On-device Training on the MAX78000 Ultra-low-power CNN Accelerator

Francesco Cenciarelli
Department of Chemical Engineering
University of Cambridge
fc545@cam.ac.uk

Abstract

On-device training is a new paradigm of edge machine learning aiming to create adaptable and portable AI systems for microcontroller units (MCUs) with limited computational and memory resources. The main focus of this research is the creation and implementation of an on-device training system for the MAX78000 SoC, an ultra-low-power convolutional neural network (CNN) accelerator. This involves an in-depth exploration of the MCU's hardware architecture and the development of a strategy for decoding and updating weights in memory. The system is designed for fine-tuning of the last fully connected layer of a CNN, and it has been tested by fine-tuning with EMNIST Letters dataset a model pre-trained on the MNIST dataset. This research shows the feasibility of on-device training on the MAX78000 MCU, yet highlights the challenges and limitations in training larger models or more complex tasks that necessitate larger datasets, or deeper neural network updates.

1 Introduction

In recent decades, the emergence of deep learning has transformed the landscape of artificial intelligence, in particular through the evolution of convolutional neural networks (CNNs). These architectures revolutionized our way of living with a myriad of new applications, ranging from speech recognition to computer vision [Liu et al., 2022]. As the accuracy of these models increased, so did their complexity, with the model necessitating more computational resources both in terms of processing speed and memory.

Concurrently, there has been significant advancement in AI-enabled portable devices, distilling decades of advances into compact, hand-held devices. This leap has been only possible thanks to the introduction of innovative hardware architectures designed to manage the intensive workloads of modern model architectures. Only in recent years, a notable trend started with the advent of AI accelerators characterized by small form factors in the millimeters scale and low energy consumption. These have led the way to a new field of research focused on the deployment of AI at the edge. Within this domain, there are low-power devices including Google Coral Edge TPU and NVIDIA Jeston Nano, as well as ultra-low-power devices, operating on just hundreds of microjules (uJ) per inference [Moss et al., 2023]. Among these new MCU accelerators, the Maxim Integrated MAX78000 stands out due to its superior performance from other currently available devices such as Greenwave GAP-8 and this research focuses on exploring its potential and expanding its capabilities [Moosmann et al., 2023].

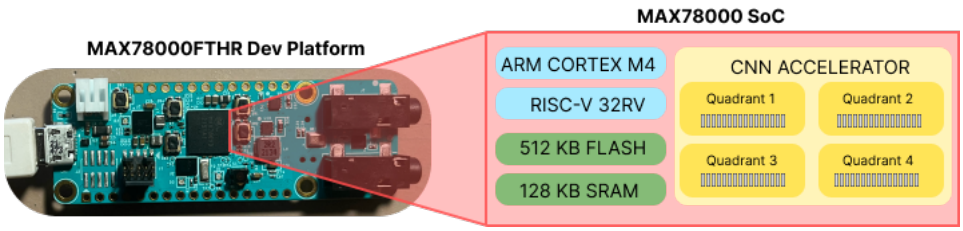

Figure 1: The MAX78000FTHR development platform and the main components integrated integrated in the MAX78000 SoC

While the MAX78000 System-on-Chip (SoC) allows for the operation of complex deep neural networks (DNNs) on battery-powered devices, it currently does not support on-device training or fine-tuning of model weights after flashing the model. The start process of re-training the model would involve gathering data from the device, training and converting the model on a separate computer, and then flashing the binary file on the device. This limitation poses significant challenges in real-world scenarios where adaptability to changes in the use cases and on-site learning are fundamental.

This research, therefore, aims to develop a framework for on-device learning via model fine-tuning on the ultra-low-power MAX78000 MCU. This process will first involve training, quantizing, translating, and loading a model into the CNN accelerator. Then the focus will shift towards managing CNN execution through low-level commands, weights unloading and decoding, and finally carrying out backpropagation on the last fully connected layer of the chosen CNN architecture. The framework will be tested by fine-tuning the model to recognize an additional class outside of the training set, assessing the classification accuracy and the achievable training performance in terms of time consumed on MAX78000.

