Hindawi
Wireless Communications and Mobile Computing
Volume 2017, Article ID 9474806, 16 pages
https://doi.org/10.1155/2017/9474806

WILEY | Hindawi

Research Article

Vision-Based Fall Detection with Convolutional Neural Networks

Adrián Núñez-Marcos,[1] Gorka Azkune,[1] and Ignacio Arganda-Carreras[2,3,4]

[1]*DeustoTech, University of Deusto, Avenida de las Universidades, No. 24, 48007 Bilbao, Spain*
[2]*Department of Computer Science and Artificial Intelligence, Basque Country University, P. Manuel Lardizabal 1, 20018 San Sebastian, Spain*
[3]*Ikerbasque, Basque Foundation for Science, Maria Diaz de Haro 3, 48013 Bilbao, Spain*
[4]*Donostia International Physics Center (DIPC), P. Manuel Lardizabal 4, 20018 San Sebastian, Spain*

Correspondence should be addressed to Adrián Núñez-Marcos; adrian.nunez@deusto.es

Received 14 July 2017; Revised 26 September 2017; Accepted 9 November 2017; Published 6 December 2017

Academic Editor: Wiebren Zijlstra

One of the biggest challenges in modern societies is the improvement of healthy aging and the support to older persons in their daily activities. In particular, given its social and economic impact, the automatic detection of falls has attracted considerable attention in the computer vision and pattern recognition communities. Although the approaches based on wearable sensors have provided high detection rates, some of the potential users are reluctant to wear them and thus their use is not yet normalized. As a consequence, alternative approaches such as vision-based methods have emerged. We firmly believe that the irruption of the Smart Environments and the Internet of Things paradigms, together with the increasing number of cameras in our daily environment, forms an optimal context for vision-based systems. Consequently, here we propose a vision-based solution using Convolutional Neural Networks to decide if a sequence of frames contains a person falling. To model the video motion and make the system scenario independent, we use optical flow images as input to the networks followed by a novel three-step training phase. Furthermore, our method is evaluated in three public datasets achieving the state-of-the-art results in all three of them.

1. Introduction

Due to the physical weakness associated with aging, the elderly suffer high ratios of falls which frequently imply negative consequences for their health. According to Ambrose et al. [1], falls are one of the major causes of mortality in old adults. This can be explained in part by the high incidence of falls in adults over the age of 65: one in three adults falls at least once per year. In addition, the impact of these falls is a major concern for health care systems. It has to be noted that falls lead to moderate to severe injuries, fear of falling, loss of independence, and death of the third individual of the elderly who suffer these accidents. Moreover, the costs associated with these health problems are not negligible: two reference countries like the United States and the United Kingdom, with very different health care systems, spent US$23.3 and US$1.6 billion, respectively, in 2008 [2]. Taking into account the growth of aging population, these expenditures are expected to approach US$55 billion by 2020.

These considerations have boosted the research on automatic fall detection to enable fast and proper assistance to the elderly (see Section 2 for a review of the state of the art). The most common strategies consist in a combination of sensing and computing technologies to collect relevant data and develop algorithms that can detect falls based on the collected data [3]. These approaches have led to the appearance of Smart Environments for elderly assistance, which had been traditionally limited to home settings [4]. However, we believe that, with the irruption of the paradigm of the Internet of Things (IoT) [5], the possibilities to extend Smart Environments, and more specifically fall detection approaches, grow considerably.

In this paper, we focus on vision-based approaches for fall detection. Cameras provide very rich information about persons and environments and their presence is becoming more and more important in several everyday environments due to surveillance necessities. Airports, train and bus stations, malls, and even streets are already equipped with cameras. More importantly, cameras are also installed in elderly care centers. Therefore, reliable vision-based fall detection systems may play a very important role in future health care and assistance systems.

The recent impact of deep learning has changed the landscape of computer vision, improving the results obtained in many relevant tasks, such as object recognition, segmentation, and image captioning [6]. In this paper, we present a novel approach in this domain which takes advantage of Convolutional Neural Networks (CNN) for fall detection (Section 3). More precisely, we introduce a CNN that learns how to detect falls from optical flow images. Given the small size of typical fall datasets, we take advantage of the capacity of CNNs to be sequentially trained on different datasets. First of all, we train our model on the Imagenet dataset [7] to acquire the relevant features for image recognition. Afterwards, following the approach of [8], we train the CNN on the UCF101 action dataset [9]. For that purpose, we calculate the optical flow images of consecutive frames and use them to teach the network how to detect different actions. Finally, we apply transfer learning by reusing the network weights and fine-tuning the classification layers so the network focuses on the binary problem of fall detection.

As a result of the research carried out, this paper presents the following main contributions:

(i) To the best of our knowledge, this is the first time that transfer learning is applied from the action recognition domain to fall detection. In that sense, the use of transfer learning is crucial to address the small amount of samples in public fall detection datasets.

(ii) We use optical flow images as input to the network in order to have independence from environmental features. These images only represent the motion of consecutive video frames and ignore any appearance-related information such as color, brightness, or contrast. Thus, we are presenting a generic CNN approach to fall detection.

2. Related Work

The literature of fall detection is divided between sensor-based and vision-based approaches. The sensor-based detection has commonly relayed on the use of accelerometers, which provide proper acceleration measures such as vertical acceleration. In the case of falls, these measures are very different compared to daily activities or confounding events (such as bending over or squatting), allowing us to discern between them. Vallejo et al. [10] and Sengto and Leauhatong [11] proposed feeding a Multilayer Perceptron (MLP), the data of a 3-axis accelerometer (acceleration values in x-, y-, and z-axis). Kwolek and Kepski [12] applied an Inertial Measurement Unit (IMU) combined with the depth maps obtained from a Kinect camera. They also made use of a Support Vector Machine (SVM) classifier, feeding it the data from the IMU and the Kinect. Approaches like the latter and [13] combined sensors with vision techniques. However, they used vision-based solutions only to ascertain the prediction of the sensor-based approach.

The purely vision-based approaches focus on the frames of videos to detect falls. By means of computer vision techniques, meaningful features such as silhouettes or bounding boxes are extracted from the frames in order to facilitate detection. Some solutions use those features as input for a classifier (e.g., Gaussian Mixture Model (GMM), SVM, and MLP) to automatically detect if a fall has occurred. The use of tracking systems is also very extended; for example, Lee and Mihailidis [18] applied tracking techniques in a close environment to detect falls. They proposed using a connected-components labeling to compute the silhouette of a person and extracting features such as the spatial orientation of the center of the silhouette or its geometric orientation. Combining this information they are able to detect positions and also falls. Rougier et al. [19] suggested using silhouettes as well, which is a common strategy in the literature. Applying a matching system along the video to track the deformation of the silhouette, they analyzed the shape of the body and finally obtained a result with a GMM. Mubashir et al. [3] tracked the person's head to improve their base results using a multiframe Gaussian classifier, which was fed with the direction of the principal component and the variance ratio of the silhouette. Another common technique consists in computing the bounding boxes of the objects to determine if they contain a person and then detect the fall by means of features extracted from it (see, for instance, [20, 21]). Following a similar strategy, Vishwakarma et al. [22] worked with bounding boxes to compute the aspect ratio, horizontal and vertical gradients of an object, and fall angle and fed them into a GMM to obtain a final answer. Many solutions are based on supervised learning, that is, extracting lots of features from raw images and using a classifier to learn a decision from labeled data. This is the case, for example, of Charfi et al. [17], who extracted 14 features, applied some transformations to them (the first and second derivatives, the Fourier transform, and the Wavelet transform), and used a SVM to do the classification step. Zerrouki et al. (2016) [23] computed occupancy areas around the body's gravity center, extracted their angles, and fed them into various classifiers, being the SVM the one which obtained the best results. In 2017, the same author extended his previous work by adding Curvelet coefficients as extra features and applying a Hidden Markov Model (HMM) to model the different body poses [14]. A less frequent technique was used by Harrou et al. [24], who applied Multivariate Exponentially Weighted Moving Average (MEWMA) charts. However, they could not distinguish between falls and confounding events, which is a major issue that is taken into account in our solution. In fact, not being able to discriminate between such situations produces a great amount of false alarms.

