Ultra-low Power DNN Accelerators for IoT: Resource Characterization of the MAX78000

Arthur Moss[†*°] Hyunjong Lee[§*°] Lei Xun[◇°] Chulhong Min[‡] Fahim Kawsar[‡¶] Alessandro Montanari[‡]
[†]Newcastle University [§]KAIST [◇]University of Southampton [¶]University of Glasgow [‡]Nokia Bell Labs, Cambridge (UK)
[*]Indicates equal contribution. [°]Work done while authors were at Nokia Bell Labs, Cambridge (UK).

ABSTRACT

The development of edge devices with dedicated hardware accelerators has pushed the deployment and inference of Deep Neural Network (DNN) models closer to users and real-world sensory systems than ever before (e.g., wearables, IoT). Recently, a further subset of these devices has emerged: ultra-low power DNN accelerators. These microcontrollers possess a dedicated hardware accelerator and are able to operate with only μJ's of energy in milliseconds of time. With their small form-factor, such devices could be used for battery-powered machine learning (ML) applications. In this work, we take a close look at one such device: the MAX78000 by Maxim Integrated. We characterize the device's performance by running five DNN models of various sizes and architectures, and analyze its operational latency, power consumption, and memory footprint. To better understand the performance characteristics, we take a step further and investigate how different layer types (operation type, kernel size, number of input and output channels) and the selection of accelerator processors affect the execution time.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; **Embedded software**; • **Computing methodologies → Neural networks**.

KEYWORDS

Resource characterisation, edge accelerators, neural networks.

ACM Reference Format:
Arthur Moss, Hyunjong Lee, Lei Xun, Chulhong Min, Fahim Kawsar, Alessandro Montanari. 2022. Ultra-low Power DNN Accelerators for IoT: Resource Characterization of the MAX78000. In *Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT) (SenSys '22), November 6–9, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3560905.3568300

1 INTRODUCTION

The development of deep learning [19] has shown great progress in the last decades. Nowadays, deep learning is used in many application domains, such as computer vision [11, 31], natural language processing [4], activity recognition [24] and user interface [20]. Deep learning algorithms can deliver human-level accuracy for these applications, however, they are both computationally intensive and memory access intensive.

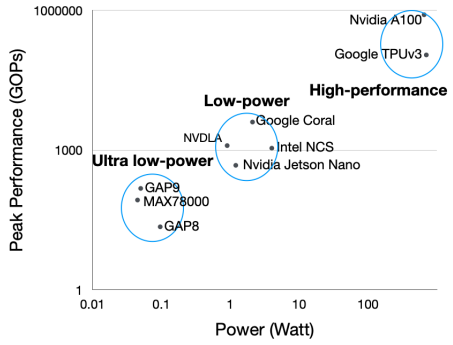

Figure 1: Peak performance of deep learning hardware accelerators [15]. Most accelerators can be categorized into 3 classes: high-performance cloud/servers, low-power mobile and embedded, and the ultra low power.

To address the computational inefficiency, many software-based solutions have been proposed to modify deep neural networks (DNNs) with the objective of fitting them in memory, increasing execution speed, and decreasing energy demand [5, 9, 17, 25, 26, 39]. However, with the growing popularity of deep learning models and increasing complexity demands from applications, novel HW architectures started to emerge to specifically accelerate DNN-based workloads [32, 34]. These dedicated accelerators typically contain large on-chip memory to reduce expensive off-chip memory accesses, large computational arrays for matrix multiplication and addition, dedicated data flow between computation cores for efficient data reuse and support low precision computing such as int-8. Initially, these hardware accelerators were designed for and deployed in the cloud (e.g., Google TPU [14]). However, in recent years, hardware accelerators are moving into mobile/embedded devices to take advantage of local computation for the low latency, energy efficiency, and privacy preservation, e.g., Apple A16 SoC [22], Nvidia Jetson [28], Intel Neural Computing Stick [13], Google Edge TPU [7]), and tiny MCU devices, (e.g., Maxim MAX78000 [12], Arm Ethos [2] and Greenwaves GAP-8 [8]). Figure 1 shows the peak performance and power of various accelerators.

