An End-to-End Workflow to Efficiently Compress and Deploy DNN Classifiers on SoC/FPGA

Romina Soledad Molina, Iván René Morales, Maria Liz Crespo, Veronica Gil Costa, Sergio Carrato, and Giovanni Ramponi, *Life Senior Member, IEEE*

***Abstract*—Machine learning (ML) models have demonstrated discriminative and representative learning capabilities over a wide range of applications, even at the cost of high-computational complexity. Due to their parallel processing capabilities, reconfigurability, and low-power consumption, systems on chip based on a field programmable gate array (SoC/FPGA) have been used to face this challenge. Nevertheless, SoC/FPGA devices are resource-constrained, which implies the need for optimal use of technology for the computation and storage operations involved in ML-based inference. Consequently, mapping a deep neural network (DNN) architecture to a SoC/FPGA requires compression strategies to obtain a hardware design with a good compromise between effectiveness, memory footprint, and inference time. This letter presents an efficient end-to-end workflow for deploying DNNs on an SoC/FPGA by integrating hyperparameter tuning through Bayesian optimization (BO) with an ensemble of compression techniques.**

***Index Terms*—Compression, deep neural networks, FPGA/SoC, machine learning (ML), workflow.**

I. Introduction

MANY new machine learning (ML) application fields require mapping deep neural network (DNN) inference processes into systems on chip based on a field programmable gate array (SoC/FPGA), to exploit some inherent favorable features of these technologies, such as low latency, low-power consumption, and high parallelism. Compression techniques become essential to deploy ML models on resource-constrained devices, preserving effectiveness in smaller and faster models [1], [2].

Long design cycles are needed to implement suitable FPGA firmware optimizing latency and resource utilization. Developers would benefit from a methodology guiding DNN implementation on SoC/FPGA, unifying existing techniques and considering the interaction between the FPGA and the processor that make up to the SoC.

Recent studies in this field usually focus on the integration of pruning and/or quantization methods for model compression, and only a few contributions address the entire development cycle [3], [4], [5]. Specifically, Fahim et al. [3] proposed a co-design workflow, including quantization-aware training and pruning, to optimize the architectures for low-power devices based on FPGA, without including knowledge distillation (KD) neither a strategy for FPGA verification. Furthermore, the co-design workflow was validated with MNIST dataset and not real-world applications. This letter in [5] presents a co-design methodology for FPGA-based DNN accelerators developed for inference at the edge, but requires a high degree of data transfer between on-chip and off-chip memory. The proposal in [4] exposes an FPGA/DNN co-design for low-end devices, but is limited to convolutional neural networks (CNN) and assumes the weights are stored in external memory.

In this letter, we propose a workflow[1] that integrates KD, pruning, and quantization to efficiently compress and deploy DNN-based classifiers on SoC/FPGA, *addressing the entire development cycle: from the ML-based architecture training to the hardware deployment*, tackling the drawbacks presented in [3], [4], and [5]. The inclusion of the KD technique, combined with hyperparameter optimization (HPO) through Bayesian optimization (BO), allows the generation of a reduced and more efficient model (student model) that mimics the overall accuracy of a teacher architecture.

HPO is carried out using BO because it is supported by the literature as one of the most effective techniques for performing this task [6]. In contrast, grid search suffers from the curse of dimensionality, while random search is more efficient than grid search but is unreliable for training some complex models [7]. In the proposed methodology, the developer defines the different topologies (teacher and student) and employs HPO to find the best set of values for a DNN topology previously defined.

The proposed workflow allows full on-chip storage of ML-based models. This makes it suitable for applications that require fast inference and energy efficiency, unlike the contributions in [4] and [5] that require a high degree of data transfer between the on-chip and off-chip memory. The reduced model is obtained at the expense of a longer training

Romina Soledad Molina and Iván René Morales are with the Multidisciplinary Laboratory (MLab), STI Unit, Abdus Salam International Centre for Theoretical Physics, 34151 Trieste, Italy, and also with the Department of Engineering and Architecture (DIA), University of Trieste, 34127 Trieste, Italy (e-mail: rmolina@ictp.it).

Maria Liz Crespo is with the Multidisciplinary Laboratory (MLab), STI Unit, Abdus Salam International Centre for Theoretical Physics, 34151 Trieste, Italy.

