ISBN 978 -987-633-574-4

A Neural Network Framework for Small Microcontrollers

César A. Estrebou[1][0000−0001−5926−8827], Martín Fleming[2], Marcos D. Saavedra[2], and Federico Adra[2]

[1] Instituto de Investigación en Informática LIDI, Facultad de Informática, Universidad Nacional de La Plata
{cesarest}@lidi.info.unlp.edu.ar
[2] Facultad de Informática, Universidad Nacional de La Plata

Abstract. This paper presents a lightweight and compact library designed to perform convolutional neural network inference for microcontrollers with severe hardware limitations. A review of similar open source libraries is included and an experiment is developed to compare their performance on different microcontrollers. The proposed library shows at least a 9 times improvement over the implementation of Google *Tensorflow Lite* with respect to memory usage and inference time.

Keywords: Machine Learning · Convolutional Neural Networks · Microcontrollers · Framework · TinyML

1 Introduction

A few years ago it was unthinkable to implement machine learning or neural network algorithms in microcontrollers, mainly for their hardware limitations. Due to cloud computing problems [1, 2] associated with computational and storage cost, network bandwidth, response latencies, power consumption, privacy and security, Edge computing started to emerge and gradually the idea of running major algorithms on microcontrollers became a reality.

On the other hand, projections made by Statista [3] estimate that the number of IoT devices connected to the Internet by 2022 will be around 16.4 billion, implying a large computing capability with low power consumption and great potential for exploitation.

Because of this, it is extremely interesting to adapt solutions from the machine learning [4] and deep learning fields so that they can run on small devices with given hardware limitations.

Today there are online platforms such as *AlwaysAI*, *Edge Impulse*, *CartesiamAI* or *Qeexo* that perform the entire process of developing a machine learning solution on a microcontroller with minimal user intervention. Companies such as *Google*, *STM*, *Mbed*, *Adafruit* and *Sparkfun* have free tools that allow implementing models created with *TensorFlow/Keras* for a limited number of microcontrollers (mainly ARM) that require 32-bit architectures with hardware that supports floating-point instructions and even SIMD or DSP instructions.

ISBN 978 -987-633-574-4

Generally, these tools are provided by microcontroller development companies or companies that provide development kits interested in promoting their products or in paying a fee to use them fully.

As a result, this limits the implementation of machine learning solutions on a large number of microcontrollers despite their popularity, low cost or additional hardware features. There are few open source machine learning libraries initiatives and very few provide support for neural networks and even fewer for convolutional networks. In general, these alternatives, besides being incipient, usually lack support and have important limitations for the wide variety of microcontrollers available in the market.

In this context we have created a small group aimed at researching and developing machine learning software for microcontrollers with significant hardware limitations, trying to cover as many of them regardless of their architecture. This paper presents an open source C/C++ library that allows to perform convolutional neural network inference on small microcontrollers without minimum hardware requirements beyond data and program memory. It also presents a tool that adapts and transforms neural network models generated with *Tensorflow/Keras* to a C, C++ or Arduino compatible version.

This article is organized as follows. Section 1 contains this introduction. Section 2 describes the process of developing machine learning models on microcontrollers.Section 3 describes open source libraries for machine learning and presents an implementation of our own.Section 3 describes open source libraries for machine learning and presents an implementation from us. In section 4, an experiment is performed to determine the performance between libraries and compare the obtained results. Finally, in section 5, conclusions and future work are presented.

2 Machine Learning in Microcontrollers

2.1 Microcontroller Development Process

Due to memory and computational capacity limitations, building machine learning models on small microcontrollers (MCUs) is a generally an impossible process. Typically, model building is done in the traditional way on a computer and then a transformation process is applied to produce a version that can be run on a microcontroller. A schematic of the steps involved in developing a machine learning model for a microcontroller is shown in Figure 1. The process starts with model selection and parameter settings. Then the model is generated using training data to finally validate its effectiveness with test data. If the result is not satisfactory, the process is restarted by reconfiguring the parameters.

