ColabNAS: Obtaining lightweight task-specific convolutional neural networks following Occam's razor

Andrea Mattia Garavagno *, Daniele Leonardis, Antonio Frisoli

Institute of Mechanical Intelligence, Scuola Superiore Sant'Anna of Pisa, Piazza Martiri della Libertà, 33, Pisa, 56127, Tuscany, Italy

ARTICLE INFO

ABSTRACT

The current trend of applying transfer learning from convolutional neural networks (CNNs) trained on large datasets can be an overkill when the target application is a custom and delimited problem, with enough data to train a network from scratch. On the other hand, the training of custom and lighter CNNs requires expertise, in the from-scratch case, and or high-end resources, as in the case of hardware-aware neural architecture search (HW NAS), limiting access to the technology by non-habitual NN developers.

For this reason, we present ColabNAS, an affordable HW NAS technique for producing lightweight task-specific CNNs. Its novel derivative-free search strategy, inspired by Occam's razor, allows to obtain state-of-the-art results on the Visual Wake Word dataset, a standard TinyML benchmark, in just 3.1 GPU hours using free online GPU services such as Google Colaboratory and Kaggle Kernel.

1. Introduction

Task-specific convolutional neural networks (CNNs) are enabling the rise of next-generation smart wearable systems and distributed sensors. The limited size of the problem to be solved allows for lightweight models. However, developing from scratch a good lightweight neural network model is not easy. A recent research area named hardware-aware Neural Architecture Search (HW NAS) is facing the problem, of tailoring the search to the precise resources of the target hardware. As of today, open-source projects such as MCUNet [1] and Micronets [2] are able to produce state-of-the-art TinyML models in around 300 GPU hours. Nevertheless, lightweight task-specific CNN are commonly designed using Transfer Learning (TL) [3].

For example Sanida et al. [4], in 2022, proposed a lightweight task-specific CNN able to diagnose COVID-19 by evaluating chest X-rays. To design the network the TL procedure is applied. MobileNetV2 [5] is used as the model backbone.

A similar approach is applied by Liu et al. [6], in 2022, to detect ships in spaceborne synthetic aperture radar (SAR) images. In this case, MobileNetV2 is used as a backbone for a YOLOv4-LITE model.

In the same year, TL is also applied by Wang et al. [7] to develop a model based on Efficientnet-B1 [8], able to detect arc fault in photovoltaic systems, by analyzing power spectrum images; by Khaki et al. [9] to develop a model based on MobileNetV2 able to detect and count wheat heads by analyzing pictures; and by Jubair et al. [10] to develop a model based on EfficientNet-B0 [8] able to early detect oral cancer by pictures of the oral cavity.

Ragusa et al. used MobileNets as a backbone for performing image polarity detection using visual attention [11] and for performing affordance detection [12] on embedded devices.

The above represents a brief overview of the rich literature involving lightweight task-specific neural network designs, using TL, presented just in 2022.

TL uses the knowledge acquired by a CNN in solving a problem, to solve another similar one, reducing the training data and the required time. This technique helps when there is not enough data and/or time to design a model from scratch. On the contrary, TL can be an overkill: task-specific problems can be coped with by significantly lighter CNNs. Even the lightest models, used in the TL procedure, are trained on large-scale datasets (i.e. ImageNet1k by Deng et al. [13]). In addition, in the case of application to specific problems, related datasets are often already available for the end-users. Generating a CNN from scratch seems then a convenient choice. Yet the search cost of HW NAS, or the cost of manual design, is still too high compared to the time required by TL, possibly explaining the prominent use of TL over HW NAS in the literature.

In this paper, we propose ColabNAS, an affordable HW NAS technique for producing lightweight task-specific CNNs. It uses a novel derivative-free search strategy, inspired by Occam's razor, that allows it to obtain state-of-the-art results on the Visual Wake Word dataset [14] in just 3.1 GPU hours using free online GPU services such as Google Colaboratory and Kaggle Kernel. Such a feature inspired its name,

which wants to emphasize the ability to be run on free subscription online services, making its use available to everyone. We believe this feature is critical to foster the use of CNNs in the variety of application tasks they can adapt, especially in the field of embedded wearable and distributed devices, and implemented by a heterogeneous population of end-users and researchers.

In this paper, we present details of the proposed technique and experimental validation on five different classification problems on online-available datasets. Then, the results obtained on the Visual Wake Word dataset are compared with state-of-the-art HW NAS methods. We provide online access to experimental data and code in the form of Google Colaboratory notebooks on GitHub.

2. Related works

In recent years standard benchmarks for tiny machine learning applications were established [15]. Imagenet1k, by Deng et al. [13], and the Visual Wake Words, by Chowdhery et al. [14], datasets were chosen to measure the performance of the most recent efforts of trying to implement deep-learning techniques for computer vision tasks on commodity hardware.

These efforts followed the path opened by manually designed lightweight models such as MobileNets [5,16,17], SqueezeNet [18], ShuffleNets [19,20], and tried to automatize the design using NAS techniques, taking into consideration hardware constraints such as Flash and SRAM occupancy, or latency, giving birth to HW NAS.

Several HW NAS make use of reinforcement learning to search for the optimal architecture under the constraints set by the target hardware. A controller produces sample architectures and is rewarded based on the validation accuracy and the hardware cost. This is the case of MNASNet [21], FPNet [22], Codesign-NAS [23], and [24]. This approach commonly implies a training phase for each architecture generated, which leads to high search costs. In particular, MNASNet [21] tries to find the Pareto optimal solution of an objective function penalized by hardware constraints using reinforcement learning. The search space is hierarchical; at each iteration, an RNN, the controller, produces a sample architecture which is trained for evaluating the accuracy and then executed on the target hardware to evaluate the latency. MNASNET declares a search cost of 40,000 GPU hours.

