Hindawi
Scientific Programming
Volume 2019, Article ID 9135196, 10 pages
https://doi.org/10.1155/2019/9135196

Research Article

Deep Recurrent Neural Networks for Edge Monitoring of Personal Risk and Warning Situations

Emanuele Torti, Mirto Musci, Federico Guareschi, Francesco Leporati, and Marco Piastra

Department of Electrical, Computer and Biomedical Engineering, University of Pavia, Pavia I-27100, Italy

Correspondence should be addressed to Emanuele Torti; emanuele.torti@unipv.it

Received 1 September 2019; Accepted 5 November 2019; Published 13 December 2019

Guest Editor: Daniele D'Agostino

Accidental falls are the main cause of fatal and nonfatal injuries, which typically lead to hospital admissions among elderly people. A wearable system capable of detecting unintentional falls and sending remote notifications will clearly improve the quality of the life of such subjects and also helps to reduce public health costs. In this paper, we describe an edge computing wearable system based on deep learning techniques. In particular, we give special attention to the description of the classification and communication modules, which have been developed by keeping in mind the limits in terms of computational power, memory occupancy, and power consumption of the designed wearable device. The system thus developed is capable of classifying 3D-accelerometer signals in real-time and to issue remote alerts while keeping power consumption low and improving on the present state-of-the-art solutions in the literature.

1. Introduction

Nowadays, unintentional falls are among the principal causes of fatal injuries. Moreover, they are the most common cause of hospitalization after nonfatal traumas. A study conducted by the World Health Organization [1] highlights that 25% of people aged over 65 years old fall every year, with an increment to 32%–42% if considering only over 70. Furthermore, even when the falls lead to less severe injuries, the associated discomfort significantly reduces the quality of life. It is worth noting that not only elderly people are affected by unintentional falls, every person with some kind of fragility related, for example, to postoperative conditions, disability, or any other disease affecting mobility are part of similar statistics. In addition, it is well known that the majority of unintentional falls happens in the home environment.

These facts highlight the importance of an automated system capable of detecting unintentional falls sending remote notifications so that timely help can be given. Among the different approaches described in the literature, embedded wearable devices are emerging as the best choice for this kind of systems. This is mainly due to their low intrusiveness, reduced power consumption, and cost effectiveness. Moreover, the recent innovations on the microcontroller units (MCUs), which are typically used for wearable devices, provide the necessary computational power that enables them to perform complex computations directly on the MCU. This allows to implement more complex methods than in the past on-board wearable devices, thus effectively enabling a very specific kind of edge computing. Among these methods for fall detection, deep learning techniques recently showed to be a very promising approach [2, 3].

In this paper, we describe an embedded framework based on edge computing and on machine learning for fall detection on-board wearable devices. In particular, we extend our system proposed in [4] with a novel strategy to improve the overall system performance together with the design of a suitable Bluetooth Low Energy (BLE) protocol for an effective minimization of data transmission.

It is worth highlighting that the adopted methodology for this personal monitoring system based on deep learning methods is very general and could be reused also for different applications beyond the reported one. As an example, deep learning methods on embedded devices can be used in different fields, such as automotive [5], security and surveillance [6], augmented reality [7], and healthcare [8].

The paper is organized as follows. In Section 2, the state of the art of both wearable devices and fall detection through machine/deep learning techniques is described. Then, the proposed system is presented and the classification and communication modules are described in detail, highlighting the reasons for the different design choices. These modules are then evaluated from different points of view, including computational complexity, power consumption, and memory occupancy in Section 3, evaluating how these modules impact on the performance of the overall system. Section 4 gives some final remarks and possible future research lines.

2. Materials and Methods

2.1. State of the Art. In the fall detection literature, two main trends can be identified: one is based on ambient monitoring, while the other harnesses wearable or portable devices. Ambient monitoring is typically based on cameras mounted in rooms. The main issues of this approach are the expensiveness, the high power consumption, and the intrusiveness of the system in terms of privacy. Portable and wearable devices do not suffer from these limitations. In the latter category, two main approaches can be found: one is typically based on smartphones [9–12] and another on several kinds of wearable devices with specific hardware [13–16]. As fall detection devices, smartphones suffer from several limitations, which affect the overall performance [17]. First, the sensors are shared and managed by the operating system in a pre-emptive fashion. In a smartphone, in fact, several applications are simultaneously executed and this implies that all sensors are shared among all the applications which need sensory data, some of which run at a high priority. Therefore, it is not possible in practice to achieve the guarantee of a fixed sampling frequency and this is a critical issue especially for artificial intelligence methods. Last but not the least, in general, the typical duration of the battery is not compatible with continuous monitoring during the entire day. On the contrary, wearable devices are designed for one specific task only, they have direct access which guarantees a fixed sampling frequency; therefore, battery duration can be made to be longer than with smartphone-based solutions.

A thorough analysis of the existing wearable system solutions for fall detection highlights two main research tracks. In the first track, we find on-board elaboration of sensory readings; typically, the acquired sensors data are filtered and then processed through techniques based on a fixed threshold or via other statistical methods, and then the results are sent to a remote device.

Cola et al. [13] proposed a head-worn device containing an accelerometer and a barometer integrated with a TI MSP430 MCU. Jung et al. [14] described a system including a three-axial accelerometer, an MCU, and a Bluetooth module. These components are attached to a jacket and are connected to each other via stretchable conductive nylon. Nyan et al. [15] proposed a system using accelerometers and gyroscopes. In this work, sensors are connected via Zigbee transceivers to a board based on an Intel PXA255 processor, where the actual processing takes place. In another system [18], a custom processor based on FPGA technology elaborates the data acquired from the accelerometers.

It is worth noting that this kind of elaboration is simple and does not require high-computational capabilities. On the contrary, the accuracy of fixed threshold-based and statistical methods is not so high and is outperformed by artificial intelligence techniques.

