Multi-Complexity-Loss DNAS for Energy-Efficient and Memory-Constrained Deep Neural Networks

Multi-Complexity-Loss DNAS for Energy-Efficient and Memory-Constrained Deep Neural Networks

Matteo Risso†, Alessio Burrello*, Luca Benini*, Enrico Macii‡, Massimo Poncino†, Daniele Jahier Pagliari†
*Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy
†Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy
‡Inter-university Department of Regional and Urban Studies and Planning, Politecnico di Torino, Turin, Italy
Corresponding Email: matteo.risso@polito.it

***Abstract*—Neural Architecture Search (NAS) is increasingly popular to automatically explore the accuracy versus computational complexity trade-off of Deep Learning (DL) architectures. When targeting tiny edge devices, the main challenge for DL deployment is matching the tight memory constraints, hence most NAS algorithms consider model size as the complexity metric. Other methods reduce the energy or latency of DL models by trading off accuracy and number of inference operations. Energy and memory are rarely considered simultaneously, in particular by low-search-cost Differentiable NAS (DNAS) solutions.**

We overcome this limitation proposing the first DNAS that directly addresses the most realistic scenario from a designer's perspective: the co-optimization of accuracy and energy (or latency) *under a memory constraint*, determined by the target HW. We do so by combining two complexity-dependent loss functions during training, with independent strength. Testing on three edge-relevant tasks from the MLPerf Tiny benchmark suite, we obtain rich Pareto sets of architectures in the energy vs. accuracy space, with memory footprints constraints spanning from 75% to 6.25% of the baseline networks. When deployed on a commercial edge device, the STM NUCLEO-H743ZI2, our networks span a range of 2.18x in energy consumption and 4.04% in accuracy for the same memory constraint, and reduce energy by up to 2.2× with negligible accuracy drop with respect to the baseline.

***Index Terms*—Deep Learning, TinyML, Energy-efficiency, NAS**

I. Introduction

Deep Learning (DL) is at the core of many modern computing applications, such as computer vision [1], sound classification [2], bio-signal analysis [3], predictive maintenance [4], etc. While DL models have been traditionally deployed on powerful cloud-based servers, evidence exists about the potential advantages of an implementation at-the-edge [5]. Edge computing could improve privacy and reduce the energy consumption at the distributed system level, by replacing the energy hungry wireless transmission of raw data with more efficient local computations and transmission of aggregated outputs [6].

This has spurred strong academic and industrial interest for so-called *TinyML*, i.e., the study of techniques and tools to enable the deployment of Machine Learning (ML) and DL models on low power, battery-operated edge devices. In this context, the key hard-requirement to be satisfied is on the memory footprint of DL models, which should match the severe constraints of edge devices, typically based on Microcontrollers (MCUs) with few MBs of Flash and RAM [7].

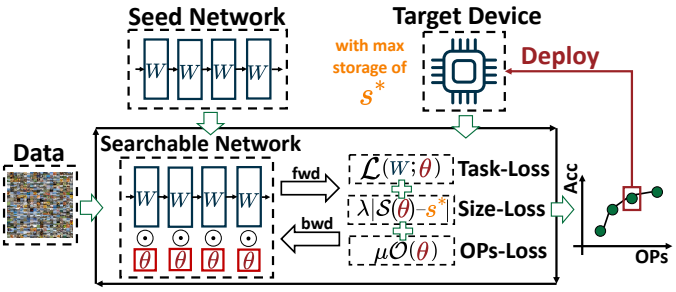

Fig. 1. Overview of the proposed approach.

At the same time, energy consumption should be minimized, typically by reducing the total number of operations (OPs) per prediction, in order to maximize the system's lifetime.

Achieving these goals through a manual tuning of a Deep Neural Network's (DNN) hyper-parameters, while maintaining a sufficient prediction accuracy, is a tedious and time-consuming process. Therefore, Neural Architecture Search (NAS) tools have emerged as new design space exploration and automation solutions, able to find DNN hyper-parameters that co-optimize prediction performance and a computational cost metric, such as the number of parameters (i.e., *Size*), the number of OPs per inference, or the latency/energy consumption [8], [9]. However, classic NASes, e.g., based on reinforcement learning, are extremely time-consuming (1000s of GPU hours), thus being inaccessible to most edge systems designers, while light-weight Differentiable NAS (DNAS) solutions are limited in the ways they can express optimization objectives and constraints. Indeed, to our knowledge, all existing DNAS methods optimize either the model size or the number of OPs *separately*. In contrast, the relevant problem from a designer's perspective is the *minimization of energy (OPs) under a given memory constraint*.

