Spatio-Temporal Convolutional Autoencoders for Perimeter Intrusion Detection

Devashish Lohani[1,2](✉), Carlos Crispim-Junior[1], Quentin Barthélemy[2], Sarah Bertrand[2], Lionel Robinault[1,2], and Laure Tougne[1]

[1] Univ Lyon, Lyon 2, LIRIS, 69676 Lyon, France
{devashish.lohani,carlos.crispim-junior,laure.tougne}@liris.cnrs.fr
[2] FOXSTREAM, Vaulx-En-Velin, France
{d.lohani,q.barthelemy,s.bertrand,l.robinault}@foxstream.fr

Abstract. In the video surveillance context, a perimeter intrusion detection system (PIDS) aims to detect the presence of an intrusion in a secured perimeter. Existing camera based approaches relies on hand crafted rules, image based classification and supervised learning. In a real world intrusion detection system, we need to learn spatio-temporal features unsupervisely (as annotated data are very difficult to obtain) and use these features to detect intrusions. To tackle this problem, we propose to use a 3D convolutional autoencoder. It is inspired from the DeepFall paper where they use it for an unsupervised fall detection task. In this paper, we reproduce their results on the fall detection task and further extend this model to detect intrusions in a perimeter intrusion dataset. We also provide an extended evaluation scheme which helps to draw essential insights from the results. Our results (The source code is available at https://gitlab.liris.cnrs.fr/dlohani/stcae_pids.) show that we correctly reproduce the results of fall detection task and furthermore our model shows competitive performance in perimeter intrusion detection task. To our knowledge, it is the first time when a PIDS is made in a fully unsupervised manner while jointly learning the spatio-temporal features from a video-stream.

Keywords: Perimeter intrusion detection · Spatio-temporal data · 3D convolutions · Convolutional autoencoder · Unsupervised learning

1 Introduction

High security installations may contain a large boundary with the need to be protected from unwanted elements entering in the boundary. A perimeter intrusion detection system (PIDS) is used to serve this purpose and it aims at detecting the presence of an intrusion in a secured perimeter. An intrusion can be defined as a moving object belonging to a category of items like human, car, truck, motorcycle, *etc.*, which is defined as unauthorized for a particular perimeter or area

B. Kerautret et al. (Eds.): RRPR 2021, LNCS 12636, pp. 47–65, 2021.
https://doi.org/10.1007/978-3-030-76423-4_4

at a given time. The same object might not be categorized as an intruder if it is outside the perimeter or if it is being allowed at a different time, *e.g.* moving cars or people that are outside the boundary are not intruders. Similarly, intrusion objects like people/cars might be allowed to move in an area around daytime for example but unauthorized for the rest of the day, hence the importance of temporality. Stationary objects, even if belonging to an unauthorized category should not be classified as an intrusion, *e.g.* cars parked inside the perimeter must not be detected as an intrusion while a moving car entering or leaving the perimeter must be classified as an intrusion. So, we can understand how difficult is to detect intrusion as it is a rare event which is both time and space dependent and further the definition of an intrusion varies according to the installation to protect and cannot be generalized.

There exists PIDS with various highly sensitive sensors like microwave sensors, electric field sensors, active infrared sensors, *etc.*, to detect changes at different wavelengths to detect intrusions [8]. However, these PIDS produce a large number of false alarms and cannot differentiate between intrusion and other objects and thus requires a lot of human resources [7].

In order to overcome the disadvantages of these sensor based PIDS, many camera based PIDS have been proposed [10,14,16,20]. A set of cameras are assigned with user-defined field-of-view of the area to be surveyed and activity is monitored by intrusion detection algorithms. These algorithms detect the movements of an intruder attempting to breach a security wall or region and alert security. The key problem with video analytics based solution is false alarm [14] which is due to inherent complications of understanding of the object detected in the video especially if the object is far from the camera. The object may appear very small in the image that makes recognition of the object more difficult. Existing PIDS algorithms detect intrusion in a supervised manner by annotating small set of intrusion classes [16,20], using hand crafted features [15,16], treating video stream as an image based data (loosing the spatio-temporal features) and thus employing image based object classification [10,20]. Thus, existing models do not learn the real nature of video, which is a spatio-temporal data. They rely on handcrafted features and treat intrusion detection as a supervised learning problem which is not generalizable in reality as intrusions occur rarely and therefore, we cannot have a large annotated database. Furthermore, we cannot train on few object classes as intrusion classes can practically be very high in number.

To learn the spatio-temporal data unsupervisely from a video stream and then detect intrusion, we propose to use a 3D convolutional autoencoder model. The model is inspired from the work of Nogas *et al.* [13] where they use it for an unsupervised fall detection problem [19]. In this paper, we reproduce their results and further extend their model to do perimeter intrusion detection in a challenging dataset. Our model detects intrusions such as moving car, people, motorcycle, truck, *etc.*, in a secure perimeter, after training in a fully unsupervised setting.

This paper is organized as follows. Section 2 presents some related works found in recent literature about camera based PIDS. Section 3 introduces the

3D convolutional autoencoder with different architectures that we tested. It also details the training and evaluation. Section 4 presents the datasets used for both tasks. Section 5 presents the results and discussion. It provides the reproduced results for fall detection task with a new evaluation scheme which provides some key insights. This section also shows competitive results on the intrusion detection task. Finally, Sect. 6 reports the general conclusions drawn, and suggests future research directions.

2 Related Work

Intelligent video surveillance is a well-established commercial technology that allows the users to monitor and secure areas with the security cameras. It uses computer vision algorithms to detect moving objects in an image and filter non-relevant movements. A Gaussian mixture model for RGB background modeling is proposed in [15], allowing to detect moving objects using background subtraction. A surveillance system is introduced in [16], using closed-circuit television (CCTV) to detect and classify vehicles. They applied real-time vehicle detection and classification algorithms. Object detection is performed with a background subtraction method where the background is modeled by using a Gaussian mixture model. In order to classify the detected vehicles, a method combining histogram of oriented gradients and artificial neural networks (ANN) was used. However, both these works extract features using hand-crafted methods and more importantly they tackle object detection/classification/tracking in open areas where there is no concept of a perimeter and thus no intrusion detection.

An on-line intrusion event detection system is proposed in [20], using a model for training an event detection system based on object tracking. They modeled the training as a multiple instance learning problem, which allowed to train the classifier from annotated events despite temporal ambiguities. But their model uses many handcrafted features and further they try to model intrusion detection with supervised learning, while in reality it is an unsupervised learning problem due to the lack of annotated data as intrusions occur rarely.