Another branch inside the vision-based fall detection systems is the adoption of 3D vision to take advantage of 3D structures. This strategy requires the use of multiple

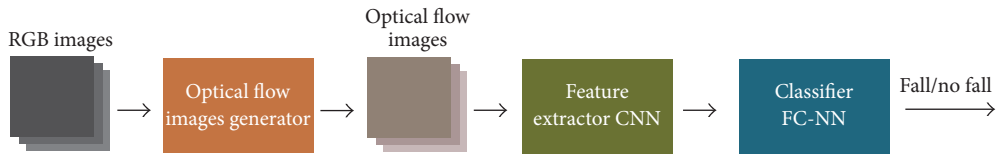


FIGURE 1: The system architecture or pipeline: the RGB images are converted to optical flow images, then features are extracted with a CNN, and a FC-NN decides whether there has been a fall or not.

cameras (passive systems) or active depth cameras such as Microsoft Kinect or time-of-flight cameras that can extract depth maps. The Kinect camera is very popular given its low price and high performance. Auvinet et al. [25] used a Kinect camera to build a 3D silhouette to then analyze the volume distribution along the vertical axis. Gasparrini et al. [26] also used such a camera to extract 3D features and then applied a tracking system to detect the falls. Kinect software provides body joints, which were used by Planinc and Kampel [27] to obtain the orientation of the major axis on their position. Diraco et al. [28] made use of depth maps to compute 3D features. Another simple and yet interesting approach was given by Mastorakis and Makris [29], who went beyond the typical 2D bounding boxes strategy and applied 3D bounding boxes. All the aforementioned methods took advantage of the 3D information provided by their camera systems. The drawbacks of such approaches are related to system deployment: they need either multiple synchronized cameras focused on the same area or active depth cameras which usually have narrow fields of view and a limited depth. Thus, from the point of view of system deployment, 2D passive systems are usually a better option, given their lower cost. It is also important to highlight that cameras are already installed in many public places, such as airports, shops, and elderly care centers. Those reasons make 2D passive camera-based fall detection a relevant application domain.

Nowadays, the use of deep neural networks is growing in many problem domains, including vision-based fall detection. Wang et al. [15] proposed using a PCAnet [30] to extract features from color images and then applied a SVM to detect falls. This approach is similar to ours but instead of a PCAnet we use a modified VGG16 architecture [31] that allows us to process various frames to take into account motion. Another research work, led by Wang et al. [16], combined Histograms of Oriented Gradients (HOG), Local Binary Pattern (LBP), and features extracted from a Caffe [32] neural network to recognize a silhouette and then applied a SVM classifier. In contrast, we avoid feature engineering completely, relying on the features learned by a CNN.

3. Materials and Methods

The design of our fall detection architecture was driven by the following objectives:

(i) To make the system independent from environmental features

(ii) To minimize the hand-engineered image processing steps

(iii) To make the system generic, so it works in different scenarios

To tackle the first objective, the key was to design a system that works on human motion, avoiding any dependence on image appearance. In that sense, a fall in a video can be expressed as a few contiguous frames stacked together. However, this is a naive approach, as the correlation between the frames is not taken into account by processing each image separately. To address this problem, the optical flow algorithm [33] was used to describe the displacement vectors between two frames. Optical flow allowed us to represent human motion effectively and avoid the influence of static image features.

In order to minimize hand-engineered image processing steps, we used CNNs, which have been shown to be very versatile automatic feature extractors [6]. CNNs can learn the set of features which better suit a given problem if enough examples are provided during their training phase. Furthermore, CNNs are also very convenient tools to achieve generic features. For that purpose, network parameters and training strategies need to be tuned.

Since time management is a crucial issue in fall detection, a way to cope with time and motion had to be added to CNNs. With that objective, we stacked a set of optical flow images and fed them into a CNN to extract an array of features $F \in \mathbb{R}^{w \times h \times s}$, where w and h are the width and height of the images and s is the size of the stack (number of stacked optical flow images). Optical flow images represent the motion of two consecutive frames, which is too short-timed to detect a fall. However, stacking a set of them the network can also learn longer time-related features. These features were used as input of a classifier, a fully connected neural network (FC-NN), which outputs a signal of "fall" or "no fall." The full pipeline can be seen in Figure 1.

Finally, we used a three-step training process for our optical flow stack-based CNN. This training methodology is adopted due to the low number of fall examples found in public datasets (Section 3.3). Furthermore, it also pursues the generality of the learned features for different falling scenarios. The three training steps and their rationale are explained in detail in Section 3.2.

3.1. The Optical Flow Images Generator. The optical flow [34] algorithm represents the patterns of the motion of objects as displacement vector fields between two consecutive images, which can be seen as a 1-channel image $I \in \mathbb{R}^{w \times h \times 1}$, where w and h are the width and height of the image that represents the correlation between the input pair. By stacking $2L$ optical flow

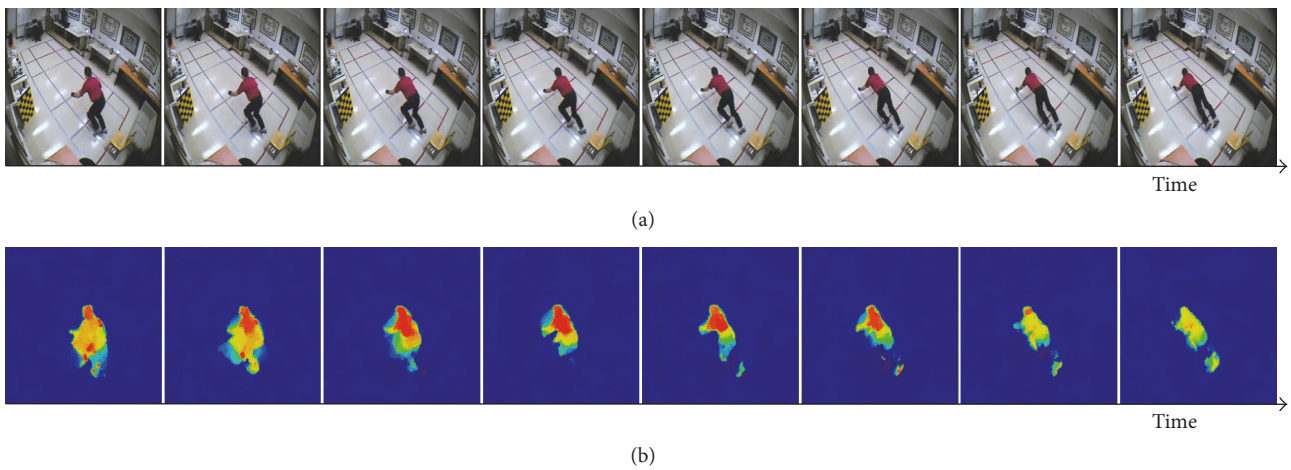


Figure 2: Sample of sequential frames of a fall from the Multiple Cameras Fall Dataset (a) and their corresponding optical flow horizontal displacement images (b).

images (i.e., L pairs of horizontal and vertical components of vector fields, d_t^x and d_t^y, respectively), we can represent a motion pattern across the stacked frames. This is useful to model short events like falls. The use of optical flow images is also motivated by the fact that anything static (background) is removed and only motion is taken into account. Therefore, the input is invariant to the environment where the fall would be occurring. However, optical flow also presents some problems, for example, with lighting changes, as they can produce displacement vectors that are not desirable. New algorithms try to alleviate those problems, although there is no way of addressing them for all the cases. However, in the available datasets the lighting conditions are stable so the optical flow algorithm seems to be the most appropriate choice.

The first part of our pipeline, the optical flow images generator, receives L consecutive images and applies the TVL-1 optical flow algorithm [35]. We chose TVL-1 due to its better performance with changing lighting conditions compared to other optical flow algorithms. We took separately the horizontal and vertical components of the vector field (d_t^x and d_t^y, resp.) and stack them together to create stacks $O \in \mathbb{R}^{224\times224\times2*L}$, where $O = \{d_t^x, d_t^y, d_{t+1}^x, d_{t+1}^y, \ldots, d_{t+L}^x, d_{t+L}^y\}$.

More precisely, we used the software tool (https://github.com/yjxiong/dense_flow/tree/opencv-3.1) provided by Wang et al. in [36] to compute the optical flow images (see Figure 2 for an example of its output). We kept the original optical flow computation parameters of Wang et al. to replicate their results in action recognition. Then we used the exact same configuration to compute the optical flow images of the fall detection datasets. As the CNN has learned filters according to the optical flow images of the action recognition datasets, using the same configuration for the fall detection images minimizes the loss of performance due to the transfer learning.

3.2. Neural Network Architecture and Training Methodology. The CNN architecture was a pivotal decision for the design of our fall detection system. There have been many architectural designs for image recognition in recent years (AlexNet [37], VGG-16 [31], and ResNet [38], among others) which have been equally used in computer vision problems. In particular, we chose a modified version of a VGG-16 network following the *temporal net* architecture of Wang et al. [8] for action recognition. The use of such architecture was motivated by the high accuracy obtained in other related domains.