Such issue has been addressed in then literature by proposing an over-parametrized network, known as supernetwork; it contains all the models belonging to the search space as sub-networks, which share weights. This solution requires only the training of the supernetwork, thus reducing the search time. Then, techniques like evolutionary algorithms [1,25,26] and gradient descent methods [2,27–31] are commonly used to search for the best sub-network. In details, MCUNet is composed of two parts an HW-NAS technique called TinyNAS, and a framework for executing deep learning on microcontrollers called TinyEngine. The search space is built considering the hardware constraints. Flash, SRAM occupancy and latency are taken into account and measured directly on the target hardware. Instead Micronets, also targeting microcontrollers, avoids the hardware in the loop by computing RAM and Flash memory occupancy and using op count as a viable proxy to latency.

Even if the search time has been drastically reduced, a supernetwork still requires a large amount of resources to be trained. ColabNAS introduces a novel approach to HW NAS that limits search time and resource usage. The low resource usage is guaranteed by the proposed search space, which contains regular architectures with small footprints that require few resources to be trained. The low search cost is achieved by adopting the novel derivative-free search strategy proposed in this paper, which reduces the number of solutions explored by leveraging the Occam's razor.

3. ColabNAS

This section describes the developed technique to search lightweight, task-specific CNNs.

Every HW NAS technique has three unique identification factors: the search space, the optimization problem formulation and the search strategy.

The search space defines the set of possible solutions which the optimization problem may have. The optimization problem establishes the boundaries of the search and the logic used to evaluate possible solutions. Finally, the search strategy describes the way the search space is explored, i.e. how the problem solution is found.

3.1. Search space and problem formulation

There are three types of search spaces: layer-wise, cell-wise and hierarchical. ColabNAS uses a cell-wise search space. It starts from a single bidimensional convolutional layer. Then, it continues by stacking couples of pooling and convolutional layers, like in the VGG16 architecture [32], which form a single cell as shown in Fig. 1, until the network's generalization capability increases.

The stopping criterion takes inspiration from the problem-solving principle that "entities should not be multiplied beyond necessity", also known as Occam's razor, attributed to English Franciscan friar William of Ockham, who lived between the thirteen and the fourteenth century.

The number of kernels of each cell added is determined by Eq. (1), where k represents the number of kernels used in the first convolutional layer. It is inspired by the procedure used in the VGG16 architecture [32], where a cell doubles the number of kernels with respect to the previous cell. However, in the proposed approach this amplification is modulated cell after cell, to limit the parameters' growth.

$$n_c = \begin{cases} k & if \quad c = 0 \\ \left\lceil \left(2 - \sum_{i=1}^{c-1} 2^{-i}\right) \cdot n_{c-1} \right\rceil & if \quad c \geq 1 \end{cases} \tag{1}$$

The search space is constrained by the network's peak RAM occupancy, Flash occupancy and by the number of multiply and accumulate operations (MACC) which is used as a rough estimate of latency as suggested by Banbury et al. [2].

$$P1 : \begin{cases} \max f(x) \\ \phi_R(x) \leq \xi_R \\ \phi_F(x) \leq \xi_F \\ \phi_M(x) \leq \xi_M \\ \xi_R, \xi_F, \xi_M > 0 \end{cases} \tag{2}$$

This leads to the problem formulation presented in Eq. (2), where function f returns the maximum validation accuracy obtained during training; function ϕ_R the network's peak RAM occupancy; function ϕ_F the network's Flash occupancy; function ϕ_M the network's number of MACC. These depend on the number of kernels used in the first layer k, the number of cells added c, and the magnitude of the network's input size s. The latter is omitted in the formulation since it is considered fixed during the search, hence the search variable $x = (k, c)$ is defined. $f(x)$ values come from training. The feasible search space will be referred as Ω.

3.2. Network's architecture details

Inspired by VGG16 [32] we decided to adopt convolutional layers with 3×3 kernels and zero padding to preserve the input size; pooling layers with 2×2 receptive field and (2, 2) stride.

The convolutional base output is reduced by applying the 2D Global Average Pooling operator to improve the model's generalization capability [33]. Subsequently, a deep fully connected layer, having the
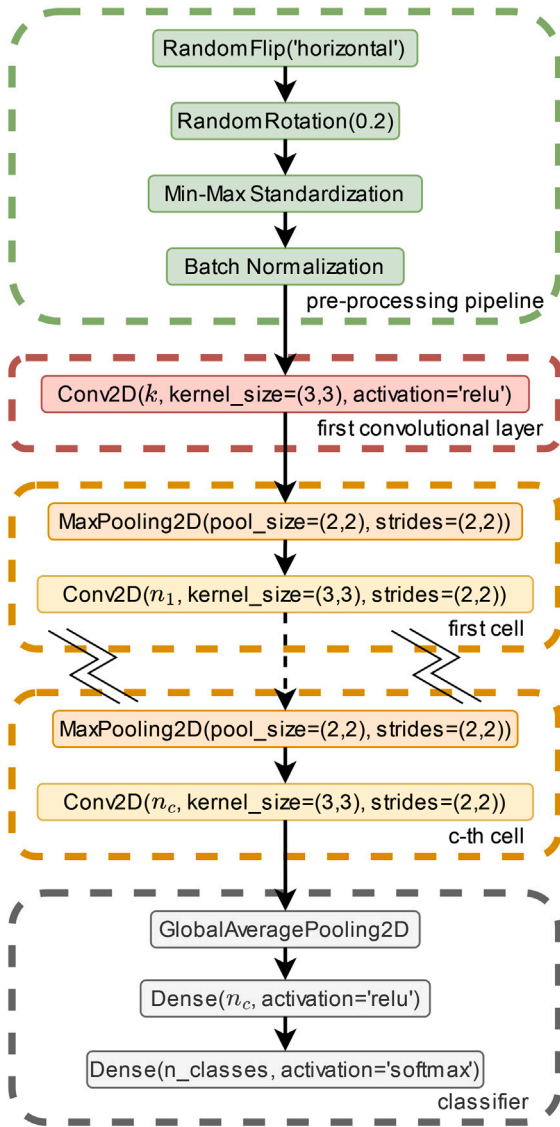
Fig. 1. Detailed graphical representation of the generic network architecture. n_c represents the number of kernels used in the cth cell. Instead, the dotted connection represents a generic number of cells added. The deep dense layer has a number of neurons equal to the kernels used in the last cell added.

number of neurons equal to the number of kernels of the last convolutional layer, further elaborates the reduced features. Finally, a single fully connected layer classifies the extracted features.

