TinyOps: ImageNet Scale Deep Learning on Microcontrollers

Sulaiman Sadiq †
ss2n18@soton.ac.uk

Jonathon Hare †
jsh2@ecs.soton.ac.uk

Partha Maji ‡
partha.maji@arm.com

Simon Craske ‡
simon.craske@arm.com

Geoff V. Merrett †
gvm@ecs.soton.ac.uk

† University of Southampton, United Kingdom
‡ ARM Ltd, Cambridge, United Kingdom

Abstract

Deep Learning on microcontroller (MCU) based IoT devices is extremely challenging due to memory constraints. Prior approaches focus on using internal memory or external memories exclusively which limit either accuracy or latency. We find that a hybrid method using internal and external MCU memories outperforms both approaches in accuracy and latency. We develop TinyOps, an inference engine which accelerates inference latency of models in slow external memory, using a partitioning and overlaying scheme via the available Direct Memory Access (DMA) peripheral to combine the advantages of external memory (size) and internal memory (speed). Experimental results show that architectures deployed with TinyOps significantly outperform models designed for internal memory with up to 6% higher accuracy and importantly, 1.3-2.2x faster inference latency to set the state-of-the-art in TinyML ImageNet classification. Our work shows that the TinyOps space is more efficient compared to the internal or external memory design spaces and should be explored further for TinyML applications.

1. Introduction

Deep Neural Networks (DNN) have found success in a variety of fields [8, 9, 19]. At the same time, the number of connected IoT devices has been constantly growing with forecasts that there will be 41.6 billion connected IoT devices by 2025 [12]. This 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 IoT device. This is desirable for a number of reasons including security, privacy, latency and cost associated with using cloud based solutions.

A significant challenge in TinyML is the severe resource constraints which makes it difficult to achieve satisfactory accuracy and latency. As such, it is imperative that all of the available resources are utilised efficiently to maximise performance of the devices. Existing works [1, 4, 15] have focused on developing models that can be deployed using only fast internal memory. While these approaches have low inference latency, they suffer from low accuracy due to the constraint on model size imposed by internal memory. Higher accuracy can be achieved by deploying larger models using external memories available across the range of MCUs. However, while external alternatives are larger (10x↑), they are significantly slower (2x↑) which increases latency.

In our work, we propose an *expand* and *accelerate* approach to deploy models with high accuracy and low latency (Figure 1). We use external memories to expand the space of deployable architectures which can achieve high accuracy. To accelerate inference latency, we develop TinyOps, an inference engine that partitions operations in the model to meet internal memory constraints and overlays data between external and internal memory via DMA commonly available on MCUs. This enables deployment with high accuracy and low internal memory-like inference latency.

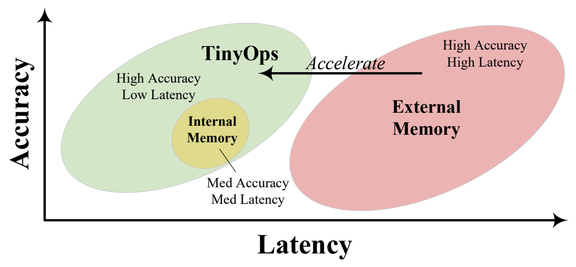

Figure 1. TinyOps accelerates inference of external memory models (2x↑). We find models from the TinyOps space significantly outperform internal memory models in accuracy *and* latency.

