Optimum Network/Framework Selection from High-Level Specifications in Embedded Deep Learning Vision Applications

Delia Velasco-Montero[1], Jorge Fernández-Berni[1], Ricardo Carmona-Galán[1], and Ángel Rodríguez-Vázquez[1]

Instituto de Microelectrónica de Sevilla (Universidad de Sevilla-CSIC), Sevilla, Spain

Abstract. This paper benchmarks 16 combinations of popular Deep Neural Networks for 1000-category image recognition and Deep Learning frameworks on an embedded platform. A Figure of Merit based on high-level specifications is introduced. By sweeping the relative weight of accuracy, throughput and power consumption on global performance, we demonstrate that only a reduced set of the analyzed combinations must actually be considered for real deployment. We also report the optimum network/framework selection for all possible application scenarios defined in those terms, i.e. weighted balance of the aforementioned parameters. Our approach can be extended to other networks, frameworks and performance parameters, thus supporting system-level design decisions in the ever-changing ecosystem of Deep Learning technology.

Keywords: Deep Learning, Convolutional Neural Networks, Embedded Vision, Performance, High-Level Specifications

1 Introduction

The number of software frameworks, acceleration libraries, network architectures, hardware modules etc. related to the Deep Learning (DL) paradigm [6] increases virtually on a daily basis. Much better accuracy and a common approach for many different inference tasks are the primary reasons for the success of this paradigm.

In particular, concerning vision, Convolutional Neural Networks (CNNs) have become the core processing architecture underpinning most of the advances in the field [15]. As a result, the spectrum of DL technological components aiming at speeding up CNNs is vast. Each of these components comes along with claims about enhancements in certain aspects of performance, ease of use, compatibility etc. While this variety in principle covers a wide spectrum of specifications, there are no guidelines for system designers to perform an optimal selection meeting prescribed application-level requirements. This is especially an issue in embedded platforms due to the heavy computational demands of CNNs [11]. In these cases, the adoption of non-optimal components could lead to the apparent impossibility of fulfilling some of those prescribed requirements.

In this paper, we analyze up to 16 combinations of popular network models and software frameworks from a practical point of view. We have implemented all of them in a Raspberry Pi 3 Model B, extracting three key performance parameters, namely accuracy, throughput and power consumption. But we do not assess them in a plain way. We propose a Figure of Merit (FoM) encoding the relative weight of each parameter according to specific high-level requirements. The results reveal that only 5 out of the 16 combinations perform the best in at least one point of the exploration domain. We also report the worst selection over all possible performance balances.

2 Baseline performance figures

Our starting point for this paper is the analysis reported in [14]. The embedded hardware platform picked for benchmarking is the Raspberry Pi (RPi) 3 Model B. It is inexpensive but still features enough computational power for real-time DL inference. Its Broadcom BCM2837 System On a Chip (SoC) comprises a Quad Core ARM Cortex-A53 1.2GHz 64-bit CPU that can work at frequencies ranging from 700 MHz up to 1.2 GHz. It also features 1GB RAM LPDDR2 900MHz.

Concerning software tools, we chose 4 open-source frameworks widely adopted these days for DL, namely Caffe, TensorFlow, OpenCV and Caffe2. Each of them relies on particular encoding techniques and libraries for optimization of both training and inference. *Caffe* [5] is a DL software tool developed by the Berkeley Vision and Learning Center (BVLC) with the help of a large community of contributors on GitHub. It requires Basic Linear Algebra Subprograms (BLAS) as the back-end for matrix and vector computation. We compiled Caffe with OpenBLAS [8], an optimized implementation of BLAS supported by ARM processors. This library accelerates algebraic operations on CPU leveraging the 4 ARM cores of the RPi. *TensorFlow* [1] expresses computations as stateful dataflow graphs that manage tensors. This enables various optimizations, e.g. the elimination of redundant copies of the same operation in the graph or the implementation of careful operation scheduling that improves data transfer performance and memory usage. We installed pre-built TensorFlow 1.3.0 for RPi [13]. This version exploits ARM hardware optimizations for computational acceleration, namely ARM Neon and VFPv4 (Vector Floating Point Unit). ARM Neon supports Single Instruction Multiple Data (SIMD) instructions for exploiting data-level parallelism whereas VFPv4 provides high-efficiency floating-point operations. Concerning *OpenCV* [9] computer vision library, its DNN module enables the use of pre-trained models for inference from other frameworks such as Caffe, Torch or TensorFlow. OpenCV version 3.3.1 was compiled to exploit both ARM Neon and VFPv3 optimizations as well. Finally, *Caffe2* [2] also uses computation graphs for network definition. The building process of Caffe2 on the RPi admits ARM Neon optimizations too.

