Dynamic ConvNets on Tiny Devices via Nested Sparsity

Dynamic ConvNets on Tiny Devices via Nested Sparsity

Matteo Grimaldi, *Member, IEEE*, Luca Mocerino, *Member, IEEE*, Antonio Cipolletta, *Member, IEEE*, and Andrea Calimera, *Member, IEEE*

***Abstract*—This work introduces a new training and compression pipeline to build Nested Sparse ConvNets, a class of dynamic Convolutional Neural Networks (ConvNets) suited for inference tasks deployed on resource-constrained devices at the edge of the Internet-of-Things. A Nested Sparse ConvNet consists of a single ConvNet architecture containing N sparse sub-networks with nested weights subsets, like a *Matryoshka* doll, and can trade accuracy for latency at run time, using the model sparsity as a dynamic knob. To attain high accuracy at training time, we propose a gradient masking technique that optimally routes the learning signals across the nested weights subsets. To minimize the storage footprint and efficiently process the obtained models at inference time, we introduce a new sparse matrix compression format with dedicated compute kernels that fruitfully exploit the characteristic of the nested weights subsets. Tested on image classification and object detection tasks on an off-the-shelf ARM-M7 Micro Controller Unit (MCU), Nested Sparse ConvNets outperform variable-latency solutions naively built assembling single sparse models trained as stand-alone instances, achieving (i) comparable accuracy, (ii) remarkable storage savings, and (iii) high performance. Moreover, when compared to state-of-the-art dynamic strategies, like dynamic pruning and layer width scaling, Nested Sparse ConvNets turn out to be Pareto optimal in the accuracy vs. latency space.**

***Index Terms*—Neural Network Compression, Internet of Things, Latency-Quality Scaling, Micro Controller Units**

I. Introduction

THE ability to deploy fast Convolutional Neural Networks (ConvNets) at the edge of the Internet-of-Things (IoT) reflects the possibility of building ubiquitous intelligent services with high efficiency and privacy standards. In many IoT applications, the end-nodes are lightweight devices powered by tiny Micro Controller Units (MCUs), characterized by small form factor, minimal storage, and memory resources, i.e., few MBs of FLASH (1-2MB) and hundreds of KBs of RAM ($\leq 512kB$), and single-core CPUs clocked at few hundreds of MHz (100-400 MHz). To bridge the gap between such stringent hardware constraints and the computational and storage requirements of modern ConvNets, a considerable research effort has been lately spent seeking optimization strategies, like pruning [1]–[3], precision scaling [4], compact neural architectures [5]–[7], computational graph rewritings [8]–[10], and computational kernel tuning [9], [11], [12]. Despite the remarkable results achieved, those solutions follow a worst-case, accuracy-driven design and optimization strategy generating static ConvNets tailored for a specific setting. Static ConvNets show one main limitation, that is, they spend the same maximal effort in all situations, neglecting run-time changes that might appear due to variations in the external environmental conditions, the quality-of-service required by the user and the surrounding context, and the resources consumed by other software routines running in parallel on the same device. A speculative and perhaps more efficient approach would exploit contextual optimizations to minimize the average resource usage improving the information-processing capability. This is a relevant topic for IoT developers as small form factor, limited cost budgets, and severe power constraints limit the available resources on an IoT device. The adoption of dynamic optimization strategies allows the run-time environment to improve latency or save energy when the application can tolerate lower quality, achieving better performance on average [13]–[15]. For instance, a video surveillance system can reduce the classification effort when the scene is empty, lowering the energy consumption, then can switch into a more accurate but expensive mode only if something suspicious is detected. Alternatively, the latency of the inference task might change to meet different throughput requirements when the resource budget at the system or the application level gets reallocated [16]. These practical examples suggest that the availability of *dynamic* ConvNets capable of trading accuracy and computational costs at run time represents a valuable tool to raise the bar of efficiency for intelligent IoT applications.

Building a dynamic ConvNet encompasses the choice of a proper control mechanism to implement the latency-quality scaling at run time. Recent works proposed several architectural-level knobs, like the network depth [17] or the layers width [18], but although the relative ease of implementation, operating on the architecture of the model may be a too coarse option limiting the latency and accuracy trade-off. Moreover, it does not alleviate the pressure on the storage memory as the full model configuration, i.e., the one at the maximum width or maximum depth, might still be too large to fit into the FLASH memory. The availability of more fine-grain control knobs to modulate latency while keeping model footprint minimal is highly desirable indeed, and model

Manuscript received XX, 20XX; revised XX, 20XX; accepted XX, 20XX. Date of publication XXXX XX, 20XX; date of current version XXXX XX, 20XX. (Corresponding author: Andrea Calimera)

Matteo Grimaldi, Luca Mocerino, Antonio Cipolletta and Andrea Calimera are with the Department of Control and Computer Engineering, Politecnico di Torino, 10129, Italy (e-mail: matteo.grimaldi@polito.it, luca.mocerino@polito.it, antonio.cipolletta@polito.it, andrea.calimera@polito.it).

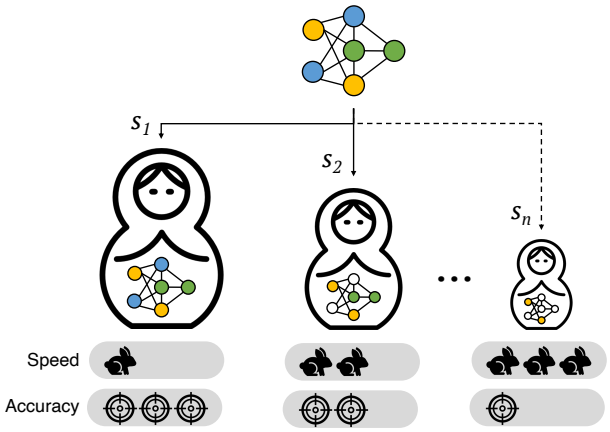


Fig. 1: A pictorial representation of a *Nested Sparse ConvNet*, a super-network containing N sub-networks with increasing value of sparsity ($s_1<s_2<...<s_N$): a low sparsity value corresponds to high accuracy, whereas a high sparsity value results in a faster inference process at the cost of lower accuracy.

sparsity is a good knob candidate. Sparse training is less prone to accuracy losses, and sparse models can be compressed via sparse encoding formats [19]. However, how to leverage the weight sparsity as the dynamic knob on compact ConvNets, e.g., the MobileNets [5], and how to deploy dynamic sparse ConvNets efficiently on tiny general-purpose cores are open issues.

