Inference speed and quantisation of neural networks with TensorFlow Lite for Microcontrollers framework

Kristian Dokic
Department of Social science
Polytechnic in Pozega
Pozega, Croatia
kdjokic@vup.hr

Marko Martinovic
Technical Department
College of Slavonski Brod
Slavonski Brod, Croatia
marko.martinovic@vusb.hr

Dubravka Mandusic
Department of Information Science and Mathematics
Faculty of Agriculture
Zagreb, Croatia
simunovic@agr.hr

Abstract— **In the last few years, microcontrollers became more and more powerful, and many authors have started to use them for different machine learning projects. One of the most popular frameworks for machine learning is TensorFlow, and their authors began to develop this framework for microcontrollers. The goal of this paper is to analyses the full connected neural networks inference speed depending on the number of neurons of one popular microcontroller (Arduino Nano 33 BLE Sense) with simple neural networks implementation, as well as the impact of neural network weights quantisation. We expected a reduction in the size of the model with the selected quantization by four times, which was achieved, but with a large number of neurons in the neural network. TensorFlow Lite for Microcontrollers is used with the Arduino Integrated Development Environment. Neural networks with two hidden layers are used with a different number of neurons.**

Keywords — Propagation speed, Quantisation, Arduino, Microcontrollers, neural network

I. INTRODUCTION

The first microcontroller is TMS 1000, designed by G. Boone and M. Cochran in 1971. It was cheap, and it cost $2 in bulk orders. It has had great success in toys, games, calculators and alarms, so more than a hundred million have been sold. It had 32 Bytes of RAM and 128 Bytes of ROM memory [1].

Almost fifty years later, microcontrollers are still cheap, but they are much more powerful than before. Amount of RAM is much higher as well as computing power. Microcontrollers with 200 kB of RAM today can recognise 10 to 20 different spoken words [2].

But in the last few years, after microcontrollers *evolution* that last almost fifty years, it is evident that a new *revolution* is taking place. That new *revolution* is manifested by the implementation of machine learning and artificial intelligence on microcontrollers. There are more and more scientific papers that deal with machine learning on microcontrollers. On the other hand, different software corporations and microcontroller manufacturers (Microsoft, Google, ST Microelectronics,...) started to offer their solutions to implement machine learning on microcontrollers. IoT paradigm further accelerates this *revolution* [3] [4] [5].

Branco et al. published a comprehensive analysis of machine learning in resource-scarce embedded systems. They made the list of algorithms and machine learning models implemented in the resource-scarce microcontrollers and here are some models: kNN, Decision Tree, ProtoNN, BONSAI, MLP, Naive Bayes, Infinite Hidden Markov, ANN and DNN. On the other hand, they categorised the optimisation techniques used in the implementation into groups: Test Until It Breaks, Minimalist, Goal-Oriented, Compression, Platform, Bit Architecture, Floating-Point Calculations, Environment, Know the Algorithm, Cloud and Model Mesh [6].

In this paper one of the most popular frameworks for machine learning, TensorFlow Lite for Microcontrollers will be analysed. One simple, fully connected neural network with two hidden layers will be created, and it will be deployed on the microcontroller to test inference speed and the level of optimisation obtained by quantisation. The number of neurons in the hidden layers will be changed from 16 to 768.

The paper is structured as follows. In Section II, related papers that deal with machine learning on different microcontroller platforms is presented. In Section III, materials and methods are presented. This section is divided into hardware description and software development. In Section IV, testing and results are described. This section is divided on propagation speed and quantisation part because they are analysed separately. Finally, discussion and conclusions are in section V.

II. RELATED WORK

A. *Inference speed*

It is well known that machine learning algorithms need massive compute resources, especially when they „learn". Usually, workstations for machine learning have powerful GPUs, but the bottleneck of implementing machine learning algorithms on microcontrollers is the small amount of computing resources. On the other hand, advantages like improved security and privacy, reduced amount of data transfer and reduced latency of prediction guarantee further development and use of machine learning on microcontrollers.

