Detect anomalies in a crowd using Deep Learning and Computer Vision

Development of an end-to-end human crowd detection, tracking and anomaly detection tool

Master's thesis in Data Science & AI

Axel Gautrand

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's Thesis 2023

Detect anomalies in a crowd using Deep Learning and Computer Vision

Development of an end-to-end human crowd detection, tracking and anomaly detection tool

Axel Gautrand

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

Detect anomalies in a crowd using Deep Learning and Computer Vision
Development of an end-to-end human crowd detection, tracking and anomaly detection tool

Axel Gautrand

Supervisor: Phi HUNG LE, La Javaness

Examiner: Adam ANDERSSON, Department of Mathematical Sciences

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: https://opencv.org/blog/multiple-object-tracking-in-realtime/

Typeset in Word

Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Detect anomalies in a crowd using Deep Learning and Computer Vision
Development of an end-to-end human crowd detection, tracking and anomaly detection tool

Axel Gautrand
Department of Computer Science & Engineering
Chalmers University of Technology

Abstract

This project stems from the Javaness R&D laboratory's initiative to enhance its expertise in deep learning algorithms for commercial innovation. Anomaly detection in crowds is a critical task for enhancing security, safety, and operational efficiency in public spaces, particularly during large-scale events like the Olympic Games 2024. Given the limitations of human ability to monitor live video surveillance effectively over extended periods, the development of artificial intelligence aims to improve attention and accuracy, reduce false alarms, and enable faster threat detection and intervention.

This thesis presents the development and implementation of an end-to-end tool designed to detect and track objects and identify anomalies in human crowds using simulated video surveillance data. Leveraging advancements in deep learning and computer vision, the project encompasses multi-object detection, multi-object online tracking, and anomaly recognition.

Key contributions include the training and evaluation of various state-of-the-art detection models on key performance metrics, benchmarking tracking algorithms for real-time application, and implementing effective baseline algorithms for anomaly detection across diverse scenarii. The system's performance was validated using comprehensive datasets, demonstrating its potential for real-time application. Additionally, a proof-of-concept web application was developed to showcase the practical implementation of the tool.

While the project achieved promising results, it also identified areas for future improvement, including dataset diversity, inference times, pipeline optimization, and increasing the complexity of anomaly detection algorithms. These areas highlight the gap between the current prototype and a fully commercialized solution. Furthermore, the tool has been developed to be as versatile as possible, requiring enhancements and adaptations for real-world commercial application.

This work lays a strong foundation for future research and practical implementations in crowd anomaly detection, emphasizing the importance of ethical considerations to respect individual freedoms and fundamental rights in order to avoid falling into authoritarian excesses such as mass surveillance.

Keywords: deep learning, computer vision, object detection, multiple object tracking, anomaly recognition.

Acknowledgements

I express my heartfelt gratitude to all my colleagues at La Javaness, whose support was crucial to the success of this project. Special thanks to Alexandre Do and Phi Hung Le for their wise and kind guidance. I am also grateful to Jules Sintes for his valuable advice and thorough proofreading, and to Adam Andersson for graciously volunteering as an examiner for this thesis.

Axel Gautrand, Paris

List of Acronyms

1. AI – Artificial Intelligence
2. DL – Deep Learning
3. CV – Computer Vision
4. CNN - Convolutional neural network
5. ViT – Vision Transformers
6. CCTV – Video surveillance / Closed-circuit television
7. R&D - Research & Development
8. OCR – Optical character recognition
9. TDD – Test driven development
10. MOD – Multiple object detection
11. MOT – Multiple object tracking
12. AD – Anomaly Detection
13. GPDR - General Data Protection Regulation
14. EDPS - European Data Protection Supervisor
15. EU - European Union
16. SOTA – State of the art
17. HD – High definition
18. UHD – Ultra High Definition
19. FPS – Frame per second
20. RGB – Red-Green-Blue
21. ANN – Artificial Neural Network
22. DNN – Deep Neural Network
23. FNN - Feedforward Neural Networks
24. CNN – Convolutional Neural Networks
25. RNN – Recurrent Neural Networks
26. LSTM – Long Short-Term Memory
27. GRU – Gated Recurrent Unit
28. R-CNN – Region-based Convolutional Neural Networks
29. FPN – Feature Pyramid Networks
30. RPN – Region Proposal Network
31. RoI – Region of Interest
32. ResNet – Residual Network
33. NMS – Non Maximum Suppression

List of Figures

List of Tables

Contents

1 Introduction

According to Russell and Norvig, artificial intelligence (AI) is defined as a discipline that aims to replicate aspects of human intelligence, such as learning, reasoning, perception, and critical thinking, through computer programs guided by logic [1]. This nascent field, spanning six decades, encompasses various sciences, theories, and techniques with the goal of emulating human cognitive capabilities. While some perceive AI as a potential threat to humanity, others view its development as an unprecedented opportunity to alleviate our workloads and enhance our productions. Consequently, Artificial Intelligence (AI) stands today as one of the most active research sectors, with contemporary systems already capable of performing complex tasks formerly reserved for human intervention.

Since the pioneering work of Paul Viola and Michael Jones in 2001, introducing a framework for face detection, the evolution of Deep Learning (DL) models has facilitated the processing of increasingly complex data, realizing numerous tasks within Computer Vision (CV). These tasks include object detection, target tracking in video sequences, and anomaly detection. The diverse applications in Computer Vision have prompted researchers and enterprises to develop various model architectures based on heterogeneous approaches with varying complexities. For instance, three state-of-the-art models for object detection, namely Convolutional Neural Networks (CNN) YOLOv8 [2], InterImage [3], and the Vision Transformer (ViT) Co-DETR [4], each exhibit a different number of parameters, namely 25M (on average across the presented model zoo), 2180M, and 348M, respectively. However, there is typically no intrinsically superior architecture for CV tasks, and the appropriateness of the system is highly correlated with the nature and characteristics of the addressed problem. Thus, exploring different approaches to a CV problem remains often insightful.

A pivotal use case common to numerous CV tasks is video surveillance. Inherent limitations in human capacity to attentively monitor live video sequences have led to a demand for artificial intelligence capable of more effectively performing this task. Individuals observing a single video screen for over twenty minutes lose 95% of their ability to maintain sufficient attention to discern important events. The addition of extra screens significantly diminishes this capacity.

This thesis presents the conceptualization and implementation of an end-to-end tool for the detection, tracking of human crowds, and anomaly detection in a simulated video surveillance (CCTV) context. The primary goal of this project is to develop deep learning algorithms and models focusing on object detection and tracking tasks, from training to evaluation, within the context of video stream processing.

1.1 Context and scope of the project

The Research and Development (R&D) team at Javaness[11] has previously undertaken various CV missions involving object detection, optical character recognition (OCR), and generative image creation through AI. However, the team has not yet had the opportunity to develop algorithms specifically tailored to video stream processing.
This project arises from the R&D laboratory's ambition to enhance its skills and knowledge in the realm of deep learning-based algorithms, particularly focusing on multiple object detection and video processing. The primary objective of this endeavor is to develop new, innovative tools for commercial use, with the aim of attracting new potential clients.
The project seeks to establish a solid and objective foundation for AI related to video stream processing. This will be achieved through in-depth considerations and work on various themes, leading to the design of an end-to-end tool dedicated to the detection, tracking of human crowds, and anomaly detection in a CCTV context. For each task, our models will initially prioritize versatility and generalization, subsequently refining their performance.

1.2 Planning and Progress

This project was conducted within the R&D sub-team responsible for unstructured data, comprising 6 members (20 across all R&D teams within the company). A work methodology derived from the agile SCRUM framework [8] was employed, utilizing sprint tickets each quantified in time units, weekly stand-up meetings, sprint reviews, and weekly meetings with the project supervisor. Additionally, a weekly code review session was organized, where each developer could share encountered issues with their colleagues, fostering team-wide skill enhancement.

Regarding the implementation methodology, we endeavored to apply Test Driven Development (TDD) [9].

This project was constrained to a 20-week timeline, focusing on four sub-parts:

- **Multiple Object Detection (MOD):** Creation of a dataset, exploration and evaluation of various state-of-the-art (SOTA) methods, retraining, and improvement of existing models.
- **Multiple Object Tracking (MOT):** Dataset creation, integration of algorithms with our MOD module, evaluation of different SOTA algorithms, and consideration of improvement avenues.
- **Anomaly Detection (AD):** Implementation of several algorithms addressing common anomaly detection use cases, including people counting, presence in restricted areas, and fall detection.
- **Creation of a web application:** Complete anomaly detection pipeline integrating MOD models with MOT and AD algorithms, implementation of a streamlit application for demonstration.

1.3 Ethical considerations

Any progress comes with its set of ethical questions, and AI is no exception to this rule. The work undertaken here raises ethical issues that are legitimate and important to address, ensuring a responsible and sustainable implementation, respectful of individual freedoms and fundamental rights, including freedom of expression.

[1] *La Javaness accelerates the transformation of large organisations through data and AI in an efficient and responsible way.*

These issues will be addressed in this section through five specific axes: Privacy and Confidentiality, Bias and Discrimination, Security, Transparency and Interpretability, Ecological Impact.

1.3.1 Privacy and Confidentiality

”Not ensuring a full ban on facial recognition is therefore a hugely missed opportunity to stop and prevent colossal damage to human rights, civic space and rule of law that are already under threat throughout the EU”

- Mher Hakobyan, Advocacy Advisor on Artificial Intelligence for Amnesty International

George Orwell's dystopian novel "1984", published in 1949, popularized the concept of "video monitoring" and introduced the figure of "Big Brother," now used to characterize institutions or practices that violate fundamental freedoms and the privacy of populations or individuals. While "video monitoring" was merely speculative science fiction at the time, it is now made possible through AI and CV, perfectly replacing and surpassing human capabilities in surveillance. These advancements have led to an explosion in the video surveillance market, with an exponential increase in the number of cameras installed in private and public spaces driven by authorities to combat insecurity.

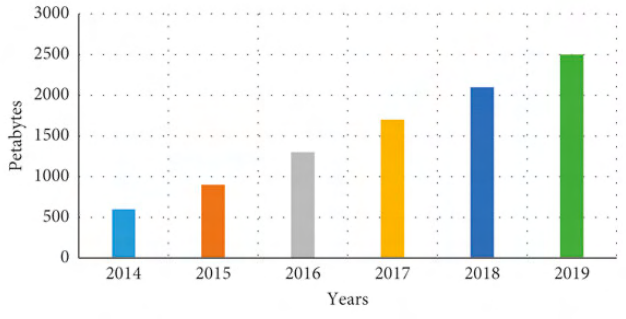

Figure 1.3.1 - Data generated by surveillance cameras [27]

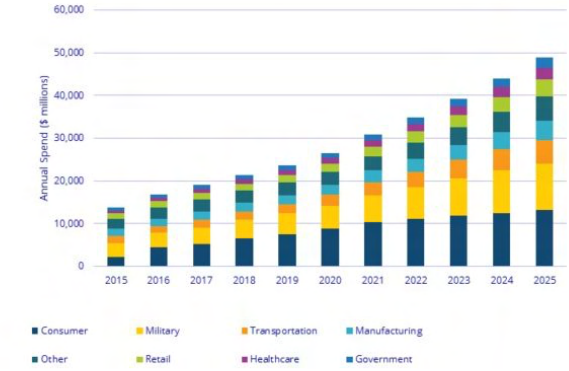

Figure 1.3.2 - Worldwide Video Surveillance Camera Market (1)

Technical advancements in facial recognition, as well as other CV breakthroughs, raise contemporary concerns regarding the preservation of human rights concerning privacy. Surveillance cameras often capture images of individuals allowing for their direct or indirect identification. This data falls under the category of "personal data" and is protected by the General Data Protection Regulation (GDPR), the European Union's (EU) reference regulation on personal data protection. The following principles (non-exhaustive) must be effectively applied:

- Data minimisation : ”Cameras can and should be used intelligently and should only target specifically identified security problems thus minimising the gathering of irrelevant footage.”
- Right of information : ” Notices can be found in EU institution buildings informing staff and visitors about the security cameras in place. These signs are mandatory because individuals affected by video-surveillance must be informed upon its installation about the monitoring, its purpose and the length of time for which the footage is to be kept and by whom.”
- Retention period : ” The European Data Protection Supervisor (EDPS) requires all European Union institutions to have clear policies regarding the use of video surveillance on their premises including on potential storage.”

The developed AI algorithms and models must consider these principles, just like the implemented hardware in public spaces, but they also pose potential new dangers that require further legislation. On December 9, 2023, negotiators from the European Parliament and the Council reached a provisional agreement on artificial intelligence legislation. This regulation aims to ensure that fundamental rights,

democracy, the rule of law, and environmental sustainability are protected against AI-related risks by ensuring that systems used in the EU are safe, transparent, traceable, non-discriminatory, environmentally friendly, and supervised by individuals rather than automation to avoid adverse outcomes. Concretely, the main prohibitions defined in this law that must be adhered to are:

- Biometric categorization systems using sensitive features (political, religious, philosophical opinions, sexual orientation, ethnicity...)
- Non-targeted extraction of facial images from the Internet or video surveillance to create facial recognition databases
- Emotion recognition in the workplace and educational institutions
- Social scoring based on social behavior or personal characteristics
- AI systems manipulating human behavior to circumvent free will
- AI used to exploit vulnerabilities in individuals (due to age, disability, social or economic situation)

However, there are certain exemptions for law enforcement services in case of a terrorist threat, search for victims, or certain defined crimes. These exceptions remain unacceptable to some human rights organizations such as Amnesty International, which considers that an "outright ban" is needed to "prevent the human rights harms that facial recognition inflicts". Finally, note that some of these prohibitions are already enforced in countries outside the EU. Five of them were specifically targeted by Reporters Without Borders as "State Enemies of the Internet" in the "Special report on Internet Surveillance" published in 2013: Bahrain, Iran, Syria, Vietnam, and China, whose Social Credit System, where citizens and businesses are given or deducted good points based on social behavior, had already been the focus of debates. Out of over 1 billion cameras in the world, more than 540 million are estimated to be Chinese.

Figure 1.3.3 - Cameras per 1,000 people (2)

1.3.2 Bias and Discrimination

The undeniable impact of AI on individuals is grounded in algorithms governed by rules and parameters. However, when these rules are biased, disproportionately affecting certain groups by giving undue weight to specific parameters, ethical concerns arise. Moreover, if these algorithms are trained on historical datasets reflecting and potentially amplifying societal biases, the consequences can be severe. This is particularly critical when biased algorithms handle sensitive human data, such as in healthcare, leading to discriminatory model decisions and significant infringements on fundamental human rights.

For instance, computer-aided diagnosis (CAD) systems have exhibited lower accuracy for black patients compared to white patients, resulting in racial bias. Another poignant example is Midjourney, a generative image model that demonstrated age and gender biases. When tasked with creating images of people in

specialized professions, the model consistently depicted older individuals as men, thereby reinforcing gender and age-related biases regarding women's roles in the workplace.

The sources of these biases are diverse, stemming from the dataset, the algorithms themselves (or other components of an AI system), or even human interpretation of model outputs.

- **Training Data Biases**: Models learn behaviors from training data, necessitating a thorough representation of the overall population without under-representation of any group. Omitting sensitive data when unnecessary helps avoid replicating discriminatory biases present in historical records. There may also be a change in the characteristics of our training dataset due to the passage of time. It is sometimes necessary to update this dataset (and then fine-tune the model) to avoid model obsolescence. In our work on person recognition in videos, efforts are made to ensure fair representation of each ethnicity or gender in the training dataset.
- **Algorithmic Biases**: These biases may arise from programming errors, algorithm design, or unanticipated uses or decisions related to data collection.
- **Cognitive Biases**: Humans are inevitably influenced by experiences or preferences, introducing cognitive biases into models and potentially leading to discriminatory conclusions.

Techniques like dataset rebalancing, model auditing, and ethical algorithm design can mitigate the risks of discriminatory biases. Transparency in the development process and collaboration with ethics and diversity experts are crucial for identifying and addressing potential biases.

The establishment of robust legal and normative frameworks is essential to ethically regulate the use of intelligent video surveillance systems. Many regulators and legislators are actively working to incorporate specific provisions preventing and correcting biases in AI, along with mechanisms for accountability in cases of unfair discrimination. Collaboration among governments, researchers, and practitioners is vital for crafting effective ethical and legal standards.

1.3.3 Security

The deployment of surveillance technologies can be exploited for malicious purposes, including intrusive monitoring or the unlawful collection of information. Specifically, cyberattacks may aim to deceive a model by "poisoning" its input data or collecting personal data used for training and model parameters. Robust security protocols are imperative to prevent any misuse of AI systems and must ensure the **integrity**, **availability**, and **confidentiality** of the model.

1.3.4 Transparency and Interpretability

An AI model is deemed "interpretable" when humans can easily comprehend its decisions and understand how a prediction is made. Conversely, when a model is too complex for human understanding, it is often labeled as a "black box." Thus, the transparency of an AI system relies less on the publication of code or datasets and more on the understanding and explicability of the predictions it generates.
The level of transparency required for an AI project depends on the impact of its technology. The greater the impact, the more significant the ethical considerations, necessitating a higher capability to explain its decisions. For example, an advertising recommendation system may require less interpretability than a medical diagnostic model.

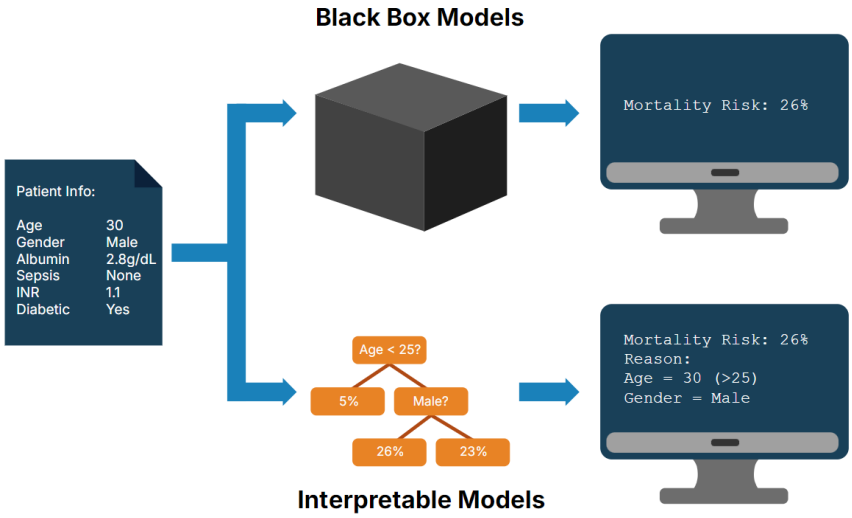

Figure 1.3.4 - Difference between black box and interpretable models (3)

In the case of video surveillance applications, sensitivity and impact on individuals can vary. When designing models, it is crucial to ensure that their level of interpretability is proportional to the significance of the problem they address. It is often advisable to implement alert systems without automated decision-making, to include human within the decision process.
There are model-specific inspection methods to gather information about the weight of variables in decision-making. The use of these methods also helps limit hidden biases and potential security vulnerabilities.

1.3.5 Ecological Impact

2023 was the warmest year on record, with global average temperatures 1.46°C above pre-industrial levels and 0.13°C above the eleven-month average for 2016. CO2 levels in the air have never been so high and the population sizes of mammals, fish, birds, reptiles and amphibians have experienced a decline of an average of 68% between 1970 and 2016, mainly due to human activity. In this context, it would be unthinkable not to consider the impact of AI systems that would be implemented in connection with video surveillance, using video data that is even heavier than text or audio data.
AI technological progress comes at an environmental cost, raising concerns about the sustainability of AI systems. The negative environmental impact of AI can be emphasized through issues such as carbon emissions, electronic waste, ecosystem disruption, and the lack of transparency and accountability.

The brilliance of AI conceals an energy-intensive process with a substantial carbon footprint. As AI models and datasets grow in complexity, so does the energy required for training and operation. OpenAI researchers reveal a doubling of computing power needed for cutting-edge AI models every 3.4 months since 2012 (in alignment with the principles of Moore's Law), projecting a significant share (14%) of global emissions from the Information and Communications Technology industry by 2040. A recent study indicates that training large AI models can emit a carbon footprint equivalent to 300 round-trip flights between New York and San Francisco, highlighting the urgency to address AI's role in climate change.

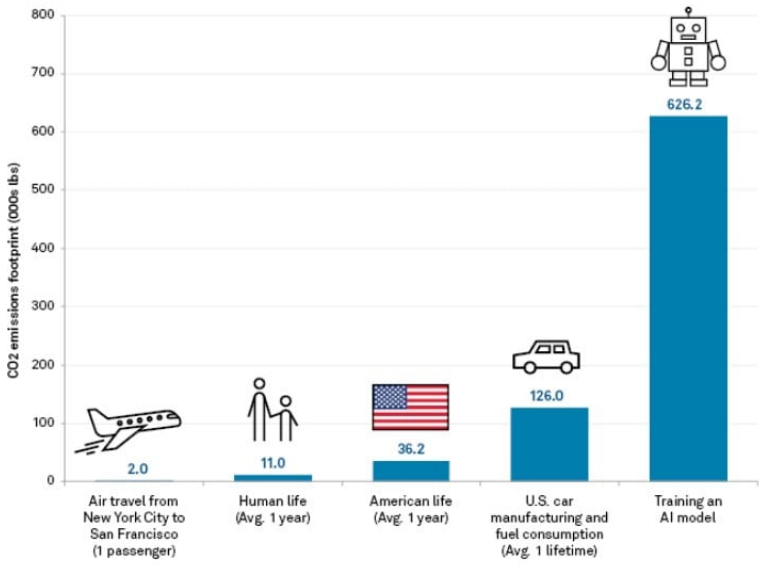

Figure 1.3.5 - CO2 emission benchmark (4)

Another environmental challenge for IA is the disposal of AI-related electronic waste. With hazardous chemicals like lead, mercury, and cadmium, e-waste endangers ecosystems and human health. By 2050, the World Economic Forum predicts e-waste to exceed 120 million metric tonnes. Proper e-waste management and recycling, enforced by stringent laws, are essential to minimize environmental harm.

In addition to posing challenges in terms of CO2 emissions and electronic waste, AI can exacerbate issues related to ecosystem disruption, as exemplified by the use of driverless automobiles. Furthermore, ethical concerns arise due to a lack of transparency and accountability, with some companies prioritizing their financial well-being and competitive edge over potential negative effects that AI technologies may have on the environment.

A multifaceted strategy is proposed to reduce AI's environmental impact. This includes funding research for energy-efficient hardware and algorithms, optimizing AI systems, promoting ethical design standards, and fostering a culture of openness and responsibility. Collaboration between stakeholders, governments, and regulatory agencies is crucial for implementing precise standards and restrictions, ensuring the ethical creation, use, and disposal of AI technologies.

AI's meteoric rise presents genuine environmental concerns, necessitating urgent attention. Recognizing and addressing the hidden environmental challenges, including the implications of Moore's Law, can initiate an informed dialogue and promote responsible practices. A sustainable future requires a harmonious integration of AI advancements and environmental preservation.

1.4 Contributions

Given the multifaceted nature of this project, it makes significant contributions to at least three computer vision tasks, culminating in the development of an end-to-end pipeline for anomaly detection in human crowds.

1.4.1 Multi-Object Detection

Efficient anomaly detection in a human crowd necessitates accurate human detection, a task complicated by numerous occlusions and varying object sizes. To address this, we curated a novel dataset by amalgamating multiple existing datasets, aiming for enhanced versatility, comprehensiveness, and global

representativeness. This dataset incorporates diverse image capture standards and better representation of various ethnic groups.