A pre-processing pipeline is included in the network's architecture. At first it applies min–max standardization, to improve gradient descent convergence rate [34]. Then, it uses batch normalization to stabilize and speed up the training [35]. Data augmentation is also applied. Both horizontal flips and random rotations are used. Fig. 1 graphically represents the generic architecture used as a search space.

3.3. Search strategy

The search strategy explores the search space in two steps. First, as stated before, given a starting number of kernels of the first layer, it starts to add cells until the network's generalization capability increases, according to Occam's razor, or until the network respects hardware constraints. Then it repeats the latter process changing the

number of kernels of the first layer according to Eq. (3).

$$k_j = \begin{cases} k_0 & if \quad j = 0 \\ 2 \cdot k_0 & if \quad j = 1 \\ 2 \cdot k_{j-1} & if \quad f(k_1^*, c_1^*) > f(k_0^*, c_0^*) \\ \frac{1}{2} \cdot k_{j-1} & if \quad f(k_1^*, c_1^*) \le f(k_0^*, c_0^*) \end{cases} \quad (3)$$

If the hardware selected for deployment allows, the process is repeated with double the number of kernels. If the solution found at the second iteration (k_1^*, c_1^*) is better than the previous one (k_0^*, c_0^*), the process continues in the same way until performance improves, if the hardware limits are not reached. Otherwise, the process continues by halving down the initial number of kernels until performance degradation is met. Also this time Occam's razor is respected. Entities are not multiplied beyond necessity.

Such a search strategy can be interpreted as a custom derivative-free method for solving the constrained optimization problem presented in Eq. (2). Algorithm 1 describes such an interpretation.

Algorithm 1 Search Strategy

Require: $x_0 \in \Omega$

Ensure: $\begin{cases} f(\widetilde{x_j}) > f(\widetilde{x_{j-1}}), \widetilde{x_j} \in \Omega & \text{always} \\ f(\widetilde{x_{j+1}}) \le f(\widetilde{x_j}) & \text{if } \widetilde{x_{j+1}} \in \Omega \end{cases}$

$\epsilon \leftarrow 0.005, k_0 \leftarrow 4, x_0 \leftarrow (k_0, 0), d \leftarrow (1, 0), t_0 \leftarrow x_0, j \leftarrow 0$
$\widetilde{x_0} = \text{EXPLORE\_NUM\_CELLS}(x_0)$
$x_1 = x_0 + t_0 \cdot d$
$\widetilde{x_1} = \text{EXPLORE\_NUM\_CELLS}(x_1)$
if $f(\widetilde{x_1}) > f(\widetilde{x_0})$ **and** $x_1 \in \Omega$ **then**
    do
        $j = j + 1$
        $t_j = x_j$
        $x_{j+1} = x_j + t_j \cdot d$
        $\widetilde{x_{j+1}} = \text{EXPLORE\_NUM\_CELLS}(x_{j+1})$
    while $f(\widetilde{x_{j+1}}) > f(\widetilde{x_j}) + \epsilon$ **and** $x_{j+1} \in \Omega$
else
    $x_1 = x_0$
    do
        $j = j + 1$
        $t_j = -\frac{x_j}{2}$
        $x_{j+1} = x_j + t_j \cdot d$
        $\widetilde{x_{j+1}} = \text{EXPLORE\_NUM\_CELLS}(x_{j+1})$
    while $f(\widetilde{x_{j+1}}) \ge f(\widetilde{x_j})$ **and** $x_{j+1} \in \Omega$
end if
Output: $\widetilde{x_j}$

It is an alternating search in the direction of the two main axes, as shown in Fig. 2. First, it explores the axis of the cells' additions (direction $d = (0, 1)$) given a starting point, using algorithm 2. Then it moves the starting point on the axis of the number of kernels of the first layer (direction $d = (1, 0)$) and repeats the search.

If the hardware selected for deployment allows, the number of kernels used in the first layer is doubled, i.e. the starting point x_0 is doubled. If the network found with the new starting point is better than the previous one, the algorithm continues doubling the starting point until the generalization capability improves, if the hardware limits are not reached. Otherwise, the algorithm continues by halving down the initial number of kernels, i.e. the starting point x_0, until performance degradation is met or an unfeasible point is found. $\widetilde{x_j}$ represents the algorithm's output.

Procedure 2 explores the axis of the cells' additions ($d = (0, 1)$) given a starting point x, with a unitary step size ($t = 1$). Practically, it continues to add cells until the generalization capability increases, if the new network is feasible.