All of these software tools allow the deployment of state-of-the-art pre-trained models of Deep Neural Networks (DNNs) for inference. In particular, we

Table 1. Performance figures for the selected DL frameworks and DNN models running on a Raspberry Pi 3 Model B. Each triplet per entry corresponds to measured metrics: [Top-5 Accuracy (%), Throughput (fps), Power Consumption (W)]

	Network in Network	**GoogLeNet**	**SqueezeNet v1.1**	**MobileNet v1 1.0-224**
Caffe	[79.23, 2.08, 3.75]	[89.02, 0.80, 3.81]	[80.81, 3.11, 3.58]	[89.00, 1.14, 3.50]
TensorFlow	[79.23, 2.58, 3.49]	[89.53, 1.74, 3.58]	[80.81, 4.22, 2.95]	[89.67, 2.41, 3.26]
OpenCV	[79.23, 1.42, 3.08]	[89.02, 1.28, 3.62]	[80.81, 4.79, 3.20]	[89.00, 2.56, 3.23]
Caffe2	[79.23, 2.55, 3.44]	[89.02, 0.92, 3.00]	[79.92, 3.74, 3.11]	[89.00, 1.80, 2.88]

benchmarked 4 well-known models trained for 1000-category image recognition, namely *Network in Network* [7], *GoogLeNet* [12], *SqueezeNet* [4] and *MobileNet* [3]. Although their architectures differ from each other, all of them contain layers performing point-wise convolutions – 1x1 filters operating across tensors – to reduce the number of model parameters. Other techniques aiming at lightening the computational load are the inclusion of Global Average Pooling layers before Fully-Connected (FC) layers – GoogLeNet –, the omission of FC layers – Network in Network and SqueezeNet – or the channel cutback and image resolution shrinking – MobileNet.

For each inference scenario – i.e. for each combination of DNN model on DL framework within the aforementioned set, running on the RPi – we have averaged the values of throughput and power consumption when processing a number of images extracted from the ImageNet dataset [10]. We have also measured the accuracy of the models over the 50k validation images of this dataset. The results are summarized in Table 1. Each triplet per entry corresponds to the benchmarked performance parameters, that is, [Top-5 Accuracy (%), Throughput (fps), Power Consumption (W)].

3 Weighted FoM

In [14], we defined the following FoM for the sake of merging the three measured performance parameters in a meaningful way:

$$\text{FoM} := \text{Accuracy} \cdot \frac{\text{Throughput}}{\text{Power}} \tag{1}$$

Note that this FoM can be expressed as 'number of correctly inferred images per joule', providing insight about the joint computational and energy efficiency of models and frameworks on the considered hardware platform. The results in [14] clearly show that SqueezeNet on OpenCV performs the best according to Eq. (1). The superiority of this combination arises from achieving a notable frame rate in a power-efficient way while trading off some accuracy in comparison with more precise networks like GoogLeNet or MobileNet. The question is: would SqueezeNet-OpenCV still be the best choice for a particular application where accuracy is more important, in relative terms, than throughput and/or power consumption?

