Energy Profiling of DNN Accelerators

Matthias Wess, Dominik Dallinger, Daniel Schnöll, Matthias Bittner,
Maximilian Götzinger and Axel Jantsch
Christian Doppler Laboratory for Embedded Machine Learning, Institute of Computer Technology
TU Wien, 1040 Vienna, Austria
{firstname}.{lastname}@tuwien.ac.at

***Abstract*—This paper introduces a novel methodology for assessing the energy efficiency of neural network accelerators at both layer and network granularity. The approach involves extracting per-layer timing reports from recorded power profiles. The power and energy consumption of three prominent neural network accelerators, namely the *Intel Neural Compute Stick 2*, the *Coral Edge TPU*, and the *NXP i.MX8M Plus* is evaluated for three different Deep Neural Networks (DNNs) using this method.**

The study investigates the relationship between decreasing sampling frequencies and the average error, as well as the detailed energy consumption of individual DNN layers and layer types. The findings reveal that latency outperforms the number of operations per layer as a predictor for both overall and dynamic energy, with errors of $10\,\%$ and $100\,\%$ respectively.

The main conclusions are: a sampling frequency of $200\,\mathrm{kHz}$ is necessary to achieve an average error of $5\,\%$; the number of operations is an inadequate predictor of energy consumption; and specific hardware settings significantly influence power and energy consumption, emphasizing the need for their consideration in estimation.

***Index Terms*—Power analysis, Deep Neural Networks, Hardware accelerators**

I. Introduction

Along with the increasing usage of Machine Learning (ML) for solving complex tasks such as computer vision, there has been plenty of research on hardware architectures to execute such algorithms at reduced power and energy budgets. These hardware accelerators reduce the overall power consumption and allow integration of the data processing into the edge devices. Specifically, the inference of ML algorithms on embedded devices has proven efficient in terms of performance and energy. Among several one-board solutions, hardware vendors also provide external USB and PCI-based inference accelerators to support the processing system for ML workloads [1]–[3].

Assessing the power and energy requirements of the various embedded devices is crucial for understanding their performance and optimizing their use in various applications. Obtaining precise measurements that reveal the energy consumption of individual DNN layers is challenging, primarily due to their short execution time. This task becomes even more difficult in the presence of other power consumers in connected electronics, which can obscure the relevant figures.

This work was supported in part by the Austrian Federal Ministry for Digital and Economic Affairs, in part by the National Foundation for Research, Technology and Development, and in part by the Christian Doppler Research Association.

Our interest is the power consumption of specific DNNs as well as their specific layers to analyze the energy efficiency of different layer types. Moreover, we attempt to understand the accuracy of proxy metrics for estimating power and energy consumption.

This paper makes the following key contributions:

- We perform experiments on the required measurement frequency to achieve accurate single-layer measurements.
- We propose a method to extract layer times from the measured power profiles by iteratively removing the last layer of the DNN.
- We profile DNNs on *Neural Compute Stick 2* (NCS2) and *Coral Edge TPU* (edge TPU) at single-layer granularity, gaining insights about the execution efficiency of different layer types on different devices.

Our primary focus is to investigate the energy efficiency of three distinct DNN accelerator architectures. We achieve this by monitoring power consumption at a high sampling frequency. Furthermore, we develop a methodology that enables us to automatically capture the power behavior during the inference phase of DNNs on these accelerators. This methodology also allows us to extract valuable information concerning the power consumption of individual layers within the Neural Network (NN). We accomplish this by correlating detailed latency reports from the accelerators with the recorded power profiles. In cases where detailed latency reports are unavailable, we employ an iterative approach where we remove the last executed layer of the network. To extract layer-specific power consumption data, we then compare the resulting power profiles. Lastly, we conduct an extensive analysis on the inference options available on the *NCS2*, *edge TPU*, and *NXP i.MX8M Plus Development Kit* (i.MX8M+).

II. Related Work

In response to the growing need to execute machine learning algorithms on embedded hardware platforms, numerous efforts have been made to compare the performance of different hardware platforms. Cantero et al. [4] compare the performance of the *edge TPU Coral Dev Board* and the *Variscite i.MX8M PLUS Board* across five distinct model architectures in various resolution settings. The results revealed that the *Coral Dev Board*, with 4 Tera Operations per Second (TOPS), achieved faster computation than the *i.MX8M Board*, which reaches up to 2.3. However, the *i.MX8M* demonstrates more efficient

resource utilization and closely approaches the performance level of the *Coral Dev Board*.

Several benchmarks have been developed to enable a fair and extensive comparison of compute capabilities [5], [6]. These benchmarks rely on a diverse set of workloads. Notably, the MLCommons consortium[1] offers the MLPerf benchmarks [7] for inference and training across diverse domains such as object detection, medical imaging, speech-to-text, and natural language processing. These benchmarks are categorized into data center, edge, mobile, and tiny platforms, representing the broad spectrum of hardware used in machine learning.

