electronics

MDPI

Article

Are Microcontrollers Ready for Deep Learning-Based Human Activity Recognition?

Atis Elsts [1,*] and Ryan McConville [2]

[1] Institute of Electronics and Computer Science (EDI), LV-1006 Riga, Latvia
[2] Department of Engineering Mathematics, University of Bristol, Bristol BS81TW, UK; ryan.mcconville@bristol.ac.uk
* Correspondence: atis.elsts@edi.lv

Abstract: The last decade has seen exponential growth in the field of deep learning with deep learning on microcontrollers a new frontier for this research area. This paper presents a case study about machine learning on microcontrollers, with a focus on human activity recognition using accelerometer data. We build machine learning classifiers suitable for execution on modern microcontrollers and evaluate their performance. Specifically, we compare Random Forests (RF), a classical machine learning technique, with Convolutional Neural Networks (CNN), in terms of classification accuracy and inference speed. The results show that RF classifiers achieve similar levels of classification accuracy while being several times faster than a small custom CNN model designed for the task. The RF and the custom CNN are also several orders of magnitude faster than state-of-the-art deep learning models. On the one hand, these findings confirm the feasibility of using deep learning on modern microcontrollers. On the other hand, they cast doubt on whether deep learning is the best approach for this application, especially if high inference speed and, thus, low energy consumption is the key objective.

Keywords: machine learning; deep learning; neural networks; activity recognition; accelerometers

Citation: Elsts, A.; McConville, R. Are Microcontrollers Ready for Deep Learning-Based Human Activity Recognition? *Electronics* **2021**, *10*, 2640. https://doi.org/10.3390/electronics10212640

Academic Editor: Maria D. R-Moreno

Received: 30 September 2021
Accepted: 25 October 2021
Published: 28 October 2021

1. Introduction

In the last decade, deep learning has emerged as the dominant machine learning paradigm, kick-starting an exponential growth of artificial intelligence applications. Nevertheless, deep learning is resource intensive and requires large datasets. Thus far, its primary focus has been on high-performance systems for both training and inference. Deep learning on edge devices has become a secondary focus point in the recent years [1].

However, most computers that exist today are microcontrollers (MCU), and these devices have much lower computational capabilities and are often subject to other restrictions, such as low energy budgets. In 2020 alone, 23.5 billion new microcontrollers devices were manufactured [2], which greatly exceeds the numbers of other, more capable systems. Nevertheless, even microcontrollers are becoming more powerful with time. In the last decade, low-power and low-performance 8-bit and 16-bit microcontrollers have been replaced with 32-bit devices. These new 32-bit microcontrollers provide an unprecedented tradeoff between low energy consumption and high performance. Furthermore, these new MCU often include instruction set architectures (ISA) specifically optimized for deep learning applications.

Therefore, it is no surprise that deep learning on MCU is an emerging research field. For instance, TensorFlow [3], a dominant deep learning software framework, released a subset called TensorFlow Lite Micro (TLM) in 2019 [4], which aims to enable execution of Neural Network (NN) models on microcontrollers. Similarly, research work on edge computing naturally offers many ideas that can be reused in the MCU context, such as NN quantization, pruning, and architecture optimizations, as well as provides many examples of light-weight NN architectures.

However, microcontrollers are different than other, more typical edge devices. MCU RAM and program memory size is typically much smaller, often measured in kilobytes [5]. Many MCU devices are battery powered; thus, energy-efficiency is a major concern. GPU and other hardware accelerators are not typically included in MCU devices. Furthermore, many MCU devices are subject to high reliability or real-time requirements; if that is the case, nearly 100% of the MCU time cannot be spent on the inference, as a large safety margin must be left unallocated. As a result, MCU applications typically rely on classical ML methods, such as Decision Trees and Random Forest (RF), insofar as they use ML at all.

The research question we ask in this paper is this: does it make sense to use deep learning on MCU, given the adaptation of the deep learning ecosystem towards MCU, and the current, more powerful models of MCU? A priori, deep learning is expected to provide an increased classification accuracy and a simplified software architecture. Using neural networks allows for simplification of the inference pipeline, as it removes the need for separate feature selection and extraction steps. However, these benefits must be balanced with decreased energy efficiency and inference speed.

We focus on human activity recognition using accelerometer data, as this application is very popular in the field, as well as requires both high accuracy and high energy efficiency if executed on wearable devices. Furthermore, NN have repeatedly been shown to provide the best accuracy for this application when compared with other ML methods [6].

Contribution. To answer the research question, we perform an experimental comparison of Convolutional Neural Network (CNN) classifiers with RF classifiers on microcontrollers. More specifically, we:

- develop machine learning models (RF and CNN) suitable for activity recognition on MCU;
- port these models to C code to enable their evaluation on MCU; and
- investigate and compare performance metrics of these models based on the settings, such as the model of the MCU, compilation options, quantization options, and input data window size.

Key results. The results show approximately similar classification accuracy for RF and NN classifiers. The results also confirm that real-time inference using NN is feasible on modern microcontrollers, as the fastest comparable (in terms of accuracy) NN classifiers have inference time on the order of 3 ms on a mid-range microcontroller. However, these speeds are still relatively slow compared with the RF classifiers, which achieve sub-ms inference speeds on the same device.

Structure. The paper is organized as follows: In Section 2, we provide an overview of the background of machine learning on microcontrollers. In Section 3, we describe our experimental design, including the dataset and parameters of the machine learning models. In Section 4, we present experimental results. Section 5 contains a discussion of the results, Section 2.4 surveys related work in the area, and Section 6 concludes the paper.

2. Background and Related Work

2.1. Machine Learning

The last decade has seen an exponential growth in deep learning; it has to some extent replaced "classical" ML techniques, such as Support Vector Machines (SVM), Linear and Logistic Regression, Bayesian estimation, Decision Trees, and Random Forests. The rapid growth of deep learning ML applications is enabled by frameworks, such as TensorFlow [3]. A version of the TensorFlow framework that is targeted towards mobile and IoT devices is called TensorFlow Lite.

A machine learning model, especially a deep learning model, can be trained directly on the data that needs to be classified. However, for many types of models, this produces suboptimal results. In particular, this is the case for the classical ML techniques, including RF. Better accuracy and training speed is possible if the data is first preprocessed and a number of *features* computed from the data. Subsequently, an ML model is trained on top of these features. Simple statistical features show good results for the activity

classification problem from acceleration data [7,8]. A great many of such features can be computed and used; however, some of the features are more informative than others for a given problem. Similarly, some features are more efficient to compute than others, leading to a two-dimensional optimization problem to find the best group of features for the application. In this work, we rely on our previous results to solve the feature extraction and feature selection problems [9]. We use these features as inputs for our RF classifiers. For the NN classifier, we use the raw data instead, as common in research literature. A brief investigation confirms that using features for training NN classifiers does not give substantially more accurate results, but it does come with an additional run-time energy cost for the feature extraction step.

