electronics

Article

Recurrent Neural Network for Human Activity Recognition in Embedded Systems Using PPG and Accelerometer Data

Michele Alessandrini †, Giorgio Biagetti †, Paolo Crippa *,†, Laura Falaschetti † and Claudio Turchetti †

DII—Department of Information Engineering, Università Politecnica delle Marche, I-60131 Ancona, Italy; m.alessandrini@univpm.it (M.A.); g.biagetti@univpm.it (G.B.); l.falaschetti@univpm.it (L.F.); c.turchetti@univpm.it (C.T.)

* Correspondence: p.crippa@univpm.it; Tel.: +39-071-220-4541

† These authors contributed equally to this work.

Abstract: Photoplethysmography (PPG) is a common and practical technique to detect human activity and other physiological parameters and is commonly implemented in wearable devices. However, the PPG signal is often severely corrupted by motion artifacts. The aim of this paper is to address the human activity recognition (HAR) task directly on the device, implementing a recurrent neural network (RNN) in a low cost, low power microcontroller, ensuring the required performance in terms of accuracy and low complexity. To reach this goal, (i) we first develop an RNN, which integrates PPG and tri-axial accelerometer data, where these data can be used to compensate motion artifacts in PPG in order to accurately detect human activity; (ii) then, we port the RNN to an embedded device, Cloud-JAM L4, based on an STM32 microcontroller, optimizing it to maintain an accuracy of over 95% while requiring modest computational power and memory resources. The experimental results show that such a system can be effectively implemented on a constrained-resource system, allowing the design of a fully autonomous wearable embedded system for human activity recognition and logging.

Keywords: recurrent neural network (RNN); deep neural network (DNN); photoplethysmography (PPG); accelerometer; human activity recognition (HAR); embedded system; STM32

Citation: Alessandrini, M.; Biagetti, G.; Crippa, P.; Falaschetti, L.; Turchetti, C. Recurrent Neural Network for Human Activity Recognition in Embedded Systems Using PPG and Accelerometer Data. *Electronics* **2021**, *10*, 1715. https://doi.org/10.3390/electronics10141715

Academic Editor: Dah-Jye Lee

Received: 17 June 2021
Accepted: 15 July 2021
Published: 17 July 2021

1. Introduction

Human activity recognition (HAR) using wearable sensors, i.e., devices directly positioned on the human body, is one of the most popular research areas, which focuses on automatically detecting what a particular human user is doing based on sensor data.

HAR is at the core of a wide variety of applications, such as smart homes [1,2], health care [3–7], surveillance [8,9], skill assessment [10], and industrial settings [11], to cite just a few.

To this end, photoplethysmography (PPG) is an optical technique commonly employed in wearables and other medical devices to measure the change in the volume of blood in the microvascular tissue. Light is emitted from a dedicated device and then reflected and absorbed at different rates during the cardiac cycle. The reflected light is read by a photo-sensor to detect those changes. The output from this sensor can then be processed to obtain a valid heart rate (HR) estimation. Being that PPG is a noninvasive method for HR estimation with respect to electrocardiography (ECG) and surface electromyography, requiring simpler body contact at peripheral sites on the body, such sensors are being more and more used in wearable devices, such as smart watches, as the preferred modality for HR monitoring in everyday activities.

However, accurate estimation of the PPG signal recorded from the subject's wrist when the subject is performing various physical exercises is often a challenging problem, as the raw PPG signal is severely corrupted by motion artifacts (MAs). These are principally due to the relative movement between the PPG light source/detector and the wrist skin of the subject during motion. In order to reduce the MAs, a number of signal processing techniques based on data derived from different sensor types, especially accelerometer data, have proven to be very useful [12–14].

Among smartphones and smart watches, built-in triaxial accelerometers are probably the most widespread sensors that can be used for activity monitoring. Because smartphones and smart watches have become very popular, the data-fusion techniques of PPG and acceleration data can be used for providing accurate and reliable information on human activity directly on such devices [15,16]. PPG sensors alone are not usually applied in HAR classification since they are not designed to capture motion signals as opposed to inertial measurement units (IMU), typically comprising accelerometers and gyroscopes. However, using a PPG sensor for HAR presents several advantages [17]: (i) wearable devices are becoming ubiquitous and almost always embed a PPG sensor, so it makes sense to exploit the information that it can provide, as it comes at no additional cost to the user of one of these PPG-enabled smartwatches or wristbands; (ii) the PPG sensor can either be used alone when other HAR sensors are unavailable, or combined with them to augment recognition performance; and (iii) this sensor can be used to monitor different physiologic parameters (heart rate, blood volume, etc.) in one solution. For these reasons, we chose to also employ the PPG signal to predict human activities.

HAR can be treated as a pattern recognition problem, and in this context, machine learning techniques have proven particularly successful. Due to recent advancements of deep learning techniques, these methods can be categorized in two main approaches: (i) conventional machine learning techniques, and (ii) deep learning-based techniques. In the first category, various machine learning methods, such as *k*-Nearest Neighbors (*k*-NN), Support Vector Machines (SVM), Gaussian Mixture Models (GMM), Hidden Markov Models (HMM) [18], Random Forests (RF) [19], and Molecular Complex Detection methods (MCODE) [20], have been adopted.

The second category, i.e., deep learning-based techniques, includes Deep Neural Networks (DNNs) [21,22], Convolutional Neural Networks (CNNs) [23,24], Autoencoders [25,26], and Recurrent Neural Networks (RNNs) [27–35].

Furthermore, recent advancements in machine learning algorithms and portable device hardware could pave the way for the simplification of wearables, allowing the implementation of deep learning algorithms directly on embedded devices based on micro-controllers (MCUs) with limited computational power and very low energy consumption, without the need for transferring data to a more powerful computer to be elaborated [36,37].

In recent years, edge computing has emerged to reduce communication latency, network traffic, communication cost, and privacy concerns. Edge devices are resource-constrained devices and cannot support high computation loads. As previously mentioned, in the literature, various machine learning methods and DNN models have been developed for HAR. Particularly, deep learning algorithms have shown high performance in HAR, but these algorithms require high computation, making them inefficient to be deployed on edge devices. To our knowledge, there are still few works that have addressed this problem specifically for the HAR classification task [36,37], as most have tested the DNN architectures on high performance processor units [17,28,38–40].

Thus, the main goal of this paper is to prove that the proposed RNN can be implemented in a low cost, low power core, while preserving good performance in terms of accuracy. To reach this goal, we proceed as follows:

- We design an RNN using PPG and triaxial accelerometer data in order to detect human activity, using a publicly available data set for its design and testing. The design and hyper-parameter optimization is performed on a computer architecture.
- After the RNN has been designed, we investigate the porting and performance of the network on an embedded device, namely the STM32 microcontroller architecture from ST, using their "STM32Cube.AI" software solution [41]. This framework allows the porting of a pre-built DNN, converting it to optimized code to be run on the constrained hardware platform.

- When porting the RNN to the embedded system, we show how the network can be simplified to better fit the microcontroller limited resources. In particular, it is demonstrated that the input data can be downsampled to a significant degree, while maintaining good accuracy and requiring fewer hardware resources in order to be implemented.

The rest of the paper is organized as follows. In Section 2, we summarize the related work and we state the motivations for our work. Section 3 summarizes the basic concepts of the RNNs. Section 4 describes the data set adopted in the experiments and the pre-processing data applied to improve the RNN performance. Section 5 reports the details of the proposed RNN architecture with a description of the main features, hardware and software to implement this network in the low-power, low-cost Cloud-JAM L4 board (STM32L476RG microcontroller). Finally, the experimental results are presented in Section 6.

2. Related Work

Nowadays, deep learning techniques have brought great improvements in signal recognition/classification and object detection.

In References [42,43], an automatic target detection and recognition in the infrared images based on a CNN is studied.

In Reference [44], a robust multi-camera multi-player tracking framework is presented. In this system, the player identity, which is commonly ignored in existing methods, is specifically considered, using a deep player identification model for players' identification, 2D localization and segmentation based on a Cascade Mask R-CNN model.

In this section, we provide a summary of the most recent deep learning techniques adopted for classification of PPG signal.

