technologies

MDPI

Article

Deep Learning Based Fall Detection Algorithms for Embedded Systems, Smartwatches, and IoT Devices Using Accelerometers

Dimitri Kraft [1,*], Karthik Srinivasan [2] and Gerald Bieber [3,*]

[1] Faculty of Computer Science and Electrical Engineering, University of Rostock, 18059 Rostock, Germany
[2] Next Step Dynamics AB, 211 19 Malmö, Sweden; karthik@nextstepdynamics.com
[3] Fraunhofer-Institut fuer Graphische Datenverarbeitung IGD, 18057 Rostock, Germany
* Correspondence: dimitri.kraft@vcric.igd-r.fraunhofer.de (D.K.); gerald.bieber@igd-r.fraunhofer.de (G.B.)

Received: 30 October 2020; Accepted: 29 November 2020; Published: 2 December 2020

Abstract: A fall of an elderly person often leads to serious injuries or even death. Many falls occur in the home environment and remain unrecognized. Therefore, a reliable fall detection is absolutely necessary for a fast help. Wrist-worn accelerometer based fall detection systems are developed, but the accuracy and precision are not standardized, comparable, or sometimes even known. In this work, we present an overview about existing public databases with sensor based fall datasets and harmonize existing wrist-worn datasets for a broader and robust evaluation. Furthermore, we are analyzing the current possible recognition rate of fall detection using deep learning algorithms for mobile and embedded systems. The presented results and databases can be used for further research and optimizations in order to increase the recognition rate to enhance the independent life of the elderly. Furthermore, we give an outlook for a convenient application and wrist device.

Keywords: fall detection; accelerometer; datasets; deep learning; neural networks; wrist; smart bands; watches; IoT devices; edge computing

1. Introduction

The independent life of an elderly person can be changed drastically after a fall. Depending on the health condition of the elderly, almost 10 percent of the people who fall will suffer from serious injuries, or might even die directly after a fall if no intermediate help is available [1]. To prevent the severe consequences of such falls, a reliable fall detection is needed. One common approach to fall detection is using wrist worn detection systems that are measuring acceleration forces. These wrist devices are gaining more and more acceptance across the population and becoming increasingly powerful in terms of computational performance that the usage of artificial intelligence is reasonable. In general, older adults appear to be interested in using such devices although they express concerns over privacy and understanding exactly what the device is doing at specific times [2]. The evaluation of mobile fall detection systems is highly sophisticated because live data from falls of elderly people are rare. Boyle et al. tried to use real-time data with 15 adults over the course of 300 days and was only able to record four falls during that time [3]. Even simulated data are barely available and they are existing only in various characteristics. Therefore, the aim of this paper is

- to present existing public datasets of fall detection,
- to describe the harmonization process for these datasets,
- to state the current accuracy of fall detection for tiny, mobile and embedded systems using deep learning algorithms,

- to invite and motivate researchers to compete and contribute in fall detection by using the existing databases and to provide various achievements for the future life of the elderly.

2. Related Work

The detection of falls can be done through several approaches and technologies. Some approaches are infrastructure based and using external cameras [4] or floor sensors [5,6], while other approaches consist of mobile sensor analysis through body worn sensors, e.g., accelerometers, gyroscopes, and air pressure sensors. Each system records a sensor data stream that can be analyzed to recognize a fall [2]. To distinguish between no fall and fall events, various strategies were employed. Often, datasets are used to train machine learning algorithms. Other approaches do not rely on certain datasets as they consist of constructing rules to distinguish between those events. The quality of such datasets is an important constraint for the quality of the trained fall detection algorithm.

2.1. Datasets

The availability of fall data is mandatory for the development of fall recognition systems. At first, datasets were interspersed throughout the scientific community until Casilari et al. [7] provided a comprehensive overview of publicly available fall detection datasets. Since then, new datasets were published. We augmented the overview of Casilari et al. by adding recently published datasets to our overview table (see Table 1) with their corresponding characteristics (see Table 2).

Table 1. Overview of fall detection datasets recorded with a body worn sensor.

Dataset	Reference	Sensors	Sensor Location	Sampling Rate	Year
DLR	[8]	A, G, M	Waist	100 Hz	2010
MobiFall	[9]	A, G, O	Thigh	87 Hz (A) 100 Hz (G, O)	2013
TST Fall Detection	[10]	A	Waist Right Wrist	100 Hz	2014
tFall	[11]	A	Thigh Hand bag	45 Hz	2014
UR Fall Detection	[12]	A	Waist	256 Hz	2014
Simulated Falls and ADL	[13]	A, G, M, O	Head, Chest, Waist, Right Ankle, Right Wrist, Right Thigh	25 Hz	2014
Cogent Labs	[14]	A, G	Chest, Thigh	100 Hz	2015
Project Gravity	[15]	A	Thigh, Wrist*	50 Hz	2015
Graz	[16]	A, O	Waist	5 Hz	2015
MUMAFall	[7]	A, G, M	Ankle, Chest, Thigh, Waist, Wrist	100 Hz / 20 Hz*	2016
SFU Fall Detection	[17]	A	Right Ankles, Left Ankles, Right Thighs, Left Thighs, Waist, Sternum, Head	128 Hz	2016
SisFall	[18]	A_1, A_2, G	Waist	200 Hz	2017
UniMiB SHAR	[19]	A	Thigh	200 Hz	2017
SmartWatch	[20]	A	Right Wrist, Left Wrist	31.25 Hz	2018
Notch	[20]	A	Right Wrist, Left Wrist	31.25 Hz	2018
CGU-BES	[21]	A	Chest	200 Hz	2018
UP Fall Detection	[22]	A, G	Waist,Wrist,Neck, Thigh, Ankle	18 Hz	2019
MARG	[23]	A	Waist	50 Hz	2019
SmartFall	[24]	A	Right Wrist, Left Wrist	31.25 Hz	2019

Note: A_i: Accelerometer, G: Gyroscope, M: Magnetometer, O: Orientation. * Some samples are recorded with 20 Hz while other are recorded with 100 Hz.

Table 2. Dataset Characteristics.

