Machine Learning on Microcontrollers

Yifei Yang
yifeiy3@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA

Yang Yang
yangy4@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA

1 INTRODUCTION

Nowadays, machine learning(ML), especially deep learning, is one of the most widely researched topics in the world. In recent years, the size of the models are growing significantly larger and consequently becoming computationally more expensive (BERT [8], a widely used pre-trained model for NLP tasks, has more than 300 million parameters), which leads to majority of the current research in the field are focusing on optimizing the algorithm and hardware for model speed-ups. However, due to the near-infinite storage space we have in most situations due to cloud computing, the size of these deep learning models are often ignored. This issue of model size rises when resources from cloud computing is no longer available, particularly, when running on IoT devices with micro-controller units (MCU) which often has megabytes of storage and kilobytes of memory rather than terabytes and gigabytes we get through cloud computing. Given the limited computing power and memory size of MCU, running these complex ML algorithms directly would be be a very challenging task.

As IoT devices with MCU are becoming increasingly prevalent, modern researches have started to notice this limitation from large model sizes, with a recent development of TinyML [14] that dynamically rescales the size of the model basing on provided machine architecture. The major challenge of running ML algorithm on MCU is the low computing power and small memory size. Therefore, there are researches that focus on optimizing the model inference library so that we can conduct large inference models such as deep learning on board [12].

On the other hand, the necessity for enabling these MCUs to run ML tasks also remains an open question. While directly performing data analytic near the sensor of the devices appears particularly useful in theory for IoT devices such as in the field of personalized healthcare [16], many of these modern applications can be achieved similarly through communications between these MCU devices with larger computational devices such as mobile phones to run the ML task with a similar effectiveness [10]. Furthermore, the tasks conducted on these IoT devices are relatively simpler and smaller in scale due to the devices' limitation in computation power; for example, most tasks would involve visual wake words rather than live stream person recognition and audio wake words rather than entire speech recognition.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Consequently, we raise the following two questions: is it worthwhile to use complex deep learning algorithms for these small scaled tasks and is it even worthwhile to do inference on MCUs. In order to figure out the practicality of running deep learning on MCUs, we have selected a commonly used and accessible micro-controller: Arduino Nano 33 BLE Sense. Then, we have chosen to run on our micro-controller on speech detection with limited library, a task that is reasonable in its requirement of model size and computational power, under three different experiment and model settings.

First, referring to previous research done in TinyML [14], we have designed our baseline model that trains a simple Convolutional Neural Network (CNN) through Tensorflow Lite Micro [7] for model inference on board. Then, we have experimented different neural networks with variable size and complexity, with which we use the results to compare the tradeoff in terms of accuracy, latency and RAM usage between models. From our comparsion, we then determine a model that produce best result on the Arduino Nano 33 BLE Sense.

Next, to figure out if it is necessary to use deep learning models to complete this task, we then implement a traditional machine learning model with Support Vector Machine (SVM). Since SVM has much fewer parameters and operations compared with neural nets, we are able to directly define inference functions instead of using inference packages.

Finally, we evaluate the performance of doing inference on another machine and send the result to the MCU to verify whether there is major improvement from embedding the inference model on the MCU. Specifically, we utilize the MCU to collect live data, which we then send to a local computer for model inference through our implementation of SVM. The inference output would then be directly sent back to the MCU to complete the task.

Due to the limitation in time, we are only able to evaluate the accuracy-latency trade off between models. The results of our three experiments offer us a relatively optimal solution for the board and task we choose and, most importantly, a clearer view of future research directions on doing machine learning on MCUs.

For rest of the paper, section 2 discusses the background of the MCU board and ML task we chose for the project. Then, section 3, 4, and 5 elaborates the three experiment and model settings we implemented as described above. Finally, we discuss our experiment results, conclusion, and potential future work in section 6 and 7.