For the sake of synthesis, in algorithm 1, x1 is explored in any case, even if it is not feasible. For the same reason the feasibility check for the

Algorithm 2 EXPLORE_NUM_CELLS(x)

Require: $x \in \Omega$

Ensure: $\begin{cases} f(x_i) > f(x_{i-1}), x_i \in \Omega & \text{always} \\ f(x_{i+1}) \leq f(x_i) & \text{if } x_{i+1} \in \Omega \end{cases}$

$\quad x_0 \leftarrow x, \, d \leftarrow (0,1), \, t \leftarrow 1, \, i \leftarrow 0$

$\quad$ **do**

$\quad\quad x_{i+1} = x_i + t \cdot d$

$\quad\quad i = i + 1$

$\quad$ **while** $f(x_{i+1}) > f(x_i)$ **and** $x_{i+1} \in \Omega$
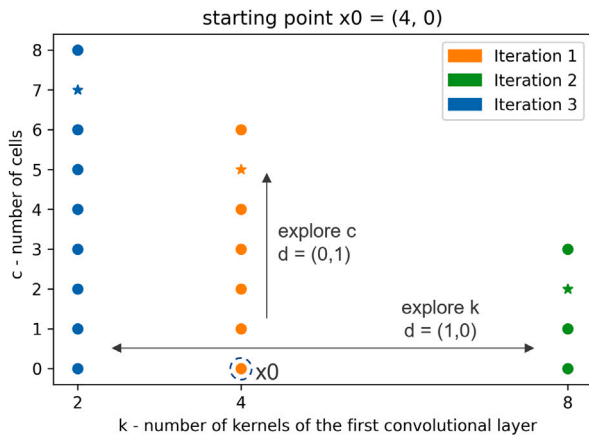
$\quad$ **return** x_i

Fig. 2. A figure that is showing the points explored by the algorithm during a sample run, in which the generalization capability does not increase during iteration 1 and gets worse during iteration 2. The star marker represents the network with the highest generalization capability for each iteration. As you can see, during each iteration the points move along the c axis. After each iteration, the starting point moves along the k axis.

point x_{j+1} is done after the exploration. Something similar happens in algorithm 2, where the network x_{i+1} is trained even if it is not feasible.

4. Experimental methods

Five task-specific classification problems are chosen to evaluate the efficacy of the proposed technique. First the models obtained by using ColabNAS are compared with the ones obtained by applying TL, a common way to design lightweight task-specific CNN [3]. The aim of this comparison is to show that ColabNAS is able to obtain much smaller models than TL at the cost of a few percentage points of accuracy on the test set, in an acceptable amount of time. Then, three different MCUs are used to show case the ability to adapting to different hardware, in low-RAM conditions, of ColabNAS. Finally, a comparison with state-of-the-art HW NAS, based on the Visual Wake Word dataset, a standard TinyML benchmark [15], is presented.

Code, models and datasets are publicly available in the form of Google Colaboratory's notebooks at the link in Section 1. All the computations have been executed on Google Colaboratory using Tesla T4 GPUs, as of today, the ones mostly offered to free subscription users.

The five classification problems have been chosen to provide a set of common task-specific datasets, publicly available with a variety of sizes spanning from a few thousand images to hundreds of thousand of images, also ensuring the possibility of reproducing the experiments proposed in this paper on free-subscription online GPU programs. Where test splits were not provided, they have been built using a 0.2 test split. i.e. 80% of data for training, 20% for testing. A brief description of the tasks follows hereafter.

4.1. Melanoma Skin Cancer

The Melanoma Skin Cancer classification task aims to discriminate between benign and malignant images of melanoma skin cancer. It is composed of a training set containing 9605 images and a test set of 1000 images. It is publicly available on kaggle under the name of "Melanoma Skin Cancer Dataset of 10 000 Images".[1]

4.2. Visual Wake Words

The Visual Wake Words classification task by Chowdhery et al. [14] aims to discriminate between images with and without human presence. It is composed of a training set containing 115,000 and a test set containing 8000 images, derived by the "minival ids" as done by Chowdhery et al. [14].

4.3. Animals-3

The Animals-3 classification task aims to discriminate between three species of animals: horse, butterfly and hen. It contains 2623 instances of horses, 2112 instances of butterflies, and 3098 instances of hens. It is a subset of the Animals-10 dataset, which is publicly available on kaggle.[2]

4.4. Flowers-4

The Flowers-4 classification task aims to discriminate between four classes of flowers: dandelion, iris, tulip, and magnolia. It contains 1052 instances of dandelion, 1054 instances of iris, 1048 instances of tulip, and 1048 instances of magnolia. It is a subset of the Flowers dataset, which is publicly available on Kaggle.[3]

4.5. MNIST

The MNIST [36] classification task aims to discriminate between ten handwritten digits: 0, 1, 2, ..., 9. It is a subset of a larger set available from NIST. It is composed of a training set containing 60,000 images and of a test set containing 10,000 images.

5. Comparison with the networks obtained using transfer learning

This section compares ColabNAS with TL. To perform TL, MobileNetV2 [5], with frozen weights trained on ImageNet1k [13], is used as the backbone. The extracted features are then compressed by a bi-dimensional global average pooling layer and then given to a shallow classifier, composed of only one dense layer. The pre-processing pipeline is the same of ColabNAS, reported in Section 3. It first applies both horizontal flips and random rotations, then applies min–max standardization, and finally uses batch normalization. Such models are trained for 20 epochs using a learning rate of 10^{-3} and a batch size of 128. Then, the trained model is fine-tuned for 10 epochs, with the backbone unfrozen, using a learning rate of 10^{-5}.

Instead, ColabNAS is run using as constraints the characteristics of the models obtained using TL, i.e. RAM occupancy, ROM occupancy, and MACC. Candidate solutions are trained for 100 epochs with a learning rate of 10^{-3} and a batch size of 128. The same input size of $224 \times 224 \times 3$, imposed by pre-trained MobileNetV2, is used for all the datasets. Results for the Visual Wake Words and MNIST dataset are not included in this comparison because of their high number of

---

[1] https://www.kaggle.com/datasets/hasnainjaved/melanoma-skin-cancer-dataset-of-10000-images?resource=download.

[2] https://www.kaggle.com/datasets/alessiocorrado99/animals10.

[3] https://www.kaggle.com/datasets/l3llff/flowers.

Table 1

Test accuracy, peak RAM occupancy, Flash occupancy, MACC and search cost of the resulting models of TL and ColabNAS (our proposed method) for the Melanoma Skin Cancer dataset.