Data-Set	# Types of ADLS/Falls	# Samples of ADL/Falls	Duration of Samples (s)		
			Min-Max	Mean	Median
DLR	15/1	1017 (961/56)	[0.27–864.33] s	18.40 s	9.46 s
MobiFall	9/4	630 (342/288)	[0.27–864.33] s	18.40 s	9.46 s
MobiAct	9/4	2526 (1879/647)	[4.89–300.01] s	22.35 s	9.85 s
TST Fall Detection	4/4	264 (132/132)	[3.84–18.34] s	8.6 s	8.02 s
tFall	not typified/8	10909 (9883/1026)	6 s	6 s	6 s
UR Fall Detection	5/4	70 (40/30)	[2.11–13.57] s	5.95 s	5.27 s
Sim Fall and ADLS	16/20	3184 (1120/1400)	[8.44–35.24] s	20.15 s	19.2 s
Cogent Labs	8/6	1968 (1520/448)	[0.53–55.73] s	13.15 s	12.79 s
Graz	8/6	1968 (1520/448)	[0.53–55.73] s	13.15 s	12.79 s
MUMAFall	8/3	531 (322/209)	15 s	15 s	15 s
SisFall	19/15	4505 (2707/1798)	[9.99–179.99] s	17.6 s	14.99 s
UniMiB SHAR	9/8	7013 (5314/1699)	1 s	1 s	1 s
SmartFall	4/4	2331 (1804/527)	N/A	N/A	N/A
SmartWatch	4/4	181 (90/91)	N/A	N/A	N/A
Notch	7/4	2563 (2456/107)	N/A	N/A	N/A
UP Fall Detection	6/5	578 (306/272)	[10–60] s	30 s	10 s
SFU Fall Detection	12/5	600 (240/360)	15 s	15 s	15 s
CGU-BES	8/4	180 (120/60)	16.5 s	16.5 s	16.5 s
MARG	5/13	432 (120/312)	[1.3–11.5] s	5.3 s	5.58 s

2.2. Position of Sensors

More than half (10 out of 19) of the datasets described in Table 1 contain waist-worn acceleration data, while only seven datasets contain wrist-worn acceleration data. We do not focus on thigh data sets due to the limited practicability in daily life. We suspect that a wrist or waist worn acceleration sensor is much easier to use and provide a higher wear comfort. Pannurat et al. pointed out that most fall detection systems using sensors mounted to the torso [25]. Krupizer et al. recently provided an overview of the most common positions for wearable sensors in fall detection systems [26]. They confirmed that the waist is a popular position for most automatic fall detection systems.

2.2.1. Wrist vs. Waist

Liao al. showed in [27] that their approach performed best on the human activity recognition task when the data are obtained by a waist worn accelerometer. In this paper, we focus on wrist-worn data, as we strongly believe that sensor solutions for the wrist are more convenient for the end user and may be adopted earlier. Furthermore, a fall detection algorithm may be easily integrated into currently available smartwatch or fitness tracker products.

2.3. Fall Detection Algorithms

Several fall detection algorithms were proposed in the last decade. These algorithms ranging from simple threshold based approaches [28,29], over handcrafted feature based machine learning algorithms [30,31] and finally to deep learning based automatic feature extraction neural networks [24,32,33]. Most of the aforementioned contributed datasets come with a detailed approach for detecting falls. Some of the relevant work may be found in Table 3. Note that the performance reported may not express the true capability of these approaches, as the validation methods of these algorithms are mixing users in their train and validation sets, resulting in a fragile evaluation process.

Table 3. Overview about published fall detection algorithms.

Source	Dataset	Falls/ ADLS [n/n]	Features	Classifier	Validation Method	Performance
[31]	MobiFall	342/ 288	Time domain	Naive Bayes, LS, ANN, SVM	75–25 split, mixed user	87.5% accuracy
[34]	MobiFall	346/ 11080	Time and frequency domain	MLP, SVM, KNN	346 test samples	97.29% sensitivity
[35]	MUMA	198/ 400	Biological risk factor	KNN	80–20 split, mixed user	84.1% accuracy
[36]	MUMA	198/ 400	Raw time series window	LSTM	80–20 split, mixed user	92% accuracy
[20]	SmartFall, Notch, Farseeing	221/ 30000	Time domain	SVM	66–33 split, mixed user, LOO CV	85% accuracy, 93% accuracy, 99% accuracy
[32]	SmartFall, Notch	221/ 30000	Raw time series window	CNN	80–20 split, mixed user	99.7% accuracy
[37]	SisFall	1798/ 2707	Time domain	SVM	10 Fold CV, mixed user	99.75% accuracy
[38]	SisFall	Not specified	Raw time series window	LSTM	Not specified	97.16% accuracy
[39]	SisFall	1575 Falls/ not specified	Time domain	SVM with RBF	5 Fold CV, mixed user	98% accuracy
[33]	SisFall	1780 Falls/ 76926 ADL	Raw time series window	CNN and various other	2 Fold CV, separated user	99.94% accuracy

2.4. 1D Convolutional Neural Networks

Convolutional neural networks (CNN) were first introduced by Krizhevsky et al. [40] and gained attention by winning the ImageNet challenge [41] with a large margin in 2012. Since then, new CNN Architectures, e.g., ResNet [42] were introduced and enhanced the performance of CNN significantly. CNN's were mostly used for the image domain. In 2016, Wang et al. [43] introduced a fully convolutional neural network (FCN) architecture to classify time series data. They validated their approach on 44 datasets from the UCR/UEA archive. CNN for time series data processing achieved state of the art performances in various domains e.g., ECG Classification, Sound Classification, and Natural Language Processing. The name giving convolution can be seen as applying and sliding a one-dimensional filter over the time series. Unlike images, the filters cover only one dimension (time) opposed to two dimensions (width and height). A convolutional neural network consists of various filters, ranging from moving average filters to more complex filters. Those filters are learned through the backpropagation algorithm. By passing a univariate time series through a convolutional neural network, multiple convolutions with different filters are applied. This may be seen as using a filter bank to extract useful features, removing outliers or general filtering. To surpass the linear nature of a convolution operation, a nonlinear function is applied after a convolution to introduce nonlinearity and therefore ensure a nonlinear transformation of the time series data. An illustration of a

one-dimensional convolutional neural network is denoted in Figure 1. Mathematically, a convolutional neural network apply the following equation for each timestamp t of the time series data $\mathbf{X}$:

$$C_t = f(\mathbf{W} * \mathbf{X}_{t-\frac{k}{2}:t+\frac{k}{2}} + \mathbf{b})$$

were $\mathbf{W}, k, t, \mathbf{b}$ equals weights of the kernel, length of the kernel, timestamp, and bias, respectively. After applying n convolutions on the input accelerometer data with length l, we settle with n channels, where each channel represents a new filtered time series. These n channels with shape $n \times l$ are then convolved with m different filters with shape $n \times m \times k$, where each m_{i-th} filter is slid across all n channels resulting in m additional time series, where each m_i channel is a sum of the convolutions of m_{i-th} filter across all n channels.

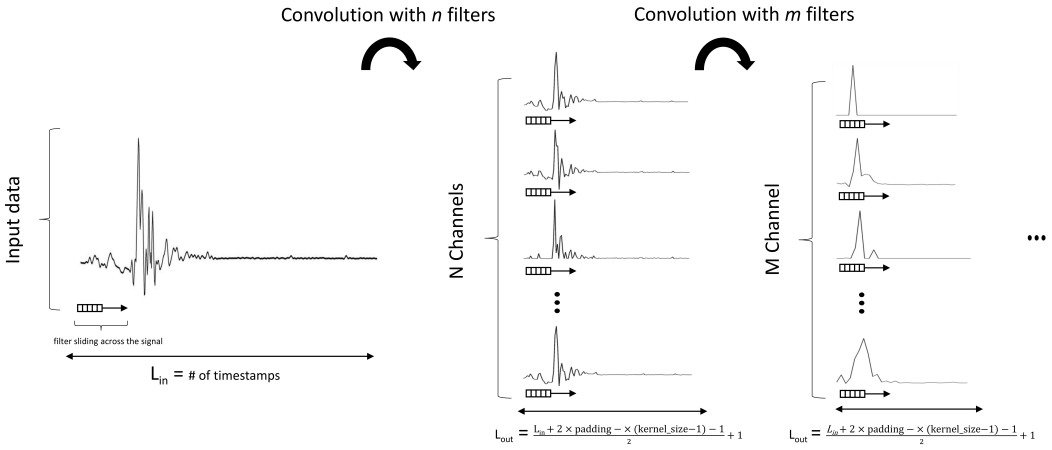


Figure 1. Illustration of a one-dimensional convolutional neural network.

