Are Mobile DNN Accelerators Accelerating DNNs?

Qingqing Cao*, Alexandru E. Irimiea‡, Mohamed Abdelfattah◇,
Aruna Balasubramanian*, Nicholas D. Lane†◇
qicao@cs.stonybrook.edu,{alexandru.irim,moh.s.abdelfattah}@gmail.com
arunab@cs.stonybrook.edu,ndl32@cam.ac.uk
*Stony Brook University ‡University of Oxford †University of Cambridge ◇Samsung AI

ABSTRACT

Deep neural networks (DNNs) are running on many mobile and embedded devices with the goal of energy efficiency and highest possible performance. However, DNN workloads are getting more computationally intensive, and simultaneously their deployment is ever-increasing. This has led to the creation of many purpose-built low-power neural accelerators to replace or augment traditional mobile CPUs and GPUs. In this work, we provide an in-depth study of one set of commercially-available mobile accelerators, the Intel Neural Compute Sticks (NCS). We perform a systematic measurement study of the latency and energy of this accelerator under a variety of DNNs including convolutional neural networks (CNNs) for vision tasks and attention-based Transformer models for NLP tasks. We compare to the mobile processors (CPU, GPU, and DSP) on a smartphone and a mobile board. Our study shows commercial mobile accelerators like NCS are not ready yet to provide the performance as claimed. We also point out directions in optimizing the model architectures to better suit these accelerators.

ACM Reference Format:
Qingqing Cao*, Alexandru E. Irimiea‡, Mohamed Abdelfattah◇,, Aruna Balasubramanian*, Nicholas D. Lane†◇ . 2021. Are Mobile DNN Accelerators Accelerating DNNs?. In *5th International Workshop on Embedded and Mobile Deep Learning (EMDL'21), June 25, 2021, Virtual, WI, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3469116.3470011

1 INTRODUCTION

The past four years has seen sizable strides in the inference-time efficiency of deep learning, and at a cost of few percentage points in accuracy the execution of the best learning algorithms for various tasks (e.g., speech, vision, language) are increasingly becoming feasible for phone, wearable and embedded platforms [25]. This capability provides a range of benefits for network edge devices. Not only are they able to use the state-of-the-art models for processing data such as images, audio, and text, instead of weaker, less complex models – they are also able to do so when network connectivity is poor or when the cloud is unavailable. On-device processing provides privacy to users where sensitive data such as spoken words, or image of faces leaves their personal device. Such benefits have pushed the industry towards adoption of on-device deep learning. For instance, in Android devices and the iPhone, on-device deep learning is used to detect if a user is in a car, is walking or running [2] and is used to detect special keywords (e.g., *"Hey Siri"*) [3]. However, due to limited resources many other types of deep models, and resulting applications, remain out of reach.

Research advances within efficient inference for deep learning has moved in two major directions, one software-centric and the other related to processor hardware. Examples of software-based innovation include improvements within the learning algorithms themselves, such as more efficient model architectures, or in software system solutions that improve performance through a better understanding of the workload [12, 14]. On the hardware side, processor designs that are purpose-built for the execution of neural architectures have proliferated [11, 33]. It is common for such accelerators to offer substantial performance gains over mainstream more general-purpose processors like GPUs, CPUs and DSPs. However, this is often achieved by making strong assumptions about the neural architecture composition to be executed; this in turn makes offering such performance broadly across a variety of architectures (especially very large and deep models) problematic.

An important question for on-device deep learning will be to understand the role of purpose-built hardware accelerators. Will accelerators make all neural network models *"cheap"* to run? Are the resource bottlenecks in deep learning inference on mainstream CPUs disappear tomorrow as accelerators become more common? Core questions of this type have been inaccessible until only recently due to the lack of off-the-shelf accelerator hardware. In comparison to the hundreds of proposed accelerator designs and approaches, there had been almost no open accessible accelerator processors suitable for comparing again other processor architecture types or mature enough to run a wide range of deep models. This has now changed with offerings being made from companies that include Nvidia [41], Huawei [19] and Intel [29].