To this end, we introduce a new class of dynamic ConvNets named *Nested Sparse ConvNets*. A Nested Sparse ConvNet is a convolutional deep neural network with a single weight-set that can be operated at N different configurations of increasing sparsity, resulting in a super-network containing N sparse sub-networks with nested weight-sets, like *Matryoshka* dolls as illustrated in Fig. 1. A low sparsity value corresponds to high accuracy, whereas a high sparsity value results in a fast yet less accurate inference. To let any ConvNet be transformed into a Nested Sparse ConvNet, this work proposes an end-to-end pipeline that comprises three main tools integrated over the full development stack:

- at training time, a gradient masking technique that properly routes the learning signals between the nested sparse networks guaranteeing convergence and high accuracy;
- at compile time, a sparse matrix compression format to fruitfully exploit the nested structure of the weights set and avoid computationally expensive decoding stages;
- at run time, dedicated compute kernels ensure efficient processing and fast switching among different sparse configurations with no additional latency cost.

To validate our proposal, we collected an extensive set of results using as benchmarks *ResNet9* [20] and two instances of *MobileNet* (V1 and V2) [5], [6] for two vision tasks, namely, image classification and object detection, deployed on an embedded system powered by an ARM Cortex-M7 MCU with $2MB$ of FLASH and $512KB$ of RAM. As it will be discussed in the experimental section, Nested Sparse ConvNets achieve an accuracy comparable to that of independently trained sparse models and outperform other scalable ConvNets obtained through existing dynamic methods, like dynamic pruning [21] and layer width scaling [18], thereby proving to be Pareto optimal in the accuracy vs. latency space.

The remainder of the paper is organized as follows. Section II reviews existing approaches to implement latency-quality scaling in ConvNets. Section III describes the proposed end-to-end pipeline consisting of the training methodology, the compression schema, and the sparse computational kernels. Section IV presents the collected experimental results through an extensive assessment of functional and extra-functional metrics. Section V discusses limitations and future works. Section VI concludes the work.

II. Related Works

This section offers a brief review of recent works on pruning strategies and compressed sparse storage formats for static ConvNets, as the proposed pipeline extends such techniques in a dynamic context. Then it describes state-of-the-art solutions for dynamic ConvNets and the limitations that our proposal aims to overcome.

Pruning. The existing methods differ in terms of the pruning policy they implement and the level of granularity at which they are applied [22]. In terms of policy, even if complex and rather elegant methods have been recently proposed, e.g., gradient- or Hebbian-based methods [19], those magnitude-based [23] are the preferred option in many modern training pipelines because of their reliability and ease of use. For what concerns the granularity, there exist three main classes. The *unstructured* pruning plays at the lower level, namely, on the individual weights of the model [23], offering a high degree of flexibility in reaching high accuracy targets. Such flexibility is paid at inference time when the potential savings brought by zeroed weights contrast with the regular code organization and memory access pattern preferred by general-purpose architectures. This issue is often solved with the aid of specialized hardware units that can accelerate the irregular flow, e.g., [24]. At a coarser granularity, *block* pruning techniques [25] group neighboring weights in specific patterns to decrease the indexing overhead and to ease the adoption of sparse compute kernels on general-purpose cores [26], [27]. At the coarsest level, *filter* pruning schemes drop entire convolutional filters [2] or cluster of convolutional filters [28], [29], achieving aggressive storage savings and speed-up at the cost of substantial accuracy loss due to fast information removal. Pruning can be combined with other optimization methods like neural architectural search [30], quantization [31], and computational graph rewriting [32] to enable different trade-offs between network accuracy and complexity.

Compressed Sparse Storage Format. Dealing with sparse arrays obtained by pruning irrelevant weights enables substantial memory savings by only storing the value and the position of the remaining non-zero entries. Many different sparse storage formats exist in literature [33] and their optimality is a function of the sparsity itself and the access pattern needed, e.g., random, streaming, or transposed access. For instance, to maximize the compression efficiency, a simple bitmap is preferable for low sparsity regimes, whereas coordinate-offset

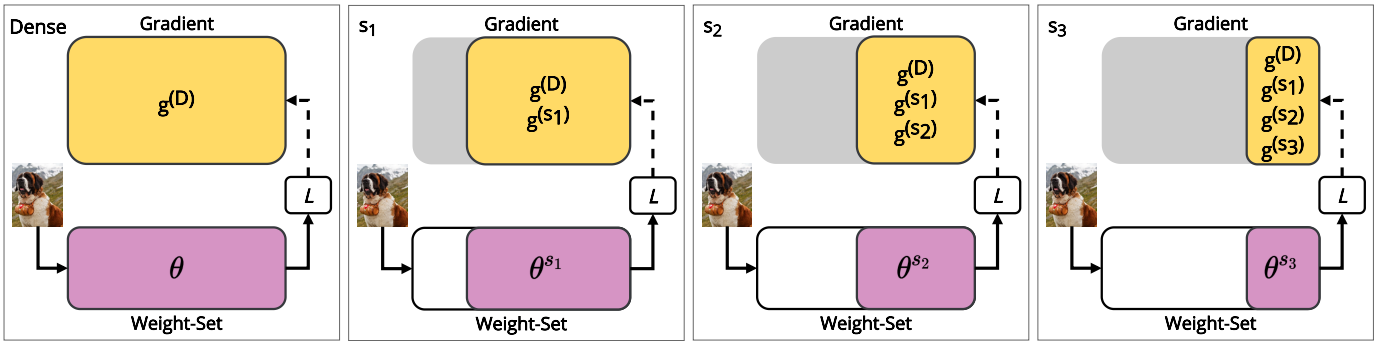


Fig. 2: Evolution of the training loop, from left to right. The full weight-set (θ) and the sub-nets (θ^{s_i}) get sorted and processed with an increasing order of sparsity value (i.e., $s_1 < s_2 < s_3$).

schemes (COO) are more suitable in high-sparse regimes [19]. Sparse storage formats like Compressed Sparse Row (CSR) or Columns (CSC) [26] allow fast row access, and so can be used to implement efficient sparse-matrix-vector and sparse-matrix-matrix operations.

Dynamic Topology. One way of building scalable ConvNets is to play with the architectural structure of the model, e.g., the width of the layers or the depth of the network. The *width multiplier* [5] was initially proposed as a static design option to scale the number of channels within each convolutional layer by a predefined ratio. Then, the authors of Slimmable Neural Networks [18] implemented a dynamic width scaling thanks to a *switchable* batch-norm introduced in the training procedure. Alternatively, the number of layers traversed during the forward pass can be modulated by attention modules or gating blocks [17] enabling a dynamic routing of the inner features, eventually with the addition of early-exit branches [34]. Notice that the total storage space is dictated by the underlying full-width model or the full-depth model plus the extra modules needed for controlling the topology at run time.

Dynamic Sparsity. Relying on the intuitive principle *the higher the sparsity, the lower the latency*, the authors of [21] proposed a training flow for deep neural models learned under concurrent sparsity levels. Despite the preliminary results conducted on Recurrent Neural Networks (RNNs) for Automatic Speech Recognition (ASR), known to be redundant and hence more reliable to pruning [35], we observed a substantial accuracy degradation on compact ConvNets for image classification tasks. Moreover, the training loop proposed in [21] is unaware of the resource usage and the achievable performance, leaving the minimization of the storage footprint and the deployment on real processing cores unsolved. Our proposal addresses both issues, offering Nested Sparse ConvNets as an effective solution for ConvNet architectures deployed on actual compute nodes for the IoT.