Cloud and mobile/embedded accelerators have been studied and characterized in previous works [1, 21, 23, 30, 37, 38]. However, there are very few benchmark studies for tiny-scale accelerators integrated into microcontrollers (MCUs) for ultra-low power inference. This paper focuses on the performance and resource characterization of a board MAX78000 [12], which contains an Arm Cortex-M4F core, a RISC-V core, and a convolution neural network (CNN) accelerator, integrated into one package. We choose the board with a better performance among the two commercially available boards (MAX78000 and GAP-8). Arm Ethods micro-NPUs

SenSys '22, November 6–9, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9886-2/22/11…$15.00
https://doi.org/10.1145/3560905.3568300

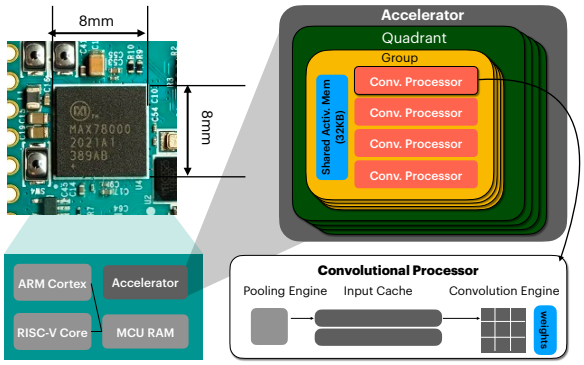

Figure 2: MAX78000 and its overall architecture.

and Greenwaves GAP-9 are not available at the time this work is done. The device supports open-source toolchains that can be used for research purposes. Previous studies and benchmarks for this device have focused only on the high-level benchmark with a limited number of DNN models [6, 27]. In our paper, we conducted detailed benchmarks and characterization at both macro and micro levels to gain new insights regarding hardware-aware model design and the breaking down of the performance. For the macro-benchmarks, 5 models varying in architecture and size with 4 different datasets (MNIST, CIFAR100, CamVid, FaceID) were characterized for latency, power, and memory footprint. The paper offers empirical insights and guidelines for practitioners and researchers interested in deploying DNN models on the MAX78000 platform for ultra-low power sensory tasks.

2 MAX78000 PLATFORM

2.1 Hardware Overview

While a number of tiny-scale accelerators have been recently proposed, e.g., MAX78000 [12], ethos-u55 and ethos-u65 [2], and GAP-8 [8], there are yet only a few products that are commercially available and provide access to and control over the underlying operations. In this study, we chose the MAX78000 platform [12] (Figure 2 (left)), developed by Maxim Integrated [1] in 2020, because it is readily available and offers open source tools and documentation useful for an in-depth analysis of its performance.

The MAX78000 is an AI microcontroller designed to run neural networks at ultra-low power on tiny-scale devices. It is composed of two major processing hardware modules: a dual-core microcontroller (MCU) and a convolutional neural network (CNN) accelerator. On the controller side, it has an Arm Cortex-M4 processor with FPU running at up to 100 MHz and a 32-Bit RISC-V co-processor running at up to 60MHz. In terms of memory, it is equipped with 512 KB of Flash and 128 KB of SRAM.

Figure 2 (right) shows the overall architecture of the CNN accelerator. The hardware-based CNN accelerator consists of 64 convolutional processors, weight storage memory of 442 KB, and data memory of 512 KB. Each convolutional processor consists of a pooling engine, input cache, weight memory, and convolution engine, and is responsible for running a convolutional operation of a single channel. These convolutional processors are grouped by four, and processors in a group share a *shared activation memory* for data input and activation data (8 bits for each processor in a 32-bit word). Four of these groups are further grouped into a quadrant. A quadrant has registers to program the execution configuration for each layer, and allows developers to configure execution details including the processor selection, kernel address, input data address, output memory address, and so on. This sub-division in quadrants, groups and processors allows fine control over which parts of the accelerator are active to further reduce power consumption.

2.2 Compilation and Execution Flow

Since training is not supported on the MAX78000, the model needs to be trained and compiled on a desktop environment, and the inference execution is performed on the chip with the model binary.

Compilation: MAX78000 supports PyTorch for model development. Maxim offers custom layer implementations for quantized and fused operations as well as pipelines for quantization-aware training or post-training quantization. After training, dedicated tools convert PyTorch checkpoint or Tensorflow-exported ONNX files to C code that can be compiled and executed on the MAX78000.

MAX78000 has a specific set of accelerated operations, which are as follows: convolution 1D with kernel sizes 1 to 9, convolution 2D with kernel sizes 1 by 1 or 3 by 3, linear (fully connected layer), max pooling, average pooling, ReLU activation, Abs activation, and batch norm. Other operations can be executed on the ARM or RISC-V cores but will incur latency penalties.

Execution: When MAX78000 executes a model, the MCU transfers accelerator configuration, input data, and weights to the CNN accelerator. Then, the accelerator executes each layer of the model in series. For each layer, it loads the kernels from its kernel memory and assigns each input channel to one convolutional processor. The acceleration is achieved by running processors in parallel.

We further explain how a unit operation is executed on one convolutional processor. First, the pooling engine reads input or activation data from the shared activation memory in a group to which the processor belongs, and performs a (max or avg) pooling operation. Second, if a pooling result can be used by adjacent convolutional operations, it is pushed into the input cache for optimization. Third, the convolution engine executes the convolutional operation. Its result is delivered to the quadrant to which the processor belongs and per-quadrant aggregated results are further sent to the master quadrant. The entire sum of products is written to the shared activation memory of convolutional processors in the next layer so that it can be used as an input for the next layer.