2 BACKGROUND

2.1 Arduino Nano 33 BLE Sense

The Arduino Nano 33 BLE Sense board features the nRF52840 processor from Nordic Semiconductors and a 32-bit ARM Cortex-M4 CPU running at 64 MHz, making it a significantly more powerful processor compared to the traditional Arduino Nano boards. The board runs at 3.3V and contains a series of embedded sensors that is able to detect humidity, pressure, and temperature through its sensors and detect sound through its microphone. BLE in the board name stands for bluetooth low energy, with which the board supports an efficient transmission of data between itself and other bluetooth devices. The board is widely used in many IoT applications, especially for wearable devices. Despite the wide range of capabilities that Arduino Nano 33 BLE Sense offers with its sensor, the board is heavily limited in computational power and memory compared to devices such as mobile phones and computers, with a CPU flash memory of 1MB and SRAM of 256KB.

Implementing and deploying an arduino program is made simple through arduino's provided IDE. The language for implementing arduino programs can be seen as a subset of C++, and finished programs can be directly compiled and sent for execution to the board through the IDE by connecting the board to the computer [3]. The microphone, bluetooth connection, and exceptional IDE support makes the board an ideal choice for our project's microspeech recognition task.

2.2 Speech Command

We want to find a task that is reasonable in scale for MCU and has some decent use cases. Speech recognition on limited vocabulary is a common task for IoT devices especially in doing audio wake words. Amazon, Apple, and others use tiny neural networks on billions of smart home devices to run always-on inferences for keyword detection. Since the vocabulary is limited, the task should be manageable even for MCUs.

We benchmark the performance on the Google Speech Commands dataset [17]. Google Speech Commands dataset is an audio dataset that consists of spoken words with different accents and background noises. It is commonly used to build and test machine learning models that detect when a single word is spoken, from a set of few target words. There were a lot of previous ml researches [18] working on this dataset, where recent state-of-the-art architecture [11] could reach an accuracy of 98.7%. However, only a few of theses models could be directly applied on MCUs with such small memory size.

Within the scope of this project, six words were chosen as our target set {yes, no, up, down, left, right}. We will experiment various models and try to find the most suitable one for our board, Arduino Nano 33 BLE Sense.

3 DEEP LEARNING WITH TENSORFLOW LITE MICRO

3.1 Tensorflow Lite Micro

Utilizing machine learning models often consists of two general steps, training the model and model inference. Due to the low computing power of MCU, it is unrealistic to train a deep neural net but it is possible to do model inference with the help of Tensorflow Lite Micro [7].

There are a couple of key challenges in deploying a deep learning model on board:

(1) Limited resources, for example the Aruduino board we are using only have a 32 KB RAM and a 1 MB Flash, which is definitely not enough for deploying a common ml models we are using today.
(2) Embedded systems always have a lot of limitation on software features due to the hardware restrictions. For example, floating points operations or dynamic memory scheduling, which are fundamental to usual programmers, might not be supported on embedded systems.
(3) Deep learning is a fast changing and evolving field, and there are lots of different operations in different model architecture. For example, tensorflow now have more than 1400 operators in total.

Due to these challenges, it is unrealistic for us to build new inference library for deep learning within couple of weeks, while Tensorflow Lite Micro already provide some solid solutions to the challenges we mentioned and can be directly used off the shelf. There are certain limitations on Tensorflow Lite Micro (e.g. rely on interpreter, which consume more SRAM and Flash; optimizations are only performed to layer level, etc. [12]). However, for the purpose of this project, we think it is already good enough to give us an idea of the performance of deep learning model inference.

3.2 Quantization

The biggest concern of utilizing deep learning models is the large number of parameter it used, which could easily exceeds the memory capacity. Quantization is a commonly used method [9] [19] to bring the neural network to a reasonable size, without reducing lots of accuracy. In deep learning, quantization is the process of approximating the original neural network with floating point numbers with a network that uses low bit width numbers. Such technique could alleviate both challenges 1 and 2 that we mentioned above, as dealing with integers would require much less computing power and memory than floating points. Moreover, quantization-aware training, the technique where we quantizing the model weights during training, has shown that it will not reduce the accuracy by a lot [1] and it will be demonstrated in our experiments.