2.5. Recurrent Neural Networks

Recurrent Neural Networks (RNN) are designed to work with time series data. Their recurrent mechanisms ensure that features in the time series data are learned jointly over time. To train a recurrent neural network, the back-propagation through time algorithm is used. Recurrent neural networks emerged in 1986 [44] and underlie various advances. One major drawback of traditional recurrent neural network is the vanishing or exploding gradient problem, which occur when the gradient of each cell is multiplied over time and gets larger or smaller with each multiplication. Training a recurrent neural network was a hard task to complete. Long Short-Term Memory (LSTM) [45] neural networks eliminated the problem of vanishing and exploding gradients by introducing forget gates. This mechanism enabled the effective training of recurrent neural network models.

2.5.1. Long Short-Term Memory

One major difference to traditional RNN models is essentially what a Long Short-Term Memory (LSTM) learns. It can learn to keep only relevant information to make predictions and forget non relevant data. This renders the LSTM easier to train and perform better on classification or regression tasks. This process is made through a sigmoid-like function, which outputs a value between zero (discard) and one (keep).

3. Experimental Setup

3.1. Data Preparation

In order to combine datasets containing wrist-worn accelerometer data, a series of steps is employed to utilize them for training. Due to different sampling rates and sensors, several datasets need special handling. Some datasets include a label for each timestamp, while others provide a sample

as a text file with the label being part of the files name. For each recording (containing either falls or activities of daily living) in a data set, we segment the recording into 10 s of non overlapping windows and down-or upsample each window to 50 Hz. The range and quantization of the raw data remain unchanged. The variety in coupling between human body and sensor, precisely the attachment of the sensor at the wrist, and also different sensor weights are not considered. Each fall type is labeled as *fall* and every other activity is labeled as *not fall*.

3.1.1. Down and Upsampling Technique

In order to perform robust down-and upsampling, we use a polyphase filtering approach provided by the Python *SciPy 1.2.3* package [46]. The acceleration signal is upsampled by a factor up, before a zero-phase low-pass FIR filter is applied, and then again downsampled by the factor $down$. The resulting sample rate is $up/down$ times the original sample rate. We pad the signal boundaries by fitting a line at the start and end of the signal to avoid overshooting at the boundaries of the signal. We are aware that a down- and upsampling of data are somehow affecting the data quality, but this limitation is necessary for a comparable dataset.

3.1.2. SmartFall, Smartwatch, and Notch

For the SmartFall, Smartwatch, and Notch datasets, the following steps are applied:

1. Cluster timestamps containing a fall to create separated 10 s segments.
2. Each segment is upsampled from 310 (31 Hz) to 500 timestamps (50 Hz)

After applying this strategy, SmartFall, Smartwatch, and Notch consist of solely fall samples. We do not use samples where a fall sample is mixed with an activity of daily living labeled as non fall. This would result in an ambiguity, as we already use samples containing fall and activities of daily living labeled as fall. To enrich the Notch, Smartwatch, and SmartFall datasets with additional activities of daily living, we added 500 random 10 s segments from the RealWorld Human Activity Recognition [47] dataset to the SmartFall, Smartwatch, and Notch datasets.

3.1.3. MUMA, UP Fall, and Sim Fall

The MUMA dataset consists of separated files containing either fall or activities of daily living, where each file consists of 300 timestamps. We cropped a 200 timestamp window (10 s) out of the time series and upsampled it to 500 timestamps. To harmonize the UP Fall dataset, we applied a slightly different strategy, as this dataset does not offer a file for each fall or activity of daily living. A peak detection algorithm provided by the Python *SciPy 1.2.3* package [46] is used to find the segments containing a fall, where we set the prominence to a 95% quantile of the Signal Magnitude Vector values. We crop a 180 timestamp window, centered around the detected peaks, and upsampled this window to 500 timestamps. The Sim Fall dataset consists of separated files containing either data of fall or activities of daily living. We crop a 250 timestamp window, centered around the center corresponding time series.

3.2. Harmonized Dataset

The resulting harmonized dataset consists of 1716 Falls and 3567 activities of daily living, where each sample consists of 500 timestamps associated with x, y, and z-axis acceleration values. We settle with a total tensor shape of $5283 \times 500 \times 3$ for our experiments. Table 4 shows the number of falls and activities of daily living for each dataset. Note that the number of activities of daily living is significantly larger than the number of falls. To compensate such unbalanced dataset, we induce a weight to the cross entropy loss function calculated by the ratio of falls and activities of daily living. Note that we did not integrate the TST Fall Detection dataset, as it is not available anymore and it is not clear which sensor is used on the wrist and which sensor is used on the waist.

Table 4. Harmonized dataset.

Dataset	Falls	ADL	Subjects	Age of Subjects
MUMA	189	428	17	18–55
Notch	198	500	7	20–35
Sim Fall	286	258	17	19–27
Smartwatch	271	500	7	21–55
SmartFall	527	500	14	21–60
UP Fall	245	1381	17	18–24
Total	1716	3567	79	18–60

3.3. Problem Formulation

The fall detection or fall classification problem may be formulated as a problem of time series classification. A fall may represent a univariate time series $X = [x_1, x_2, \ldots, x_t]$ with an ordered set of real values or a multivariate time series $M = [X^1, X^2, \ldots, X^n]$, where M consists of different univariate time series X. A dataset may be formed by pairing the time series data X with a label Y to a tuple (X^i, Y^i). This label is a numerical representation of a class label either *fall* with $Y^i = 1$ or *non fall* with $Y^i = 0$. The task is to train a classifier on a dataset consisting of multiple time series in order to map the space of possible inputs to a probability distribution over the class variable values, referred to as labels.

3.4. Preprocessing

Contrary to other approaches e.g., [32], we do not think that a scaling or standardization preprocessing technique should be performed on the input data, as the magnitude information is a crucial aspect to distinguish between falls and activities of daily living. To evaluate the impact of preprocessing on the performance, we apply a min-max scaling of the training and testing dataset.

Signal Magnitude Vector (SMV)

To reduce the computational load, we solely use the acceleration data, precisely the Signal Magnitude Vector. The Signal Magnitude Vector maps the three-dimensional acceleration vector $X \in \mathbb{R}^3$ to a one-dimensional acceleration vector with no orientation information. To compute the Signal Magnitude Vector, the following equation is applied for each time series **X**:

$$SMV = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

The Signal Magnitude Vector transformation greatly reduces the input size for neural networks and thus reducing the required memory and computational load of an IoT device. Note that running deep learning models on IoT devices requires a trade-off between classification performance and computational complexity.

