pass of the entire image through CNN	Single forward pass of the entire image through CNN
Region Proposal Method	Selective Search	Selective Search	Grid-based (no region proposals)
Inference Time per Image	∼49 seconds	∼2 seconds	$<$ 0.02 seconds (real-time)
Real-time Capability	No	No	Yes
Classification Method	SVM (Support Vector Machine)	Softmax Layer	Single-stage Regression
Output Predictions	Bounding boxes only	Bounding boxes with classification	Concurrent bounding boxes and class predictions
Performance with Small Objects	Good at finding small objects	Good at finding small objects	Struggles with detecting small objects

Table 2.3: Comparison of Object Detection Models. Adapted from (Malhotra and Garg, 2020)

2.2 Tracking in Object Detection

Object tracking is a critical component in computer vision systems, particularly in applications that require monitoring and analysing the movement of individuals over time. In the context of this project, tracking algorithms complement object detection models by maintaining the identity of detected objects across successive frames, enabling accurate people counting and movement analysis.

This section presents a comparison of three prominent tracking algorithms: **ByteTrack** (Ma et al., 2023), **StrongSORT** (Du et al., 2023), and **DeepSORT** (Pujara and Bhamare, 2022). Each of these algorithms offers unique approaches to solving the challenges of object tracking, such as maintaining object identities, handling occlusions, and ensuring real-time performance.

- **ByteTrack**: A highly efficient tracking algorithm designed to associate both low- and high-confidence detections, improving tracking performance in challenging environments.
- **StrongSORT**: A recent enhancement of DeepSORT, leveraging stronger appearance models and improved association strategies.
- **DeepSORT**: An extension of the SORT algorithm, which incorporates appearance features for more robust tracking in complex scenarios.

The objective of this comparison is to evaluate these algorithms in terms of their performance, computational efficiency, and suitability for real-time visitor monitoring in historical monuments. By analysing their strengths and limitations, we aim to identify the most appropriate algorithm for integration into the system.

2.2.1 Possible challenges to Tracking Algorithms

- **Occlusion**: Objects may become partially or fully obscured by other objects in the scene, making it difficult to maintain their identities.
- **Appearance Variation**: Changes in an object's appearance, caused by shifts in orientation, movement, or variations in the camera's perspective, can complicate tracking.
- **Motion Blur**: Rapid object movement or camera motion can result in blurred images, reducing the accuracy of object detection and tracking.
- **Low Resolution**: Tracking performance may degrade when working with low-resolution video feeds, as finer details of objects are harder to discern.
- **Real-Time Processing**: Multi-object tracking algorithms must operate in real time, which becomes particularly challenging in complex or crowded environments.

(Pujara and Bhamare, 2022)

2.2.2 ByteTrack

ByteTrack is a multi-object tracking (**MOT**) algorithm designed to enhance tracking performance by leveraging both **high-** and **low-confidence** detections. Like other MOT frameworks, ByteTrack utilizes the **Kalman filter** for predicting object trajectories and the **Hungarian algorithm** for associating detections with existing tracks (Fig. 2.9). However, its unique strength lies in its ability to improve object association by incorporating low-confidence detections that are often discarded in traditional approaches.

The algorithm introduces a data association method called **BYTE**, which filters out false positives from low-confidence detections while retaining potential true targets. ByteTrack categorizes detections into three groups based on confidence levels: **high-confidence boxes**, **low-confidence boxes**, and boxes with **very low confidence**, which are discarded. The high-confidence detections are matched with existing trajectories in the first pass, while unmatched low-confidence detections are considered in a second pass to improve continuity and reduce missed targets.

ByteTrack further enhances robustness by retaining unmatched detections for a limited number of frames (e.g., 30). If the target reappears within this timeframe, the trajectory is reactivated; otherwise, it is discarded. This multi-stage matching process ensures greater continuity in object tracking, even in challenging scenarios where objects might briefly disappear or have inconsistent detections.

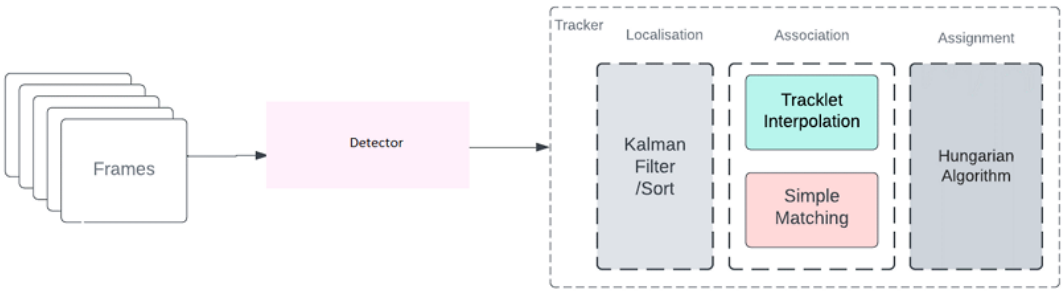

Figure 2.9: Illustration of the Fast R-CNN architecture. From (Pujara and Bhamare, 2022)

2.2.3 StrongSORT

StrongSORT is an enhanced version of the DeepSORT algorithm, designed to improve tracking accuracy and robustness in challenging scenarios. One of its key innovations is the introduction of the **Appearance-Free Link (AFLink)** model. Unlike traditional tracking approaches that rely heavily on appearance features, AFLink uses statistical methods to associate short tracklets into complete trajectories. This is achieved by formulating the association as a minimum cost flow problem, which is solved using the network simplex algorithm. AFLink leverages motion and spatial information to compute association costs, making it effective even when appearance features are unreliable or unavailable.

Another significant improvement in StrongSORT is the implementation of **Gaussian-Smoothed Interpolation (GSI)**. This technique predicts the position of objects in frames where they are not detected, using both their motion and the motion patterns of other objects in the scene. By applying a Gaussian filter to smooth object trajectories, GSI reduces the impact of noise and enhances the accuracy of position predictions for undetected frames.

StrongSORT demonstrates superior performance compared to DeepSORT, particularly in scenarios involving partial occlusions or significant appearance changes. Its ability to accurately track objects under such conditions is further strengthened by the use of a CNN-based model trained for person re-identification (ReID) tasks. These enhancements make StrongSORT highly effective in complex environments, where maintaining object identities is critical.

