Real-time Vision-based Fall Detection

with

Motion History Images and Convolutional Neural Networks

Truls Haraldsson

Computer Science and Engineering, master's level
2018

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

Luleå Tekniska Universitet

Institutionen för System- och rymdteknik

Master's thesis

Real-time Vision-based Fall Detection with Motion History Images and Convolutional Neural Networks

Author
T. Haraldsson

Supervisor
Dr. D. Walther
Dr. J. Hallberg

Examinator
Dr. J. Hallberg

October 5, 2018

Abstract

Falls among the elderly is a major health concern worldwide due to the serious consequences, such as higher mortality and morbidity. And as the elderly are the fastest growing age group, an important challenge for society is to provide support in their every day life activities. Given the social and economical advantages of having an automatic fall detection system, these systems have attracted the attention from the healthcare industry. With the emerging trend of Smart Homes and the increasing number of cameras in our daily environments, this creates an excellent opportunity for vision-based fall detection systems. In this work, an automatic real-time vision-based fall detection system is presented. It uses motion history images to capture temporal features in a video sequence, spatial features are then extracted efficiently for classification using depthwise convolutional neural network. The system is evaluated on three public fall detection datasets, and furthermore compared to other state-of-the-art approaches.

Contents

1 Introduction

Falls among the elderly is a major health concern worldwide due to serious consequences such as higher morbidity, mortality, lower functional abilities, and long-time admission to recovery facilities [1]. As we age, our bodies goes trough many physical changes becoming more fragile to falls [2]. For example, our reaction times get slower [3], which has been associated with a greater risk of falling [4]. Our vision is poorer, making it harder for us to detect obstacles. Medication can also affect us by making us sleepy or dizzy which in turn affect our balance.

Socialstyrelsen [5] reported that in Sweden, more than 1000 people die each year as a result of a fall, and 70000 people get such serious injuries that they have to be put in a hospital. A significant part of these accidents occurred among the elderly. The World Health Organization [6] reported that 30% of the elderly experience at least one fall each year, and that the frequency of falls increases exponentially with higher age. Meanwhile, shortage in manpower is becoming an increasing problem around the world [7]. Because of this, there is an interest from the healthcare industry to have a system that can assist elderly and personnel in case of a fall.

1.1 Background

Sigma Connectivity is developing a smart home system that aims to assist elderly in their everyday living, as well as personnel in their work. They have already developed a system that can detect anomaly behaviour such as when a person stays for too long in the bathroom, or the front door has been open for too long i.e forgot to close the door. The current system is built around a central hub that is installed locally and communicates with a set of sensors. More sensors can be added to the system after the installation in an ad hoc manner. A suitable extension and enhancement to this smart home system would be a fall detection system that can be easily integrated with the existing hub.

1.2 Motivation

Fall detection is an important service for the healthcare, especially for the elderly, and a reliable system to detect a fall early is a necessity to reduce the post-effects of falls [8, 9]. As the number of people over 60 years is growing faster than any other age group [6], a challenge to handle the increasing number of fall accidents emerge. An automatic fall detection system could increase the independent living ability among elderly, and also reduce the manual labour in terms of presence of support staff.

Several approaches for fall detection systems have been proposed [10] over the years. These can be divided into three categories; *wearable-*, *ambient-*, and *vision* based systems. Wearable based systems utilize accelerometers and gyroscopes to detect a fall, while ambient based systems analyze vibrations and sound to distinguish a fall from normal activity. Vision based systems use cameras and image processing algorithms to monitor a person's body pose or motion as a way to detect a fall. Each of these categories have their pros and cons as explained further in Section 2.

For this project, a vision based solution is desirable since the company has already developed special cameras for elderly care. A vision based system is also interesting when considering recent years of advancement in deep learning [11], especially image classification [12, 13]. At a wider scope, not just fall detection for indoor usage, cameras are installed at several public places, which would make it possible to use the already installed infrastructure for deployment of a fall detection system over a larger area.

1.3 Problem definition

As part of a larger project that aims to modernize elderly care using modern technologies, the goal of this project will be to implement a fall detection system that could potentially be used as a subsystem for assisting elderly, support staff and other personal. Such a system would have to have include the following features:

- The system should work in real time, that is the system should detect a fall when it happens. More specifically, the fall detection algorithm needs to operate at a rate faster than the input signal i.e the video stream.
- The system should be able to recognize a fall up to six meters away, which is reasonable from a resident perspective where a normal sized room would be around 24 square meters, i.e. 6x4 meters.
- Low intrusiveness is desired as the persons using the systems should not feel that their privacy is intruded.
- Interoperability, as in the system should give a confidence value of how certain the system is that it has detected a fall. This would allow other systems, such as an alarm system, to listen for events and make decisions based on the confidence value.

1.4 Delimitations

This project will develop a vision based fall detection system even though there are other approaches. The main goal will be to develop a model using machine learning that can detect falls using images i.e. a video stream. As of such, the installation of a real system using cameras and computational units will be left out. Also, the part of the system that would alarm a third party in case of fall event will not be developed.

1.5 Thesis structure

The remainder of this thesis is organized as follows: In Section 2 we review some existing work within the domain of fall detection systems. In Section 3 we take a closer look at how machine learning works and compare two different pre-processing algorithms for motion detection. In Section 4 the system implementation is described in more details. In Section 5 we evaluate the system with respect to the problem definition. In Section 6 we compare our solution with already existing ones, followed by a discussion about how the work went and drawbacks of the current system.

2 Related work

Plenty of research have been done within the domain of automatic fall detection systems [10, 14, 15], this include development of new algorithms and hardware technologies. These systems often fall into one of three categories as described by Mubashir [10]: *Wearable*-, *Ambient*-, or *Vision*-based systems.

Wearable based systems are a common solution and include the usage of accelerometers and gyroscopes, or even barometers. These sensors are often attached on the persons body, common places are around the waist and wrists, under the armpit, or behind the ear's lobe.

In [16] they gathered acceleration data using a tri-axial accelerometer attached to the person. This data was then analyzed and compared to threshold values. Vallejo [17] and Leauhatong [18] proposed to feed acceleration data into a multilayered perceptron (MLP) instead of using threshold values. Ojetola [19] extended the use of accelerometer based devices by including tri-axial gyroscopes. The system used a machine learning decision tree for detection. Tabar [20] developed a smart home system where the user was equipped with a badge that could communicate wirelessly with network nodes. The badge provided a voice communication channel to a command center in case of a fall. The system could also approximate a persons geographical position, which was used to detect inactivity. Finally cameras was used to visually validate the prediction of the sensors.

These system are relatively cheap compared to other systems because of the low cost of using accelerometers. However, the problems encountered with these systems are related to their sensors' sensitivity leading to low accuracy, and the fact that older people either forget or not willingly wear the devices.

Ambient based systems attempt to use audio, visual and vibrational data. Xiaodan [21] proposed a system that uses audio signal from a far-field microphone. The audio segments was then classified using a support vector machine (SVM). In [22], sound and vibrational data were used as input into a pattern recognition algorithm to discriminate between fall and other events. Toreyin [23] fused sound, vibrational and passive infrared sensors. Wavelet features were extracted from raw sensor output and then analyzed to detect a fall. These features are then simultaneously used to train a Hidden Markov model (HMM) to distinguish regular and irregular activities of an elderly person.

These systems are often cost effective and less intrusive compared to the wearable based systems. However, as with the wearable, the accuracy of these system often suffer from the fact that the sensors are to sensitive. That is, in case of the pressure sensors, they sense pressure from everything in and around the object.

Vision based systems focuses purely on the frames from video streams to detect a fall. These systems usually use RGB- or depth-cameras together with image processing algorithms that extract feature such as bounding boxes or silhouettes to facilitate the detection of falls.

Extracting shape related features is a widely used technique [24, 25, 26] when detecting a fall. In [24] they use the aspect ratio of width and height to detect a fall. Miaou [25], improves on this by knowing the height and width aspect ratio, of the subjects, beforehand and thus able to have a more reliable system. Mirmahboub [26] extract the silhouette of a person using background subtraction and extracting several features from the silhouette area. Then a SVM was applied for classification of these silhouette-related features. Zerrouki and Houacine [27] also used and SVM to classify posture of a silhouette, but then also used these predictions as input to a HMM to classify video sequences.

Motion analysis to distinguish normal daily activities such as walking, sitting, etc, from an abnormal activity as falling have been studied [28, 29, 30, 31]. In [28], time and time of motion is integrated into a integrated time motion image which is like a database that stores time and motion. Finally a MLP neural network is employed for classification of motions of a fall event. Lin [29] used a similar approach where a motion history image was created using a sequence of frames. Acceleration and angular acceleration was then extracted using a bounding ellipse to further increase the detection accuracy. Caroline [30] showed that the 3D head trajectory can be tracked using a single calibrated 2D camera, and finally the velocities of the head are used to characterize a fall event. Charif [32] extracted spatio-temporal features by analyzing a sequence of silhouettes that were extracted using background subtraction. These features were then classified using a SVM. Recently Núñez-Marcos [31] showed that using a convolutional neural network (CNN) can achieve state of the art accuracy. Their system constructed optical flow images that was then fed into a CNN for classification.

