Complex & Intelligent Systems (2024) 10:5005–5031
https://doi.org/10.1007/s40747-024-01422-2

ORIGINAL ARTICLE

A novel Voronoi-based convolutional neural network framework for pushing person detection in crowd videos

Ahmed Alia[1,2,3] · Mohammed Maree[4] · Mohcine Chraibi[1] · Armin Seyfried[1,2]

Received: 22 August 2023 / Accepted: 12 March 2024 / Published online: 15 April 2024

Abstract

Analyzing the microscopic dynamics of pushing behavior within crowds can offer valuable insights into crowd patterns and interactions. By identifying instances of pushing in crowd videos, a deeper understanding of when, where, and why such behavior occurs can be achieved. This knowledge is crucial to creating more effective crowd management strategies, optimizing crowd flow, and enhancing overall crowd experiences. However, manually identifying pushing behavior at the microscopic level is challenging, and the existing automatic approaches cannot detect such microscopic behavior. Thus, this article introduces a novel automatic framework for identifying pushing in videos of crowds on a microscopic level. The framework comprises two main components: (i) feature extraction and (ii) video detection. In the feature extraction component, a new Voronoi-based method is developed for determining the local regions associated with each person in the input video. Subsequently, these regions are fed into EfficientNetV1B0 Convolutional Neural Network to extract the deep features of each person over time. In the second component, a combination of a fully connected layer with a Sigmoid activation function is employed to analyze these deep features and annotate the individuals involved in pushing within the video. The framework is trained and evaluated on a new dataset created using six real-world experiments, including their corresponding ground truths. The experimental findings demonstrate that the proposed framework outperforms state-of-the-art approaches, as well as seven baseline methods used for comparative analysis.

Keywords Artificial intelligence · Deep learning · Convolutional neural network · Intelligent video and image analytics · Intelligent systems · Pushing detection · Crowd dynamics

✉ Ahmed Alia
a.alia@fz-juelich.de
https://www.fz-juelich.de/profile/alia_a

Mohammed Maree
mohammed.maree@aaup.edu
https://www.aaup.edu/mohammed.maree

Mohcine Chraibi
m.chraibi@fz-juelich.de
https://www.fz-juelich.de/profile/chraibi_m

Armin Seyfried
a.seyfried@fz-juelich.de
https://www.fz-juelich.de/profile/seyfried_a

1 Institute for Advanced Simulation, Forschungszentrum Jülich, 52425 Jülich, Germany

2 Faculty of Architecture and Civil Engineering, University of Wuppertal, 42285 Wuppertal, Germany

3 Department of Information Technology, Faculty of Engineering and Information Technology, An-Najah National University, Nablus, Palestine

4 Department of Information Technology, Faculty of Information Technology, Arab American University, Jenin, Palestine

Introduction

With the rapid development of urbanization, the dense crowd has become widespread in various locations, such as religious sites, train stations, concerts, stadiums, malls, and famous tourist attractions. In such highly dense crowds, pushing behavior can easily arise, which may further increase crowd density. This could pose a threat not only to people's comfort but also to their safety [1–4]. People may start pushing for different reasons. (1) Saving their lives in emergencies or tense scenarios [5–7]. (2) Grabbing a limited resource, such as gaining access to a crowded subway train [8, 9]. (3) Accessing a venue more quickly; for instance, in crowded event entrances, some pedestrians start pushing others to enter the event faster [10–12]. The focus of this article is the pushing that occurs in crowded event entrances due to the availability of public real-world experiments about such entrances. In this context, Lügering et al. [10] defined pushing as "a behavior that can involve using arms, shoulders, or elbows; or simply the upper body, in which one person actively applies force to another person (or people) to overtake, while shifting their direction to the side or back, or force them to move forward more quickly". Additionally, using gaps in the crowd is considered as a strategy of pushing because it is a form of overtaking [10]. For more clarity, the definition of pushing behavior adopted in this article, published in 2022, describes it as a tactic pedestrians use to move forward more quickly through dense crowds [10], rather than as a strategy for fighting [13]. Understanding the microscopic dynamics of pushing plays a pivotal role in effective crowd management, to help safeguard the crowd from tragedies and promoting overall well-being [1, 14–17]. The study [10] has introduced a manual rating system to understand pushing dynamics at the microscopic level. The method relies on two trained psychologists to classify pedestrians' behaviors over time in a video of crowds into pushing or non-pushing categories, helping to know when, where, and why pushing behavior occurs. However, this manual method is time-consuming, tedious and prone to errors in some scenarios. Additionally, it requires trained observers, which may not always be feasible. Consequently, an increasing demand is for an automatic approach to identify pushing at the microscopic level within crowd videos. Detecting pushing behavior automatically is a demanding task that falls within the realm of computer vision. This challenge arises from several factors, such as dense crowds gathering at event entrances, the varied manifestations of pushing behavior, and the significant resemblance and overlap between pushing and non-pushing actions.

Recently, machine learning algorithms, particularly Convolutional Neural Network (CNN) architectures, have shown remarkable success in various computer vision tasks, including face recognition [18], object detection [19–21], image classification [22] and abnormal behavior detection [23]. One of the key reasons for this success is that CNN can learn the relevant features [24–26] automatically from data without human supervision [27, 28]. As a result of CNN's success in abnormal behavior detection, which is closely related to pushing detection, some studies have started to automate pushing detection using CNN models [29–31]. For instance, Alia et al. [29] introduced a deep learning framework that leverages deep optical flow and CNN models for pushing patch detection in video recordings. Another study [30] introduced a fast hybrid deep neural network model based on GPU to enhance the speed of video analysis and pushing patch identification. Similarly, the authors of [31] developed an intelligent framework that combines deep learning algorithms, a cloud environment, and live camera stream technology to annotate the pushing patches in real-time from crowds accurately. Yet, the current automatic methods focus on identifying pushing behavior at the level of regions (macroscopic level) rather than at the level of individuals (microscopic level), where each region can contain a group of persons. For more clarity, "patch level" refers to identifying regions that contain at least one person engaged in pushing behavior. In contrast, "individual level" refers to detecting the persons joining in pushing. Figure 1a shows a visualized example that demonstrates the identification of pushing behavior based on levels of patch and individual. In other words, the automatic approaches reported in the literature can not detect pushing at the microscopic level, limiting their contributions to help comprehend pushing dynamics in crowds. For example, they cannot accurately determine the relationship between the number of individuals involved in pushing behavior and the onset of critical situations, thereby hindering a precise understanding of when a situation may escalate to a critical level.

To overcome the limitations of the aforementioned methods, this article introduces a novel Voronoi-based CNN framework for automatically identifying individuals engaging in pushing behavior in crowd video recordings, using a single frame every second. It requires two types of input: the crowd's video recordings and individuals' trajectory data. The proposed framework comprises two components: feature extraction and detection. The first component utilizes a novel Voronoi-based EfficientNetV1B0 CNN architecture for feature extraction. The Voronoi-based method [32] integrates the Voronoi Diagram, Convex Hull [33], and a new dummy point generation technique to identify the local region of each person every second in the input video. The boundaries of local regions are determined based on the input pedestrian trajectory data. Subsequently, the EfficientNetV1B0 (excluding its original multiclass classification part) model [34] is used to extract deep features from these regions. In this article, the local region is defined as the zone focusing only on a single person (target person), including his surrounding spaces and physical interactions with his direct neighbors. This region is crucial in guiding the proposed framework to focus on microscopic behavior. On the other hand, the second component utilizes a fully connected layer coupled with a Sigmoid activation function to create a binary classification working instead of the original multi-class classification part in the EfficentNetV1B0. This adaptation is crucial for processing deep features and effectively differentiating between pushing and non-pushing behaviors in individuals. Finally, the adapted EfficientNetV1B0 is trained from scratch on a dataset of labeled local regions generated from six real-world video experiments with their ground truths [35].

-The main contributions of this work are summarized as follows:

1. To the best of our knowledge, this article introduces the first framework for automatically identifying pushing behavior at the individual level in videos of human crowds. In contrast, previous works in the literature [29–31] have focused on detecting such behavior at the patch

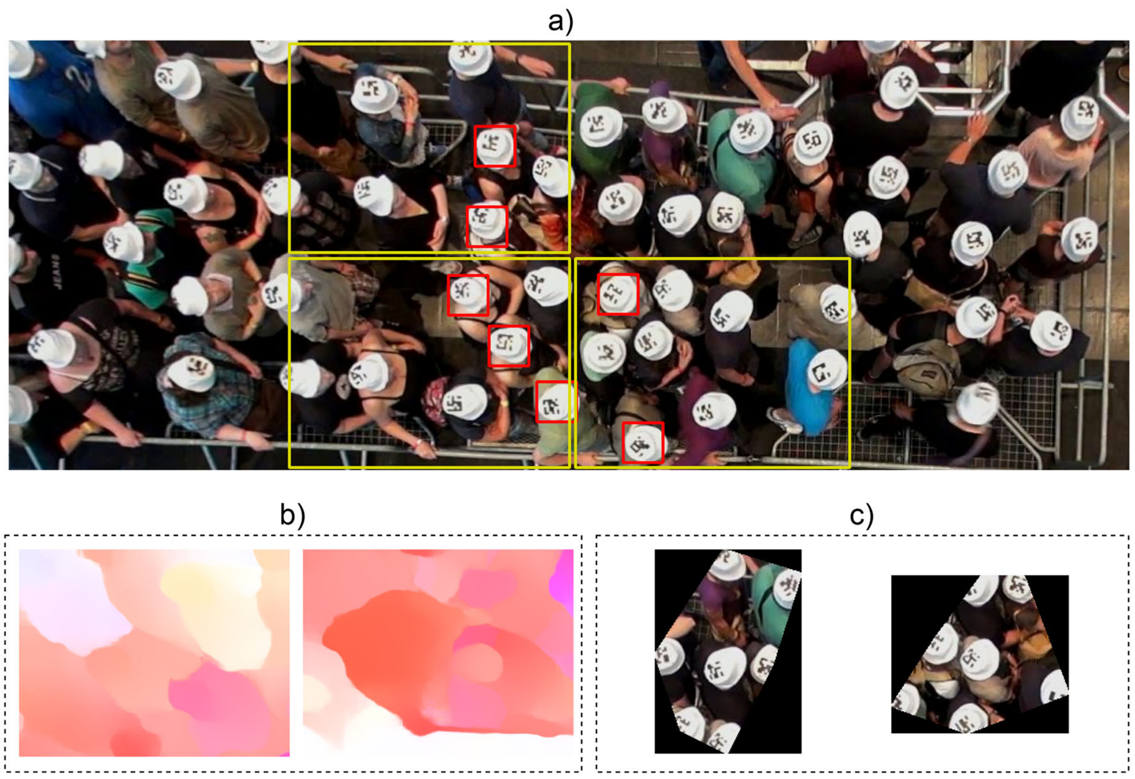

Fig. 1 Illustrated examples: **a** annotated frame with yellow rectangles, each representing a patch, showing an example of pushing behavior detection at the patch level. Annotated frame with red squares, illustrating pushing behavior detection at the individual level. **b** Examples of motion information map samples. **c** Examples of local region samples

level. Figure 1a provides a visualization of detection methods at individual and patch levels.

2. The framework utilizes a novel Voronoi-based approach with a trained and adapted EfficientNetV1B0-based CNN model. The novel Voronoi-based approach incorporates the Voronoi Diagram, Convex Hull, and a new dummy point generation technique.
3. The article introduces a new dataset comprising both pushing and non-pushing local regions, derived from six real-world experiments, each with corresponding ground truths. It is important to note that this dataset differs from those in previous works [29, 31]. In the current dataset, the samples consist of local regions. In contrast, in the previous datasets, each sample represents a visual motion information map of the crowd within a specific patch—a region ranging from 1.2 to 2.4 m^2 on the ground—and a specific duration, Fig. 1b and c show examples of motion information maps and local region samples. Additionally, the size of the new dataset is three times larger than that of the previous datasets.

The remainder of this article is organized as follows. "Related work" reviews some automatic approaches to abnormal behavior detection in videos of crowds. The architecture of the proposed framework is introduced in "Proposed framework architecture". "Training and evaluation metrics" presents the processes of training and evaluating the framework. "Experimental results and discussion" discusses experimental results and comparisons. Finally, the conclusion and future work are summarized in "Conclusion and future work".

Related work

This section begins by providing an overview of Voronoi Diagrams-based Deep Learning in Computer Vision. Subsequently, it explores CNN-based approaches for automatic video analysis and detecting abnormal behavior in crowds. The discussion then shifts to techniques specifically designed for automatically identifying pushing incidents within crowd videos.

Voronoi diagram-based deep learning in computer vision

A Voronoi diagram partitions a plane into regions based on the distance to a set of points, known as seeds. Each region is defined by all points closer to its corresponding seed than any other (for more details about Voronoi diagram, see "Voronoi-based local region extraction", direct neighbor identification step). Recently, integrating Voronoi diagrams with deep learning algorithms has led to the development of efficient applications in various computer vision tasks. These tasks include visual-query-based retrieval systems over image datasets [36], object detection [37], synthetic dataset generation [38], and image classification [39, 40].

CNN-based abnormal behavior detection

Typically, behavior is considered abnormal when it is seen as unusual in specific contexts. This implies that the definition of abnormal behavior depends on the situation [41]. To illustrate, running inside a bank might be considered abnormal behavior, while running at a traffic light could be viewed as normal [42]. Several behaviors have been automatically addressed in abnormal behavior detection applications in crowds, including walking in the wrong direction [43], running away [44], sudden people grouping or dispersing [45], human falls [46], suspicious behavior, violent acts [47], abnormal crowds [48], hitting, and kicking [13].

Tay et al. [41] developed a CNN-based method for identifying abnormal activities from videos. The researchers specifically designed and trained a customized CNN to extract features and label samples, using a dataset comprising both normal and abnormal samples. Similarly, study [49] introduced a novel CNN-based system for detecting abnormal crowd behavior in indoor and outdoor settings. This system leverages the strengths of pre-trained DenseNet121 and EfficientNetV2 models, which were fine-tuned for enhanced feature extraction. Subsequently, the study introduced innovative modifications to multistage and multicolumn models, specifically tailored to identifying various crowd behaviors, including sudden motion changes, panic events, and human flock movement. A new method using CNNs has been developed in [50] for the real-time detection of abnormal situations, such as violent behaviors. This method comprises the Convolutional Block Attention Module combined with the ResNet50 architecture to enhance feature extraction. In [51], the authors proposed a Densely Connected Convolutional Neural Network (DenseNet121)-based approach to identify abnormal behaviors in surveillance video feeds, achieving near real-time performance. Almahadin et al. [52] have developed an innovative model to identify abnormal behaviors in video sequences of crowded scenes. They integrated convolutional layers, Long Short-Term Memory networks, and a sigmoid-based output layer to extract spatiotemporal features and detect abnormal behaviors effectively. Nevertheless, constructing accurate CNNs requires a substantial training dataset, often unavailable for many human behaviors.

To address the limited availability of large datasets containing both normal and abnormal behaviors, some researchers have employed one-class classifiers using datasets that exclusively consist of normal behaviors. Creating or acquiring a dataset containing only normal behavior is comparatively easier than obtaining a dataset that includes both normal and abnormal behaviors. [53, 54]. The fundamental concept behind the one-class classifier is to learn exclusively from normal behaviors, thereby establishing a class boundary between normal and undefined (abnormal) classes. For example, Sabokrou et al. [53] utilized a pre-trained CNN to extract motion and appearance information from crowded scenes. They then employed a one-class Gaussian distribution to build the classifier, utilizing datasets of normal behavior. Similarly, in [54, 55], the authors constructed one-class classifiers by leveraging a dataset composed exclusively of normal samples. In [54], Xu et al. employed a convolutional variational autoencoder to extract features, followed by the use of multiple Gaussian models to detect abnormal behavior. Meanwhile, in [55], a pre-trained CNN model was employed for feature extraction, while one-class support vector machines were utilized for detecting abnormal behavior. Another study by Ilyas et al. [56] conducted a separate study where they utilized a pre-trained CNN along with a gradient sum of the frame difference to extract meaningful features. Subsequently, they trained three support vector machines on normal behavior data to identify abnormal behaviors. In general, one-class classifiers are frequently employed when the target behavior class or abnormal behavior is rare or lacks a clear definition [57]. However, pushing behavior is well-defined and not rare, particularly in high-density and competitive situations. Furthermore, this type of classifier considers new normal behavior as abnormal.