2.2. Microcontrollers

A microcontroller is a small computer on a single integrated circuit ("chip"). It contains a processing unit, on-board memory (RAM & program memory), and input/output peripherals. For the purposes of this paper, we make no distinction between microcontrollers and microcontroller-based Systems-on-Chip (SoC), which may integrate multiple microcontroller cores with a larger number of peripherals.

Many modern microcontrollers are based on the ARM RISC (reduced instruction set computing) architectures. Specifically, ARM Cortex-M 32-bit cores are widely used in low-power embedded systems. Challengers, such as RISC-V, are appearing; RISC-V offers a fully open instruction set architecture (ISA) that does not require any licensing fees. Chip companies typically build their microcontrollers around such an ARM or a RISC-V core. This marks a significant change from just a decade ago, when most controllers were based on custom 8-bit and 16-bit architectures. The modern 32-bit microcontrollers has access to a larger memory space and have a much higher computational performance, while keeping similar or better energy-efficiency [10].

This evolution means that it is now possible to run machine learning classifiers directly on the microcontroller. The most important features for machine learning are:

- Current consumption in the active mode. The MCU needs to be energy efficient to avoid rapidly running out of battery when machine learning is used.
- RAM size. Neural network models need to fit in the RAM of the microcontroller; this means that most state-of-the-art models cannot be directly used.
- Flash memory size. Other models, such as Random Forest classifiers, can be placed in the flash memory instead of RAM; the former usually is larger, meaning that larger models can be used.
- Instruction set. Cortex-M family microcontrollers use ARMv7-M and ARMv8-M instruction sets, which include instructions specifically optimized for neural networks. There are supporting libraries, such as CMSIS-NN, that allow the use of these instructions.

Table 1 shows examples of microcontrollers. **ATSAM3X8E** is a microcontroller that belongs to the SAM3X family [11] produced by Microchip Technology. It is used in the highly popular Arduino devices, particularly in Arduino Due (https://store.arduino.cc/arduino-due, accessed on 24 October 2021). It is a basic Cortex-M3 microcontroller, which offers quite high performance for applications that do not need to perform many floating-point operations. This is the only microcontroller in the table that does not include a hardware floating point unit (FPU).

nRF52840 [12] and **nRF5340** [13] are SoC produced by Nordic Semiconductors. They are very energy efficient, on par with the best other examples in the class, for example, the SoC in the Texas Instrument Simplelink series. nRF52840 uses Cortex-M4 core with a FPU; it has the lowest active-mode current consumption and cost from the microcontrollers compared in the Table. Many wearable devices use this chip [14,15]. nRF5340 is a more recent model based on the next-generation Cortex-M33 core; it is the only one in the table that supports the ARMv8 ISA with several new instructions (the other MCU use the ARMv7 ISA).

STM32F746NG [16] belongs to the STM32F series produced by STMicroelectronics. It features a more powerful and faster Cortex-M7 core with FPU, as well as more peripherals, compared with the rest of the devices in the Table 1.

Table 1. Example microcontrollers.

Name	Core	Clock Freq. MHz	Active Mode Current, mA	RAM kb	Flash kb	Price EUR
ATSAM3X8E	Cortex-M3	84	70.9	96	512	8.38
nRF52840	Cortex-M4F	64	3.3	256	1024	3.43
nRF5340	Cortex-M33	128	7.3	512	1024	7.20
STM32F746NG	Cortex-M7F	216	102.0	320	1024	13.34

2.3. Machine Learning on Microcontrollers

Unlike conventional computing systems, microcontroller operating systems do not support the typical ML frameworks. These frameworks require a high-level programming language environment and libraries, which are not part of the software typically running on microcontrollers. Therefore, the key challenge is to enable either interpretation or direct execution of ML models in a very limited operational environment.

In order to solve this challenge for NN, works rely on the TensorFlow Lite Micro framework [4], which utilizes the interpretation approach. The framework allows you to take a TensorFlow Lite model and upload it to a microcontroller without making any changes to the model. During run-time, the TensorFlow Lite model is interpreted by the TensorFlow Lite Micro support library, which is written in C++ and, therefore, can be compiled to microcontroller-specific machine code. However, a key limitation of this approach is that not all TensorFlow Lite operations are supported by the TensorFlow Lite Micro C++ library. For example, it is not possible to run models that use LSTM layers, 1D-convolutions, and a few others. Hybrid quantization is also not supported by the framework: the model has to be either fully quantized to 8-bit integers or not quantized at all.

In contrast, for RF models it is more natural to use the direct execution approach. A RF model can be translated to C programming language code in a fairly straightforward way, by interpreting a decision tree as a series of `if/else` statements. Afterwards, the C code can be compiled and translated to microcontroller-specific machine code that is directly executable on the microcontroller. Open-source software includes a number of solutions capable of generating C code from a given machine learning classifier, for example, the `sklearn-porter` (https://github.com/nok/sklearn-porter, accessed on 24 October 2021), `emlearn` (https://github.com/emlearn/emlearn, accessed on 24 October 2021), and `m2cgen` (https://github.com/BayesWitnesses/m2cgen, accessed on 24 October 2021) libraries. However, the existing solutions produce C code that is suboptimal for low-power microcontrollers; therefore, for this work, we developed our own code generator that works on `scikit-learn` Random Forest and Decision Tree models.

2.4. Related Work

Many different types of machine learning models have been successfully used for activity recognition from accelerometer data [7], for instance, Random Forest, Logistic Regression, Neural Network, and Conditional Random Field models. Furthermore, these models result in comparable predictive performance for the problem [7].

Existing ML solutions for microcontrollers typically use classical ML classifiers, such as RF, SVM, and Bayesian models [17–20].

A game-changing innovation in the field is the TensorFlow Lite Micro framework [4], which makes it relatively straightforward to deploy NN models on microcontrollers. However, due to its recency, there have been few academic works using this framework. Among them, some key papers engage in meta-level research. Banbudy et al. [21] perform NN architecture search in order to make the resulting models more suitable for microcontrollers.

Heim et al. [22] develop a toolchain that helps to accurately measure performance of NN classifiers and optimize them for deployment on microcontrollers. The actual applications are tackled by a few other papers. Crocion et al. [23] use an on-board NN to estimate Li-Ion battery State-of-Health, which is an important problem for battery-powered embedded systems. Coffen et al. [24] apply on-board NN in order to classify hand gestures from wearable sensor data.

Recently, the MLPerf working group has published "Tiny Benchmark" [25], a benchmark suite for comparing the performance of different NN models on different low-power hardware in an objective an repeatable way. This benchmark targets four application areas; however, activity classification is not one of these areas.

None of the research above compares deep learning and classical ML performance on modern microcontroller hardware platforms for the activity classification problem.

3. Experimental Design

3.1. Dataset

To conduct the experiments, we utilize the PAMAP2 dataset [26]. It is already widely used by research papers in the area and presents a relatively challenging target in comparison with other lab-based datasets for supervised learning.