Highlighting the importance of comparing power consumption for specific hardware platforms on certain benchmark applications is the fact that the MLCommons consortium released the MLPerf Tiny benchmark [6], which also focuses on power consumption. MLPerf Tiny [6] includes several benchmark tasks: Keyword Spotting, Visual Wake Words, Image Classification, and Anomaly Detection. By providing standardized evaluation criteria, researchers can compare and analyze the energy efficiency of different DNN accelerators and architectures. However, one limitation is that the benchmark results are presented in a summarized form, which provides a broad overview but may lack detailed insights. To conduct these evaluations, the EnergyRunner[2] framework is employed.

Furthermore, leveraging the MLPerf benchmark set, Libutti et al. [8] have measured the power consumption and performance of the edge TPU (adopted from [9]) and Intel NCS2 [1]. They explored various inference modes and settings, yet their findings were limited to reporting overall energy consumption per network without providing detailed insights at a more granular level.

Blott et al. [10] measure the latency and power consumption of many devices, including FPGA, GPU, edge TPU, and VLIW processors for inference of diverse DNNs they focus on gaining a better understanding of the design space with regards to pruning and quantization.

Finally, there have been efforts to estimate the energy consumption of hardware platforms execution NNs. Reif et al. present *Precious* [11], an approach for estimating the energy consumption of ML models based on linear and random forest regressors. In particular, their implementation estimates execution times as well as the power draw of Convolutional Neural Networks (CNNs) on embedded accelerator hardware for NNs (i.e., Google Coral edge TPU [2]). However, it is restricted to a few layertypes and limited layer settings. Other more accurate approaches [12], [13] require in-depth knowledge of the accelerator design and target the hardware design space exploration domain.

In contrast to other works which report the lump latency, power, and energy for inference, we aim to gain a deeper understanding of the energy efficiency of single layers to provide the basis for energy-efficient neural architecture search and layer energy estimation. We, therefore, analyze the power consumption at much finer granularity to obtain latency, power, and energy measurements for individual layers. To do so, we develop a methodology to extract layer times based on the gathered power profiles. This granularity resolution allows us to relate latency and energy of specific layer types and draw conclusions about their efficiency of execution on the given architecture.

[1]https://mlcommons.org, accessed: 2023-05-25

[2]https://github.com/eembc/energyrunner, accessed: 2023-05-25

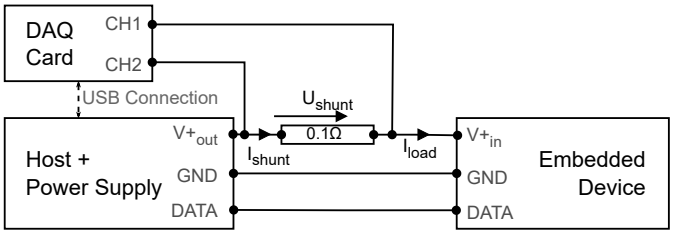


Fig. 1: Power Measurement Circuit. The shunt resistance $R_{\text{shunt}} = 0.1\,\Omega$. U_{shunt} and I_{shunt} are voltage drop and current at the shunt, respectively.

III. Experimental Setup

The measurements are performed on a host computer with an Intel-i7-8565U at 1.80 GHz with 16 GB of RAM and an NCS2 as well as an edge TPU, both connected as ML co-processors connected via USB3.0. For the i.MX8M+, which is also powered via USB 3.0, we measure the power of the entire system. For the measurement setup, we select a USB-1608GX Data Acquisition (DAQ) card due to the good programmability, high sampling rate and flexible measurement ranges. To gather the voltage drop across a shunt resistor was placed on the power line of the Device Under Test (DUT) (see Fig. 1) with a maximum sampling frequency of 500 kHz.

We then compute the power consumption of the DUT as

$$P_{\text{total}} = (U_{\text{supply}} - U_{\text{shunt}}) \cdot (U_{\text{shunt}}/R_{\text{shunt}}), \tag{1}$$

where R_{shunt} is the shunt resistance, and U_{shunt} is the voltage drop at the shunt. Due to the high input resistance, we can neglect the current through the DAQ card and assume $I_{\text{shunt}} = I_{\text{load}}$. Important to note is that the power supply lines of the USB cable from the host to the DUT have been cut, and the DUT is powered exclusively by the dedicated power supply.

A. Intel NCS2

The NCS2 USB 3.0 ML accelerator implements a MYRIAD X Visual Processing Unit (VPU) with 16 Streaming Hybrid Architecture Vector Engine (SHAVE) processors operating at 700 MHz. For the computation of a DNN, the NCS2 provides a maximum nominal performance of 1 TOPS in Floating-Point 16 (FP16) format. Inference on the accelerator is performed via the OpenVino toolkit, supporting up to 4 parallel inference requests (see section V-E).

B. Coral Edge TPU

The Google Coral machine-learning accelerator co-processor is a USB device embedding the Google Edge edge TPU ASIC. The Coral accelerator performs inference operations on TensorFlow Lite[3] models with a peak current of 900 mA at 5 V. The accelerator maintains a computational throughput of 4 TOPS at a computational efficiency of 2 TOPS per Watt [2]. The edge TPU supports two operational modes: the standard mode and the maximum frequency, which are discussed in section V-E.

