On the Correlation of CNN Performance and Hardware Metrics for Visual Inference on a Low-Cost CPU-based Platform

Delia Velasco-Montero[1], Jorge Fernández-Berni[1], Ricardo Carmona-Gálán[1], Angel Rodríguez-Vázquez[1]
[1]Instituto de Microelectrónica de Sevilla (Universidad de Sevilla-CSIC), Sevilla, Spain
delia@imse-cnm.csic.es

***Abstract*—While providing the same functionality, the various Deep Learning software frameworks available these days do not provide similar performance when running the same network model on a particular hardware platform. On the contrary, we show that the different coding techniques and underlying acceleration libraries have a great impact on the instantaneous throughput and CPU utilization when carrying out the same inference with Caffe, OpenCV, TensorFlow and Caffe2 on an ARM Cortex-A53 multi-core processor. Direct modelling of this dissimilar performance is not practical, mainly because of the complexity and rapid evolution of the toolchains. Alternatively, we examine how the hardware resources are distinctly exploited by the frameworks. We demonstrate that there is a strong correlation between inference performance – including power consumption – and critical parameters associated with memory usage and instruction flow control. This identified correlation is a preliminary step for the development of a simple empirical model. The objective is to facilitate selection and further performance tuning among the ever-growing zoo of deep neural networks and frameworks, as well as the exploration of new network architectures.**

***Keywords*—convolutional neural networks, deep learning, edge inference, embedded vision, hardware performance, software frameworks**

I. Introduction

Deep Learning (DL) [1] has emerged as the reference paradigm for applications demanding accurate inference. In particular, concerning computer vision, Convolutional Neural Networks (CNN) are being employed for multiple tasks, ranging from image recognition to pixel-wise segmentation. This versatility along with much higher accuracy in comparison with classical vision approaches come at the cost of notably increasing the requirements for computational and memory resources [2]. This constitutes a major challenge for the implementation of CNNs in embedded systems [3].

The relevance gained by the DL paradigm in the last few years has driven the development of several software frameworks for prototyping and practical deployment of CNNs. While globally targeting the same functionality, each of these frameworks follows a particular approach and exploits specific libraries to deal with the massive computational load demanded by deep neural networks. For instance, matrix-matrix or matrix-vector operations, which are the backbone of CNNs, can be realized by Basic Linear Algebra Subroutines (BLAS) [4] [5] available in a number of libraries: Atlas [6], MKL [7], OpenBLAS [8] [9], Eigen [10], cuBLAS [11], etc. This diversity of strategies and tools result in remarkable different inference performance from DL frameworks, even when they are running the same CNN model on a common hardware platform. Direct modelling of this heterogeneous performance is unmanageable due to the complexity of the frameworks and their rapid evolution.

In this context, research on CNN is usually focused on a straightforward assessment. For instance, some works disseminate a throughput comparison of various DNN frameworks [12]–[14], including Caffe, TensorFlow, Torch, CNTK, MXNet, etc. Even so, actually focused on embedded platforms, fewer contributions have evaluated the efficiency of DNN software tools for computer vision at the edge [15]–[18]. All these benchmarks extract direct metrics from CNN inference. Although more customized and specific CNN implementations on CPU-based embedded systems have been reported [19] [20], a generalized study should include popular DL frameworks that can operate on a wide range of embedded devices.

In this paper we explain performance of embedded DL inference indirectly through metrics of hardware exploitation that can be easily measured, alternatively to such usually followed direct approach. The analysis is carried out on a Raspberry Pi 3 Model B [21] (RPi), an inexpensive embedded computer featuring a 4-core ARM Cortex A-53 CPU. We first report the performance achieved by four popular DNN frameworks in terms of throughput and CPU utilization when performing 1000-category image classification. We then correlate these performance figures with hardware events registered during inference, pointing out the critical aspects at both software and hardware level affecting each other. Finally, we show that some of the registered hardware events exhibit a strong correlation with power consumption as well. Overall, we are setting the foundations for the next step in our research, namely the development of a methodology for simple empirical modelling of DL inference on CPU-based embedded platforms.

