Deep Learning Algorithms Elaboration for Embedded Systems Implementation

Luigi Capogrosso
Department of Computer Science, University of Verona
luigi.capogrosso@univr.it

***Abstract*—Recent advances in deep learning have brought a step-change in the abilities of machines in solving complex problems like object recognition, person detection, and pose estimation. Although many of these tasks are important on mobile and embedded devices, especially for sensing and mission-critical applications such as autonomous driving and video surveillance, existing deep learning solutions often require a large number of computational resources to run. Running these models on embedded devices can lead to long runtimes and the consumption of abundant amounts of resources, including CPU, memory, and power, even for simple tasks. Without a solution, the hoped-for advances in embedded machine/deep will not arrive. This thesis proposes novel techniques that guarantee a smooth transition of deep learning technology from a scientific environment with virtually unlimited computing resources into embedded systems. At the moment, specifically deal with the two typologies of deep learning techniques for achieving this goal: *split computing* and *tinyML*.**

***Index Terms*—Deep Neural Networks, Embedded Devices, Split Computing, tinyML, Simulation, Testing**

I. Introduction

In the last decade, deep neural networks (DNNs) achieved state-of-the-art performance in a broad range of problems, spanning from object classification and feature detection to speech recognition and predictive maintenance. This success comes at a price: the computational requirements of some DNNs preclude their deployment on most of the resource-constraint devices, such as mobile phones available today; we refer to this scenario as *local-only computing* (LC). A possible alternative consists of running simplified models, such as MobileNetV2 on the devices, but this has an impact on the overall accuracy. The current approach, usually referred to as *remote-only computing* (RC), consists in transferring the data captured by the device to a high-performance machine through a communication network and then sending back the result to the device. In this case, the communication network may become a bottleneck and, in any case, it should be properly configured to match the quality of service requirements of the pattern recognition application. As a compromise between LC and RC approaches, the *split computing* (SC) frameworks propose to divide the DNN model into a "head" on the sensing device and a "tail" on the remote server.

Starting from the above examples, it is understandable that the design of a distributed deep learning application results in moving into a *three-dimensional design space exploration*. A given implementation is determined by the choice of the computation platform, the communication architecture, and the deep neural network. Whereas the first two dimensions are deterministic, in the sense that a given choice of platform or communication architecture brings to a certain performance, dealing with DNNs introduces uncertainty. Actually, a DNN is a statistical classifier, with millions of parameters and plenty of architectures, whose timing is not deterministic as so as its efficacy, usually measured in terms of accuracy. It follows that a given DNN can be evaluated only by training it and testing on some validation partition. When it comes to distributed architectures, and split computing strategies, the situation becomes even harder, since it type of split requires specific training. It turns out that deciding on a properly distributed architecture hosting DNNs, and manipulating diverse split computing configurations requires days.

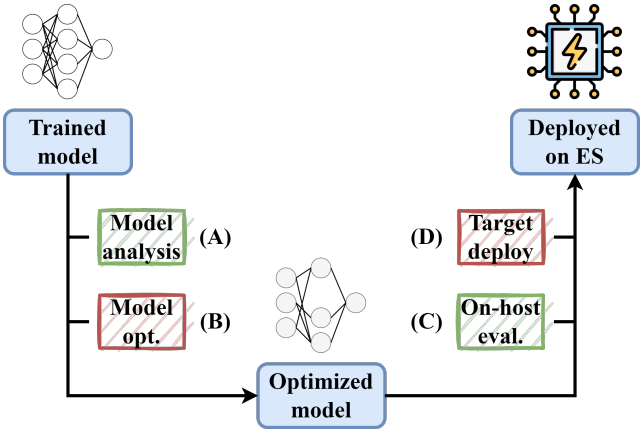

Fig. 1. Steps to enable a smooth transition of deep learning algorithms into embedded systems. The boxes in green, thus the *model analysis* (A) and *on-host evaluation* (C) sections are the phases where we started to set up our work. On the other hand, the boxes in red, *i.e.*, sections *model optimization* (B) and *target deployment* (D), are those on which future work will be focused.

Furthermore, over the decades there has been tremendous research focus dedicated to meliorating embedded technologies for use in resource-limited environments. This research has been driven by the fact that it is the key strategy for providing real-time solutions for many complicated, and safety-critical real-world applications. In this regard, the microcontroller unit (MCU) based embedded systems have garnered tremendous attention, primarily due to the low power requirement, secondly due to the crucial performance and reliability traits such as safety, security, maintainability, and adaptability.

This leads to the need to adopt a new paradigm: *tinyML*. It's only recently that we've been able to run ML on a microcontroller at all, which means that hardware, software, and research are all changing extremely quickly.

Therefore, we can easily understand how embedded deep learning will open up a series of possibilities for applications in IoT devices, such as autonomous cars, and other devices so that they have intelligent functionalities that today are restricted to computers and smartphones. We will see voice interfaces in almost everything in the future. As soon as we can create suitable voice interfaces at a low cost, we will have them on any consumer item, replacing buttons on any device, especially if you think of devices combining audio and video.

