External Memory Is All You Need: Tiny Deep Learning on MCUs

Sulaiman Sadiq[†]
ss2n18@soton.ac.uk

Jonathon Hare[†]
jsh2@ecs.soton.ac.uk

Simon Craske[‡]
simon.craske@arm.com

Partha Maji[‡]
partha.maji@arm.com

Geoff Merrett[†]
gvm@ecs.soton.ac.uk

[†] University of Southampton, United Kingdom
[‡] ARM Research, Cambridge, United Kingdom

Abstract

The majority of works in TinyML focus on deploying models within size constraints of fast but limited internal storage and memory. While external alternatives are typically ignored, recent empirical work showed that models deployed with slower external memory combined with overlaying techniques outperform internal memory approaches in accuracy and, somewhat surprisingly, latency. We perform an in-depth analysis to explain these findings and further motivate efficient model design for MCUs. We find that models designed to fit within internal storage constraints lie beyond a *point of diminishing returns* where accuracy improvement is achieved at the cost of significant extra computation, which is focused in inefficient operations leading to high latency. We harness cheap external memories to alleviate internal storage constraints and further propose an efficient overlaying strategy in TinyOpsV2 with up to 10% lower latency. Using our insights, we deploy efficient models achieving 6.7% higher accuracy and 1.4x faster inference latency than internal memory approaches. Additionally, we outperform prior works utilising external memory with 2.9% higher accuracy, setting a new state of the art in TinyML ImageNet classification. Our work suggests using external memories is the way forward for tiny deep learning on MCUs.

1 Introduction

The recent success of Deep Neural Networks (DNNs) [4, 5, 11] combined with the increase in IoT devices [6] has led to the development of the field of Tiny Machine Learning (TinyML) which aims to develop models and frameworks suitable for inference locally on the microcontroller (MCU) based IoT devices. The conventional approach that most works adopt is to squeeze the largest model into fast but limited internal storage (↓2048KB Flash) and memory (↓512KB SRAM). Existing works include quantisation, neural architecture search (NAS) or scaling down width or input resolution of existing mobile models to meet the devices on-chip constraints [8, 7, 2, 10, 1]. External memories on MCUs are typically ignored due to associated energy and latency overheads which can be up to 2x higher. Recent work [10] has however empirically shown that external memory based approaches combined with overlaying techniques are better in performance and energy efficiency which questions whether considering only internal memory for tiny deep learning is the optimal approach.

We perform a systematic analysis of architectures derived from the mobile search space for the internal and external memory design spaces in addition to the operations (Conv, 3x3, 5x5, 7x7 DepConv) in

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

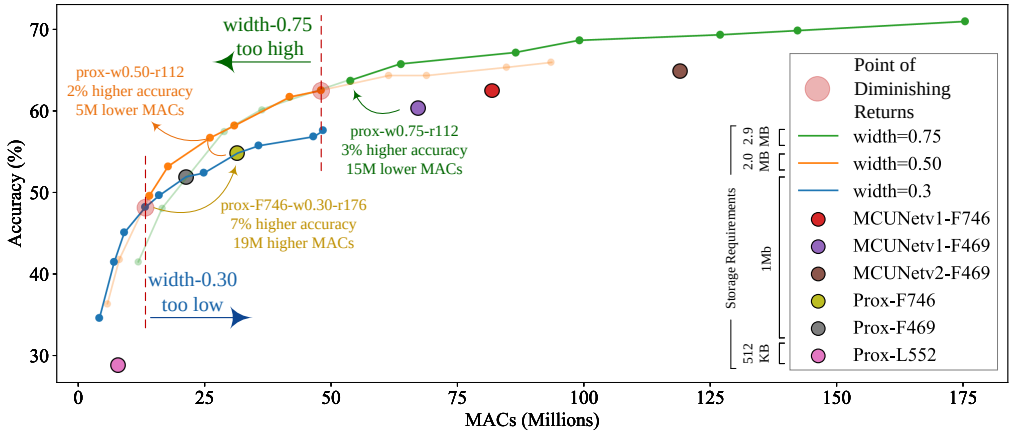

Figure 1: Accuracy vs. MACs curves drawn by fixing width multiplier and increasing input resolution with widths chosen for diverse storage constraints. A ProxylessNAS model scaled down to width 0.50 and resolution 128 is referred to as prox-w0.50-r128. Beyond the *point of diminishing returns*, increasing resolution from prox-w0.30-r112 to prox-w0.30-r176 according to F746 MCU constraints achieves higher accuracy (7%) at the cost of significant extra MACs (19M). In comparison, prox-w0.50-r112 achieves 2% higher accuracy with 5M less MACs. Similarly, NAS based architectures derived under internal storage constraints are sub-optimal compared to higher width architectures.