II. Performance Analysis

A. Hardware Platform

All the experiments reported in this paper refers to the Quad Core ARM Cortex-A53 1.2GHz 64-bit CPU [22] [23] included

978-1-7281-3227-3/19/$31.00©2019 IEEE

in the Broadcom BCM2837 System-on-a-Chip (SoC) of the Raspberry Pi 3 Model B. Each core of this CPU is in turn an ARMv8-A processor capable of independently executing instructions.

Cortex-A53 processors exhibit two memory systems, namely Level 1 (L1) and Level 2 (L2). The L1 memory system includes, per core processor, separate instruction and data caches (I-cache, D-cache), and a Memory Management Unit (MMU). The MMU in turn features one Translation Lookaside Buffer (TLB) – a two-level cache for instruction and data that translates between virtual and physical addresses. Instruction caching and dynamic branch prediction are also allowed in order to increase overall performance and reduce power consumption. The L2 memory system contains a unified cache, which is shared between the cores. Specifically, for the SoC of the RPi, L1 amounts to 32KB whereas L2 comprises 512KB.

Each ARMv8-A core implements the so-called Advanced Single Instruction Multiple Data architecture – commonly referred to as ARM NEON technology – as well as vector floating-point (VFP) operations for acceleration [24].

In addition to the SoC, the Raspberry Pi features 1GB RAM LPDDR2 900MHz, where we load the CNN weights and keep intermediate results while running the networks. The non-volatile storage capacity of the system is provided by an attached micro-SD card. The operating system is Raspbian [25].

B. Software Frameworks

Caffe [26] implements convolutions as image-to-column transformation (*im2col*) plus General Matrix-Matrix Multiplication (GEMM), using Basic Linear Algebra Subprograms (BLAS) as the back-end for GEMM. According to our tests – not reported in this paper – OpenBLAS [9] is the BLAS library supported by the RPi CPU, and compatible with Caffe, that better leverages the four cores of the ARM Cortex-A53. **TensorFlow** [27] expresses computations as static graphs, which are built just once and run repeatedly for inference. It makes use of the Eigen library [10] to generate efficient parallel code for multicore CPUs. We installed pre-built TensorFlow 1.3.0 for RPi [28]. This version exploits ARM hardware optimizations – NEON and VFP – for computational acceleration. **OpenCV** [29] implements a module that allows the use of pre-trained models for inference from other frameworks. We took CNN model files from Caffe. OpenCV version 3.3.1 was compiled to exploit both ARM NEON and VFP optimizations as well. **Caffe2** [30] is designed to be lightweight, modular and mobile-oriented. It also uses static graphs for network definition and the Eigen library for matrix calculation. Caffe2 is optimized for ARM CPUs with NEON.

C. Inference Performance

One of the consequences of the high computational demand of CNN models is that the temperature of the RPi's ARM Cortex-A53 SoC can rapidly increase during inference. This forces the CPU frequency, and thereby the throughput, to go down. To take this aspect into account, we measured the following four performance metrics after each processed image over a long period – 6 minutes – of continuous inference:

- **Throughput**. It is calculated as the inverse value of the total time required per image when batch size is set to 1 – this includes the time required to read and pre-processing the image, perform the inference, extract the metrics and save the results.
- **CPU utilization**. It was measured by using the Python `psutil` library.
- **CPU frequency and temperature**. We used the `vcgencmd` tool to check the variations of the SoC's temperature and frequency caused by the CNN-based inference. Although the ARM Cortex-A53 CPU ideally operates at a maximum frequency of 1.2 GHz, the chip temperature can alter the instantaneous CPU frequency.

To identify performance trends on each DL framework, these metrics were measured for three CNNs with different architectures capable of recognizing 1000 image categories, namely SqueezeNet [40], GoogLeNet [41], and ResNet-50 [42]. We used pre-trained implementations of these models provided by each framework [31]–[39]. Table I summarizes main architectural and operational aspects of them. Python was the coding language we used since it is the language through which all of the network definitions are available for all of the frameworks. Furthermore, all these DL tools use single-precision floating-point data format (*float32*).

