# Performance Assessment of Deep Learning Frameworks through Metrics of CPU Hardware Exploitation on an Embedded Platform

Original Scientific Paper

**Delia Velasco-Montero**
Instituto de Microelectrónica de Sevilla
Universidad de Sevilla-CSIC
Sevilla, Spain
delia@imse-cnm.csic.es

**Jorge Fernández-Berni**
Instituto de Microelectrónica de Sevilla
Universidad de Sevilla-CSIC
Sevilla, Spain
berni@imse-cnm.csic.es

**Ricardo Carmona-Galán**
Instituto de Microelectrónica de Sevilla
Universidad de Sevilla-CSIC
Sevilla, Spain
rcarmona@imse-cnm.csic.es

**Ángel Rodríguez-Vázquez**
Instituto de Microelectrónica de Sevilla
Universidad de Sevilla-CSIC
Sevilla, Spain
angel@imse-cnm.csic.es

***Abstract*** *– In this paper, we analyze heterogeneous performance exhibited by some popular deep learning software frameworks for visual inference on a resource-constrained hardware platform. Benchmarking of Caffe, OpenCV, TensorFlow, and Caffe2 is performed on the same set of convolutional neural networks in terms of instantaneous throughput, power consumption, memory footprint, and CPU utilization. To understand the resulting dissimilar behavior, we thoroughly examine how the resources in the processor are differently exploited by these frameworks. We demonstrate that a strong correlation exists between hardware events occurring in the processor and inference performance. The proposed hardware-aware analysis aims to find limitations and bottlenecks emerging from the joint interaction of frameworks and networks on a particular CPU-based platform. This provides insight into introducing suitable modifications in both types of components to enhance their global performance. It also facilitates the selection of frameworks and networks among a large diversity of these components available these days for visual understanding.*

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

## 1. INTRODUCTION

Convolutional neural networks (CNNs) based on deep learning (DL) [1] are rapidly replacing classical computer vision algorithms because of their superior accuracy in terms of several tasks such as image classification, object detection, and segmentation. However, this advantage comes at the cost of increasing memory and computational requirements [2], setting a challenge for the implementation of CNN-based vision systems on resource-constrained embedded platforms [3].

The popularity of the DL paradigm for computer vision has prompted the release of several software frameworks for CNN inference. While globally targeting the same functionality, each one is oriented to enhance certain aspects of performance, ease of use, compatibility, etc. In fact, these tools exploit particular optimization libraries to deal with the demanding computational re-

quirements of CNNs. For instance, convolutional and fully-connected layers composing CNNs can be expressed as matrix-matrix or matrix-vector operations, respectively. These operations can be optimally calculated using Basic Linear Algebra Subroutines (BLAS) [4]-[5], which are implemented by several libraries underlying the DL frameworks: ATLAS [6], MKL [7], OpenBLAS [8]-[9], Eigen [10], cuBLAS [11], etc. This diversity of techniques conducted by CNN software tools results in remarkably different inference performance, even running the same network on a particular hardware device.

In this context, most previous works have followed a direct constrained approach to assess this dissimilarity in framework performance. For instance, comparisons based exclusively on throughput among frameworks – including Caffe, TensorFlow, Torch, CNTK, and MXNet – have been reported [12]-[14]. Other works have assessed inference performance on embedded vision systems through high-level metrics [15]-[18]. More customized and specific CNN implementations on CPU-based embedded systems have been published as well [19]-[20]. In contrast, the scope of this paper encompasses DL frameworks that can operate on a wide range of embedded devices.

All of the aforementioned contributions are focused on straightforward benchmarking to evaluate the dissimilar performance exhibited by DL tools. However, such approach, lacking insight into how hardware and software interact, is not suitable to keep up with the rapid evolution of CNN frameworks. New network architectures are reported almost on a daily basis and accurate modeling is required to predict their behavior in a targeted system, thereby facilitating practical deployments. All in all, in this paper we follow a bottom-up approach to explain CNN inference performance through meaningful low-level metrics reflecting how hardware is exploited by DL software frameworks. The identified correlations constitute the first step in our research to develop the aforementioned accurate modeling.

The rest of the paper is organized as follows. Networks, frameworks and the embedded platform under study are introduced in Section 2. We benchmark the performance of these technological DL components in Section 3, pointing out relevant aspects. In Section 4, we qualitatively examine how hardware resources are exploited during inference in order to explain the benchmarking results. Section 5 quantitatively highlights the correlation of these hardware events with performance figures. Finally, critical remarks for framework selection according to application requirements are described in Section 6. Conclusions are drawn in Section 7.

## 2. BENCHMARKING COMPONENTS AND SET-UP

### 2.1. HARDWARE PLATFORM

The selected CPU-based platform for our study is the Raspberry Pi 3 Model B [21] (RPi), an inexpensive embedded platform, which features a Quad Core ARM Cortex-A53 1.2GHz 64-bit CPU [22][23] – including four ARMv8-A processors – on a Broadcom BCM2837 System-on-a-Chip (SoC).

The ARM Cortex-A53 processor includes a two-level memory system. The level 1 (L1) memory system includes, per core processor, separate instruction and data caches (I-cache, D-cache), and a memory management unit (MMU). The MMU includes one translation lookaside buffer (TLB) per core – a cache for instruction and data that translates between virtual and physical addresses. The level 2 (L2) memory system features a unified cache, which is shared between the cores. Specifically, on the Broadcom BCM2837 of the RPi 3B, L1 and L2 comprise 32KB and 512KB, respectively.

For computation acceleration, an advanced single instruction multiple data (SIMD) architecture – also known as NEON technology – is implemented on this SoC. Each ARMv8-A core makes use of 128-bit NEON and 32-bit Vector Floating-Point (VFP) registers to accelerate scalar and vector operations [24]. In addition, instruction caching and dynamic branch prediction are introduced in the ARM architecture to increase overall performance and reduce power consumption.

Regarding the external memory where network weights and working data for inferencing are stored, the RPi features 1GB RAM LPDDR2 900MHz. An attached micro-SD card provides extra non-volatile storage capacity to the system.