The project explores distinct model architectures, training two object detectors -One-Stage and Two-Stage- on the new dataset using various approaches, followed by a comprehensive performance comparison. Additionally, it delves into data processing intricacies, providing a robust foundation for future human detection research in crowd scenarios and exploring alternative promising approaches.

1.4.2 Multi-Object Tracking

The efficacy of each module within the resulting pipeline is contingent on the performance of the preceding module. Multi-object tracking involves assigning unique identifiers to objects across video frames, demanding a reliable object detector and contributing to anomaly detection capabilities.
We evaluate several tracking algorithms with diverse structures, establish an evaluation benchmark based on multiple metrics, and contemplate hybrid algorithms adaptable to diverse video sources, prioritizing versatility over performance.

1.4.3 Anomaly Recognition

Anomaly detection in videos is an area that often receives less attention compared to other computer vision domains. The challenges stem from the absence of a 'classic' use case, the more restrictive nature of data annotation, and the prerequisite for a preceding multi-object tracking module. Compounded by the lack of a universally accepted evaluation benchmark, this area presents complexities within the scope of a time-constrained research project.
Within this section, we contribute by implementing algorithms targeting three specific anomalies: evaluating crowd density, detecting unusual presence in defined areas, and identifying instances of individuals falling.
To enhance fall detection, we integrate the state-of-the-art ViTPose [5] model, a Vision Transformer dedicated to accurate body pose estimation. Furthermore, we explore additional strategies for anomaly detection, leveraging the capabilities of deep learning models to deepen our understanding of the field.
The proposed solutions, although rooted in simplicity, fill a gap in research by presenting high-performance, adaptable algorithms for these specific use cases. These contributions may serve as a valuable foundation for future endeavors in anomaly detection research, particularly in scenarios involving human crowds.

1.4.4 End-to-end Pipeline through a Web Application

The culmination of our efforts results in a proof of concept—a robust end-to-end pipeline connecting Multi-Object Detection, Multi-Object Tracking, and Anomaly Detection modules. This is complemented by the creation of a demonstrative web platform using Streamlit.
The platform allows users to interactively explore each module, select specific models or algorithms for inference, download the output and includes direct visualization features. We emphasize the intuitiveness of the platform, accompanied by comprehensive documentation to facilitate user-friendly navigation and adoption.

2 Theory

This section aims to introduce the main concepts discussed in this thesis. We assume a background in computer science, machine learning and mathematics on the part of the reader.

2.1 Video data and video processing

Video is a dynamic representation of visual information, comprising a sequence of images referred to as frames displayed at a rapid rate. In the digital realm, images are stored as a mosaic of tiny squares called pixels. These pixels store color information, represented as numerical values. Each frame captures a specific moment in the visual scene, and the rapid succession of frames creates the illusion of motion. Crucial factors influencing video quality include resolution and frame rate.

Similar to audio signals, video signals can be expressed numerically as a time series. In our digital context, each pixel's color and intensity are assigned numerical values, creating a digital representation of visual content. For this project, we'll be working mainly with Red-Green-Blue (RGB) vector images. In RGB, each pixel is represented by three numbers corresponding to red, green, and blue values, each ranging from 0 to 255 (as these are 8-bit numbers). The combination of these values generates a specific shade among the 16.8 million possible color combinations.

The spatial resolution of a video, denoting the number of pixels per unit area, dictates the clarity and detail of visual information. Common resolutions include High Definition (HD) with 1920x1080 pixels and Ultra High Definition (UHD) with 3840x2160 pixels. A UHD image is represented as an array of size (3840, 2160, 3), with 3840 as the width, 2160 as the height, and 3 representing the RGB values of each pixel.

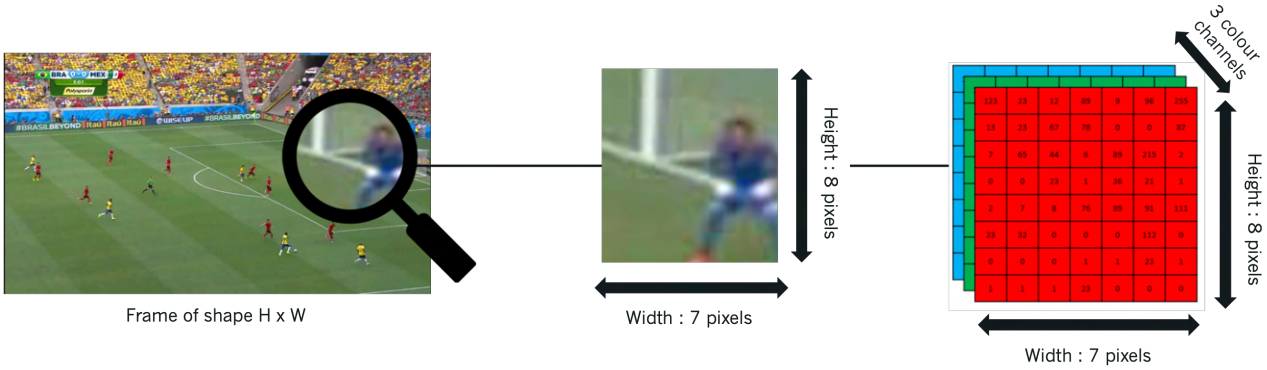

Figure 2.1.1 – Schematic representation of a RGB image

In our project pipeline, we may find it necessary to resize video frames. However, given that digital electronics operate with discrete numbers while human experiences are often analog, we need to consider the sampling rate. The Shannon-Nyquist theorem helps us determine the optimal point at which all necessary information is recorded without unnecessary resource consumption. Anti-aliasing techniques are employed where needed to prevent distortion.

The frame rate, measured in frames per second (fps), indicates how many individual frames are displayed in one second. Standard frame rates range from 24 fps for cinematic content to 60 fps for smoother

motion. In this project, video frames are typically sampled at a rate of fps = 30, meaning a 1-second video sequence consists of 30 frames.

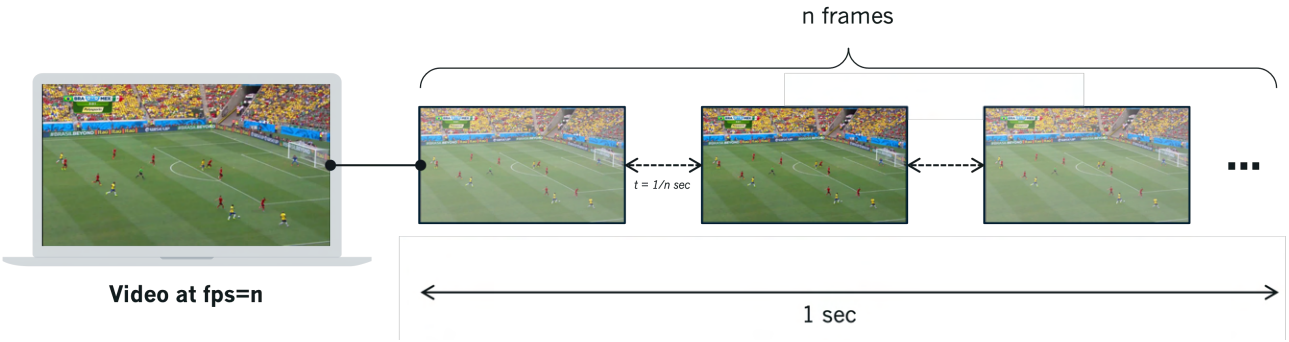

Figure 2.1.2 - Video schematic

We are talking about "offline" processing when analyzing pre-recorded video data after the recording has completed, and "online" processing when analyzing video in real-time as it is being captured or streamed. During this project, we will use an offline process, but we will try to develop models and algorithms that can meet the requirements of an online process, particularly in terms of frames processed per second (we will use a real-time offline processing).

2.2 Computer Vision

2.2.1 Essential Concepts

In the descriptions of our tracking algorithms and evaluation metrics, we will touch upon certain notions or objects from Computer Vision that are essential to introduce for a nuanced understanding of the concepts presented.

2.2.1.1 IoU (intersection over union)

IoU, or Intersection over Union, is a commonly used metric in the field of computer vision for assessing the accuracy of object detection algorithms. It measures the area of overlap between the predicted bounding box and the ground truth bounding box relative to the area of their union. This metric provides an indication of how accurately objects are localized within an image. A higher IoU value signifies a better match between the model's predictions and the actual location of objects.

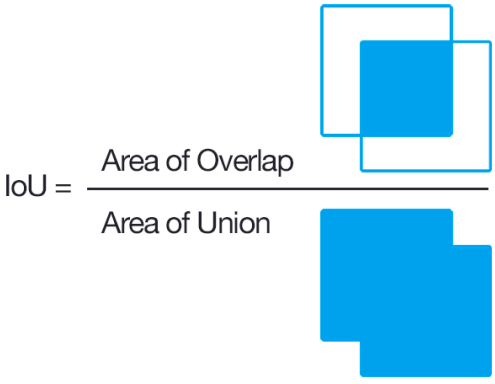

Figure 2.2.1 - IoU measure between two bouding boxes (5)

2.2.1.2 Kalman Filter

The Kalman filter is an algorithm used for estimating the state of a dynamic system from a series of noisy measurements. Within tracking algorithms, the Kalman filter is employed to estimate and predict the trajectories of moving objects based on their previous positions.

Mathematically, the Kalman filter follows two main steps: prediction and update. In a discrete context, it acts as a recursive estimator: to estimate the current state, only the previous state's estimate and the current measurements are needed.
The prediction of the actual state is achieved through the following linear evolution law:

$$X_{k|k-1} = F_k X_{k-1|k-1} + B_k u_k$$

and predicting the covariance of the estimation error:

$$P_{k|k-1} = F_k P_{k-1|k-1} {F_k}^T + Q_k$$

with:

- $X_{k-1|k-1}$, the state estimate at time k based on all observations up to time $k-1$
- $P_{k-1|k-1}$, the estimate error covariance at time k based on all observations up to time $k-1$
- F_k, the state transition matrix for the time step from $k-1$ to k
- B_k, the control input model at time k
- u_k, the control vector at time k
- Q_k, the process noise covariance at time k

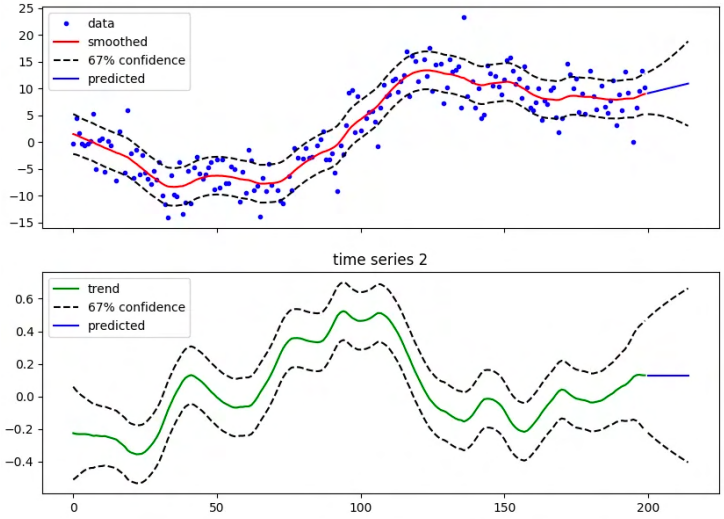

Figure 2.2.2 - Performance exemple of Kalman Filter on a Time Series (6)

In practical terms, the Kalman filter enables a more efficient track association process between detections at state k and the position estimates of detections from state $k-1$, as opposed to directly using the detections from state k-1 while assuming the object's inter-frame movement to be negligible. This method significantly improves the accuracy of tracking by dynamically adjusting the predicted positions based on previous movements and the latest observations, thereby accommodating for any potential changes in speed or direction of the tracked objects.

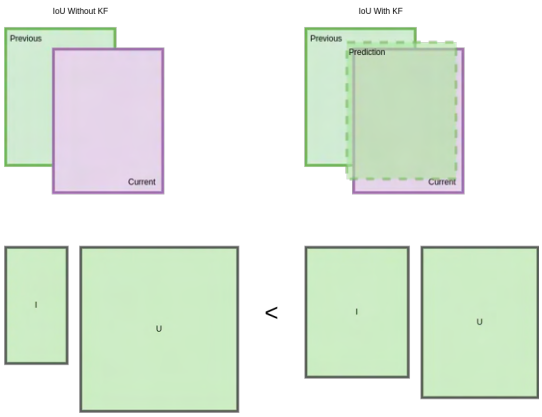

Figure 2.2.3 - Advantage of using Kalmann filter predictions for IoU (6)

2.2.1.3 **Hungarian Algorithm**

The Hungarian algorithm is utilized to optimally match detected objects with tracked objects in tracking. Its goal is to minimize the total cost of association by assigning each detected object to a tracked object, based on criteria such as spatial proximity and similarity between object features.

Here's a general outline of its 6 main operations:

1 - Initialization

Applied to tracking task, the algorithm initializes the cost matrix representing the distance between detected objects in two successive images (using future position estimates from the Kalman filter).

Let C be an $n * n$ cost matrix where $C_{i,j}$ represents the cost of associating element i from the first partition with element j from the second partition.

2 - Matrix Reduction

The algorithm performs operations to reduce the initial cost matrix. This generally involves subtracting the minimum of each row and each column from the cost matrix, so that each row and each column contain at least one zero.

For each row of matrix C, subtract the row minimum from all elements in that row. For each column of matrix C, subtract the column minimum from all elements in that column.

3 - Zero Selection

The algorithm selects zeros in the cost matrix in such a way that no zero shares the same row or column. This ensures that each element is associated with only one other element.

Select a set of zeros in C so that no zero shares the same row or column. Mark a potential association for each selected zero.

4 - Zero Marking

Once zeros have been selected, the algorithm marks these zeros as "potential associations." Then, it tries to mark as many zeros as possible while maintaining the goal of associating each element with only one other element.

Mark as many zeros as possible while adhering to the rule of not sharing the same row or column.

5 - Feasibility Test

The algorithm checks if a feasible solution has been found. A feasible solution means that it's possible to associate each element with only one other element without violating the algorithm's constraints.

Check if each element is associated with exactly one other element. If so, a feasible solution has been found. Otherwise, proceed to the next step.

6 - Cost Updating

If a feasible solution has not been found, the algorithm updates the costs in the matrix to make a new attempt more likely. This involves adjusting the costs of unselected elements or other strategies aimed at facilitating the selection of a feasible solution.

If a feasible solution hasn't been found, adjust the costs in matrix C to make a new attempt more likely. For example, if no zero was selected in a row or column, increase all elements in that row or column by the same value.

Repetition

Steps 2 through 6 are repeated until a feasible solution is found. Once a feasible solution is identified, it is considered the optimal solution to the association problem.

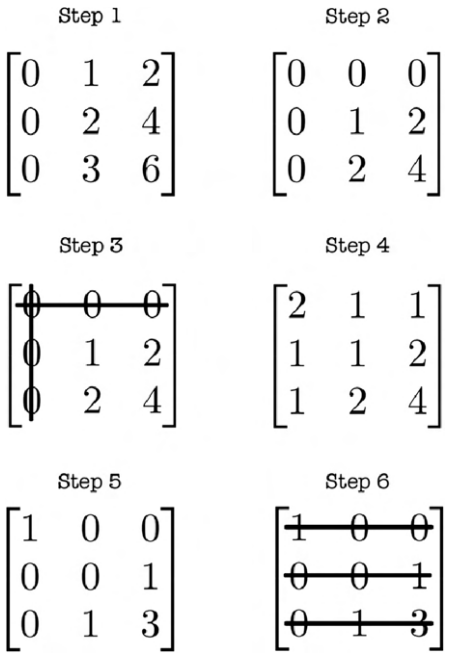

Figure 2.2.4 - Process of Hungarian Algortihm [29]

The Hungarian algorithm guarantees finding the optimal solution for the bipartite matching problem. However, its complexity is $O(n^3)$, where n is the size of the cost matrix, making it less efficient for processing large instances. Despite this, it remains widely used due to its guarantee of finding the optimal solution.

2.2.2 Discussed Tracking Algorithms

For our benchmark, we have selected detection-based tracking algorithms that exhibit various levels of complexity and have provided, or are currently offering, state-of-the-art performance. We prefer versatile algorithms, those that are plug-and-play or can be easily integrated into an architecture if possible.

Therefore, we will work with SORT (Simple Online and Real-time Tracking) [6], DeepSORT (Simple Online and Realtime Tracking with a Deep Association Metric) [7], BYTETrack [8] and SparseTrack [9].

2.2.2.1 **SORT (Simple Online and Real-time Tracking)**

SORT is a straightforward implementation of a multi-object tracking framework based on basic data association and state estimation techniques. It's designed for online tracking applications where only past and current images are available, and where the method generates track identities on the fly. It lacks specific features to handle occlusion or the re-introduction of objects; its aim is to provide a foundation for the development of more sophisticated algorithms. Understanding its structure and operation is crucial for discussing other tracking techniques.

The operation of SORT revolves around three key aspects: estimation, association, and the creation and deletion of tracks.

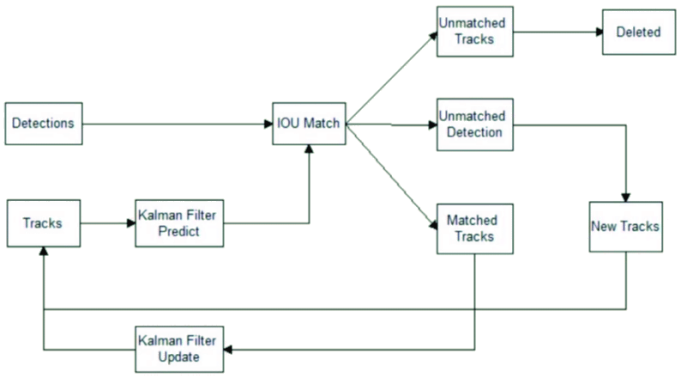

Figure 2.2.5 - SORT process (7)

Estimation

The estimation phase of SORT is tasked with predicting the future positions of tracked objects in video frames. In other terms, it involves estimating the inter-frame movements of objects. To achieve this, SORT relies on a linear constant velocity model, assuming that each target moves at a constant speed between frames. This model is applied independently to each tracked object, without considering the movement of the camera or other objects.
The state of each track (i.e., identity) is represented by a state vector, typically denoted as x, which includes information such as the horizontal and vertical position of the target's center, size, aspect ratio, as well as velocity components in each direction. When a detection is associated with a track, the detection's bounding box is used to update the track's state. The adjustment and estimation of the target's velocity components are performed using a Kalman filter, an algorithm generally used to estimate the state of a dynamic system from a series of noisy measurements.

Association

The association phase of SORT optimally matches object detections in a frame with tracks from previous images and their position estimates provided by the estimation phase, all while minimizing a defined association cost.
This association cost is a measure of the likelihood that a detection is associated with a track. It is calculated between each possible pairing as the Intersection over Union (IoU) distance of the areas of the predicted and estimated position bounding boxes. In more recent tracking algorithms, the association cost can incorporate features other than IoU, such as a measure of appearance similarity or contextual information about the image.
Using the cost matrix of associations, SORT seeks to find the optimal combination that minimizes the total association cost. This task is executed by a Hungarian algorithm. As a result of this combinatorial algorithm, each combination of associations is rejected if the IoU between the two proposals is less than a minimum IoU threshold defined as a parameter of the algorithm.

Identities Creation

When objects enter and exit the frame, there is a need to assign or remove their identities, represented by unique ID numbers.

For identity creation, SORT simply treats all detections not matched by the Hungarian algorithm and presenting a confidence score above a certain threshold as a new track. This new track is initialized using the coordinates of the bounding box, and its velocity is set to zero. The covariance of the velocity component is set with high values to reflect the uncertainty of the measurement. To avoid a high number

of false positives, any new track is subjected to a probationary period during which it must be associated with other detections to be validated.
A track will be deleted if it is not associated with any detection for a defined number of frames. This helps avoid having too many tracks to consider, which, in addition to better reflecting the reality of targets exiting the field of view, contributes to the algorithm's efficiency.

2.2.2.2 DeepSORT (Simple Online and Real-time Tracking with a Deep Association Metric)

DeepSORT is an enhancement of the SORT framework that aims to better handle occlusions and thus reduce the number of identity switches during tracking.
Its operation is based on that of SORT, with a few notable differences, the most significant of which is the incorporation of a pre-trained CNN (Convolutional Neural Network) to extract significant features from the image within the bounding box of the object. These extracted features are represented as feature vectors. Each detected object is thus associated with a feature vector that abstractly represents it in a vector space. In our case, the CNN inherent to DeepSORT is specifically trained for pedestrian discrimination on a large dataset for person re-identification. These feature embeddings allow for the replacement of the previous association metric with a new, higher-level one that combines information on the movements of objects with appearance characteristics. This results in a more precise and robust association by capturing richer information on the appearance and characteristics of objects.

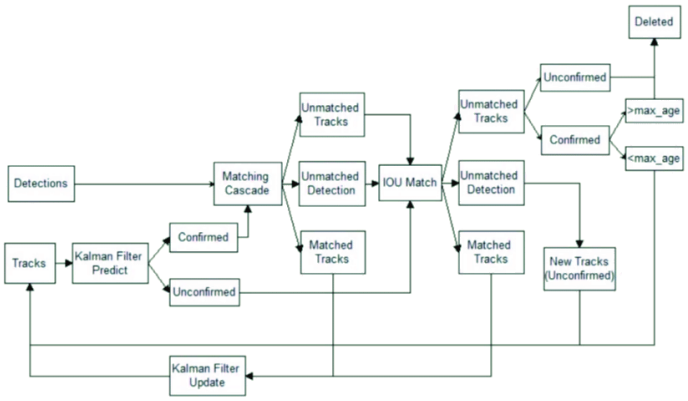

Figure 2.2.6 - DeepSORT process (8)

New Association Metric

More specifically, this new metric is based on the calculation of two distances - the Mahalanobis distance and a cosine similarity distance.

The Mahalanobis distance calculation between the estimated states of the Kalman filter and the new detections allows for better consideration of the inter-frame movements of objects. It is obtained using the following formula:

$$d^{(1)}(i,j) = (d_j - y_i)^T S_i^{-1} (d_j - y_i)$$

Where (y_i, S_i) represents the projection in our measurement space of track i and d_j that of the bounding box of the j-th detection.

Just as with updating the states of the Kalman filter, the Mahalanobis distance considers its uncertainty by measuring how many standard deviations the detection is away from the track.

In its associative process, DeepSORT incorporates a distance based on an appearance descriptor obtained through a CNN. For each detection d_j, the CNN calculates an appearance descriptor r_j whose

norm equals 1. For each track k a gallery of appearance descriptors $R_k = \{r^{(i)}{}_k\}^{L_k}{}_{k=1}$ is maintained over the last $L_k = 100$ last frames.
The second distance used by DeepSORT is the measure of cosine similarity between the aspects of the i-th track and the j-th detection:

$$d^{(2)}(i,j) = \min\{1 - r_j{}^T r^{(i)}{}_k \mid r^{(i)}{}_k \in R_j\}$$

This cosine distance specifically addresses the issue of sudden camera movements altering the Mahalanobis distance measurements and cases of prolonged occlusions.

The new association metric used by DeepSORT is obtained by combining these two distances with the formula:

$$c_{i,j} = \lambda d^{(1)}(i,j) + (1-\lambda)\mathrm{d}^{(2)}(\mathrm{i},\mathrm{j})$$

where $c_{i,j}$ is the association cost of track i with detection j, and λ is a hyperparameter associated with the consideration of each of the two distances in the final cost. It appears prudent to adjust λ towards a lower value in the presence of abrupt camera movements that distort position estimation.

Cascade Association

DeepSORT addresses the association problem through cascaded matching. This matching cascade prioritizes objects that have been seen more frequently, by first associating detections with the most recent tracks before considering lost tracks. This methodology accounts for the increasing uncertainty associated with the position of objects when they are occluded for an extended period.