In this paper, we will begin to address these open questions by presenting a performance characterization of how DNNs perform under one of the very few commercially[1] available hardware accelerators purpose-built to execute deep models efficiently: the Intel Neural Compute Stick (NCS) Version 1 and Version 2 [29]. The contributions of this work are as follows:

- **DNN performance under commodity accelerators.** Detailed experimental results highlighting the performance of various

[1]EdgeTPU, for example, can only support limited types of CNNs

EMDL'21 , June 25, 2021, Virtual, WI, USA

ACM ISBN 978-1-4503-8597-8/21/06…$15.00
https://doi.org/10.1145/3469116.3470011

popular DNNs under the NCS versions 1 and 2 (abbreviated NCS1 and NCS2). These results highlight that these mobile accelerators are evolving and newer version provides lower inference latency and less energy footprint.

- **Comparisons to processor alternatives.** As mobile and embedded hardware increasingly offers a variety of processor types, we compare the performance under deep model workloads of the NCS1 and NCS2 to the GPU, CPU and DSP from a representative Qualcomm SoC and Nvidia mobile board. The empirical comparisons show that mobile accelerators like NCS devices are not a competitive as mobile GPUs and DSPs for the studied CNNs. Moreover, NCS accelerators are not even close to the CPU performance when running NLP models that are becoming more prevalent in multi-modal vision applications.
- **NCS-Aware CNN Optimization.** Given that NCS devices are not ready to accelerate the DNNs, we explore the design space of DNN model architectures and adapt DNNs to be more accelerator-friendly. We use one of our observations to optimize a CNN specifically for NCS2 as a proof-of-concept. We show that by adjusting filter depths of InceptionV3, we can achieve a 2× speedup with ~1% accuracy loss. We show that optimizing a CNN for a specific device in that way can lead to large efficiency gains.

2 MOBILE DNN HARDWARE

Mobile CPUs, GPUs and DSPs. Mobile CPUs such as ARM processors have single instruction multiple data (SIMD) capabilities to parallelize compute intensive operations. Most deep learning frameworks on mobile, such as TensorFlow lite (TFLite) or Qualcomm Snapdragon Neural Processing Engine (SNPE) use ARM's SIMD instruction set called NEON. This allows the explicit use of SIMD features in ARM's processors thereby accelerating DNN inference. However, mobile CPUs have limited parallelization for compute intensive applications – they are natively designed for more general tasks and have complex control logic and lower compute density, which are mainly optimized for serial, control-heavy operations. GPUs are designed to accelerate graphics applications thus having stronger SIMD capabilities built for massively parallel workloads. Deep learning execution frameworks using GPUs typically utilize many GPU shader cores (usually range from tens to hundreds) to parallelize the mathematical computations. GPUs are built to have high compute density and high computations per memory access, which are optimized for parallel operations. Mobile DSPs (e.g. Hexagon DSP) use SIMD extentions, which enhance the original SIMD by introducing vector execution units. Another feature of DSPs is that they often support integer based operations (8 or 16-bit) to further reduce compute clock cycles and energy footprints.

Neural Accelerators: Myriad VPUs. Intel Movidius Neural Compute Stick (NCS) [29] is one of the first commercially available accelerators for mobile, wearable, and embedded systems. It is powered by a low-power, high-performance Vision Processing Unit (VPU) designed to handle a range of applications such as Deep Neural Network-based classification, object tracking, indoor navigation and 3D vision applications. At the heart of the NCS is the Myriad-2 VPU. Myriad-2 provides up to 1TOPs of performance at 180 MHz within a nominal 1W of power consumption. The Myriad-2 contains twelve 128-bit vector processing cores called SHAVE which compute most of the neural network load [9]. There are 2-MB of on-chip memory that can be configured to accommodate different instruction and data mixes depending on the workloadAdditionally, myriad-2 supports stream processingsimilar to GPUs. NCS2 [22] has a Myriad-X VPU that runs at 600 MHz (compared to 180 MHz in Myriad-2) and increases the number of SHAVE cores to 16 (up from 12 in Myriad-2). Additionally, it includes a new "neural compute engine" dedicated for deep learning workloads specifically. This suggests that the flexible SHAVE cores were too configurable for DNN workloads, and instead, a simpler dot product engine was requisite to accelerate DNN computations with higher efficiency.