In this work, we address this issue proposing a novel problem formulation (shown in Fig. 1) that can be applied to any DNAS, and allows to find a set of Pareto-optimal architectures in the accuracy vs OPs space, under a fixed model size constraint. We validate our formulation in combination with a simple DNAS, that performs a fine-grained search over the number of channels in Convolutional Neural Network (CNNs) layers. With experiments on three different benchmarks from the TinyMLPerf suite [10] we show that, starting from networks already optimized for edge deployment, our method can

Accepted for publication at the ISLPED 2022 ACM/IEEE International Symposium on Low Power Electronics and Design.

further improve and enrich the Pareto frontier. Deploying the automatically discovered architectures on a real edge device, the STM NUCLEO-H743ZI2, we show that we can reduce the energy consumption up to 2.2× with negligible accuracy drop, while also cutting memory occupation, compared to the baseline hand-tuned CNNs. Our code is open-sourced at: `https://not-yet-available`.

II. Background and Related Works

Initial attempts to enable inference at-the-edge were based on hand-crafting efficient DNN architectures. Notable examples include SqueezeNet [11], MobileNets [12], EfficientNet [13], etc. Such models, originally proposed for mobile deployment, are the result of a long and time-consuming manual tuning of hyper-parameters based on heuristics and human ingenuity. While very efficient, hand-tuned DNNs are never a one-size-fits-all: they represent a single point in the accuracy versus complexity space, which cannot fit all deployment scenarios. For instance, they cannot be directly applied in TinyML use cases, where hardware has much tighter constraints with respect to mobile systems. Accordingly, in order to prevent designers from having to repeat such hyperparameters' hand-tuning from scratch for each prediction task and deployment target, research has recently focused on automated DNN optimization solutions referred to as "NAS".

Early NAS algorithms explore the search space by means of Evolutionary Algorithms (EA) [14] or training a Reinforcement Learning (RL) agent [15]–[17]. At each iteration they sample and *fully train* one or more architectures from the search space. The obtained accuracy and cost (e.g., model size, OPs, etc.) are then used to generate the EA fitness function or RL reward. Such solutions are extremely powerful, allowing to accommodate any combination of optimization target and constraints. However, they do not scale well with the dimension of the search space, requiring thousands of GPU hours for a single search, due to the repeated sequence of sampling, training, and evaluation. So-called "proxies", e.g., training for a reduced number of epochs or on a reduced dataset, can cut the search complexity, but may also undermine the quality of results [9].

Alternatively, search costs can be reduced resorting to more recent DNAS methods, which optimize a DNN architecture *while training it*, using the same gradient descent algorithms to optimize both the weights and the network hyperparameters (e.g., depth, neurons, receptive field, etc). One way to achieve this result is through the use of *super-nets*, large DNNs including multiple alternative implementations of each layer/module, with different hyperparameters settings [18]. Each super-net path corresponds to a potential final architecture, and the optimal one is selected during training, using a differentiable relaxation of the problem: alternative layers' are combined by means of a set of continuous and trainable *architectural weights*, which are then optimized by gradient-descent, so to assign larger weights to the alternatives that maximize the metric(s) of interest. At the end of the training, a discretization step selects a single path, typically the one associated with the largest architectural weights.

In DNASes, networks that are simultaneously accurate and low-complexity are typically found enhancing the standard, task-dependent, loss function $\mathcal{L}$ with an additional regularization term $\mathcal{R}$, that models the cost metric to be optimized (size, OPs, energy/latency, etc.) as a differentiable function. The overall optimization goal becomes:

$$\min_{W,\theta} \mathcal{L}(W;\theta) + \lambda\mathcal{R}(\theta) \tag{1}$$

where W is the set of trainable weights of the network (e.g., convolutional kernels), θ is the set of NAS architectural weights encoding the different paths in the super-net and λ is a scalar regularization strength that controls the relative importance between the task-specific loss and complexity loss.

While super-net-based DNAS can find an optimized architecture with a single training, thus being much faster than RL/EA methods, such training is still tricky when dealing with large search spaces, due to the explosion of the super-net size, which causes huge training time and memory overheads with respect to a "normal" DNN. ProxylessNAS [9] tackles the memory problem by sampling at most two super-net paths for each batch of inputs.