Recently, an intelligent intrusion detection system with detection, classification, tracking, and action recognition of an intruder is introduced [10]. They proposed an integrated acquisition device combining optical and thermal cameras, a virtual fence to set the boundary between surveillance and external areas in a graphical user interface, a background model designed to detect moving objects and a convolutional neural network (CNN) to classify moving objects as either intruders or wild animals. Their model also relies on the fact that we have annotated data.

All the above models learn spatial and temporal features of a video stream independent of each other. They treat video frames as still images and learn spatial features, then they treat the temporal succession of spatial features. Overall, none of the existing PIDS learns spatio-temporal features from a video jointly and furthermore, they try to solve perimeter intrusion detection with a supervised learning approach.

Our work draws inspiration from the DeepFall paper [13]. This work is focused on detecting human falls from a video stream in an unsupervised manner (without any annotated data). They formulate the fall detection problem as an anomaly detection problem. They present a novel use of deep spatio-temporal convolutional autoencoders to learn spatial and temporal features from normal activities during training, *i.e.*, they first learn "what is normal". Then during testing, they detect the events which have a high reconstruction error, that is to say the falls. They also present a new anomaly scoring method that combines the reconstruction scores of frames across video sequences to detect falls. Furthermore, they show superior results in comparison to traditional autoencoder and convolutional autoencoder methods to identify falls.

In this work, we reproduce the results of the DeepFall paper [13] and draw key insights from them. We further tackle the problem of intrusion detection as an anomaly detection problem. We train a spatio-temporal convolutional autoencoder to understand "what is not an intrusion" and detect intrusions in testing videos by marking frames with high reconstruction error.

3 3D Convolutional Autoencoders

While 2D-CNN learns appropriate representations for image classification, detection and segmentation tasks [11], they are incapable of capturing the temporal information encoded in consecutive frames for video analysis problems [23]. One widely used solution to this is to add convolutional long short-term memory (ConvLSTM) [17] layers on top of 2D-CNN layers [3]. However, these approaches make the implicit hypothesis that spatial and temporal dimensions are independent and can be processed sequentially, missing the existing correlations between these dimensions.

A 3D kernel can be used to extract both spatial and temporal features from a video by convolving it with the volume formed by stacking temporally contiguous frames of the video [1]. This 3D convolution operation captures spatio-temporal information encoded in the video as information from these contiguous frames is cohesively used to form feature maps [9]. 3D-CNN is better suited for spatio-temporal feature learning than 2D-CNN [18] and it has been also used in the form of an autoencoder [21,23]. Such a 3D autoencoder learns representations that are locally invariant to spatio-temporal deformations of the video encoded by the 3D convolutional feature maps. It is sometimes referred as deep spatio-temporal convolutional autoencoder (DSTCAE) [13].

The idea is to learn the regular/normal visual information from video sequences. The intuition is that the trained autoencoder is able to reconstruct the motion features presented in regular videos with low error but unable to accurately reconstruct motions in irregular videos. In other words, the autoencoder can model the complex distribution of the regular dynamics of appearance changes.

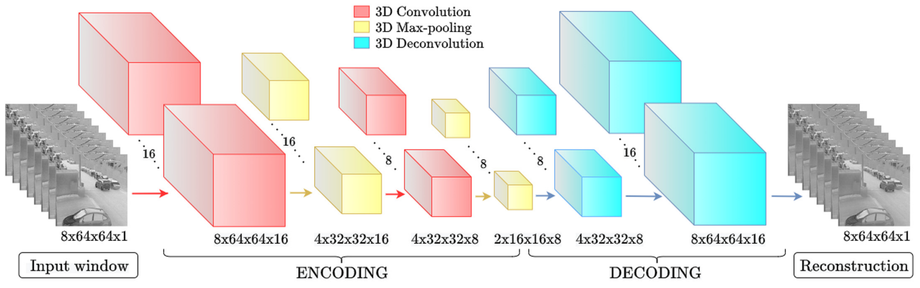

Fig. 1. Network architecture of DSTCAE-Deconvolution. The encoder is composed of layers with 3D convolution (red) followed by 3D max-pooling (yellow) and decoder is composed of 3D deconvolution (blue) layers. Each layer has dimensions: time window length × height × width × number of feature maps.

3.1 Input Window Construction

A 3D convolutional autoencoder takes a volume formed by stacking temporally contiguous frames of the video as input and reconstructs it. We refer to these volumes as windows and generate them by applying a temporal sliding window to video frames.

For a video with V frames, window length T, no padding and stride B (in temporal axis), the number of windows (D) generated [13] is given by:

$$D = \left\lfloor \frac{V - T}{B} \right\rfloor + 1. \tag{1}$$

These windows are fed into the network as follows. For an input video, we select first T frames and feed this window to the network. Then we shift by B frames temporally and select next T frames and so on until we cover all the V video frames.

3.2 Architecture Design

We evaluate three variants of the model. In Fig. 1, we illustrate the overall network outline with deconvolution model. Input video is fed as windows to the network where it is encoded by 3D convolution [9] and 3D max-pooling and decoded with a deconvolution operation [5] to obtain the reconstructed window. Encoding and decoding for the three models are illustrated in Fig. 2 and described in detail below.

Encoder: We set the window length $T = 8$, stride $B = 1$ in Eq. (1), resize input video frames to 64×64 and use grayscale image with 1 channel, thus the shape of input hyper-cuboid is 8×64×64×1. This input is encoded with a series of 3D convolution and 3D max-pooling layers. 3D convolutions operate with kernel of 5×3×3, 1×1×1 stride and same padding. The max-pooling layers have

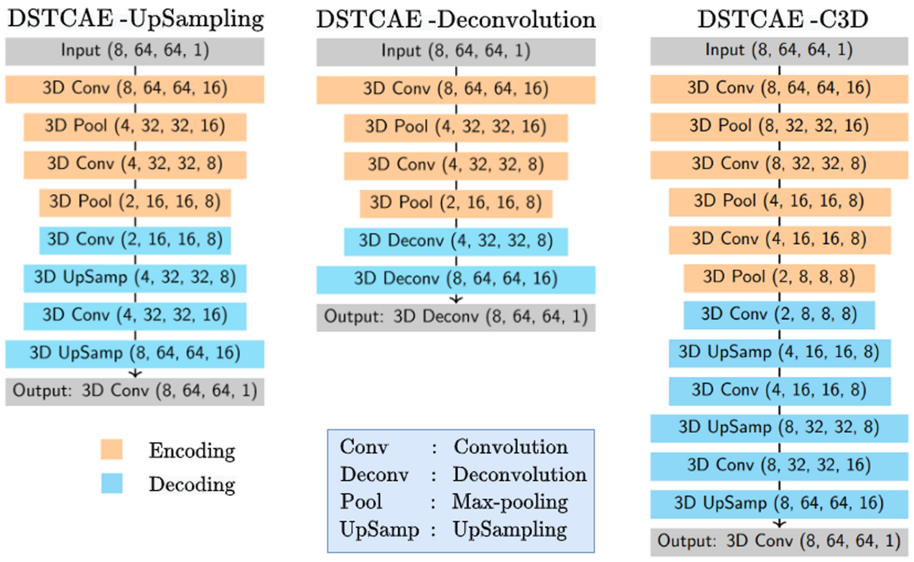

Fig. 2. Encoding and decoding configurations of DSTCAE-UpSampling, DSTCAE-Deconvolution and DSTCAE-C3D networks.

stride and kernel dimensions of 2×2×2 with same padding. This signify that each max-pooling layer reduce all input dimensions (time window length, height and width) by a factor of 2. Figure 2 shows specifications of encoding and decoding for the three models.