Fig. 1 depicts the temporal evolution of the metrics above defined when performing image recognition with SqueezeNet. Fig. 2 shows the average values of CPU utilization and throughput for all the cases during the 6-minute period inference. The following aspects must be emphasized:

- CPU utilization is quite stable for each framework over the inference period, but its average varies significantly among frameworks. Caffe reaches the highest value for the three network models tested.
- There are different patterns of temperature evolution. When the temperature is approaching 80°C, the processor protects itself by downclocking, which in turn decreases the throughput. This has a great impact on the total number of processed images over the test period.
- In spite of the fact that Caffe is apparently the framework making the most of the CPU, its throughput is the lowest

TABLE I
MAIN PARAMETERS DEFINING THE ASSESSED CNN ARCHITECTURES.

	GoogLeNet	ResNet-50	SqueezeNet-v1.1
Model Repository	[31]–[33]	[34]–[36]	[37]–[39]
Top-1 (%) accuracy[a]	69.2±0.4	72.6±0.1	58.3±0.0
Top-5 (%) accuracy[a]	89.0±0.1	91.0±0.0	80.0±0.1
Input size	1x224x224x3	1x224x224x3	1x227x227x3
#Outputs	1000	1000	1000
#Conv. layers	57	53	26
#Fully-Conn. layers	1	1	0
#weights	∼7.0M	∼25.6M	∼1.2M
#MACs	∼1.6G	∼3.9G	∼396k

[a]Accuracy measured over the validation set of the ImageNet ILSVRC 2012 dataset, without any data augmentation. Random initialization of weights leads to small deviations in accuracy even if it is the same model but trained on each framework.

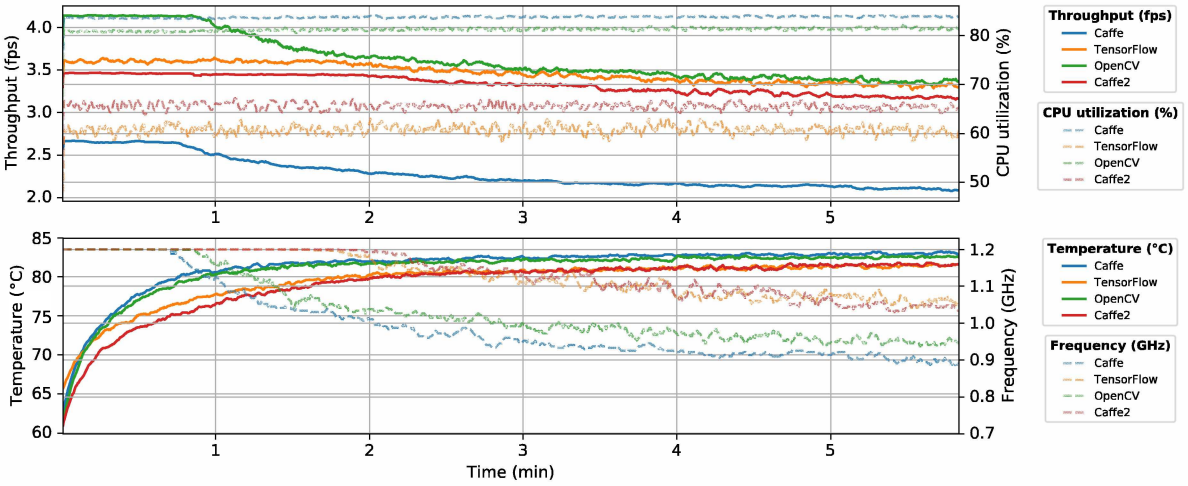

Figure 1. Extracted performance metrics when running SqueezeNet. Similar trends are observed for GoogLeNet and ResNet-50.