Other methods, such as FbNetV2 [19], MorphNet [8] and PIT [20] replace the super-net with a standard DNN with a unique path, usually denoted as *seed network*. The search space is formed by sub-architectures of the seed, obtained by reduction of its hyper-parameters. In practice, this result is obtained *masking* different slices of each layer's weights with binary parameters, so that the slices multiplied with a 0 are effectively eliminated from the layer. The continuous relaxation of the binary mask is then optimized, similarly to the architectural weights in a super-net DNAS, with the objective of reducing the network complexity, by eliminating unimportant parts of each layer (in that, this approach is similar to a *structured pruning* [5]). The usage of masks introduces a minimum overhead with respect to a normal training of the seed [20], reducing the search time and memory requirements significantly compared to super-net approaches, and representing a further step towards lightweight NAS.

One drawback of mask-based DNAS is that the search-space definition is less flexible, since it can only include reduced variants of the seed. Nevertheless, this is traded with a much more fine-grained search granularity, hardly reproducible with multiple paths in a super-net. For instance, considering a convolutional layer with 32 channels, a mask-based DNAS can easily explore all variants of the number of feature maps with a granularity of 1 (i.e., 31, 30, 29, and so on); doing the same with a super-net DNAS would require a huge network with 32 alternative versions just for that layer.

III. Proposed Method

DNAS tools that follow the formulation of (1) have two main limitations. First, $\mathcal{R}$ models a *single cost metric*, i.e., either the model size, the number of OPs, or a differentiable approximation of the latency or energy consumption, as a

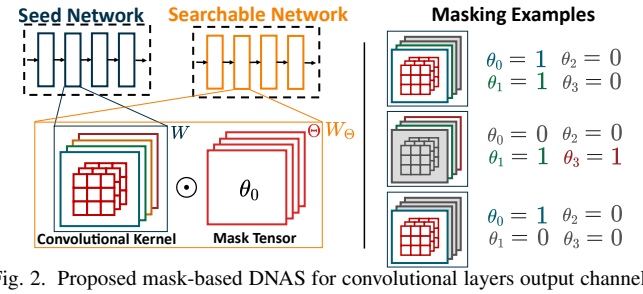

Fig. 2. Proposed mask-based DNAS for convolutional layers output channels.

function of the DNN hyper-parameters [8], [9]. Second, cost is considered as an objective to minimize, rather than a constraint. While this is appropriate for some metrics (e.g., OPs, latency or energy), it is sub-optimal when considering memory occupation. In fact, most designers are interested in finding the "best" model (e.g., the most accurate or the best balance between accuracy and energy consumption) that *fits a memory constraint*, given by the target hardware. Doing so with (1) requires repeating the DNAS multiple times, sweeping λ, until a model with appropriate memory footprint is found.

We propose a new DNAS formulation that addresses both problems, allowing to consider both memory occupation and other cost metrics (OPs, in our experiments) simultaneously, taking the former as a constraint and the latter as an objective. In practice, this enables the search for Pareto-Optimal architectures in the Accuracy vs OPs plane, *around a fixed memory budget*. We apply the proposed method on top of a mask-based DNAS that explores the number of channels C_{out} of Convolutional layers in a CNN. However, our formulation is agnostic of the specific search method, and can be applied to *any* DNAS, including super-net based ones.

The rest of this section is organized as follows. Sec. III-A describes the considered search space, while Sec. III-B presents the multi-regularization loss approach at the core of our method. Finally, Sec. III-C details the training algorithm.

A. Differentiable Channels Search

We apply our formulation to a simple but powerful mask-based DNAS, which optimizes the number of output channels of all convolutional kernels in a CNN seed. This is a generalization of the approach proposed in [8], which instead of adding explicit mask weights, made use of Batch Normalization (BN) parameters to eliminate some output channels. In contrast, our approach can work also when BN is not present, hence being applicable to *any* CNN.

Fig. 2 schematizes the proposed search scheme. Starting from the seed network, a "searchable" model is built modifying the weight tensor $W^{(n)}$ of each convolutional layer, with size $C_{out}^{(n)} \times K_x^{(n)} \times K_y^{(n)} \times C_{in}^{(n)}$, where $C_{in}^{(n)}$, $C_{out}^{(n)}$ are the input/output channels and $K_x^{(n)}$/$K_y^{(n)}$ are the horizontal/vertical kernel sizes. The searchable weights $W_\Theta^{(n)}$ are obtained as:

$$W_\Theta^{(n)} = W^{(n)} \odot \mathcal{H}(\theta^{(n)}) \tag{2}$$

where $\theta^{(n)}$ is a trainable mask tensor with $C_{out}^{(n)}$ elements, $\odot$ is the Hadamard product, and $\mathcal{H}$ is a Heaviside step function with a fixed threshold $th = 0$, which has the effect of binarizing $\theta^{(n)}$ to 0/1 (1 if $\theta \geq th$, else 0). The product is *broadcast* to an entire weight filter, i.e., the same element $\theta_i^{(n)}$ is multiplied with an entire slice $W_i^{(n)}$ of the weight tensor, of size $K_x^{(n)} \times K_y^{(n)} \times C_{in}^{(n)}$. Therefore, the i-th element of the mask vector controls whether the i-th output channel is removed from the network ($\mathcal{H}(\theta_i^{(n)}) = 0$) or kept alive ($\mathcal{H}(\theta_i^{(n)}) = 1$). The obtained "searchable" network is then inserted in a normal training loop, where W and θ are trained together.

Specifically, in each forward pass of the training, Heaviside binarization has the effect of sampling a single architecture from the search-space (see the examples on the right of Fig. 2). During backward passes, instead, a Straight-Through Estimator (STE) based on the BinaryConnect [21] approach lets gradients flow through the network despite the presence of the non-differentiable Heaviside Function.

B. Multi-Regularization Loss

Most state-of-the-art DNAS tools [8], [9], [19] sum the task-specific loss $\mathcal{L}$ and the complexity term $\mathcal{R}$, scaled by a strength constant, using the scheme of (1). MorphNet [8] and FBNetV2 [19] regularize either against the number of parameters or against the number of OPs, while Proxyless-NAS [9] tries to optimize the latency directly, using a model obtained fitting the latency measurements obtained profiling layers with different hyper-parameters combinations. More recently, UDC [22] proposed a different approach, where the regularization term includes a specific cost target r^* to be satisfied, and the regularization term becomes $|\mathcal{R}(\theta) - r^*|$.

Building upon these ideas, we propose a novel formulation with *two complexity loss terms*, which drive the DNAS towards a desired region of the search space, while still allowing the exploration of accuracy versus complexity trade-offs. The proposed optimization problem formulation takes the form:

$$\min_{W,\theta} \mathcal{L}(W;\theta) + \lambda|\mathcal{S}(\theta) - s^*| + \mu\mathcal{O}(\theta) \tag{3}$$

In the equation, $\mathcal{S}$ models the size (i.e., memory footprint) of the DNN as a function of the architecture parameters θ. In particular, for a CNN with N convolutional layers, $\mathcal{S}$ is computed as the total number of *effective* (i.e., non-masked) parameters in those layers, i.e.:

$$\mathcal{S}(\theta) = \sum_{n=0}^{N} \mathcal{S}^{(n)}(\theta) = \sum_{n=0}^{N} C_{out}^{(n-1)}(\theta) C_{out}^{(n)}(\theta) K_x^{(n)} K_y^{(n)} \tag{4}$$

where $C_{out}^{(n-1)} = C_{in}^{(n)}$ and, for the 1st layer $C_{out}^{(n-1)}$ is fixed and equal to the number of channels in the input data.

Since, as explained above, memory occupation is usually a constraint that DNNs should respect for edge deployment, rather than a metric to optimize, we follow the approach of [22], minimizing the absolute value difference from a target size s^* which depends on the hardware.

We associate this cost term with a relatively large and *fixed* regularization strength λ, with $\lambda >> \mu$, thus forcing the NAS to find a set of θ^* parameters that yield $\mathcal{S}(\theta^*) \approx s^*$. This

Algorithm 1

for $i \leftarrow 1, \ldots, \text{Epochs}_{\text{wu}}$ **do** # warmup loop
    Update W based on $\nabla_W \mathcal{L}(W)$
end for
while not converged **do** # search loop
    Update W, θ based on $\nabla_{W,\theta}(\mathcal{L}(W;\theta) + \lambda|\mathcal{S}(\theta) - s^*| + \mu\mathcal{O}(\theta))$
end while
for $i \leftarrow 1, \ldots, \text{Epochs}_{\text{ft}}$ **do** # fine-tuning loop
    Update W based on $\nabla_W \mathcal{L}(W)$
end for

allows us to immediately respect the memory constraint in each search, without a lengthy sweep of λ values. The way λ is calculated for a given seed is detailed in Sec. III-C.