2.2.2.3 BYTETrack

BYTETrack is a state-of-the-art tracking algorithm designed to address the issue of high false-negative rates revealed by the confidence threshold defined within SORT. To overcome this, BYTETrack implements a new method of association by using almost all detections, rather than only those with a high confidence score.
The foundational premise of BYTETrack is inspired by a quote from Hegel: "What is rational is real; what is real is rational." Hegel believed that the universe is rational, and everything that happens in the real world is necessarily rational, even if it may seem irrational or contradictory at first glance.
By extending this logic, it can be considered that low-confidence detections sometimes indicate the existence of objects, often occluded ones. Filtering out these objects leads to tracking errors and results in significant detection omissions and fragmented trajectories.

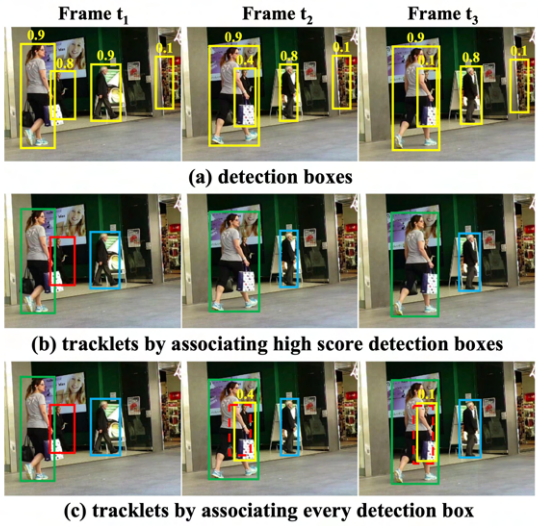

Figure 2.2.7 - Reasonning behind ByeTrack. In dot, the predictions of Kalmann Filter [8]

BYTETrack first matches bounding boxes with the highest scores to tracks based on movement or appearance similarity (in our case, we will use a motion-based association metric as in SORT). The result obtained after this initial association step corresponds to part (b) of the figure mentioned earlier.
Subsequently, the remaining tracks and detections with lower scores are matched using the same principle as in the first phase. This results in less fragmented and more robust tracking, as illustrated in part (c) of the figure above.
Due to its simplicity, an infinite number of other methods can be compatible with BYTETrack. In the aftermentioned pseudocode, for instance, we have the freedom to choose any similarity measure for association, whether it's based on re-identification methodology, attention mechanisms, movement evaluation, or chain creation. In its research paper, BYTETrack is applied to enhance 9 state-of-the-art trackers.

Algorithm 1: Pseudo-code of BYTE.

Input: A video sequence V; object detector Det; detection score threshold τ
Output: Tracks $\mathcal{T}$ of the video

Initialization: $\mathcal{T} \leftarrow \emptyset$
for *frame* f_k *in* V **do**
  /* Figure 2(a) */
  /* predict detection boxes & scores */
  $\mathcal{D}_k \leftarrow \texttt{Det}(f_k)$
  $\mathcal{D}_{high} \leftarrow \emptyset$
  $\mathcal{D}_{low} \leftarrow \emptyset$
  for d *in* $\mathcal{D}_k$ **do**
    if $d.score > \tau$ **then**
      $\mathcal{D}_{high} \leftarrow \mathcal{D}_{high} \cup \{d\}$
    end
    else
      $\mathcal{D}_{low} \leftarrow \mathcal{D}_{low} \cup \{d\}$
    end
  end
  /* predict new locations of tracks */
  for t *in* $\mathcal{T}$ **do**
    $t \leftarrow \texttt{KalmanFilter}(t)$
  end
  /* Figure 2(b) */
  /* first association */
  Associate $\mathcal{T}$ and $\mathcal{D}_{high}$ using Similarity#1
  $\mathcal{D}_{remain} \leftarrow$ remaining object boxes from $\mathcal{D}_{high}$
  $\mathcal{T}_{remain} \leftarrow$ remaining tracks from $\mathcal{T}$
  /* Figure 2(c) */
  /* second association */
  Associate $\mathcal{T}_{remain}$ and $\mathcal{D}_{low}$ using similarity#2
  $\mathcal{T}_{re-remain} \leftarrow$ remaining tracks from $\mathcal{T}_{remain}$
  /* delete unmatched tracks */
  $\mathcal{T} \leftarrow \mathcal{T} \setminus \mathcal{T}_{re-remain}$
  /* initialize new tracks */
  for d *in* $\mathcal{D}_{remain}$ **do**
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{d\}$
  end
end
Return: $\mathcal{T}$

Track rebirth [70,89] is not shown in the algorithm for simplicity. In green is the key of our method.

Figure 2.2.8 - Pseudo-code of ByteTrack [8]

2.2.2.4 SparseTrack

SparseTrack is among the most recent advancements in state-of-the-art multi-object tracking, following the detection-association paradigm. It aims to address both the limitations of BYTETrack when applied to sequences with numerous occlusions and those of methods using temporal modeling mechanisms or trajectory querying, which are often more computationally expensive and not suitable for online use.
SparseTrack introduces the concept of "pseudo-depth" for bounding boxes. Based on the assumption that the camera is positioned above a relatively flat ground, this measure aims to simulate the consideration of a third dimension for association purposes. It represents the "pseudo-depth" of a

bounding box by the distance between its bottom and the bottom of the image. The pseudo-depth is used to modify prioritization in the association process, based on the premise that among adjacent bounding boxes, depths play a more significant role than the bounding boxes' coordinates in linking detections to tracks.

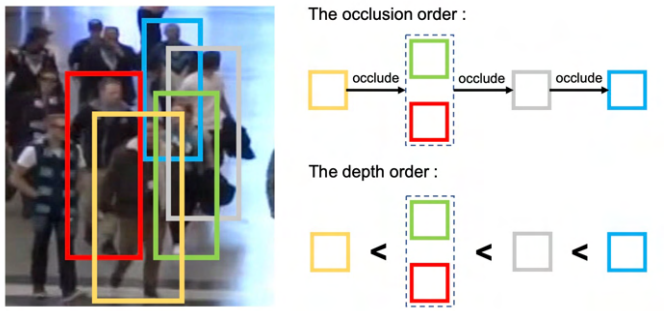

Figure 2.2.9 - Occlusion Order vs Pseudo-Depth order [9]

Based on this notion of pseudo-depth, SparseTrack then introduces a Depth Cascade Matching (DCM) algorithm that associates tracks and detections based on pseudo-depth. More specifically, DCM segregates tracks and detections into subsets based on the distributions of their pseudo-depths. Then, the algorithm performs IoU-based association in ascending order of pseudo-depth (from closest to farthest) for each defined subset.

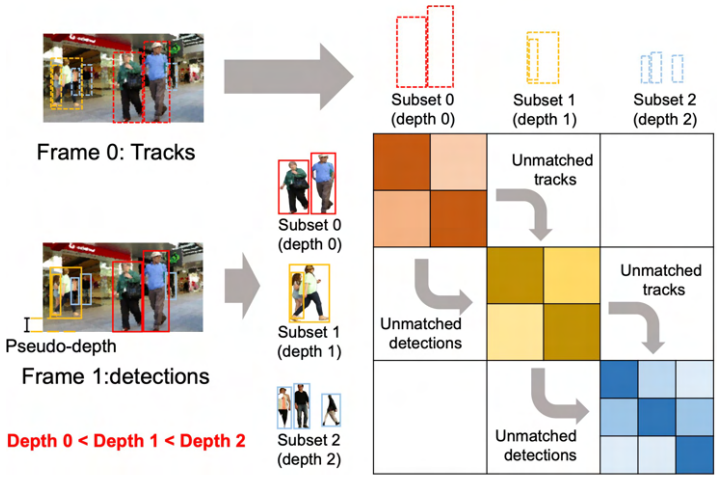

Figure 2.2.10 - Functionning of DCM [9]

Finally, in the spirit of BYTETrack, SparseTrack also considers almost all bounding boxes, separating detections into two sets based on their confidence scores. The DCM algorithm is applied to each set of detections. Detections with lower scores will also receive tracks and detections unassociated from the DCM applied to detections with higher scores.

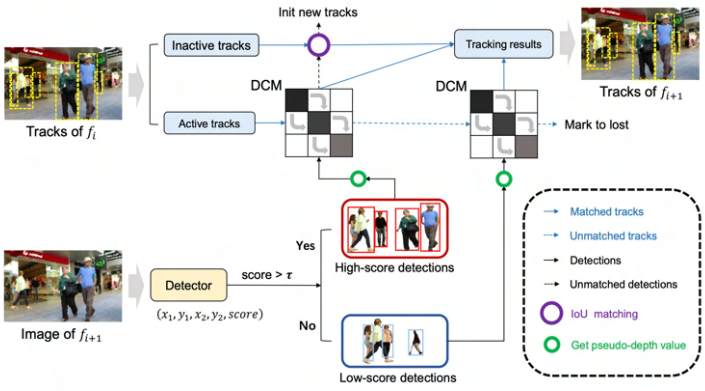

Figure 2.2.11 - Global Functionning of SparseTrack [9]

Similar to the previously discussed trackers, SparseTrack meets one of our main selection criteria by being plug-and-play. It can be easily integrated with other trackers due to its versatility and the simplicity of its architecture, the essence of SparseTrack being its Depth Cascade Matching (DCM) algorithm.

2.2.3 Metrics

Within this work, we will be required to assess tracking algorithms. As such, it is imperative to select well-chosen and complementary metrics that reveal all the performance characteristics of a tracking system. We will choose to base our benchmark on three metrics: HOTA [10], MOTA [11], and IDF1 [12].

In this part, the "detections" we refer to will be of the same nature as those produced by our implemented detector. However, the presence of thresholds and operations inherent to tracking algorithms, acting at the level of detections, can modify them. Therefore, it is the detections outputted by our trackers that will be addressed here.

2.2.3.1 HOTA Metric (Higher Order Tracking Accuracy)

The HOTA metric can be viewed as a blend of three IoU (Intersection over Union) scores related to performance in detection, localization, and association aspects of our system, thus considering every critical facet of a tracking algorithm.

IoU Score Related to Detection

This score aims to assess the alignment between detections within the ground-truth trajectory and the predicted trajectory. To achieve this, it employs a principle similar to the one used in the association process of our tracking algorithms: it defines an IoU threshold (Loc-IoU) above which the predicted and ground-truth detections are considered to intersect, and then a one-to-one correspondence is determined between detections of both trajectories via the Hungarian algorithm. Each pair of associations is termed a "True Positive" (TP), and thus, the set of TPs can be seen as the intersection between the two sets of detections. Detections within the ground-truth and predicted trajectories that are not associated are labeled as "False Negatives" (FN) and "False Positives" (FP), respectively.

Det-IoU is obtained by the formula:

$$Det\ IoU = \frac{|TP|}{|TP| + |FN| + |FP|}$$

DetA (detection accuracy) is the Det-IoU metric generalised to all our trajectories:

$$DetA = \ Det\ IoU = \frac{|TP|}{|TP| + |FN| + |FP|}$$

IoU Score Related to Localisation

The localization score, referred to as Loc-IoU, measures the localization accuracy of a True Positive (TP). It is simply calculated by dividing the intersection of the predicted and ground-truth bounding boxes by their union. Hence, it serves as a measure of the quality of the predicted positions.

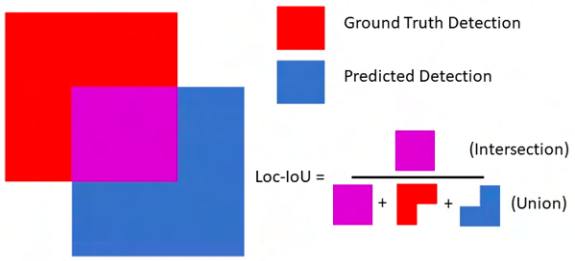

Figure 2.2.12 - Fonctionning of IoULoc (9)

This metric is generalized across all trajectories by calculating the average of the Loc-IoU scores computed for each association:

$$LocA = \frac{1}{|TP|} \sum_{c \in TP} Loc\ IoU(c)$$

The HOTA metric is unique in our benchmark for incorporating a measure of the localization of predictions.

IoU Score Related to Association

Finally, this last IoU score is dedicated to measuring a tracker's effectiveness in maintaining a coherent link over time between detections and an identity (i.e., a track). Unlike DetA, which informed about the proportion of detections that had been associated, AssA will provide us with information on the quality of these associations, on the alignment between predicted and actual tracks.

It measures the overall alignment on each True Positive (TP) between the entire trajectories associated with the predicted and ground-truth detections. This alignment (named Ass-IoU) is obtained with:

$$Ass\ IoU = \frac{|TPA|}{|TPA| + |FNA| + |FPA|}$$

where TPA (True Positive Associations), FNA (False Negative Associations), and FPA (False Positive Associations) represent the numbers of correctly made detection associations, those from the ground-truth trajectory not present in the predicted trajectory, and those from the predicted trajectory not present in the ground-truth, respectively. FPAs may correspond to another ground-truth trajectory or possess no identity.
These values can be illustrated in the figure provided below:

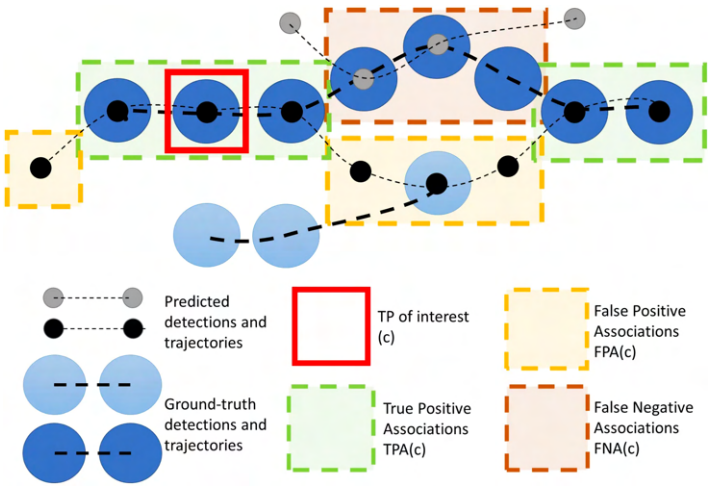

Figure 2.2.13 - Illustration of TPA, FNA, FPA values (10)

The process of calculating the track association score is performed offline (once the tracking is fully completed and the predictions of future positions of a track are already known).

Loc-IoU, Det-IoU, and Ass-IoU share the same fundamental structure: a ratio between intersection and union. While Det-IoU informs us about the macro correspondence between the detections of the predicted trajectory and the ground-truth trajectory, Ass-IoU and Loc-IoU provide more detailed information on the inter-detection alignment between the two tracks and the quality of the bounding box alignments of the associations, respectively. It is important to note that Det-IoU operates solely based on detections without considering the quality of the tracker's association, whereas, conversely, Ass-IoU is based on the robustness of the association, independent of the number of detections considered.

In the same way as mentioned previously, we are able to generalize this measure across our entire dataset (AssA score) by calculating the average of the scores obtained for each TP:

$$AssA = \frac{1}{|TP|} \sum_{c \in TP} Ass\ IoU(c)$$

$$AssA = \frac{1}{|TP|} \sum_{c \in TP} \frac{|TPA(c)|}{|TPA(c)| + |FNA(c)| + |FPA(c)|}$$

Combining IoU Scores

The HOTA metric is defined as the geometric mean of the IoU scores DetA and AssA using the following formula:

$$Hota_{\alpha} = \sqrt{DetA_{\alpha} \times AssA_{\alpha}}$$

$$Hota_{\alpha} = \frac{\sqrt{\sum_{c \in TP} Ass\ IoU_{\alpha}(c)}}{|TP_{\alpha}| + |FN_{\alpha}| + |FP_{\alpha}|}$$

$$Hota = \int_{0<\alpha<1} Hota_{\alpha}$$

$$Hota = \frac{1}{19} \sum_{\substack{\alpha=0.05 \\ \alpha+=0.05}}^{0.95} Hota_{\alpha}$$

Each of the metrics DetA and AssA is defined using a Hungarian algorithm based on a certain threshold value of Loc-IoU. Here, α corresponds to this threshold value.
The HOTA metric that is ultimately used is obtained by integrating the $HOTA_{\alpha}$ scores obtained for each value of α ranging from 0.05 to 0.95, with a step of 0.05.

2.2.3.2 MOTA Metric (Multiple Object Tracking Accuracy)

The MOTA metric is a performance measure for tracking that operates by iterating over detections (as opposed to considering the tracks in their entirety). It is calculated by accounting for three types of errors: false positives (incorrect detections incorporated into the track - FP), false negatives (instances where the target is not included but should have been - FN), and identity switches (incorrect identity changes - IDSW).

Identity switches are tracking errors that occur when the system wrongly associates a detection from one track to the ground truth track. For example, if in frame 1, the correspondence is established between the ground truth track P1 and the tracking model's track (ID3) recorded as the pair (1, 3). If, in the next frame, the identity P1 is associated with a different identity (ID4), the mismatch error (IDSW) is incremented by 1. The pair (1,3) will be removed, and the pair (1,4) will be considered in subsequent frames.
With this information, we can calculate the MOTA value by subtracting from 1 the sum of false positives, false negatives, and identity switches divided by the total number of true positives (gtDet), using the formula:

$$MOTA = 1 - \frac{|FN| + |FP| + |IDSW|}{|gtDet|}$$

2.2.3.3 IDF1 Metric (Integrated Detection and False Positive Identification)

The IDF1 metric performs a bijective matching between the set of real (ground truth) trajectories and the predicted trajectories. IDF1 defines new types of matches:

- IDTP (Identity True Positives): These are the associations of detections correctly established on the overlapping parts (where S ≥ α, S being an IoU threshold) of the trajectories being matched.
- IDFN (Identity False Negatives): These are the detections from the real trajectory not present in the predicted trajectory, either because the object was not detected by the detector or because the detection was matched with a different trajectory.
- IDFP (Identity False Positives): These are the predicted detections falsely associated with a trajectory. This can be due to the prediction of a non-existent object in the image or to the prediction of an existing object but falsely associated with the trajectory.

These matches allow us to calculate the IDF1 score using the following formula:

$$IDF1 = \frac{|IDTP|}{|IDTP| + 0.5|IDFN| + 0.5|IDFP|}$$

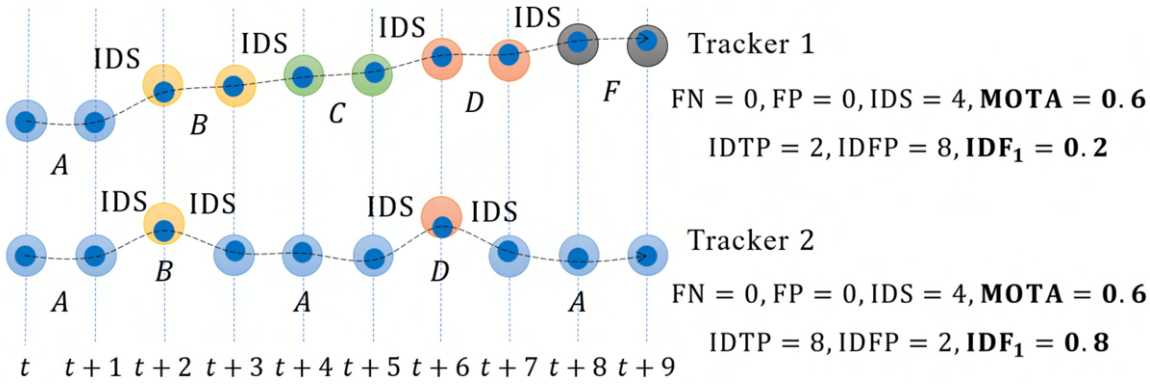

Figure 2.2.14 - Measurement difference between MOTA and IDF1 on examples on two trackers [30]

This metric is a better measure of the coherence and robustness of a track than the MOTA metric, which provides more insight into the completeness and conciseness of the detections defining the tracking.

Each of these metrics is expressed with a score from 0 to 1 (from 0 to 100 when expressed as a percentage), where 0 indicates a poor score and 1 a perfect score. They measure the performance of one or several characteristics of tracking, from detection to association and localization.
The HOTA metric is the most "balanced" and inclusive of each aspect of detection-based tracking. Moreover, it is the only one to integrate a measure of the localization of predictions. The MOTA metric provides information on detection performance levels, and IDF1 focuses on associations.

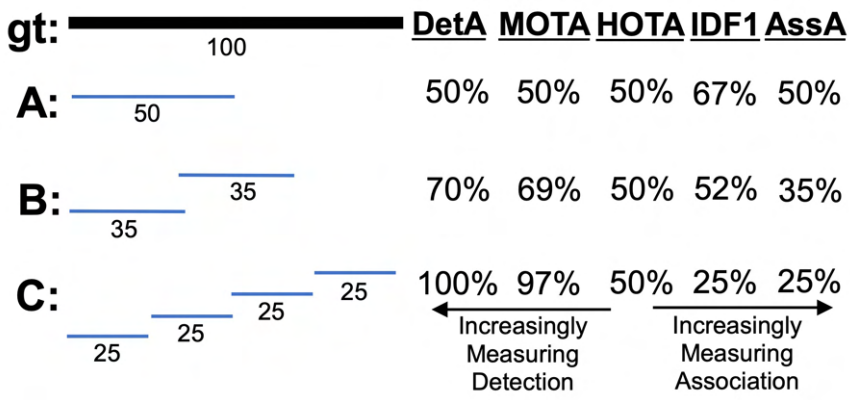

Figure 2.2.15 - Information carried by metrics (10)

The choice of the best tracker does not depend on a particular metric but rather on the specific application case. In our situation, where we do not prioritize one aspect over another, we will choose HOTA as the best metric, as it does not induce a particular bias between detection and association.

2.3 Neural network and Deep Learning

2.3.1 Glossary

In this work, we will explore various artificial intelligence algorithms designed to address challenges presented by diverse problem scenarios. This subsection provides precise definitions for the technical terms specific to artificial intelligence and introduces the methodologies that will guide our exploration.

- **Supervised / unsupervised learning:** These are the two main approaches of a machine learning problem. Supervised learning involves training a model using labeled data, where the algorithm learns to map input data to corresponding output labels. In unsupervised learning, the model explores the inherent structures of the dataset independently, without explicit guidance or labeled examples.

- **Classification / regression:** Classification and regression represent distinct tasks within supervised learning, each tailored to different prediction objectives. Classification focuses on assigning input data to predefined categories or classes, determining the appropriate label for a given input while regression is geared towards predicting numerical values, estimating a numerical output rather than discrete classes.

2.3.2 Definitions

Neural networks are drawing their inspiration from the workings of biological brains. It consists of interconnected nodes, or "neurons," organized in layers that collaboratively process and learn from input data. Each neuron receives information from neighbouring neurons, applies an activation function (for non-linearity), and transmits the output to subsequent neurons. The learning process involves optimizing a defined loss function, primarily dependent on the specific task at hand.

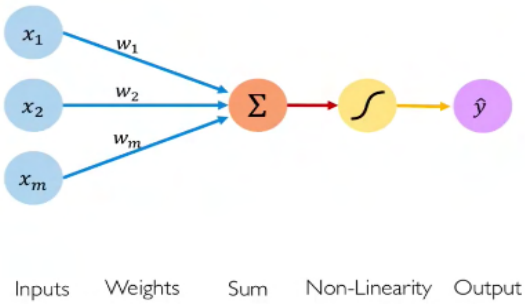

Figure 2.3.1 - Forward propagation of information through a single neuron (11)

Deep Learning (DL) represents a subset of machine learning techniques that relies on artificial neural networks (ANNs) incorporating feature learning (aka representation learning), a set of techniques enabling a system to autonomously discover the representations needed for feature detection or classification from raw data.