C. I.MX8M Plus

The i.MX8M+ processor is a heterogeneous multi-core processor developed by NXP. The processor incorporates an embedded Vivante VIP8000 Neural Processing Unit (NPU) that provides 2.3 TOPS of computing power. To optimize the data exchange between the computing units, the NPU shares the high-speed internal memory bus with the Central Processing Units (CPUs). Similar to the edge TPU, the Vivante VIP8000 performs 8-bit Integer (INT8) operations, accelerating the TensorFlow Lite execution.

D. Software

For the measurements of the NCS2, we make use of the OpenVino toolkit [14], which offers many convenient tools for converting, optimizing, benchmarking, and analyzing neural networks. The model optimization and conversion tools support networks generated in popular deep learning frameworks/formats such as PyTorch[4], TensorFlow[5], and ONNX[6] as source workloads. Performance metrics such as latency and throughput of individual layers can be extracted with the help of the API benchmark application.
For the measurements of the edge TPU and the i.MX8M+, the models are converted into TensorFlow Lite format and post-training quantized to INT8 format. The inference on the edge TPU and the i.MX8M+ is driven through the TensorFlow Lite inference engine, offloading the workload to the accelerators via the delegate functionality. Both, the edge TPU and the i.MX8M+ only support fully-quantized 8-bit TensorFlow Lite models. For the edge TPU the models need to be specifically compiled (using the edge TPU Compiler) [15]. For the i.MX8M+ no compilation is necessary prior to the execution. To access the DAQ card, we use the MCC Universal Library[7].

E. The workload

As workload for the measurements, we selected three different DNNs for image classification and object detection, namely MobileNetV2 [16], YOLOv3, and YOLOv3-tiny [17].

[3]https://www.tensorflow.org/lite, accessed: 2023-05-25
[4]https://pytorch.org, accessed: 2023-05-25
[5]https://www.tensorflow.org, accessed: 2023-05-25
[6]https://onnx.ai/, accessed: 2023-05-25
[7]https://github.com/mccdaq/uldaq, accessed: 2023-05-25

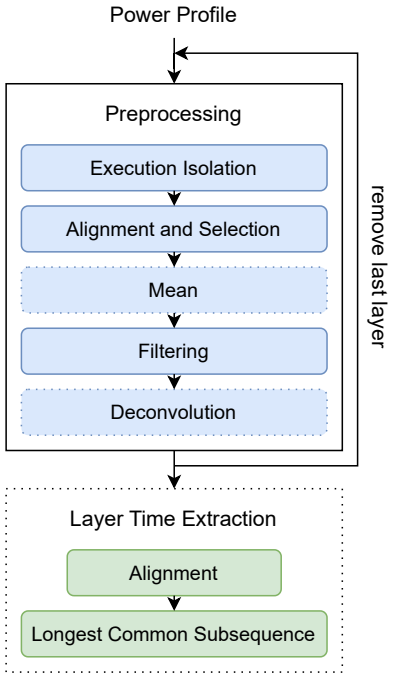


Fig. 2: The signal processing pipeline used for the power profile extraction and layer time extraction

IV. Methodology

This section describes the proposed methodology and signal processing pipeline designed to facilitate precise benchmarking and profiling. We provide a detailed description of the pipeline, which enables the semi-automatic annotation of power and energy consumption for each executed DNN on the hardware platform.

When per-layer timing reports are available on the hardware platforms, the times can be annotated into the power profile. In that case, the major challenge is to identify the start of the first layer. Alternatively, we can retrieve the per-layer timing information by iteratively removing the network layers one by one and comparing the recorded power profiles of the modified and the original network. This approach ensures synchronization between the layer latency annotations and the power profile. Also, some devices operate less efficiently when in profiling mode, which we can avoid. Our method, can generate layer-wise execution time reports for potentially any hardware with a characteristic per-layer power profile.

Fig. 2 depicts the implemented signal processing pipeline. The pipeline consists of two sub-pipelines: pre-processing and layer-profile extraction. For the benchmarking, we also record the power profile of the initialization, the warm-up phase, and multiple iterations of the DNN executed on the target device.

The pre-processing is critical to isolate a single execution power profile from the recorded data. To reach this target, we first detect the single executions by applying a sliding window with the width according to the execution time of the DNN. To ensure the correct cutout of the actual execution

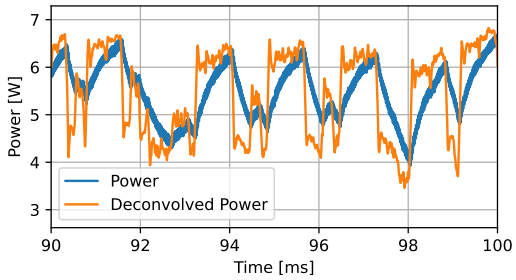


Fig. 3: The recorded power profile for the i.MX8M+ exhibits smoothing effects in comparison to the deconvolved power profile