More concretely, we replaced the input layer of VGG-16 so that it accepted a stack of optical flow images $O \in \mathbb{R}^{w\times h\times2L}$, where w and h are the width and height of the image and $2L$ is the size of the stack (L is a tunable parameter). We set $L = 10$, the number of optical flow images in a stack, following [39], as a suitable time window to accurately capture short-timed events such as falls. The whole architecture (see Figure 3) was implemented using the Keras framework [40] and is publicly available (https://github.com/AdrianNunez/Fall-Detection-with-CNNs-and-Optical-Flow).

We followed a three-step training process to train the network for fall detection with a double objective:

(i) To address the low number of fall samples in public datasets: a deep CNN learns better features as more labeled samples are used in the training phase. For instance, the Imagenet dataset, which is widely used for object recognition tasks in images, has 14 million images [7]. Current fall datasets are very far from those figures; thus, it is not feasible to learn robust and generic features for fall detection based only on those datasets. In such cases, transfer learning has shown to be a suitable solution [41].

(ii) To build a generic fall detector which can work on several scenarios: this requires developing a generic feature extractor, able to focus only on those motion patterns that can discriminate fall events from other corporal motions.

The training steps for transfer learning, summarized in Figure 4, are the following:

FIGURE 3: VGG-16 architecture: convolutional layers in green, max pooling layers in orange, and fully connected layers in purple. We followed the same notation of the original paper.

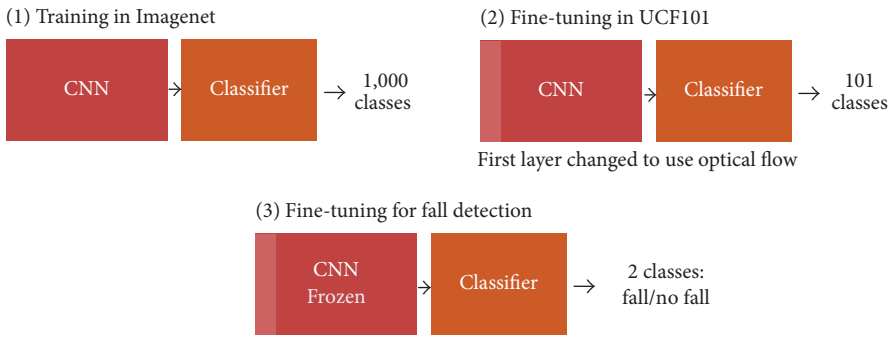


FIGURE 4: The transfer learning process applied to the neural network. (1) Full training with Imagenet to learn a generic feature extractor. (2) Training in UCF101 to learn to model motion. (3) Fine-tuning to build a fall detector.

(1) We trained the original VGG-16 net in the Imagenet dataset [7], which has over 14 million images and 1,000 classes. This is a standard practice in the deep learning literature [42], as the network learns generic features for image recognition; for example, it can distinguish corners, textures, basic geometric elements, and so on. Even though the target is to process optical flow images rather than RGB images, Wang et al. [8] argue that the generic appearance features learned from Imagenet provide a solid initialization for the network in order to learn more optical flow oriented features. This is due to the generic features learned in the first layers of the network, as they are useful for any domain. Then, only the top part must be tuned to adapt to a new dataset or domain.

(2) Based on the CNN trained on Imagenet, we modified the input layer to accept inputs $O \in \mathbb{R}^{224\times224\times20}$, where 224×224 is the size of the input images of the VGG-16 architecture and 20 is the stack size, as described in [8]. Next, we retrained the network with the optical flow stacks of the UCF101 dataset [9]. This dataset contains 13,320 videos and has 101 human actions in them. This second step allowed the network to learn features to represent human motion, which could later be used to recognize falls.

(3) In the final step, we froze the convolutional layers' weights so that they remained unaltered during training. To speed up the process, we saved the features extracted from the convolutional layers up to the first fully connected layer, hence having arrays of features $F \in \mathbb{R}$ of size 4,096 for each input stack. Basically, the third step consists in fine-tuning the remaining two fully connected layers, using dropout regularization [43] with 0.9 and 0.8 dropping probabilities.

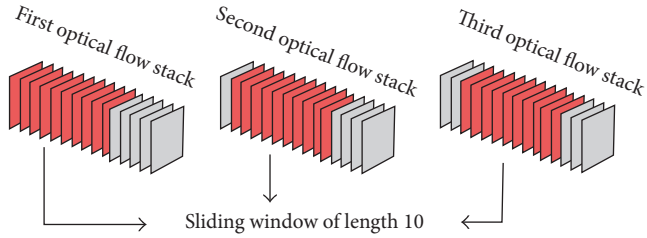


FIGURE 5: Sliding window method to obtain stacks of consecutive frames.

For the fine-tuning with a fall dataset, we extracted L frames from fall and "no fall" sequences (extracted from the original videos) using a sliding window with a step of 1 (see Figure 5). This way, we obtained $N - L + 1$ blocks of frames, assuming N is the number of frames in a given video and L the size of the block, instead of N/L from a nonoverlapping sliding window. We did not apply other data augmentation techniques. To deal with imbalanced datasets we resampled (without replacement) the data labeled as "no fall" to match the size of the data labeled as fall.

Even after balancing the datasets, the learning of the fall class seemed to be difficult. As we could see from the results of Section 4, the network did not perform as good as with the "no fall" class. Thus, we needed alternative ways of increasing the importance of the fall class in the learning process such as modifying the loss function. This function of a neural network expresses how far our predictions are from the ground truth, being the guide for weight updates during

Table 1: Number of frames of each dataset, distribution of frames per class (fall and "no fall"), and number of fall/"no fall" samples (sequences of frames corresponding to a fall or a "no fall" event).

Dataset	Total frames	Fall frames	No fall frames	Falls #	No falls #
URFD	11,936	900	11,036	30	973
Multicam	261,137	7,880	253,257	184	376
FDD	108,476	3,825	104,651	127	221

training. In particular, we chose the binary cross-entropy loss function, defined in

$$\text{loss}(p,t) = -\left(t \cdot \log(p) + (1-t) \cdot \log(1-p)\right), \quad (1)$$

where p is the prediction of the network and t is the ground truth. A way of increasing the importance of a class consists in adding a scaling factor or "class weight" to the loss function. Therefore, the loss function is finally given by

$$\text{loss}(p,t) = -\left(w_1 \cdot t \cdot \log(p) + w_0 \cdot (1-t) \cdot \log(1-p)\right), \quad (2)$$

where w_0 and w_1 are, respectively, the weights for the "fall class" and "no fall class," p is the prediction of the network, and t is the ground truth. A class weight of 1.0 means no change in the weighting of that class. The use of a higher class weight for the class 0, that is, w_0, penalizes the loss function for every mistake made on that class more than the mistakes on class 1. A neural network always tries to minimize the loss by adapting its weights; this is the base of the backpropagation algorithm [44]. Therefore, by using this modified loss function, we are encouraging the network to prioritize the learning of one of the classes. However, this might come at the price of worsening the learning of the other class. For that reason, in Section 4 we present the metrics that show the performance of each class separately.

Although the use of a w_0 greater than 1.0 biases the learning towards falls (in case of $w_1 = 1$), we argue that this bias is convenient in fall detection because of the importance of detecting a fall even at the price of having some false alarms. A missed detection would be critical in the health of the elderly and is, therefore, something to avoid.

3.3. Datasets. We selected three datasets that are often used in the literature, which makes them suitable for benchmarking purposes: the UR Fall Dataset (URFD) (http://fenix.univ.rzeszow.pl/~mkepski/ds/uf.html), the Multiple Cameras Fall Dataset (Multicam) (http://www.iro.umontreal.ca/~labimage/Dataset/), and the Fall Detection Dataset (FDD) (http://le2i.cnrs.fr/Fall-detection-Dataset?lang=fr):

(i) URFD contains 30 videos of falls and 40 videos of Activities of Daily Living (ADL), which we label as no falls.

(ii) Multicam contains 24 performances (22 with at least one fall and the remaining two with only confounding events). Each performance has been recorded from 8 different perspectives. The same stage is used for all the videos, with some furniture reallocation.

(iii) FDD contains 4 different stages (in contrast to the previous ones) with multiple actors.

The available datasets are recorded in controlled environments with various restrictions:

(i) There is only one actor in a video.

(ii) Images are recorded under suitable and stable lighting conditions, avoiding dark situations or abrupt lighting changes.

The falls appear in different positions of the scenario, far and near of the camera. Specially in the Multicam dataset, where there are eight cameras available, the distance to the camera varies significantly. In the FDD dataset, some falls are also far from the camera, although this is not a general case like in Multicam.

