

# Entropy Sampling-Based Neural Architecture Search for Resource-Constrained Microcontroller Targets

Christian Heidorn<sup>1</sup>, Frank Hannig<sup>1</sup>, Dominik Riedelbauch<sup>2</sup>, Christoph Strohmeyer<sup>2</sup>, Jürgen Teich<sup>1</sup>

<sup>1</sup> Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

<sup>2</sup> Schaeffler Technologies AG & Co. KG, Herzogenaurach, Germany

**Abstract**—Neural architecture search (NAS) is a popular approach for the exploration of neural network (NN) architectures. Recently proposed hardware-aware NAS techniques even take resource constraints, such as FLOP count and number of weights, into account. Still, in typical NAS search spaces, a significant portion of candidate NNs may be infeasible when it comes to satisfying tight memory (i.e., RAM and ROM) and timing constraints, particularly in the case of microcontroller targets. As evaluating each design point can be quite time-intensive, we first show how to pre-process a given design space to a reduced set of only feasible (resource constraint fulfilling) solutions, and then efficiently sampling from this set of only feasible solutions by proposing an entropy-based sampling technique and the optimization goal to maximize accuracy. We demonstrate that our approach is able to find feasible solutions with similar accuracy to other hardware-aware NAS techniques, but already after a much lower number of model evaluations, with examples taken from the MLPerf Tiny Benchmark suite.

## I. INTRODUCTION

Designing neural networks for resource-constrained edge devices strives to achieve the highest possible performance in terms of accuracy while satisfying constraints on computational costs (e.g., the number of floating-point operations, short FLOPs) and memory footprint (e.g., the number of parameters and intermediate results). Achieving these contradicting design goals is very challenging.

Here, *neural architecture search* (NAS) techniques that are aware of the target hardware on which the neural architecture is going to be deployed have received more and more interest [1–4]. Recently, so-called *one-shot* and *zero-shot* NAS approaches have been introduced [1, 2, 5, 6]. One-shot NAS techniques typically train one large supermodel. The resulting supermodel includes many submodels that can subsequently be evaluated and checked for compliance with the hardware constraints. But depending on the supermodel and the capabilities of a target microcontroller (i.e., amount of available ROM and RAM), the design space may contain no or only a small number of feasible solutions that fulfill respective constraints.

Zero-shot NAS has been introduced with the goal of ranking the accuracies of NN candidates without the need of training [7]. The advantage of zero-shot NAS is that many different architectures can be evaluated in a short amount of time. However, proposed zero-shot proxies do not necessarily achieve high correlations with respect to accuracy, and must be individualized to the type of NNs as well as to the dataset on which the found neural architectures are trained. Still, all previously presented approaches are burdened not only by extremely large search spaces to be explored, but also by the fundamental problem that

only a few of these design points comply with target-induced resource constraints. As a remedy, this paper presents first a technique that carves out the set of feasible design points (in terms of allowed execution time and memory footprints) from the initially huge design spaces of NAS to subsequently apply an efficient search based on entropy-sampling for guiding the exploration to regions of high accuracy where each sampled point is evaluated for accuracy by training. Our contributions are as follows:

- Reduction of typically extremely large NAS design spaces  $X$  of NN candidates to a far smaller design space  $X^f$ , containing only feasible NN candidates that meet a microcontroller’s constraints, i.e., RAM, ROM constraints as well as user-defined execution time constraints.
- Efficient representation of the space  $X^f$  of feasible solutions by generic tree structures, and characterization of promising solutions by clustering NNs in the feasible search space based on an entropy measure.
- Proposing a method called entropy-based sampling of feasible solutions to prioritize architectures with higher entropies that are more likely to provide higher accuracies, leading to a reduction of both training overhead and thus search time.

Moreover, it is demonstrated that our proposed NAS approach is flexibly applicable to quite different types of NN design spaces, datasets, and types of constraints for different microcontroller targets. Finally, our experimental evaluations witness that our proposed NAS technique offers a significant reduction in search time by needing to evaluate a fundamentally smaller number of design points in order to identify feasible designs with comparable accuracy to existing NAS methods (multi-shot, one-shot, and zero-shot) that do not consider deployment constraints.

## II. RELATED WORK

Common to both NAS techniques in general and hardware-aware NAS techniques in particular is a definition of a *search space* of NN candidates using either building blocks to compose candidates of, or prescribing the structure of a set of candidate NNs [8]. To identify candidates of high quality, each candidate must be evaluated using figures of merit such as accuracy on a test dataset, required number of FLOPs, or model complexity, e.g., in terms of the number of weights. The *search strategy* defines how the search space of NN candidates is explored. In *multi-shot* NAS, each NN candidate is evaluated by training during exploration. In this context, Bayesian optimization [4] and evolutionary algorithms [3, 9] have emerged as some of the most

popular exploration approaches. Multi-shot approaches usually require a considerable amount of exploration time. Liberis *et al.* [3] presented a multi-shot search strategy, which uses aging evolution to explore different neural network candidates with the objectives of minimizing memory consumption, number of FLOPs, and number of weights, while maximizing the accuracy. Here, the exploration of NN candidates for the keyword spotting dataset [10] took up to 39 GPU-days. To reduce the huge search time induced by training, two approaches have emerged: *one-shot* and *zero-shot* NAS. In the case of one-shot NAS, submodels are sampled from a so-called *supermodel*, which only needs to be trained once. An example of this is differential architecture search (DARTS) [11]. Here, all models are treated as different submodels of a supermodel, and weights are shared between models that share the same layers. One-shot methods can have drawbacks, such as so-called *rank disorder*<sup>1</sup>, high memory requirements for training the supermodel, and poor test generalization, making the one-shot approach less preferable over the multi-shot approach [8, 12]. One-shot NAS approaches for resource-constrained devices have been presented for microcontrollers within the MCUNet [1], MicroNets [2], and DUCCIO [5] frameworks. These approaches can be seen as two-stage optimization processes: first, a supermodel is trained with the aim of maximizing accuracy; then, in a second stage, a submodel is searched within the supermodel (e.g., with an evolutionary algorithm) that satisfies the constraints of the target microcontroller. This approach obviously suffers from the inefficiency that only a small proportion of the search space might meet the strict requirements of microcontroller resources. In the worst-case scenario, where only one trained supermodel is available and a microcontroller with tight memory constraints given, it may even be impossible to find any feasible submodel in the supermodel. This means that both the search space and the supermodel implementation must be refined, a process that can be very labor-intensive, particularly if done manually. Rather than training and sampling infeasible solutions during exploration, our approach proposed in this paper first reduces the search space to contain only valid solutions that meet the target platform’s constraints (i.e., available memory and number of operations).

