Efficient On-device Training and Object Detection on Microcontrollers: A Study of MCUNet and Tiny Training Algorithms

Isabella Pedraza Pineros
Department of Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
ipedraza@mit.edu

Abstract

Recent advancements in machine learning (ML) have paved the way for deploying complex algorithms on resource-constrained devices like microcontrollers (MCUs) [7, 8]. This paper presents two projects: On-device training and MCUNet. The former focuses on enabling on-device model training with minimal memory consumption, while the latter provides a framework for efficient object detection on MCUs. These projects address the critical challenge of executing deep learning tasks on devices with limited computational resources, opening new possibilities for edge AI applications.

1 Introduction

The proliferation of low-cost, low-energy microcontrollers has ushered in a new era of interconnected devices, transforming everyday objects into smart, data-generating tools. In recent years, TinyML has emerged as a fast-paced and upcoming field that capitalizes on the ubiquity and efficiency of microcontrollers to bring deep learning models directly to the edge, enabling on-device data analytics and vastly expanding the scope of AI applications [7]. TinyML techniques seek to not only democratizes access to AI technologies, but also play pivotal roles in areas like healthcare and privacy [7].

Two groundbreaking techniques that have surfaced from TinyML research are on-device training and inference on microcontrollers. On-device training involves adapting pre-trained models to new data collected directly from sensors, empowering models to continually learn and adapt post-deployment, enhancing their predictive accuracy and utility in a myriad of applications, especially where user-specific customization is essential. This is particularly advantageous in terms of privacy, as it negates the need to share potentially sensitive information externally [8]. Inference on microcontrollers involves using frameworks such as TFLM [4], CMSIS-NN [5], MicroTVM [9]. This paper specifically focuses on using the co-design framework MCUNet which adapt models to fit and run on microcontrollers while maintaining accuracy [6, 7].

However, the deployment of on-device training and inference on microcontrollers is not without its challenges. Microcontrollers are typically constrained by limited resources, particularly in terms of memory (SRAM) and storage (Flash), which are crucial for the execution of deep learning models [7]. Moreover, on-device training is further limited by low bit-precision, which affects the optimization of quantized neural network graphs, and constrained hardware resources, which affect the possibility of full back-propagation [8]. The memory required for both the inference and training phases of deep learning models, particularly for the storage of intermediate activations during back-propagation, often exceeds what is available on these devices [8].

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

This paper summarizes the strategies and solutions developed to overcome these hurdles, highlighting the advancements that continue to shape the future of on-device machine learning and open new frontiers for AI applications on the edge. In addition, this paper delves into the process of deploying these strategies on real-life microcontrollers.

2 Related Work

2.1 TinyNAS

TinyNAS [7] is a two-stage neural architecture search (NAS) specifically tailored for deployment on microcontrollers with strict resource constraints. Through it's two-stage architecture, it is able to efficiently manage diverse resource constraints while maintaining a low search cost [7]. The first stage of TinyNAS involves scaling the input resolution and width multiplier of the mobile search space to fit the diverse resource constraints of different microcontrollers [7]. Then, to efficiently search and evaluate this search space, TinyNAS randomly samples networks from the search space, focusing on collecting the CDF of FLOPs [7]. The second stage of TinyNAS specializes the network architecture within the optimized search space by performing a one-shot neural architecture search by training one super network and then conducting an evolution search on that network. With these steps, TinyNAS is able to find the best model given a set of resource constraints while achieving the highest accuracy [7].

2.2 TinyEngine & Tiny Training Engine

TinyEngine and Tiny Training Engine are significant advancements in addressing the challenges of deploying models on the limited resources of microcontrollers. These engines demonstrate remarkable improvements in memory efficiency and computational performance, especially when compared with traditional inference libraries that require extra runtime memory to store a model's meta-information [4, 5, 7, 8, 9].

2.2.1 TinyEngine

TinyEngine uses selective compilation, model-adaptive memory scheduling, computation kernel specialization, and in-place depth-wise convolution to better utilize microcontroller resources [7]. Selective compilation means TinyEngine only compiles the operations used by a given model into the binary in order to reduce memory overheard [7]. Model-adaptive memory scheduling means TinyEngine uses an entire network's topology to schedule memory to align memory allocation with the actual computational needs of a model [7]. Computation kernel specialization means TinyEngine uses a variety of optimization techniques [1] to eliminate branch instruction overheads and enhances computational efficiency [7]. Lastly, in-place depth-wise convolution means TinyEngine performs depth-wise convolutions which allow the input activation of a channel to be overwritten once its computation is complete in order to reduce peak memory usage [7].

