Reproducing MCUNet: Deploying Tiny Deep Learning Models on IoT Devices

Devin Setiawan, Vinayak Jha

1. Project Overview / Introduction

The paper "MCUNet: Tiny Deep Learning on IoT Devices" addresses the challenge of performing machine learning on tiny Internet of Things (IoT) devices that are based on microcontroller units (MCUs). The primary problem is that microcontrollers have extremely limited memory (SRAM) and storage (Flash), which are 2-3 orders of magnitude smaller than even mobile phones, making it very difficult to deploy standard deep learning models. For example, a state-of-the-art ARM Cortex-M7 MCU has only 320kB of SRAM and 1MB of Flash storage, which is insufficient for models like ResNet-50 or even quantized MobileNetV2. Running deep learning models efficiently on resource-constrained devices like MCUs is of great importance because the number of IoT devices is rapidly increasing. This presents a significant opportunity for tiny machine learning (TinyML), enabling data analytics to be performed directly near the sensor. By processing data locally on these low-cost, low-energy microcontrollers, the scope of AI applications can be dramatically expanded to include areas such as smart manufacturing, personalized healthcare, precision agriculture, and automated retail. ***Our project aims to replicate the results of MCUNet by deploying a neural network model on a microcontroller and evaluating its performance in terms of accuracy, memory usage, and inference speed.*** Additionally, we plan to document and reflect on the implementation process, identifying any challenges or ambiguities we encounter, as well as aspects of the paper and open-source codebase that are particularly well-documented or easy to follow. This commentary will aim to validate the paper's claims through reproduction and serve as a practical guide for others seeking to implement deep learning on MCUs.

2. Objectives

The primary objective of this project is to reproduce the results presented in the MCUNet paper, with a particular focus on the Visual Wake Word (VWW) (i.e. person detection) task. Our goal can be listed as follow:

1. Replicate the MCUNet paper's person detection task using TinyNAS and TinyEngine.
2. Deploy the trained and quantized model on a microcontroller (e.g., STM32 or Raspberry Pi Pico).
3. Benchmark model performance on-device in terms of accuracy, latency, and memory usage.
4. Document the replication process, highlighting challenges, successful strategies, and insights from implementation.

We aim to implement the complete workflow described in the paper from model design using neural architecture search to quantization and final deployment on a resource-constrained microcontroller platform. In the paper they used various hardwares but mainly reported the result using STM32F746. We will perform the experiment on the same, or similar platform such as other variants of STM32 or Raspberry Pi Pico, depending on the availability of hardware.

An additional goal of the project is to gain a working knowledge of the key components underlying MCUNet, including the design principles behind TinyNAS, the quantization methods used to compress models, and the system-level optimizations integrated into TinyEngine. Throughout this process, we will carefully document our workflow, noting both the implementation challenges we encounter and the parts of the original paper or open-source resources that are especially helpful or clearly explained. This will allow us not only to validate the MCUNet approach through replication but also to reflect critically on its reproducibility and applicability, thereby creating a useful reference for others working in the area of TinyML.

3. Methodology

First, we will conduct a deeper review of the MCUNet paper, and their approach. Along with reviewing the paper, we would also look at their implementation on GitHub to get a better understanding of their approach. Since we will also perform code review, we expect to spend about a week on this portion.

Each team member will get a corresponding module of MCUNet (TinyEngine or TinyNAS) to deploy on the hardware. We will first use the TinyNAS framework to recreate the person detection model and apply quantization techniques to reduce the model size for compatibility with microcontroller memory limits. This model will then be deployed using TinyEngine, the lightweight inference engine introduced in the paper.

Next, we plan on evaluating the deployed model using the Visual Wake Words dataset on the STM32F746 embedded platform. The embedded platform is tentative and subject to change due to availability. We will also compare the accuracy, latency, SRAM usage, and Flash usage to those reported in the paper. Based on the time remaining, we would like to attempt to reimplement either TinyEngine or TinyNAS. Note that this will be done after we have successfully deployed the official implementation. The previous step will be repeated, and the mentioned metrics will be recollected.

Finally, time permitting, we will also deploy MobileNetV2 or ProxylessNAS Mobile for the Visual Wake Words task

4. Expected Outcomes

The outcome of the project is the working implementation of the MCUNet architecture on either of the STM32F412, STM32F746, STM32F765 and STM32H743 platforms. The priority would be given to the STM32F746 embedded platform since the paper contains more results under that platform for the chosen dataset (Visual Wake Words). Further, we also aim to achieve similar accuracy, latency constraint satisfiability, and SRAM and Flash usage. For this dataset, the paper also uses the MobileNetV2 and ProxylessNAS Mobile as the baseline. Given that we will attempt to reimplement at least one significant portion of MCUNet (TinyEngine or TinyNAS) ourselves, we might not have sufficient time for testing these baselines. Hence, these would be tested time permitting. Finally, the artifacts would be a short demo showing the model performing image detection on real input and a report summarizing findings and replication effort.

5. Team Members and Roles

The team members are Devin Setiawan and Vinayak Jha. Since the MCUNet is composed of two components (TinyNAS and TinyEngine), which are optimized independently, the responsibility of developing and deploying each component would be split amongst the team members. Other tasks such as data processing, and training would be appropriately split since these tasks can be done independently by either of the team members.

References

1. Lin, Ji, et al. ‘MCUNet: Tiny Deep Learning on IoT Devices’. *arXiv [Cs.CV]*, 2020, http://arxiv.org/abs/2007.10319. arXiv.

2. mit-han-lab. “GitHub - Mit-Han-Lab/Mcunet: [NeurIPS 2020] MCUNet: Tiny Deep Learning on IoT Devices; [NeurIPS 2021] MCUNetV2: Memory-Efficient Patch-Based Inference for Tiny Deep Learning.” GitHub, 2020, github.com/mit-han-lab/mcunet.