While the showcased training process was limited by the memory size of the MAX78000, this research serves as a proof-of-concept study demonstrating the feasibility of adaptable machine learning development on ultra-low-power devices. The following sections will first delve deeper into the MAX78000 and then outline the proposed on-device training strategy.

2 Background

2.1 Introduction to MAX78000 and CNN acceleration

The MAX78000 is a state-of-the-art dual-core AI MCU able to carry out ultra-low-power inference on CNNs. It is based on an SoC with an Arm Cortex-M4 core and an RISC-V RV32 core, operating through an AHB bus matrix interfacing with a CNN accelerator engine. This hardware-based accelerator features dedicated memory, with 442KB SRAM for weights and 512KB for data. The architecture can handle computationally heavy workloads by parallelized operations across 64 processors, designed in a quadrant layout, each containing 16 processors. Each of these quadrants can be controlled and configured by a programmable register. A key feature of the SoC and the reason why it was chosen is the ability to run efficiently DNNs and CNNs by synchronizing multiple layers, to both fuse operations, strategically unloading data to prevent overlap across layers and utilizing signed 8-bit integers, ranging from -128 to 127 for weight representation.

For practical implementation, this project employs the MAX78000FTHR development board, shown in Figure 1. This board offers the MAX78000 SoC in its CTBGA format while also allowing a range of peripherals and simplifying the programming process through a USB-connected DAPlink on a surface-mounted MAX32625 microcontroller.

2.2 Model training, synthesis and deployment for custom CNN accelerator

Developing models for the MAX78000 SoC involves utilizing the Maxim Integrated Embedded Software Development Kit (MSDK). This toolkit guides developers through the process of model

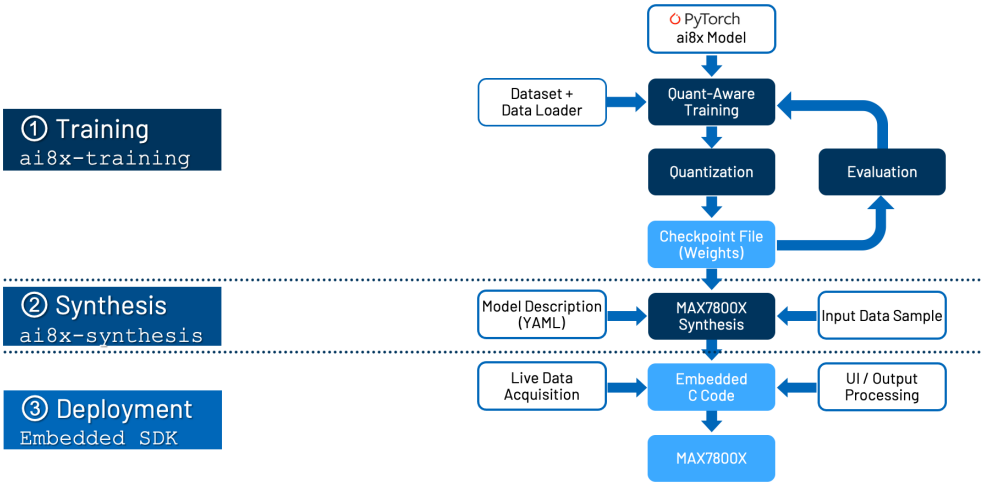

Figure 2: Training, Synthesis and Deployment pipeline for the MAX78000 (Reproduced from MaximIntegratedAI)

training, synthesis (converting into a format readable by the CNN accelerator), and deployment on the MCU. The development begins by designing a model in PyTorch using the custom *ai8x-training* library [MaximIntegratedAI]. This allows to use of neural network modules with fused operators, enabling efficient in-flight pooling and activation within the CNN accelerator. The library also supports quantization-aware training, accommodating a range of bit weights from 1 to 8 bits.

Post-training, the model must undergo a process of quantization and then synthesis into C code to be flashed into the MCU. This can be done by using the Maxim integrated *ai8x-synthesis* library. The conversion of the trained model is dictated by a detailed Network (.yaml) file, specifying the allocation of each layer operation to processors. The model is encoded into a header file (*weight.h*), where the weights are stored in *uint32_t* data types and ordered by processor number. The C code can be uploaded and debugged using ARM GCC and OpenOCD provided on the Maxim Integrated toolchain. This whole process used to train and load a CNN model into MAX78000 is illustrated in Figure 2.