Sergio Carrato and Giovanni Ramponi are with the Department of Engineering and Architecture (DIA), University of Trieste, 34127 Trieste, Italy.

Veronica Gil Costa is with CONICET, National University of San Luis, San Luis 5700, Argentina.

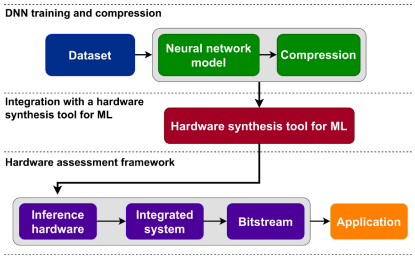

Fig. 1. Workflow to efficiently compress and deploy DNN-based classifiers on SoC/FPGA.

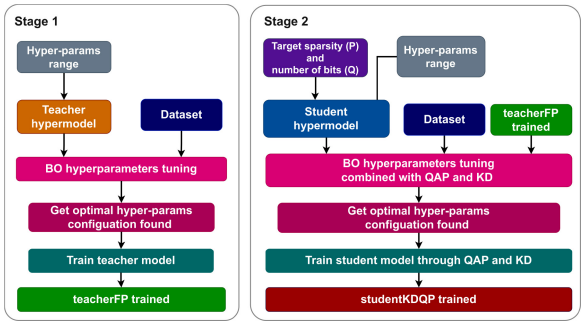

Fig. 2. Training and compression stages.

time and a more extensive search space for HPO. The main contributions of this letter can be summarized as follows.

1) A workflow based on KD combined with quantization-aware pruning (QAP) and HPO, aiming to guide the developer in obtaining and compressing ML-based models and implementing them in SoC/FPGA,
2) An SoC/FPGA framework containing the ComBlock IP core [8], which facilitates the communication protocols, and the control and evaluation of ML-based hardware from the SoC's soft-core or hard-core processor,
3) The incorporation of KD and BO in the training steps, allowing a full on-chip deployment on FPGA, thus avoiding off-chip communication bottlenecks.

The remainder of this letter is organized as follows. Section II introduces the workflow to compress and deploy DNNs on SoC/FPGA. The experiments and results are presented in Section III. Finally, Section IV summarizes the conclusions.

II. Workflow

The main steps of the proposed workflow are shown in Fig. 1. The input to the workflow is the dataset used to train and compress the target model, as described in Section II-A. Hereafter, a data structure is generated with information regarding the layers, weights, and bias of the model. This data structure is used by a hardware synthesis tool for ML to translate the DNN model, as presented in Section II-B. Subsequently, inference hardware is created and integrated into an assessment template to verify its functionality on the SoC/FPGA device, as introduced in Section II-C.

The implementation details are available in the repository: https://github.com/RomiSolMolina/workflowCompressionML.

A. DNN Training and Compression

The training and compression of the DNN classifier include the following stages (Fig. 2):

1) Stage 1—Teacher Training: The teacher hypermodel is created by the user defining the number and type of layer, and the search space, which is a range of values per layer defined with a heuristic approach by the user. Using the dataset and the teacher hypermodel, the BO process is performed according to the number of iterations specified by the developer. BO is derived from probabilistic models and aims to maximize or minimize an objective function (e.g., model accuracy or loss) by efficiently exploring the hyperparameter space and identifying the configuration that maximizes or minimizes the function. In this process, BO guides the search of hyperparameters using past evaluation results. During the search, a new architecture is compiled and trained for each iteration, whereas the optimizer looks at a combination of hyperparameters within the defined search space, considering the validation accuracy as the objective function.

After optimizing, the network is trained with the optimal configuration found, generating a teacher model with 32-bit floating-point precision (teacherFP). Alternatively, a pretrained model can be used as a teacherFP.

2) Stage 2—Student-Aware Training: The corresponding student hypermodel is constructed, and a search space is created within a range of values that defines the number of neurons and kernels per layer. Furthermore, the hypermodel is defined with n-bit fixed-point precision (where n is the number of bits) for the quantization (Q) method and with target sparsity for pruning (P). Because the student network has a reduced number of parameters compared to the teacher network, the range of values for the search space (for each layer) is set to a smaller number. Using the dataset, the student hypermodel, and teacherFP model, the BO process is performed according to the number of iterations specified by the developer. The learning approach chosen during the optimization is KD, which is combined with QAP to transfer the knowledge of the teacherFP to a smaller architecture. QAP incorporates quantization and pruning into the student training process.