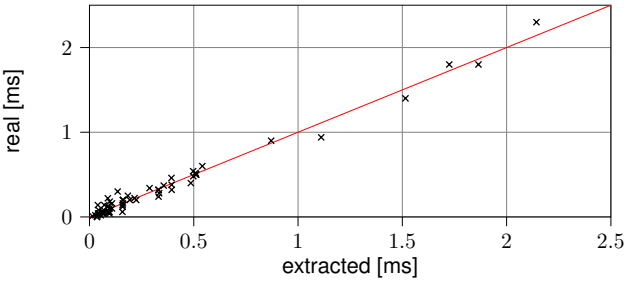


Fig. 4: Comparison of the extracted layer times vs. measured layer times for MobileNetV2 on NCS2

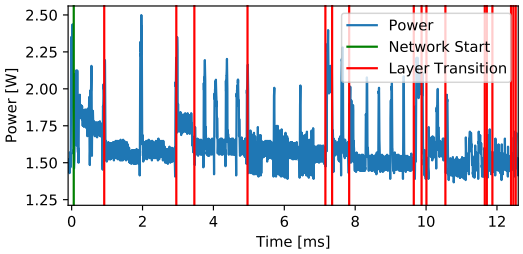


Fig. 5: The power profile of MobileNetV2 executed on NCS2 with annotated layer transitions

profiles, we compute the maxima and minima of the filtered signal and select the detected peaks with similar inter-peak distances as center points for the cutouts. Additional adaptive thresholds and edge detection ensure clean cutouts of the execution profiles. Next, the cutouts are pairwise iteratively aligned by minimizing the Euclidean distance between the cutout signals. We use the isolation forest algorithm [18] to filter out anomalies in the cutouts. In an optional next step, we compute the means of the remaining cutouts to reduce noise on the measured signal. To obtain accurate power profiles of the DNN executed on a target device, we need to address the effect of the capacitors within the power supply circuit. Capacitors act as short-term energy storage devices, smoothing the signal but distorting the actual power consumption in the time domain. to the device during the execution of the DNN. These distortions can introduce measurement inaccuracies, potentially leading to the misattribution of energy consumed by a previous layer to the next layer. To solve this problem, we perform a deconvolution with the impulse response of the normalized capacitor discharge curve. The value of τ of the discharge curve is empirically determined as part of the pre-processing step, and the deconvolution is applied to each recorded power profile. The effect of this deconvolution step is shown in Fig. 3.

For our semi-automated layer time extraction, we iteratively remove the last layer of the network and record the profile of the resulting network. The resulting profile usually exhibits a shorter execution time and possibly visible artifacts in the power profile after the output layer (which can be denoted to reading the output data). To neglect these artifacts, we determine the longest common sub-sequence between the aligned power profiles of the current sub-network and the previously measured networks. Computing the differences between the identified longest common sub-sequences we retrieve the execution times of each layer.

Fig. 4 shows the layer times extracted with our pipeline compared to the reports generated by OpenVino for the NCS2. Based on this method, we achieve an average error of $42\mu s$ for the execution of MobileNetV2 on the NCS2. At an average layer time of $311\mu s$, this means that we have to consider that for short layers, the error of the extracted layer times is rather large. However, since we are mainly interested in the efficiency of the large layers which make up for most of the energy consumption, we proceed with this approach.

To verify the method, we first visualize the recorded measurements to analyze the power profile at 500 kHz sampling frequency. We can now visualize the timestamps of the transitions between the layers. Fig. 5 shows excerpts of such measurements of MobileNetV2 executed with batch size one on the NCS2 in synchronous mode.

Based on our observations, we find that both the NCS2 with $P_{\text{base}} \sim 1.4\,\text{W}$ and the edge TPU with $P_{\text{base}} \sim 1.1\,\text{W}$ exhibit a relatively high base power consumption after the network is loaded in comparison to the dynamic power P_{dyn} which is consumed additionally during the execution of the DNNs. The i.MX8M+ has an even higher idle power consumption with a $P_{\text{base}} \sim 2.9\,\text{W}$. As in this case we are not only measuring the NPU but also the CPU and memory components

Therefore, we denote the total power consumption as

$$P_{\text{total}} = P_{\text{base}} + P_{\text{dyn}} \tag{2}$$

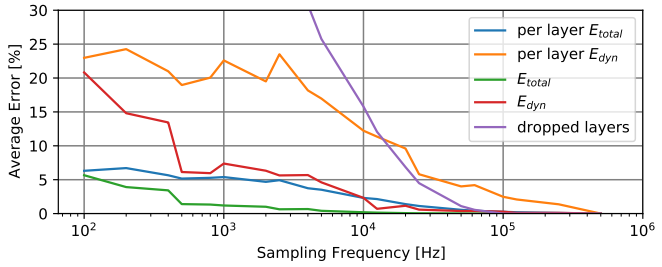
for the further experiments. We approximate the energy consumption of the entire network and the single layers with

$$E \approx t \cdot \overline{P}, \tag{3}$$

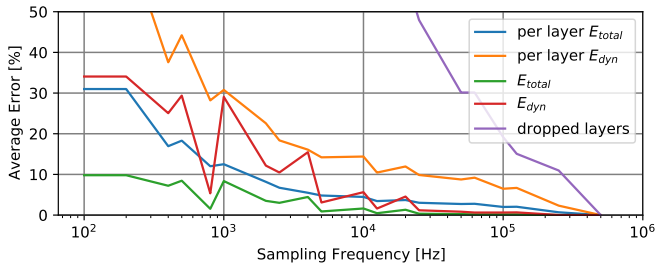
where t is the reported execution time and $\overline{P}$ is the mean power consumption of the executed layers and the entire network. With (2) and (3), we can compute the energies consumed due to base power P_{base} and dynamic power P_{dyn}.