A neural network is deemed a "Deep Neural Network" (DNN) when it incorporates layers of hidden neurons between the input and output layers. The depth of the network increases with the number of layers.
There are many types of layers used to build a DNN including:

- **The convolutional layer (CONV):** The convolutional layer serves as the initial stage for extracting features from input data. In this layer, a convolution operation is applied. This operation is basically an aggregation of local pixel[2] information to capture relationships between neighboring pixels. Convolution is performed between the input data and a filter, also known as a "kernel", which slides across the input at different positions. At each position, an element-wise multiplication occurs between the values of the kernel matrix and the pixel values in the input it lines-up with. The results of these multiplications are summed to produce a single value, constituting the output of the convolution operation for that specific position of the kernel in the input.
  The output of this layer is referred to as "Feature Map" providing key information about the content within the input data. Each element of the feature map represents the activation of a specific neuron in the network, and its value represents the degree to which the corresponding feature of the neuron is present in the input.

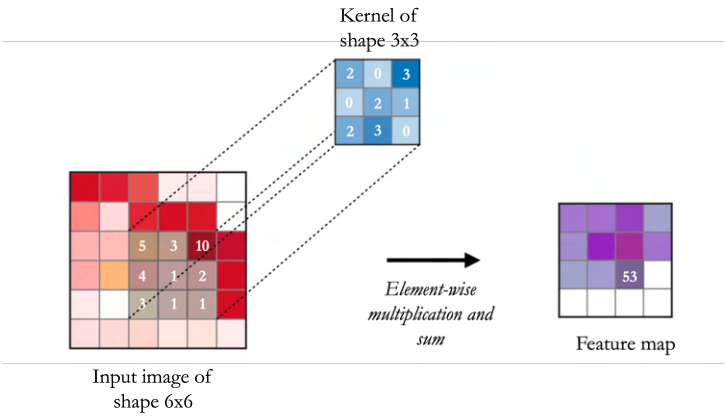

Figure 2.3.2 - Convolution operation applied to a 2D input. This process can be generalized in other dimensions (12)

[2] reminder: in the digital realm, images can be represented by arrays of size width x height x channels, whose values represent pixel colors or intensities.

The size of the output feature map is influenced by factors such as the size of the input, the size of the kernel, and the stride - the numbers of pixels the window moves after each operation. The number of kernels affects the depth of the output and the number of feature map generated within the convolutional layer.

- **The pooling layer (POOL):** The pooling layer is a sub-sampling operation typically applied after a convolutional layer when the dimension of the data is still too large. The principle is similar to that of the convolutional layer, but simpler. A filter "slides" over each region of our input data, and for each filter position, we retrieve the minimum, maximum, median or average value of the set of pixels concerned. The most popular types of pooling are max and average pooling.

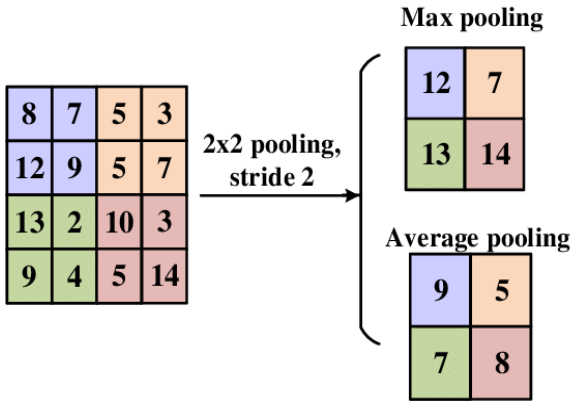

Figure 2.3.3 - Max and average Pooling layer operation oproaches [28]

- **The fully connected layer (FC):** The Fully Connected layer consists of weights, biases, and neurons, facilitating connections between two different layers. Typically positioned before the output layer, the fully connected layers are crucial components found in the latter part of a DNN architecture.
After flattening the data, the fully connected layer connects every neuron in one layer to every neuron in the next, facilitating the learning of complex patterns and hierarchical features. Fully Connected layers are crucial for tasks like classification, where the final layers contribute to making predictions based on the learned features.

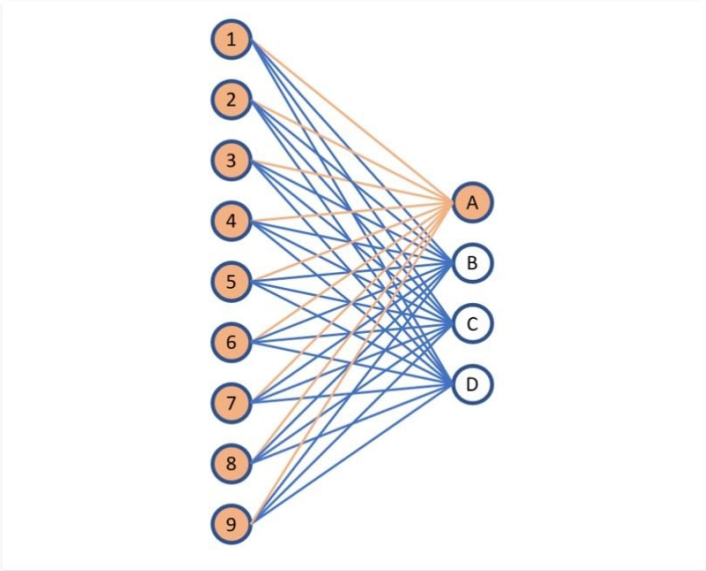

Figure 2.3.4 - Schematization of a convolutional layer vs a fully connected layer (13)

2.3.3 Main deep neural networks architectures

In supervised learning with Deep Neural Networks (DNNs), the main types of architectures commonly used are:

- **Feedforward Neural Networks (FNNs):** FNNs are the most basic form of neural networks. They consist of an input layer, one or more hidden layers, and an output layer. Information flows through the network in one direction, from the input layer to the output layer, without any cycles or loops. Each neuron in a layer is connected to every neuron in the subsequent layer. The activation of neurons is determined by a weighted sum of inputs, passed through an activation function. FNNs are versatile, easy to train and suitable for tasks where the input and output data have a clear mapping, making them widely used in pattern recognition tasks.

- **Recurrent Neural Networks (RNNs):** RNNs have connections that form a directed cycle, allowing them to capture sequential dependencies in data. They have a hidden state that retains information about previous inputs in the sequence. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are specialized RNN architectures designed to address the vanishing gradient problem. RNNs are suitable for tasks where the order of sequence of data is crucial, such as natural language processing, time series prediction, and speech recognition. The ability to retain memory of previous inputs makes them effective in handling sequential data.

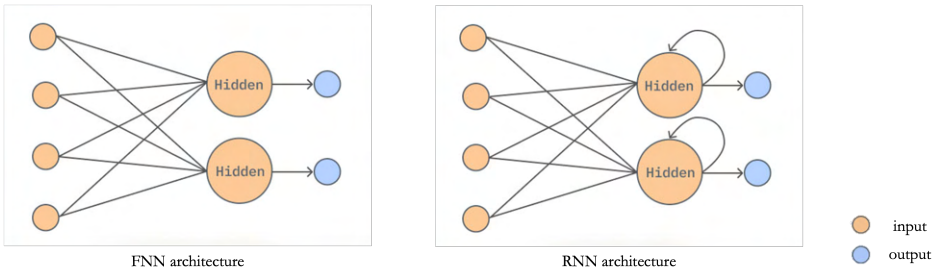

Figure 2.3.5 - FNN architecture vs RNN architecture (14)

- **Convolutional Neural Networks (CNNs):** CNNs are designed specifically for tasks involving grid-like spatially structured data, such as images. CNNs typically have convolutional layers followed by pooling layers and fully connected layers for classification. CNNs excel in capturing spatial hierarchies and local patterns. The convolutional layers help in detecting local features like edges, textures, and shapes, and the pooling layers reduce the spatial dimensions while retaining important information. CNNs are widely used in image classification, object detection, and image segmentation tasks. This structure will serve as the basis for the design of many of the models studied in this thesis.

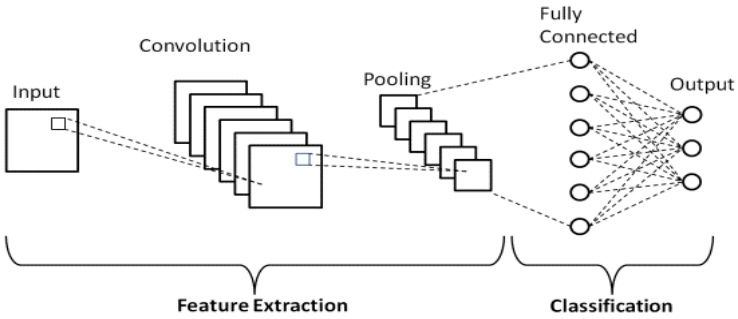

Figure 2.3.6 – Basic CNN architecture (15)

2.3.4 Deep neural networks for image processing

In image processing, DNNs especially CNNs, particularly useful for image processing, serve as the backbone for intricate feature extraction and pattern recognition. Several more complex architectures, including R-CNN (Region-based Convolutional Neural Networks) [13], FPN (Feature Pyramid Networks) [14] and RPN (Region Proposal Network) [15], have paved the way for more efficient and accurate object detection.

2.3.4.1 R-CNNs for image processing

The inability of standard CNNs to detect a variable number of objects in an image (due to the fixed length of its output layer) led to the need for a solution incorporating a region proposal module.
R-CNN pioneered a two-step approach to object detection, starting with region proposal and followed by classification.
During the region proposal phase, potential object regions are generated using selective search [25].
These cropped regions then undergo CNN-based feature extraction, transforming each region proposal into a fixed-length feature vector which will be fed into fully connected layers responsible for object classification and bounding box regression. The classification layer predicts the probability of the presence of an object within the region proposal, and the regression layer refines the bounding box coordinates to better fit the object.

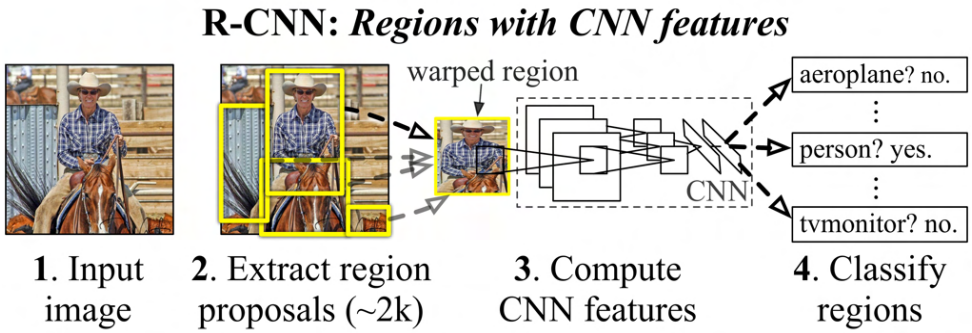

Figure 2.3.7 - Object detection system for R-CNN [13]

While R-CNN laid the foundation for object detection, its computational inefficiency arises from independently processing each of the ~2000 proposed region. These limitations prompted advancements towards a more streamlined architecture, Fast R-CNN [16].

Fast R-CNN introduced the RoI (Region of Interest) pooling layer. Instead of independently processing each of the region proposal, Fast R-CNN performs feature extraction on the entire image and then uses RoI pooling to extract features specific to each region proposal. This significantly improves efficiency by eliminating redundant computations.
Still, both architectures use selective search to find out region proposals. The selective search algorithm being a time-consuming fixed algorithm, no learning is happening at that stage, resulting in bad candidate region proposals.

To make up for this shortfall, Faster R-CNN [15] takes a step further by integrating a RPN directly into the network instead of selective search. The RPN is a convolution network that functions as a cognitive "attention" mechanism for our Faster R-CNN module, telling it "where to look" within the image. It generates a set of rectangular object proposals as a part of the overall architecture (each with an “objectness” score, a membership measure to a set of object classes vs. background). The ROI cropping

is done on the extracted feature map streamlining the entire object detection process. This innovation improves both speed and accuracy, making Faster R-CNN one of the most reliable SOTA models.

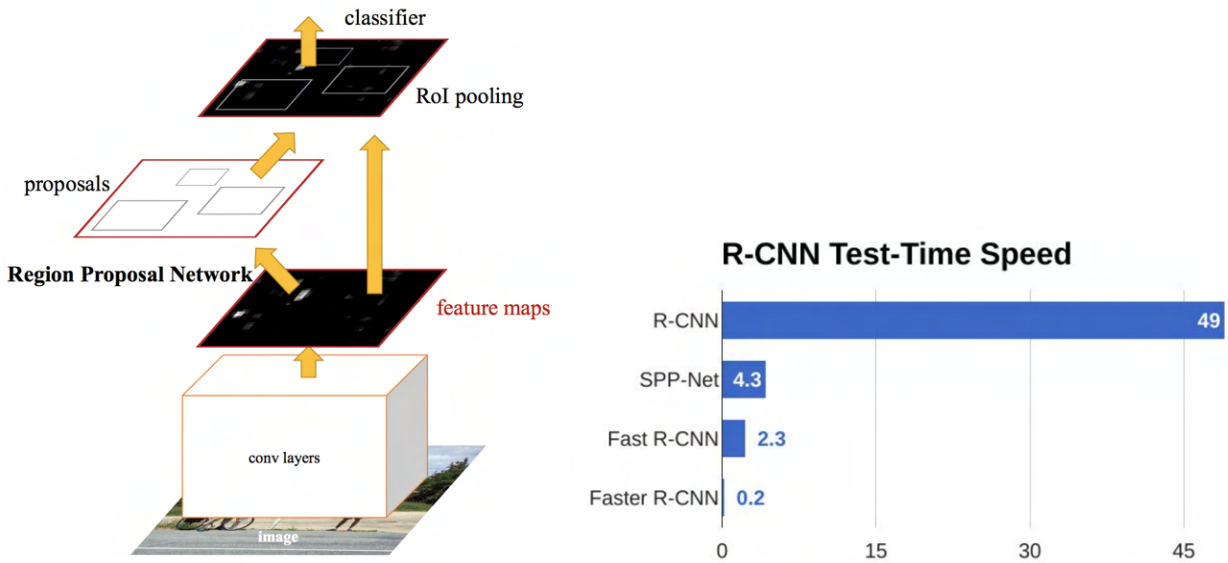

Figure 2.3.8- Structure of Faster-RCNN & Comparison of test time speed for R-CNN object detectors. The RPN module serves as the 'attention' of this unified network [15] (16)

2.3.4.2 FPNs for image processing

Feature Pyramid Networks (FPNs) tackle challenges related to varying object sizes within an image. CNN operates on a hierarchical structure where the resolution of the feature map decreases with each layer while the semantic richness captured by deeper layers intensifies. The trade-off is that semantically robust features become spatially coarser due to downsampling.
FPNs addresses this problem by employing an architecture consisting of two pathways: a normal feed-forward CNN (bottom-up pathway) and a top-down design with lateral connections, enabling the creation of a feature pyramid that merges features from different layers and thus enhances the network's ability to detect objects at both fine and coarse levels.

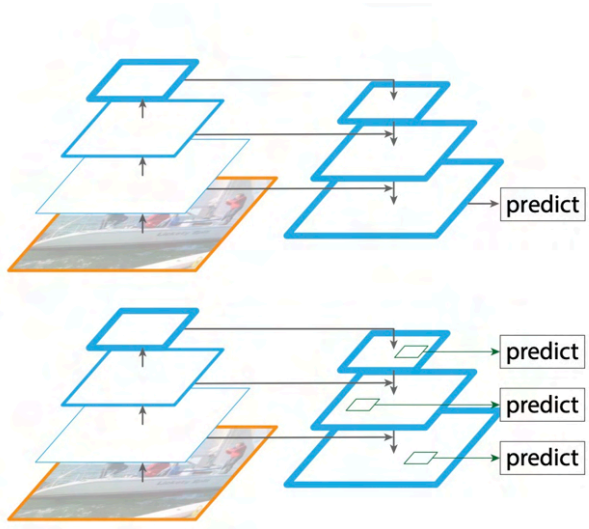

Figure 2.3.9 - Two FPNs architectures. On top predictions are made only on the finest level while on the bottom predictions are made at all levels (17)

FPNs also have the advantage of being extremely versatile in their possible applications. They find applications in improving both RPN and Fast R-CNN / Faster R-CNN. For RPN, FPN replaces the single-scale feature map with a feature pyramid, improving speed and accuracy. For Fast R-CNN and

Faster R-CN, FPN enables the assignment of RoIs of different scales to pyramid levels, enhancing the adaptability of the region-based object detector.

2.3.4.3 SOTA's approaches for object detections

Building upon the principles elucidated by the fundamental DNNs for image processing, contemporary approaches push the boundaries of performance. In particular, we'll be looking at the Detectron2 framework and the YOLOv8 (You only look once) object detection model.

Building upon the success of its predecessor, Detectron2 is a state-of-the-art object detection system developed by Facebook AI Research (FAIR). Detectron2 is built on a modular and flexible platform, allowing easy experimentation and customization. It supports various computer vision tasks including object detection. In our case, the most interesting aspect of detectron2 is its large zoo of SOTA models for object detection, including Cascade R-CNNs [17] and Faster R-CNNs.
In this thesis, we will use the "R101-FPN" model provided by Detectron2. This model is an instance of the two-steps Faster R-CNN architecture with a ResNet-101 backbone and a FPN serving as RPN. It is therefore a combination of ResNet-101 and previous concepts we have discussed in this work.
ResNet-101 is a CNN architecture that belongs to the ResNet (Residual Network) family. "101" stands for the number of layers. It was introduced to address the challenge of training very deep neural networks, its key innovation being the use of residual learning blocks, which introduce shortcut connections or skip connections.

While Faster RCNNs work by detecting possible regions of interest using a RPN, YOLO represents a paradigm shift in object detection by adopting a single-step approach by using only one neural network to predict both bounding boxes and class probabilities. YOLO's name, "You only look once", comes from the fact that, unlike methods using an RPN, it traverses an image using a single iteration and thus, achieves impressive real-time object detection speeds.
YOLO has seen many improvements in its architecture since its first release in 2015. However, its core operating idea remains the same in each version. YOLO works by dividing an image into a grid for which each cell predicts bounding boxes and class probabilities simultaneously.

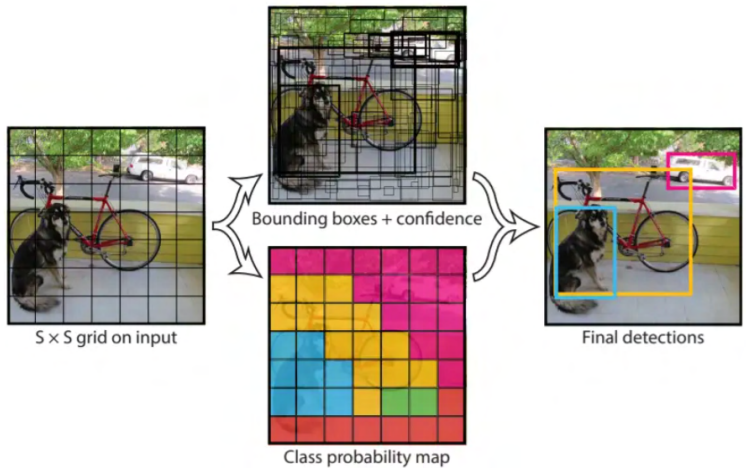

Figure 2.3.10 - Philosophy of YOLOv1 to tackle the single-step object detection (18)

YOLOv8, the latest release of YOLO, makes use of a few key components to perform object detection: A backbone composed of convolutional layers that extract relevant features from the input image and a novel decoupled head responsible for producing the final output of the model in the form of bounding boxes and object classes from the features maps output by the backbone.

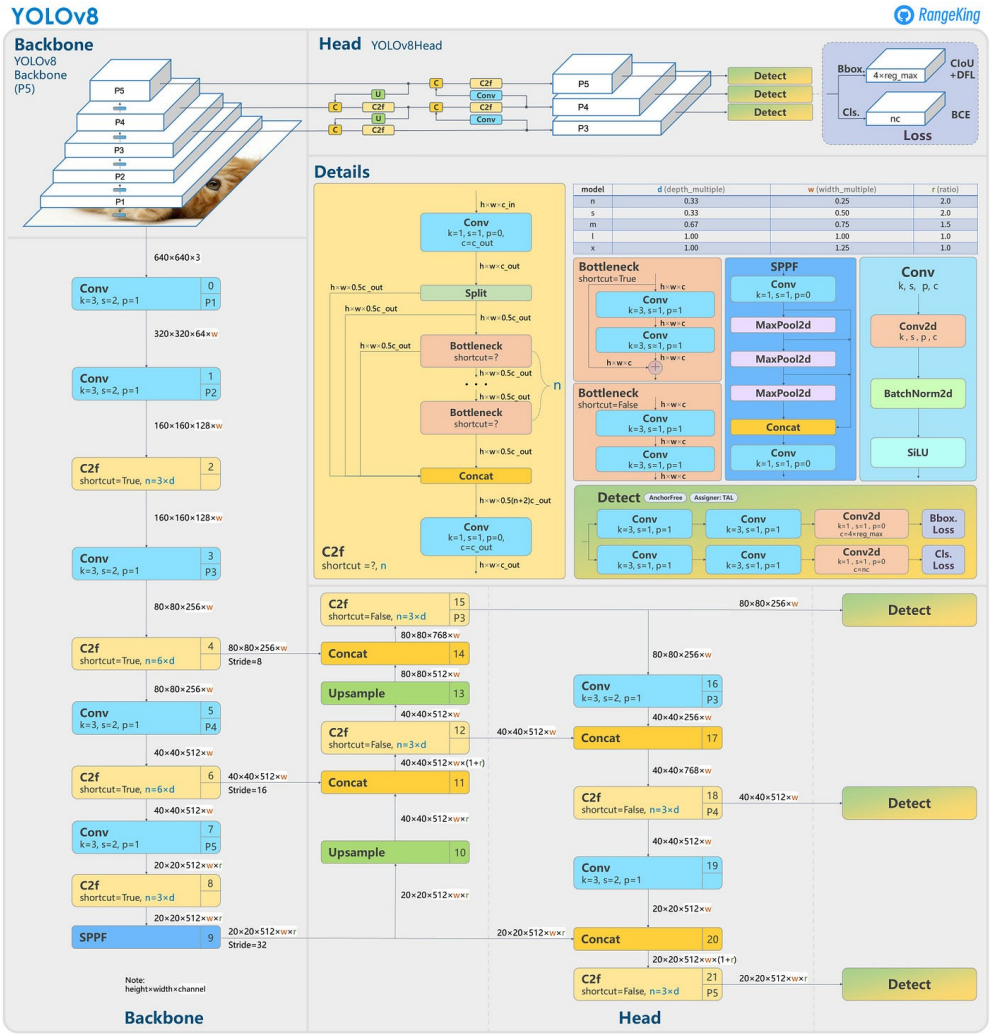

Figure 2.3.11 - YOLOv8 architecture (19)

More specifically, the "SPPF" layer processes features at different scales, and "Upsample" layers increase feature map resolution. The "C2f" module combines high-level features with context for improved accuracy. The detection module uses convolution and linear layers for mapping features to output bounding boxes and classes.

In YOLOv8, the head is decoupled, handling objectness, classification, and regression tasks independently. This design enhances accuracy by allowing each branch to focus on its specific task. The head, optimized for speed and accuracy, employs convolutional layers followed by a linear layer to predict bounding boxes and class probabilities, with attention to channel count and kernel sizes.

The overall architecture of YOLOv8 is designed to be fast and efficient, while still achieving high detection accuracy.