Heart Rate Variability (HRV) is the continuous fluctuation of period length between cardiac cycles, which can be used for the diagnosis of cardiovascular diseases, such as myocardial infarction and cardiac arrhythmia. In Reference [45], an RNN based on bidirectional long short-term memory (biLSTM) is introduced for accurate PPG cardiac period segmentation to derive three important indexes for HRV estimation.

biLSTM is an improved version of long short-term memory (LSTM), which receives forward and backward feature inputs in order to gain information behind and ahead of a specific sample point.

In the study [46], a new hybrid prediction model is proposed by combining ECG and PPG signals with an RNN to estimate blood pressure continually within the RNN structure; a biLSTM is used as the input hidden layer to look for contextual features both forward and backward, while a rectified linear unit (ReLU) layer is selected as the last hidden layer.

In Reference [47], different CNN architectures for PPG-based heart rate estimation are investigated. To train the network, an end-to-end learning approach that takes the time-frequency spectra of synchronized PPG and accelerometer signals as the input and provides the estimated heart rate as the output, is adopted.

A deep learning model for heart rate estimation using a single-channel wrist PPG signal is proposed in [48]. The model contains three components: a CNN, an LSTM, and a fully connected network (FCN).

The input data segmented into eight windows of 1 second duration is passed to the CNN-LSTM feature extractor by performing five parallel convolutions, thereby providing diverse feature representations from the input signal at various receptive fields.

In Reference [17], a novel method is adopted to extract meaningful features from the PPG to predict human activities that combines convolutional and recurrent layers. The convolutional layers are set as feature extractors and provide abstract representations of the three CS, RS and MS data in feature maps, while the recurrent layers model the temporal dynamics of the activation of the feature maps.

3. Brief of RNNs

While traditional neural networks are characterized by the complete connection between adjacent layers, recurrent neural networks (RNNs) can map target vectors from the entire history of the previous map. The structure of an RNN network is shown in Figure 1.

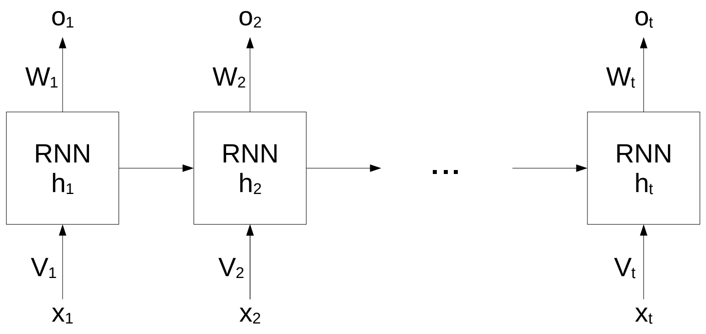

Figure 1. Structure of an RNN.

In this architecture, each node produces a current hidden state h_t and output o_t by using current input x_t and previous hidden state h_{t-1} as follows:

$$h_t = f(W_h h_{t-1} + V_h x_t + b_h) \tag{1}$$

$$o_t = f(W_o h_t + b_o) \tag{2}$$

where W and V are the weights for the hidden layers in recurrent connections, b denotes the bias for hidden and output states, and f is an activation function.

Although an RNN is very effective in modeling the dynamic of a continuous data sequence, it may encounter the problem of gradient disappearance and explosion [49] when modeling long sequences. In order to overcome this issue, Hochreiter et al. [50] propose a variant type of RNN, based on the LSTM, which combines learning with model training without additional domain knowledge. The structure of the LSTM unit is shown in Figure 2.

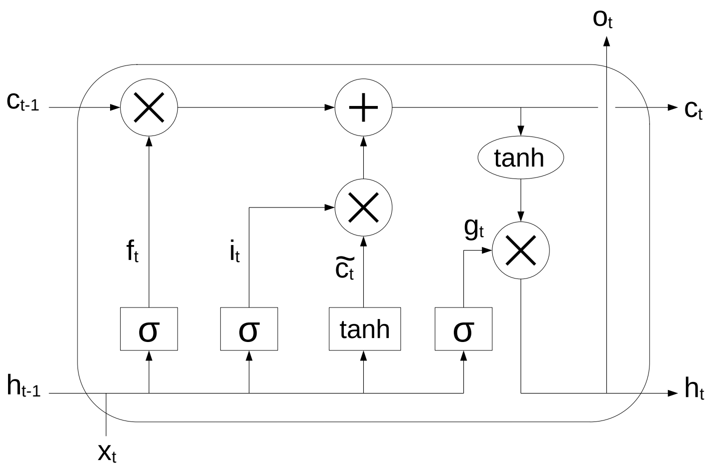

Figure 2. LSTM cell unit.

The following equations show the long-term and the short-term states and the output of each layer at each time step:

$$f_t = \sigma(W_x^T f \cdot x_t + W_h^T f \cdot h_{t-1} + b_f) \tag{3}$$

$$i_t = \sigma(W_x^T i \cdot x_t + W_h^T i \cdot h_{t-1} + b_i) \tag{4}$$

$$\tilde{c_t} = \tanh(W_x^T c \cdot x_t + W_h^T c \cdot h_{t-1} + b_c) \tag{5}$$

$$g_t = \sigma(W_x^T g \cdot x_t + W_h^T g \cdot h_{t-1} + b_g) \tag{6}$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tilde{c_t} \tag{7}$$

$$o_t, h_t = g_t \otimes \tanh(c_t) \tag{8}$$

where W_{x_f}, W_{x_i}, W_{x_o}, W_{x_g} denote the weight matrices for the corresponding connected input vector, W_{h_f}, W_{h_i}, W_{h_o}, W_{h_g} represent the weight matrices of the short-term state of the previous time step, and b_f, b_i, b_o, b_g are bias terms. Here, the symbol $\otimes$ denotes point-wise multiplication.

4. Data Set

We used a recent data set that is publicly available [51] and includes PPG and tri-axis accelerometer data from seven different subjects performing five series of three different activities (resting, squat, and stepper). The four signals are simultaneously acquired with a sampling frequency of 400 Hz and include a total of 17,201 s of recording data. The seven adult subjects include three males and four females aged between 20 years and 52 years.

The PPG and accelerometer signals were recorded from the wrist during some voluntary activity, using the Maxim Integrated MAXREFDES100 health sensor platform. This platform integrates one biopotential analog front-end solution (MAX30003/MAX30004), one pulse oximeter and heart-rate sensor (MAX30101), two human body temperature sensors (MAX30205), one three-axis accelerometer (LIS2DH), one 3D accelerometer and 3D gyroscope (LSM6DS3), and one absolute barometric pressure sensor (BMP280). Particularly, the PPG signals were acquired at the ADC output of the photodetector with a pulse width of 118 μs, a resolution of 16 bits and a full-scale range of 8192 nA, lighted with the green LED. The three-axis accelerometer signal values correspond to the MEMS output with a 10-bit resolution, left-justified, ±2 g scale and axes oriented as shown in Figure 3, with z pointing toward the experimenter's wrist.

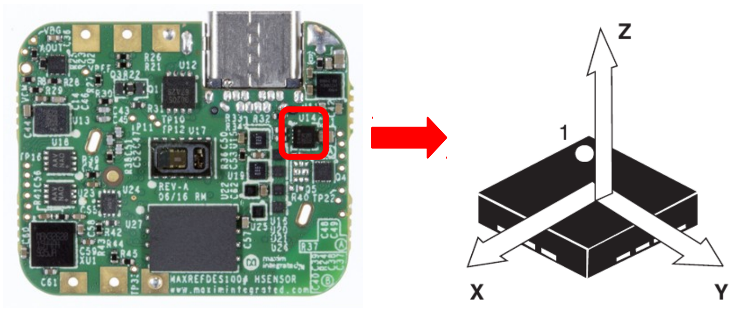

Figure 3. Accelerometer orientation axis.

For the data acquisition, the following measurement set-up was followed as shown in Figure 4: (1) positioning of the sensor directly on the wrist; (2) insertion of the sensor inside a specific weight lifting bracelet, adjustable by a hook-and-loop closure, with optimal elastic characteristics that make it particularly suitable to guarantee perfect adherence of the sensor device to the skin surface; (3) verification of the correct wiring, as the loss of adhesion to the skin-device interface would cause the addition of high frequency noise in the acquired signals, making them unusable; (4) use of the sensor with the cable in "tethered" mode, where the cable comes out from the rear end of the band thus still guaranteeing freedom of movement.