All videos have been segmented in frames and divided between falls and no falls, following the provided annotations. Table 1 summarizes the most relevant figures of each dataset.

In Section 4, we compare our results with the state of the art on all three datasets. Furthermore, we believe that the combination of the three datasets provides also a good indicator of the generality of our approach.

4. Results and Discussion

To validate our fall detector system we set up several experiments using the datasets of Section 3.3. In particular, we conducted four types of experiments, namely, (i) experiments for network configuration analysis, with the aim of finding the most suitable configuration for the problem; (ii) experiments to compare our method with the state-of-the-art approaches for fall detection; (iii) experiments to test the system in different lighting conditions; and (iv) an experiment to prove the generality of the system by combining all datasets.

4.1. Evaluation Methodology. From the point of view of supervised learning, fall detection can be seen as a binary classification problem on which a classifier must decide whether specific sequences of video frames represent a fall or not. The most common metrics to assess the performance of such a classifier are *sensitivity*, also known as recall or true positive rate, and *specificity* or true negative rate. These metrics are not biased by imbalanced class distributions, which make them more suitable for fall detection datasets where the number of fall samples is usually much lower than the number of nonfall samples. For fall detection, the sensitivity is a measure of how good our system is in predicting

Table 2: Results of our system on the three datasets (URFD, FDD, and Multicam) using different setups. In the minibatch size column, "full" means batch training and w_0 is the value of the weight given to the fall class. Notice that the ReLU activation function is always preceded by batch normalization. Sens. refers to sensitivity, whereas Spec. is used for specificity.

Dataset	Learning rate	Minibatch size	w_0	Activation function	Sens.	Spec.
URFD	10^{-5}	64	1	ReLU	100.0%	94.86%
Multicam	10^{-3}	Full	1	ReLU	98.07%	96.20%
FDD	10^{-4}	1,024	2	ELU	93.47%	97.23%

falls, whereas specificity measures the performance for "no falls." Nevertheless, for the sake of comparison with other existing approaches, we also computed the accuracy in the case where no other metric was given in their original papers. Therefore, the three evaluation metrics we used are defined as follows:

$$\begin{aligned} \text{Sensitivity/Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\ \text{Specificity} &= \frac{\text{TN}}{\text{TN} + \text{FP}}, \\ \text{Accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \end{aligned} \tag{3}$$

where TP refers to *true positives*, TN to *true negatives*, FP to *false positives,* and FN to *false negatives*. Given a video, we evaluated the performance of the system for each optical flow stack; that is, we checked the prediction for a stack with respect to its real label. We used a block size of 10 consecutive frames to create the stack. This number was empirically found by [39]. Consequently, we define the values mentioned above as follows:

(i) TP: an optical flow stack labeled as fall and predicted as fall

(ii) FP: an optical flow stack labeled as "no fall" and predicted as fall

(iii) TN: an optical flow stack labeled as "no fall" and predicted as "no fall"

(iv) FN: an optical flow stack labeled as fall and predicted as "no fall"

4.2. Experimental Setup. We conducted four main experiments:

(1) Search for the best configuration*:* we evaluated different network configurations in terms of sensitivity and specificity to analyze their impact on each dataset. More specifically, we investigated the role of the learning rate, minibatch size, class weight (explained in Section 3.2), and the use of the Exponential Linear Unit (ELU) [45] activation function compared with the Rectified Linear Unit (ReLU) preceded by Batch Normalization [46] (as discussed by Mishkin et al. in [47]). Regarding the minibatch size, for some experiments we used batch training (whole data is seen in each update) instead of minibatch training (different data chunks per update).

(2) Comparison with the state of the art*:* using the best configuration found in the first experiment, we compared our results with those of the literature. Again, the evaluation was performed in terms of sensitivity and specificity and, in some specific cases, accuracy was also used.

(3) Test with different lighting conditions*:* in order to provide an understanding of how the system would cope with different lighting conditions (not seen in the datasets of Section 3.3), we conduct two experiments: one with the images darkened and another one with a dynamic light. The evaluation was performed using the sensitivity and specificity in all the cases.

(4) Generality test*:* to know to which extent our solution is generic, we made an experiment combining all three datasets. The evaluation was again performed using the sensitivity and specificity values.

Regarding the optimization of the network parameters, we used Adam (all parameters apart from the learning rate are set as mentioned in [48]) for 3,000 to 6,000 epochs, depending on the experiment's computational burden. For slow training in the first experiment (search for the best configuration), we applied early stopping with a patience of 100 epochs, that is, if the loss is not improving for 100 epochs the training stops. This is a suboptimal greedy strategy to get the best model that avoids full training when it may not be necessary. We always used a value of 1 for the class weight w_1 in the loss function (see (2)), as we were not interested in increasing the importance of the "no fall" class.

4.3. Best Configuration Results. In the search of the best network configuration, we split each dataset into training and test set with an 80 : 20 ratio and a balanced distribution of labels (0 or 1, fall, or "no fall").

The search in the space of configurations included multiple experiments using different learning rates. We also varied the values of the minibatch size (with the option of batch training) applying a base w_0 class weight of 2 at first and then modifying it in the directions that seemed promising. The ReLU with Batch Normalization and ELU options were used equally, although we chose the most encouraging one if the results were favorable to it. The results are summarized in Table 2.

The results of the different configurations are essentially dependent on the number of samples we had to train the network. However, we observed that some configurations and

a range of values for hyperparameters performed well in all the datasets:

(i) Learning rate: the network performed the best with learning rate values around 10^{-3} and 10^{-5}. In fact, the best models for the three datasets use 10^{-3}, 10^{-4}, and 10^{-5} as learning rate. However, a higher or lower value creates some problems that are reflected in the sensitivity and specificity. More concretely, we often see that the falls are learned better. This behavior could be explained by the analysis given in Section 4.3.1. In other cases, extreme values for the learning rate harm the performance in "no fall" class (specificity). This is natural as in both cases (very high and low learning rate), it is harder to make the network converge: when the learning rate is high the network takes too big steps to be able to get to the minimum of the loss function and when it is low it stays far from the minimum, possibly due to a saddle point where it gets stuck.

(ii) Minibatch size: we used minibatch sizes ranging from 64 to 1,024 using powers of 2, as it is commonly seen in the literature, and batch training, where all the samples of a dataset are used to update the weights in each epoch (notice that this is possible due to the low amount of data of these datasets). In the case of URFD, we employed smaller batch sizes because the amount of samples is not high enough. We obtained the best results using batch training or a minibatch size of 1,024, so we deduce that a large amount of samples in each batch is highly beneficial, as more data allow the network to learn a better gradient. Otherwise, if a class is underrepresented in a minibatch, applying different class weights may cause problems in the gradient calculations, thus making more difficult the convergence of the learning process. The other values (64, 128, and 256) seem not to affect significantly the results. For some cases a small value of 64 performs better than 256, whereas the opposite case also exists depending on the dataset. Therefore, the results obtained by small minibatch sizes may be explained by the randomness of the batch construction.

(iii) Class weight: the class weight w_0 is important to increment the importance of the fall class at training time, as the "no fall" class was empirically found to be learned better than the fall class. This is reflected in the results with high values of the specificity ("no fall" class performance) in contrast to the lower values of sensitivity (fall class performance). When this happens, a value of w_0 higher than 1.0 helps achieving higher sensitivity. The base value was set 2.0 and incremented or decremented when we observed promising results.

The results show that a value for w_0 of 2.0 is adequate in general, although we see a value of 1.0 for the best results in URFD and Multicam. However, in the case of URFD, we got very similar results for the value of 2.0 (having in both cases perfect sensitivity). Therefore, if we had to pick a standard value for all the datasets, a value of 2.0 would have been adequate too, even though it did not produce the highest result in all the datasets.

(iv) ELU versus ReLU with BN: motivated by the work of Mishkin et al. [47] on different configurations for CNN architectures, we tested ELU and ReLU with Batch Normalization in our experiments to know which one could be more beneficial. In general, ReLU produces rather stable metrics, even improving the results when the batch size is higher, while ELU destabilizes when we do batch training; that is, we can obtain a 100% of sensitivity and 0% of specificity for some values. However, it is not the general case, as there are some exceptions; for example, the best result for FDD was obtained while using ELU. In any case, we can assume that ELU is not as reliable as the combination of ReLU and Batch Normalization which, following our experiments, shows a more stable behavior.

4.3.1. Analysis of False Alarms and Missed Detection. To further analyze the system and the performance of the best configuration, in this part we will discuss the false positives (false alarms) and false negatives (missed detection) produced by the network for the FDD dataset. We selected this dataset for its greater variety of falls with respect to URFD and Multicam, that is, different ways of falling with rare cases included. We sampled 72 sequences for the analysis from all the sequences with errors; half of the samples were fall sequences and the other half were "no fall" sequences.

