applied sciences

MDPI

Article

Toward Real-Time, Robust Wearable Sensor Fall Detection Using Deep Learning Methods: A Feasibility Study

Haben Yhdego [1], Christopher Paolini [2] and Michel Audette [1,*]

[1] Electrical and Computer Engineering, Old Dominion University, Norfolk, VA 23529, USA; hyhde001@odu.edu
[2] Electrical and Computer Engineering, San Diego State University, San Diego, CA 92182, USA; cpaolini@sdsu.edu
* Correspondence: maudette@odu.edu

Abstract: Real-time fall detection using a wearable sensor remains a challenging problem due to high gait variability. Furthermore, finding the type of sensor to use and the optimal location of the sensors are also essential factors for real-time fall-detection systems. This work presents real-time fall-detection methods using deep learning models. Early detection of falls, followed by pneumatic protection, is one of the most effective means of ensuring the safety of the elderly. First, we developed and compared different data-segmentation techniques for sliding windows. Next, we implemented various techniques to balance the datasets because collecting fall datasets in the real-time setting has an imbalanced nature. Moreover, we designed a deep learning model that combines a convolution-based feature extractor and deep neural network blocks, the LSTM block, and the transformer encoder block, followed by a position-wise feedforward layer. We found that combining the input sequence with the convolution-learned features of different kernels tends to increase the performance of the fall-detection model. Last, we analyzed that the sensor signals collected by both accelerometer and gyroscope sensors can be leveraged to develop an effective classifier that can accurately detect falls, especially differentiating falls from near-falls. Furthermore, we also used data from sixteen different body parts and compared them to determine the better sensor position for fall-detection methods. We found that the shank is the optimal position for placing our sensors, with an F1 score of 0.97, and this could help other researchers collect high-quality fall datasets.

Keywords: fall detection; imbalanced dataset; sliding-window segmentation; wearable sensors; deep learning

Citation: Yhdego, H.; Paolini, C.; Audette, M. Toward Real-Time, Robust Wearable Sensor Fall Detection Using Deep Learning Methods: A Feasibility Study. *Appl. Sci.* **2023**, *13*, 4988. https://doi.org/10.3390/app13084988

Academic Editor: Bang Wang

Received: 12 March 2023
Revised: 5 April 2023
Accepted: 13 April 2023
Published: 16 April 2023

1. Introduction

The data from the World Health Organization show that approximately six hundred thousand falls occur yearly, which account for the second-highest mortality rate following traffic accidents [1]. One-third of the aging population falls each year. USA statistics show that about 36 million falls occur annually and one-fourth (28%) of people aged 65 and above report falls yearly [2]. Of these falls, 37% of them (8 million) were injured and needed medical treatment [2]. Due to these reasons, there is a national imperative to develop cost-effective real-time fall detection with new sensor technologies and methods.

Fall-detection systems help us differentiate falls from non-fall activities so that an alert system to a remote monitoring point is automatically emitted as soon as the patient falls or in order to enable the deployment of a protective airbag, if the detection can be sufficiently accelerated [3]. Recently, several methods have been proposed using different sensors with varying performance levels [4–6]. The most common sensor technologies used for fall recognition are categorized into three classes: camera sensors, infrared sensors, and wearable sensors. Infrared and camera-based sensors are expensive sensors and as they record audiovisual signals, they have issues related to the privacy of patients and the stationary aspect of the system. Due to these limitations, wearable sensor-based fall

detection offers a cheaper alternative to detect falls based on one or more wearable sensors that are attached to the user's body or clothing so they can be carried everywhere.

The analysis of signals obtained from wearable sensors mounted on the human body is commonly used to monitor the health status of older patients with motion-assistive devices [7]. These sensors typically generate complex hip motion signals that are difficult to interpret without expert intervention. A computationally efficient fall-detection modeling technique is required to provide a meaningful characterization of the sensor data, which could be leveraged to trigger the pneumatic actuation of a proactive airbag. This automatically analyzes sensor readings to infer the kind of human activity performed by a user. Many researchers have been developing supervised-based fall-detection methods in the last decades [4–6]. However, there are still some limitations to obtaining fall datasets that contain near-fall activities, which are the cause of false positives and false negatives.

The most common methods for fall-detection systems that use wearable sensors involve thresholding or machine learning techniques [4]. In the simple threshold fall-detection method, they calculate the threshold values of either the magnitude or the vertical projection of the acceleration [8,9]. When either values are greater than the specified thresholds, a fall is detected. In a related work, Viet uses both upper thresholds for the acceleration magnitude [10]. If the upper threshold is exceeded in less than 1 s, a fall template using a wavelet transform is used for comparison with the acceleration signal. A fall is detected if the comparison produces a high similarity value. Although automatic fall detection using a threshold-based method of individual parameters calculated from accelerometry measurements has a high sensitivity, it has a relatively low specificity [11]. For example, Purwar used a triaxial accelerometer to set the acceleration and torso orientation thresholds through experiments to detect falls, which achieved an accuracy of 81% [12].

Montesinos [5] and Klenk [6] have thoroughly compared the performance of basic thresholding techniques with a wide set of supervised learning solutions. It is difficult to use threshold techniques as we decide the upper threshold value of fall and non-fall activities for new test subjects. Due to this limitation, recent approaches [13] employ feature extraction engineering and machine learning classifiers to improve the detection accuracy.

In the last decade, many fall-detection methods have been developed using classical machine learning techniques [14] and deep learning [15–17]. Some of the developed fall-detection systems use logistic regression [14], naive Bayes [18,19], decision tree [18,19], support vector machines [18], and k-nearest neighbors [18–20]. Deep learning algorithms have also been applied to gait analysis and fall-detection systems. Other methods detect falls using inertia sensor signals by converting the 1D sensor signals to 2D images of spectrotemporal footprints and then applying CNN to classify those converted images [21,22].

As a wearable sensor-based fall-detection method generates sequential datasets, it has the advantage of using sequential models such as LSTM. The sensor fusion of the accelerometer and gyroscope data streams using a hybrid CNN-LSTM method [15], as well as LSTM-based activity recognition [16,23], is used for the recognition of falls. The study in [16] uses a single inertial sensor placed in the trunk. The CNN and LSTM model inputs are the raw acceleration and angular velocity signals. The problem with Ruben's [15] approach is that he uses KNN classifiers—it is challenging to use such algorithms for real-time fall detection due to the Euclidean distance computational time.

Long short-term memory (LSTM) and transformers are popular algorithms used in sequence models to encode and process sequential data. In this article, we extend our previous works in [17,24] by proposing a new technique for fall-detection systems. Our approach is based on a vision transformer [25–27] and deep LSTM [28], which are used for image recognition and computer vision. First, we collected fall datasets, pre-processed, segmented, and balanced minor-class data samples. Fall kinematics data have different lengths, and it is difficult to include the fall event in a single-window input sequence. Thus, we introduce different sliding-window segmentation methods of the input sequence that help us to develop a real-time fall-detection system.

Recently, many researchers have been developing vision transformers (ViT) [27] and deep LSTM [28] methods for image classification. This paper explores convolutional feature extractors with deep learning feature fall-detection systems on wearable sensor datasets. Two deep learning models are used as our imminent detection model, the LSTM encoder and the transformer encoder. To balance the minor classes, several techniques—resampling, data augmentation, and customized loss functions—are used. We collect data using an accelerometer and a gyroscope sensor to detect the impending fall correctly, while reducing false positives and false negatives and collecting near-fall datasets. We also propose several sliding-window segmentation techniques that include the important signal information in a single observation window across the various features of sequential data to detect falls.

2. Materials and Methods

Two competing deep learning methods are used to detect falls from non-fall activities. We use wearable sensor signals to train and test the two models—LSTM and transformer encoders. First, we present the collected fall datasets and then explain the required pre-processing, sliding-window segmentation, and balancing of the dataset. Lastly, the different deep learning models are explained.