While single RGB-cameras have been frequently used, 3D vision systems have been adopted too to take advantage of 3D structures. These systems requires multiple cameras or depth cameras such as Microsoft Kinect, Intel Real Sense or time-of-flight that can extract depth information. Yang [33] used a Kinect camera to extract 3D information from both the floor and subject to detect a fall. Auvinet [34] used multiple cameras to create a 3D shape of the person. Fall events were then detected by analyzing the volume distribution along the vertical axis. Zian-Peng [35] proposed a method that tracks the joints of a person using the Kinect SDK. An SVM was then used to classify a fall using the head joint trajectory as input. In [36] a Kinect camera was used to create 3D bounding boxes around the target. The velocity of the target was then calculated using the contraction and expansion of the width, height and depth of the boxes.

Even though 3D vision based systems brings more information to analyze, they have their drawbacks related to deployment. Multiple camera systems have to be synchronized to target same area, and depth cameras have limited range, narrower sight and a higher price tag. Thus, from the point of view of system deployment, 2D passive systems are usually a better option, given their lower cost.

The main disadvantage of vision based systems is the risk of occlusion. Also, in case where a classifier has been trained using a single RGB-camera, the systems become viewpoint-dependent which is problematic if you have to move the camera.

As seen in previous works, from the view of methodology, fall detection methods could be divided into rule based and machine learning based methods. Rule based methods suffer from generality, that is, the systems often becomes case specific. To mitigate this problem, machine learning has been applied. Still, a lot of systems use hand crafted features such as head tracking, acceleration, volume boxes, etc., as input to these machine learning models, which pose a problem for a general solution. In [31] they showed that classifying features learned by a convolution neural network can achieve state-of-the-art performance. In this work, a similar approach is taken where a CNN is used for spatial feature extraction, but as input, embedded temporal features from a video sequence called motion history images are used.

3 Theory

In this section the theory and mechanisms behind a neural network's learning capabilities are looked into in more detail, in particular neural networks using *supervised learning* for the *binary classification* problem. We will cover *cost functions* and how they relate to the *backpropagation algorithm* that is the essential part of the learning procedure, we will also look at how neural networks can learn spatial relationship in image data using convolutional operations, and how this convolutional operation can be optimized. At the end we will go through two techniques for how one can capture temporal relationship between images in an video stream. This section also lays the ground for the choices of techniques with respect to the problem definition in Section 1.3.

3.1 Neural network

Neural networks, a biologically-inspired programming paradigm that let computers learn from observational data. Mathematically a neural network can be seen as a function

$$f : X \mapsto Y,$$

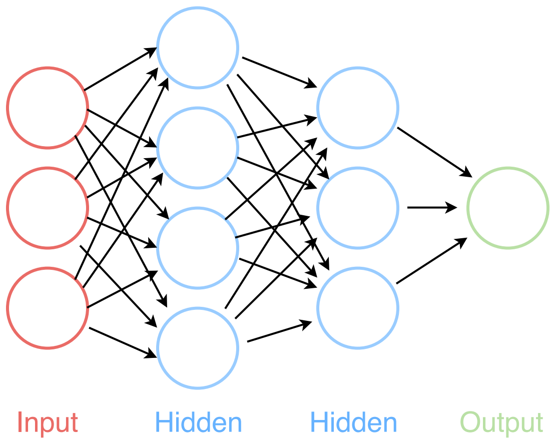

Figure 1: A neural network showing the input-, hidden- and output layers.

where $f(x)$ is a composition of other functions $g_i(x)$, that can further be decomposed into other functions. This can be illustrated as a network of nodes that have arrows showing the dependencies between functions, see figure 1. Networks, for practical reasons, are often structured in layers that are stacked together where a layer l_k's output is layer l_{k+1}'s input. These layers are generally called *densely* or *fully connected* layers in the machine learning literature. Generally, a neural network with m layers have one input layer l_1, one output layer l_m, and an arbitrary number of hidden layers l_k for $1 < k < m$ in between.

A single neuron can be described as a composition of an *input function* and an *activation function*, where the input function is a weighted sum of all incoming connections from the previous layer's neurons, and the activation functions is a non-linear function described further in Section 3.1.1. For a node i in layer l_k, the input function can be described as

$$a_i^k = b_i^k + \sum_{j=1}^{n_{k-1}} w_{ji}^k o_j^{k-1}, \tag{1}$$

where b_i^k is a bias, w_{ij}^k is the weight of the connection between node j in layer l_{k-1} and node i in layer l_k, o_j^{k-1} is the output of the activation function for node j in layer l_{k-1}, and n_{k-1} is the number of nodes in layer l_{k-1}. For a visualization see figure 2.

3.1.1 Activation functions

The neuron in a neural network is just a function, as mentioned before, that takes an input and gives an output. These functions are called activation functions, inspired from the biological neuron in a brain that fires a signal when activated. In order to model a non-linear response variable, i.e. a class label, with respect to the explanatory variables, the activation functions have to be

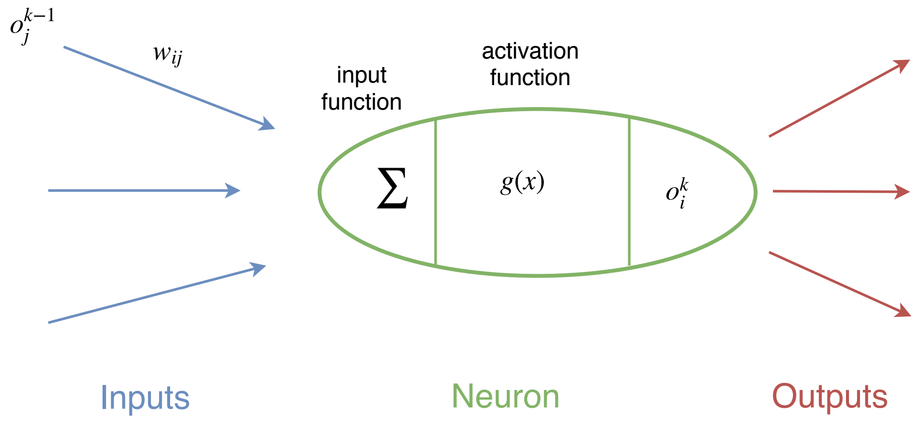

Figure 2: Shows a single neuron and how it sums together the incoming signals that are then fed as input to the activation function $g(x)$.

non-linear. That is, if the activation functions were linear they could just be summed up into a new single linear function. Two such functions, see figure 3, are *rectifier*

$$ReLU(x) = max(0, x), \tag{2}$$

and *sigmoid*

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3}$$

Rectifier is extensively used as the activation function within the hidden layers of neural networks, as it has a desired property that the derivative of the function is easily calculated, i.e. either 0 or 1. This is favourable, as we will see later, for the backpropagation algorithm. It has also been demonstrated in [37] that ReLU enables better training of deeper networks when used in the hidden layers compared to other activation functions. Sigmoid has the property that it always output a value in the interval (0, 1), which is well suited for the binary classification problem.

3.1.2 Cost function

In the binary classification problem, the task is, given a set of elements to divide them into two sets. This can be defined as

$$F : X \mapsto Y, \tag{4}$$

where F is a function that maps $x \in X$ to $y \in Y$, where $Y = \{0, 1\}$. To estimate how well a function F, i.e. a neural network, is at correctly classifying input data, a cost function is used, such that

$$C : F \mapsto \mathbb{R},$$

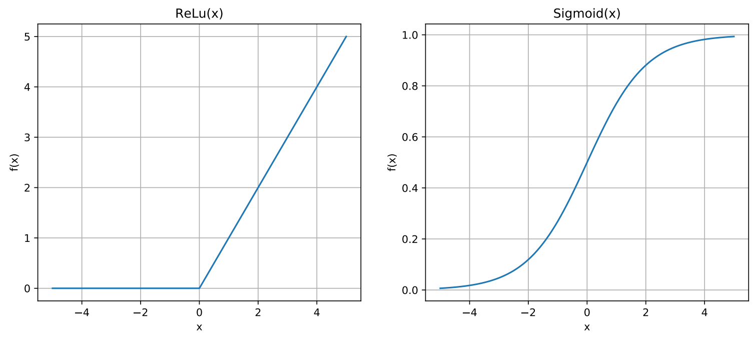

Figure 3: The activation functions ReLu and Sigmoid

where C is a cost function, and F is the model to be evaluated. Given (x, y) such that y is the class of x, and $\hat{y}$ is the value of x under the transformation F in equation 4. Then the *binary cross entropy* cost function can be defined as

$$C = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}). \tag{5}$$

Note that equation 5 is only defined when $0 < \hat{y} < 1$. Because of this, a function like sigmoid, defined in equation 3, is well suited as the activation function in the output layer.

To increase the importance of one class, the weighted cost function is defined as

$$C_W = -w_1 \cdot y \log \hat{y} - w_0 \cdot (1 - y) \log(1 - \hat{y}), \tag{6}$$

where w_0 and w_1 are class weights for respective class $y \in Y = 0, 1$. This effectively penalizes a miss prediction on a class with a higher weight associated to it.

3.1.3 Backpropagation

Learning in neural networks can be described as:

- Given a class of functions F and a set of observational data, learning is finding the optimal function $f^* \in F$, such that for a given cost function $C : F \mapsto \mathbb{R}$, the optimal solution f^*, $C(f^*) \leq C(f) \forall f \in F$.