2.3 On-device training and Model Fine-tuning overview and challenges

Edge machine learning research has traditionally focused on developing more efficient model architectures, maximizing performance while minimizing resource consumption. This often involves techniques like neural architecture search or model compression via pruning, tailoring a neural network to fit into the constraint memory of edge devices Cai et al. [2019]. However, recent advances are shifting the paradigm towards real-time on-device algorithm training. Studies such as TinyOL demonstrated the feasibility of using online updates to modify the neural network parameters Ren et al. [2021]. Building on this paradigm, subsequent research explored new methodologies, combining sparse updates with quantization-aware scaling Lin et al. [2022]. These allow for more comprehensive model modifications, mitigating the issue of catastrophic forgetting often seen when updating only the last layers of the neural networks. On the other hand, other innovative strategies involve leveraging Few Shot Learning techniques, coupled with sparse updates, to enhance model accuracy while addressing the challenges of data scarcity in practical IoT applications Snell et al. [2017]Kwon et al. [2023].

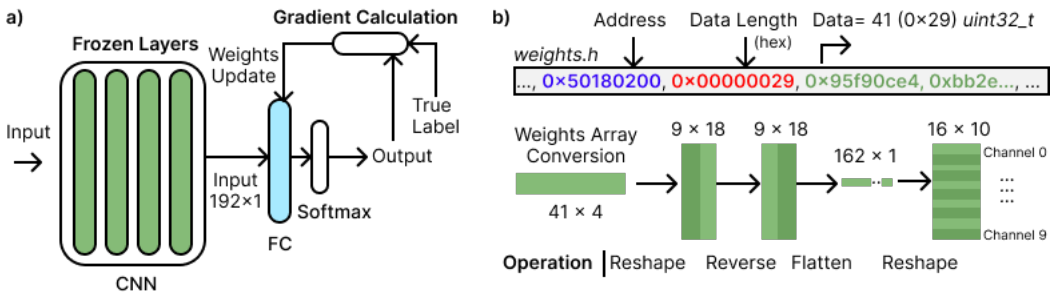

Figure 3: (a) On-device training diagram showing the division between trainable layer and frozen layers, (b) Model weights decoding operations and parsing mechanism inside the MAX78000 memory.

Despite these breakthroughs, efficiently training even the last fully connected layer of a CNN on specialized ultra-low-power devices remains a challenge. While previous research has tried to interface with the CNN accelerator, this research aims to practically demonstrate the feasibility of model training on the MAX78000 SoC. The core objective is to develop a weight unload layer, showcasing how the intermediary outputs of the CNN accelerator can be used for a complete backward pass on the final layer of a CNN. This research sets the stage for applying new techniques using sparse updates on the ultra-low-power device MAX78000.

3 Implementation

3.1 On-device training strategy and diagram

The creation of an on-device training system involved a two-phase process: initial pre-training and uploading of the model on the MAX78000 MCU, followed by the creation of a training algorithm inside the device. The first phase involved training a CNN on an external device as explain in the previous section. The second phase instead involved deploying and testing the on-device training setup shown in Figure 3a. The setup is divided into two main components: a static frozen convolutions layers segment and a dynamically trainable segment. The frozen part, is trained on the separate device using and is responsible for the primary feature extraction. Its outputs were then offloaded, combined with the weights from the fully connected layer, and processed through a softmax activation function. The resulting output was used for the backward pass, including gradient calculation, to update the weight and biases of the fully connected layer. By introducing new data from an alternate dataset and corresponding true labels, the system could adapt the last fully connected layer to recognise new classes absent from the original dataset.

3.2 CNN weights decoding and memory management

A critical aspect of the implementation for the on-device learning system was the decoding of the CNN weights. This process involved converting the weights array from the automatically generated *weights.c* file. The weights, organized by processors, were decoded through a pipeline involving multiple steps of reshaping as shown in Figure 3b. In particular, the fully connected layer of interest had a total of 1920 weights that were spread in 12 different processors. Alongside weights decoding, functions to update and set the weights were implemented in order to carry out on-device learning.

