embeddedworld2020
Exhibition&Conference
... it's a smarter world

# Artificial Intelligence on Microcontrollers

1st Raphael Zingg, MSc
*Institute of Embedded Systems*
*Zurich University of Applied Sciences*
Winterthur, Switzerland
raphael.zingg@zhaw.ch

2nd Matthias Rosenthal, PhD
*Institute of Embedded Systems*
*Zurich University of Applied Sciences*
Winterthur, Switzerland
matthias.rosenthal@zhaw.ch

***Abstract*—Using artificial intelligence algorithms such as neural networks on microcontrollers offers several possibilities but reveals challenges: limited memory, low computing power and no operating system. In addition, an efficient workflow to port neural networks algorithms to microcontrollers is required. Currently, several frameworks that can be used to port neural networks to microcontrollers are available. This paper evaluates and compares four of them: "TensorFlow Lite for Microcontrollers" from Google, "Neural Network on Microcontroller" from Jianjia Ma with the CMSIS-NN backend from ARM, X-CUBE-AI from STMicroelectronics and the e-Ai Solution from Renesas. The frameworks differ considerably in terms of workflow, features and performance. Depending on the application, one framework is better suited than another. Neural networks that are ported to microcontrollers with those frameworks are static. This means that once they are integrated into the firmware they can no longer be adapted. In the context of this work, possibilities of unsupervised domain adaptation learning on microcontrollers was investigated and is discussed.**

***Index Terms*—Machine Learning, Neural Networks, Artificial Intelligence, Embedded Systems, Microcontrollers, Tensorflow Lite for Microcontrollers, CMSIS-NN, CUBEMX-AI, NNoM, Benchmark, Embedded Unsupervised Domain Adaptation**

## I. Introduction

Neural networks on microcontrollers have a huge potential for new applications. Because as [1] states: "There are billions of microcontrollers manufactured each year and often those microcontrollers operate without relying on network connectivity. Running neural networks on microcontrollers reduces the required bandwidth, preserves privacy and allows the use of new algorithms on microcontrollers".
Running neural networks on microcontrollers also reveals some challenges: limited memory, low computing power and no operating system. However, there exits a wide range of frameworks which simplify the usage of neural networks on microcontrollers. Such frameworks are: "TensorFlow Lite for Microcontrollers (tfLite)" from Google [1], "Neural Network on Microcontroller" (NNoM) from Jianjia Ma [3] with the CMSIS-NN backend from ARM [4], X-CUBE-AI from STMicroelectronics [2] and the e-Ai Solution from Renesas [5]. Those frameworks have the same workflow: A pre-trained neural network, which is defined in a high level machine learning API such as keras [6], is optimized and converted into source code. This source code can be integrated into a firmware and deployed to a microcontroller. Section II gives an overview of those frameworks and section III shows the results of the evaluation. Currently, on the market available microcontroller based neural network systems, need a complete firmware update to make changes to the neural network. However, many microcontrollers are deployed in environments which are not completely known or which change over time. Generalized neural networks can help to overcome this problem but are often not flexible enough. An ideal solution would be, if the neural network, which is running on a microcontroller, could adapt itself to such a changing environment. Section V presents a new developed solution for adaptive neural networks on microcontrollers.

## II. Framework Overview

Four frameworks to port neural networks to microcontrollers are shortly presented. For every framework an application based on the mnist [7] dataset was programmed. The source code and a how-to guide is available on github [8].

### A. TensorFlow Lite for Microcontrollers (tfLite)

"TensorFlow Lite for Microcontrollers is an experimental port of TensorFlow Lite designed to run machine learning models on microcontrollers and other devices with only kilobytes of memory. It doesn't require operating system support, any standard C or C++ libraries, or dynamic memory allocation. The core runtime fits in 16 KB on an Arm Cortex M3, and with enough operators to run a speech keyword detection model, takes up a total of 22 KB " [1]. TensorFlow Lite for Microcontrollers translates a tensorflow lite model into source code and is open source.

### B. X-CUBE-AI (cube)

X-CUBE-AI is an extension for the STM32CUBEMX application. It extends the application with "capabilities with automatic conversion of pre-trained neural network and integration of generated optimized library into the user's project" [2]. It supports supports various deep learning frameworks such as Keras, TensorFlow Lite, Caffe, ConvNetJs, and Lasagne. It supports 8-bit quantization and is free [2] but not completely open source.