In other words, given a neural network and a set of weights, learning is finding optimal weights for solving the given task. Backpropagation [38], short for backwards propagation, is an algorithm for supervised learning that approximate f^* using gradient decent. The algorithm tries to minimize the cost function C with respect to the weights w_{ij}^k by calculating the gradient $\frac{\delta C}{\delta w_{ij}^k}$, such that

$$w_{ij}^t = w_{ij}^{t-1} - \alpha \frac{\delta C}{\delta w_{ij}^{t-1}}, \tag{7}$$

where α is the *learning rate*, and t is the time step. If α is small the learning takes longer time and also risk getting stuck in a local minimum, if too large the algorithm might have problem finding a minimum. The gradient term in equation 7 can be expanded using the chain rule, thus

$$\frac{\delta C}{\delta w_{ij}^k} = \frac{\delta C}{\delta a_j^k} \frac{\delta a_j^k}{\delta w_{ij}^k}. \tag{8}$$

The second term on the right hand side in equation 8 can be calculated using simple derivation of equation 1,

$$\frac{\delta a_j^k}{\delta w_{ij}^k} = \frac{\delta}{\delta w_{ij}^k} \sum_{l=0}^{n_{k-1}} w_{lj}^k o_i^{k-1} = o_i^{k-1}. \tag{9}$$

The first term on the right hand side in equation 8 is however dependent on the activation functions, as well as which layer is considered. Using a model with m layers for binary classification where the last layer contains one neuron, then rewriting the cost function from equation 5 for the output layer gives

$$C = -y \log g(a_1^m) - (1-y) \log(1 - g(a_1^m)),$$

where g(x) is an activation function from Section 3.1.1. The error term then becomes

$$\frac{\delta C}{\delta a_1^m} = -\left(\frac{y}{g(a_1^m)} - \frac{1-y}{1-g(a_i^m)} \right). \tag{10}$$

Using equation 10 in equation 8 gives

$$\frac{\delta C}{\delta w_{i1}} = -\left(\frac{y}{g(a_1^m)} - \frac{1-y}{1-g(a_i^m)} \right) o_i^{m-1}. \tag{11}$$

For the hidden layers $1 \leq k < m$ the error term becomes

$$\frac{\delta C}{\delta a_j^k} = \sum_{l=1}^{n_{k+1}} \frac{\delta C}{\delta a_l^{k+1}} \frac{\delta a_l^{k+1}}{a_j^k}. \tag{12}$$

Rewriting equation 1 using an activation function $g(x)$ for layer $k+1$ gives

$$a_l^{k+1} = \sum_{j=1}^{n_k} w_{jl}^{k+1} g(a_j^k),$$

then

$$\frac{\delta a_l^{k+1}}{a_j^k} = w_{jl}^{k+1} g'(a_j^k). \tag{13}$$

Using equation 13 and 12 gives the final equation for the error term in the hidden layers

$$\frac{\delta C}{\delta a_j^k} = g'(a_j^k) \sum_{l=1}^{n_{k+1}} w_{jl}^{k+1} \frac{\delta C}{\delta a_j^{k+1}}. \tag{14}$$

Finally, putting it all together using the partial derivative in equation 8 and the error function for the hidden layers in equation 14 gives

$$\frac{\delta C}{\delta w_{ij}^{k}} = g'(a_j^k) o_i^{k-1} \sum_{l=1}^{n_{k+1}} w_{jl}^{k+1} \frac{\delta C}{\delta a_l^{k+1}}. \tag{15}$$

As seen, the error term for a node in layer l_k is dependent on the error terms from layer l_{k+1}, i.e. the error is propagating from the final layer towards the first layer. The error term is also dependent on the derivative $g'(x)$, which is why *ReLU* is suitable, as it is computational inexpensive to calculate its derivative.

3.1.4 Convolution layers

In machine learning, specifically for object detection and classification of images, a neural network applies a convolutional operation as a mean of learning spatial relationships within the data. In it's most basic form, a convolutional operation can be seen as sliding a filter or kernel over an input image, and at each step calculating the dot product between the kernel and the image. This produces a new image called *feature map*. The speed or step size of the kernel operation is called *stride*. See figure 4 for a visualization of the convolutional operation.

A convolutional layer takes as input an $I_w \times I_h \times I_d$ feature map $\mathbf{I}$ and produces a $O_w \times O_h \times O_d$ feature map $\mathbf{O}$, where I_w and I_h is the width and height of the input, I_d is the number of input channels (input depth), O_w and O_h is the width and height of the output, and O_d is the number of output channels. The convolutional operation is parameterized by a kernel $\mathbf{K}$ of size $K_w \times K_h \times I_d \times O_d$, where K_w and K_h is the width and height of the kernel, and I_d and O_d is the input and output depth respectively. The n_{th} output map for convolution with a stride of one is computed as

$$\mathbf{O}_{k,l,n} = \sum_{i,j,m} \mathbf{K}_{i,j,m,n} \cdot \mathbf{I}_{k+i,l+j,m}. \tag{16}$$

The cost of this convolution is thus a function of kernel size, input size, and output depth. Assuming a stride of one, the computational cost for a convolution is

$$I_w \cdot I_h \cdot I_d \cdot K_w \cdot K_h \cdot O_d. \tag{17}$$

where the cost depends multiplicative of the including terms.

3.1.5 Depthwise separable convolution

A *depthwise seperable convolutional* layer consists of two sub layers. The basic idea is to replace a fully convolution operation, as in equation 16, with a factorized operation performed in two steps. The first operation is a *depthwise convolution*, that applies one convolutional filter per input channel. The second operation, called a *pointwise convolution*, is a 1×1 convolution that builds new

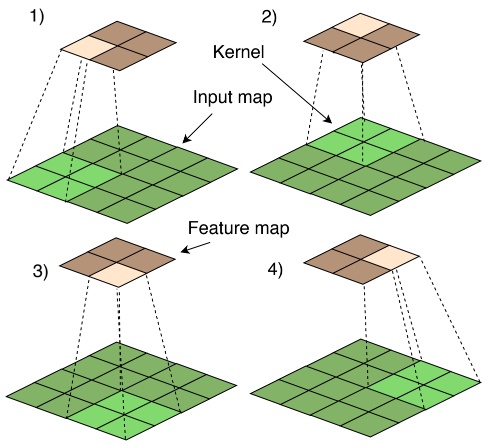

Figure 4: Shows a convolutional operation between a kernel of size 2×2 and an image of size 4×4 using a stride of two. The resulting feature map thus have size 2×2.

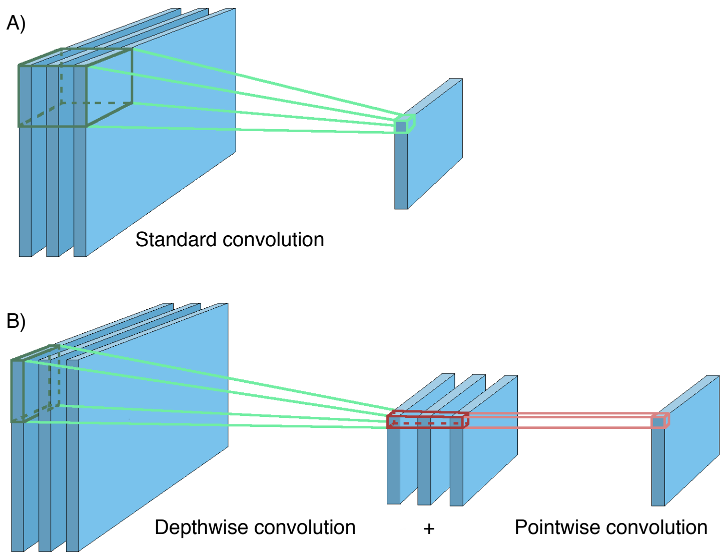

Figure 5: In A) a standard convolution is done by sliding a kernel across all channels. In B) a depthwise convolution using a single filter per channel, followed by a pointwise convolution to create the final feature map.

Table 1: Depthwise seperable vs standard convolution. Table taken from [39].

Model	ImageNet accuracy	Million Mult-adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

features through a linear combination of the output from the depthwise operation. See figure 5 for a visual comparison between standard and depthwise separable convolution.

A depthwise convolutional layer with one filter per input channel can be written as

$$\hat{\mathbf{O}}_{k,l,m} = \sum_{i,j} \hat{\mathbf{K}}_{i,j,m} \cdot \mathbf{I}_{k+i,l+j,m}, \tag{18}$$

where the m_{th} input feature in $\mathbf{I}$ is processed by the m_{th} filter in $\hat{\mathbf{K}}$ to produce the m_{th} feature in $\hat{\mathbf{O}}$. Thus the computational cost of the depthwise operation is

$$I_w \cdot I_h \cdot I_d \cdot \hat{K}_w \cdot \hat{K}_h.$$

The pointwise operation using $\hat{\mathbf{O}}$ as input is described as

$$\mathbf{I}_{k,l,n} = \sum_{m} \hat{\mathbf{K}}_{1,1,m,n} \cdot \hat{\mathbf{O}}_{k,l,m},$$

with the computation cost of

$$I_w \cdot I_h \cdot I_d \cdot 1 \cdot 1 \cdot O_d,$$

where $\hat{K}_w = \hat{K}_h = 1$. The total computational cost for a depthwise seperable convolutional layer is thus the sum of the depthwise and pointwise operations,

$$I_w \cdot I_h \cdot I_d \cdot (k^2 + O_d),$$

were $k = \hat{K}_w = \hat{K}_h$. This effectively reduce the computational cost compared to standard convolutional operations, see equation 17, by a factor of

$$\frac{k^2 \cdot O_d}{k^2 + O_d}.$$

This is desirable from the real time requirements stated in the problem definition 1.3. Looking at table 1, one can see a significant difference in the number of parameters between a depthwise separable convolutional network and a standard convolutional network.