MAX78000 has an advanced feature (also called FIFO Processing) for larger inputs whose input and the feature map cannot fit in its activation memory. In this feature, the data is streamed into the accelerator first-in first-out. The accelerator computes the partial data, up to a specified layer, which has a smaller feature map that can fit in the activation memory of the accelerator. This is using the characteristic of CNN models that the feature map size decreases in its later layers. This is useful when the accelerator has to process larger data (up to 1024K pixels) than its capacity (8K pixels). One of the models we use for benchmarking uses this feature.

[1] https://www.maximintegrated.com

Table 1: Models used for the macro benchmarks.

Model	Layer Count	Accuracy	Hardware Operations (Millions)	Weight Memory (KB)	Peak Activation Memory (KB)	Dataset (Input Size)
ConvNet5	5	99.44 (Top-1) 100 (Top-5)	10.90	71.20	62.4	MNIST [18] (28x28x3)
FaceIDNet	9	78.9 (Female) 88.9 (Male)	56.30	176.10	460.8	FaceID [29] (120x160x3)
SimpleNet	14	54.2 (Top-1) 82.79 (Top-5)	18.50	381.80	40.96	CIFAR100 [16] (32x32x3)
ResSimpleNet	17	51.33 (Top-1) 78.85 (Top-5)	18.60	381.80	61.44	CIFAR100 [16] (32x32x3)
UNet	19	91.05 (pixel-wise classification)	187.80	281.30	294.91	CamVid [3] (48x48x48)

3 MACRO BENCHMARKS

To design intelligent services on tiny devices, we need to know the capabilities of the accelerators, and its time and energy cost for running. We design and conduct a set of benchmarks aimed at characterizing the performance and resource usage of the MAX78000. Here we consider them at a macro level, analyzing end-to-end latency and power consumption as well as overall memory footprint.

3.1 Experimental Setup

Equipment: The equipment included: a MAX78000 Evaluation Kit[2] (EV Kit), a High Voltage Power Monitor[3] (by Monsoon Solutions), and a workstation. The EV Kit was selected because it exposes power pins used to power only the MAX78000 chip. The regulators and other peripherals on the EV Kit were disabled and the board was powered through the Monsoon power monitor set at 3.3V. This enabled us to monitor the power drawn only by the MAX78000 chip. The power meter we used has a sampling rate of 5kHz. Since the inference time, from model and data loading to model execution, is from 8.3ms to 60.7ms, this sampling rate has sufficient granularity for precise measurements. In other words, the data can yield an accurate breakdown of power consumption per main operations involved in executing an inference.

Models: Five CNN models of different architectures and sizes were selected for the benchmark experiment. All of the models were for vision applications (recognition and classification), and a total of four different data sets were used. The characteristics of the models and the data sets used for training are reported in Table 1.

The models were selected to ensure a variation in their architecture and size and to ensure that the effect of different data dimensions could be analyzed. These include classic feed-forward architectures (ConvNet5, FaceIDNet [29], SimpleNet [10]), models with residual blocks (ResSimpleNet [11]), and encoder-decoder architectures (UNet [33]). Table 1 shows the number of operations involved in running inference for each model and the amount of weight memory used inside the accelerator to give an indication of the size and complexity of each model. Accuracy and the peak activation memory were also included. Interestingly, FaceIDNet has the highest peak activation memory, even though it is smaller than UNet. This can be explained by its architecture, between layer 0 and layers 1 and 2 of FaceIDNet, the input size rapidly expands to be larger than any layer input in any of the models. In these 2 layers 40% of the model's hardware operations occur. All the models used were platform-specific, created using Maxim's Pytorch-based ai8x libraries that take into account the limitations of the MAX78000.

Execution and Measurements: There are a number of optimization parameters that could affect the performance and resource utilization of models executed on neural accelerators (e.g., voltage/frequency scaling of the HW, model design and compression, etc.). Often the tuning of these parameters requires finding a trade-off between memory/latency and accuracy (e.g., the bit width used to quantize the model's weights). As an initial characterization of the MAX78000 platform where we want to explore its performance for various potential applications, we use the default compilation and optimization parameters provided by Maxim for each model. This means that all models are executed with weights quantized at different bit widths ranging between 2bit and 8bit to ensure they fit in the accelerator memory. All MCU compiler optimizations are also disabled. During the execution of the models only the ARM core and the CNN accelerator are active and running at their maximum frequency (100MHz and 50MHz respectively), while the RISC-V is disabled. We are keeping the ARM core active during inference resulting in slightly higher power consumption (MCU + CNN accelerator). However, it represents a more realistic scenario where the MCU is performing other tasks while the accelerator runs a model (e.g., preparing the next input, interfacing with peripherals, or handling user input). Since we are interested in benchmarking only the CNN accelerator component, in our measurements we do not consider all the operations involved in a typical sensing pipeline such as sensor reading, data transmission, and data management.

