Efficient CNN Inference on Ultra-Low-Power MCUs via Saturation-Aware Convolution

Shiming Li
Uppsala University
Sweden
shiming.li@it.uu.se

Luca Mottola
Politecnico di Milano
Italy
luca.mottola@polimi.it

Yuan Yao
Uppsala University
Sweden
yuan.yao@it.uu.se

Stefanos Kaxiras
Uppsala University
Sweden
stefanos.kaxiras@it.uu.se

***Abstract*—Quantized CNN inference on ultra-low-power MCUs incurs unnecessary computations in neurons that produce saturated output values. These values are too extreme and are eventually clamped to the boundaries allowed by the neuron. Often times, the neuron can save time by only producing a value that is extreme enough to lead to the clamped result, instead of completing the computation, yet without introducing any error. Based on this, we present *saturation-aware convolution*: an inference technique whereby we alter the order of computations in convolution kernels to induce earlier saturation, and value checks are inserted to omit unnecessary computations when the intermediate result is sufficiently extreme. Our experimental results display up to 24% inference time saving on a Cortex-M0+ MCU, with zero impact on accuracy.**

I. Introduction

TinyML applications such as convolutional neural networks (CNNs) empowers complex data processing on ultra-low-power devices under power-scarce environments [1, 2, 3]. The CNN inference latency translates almost linearly to energy consumption [4, 5] as these microcontrollers (MCUs) mostly lack features such as DVFS [6, 7].

Time is wasted on neurons producing saturated values. The majority of CNNs' inference time is spent on multiply-accumulate operations (MACs) in convolutional and fully-connected layers. Their operation can be generally described as $a = bias + \sum_{i=0}^{m-1} x_i \cdot w_i$, where a is the accumulation, m is the number of computation steps, x_i are the inputs, w_i and $bias$ represent the weights and bias, respectively. However, a is not fed to the output neuron directly. It may be too large or too small, and causes the output neuron to saturate. Then, the actual value of the neuron is clamped within specified boundaries, which can be introduced by, for example, the value range of the neuron's data type. Computation time is wasted, because regardless of a, a saturated neuron's value is fixed.

We exemplify this in Figure 1, showing the computation trace of a convolution operation for an output neuron in the hand gesture recognition CNN from the STM32 AI Model Zoo [8]. Since the output is of type `int8`, the result of the convolution is clamped to -128. The last 10 steps are wasted.

This work is supported by the Swedish Foundation for Strategic Research (SSF) grant FUS21-0067. Part of the data analysis was enabled by resources in project UPPMAX 2025/2-258 provided by Uppsala University at UPPMAX.

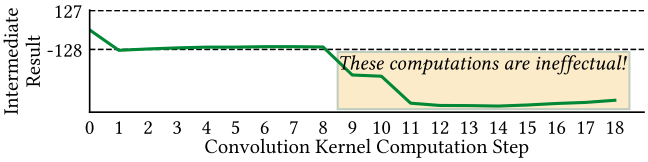


Fig. 1: An example of the computation trace of a convolution operation, where the output neuron's final value saturates.

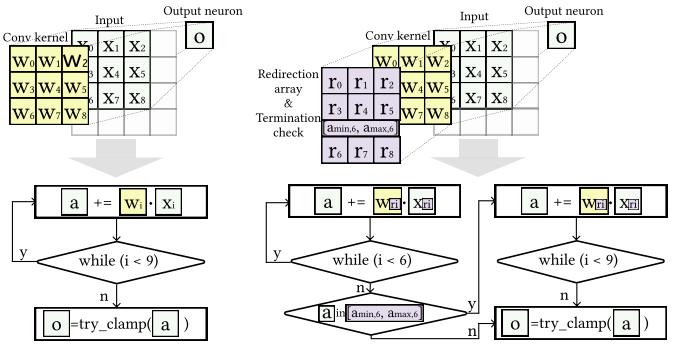
We find that this phenomenon is common in CNNs on ultra-low-power MCUs, which are typically quantized networks where the data types have smaller value ranges and often use ReLU-like activation functions, further tightening the boundaries. CNNs like GMP [8] also have dynamic boundaries [9] that bear similar effects. Existing literature extensively explores redundancy in neural networks, for example, in weights or connections [10, 11, 12], value precision [11, 13, 14, 15], and even feature maps [16, 17] or network structure [18, 19, 20, 21, 22, 9]. Most of these solutions fail to exploit redundancy at per-neuron granularity. Closest to our work is SnaPEA [23]. It analyzes neuron-level redundancy by terminating computations that cannot produce a positive result, by relying on the condition that the inputs of each layer are all positive values.