	Test Acc. [%]	RAM [kiB]	Flash [kiB]	MACC [MM]	Search cost [hh]:[mm]
TL	88	2523	2650	300	00:16
Our	91.1	547	75.2	44	03:52

Table 2

Test accuracy, peak RAM occupancy, Flash occupancy, MACC and search cost of the resulting models of TL and ColabNAS (our proposed method) for the Animals-3 dataset.

	Test Acc. [%]	RAM [kiB]	Flash [kiB]	MACC [MM]	Search cost [hh]:[mm]
TL	99.2	2523	2653	300	00:11
Our	93.2	988	197.3	153	03:02

Table 3

Test accuracy, peak RAM occupancy, Flash occupancy, MACC and search cost of the resulting models of TL and ColabNAS (our proposed method) for the Flowers-4 dataset.

	Test Acc. [%]	RAM [kiB]	Flash [kiB]	MACC [MM]	Search cost [hh]:[mm]
TL	99.2	2523	2653	300	00:07
Our	93.8	546	59.7	44	01:20

Table 4

Available RAM and Flash, and the CoreMark score of each hardware target considered for the experiment.

STM32 MCU	RAM [kiB]	Flash [kiB]	CoreMark
L010RBT6	20	128	75
L151UCY6DTR	32	256	93
L412KBU3	40	128	273

Table 5

Test accuracy, RAM and Flash occupancy, MACC and search cost of the resulting models for each target, indicated in its abbreviated form on the first column, for the Melanoma Skin Cancer dataset.

Target abbr.	Acc [%]	RAM [kiB]	Flash [kiB]	MACC [k]	Search cost [hh]:[mm]
L0	86.5	19.5	8.3	92	00:17
L1	88.7	22	14.4	654	00:19
L4	90.1	32.5	31.84	2075	00:17

Table 6

Test accuracy, RAM and Flash occupancy, MACC and search cost of the resulting models for each target, indicated in its abbreviated form on the first column, for the Visual Wake Words dataset.

Target abbr.	Acc [%]	RAM [kiB]	Flash [kiB]	MACC [k]	Search cost [hh]:[mm]
L0	69.4	19	8.02	227	2:11
L1	74.5	22.5	18.5	657	3:04
L4	77.8	33	44.9	2086	2:47

images which makes ColabNAS exceed the maximum amount of time guaranteed for free users on Google Colaboratory.

The metrics used for the comparison are the test accuracy, the peak RAM occupancy, the Flash occupancy and the number of MACC. Post training quantization (PTQ) has been applied to all the models. X-CUBE-AI software from ST Microelectronics has been used to measure the peak RAM occupancy and the Flash occupancy, while the number of MACC has been computed using Keras backend of Tensorflow.

5.1. Melanoma Skin Cancer

Table 1 shows the comparison for the Melanoma Skin Cancer classification task. In this case, ColabNAS outperformed the result obtained by applying TL, by all means. ColabNAS produced a network that is 3.1 per cent points more precise, occupies 4.6 times less RAM, 35.2 times less Flash and performs 5.2 times less MACC compared to the TL's result.

5.2. Animals-3

Table 2 shows the comparison for the Animals-3 classification task. In this case, ColabNAS produced a model that occupies 2.6 times less RAM, 13.4 times less Flash and performs 2 times less MACC while being 6 per cent points less precise compared to the TL's result.

5.3. Flowers-4

Finally, Table 3 shows the comparison for the Flowers-4 classification task. In this case, ColabNAS produced a model that occupies 4.6 times less RAM, 44.4 times less Flash and performs 6.8 times less MACC while being 5.4 per cent points less precise compared to the TL's result.

6. Evaluation of the hardware-aware feature

This section evaluates the hardware-aware feature of ColabNAS by providing results for different hardware targets. To highlight the ColabNAS feature of providing lightweight CNNs, three low-RAM STMicroelectronics (STM) MCUs from the Ultra-low Power series have been selected: the L010RBT6 (abbr. L0), the L151UCY6DTR (abbr. L1) and

the L412KBU3 (abbr. L4). The generated networks were evaluated using the STM32 X-Cube-AI software.

Table 4 summarizes the key features of each MCU, i.e., available RAM and Flash, and CoreMark score, a benchmark for comparing performances of commercial MCUs. The values in Table 4 set the constraints for running ColabNAS on each target. RAM and Flash were provided as is, while the MACC upper bound was obtained multiplying by 10^4 the CoreMark score of the target, in order to allow a fair exploration of the search space. $50 \times 50 \times 3$ input size has been used to cope with MCUs constrained resources. The results for each dataset follow.

6.1. Melanoma Skin Cancer

Table 5 shows the resulting models for each target for the Melanoma Skin Cancer dataset. As can be seen, ColabNAS was able to provide a feasible model for each target, adapting to the resource available. The larger the target's resources the larger the model is. Models show a mean test accuracy drop of 1.8 per cent points while lowering the target's resources. The drop is more significant while passing from L1, the medium target, to L0, the small target.

6.2. Visual Wake Words

Table 6 shows the resulting models for each target for the Visual Wake Words dataset. As can be seen, ColabNAS was able to provide a feasible model for each target, adapting to the resource available. The larger the target's resources the larger the model is. Models show a mean test accuracy drop of 4.2 per cent points while lowering the target's resources. The drop is more significant while passing from L1, the medium target, to L0, the small target.

6.3. Animals-3

Table 7 shows the resulting models for each target for the Animals-3 dataset. ColabNAS was able to provide a feasible model for each target, adapting to the resource available. The larger the target's resources the larger the model is. Models show a mean test accuracy drop of 8.65

Table 7
Test accuracy, RAM and Flash occupancy, MACC and search cost of the resulting models for each target, indicated in its abbreviated form on the first column, for the Animals-3 dataset.