3.5. Data Augmentation

Data Augmentation can be viewed as an injection of prior knowledge about the invariant properties of the data against certain transformations. Augmented data can cover unexplored input space, prevent overfitting, and improve the generalization ability of a deep learning model [48]. Um et al. pointed out that time series augmentation significantly improves the classification accuracy of neural network architectures from 77.54% to 86.88 % in the domain of Parkinson's Disease classification in Alzheimer patients [49]. In the domain of fall detection, augmentation showed already an enormous performance boost in other works [32]. Before augmentation, one requires knowledge about the semantic characteristics of the input data. Scaling of the input data may induce bias, as falls typically have a defined range of acceleration values. To evaluate the effect of data augmentation, we apply a combination of the following transformations dynamically during training:

- Shifting with a probability of $p = 0.75$ and random shift value of $[-150 \dots 150]$ in samples along the time axis (only used during signal magnitude vector training)
- Rotation with a probability of $p = 0.75$ around x, y, and z-acceleration axes with a random angle between $[-180° \dots 180°]$ (only used during three-axis training)

We suspect that shifting in the time dimension does not contribute to the performance of a one-dimensional CNN, as the learned filters are translation-invariant by definition. However, by shifting (rolling) the vector in the time dimension, elements that roll beyond the last position of the vector are re-introduced at the first position. This induces some additional variance to data and may affect the performance of the neural networks.

3.6. Model Architectures

To compare our IoT neural network to other neural network architectures, we choose a number of different architecture types. We utilized the PyTorch 1.2.0 Framework [50] for training and testing our approach. All models were trained 250 epochs each with the Adam Optimizer [51] with initial learning rate of 0.001, batch size of 32, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and no weight decay using a weighted Cross Entropy loss function. As our convolutional neural networks use ReLU (Rectifier Linear Unit) activation functions, all convolutional layers were initialized via He initialization [52]. The ReLU is a nonlinear activation function which is defined as $\sigma(x) = max(0, \mathbf{x})$.

3.6.1. Classic 1D CNN

CNN-3B3Conv [32] consists of three-layer blocks. The first block consists of three convolutional layers and one maxpooling layer. Each of the convolutional layer consists of 64 kernels with a size of 4 and a maxpooling size of 2. The second block also consists of three convolutional layers and one maxpooling layer, this time with a kernel size of 3 while the maxpooling size remains unchanged. The third block consists of three fully-connected layers with 64 neurons, 32 neurons, and two neurons. To deal with the different input size (500 opposed to 32), we changed the pooling size of the maxpooling parameter to 15.

3.6.2. 1D ResNet

We use a small-scale 1D ResNet model consisting of two basic residual connection blocks. We settle with a convolutional layer followed by two blocks with two convolutional layers and a shortcut connection each block. The kernel size is set to 8 for the first convolutional layer and is changed to four for all consecutive convolutional layers. The classification head consists of a fully connected layer mapping 64 neurons to two neurons.

3.6.3. LSTM

The LSTM Classifier consists of a single unidirectional LSTM layer with 128 hidden neurons followed by a dropout layer with a dropout probability of $p = 0.25$. A fully connected layer is used to map the 128 hidden neurons to two neurons for classification.

3.7. Proposed Model Architecture

Our baseline CNN for fall classification includes several changes compared to Santos et al. [32]. As Santos et al. pointed out in their evaluation, a deeper network does not necessary increase the performance of classification. Our approach to fall detection consists of greatly reducing the number of layers and filters of the convolutional neural network. Furthermore, we replaced the computational expensive max-pooling layer with efficient strided convolutional layer [53]. By removing the flattening layer right before the fully connected layer and replacing them with a global average pooling (GAP) layer followed by a 1×1 convolutional layer, we further reduced the number of trainable parameter significantly. To effectively deploy our model to IoT devices, we additionally use quantization

which reduce the footprint of the model drastically. After employing an extensive gridsearch, we settle with the following structure:

$$C(8) - ReLU - C(16) - ReLU - C(32) - ReLU - GAP - C(2)$$

Each convolutional layer $C(N)$, with N channels, uses a stride of 4 and a kernel size of 8 with no padding, except for the last convolutional layer which uses stride and kernel size of 1 with no padding.

3.7.1. Maxpooling vs. Strided Convolutions

Strided Convolutions reduce, similar to maxpooling, the size of the resulting feature map and thus reduce the computational complexity of the processed data. We replaced the maxpooling layers with convolutional layers with stride 4 to eliminate computational expensive calculation of the maximum value and additionally reduce the number of dot products by a factor of 4. A strided convolution calculates the dot product for every $n - th$ timestamp by skipping n timestamps with each calculation resulting in a downsampling by a factor of n.

3.7.2. Global Average Pooling Layer

Global Average Pooling (GAP) allows the Convolutional Neural Network to process time series with different lengths and reduces the number of learned parameters significantly [54]. Furthermore, a GAP layer enables us to use Class Activation Maps for visualization. Global Average Pooling calculates the mean for each channel (activation map) resulting in a vector of averaged activation values of each channel.

3.7.3. Learnable Parameters

The amount of learnable parameters affect the training time, computational complexity, and footprint on the device. By replacing the flattening layer with GAP and Maxpooling layer with strided convolutions, we reduced the number of learnable parameters to around 5000 opposed to 72.512 in [32] and further reduced the required memory for storing the activation maps (channels) significantly. Our model is approx. 14 times smaller than [32] and can be easily used on small scale IoT devices as it only uses approx. 20 KB of RAM for storing the weights and approx. 10 KB for storing the activation values for each channel.

3.7.4. Quantization

Quantization reduces the footprint of a model greatly, by converting the weights of the neural network from four byte floats to one byte unsigned integer. Wu et al. [55] showed that a quantization occurs with a minor loss in accuracy while reducing the computation time greatly. We use a Quantization method provided by the PyTorch 1.4.0 Framework, in particular the min and max values to compute the necessary quantization parameters. We do not evaluate different quantization strategies to conserve space.

4. Evaluation

4.1. Evaluation Method

In order to employ a robust evaluation process, our approach consists of applying Leave One Out (LOO) cross validation on whole datasets, e.g., using the MUMA Dataset for testing and the remaining five wrist-worn datasets for training. We reset the weights and the optimizer parameters with each fold. This procedure is repeated five times until each dataset has been used as a testing dataset. To discard potential weight initialization bias, we repeated the LOO cross validation five times and averaged the results.

4.1.1. Evaluation Metric

We assess the classification performance of our deep learning model by using the weighted F_1 score, precision, and recall. Note that, in the binary classification task, recall of the positive class (fall) is denoted as sensitivity and recall of the negative class (not fall) as specificity.