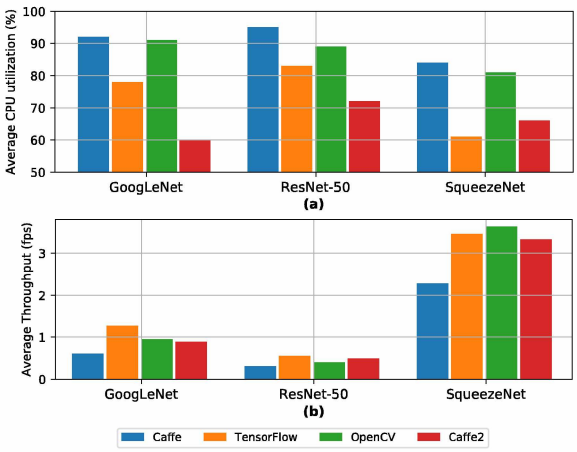

Figure 2. Average values of CPU utilization (a) and throughput (b) during a 6-minute period of continuous inference.

for the three CNNs. (Actually, we have observed this seeming contradiction for still two more models, namely Network-in-Network [43] and MobileNet [44].)

Next, we delve into the details of hardware exploitation in order to elucidate the underlying reasons for this behavior, in particular for the contradiction arising in Caffe between CPU utilization and throughput.

III. Hardware Exploitation Analysis

A. Methodology

We extracted statistics on the processing load and memory usage of the CPU for the four analyzed frameworks when performing inference with SqueezeNet, GoogLeNet and ResNet-50. For the sake of a fair comparison, we set a fixed number of images, $N = 50$, to be processed in all cases. Otherwise, the resulting metrics would be biased by the different inference pace of each framework. These 50 images were randomly taken from the ImageNet dataset [45] – using different input images does not change the quantitative outcomes of our study. The targeted parameters were obtained from the `perf` tool [46], which gathers data through counters and event monitors provided by the Performance Monitoring Unit (PMU) included in the Cortex-A53 processor. In particular, we gathered data related to PMU hardware events [47]. In order to dismiss statistics related to the load of the CNN model weights – our analysis is focused on inference processing –, a two-phase approach was carried out. Firstly, we collected statistics when running the whole inference script (s_1). Secondly, we singled out the counts associated with the creation and load of the model architecture (s_2). Thus, the statistics employed to compare hardware performance are derived as $(s_1 - s_2)/N$, i.e., statistics that represent per-image performance. In order to reduce estimation errors – keep in mind that `perf` provides count estimates –, we averaged the values from 5 measurements.

B. Experimental Results and Discussion

Fig. 3 depicts the most representative parameters among all the gathered statistics. Let us carefully examine them. First, note that the particular coding techniques and libraries making up Caffe render, for the RPi's CPU, the highest number of instructions (Fig. 3(a)) and demand the highest number of memory accesses (Fig. 3(c)) in all cases. The processor does its best to cope with these requirements. That's why the rates of instructions per second (Fig. 3(b)) and memory access per second (Fig. 3(d)) are also the highest for Caffe, which in turn explains the fact that this framework reaches the highest value of CPU utilization mentioned in Section II-C. However, even executing more instructions per second and fetching more data per second than any other framework, Caffe attains the lowest throughput due to its notably greater demand of processing and memory as a whole. We must also point out that Caffe is the framework for which the CPU applies branch prediction more extensively (Fig. 3(k)). This means that the processor