2.2.4 DeepSORT

DeepSORT is a multiple object tracking (**MOT**) algorithm designed to improve tracking accuracy and robustness in complex and crowded environments. It builds upon the foundation of the **SORT** (*Simple Online and Real-Time Tracking*) algorithm, which uses the **Kalman filter** for motion prediction and the **Hungarian algorithm** for associating detections across frames.

DeepSORT enhances this approach by integrating a deep learning-based **appearance descriptor**, typically derived from **convolutional neural networks (CNNs)**, to improve the precision of object association. This descriptor captures the visual features of detected objects, enabling the system to maintain accurate tracking even when objects undergo changes in appearance, such as shifts in pose, lighting variations, or partial occlusion.

The combination of **appearance** and **motion information** significantly boosts tracking performance. As a result, DeepSORT is well-suited for applications such as **video surveillance**, **autonomous driving**, and **human-computer interaction**, where accurate and reliable tracking is essential.

Testing DeepSORT

To evaluate the performance of DeepSORT integrated with YOLOv5, a series of tests were conducted. Initially, the MP4 video samples previously used to test YOLOv5 as a standalone algorithm were utilized (Fig. 2.10). Following this, tests were performed using live camera footage streamed via Sockets from the Raspberry Pi 5.

Figure 2.10: Colored MP4 Video Sample with YOLOv5 and DeepSORT (using the X PyTorch model)

2.2.5 Comparison and Conclusion

Here is a comparison of three popular multi-object tracking algorithms: StrongSORT, ByteTrack and DeepSORT highlighting their key features, strengths, and limitations.

StrongSORT:

- **Appearance Embedding**: Uses ReID models.
- **Association Method**: Nearest neighbor search combined with AFLink.
- **Additional Features**: Includes GSI (Gaussian Smoothed Interpolation), MC (Motion Compensation), EMA (Exponential Moving Average), and ECC (Edge Correlation Coefficient).
- **Strengths**: High accuracy, lightweight, better handling of complex scenarios.
- **Limitations**: Slower performance, issues with occlusion and missed detections.

ByteTrack (Pujara and Bhamare, 2022):

- **Appearance Embedding**: Uses a matching template.
- **Association Method**: BYTE algorithm.
- **Additional Features**: None.
- **Strengths**: High speed, simplicity, and flexibility; can work with other association techniques.
- **Limitations**: Vulnerable to occlusion, occasional mismatches in associations.

DeepSORT:

- **Appearance Embedding**: Uses CNN-based features.
- **Association Method**: Simple linear assignment.
- **Additional Features**: None.
- **Strengths**: High accuracy.
- **Limitations**: Struggles with occlusion.

Note: Testing Environment:

These and all other performance tests involving Machine Learning algorithms were conducted on a machine with the following specifications:

- **CPU:** AMD Ryzen 7 5800x3d
- **GPU:** NVIDIA GeForce RTX 4070 Super
- **RAM:** 16GB DDR4
- **OS:** Ubuntu 22.04 LTS

Conclusion

With the comparison of the tracking algorithms complete, it was concluded that **DeepSORT** is the most suitable choice for this project. This decision stems from its ability to provide a balanced approach that aligns well with the project's key requirements: high consistency in object tracking and real-time performance.

While both **StrongSORT** and **ByteTrack** offer improvements in specific areas, such as enhanced handling of missed detections and increased speed, respectively, they also come with limitations. StrongSORT's additional complexity and slower processing make it less suitable for real-time applications. ByteTrack, on the other hand, sacrifices some tracking robustness when dealing with occlusions and may result in occasional mismatches, which could compromise accuracy in crowded environments.

DeepSORT strikes the right balance by leveraging appearance embeddings and reliable motion modeling to achieve robust object association without significant computational overhead. It performs well in challenging scenarios, including partial occlusions, while maintaining a consistent level of accuracy. Additionally, its relatively lightweight implementation ensures that the system operates within the real-time constraints required for effective visitor monitoring.

Therefore, DeepSORT was selected as the tracking algorithm for this project, providing the best trade-off between performance, accuracy, and resource efficiency, making it ideal for real-time monitoring in a historical monument context.

Chapter 3

System Analysis and Design

This section provides a description of the main aspects related to the preparation of the project work, system analysis and an overview of the main used technologies.

3.1 Requirements

Given the historical significance of the monument where this project will be implemented, special consideration must be given to the discreetness of the hardware used. All equipment should blend unobtrusively with the environment to avoid detracting from the monument's historical context.

With this in mind, the primary goal is to develop a system that assists human personnel in managing access to a specified area by counting the number of individuals entering. This system should serve as a supportive tool for human operators, rather than fully automating the management process.

For security reasons, an essential requirement is a simple authentication mechanism, ensuring that only authorized users can access the system.

An intuitive user interface (UI) is also crucial. The interface should be straightforward, requiring no specialized training for users. Additionally, the UI should be accessible across multiple platforms, ensuring full functionality regardless of the device or operating system.

A basic CRUD (Create, Read, Update, Delete) capability is necessary to manage information about rooms and cameras, allowing users to adjust settings, add new cameras, and monitor specific spaces easily.

Since the operator may not be monitoring the system continuously, an alert system is needed to notify the user if certain thresholds are surpassed. This feature will help the operator respond promptly when necessary, even if they are not actively watching the system.

As this project incorporates object detection models using cameras, it would be beneficial for the video feed from the cameras to be accessible through the UI. This feature would enable the user to verify the algorithm's accuracy without needing to be physically present in the monitored area.

Finally, the ability to store data collected by the system for later analysis is highly desirable. This historical data should be available through the UI for statistical or auditing purposes, providing valuable insights into visitor trends and system performance.