To address the limitations of CNN-based and one-class classifier approaches, multiple studies have explored the combination of multi-class CNNs with one or more handcrafted feature descriptors [23, 56]. In these hybrid approaches, the descriptors are employed to extract valuable information from the data. Subsequently, CNN learns and identifies relevant features and classifications based on the extracted information. For instance, Duman et al. [42] employed the classical Farnebäck optical flow method [58] and CNN to identify abnormal behavior. They used Farnebäck and CNN to estimate direction and speed information and then applied a convolutional long short-term memory network to build the classifier. Hu et al. [59] employed a combination of the histogram of gradient and CNN for feature extraction, while a least-squares support vector was used for classification. Direkoglu [23] utilized the Lucas-Kanade optical flow method and CNN to extract relevant features and identify “escape and panic behaviors”. Almazroey et al. [60] used Lucas–Kanade optical flow, a pre-trained CNN, and feature selection methods (specifically neighborhood component analysis) to extract relevant features. These extracted features were then used to train a support vector machine classifier. A framework to analyze video sequences in large-scale Hajj crowds was proposed by Aldayri et al. [61]. It integrates convolution operations, Convolutional Long Short-Term Memory, and Euclidean Distance to achieve its objectives.

Hybrid-based approaches could be more suitable for automatically detecting pushing behavior due to the limited availability of labeled pushing data. Nevertheless, most of the reviewed hybrid-based approaches for abnormal behavior detection may be inefficient for detecting pushing since (1) the descriptors used in these approaches can only extract limited essential data from high-density crowds to represent pushing behavior. (2) Some CNN architectures commonly utilized in these approaches may not be effective in dealing with the increased variations within pushing behavior (intra-class variance) and the substantial resemblance between pushing and non-pushing behaviors (high inter-class similarity), which can potentially result in misclassification.

CNN-based pushing behavior detection

In more recent times, a few approaches that merge effective descriptors with robust CNN architectures have been developed for detecting pushing regions in crowds. For example, Alia et al. [29] introduced a hybrid deep learning and visualization framework to aid researchers in automatically detecting pushing behavior in videos. The framework combines deep optical flow and visualization methods to extract the visual motion information from the input video. This information is then analyzed using an EfficientNetV1B0-based CNN and false reduction algorithms to identify and label pushing patches in the video. The framework has a drawback in terms of speed, as the motion extraction process is based on a CPU-based optical flow method, which is slow. Another study [30] presented a fast hybrid deep neural network model that labels pushing patches in short videos lasting only two seconds. The model is based on an EfficientNetB1-based CNN and GPU-based deep optical flow.

To support the early detection of pushing patches within crowds, the study [31] presented a cloud-based deep learning system. The primary goal of such a system is to offer organizers and security teams timely and valuable information that can enable early intervention and mitigate hazardous situations. The proposed system relies mainly on a fast and accurate pre-trained deep optical flow, an adapted version of the EfficientNetV2B0-based CNN, a cloud environment and live stream technology. Simultaneously, the optical flow model extracts motion characteristics of the crowd in the live video stream, and the classifier analyzes the motion to label pushing patches directly on the stream. Moreover, the system stores the annotated data in the cloud storage, which is crucial to assist planners and organizers in evaluating their events and enhancing their future plans.

To the best of our knowledge, current pushing detection approaches in the literature primarily focus on identifying pushing at the patch level rather than at the individual level. However, identifying the individuals involved in pushing would be more helpful for understanding the pushing dynamics. Hence, this article introduces a new framework for detecting pushing individuals in videos of crowds. The following section provides a detailed discussion of the framework.

Proposed framework architecture

This section describes the proposed framework for automatic pushing person detection in videos of crowds. As depicted in Fig. 2, there are two main components: feature extraction and detection. The first component extracts the deep features from each individual’s behavior. In contrast, the second component analyzes the extracted deep features and annotates the pushing persons within the input video. The following sections will discuss both components in more detail.

Feature extraction component

This component aims to extract deep features from each individual’s behavior, which can be used to classify pedestrians as pushing or non-pushing. To accomplish this, the component consists of two modules: Voronoi-based local region extraction and EfficientNetV1B0-based deep feature extraction. The first module selects a frame from the input video every second and identifies each person’s local region based on the pedestrian trajectory data within those extracted frames. Subsequently, the second module extracts deep features from each local region and feeds them to the next component for pedestrian detection. Before diving into these modules, let us define the local region term at one frame.

A frame f_t is captured every second from the input video. Here, t represents the timestamp, in seconds, since the start of the video and can range from 1 to T, where T is the total duration of the video in seconds. We can analyze individual pedestrians within each of these frames, such as f_t. The positions are given by discrete trajectories that assign a position $\langle x, y\rangle_i$ to each person i at frame f_t: $(\{[i, t, x, y]\}_{t=1}^{T})$. This revision clarifies that: Let $\mathcal{N}_i$ denote the set of pedestrians whose Voronoi cells are adjacent to that of pedestrian i. Specifically, pedestrian j belongs to $\mathcal{N}_i$ if and only if their

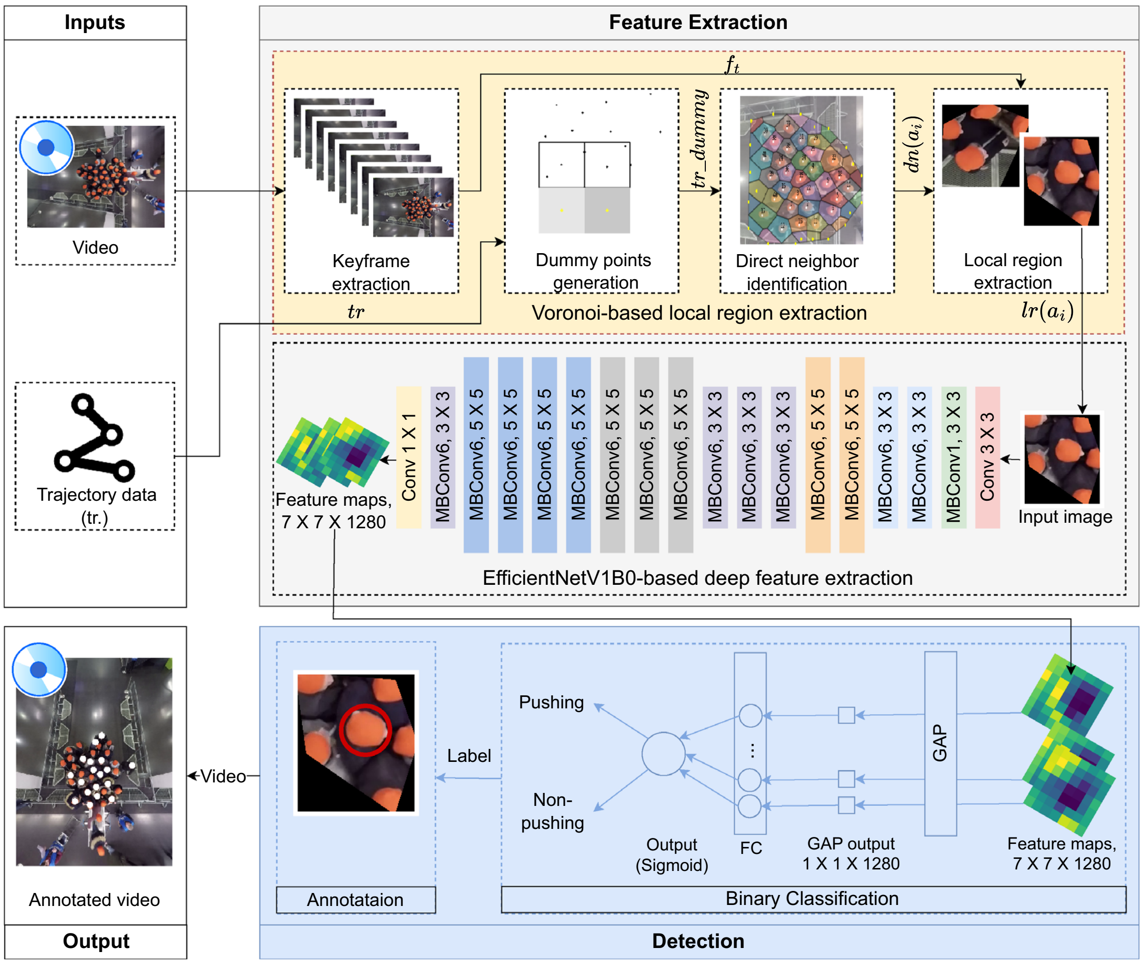

Fig. 2 The architecture of the proposed framework. In f_t, f signifies an extracted frame, while t indicates its timestamp in seconds, counted from the beginning of the input video (with t taking values like $1, 2, 3, \ldots$). For a target person i at f_t, $\mathcal{L}_i(f_t)$ denotes the local region, while $\mathcal{N}_i(f_t)$ represents the direct neighbors of i at f_t. The input trajectory data denoted by tr assists the Voroni-based local region extraction method in identifying the boundaries of each $\mathcal{L}_i(f_t)$. The term tr_dummy refers to the data that includes the generated dummy points and those in tr. FC stands for fully connected layer, while GAP refers to global average pooling

Voronoi cells share a boundary. The local region for pedestrian i at f_t, $\mathcal{L}_i$, forms a two-dimensional closed polygon, defined by the positions of all pedestrians in $\mathcal{N}_i$. As illustrations, Fig. 3a provides examples of both $\mathcal{N}_i$ (left image) and $\mathcal{L}_i$ (right image).

The region $\mathcal{L}_i$ encapsulates the crowd dynamics around individual i, reflecting potential interactions between i and its neighbors $\mathcal{N}_i$. Notably, the characteristics around a pushing individual might diverge from those around a non-pushing one, a distinction pivotal for highlighting pushing behaviors. Figure 3b showcases examples of such $\mathcal{L}_i$ regions for pushing and non-pushing individuals. The following section introduces a novel method for extracting $\mathcal{L}_i$.

Voronoi-based local region extraction

This section presents a novel method for extracting the local regions of pedestrians from the input video over time t. Besides the input video recordings, this method requires trajectory data $\{[i, t, x, y]\}_{t=1}^{T}$ to determine the coordinates $\langle x, y\rangle_j$ at frame f_t, which aids in identifying the corners of the polygonal local region $\mathcal{L}_i$ at f_t. The technique consists of several steps: frame extraction, dummy points generation, direct neighbor identification, and local region extraction.

Based on the definition of $\mathcal{L}_i$ presented earlier, the determination of each i's regional boundary is contingent upon $\mathcal{N}_i$ at f_t ($\mathcal{N}_i(f_t)$). Nonetheless, this definition might not always guarantee the inclusion of every i within their respective

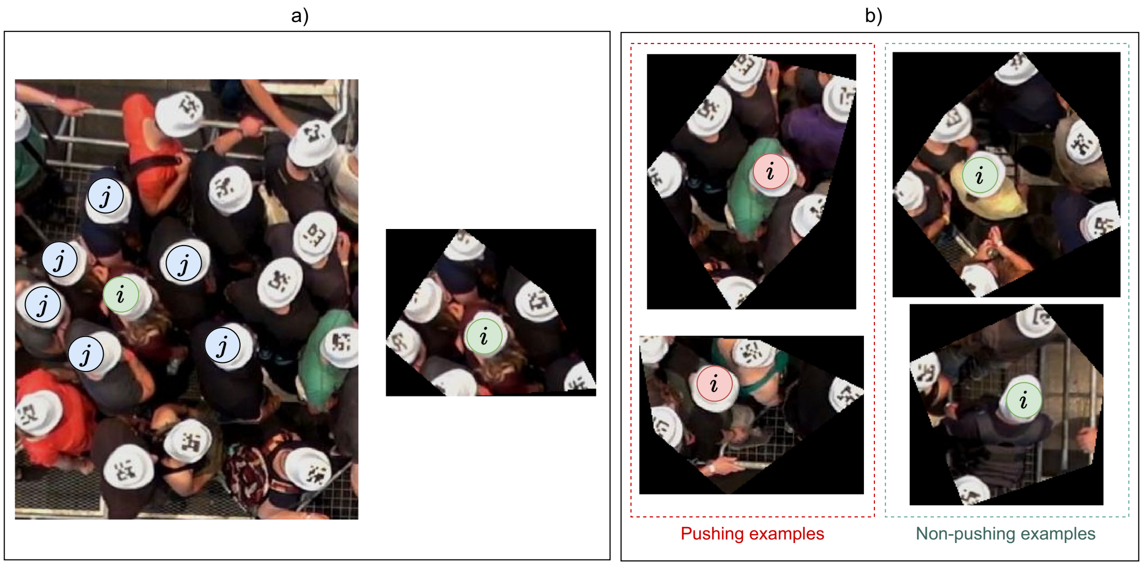

Fig. 3 An illustration of direct neighbors (**a**) and examples of local regions (**b**). The red circles represent individuals engaged in pushing, while the green circles represent individuals not involved in pushing. Direct neighbors j of a person i are indicated with blue circles

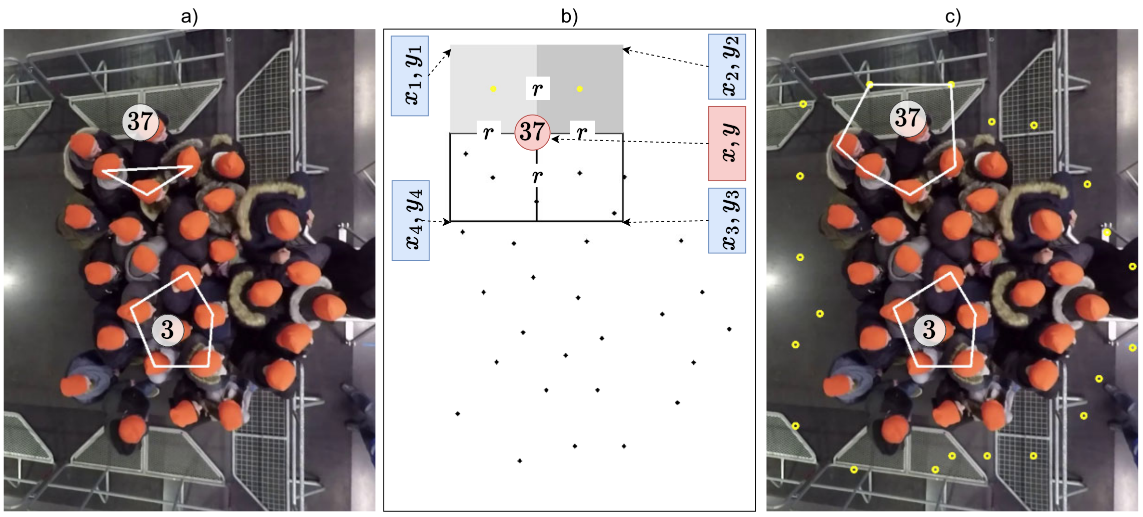

Fig. 4 An illustration of the effect of dummy points on creating the local regions, as well as a sketch of the dummy points generation technique. **a** $\mathcal{L}_{37}$ and $\mathcal{L}_3$ without dummy points. **b** A sketch of the dummy points generation technique. **c** $\mathcal{L}_{37}$ and $\mathcal{L}_3$ with dummy points. The white polygon represents the border of the local regions. Yellow small circles refer to the generated dummy points, while black points in b denote the positions of pedestrians. r is the dimension of each square

local region. This can be particularly evident when i at f_t lacks neighboring points from all directions, exemplified by person 37 in Fig. 4a. To address this issue, we introduce a step to generate dummy points. This involves adding points around each i at f_t in areas where they lack direct neighbors. This ensures every i remains encompassed within their local regions, as illustrated by person 37 in Fig. 4c. For this purpose, as depicted in Fig. 4b and Algorithm 1, firstly, this step involves reading the trajectory data of i that corresponds to f_t (Algorithm 1, lines 1–8). Concurrently, the area surrounding every i is divided into four equal square regions, each can accommodate at least one i (Algorithm 1, lines 9–17). The location $\langle x, y\rangle_i$ corresponds to the first 2D coordinate of each region (Algorithm 1, lines 12–13). In contrast, the remaining 2D coordinates ($\langle x1, y1\rangle$,$\langle x2, y2\rangle$,$\langle x3, y3\rangle$, $\langle x4, y4\rangle$) required for identifying the regions can be deter-