### 2.2. SOFTWARE FRAMEWORKS

On this hardware device, we employ a Raspbian v9.4 Linux Kernel v4.14 [25] operating system and build – using a g++ compiler v6.3.0 – the following popular software tools for DL inference in order to evaluate their performance:

- **Caffe** [26] applies image-to-column transformation (im2col) plus General Matrix-Matrix Multiplication (GEMM) to implement convolutions. On RPi's CPU, two Basic Linear Algebra Subprograms (BLAS) can be set as the back-end for GEMM at compilation time, namely OpenBLAS [9] and Atlas [6]. Firstly, we carried out a preliminary test with different configurations in order to identify the best one among (a) ATLAS, (b) OpenBLAS configured by default, and (c) OpenBLAS established for leveraging the four cores. To this end, we run inference over a 6-minute period with a batch size of 1 image and measured average throughput obtained for each Caffe configuration. To find the performance trend, we tested various CNNs, as reported in Table 1. According to these results, OpenBLAS is a BLAS library supported by RPi's CPU and compatible with Caffe, that better leverages the four cores of the ARM Cortex-A53. We will use this configuration in the rest of the paper.

- **TensorFlow** [27] builds a static graph for expressing network computation operations. Once built and optimized, it can be repeatedly executed for image inference. This framework makes use of the Eigen library [10] to generate efficient parallel code for multicore CPUs. We installed pre-built TensorFlow v1.3.0 for RPi [28], which exploits ARM hardware optimizations – NEON and VFP – for computational acceleration.
- **OpenCV** [29] library offers a module for inferencing using pre-trained network files from other frameworks. We loaded network model files in Caffe format. OpenCV version 3.3.1 was compiled to exploit both ARM NEON and VFP optimizations as well.
- **Caffe2** [30] arose as a new lightweight, modular and mobile-oriented framework. Different from Caffe, it uses static graphs for network definition and the Eigen library for matrix calculation. Caffe2 is also optimized for ARM CPUs with NEON.

Single-precision floating-point data format (float32) is used for data storage and computation on all of these frameworks.

**Table 1.** Preliminary benchmark on throughput for selecting the most suitable acceleration library for Caffe.

| | (a) ATLAS | (b) OpenBLAS | (c) OpenBLAS (4) |
|---|---|---|---|
| GoogLeNet [31] | 0.3 fps | 0.4 fps | 0.6 fps |
| ResNet-50 [32] | 0.1 fps | 0.2 fps | 0.3 fps |
| SqueezeNet-v1.1 [33] | 1.2 fps | 1.5 fps | 2.3 fps |
| MobileNet-v1-224 [34] | 0.5 fps | 0.7 fps | 0.9 fps |

### 2.3. CONVOLUTIONAL NEURAL NETWORKS

The inference performance of each software tool was exhaustively assessed for three networks performing 1000-category image classification, namely GoogLeNet [31], ResNet-50 [32], and SqueezeNet [33]. The pre-trained weights of these models are supplied by each framework, each one using a different file format [35]-[43]. Their different architectures are highlighted in Table 2. Note that a tradeoff exists between complexity and accuracy. Although we used publicly available pre-trained models, we run these networks for 1000-category image classification on the ImageNet ILSVRC 2012 validation dataset [44] – without any data augmentation – to check the reported network accuracy.

Given that we aim at real-time applications at the edge where latency is critical, the batch size was set to 1. These networks were assessed using the corresponding Python API of each framework – specifically, we used Python 2.7.13.

**Table 2.** Main parameters defining CNN architectures

| | GoogLeNet | ResNet-50 | SqueezeNet-v1.1 |
|---|---|---|---|
| Repository | [35] - [37] | [38] - [40] | [41] - [43] |
| Top-1 (%) Acc.[1] | 69.2 ± 0.4 | 72.6 ± 0.1 | 58.3 ± 0.0 |
| Top-5 (%) Acc.[1] | 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-C.layers | 1 | 1 | 0 |
| #weights | ~7.0M | ~25.6M | ~1.2M |
| #MACs | ~1.6G | ~3.9G | ~396M |

[1] As each framework provides a different pre-trained model file, specific details on the training parameters must be found in the indicated source repositories. Random initialization of weights leads to small deviations in accuracy even if it is the same architecture but trained on each framework.

## 3. PERFORMANCE ANALYSIS

Firstly, we benchmark CNN models in terms of high-level performance metrics required for meeting application specifications, namely throughput, memory footprint, and power consumption.

However, a preliminary evaluation of CNN inference on this platform evidences that the high computational demand of these networks increases the SoC temperature, which in turn has an impact on instantaneous throughput. To take this aspect into account, we also registered the high-level CPU status (temperature, frequency, and utilization) after each processed image on a long-term period (i.e., 6 minutes) of continuous inference.

Overall, the following performance figures have been considered:

- Throughput. It was calculated as the inverse value of the total per-image processing time –including the time required to read and pre-process the input image, perform the inference, extract the CPU metrics and save the long-term analysis results.
- CPU utilization. It was measured by using the Python psutil library.
- CPU frequency and temperature. We used the vcgencmd tool after each inference to monitor CPU frequency and temperature. Under normal conditions, the ARM Cortex-A53 CPU can run at 1.2 GHz. However, a high SoC temperature will force CPU downclocking.
- Memory footprint. It was taken from the Unique Set Size (USS) metric provided by the psutil library. This value represents effective physical memory allocation for running the process – comprising the required memory for process-

ing one image plus CNN weights plus unshared libraries, including the framework library itself.

- Power consumption. We employed an external Keysight N6705C DC Power Analyzer to measure the instantaneous power demanded by the networks.
- An example of the temporal evolution of CPU status and its instantaneous impact on throughput is shown in Fig. 1. This plot corresponds to a 6-minute period of SqueezeNet inference.
- The average values of performance metrics for all network/framework pairs under study are shown in Fig. 2. These parameters are relevant for real-time applications, and some aspects must be remarked:
- Temperature increases following different patterns, as exemplified in Fig. 1. When the CPU reaches 80°, the processor protects itself by reducing its frequency, which in turn decreases the throughput.