We omit computations from saturated neurons. We propose an inference technique which executes convolution kernels while dynamically omitting the unnecessary computations in saturated neurons *without introducing error*. We integrate our design into a modified, arm-v6m compatible version of TinyEngine, the code generation and inference engine of the state-of-the-art MCUNet [24]. We conduct experiments on a STM32 development board with a Cortex-M0+ MCU and observe up to 24% time saving across 7 CNNs we test.

II. Saturation-Aware Convolution

We propose saturation-aware convolution, where convolution kernels allow neurons that would definitely produce a saturated value to terminate computation early, without error.

Omitting ineffectual computations with zero error. We only allow terminating a convolution operation when the intermediate result is too extreme, and is bound to cause saturation. This can be done for any computation step by checking the most extreme possible future result. For example, in the case of an



(a) *Conventional convolution operation.* (b) *Saturation-aware convolution.*

Fig. 2: Conventional and saturation-aware convolution.

`int8` CNN, any future input value x is in $[-128, 127]$. Already knowing the weight value w_{i+1} at computation step $i+1$, the result of this step must be within $[-128 \cdot w_{i+1}, 127 \cdot w_{i+1}]$ if w_{i+1} is non-negative, or $[127 \cdot w_{i+1}, -128 \cdot w_{i+1}]$ if w_{i+1} is negative. Summing up all possible limits of future results, we know a_i's future deviation will not exceed $[d_{min,i}, d_{max,i}]$, where $d_{min,i} = \sum_{j=i+1}^{m-1} \begin{cases} -128 \cdot w_j, \text{if } w_j \geq 0 \\ 127 \cdot w_j, \text{if } w_j < 0 \end{cases}$, and $d_{max,i} = \sum_{j=i+1}^{m-1} \begin{cases} 127 \cdot w_j, \text{if } w_j \geq 0 \\ -128 \cdot w_j, \text{if } w_j < 0 \end{cases}$.

Note that values involved to compute $d_{min,i}$ and $d_{max,i}$ are compile-time knowledge, easing the need for extra computation at run-time. A convolution kernel can practically utilize the information of $d_{min,i}$ and $d_{max,i}$ to compare a_i against a range $[a_{min,i}, a_{max,i}]$, where $a_{min,i} = -128 - d_{max,i}$ and $a_{max,i} = 127 - d_{min,i}$. If a_i has left the range, it means even the most extreme future deviation of a cannot bring it back into $[-128, 127]$, and a_i can be deemed "too extreme" and will definitely lead to saturation.

Reordering computations to induce earlier saturation. The computation order in a convolution kernel affects the changes of intermediate results. Intuitively, we prefer the intermediate value of a to stabilize more quickly [25], leaving less space for deviation in the future. Thus, we reorder the computation in a convolution kernel by the absolute value of weight w_i.

The order does not affect the final result of a, since in a quantized CNNs with integer types, the additions and multiplications are commutative. However, it effectively increases the percentage of unnecessary computations that can be omitted without introducing error. According to our experiments, this is increased by four times, from 5% to 20%, as discussed next.