Algorithm 1 Pseudo code for generating dummy points.

```
Inputs:
    $tr = \{row_1, row_2, row_3, \ldots\}$,        // A file of pedestrian trajectory data
    where each $row$ =[pedestrian Id, frame order in the corresponding video, x-coordinate of pedestrian Id, y-coordinate of pedestrian Id]
    $fps$: the frame rate in the corresponding video
    $r$: the dimension of each square region in pixel unit
Outputs:
    // A file of pedestrian trajectory data with dummy points
    $tr\_dummy = tr \cup \{row_1, row_2, row_3, \ldots\}$
    where $row$ =["dummy point", frame order in the corresponding video, x-coordinate of the dummy point, y-coordinate of the dummy point]

1: $file \leftarrow$ open($tr$)
2: $file\_dummy \leftarrow$ open($tr\_dummy$)
3: while not EOF($file$) do
4:     $rec \leftarrow$ read($file$)
5:     if $rec[1]\,\%\,fps = 0$ then
6:         write(rec) to $file\_dummy$
7:     end if
8: end while
9: $regions \leftarrow [[\,]]$
10: while not EOF($file\_dummy$) do
11:     $rec \leftarrow$ read($file\_dummy$)
12:     $x \leftarrow rec[2]$
13:     $y \leftarrow rec[3]$
14:     append ($[x - r, y + r]$) to regions
15:     append ($[x + r, y + r]$) to regions
16:     append ($[x + r, y - r]$) to regions
17:     append ($[x - r, y - r]$) to regions
18:     while $corner$ in $regions$ do
19:         if empty(area($[x, y], corner$)) then
20:             $dummy\_point \leftarrow \left[\frac{x+corner[0]}{2}, \frac{y+corner[1]}{2}\right]$
21:             $dumy\_rec \leftarrow [0, rec[0], dummy\_point[0], dummy\_point[1]]$
22:             append ($dumy\_rec$) to $file\_dummy$
23:         end if
24:     end while
25: end while
26: $tr$.close()
27: $tr\_dummy$.close()
```

mined by:

$$\begin{aligned}\langle x1, y1\rangle &= \langle x - r, y + r\rangle\\ \langle x2, y2\rangle &= \langle x + r, y + r\rangle\\ \langle x3, y3\rangle &= \langle x + r, y - r\rangle\\ \langle x4, y4\rangle &= \langle x - r, y - r\rangle,\end{aligned} \tag{1}$$

where r is the dimension of each square region. Subsequently, each region is checked to verify if it has any pedestrians. In case a region is empty, a dummy point in its center is appended to the input trajectory data. Figure 4b illustrates an example of four regions surrounding person 37 and two dummy points (yellow dots in first and second empty regions), see Algorithm 1, lines 18–24. After generating the dummy points for all i at f_t, the trajectory data is forwarded to the next step, direct neighbor identification. Figure 4c shows a crowd with dummy points in a single f_t.

The third step, direct neighbor identification, employs a combination of Voronoi Diagram [32] and Convex Hull [33] to find $\mathcal{N}_i(f_t)$ from the input trajectory data with dummy points. A Voronoi Diagram is a method for partitioning a plane into several polygonal regions (named Voronoi cells $\mathcal{V}$s) based on a set of objects/points (called sites) [32]. Each $\mathcal{V}$ contains edges and vertices, which form its boundary. Figure 5a depicts an example of a Voronoi Diagram for 51 $\mathcal{V}s$ of 51 sites, where black and yellow dots denote the sites. In the same figure, the set of sites contains $\langle x, y\rangle_i$ (dummy points are included) at a specific f_t, then each $\mathcal{V}_i$ includes only one site $\langle x, y\rangle_i$, and all points within $\mathcal{V}_i$ are closer to site $\langle x, y\rangle_i$ than any other sites $\langle x, y\rangle_q$. Where $q \in$ all i at that f_t, and $q \neq i$.

Furthermore, $\mathcal{V}_i$ and $\mathcal{V}_q$ at f_t are considered adjacent if they share at least one edge or two vertices. For instance, as seen in Fig. 5, $\mathcal{V}_4$ and $\mathcal{V}_{34}$ are adjacent, while $\mathcal{V}_4$ and $\mathcal{V}_3$ are not adjacent. Since the Voronoi Diagram contains unbounded cells, determining the adjacent cells for each $\mathcal{V}_i$ at f_t may yield inaccurate results. For instance, most cells of yellow points, which are located at the scene's borders, are

Algorithm 2 Pseudo code of direct neighbor identification step

Inputs:
tr_dummy={$row_1, row_2, row_3, \ldots$},
where each row =[dummy point or pedestrian Id, frame order in the corresponding video, x-coordinate point, y- coordinate]
Outputs:
$direct\_neighbor$={$row_1, row_2, row_3, \ldots$},
where each row =[frame order in the corresponding video, pedestrian Id, [direct neighbors (Ids) of pedestrian Id]]

```
1: $file \leftarrow$ open($tr\_dummy$)
2: $file\_dn \leftarrow$ open($direct\_neighbor$)
3: $data \leftarrow$ load($file$)
4: $frames \leftarrow$ unique($data$[:, 0])
5: while $fr$ in $frames$ do
6:     $data\_fr \leftarrow$ filter($data$ , $fr$)
7:     $vor\_diagram \leftarrow$ Voronoi($data\_fr$[:, 2 : 4])
8:     $CH \leftarrow$ ConvexHull($data\_fr$[:, 2 : 4])
9:     for each $region \in vor\_diagram$.regions do
10:        $vor\_diagram.region \leftarrow vor\_diagram.region \cap CH$
11:    end for
12:    $cells \leftarrow vor\_diagram$.regions
13:    while $i$ in $data\_fr$[:, 0] do
14:        $cell \leftarrow cells[i]$
15:        $dn\_cells \leftarrow$ find_direct_neighbor_cells ($cell$)
16:        $dn\_i \leftarrow dn\_cells$.sites
17:        $rec \leftarrow [fr, i, dn\_i]$
18:        write ($rec$) to $file\_dn$
19:    end while
20: end while
21: $file$.close()
22: $file\_dn$.close()
```

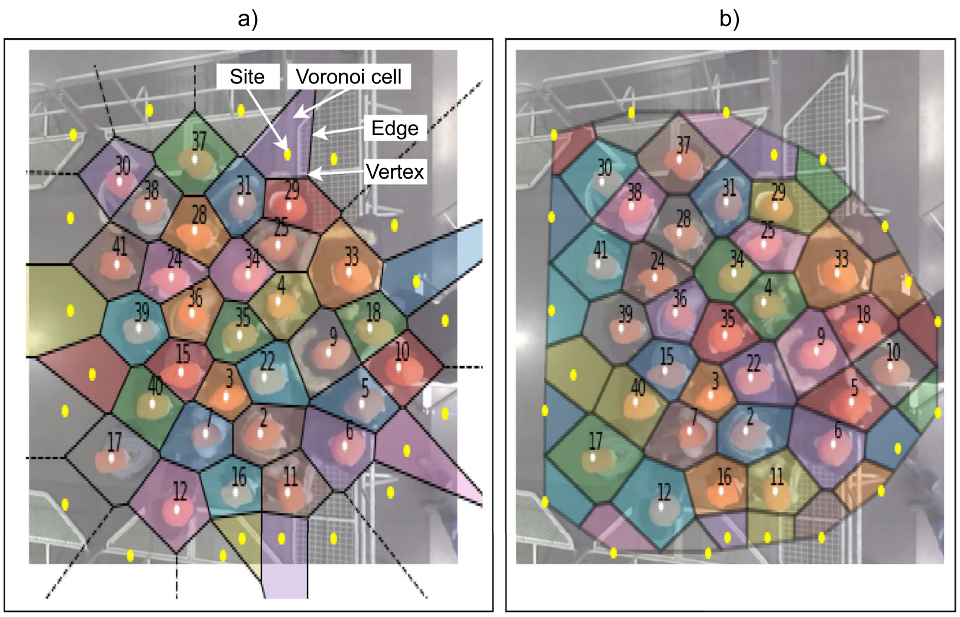

Fig. 5 **a** Example of a simple Voronoi decomposition. **b** Example of bounded Voronoi decomposition. Both are constructed using 30 pedestrian points and 21 dummy points

unbounded cells, as depicted in Fig. 5a. For further clarity, $\mathcal{V}_i(f_t)$ becomes unbounded when i is a vertex of the convex hull that includes all instances of i at f_t. As a result, the Voronoi Diagram may not provide accurate results when determining adjacent cells, which is a crucial factor in identifying $\mathcal{N}_i(f_t)$. To overcome such limitation, Convex Hull [33] is utilized to finite the Voronoi Diagram (unbounded cells) as shown in Fig. 5b. The Convex Hull is the minimum convex shape that encompasses a given set of points, forming a polygon that connects the outermost points of the set while ensuring that all internal angles are less than 180° [62]. For this purpose, the intersection of each $\mathcal{V}_i(f_t)$ with Convex Hull of all i at f_t is calculated, then the $\mathcal{V}_i(f_t)$ in the diagram are updated based on the intersections to obtain the bounded Voronoi Diagram of all i at f_t (Algorithm 2, lines 5–12). In more details, the Convex Hull of all i at f_t is measured (Algorithm 2, line 8). After that, the intersection between $\mathcal{V}_i(f_t)$ and the Convex Hull at f_t is computed. And finally, we update the Voronoi Diagram at each f_t using the calculated interactions to obtain the corresponding bounded one as shown in Fig. 5b (Algorithm 2, lines 8–11). After creating the bounded Voronoi Diagram, individuals in the direct adjacent Voronoi cells of $\mathcal{V}_i(f_t)$ are $\mathcal{N}_i(f_t)$, (Algorithm 2, lines 12–20). For example, in Fig. 5b, direct adjacent Voronoi cells of $\mathcal{V}_3$ at f_t are $\{\mathcal{V}_2, \mathcal{V}_{22}, \mathcal{V}_{35}, \mathcal{V}_{15}, \mathcal{V}_7\}$, and $\mathcal{N}_3 = \{2, 22, 35, 15, 7\}$.

The last step, local region extraction, aims to extract the local region of each i at f_t, where $i \notin$ dummy points. The step firstly finds $\mathcal{L}_i(f_t)$ based on each $\langle x, y\rangle_j$, where $j \in \mathcal{N}_i(f_t)$, Fig. 4c. Then, $\mathcal{L}_i(f_t)$ are cropped from corresponding f_t and passed to the next module, which will be discussed in the next section. Figure 3b displays examples of cropped local regions.

EfficientNetV1B0-based deep feature extraction

To extract the deep features from each individual's behavior, the feature extraction part of EfficientNetV1B0 is used over their local regions $\mathcal{L}_i(f_t)$. EfficientNetV1B0 is a CNN architecture that has gained popularity for various computer vision tasks due to its efficient use of resources and fewer parameters than other state-of-the-art models [34]. Furthermore, it has achieved high accuracy on multiple image classification datasets. Additionally, the experiments in this article ("Comparison with five baseline frameworks based on popular CNN architectures") indicate that combining EfficientNetV1B0 (without local classification part) with local regions yields the highest accuracy compared to other popular CNN architectures integrated with local regions. Therefore, EfficientNetV1B0's feature extraction part is employed to find more helpful information from each individual's behavior.

The architecture of the efficientNetV1B0-based deep feature extraction model is depicted in Fig. 2. Firstly, it applies a 3×3 convolution operation to the input image, a local region with dimensions of $224 \times 224 \times 3$. Following this, 16 mobile inverted bottleneck convolution (MBConv) blocks [63] are employed to extract deep features (feature maps) $\in \mathbb{R}^{7\times7\times320}$ from each $\mathcal{L}_i(f_t)$. In more detail, the MBConv blocks used consist of one MBConv1, 3×3, six MBConv6, 3×3, and nine MBConv6, 5×5. Figure 6 illustrates the structure of the MBConv block, which employs a 1×1 convolution operation to expand the depth of feature maps and capture more information. A 3×3 depthwise convolution follows this to decrease the computational complexity and number of parameters. Additionally, batch normalization and swish activation [64] are applied after each convolution operation.

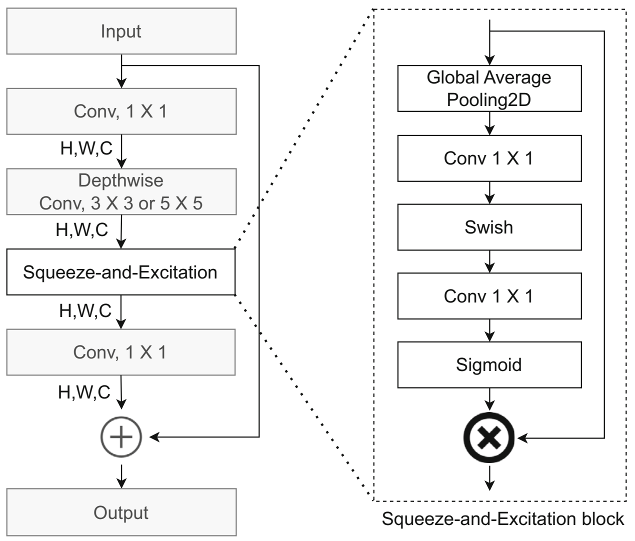

Fig. 6 The architecture of MBConv block

The MBConv then employs a Squeeze-and-Excitation block [65] to enhance the architecture's representation power. The Squeeze-and-Excitation block initially performs global average pooling to reduce the channel dimension. Then it applies an excitation operation with Swish [64] and Sigmoid [66] activations to learn channel-wise attention weights. These weights represent the significance of each feature map and are multiplied by the original feature maps to generate the output feature maps. After the Squeeze-and-Excitation block, another 1×1 convolution with batch normalization is used to reduce the output feature maps' dimensionality, resulting in the final output of the MBConv block.

The main difference between MBConv6 and MBConv1 is the depth of the block and the number of operations performed in each block; MBConv6 is six times that of MBConv1. Note that MBConv6, 5×5 performs the identical operations as MBConv6, 3×3, but MBConv6, 5×5 applies a kernel size of 5×5, while MBConv6, 3×3 uses a kernel size of 3×3.

