





# An Affordable Hardware-Aware Neural Architecture Search for Deploying Convolutional Neural Networks on Ultra-Low-Power Computing Platforms

Andrea Mattia Garavagno<sup>1,2</sup> , Edoardo Ragusa<sup>1\*</sup> , Antonio Frisoli<sup>2\*\*</sup> , and Paolo Gastaldo<sup>1</sup> 

<sup>1</sup>Department of Electrical, Electronic, Telecommunications Engineering, Naval Architecture (DITEN), University of Genoa, 16126 Genoa, Italy

<sup>2</sup>Department of Excellence in Robotics & AI, Institute of Mechanical Intelligence, Scuola Superiore Sant'Anna, 56127 Pisa, Italy

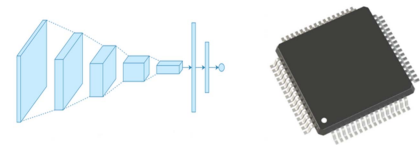
\*Member, IEEE

\*\*Senior Member, IEEE

Manuscript received 2 February 2024; revised 13 March 2024 and 26 March 2024; accepted 8 April 2024. Date of publication 10 April 2024; date of current version 22 April 2024.

**Abstract**—Hardware-aware neural architecture search (HW-NAS) allows the integration of convolutional neural networks (CNNs) in microcontrollers devices by automatically designing neural architectures that can fit prearranged hardware constraints. However, state-of-the-art HW-NAS target high-performance microcontrollers, whose power consumption does not meet sensing nodes requirements. This letter presents a HW-NAS generating tiny CNNs that can run on ultra-low-power microcontrollers, featuring a lightweight search procedure enabling its execution even on embedded devices. Empirical results on three well-known benchmarks for tiny computer vision proved that the proposed HW-NAS was able to generate tiny CNNs while preserving state-of-the-art classification accuracy.

## Automatic Design of Convolutional Neural Networks for Ultra-Low-Power Computing Platforms



**Index Terms**—Sensor applications, hardware-aware neural architecture search (HW-NAS), TinyML, ultra-low-power computing.

## I. INTRODUCTION

Sensing nodes can benefit from real-time inference procedures supported by machine learning algorithms. For example, teleceptive sensing, i.e., sensing without physical contact (e.g., optical sensors, radars, or depth sensors), strongly relies on convolutional neural networks (CNNs) [1]. CNNs, though, bring about high computing costs and nontrivial training procedures. These features are not compliant with sensing nodes that must deal with severe energy constraints [2]. Hardware-aware neural architecture search (HW-NAS) is an appealing solution because it provides an automatic procedure that generates and trains suitable CNNs given the constraints that characterize the physical device running the inference phase.

The existing HW-NAS [3], [4], [5] generates tiny CNNs suitable for execution in high-end microcontroller units (MCUs) empowered with large memories; indeed, they in general involve resource-demanding optimization procedures. Two distinct qualities would allow a broader utilization of such techniques in sensing nodes: 1) the ability of generating tiny CNNs compliant with tighter hardware constraints (e.g., when the target is a ultra-low power MCU) and 2) a limited computation cost of the NAS procedure. In practice, a procedure requiring hundreds of hours of graphical processing unit (GPU) computing may not be suitable in some applications. For example, a gateway producing custom CNNs for internet of things (IoT) edge nodes on local data, without accessing to the cloud, needs a HW-NAS so that light can run on embedded devices.

Battery-operated nodes often rely on ultra-low-power MCUs as computing engine; near-sensor computing [6] is a very good example.

Corresponding author: Andrea Mattia Garavagno (e-mail: [AndreaMattia.Garavagno@edu.unige.it](mailto:AndreaMattia.Garavagno@edu.unige.it)).

Associate Editor: Jesus M Corres.

Digital Object Identifier 10.1109/LENS.2024.3387056

Recent works [7], [8], [9] proved that NAS techniques can be exploited for the design of CNNs that comply with the tight constraints imposed by such class of microcontrollers. In addition, preliminary results showed that, in the case of very constrained networks, the search procedure can be also executed without the support of GPUs.