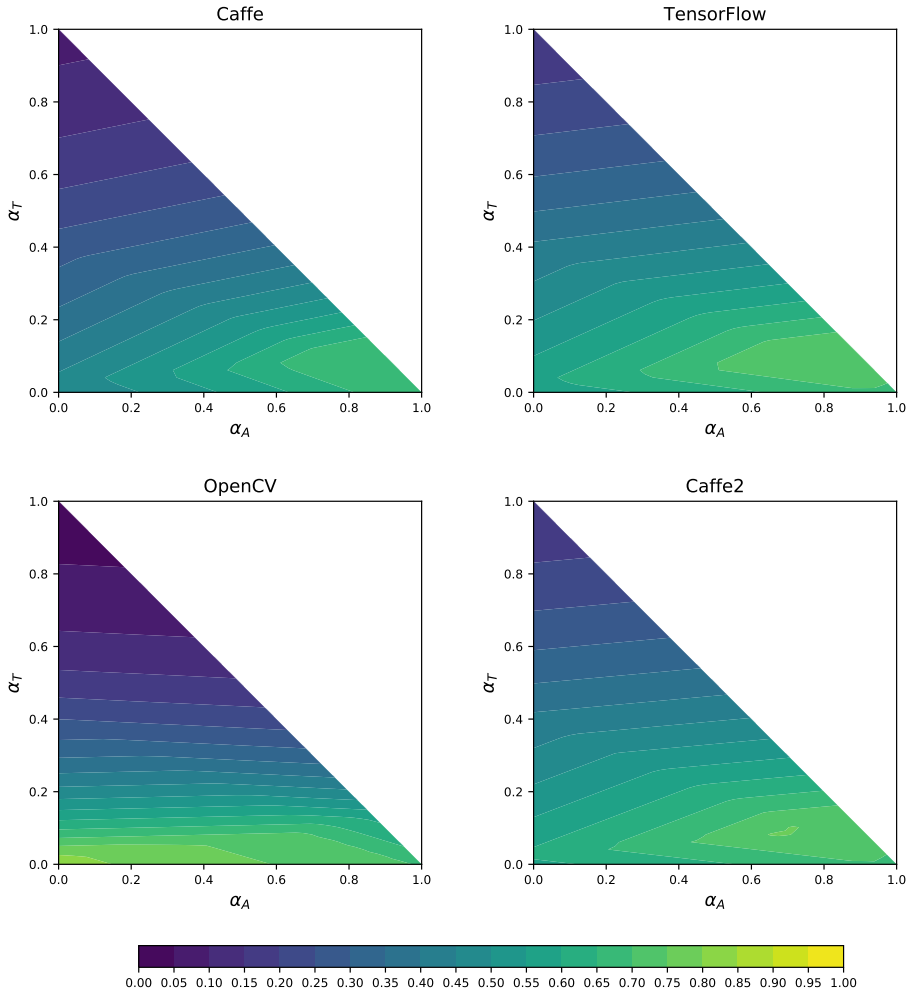

Fig. 1. Normalized weighted FoM for Network in Network model.

In order to answer this question, we propose a re-definition of the FoM in such a way that the relative weight of each performance parameter on high-level specifications can be taken into account. Let $[\alpha_\mathrm{A}, \alpha_\mathrm{T}, \alpha_\mathrm{P}]$ be the relative importance of the metrics [Accuracy, Throughput, Power] respectively, satisfying:

$$\sum_i \alpha_i = 1, \alpha_i \in [0, 1] \tag{2}$$

where i corresponds to $\mathrm{A}, \mathrm{T}, \mathrm{P}$. It follows from Eq. (2) that, for instance, a vector $[\alpha_\mathrm{A}, \alpha_\mathrm{T}, \alpha_\mathrm{P}] = [0.70, 0.15, 0.15]$ defines an application scenario where accu-