3.1.6 Transferred learning

Transferred learning is a technique in deep learning that aims to transfer knowledge between two related domains, i.e. from a *source* network to a *target* network. By knowledge in neural networks, one mean the weights w_{ij}^k. The actual transfer is done by first training the source network on a given task, then the weights are transferred to the target network and retrained on a new task. This requires the network structures to be equal, i.e. same number of neurons and layers. The reason why this works, in the case of convolutional neural networks, is because the first layers generally learn basic features such as lines, edges, corners etc, while the later layers combine these features to learn to recognize higher level features such as squares, circles, and ultimately domain specific features e.g. cats, dogs, etc.

Transferred learning is also useful in the case when the target domain's dataset is limited. In [31] they trained their network on ImageNet [40] and retrained the last layers on their fall detection datasets.

3.2 Motion tracking

Motion tracking in computer vision is a process to distinguish a moving object from the background in a sequence of images. Two such methods are *optical flow* and *background subtraction* together with *motion history images.*

3.2.1 Background subtraction

A common pre-processing step in motion analysis is background subtraction. By having a fixed camera, a moving object can be distinguished from the background by calculating the frame difference between two consecutive frames. The difference function D can be defined as

$$D(x, y, t) = |I(x, y, t) - I(x, y, t - 1)|, \tag{19}$$

where $I(x, y, t)$ is the pixel value at (x, y) at time t for an image I.

3.2.2 Motion history image

A motion history image (MHI) is a way to represent motion over time in a single image. It keeps a history of temporal changes for each pixel, which then decays over time. The MHI $H_\tau(x, y, t)$ can be computed using an update function $\Psi(x, y, t)$:

$$H_\tau = \begin{cases} \tau & \text{if } \Psi(x, y, t) = 1. \\ max(0, H_\tau(x, y, t-1) - \delta & \text{otherwise.} \end{cases} \tag{20}$$

Here (x, y) and t shows the position and time, $\Psi(x, y, t)$ gives the motion in the current image, the duration τ decides the temporal extent of the movement (e.g. in terms of frames), and δ is the decay parameter. This function is called for every new frame to be analyzed in the video sequence. The resulting image will be in grayscale where more recently moving pixels are brighter and vice-versa.

The update function Ψ can be defined through a function as in equation 19,

$$\Psi(x,y,t) = \begin{cases} 1 & \text{if } D(x,y,t) > \xi, \\ 0 & \text{otherwise,} \end{cases} \tag{21}$$

where ξ is a threshold value, that is, the motion is only detected when the difference is larger than ξ. The time complexity of H_τ is linearly dependent on the number of pixels in the image. That is, calculating the difference between two images using the background subtraction method defined in 19 and then updating each pixel in the MHI, the total cost is

$$O(N), \tag{22}$$

where N is the number of pixels in the images.

3.2.3 Lucas-Kanade

The Lucas-Kanade optical flow algorithm is another way to estimate the motion in a sequence of images. The method tries to estimate the motion between two consecutive images by associating a vector (u, v) to each pixel by comparing the two consecutive images. These vectors are calculated through a *warp* operation that tries to align the two images, the result is then the motion between the two images. The algorithm makes one assumption:

- The two images are separated by a small time difference Δt, such that the moving objects are not displaced significantly, i.e. the algorithm works best for slow moving objects.

There are many variants of the Lucas-Kanade algorithm, each one with a slightly different computational complexity. In [41] they give an analysis for the time complexity of the original Lucas-Kanade implementation, which is

$$O(n^2 \times N + n^3), \tag{23}$$

where N is the number of pixels in the image and n is the number of warp parameters. For more details regarding the Lucas-Kanade algorithm and the warp parameters see [41].

Comparing Lucas-Kanade and MHI with respect to the real time constraints defined in Section 1.3 in terms of time complexity, it is clear from equation 23 and 22 that generating MHI is faster. On the other hand, motion history images provide less information compared to optical flow images, which could lead to poorer classification accuracy. Another reason opting for MHI over Lucas-Kadane has to do with the earlier assumption about the optical flow algorithm, this requires sampling images at a higher rate. In other words, it is possible to tune the sampling rate when updating the MHI such that every n_{th} frame is used, which can be used to tune the speed of the algorithm relative to the input signal i.e a video stream.

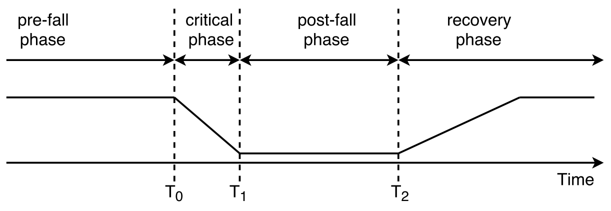

Figure 6: The four phases of a fall. The figure shows a fast downwards motion between T_0 and T_1, followed by a longer time period of inactivity, and finally a recovery-phase with a slower upward motion.

3.3 Definition of a fall

Noury [42] gave a definition of a fall. It consists of four phases, see figure 6; *pre-fall*, *critical*, *post-fall* and *recovery*. During the pre-fall phase the person performs usual activities of daily life like sitting and walking. The critical phase is when the body suddenly move towards the ground and ends with a sudden vertical stop. Next is the post-fall phase and is recognized by the inactivity of the person. Finally, the recovery phase is when the person either gets up on their feet by themselves, or by the help from anyone else.

4 Implementation

This section describes implementation details of the system. The system can be divided into four modules or steps, an input step that collects data, followed by a pre-procssing step where motion history images are generated, followed by a convolutional neural network for feature extraction, and finally a neural network of densely interconnected neurons that is used for classification. The implementation of the system is done using the computer vision library OpenCV [43], and two machine learning libraries TensorFlow [44] and Keras [45].

4.1 Input

The input module captures image data using a camera and forwards it to the preprocessing module. The current implementation is simulating a 25 *frames per second* (FPS) camera by using already stored and captured video data. More details of the camera used can bee seen in Section 4.4.

4.2 Pre-processing

The pre-processing step has three important tasks, sampling images from an input stream, re-scaling the sampled images, and finally generating motion history images. These steps are executed in that order, see figure 7.

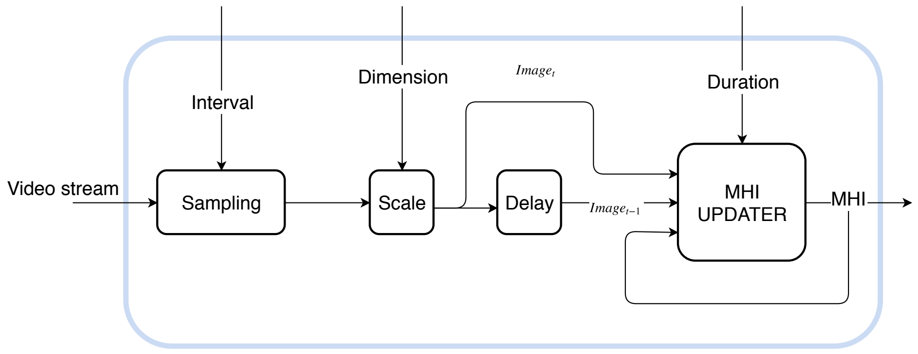

Figure 7: Shows an overview of the pre-processing module. It takes an image stream as input and outputs an motion history image.

The preprocessing module holds a state, that is the MHI, and is responsible for updating the state whenever a new image is provided to the module. There are three parameters that defines the behaviour of the state-updating-procedure:

- *Interval* specifies at which rate images are sampled from the image stream, e.g. if interval equals two, then every second image will be used for updating the MHI.
- *Dimension* specifies the width and height of the MHI in pixels.
- *Duration* specifies the number of frames embedded in the MHI, e.g. if duration is equal to five, then the last five sampled images are part of the MHI. It should be noted that the module only has to store the previous image as well as the current MHI, because once the MHI is updated the previous images can be discarded.

Using the aforementioned parameters, the module can be tuned to capture a time period T seconds of video footage in the MHI given a input stream with a given *frames-per-second* (FPS). This is described as

$$T = \frac{duration \times interval}{FPS}. \tag{24}$$

As seen in equation 24 the two variables interval and duration can be used to fine tune the time span. The preprocessing module uses $duration = 40$, $interval = 2$, and $dimensions = 128 \times 128$.

4.2.1 Updating MHI

The MHI updating function seen in figure 7 is an implementation based of H_τ, defined in 20. Below is the pseudocode of the implemented algorithm.