Detection component

The primary objective of the detection component is to analyze the feature maps generated by the previous component and categorize individual behaviors as either pushing or non-pushing. This task requires a binary classification followed by an annotation process. Unfortunately, the classification part of the original EfficientNetV1B0 architecture is not designed for binary tasks and is thus unsuitable for identifying pushing behavior. Consequently, we have adapted the classification part of EfficientNetV1B0 to support binary classification. In addition to extracting deep features in the preceding component, this modification allows the EfficientNetV1B0 to

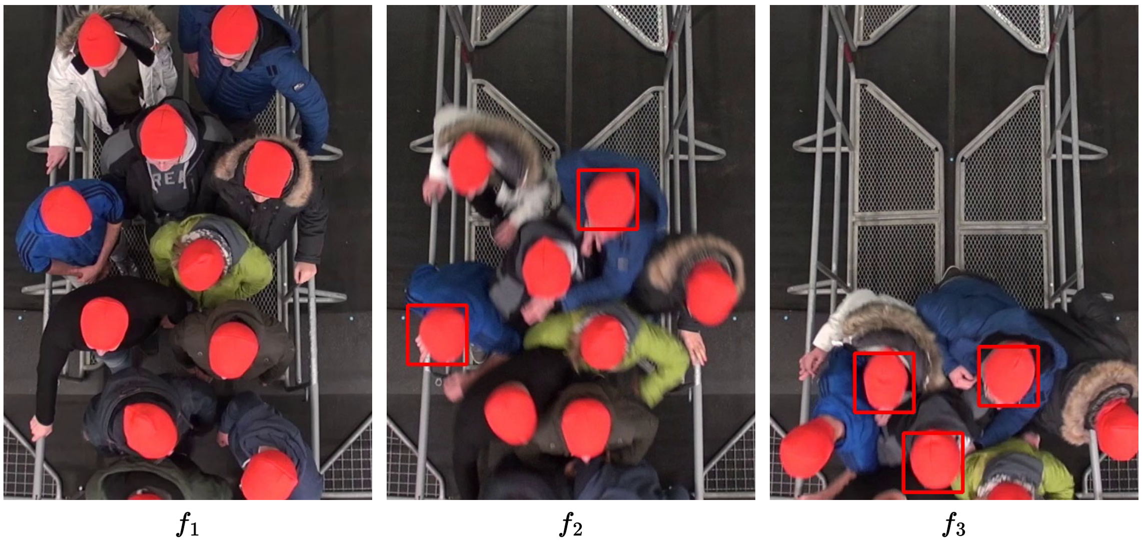

Fig. 7 A visualized example of a sequence of frames annotated with red rectangles highlighting the individuals participating in pushing. The original frames were taken from [35]

classify individual behaviors as pushing or non-pushing in the detection component. To perform the binary classification task, we combine a 1×1 convolution operation, 2D global average pooling, a fully connected layer with a single neuron, and a Sigmoid activation function. Figure 2 shows the classifier. To gain more information by increasing the number of channels in feature maps, the 1×1 convolutional operation is used. The new dimension of feature maps for each $\mathcal{L}_i(f_t)$ is $7 \times 7 \times 1280$. After that, the global average pooling2D is utilized to transform the feature maps to $1 \times 1 \times 1280$ and feed them to the fully connected layer with one neuron, which produces an output $x \in \mathbb{R}$. Subsequently, the Sigmoid function σ is applied to x, transforming it into a value between 0 and 1. The sigmoid function, commonly used in binary classification, is defined as Eq. (2):

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2}$$

where $\sigma(x)$ represents the probability of the pushing label for the corresponding i at f_t, and e denotes the mathematical constant known as Euler's number. Finally, the classifier uses a threshold value to identify the class of i at f_t as Eq. (3):

$$\text{Class}(i, f_t) = \begin{cases} \text{pushing} & \text{if } \sigma(x) \geq \text{threshold} \\ \text{non-pushing} & \text{if } \sigma(x) < \text{threshold} \end{cases} \tag{3}$$

By default, the threshold value for binary classification is configured to 0.5, which is suitable for datasets exhibiting a balanced distribution. Unfortunately, the new pushing dataset created in "A novel dataset preparation" for training and evaluating the proposed framework is imbalanced. As such, using the default threshold may lead to poor performance of the introduced trained classifier on that dataset [68]. Therefore, fine-tuning the threshold in the trained classifier becomes essential for improving accuracy across both pushing and non-pushing classes. The methodology for finding the optimal threshold for the classifier will be explained in detail in "Evaluation metrics and performance improvement". Following training and adjusting the classifier's threshold, it can categorize individuals i as pushing or non-pushing. At the same time, the annotation process draws a red rectangle around the head of each pushing person in the corresponding frames f_t (see Fig. 7) and finally generates an annotated video.

The following section will discuss the training and evaluating processes of the proposed framework.

Training and evaluation metrics

This section introduces a novel labeled dataset, as well as presents the parameter setups for the training process, evaluation metrics, and the methodology for improving the framework's performance on an imbalanced dataset.

A novel dataset preparation

Here, it is aimed at creating the labeled dataset for training and evaluating the proposed framework. The dataset consists of a training set, a validation set for the learning process, and two test sets for the evaluation process. These sets comprise $\mathcal{L}_i(f_t)$ labeled as pushing or non-pushing. In this context, each pushing $\mathcal{L}_i(f_t)$ means i at f_t contributes pushing, while every non-pushing $\mathcal{L}_i(f_t)$ indicates that i at f_t follows the

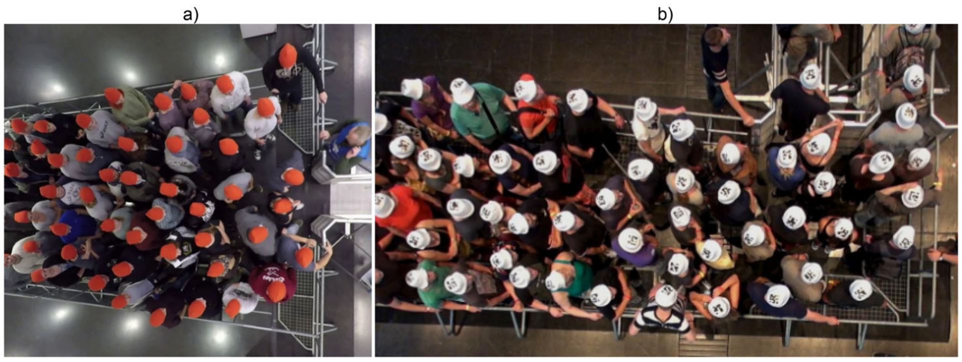

Fig. 8 Overhead view of exemplary experiments. **a** Experiment 270, as well as Experiments 50, 110, 150, and 280 used the same setup but with different widths of the entrance area ranging from 1.2 to 5.6 m based on the experiment [35]. **b** Experiment entrance_2 [67] The entrance gate's width is 0.5 m in all setups

Table 1 Characteristics of the chosen experiments

Video experiment[a]	Width (m)	Pedestrian total	Number of gates	Duration (s)	Resolution
50	4.5	42	1	37	1920 × 1440
110	1.2	63	1	53	1920 × 1440
150	5.6	57	1	57	1920 × 1440
270	3.4	67	1	59	1920 × 1440
280	3.4	67	1	67	1920 × 1440
Entrance_2	3.4	123	2	125	1920 × 1080

m meter, *s* second

[a]The same names as reported in [35, 67]

social norm of queuing. The following will discuss the data sources and methodology used to prepare the sets.

The dataset preparation is based on three data sources: (1) six videos of real-world experiments of crowded event entrances. (2) Pedestrian trajectory data. (3) Ground truths for pushing behavior. Six video recordings of experiments with their corresponding pedestrian trajectory data are selected from the data archive hosted by Forschungszentrum Jülich [35, 67]. This data is licensed under CC Attribution 4.0 International license. The experimental situations mimic the crowded event entrances, and static top-view cameras were employed to record the experiments with a frame rate of 25 frames per second. For more clarity, Fig. 8 shows overhead views of exemplary experiments, and Table 1 summarizes the various characteristics of the chosen experiments. Additionally, ground truth labels constructed by the manual rating system [10] are used for the last data source. In this system, social psychologists observe and analyze video experiments frame-by-frame to manually identify individuals who are pushing over time. The experts use PeTrack software [69] to manage the manual tracking process and generate the annotations as a text file. For further details on the manual system, readers can refer to Ref. [10].

Here, the methodology used for papering the dataset is described. As shown in Fig. 9, it consists of two phases: local region extraction; and local region labeling and set generation. The first phase aims to extract local regions (samples) from videos while avoiding duplicates. To accomplish this, the phase initially extracts frames from the input videos second by second. It employs After that the Voronoi-based local region extraction module to identify and crop the samples from the extracted frames based on the trajectory data. Table 2 demonstrates the number of extracted samples from each video, and Fig. 3b shows several examples of local regions. Preventing the presence of duplicate samples between the training, validation, and test sets is crucial to obtain a reliable evaluation for the model. Therefore, this phase removes similar and slightly different samples before proceeding to the next phase. It involves using a pre-trained MobileNet CNN model to extract deep features/embeddings from the samples and cosine similarity to find duplicate or near duplicate samples based on their features [70]. This technique is more robust than comparing pixel values, which can be sensitive to noise and lighting variations [71]. Table 2 depicts the number of removed duplicate samples.

On the other hand, the local region and set generation phase is responsible for labeling the extracted samples and producing the sets, including one training set, one validation set, and two test sets. This phase utilizes the ground truth label of each i at f_t to label the samples ($\mathcal{L}_i(f_t)$). If i at f_t contributing to pushing, $\mathcal{L}_i(f_t)$ is categorized as pushing; otherwise, it is classified as non-pushing. Examples of pushing samples can be found in Fig. 3b. The generated labeled dataset from all video experiments comprises 3384 pushing samples and 8994 non-pushing samples. The number of extracted pushing and non-pushing samples from each video is illustrated

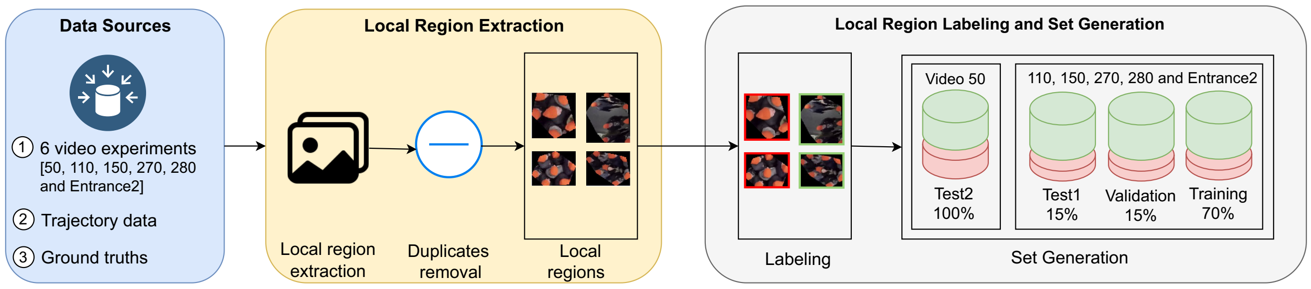

Fig. 9 Pipeline of dataset preparation. In the part 'Local Region Labeling and Set Generation', red refers to the pushing class and pushing sample, while the non-pushing class and non-pushing sample are represented in green. The local region extraction component uses trajectory data to determine the coordinates of each person

Table 2 Summary of the prepared sets

Video	Number of samples			Labeled dataset		Training set		Validation set		Test set1		Test set2	
	Original	Deleted	Distinct	P	NP	P	NP	P	NP	P	NP	P	NP
110	1046	1	1045	548	497	365	331	99	84	84	82		
150	1469	70	1399	625	774	455	547	83	113	87	114		
270	1627	11	1616	577	1039	401	727	84	161	92	151		
280	1822	44	1778	287	1491	213	1104	44	181	30	206		
Entrance_2	6204	325	5879	1030	4849	726	3403	156	715	148	731		
Total	12,168	451	11,717	3067	8650	2160	6112	466	1254	441	1284		
50[a]				317	344							317	344

[a] Video 50 is used exclusively for the evaluation process, while the remaining video experiments will be employed for both training and evaluation

in Table 2. After creating the labeled dataset, the sets are generated from the dataset. Specifically, the second phase randomly divides the extracted frames from video experiments 110, 150, 270, 280, and Entrance_2 into three sets: 70%, 15%, and 15% for training, validation, and test sets, respectively. Then, using these sets, it generates the training, validation, and test (test set 1) sets from the labeled corresponding samples ($\mathcal{L}_i(f_t)$). Another test set (test set 2) is also developed from the labeled samples extracted from the complete video experiment 50. Table 2 shows the summary of the generated sets.

To summarize, four labeled sets were created: the training set, which consists of 2160 pushing samples and 6112 non-pushing samples; the validation set, which contains 466 pushing samples and 1254 non-pushing samples; the test set 1, which includes 441 pushing samples and 1284 non-pushing samples; and the test set 2, comprising 317 pushing samples and 344 non-pushing samples. It's crucial to emphasize two key aspects where the new dataset significantly deviates from the datasets outlined in Refs. [29, 31]. First, the samples in the introduced dataset capture the crowd's appearance surrounding each pedestrian i ($\mathcal{L}_i(f_t)$). This contrasts with samples from previous datasets, which represent the visual motion information within a region ranging from 1.2 to 2.4 m^2 on the ground. While the previous datasets were focused on analyzing behavior at a patch level, the new dataset is better suited for studying pushing behavior at an individual level. Figure 1b and c provide examples of motion information maps and local region $\mathcal{L}_i$ samples, respectively. Second, the new dataset is significantly larger, containing 12,378 samples, compared to 3780 and 3941 samples in the datasets reported in Refs. [29, 31], respectively. In other words, considering the local region around a person also results in a larger dataset than the patch approach.

Parameter setup

Table 3 shows parameters used during the training process. The default values for the learning rate and batch size that are typically used in training CNN architectures on ImageNet in Keras were selected. Additionally, other parameters were fine-tuned through experimentation to achieve optimal performance with the new dataset. To prevent overfitting, the training was halted if the validation accuracy did not improve after 20 epochs.

Evaluation metrics and performance improvement

This section will discuss the metrics chosen for evaluating the performance of the proposed framework. Additionally,

Table 3 The hyperparameter values used in the training process

Parameter	Value
Optimizer	Adam
Loss function	Binary cross-entropy
Learning rate	0.001
Batch size	32
Epoch	100

it will explore the methodology employed to enhance the performance of the trained imbalanced classifier, thereby improving the overall effectiveness of the framework.

Given the imbalanced distribution of the generated local region dataset, the framework exhibits a bias toward the majority class (non-pushing). Consequently, it becomes crucial to employ appropriate metrics for evaluating the performance of the imbalanced classifier. As a result, a combination of metrics was adopted, including macro accuracy, True Pushing Rate (TPR), True Non-Pushing Rate (TNPR), and Area Under the receiver operating characteristic Curve (AUC) on both test set 1 and test set 2. The following provides a detailed explanation of these metrics.

TPR, also known as sensitivity, is the ratio of correctly classified pushing samples to all pushing samples, and it is defined as:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FNP}}, \tag{4}$$

where TP and FNP denote correctly classified pushing persons and incorrectly predicted non-pushing persons.

TNPR, also known as specificity, is the ratio of correctly classified non-pushing samples to all non-pushing samples, and it is described as:

$$\text{TNPR} = \frac{\text{TNP}}{\text{TNP} + \text{FP}}, \tag{5}$$

where TNP and FP stand for correctly classified non-pushing persons and incorrectly predicted pushing persons.

Macro accuracy, or balanced accuracy, is the average proportion of correct predictions for each class individually. This metric ensures that each class is given equal significance, irrespective of its size or distribution within the dataset. For more clarity, it is just the average of TPR and TNPR as:

$$\text{Macro accuracy} = \frac{\text{TPR} + \text{TNPR}}{2}. \tag{6}$$

AUC is a metric that represents the area under the Receiver Operating Characteristics (ROC) curve. The ROC curve illustrates the performance of a classification model across various threshold values. It plots the false positive rate (FPR) on the horizontal axis against the true positive rate (TPR) on the vertical axis. AUC values range from 0 to 1, where a perfect model achieves an AUC of 1, while a value of 0.5 indicates that the model performs no better than random guessing [72]. Figure 10a shows an example of a ROC curve with AUC value.

As mentioned above, the binary classifier employs a threshold to convert the calculated probability into a predicted class. The pushing class is predicted if the probability exceeds the threshold; otherwise, the non-pushing label is predicted. The default threshold is typically set at 0.5. However, this value leads to poor performance of the introduced framework because EfficientNetV1B0 and classification were trained on imbalanced dataset [68]. In other words, the default threshold yields a high TNPR and a low TPR in the framework. To address the imbalance issue and enhance the framework's performance, it becomes necessary to determine an optimal threshold that achieves a better balance between TPR and FPR (1-TNPR). To accomplish this, the ROC curve is utilized over the validation set to identify the threshold value that maximizes TPR and minimizes FPR. Firstly, TPR and TNPR are calculated for several thresholds ranging from 0 to 1. Then, the threshold that yields the minimum value for the following objective function (Eq. (7) is considered the optimal threshold:

$$\text{Objective function} = |\text{TPR} - \text{TNPR}|. \tag{7}$$

As shown in Fig. 10a, the red point refers to the optimal threshold of the classifier used in the proposed framework, which is 0.038.

