CNN Inference Costs Estimation on Microcontrollers: the EST Primitive-based Model

Thomas Garbay, Petr Dobias, Wilfried Dron, Pedro Lusich, Imane Khalis, Andrea Pinna, Khalil Hachicha, Bertrand Granado

CNN Inference Costs Estimation on Microcontrollers: the EST Primitive-based Model

Thomas Garbay*, **Petr Dobias***, **Wilfried Dron**†, **Pedro Lusich**†, **Imane Khalis**†,
Andrea Pinna*, **Khalil Hachicha***, **Bertrand Granado***
*Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
†Wisebatt, Paris, France
Correspondence to:
{thomas.garbay, bertrand.granado}@sorbonne-universite.fr
{wilfried.dron}@wisebatt.com

Abstract—**Neural network inference on embedded devices will have an important industrial impact on our society. Embedded devices are ubiquitous in many fields, like human activity recognition or visual object detection. As a matter of fact, Convolutional Neural Networks (CNNs) are now the best modality to solve most of computer vision problems. Although, the accuracy offered by these algorithms has a cost: an important energy consumption, a high execution time, and a significant memory footprint. This cost is a major challenge to implement CNNs within embedded devices with limited computational power, memory space and energy available. This makes prior estimation about the impact of a CNN on a given microcontroller, a design key point before applying neural network compression techniques. We introduce the EST primitive-based model to estimate the impact of a CNN on a microcontroller, regarding the latency, the power consumption and the needed memory space. The target hardware is the STM32L496ZG with CPU ARM Cortex M4 running at 14 different frequencies. Our model shows an average estimation error of 13.66% on latency, 5.52% on power consumption and 2.09% on needed memory space.**

Index Terms—**CNN, Microcontrollers, Estimation, Model**

I. Introduction

Deep learning within the Internet of Things (IoT) is a high-growing field of interest, at the intersection of machine learning and embedded systems. Recent International Data Corporation forecast estimates that 79.4 zettabytes (ZB) of data will be generated by IoT devices in 2025. Billions of these data are already used for decision-making applications. Such applications analyze signals from a wide range of sensors. They use machine learning algorithms and are deployed on Micro Controllers Units (MCUs), in an edge computing paradigm. Among them are visual object detection [1], predictive maintenance [2], or the well-known keyword spotting [3].

One of the most efficient models of machine learning algorithms since 2012 [4] is the Convolutional Neural Network (CNN), which has proved its efficiency in a wide range of vision and natural language processing applications. As problem complexity increased, new CNNs were developed to solve these issues at the expense of computational costs. For instance, VGG16 [5] needs billions of operations to compute one image [6]. This high computation cost has an impact on the energy consumption, the execution time and the need for significant memory space of the system. Whereas it is possible to consider a high impact on a system like a Many-Core Server or a Graphical Processing Unit (GPU), it cannot be considered in a MCU where the resources are scarce.

MCUs are embedded devices. On such devices, the goal of the designers is to satisfy the application constraints, like real-time execution for example, with limited resources. To do so, embedded algorithms are generally optimized in number of operations per second to: (1) realize a processing to comply with latency application requirements, and to (2) operate continuously with a minimal impact on a device battery life. To realize this optimisation, the first step is to determine the algorithm performances on the MCU. To do so, a costs estimation model on MCUs will help to reach the embedded application requirements. Therefore, we need such a model for CNNs integration within embedded systems.

Today, the easiest way to define a CNN for decision-making applications is to adapt an existing already trained CNN. The method to realize this adaptation is the transfer learning. It allows to train a neural network, even if the train dataset is not significant enough. However, transfer learning does not take into account embedded applications limits. The most optimal approach would be to create a new neural network specific to the embedded application, but there is still no clear design rules to create a CNN architecture. A recent field, Neural Architecture Search (NAS), aims at addressing this issue. Even if it is a very promising field, it presents two major drawbacks. Firstly, NAS focuses only on accuracy without taking into account embedded constraints such as execution time, energy and memory. Secondly, the computing time needed to find a solution regarding one constraint, only accuracy for example, is extremely important, hundreds of GPUs hours [7]. Computing time to find a solution that satisfies more than one constraint could become prohibitive. Consequently, a wise strategy is to use transfer learning method combined with an adaptation to the targeted MCU.