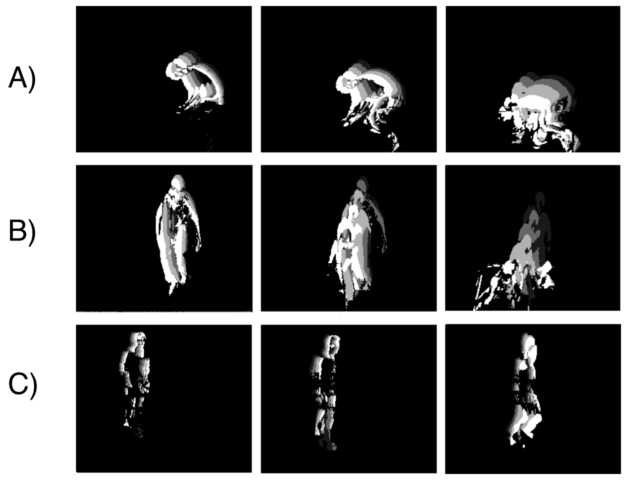

Figure 8: Shows generated MHIs. In A) a person is falling from a chair, B) a person is falling from standing position, C) a person walking around.

```
   Input  : A MHI $mhi$, the current image $I_c$, the previous image $I_p$, a
            duration $\delta$, and a threshold $\xi$.
   Output: An updated MHI $mhi^*$.
1 $I_d \leftarrow$ DiffRGB($I_c$, $I_p$);
2 $I_g \leftarrow$ ConvertToGrayScale($I_d$);
3 $I_t \leftarrow$ BinaryThreshold($I_g$, $\xi$);
4 $mhi^* \leftarrow$ Decay($mhi$, $\delta$);
5 $mhi^* \leftarrow$ Add($mhi^*$, $I_t$);
```

The algorithm first performs background subtraction using two consecutive sampled images as described in Section 3.2.1. The result from this operation can be seen as a snapshot of the motion between the two images. Then the algorithm converts the difference-image I_d to grayscale such that brighter pixels indicates more movement and vice versa. It should now be noted that OpenCV stores pixels values in the range $[0, 1]$, and of such, when deciding if a pixel have *moved* enough, each pixel is compared to a threshold value $\xi = 0.1$. The result I_t from this operation is now an image that contains only black or white pixels. Finally the state of the MHI gets updated in two steps. First by reducing all pixel values with $\frac{1}{duration}$, and secondly by adding the binary image I_t to the MHI. The result from this algorithm can bee seen in figure 8.

4.3 Feature extraction & Classification

The neural network consists of two parts, a convolutional network for feature extraction, and a densely connected neural network for classification. The whole network is implemented using the two libraries TensorFlow [44] and Keras [45].

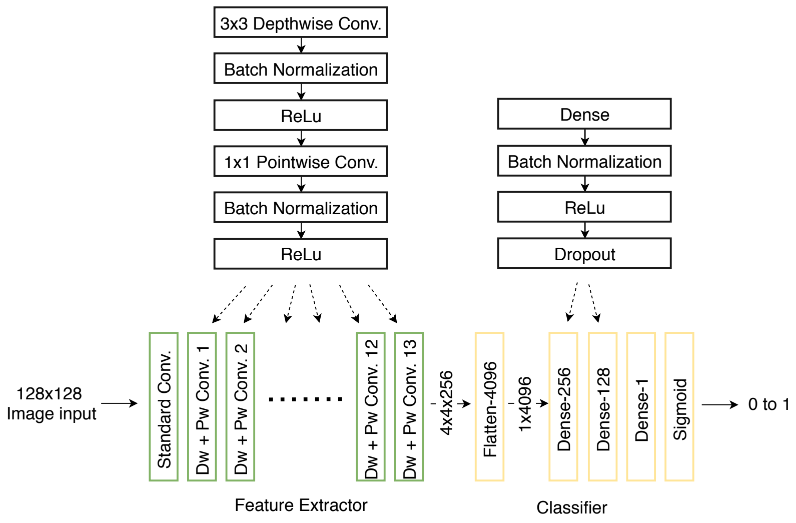

Figure 9: Shows the whole neural network. It consists of one standard convolutional layer, followed by 13 depthwise separable convolutional layers, and finally four fully connected layers. Note that everything in Keras is defined as layers, including the activation functions, batch normalization, and dropouts.

Keras is a wrapper library for TensorFlow, written in Python.

As a feature extractor, a modified version of MobileNets [39] was used, were all layers were kept except the last densely connected layers, which were removed. MobileNets employs depthwise separable convolutional layers as described in Section 3.1.5 to efficiently perform convolutional operations. MobileNets also has a parameter $\alpha \in (0, 1]$ that affects the width of the convolutional layers, i.e a layer with an input depth of N and output depth M becomes αN respectively αM. Implementing MobileNet was done using Keras' `keras.applications.MobileNet`, which is a predefined model, with a value of $\alpha = 0.25$. Because MobileNet was originally designed for a 1000-class classification problem, the last densely layers had to be removed and replaced with a new densely connected layers that would fit the binary classification problem. This was easily done as models in Keras are defined by stacking layers on top of each other, and thus only having to modify a limited part of the current pre-implemented model. The feature extractor can be seen in figure 9.

The classifier consists of four layers, one input layer and three densely connected layers. The input layer is `keras.layers.Flatten` that converts the output from the feature extractor, with shape $\mathbb{R}^{4\times4\times256}$, to a shape $\mathbb{R}^{4096}$. Then there are two densely connected layers, implemented using `keras.layers.Dense`, with the size of 256 and 128 neurons each, using the activation function ReLu as defined in 2. These two dense layers have a dropout chance of 0.9 and 0.8 respectively to prevent overfitting [46], and is implemented using `keras.layers.Dropout`. The last densely connected layer consists of one neuron using sigmoid, see equation 3, as activation function. See figure 9 for an overview of the classifier.

Both the classifier and feature extractor make use of batch normalization [47], and was implemented using Keras' `keras.layers.BatchNormalization`. Batch normalization has been shown to make the training of deep neural networks more efficient while also easing the parameter initialization.

4.4 Dataset

The *Fall Detection Dataset* (FDD) [32] was used for training. It was recorded using a camera with 25 FPS and a resolution of 320×240 pixels, and consists of 190 videos in different simulated environments including office, coffee room, home, and lecture room. By simulate, it means that the environments were staged and persons in the videos were falling on purpose. The falls were performed for different angles, falling backwards, forwards or to the sides, and also from different positions such as sitting and standing. Some videos would not include any fall but rather have similar motions such as picking up an object from the floor, or sitting down in a sofa or on a chair.

For training efficiency, not having to pre-process the dataset all the time, the whole FDD was converted to a new dataset called MD that consisted solely of motion history images. This was done by feeding the FDD frame by frame through the preprocessing module and recording the output. The generated MHI was divided into two classes, *fall* and *non-fall*. For a MHI that belongs to the fall-class, it has to have the most recent sampled image in the *critical-phase* interval as seen in the definition of a fall in Section 3.3, otherwise the MHI would belong to the non-fall-class

The generated MD can be seen in table 2. The imbalance of the dataset is because the original dataset, the FDD, consists of 75911 frames of which 4908 are labeled as fall and 71003 are labeled as non-fall. The reason there are more MHIs labeled as fall than there are frames labeled as fall in the original dataset is because an MHI can extend over several frames and thus include frames that are labeled as non-fall.

MD was further divided into three subsets: MD_{train}, MD_{val}, and MD_{test} with a ratio of 0.7, 0.2, and 0.1 respectively. To avoid any correlation between the subsets, MHIs originating from the same video would be in the same subset. The MD_{train} set was used for training, the MD_{val} set was used for validation during training and for finding good parameters for the model, and MD_{test} set was used for evaluation.

4.5 Training procedure

Two techniques were used for training the network, *transferred learning* as described in Section 3.1.6, and *fine tuning* where one train the network with a lower learning rate, see equation 7 to see how learning rate affects training. The training was done in three steps.

- Firstly, transferred learning was used where ImageNet [40] was the source

Table 2: Shows the number of frames that are labeled as fall and non-fall.

Dataset	Total frames	Fall frames	Non-fall frames
FDD	75911	4908 (6.5%)	71003 (93.5%)
MD	69716	9786 (14.0%)	59930 (86.0%)
$\mathrm{MD}_{\mathrm{train}}$	50743	6742 (13.3%)	44001 (86.7%)
$\mathrm{MD}_{\mathrm{val}}$	13146	2057 (15.6%)	11089 (84.4%)
$\mathrm{MD}_{\mathrm{test}}$	5827	987 (16.9%)	4840 (83.1%)

domain. This was easily done as Keras had weights for the convolutional layers of MobileNet pre-trained on ImageNet.

- Secondly, each MHI in MD_{train} was propagated through the feature extractor while recording the feature maps outputted from the last convolutional layers. These output features, called *bottleneck features*, were then used to train solely the classifier. This saved time as the data only had to flow through the convolutional layers once.

- Thirdly, to fine tune the network, the classifier was appended to the last convolutional layer of the feature extractor and trained as a whole using the generated MD_{train} but at a lower learning rate. The convolutional layers was initiated with the weights learned from ImageNet and the densely connected layers were initiated with the weights learned from previous step. By freezing all but the last convolutional layer and the densely connected layers, the weights of the non-frozen layers were the only ones that got updated during backpropagation.

For step two, the training was done for 3000 epochs using a learning rate of 1×10^{-3}. To avoid unnecessary training, the training would stop if no improvement had been done in the last 100 epochs. The same number of epochs and early-stopping method were used in step three but at a lower learning rate of 1×10^{-4}. To cope with the imbalanced dataset, the weighted cost function defined in 6 was used with a weight of 10 associated to the fall-class and a weight of 1 for the non-fall-class.

