Multi-objective Framework for Training and Hardware Co-optimization in FPGAs

Multi-objective Framework for Training and Hardware Co-optimization in FPGAs

Mohammad Amir Mansoori and Mario R. Casu

Department of Electronics and Telecommunications, Politecnico di Torino, Italy

Abstract. Although several works have recently addressed the problem of performance co-optimization for hardware and network training for Convolutional Neural Networks, most of them considered either a fixed network or a given hardware architecture. In this work, we propose a new framework for joint optimization of network architecture and hardware configurations based on Bayesian Optimization (BO) on top of High Level Synthesis. The multi-objective nature of this framework allows for the definition of various hardware and network performance goals as well as multiple constraints, and the multi-objective BO allows to easily obtain a set of Pareto points. We evaluate our methodology on a network optimized for an FPGA target and show that the Pareto set obtained by the proposed joint-optimization outperforms other methods based on a separate optimization or random search.

Keywords: Machine Learning, FPGA, High-Level Synthesis, Bayesian Optimization, Co-optimization

1 Introduction

Like in other Machine Learning (ML) models, in Convolutional Neural Networks (CNNs) the number of layers, neurons, kernel size, number of filters, etc., can be considered as hyper-parameters to tune in order to maximize the accuracy achievable during training. When it comes to implementing a CNN in FPGA using a dedicated accelerator, however, the accuracy requirements and the corresponding network architecture that meets those requirements might be in contrast with hardware-related requirements and constraints, such as latency and FPGA resources utilization. Therefore, a trade-off must be found, and this can be done by solving a multi-objective optimization problem.

In recent years, several approaches based on Hardware-aware Neural Architecture Search (Hw-NAS) have addressed the problem of co-optimizing the network and its hardware accelerator [1,2]. To solve the multi-objective optimization problem, some works use a two-stage optimization, which we call *separate* method [3, 4], in which the network parameters are first tuned to maximize the accuracy, and then the hardware configurations are tuned to meet the hardware constraints (e.g., data precision can be reduced). Other works try to reshape the problem into a single-objective one subject to some constraints on hardware

performance [5,6]. Another method combines multiple objectives in a single function and uses single-objective optimization approaches [7], even though merging multiple objectives can limit the performance of the optimization and degrade the final Pareto-optimal sets. The last methodology is to employ a *truly* multi-objective optimization approach to obtain the non-dominated Pareto solutions. This approach has attracted considerable attention in the evolutionary computation community [8]. However, the computational complexity in evolutionary algorithms is the main limitation of these approaches.

Not all Hw-NAS approaches co-optimize the network and its accelerator: most of them use a *fixed hardware configuration* (fixed-Hw) where the search space is limited to the model architecture [1]: if the hardware requirements are not met, the network architecture must change. Another category uses *multiple hardware configurations* (multi-Hw) in which the search space combines hardware configurations and network architectures, which is also our approach.

The optimization algorithms in Hw-NAS are usually divided into Reinforcement Learning (RL), Evolutionary algorithms (EAs), Gradient-based methods, and Bayesian Optimization (BO) methods [9]. RL and EA have been recently used for both *fixed-Hw* and *multi-Hw* categories (e.g., the Genetic Algorithm in [10]). Despite their proven effectiveness in several works, they have some drawbacks: EAs are computationally intensive due to their requirement for a large *population* size in each *generation*; in RL, customizing the *policies* and *reward function* for each optimization problem is a challenge.

Gradient-based methods fall in the *fixed-Hw* category. In these methods a super-network containing all the possible realizations in the search space is trained and a sub-network is sampled in each Hw-NAS iteration, guiding the search to the optimal network [11]. Although hardware metrics can be included in the loss function of the super-network, the search space includes only the network configurations, hence the fixed-Hw categorization.