The most recent approach, eliminating the training of NN candidates for their evaluation, is called zero-shot or training-free NAS. Here, so-called *zero-shot proxies* are used to rank the expected accuracy of NN candidates without any training [7, 13]. MONAS [6] is one of the first works to include zero-shot NAS for microcontrollers. MONAS has only two objectives, namely minimizing the number of floating-point operations while maximizing the accuracy, utilizing an accuracy proxy based on FLOPs and the neural tangent kernel (NTK). In AEMONAS [14], the authors propose an entropy indicator to increase diversity between NN candidates during exploration. This differs from the entropy score used in previous work [15], which assumes that higher entropy (measured by the number

<sup>1</sup>Rank disorder is the term used to describe whether the key assumption of one-shot methods—that the ranking of submodels evaluated with the supermodel is relatively consistent with the ranking that would be obtained from training them independently—is not met.

of weights and sizes of intermediate feature maps) indicates greater information storage capacity and, consequently, higher achievable accuracy. Unfortunately, MONAS and AEMONAS have only been evaluated on NAS-Bench-201 and cell-based design spaces and do not account for the memory constraints of the target microcontroller. Thus, solutions might be selected that do not fit the microcontroller constraints, i.e., solutions found may be impossible to be deployed. As with one-shot NAS, zero-shot proxies can only estimate accuracy and are subject to rank disorder. This means that candidate networks with high accuracy estimates may achieve lower accuracy after training.

In our approach, we propose a computationally inexpensive, entropy-based architectural zero-shot proxy for neural networks [15, 16] to create clusters of neural network (NN) candidates according to their entropy values. A subsequent entropy-based search strategy then guides the exploration of design points of the feasible only search space towards high-accuracy solutions by sampling clusters of higher entropy more frequently. Each sampled design point is finally evaluated for accuracy based on training and not based on zero-shot proxy scores. In summary, our approach first reduces the search space to only feasible design points, and then uses a zero-shot proxy based on entropy to cluster regions of points together in order to need to efficiently sample candidates from clusters with the likelihood of higher accuracy more often.

### III. CONSTRAINT-DRIVEN DESIGN SPACE REDUCTION

Rather than imposing hardware constraints after or during the evaluation of search candidates, we treat hardware-aware NAS as a constrained optimization problem. In the first step, a search space  $X$  of neural network candidates (Section III-A) is reduced to the *feasible search space*  $X^f$ , which contains only solutions that meet the given user- and target-related deployment constraints (Section III-B and Section III-C). Then, only the search space  $X^f$  of feasible solutions is explored by proposing a novel entropy sampling-based search strategy with the objective of maximizing the accuracy (Section III-D).

#### A. Initial Search Space

In general, let a neural network  $x$  consist of  $L \in \mathbb{N}$  interconnected layers, including various types of convolutional layers (e.g., standard, depthwise separable, and inverted bottleneck convolutions), fully connected layers, and activation layers. In hardware-aware NAS [1, 2], a search space is usually defined as a fixed network structure, the so-called backbone architecture, which is typically based on experience from existing neural networks (e.g., MobileNets [17] or DS-CNNs [2]). The backbone architecture describes the structure (i.e., type of layers and interconnections), whereas the design space comprises different networks of the same structure, but with varying depth (i.e., number  $L$  of layers) and width (i.e., number  $M_i$  of output neurons or filters per layer) [18]. For example, a search space can be defined by (a) the number  $L$  of layers:

$$L \in \mathcal{L} = \{\lambda \in \mathbb{N} \mid L^{\text{lo}} \leq \lambda \leq L^{\text{up}}\},$$

where  $L^{\text{lo}} \in \mathbb{N}$  and  $L^{\text{up}} \in \mathbb{N}$  define the lower and upper bounds of the number of layers, and (b) the number  $M_i$  of filters or output neurons per layer  $l_i$ , with  $1 \leq i \leq L$ :

$M_i \in \mathcal{M}_i = \{s \cdot m + M_i^{\text{lo}} \mid m \in \mathbb{N}_0 \wedge 0 \leq m < m_i^{\text{up}}\}$ ,  $m_i^{\text{up}} \in \mathbb{N}$  and  $s \in \mathbb{N}$  is a given step size, and  $M_i^{\text{lo}} \in \mathbb{N}$  denotes the lower bound and  $M_i^{\text{up}} = (m_i^{\text{up}} - 1) \cdot s + M_i^{\text{lo}}$  the upper bound of the number of filters of layer  $l_i$ , respectively. With  $L \in \mathcal{L}$ , the search space  $X_L$  is defined as follows<sup>2</sup>:

$$X_L = \{(M_i)_{i=1, \dots, L} \mid M_i \in \mathcal{M}_i\}.$$

With this notation, the search space  $X$  can be defined as  $X = \bigcup_{L \in \mathcal{L}} X_L$ , and its corresponding size (cardinality) is given by

$$|X| = \sum_{L \in \mathcal{L}} \prod_{i=1}^L \underbrace{m_i^{\text{up}}}_{|\mathcal{M}_i|}. \quad (1)$$

As a running example, we introduce the search space of an  $L = 2$  layer multi-layer perceptron (MLP) with  $M_1^{\text{lo}} = M_2^{\text{lo}} = 16$ , and  $M_1^{\text{up}} = M_2^{\text{up}} = 128$ ,  $s = 16$ . The two-dimensional search space is visualized in Fig. 1.