### C. e-AI Solution (e-Ai)

The e-AI Solution is a part of the Renesas web-compiler/$e^2$-studio: "With its flexible and scalable embedded artificial intelligence (e-AI) concept, Renesas offers a future-proof, real-time, low power AI processing solution that is unique in the industry and addresses the specific needs for artificial intelligence in embedded devices at the endpoint." [5]. The e-Ai solution translates tensorflow models into source code. It is not completely open source.

### D. Neural Network on Microcontroller (NNoM) with ARM CMSIS-NN backend

"NNoM is a higher-level layer-based neural network library specifically for microcontrollers" [3]. The aims of NNoM is to provide a light-weight, user-friendly and flexible interface for fast deploying. NNoM will manage the structure, memory and everything else for the developer. It has the option to use the optimized ARM-CMSIS-NN functions as backend for calculations. The NNoM framework translates keras models into source code [3]. It is open source and available on github [3].

## III. Evaluation of Frameworks

In order to evaluate the frameworks an application based on the mnist dataset for every framework was programmed. Two different neural networks were used. A simple neural network with of just one dense layer (*DENSE*) and a more complex neural network with convolutional layers (*CNN*). The source code of the applications and guides are available on github [8]. As hardware the InES CT Board [9] which is based on a STM32F429 [10] microcontroller was used. For the evaluation the memory requirement, runtime and accuracy of the inference was analysed.

### A. Memory Footprint

The memory of the applications is analysed with the *arm-none-eabi-size* utility [11]. An example output of this utility is given in listing 1.

```
text    data    bss    dec    hex    filename
92168   11556   1724   105448 19be8  TFLIT.elf
```

Listing 1
Output of arm-none-eabi-size

"Typically the flash consumption of a application will be *text* + *data* and the RAM consumption of the application will be *data* + *bss* [12]". Table I summarizes the memory requirements of every application:

| | tfLite | cube | NNoM | e-Ai |
|---|---|---|---|---|
| *DENSE* RAM | 13'280 | 5'264 | 2'704 | 33'176 |
| *DENSE* Flash | 103'724 | 52'456 | 26'836 | 42'316 |
| *CNN* RAM | 69'336 | 38'032 | 2'704 | × |
| *CNN* Flash | 159'780 | 85'432 | 91'504 | × |

TABLE I
Memory footprint of firmware of different applications (bytes)

Table I shows the differences in how the investigated frameworks manage memory. Moreover, it is visible that the optimization of the tfLite, cube and NNoM framework yield in significant less RAM usage than the not optimized e-Ai code. The neural network *CNN* was not ported to the microcontroller with the e-Ai framework and could therefore not be measured.

### B. Inference Runtime

To measure the inference runtime general purpose input output (GPIO) pins of the STM32F429 were used. Only the time of inference runtime was measured, not the data transfer to the microcontroller or data conversions. Listing 2 shows how the inference runtime measurements were done:

```
/* start time measurement */
HAL_GPIO_WritePin(GPIOB, PIN_0, SET);

/* run neural network */
pred = MX_X_CUBE_AI_Process(inputPicture);

/* stop time measurement */
HAL_GPIO_WritePin(GPIOB, PIN_0, RESET);
```

Listing 2
Code snipped from runtime measurement

Table II shows the results of the runtime measurements:

| | tfLite | cube | NNoM | e-AI |
|---|---|---|---|---|
| *DENSE* runtime | 0.99 | 5.01 | 0.20 | 2.054 |
| *CNN* runtime | 113.08 | 350.92 | 20.03 | × |

TABLE II
Runtime of minimal neural on the microcontroller (ms)

There are significant differences of the measured runtimes. One prediction of a single mnist image with the neural network generated by the NNoM framework takes only $0.20ms$. The reason is that NNoM uses the optimized-CMSIS-NN kernels as backend [3].

### C. Inference Accuracy

The original accuracy of the model implemented in keras was: 92.57% for the dense model and 98.36% for the cnn model. The accuracy of the converted networks running on the microcontroller was measured. The results are shown in table III:

| | tflite | cube | NNoM | e-AI |
|---|---|---|---|---|
| *DENSE* accuracy | 91.64% | 91.96% | 92.13% | 92.45% |
| *CNN* accuracy | 93.56% | 98.20% | 97.55% | × |

TABLE III
Accuracy (%) of converted neural networks on mnist test set

The accuracy of every neural network is still sufficient and does not differ a lot from the original networks.

