Alma Mater Studiorum · University of Bologna

School of Science
Department of Physics and Astronomy
Master Degree in Physics

Video analysis for a detailed reconstruction of individual and crowd dynamics

Supervisor:
Prof. Armando Bazzani

Submitted by:
Daniele Pucci

Academic Year 2023/2024

Abstract

Crowd density estimation and crowd dynamics analysis are critical tasks in modern urban management, event planning, and public safety. This thesis explores a range of computer vision methodologies to estimate crowd density in the unique context of Venice's crowded public spaces. The study implements and evaluates several techniques, including the state-of-the-art YOLOv8 object detection model, a custom U-Net for crowd segmentation, a traditional background subtraction algorithm, and a novel approach based on trajectory tracking. Each method is evaluated in terms of accuracy and computational efficiency on both GPU-powered systems and resource-constrained devices like the Raspberry Pi 5.

While YOLOv8 delivers high accuracy in low-density crowds, it struggles in dense settings and demands substantial computational resources. In contrast, the U-Net-based model shows reliable performance across both low and high-density scenarios, though its counting accuracy can be imprecise. The traditional background subtraction approach excels in real-time processing on low-power devices but its output tends to be less precise and harder to interpret. The novel tracking-based approach offers a promising alternative for estimating crowd density by analyzing trajectory patterns, though quantitative evaluation wasn't possible and more research need to be conducted.

The results provide a comparative framework to guide the selection of crowd analysis techniques based on the density of the crowd and the available computational resources. This work not only contributes to the research in crowd analysis and computer vision, but also lays the foundation for future research in real-time crowd management and analysis.

Contents

Chapter 1

Introduction

Crowds, consisting of individuals gathered within a specific physical space, have long been subjects of interest in both social and computational fields. Nowadays, one of the most interesting and active research topic of research within computer vision is the analysis of crowds [1][2], driven by advances in machine learning, computational power, and the vast amount of data generated daily from surveillance systems. These holds significant potential for understanding crowd behaviour and movement patterns, and preventing dangerous situations, but also requires effective analysis to extract value [2]. As a result, crowd analysis has become a key aspect of urban management, public safety and event planning. Crowd analysis encompasses a wide range of tasks, but at its core, it can be roughly divided into the task of understanding crowd behaviour [2], or measuring or estimating the number of people in a place, usually referred to as *crowd counting* [3] or *crowd density estimation*. Crowd counting is a particularly important task, because of the desire to make population estimates without physical measurements. This is especially useful in unconstrained environments where multiple entry and exit points exist and traditional methods of measurement, such as ticket sales or turnstile counts, are impractical or impossible. The relevance of crowd analysis extends far beyond academic curiosity. Attendance and density statistics are useful for measuring the success of an event and the planning of future events. Understanding crowd density is crucial in predicting potentially dangerous situations, such as stampedes or compressive asphyxia, where overcrowding can lead to tragic consequences [4]. With more than 420,000 academic papers published on crowd analysis in the last decade (see Fig. 1.1), it is evident that this field has many application across multiple domains, including urban planning, public safety, and social behavior.

While many works in the literature focus on the analysis of available data to solve one or more cited problems, or the creation and validation of mathematical [5] and physical models [6][7] and simulations [8], this thesis takes a step back and specifically address the first step in crowd analysis, which it the gathering of meaningful information from raw visual data (video and image sources). The ability to interpret crowd density from video

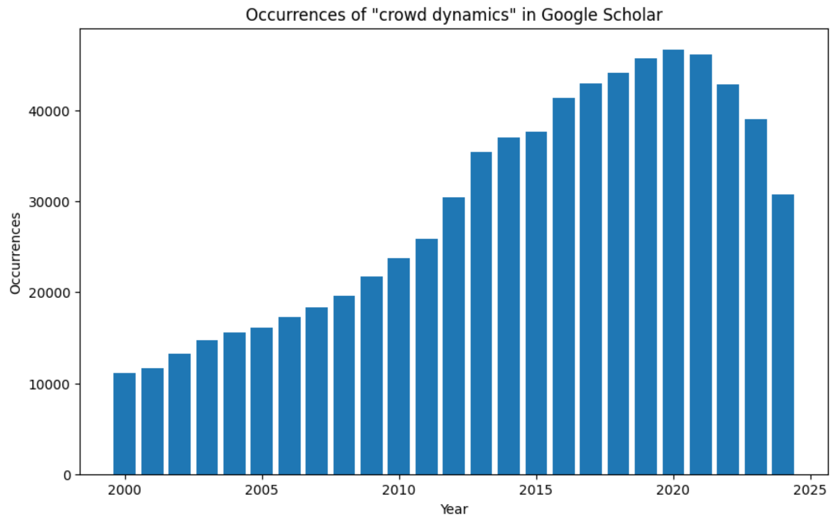


Figure 1.1: Occurrences of the keyword *crowd analysis* on Google Scholar from the year 2000.

feeds has immediate applications in public event management, transportation systems, and emergency response. In cities like Venice, narrow alleyways and public squares regularly host huge numbers of tourists, and effective crowd monitoring is essential.

This work explore a variety of methodologies that have been extensively studied in the literature, including deep learning-based object detection, segmentation models, and more traditional image processing techniques. For example, YOLOv8 [9] represents an object detection framework that is often considered the state-of-the-art for real-time applications [10][11]. Its ability to quickly detect and track individuals within a scene makes it a popular choice for crowd counting and density estimation, particularly in lower-density environments. However, as this thesis will demonstrate, YOLO struggles in more densely packed scenarios, where individuals are more likely to overlap or occlude one another, thus requiring alternative methods.

Another key model explored in this thesis is the U-Net architecture [12], which has been successfully applied in various domains for tasks such as image segmentation, and it was thus considered a natural candidate for crowd density estimation in this study. By segmenting people's heads in crowded environments, U-Net allows for a more refined estimation of crowd density, even in highly congested spaces where object detection models may fail.

In addition to deep learning approaches, this thesis also incorporates a more traditional computer vision approach: background subtraction [13][14]. This technique is simpler and less computationally intensive than deep learning models and can thus provide fast and effective foreground detection, making it suitable for real-time crowd analysis on low-power, CPU-only devices such as the Raspberry Pi. Background subtraction is

less sophisticated than machine-learning-based methods, but remains a viable solution for specific crowd monitoring tasks in constrained environments where computational resources are limited.
Finally, this thesis introduces a novel approach to estimating crowd density based on tracking individuals over time. By analyzing the features of people's trajectories (such as length and linearity), it is possible to infer crowd density without relying on precise object detection, segmentation or tracking. This method offers a promising alternative in cases where traditional detection and counting approaches fail due to crowd size or occlusion.

The primary goal of this thesis is to evaluate the effectiveness of different methods for crowd density estimation and dynamic reconstruction, particularly in the urban landscape of Venice. Each method offers distinct advantages and disadvantages depending on the specific context in which it is applied. The thesis provides a comparative analysis of these methods, with particular attention to their computational performance and accuracy in both sparse and densely packed crowd scenarios.

Alongside with the broader context of crowd analysis literature, the thesis contributes to improve public safety, urban planning, and event management through the effective use of video data. The natural applications of this work extend beyond the only crowd analysis: each sub-field of image/video analysis or computer vision needs a way to extract data from cameras and thus it is a potential target of this research. Moreover, the ability to extract features from a video can be useful in many other typically-uncorrelated fields, such as wildlife monitoring, sports and physics (e.g. analysing the trajectory of a particle from a video).

This thesis is organized as follows: the problem is first presented and related works are reviewed to provide context and background. Next, selected models and algorithms are explained in detail, implemented and applied to the available dataset of crowd scenes from Venice. Their computational performance is then measured, comparing the suitability of each method for real-time analysis on different hardware platforms, including a CUDA-capable GPU and a Raspberry Pi 5. Finally conclusions are drawn and directions for future development are outlined.

Chapter 2

Related works

Methods of crowd counting can be broadly divided into two classes: pixel/feature-level approaches and object-level approaches.

Usually, pixel-level approaches are based on traditional computer vision techniques and are thus slim and fast. Most of this algorithms makes large use of background (BG) subtraction: the differences lay on how the background is obtained and how the foreground is interpreted in order to have an estimation of people count. Davies et al. [15] used an edge detector and correlated the number of pixel feature to the number of people in the scene, while Regazzoni et al. used vertical edges. Cho et al. [16] approach was based instead on the ratio of foreground to background pixel counts in the scene.

Object-level analysis aims to extract and identify individual subjects in a scene and consequently, tends to produce more accurate results when compared to pixel-based methods. However, object-level approaches are generally only applicable to lower density crowds, because the clutter and severe occlusion that occurs in high-density crowds make such approaches much more challenging [4][1]. Nevertheless, many advances in object detection was done thanks to the progress in Machine Learning [17] and nowadays, many Deep Learning [18] models compete for the best performances. Usually, these networks require training and the more or less extensive use of GPUs, as opposed to the previously-cited traditional computer vision approaches.

In particular, Deep Neural Networks (DNNs) based on Convolutional Neural Networks (CNNs) [19], have emerged as the de facto standard approach for detecting objects [20]. Among these, models of the YOLO [21] series are off-the-shelf networks that are easy to use and have significantly increased in popularity due to their real-time performance and accurate object localization capabilities. Indeed they are widely recognized as a state-of-the-art algorithm when it comes to object detection [10][11]. From the original paper in 2015 by Redmon et al. [21], YOLO has seen multiple iterations, up to YOLOv10 [22] which is currently (Sept. 2024) only a Preprint under review. Briefly: YOLOv2 [23] was released in 2016 and improved the original model, for instance by incorporating Batch Normalization [24] (see Appendix A.2 for detail); YOLOv3 [25], launched in 2018, further enhanced the model's performance using a more efficient backbone network and Spatial Pyramid Pooling [26]; YOLOv4 [27] was released in 2020, introducing innovations like Mosaic Data Augmentation [28] and a new loss function; YOLOv5 [29] further improved the model's performance and added new features such as hyperparameter optimization[1] [30]; YOLOv6 [31] was open-sourced in 2022 and is in use in many of the company's autonomous delivery robots; YOLOv7 added additional tasks such as pose estimation; YOLOv8 [9] introduced big improvements for enhanced performance and efficiency; YOLOv9 [32] was published in Feb. 2024 and include improvements in efficiency and accuracy in real-time detection.

In Fig. 2.1 the performances of YOLOv9 with respect to the previous versions is plotted. The measured performance in the graph is the Average Precision (AP) over the MS COCO dataset [33], a widely used benchmark for object detection models, consisting of diverse images with complex scenes, that provides standardized evaluation metrics to compare models. See in Appendix A.1 for the definition of Average Precision.

Other CNN-based networks have been used for crowd counting, like U-shaped networks (U-Nets): Cao et al. developed CrowdUNet [34], while Marcellino et al. [35] used a modified U-Net with Scale Pyramid to count people.

To achieve the goals of this work, the analysis of the crowd dynamics (i.e., how crowd detection changes over time) was considered useful to have a better understanding of crowd density [36] (see section 3.4), so in the following, a brief review of tracking methods is also presented.

The most successful methods currently available in literature can be broadly grouped into three main categories [20]: tracking-by-detection [37][38][39], tracking-by-regression [40][41][42], and tracking-by-attention [43][44][45]. In tracking-by-detection, detections are computed independently for each frame and associated with tracks in subsequent steps. Tracking-by-regression unifies detection and motion analysis, with a single module that simultaneously locates the detections and their displacement w.r.t. the previous frame. Finally, in tracking-by-attention, an end-to-end deep tracker based on self-attention [46] manages the life-cycle of a set of track predictions through the video sequence.

Since the advent of deep learning, advances in object detection drove the community towards tracking-by-detection [20]. Among the most successful works, SORT [47] uses a combination of Kalman filter [48] and Hungarian algorithm [49] to associate detections between different frames and thus reconstruct the dynamic of the detected object. Track-

[1]Hyperparameters are the parameters which must be configured before the training process start (such as learning rate, batch size, number of layers or neurons) and they govern the behaviour of the learning process, as opposed to the parameters (or weights) learned from the data. Hyperparameter optimization is the process of determining the best set of hyperparameters that yields an optimal model, by using cross-validation and choosing the set of values that maximizes a predefined loss function.

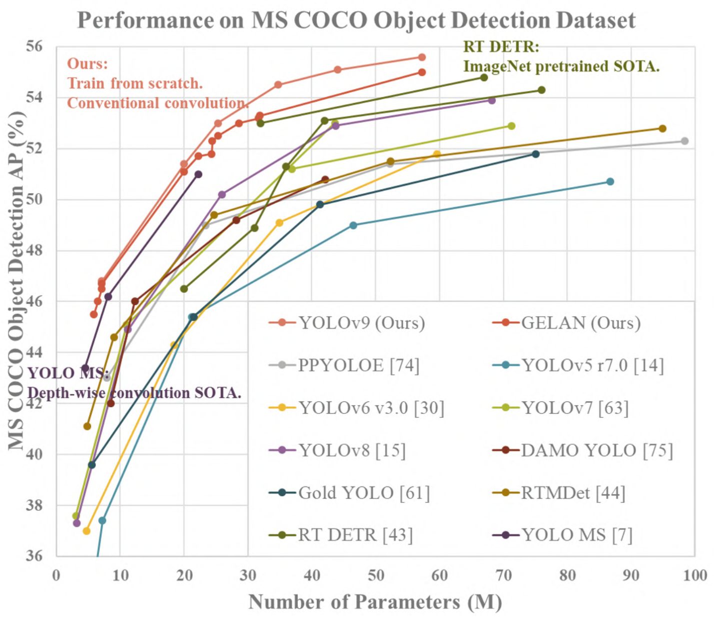


Figure 2.1: Comparisons of the real-time object detectors on MS COCO dataset. YOLOv9 method surpassed all previous methods in terms of object detection performance. Image source [32]

tor [50] pushes tracking-by-detection to the edge by relying solely on an object detector to perform tracking. CenterTrack [51] provides a point-based framework for joint detection and tracking based on CenterNet [52] Similarly, RetinaTrack [53] extends RetinaNet [54] to offer a conceptually simple and efficient joint model for detection and tracking, leveraging instance-level embeddings. More recently, DeepSORT [55][56] and ByteTrack [57][58] further establish this paradigm, unleashing the full potential of YOLO [21][9]: notably, it uses almost every predicted detection, and not only the most confident ones.

The downside of these works is that they utilise bounding boxes (i.e., rectangle boxes that enclose detected objects) to perform both detection and tracking. Later on (section 3.4), it is discussed the need for a tracker that works with point detections and a simpler version of SORT is presented for this purpose.

Chapter 3

Working principle of selected models and algorithms

In this chapter, some of the previously cited methods are better explained, and their efficiency and limitations are discussed. These are: YOLOv8, a custom U-Net implementation, a backgroud-subtraction-based algorithm and, regarding tracking, a simplified implementation of SORT. Both YOLO and U-Net architecture are based on CNNs and their core functionality is similar, but the way they are practically used and trained is quite different.

Convolutional Neural Networks (CNNs) [19] are a class of deep learning [18] models designed to analyze and process images. As opposed to traditional image processing techniques, that relied on features like edges or contours, CNNs automatically learn hierarchical feature representations from raw image data. Like all Artificial Neural Networks (ANNs), CNNs are comprised of neurons that self-optimise through learning [59]. Mimicking a biological neural network in animal brains, each neuron receives an input, perform a operation and passes the result to a neuron on the next layer. The key aspect of a CNN architecture is that hidden layers include one or more *convolutional layers*: a layer that perform convolution between the input matrix (the image) and a kernel, (a small square matrix). Considering images as discrete functions on a finite 2D domain, the convolution operation between two images f and g can be defined as:

$$(f * g)[m, n] = \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} f[j, k]g[m - j, n - k]$$

The weights of convolutions' kernel is what is learnable and changes with training.

After each convolutional layer, the output is passed through a non-linear *activation function* that introduce non-linearity into the model, enabling it to learn complex patterns and preventing the vanishing gradient problem.

Then, pooling layers aim to gradually reduce the spatial dimension of the feature map, decreasing the number of parameters and the computational complexity of the model. In CNNs, 2×2 *max-pooling layers* are usually used, so the maximum value from a 2×2 window is selected and the dimension is thus scaled down to 1/4 of the original size.

3.1 YOLO (You Only Look Once)

In Chapter 2, iterations of YOLO architecture up to YOLOv10 have been presented. However, YOLOv8 has been used for this work. This choice was done because, while YOLOv9 and YOLOv10 offer novel innovations, the improvements were considered to be insufficient to reject the use of YOLOv8, which provides a reliable tool for this research. In particular, YOLOv8 has a comprehensive documentation and tutorials [60] which make it easier to understand, use in a custom code and debug if necessary. Moreover, it has been available for a longer time and it has undergone extensive testing, improvements, and community support since its release in January 2023. YOLOv9 is relatively new (Feb. 2024) and has fewer real-world applications and case studies available: as far as it can be determined, only one real-world application [61] shows YOLOv9 usage and how it exceed previous iterations' performances. The same argument can be applied even more so to the newer YOLOv10.

The name YOLO stands for *You Only Look Once*, referring to the fact that it is able to accomplish the detection task with a single pass of the network, as opposed to previous approaches that divided the task into two steps, where the first step detects possible regions with objects and the second step run a classifier on the proposals [10]. To accomplish this, YOLO divides the input image into an $S \times S$ grid and predicts B bounding boxes of the same class, along with its confidence for C different classes per grid element. Each bounding box prediction consists of five values: Pc, bx, by, bh, bw, where Pc is the confidence score for the box that reflects how confident the model is that the box contains an object; bx and by are the coordinates of the box's center, and bh and bw are the height and width of the box. The output of YOLO is a tensor of $S \times S \times (B \times 5 + C)$, optionally followed by non-maximum suppression (NMS) to remove duplicate detections.

As already mentioned, YOLO is a CNN, and YOLOv1 architecture comprises 24 1×1 convolutional layers followed by 2 fully connected layers that predict the bounding box coordinates and probabilities. All layers used Leaky ReLU [62]:

$$\text{LReLU} = \max(ax, x) \qquad a < 1$$

except for the last one, which used a linear activation function. This simple architecture, along with its novel full-image one-shot regression, made it much faster than the existing object detectors but it had limitations. In particular, it could only detect at most two objects of the same class in the grid cell ($B = 2$ in the original paper), limiting its ability to predict nearby objects. Also, it struggled to predict objects with aspect ratios not seen in the training data.

YOLOv8 has a more complex architecture whose details are depicted in Fig. 3.1. Briefly, convolution modules include Batch Normalization and SiLU (Sigmoid Linear Unit) activation function [63], defined as:

$$\mathrm{SiLU}(x) = x \ \mathrm{sigmoid}(x) = \frac{x}{1 + e^{-x}}$$

These modules are alternated with the so-called C2f module, that combines high-level features with contextual information to improve detection accuracy. In YOLOv8, objectness, classification, and regression tasks are performed independently allowing each branch to focus on its task and improves the model's overall accuracy. In the output layer, the sigmoid function is used as the activation function for the objectness score, representing the probability that the bounding box contains an object, while softmax function:

$$\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \qquad C \geq 1$$

is applied for the class probabilities, representing the objects' probabilities belonging to each possible class.