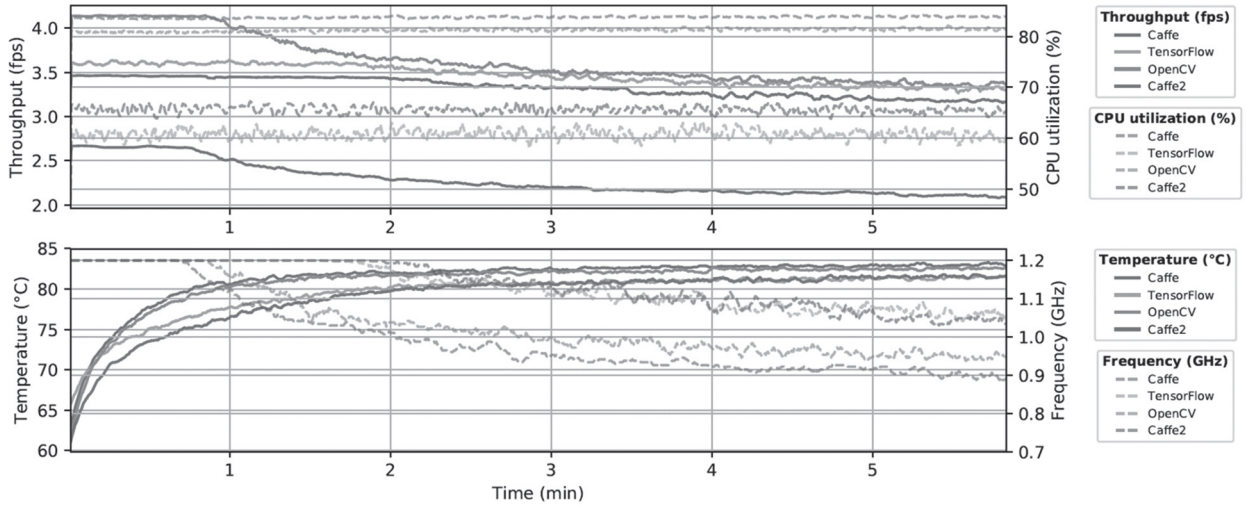
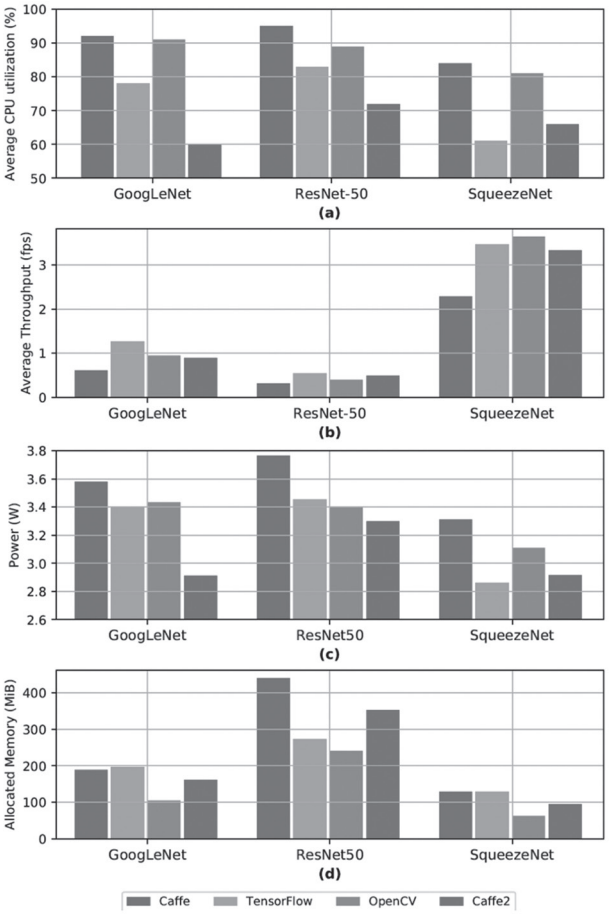



**Fig. 1.** Long-term evaluation of SqueezeNet. Similar trends are observed for GoogLeNet and ResNet-50.




- CPU utilization and allocated memory are quite stable over the test period. However, particular acceleration libraries exploited by the tools make them allocate different amounts of memory – for instance, the well-known tradeoff between memory and computation speed introduced by im2col transformation. In fact, such diversity of coding techniques also explains the differences in CPU utilization and throughput.
- Average throughput differs significantly among frameworks. This has influence on the total number of processed images over the inference period.
- Both framework coding and network architecture affect the instantaneous power consumption.
- Caffe is distinguished for the highest CPU utilization, what quickly increases its temperature and makes the system demand high power. However, in spite of apparently making the most of the CPU, Caffe's throughput is the lowest among the frameworks for these three CNNs.

**Fig. 2.** Average values of (a) CPU utilization, (b) throughput, (c) power consumption, and (d) allocated memory[1].

[1]MiB refers to mebibyte. It is equivalent to 220 bytes.