3 UNDERSTANDING CNN PERFORMANCE

In this section, we characterize inference latency and energy of the CNNs on three conventional mobile processors: CPU, GPU, and DSP as well as two mobile neural accelerators: NCS1 and NCS2. Our key takeaways are below:

- Takeaway 1: Conventional mobile processors like GPU and DSP are the winners compared to NCS accelerators in both latency and energy efficiency.
- Takeaway 2: We study why these CNNs are slower on the NCS than the GPU and DSP via a roofline analysis. We show that NCS accelerators are memory-bound due to limited available memory (e.g. NCS1 has 2MB on-chip memory).
- Takeaway 3: Vectorization and "memory wall" effects have a big impact on latency for dedicated accelerators like the NCS2 – by aligning computation size to the device vector size and on-chip memory, large speedups could be attained.

CNN Workloads. We study four widely used image classification CNN models: SqueezeNet [20], MobileNetV2 [35], ResNet50 [18], and InceptionV3 [38]. SqueezeNet and MobileNetV2 are designed specifically for mobile devices, while ResNet50 and InceptionV3 provides better accuracy and are commonly used as benchmarking workloads. The accuracy is measured on the image classification task (ImageNet 2012 [34]). Table 1 summarizes the model complexity and accuracy statistics. The number of million parameters measures the model size and the number of million FLOPs (floating point operations) reflects the model computational complexity.

CNN Model	Million FLOPs	Million Params	Top1 Accuracy
SqueezeNet	1683.1	1.25	57.5%
MobileNetV2	608.8	3.51	72.0%
ResNet50	7022.3	25.56	77.2%
InceptionV3	5744.4	25.57	78.8%

Table 1: Model stats for the four studied CNNs.

Measurement Setup and Metrics. We compare the performance of different processors based on inference latency and energy, two critical metrics on mobile devices. The *inference latency* is the time between feeding the image and finishing the classification. All reported numbers are averaged over 10 runs. We use Monsoon Power Monitor [37] to measure the power for the OnePlus 3 Android smartphone [31]. To measure the power of NCS1, we use a on-the-go cable to connect the NCS1 into the phone. As NCS2 doesn't support Android hosts, we measure the current and voltage on a PC, with a USB 3.0 measurement tool (MakerHawk UM34C [6]) and we scale the voltage and current measurements to make them comparable to the mobile platform.

We use the *NCSDK* [30] for the model inference on NCS1 and the *OpenVINO* [23] for NCS2. We use *TFLite* [8] for mobile CPU inference[2] and the *SNPE* [32] for GPU and DSP inference.

GPU and DSP Accelerate CNNs Way More than the NCS Accelerators. Figure 1a shows the inference latency of running the CNN models on the NCS1 and NCS2 compared with the smartphone CPU, GPU, and DSP. All devices demonstrate speedups over the CPU baseline for all CNNs. However, to our surprise, the NCS1 armed with the vision-oriented SHAVE cores turns out to be slower than conventional mobile GPU and DSP. This is likely the reason that the NCS2 abandoned the SHAVE cores for CNN acceleration, and instead opted to add a dedicated neural compute engine (NCE) specifically for DNNs. This is reflected in our latency measurements: NCS2 consistently beats NCS1 as expected, and is often faster than GPUs as well. DSPs, however, are still the fastest accelerator for CNNs over all of our workloads.

Fig. 1b shows the network energy consumption of over the five platforms. Again, the DSP was the clear winner, as it consumes the minimum energy across all models. The NCS1/2 consume more energy than the GPU, but much less than the CPU in all cases. This is surprising because mobile accelerators like NCS1 and NCS2 are designed to run the CNNs in a more energy-efficient way due to specialized chip design.

Based on the latency and energy study, the recent general notion [9, 10, 28] that specialized DNN hardware accelerator has short latency and impressive power efficiency does not hold in our case. Conventional mobile processors like GPU and DSP have highly optimized both hardware and software stacks and are able to achieve lower latencies than the NCS devices. Though DSP gives impressive low latency and energy results, it requires quantizing the CNNs into integer models and often needs additional model re-training efforts to avoid accuracy loss. Using GPU for inference however does not require such deployment cost. We suggest that when selecting the execution engine for CNNs on mobile devices, one can choose between the GPU and DSP if they are available based on the latency and energy constraints as well as re-training efforts.

Roofline Analysis of CNNs. To further understand why NCS accelerators are not comparable to the GPU or DSP, we use the roofline plots [43] to analyze the processors' performance.