Furthermore, we add a further loss term $\mathcal{O}$ to model additional complexity-related metrics. In this work, $\mathcal{O}$ models the total number of OPs per prediction, which correlates with inference latency and energy consumption:

$$\mathcal{O}(\theta) = \sum_{n}^{N} \mathcal{S}^{(n)}(\theta) O_x^{(n)} O_y^{(n)} \tag{5}$$

where $O_x^{(n)}$ and $O_y^{(n)}$ are respectively the output feature map width and height of the n-th convolutional layer. We consider this metric as a general and easy-to-compute estimate of the inference cost of a model. However, our formulation is not limited to this specific expression for $\mathcal{O}$, and would be equally effective using more precise, profile-based energy or latency estimates, such as those proposed in [9].

Differently from $\mathcal{S}$, $\mathcal{O}$ is treated as an objective, not a constraint, and its importance is weighted by μ, which is the main knob used in our method to generate different final architectures starting from a single seed. To summarize, the formulation of (3) will produce DNNs with a size around s^*. With a small μ, the DNAS will focus on minimizing $\mathcal{L}$, producing networks as accurate as possible (for that size constraint), while large μs will cause to partially sacrifice the accuracy in exchange for fewer OPs.

Intuitively, since the OPs of a convolutional layer are equal to the parameters multiplied by the output feature map size (see Eq. 5), and since feature sizes tend to reduce going forward in the network, due to the effect of pooling, strided convolution, etc, OPs reduction under a fixed size budget can be obtained masking more channels in the *initial* layers of the DNN, and less channels in the *final* ones. We show that our formulation produces precisely this behavior in Sec. IV.

C. Training Procedure

Alg. 1 shows the overall training scheme of our DNAS. We start with a *warmup phase*, i.e., a normal training of the full seed network, in which all masking parameters θ are frozen at the initialization value (i.e., 1), and only the normal weights W are trained. The training objective in this phase consists solely of the task-specific loss function $\mathcal{L}$. Noteworthy, warmup can be performed just once, saving the learned weights and reusing them for all following searches.

The second phase coincides with the actual architecture optimization. In this step, the weights W and the masking parameters θ are optimized together, to minimize the cumulative loss function of (3). The number of search epochs is controlled with an early-stop mechanism which monitors the task loss $\mathcal{L}$ on an unseen validation-split, and stops the search when the loss stops improving. When a validation set is not provided in the considered benchmark, we generate it by randomly sampling 10% of the training set.

Lastly, in the *fine-tuning* phase, similarly to warmup, only the weights W are trained against the task loss $\mathcal{L}$, while the θ architectural parameters are frozen to the final learned values. In all our experiments, we set the warmup and fine-tuning epochs $\text{Epochs}_{\text{wu}}$ and $\text{Epochs}_{\text{ft}}$ equal to the number of training epochs used in the papers proposing each benchmark task.

For a given target size s^*, the size strength λ is determined with the formula $\lambda = {}^{\mathcal{L}(\theta_{seed})}/_{|\mathcal{S}(\theta_{seed}) - s^*|}$, where $\mathcal{S}(\theta_{seed})$ and $\mathcal{L}(\theta_{seed})$ are the model size and task loss of the full seed network after warmup. The rationale is to have similar values for the first two addends of (3) at the beginning of a search, so that the DNAS does not just shrink the network in the first iteration, ignoring completely the impact on accuracy. We found this heuristic to work well, but we also noticed that, as expected, varying λ in a reasonable range ($\pm$ one order of magnitude) does not alter the search results significantly, since the term $\mathcal{S}(\theta) - s^*$ is quickly brought close to zero in the search phase. Importantly, this means that λ can be computed in closed-form and does not have to be swept.

Having fixed the target size (and λ), multiple iterations of Alg. 1 with different values of μ generate a front of Pareto-optimal architectures in the Accuracy versus OPs space. Specifically, we always run a first search with $\mu = 0$, to find the most accurate network which satisfies the s^* constraint, without taking into account the number of OPs. We then progressively increase μ to find less accurate and more efficient architectures. Importantly, too large μ values (violating $\lambda >> \mu$) lead to low-quality results, since the DNAS tries to dramatically reduce the number of OPs while simultaneously keeping the model size close to the target. This produces sub-optimal DNN architectures, with worse accuracy than those of smaller size. Thus, whenever increasing μ with fixed s^* degrades the accuracy too much (in our experiments, we limit to a 5% degradation w.r.t. the case $\mu = 0$), we simply stop the exploration and switch to a lower size target.