Branco et al. concluded that time is a constraint in almost any system. Microcontroller speed depends on the clock frequency but on different microcontrollers the number of clock cycles to finish machine cycles can vary. Some operation can take more time than others. Finally, they counted five factors that affect execution time: memory type, clock frequency, architecture, floating-point calculations and data representation [6].

Fedorov et al. at the beginning of their paper noted that different processors have different compute characteristics. They cite that Nvidia 1080Ti GPU has 10 TFLOPs/Sec, ARM CPU in Raspberry Pi has 50 GOPs/Sec, but ATmega328P

978-1-7281-6445-8/20/$31.00 ©2020 IEEE

from Arduino Uno has only 4 MOPs/Sec. They designed optimised convolution neural network for some microcontrollers, and they reached more than 95% accuracy with ten-class MNIST problem. Latency on microBit 16 MHz ARM Cortex-M0 is 115,40 ms, but on STM32F413 100MHz ARM Cortex-M4 is 27,06 ms [7].

Gural and Murmann also proposed a strategy for optimal memory convolutions on microcontrollers, and they reached the classification accuracy of 99.15% with Arduino implementation of the ten class MNIST classification task. They fit the weights, network specifications and activations within 2KB SRAM. Their inference time was 684 ms. The microcontroller on Arduino board was ATmega328P on 16MHz [8].

Gupta et al. used the same Arduino board with ATmega328P 16MHz and tested their ProtoNN model inspired by k-Nearest Neighbor. They reached the classification accuracy of 96.5% with the ten class MNIST classification task. They also fit the weights, network specifications and activations within 2KB SRAM. Their inference time was 93.29 ms [9].

Bayerl et al. used ARM HiKey 960 development board equipped with an ARMv8 octa-core SoC 2.4 GHz and 3 GB of RAM. They used it to develop and test a model for a speaker's identity verification. Their model consists of 2D convolutional layer and eight filters. They trained a system for a 12-class problem, and finally, their model was 49KB in size. Model inference time with ten words sample was 387ms with proposed OMG protection [10] [11].

B. Quantisation

Neural networks are usually trained using 32-bit floating-point data, but some authors have shown that high precision is not required in production/inference mode. Proces of reducing the number of bits that can be used to make a neural network model faster and smaller is called quantisation. It has been demonstrated that activation and weights can be converted to 8-bit integers. Some authors tried to convert the 32-bit floating-point to 4, 2 or 1-bit, and they also made a great success.

Many papers deal with a quantisation. Fielser et al. wrote about quantisation in neural networks in the 1990s. The main reason to quantise neural network models before three decades was to easier connect various hardware devices to computers with neural networks. Today situation is different, and the main reason for quantisation is the huge size of neural networks and the tendency to use them on less powerful processors. Guo concluded that „quantised neural networks promote the application of deep learning models in mobile devices and embedded systems“ [12] [13].

Many authors suggested that quantisation is essential for real-world embedded applications with limited computing power. Lin et al. converted floating-point model of deep convolution network to a fixed-point model with optimised bit width. They reached more than 20% reduction in the model size without a loss in accuracy [14].

Sinha et al. used quantisation when they developed a neural network for a low-power edge device for a robotic system in agriculture They decided to use microcontroller board based on 32-bit ARM Cortex-M4F with 48MHz clock, 384 KB RAM and 1 MB flash memory. They also used TensorFlow Lite for Microcontrollers for the inference part, but the model was trained using Keras. The trained model that consists of weights and biases in floating-point format was converted to a TensorFlow Lite FlatBuffer file. This file consists of 8-bit weights in integer format, and after that conversion that includes quantisation, the file was converted to a C byte array. That model was deployed to the device, and the model size was 65KB [15].