BO approaches consider a Gaussian Process for each objective, which is a natural fit for the optimization problems in Hw-NAS, and no customization is required in contrast to RL methods [12,13]. In addition, BO enables a truly multi-objective optimization. Despite these advantage, to the best of our knowledge, BO methods in FPGAs have been used only for the fixed-Hw category.

In this paper, we propose a new framework based on Multi-Objective Bayesian Optimization with Constraints (MOBOC) on top of High Level Synthesis (HLS) to jointly optimize the network architecture and the HLS-based hardware configurations for FPGA devices. The search space supports multi-Hw configurations. In this multi-objective framework, we can assign multiple objectives and constraints related to the hardware and network performance, and use the truly multi-objective BO approach to find the optimal Pareto sets.

2 Proposed Methodology

Our method can be divided in the four parts shown in Fig. 1 (B-E) and connected inside the multi-objective BO framework (A). MOBOC will update the samples

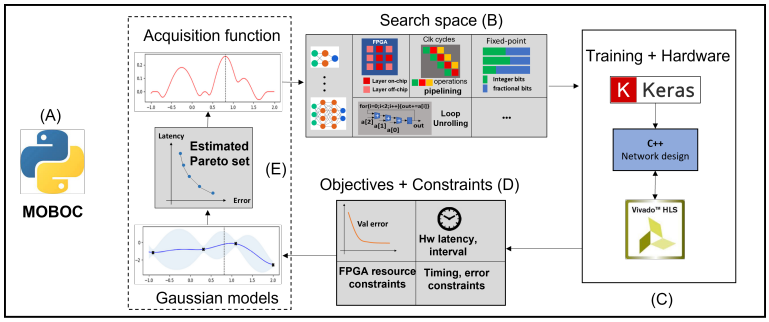


Fig. 1: Proposed methodology for training and hardware co-optimization in FPGAs.

from the search space based on the expected improvement in the Pareto sets to find the optimum configurations for network and hardware accelerator. In the following, we illustrate in detail MOBOC and each part of the framework.

Multi-Objective BO with Constraints (MOBOC): BO is a method to optimize black-box functions that are expensive to evaluate. In MOBOC, a Gaussian model is initially built for each objective and constraint using few samples. An acquisition function is also built, based on the expected improvement of the estimated Pareto sets obtained from the Gaussian models. The maximum value of this function will suggest a new point in the search space. Objectives and constraints are evaluated for the new point and guide the search towards the optimum configurations by updating the Gaussian model at each iteration [14].

Search Space: It combines all the values of training hyper-parameters and HLS hardware configurations. Although the framework supports any ML model with an HLS code ready for FPGA implementation, in this work we focus on a CNN classifier for the MNIST dataset inspired from Lenet-5 [15] and consisting of two Convolutional, two max-pooling, and three dense layers. For each layer, the hyper-parameters include the number of neurons, filters, and kernel size.

Several HLS hardware knobs can configure the FPGA accelerator, such as the choice between on-chip or off-chip memory for storing the weights of a layer, specific HLS pragmas for loop pipelining and for loop unrolling with a configurable unroll factor, enabling or disabling the Dataflow pragma for task-level concurrency, the configuration of the fixed-point precision, and the value of the clock frequency. The optimal choice of these HLS *directives* and hardware parameters is achieved by MOBOC in conjunction with the optimal training parameters.

Function Evaluations: At each iteration, a new suggested point in the search space must be evaluated. For this evaluation, we used Keras for training and HLS C-simulation and Synthesis for hardware verification.

Objectives and Constraints: Even though we can assign any number of objectives and constraints, the larger the number, the longer the time to update the Bayesian model in each iteration. In this work, we selected to optimize the prediction error on the validation set, the hardware latency, and the throughput, with constrained FPGA resources (BRAM, DSP48, FF, LUT), clock period, and

maximum admissible error ($< 10\%$). Note that instead of using the prediction error during training, we use the prediction error after execution in hardware: in this way the error includes also the effect of the fixed-point precision.
Update Bayesian model: The new evaluation is used to update the Gaussian models. For the acquisition function, Predictive Entropy Search for Multi-objective Optimization with Constraints (PESMOC) [16] is used to update the expected improvement of the estimated Pareto set. After the update, the maximum value of the acquisition function corresponds to the parameters of the new suggestion. Convergence criteria or maximum iterations stop the procedure.