III. Building Nested Sparse ConvNets

A. Training

Training a Nested Sparse ConvNet is like concurrently learning N sub-networks with increasing sparsity encapsulated within a single set of weights θ. Collecting and composing the learning contributions coming from (and directed to) the many sparse sub-networks is a challenging problem as the learning of the weights shared among multiple sub-networks must be properly balanced to avoid sudden accuracy drops. For a better understanding, let's recall how pruning techniques for static ConvNets actually work. Early methods, e.g., [23], suggested that pruned weights must be bypassed during the gradient updates, but most recent works [22] introduced an improved pruning-while-training strategy that *regrow* lost connections achieving higher accuracy results. This is the starting point for our proposal. Managing the *regrowth* mechanism for a Nested Sparse ConvNet is not straightforward as the current "state" of a single weight (i.e., pruned or not-pruned) might differ among the N sub-networks, generating conflicts that may prevent convergence. To handle these constraints that may bubble up during training, we developed a novel method, referred to as *gradient masking*, precisely conceived to route the learning signals among the sub-networks.

An abstract and pictorial view of the dynamics governing the training steps of a Nested Sparse ConvNet using *gradient masking* is reported in Fig. 2. The example is for N=3 sub-networks of increasing sparsity $s_1 < s_2 < s_3$ and illustrates the run of a single training step. The three sub-networks are evaluated in sequence, following an increasing order of sparsity, from low (s_1) to high (s_3), as depicted within the three frames labeled as s_1, s_2, s_3. The first frame on the left (labeled as Dense) is for the full weight-set θ (i.e., sparsity s_0=0%). The dense training ensures stability, but the dense network is not included in the final model deployed for inference at run time. Within each frame, the corresponding sub-network undergoes a pruning-while-training procedure consisting of a forward (solid line) and a backward (dashed line) pass, with L as the training loss driving the learning procedure, and s_i as the sparsity constraint. Referring to the example in the picture, the four frames processed in sequence are iterated for a fixed number of training steps. The weights pruned within each frame to reach the desired sparsity s_i no longer contribute in the following stages, neither to the forward nor to the backward propagation; this is illustrated in Fig. 2 with the shadowed gray regions. For instance, the gradient computation from the sub-network with sparsity s_2, i.e., $g^{(s_2)}$, does not interfere with the previously computed gradients, i.e.,

Algorithm 1: Nested Sparse Training

```
1  Function main(steps, S, block_shape, optimizer):
2      for t in steps do
3          optimizer.zero_grad() // Ĝ = 0
4          soft_labels = forward (θ)
5          Ĝ += backward (θ)
6          if pruneStep (t) then
7              for s in S do
8                  M^s = getMask (θ, s, block_shape)
9                  θ^s = θ ∘ M^s
10                 forward(θ^s, soft_labels)
11                 ĝ^s = backward (θ^s)
12                 Ĝ += M^s ∘ ĝ^s // Gradient Masking
13             end
14         optimizer.step() // θ update
15     end
16     M = { getMask (θ, s, block_shape) for s in S }
17     return θ, M
18 Function getMask(θ, s, block_shape):
19     blocks = groupBlocks (θ, block_shape)
20     ŝ = ⌊s · |θ|⌋ · |θ|^{-1} // Sparsity Approximation
21     idx = rankBlocks (blocks, ŝ)
22     mask = ones_like (θ)
23     mask[idx] = 0
24 return mask
```

$g^{(s_1)}$. This allows the entire weight-set θ to evolve during the pruning-while-training process, while ensuring that each sparse sub-network is learned considering its own gradient contribution. The effect of the *gradient masking* is twofold: first, it allows less sparse (and possibly more accurate) sub-networks to influence the weights of the more sparse and weaker ones; second, it shields the more sparse (and hence less accurate) sub-networks, preventing abrupt changes in the learning curve.

The three nested weight-sets $\{\theta^{(s_1)}, \theta^{(s_2)}, \theta^{(s_3)}\}$, which are all contained in the whole weight-set θ, are identified, processed, and represented in the form of a set of binary masks $M = \{M^{(s_1)}, M^{(s_2)}, M^{(s_3)}\}$, with $\theta^{(s_i)} = \theta \circ M^{(s_i)}$[1]. This formulation can be generalized to any generic number of sub-networks N of increasing sparsity s_i, resulting into a set of N binary masks $M = \{M^{(s_1)}, ..., M^{(s_N)}\}$, and thus N weights subsets $\theta^{(s_i)}$. Each mask M^{s_i} is obtained through a magnitude-based rank and prune procedure over the weight-set θ. Weights with lower magnitude are pruned first, until reaching the desired sparsity s_i while enforcing the nesting of all weight-sets θ_i:

$$s_1 < s_2 < ... < s_N \;\Rightarrow\; \theta^{(s_1)} \supset \theta^{(s_2)} \supset ... \supset \theta^{(s_N)}. \quad (1)$$

The pseudo-code of the training loop is reported in Algorithm 1. It takes as inputs the set of sparsity levels $S = \{s_1, ..., s_N\}$ and the *block_shape* ($m \times n$), returning the weight-set θ and the set of masks $M = \{M^{(s_1)}, ..., M^{(s_N)}\}$. The training loop alternates dense and sparse training epochs, according to a fixed scheduler (line 6). At the beginning of each epoch, the gradient is zeroed (line 3), then the forward and backward passes are performed on the weight-set θ (lines 3-5) as a whole (the dense training frame in Fig.2). The set of weights is directly updated using the gradient value (line 14) during the dense steps. During the sparse training steps (the s_i frames in Fig.2), for each sparsity level s (line 7), the `getMask` function generates a mask M^s (line 8). This mask is multiplied point-wise with θ to extract the sparse sub-network θ^s (line 9) and then used to complete the forward and backward passes (lines 10-11). For the sparse sub-networks, the predictions of the dense model (line 4) are used as soft labels (line 10) as a form of in-place distillation [36]. At last, the local gradient $\hat{g^s}$ relative to the sub-network θ^s is masked and merged with the previous gradient contributions (line 12). Once the contributions of each sub-network are accumulated in the global gradient $\hat{G}$, the weight-set θ is updated (line 14). At the end of the training, both the weight-set θ and the set of nested masks M are returned (lines 16-17).

The `getMask` function used to obtain the binary mask $M^{(s_i)}$ under a given sparsity value s_i works as follows. First, weights are grouped into blocks of shape $m \times n$ (line 19), where m is in the output-channels axis. Second, the value of the target sparsity is approximated to the closest value that guarantees an integer number of blocks to prune (line 20). Third, blocks are ranked according to their magnitude (L^2-norm) through the `rankBlocks` function that returns the position (idx) of the sorted weights in descending order (line 21). Fourth, the least important $\hat{s_i} \cdot |\theta|$ weights are pruned by setting to zero their values and the corresponding items in the binary mask $M^{(s_i)}$ (lines 22-23).