the architectures to explain these findings and motivate future model design for deep learning on MCUs. We show that trying to extract maximum accuracy under internal storage constraints leads to deployment of degenerate architectures. We alleviate the internal storage constraint by using slower but plentiful external memories and accelerate the inference latency with the 10% faster overlaying strategy of TinyOpsV2. We deploy efficient models from the TinyOpsV2 design space to achieve record ImageNet performance with up to 6.7% higher accuracy and 1.4x faster inference latency.

2 Design Limitations of Internal Storage and Memory

We analysed MCU models (MCUNetV1/2) derived for internal memory and mobile models (e.g. EfficientNet, ProxylessNAS, MNASNet, MbV2/3) which were scaled according to the diverse internal storage (Flash) and memory (SRAM) constraints of MCU devices including L552 (192KB SRAM/512KB Flash), F469 (256KB/1MB), F746 (320KB/1MB) and the H743 (512KB/2MB) by ST microelectronics. The storage and memory requirement dictated by the parameter count and tensor sizes were controlled via channels or width of the model and input resolution respectively. A ProxylessNAS model scaled down to width 0.50 and resolution 128 is referred to as prox-w0.50-r128.

Diminishing Returns: Our analysis revealed that for any particular width multiplier imposed by internal storage constraints, there is an input resolution for which that width multiplier is optimal. We term this as the *point of diminishing returns* as increasing input resolution beyond this point when scaling existing mobile models or specialising architecture (MCUNet) under internal storage constraints increases the computation significantly for a minor accuracy gain. This can be observed in Figure 1 where ProxylessNAS scalings and NAS based MCUNet architectures are outperformed by higher width scalings. However, the constraint of internal storage cannot accomodate the higher widths. On the other hand, when reducing the complexity of the network to meet a certain MAC budget, we observe that for a high width multiplier with lower resolution, the accuracy drops significantly for a minor decrease in computation compared to a better scaling with the same MACs which advocates for a balanced selection of the width and input resolution.

Skewed Scaling of Computation: We found that uniformly scaling down width of models derived from the mobile search space concentrates computation in DepConv operations. This