2.1. Data

One of us (CP) conducted fall experiments on human subjects in the Neuromechanics and Neuroplasticity Laboratory of San Diego State University. He collected the fall dataset with a sampling rate of 200 HZ [29] based on accelerometers mounted on different parts of the subject's body. The laboratory is equipped with wireless 3D motion-capture cameras that record human subject movements [30]. The data were collected from sixteen subjects between the age of 20 and 50 years; two of them are females. The Noraxon myoMOTION research inertial measurement unit (IMU) sensors measure features such as acceleration and angular velocity [29]. The datasets contain activities such as near-fall, backward and forward fall, obstacle fall, and ADL (activities of daily living).

2.2. Data Pre-Processing

First, the datasets are pre-processed to remove high-frequency noise. The signal noise comes from the sensor itself or from the movement of the body. To remove the noise signal, we use a first-order low-pass Butterworth filter [31,32]. Despite the fact that the second-order Butterworth filter provides better results, the first-order low-pass filter is selected because it requires fewer computations and is efficient enough to remove fall-activity-related signals.

Feature normalization is a data pre-processing method that is used to map two or more sets of feature values, with disparities in their respective ranges, to a common range. The variations in accelerometer and gyroscope units such as revolutions per second are unlikely to be equivalent. Furthermore, feature normalization enables our model to converge rapidly while ensuring that the contribution of the feature is equivalent [33–35]. Thus, it is necessary to normalize the sensor measurements that have been selected as inputs to our model. It appears that the way of normalizing the features plays an important role in the overall success of the performance of our algorithm. In our approach, the data points for each feature are normalized using the minimum and maximum of each feature. The feature-normalization pre-processing is performed on the 3D acceleration signals $A(t) = [A_x(t), A_y(t), A_z(t)]$ and 3D gyroscope signals $G(t) = [G_x(t), G_y(t), G_z(t)]$. The general formula is given as follows:

$$\widehat{Feat} = \frac{Feat - min(Feat)}{max(Feat) - min(Feat)} \quad (1)$$

where Feat is the vector of features x, y, and z of the accelerometer and gyroscope, and $\widehat{Feat}$ is the normalized feature vector calculated using the maximum and minimum values of the vector of features.

2.3. Sliding-Window Segmentation Methods

The collected dataset has a variable temporal duration of the sequence since the data collection time period for each subject is different. Although most research papers [15,16,36] use a more than 2-s sliding-window segmentation in their method, real-time fall-detection applications should respond in at least less than 0.2 s. Therefore, we implement competing sliding-window segmentation techniques to detect falls in near-real time, anticipating that improvement in the hardware efficiencies will ultimately compress this response time. We compare three methods of window segmentation: conventional sliding window, peak-detection sliding window, and dynamic multi-window sliding.

2.3.1. Conventional Sliding Window

The segmentation of signal data plays a critical role in signal recognition and prediction. A sliding-window approach segments the continuous time series dataset into short segments. Many researchers [15,16,36] use the sliding-window approach to engineer the feature of the fall datasets. There are two methods of sliding-window segmentation for fall detection: fixed-size non-overlapping sliding windows and fixed-size overlapping sliding windows. We use fixed-size overlapping sliding windows because it enables us to generate more data by overlapping them. The fixed-size overlapping sliding window technique processes the signal data along the temporal axis based on the window size and stride size (step size). Here, we segment the sample signal into a 0.2 s sample signal for each label using a fixed-size overlapping sliding window. The size of window that overlaps is 0.1 s. If we obtain a single row of falls in this 0.2 s window signal, this window is labeled a fall.

2.3.2. Peak-Detection Windowing

The human gait contains peak acceleration and angular velocity signals when fall and near-fall activities occur. Hence, the need to accurately detect the fall of senior subjects requires us to differentiate the peak signals of near-falls from those of fall events. Due to the alternation in the signal values, we take the highest signal values of the acceleration as the basis to define the observation window of the detector.

Therefore, to use these advantages, we start the engineering of sliding-window features from the maximum magnitude of the acceleration signal (as shown in Algorithm 1) rather than using the conventional engineering of sliding-window features. This kind of windowing helps us to include the important peak signal value in a single window, besides paying great attention to the maximum-value signal. If we used conventional windowing based on the raw signal, the important information may be shared in different windows. For our proposed method, the maximum values of the acceleration and angular velocity simulated in the accelerometer and gyroscope datasets, consisting of 3D acceleration data $acc(t) = [acc_x(t), acc_y(t), acc_z(t)]$ and 3D angular velocity $gyro(t) = [gyro_x(t), gyro_y(t), gyro_z(t)]$ [22], are calculated using the magnitude of these vectors for the i^{th} sample as follows:

$$\|acc_i(t)\| = \sqrt{acc_{xi}^2(t) + acc_{yi}^2(t) + acc_{zi}^2(t)}\,. \tag{2}$$

$$\|gyro_i(t)\| = \sqrt{gyro_{xi}^2(t) + gyro_{yi}^2(t) + gyro_{zi}^2(t)}\,. \tag{3}$$

Next, the maximum magnitude of the signal calculated above is determined as follows.

$$\|acc_{max}\| = max(\{\|acc_i\| : i \in [1 : M]\}), \tag{4}$$

$$\|gyro_{max}\| = max(\{\|gyro_i\| : i \in [1 : M]\}), \tag{5}$$

where M is the number of samples. Using the above two equations and the size of the window (which is 0.2 s in our case), the sliding window around the peak signal is as shown in Figure 1.

Algorithm 1 Calculating the maximum value and labeling the window

input : $acc(t) = [acc_x(t), acc_y(t), acc_z(t)]$ and
$gyro(t) = [gyro_x(t), gyro_y(t), gyro_z(t)]$
output: Observation windows and their labels

for *all data sequence activities of FallData* **do**
  Calculate the magnitude using equation 2 and 3;
  Calculate the maximum value using equation 4 and 5;
  if *FallData is ADL* **then**
    Label all the observation windows as ADL;
  else
    if *the observation window is maximum-value window* **then**
      Label the window as Fall;
    else
      Label ADL;

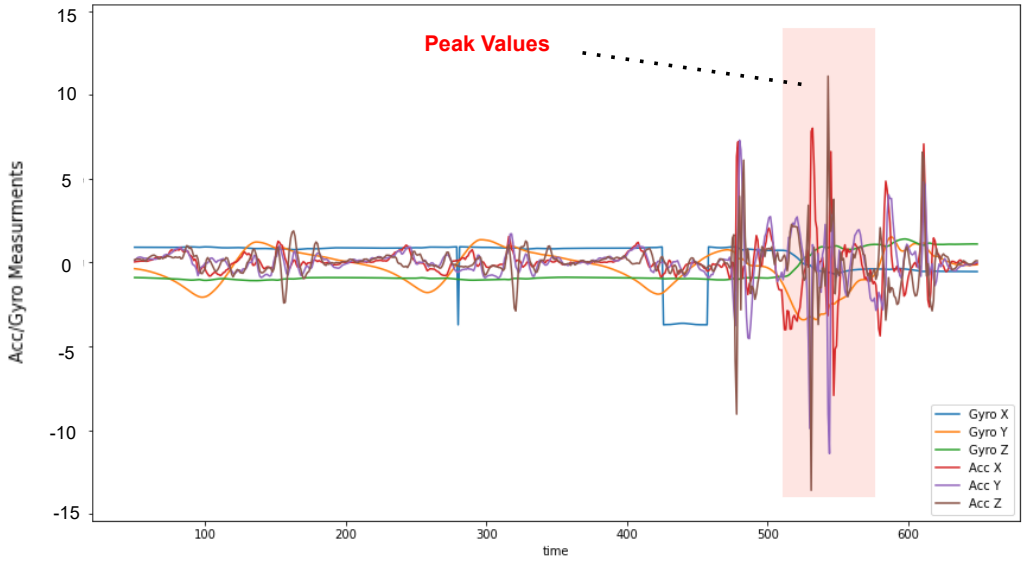

Figure 1. Sliding-window feature engineering based on the maximum signal (peak values).