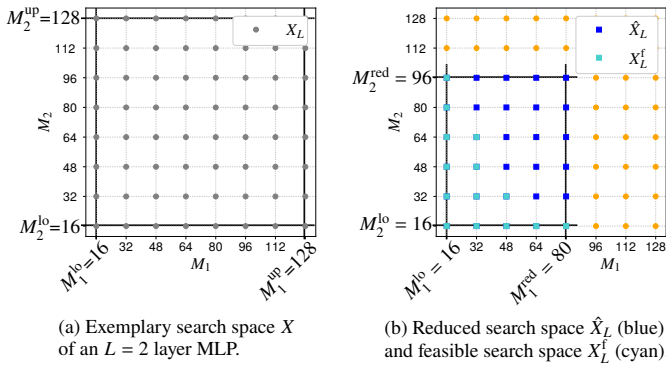


Fig. 1: (a) An example of  $X$  (grey dots) for an  $L = 2$  multi-layer perceptron, with the respective lower and upper bounds  $M_1^{\text{lo}} = M_2^{\text{lo}} = 16$ , and  $M_2^{\text{up}} = M_2^{\text{up}} = 128$ ,  $s = 16$ ; (b) The orange dots represent the non-feasible solutions that do not meet the constraints, the blue squares the reduced search space  $\hat{X}_L$ , with respective reduced upper bounds, and the cyan squares the feasible search space  $X_L^f$ .

## B. Reduction of Search Space

In order to reduce a search space  $X_L$  to a search space containing only feasible solutions  $X_L^f$ , we first determine whether there exists at least one feasible solution in  $X_L$ . This involves checking whether the network defined by the lower bounds  $M_1^{\text{lo}}, \dots, M_i^{\text{lo}}, \dots, M_L^{\text{lo}}$  is feasible. If not,  $X_L$  and any search space subsets with more than  $L$  layers can be discarded and removed from the overall search space. Else, we can then determine a reduced search box  $\hat{X}_L$  spanned by reduced upper bounds  $M_i^{\text{red}} \leq M_i^{\text{up}}, \forall i : 1 \leq i \leq L$ , by solving the following optimization problem:

$$M_i^{\text{red}} = \max_{x \in X} M_i \quad (2a)$$

$$\text{subject to } U^{\text{ROM}}(x) \leq B^{\text{ROM}} \quad (2a)$$

$$U^{\text{RAM}}(x) \leq B^{\text{RAM}} \quad (2b)$$

$$\text{FLOPs}(x) \leq B^{\text{FLOPs}}. \quad (2c)$$

As constraints, we formulate ROM size (Eq. (3)), RAM size (Eq. (4)), and FLOP count (Eq. (5)) constraints that are introduced

<sup>2</sup>Notation for  $n$ -tuple:  $(a_1, a_2, \dots, a_n) = (a_i)_{i=1, \dots, n}$

in detail in Section III-C. In Fig. 1b, the reduced search space  $\hat{X}_L$ , with the respective reduced upper bounds ( $M_1^{\text{red}}$  and  $M_2^{\text{red}}$ ) of the initial search space  $X$  of the running example is shown. Subsequently, we also check whether the reduced upper bound vector  $(M_1^{\text{red}}, \dots, M_L^{\text{red}})$  satisfies the constraints in Eqs. (2a) to (2c). If not, we can determine the set of feasible solutions with  $L$  layers  $X_L^f$  (visualized by the cyan squares in Fig. 1b) by a scan of the reduced box to create an efficient tree representation (Section III-E). The overall feasible search space  $X^f$  is then defined by  $X^f = \bigcup_{L \in \mathcal{L}} X_L^f$ .

According to our evaluations, this reduction can be accomplished typically as fast as a few seconds, but no longer than a few minutes for networks with a quite large number of layers (see experiments in Section IV-B).

## C. Microcontroller-Specific Constraints

As with previous hardware-aware NAS approaches for microcontrollers [1, 2], we consider search spaces consisting of chain-based architectures (e.g., MobileNets [17], DS-CNNs [2]), which are made up of a sequential chain of layers containing convolutional neural networks and multi-layer perceptrons (including autoencoder architectures). For microcontrollers and their associated libraries [1, 2], it can generally be assumed that the static weights of the neural network are stored in flash memory or ROM, while the intermediate results are kept in RAM. Taking this into consideration, we can define constraints for ROM, RAM, and FLOP count as an estimate of execution time in the following.

1) *ROM Constraint*: Given a ROM bound constraint  $B^{\text{ROM}} \in \mathbb{N}$  kB. Then, a neural network  $x \in X$  is feasible iff the required ROM  $U^{\text{ROM}}(x) \leq B^{\text{ROM}}$  with

$$U^{\text{ROM}}(x) = \sum_{i=1}^L \rho_{t,i} \leq B^{\text{ROM}}, \quad (3)$$

where  $\rho_{t,i}$  denotes the number of bytes required for storing the weights of a layer  $l_i$  and the corresponding program fragments in the ROM. For a neural network layer  $l_i$ ,  $\rho_{t,i}$  can be calculated as  $\rho_{t,i} = w_{t,i} \cdot D_i + cs_t$ , where  $cs_t$  denotes the resulting code size for the respective layer of type  $t$  (e.g.,  $t \in T = \{\text{CONV}, \text{FC}, \dots\}$ ), and  $w_{t,i}$  denotes the number of weights of layer  $l_i$ <sup>3</sup> with a data type of size  $D_i$  bytes.