We plot the ratio of FLOPs to number of parameters in Fig. 2 for each CNN as the operational intensity of each model. Roofline plots provide an easy way to visualize and compare the performance of different kinds of processors. On the x-axis, we have operational intensity, while the y-axis displays peak performance. Operational intensity equals the number of operations possible per available byte of data coming from external memory. Peak performance is the maximum number of computations possible, given the execution units available on chip. The roofline plot therefore relates performance to memory bandwidth as we will see. A typical roofline consists of two straight lines. The first one has constant slope (equal to external memory bandwidth) and shows the range of operational intensities at which the device has compute capacity to spare, but performance is limited by memory bandwidth. The second line is horizontal and is equal to the maximum compute capacity of the chip. Depending on operational intensity, a workload that intersects the roofline in the first (slanted) portion is hitting memory bandwidth limitations, and a workload that intersects the horizontal portion is compute bound.

Fig. 2 shows the roofline plot for all the five devices. According to the horizontal lines, we see the devices ordered in terms of their raw compute capacity. However, as we can see, the NCS devices have lower memory bandwidths (smaller slopes on slanted portion of roofline) compared to devices on the Snapdragon 820 chipset. If we look at the workloads except SqueezeNet, they have a computational intensity high enough to saturate the compute on CPU/GPU/DSP, but they cannot saturate the performance on NCS1/NCS2 because there isn't enough memory bandwidth to keep up with the computation capacity. This can be seen on the plot by identifying whether the DNN vertical line intersects with the horizontal (compute-bound) or slanted (memory-bound) portions of the roofline. SqueezeNet model is small enough to fit in all processors and is compute-bound.

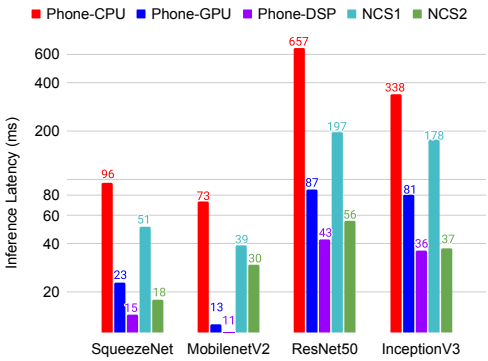
Vectorization & On-Chip Memory Effects on NCS2. Since it is difficult if impossible to change the hardware for better CNN performance, one can adapt the CNN models to better fit on the hardware processors. In this section we perform targeted experiments on specific DNN layers to expose different properties of our studied hardware platforms. Our goal is to study how to systematically change the network architectures that suit different hardware processors.

In the first experiment, we vary the filter depths of common 1×1, 3×3 and 5×5 filters to investigate the impact on latency on the NCS2 accelerator. Fig 3 shows the result for the NCS2 device. As we increase the filter depths, we observe abrupt increases in latency at specific depths. We call these *vectorization* effects as they are due to the compute size of the filter exceeding the vector instruction parallelism in the NCS2, thereby issuing a new (underutilized) vector instruction to complete the computation. For example, if vector width is 16 for multiply-accumulate instructions (MACs), and we have 17 MACs in our filter, then two vector instructions will need to be issued to compute the 17 MACs – the first will be fully utilized (16/16 MACs) while the second will be underutilized (1/16 MACs).

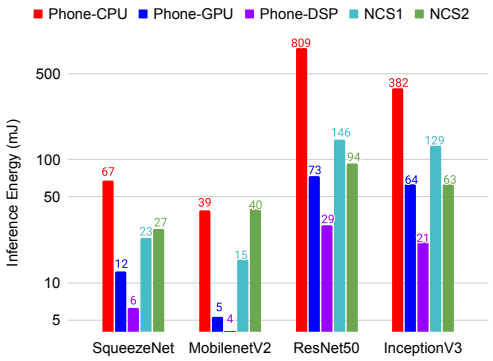
As shown in Fig. 3, the abrupt jumps for NCS2 generally occur after powers of two – this can be seen at depths 64, 128 and 256. There are also smaller jumps in latency at multiples of powers of two, for example 5×5 filters exhibit at jump at depth 192 (3×3 filters also increase in latency, but to a lesser extent). Such vectorization effects were much less on other devices (hence our focus on NCS2). This is likely because NCS2 uses a dedicated neural compute engine with simple flow control and a limited instruction set.