After we estimate the maximum acceleration and decide on the size of the sliding window, the algorithm above gives us the input features and their labels, which are formed by simply concatenating the six features of the accelerometer and gyroscope, $\{Acc_{xj}, Acc_{yj}, Acc_{zj}, Gyro_{xj}, Gyro_{yj}, Gyro_{zj} | j \in [r_o - \frac{w}{2} f_s, r_o + \frac{w}{2} f_s]\}$, where w is the duration of the observation window (0.2 s) and f_s the sampling rate of the sensor (200 Hz) [22]. Finally, the size of the input features (N_i) that depends on the duration of the observation window (w) is $N_i = 6 * (w * f_s + 1)$.

2.3.3. Sliding Multi-Window

Contrary to the above two methods, the multi-window segmentation approach uses a dynamic main window based on the label of the datasets and sliding conventional sub-windows inside the main window. We use a fixed-size overlapping sliding window for the sub-windows inside the label-based main window. Using the main window based on the label of the datasets and the window size (0.2 s) of the sub-windows, the sliding-window approach looks like Algorithm 2. The observation window is extracted as shown in Figure 2.

Algorithm 2 Sliding Multi-window

input : $acc(t) = [acc_x(t), acc_y(t), acc_z(t)]$ and
$gyro(t) = [gyro_x(t), gyro_y(t), gyro_z(t)]$
output: Observation windows and their labels

for *all data sequence activities of fall datasets* **do**
  if *Label is ADL* **then**
    Sliding sub-window from the start to the end of ADL label index
    Label all the sub-window observation windows as ADL;
  else if *Label is Fall* **then**
    Sliding sub-window from the starting to the end of Fall label index
    Label all the sub-window observation windows as Fall;

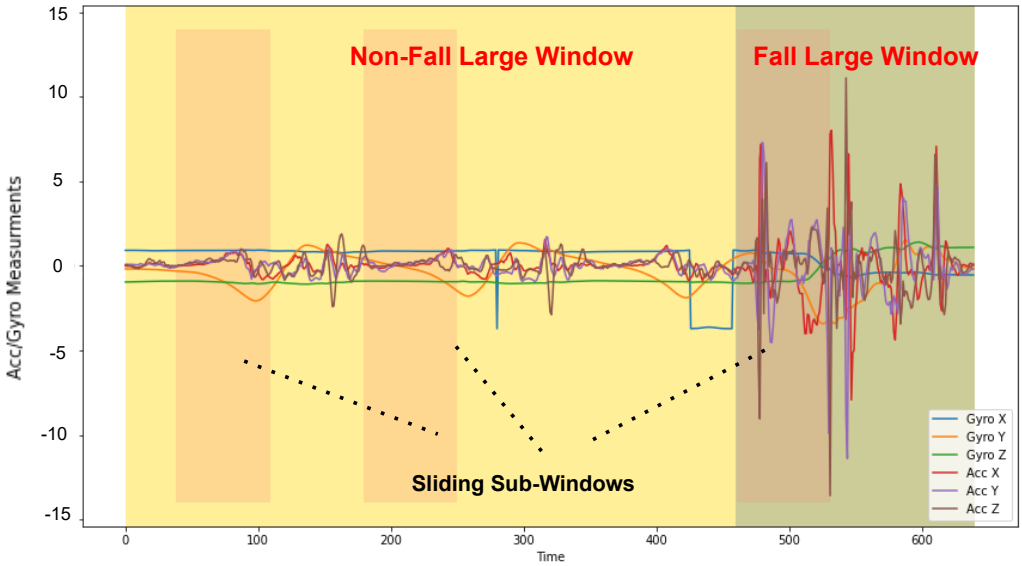

Figure 2. Sliding Multi-window feature engineering.

2.4. Handling Imbalanced Datasets

Our collected dataset has 13% of the fall label dataset and 87% of non-fall activities, they are highly imbalanced data. The main problem of not considering such imbalanced datasets is our deep learning models make our minor label classes suffer from low results, although the accuracy of those minority classes is the most important one. We present different methods for balancing the datasets: resampling techniques, data augmentation, and customized loss function. These three methods are compared to balance our training datasets so that the deep learning methods will create results that will not suffer in the minor classes.

2.4.1. Resampling Techniques

The main objective of balancing datasets is to increase the number of the minority class or decrease the number of the majority class. This is performed to obtain an equivalent frequency of instances for both classes. The most familiar resampling approaches for resolving the imbalanced-dataset problem are oversampling the minority class and undersampling the majority class. The well-known undersampling method is called random undersampling, and it randomly deletes window signals from the major classes of the training dataset. The most common and simplest method of this oversampling of the minority class is random oversampling, which simply duplicates random sequences of window signals from the minority class in the training datasets. Undersampling for the majority class loses some of the information, whereas oversampling for the minority

class does not lose any data. Undersampling helps to improve the run-time and storage problems by reducing the number of training data samples, even though it loses some of the information, whereas oversampling for the minority class does not lose any data, but it increases the likelihood of overfitting since it replicates the minority class events.

Thus, these duplicated examples do not provide new information to the model. Due to the above reason, we also try to compare another method called the Synthetic Minority Oversampling Technique (SMOTE) [37] to random undersampling and random oversampling. This approach of balancing datasets helps us to avoid overfitting, which occurs when exact replicas of minority instances are added to the main dataset. A subset of data is taken from the minority class as an example, and then new similar synthetic instances are created. These synthetic instances are then added to the original dataset. The new dataset is used as a sample to train the classification models. SMOTE works by selecting random examples from the minority class that are close to most of the feature variables. Subsequently, its k nearest minority-class neighbors are found, and then the line segment in the feature space is formed to obtain a synthetic example generated randomly in this line segment [37].

2.4.2. Data Augmentation

A variety of transformations can be applied to data in order to augment it with prior knowledge about its invariant properties. By augmenting the input data, DL models can better cover the unexplored input space, prevent overfitting, and improve generalization [38]. In image recognition, it is well known that minor changes due to jittering, scaling, cropping, warping, or rotating do not affect the data labels. Wearable sensor data can be transformed in a label-preserving way, but this is not intuitive [39].

Rotation: Data from wearable sensors are subjected to rotational data augmentation to introduce label-invariant variability [39]. When the sensor is placed upside down, the readings can be inverted without changing the labels. As a result, 180-degree rotations can be used to simulate upside-down sensor placements by augmenting the existing data. Based on the acceleration vector $[a](t) = [a_x(t), a_y(t), a_z(t)]$ for time t, which contains acceleration components along the x, y, and z axes, respectively, a new vector $[a] * r(t)$ can be obtained by rotating $[a](t)$ 180 degrees in the x, y, and z axes.

MixUp: MixUp augementation is a technique that combines two examples of data to enhance them in a more meaningful way [40]. A temporal MixUp is a method in which all values of a set of features in the first input sequence (observation window) of the sensor are multiplied by a random value, m, and then added, with all features of the same sensor from a second randomly selected input sequence (observation window) of the sensor multiplied by m. Generally, m is chosen between the input sequence's minimum and maximum and is selected randomly between the two. As a result of this operation, all six features of the accelerometer and gyroscope are affected.

CutMix: By using the CutMix technique, two data samples can be combined [41]. A temporal CutMix is a method that selects a random time segment from a first input sequence (observation window) and a random time segment from a random second input sequence (observation window) of both labels based on time-segment selection. Both sequences have two random segments that start at the same time step and end at the same time step. Then, CutMix selects a random set of sensors and replaces the channel values of the first multi-feature segment with the channel values of the second multi-feature segment. Whenever the size of the random time segment is determined at random, it ranges between the maximum and minimum values of the hyperparameters. It turns out that the channel probability is a hyperparameter that represents the chance that each channel will be selected for this operation on a given day.