On the second track, several works adopted more sophisticated methods, but by abandoning the on-board processing in favor of remote elaboration. In these approaches, the wearable device acquires data from the sensors and then sends them to a workstation that performs the elaboration. A relevant example is the SHIMMER (Sensing Health with Intelligence, Modularity, Mobility, and Experimental Reusability) integrated sensors platform [19]. In [20], the data acquired by the SHIMMER 3D accelerometers is sent via Bluetooth to a remote workstation, which performs the classification through a Support Vector Machine (SVM) classifier. Authors also compared the SVM with K-Nearest Neighbours (KNN) and complex trees. A similar study is conducted in [3], where gyroscopes and accelerometers data are processed by machine learning approaches.

Apart from the SHIMMER, other Commercial Off-the-Shelf (COTS) devices are emerging. In particular, the SensorTile board produced by STMicroelectronics is attracting the researchers because of its computing and memory capability together with low power consumption. The core of this device is the STM32L476JGY MCU with a maximum clock frequency of 80 MHz. The board is also equipped with two three-axial accelerometers, a magnetometer, a barometer, and a gyroscope. It also integrates a Bluetooth 4.1 transceiver, which is a popular protocol for IoT applications.

The mounted MCU is an ARM Cortex M4 core, equipped with a Floating-Point Unit (FPU) which is fully compliant with the IEEE single-precision floating-point standard. The board is also equipped with 1 MB of flash memory and 128 KB of SRAM. This device has been successfully employed in human activity recognition [21] and fall detection [4]. In the above perspective, the fall detection system presented in [4] performs data classification through deep learning methods elaborated on the device. This means that it outperforms the other devices in terms of accuracy because it adopts deep learning methods and reduces power consumption since the elaboration is performed on board, without the need for continuous data transfers. However, this system can be further optimized both from the point of view of computational complexity and power consumption. In this paper, we present a significant improvement of our previous work described in [4]; in particular, the system has been enriched with a pre-elaboration step which prevents the MCU from classifying data that does not contain relevant information. Moreover, a communication protocol based on the BLE standard has been designed in order to minimize the communications between the SensorTile board and a remote host.

2.2. Deep Learning. Deep learning methods are currently the state-of-the-art approach for many computer vision and signal processing problems. In particular, to process time series signals (i.e., data acquired by sensors over time), Recurrent Neural Networks (RNNs) are considered the best solution [22]. Such networks are a specific kind of artificial neural network, in which part of the output is fed back as input.

Generally speaking, an RNN can be described by

$$y^{(t)} = wg\left(Wx^{(t)} + Uh^{(t-1)} + b\right) + c, \qquad (1)$$

where $x^{(t)}$ and $y^{(t)}$ are the input and the output at the time t, respectively, W, U, w, b, and c are the network parameters, and g denotes a nonlinear function. The term $h^{(t)}$ indicates the hidden state, which is defined as follows:

$$h^{(t)} = g\left(Wx^{(t)} + Uh^{(t-1)} + b\right) + c. \qquad (2)$$

The drawback of this kind of network is the training phase, which is very hard to perform both in theory and in practice. The standard training method for RNNs is *temporal unfolding*, where each training input sequence of predefined length is fed as input to the unfolded network. This technique is shown in Figure 1.

To analyze time series (e.g., accelerometers acquisitions over time) the input data stream is scanned through a sliding window of a suitable size, which must match the predefined level of unfolding for training.

Once the training is completed, the obtained RNN can be used to analyze an input stream by sliding a window of size $\omega\omega$ over the input stream, resetting and rerunning the RNN for each input window. This process is known as *inference* and is used to recognize specific patterns in the input. To reduce the computational cost of inference, the input window ω_i is typically slid at interval $s \gg 1$ of constant length called *strides*. All the concepts related to the sliding window technique are shown in Figure 2.

Long Short-Term Memory (LSTM) [23, 24] cells are a particular kind of RNN which are able to detect and reproduce long-term temporal dependencies. They feature the capability to learn how to forget and filter part of their hidden state during the inference. The main advantage is that those networks are easier to train because they do not suffer from the so-called *vanishing gradient* problem. On the contrary, they are more complex than standard RNNs from the computational point of view. The behavior of an LSTM cell is described by the following equations:

$$c_{in}^{(t)} = \tanh\left(W_{xc}x^{(t)} + W_{hc}h^{(t-1)} + b_c\right), \qquad (3)$$

$$i^{(t)} = \text{sigmoid}\left(W_{xi}x^{(t)} + W_{hi}h^{(t-1)} + b_i\right), \qquad (4)$$

$$o^{(t)} = \text{sigmoid}\left(W_{xo}x^{(t)} + W_{ho}h^{(t-1)} + b_o + b_{\text{forget}}\right), \qquad (5)$$

$$f^{(t)} = \text{sigmoid}\left(W_{xf}x^{(t)} + W_{hf}h^{(t-1)} + b_f\right), \qquad (6)$$

$$c^{(t)} = f^{(t)}c^{(t-1)} + i^{(t)}c_{in}^{(t)}, \qquad (7)$$

$$h^{(t)} = o^{(t)}\tanh\left(c^{(t)}\right), \qquad (8)$$

where $W \in \mathbb{R}^{LS\times LS}$, $x^{(t)}, h^{(t)}, c_{in}^{(t)}, i^{(t)}, o^{(t)}, f^{(t)}, c^{(t)}, b \in \mathbb{R}^{LS}$. LS is a hyperparameter, called the *LSTM size*, and is defined upfront by design as constant among all cells.

Figure 3 shows the typical structure of an LSTM cell, where $x^{(t)}$ and $h^{(t-1)}$ are given as input for computing equations (3)–(6), indicated as circles in the figure. The small circles with a point inside indicate the element-wise multiplications needed for preparing the inputs for the evaluation of equations (7) and (8). The result of equation (8) is the output of the cell.