3.3 Models

We use Mel Frequency Cepstral Coefficients (MFCC) to preprocess the input WAV file data. Basically, MFCC metrics helps us to transform the waveforms from the time-domain signals into the time-frequency-domain signals by computing the short-time

Fourier transform (STFT) to convert the waveforms to as spectrograms. The MFCCs are the amplitudes of the resulting spectrum which will be used as the input features for the machine learning models.

We experiment the following four specific neural net architectures:

(1) The first model we choose to use is a 2D Convolutional neural network [4]. The architecture of the model are two convolution layers with max pooling and followed by a dense layer for classification. We choose this model as a benchmark for our performance evaluation as CNNs are commonly used for audio tasks with spectrograms.
(2) The second model is fc4, which basically consists of four fully connect layers with dropout and batch normalization. Such architecture could be one of the easiest and smallest neural network to use.
(3) The third model is MobileNet [15], which was specifically designed for doing inference on mobiles. We want to see if it is still applicable on an even smaller MCU.
(4) The fourth model is DS-CNN, which is suggested as a benchmark for the Google Speech Commands dataset [5]. Another reason I choose this model is that among the models with highest performances (DS-CNN could achieve an accuracy of about 95% on the original dataset), it is also the one with relatively few parameters.

Finally, after experimenting with different feature sizes and different architectures, we configured a 1D-CNN network, consists of two 1D-convolution layer, which we think is the most suitable for our task and Arduino board.

3.4 Evaluation

We evaluate the model in three aspects: accuracy, running time, and peak RAM usage. Note that data preprocessing are the same for all models, so the preprocessing time should be pretty similar (about 80ms). Therefore, instead of comparing running time, we could just compare the inference time. Additionally, even for a single model, we can choose different model size and different input feature size. Generally, this cause a latency-accuracy trade-off where larger model could give higher accuracy but higher inference time.

The table includes the result of running the four models. The results are not fine tuned and only include one of the specific parameter settings. However, through these results, we could make the following conclusions:

- Quantization is critical in doing deep learning on embbeded systems. As we can see even under same settings, using quantization could significantly decrease inference time and RAM usage.
- Without quantization, MobileNet will exceed the the RAM limit and even with quantization, the inference time is way too long. Therefore, we cannot directly apply MobileNet on micro-controllers.
- While ds-CNN has a really high accuracy, the 2201ms inference time is till a bit too long for doing continuous audio keyword detection. We tried other parameter settings for the model, but the inference time cannot decrease a lot due to the relatively complex structure. Therefore, we believe for a simple task like six keywords speech recognition, it is not reasonable to use large and deep neural nets.
- Since CNN model gives much higher accuracy than fc4 model, we believe using small-scale CNN is a nice way for this specific task. Finally, after trying different structure and different number of parameters, we propose the quantized 1D-CNN, which gives a pretty satisfying accuracy (from the confusion matrix we can see the accuracy for every command is above 89%) and very low inference time.

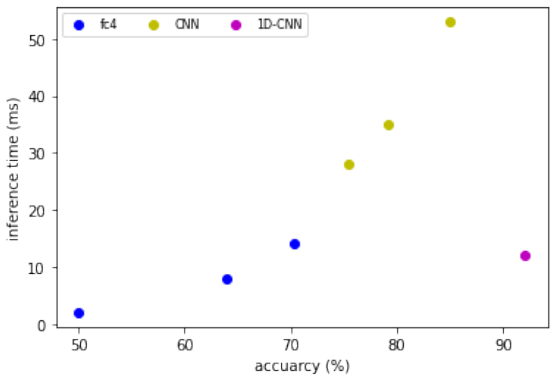

Figure 1: Scatter plot for models with different parameters

	DOWN	LEFT	NO	RIGHT	UP	YES
DOWN	91.9%	1.6%	4.8%	0%	1.6%	0%
LEFT	0%	91.2%	0%	1.8%	0%	7.0%
NO	3.1%	0%	93.8%	0%	1.5%	1.5%
RIGHT	0%	5%	0%	95%	0%	0%
UP	4.7%	3.1%	1.6%	1.6%	89.1%	0%
YES	0%	6.4%	2.1%	0%	0%	91.5%
F1 SCORE	0.92	0.88	0.93	0.96	0.93	0.91