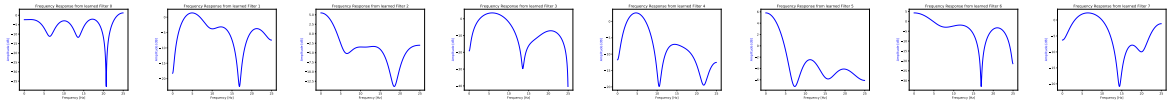
$$precision = \frac{TP}{TP + FP} \qquad recall = \frac{TP}{TP + FN} \qquad F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

4.2. Effect of Augmentation

In line with other work, we observed performance improvements while using data augmentation techniques for all tested models. The results achieved with data augmentation are denoted in Tables 5 and 6. While we observe that our three-axis model increased in terms of performance, our single-axis IoT-CNN achieved only little or negligible improvement. Larger models, e.g., CNN-3B3Conv, improved in performance while using dynamic data augmentation during three-axis and single-axis training training.

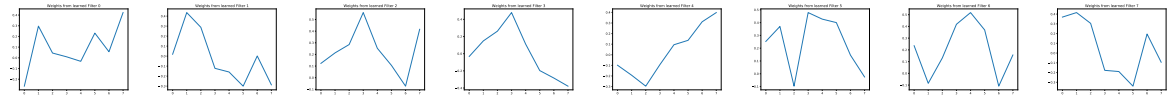
4.3. Learned Filters

To visualize the frequency response of the learned filters in the first convolutional layer, we use the *freqz* method, provided by the Python package *scipy* [46]. A lot of filters resemble low pass, bandstop, high pass, and further combination of low and highpass filters. The learned kernel weights with their respective frequency response are denoted in Figures 2 and 3. Note that most learned filters activate on strong edge shaped structures.



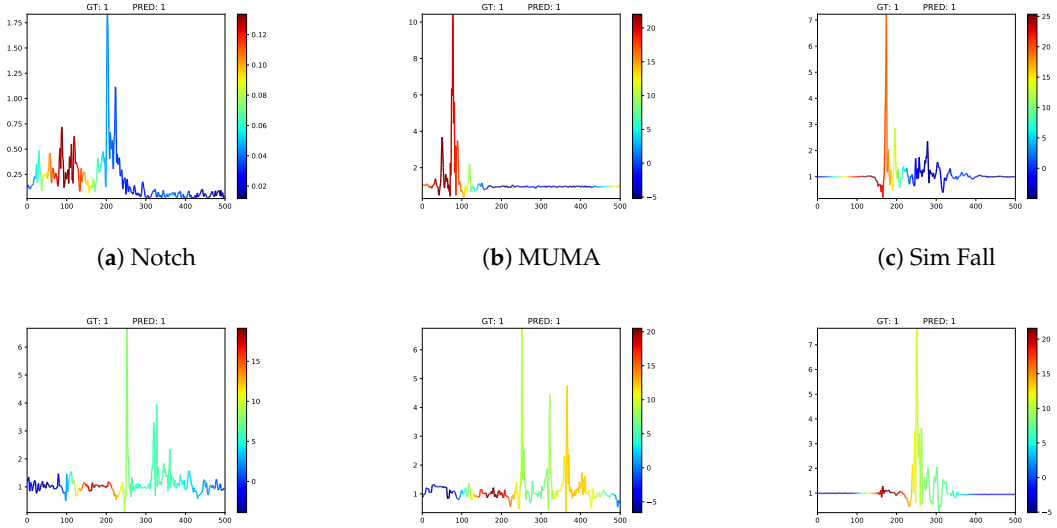
(**a**) (**b**) (**c**) (**d**) (**e**) (**f**) (**g**) (**h**)

Figure 2. Frequency responses of the first convolutional layer.



(**a**) (**b**) (**c**) (**d**) (**e**) (**f**) (**g**) (**h**)

Figure 3. Learned kernel weights of the first convolutional layer.



(**a**) Notch (**b**) MUMA (**c**) Sim Fall

(**d**) SmartWatch (**e**) SmartFall (**f**) UP Fall

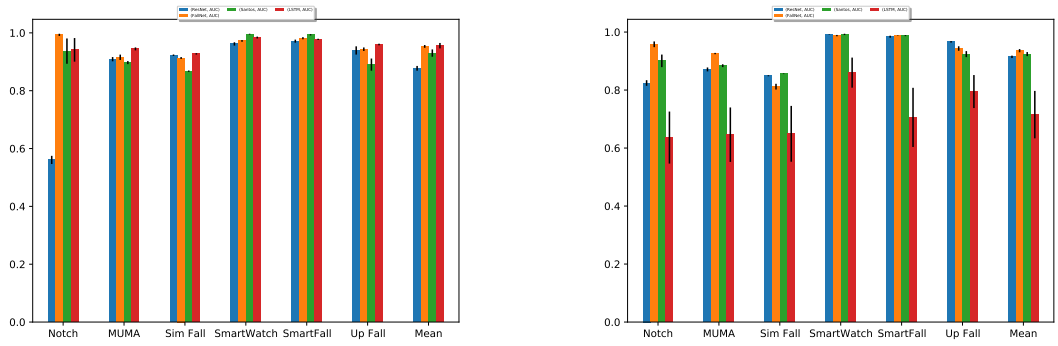
Figure 4. Class Activation Map examples for fall data sampled from each dataset.

4.4. Class Activation Maps

Class Activation Maps (CAM) for time series data, introduced by Zhou et al. in [56], indicate which regions in a time series contribute to the decision-making process of a neural network. A Class Activation Map (CAM) for a class c may be computed by the following equation:

$$CAM_c(t) = \sum_m w_m^c A_m(t)$$

where $A_m(t)$ is the univariate time series for the variable $m \in [1, M]$, which is in fact the result of applying the $m - th$ filter and w_m^c is the weight between the $m - th$ filter (last convolutional layer) and the output neuron of class label c [54]. Examples of class activation maps with a sample for each dataset are depicted in Figure 4. In most cases, the impact region (region with the highest acceleration magnitude) and free fall phase (region right before the impact) are the most contributing parts to the decision-making of our convolutional neural network.



(**a**) AUC 1-Axes with Data Augmentation (**b**) AUC 3-Axes with Data Augmentation

Figure 5. AUC Results with data augmentation for single-axis (left) and three-axis (right). Bar charts showing the AUC performance for each dataset and network.

4.5. Results

Referring to Tables 5 and 6, we demonstrated that our proposed algorithm performs very accurate by an $F_1 = 0.96$ without, and $F_1 = 0.97$ with data augmentation (both on SMV). The results in Figure 5 suggest that a larger number of parameters does not lead to increased performance in terms of AUC. Furthermore, the three-axis models do not perform better than their single-axis equivalent. The LSTM neural network especially shows increasing performance when using the Signal Magnitude Vector as input. In particular, the three-axis version of our IoT-CNN does perform poorly on the MUMA dataset and SimFall dataset if no augmentation is applied. We suspect that this may be due to the relative small number of parameters compared to other models. All CNNs perform better on the SmartWatch, SmartFall, Notch, and UP Fall datasets. This indicates that certain datasets are easier to handle than other and may contain less variety. Note that the SmartWatch, SmartFall, and Notch dataset was published by the same researchers. We further augmented the SmartFall, Smartwatch, and Notch dataset with samples of a different dataset. This is done due to the sequence length of a fall in these datasets, as they only contain fall events shorter than 2 s. Regarding the LSTM-based neural network, the single-axis version shows a comparable performance with respect to their CNN counterparts. Quantization, on the other hand, achieved, as expected, worse results. In line with other work, quantization reduces the classification performance (see Table 7). While the performance on most datasets remains comparable to the results without quantization, the performance on the Notch dataset decreases by a large amount. This may be due to the gravitational offset from the used sensor.