2.4.3. Customized Loss Function

Lastly, we use a customized loss function to handle the imbalanced classes. To train our fall dataset, we customized the focal loss presented by Lin et al. [42] that is used for object detection. We use our modified loss function used to tune our models parameters

while training our fall dataset. Even though the wrongly classified samples are penalized more than the correct ones, in the fall-detection settings, due to the imbalanced sample size, the loss function is overwhelmed with non-fall activity (ADL) classes. Customized focal loss addresses this problem and is designed in such a way that it reduces the loss for the ADL classes, and thus, the network can focus on training the fall classes. During supervised training of the dataset, our deep learning model is optimized end-to-end using a customized weighted focal loss in Equation (6):

$$L_{FocalLoss} = \sum_{i=1}^{c=2} w_i (1 - p_i)^{\gamma} \log(p_i), \tag{6}$$

where

$$w_i = \frac{n_0 + n_1}{2 * n_i} \tag{7}$$

and n_0 is the number of non-fall classes, n_1 is the number of fall classes, and γ is a focusing parameter whose value is $\gamma >= 0$. This focusing parameter is tailored to reduce the influence of higher-confidence classified samples of ADL classes in the loss. The higher the γ, the higher the rate at which easy-to-classify examples are down-weighted. If $\gamma = 0$, a weighted focal loss is equivalent to a weighted binary cross-entropy loss.

2.5. Sensor Positions and Sensor Types

Many fall-detection studies [43] utilize an accelerometer as the primary sensor to determine falls. Chernbumroong [43] evaluates the importance of different sensors in a multisensory scheme, concluding that the most relevant features are derived from two particular acceleration components (Z and Y). A gyroscope is considered computationally expensive and is less useful in improving our understanding of the dynamics of falling [4,44,45]. In contrast to this, Nguyen et al. [46] report that a sensing module (including a gyroscope and an accelerometer) consumes only about 3% more power when both sensors are active compared to only measuring the acceleration magnitudes when only one sensor is activated.

Furthermore, the choice of sensors to use depends on the type of dataset (whether it contains near-fall activities or not). Using only acceleration measurements can result in many false positives and false negatives caused by near-fall activities, such as sitting down fast on a mattress. Near-fall and fall activities have almost the same vertical signal variation, making them difficult to differentiate. False positives and false negatives caused by near-fall activities can be reduced significantly by using the gyroscope's angular velocity measurements. Our paper compares an accelerometer-based algorithm and a gyroscope-based algorithm intended for fall detection. The results indicate that the performance of the algorithm that uses gyroscope signals is more effective when the dataset contains near-fall activities.

We collected data from sixteen different positions: the head, pelvis, upper and lower thoracic, upper and lower arms (wrists), hand, thigh, shank, and foot. Most studies in the literature review on fall recognition have been conducted using a single wearable device without taking into account the device's location on the user's body. The position information provided by the wearable device can, however, assist in improving the performance of the fall-detection system. Several body locations are compared in this study to determine the most optimal location of the sensors for fall-detection systems.

2.6. Proposed Model Architecture

As shown in Figure 3, the proposed overall model architecture contains three parts: a feature extractor, an LSTM or transformer encoder block, and our linear classifier. In the feature-extractor part, three convolutions with different filter sizes are used to extract features from the segmented input sequence. The extracted features and the input sensor signal are concatenated to obtain 4N, where N is the number of features of the input signals. In the next block, we will use two different types of models to extract the dependency

among the input sensor signals. The first has three stacked bidirectional LSTM encoders, and the second has three stacked transformer encoders. Each LSTM or transformer encoder layer is followed by layer normalization and then passes through the Gelu activation function. The global average pooling layer and the dense layer are applied to the result found from the last LSTM or transformer encoder block. Finally, a sigmoid classifier is used for classifying falls from non-fall activities.

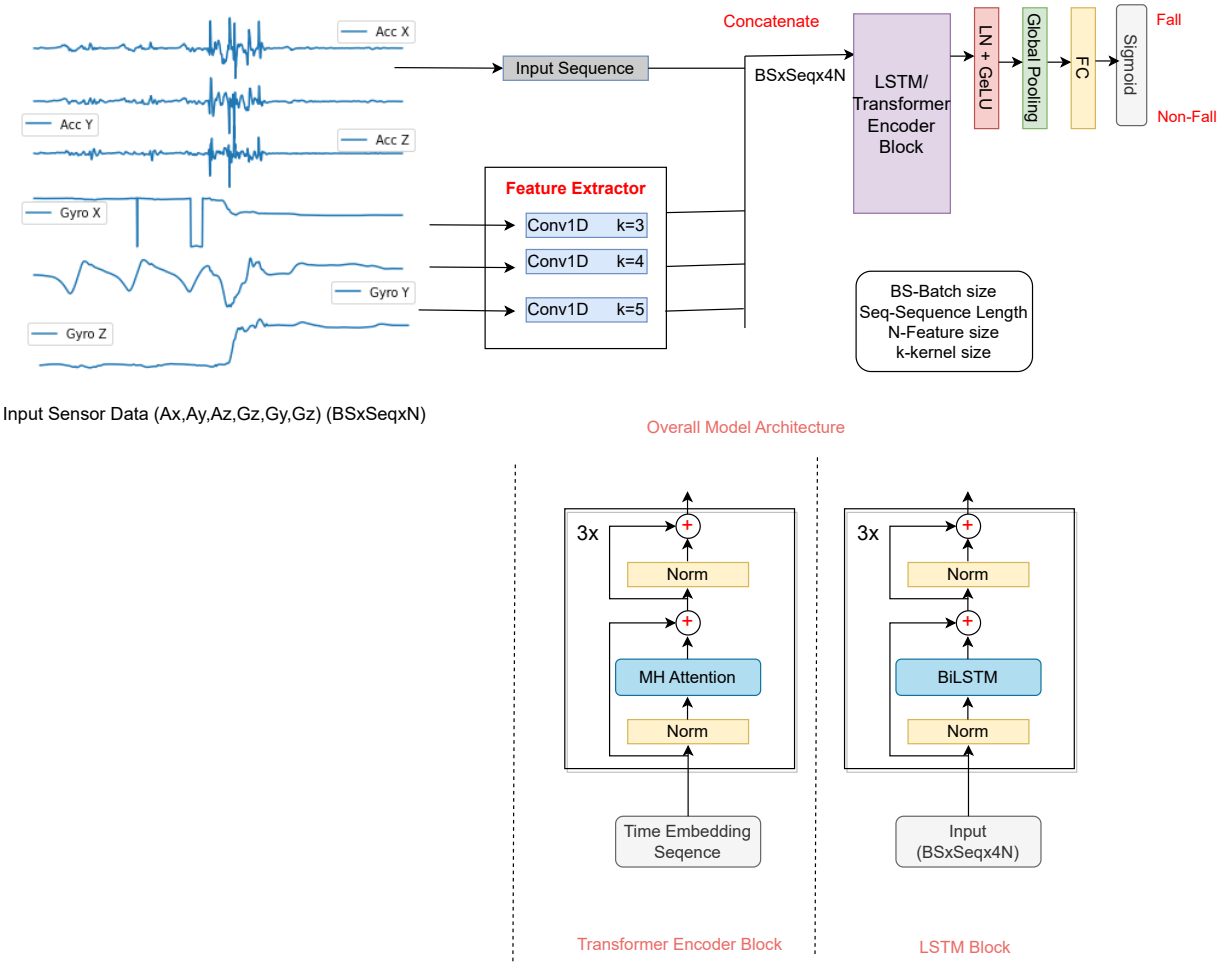

Figure 3. Overall model architecture, transformer encoder block, and LSTM encoder block.

2.6.1. Feature Extractor

In a feature extractor block, kernels or filters are applied to project input features onto another meaningful dimension to produce hidden feature maps. Using hidden feature maps as the input for the following layers, the networks can accurately extract or comprehend the original input's semantic meaning. With a convolutional layer, the kernels (filters) scan multiple cells at once, revealing hidden meanings and effects between them. Convolutional layers are often used because of their ability to extract adjacent cells and faster inference times compared to RNNs.