Other devices such as CPU/GPU have much finer-grained parallelism. Even NCS1 uses SHAVE cores for DNN acceleration, which has a finer-grained and more complex computation paradigm. In these cases, we expect that if 17 MACs need to be computed, a CPU (for example) can issue one vector instruction for 16 of the MACs, and one normal non-vector instruction for the remaining MAC, thereby displaying a smoother execution profile. However, the NCS2 is important to study in this detail as it is indicative of future dedicated accelerators for DNNs. We therefore recommend the careful design of DNNs to align to take advantage of such vectorization effects in DNN accelerators.

[2] inference using TFLite on the CPU is faster than SNPE on the CPU



(a) Inference latency



(b) Inference energy

Figure 1: Inference latency and energy for the studied CNNs on the mobile processors and NCS accelerators.

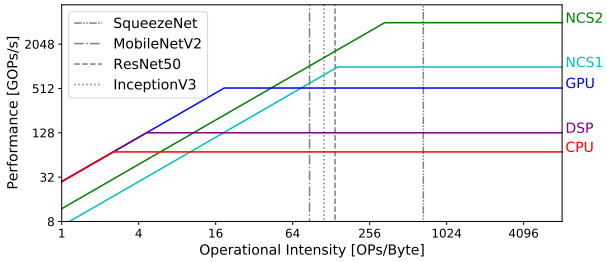


Figure 2: Roofline plot showing different devices and CNNs

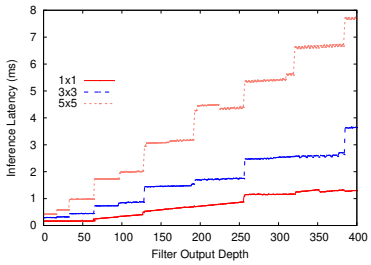


Figure 3: Vectorization effects of NCS2

We continue to increase the filter depths in Fig. 6 to study macro-scaling effects at larger filter sizes. We consistently observe a much larger jump (which we call the "memory wall") for different filter window sizes as shown on the plot. This dramatic increase in latency is likely due to each individual filter exceeding the size of the filter on-chip memory capacity on the NCS2, causing the filter reads to occur through external memory (which is often an order of magnitude slower). Curiously, the memory wall occurs at smaller filter depths for smaller filter window sizes. For 1×1 filters, it occurs at 1053, for 3×3 at 1316 and for 5×5 at 1755. This raises the question of whether OpenVINO software limitation could also be a contributing culprit to these memory walls. Another possible reason could be that different filter window sizes are laid out differently in on-chip memory causing larger window sizes to more efficiently use the on-chip memory resources. In any case, it is important to identify these memory walls for non-flexible dedicated DNN accelerators such as the NCE on NCS2 as they can impact performance by a large margin.

4 MEASUREMENT STUDY OF NLP MODELS

DNNs have brought rapid progress in many vision tasks that not only use CNN models but also neural building blocks widely used to understand language features. For example, recently multi-modality tasks such as image captioning [24] and visual question answering [7] require joint modeling of image and text input data. Recent state-of-the models [26, 27, 39] all use DNNs that include both the CNNs and a set of NLP model building blocks called Transformers [42]. In this section, we specifically discuss the NLP part (i.e. Transformer models) of these DNNs for multi-modal vision task.

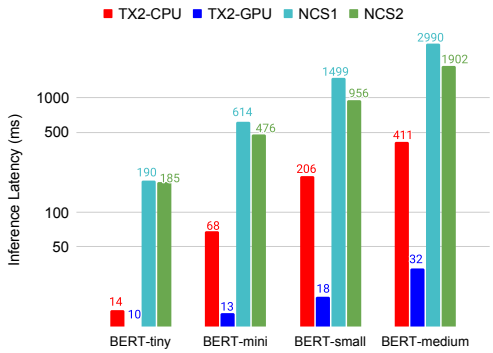
NLP Workloads. We study four variants of the pre-trained BERT models [13, 40]: BERT-tiny, BERT-mini, BERT-small, and BERT-medium. The accuracy is measured on the dev set of the sentiment analysis task [36] (SST [34]). Sentiment analysis is a common application used in many mobile devices such as Siri, Alexa and Google Home [1, 4]. Table 2 summarizes the model complexity and accuracy statistics. Similarly to the CNNs, the number of million parameters measures the model size and the number of million FLOPs (floating point operations) reflects the model computational complexity.