2.2.2 Tiny Training Engine

To address the unique challenges of on-device training, Tiny Training Engine (TTE) introduces Quantization-Aware Scaling and Sparse Update. These techniques allow for the efficient computation of gradients and their application to model optimization while skipping less important layers and sub-tensors. The engine leverages compile-time auto-differentiation, generating a static backward graph that can be pruned and optimized, thus significantly reducing memory and computational requirements. TTE's compiler translates these algorithmic innovations into executable binaries, optimized for the target hardware, which minimizes runtime library size and obviates the need for host languages like Python that typically consume substantial memory [8].

3 Methodology

The on-device training demo and MCUNet demo are deployed on an STM32F746 Discovery Kit with 340 kB SRAM and 1 MB Flash connected to an Arducam with a 2MP SPI Camera, SPI speed of 8 MHz, and frame buffer of 8MByte. All code was built and developed on an M1 Mac.

3.1 On-Device Training

On-device training is an emerging approach in machine learning that enables models to be trained directly on the devices they are deployed, such as smartphones and IoT devices. This method enhances data privacy and reduces latency by eliminating the need to send data to remote servers, but it also presents challenges due to the limited computational resources of these devices. Addressing these challenges involves innovative model design and efficient learning algorithms, making AI more responsive and accessible in various real-world applications [8].

3.1.1 Implementation

There are two demos that demonstrate on-device training capabilities. The first demo focuses on the implementation of training on the visual wake words [3] dataset. This demo is implemented by first following the training tutorial in the TinyEngine github [1]. Then, in order to access the UART to actually train on the visual wake words [3] data, a python script was run (provided by TAs).

The second demo was focused on a fun application of on-device training. It focuses on detecting whether an image is a dog or a cat. This was possible by using 18 images of abyssian cats and 18 pictures of bulldogs from the Oxford-IIIT-Pet dataset [10]. This implementation followed closely the aforementioned steps, setting cats to be label 0 and dogs to be label 1. In the demo, we can see the results when we perform inference after the training as "No Person" when we encounter a cat and "Person" when we encounter a dog.

3.1.2 Discussion

After training over the combined 36 images of cats and dogs, by counting how many images were correctly labeled, the model ended up having a 65% accuracy. There are a couple of reasons why this accuracy was perhaps so low. One reason could be the lack of more diverse data – since we only had 36 images in total, and only included one breed of cats and one breed of dogs, we may not have had enough data to train a more robust model. In addition, the pretrained model used was just the one provided in the training tutorial in the TinyEngine github [1]. Using a more robust model could potentially lead to better results.

3.1.3 Future Work

The results obtained from the initial foray into on-device training open some new ways for several key areas of development. Firstly, an improvement would be to train a model capable of distinguishing all 37 unique breeds in the Oxford-IIIT-Pet dataset [10]. This enhancement would not only test the model's classification abilities on a more intricate level but also significantly broaden its application scope. It could even be potentially used by animal shelters to quickly distinguish what kind of breed a new rescue is without having to pay for expensive DNA tests. Another, larger scope improvement could be to optimizing the training efficiency, particularly in reducing the extended training durations, as observed in our preliminary demos, since the first demo's training can take upwards of 7 minutes. This could involve innovative approaches in algorithm refinement or development of more computationally efficient model architectures tailored for the constraints of on-device processing.

3.2 MCUNet

MCUNet is a framework that uses TinyNAS and TinyEngine in tandem to deploy models onto microcontrollers. The synergy between TinyEngine's memory-efficient inference capabilities and TinyNAS's architecture design ensures that even under extreme resource constraints, microcontrollers can host more capable deep learning models with higher accuracy [7].

3.2.1 Implementation

There are two demos that demonstrate MCUNet capabilities. The first demo focuses on the implementation of the person detection model linked in the MCUNet github [2]. This demo is implemented by first following the inference tutorial in the TinyEngine github [1] and then, in order to implement the bounding boxes seen in the tutorial, the det_post_procesing () from the openmv−face−mask−detection folder was copied into main.cpp file of the demo and adapted to the

demo's variables. In addition, examplemodule.c was used as inspiration to create the actual bounding box lines and rectangles that outline people detected by the microcontroller. To test this demo, 10 images from a personal selection were chosen. This set of images were composed of 5 images of people, and 5 images of random items.