2.3. Fall Detection with Deep Learning for Embedded System. Different deep learning methods have been successfully used for fall detection [2, 25–28]. Analyzing those systems, we see that all these methods rely either on models with a huge number of parameters or on remote communication. In terms of potential criticalities, in the former approach, the set of binary parameters (usually called the *model* of a network) can easily become too heavy to be elaborated in real-time on a wearable device, while in the latter approach, if we consider a 24/7 monitoring, intensive data communication might well drain the battery charge too quickly [29].

The implementation of deep learning methods on an embedded system is a topic of current interest, as witnessed by the development of TensorFlow Lite [30]. This software is a reduced version of the complete TensorFlow software framework and can be executed on mobile and embedded devices. However, this software is still in an early development stage and at present has some limitations: first, only some MCUs are currently supported and, among the low-power microcontrollers, only the ARM Cortex M3 MCU is currently supported. Moreover, the part of the complete TensorFlow framework which is implemented is at present insufficient to implement LSTMs. To the best of the authors' knowledge, RNNs have been successfully deployed to an ARM Cortex M4 MCU only in our previous work [4], where we described a runtime inference module based on an RNN network. In this work, the communication module plus other energy-saving provisions were not discussed and, as we will see here, such improvements can be significantly beneficial to the whole embedded module.

2.4. Architecture of the Proposed Edge Computing System. Overall, the proposed software system can be divided into two main components: the first component is devoted to the offline training of the LSTM network, while the second relates to the real-time classification of sensor readings and the communication to a gateway of relevant events. The training of a deep network directly onboard with a typical MCU like the one used in this project is unfeasible due to

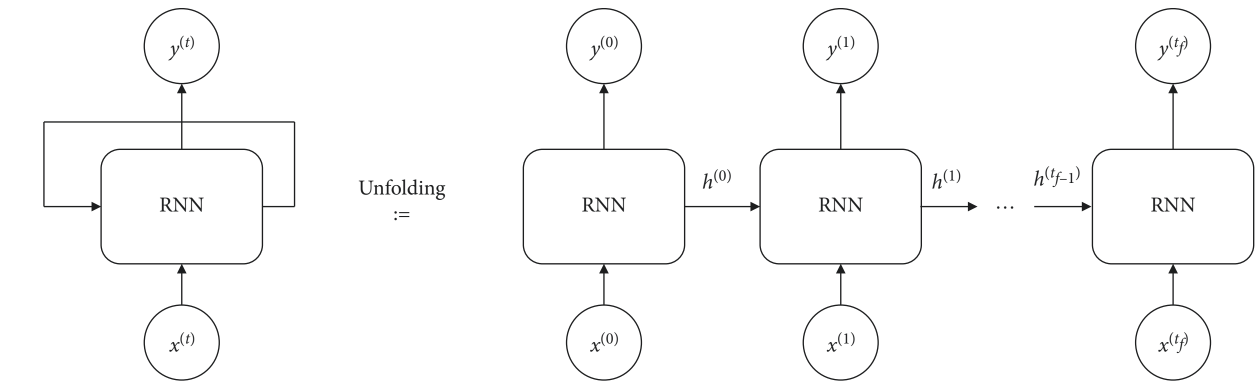

Figure 1: A single RNN cell is unfolded for the training.

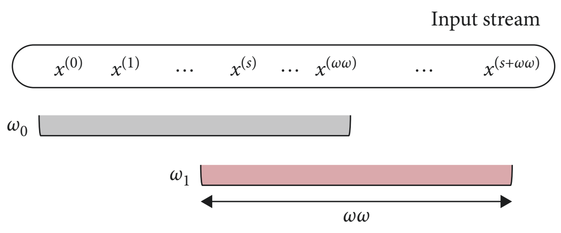
Input stream
$x^{(0)}$ $x^{(1)}$ … $x^{(s)}$ … $x^{(\omega\omega)}$ … $x^{(s+\omega\omega)}$
ω_0
ω_1
$\omega\omega$

Figure 2: An example of sliding window ω_i with width $\omega\omega$ and stride $s < \omega\omega$.

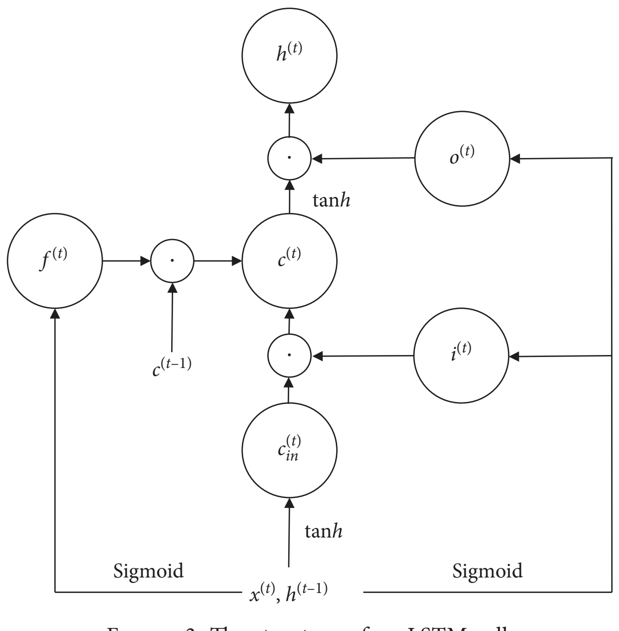

Figure 3: The structure of an LSTM cell.

memory and computational power constraints. Thus, network training must be performed off-board, on a workstation. In the proposed system, the training is performed on a Dell 5810 workstation using TensorFlow 1.8. The training set was collected during an extensive campaign conducted at the University of Pavia in which over 40 volunteers participated in recording sensory data while performing simulated activities and falls, according to a predefined protocol of 17 standard maneuvers. Each recording is associated to a video sequence that describes the performed activity, and annotations are added subsequently by identifying and marking the actual time intervals in which specific events took place [31]. Once trained, the model can be deployed on the MCU.

