MbedML: A Machine Learning Project for Embedded Systems

César A. Estrebou[1][0000−0001−5926−8827], Martín Fleming[2], Marcos 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 article describes the tasks being carried out within the framework of a research and development project on machine learning techniques and algorithms applied to small devices. It includes a brief review of the available technologies, online development platforms, work methodology and open source software for the implementation of solutions. Experiments carried out on a proper implementation of an inference algorithm for convolutional neural networks are also presented with interesting preliminary results regarding existing implementations.

Keywords: Machine learning · Embedded Systems· Internet of Things · Convolutional Neural Networks

1 Introduction

As reported by Cisco [1] and Statista[2], it is estimated that the number of IoT devices connected to the Internet by 2021 will be between 10,600 and 13,800 million. Many of these devices, limited in both hardware resources and processing capacity, upload information to the cloud to be processed, which leads to problems [3, 4] related to bandwidth, response delays, high computational and storage costs, higher energy consumption, among others. To address these problems, the popular concept of *Edge Computing* arises, which refers to the transfer of total or partial computing from the cloud to the devices located at the edge of the network. In this way you can take advantage of the computing power of these low-power devices that can execute millions of instructions per second despite their limitations. In general, machine learning and in particular deep learning have the potential to make important contributions since they can provide robust solutions [5] as long as they can be adapted to the capacity of the edge computing devices.

Both the area related to *edge devices* and machine learning have received a lot of attention from large hardware and software companies. This drive that combines machine learning techniques with different platforms for embedded systems, brings together and facilitates the development of applications that run on small microcontrollers, something that until recently seemed unthinkable.

With all of the above as motivation comes this research and development project aimed at documenting, studying and implementing traditional machine learning techniques adapted to small devices.

2 Software and Hardware Tools

The main objectives of this project are to document, analyze, test and develop both machine learning and deep learning tools and techniques adapted to devices with limited computing capacity and resources.

Part of the work carried out in the project includes the study of different software tools that develop solutions based on machine learning techniques on microcontrollers. The following sections briefly discuss the work done so far.

2.1 Online Development Platforms, Frameworks and Libraries

There are several online platforms that automate the entire development process, or at least a good part of it. Platforms such as *AlwaysAI*, *Edge Impulse*, *Qeexo*, and *Cartesiam.AI*, with just data loading and microcontroller model selection, automatically scan a wide variety of algorithms with different configurations and perform the deployment of the final application in the microcontroller. The *OctoML* platform optimizes models previously built by the user using machine learning techniques for efficient execution in the microcontrollers it supports. Although in general, most of the platforms support the Arm Cortex-M MCUs, the low number of supported devices is objectionable.

The open source libraries and frameworks for developing machine learning applications on microcontrollers are few. The software with the greatest potential that has been surveyed and analyzed so far is briefly described below:

- MicroML: Implements Scikit-learn (Python) algorithms in C code. Supports Decision Trees, Random Forest, XGBoost, Gaussian NB, SVM, SEFR.
- eMLearn: Supports Decision Trees, Random Forest, XGBoost, Gaussian NB, Keras fully connected neural networks and audio features extraction.
- TensorFlow Lite: Support for neural networks generated with TensorFlow. Requires 32-bit MCU architectures (ARM and ESP32). Provides tools to adapt your models to microcontrollers.
- TinyML: Supports TensorFlow Lite neural network models in MCU ARM.

2.2 Project Microcontrollers

At this time the project has several development boards to perform tests on the different implementations of machine learning algorithms. In the future, several more are planned to be incorporated. The MCUs currently being experimented on are *Arm Cortex-M3*, *Tensilica L106* and *Xtensa LX6* and the technical characteristics can be seen in the table 1. The decision of the embedded models is based on aspects such as local availability, low cost, computing capacity (medium

to low) 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 there are many popular devices without this feature.

Table 1: Relevant technical characteristics of the MCUs used in the project.

Board	MCU	Cores	Clock	Data	Prog.	Connectivity	US$
Stm32f103c8t6	Arm Cortex-M3	1	72MHz	20KiB	64KiB	No	3,50
NodeMCU ESP8266	Tensilica L106	1	80MHz	80KiB	32KiB	Wi-Fi	3,50
Esp32-Wroom	Xtensa LX6	2	160MHz	520KiB	448KiB	Wi-Fi+BT	8,00

2.3 Machine Learning for Microcontrollers

Due to hardware limitations in terms of data and program memory, it is impossible to perform the generation of the machine learning models (training) on the microcontroller (MCU). For this reason, the model is built in a traditional way on a computer and the corresponding verification tests are performed.