2) *RAM Constraint*: Given a RAM bound constraint  $B^{\text{RAM}} \in \mathbb{N}$  kB. A neural network  $x \in X$  is feasible iff the RAM usage  $U^{\text{RAM}}(x) \leq B^{\text{RAM}}$ . Now, when applying layer-sequential processing, the RAM used to store the inputs of layer  $l_i$  can be reused to store the results of the next layer  $l_{i+1}$ . Note that this assumption also holds for neural networks having inverted bottleneck convolutions or depthwise separable convolutions with residual connections, which can be fused to one layer as shown in [19, 20]. For each layer  $l_i$ , let  $u_i$  denote the number of intermediate results<sup>4</sup> (each of size  $D_i$  bytes),  $u_0$  denote the number of input neurons of the first layer  $l_1$  (each of size  $D^{\text{in}}$  bytes). The RAM usage  $U^{\text{RAM}}$  can be calculated as

<sup>3</sup>For example, the number of weights of a fully connected layer is given by  $w_{\text{FC},i} = M_i \cdot N_i$ , where  $M_i$  denotes the number of output neurons and  $N_i$  denotes the number of input neurons.

<sup>4</sup>For a fully connected layer, the number of intermediate results equals the number of output neurons  $u_i = M_i$ , for a convolutional layer  $u_i = R_i^{\text{out}} \cdot C_i^{\text{out}} \cdot M_i$ , where  $R_i^{\text{out}}$  is the number of rows and  $C_i^{\text{out}}$  denotes the number of columns of the output feature map, and  $M_i$  denotes the number of output feature maps.

$$\begin{aligned}
U_{\text{even}}(x) &= \max \left\{ \max_{1 \leq i \leq L} \{D_i \cdot u_i \mid i \text{ even}\}, D^{\text{in}} \cdot u_0 \right\} \\
U_{\text{odd}}(x) &= \max_{1 \leq i \leq L} \{D_i \cdot u_i \mid i \text{ odd}\} \\
U^{\text{RAM}}(x) &= U_{\text{even}}(x) + U_{\text{odd}}(x) \leq B^{\text{RAM}}. \quad (4)
\end{aligned}$$

3) *Number of FLOPs Constraint:* As measuring the inference time for each architecture can be very time-consuming and depends on the target hardware platform, the number of floating-point operations (FLOPs) is often used as an adequate estimate. In the case of microcontrollers, previous works have shown that the number of FLOPs of a given NN model is strongly correlated with the inference time (i.e., the higher the number of FLOPs, the higher the inference time) [2, 3]. Let  $B^{\text{FLOPs}} \in \mathbb{N}$  denote a threshold on the maximum amount of floating-point operations per network inference. Then, a neural network candidate  $x \in X$  is called feasible under this constraint iff the number of floating-point operations  $\text{FLOPs}(x) \leq B^{\text{FLOPs}}$ , with

$$\text{FLOPs}(x) = \sum_{i=1}^L R_i^{\text{out}} \cdot C_i^{\text{out}} \cdot M_i \cdot N_i \cdot K_i^{\text{w}} \cdot K_i^{\text{h}} \cdot f_i, \quad (5)$$

where  $R_i^{\text{out}}$ ,  $C_i^{\text{out}}$  denote the dimensions of the output feature map,  $M_i$  denotes the number of filters,  $N_i$  denotes the number of input feature maps,  $K_i^{\text{w}}$  and  $K_i^{\text{h}}$  the respective width and height of the filter kernel<sup>5</sup>, and  $f_i$  denotes the number of floating-point operations required per iteration for the specific layer type  $t^6$ .

#### D. Entropy-based Sampling from $X^{\text{f}}$

Even after the proposed search space reduction,  $X^{\text{f}}$  can still consist of hundreds of thousands of feasible solutions in cases. Previous works [15, 21] have introduced and applied the *principle of maximum entropy* to neural networks, based on the assumption that the entropy associated with the architecture of an NN correlates with the achievable accuracy. Given a neural network consisting of  $L$  layers, the entropy of a neural network is defined as<sup>7</sup>

$$H = \log_2(u_L) \sum_{i=1}^L \log_2(\omega_i). \quad (6)$$

In Eq. (6),  $\omega_i$  denotes the “width” of a layer<sup>8</sup>, and  $u_L$  the size of the output feature map of the last layer. In our approach, we use the entropy to prioritize sampling of NN candidates from the search space of feasible NNs in  $X^{\text{f}}$  towards “promising” solutions, i.e., those that have a higher associated entropy (value). To achieve this, we calculate the entropy of each feasible NN candidate in  $X^{\text{f}}$ . The NN candidates are then clustered according to their entropies using the well-known k-means clustering. Finally, for the exploration of  $X^{\text{f}}$ , we randomly draw samples from a geometric distribution, with probability  $P(c)$  of samples in cluster  $c$  given by

$$P(c) = (1 - p)^c \cdot p, \quad (7)$$

thus giving preference to clusters with higher entropy solutions, assuming that the clusters are sorted by descending mean of the

<sup>5</sup>For a fully connected layer  $R_i^{\text{out}} = C_i^{\text{out}} = 1$ ,  $M_i$  is the number of output neurons,  $K_i^{\text{w}} = K_i^{\text{h}} = 1$ , and  $N_i$  is the number of input neurons.

<sup>6</sup>As an example, for the multiply-accumulate operations of convolutional and fully connected layers  $f_i = 2$ , for ReLU activation layers  $f_i = 1$ .

<sup>7</sup>As proposed in [21], we use the logarithm with base 2.