V. Results

This section summarizes the results of applying the power profiling methodology presented in section IV.

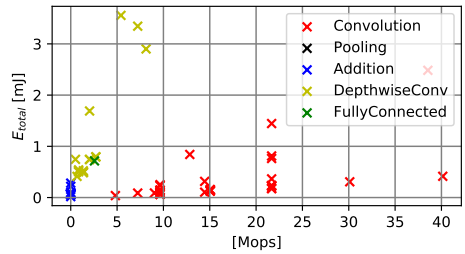


(a) NCS2

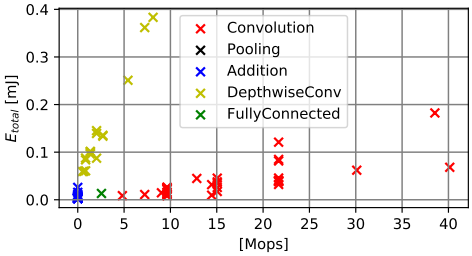


(b) edge TPU

Fig. 6: The error of measured energy and percentage of dropped layers in % for MobileNetV2 on the NCS2 and edge TPU with respect to 500 kHz sampling frequency



(a) NCS2



(b) edge TPU

Fig. 7: Total energy vs. number of operations for MobileNetV2 on the NCS2 and edge TPU

A. Sampling frequency

We first study the influence of the sampling frequency on the resulting errors with regards to the number of profiled layers, the energy consumption per layer, and the energy consumption for the entire network to gain an understanding for the required measurements frequencies for future devices.

Fig. 6a shows the error at a given sampling frequency compared to the results obtained with a sampling frequency of 500 kHz for the NCS2. The error of the total energy E_{total} is below 5 % at sampling rates above 100 Hz because the base power is a relatively major component. The error for the dynamic energy E_{dyn} grows faster with decreasing sampling frequency. The results show that a fine-grained understanding of the per-layer energy consumption requires a 10 kHz sampling frequency or higher for the NCS2. The denoted error per layer considers only the layers actually recorded during measurement. Notably, with decreasing sampling frequency, the number of dropped layers (layers with execution time shorter than the sampling window) quickly grows and amounts to 15 % at 10 kHz sampling frequency.The results for repeating the same measurements on the edge TPU are depicted in Fig. 6b. Compared with the NCS2, the edge TPU executes MobileNetV2 7 × faster, which leads to higher relative errors at the same sampling frequency. As expected the average network and per-layer error scales with the compute performance of the device and the latency of the network layers.

B. Energy versus the number of operations

Comparing the measured energy consumption of the layers with the number of operations, we note that for both accelerators, not all layers are computed with equal energy efficiency.

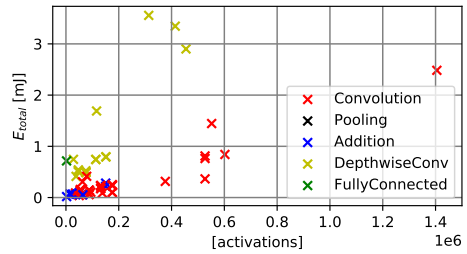
Fig. 7 shows the energy consumed for specific numbers of operations of different layers in MobileNetV2. Naturally, more operations lead to higher energy consumption, but the slope differs quite for different layer types. Depth-wise convolution layers (DepthwiseConv) have fewer operations but consume 4-6 times more energy per operation than ordinary convolutions. This behavior is explainable by the lower computational efficiency and the longer layer execution times of the NCS2 as well as the edge TPU computing depth-wise separable convolution layers. Moreover, within a layer type, there are significant differences. The energy efficiency difference between different convolution layers is up to 3× for the edge TPU and 4× for the NCS2. Thus, the number of operations is a poor proxy for estimating energy consumption for both accelerators.

C. Energy versus the number of activations

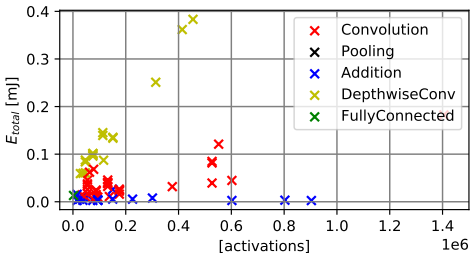
Considering the number of activations (Fig. 8), we see similar patterns for the NCS2 and the edge TPU, as the depth-wise convolution layers are computed with lower energy efficiency. Again, not only between different layer types but also within a layer type, there are significant differences in terms of energy efficiency per activation.