The conv1D extracts features from a one-dimensional sequence of acceleration and gyroscopic data. The model learns to extract features from sequences of observations and map the internal features to the LSTM or transformer encoder. In order to extract dense feature representations of our input signals for effective learning, three different representations of features with varying filters sizes are computed.

2.6.2. LSTM Encoder

For sequence modeling, long short-term memory (LSTM) has been considered a highly successful RNN architecture. When time is a relevant feature, the easiest way to handle it is to concatenate the time features with the input and use the LSTM model. Due to their ability to learn how and when to forget and when not to use gates, LSTMs and their bidirectional variants are popular.

When sequential data are processed, the LSTM often ignores future information. Based on LSTM, BiLSTM processes the data in the series forward and reverse, connecting the two hidden layers to the same output layer [47] and storing previous and subsequent information as the current time basis for the time series data [48,49]. Accordingly, multi-directional LSTM is more accurate than unidirectional LSTM. In BiLSTM [50], the hidden layer output includes both forward and backward activation outputs.

$$\begin{aligned} \overrightarrow{h_t} &= \sigma(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h_{t-1}} + b_{\overrightarrow{h}}) \\ \overleftarrow{h_t} &= \sigma(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h_{t-1}} + b_{\overleftarrow{h}}) \\ H_t &= W_{x\overrightarrow{h}} \overrightarrow{h} + W_{\overleftarrow{h}y} \overrightarrow{h} + b_y \end{aligned} \tag{8}$$

where W and b are weight matrices and vectors, t is the current iteration of the recurrent network, σ is the activation function, H_t is the input of the hidden layer, and the output is generated by updating the forward structure $\overrightarrow{ht}$ and the backward structure $\overleftarrow{ht}$.

2.6.3. Transformer Encoder

For the transformer model to function properly, we need to attach the meaning of time to our input features. In the original NLP model, a collection of superimposed sinusoidal functions was added to each input embedding. We now require a different representation because our inputs are scalar values and not distinct words (tokens). After we concatenate the input features to our transformer encoder, we encode the sequence of time (time embedding), which is hidden in our signal data. We implement the existing Time2Vec method [50] for time embedding. This time embedding is a vector representation just like a normal embedding layer that can be added to the neural network architecture to improve the performance of a model and overcome the temporal indifferences of the transformer. The mathematical representation of Time2Vec shown in Equation (9) for the ideas of periodic and non-periodic patterns, as well as the invariance to time rescaling, are used [50]:

$$\textbf{Time2Vec}(\tau)[i] = \begin{cases} \omega_i \tau + \varphi_i & \text{for} \quad i = 0 \\ F(\omega_i \tau + \varphi_i) & \text{for} \quad 1 \le i \le k \end{cases} \tag{9}$$

where F is a periodic function and φ_i and ω_i are learnable parameters.

In this paper, we use the transformer encoder based on the ViT [27] that is developed for image recognition. In a recent publication, Vaswani et al. proposed the implementation of multi-head attention (transformer) [51]. The functionality of a multi-head attention layer is to concatenate the attention weights of n single-head attention layers and then apply a nonlinear transformation with a dense layer. Having the output of n single-head layers allows the encoding of multiple independent single-head layer transformations into the model. Therefore, the model can focus on multiple time series steps at once. Increasing the number of attention heads affects the ability of a model to capture long-distance dependencies positively [52].

3. Results and Discussions

We use balanced accuracy and F1 score performance metrics to compare the different methods. Balanced accuracy is a raw accuracy, where each sample is weighted according to its actual class's inverse prevalence. It helps us deal with imbalanced datasets by avoiding inflated performance estimates in our datasets. If the classifier performs equally well on

either class, this term reduces to the standard accuracy. In contrast, if the classical accuracy is above chance only because the classifier takes advantage of an imbalanced test set, then the balanced accuracy, as appropriate, will drop to $\frac{1}{n\_classes}$.

$$F1\ Score = \frac{2 * (precision * recall)}{precision + recall} \tag{10}$$

$$Balanced\ Accuracy = \frac{specificity + sensitivity}{2} \tag{11}$$

where $Precision = \frac{TP}{TP+FP}$, $Specificity = \frac{TN}{TN+FP}$, and $Recall(Sensitivity) = \frac{TP}{TP+FN}$. The experiments have been implemented using the PyTorch framework and the labeled datasets used for training, validation, and testing with 50%, 20%, and 30%, respectively. We run the training twenty times to obtain the average results with a learning rate of 0.0001, a batch size of 64, and 1300 epochs.

Furthermore, we conducted five different experiments so that each experiment was designed with specific goals, including comparison with existing fall-detection methods, finding the optimal sensor position for fall detection, and comparison of different techniques to balance the fall datasets. The experiment and results obtained are discussed below.

3.1. Comparing the Different Proposed Data Segmentation Methods

The purpose of this experiment is to evaluate the impact of sliding-window segmentation on deep-learning-based fall-detection approaches. It has not been evaluated how segmentation impacts fall datasets. An inappropriate segmentation method, such as the conventional sliding window, can decrease the classifier's overall performance. We found that using multi-window segmentation increased the F1 score for our dataset. There is a difference in the length of the fall data between the different datasets, and the highest F score can be obtained when the window size is the same as the fall data. When the length of the fall data is known and uniform, selecting the window size is trivial. The real world, however, includes fall data of varying lengths. As this dataset has a varied length of fall data, the results from our dataset, shown in Table 1, can more accurately reflect the classifier's performance in real time. Multi-window segmentation has a better result than the other two segmentation methods.

Table 1. Results for the different segmentation methods.

Segmentation Methods	F1 Score	Balanced Accuracy
Conventional Sliding Window	0.91	0.92
Peak-Detection Window	0.95	0.94
Multi-window Sliding	0.97	0.98

Although conventional sliding windows use overlapping windows, the window signal for a fall event does not always align perfectly with the fall patterns observed in the input signals. As a result, our classifiers would ordinarily produce false negatives and false positives. To solve this problem, we propose more robust data-segmentation methods. When windowing from peak signals, the window is aligned with the fall patterns of the input signals. Nevertheless, capturing the entire fall in one observation window remains difficult. Using multi-windows, we can segment the data according to the class of labels. Multi-windowing segmentation not only provides better results but also helps detect falls on a real-time basis (0.2 s). Because the data are segmented based on the label, we can make the input sequence (observation windows) small.

3.2. Experiment on the Proposed Techniques for Handling the Imbalanced Datasets

This experiment aimed to compare the robustness of the proposed model for imbalanced datasets. We compare different methods for handling imbalanced datasets. The customized loss function and data augmentation improve the specificity and sensitivity of the

model, resulting in a balanced accuracy of 0.92 and 0.98, respectively. However, neither resampling technique reduces the number of false positives nor false negatives. Multi-window segmentation data from the shin-mounted sensor are used to train these balancing techniques. Table 2 shows how all the methods performed in the evaluation set. In contrast to image-classification problems, our results indicate that downsampling and upsampling did not help the sensor signals.

Table 2. Results for various balancing methods of the dataset samples.

Balancing Techniques	F1 Score	Balanced Accuracy
Undersampling	0.86	0.84
Oversampling	0.89	0.87
Data Augmentation	0.91	0.92
Customized Loss Function	0.97	0.98

3.3. Comparison of Fall Detection with Different Sensor Positions

The purpose of this experiment is to demonstrate the capability of the proposed LSTM-based model for predicting the optimal position of the sensors for fall detection based on the observation window generated with the help of multi-window sliding. Therefore, we train a transformer encoder model using fall datasets (accelerometers and gyroscopes) from all 16 sensor positions and evaluate it using the test set. The average performance of the left and right sensor positions of the forearm, upper arm, hand, shank, hip, and foot is used to calculate the performance of the forearm, upper arm, hand, shank, hip, and foot, respectively. Furthermore, the thoracic result is the average of the upper and lower thoracic locations. We measure the model's balanced accuracy and F1 score to evaluate its performance. Based on the LSTM encoder-based model, we present the results for various sensor positions in Table 3. For the shank position, the model achieves a remarkable F1 score of 0.97. In general, the F1 score of the model is higher than 0.90, except for the foot and hand, where the F1 score is 0.88 and 0.85, respectively. The results indicate that different sensor locations give us various performances and that the shank is the optimal location for the sensors.