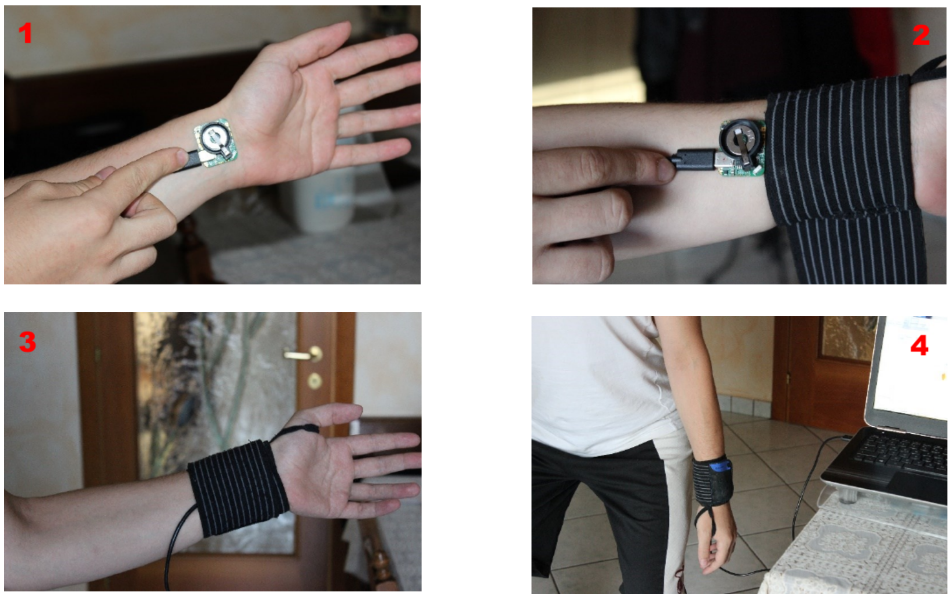

Figure 4. PPG measurement set-up: (**1**) reference point for the positioning of the sensor; (**2**,**3**) insertion of the device inside the appropriate band; (**4**) preparation for a measurement session.

Of the data set, the first five subjects were used for the training phase, while the last two subjects were left for the final testing.

4.1. Data Pre-Processing

The PPG and accelerometer data from every single recording session are combined to obtain a series of four-dimension input data.

A preliminary cleaning of the data is performed for the presence of occasional spikes, including NaN points, probably due to glitches in the communication channel during acquisition. Those are always single points, so they can easily be fixed in software by interpolating the two adjacent points. This cleaning is performed on the five training subjects only, to improve the training process. Data from the two test subjects are left unaltered, to account for transmission errors in real-life applications and to not add overhead to a possible embedded implementation (tests on the computer have shown this to make no difference on the results).

The data are then split in partially overlapping windows of the same size. The choice of window size and overlapping is explained in detail in Section 4.3.

Before feeding the neural network with the resulting inputs, preliminary tests have shown that some basic normalization of data is needed for PPG to achieve acceptable results. It has been already mentioned that it is extremely sensitive to movement. As an example, Figure 5 shows PPG data from a single subject performing five series of the same exercise. It can be seen that the signal varies greatly not only between series, but also in the short term during the same recording.

To better isolate the PPG signal trend from the motion artifacts, we apply statistical standardization to the data, that is, we scale the data so that the resulting mean and standard deviation are 0 and 1, respectively, according to the following formula:

$$\mathrm{PPG}_{std} = \frac{\mathrm{PPG} - \mu}{\sigma} \tag{9}$$

with μ and σ being the original mean and standard deviation, respectively.

In order to ensure that the data can be processed in real time when porting the RNN to the embedded system, μ and σ are computed independently for each window of the incoming data, and so is the standardized signal. Standardization thus transforms each input signal window into another vector of the same length but with predefined mean and variance. Moreover, per-window standardization has the added benefit of also compensating somewhat rapid signal variations between windows. The results of this per-window standardization are still shown in the same Figure 5, where the right panel shows standardized data for 1200 sample windows with no overlapping. The non-overlapping output windows are simply juxtaposed on the graph for ease of representation.

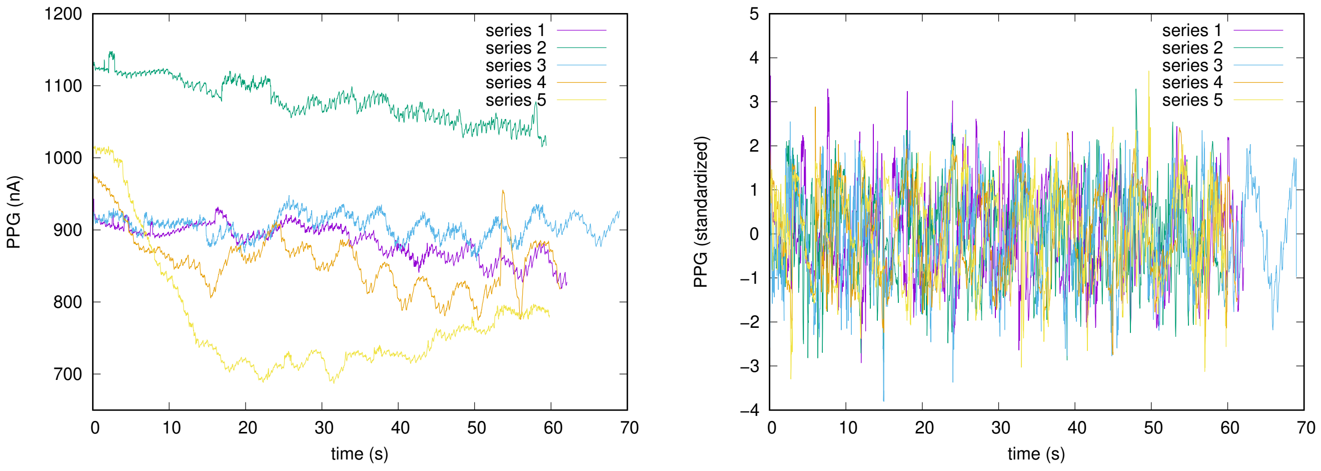

Figure 5. PPG data from subject 1, recorded during the squat activities, as originally acquired (**left panel**) and after standardization on non-overlapping windows of 1200 samples (**right panel**).

On the other hand, accelerometer signals are more regular than PPG, suffering only from low-magnitude noise, which is intrinsic in accelerometers. Figure 6 shows, as a matter of example, the accelerometer data from the recordings of a single subject in one activity. The only issue that must be addressed is that the data generally have a fixed offset, approximately constant, due to projection of gravity acceleration across the three spatial axes. Being that this offset is practically random for the purpose of data analysis, we remove it by subtracting the mean value from the data:

$$ACC_0 = ACC - \mu \tag{10}$$

with μ being the original mean.

Moreover, as can be seen in the same figure, the offset can change abruptly during the same exercise, due to the subject unconsciously changing position. So again, we choose to subtract the mean value in single data windows, individually. The resulting processed signals for the same data are always shown in the right panel of the same Figure 6. While this procedure may not be optimal, for the few data windows crossing an offset change, nevertheless, it is computationally lightweight so as to be implemented in real time in an embedded system and, as the figure shows, it results in a good filtering of the signal.

Preliminary tests have shown that normalization of acceleration values according to their standard deviation has a negative effect on final accuracy, with the normalization layer of the RNN itself leading to better results.

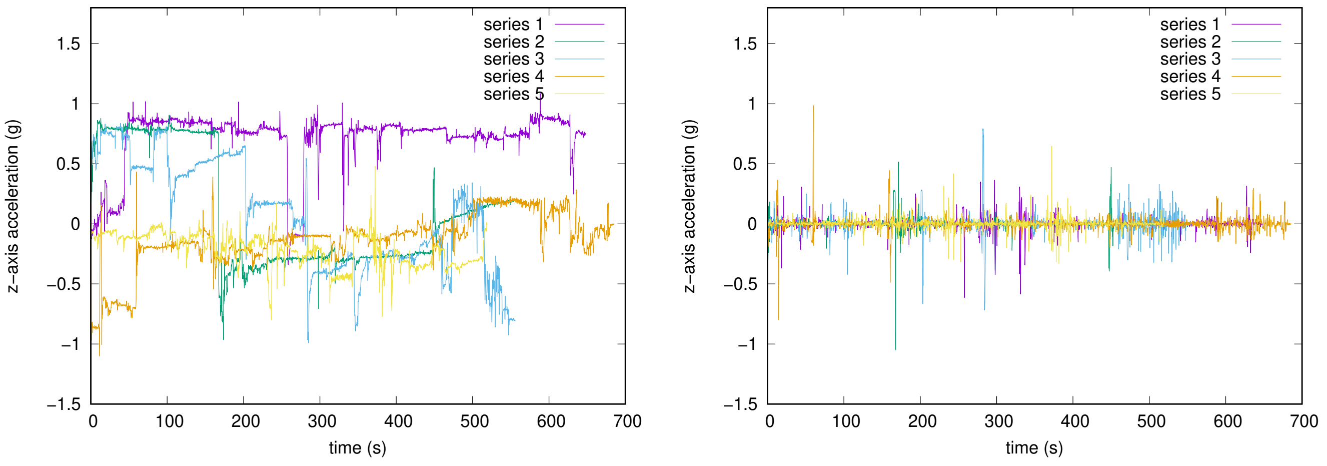

Figure 6. *Z*-axis acceleration data from subject 2, recorded while resting. Original data (**left panel**), and after mean-value subtraction (**right panel**), using non-overlapping windows of 1200 samples.