Experimental results and discussion

Here, several experiments were conducted to evaluate the performance of the proposed framework. Initially, the performance of the proposed framework itself is assessed. Subsequently, It is compared with five other CNN-based frameworks. After that, two customized CNN architectures in the abnormal behavior detection field were used for further evaluation of the proposed framework. The influence of the deep feature extraction module on the proposed framework's performance is also investigated. Subsequently, the manuscript explores the impact of dummy points on the framework's performance. This is followed by a comparison with two state-of-the-art approaches for pushing behavior detection. All experiments and implementations were performed on Google Colaboratory Pro, utilizing Python 3 programming language with Keras, TensorFlow 2.0, and OpenCV libraries. In Google Colaboratory Pro, the hardware setup comprises an NVIDIA GPU with a 15 GB capacity and a system RAM of 12.7 GB. Moreover, the framework and all the baselines developed for comparison in the experiments

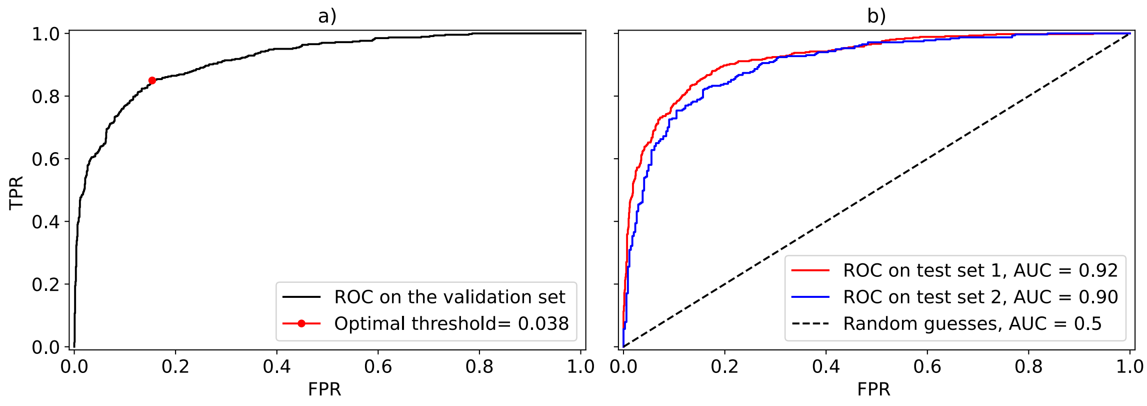

Fig. 10 ROC curves for the introduced framework. **a** ROC curve with an optimal threshold on the validation set. **b** ROC curves with AUC values on test set 1 and test set 2. *TPR* true pushing rate, *FPR* false pushing rate

were trained using the same sets (Table 2) and hyperparameter values (Table 3).

Performance of the proposed framework

The performance of the proposed framework was evaluated using the generated dataset (Table 2) and various metrics, including macro accuracy, TPR, TNPR, and AUC. We first trained the proposed framework's EfficientNetB0-based deep feature extraction module and detection component on the training and validation sets. Subsequently, the framework's performance on test set 1 and test set 2 were assessed.

Table 4 shows that the introduced framework, with the default threshold, obtained macro accuracy of 83%, TPR of 74%, and TNPR of 92% on test set 1. On the other hand, it achieved 82% macro accuracy, 88% TNPR, and 76% TPR on test set 2. However, it is clear that the TPR is significantly lower than the TNPR on both test sets, see Fig. 11a and c. To balance the TPR and TNPR and improve the TPR, the optimal threshold is 0.038, as shown in Fig. 10a. This threshold increases TPR by 12% and 7% on test set 1 and test set 2, respectively, without affecting the accuracy, see Fig. 11b and d. In fact, the framework's accuracy improved by 2% on test set 1. The ROC curves with AUC values for the framework on the two test sets are shown in Fig. 10b, with AUC values of 0.92 and 0.9 on test set 1 and test set 2, respectively.

To summarize, with the optimal threshold, the proposed framework achieved an accuracy of 85%, TPR of 86%, and TNPR of 84% on test set 1, while obtaining 82% accuracy, 81% TPR, and 83% TNPR on test set 2. The next section will compare the framework's performance with five baseline systems for further evaluation.

Comparison with five baseline frameworks based on popular CNN architectures

In this section, the results of further empirical comparisons are shown to evaluate the framework's performance against five baseline systems. Specifically, it explores the impact of the EfficientNetV1B0-based deep feature extraction module on the overall performance of the framework. To achieve this, EfficientNetV1B0 in the deep feature extraction module of the proposed framework was replaced with other CNN architectures, including EfficientNetV2B0 [73] (baseline 1), Xception [74] (baseline 2), DenseNet121 [75] (baseline 3), ResNet50 [76] (baseline 4), and MobileNet [77] (baseline 5). To ensure fair comparisons, the five baselines were trained and evaluated using the same sets, hyperparameters, and metrics as those used for the proposed framework.

Before delving into the comparison of the results, it is essential to know that CNN models renowned for their performance on some datasets may perform poorly on others [78]. This discrepancy becomes more apparent when datasets differ in several aspects, such as size, clarity of relevant features among classes, or overall data quality. Powerful models can be prone to overfitting issues, while simpler models may struggle to capture relevant features in complex datasets with intricate patterns and relationships. Therefore, it's crucial to carefully select or develop an appropriate CNN architecture for a specific issue. For instance, EfficientNetV2B0 demonstrates superior performance compared to EfficientNetV1B0 across various classification tasks [73], including the ImageNet dataset. Moreover, it surpasses the

Table 4 Performance of the proposed framework on both test sets

Threshold	Test set 1%				Test set 2%			
	Macro accuracy	TNPR	TPR	(TPR-TNPR)	Macro accuracy	TNPR	TPR	(TPR-TNPR)
Default: 0.5	83	92	74	18	82	88	76	12
Optimal: 0.038	**85**	84	86	**2**	82	81	83	2

TNPR and TPR are true non-pushing rate and true pushing rate, respectively
Bold font denotes the best performance

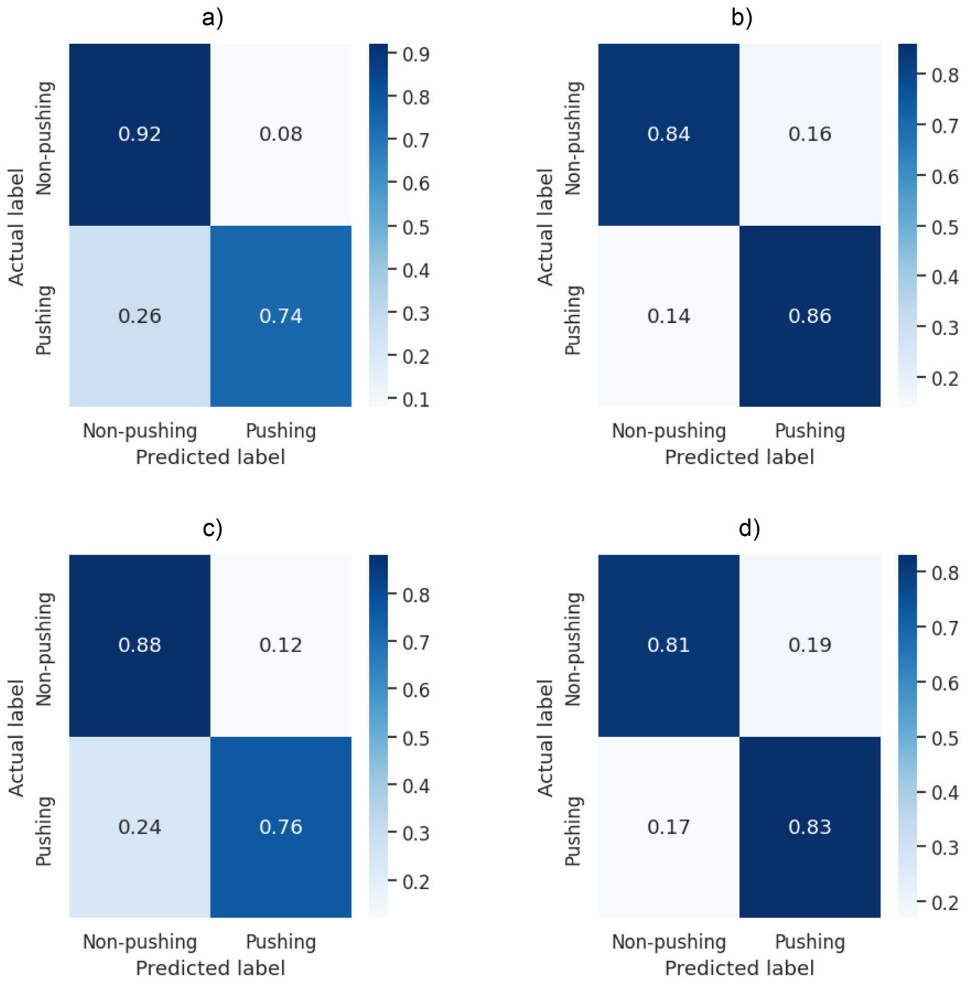

Fig. 11 Confusion matrix of the proposed framework on **a** Test set 1 with default threshold. **b** Test set 1 with the optimal threshold. **c** Test set 2 with default threshold. **d** Test set 2 with the optimal threshold

previous version in identifying regions that exhibit pushing persons in motion information maps of crowds [29, 31]. These remarkable outcomes can be attributed to the efficient blocks employed for feature extraction, namely the Mobile Inverted Residual Bottleneck Convolution [63] and Fused Mobile Inverted Residual Bottleneck Convolution [79]. Nevertheless, it should be noted that the presence of these efficient blocks does not guarantee the best performance in identifying pushing individuals based on local regions within the framework. Hence, in this section, the impact of six of the most popular and efficient CNN architectures on the performance of the proposed framework was empirically studied. For clarity, EfficientNetV1B0 was used within the framework, while the remaining CNN architectures were employed in the baselines.

The performance results of the proposed framework, as well as the baselines, are presented in Table 5 and visualized in Fig. 12. The findings indicate that EfficientNetV1B0 with optimal threshold leads the framework to achieve superior macro accuracy and AUC with balanced TPR and TNPR compared to CNNs used in baselines 1–5. This can be attributed to the architecture of EfficientNetV1B0, which primarily relies on the Mobile Inverted Residual Bottleneck Convolution with relatively few parameters. As such, the architectural design proves to be particularly suited for the generated dataset focusing on local regions. The visualiza-

Table 5 Comparative analysis of the developed framework and the five CNN-based frameworks

Framework	Threshold	Test set 1(%)				Test set 2(%)			
		M. acc	TNPR	TPR	| TPR-TNPR |	M. acc	TNPR	TPR	| TPR-TNPR |
The framework	Default: 0.5	83	92	74	18	82	88	76	12
	Optimal: 0.038	**85**	84	86	**2**	**82**	81	83	**2**
Baseline 1	Default: 0.5	83	91	74	17	80	91	69	22
	Optimal: 0.167	84	88	80	8	81	89	74	15
Baseline 2	Default: 0.5	79	91	67	24	77	90	64	26
	Optimal: 0.062	81	80	81	1	78	76	79	3
Baseline 3	Default: 0.5	77	92	62	30	74	89	58	31
	Optimal: 0.038	81	77	84	7	76	70	83	13
Baseline 4	Default: 0.5	70	92	49	43	75	87	64	23
	Optimal: 0.024	75	81	69	12	79	88	77	11
Baseline 5	Default: 0.5	79	94	65	29	77	92	62	30
	Optimal: 0.076	83	83	84	1	80	80	79	1

M. acc means macro accuracy. TNPR and TPR are true non-pushing rate and true pushing rate, respectively
Bold font denotes the best performance

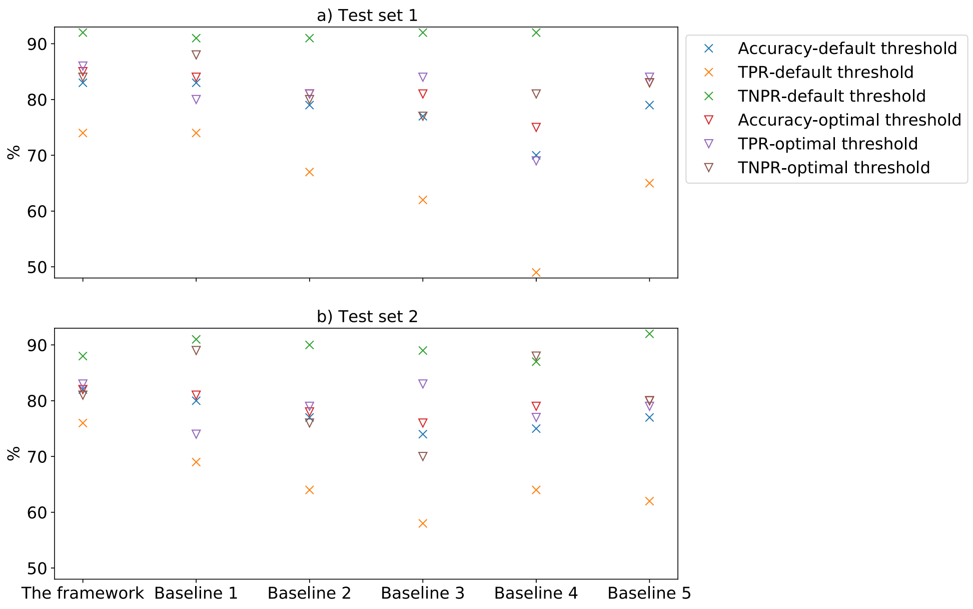

Fig. 12 Comparison between the framework (based on EfficientNetV1B0) with the baseline frameworks based on other popular CNN architectures

tion in Fig. 13 shows the optimal threshold values for the baselines. These thresholds, as shown in Table 5 and Fig. 12, mostly improved the macro accuracy, TPR, and balanced TPR and TNPR in the baselines. For example, baseline 1 with optimal threshold achieved 84% macro accuracy, roughly similar to the proposed framework. However, it fell short of achieving a balanced TPR and TNPR along with improving TPR on both test sets as effectively as the framework. To provide further clarity, baseline 1 achieved 80% TPR with 8% as the difference between TPR and TNPR, whereas the proposed framework attained an 86% TPR with 2% as the difference between TPR and TNPR on test set 1. Similarly, on test set 2, the framework achieved 81% TPR, while baseline 1 achieved a TPR of 74%.