3.2 U-Net

In this thesis, a custom U-Net to detect people's head has been implemented and trained. Using a UNet in crowd counting is not a new idea [34][35] , but here an original code based on PyTorch was developed and it is available on GitHub alongside with testing and documentation [64]. Among CNNs, U-Net has been chosen because it is particularly well-suited for tasks where precise localization is essential: as opposed to YOLO's bounding boxes, U-Net provides pixel-wise segmentation and it is thus more suited for detecting small, densily packed objects like heads in a crowd. Moreover, U-Net models are flexible with input's aspect ratio and are more practical for the available dataset of this project (see section 4.1 for details).

The U-Net architecture was initially proposed in 2015 by Ronneberger et al. as a solution to medical image segmentation problems [12], but it was quickly adopted for many other tasks. It has a unique structure that makes it particularly effective for tasks with high resolution inputs and outputs, like image segmentation, super resolution (up-scaling low resolution to high resolution images), or diffusion models (transforming Gaussian noise to newly generated images).

The U-Net model is based on a Fully Convolutional Neural Network (Fully CNN) [65] and consists of two main parts: an encoder (contracting) path and a decoder (expanding) path, which are connected through a bottleneck and skip connections.

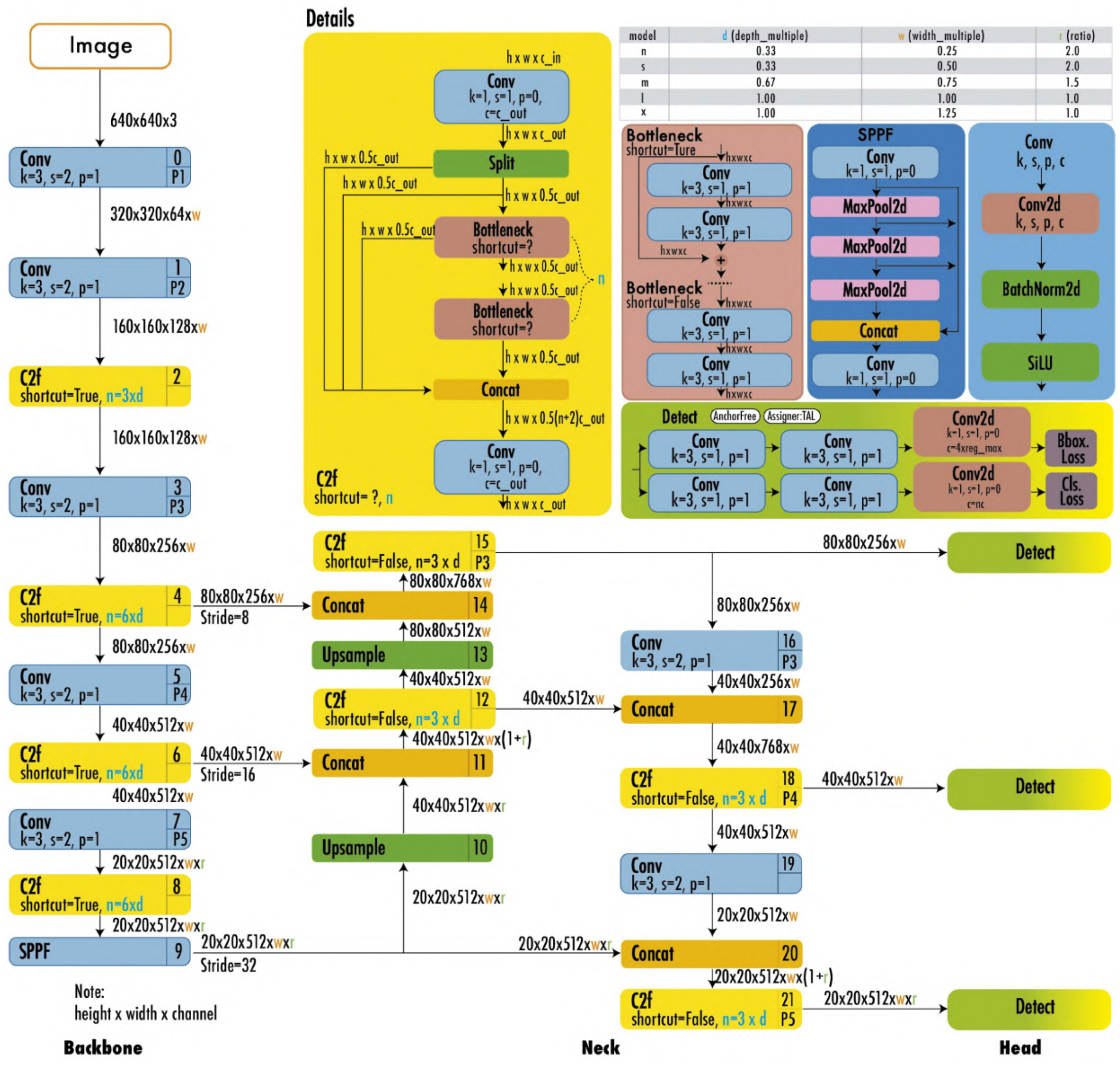


Figure 3.1: YOLOv8 structure. The model's convolutional modules include a Bath Normalization and SiLU activation function. These are alternated with C2f modules that combines high-levele features with contextual information to improve detection accuracy. Image source [10]

- The encoder is responsible for extracting features from the input image. It reduces the spatial dimensions of the image while increasing the number of channels. Every stage in the encoder consists indeed of two 3x3 convolutions, each followed by ReLU activation functions and a downsampling 2x2 max-pooling operator. As already mentioned, this last is the equivalent of picking the largest value in a non-overlapping window across the image, and in doing so the spatial dimensions are halved and the number of channels doubled. The Rectified Linear Unit (ReLU) activation function is used instead to introduce non-linearity into the network and it is defined as $\text{ReLU}(x) = \max(0, x)$

- The decoder is responsible for reconstructing the segmentation mask from the compressed feature representation produced by the encoder. Each stage in the decoder consists of upsampling operators followed by two 3x3 convolutions with ReLU activation functions. The upsampling operation is typically done through transposed convolutions and it restores the spatial resolution of the features that were lost during the encoding phase, refining the segmentation boundaries.

- The encoder and decoder are symmetrical and one of the most distinctive features of U-Net is the use of skip connections, that connect the encoder's to the corresponding decoder's layers. These connections directly transfer high-resolution features from the contracting path to the expanding path, helping the network to retain fine details that might otherwise be lost in the encoding process. To do that, each skip connection takes a copy of the features from the encoder and concatenates it onto its opposing stage in the decoder, meaning that subsequent convolutional layers can operate over both the decoder's and the encoder's feature.
  The intuition is that the decoded features might contain more semantic information (i.e., *in this region there is object x*), whereas the encoded features contain more spatial information (i.e., *these are the pixels where the object is*). Combining both the encoder's and the decoder's feature together, one can in theory achieve pixel-perfect segmentation.

- Finally, the bottleneck is the bridge between the intermediary features of the network, where the decoder "switches" into the decoder. In this phase, the features are downsampled with a 2x2 max pooling, then they are passed through a 3x3 convolutional layer with ReLU activation function and then they are upsampled again by a 2x2 transposed convolution.

This design gives the model its name: the U-Net. The structure can be better understood by looking at Fig. 3.2, which shows an example for a 4-layer U-Net, with 32x32 pixels in the lowest resolution step.

As already mentioned, in this thesis work, a general U-Net architecture has been implemented in Python using PyTorch (whereas the original implementation was in

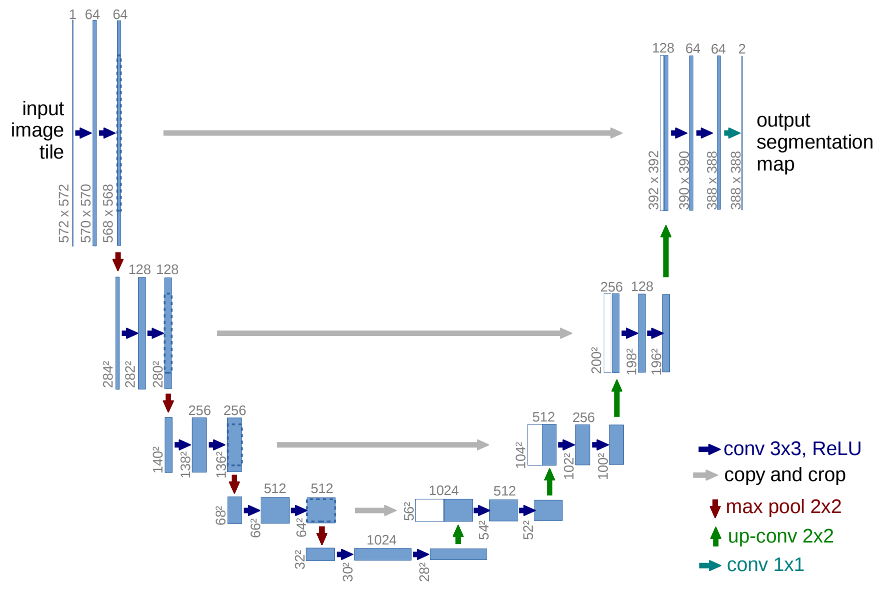


Figure 3.2: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. Image source: [12]

Caffe [66], a deep learning framework originally developed at University of California, Berkeley): the full code, together with testing and documentation, can be found on GitHub [64].

3.3 Traditional computer vision algorithm

Throughout this work, a custom algorithm that uses traditional computer vision techniques has been developed. This type of approach is simple, fast and doesn't require training nor GPU to perform real-time analysis. Its inference can also be performed in real-time from a Raspberry PI, as shown in section 4.6. The downsides are that the results are harder to interpret than those of the networks previously cited and it can only be applied in videos or known scenarios, since it relies on background subtraction. Also, still objects are usually not detected with this class of methods.

With *background subtraction* it is meant every technique which allows an image's foreground to be extracted for further processing. To achieve this, first the background is obtained, then a simple pixel-wise subtraction of it from the image, gives the foreground as the result. The background could be obtain in several ways [14][67][68][69], including calculating for each pixel the median or average value over a set of different (random) frames from a video. By doing this calculation for each of the 3 channels (red, green, blue) an RGB version of the background can also be obtained. Another method consists, starting from a blank background, in iteratively update it by performing a weighted sum between it and the current frame, so that $\boldsymbol{B}^{(t+1)} = \boldsymbol{B}^{(t)} + \alpha \boldsymbol{F}$, where $\boldsymbol{B}^{(t)}$ and $\boldsymbol{F}^{(t)}$ respectively indicate the background and current frame at discrete temporal step t. The weight value $\alpha < 1$ determines how fast the background is formed and changes: the greater the weight α, the more the background is dependent on the current frame and quickly changes with it [13].

The obtained foreground can then be processed to obtain useful data for crowd analysis: for instance, one of the simplest methods could be to apply a threshold and directly correlate the number of non-zero pixels with the number of people in the current frame.

3.4 Tracking people to estimate crowd density

In this section it is described an alternative method to estimate the crowd density which is based on tracking. The fundamental principle of this approach is based on the idea that the ability of people to move freely within a space is inversely proportional to the crowd's density. The core concept is to use few tracked individuals as "test particle" to infer the "permeability" of the crowd: if the crowd is sparse, individuals can move relatively freely, suggesting lower crowd density; on the other hand, if the individuals appear stationary or constrained in their movement, it implies that the crowd is more packed, indicating higher density. This can be quantitatively measured by tracking movement trajectories across video frames and calculating metrics such as average displacement and speed.

The advantage of this technique is that it doesn't require that all the people are detected and tracked, but few of them are enough. However, the accuracy of the tracking system (and thus of the detection) is crucial for it to work.

Trackers are usually designed to work with bounding boxes, but when dealing with high density crowds, individuals are usually very occluded and their appearances and bounding box's sizes vary substantially, making the standard tracking methods not suitable in this case. For instance, despite being simpler than other trackers, even SORT [47] uses both bounding box position and sizes.

Thus, a modified version of the SORT algorithm that exclusively utilizes point detections has been implemented, focusing solely on the detected positions of objects to solve the Multiple Object Tracking (MOT) problem. While this simpler approach can lead to inaccurate data associations (i.e., where one object may incorrectly be assigned the identity of another between frames), this error is less significant when estimating crowd density, which does not strictly require precise tracking of each individual.

Indeed, if the tracking algorithm incorrectly associates one person with another between consecutive frames due to crowd congestion, this is actually indicative of a dense environment, since in such cases, the resulting trajectories will be short, overlapping, or irregular, which reflects high density. In contrast, when the crowd is sparse and there are few people, the tracking system, even if so simple, will correctly associate the peoples's position across frames, leading to a faster, straighter and smoother trajectory.

Using this reasoning, the tracking ability to handle dense scenes becomes less about accuracy in person-to-person associations and more about the overall representation of movement patterns, making this simplified approach very practical and effective for crowd density estimation.

Examples of this novel methods are later shown in section 4.5, where YOLO tracking method is used and trajectories drawn.

Chapter 4

Application and results

Here, the models and algorithms described in the previous chapter are applied to crowd images and videos in the context of Venice, Italy and the results are discussed and compared. The goal is to assess the performances of such models in tasks like crowd density estimation, crowd counting and flow reconstruction.

4.1 The dataset

Over the years, the City Lab of University of Bologna has collected data from cameras in Venice.

The dataset used in this work is composed of:

- 375 6016×4000 images, each matched with a list of coordinates for the people's head. Examples of those images can be seen in Fig. 4.1. The crowd density in those images varies substantially, and the people count goes from a minimum of 18 people to a maximum of 3975. The location, camera angle, zoom level and field of view are also variable, making the labelled dataset diversified.
- 35 FullHD (1920×1080), 5 minute videos at 3 fps, taken from three different cameras placed in alleys; frames of which can be seen in Fig. 4.2
- 10 low-quality (768×576), 5 minute videos at 8 fps, from a camera in Piazza San Marco. Frames visible in Fig. 4.3.

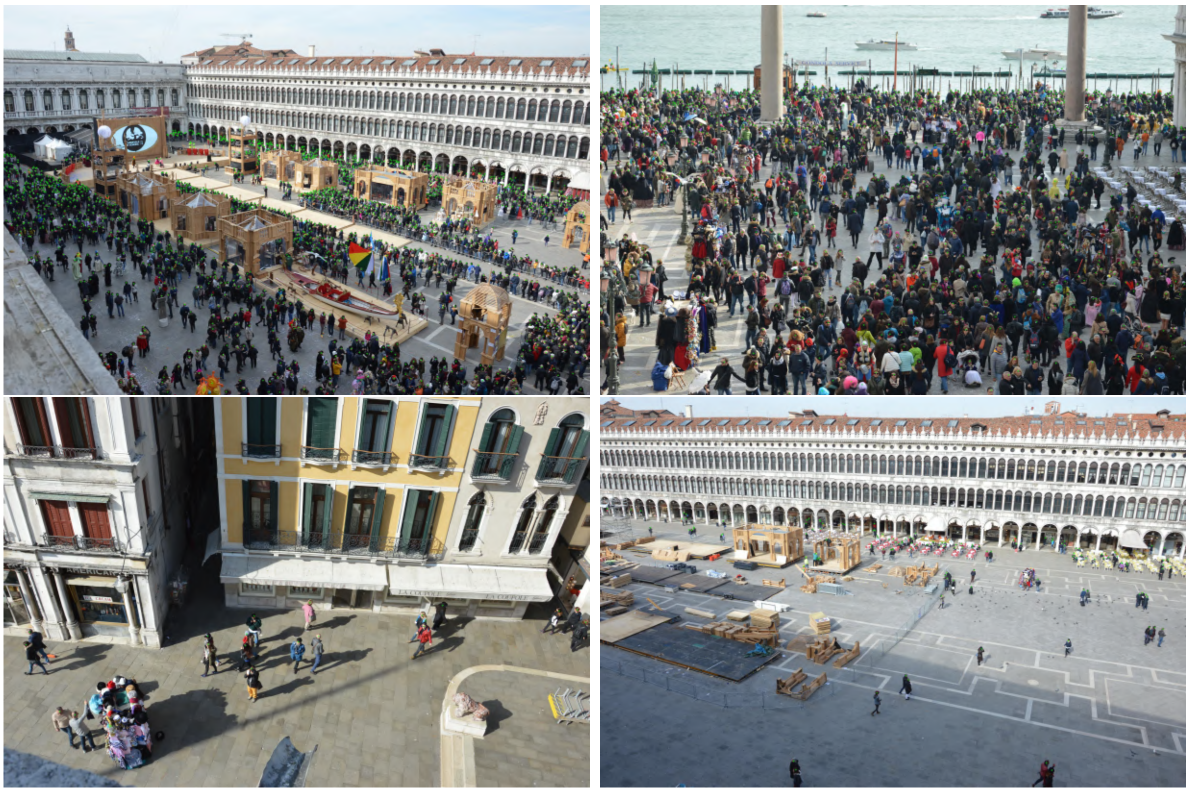


Figure 4.1: Examples from the labelled dataset. Green marks have been drawn at the right coordinates to represent the label.

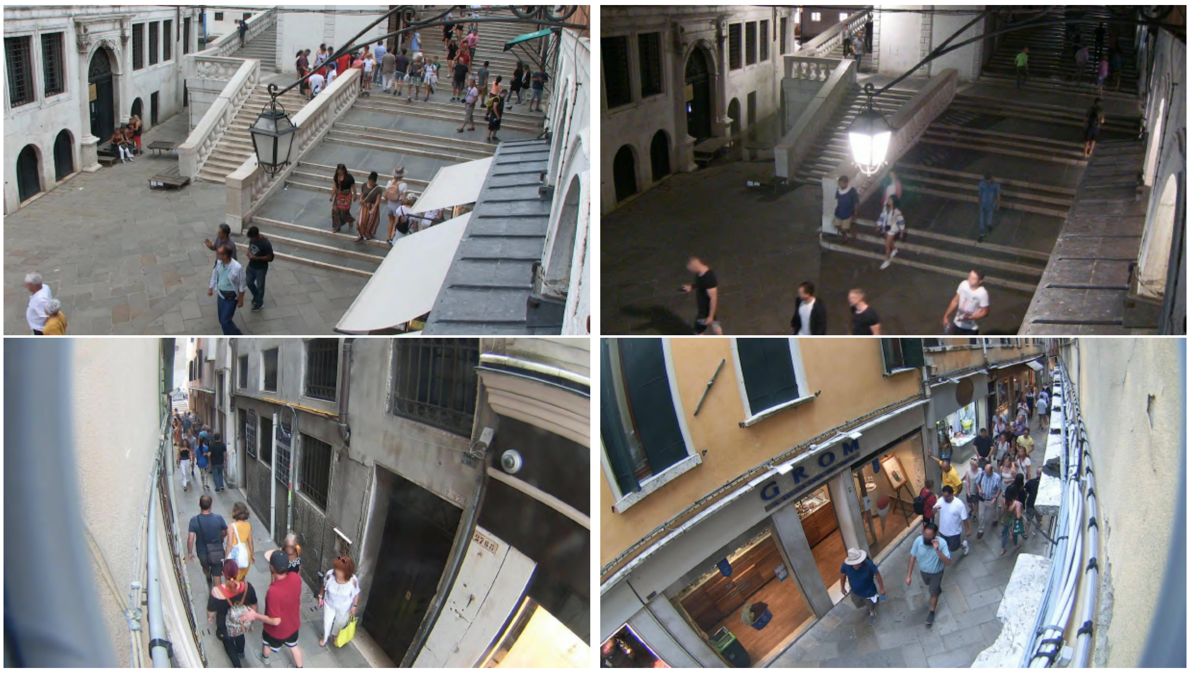


Figure 4.2: Examples of frames from the video dataset. All the videos are 5 minutes long, taken at different time of the day.