3.2 System Analysis

3.2.1 System Architecture

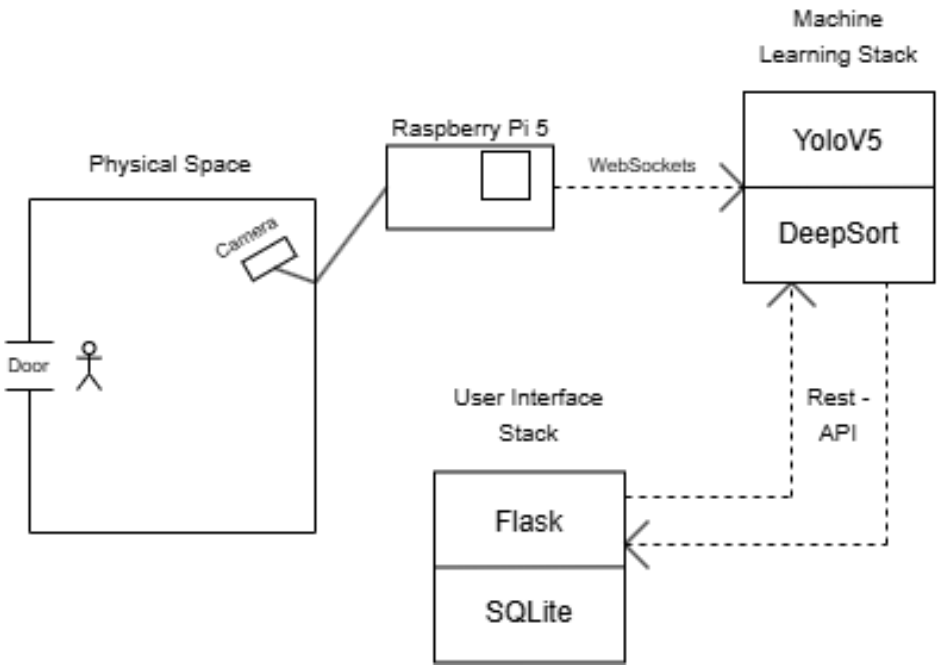

Figure 3.1: System Architecture Diagram of the Project

The system architecture (Fig.3.1) for the project is designed to ensure seamless interaction between the hardware, machine learning models, and the user interface. The main components and their roles are described as follows:

- **Physical Space**:
  - Represents the monitored area, such as a room or hallway, equipped with a camera positioned to capture the entrance and exit points. The camera continuously streams video data to the processing unit.
- **Raspberry Pi 5**:

  - Acts as the central processing hub, capturing the video feed from the camera and forwarding it to the machine learning stack. It is responsible for ensuring real-time data transfer via Sockets.

- **Machine Learning Stack (ML-Stack)**:
  - **YOLOv5**: A real-time object detection model used to identify people in the video feed and generate bounding boxes for detected objects.
  - **DeepSORT**: A tracking algorithm that associates detected objects across frames, enabling accurate counting of people entering or exiting the monitored space.
  - The ML-Stack processes the video feed and sends the results (e.g., people count, entry/exit events) to the user interface stack via a REST API. This stack is containerized using Docker to facilitate management and interaction.

- **User Interface Stack (UI-Stack)**:
  - **Flask**: A lightweight web framework that serves as the backend for the system's user interface. It handles incoming API requests, processes data, and manages user authentication.
  - **SQLite**: A relational database used to store historical data such as people counts, timestamps, and system logs. It supports the CRUD operations required for managing rooms and cameras.

Data Flow:

- Video data flows from the camera to the Raspberry Pi 5, which streams it to the ML-Stack via Sockets.
- The processed data, including people counts and entry/exit events, is sent to Flask via a REST API. Configuration information, such as the boundary polygon coordinates, is also sent from Flask to the Dockerized ML-Stack.
- Flask stores the data in SQLite and provides it to the user through an intuitive web-based interface.

Key Considerations:

- **Real-Time Processing**: The use of Sockets and efficient machine learning models ensures near real-time object detection and tracking.
- **Scalability**: The architecture supports adding multiple cameras and rooms as needed.
- **User-Friendly Interface**: The Flask-based UI ensures the system is accessible and easy to use, even for non-technical staff.
- **Data Persistence and Analytics**: SQLite provides long-term storage for historical data, allowing users to analyse visitor trends and system performance.

3.2.2 Class Diagram

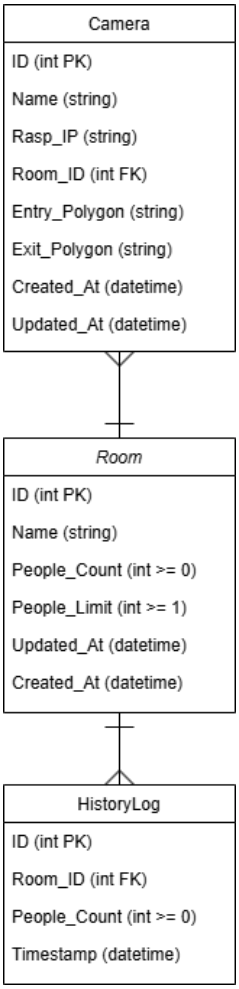

Figure 3.2: Class Diagram of the Project

Class Diagram Description

The class model (Fig.3.2) outlines the key entities and their relationships for the application designed to manage access and monitor the flow of people in a designated space. The main entities are **Camera**, **Room**, and **HistoryLog**, each serving a distinct purpose within the system:

- **Camera**:
  - **ID**: Unique identifier for each camera.
  - **Name**: Descriptive name for the camera.

- **Rasp_IP**: IP address of the Raspberry Pi responsible for transmitting the camera feed.
- **Room_ID**: Foreign key linking the camera to a specific room.
- **Entry_Polygon** and **Exit_Polygon**: Define the areas within the camera's field of view for detecting entries and exits.
- **Created_At** and **Updated_At**: Timestamps for tracking when the camera record was created and last updated.

- **Room**:
  - **ID**: Unique identifier for each room.
  - **Name**: Name of the room.
  - **People_Count**: Current number of people in the room (must be ≥ 0).
  - **People_Limit**: Maximum number of people allowed in the room (must be ≥ 0).
  - **Created_At** and **Updated_At**: Timestamps for tracking room record creation and updates.

- **HistoryLog**:
  - **ID**: Unique identifier for each log entry.
  - **Room_ID**: Foreign key linking the log entry to a specific room.
  - **People_Count**: The number of people recorded at the **Timestamp**.
  - **Timestamp**: The date and time of the log entry.

Relationships:

- **Camera** is linked to **Room** through **Room_ID** to associate each camera with a specific monitored space.
- **HistoryLog** is linked to **Room** through **Room_ID** to store historical data about people counts for each room.