D. Energy versus latency

Fig. 9a and 9c depict the total energy and the dynamic vs. the runtime for the layers in MobileNetV2 on the NCS2. The total energy consumption correlates well with the runtime



(a) NCS2



(b) edge TPU

Fig. 8: Total energy vs. number of activations (output feature map size) for MobileNetV2 on the NCS2 and edge TPU

per layer (see Fig. 9a and 9b). One of the main reasons for this is the relatively high base power consumption of the NCS2 and the edge TPU compared to the dynamic power. When increasing the number of parallel inference requests for the NCS2 up to 4, the relation between dynamic and base energy shifts more towards dynamic energy. Thus, we also investigate the correlation between the dynamic energy per layer and the runtime. We find that the correlation is still fairly good; a layer that needs more time also needs more dynamic energy. However, the data points in Fig. 9c and 9d are not lined up on a straight line with a constant slope, which means the correlation is imperfect. Some layers consume up to 2 times more dynamic energy per time unit (more dynamic power) than others. Interestingly, on the NCS2, depth-wise separable convolution layers tend to consume less power than convolution layers.

Combining this observation with what we see in Fig. 7, depth-wise separable convolution layers are less efficiently executed by the hardware than convolution layers: they have fewer operations and require relatively more time but less power. It seems that they cannot keep the hardware as busy as ordinary convolution layers. One possible explanation is that depth-wise convolution layers are limited by the memory hierarchy rather than the computational capabilities of the NCS2 and the edge TPU. We found that the remaining layer types used in MobileNetV2 generally consume a higher amount of energy per operation than convolutional layers but are only responsible for an almost negligible portion of the total energy consumed per network.

E. Hardware settings

The NCS2 can be operated in two different modes, synchronous and asynchronous. In summary, synchronous means purely sequential execution while asynchronous allows pipelined execution. The *nireq* factor in Table I gives the number of requested parallel inferences. The inference latency slightly increases when pipelining inference requests. In contrast, the throughput, measured in frames per second, improves fairly significantly: between $33\,\%$ and $77\,\%$ when moving from sync to async-2 mode and between $5\,\%$ and $25\,\%$ when moving from async-2 to async-3. No improvement is found for the async-4 mode because the maximal amount of nireq has already been reached at async-3.

A better hardware utilization in the async-2 and async-3 modes, compared to sync-1, is reflected in the increased average power consumption, increasing by $17\,\%$ to $43\,\%$.

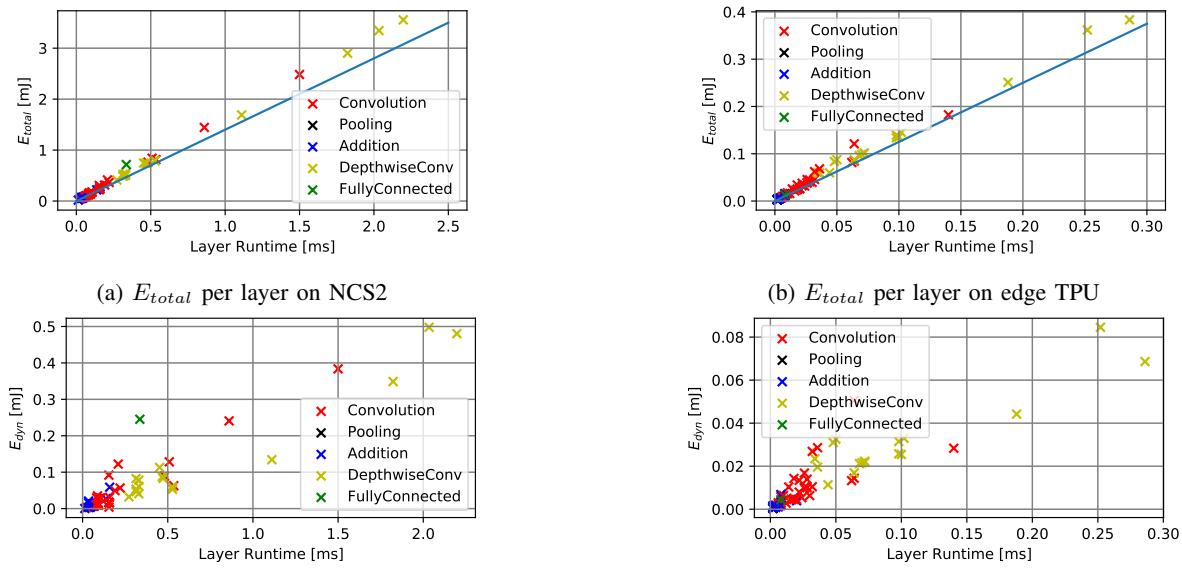
While the async modes increase the hardware utilization and power consumption, the total energy per inference is reduced, which reflects an overall better usage of hardware and energy resources. Comparing the energy values for the base and dynamic energy for the different execution modes, we can see that the total energy reduction per inference comes mostly from amortizing the high base energy over several inferences. For example, the dynamic energy per inference for executing YOLOv3 with 4 inference requests is almost twice as high as the base energy.