For each model we measure the latency necessary to set up the accelerator, to load weights and data and for the inference using an onboard timer with μs resolution. For the same operations, power is measured with the Monsoon Power Monitor. Weights memory and feature map peak memory consumption are recorded after each model has been synthesized.

3.2 Memory Footprint

In DNN inference, off-chip memory access costs orders of magnitude more energy and execution time than on-chip memory access and inference itself [35]. Therefore, it is important to keep all weights and feature maps on-chip to gain the efficiency of hardware accelerator. As shown in Table 1, all five models are below the on-chip memory limits for both weight and feature maps. This is thanks to the fact that the accelerator supports different bit widths for weights and feature maps. The model designer can use less than 8bits to represent the weights and can also mix different bit widths within the same model. For example, the SimpleNet model uses 2bit integer weights on most layers, except for a few layers where 8bit and 4bit are used. This offers great flexibility in deploying even large models on the accelerator albeit with a potential accuracy degradation. The same flexibility is not yet available in popular software frameworks to run DNN models on microcontrollers, such as Tensorflow Lite for Microcontrollers [36], which only supports 8bit integer kernel weights and 32bit integer bias weights. We wanted to compare the performance difference between running models on the ARM M4 Core and on the dedicated accelerator. However, due to this TF-Lite Micro limitation, only the ConvNet5 can be deployed

[2]https://www.maximintegrated.com/en/products/microcontrollers/MAX78000EVKIT.html
[3]https://www.msoon.com/high-voltage-power-monitor

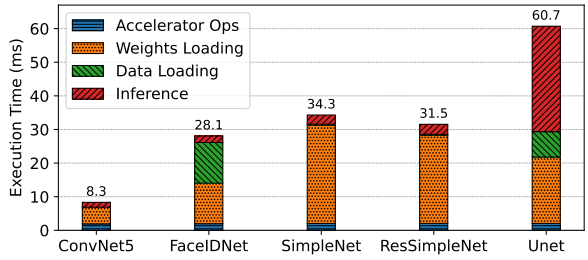

Figure 3: Latency of the five models with a breakdown of the four main operations involved in executing an inference.

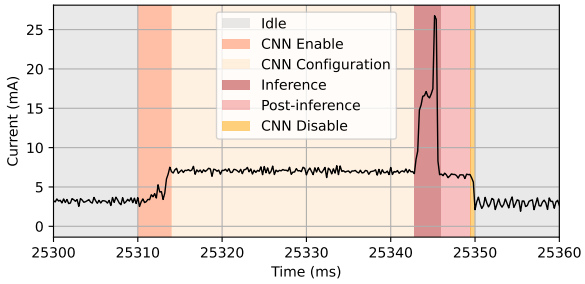

Figure 4: Current Profile of SimpleNet.

on the ARM Core of the MAX78000. The other models exceed the flash and RAM memory available to the MCU.

3.3 Execution Time

Inference execution time is a crucial metric for small systems that need to react promptly to new input data. Executing an inference on the accelerator involves different operations including setting it up, loading weights and data, executing the model, and offloading any result to the MCU. Figure 3 shows the execution time of each model divided into the four main operations.

Accelerator Operations: They configure and control the CNN peripheral: Enable, Initialize, Configure, Start, and Stop. With the exception of CNN Configure these instructions have consistent execution time across all the models. For instance: CNN Initialize is consistently 1815μs, while CNN Start and Stop are always 14μs. CNN Configure sets the model architecture parameters for the four processor groups, taking from 52μs for ConvNet5 to 190μs for UNet. Larger models with more parameters take longer to configure.

Weight Loading: This loads the kernel weights and the bias if used. The weight loading latency was found to increase proportionally to the weight memory size. For example, ConvNet5 uses only 71.2Kb of weight memory, taking 4.926ms to load. SimpleNet and ResSimpleNet use 381.8Kb of weight memory, over 5 times more, and the most of all the models. Their loading times are 29.24ms and 26.21ms. The 5-fold increase in weight memory size caused a 5-fold increase in the loading latency. From the 3ms difference between SimpleNet and ResSimpleNet we can also infer that model architecture has a small effect on the weight loading latency.

Data Loading: The data loading latency varies between the models according to the input size. For instance, UNet has an input size of 48×48×48, taking 7.54ms. SimpleNet and ResSimpleNet have smaller inputs of 32×32×3 taking 0.28ms. The data is loaded almost 27 times faster because the input size 36 times smaller. ConvNet5 is even faster, taking 55μs for its input of 28×28×3. This shows that larger input data samples take longer to load. FaceIDNet is an exception. Although having an input half the size of UNet, 120×160×3, it takes the longest to load at 12.1ms. The large input resolution needs to be streamed in using FIFO registers. This increases latency as data is being fetched incrementally, not all at once.