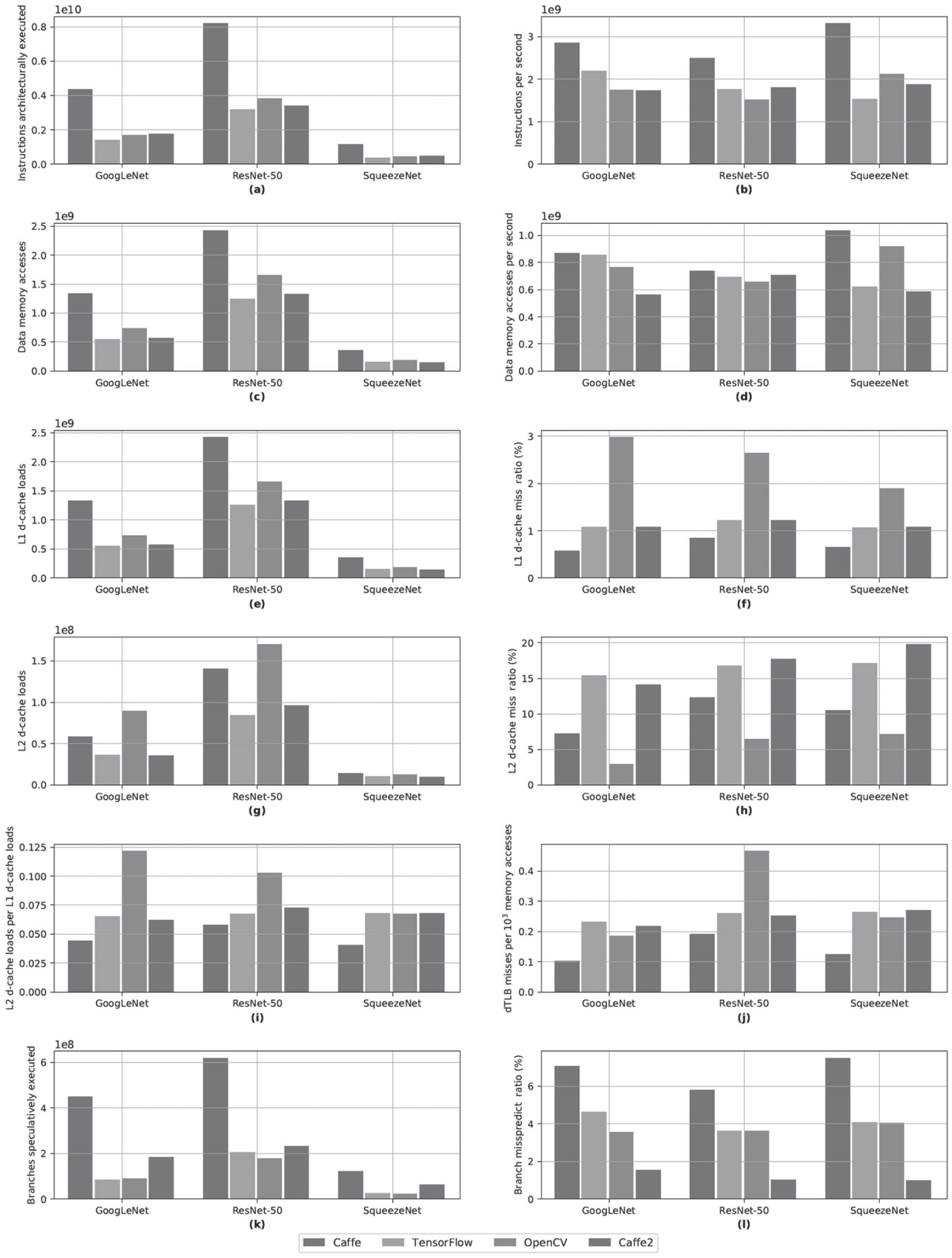



**Fig. 3.** Per-image hardware events registered during inference: (a) instructions architecturally executed, (b) instructions per second, (c) data memory accesses, (d) data memory accesses per second, (e) L1 d-cache loads, (f) L1 d-cache miss ratio, (g) L2 d-cache loads, (h) L2 d-cache miss ratio, (i) L2 d-cache loads per L1 d-cache loads, (j) dTLB misses per $10^3$ memory accesses, (k) branch speculatively executed, and (l) branch mispredict ratio.

Actually, the same behavior has been observed for other CNNs running on this framework, such as Network-in-Network [45] or MobileNet [34].

Thus, we propose an analysis based on hardware exploitation in order to elucidate the underlying reasons for these results.

## 4. HARDWARE EXPLOITATION ANALYSIS

We can extract aggregated statistics on both the processor and the memory system from six event counters provided by the so-called performance monitoring unit (PMU) available in the Cortex-A53 processor. In particular, we employed the perf tool [46] to gather PMU hardware events [47].

For the sake of a fair comparison, we will compare per-image hardware statistics. To this end, we gathered the metrics corresponding to the complete inference script (s1), which runs inference on N=50 images randomly selected from the ImageNet dataset [44]. Then, we singled out the statistics derived from loading the network and libraries (s2) by running the same script but set to N=0. Thus, the per-image statistics we will assess are derived from the expression (s1-s2)/50. Moreover, we averaged values from five measurements in order to reduce perf estimation errors due to multiplexing events [46].

Fig. 3 depicts the most representative parameters among all statistics gathered. Next, we will carefully examine them.

Let us first focus on Caffe as an example of elaborating on performance behavior upon these metrics. Caffe's coding strategies and underlying libraries – OpenBLAS for convolutions – make it demand the highest number of processing instructions (Fig. 3(a)) and data memory accesses (Fig. 3(c)) for the three networks. To deal with these requirements, the processor renders the highest rates of instructions and memory fetches per second (Figs. 3(b) and 3(d)). These rates explain the high CPU utilization of Caffe reported in Section 3 (Fig. 2(a)). In addition, branch prediction implemented on the RPi ARM processor is intensively applied by Caffe (Fig. 3(k)). Under branch prediction, instructions of a branch of code are executed before checking whether they need to be executed. Succeeding in branch prediction speeds up computation. Nonetheless, Caffe's poor prediction performance (Fig. 3(l)) forces the CPU to execute more unnecessary instructions. Concerning cache exploitation, keeping re-used data at a higher level of the memory hierarchy will reduce data access latency. Caffe significantly makes the most of level 1 and level 2 caches, loading high amounts of data (Figs. 3(e) and 3(g)) with low miss rates (Figs. 3(f), 3(h) and 3(i)). The exploitation of TLB by Caffe is also significant (Fig. 3(j)). In fact, the OpenBLAS library underlying this framework is highly oriented to reduction of TLB misses by keeping part of one of the operands in the L1 cache. Definitely, these aggregated hardware metrics gathered for Caffe (appropriate cache exploitation, but the highest demand for processing and memory) suggests unfit coding in this framework when it comes to leveraging RPi's ARM instruction set. This explains poor throughput but high CPU utilization of Caffe on this platform.