Once the optimization process is completed, the selected hyperparameter values are delivered to the developer. With this, the knowledge of teacherFP is distilled into the student network, previously defined by the number of bits and target sparsity used in the optimization process. QAP is employed along with the learning process through KD. Finally, the compressed *studentKDQP* model is trained.

3) General Key Points: BO, implemented through the kerasTuner [9], performs parameter searches based on developer-configured settings, generating architectures with more or fewer parameters. The heuristic search spaces for teacher optimization range between 32 and 300, whereas for the student from 1 to 20. Fig. 3 shows an example of the search space definition considering one convolutional layer. To tune the number of kernels in the first layer, the algorithm searches for an optimal value between 32 and 128 in 8-steps incremental. The optimal learning rate for teacher and student models is selected from 0.01, 0.001, or 0.0001.

The dataset was divided into training, validation, and testing (70%, 15%, and 15%, respectively) for both stages of the

```
model.add(Conv2D(
    hp.Int("conv_1", min_value=32, max_value=128, step=8),
    (3, 3), padding="same", input_shape=INPUT_SHAPE))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Fig. 3. Example of the search space definition.

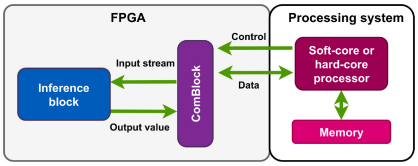

Fig. 4. Hardware assessment framework.

DNN training and compression process. When using pre-trained networks from SOTA, it is necessary to fine-tune the architecture with the corresponding dataset to adapt it to a specific problem.

B. Integration With Hardware Synthesis Tool for ML

After DNN training and compression, the model is saved in a data file suitable for the backend tools to perform hardware synthesis for ML-based designs. We chose hls4ml [10] because it is open-source and supports as high-level synthesis (HLS) backends Vivado HLS, Intel HLS, and Vitis HLS in the experimental mode. However, the proposed workflow can be adapted to other hardware synthesis tools for ML applications.

The DNN model is saved in a hierarchical data format (.h5 file) containing the data structure of the model, such as the layers, weights, and biases. This is the input for the hls4ml package that is linked to an HLS tool that allows the generation of hardware from the C, C++, and System C codes. hls4ml creates an HLS project based on the configuration provided by the developer, such as the FPGA part, reuse factor, data precision, and latency/resource strategies. Once the HLS project is generated, a hardware block can be synthesized and exported to the FPGA register transfer level (RTL).

C. Hardware Assessment Framework

An SoC/FPGA framework was developed to test the deployment of DNN models in hardware (Fig. 4). The assessment framework includes an embedded processor (soft core or hard core) that acts as a programmable control device. The DNN inference hardware exported from the HLS is placed within the framework design to enable flawless interchangeability during the testing iterations. Communication between the processor and the inference hardware is carried out using ComBlock [8], an open-source, portable, and customizable IP block designed to simplify the interaction between the embedded processor and FPGA. We selected ComBlock because of its easy integration with soft-core and hard-core processors. However, the developer can select a different hardware communications block, such as a DMA controller.

Through ComBlock, the processor sends the input stream to the inference block and retrieves the results of the DNN classifier. Algorithm 1 summarizes how easily the processor interacts with the FPGA, using ComBlock for initialization and data streaming.

Algorithm 1 Pseudocode Running on the Processor

Input: *inputSignal*[*INPUT_SIZE*]
1: *comBlock.fifo_flush*(*fifo*)
2: *comBlock.register_write*(*reset_DNN*, 0)
3: **while** true **do**
4: **for** i = 0, i < INPUT_SIZE **do**
5: *comBlock.fifo_write*(*inputSignal*[*i*])
6: **end for**
7: *outVal* ← *comBlock.register_read*(*dnnOutput*)
8: **end while**

III. Experiments and Results

A. Real Case Study Applications