4.2. Data Downsampling

The original sample rate of the data (400 Hz) can impose a significant load on the processor and memory of an embedded device. Moreover, previous works show that classification of human activity does not require high sample rates [52].

For this reason, a crucial part of the work is examining varying degrees of downsampling of the original signals to find an optimal combination of accuracy and performance on constrained hardware platforms.

To efficiently downsample data, we chose not to use resampling algorithms that require digital filters, which would add significant computational cost when implemented in the final embedded system. We instead used a simple decimation procedure in which 1 out of M samples are retained, discarding the rest. This leads to sample rates corresponding to integer decimation factors only. Mathematically, this is equivalent to transforming the original signal $x[n]$ to a new signal $y[n]$, such as the following:

$$y[n] = x[Mn] \tag{11}$$

with M being the decimation factor.

A new RNN must be built and trained for every sample rate because the size of the network layers depend on the size of input data windows.

In the rest of the paper, when talking about the number of samples in data windows, we will always refer to the samples before downsampling in order to avoid confusion.

4.3. Data Windowing

The window length and overlapping are important hyper-parameters in neural networks, as well as other machine learning algorithms [53,54]. Being that w is the number of samples in a window and o is the number of overlapping samples between adjacent windows, the n-th data window corresponds to samples in the following range:

$$[n(w - o), n(w - o) + w - 1] \tag{12}$$

with $n >= 0$.

To find the best combination for our particular network, we conducted a series of tests with various values of the two parameters. It is common practice, when training a neural network, to further split the training data in two sets: data actually used to fit the network weights, and validation data to monitor the performance of the network during the various training epochs.

Since the number of different subjects in the data set is small and different subjects inevitably have substantial differences in their data, the statistical distribution of the data might not be uniform enough, and so choosing a single partition of training and validation data might not lead to representative results. So, we decided to adopt a cross-validation strategy, that is, for every window length and overlapping combination we trained five networks, isolating each time a different subject for the validation (the network architecture is explained in detail in Section 5). The resulting accuracy of every combination was then computed as the average of the maximum accuracy obtained for the validation data in every test during the training epochs.

Being that this process is quite time-consuming, we examined a limited combination of parameters in the neighborhood of what was already tested in [53], and with a downsampling factor of 10. As can be seen in Table 1, the best accuracy was reached with a window of 1200 samples (before downsampling), corresponding to 3 seconds and 50% overlapping.

Table 1. Validation set accuracy for different window lengths and overlapping, best result displayed in bold.

	Overlap	
Window	**25%**	**50%**
1000	89.95%	90.15%
1100	90.07%	91.70%
1200	91.25%	**91.79%**
1300	91.30%	91.61%
1400	89.06%	90.52%
1500	89.86%	90.16%

The final network used in the rest of the article was trained with the mentioned windowing parameters, and using all the 5 training subjects (no validation data).

4.4. Data Augmentation

Since the number of inputs belonging to the three different activities are not equally represented, the network might end up being biased towards a specific class. A simple technique to address this problem is oversampling [55], a form of data augmentation where the data from classes with less occurrences are duplicated as needed, so that the data used for training are more uniformly distributed among the different classes. ("Oversampling" in this context must not be confused with data resampling in time domain, performed independently).

Table 2 shows the number of input windows for the 3 classes, limited to the 5 subjects used for training, before and after data augmentation.

To summarize, Table 3 shows the number of inputs of the 7 subjects, before and after the oversampling applied to the first 5 ones. Oversampled data were used to train the final network.

Table 2. Number of data windows for the 3 classes, limited to the first 5 subjects.

Activity	**Original**	**Oversampled**
resting	6871	6871
squat	773	6388
stepper	1045	6473

Table 3. Number of data windows for the 7 subjects (oversampling applied to the first 5 subjects only).

Subject	**Original**	**Oversampled**
1	2663	6197
2	2364	5693
3	1193	2469
4	1205	2559
5	1264	2814
6	1288	1288
7	1333	1333
total	11,310	22,353

5. Rnn Architecture

The RNN used in this paper is depicted in Figure 7. It is based on an architecture commonly used with time-based sensor data [54–56] and consisting of a combination of fully connected layers and LSTM cells.

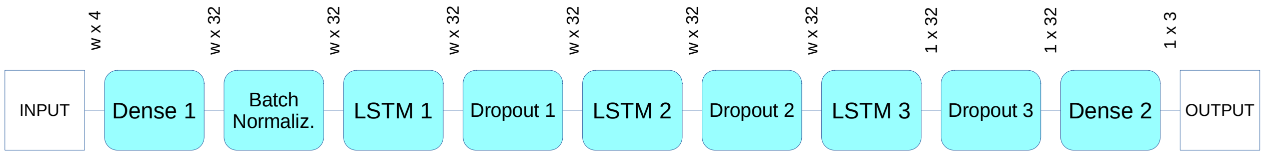

Figure 7. RNN architecture.

Input data are assembled from PPG and the three acceleration axes, resulting in four-dimension time-series. Data are then fed to the network in windows of size $w \times 4$, with the parameter w being the size in time points of a single data window as described in Section 4.3.

The first layer is a fully-connected one (dense), with the purpose of identifying relevant features in the input data. In this layer, the generic n-th neuron produces an output value y_n, according to the $x_1, \ldots x_m$ inputs to the layer and the w_{nj} neuron weights associated to every input. Specifically,

$$y_n = \phi\left(\sum_{j=1}^{m} w_{nj}x_j + b_n\right) \tag{13}$$

where ϕ is an activation function and b_n is a bias value.

Next, there is a batch normalization layer, which normalizes the mean and standard deviation of the data globally, operating on single batches of data as the training progresses. For every input data batch x, its output is the following:

$$y = \gamma \cdot \frac{x - \overline{x}}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{14}$$

where $\overline{x}$ and σ^2 are the mean value and variance of the data batch, respectively, and γ, ϵ, β are internal trainable parameters of the layer.

The core of the recurrent neural network, then, is represented by three cascaded LSTM layers, whose internal architecture was briefly explained in Section 3. Each one is followed by a dropout layer that randomly discards a part of the input to reduce overfitting.

Finally, there is a fully-connected layer of size 3 that, together with the Sparse Categorical Cross-entropy loss function assigned to the network, performs the classification in one of the three classes. The loss function, or cost function in more general terms of optimization problems, represents the error that must be minimized by the training process. The specific representation of the error depends on the particular function assigned to the network. For the Categorical Cross-entropy function, the error is as follows:

$$J(w) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{15}$$

where w is the set of model parameters, e.g., the weights of the RNN, N is the number of input test features, y_i and $\hat{y}_i$ are the true and predicted classes respectively, expressed numerically.

The intermediate layers have size 32; this hyper-parameter was determined experimentally, starting with a larger value and decreasing it until the accuracy varied significantly.

Table 4 shows the details of the individual layers. The RNN, as built in this configuration, has 25,283 trainable parameters.

Table 4. Details of RNN layers.