Concerning the other three software tools, there are also remarkable results. A distinctive reduction in their number of executed instructions with respect to Caffe is highlighted in Fig. 3(a). This suggests an efficient exploitation of the processor SIMD instruction set – note that these three frameworks allow to leverage ARM hardware optimizations at compile time. TensorFlow coding efficiency is remarkable, as revealed by its lowest number of executed instructions and data movements (Figs. 3(a) and 3(c)), in addition to high operation rates (Figs. 3(b) and 3(d)). This explains high throughput achieved by TensorFlow (Fig. 2(b)). Regarding OpenCV, even making poor use of the highest level of cache, i.e., L1, it is the best by far on exploiting level 2 cache (Figs. 3(e) - 3(h)). This leads to OpenCV low memory requirements (Fig 2(d)) and high frame rates (Fig 2(b)) – note that external memory accesses take several CPU cycles, which OpenCV saves upon its efficient use of L2. The performance of TensorFlow and Caffe2 is similar in relation to cache exploitation. Finally, Caffe2 is remarkable in terms of effective branch prediction.

## 5. CORRELATIONS BASED ON HARDWARE METRICS

Beyond the results and discussion above, further conclusions can be extracted by identifying correlations among the numerical results presented in sections 2, 3, and 4.

### 5.1. CORRELATION ON AGGREGATED HARDWARE EVENTS

To start with, the network architectures can be assessed from a hardware exploitation perspective. For each network under consideration, Fig. 4(a) reveals that the number of instructions executed on the processor (shown in Fig. 3(a)) is clearly consistent with the amount of multiply-accumulate (MAC) operations required for inference (Table 2) – with each framework exhibiting a distinctive relationship, as previously discussed. Similarly, data memory accesses are correlated with the number of parameters learnt in the network (Fig. 4(b)). These correlations allow us to extract a preliminary estimation of the expected hardware resource requirements – executed instructions or memory accesses – simply calculating the number of MAC and weights of the particular network that will eventually run on a specific framework. Indeed, this a priori estimation provides insight into the expected CNN performance in terms of throughput or power consumption, as discussed in sections 4 and 5.2, respectively.

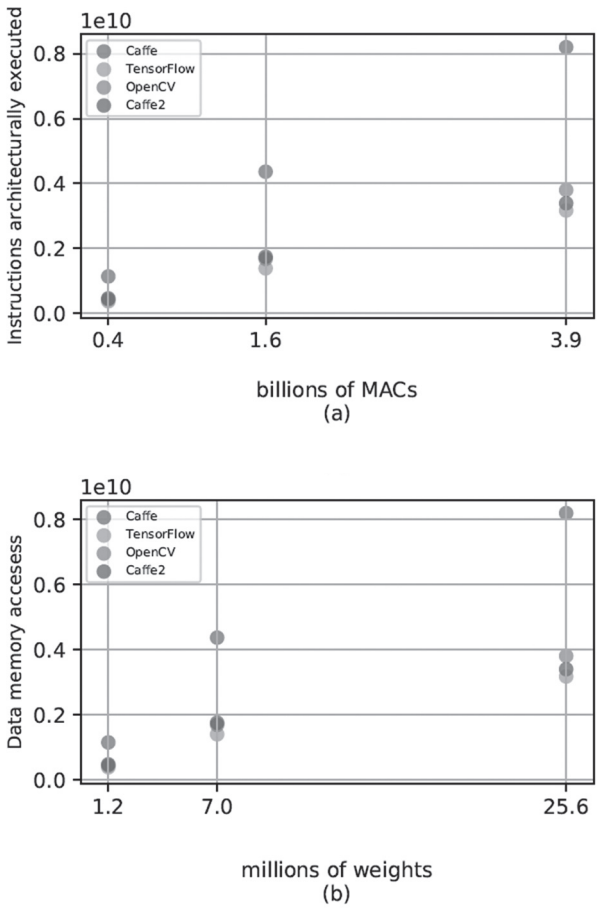



**Fig. 4.** Correlation between hardware statistics (y-axis) and parameters defining the network architecture (x-axis) for the three CNNs.

In addition, the specific acceleration libraries exploited by each framework give rise to a range of memory requirements on the system. Indeed, the amount of memory resources per framework roughly follows a linear pattern, as depicted in Fig. 5. Therefore, for each framework, we can estimate memory requirements from the number of weights in the network. The applicability is straightforward since very deep networks comprise a great deal of weights, and embedded platforms feature limited RAM – 1GB in the case of RPi 3B – to be shared among running edge applications and services, such as sensor management or networking.

All in all, the analysis carried out demonstrates that concerning visual inference applications on resource-constrained embedded platforms, optimizing hardware resources – executed instructions and memory accesses – are mandatory in order to boost performance. To illustrate this, a nearly linear pattern between throughput and data memory accesses is also identified in our platform (Fig. 6), making it possible to obtain good estimates of performance from this aggregated hardware metric.

### 5.2. CORRELATION ON TEMPORAL SAMPLES

In addition to the aggregated statistics analyzed in Section 4, we sampled hardware metrics every 10 milliseconds. Fig. 7 profiles instantaneous power vs. three hardware metrics simultaneously measured – namely L1 and L2 d-cache loads per second, and instructions per second – during four consecutive inferences. To obtain this profiling, thorough temporal alignment was necessary because of different measurement sources employed, i.e., event counters in the processor and the same external power analyzer mentioned in Section 3, that is, Keysight N6705C DC. Plots similar to that in Fig. 7 have been obtained for all analyzed networks and frameworks. The Pearson correlation coefficients of these aligned temporal samples of hardware statistics and power consumption are presented in Table 3. It is worth noting that these coefficients range between 0.54 and 0.95, being greater than 0.80 in most cases.

Taking into account the importance of power consumption in embedded vision applications, 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 model and characterize embedded platforms in terms of energy.