Compared to other baselines that utilize optimal thresholds on test set 1, the proposed framework outperformed them regarding macro accuracy, TPR, and TNPR. Similarly, on test set 2, the framework surpasses all baselines except for the ResNet50-based baseline (baseline 4). However, it is essential to note that this baseline only achieved better TNPR, whereas the introduced framework excels in macro accuracy and TPR. As a result, the framework emerges as the superior choice on test set 2. To alleviate any confusion in the comparison, Fig. 14 shows the ROC curves with AUC values compared to its baselines on test set 1. Likewise, Fig. 15 depicts the same for test set 2. The AUC values show that the proposed framework achieved better performance than the baselines on both test sets. Moreover, they substantiate that

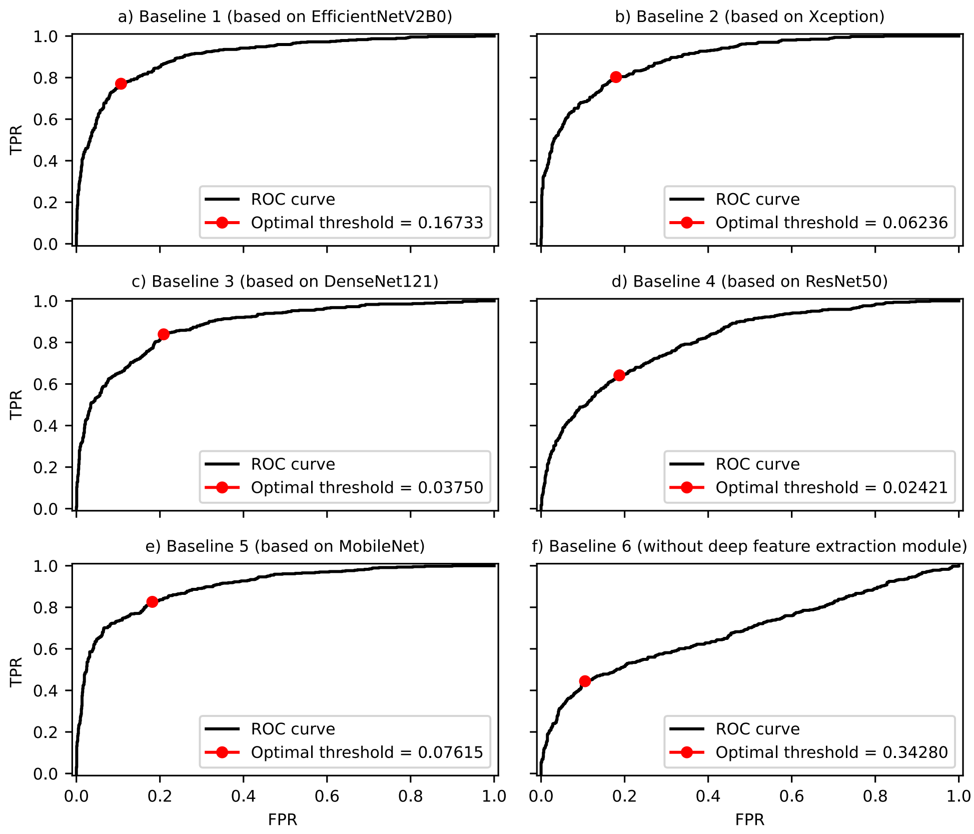

Fig. 13 ROC curves with optimal thresholds for the baselines over the validation set. TPR stands for true pushing rate, while FPR refers to false pushing rate. *ROC* receiver operating characteristics

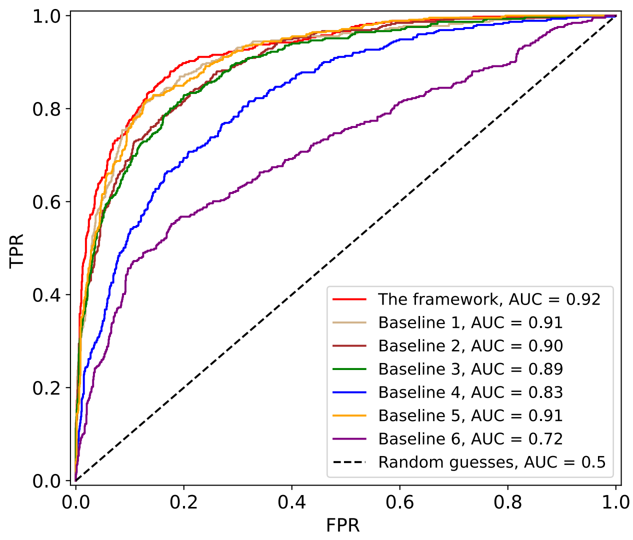

Fig. 14 ROC curves with AUC values on the test set 1. Comparison between the introduced framework (based on EfficientNetV1B0) with five baselines based on different CNN architectures, as well as the one baseline without the deep feature extraction module (baseline 6). *TPR* true pushing rate, *FPR* false pushing rate

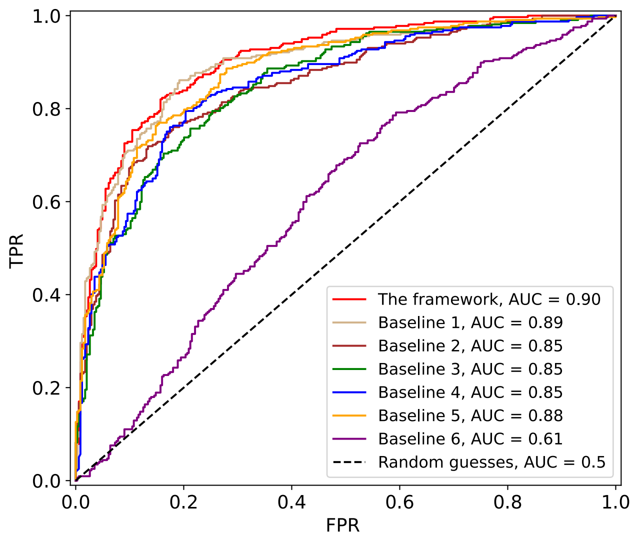

Fig. 15 ROC curves with AUC values on the test set 2. Comparison between the framework (based on EfficientNetV1B0) with five baselines based on different CNN architectures, as well as the one baseline without the deep feature extraction module (baseline 6). *TPR* true pushing rate, *FPR* false pushing rate

EfficientNetV1B0 is the most suitable CNN for extracting deep features from the generated local region samples.

In conclusion, the experiments demonstrate that the proposed framework, utilizing EfficientNetV1B0, achieved the highest performance compared to the baselines relying on other CNN architectures on both test sets. Furthermore, the optimal thresholds in the developed framework and the baselines resulted in a significant improvement in the performance across both test sets.

Table 6 Comparison to CNN-1 and CNN-2

Framework	Optimal threshold	Test set 1(%)			Test set 2(%)		
		Macro accuracy	TNPR	TPR	Macro accuracy	TNPR	TPR
The framework	0.038	85	84	86	82	81	83
CNN-1	0.23	73	71	75	64	40	88
CNN-2	0.0076	71	71	71	75	66	85

TNPR and TPR are true non-pushing rate (Sensitivity) and true pushing rate (Specificity), respectively

Comparison with customized CNN architectures in abnormal behavior detection field

Here, there are two objectives: (1) evaluating the performance of some existing CNN models developed to detect abnormal human behavior for pushing detection purposes. (2) Further evaluation of the trained binary classifier (EfficientNetB0 with fully connected layer and Sigmoid activation function). The customized architectures are CNN-1 [23] and CNN-2 [41]. The first architecture, CNN-1, employed 75 × 75 pixels as an input image. Furthermore, three convolutional layers, batch normalization, and max pooling operations were used for feature extraction. The developers of this model utilized a fully connected layer with a softmax activation function for classification. The second architecture, CNN-2, downsized the input images to 32 × 32 pixels before employing three convolutional layers with three max-pooling layers. For classification, it used two fully connected layers, with the first layer based on a ReLU activation function and the second layer employing a softmax activation function.

The results in Table 6 and Fig. 16 demonstrate that our classifier surpassed CNN-1 and CNN-2 by at least 11% in all metrics, including macro accuracy, TPR, TNPR, and AUC on test set 1. In contrast, on Test Set 2, the developed classifier achieved a macro accuracy of 82%, significantly outperforming CNN-1 and CNN-2, which attained 40% and 66%, respectively. Furthermore, the two customized CNN architectures exhibited high False Pushing Rates (FPR) of over 34%, while the proposed classifier's FPR was 19%. Additionally, as shown in Fig. 17, the AUC of our classifier stands at 90%, in comparison to CNN-1 and CNN-2, which achieved 74% and 83%, respectively.

To sum up, the proposed binary classifier has demonstrated significant superiority over CNN-1 and CNN-2 across both test sets. Given the high complexity of pushing detection in crowded environments, the simple architectures of CNN-1 and CNN-2 fall short of effectively identifying pushing MIM-patches.

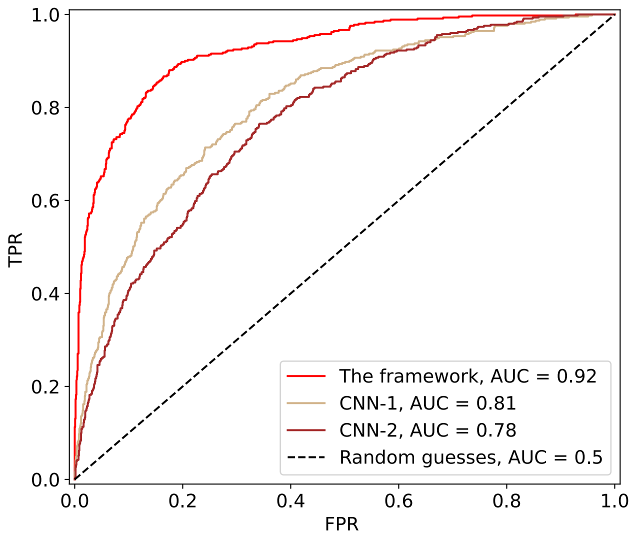

Fig. 16 ROC curves of our classifier and the two customized CNNs on test set 1

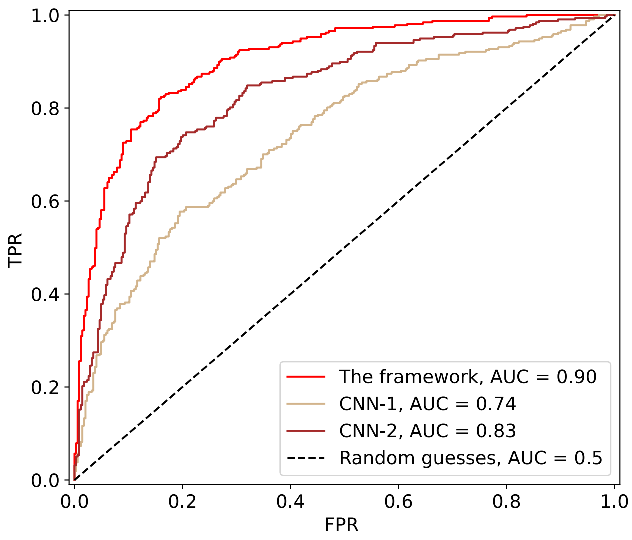

Fig. 17 ROC curves of our classifier and the two customized CNNs on test set 2

Impact of deep feature extraction module

This section aims to investigate how the deep feature extraction module affects the framework's performance. For this purpose, a new baseline (baseline 6) is developed, incorporating a Voronoi-based local region extraction module and detection component. In other words, the deep feature extrac-

Table 7 Performance results of the baseline 6

Threshold	Test set 1(%)			Test set 2(%)		
	Macro accuracy	TNPR	TPR	Macro accuracy	TNPR	TPR
Default: 0.5	59	97	18	58	59	57
Optimal: 0.342	67	91	44	59	38	79

TNPR and TPR are true non-pushing rate (Sensitivity) and true pushing rate (Specificity), respectively

tion module is removed from the proposed framework to construct this baseline.

Table 7 demonstrates that the baseline exhibited poor performance, with macro accuracy of 67% on test set 1 and 59% on test set 2. Additionally, Figs. 14 and 15 illustrate AUC values of 72% on test set 1 and 61% on test set 2 for baseline 6. Comparing this baseline with the weakest baseline in Table 5, which utilizes ResNet50, it is evident that deep feature extraction leads to macro accuracy improvement of at least 8% on test set 1 and at least 20% on test set 2. Similarly, deep feature extraction enhances AUC values by at least 11% on test set 1 and more than 24% on test set 2.

In summary, the deep feature extraction module significantly enhances the performance of the framework.

Impact of dummy points

This section aims to evaluate the impact of adding dummy points on the performance of the proposed framework. For this purpose, a new dataset, identical to the original (Table 2) but without dummy points, was prepared. The new dataset was then used to train and evaluate the framework (named baseline 7). Figure 18, on the right in a, b, c, and d, displays several examples of local regions generated without using dummy points.

In the proposed framework, the local region is crucial in assisting it to identify the behavior of each individual i. This is because the local region encompasses the crowd dynamics around i, thereby reflecting potential interactions between i and its neighbors. It is clear that $\mathcal{L}$ of person i located at the borders of crowds when dummy points are not added, fail to encompass the crowd dynamics surrounding i (e.g., samples on the right side in Fig. 18a–c). This leads to losing valuable information about the i's behavior, decreasing the framework's performance. Meanwhile, the examples on the left (Fig. 18a–c) illustrate how the dummy point technique helps form $\mathcal{L}_i$ that encompasses the space around i. Moreover, Fig. 18d illustrates that the dummy point technique is not applied to i who is surrounded by neighbors in all directions. As Table 8 illustrates, incorporating dummy points enhanced the performance of the proposed framework on both test sets, improving the macro accuracy, TPR, and TNPR by 5%. In contrast, the framework without the dummy points technique (baseline 7) achieved lower performance, with a macro accuracy of 62%, a TNPR of 57%, and a TPR

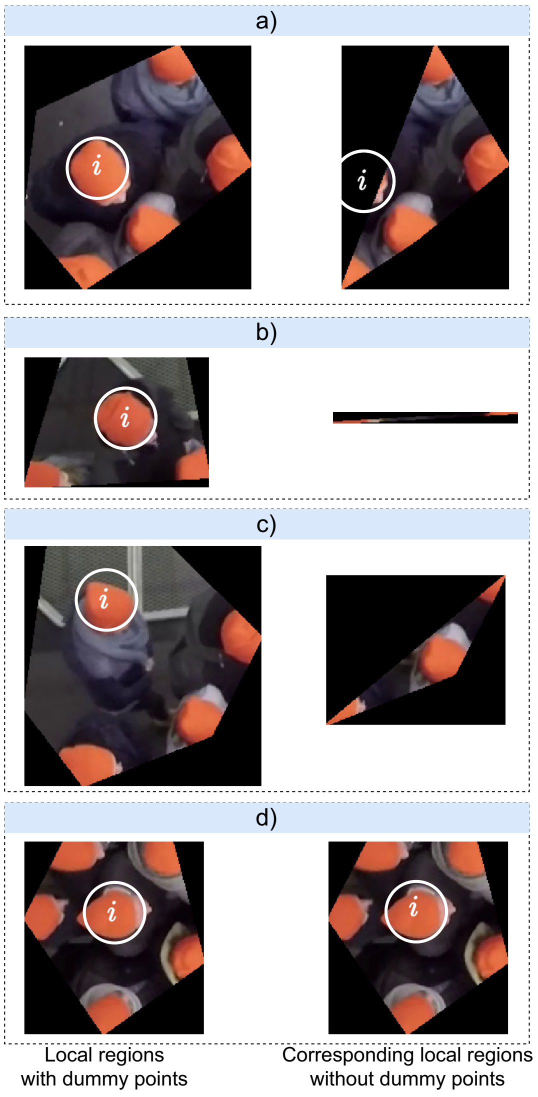
a)
b)
c)
d)
i
Local regions with dummy points
Corresponding local regions without dummy points

Fig. 18 Examples of local regions with dummy points (left samples) and without dummy points (right samples). The white circle represents the target person i in the local region $\mathcal{L}_i$

of 66%, compared to the developed framework. Moreover, the dummy point technique increased AUC in the framework by 3% and 26% over test set 1 and test set 2, respectively (Figs. 19, 20).

Table 8 Comparison to baseline 7

Framework	Optimal threshold	Test set 1(%)			Test set 2(%)		
		Macro accuracy	TNPR	TPR	Macro accuracy	TNPR	TPR
The framework	0.038	85	84	86	82	81	83
Baseline 7	0.030	80	79	81	62	57	66

TNPR and TPR are true non-pushing rate (Sensitivity) and true pushing rate (Specificity), respectively

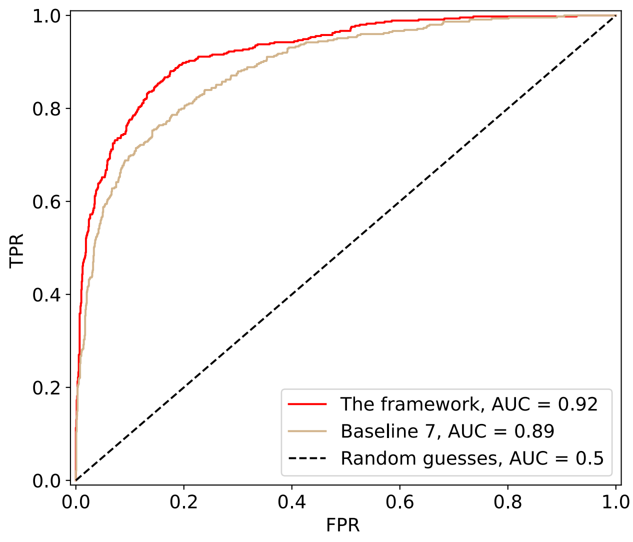

Fig. 19 ROC curves of the proposed framework and baseline 7 on test set 1

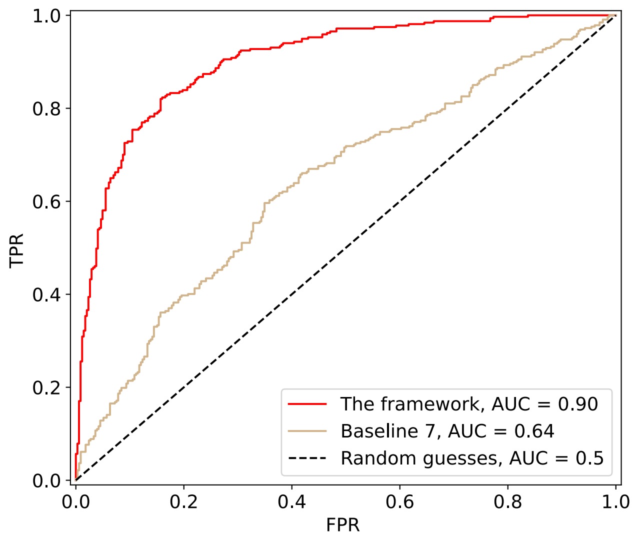

Fig. 20 ROC curves of the proposed framework and baseline 7 on test set 2

Performance evaluation against existing pushing detection approaches

The primary aim of this section is to assess the performance of the proposed framework by comparing it with existing automatic approaches for detecting pushing behavior in dense crowds. As mentioned in the literature, two main approaches have been published [29, 31], along with a more concise method referenced in [30], which is part of Ref. [29]. This comparison will focus on the primary approaches detailed in [29, 31]. It is noteworthy that these approaches employ a patch-based methodology, targeting the identification of patches containing at least one individual engaged in pushing. Each patch typically encompasses an area ranging from 1.2 to 2.55 m^2 on the ground. To clarify, both approaches employ a similar strategy for patch identification. They start by extracting Motion Information Maps (MIM) from consecutive frames of video or live streams that capture dense crowds. Every MIM, along with its initial corresponding frame, is then divided into a grid of MIM-patches, arranged into rows and columns as defined by the user. Figure 21a displays an example with 2×4 patches on the left side and a single MIM-patch on the right side. This patch identification strategy enables the trained classifiers in both approaches to mark pushing patches within the crowd. The red rectangle in Fig. 21a highlights an example of such annotated pushing patches.