Decoder: We can decode either via upsampling or deconvolution. The upsampling method (DSTCAE-UpSampling) uses a 3D convolution with same parameters as in encoding, followed by a fixed upsampling operation to upscale the input. The upsampling operation uses upsampling factors of $2 \times 2 \times 2$, meaning matrix elements are repeated across each dimension such that the extent of all dimensions is doubled. The DSTCAE-Deconvolution architecture uses 3D deconvolutions [22] with kernel of $5 \times 3 \times 3$, $2 \times 2 \times 2$ stride and same padding [23]. Like upsampling, this results in doubling each dimension of the input. In both methods, the final reconstructed window has exactly the same dimensions as that of the input window.

Finally, the DSTCAE-C3D network is inspired by the work of Tran *et al.* [18], having the same encoding and decoding as DSTCAE-UpSampling, but with an extra 3D convolution + 3D max-pooling layer in encoding, and an extra 3D convolution + 3D upsampling layer in decoding (Fig. 2). These extra max-pooling and upsampling layers have 1×2×2 kernel dimensions, meaning they result in only spatial dimension change. Thus, it allows to train a deeper network without collapsing the temporal dimension.

A dropout layer with dropout probability of 0.25 is applied after second layer in all three models. We use the ReLU activation function for all hidden layers and tanh activation function at the output layer to constrain the reconstructed pixel values in the range [-1, 1], in order to be comparable to the input. The total training parameters for UpSampling, Deconvolution and C3D models are 15,889, 15,889 and 21,665 respectively.

3.3 Training

All three variants of 3D convolutional autoencoder are trained[1] only on videos with normal behaviour, *i.e.* without any falls or intrusion. All the frames in the videos are resized to 64×64, and pixels are rescaled by dividing values by 255 to keep them in the range [0, 1], and then subtracting the per-frame mean from each frame, resulting in pixel values to be in the range $[-1, 1]$.

The training loss of this network is the mean squared error, given by:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|\text{flat}(I_i) - \text{flat}(O_i(\theta))\|_2^2 \,, \tag{2}$$

where $I_i \in \mathbb{R}^{64\times64\times T}$ is the i^{th} window of the input batch of size N, $O_i \in \mathbb{R}^{64\times64\times T}$ is the corresponding reconstructed output window, θ denotes the network parameters, flat(.) is the flattening operator, which flattens the input array into a one dimensional vector and $\|\cdot\|_2$ denotes the Euclidean norm.

The training batch size is set to $N = 32$ for all experiments, where each element I_i of the batch consists of a stack of $T = 8$ frames. The training is performed with Adadelta optimizer for 500 epochs. These parameters were chosen to reproduce the exact results for the fall detection task, and we found no significant reduction in loss after further training.

3.4 Detection of Abnormal Events

Since we train our models only on videos without anomalous events by minimizing the reconstruction error (RE), during testing phase the anomalous (falls or intrusions) frames generally have a higher reconstruction error. We use this reconstruction error for anomaly detection. Given a test video sequence, we apply a sliding window as described in Sect. 3.1. For the i^{th} window I_i, the network outputs a reconstruction of this window O_i. The reconstruction error $r_{i,j}$ between the j^{th} frame of I_i and O_i is calculated as:

$$r_{i,j} = \|\text{flat}(I_{i,j}) - \text{flat}(O_{i,j})\|_2^2 \,. \tag{3}$$

Figure 3 (a) shows the sliding windows and associated reconstruction errors of frames. Since a single frame can be a part of upto $T = 8$ windows, therefore it can have different reconstruction error scores corresponding to each window.

[1] All experiments were done on NVIDIA GeForce GTX 1080, with 12 GB of RAM.

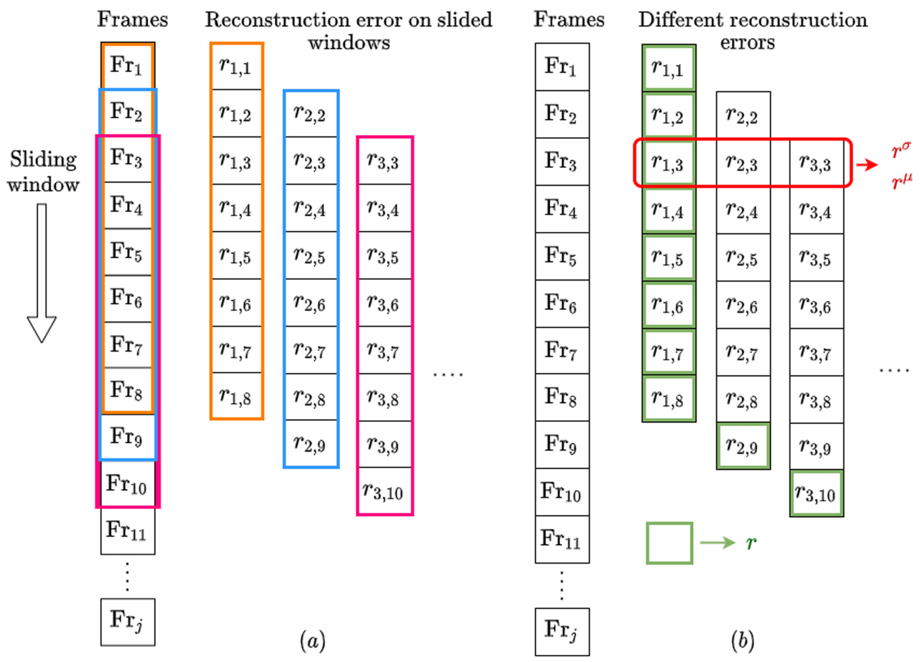
Frames
Reconstruction error on slided windows
Sliding window
Different reconstruction errors
r^{σ}
r^{μ}
r
(a)
(b)

Fig. 3. Illustration of errors: sliding windows on video with j frames (a), and different reconstruction error scores per frame (b).