Fedorov et al. in the previously mentioned paper about optimised convolution neural network also used quantisation. They converted floating-point weights and activations to 8-bit integers [7].

Gupta et al. from previously mentioned paper about ProtoNN model also converted floating-point weights and activations to 8-bit integers. They noted that Atmega328P microcontroller integer arithmetic takes about 0.1µs/operation, but software-based floating-point arithmetic takes about 6µs/operation. They also store a table of approximate exponential values to avoid computing the exponentials [9].

Lai and Suda proposed CMSIS-NN as the answer to neural networks implementation challenges on microcontrollers. „CMSIS-NN is a collection of efficient neural network kernels developed to maximise the performance and minimise the memory footprint of neural networks on Arm Cortex-M processor cores targeted for intelligent IoT edge devices.“. Authors riched 4.6X improvement in runtime, and they used fixed-point quantisation. They used a unique way of quantisation with N as a fixed fraction length for every layer. That allows them to change range and step [16] [17].

Some authors have not stopped with 8-bit integers, and there are papers with CNNs implementations with only two bits weights [18] [19].

III. Materials and methods

This section consists of two parts. In the first part, Arduino Nano 33 BLE Sense microcontroller is described. In the second part of the section, software development is described.

A. Hardware part

Arduino Nano 33 BLE Sense has been used in this paper for testing purposes. This microcontroller board was announced in 2019 [20]. It is powered by nRF52840 processor (Cortex M4F). The clock is 64MHz, and the board has 256 KB SRAM and 1 MB flash memory. It also has implemented many interfaces, like USB, I2S, I2C, SPI, UART [21]. It can be seen in Figure 1.

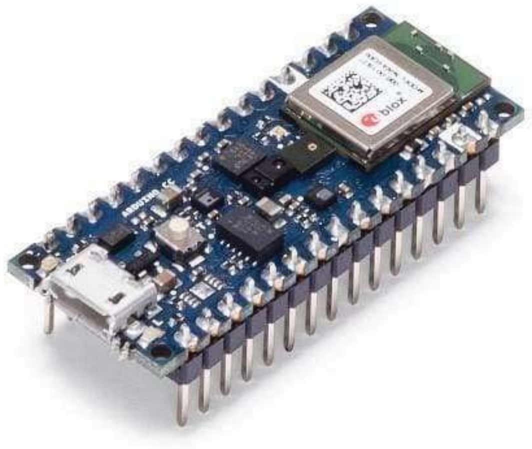

Fig. 1. – Arduino Nano 33 BLE Sense

B. Software development

In the paper, TensorFlow Lite for Microcontrollers (TFLM) has been used. TensorFlow is an interface and an implementation for executing machine learning algorithms presented in 2015 [22]. The Lite version for Microcontrollers is an experimental port of TensorFlow Lite and primarily designed for microcontrollers with only a few KB of memory. One of the main advantages of TFLM is that existing TensorFlow environments can be used for development, training and testing. If the final deployment device is a microcontroller, then a model has to be converted to TensorFlow Lite and finally to TFLM data structures [23].

On the Figure 2 training workflow with TensorFlow Lite for Microcontrollers can be seen. The first step is training with Keras or some other framework. When results are acceptable, then a model can be saved. After saving, a model has to be converted to TF Lite format [24].

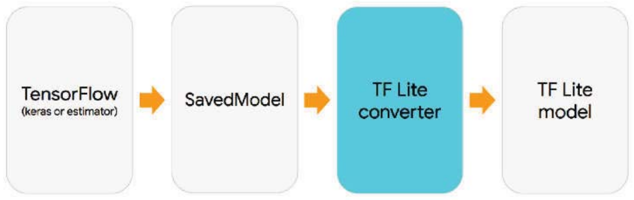

Fig. 2. – Training workflow [24]

The model has to be deployed to the microcontroller, and some data has to be available for processing. Finally, some output data are results of TF Lite Interpreter calculations as it can be seen in Figure 3 [24].