BERT Model	Million FLOPs	Million Params	SST Accuracy
BERT-tiny	4.4	311.1	80.4%
BERT-mini	11.3	2450.0	84.6%
BERT-small	29.1	9730.8	88.1%
BERT-medium	41.7	19459.3	90.5%

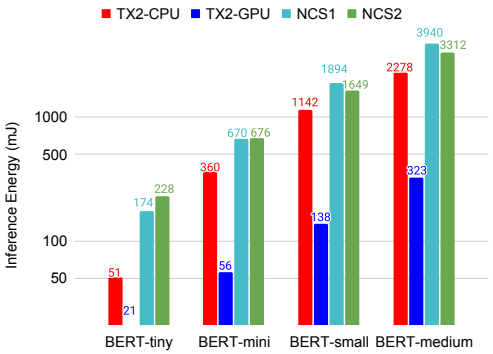
Table 2: FLOPs and accuracy stats of the four BERT models.

Measurement Setup and Metrics. Since the mobile phone does not easily support running the BERT models on the GPU. We instead use the Jetson TX2 mobile board [41] which has an ARM CPU and CUDA GPU. We compare the performance of the TX2 CPU and GPU with the NCS2 accelerator based on inference latency and energy. The *inference latency* is the time between feeding the input text (128 tokens) and finishing the classification. All reported numbers are averaged over 10 runs. We use the internal hardware-based power monitor available on the TX2 board to measure the energy. We use the *NCSDK* [30] for the model inference on NCS1, the *OpenVINO* [23] for NCS2, and TensorFlow [15] for inference on the TX2 CPU and GPU.

The NCS Accelerators Are Not Ready for Accelerating Transformer NLP Models. Figure 4a shows the inference latency of running four BERT models on the NCS1 and NCS2 compared with the smartphone CPU, GPU, and DSP. Suprisingly, both the NCS1 and NCS2 are 5 $\sim$ 10x slower than the CPU and are 20 $\sim$ 60 slower than the GPU. Fig. 4b shows the model energy consumption of over the five platforms. Again, although the gap is smaller than the



(a) Inference latency



(b) Inference energy

Figure 4: Inference latency and energy for the studied BERT models on the TX2 mobile board and NCS2 accelerator.

latency, NCS devices consume (3 ∼ 12)x more energy than the CPU and GPU. We hypothesize this is likely because neural accelerators designed for vision tasks are optimized for CNNs not for other types of model architectures.

Roofline Analysis of BERT Models. As described earlier, in the roofline model, the DNN vertical line intersects with the horizontal (compute-bound) or slanted (memory-bound) portions. Figure 5 shows all BERT models are compute-bound on the CPU and GPU and are memory-bound on the NCS2. On the NCS1, smaller models like BERT-tiny and BERT-mini are memory-bound but larger models like BERT-small and BERT-medium are compute-bound.

5 OPTIMIZING CNNS FOR NCS: A CASE STUDY

As is hard to change the underlying hardware to optimize CNN inference, we could change the CNN model design that best use the hardware processor features.

Specifically, we leveraged the vectorization effects to prune the InceptionV3 CNN for the NCS2 accelerator. In this proof-of-concept optimization attempt, we kept the experiment simple. To recoup most of the gains from vectorization, we decreased the output filter depths in InceptionV3 to the closest power of two – this decreased its number of parameters to from 25M to 12M parameters. We then retrained the CNN using stochastic gradient descent (SGD) with momentum 0.9, weight decay 10^{-4} and learning rate 0.1 (which was divided by 10 every 30 epochs). Even though we decreased the number of parameters by more than half, the accuracy only dropped by 1.2%. For NCS2, this device-aware pruning resulted in a drop of latency from 37 ms to 20 ms, and power dropped slightly by 7%. Overall, this improved perf/watt by ∼ 2×.

		Latency	Power	Perf/Watt
Original	NCS2	37 ms	633 mW	43 fps/W
	DSP	**36 ms**	**587 mW**	**47 fps/W**
Pruned	NCS2	**20 ms**	592 mW	84 fps/W
	DSP	23 ms	**478 mW**	**91 fps/W**

Table 3: Latency, power and efficiency of InceptionV3 before/after NCS-aware pruning on NCS2 and DSP.