For example, the frame Fr_3 has three reconstruction errors $r_{1,3}$, $r_{2,3}$ and $r_{3,3}$, in reference to first, second and third window respectively. Since we focus on frame level evaluation, we need one single reconstruction error value for each frame. We propose two ways to obtain the per-frame reconstruction error scores, which are described below.

Reconstruction Error *r*: A simple way to obtain a per frame reconstruction error is to get the reconstruction error of a frame from the first window it appears in. Since we use a temporal sliding window with window length T and stride $B = 1$, this means that a frame can appear on a maximum of T windows. The reconstruction error r for video frames is obtained as follows. r scores for the first T frames are obtained from the first window, then we slide our window temporally by 1 frame ($B = 1$) and r for the $(T + 1)^{\text{th}}$ frame is obtained from the second window and we similarly obtain r scores for next N frames from next N windows.

In Fig. 3 (b), the reconstruction error r is marked with green color. We can observe that for the first 8 frames, r is taken from first window, then from 9^{th} frame onwards, r is taken from last frame of each new window. For the j^{th} frame with $m = \max(1, j - T + 1)$, we obtain r_j as:

$$r_j = r_{m,j}. \tag{4}$$

Cross-Window Reconstruction Errors r^μ and r^σ: Another way to obtain a per frame reconstruction error can be to evaluate the statistics of a frame from the different temporal windows it appears in. Since each window that a frame appears in provides a different temporal context within which this frame can be viewed, we need to consider all the reconstruction errors obtained for a frame across different windows [12].

For the j^{th} frame of the i^{th} window, an anomaly score can be computed based on the mean r_j^μ or standard deviation r_j^σ of the reconstruction errors across temporal contexts with window length T. With $k = \min(j, T)$, we obtain r_j^μ and r_j^σ as follows[2]:

$$\begin{aligned} r_j^\mu &= \tfrac{1}{k}\textstyle\sum_{i=j-k+1}^{j} r_{i,j} \\ r_j^\sigma &= \sqrt{\tfrac{1}{k}\textstyle\sum_{i=j-k+1}^{j}\left(r_{i,j} - r_j^\mu\right)^2}. \end{aligned} \tag{5}$$

In Fig. 3 (b), the cross-window RE score calculation is depicted with red rectangle. Frame 3 appears in 1st, 2st and 3st window, therefore r_3^μ and r_3^σ are calculated using Eq. (5) with $r_{1,3}$, $r_{2,3}$ and $r_{3,3}$ respectively.

A high value of r_j^μ or r_j^σ means that the j^{th} frame, when appearing at different positions in different windows, is reconstructed with a high average error. For a normal activity or non-intrusion case, the reconstruction error of a frame should not vary a lot with its position in subsequent windows and if it does, then this may indicate anomalous behaviour, such as a fall or an intrusion. Similarly, a high value of r_j for a frame may indicate anomalous behaviour.

3.5 Evaluation Metrics

To check whether these frame level reconstruction error scores are sufficiently high to raise an alarm, we need to choose a threshold. But by choosing a fixed threshold, our evaluation will be biased to this particular dataset and threshold choice. Thus, to be independent from a fixed threshold, we vary the threshold from lowest to highest value of the reconstruction error score and obtain a receiver operating characteristic (ROC) curve [6] and the precision-recall (PR) curve [4]. The area under the curve (AUC) is computed for the ROC and PR curves, *i.e.* AUROC and AUPR respectively with fall or intrusion as the class of interest and this is used as a performance indicator. Higher the value of AUROC or AUPR, better is our model at classifying between anomalous frames and normal activity frames. However, AUPR must be used in case of highly imbalanced classes in the dataset [2,4], that is the case of anomaly detection tasks where normal activity frames (*i.e.* true negatives) are over-represented in the dataset.

For each test video, the RE scores (r, r^μ or r^σ) obtained for each frame are used to calculate AUC of the ROC and PR curve. Following [13], a first metric called "AUROC per video" is computed on each test video, and the average and standard deviation across all test videos are reported. In this metric, the succession of thresholds to separate classes and to estimate the ROC curve is

[2] In this paper, r_j^μ and r_j^σ correspond to C_μ^j and C_σ^j defined in [13].

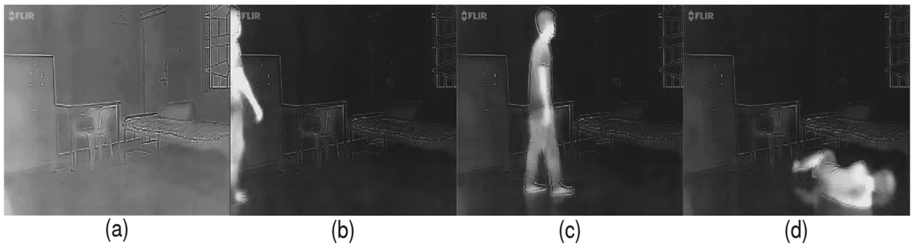

Fig. 4. Some frames drawn from Fall dataset [19]: non-fall frames, like an empty scene (a), a person entering (b), a person in the scene (c); and a fall frame (d).

not common to all test videos. Since the succession of thresholds is adapted to each test video, the ROC curve is in risk to be over-fitted, providing an overly optimistic AUROC score. Consequently, a second metric called "AUC all videos" is computed on ROC and PR curves, but on the whole test set with a threshold common to all test videos, which is the standard way to compute AUC [6]. Using this metric, we obtain AUROC and AUPR scores and they actually measure the generalization power of the detection models.

4 Datasets

Two datasets are used to train and test models: the first one for fall detection, and the second one, which is a private dataset, for perimeter intrusion detection.

4.1 Fall Dataset

This dataset is used for the fall detection task. In this task, we have a video camera which monitors the activity of a person in an area and the aim is to detect and alert as soon as a person falls. The problem here is quite similar to the intrusion detection as it is also an unsupervised task on a video stream [13]. We evaluate the model for detecting falls on the Thermal Fall Detection Activity Recognition dataset [19]. This dataset consists of videos captured by a FLIR ONE thermal camera mounted on an Android phone in a room setting with a single view with either 25 or 15 frames per second (FPS). The dataset contains a total of 44 videos. The training set has 9 videos without any fall event and the testing set contains 35 videos with fall events (828 fall frames out of 36,391 frames). The resolution of the thermal images is 640×480. Figure 4 shows some raw frames of the thermal dataset. We pre-process the dataset using Eq. (1) and obtain 22,053 windows to train the studied models.