This design supports real-time monitoring, historical data analysis, and system flexibility by allowing multiple cameras per room and maintaining a log of people counts over time.

3.3 Technological Background

3.3.1 Sockets

Their Role in System Communication

The communication between the Raspberry Pi and the Machine Learning (ML) Stack is a critical component of this project, as it ensures the real-time transmission of video data and the corresponding results. To facilitate this connection, **Sockets** (Xue and Zhu, 2009) were utilized due to their ability to establish a persistent, low-latency, and reliable communication channel.

Why Sockets?

Traditional communication protocols such as HTTP operate in a request-response model, which introduces latency and inefficiency for real-time applications. In contrast, Sockets allow for:

- **Flexible Communication**: Sockets allow custom protocols to be implemented, providing precise control over data exchange.
- **Low Latency**: Sockets can establish direct communication paths, reducing delays compared to higher-level protocols like HTTP.

Benefits of Using Sockets in This Project

The use of Sockets in this project provides several key advantages:

- **Real-Time Performance**: Ensures minimal delay between video capture, processing, and result visualization.
- **Efficient Resource Utilization**: Sockets enable efficient data exchange without the overhead of repeated connection setups, making them ideal for continuous video streaming and analysis.
- **Scalability**: The Socket-based communication model can be extended to support multiple Raspberry Pi devices and cameras, ensuring the system's scalability.

Overall, Sockets play a vital role in achieving the real-time performance and flexibility required for the system's functionality.

3.3.2 YOLOv5

Its Role in Object Detection

YOLOv5 (*You Only Look Once*) is a state-of-the-art object detection algorithm used in this project to identify people in the monitored area. Its ability to perform real-time detection

with high accuracy makes it a critical component of the system. YOLOv5 processes each video frame and generates bounding boxes and class labels for detected objects.

Why YOLOv5?

YOLOv5 was chosen for this project due to its unique combination of speed and accuracy, which is essential for real-time monitoring. Key reasons for selecting YOLOv5 include:

- **Real-Time Capability**: YOLOv5 can process video frames at a high frame rate, ensuring timely detection of objects.
- **Efficiency**: YOLOv5's architecture is optimized for low resource usage, making it suitable for deployment on resource-constrained devices.
- **Robust Performance**: It performs well in identifying people, which is the primary objective of the project, even in challenging scenarios.

Benefits of Using YOLOv5 in This Project

Integrating YOLOv5 into the system offers several advantages:

- **Accurate Detection**: Provides reliable identification of people, ensuring the system's core functionality is met.
- **Real-Time Detection**: Processes video frames quickly enough to support live monitoring.
- **Resource Efficiency**: Performs object detection with minimal computational overhead, ensuring compatibility with the system's hardware.

Overall, YOLOv5 is a foundational element of the system, enabling precise and efficient real-time object detection.

3.3.3 DeepSORT

Its Role in Object Tracking

DeepSORT (*Deep Simple Online and Real-Time Tracking*) is a multiple-object tracking algorithm used to maintain the identity of detected objects across successive frames. In this project, it extends the capabilities of YOLOv5 by tracking individuals as they move through the monitored space. This is essential for accurately counting entries and exits.

Why DeepSORT?

DeepSORT was selected for its robust tracking capabilities, which are critical for this project. The key reasons include:

- **Appearance-Based Association**: DeepSORT uses appearance embeddings derived from convolutional neural networks (CNNs) to associate detections across frames, enhancing tracking accuracy.
- **Motion Prediction**: Incorporates a Kalman filter to predict object trajectories, even in the presence of partial occlusions.
- **Scalability**: The algorithm is computationally efficient, enabling real-time tracking on hardware like the Raspberry Pi.

Benefits of Using DeepSORT in This Project

The use of DeepSORT adds significant value to the system:

- **Accurate People Tracking**: Maintains the identity of individuals across frames, allowing for precise counting and monitoring.
- **Robustness in Complex Scenarios**: Handles occlusions and changes in object appearance effectively.
- **Real-Time Capability**: Operates efficiently alongside YOLOv5, ensuring minimal latency in tracking tasks.

By combining DeepSORT with YOLOv5, the system achieves robust, real-time object detection and tracking, fulfilling its goal of accurate visitor monitoring.

3.3.4 Docker

Its Role in Containerizing the ML Stack

In this project, **Docker** (Susnjara and Smalley, 2024) plays a crucial role by containerizing the Machine Learning (ML) Stack, which includes the YOLOv5 and DeepSORT models. Docker provides an efficient and consistent environment for running the ML Stack, regardless of the underlying system architecture, ensuring reproducibility and modularity across deployments.

Why Docker?

Containerizing the ML Stack using Docker offers several key advantages that are essential to this project:

- **Environment Consistency**: Docker containers ensure that the ML Stack runs in the same environment every time, eliminating compatibility issues related to dependencies or software versions.

- **Portability**: Containers can be deployed on any machine that supports Docker, which allows the ML Stack to be easily transferred and deployed on different systems.

- **Scalability and Modularity**: Docker enables each component of the ML Stack to be containerized separately, allowing for modular scaling and easy updates to individual components without impacting the entire system.

- **Resource Efficiency**: Docker containers are lightweight, making them efficient in terms of resource usage compared to full virtual machines.

Benefits of Using Docker in This Project

The use of Docker for containerizing the ML Stack enhances the system's flexibility, scalability, and ease of maintenance:

- **Simplified Updates and Maintenance**: With Docker, updates to the ML Stack (e.g., model retraining or library updates) can be implemented simply by rebuilding the container, which helps streamline maintenance.

- **Scalability**: Docker enables horizontal scaling, allowing multiple instances of the ML Stack to be deployed across multiple devices if needed.

- **Isolation and Security**: Containerization isolates the ML Stack, reducing the risk of conflicts with other system processes and providing an additional layer of security.

Overall, Docker's role in this project is central to ensuring that the ML Stack is robust, portable, and easy to manage, making it an ideal choice for containerizing complex, resource-intensive applications.

3.3.5 Flask

Its Role as the User Interface

In this project, **Flask** (Ashray, 2024) serves as the backend framework for the user interface (UI), providing a seamless connection between the user and the system. Flask is a lightweight and flexible web framework for Python, designed to build web applications quickly and efficiently. Its simplicity and extensibility make it an ideal choice for this project, where the primary focus is to offer an intuitive and accessible interface for interacting with the system.