3 Methods

To efficiently detect anomalies in a crowd, our approach involves addressing several critical aspects:

1. **Event Detection Algorithms:** We require algorithms capable of extracting contextual information, particularly from people's movements in video data.
2. **Real-time Positional Knowledge:** The accurate tracking of each detected person's position throughout the video is crucial for effective anomaly detection.
3. **Person Detection:** Identifying and detecting individuals present in the video is a fundamental prerequisite.

In practice, these requirements lead us to organize our work into a three-tiered pyramid structure, with each level dedicated to a specific computer vision task: anomaly detection, multiple object tracking, and multiple object detection.
This pyramid architecture signifies that the overall performance of our solution heavily depends on the efficiency of the lower modules. Efficient people detection is essential for achieving high-performance tracking, and robust tracking performance is imperative for effective anomaly detection.

This section will detail the work carried out on each module, as well as the implementation of a demonstrative end-to-end application. These methods are based on the theoretical concepts introduced earlier.

3.1 Framework and tools overview

During this project, our solution will be developed mainly with Python, an interpreted, multi-paradigm and object-oriented programming language. Python is very versatile due to its wide range of open-source libraries that meet all the needs encountered when building a DL pipeline.

Library	Usage
pytorch	Deel learning framework
torchvision	Computer vision framework
OpenCV – cv2	Computer vision framework
numpy	Arrays and matrix operations
ultralytics	Object detection
streamlit	Creation of interactive data applications

Table 3.1.1 - Main Python libraries used for this thesis

We used external resources such as the VSCode code editor, Amazon S3 buckets for storing datasets and models and gitlab, a git-based DevOps software that can develop, secure, and operate software. We

also used collaborative work software, Confluence as a web-based corporate wiki and Jira for project-specific ticket management. We also benefited from access to a remote server with a 32GB Nvidia GPU, enabling us to train models rapidly.

3.2 Multi-object Detection

Our primary goal in the MOD module is to achieve real-time detection of people in videos, a crucial module influencing subsequent module performance. Focusing on a single class (class 0, “person” or "human"), we address a regression problem, emphasizing bounding box coordinate detection without classification. Our objectives include real-time operation, handling occlusion challenges in crowded scenes, ensuring dataset diversity, model versatility across various video sources, and benchmarking two state-of-the-art methods: One-step for speed and Two-steps for reliability.

3.2.1 Datasets

The Common Objects in Context (COCO) [18] dataset is a widely used large-scale dataset for object detection, segmentation, and captioning. It contains over 200,000 images with annotations for 80 object categories (including the "person" class we're interested in here), making it valuable for training and evaluating computer vision models.

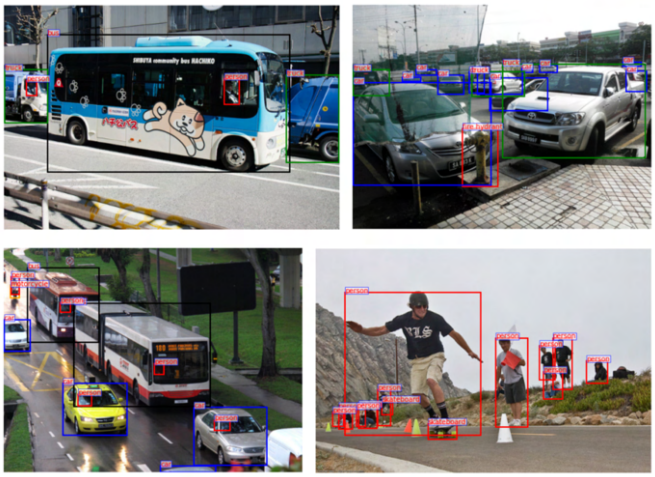

Figure 3.2.1 - Annotated samples on COCO dataset

YOLOv8 provides a model zoo pretrained on COCO. We ran a pretrained model inference to test its performance on crowd images.

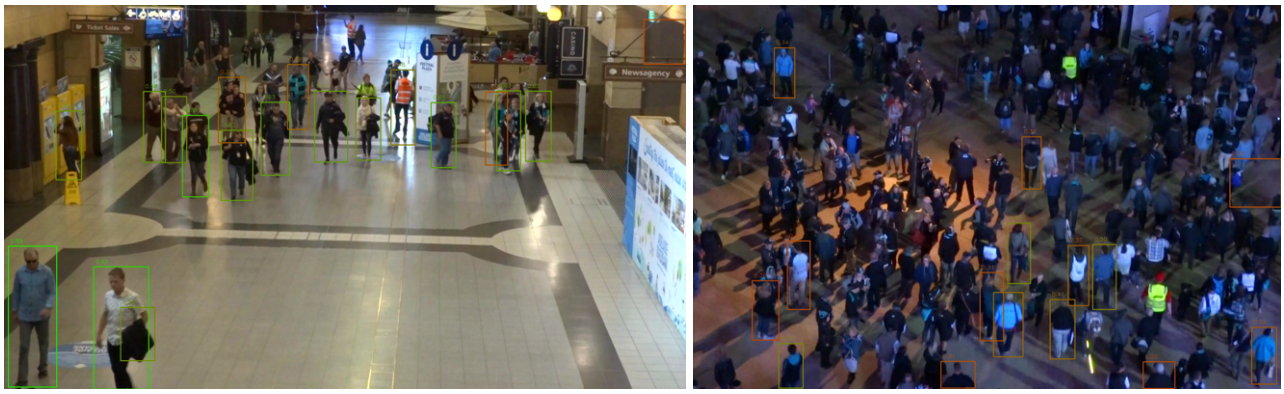

Figure 3.2.2 - Inference on pretrained YOLOv8n model on the COCO dataset. Confidence scores are entered at the top of the bounding box, and the color varies according to the score (red for low confidence, green for high confidence).

Despite good detection of certain targets, the performance of the pre-trained model is unsatisfactory due to the different characteristics between the "person" objects annotated in COCO and the humans present in crowd images, which are much smaller, represented in poorer image quality and more often occluded. This confirmed our need to build up a dataset dedicated to the specific detection of people in crowd images. We want a versatile, fast and high-performance detector. To achieve this, we need to build up a large dataset, with a variety of image formats and annotated people (different heights, inclusion of all ethnic groups, etc.). Our first task was therefore to search for datasets that could be used for our task, or to create our own by manual annotation if necessary.
This benchmark revealed that few datasets dedicated specifically to the detection of humans in crowd scenes exist. We specifically targeted 4 datasets which, when combined, fulfilled all the requirements we had set ourselves for this task: CrowdHuman [19], CityPersons [20], SmartCity [21], WiderPersons [22].

Dataset	Number of Training Images	Number of Validation Images	Average Human Density	Description
CrowdHuman	15,000	4,370	23	Crowd scenes of varying densities
CityPersons	2,975	500	7	Pedestrians recorded in street scenes from 50 cities
SmartCity	50	0	7	CCTV-images captured in 10 different cities
WiderPersons	8,000	1,000	31	Images from a wide range of scenarios, with various kind of occlusion

Table 3.2.1 - Datasets used to detect people in a crowd

Later, we will implement a fall detection algorithm in the anomaly detection module. It will then be shown that our dataset, comprising mainly standing humans, suffers from too little diversity concerning the poses of annotated humans, which will impair the performance of our detector when a person falls. To alleviate this problem, we further increase the size of our dataset with additional third-party datasets dedicated to the detection of humans in reclining, sitting or acting positions. These datasets are Newfall [23] and ppl [24].

Dataset	Number of Training Images	Number of Validation Images	Average Human Density	Description
Newfall	91	0	1	Images of people lying or falling
ppl	658	288	3	Images of people in action in less conventional positions

Table 3.2.2 - Third-party datasets used to help the detetions of unusual positions

We needed to define a standard format, common to all the images and annotations in our dataset. We chose the YOLO data format and we implemented data processing methods to transform our data. The YOLO format consists of one *.txt file per image within our dataset. If there are no objects in an image, no *.txt file is necessary. The *.txt file should be formatted with one line per object in the class x_center y_center width height format. The bounding box coordinates should be in normalized xywh (from 0 to 1). The data is organized in two datasets, train and val.

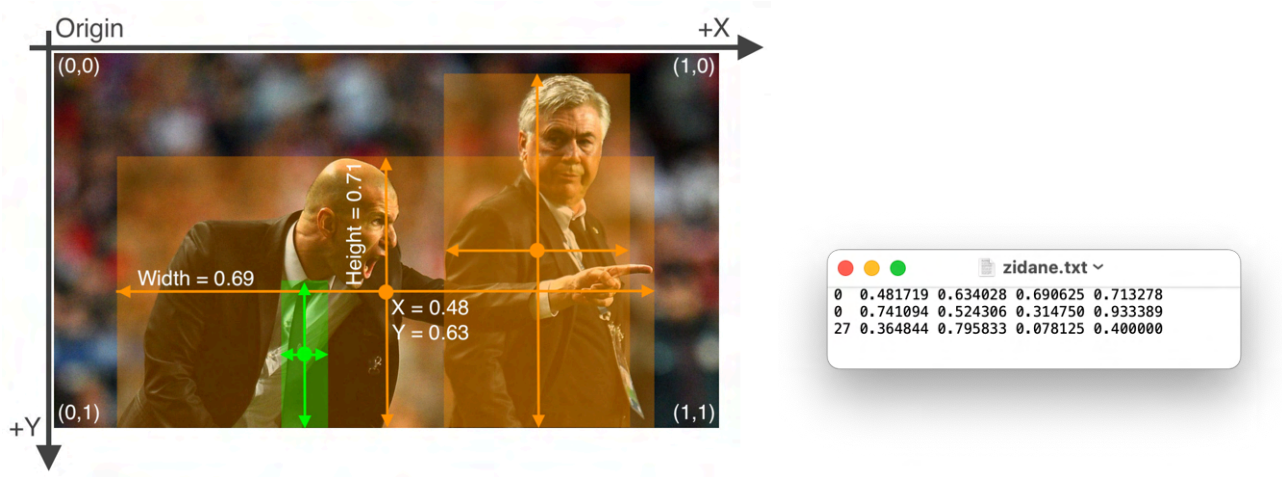

Figure 3.2.3 - YOLOv8 annotation format (20)

Given the final size of our dataset, there was a high probability that a few annotation or image format errors would have crept in. Although manual verification was not an option due to the amount of time allocated to this project and the size of the dataset, we nevertheless carried out a data cleaning phase, implementing methods to verify correct formatting. In particular, we focused on three anomalies: corrupted images, duplicate annotated boxes and coordinates in YOLO, non-normalized or out-of-bounds format.
After cleaning, we have 26239 images and ~605,000 occurrences in our training dataset, 5897 images for ~135,000 occurrences in our validation dataset.

Figure 3.2.4 - Sample images from our training dataset

3.2.2 Object detection models

Once we've built up our dataset, we'll move on to the models to be trained. For reliability, we'll be training two different One-Step models, renowned for their speed, and a Two-Steps model, renowned for its reliability and adaptability. We'll use the YOLOv8 model pre-trained on COCO as a baseline.

The Two-Steps model we have chosen is the R101-FPN model of Detectron2's zoo model. It is a variant of the Faster R-CNN architecture incorporating a FPN backbone for hierarchical feature extraction from input images with a bottom-up ResNet network, composed of four stages, each housing bottleneck blocks. This backbone is further enriched with lateral connections and a top-down pathway, resulting in the creation of feature pyramids. The architecture also encompasses a RPN employing convolutional layers for objectness classification and anchor regression.

Under Detectron2, we don't have epochs, we are dealing with iterations. We start training our model on our GPU over 25,000 iterations, saving the state of our model every 1,000 epochs. We use a batch_size of 8 and a learning rate of 0.00025. We record the values of the various losses of our model on each dataset, which we'll use to select the optimal model.

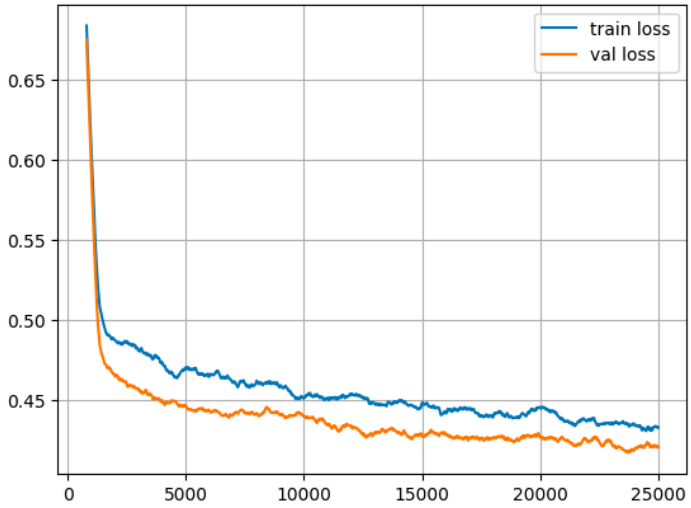

Figure 3.2.5 - Total values of val and train loss during the training of the Faster RCNN

Still under Detectron2 and with the same parameters, we train a One-stage model named RetinatNet [14]. Its architecture is also based on an FPN backbone placed on top of a feedforward ResNet Architecture. To this Backbone RetinaNet attaches two sub-networks, one dedicated to the classification of anchor boxes and the other to the regression of these boxes to the ground truth object box.

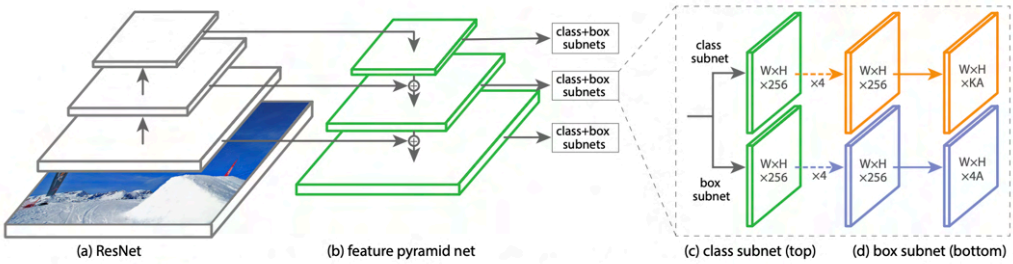

Figure 3.2.6 - Architecture of RetinaNet (21)

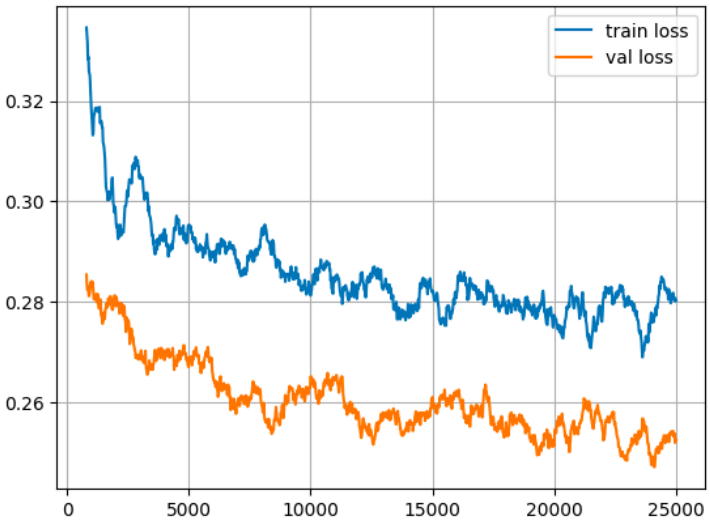

Figure 3.2.7 - Total values of val and train loss during the training of the RetinaNet

Finally, our last trained model will be an instance of YOLOv8, "YOLOv8n", with 3.2M parameters. This is the lightest and fastest architecture of YOLOv8, but the least robust.
We train this model in three modes:

- The model pre-trained on COCO, which will serve as the baseline for our evaluation benchmark
- The model trained on crowd datasets without including newfall and ppl datasets
- Finally, the model trained on our entire dataset.

The models will be trained over 350 epochs, with batch sizes of 16 and a learning rate of 0.01. An early stopping with a patience of 50 epochs has been implemented and activated during training.

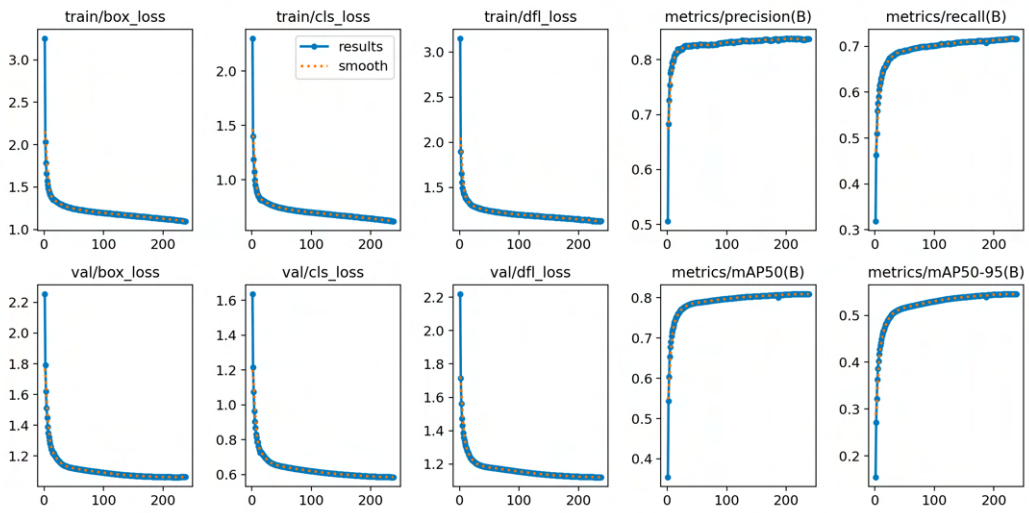

Figure 3.2.8 - Loss and metric values recorded during YOLOv8 training

In summary, for our benchmark, we will train 4 different models through three different architectures and utilize a pre-trained YOLO model.

Detection Model	**Number of Parameters**	**Category**	**Description**
Faster RCNN: *R101-FPN*	61 M	Two-Stage	Utilizes a region-based network with shared features for faster and more accurate object detection
RetinaNet: *R_101_FPN*	57 M	One-Stage	Introduces a focal mechanism to balance object detection across different scales using a convolutional neural network
Yolov8: YOLOv8m	26 M	One-Stage	Offers a real-time object detection approach by performing detection in a single network pass, optimizing computational efficiency.

Table 3.2.3 - Summary of the approaches tested

3.3 Multi-object Tracking

In our study, we will employ a detection-based tracking method. In this approach, objects are identified in each frame of the video sequence and then followed in subsequent frames by matching corresponding object detections. This generally allows for the conceptualization of algorithms that strike a good balance between reliability of outcomes and computation speed, appearing to be the most suitable representation to address our problem. There are already algorithms capable of tracking based on detections; hence, our initial task will involve identifying high-performing detection-based tracking algorithms. We will select a few for further evaluation before investigating potential avenues for improvement.

We have already conducted a benchmark evaluation of multi-object detection models for our application. We will reuse these models, which are trained to be easily generalized, fast, and, ultimately, effective/reliable.

We will ensure our system can operate in real-time on a 30 fps video (i.e., it can perform detections and tracking, excluding video encoding/decoding, in less than 0.03 seconds per frame), efficiently track targets with minimal losses, and, most importantly, be versatile in functioning across videos from various sources regardless of their context (e.g., CCTV, demonstrations, stadiums) or condition (e.g., lighting, image quality, image format).

In conclusion, we will discuss the potential for our system to integrate other tracking algorithms based on different methodologies.

3.4 Anomaly Detection

In this section, we will introduce implementations of algorithms dedicated to anomaly detection in crowds. To operate, these algorithms will use video data of human crowds and tracking data provided by our SparseTrack algorithm. These algorithms exhibit varying levels of complexity and are not directly evaluable due to the absence of dedicated Open-Source Datasets. We could focus on detecting a single anomaly and manually annotate a dataset for the task, but this would require more time and falls outside the scope of our project. The idea here is instead to present outlines of simple algorithms that can serve as baselines for more in-depth work.
These algorithms will focus on detecting 4 types of anomalies: Exceeded Capacity, Exceeded Presence Time, Abnormal Location, and Fall Detection.
Our implementations will need to operate online but may require a period of video prior to their proper functioning. We will detail each of these algorithms, their objectives and their implementations.

3.4.1 Exceeded Capacity

3.4.1.1 Objective

With this anomaly, we aim to trigger an alert when the camera detects an excessive number of object occurrences within a defined area. Various levels of complexity are possible.
Here, we define 5 levels:

1. Counting the number of people per frame and comparing it to a predefined threshold.
2. Counting the number of people over segments of x seconds.
3. Automatically defining the threshold based on average attendances for the given time period, followed by counting the number of people per frame.

4. Filtering objects included in the crowd count (by redefining the tracked classes and implementing a presence authorization policy). For example, medical staff should not be included in the crowd count in a hospital waiting room.
5. Implementing a count with moving camera or multi-camera systems.

We will focus on implementing an algorithm that accommodates complexity levels 1 to 3.

3.4.1.2 Implementation

To implement an algorithm capable of alerting to this anomaly, we will exclusively use the tracking data from our system. Realizing the first level of complexity is relatively straightforward. We only need to define a fixed threshold beyond which a capacity alert will be triggered, then iterate through each frame one by one, counting the number of present identities and comparing this count to our threshold.
We then store each of the frame numbers during which the alert was triggered, as well as the relevant identities.

However, in its current state, our algorithm is highly susceptible to both false positives and false negatives. Furthermore, there are few application cases where we require frame-accurate precision (equivalent to 0.03 seconds for a video at 30 fps).
We enhance our algorithm by implementing a "sliding window" over which we calculate the average number of identities detected across the frames and then check for alert triggers. The window's length should be defined according to our specific needs and the desired level of precision. In a case where we want our system to have a reaction time of less than 1 second when using a 30-fps video, we would use a window size of fewer than 30 frames.
It should be noted that, from now on, our algorithm will not function in the first video second. Depending on the use case, sometimes we may not want to check for capacity excess against a fixed threshold but instead check for a sudden influx change. This information can be obtained through the third version of our algorithm.

For the third level of complexity, we need to define the concept of a "cycle." In our scenario, a cycle will determine a complete iteration of the algorithm. A cycle will be divided according to a unit of time (i.e., a number of frames). For example, in the case of tracking applied to the surveillance of a public space, the cycle of our algorithm will be one day, which we will choose to divide into 24 hours.
Our algorithm will operate in several phases: the "Collection" phase, where it will not have the capability to trigger an alert, and then the "Calculation," "Alert," and "Update" phases, constituting the inference process.

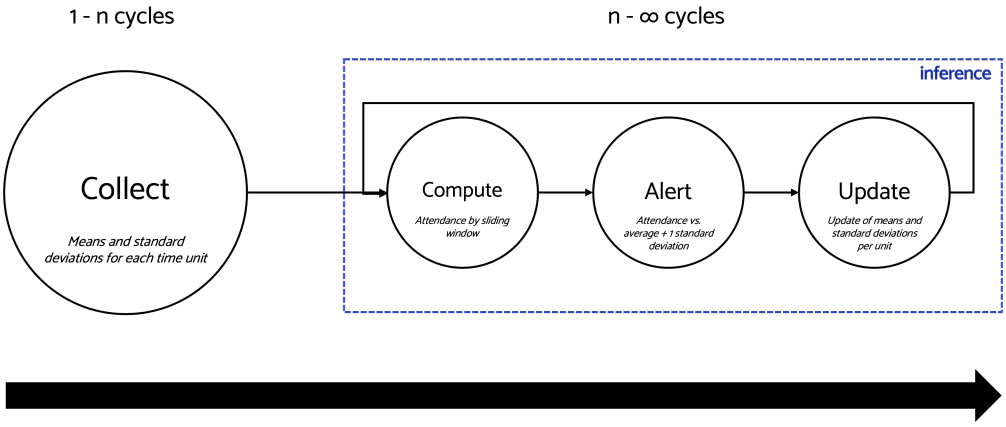

Figure 3.4.1 - Exceed Capacity Algorithm Process

- During the initial cycles, our algorithm calculates the average attendance and their standard deviations for each time unit without the ability to trigger alerts. This is the "Collection" phase, or the "data training" phase. The longer this phase, the more our algorithm will operate with tracking data that are representative of different attendance situations, but the later it will be ready to trigger alerts.
- Then, from cycle n where we consider our algorithm to be sufficiently "trained," we can proceed with inference. The first step is to calculate the attendance over the last x frames (according to the sliding window method defined previously).
- This value is then compared to the historical average value obtained for the time unit. If the average attendance obtained over the last x frames exceeds the sum of the average over the time unit plus a standard deviation, we are then 68% certain that there is an abnormal attendance, and we trigger an alert. Note that this confidence interval can be adjusted according to specific needs.
- Finally, at the end of the cycle, the algorithm updates its database by incorporating the average attendance values obtained for each time unit of the cycle into its history and updates the standard deviations.