In contrast to the NCS2, the edge TPU does not allow multiple parallel inference requests but can be operated at two different clock frequency settings. Table I lists the frequency settings standard (std) and maximum (max).

The three test networks gain a speedup of between $10\,\%$ and $50\,\%$ when switching from std to max frequency mode. Due to the lower execution time with maximum frequency the base energy decreases while the dynamic energy increases for all three networks. On the other hand, with maximum frequency, the total energy decreases for all three networks while the mean power consumption increases by up to $15\,\%$.

Depending on the selected hardware settings and the network architecture, the edge TPU is 2-10 times more efficient regarding total energy consumed per parameter and operation than the NCS2. This difference can be explained by the difference in data types the two operators are operating on (INT8 and FP16).

Lastly, the i.MX8M+ NPU also only supports INT8 execution. For comparison we also provide the numbers for execution on the CPU. As the measurements for the i.MX8M+ also include the power consumption of the CPU and other components, the base power consumption is significantly higher for the i.MX8M+ than for the edge TPU and the NCS2. Interestingly, for YOLOv3 and YOLOv3-tiny the i.MX8M+ outperforms the edge TPU in terms of latency. However, even though the execution time is smaller E_{total} and E_{dyn} are still higher than for the edge TPU. Additionally, we can see that for MobileNetV2 the edge TPU outperforms the i.MX8M+. We assume that this is due to the fact that the edge TPU is a better fit for depthwise convolution.



(a) E_{total} per layer on NCS2

(b) E_{total} per layer on edge TPU

(c) E_{dyn} per layer on NCS2

(d) E_{dyn} per layer on edge TPU

Fig. 9: Total and dynamic energy for the layers of MobileNetV2; the blue line denotes the base energy consumed by the edge TPU during the layer runtime

VI. Conclusion

We presented a methodology on how to perform power measurements an how to extract additional information of the power profiles. The method allows to gain additional insights with reduced additional development effort per platform. However, the method has some limitations regarding accuracy of the extracted data. It has to be considered, that the shorter the execution time of a NN the less accurate this method works. Therefore, we expect difficulties of applying this method for too powerful platforms or for small networks.

From our experiments with three networks and the accelerator platforms, we draw the following main conclusions, some expected and others less conspicuous.

- To obtain an average power measurement error lower than 15 %, a sampling frequency of 10 kHz or higher is required. For an error below 5 %, a sampling frequency of 200 kHz is recommended for the two USB accelerators.
- The number of operations is a poor predictor of energy consumption. Latency is a much better predictor with, in our experiments, an expected error margin of around 10 %. However, the correlation between latency and energy usage for individual layers can vary by up to 2× for dynamic energy, which means dynamic energy estimation based only on latency may be off by up to 100 %. Which means that for correctly predicting dynamic energy, additional factors would have to be considered.
- Settings of the hardware can significantly influence latency, throughput, and energy consumption. Specifically, the async-3 mode on NCS2 improves throughput by up to 100 % and energy per inference by up to 35 %, at a cost of increasing latency by up to 48 %.
- On the NCS2, increasing the number of parallel inference requests shifts the relation between base power and dynamic power towards the latter. As a further consequence, this aggravates the task of estimating the total energy due to the non-linear nature of the relationship between layer runtime and dynamic energy.
- For both USB accelerators, we can adjust the relationship between power consumption and energy per image through either the number of parallel inference requests or the clock frequency. In both cases, the power consumption increases when switching to the higher throughput modes, but with the overall result of a lower energy per image.

With our study, we wanted to obtain a better understanding of power and energy usage in specific state-of-the-art networks with diverse layer combinations on a given hardware platform. We conclude that latency can be used as a first-order estimate for power and energy consumption for a network and individual layers. However, if a more detailed understanding of a network and its layers is required for system energy budgeting or network optimizations, more detailed and precise measurements are required because the dependencies and influences are often non-linear and non-intuitive. In addition to providing insight into what can be derived from accurate power profiles of DNN accelerators, the understanding gained in this study can also be applied to power consumption estimation and energy-aware neural architecture search.

References

[1] Intel. Neural compute stick documentation. https://www.intel.com/content/www/us/en/developer/articles/guide/get-started-with-neural-compute-stick.html, 2019. Accessed: 2023-05-25. 1, 2

TABLE I: Speedup comparison of different Networks. *nireq* denotes the number of parallel inference requests. *Freq* denotes the frequency setting for the edge TPU.