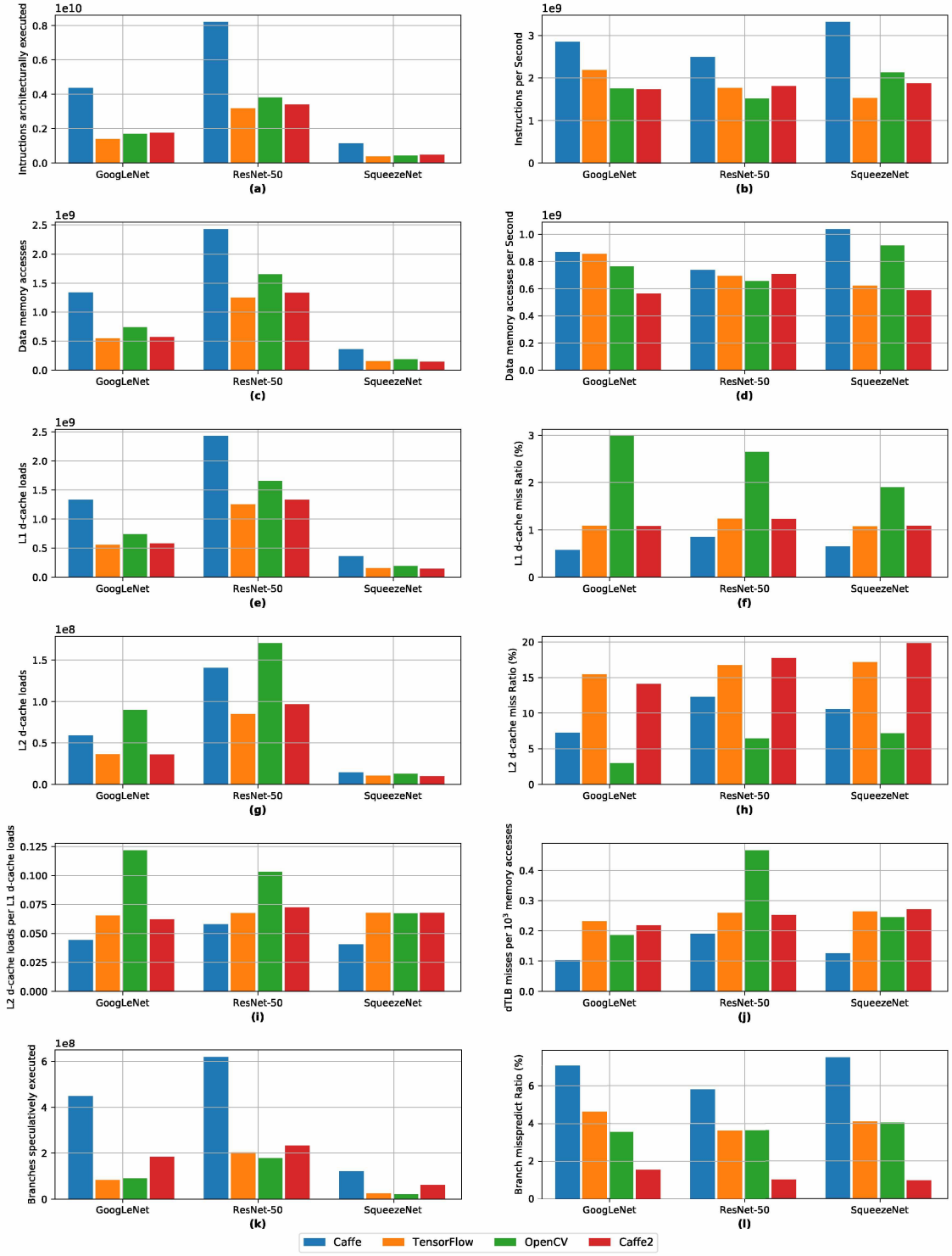

Figure 3. Hardware event statistics registered for 50 images consecutively inferred.

executes instructions before knowing for sure whether they will be finally executed or not. If the prediction was correct, the result is available sooner, thereby accelerating inference. In the case of Caffe, the performance of the CPU in terms of branch prediction is poor (Fig. 3(l)), adding up instructions uselessly executed. Concerning cache exploitation, Caffe is distinctively good at loading data at L1 (Fig. 3(e)) which will be successfully fetched later on (Fig. 3(f)). The exploitation of L2 and TLB by Caffe is also notable (Figs. 3(g)-3(j)) – note that the OpenBLAS library, exploited by Caffe, is highly oriented