The PAMAP2 dataset contains data of multiple physical activities performed by 9 subjects wearing 3 inertial measurement units (IMU). The IMU are placed over the wrist on the dominant arm, on the chest, and on the dominant side's ankle. Each IMU samples 3D accelerometer data, 3D magnetometer, and 3D gyroscope data with 100 Hz frequency. The participants also wear a heart rate monitor. The 12 "protocol" activities in the PAMAP2 dataset are: lying, sitting, standing, ironing, vacuum cleaning, ascending stairs, descending stairs, walking, Nordic walking, running, and rope jumping. There are 2.87 million data samples for each sensor in the "protocol" activities of the PAMAP2 dataset.

In this paper, we restrict ourselves to the wrist-based accelerometer sensor. This is because we are interested in low-energy-consumption activity recognition, and because wearable accelerometers are most typically placed on wrists. Using all three accelerometer sensors would increase the energy consumption threefold in a real system, while our preliminary experiments showing only marginal increase in accuracy. Turning on an gyroscope sensor on a real sensor leads to a massive increase of energy consumption in a real system, as low-power gyroscopes typically consume two to three orders of magnitude more energy than low-power accelerometers. For instance, the ICM-20948 [27] low-power IMU consumes 68.9 µA in accelerometer-only mode, but 1230 µA in gyroscope-only mode, while standalone accelerometers, such as MC3635 [28], consume only 2.8 µA when sampling at 100 Hz. Magnetometers are in-between accelerometers and gyroscopes in terms of energy-consumption (90 µA in magnetometer-only mode for ICM-20948), but they have much lower informative value on their own compared with the other two sensors.

3.2. Dataset Preparation

In order to run activity classification on the dataset, the data is first segmented in short windows. We experiment with both 128-sample and 256-sample windows (1.28 and 2.56 s, respectively), as these are typical values shown to give good results for this application [7]. We only consider windows with at least $\frac{2}{3}$ majority labels.

We use cross-validation with *one subject left out* test data, and rely on the micro F_1 score as the measure of classification accuracy. The approach works as follows: the nine human subjects in the dataset are processed iteratively. In each iteration, we train the selected classifier on eight subjects and use the single subject left out as a test data to obtain a test F_1 score. Finally, we report a single number: the mean of the test F_1 scores.

This approach allows us to use windows with a large overlap (i.e., have a lot of very similar input data samples) without worrying about corrupting the results due to potential overfitting. While this is a challenging setup, it is also more realistic in that it also reflects typical applications where a wearable device will not be trained on the user wearing

it. Initially, experiments showed that larger overlaps show better results with the deep learning classifiers. This is not surprising: larger overlaps means that the total number of input windows is larger, and deep learning is known to require large amounts of input data for good performance. Based on the empirical results shown in Section 4.3, we use the maximum possible window overlap for all classifiers: 127 sample overlap for 128 sample windows, and 255 sample overlap for 256 sample windows.

3.3. Classification with Neural Networks

Architectures

While it has been shown numerous times that neural networks are able to accurately perform activity recognition from raw accelerometer data, there are numerous neural network architectures proposed to do this [6,7,29]. Typically, however, these architectures are designed with too little regard for energy efficiency, and they often use functionality not supported in typical microcontrollers with current deep learning frameworks, such as the lack of support for recurrent neural networks in TensorFlow Lite for Microcontrollers.

With this limitation in mind, we select two different architectures from the neural network activity recognition literature which can be implemented in this setting. The first is PerceptionNet [6], which is a four-layer neural network architecture, comprised of three convolutional layers and a final fully connected layer. We also implement another convolutional model, which we call DeepCNN, originally proposed by Ordóñez and Roggen [29] to demonstrate the superiority of recurrence (which, as noted previously, cannot be used here) for activity classification. This network is similar to PerceptionNet in that it consists of four convolutional layers, but it also includes two additional fully connected layers. Notable hyperparameters describing all neural network models can be found in Table 2, and a visualization of each architecture can be found in Figure 1, with the visualizations produced using net2vis [30].

Table 2. A comparison of the three neural network architecture main features.

	PerceptionNet	**DeepCNN**	**Custom**
Input Shape	(3, 128, 1)	(128, 3, 1)	(1, 128, 3)
Convolutional Layers	3	4	1
Convolutional Filters	48, 96, 96	64,64,64,64	32
Convolutional Strides	1, 1, 3	1	(1, 8)
Convolutional Kernels	(1, 15), (1, 15), (3,15)	(5,1)	(1,64)
Pooling Layers	3	0	2
Dense Hidden Layers (Neurons)	0 (0)	2 (128, 128)	1 (16)

While both models are reported to perform well, as shown in Section 4, both are too large to run on the microcontrollers. Further, when their power is measured, it is significantly larger than that of the Random Forest method. Thus, we designed a new smaller architecture in an effort to maintain predictive performance while minimizing memory and energy usage. The resulting model consists of a single convolutional layer, with 32 one-dimensional convolutional filters, a fully connected layer with 16 neurons, and a final output layer with a neuron for each class. Other significant architecture efficiency decisions include increasing the stride length of the convolutions to 8 with a kernel size of 64.

The general principle of this approach was to design a lightweight CNN with 1D convolutions. As (large) CNNs have been shown to be effective at this task, and as recurrent layers are not possible on the microcontrollers currently, we use a single convolutional layer and a single fully connected layer with a small number of filters and neurons. We further reduce the number of operations by increasing the stride length. Experimentally, we considered other additions to the architecture, such as the use of use of more filters, but found that they offered little performance gains but at the cost of increased energy usage. Figure 1 shows the three architectures evaluated, with Figure 1a showing the architecture of the custom

CNN. Note that we include two dropout layers (0.5) in the architecture, in order to improve predictive performance, but these have no effect on energy usage as they are deactivated during inference. Further, our custom model makes use of L2 activity regularization (0.01) on the convolutional and fully connected layers, as well as kernel regularization (0.05) on the fully connected layer.

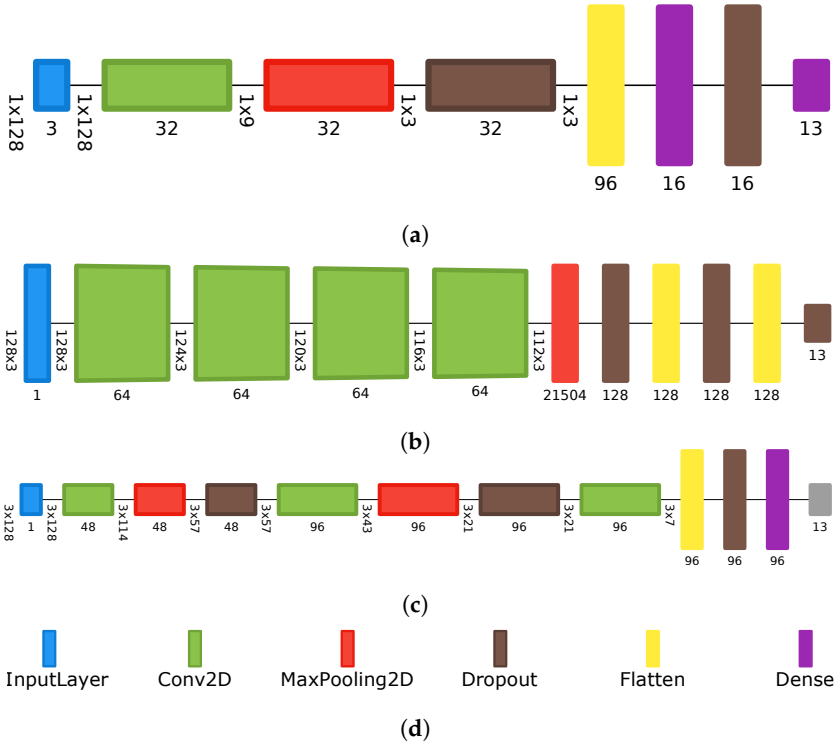

Figure 1. Visualizations of the different architectures evaluated. (**a**) The architecture of the custom CNN. (**b**) The architecture of the deep CNN. (**c**) The architecture of PerceptionNet. (**d**) Architecture legend.