Inference: This operation is the inference of the CNN, it includes the computation of each layer's output in sequence, data unloading and softmax operations. Softmax is implemented in software on the MCU, not on the hardware accelerator. However since it operates on small data the overhead introduced by the unloading of the last layer's data from the accelerator and the computation of the softmax is very limited (in the order of 10μs, it varies according to the number of data outputs).

We observed that the inference latency increases with model size: from 1.44ms for ConvNet5, to 2.74ms and 2.99ms for SimpleNet and ResSimpleNet. Doubling the number of hardware operations effectively doubled the latency, see Table 1. FaceIDNet had far more hardware operations but an inference latency of 1.93ms. This can be explained by its use of FIFO registers to fetch input data incrementally, which is initialized and occurs during the inference. The data loading and inference happens simultaneously so their latency values should be summed, giving 14ms. UNet had an inference latency of 31.4ms, it is the largest model with the most hardware operations, and the largest data input.

We also compared the execution time of models running on the accelerator with those running on just the MCU core. As mentioned in the previous section, only ConvNet5 is sufficiently small to fit in the memory dedicated to the ARM core using TF-Lite Micro. When clocking the ARM core at the maximum frequency (100MHz) this model runs in 10.1s against the 8.3ms when running on the accelerator. The speedup is thanks to the parallel execution of the convolutional operations within the accelerator.

Key takeaways: The MAX78000 accelerator is specifically optimized for parallel convolution operations and models that use these operations extensively. Many device configuration operations never changed or varied by only a few milliseconds, leaving little room to optimize. It was clear that even the inference latency represented a limited portion of the end-to-end execution latency of the platform. The main bottlenecks we identified concerned memory access to load weights and input data rather than operation execution, see Figure 3. To reduce the weight-loading latency the weight memory size must be decreased, either by using small kernels or by reducing the size and complexity of the model (via compression, pruning, and quantization). The MAX78000 aids in this, allowing integer kernel weights of 8, 4, 2, and 1 bit. To reduce data loading latency, efforts to pre-process data to reduce input size must be considered.

3.4 Energy Consumption

Energy consumption is a critical metric when considering this kind of platform, which is described as capable of running DNN's using μJ's worth of energy. To characterize its energy consumption, we measured its current draw while running each model.

Current Profile: Figure 4 shows the current profile when executing SimpleNet. The effect of CNN operations on the current draw can be observed. For instance, when the accelerator is enabled, the

Table 2: Average power draw and energy consumed during an inference (including weights and data loading).

Model	Average Power (mW)	Energy (μJ)
ConvNet5	26.00	215.80
FaceIDNet	32.07	901.17
SimpleNet	25.34	869.16
ResSimpleNet	25.00	787.50
UNet	56.44	3425.91

current draw doubles, from idling around 3.5mA to 7.5mA. The next section, CNN Configuration, covers all operations that prepare for an inference including: configuration, weight-loading, and data-loading. These operations concern data movement and processor register configuration, and were found to have a negligible impact on the current being drawn. The Inference is the activation of the CNN, which is the most power intensive operation peaking near 27mA, a 4-fold increase. In Post-inference, the current fell back to 7.5mA and then 3.5mA, accounting for the inference finishing, any processing MCU-side, and the peripheral being disabled. All the models produced current profiles similar to Figure 4. This demonstrates that while the model architecture or data input can vary, the MAX78000 will respond consistently in terms of its current draw and energy consumption.

Average Power: Table 2 shows the average power draw and energy consumed during an inference for each model. UNet consumed the most power on average at 56.44mW. This was expected because it is the largest model with the most hardware operations, see Table 1. FaceIDNet was the second with an average of 32.07mW, followed by SimpleNet (25.34mW) and ResSimpleNet (25mW).

The average power draw of ConvNet5 was 26mW, fractionally exceeding SimpleNet and ResSimpleNet. This might seem odd due to the difference in model size but is likely explained by the inference latency of each model. Both SimpleNet and ResSimpleNet have a higher peak power draw than ConvNet5. They also have inference times of 2.74ms and 2.99ms to ConvNet5's 1.44ms. When the power draw of each model is averaged over their the execution time, ConvNet5 having a slightly higher average power draw is not so strange. ResSimpleNet having a lower average power draw than SimpleNet is not unexpected either. ResSimpleNet has the larger execution latency due to its additional layers, averaging the power consumption over this additional time accounts for the difference-especially as they both draw similar amounts of power.