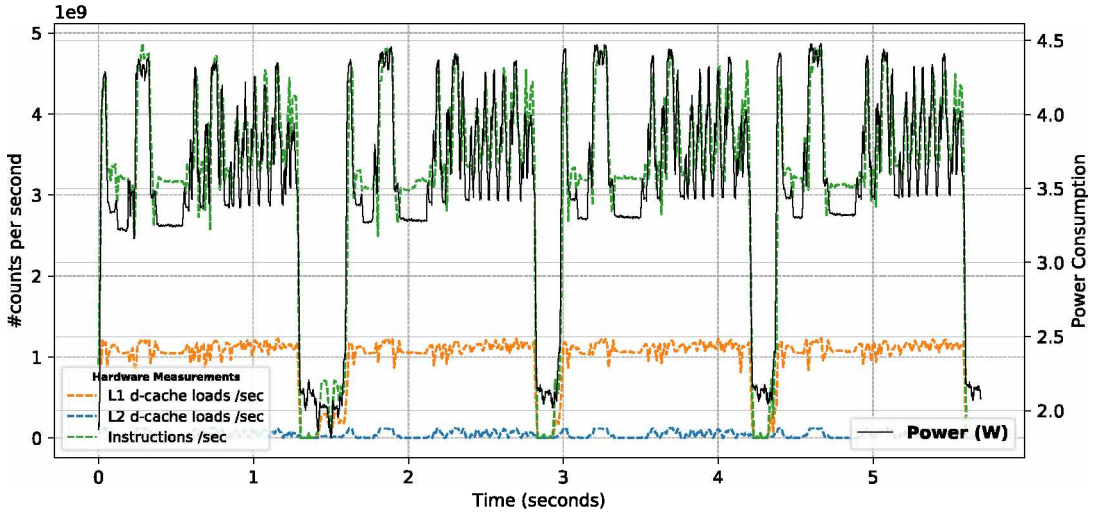

Figure 4. Power consumption and correlated hardware metrics when running GoogLeNet on Caffe four times.

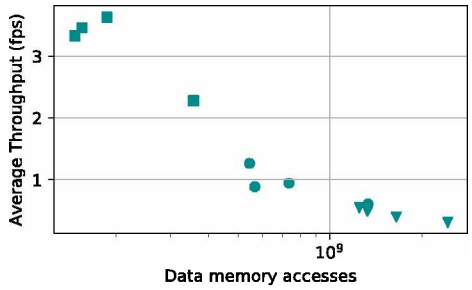

Figure 5. Alignment between throughput and one hardware metric for the 12 combinations of assessed frameworks and networks. Note the logarithmic scale in the x-axis.

TABLE II
PEARSON CORRELATION COEFFICIENT BETWEEN INSTANTANEOUS POWER CONSUMPTION AND THREE HARDWARE METRICS FOR GOOGLENET.

	L1-d cache loads /sec	L2-d cache loads /sec	Instructions /sec
Caffe	0.85	0.72	0.94
TensorFlow	0.95	0.88	0.92
OpenCV	0.89	0.66	0.89
Caffe2	0.82	0.79	0.80

to this accomplishment [8]. This suggests that the main reason for the poor coupling between Caffe and this CPU could be a poor mapping between the high-level instructions in Caffe's source code and the processor's instruction set.

With respect to the other three frameworks, there are also differences to be highlighted. The instruction reduction with respect to Caffe showed in Fig. 3(a) suggests a better exploitation of the ARM SIMD instruction set. In fact, these frameworks leverage the ARM hardware optimizations by compilation. TensorFlow stands out as the most efficient framework, requiring the minimum number of instructions and memory accesses to complete the inference (Fig. 3(a) and Fig. 3(c), respectively). This characteristic, in conjunction with high rates of instruction execution and memory access (Fig. 3(b) and Fig. 3(d), respectively), enable the highest average throughput achieved by TensorFlow for GoogLeNet and ResNet-50; OpenCV is the best option for SqueezeNet (see Fig. 2(b)). The most effective framework in terms of branch prediction is Caffe2. Regarding cache memory exploitation, TensorFlow and Caffe2 present a similar performance. OpenCV makes a poor use of L1 but is the best by far on exploiting L2 (Figs. 3(e)-3(h)).

Concerning the differences between the three studied CNN architectures, number of executed instructions in Fig. 3(a) exhibit a concordance with the number of MAC operations reported in Table I – although each framework depicting a distinctive relationship, as explained above. Likewise, the more weights the network has (Table I), the more data memory accesses it requires (Fig. 3(c)). In addition, the extracted hardware metrics have a remarkable correlation with throughput for all the networks as highlighted in Fig. 5, where previously reported values are scattered showing a nearly linear pattern.