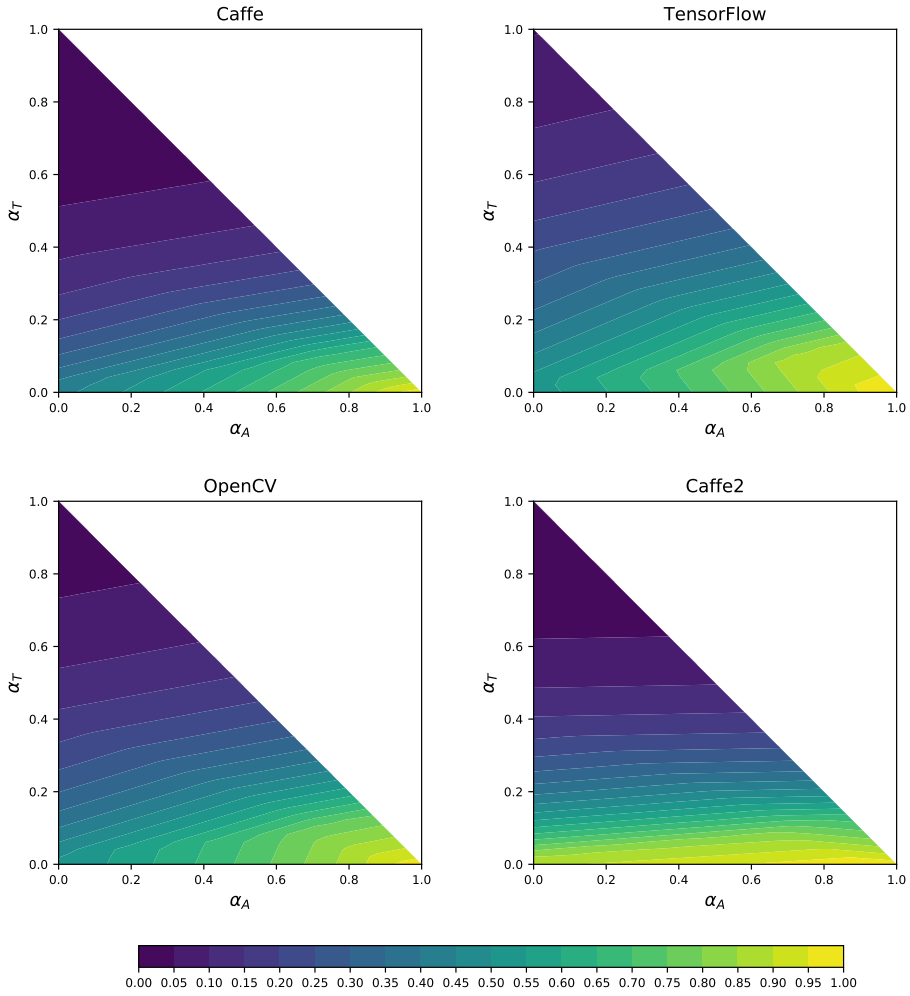

Fig. 2. Normalized weighted FoM for GoogLeNet model.

racy presents a relative preeminence of 70% on the targeted performance, being throughput and power consumption equally important with a weight of 15%.

A new weighted FoM can now be defined as:

$$\text{FoM}(\alpha_\text{A}, \alpha_\text{T}, \alpha_\text{P}) := \text{Accuracy}^{3\alpha_\text{A}} \cdot \frac{\text{Throughput}^{3\alpha_\text{T}}}{\text{Power}^{3\alpha_\text{P}}} \tag{3}$$

Hence, Eq. (1) is just a particular case of the generalized FoM in Eq. (3): when $\alpha_i = 1/3\ \forall i$, we are assuming that the three performance parameters have the same impact on application-level requirements. Any other distribution of α_i