Once the model is obtained, a quantization is usually performed [5, 6] in order to reduce its size and improve performance on the microcontroller. Then a tool is used to export the model, usually in C/C++ language, together with the necessary functions to carry out the inference.

Finally, the application is compiled and if the executable fits the required data and program memory size, it is deployed on the device.If the executable

ISBN 978 -987-633-574-4

does not meet the memory requirements or behaves unstable, it is returned to the model optimization point or to the development starting point to reconfigure the model parameters.

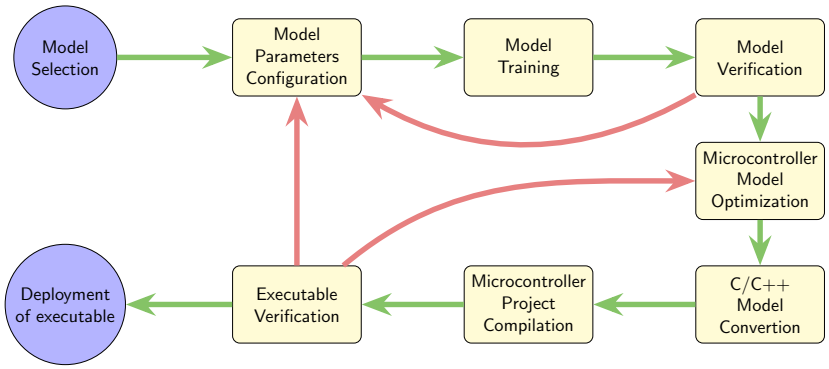


Fig. 1: Development cycle of a machine learning model for microcontrollers.

3 Neural Networks for Microcontrollers

3.1 Existing Libraries and Frameworks

In the following, this section briefly describes the open source libraries and frameworks available for the development of machine learning applications for microcontrollers.

Tensorflow Lite Micro [7, 8] for microcontrollers requires 32-bit platforms and is coded in C++ 11. It primarily supports architectures of the *ARM Cortex-M* series and has been ported to other architectures such as *Esp32*. The framework is available as an Arduino library. It can also generate projects for development environments, such as *Mbed*. It is open source and can be included in any C++ 11 project.

μTensor [9] is a lightweight machine learning inference framework built in *Tensorflow Lite* that is optimized for ARM architecture-based microcontrollers. It takes a model generated in *Tensorflow* and produces .cpp and .hpp files containing C++ 11 code to perform the inference. It does not currently support softmax functionality.

ARM CMSIS-NN [10] has a library for fully connected and convolutional neural networks named CMSIS-NN (Cortex Microcontroller Software Interface

Standard Neural Network) that maximizes the performance of Cortex-M processors with support for SIMD and DSP instructions. It includes support for 8-bit and 16-bit data types for neural networks with quantized weights.

EdgeML [11] is a library of machine learning algorithms for severely resource-constrained microcontrollers. It allows training, evaluation and deployment on various target devices and platforms. *EdgeML* is written in *Python* using *Tensorflow/Keras* and supports *PyTorch* and optimized C++ implementations for certain algorithms. Convolutional neural networks are not supported at the moment.

Eloquent TinyML [12] is an Arduino library that aims to simplify the deployment of *Tensorflow Lite* models for compatible Arduino board microcontrollers using the Arduino IDE. Starting from a model exported with *Tensorflow Lite*, this library exposes an interface to load a model and run inferences.

3.2 EmbedIA-NN, an Ultralight Library

In general, the libraries and frameworks mentioned in the 3.1 section, although they have their advantages, also have some important disadvantages, especially for microcontrollers with severe limitations. The most relevant is that they are mostly developed and optimized for specific architectures such as *ARM Cortex-M*, for 32-bit microcontrollers and/or microcontrollers with support for floating-point, DSP or SIMD instructions. This excludes devices of other architectures or devices that do not have hardware for specialized mathematical computation. Another limitation that these libraries usually have is that they are developed for C++ 11 and supported on heavy software architectures, based on objects with inheritance and polymorphism that increase the size of the programs and slow down the inference time of the algorithms. This approach may be viable for microcontrollers with good memory size and hardware resources that accelerate mathematical computation, but it is unsuitable for microcontrollers with low computational capacity and limited hardware resources.