Energy Consumption: The energy consumption of the MAX78000 during an inference was found to vary between models, see Table 2. ConvNet5 consumed the least energy, approximately 0.22mJ. This can be explained by it being the smallest model in terms of hardware operations. It also has the lowest latency. The quicker the inference is completed the less energy that is used. UNet's energy consumption was found to be 3.43mJ, greater than the sum of the energy consumption of the other models. A larger model that runs longer will inevitably consume more energy. ResSimpleNet consumes less energy than SimpleNet, 0.788mJ to 0.869mJ. It is the differences in model architecture causing this. ResSimplnet has more layers and more hardware operations, although they are residual layers which are computationally less intense requiring less energy than

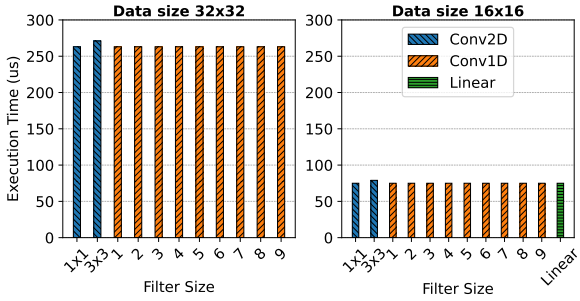

Figure 5: Latency of different operation types and kernel sizes, on two different input data sizes. Linear layers cannot run a 32x32 image, due to insufficient activation memory, hence only compared for the smaller input.

Conv2D layers. This is why the difference is not that much. FaceIDNet has an energy consumption that exceeds models with more layers. As explained previously, it is a compact model with many more hardware operations than, for example, SimpleNet. It also requires the use of FIFO registers adding to the energy overhead.

Key takeaways: Smaller models use less energy with a lower average power consumption because they have less hardware operations and execute faster. So, as with latency, to reduce the average power draw and energy consumption the focus should be on model optimization and data pre-processing. Energy overhead is reduced by using less hardware operations and smaller and less-dense inputs.

4 MICRO BENCHMARKS

We further conduct a micro benchmark to investigate the system implications for maximizing the acceleration benefit. More specifically, we study the impact of two key factors on the latency of the model execution; (a) the model architecture and (b) the selection of convolutional processors.

4.1 Impact of Model Architecture

For this benchmark, we use a two-layer model by default and report the end-to-end execution time of the model. The two layers are both Conv2D with kernel sizes of 3×3 and padding of 1. Throughout the experiment, all the variances of models (operation type, kernel sizes, number of output channels) are made in the second layer.

Operation type and kernel size: Figure 5 shows the execution time for different operation types (Conv2D, Conv1D, and Linear) and kernel sizes (1 to 9 for Conv1D and 1×1 and 3×3 for Conv2D). Surprisingly, for the same data size, there was no difference in the execution time regardless of the operation type and kernel size. This is because the convolution engine in each convolutional processor takes a 2-dimensional convolution of a 3×3 kernel as a basic execution unit. In case when less kernel size is used, the other data is filled with zeros.

Number of input channels: In this experiment, we use a single layer model of convolution 2D with kernel sizes of 3×3 and padding of 1, with 4 output channels, and manipulate the number of input channels of the layer. This was to avoid measuring the data writing to the layer of interest and to solely measure the time for processing

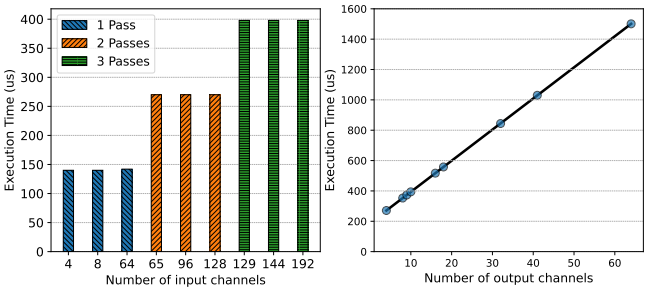

Figure 6: (Left) Execution time under different number of input channels. (Right) Execution time under different number of output channels.

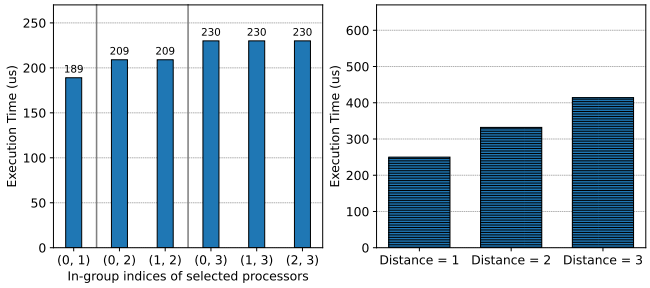

Figure 7: (Left) Performance of intra-group processor selection. (Right) Performance of inter-group processor selection.

a layer. Figure 6 shows the execution time with the different number of input channels. It also shows an interesting result, which is that the execution time does not increase proportionally to the number of input channels. It rather increases in a stepwise manner, i.e., at every 64 channels. This is because the CNN accelerator has 64 convolutional processors and accordingly, the maximum number of operations that can be executed in parallel is also 64. If the number of input channels is greater than 64, the accelerator performs multiple passes over the input data to complete the layer.