A preliminary version of our work was presented in [11], [12], [13], and [14], where we applied Q, P, and KD for model compression to two real-world applications, and we provided evidence of performance improvement when targeting SoC/FPA devices. The former application (1D-multilayer perceptron (MLP)) is focused on 1-D signals; it is a pulse shape discriminator (PSD) based on a MLP, to be used for event recognition in cosmic rays studies [12], [13]. The latter application is in the field of object classification in 2D-images; its aim is moth classification in the context of pest detection. The solution in [11] (2D-CNN) is based on ad-hoc CNN trained with a dataset obtained from in-field traps through an IoT system. This application is further developed in [14] (2D-VGG16) using the larger pretrained teacher network (VGG16) [14], and a public dataset (Pest24 [15]) that was employed for fine-tuning the model. With this application, we aimed to illustrate that the developer may have a previously trained model—in this case, extracted from [14]—and compress it through the proposed methodology (without the need to generate the teacher model from scratch), and the potential of the compression method when working with larger teacher architectures. Moreover, we extended the 2-D classification problem using MobileNetV2 (2D-MobileNetV2) as a teacher model, which was previously fine-tuned with Pest24 and modified with a classifier suitable for pest classification. This option was selected because MobileNetV2 is a topology engineered for operation in resource-constrained environments.

In this letter, we apply the end-to-end workflow in both real-case studies, complementing the results in [11], [12], [13], and [14].

B. Workflow Assessment

Table I summarizes the most relevant results. In 1D-MLP, the teacher is an MLP with four fully connected layers. The 2D-CNN teacher is composed of four convolutional and three fully connected layers. For 2D-VGG16 and 2D-MobileNetV2, the teacher models were obtained through transfer learning after fine-tuning the network for Pest24 dataset. For the four architectures, the overall accuracy was above 97%.

Regarding the compressed models, the 1D-MLP student network was implemented and tested on an Artix-7 FPGA board, using an embedded soft-core processor based on Microblaze. The 2D-VGG16 and 2D-CNN student models were implemented and tested on an UltraScale+ ZCU102 with

TABLE I
EVALUATION. THE ACRONYMS IN THE TABLE ARE P: PARAMETERS, CR: COMPRESSION RATIO, NL: NUMBER OF LAYERS (ONLY CONVOLUTIONAL AND FULLY CONNECTED LAYERS), OA: OVERALL ACCURACY, SP: SPARSITY, AND L: LATENCY WITHOUT DATA TRANSFER AT 200 MHZ (5-NS CLOCK CYCLE)

	Teacher model		Compressed student models						Hardware implementation of the compressed student models					
Application	**P**	**OA[%]**	**P**	**CR**	**NL**	**OA[%]**	**Bits**	**SP[%]**	**FPGA**	**BRAM [%]**	**DSP[%]**	**FF[%]**	**LUT[%]**	**L [clk]**
1D-MLP	16,352	99.7	529	30.91	1	98.96	8	20.00	Artix-7	0.00	14.00	2.75	26.27	10
2D-CNN	723,499	99.19	3,699	195.59	11	94.11	8	50.00	ZCU102	5.70	10.90	6.00	18.79	17,411
2D-VGG16	14,818,695	98.87	3,207	4,898.50	16	96.67	8	60.00	ZCU102	10.84	17.54	6.02	15.40	18,005
2D-MobileNetV2	3,410,286	97.52	4,595	743.16	5	95.6	8	50.00	ZCU102	11.00	1.78	8.00	13.00	16,472

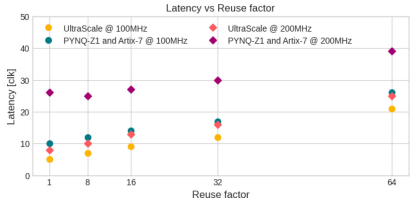

Fig. 5. Latency (in clock cycle units) for 1D-MLP. Data from HLS reports at different clock frequencies (100 and 200 MHz).

an embedded hard-core processor. Table I shows the post-place and route resource utilization for the compressed models.

We analyze 1D-MLP student latency with reuse factors of 1, 8, 16, 32, and 64. This parameter, introduced by hls4ml, is intrinsically associated with the use of multipliers in the FPGA to compute a layer of neuron values. A low-reuse factor implies a low-latency and high-resource utilization. The results are presented in Fig. 5. An increased latency (in clock cycle units) is evident when running at 200 MHz, compared to 100 MHz. Such difference is more noticeable in the Artix-7 FPGA and the SoC-based PYNQ-Z1, using a 28 nm process technology, versus the 16 nm fabric of the UltraScale+ SoC.