False Positives or False Alarms. A false alarm is given when the system predicts as a fall a stack of optical flow that was labeled as "no fall." In the set of samples used for the analysis, we detected some common sources of errors for the majority of the cases, while others were uncommon cases. The errors were found stack-wise, that is, each 10 frames. As we are using a sliding window approach, the errors may be overlapping. We computed the amount of stacks per error source and we ordered the errors taken into account that number of stacks for each source divided by the total amount of stacks of all the sources.

(i) In 51.41% of the stacks, we observed that the system was learning that the frames before and after the fall were also considered part of the fall. More accurately, the frames containing the actor destabilizing (previous to the fall) and the frames where the actor was already in the floor (after the fall) were predicted as a fall.

(ii) With a lower occurrence rate, yet appearing in various events, we have the following cases:

(a) 14.06%: the actor slowly bends down and then lies down on the floor.

(b) 9.64%: the actor enters the FOV of the camera walking.

(c) 6.43%: from a seated position in a chair, the actor bends down to grab an object in the floor.

(d) 5.62%: the actor exits the FOV of the camera walking.

(iii) Finally, the remaining cases have a small occurrence rate, appearing in a few stacks and in a unique sequence. For example, (i) when the actor is lying down on the floor; (ii) when a small part of the actor goes out of the FOV of the camera; and (iii) when the actor grabs something from the floor while maintaining his legs stiff.

False Negatives or Missed Detection. When optical flow stacks labeled as fall are fed into the network and the predicted output is "no fall," a missed detection is given. We observed the following source of errors for the analyzed optical flow stacks of the set of 36 sequences (ordered in the same way as the false positives):

(i) 42.82% of the cases do not contain anything special during the fall; thus, we hypothesize that the network may not be learning a specific feature correctly.

(ii) Two other events compose the 14.65% and 10.70% of the cases: (i) the actor walks swinging, trying not to fall, for almost 2 seconds (48 frames) and (ii) the actor falls while grabbing a ball in his hands and all the movement occurs in the axis perpendicular to the floor.

(iii) Even with lower occurrence rates (7.04% and 5.92%), we have two events: (i) the lying-on-the-floor position is not detected as fall and (ii) the trunk starts the fall, while the legs stay stiff.

(iv) Finally, there are more minor events than in the case of the false positives, for example: (i) the actor starts the fall while being on his knees; (ii) the actor does not entirely fall to the floor, ending in a quadruped position; and (iii) the actor is seen almost from above and the fall is not appreciated.

We observed that all those errors could be classified into two groups depending on the source of the error:

(1) Events that do not appear many times in the datasets are difficult to learn, as the network does not have enough samples: for example, falling while on the knees or detecting falls from a top view of the person, which are among the analyzed samples. This may also be the case of almost the majority of the false negatives, where there is no explanation for the error apart from the incapability of the network to learn correctly specific features.

Our hypothesis is that the system can learn those rare cases with the proper amount of data. Judging from the results obtained in later Sections 4.5 and 4.6, we believe the generalization capability of the system is higher and we are only limited by the available datasets (see Section 3.3 for their limitations). Thus, this source of error could be addressed by our system.

Table 3: Comparison between our approach and others from the vision-based fall detection literature for URFD.

Proposal	Sensitivity/Recall	Specificity	Accuracy
Zerrouki and Houacine (2017) [14]	-	-	96.88%
Ours	100.0%	92.00%	95.00%

Table 4: Comparison between our approach and others from the literature of vision-based fall detection for Multicam.

Proposal	Sensitivity/Recall	Specificity
Wang et al. [15]	89.20%	90.30%
Wang et al. [16]	93.70%	92.00%
Ours	99.00%	96.00%

Table 5: Comparison between our approach and others from the vision-based fall detection literature for FDD.

Proposal	Sensitivity/Recall	Specificity	Accuracy
Charfi et al. (2012) [17]	98.00%	99.60%	-
Zerrouki and Houacine (2017) [14]	-	-	97.02%
Ours	99.00%	97.00%	97.00%

(2) The second source of error comes from the limitations of the cameras and the optical flow algorithm. The quality of the images is given by the cameras and, therefore, it is an intrinsic feature of the datasets. The optical flow algorithm has also its own limitations (discussed in Section 3.1); for example, in the case of a long distance between the actor and the camera, the optical flow algorithm is not able to capture the movement of the person and the output result is a blank image. This is a limitation of our system that must be studied case by case and included in the future work (Section 5).

4.4. Results and Comparison with the State of the Art. To compare the best models found in the previous section with the state of the art, we used a 5-fold cross-validation for URFD and FDD and a leave-one-out cross-validation for Multicam following Rougier et al. [19], in order to compare on equal conditions. In this last case, we split the dataset into 8 parts of the same size, each one containing all the videos recorded by a specific camera. We trained with 7 of those parts and tested with the remaining one; the final result is an average of the metrics given by each test. Nevertheless, for the three datasets, we balanced each fold in order to have the same amount of falls and "no falls." Notice that we retrain all the networks so that no information is brought from the experiments in Section 4.3.

Tables 3, 4, and 5 show the results obtained by our approach on each dataset compared to others. For the sake of a fair comparison, we selected the papers of the state of

the art which meet two requirements: (1) they work only on RGB data (no depth maps or accelerometer data) and (2) they provide results using publicly available datasets.

Due to the different performance metrics used by other researchers and the comparison criteria established for this paper in Section 4.1, it is not possible to make a general claim in terms of performance. Hence, we will discuss the results obtained for each dataset:

URFD

(i) Harrou et al. [24] and Zerrouki et al. (2016) [23] used both URFD and FDD but do not specify which dataset was used to obtain their results or how they combined the performance on both datasets. For these reasons, these works are not included in the comparison tables. Harrou et al. [24] reported results by means of the False Alarm Rate (FAR, or False Positive Rate) and Missed Detection Rate (MDR, or False Negative Rate) metrics, obtaining a FAR of 7.54% and a MDR of 2.00%. Using our systems, we obtained a FAR of 9.00% and a MDR of 0.00%. Zerrouki et al. (2016) [23] reported a sensitivity of 98.00% and specificity of 89.40%, while we obtained 100.0% and 92.00%, respectively. Again, our values correspond to training and testing only in URFD.

(ii) In a different work, Zerrouki and Houacine (2017) [14] reported an accuracy of 96.88% in this dataset, while we obtained 95%. Since the dataset is very imbalanced (see Table 1), it suffers the problem known as the accuracy paradox, where a higher accuracy does not imply a higher predictive power. In fact, a system predicting "no fall" for all samples in the dataset would obtain a 91.53% of accuracy without detecting any of the existing falls. For that reason, as explained in Section 4.1, we chose to show sensitivity and specificity values instead.

Multicam

(i) Both Wang et al. [15] and Wang et al. [16] evaluated the performance of their systems over stacks of 30 frames. Our system instead outperforms their results by using only stacks of 10 frames. More precisely, our system achieves a sensitivity of 99% and a specificity of 96%, while Wang et al. [15] obtained 89.20% and 90.30% and Wang et al. [16] 93.70% and 92.00% for the same metrics, respectively.

(ii) Regarding the work by Auvinet et al. [25] and Rougier et al. [19], they are not included in the comparison tables due to their evaluation methodology (http://www.iro.umontreal.ca/~labimage/Dataset/technicalReport.pdf). They used a criterion that sets a variable called t_{fall}, representing the frame where the fall starts, therefore dividing each video into two parts. If a fall is detected after t_{fall}, it is considered a TP; otherwise, it is a FN. Before t_{fall}, if a fall is predicted all the "no fall" period is considered a FP; otherwise, it is a TN. We believe the evaluation methodology is not useful to compare our solution with other authors; thus, we discard its use.

Furthermore, Auvinet et al. [25] used ad hoc threshold values to detect a lying-on-the-floor position, not the fall sequence itself. With this method, they reported sensitivity and specificity values up to 99.70%, while we obtained 99.00% and 96.00%, respectively. Rougier et al. [19], with an accuracy value of 99.70% compared to our 97.00%, performed fall detection in a video-level, not by stacks of frames. Hence, their system needs a specific data framing, using videos as input instead of a continuous stream of images, which is not ideal for real world deployments.

Moreover, Auvinet et al. [25] used a 3D vision system, which provides them with more information about the entire scenario than in our approach, although we are only three points behind in specificity from their results, showing that our system performs comparably even with less information.