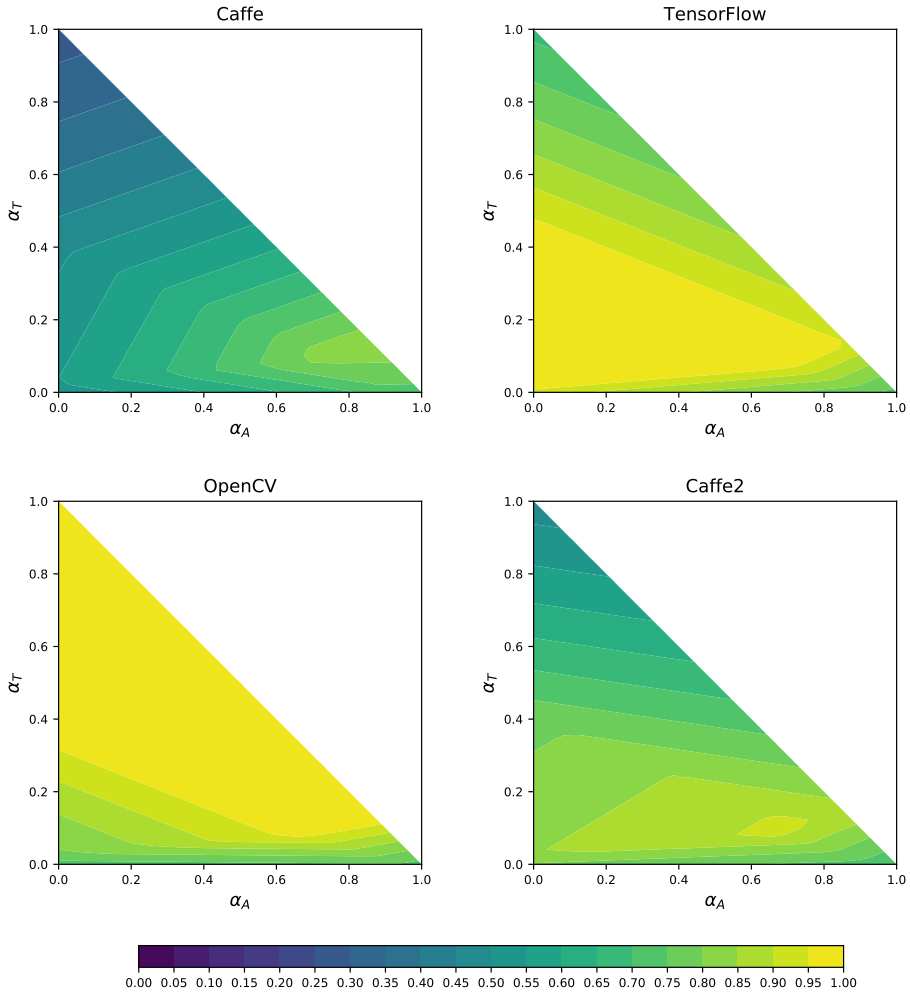

Fig. 3. Normalized weighted FoM for SqueezeNet model.

assigns a specific weight to their corresponding factor in Eq. (3) by setting its exponent to a value greater or less than unity. Extreme cases occur when any α_i is set to 1, boosting the influence of its associated performance parameter on the FoM at the cost of completely dismissing the other ones.

By introducing all the possible combinations of α_i in Eq. (3) together with the triplets in Table 1, we can find out the optimum network/framework choice per application scenario defined in terms of weighted balance of accuracy, throughput and power consumption. For each scenario – i.e. for each combination of α_i – we normalize the resulting FoM with respect to its maximum value obtained for

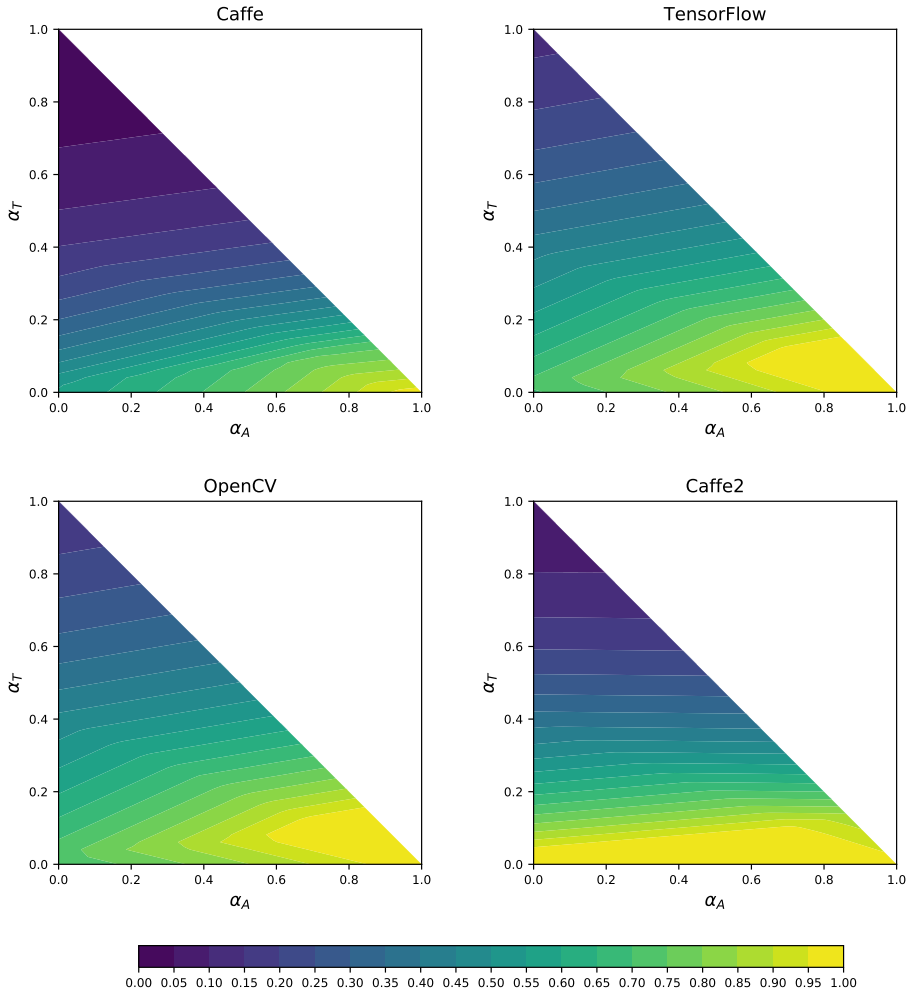

Fig. 4. Normalized weighted FoM for MobileNet model.

the optimum network/framework pair. This permits to quickly identify the best selection with a FoM equal to 1. Likewise, it allows to easily assess how far each pair is from the optimum point.