This brief aims to give further insights on the approach to HW-NAS proposed in [8]. In particular, the goals are as follows:

- 1) to prove empirically that even using a lightweight search procedure the tailored HW-NAS can adapt to the hardware constraints imposed by the target platform; experiments show that the revised procedure can generate custom tiny CNNs that can run in real-time on low-power MCUs;
- 2) to show that the search can run on embedded systems; to do this, a constraint on the available memory of the device that executes the NAS procedure is added to the optimization problem.

In detail, the generated CNNs achieved state-of-the-art results in the human-recognition tasks on the Visual Wake Word (VWW) dataset, a standard TinyML benchmark. The HW-NAS procedure ran on a laptop with 16 GB of RAM in less than 4 CPU hours.<sup>1</sup>

## II. BACKGROUND

### A. Related Works

In recent years, great efforts have been made to implement CNNs on devices with limited computational capability. Manually designed CNNs for mobile devices [10], [11], [12] have surged. Then, the focus shifted to approaches that tried to automatize the process [13], [14], which gave birth to HW-NAS, i.e., an automatized procedure that takes

<sup>1</sup>code available at <https://github.com/AndreaMattiaGaravagno/NanoNAS>.

Table 1. Hardware Features of the High-End MCUs Targeted by State-of-the-Art HW-NAS versus the MCUs Targeted by This Work [15]

STM32 MCU	RAM [kiB]	Flash [kiB]	CoreMark
high performance line			
F412ZG	256	1024	608
ultra-low-power line			
L010RBT6	20	128	75
L151UCY6DTR	32	256	93
L412KBU3	40	128	273

into consideration the available resources of the target hardware, (e.g., the amount of RAM, flash memory, or FLOPS).

State-of-the-art HW-NAS targeted the high-performance MCUs line of STMicroelectronics:  $\mu$ NAS [5], MCUNet [4], and Micronets [3]. Table 1 summarizes the differences between a high-end MCU and the widespread ultra-low-power line hosted in many sensing nodes. Recently, Garavagno et al. [7], [8], [9] addressed the design of a HW-NAS that targets ultra-low-power MCUs. These works showed that the constraints imposed by the ultra-low-power line require an ad-hoc approach that deviates from state-of-the-art HW-NAS.

### B. Lightweight HW-NAS: Overview

Three hallmarks characterize a HW-NAS: the search space, the optimization problem, and the search strategy. The HW-NAS presented in [8] relies on constrained cell-wise search space, setting the basis for achieving two goals: 1) the ability to generate CNNs that fit hard constraints on the available resources of the target hardware and 2) the opportunity to properly modulate the computation cost of the search process. The first goal is a prerequisite for a HW-NAS that should deal with ultra-low-power MCUs. The second goal is a prerequisite for a HW-NAS that should run on embedded systems.

In the adopted search space, a candidate solution stems from an established structure, which is organized as follows [8]:

- 1) a preprocessing pipeline performing min-max standardization and batch normalization on the input data;
- 2) a convolutional layer with  $k$  kernels;
- 3)  $c$  cells;
- 4) a classifier with a global average pooling layer followed by a dropout layer, which feeds a final layer having Softmax activation and a number of neurons equal to the number of classes.

Here, the cell is a sequence of three layers: a 2-Dmax pooling, a batch normalization layer, and a convolutional layer. Every max pooling layer halves the input resolution using a  $2 \times 2$  receptive field with (2,2) stride. Convolutional layers use (3,3) kernels with (1,1) stride and rectified linear unit (ReLU) activation. The number of kernels  $n_c$  used in the  $c$ th cell depends on  $k$  according to the following rule:

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

Thus, in the adopted search space admissible solutions can be described by two parameters:  $k$  defines the number of kernels used in the first convolutional layer, and  $c$  defines the number of cells.

## III. PROPOSED LIGHTWEIGHT HW-NAS

The lightweight HW-NAS inherits the overall design from [8]. Indeed, enhanced versions of both the optimization problem and the search strategy are adopted. The goal is to combine the capability of generating tiny CNNs that can run in real-time on sensing nodes with the ability of running the NAS itself on an embedded system that may play the role of a central unit in a sensor network.