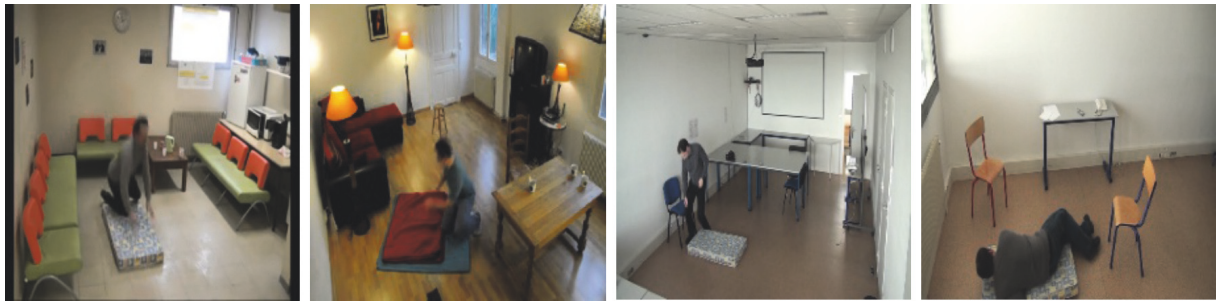
The use of 3D vision systems stirs up an interesting discussion about the strengths and weaknesses of both 3D and 2D systems. Regarding detection performance, based on the available experiments and results, the differences between both approaches are minimal. 3D vision systems show higher sensitivity and specificity, but the difference is low (around 3 points for Multicam). However, there are some other aspects to be considered to decide which approach is the most suitable for a given application. For instance, 3D vision systems are often based on active sensors, such as structured light (e.g., Kinect) or time-of-flight cameras, which may be used even in dark conditions. That is not the case of passive RGB cameras. Even though we show promising results in simulated scenarios with poor light conditions (Section 4.5.1), those cameras cannot work without external light.

On the other hand, as far as system deployment is concerned, 3D vision systems usually present higher costs. First of all, reliable 3D cameras are expensive compared to passive 2D cameras. The Kinect is an exception, but it has several limitations: it does not work in sunny environments and its range is limited to 4-5 meters. Second, 2D passive cameras are already common in public spaces. So it seems natural to try to take advantage of the existing installations.

In conclusion, we can claim that each system has its own advantages and drawbacks; hence, they should be selected depending on the specific application domain. This fact stresses even more the need of working on fall detection systems for different sensor deployments.

FDD

(i) As in the case of URFD, Harrou et al. [24] and Zerrouki et al. (2016) [23] mentioned the combined



(a)

(b)

Figure 6: Original images of the FDD dataset (a) and the same images after artificial darkening (b).

use of FDD but only provided a single result, thus making the comparison with them unclear. Harrou et al. reported a FAR of 7.54% and a MDR of 3.00%, while we get 3.00% and 1.00%. Zerrouki et al. provided a sensitivity of 98.00% and specificity of 89.40%, while we obtained 99.00% and 97.00%, respectively (again, only in FDD).

(ii) Similar to the URFD case, Zerrouki and Houacine (2017) [14] presented a 97.02% of accuracy for the FDD dataset, while we obtained 97.00%. As in the previous case with Zerrouki and Houacine in URFD, the evaluation using pure accuracy is misleading, as a system predicting always "no fall" would obtain a 96.47% of accuracy with null predictive power.

(iii) Charfi et al. [17] used ad hoc hand-tuned thresholds in their system to detect a fall (11 consecutive fall predictions), indeed obtaining very high results: 98.00% of sensitivity and 99.60% of specificity. It is not clear how this system would perform in other datasets, as the thresholds were hand-tuned for FDD without any other proof of generalization.

4.5. Experiments with Lighting Conditions. As discussed in Section 3.3, the public benchmark datasets used for this research present stable lighting conditions, providing suitable lighting for artificial vision tasks. However, in real world scenarios events like sunlight coming through the windows, a lamp switching on and off and so forth is quite frequent. Therefore, we decided to test how our system would behave under those circumstances. To that end, we modified the original images of the FDD dataset, thus creating different artificial lighting conditions, and observed how the fall detection system performs. We divided the experiments into two parts:

(1) Static lighting experiments, where we changed the lighting conditions to simulate night-like scenarios.

(2) Dynamic lighting experiments, where we added a dynamic artificial lighting that smoothly increases its intensity from frame to frame until reaching a specific value. Afterwards, the intensity decreases again to achieve the initial lighting conditions.

4.5.1. Static Lighting. In this first part, for every frame in a video, we subtract a constant value of 100 to each pixel of each channel (three channels in RGB) so that they get darkened as if it was night (see Figure 6). With these new images, we will do the following two experiments.

Training on Original Images Only. We divided the dataset of 80 : 20 ratio into two sets, train and validation, and balanced the class distribution of the train set. We selected darkened images for the train set and the original ones (unchanged) for the validation set. Then, we trained the model for 3,000 epochs with a learning rate of 0.001, batch training, a w_0 of 2, and ELU (the best configuration for the FDD dataset found in Section 4.3). We obtained a sensitivity of 45.85% and a specificity of 98.67%.

The result is coherent with the fact that falls are difficult to detect when the actor approaches the floor. It is the darkest area in the image; thus, the actor is not distinguishable, as it gets fused with the darkness. Therefore, any lying-on-the-floor position is very difficult to detect.

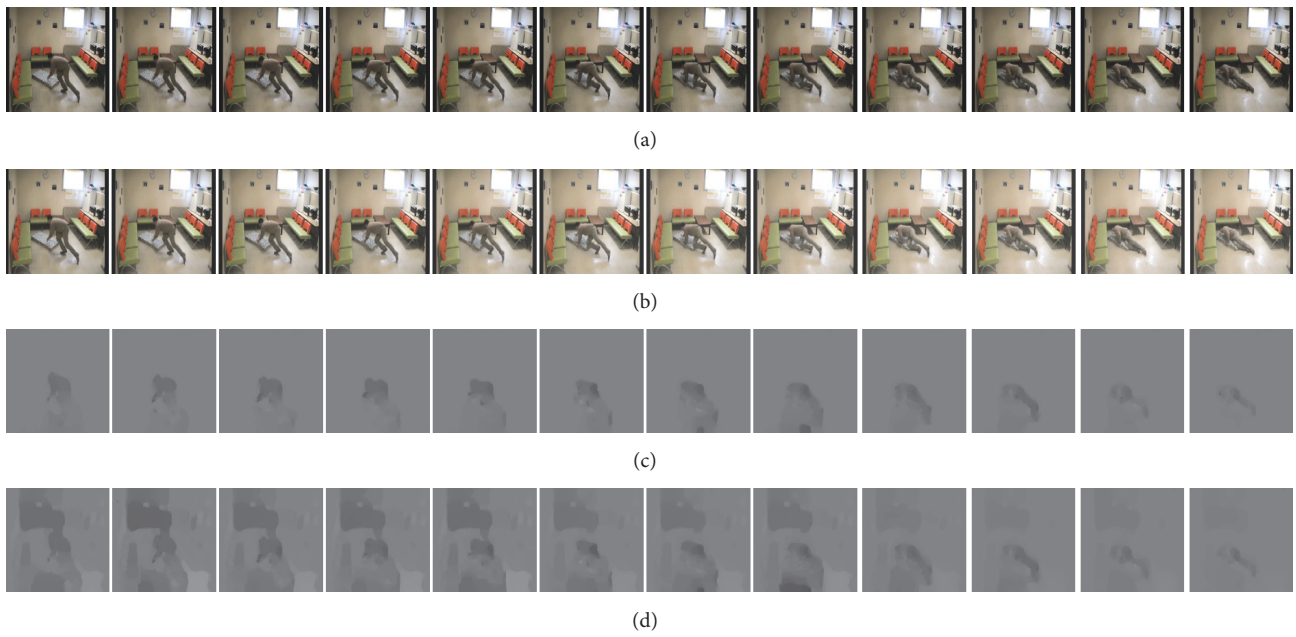


Figure 7: Original images of the FDD dataset in (a); the same images with simulated dynamic lighting in (b); optical flow images without the lighting change in (c); and optical flow images with the lighting change in (d). The 12 frames correspond to half a second of the original video (recorded at 25 FPS).

Train and Test with Darkened Images. We used exactly the same configuration and train/test partition of the previous experiment, but this time the images from the training set were also darkened. After training, the sensitivity went up to 87.12%, while the specificity decreased a bit (94.92%). The best result obtained with this configuration for the original dataset was 93.47% of sensitivity and 97.23% of specificity. We believed the difference is not that large taking into account the level of darkening applied and the fact that we did not explore the best configuration for the new images.