Figure 2: Confusion matrix for 1D-CNN

Model name	Accuracy	Inference time	RAM usage
2D CNN	79.2%	115ms	28.1k
2D CNN quantized	78.2%	25ms	9k
fc4	70.5%	52ms	7.1k
fc4 quantized	70.3%	14ms	4.6k
MobileNet	85.6%		
MobileNet quantized	85.8%	19043ms	211.2k
ds-CNN	93.3%	14480ms	97.9k
ds-CNN quantized	93.2%	2201ms	36.6k
1D-CNN quantized	92.1%	12ms	5.1k

4 SVM ON BOARD

With their complex model structure, the neural networks in section 3, if not due to the memory and computational constraint of MCU,

are capable of classifying significantly more difficult speech recognition than the simple words specified in our limnited speed detection task. Simple machine learning models, while not as scalable as neural networks in terms of task difficulty, require significantly less computational complexity and parameters for storage, the limitation for both of which are our primary concern in implementing ML models on MCU. To evaluate the necessity of using complicated neural nets for MCU tasks, we have implemented an SVM model to compare its accuracy and latency with the previously analyzed neural networks. The model structure and evaluation results are described in the sections below.

4.1 Model Structure

Our model is implemented through sklearn's svm package [13] with a second degree polynomial kernel. Similarly for training the neural net models in the section above, MFCC is used to preprocess each audio file for our model. Due to SVM model's inherent quadratic runtime with respect to the data points, we have only selected a subset of voice samples from the speech command dataset due to the constraint of time; specifically, 64 samples of each word category are randomly selected for training.

4.2 Evaluation

Our simple SVM model has achieved a training accuracy of 85.2%. Then, we loaded the model into our arduino board to verify the real time prediction accuracy through us playing 10 randomly selected audio sample from each word category into the microphone of the board and manually verify the model inference. Since the speech recognition dataset is considerably larger than the subset we have picked for training data, the probability of us using one of the training samples for evaluation is negligible. Overall, we have achieved a 68.3% accuracy of the model, which we believe the discrepancy comes from insufficient training data, simplicity in SVM structure, and the unavoidable difference between the recorded sample and testing input due to background noise.

The latency of the model is determined by the time difference between the post processing of sound in the microphone for the current voice sample and our model making the final inference output. Out of the 20 word pronunciations in our evaluation, we measured the average latency for our SVM model to be 8.1 milliseconds.

5 INFERENCE ON REMOTE MACHINES

So far, we are forced to compromise the prediction accuracy of ML models through either oversimplifying trained models or picking simple models from the start due to the memory and computation power constraint of the MCU. However, such concessions can be completely avoided through the inherent implementation of IoT devices that allow them to communicate with other machines where such limitations are not an issue.

Intuitively, the communication mechanism between devices has the drawback of introducing additional latency and energy consumption that could undermine the possible accuracy and speed gain of running inference models on a machine with greater computational power and capacity. However, especially with the prevalence and considerable computational power in modern mobile devices, the communication channel between these IoT devices running on MCU to a more computationally capable device (i.e. smartphones) is often not long. These short channels, along with modern technology in reducing energy consumption for communication such as BLE, would significantly reduce the impact of latency and energy consumption from running model inference on remote machines.

As a result, we have implemented a model for the limited vocabulary speech regonition task on our Arduino Nano 33 BLE Sense where inference is performed on a computer to inspect the actual performance tradeoffs for running inference models remotely. Details of the model implementation and our evaluation results are further discussed in the sections below.

5.1 Model Structure

An illustration of our remote inference model is shown in Figure 3. Upon receiving audio on its microphone, the arduino board sends the input to a computer nearby to run the inference task. In order to maintain consistency to previous model structures, the received audio samples are first converted from the WAV format through MFCC on the board. The result of the inference is then sent back to the board from the computer to produce responses according to the classified result. The communication between the model is done through our board's BLE support, and we have manually set the distance for communication to be 1 meter, which is approximately the maximum distance between people's wearable smart devices and their carried phones. Finally, we reused the SVM classifier implemented in section 4 to help visualize the performance tradeoff of running the model inference on remote devices.