B. Compression

Fig. 3 illustrates an example of the proposed sparse matrix compression format, named *NestedCSR*, for a nested model trained for three generic sparsity levels $s_1 < s_2 < s_3$ and using a 1×2 block shape. It is worth emphasizing that the compression format is general and can be used for any number of sparsity levels or block sizes. At the lower sparsity level (s_1), the matrix comprises the red, green, and blue non-zero blocks; at the medium sparsity level (s_2), the red and green blocks; at the high sparsity level (s_3), the red blocks only. As shown in the picture, the three configurations are a composition of three disjoint sparse matrices, and this is precisely the property exploited by *NestedCSR*. Each sparse sub-set is compressed using a block CSR format [26]: the *nz-values* array stores the values of the non-zero blocks in row-major order, the *nz-iidx* array stores the number of non-zero blocks on each row, and the *nz-jidx* the column position of each non-zero blocks. The three arrays of each sparse sub-set are concatenated row-wise, from the most sparse to the least sparse (from red to blue in Fig. 3). We designed our compression format on top of block CSR, as this perfectly suits our case scenario: a wide range of medium-to-high sparse matrices [19]. Conversely, other sparse formats, like COO, work better for matrices with high sparse regimes ($\geq 95\%$).

The footprint of a block-sparse matrix W with dimensions $R \times C$ encoded through *NestedCSR* depends on the block shape

[1] $\circ$ indicates the Hadamard product between two matrices.

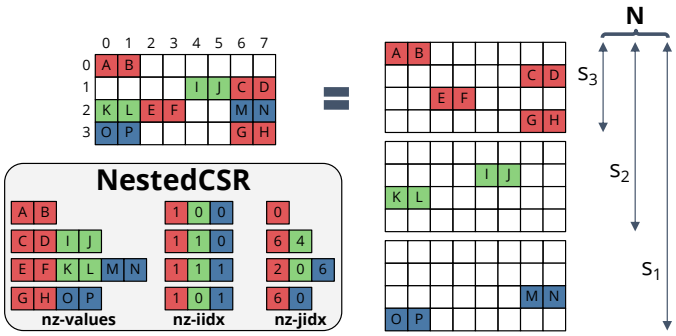


Fig. 3: Example of the proposed NestedCSR format applied to a 1×2 block sparse matrix W that can work in three sparsity levels $\{s_1, s_2, s_3\}$.

$m\times n$ and the number of sparsity levels (N). The following equation describes the size of the array:

$$\begin{aligned} |\textit{nz-values}| &= (1-s_{min}) \cdot R{\cdot}C \\ |\textit{nz-iidx}| &= N \cdot R \\ |\textit{nz-jidx}| &= (1-s_{min}) \cdot \frac{R{\cdot}C}{n{\cdot}m} \end{aligned} \tag{2}$$

As can be inferred from the equations, the amount of storage memory is weakly affected by the number of nested configurations. The number of sparse sub-networks (N) affects the size of *nz-iidx*, which is usually negligible compared to that of the other two arrays. Therefore, the overall memory footprint gets defined by the smallest adopted sparsity value (s_{min}), which is crucial for effective and efficient deployment.

To accelerate the processing of a nested and compressed sparse layer on a general-purpose core, we implemented a custom compute kernel that performs a matrix multiplication $C = A \cdot B$ between a sparse matrix (A) encoded using the *NestedCSR* format and a dense matrix (B), as shown in fig. 4a. The kernel handles both fully connected and convolutional layers, adopting a convolution-as-GEMM implementation for the latter [26], [37].

Like in classical CSR-based sparse matrix multiplication, the whole operation is a sequence of small matrix operations between M columns of the dense matrix and 1 row of the sparse matrix as shown in Fig. 4b. Such implementation reduces the cost of the indirection process needed to access one element of the sparse matrix across multiple multiply-and-accumulate (MAC) operations. Specifically, it was experimentally found out that M=4 represents a good trade-off between data-reuse and register pressure on small MCUs. Following the NestedCSR format, since a single row of the sparse matrix is encoded as N sparse components, the multiplication is decomposed as N sparse operations at most, as shown in fig 4c. Depending on the sparsity value s_i selected at run time, only a fraction of such operations is processed, exploiting the model sparsity as a practical knob to reduce the overall compute workload. In the proposed implementation, there is no additional cost from switching the sparsity level, as the kernel can be specialized at compile time and then called at run time based on the input s_i of the procedure.

IV. Results

A. Experimental Set-up

1) Tasks, Datasets, and ConvNets: The proposed pipeline was tested and evaluated on image classification (IC) and object detection (OD) tasks using the following datasets.

a) CIFAR-10/100 (IC) [38]: $60k$ 32×32 RGB images annotated with 10/100 labels and split into $45k$ samples for training, $5k$ for validation, and $10k$ for testing.

b) PASCAL VOC (OD) [39]: 15870 RGB images picked from the 2007 and 2012 PASCAL Visual Object Classes Challenge, counting of 37813 objects annotated with 20 different labels. As suggested in [40], VOC07 and VOC12 *trainval* data were used for training, using VOC07 for testing. We reduced the number of classes to the top-10 labels recognized by the full-scale model. The image resolution was re-scaled to 160×160 with a bi-linear interpolation; this is mandatory due to the strict memory constraints of the target MCU ($512KB$ of RAM, $2MB$ of FLASH).

The ConvNets used as benchmarks are lightweight models suitable for the IoT segment and hence portable onto tiny cores. Specifically, we operated ResNet (ResNet9) [20] for IC on CIFAR-100, MobileNetV1 [5] for IC on CIFAR-10, MobileNetV2 [6] as backbone of the Single Shot Detector (SSD) [40].

2) Training: The training procedure for the IC tasks was driven by the SGD optimizer (momentum 0.9, weight decay 0.0005) for 300 epochs with batch size 128. The learning rate followed a cosine annealing schedule starting from 0.05. The same procedure applied for training the SSD, except for the batch size which was set to 32. Images were flipped and rotated for data augmentation on the IC tasks, whereas we replicated the strategy presented in [40] for OD. All networks were trained from scratch and initialized with random weights. Each training experiment was repeated three times using different seeds, and the collected results were averaged. For what concerns the sparse networks, we used S=$\{70\%$, 80%,$90\%\}$ as the sparsity set and a constant block shape 1×2 for each sparsity. Finding the optimal set S to achieve the best accuracy, latency, and storage trade-off is out of the scope of this work. As suggested by previous works on sparse networks [27], the first layer of each ConvNet under test is kept dense. The training algorithm was implemented within the PyTorch framework ($v1.5.1$) and accelerated with a single consumer graphic card by NVIDIA (Titan Xp).