HW	Network	*nireq*	$F_{\text{thr}}(\text{fps})$	$T_{\text{lat}}(\text{ms})$	P (mW)	$E_{\text{total}}(\text{mJ})$	$E_{\text{base}}(\text{mJ})$	$E_{\text{dyn}}(\text{mJ})$	$E/Gop(\text{mJ})$	$E/Mpar(\text{mJ})$
NCS2 w.o. Host FP16	YOLOv3-tiny	1	21.2	41	2165	101.93	65.91	36.02	18.32	11.52
		2	35.3	52	2670	75.55	39.61	35.94	13.58	8.54
	5.6 Gop	3	43.1	46	2995	69.42	32.45	36.97	12.47	7.85
	8.8 Mpar	4	43.1	44	2954	68.54	32.48	36.06	12.32	7.75
	YOLOv3	1	2.6	363	2505	960.92	537.04	423.88	14.69	15.61
		2	4.4	400	3413	769.61	315.69	453.92	11.76	12.50
	65.8 Gop	3	4.7	425	3615	764.89	296.22	468.67	11.69	12.42
	61.6 Mpar	4	4.9	390	3604	742.50	288.43	454.07	11.35	12.06
	MobileNetV2	1	49.3	21	1806	36.60	28.37	8.23	60.84	10.55
		2	87.2	23	2118	24.29	16.06	8.23	40.38	7.00
	0.6 Gop	3	90.4	31	2164	23.95	15.49	8.46	39.81	6.90
	3.4 Mpar	4	92.4	53	2162	23.39	15.15	8.24	38.88	6.74

HW	Network	*Freq*	$F_{\text{thr}}(\text{fps})$	$T_{\text{lat}}(\text{ms})$	P (mW)	$E_{\text{total}}(\text{mJ})$	$E_{\text{base}}(\text{mJ})$	$E_{\text{dyn}}(\text{mJ})$	$E/Gop(\text{mJ})$	$E/Mpar(\text{mJ})$
Edge TPU w.o. Host INT8	YOLOv3-tiny	std	46.3	22.3	1407	30.40	22.28	8.12	5.46	3.44
		max	51.0	19.6	1528	29.95	20.21	9.73	5.38	3.39
	YOLOv3	std	6.3	158.3	1519	240.50	163.27	77.23	3.68	3.91
		max	7.0	142.0	1657	235.36	147.29	88.06	3.60	3.82
	MobileNetV2	std	331.3	3.0	1422	4.29	3.11	1.18	7.13	1.24
		max	512.3	1.9	1658	3.23	2.02	1.21	5.37	0.93

HW	Network	*Freq*	$F_{\text{thr}}(\text{fps})$	$T_{\text{lat}}(\text{ms})$	P (mW)	$E_{\text{total}}(\text{mJ})$	$E_{\text{base}}(\text{mJ})$	$E_{\text{dyn}}(\text{mJ})$	$E/Gop(\text{mJ})$	$E/Mpar(\text{mJ})$
i.MX8M+ INT8	YOLOv3-tiny	npu	102.8	9.7	4398	42.78	26.32	16.47	7.64	4.84
		cpu	1.3	758.5	3917	2971.33	2421.60	549.73	534.41	335.93
	YOLOv3	npu	9.5	105.0	4788	502.51	289.93	212.58	7.64	8.16
		cpu	0.1	7706.6	3434	26462.35	20632.56	5829.79	402.16	429.58
	MobileNetV2	npu	97.1	10.3	3807	39.21	28.11	11.11	65.35	11.53
		cpu	8.4	119.7	3375	403.97	327.90	76.07	673.28	118.81

[2] Coral. Usb accelerator datasheet. https://coral.ai/docs/accelerator/datasheet/, 2021. Accessed: 2023-05-25. 1, 2, 3

[3] NXP. i.mx 8m plus documentation. https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-8-applications-processors/i-mx-8m-plus-arm-cortex-a53-machine-learning-vision-multimedia-and-industrial-iot:IMX8MPLUS, 2023. Accessed: 2023-05-25. 1

[4] David Cantero, Iker Esnaola-Gonzalez, José Miguel-Alonso, and Ekaitz Jauregi. Benchmarking object detection deep learning models in embedded devices. *Sensors*, 22(11), 2022. 1

[5] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Christopher Ré, and Matei Zaharia. Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark. *ACM SIGOPS Oper. Syst. Rev.*, 53(1), 2019. 2

[6] Colby R. Banbury et al. Mlperf tiny benchmark. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks*, 2021. 2

[7] V. J. Reddi et al. Mlperf inference benchmark. In *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020. 2

[8] Leandro Libutti, F. Igual, L. Piñuel, Laura C. De Giusti, and M. Naiouf. Benchmarking performance and power of USB accelerators for inference with MLPerf. In *1st Workshop on Accelerated Machine Learning (AccML)*, 2020. 2

[9] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017. 2

[10] Michaela Blott et al. Evaluation of optimized cnns on heterogeneous accelerators using a novel benchmarking approach. *IEEE Trans. Computers*, 70(10), 2021. 2

[11] Stefan Reif, Benedict Herzog, Judith Hemp, Timo Hönig, and Wolfgang Schröder-Preikschat. Precious: Resource-demand estimation for embedded neural network accelerators. In *First International Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware*, 2020. 2

[12] Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD*, 2019. 2

[13] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David M. Brooks. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA*, 2014. 2

[14] Intel. OpenVINO Toolkit. https://software.intel.com/en-us/openvino-toolkit, 2021. 3

[15] Coral. Tensorflow models on the edge tpu. https://coral.ai/docs/edgetpu/models-intro/, 2021. 3

[16] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2018. 3

[17] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. 3

[18] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM)*, 2008. 4