### A. Optimization Problem

HW-NAS seeks the best neural architecture for the target hardware in a search space  $\mathcal{S}_S$ . This translates into a constrained optimization problem where the objective function describes the metric adopted to evaluate candidate solutions (i.e., CNNs), and the constraints set the boundaries of the search. In the proposed problem formulation, a candidate CNN is evaluated by training it and assessing its validation accuracy. The constraints for a candidate CNN are the RAM, flash memory, and multiply and accumulate (MAC) operations available on the target hardware that hosts the inference phase. The latter quantity is used as a valuable proxy for the latency [3]. In addition, a specific constraint limits the available memory for the training phase on the device hosting the search procedure.

This leads to the problem formulation  $P$

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

where  $x = (k, c)$ . Function  $f$  returns the validation accuracy. Parameters  $\xi_R$ ,  $\xi_F$ , and  $\xi_M$  represent, respectively, the upper bounds for RAM usage, flash usage, and MAC operations on the target hardware;  $\Theta_T$  sets the bounds on RAM usage for the search procedure. Thus, function  $\phi_R$  returns the CNN's RAM occupancy,  $\phi_F$  returns the CNN's Flash occupancy, and  $\phi_M$  returns the number of MAC operations required by the CNN, while  $\theta$  returns the RAM occupancy on the device hosting the NAS. These quantities depend on the number of kernels  $k$  used in the first convolutional layer, the number of cells  $c$ , the magnitude of the network's input size  $vi$ , and the adopted platform, in this case TF Lite Micro [16]. The search variable  $x = (k, c)$  does not include  $vi$ , which is a fixed quantity defined by the user.

### B. Search Strategy

The optimization problem  $P$  is solved with a derivative-free technique, as  $f(x)$  is not differentiable [8]. This procedure compares candidate networks by assessing their validation score  $f(x)$ ; so, in principle, a full training procedure of each candidate CNN is required. Training, though, is a demanding step; the computation cost of the NAS itself is roughly proportional to the number of training procedures to be completed for implementing the search strategy. Thus, an NAS that should run on resource-constrained devices needs a custom design.

Algorithm 1 formalizes the proposed search strategy, which relies on the following three factors.

- 1) The adopted search space inherently generates tiny CNNs as candidate solutions.
- 2) Early stopping is exploited to limit the computation cost of the training process (Section III-C will give further insight into this aspect).
- 3) A specific constraint is inserted to take into account the resource availability in the platform that runs the NAS.

The search starts with  $k = 1$  kernels in the first layer. Given  $k$ , the algorithm increments the number of cells  $c$  until the corresponding CNN( $k, c$ ) 1) satisfies the constraints imposed by parameters  $\xi_R$ ,  $\xi_F$ , and  $\xi_M$  and 2) scores a validation accuracy  $f(k, c)$  larger than that achieved with CNN( $k, c - 1$ ). Then, the same process is completed with  $k + 1$ . If the best CNN obtained with  $k + 1$  betters the best CNN obtained with  $k$ , the search continues by further incrementing  $k$ . Otherwise, the search stops.

When the training of a network CNN( $k, c$ ) cannot be completed due to limited resources available on the computing system running the NAS, the related accuracy  $f(k, c)$  is set to 0. Hence, the search strategy can also run on resource-constrained embedded systems.

**Algorithm 1:** Search Strategy.

```

k ← 1           ▷ Minimum number of kernels of the first layer
(c, best) = BestCNN(k)           ▷ call the procedure
repeat
    max = best
    k ← k + 1           ▷ increase number of kernels
    (c, best) = BestCNN(k)           ▷ call the procedure
until best > max
return (k, c), max
procedure BESTCNN(k)
    best = 0, Δ = 0
    c ← 0           ▷ no cells
    repeat
        train CNN(k, c)           ▷ train the new candidate
        f(k, c) = CNN(k, c)           ▷ validation accuracy
        Δ = f(k, c) - best           ▷ check if accuracy is increasing
        best = f(k, c)
        c ← c + 1           ▷ add a cell
    until CNN(k, c) satisfies (2) AND Δ > 0
    return c, best

```

**C. Early Stop Strategy**

CNNs are trained using gradient descent techniques, which require applying the backpropagation algorithm multiple times. This letter focuses on tiny networks featuring a small set of parameters. Hence, the representation capability is limited, and one cannot exploit standard approaches to speed up the convergence of the training. It is well known, though, that early stopping may play the role of a regularization technique, as, in general, the loss function dramatically decreases in the first few epochs of training. Thus, one can rely on a coarse version of early stopping to find a tradeoff between the computational cost of the search strategy and the ability to find an effective setting for the pair  $(k, c)$ . Such a solution is viable in that the actual goal is not to get a reliable estimation of  $f$  itself, but only to adopt a good criterion for deciding if a CNN is more promising than the others.