In the literature, there are several strategies to optimize both memory occupancy and power consumption. One of the most popular techniques is integer quantization. This method requires the definition of a range of values for parameters and variables and uses 8-bit integer encoding for converting back and forth floating-point values into such range. The gain in memory occupancy is clear since every parameter or variable has a footprint four times lower than adopting floating point. However, from the computational point of view, quantization raises also some critical issues. In particular, the LSTM cell requires both linear and nonlinear operations. If the precision loss in linear operations is negligible, the nonlinear operations suffer from a substantial inaccuracy compared to the floating-point counterparts. In addition, depending on the approach adopted, quantization may require several bidirectional conversions between integers and floats, thus making the overall gain in terms of computational efficiency to be clearly assessed.

Another possible strategy which is very popular in custom hardware architectures is the fixed-point representation. In this technique, the data are represented using a subset of the bits of a word for the integer part and the remaining bits for the decimal part. In this way, the basic mathematical operations can be performed using only the integer arithmetic unit of the MCU. However, according to the experiments performed in our laboratory, the precision is again a critical issue because, also in this case, the nonlinear operations suffer from significant precision loss. We also evaluated a hybrid approach when the linear operations are performed in fixed-point format and the nonlinear functions adopt the floating-point format. In this case, a conversion between the two different numeric representations is mandatory. This leads to a significant loss in computational time, which does not allow to achieve the real-time constraint. Moreover, this solution performs even worse than a pure floating-point inference because the adopted MCU is equipped with a floating-point unit, which can perform basic arithmetic operations in only one clock cycle. Therefore, for those reasons, we adopted the single-precision floating-point representation in our runtime module.

Once the LSTM network is deployed on the SensorTile device, the wearable system is ready to act as a wearable, intelligent fall detector. The device should be worn by the monitored subject and, at this point, it begins to acquire sensory data and to classify events. If the classifier detects a fall or a warning situation, the device sends via BLE a message to the gateway, which then forwards it to the cloud for the notification of the alert to the service actors that are designated to intervene.

Both the classifier module and the BLE protocol developed in this work are described in detail in the following sections.

The general architecture of the fall detection system is depicted in Figure 4.

The LSTM training is performed on a workstation using TensorFlow, and the model is then deployed on the MCU by uploading a custom firmware. Events classification is performed online and in real-time and, when dangerous situations are detected, the device issues a BLE message to a gateway that forwards this information to a remote monitor through the cloud.

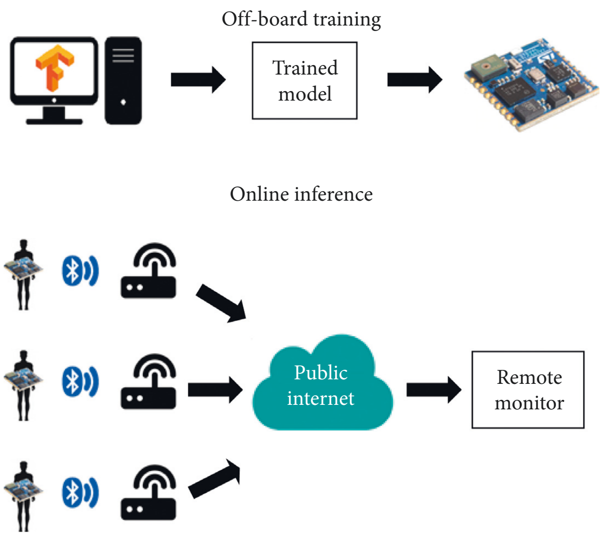

Figure 4: The architecture of the proposed system.

2.5. Inference Runtime Module on the MCU.

As a basis for our classifier module, we adopted the same network architecture described in [4]. Such an architecture includes two LSTM layers, two fully connected layers, and a *softmax* layer. The inner dimension of the LSTM cell is 32. In [4], this network architecture performs the classifying inference in real-time by considering a 1 second window width.

In the work described here, we have improved the overall classifying module by introducing a procedure for the preliminary detection of operating conditions, in particular, whether the device has been worn by the monitored subject, which engages the full classifying network only when sensor readings are compatible with the occurrence of a significant event. We also modified the initialization phase to perform some self-diagnostics and signal to the gateway of hardware malfunctions, if any. In particular, the system can detect hardware malfunctions related to the sensors, i.e., if the chip is not responding to the MCU requests. The flow chart of this extended classifying module is shown in Figure 5.

The first step is the initialization of the accelerometers, the BLE module, and the inference module. The latter module is the most interesting one since it is not a standard software routine. It manages all the memory allocations for the network variables and initializes the weights matrices by loading the values from the flash memory.

During the accelerometer initialization, the routine checks if there are hardware malfunctions and, in this case, issues a BLE message to the gateway.

When all the initializations have been performed correctly, the main loop begins.

At every second, a window containing the accelerometer readings is ready to be processed. The first step is to compute the variance of the accelerations. The ARM Cortex M family can exploit the ARM CMSIS library which includes several floating-point routines among which there is the variance computation [32]. However, the variance is computed there using the standard formula, which is not optimized. For this reason, we implemented the variance computation according to the Welford online algorithm:

$$\begin{aligned} M_{2,n} &= \sum_{i=1}^{n} \left(x_i - \overline{x_n}\right)^2, \\ M_{2,n} &= M_{2,n-1} + \left(x_n - \overline{x_{n-1}}\right)\left(x_n - \overline{x_n}\right), \\ s_n^2 &= \frac{M_{2,n}}{n-1}, \end{aligned} \tag{9}$$

where x_i is the ith sample and $\overline{x_n}$ is the mean after n samples. This algorithm computes the variance inspecting each sample only once, avoiding to loop over the data to compute the mean of the sample window. The obtained variance is compared with two different thresholds. As said before, the first comparison is used to know if the device is worn by the monitored subject (*wear threshold*), while the latter detects if the dynamics of the signal is compatible with the occurrence of interesting events (*classify threshold*). These two thresholds have been experimentally estimated by recording the accelerometer data in different conditions. For the wear threshold, the device has been put on a flat surface in different positions, in order to record the accelerometer outputs at different orientations. After that, we evaluated the accelerometer readings when the device is worn by people performing daily activities such as walking, standing up, and sitting down.