This version of the algorithm aims to be as generalizable as possible and would require modifications for optimal use in a specific use case. For example, it would be prudent to employ more advanced techniques for modeling the distribution of tracking data.
For optimal detection of exceeded capacity, we aim to achieve tracking that minimizes the number of false positives and false negatives.

3.4.2 Exceeded Presence Time

3.4.2.1 Objective

This anomaly aims to ensure that the duration of an identity's presence does not exceed a certain limit. Once again, the algorithm will operate solely based on tracking data. It seems feasible to define an algorithm that can automatically detect the abnormal presence time of an identity within an area, based on statistical concepts of data modeling. However, such an algorithm seems less practical than a simpler one that operates with a maximum duration threshold, which we present here.

3.4.2.2 Implementation

Our algorithm requires the definition of two thresholds: a maximum allowed presence duration threshold and an identity "forgetting" duration threshold. Its operation is relatively straightforward: it first scans the detections obtained from our tracking and then determines if the identity associated with the detection exists within its database. If the algorithm does not recognize the identity, it stores the index of the frame where the identity was first detected. Otherwise, our algorithm performs a new test by calculating the age of the identity (by subtracting the index of the current frame from the index of the frame where the identity was first detected, given the known number of frames per second). It then compares this duration to the threshold and triggers an alert, or not, accordingly.

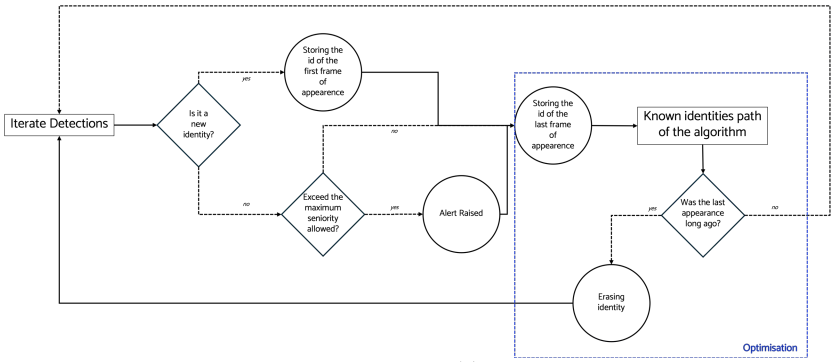

Figure 3.4.2 - Exceeded Presence Time Algorithm Process

We have also implemented an optimization module in our algorithm. The algorithm also stores the frame numbers of the last sightings of each identity. With each new frame, it scans its known identities and forgets those that are too old. This optimizes the computing time of our algorithm and its memory consumption.

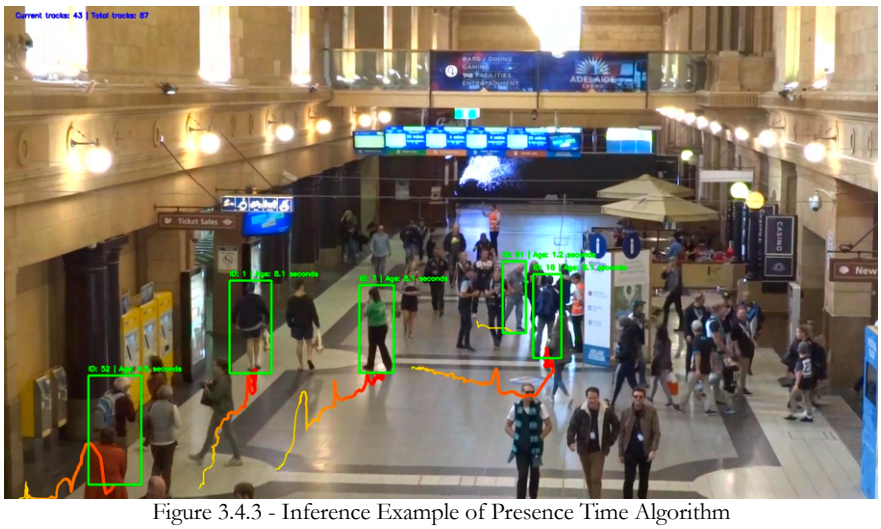

Figure 3.4.3 - Inference Example of Presence Time Algorithm

For optimal operation, this algorithm must rely on tracking that, above all, minimizes the number of identity switches.

3.4.3 Abnormal Location

3.4.3.1 Objective

Sometimes, a pattern of object trajectories can be observed within an area. The goal of the algorithm implemented here is to trigger an alert when an object follows an unusual trajectory and ends up in a forbidden or restricted access area.

3.4.3.2 Implementation

The implementation of this algorithm bears some similarities to that of our excess capacity algorithm. Our approach will be to divide the input image into a grid of subspaces and then determine which cell of this grid represents an unusual location. Once again, we will limit ourselves to tracking data, although potential improvements to this algorithm could be made possible through the use of video data (e.g., edge detection). However, we will use frame size data.

Like our excess capacity detection algorithm, the algorithm implemented here requires certain training cycles to collect data representative of the "normal" locations of our objects. The behavior of our algorithm during this training phase is as follows:

1. We start by defining the size of our position grid: the size of each cell and accordingly, the number of vertical and horizontal cells making up the grid. There are several philosophies here, either defining the number based on size or defining the size based on the number.
2. We then scan the detections of each frame and update our position grid.
3. From this, we update a second grid or "heatmap" containing the normalized position data of our objects.

Figure 3.4.4 - Inference Example of Position Heatmaps Computation

After a certain number of frames, we can consider that our algorithm is well trained, and we can then launch the inference phase. This phase adds a unique step of verifying the normality of a location before the update of our position grid. If the grid corresponding to the object's location has a percentage of positions lower than a threshold, then an alert will be triggered.

To achieve optimal detection of abnormal locations, we must ensure to define tracking that establishes reliable object locations, minimizing false positives above all and maximizing true positives.

Figure 3.4.5- Inference Example of Heatmaps and Presence Time Computation

3.4.4 Fall Detection

3.4.4.1 Objective

The goal of this final algorithm is more concrete than the others; it aims to enable the detection of human falls. We define a "fall" as abrupt movements that vertically bring the lower part of a body closer to its upper part relative to the ground.

3.4.4.2 Implementation

This time, our algorithm will need to exploit video data in addition to tracking data. Our approach to solving the fall detection issue relies on estimating the human skeleton's position to establish conditions indicative of a fall.

There are already reliable and fast open-source human pose detection models available. One such model is the Vision Transformer (VIT-Pose), which shows state-of-the-art performance in the field and is particularly noted for its simplicity, scalability, and flexibility. Due to time constraints and because the "vitpose-l" model from the VIT-Pose model zoo appears to empirically meet our needs, we will not train a specific model for this task. We will use the COCO format, composed of 17 keypoints, to define a pose, tracking the positions of the eyes, ears, nose, shoulders, elbows, hands, hips, knees, and feet.

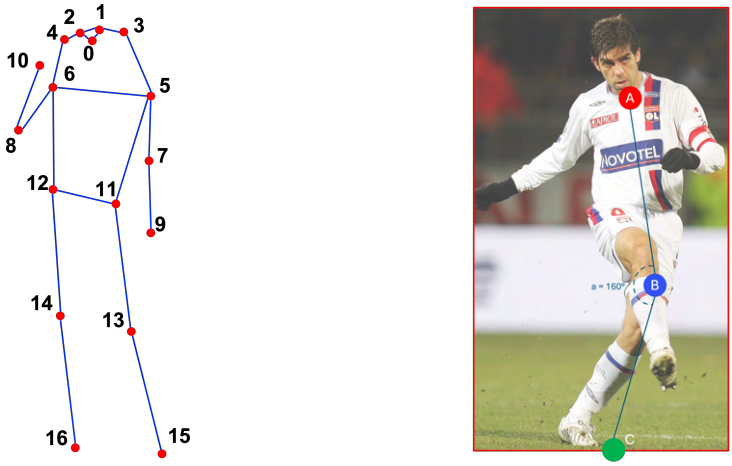

Figure 3.4.6 - Keypoints Definition and attachment points

The process we use to detect falls is as follows:

1. We scan through the frames of our video. Using detections from our tracking, we crop our image around our object of interest.
2. We then apply VIT-Pose to the cropped bounding box images to obtain the skeletons of the people we have identified. Our algorithm retains the coordinates of the keypoints before forgetting them, as done in the optimization module related to presence time.
3. For each detection, our algorithm calculates the attachment points related to the upper body and the lower body from the pose's keypoints. These are two representative points of the average positions of the keypoints linked to the upper body and those linked to the lower body. Let's name A the center of gravity of the upper body and B of the lower body. C will designate the center of the bottom of the bounding box.
4. We then calculate the angle α between vectors AB and BC defined by our attachment points. To ensure a consistent representation of the angle and thus avoid ambiguities related to supplementary or complementary angles between our two vectors, we use the cosine of the angle between vectors AB and BC. By using the cosine of the angle, we obtain a numerical measure of the similarity or alignment between our two vectors. Then, we use the inverse cosine function to obtain the angle in radians, which we convert into degrees for more intuitive interpretation. This process ensures that the calculated angle is within the expected range of values and facilitates the interpretation of the obtained results.
5. We then determine whether the object has fallen based on the obtained angle value. The closer the angle between the two vectors is to zero, the more aligned they are, and the object doesn't seem to be falling; the closer the angle is to 90°, the less aligned the vectors are, thus more representative of a falling position in humans.
6. In cases where center A is "lower" than center B, a fall might not be detected. Thus, we change its status to "positive fall" because we assess the object's position as indicative of a fall.

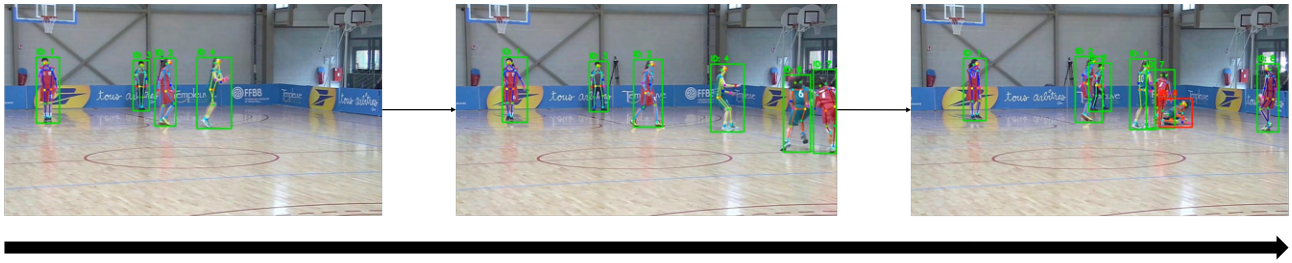

Figure 3.4.7 - Fall Detection of a Sport Video

Note that this algorithm detects a "falling position" rather than the fall itself, and it would be wise to work on a version 2 of this algorithm that considers the speed or even acceleration of the angle between our two vectors rather than its value. Indeed, in our case, a person lying down will be identified as falling, while we will have difficulty precisely detecting the beginning of the fall. To calculate the angular acceleration between AB and BC over time, we would need periodic recordings of these angle values, from which we could derive the angular acceleration by estimating the angular speeds.

3.5 Demonstration Application

As the development of this system is intended as a proof of concept for a future commercial application, we have focused on creating an intuitive, appealing, and interactive demonstrator that enables easy visualization and evaluation of each tool developed in this project. Constructed in a pyramidal structure, our final system operates such that each module functions based on the results of the preceding one. Consequently, we have constructed a comprehensive pipeline that links each module to enable anomaly detection in a crowd. This setup allows for obtaining the interim results from each stage of our final pipeline, ranging from object detection to tracking and anomaly detection.

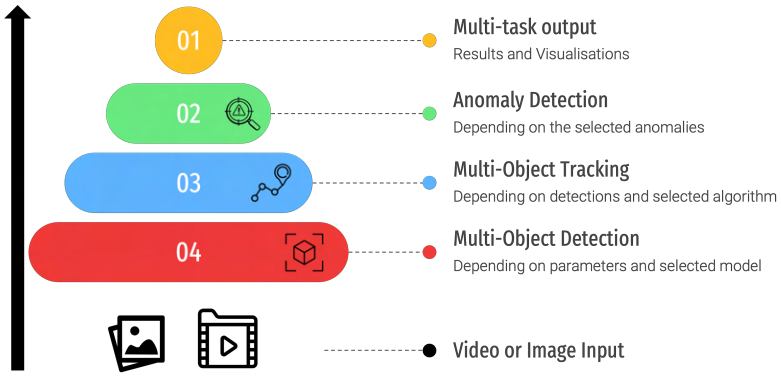

Figure 3.5.1 - Structure of the final pipeline

For development, we utilized Streamlit, an open-source Python library specialized for web application development with a data-centric focus. We incorporated all essential user information as well as numerous setting options to offer users the greatest possible freedom to manipulate the application. In addition, our application supports caching of models, parameters, or data to optimize processing times. Lastly, it provides extensive visualizations accompanied by the outcomes of our system.

4 Results

This section aims to present the various results obtained for each module of this project, whether they are quantifiable or not. The final section showcases the ultimate proof of concept for our project.

4.1 Multi-object Detection

4.1.1 Dataset

As mentioned in the previous section, we iteratively created our dataset. Initially, we targeted pre-existing crowd or human image datasets and then enriched this with third-party datasets to allow for the detection of humans in more specific positions. After a cleaning phase, this dataset comprised 26,239 images and ~605,000 instances of humans in our training dataset, along with 5,897 images for approximately ~135,000 instances in our validation dataset.
Based on this dataset, we obtain the following correlogram:

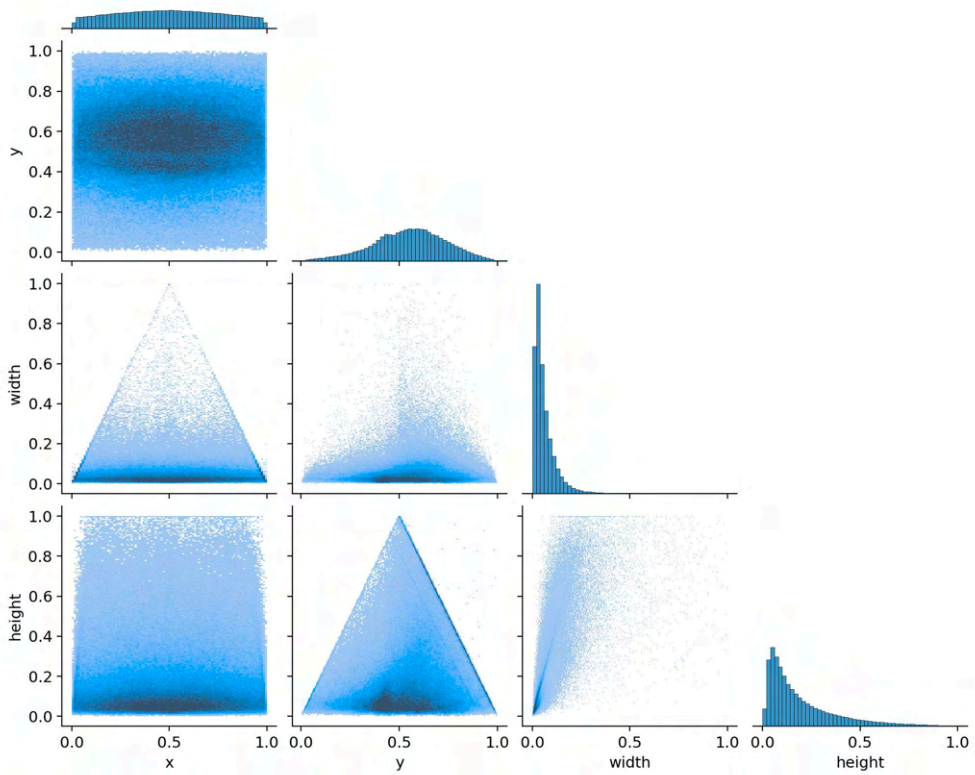

Figure 4.1.1 - Correlogram of the final dataset

The correlogram provides insights into the relationships between the coordinates of our annotations. Among other findings, we observe an evident linear correlation between the width and height of our bounding boxes, which seems coherent given that our dataset consisting primarily of standing people, their detected bounding boxes tend to increase in both dimensions together. Our dataset shows a uniform distribution across the x coordinates of our bounding boxes, whereas our y coordinates appear to follow a normal distribution. Ideally, the latter should also be uniform to avoid bias towards any particular area, but we consider that we have enough data at our disposal for this not to be an issue in our case.

4.1.2 Evaluating Detection Models

Once our detection models are trained, we can evaluate them on our dedicated dataset. For tracking purposes, while minimizing the number of false detections is wise, it's more crucial to have as few false negatives as possible. We are looking for a model that shows good precision and recall metrics, with particular attention to recall (the model's ability to capture all positive examples in the dataset). For each evaluated model, we record metrics including MaP50 (mean average precision at 50% IoU threshold), MaP50-95 (mean average precision averaged over multiple IoU thresholds from 50% to 95%), AP (average precision), AR (average recall), inference time, and we present the confusion matrices calculated with the same detection confidence threshold set at 0.3 and an IoU threshold at 0.5 for target association.

Detection Model	**AP50 ↑**	**AP50-95 ↑**	**AR50 ↑**	**Confusion Matrix**	**Inference Time (ms) ↓**
Faster RCNN	0.725	0.430	0.743	100826 | 38272 34901 | -	50.7
RetinaNet	0.641	0.411	0.758	102842 | 57644 32885 | -	48.3
Yolov8: *Pre-trained on COCO*	0.885	0.423	0.389	52752 | 6840 82975 | -	2.1
Yolov8: *Trained on crowdhuman*[3]	0.837	0.602	0.725	98417 | 19128 37310 | -	4.6
Yolov8: *Trained on ppl*[4]	0.836	0.604	0.729	98956 | 19380 36771 | -	4.8

Table 4.1.1 - Detection models benchmark

It should be noted that the interpretation of the metrics obtained plays a significant role in selecting the best model. For instance, a model that performs well a priori but is not accurate in regressing the coordinates of the bounding boxes may be preferred over a less versatile model that generates too few box proposals. However, an inaccurate prediction that fails to meet the IoU threshold for association with a label will generate both a false positive and a false negative (the label will not be detected either), affecting both the AP and the AR of our model. The annotation of minimally sized objects, such as heavily occluded people, remains extremely sensitive to errors during IoU-based association. Even if we

[3] *Yolov8: Trained on crowdhuman refers to the model trained on our dataset excluding the newfall and ppl data*

[4] *Yolov8: Trained on ppl refers to the model trained on the entirety of the dataset.*

assume the occurrences of these are negligible relative to the size of our dataset, we keep their existence in mind.

Additionally, F1-confidence, precision-recall, precision-confidence, and recall-confidence curves were calculated for each of the evaluated models. These curves provide complementary information on the specific performances of our models.

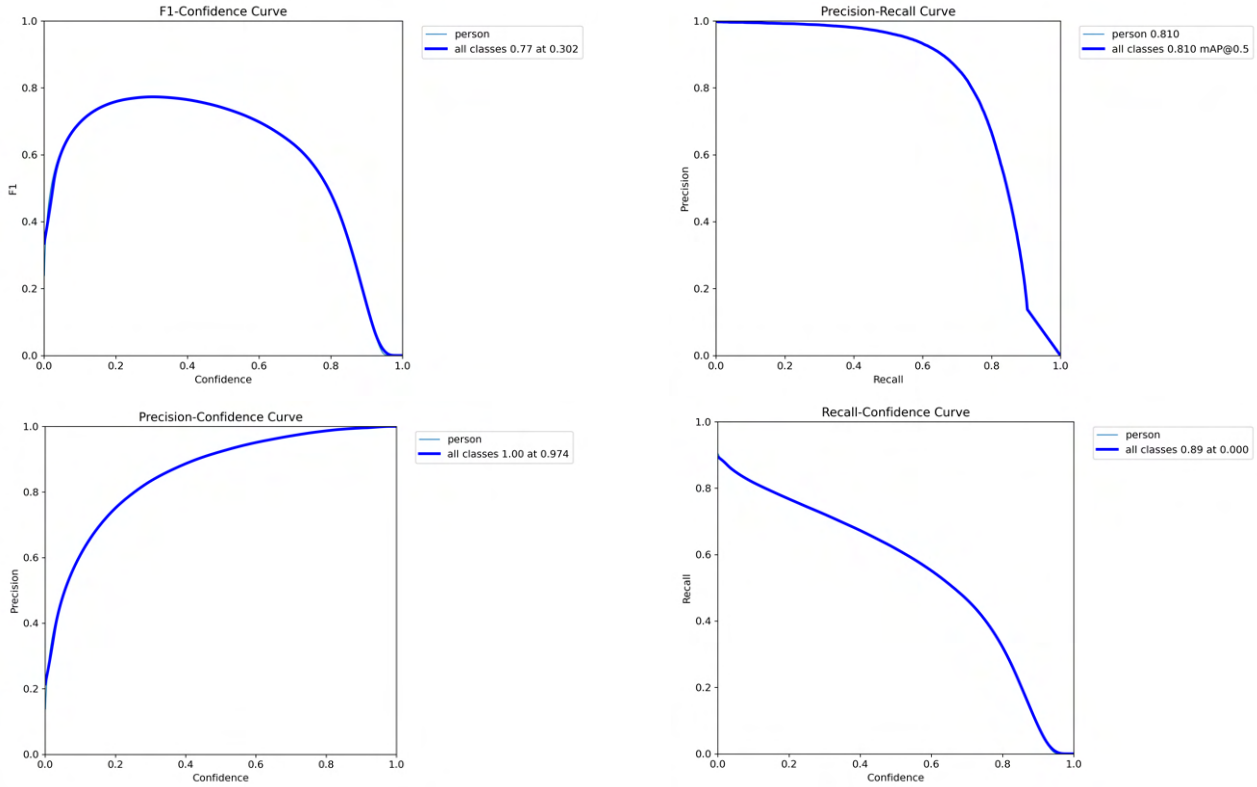

Figure 4.1.2 - Performance curves for our Yolov8 model

The F1-confidence, precision-recall, precision-confidence, and recall-confidence curves offer insightful details about the performance nuances of the evaluated models, highlighting the intricate balance between precision and recall. The precision-recall curve demonstrates how effectively a model can identify only relevant objects (precision) while ensuring all relevant objects are detected (recall), showcasing the precision/recall trade-off directly. As precision increases, recall often decreases, and vice versa, indicating that improving one aspect may lead to compromises in the other. Precision-confidence and recall-confidence curves further illustrate how altering the confidence threshold of detection impacts precision and recall, respectively. A higher confidence threshold may improve precision by reducing false positives but at the risk of increasing false negatives, thereby lowering recall. Conversely, a lower threshold might boost recall at the expense of precision. The F1-confidence curve consolidates this information, providing a singular measure that balances precision and recall, thereby serving as an effective metric to evaluate the overall performance of a model against different confidence thresholds.