Then, considering a full training with  $n_{ep}$  epochs, if the estimation is performed after  $n'_{ep}$  epochs, the speedup is roughly  $n_{ep}/n'_{ep}$ . Interestingly, this solution impacts only the time (and energy) without effects on the memory occupation. Obviously, when  $n_{ep} \gg n'_{ep}$ , the speedup is higher, but the risk of wrong selections grows coherently.

Thus, the proposed HW-NAS utilizes early stopping to reduce the computation cost of the search procedure and at the same time to tackle issues, such as overfitting and local minima. Section IV-C will show empirically that one can even stop training after very few epochs without hindering the effectiveness of the NAS.

**IV. EXPERIMENTS**

The experiments involved three datasets: VWW [17], Cifar-10 (C-10) [18], and melanoma skin cancer [19]. VWW and C-10 are standard benchmarks for Tiny Visual ML. The target platforms belong to the STM32 ultra-low-power line: L010RBT6 (from now on L0), L151UCY6DTR (L1), and L412KBU3 (L4). The constraints in (2) were set according to Table 1; the MAC upper bound was obtained by rescaling the CoreMark score of a  $10^4$  factor. Early stopping ( $n'_{ep} = 3$ ) was adopted in the search strategy. The CNN selected by the NAS was then trained for 100 epochs with a batch size of 128 and a learning rate of  $10^{-2}$  using the Adam [20] optimizer. The validation set collected 10% of the original training set. Generalization performance was measured on public test sets.

Table 2. Performance of the Proposed HW-NAS on Target MCUs

MCU	NAS		Tiny CNN			Performance	
	Arch. (k,c)	Cost [hh]:[mm]	RAM [kiB]	Flash [kiB]	MAC [MM]	Acc. [%]	Lat. [mS]
Visual Wake Words ( $v_i = 50 \times 50$ )							
L0	(3,3)	1:39	20	10.7	0.41	71.7	56.2
L1	(4,5)	2:39	24	21.1	0.66	74.1	62
L4	(6,4)	3:17	28.5	23.7	1.27	77	87.9
Cifar-10 ( $v_i = 32 \times 32$ )							
L0	(6,4)	1:07	15.5	23.9	0.54	64.8	40.1
L1	(8,4)	1:15	16.5	28.8	0.7	68.5	50.3
L4	(14,4)	1:46	21	53.26	1.55	72.6	119.3
Melanoma Skin Cancer ( $v_i = 50 \times 50$ )							
L0	(1,3)	0:07	19.5	8.6	0.09	86.9	29.7
L1	(4,4)	0:14	23	16.26	0.65	88.8	61.9
L4	(6,4)	0:20	28.5	23.65	1.27	91.8	87.9

Table 3. Comparison With Existing HW-NAS on VWW Dataset

Model	Acc [%]	RAM [kiB]	Flash [kiB]	MACC [MM]	Serch time [hh]:[mm]
Proposal	77	28.5	23.7	1.3	3:17 CPU / 1:08 GPU
ColabNAS [7]	77.6	31.5	20.83	2	7:09 CPU / 2:28 GPU
Micronets [3]	76.8	70.5	273.8	3.3	n.a GPU
MCUNet [4]	87.4	168.5	530.5	6	> 300:00 GPU

We measured runtime RAM and flash using X-CUBE-AI 8.1.0 with Tensorflow Lite Micro. Publicly available models of state-of-the-art networks were downloaded in the TF Lite Micro format. The HW-NAS was executed on a laptop with an 11th Gen IntelCore™i7-11370H CPU @ 3.30 GHz, 16 GB of RAM, and 512 GB of SSD.