In the remaining sections we refer to *Dense* as the dense baseline network, *Single Sparse* as the model optimized for a single sparsity level [23], *Nested Sparse ConvNets* for our proposal, *Slimmable* as the dense dynamic model obtained by layers width scaling [18], and *DSNN* as the dynamic sparse model [21]. For *Slimmable* we adopted the official repository[2], whereas for *DSNN* we used an in-house implementation as no open-source code was available at the time of this writing.

3) Deployment: The collected performances refer to an off-the-shelf NUCLEO-F767ZI board powered by an ARMCortex-M7 MCU operating at $216MHz$. The board hosts 512KB of on-chip SRAM and $2MB$ of FLASH. An in-house extension of the CMSIS-NN library v.5.6.0 [37]

[2]https://github.com/JiahuiYu/slimmable_networks

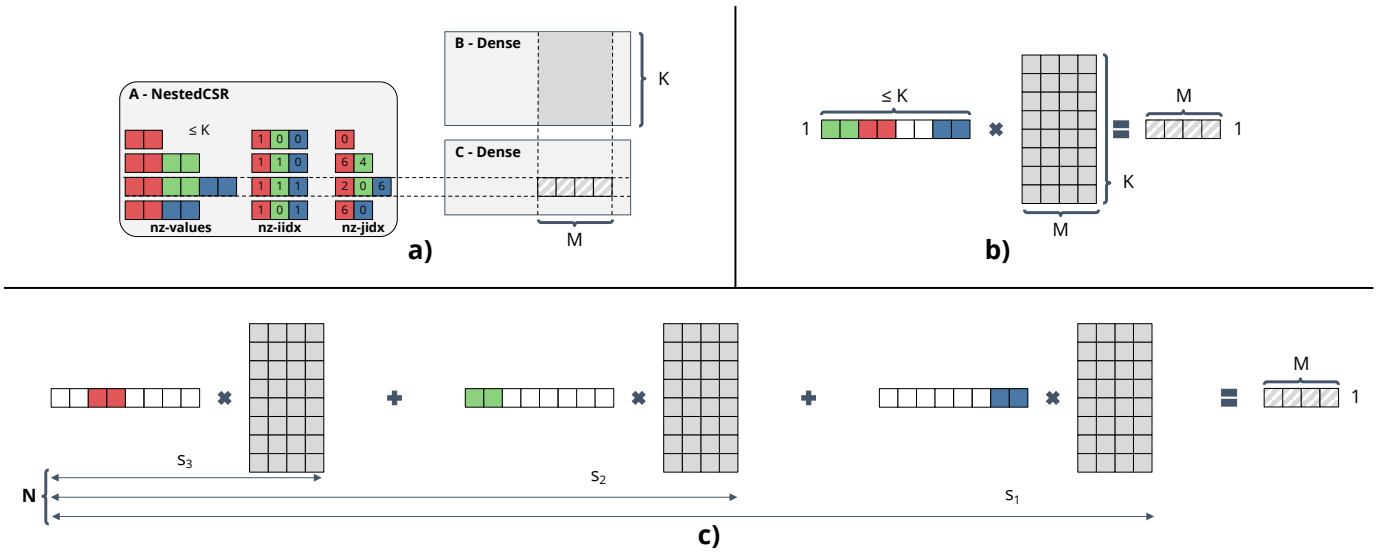


Fig. 4: Example of the proposed compute kernel performing a sparse matrix-matrix multiplication (a) between a 1×2 block sparse matrix A encoded using the NestedCSR format and a dense matrix B with K rows. The entire matrix multiplication is decomposed as a sequence of smaller operations (b) between 1 row of A and M columns of B. Such inner operation is carried out as at most N operations (c) depending on the selected sparsity value $s_i \in \{s_1, s_2, s_3\}$.

TABLE I: Accuracy results for MobileNetV1 on CIFAR-10. Best results for each sparsity level are highlighted in bold.

Training	Sparsity [%]	Accuracy Top-1 [%]			
		w=1.00	w=0.75	w=0.50	w=0.25
Dense	0	90.08	89.35	88.32	85.31
Single Sparse	70	89.70	**88.56**	87.27	**83.32**
	80	89.02	88.13	**87.04**	73.22
	90	**88.81**	86.02	75.20	57.88
DSNN [21]	70	86.30	86.21	84.09	78.84
	80	86.42	85.96	83.69	76.10
	90	85.49	84.62	81.78	72.22
Ours	70	**89.90**	88.48	**87.55**	83.29
	80	**89.20**	**88.24**	86.95	**82.12**
	90	88.50	**87.03**	**85.86**	**78.20**

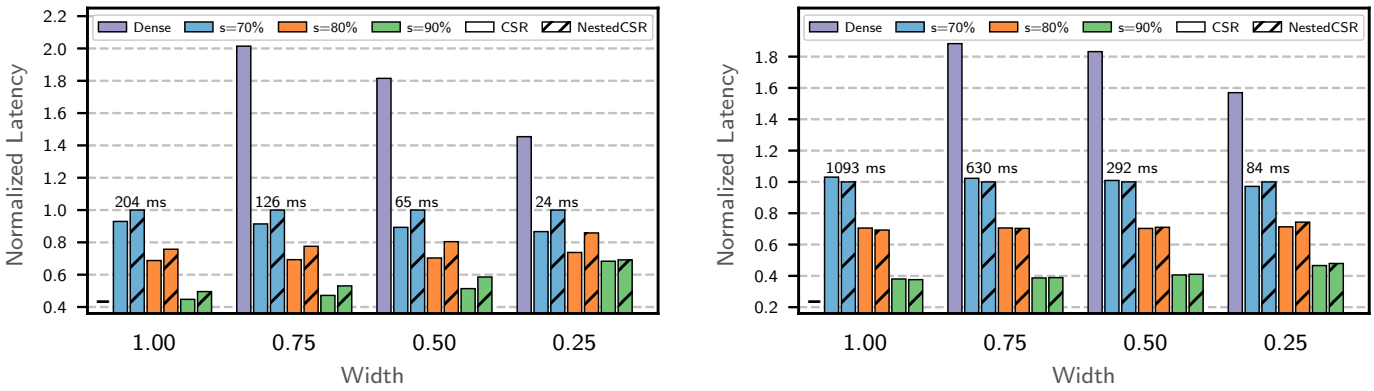
was integrated with the sparse matrix multiplication kernels described in the previous section, with a 1×2 block-shape to exploit the Single Instruction Multiple Data media accelerator of the M7 core [26]. In compliance with the arithmetic requirements of the CMSIS-NN library, the ConvNets were quantized to 8-bit using a layer-wise symmetric binary scaling [4]. We adopted the GNU Arm Embedded Toolchain (version 6.3.1) for cross-compilation.

B. Training Evaluation

To assess the quality and generalization properties of the proposed nested training, we analyzed the accuracy achieved over the IC tasks by ConvNet architectures of decreasing information capacity, that is, rescaled by means of the width multiplier factor $w \in \{1.00, 0.75, 0.50, 0.25\}$. Such a scaling operation must not be confused with the dynamic width scaling of [18], which is discussed later in Section IV-D. The results are collected in Tab. I and Tab. II.