Why Flask?

The decision to use Flask was driven by several factors:

- **Lightweight and Minimalistic**: Flask is designed to be lightweight, allowing developers to build applications without unnecessary overhead.
- **Ease of Integration**: Flask integrates seamlessly with other components of the system, including the ML Stack and the SQLite database.
- **Flexibility and Extensibility**: Flask provides a simple structure that can be easily extended with plug-ins or custom middleware, adapting to the specific needs of the project.
- **Cross-Platform Accessibility**: Flask enables the UI to be accessed from any device with a web browser, ensuring platform independence.

Benefits of Flask in This Project

The use of Flask as the UI framework provides several advantages:

- **User-Friendly Interface**: The Flask-based UI simplifies the interaction between the user and the system, requiring no specialized training to operate.
- **Rapid Development**: Flask's simplicity and built-in tools facilitate rapid development and iteration, enabling quick adjustments based on user feedback.
- **Scalability and Maintainability**: Flask's modular design allows for easy scaling and updating of individual components without disrupting the overall system.

Overall, Flask plays a vital role in making the system accessible and easy to use, ensuring that end-users can efficiently monitor and manage visitor flow in real time.

3.3.6 SQLite

Its Role in Data Management

In this project, **SQLite** (Gaffney et al., 2022) serves as the database system for managing and storing data. SQLite is a lightweight, self-contained, and serverless relational database management system. Its simplicity and efficiency make it an ideal choice for projects that require minimal configuration and low resource usage.

Why SQLite?

SQLite was chosen for this project due to several advantages:

- **Lightweight and Self-Contained**: SQLite operates without the need for a separate server process, making it easy to integrate into the system with minimal overhead.

- **Ease of Use**: The database is easy to set up and manage, with no need for complex configurations or maintenance.
- **Efficiency**: Despite its lightweight nature, SQLite provides robust performance for managing small to medium datasets, which aligns well with the project's needs.
- **Portability**: SQLite databases are stored in a single file, making them easy to share, back up, and transfer between systems.
- **Simplicity of the Data**: The data that is to be saved in the SQLite Database has a very simple structure, which makes the usage of a more complex and robust database system unnecessary.

Benefits of Using SQLite in This Project

SQLite offers several benefits that are crucial to the success of this project:

- **Seamless Integration with Flask**: The tight integration between Flask and SQLite simplifies the process of managing database interactions within the web application.
- **Data Persistence and Accessibility**: Ensures that all collected data is persistently stored and easily accessible for real-time monitoring and historical analysis.
- **Low Resource Usage**: Requires minimal system resources, leaving the host system's resources free to be used by other tasks.

Overall, SQLite provides a reliable and efficient solution for managing the data generated by the system, supporting both real-time operations and long-term data analysis.

Chapter 4

Implementation

In this chapter, a detailed exploration of the practical steps taken to implement the system is provided. The development of a system capable of real-time visitor monitoring and management relies on the integration of several advanced technologies, each playing a crucial role in achieving the desired functionality.

We begin by discussing the practical implementation details. This includes the setup and configuration of the hardware components, the deployment and optimization of machine learning models, the design and development of the user interface, and the integration of these components into a cohesive system. We also discuss the challenges encountered during implementation and the solutions adopted to overcome them.

The goal of this chapter is to provide a comprehensive overview of how the theoretical concepts were translated into a working system, highlighting the technical and practical aspects of the project. This will serve as a foundation for evaluating the system's performance and effectiveness in later sections.

4.1 Live Feed Component: Raspberry Pi

The Raspberry Pi 5 serves as the core device responsible for capturing video feeds from the connected camera and transmitting them to the Machine Learning (ML) stack via Sockets. The hardware setup involved:

- Connecting a compatible USB or CSI (Camera Serial Interface) camera to the Raspberry Pi 5.
- Ensuring stable power supply and network connectivity (via Ethernet or Wi-Fi).
- Using a microSD card with sufficient storage for the operating system and software dependencies.

4.1.1 Software Installation

To enable video streaming and communication via Sockets, the following software and libraries were installed on the Raspberry Pi 5:

- **Operating System**: Installed the official *Raspberry Pi OS* for compatibility and stability.
- **Python Environment**: Set up Python 3 with virtual environments to manage dependencies.
- **OpenCV**: Used for video capture and processing.
- **Socket Library**: Installed the `socket` Python library for establishing Socket connections.

4.1.2 Configuration

The Raspberry Pi 5 was configured to stream video data to the ML stack:

- Implemented a Python script (link to the code is available in Appendix A) to continuously capture video frames from the camera and encode them for transmission.
- Established a Socket server-client communication model:
  - The Raspberry Pi acts as the client, sending video frames to the ML stack.
  - The ML stack acts as the server, receiving and processing the frames.

4.1.3 Implementation Challenges

During the setup and configuration process, several challenges were encountered:

- **Resource Constraints**: The Raspberry Pi 5 has limited computational resources, requiring optimization of video encoding to minimize latency without sacrificing quality.
- **Camera Compatibility**: Ensuring the camera's drivers and settings were compatible with the Raspberry Pi OS and `OpenCV`.

4.1.4 Outcome

The configured Raspberry Pi 5 successfully streams live video feeds from the connected camera to the ML stack in real-time via Sockets. This setup ensures reliable, low-latency communication, enabling the ML stack to process video data efficiently and provide real-time tracking results.

4.2 Machine Learning Component: ML-Stack

4.2.1 Overview of the ML-Stack

The ML-Stack is responsible for real-time object detection and tracking. It integrates **YOLOv5** for object detection and **DeepSORT** for tracking, containerized within a **Docker** environment to ensure modularity and portability. The stack processes video streams received via Sockets and provides real-time tracking results.

4.2.2 Software Installation and Setup

The ML-Stack setup involved installing and configuring the following components:

- **Docker and Docker Compose**:
  - Installed Docker for containerization and Docker Compose to manage multi-container environments.
  - Created a Dockerfile to define the YOLOv5 and DeepSORT dependencies, including Python libraries such as `torch`, `opencv-python`, and `numpy`.
- **YOLOv5 and DeepSORT**:
  - Created a script (link to the code is available in Appendix A) that allows to apply both YOLOv5 and DeepSORT to the frames collected via the Raspberry Pi.
  - Customized configuration variables to specify detection thresholds and tracking parameters.