4.2 Perimeter Intrusion Dataset

This private dataset consists of videos taken from a single thermal camera mounted at a fixed position with a single view in the outdoor uncontrolled setting. The videos are taken at 25 FPS with 400×296 frame size resolution and

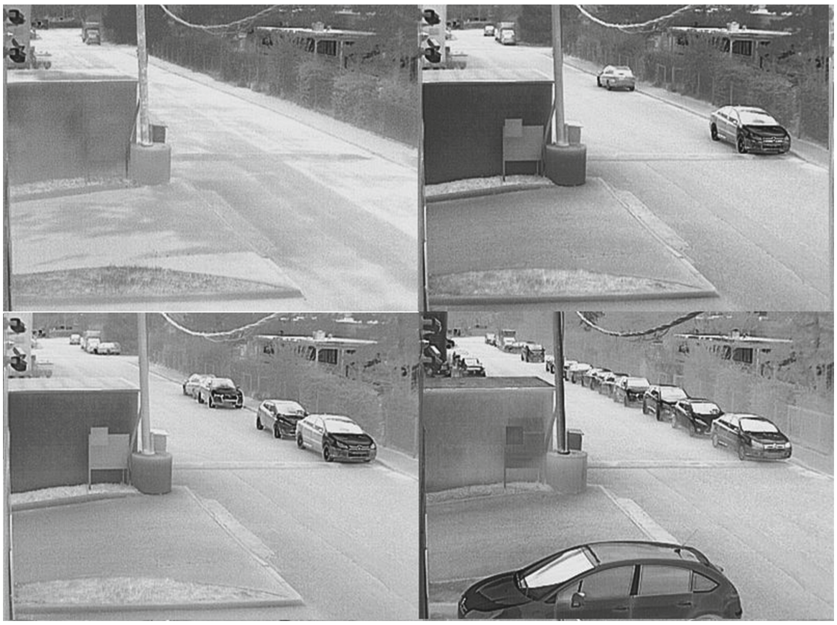

Fig. 5. Some frames drawn from Perimeter Intrusion dataset without intrusion.

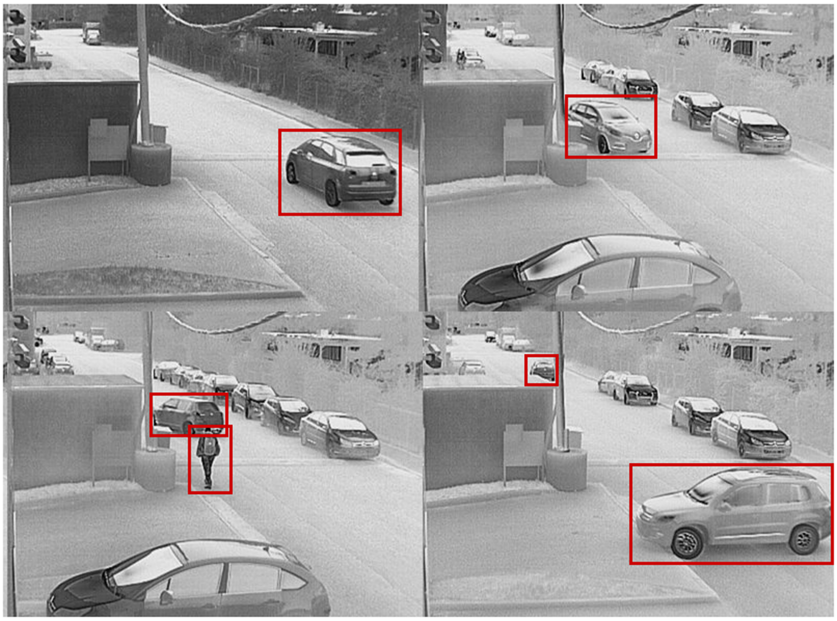

Fig. 6. Some frames drawn from Perimeter Intrusion dataset with intrusions, and intruders like persons or vehicles are labeled in red boxes. (Color figure online)

are then down-sampled at 5 FPS. These videos are intended to monitor the movement of any intruder designated object in the field of view of the camera. A total of 180 videos was collected with 80 videos for training containing only non-intrusion activities and 100 videos for testing. Out of these 100 test videos, 70 test videos contain intrusion and non-intrusion frames and 30 videos contain only non-intrusion frames. This 30% of only non-intrusion videos for testing is important in order to verify if the model is capable to distinguish between intrusion and non-intrusion activities. Each training video is converted into windows using Eq. (1) and we had a total of 47,998 windows for training.

The complexity of the dataset can be seen in Fig. 5 and 6 with some sample snapshots of the videos. Since video is taken outside, we have different daylight timing of the day/night and different weather conditions. Very often the strong wind wobbles the camera, or an electric wire in front of it and herbs nearby. The camera covers an intersection of the road with a long deep view of one road. Unlike the Fall dataset, here abnormality can be of any type like some person, a bike, car, truck, other vehicle or even a group of them. As discussed in the Introduction, an object belonging to an intruder class (like car, person, other vehicles) is considered an intrusion only if it shows movement in the monitored area, regardless of time of the day. They can come and go to any of the three entry/exit points of roads. Sometimes human intruder appears or disappears into the herbs seen on the right side of the video frames. Multiple intrusions are often present at some given instant. This makes intrusion detection very difficult. Furthermore, some cars are frequently parked and should not be detected as an intrusion. Since the camera captures a long view of one road, objects appear very small as they go far away, and their detection becomes even more complicated.

5 Results and Discussion

We evaluate the models on two different tasks, namely fall detection [19] and intrusion detection. With fall detection, we try to reproduce the results of the paper [13]. All three variants of 3D convolutional autoencoder were trained and tested on both tasks.

5.1 Reproducibility on the Fall Detection Task

The results of all three models are presented in Table 1. The training time is the total time taken in minutes to train a particular model with all the training set. Similarly, testing time is the total time taken to test all the test set with a particular model. In column "AUROC per video", we evaluate AUROC score for each test video separately and report average value with associated standard deviation (in brackets) for all the test videos in order to compare our reproduced results with the paper [13].

We can observe that we were able to reproduce the paper results correctly, some slight differences are possibly due to different model weight initialization. We can also observe that all three models perform equivalently well with r and

Table 1. Reproducibility of results of DeepFall [13] for different models with different reconstruction errors (RE) to evaluate: (i) computational times, (ii) AUROC per video, average +/- standard deviation across all videos of the test set, and (iii) AUC (ROC and PR) for all test videos.