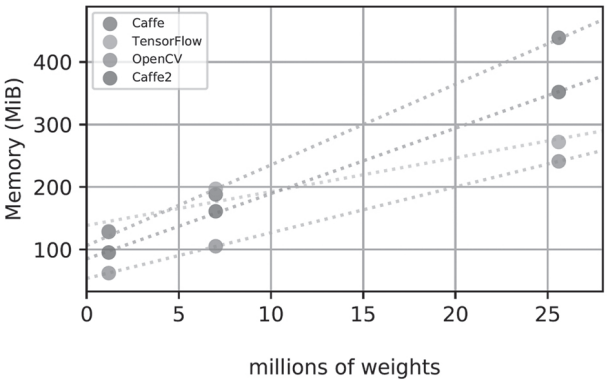



**Fig. 5.** Memory requirements vs. network weights.

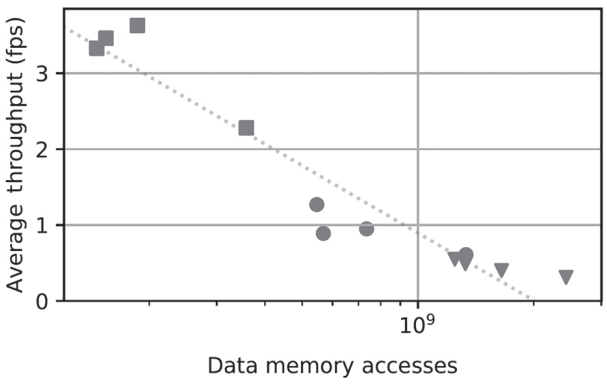



**Fig. 6.** Alignment between throughput and the number of memory-access hardware events for the 12 combinations of assessed frameworks and networks.

Square points represent SqueezeNet, circular points are associated to GoogLeNet, and triangular points refer to ResNet-50.

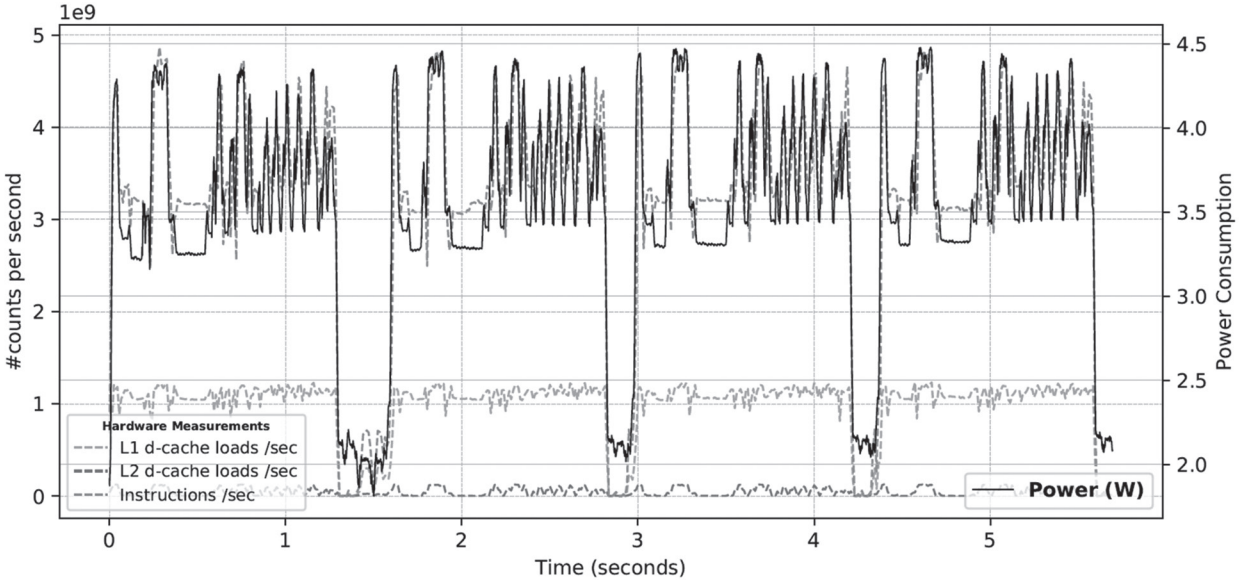



**Fig. 7.** Instantaneous hardware metrics and power consumption on four inferences of GoogLeNet on Caffe. A high correlation can be visually identified.