C. Power Consumption

Besides explaining throughput and CPU usage as discussed, the statistics extracted with the `perf` tool also exhibit correlation with power consumption. Fig. 4 depicts instantaneous power measured with a Keysight N6705C DC Power Analyzer vs. three hardware metrics simultaneously sampled every 10 milliseconds. This figure correspond to four consecutive GoogLeNet inferences running on Caffe. Similar results have been obtained for the other frameworks and networks. Table II summarizes the Pearson correlation coefficients between these metrics and power consumption for each framework running GoogLeNet. Note that the coefficients are greater than 0.66 in all cases, reaching a value of 0.95 for L1 D-cache accesses during inference on TensorFlow. Taking into account the importance of power consumption in embedded vision applications,

its relevance in optimization loops [48], and how difficult its direct measurement is – supply pins must be accessible and special equipment like the aforementioned power analyzer is required –, the proposed hardware metrics constitute a simple way to characterize embedded platforms.

IV. Conclusion

An optimal selection of DL software framework and DNN architecture for a particular embedded hardware platform definitely make a difference in terms of performance. Specific coding strategies and acceleration libraries implemented by the frameworks exploit the underlying hardware in diverse manners, giving rise to a wide range of inference rates and power profiling even on the same network model. Instead of a direct modelling of the expected performance and power consumption of DL frameworks and DNNs on a particular CPU-based platform, we propose to carry out such modelling through metrics of hardware exploitation. These metrics can be easily extracted through standard tools. In this paper we present a preliminary study that supports the applicability of our proposed approach. Our next step will be to develop a performance model based on such metrics and insert it into an optimization loop in order to determine the best selection according to prescribed specifications.