3.3 Dataset choice and preparation

In order to develop and test the system, the MNIST dataset was chosen for its simplicity and compatibility with the MAX78000's memory constraints and was used for pre-training the CNN model for a 10-class image classification task. The choice was made over the more complex CIFAR-10 dataset, that did not allow for a full backward pass due to the MAX78000 limited memory. For

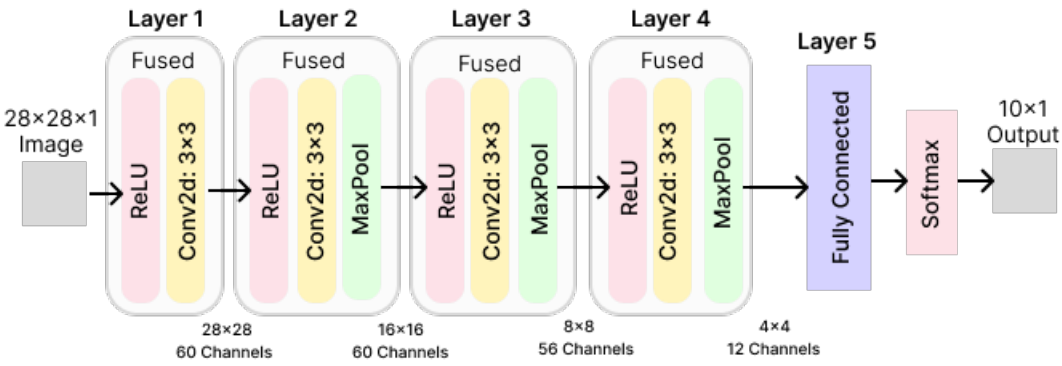

Figure 4: CNN Representation showing the output sizes of each of the five layers

the fine-tuning process the EMNIST Letters dataset was used and in particular the class representing "0" on the MNIST dataset was retrained to recognise the letter "p". In preparation, MNIST images (28x28 pixels) were directly utilized, while EMNIST Letters images, used for on-device training, were converted to C code, with values clamped between -128 and 127 and formatted as an array of *uint32_t*.

3.4 Pre-training the convolutional neural network

Pre-training the neural network is a fundamental step to establish a robust base model capable of extracting features before on-device fine-tuning. The model architecture used in this study is composed of 4 convolutional layers followed by a fully connected layer and is displayed in Figure 4. Each covolution layer is composed of fused operations, where the 2D convolutions are fused with ReLU activations and MaxPooling from layer 2 to 4. The model was trained using the *ai8x-training* repository and a Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.1 for 200 epochs. After training the model was quantized to have 8-bits integers weights to be loaded on the MAX78000 board.

3.5 Fine-tuning loop inside the device

The on-device fine-tuning process focused on the last layer of the network, Layer 5 in Figure 4. It began by instructing the CNN accelerator to split the operation of the frozen part of the model, comprising the convolutions layer, and the fully connected layer. After this mechanism was tested, the focused shifted towards decoding the weights and carrying out backpropagation updates.

Given the memory limitation of the MAX78000, the maximum dataset size that could be loaded on the device flash memory was found to be 30 images. This constraint posed quite a great challenge and highlighted the complexity of dealing with ultra-low-power devices. However, to effectively deploy fine-tuning, a strategy where updates are carried out after every was carried out. This approach was crucial to have a dynamic learning process on the new class. The updates utilized cross-entropy loss (L) for optimization, where y is the true label and $\hat{y}$ is the predicted output.

$$L = -\sum_{i} y_i \log(\hat{y}_i) \tag{1}$$

The weights (W) updates were done calculating the gradient of the loss function with respect to the weights $\nabla_W L$ and multiplying it by a learning rate, which was chosen to be $\eta = 0.5$.

$$W = W - \eta \cdot \nabla_W L \tag{2}$$

The iterative process continued for multiple epochs and the following algorithm showcases this fune-tuning process, hihglighting the steps of forward and backward propagation.