3.4. Classification with Random Forests

3.4.1. Features

It is widely known that activity classification with Random Forests does not give good results if performed directly on the raw data. Instead, multiple *features* should be first extracted from the raw data, which then can be passed to the Random Forest as input data. Features can be time-domain, frequency-domain, and other; they can be statistical or hand-crafted for a specific application.

Fortunately, we can automate the selection and extraction of features by reusing our previous work [9], which provides a software framework for this purpose. The framework allows the user to define the relative importance of energy consumption and classification accuracy. Given this input, it will automatically select the groups of features with the best trade-offs between these two metrics and quantify the energy required to compute these features on a microcontroller.

3.4.2. Other Parameters

The following parameters also affect RF performance:

- number of trees;
- maximum depth of trees; and
- the splitting metric, and the minimum number of items per class.

A larger number of trees and deeper trees may give better classification accuracy, but require more space in the program memory and decrease the classification speed. We look at these trade-offs in the performance evaluation (Section 4). As for the splitting metric and other parameters, in this work, we leave these values to the defaults provided by scikit-learn.

4. Experimental Evaluation

4.1. Experimental Settings

In this section, we describe the main experimental settings used in the performance evaluation of the different machine learning classifiers.

Table 3 shows the main experimental settings selected for the evaluation. We design and evaluate classifiers that recognized the 12 "protocol" activities in the PAMAP2 dataset. In the evaluation, we experiment with different microcontrollers, but focus on nRF5340 as our main target, at is it a mid-range microcontroller with a relatively high RAM and flash memory capacity, low price, and the most up-to-date instruction set (Table 1). While more powerful MCU, such as STM32F746NG, are available, they have much higher price and energy consumption per operation.

Table 3. Experimental settings.

Property	Value
Common Settings	
Window size	128 or 256
Dataset	PAMAP2
Number of classes	12
Neural Network	
Framework	TensorFlow Lite Micro v2.4.0
Training epochs	200
Batch size	4096
Quantization	int8
Convolutional layers	1
Convolution filters	32
Random Forest	
Framework	scikit-learn
Number of trees	50
Max depth	9
Number of features	9

Regarding Random Forests, the most important parameters to select are the number of trees, the maximal tree depth, and the pre-computed features to use in the classification. The values of the former two are selected to enable running the classifier on all of our target microcontrollers. Regarding features, we rely on our previous work on feature selection [9]. We pre-compute a large number of features, and use the greedy algorithm [9] to select the ones with the best energy/accuracy tradeoff. What remains to be done is to determine the number of features to use. By looking at the classification accuracy obtained by using progressively more features, we can see that the accuracy plateaus after more than nine features are used (Figure 2). Hence, we use the best nine features in our further experiments.

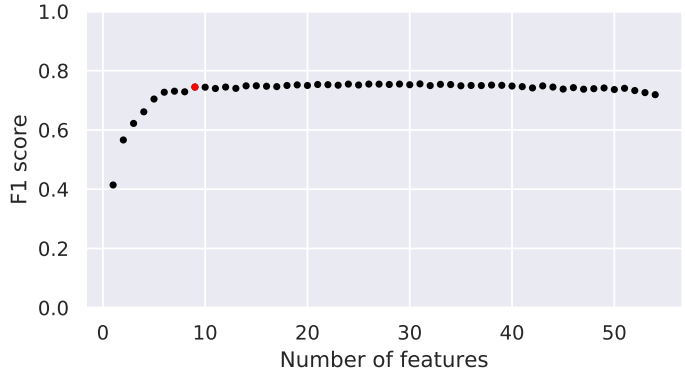

Figure 2. Feature selection for the Random Forest. The red dot marks the near-top accuracy achieved at 9 features.

Regarding Neural Networks, it is important to verify that the training is complete, and that no large overfitting or underfitting occurs. To control this, we plot the training history (Figure 3). It is clear that, after a few tens of epochs, the maximum accuracy on the test data is achieved. Increasing the number of epochs neither increases nor increases the accuracy on the test data. For extra safety, we use 200 training epochs in our further experiments.

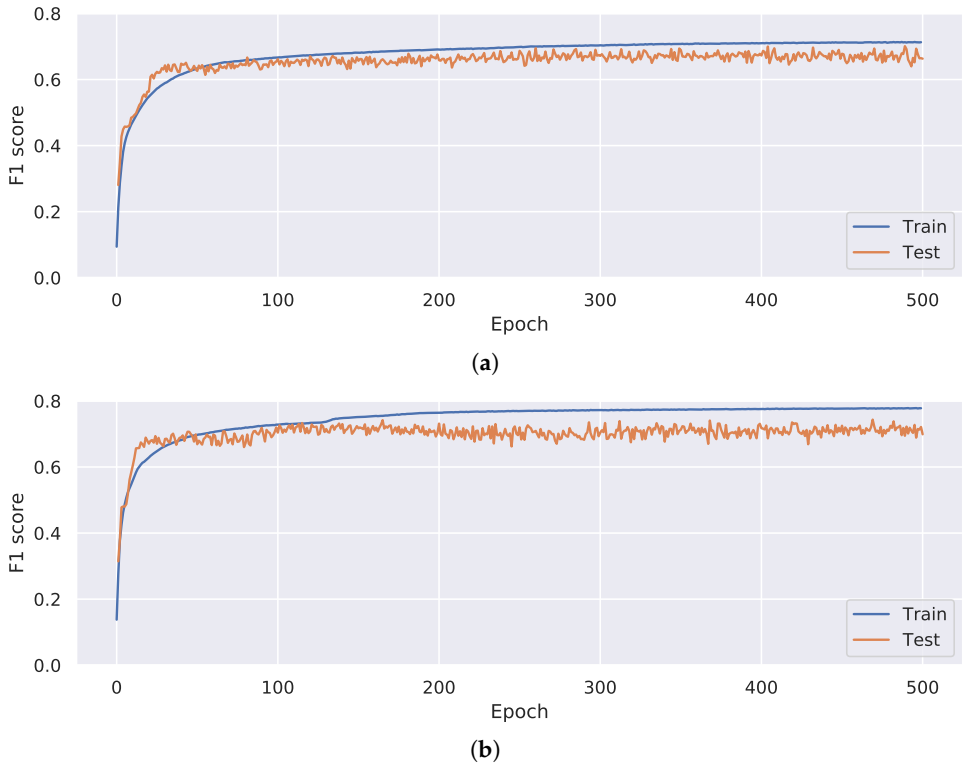

Figure 3. Custom CNN score evolution on training and test subsets. (**a**) Window size 128. (**b**) Window size 256.