Table 5. Results (F1 Score) without Data Augmentation.

Dataset	**Our**				**LSTM**				**Santos**				**ResNet**			
	SMV		**3 Axis**		**SMV**		**3 Axis**		**SMV**		**3 Axis**		**SMV**		**3 Axis**	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Notch	0.98	0.01	0.97	0.03	1.00	0.00	0.95	0.01	0.92	0.04	0.94	0.01	0.77	0.14	0.91	0.05
MUMA	0.92	0.00	0.80	0.04	0.92	0.01	0.75	0.04	0.91	0.01	0.91	0.01	0.90	0.01	0.89	0.02
Sim Fall	0.89	0.00	0.83	0.04	0.88	0.02	0.80	0.12	0.88	0.01	0.89	0.00	0.90	0.01	0.89	0.01
SmartWatch	0.99	0.00	1.00	0.00	1.00	0.00	0.95	0.04	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00
SmartFall	0.99	0.00	1.00	0.00	0.99	0.00	0.92	0.02	1.00	0.00	1.00	0.00	0.99	0.00	1.00	0.00
UP Fall	0.98	0.00	0.90	0.02	0.98	0.00	0.78	0.07	0.99	0.00	0.97	0.00	0.99	0.00	0.96	0.02
Average	**0.96**	0.00	0.91	0.02	**0.96**	0.01	0.86	0.05	0.95	0.01	0.95	0.00	0.93	0.03	0.94	0.02

Table 6. Results (F1 Score) with Data Augmentation.

Dataset	**SMV**		**3 Axis**		Our **LSTM** **SMV**		**3 Axis**		**Santos** **SMV**		**3 Axis**		**ResNet** **SMV**		**3 Axis**	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Notch	0.97	0.01	0.95	0.03	1.00	0.00	0.93	0.04	1.00	0.00	0.96	0.01	0.97	0.03	0.93	0.01
MUMA	0.93	0.01	0.94	0.01	0.91	0.01	0.84	0.06	0.92	0.00	0.92	0.00	0.92	0.01	0.94	0.01
Sim Fall	0.93	0.00	0.90	0.01	0.93	0.01	0.81	0.10	0.89	0.00	0.90	0.00	0.94	0.00	0.90	0.01
SmartWatch	0.99	0.00	1.00	0.00	0.99	0.00	0.95	0.03	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00
SmartFall	0.99	0.00	1.00	0.00	0.98	0.00	0.92	0.03	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00
UP Fall	0.98	0.00	0.99	0.00	0.97	0.00	0.91	0.05	0.98	0.00	0.99	0.00	0.98	0.00	0.99	0.00
Average	**0.97**	0.00	**0.96**	0.01	0.96	0.01	0.89	0.05	0.97	0.00	0.96	0.00	**0.97**	0.01	0.96	0.01

Table 7. Results (Precision, Recall, and Weighted F1 Score) using our CNN with Data Augmentation and Weight Quantization.

Dataset	Our SMV					
	Precision		Recall		F1	
	μ	σ	μ	σ	μ	σ
Notch	0.854	0.092	0.853	0.078	0.826	0.110
MUMA	0.926	0.017	0.910	0.024	0.912	0.023
Sim Fall	0.908	0.016	0.902	0.023	0.902	0.023
SmartWatch	0.982	0.004	0.982	0.004	0.982	0.004
SmartFall	0.976	0.009	0.975	0.010	0.975	0.010
UP Fall	0.952	0.013	0.940	0.035	0.942	0.029
Average	0.933	0.025	0.927	0.029	0.923	0.033

5. Discussion

Historically, fall detection algorithms were discrete and engineers developed a mobile system especially for this purpose. Today, neural networks can be exploited on embedded systems, so a flexible structure can be used. Because of the connectivity of IoT devices, a broad database is or will be available in future. Therefore, the realization of a reliable fall detection becomes available. The performance consideration of fall detection is ambiguous as far as varying datasets are used. Therefore, we identified available datasets and evaluated the capabilities of a small-scale neural network. Large networks that can run on servers or high end machines might outperform small-scale neural networks, but these can be integrated on edge or small devices and are more relevant for the real life scenarios, as the computation is done on the device. This further ensures (a) that sensible data are processed on the device and no network connection is needed, besides the alarming mechanism, and (b) that privacy is protected. Regarding the existing fall datasets, we assume that resampling, range, resolution, and sensor type (internal filtering) only have a minor effect. However, this has to be confirmed by further research.

6. Outlook and Application

Typical deep neural network models require high computational power for training a large scale neural network. However, the inference is much less energy consuming. As we could show in our research, even small models provide a reasonable recognition rate of 97%. The representation of weights can be shrunk from 32-bit floats to unsigned 8-bit integers without a remarkable loss of accuracy and reduction in the memory footprint on the device. By levering compact architecture with quantized weights, we are able to use our efficient neural network on embedded devices. This research forms the center of the lightweight wrist-based fall detection device, called UMA, designed by the company Next Step Dynamics, which has a long battery life as the fall detection is processed in a dual recognition pipe—an acceleration sensor monitors all movements and a small size trigger algorithm works as a gatekeeper and detects very roughly possible falls. Due to the dual recognition pipe, a possible requirement of a fast processing speed of the neural net is not necessary. Furthermore, the trigger algorithm ensures a continuous processing and provides easy to handle segmented data to the second inference step. The device (based on a Nordic nRF9160) is shown in Figure 6. We implemented our proposed neural network on the UMA fall detection device and validated its capabilities in a real-life scenario in future work.

Figure 6. UMA Device with an integrated neural net for fall detection.

Elderly persons are usually very calm; even a normal adult is performing moderate to vigorous leisure time physical activity only less than one hour a day [57]. The simple gatekeeper algorithm clears inactive periods, so the deep learning algorithm is used only for active periods.

7. Conclusions

In this paper, we presented 19 datasets with fall raw data, assessed at the wrist (7), waist (10), and/or other positions (2) with accelerometers. We illustrated that our optimized neural network could be applied on an embedded system like IoT devices, smart watches, or activity trackers. In former research, the consideration of the accuracy of fall detection was performed within a focus group and its dataset. Because of the identification of multiple datasets, a broader evaluation was feasible. We could show that neural networks are performing well on our harmonized dataset. Furthermore, the increasing calculation power of mobile devices enables the usage of deep learning algorithm for fall detection on wrist based embedded systems. We demonstrated that small scale convolutional neural networks achieve a reasonable accuracy of 97% on our harmonized fall detection data set. While applying quantization, our neural network performs less accurately, which may be addressed in future work. We suspect that calibration of the neural network based on the activity of the user may enhance the performance significantly by lowering the false positive rate. For future work, we see that the sensitivity and specificity of fall detection is highly relevant in the everyday usage. A waterproof, wrist based sensing device can be worn 24/7 and should indicate no false detection. This is a high demand and requires the knowledge about the general condition about the user. Very active people are moving differently compared to passive and calm people. This requirement of a low false positive rate can be achieved by individualized algorithms or calibration. We assume that even energy efficient mobile wrist devices allow a reliable fall detection system to assist the elderly in everyday life.