All in all, Figs. 1, 2, 3 and 4 depict the normalized values of the proposed weighted FoM for each DNN model per software framework. Note that we only consider α_{A} and α_{T} since, according to Eq. (2), once their values are established, α_{P} is automatically determined from $\alpha_{\mathrm{P}} = 1 - \alpha_{\mathrm{A}} - \alpha_{\mathrm{T}}$. In these plots, the maximum is represented in yellow. Points getting far from this maximum turn green and eventually blue for the lowest values of the weighted FoM. A quick look

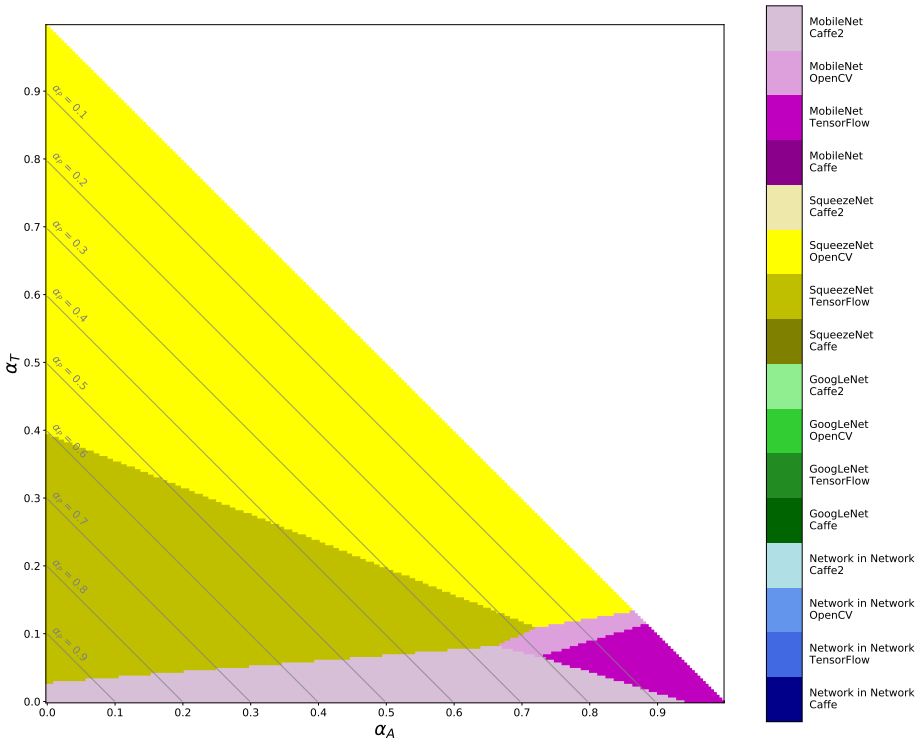

Fig. 5. Optimal network/framework selection according to the weighted FoM defined in Eq. (3). Only a reduced set of the analyzed combinations performs the best in at least one point of the exploration domain.

permits to conclude that SqueezeNet-OpenCV keeps being the best choice for most application scenarios. This is confirmed by Fig. 5 where we represent the optimum selection, i.e. the network/framework pair reaching the maximum FoM, for the whole domain of α_i. Finally, Fig. 6 shows which pairs perform the worst in this domain, producing the lowest values of the weighted FoM. These lowest values are depicted in Fig. 7, highlighting the fact that some network/framework combinations can be really far from optimum performance.

4 Discussion

The first significant outcome from these results is that SqueezeNet is the most suitable network for the majority of the explored performance weightings. For application scenarios where throughput is critical, SqueezeNet must be implemented on OpenCV. When energy efficiency gains relevance, TensorFlow makes the most of this network. For relaxed frame rate requirements, MobileNet performs the best on combining high accuracy and reduced power consumption. In