To address this adaptation, deep learning researchers have developed reduction techniques. For example, Han et al. [6] applied pruning, quantization and Huffman coding to reduce the cost of their neural network. Hinton et al. [8] applied knowledge distillation techniques to train a smaller network with the approximation function of a bigger one. Denil et al. [9] used low rank factorization to represent the weight matrix as low rank product of two smaller matrices. A question remains unanswered: what are the most appropriate reduction techniques for a specific problem with its own constraints?

In order to address this question, it is mandatory to estimate the impact of a CNN on a given MCU, to find the best optimizing policy. In this article we propose an estimation model for energy, surface and time: the EST model. It aims to help designers to choose the most appropriate neural network reduction techniques in order to respect their design constraints.

The remainder of this paper is organized as follows: Section II discusses related work on estimation models. Section III details our estimation method. Section IV presents estimation results of LeNet5 neural network running on the STM32L496ZG MCU. We conclude and discuss our future work in Section V.

II. Related Work

This section summarizes estimation models regarding execution time on different hardware, estimation of energy consumption and static memory. Works on estimation models are not dedicated to MCU architectures, we provide here several works on different architectures, Field Programmable Gate Array (FPGA), Tensor Processing Unit (TPU), GPU or CPU that can be adapted to MCU architectures.

In the recent years, two different approaches have attempted to estimate neural network latency: machine learning prediction (MLP) models and analytical prediction (AP) models. Amaris et al. [10] compare MLP models on GPUs to AP models and conclude that although AP models give better prediction, they needed specific knowledge of architectures. However, MLP models need databases that are not always available. Mu et al. [11] worked on a FPGA time model estimation and introduced a three-step model with a new data structure, the LoopTree, followed by a coarse-grained model and by an OpenCL-source-code-based fine-grained model. In addition to the engineering complexity task, the knowledge of the hardware design target board is necessary to make FPGA design efficient. Moreover, design specifications such as tiling factors, memory layout or attributes can affect the design performance significantly.

Regarding TPUs, inference time prediction was made by Kaufman et al. [12]. They used a three-step methodology based on data flow graphs and computed the runtime of an entire program by summing the runtimes of its kernel executions.

An alternative approach is used by authors of FBNet [13] on mobile CPU of Samsung Galaxy S8 and iPhoneX. They measured neural networks latency thanks to an operator LUT. The same methodology was applied by authors of ChamNet [14] on Snapdragon 835 mobile CPU and Hexagon v62 DSP. They also highlighted that optimization based on direct latency measurements instead of FLOPs can better explore hardware features. Then, they predicted energy consumption through a combination of Gaussian process model and Bayesian optimization.

For energy estimation, Tiwari et al. [15] presented a low-level model based on instruction energy costs. It is based on the sum of energy costs of the executed instructions and inter-instructions effects. According to the authors, circuit state, resource constraints and cache misses are the causes of these effects. This linear model is indeed easy to compute but does not take into account the hardware specific characteristics. Moreover, most of MCUs are running on a battery. It is therefore necessary to take into account its modelisation.

To address this issue, Dron et al. [16] introduced a method to estimate a network lifetime using an emulated application with a non-ideal battery model and a low-level description of the node hardware.

Finally, the total neural network memory size can be determined, as done by Blott et al. [17], by multiplying the sum of the weight requirements by the size of the given data type.

All in all, the limitation of these approaches is the model granularity. On the one hand, it can be easy to compute but do not reflect the hardware features. On the other hand, it can be hardware specific and thus a complex engineering task. Moreover the above estimation solutions share the limitation of one different estimation process for each feature. To the best of our knowledge, there is no estimation model for CNN inference costs on MCUs capable of accurately estimate three main features namely (1) latency (2) static memory space and (3) power consumption.

In the Section III we propose our EST model to overcome these limitations and provide a three main features estimation model. This model is based on CNN primitives.

III. Primitive-Based Model

This section explains the main principle of the primitive-based model. We explain what are the primitives and we apply this decomposition to LeNet5 neural network to validate our approach.

A. *Convolution neural network, a general composition*

Convolutional neural networks are mainly composed of two parts: the feature extractor and the classifier. Figure 1 illustrates its general architecture.