It is evident that current approaches cannot detect individuals who join in pushing, a capability offered by the proposed framework. Consequently, comparing these approaches directly with the developed framework would be unfair, as they serve different purposes. To facilitate a fair and effective, the patch identification strategy in both existing approaches has been modified to operate at a microscopic level, as opposed to the patch level. The enhanced patch identification strategy employs the input trajectory data to find a square patch for each person (i), with i's position serving as the center of the patch. The dimensions of this patch are approximately 75 cm on the ground. This dimension was chosen after observing various dimensions in video experiments of entrances employed in this work. We found that this particular dimension ensures the patch not only covers individual i but also captures the surrounding crowd dynamics, thereby providing insight into the interactions between individuals i and their direct neighbors($\mathcal{N}_i$).

For instance, as illustrated in Fig. 21b, the patches for individuals numbered 53 and 66 are marked with white squares, along with the corresponding MIM-patches for the areas surrounding individuals 53 and 66. For the sake of clarity and

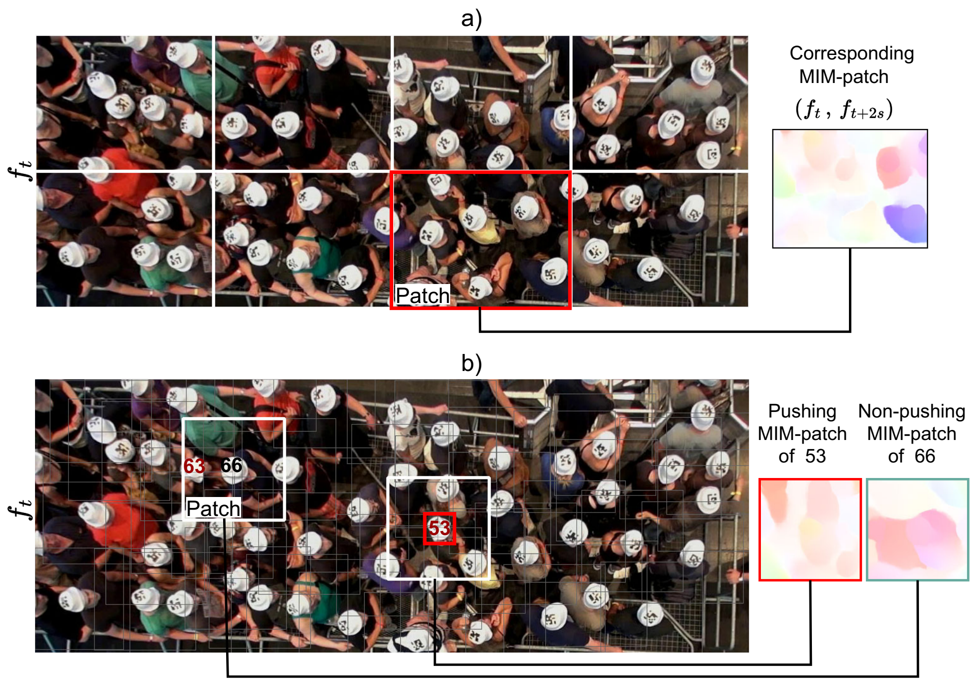

Fig. 21 Visual Examples of patches and MIM-patches in related Works [29, 31]: **a** a 2 × 4 patch array with a MIM-patch generated by related works. **b** Two examples of square patches (highlighted in white) and their corresponding MIM-patches created by DL4PuDe and Cloud-DL4PuDe. The numbers 53, 63, and 55 denote the IDs of three pedestrians. Moreover, in **b**, the dark gray color aims to give readers an overview of the square patches created by the enhanced patch identification strategy while ensuring that the original frame remains visible. In the notation f_t, f signifies a frame at timestamp t in seconds s. MIM stands for Motion Information Map

Table 9 Comparison to state-of-the-art automatic pushing detection approaches

Framework	Optimal threshold	Test set 1(%)			Test set 2(%)		
		Macro accuracy	TNPR	TPR	Macro accuracy	TNPR	TPR
The framework	0.038	85	84	86	82	81	83
DL4PuDe	0.023	77	76	78	62	44	80
Cloud-DL4PuDe	0.04	77	75	78	61	48	74

TNPR and TPR are true non-pushing rate (Sensitivity) and true pushing rate (Specificity), respectively

ease of discussion, the first [29] and second [31] approaches with enhanced patch identification strategy shall henceforth be referred to as "DL4PuDe" and "Cloud-DL4PuDe," respectively.

To compare Dl4PuDe and Cloud-DL4PuDe approaches with the proposed framework, it is necessary to train and evaluate them using a dataset that includes both pushing and non-pushing square MIM-patches. Furthermore, the setup of parameters, as outlined in Table 3, will be used in the training process. For this purpose, we created such a dataset using the same video experiments, trajectory data, and ground truth data employed in preparing the dataset for the proposed framework. Initially, MIMs are extracted from the video experiments using a combination of deep optical flow and color wheel methods, similar to the previous approaches. For more information, we refer the reader to Ref. [31], specifically Section IV.A.2. Next, the improved patch identification strategy is employed to extract patches from those MIMs. Afterward and based on the ground truth data, MIM-patches containing a person i engaged in pushing are labeled as pushing; otherwise, they are labeled non-pushing. For more clarity, in Fig. 21b, the MIM-patch for individual 53 is categorized as pushing due to the pushing behavior of person 53. Conversely, the MIM-patch for individual 66 is non-pushing, given that person 66 does not join in such behavior. Finally, the same splitting technique used to generate the dataset for training and evaluating the proposed framework was also employed to create the new dataset for DL4PuDe and Cloud-DL4PuDe approaches.

Table 9 displays the comparison results between the proposed framework and the enhanced DL4PuDe and Cloud-DL4PuDe approaches. The results indicate that the proposed framework significantly outperformed both approaches on both test sets. On test set 1, the proposed framework achieved a minimum 8% improvement in macro accuracy, TPR, and TNPR compared to the related approaches. In contrast, on test set 2, DL4PuDe and Cloud-DL4PuDe exhibited high false positive rates (FPR) with 62%and 61% macro accuracy, respectively. Meanwhile, the framework achieved an 82% macro accuracy on test set 2. Figure 22 presents the confusion matrices for each approach on both test sets. Moreover, as shown in Figs. 23 and 24, the framework

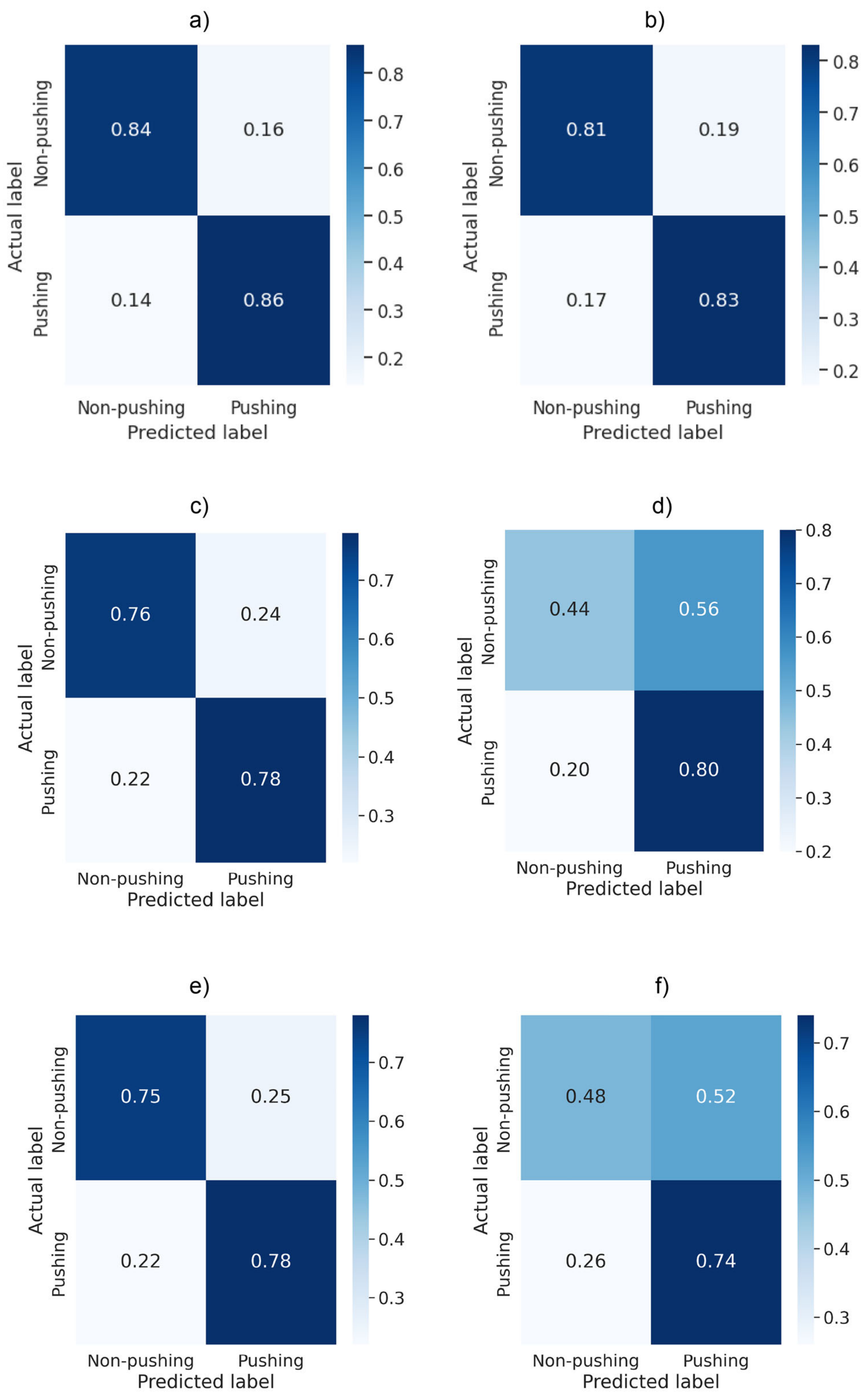

Fig. 22 Confusion matrices for the proposed framework, DL4PuDe, and Cloud-DL4PuDe with the optimal threshold: **a** proposed Framework on Test Set 1, **b** proposed framework on Test Set 2, **c** DL4PuDe on Test Set 1, **d** DL4PuDe on Test Set 2, **e** Cloud-DL4PuDe on Test Set 1, **f** Cloud-DL4PuDe on Test Set 2

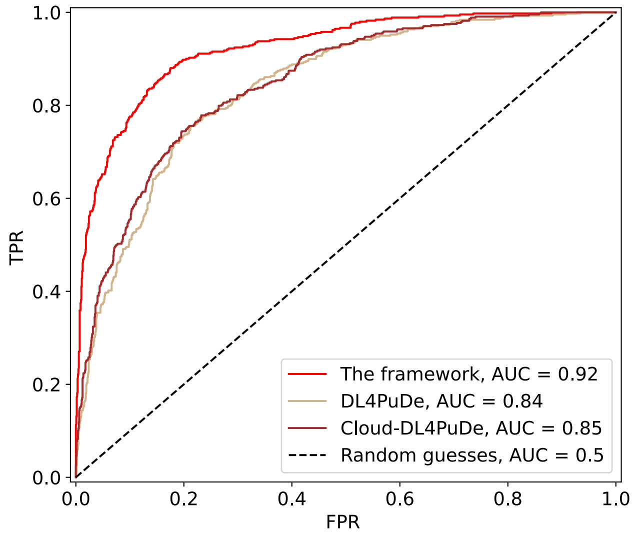

Fig. 23 ROC curves and AUC values for Test Set 1: a comparison between the introduced framework, DL4PuDe, and Cloud-DL4PuDe

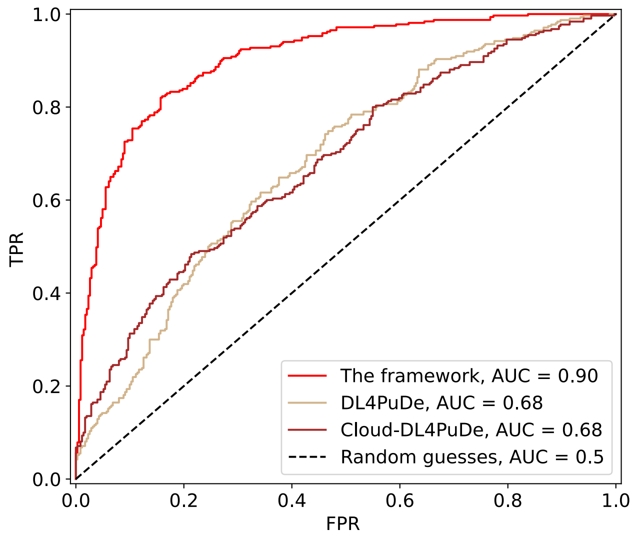

Fig. 24 ROC curves and AUC values for Test Set 2: a comparison between the introduced framework, DL4PuDe, and Cloud-DL4PuDe

consistently outperformed the other approaches in terms of the AUC metric. It achieved an improvement of at least 7% on test set 1 and 22% on test set 2. These results can be attributed to the fact that square patches may contain both pushing and non-pushing behavior simultaneously. This can lead the CNN classifier to learn irrelevant features from the patches. For more clarity, the patch of person 66 in Fig. 21b is classified as non-pushing because person 66 is not engaged in pushing behavior, even though person 63 within the same patch is involved in pushing. In summary, our developed framework achieved superior performance, demonstrating improvements of at least 8% in macro accuracy, TPR, TNPR, and AUC on both test sets compared to the enhanced DL4PuDe and cloud-DL4PuDe approaches. Furthermore, both approaches experience overfitting when attempting to detect pushing behavior at the microscopic level. This serves as evidence that our novel approach, used in our framework to identify the local regions of each person, efficiently assists EficientNetV1B0-based CNN in learning the relevant features for pushing behavior.

Conclusion and future work

This article introduced a new framework for automatically identifying pushing at the microscopic level within video recordings of crowds. The proposed framework utilizes a novel Voronoi-based method to determine the local region of each person in the input video over time. It further applies EfficientNetV1B0 to extract deep features from these local regions, capturing valuable information about individual behavior. Finally, a fully connected layer with a Sigmoid activation function is employed to analyze the deep features and annotate the pushing persons over time in the input video. To train and evaluate the performance of the framework, a novel dataset was created using six real-world experiments with their trajectory data and corresponding ground truths. The experimental findings demonstrated that the proposed framework surpassed state-of-the-art approaches, as well as seven baseline methods in terms of macro accuracy, true pushing rate, and true non-pushing rate.