$$MACs_{conv1} = \alpha^2\beta^2 \times R^2EC_{in}^2 \quad (1)$$

$$MACs_{conv2} = \alpha^2\beta^2 \times R^2EC_{in}C_{out} \quad (2)$$

$$MACs_{depconv} = \boldsymbol{\alpha}\beta^2 \times R^2EC_{in}K^2 \quad (3)$$

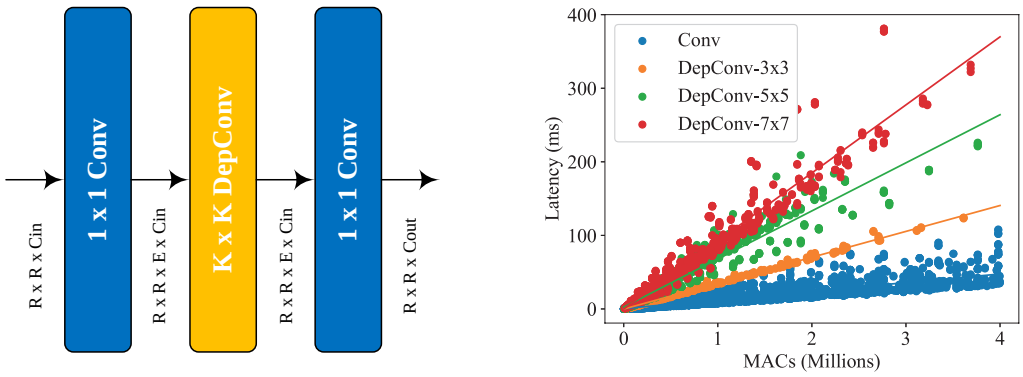

Figure 2: Left: Structure of MobileInvertedConv block. The blocks have 1x1 Conv and DepConv operations with varyiable filter sizes. **Right:** Latency of operations used in MobileInvertedConv blocks with varying complexity (MACs) sampled from scalings of mobile models. We find that 1x1 Convs are more efficient than DepConvs. Within the DepConv operations, 3x3 are the most efficient.

is explained by looking at the structure of the MobileInvertedConv blocks (Figure 2) which are cascaded together to form the models and perform 95% of the total computation in the model. For some width and resolution multipliers, $\alpha, \beta < 1$ respectively, the computation (MACs) performed in operations of the MobileInvertedConv block can be calculated as in Equations 1 - 3 where R, E, C_{in} and C_{out} are the input resolution, expansion ratio and input and output channels respectively. As shown, lowering the width through a uniform multiplier, α, quadratically decreases computation by α^2 in Conv operations, while decreasing only linearly by α in DepConv operations. This has the effect of concentrating a larger percentage of computation in DepConv operations which are less efficient than Conv operations as shown in Figure 2. This leads to lower width scalings having high inference latency compared to higher widths under the same MAC budget. We observed the same behavior in EfficientNet, MNASNet and MbV3 scalings as well as other inference frameworks (STM32CubeAI, TFLite, TinyOpsV2). We note that even though higher width scalings would have the lowest inference latency, they would have low accuracy as discussed in the previous section which leads us to deploy balanced scalings in Section 4.

3 External Memories are All You Need

External Memories and TinyOpsV2 In our work, we propose alleviating the internal storage and memory constraints using external alternatives which are already available on MCUs so provide an efficient way of bypassing the constraints. We improved upon the overlaying approach of [10].which performs inference from slow external memory with internal-memory like latency. Firstly, we modified the buffer allocation strategy so that any unused SRAM is allocated to the buffers used for filter overlaying. This allowed overlaying of more filters in the network from slower external storage to internal memory and reduced latency up to 10%. Secondly, for devices with larger cache, such as the H743 with 16KB cache, we did not observe any benefit of overlaying filters hence this was disabled. In the interest of space we focus only on key differences and encourage the reader to refer to [10] for further information on the TinyOps overlaying strategy.

4 Experiments and Results

Model Deployment: We compare performance of models in the internal and TinyOpsV2 design space on ImageNet classification. We used standard MCUs (F469, F746, H743) used in TinyML, supplemented with 8MB of External Flash and SDRAM. Further details can be found in Appendix A.1. Models were deployed with Tensorflow-Lite Micro (TFLM) [3] and TinyOpsV2 with post-training INT8 quantisation. Due to unavailability of the MCUNet inference framework, MCUNetV1 models were deployed with TinyOpsV2. We used our insights to deploy balanced scalings of the MbV3 architecture which used only efficient 3x3 DepConvs and 1x1 Convs (Figure 2).

Table 1: Models from the TinyOpsV2 design space outperform MCUNet designed for internal memory. Additionally, we use our insights to deploy balanced scalings of the MbV3 architecture to outperform MNASNet architectures. MCUNetV2[9] statistics were taken from authors papers as models weren't available. Models marked as (Repr.) are reproduced by us.