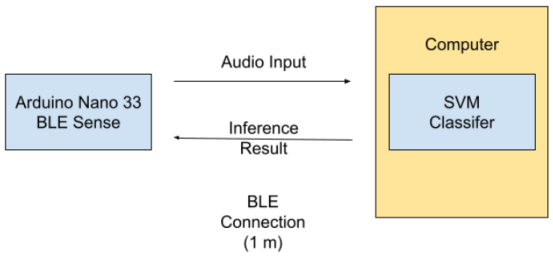

Figure 3: Remote inference model layout

In order to construct the bluetooth connection between our arduino board and computer, we utilized python's bleak library [2] to establish an asynchronous platform for communication. Then, we have adapted the approach by [6] to create a Generic Attribute Profile (GATT) service for the microphone and specify the send and receive characteristics of the service for the bluetooth connection on the Arduino IDE. The service and the characteristics are specified by their Universally Unique IDs (UUID) which our computer is able to match for sending and receiving information from the board. The captured sound sample from the microphone is sent to

the computer as a bytearray, which the computer would first need to convert the array to the corresponding root mean square value needed for the inference model. The final inference output is then also converted back to bytes for our arduino board to receive. Since the microphone input is sent in a stream manner, model inference and result sending is done asynchronously on the computer end.

5.2 Evaluation

Since the same SVM model is used for our remote inference model than the model on MCU described in section 4, there will be no prediction accuracy gains. Instead, we focus on the response latency of our model to evaluate the impact of additional overhead from communication with a remote device. Similarly to section 4, the response latency is determined by the time difference between immediately after the board finished processing voice sample and the production of response after receiving the model inference result. We evaluated the model through 20 word pronunciations, and resulted in an average latency of 21.7 milliseconds.

When measuring the average response latency of the entire model, we also measured on the computer side the time needed to complete the model inference to determine the latency by the communication channel. From the 20 word pronunciations, we computed the average latency to be 0.8 milliseconds, which is as expected due to the significant increase in computation power our computer has compared to the Arduino board. As a result, we are able to approximate the average latency of the communication channel to be 19.9 milliseconds.

6 DISCUSSION

From the evaluation results of our three experiments, we are able to respond to the two initial questions raised in section 1: The necessity of using deep learning algorithms for MCU applications and the necessity of implementing inference models on MCU boards.

Comparing the performance of our selected quantized 1D-CNN model to our implementation of the simple SVM model in the table below, we are able to see that neural networks are able to perform considerably better in model prediction accuracy while having a similar inference latency, showing that running deep learning algorithms regarding MCU applications is worthwhile. However, from the result shown in 3.4, it is also highly nontrivial to determine a deep learning model implementations for MCU tasks as naive model with no optimizations such as MobileNet suffer from very high latency and RAM usage, making them impractical.

Model name	Accuracy	Inference time
1D-CNN	92.1%	12ms
SVM	68.3%	8.1ms

For our second question, we report the latency of our SVM on board and the latency of running SVM on a remote computer in the table below. From the table, we can see that we are able to obtain a 10x speed up running inference on the computer. While in the real world, the remote computing devices would be more likely to be done on phones for most MCU powered IoT devices, we believe running inference models on these devices would still achieve a substantial speed up due to the drastic increase in computation power and memory restriction. From the table, we can see the communication overhead for bluetooth is about 20ms, which is negligible compared to the inference speed up, especially for more complicated neural net models described in section 3 that have inference times with thousands of milliseconds. Running model inference remotely also eliminates the limitations in memory and computation power of the MCU, making us able to achieve more complicated tasks.