<sup>8</sup> $\omega_i = M_i$  for an FC layer and  $\omega_i = (M_i \cdot K_i^{\text{w}} \cdot K_i^{\text{h}})/g_i$ , with  $g_i$  being the number of groups, i.e., for a depthwise convolutional layer ( $M_i = g_i$ ).

entropy of the contained NN candidates (see the example of  $P(c)$  for the different clusters in Fig. 2).

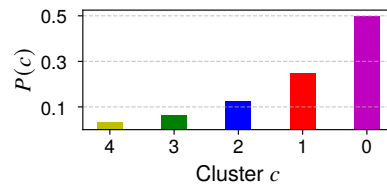


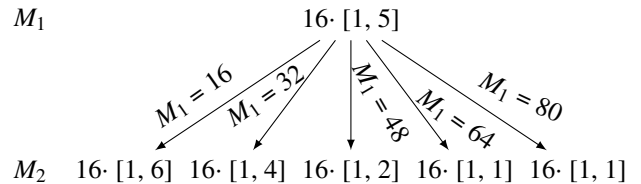
Fig. 2: Probability distribution  $P(c)$  (Eq. (7)) of the five entropy clusters  $c$  for  $p = 0.5$  for the running example.

#### E. Search Space Representation with Generic Trees

In the following, we propose representing  $X_L^{\text{f}}$  by a generic tree, also known as a general or n-ary tree, to reduce the memory needed for storing the search space [22]. Consider, for example, the search space  $X_L^{\text{f}}$  of Fig. 1, with all feasible configurations for  $M_1$  and  $M_2$ , listed in Fig. 3a, and the corresponding generic tree, see Fig. 3b. Formally, for each  $L \in \mathcal{L}$ , we construct a generic tree whose depth equals the number of layers  $L$ . Each level of the tree corresponds to one design parameter (e.g.,  $M_i$ ) and each node at level  $i$  represents a range for  $M_i$ . During construction, the values of  $M_i$  are recursively combined into ranges. The generic tree representation of feasible solutions can reduce the storage (see Section IV-B) while preserving their hierarchical structure.

$M_1$	16	...	16	32	...	32	48	48	64	80
$M_2$	16	...	96	16	...	64	16	32	16	16

(a) Search space  $X_L^{\text{f}}$  of running  $L = 2$  MLP example.



(b)  $X_L^{\text{f}}$  of running example represented by a generic tree.

Fig. 3: Example of  $X_L^{\text{f}}$  for the running example of an  $L = 2$  layer network consisting of 14 valid combinations with respective filters  $M_i$  (a), with  $1 \leq i \leq 2$ , represented as a generic tree (b).

## IV. EXPERIMENTAL EVALUATION

The following experiments were performed using the Keyword Spotting (KWS), CIFAR-10, Anomaly Detection (AD), and Visual Wake Words (VWW) datasets, all of which are part of the MLPerf Tiny Benchmark suite [23]. Based on or inspired by the literature, we consider four different backbone architectures as search spaces: depthwise separable convolutional neural networks (DS-CNN), ConvMixer [24], Autoencoder [2], and MobileNets [17]. This enables us to evaluate whether our approach can provide high-accuracy solutions in a short time and determine effective combinations of architectures and datasets. All neural network models have been trained using the Adam optimizer [25] with a learning rate of 0.001. The respective number of training epochs has been chosen as 20 for DS-CNN, 25 for the Autoencoder, and 100 for ConvMixer and MobileNet. All networks are assumed to operate on single-precision floating-point numbers. As target devices, we considered the following microcontroller units (MCUs) from

STMicroelectronics with corresponding memory constraints: A small MCU, STM32F412 ( $B^{\text{RAM}}=256$  kB,  $B^{\text{ROM}}=1$  MB), and a large MCU, STM32H743 ( $B^{\text{RAM}}=512$  kB,  $B^{\text{ROM}}=2$  MB). For determining the reduced upper bounds within our proposed search space reduction problem (see Section III-B), we utilized the APOPT solver in GEKKO [26]. For comparison with multi-shot approaches (Section IV-A), we benchmarked our approach against Bayesian optimization using the Tree-structured Parzen Estimator (TPE) [27]. For the entropy-based sampling approach (see Section III-D), the number of clusters has been fixed to 100 with  $p = 0.5$ . To compare with one-shot NAS (Section IV-B), we employed the prevalent differential architectural search (DARTS) approach [11], as used in [2] and implemented in Microsoft NNI<sup>9</sup>. The constraints have been set accordingly to meet the limitations of the small MCU in the following experiments.

1) *DS-CNN for KWS*: The search space of the DS-CNN is based on the research by Banbury *et al.* [2], in which the number of layers is based on the presented results for three found architectures declared as MicroNets. Given the possible number of layers,  $L \in \mathcal{L} = \{6, 7, 8\}$ , and for the number of filters, where we defined a step size of  $s = 28$ , and a constant lower bound  $M_i^{\text{lo}} = 84$ , and  $m_i^{\text{up}} = 8$ ,  $M_i \in \{84, 112, 140, 168, 196, 224, 252, 280\}$ ,  $1 \leq i \leq L$ , the search space consists of  $|X| = 8^6 + 8^7 + 8^8 = 19,136,512$  NN candidates. Our reduction of the search space yields  $|X^f| = 167,714$  NN candidates and took about 19 seconds on a desktop PC. Storing  $X^f$  as a list requires 5 MB, whereas modelling the search space using the proposed generic tree structure requires only 1.2 MB.