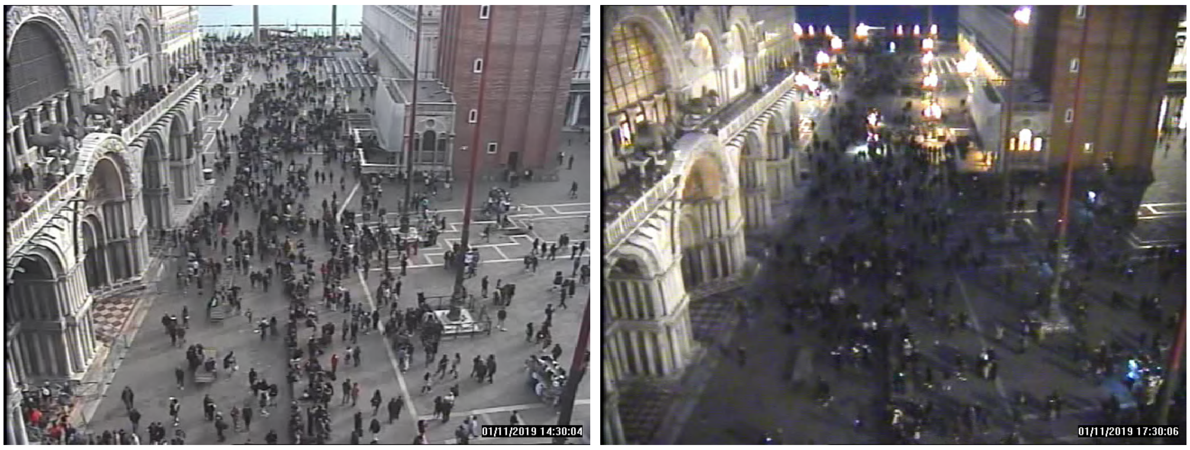


Figure 4.3: Frame examples from low-quality videos of Piazza San Marco.

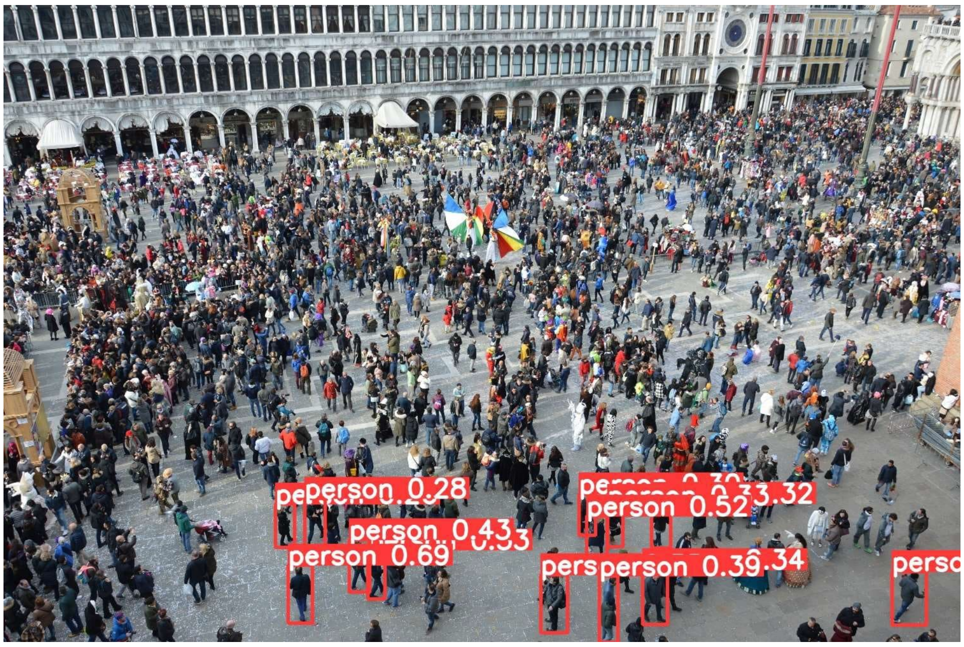


Figure 4.4: Example of pre-trained YOLO detection inference on an image of the dataset. It is clearly visible how the model works only in low crowd density conditions.

4.2 YOLO

The version of YOLO `YOLO8x`, pre-trained on COCO dataset [33] was tested both for detection on images and for tracking on videos. Example of results can be seen in Fig. 4.4 and Fig. 4.5 that shows detections and trackings respectively. Clearly, YOLO only works in low-density crowd scenarios, which is expected, since the model has been trained on detecting and classifying the whole body while in a high-density scenario, people are usually severely occluded and only the upper part of the body is visible.

In Fig. 4.5, it is important to notice that, although not all the people are detected, the ones correctly identified always maintain a unique ID. This allows to reconstruct partial trajectories for some of people, which will be a useful information to estimate crowd density as described in section 3.4. Also, it is true that people are only recognized when they are closer to the camera, but the great majority of them are detected and tracked with no false positive error. This means that in a moderate density situation like the one in Fig. 4.5, it is possible and easy to count the number of people passing by in an alley. Note however that it is difficult to quantify the correctness of the counting for this study case, since no labels for videos are available.

The same argument could be applied as well for the other videos of alleys, even at night time. Some frames of the tracking results on those videos can be seen in Fig. 4.6 and Fig. 4.7.

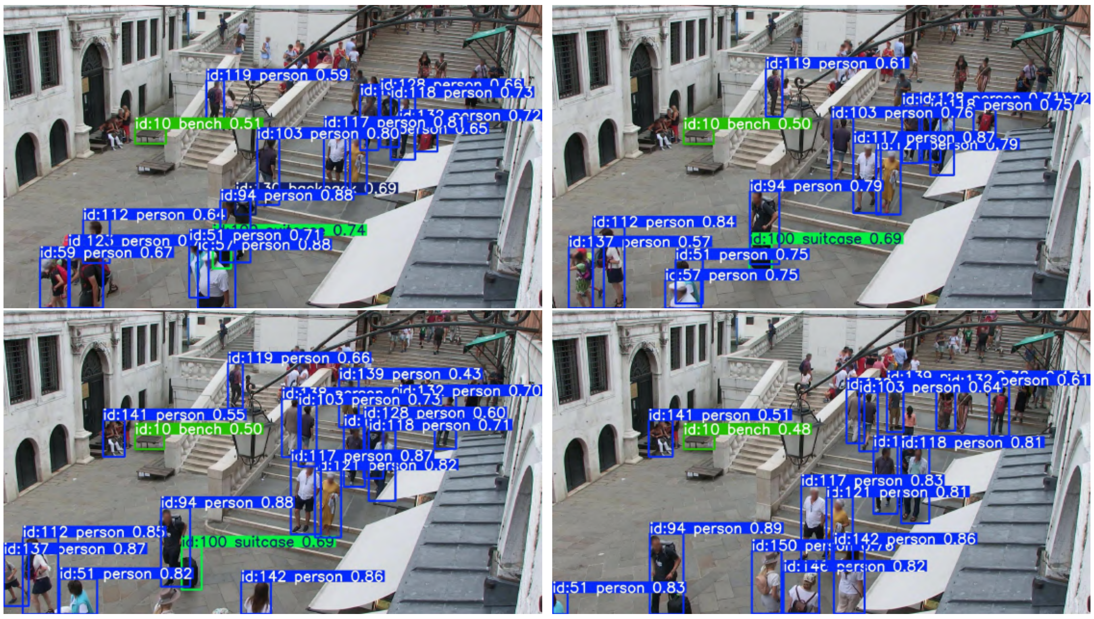


Figure 4.5: Example of pre-trained YOLO tracking inference on four frames of a video from the dataset. The model only detect less occluded people, but keep them tracked with the same ID. The images are 6 frames (2 seconds) apart.

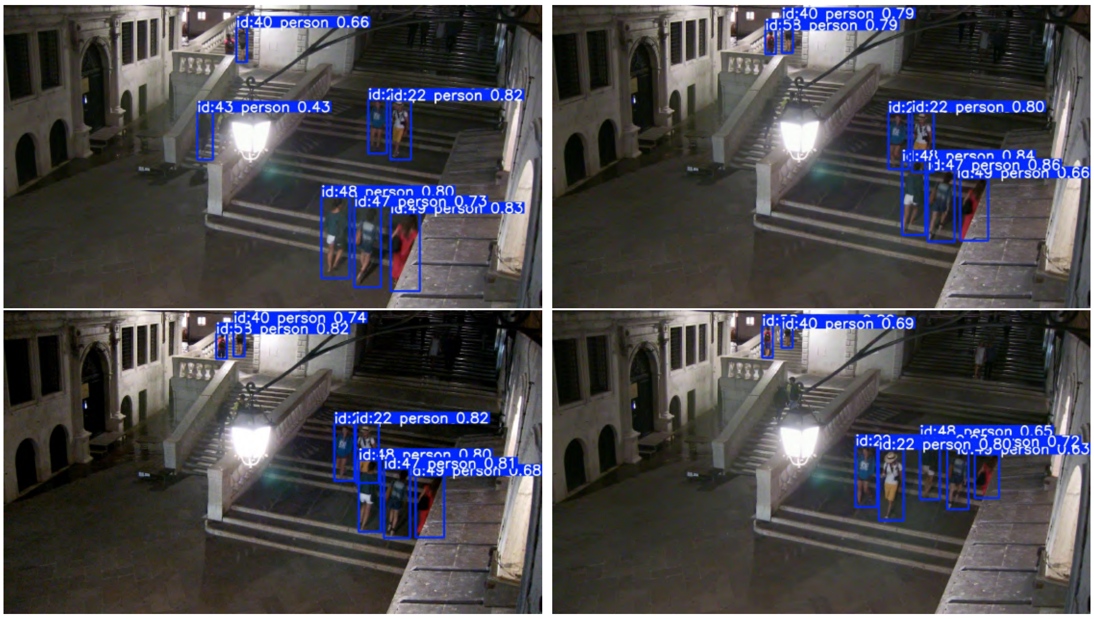


Figure 4.6: Example of pre-trained YOLO tracking inference on four frames of a video shot at night time. The model only detect the most illuminated people on the foreground, but surprisingly keep them tracked with the same ID. The images are 6 frames (2 seconds) apart.

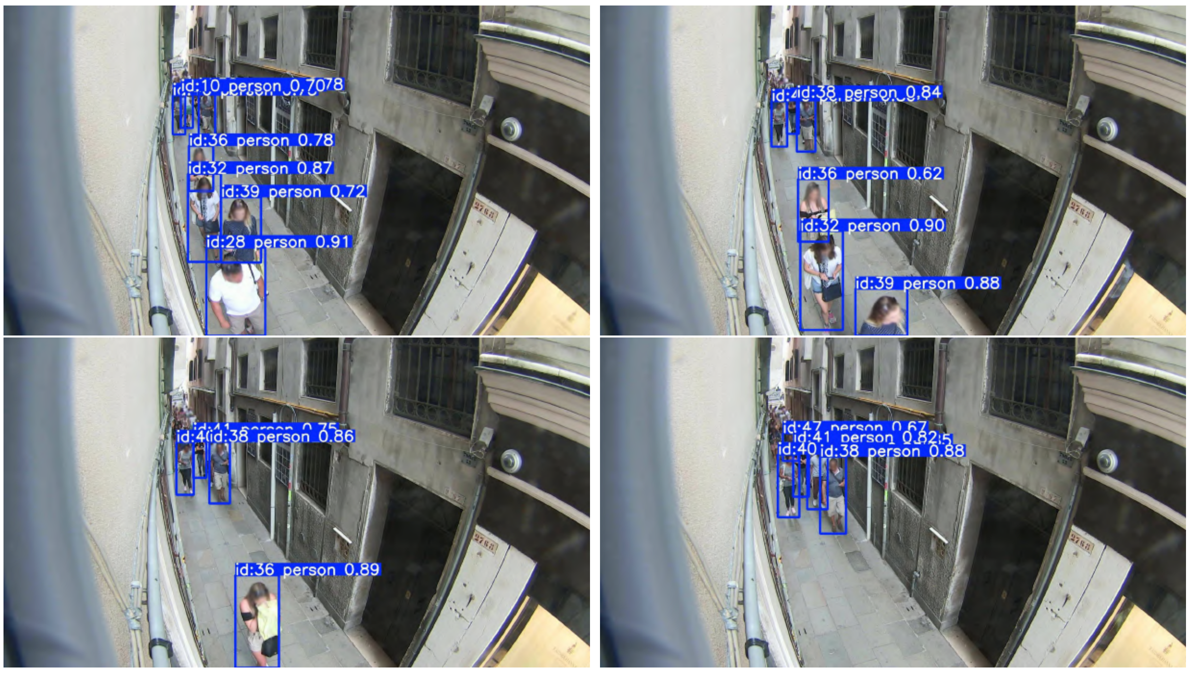


Figure 4.7: Example of pre-trained YOLO tracking inference on four frames of a video from the dataset. The model only detect the people on the front, but keep them tracked with the same ID and counting them is thus feasible and easy. The images are 6 frames (2 seconds) apart.

Regarding the images of Piazza San Marco, the higher crowd density and the great number of access points to the square make the previous logic inapplicable, so a version of YOLO was trained to detect the people's head with the aim to etimate their density and number.

4.2.1 Training YOLO

YOLO is designed to work with bounding boxes, while the available labels in the dataset consist only in coordinates of the heads' center. An attempt to train YOLO has been made anyway, by creating bounding boxes of a fixed size of 30px around the given coordinates. The labelled dataset was divided into a train set (265 images, $\sim 70\%$) and a validation set (110 images, $\sim 30\%$). Due to the relatively small number of images but their large size, data augmentation techniques[70][71][72] were involved. For the training set these consist of:

- a random horizontal flip with a probability of 0.5;
- a random small rotation in a range of ± 5 degrees;
- a random crop of the image, that takes a random portion with an area between 0.1 and 1.0 times the area of the image. This transformation is followed by a resizing step to ensure that all images fed to the model have the same size (256×256 pixels)

Instead, the transformations on the validation set should not contain any randomness in order to ensure consistent evaluation results, so the validation images were just subjected to a center crop of a fixed size (2048×2048 i.e., 8 times the 256×256 final size) followed by a resizing to the same 256×256 pixels as the training images. Selecting the center of the images not only avoid any randomness, but also makes the central area assume more importance, and the model results better for that zone.

YOLOv8 was then trained for 100 epochs and the results are shown in Fig. 4.8, where train and validation losses are plotted, alongside with precision and recall metrics (see Appendix A.1 for the definition). Unfortunately, even if precision and recall seem to gradually increase, and the training loss to slowly decrease, validation loss stays high throughout the training. This indicates that the model is not able to generalize to unseen data. Indeed, even when applied to the same images used in training, no detections at all are performed, rendering the trained model completely useless. Therefore, no further experimentation or usage of the model have been conducted, as its inability to perform even on familiar data renders it unreliable for any practical purpose.

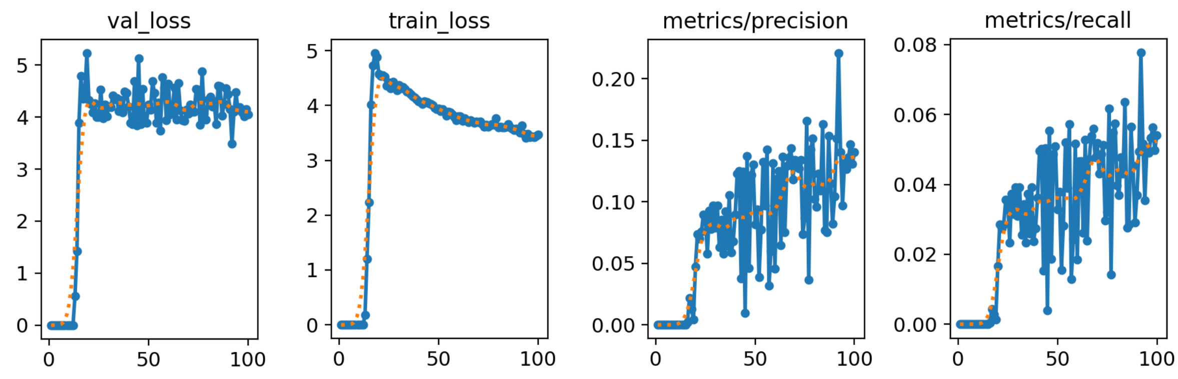


Figure 4.8: Results of YOLO training with the aim to detect people's head. From left to write, the plots show respectively validation loss value, training loss value, precision and recall plotted against the 100 training epochs. The validation loss maintaining an high value is an indication of poor generalization of the model, which indeed can't predict correct bounding boxes.

4.3 U-Net

To train a U-Net, images and corresponding segmentation masks are needed. As already mentioned, the available labelled dataset for this project consists in 375 6016 $\times$ 4000px images, each matched with a list of coordinates indicating the position of people's head. In order to use the U-Net on such a dataset, binary masks have been created, with a circle centered on each (x, y) pair of coordinates.

In theory, the radius of each circle should vary according to the size of the head it represents. In particular, people further away have a smaller apparent dimension than the ones on the foreground. There are two easy way to account for the variable distance between the camera and people:

- using two cameras with a known distance between the two, a binocular vision approach could be applied [13][1];
- if the the geometry of the place is known, a perspective correction can be used in order to get a rough 2D reconstruction from a single point of view (see Appendix B.2 for details).

Unfortunately, given the the use of a single camera and the different locations and framing of the photos, neither of the two methods could be used. Therefore, another approach has been attempted: after detecting edges in each image using a Canny edge detection [73] (as implemented in OpencCV [74]), small circles was expanded until they hit an edge. In practice however, the results of this operation (Fig. 4.9b) are not really better

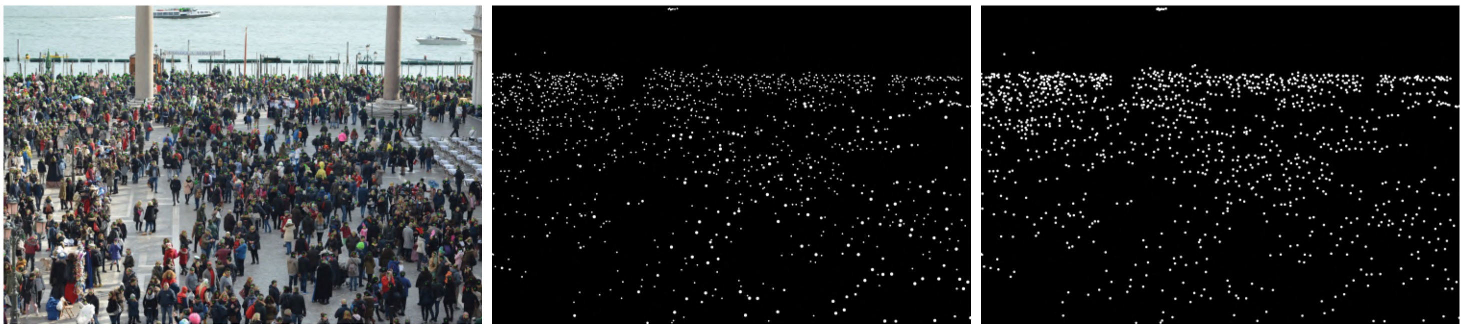

(a) (b) (c)

Figure 4.9: Visual comparison between masks created using expanding circles and simple circles with a fixed radius for an example image: **(a)** original image with green marks as label; **(b)** binary mask created by expanding little circles until they hit an edge (detected with Canny edge detector); **(c)** binary mask formed by circles with a fixed radius value (15px).

then setting a single value for the radius of all circles (Fig. 4.9c) and the difference in circles' radius seems to be random and uncorrelated with the actual head dimension. This is because even if it's true that nearer people have apparently bigger heads, the foreground portion of the image is also more detailed and this make so that more edges are detected in that area.