Table 3. Results for various sensor positions.

Sensor Positions	F1 Score	Balanced Accuracy
Foot	0.88	0.90
Shank	0.97	0.98
Pelvic	0.94	0.92
Hand	0.85	0.84
Forearm	0.93	0.94
Upperarm	0.96	0.96
Head	0.90	0.91
Hip	0.93	0.92
Thoracic	0.94	0.95

3.4. Experiment on Fall Detection with Different Sensor Types

The data obtained from the three-axis accelerometer and gyroscope are combined to obtain information about the global acceleration and orientation of the body, which in turn are inputs into the algorithm. One of the objectives of this study was to examine and compare the relative effectiveness of accelerometer-based algorithms and accelerometer-plus-gyroscope-based algorithms intended for fall detection as part of our comparative study. In the near-fall dataset, we find that the algorithm that uses gyroscope signals has a markedly better performance than the algorithm that does not use gyroscope signals. As shown in Table 4, using a gyroscope in conjunction with an accelerometer as part of the detection process can potentially lead to incremental benefits compared to using an accelerometer for the detection process.

Table 4. Results for the different sensor types.

Balancing Techniques	F1 Score	Balanced Accuracy
Accelerometer	0.90	0.92
Accelerometer + Gyroscope	0.97	0.98

Therefore, to detect falls with low energy consumption, we use acceleration along the x, y, and z axes and angular velocity along the x, y, and z axes. The evidence gathered in this study suggests that assessing wearable sensors located on the shins through acceleration and angular velocity features may represent an optimal combination to discriminate falls from non-falls (including near-falls).

3.5. Comparing Our Proposed Methods to Existing Methods

As a final experiment, we compare the performance of our proposed methods with LSTM and CNN-LSTM [16,23]. This experiment proposed a fall-detection study using competing deep learning methods that use multi-windowing and competing methods of balancing the class data samples. We implemented the existing methods to compare them with our method and evaluate them using our dataset. Table 5 and and the confusion matrix in Figure 4 compare the best results of the LSTM encoder and transformer encoder with those of [16,23]. Compared to [16,23], which reported mean F1 scores of 0.89 and 0.86, respectively, the proposed LSTM encoder performs better with a mean F1 score of 0.97 for shank datasets. Moreover, the proposed transformer encoder method with multi-window segmentation identified falls against non-fall activities with an average F1 score of 0.96 for the shank datasets. These performances show that our proposed approach improves the results by using multi-windowing segmentation and a customized loss function for handling imbalanced datasets while using six features (three acceleration and three angular velocity features).

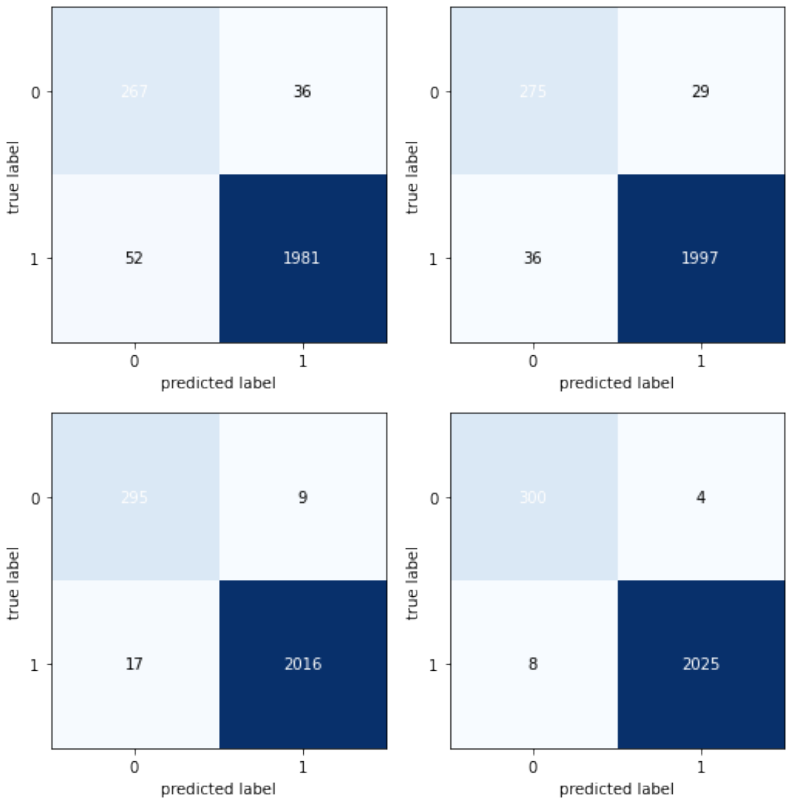

Figure 4. Confusion matrix for the different methods: LSTM [23], CNN-LSTM [16], transformer encoder (our method), and LSTM encoder (our method), from left to right.

Table 5. Comparison of our proposed methods against existing methods.

Balancing Technique	F1 Score	Balanced Accuracy
LSTM [23]	0.86	0.92
CNN-LSTM [16]	0.89	0.94
Our proposed (transformer-based)	0.96	0.97
Our proposed (LSTM-based)	0.97	0.98

Lastly, we visualized the actual and predicted classes of the LSTM-based model for the sample data points. As shown in Figure 5, the visualization shows the data points and their corresponding predicted and actual classes. In the third column of the first row, the ADL, a near-fall activity, is correctly classified as a non-fall activity. This shows that our model is stable even for near-fall events.