## IV. Adaptive Neural Networks on Embedded Systems

Adaptive in this paper is defined as: "Able to adapt to a changing environment". Currently, neural networks on microcontrollers are not adaptive because if the neural network is deployed they are hard to change. The data which the neural network is trained on is called source domain data. This data differs from the data which the neural network sees in the real world, called target domain data. Domain adaptation methods adapt a neural network such that it performs also well on target domain data. Often the target domain data comes without any labels. In this case the domain adaption task is unsupervised. A method which can be applied to many domain adaptation tasks is "Asymmetric Tri-Training for Unsupervised Domain Adaptation" (atda) [14]. An overview of the method is given by figure 1 from [14]:

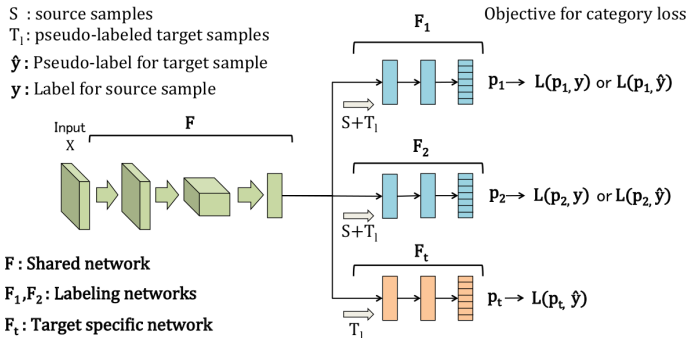


Fig. 1. Atda "includes a shared feature extractor (F), classifiers for labeled samples ($F_1$ , $F_2$), which learn from labeled source samples, and newly labeled target samples. In addition, a target-specific classifier ($F_t$) learns from pseudo-labeled target samples. Our method first trains networks from only labeled source samples, then labels the target samples based on the output of $F_1$ , $F_2$ . We train all architectures using them as if they are correctly labeled samples" [14].

Atda is not well suited for embedded devices, because atda requires a large amount of source domain data for the domain adaptation process and furthermore the used neural network has too many parameters in order to fit on a microcontroller. Therefore, the algorithm was adjusted such that it may run on microcontrollers.

## V. Embedded Unsupervised Domain Adaptation

In order for atda to work on a microcontroller, several adaptions of atda were done:

- Reduced the number of network parameters
- Reduced the size of required source domain data
- Implemented algorithm using only keras operations

Implementing the above results in an algorithm called **emb-atda**.

To compensate for the reduced network parameters and the reduced size of the source domain data set, emb-atda requires a small set of labeled target domain data in order to achieve comparable accuracy. This dataset is called boost dataset and contains 500 target domain images with labels. It is used at the start of the domain adaption process. Emb-atda is therefore not completely unsupervised at the start of the domain adaptation. However, atda also used 1000 labeled target samples to find optimal hyperparameters [14].

A python implementation and guides on how to use it is available on github [8].

### A. Experimental Results

Extensive evaluations of emb-atda on different domain adaptation datasets was performed. The datasets used for the experiments were mnist , mnistm [15] and svhn [16]. Table IV shows the summary of the achieved results.

| **Source Domain:** / **Target Domain:** | mnist / mnistm | mnist / svhn | svhn / mnist |
|---|---|---|---|
| **Method** | | | |
| MMD [17] | 76.9 | - | 71.1 |
| Dann [18] | 81.5 | 35.7 | 71.1 |
| Atda [14] | **94.2** | 52.8 | 86.2 |
| Emb-atda | 87.2 | **58.5** | **89.1** |

TABLE IV
Accuracy comparison from several domain adaptation methods

Eemb-atda achieves comparable or better results as the other methods. The best results are achieved if the source domain data is the most complex (svhn) and the target domain data is the simplest one (mnist). The table V summarizes the required parameters of the different neural networks:

| **Source Domain:** / **Target Domain:** | mnist / mnistm | mnist / svhn | svhn / mnist |
|---|---|---|---|
| Atda | 996'110 | 19'650'526 | 19'650'526 |
| Emb-atda | 368'030 | 9'851'486 | 1'573'982 |

TABLE V
Required neural network parameters comparison

As table V shows, emb-atda works with significant less parameters that atda does. This is very beneficial for the use of emb-atda on a microcontrollers, because those devices have often very little memory.