Thus, the same value of 15 pixels has been set for each circle and binary masks have been created. These are what it was then used to train an instance of the U-Net previously described.

4.3.1 Training the U-Net

A U-Net with 4 layers, with 64, 128, 256, 512 channels respectively has been used to perform the segmentation of people's head. As for the training of YOLO model, the labelled dataset was divided into a train set and a validation set and the same data augmentation techniques were performed. The model was then trained for 1000 epochs. At each epoch, the training and validation losses have been calculated. The chosen loss function is the `torch.nn.BCEWithLogitsLoss` [75]: this combines a Sigmoid and a Binary Cross Entropy (BCE) Loss in one single layer and can be described as:

$$l(x,y) = \{l_1 \dots l_N\}^T \quad , \quad l_n = -\omega_n \left[y_n \cdot \log \sigma(x_n) + (1-y_n) \cdot \log(1-\sigma(x_n))\right]$$

where N is the batch size.

In addition to monitoring the losses, on the validation set the accuracy and Dice score [76][77] were calculated as well. Accuracy is just the ratio between the number of correctly predicted pixels and the total number of pixels in the image, while Dice score

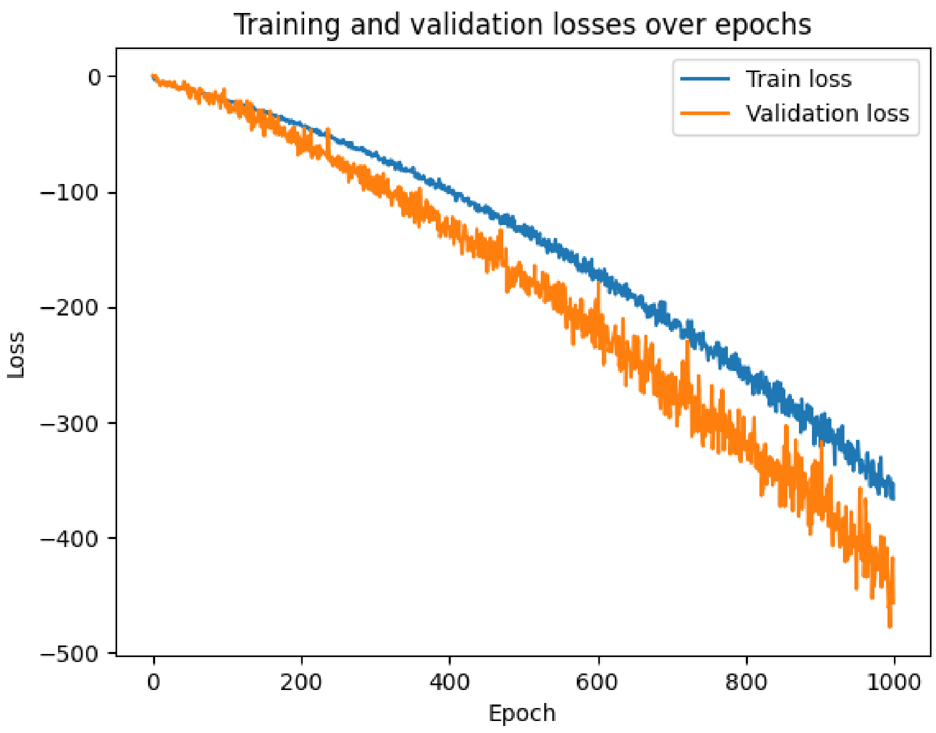
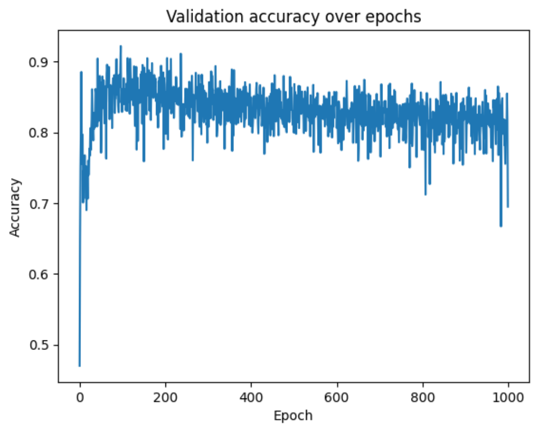


Figure 4.10: Plot of the training and validation losses of the custom U-Net. The losses values are plotted over the epochs. Both of them decreasing indicates that the model is learning effectively without overfitting.

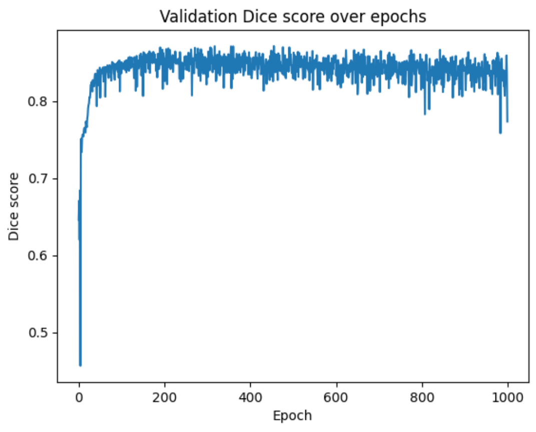
(also called F1-score) is the harmonic mean of Precision and Recall. See Appendix A.1 for definition's details.

Fig. 4.10 shows the loss curves over the 1000 epochs, illustrating the model's performance on the training and validation sets. Both the training and validation losses exhibit a descending trend as the number of epochs increases, indicating that the model is learning effectively. The decrease in training loss reflects the model's capacity to fit the training data, while the reduction in validation loss demonstrates the model's generalization ability to unseen data.

In Fig. 4.11 one can see plots of accuracy and Dice score over the validation set. Both metrics rapidly approach their maximum within the first ~ 150 epochs and remain consistently high throughout the training process, even if with slight fluctuations in later epochs.



(a)



(b)

Figure 4.11: Plot of two metrics on the validation set over the 1000 epochs of training: **(a)** Accuracy; **(b)** Dice score. Both metrics are better explained and defined in Appendix A.1.

From these two images, a continuous decrease in both the training and validation losses can be observed, while the accuracy and Dice scores quickly reach high values and stabilized within the first epochs. This discrepancy between the behavior of the losses and the one of the metrics can be explained by the differences in how they capture model performance. In fact, the loss function is sensitive to small errors in prediction boundaries: as the model trains, it continues to refine its predictions, resulting in a steady decrease in loss. However, accuracy and Dice score focus on more global aspects of prediction correctness, and they quickly stabilize once the model captures the major structures in the data. These metrics are less sensitive to minor improvements in the prediction boundaries, which explains why they plateau early while the losses continues to decrease.

This hypothesis is also supported by a visual comparison of the model results: Fig. 4.12, 4.13 and 4.14 show the model's prediction on a batch from the validation set at different epochs. The model refines the prediction for each epoch, but it is *generally correct* since the early ones. The slight decrease of accuracy and the greater fluctuations in both the accuracy and the Dice score in Fig. 4.11 can also be justified: it's easy to see how in the last epochs, each head gets detected slightly better (thus the losses decrease), but the overall prediction is noisier, making the prediction less *accurate*.

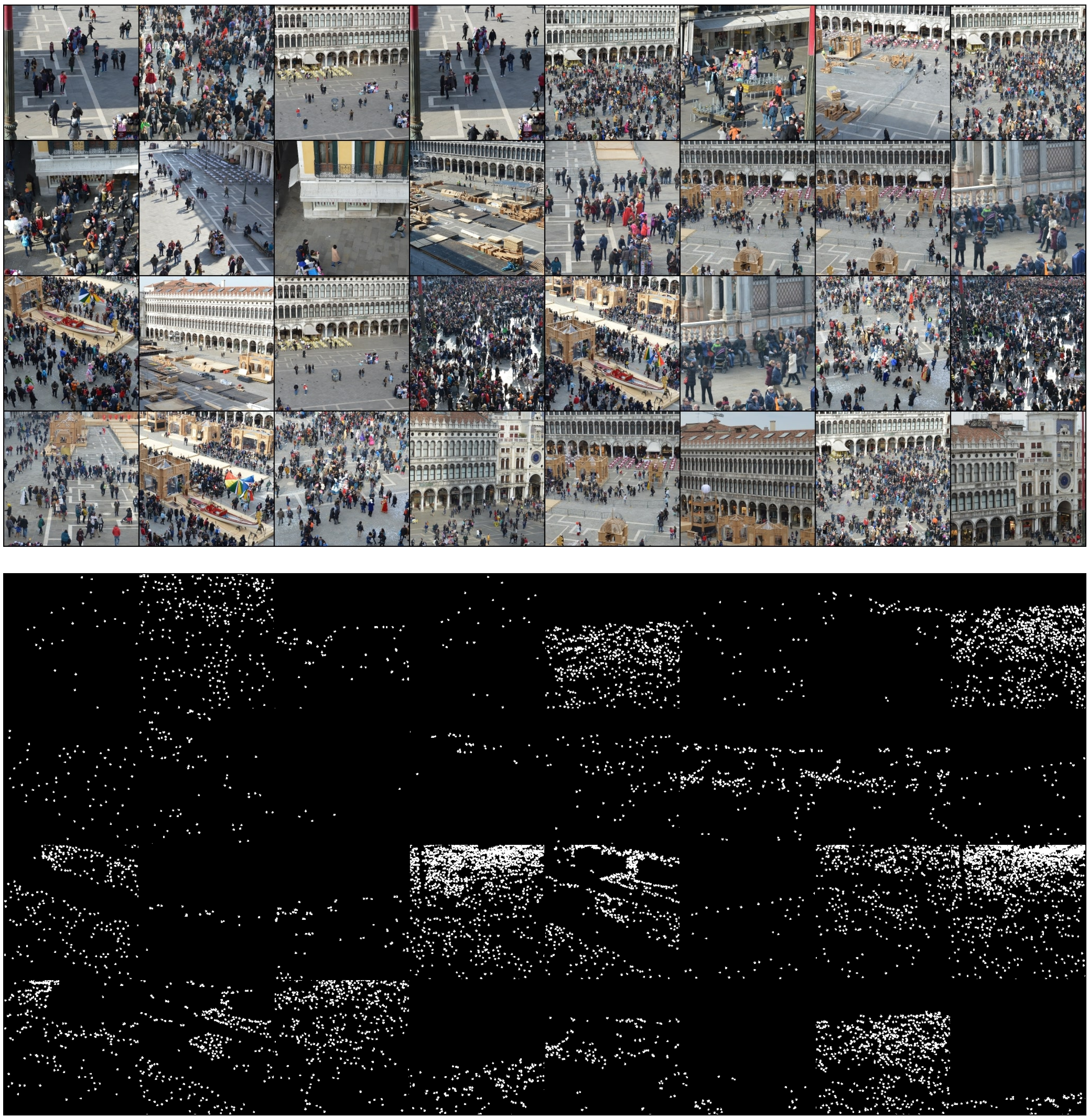
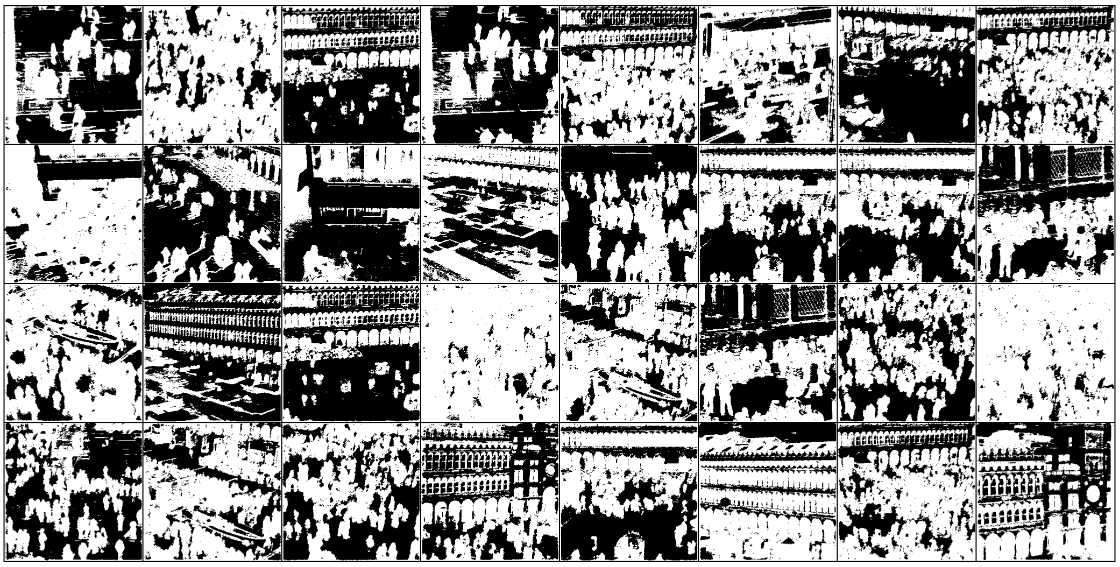
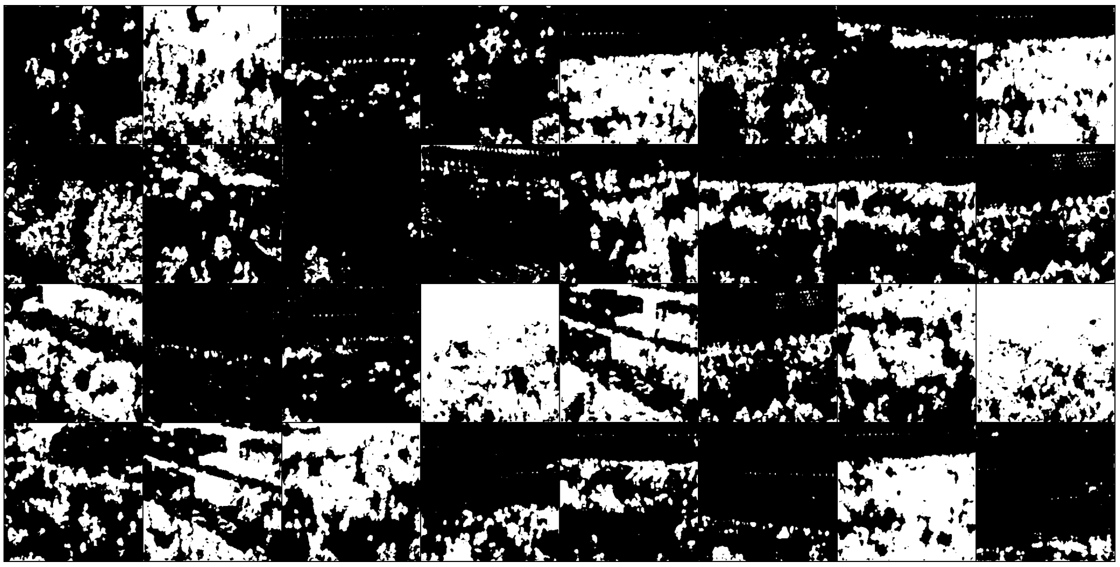


Figure 4.12: Example of a batch from the validation dataset and the corresponding binary mask consisting of a circle around head head.


(a) Prediction at epoch 0


(b) Prediction at epoch 50

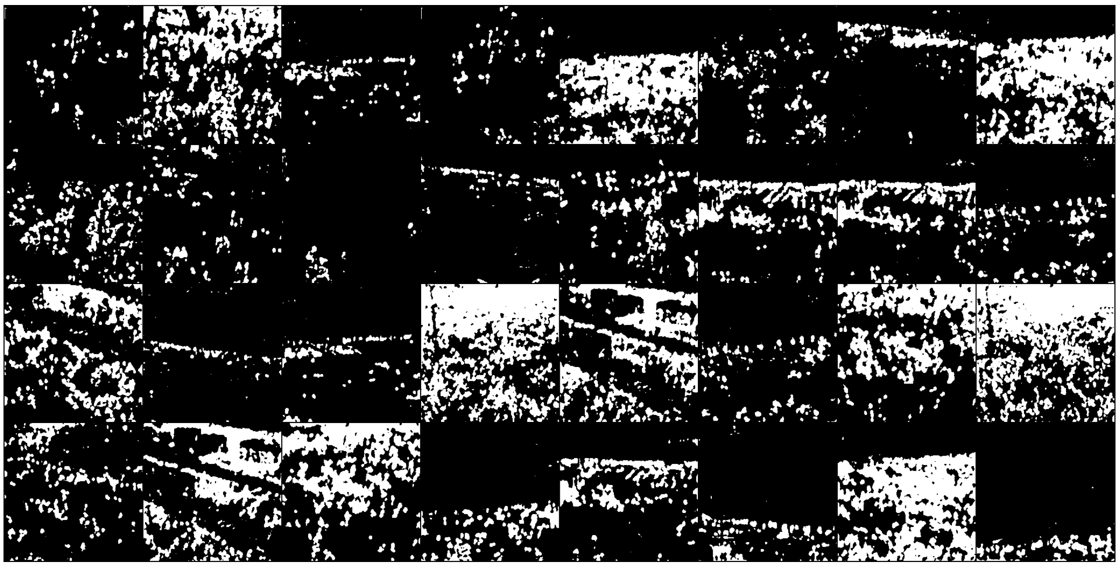
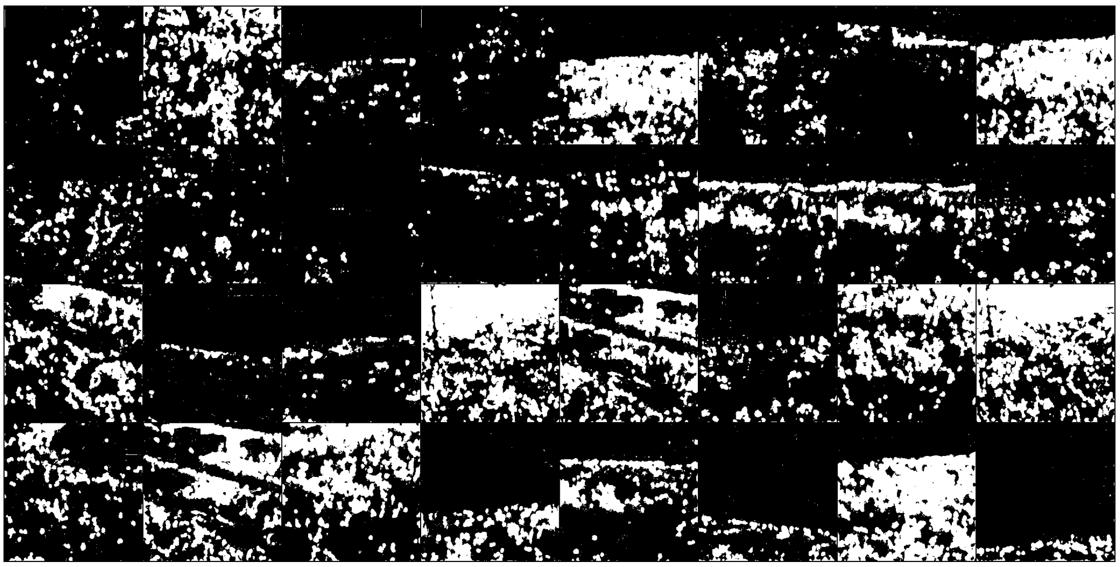
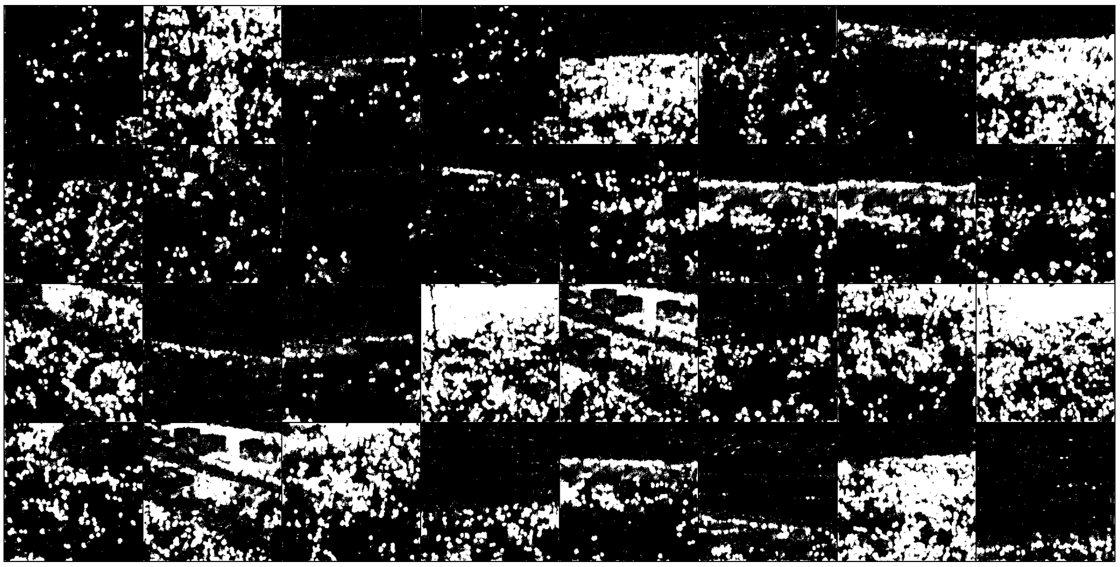

(c) Prediction at epoch 100

Figure 4.13: (1 of 2) Prediction of the model on the batch from Fig. 4.12 at different training stage. It is clear how the general *correctness* of the model is present since early epochs, but the model continue to refine the prediction.