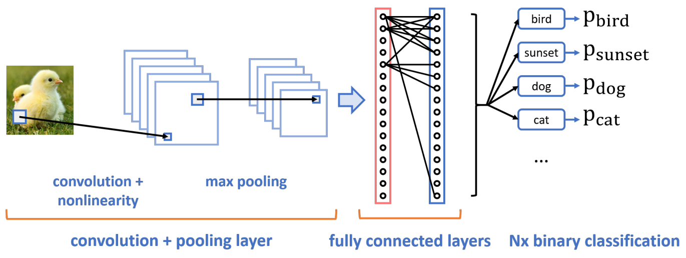

Fig. 1: CNNs architectures
source: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks

The feature extractor, represented on the left part of Figure 1, is composed of:

- convolutional layers: the basis of feature extraction;
- pooling layers: compress the information;
- non linearity activation function layers: normalize the input layer value.

The classifier, represented on the right part of Figure 1, is mainly composed of:

- fully connected layers: linear combination based on the previous feature extraction process;
- non linearity activation function layers.

The same kind of layers composes CNNs. The differences between CNN architectures can be the number and the order of layers, the kernel size or the activation function applied. As a consequence, we decomposed CNN in several primitives. This decomposition aims at simplifying the estimation impact of a CNN for a MCU. A major advantage of a primitive-based model is its good scalability.

We have identified four families of basic primitives to decompose CNN, namely:

- (1) features extraction primitives
- (2) activation function primitives
- (3) pooling primitives
- (4) fully connected primitives

Thanks to this CNN decomposition model, we evaluate the unit cost of these primitives for a specific MCU. Knowing the unit cost for one primitive running on a specific MCU, we inferred the total impact of its use, by multiplying its unit cost by the number of time it is applied in the CNN.

B. *LeNet5 primitive-based model*

To validate our primitive-based model, we use LeNet5 [18] CNN. It is represented in Figure 2. The measurement of its requirements aims to validate the primitive-based model estimation. LeNet5 network is an old state of the art CNN. However, its requirements regarding space memory and computing well illustrate the aim to successfully deploy deep learning models within MCUs [19], [20].

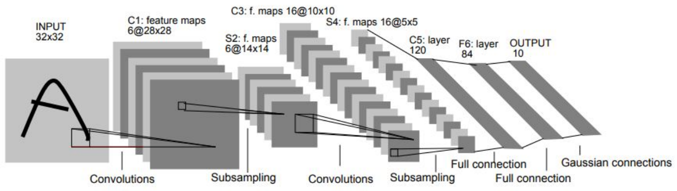

Fig. 2: The LeNet5 neural network [18]

LeNet5 is composed of 2 convolutional layers, 2 average pooling layers and 3 fully connected layers. Activation functions applied are sigmoid (1), and softmax (2) for the last layer.

$$sigmoid(x) = \frac{1}{1 + e^{(-x)}} \tag{1}$$

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{2}$$

TABLE I: LeNet5 neural network primitives

Primitive	In/Out	Kernel size	Number of applications
Convolution	x	5*5	14 304
Average Pooling	x	2*2	1 576
Fully Connected 1	400/120	x	1
Fully Connected 2	120/84	x	1
Fully Connected 3	84/10	x	1
Sigmoid	x	x	6 508
Softmax	10/10	x	1

The primitive decomposition of LeNet5 is referenced in Table I. The number of applications for each primitive was determined based on LeNet5 architecture. The input image is gray level, size 32 by 32. The convolution kernel size is 5 by 5 with a stride of 1. The average pooling kernel size is 2 by 2 with a stride of 2. LeNet5 was trained with TensorFlow framework, on the MNIST dataset. The test result showed an accuracy of 97.92% on the MNIST test dataset. Weights were retrieved from this TensorFlow model, then implemented within the hardware.

IV. Implementation

A. *Experimental Framework*

To describe each primitive and LeNet5 network, experiments were run using C/C++ code, with 32 bit floating point format. The target hardware is a Nucleo-144 development board powered by $V = 3.3V$, equipped with STM32L496ZG MCU based on the ARM Cortex M4 processing unit with 1MB of flash memory and 320KB of SRAM. The code with LeNet5 weights ran on-chip at 14 different frequencies, from $100kHz$ to $80MHz$. The compiler used is ARM compiler version 5.06 with a level 3(-O3) optimization.