4.2. Random Forest Evaluation

Figure 4 shows the tradeoff between the window size and the number of trees, in terms of classification accuracy and performance. A very small number of trees, such as 10 trees, leads to reduced classification accuracy. However, adding more than 50 trees results in marginal gains.

Larger window size leads to better classification accuracy, since the classifier is able to see more of the input data. It also has only a constant impact on the speed, since the only part that changes is the feature extraction step, which does not depend on the RF parameters. Finally, the RF operating on a larger window actually takes less memory, as it is simpler. If a large number of trees is desired, then, using a large window size is a tradeoff almost certainly worth making.

Finally, we can note that a single inference only requires around 1 ms to execute. To spell the conclusion out, RF for activity classification are very fast and efficient. Using an RF for the task allows for performance of hundreds of inferences in a second, without even requiring that the MCU is 100% committed to this task alone. If a high energy efficiency is desired instead, very low MCU duty cycle can be achieved, spending up to 99.9% of the MCU time in a low-power mode.

4.3. Neural Network Architecture Evaluation

Table 4 shows the effect of the window overlap size on each of the models. For DeepCNN and PerceptionNet, the mean and standard deviation (in brackets) is reported over three runs, while, for the custom model, this is reported over ten runs (due to increased efficiency).

Table 4. Window overlap size effect.

Window Overlap	DeepCNN	PerceptionNet	Custom
99%	0.73 (0.002)	0.68 (0.008)	0.69 (0.006)
66%	0.74 (0.005)	0.66 (0.001)	0.52 (0.010)
50%	0.74 (0.002)	0.64 (0.011)	0.42 (0.19)

When the overlap between windows used in the training data is small, the state-of-the-art models show much better performance than our custom CNN. However, when the amount of training data is increased by setting the maximum overlap, the custom model achieves nearly the same accuracy, which is also similar to the results achieved by the 50-tree RF classifier (Figure 4a).

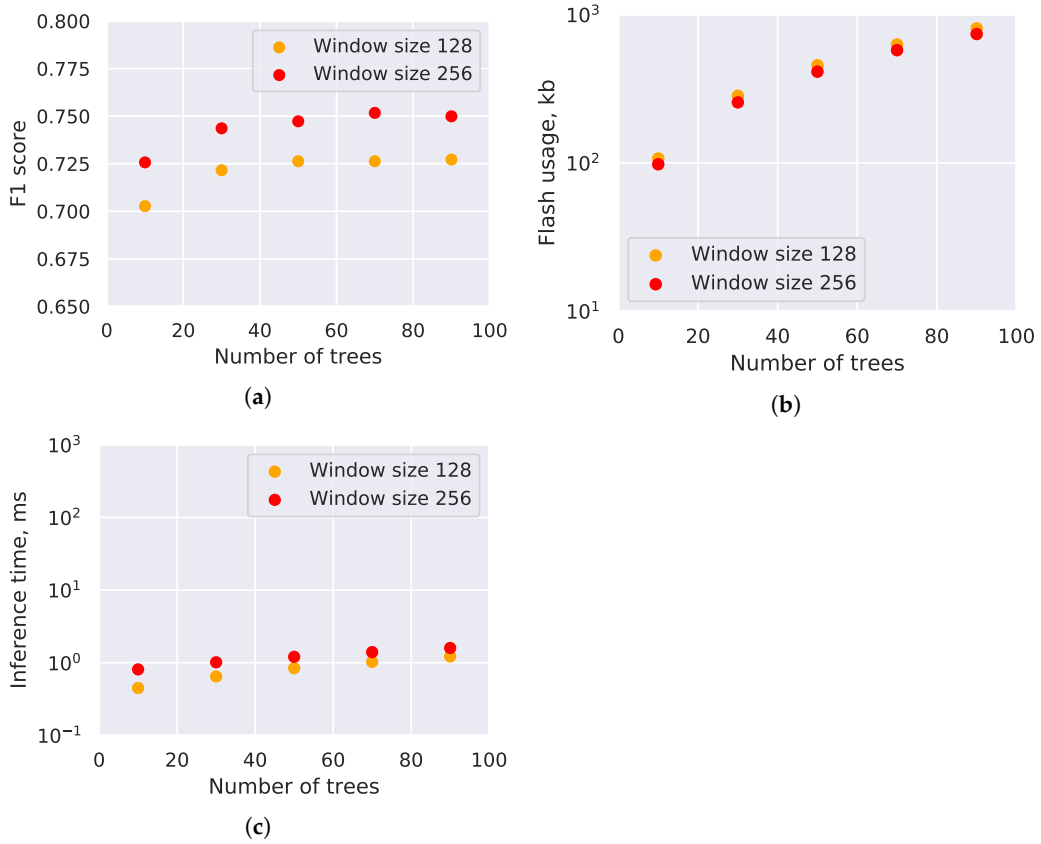

Figure 4. Random Forest accuracy and performance. Results on nRF5340. (**a**) F_1 scores. (**b**) Flash memory usage. (**c**) Inference time, including feature extraction time.

Unlike the other experiments, the comparison of the different CNN architectures is not done on a microcontroller, but instead on a more powerful system. The CNN architectures from the research literature are not suitable for execution on microcontrollers due to their size (Figure 5b), so we use Raspberry Pi Model 4B instead. The software libraries remain the same as for MCU; in particular, we use the TensorFlow Lite Micro framework, and we compile the software libraries with CMSIS-NN support enabled.

Figure 5 shows the results. The custom model demonstrates similar classification accuracy while being approximately 290 times faster than the state-of-the-art models. With window size 128, it also 40 times smaller than the PerceptionNet and 263 times smaller than the DeepCNN. With window size 256, the size difference is approximately 33 times with PerceptionNet and 430 times with the DeepCNN. These results demonstrate the importance of developing custom CNN models suitable for microcontrollers.