(a) Prediction at epoch 200


(b) Prediction at epoch 400

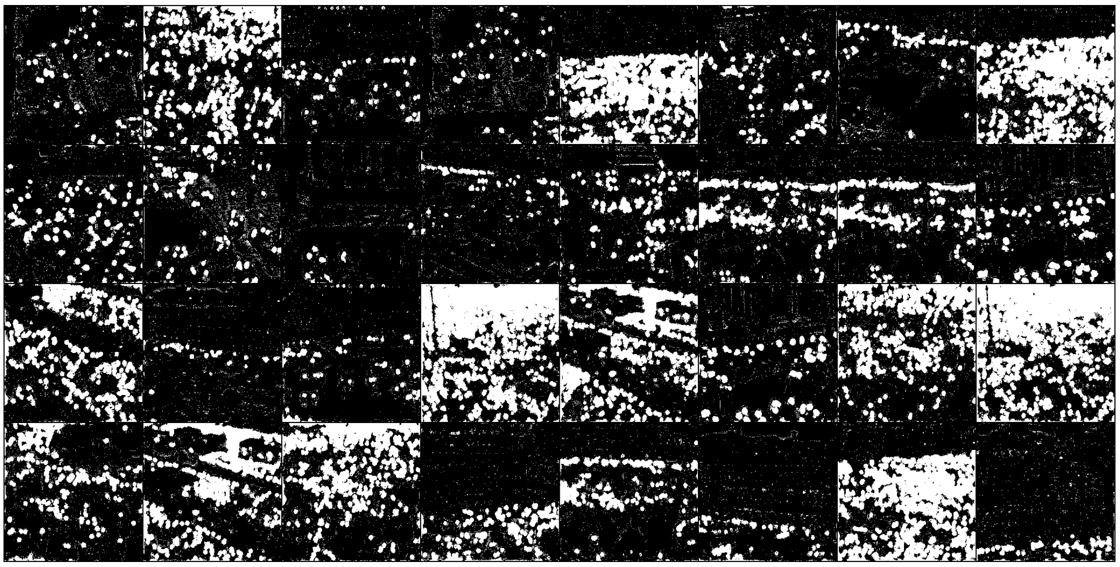

(c) Prediction at epoch 1000

Figure 4.14: (2 of 2) Prediction of the model on the batch from Fig. 4.12 at different training stage. It is clear how the general *correctness* of the model is present since early epochs, but the model continue to refine the prediction.

In general, which is the stage of the training to use on the inference process strictly depends on the model's application: using the early weights the model is less accurate but also less noisy and could be better for instance as a crowd density estimator, while a model which is more accurate in defining the segmentation borders can be used for crowd counting or in individual dynamic reconstruction, where it is more important to distinguish and divide the people. Moreover, as an idea for future development, more than one weight can be employed and a combination of their predictions be used as the actual result. For instance a weighted sum of the result of different model's weight in a given pixel can represent the confidence level of such model. Of course, this results in longer inference time and it was thus considered to be out of the scope of this work.

The trained U-Net model was then used to perform inference on other images. Among the different model weights, the ones obtain after 700 epochs visually seemed to perform the best on unseen data (namely, frames of the unlabelled videos), balancing between segmentation precision and noise. The following discussion is thus based on these specific model weights.

First, the model was re-applied to training and validation datasets, this time without cropping the images. This was done as a further test on labelled dataset, and also to get the prediction for the non-central part of the images as well. Processing the images with no resizing would lead to bigger computational times and make the model detect smaller details than heads, (example in Fig. 4.15). On the other hand, directly resizing the 6016×4000 images to a small resolutions like 256×256 would certainly induce a loss of details and wouldn't maintain the aspect ratio, making the model's predictions ineffective. A good solution is something in the middle: each image was resized to 1/8 of its original size (just like for the validation set during training) and then a sliding window method could be applied: each resized image would be divided into 256×256 blocks and the model would then be applied on each of them. However, this would increase the computational time and the results of the model directly applied on the resized images were considered good enough for the subsequent analysis, so the following figures and discussion are based on of the model's application over 752×500 images. In Fig. 4.16 it is shown a visual comparison between a model's thresholded prediction over an image from the validation set during training (i.e., the prediction over the cropped image) and over the same image but without cropping. The output of the model over the non-cropped images has thinner and noisier prediction, which is expected due to the larger dimension of the image where it was applied. The noise can be easily removed using erosion (see Appendix B.1 for explanation) and the result of this operation is shown in Fig. 4.17.

In Fig. 4.18 the non-thresholded predictions are instead directly drawn over frames of different video using a color map scale. It is visible how the model makes wrong predictions, in those scenarios very different from the images used for training. For the videos where the predictions are acceptable (e.g., upper-right in Fig. 4.18), those are then converted to a binary image with a simple threshold and eroded to remove noise.

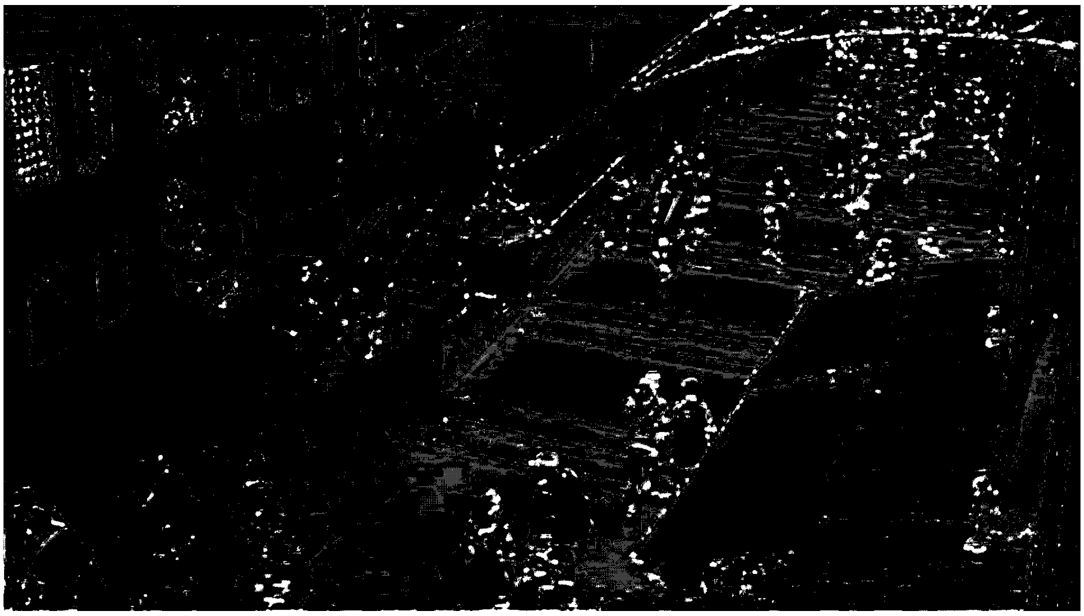


Figure 4.15: Result of the application of the U-Net trained for 700 epochs on a non-resized image. The model incorrectly detects smaller features than heads, like edges of people and objects.

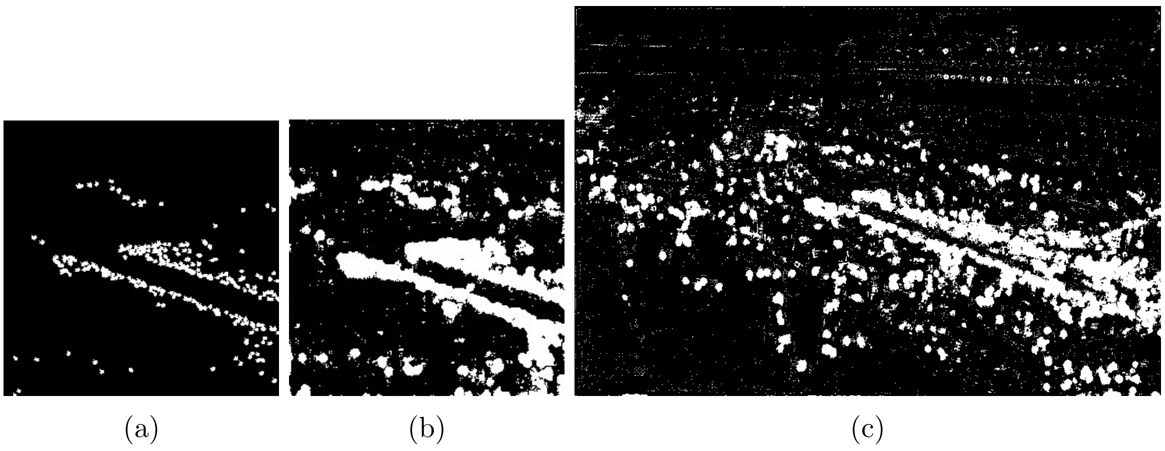


Figure 4.16: Difference between the U-Net output applied on the same image from the validation set during training (with a central crop) and after training (without). As expected, the larger image is has thinner and noisier prediction than the original cropped one. **(a)** Cropped binary mask; **(b)** results on the cropped image obtained during training; **(c)** re-application of the model over the full image.

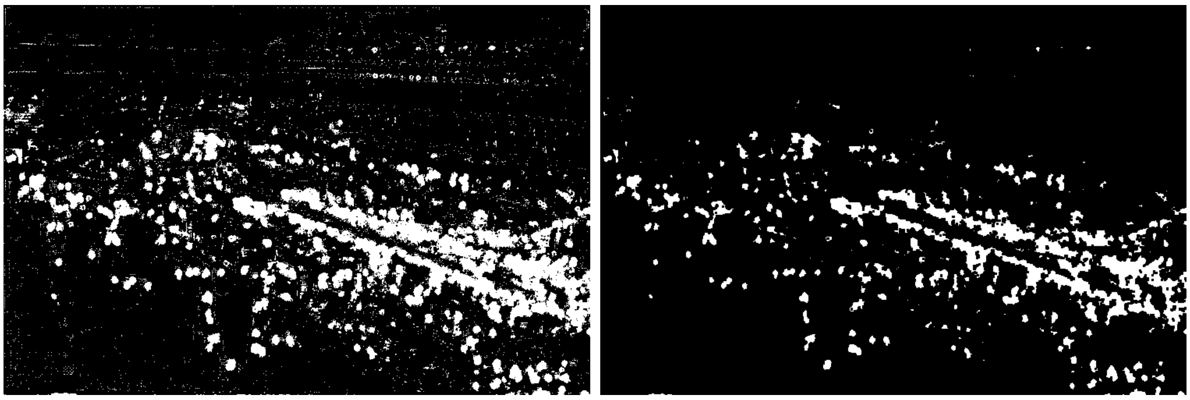


Figure 4.17: Noise removal using erosion for the example image of Fig. 4.16c.

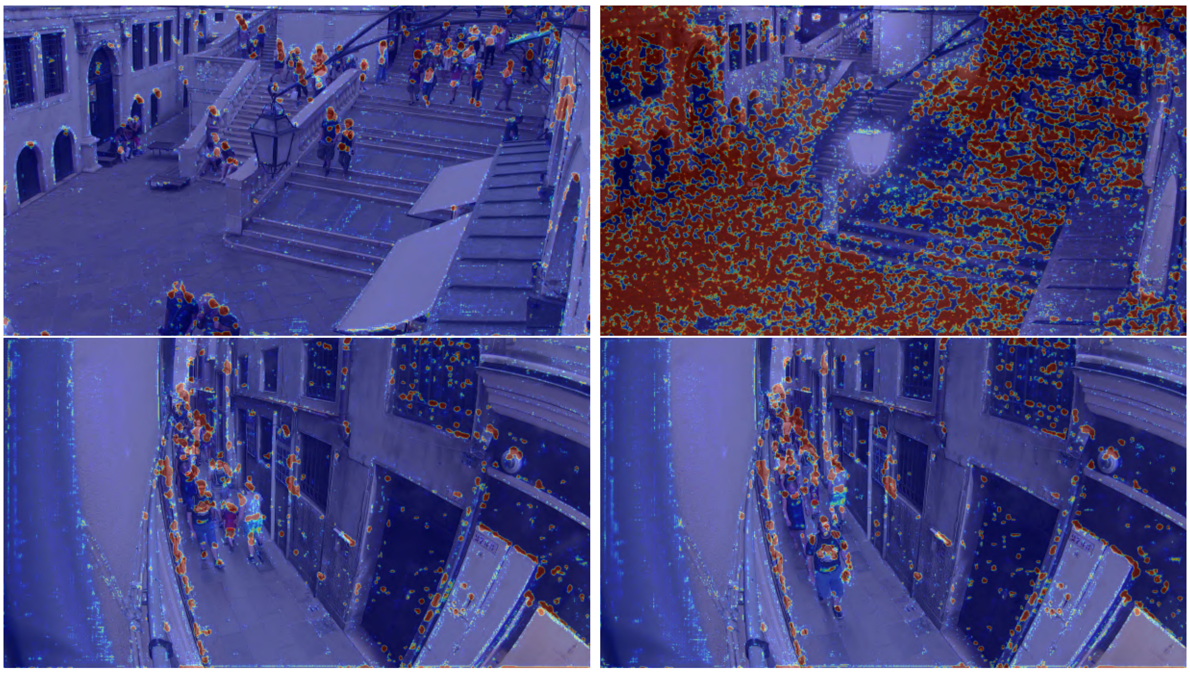


Figure 4.18: Example of U-Net inference on frames of different videos. No threshold is applied on the model's result. Many false positive error are visible, probably due to the different scenario between the videos and the images from the dataset. In particular, the image on the upper right is from a night-time video and the bottom ones are from a camera angle much closer than the one of the images used for training. The video whose frame is on the upper left shows instead good results, with few false positive errors.

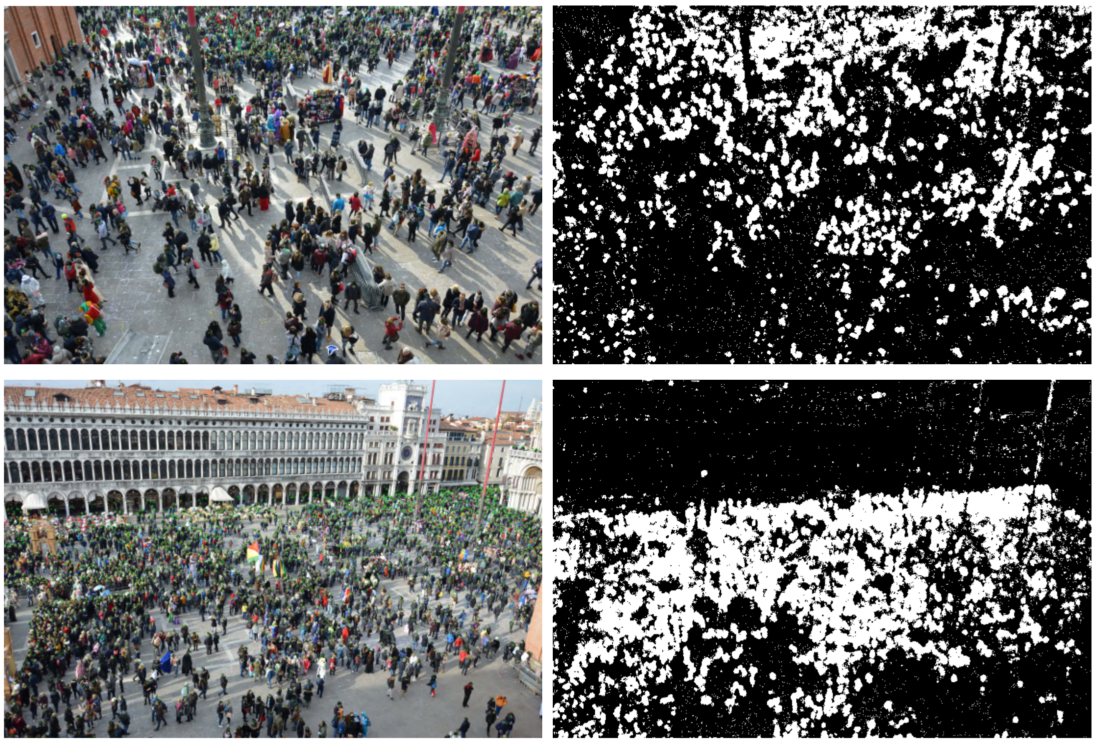


Figure 4.19: Example of two images and corresponding U-Net inference that present a noticeable difference in framing, camera inclination and zoom level.

4.3.2 Crowd density estimation using the U-Net

A first estimate of the crowd density can be made simply by creating a correspondence between the number of non-zero pixels in the U-Net output and the number of people in the scene. However, calibrating such a correspondence is not a simple task when the location and framing of each photo is different: the portion of the image that represent the walkable area, the distance between people and the camera and its inclination are for instance some of the factors to take into account. An example is shown in Fig. 4.19, which shows two images with a noticeable difference in framing (with or without buildings), camera angle and zoom level. However, once the camera's position and zoom level are set, the walkable space can be defined, perspective correction optionally performed and the cited correlation easily established. It is in fact clear how denser areas produce a greater concentration of non-zero pixels.