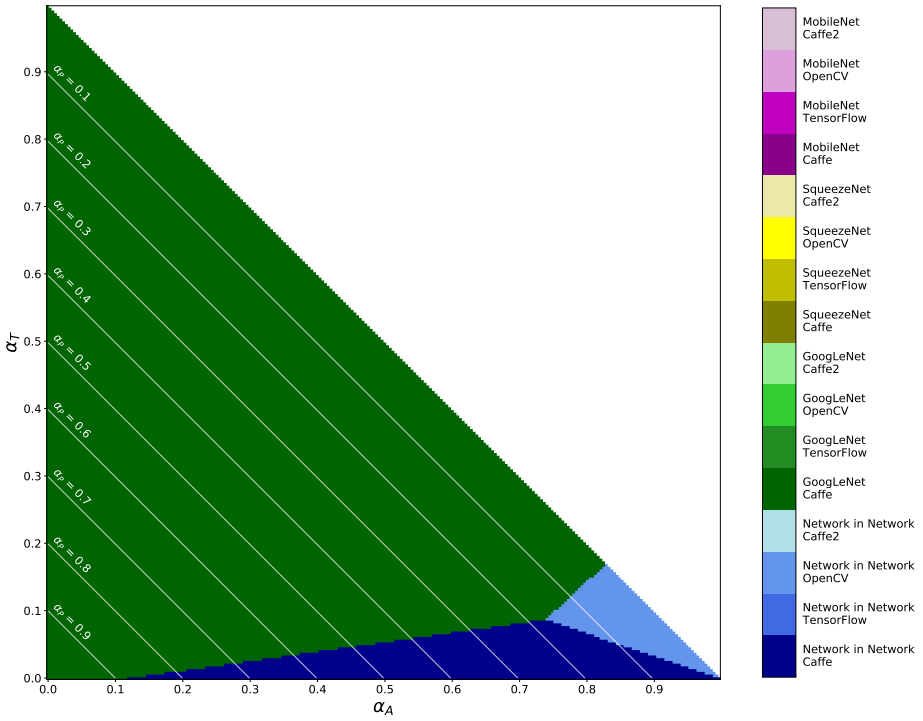

Fig. 6. Worst network/framework selection according to the weighted FoM. The normalized values of the FoM for each point in this plot are represented in Fig. 7.

fact, the lowest consumption – 2.88W – is attained when running this network on Caffe2 – both model and framework have been designed for mobile applications. When power loses importance, keeping relaxed constraints on throughput, MobileNet on TensorFlow is the best choice. In practical terms, we have narrowed down the original list of 16 network/framework pairs to 5 combinations to be selected.

Note that neither Network in Network nor GoogLeNet ever reach a maximum FoM, no matter the software framework considered. On the contrary, when running on OpenCV or Caffe, Network in Network performs the worst because of its lowest accuracy at moderate frame rate and high power consumption. Nevertheless, the combination covering most of the area in Fig. 6 is GoogLeNet on Caffe. The high accuracy from this network do not make up for their joint poorest performance in terms of throughput and power consumption. In particular, throughput is notably lower than the best one – 0.80fps vs. 4.79fps for SqueezeNet-OpenCV. This is why the normalized FoM in Fig. 7 degrades for increasing values of α_{T}.