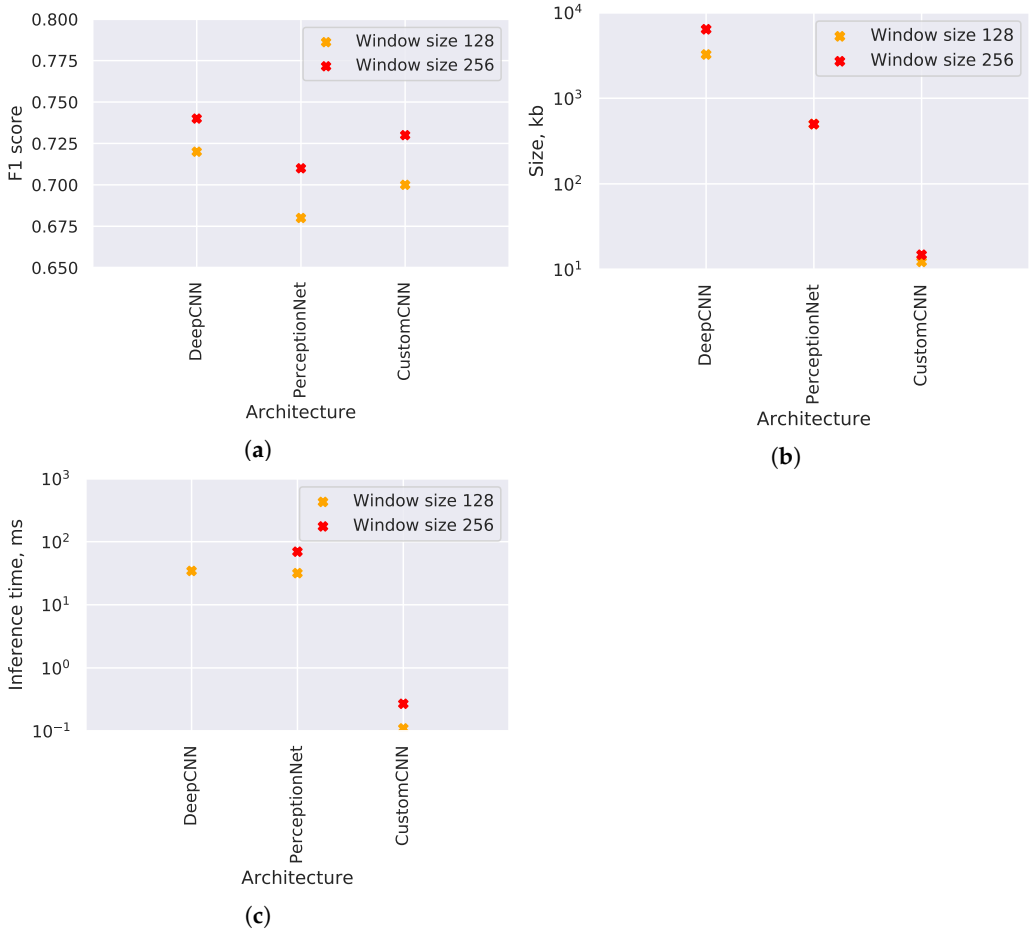

Figure 5. CNN accuracy and performance depending on the architecture. Results on Raspberry Pi; DeepCNN with window size 256 fails to fully build on this system; its inference speed is not measured. (**a**) F_1 scores. (**b**) Memory usage. (**c**) Inference time.

4.4. Neural Network Optimization Evaluation

The impact of quantization is shown in Figure 6. Only unquantized models and models with full `int8` quantization are supported by TensorFlow Lite Micro at the moment. The results show that quantization has some positive impact on the model size; however, it is even more important for execution speed, achieving an order of magnitude improvement.

Figure 7 shows the impact from enabling hardware FPU and enabling software optimizations for the target hardware by using the CMSIS-NN library. In this particular instance, the FPU has no effect on the speed, as the model is fully quantized and does not require any floating point operations during runtime. In contrast, enabling the CMSIS-NN library has the largest impact from all options tested, speeding up the inference by two orders of magnitude. Other, earlier models that we developed did require FPU despite also being fully quantized; however, the impact of FPU for those models was still much smaller than the impact of CMSIS.

4.5. Neural Network and Random Forest Comparison

Figure 8 shows the performance comparison between CNN and RF models. The CNN are optimized for the maximum performance, with CMSIS-NN and FPU enabled, as well as `int8`-quantized (Table 3). Nevertheless, the inference speed of the RF classifiers is several times higher depending on the window size and on the microcontroller model.

The differences between the microcontroller models are relatively small and are mostly explained by their different CPU clock frequencies.

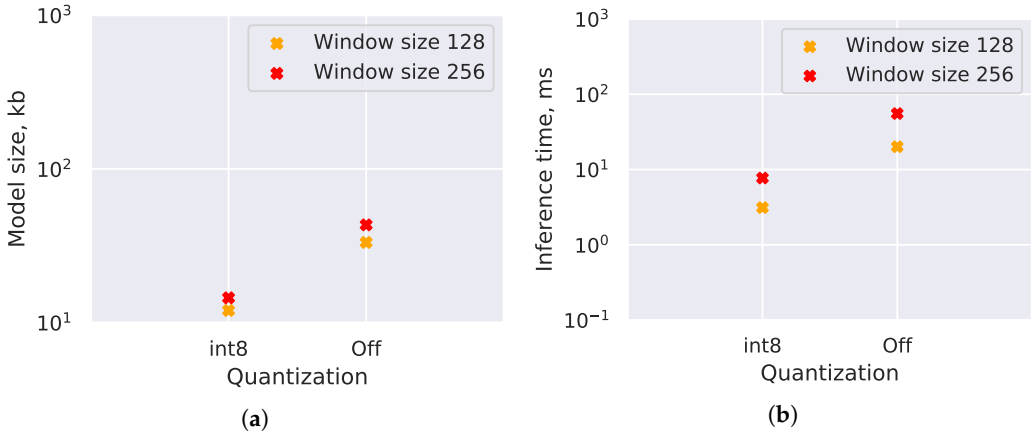

Figure 6. The impact of quantization on the performance of the custom CNN. Results on nRF5340. (**a**) Memory usage. (**b**) Inference time.

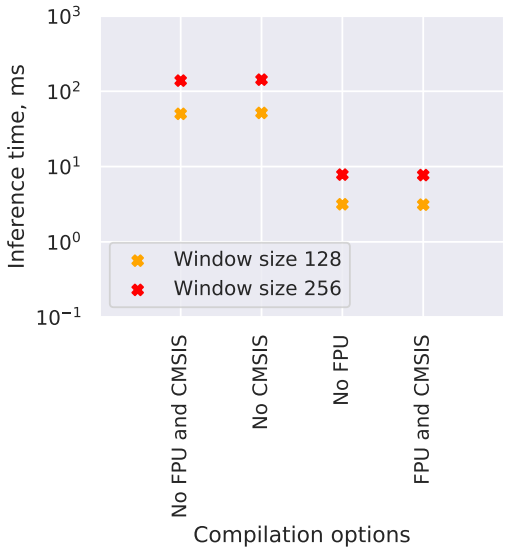

Figure 7. The custom CNN inference speed depending on build options. Results on nRF5340.

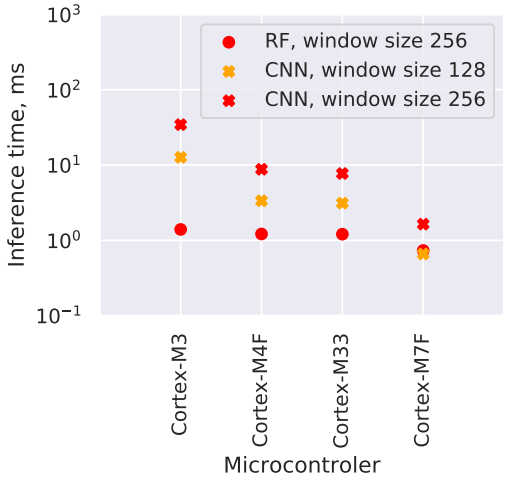

Figure 8. The custom CNN and RF inference speed comparison on different microcontrollers. Cortex-M3 was not able to run the CNN with window size 256 due to its limited RAM. RF with feature window size 128 not shown, as its speed closely matches the RF with window size 256.