II. Achieved Results

In this paper [1], we propose a fast procedure to select the best split location for a generic DNN architecture that, for the first time, is predictive of the accuracy that the system will have once retrained. The procedure is dubbed I-SPLIT, where "I" stands for interpretability. I-SPLIT builds upon the concept of *importance* or saliency of a neuron, which is related to the gradient it possesses with respect to the decision towards the correct class, for specific input. Importance is exploited with success in the Grad-CAM approach: Grad-CAM creates an input neuron saliency map that indicates which parts of an input image are more important for deciding a specific class. In particular, the Grad-CAM approach has been proved to be strongly dependent on the given trained model on which it runs (it passes the "sanity check" test), while other approaches do not, making it perfectly suited to our purposes.

I-SPLIT exploits Grad-CAM by creating for a given image multiple saliency maps, one for each layer of the network, which we rename as *importance maps*. Each layer-based importance map can be accumulated in a single value, accounting for how many important neurons it is formed by. Therefore, a single image gives rise to multiple CU*mulated* I*mportance* (CUI) values, one for each layer, that can be rearranged into a CUI curve. Multiple validation images create multiple CUI curves, that summed together do create a statistic of how much a layer is, in general, decisive for the right class. A layer that exhibits a high CUI value needs to be preserved, *i.e.*, the bottleneck should be injected right after this layer. Higher CUI values are predictors of high accuracy, and the ranking over CUI allows one to easily select the optimal splitting point.

Several are the advantages of I-SPLIT. The process is computationally efficient: the evaluation of the CUI values for N potential splitting points requires one step of backpropagation for each image of a given validation set, instead of N complete retraining sessions; in practice, for 10 potential splitting points to evaluate, I-SPLIT requires 150 minutes on a VGG network, instead of 6 hours needed to for the retraining. Moreover, we are able to discriminate, among layers of the same size which one is best suitable as the splitting point. Finally, and most interestingly, our approach shows that optimal splitting points are conditioned on the specific classes that the network is expected to process: indeed, specific classes trigger specific neurons, which in turn highlight layers which are possibly different. This provides the fresh-new concept of *class-dependent split decision*, which allows us to adapt the splitting point depending on the classes taken into account.

Furthermore, in [2], we propose Prometheus: a simulation software that eases the design of a distributed architecture with one or more DNNs operating inside. Other than accurately mimicking diverse communication protocols and memory requirements, Prometheus adds a unique feature, which is that of suggesting the proper configuration to simulate, and in particular how to cut the network to provide optimal performances in terms of accuracy. The rationale is to preserve the portions of the network where crucial decisions are taken. As a result, we show how the proposed framework unveils the performance bottlenecks of the architecture and facilitates the discovery of design options that satisfy users' requirements without the need for a complete real test setup.

Finally, we proposed a systematic review on tinyML. In this article [3], firstly we give a precise definition of what tinyML is, talking about their benefits and constraints. We then show how efficient deep learning can help in the realization of tinyML devices, and the seminal work there. Secondly, we list the use cases, models, datasets, and benchmarks that represent the current state-of-the-art about tinyML. In addition, we introduce the role of Industry 4.0 in the tinyML-IoT scenario. We believe that this review will serve as an information cornerstone for the tinyML research community and pave the way for further research in this direction.

III. Thesis Completion

In the near future, we plan to continue to improve on what we have achieved so far. Specifically, concerning I-SPLIT, future works will include further investigation of interpretability methods as a way to extract additional metrics to be used in the generation of the I-SPLIT curve. Regarding Prometheus, instead, we will focus on investigating specific techniques of tensor reconstruction to handle packet losses in an UDP transmission.

In addition, we will begin to analyze with new works the model optimization and target deployment stages of the Fig. 1. Specifically, dealing with the three main typologies of deep learning techniques for achieving this goal: *(i)* neural architecture search (NAS), *(ii)* quantized neural networks, *(iii)* network pruning, and *(iv)* structural efficiency.

Finally, another important field of study in order to achieve the objectives of this thesis is *neuromorphic computing*, *i.e.*, the use of very-large-scale integration (VLSI) systems containing electronic analog circuits to mimic neuro-biological architectures present in the nervous system.

References

[1] F Cunico, L Capogrosso, F Setti, D Carra, F Fummi, and M Cristani, *"International Conference on Pattern Recognition"*, ICPR 2022.

[2] L Capogrosso, F Cunico, A Lucchese, M Cristani, F Fummi, D Quaglia, *"International Conference on Computer-Aided Design"*, ICCAD 2022.

[3] L Capogrosso, F Cunico, F Fummi, M Cristani, *"IEEE Transactions on Pattern Analysis and Machine Intelligence"*, PAMI 2022.