Acknowledgment

This work was supported by United States Office of Naval Research through ONR NICOP N00014-19-1-2156, by Spanish Government MINECO (European Region Development Fund, ERDF/FEDER) through Project TEC2015-66878-C3-1-R, by Spanish Government through FPU Grant FPU17/02804, and by EU H2020 MSCA ACHIEVE-ITN, Grant No 765866.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.
[2] M. Verhelst and B. Moons, "Embedded deep neural network processing," IEEE Solid-State Circuits Magazine, vol. 9, no. 4, pp. 55–65, 2017.
[3] V. Sze, "Designing hardware for machine learning," IEEE Solid-State Circuits Magazine, vol. 9, no. 4, pp. 46–54, 2017.
[4] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for fortran usage," ACM Trans. Math. Softw.,vol. 5, no. 3, pp. 308–323, Sep. 1979.
[5] T. e. a. Kielmann, Basic linear algebra subprograms (BLAS), 2011.
[6] Automatically tuned linear algebra software (ATLAS). [Online]. Available: http://math-atlas.sourceforge.net
[7] Intel Math Kernel Library. [Online]. Available: https://software.intel.com/en-us/mkl
[8] Goto, K. et al., "Anatomy of high-performance matrix multiplication," ACM Trans. Math. Softw., vol. 34, no. 3, pp. 12:1–12:25, May 2008.
[9] OpenBLAS, optimized BLAS library based on GotoBLAS2 1.13 BSD version. [Online]. Available: https://github.com/xianyi/OpenBLAS
[10] Guennebaud, G. et al., "Eigen." [Online]. Available: http://eigen.tuxfamily.org/
[11] Dense linear algebra on GPUs. [Online]. Available: https://developer.nvidia.com/cublas
[12] Bahrampour, S. et al, "Comparative study of Caffe, Neon, Theano, and Torch for deep learning," arXiv, no. 1511.06435, 2015.
[13] Hanhirova, J. et al., "Latency and throughput characterization of convolutional neural networks for mobile computer vision," arXiv, no. 1803.09492, 2018.
[14] Shaohuai, S. et al., "Benchmarking state-of-the-art deep learning software tools," arXiv, no. 1608.07249, 2016.
[15] Ignatov A., et al., "AI Benchmark: running deep neural networks on android smartphones," arXiv, no. abs/1810.01109, 2018.
[16] Velasco-Montero, D. et al., "Optimum selection of DNN model and framework for edge inference," IEEE Access, vol. 6, pp. 51 680–51 692, 2018.
[17] Zhang, X. et al., "pCAMP: Performance comparison of machine learning packages on the edges," USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18). USENIX Association, 2018.
[18] Pena, D. et al., "Benchmarking of CNNs for low-cost, low-power robotics applications," RSS Workshop: New Frontier for Deep Learning in Robotics, 2017.
[19] Lee, Sung-Jin et al., "Efficient SIMD implementation for accelerating convolutional neural network,", pp. 174–179, 2018.
[20] Liangzhen Lai, L. et al., "CMSIS-NN: Efficient neural network kernels for Arm Cortex-M CPUs," arXiv, no. 1801.06601, 2018.
[21] "Raspberry Pi 3 Model B." [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[22] ARM, ARM Cortex-A53 MPCore Processor.Technical Reference Manual. [Online]. Available: https://developer.arm.com/docs/ddi0500/g
[23] ARM Processors.Cortex-A53. [Online]. Available: https://developer.arm.com/products/processors/cortex-a/cortex-a53
[24] ARM, ARM Cortex-A53 MPCore Processor Advanced SIMD and Floating-point Extension. Technical Reference Manual.
[25] Raspbian. [Online]. Available: https://www.raspberrypi.org/downloads/raspbian/.
[26] Jia, Y. et al., "Caffe: Convolutional architecture for fast feature embedding," arXiv, no. 1408.5093, 2014.
[27] Abadi, M. et al., "Tensorflow: A system for large-scale machine learning," 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283, 2016.
[28] A Docker image for Tensorflow. [Online]. Available: https://github.com/DeftWork/rpi-tensorflow
[29] OpenCV. [Online]. Available: https://opencv.org/
[30] Caffe2. [Online]. Available: https://caffe2.ai/
[31] BAIR/BVLC GoogLeNet Model. [Online]. Available: https://github.com/BVLC/caffe/tree/master/models/bvlc_googlene
[32] Inception V1. [Online]. Available: http://download.tensorflow.org/models/inception/v1/2016/08/28.tar.gz
[33] Caffe2 Models. BVLC GoogLeNet. [Online]. Available: https://github.com/caffe2/models/tree/master/bvlc/googlenet
[34] Deep Residual Learning for Image Recognition. [Online]. Available: https://github.com/KaimingHe/deep-residual-networks
[35] ResNet V1 50. [Online]. Available: http://download.tensorflow.org/models/resnet/v1/50/2016/08/28.tar.gz
[36] Caffe2 Models. ResNet50. [Online]. Available: https://github.com/caffe2/models/tree/master/resnet50
[37] SqueezeNet v1.1. [Online]. Available: https://github.com/DeepScale/SqueezeNet/tree/master/SqueezeNet/v1.1
[38] Caffe to TensorFlow. [Online]. Available: https://github.com/ethereon/caffe-tensorflow
[39] Caffe2 Models. SqueezeNet. [Online]. Available: https://github.com/caffe2/models/tree/master/squeezenet
[40] Iandola, F. et al., "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1MB model size," arXiv, no. 1602.07360, 2016.
[41] Szegedy, C. et al., "Going deeper with convolutions," arXiv, no. 1409.4842, 2014.
[42] Kaiming, He et al., "Deep residual learning for image recognition," arXiv, no. 1512.03385, 2015.
[43] M. Lin, Q. Chen, and S. Yan, "Network in network," arXiv, no. 1312.4400, 2013.
[44] Howard, A. et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv, no. 1704.04861, 2017.
[45] Russakovsky, O. et al., "ImageNet large scale visual recognition challenge," International Journal of Computer Vision (IJCV), vol. 115, no. 3, pp. 211–252, 2015.
[46] perf: Linux profiling with performance counters. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main Page
[47] ARM, ARM architecture reference manual. ARMv8, for ARMv8-A architecture profile, 2017.
[48] T. Y. et al., "Netadapt: Platform-aware neural network adaptation for mobile applications," arXiv, no. 1804.03230, 2018.