2) *ConvMixer for CIFAR-10 and MobileNets for VWW*: Trockman and Kolter [24] proposed an architecture called ConvMixer. Although no neural architecture search was performed, the authors explored different depths and numbers of filters per layer. For the latter, all layers had the same number of filters (either 128 or 256 per layer). We extend the proposed search space by adding a varying number of filters, as well as the given possible number of layers  $L^{\text{lo}} = 8$  and  $L^{\text{up}} = 16$ , with a lower bound of  $M_i^{\text{lo}} = 64$ ,  $s = 16$ , and  $m_i^{\text{up}} = 8$ . After applying our reduction method to the

search space,  $X^f$  consists of 17,154 feasible NN architectures, their representation requiring only 44 kB of storage. The search space reduction took about 2.5 minutes. Similar to the ConvMixer on CIFAR-10 and based on Banbury *et al.* [2], the MobileNet search space is defined, with the given possible number of layers  $L^{\text{lo}} = 8$  and  $L^{\text{up}} = 16$ , with a lower bound of  $M_i^{\text{lo}} = 64$ ,  $s = 16$ , and  $m_i^{\text{up}} = 8$ .

3) *Autoencoder for Anomaly Detection*: With  $L^{\text{lo}} = 5$  and  $L^{\text{up}} = 10$ ,  $M_i^{\text{lo}} = 32$ ,  $s = 16$ , and  $m_i^{\text{up}} = 8$ . For comparability with the baseline model, we fixed the size of the bottleneck layer (the final encoder layer) to eight neurons. In the encoder, each subsequent layer must have a number of output neurons that is less than or equal to the number of neurons in the previous layer. Conversely, in the decoder, each layer must have a number of output neurons that is greater than or equal to the number of neurons in the preceding layer. Then,  $X$  is constructed according to these Autoencoder properties. After search space reduction, the number of feasible solutions amounts to  $|X^f|=176,963$ . We reimplemented the described search spaces for DS-CNN and ConvMixer in NNI. Modelling the search space adhering to the Autoencoder properties was not possible due to missing functionality in NNI.

#### A. Entropy-based Sampling vs. Multi-shot Search

In the following experiment, we have tracked the exploration progress by observing the highest accuracy achieved after sampling and training  $k$  samples for three different multi-shot approaches, i.e., random search, Bayesian optimization with TPE [27], and our entropy-based sampling approach for (a) DS-CNN on KWS, (b) ConvMixer on CIFAR-10, and (c) Autoencoder on AD. The results are shown in Fig. 4. In the case of DS-CNN on KWS, in all five runs, entropy-based sampling from  $X^f$  found a solution with an accuracy of over 97.4% after around 40 samples. This accuracy was not achieved by TPE, even after exploring as many as 100 samples. This observation also holds for the ConvMixer and Autoencoder experiments, where entropy-based sampling achieves significantly higher accuracy much earlier than the other strategies. For ConvMixer, one TPE run also achieved the same best accuracy of 93.8% as entropy-based sampling. Nevertheless, for all benchmarks, random search

<sup>9</sup>Microsoft, “Neural Network Intelligence (NNI)”, <https://github.com/microsoft/nni>

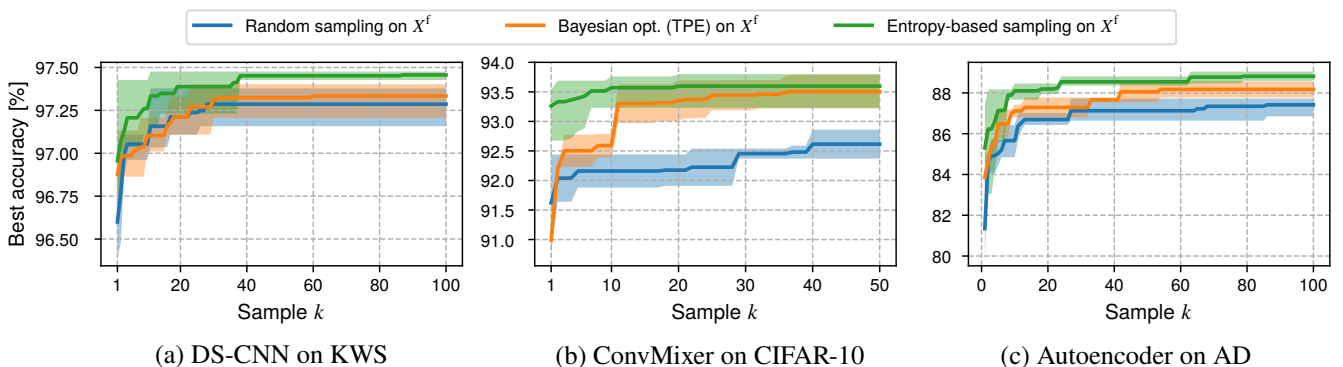


Fig. 4: Entropy-based sampling (green) vs. multi-shot approaches, i.e., random search (blue), Bayesian optimization with TPE (orange) for DS-CNN on KWS, ConvMixer on CIFAR-10, and Autoencoder on anomaly detection (AD). Shown is the best (highest) accuracy after evaluation of  $k$  samples from  $X^f$  after training for five runs. The solid lines indicate the average best accuracy.

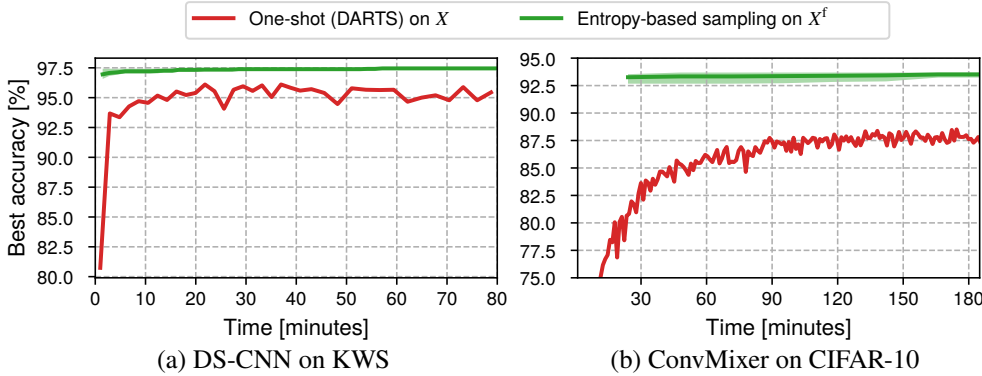


Fig. 5: Comparison of entropy-based sampling (green) and training the supermodel of DARTS (red) with respect to the achieved best accuracy over time for (a) DS-CNN search space on the KWS benchmark and (b) ConvMixer on CIFAR-10. In the case of DARTS, the time for training the supermodel and the current achieved accuracy is depicted.

and TPE tend to obtain lower average accuracy values after the same number of samples.