Target abbr.	Acc [%]	RAM [kiB]	Flash [kiB]	MACC [k]	Search cost [hh]:[mm]
L0	67.9	19	8.03	227	00:09
L1	75.8	22.5	18.65	657	00:17
L4	85.2	33	44.86	2086	00:09

Table 8
Test accuracy, RAM and Flash occupancy, MACC and search cost of the resulting models for each target, indicated in its abbreviated form on the first column, for the Flowers-4 dataset.

Target abbr.	Acc [%]	RAM [kiB]	Flash [kiB]	MACC [k]	Search cost [hh]:[mm]
L0	79.2	18.5	6.67	211	00:05
L1	84.4	21.5	10.91	633	00:08
L4	91.3	32.5	31.91	2075	00:10

Table 9
Test accuracy, RAM and Flash occupancy, MACC and search cost of the resulting models for each target, indicated in its abbreviated form on the first column, for the MNIST dataset.

Target abbr.	Acc [%]	RAM [kiB]	Flash [kiB]	MACC [k]	Search cost [hh]:[mm]
L0	88.2	19.5	9.79	233	01:07
L1	95.6	22.5	18.80	657	01:41
L4	98	33	45.23	2087	01:29

per cent points while lowering the target's resources. The drop is more significant while passing from L1, the medium target, to L0, the small target.

6.4. Flowers-4

Table 8 shows the resulting models for each target for the Flowers-4 dataset. As can be seen, ColabNAS was able to provide a feasible model for each target, adapting to the resource available. The larger the target's resources, the larger the model is. Models show a mean test accuracy drop of 6.05 per cent points while lowering the target's resources. In this case, the drop is more significant while passing from L4, the large target, to L1, the medium target.

6.5. MNIST

Table 9 shows the resulting models for each target for the MNIST dataset. ColabNAS was able to provide a feasible model for each target, adapting to the resource available. The larger the target's resources the larger the model is. Models show a mean test accuracy drop of 4.9 per cent points while lowering the target's resources. The drop is more significant while passing from L1, the medium target, to L0, the small target.

6.6. Final considerations

ColabNAS proven to be able to adapt to targets with low RAM availability, providing feasible solutions tailored to the available resources. The obtained models seem to depend more on the target's characteristics than on the dataset. Chosen a target, resulting model's RAM and MACC seem to remain almost constant across the different datasets explored in this section, while the flash occupancy shows higher variations. The search time appears proportional to the number of images in the dataset, given the fixed input size. By further lowering the input size, it should be possible to obtain networks with even smaller RAM occupancies, probably at the cost of further reducing test accuracies.

Table 10
Test accuracy, RAM and Flash occupancy, latency and input size for the chosen models from Micronets, MCUNET and ColabNAS (our) for the Visual Wake Words dataset.

Project	Acc [%]	RAM [kiB]	Flash [kiB]	Latency [mS]	Input size
MCUNet	87.4	168.5	530.52	2.16	$64 \times 64 \times 3$
Micronets	76.8	70.5	273.81	1.15	$50 \times 50 \times 3$
Our	77.6	31.5	20.83	0.432	$50 \times 50 \times 3$

7. Comparison with state-of-the-art hardware-aware NAS techniques on the Visual Wake Word dataset

This section compares ColabNAS with two state-of-the-art HW NAS techniques for CNNs: MicroNets by Banbury et al. [2] and MCUNet by Lin et al. [1]. The comparison is based on the Visual Wake Word dataset [14], a standard benchmark for TinyML models [15]. Imagenet1k [13] is not included, given its general-purpose nature, which is not compatible with the task-specific nature of our method.

Only the model with lowest RAM occupation has been selected from both projects.[4] Then, we ColabNAS has been ran on the Visual Wake Word dataset using the same hardware target of Micronets, the one having the smallest RAM among the targets of the two projects: the STMF446RE MCU having 128 kiB of RAM, 512 kiB of Flash and a CoreMark score of 608. The input size is also the same as Micronets: $50 \times 50 \times 3$.

Table 10 shows the comparison results, which are also graphically represented in Fig. 3. All the models are in TFLite format. They are fully quantized to perform 8-bit inference. Test accuracy, RAM occupancy and Flash occupancy are compared, as in the previous cases. Since there is no direct way to measure MACC using the TFLite API, no value for them is presented. To have a comparison of the execution times, a latency value is measured using the IPython magic command "%timeit" alongside the TFLite interpreter invocation. The input tensor content is random. All the measurements were performed during the same Google Colaboratory session with a dual-core Intel(R) Xeon(R) CPU at 2.20 GHz. No GPU was involved.

Our model improves by all means the one proposed by Micronets. However, the solution proposed by MCUNet is still more accurate. It offers 9.8 per cent points accuracy higher than our solution while occupying 5.35 times more RAM, 25.47 times more Flash and being 5 times slower.

7.1. Search costs comparison

As stated by Lin et al. [1] MCUNet spent 300 GPU hours to produce its architecture. Instead, MicroNets by Banbury et al. [2] does not declare the time spent on finding the architecture. However, they run DNAS for 200 epochs to find it. DNAS by Liu et al. [28], uses 1.5 GPU days (does not include the selection cost (1 GPU day) or the final evaluation cost by training the selected architecture from scratch (1.5 GPU days)) to find a network over the CIFAR-10 dataset, which contains 60 000 32×32 color images in 10 classes, with 6000 images per class. In this case, DNAS is run for 100 epochs. Considering that the visual wake word training set is composed of 107,954 images, that the input volume used is $50 \times 50 \times 1$ and that the epochs used are double, the time spent on it is higher than the time spent on the CIFAR-10. Hence ColabNAS results considerably faster than MCUNet and MicroNets.