4.5.2. Dynamic Lighting. In real world scenarios, lighting conditions are not as stable as in a lab environment. For example, a lamp may be switched on/off in the background, generating displacement vectors in the optical flow algorithm. To simulate this type of scenarios in the FDD dataset, we added a progressive change of lighting that takes 32 frames to light up and fade. As the video was recorded at 25 frames per second (FPS), the lighting lasts for about 1.3 seconds.

To produce this dynamic light change, for each frame, channel, and pixel we multiplied the original value by a sinusoidal function so that the transition between frames emulates real light conditions. This modification was done once per video at its first 32 frames (see Figure 7). In order to achieve more realistic illumination conditions, a single lighting change was applied on each video. This part again is divided into two experiments.

Train with the Original Images and Test with the New Ones. To test how the system reacts to dynamic lighting conditions, we trained the model with the original images and tested with the new ones (as explained in the first paragraph). Again, we divided the dataset into an 80 : 20 ratio (keeping the same data in each partition as in Section 4.5.1). We trained the model for 3,000 epochs with a learning rate of 0.001, batch training, a w_0 of 2, and ELU. The result is a 28.04% of sensitivity and a 96.35% of specificity.

The result is coherent with the data, as the dynamic lighting generates lots of displacement vectors in the image. This confuses the network that has been trained to see only the displacement vectors of the moving person.

Train and Test with the New Images. Finally, we check how the system is able to adapt to this lighting change if the classifier is properly trained. To this end, we used the new images in the train set and the validation set (the same partition of sets as in the first part of this experiment). The configuration of the training and the network is also the same as in the previous part. This time the system obtains a 90.82% of sensitivity and a 98.40% specificity.

Like in the previous experiment with darkness, both metrics increase significantly when trained with the new data. In particular, this change is big in the case of the sensitivity, which goes from 28.04% to 90.82% after being trained with the modified samples. This is a proof of the capability of the network to adapt to new circumstances (darkness or lighting changes in this case) if the appropriate data is used in training time.

4.6. Generality Test. One of the main drivers of our system design was the generality (Section 3), that is, to develop a fall detector able to perform in different scenarios. While previous experiments in this paper considered each dataset individually, here we tried to avoid any singular feature

Table 6: Experiment with the system trained with the three datasets combined (URFD, Multicam, and FDD). Results are shown on the combination and the individual sets.

Test set	Sensitivity/Recall	Specificity
URFD + Multicam + FDD	94.00%	94.00%
URFD	100.0%	99.00%
Multicam	85.00%	84.00%
FDD	97.00%	98.00%

associated with a specific dataset. For that purpose, we generated a new dataset as the combination of the three previously used as follows:

(1) In order to give equal weight to all three datasets, we resampled the two largest sets to match the size and class distribution of the smallest one (URFD). With this change, the three datasets had the same relevance (amount of samples) and both classes (fall and "no fall") were balanced. More concretely, each dataset has 960 samples, 480 fall and 480 "no fall" samples.

(2) In order to apply a 5-fold cross-validation, we divided each dataset into 5 groups, each group containing the same amount of fall/"no fall" samples.

The results of the experiment with the performance on the combined set but also on the samples of each set individually are shown in Table 6. The network configuration was the following: a learning rate of 10^{-3}, a batch size of 1,024, and a w_0 of 2.0, and we used ReLU with Batch Normalization. The results correspond to the 5-fold cross-validation, with each fold trained for 1,000 epochs.

We believe that these results support our claim of having a generic fall detector system, mainly based on two reasons:

(1) Our system has been tested in three different public datasets, obtaining the state-of-the-art results on all three of them. To the best of our knowledge, our system is the first one achieving such results on those three reference datasets. Notice that all three datasets present different characteristics. For example, the fall annotating criterion of FDD differs from the other two datasets. While FDD considers that a fall starts when a person is inclined around 45 degrees with respect to the floor, Multicam and URFD label a fall when instability is perceived. Also, FDD contains totally different scenarios in contrast to the other two datasets. Another major difference is that Multicam records the same falls from different point of views or perspectives, since eight cameras are used in the same stage. Even though there are significant differences among the datasets, our system achieves a performance equal to the best state-of-the-art algorithms on those datasets. We think this is a solid proof of the generality of our fall detector.

(2) As the experiments performed for each dataset implied training a FC-NN classifier for each of them, it can be argued that the fall detector is properly trained in each case for the given dataset. The experiment shown in Table 6 tries to refute that reasoning. We combined the three datasets both for training and testing, with all their differences. As can be seen, when we tested the system using the videos from all the datasets, we obtained very high detection rates (sensitivity and specificity of 94%). When observing the results obtained on each individual dataset, our results still remain high, except for Multicam (sensitivity of 85% and specificity of 84%). This can be explained because Multicam uses different perspectives to record the same events. When we generated the combined dataset, we discarded many frames from Multicam to keep the influence of each dataset equal. In that process, we lost many frames which may have been helpful for our network to learn the different perspectives. Thus, we can conclude that to tackle the perspective issue, more images are needed in the training process. However, the results back up our generality claim, since the system was able to learn generic features from different datasets and showed high detection rates. Under more realistic conditions (real world environment), the system may get poor results unless trained in real world data too. Therefore, the key to obtain generic features and being able to generalize well is training the system with a huge amount of inhomogeneous data.

The generalization capabilities of our system came mainly from two design decisions: (i) the use of optical flow images, which only model the motion information contained in consecutive frames and avoid any dependence with appearance features, and (ii) the three-step training phase, where the network learns generic motion-related features for action recognition.

5. Conclusions

In this paper, we presented a successful application of transfer learning from action recognition to fall detection to create a vision-based fall detector system which obtained the state-of-the-art results in three public fall detection datasets, namely, URFD, Multicam, and FDD. To the best of our knowledge, this is the first fall detector system to achieve that, showing the generality of the approach. We further tested the generality of our system with various experiments with different lighting conditions and an extra experiment where we combined all three datasets. We firmly believe that the results obtained constitute a solid base for our generality claim.

The key ideas presented throughout the manuscript are as follows:

(i) The use of optical flow as input to the network: in contrast to other approaches, by computing optical flow images and stacking them we (i) take into account the correlation among consecutive frames (compared to those who use each frame separately) and (ii) achieve environmental independence avoiding any appearance-based features, thus making the system applicable to different scenarios. This algorithm also

has its drawbacks, as stated in Section 3.1. Nevertheless, with the appropriate training we can make up for it and obtain very good results, as demonstrated in Section 4.5.

(ii) The use of CNN retrained in different datasets and for different problems: apart from creating a powerful feature extractor, this allows us to not depend on hand-engineered features, which are usually very hard to design and are prone to be too specific to a given setup. In contrast, our CNN learned generic features relative to the problem domain.

(iii) Transfer learning: to overcome the problems posed by the low number of samples in fall datasets and learn generic features, we adopted transfer learning techniques. More concretely, we presented a three-step training process which has been successfully performed for fall detection.

We believe that the presented vision-based fall detector is a solid step towards safer Smart Environments. Our system has been shown to be generic and works only on camera images, using few image samples (10) to determine the occurrence of a fall. Those features make the system an excellent candidate to be deployed in Smart Environments, which are not only limited to home scenarios. Based on emerging IoT architectures, the concept of Smart Environments can be extended to many other everyday environments, providing the means to assist the elderly in several contexts.

However, there is still ground for improvement. In order to bring vision-based fall detection to real world deployments, we envisage three potential research directions:

(1) Further research on transfer learning with fall detection datasets is warranted in order to improve our generic feature extractor. Currently, our third training step is limited to fine-tuning (Section 3), where the convolutional layers have their weights frozen and only the classifier layer is really trained. We would like to consider alternative ways; going deeper in the way convolutional layers can be adapted to fall datasets. However, those experiments must be carried out carefully to avoid too specific feature extractors, which may perform better in a certain dataset but at the expense of losing generality.

(2) Using optical flow images provides a great representational power for motion, but also involves the heavy computational burden of preprocessing consecutive frames and drawbacks concerning lighting changes. Following the philosophy of end-to-end learning, we would like to avoid any image preprocessing step and work only on raw images in the future. Therefore, more complex network architectures will have to be designed to learn complete and hierarchical motion representations from raw images.

(3) As the public datasets have only one actor per video we believe that the next step in the field of fall detection would be the multiperson fall detection. For this task, we think that region-based CNNs (R-CNN) [49] could be a promising research direction, with the aim of automatically detecting different persons in images and analyze those regions with our fall detection system.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

The authors gratefully acknowledge the support of the Basque Government's Department of Education for the predoctoral funding and NVIDIA Corporation for the donation of the Titan X used for this research. They also thank Wang et al. [8] for making their work publicly available.