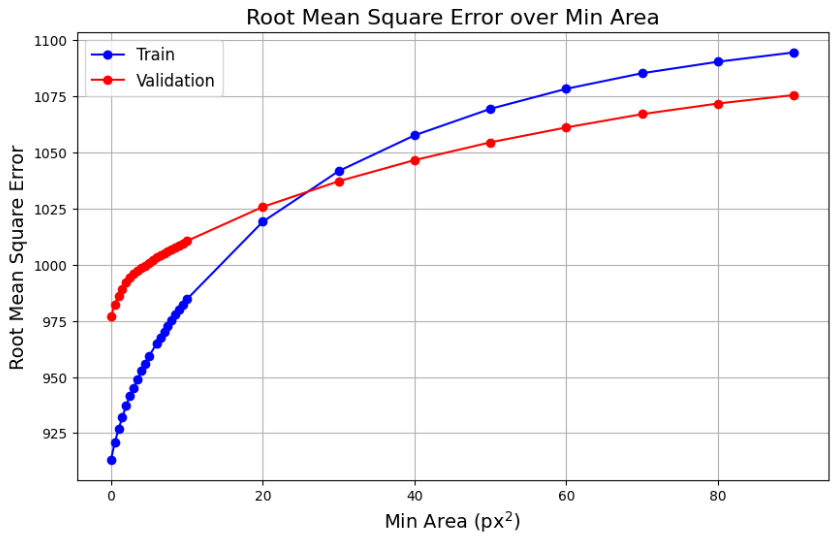


Figure 4.20: Root Mean Square Error (RMSE) between predicted and true count for both the training and validation set. The number of predicted counts was obtained by counting the contours whose area is greater than A_{min} and the RMSE is dependent on this threshold value.

4.3.3 Crowd counting and detection using the U-Net

From the eroded images, the number of people can be evaluated, performing the so-called *crowd counting*. The simplest method is to detect and count the borders in the binary image [13][14]. To account for eventual noise still present after the erosion, only the contours with an area bigger than a certain threshold A_{min} is considered. This approach was applied to both the training and the validation sets and the number of people was calculated for different values of the threshold A_{min}. The mean square error (MSE) between the predicted count and the true one was calculated and its square root (RMSE) is plotted in Fig. 4.20. This also served to empirically obtain the optimal value of the threshold and it is clear that it equals to zero. The reason can be understood by looking at Fig. 4.21 which plots the predicted count against the true count for $A_{min} = 0$: it is obvious that the number of people tends to approximately be the same, even for a larger crowd. This is because, while this approach of counting borders is very simple and fast, if two circles (i.e., two detected heads) touch even for a single pixel, they are considered as a single contour and counted only once. So, $A_{min} = 0$ is the value that minimizes the MSE simply because it maximizes the number of detections. Indeed in Fig. 4.22, where it is shown the image with the largest crowd and the corresponding U-Net prediction, it is clear that many circles can be identified, but they will be considered as a single contour.

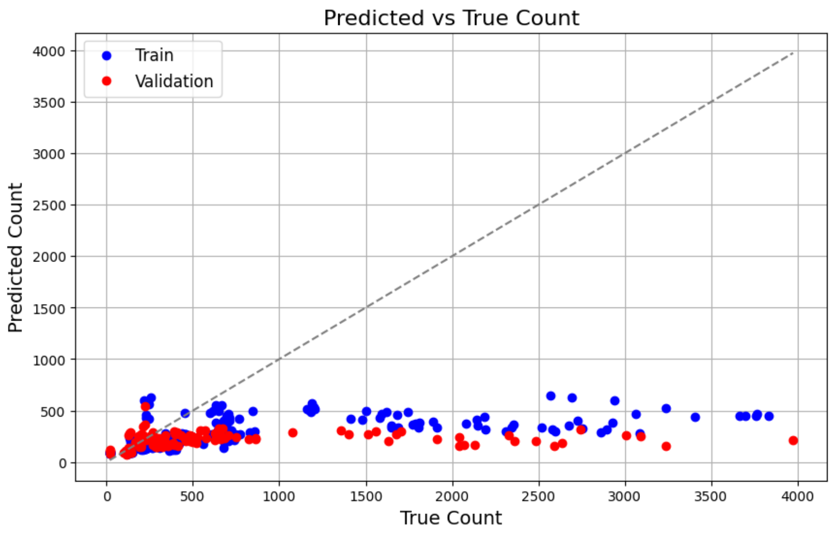


Figure 4.21: Predicted count over true count for both the training and validation set. The predicted count was obtained by counting the number of contours whose area is greater than $A_{min} = 0$ (i.e., considering all contours). The predicted count tends to be approximately the same regardless of the actual number of people present in the image.

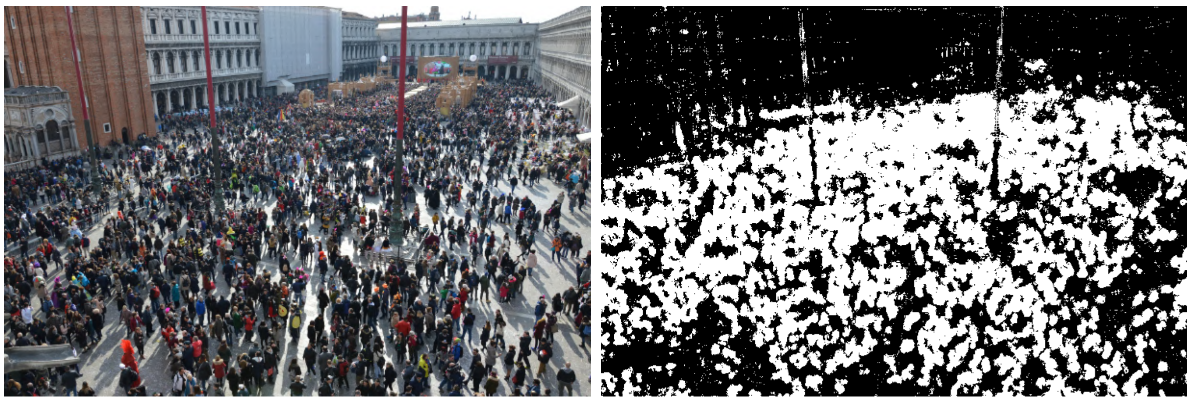


Figure 4.22: The image from the dataset with the largest number of people (3975) and the corresponding U-Net inference. It is clear that, even if the U-Net output is acceptable and generally correct, most of the circles (i.e., detected people) are considered as a single contour and thus counted only once. However if the location variables (distance from the camera, zoom level, etc) are known, the number of white pixels can be associated to the number of people.

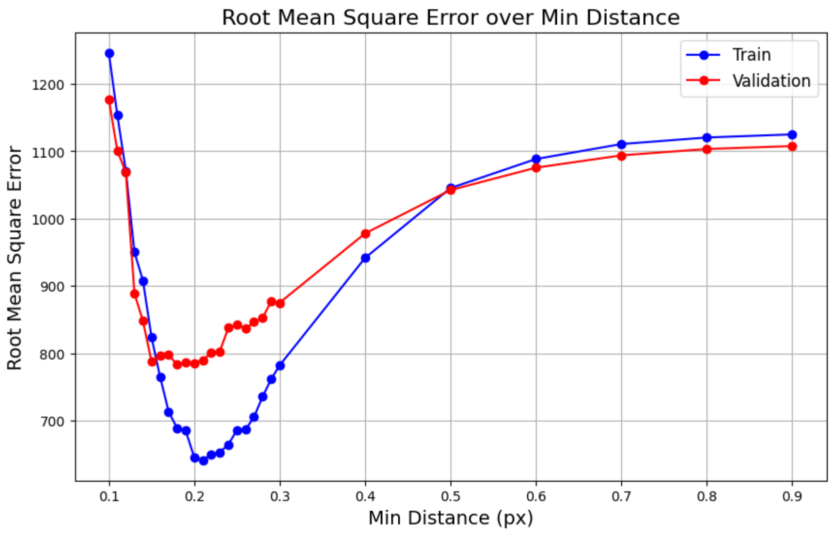


Figure 4.23: Root Mean Square Error (RMSE) between predicted count and true count for both the training and validation set. The number of predicted count was obtained by counting the local maxima of the distance function whose value is greater than d_{min} and the RMSE is dependent on this threshold value. The distance is measured in pixels and normalized between 0 and 1 for each image.

The implemented solution to this problem is the following: on the eroded binary image of Fig. 4.17, a distance function [78][79] is calculated. This returns the distance from the closest zero-value pixel at each location. Obviously, the distance function's value for a zero-value pixel is zero. Then, local maxima of this function correspond to centers of circles, and it is thus the number of those maxima that are associated to the number of people. Since zero-value pixels are also considered local maxima, a threshold d_{min} for a minimum distance value on each maximum has to be used. Of course, higher values of this parameter leads to fewer accepted detection and this threshold also allows to discard eventual small point-like detection that could be noise. Similarly to what was done with contours, the optimal threshold value d_{min} was empirically obtained by calculating the MSE between predicted counts and true ones for different value of d_{min}. These are plotted in Fig. 4.23, from which it is obtain that the optimal value is $d_{min} \simeq 0.2$. Note that the distance value was normalized between 0 and 1, for each image individually.

Fig. 4.24 shows the predicted number of people against the true ones for $d_{min} = 0.2$. The result is better then using the number of contours as in Fig. 4.21, but still has a RMSE of ~ 646 for the training set and ~ 786 for the validation set.

Finally, the position of local maxima is used as the predicted position of people. Results are shown in Fig. 4.25.

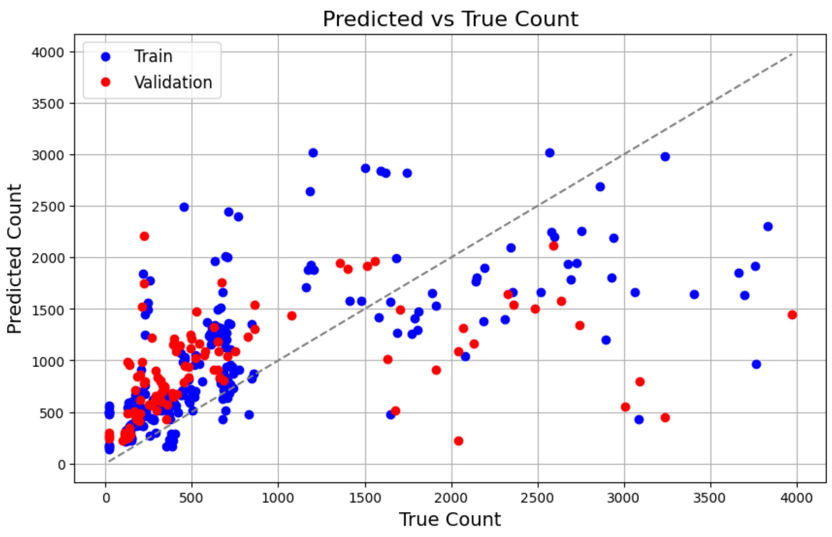
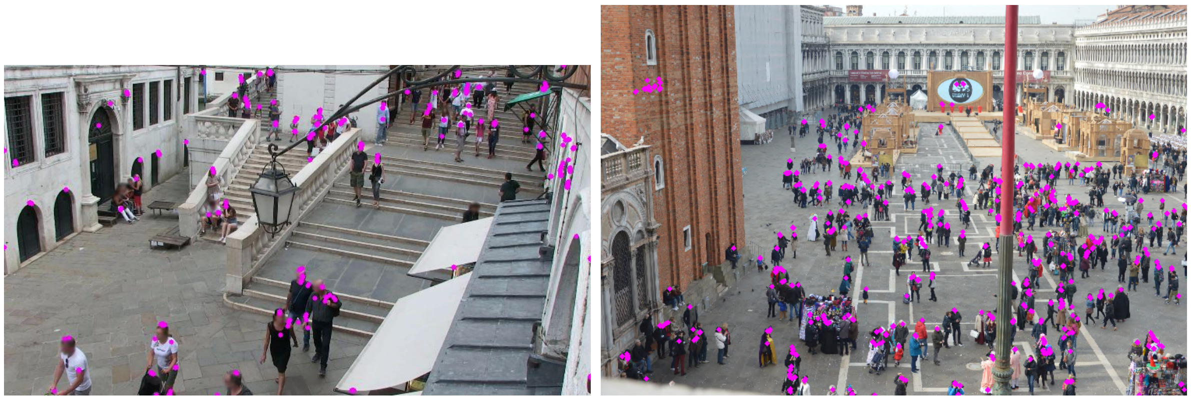


Figure 4.24: Predicted count over true count for both the training and validation set. The predicted count was obtained by counting the number of local maxima of the distance function whose value is grater than $d_{min} = 0.2$.



(a) (b)

Figure 4.25: Examples of U-Net head detection using the distance function method. The model obviously works better on the same type of images where it has been trained (figure b), but correctly detects people's head even in unseen images (figure a). In both cases however, some false positive error are visible.

4.4 Traditional algorithm

As already mentioned in section 3.3, getting the background is a crucial step in traditional computer vision algorithms. This can be obtained “statically” (i.e., calculating the median value for each pixel in different frames) or “dynamically” (i.e., performing a weighted sum between the current frame and the background). In the work case of this thesis, the first method was used, since it resulted to be more consistent and stable. In fact, as the videos are not very long, the scene background doesn’t change much due for instance to different illumination at various time or weather. In a real continuous application it would be suitable instead to calculate the background using the weighted sum method, with a small value for the weight to slowly account for those changes.

Once the background is obtained and subtracted from the current pixel, a threshold is performed to obtain a binary image. Similarly to what is described in the previous section, a density estimation can be obtain by correlating the number of non-zero pixels with the number of people.

Regarding object detection, this traditional method doesn’t output circles over heads, but instead detect all the moving pixels (i.e., the whole body of people), so the distance function technique can’t be applied. This means that with an high density crowd, it is not possible to detect the position of each person, since even a small touch between two of them will result in only one detection. A general correspondence between the number of non-zero pixels and the crowd density is all it can be established, since counting the number of contour will lead to an underestimation of the people count. Note however that labelled data on video are not available for this project, so no quantitative measurements like in section 4.3.3 can be established.

Qualitatively, results of the traditional algorithm can be seen in Fig. 4.26, that shows it performing over the same videos of Fig. 4.5. Some imperfections are visible and when the detections of different people overlap, they are considered as a single contour.

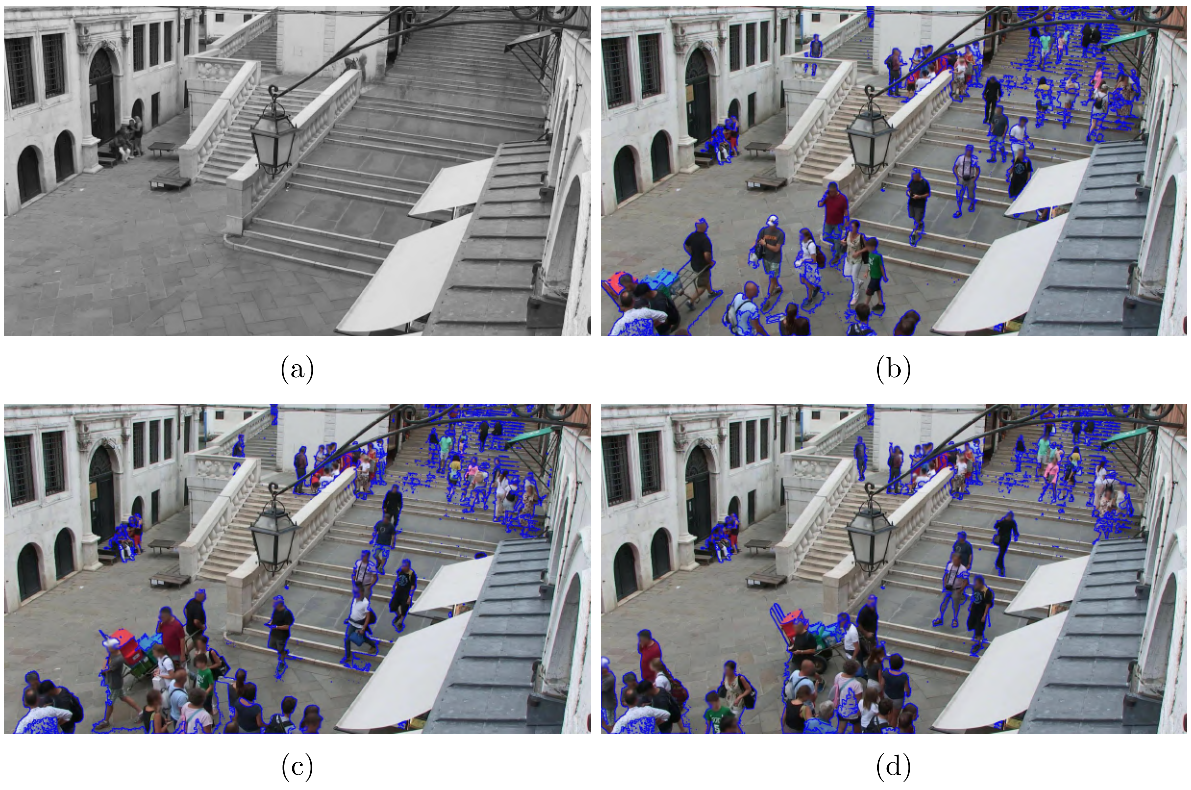


Figure 4.26: Application of the background subtraction algorithm to a video of a alley. A threshold value of 30 (from the range $[0, 255]$) was empirically evaluated to work best and used. **(a)** Gray-scale background obtained by taking the median pixel value out of 50 random frames; **(b,c,d)** Different frames of the resulted video with detected contour drawn as blue lines. The images are 6 frames (2 seconds) apart.

4.5 Tracking algorithm

As described in section 3.4, tracking people could be an alternative approach to estimate crowd density. Results of YOLO tracking on videos of alleys is shown for instance in Fig. 4.5. The YOLO model allows for accurate tracking of individuals by associating detected objects across multiple frames. This enables the reconstruction of trajectories for each person throughout the video. In the example shown in Fig. 4.27 the trajectories found in the 900-frames-long video (approximately 5 minutes) are all plot together. A general directional tendency emerges: the majority of paths aligns with a north-east to south-west direction, indicating that crowd flow follows a common pattern, which can be visualized as a rough vector field.

Furthermore, the characteristics of the trajectories suggest that the crowd is relatively unconstrained in its movement: the fact that the trajectories are long and also that some of them cross the dominant flow direction implies that individuals are not obstructed or packed closely together. In denser crowds, we would expect more erratic and shorter paths as people navigate around one another, leading to increased interactions and collisions in their trajectories. Here, the relatively smooth and linear paths indicate a lower crowd density, correctly suggesting more open space for movement.

Note however that conducting a quantitative analysis of crowd density remains a challenge due to the absence of labeled ground-truth data. Without accurate labels indicating the true crowd density in each video frame, it's difficult to establish a concrete relationship between the observed trajectories and actual crowd density. In future works, access to labelled datasets could enable a more robust quantitative analysis.