Nested Sparse vs. Single Sparse Training. Intuitively, training a network for a single sparsity level should be a best-case scenario because the parameters get optimized for one specific sparsity level only. On the other hand, training a Nested Sparse ConvNet encompasses the concurrent optimization of multiple sub-networks with shared weights. Nonetheless, Nested Sparse ConvNets outperform individually trained sparse models in many cases, and when they achieve a lower accuracy, the gap is rather low: the worst-case accuracy drop is 0.31% for MobileNetV1 and 0.96% for ResNet9. The gradient masking technique attains high accuracy indeed, even when classical single sparsity pruning does not. For instance, the single sparse MobileNetV1@w=0.25 with s=90% suffers from a drastic accuracy drop (57.88%), whereas the Nested Sparse model is 20.32% more accurate (78.20%), closing the gap with the least sparse configurations (83.29% with s=70%). The gradient masking technique also improves the least sparse instances due to the proper involvement of the dense model in the training loop. This can be inferred from the results collected on the Nested Sparse ResNet9@w=0.75 with s=70%, which shows $\approx 1\%$ more accurate than its single sparse model counterpart, hence closer to the dense model.

Nested Sparse vs. Dynamic Sparse NN (DSNN) Even though training DSNNs has proven effective on RNNs for ASR [21], our results reveal quality drops on tiny ConvNets for IC tasks. The DSNN training on MobileNetV1 is 3.40% less accurate than the single sparse configuration and 13.65% less on the ResNet9. Except for ResNet9@w=1.00 with s=90%, Nested Sparse ConvNets outperform DSNNs, with an increasing gap for smaller networks with lower width and higher sparsity (the highest gap is for ResNet9@w=0.25 with s=90%).



(a) MobileNetV1- CIFAR10 (b) ResNet9 - CIFAR100

Fig. 5: Latency values normalized for each width to the NestedCSR@s=70%. The latency of the dense model at w=1.00 is not shown as it exceeds the FLASH memory of the adopted device (2MB).

TABLE II: Accuracy results for ResNet9 on CIFAR-100. Best results for each sparsity level are highlighted in bold.

Training	Sparsity [%]	Accuracy Top-1 [%]			
		w=1.00	w=0.75	w=0.50	w=0.25
Dense	0	73.78	72.24	69.66	63.05
Single Sparse	70	72.93	71.09	68.29	**58.90**
	80	72.61	70.90	67.72	**57.40**
	90	**72.15**	**69.98**	65.04	52.15
DSNN [21]	70	72.9	70.48	63.38	45.25
	80	72.83	69.70	62.48	44.69
	90	71.62	67.56	60.15	40.92
Ours	70	**73.56**	**72.04**	**68.82**	58.70
	80	**72.94**	**71.05**	**68.38**	57.30
	90	71.19	69.59	**65.92**	**52.93**

C. Encoding Format Evaluation

Tab. III reports the storage profiles for ResNet9 and MobileNetV1, showing that Nested Sparse ConvNets achieve remarkable savings. Three nested sparse configurations require as low as $1016kB$ (54% smaller than the dense baseline) for ResNet9@w=1.00, and $1464kB$ (53% smaller) for MobileNetV1@w1.00. Interestingly, a Nested Sparse ConvNet takes almost the same storage of its least sparse configuration. For instance, encoding a single instance with sparsity 70% using block CSR [26] takes $1014KB$ for ResNet9@w=1.00 (a mere $2kB$ less than NestedCSR) and $1458kB$ for MobileNetV1@w=1.00 ($6kB$ less than NestedCSR). The models rescaled to the other widths follow the same trend, confirming the effectiveness of the NestedCSR format across a wide set of model configurations.

The performance attainable with the NestedCSR format further improved with the aid of the custom-designed compute kernels. Fig. 5 reports a comparative analysis for ResNet9 and MobileNetV1, both dense and sparse versions, using a classical CSR [26] and the proposed NestedCSR. The sparse kernels introduce a substantial speed-up compared to the dense versions as expected, but even more remarkable, they make Nested Sparse ConvNets reach comparable performance to

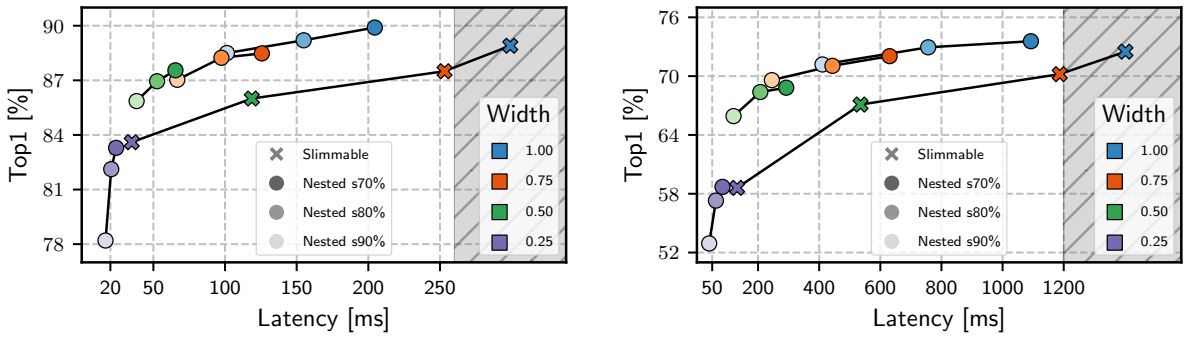
TABLE III: Storage footprint of ResNet9 trained on Cifar100 and MobileNetV1 trained on CIFAR10. *Single* sparse models encoded with a block CSR [26]. *Nested* sparse models encoded with the proposed block NestedCSR format.

Model	Method	Sparsity [%]	Storage [KB]			
			w=1.00	w=0.75	w=0.50	w=0.25
MobileNetV1	Dense	0	3132	1774	800	208
	Single	70	1458	834	384	106
	Nested	{70, 80, 90}	1464	839	387	108
ResNet9	Dense	0	2232	1259	562	143
	Single	70	1014	575	260	68
	Nested	{70, 80, 90}	1016	576	260	68

single sparse ConvNets. Referring to ResNet9, nested kernels perform slightly better than single sparse kernels (1.83% on average) for high widths (w=1.00 and w=0.75), and show some overhead for low width (4.04% in the worst case). For MobileNetV1, the nested kernels perform moderately worse (10.91% slower on average) and the overhead increases more notably for more sparse and smaller networks (up to 14.08% in the worst case). The different internal structure of ResNet9 and MobileNetV1 is the source of such gap. In MobileNetV1, there are many convolutional layers, but only the 1×1 pointwise layers are sparsified, whereas in ResNet9, there are fewer convolutional layers, but they are all sparse and also show more channels with larger kernels (3×3). Despite those penalties, nested kernels still preserve the latency gain brought by sparsity. Moreover, a naive implementation of multiple sparse networks stored as separate instances would not fit on the device due to the memory constraints, an issue we overcome by means of our nested solution.