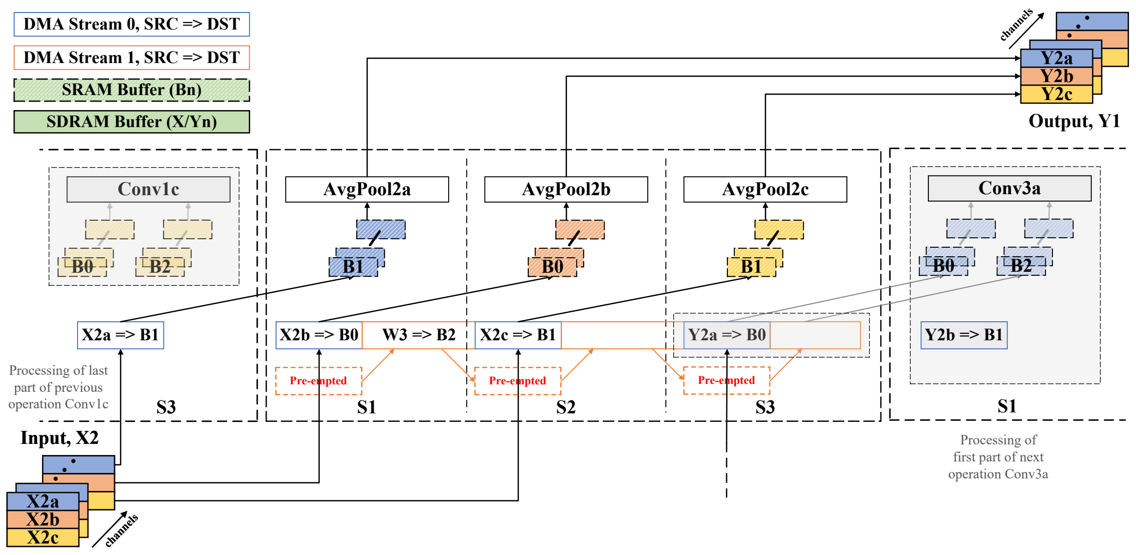

Figure 2. A pipeline is established between the DMA and CPU when executing operations in the inference graph. Inputs (X1) are partitioned and overlayed in smaller fast SRAM buffers. Filter weights are overlayed in parallel (W3=>B2) for further latency acceleration.

We find that architectures in the TinyOps design space, derived by simply scaling pre-designed models, are able to signficantly outperform models specially designed for internal memory with up to 6% higher accuracy and crucially 1.3-2.2x faster latency as well. This shows that the TinyOps space is full of efficient architectures that can be studied further for performance gains over sub-optimal internal memory only models. Additionally, compared to using external memories exclusively, TinyOps is able to accelerate inference 1.2-2x.

Related Work: A common approach to deploying larger models on MCUs includes quantisating weights and activations to reduce the memory footprint. Various fixed point precisions have been shown to work using fixed or mixed-bit schemes [4, 5, 11, 14, 24] in addition to vector quantisation approaches [7, 20]. We use standard INT8 quantisation due to the native support for 8-bit operands in the MCUs instruction sets. For maximum performance, architectures can be hand-crafted for low memory and computational footprint [2, 10, 18, 22]. Recently, automated architecture design using Neural Architecture Search (NAS) has also shown very good performance [1, 15, 17, 21, 25, 25]. In our experiments, we use different scalings of NAS designed models to vary complexity according to the required constraints or performance. MCUs have a number of inference frameworks [3, 6, 15, 16, 23], however, these perform inference with all weights and activations in either internal or external memory while some don't support the range of devices limiting portability. In our work, we utilise the TensorFlowLite-Micro (TFLM) framework with CMSIS-NN kernels [13] due to their open-source nature and portability across devices. Other existing kernels [4, 15, 16], are optimised at the lower level to maximise throughput, however we focus on high level model design which makes kernel optimisation an orthogonal approach.

2. TinyOps Architecture

The TinyOps engine seamlessly integrates into the software stack between the inference graph interpreter and the Hardware Abstraction Layer (HAL). When the interpreter invokes the operations (Conv, Add, Pool) in the inference graph, TinyOps executes the operations by interfacing with the low-level kernels and DMA via the HAL for portability. TinyOps uses external memory (SDRAM) as main memory and *overlays only frequently accessed data* in internal memory (SRAM) to reduce inference latency.

Execution Pipeline: TinyOps catches operator calls from the interpreter and partitions operations with large SRAM requirement into tiny operations which sequentially process smaller independent blocks or *tiny tensors* of the input tensor. The tiny tensors are produced by partitioning input tensors into equally sized blocks which are small enough to fit into fast internal SRAM. This enables tensors for tiny operations to be copied in from SDRAM to SRAM for faster inference. The tiny tensor copying is offloaded to the DMA which operates independently of the CPU. Using a ping-pong buffering strategy, a pipeline is established between the DMA and CPU to efficiently execute the NN operation (Figure 2). While the CPU processes the first block of the data, the DMA prepares the second block of

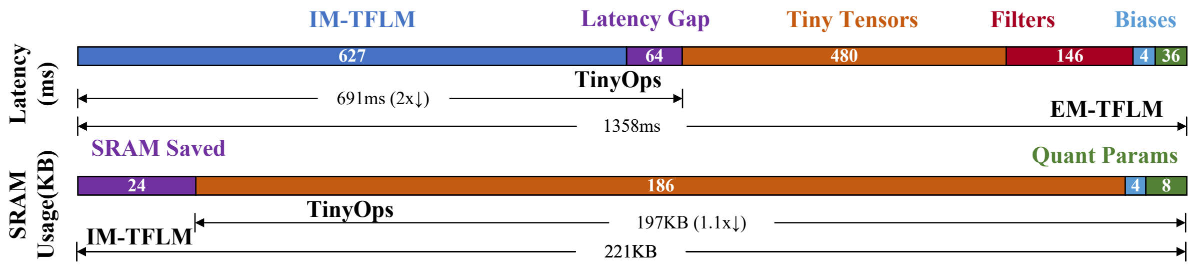

Figure 3. TinyOps accelerates latency 2x compared to external memory (EM-TFLM) by overlaying only frequently accessed data.

data by copying in from SDRAM to SRAM. After finishing processing of the first block of data, the CPU frees the first buffer and processes the second block of data while the DMA begins to load the third data part into the newly freed buffer by the CPU and so on. The tiny operations with reduced peak SRAM usage are sequentially executed in this manner until the entire operation has been performed.