Considering image classification, we addressed the compression of the ML-based model in [11]. For 2D-VGG16, a suitable network was obtained by employing BO for HPO, considering an implementation on a high-end FPGA but with twice the number of parameters compared to the [14] model (which was generated by manually tuning the hyperparameters). For the cosmic ray application, a reduced architecture was generated compared to the ones obtained in [12] and [13] (26% and more than 70% FPGA usage, respectively).

The experiments show that our proposed methodology successfully generates compressed models, leading to a fully on-chip memory-mapped implementation on the FPGA. Integrating KD into the ensemble of compression techniques contributes to achieving a balanced student model in terms of size, computational efficiency, and accuracy. The methodology addresses the entire development cycle, overcoming the limitations outlined in previous works, such as [3], [4], and [5]. Furthermore, the ComBlock in the hardware assessment framework facilitates the model performance evaluation in the FPGA by simplifying the communication interfaces.

IV. CONCLUSION

In this letter, we presented an end-to-end workflow for efficient DNN-based classifier compression and deployment on SoC/FPGA. Combining pruning, quantization, and KD leads to a good tradeoff between resource utilization and quality metrics. BO automatically selects an optimal combination of hyperparameters for the teacher and student models. An improvement in latency and energy efficiency was achieved owing to the minimal off-chip data transactions and reduced computational operations required to complete the inference. Future work could include neural architecture search (NAS) as an extension of the methodology.

REFERENCES

[1] T. Aarrestad et al., "Fast convolutional neural networks on FPGAs with hls4ml," *Mach. Learn., Sci. Technol.*, vol. 2, no. 4, Jul. 2021, Art. no. 045015.

[2] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artif. Intell. Rev.*, vol. 53, no. 7, pp. 5113–5155, 2020.

[3] F. Fahim et al., "hls4ml: An open-source codesign Workflow to empower scientific low-power machine learning devices," in *Proc. Res. Symp. Tiny Mach. Learn.*, 2021, pp. 1–10.

[4] C. Hao et al., "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Las Vegas, NV, USA, 2019, pp. 1–6.

[5] J. Haris, P. Gibson, J. Cano, N. B. Agostini, and D. Kaeli, "SECDA: Efficient hardware/software co-design of FPGA-based DNN accelerators for edge inference," in *Proc. IEEE 33rd Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, Belo Horizonte, Brazil, 2021, pp. 33–43.

[6] A. H. Victoria and G. Maragatham, "Automatic tuning of hyperparameters using Bayesian optimization," *Evolving Syst.*, vol. 12, no. 1, pp. 217–223, 2021.

[7] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 2, pp. 1–25, 2012.

[8] K. S. Mannatunga et al., "Design for portability of reconfigurable virtual instrumentation," in *Proc. 10th Southern Conf. Program. Logic (SPL)*, Buenos Aires, Argentina, 2019, pp. 45–52.

[9] T. O'Malley et al., "Keras tuner." 2019. [Online]. Available: https://github.com/keras-team/keras-tuner

[10] J. Duarte et al., "Fast inference of deep neural networks in FPGAs for particle physics," *J. Instrum.*, vol. 13, no. 07, 2018, Art. no. P07027.

[11] A. Suárez, R. S. Molina, G. Ramponi, R. Petrino, L. Bollati, and D. Sequeiros, "Pest detection and classification to reduce pesticide use in fruit crops based on deep neural networks and image processing," in *Proc. 19th Workshop Inf. Process. Control (RPIC)*, San Juan, Argentina, 2021, pp. 1–6.

[12] R. S. Molina et al., "Compression of NN-based pulse-shape discriminators in front-end electronics for particle detection," in *Proc. Appl. Electron. Pervading Ind., Environ. Soc.*, 2021, pp. 93–99.

[13] L. G. G. Ordóñez et al., "Pulse shape discrimination for online data acquisition in water cherenkov detectors based on FPGA/SoC," in *Proc. 37th Int. Cosmic Ray Conf.*, Berlin, Germany, 2021, pp. 1–8.

[14] R. S. Molina et al., "ML-based classifier for precision agriculture on embedded systems," in *Proc. Appl. Electron. Pervading Ind., Environ. Soc.*, 2023, pp. 117–124.

[15] Q.-J. Wang et al., "Pest24: A large-scale very small object data set of agricultural pests for multi-target detection," *Comput. Electron. Agriculture*, vol. 175, Aug. 2020, Art. no. 105585.