IV. Experimental Results

A. Setup

We evaluated the proposed NAS on three datasets taken from the MLPerf Tiny Benchmark Suite [10]. As seed networks, we used the reference architectures proposed in the suite for each dataset. The *Image Classification* (IC) benchmark is based on the well-known CIFAR-10 dataset, which consists of 60000 32x32x3 RGB images belonging to 10 classes. The reference CNN is a customized ResNet [1] with 8 convolutional layers. The *Visual Wake Word* (VWW) task considers the MSCOCO 2014 dataset, with 109619 96x96x3

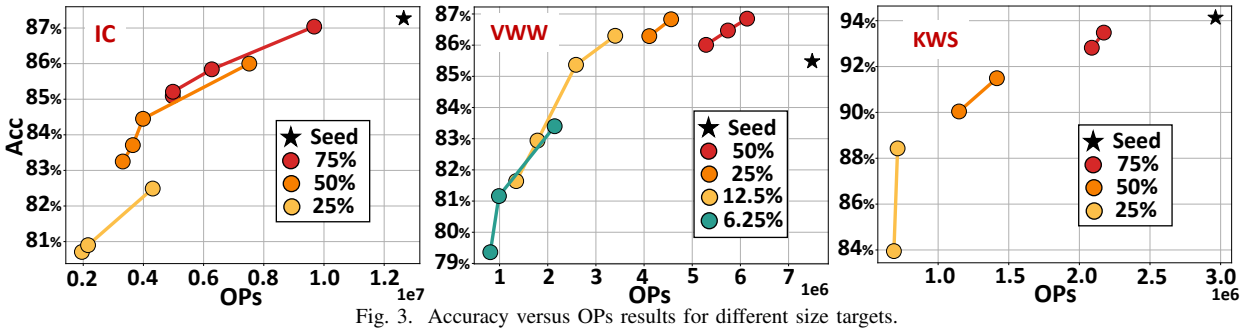

Fig. 3. Accuracy versus OPs results for different size targets.

RGB images, and the objective is detecting whether at least a person is present in the input. The reference architecture is a MobileNetV1 [12] with a width multiplier of 0.25. Lastly, the *KeyWord Spotting* (KWS) benchmark is based on the Speech Commands v2 dataset, which contains 105,829 utterances, to be classified in 12 classes including 10 words and two special labels ("unknown" and "silence"). The reference architecture is the Depthwise Separable CNN (DS-CNN) described in [2]. MLPerf Tiny includes a fourth Anomaly Detection benchmark. However, the reference DNN is an Autoencoder composed only of Dense layers, for which the model size and number of OPs are directly proportional (i.e., there is no degree of freedom to reduce the OPs under a fixed size budget). So, we did not consider that task, since it would not benefit from our formulation, which would become equivalent to (1).

Our DNAS is implemented in PyTorch v1.10.2. Some relevant architectures found by our tool are then deployed on a commercial edge device, the NUCLEO-H743ZI2, in order to estimate energy consumption. We convert ONNX graphs exported from PyTorch into C code using the proprietary X-Cube-AI toolchain of STM. In this work, we deploy floating-point models, but note that integer quantization is fully-orthogonal to our method. All results are reported on test sets.

B. Search-Space Exploration

Fig. 3 shows the results obtained applying the proposed DNAS on the three benchmarks. Each plot reports the found architectures (represented with coloured dots) and the seed (represented with a black star) in the Accuracy versus OPs space. Different colors correspond to different size targets s^*. To validate our approach, we initially set s^* to be respectively 75%, 50% and 25% of the original size of each seed network. In a real scenario, s^* would depend on the hardware, so this setup simulates targeting three different MCUs with progressively less available memory. Within the curve relative to each s^* target, different points are obtained changing the OPs regularization strength μ.

The left-most graph shows the results obtained on the IC task. Considering all three memory targets, the DNAS is able to find networks that span almost one order of magnitude in OPs 1.96M-9.86M), and $\pm$ 6.3% in accuracy. Moreover, under the 75% size constraint, we obtain a network that achieves a negligible accuracy drop with respect to the seed (-0.23%), while reducing the number of OPs by 1.3x (12.7M vs 9.86M).