### B. Comparison with One-Shot Hardware-Aware NAS

Time measurements for the entropy-based sampling approach and for training the DS-CNN-supermodel on the KWS dataset were conducted on an NVIDIA GeForce RTX 2080. Fig. 5 shows the results of comparing the best accuracy of DARTS, plotted as training progress over time, with our entropy-based sampling approach for (a) DS-CNN on KWS and (b) ConvMixer on CIFAR-10. For DS-CNN, training the supermodel on the search space  $X$  for 50 epochs (80 minutes) achieved a highest accuracy of 96.1% on the KWS dataset. Entropy-based sampling (40 samples, 20 training epochs per sample) achieves a highest accuracy of 97.4% to 97.5% depending on the run. Similarly, ConvMixer on CIFAR-10 reached an accuracy of around 87.5% after 180 minutes, comparable to other similar works, where an accuracy of 85% was reported for the trained supermodel [5]. In the case of entropy-based sampling on  $X^f$ , the first trained NN candidate (100 epochs) already reached accuracy values between 92.8% and 93.5%. Unlike typical one-shot NAS methods, which require additional fine-tuning of sampled subnetworks after training the supermodel, our approach yields competitive models directly after entropy-based sampling from  $X^f$ , i.e., without any further fine-tuning. The overall results are summarized in Table I. For DS-CNN on KWS, we reimplemented and trained the architectures from [2] and trained the respective models for the same number of epochs. For the Autoencoder, we report the reference implementation results from [2, 23], which we also trained for the same number of epochs. For Zero-shot MONAS [6] and LCMNAS [9] the required ROM is derived from the number of weights. However, no values on the intermediate features maps and, in case of LCMNAS no number of operations could be found. As can be seen, our novel entropy-based sampling approach is able to find solutions that satisfy microcontroller-specific memory constraints ( $B^{\text{ROM}}$ ,  $B^{\text{RAM}}$ ) and performance requirements ( $B^{\text{FLOPs}}$ ), achieving similar or even higher accuracy across all benchmarks.

## V. CONCLUSION

We proposed a novel 2-step NAS approach that accounts for both user and resource constraints for microcontroller targets. It first reduces a given search space to one that only contains feasible NN candidates. In a subsequent entropy-based sampling

TABLE I: Top1-accuracy, ROM, RAM, and FLOPs compared for state-of-the-art configurations with our proposed entropy-based sampling (EBS) on  $X^f$  for different model and dataset pairs. If  $B^{\text{FLOPs}}$  is not specified, the number of FLOPs is not constrained.

	Acc. [%]	ROM [kB]	RAM [kB]	FLOPs
<b>DS-CNN on KWS; <math>B^{\text{ROM}}=1</math> MB; <math>B^{\text{RAM}}=256</math> kB</b>				
Micronet small [2]	96.7	236	241	11 M
<b>EBS on <math>X^f</math>, <math>B^{\text{FLOPs}}=10</math> M</b>	<b>97.1</b>	<b>212</b>	<b>213</b>	<b>10 M</b>
Micronet large [2]	97.3	1,985	611	85 M
<b>EBS on <math>X^f</math></b>	<b>97.5</b>	690	235	27 M
<b>ConvMixer on CIFAR-10; <math>B^{\text{ROM}}=1</math> MB; <math>B^{\text{RAM}}=256</math> kB</b>				
MONAS small [6]	92.2	1,360	-	47 M
<b>EBS on <math>X^f</math></b>	<b>93.8</b>	<b>912</b>	246	59 M
<b>ConvMixer on CIFAR-10; <math>B^{\text{ROM}}=2</math> MB, <math>B^{\text{RAM}}=512</math> kB</b>				
MONAS large [6]	93.9	3,400	-	121 M
LCMNAS large [9]	93.9	1,600	-	-
ConvMixer [24]	93.8	1,584	524	103 M
<b>EBS on <math>X^f</math>, <math>B^{\text{FLOPs}}=100</math> M</b>	<b>94.1</b>	<b>1,451</b>	<b>458</b>	<b>94 M</b>
<b>Autoencoder on AD; <math>B^{\text{ROM}}=1</math> MB, <math>B^{\text{RAM}}=256</math> kB</b>				
Autoencoder reference [23]	84.7	1,100	5.63	248 K
<b>EBS on <math>X^f</math></b>	<b>89.0</b>	<b>994</b>	5.76	249 K
<b>MobileNet on VWW; <math>B^{\text{ROM}}=1</math> MB, <math>B^{\text{RAM}}=256</math> kB</b>				
MicroNet-VWW-4 [2]	82.5	-	-	37 M
<b>EBS on <math>X^f</math></b>	<b>86.3</b>	805	246	<b>35 M</b>

strategy, we characterize and cluster feasible NN candidates according to their entropy, and then sample higher-entropy clusters more frequently to identify superior network architectures. We demonstrated that highly accurate solutions can be found with just a few samples. Thus, our approach enables target-specific NAS, where a highly accurate neural network can be determined quickly that meets the constraints of a given microcontroller as well as user-given execution time constraints. As part of future work, we would like to incorporate quantization into the sampling strategy based on methods of explainable AI [28].

**Acknowledgment:** This work was supported by the Schaeffler Hub for Advanced Research at Friedrich-Alexander-Universität Erlangen-Nürnberg (SHARE at FAU).