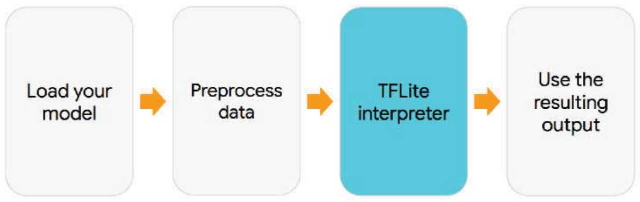

Fig 3. – Inference workflow [24]

In this paper above described training and inference, workflows have been used, and for testing purposes, a model of a sine function has been trained. It is one of the so-called „hello world“ examples, but it is sufficient to analyse the influence of quantisation and the number of neurons on the speed and size of the neural network inference. The first part that can be seen in Figure 2 has been done in the Google Colaboratory environment. It is available on the address: https://github.com/kristian1971/TFML_SIN_milis . The file name is create_sine_model.ipynb, and it is very similar to the example from TensorFlow Lite repository. The first step was a sine function generation and adding some noise. It can be seen in figures four and five.

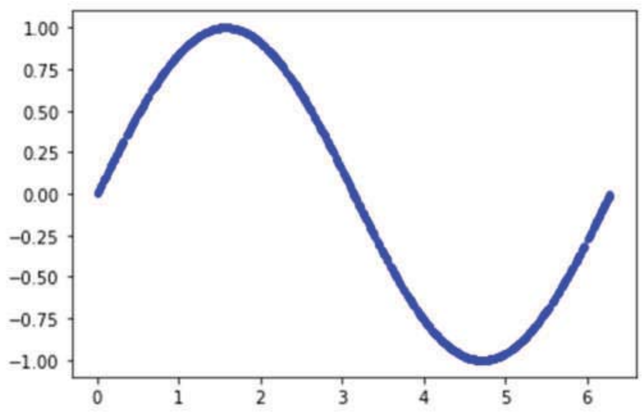

Fig 4. – Sine function generated in Google Colaboratory

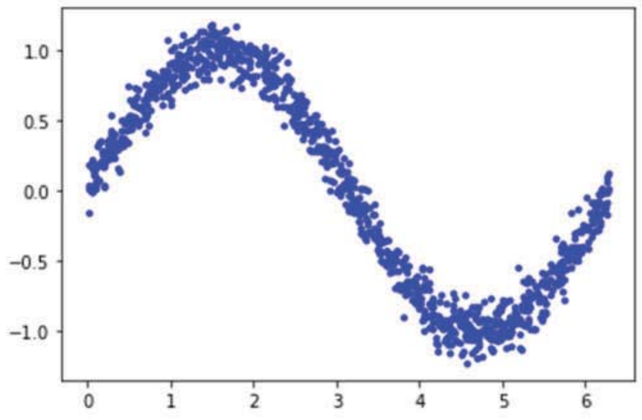

Fig 5. - Sine function with added noise generated in Google Colaboratory

These data have been used to train a neural network that has been created using Keras. A neural network is simple. It is fully connected with one input and one output value. It has two hidden layers with sixteen neurons in every layer with relu activation functions. Structure of a neural network can be seen in figure six. A neural network has been trained for only 600 epochs because accuracy is not objective of the paper.

After training the model was in computer memory and it must be converted to TensorFlow Lite format with the TensorFlow Lite Converter. The converter can quantise the model, and it can be done with „OPTIMIZE_FOR_SIZE“ key before conversion. Both models have been saved for a particular neural network structure, and size difference has been analysed.

The final step in the training part is a conversion from TensorFlow Lite format to a C source file. A C source file can be made with *xxd* tool, and it is used because many microcontrollers platforms do not have support for file storage. The *xxd* tool is a part of TensorFlow Lite converter Python API, and it produces a C char array that can be compiled [4].