4.1.3 Inference Examples

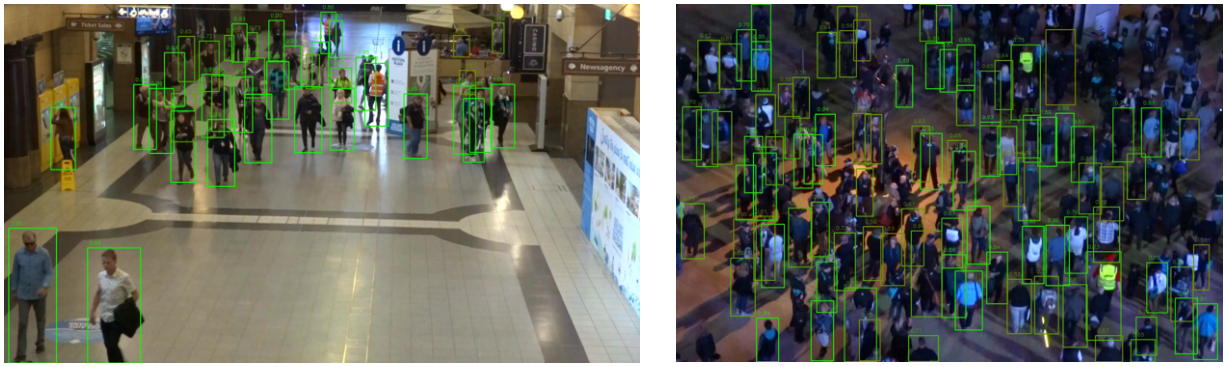

Figure 4.1.3 - Inference examples of Faster RCNN (with a threshold of 0.5)

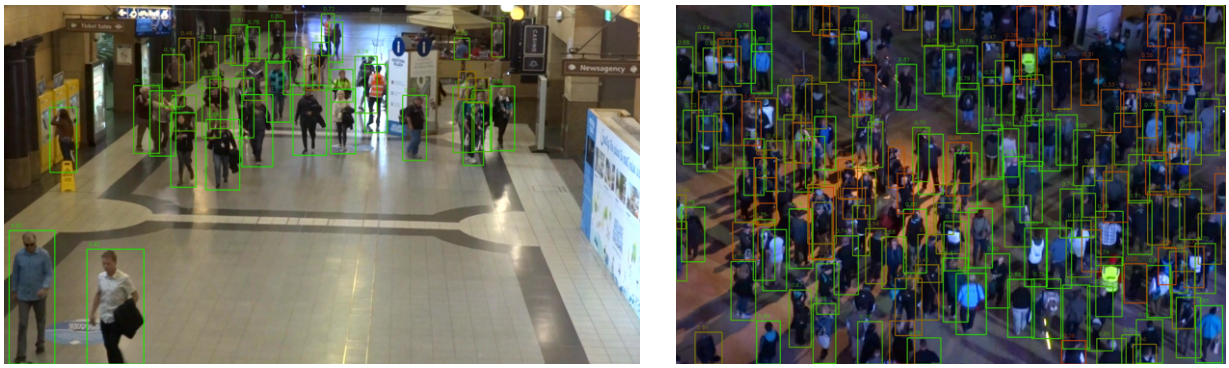

Figure 4.1.4 - Inference examples of YOLO (with a threshold of 0.2)

4.2 Multi-object Tracking

We will evaluate our tracking algorithms using our YOLOv8 model, specifically trained for the task of detecting people in crowds, and a fine-tuned version of this same model on a portion of the MOT20 [26] annotations. Utilizing these two models allows us to compare the performance of our algorithms using a widely generalizable detection model and the performance obtained when using a model more closely tailored to the use case - with a similar image capture environment.
For each model, we will record its inference speed and performance on our tracking benchmark.

4.2.1 Dataset

To evaluate our tracking, we will use 4 videos from the MOT20 dataset for which we have tracking annotations. These videos represent street scenes of varying densities, the table below lists the specific characteristics of each set. We will subsequently observe that these have a significant impact on the performance of our algorithms.

Set	Density	Length (min:s)
MOT20-01	62.1	(00:17)
MOT20-02	72.7	(01:51)
MOT20-03	148.3	(01:36)
MOT20-05	226.6	(02:13)
Combined	**149.7**	**(05:57)**

Table 4.2.1 - MOT20 sets characteristics

4.2.2 Inference Time

Inference time is a critical metric for applications involving real-time tracking. Ensuring that the total time taken to process each frame remains within the confines of the frame rate is crucial for maintaining a seamless tracking experience. For each frame, we measured the inference times of the trackers to perform future position prediction, track association, and their creation/deletion. The values provided here correspond to the total and average inference time per frame, calculated across various videos from the MOT20 dataset at 25 fps.
For an end-to-end tracking application, these inference times need to be considered alongside the time required for object detection as well as for pre-processing and post-processing of images.

Tracking Algorithm	**Set**	**Total Inference Time (s) ↓**	**Mean Inference Time (ms) ↓**
SORT / *fine-tuned version*	MOT20-01	4.32 / *3.01*	10.07 / 7.03
	MOT20-02	31.65 / *21.97*	11.38 / *7.90*
	MOT20-03	61.76 / 42.93	25.68 / *17.85*
	MOT20-05	104.99 / *82.80*	31.67 / *24.98*
	Combined	**202.72 / *150.71***	**22.68 / *16.85***
DeepSORT / *fine-tuned version*	MOT20-01	110.5 / *60.30*	257.55 / *140.63*
	MOT20-02	913.62 / *417.15*	328.40 / 149.95
	MOT20-03	4488.57 / *1935.45*	1866.42 / *804.76*
	MOT20-05	9185.71 / *5472.30*	2771.04 / *1651.70*
	Combined	**14798.4 / *7885.2***	**1645.33 / *822.20***
BYTETrack / *fine-tuned version*	MOT20-01	2.94 / *2.49*	6.86 / *5.81*
	MOT20-02	21.0 / *18.32*	7.55 / *6.58*
	MOT20-03	39.52 / *31.58*	16.43 / *13.13*
	MOT20-05	68.94 / *63.22*	20.8 / *19.07*
	Combined	**132.4 / *115.61***	**14.82 / *12.93***
SparseTrack / *fine-tuned version*	MOT20-01	3.69 / *2.92*	8.60 / *6.81*
	MOT20-02	24.96 / *20.35*	8.97 / *7.32*
	MOT20-03	46.75 / *35.97*	19.44 / *14.96*
	MOT20-05	82.97 / *74.51*	25.03 / 22.48
	Combined	**158.37 / *133.75***	**17.72 / *15.97***

Table 4.2.2 - Trackers inference time benchmark

Let's note that the inference time of our algorithms relies more on the number of detections to consider rather than on the inherent structures of our algorithms, which are quite similar and simple for SORT, BYTETrack, and SparseTrack and are plug-and-play.
Within our exeperiment, the image processing pipeline of DeepSort is not optimized, and it is likely possible to reduce the times obtained by tweaking parameters, especially those related to memory or batch sizes. However, our experience proves that it will still be too lengthy for a real-time tracking application without reducing the fps.

Also, note that for tracking following a detection-association paradigm, processing times are largely correlated with the number of detections generated by our detector. Therefore, while real-time performance is largely feasible for most of our scenarios, those featuring very high density may not be able to operate in an online process with "static" parameters. In such a case, a reduction in video fps or optimization of the parameters of our trackers may be necessary.

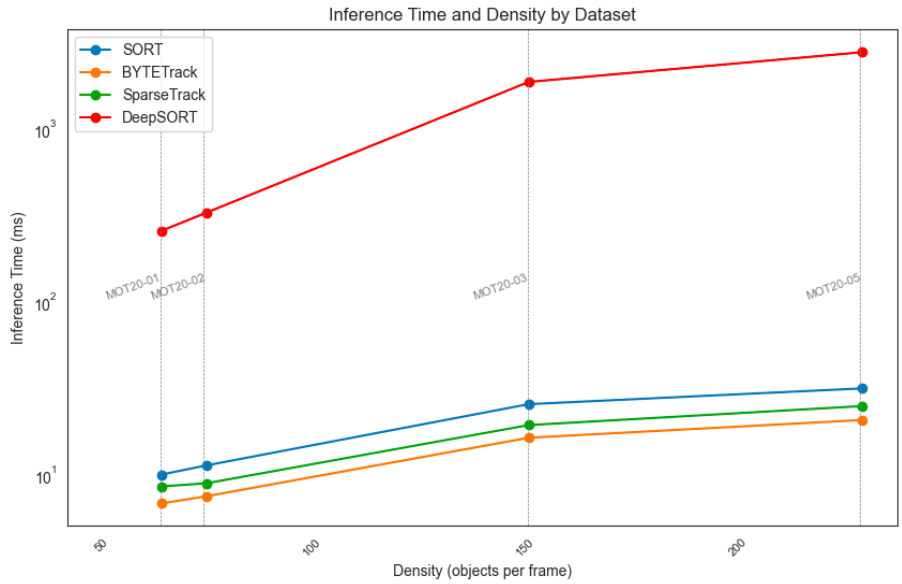

Figure 4.2.1 - Tracker's inference times depending on set density

4.2.3 Selected Metrics

As discussed in 2.2.3, to effectively evaluate tracking, we must select well-chosen and complementary metrics that reveal all the characteristics of our system. We have chosen to base our benchmark on three metrics: HOTA, MOTA, and IDF1 metrics.

Tracking Algorithm	**Set**	**HOTA ↑**	**MOTA ↑**	**IDF1 ↑**
SORT / *fine-tuned version*	MOT20-01	33.92 / 44.70	54.48 / 56.72	45.89 / 60.51
	MOT20-02	27.83 / 39.73	53.52 / 53.56	38.65 / 50.79
	MOT20-03	28.00 / 38.03	56.30 / 49.71	43.64 / 59.99
	MOT20-05	18.81 / 35.60	44.30 / 53.79	30.38 / 51.83
	Combined	**23.65 / 37.13**	**49.05 / 52.68**	**35.89 / 54.12**
DeepSORT / *fine-tuned version*	MOT20-01	44.28 / 53.14	63.30 / 60.44	59.66 / 64.31
	MOT20-02	37.54 / 46.95	66.38 / 56.42	49.44 / 55.35
	MOT20-03	47.24 / 51.32	77.84 / 67.46	74.00 / 68.80
	MOT20-05	35.52 / 47.17	67.32 / 63.44	51.48 / 59.13
	Combined	**39.84 / 48.53**	**70.03 / 63.54**	**58.04 / 61.52**

BYTETrack / *fine-tuned version*	MOT20-01	49.88 / 53.17	75.63 / 73.29	80.80 / 82.28
	MOT20-02	44.59 / 48.07	73.41 / 71.41	70.99 / 72.19
	MOT20-03	50.59 / 58.48	79.13 / 69.84	85.35 / 81.66
	MOT20-05	40.3 / 51.81	70.02 / 76.85	72.05 / 79.99
	Combined	**44.37 / 53.26**	**73.10 / 74.11**	**75.85 / 79.43**
SparseTrack / *fine-tuned version*	MOT20-01	50.86 / 53.25	75.22 / 71.69	84.10 / 84.08
	MOT20-02	44.61 / 48.35	71.64 / 69.68	77.07 / 78.06
	MOT20-03	50.89 / 58.49	78.56 / 68.91	88.62 / 82.09
	MOT20-05	40.03 / 50.44	68.74 / 75.86	78.39 / 84.54
	Combined	**44.36 / 52.53**	**71.96 / 73.02**	**81.24 / 83.02**

Table 4.2.3 - Trackers performance benchmark

We can also take a closer look at the performance metrics that make up the HOTA scores of our trackers, particularly SparseTrack and ByteTrack, which appear to exhibit similar behaviors based on our results in Table 4.2.3 :

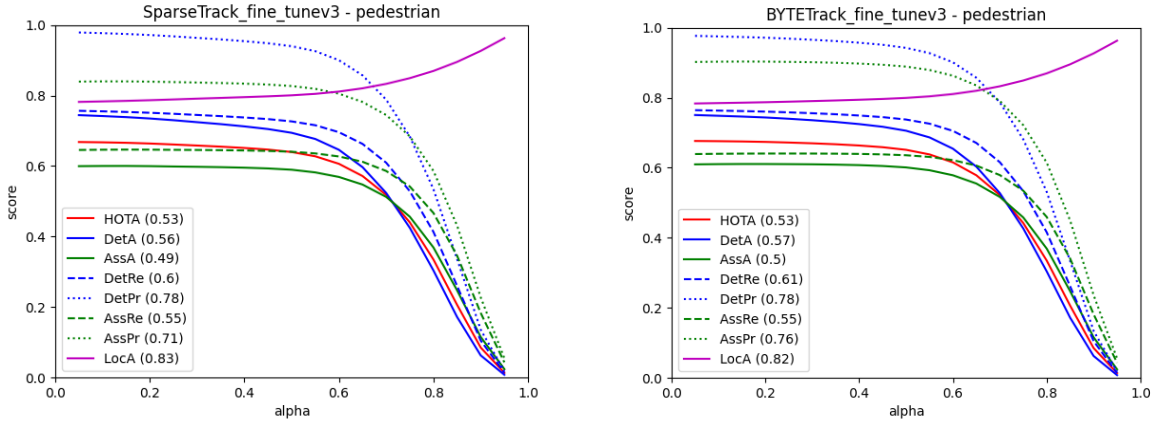

Figure 4.2.2 - HOTA evaluation curves for Sparsetrack and BYTETrack

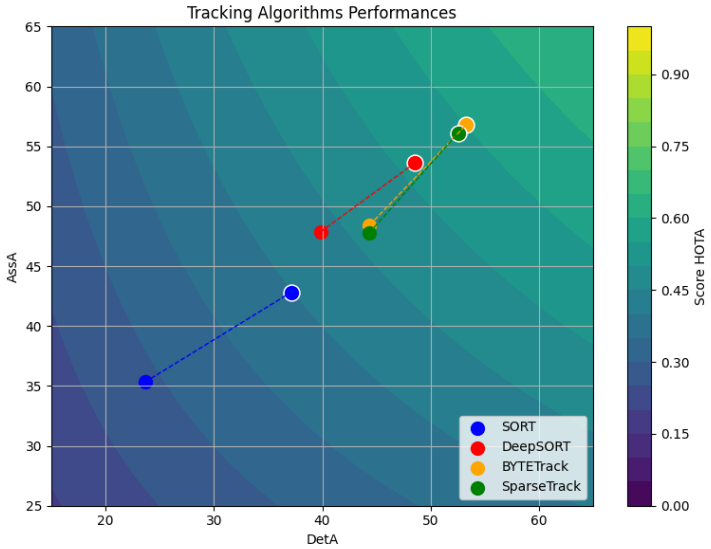

Figure 4.2.3 - HOTA, DetA and AssA values of our trackers

4.2.4 Inference Example

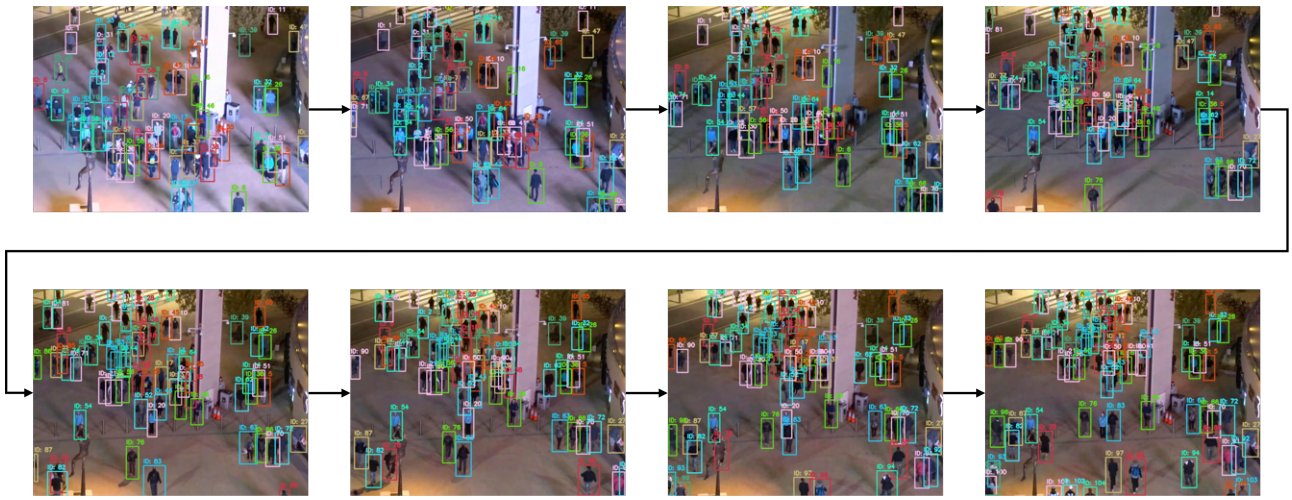

Figure 4.2.4 - Inference example of SparseTrack tracker

4.3 Anomaly Detection

Regarding anomaly detection, we lack specific evaluation metrics and benchmarks. As the research field is relatively recent, it has not yet attracted significant interest from the scientific community for the applications we have studied. Consequently, we have predominantly worked on algorithms based on empirical heuristics. In the absence of dedicated datasets, we are currently able to evaluate our anomaly detection algorithms only visually. Nonetheless, these provide useful baselines for more technical and advanced research on the subject.

Figure 4.3.1- Presence time, abnormal location & Exceeded capacity inference example

Figure 4.3.2 - Fall Detection Example using SparseTrack tracker and Yolov8 detector

5 Discussion

This section offers a thorough exploration of our key findings, providing a detailed analysis of the methods and results we've developed. It sheds light on both the strengths and potential limitations inherent in our outcomes.

5.1 Results Overview

The final outcome of this project is a versatile multitask anomaly detection tool capable of performing multi-object detection, online tracking, and anomaly detection. Additional features include pose detection and the evaluation of the models and algorithms integral to the tool. This proof-of-concept tool demonstrates significant potential for future commercial development. All applications of these technologies must adhere to individual freedoms, fundamental human rights and future laws on AI currently under discussion at the Council of Europe.
We believe our results are promising, yet there is a notable gap between this R&D demonstration tool and a viable commercial application. To bridge this gap, several enhancements are necessary:

- Dataset Diversity: Incorporating more diverse datasets that include a broader range of human poses is crucial. It could also be interesting to incorporate images of people in motion to better manage these specific cases.
- Inference Times: Improving the inference times of the models is essential to create a significant margin in the pipeline processing times.
- Adaptability: Increasing the models' adaptability to handle various human sizes.
- Context-Specific Tracker: Developing a tracker specifically designed for CCTV-like contexts, utilizing constraints related to entry and exit zones for identity creation and deletion.
- Pipeline Optimization: Our current pipeline is not yet optimized for real-world applications. Practical deployment would require developing an optimal pipeline tailored to the selected detection and tracking models.

Overall, we have established a solid initial benchmark for the various models and methods applied to the tasks of human crowd detection and tracking. We have also established a strong baseline for crowd anomaly detection algorithms based on heuristics or pose estimation keypoints. We hope this serves as a foundation for future open-source research projects on the subject.
For optimal performance, retraining the system with real-life data will be necessary.

5.2 Task Specific Discussion

5.2.1 Multi-Object Detection

In this study, we have constituted a large, diverse, and inclusive dataset of crowd images. On this dataset, we have trained 4 models from 3 different architectures using both One-Step and Two-Steps approaches. Subsequently, we evaluated each of these models to which we added a pre-trained model serving as a baseline for our benchmark.
Within the scope of this work, our objective is to develop an optimal detection model for object tracking tasks. Tracking algorithms generally rely on a method of target association across frames which can be computationally costly. Bearing this in mind, we instead prioritize obtaining a detection model that is fast and versatile over one that would be more reliable but slow and less generalizable.

As expected, the results obtained table 5 show that the YOLOv8n-crowdhuman and YOLOv8n-ppl models present closely-aligned results, given the similarities in their training. We will subsequently prioritize the YOLOv8n-ppl model as it is assessed to be more easily generalizable.
Moreover, it seems evident that our best choice is to opt for the YOLOv8-ppl model as it significantly outperforms its competitors in the aspects most crucial to us. The key point advocating for this model is its speed. YOLOv8 processes images 10 times faster than our RetinaNet or Faster R-CNN. Theoretically, it can process up to 205 frames per second compared to just 20 frames for the faster R-CNN, which makes it a favorite for real-time, online applications such as real-time target tracking. Its speed allows us more leeway in setting our tracking algorithms and enables us to focus on defining reliable algorithms rather than swift ones (still within the realm of real-time at 30 fps).
In addition, its other performance metrics remain very satisfactory. Granted, YOLO only ranks third for the recall metric in our benchmark, but it is close to its predecessors, RetinaNet and Faster R-CNN, which YOLO significantly surpasses in terms of precision. When comparing the f1scores50 (harmonic means of the AP50 and AR50 scores) for these three models, YOLO far outpaces Faster R-CNN and RetinaNet with respective scores of 0.779, 0.732, and 0.695. However, we should bear in mind that in a tracking application, false positives have less of an impact on results than false negatives. Thus, it might still be worth including Faster R-CNN in a detection-based tracking system because it exhibits a better recall than YOLO and more balanced AP/AR than RetinaNet.

5.2.2 Multi-Object Tracking

With our chosen set of benchmark metrics, we can determine whether a tracker is competent and in which particular characteristic it excels (in its definition of detections or its association capability). Our goal is to define the most versatile tracker possible that can operate in real time. To pursue this goal, we have previously defined a detector trained on a large-sized dataset with highly diverse images. This pursuit of versatility and our absence of case-specific application led us to prioritize the results obtained on the HOTA metric. Unlike IDF1 and MOTA metrics, which respectively bias measures of association and detection, the HOTA metric represents a balance between performance measures of detection and association of our trackers, and thereby suits our pursuit of a versatile system.

The first analysis we perform on our benchmark concerns inference times. As a reminder, the average inference time of our YOLOv8 detection model is around 5ms to 8ms per frame with a default image size of 640x640. Using a larger image size allows for better detector accuracy but also increases the inference time (up to 30ms for an image size of 1088x1664). We aim to simulate an online tracking process and thus operate as close to real-time as possible. We will also assume that the optimization of our pipeline is "perfect," meaning that it requires less than 1ms of inference time, something which seems

challenging to achieve. At a rate of 30 frames per second (which represents a maximum time of 33.3 ms per frame), we limit the inference times of our trackers to 33ms - 1ms - 8ms = 24 ms per frame.
Firstly, let's note that the average inference times of our trackers are generally below this limit, with the exception of DeepSort. Figure 4.2.1 show that DeepSort presents an inference time well beyond the limit we have set. This is due to its feature extraction via CNN followed by its associative process based on the visual features of detection. As this process is too demanding in terms of computation time, we exclude DeepSort for real-time use case (although it may be possible to optimize it by adjusting memory storage parameters related to tracks, even at the risk of some qualitative losses).
No other tracker shows an elimination-worthy inference time. ByteTrack presents the best time, followed closely by SparseTrack, then SORT, which is the slowest of the evaluated algorithms.

In general, due to its simplistic structure, SORT is the weakest of our evaluated trackers. From figure 4.2.2, ByteTrack and SparseTrack both present similar HOTA metrics for each α value (IoU threshold used for association). The two trackers differ on their MOTA and IDF1 scores. ByteTrack presents a higher MOTA score while SparseTrack dominates on the IDF1 score. We can infer from this that SparseTrack manages to provide more coherent and robust tracking over time, whereas ByteTrack results in more complete tracking by considering a more accurate number of detections.