5 Evaluation

In this section, we evaluate the system with respect to the problem definition in Section 1.3. More precisely, three test are performed to test the real-time and detection performance of the system, followed up by a reasoning why the range, intrusiveness, and interoperability requirements are satisfied.

5.1 Throughput test

Testing the real-time requirement of the system was done by measuring the number of classifications the system can make per second. The throughput test was designed such that the neural network was fed one MHI at a time, this was done for 100000 MHI, while measuring the time it took for classifying all the data. The test was carried out on a HP EliteDesk with an Intel i7 7th gen CPU. It took 677 seconds to classify 100000 images, which resulted in 147 classifications per second. This can be related to the input signal of 25 FPS, which makes this classifier work in real-time.

5.2 Detection test

From the point of view of binary classification, the system has to decide whether a MHI represents a fall or not. Three metrics to assess the performance of the system are; *accuracy*, *sensitivity*, and *specificity*. Where sensitivity tells how well the system is at detecting actual falls, and specificity is a measurement of how well the system is at classifying non-falls correctly. Accuracy is just the overall ratio between correctly and wrongly classified data. Sensitivity and specificity are particular useful as they are not biased by imbalanced datasets, which is the case of the fall detection datasets. These metrics are defined as:

$$\begin{aligned} Sensitivity &= \frac{TP}{TP+FN} \\ Specificity &= \frac{TN}{TN+FP} \\ Accuracy &= \frac{TP+TN}{TP+TN+FP+FN} \end{aligned}$$

where TP = True-Positive, TN = True-Negative, FP = False-Positive, FN = False-Negative. Thus, for a given video, the performance of the system is evaluated for each MHI, that is, we check if a prediction is the same as the label for a given MHI. With this, the definition of the values mentioned above is:

- TP: a MHI is predicted as fall when labeled as fall.
- TN: a MHI is predicted as non-fall when labeled as non-fall.
- FP: a MHI is predicted as fall when labeled as non-fall.
- FN: a MHI is predicted as non-fall when labeled as fall.

The test was carried out on the dataset MD$_{test}$ defined in Section 4.4, and the result can bee seen in the confusion matrix in table 3. In other words, when a fall happens over n frames, the system correctly classifies 91.90% of these n frames, while 7.5% of all non-fall frames also get classified as a fall.

5.3 Generality test

One key aspect of analyzing the motion instead of the posture, is that the system have the potential to be independent of the background. To test whether the system had learned and generalized the falling motion, the system was trained on FDD as described in Section 4.5 and then evaluated using the same methodology

Table 3: Shows the number of TP, FN, TN, and FP for MD_{test}.

	Actual Fall	Actual No Fall
Pred. Fall	908	338
Pred. No Fall	79	4502

Table 4: Shows the sensitivity, specificity, and accuracy on the three different datasets.

Dataset	Sensitivity	Specificity	Accuracy
MD_{test}	91.90 %	93.00 %	92.83 %
URFD	74.90 %	88.20 %	80.20 %
MCFD	33.99 %	88.76 %	83.95 %

as in the previous section, the detection test, on two other public datasets; MCFD [48], and URFD [49]. The results can be seen in table 4.

5.4 Fall detection range

Using RGB images as input to the system, theoretically allows the system to detect a fall from any distance, as long as the resolution is high enough. The FDD used for training simulated different environments, including a normal sized room, where the distance between the camera and the person falling varied between videos. There are no information about the dimensions of these rooms, but from the videos one could estimate that areas were around the desired 6×4 meters.

5.5 Low intrusiveness

Low intrusiveness was another requirement. This was partly solved trough the usage of MHI, as only the silhouette was analyzed. And by only analyzing the motion of a person's silhouette, the system will not actually *see* you when you are still, for example when you are sleeping.

5.6 Interoperability

The neural network designed gives an output in the interval $(0, 1)$ which could be interpreted as a confidence value of how certain the system is that it has detected a fall. This could in turn be used as a one-way communication to a third party system.

6 Discussion

In this section we will compare our system's performance with other vision-based systems, as well as discussing drawbacks and further improvements of

Table 5: Comparison between our system and other approaches that used FDD for evaluation.

Proposal	Sensitivity	Specificity	Accuracy
Núñez-Marcos [31]	99.00 %	97.00 %	97.00 %
Charif [32]	98.00 %	99.69 %	99.61 %
Zerrouki and Houacine [27]	-	-	97.02 %
Ours	91.90 %	93.00 %	92.83 %

the system. We will also outline some of the problems encountered during the development phase and ending with some thoughts about deep learning and ethics related issues.

6.1 Comparison

One problem when comparing the result from Section 5 with other vision-based systems is the lack of public datasets, which have led many to create their own dataset. As of such, the comparison will be between our system and other vision-based systems that have used the FDD. Table 5 shows the results from Section 5 compared to others on the FDD.

Núñez-Marcos [31] have achieved the state-of-art classification performance. They have evaluated their system in a similar way as we did, that is, frame by frame classification. They used the more complex VGG network as a classifier, and one can compare the number of trainable parameters between VGG and MobileNet with $\alpha = 1$ in [39], where VGG has 138 million parameters compared to 0.5 million for MobileNet. Note that we used $\alpha = 0.25$ and of such have an even lower complexity. There was no mention in their work if their system method works in a real-time environment, they however used a TitanX from NVIDIA during training.

Charif [32] evaluated their system using a majority vote over a consecutive number of frames. In their case they classified 18 images individually and then made a final decision. Their result is impressive and shows that a fall detection system using simpler classification methods compared to deep neural networks are prominent.

Zerrouki and Houacine [27] evaluated their system on sub-video sequences where a sequence was either labeled fall or non-fall. They however only reported the accuracy and can be misleading in case the dataset is imbalanced, as in the case of FDD.

6.2 Throughput

The throughput test showed that the system could make 147 classification per seconds. Even though the test was performed on a high-end CPU, the classification rate was several times higher than the input signal of 25 FPS, which indicate that the system could probably be deployed on less powerful systems.

Also, since the input signal is sampled every other frame, the underlying hardware running the classifier only has to work at 12.5 FPS to keep up. This, tuning the interval as an optimization for the underlying hardware, is a strength of the current system as long as the accuracy can remain. More thoroughly testing of how the sampling rate impacts the accuracy should be done.

6.3 System drawbacks & Improvements

Even though the system have been shown to work fairly well, there are room for improvements.

6.3.1 Input

One obvious drawback of the current implementation is the usage of RGB images as input. The system can not operate properly during nighttime, as the pre-processing module cannot distinguish a moving person from the background if the input feed is all black. An alternative solution for fixing this would be to use infrared cameras or any other similar daylight-independent cameras instead.

6.3.2 Pre-processing

A problem, associated with brightness, is how the MHIs are generated. Specifically the background subtraction function defined in 19 is sensitive to light variations, e.g. if a light is turned on it would effectively mean that all pixels *moved* and the resulting MHI would be covered with only white pixels. There are other, more advanced, background subtraction algorithms that partly solves the problems with light variations, like removing shadows. Since light variations in the input stream is an intrinsic feature of RGB cameras, one could also argue that the drawback is at the input level, and that it could be solved by using e.g. a time-of-flight camera.

6.3.3 Feature extractor & Classifier

During training, it was often found that the network would overfit on the training data. This is the reason why there are dropout layers in the current implementation, as they prevent overfitting to some extent. It could be that the network is too deep, and that the solution is just to create a shallower network. However, during early stages of development, it was found that deeper networks performed better than shallower. A possible reason behind this is that a network not only has to recognize a persons' silhouette, but also has to differentiate silhouette postures. Another and probably better solution to avoid overfitting would be to have a larger dataset. Maybe combining the three public datasets into one.

It was found that the system had problems generalizing the motion of a fall across datasets, as seen in table 4. The reason behind this could be that supervised learning in general is fundamentally dependent on the data that it is training on. For example, a neural network designed to detect persons, if trained on pictures only taken from the front side of a person, would have a hard time detecting a person from the side. In other words, the classifier becomes viewpoint dependent. This would explain the poor performance on URFD and MCFD, as their camera setup is different. This is an important problem to solve in order to make the deployment of a vision-based fall detection systems more convenient, as installation places i.e. houses etc, are for the most different.

The current classifier has only been trained on FDD, and that dataset contains only one actor per video. Inherently, the system cannot detect multiple falls simultaneously. This is an interesting problem, and could be solved by image segmentation, where one first extract individual regions containing the persons, and then analyzing these regions.

Decision making is currently done at a frame-by-frame level, which could lead to isolated decisions. Charif [32] showed good result using a majority vote method to overcome this problem, and would be an interesting and fairly easy extension to the current decision making procedure.

As of now, the input layer of the feature extractor takes a 3-channeled input, while the generated MHI are only 1-channeled. This was solved by copying the grayscale images across all three channels. This is of course a waste of resources, and one should reduce the depth of the input layer to one.

6.4 Problems encountered

During the early stage of development, when choosing FDD as the dataset, it was found that the annotation of FDD was incomplete. Less than half of all the videos were annotated, and we had no access to the tools that had been used for labeling. Because it is desired to have a uniform labeling scheme for the entire dataset, the entire dataset had to be relabeled using a custom made tool.