---

[4] In the case of MicroNets the "vww2_50_50_INT8.tflite" model has been downloaded from the ARM's GitHub web page https://github.com/ARM-software/ML-zoo/blob/master/models/visual_wake_words/micronet_vww2/tflite_int8/vww2_50_50_INT8.tflite. In the case of MCUNet, the "mcunet-10fps-vww" model has been downloaded from the laboratory's web page https://hanlab.mit.edu/projects/tinyml/mcunet/release/mcunet-10fps_vww.tflite.
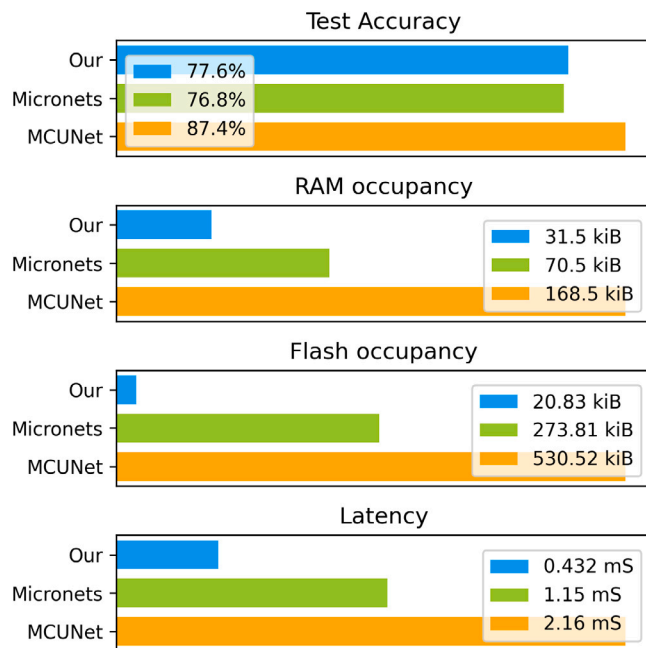
Fig. 3. Graphical representation of test accuracy, RAM and flash occupancy, latency and input size for the chosen models from Micronets, MCUNET and ColabNAS (our) for the Visual Wake Words dataset.

8. Conclusion

In this paper, we proposed ColabNAS: an affordable HW NAS technique for designing task-specific CNNs, able to work with low-RAM MCUs. Its novel search strategy, inspired by Occam's razor, has a low search cost. It obtains state-of-the-art results on the visual wake word dataset in just 3.1 GPU hours, improving, by all means, the solution found by Micronets.

The low search cost allows its execution on free online services such as Kaggle Kernel or Google Colaboratory, without owning a high-end GPU. Given the current global chip shortage, this can help those end-users and researchers who want to approach application of lightweight CNNs for custom classification problems.

It also provides task-specific lightweight CNNs that occupy on average 3.9 times less RAM, 31 times less Flash, and have 4.6 less MAC than those obtained by applying TL and FT, based on MobileNetV2 on the same task, one of the most used techniques to obtain task-specific lightweight CNNs in research. This is at the cost of losing 2.8 per cent accuracy points on average. Such a trade-off can target deployment on the growing field of wearable and distributed devices with embedded electronics hardware. Moreover, reducing the computational cost of the final application turns into more efficient power consumption. In a world where artificial intelligence is becoming more and more diffused, reducing energy consumption is a fundamental step towards a more sustainable future.

CRediT authorship contribution statement

Andrea Mattia Garavagno: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing – original draft. **Daniele Leonardis:** Writing – original draft, Data plotting, Supervision. **Antonio Frisoli:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the link to my data/code in the paper.

References

[1] J. Lin, W.M. Chen, H. Cai, C. Gan, S. Han, Memory-efficient patch-based inference for tiny deep learning, Adv. Neural Inf. Process. Syst. 34 (2021) 2346–2358.

[2] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V.j. Reddi, M. Mattina, P. Whatmough, Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers, Proc. Mach. Learn. Syst. 3 (2021) 517–532.

[3] M. Iman, H.R. Arabnia, K. Rasheed, A review of deep transfer learning and recent advancements, in: MDPI, 2023.

[4] T. Sanida, A. Sideris, D. Tsiktsiris, M. Dasygenis, Lightweight neural network for COVID-19 detection from chest X-ray images implemented on an embedded system, Technologies 10 (2022) 37, http://dx.doi.org/10.3390/technologies10020037.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: CVPR, 2018.

[6] S. Liu, W. Kong, X. Chen, M. Xu, M. Yasir, L. Zhao, J. Li, Multi-scale ship detection algorithm based on a lightweight neural network for spaceborne SAR images, Remote Sens. 14 (2022) 1149, http://dx.doi.org/10.3390/rs14051149.

[7] Y. Wang, C. Bai, X. Qian, W. Liu, C. Zhu, L. Ge, A DC series arc fault detection method based on a lightweight convolutional neural network used in photovoltaic system, Energies 15 (2022) 2877, http://dx.doi.org/10.3390/en15082877.

[8] M. Tan, Q.V. Le, EfficientNet: Rethinking model scaling for convolutional neural networks, in: Proceedings of the 36th International Conference on Machine Learning, Vol. 97, PMLR, 2019, pp. 6105–6114.

[9] S. Khaki, N. Safaei, H. Pham, L. Wang, WheatNet: A lightweight convolutional neural network for high-throughput image-based wheat head detection and counting, Neurocomputing 489 (2022) 78–89, http://dx.doi.org/10.1016/j.neucom.2022.03.017.

[10] F. Jubair, O. Al-karadsheh, D. Malamos, S. Al Mahdi, Y. Saad, Y. Hassona, A novel lightweight deep convolutional neural network for early detection of oral cancer, Oral Dis. 28 (2022) 1123–1130, http://dx.doi.org/10.1111/odi.13825.

[11] E. Ragusa, T. Apicella, C. Gianoglio, R. Zunino, P. Gastaldo, Design and deployment of an image polarity detector with visual attention, Cogn. Comput. 14 (2022) 261–273, http://dx.doi.org/10.1007/s12559-021-09829-6.