Author Contributions: Conceptualization, D.K. and G.B.; methodology, G.B. and K.S.; software, D.K.; data curation, D.K.; writing—original draft preparation, D.K., K.S. and G.B.; writing—review and editing, D.K., K.S. and G.B.; supervision, G.B. All authors have read and agreed to the published version of the manuscript.

Funding: The research was funded by Next Step Dynamics AB..

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Stevens, J.A.; Corso, P.S.; Finkelstein, E.A.; Miller, T.R. The costs of fatal and non-fatal falls among older adults. *Inj. Prev.* **2006**, *12*, 290–295. [CrossRef]
2. Chaudhuri, S.; Thompson, H.; Demiris, G. Fall detection devices and their use with older adults: A systematic review. *J. Geriatr. Phys. Ther.* **2014**, *37*, 178–196. [CrossRef]

3. Boyle, J.; Karunanithi, M. Simulated fall detection via accelerometers. In Proceedings of the 2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Vancouver, BC, Canada, 20–25 August 2008; pp. 1274–1277.
4. Nunez-Marcos, A.; Azkune, G.; Arganda-Carreras, I. Vision-based fall detection with convolutional neural networks. *Wirel. Commun. Mob. Comput.* **2017**, *2017*, 9474806. [CrossRef]
5. Alwan, M.; Rajendran, P.J.; Kell, S.; Mack, D.; Dalal, S.; Wolfe, M.; Felder, R. A smart and passive floor-vibration based fall detector for elderly. In Proceedings of the 2006 2nd International Conference on Information & Communication Technologies, Damascus, Syria, 24–28 April 2006; Volume 1, pp. 1003–1007.
6. Daher, M.; Diab, A.; El Najjar, M.E.B.; Khalil, M.A.; Charpillet, F. Elder tracking and fall detection system using smart tiles. *IEEE Sens. J.* **2016**, *17*, 469–479. [CrossRef]
7. Casilari, E.; Santoyo-Ramón, J.A.; Cano-García, J.M. UMAFall: A Multisensor Dataset for the Research on Automatic Fall Detection. *Procedia Comput. Sci.* **2017**, *110*, 32–39. [CrossRef]
8. Frank, K.; Vera Nadales, M.J.; Robertson, P.; Pfeifer, T. Bayesian recognition of motion related activities with inertial sensors. In Proceedings of the 12th ACM International Conference Adjunct Papers on Ubiquitous Computing—Adjunct, Copenhagen, Denmark, 26–29 September 2010; Bardram, J.E., Ed.; ACM: New York, NY, USA, 2010; p. 445. [CrossRef]
9. Vavoulas, G.; Pediaditis, M.; Spanakis, E.G.; Tsiknakis, M. The MobiFall dataset: An initial evaluation of fall detection algorithms using smartphones. In Proceedings of the IEEE 13th International Conference on Bioinformatics and Bioengineering (BIBE), Chania, Greece, 10–13 November 2013; pp. 1–4. [CrossRef]
10. Gasparrini, S.; Cippitelli, E.; Gambi, E.; Spinsante, S.; Wåhslén, J.; Orhan, I.; Lindh, T. Proposal and Experimental Evaluation of Fall Detection Solution Based on Wearable and Depth Data Fusion. In *ICT Innovations 2015*; Advances in Intelligent Systems and Computing; Loshkovska, S.; Koceski, S., Eds.; Springer: Cham, Switzerland, 2016; Volume 399, pp. 99–108. [CrossRef]
11. Medrano, C.; Igual, R.; Plaza, I.; Castro, M. Detecting falls as novelties in acceleration patterns acquired with smartphones. *PLoS ONE* **2014**, *9*, e94811. [CrossRef]
12. Kwolek, B.; Kepski, M. Human fall detection on embedded platform using depth maps and wireless accelerometer. *Comput. Methods Programs Biomed.* **2014**, *117*, 489–501. [CrossRef]
13. Özdemir, A.T.; Barshan, B. Detecting falls with wearable sensors using machine learning techniques. *Sensors* **2014**, *14*, 10691–10708. [CrossRef]
14. Ojetola, O.; Gaura, E.; Brusey, J. Data set for fall events and daily activities from inertial sensors. In Proceedings of the 6th ACM Multimedia Systems Conference, Portland, OR, USA, 18–20 March 2015; Ooi, W.T., Feng, W., Liu, F., Eds.; ACM: New York, NY, USA, 2015; pp. 243–248. [CrossRef]
15. Vilarinho, T.; Farshchian, B.; Bajer, D.G.; Dahl, O.H.; Egge, I.; Hegdal, S.S.; Lones, A.; Slettevold, J.N.; Weggersen, S.M. A Combined Smartphone and Smartwatch Fall Detection System. In Proceedings of the CIT/IUCC/DASC/PICom 2015, Liverpool, UK, 26–28 October 2015; pp. 1443–1448. [CrossRef]
16. Wertner, A.; Czech, P.; Pammer-Schindler, V. An Open Labelled Dataset for Mobile Phone Sensing Based Fall Detection. In Proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Coimbra, Portugal, 22–24 July 2015; Zhang, P., Silva, J.S., Lane, N., Boavida, F., Rodrigues, A., Eds. MobiQuitous: Coimbra, Portugal, 2015. [CrossRef]
17. Aziz, O.; Musngi, M.; Park, E.J.; Mori, G.; Robinovitch, S.N. A comparison of accuracy of fall detection algorithms (threshold-based vs. machine learning) using waist-mounted tri-axial accelerometer signals from a comprehensive set of falls and non-fall trials. *Med. Biol. Eng. Comput.* **2017**, *55*, 45–55. [CrossRef]
18. Sucerquia, A.; López, J.D.; Vargas-Bonilla, J.F. SisFall: A Fall and Movement Dataset. *Sensors* **2017**, *17*, 198, [CrossRef]
19. Micucci, D.; Mobilio, M.; Napoletano, P. UniMiB SHAR: A Dataset for Human Activity Recognition Using Acceleration Data from Smartphones. *Appl. Sci.* **2017**, *7*, 1101. [CrossRef]
20. Mauldin, T.R.; Canby, M.E.; Metsis, V.; Ngu, A.H.H.; Rivera, C.C. SmartFall: A Smartwatch-Based Fall Detection System Using Deep Learning. *Sensors* **2018**, *18*, 3363. [CrossRef]
21. Chan, H.L. CGU-BES Dataset for Fall and Activity of Daily Life. *Figshare* **2018**. [CrossRef]
22. Martínez-Villaseñor, L.; Ponce, H.; Brieva, J.; Moya-Albor, E.; Núñez-Martínez, J.; Peñafort-Asturiano, C. UP-Fall Detection Dataset: A Multimodal Approach. *Sensors* **2019**, *19*, 1988, [CrossRef]
23. Cotechini, V.; Belli, A.; Palma, L.; Morettini, M.; Burattini, L.; Pierleoni, P. A dataset for the development and optimization of fall detection algorithms based on wearable sensors. *Data Brief* **2019**, *23*, 103839. [CrossRef]