One aspect that have affected the whole project is the time consuming task of training neural networks, in particular the convolutional layers. It could take days just to test one set of parameters, and was the reason behind using the greedy method of *early stopping* mentioned in Section 4.5.

There were several reasons behind using a pre-implemented network, i.e. MobileNet, instead of constructing one from scratch. Using a network that have been shown to converge well towards an optimal solution seemed like a reasonable choice, and in fact was also done in [31] where they used VGG. It was also found early on that transferred learning using weights learned from ImageNet gave a significant boost in classification accuracy, however training a custom network from scratch on ImageNet would be a too time consuming process relative to the time window of this project. Therefor, using a model that already had pre-trained weights was suitable, as it saved a lot of time.

The limited size of public fall detection datasets have been mentioned before to be an obstacle for deep learning solutions in the domain of automatic fall detection [14]. Fortunately transferred learning can mitigate this problem to some extent, but a larger and more diverse dataset including new areas and peoples would be desirable. There are other techniques such as data augmentation were one transform the data, e.g. rotate and scale, to create a large dataset. This was tested, but led to quicker overfitting. The reason for this could be that the transformations was not *good* enough, that is, since rotating a picture 90 degrees would mean that the person would fall from the wall, we used much less rotation and thus led to almost identical images. Maybe it would have been enough just to flip the images horizontally, such that a person falling from the left would fall from the right or vice-versa. Because data augmentation practically increases the data size and in turn training time, this method was abandoned.

6.5 Tested but not reported

The most time consuming process have been to test different set of parameters to find a good trade-off between accuracy and throughput. Parameters tested were;

- *dimension* of 128×128, 160×160, 192×192, and 224×224. Curiously, using 128×128 or 160×160 gave the best sensitivity, while 224×224 gave the worst.
- $\alpha = 0.25$, $\alpha = 0.50$, $\alpha = 0.75$, $\alpha = 1$. A similar behaviour here as with the dimensions, $\alpha = 0.25$ gave the best sensitivity and $\alpha = 1.0$ gave the worst.
- *interval* varying between 1 and 5 while keeping the *duration* constant. And the other way around, varying *duration* and keeping *interval* constant. It was found that capturing a time period around $T = 3$ seconds, calculated using equation 24, was optimal. The duration parameter was also found to be of importance. We believe that if the duration is to short the whole fall will not be captured leading to poorer performance.
- *learning rate* ranging from 0.0001 to 0.001. Lower learning rate was found to converge better, which was expected.
- *class weights* 1:1, 1:5, 1:10, 1:15, and 1:20. It was found that the sensitivity gradually improved when increasing the weight of *fall* from 1 to 10, but after that no significant improvement was done. In fact the system had a hard time recognizing a *no fall* when the class weights where 1:15 and 1:20.
- *sizes* of the layers, i.e number of neurons, in the classifier including 128, 256, 512, 1024 for both the first and second densely connected layers. Lower sizes was found to increase the accuracy, which was a win-win-situation, as it also increased throughput of the system.
- *dropout* chances for the first and second densely connected layers. This was particular difficult because it was correlated to the size of these layers.

6.6 Ethics

As this project is about monitoring actual persons in their every day life, I would like to say a couple of things regarding the integrity problem associated with this kind of system.

Deep learning inherently requires vast amount of data to perform well in term of classification accuracy, which means that for improving this system more data would be required. Especially more diverse data, i.e. not simulated data with the same actors, as is the case of this project. One solution to this would be to collect data from the individuals that are using the system. This becomes a paradox-like scenario where we design a system that should be less intrusive, but at the same time requires a lot of personal data to be implemented. Using motion history images, as we have done, limits the intrusiveness to some extent, as only gray scaled silhouettes are analyzed.

6.7 My own thoughts

I think this have been a really interesting project, that shows how powerful, but at the same time, limited deep learning is. The network became surprisingly good at recognizing falls just from analyzing silhouettes of persons, but at the same time, the network had a really hard time when exposed to a completely new dataset.

Deep learning really is a revolutionary technology that will probably affect our entire society from ground up. With the increasing trend of internet of things, it would be interesting to see in a couple of years if we can have machine learning algorithms executed on the edge of the network. That is, intelligent decision making is performed at sensor level, rather than at a data centers. In the case of vision-based fall detection, an edge node that could make intelligent choices when it detects a fall, would never have to send any images to third party instances, but just tell if a person have fallen or not. This would solve the intrusiveness problem. But would we trust one such *intelligent* system?

7 Conclusions and future work

In this work, we presented a real-time vision-based fall detection solution. The system captures temporal features using motion history images, that are used as input to a deep convolutional neural network. The network employs depth-wise separable convolutional layers to efficiently extract spatial features from the motion history images. The system performed fairly well against previously proposed systems in terms of sensitivity and specificity on the public fall detection dataset FDD. The system was also evaluated on two other public datasets to test the generality of the system. However, it was shown that the system is very dependent on the training data and could not generalize the moving motion of a fall across different datasets. The pre-processing module that generates motion history images can be manually tuned to capture longer or shorter time periods as well as the sampling frequency. These two parameters can be used for optimizing the system in terms of throughput and accuracy. Another important aspect of the pre-processing module is to limit the intrusiveness of a persons every day life, and was done by analyzing the motion of the silhouette instead of the raw input data. The preprocessing algorithm presented in this work have a couple of drawbacks, were light changes are among the severe ones. Other pre-processing methods should be tested here to reduce the influence of varying lighting conditions.

Using depthwise separable convolutions have been a great success with respect to the real-time requirement, but further experimentation using a shallower networks compared to MobileNet would be an interesting step. Especially if one could implement the neural network on a embedded system with limited computational power and still achieve the real-time requirement.

It was found that transferred learning from the ImageNet dataset improved the systems classification accuracy, even though the system classified gray-scale motion history images. For future work, one should further investigate transferred learning, in particular from other domains such as video classifications.

There are some parts still left to do: Using a real camera, as the current input module is simulating a camera using stored videos, an API to integrate with other systems, further development to make the decision making more robust i.e higher sensitivity and specificity.

References

[1] Laurence Z. Rubenstein. “Falls in older people: epidemiology, risk factors and strategies for prevention”. In: *Age and Ageing* 35.suppl_2 (2006), pp. ii37–ii41. DOI: 10.1093/ageing/afl084. eprint: /oup/backfile/content_public/journal/ageing/35/suppl_2/10.1093/ageing/afl084/2/afl084.pdf. URL: http://dx.doi.org/10.1093/ageing/afl084.

[2] Lenise A. Cummings-Vaughn and Julie K. Gammack. “Falls, Osteoporosis, and Hip Fractures”. In: *Medical Clinics of North America* 95.3 (2011). Geriatric Medicine, pp. 495–506. ISSN: 0025-7125. DOI: https://doi.org/10.1016/j.mcna.2011.03.003. URL: http://www.sciencedirect.com/science/article/pii/S0025712511000174.

[3] Martin Gronbech Jorgensen et al. “Novel use of the Nintendo Wii board as a measure of reaction time: a study of reproducibility in older and younger adults”. In: *BMC Geriatrics* 15.1 (2015), p. 80. ISSN: 1471-2318. DOI: 10.1186/s12877-015-0080-6. URL: https://doi.org/10.1186/s12877-015-0080-6.

[4] Y Lajoie and S.P Gallagher. “Predicting falls within the elderly community: comparison of postural sway, reaction time, the Berg balance scale and the Activities-specific Balance Confidence (ABC) scale for comparing fallers and non-fallers”. In: *Archives of Gerontology and Geriatrics* 38.1 (2004), pp. 11–26. ISSN: 0167-4943. DOI: https://doi.org/10.1016/S0167-4943(03)00082-7. URL: http://www.sciencedirect.com/science/article/pii/S0167494303000827.

[5] Socialstyrelsen. *Socialstyrelsen Fallolycker*. 2016. URL: http://www.socialstyrelsen.se/fallolyckor/statistikomfallolyckor (visited on 12/22/2017).

[6] World Health Organization. *WHO global report on falls prevention in older age*. 2008. URL: http://apps.who.int/iris/handle/10665/43811.

[7] Global Health Workforce Alliance and World Health Organization. *A Universal Truth: No Health Without a Workforce*. 2013. URL: http://www.who.int/workforcealliance/knowledge/resources/hrhreport2013/en/.

[8] R. Jan Gurley et al. “Persons Found in Their Homes Helpless or Dead”. In: *New England Journal of Medicine* 334.26 (1996). PMID: 8637517, pp. 1710–1716. DOI: 10.1056/NEJM199606273342606. eprint: http://dx.doi.org/10.1056/NEJM199606273342606. URL: http://dx.doi.org/10.1056/NEJM199606273342606.

[9] Nayak U. S. Wild D. and Isaacs B. “How dangerous are falls in old people at home?” In: *British Medical Journal* 282.6260 (1981), pp. 266–268.

[10] Muhammad Mubashir, Ling Shao, and Luke Seed. “A survey on fall detection: Principles and approaches”. In: *Neurocomputing* 100.Supplement C (2013). Special issue: Behaviours in video, pp. 144–152. ISSN: 0925-2312. DOI: `https://doi.org/10.1016/j.neucom.2011.09.037`. URL: `http://www.sciencedirect.com/science/article/pii/S0925231212003153`.