As expected, working with better detections by fine-tuning the model significantly increases the performance of our tracking. This has an even greater effect if the tracker in question displays poor results. Our choice to work with a fine-tuned model rather than directly with the ground-truth detections reflects our desire to maintain a systemic dimension (detector + tracker) in our evaluation.
In our case, the cascade association process with depth from SparseTrack appears to yield better associative results at the expense of a more extensive detection filtering affecting the MOTA metric. Due to this particular filtering, SparseTrack does not perform as well as ByteTrack in finding all ground-truth detections, but it is better at limiting false positives. It is also more robust in its association, avoiding splitting a single ground-truth trajectory into several tracks, but it is more prone to merging several objects onto the same track.

The choice of the best tracker does not seem absolute and relies on the specific needs of the problem. We aim to achieve the most versatile system possible. By observing the HOTA results on each video of the MOT dataset, we note that although ByteTrack has a better global metric, it only outperforms SparseTrack on MOT20-05. We thereby deduce that SparseTrack is the most generalizable of our two trackers and is therefore our preferred option for a tracking task within crowds.

5.2.3 Anomaly Detection

In this project, we have defined several algorithms serving as baselines for anomaly detection. Our work was constrained by time and resource limitations that prevented us from generating dedicated anomaly datasets and from contemplating more advanced solutions to the problem. However, we hope that the established base will be further elaborated upon in future studies.
Moreover, we are convinced of the positive societal impact of such technological advancements made available as open-source, whether in terms of security, health, or commerce.
For instance, an AI capable of fall detection could be employed to monitor the safety of elderly individuals in retirement homes or to ensure enhanced workplace safety. Similarly, an AI for detecting prolonged presence could be applied to patient tracking in healthcare establishments, enabling medical staff to optimize their time or monitor adherence to medical protocols.

6 Conclusion

This study presents a comprehensive proof-of-concept for an end-to-end multitask anomaly detection tool, adept in multi-object detection, online tracking, and anomaly detection. The tool also demonstrates functionalities such as pose detection and a robust evaluation framework for the integrated models and algorithms. While promising, the project reveals several areas needing improvement for transitioning from a research demonstration to a commercially viable product.

6.1 Key Findings and Implications

Our results highlight the effectiveness of the YOLOv8n-ppl model for object detection tasks due to its remarkable speed and satisfactory performance metrics. This model's capability to process up to 205 frames per second positions it favorably for real-time applications, significantly surpassing competitors like Faster R-CNN and RetinaNet. This speed advantage enables a greater focus on refining tracking algorithms within real-time constraints.
For tracking, SparseTrack emerged as the preferred choice due to its superior generalizability across diverse scenarios, despite ByteTrack's higher MOTA scores. SparseTrack's robustness in maintaining coherent tracking and minimizing false positives makes it suitable for densely populated environments typical of crowd monitoring.

6.2 Enhancements and Future Work

To bridge the gap towards commercial deployment, several enhancements are necessary:

- **Dataset Diversity**: Incorporating a broader range of human poses and contexts to enhance model generalizability.
- **Inference Times**: Further reducing model and especially trackers inference times to bolster real-time processing capabilities.
- **Adaptability**: Enhancing model adaptability to handle variations in human sizes and behaviors.
- **Context-Specific Tracker**: Developing trackers optimized for specific contexts, such as CCTV monitoring, by introducing entry and exit zone constraints.
- **Pipeline Optimization**: Creating an optimized pipeline designed for selected detection and tracking models to ensure efficient real-world application.

6.3 Societal Impact and Ethical Considerations

The open-source nature of this tool harbors significant potential for societal benefits, particularly in security, healthcare, and commerce. For instance, fall detection AI could enhance safety in eldercare facilities, while prolonged presence detection could optimize patient monitoring in healthcare settings. However, deploying these technologies necessitates strict adherence to ethical guidelines, respecting individual freedoms and fundamental human rights, and compliance with emerging AI regulations.

In summary, this project establishes a robust initial benchmark for human crowd detection, tracking, and anomaly detection algorithms. The findings and methodologies provide a solid foundation for future research and development. Continued efforts to enhance dataset diversity, model adaptability, and pipeline efficiency will be crucial in transitioning from a research prototype to a deployable commercial application. By advancing these technologies responsibly, we can harness their potential for significant societal benefits while ensuring ethical compliance and respect for human rights.

7 Sources

Images

(1) https://blogs.idc.com/2020/06/19/explore-the-video-surveillance-market-with-idc/
(2) https://www.comparitech.com/vpn-privacy/the-worlds-most-surveilled-cities/
(3) https://www.interpretable.ai/interpretability/what/
(4) https://www.forbes.com/sites/glenngow/2020/08/21/environmental-sustainability-and-ai/
(5) https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/
(6) https://medium.com/@kudryavtsev_ia/high-performance-sort-tracker-in-rust-9a1dd18c259c#:~:text=in%20the%20SORT%20algorithm%20Kalman,by%20the%20filter%20through%20observations.
(7) https://www.researchgate.net/figure/Object-tracking-procedure-of-SORT-14_fig1_352498971
(8) https://www.researchgate.net/figure/Object-tracking-procedure-of-DeepSORT-13_fig2_352498971
(9) https://jonathonluiten.medium.com/how-to-evaluate-tracking-with-the-hota-metrics-754036d183e1
(10) https://autonomousvision.github.io/hota-metrics/
(11) https://docs.ultralytics.com/datasets/detect/#ultralytics-yolo-format
(12) https://paperswithcode.com/method/retinanet
(13) https://medium.com/analytics-vidhya/neural-network-part1-inside-a-single-neuron-fee5e44f1e
(14) https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks
(15) https://builtin.com/machine-learning/fully-connected-layer
(16) https://www.v7labs.com/blog/recurrent-neural-networks-guide
(17) https://www.upgrad.com/blog/basic-cnn-architecture/
(18) https://www.researchgate.net/publication/340712186_Activity_Monitoring_for_ICU_Patients_Using_Deep_Learning_and_Image_Processing
(19) https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610
(20) https://pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/
(21) https://medium.com/@VK_Venkatkumar/yolov8-architecture-cow-counter-with-region-based-dragging-using-yolov8-e75b3ac71ed8

Other

- Russell S., and Norvig P., (2016). Artificial Intelligence: A Modern Approach. Pearson Education Limited.
- https://www.nytimes.com/2023/05/30/technology/ai-threat-warning.html
- https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Viola_et_Jones
- https://en.wikipedia.org/wiki/Artificial_intelligence_for_video_surveillance
- https://arxiv.org/abs/2305.09972
- https://arxiv.org/abs/2211.05778
- https://arxiv.org/abs/2211.12860
- https://en.wikipedia.org/wiki/Scrum_(software_development)#:~:text=Scrum%20is%20an%20agile%20team,and%20commonly%20lasts%20two%20weeks.
- https://fr.wikipedia.org/wiki/Test_driven_development
- https://www.amnesty.org/en/latest/campaigns/2015/03/easy-guide-to-mass-surveillance/
- https://www.amnesty.org/fr/latest/news/2023/12/eu-blocs-decision-to-not-ban-public-mass-surveillance-in-ai-act-sets-a-devastating-global-precedent/
- https://en.wikipedia.org/wiki/Nineteen_Eighty-Four
- https://www.cnil.fr/fr/la-videosurveillance-videoprotection-sur-la-voie-publique
- https://edps.europa.eu/data-protection/data-protection/reference-library/video-surveillance_en
- https://www.europarl.europa.eu/news/fr/press-room/20231206IPR15699/loi-sur-l-intelligence-artificielle-accord-sur-des-regles-globales
- https://web.archive.org/web/20130831072750/http://surveillance.rsf.org/en/
- https://ghostarchive.org/archive/ToOUz
- https://www.wsj.com/articles/a-billion-surveillance-cameras-forecast-to-be-watching-within-two-years-11575565402
- https://www.researchgate.net/publication/360555992_An_Effective_Video_Summarization_Framework_Based_on_the_Object_of_Interest_Using_Deep_Learning

- https://fra.europa.eu/fr/publication/2022/bias-algorithm
- https://www.midjourney.com/home?callbackUrl=%2Fexplore
- https://www.ibm.com/blog/shedding-light-on-ai-bias-with-real-world-examples/
- https://www.thomsonreuters.com/en-us/posts/wp-content/uploads/sites/20/2023/08/Addressing-Bias-in-AI-Report.pdf
- https://www.wwt.com/article/introduction-to-ai-model-security
- https://www2.deloitte.com/content/dam/Deloitte/nl/Documents/innovatie/deloitte-nl-innovation-bringing-transparency-and-ethics-into-ai.pdf
- https://earth.org/2023-to-be-hottest-year-ever-recorded-at-1-4c-above-pre-industrial-levels/
- https://earth.org/co2-levels-peak/
- https://earth.org/68-decline-in-species-population-sizes/
- https://earth.org/the-green-dilemma-can-ai-fulfil-its-potential-without-harming-the-environment/
- https://en.wikipedia.org/wiki/Moore%27s_law
- https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/
- https://www.spglobal.com/marketintelligence/en/news-insights/trending/HyvwuXMO9YgqHfj7J6tGlA2 - College of information and Computer Sciences at University of Massachusetts Amherst
- https://www3.weforum.org/docs/WEF_A_New_Circular_Vision_for_Electronics.pdf
- https://earth.org/pros-and-cons-of-self-driving-cars/
- https://arxiv.org/abs/2204.12484
- https://streamlit.io/
- https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem
- https://en.wikipedia.org/wiki/Feature_learning
- https://medium.com/analytics-vidhya/neural-network-part1-inside-a-single-neuron-fee5e44f1e
- https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels
- https://www.researchgate.net/figure/Pooling-layer-operation-oproaches-1-Pooling-layers-For-the-function-of-decreasing-the_fig4_340812216
- https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b
- https://www.v7labs.com/blog/recurrent-neural-networks-guide
- https://www.upgrad.com/blog/basic-cnn-architecture/
- https://paperswithcode.com/paper/rich-feature-hierarchies-for-accurate-object
- https://arxiv.org/abs/1612.03144
- https://arxiv.org/abs/1506.01497v3
- https://arxiv.org/abs/1504.08083
- https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
- https://ai.meta.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library-/
- https://docs.ultralytics.com/fr/
- https://arxiv.org/pdf/1712.00726.pdf
- https://machinelearningknowledge.ai/a-brief-history-of-yolo-object-detection-models/
- https://medium.com/@VK_Venkatkumar/yolov8-architecture-cow-counter-with-region-based-dragging-using-yolov8-e75b3ac71ed8
- https://cocodataset.org/#home
- https://www.crowdhuman.org/
- https://paperswithcode.com/dataset/citypersons
- https://paperswithcode.com/dataset/smartcity
- http://www.cbsr.ia.ac.cn/users/sfzhang/WiderPerson/
- https://universe.roboflow.com/falldataset/newfall
- https://universe.roboflow.com/ws-jhark/ppl-ees11
- https://production-media.paperswithcode.com/methods/Screen_Shot_2020-06-07_at_4.22.37_PM.png

8 Bibliography

[1]. Russell, Stuart, and Peter Norvig. Artificial Intelligence: A Modern Approach. Pearson Education Limited, 2016.
[2]. Reis, Dillon, et al. "Real-time flying object detection with YOLOv8." arXiv preprint arXiv:2305.09972 (2023).
[3]. Wang, Wenwei, et al. "InternImage: Exploring large-scale vision foundation models with deformable convolutions." arXiv preprint arXiv:2211.05778 [cs.CV] (2022).
[4]. Zong, Zongwei, et al. "DETRs with collaborative hybrid assignments training." arXiv preprint arXiv:2211.12860 (2022).
[5]. Xu, Yilun, et al. "ViTPose: Simple vision transformer baselines for human pose estimation." arXiv preprint arXiv:2204.12484 (2022).
[6]. Bewley, Alex, et al. "Simple online and realtime tracking." arXiv preprint arXiv:1602.00763 [cs.CV] (2016).
[7]. Wojke, Nicolai, Alex Bewley, and Dietrich Paulus. "Simple online and realtime tracking with a deep association metric." arXiv preprint arXiv:1703.07402 [cs.CV] (2017).
[8]. Zhang, Yifu, et al. "ByteTrack: Multi-object tracking by associating every detection box." arXiv preprint arXiv:2110.06864 [cs.CV] (2021).
[9]. Liu, Zhichao, et al. "SparseTrack: Multi-object tracking by performing scene decomposition based on pseudo-depth." arXiv preprint arXiv:2306.05238 [cs.CV] (2023).
[10]. Luiten, Jonathon, et al. "HOTA: A higher order metric for evaluating multi-object tracking." arXiv preprint arXiv:2009.07736 [cs.CV] (2020).
[11]. Ristani, Ergys, et al. "Performance measures and a data set for multi-target, multi-camera tracking." arXiv preprint arXiv:1609.01775 [cs.CV] (2016).
[12]. Bernardin, Keni, et al. "Multiple object tracking performance metrics and evaluation in a smart room environment." Proceedings of IEEE International Workshop on Visual Surveillance (2006).
[13]. Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." arXiv preprint arXiv:1311.2524 (2014).
[14]. Lin, Tsung-Yi, et al. "Focal loss for dense object detection." arXiv preprint arXiv:1708.02002 (2017).
[15]. Ren, Shaoqing, et al. "Faster R-CNN: Towards real-time object detection with region proposal networks." arXiv preprint arXiv:1506.01497 (2015).
[16]. Girshick, Ross. "Fast R-CNN." arXiv preprint arXiv:1504.08083 (2015).
[17]. Cai, Zhaowei, and Nuno Vasconcelos. "Cascade R-CNN: Delving into high quality object detection." arXiv preprint arXiv:1712.00726 (2017).
[18]. Lin, Tsung-Yi, et al. "Microsoft COCO: Common objects in context." arXiv preprint arXiv:1405.0312 (2014).
[19]. Shao, Shuai, et al. "CrowdHuman: A benchmark for detecting human in a crowd." arXiv preprint arXiv:1805.00123 [cs.CV] (2018).
[20]. Zhang, Shanshan, et al. "CityPersons: A diverse dataset for pedestrian detection." arXiv preprint arXiv:1702.05693 (2017).
[21]. Zhang, Li, et al. "Crowd counting via scale-adaptive convolutional neural network." arXiv preprint arXiv:1711.04433 [cs.CV] (2017).
[22]. Zhang, Shaoli, et al. "WiderPerson: A diverse dataset for dense pedestrian detection in the wild." IEEE Transactions on Multimedia (TMM) (2019).
[23]. falldataset. "newfall Dataset." Roboflow Universe, 2022. Retrieved from https://universe.roboflow.com/falldataset/newfall.
[24]. ws. "ppl Dataset." Roboflow Universe, 2023. Retrieved from https://universe.roboflow.com/ws-jhark/ppl-ees11.
[25]. Uijlings, Jasper R. R., et al. "Selective search for object recognition." International Journal of Computer Vision 104.2 (2013): 154–171. https://doi.org/10.1007/s11263-013-0620-5.
[26]. Dendorfer, Patrick, et al. "MOT20: A benchmark for multi-object tracking in crowded scenes." arXiv preprint arXiv:1906.04567 [cs] (2020).
[27]. Haq, Hamid Bux, et al. "An effective video summarization framework based on the object of interest using deep learning." Mathematical Problems in Engineering 2022 (2022): 1–25. https://doi.org/10.1155/2022/7453744.
[28]. Huo, Yuankai, et al. "Deep neural networks on chip: A survey." BigComp, 2020. https://doi.org/10.1109/BigComp48618.2020.00016.
[29]. Hieronymus, Magnus, and J. Nycander. "Finding the Minimum Potential Energy State by Adiabatic Parcel Rearrangements with a Nonlinear Equation of State: An Exact Solution in Polynomial Time." Journal of Physical Oceanography, vol. 45, 2015, pp. 1504-1523. https://doi.org/10.1175/JPO-D-14-0174.1.
[30]. Huang, Yanru, et al. "SQE: A Self Quality Evaluation Metric for Parameters Optimization in Multi-Object Tracking." arXiv preprint arXiv:2004.07472 [cs.CV] (2020). https://doi.org/10.48550/arXiv.2004.07472.

9 Appendix

9.1 Demonstration Application

Our web application aims to enable the user to individually evaluate multiple model configurations, algorithms, and parameters on their own data. As our system and pipeline have been designed for broad generalization, it should be capable of delivering satisfactory results for a wide range of real-life data. Based on our obtained results, we recommend an optimal configuration for each 3 features of our system in Table 9. However, the user has the liberty to adjust each entry according to their preference. Of course, many more parameters are considered within our different algorithms.

Parameter	**Value**
Detection	
Pre-trained Detection Model	Yolov8-ppl
IoU threshold for NMS	0.70
Detection threshold	0.20
Tracking	
Pre-trained Detection Model	Yolov8-ppl
Detection threshold	0.08
Tracking Algorithm	SparseTrack
IoU threshold for association	0.6 for SparseTrack *Depends on the tracker selected*
Anomaly Detection	
Anomaly to Detect	All are selected
Pre-trained Detection Model	Yolov8-ppl
Tracking Algorithm	SparseTrack
Specific to fall detection	
Pose Estimator Model	VITTransformer
Keypoints file	None

Table 9.1.1 - Recommended parameters for optimal usage

We have also incorporated a maximum amount of information within our web application to guide the user. This is evident on the homepage and also through a mouse hover over the various parameter fields to obtain more specific information.

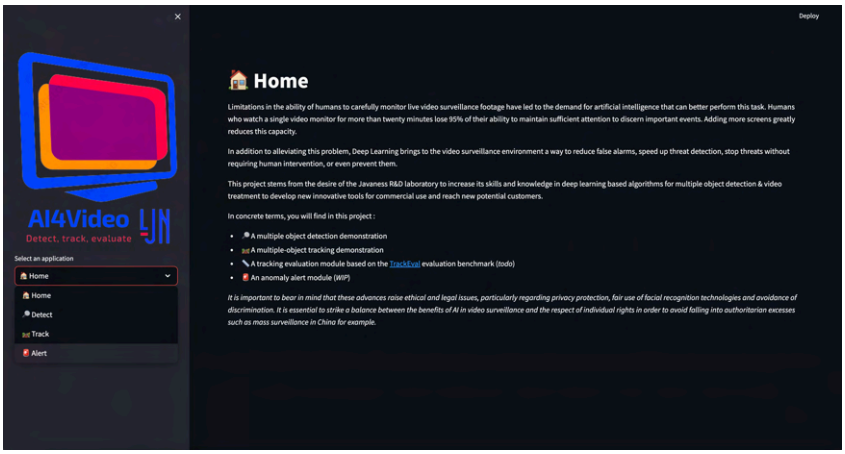

Figure 9.1.1 - Home Page of the Demonstration Web Application

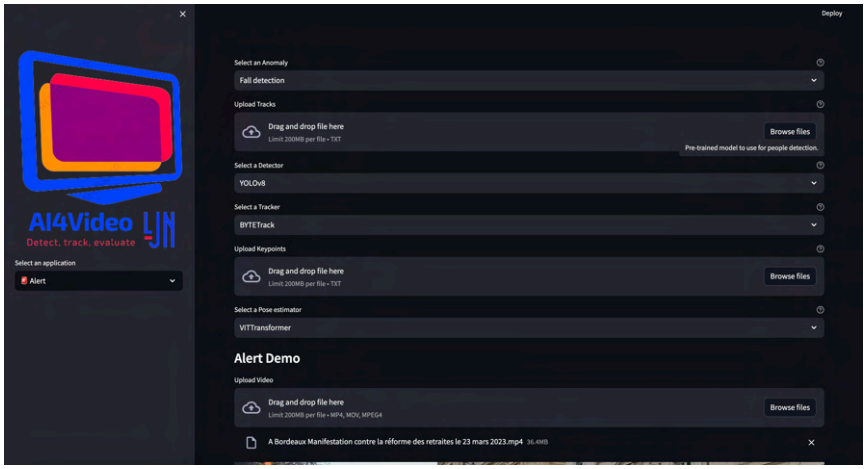

Figure 9.1.2 - Sample of possible parameters tuning for Fall Detection

Ultimately, at the end of the inference, the user can easily view the results of our system and also, with a simple click, download the output files from our pipeline (namely the detection coordinates, tracking data, pose detection, and anomaly detection data).

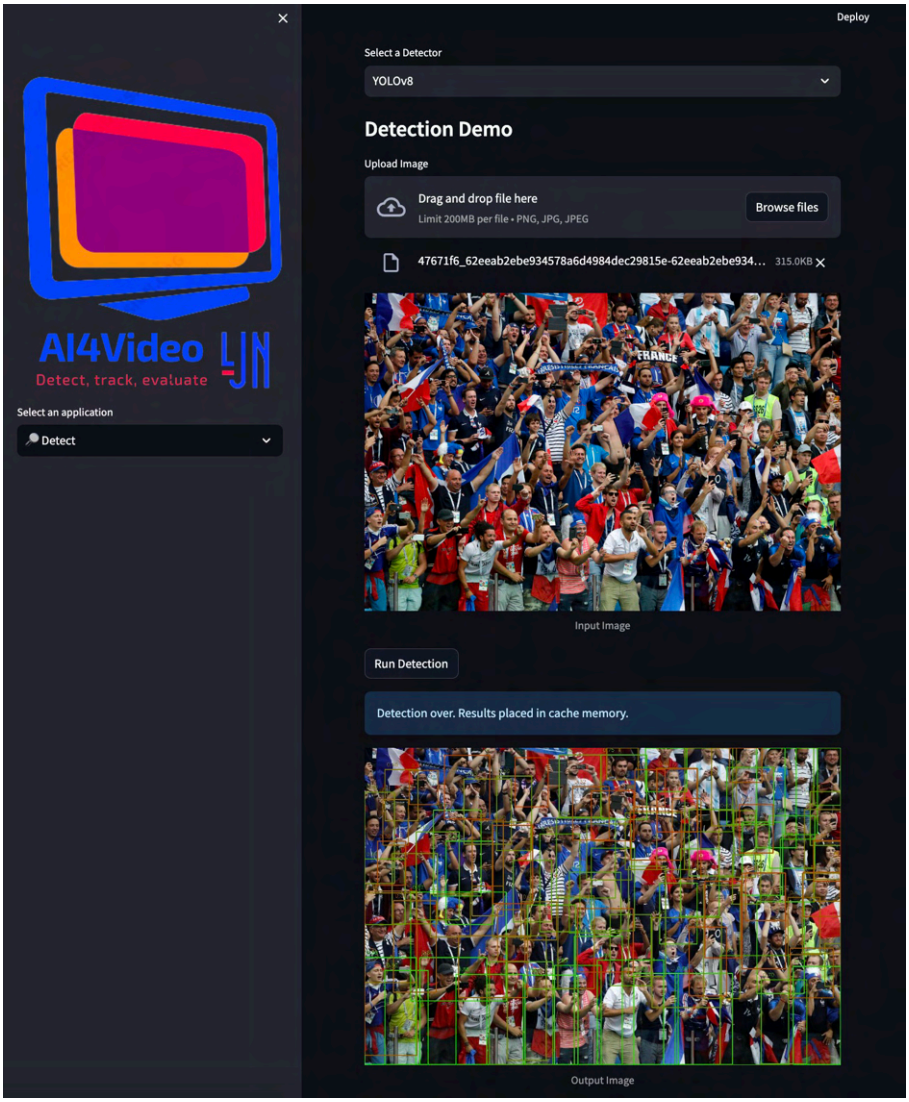

Figure 9.1.3 - Results visualization example. The user can also download the resulting files by clicking on a button