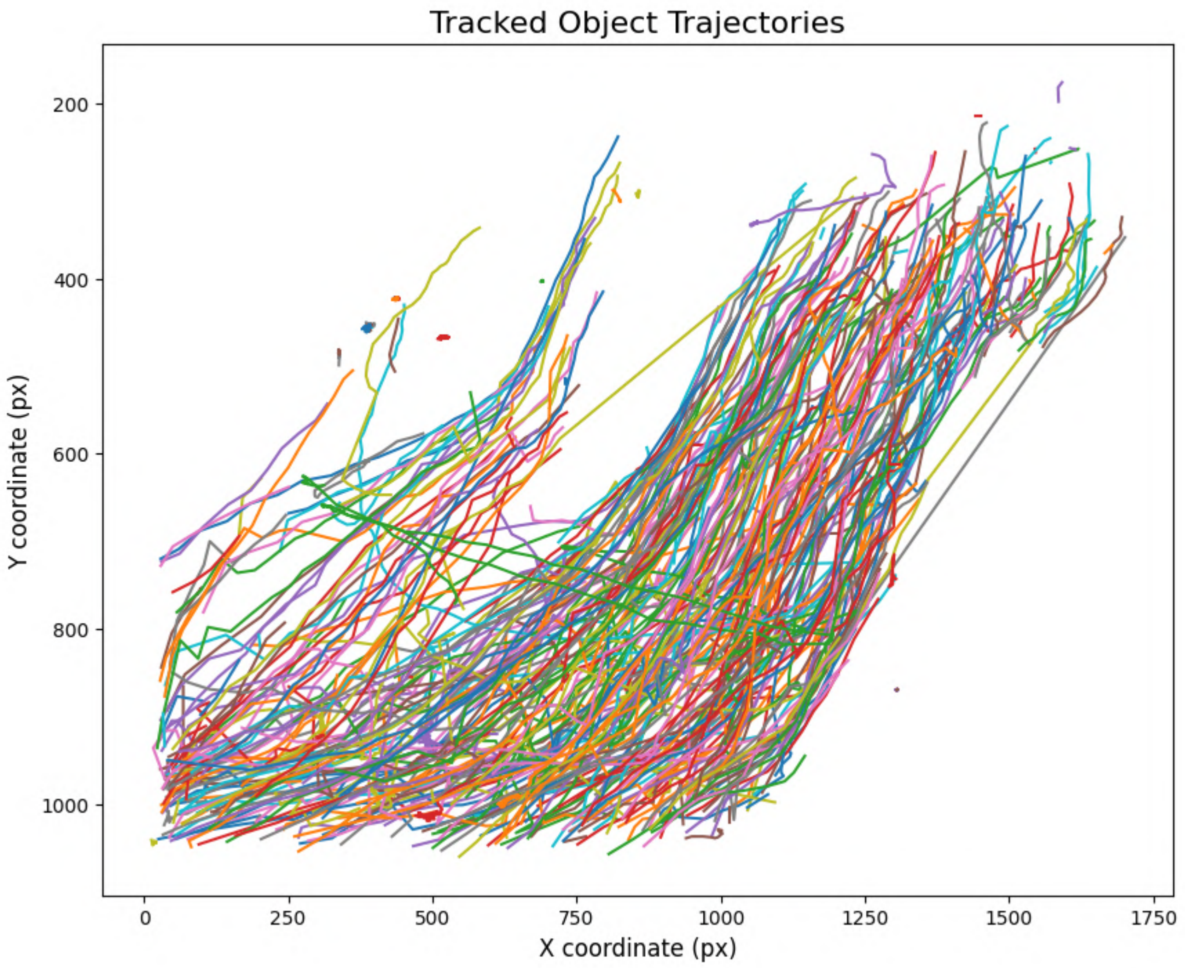


Figure 4.27: Trajectories found by YOLO tracking method on the alley video of Fig. 4.5. The paths being so long and with a prevailing direction is an indication of a free-flowing crowd. Also, some trajectories that crosses this direction are visible, indicating that people are not densely packed and movement against the general flow are feasible.

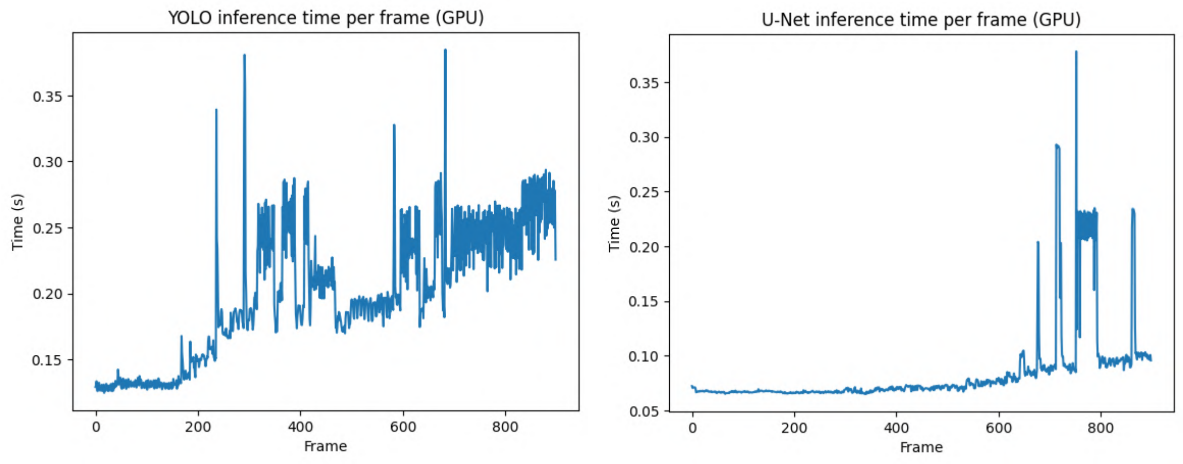


Figure 4.28: YOLO and U-Net inference time using GPU. Both slightly increase at the end, due to thermal throttling. The average inference time is 199.973 ms for YOLO and only 84.977 ms for the U-Net.

4.6 Computational time analysis

In this chapter, computational time for the three methods are evaluated and compared, using two different machines: a computer equipped with 16 GB of RAM and a CUDA capable GPU: NVIDIA GeForce GTX 1050 Ti with Max-Q Design and a Raspberry Pi 5 with 4 GB of RAM.

The computational times of YOLO and U-Net inference when the GPU is used and the models are applied to the same video of Fig. 4.5, 4.27 and 4.25a, are shown in Fig. 4.28. The average time required to compute a frame is 199.973 ms for YOLO and only 84.977 ms for the U-Net. Both of them slightly increase with time, due to hardware overheating and the subsequent thermal throttling. YOLO employs basically the same time for both the predicting and tracking methods. Also, since its output are bounding boxes ready to be counted and individuated, the calculated times for the U-Net comprehend the actual model inference plus the people counting algorithm that uses the distance function method, described in section 4.3.3.

In Fig. 4.29 the same computational times using the Raspberry's CPU are plotted. The average time required to compute a frame in this case is 8.263 s for YOLO and 2.939 s for the U-Net. Note however that only the first 73 frames are plotted for YOLO, as the Raspberry small RAM quickly saturated and the device halted.

Finally, the computational speed of the method based on background subtraction was evaluated. This algorithm is designed in C++ to work with CPU and thus it is not tested on GPU. The output performance is visible in Fig. 4.30 and the average time to compute a single frame on the Raspberry is 48.24 ms.

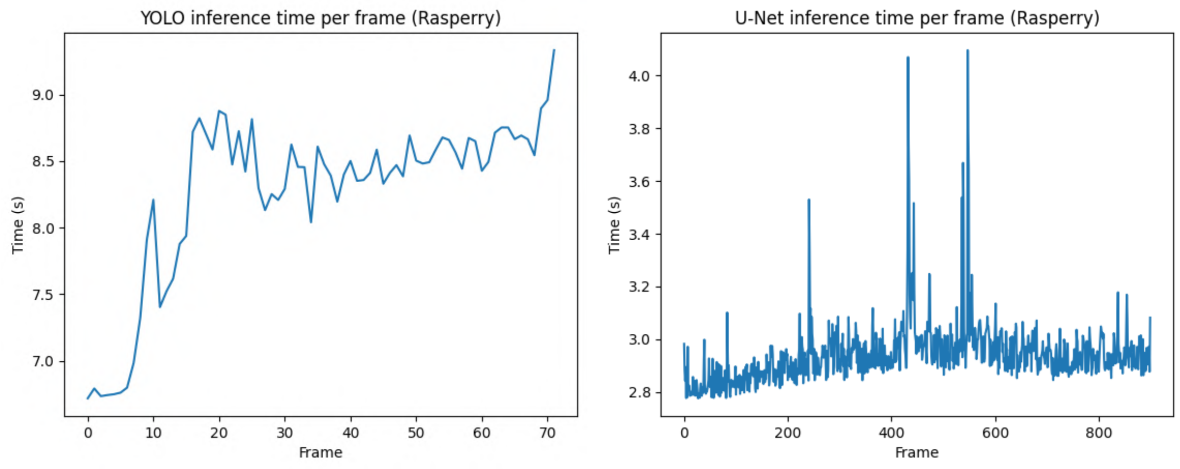


Figure 4.29: YOLO and U-Net inference time using CPU on a Raspberry PI 5. The average inference time is 8.263 s for YOLO and 2.939 s for the U-Net.

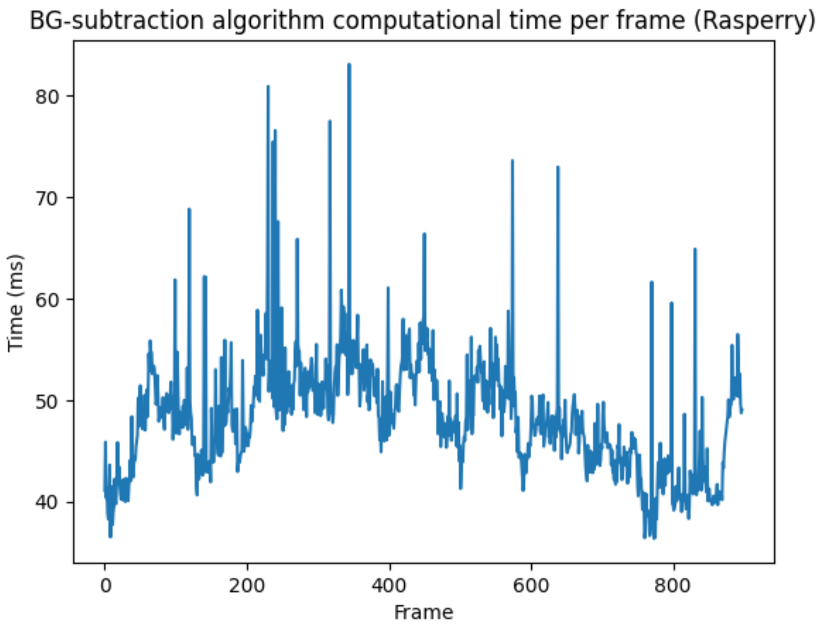


Figure 4.30: Computational time needed to apply the BG-subtraction-based algorithm from a Raspberry PI 5. The average time is 48.24 ms.

Chapter 5

Conclusion and future work

In this work, different methodologies for analyzing crowd density in public spaces were explored, with a focus on Venice as the working case. Venice is an interesting case study for crowd counting, because the wide crowded squares and the narrow alleys offer a great variety of situations and scenarios. Also, with the huge number of tourists that annually visit the city, Venice is a perfect example of what is referred to as overcrowding. This phenomenon can lead to potentially dangerous situations such as stampedes and compressive asphyxia. Estimating crowd density is a crucial step to predict such tragic scenarios.
In literature there exists different methods for crowd density estimation and dynamic reconstruction, and the primary of this thesis was to evaluate the effectiveness of these methods, with particular attention to their computational performances and accuracy.

The methods included the use of YOLOv8, a custom U-Net implementation, a traditional computer vision algorithm based on background subtraction, and a novel approach based on tracking. Each method was examined, applied to real-world images and videos from Venice's squares and alleys, and evaluated for both performance and computational efficiency. The speed measurements were conducted on both a CUDA-enabled GPU and a Raspberry Pi 5, to assess their viability across different hardware platforms.

- YOLOv8 was selected as an off-the-shelf object detection model, known for its real-time performances and considered as the state-of-the-art in lower density scenarios. The model demonstrated strong performance in both detecting and tracking people in less crowded environments, producing consistent bounding boxes that could be used for both crowd counting and density estimation. However its performance degraded significanlty in higher-density scenarios where it struggled to make accurate predictions due to occlusions people. Additionally, YOLOv8 proved to be computational expensive, requiring GPU to maintain a functional (and still low) FPS rate in real-time scenarios, and was the slowest of the methods tested. An attempt to fine-tune YOLOv8 to specifically detect people's head was unsuccessful and no custom YOLO model could be used for this project, limiting its flexibility.

- The custom U-Net model developed for the crowd counting problem performed segmentation tasks, with the goal of identifying individuals in the crowd. The model was successfully trained to segment people's head and its output is a binary image in which people's heads are represented as white circles. This provided an effective means of crowd density estimation and counting. A correlation between the number of white pixels in the mask and the crowd density can be established if the camera parameters such as zoom and angle were known to account for perspective distortion, allowing for reasonable density estimates in both sparse and dense crowd scenarios. However, while the method allowed for counting individuals, it lacked precision in high-density situations due to occlusion and head size variability. Computationally, U-Net performed faster than YOLO but still did not achieve real-time performance on the Raspberry Pi 5, making it better suited for offline processing or more powerful hardware.

- The traditional computer vision algorithm based on background subtraction demonstrated significant advantages in terms of speed and efficiency. Designed to run on CPU, this method achieved real-time performance even on the Raspberry Pi 5, making it particularly suitable for lightweight and constrained hardware. This technique successfully detected foreground objects by subtracting a reference background image from the current frame. The foreground could then be used to estimate crowd density again by counting the number of non-zero pixels. In lower-density settings, with the appropriate steps such as threshold and simple morphological operations, the method was able to accurately detect people and track their movements. However, since the model relies on video streams to generate the background image, it is restricted to dynamic video analysis and could not be applied to labelled images to quantitatively measure its accuracy.

- An original concept was proposed in this work, which leverages people's movement trajectories to infer crowd density. This is based on the idea that in low-density crowds, individuals movements and trajectories are smoother, longer and more linear, while in high-density situations, people's trajectories become shorter and more chaotic, as movement is constrained. This method does not require precise tracking of all individuals across frames, as any failure in tracking due to incorrect data association and subsequent identity switching could still be interpreted as an indicator of higher density. Despite the promising nature of this concept, quantitative testing wasn't possible, since no labelled videos are available. Nevertheless, the approach has potential for future work, particularly in scenarios where trajectory patterns can be captured over time.

Future works may concentrate in the improvement of different aspects of this thesis, starting from a better data acquisition that would lead to better labelled datasets and a subsequent proper validation of the dynamic parts of the different models. Indeed, this aspect requires some kind of ground truth for videos, while it was only available for stationary images.

Improvements can also be done on the U-Net model, for instance by changing the labelled masks and training it to output an heatmap, rather then a binary segmentation. This would lead to some improvements in higher density scenarios, as an output image in which pixels value reflects people's density would carry greater information than just binary values. Regarding post process operations, better methods to get useful information from binary outputs (either of the U-Net or the traditional algorithm) can be explored. Cameras could be fixed, walkable area defined and perspective correction adopted. This way, the more-than-once cited relation between number of pixels and people density can finally be established.

Finally, more exploration are certainly needed regarding the the usage of trajectories features to infer crowd density. The original concept seems to have good foundations, but no proper validation nor even quantitative measures were performed. Future works may develop in this direction, leveraging the good quality of state-of-the-art object trackers.

In conclusion, the methods explored in this thesis highlight the varying strengths and limitations of different approaches to crowd density estimation. YOLOv8, while powerful in low-density cases, suffers from high computational costs and struggles in denser scenes. The U-Net segmentation model provides an effective way of counting and estimating density through binary masks, and can be applied to static images, but still isn't able to achieve real-time performance. The traditional background subtraction method proves to be computationally efficient and viable for real-time applications on low-power devices, though it is limited to video-based analysis. Finally, the trajectory-based approach holds promise but requires further validation with labeled data. Together, these methods provide a comprehensive toolkit for analyzing crowd dynamics in urban settings, with specific applications to real-world environments like Venice.

Appendix A

Metrics and Optimization Techniques in Object Detection

A.1 Object detection metrics

Intersection over Union (IoU) is the ratio of the intersection area to the union area of the predicted bounding box and the ground truth bounding box. Indicating with A and B the bounding box and the ground truth box:

$$IoU(A,B) = \frac{A \cup B}{A \cap B}$$

The average precision (AP), traditionally called mean average precision (mAP), is the commonly used metric for evaluating the performance of object detection models [10]. It is based on precision and recall metrics: precision measure the accuracy of the model's positive predictions, while recall measures the proportion of actual positive cases that the model correctly identifies. Using the definition of true positive (TP), true negative (TN), false positive (FP) and false negative (FN), they can be defined as:

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

There is often a trade-off between precision and recall: for example, increasing the number of detected objects (higher recall) can result in more false positives (lower precision). To account for this trade-off, the AP metric calculates and averages the area under the precision-recall curve (which plots precision against recall) at different confidence thresholds.

Another metric that uses a combination of precision and recall is the Dice score (also known as F1-score), which is the harmonic mean of Precision and Recall.

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$$

Finally, the accuracy is just the ratio between the number of correctly predicted pixels and the total number of pixels in the image:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

A.2 Batch normalization

Batch normalization (also known as BatchNorm) is a technique proposed by Sergey Ioffe and Christian Szegedy [24], which helps to stabilize and accelerate the training process of artificial neural networks by normalizing the input data of each layer within a neural network. The result is a faster convergence rate and improved generalization of the model. The process of batch normalization it the following: first, given a mini-batch $B = \{x_1, x_2, \ldots, x_m\}$, the mean μ_B and variance σ_B^2 are computed:

$$\mu_B = \sum_{i=1}^{m} x_i \qquad \sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2$$

Then each input is normalized:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

where ϵ is an arbitrary small constant added to avoid division by zero and ensure numerical stability. The resulting normalized values $\hat{x}_i$ have zero mean and unit variance, thus, they are scaled and shifted using the learnable parameters β and γ:

$$y_i = \gamma \hat{x}_i + \beta$$

allowing the network to retain the ability to represent complex transformations, as the output of the normalized values are adjusted to any desired range.

Although batch normalization has become popular due to its strong empirical performance, the working mechanism of the method is not yet well-understood and the reasons behind its effectiveness remain under discussion.

Appendix B

Image processing algorithms

B.1 Binary erosion of images

Erosion is a fundamental morphological operation, defined for binary images (but later extended to grayscale as well) that shrink or "erode" the boundaries of the foreground object (i.e., the white pixels). Erosion works by sliding a structuring element (a small, predefined shape like a square or circle) across the image. For each pixel in the image, it checks whether the structuring element completely fits within the foreground region. If it does, the pixel remains white (1); otherwise, it is turned into a background pixel (0). Essentially, the operation "erodes" the white regions by removing pixels that do not fit the shape of the structuring element. For instance, if using a 3×3 square structuring element, the algorithm will remove any white pixel that does not have at least eight white neighbours.