Layer	Output Size	Parameters
Dense 1	$(-, w, 32)$	160
Batch Norm.	$(-, w, 32)$	128
LSTM 1	$(-, w, 32)$	8320
Dropout 1	$(-, w, 32)$	0
LSTM 2	$(-, w, 32)$	8320
Dropout 2	$(-, w, 32)$	0
LSTM 3	$(-, 32)$	8320
Dropout 3	$(-, 32)$	0
Dense 2	$(-, 3)$	99

5.1. Hardware and Software

For the first part of the design and hyper-parameter optimization, the RNN was developed with TensorFlow 2.4.1 and Keras 2.4.0. The network and the related algorithms were initially developed on the Google Colaboratory platform; then, the final computations were performed on a computer with an Intel Core i7-6800K CPU, 32 GiB of RAM and an NVIDIA GeForce GTX 1080 GPU.

For the embedded part, we tested the RNN on a Cloud-JAM L4 board (https://www.rushup.tech/jaml4, accessed on 1 June 2021), which, for its small form factor and integrated Wi-Fi, can represent a valid prototyping base for a wearable system. While it features a set of inertial and environmental sensors, it is not a complete system with PPG sensor and other needed features. Nevertheless it allows testing the RNN on a real hardware and evaluating its performance in terms of memory and execution time, should a full-featured monitoring system be designed. The classification of test data is done in real time by providing input data to the board from the test set via a serial interface. This also ensures reproducibility of the results with respect to the other tests.

The board features an STM32L476RG microcontroller (https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-ultra-low-power-mcus/stm32l4-series/stm32l4x6/stm32l476rg.html, accessed on 1 June 2021), with an Arm® 32-bit Cortex®-M4 CPU + FPU, frequency up to 80 MHz, 1 MiB flash memory, 128 KiB RAM (but limited to 96 KiB for practical reasons) and about 3 mA of CPU current consumption at full speed.

The porting of the neural network to the STM32 architecture is made possible by a software framework from ST, named "STM32Cube.AI" [41] (current version 6.0.0), integrated in the STM32Cube IDE. The software is a complete solution to import a Keras (or other) model, analyze it for compatibility and memory requirements, and convert it to an optimized C implementation for the target architecture. The generated network can then be evaluated with test input data, both on the computer and the actual device to obtain various metrics, such as execution time, number of specific hardware operations and accuracy.

All the software developed for this article is publicly available at https://github.com/MAlessandrini-Univpm/rnn-ppg-har, accessed on 1 June 2021, published in July 2021.

6. Experimental Results

The final RNN was tested on both the computer and the MCU, with several decimation factors. For every factor, the network was trained and tested with the following parameters:

- Windows of 1200 samples (before decimation) and 50% overlapping.
- Data augmentation applied.
- Five subjects used for training, with no further split for validation.
- Test performed on the last two subjects, not involved in training.
- A total of 100 training epochs.

In addition to the other hyper-parameters already discussed, the number of epochs was chosen experimentally by examining the training accuracy and loss value during the training stage. Figure 8 shows the progress of accuracy (estimated on the training material itself) and loss with respect to the training epochs for the network with no downsampling (original data at 400 Hz). It can be seen that at about 100 epochs, the values reach convergence.

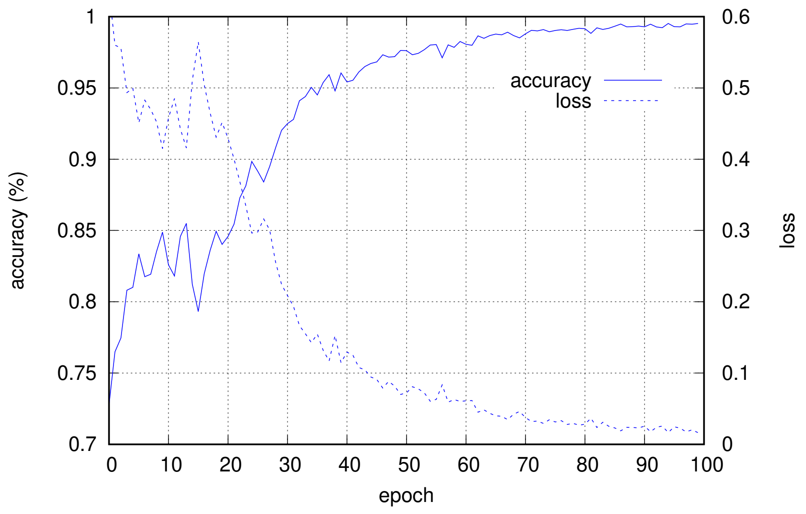

Figure 8. Accuracy and loss progress with respect to training epochs.

Table 5 shows the accuracies and resource usage obtained by the training and the final test for the various RNNs, for both computer and MCU.

On the computer side, reported times are the total time for the training and test stages, respectively.

On the embedded system, every RNN requires a given amount of flash and RAM memory, reported by the framework during the initial analysis. Flash memory requirements do not depend on the sample rate, but only on the network architecture, namely, the quantity of weights and other parameters that are read-only values after the training is done. As shown, the amount of flash memory required is well below the available quantity.

RAM memory, on the other hand, is more limited (96 KiB in this case) and its usage is strongly dependent on the size of input data (and so on the sample rate). Moreover, part of the RAM is needed by the program besides data structures belonging to the RNN. It can be seen that not all the configurations can fit in RAM; combinations that would require more than 100% of RAM could not be executed on the MCU. (An alternative practice to fit a DNN model to a constrained architecture is converting it to TensorFlow Lite format. Unfortunately, the current STM32Cube.AI version—6.0.0—does not support some specific operations generated by the T.F. Lite converter for our model).

Table 5. Experimental results, best results displayed in bold.

		Decimation Factor									
		1	**10**	**20**	**30**	**40**	**50**	**60**	**80**	**100**	**150**
	sample rate (Hz)	400	40	20	13.33	10	8	6.67	5	4	2.67
	window samples	1200	120	60	40	30	24	20	15	12	8
PC	training accuracy (%)	99.52	99.89	99.91	99.93	99.88	99.85	99.78	99.67	99.52	99.10
	test accuracy (%)	94.28	94.96	94.12	93.82	**95.54**	93.97	93.17	92.76	92.45	90.81
	training time (s)	6797	1252	933	810	771	731	701	641	656	625
	total test time (s)	5.10	1.52	1.22	1.19	1.21	1.08	1.06	0.98	1.28	1.01
MCU	test accuracy (%)	-	94.96	94.12	93.82	**95.54**	93.97	93.17	92.76	92.45	90.81
	RAM usage (%)	-	34.4	17.7	12.5	9.4	7.3	6.2	5.2	4.2	3.1
	flash usage (%)	-	9.7	9.7	9.7	9.7	9.7	9.7	9.7	9.7	9.7
	MACC operations (k)	-	3,026	1,513	1,009	757	605	504	378	303	202
	average test time (ms)	-	601.6	301.0	200.2	150.1	120.2	100.0	75.1	60.1	40.2
	CPU usage (%)	-	40.1	20.1	13.4	10.0	8.0	6.7	5.0	4.0	2.7

Timing results are computed by running the RNN on the actual device (see Section 5.1). A dedicated firmware application is provided by the framework; the IDE tool can communicate with such an application on the board, send it the test data to make it run the neural network inference on the hardware and finally, obtain the statistics on performance. Every time a different model is used, a series of operations are needed: generating the code, compiling it, programming it on the MCU flash memory and finally, running the test.

Presumably for a limitation of the firmware validation application, the program stops working if the input and reference data provided are too big, so it was not possible to use the full test data (consisting of more than 2000 rows). A subset of the data (100 rows) had to be used. Since the application reports the average time needed for every inference, the timing results are still meaningful. Indeed, the reported test time for the MCU is the average time of a single data input.

About the accuracy, to have a meaningful comparison with results on the computer using the full test data, we referred to the validation performed by the toolkit on the computer; this uses the same C code generated for the MCU and so it is expected to provide equivalent numerical results.

The CPU percentage usage was computed as the ratio between the average inference time reported by the validation application and the duration of a data window (3 s), multiplied by a factor of two to account for 50% overlapping of the data windows. This parameter can give an estimation of the capability of the embedded system to handle the data classification in real time and the CPU time remaining for other concurrent activities. The table also reports the number of MACC operations, in rounded thousand units, required for a single inference.

It can be seen from the results that the accuracy does not decrease while downsampling the data down to 10 Hz (in fact, it actually increases), corresponding to a CPU usage of 10%, leaving plenty of execution time for other concurrent activities, or alternatively, allowing the reduction of the CPU clock frequency to achieve a lower power consumption. Note that the CPU usage does not include data pre-processing, that is, normalization of the mean value and/or standard deviation (see Section 4.1) that would be needed if data are acquired in real time. Those operations are much simpler than RNN inference, and so should not add a significant overhead.