D. Latency-Quality Scaling

Fig. 6 depicts the latency vs. accuracy trade-off achievable by Nested Sparse ConvNets. The best dynamic behavior is for larger widths. Looking at MobileNetV1@w=1.00, an increase of sparsity from 70% to 90% has minimal effect on accuracy (1.4%), but the speed-up is substantial: up to 51% of latency reduction. ResNet9@w=1.00 follows the same trend (Fig. 6b), where a higher sparsity level improves latency by 62% with a



(a) MobileNetV1- CIFAR10 (b) ResNet9 - CIFAR100

Fig. 6: Latency-accuracy scaling for Slimmable ConvNets and Nested Sparse ConvNets. Grey area shows the unfeasible solution space for the adopted MCU, i.e., FLASH footprint $> 2MB$.

moderate effect on accuracy (2.37% loss). Rescaling the model width makes the trade-off slightly worse as smaller ConvNets are less resilient to sparsity. As a result, the accuracy gap increases and the latency speed-up reduces when the ConvNets architecture shrinks down. Nonetheless, for the smaller nets (w=0.25), the accuracy drop of 5.09% for ResNet9 and 5.77% for MobileNetV1 come with a large speed-up, 52% and 31% respectively.

Fig. 6 also shows the dynamic behavior of ConvNets optimized with the *Slimmable* approach [18] offering a direct comparison with our approach. Slimmable networks at maximum width w=1.00 get too large to fit into the FLASH memory ($2MB$), and only three configurations out of four can be deployed on-device. Thanks to the proposed training and compression pipeline instead, Nested Sparse ConvNets meet the memory constraint even at full scale (w=1.00). Except for the smallest width (w=0.25), Nested Sparse ConvNets at s=70% and s=80% turn out to be more accurate and faster than the slimmable models. The *Pareto analysis* reveals that the three rescaled Nested Sparse ConvNets (w=$\{0.75, 0.50, 0.25\}$) outperform the slimmable counterparts, originating eight Pareto optimal implementations that, if stored together, consume less storage than a slimmable model. Precisely, $904kB$ for ResNet9 and $1334kB$ for MobileNetV1, that is, 28% and 25% less than the deployable configurations of the *slimmable* models ($w \leq 0.75$). The downside is that a single Nested Sparse ConvNet presents a moderate scaling capacity compared to a slimmable model, which is intuitive as the sparsity acts as a fine-grain control knob both on accuracy and latency. However, the low storage footprint paves the way to an attractive hybrid solution, where the width multiplier serves as a static knob complementary to the dynamic sparsity.

It is worth emphasizing that other scalable training methods, e.g., *EfficientNet* [41], *TinyNet* [42], and *OFA* [43], play statically, i.e., at *design time*, on the input resolution and on the topology of the model architecture, i.e., on the width, depth, and kernel sizes, with the aim to achieve a higher accuracy with the same resource budget. Such scaling methods are of utter importance to the design of efficient ConvNets, but their purpose differs from ours. We demonstrated that tweaking at *run time* the accuracy-latency trade-off via sparsity is feasible even with a reduced storage footprint, as only one compressed weight-set must be stored on-device for a Nested Sparse ConvNet. Therefore, our solution can be used on top of existing neural architectural design methods.

TABLE IV: SSD-MobileNetV2. Best results for each sparsity level are highlighted in bold.

Training	Sparsity [%]	w=0.50 mAP [%]	w=0.50 Storage [kB]	w=0.50 Latency [ms]	w=0.35 mAP [%]	w=0.35 Storage [kB]	w=0.35 Latency [ms]
Dense	0	68.32	869	1549	63.42	523	998
Single Sparse	70	66.01	508	1080	60.58	329	752
	80	62.72	407	972	55.20	274	689
	90	29.40	306	862	23.06	219	625
Ours	70	**68.30**		1225	**63.12**		883
	80	**66.37**	514	1103	**61.03**	334	807
	90	**60.33**		951	**55.84**		712

E. Object Detection

This last subsection aims to show the generalization capability of our approach on tasks different from image classification. Specifically, we evaluated a Nested MobileNetV2 on a bounding-box detection task. The results reported in Tab. IV refer to configurations at w=$\{0.50, 0.35\}$, which are those meeting the FLASH memory constraint for our target MCU. The Nested Sparse object-detector gets more accurate than the sparse models trained as separate instances. For the most sparse configurations (i.e., s=90%), it is 31.85% more accurate (average over the two widths), confirming the stability of the proposed training loop. With regard to the latency, the conclusions brought by the image classification tasks do hold also here: the sparse models are faster than the dense models and the nested configurations are slightly slower than single sparse instances. Also in this case, a hybrid solution build through a superimposition of width scaling and nested sparsity enables a wider latency-accuracy working space (from configuration w=0.50 and s=70% to w=0.35 and s=90%) $\Delta\text{Top-1}/\Delta L = 12.46(\%)/368(ms)$ while cumulatively occupying $848kB$, which is less than the single dense model at w=0.50.

V. Current Limitations and Future Works

The proposed training and compression pipeline enables the use of model sparsity as a dynamic knob on tiny off-the-shelf devices. Although the experimental assessment revealed that Nested Sparse ConvNets outperform other dynamic strategies while occupying a smaller storage footprint, some issues have not been addressed in the current version of the work. First, the choice of the sparsity levels is fixed manually prior to training. However, as the trade-off accuracy vs. latency enabled by sparsity depends on the model architecture and the task, designing the optimal set of sparsity values is not trivial and should be automated. Second, although using the same sparsity ratio for all layers of the network was proven effective in previous works [27], exploiting the effects that different layers have on both accuracy [44], [45] and latency [46] may lead to new Pareto solutions. Thus, a possible future development aimed at overcoming such limitations can integrate an automatic search engine (like those presented in [47], [48], [49]) in the proposed pipeline such that multiple sparse configurations are sampled and tested at training time to optimize storage, latency, and accuracy simultaneously.

VI. Conclusions

Nested Sparse ConvNets represent a novel class of dynamic models conceived to trade-off latency with accuracy at run time leveraging sparsity as the scaling knob. We introduced a novel training procedure capable of reaching highly accurate predictions, and, in conjunction with a new storage format and a library of custom compute kernels, it enables the deployment of elastic ConvNets on tiny off-the-shelf devices. An extensive experimental assessment on tiny visual computing tasks deployed on a low-end IoT node powered by an ARM M7 MCU reveals that Nested Sparse ConvNets are processed efficiently, outperform state-of-the-art dynamic strategies achieving optimality in the accuracy-latency objective space, and can therefore represent a new alternative for widening the adoption of energy-efficient adaptable computer vision tasks at the edge of the IoT.