Number of output channels: Figure 6 shows the execution time for different numbers of output channels (number of input channels is fixed for this experiment). The result shows that the end-to-end latency increase is proportional to the increase in the number of output channels. This is because the convolutional processors compute one pixel of an output channel at a time. Each additional output channel incurs the computation latency of one output channel.

Key takeaways: The findings imply that MAX78000-specific model tuning needs to be conducted to maximize the acceleration benefit, i.e., by considering the aforementioned results. For example, the number of input/output channels of each layer should be chosen carefully to avoid unnecessary overhead. The number of channels should ideally be a multiple of 64 to ensure good utilization of the 64 processors and good model accuracy (i.e., a higher number of channels typically results in better accuracy since more features are computed) while avoiding latency penalties because the accelerator needs to perform multiple passes on the data (Figure 6).

4.2 Impact of Convolutional Processor Selection

MAX78000 allows developers to select which convolutional processors out of 64 to be used in each layer. We study the impact of processor selection on execution time. For this study, we use the same two-layer model as in §4.1, but set the number of input/output channels in both layers to 2. Similar to the previous experiments, we change the processor selection only in the second layer.

Intra-group selection: Figure 7 shows the execution time when two processors are selected in the same group. Interestingly, the execution time is different depending on the last index out of two processors, but regardless of the first index. We see a latency increase of approximately 20% between selecting processors (0, 1) and (0, 3) which could add significant overhead when applied to larger layers and multiplied for many layers within the model. This is because, when the final output of the current layer is written to the shared activation memory to be used as an input for the next layer, the memory is accessed in a sequential manner in the shared activation memory of a group, i.e., from processor 0 to processor 3. However, memory writing stops when the output is written to all assigned convolutional processors in a group.

Inter-group selection: Figure 7 shows the execution time when two processors are selected in different groups. The distance here denotes the difference between the indexes of two groups of the selected processors. It shows that as the distance grows, the total execution time increases. For example, although the same number of processors is used, the execution time increases from X to Y when the inter-group distance increases from 1 (250 μs) to 3 (414 μs), respectively. This is because memory writing is conducted in a sequential manner, i.e., group by group. That is, rather than writing the data into multiple groups at once, MAX78000 writes the data to the group after the memory access in the previous group is finished. However, we observed that it can directly jump to the shared memory of the group with the smallest index, so only the distance between groups affects the performance.

Note that we also conducted experiments with a different number of input channels and the same trends were observed.

Key takeaways: The results in this subsection imply that it is important to use the minimum number of groups with consecutive processors for runtime performance optimization. While the latency overhead seems small in absolute terms ($\approx 200\mu$s), it adds up quickly for models with many layers and results in significant penalties in terms of inference time, and consequently energy consumption. The optimal processor placement is still an open problem given that automatic tools are not provided. We leave this for future work.

5 CONCLUSION

In this paper, we conducted a variety of benchmark studies to characterize the resource and performance of the ultra-low power DNN accelerator, MAX78000. First, we analyzed the operational latency, power consumption, and memory footprint of five DNN models with various sizes and architecture. Second, we further investigated the system implications in terms of the model architecture and convolutional processor selection in order to maximize the acceleration. Beyond the numbers, our benchmark study further offers meaningful insights for the development of on-device AI systems on ultra-low power, tiny-scale AI accelerators.

REFERENCES

[1] Mattia Antonini, Tran Huy Vu, Chulhong Min, Alessandro Montanari, Akhil Mathur, and Fahim Kawsar. 2019. Resource characterisation of personal-scale sensing models on edge accelerators. In *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*. 49–55.

[2] Arm. 2022. Ethos-U55. https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u55.

[3] Gabriel J Brostow, Julien Fauqueur, and Roberto Cipolla. 2009. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters* 30, 2 (2009), 88–97.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Advances in neural information processing systems (NeurIPS)*. 1877–1901.

[5] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-all: Train one network and specialize it for efficient deployment. In *International conference on learning representations (ICLR)*.

[6] Mitchell Clay, Christos Grecos, Mukul Shirvaikar, and Blake Richey. 2022. Benchmarking the MAX78000 artificial intelligence microcontroller for deep learning applications. In *Real-Time Image Processing and Deep Learning 2022*, Vol. 12102. SPIE, 47–52.

[7] Google. 2022. Edge TPU. https://cloud.google.com/edge-tpu.

[8] Greenwaves. 2022. Ultra low power GAP processors. https://greenwaves-technologies.com/low-power-processor/.

[9] Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International conference on learning representations (ICLR)*.

[10] Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. 2016. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. *arXiv preprint arXiv:1608.06037* (2016).

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Conference on computer vision and pattern recognition (CVPR)*. 770–778.