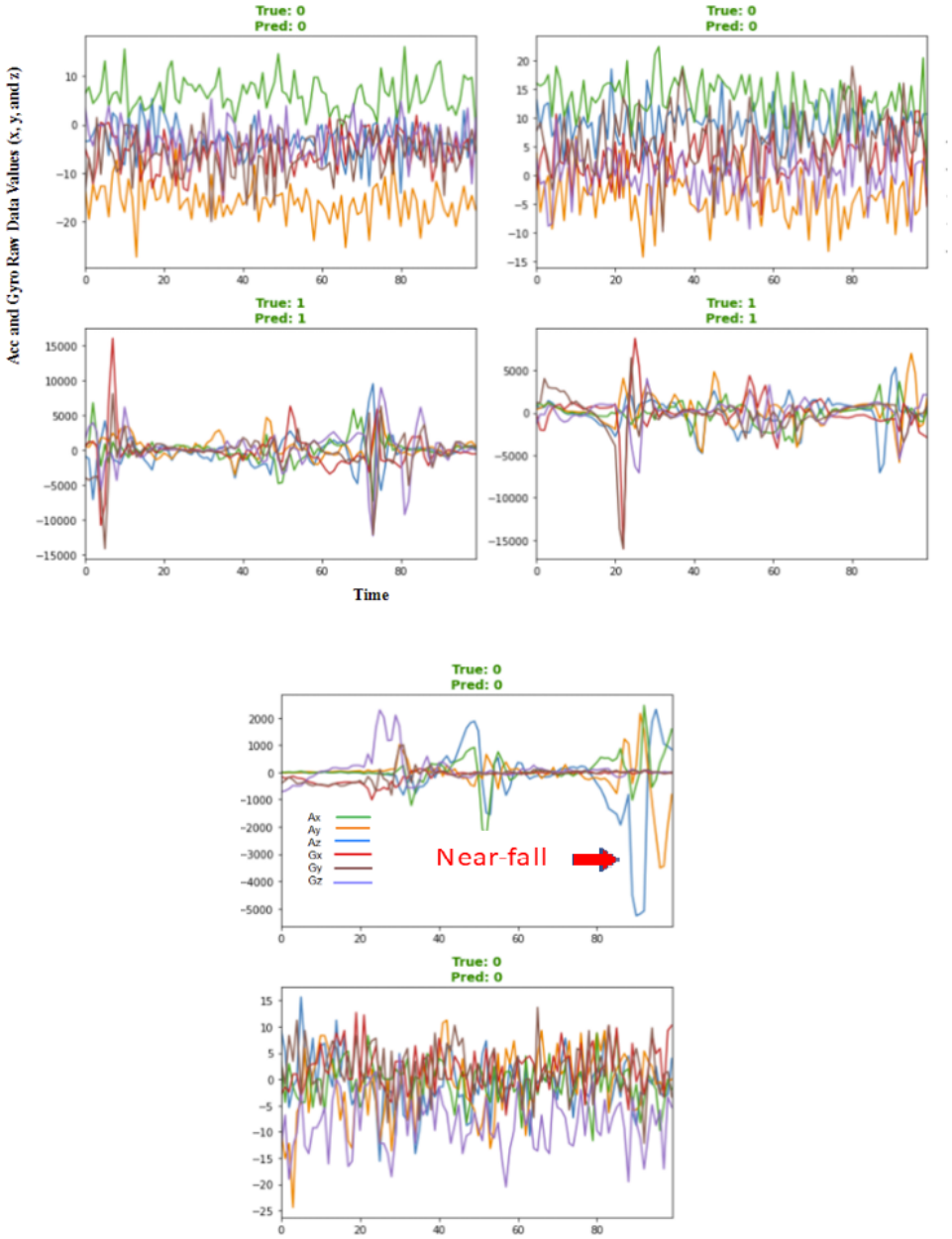

Figure 5. Sample predicted classes using the LSTM encoder-based model. Non-fall activity (class 0), fall activity (class 1), and True, Pred are the actual and predicted classes, respectively.

4. Conclusions and Future Works

In this study, we develop a deep learning method for wearable sensor-based fall detection in a real-time setting. First, we implement and evaluate the impact of different types of sliding-window segmentation on deep-learning-based fall-detection methods. To our knowledge, there are no studies on the evaluation and implementation of different sliding-window segmentation methods other than the conventional sliding window for wearable sensor-based fall detection. Using multi-window segmentation helps us to detect falls in real time (0.2 s). Using a gyroscope in conjunction with an accelerometer as part of the detection process can potentially lead to incremental benefits compared to merely using an accelerometer for the detection process. Another important finding is that the use of convolution-based learned features extracted with different kernels tends to increase the performance of the classifier. Moreover, from the performance presented above, it is better to use the customized loss function than resampling techniques and data augmentation for handling the imbalanced nature of the fall datasets. Balancing the sample dataset using upsampling of the minor class or downsampling of the major class did not help.

Although using wearable sensors has the advantage of flexibility and privacy over camera-based fall detection, camera video recording is still used for manual data annotation. Data labeling for supervised machine learning methods is expensive, as we use video recording and playback of each patient's video to label the data. Future work in this study will employ self-supervised and unsupervised machine learning algorithms. We will create a large database by segmenting the input sequences from different positions with different sensors using multi-window or peak-detection windowing techniques and feed it to a variational autoencoder with LSTM and a transformer to perform unsupervised clustering.

5. Limitations

This work involved collecting and examining fall datasets from multiple subjects. The preliminary analysis conducted on fewer subjects showed a promising performance gain over the previous system. However, testing on additional subjects with various characteristics (different gender or body mass index and senior subjects with dementia, Alzheimer's, and Parkinson's) and different types of accelerometer sensors are required to determine the system's reliability. The information gained from this study would guide us in re-designing the embedded platform for real-time inference, because an immediate and accurate identification of falls is critical for the timely deployment of pneumatic protection actuation. The long-term goal of developing automatic fall-detection systems is to trigger the deployment of injury-mitigation mechanisms, such as an airbag.

Author Contributions: Conceptualization, H.Y. and M.A.; methodology, H.Y.; software, H.Y.; validation, H.Y., M.A. and C.P.; formal analysis, H.Y.; investigation, H.Y.; data curation, C.P.; writing—original draft preparation, H.Y., M.A. and C.P.; writing—review and editing, H.Y., M.A. and C.P.; visualization, H.Y.; supervision and project administration, M.A.; funding acquisition, M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: A sample of the de-identified dataset and supplementary material of the implementation will be available at the GitHub link (https://github.com/HabenGirmayYhdego, accessed on 1 April 2023).

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

References

1. WHO. Falls. 2016. Available online: http://www.who.int/mediacentre/factsheets/fs344/en/ (accessed on 1 December 2022).
2. Morel, B.; Kakara, R.; Henry, A. Trends in Non fatal Falls and Fall-Related Injuries Among Adults Aged ≥65 Years—United States, 2012–2018. *MMWR Morb. Mortal Wkly. Rep.* **2020**, *69*, 875–881.