Mathematically, let E be an integer grid, and A a binary image on E. The erosion of the binary image A by the structuring element B is defined by:

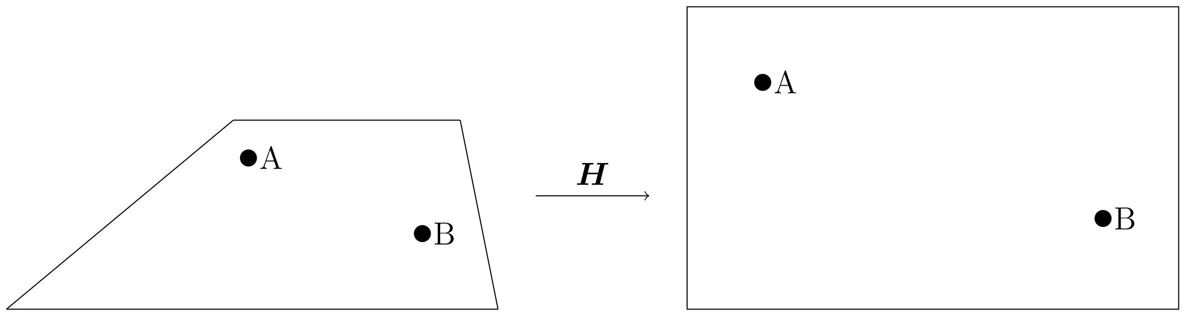
$$A \ominus B = \{z \in E \mid B_z \subseteq A\}$$

where B_z is the translation of B by the vector z: $\quad B_z = \{b + z \mid b \in B\} \quad \forall z \in B$

B.2 Perspective correction

Perspective correction is a geometric transformation used in image processing to adjust the apparent distortion of an object in an image due to the angle of viewing. This technique is often used to make objects appear as if they are viewed from a perpendicular perspective (i.e., from above for the case of this thesis). In fact, when an image is created, the pixels representing an object in 3D space are projected onto a 2D plane (the image); if the camera is not perpendicular to the object, this projection leads to geometric distortion. Perspective correction undoes this distortion by transforming the points in the 2D image back into their correct positions.

The transformation typically uses four key points in the image, which correspond to the corners of the distorted object (usually a rectangle appearing as a trapezoid, like shown below). These points are then mapped to the corners of the actual shape, resulting in a corrected, rectangular appearance:



The mathematical operation used for perspective correction is a homography transformation, which relates the coordinates of the points in the distorted image to the coordinates of the points in the undistorted image:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where $\boldsymbol{H}$ is a 3×3 transformation matrix called *homography matrix*; (x, y) are the coordinates of the original point and (x', y') are the coordinates of the point after transformation.

Given a set of points (typically four), the goal of perspective correction is to compute the matrix $\boldsymbol{H}$ that maps the distorted image coordinates to the correct coordinates. This is easily done using OpenCV [74] `getPerspectiveTransform` and `warpPerspective` methods.

Bibliography

[1] Dimitrios Chrysostomou, Georgios Ch. Sirakoulis, and Antonios Gasteratos. “A bio-inspired multi-camera system for dynamic crowd analysis”. In: *Pattern Recognition Letters* 44 (2014). Pattern Recognition and Crowd Analysis, pp. 141–151. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2013.11.020. URL: https://www.sciencedirect.com/science/article/pii/S0167865513004650.

[2] Swathi H.Y., G. Shivakumar, and H.S. Mohana. “Crowd Behavior Analysis: A Survey”. In: *2017 International Conference on Recent Advances in Electronics and Communication Technology (ICRAECT)*. 2017, pp. 169–178. DOI: 10.1109/ICRAECT.2017.66.

[3] Antoni B. Chan, Zhang-Sheng John Liang, and Nuno Vasconcelos. “Privacy preserving crowd monitoring: Counting people without people models or tracking”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–7. DOI: 10.1109/CVPR.2008.4587569.

[4] Jason M. Grant and Patrick J. Flynn. “Crowd Scene Understanding from Video: A Survey”. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 13.2 (Mar. 2017). ISSN: 1551-6857. DOI: 10.1145/3052930. URL: https://doi.org/10.1145/3052930.

[5] Julio Cezar Silveira Jacques et al. “Understanding people motion in video sequences using Voronoi diagrams”. In: *Pattern Analysis and Applications* 10.4 (Oct. 2007), pp. 321–332. ISSN: 1433-755X. DOI: 10.1007/s10044-007-0070-1. URL: https://doi.org/10.1007/s10044-007-0070-1.

[6] Dirk Helbing and Péter Molnár. “Social force model for pedestrian dynamics”. In: *Phys. Rev. E* 51 (5 May 1995), pp. 4282–4286. DOI: 10.1103/PhysRevE.51.4282. URL: https://link.aps.org/doi/10.1103/PhysRevE.51.4282.

[7] Anders Johansson, Dirk Helbing, and Pradyumn Kumar Shukla. “Specification of the Social Force Pedestrian Model by Evolutionary Adjustment to Video Tracking Data.” In: *Advances in Complex Systems* 10 (Dec. 2007), pp. 271–288. DOI: 10.1142/S0219525907001355.

[8] Dirk Helbing et al. “Self-Organized Pedestrian Crowd Dynamics: Experiments, Simulations, and Design Solutions”. In: *Transportation Science* 39.1 (2005), pp. 1–24. DOI: 10.1287/trsc.1040.0108. eprint: https://doi.org/10.1287/trsc.1040.0108. URL: https://doi.org/10.1287/trsc.1040.0108.

[9] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *Ultralytics YOLO*. Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics.

[10] Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. “A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS”. In: *Machine Learning and Knowledge Extraction* 5.4 (2023), pp. 1680–1716. ISSN: 2504-4990. DOI: 10.3390/make5040083. URL: https://www.mdpi.com/2504-4990/5/4/83.

[11] Muhammad Yaseen. *What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector*. 2024. arXiv: 2408.15857 [cs.CV]. URL: https://arxiv.org/abs/2408.15857.

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV]. URL: https://arxiv.org/abs/1505.04597.

[13] Daniele Pucci. “Analisi di video con visione binoculare”. 2022. URL: http://amslaurea.unibo.it/27296/.

[14] Gopal Thapa, Kalpana Sharma, and Mrinal Ghose. “Moving Object Detection and Segmentation using Frame Differencing and Summing Technique”. In: *International Journal of Computer Applications* 102 (Sept. 2014), pp. 20–25. DOI: 10.5120/17828-8647.

[15] Anthony C. Davies, Jiaqi Yin, and Sergio A. Velastín. “Crowd monitoring using image processing”. In: *Electronics & Communication Engineering Journal* 7 (1995), pp. 37–47. URL: https://api.semanticscholar.org/CorpusID:18502218.

[16] Siu-Yeung Cho, T.W.S. Chow, and Chi-Tat Leung. “A neural-based crowd estimation by hybrid global learning algorithm”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29.4 (1999), pp. 535–541. DOI: 10.1109/3477.775269.

[17] A. L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229. DOI: 10.1147/rd.33.0210.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[19] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4 (Apr. 1980), pp. 193–202. ISSN: 1432-0770. DOI: 10.1007/BF00344251. URL: https://doi.org/10.1007/BF00344251.

[20] Gianluca Mancusi et al. "TrackFlow: Multi-Object Tracking with Normalizing Flows". In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023, pp. 9497–9509. DOI: 10.1109/ICCV51070.2023.00874.

[21] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. URL: https://arxiv.org/abs/1506.02640.

[22] Ao Wang et al. *YOLOv10: Real-Time End-to-End Object Detection*. 2024. arXiv: 2405.14458 [cs.CV]. URL: https://arxiv.org/abs/2405.14458.

[23] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6517–6525. DOI: 10.1109/CVPR.2017.690.

[24] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG]. URL: https://arxiv.org/abs/1502.03167.

[25] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV]. URL: https://arxiv.org/abs/1804.02767.

[26] Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9 (2015), pp. 1904–1916. DOI: 10.1109/TPAMI.2015.2389824.

[27] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV]. URL: https://arxiv.org/abs/2004.10934.

[28] Hao Zhang, Shuaijie Zhang, and Renbin Zou. *Select-Mosaic: Data Augmentation Method for Dense Small Object Scenes*. 2024. arXiv: 2406.05412 [cs.CV]. URL: https://arxiv.org/abs/2406.05412.

[29] Glenn Jocher. *YOLOv5 by Ultralytics*. Version 7.0. May 2020. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5.

[30] Li Yang and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice". In: *Neurocomputing* 415 (2020), pp. 295–316. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.07.061. URL: https://www.sciencedirect.com/science/article/pii/S0925231220311693.

[31] Chuyi Li et al. *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. 2022. arXiv: 2209.02976 [cs.CV]. URL: https://arxiv.org/abs/2209.02976.

[32] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. *YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information.* 2024. arXiv: 2402.13616 [cs.CV]. URL: https://arxiv.org/abs/2402.13616.

[33] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 740–755. DOI: 10.1007/978-3-319-10602-1_48.

[34] Zhou Cao et al. “CrowdUNet: Segmentation assisted U-shaped crowd counting network”. In: *Neurocomputing* 601 (2024), p. 128215. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2024.128215. URL: https://www.sciencedirect.com/science/article/pii/S092523122400986X.

[35] Marcellino, Tjeng Wawan Cenggoro, and Bens Pardamean. “UNET++ with Scale Pyramid for Crowd Counting”. In: *ICIC Express Letters* 16 (Jan. 2022), pp. 75–82. DOI: 10.24507/icicel.16.01.75.

[36] Anum Ilyas and Narmeen Bawany. “Crowd dynamics analysis and behavior recognition in surveillance videos based on deep learning”. In: *Multimedia Tools and Applications* (Sept. 2024). ISSN: 1573-7721. DOI: 10.1007/s11042-024-20161-7. URL: https://doi.org/10.1007/s11042-024-20161-7.

[37] Erik Bochinski, Volker Eiselein, and Thomas Sikora. “High-Speed tracking-by-detection without using image information”. In: *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 2017, pp. 1–6. DOI: 10.1109/AVSS.2017.8078516.

[38] Jerome Berclaz et al. “Multiple Object Tracking Using K-Shortest Paths Optimization”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.9 (2011), pp. 1806–1819. DOI: 10.1109/TPAMI.2011.21.

[39] Chanho Kim et al. “Multiple Hypothesis Tracking Revisited”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 4696–4704. DOI: 10.1109/ICCV.2015.533.

[40] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. “Detect to Track and Track to Detect”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 3057–3065. DOI: 10.1109/ICCV.2017.330.

[41] David Held, Sebastian Thrun, and Silvio Savarese. “Learning to Track at 100 FPS with Deep Regression Networks”. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 749–765. ISBN: 978-3-319-46448-0.

[42] Qiankun Liu et al. “GSM: Graph Similarity Model for Multi-Object Tracking”. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. Ed. by Christian Bessiere. Main track. International Joint Conferences on Artificial Intelligence Organization, July 2020, pp. 530–536. DOI: 10.24963/ijcai.2020/74. URL: https://doi.org/10.24963/ijcai.2020/74.

[43] Tim Meinhardt et al. *TrackFormer: Multi-Object Tracking with Transformers*. 2022. arXiv: 2101.02702 [cs.CV]. URL: https://arxiv.org/abs/2101.02702.

[44] Bin Yan et al. “Learning Spatio-Temporal Transformer for Visual Tracking”. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 10428–10437. DOI: 10.1109/ICCV48922.2021.01028.

[45] Jialian Wu et al. “Track to Detect and Segment: An Online Multi-Object Tracker”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 12347–12356. DOI: 10.1109/CVPR46437.2021.01217.

[46] Ashish Vaswani et al. “Attention is all you need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.

[47] Alex Bewley et al. “Simple online and realtime tracking”. In: *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, July 2016. DOI: 10.1109/icip.2016.7533003. URL: http://dx.doi.org/10.1109/ICIP.2016.7533003.

[48] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552. eprint: https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35_1.pdf. URL: https://doi.org/10.1115/1.3662552.

[49] Harold W. Kuhn. “The Hungarian Method for the Assignment Problem”. In: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Ed. by Michael Jünger et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 29–47. ISBN: 978-3-540-68279-0. DOI: 10.1007/978-3-540-68279-0_2. URL: https://doi.org/10.1007/978-3-540-68279-0_2.

[50] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe. “Tracking Without Bells and Whistles”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2019. DOI: 10.1109/iccv.2019.00103. URL: http://dx.doi.org/10.1109/ICCV.2019.00103.

[51] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. *Tracking Objects as Points*. 2020. arXiv: 2004.01177 [cs.CV]. URL: https://arxiv.org/abs/2004.01177.

[52] Kaiwen Duan et al. “CenterNet: Keypoint Triplets for Object Detection”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 6568–6577. DOI: 10.1109/ICCV.2019.00667.

[53] Zhichao Lu et al. *RetinaTrack: Online Single Stage Joint Detection and Tracking.* 2020. arXiv: 2003.13870 [cs.CV]. URL: https://arxiv.org/abs/2003.13870.

[54] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection.* 2018. arXiv: 1708.02002 [cs.CV]. URL: https://arxiv.org/abs/1708.02002.

[55] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple Online and Realtime Tracking with a Deep Association Metric". In: *2017 IEEE International Conference on Image Processing (ICIP).* IEEE. 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962.

[56] Nicolai Wojke and Alex Bewley. "Deep Cosine Metric Learning for Person Re-identification". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV).* IEEE. 2018, pp. 748–756. DOI: 10.1109/WACV.2018.00087.

[57] Yifu Zhang et al. *ByteTrack: Multi-Object Tracking by Associating Every Detection Box.* 2022. arXiv: 2110.06864 [cs.CV]. URL: https://arxiv.org/abs/2110.06864.

[58] Yifu Zhang et al. *ByteTrackV2: 2D and 3D Multi-Object Tracking by Associating Every Detection Box.* 2023. arXiv: 2303.15334 [cs.CV]. URL: https://arxiv.org/abs/2303.15334.

[59] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks.* 2015. arXiv: 1511.08458 [cs.NE]. URL: https://arxiv.org/abs/1511.08458.

[60] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *Ultralytics YOLO.* Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics.

[61] Dongmei Wang et al. "Ea-yolo: efficient extraction and aggregation mechanism of YOLO for fire detection". In: *Multimedia Systems* 30.5 (Sept. 2024), p. 287. ISSN: 1432-1882. DOI: 10.1007/s00530-024-01489-4. URL: https://doi.org/10.1007/s00530-024-01489-4.

[62] Andrew L. Maas. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: *30th International Conference on Machine Learning.* 2013. URL: https://api.semanticscholar.org/CorpusID:16489696.

[63] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs).* 2023. arXiv: 1606.08415 [cs.LG]. URL: https://arxiv.org/abs/1606.08415.

[64] Daniele Pucci. *PyTorch implementation of a U-Net.* Version 1.0.0. Sept. 2024. URL: https://github.com/puccj/U-Net.

[65] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation.* 2015. arXiv: 1411.4038 [cs.CV]. URL: https://arxiv.org/abs/1411.4038.

[66] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).

[67] Mohand Said Allili, Nizar Bouguila, and Djemel Ziou. “A Robust Video Foreground Segmentation by Using Generalized Gaussian Mixture Modeling”. In: *Fourth Canadian Conference on Computer and Robot Vision (CRV ’07)*. 2007, pp. 503–509. DOI: 10.1109/CRV.2007.7.

[68] Jaime Gallego, Montse Pardas, and Jose-Luis Landabaso. “Segmentation and tracking of static and moving objects in video surveillance scenarios”. In: *2008 15th IEEE International Conference on Image Processing*. 2008, pp. 2716–2719. DOI: 10.1109/ICIP.2008.4712355.

[69] Girisha. R and Murali. S. “Segmentation of motion objects from surveillance video sequences using partial correlation”. In: *2009 16th IEEE International Conference on Image Processing (ICIP)*. 2009, pp. 1129–1132. DOI: 10.1109/ICIP.2009.5414526.

[70] A. P. Dempster, N. M. Laird, and D. B. Rubin. “Maximum Likelihood from Incomplete Data Via the EM Algorithm”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (Dec. 2018), pp. 1–22. ISSN: 0035-9246. DOI: 10.1111/j.2517-6161.1977.tb01600.x. eprint: https://academic.oup.com/jrsssb/article-pdf/39/1/1/49117094/jrsssb_39_1_1.pdf. URL: https://doi.org/10.1111/j.2517-6161.1977.tb01600.x.

[71] Donald B. Rubin. “The Calculation of Posterior Distributions by Data Augmentation: Comment: A Noniterative Sampling/Importance Resampling Alternative to the Data Augmentation Algorithm for Creating a Few Imputations When Fractions of Missing Information Are Modest: The SIR Algorithm”. In: *Journal of the American Statistical Association* 82.398 (June 1987). Full publication date: Jun., 1987, pp. 543–546. ISSN: 01621459. DOI: 10.2307/2289460. URL: http://www.jstor.org/stable/2289460.

[72] Connor Shorten and Taghi M. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning”. In: *Journal of Big Data* 6.1 (July 2019), p. 60. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0.

[73] John Canny. “A Computational Approach To Edge Detection”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* PAMI-8 (Dec. 1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[74] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[75] PyTorch Contributors. *PyTorch Documentation*. Function: `BCEWithLogitsLoss`. PyTorch. 2023. URL: https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html#torch.nn.BCEWithLogitsLoss.

[76] Lee R. Dice. “Measures of the Amount of Ecologic Association Between Species”. In: *Ecology* 26.3 (1945), pp. 297–302. DOI: https://doi.org/10.2307/1932409. eprint: https://esajournals.onlinelibrary.wiley.com/doi/pdf/10.2307/1932409. URL: https://esajournals.onlinelibrary.wiley.com/doi/abs/10.2307/1932409.

[77] Tage Sørensen et al. *A method of establishing group of equal amplitude in plant sociobiology based on similarity of species content and its application to analyses of the vegetation on Danish commons*. 1948. URL: https://api.semanticscholar.org/CorpusID:135206594.

[78] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. “Distance Transforms of Sampled Functions”. In: *Theory of Computing* 8.19 (2012), pp. 415–428. DOI: 10.4086/toc.2012.v008a019. URL: https://theoryofcomputing.org/articles/v008a019.

[79] Gunilla Borgefors. “Distance transformations in digital images”. In: *Computer Vision, Graphics, and Image Processing* 34.3 (1986), pp. 344–371. ISSN: 0734-189X. DOI: https://doi.org/10.1016/S0734-189X(86)80047-0. URL: https://www.sciencedirect.com/science/article/pii/S0734189X86800470.

Acknowledgments

Huge thanks to Professor Armando Bazzani, who has been following me tirelessly for these last three years of academic career together. Thanks for the invaluable guidance of a mentor, and for the suggestions always delivered in a manner that raise equality and collaboration, making our discussions feel more like a conversation among peers. Thanks, because nothing of what came and will come in this academic journey, would have been without his help.

Heartfelt thanks to my beloved Cristina, for giving me those gentle pushes I always need, like only she can do, when only she can understand it's necessary. Thanks for accompanying me so closely, as everything, every thing would have been and will be more difficult and a little more unbearable without her.

Thanks to my family, for the opportunity they gave me to live and study for five years in another city, demonstrating in thousands of way their love, and always accepting the path I've chosen for myself, even if that means to leave them.

Thanks to all my friends, aware of having such a shitty and cat-like friend who disappears for weeks or months, but never making me feel bad about it, and being always welcoming when I really need.

Thanks to Filippo, for the practical help he's always available to give and for the care to my thesis and general academic career he's always demonstrated. And thanks, above all, for that gift he promised, pushing me to go forward with this work.