It can also be seen that the accuracies achieved by the MCU implementation are identical to the ones obtained on the computer. This is presumably due to the differences between the two models being relatively small: apart from the limited precision of the microcontroller FPU (32 bits), the model does not require further compression or quantization to fit on the embedded system.

Figure 9 shows the confusion matrix from the classification of test data in the same setup. It can be seen that the squat and stepper activities are the ones suffering from the larger mistake rates, while the resting activity is recognized correctly in 98% of the cases. This may be due in part to the amount of original input data being substantially less for squat and stepper activities with respect to resting.

In the current setup, the accuracy of the testing stage reaches a maximum of 95.54% for a decimation factor 40. While splitting the data set into five training subjects and two testing subjects is a natural choice, the limited size of the data set can lead to a bias in the results, according to the chosen partition. Moreover, it can be seen from Table 5 that by increasing the decimation factor, the difference between the training and testing accuracies increases.

To test the effect of such a bias, we repeated the previous tests with a leave-one-subject-out, cross-validation strategy. This means testing seven models for every experiment in which six subjects are used for training and one (different each time) for testing. Table 6 shows the test accuracy for this setup, averaged over all the models. Since reducing the test material with respect to training can increase the overfitting effect, we repeated the tests with 50 epochs in addition to 100.

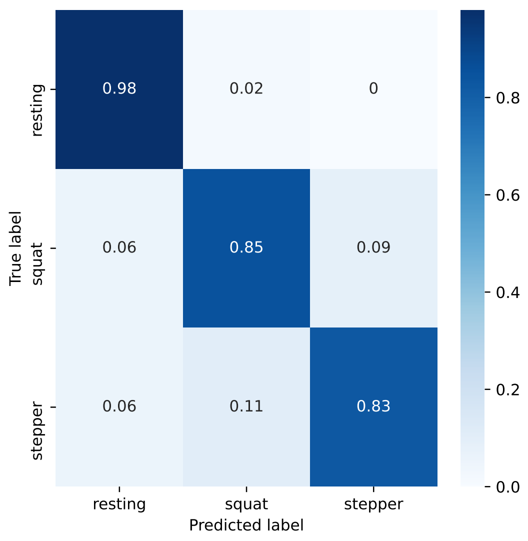

Figure 9. Confusion matrix of test data classification.

Table 6. Testing accuracies of leave-one-subject-out cross-validation.

	Decimation Factor								
Epochs	**10**	**20**	**30**	**40**	**50**	**60**	**80**	**100**	**150**
50	88.71%	88.80%	87.38%	86.95%	86.21%	85.00%	85.13%	84.81%	82.66%
100	88.01%	89.43%	88.13%	88.30%	85.41%	84.91%	86.24%	83.88%	83.29%

It can be seen that in this configuration, the accuracies are significantly lower. Again, this can be explained by the data set being of limited size, and so a single subject may not be representative enough to be used for testing. Indeed, if one better examines a single case, for example, the one at decimation 40 and 100 epochs that results as the best one in Table 5, it can be seen that a few subjects can negatively influence the average results, while most of them have accuracies similar to the better ones reported earlier. This is shown in Table 7.

Table 7. Detail of test accuracy for single subjects in cross-validation, decimation 40, 100 epochs.

Subject	Test Accuracy
1	89.11%
2	72.59%
3	92.54%
4	96.68%
5	90.27%
6	95.19%
7	96.47%

This, again, confirms that the limited size of the data set can limit the generality of the results, producing a strong bias, according to the subject partition. A wider data set could solve those kinds of problems and provide more general results; this can be the subject for future work in this field.

Table 8 reports a list of the state-of-the-art works related to HAR in terms of the employed algorithm, type of signal, data set used for experimentation, number of classes for each data set, hardware used for testing and performance. The commonly used metrics to evaluated the validity of the HAR algorithms are accuracy and F1 score: accuracy is the ratio of the sum of true positives (TP) and true negatives (TN) to the total number of records; the F1 score is an evaluation of the test's accuracy calculated as a weighted average of the precision and recall, where precision is defined as $TP/(TP+FP)$ with FP = false positives, and recall as $TP/(TP+FN)$ with FN = false negatives. By making a comparison with the methods present in Table 8, an evaluation of the contribution of the proposed work can be made. Regarding the data, accelerometer and gyroscope signal sources are the most commonly used in the state of the art since these signals are simple to acquire. So, many works focused on the popular and publicly available UCI HAR data set, which contains six activities (walking, walking upstairs, walking downstairs, sitting, standing, laying down). However, data sets containing PPG signals are relatively less common and more limited in the number of presented activities, but it is still an interesting topic because a PPG sensor is already embedded in smartwatches or wristbands and can either be used alone when other HAR sensors are unavailable, or combined with them to improve recognition performance; moreover, this sensor can be used to monitor different physiologic parameters in one device. Finally, as can be seen, the results obtained with the proposed method are in line with those of the state of the art, especially considering the few works that have experimented the implementation on microcontrollers.

Table 8. Performance of the state-of-the-art HAR methods.

Reference	Employed Algorithm	Signal Source	Dataset	Number of Classes	Implemented in MCU	Hardware for Testing	Performance Metrics
Ordóñez et al. [28]	LSTM	ACC	(1) Opportunity [57] (2) Skoda Mini Checkpoint [58]	(1) 18 (2) 10	–	GPU	(1) F1 score: 86.40% (2) F1 score: 95.80%
Agarwal et al. [40]	RNN/LSTM	ACC	WISDM [59]	6	–	Raspberry Pi 3 (ARM Cortex A53)	Accuracy: 95.78% F1 score: 95.73%
Zhao et al. [38]	LSTM	IMU	(1) UCI HAR [60] (2) Opportunity [57]	(1) 6 (2) 18	–	Intel-i7 CPU, NVIDIA GTX 960M GPU	(1) F1 score: 93.54% (2) F1 score: 90.20%
Novac et al. [37]	(1) CNN (2) MLP	IMU	UCI HAR [60]	6	✓	SparkFun Edge board (ARM Cortex-M4F)	(1) Accuracy: 92.88% (2) Accuracy: 88.94%
Novac et al. [36]	CNN	IMU	UCI HAR [60]	6	✓	(1) Nucleo-L452RE-P (2) SparkFun Edge board (ARM Cortex-M4F)	Accuracy: 92.46%
Mekruksavanich et al. [39]	LSTM	IMU	(1) UCI HAR [60] (2) USC HAD [61]	(1) 6 (2) 12	–	Intel i5-8400 CPU, NVIDIA RTX2070 GPU	(1) Accuracy: 91.23% (2) Accuracy: 85.57%
Boukhechba et al. [17]	CNN+RNN	PPG	custom	5	–	Huawei Watch 2 (smartwatch)	F1 score: 78.00%
Biagetti et al. [62]	KLT+GMM	PPG+ACC	Physionet [63]	4	–	CPU	Accuracy: 78.00%
Biagetti et al. [53]	PBP	PPG+ACC	PPG [51]	3	–	Intel i7 CPU, NVIDIA 1080 GPU	Accuracy: 96.42%
This method	RNN/LSTM	PPG+ACC	PPG [51]	3	✓	STM32L476RG micro (ARM Cortex-M4F)	Accuracy: 95.54%

7. Conclusions

In this paper, an RNN was built for human activity recognition, using PPG and accelerometer data from a publicly available data set. The RNN was then ported to an embedded system based on an STM32 microcontroller, using a specific toolkit for the porting of the network model to the mentioned architecture. The results show that an accuracy of more than 95% is achieved in the classification of test data, and that the sample rate of the acquired data can be downsampled down to 10 Hz, while maintaining the same accuracy. This, in turn, allows the network to be run on the embedded device, using modest hardware resources, paving the way to a fully autonomous activity classifier implemented as a wearable embedded device, using commonly available and cheap microcontrollers.

Author Contributions: Conceptualization, M.A., G.B., P.C., L.F. and C.T.; investigation, M.A., L.F. and C.T.; methodology, M.A., G.B., L.F. and C.T.; project administration, P.C. and C.T.; software,

M.A., G.B. and L.F.; supervision, P.C. and C.T.; validation, M.A. and G.B.; visualization, M.A. and L.F.; writing—original draft, M.A., L.F. and C.T.; writing—review and editing, M.A., G.B., P.C., L.F. and C.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: This work used data publicly available from [51].