Latency for an inference to core operating frequencies was carried out using a hardware timer running on the same clock as core clock. Then the timer value was multiplied by core period. In order to measure the current, we used ARM-Keil UlinkPlus probe on the jumper JP5(IDD) of Nucleo-144 board, to assess MCU power only. This probe providing a sample rate of 20MHz based on 16-bit delta-sigma technology, precision on current measurement $\pm 2\%$ and a resolution of 200nA.

For current consumption estimation, the primitive-based model was implemented in Wisebatt tool [21] on which simulations were performed. Wisebatt allows a precise energy consumption estimation of embedded systems and IoT using the research work of Dron et al. [22] [16].

Regarding the needed memory space, we added the code and data size requirements for each primitive and compared it to the LeNet5 model code and data size. This information was generated by the compiler in a MAP file.

B. *Results*

Latency measurements were based on 10 batches of 100 measures of each primitive. Average current measurements were performed on full runs of primitives, based on their respective latency at the measured frequencies. Then, the LeNet5 network was described thanks to primitive-based model and the inference cost was evaluated in the same way as primitive unit costs.

The estimation results for needed memory space is referenced in Table II. There is a difference of 2.09% between the estimation and the real requirement.

The accuracy between the real and the estimated latency per inference is highlighted in Figure 3. We observe that our primitive-based model expresses well the real behavior of latency according to running frequencies. Over all frequencies, the average difference between estimation and real latency per inference is an overestimation of 13.66%. Because our model overestimates latency on each frequency, it ensures that if the estimated neural network latency fits design constraints, real latency will also respect them.

Comparison between real and estimated average current consumption is shown in Figure 4. The average accuracy is 5.52% over all frequencies. Thanks to the combination of our model and Wisebatt tool, we were able to do an accurate estimation. Moreover, through this estimation, we will model the impact of a CNN on a complete embedded system battery life.

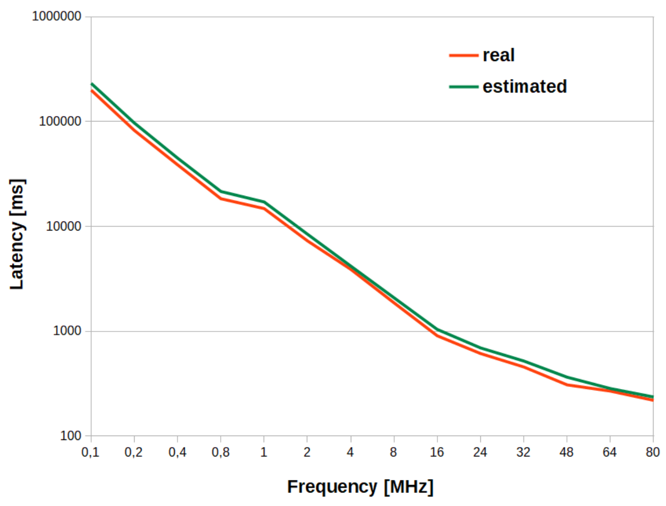

Fig. 3: Latency for one LeNet5 inference primitive-based model estimation, logarithmic scale

TABLE II: LeNet5 code and data size estimation

Code + Data size [kB]		Accuracy [%]
Primitive-based model	LeNet5	
251,60	256,98	2,09

V. Conclusion

In this paper, we present our EST primitive-based model to estimate CNN costs on a MCU. Its aim is to give key design features to choose wisely neural networks reduction techniques. Indeed, knowing quickly the reduction techniques impact on CNNs costs, will help choose the most adapted to apply for respecting embedded application limits.

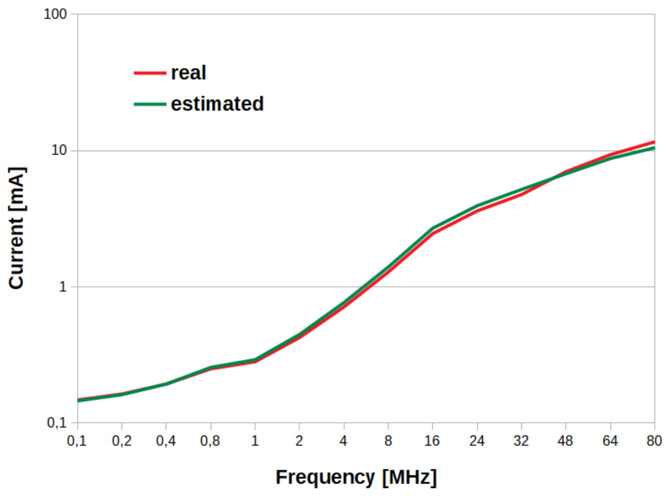

Fig. 4: Average current for one LeNet5 inference primitive-based model estimation, logarithmic scale