Partitioning Strategy: Tensor partitioning is performed in the H dimension since TFLM uses the NHWC data layout format. The H dimensional partitioning therefore produces tiny tensors occupying contiguous memory blocks which avoids 2D DMA transfers not natively supported by on-chip DMAs on MCUs. TinyOps uses a memory budget adaptive partitioning strategy to determine buffer sizes according to the diverse constraints of different platforms. The SRAM buffer sizes are determined by the largest tensor in the model which is partitioned into equally sized tiny tensors according to the constraint. The remaining operations are then partitioned into the largest tiny tensors that fit the SRAM buffers.

Data Overlaying: TinyOps' ping-pong buffering strategy requires four SRAM buffers for overlaying tiny input tensors of two-input Operations (Add). We are able to further reduce latency as shown in Figure 3 by overlaying filters and biases of one-input parametric operations (Conv, DepConv) from external memory in the otherwise unused buffers. Additionally, TinyOps overlays frequently accessed quantisation parameters. By using pre-emptive scheduling between multiple DMA streams (Figure 2), the overlays are achieved using only a single DMA of the platforms.

3. Experiments and Results

Experimental Setup: We compare accuracy, latency and energy efficiency of models deployed with TinyOps and TFLM using internal/external memory (IM/EM-TFLM) on ImageNet classification. We demonstrate the strength of models in TinyOps' design space by comparing scaled variations of mobile models (ProxylessNAS, MobileNetV3, MNASNet) in the TinyOps space with the optimal scaling for internal memory and the MCUNet family of models. MCUNet models were taken from the authors repository [15] and deployed with TinyOps as their memory footprint was too high for IM-TFLM. All models were quantised with standard INT8 post-training quantisation and deployed to ARM Cortex M4 and M7 based MCUs. Internal memory and storage were supplemented using SDRAM and ExtFlash on the Flexible Memory Controller (FMC) and Quad Serial Port Interface (QSPI) peripherals respectively with sizes as shown in Table 1.

Table 1. Memory specs of MCU platforms used in experiments.

PLATFORM	INTERNAL MEM (KB)		EXTERNAL MEM (KB)	
	SRAM	FLASH	SDRAM	EXTFLASH
STM32F469	256	1024	8192	8192
STM32F746	320	1024	8192	8192

ImageNet Classification: Compared to the MCUNet-F746 model designed for internal memory, we achieve up to a 2% increase in accuracy whilst simultaneously reducing latency by 2.2x on the F746 with MNASNet-w1.0-r96 as shown in Table 2. TinyOps memory budget adaptive partitioning strategy is also able to deploy the MNASNet-w1.0-r96 model within the tighter memory constraint of the F469 where we observe a 4% higher accuracy than MCUNet-F469 with 1.2x less latency. Similarly, the optimal scalings of a ProxylessNAS model for internal memory on both devices are outperformed by better scalings found in the TinyOps space. We note that in these cases, the TinyOps models have lower complexity (MACs) than the corresponding models which might yield lower latency. However, this latency decrease does not purely occur due to the lower number of MACs. As seen, MNASNet-w1.0-128 outperforms MCUNet-F746 with 6% higher accuracy and 1.3x lower latency with 1.3x more MACs. This puts into question the assumptions of prior works that end to end latency or accuracy is correlated with the FLOPs or MACs of a model [1, 15]. It is possible that the assumptions may hold true for only simplistic memory hierarchies and not carry over to more

Table 2. Models derived from the TinyOps space outperform optimally scaled models from the internal memory design space (proxyless-w0.25-r160, proxyless-w0.25-r192)[1]. TinyOps models also outperform the MCUNet family of models in accuracy and latency on both devices. The model stats show the amount of memory required to store the weights (Size) and the RAM required for activation tensors.

Platform	Model Stats					Deployment Stats			
	Name	Design Space	Size (MB)	RAM (KB)	MACs (M)	Int8 Acc (%)	Latency (ms)	Current (mA)	Energy (mJ)
F469	Proxyless-w0.25-r160	Internal	0.72	218	16.1	46.0	1007	41	206
	Proxyless-w0.40-r080	**TinyOps**	1.27	171	10.9	**46.6**	**628**	75	236
	MbV3-w0.40-r096	**TinyOps**	1.13	217	8.9	**46.1**	**512**	77	197
	MCUNet-F469	Internal	0.96	468	67.3	60.3	3391	70	1186
	MNASNet-w1.00-r080	**TinyOps**	4.72	330	48.2	**60.8**	**2575**	82	1056
	MNASNet-w1.00-r096	**TinyOps**	4.72	397	58.8	**64.0**	**2922**	81	1183
F746	Proxyless-w0.25-r192	Internal	0.72	286	23.1	49.2	623	96	299
	Proxyless-w0.85-r064	**TinyOps**	3.48	303	21.5	**50.5**	**410**	128	262
	MCUNet-F746	Internal	0.97	492	81.8	61.8	1867	124	1157
	MNASNet-w1.00-r096	**TinyOps**	4.72	397	58.8	**64.0**	**866**	126	546
	MNASNet-w1.00-r128	**TinyOps**	4.72	568	103.5	**68.2**	**1423**	127	904