References

[1] E. Russo, M. Palesi, S. Monteleone, D. Patti, A. Mineo, G. Ascia, and V. Catania, "Dnn model compression for iot domain-specific hardware accelerators," *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6650–6662, 2022.

[2] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[3] M. Grimaldi, V. Peluso, and A. Calimera, "East: Encoding-aware sparse training for deep memory compression of convnets," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020, pp. 233–237.

[4] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[7] J. Lin, W.-M. Chen, Y. Lin, j. cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 11 711–11 722.

[8] A. Cipolletta and A. Calimera, "Dataflow restructuring for active memory reduction in deep neural networks," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 114–119.

[9] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "Memory-efficient patch-based inference for tiny deep learning," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 2346–2358.

[10] B. H. Ahn, J. Lee, J. M. Lin, H.-P. Cheng, J. Hou, and H. Esmaeilzadeh, "Ordering chaos: Memory-aware scheduling of irregularly wired neural networks for edge devices," in *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., vol. 2, 2020, pp. 44–57.

[11] A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, "Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 871–875, 2020.

[12] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "Fann-on-mcu: An open-source toolkit for energy-efficient neural network inference at the edge of the internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4403–4417, 2020.

[13] T. Yang, H. Feng, S. Gao, Z. Jiang, M. Qin, N. Cheng, and L. Bai, "Two-stage offloading optimization for energy–latency tradeoff with mobile edge computing in maritime internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5954–5963, 2020.

[14] M. Alioto, V. De, and A. Marongiu, "Energy-quality scalable integrated circuits and systems: Continuing energy scaling in the twilight of moore's law," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 4, pp. 653–678, 2018.

[15] M. Alioto, "Energy-quality scalable adaptive vlsi circuits and systems beyond approximate computing," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 127–132.

[16] L. Mocerino and A. Calimera, "Fast and accurate inference on microcontrollers with boosted cooperative convolutional neural networks (bc-net)," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 1, pp. 77–88, 2021.

[17] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[18] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *International Conference on Learning Representations*, 2018.

[19] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks," *Journal of Machine Learning Research*, vol. 22, no. 241, pp. 1–124, Sep. 2021.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2016, pp. 770–778.

[21] Z. Wu, D. Zhao, Q. Liang, J. Yu, A. Gulati, and R. Pang, "Dynamic sparsity neural networks for automatic speech recognition," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 6014–6018.

[22] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *arXiv e-prints*, vol. arXiv:1902.09574, 2019.

[23] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015.

[24] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 27–40.

[25] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. van den Oord, S. Dieleman, and

K. Kavukcuoglu, "Efficient neural audio synthesis," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 2410–2419.
[26] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 548–560, jun 2017.
[27] E. Elsen, M. Dukhan, T. Gale, and K. Simonyan, "Fast sparse convnets," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
[28] C. Gamanayake, L. Jayasinghe, B. K. K. Ng, and C. Yuen, "Cluster pruning: An efficient filter pruning method for edge ai vision applications," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 802–816, 2020.
[29] V. Radu, K. Kaszyk, Y. Wen, J. Turner, J. Cano, E. J. Crowley, B. Franke, A. Storkey, and M. O'Boyle, "Performance aware convolutional neural network channel pruning for embedded gpus," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 2019, pp. 24–34.
[30] I. Fedorov, R. M. Navarro, H. Tann, C. Zhou, M. Mattina, and P. N. Whatmough, "UDC: unified DNAS for compressible tinyml models," *CoRR*, vol. abs/2201.05842, 2022.
[31] M. van Baalen, C. Louizos, M. Nagel, R. A. Amjad, Y. Wang, T. Blankevoort, and M. Welling, "Bayesian bits: Unifying quantization and pruning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 5741–5752.
[32] G. Li, X. Ma, X. Wang, L. Liu, J. Xue, and X. Feng, "Fusion-catalyzed pruning for optimizing deep learning on intelligent edge devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3614–3626, 2020.
[33] U. W. Pooch and A. Nieder, "A survey of indexing techniques for sparse matrices," *ACM Comput. Surv.*, vol. 5, no. 2, p. 109–133, jun 1973.
[34] X. Zhu and M. Bain, "B-cnn: branch convolutional neural network for hierarchical classification," *arXiv preprint arXiv:1709.09890*, 2017.
[35] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring sparsity in recurrent neural networks," *arXiv preprint arXiv:1704.05119*, 2017.
[36] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.
[37] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.
[38] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009.
[39] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.
[40] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 21–37.
[41] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 6105–6114.
[42] K. Han, Y. Wang, Q. Zhang, W. Zhang, C. Xu, and T. Zhang, "Model rubik's cube: Twisting resolution, depth and width for tinynets," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 353–19 364, 2020.
[43] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020.
[44] C. Zhang, S. Bengio, and Y. Singer, "Are all layers created equal?" in *ICML 2019 Workshop on Identifying and Understanding Deep Learning Phenomena*, 2019.
[45] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 2498–2507.
[46] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
[47] J. Yu and T. Huang, "Autoslim: Towards one-shot architecture search for channel numbers," *arXiv preprint arXiv:1903.11728*, 2019.
[48] T.-W. Chin, A. S. Morcos, and D. Marculescu, "Joslim: Joint widths and weights optimization for slimmable neural networks," in *Machine Learning and Knowledge Discovery in Databases. Research Track*, N. Oliver, F. Pérez-Cruz, S. Kramer, J. Read, and J. A. Lozano, Eds. Cham: Springer International Publishing, 2021, pp. 119–134.
[49] G. Li, X. Ma, X. Wang, H. Yue, J. Li, L. Liu, X. Feng, and J. Xue, "Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning," *Journal of Systems Architecture*, vol. 124, p. 102431, 2022.

Matteo Grimaldi received Ph.D. in Computer Engineering from the Politecnico di Torino in 2021, where he is currently working as a postdoctoral researcher. His research interests include efficient algorithms and design strategies for resource-driven compression and optimization of deep learning models.

Luca Mocerino received M.Sc. in Computer Science at Politecnico di Torino (2017). There, since 2018, he has been a research assistant and Ph.D. Candidate in Computer and Control Engineering with the EDA group. His research interests mainly focus on hardware and software optimizations for deep learning algorithms.

Antonio Cipolletta received the M.Sc. and the Ph.D. in Computer Engineering from Politecnico di Torino in 2018 and 2022, respectively. He also received the M.Sc. in Electrical and Computer Engineering from the University of Illinois at Chicago (2019). His main research interests focus on hardware-software co-design for deep learning acceleration and parallel computing systems.

Andrea Calimera received the M.Sc. degree in Electronic Engineering and the Ph.D. degree in Computer Engineering both from Politecnico di Torino. He is currently an Associate Professor of Computer Engineering at Politecnico di Torino. His research interests cover the areas of design automation of digital circuits and embedded systems with emphasis on optimization techniques for low-power and reliable ICs, dynamic energy/quality management, logic synthesis, design flows and methodologies for emerging computing paradigms.