The proposed framework has some limitations. First, it was designed to work exclusively with top-view camera video recordings that include trajectory data. Second, it was trained and evaluated based on a limited number of real-world experiments, which may impact its generalizability to a broader range of scenarios. Our future goals include improving the framework in two key areas: (1) enabling it to detect pushing persons from video recordings without the need for trajectory data as input. (2) Improving its performance in terms of macro accuracy, true pushing rate, and true non-pushing rate by: (a) Enlarging the dataset by utilizing additional videos from real-world experiments. These videos will encompass various scenarios. (b) Employing transfer learning and data augmentation techniques. (c) Processing a sequence of frames instead of a single frame, to extract more valuable features.

Acknowledgements The authors are thankful to Anna Sieben, Helena Lügering, and Ezel Üsten for the valuable discussions, manual annotation of the pushing behavior in the video of the experiments.

Author Contributions Conceptualization, A.A.; methodology, A.A., A.S.; software, A.A.; validation, A.A.; formal analysis, A.A.; investigation, A.A.; data curation, A.A.; writing—original draft preparation, A.A.; writing—review and editing, A.A., M.M., M.C. and A.S.; supervision, M.M., M.C. and A.S.; All authors have read and agreed to the published version of the manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was funded by the German Federal Ministry of Education and Research (BMBF: funding number 01DH16027) within the Palestinian-German Science Bridge project framework, and partially by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)-491111487.

Availability of data and code All videos and trajectory data used in generating the patch-based dataset were obtained from the data archive hosted by the Forschungszentrum Jülich under CC Attribution 4.0 International license [35, 67]. The implementation of the proposed framework, codes used for building training and evaluating the models, as well as test sets and trained models are publicly available at: https://github.com/PedestrianDynamics/VCNN4PuDe or at [80] (accessed on 23 July 2023). The training and validation sets are available from the corresponding authors upon request.

Declarations

Conflict of interest The authors declare that there is no conflict of interest regarding the publication of this article.

Ethical approval The experiments used in the dataset were conducted according to the guidelines of the Declaration of Helsinki and approved by the ethics board at the University of Wuppertal, Germany. Informed consent was obtained from all subjects involved in the experiments.

References

1. Feldmann S, Adrian J (2023) Forward propagation of a push through a row of people. Saf Sci 164:106173
2. Li X, Xuan X, Zhang J, Jiang K, Liu W, Yi R, Song W (2021) Experimental study on the movement characteristics of pedestrians under sudden contact forces. J Stat Mech Theory Exp 2021(6):063406
3. Adrian J, Seyfried A, Sieben A (2020) Crowds in front of bottlenecks at entrances from the perspective of physics and social psychology. J R Soc Interface 17(165):20190871
4. Wang C, Weng W (2018) Study on the collision dynamics and the transmission pattern between pedestrians along the queue. J Stat Mech Theory Exp 2018(7):073406
5. Keating John P (1982) The myth of panic. Fire J 76(3):57–61
6. Sime Jonathan D (1983) Affiliative behaviour during escape to building exits. J Environ Psychol 3(1):21–41
7. Li J, Wang J, Xu S, Feng J, Li J, Wang Z, Wang Y (2022) The effect of geometric layout of exit on escape mechanism of crowd. In: Building simulation. Springer, pp 1–10
8. Goyal T, Kahali D, Rastogi R (2020) Analysis of pedestrian movements on stairs at metro stations. Transp Res Procedia 48:3786–3801
9. CroMa Project (2018) Crowd management in transport infrastructures (project number 13n14530 to 13n14533). https://www.croma-projekt.de/de
10. Üsten E, Lügering H, Sieben A (2022) Pushing and non-pushing forward motion in crowds: a systematic psychological observation method for rating individual behavior in pedestrian dynamics. Collect Dyn 7:1–16
11. Adrian J, Boltes M, Holl S, Sieben A, Seyfried A (2018) Crowding and queuing in entrance scenarios: influence of corridor width in front of bottlenecks. arXiv preprint arXiv:1810.07424
12. Sieben A, Seyfried A (2023) Inside a life-threatening crowd: analysis of the love parade disaster from the perspective of eyewitnesses. arXiv preprint arXiv:2303.03977
13. Kooij Julian FP, Liem Martijn C, Krijnders Johannes D, Andringa Tjeerd C, Gavrila Dariu M (2016) Multi-modal human aggression detection. Comput Vis Image Underst 144:106–120
14. Wang C, Shen L, Weng W (2020) Experimental study on individual risk in crowds based on exerted force and human perceptions. Ergonomics 63(7):789–803
15. CrowdDNA Project (2020) Technologies for computer-assisted crowd management, fetopen-01-2018-2019-2020 fetopen challenging current thinking (project number 899739). https://crowddna.eu/
16. BaSiGo project (2012) Bausteine für die sicherheit von großveranstaltungen (project number 13n12045). https://www.vfsg.org/basigo-wiki/
17. Metivet T, Pastorello L, Peyla P (2018) How to push one's way through a dense crowd. Europhys Lett 121(5):54003
18. Andre B, Ricky AY, Said A, Aditya K et al (2023) Student attendance with face recognition (lbph or cnn): systematic literature review. Procedia Comput Sci 216:31–38
19. Lu W, Lan C, Niu C, Liu W, Lyu L, Shi Q, Wang S (2023) A cnn-transformer hybrid model based on cswin transformer for uav image object detection. IEEE J Sel Top Appl Earth Obs Remote Sens 16:1211–1231
20. Dong Y, Jiang Z, Tao F, Zhumu F (2023) Multiple spatial residual network for object detection. Complex Intell Syst 9(2):1347–1362
21. Ning C, Li Menglu S, Hao Y, Xueping YL (2021) Survey of pedestrian detection with occlusion. Complex Intell Syst 7:577–587
22. Liu Q, Wang X, Wang Y, Song X (2023) Evolutionary convolutional neural network for image classification based on multi-objective genetic programming with leader-follower mechanism. Complex Intell Syst 9(3):3211–3228
23. Direkoglu C (2020) Abnormal crowd behavior detection using motion information images and convolutional neural networks. IEEE Access 8:80408–80416
24. Alia AF, Taweel A (2017) Feature selection based on hybrid binary cuckoo search and rough set theory in classification for nominal datasets. Algorithms 14(21):65
25. Alia A, Taweel A (2021) Enhanced binary cuckoo search with frequent values and rough set theory for feature selection. IEEE Access 9:119430–119453
26. Alia A, Taweel A (2016) Hybrid nature inspired algorithms and rough set theory in feature selection for classification: a review. Int J Innov Res Comput Commun Eng 3:7
27. Gan H, Chengguo X, Hou W, Guo J, Liu K, Xue Y (2022) Spatiotemporal graph convolutional network for automated detection and analysis of social behaviours among pre-weaning piglets. Biosyst Eng 217:102–114
28. Gan H, Mingqiang O, Huang E, Chengguo X, Li S, Li J, Liu K, Xue Y (2021) Automated detection and analysis of social behaviors among preweaning piglets using key point-based spatial and temporal features. Comput Electron Agric 188:106357
29. Alia A, Maree M, Chraibi M (2022) A hybrid deep learning and visualization framework for pushing behavior detection in pedestrian dynamics. Sensors 22(11):4040

30. Alia A, Maree M, Chraibi M (2022) A fast hybrid deep neural network model for pushing behavior detection in human crowds. In: 2022 IEEE/ACS 19th international conference on computer systems and applications (AICCSA). IEEE, pp 1–2
31. Alia A, Maree M, Chraibi M, Toma A, Seyfried A (2023) A cloud-based deep learning framework for early detection of pushing at crowded event entrances. IEEE Access 11:45936–45949
32. Green Peter J, Robin S (1978) Computing dirichlet tessellations in the plane. The Comput J 21(2):168–173
33. Andrew Alex M (1979) Another efficient algorithm for convex hulls in two dimensions. Inf Process Lett 9(5):216–219
34. Tan M, Le Q (2019) Efficientnet: rethinking model scaling for convolutional neural networks. In: International conference on machine learning. PMLR, pp 6105–6114
35. Adrian J, Boltes M, Holl S, Sieben A, Seyfried A (2018) Crowds in front of bottlenecks from the perspective of physics and social psychology. https://doi.org/10.34735/ped.2018.1
36. Chadha A, Andreopoulos Y (2017) Voronoi-based compact image descriptors: efficient region-of-interest retrieval with vlad and deep-learning-based descriptors. IEEE Trans Multimed 19(7):1596–1608
37. Bayar G, Bilir T (2023) Estimation of multiple crack propagation pattern in concrete using voronoi tessellation method. Sādhanā 48(3):165
38. Warren P, Raju N, Prasad A, Hossain MS, Subramanian R, Kapat J, Manjooran N, Ghosh R (2024) Grain and grain boundary segmentation using machine learning with real and generated datasets. Comput Mater Sci 233:112739
39. Moukheiber D, Mahindre S, Moukheiber L, Moukheiber M, Wang S, Ma C, Shih G, Peng Y, Gao M(2022) Few-shot learning geometric ensemble for multi-label classification of chest X-rays. In: MICCAI workshop on data augmentation, labelling, and imperfections. Springer, pp 112–122
40. Wentao S, Lemoine Jeffrey M, Shawky Abd-El-Monsif A, Singha Manali P, Limeng YS, Ramanujam J, Michal B (2020) Bionoinet: ligand-binding site classification with off-the-shelf deep neural network. Bioinformatics 36(10):3077–3083
41. Tay NC, Connie T, Ong TS, Goh KOM, Teh PS (2019) A robust abnormal behavior detection method using convolutional neural network. In: Computational science and technology. Springer, pp 37–47
42. Elvan D, Osman AE (2019) Anomaly detection in videos using optical flow and convolutional autoencoder. IEEE Access 7:183914–183923
43. Javan RM, Levine Martin D (2013) An on-line, real-time learning method for detecting anomalies in videos using spatio-temporal compositions. Comput Vis Image Underst 117(10):1436–1452
44. Singh G, Khosla A, Kapoor R (2019) Crowd escape event detection via pooling features of optical flow for intelligent video surveillance systems. Int J Image Graph Signal Process 10(10):40
45. George M, Bijitha CV, Jose BR (2018) Crowd panic detection using autoencoder with non-uniform feature extraction. In: 2018 8th international symposium on embedded computing and system design (ISED). IEEE, pp 11–15
46. Leoni SG, Takako EP, Henrique MK, de Carvalho R, da Silva E, Ivanovitch S, Theo L (2019) Accelerometer-based human fall detection using convolutional neural networks. Sensors 19(7):1644
47. Mehmood A (2021) Lightanomalynet: a lightweight framework for efficient abnormal behavior detection. Sensors 21(24):8501
48. Zhang X, Zhang Q, Shuo H, Guo C, Hui Yu (2018) Energy level-based abnormal crowd behavior detection. Sensors 18(2):423
49. Ekanayake EMCL, Yunqi L, Cuihua L (2022) Crowd density level estimation and anomaly detection using multicolumn multistage bilinear convolution attention network (mcms-bcnn-attention). Appl Sci 13(1):248
50. Hwang I-C, Kang H-S (2023) Anomaly detection based on a 3d convolutional neural network combining convolutional block attention module using merged frames. Sensors 23(23):9616
51. Patwal A, Diwakar M, Tripathi V, Singh P (2023) An investigation of videos for abnormal behavior detection. Procedia Comput Sci 218:2264–2272
52. Ghayth A, Maheswari S, Mohammad H, Sathasivam SS, Prakash KB, Pankaj D, Vibhute Amol D, Sudhakar S (2024) Enhancing video anomaly detection using spatio-temporal autoencoders and convolutional lstm networks. SN Comput Sci 5(1):190
53. Sabokrou M, Fayyaz M, Fathy M, Moayed Z, Klette R (2018) Deep-anomaly: fully convolutional neural network for fast anomaly detection in crowded scenes. Comput Vis Image Underst 172:88–97
54. Ming X, Xiaosheng Yu, Chen D, Chengdong W, Jiang Y (2019) An efficient anomaly detection system for crowded scenes using variational autoencoders. Appl Sci 9(16):3337
55. Smeureanu S, Ionescu RT, Popescu M, Alexe B (2017) Deep appearance features for abnormal behavior detection in video. In: International conference on image analysis and processing. Springer, pp 779–789
56. Ilyas Z, Aziz Z, Qasim T, Bhatti N, Hayat MF (2021) A hybrid deep network based approach for crowd anomaly detection. Multimed Tools Appl 80:1–15
57. Khan Shehroz S, Madden Michael G (2014) One-class classification: taxonomy of study and review of techniques. Knowl Eng Rev 29(3):345–374
58. Farnebäck G (2003) Two-frame motion estimation based on polynomial expansion. In: Scandinavian conference on Image analysis. Springer, pp 363–370
59. Yan H (2020) Design and implementation of abnormal behavior detection based on deep intelligent analysis algorithms in massive video surveillance. J Grid Comput 18(2):227–237
60. Almazroey AA, Jarraya SK (2020) Abnormal events and behavior detection in crowd scenes based on deep learning and neighborhood component analysis feature selection. In: Arni SR, Rao CR (eds) Joint European-US Workshop on Applications of Invariance in Computer Vision. Springer, pp 258–267
61. Aldayri A, Albattah W (2023) A deep learning approach for anomaly detection in large-scale hajj crowds. Vis Comput 1–15
62. Baíllo A, Chacón JE (2021) Statistical outline of animal home ranges: an application of set estimation. In: Handbook of statistics, vol 44. Elsevier, pp 3–37
63. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L-C (2018) Mobilenetv2: inverted residuals and linear bottlenecks. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018. Salt Lake City, UT, USA, pp 4510–4520
64. Ramachandran P, Zoph B, Le Quoc V (2017) Searching for activation functions. arXiv preprint arXiv:1710.05941
65. Hu J, Shen L, Sun G (2018) Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp 7132–7141
66. Han J, Moraga C (1995) The influence of the sigmoid function parameters on the speed of backpropagation learning. International workshop on artificial neural networks, IWANN ’95 Malaga-Torremolinos. Springer, Spain, pp 195–201
67. Entrance 2, entry with guiding barriers (corridor setup) (2013). https://doi.org/10.34735/ped.2013.1
68. Carmen E, Landrum Gregory A, Nadine S, Nikolaus S, Sereina R (2021) Ghost: adjusting the decision threshold to handle imbalanced data in machine learning. J Chem Inf Model 61(6):2623–2640
69. Boltes M, Seyfried A, Steffen B, Schadschneider A (2010) Automatic extraction of pedestrian trajectories from video recordings. In: Pedestrian and evacuation dynamics 2008. Springer, pp 43–54

70. Moore BE, Corso JJ (2020) Fiftyone. GitHub. Note: https://github.com/voxel51/fiftyone
71. Zheng S, Song Y, Leung T, Goodfellow I (2016) Improving the robustness of deep neural networks via stability training. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA, pp 4480–4488
72. Zachary DV, Eric L, Mohamad H, Dita M, Kim P, Alexandra S, Stephen K, Wai Eugene K, Philippe P (2021) Using a national surgical database to predict complications following posterior lumbar surgery and comparing the area under the curve and f1-score for the assessment of prognostic capability. Spine J 21(7):1135–1142
73. Tan M, Le Q (2021) Efficientnetv2: smaller models and faster training. In: International conference on machine learning. PMLR, pp 10096–10106
74. Chollet F (2017) Xception: deep learning with depthwise separable convolutions. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, HI, USA, pp 1251–1258
75. Liu GH, Zhuang M, Laurens VD, Weinberger KQ (2017) Densely connected convolutional networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, HI, USA, pp 4700–4708
76. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE conference on computer vision and pattern recognition. Las Vegas, NV, USA, pp 770–778
77. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861
78. Yamashita R, Nishio M, Do RKG, Togashi K (2018) Convolutional neural networks: an overview and application in radiology. Insights Imaging 9(4):611–629
79. Gupta S, Tan M (2019) Efficientnet-edgetpu: creating accelerator-optimized neural networks with automl. Google AI Blog 2:1
80. Alia A, Maree M, Chraibi M, Seyfried A (2023) VCNN4PuDe: a novel voronoi-based CNN framework for pushing person detection in crowd videos. https://doi.org/10.5281/zenodo.8175476