References

[1] A. F. Ambrose, G. Paul, and J. M. Hausdorff, "Risk factors for falls among older adults: A review of the literature," *Maturitas*, vol. 75, no. 1, pp. 51–61, 2013.

[2] J. C. Davis, M. C. Robertson, M. C. Ashe, T. Liu-Ambrose, K. M. Khan, and C. A. Marra, "International comparison of cost of falls in older adults living in the community: A systematic review," *Osteoporosis International*, vol. 21, no. 8, pp. 1295–1306, 2010.

[3] M. Mubashir, L. Shao, and L. Seed, "A survey on fall detection: principles and approaches," *Neurocomputing*, vol. 100, pp. 144–152, 2013.

[4] L. Chen, J. Hoey, C. D. Nugent, D. J. Cook, and Z. Yu, "Sensor-based activity recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 42, no. 6, pp. 790–808, 2012.

[5] A. Caragliu, C. Del Bo, and P. Nijkamp, "Smart cities in Europe," *Journal of Urban Technology*, vol. 18, no. 2, pp. 65–82, 2011.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] J. Deng, W. Dong, and R. Socher, "ImageNet: a large-scale hierarchical image database," in *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, Miami, Fla, USA, June 2009.

[8] L. Wang and et al., *Towards good practices for very deep two-stream convnets*, 2015.

[9] S. Khurram, A. R. Zamir, and M. Shah, *Amir Roshan Zamir, and Mubarak Shah., UCF101: A dataset of 101 human actions classes from videos in the wild*, 2012.

[10] M. Vallejo, C. V. Isaza, and J. D. Lopez, "Artificial Neural Networks as an alternative to traditional fall detection methods," in *Proceedings of the 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC 2013*, pp. 1648–1651, Japan, July 2013.

[11] A. Sengto and T. Leauhatong, "Human falling detection algorithm using back propagation neural network," in *Proceedings of the 5th 2012 Biomedical Engineering International Conference, BMEiCON 2012*, Thailand, December 2012.

[12] B. Kwolek and M. Kepski, "Human fall detection on embedded platform using depth maps and wireless accelerometer," *Computer Methods and Programs in Biomedicine*, vol. 117, no. 3, pp. 489–501, 2014.

[13] F. Harrou, N. Zerrouki, Y. Sun, and A. Houacine, "Statistical control chart and neural network classification for improving human fall detection," in *Proceedings of the 8th International Conference on Modelling, Identification and Control, ICMIC 2016*, pp. 1060–1064, Algeria, November 2016.

[14] N. Zerrouki and A. Houacine, "Combined curvelets and hidden Markov models for human fall detection," *Multimedia Tools and Applications*, pp. 1–20, 2017.

[15] S. Wang, L. Chen, Z. Zhou, X. Sun, and J. Dong, "Human fall detection in surveillance video based on PCANet," *Multimedia Tools and Applications*, vol. 75, no. 19, pp. 11603–11613, 2015.

[16] K. Wang, G. Cao, D. Meng, W. Chen, and W. Cao, "Automatic fall detection of human in video using combination of features," in *Proceedings of the 2016 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2016*, pp. 1228–1233, China, December 2016.

[17] I. Charfi, J. Miteran, J. Dubois, M. Atri, and R. Tourki, "Definition and performance evaluation of a robust SVM based fall detection solution," in *Proceedings of the 8th International Conference on Signal Image Technology and Internet Based Systems, SITIS 2012*, pp. 218–224, Italy, November 2012.

[18] T. Lee and A. Mihailidis, "An intelligent emergency response system: Preliminary development and testing of automated fall detection," *Journal of Telemedicine and Telecare*, vol. 11, no. 4, pp. 194–198, 2005.

[19] C. Rougier, J. Meunier, A. St-Arnaud, and J. Rousseau, "Robust video surveillance for fall detection based on human shape deformation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 5, pp. 611–622, 2011.

[20] S.-G. Miaou, P.-H. Sung, and C.-Y. Huang, "A customized human fall detection system using omni-camera images and personal information," in *Proceedings of the 1st Transdisciplinary Conference on Distributed Diagnosis and Home Healthcare, D2H2 2006*, pp. 39–42, USA, April 2006.

[21] C.-L. Liu, C.-H. Lee, and P.-M. Lin, "A fall detection system using k-nearest neighbor classifier," *Expert Systems with Applications*, vol. 37, no. 10, pp. 7174–7181, 2010.

[22] V. Vishwakarma, C. Mandal, and S. Sural, in *Pattern Recognition and Machine Intelligence*, Automatic detection of human fall in video, Ed., pp. 616–623, 2007.

[23] N. Zerrouki, F. Harrou, A. Houacine, and Y. Sun, "Fall detection using supervised machine learning algorithms: A comparative study," in *Proceedings of the 8th International Conference on Modelling, Identification and Control, ICMIC 2016*, pp. 665–670, Algeria, November 2016.

[24] F. Harrou, N. Zerrouki, Y. Sun, and A. Houacine, "A simple strategy for fall events detection," in *Proceedings of the 14th IEEE International Conference on Industrial Informatics, INDIN 2016*, pp. 332–336, France, July 2016.

[25] E. Auvinet, F. Multon, A. Saint-Arnaud, J. Rousseau, and J. Meunier, "Fall detection with multiple cameras: An occlusion-resistant method based on 3-D silhouette vertical distribution," *IEEE Transactions on Information Technology in Biomedicine*, vol. 15, no. 2, pp. 290–300, 2011.

[26] S. Gasparrini, E. Cippitelli, S. Spinsante, and E. Gambi, "A depth-based fall detection system using a Kinect® sensor," *Sensors*, vol. 14, no. 2, pp. 2756–2775, 2014.

[27] R. Planinc and M. Kampel, "Introducing the use of depth data for fall detection," *Personal and Ubiquitous Computing*, vol. 17, no. 6, pp. 1063–1072, 2013.

[28] G. Diraco, A. Leone, and P. Siciliano, "An active vision system for fall detection and posture recognition in elderly healthcare," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, DATE 2010*, pp. 1536–1541, deu, March 2010.

[29] G. Mastorakis and D. Makris, "Fall detection system using Kinect's infrared sensor," *Journal of Real-Time Image Processing*, vol. 9, no. 4, pp. 635–646, 2012.

[30] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "PCANet: a simple deep learning baseline for image classification?" *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5017–5032, 2015.

[31] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014.

[32] Y. Jia, E. Shelhamer, J. Donahue et al., "Caffe: convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, pp. 675–678, ACM, Orlando, Fla, USA, November 2014.

[33] S. S. Beauchemin and J. L. Barron, "The Computation of Optical Flow," *ACM Computing Surveys*, vol. 27, no. 3, pp. 433–466, 1995.

[34] J. J. Gibson, "The Perception of Visual Surfaces," *The American Journal of Psychology*, vol. 63, no. 3, p. 367, 1950.

[35] F. A. Hamprecht, C. Schnörr, and B. Jähne, ""A duality based approach for realtime TV-L 1 optical flow," in *Pattern Recognition*, pp. 214–223, 2007.

[36] L. Wang, Y. Xiong, Z. Wang et al., "Temporal segment networks: Towards good practices for deep action recognition," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 9912, pp. 20–36, 2016.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS '12)*, pp. 1097–1105, Lake Tahoe, Nev, USA, December 2012.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pp. 770–778, July 2016.

[39] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Proceedings of the 28th Annual Conference on Neural Information Processing Systems 2014, NIPS 2014*, pp. 568–576, can, December 2014.

[40] C. François and etal., "Keras," 2015, https://github.com/fchollet/keras.

[41] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the 27th IEEE Conference on Computer Vision and Pattern Recognition (CVPR '14)*, pp. 1717–1724, IEEE, Columbus, Ohio, USA, June 2014.

[42] S. Herath, M. Harandi, and F. Porikli, "Going deeper into action recognition: A survey," *Image and Vision Computing*, vol. 60, pp. 4–21, 2017.

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[44] R. Hecht-Nielsen, "Theory of the backpropagation neural network," *Neural Networks*, vol. 1, no. 1, p. 445, 1988.

[45] D. Clevert, T. Unterthiner, G. Povysil, and S. Hochreiter, "Rectified factor networks for biclustering of omics data," *Bioinformatics*, vol. 33, no. 14, pp. i59–i66, 2017.

[46] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015.

[47] D. Mishkin, N. Sergievskiy, and J. Matas, *Systematic evaluation of CNN advances on the ImageNet*, 2016.

[48] D. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014.

[49] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142–158, 2016.