- **Socket Integration**:
  - Installed the `sockets` Python library to establish server-side Socket communication with the Raspberry Pi.
  - Configured the ML-Stack as a Socket server to receive video frames and send processed results.

4.2.3 Docker Containerization

To ensure consistency and portability, the ML-Stack was containerized using Docker:

- **Dockerfile**:
  - Defined the base image as `nvidia/cuda:11.8.0-cudnn8-runtime-ubuntu22.04` to ensure NVIDIA driver compatibility.
  - Installed required dependencies, including YOLOv5 and DeepSORT libraries.
  - Added the Socket server and YOLOv5/DeepSORT integration script.

4.2.4 Configuration

The ML-Stack was configured to process video streams and deliver tracking results:

- Configured YOLOv5 for object detection:
  - Specified the target class (people) to optimize detection performance.
  - Adjusted confidence thresholds to reduce false positives.
- Configured DeepSORT for tracking:
  - Set up appearance-based tracking to maintain object identities across frames.
  - Tuned parameters for motion prediction using the Kalman filter.
- Implemented Socket message handlers:
  - Handlers for receiving video frames from the Raspberry Pi.
  - Handlers for sending tracking results back to the user interface.

4.2.5 Implementation Challenges

During the setup, several challenges were encountered:

- **Dependency Conflicts**: Resolving library version conflicts between YOLOv5, DeepSORT, and Docker.
- **Socket Reliability**: Ensuring stable communication between the Raspberry Pi and the ML-Stack.

4.2.6 Outcome

The ML-Stack was successfully configured to process video streams in real-time, leveraging the combined strengths of YOLOv5 and DeepSORT for accurate object detection and tracking. The use of Docker ensured consistent deployment and streamlined the integration with other system components.

4.3 Software Engineering Component: UI-Stack

The UI-Stack is a Flask-based web application that provides an interface for users to interact with the system. It integrates **SQLite** as a lightweight database engine for storing historical data, configurations, and user information. The UI stack consists of two main components: the **frontend**, which manages the user interface, and the **backend**, which handles server-side logic and database operations.

4.3.1 Frontend: HTML, CSS/Bootstrap, and JavaScript

The frontend is responsible for delivering a user-friendly interface for managing the system. The following technologies and frameworks were used:

- **HTML**: Provides the structure for the web pages.
- **CSS/Bootstrap**: CSS is used for styling, while Bootstrap ensures responsive design and modern UI components.
- **JavaScript**: Enables dynamic interactivity, such as real-time updates and asynchronous data fetching using AJAX.

Frontend Setup

- Designed HTML templates with Flask's `Jinja2` templating engine to dynamically render content based on data from the backend (Fig.4.1).
- Integrated Bootstrap for consistent styling and a responsive layout, ensuring compatibility across devices.
- Used JavaScript and AJAX for:
  - Fetching live camera feeds and processed data from the backend.
  - Updating UI elements, such as real-time people counts and historical data visualizations.

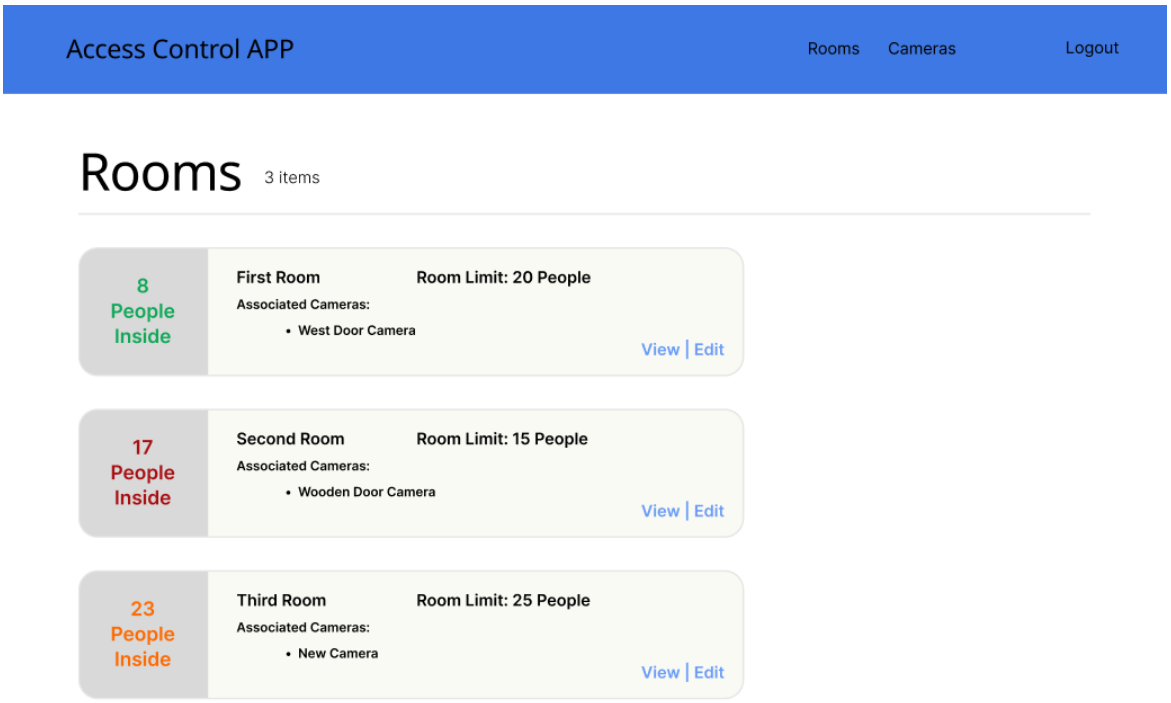

Figure 4.1: Mockup of the main page of the Project

Frontend Challenges

- **Responsiveness**: Ensuring that the UI functions seamlessly across different screen sizes and devices.
- **Dynamic Updates**: Managing frequent real-time updates without overloading the client-side browser.

4.3.2 Backend: Flask and SQLite

The backend is the core of the UI stack, managing server-side logic and interactions with the SQLite database.