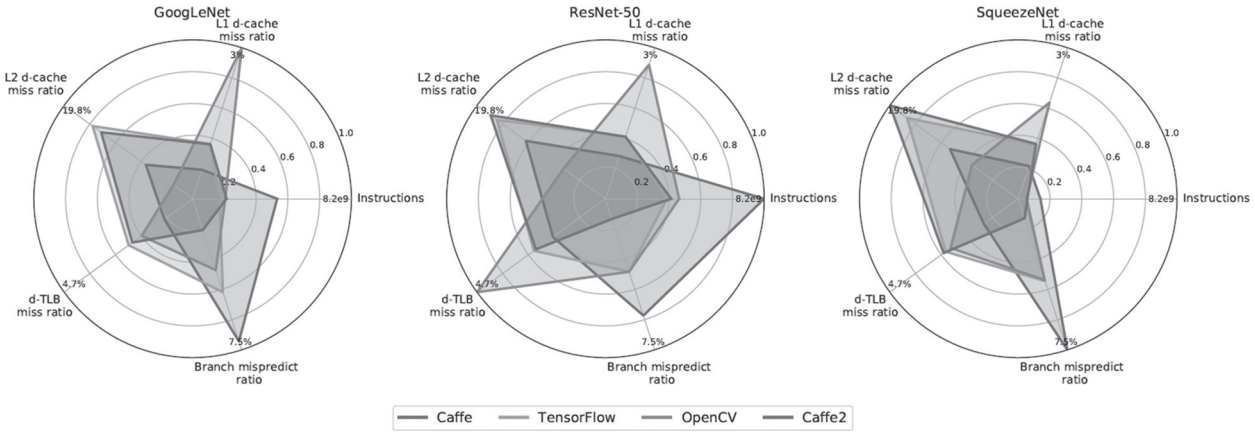



**Fig. 8.** Polar charts with the most representative results of the proposed hardware exploitation analysis .

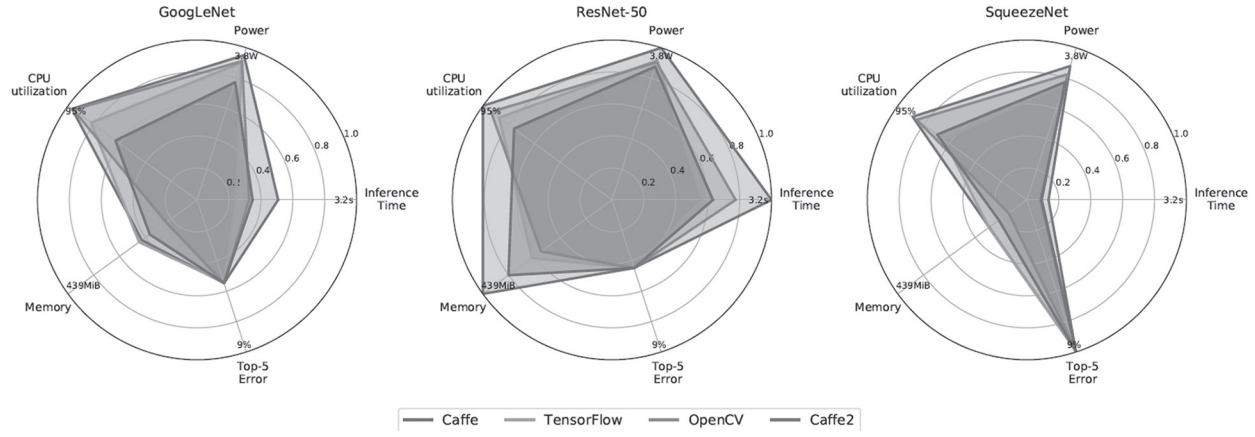



**Fig. 9.** Assistance graphs for optimum framework and network selection.

**Table 3.** Pearson correlation coefficient between instantaneous power consumption and three hardware metrics.

| | | L1 d-cache loads/sec | L2 d-cache loads/sec | Instructions /sec |
|---|---|---|---|---|
| GoogLeNet | 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 |
| ResNet-50 | Caffe | 0.79 | 0.61 | 0.78 |
| | TensorFlow | 0.76 | 0.68 | 0.73 |
| | OpenCV | 0.86 | 0.54 | 0.85 |
| | Caffe2 | 0.67 | 0.73 | 0.61 |
| SqueezeNet | Caffe | 0.94 | 0.82 | 0.95 |
| | TensorFlow | 0.80 | 0.80 | 0.80 |
| | OpenCV | 0.94 | 0.70 | 0.94 |
| | Caffe2 | 0.86 | 0.81 | 0.86 |

## 6. APPLICABILITY OF THE STUDY: NETWORK/ FRAMEWORK SELECTION

The extracted hardware information can be wisely used to facilitate the selection of optimum DL components according to application specifications. To simplify this task, Fig. 8 provides insight into a quick visual comparison of hardware exploitation on the benchmarked components. Five hardware metrics have been particularly selected and compared in these charts. For each metric, represented values have been normalized with respect to the maximum measured value (indicated on the external circumference) and there is a scale factor along the radial axis. Thus, for example, the branch mispredict ratio for GoogLeNet/TensorFlow (Fig. 8, left chart, orange line) is around 0.6 times 7.5%, which is the maximum measured for this metric. Therefore, the value of the metric at that point is 4.5%. Fig. 8 immediately suggests the bottlenecks of each framework on this CPU-based platform: the higher the values of these metrics, the worse the performance we must expect. For instance, Caffe is defined for executing more instructions (aggravated with more branch mispredictions) in the three network cases. On the other hand, TensorFlow, OpenCV and Caffe2 stand out in terms of requiring fewer instructions, low L2 cache misses, and reduced branch mispredict rates, respectively.

Finally, another graphical comparison, in this case in terms of high-level performance metrics, is presented in Fig. 9. Once again, the values have been normalized with respect to the measured maxima. It is straightforward to match low latency with TensorFlow, minimum memory allocation with OpenCV, and reduced power consumption with Caffe2. Comparing the networks globally, note that low latency, memory footprint, and power consumption of SqueezeNet (Fig. 9, right chart) are traded off with a reduced inference accuracy.

## 7. CONCLUSIONS

This paper demonstrates that low-level hardware exploitation parameters can be effectively used to model the expected behavior of CNNs running on DL frameworks built on a CPU-based embedded platform. The identified correlations between such parameters and performance metrics allow us to highlight bottlenecks and limitations in the interaction between hardware and software. This is critical to selecting the most suitable components according to prescribed specifications. The proposed graphical representations prove the validity of these considerations in practical terms. In the near future, we will be reporting the analytical models for accurate performance prediction we are developing upon the results presented herein.


### ACKNOWLEDGMENT

The current archival periodical article is based on the conference presentation [48].

This work was supported by the Spanish Government MICINN (European Region Development Fund, ERDF/ FEDER) through project RTI2018-097088-B-C31, by the Spanish Government through FPU Grant FPU17/02804, by the European Union H2020 MSCA through project ACHIEVE-ITN (Grant No. 765866), and by the US Office of Naval Research through Grant No. N00014-19-1-2156.


## 8. REFERENCES:


[1] Y. LeCun, Y. Bengio, G. Hinton, "Deep Learning", Nature, Vol. 521, No. 7553, 2015, pp. 436–444.

[2] M. Verhelst and B. Moons, "Embedded Deep Neural Network Processing", IEEE Solid-State Circuits Magazine, Vol. 9, No. 4, 2017, pp. 55–65.

[3] V. Sze, "Designing Hardware for Machine Learning", IEEE Solid-State Circuits Magazine, Vol. 9, No. 4, 2017, pp. 46–54.

[4] C. L. Lawson, R. J. Hanson, D. R. Kincaid, F. T. Krogh, "Basic Linear Algebra Subprograms for FORTRAN Usage", ACM Transactions Mathematical Software, Vol. 5, No. 3, 1979, pp. 308–323.

[5] T. Kielmann, Basic Linear Algebra Subprograms (BLAS), 2011.