In this article we present the development of a compact and lightweight open source library, designed for microcontrollers that are really limited both in memory and hardware. It is implemented in C, C++ and Arduino code so that it can be compiled on any platform that supports these programming languages. It provides functionalities to perform inference and debugging of the models from the microcontroller. It supports different neural network layers and activation functions including convolutional, max pooling, flatten, fully connected, ReLU and softmax. At the moment no optimizations were implemented to take advantage of advanced hardware instructions for specific microcontrollers, but there are plans to incorporate them in the future. However, optimizations are implemented for fixed-point arithmetic in 32 bits, 16 bits and 8 bits. This speeds up inferences, reduces program size and RAM usage on microcontrollers without floating point support.

ISBN 978 -987-633-574-4

In addition to the library, there is a tool that converts a model created in *Tensorflow/Keras* to C code. It also allows to generate a C, C++ or Arduino project that includes functions to perform the inference on the converted model including fixed-point optimization options.

4 Library Benchmarking Experiment

4.1 Description of the experiment

In order to determine the performance of the library, it was decided to perform an experiment by building a convolutional neural network model [13] on *Tensorflow/Keras* to recognize images of ten handwritten digits.

With the model built, a single project was developed and replicated for each library in the section 3.1 and in the four Embedia-NN implementations (8-bit, 16-bit and 32-bit floating point and fixed point). The source code for the project includes the model, the neural network functionalities to perform inference and a minimum of serial communication functionality so that each microcontroller can receive a sample and send the classification result, along with the effectiveness and time required. As part of the experiment, each project was compiled and deployed on the five selected microcontrollers. Each image of the test dataset was then submitted and each classification response was computed to determine the performance of the microcontroller-library combination. The features considered for benchmarking the different libraries were, program memory size, data memory size, inference time, and test dataset success rate.

4.2 Microcontrollers of the Experiment

The choice of the microcontrollers used in the experiment was based on aspects such as local availability, low cost, low to medium-low computational capacity and availability of open source software. Regarding connectivity it was decided to incorporate both IoT and non- IoT devices, since from the point of view of machine learning and neural networks there are many popular and interesting devices with and without this feature.

For testing purposes, 5 microcontrollers of varying characteristics were used. These MCUs are *ATmega2560*, *Arm Cortex-M3*, *Tensilica L106*, *Xtensa LX6* and *RP2040* and the technical characteristics can be seen in the table 1.

4.3 Experiment Dataset

MNIST (Modified National Institute of Standards and Technology database) is a dataset frequently used to evaluate image classification algorithms in areas of machine learning, neural networks and image processing. The chosen dataset is a reduced version of the UCI [14] repository, provided by in the *Scikit-learn* library [15]. This comprises a selection of 1797 grayscale images from the original dataset with handwritten digits centered in an 8x8 pixel area.

For model training and testing, the dataset was divided into 80An example of the dataset can be seen in Figure 2a.

ISBN 978 -987-633-574-4

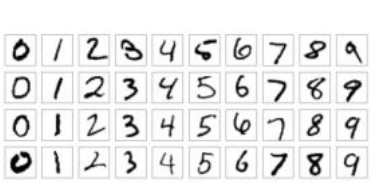
Development Board	MCU	Clock	Memory			Flot. Pt.	Connectivity
			Bits	Data	Prog.		
Arduino Mega	ATmega2560	16MHz	8	8KiB	256KiB	No	No
Stm32f103c8t6	Arm Cortex-M3	72MHz	32	20KiB	64KiB	No	No
NodeMCU ESP8266	Tensilica L106	80MHz	32	80KiB	512KiB	Si	Wi-Fi
ESP32-WROOM	Xtensa LX6	160MHz	32	320KiB	512KiB	Si	Wi-Fi+BT
Raspberry Pi Pico	RP2040	133MHz	32	264KiB	2MiB	No	No