[12] Maxim Integrated. 2022. MAX78000. https://www.maximintegrated.com/en/products/microcontrollers/MAX78000.html.

[13] Intel. 2022. Intel® Neural Compute Stick 2 (Intel® NCS2). https://www.intel.com/content/www/us/en/developer/tools/neural-compute-stick/overview.html.

[14] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *International symposium on computer architecture (ISCA)*. 1–12.

[15] Martin Kaiser, René Griessl, Nils Kucza, C Haumann, Lennart Tigges, K Mika, Jens Hagemeyer, Florian Porrmann, Ulrich Rückert, Micha vor dem Berge, et al. 2022. VEDLIoT: very efficient deep learning in IoT. In *Design, Automation and Test in Europe Conference (DATE)*. 963–968.

[16] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. (2009).

[17] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *International Conference on Information Processing in Sensor Networks*.

[18] Yann LeCun. 1998. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[20] Hao Liu, Hanting Ye, Jie Yang, and Qing Wang. 2021. Through-Screen Visible Light Sensing Empowered by Embedded Deep Learning. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 478–484.

[21] Wei Lou, Lei Xun, Amin Sabet, Jia Bi, Jonathon Hare, and Geoff V Merrett. 2021. Dynamic-OFA: Runtime DNN architecture switching for performance scaling on heterogeneous embedded platforms. In *Conference on computer vision and pattern recognition workshop (CVPR'W)*. 3110–3118.

[22] Philip Michaels. 2022. A16 Bionic vs A15 Bionic — what it means for the new iPhone 14. https://www.tomsguide.com/news/a16-bionic-vs-a15-bionic.

[23] Chulhong Min, Akhil Mathur, Alessandro Montanari, and Fahim Kawsar. 2022. SensiX: A System for Best-effort Inference of Machine Learning Models in Multi-device Environments. *IEEE Transactions on Mobile Computing* (2022).

[24] Chulhong Min, Alessandro Montanari, Akhil Mathur, and Fahim Kawsar. 2019. A Closer Look at Quality-Aware Runtime Assessment of Sensing Models in Multi-Device Environments. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems (SenSys '19)*.

[25] Alessandro Montanari, Mohammed Alloulah, and Fahim Kawsar. 2019. Degradable Inference for Energy Autonomous Vision Applications. In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp/ISWC '19)*.

[26] Alessandro Montanari, Manuja Sharma, Dainius Jenkus, Mohammed Alloulah, Lorena Qendro, and Fahim Kawsar. 2020. ePerceptive: Energy Reactive Embedded Intelligence for Batteryless Sensors. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 382–394.

[27] Afshin Niktash. 2021. Developing Power-optimized Applications on the MAX78000. https://www.maximintegrated.com/en/design/technical-documents/app-notes/7/7417.html.

[28] Nvidia. 2022. Advanced AI Embedded Systems. https://www.nvidia.com/en-gb/autonomous-machines/embedded-systems/.

[29] Erman Okman and Gorkem Ulkar. 2020. Face Identification using MAX78000. https://www.maximintegrated.com/en/design/technical-documents/app-notes/7/7364.html.

[30] Hishan Parry, Lei Xun, Amin Sabet, Jia Bi, Jonathon Hare, and Geoff V Merrett. 2021. Dynamic transformer for efficient machine translation on embedded devices. In *ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*.

[31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems (NeurIPS)*.

[32] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. 2021. AI accelerator survey and trends. In *High Performance Extreme Computing Conference (HPEC)*.

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.

[34] Mehdi Safarpour, Tommy Z Deng, John Massingham, Lei Xun, Mohammad Sabokrou, and Olli Silvén. 2021. Low-Voltage Energy Efficient Neural Inference by Leveraging Fault Detection Techniques. In *IEEE Nordic Circuits and Systems Conference (NorCAS)*.

[35] Vivienne Sze, Yu-Hsin Chen, Joel Emer, Amr Suleiman, and Zhengdong Zhang. 2017. Hardware for machine learning: Challenges and opportunities. In *Custom Integrated Circuits Conference (CICC)*.

[36] TensorFlow.org. 2022. TensorFlow Lite for Microcontrollers. https://www.tensorflow.org/lite/microcontrollers.

[37] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. 2019. Benchmarking TPU, GPU, and CPU platforms for deep learning. *arXiv preprint arXiv:1907.10701* (2019).

[38] Lei Xun, Long Tran-Thanh, Bashir M Al-Hashimi, and Geoff V Merrett. 2019. Incremental training and group convolution pruning for runtime DNN performance scaling on heterogeneous embedded platforms. In *ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*.

[39] Lei Xun, Long Tran-Thanh, Bashir M Al-Hashimi, and Geoff V Merrett. 2020. Optimising resource management for embedded machine learning. In *Design, Automation and Test in Europe Conference (DATE)*.