Models	RE	Time		AUROC per video		AUC all videos	
		Training	Testing	[13]	Ours	ROC	PR
DSTCAE UpSampling	r^σ	309.52 min	49.88 s	0.96(0.03)	0.96(0.02)	0.96	**0.29**
	r^μ		48.61 s	0.95(0.04)	0.94(0.04)	0.88	0.23
	r		47.11 s	–	0.94(0.04)	0.89	0.24
DSTCAE Deconvolution	r^σ	311.01 min	56.31 s	0.96(0.02)	0.96(0.02)	0.96	**0.27**
	r^μ		55.94 s	0.94(0.04)	0.94(0.04)	0.88	0.23
	r		54.92 s	–	0.94(0.04)	0.89	0.21
DSTCAE C3D	r^σ	310.50 min	55.98 s	0.97(0.02)	0.96(0.03)	0.95	**0.25**
	r^μ		54.52 s	0.93(0.07)	0.90(0.07)	0.85	0.19
	r		54.23 s	–	0.91(0.06)	0.87	0.21

r^μ. Even though we do not observe a high difference in testing times for models with r, r^μ and r^σ but still models with r takes the least time. This is because calculating cross-window RE score induces latency in the system (need to wait for scores of next 7 frames to calculate score for current frame) while for r we can get the frame score at current window without any delay. In our experiments, C3D model did not outperform the other two models contrary to the claim in DeepFall [13]. We can observe that all models have similar performance for r^σ, however for r^μ and r, DSTCAE-UpSampling and DSTCAE-Deconvolution have performed slightly better than C3D model. The training time is observed as approximately the same for all models. UpSampling models are the fastest during testing and with the best performance.

To qualitatively understand the difference between ROC and PR curves computed on the whole test set, as explained in Sect. 3.5, Fig. 7 plots these two curves for the different models and RE scores. The ROC curve shows overall good performance for all the models, but we can remark that models with r^σ perform superior to others. However, all models have very poor performances in the PR curve, showing that the fall class is not well separated from the non-fall class.

In order to quantitatively assess these results, we can refer to the column "AUC all videos" of Table 1. The AUROC values show that the models with r^σ do not degrade their performance in comparison to AUROC per video, indicating that this RE score is able to capture inter-video variabilities well. The models with r^μ and r show an approximately 6% degradation in performance. But AUROC score is not preferred for a highly imbalanced dataset because ROC curves may provide an excessively optimistic view of the performance [2]. Instead, when dealing with highly skewed datasets, PR curves give a more informative picture of an algorithm's performance [4]. This is the case in fall detection because fall frames are rare in the videos, hence the fall test set has highly

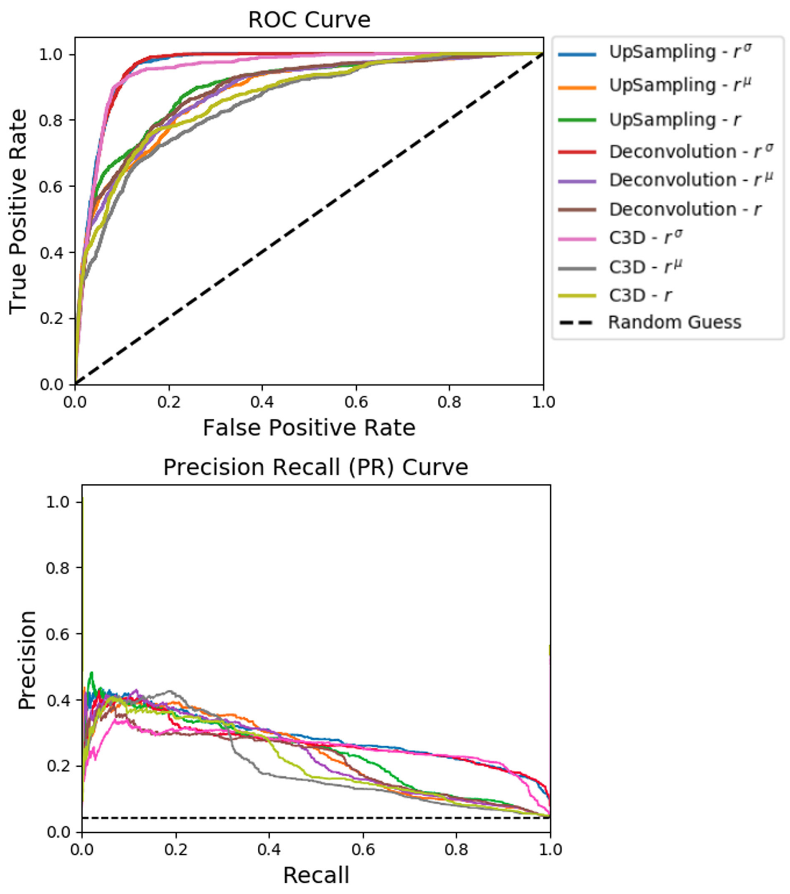

Fig. 7. ROC (top) and PR (bottom) curves (of type "all videos") for the fall detection task, for different models and RE scores.

imbalanced class proportion. In other words, AUPR is more sensitive to misclassification of fall classes. Contrary to the AUROC scores, we can observe that we have poor AUPR scores for all the models. This indicates that these models are not able to correctly detect falls in videos.

5.2 Application to the Perimeter Intrusion Detection Task

Figure 8 shows the evolution of reconstruction error r for a test video from Perimeter Intrusion dataset when tested with DSTCAE-UpSampling. The normal activity (no intrusion) has a low r score. When an intrusion enters the video, the r score starts increasing and reaches a peak when the intruder is closest to the camera. This r score decreases as the intrusion goes far away from the camera and gradually disappears. We can also observe that there are three peaks and they correspond to intrusion activities.