3. Casilari, E.; Luque, R.; Moron, M.J. Analysis of Android Device-Based Solutions for Fall Detection. *Sensors* **2015**, *15*, 17827–17894. [CrossRef] [PubMed]
4. Figueiredo, I.N.; Leal, C.; Pinto, L.; Bolito, J.; Lemos, A. Exploring smartphone sensors for fall detection. *mUX J. Mob. User Exp.* **2016**, *5*, 1–17. [CrossRef]
5. Montesinos, L.; Castaldo, R.; Pecchia, L. Wearable Inertial Sensors for Fall Risk Assessment and Prediction in Older Adults: A Systematic Review and Meta-Analysis. *IEEE Trans. Neural Syst. Rehabil. Eng.* **2018**, *26*, 573–582. [CrossRef] [PubMed]
6. Klenk, J.; Becker, C.; Lieken, F.; Nicolai, S.; Maetzler, W.; Alt, W.; Zijlstra, W.; Hausdorff, J.; van Lummel, R.; Chiari, L.; et al. Comparison of acceleration signals of simulated and real-world backward falls. *Med. Eng. Phys.* **2011**, *33*, 368–373. [CrossRef] [PubMed]
7. Aimée, K. Bright and Lynne Coventry—Assistive technology for older adults: psychological and socio-emotional design requirements. In Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '13), Rhodes, Greece, 29–31 May 2013.
8. Dai, E.J.; Bai, X.; Yang, Z.; Shen, Z.; Xuan, D. PerFallD: A pervasive fall detection system using mobile phones. In Proceedings of the 2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), Mannheim, Germany, 29 March–2 April 2010.
9. Lee, R.Y.; Carlisle, A.J. Detection of falls using accelerometers and mobile phone technology. *Age Ageing* **2011**, *40*, 690–696. [CrossRef]
10. Viet, V.; Choi, D.-J. Fall Detection with Smart Phone Sensor. In Proceedings of the 3rd International Conference on Internet (ICONI), Sepang, Malaysia, 15–19 December 2011.
11. Kangas, M.; Vikman, I.; Wikl, er J.; Lindgren, P.; Nyberg, L.; Jämsä, T. Sensitivity and specificity of fall detection in people aged 40 years and over. *Gait Posture* **2009**, *29*, 571–574. [CrossRef]
12. Purwar, A.; Jeong, D.U.; Chung, W.Y. Activity monitoring from real-time triaxial accelerometer data using sensor network. In Proceedings of the 2007 International Conference on Control, Automation and Systems, Seoul, Republic of Korea, 17–20 October 2007; pp. 2402–2406. [CrossRef]
13. Özdemir, A.T.; Barshan, B. Detecting Falls with Wearable Sensors Using Machine Learning Techniques. *Sensors* **2014**, *14*, 10691–10708. [CrossRef]
14. Ramachandran, A.; Karuppiah, A. A Survey on Recent Advances in Wearable Fall Detection Systems. *BioMed Res. Int.* **2020**, *2020*, 2167160. [CrossRef]
15. Delgado-Escano, R.; Castro, M.; Coza, J.R.; Guil, M.J.N.; Casilari, E. A cross-dataset deep learning-based classifier for people fall detection and identification. *Comput. Methods Prog. Biomed.* **2020**, *184*, 105265. [CrossRef]
16. Nait Aicha, A.; Englebienne, G.; van Schooten, K.S.; Pijnappels, M.; Kröse, B. Deep Learning to Predict Falls in Older Adults Based on Daily-Life Trunk Accelerometry. *Sensors* **2018**, *18*, 1654. [CrossRef]
17. Yhdego, H.; Li, J.; Paolini, C.; Audette, M. Wearable Sensor Gait Analysis of Fall Detection using Attention Network. In Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Houston, TX, USA, 9–12 December 2021.
18. Putra, I.P.E.S.; Brusey, J.; Gaura, E.; Vesilo, R. An Event-Triggered Machine Learning Approach for Accelerometer-Based Fall Detection. *Sensors* **2018**, *18*, 20. [CrossRef] [PubMed]
19. 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]
20. Medrano, C.; Igual, R.; Plaza, I.; Castro, M. Detecting falls as novelties in acceleration patterns acquired with smartphones. *PLoS ONE* **2014**, *9*, e94811. [CrossRef]
21. Yhdego, H.; Li, J.; Morrison, S.; Audette, M.; Paolini, C.; Sarkar, M.; Okhravi, H. Towards Musculoskeletal Simulation-Aware Fall Injury Mitigation: Transfer Learning with Deep CNN for Fall Detection. In Proceedings of the 2019 Spring Simulation Conference (SpringSim), Tucson, AZ, USA, 29 April–2 May 2019; pp. 1–12.
22. Fakhrulddin, A.H.; Fei, X.; Li, H. Convolutional neural networks (CNN) based human fall detection on body sensor networks (BSN) sensor data. In Proceedings of the International Conference on Systems and Informatics (ICSAI), Hangzhou, China, 11–13 November 2017.
23. Paolini, C.; Soselia, D.; Baweja, H.; Sarkar, M. Optimal Location for Fall Detection Edge Inferencing. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9–13 December 2019; pp. 1–6.
24. Yhdego, H.; Audette, M.; Paolini, C. Fall Detection Using Self-Supervised Pre-Training Model. In Proceedings of the 2022 Annual Modeling and Simulation Conference (ANNSIM), San Diego, CA, USA, 18–20 July 2022; pp. 361–371. [CrossRef]
25. Heo, B.; Yun, S.; Han, D.; Chun, S.; Choe, J.; Oh, S.J. Rethinking Spatial Dimensions of Vision Transformers. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, BC, Canada, 11–17 October 2021; pp. 11936–11945.
26. Yin, H.; Vahdat, A.; Alvarez, J.M.; Mallya, A.; Kautz, J.; Molchanov, P. A-ViT: Adaptive Tokens for Efficient Vision Transformer. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 18–24 June 2022; pp. 10809–10818.
27. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 3–7 May 2021.

28. Tatsunami, Y.; Taki, M. Sequencer: Deep LSTM for Image Classification. In Proceedings of the 2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Houston, TX, USA, 9–12 December 2021.
29. Noraxon USA Inc. MyoRESEARCH 3.14 User Manual. 2019. Available online: https://www.noraxon.com/ (accessed on 1 October 2022).
30. Neuromechanics and Neuroplasticity Laboratory-ENS 216, San Diego State University. 2020. Available online: https://ens.sdsu.edu/dpt/research/faculty-research-interests/neuromechanics-and-neuroplasticity-lab/ (accessed on 1 October 2022).
31. Sucerquia, A.; López, J.D.; Vargas-Bonilla, J.F. SisFall: A Fall and Movement Dataset. *Sensors* **2017**, *17*, 198. [CrossRef]
32. Yang, X.; Xiong, F.; Shao, Y.; Niu, Q. WmFall: WiFi-based multistage fall detection with channel state information. *Int. J. Distrib. Sens. Netw.* **2018**, *14*, 1550147718805718. [CrossRef]
33. Aksoy; Selim; Haralick, R.M. Feature normalization and likelihood-based similarity measures for image retrieval. *Pattern Recognit. Lett.* **2001**, *22*, 563–582. [CrossRef]
34. LeCun, Y.A.; Bottou, L.; Orr, G.B.; Muller, K.R. Efficient BackProp. In *Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science Montavon*; Orr, G., Müller, G.B., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7700.
35. Sergey, I.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv* **2015**, arXiv:1502.03167.
36. Putra, I.P.E.S.; Vesilo, R. Window-size impact on detection rate of wearable-sensor-based fall detection using supervised machine learning. In Proceedings of the 2017 IEEE Life Sciences Conference (LSC), Sydney NSW, Australia, 13–15 December 2017; pp. 21–26.
37. Nitesh, V.; Kevin, W.; Bowyer, L.; Hall, O.; Kegelmeyer. P. SMOTE: Synthetic minority over-sampling technique. *J. Artif. Int. Res.* **2002**, *16*, 321–357.
38. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
39. 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 (ICMI '17), Glasgow, UK, 13–17 November 2017; pp. 216–220.
40. Zhang, H.; Cisse, M.; Dauphin, Y.; Lopez-Paz, D. Mixup: Beyond Empirical Risk Minimization. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
41. Yun, S.; Han, D.; Oh, S.; Chun, S.; Choe, J.; Yoo, Y.J. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 29 October–1 November 2019; pp. 6022–6031.
42. Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. Focal Loss for Dense Object Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **2020**, *42*, 318–327. [CrossRef]
43. Chernbumroong, S.; Cang, S.; Yu, H. Genetic Algorithm-Based Classifiers Fusion for Multisensor Activity Recognition of Elderly People. *IEEE J. Biomed. Health Inform.* **2015**, *19*, 282–289. [CrossRef] [PubMed]
44. Lorincz, K.; Chen, B.R.; Challen, G.W.; Chowdhury, A.R.; Patel, S.; Bonato, P.; Welsh, M. Mercury: A wearable sensor network platform for high-fidelity motion analysis. *SenSys* **2009**, *9*, 183–196.
45. Abbate, S.; Avvenuti, M.; Corsini, P.; Light, J.; Vecchio, A. Monitoring of human movements for fall detection and activities recognition in elderly care using wireless sensor network: A survey. In *Wireless Sensor Networks: Application*; Merrett, G., Tan, Y.K., Eds.; CRC Press: Boca Raton, FL, USA, 2010.
46. Gia, T.N.; Sarker, V.K.; Tcarenko, I.; Rahmani, A.M.; Westerlund, T.; Liljeberg, P.; Tenhunen, H. Energy efficient wearable sensor node for IoT-based fall detection systems. *Microprocess. Microsyst.* **2018**, *56*, 34–46.
47. Cui, Z.; Ke, R.; Pu, Z.; Wang, Y. Deep Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction. *arXiv* **2018**, arXiv:1801.02143.
48. Kim, J.; Moon, N. BiLSTM model based on multivariate time series data in multiple fields for forecasting trading area. *J. Ambient. Intell. Hum. Comput.* **2019**, 1–10. [CrossRef]
49. Graves, A.; Mohamed, A.R.; Hinton, G. Speech recognition with deep recurrent neural networks. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 6645–6649.
50. Kazemi, S.M.; Goel, R.; Eghbali, S.; Ramanan, J.; Sahota, J.; Thakur, S.; Brubaker, M. Time2Vec: Learning a Vector Representation of Time. *arXiv* **2019**, arXiv:1907.05321.
51. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Polosukhin, I. Attention Is All You Need. In Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017.
52. Tang, G.; Müller, M.; Rios, A.; Sennrich, R. Why Self-Attention? A Targeted Evaluation of Neural Machine Translation Architectures. *arXiv* **2018**, arXiv:1808.08946.