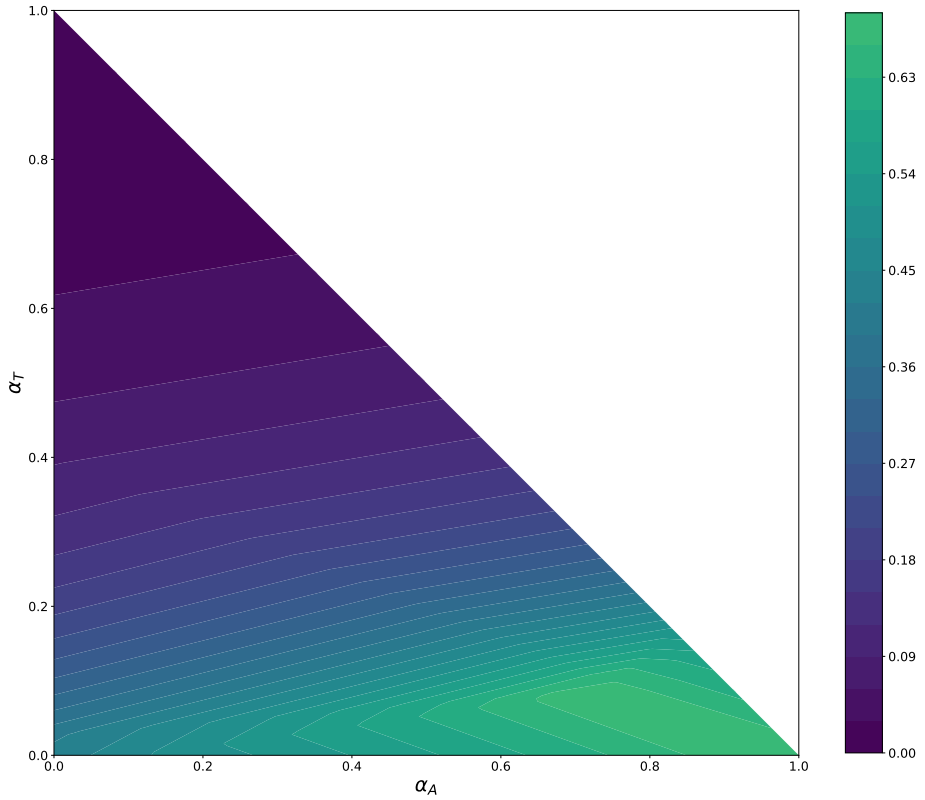

Fig. 7. Lowest values of the normalized FoM. They correspond to the network/framework pairs represented in Fig. 6.

We must emphasize that this comparative study is strictly quantitative in the sense that our discussion is exclusively focused on measurable performance. Thus, the FoM defined in Eq. (3) does not include any qualitative parameters encoding the suitability of a particular model, framework or hardware platform according to, for instance, a long-term technological strategy in a company or a product development roadmap. However, taking into account the current lack of standardization in the field of DL-based vision, and its rapid evolution pace, qualitative considerations should not have too much influence yet on the final decision to be made. The conclusions drawn in this manuscript should not therefore change significantly when such considerations were included in the analysis.

5 Conclusions

We have described a pragmatic approach to support system designers when making decisions in the vast ecosystem of DL components for embedded vision.

The merit of a particular realization must be assessed according to the weighted relevance of key operation parameters on application-level specifications. The FoM proposed in this paper aims at facilitating this evaluation. When applied for 1000-category image recognition on a low-cost embedded platform, it has proved to be effective on filtering out most of the studied alternatives. Specifically, we have demonstrated that SqueezeNet-OpenCV presently constitutes the best choice to carry out this task in most application scenarios. In the future, we plan to extend our analysis to more hardware platforms, DNN models and software frameworks, also incorporating other essential performance parameters like memory.

Acknowledgments

This work was supported by Spanish Government MINECO (European Region Development Fund, ERDF/FEDER) through Project TEC2015-66878-C3-1-R, by Junta de Andalucía CEICE through Project TIC 2338-2013 and by EU H2020 MSCA ACHIEVE-ITN, Grant No 765866.

References

1. Abadi, M. et al.: TensorFlow: A System for Large-Scale Machine Learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). pp. 265–283 (2016)
2. Caffe2: https://caffe2.ai/, (Accessed: July 2018)
3. Howard, A. et al.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv (1704.04861) (2017)
4. Iandola, F. et al.: SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <1MB Model Size. arXiv (1602.07360) (2016)
5. Jia, Y. et al.: Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv (1408.5093) (2014)
6. LeCun, Y., Bengio, Y., Hinton, G.: Deep Learning. Nature 521(7553), 436–444 (2015)
7. Lin, M., Chen, Q., Yan, S.: Network In Network. arXiv (1312.4400) (2013)
8. OpenBLAS, optimized BLAS library based on GotoBLAS2 1.13 BSD version: https://github.com/xianyi/OpenBLAS, (Accessed: July 2018)
9. OpenCV: https://opencv.org/, (Accessed: July 2018)
10. Russakovsky, O. et al.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115(3), 211–252 (2015)
11. Sze, V. et al.: Efficient Processing of Deep Neural Networks: A Tutorial and Survey. Proceedings of the IEEE 105(12), 2295–2329 (2017)
12. Szegedy, C. et al.: Going Deeper with Convolutions. arXiv (1409.4842) (2014)
13. TensorFlow for RPi: A Docker image for Tensorflow. https://github.com/DeftWork/rpi-tensorflow, (Accessed: July 2018)
14. Velasco-Montero, D. et al.: Performance Analysis of Real-Time DNN Inference on Raspberry Pi. In: SPIE Commercial + Scientific Sensing and Imaging (April 2018)
15. Verhelst, M., Moons, B.: Embedded Deep Neural Network Processing. IEEE Solid-State Circuits Magazine 9(4), 55–65 (2017)