Table 1: Relevant technical characteristics of the microcontrollers used in the experiment

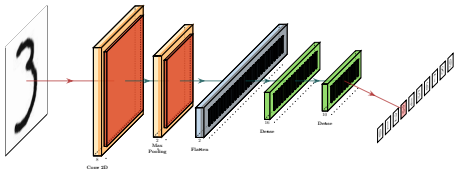
4.4 Experiment Model

A convolutional neural network (CNN or ConvNet) model [13, 16] was used to carry out the experiment tests. This type of networks are multi-layer artificial neural networks specialized in handling two-dimensional input data. Typically, their architecture is composed of combinations of convolutional, nonlinear, pooling and fully connected layers. The convolutional layer takes an image and decomposes it into different feature maps. The sequencing of various layers generates different levels of abstraction as the information progresses through the network. In the first layers low level features such as edges are obtained while in the last layers more complex and abstract structures such as parts of objects are detected. Finally the features extracted by the convolutional layers are processed by one or more layers of fully connected neurons that end up classifying the input image.

To determine the architecture of the network model we experimented with different combinations of layers in order to guarantee a good percentage of effectiveness and a low number of hyper-parameters. This last feature is of fundamental importance to maintain a small byte size to ensure that the model fits on all test microcontrollers. Figure 2b shows the architecture scheme of the convolutional neural network model generated with *Tensorflow/Keras* for testing.



(a) Digits of the MNIST dataset[15]



(b) Convolutional neural network scheme

Fig. 2: Dataset and scheme of the convolutional neural network model used in the experiment.

ISBN 978 -987-633-574-4

4.5 Comparisons and Results

The libraries *Google Tensorflow Lite*, *Eloquent TinyML*, *µTensor (microTensor)*, and *EmbedIA* in their four versions were taken to perform the tests.

It should be mentioned that two of the libraries mentioned in the 3.1 section were not considered. One of them was *CMSIS-NN* which in its official site indicates that it is possible for the library to work with processors of series prior to those supported. However, we were not able to compile the projects because it apparently requires SIMD or DSP instructions, support that the chosen microcontrollers do not have. Another one was *Microsoft EdgeML* which was also excluded from the tests because at the moment it does not support the convolutional layers included in the test model. Regarding the *µTensor* it should be mentioned that since it does not support softmax layers, the latter was replaced by a fully connected layer for the tests.

Table 1 shows the values of data memory, program memory, inference time and accuracy of the test performed for each version of the library in each microcontroller.

MCU	Library	Variant	Program Mem. (Kib)	Data Mem. (Kib)	Inference Time (μs)	Accuracy (%)
ATMega 2560 Arduino Mega	Embedia NN	Floating Pt.	14.04	5.75	75498	98.89
		Fixed Pt. 32 bits	15.49	5.75	87408	98.89
		Fixed Pt. 16 bits	**11.38**	**3.09**	**37757**	**98.89**
		Fixed Pt. 8bits	9.47	1.77	15221	89.72
STM32f103c8t6 Bluepill	µTensor	Floating Pt.	31.26	4.95	5945	98.89
	Embedia NN	Floating Pt.	23.01	0.67	9834	98.89
		Fixed Pt. 32 bits	19.02	0.67	2746	98.89
		Fixed Pt. 16 bits	**15.54**	**0.54**	**2449**	**98.89**
		Fixed Pt. 8 bits	14.32	0.48	2384	89.72
Tensilica L106 NodeMCU	Eloquent TinyML	Floating Pt.	130.17	25.16	11531	98.89
	Tensorflow Lite	Floating Pt.	115.61	23.46	11549	98.89
	Embedia NN	Floating Pt.	17.12	19.63	8213	98.89
		Fixed Pt. 32 bits	15.94	5.88	5012	98.89
		Fixed Pt. 16 bits	**13.18**	**3.39**	**1489**	**98.89**
		Fixed Pt. 8 bits	12.02	2.11	1705	89.72
Xtensa LX6 Esp 32 Devkit	Eloquent TinyML	Floating Pt.	201.49	12.96	1885	98.89
	Tensorflow Lite	Floating Pt.	191.34	8.90	794	98.89
	Embedia NN	Floating Pt.	19.03	0.94	**284**	**98.89**
		Fixed Pt. 32 bits	18.31	0.94	341	98.89
		Fixed Pt. 16 bits	**15.81**	**0.81**	367	**98.89**
		Fixed Pt. 8 bits	14.54	0.75	361	89.72
RP 2040 Raspberry Pico	Eloquent TinyML	Floating Pt.	90.98	23.28	12833	98.89
	Tensorflow Lite	Floating Pt.	129.53	21.99	10862	98.89
	µTensor	Floating Pt.	29.14	12.47	16400	98.89
	Embedia NN	Floating Pt.	9.54	6.07	9468	98.89
		Fixed Pt. 32 bits	6.32	2.38	3241	98.89
		Fixed Pt. 16 bits	**6.05**	**2.25**	**1258**	**98.89**
		Fixed Pt. 16 bits	5.98	2.19	1291	89.72