To analyse propagation speed depends on a number of neurons, neural networks with a different number of neurons in hidden layers have been tested. The number of neurons that have been used on the beginning was 16 in both hidden layers, and this model can be seen in Figure 6. The highest number of neurons was 768.

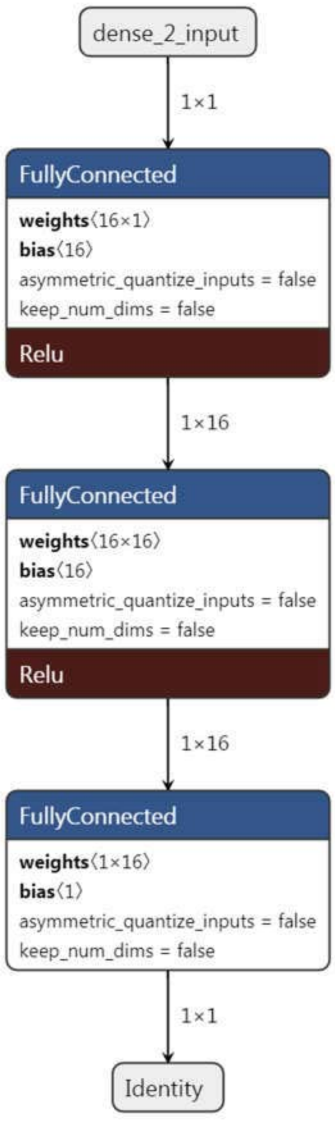

Fig. 6 – Neural network structure

Model deployment to the microcontroller has been done with Arduino IDE as it can be seen in Figure 3. There are two repositories with files on GitHub. First one is with code that measures propagation speed with *milis()* function, and it is on the address: https://github.com/kristian1971/TFML_SIN_milis . The second repository is with code that is used with an oscilloscope, and it is on the address: https://github.com/kristian1971/TFML_SIN_oscilloscope .

IV. Testing and Results

A. *Propagation speed*

Two methods have been used for propagation/inference speed measuring. First one is the software method, and the second one can be called the hardware method. The second one has been used to prove the accuracy of the first method.

First method uses function *micros()* that counts microseconds from microcontroller reset. To measure time between two events, the difference in the state of the microsecond counter should be calculated. Here is the code for measuring time:

```
first = micros();
TfLiteStatus invoke_status = interpreter->Invoke();
second = micros();
timediff = second - first;
```

Second method is with oscilloscope. Here is the code for microcontroller pin state changing:

```
if (led==1)
        {digitalWrite(9, HIGH);
        led = 0;}
 else
        {digitalWrite(9, LOW);
        led = 1;
```

After every data propagation one pin on microcontroller change state from HIGH to LOW or from LOW to HIGH. Oscilloscope probe is connected to this pin and frequency is measured. Oscilloscope screenshot can be seen on the Figure 7.

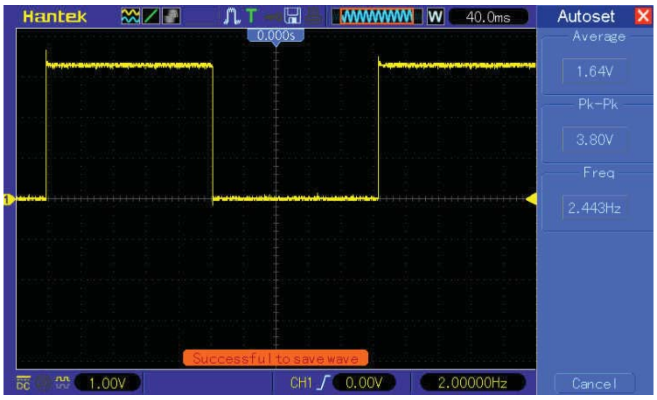

Fig. 7. - Oscilloscope screenshot

Oscilloscope test has been used to prove the accuracy of the first method and it was obviously that software method is accurate.