If the variance is lower than the wear threshold, the device sends a BLE message to the gateway in order to signal that the device is not worn by the monitored subject. Otherwise, the second threshold is considered. If the dynamics of the reading is low, a more sophisticated and expensive classification is not necessary, since the occurrence of any relevant events can be simply ruled out. Otherwise, when the sensory dynamics are higher, readings are passed as input to the classification module, which acts as

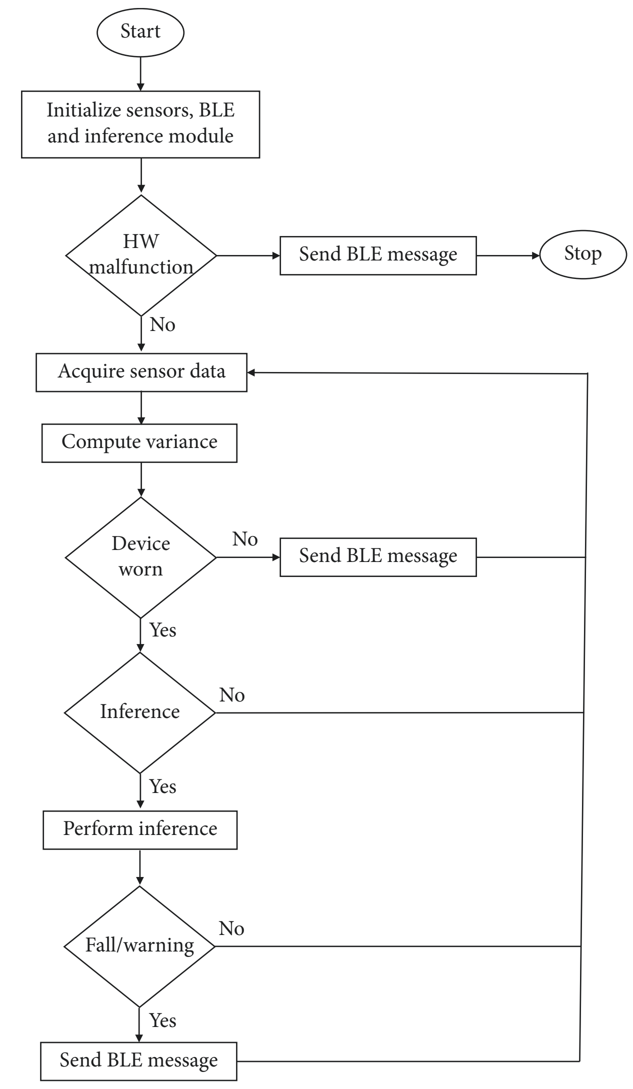

Figure 5: The flow chart of the classification module implemented on the ARM Cortex M4 MCU.

described in [4]. When the classification module detects in turn either a fall or a warning situation, the BLE module sends an alert message to the gateway, which notifies the service actors that are designed to intervene.

The firmware of the wearable device repeats these operations every second until the user switches off the system.

2.6. Bluetooth Low Energy Protocol. The communication between the wearable device and the gateway relies on the BLE protocol. This protocol is designed to transmit data only occasionally. Figure 6 shows the BLE protocol stack [33].

As can be seen from Figure 6, a generic BLE application is made up of three main components: the Application, the Host, and the Controller [34]. The Application is the highest level and contains all the logic and data handling.

The Host consists of the following layers:

(i) Logical Link Control and Adaptation Protocol (L2CAP): it encapsulates data into BLE packets and manages data fragmentation and recombination tasks.

(ii) Attribute Protocol (ATT): it is a simple client/server protocol based on attributes presented by a device. A client requests data from a server, and the server then sends data to its clients.

(iii) Generic Attribute Profile (GATT): it adds a data model and hierarchy defining how data are organized and exchanged between different applications.

(iv) Generic Access Profile (GAP): it controls advertising and connections and specifies how devices perform control procedures such as device discovery, connection, and security levels.

(v) Security Manager Protocol (SMP).

The Controller includes the following layers:

(i) Link Layer (LL): it is in charge of establishing connections and filters out advertising packets depending on the Bluetooth address or based on the data itself

(ii) Physical Layer (PHY): it contains the circuitry to modulate and demodulate analog signals and to convert them into digital signals

The GATT is the most important component to develop in order to design an effective protocol. It is organized in *Services*, each one containing one or more *Characteristics*. The BLE standard defines Services for the most common and general task of an application, such as the *Battery Service*, which includes the *Battery Level* characteristic, containing the charge percentage of the battery. Besides the default services, custom services and characteristics can be added to the GATT. This is of crucial importance for the fall detection system since the actual BLE standard does not include a service related to this task. For this reason, we defined a custom service with specific characteristics. Figure 7 shows a diagram of the GATT services included in our wearable device.