Model name	Inference time	Communication Overhead
SVM on board	8.1ms	N/A
SVM	0.8ms	19.9ms

7 CONCLUSION AND FUTURE WORK

In this project, we explored the possibility of doing machine learning task on micro-controllers, specifically audio keyword recognition on the Arduino Nano 33 BLE Sense. We first experiment different Neural Net neural net architectures and use Tensorflow Lite Micro as model inference library. Then we tried SVM, as a typical traditional ML algorithm, and implement the inference on board by ourselves. Finally, we explore doing model inference on a remote machine and use BLE to communicate. After comparing the model accuracy and latency, we realize for a relatively simple task, like limited keywork recognition, using simple neural net is the relatively best option. Additionally, since communication using BLE does not have a really high latency, we believe it is also reasonable to use bluetooth and do the ml task on a machine with less computation and memory constraint. A demo of the implementation can be found: https://github.com/YoungY0Y/ml-on-IoT/blob/main/demo.mp4

Since this is only a six-week project, there are a couple of things that we think are worth doing in the future:

- Take energy consumption into consideration while evaluating different metrics
- Further experiment with different types of ML model
- Try using wifi connection to connect with cloud computing for doing ML task
- Further experiment with more complex task and microcontrollers with different configurations

ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

REFERENCES

[1] [n. d.]. Model optimization nbsp;: nbsp; Tensorflow Lite. https://www.tensorflow.org/lite/performance/model_optimization
[2] [n. d.]. Python Bleak Package. https://github.com/hbldh/bleak.
[3] Arduino. [n. d.]. Arduino Software. https://www.arduino.cc/en/software.
[4] Sercan O. Arik, Markus Kliegl, Rewon Child, Joel Hestness, Andrew Gibiansky, Chris Fougner, Ryan Prenger, and Adam Coates. 2017. Convolutional Recurrent Neural Networks for Small-Footprint Keyword Spotting. arXiv:1703.05390 [cs.CL]
[5] Colby R. Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, David Patterson, Danilo Pau, Jae sun Seo, Jeff Sieracki, Urmish Thakker, Marian Verhelst, and Poonam Yadav. 2021. Benchmarking TinyML Systems: Challenges and Direction. arXiv:2003.04821 [cs.PF]
[6] C.Thomas Brittain. [n. d.]. How to Send Data between PC and Arduino using Bluetooth LE.

[7] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Shlomi Regev, Rocky Rhodes, Tiezhen Wang, and Pete Warden. 2021. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. arXiv:2010.08678 [cs.LG]

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[9] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. arXiv:1510.00149 [cs.CV]

[10] Farhad Hossain, Md. Liakot Ali, Md. Zahurul Islam, and Hossen Mustafa. 2016. A direction-sensitive fall detection system using single 3D accelerometer and learning classifier. In *2016 International Conference on Medical Engineering, Health Informatics and Technology (MediTec)*. 1–6. https://doi.org/10.1109/MEDITEC.2016.7835372

[11] Byeonggeun Kim, Simyung Chang, Jinkyu Lee, and Dooyong Sung. 2021. Broadcasted Residual Learning for Efficient Keyword Spotting. arXiv:2106.04140 [cs.SD]

[12] Ji Lin, Wei-Ming Chen, John Cohn, Chuang Gan, and Song Han. 2020. MCUNet: Tiny Deep Learning on IoT Devices. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[14] Daniel Situnayake Pete Warden. 2019. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, Inc.

[15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2019. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381 [cs.CV]

[16] Emanuele Torti, Alessandro Fontanella, Mirto Musci, Nicola Blago, Danilo Pau, Francesco Leporati, and Marco Piastra. 2019. Embedding Recurrent Neural Networks in Wearable Systems for Real-Time Fall Detection. *Microprocessors and Microsystems* 71 (2019), 102895. https://doi.org/10.1016/j.micpro.2019.102895

[17] P. Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *ArXiv e-prints* (April 2018). arXiv:1804.03209 [cs.CL] https://arxiv.org/abs/1804.03209

[18] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2018. Hello Edge: Keyword Spotting on Microcontrollers. arXiv:1711.07128 [cs.SD]

[19] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. 2017. Trained Ternary Quantization. arXiv:1612.01064 [cs.LG]