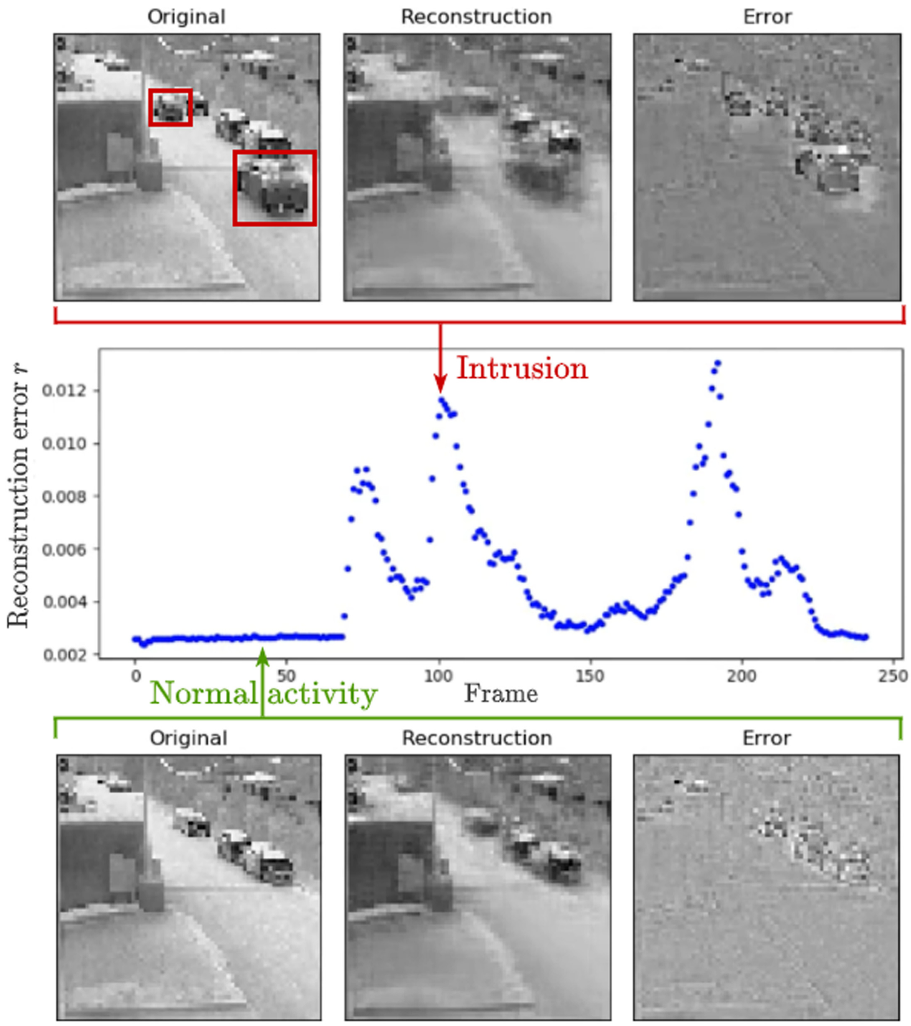

Fig. 8. Evolution of reconstruction error r for a test video from Perimeter Intrusion dataset. The original (64×64 resized), reconstructed and error frames are shown for an intrusion (top) and a normal activity (bottom). The three peaks correspond to intrusion activities with high r score.

The three images below the curve show the original frame at the point, its reconstructed frame, and the associated error map. We can observe that for normal activity, the error map correctly reveals no movement activity of the parked cars and thus no intrusion. In the images above the curve, we can see the image associated with a high reconstruction error score. We can observe that the reconstructed frame and error map shows the movement information of

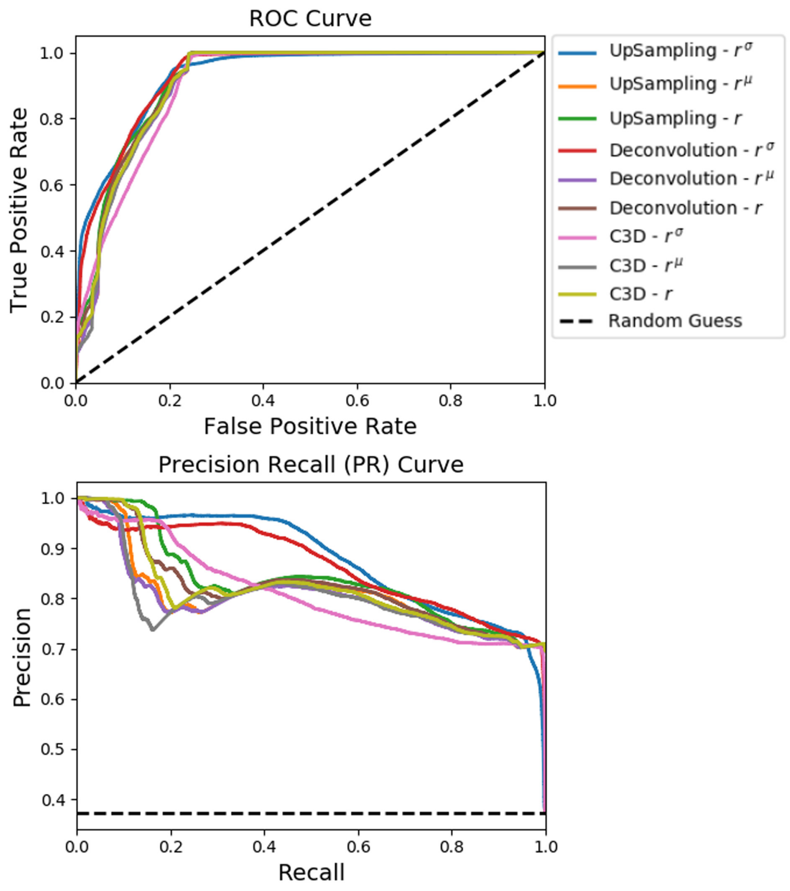

Fig. 9. ROC (top) and PR (bottom) curves for the perimeter intrusion detection task, for different models and RE scores.

two intruders (two cars), thus correctly detecting an intrusion frame with high reconstruction error score.

Figure 9 shows the ROC and PR curves for the perimeter intrusion detection task tested over all the videos of the test set, for different models and RE scores. We can observe that in ROC curve, it is difficult to assess which model has better performance. The PR curve however highlights the differences among models. We observe that UpSampling r^σ (in blue) and C3D r^σ (in pink) have similar ROC curves but their PR curves clearly show that UpSampling r^σ has better performance (larger area).

To quantitatively analyze these curves, AUC are listed in Table 2. We report the AUROC and AUPR scores over all the videos of the test set. We observe that the training and testing time is almost similar for all the models. Even though all models have approximately the same performance in terms of AUROC score, we

Table 2. Results on perimeter intrusion detection task for different models with different reconstruction errors (RE) to evaluate: (i) computational times, and (ii) AUC (ROC and PR) for all test videos of the Perimeter Intrusion dataset.

Models	RE	Time		AUC all videos	
		Training	Testing	ROC	PR
DSTCAE UpSampling	r^{σ}	590.25 min	55.19 s	0.93	**0.88**
	r^{μ}		52.05 s	0.91	0.81
	r		51.24 s	0.92	0.83
DSTCAE Deconvolution	r^{σ}	594.95 min	61.15 s	0.93	**0.86**
	r^{μ}		59.57 s	0.91	0.80
	r		58.55 s	0.91	0.82
DSTCAE C3D	r^{σ}	591.10 min	60.38 s	0.90	0.81
	r^{μ}		59.46 s	0.91	0.80
	r		57.98 s	0.91	0.82

observe that DSTCAE-UpSampling r^{σ} has highest performance of 0.88 in terms of AUPR score. Furthermore, it can be observed that upsampling models have lowest computational times. Unlike for Fall detection task, C3D models rank last among other evaluated models. Here, the gap between AUROC and AUPR scores is smaller than in the case of the fall detection (Table 1). This indicates that our models perform well regardless of the evaluation measure. The results without cross-window scores, *i.e.* with r models, are close to models with r^{σ} and r^{μ}. Furthermore, since the r score of the current frame is obtained only from the current window (inducing zero latency), it is compatible to be used in a real-time setting. Results indicate that the 3D convolutional autoencoder can successfully model intrusion events unsupervisely.