In table 1, results can be seen. The largest neural network that can fit in microcontroller flash memory is one with two hidden layers with 768 neurons. The propagation time for that neural network is about 0,2 seconds.

Table 1. – Average propagation time

Neurons in one hidden layer	Weights between 1st and 2nd layer	Average propagation time
16	256	192,34 µs
32	1024	529,92 µs
48	2304	1036,96 µs
64	4096	1732,54 µs
80	6400	2527,24 µs
96	9216	3558,60 µs
112	12544	4731,20 µs
128	16384	6100,82 µs
256	65536	23285,50 µs
512	262144	91709,04 µs
768	589824	205020,10 µs

On Figure 8 graph with average propagation times can be seen. Because of the most operations in a fully connected neural network that is used are between the first and the second hidden layer, linear connection between a number of weights between 1st and 2nd layer, and propagation time can be observed.

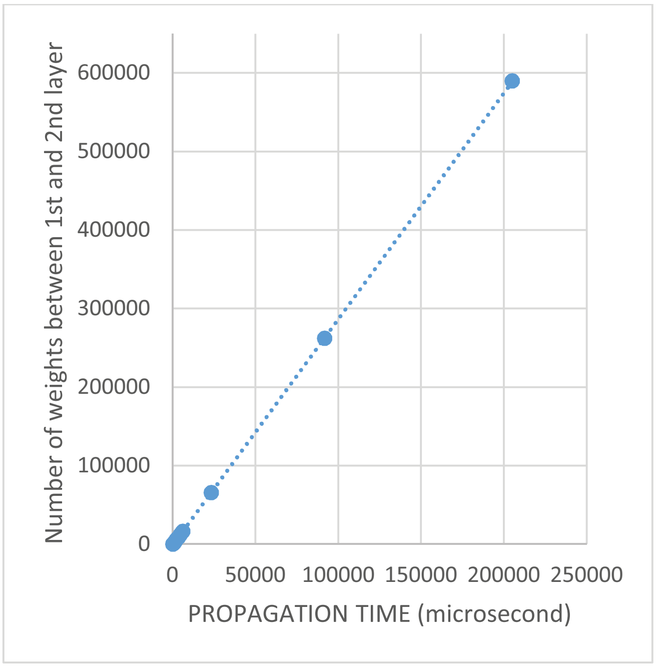

Fig. 8. – Average propagation time graph

B. *Quantisation*

Quantisation after neural network training is used to reduce the model size and microcontroller latency. There are several different option of post-training quantization that can be used in TensorFlow Lite for microcontrollers. Dynamic range, Full integer and Float16 quantization are available. In our case optimization for size has been chosen so weights are converted from 32-bit floating point to 8-bit integer. In optimization setup OPTIMIZE_FOR_SIZE has been chosen in model conversion. Expected size reduction is 4 times [22] [25].

As previously stated, most of the model data are weights between the first and the second hidden layers. If the number of neurons in the first and the second hidden layer doubles, the number of weights will be four times bigger. On the other hand, a neural network model also includes some fixed parts that do not depend on a number of neurons. It can be expected that quantisation will reduce model size four times if float32 weights are converted to int8 weights, but this fixed part will disrupt expected calculation. In Table 2, the sizes of the basic and quantised models for different numbers of neurons can be seen.

Table 2. – Model size

Neurons in one hidden layer	Basic model	Quantised model	Rate
16	2656	2640	1,006060606
32	5876	2880	2,040277778
48	11268	4432	2,542418773
64	18692	6480	2,884567901
80	28164	9040	3,115486726
96	39684	12112	3,276420079
112	53252	15696	3,392711519
128	68868	19792	3,479587712
256	267524	70992	3,768368267
512	1058052	271696	3,894249455
768	2372868	603472	3,932026672