Once the model is generated, a transformation tool is used to reduce its size and thus to fit the MCU's limitations. These types of tools export the model data, adapting from the data types to including the necessary code to execute the model. Finally, tests are performed to verify the effectiveness of the adaptation.

3 Experiments and Results in Convolutional Networks

At the time of writing this article, the team is developing tests on convolutional neural network models on the 3 MCUs used in the project. To perform the tests, it was decided to build a convolutional model with the ability to distinguish a digit in an image. As a data source, the UCI repository database [7] was selected, which is a reduced version of the MNIST database [8] widely used to evaluate image classification algorithms in different areas such as computer vision, machine learning and neural networks. This dataset comprises some 5,620 grayscale images with handwritten digits centered in an 8x8 pixel area.

A convolutional neural network [6] (CNN or ConvNet) was used as a model. This type of network has in its architecture a series of convolutional layers with a nonlinear activation function in its output such as ReLU or tanh. Each layer of the network transforms the representation by applying filters that give a higher level of abstraction. For example, the first layer detects edges, then the second layer detects contours from those edges, then the third layer detects structures from contours and so on until an object is detected.

The objective of the 2 experiments performed was to measure the performance of a convolutional model on different MCUs through the implementations of 2 libraries. For this, a convolutional network was trained to classify 8x8 digit images from the UCI database. In the first experiment the Eloquent TinyML library was used in 2 of the MCUs, leaving out of the test the ARM Cortex-M3 MCU because it was no longer supported. In the second experiment we used a

proprietary implementation of the convolutional neural network inference algorithm that works on the different architectures of the 3 MCUs. Table 2 shows the test results of each experiment. In this it can be seen that the proposed algorithm occupies much less program memory and data than the Eloquent TinyML implementation. Even the significant difference in average inference runtime can also be observed.

Table 2: Experiment results

Exper.	Library	Data Mem.	Prog. Mem.	MCU	Time	Accuracy
1	Eloquent TinyML	Xtensa LX6	23.12 KiB	470.6 KiB	2270 us	97,8 %
2	Own	Xtensa LX6	14.21 KiB	277.1 KiB	606 us	97,8 %
1	Eloquent TinyML	Tensilica L106	51.20 KiB	391.5 KiB	11167 us	97,8 %
2	Own	Tensilica L106	31.01 KiB	274.0 KiB	7568 us	97,8 %
2	Own	Stm32103	2.14 KiB	27.3 KiB	8624 us	97,8 %

4 Final Comments

This article has presented a research and development project of machine learning applied to microcontrollers with low computational power and limited data and program memory. A brief review of both the technologies and the available software has been included and a proper implementation of convolutional neural networks has been presented with satisfactory preliminary results. In the future, we will continue to experiment with algorithms and machine learning techniques in small devices, both our own and those of third parties.

References

1. Cisco, Cisco Annual Internet Report (2018–2023) White Paper https://www.cisco.com/ Last accessed 30 March 2021
2. Statista, Internet of Things (IoT) and non-IoT active device connections worldwide from 2010 to 2025 https://www.statista.com/ Last accessed 30 March 2021
3. Farhan, L., Kharel, R., Kaiwartya, O., Quiroz-Castellanos M., Alissa, A. and Abdulsalam M., A Concise Review on Internet of Things (IoT) - Problems, Challenges and Opportunities, 11th International Symposium on Communication Systems, Networks & Digital Signal Processing, pp. 1-6. Budapest, Hungary, 2018
4. Shekhar, S. and Gokhale A., Dynamic Resource Management Across Cloud-Edge Resources for Performance-Sensitive Applications, 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 707-710, NJ, USA, 2017
5. Sharma, K., and Nandal, R., A Literature Study On Machine Learning Fusion With IOT", 3rd International Conference on Trends in Electronics and Informatics, 2019
6. Goodfellow, I., Bengio, Y., Courville, A. Deep Learning. 1st edn. MIT Press, 2016
7. Blake, C.L. and Merz, C.J., Optical Recognition of Handwritten Digits Dataset, 1998. https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits Last accessed 30 March 2021
8. LeCun, Y. and Cortes, C., MNIST handwritten digit database, 2010 http://yann.lecun.com/exdb/mnist/. Last accessed 30 March 2021