5.3 Discussion

Since both perimeter intrusion detection and fall detection have highly imbalanced classes, thus AUPR is more suitable metric than AUROC. We observe that evaluated architectures have a better performance in perimeter intrusion detection as compared to fall detection in terms of AUPR scores. Furthermore, the gap between AUROC and AUPR scores is lower in intrusion detection in comparison to fall detection. This can be attributed to the fact that in intrusion detection we are trying to detect movement of an intruder in a designated space: the results show that the 3D convolutional autoencoder is able to capture any movement well with the 3D spatio-temporal convolutions. However, in fall detection, we have a more difficult problem: the model needs to detect a particular type of movement, *i.e.* fall of a person, but not the other movements like walking, running, gesticulating, *etc.*. As results demonstrate, the 3D convolutional autoencoder classifies the two classes with lower performances in this case.

In convolutional autoencoders, there are two methods to apply a deconvolution operation [5]: (i) a convolution (filtering step) followed by an upsampling (interpolation step), and (ii) a deconvolution, also called a transposed convolution, which learns the weights in a single step. Concerning 3D networks, there is no evidence about the best method to deconvoluate 3D data. On both tasks, the UpSampling based method seems to be faster along with better detection scores than the Deconvolution one, although these improvements are quite marginal.

6 Conclusion

In this paper, we evaluated different forms of a 3D convolutional autoencoder for two unsupervised tasks. We also provided an extended evaluation using the metric "AUROC/AUPR for all videos" which evaluates capability of a model to capture inter-video variabilities. On the task of reproducibility of fall detection, we successfully reproduced the results of the Deepfall paper. We conclude that models with r^{σ} as reconstruction error have highest performance both in terms of AUROC per video and AUC for all videos. We observe a degradation in performance of models with r and r^{μ} when evaluated for AUROC all videos. This shows that r^{σ} captures inter-video variabilities better than other two metrics. The high gap between AUROC and AUPR values shows the limitation of current models for the fall detection task.

We further evaluated these models for perimeter intrusion detection in a challenging thermal video dataset. We can conclude that we have approximately similar performance for all the models. The models with upsampling were the fastest during testing and provided best results with r^{σ}. We observe that we have a smaller gap between AUROC and AUPR scores as compared to the fall detection results. This shows that these models capture inter-video variabilities better for the task of perimeter intrusion detection. Our results indicate that the 3D convolutional autoencoder models intrusion detection very well. To our knowledge, it is the first time that intrusion detection was carried out in a completely automatic and unsupervised manner.

For future works on the intrusion detection task, robustness of the model on different lighting conditions, sudden changes of luminosity and very slow intruder displacement needs to be further examined. We will also explore ways on how to choose a fixed threshold for the RE score, in order to allow a practical implementation of this PIDS.

References

1. Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., Baskurt, A.: Spatio-temporal convolutional sparse auto-encoder for sequence classification. In: BMVC, pp. 124.1–124.12 (2012)
2. Branco, P.O., Torgo, L., Ribeiro, R.P.: A survey of predictive modelling under imbalanced distributions. arXiv preprint arXiv:1505.01658 (2015)
3. Chong, Y.S., Tay, Y.H.: Abnormal event detection in videos using spatiotemporal autoencoder. In: ISNN, pp. 189–196 (2017)

4. Davis, J., Goadrich, M.: The relationship between Precision-Recall and ROC curves. In: ICML, pp. 233–240 (2006)
5. Dumoulin, V., Visin, F.: A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285 (2016)
6. Fawcett, T.: An introduction to ROC analysis. Pattern Recognit. Lett. **27**, 861–874 (2006)
7. Fennelly, L.J., Perry, M.: Physical security: 150 things you should know. Butterworth-Heinemann (2016)
8. Garcia, M.L.: Vulnerability assessment of physical protection systems. Elsevier(2005)
9. Ji, S., Xu, W., Yang, M., Yu, K.: 3D convolutional neural networks for human action recognition. IEEE Trans. Pattern Anal. Mach. Intell. **35**, 221–231 (2012)
10. Kim, S.H., Lim, S.C., et al.: Intelligent intrusion detection system featuring a virtual fence, active intruder detection, classification, tracking, and action recognition. Ann. Nucl. Energy **112**, 845–855 (2018)
11. Liu, L., et al.: Deep learning for generic object detection: a survey. Int. J. Comput. Vis. **128**, 261–318 (2020)
12. Nogas, J., Khan, S.S., Mihailidis, A.: Fall detection from thermal camera using convolutional LSTM autoencoder. In: ARIAL Workshop, IJCAI (2018)
13. Nogas, J., Khan, S.S., Mihailidis, A.: Deepfall: non-invasive fall detection with deep spatio-temporal convolutional autoencoders. J. Healthc Inform. Res. **4**, 50–70 (2020)
14. Norman, B.C.: Assessment of video analytics for exterior intrusion detection applications. In: ICCST, pp. 359–362 (2012)
15. Prakash, U., Thamaraiselvi, V.: Detecting and tracking of multiple moving objects for intelligent video surveillance systems. In: ICCTET, pp. 253–257 (2014)
16. Saran, K., Sreelekha, G.: Traffic video surveillance: vehicle detection and classification. In: ICCC, pp. 516–521 (2015)
17. Shi, X., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.K., Woo, W.C.: Convolutional LSTM network: a machine learning approach for precipitation nowcasting. In: NIPS, pp. 802–810 (2015)
18. Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M.: Learning spatiotemporal features with 3D convolutional networks. In: ICCV, pp. 4489–4497 (2015)
19. Vadivelu, S., Ganesan, S., Murthy, O.V.R., Dhall, A.: Thermal imaging based elderly fall detection. In: Chen, C.-S., Lu, J., Ma, K.-K. (eds.) ACCV 2016. LNCS, vol. 10118, pp. 541–553. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54526-4_40
20. Vijverberg, J.A., Janssen, R.T., de Zwart, R., de With, P.H.: Perimeter-intrusion event classification for on-line detection using multiple instance learning solving temporal ambiguities. In: ICIP, pp. 2408–2412 (2014)
21. Wang, X., Xie, W., Song, J.: Learning spatiotemporal features with 3DCNN and ConvGRU for video anomaly detection. In: ICSP, pp. 474–479 (2018)
22. Zeiler, M.D., Krishnan, D., Taylor, G.W., Fergus, R.: Deconvolutional networks. In: CVPR, pp. 2528–2535 (2010)
23. Zhao, Y., Deng, B., Shen, C., Liu, Y., Lu, H., Hua, X.S.: Spatio-temporal autoencoder for video anomaly detection. In: ACM MM, pp. 1933–1941 (2017)