5. Discussion

5.1. Neural Network Performance Factors

A number of factors impact the speed of Neural Networks on microcontrollers.

Neural Network architecture and quantization. It goes without saying that NN architecture plays a big role in the performance. The results allow for quantification of the exact impact of a few different architectures. We show that a simple custom CNN model gives almost identical classification accuracy compared with state-of-the-art CNN models, while having more than an order of magnitude lower size and faster inference speed, hence proportionally lower energy consumption. Quantization has an order of magnitude impact on the speed, as well, although its impact on the size is much more limited.

Microcontroller core architecture. The results show clear differences between the different microcontrollers with Cortex-M cores. However, most of these differences can be explained by improvements in MCU clock frequency. If we control for this factor, the actual differences between Cortex-M3, Cortex-M4, Cortex-M7, and Cortex-M33 architectures are relatively small: a few percent to a few tens of percent. The updated instruction sets of the newer MCU models do have some impact, but it pales in comparison with other items in this list.

Software specifically optimized for the target hardware. Enabling the CMSIS-NN hardware acceleration library has one-to-two orders magnitude impact on the CNN performance (Figure 7). Only major changes in the CNN architecture itself have a higher impact. If the CNN architecture is fixed, then, using the CMSIS-NN library is the most important factor required to achieve fast inferences on ARM-based microcontrollers.

5.2. Potential Criticisms of This Work

In this section, we review and address some potential criticism of this work, and clarify its scope and assumptions.

Training data amount.

Issue: Deep learning is known to benefit from larger datasets relative to classical ML methods. Increasing the training data amount will increase the accuracy substantially more for CNN than for RF, making the CNN more competitive.

Our comment: Data collection in ML is not a straightforward process, as larger datasets often require extensive efforts to collect and label, and increased sizes often lead to increased issues with data quality. Many real-world applications require good performance from reasonably-sized datasets.

Complexity of applications.

Issue: Modern wearable devices include an ever increasing set of sensors that can help with activity recognition and analysis, and more sensors including on-board cameras are constantly proposed by the research community. These enable new, more complex activity recognition applications that would benefit from the more powerful CNN classifiers.

Our comment: The accelerometer sensor has proven to be an accurate, energy-efficient, and versatile component of activity recognition system; it consumes several orders of magnitude less energy compared with gyroscopes and embedded cameras. For rough-granularity activities, such as the ones analyzed in this work, accelerometers are typically considered sufficient in the existing literature. We make no claims about higher-granularity activity recognition nor about more complex applications (e.g., skill assessment, quality-of-movement assessment).

Robustness of classification.

Issue: Deep learning is known to be more robust than classical ML techniques, especially those which rely on manually engineered features.

Our comment: We admit that the performance on a dataset collected in a controlled environment may not translate to similar performance on real-world data, and, in our future work, we will be looking at the robustness of ML classifiers for activity recognition. However, the features used by the RF are not manually engineered ones specific for this application; they are common statistical features used in wide variety of areas. Furthermore,

the results are obtained without any frequency-domain features; these common features can be added to the RF input data, further increasing the RF accuracy.

Moore's Law.

Issue: Moore's Law renders performance concerns irrelevant to longer time frames.

Our comment: It is not clear how long Moore's Law will continue to last. Regardless, our focus is on low-cost, energy-efficient common microcontroller. The performance metrics in this field require a tradeoff between computational speed from one side, and energy efficiency and unit cost from the other side. Moore's Law in this field may manifest not only as performance increase but also as decreased cost and energy usage, while keeping performance at the same level. Cheap, low-performance microcontrollers are not going to disappear.

Dedicated hardware.

Issue: Hardware accelerators for deep learning are going to become commonplace, rendering the current comparison irrelevant.

Our comment: The evaluated micro-controllers already include ISA optimized for deep learning applications. Using these optimized instructions leads to an order-of-magnitude improvement in the CNN inference speed; in contrast, the RF performance is not affected. Further optimizations and hardware add-ons may come; however, the concerns about unit cost and energy efficiency make it unlikely that massive deep learning accelerators are going to be deployed most microcontroller devices.

6. Conclusions

We select human activity recognition as our target application and compare the accuracy and performance of deep learning models (specifically, CNN) with a classical ML technique (Random Forest). The results show that the RF achieves slightly better classification accuracy with several times higher performance, even when a heavily optimized version of a CNN is used. It is also worth stressing that the RF classifier required an order of magnitude less human effort and expertise to design than the custom CNN. The same can be said about the computational time required to train the two respective classifiers for our particular experiments. These results cast doubt on the suitability of deep learning for this particular application, at least on the current generation of microcontrollers. As a result, the question posed in the title of this paper has a negative answer at this time.

We also identify the main performance factors for deep learning on microcontrollers. In decreasing order, they are: NN architecture; enabling hardware optimized libraries, such as CMSIS-NN; NN quantization; microcontroller clock frequency; and, finally, microcontroller instruction set.

Depending on the network architecture, other aspects, such as hardware FPU, may also be important. For example, if network quantization is either not used at all or if hybrid quantization is applied, then floating point operations are necessary, and FPU becomes important. In the future, we plan to study the impact of the network architecture choices on the performance, including investigating the exact conditions when hardware FPU is beneficial, as well as performance impact of architecture aspects, such as the convolution filter dimensions, stride length, and padding.

Author Contributions: Conceptualization, A.E.; methodology, A.E. and R.M.; software, A.E. and R.M.; validation, A.E. and R.M.; investigation, A.E. and R.M.; resources, A.E. and R.M.; writing—original draft preparation, A.E. and R.M.; writing—review and editing, A.E. and R.M.; visualization, A.E. and R.M.; project supervision, A.E.; project administration, A.E.; funding acquisition, A.E. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the ERDF Activity 1.1.1.2 "Post-doctoral Research Aid" (No. 1.1.1.2/VIAA/2/18/282). It was also partially supported by the SPHERE Next Steps Project funded by the UK Engineering and Physical Sciences Research Council (EPSRC), Grant EP/R005273/1.

Data Availability Statement: The PAMAP2 dataset used in this work is available at https://archive.ics.uci.edu/ml/datasets/PAMAP2+Physical+Activity+Monitoring, accessed on 24 October 2021.

Acknowledgments: The authors thanks Didzis Lapsa for his help on microcontrollers and the experimental evaluation.