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

References

1. Cicirelli, F.; Fortino, G.; Giordano, A.; Guerrieri, A.; Spezzano, G.; Vinci, A. On the design of smart homes: A framework for activity recognition in home environment. *J. Med. Syst.* **2016**, *40*, 1–17. [CrossRef]
2. Rashidi, P.; Cook, D.J. Keeping the resident in the loop: Adapting the smart home to the user. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **2009**, *39*, 949–959. [CrossRef]
3. Boukhechba, M.; Chow, P.; Fua, K.; Teachman, B.A.; Barnes, L.E. Predicting social anxiety from global positioning system traces of college students: Feasibility study. *JMIR Ment. Health* **2018**, *5*, e10101. [CrossRef]
4. Boukhechba, M.; Daros, A.R.; Fua, K.; Chow, P.I.; Teachman, B.A.; Barnes, L.E. DemonicSalmon: Monitoring mental health and social interactions of college students using smartphones. *Smart Health* **2018**, *9*, 192–203. [CrossRef]
5. Patel, S.; Park, H.; Bonato, P.; Chan, L.; Rodgers, M. A review of wearable sensors and systems with application in rehabilitation. *J. Neuroeng. Rehabil.* **2012**, *9*, 1–17. [CrossRef]
6. Avci, A.; Bosch, S.; Marin-Perianu, M.; Marin-Perianu, R.; Havinga, P. Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. In Proceedings of the 23th International Conference on Architecture of Computing Systems 2010, VDE, Hannover, Germany, 22–25 February 2010; pp. 1–10.
7. Mazilu, S.; Blanke, U.; Hardegger, M.; Tröster, G.; Gazit, E.; Hausdorff, J.M. GaitAssist: A daily-life support and training system for parkinson's disease patients with freezing of gait. In Proceedings of the SIGCHI conference on Human Factors in Computing Systems, Toronto, ON, Canada, 26 April–1 May 2014; pp. 2531–2540.
8. Chen, L.; Wei, H.; Ferryman, J. A survey of human motion analysis using depth imagery. *Pattern Recognit. Lett.* **2013**, *34*, 1995–2006. [CrossRef]
9. Taha, A.; Zayed, H.H.; Khalifa, M.; El-Horbaty, E.S.M. Human activity recognition for surveillance applications. In Proceedings of the 7th International Conference on Information Technology, Amman, Jordan, 12–15 May 2015; pp. 577–586.
10. Kranz, M.; Möller, A.; Hammerla, N.; Diewald, S.; Plötz, T.; Olivier, P.; Roalter, L. The mobile fitness coach: Towards individualized skill assessment using personalized mobile devices. *Pervasive Mob. Comput.* **2013**, *9*, 203–215. [CrossRef]
11. Stiefmeier, T.; Roggen, D.; Ogris, G.; Lukowicz, P.; Tröster, G. Wearable activity tracking in car manufacturing. *IEEE Pervasive Comput.* **2008**, *7*, 42–50. [CrossRef]
12. Biagetti, G.; Crippa, P.; Falaschetti, L.; Orcioni, S. Motion Artifact Reduction in Photoplethysmography using Bayesian Classification for Physical Exercise Identification. In Proceedings of the International Conference on Pattern Recognition Applications and Methods, SCITEPRESS 2016, ICPRAM 2016, Rome, Italy, 24–26 February 2016; pp. 467–474.
13. Biagetti, G.; Crippa, P.; Falaschetti, L.; Orcioni, S. Reduced complexity algorithm for heart rate monitoring from PPG signals using automatic activity intensity classifier. *Biomed. Signal Process. Control* **2019**, *52*, 293–301. [CrossRef]
14. Zhang, Z.; Pi, Z.; Liu, B. TROIKA: A General Framework for Heart Rate Monitoring Using Wrist-Type Photoplethysmographic Signals During Intensive Physical Exercise. *IEEE Trans. Biomed. Eng.* **2015**, *62*, 522–531. [CrossRef]
15. Khan, A.M.; Lee, Y.; Lee, S.Y.; Kim, T. Human Activity Recognition via an Accelerometer-Enabled-Smartphone Using Kernel Discriminant Analysis. In Proceedings of the 2010 5th International Conference on Future Information Technology, Busan, Korea, 20–24 May 2010; pp. 1–6.
16. Dernbach, S.; Das, B.; Krishnan, N.C.; Thomas, B.L.; Cook, D.J. Simple and Complex Activity Recognition through Smart Phones. In Proceedings of the 2012 Eighth International Conference on Intelligent Environments, Guanajuato, Mexico, 26–29 June 2012; pp. 214–221.
17. Boukhechba, M.; Cai, L.; Wu, C.; Barnes, L.E. ActiPPG: Using deep neural networks for activity recognition from wrist-worn photoplethysmography (PPG) sensors. *Smart Health* **2019**, *14*, 100082. [CrossRef]
18. Attal, F.; Mohammed, S.; Dedabrishvili, M.; Chamroukhi, F.; Oukhellou, L.; Amirat, Y. Physical human activity recognition using wearable sensors. *Sensors* **2015**, *15*, 31314–31338. [CrossRef]
19. Casale, P.; Pujol, O.; Radeva, P. Human activity recognition from accelerometer data using a wearable device. In Proceedings of the Iberian Conference on Pattern Recognition and Image Analysis, Las Palmas de Gran Canaria, Spain, 8–10 June 2011; pp. 289–296.
20. Lu, Y.; Wei, Y.; Liu, L.; Zhong, J.; Sun, L.; Liu, Y. Towards unsupervised physical activity recognition using smartphone accelerometers. *Multimed. Tools Appl.* **2017**, *76*, 10701–10719. [CrossRef]
21. Walse, K.H.; Dharaskar, R.V.; Thakare, V.M. Pca based optimal ann classifiers for human activity recognition using mobile sensors data. In *Proceedings of the First International Conference on Information and Communication Technology for Intelligent Systems*; Springer: Berlin/Heidelberg, Germany, 2016; Volume 1, pp. 429–436.