[12] E. Ragusa, C. Gianoglio, S. Dosen, P. Gastaldo, Hardware-aware affordance detection for application in portable embedded systems, IEEE Access 9 (2021) 123178–123193, http://dx.doi.org/10.1109/ACCESS.2021.3109733.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, Li Fei-Fei, ImageNet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255, http://dx.doi.org/10.1109/CVPR.2009.5206848.

[14] A. Chowdhery, P. Warden, J. Shlens, A. Howard, R. Rhodes, Visual wake words dataset, 2019, arXiv preprint, arXiv:1906.05721.

[15] C.R. Banbury, V.J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, ..., P. Yadav, Benchmarking tinyml systems: Challenges and direction, 2020, arXiv preprint arXiv:2003.04821.

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

[17] A. Howard, M. Sandler, G. Chu, L.C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q.V. Le, H. Adam, Searching for mobilenetv3, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1314–1324.

[18] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size, 2016, arXiv preprint arXiv:1602.07360.

[19] X. Zhang, X. Zhou, M. Lin, J. Sun, Shufflenet: An extremely efficient convolutional neural network for mobile devices, in: CVPR, 2018.

[20] N. Ma, X. Zhang, H.T. Zheng, J. Sun, Shufflenet v2: Practical guidelines for efficient cnn architecture design, in: ECCV, 2018.

[21] M. Tan, B. Chen, R. Pang, V. Vasudevan, Q.V. Le, Mnasnet: Platform-aware neural architecture search for mobile, 2018, CoRR, abs/1807.11626.

[22] Y. Yang, C. Wang, L. Gong, X. Zhou, Fpnet: customized convolutional neural network for fpga platforms, in: 2019 International Conference on Field-Programmable Technology (ICFPT), IEEE, 2019, pp. 399–402.

[23] M.S. Abdelfattah, Ł. Dudziak, T. Chau, R. Lee, H. Kim, N.D. Lane, Best of both worlds: Automl codesign of a cnn and its hardware accelerator, in: 2020 57th ACM/IEEE Design Automation Conference (DAC), IEEE, 2020, pp. 1–6.

[24] L. Yang, Z. Yan, M. Li, H. Kwon, L. Lai, T. Krishna, V. Chandra, W. Jiang, Y. Shi, Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks, in: 2020 57th ACM/IEEE Design Automation Conference (DAC), IEEE, 2020, pp. 1–6.

[25] H. Cai, C. Gan, T. Wang, Z. Zhang, S. Han, Once-for-all: Train one network and specialize it for efficient deployment, 2019, arXiv preprint arXiv:1908.09791.

[26] A. Marchisio, A. Massa, V. Mrazek, B. Bussolino, M. Martina, M. Shafique, Nascaps: A framework for neural architecture search to optimize the accuracy and hardware efficiency of convolutional capsule networks, in: Proceedings of the 39th International Conference on Computer-Aided Design, 2020, pp. 1–9.

[27] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, K. Keutzer, Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 10734–10742.

[28] H. Liu, K. Simonyan, Y. Yang, DARTS: differentiable architecture search, in: 7th International Conference on Learning Representations, ICLR, 2019.

[29] X. Zheng, C. Yang, S. Zhang, Y. Wang, B. Zhang, Y. Wu, Y. Wu, L. Shao, R. Ji, Ddpnas: Efficient neural architecture search via dynamic distribution pruning, Int. J. Comput. Vis. (2023) 1–16.

[30] S. Li, Y. Mao, F. Zhang, D. Wang, G. Zhong, Dlw-nas: Differentiable light-weight neural architecture search, Cogn. Comput. (2022) 1–11.

[31] H. Cai, L. Zhu, S. Han, Proxylessnas: Direct neural architecture search on target task and hardware, 2018, CoRR, abs/1812.00332.

[32] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.

[33] M. Lin, Q. Chen, S. Yan, Network in network, 2014, CoRR, abs/1312.4400.

[34] M. Shanker, M.Y. Hu, M.S. Hung, Effect of data standardization on neural network training, Omega 24 (4) (1996) 385–397, http://dx.doi.org/10.1016/0305-0483(96)00010-2.

[35] S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Vol. 37, PMLR, 1996, pp. 448–456.

[36] Y. LeCun, C. Cortes, C.J.C. Burges, MNIST handwritten digit database, 2010, ATT Labs [Online], available: http://yann.lecun.com/exdb/mnist.

Andrea Mattia Garavagno received his B.Sc. degree in Electronic Engineering from the University of Genova, Italy, in 2018 and his M.Sc. degree with honours in Embedded Computing Systems, from Scuola Superiore Sant'Anna and the University of Pisa, Italy, in 2022. He co-authored the Italian book "Introduzione al Progetto di Sistemi a Microprocessore" and the English book, "Introduction to Microprocessor-Based Systems Design" both published by Springer.

Daniele Leonardis received the BS and MS degrees in automation engineering from the Politecnico of Bari, Italy, in 2007 and 2009, respectively, and the Ph.D. degree in Innovative technologies at Scuola Superiore Sant'Anna, Pisa, Italy, in 2015. He is a researcher in mechanical engineering at the Institute Of Mechanical Intelligence: his research interests deal with haptic feedback applied to telepresence and virtual embodiment, bio-signals for controlling wearable robotic devices applied for rehabilitation and assistance.

Antonio Frisoli received the M.Sc. degree in mechanical engineering from Scuola Superiore Sant'Anna, Italy, in 1988, and the Ph.D. degree with honours in industrial and information engineering from Scuola Superiore Sant'Anna, Italy, in 2002. He is full professor of mechanical engineering with Scuola Superiore Sant'Anna, where he is currently a head of the Human–Robot Interaction area at the Institute of Mechanical Intelligence and former chair of the IEEE Technical Committee on Haptics. His research interests concern the design and control of haptic devices and robotic systems, rehabilitation robotics, advanced HRI, and kinematics. He is a member of the IEEE.