Algorithm 1 On-device Last Layer Fine-tuning on MAX78000

```
for epochs = 1 to N do
    for input = 1 to M do
        Initialise CNN Accelerator
        Load Weights and Data
        Forward Propagate until Conv Layer 4
        Unload Conv Layer 4 output and Weights
        Back Propagate through the FC Layer
        Update Weights and Set them in memory
    end for
end for
```

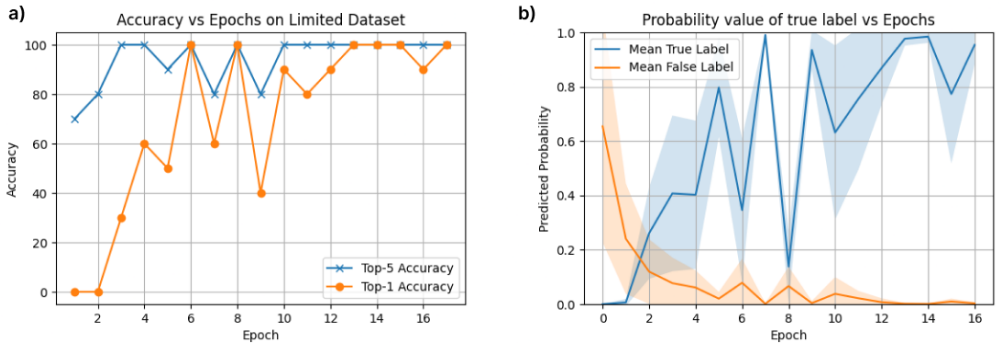

Figure 5: (a) Training accuracy of classification of letter "p" against epoch number with model fine-tuning on-device, (b) Probability value of true label "p" against epoch number.

4 Experiments

4.1 Pre-training of the convolutional neural network

The pre-training process of the convolutional neural network model described in Figure 4 was carried out on a MacBook Pro M3. The model, designed specifically for high accuracy in digit recognition achieved a remarkable 99.5% accuracy on the MNIST Dataset. Post-training, a quantization step was carried out to obtain a C code featuring 8-bit weights that could be then deployed on the MAX78000FTHR.

4.2 On-device training result

The core of the experiments revolved around the concept of on-device training, specifically focused on the ability of MAX78000 to learn and recognize a new class, absent in the initial pre-trained model. The approach involved retraining the MNIST model to identify a class from the EMNIST Letters dataset. For this experiment, the letter "p" was selected for retraining. The memory allowed only a maximum of 30 sample images to be stored in the device. For this reason, 20 "p" images were used for training and 10 for testing the model performance across multiple epochs. The results are illustrated in Figure 5a and demonstrate clearly that as the number of epochs increased, so did the model's accuracy in the identification of the new class "p". Initially, at epoch one, the letter "p" was only recognized within the top-5 predictions. However as the training progressed the top-1 accuracy improved significantly, reaching a perfect 100% by epoch 16. This result shows the success of the learning process despite the constraints of the MAX78000 board. An interesting observation is the instances of catastrophic forgetting during training, evident at epoch 9, where the accuracy suddenly drops to 40%. This phenomenon highlights the challenges involved in on-device learning and suggests that in the future more comprehensive updates of multiple layers should be preferred.

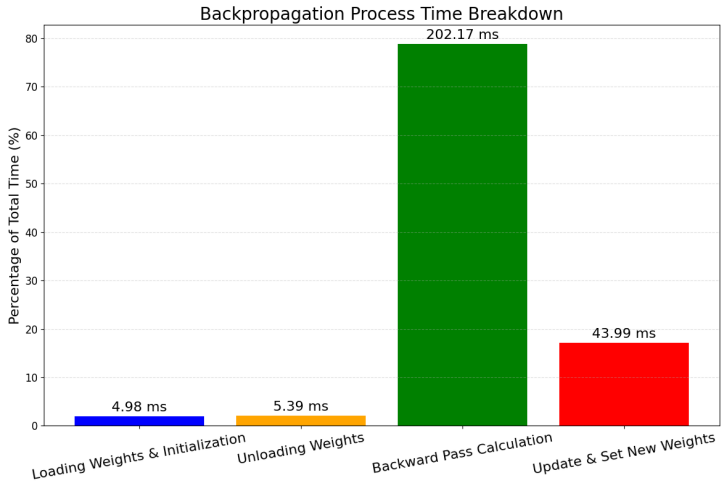

Figure 6: On-device training timing breakdown between various tasks

Another interesting result is shown in Figure 5b, showing the shift in classification probabilities between the true label "p" and a commonly mistaken false label "4" from the MNIST dataset. Initially, the model predicts with a high probability the number "4" but during training the probability of identifying "p" increases to reach values around 0.9 in the last epochs. On the other hand, the probability of mistaking "p" for label "4" decreases dramatically to nearly zero.