[11] Yann LeCun, Y Bengio, and Geoffrey Hinton. “Deep Learning”. In: 521 (May 2015), pp. 436–44.

[12] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385`.

[13] R. Girshick et al. “Region-Based Convolutional Networks for Accurate Object Detection and Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.1 (2016), pp. 142–158. ISSN: 0162-8828. DOI: `10.1109/TPAMI.2015.2437384`.

[14] Shehroz S. Khan and Jesse Hoey. “Review of fall detection techniques: A data availability perspective”. In: *Medical Engineering & Physics* 39 (2017), pp. 12–22. ISSN: 1350-4533. DOI: `https://doi.org/10.1016/j.medengphy.2016.10.014`. URL: `http://www.sciencedirect.com/science/article/pii/S1350453316302600`.

[15] Zhong Zhang, Christopher Conly, and Vassilis Athitsos. “A Survey on Vision-based Fall Detection”. In: *Proceedings of the 8th ACM International Conference on PErvasive Technologies Related to Assistive Environments*. PETRA ’15. Corfu, Greece: ACM, 2015, 46:1–46:7. ISBN: 978-1-4503-3452-5. DOI: `10.1145/2769493.2769540`. URL: `http://doi.acm.org/10.1145/2769493.2769540`.

[16] Wu F. et al. “Development of a Wearable-Sensor-Based Fall Detection System”. In: *International Journal of Telemedicine and Applications* (2015). ISSN: 1687-6415. DOI: `http://doi.org/10.1155/2015/576364`.

[17] M. Vallejo, C. V. Isaza, and J. D. López. “Artificial Neural Networks as an alternative to traditional fall detection methods”. In: *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. 2013, pp. 1648–1651. DOI: `10.1109/EMBC.2013.6609833`.

[18] A. Sengto and T. Leauhatong. “Human falling detection algorithm using back propagation neural network”. In: *The 5th 2012 Biomedical Engineering International Conference*. 2012, pp. 1–5. DOI: `10.1109/BMEiCon.2012.6465460`.

[19] O. Ojetola, E. I. Gaura, and J. Brusey. “Fall Detection with Wearable Sensors–Safe (Smart Fall Detection)”. In: *2011 Seventh International Conference on Intelligent Environments*. 2011, pp. 318–321. DOI: `10.1109/IE.2011.38`.

[20] Ali Maleki Tabar, Arezou Keshavarz, and Hamid Aghajan. "Smart Home Care Network Using Sensor Fusion and Distributed Vision-based Reasoning". In: *Proceedings of the 4th ACM International Workshop on Video Surveillance and Sensor Networks*. VSSN '06. Santa Barbara, California, USA: ACM, 2006, pp. 145–154. ISBN: 1-59593-496-0. DOI: `10.1145/1178782.1178804`. URL: `http://doi.acm.org/10.1145/1178782.1178804`.

[21] Xiaodan Zhuang et al. "Acoustic fall detection using Gaussian mixture models and GMM supervectors". In: *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2009, pp. 69–72. DOI: `10.1109/ICASSP.2009.4959522`.

[22] Y. Zigel, D. Litvak, and I. Gannot*. "A Method for Automatic Fall Detection of Elderly People Using Floor Vibrations and Sound;Proof of Concept on Human Mimicking Doll Falls". In: *IEEE Transactions on Biomedical Engineering* 56.12 (2009), pp. 2858–2867. ISSN: 0018-9294. DOI: `10.1109/TBME.2009.2030171`.

[23] B. Ugur Toreyin et al. "Falling Person Detection Using Multi-Sensor Signal Processing". In: *EURASIP Journal on Advances in Signal Processing* 2008.1 (2007), p. 149304. ISSN: 1687-6180. DOI: `10.1155/2008/149304`. URL: `https://doi.org/10.1155/2008/149304`.

[24] Ji Tao et al. "Fall Incidents Detection for Intelligent Video Surveillance". In: *2005 5th International Conference on Information Communications Signal Processing*. 2005, pp. 1590–1594. DOI: `10.1109/ICICS.2005.1689327`.

[25] S. G. Miaou, Pei-Hsu Sung, and Chia-Yuan Huang. "A Customized Human Fall Detection System Using Omni-Camera Images and Personal Information". In: *1st Transdisciplinary Conference on Distributed Diagnosis and Home Healthcare, 2006. D2H2*. 2006, pp. 39–42. DOI: `10.1109/DDHH.2006.1624792`.

[26] B. Mirmahboub et al. "Automatic Monocular System for Human Fall Detection Based on Variations in Silhouette Area". In: *IEEE Transactions on Biomedical Engineering* 60.2 (2013), pp. 427–436. ISSN: 0018-9294. DOI: `10.1109/TBME.2012.2228262`.

[27] Nabil Zerrouki and Amrane Houacine. "Combined curvelets and hidden Markov models for human fall detection". In: *Multimedia Tools and Applications* 77.5 (2018), pp. 6405–6424. ISSN: 1573-7721. DOI: `10.1007/s11042-017-4549-5`. URL: `https://doi.org/10.1007/s11042-017-4549-5`.

[28] H. Foroughi et al. "An eigenspace-based approach for human fall detection using Integrated Time Motion Image and Neural Network". In: *2008 9th International Conference on Signal Processing*. 2008, pp. 1499–1503. DOI: `10.1109/ICOSP.2008.4697417`.

[29] C. Y. Lin et al. “Vision-Based Fall Detection through Shape Features”. In: *2016 IEEE Second International Conference on Multimedia Big Data (BigMM)*. 2016, pp. 237–240. DOI: 10.1109/BigMM.2016.22.

[30] C. Rougier et al. “Monocular 3D Head Tracking to Detect Falls of Elderly People”. In: *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*. 2006, pp. 6384–6387. DOI: 10.1109/IEMBS.2006.260829.

[31] A. Núñez-Marcos, G. Azkune, and I. Arganda-Carreras. “Vision-Based Fall Detection with Convolutional Neural Networks”. In: (2017). DOI: https://doi.org/10.1155/2017/9474806.

[32] Imen Charfi et al. “Optimized spatio-temporal descriptors for real-time fall detection: Comparison of support vector machine and Adaboost-based classification”. In: 22 (Oct. 2013), pp. 041106–041106.

[33] Lei Yang, Yanyun Ren, and Wenqiang Zhang. “3D depth image analysis for indoor fall detection of elderly people”. In: *Digital Communications and Networks* 2.1 (2016), pp. 24–34. ISSN: 2352-8648. DOI: https://doi.org/10.1016/j.dcan.2015.12.001. URL: http://www.sciencedirect.com/science/article/pii/S2352864815000681.

[34] E. Auvinet et al. “Fall Detection With Multiple Cameras: An Occlusion-Resistant Method Based on 3-D Silhouette Vertical Distribution”. In: *IEEE Transactions on Information Technology in Biomedicine* 15.2 (2011), pp. 290–300. ISSN: 1089-7771. DOI: 10.1109/TITB.2010.2087385.

[35] Z. P. Bian et al. “Fall Detection Based on Body Part Tracking Using a Depth Camera”. In: *IEEE Journal of Biomedical and Health Informatics* 19.2 (2015), pp. 430–439. ISSN: 2168-2194. DOI: 10.1109/JBHI.2014.2319372.

[36] Georgios Mastorakis and Dimitrios Makris. “Fall detection system using Kinect’s infrared sensor”. In: *Journal of Real-Time Image Processing* 9.4 (2014), pp. 635–646. ISSN: 1861-8219. DOI: 10.1007/s11554-012-0246-9. URL: https://doi.org/10.1007/s11554-012-0246-9.

[37] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pp. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a.html.

[38] P. J. Werbos. “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. ISSN: 0018-9219. DOI: 10.1109/5.58337.

[39] Andrew G. Howard et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[40] ImageNet. *ImageNet*. 2018. URL: www.image-net.org (visited on 04/23/2017).

[41] Simon Baker and Iain Matthews. "Lucas-Kanade 20 Years On: A Unifying Framework". In: *International Journal of Computer Vision* 56.3 (2004), pp. 221–255. ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000011205.11775.fd. URL: https://doi.org/10.1023/B:VISI.0000011205.11775.fd.

[42] N. Noury et al. "A proposal for the classification and evaluation of fall detectors". In: *IRBM* 29.6 (2008), pp. 340–349. ISSN: 1959-0318. DOI: https://doi.org/10.1016/j.irbm.2008.08.002. URL: http://www.sciencedirect.com/science/article/pii/S1959031808001243.

[43] OpenCV. *OpenCV*. 2018. URL: https://opencv.org/ (visited on 04/23/2017).

[44] TensorFlow. *TensorFlow*. 2018. URL: https://www.tensorflow.org/ (visited on 04/23/2017).

[45] Keras. *Keras*. 2018. URL: https://keras.io/ (visited on 04/23/2017).

[46] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: 15 (June 2014), pp. 1929–1958.

[47] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[48] E. Auvinet et al. "Multiple cameras fall dataset". In: (2010).

[49] Bogdan Kwolek and Michal Kepski. "Human fall detection on embedded platform using depth maps and wireless accelerometer". In: *Computer Methods and Programs in Biomedicine* 117.3 (2014), pp. 489–501. ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2014.09.005. URL: http://www.sciencedirect.com/science/article/pii/S0169260714003447.