Flask Framework

- **Routing**: Configured Flask routes to handle requests and serve pages dynamically.
- **CRUD**: Implemented logic to enable the use of CRUD actions on the application's models (Fig.4.2).

- **REST API Endpoints**: Implemented API endpoints to:
  - Fetch real-time data from the ML-Stack.
  - Process and store historical data in the SQLite database.
  - Enable user authentication and configuration management.

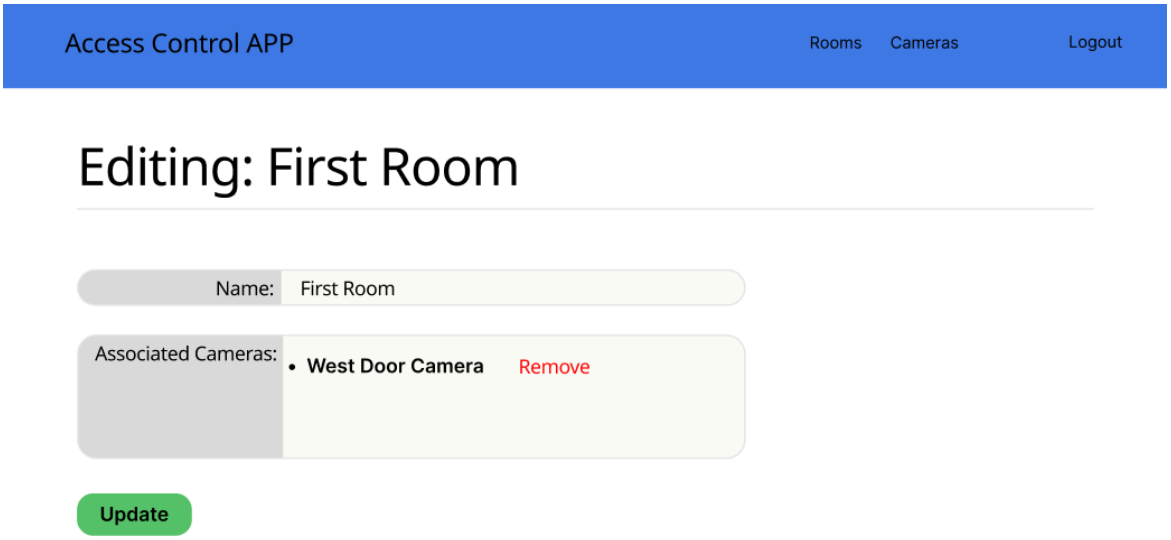

Figure 4.2: Mockup of the main page of the Project

SQLite Database

- Configured SQLite as the database engine to store:
  - **Room and Camera Information**: Tracks room configurations and associated cameras.
  - **Historical Data**: Logs people counts and timestamps for trend analysis.
  - **User Authentication Data**: Stores credentials for secure access to the system.
- Designed the database schema to ensure scalability and efficient querying.

Backend Challenges

- **Database Optimization**: Ensuring that SQLite queries are efficient for both real-time updates and historical data retrieval.
- **Error Handling**: Managing edge cases, such as interrupted Socket connections or malformed client requests.
- **Security**: Implementing user authentication and preventing unauthorized access to sensitive data.

4.3.3 Outcome

The Flask-based UI stack successfully integrates the frontend and backend components to provide an intuitive, real-time interface for monitoring and managing the system. The use of SQLite ensures lightweight yet reliable data storage, while the combination of Flask, Bootstrap, and JavaScript delivers a modern, responsive user experience.

Chapter 5

Case Study: Real-Time Access Monitoring in a Residential Setting

5.1 Setup and Configuration

5.1.1 Overview of the Setup

The case study was conducted in a residential setting to evaluate the performance of the system in a real-world scenario. The objective was to test the system's ability to accurately monitor access to a designated space using live video footage.

5.1.2 Physical Setup

The physical setup was as follows:

- **Camera Placement**: The camera, mounted alongside the Raspberry Pi, was positioned at the top of a staircase, approximately 2 meters above the ground level.
- **Field of View (FOV)**: The camera's FOV covered a hallway with doors visible on the right and left sides at the edges of the frame. The evaluation point—a door—was located 3.5 meters away from the camera but not directly visible in the footage due to the angle.
- **Camera Angle**: The camera was angled downward to capture movement across the ground level, ensuring a clear view of the area near the evaluation point.

5.1.3 System Configuration

The system was configured as follows:

- **Raspberry Pi 5 and Camera**: The Raspberry Pi captured video footage at a resolution of 720p and transmitted it to the ML-Stack via Sockets for processing.

- **Machine Learning Processing**: The YOLOv5 and DeepSORT integration was deployed using a Python script (early stages of development) to process the live video feed and manage the count in real time.

5.2 Live Footage Analysis

The live footage demonstrates the YOLOv5 and DeepSORT integration, along with a basic access counting system. This implementation highlights the primary functionality of the project.

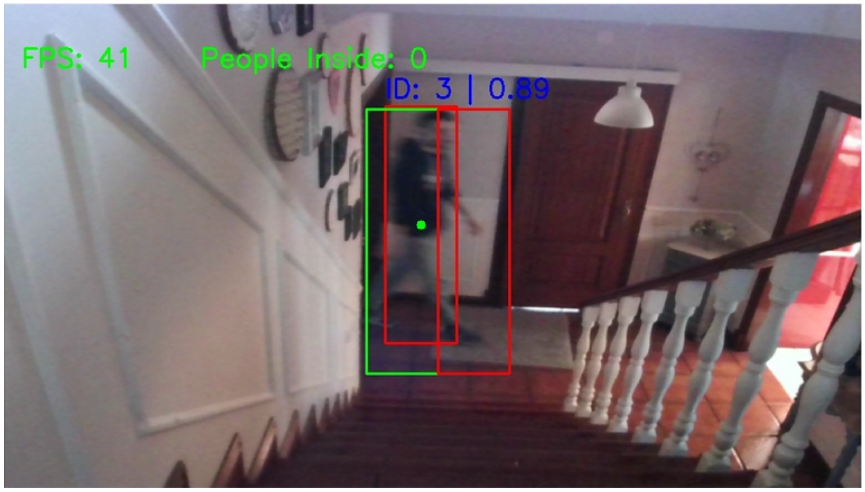

Figure 5.1: Live Video with YOLOv5 and DeepSORT, showing a count of 0 (using the X PyTorch model).