## REFERENCES

- [1] J. Lin, W. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS)*, Curran Associates Inc., 2020, pp. 11 711–11 722.
- [2] C. R. Banbury *et al.*, "MicroNets: Neural network architectures for deploying TinyML applications on commodity microcontrollers," in *Proceedings of the Conference on Machine Learning and Systems (MLSys)*, mlsys.org, 2021.
- [3] E. Liberis, L. Dudziak, and N. D. Lane, " $\mu$ NAS: Constrained neural architecture search for microcontrollers," in *Proceedings of the 1st Workshop on Machine Learning and Systems (EuroMLSys)*, ACM, 2021, pp. 70–79. doi: 10.1145/3437984.3458836.
- [4] M. Deutel, G. Kontes, C. Mutschler, and J. Teich, "Combining multi-objective Bayesian optimization with reinforcement learning for TinyML," *ACM Transactions on Evolutionary Learning and Optimization (TELO)*, vol. 5, no. 3, 17:1–17:21, 2025. doi: 10.1145/3715012.
- [5] A. Burrello, M. Risso, B. A. Motetti, E. Macii, L. Benini, and D. J. Pagliari, "Enhancing neural architecture search with multiple hardware constraints for deep learning model deployment on tiny IoT devices," *IEEE Transactions on Emerging Topics in Computing*, vol. 12, no. 3, pp. 780–794, 2024. doi: 10.1109/TETC.2023.3322033.
- [6] Y. Qiao, H. Xu, Y. Zhang, and S. Huang, "MicroNAS: Zero-shot neural architecture search for MCUs," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, IEEE, 2024, pp. 1–2. doi: 10.23919/DATe58400.2024.10546710.
- [7] J. Mellor, J. Turner, A. J. Storkey, and E. J. Crowley, "Neural architecture search without training," in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, PMLR, 2021, pp. 7588–7598.
- [8] C. White *et al.*, "Neural architecture search: Insights from 1000 papers," *The Computing Research Repository (CoRR)*, 2023. arXiv: 2301.08727 [cs.LG].
- [9] V. Lopes and L. A. Alexandre, "Toward less constrained macro-neural architecture search," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 2, pp. 2854–2868, 2025. doi: 10.1109/TNNLS.2023.3326648.
- [10] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *The Computing Research Repository (CoRR)*, 2018. arXiv: 1804.03209 [cs.CL].
- [11] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, OpenReview.net, 2019.
- [12] H. Shi, R. Pi, H. Xu, Z. Li, J. T. Kwok, and T. Zhang, "Bridging the gap between sample-based and one-shot neural architecture search with BONAS," in *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS)*, Curran Associates Inc., 2020, pp. 1808–1819.
- [13] G. Li, Y. Yang, K. Bhardwaj, and R. Marculescu, "ZiCo: Zero-shot NAS via inverse coefficient of variation on gradients," in *Proceedings of the International Conference on Learning Representations (ICLR)*, OpenReview.net, 2023.
- [14] J. Chu, X. Yu, S. Yang, *et al.*, "Architecture entropy sampling-based evolutionary neural architecture search and its application in osteoporosis diagnosis," *Complex & Intelligent Systems*, vol. 9, pp. 213–231, 2023. doi: 10.1007/s40747-022-00794-7.
- [15] X. Shen *et al.*, "DeepMAD: Mathematical architecture design for deep convolutional neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2023, pp. 6163–6173. doi: 10.1109/CVPR52729.2023.00597.
- [16] Z. Sun, M. Lin, X. Sun, Z. Tan, H. Li, and R. Jin, "MAE-DET: Revisiting maximum entropy principle in zero-shot NAS for efficient object detection," in *Proceedings of the International Conference on Machine Learning (ICML)*, PMLR, 2022, pp. 20 810–20 826.
- [17] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *The Computing Research Repository (CoRR)*, 2017. arXiv: 1704.04861v1 [cs.CV].
- [18] I. Radosavovic, R. P. Kosaraju, R. B. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Computer Vision Foundation / IEEE, 2020, pp. 10 425–10 433. doi: 10.1109/CVPR42600.2020.01044.
- [19] M. Deutel, F. Hannig, C. Mutschler, and J. Teich, "Fused-layer CNNs for memory-efficient inference on microcontrollers," in *Proceedings of the Workshop on Machine Learning and Compression @ NeurIPS 2024*, OpenReview.net, 2024.
- [20] A. Lavin, "On the efficiency of convolutional neural networks," *The Computing Research Repository (CoRR)*, 2024. arXiv: 2404.03617 [cs.LG].
- [21] D. A. Roberts, S. Yaida, and B. Hanin, "The principles of deep learning theory," *The Computing Research Repository (CoRR)*, 2021. arXiv: 2106.10165 [cs.LG].
- [22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd. MIT Press, 2009, ISBN: 978-0-262-03384-8.
- [23] C. R. Banbury *et al.*, "MLPerf Tiny Benchmark," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.
- [24] A. Trockman and J. Z. Kolter, "Patches are all you need?" *Transactions on Machine Learning Research*, 2023.
- [25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, OpenReview.net, 2015.
- [26] L. Beal, D. Hill, R. Martin, and J. Hedengren, "GEKKO optimization suite," *Processes*, vol. 6, no. 8, 2018. doi: 10.3390/pr6080106.
- [27] S. Watanabe, "Tree-structured Parzen estimator: Understanding its algorithm components and their roles for better empirical performance," *The Computing Research Repository (CoRR)*, 2023. arXiv: 2304.11127 [cs.LG].
- [28] M. Sabih, F. Hannig, and J. Teich, "Utilizing explainable AI for quantization and pruning of deep neural networks," *The Computing Research Repository (CoRR)*, 2020. arXiv: 2008.09072 [cs.CV].