Saturation-aware convolution execution flow. We illustrate and compare conventional and saturation-aware convolution in Figure 2. The output neuron's value o is produced by filtering out any extreme value of a through clamping. The example saturation-aware convolution operation in Figure 2b executes convolution with extra information. In this example, a termination check is inserted at the sixth computation step. The check examines the intermediate result of a against $[a_{min,6}, a_{max,6}]$, and if the value is too extreme, the convolution operation will be terminated, and the intermediate value of a_6 will be used to produce the result to be fed to the neuron. Moreover, instead of executing $x_i \cdot w_i$ in the natural number order, Figure 2b introduces a redirection array to induce earlier saturation.

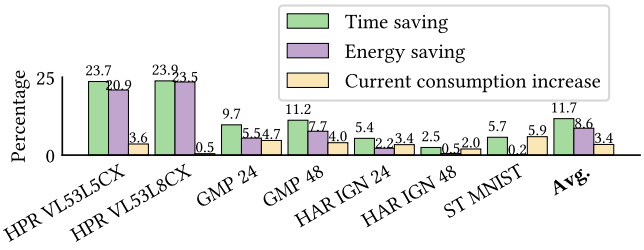


Fig. 3: Inference latency reduction, current consumption increase and energy saving with saturation-aware convolution.

III. Evaluation

We implement saturation-aware convolution and evaluate CNN model inference speed and energy consumption. Experiments are conducted on the STM32 NUCLEO-G0B1RE development board with a Cortex-M0+ MCU, using 7 lightweight CNNs from the open-source STM32 Model Zoo [8, 26].

Implementation. We implement scripts to analyze models, generate the computation order and find the layers where to apply saturation-aware convolution. We allow up to 2 checks per kernel, and we insert them in each convolution kernel at the positions where the most computation steps can be omitted statistically, which we decide by profiling the kernels' behavior with sample inputs not used in the evaluation.

We integrate saturation-aware convolution into a modified, arm-v6m compatible version of TinyEngine [24], the state-of-the-art neural network code generation and inference engine for MCUs. TinyEngine takes in CNN models in TenforFlow Lite `.tflite` format, and generates `.c` code that can be compiled and run on MCUs.

Results. Saturation-aware convolution effectively reduces inference latency and gains energy saving. We display the time and energy saving in Figure 3. We achieve *up to 23.9%* time saving, and up to 23.5% energy saving on HPR VL53L8CX, averaging 11.7% and 8.6%, respectively, across all workloads. We observe only a small increase in current consumption with saturation-aware convolution, only 3.6% on average, as in Figure 3 (yellow bar). We compare the CNNs's outputs for each single experiment with the baseline, and verify that our technique introduces *strictly zero error*.

IV. Conclusion

We present saturation-aware convolution, where a convolution kernel executes the computations in an altered order to induce saturation and predicts saturation dynamically via infused compile-time information, without introducing error. Based on experiments involving 7 CNN workloads, our technique shows up to 24% inference latency reduction and up to 23.5% energy saving, with zero impact on accuracy.

REFERENCES

[1] N. A. Bhatti, M. H. Alizai, A. A. Syed, and L. Mottola, "Energy Harvesting and Wireless Transfer in Sensor Network Applications: Concepts and Experiences," *ACM Trans. Sen. Netw.*, vol. 12, Aug. 2016.

[2] S. Ahmed, B. Islam, K. S. Yildirim, M. Zimmerling, P. Pawełczak, M. H. Alizai, B. Lucia, L. Mottola, J. Sorber, and J. Hester, "The Internet of Batteryless Things," *Commun. ACM*, vol. 67, pp. 64–73, Feb. 2024.

[3] H. R. Mendis, K. S. Yıldırım, M. Zimmerling, L. Mottola, and P.-C. Hsiu, "Special session-intermittent tinyml: Powering sustainable deep intelligence without batteries," in *ACM SIGBED International Conference on Embedded Software (EMSOFT)*, 2025.