Table 2: Comparison of memory footprint and inference time required by the libraries in each microcontroller.

ISBN 978 -987-633-574-4

A significant advantage of the different EmbedIA-NN implementations over the other libraries can be seen in the table 2 and the charts in figure 3. While the 8-bit and 16-bit fixed-point implementations stand out, the latter is better because it maintains the same level of accuracy as the other libraries, while the former falls around 10%. Another remarkable aspect is that these two implementations exceed, on average, at least 9 times the memory and inference time requirements of the *Google Tensorflow Lite* and *Eloquent TinyML* libraries.

Another interesting aspect to note is the difference in performance for fixed-point arithmetic implementations on those processors such as *Stm32f103c8t6* and *RP2040* that do not have support for floating-point arithmetic.

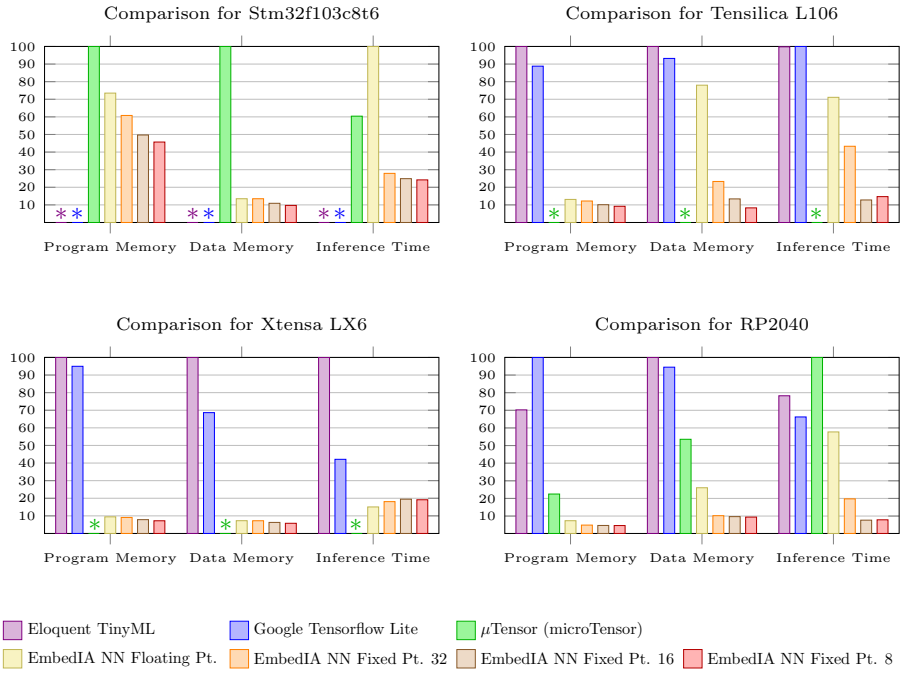


Fig. 3: Comparison of memory usage and time consumption between libraries for different microcontrollers. The unit is expressed as a percentage with respect to the library that had the highest value in the evaluated characteristic.