It is worth noticing that, in our wearable system, the BLE standard *Battery Service* coexists with the custom *Fall Service* we defined. The Fall Probability and Warning Probability characteristics are represented as unsigned 8 bit integers since their value ranges from 0 to 100. Also, the *Status* is an unsigned 8 bit integer. In this case, the least significant bit is used to signal if the device is worn or not, while the bit in position 1 is used for alerting in the case of hardware malfunctions. The *Wear Threshold*, *Classify Threshold*, and *Alert Threshold* are characteristics that can be written by the gateway in order to change the value of their respective variables used as a threshold in the inference module. These characteristics are represented as single-precision floating-point numbers. The *Postural Monitoring* is a long characteristic, which can have a maximum size of 512 bytes, according to the BLE standard. It is used to transmit 10 seconds of recording to the gateway, in order to monitor the accelerometer values which are related to the status of the monitored subject after a fall. Even if the BLE standard defines the maximum size of a long characteristic as

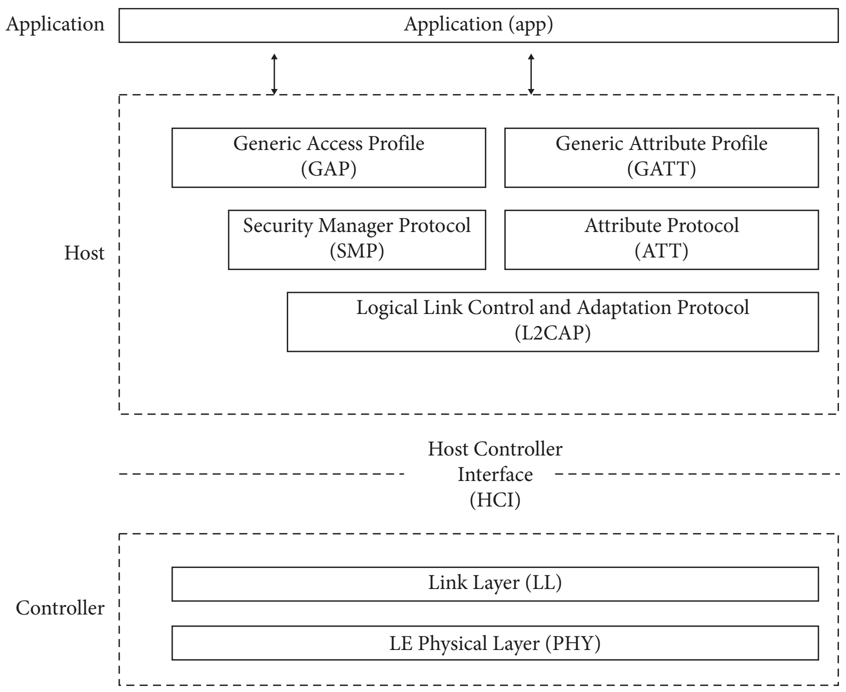
Application
Application (app)
Host
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Security Manager Protocol (SMP)
Attribute Protocol (ATT)
Logical Link Control and Adaptation Protocol (L2CAP)
Host Controller Interface (HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

Figure 6: The BLE stack.

512 bytes, in the SensorTile this limit has been set to 256 bytes by the vendor. The 3D-accelerometer data are sampled at a frequency of 100 Hz; therefore, the total amount of data needed for the postural monitoring is 1200 bytes. Since this value exceeds the size limit of the device, we decided to under-sample the acquired data by a factor of 5, achieving a data size to transmit of 240 bytes, which is compatible with the restriction of our device.

3. Results and Discussion

In an offline computational validation, the embedded component has been tested against the TensorFlow results in order to ensure that the classification module produces the correct outputs. We tested the wearable system classification module by feeding different prerecorded signals sequences to the network as inputs. The results of the embedded classification module differed from those obtained with TensorFlow of about 10^{-7}, which is a negligible error considering that the outputs are probability values ranging from 0 to 1. The classification achieves an accuracy of 90%. The total number of tracks acquired during the campaign performed at the University of Pavia is 18032. About 80% of those tracks have been used to train the system, while the remaining 20% as test set. It is worth noticing that the size of this database is much bigger than the size of other databases in the literature.

To evaluate the impact of the proposed classification module compared to the one described in [4], it is necessary to estimate the computational complexity of the Welford online algorithm, used for the variance estimation, which is the main modification to the runtime module, whereas the comparison between the variance and the threshold can be neglected since their computational weight is not comparable to the whole network complexity. The Welford online algorithm requires $30\omega\omega + 5$ FLOPS, which is significantly lower than the number of FLOPS needed by the data classification complexity reported in [4]. This consideration is confirmed by the experimental data, which highlighted that the differences in the elaboration times are negligible. On the other hand, the gain in computational time is evident when considering that the classification is performed only for certain signal windows. This means that also the power consumption decreases since fewer operations are needed and the MCU can be put into *sleep* mode for a longer time than performing the classification.

In addition, if we consider memory occupancy, the impact of the proposed module is negligible compared to the work in [4]. Indeed, the Welford algorithm and the BLE module require only some scalar variables, except for the postural monitoring, which needs a $3\omega\omega$ long array of float elements, which is significantly smaller than the 82 kB taken by the network parameters.

Considering the BLE communication module, it uses the very low-power BLE single-mode network processor integrated in the SensorTile board. This network processor has a current consumption of about 1.7 μA when the module is active but not transmitting to the gateway. On the other hand, the maximum drained current is about 8.2 mA, which is higher than the current consumption of the MCU when performing the classification (about 5 mA [4]). This is a critical issue because it limits the quantity of data that can be transmitted without draining the battery charge too fast. In fact, continuous data transmission will nearly reduce to one-third of the battery charge duration, since the absorbed current diminishes from 5 mA to about 13 mA. For this reason, the proposed communication protocol has been designed in order to transmit only alerts related to particular