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

References

1. Satyanarayanan, M. The Emergence of Edge Computing. *Computer* **2017**, *50*, 30–39. [CrossRef]
2. Microcontroller Unit (MCU) Shipments Worldwide from 2015 to 2023. Available online: https://www.statista.com/statistics/935382/worldwide-microcontroller-unit-shipments/ (accessed on 24 October 2021).
3. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. Tensorflow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
4. David, R.; Duke, J.; Jain, A.; Janapa Reddi, V.; Jeffries, N.; Li, J.; Kreeger, N.; Nappier, I.; Natraj, M.; Wang, T.; et al. TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems. In Proceedings of the 4th MLSys Conference, San Jose, CA, USA, 4–7 April 2021.
5. Bormann, C.; Ersue, M.; Keranen, A. Terminology for Constrained-Node Networks. RFC 7228, IETF: Fremont, CA, USA 2014. Available online: https://datatracker.ietf.org/doc/html/rfc7228 (accessed on 24 October 2021).
6. Kasnesis, P.; Patrikakis, C.Z.; Venieris, I.S. PerceptionNet: A deep convolutional neural network for late sensor fusion. In *Proceedings of SAI Intelligent Systems Conference*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 101–119.
7. Twomey, N.; Diethe, T.; Fafoutis, X.; Elsts, A.; McConville, R.; Flach, P.; Craddock, I. A Comprehensive Study of Activity Recognition Using Accelerometers. *Informatics* **2018**, *5*, 27. [CrossRef]
8. Elsts, A.; McConville, R.; Fafoutis, X.; Twomey, N.; Piechocki, R.; Santos-Rodriguez, R.; Craddock, I. On-Board Feature Extraction from Acceleration Data for Activity Recognition. In Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN), Madrid, Spain, 14–16 February 2018.
9. Elsts, A.; Twomey, N.; McConville, R.; Craddock, I. Energy-efficient activity recognition framework using wearable accelerometers. *J. Netw. Comput. Appl.* **2020**, *168*, 102770. [CrossRef]
10. Ko, J.; Klues, K.; Richter, C.; Hofer, W.; Kusy, B.; Bruenig, M.; Schmid, T.; Wang, Q.; Dutta, P.; Terzis, A. Low power or high performance? A tradeoff whose time has come (and nearly gone). In *European Conference on Wireless Sensor Networks*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 98–114.
11. SAM3X/SAM3A SeriesAtmel: SMART ARM-based MCU. Available online: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf (accessed on 24 October 2021).
12. nRF52840: Product Specification v1.2. Available online: https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.2.pdf (accessed on 24 October 2021).
13. nRF5340: Objective Product Specification v0.5. Available online: https://infocenter.nordicsemi.com/pdf/nRF5340_OPS_v0.5.pdf (accessed on 24 October 2021).
14. Fafoutis, X.; Vafeas, A.; Janko, B.; Sherratt, R.S.; Pope, J.; Elsts, A.; Mellios, E.; Hilton, G.; Oikonomou, G.; Piechocki, R.; et al. Designing wearable sensing platforms for healthcare in a residential environment. *EAI Endorsed Trans. Pervasive Health Technol.* **2017**, *12*, e1.
15. Vafeas, A.T.; Fafoutis, X.; Elsts, A.; Craddock, I.J.; Biswas, M.I.; Piechocki, R.J.; Oikonomou, G. Wearable Devices for Digital Health: The SPHERE Wearable 3. In Proceedings of the Embedded Wireless Systems and Networks (EWSN): On-Body Sensor Networks (OBSN 2020), Lyon, France, 17–19 February 2020.
16. STM32F745xx STM32F746xx. Available online: https://www.st.com/resource/en/datasheet/stm32f746ng.pdf (accessed on 24 October 2021).
17. Zheng, L.; Wu, D.; Ruan, X.; Weng, S.; Peng, A.; Tang, B.; Lu, H.; Shi, H.; Zheng, H. A novel energy-efficient approach for human activity recognition. *Sensors* **2017**, *17*, 2064. [CrossRef] [PubMed]
18. Sudharsan, B.; Patel, P.; Breslin, J.G.; Ali, M.I. Ultra-fast machine learning classifier execution on iot devices without sram consumption. In Proceedings of the 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), Kassel, Germany, 22–26 March 2021; pp. 316–319.
19. Boni, A.; Pianegiani, F.; Petri, D. Low-power and low-cost implementation of SVMs for smart sensors. *IEEE Trans. Instrum. Meas.* **2007**, *56*, 39–44. [CrossRef]
20. Leech, C.; Raykov, Y.P.; Ozer, E.; Merrett, G.V. Real-time room occupancy estimation with Bayesian machine learning using a single PIR sensor and microcontroller. In Proceedings of the 2017 IEEE Sensors Applications Symposium (SAS), Glassboro, NJ, USA, 13–15 March 2017; pp. 1–6.
21. Banbury, C.; Zhou, C.; Fedorov, I.; Matas, R.; Thakker, U.; Gope, D.; Janapa Reddi, V.; Mattina, M.; Whatmough, P. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. In Proceedings of the 4th MLSys Conference, San Jose, CA, USA, 4–7 April 2021.
22. Heim, L.; Biri, A.; Qu, Z.; Thiele, L. Measuring what Really Matters: Optimizing Neural Networks for TinyML. *arXiv* **2021**, arXiv:2104.10645.
23. Crocioni, G.; Pau, D.; Delorme, J.M.; Gruosso, G. Li-Ion Batteries Parameter Estimation With Tiny Neural Networks Embedded on Intelligent IoT Microcontrollers. *IEEE Access* **2020**, *8*, 122135–122146. [CrossRef]

24. Coffen, B.; Mahmud, M.S. TinyDL: Edge Computing and Deep Learning Based Real-time Hand Gesture Recognition Using Wearable Sensor. In Proceedings of the 2020 IEEE International Conference on E-health Networking, Application & Services (HEALTHCOM), Shenzhen, China, 1–2 March 2021; pp. 1–6.
25. Banbury, C.; Reddi, V.J.; Torelli, P.; Holleman, J.; Jeffries, N.; Kiraly, C.; Montino, P.; Kanter, D.; Ahmed, S.; Pau, D.; et al. MLPerf Tiny Benchmark. *arXiv* **2021**, arXiv:2106.07597.
26. Reiss, A.; Stricker, D. Creating and Benchmarking a New Dataset for Physical Activity Monitoring. In Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments, Heraklion, Greece, 6–8 June 2012; pp. 1–8. [CrossRef]
27. ICM-20948: World's Lowest Power 9-Axis MEMS MotionTracking™ Device. Available online: https://invensense.tdk.com/wp-content/uploads/2016/06/DS-000189-ICM-20948-v1.3.pdf (accessed on 24 October 2021).
28. MC3635 3-Axis Accelerometer. Available online: https://mcubemems.com/wp-content/uploads/2017/09/MC3635-Datasheet-APS-048-0044v1.5.pdf (accessed on 24 October 2021).
29. Ordóñez, F.J.; Roggen, D. Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. *Sensors* **2016**, *16*, 115. [CrossRef] [PubMed]
30. Bäuerle, A.; Ropinski, T. Net2Vis: Transforming Deep Convolutional Networks into Publication-Ready Visualizations. *arXiv* **2019**, arXiv:2106.07597.