The center graph reports the Pareto fronts obtained for the VWW task. In this case, we found that the results obtained with the 75% and 50% size constraints are completely outperformed by those obtained with lower memory, which achieve higher accuracy with fewer OPs (only the 50% curve is shown in the graph, for clarity). This means that the reference network used for this task is strongly over-parameterized. Therefore, forcing to optimize OPs with a too high s^*, leads to unbalanced architectures which attain same accuracy of smaller ones (see orange vs. red curves in mid graph of Fig. 3). Thus, we decided to add two additional memory targets (i.e., 12.5% and 6.25% of the seed) in order to show more insightful trade-offs. The results demonstrate once again that our DNAS is able to find a rich collection of Pareto-optimal architectures for multiple memory constraints. The NAS results span almost one order of magnitude in terms of OPs (0.81M-6.14M). Furthermore, many of the found architectures Pareto-dominate the seed, even at 12.5% size (+0.39% accuracy with 2.5$\times$ OPs reduction and +0.82% accuracy with 2.2$\times$ OPs reduction).

Lastly, the right-most plot in Fig. 3 shows the results on the KWS task. In this case, Pareto-fronts are not as rich as for the other two benchmarks, due to the peculiarities of the seed network. In fact, DS-CNN includes strided convolutions and pooling only in the first and last convolutional layers. Consequently, all intermediate feature map sizes are identical, with OPs and model size strongly correlated. Nonetheless, we still find multiple networks for each size constraint, although the trade-off between OPs and accuracy is less favorable. At most, for the 50% size target, we obtain an OPs difference of 1.2$\times$ in exchange for an accuracy degradation of 1.45%.

C. Architecture Details

As an example of the architectures found by our DNAS, Fig. 4 reports four of the networks generated for the IC benchmark, under the 75% and 25% size constraints. The models labeled with "-H" (high-OPs) are obtained with $\mu = 0$, i.e., performing the search with the only constraint of respecting the target size, without OPs reduction. Instead, the networks labeled with "-L" (low-OPs) are obtained with with $\mu \neq 0$, and in particular, they correspond to the points with fewest OPs in the 75% and 25% Pareto fronts of the graph in Fig. 3. Each rectangle represents a convolutional layer, and numbers inside them correspond to $C^{(n)}_{out,final}/C^{(n)}_{out,seed}$.

These examples demonstrate that our formulation generates meaningful results. First, as expected, a lower target size

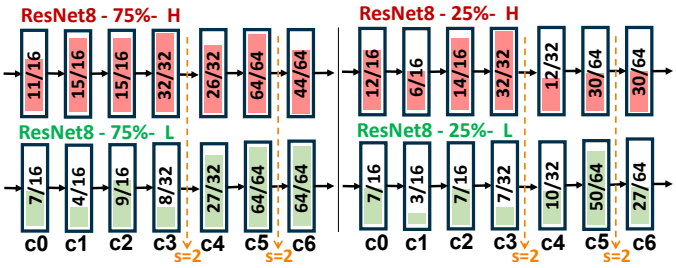

Fig. 4. Examples of found architectures for the IC benchmark.

TABLE I
DETAILED DEPLOYMENT RESULTS FOR THE IC AND VWW BENCHMARKS. THE MEM. FIELD EXPRESS WEIGHTS MEMORY IN KB.

Task	Network	Acc.	Mem. [kB] (% ≠ constr.)	Lat. [ms]	En. [mJ]
IC	Seed	87.27%	310	125	29.3
	75%-H	87.04%	231.6 (-0.37%)	110	25.7
	75%-L	85.09%	233.5 (+0.44%)	57.4	13.4
	50%-H	86.00%	155.8 (+0.54%)	87.7	20.5
	50%-L	83.22%	156 (+0.65%)	42.1	9.84
	25%-H	82.49%	74.96 (-3.3%)	55.9	13.1
	25%-L	80.71%	75.6 (-2.5%)	27.4	6.42
VWW	Seed	85.48%	832.4	115	26.9
	25%-H	86.83%	206.1 (-0.97%)	81.4	19.1
	25%-L	86.29%	208.2 (+0.05%)	72.9	17.1
	12.5%-H	86.30%	104.1 (+0.03%)	69.5	16.3
	12.5%-L	81.64%	103.8 (-0.2%)	34.6	8.09
	6.25%-H	83.40%	50.32 (-3.3%)	53.5	12.5
	6.25%-L	79.36%	52.12 (+0.2%)	24.5	5.73

results in more masked channels, regardless of the OPs regularization strength. Moreover, the "-L" networks have fewer channels in their *initial* layers, which are those that contribute more to the total OPs, due to the larger resolution of their input/output feature maps. The considered ResNet8 has two convolutional layers with stride $s = 2$ (c3 and c5), indicated by yellow dashed lines in Fig. 4, which reduce the feature map sizes of downstream layers by a factor 4. As evident from the figure, our DNAS reduces much more aggressively the layers before c3 when μ increases.