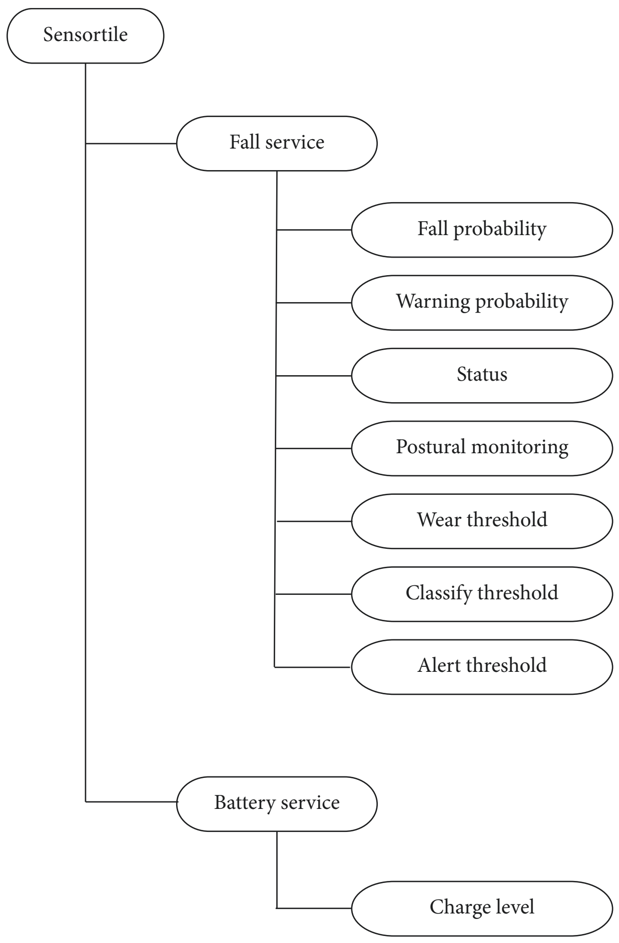

Figure 7: The GATT services and characteristics of our wearable system.

events. Moreover, the number of packets to transmit and receive to/from the gateway is minimal. When considering unsigned 8-bit integers, these data can fit in a single BLE packet, while float data require two BLE packets. The main limit of the proposed protocol is represented by the Postural Monitoring long characteristic, which requires 12 packets to be transmitted to the gateway. For this reason, the Postural Monitoring should be performed only when it is strictly necessary, in order to preserve the battery charge.

It is worth noticing that the transmitted data are related to events that are infrequent, therefore, the BLE radio is inactive for most of the time, keeping the BLU module current consumption negligible.

The improvements with respect to the work presented in [4] are summarized in Table 1.

Table 1 clearly shows that the proposed system improves and outperforms our previous work, both from the computational and power consumption point of view. The table shows both the best case and the worst case analysis of the proposed solution. The first is related to all those situations when the variance is below the inference threshold and therefore it is possible to save computational power by avoiding to evaluate the whole LSTM network. The latter is about data that should be analyzed by the LSTM network,

Table 1: Comparison between [4] and this work.

System	Processing time (ms)	Memory occupancy	Power consumption
[4]	342	82 kB	5 mA for 342 ms
This work best case	0.264	83.2 kB	5 mA for 0.264 ms and 1.7 μA for BLE
This work worst case	~342	83.2 kB	5 mA for 342 ms and 8.2 mA for BLE

and a BLE message is sent to the remote host. It is worth noticing that the best case is the more frequent one, since the variance threshold has been estimated in order to avoid to perform inference on data related to normal daily living activities. Therefore, it is possible to say that the proposed system is operating in the best case conditions for the majority of the time, allowing a significant gain in power consumption. Moreover, this modification requires a negligible memory occupancy increase, as it can be seen from Table 1. Finally, this system is capable of communication with a remote host, through a BLE protocol that has been designed in order to minimize data transfers (again to reduce the impact on the power consumption).

4. Conclusions

In this paper, we described the development of an edge-computing wearable device for personal monitoring exploiting deep learning methods, capable of detecting unintentional falls. In particular, we discussed the optimization of the real-time classification module embedded on the wearable device, together with the developed strategies to avoid unnecessary computations and to reduce power consumption.

Those strategies have been developed after analyzing the data collected during an extensive campaign conducted at the University of Pavia that allowed to carry out one of the biggest databases that can be found in the literature. In particular, the two thresholds used to avoid unnecessary computations have been defined after a careful analysis of this data.

We also described the developed BLE protocol, in order to minimize the communications between device and gateway, enabling a suitable alerting when specific events happen, without affecting the battery charge duration.

These results clearly improve on and further complete our previous system described in [4], enriching it with significant extensions. To the best of the authors' knowledge, this is the first edge computing wearable system for fall detection including such deep learning techniques and with the level of performance obtained.

Future works will be focused on integrating data from different kinds of sensors (i.e., barometers and/or gyroscopes) in order to improve the classification accuracy.

Moreover, the developed prototype could include different hardware architectures, with better potential support for the quantization of both network parameters and nonlinear operation processing. This is a further investigation line to explore for our research.

Data Availability

The labeled Sisfall dataset used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

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

Acknowledgments

This work received financial support from Regione Lombardia under the "Intelligent Personal Health Safety Domotic Monitoring" (IPHSDM) project (ID: 379273).

References

[1] WHO, *WHO Global Report on Falls Prevention in Older Age*, World Health Organization, Geneva, Switzerland, 2008.

[2] F. J. Ordóñez and D. Roggen, "Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition," *Sensors*, vol. 16, no. 1, p. 115, 2016.

[3] M. Hemmatpour, R. Ferrero, B. Montrucchio, and M. Rebaudengo, *A Neural Network Model Based on Co-occurrence Matrix for Fall Prediction*, pp. 241–248, Springer, Cham, Switzerland, 2017.

[4] E. Torti, A. Fontanella, M. Musci et al., "Embedded real-time fall detection with deep learning on wearable devices," in *Proceedings of the 21st Euromicro Conference on Digital System Design, DSD 2018*, pp. 405–412, Prague, Czech Republic, August 2018.

[5] F. Falcini, G. Lami, and A. M. Costanza, "Deep learning in automotive software," *IEEE Software*, vol. 34, no. 3, pp. 56–63, 2017.

[6] G. Sreenu and M. A. Saleem Durai, "Intelligent video surveillance: a review through deep learning techniques for crowd analysis," *Journal of Big Data*, vol. 6, no. 1, 2019.

[7] O. Akgul, H. I. Penekli, and Y. Genc, "Applying deep learning in augmented reality tracking," in *Proceedings of the 12th International Conference on Signal Image Technology and Internet-Based Systems, SITIS 2016*, pp. 47–54, Naples, Italy, December 2017.