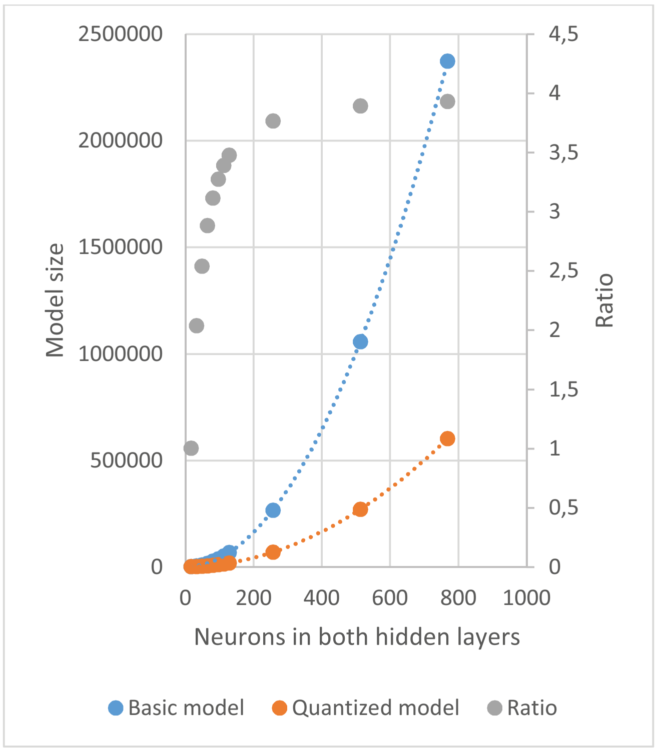

Fig. 9. – Basic and quantised model sizes

In Figure 9 graph with basic and quantised model sizes can be seen. The sizes of these two models are connected by lines, but on the secondary axis, the ratio of these sizes can be seen. The ratio is represented on the graph by the points located at the top left of the graph.

V. Discussion and Conclusion

The simple fully connected neural network has been used in the paper, and it is expected that there is a linear connection between a number of weights between 1st and 2nd layer and propagation time. The neural network has only one input and only one output, so there are no so many calculations on these levels so we can disregard this.

With collected data, the number of multiply and sum operations per second can be calculated in real conditions, and it is about 2,87 million multiplications and additions per second. It is not as fast as it could be because of at least two reasons. First, the Arduino platform includes some processes on a microcontroller that cannot be controlled, and they increase the overhead time. Second, in some situations, we cannot have a full neural network model in memory, so some time is lost for data transfer between flash and SRAM.

Model quantisation after training reduces model as it can be seen in Figure 9, but the model has to be huge to reach almost four times reduction in size. The size ratio of the regular and quantized model reached 3.5 with 256 neurons in both hidden layers. It would be interesting to research how much the accuracy of the model is reduced by applying quantization.

In future research, some other microcontrollers will be included, as well as the different development platforms. Some other machine learning models will be tested like CMSIS-NN, ProtoNN or some adapted neural networks.

VI. Bibliography

[1] S. Augarten, “The most widely used computer on a chip: The TMS 1000,” *State of the Art: A Photographic History of the Integrated Circuit,* 1983.

[2] L. Fried, “Making machine learning arduino compatible: A gaming handheld that runs neural networks-[Resources_{H}{a}{n}ds On],” *IEEE Spectrum,* vol. 56, p. 14–15, 2019.

[3] ST microelectronics, "X-cube-ai: AI expansion pack for stm32cubemx," ST microelectronics, 2019. [Online]. Available: https://www.st.com/en/embedded-software/x-cube-ai.html. [Accessed 2 May 2020].

[4] Google Inc., "For Mobile & IoT," Google Inc., 2019. [Online]. Available: https://www.tensorflow.org/lite. [Accessed 22 April 2020].

[5] Microsoft, “The Edge Machine Learning library,” Microsoft Research India, 2019. [Online]. Available: https://github.com/microsoft/EdgeML. [Accessed 3 May 2020].