4.3 Time considerations during the training process

This research also aimed to assess the on-device training framework performance from a temporal point of view. While the MAX78000 is optimized at quick inference, its efficiency in carrying out a backward pass was investigated.

Timing for each training step was monitored using low-level functions provided by Maxim Integrated. The result of this analysis, comprising timing breakdowns is shown in Fig 6. The plot shows that initial operations, such as weight loading, initialization, and the forward pass were executive in under 5 ms. However, other steps in the proposed framework, such as memory unloading and weight updating, were less time-efficient, being executed in 43.99 ms, occupying almost 20% of the total time. The backward pass, without any hardware acceleration, was the most consuming step by far. It requires over 200 ms to complete, which is equivalent to around 80% of the total update time.

5 Limitations and Conclusion

This research embarked on the goal of demonstrating the feasibility of training a neural network directly on the MAX78000 SoC. The efforts resulted in the successful fine-tuning of a pre-trained MNIST model composed of 4 convolution layers, allowing it to recognize a new class from the EMNIST letters dataset using a limited amount of data. This achievement also shows that the most inference-optimized devices have the potential to implement on-device learning.

However, the research process brought to light a variety of limitations, particularly regarding the approach employed and the constraints of the MAX78000. One of the most pronounced limitations was the training of only the last layer of the network. While this method proved to be sufficient to re-train one class, it restricts the learning process of the network. Further studies involving more comprehensive sparse updates of deeper layers could lead to more substantial improvements. However, these improvements are always going to be limited by hardware memory limitations.

Another important challenge encountered was related to the storage capacity in the MAX78000 flash memory. Only after the implementation of the system, it was discovered that the maximum allowed

device storage is up to 30 images. This limitation still did not hinder the main objective but poses serious doubts about the scalability of this approach using more extensive datasets or more complex models. In fact, in this study, while the model's performance was satisfactory, it is essential to know that training with such a small dataset does not adequately represent real-world scenarios where a wide range of data is needed for successful deployment. To circumvent these memory issues different approaches should be investigated. In particular, one approach is to stream data directly into the device to facilitate extensive training with a larger set of data.

While this study demonstrated the potential for on-device layer training on the MAX78000 SoC, the limitations encountered underscore the need for a new approaching to managing memory and the learning process in this advanced hardware.

References

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *CoRR*, abs/2201.03545, 2022. URL `https://arxiv.org/abs/2201.03545`.

Arthur Moss, Hyunjong Lee, Lei Xun, Chulhong Min, Fahim Kawsar, and Alessandro Montanari. Ultra-low power dnn accelerators for iot: Resource characterization of the max78000. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, SenSys '22, page 934–940, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450398862. doi: 10.1145/3560905.3568300. URL `https://doi.org/10.1145/3560905.3568300`.

Julian Moosmann, Hanna Mueller, Nicky Zimmerman, Georg Rutishauser, Luca Benini, and Michele Magno. Flexible and fully quantized ultra-lightweight tinyissimoyolo for ultra-low-power edge systems, 2023.

MaximIntegratedAI. Maximintegratedai/ai8x-training: Model training for adi's max78000 and max78002 ai devices. URL `https://github.com/MaximIntegratedAI/ai8x-training`.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. August 2019.

Haoyu Ren, Darko Anicic, and Thomas Runkler. TinyOL: TinyML with Online-Learning on microcontrollers. March 2021.

Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256KB memory. June 2022.

Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. March 2017.

Young D Kwon, Rui Li, Stylianos I Venieris, Jagmohan Chauhan, Nicholas D Lane, and Cecilia Mascolo. TinyTrain: Deep neural network training at the extreme edge. July 2023.