[6] Automatically Tuned Linear Algebra Software (ATLAS), http://math-atlas.sourceforge.net (accessed: 2019)

[7] Intel Math Kernel Library, https://software.intel.com/en-us/mkl (accessed: 2019)

[8] K. Goto, R. A. van de Gejin, "Anatomy of high-performance matrix multiplication", ACM Transac-

tions Mathematical Software, Vol. 34, No. 3, 2008, pp. 12:1–12:25.

[9] OpenBLAS, optimized BLAS library based on GotoBLAS2 1.13 BSD version, https://github.com/xianyi/OpenBLAS (accessed: 2018)

[10] Eigen, http://eigen.tuxfamily.org/ (accessed: 2019)

[11] Dense Linear Algebra on GPUs, https://developer.nvidia.com/cublas (accessed: 2019)

[12] S. Bahrampour, N. Ramakrishnan, L. Schott, M. Shah, "Comparative Study of Caffe, Neon, Theano, and Torch for Deep Learning", arXiv, No. 1511.06435, 2015.

[13] J. Hanhirova et al., "Latency and Throughput Characterization of Convolutional Neural Networks for Mobile Computer Vision", arXiv, No. 1803.09492, 2018.

[14] S Shi, Q. Wang, P. Xu, X. Chu, "Benchmarking State-of-the-Art Deep Learning Software Tools", arXiv, No. 1608.07249, 2016.

[15] A. Ignatov et al. "AI Benchmark: Running Deep Neural Networks on Android Smartphones", arXiv, No. abs/1810.01109, 2018.

[16] D. Velasco-Montero, J. Fernandez-Berni, R. Carmona-Galan, A. Rodriquez-Vazquez, "Optimum Selection of DNN Model and Framework for Edge Inference", IEEE Access, Vol. 6, 2018, pp. 51680–51692.

[17] X. Zhang, Y. Wang, W. Shi, "pCAMP: Performance Comparison of Machine Learning Packages on the Edges", USENIX Workshop on Hot Topics in Edge Computing, Boston, MA, USA, 10.7.2018.

[18] D. Pena, A. Forembski, X. Xu, D. Moloney, "Benchmarking of CNNs for Low-Cost, Low-Power Robotics Applications", RSS Workshop: New Frontier for Deep Learning in Robotics, Boston, MA, USA, 15.7.2017.

[19] S.-J. Lee, S.-S. Park, K.-S. Chung, "Efficient SIMD implementation for accelerating convolutional neural network", Proceedings of the 4th International Conference on Communication and Information, Qingdao, China, 2-4.11.2018, pp. 174–179.

[20] L. Liangzhen, N. Suda, V. Chandra, "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs", arXiv, No. 1801.06601, 2018.

[21] Raspberry Pi 3 Model B, https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (accessed: 2019)

[22] ARM, ARM Cortex-A53 MPCore Processor, Technical Reference Manual, https://developer.arm.com/docs/ddi0500/g (accessed: 2019)

[23] ARM Processors. Cortex-A53, https://developer.arm.com/products/processors/cortex-a/cortex-a53 (accessed: 2019)

[24] ARM, ARM Cortex-A53 MPCore Processor Advanced SIMD and Floating-point Extension. Technical Reference Manual.

[25] Raspbian, https://www.raspberrypi.org/downloads/raspbian/ (accessed: 2018)

[26] Y. Jia 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," Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, GA, USA, 2-4.11.2016, pp. 265–283.

[28] A Docker image for Tensorflow, https://github.com/DeftWork/rpi-tensorflow (accessed: 2018)

[29] OpenCV, https://opencv.org/ (accessed: 2018)

[30] Caffe2, https://caffe2.ai/ (accessed: 2018)

[31] C. Szegedy et al., "Going deeper with convolutions", arXiv, No. 1409.4842, 2014.

[32] H. Kaiming, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition", arXiv, No. 1512.03385, 2015.

[33] F. Iandola et al., "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1MB model size", arXiv, No. 1602.07360, 2016.

[34] A. Howard, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", arXiv, No. 1704.04861, 2017.

[35] BAIR/BVLC GoogLeNet Model, https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet (accessed: 2018)

[36] Inception V1, http://download.tensorflow.org/models/inception_v1_2016_08_28.tar.gz (accessed: 2018)

[37] Caffe2 Models, BVLC GoogLeNet, https://github.com/caffe2/models/tree/master/bvlc_googlenet (accessed: 2018)

[38] Deep Residual Learning for Image Recognition, https://github.com/KaimingHe/deep-residual-networks (accessed: 2018)

[39] ResNet V1 50, http://download.tensorflow.org/models/resnet_v1_50_2016_08_28.tar.gz (accessed: 2018)

[40] Caffe2 Models, ResNet50, https://github.com/caffe2/models/tree/master/resnet50 (accessed: 2018)

[41] SqueezeNet v1.1, https://github.com/DeepScale/SqueezeNet/tree/master/SqueezeNet_v1.1 (accessed: 2018)

[42] Caffe to TensorFlow, https://github.com/ethereon/caffe-tensorflow (accessed: 2018)

[43] Caffe2 Models, SqueezeNet, https://github.com/caffe2/models/tree/master/squeezenet (accessed: 2018)

[44] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge", International Journal of Computer Vision, Vol. 115, No. 3, 2015, pp. 211–252.

[45] M. Lin, Q. Chen, S. Yan, "Network in network," arXiv, No. 1312.4400, 2013.

[46] perf: Linux profiling with performance counters, https://perf.wiki.kernel.org/index.php/Main_Page (accessed: 2019)

[47] ARM, ARM Architecture Reference Manual. ARMv8, for ARMv8-A architecture profile, 2017.

[48] D. Velasco-Montero, J. Fernandez-Bemi, R. Carmona-Galan, A. Rodriquez-Vazquez, "On the Correlation of CNN Performance and Hardware Metrics for Visual Inference on a Low-Cost CPU-based Platform", Proceedings of the 26th IEEE International Conference on Systems, Signals and Image Processing, Osijek, Croatia, 5-7.6.2019, pp. 249–254.