[6] S. Branco, A. G. Ferreira and J. Cabral, “Machine Learning in Resource-Scarce Embedded Systems, FPGAs, and End-Devices: A Survey,” *Electronics,* vol. 8, p. 1289, 2019.

[7] I. Fedorov, R. P. Adams, M. Mattina and P. Whatmough, “SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox and R. Garnett, Eds., Curran Associates, Inc., 2019, p. 4977–4989.

[8] A. Gural and B. Murmann, “Memory-optimal direct convolutions for maximising classification accuracy in embedded applications,” in *International Conference on Machine Learning*, 2019.

[9] C. Gupta, A. S. Suggala, A. Goyal, H. V. Simhadri, B. Paranjape, A. Kumar, S. Goyal, R. Udupa, M. Varma and P. Jain, “Protonn: Compressed and accurate knn for resource-scarce devices,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.

[10] S. P. Bayerl, F. Brasser, C. Busch, T. Frassetto, P. Jauernig, J. Kolberg, A. Nautsch, K. Riedhammer, A.-R. Sadeghi, T. Schneider and others, "Privacy-Preserving Speech Processing via STPC and TEEs".

[11] S. P. Bayerl, T. Frassetto, P. Jauernig, K. Riedhammer, A.-R. Sadeghi, T. Schneider, E. Stapf and C. Weinert, "Offline model guard: Secure and private ML on mobile devices," *DATE 2020,* 2020.

[12] E. Fiesler, A. Choudry and H. J. Caulfield, “Weight discretisation paradigm for optical neural networks,” in *Optical interconnections and networks*, 1990.

[13] Y. Guo, “A survey on methods and theories of quantised neural networks,” *arXiv preprint arXiv:1808.04752,* 2018.

[14] D. Lin, S. Talathi and S. Annapureddy, “Fixed point quantisation of deep convolutional networks,” in *International Conference on Machine Learning*, 2016.

[15] A. Sinha, N. Kumar, M. Mohanan, M. D. Rahman, Y. Quemener, A. Mim and S. Ilić, “Quantised deep learning models on low-power edge devices for robotic systems,” *arXiv preprint arXiv:1912.00186,* 2019.

[16] Y. Zhang, N. Suda, L. Lai and V. Chandra, “Hello edge: Keyword spotting on microcontrollers,” *arXiv preprint arXiv:1711.07128,* 2017.

[17] L. Lai and N. Suda, “Enabling Deep Learning at the loT Edge,” in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018.

[18] R. Andri, L. Cavigelli, D. Rossi and L. Benini, “YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights,” in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016.

[19] A. Zhou, A. Yao, Y. Guo, L. Xu and Y. Chen, “Incremental network quantisation: Towards lossless cnns with low-precision weights,” *arXiv preprint arXiv:1702.03044,* 2017.

[20] Arduino team, “The Arduino Nano 33 BLE and BLE Sense are officially available!,” 31 July 2019. [Online]. Available: https://blog.arduino.cc/2019/07/31/the-arduino-nano-33-ble-and-ble-sense-are-officially-available/. [Accessed 17 April 2020].

[21] A. Raj, “Arduino Nano 33 BLE Sense Review - What’s New and How to Get Started?„” CircuitDigest, 08 November 2019. [Online]. Available: https://circuitdigest.com/microcontroller-projects/arduino-nano-33-ble-sense-board-review-and-getting-started-guide. [Accessed 03 April 2020].

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin and others, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *arXiv preprint arXiv:1603.04467,* 2016.

[23] TensorFlow, "TensorFlow Lite for Microcontrollers," 31 March 2020. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers. [Accessed 15 April 2020].

[24] D. Situnayake, “tinyML Summit - Advances in ultra-low power Machine Learning technologies and applications,” 28 October 2019. [Online]. Available: https://www.tinyml.org/summit/meetups/bay-area-situnayake.pdf. [Accessed 1 May 2020].