When we ran the same pruned CNN on the DSP, the gains were not as dramatic because this pruning was done specifically for NCS2. As Table 3 shows, NCS2 becomes 15% faster than DSP after pruning, even if it was 3% before pruning. However, the DSP still wins in overall efficiency. We used just one of our layer-wise investigation takeaways, and a significant speedup could already be observed for the NCS2 accelerator. This confirms that device-specific tuning of CNNs is requisite to be able to make the most out of dedicated CNN accelerators like the NCS2. However, well-known DNNs are typically design with commodity processors in mind which overlooks the specific properties of fixed-function accelerators. We anticipate that in the future more device-specific DNN design will occur as these new DNN accelerators become more mainstream.

6 RELATED WORK

Processor Hardware and System Optimization. As the compute cost of machine learning models continues to grow, and their adaptation increases, more optimizations are required to deploy them in practice. There is also an increase desire for ASIC accelerators specifically designed for NN workloads. Many of these optimizations are achieved using custom hardware acceleration blocks [10, 16]. Most of these solutions are designed to speed up inferences, while others aim to accelerate the training phase. Given popularity of DNNs it's not surprising that there has been a spate of publications, prototypes, and commercial hardware accelerations. While less flexible than other platform they can offer better energy-efficiency and smaller silicon area footprint.

DNN Benchmarks. AI benchmarks [21] studies eight computer vision workloads for various mobile SoCs. [17] characterizes the CNN latency and throughput for mobile vision tasks. Both of them don't study the CNN performance for commodity hardware accelerators. [5] discusses various deep learning workloads for training and inference on server-level processors.

7 CONCLUSION

Motivated by the looming arrival of neural network accelerators in embedded and mobile devices, we have conducted systematic measurement studies that considers a type of commercially available neural network accelerator: the Intel Neural Compute Stick. We compared the latency and energy performance of CNNs and NLP models to conventional mobile processors like CPUs, GPUs and DSPs. We find that without dedicated software stack optimization, mobile accelerators like NCS are not ready yet to replace GPUs and DSPs. Our findings shed light on architectural properties of fixed function accelerators such as the NCS1/2, and our takeaways and initial accelerator-aware optimizations for CNNs can be further exploited to attain an improvement in efficiency for emerging mobile DNN accelerators.

REFERENCES

[1] 2013. Sentiment analysis and artificial intelligence: Siri, should I open this email? https://venturebeat.com/2013/04/01/sentiment-analysis-and-artificial-

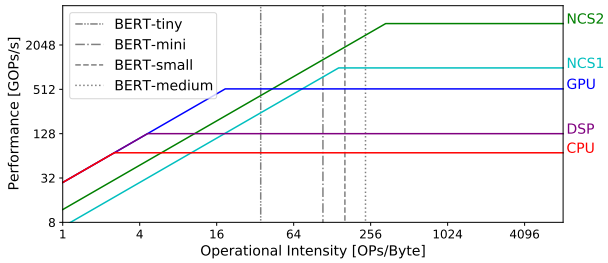


Figure 5: Roofline plot showing different devices and BERT

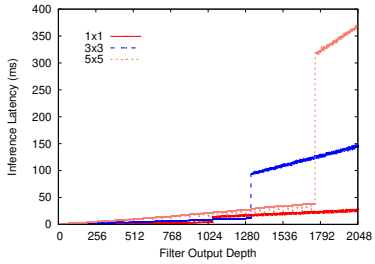


Figure 6: Memory effects of NCS2

intelligence-siri-should-i-open-this-email/

[2] 2018. Google APIs for Android. https://developers.google.com/android/reference/com/google/android/gms/location/DetectedActivity.

[3] 2018. Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant. https://machinelearning.apple.com/2017/10/01/hey-siri.html.

[4] 2019. Beyond Siri, Google Assistant, and Alexa – what you need to know about AI Conversational Applications. https://www.kdnuggets.com/beyond-siri-google-assistant-and-alexa-what-you-need-to-know-about-ai-conversational-applications.html/

[5] R. Adolf, S. Rama, B. Reagen, G. Wei, and D. Brooks. 2016. Fathom: reference workloads for modern deep learning methods. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, New York, NY, USA, 1–10. https://doi.org/10.1109/IISWC.2016.7581275

[6] Amazon. 2019. MakerHawk UM34C. https://www.amazon.com/MakerHawk-Bluetooth-Voltmeter-Multimeter-Resistance/dp/B07DK4GDSP. accessed 22-October-2019.