Platform	Model	Design Space	MACs (M)	Params (M)	QAT	Acc (%)	Latency (ms)
F469	MCUNetV1-F469-int8 [8] (Repr.)	Internal	67.3	0.73	N	59.47	3022
	MNASNet-w1.00-r080 [10] (Repr.)	TinyOpsV2	48.2	4.38	N	60.83	2146
	MbV3-w0.75-r128 (Ours)	TinyOpsV2	43.5	2.49	N	**62.58**	**1460**
	MCUNetV2-F469 [7]	Internal	119	<1	Y	64.9	-
	MbV3-w1.00-r160 (Ours)	TinyOpsV2	**111.3**	3.96	N	**68.19**	3942
F746	MCUNetV1-F746-int8 [8] (Repr.)	Internal	81.8	0.74	N	61.47	1838
	MNASNet-w1.00-r128 [10] (Repr.)	TinyOpsV2	103.5	4.38	N	68.01	1367
	MbV3-w1.00-r160 (Ours)	TinyOpsV2	111.3	3.96	N	**68.19**	**1307**
H743	MCUNetV1-H743-int8 [8] (Repr.)	Internal	125.9	1.7	N	67.9	1158
	MNASNet-w1.00-r144 (Ours)	TinyOpsV2	**142.8**	4.38	N	**69.55**	**1009**
	MCUNetV2-H743 [7]	Internal	256	<2	Y	71.8	-
	MNASNet-w1.00-r192 (Ours)	TinyOpsV2	**231.3**	4.38	N	**72.02**	1625

Table 2: Models from external design space have lower energy per inference and lower latency. Models in internal memory design space were deployed with Tensorflow Lite Micro (TFLM).

Platform	Model	Design Space	MACs (M)	Params (M)	Acc (%)	Current (mA)	Power (mW)	Energy (mJ)
F469	proxyless-w0.30-r152	Internal	23.75	0.72	51.96	51	255	308
	mbv3-w0.50-r112	**TinyOpsV2**	**16.28**	**1.33**	**52.37**	**74**	**370**	**250**
F746	proxyless-w0.30-r176	Internal	31.5	0.72	53.68	146	730	501
	mbv3-w0.55-r128	**TinyOpsV2**	**27.7**	**1.55**	**58.29**	**162**	**810**	**373**

Accuracy and Latency: We achieve 6.4% higher accuracy and 1.4x lower latency than MCUNetV1 on the F746 with the MbV3-w1.00-r160. Even though the MbV3-w1.00-r160 has higher MACs, the latency is lower due to computation in more efficient 3x3 DepConv operations. On the F469, MbV3-w0.75-r128 outperforms MNASNet-w1.00-r080 since it is a better scaling where the width is not too high for the resolution. For the H743, we used scalings of MNASNet since MbV3 was limited by maximum accuracy of 70.5% achieved by the base model. Compared to MCUNetV2 the mobile models yield up to 4% higher accuracy under the same MAC budget without quantisation aware training (QAT).

Energy Efficiency: We compare the required energy per inference of internal and external memory based solutions. We can observe in Table 2 that even though TinyOpsV2 has a higher power consumption, this is easily offset by the lower latency of the efficient models and scalings in the TinyOpsV2 design space leading to lower energy per inference in addition to higher accuracy.

5 Conclusions

In our study we showed that the constraint of internal storage leads to deployment of degenerate architectures which yield sub-optimal accuracy and latency. We proposed an efficient alternative of using available external memories with the overlaying strategy of TinyOpsV2. We outperformed prior internal memory approaches with up to 6.7% higher accuracy and 2.3x faster inference latency in addition to achieving 2.9% higher accuracy and 1.5x faster latency than previous external memory approaches to set the new state of the art in TinyML ImageNet classification. We demonstrate how it is possible to squeeze more performance out of low-powered MCUs for edge inference. There is ofcourse, the potential for misuse of this technology which could lead to negative societal impacts, however these are rather application dependent. We note that we manually selected scalings and architectures via hit and trial which might be sub-optimal and further performance might be achievable via NAS in the external memory design space. Nevertheless, we observed that architectures we deployed significantly outperformed prior approaches, suggesting that external memory based model design is the way forward in TinyML.

Acknowledgements

This work was supported by the UK Research and Innovation (UKRI) Centre for Doctoral Training in Machine Intelligence for Nano-electronic Devices and Systems [EP/S024298/1] and the Engineering and Physical Sciences Research Council (EPSRC) International Centre for Spatial Computational Learning [EP/S030069/1]. The authors acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work.

References

[1] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems*, 3, 2021.

[2] Alessandro Capotondi, Manuele Rusci, Marco Fariselli, and Luca Benini. Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(5):871–875, 2020.

[3] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems*, 3, 2021.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[6] International Data Corporation IDC. The growth in connected iot devices. *IDC Media Center*, 2019.