## VI. Conclusion

In this paper four frameworks for embedded neural networks are evaluated. The evaluation has shown that the compared frameworks differ in terms of memory requirement, inference runtime and inference accuracy. For every framework a guide as well as an example software is provided. The neural networks generated with those frameworks are static, however, many environments which deploy embedded devices need to be flexible. It would be beneficial if the deployed neural networks could adapt itself to a changing environment. Unsupervised methods can solve this problem. One method, atda, was adapted such that the method can be ported to a microcontroller. The prototype of this method is called emb-atda and available on github. Emb-atda achieves comparable results as other domain adaptation approaches but uses less network parameters and does not require huge datasets.

## REFERENCES


[1] Tensorflow, "TensorFlow lite for Microcontrollers" [online] https://www.tensorflow.org/lite/microcontrollers, accessed 18-Nov-2019

[2] STMicroelectronics, "STM32Cube initialization code generator", [online] https://www.st.com/en/development-tools/stm32cubemx.html, accessed 18-Nov-2019

[3] J. Ma, "A higher-level Neural Network library for microcontrollers", [online] https://github.com/majianjia/NNoM, accessed 18-Nov-2019

[4] Liangzhen Lai and Naveen Suda and Vikas Chandra, Efficient Neural Network Kernels for Arm Cortex-M CPUs, CoRR, Volume abs/1801.06601, 2018

[5] Renesas, "e-AI Solution", [online] https://www.renesas.com/jp/en/solutions/key-technology/e-ai.html, accessed 18-Nov-2019

[6] Keras, "Keras: The Python Deep Learning library", [online] https://keras.io/, accessed 18-Nov-2019

[7] Y. LeCun, C. Cortes and C. J.C. Burge, "THE MNIST DATABASE of handwritten digit", [online] http://yann.lecun.com/exdb/mnist/, accessed 19-Nov-2019

[8] Raphael Zingg, Matthias Rosenthal "Artificial Intelligence on Microcontrollers", [online] https://github.com/InES-HPMM/ai_on_mcu, accessed 14-Jan-2020

[9] InES, "InES CT Board", [online] https://ennis.zhaw.ch/wiki/doku.php, accessed 19-Nov-2019

[10] STMicroelectronics, "Discovery kit with STM32F429ZI MCU", [online] https://www.st.com/en/evaluation-tools/32f429idiscovery.html, accessed 07-Jan-2020

[11] Debian Manpages "GNU Development Tools arm-none-eabi-size(1)", [online] https://manpages.debian.org/jessie/binutils-arm-none-eabi/arm-none-eabi-size.1.en.html, accessed 08-Jan-2020

[12] François Baldassari , "Code Size Optimization: Measure Twice, Cut Once", [online] https://interrupt.memfault.com/blog/best-firmware-size-tools, accessed 08-Jan-2020

[13] Hal Daumé, "natural language processing blog - Domain adaptation vs. transfer learning" [online] https://nlpers.blogspot.com/2007/11/domain-adaptation-vs-transfer-learning.html, accessed 08-Jan-202

[14] S. Kuniaki, U. Yoshitaka and Harada Tatsuya, "Asymmetric tri-training for unsupervised domain adaptation", Proceedings of the 34th International Conference on Machine Learning, Volume 70, pp 2988-2997, 2017

[15] Clayton Mellina, "Domain-Adversarial Training of Neural Networks in Tensorflow" [online] https://github.com/pumpikano/tf-dann, accessed 19-Nov-2019

[16] Yuval Netzer and Tao Wang and Adam Coates and Alessandro Bissacco and Bo Wu and Andrew Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning", [online] http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf, 2011

[17] M. Long and J. Wang, "Learning Transferable Features with Deep Adaptation Networks", CoRR, Volume abs/1502.02791, 2019

[18] G. Yaroslav and L. Victor "Unsupervised Domain Adaptation by Backpropagation", Proceedings of the 32nd International Conference on Machine Learning, Volume 37, pp. 1180-1189, 2015