D. Embedded Deployment

Table I summarizes the deployment results on the NUCLEO-H743ZI2 for the IC and VWW benchmarks. We do not report KWS results because, as explained above, the structure of the reference CNN makes the trade-offs less interesting. The table reports two DNNs for each target size (high-OPs "-H" and low-OPs "-L"), corresponding to the two extremes of each Pareto front of Fig. 3, neglecting the fully-dominated 50% front for VWW. Additionally, for comparison, we also deploy the baseline seed networks.

The Mem. column reports the memory occupation of each model, and the difference in percentage from the imposed constraint. As shown, all networks are within $\pm 3.3\%$ from the target, showing that our constraint formulation produces the expected results. Further, on the IC task we find solutions with energy consumption spanning from 25.7mJ to 13.4mJ, 20.5mJ to 9.84mJ and 13.1mJ to 6.42mJ respectively for the 75%, 50% and 25% targets. Noteworthy, the 75%-H network reduces the energy consumption by 2.2× with respect to the seed, whit negligible accuracy drop. Similarly, the deployed solutions for VWW with 25%, 12.5% and 6.25% size consume respectively from 19.1mJ to 17.1mJ, from 16.3mJ to 8.09mJ and from 12.5mJ to 5.73mJ, and the 12.5%-H CNN reduces the energy consumption of the seed by 1.7× while improving accuracy of 0.9%.

V. CONCLUSIONS

We have proposed a new DNAS formulation that can be used to enhance existing tools allowing them to find DNNs with optimal trade-offs between accuracy and inference complexity, under fixed memory constraints. With experiments on three different real-world edge-relevant use-cases, we have shown the effectiveness of our method, which is able to reduce the energy consumption by up to 2.2× with respect to hand-tuned baseline models.

REFERENCES

[1] K. He *et al.*, "Deep residual learning for image recognition," in *Proc. IEEE CVPR*, 2016, pp. 770–778.
[2] Y. Zhang *et al.*, "Hello edge: Keyword spotting on microcontrollers," *arXiv:1711.07128*, 2017.
[3] A. Burrello *et al.*, "Q-ppg: Energy-efficient ppg-based heart rate monitoring on wearable devices," *IEEE Trans. Biomed. Circuits Syst.*, 2021.
[4] A. Burrello *et al.*, "Predicting hard disk failures in data centers using temporal convolutional neural networks," in *Proc. Euro-Par*, 2020, pp. 277–289.
[5] F. Daghero *et al.*, "Energy-efficient deep learning inference on edge devices," in *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*, ser. Advances in Computers, S. Kim *et al.*, Eds. Elsevier, 2021, vol. 122, ch. 8, pp. 247–301.
[6] W. Shi *et al.*, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct 2016.
[7] ST Microelectronics. STM32H7. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html
[8] A. Gordon *et al.*, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *Proc. IEEE CVPR*, 2018, pp. 1586–1595.
[9] H. Cai *et al.*, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv:1812.00332*, 2018.
[10] C. R. Banbury *et al.*, "Benchmarking tinyml systems: Challenges and direction," *arXiv:2003.04821*, 2020.
[11] F. N. Iandola *et al.*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size," *arXiv:1602.07360*, 2016.
[12] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.
[13] M. Tan *et al.*, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*. PMLR, 2019, pp. 6105–6114.
[14] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. ICML*. PMLR, 2017, pp. 2902–2911.
[15] B. Zoph *et al.*, "Neural architecture search with reinforcement learning," *arXiv:1611.01578*, 2016.
[16] B. Zoph *et al.*, "Learning transferable architectures for scalable image recognition," *Proc. IEEE/CVF CVPR*, pp. 8697–8710, 2018.
[17] M. Tan *et al.*, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proc. IEEE CVPR*, 2019, pp. 2820–2828.
[18] H. Liu *et al.*, "Darts: Differentiable architecture search," *arXiv:1806.09055*, 2018.
[19] A. Wan *et al.*, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," in *Proc. IEEE/CVF CVPR*, 2020, pp. 12 965–12 974.
[20] M. Risso *et al.*, "Pruning in time (pit): A lightweight network architecture optimizer for temporal convolutional networks," in *Proc. 58th ACM/IEEE DAC*, 2021, pp. 1015–1020.
[21] M. Courbariaux *et al.*, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Adv. Neural Inf. Process. Syst.*, vol. 28, 2015.
[22] I. Fedorov *et al.*, "Udc: Unified dnas for compressible tinyml models," *arXiv:2201.05842*, 2022.