The second demo was focused on a fun application of MCUNet. It focuses on detecting whether the image is a person or my cat, Maggie. This implementation followed closely the aforementioned steps, but rather than implementing bounding boxes, it made a relatively small update to the detectResponse function in lcd.cpp. To test this demo, 10 images from a personal selection were chosen. This set of images were composed of 6 images of Maggie, and 4 images of people.

3.2.2 Discussion

The implementation of MCUNet through two distinct demos provides valuable insights into the practical application and scalability of this framework in real-world scenarios. The successful adaptation that occured in the second demo illustrates the flexibility and adaptability of MCUNet in custom applications, highlighting MCUNet's ability to handle detailed image processing tasks on resource-limited devices. Both demos ended up having 100% accuracy.

3.2.3 Future Work

To further examine the effectiveness of MCUNet, these demos could further test different visual wake words [3] models within the mcunet model zoo [2] to examine how these different models adapt to customization. On a larger scope, both demos bring to light areas for future improvement. For instance, the process of adapting existing code for specific functionalities, though successful, indicates a possible need for more streamlined or user-friendly methods of customization within the MCUNet framework. Additionally, the performance in diverse and dynamically changing environments, a common characteristic in real-world applications, remains an area ripe for exploration. The adaptability of MCUNet to various object detection scenarios, beyond the relatively controlled conditions of the demos, would be a critical aspect of future research and development.

4 Conclusion

This paper has highlighted significant advancements in TinyML, focusing on on-device training and MCUNet, and their application in deploying machine learning models on microcontrollers. These approaches demonstrate the potential of performing complex deep learning tasks on resource-limited devices, showcasing improvements in memory efficiency and computational performance.

On-device training, through the use of visual wake words and the Oxford-IIIT-Pet dataset, illustrates the method's capacity for enhancing data privacy and reducing latency. Despite challenges in data diversity and training efficiency, this technique shows promise for broader application in real-world scenarios. Future efforts aimed at recognizing more diverse datasets and optimizing training processes underscore the growing scope and efficiency of on-device machine learning.

MCUNet's implementation in object detection, including person detection and differentiating between humans and pets, highlights its robustness and adaptability in handling detailed image processing tasks. The successful customization in these demos not only showcases MCUNet's flexibility but also points to potential improvements in user-friendly adaptation and performance in dynamic settings.

Overall, the strides made in on-device training and MCUNet represent a significant leap in TinyML. They mark a move towards more accessible and integrated AI technologies, especially in environments with stringent resource constraints. The continued development in these areas is set to expand the reach and impact of AI, bringing advanced computational models to a wider array of devices and applications.

Acknowledgments and Disclosure of Funding

This paper would like to thank the entire course staff MIT's TinyML (6.5490) course and MIT-HAN lab for the STM32F746 Discovery Kit MCU, Arducam camera, jumper cables, and overall support in this project.

References

[1] mit-han-lab/tinyengine: [NeurIPS 2020] MCUNet: Tiny Deep Learning on IoT Devices; [NeurIPS 2021] MCUNetV2: Memory-Efficient Patch-based Inference for Tiny Deep Learning; [NeurIPS 2022] MCUNetV3: On-Device Training Under 256KB Memory.

[2] mit-han-lab/mcunet, December 2023. original-date: 2022-06-14T17:19:15Z.

[3] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. Visual Wake Words Dataset, June 2019. arXiv:1906.05721 [cs, eess].

[4] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Shlomi Regev, Rocky Rhodes, Tiezhen Wang, and Pete Warden. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems, March 2021. arXiv:2010.08678 [cs].

[5] Liangzhen Lai, Naveen Suda, and Vikas Chandra. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs, January 2018. arXiv:1801.06601 [cs].

[6] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. MCUNetV2: Memory-Efficient Patch-based Inference for Tiny Deep Learning, October 2021. arXiv:2110.15352 [cs].

[7] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. MCUNet: Tiny Deep Learning on IoT Devices, November 2020. arXiv:2007.10319 [cs].

[8] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-Device Training Under 256KB Memory, November 2022. arXiv:2206.15472 [cs].

[9] Chen Liu, Matthias Jobst, Liyuan Guo, Xinyue Shi, Johannes Partzsch, and Christian Mayr. Deploying Machine Learning Models to Ahead-of-Time Runtime on Edge Using MicroTVM, April 2023. arXiv:2304.04842 [cs].

[10] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs, 2012.