As the reader may have noticed, the microcontroller *ATMega2560* was not included in the graphs in the figure 3 because it only fit in memory the executables corresponding to the EmbedIA-NN implementations. As an example of the library's potential, a prototype was created for this microcontroller that recognizes handwritten digits on a 240x320 pixel graphics display with a built-in touch screen. This example integrates the experiment model, the inference func-

ISBN 978 -987-633-574-4

tions, the graphics routines code and the touch screen handling code into only 24Kib of program memory and 6Kib of RAM.

Fig. 4: ATMega2560 example integrating model, inference functions, graphics and touchscreen functions in 24Kib of program memory and 6Kib of RAM.

5 Conclusions and Future Work

This paper presented an ultralight and compact library for neural networks, designed to run on small microcontrollers with severe hardware limitations, combined with a *Tensorflow/Keras* model conversion tool and automatic code generation for C/C++ language. It was compared with other alternatives and it was shown that the 16-bit fixed-point implementation achieves at least a 9 times improvement over the memory footprint and inference time of other libraries, while maintaining the same accuracy. The advantage of EmbedIA lies in its combination with the model conversion tool that generates C language projects incorporating only the strictly necessary source code, while other C++ libraries implement class-based software architectures with inheritance and polymorphism that consume a considerable amount of data memory and program memory, and also slow down program execution. EmbedIA is an open source library that is part of a recently emerged project and will be released soon. For the future, it is planned to incorporate in a gradual way: machine learning and neural network algorithms for small microcontrollers; support for taking advantage of microcontroller hardware features with SIMD or DSP instructions; examples with practical, interesting and meaningful models for popular platforms such as Arduino.

In the short term we plan to incorporate development boards with ARM processors to compare with libraries that only support these microcontrollers.

References

1. L. Farhan, R. Kharel, O. Kaiwartya, M. Quiroz-Castellanos, A. Alissa, and M. Abdulsalam, "A concise review on internet of things (iot) -problems, challenges and opportunities," in *2018 11th International Symposium on Communication Systems, Networks Digital Signal Processing (CSNDSP)*, pp. 1–6, July 2018.
2. S. Shekhar and A. Gokhale, "Dynamic resource management across cloud-edge resources for performance-sensitive applications," in *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 707–710, May 2017.
3. "Internet of things (iot) and non-iot active device connections worldwide from 2010 to 2025." https://www.statista.com/. Accessed: 2021-07-26.
4. K. Sharma and R. Nandal, "A literature study on machine learning fusion with iot," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 1440–1445, April 2019.
5. R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *CoRR*, vol. abs/1806.08342, 2018.
6. N. Mitschke, M. Heizmann, K.-H. Noffz, and R. Wittmann, "A fixed-point quantization technique for convolutional neural networks based on weight scaling," in *IEEE International Conference on Image Processing*, pp. 3836–3840, Sep. 2019.
7. R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden, "Tensorflow lite micro: Embedded machine learning on tinyml systems," 2021.
8. "Tensorflow Lite." https://www.tensorflow.org/lite. Accessed: 2021-08-01.
9. "uTensor." https://github.com/uTensor/uTensor. Accessed: 2021-08-01.
10. L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," 2018.
11. Dennis, Don Kurian and Gopinath, Sridhar and Gupta, Chirag and Kumar, Ashish and Kusupati, Aditya and Patil, Shishir G and Simhadri, Harsha Vardhan, "EdgeML: Machine Learning for resource-constrained edge devices."
12. "Eloquent TinyML." https://github.com/eloquentarduino/EloquentTinyML/. Accessed: 2021-07-26.
13. I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. http://www.deeplearningbook.org.
14. E. Alpaydin and C. Kaynak, "Optical Recognition of Handwritten Digits." UCI Machine Learning Repository, 1998.
15. Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.
16. L. Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, pp. 396–404, Morgan Kaufmann, 1990.