[4] W. Song, S. Kaxiras, T. Voigt, Y. Yao, and L. Mottola, "TaDA: Task Decoupling Architecture for the Battery-less Internet of Things," in *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*, SenSys '24, (New York, NY, USA), p. 409–421, Association for Computing Machinery, 2024.

[5] D. Romano, L. Mottola, and T. Voigt, "Neuro-C: Neural Inference Shaped by Hardware Limits," in *Proc. of ACM EuroSys*, 2026.

[6] S. Ahmed, A. Bakar, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola, "The Betrayal of Constant Power× Time: Finding the Missing Joules of Transiently-Powered Computers," in *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, pp. 97–109, 2019.

[7] S. Ahmed, A. Qurat, J. Siddiqui, L. Mottola, and M. H. Alizai, "Intermittent Computing with Dynamic Voltage and Frequency Scaling," in *Proceedings of the 2020 International Conference on Embedded Wireless Systems and Networks*, pp. 1–12, Junction Publishing, 2020.

[8] STMicroelectronics, "STM32 Model Zoo." https://github.com/STMicroelectronics/stm32ai-modelzoo, 2023. Accessed on August 20, 2025.

[9] M. Song, J. Zhao, Y. Hu, J. Zhang, and T. Li, "Prediction Based Execution on Deep Neural Networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 752–763, 2018.

[10] S. Han, J. Pool, J. Tran, and W. Dally, "Learning Both Weights and Connections for Efficient Neural Network," *Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, vol. 28, 2015.

[11] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv preprint arXiv:1510.00149*, 2015.

[12] R. Barjami, A. Miele, and L. Mottola, "Intermittent Inference: Trading a 1% Accuracy Loss for a 1.9x Throughput Speedup," in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 647–660, 2024.

[13] M. Courbariaux, Y. Bengio, and J.-P. David, "Training Deep Neural Networks with Low Precision Multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[14] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *Journal of Machine Learning Research*, vol. 18, no. 187, pp. 1–30, 2018.

[15] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2704–2713, 2018.

[16] T. Verelst and T. Tuytelaars, "Dynamic Convolutions: Exploiting Spatial Sparsity for Faster Inference," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2317–2326, 2020.

[17] R. Liu, Y. Leng, S. Tian, S. Hu, C.-F. R. Chen, and S. Yao, "DynaSpa: Exploiting Spatial Sparsity for Efficient Dynamic DNN Inference on Devices," in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 422–435, 2024.

[18] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks," in *International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, IEEE, 2016.

[19] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-Scale Dense Networks for Resource Efficient Image Classification," in *International Conference on Learning Representations (ICLR)*, 2017.

[20] M. Wołczyk, B. Wójcik, K. Bałazy, I. T. Podolak, J. Tabor, M. Śmieja, and T. Trzcinski, "Zero Time Waste: Recycling Predictions in Early Exit Neural Networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2516–2528, 2021.

[21] S. Jeon, Y. Choi, Y. Cho, and H. Cha, "HarvNet: Resource-Optimized Operation of Multi-Exit Deep Neural Networks on Energy Harvesting Devices," in *International Conference on Mobile Systems, Applications and Services (MobiSys)*, pp. 42–55, 2023.

[22] Y. Wu, Z. Wang, Z. Jia, Y. Shi, and J. Hu, "Intermittent Inference with Nonuniformly Compressed Multi-Exit Neural Network for Energy Harvesting Powered Devices," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

[23] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R. K. Gupta, and H. Esmaeilzadeh, "SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 662–673, 2018.

[24] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han, *et al.*, "MCUnet: Tiny Deep Learning on IoT Devices," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11711–11722, 2020.

[25] F. Bambusi, F. Cerizzi, Y. Lee, and L. Mottola, "The Case for Approximate Intermittent Computing," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 463–476, 2022.

[26] STMicroelectronics, "STM32 Model Zoo Services." https://github.com/STMicroelectronics/stm32ai-modelzoo-services, 2023. Accessed on August 20, 2025.