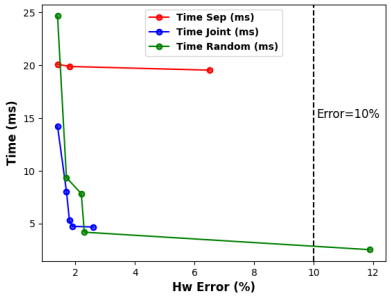
3 Results

The FPGA target is a Zynq-7000 SoC (XC7Z020-CSG484). For MOBOC, the latest *Spearmint* package is used [16]. The search space is described in Tab. 1, in which *On Chip* refers to the option of selecting on-chip BRAMs to store weights and bias values for each layer (as opposed to an off-chip external DRAM), hence 5 options are available for 5 layers. Total and Integer bits in Tab. 1 determine the fixed-point data precision that can be individually configured for weights, activations, and accumulations for the intermediate calculations (i.e., sum of products). Note that the search space size is on the order of 10^{16} total configurations—even without considering other parameters such as the learning rate, which prevents to do an exhaustive search of the best configurations. However, the BO convergence would be slower: in the current setting 100 BO iterations are sufficient to obtain a Pareto set, with more parameters more iterations would be needed.

Fig. 2 shows the Pareto-optimal points in the space of prediction error evaluated in hardware (Hw error, x axis) and execution latency (Time, y axis) achieved by three different methods: Random search, Separate, and the proposed Joint method. The search space for all the methods is the same and the same limit of 100 iterations is used. In the Random case, we randomly choose in each iteration a point in the search space. In the Separate method, we first optimize the network hyper-parameters to obtain the best accuracy, and then BO optimizes the hardware configurations to maximize the hardware performance. In the Joint method, BO jointly optimizes the hyper-parameters and the hardware configuration to maximize both prediction accuracy and performance. Note that since in Random search we cannot set an error constraint, there is one Pareto point with error larger than the admissible threshold ($> 10\%$). With the excep-

Configs	**Range**	**Configs**	**Range**
#Filters Conv1	[2,20]	Dataflow	active/inactive
		On Chip	active/inactive
#Filters Conv2	[2,20]	Clk (ns)	[8,20]
		Total bits	[10,20]
Kernel size	2*([1,4]) + 1	Integer bits	[1,9]
#Neurons Dense1/2	[50,150]	Conv1/2 UF	2^([0,4])

Table 1: Search space: network and hardware parameters (UF:Unroll Factor)



Random		Separate		Joint	
Error (%)	Time (ms)	Error (%)	Time (ms)	Error (%)	Time (ms)
1.4	24.67	1.4	20.1	1.4	14.22
1.7	9.36	1.8	19.89	1.7	8.06
2.2	7.83	6.5	19.54	1.8	5.31
2.3	4.19	-	-	1.9	4.74
11.9	2.53	-	-	2.6	4.68

Fig. 2: Pareto-points found by the joint method, random search, and separate method in the space of prediction error (Hw error) and execution latency (Time).

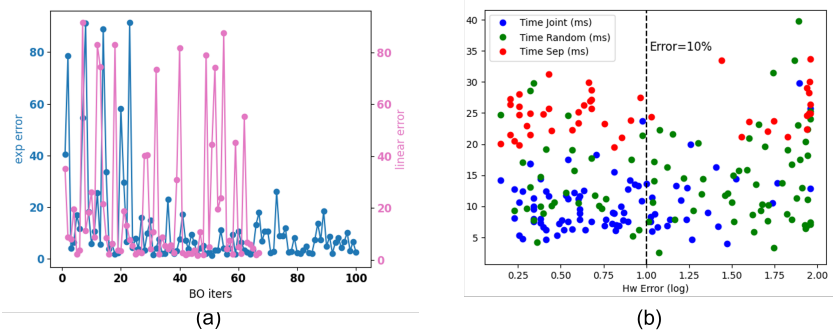


Fig. 3: (a) Hardware inference error in our joint method in each BO iteration (linear and exponential), (b) Total points suggested by the joint, separate, and random search methods; note the concentration of the joint method on low errors ($< 10\%$).