[7] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. *arXiv preprint arXiv:2110.15352*, 2021.

[8] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319*, 2020.

[9] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. 2021.

[10] Sulaiman Sadiq, Jonathon Hare, Partha Maji, Simon Craske, and Geoff Merrett. Tinyops: Imagenet scale deep learning on microcontrollers. In *Proceedings of the IEEE conference on computer vision and pattern recognition Workshop on Efficient Deep Learning for Computer Vision*, 05 2022.

[11] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

6 Paper Checklist

Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? Yes. We show limitations of internal memory in Section 2and they can be overcome through external memories in Section 3 with supporting experimental results in 4.

Have you read the ethics review guidelines and ensured that your paper conforms to them? Yes.

Did you discuss any potential negative societal impacts of your work? Yes (Section 5).

Did you describe the limitations of your work? Yes. As discussed, in Section 4, we use our insights to manually identify better architectures. However, other approaches, e.g. NAS algorithms, might be able to yield better performance.

Did you state the full set of assumptions of all theoretical results? Yes. We calculate how computation is scaled in architectures assuming that they are derived from the mobile search space with the structure of Mobile Inverted Conv blocks as shown in Figure 2 in Section 2.

Did you include complete proofs of all theoretical results? Yes. The calculations for how computation is scaled in architectures are in Section 2.

Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? No. Due to constraints from our funding sponsors, we are unable to share code.

Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? Yes. These are given in Appendix B.

Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? No. Due to compute constraints and the time required for ImageNet training we trained every architecture only once with the same set of hyper-parameters.

Did you include the amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? Yes. This is given in Appendix B.

If your work uses existing assets, did you cite the creators? Yes.

Did you mention the license of the assets? Yes

Did you include any new assets either in the supplemental material or as a URL? No.

Did you discuss whether and how consent was obtained from people whose data you're using/curating? N/A.

Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? N/A.

Did you include the full text of instructions given to participants and screenshots, if applicable? N/A.

Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? N/A.

Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? N/A.

A Deployment

A.1 Platforms

We used the STM32F746G Discovery, STM32F469I Discovery and STMH743I Eval boards for deployment in our experiments. We use ARM Cortex-M class devices including M33, M4 and M7 based devices. We used off-the-shelf development boards with SDRAM as volatile memory connected on the FMC interface. For non-volatile load memory we use NOR Flash on the QSPI interface, up to 16x larger than the internal flash as shown in Table 3.

Table 3: Cortex-M based off-the-shelf platforms used with varying specifications and constraints.

Platform	Architecture	Clock (MHz)	Internal			External	
			D-Cache (KB)	SRAM (KB)	Flash (KB)	FMC (KC)	O/QSPI (KB)
STM32F469	M4	180MHz	✗	256	1024	8192	8192
STM32F746	M7	216MHz	4KB	320	1024	8192	8192
STM32H743	M7	400MHz	16KB	512	2048	8192	8192

A.2 Energy Efficiency

Being development boards, the Discovery and Evaluation kits we used had a number of unprogrammable peripherals (LCD, Ethernet, etc) that weren't needed but contributed to the power consumption. Including these in our energy measurements would be disadvantageous to the internal memory deployment scenarios. For fair comparison, this power consumption was calibrated away by setting the MCUs to standby mode which has microWatt power consumption and adjusting accordingly.

B Training & Testing Details

We trained the networks on ImageNet for 150 epochs using the standard SGD optimizer with momentum 0.9 and weight decay 1e-4. A cosine annealing learning rate was used with a starting learning rate 0.05. Random resized cropping and random horizontal flipping was applied to the training data. Training was carried out on an internal 4-GPU setup with GTX1080TI and a 2-GPU setup with V100s. Training each model took approximately 20 hours. We used TensorFlow's int8 post-training quantization (both activation and weights are quantised to int8) with 500 samples from the training data used as the calibration dataset. For deployment, explicit padding layers of MCUNetV1 were fused to reduce latency for fair comparison. Models were deployed to the MCUs with TinyOpsV2 and TensorflowLite-Micro (TFLM) available under an Apache 2.0 license.