Table 3. Latency, Current and Energy/Inference statistics measured with proxyless-w0.25-r160 and proxyless-w0.25-r192 on the F469 and F746. Even with higher current consumption on the F746, TinyOps hybrid approach has lower energy per inference due to lower latency.

Platform	Latency (ms)			Current (mA)			Energy (mJ)		
	Im-Tflm	Em-Tflm	TinyOps	Im-Tflm	Em-Tflm	TinyOps	Im-Tflm	Em-Tflm	TinyOps
F469	1007	2167	1059	41	78	72	206	845	381
F746	623	891	687	96	120	130	299	535	447

holistic views of the MCUs memory architecture. However, we leave a deeper latency analysis as a future work.

Energy Efficiency: TinyOps is able to reduce energy per inference on both devices 1.3-2.2x compared to EM-TFLM by accelerating the inference latency as seen in Table 3. On the F469, TinyOps lowers the current draw by 6mA compared to EM-TFLM even though it uses the additional DMA peripheral of the microcontroller. This occurs due to multiple reads of filters and input tensor elements from SDRAM in Conv and DepConv operations which have higher energy consumption. On the other hand, TinyOps overlays inputs and filters from SDRAM to SRAM using the DMA such that there is only one read from SDRAM and any repeated reads are from SRAM which have a lower energy cost. This gives an overall reduction in current draw. The same behavior is not observed in the F746 as the on-chip cache is able to store frequently accessed SDRAM data and provide low energy access from cache itself. In this case, the overall consumption of SDRAM decreases and is outweighed by the DMA leading to higher current draw. Compared to IM-TFLM, TinyOps has minimal latency overhead. The current draw is higher due to the extra power consumption of SDRAM and the DMA. However, this can be offset by efficient models in the TinyOps space with the same or higher accuracy as shown in Table 2.

[1]We evaluated all scalings on the pareto frontier for Proxyless, Mbv2, Mbv3 and MNASNet out of which Proxyless gave highest accuracy for the internal memory design space.

4. Conclusion and Future Work

We introduced TinyOps, which accelerates inference of high accuracy architectures in external memory by a partitioning and overlaying scheme which uses the on-chip DMA in parallel with the CPU to maximise throughput. The partitioning scheme was shown to be able to handle diverse memory budgets enabling deployment over a range of devices. TinyOps outperformed the previous state of the art internal memory models on ImageNet classification with up to 6% higher accuracy and 2.2x faster inference. The significant performance gains demonstrate the efficiency of the TinyOps space and shows that focusing on internal memory only deployment is sub-optimal for high performance imagenet scale deep learning on MCUs. This opens up interesting avenues of future work including using NAS to derive efficient architectures from the TinyOps space as opposed to simply scaling pre-designed mobile models. Deeper latency analysis of the models used in this work can also give insights into manual model and search space design for MCUs. At the lower level, kernels could be optimised orthogonally to the models to further accelerate inference of the TinyOps space.

5. Acknowledgments

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]. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising. The authors acknowledge the use of the IRIDIS High Performance Computing Facility, in the completion of this work. Data for this work is available at https://doi.org/10.5258/SOTON/D2188.

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] Andrew Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.

[3] Alessio Burrello, Angelo Garofalo, Nazareno Bruschi, Giuseppe Tagliavini, Davide Rossi, and Francesco Conti. Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus. *IEEE Transactions on Computers*, 70(8):1253–1268, 2021.

[4] 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.

[5] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[6] 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.

[7] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320*, 2020.

[8] 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.

[9] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdelrahman 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.

[10] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[11] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

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

[13] Liangzhen Lai, Naveen Suda, and Vikas Chandra. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601*, 2018.

[14] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[15] 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.

[16] ST Microelectronics. Stm32x-cube-ai. 2019.

[17] Sulaiman Sadiq, Partha Maji, Jonathan Hare, and Geoff Merrett. Deff-arts: Differentiable efficient architecture search. 2020.

[18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[19] 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.

[20] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. *arXiv preprint arXiv:1907.05686*, 2019.

[21] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[22] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[23] uTensor. utensor.

[24] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. 2011.

[25] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.