tion of one lucky point detected by the Random search, all the other points are Pareto-dominated by those found by the Joint method.

To help speed-up the BO convergence toward low prediction error, we modified the objective function with a non-linear exponential of the error. Fig. 3(a) shows the error as a function of the BO iterations: the exponential definition of the error helps reach a quicker convergence than a linear definition does. The final results in this work are based on the exponential error.

Finally, Fig. 3(b) shows all the points found in 100 iterations. For better clarity, a log scale is used for the error. Notice how the error threshold concentrates the points below 10% for our joint MOBOC method, which helps in better optimization convergence by searching in the area of interest.

4 Conclusions

We proposed a joint optimization of CNN training and hardware configurations in FPGAs, within a multi-objective Bayesian Optimization (BO) frame-

work. In contrast to more common BO approaches, the search space includes multiple hardware configurations in addition to the hyper-parameters used for CNN training. Moreover, conflicting objectives related to training and hardware performance can be considered separately to achieve a Pareto-optimal set of configurations without merging them into a single objective function. We compared our joint methodology with conventional approaches, random search and separate optimization method. The Pareto set achieved with our method outperforms those obtained with random and separate methods, with 1.7× and 1.4× improvement in execution time for the minimum error, respectively.

References

1. H. Benmeziane *et al.*, "A comprehensive survey on hardware-aware neural architecture search," *CoRR*, vol. abs/2101.09336, 2021.
2. L. Sekanina, "Neural architecture search and hardware accelerator co-search: A survey," *IEEE Access*, vol. 9, pp. 151 337–151 362, 2021.
3. H. Cai *et al.*, "Once-for-all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020.
4. S. Han *et al.*, "Design automation for efficient deep learning computing," *CoRR*, vol. abs/1904.10616, 2019.
5. M. Tan *et al.*, "Mnasnet: Platform-aware neural architecture search for mobile," *CoRR*, vol. abs/1807.11626, 2018.
6. B. Wu *et al.*, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
7. C. Hsu *et al.*, "MONAS: multi-objective neural architecture search using reinforcement learning," *CoRR*, vol. abs/1806.10332, 2018.
8. X. Chu *et al.*, "Multi-objective reinforced evolution in mobile neural architecture search," *CoRR*, vol. abs/1901.01074, 2019.
9. M. Parsa *et al.*, "Pabo: Pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design," in *ICCAD*, 2019, pp. 1–8.
10. Y. Yu *et al.*, "Software-defined design space exploration for an efficient dnn accelerator architecture," *IEEE Trans. Computers*, vol. 70, no. 1, pp. 45–56, 2021.
11. H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *CoRR*, vol. abs/1812.00332, 2018.
12. D. Stamoulis *et al.*, "Hyperpower: Power- and memory-constrained hyperparameter optimization for neural networks," in *DATE*, 2018, pp. 19–24.
13. E. Liberis *et al.*, "μNAS: constrained neural architecture search for microcontrollers," in *1st Workshop on Machine Learning and Systems*, 2021, p. 70–79.
14. S. Greenhill *et al.*, "Bayesian optimization for adaptive experimental design: A review," *IEEE Access*, vol. 8, pp. 13 937–13 948, 2020.
15. Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
16. E. Garrido-Merchán *et al.*, "Predictive entropy search for multi-objective bayesian optimization with constraints," *Neurocomputing*, vol. 361, pp. 50–68, 2019.