[7] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *Proc. of the IEEE international conf. on computer vision*. 2425–2433.

[8] TensorFlow Authors. 2018. TensorFlow Lite. https://www.tensorflow.org/. accessed 8-April-2017.

[9] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma. 2015. Always-on Vision Processing Unit for Mobile Applications. *IEEE Micro* 35, 2 (Mar 2015), 56–66.

[10] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE, New York, NY, USA, 367–379.

[11] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[12] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv:1602.02830* (2016).

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc of the 2019 Conf. of the NAACL-HLT*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[14] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv:1412.6115* (2014).

[15] Google. 2018. TensorFlow. https://www.tensorflow.org/. accessed 8-May-2018.

[16] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE*. IEEE, 243–254.

[17] Jussi Hanhirova, Teemu Kämäräinen, Sipi Seppälä, Matti Siekkinen, Vesa Hirvisalo, and Antti Ylä-Jääski. 2018. Latency and Throughput Characterization of Convolutional Neural Networks for Mobile Computer Vision. In *Proc. of the 9th ACM Multimedia Systems Conf.* (Amsterdam, Netherlands) *(MMSys '18)*. ACM, New York, NY, USA, 204–215. https://doi.org/10.1145/3204949.3204975

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

[19] Huawei. 2018. Kirin 970. http://www.hisilicon.com/en/Media-Center/News/Key-Information-About-the-Huawei-Kirin970.

[20] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv:1602.07360* (2016).

[21] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, and L. Van Gool. 2018. AI Benchmark: Running Deep Neural Networks on Android Smartphones. *ArXiv e-prints* (Oct. 2018). arXiv:1810.01109 [cs.AI]

[22] Intel. 2019. Intel Neural Compute Stick 2. https://software.intel.com/en-us/neural-compute-stick. accessed 22-October-2019.

[23] Intel. 2019. OpenVINO Toolkit. https://software.intel.com/en-us/openvino-toolkit. accessed 21-October-2019.

[24] Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proc. of the IEEE conf. on computer vision and pattern recognition*. 3128–3137.

[25] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. 2016. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices. In *2016 15th ACM/IEEE International Conf. on Information Processing in Sensor Networks (IPSN)*. 1–12.

[26] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 13–23.

[27] Jiasen Lu, Vedanuj Goswami, Marcus Rohrbach, Devi Parikh, and Stefan Lee. 2020. 12-in-1: Multi-Task Vision and Language Representation Learning. 10437–10446.

[28] David Moloney, Brendan Barry, Richard Richmond, Fergal Connor, Cormac Brick, and David Donohoe. 2014. Myriad 2: Eye of the computational vision storm. In *Hot Chips 26 Symposium (HCS), 2014 IEEE*. IEEE, 1–18.

[29] Movidius. 2018. Movidius Neural Compute Stick. https://developer.movidius.com/. accessed 23-March-2018.

[30] Movidius. 2018. SDK for the Neural Compute Stick. https://github.com/movidius/ncsdk. accessed 21-April-2018.

[31] OnePlus. 2018. OnePlus 3. https://www.oneplus.com/3.

[32] Qualcomm. 2018. Snapdragon Neural Processing Engine. https://developer.qualcomm.com/software/snapdragon-neural-processing-engine. accessed 13-May-2018.

[33] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 267–278.

[34] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[35] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*. 4510–4520.

[36] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. of the 2013 conf. on empirical methods in natural language processing*. 1631–1642.

[37] Monsoon Solutions. 2018. Monsoon Power Monitor. https://www.msoon.com/LabEquipment/PowerMonitor/.

[38] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proc. of the IEEE conf. on computer vision and pattern recognition*. 2818–2826.

[39] Hao Tan and Mohit Bansal. 2019. LXMERT: Learning Cross-Modality Encoder Representations from Transformers. In *Proc. of the 2019 (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 5100–5111. https://doi.org/10.18653/v1/D19-1514

[40] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv:1908.08962* (2019).

[41] Nvidia TX2. 2018. Nvidia TX2. https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[43] Samuel Williams, Andrew Waterman, , and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures. *Commun. ACM* (2009).