Results show that our proposed EST model accurately estimates LeNet5 network costs, within a STM32L496ZG based on a ARM Cortex M4 processing unit. Over 14 different frequencies, the EST model achieves, for the entire LeNet5 network, an average estimation error for one inference of 13.66% on latency, 5.52% on current and 2.09% on needed memory space for the code and data size.

Our future works will evolve around a neural networks costs estimation thanks to the EST model, over other MCUs. Moreover, neural networks downsizing techniques will be applied on bigger networks. Finally, we will characterize downsizing techniques influences on the neural network costs on Microcontroller Units.

References

[1] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, “Visual Wake Words Dataset,” *arXiv:1906.05721 [cs, eess]*, June 2019. arXiv: 1906.05721.

[2] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi, “Machine Learning for Predictive Maintenance: A Multiple Classifier Approach,” *IEEE Transactions on Industrial Informatics*, vol. 11, pp. 812–820, June 2015.

[3] Y. Zhang, N. Suda, L. Lai, and V. Chandra, “Hello Edge: Keyword Spotting on Microcontrollers,” *arXiv:1711.07128 [cs, eess]*, Feb. 2018. arXiv: 1711.07128.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[5] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv:1409.1556 [cs]*, Sept. 2014. arXiv: 1409.1556.

[6] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” *arXiv:1510.00149 [cs]*, Oct. 2015. arXiv: 1510.00149.

[7] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” 2019.

[8] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network,” *arXiv:1503.02531 [cs, stat]*, Mar. 2015. arXiv: 1503.02531.

[9] M. Denil, B. Shakibi, L. Dinh, and M. Ranzato, “Predicting Parameters in Deep Learning,” p. 9.

[10] M. Amaris Gonzalez, M. Dyab, D. Trystram, R. Camargo, and A. Goldman, *A Comparison of GPU Execution Time Prediction using Machine Learning and Analytical Modeling*. Nov. 2016.

[11] J. Mu, W. Zhang, H. Liang, and S. Sinha, “Optimizing opencl-based cnn design on fpga with comprehensive design space exploration and collaborative performance modeling,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, June 2020.

[12] S. J. Kaufman, P. M. Phothilimthana, Y. Zhou, C. Mendis, S. Roy, A. Sabne, and M. Burrows, “A learned performance model for tensor processing units,” 2021.

[13] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search,” 2019.

[14] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, P. Vajda, M. Uyttendaele, and N. K. Jha, “Chamnet: Towards efficient network design through platform-aware model adaptation,” 2018.

[15] V. Tiwari, S. Malik, and A. Wolfe, “Power analysis of embedded software: A first step towards software power minimization,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, pp. 437–445, 1994.

[16] W. Dron, S. Duquennoy, T. Voigt, K. Hachicha, and P. Garda, “An Emulation-Based Method for Lifetime Estimation of Wireless Sensor Networks,” in *2014 IEEE International Conference on Distributed Computing in Sensor Systems*, pp. 241–248, May 2014. ISSN: 2325-2944.

[17] M. Blott, L. Halder, M. Leeser, and L. Doyle, “Qutibench: Benchmarking neural networks on heterogeneous hardware,” *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 4, pp. 1–38, 2019.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[19] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden, “Tensorflow lite micro: Embedded machine learning on tinyml systems,” 2021.

[20] Y. Zhang, N. Suda, L. Lai, and V. Chandra, “Hello edge: Keyword spotting on microcontrollers,” 2018.

[21] “Wisebatt | The best way to start hardware.”

[22] W. Dron, K. Hachicha, and P. Garda, “A fixed frequency sampling method for wireless sensors power consumption estimation,” in *2013 IEEE 11th International New Circuits and Systems Conference (NEWCAS)*, pp. 1–4, June 2013.