22. Hammerla, N.Y.; Halloran, S.; Plötz, T. Deep, convolutional, and recurrent models for human activity recognition using wearables. *arXiv* **2016**, arXiv:1604.08880.
23. Chen, Y.; Xue, Y. A deep learning approach to human activity recognition based on single accelerometer. In Proceedings of the 2015 IEEE International Conference on Systems, Man, and Cybernetics, Hong Kong, China, 9–12 October 2015; pp. 1488–1492.
24. Jiang, W.; Yin, Z. Human activity recognition using wearable sensors by deep convolutional neural networks. In Proceedings of the 23rd ACM international conference on Multimedia, Brisbane, Australia, 26–30 October 2015; pp. 1307–1310.
25. Almaslukh, B.; AlMuhtadi, J.; Artoli, A. An effective deep autoencoder approach for online smartphone-based human activity recognition. *Int. J. Comput. Sci. Netw. Secur.* **2017**, *17*, 160–165.
26. Wang, A.; Chen, G.; Shang, C.; Zhang, M.; Liu, L. Human activity recognition in a smart home environment with stacked denoising autoencoders. In Proceedings of the International Conference on Web-Age Information Management, Nanchang, China, 3–5 June 2016; pp. 29–40.
27. Singh, D.; Merdivan, E.; Psychoula, I.; Kropf, J.; Hanke, S.; Geist, M.; Holzinger, A. Human activity recognition using recurrent neural networks. In Proceedings of the International Cross-Domain Conference for Machine Learning and Knowledge Extraction, Reggio, Italy, 29 August–1 September 2017; pp. 267–274.
28. Ordóñez, F.J.; Roggen, D. Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition. *Sensors* **2016**, *16*, 115. [CrossRef]
29. Pienaar, S.W.; Malekian, R. Human activity recognition using LSTM-RNN deep neural network architecture. In Proceedings of the 2019 IEEE 2nd Wireless Africa Conference (WAC), Pretoria, South Africa, 18–20 August 2019; pp. 1–5.
30. Krishna, K.; Jain, D.; Mehta, S.V.; Choudhary, S. An lstm based system for prediction of human activities with durations. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* **2018**, *1*, 1–31. [CrossRef]
31. Nafea, O.; Abdul, W.; Muhammad, G.; Alsulaiman, M. Sensor-Based Human Activity Recognition with Spatio-Temporal Deep Learning. *Sensors* **2021**, *21*, 2141. [CrossRef]
32. Guan, Y.; Plötz, T. Ensembles of deep lstm learners for activity recognition using wearables. *Proc. ACM Interact. Mobile Wearable Ubiquitous Technol.* **2017**, *1*, 1–28. [CrossRef]
33. Xia, K.; Huang, J.; Wang, H. LSTM-CNN architecture for human activity recognition. *IEEE Access* **2020**, *8*, 56855–56866. [CrossRef]
34. Zebin, T.; Sperrin, M.; Peek, N.; Casson, A.J. Human activity recognition from inertial sensor time-series using batch normalized deep LSTM recurrent networks. In Proceedings of the 2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Honolulu, HI, USA, 18–21 July 2018; pp. 1–4.
35. Mutegeki, R.; Han, D.S. A CNN-LSTM approach to human activity recognition. In Proceedings of the 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Fukuoka, Japan, 19–21 February 2020; pp. 362–366.
36. Novac, P.E.; Boukli Hacene, G.; Pegatoquet, A.; Miramond, B.; Gripon, V. Quantization and Deployment of Deep Neural Networks on Microcontrollers. *Sensors* **2021**, *21*, 2984. [CrossRef] [PubMed]
37. Novac, P.E.; Castagnetti, A.; Russo, A.; Miramond, B.; Pegatoquet, A.; Verdier, F.; Castagnetti, A. Toward unsupervised Human Activity Recognition on Microcontroller Units. In Proceedings of the 2020 23rd Euromicro Conference on Digital System Design (DSD), Kranj, Slovenia, 26–28 August 2020; pp. 542–550.
38. Zhao, Y.; Yang, R.; Chevalier, G.; Gong, M. Deep Residual Bidir-LSTM for Human Activity Recognition Using Wearable Sensors. *Math. Probl. Eng.* **2018**, *2018*, 7316954.
39. Mekruksavanich, S.; Jitpattanakul, A. Biometric User Identification Based on Human Activity Recognition Using Wearable Sensors: An Experiment Using Deep Learning Models. *Electronics* **2021**, *10*, 308. [CrossRef]
40. Agarwal, P.; Alam, M. A Lightweight Deep Learning Model for Human Activity Recognition on Edge Devices. *Procedia Comput. Sci.* **2020**, *167*, 2364–2373. [CrossRef]
41. STMicroelectronics. STM32 Solutions for Artificial Neural Networks. 2021. Available online: https://www.st.com/content/st_com/en/ecosystems/stm32-ann.html (accessed on 16 April 2021).
42. Zhang, R.; Mu, C.; Yang, Y.; Xu, L. Research on simulated infrared image utility evaluation using deep representation. *Procedia Comput. Sci.* **2018**, *27*, 013012. [CrossRef]
43. Zhang, R.; Xu, L.; Yu, Z.; Shi, Y.; Mu, C.; Xu, M. Deep-IRTarget: An Automatic Target Detector in Infrared Imagery using Dual-domain Feature Extraction and Allocation. *IEEE Trans. Multimed.* **2021**. [CrossRef]
44. Zhang, R.; Wu, L.; Yang, Y.; Wu, W.; Chen, Y.; Xu, M. Multi-camera multi-player tracking with deep player identification in sports video. *Pattern Recognit.* **2020**, *102*, 107260. [CrossRef]
45. Xu, K.; Jiang, X.; Ren, H.; Liu, X.; Chen, W. Deep Recurrent Neural Network for Extracting Pulse Rate Variability from Photoplethysmography During Strenuous Physical Exercise. In Proceedings of the 2019 IEEE Biomedical Circuits and Systems Conference (BioCAS), Nara, Japan, 17–19 October 2019; pp. 1–4.
46. Senturk, U.; Yucedag, I.; Polat, K. Repetitiveneural network (RNN) based blood pressure estimationusing PPG and ECG signals. In Proceedings of the Repetitive Neural Network (RNN) Based Blood Pressure Estimation Using PPG and ECG Signals, Ankara, Turkey, 19–21 October 2018; pp. 1–4.
47. Reiss, A.; Indlekofer, I.; Schmidt, P.; Van Laerhoven, K. Deep ppg: Large-scale heart rate estimation with convolutional neural networks. *Sensors* **2019**, *19*, 3079. [CrossRef] [PubMed]

48. Shyam, A.; Ravichandran, V.; Sp, P.; Joseph, J.; Sivaprakasam, M. PPGnet: Deep Network for Device Independent Heart Rate Estimation from Photoplethysmogram. In Proceedings of the 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Berlin, Germany, 23–27 July 2019.
49. Hochreiter, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* **1998**, *6*, 107–116. [CrossRef]
50. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
51. Biagetti, G.; Crippa, P.; Falaschetti, L.; Saraceni, L.; Tiranti, A.; Turchetti, C. Dataset from PPG wireless sensor for activity monitoring. *Data in Brief* **2020**, *29*, 105044. [CrossRef]
52. Brophy, E.; Muehlhausen, W.; Smeaton, A.F.; Ward, T.E. CNNs for Heart Rate Estimation and Human Activity Recognition in Wrist Worn Sensing Applications. In Proceedings of the 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Austin, TX, USA, 23–27 March 2020; pp. 1–6.
53. Biagetti, G.; Crippa, P.; Falaschetti, L.; Focante, E.; Martínez Madrid, N.; Seepold, R. Machine Learning and Data Fusion Techniques Applied to Physical Activity Classification Using Photoplethysmographic and Accelerometric Signals. *Procedia Comput. Sci.* **2020**, *176*, 3103–3111. [CrossRef]
54. Musci, M.; De Martini, D.; Blago, N.; Facchinetti, T.; Piastra, M. Online Fall Detection using Recurrent Neural Networks on Smart Wearable Devices. *IEEE Trans. Emerg. Top. Comput.* **2020**. [CrossRef]
55. Eddins, S. Classify ECG Signals Using LSTM Networks. 2018. Available online: https://blogs.mathworks.com/deep-learning/2018/08/06/classify-ecg-signals-using-lstm-networks/ (accessed on 16 April 2021).
56. Chevalier, G. LSTMs for Human Activity Recognition. 2016. Available online: https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition (accessed on 16 April 2021).
57. Chavarriaga, R.; Sagha, H.; Calatroni, A.; Digumarti, S.T.; Tröster, G.; Millán, J.D.R.; Roggen, D. The Opportunity Challenge: A Benchmark Database for on-Body Sensor-Based Activity Recognition. *Pattern Recogn. Lett.* **2013**, *34*, 2033–2042. [CrossRef]
58. Zappi, P.; Lombriser, C.; Stiefmeier, T.; Farella, E.; Roggen, D.; Benini, L.; Tröster, G. Activity Recognition from On-Body Sensors: Accuracy-Power Trade-Off by Dynamic Sensor Selection. In *Wireless Sensor Networks*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 17–33.
59. Kwapisz, J.R.; Weiss, G.M.; Moore, S.A. Activity Recognition Using Cell Phone Accelerometers. *SIGKDD Explor. Newsl.* **2011**, *12*, 74–82. [CrossRef]
60. Anguita, D.; Ghio, A.; Oneto, L.; Parra, X.; Reyes-Ortiz, J.L. A public domain dataset for human activity recognition using smartphones. In Proceedings of the Esann, Bruges, Belgium, 24–26 April 2013; Volume 3, p. 3.
61. Zhang, M.; Sawchuk, A.A. USC-HAD: A Daily Activity Dataset for Ubiquitous Activity Recognition Using Wearable Sensors. In Proceedings of the 2012 ACM Conference on Ubiquitous Computing, UbiComp ’12, Pittsburgh, PA, USA, 5–8 September 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 1036–1043.
62. Biagetti, G.; Crippa, P.; Falaschetti, L.; Orcioni, S. Human Activity Recognition Using Accelerometer and Photoplethysmographic Signals. In *Intelligent Decision Technologies 2017*; Springer International Publishing: Cham, Switzerland, 2018; pp. 53–62.
63. Casson, A.J.; Vazquez Galvez, A.; Jarchi, D. Gyroscope vs. accelerometer measurements of motion from wrist PPG during physical exercise. *ICT Express* **2016**, *2*, 175–179. [CrossRef]