24. Mauldin, T.; Ngu, A.H.; Metsis, V.; Canby, M.E.; Tesic, J. Experimentation and analysis of ensemble deep learning in iot applications. *Open J. Internet Things* **2019**, *5*, 133–149.
25. Pannurat, N.; Thiemjarus, S.; Nantajeewarawat, E. Automatic Fall Monitoring: A Review. *Sensors* **2014**, *14*, 12900–12936. [CrossRef]
26. Krupitzer, C.; Sztyler, T.; Edinger, J.; Breitbach, M.; Stuckenschmidt, H.; Becker, C. Hips Do Lie! A Position-Aware Mobile Fall Detection System. In Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications (PerCom); Communications, Athens, Greece, 19–23 March 2018; pp. 1–10. [CrossRef]
27. Liao, M.; Guo, Y.; Qin, Y.; Wang, Y. The application of EMD in activity recognition based on a single triaxial accelerometer. *Bio-Med. Mater. Eng.* **2015**, *26* (Suppl. 1), S1533–S1539. [CrossRef]
28. Bourke, A.K.; O'Brien, J.V.; Lyons, G.M. Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm. *Gait Posture* **2007**, *26*, 194–199. [CrossRef]
29. Kangas, M.; Konttila, A.; Winblad, I.; Jämsä, T. Determination of simple thresholds for accelerometry-based parameters for fall detection. In Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Lyon, France, 22–26 August 2007; pp. 1367–1370. [CrossRef]
30. Salomon, R.; Lueder, M.; Bieber, G. iFall—A New Embedded System for the Detection of Unexpected Falls. In Proceedings of the Eighth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Mannheim, Germany, 29 March–2 April 2010. [CrossRef]
31. Vallabh, P.; Malekian, R.; Ye, N.; Bogatinoska, D.C. Fall detection using machine learning algorithms. In Proceedings of the 2016 24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 22–24 September 2016; pp. 1–9. [CrossRef]
32. Santos, G.L.; Endo, P.T.; Monteiro, K.H.C.; Rocha, E.S.; Silva, I.; Lynn, T. Accelerometer-Based Human Fall Detection Using Convolutional Neural Networks. *Sensors* **2019**, *19*, 1644. [CrossRef]
33. Wang, G.; Li, Q.; Wang, L.; Zhang, Y.; Liu, Z. Elderly Fall Detection with an Accelerometer Using Lightweight Neural Networks. *Electronics* **2019**, *8*, 1354. [CrossRef]
34. Jahanjoo, A.; Tahan, M.N.; Rashti, M.J. Accurate fall detection using three-axis accelerometer sensor and MLF algorithm. In Proceedings of the 2017 3rd International Conference on Pattern Recognition and Image Analysis (IPRIA), Shahrekord, Iran, 19–20 April 2017; pp. 90–95. [CrossRef]
35. Ramachandran, A.; Adarsh, R.; Pahwa, P.; Anupama, K.R. Machine Learning-based Fall Detection in Geriatric Healthcare Systems. In Proceedings of the 2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), Indore, India, 16–19 December 2018; pp. 1–6. [CrossRef]
36. Wisesa, I.W.W.; Mahardika, G. Fall detection algorithm based on accelerometer and gyroscope sensor data using Recurrent Neural Networks. *IOP Conf. Ser. Earth Environ. Sci.* **2019**, *258*, 012035. [CrossRef]
37. Hussain, F.; Hussain, F.; Ehatisham-ul Haq, M.; Azam, M.A. Activity-Aware Fall Detection and Recognition Based on Wearable Sensors. *IEEE Sens. J.* **2019**, *19*, 4528–4536. [CrossRef]
38. Musci, M.; De Martini, D.; Blago, N.; Facchinetti, T.; Piastra, M. Online fall detection using recurrent neural networks. *arXiv* **2018**, arXiv:1804.04976.
39. Liu, K.C.; Hsieh, C.Y.; Hsu, S.J.P.; Chan, C.T. Impact of Sampling Rate on Wearable-Based Fall Detection Systems Based on Machine Learning Models. *IEEE Sens. J.* **2018**, *18*, 9882–9890. [CrossRef]
40. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*; AcM: New York, NY, USA, 2012; pp. 1097–1105.
41. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009.
42. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Chengdu, China, 15–17 December 2017; pp. 770–778.
43. Wang, Z.; Yan, W.; Oates, T. Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline. *CoRR* **2016**, *abs/1611.06455*. Available online: https://arxiv.org/abs/1611.06455 (accessed on 30 October 2020).
44. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]

45. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
46. Virtanen, P.; Gommers, R.; Oliphant, T.E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; others. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* **2020**, *17*, 261–272. [CrossRef]
47. Sztyler, T.; Stuckenschmidt, H. On-body Localization of Wearable Devices: An Investigation of Position-Aware Activity Recognition. In Proceedings of the 2016 IEEE International Conference on Pervasive Computing and Communications (PerCom), IEEE Computer Society, Sydney, Australia, 14–19 March 2016; pp. 1–9. [CrossRef]
48. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA. **2016**.
49. Um, T.T.; Pfister, F.M.J.; Pichler, D.; Endo, S.; Lang, M.; Hirche, S.; Fietzek, U.; Kulić, D. Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks. In Proceedings of the 19th ACM International Conference on Multimodal Interaction, Glasgow, Scotland, 13–17 November, 2017; Lank, E., Ed.; ACM: New York, NY, 2017; pp. 216–220. [CrossRef]
50. Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic differentiation in PyTorch. In Proceedings of the NIPS 2017 Workshop Autodiff Submission, Long Beach, CA, USA, 9 December, 2017.
51. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
52. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1026–1034.
53. Springenberg, J.T.; Dosovitskiy, A.; Brox, T.; Riedmiller, M. Striving for simplicity: The all convolutional net. *arXiv* **2014**, arXiv:1412.6806.
54. Fawaz, H.I.; Forestier, G.; Weber, J.; Idoumghar, L.; Muller, P. Deep learning for time series classification: A review. *CoRR* **2018**, *abs/1809.04356*. Available online: https://arxiv.org/abs/1809.04356 (accessed on 30 October 2020).
55. Wu, J.; Leng, C.; Wang, Y.; Hu, Q.; Cheng, J. Quantized convolutional neural networks for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 4820–4828.
56. Zhou, B.; Khosla, A.; Lapedriza, A.; Oliva, A.; Torralba, A. Learning deep features for discriminative localization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 2921–2929.
57. Aadahl, M.; Andreasen, A.H.; Hammer-Helmich, L.; Buhelt, L.; Jørgensen, T.; Glümer, C. Recent temporal trends in sleep duration, domain-specific sedentary behaviour and physical activity. A survey among 25–79-year-old Danish adults. *Scand. J. Public Health* **2013**, *41*, 706–711. [CrossRef]