Figure 5.2: Live Video with YOLOv5 and DeepSORT, showing a count of 1 (using the X PyTorch model).

5.2.1 Counting Mechanism

When comparing Figures 5.1 and 5.2, it is evident that:

- Moving from **left to right** increases the count as the center of the bounding box transitions from the green polybox to the red polybox.
- Conversely, moving from **right to left** decreases the count as the bounding box crosses the polyboxes in reverse order.

This behavior demonstrates the system's ability to accurately track entries and exits, even when the evaluation point (the door) is not directly visible. The implemented counting mechanism ensures real-time monitoring of movements, fulfilling the system's intended purpose.

5.2.2 Positive Aspects

Despite the challenges encountered, the system demonstrated several strengths during this test, showcasing its effectiveness in real-world scenarios:

- **Robust Object Identification:** The algorithm successfully identified individuals regardless of their clothing, including cases where hats or shirts were of a similar color to the background. This indicates the robustness of YOLOv5 and DeepSORT in handling variations in appearance.
- **Accurate Handling of Multiple Individuals:** The system managed to track and count individuals accurately even when two people crossed the evaluation area simultaneously, including in opposite directions. While this test only involved two individuals due to the small area of the setup, it demonstrates the system's ability to maintain tracking integrity in moderately complex scenarios.
- **Minimal Processing Delay:** The delay between a person crossing the evaluation area and the processed results appearing on screen was consistently under one second. Considering the various processes involved - capturing the live feed, transmitting it via Sockets, and performing object detection and tracking — this delay is both acceptable and indicative of a well-optimized system. Notably, there was no observable lag during the test.

These positive aspects highlight the system's potential for effective real-time monitoring and provide a strong foundation for further refinement and scalability.

5.2.3 Aspects to Improve

While the system demonstrated its ability to monitor entries and exits effectively, several challenges were encountered during this test that highlight areas for potential improvement:

- **Missed Detections for Fast Movements:** When individuals crossed from the left to the right (entering the room) at a fast pace or while running, the count occasionally failed to increase. This occurred because the ML algorithm did not identify the person in time before they exited the evaluation area.
  - **Proposed Solution:** Adjust the camera angle so that it is positioned diagonally toward the door. This would give the algorithm more time to detect individuals as they approach the evaluation area.
- **Limited Field of View (FOV):** In this test setup, the camera's proximity to the evaluation area restricted the FOV, reducing the time available for the algorithm to detect and track individuals.
  - **Proposed Solution:** Increase the distance between the camera and the evaluation area to widen the FOV. A larger FOV allows the system to observe individuals for a longer period, improving detection reliability.

Addressing these issues in future setups will enhance the system's ability to monitor fast-moving individuals and improve overall accuracy and reliability in real-world scenarios.

Chapter 6

Conclusion

This project aimed to develop a real-time visitor monitoring system to address the challenges of preserving historical monuments while managing increasing tourist numbers. Through the integration of state-of-the-art computer vision algorithms, including YOLOv5 for object detection and DeepSORT for tracking, the system successfully achieved its objectives. The use of Sockets ensured efficient communication, while Flask provided a user-friendly interface, and Docker facilitated modular deployment.

The developed system demonstrated its capability to accurately count and track visitors in real-time, contributing to better visitor management and the long-term preservation of cultural heritage sites like the Convento de Cristo. Moreover, the project showcases the potential of combining advanced technology with heritage conservation, highlighting its applicability in other domains.

Despite its success, the project faced challenges, such as handling occlusions and optimizing performance. These experiences provided valuable lessons in balancing accuracy and efficiency within real-world constraints.

While the system meets current needs, future work could focus on integrating more advanced tracking algorithms, enhancing the user interface, and scaling the system to manage multiple sites. Additional features, such as predictive analytics, could further enhance its value.

In conclusion, this project has made significant contributions to both the field of computer vision and the management of cultural heritage sites, demonstrating the transformative potential of technology in preserving history.

Bibliography

Ashray (23 July 2024). Understanding flask framework: Installation, features & expert insights. https://www.analyticsvidhya.com/blog/2021/10/flask-python/.

Du, Y., Zhao, Z., Song, Y., Zhao, Y., Su, F., Gong, T., and Meng, H. (2023). Strongsort: Make deepsort great again. *IEEE Transactions on Multimedia*, 25:8725–8737.

Gaffney, K. P., Prammer, M., Brasfield, L., Hipp, D. R., Kennedy, D., and Patel, J. M. (2022). Sqlite: past, present, and future. *Proc. VLDB Endow.*, 15(12):3535–3547.

Gandhi, R. (2018). R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

Girshick, R. (2015). Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448.

Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524.

Jaiswal, A., Babu, A. R., Zadeh, M. Z., Banerjee, D., and Makedon, F. (2021). A survey on contrastive self-supervised learning. *Technologies*, 9(1).

Ma, L. V., Hussain, M. I., Park, J., Kim, J., and Jeon, M. (2023). Adaptive confidence threshold for bytetrack in multi-object tracking. In *2023 12th International Conference on Control, Automation and Information Sciences (ICCAIS)*, pages 370–374.

Malhotra, P. and Garg, E. (2020). Object detection techniques: A comparison. In *2020 7th International Conference on Smart Structures and Systems (ICSSS)*, pages 1–4.

Pujara, A. and Bhamare, M. (2022). Deepsort: Real time & multi-object detection and tracking with yolo and tensorflow. In *2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*, pages 456–460.

Susnjara, S. and Smalley, I. (6 June 2024). What is docker? https://www.ibm.com/topics/docker.

Terven, J., Córdova-Esparza, D.-M., and Romero-González, J.-A. (2023). A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716.

Xue, M. and Zhu, C. (2009). The socket programming and software design for communication based on client/server. In *2009 Pacific-Asia Conference on Circuits, Communications and Systems*, pages 775–777.

Appendix A

Code Repositories

- **WebCamStream-CLIENT**: https://github.com/MEI-ML-RVM/WebCamStream-CLIENT
- **MLStack-DOCKER**: https://github.com/MEI-ML-RVM/MLStack-DOCKER
- **UIStack-FLASK**: https://github.com/MEI-ML-RVM/UIStack-FLASK