[8] P. Teikari, R. P. Najjar, L. Schmetterer, and D. Milea, "Embedded deep learning in ophthalmology: making ophthalmic imaging smarter," *Therapeutic Advances in Ophthalmology*, vol. 11, Article ID 251584141982717, , 2019.

[9] S. Biswas, T. Bhattacharya, and R. Saha, "On fall detection using smartphone sensors," in *Proceedings of the 2018 International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2018*, pp. 1–4, Chennai, India, March 2018.

[10] Y. Lee, H. Yeh, K. H. Kim, and O. Choi, "A real-time fall detection system based on the acceleration sensor of smartphone," *International Journal of Engineering Business Management*, vol. 10, Article ID 184797901775066, , 2018.

[11] J. A. Santoyo-Ramón, E. Casilari, and J. M. Cano-García, "Analysis of a smartphone-based architecture with multiple mobility sensors for fall detection with supervised learning," *Sensors*, vol. 18, no. 4, p. 1155, 2018.

[12] Y. W. Hsu, K. H. Chen, J. J. Yang, and F. S. Jaw, "Smartphone-based fall detection algorithm using feature extraction," in *Proceedings of the 2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics, CISP-BMEI 2016*, pp. 1535–1540, Datong, China, October 2017.

[13] G. Cola, M. Avvenuti, P. Piazza, and A. Vecchio, *Fall Detection Using a Head-Worn Barometer*, Springer, Cham, Switzerland, 2017.

[14] S. Jung, S. Hong, J. Kim et al., "Wearable fall detector using integrated sensors and energy devices," *Scientific Reports*, vol. 5, no. 1, p. 17081, 2015.

[15] M. N. Nyan, F. E. H. Tay, and E. Murugasu, "A wearable system for pre-impact fall detection," *Journal of Biomechanics*, vol. 41, no. 16, pp. 3475–3481, 2008.

[16] D. Giuffrida, G. Benetti, D. De Martini, and T. Facchinetti, "Fall detection with supervised machine learning using wearable sensors," in *Proceedings of the International Conference on Industrial Informatics*, Espoo, Finland, July 2019, In press.

[17] R. Igual, C. Medrano, and I. Plaza, "Challenges, issues and trends in fall detection systems," *BioMedical Engineering OnLine*, vol. 12, no. 1, p. 66, 2013.

[18] S. Abdelhedi, M. Baklouti, R. Bourguiba, and J. Mouine, "Design and implementation of a fall detection system on a Zynq board," in *Proceedings of the 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pp. 1–7, Agadir, Morocco, December 2016.

[19] A. Burns, B. R. Greene, M. J. McGrath et al., "SHIMMER—a wireless sensor platform for noninvasive biomedical research," *IEEE Sensors Journal*, vol. 10, no. 9, pp. 1527–1534, 2010.

[20] F. Hossain, M. L. Ali, M. Z. Islam, and H. Mustafa, "A direction-sensitive fall detection system using single 3D accelerometer and learning classifier," in *Proceedings of the 2016 International Conference on Medical Engineering, Health Informatics and Technology (MediTec)*, pp. 1–6, Dhaka, Bangladesh, December 2016.

[21] A. Nicosia, D. Pau, D. Giacalone, E. Plebani, A. Bosco, and A. Iacchetti, "Efficient light harvesting for accurate neural classification of human activities," in *Proceedings of the 2018 IEEE International Conference on Consumer Electronics, ICCE 2018*, pp. 1–4, 2018.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, USA, 2016.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] F. A. Gers, "Learning to forget: continual prediction with LSTM," in *Proceedings of the 9th International Conference on Artificial Neural Networks: ICANN'99*, pp. 850–855, Edinburgh, UK, September 1999.

[25] T. Mauldin, M. Canby, V. Metsis, A. Ngu, and C. Rivera, "SmartFall: a smartwatch-based fall detection system using deep learning," *Sensors*, vol. 18, no. 10, p. 3363, 2018.

[26] A. Nait Aicha, G. Englebienne, K. van Schooten, M. Pijnappels, and B. Kröse, "Deep learning to predict falls in older adults based on daily-life trunk accelerometry," *Sensors*, vol. 18, no. 5, p. 1654, 2018.

[27] R. Rajagopalan, I. Litvan, T.-P. Jung, R. Rajagopalan, I. Litvan, and T.-P. Jung, "Fall prediction and prevention systems: recent trends, challenges, and future research directions," *Sensors*, vol. 17, no. 11, p. 2509, 2017.

[28] B. Hu, P. C. Dixon, J. V. Jacobs, J. T. Dennerlein, and J. M. Schiffman, "Machine learning algorithms based on signals from a single wearable inertial sensor can detect surface- and age-related differences in walking," *Journal of Biomechanics*, vol. 71, pp. 37–42, 2018.

[29] T. Nguyen Gia, V. K. Sarker, I. Tcarenko et al., "Energy efficient wearable sensor node for IoT-based fall detection systems," *Microprocessors and Microsystems*, vol. 56, pp. 34–46, Feb. 2018.

[30] TensorFlow lite, 2019, https://tensorflow.org/lite/.

[31] M. Musci, D. De Martini, N. Blago, T. Facchinetti, and M. Piastra, "Online fall detection using recurrent neural networks," 2018, https://arxiv.org/abs/1804.04976.

[32] ARM CMSIS Library, 2019, https://developer.arm.com/embedded/cmsis.

[33] L. Leonardi, G. Patti, and L. Lo Bello, "Multi-hop real-time communications over Bluetooth low energy industrial wireless mesh networks," *IEEE Access*, vol. 6, pp. 26505–26519, 2018.

[34] K. Townsend, *Getting Started with Bluetooth Low Energy*, O'Reilly, Newton, MA, USA, 2014.