**A. Evaluation of the Hardware-Aware Feature**

The first experiment wanted to prove that the proposed HW-NAS can adapt the generated CNNs to the target hardware, even in the case of ultra-low-power MCUs. Table 2 presents the outcomes of this experiment. The columns of the Table are divided into four sectors. The first sector refers to the MCUs. The second sector gives the pair  $(k, c)$  characterizing the selected CNN and the cost of the NAS search procedure, expressed in time. The third sector shows the features of the selected CNN: RAM occupancy, flash occupancy, and MAC operations. The last sector gives the performance of the CNN in terms of test accuracy and latency (inference time per image). Latency was assessed using STM32Cube.AI runtime on L4 in balanced mode.

The proposed HW-NAS adapted the architecture of the generated CNNs to each platform. The search cost raised as the hardware resources of the target device increased because of the broader search space. The search costs in Table 2 indeed represent a worst case, as they were obtained without involving any GPU. The experiment involving the L4 MCU as a target on the VWW was repeated by executing the NAS on a Raspberry Pi 4 with 4 GB of RAM; the search cost was 34:29 [hh]:[mm]. This result confirmed that the proposed HW-NAS can also run successfully on resource-constrained devices.

As expected, the larger CNNs have been generated when the L4 MCU was selected as target. The generalization performance of the selected CNNs again scales with their size. Overall, the proposed method can cope with very hard constraints by balancing the generalization performance with the available hardware resources.

**B. State-of-the-Art Comparison**

Table 3 presents a comparison on the VWW dataset with three state-of-the-art HW-NAS for MCUs: MCUNet [4], Micronets [3], and ColabNAS [7]. The first two works focus on high-performance MCUs, whereas Garavagno et al. [7] targeted ultra-low-power MCUs. Thus, the comparison involves—for MCUNet and Micronets—the smallest CNN among those presented in the respective works. In the case of MCUNet the target device was a STM32F412 [4], while for Micronets



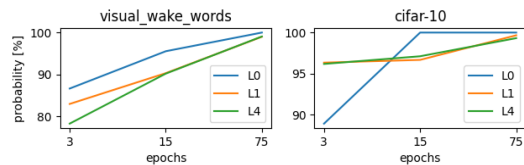


Fig. 1. Probability of selecting the best CNN based on the epochs.

and ColabNAS, the target was a STM32F446RE [3]. For the proposed HW-NAS the table reports the CNN selected when L4 was the target.

The table tabulates that the CNN selected by the proposed HW-NAS achieved on the VWW dataset the same accuracy scored by the CNNs selected, respectively, by Micronets [3] and ColabNAS [7]. However, the last two CNNs require more RAM, more flash memory, and more MAC. In fact, the CNN selected by MCUnet [4] achieved the highest accuracy, but again, this CNN cannot be hosted on a ultra-low-power MCU, such as the L4. It is worth noting that Lin et al. [4] showed that a well-known lightweight CNN for image classification tasks, MobileNetV2, requires at least 256 kB of RAM to achieve a satisfactory accuracy on VWW. In fact, this amount of RAM is not available on ultra-low-power MCUs.

The proposed HW-NAS resulted the best in term of search cost. Such outcome confirms the ability to select effective architectures with a lightweight search strategy.

### C. Early Stopping: Analysis

The early stopping criterion is the key factor behind the algorithm's low search cost. Such an approach, though, may affect the ability to find an effective CNN. Fig. 1 shows the probability of selecting the very same CNN singled out using 100 epochs when applying the early stopping criterion with, respectively, 3, 15, and 75 epochs. A training with 100 epochs was used as reference because such setup almost always ensures the convergence to a stable local minima [7]. The plot in Fig. 1 gives on the x-axis the number of epochs and on the y-axis the probability, which has been assessed empirically.

The experiment confirms that even using a number of epochs as small as 3, the proposed HW-NAS selects with high probability (> 78%) the most promising candidate, i.e., the CNN that would have been selected with 100 epochs. This is a major result if one thinks that the search time scales linearly with the number of epochs, providing a theoretically estimated speed-up of 100/3. However, model initialization and the dataset loading procedure, which must be repeated before every training, strongly degrades such speed-up even more in the presence of large datasets, as in this case.

## V. CONCLUSION

Many applications can profit from the integration of deep learning into sensing nodes; however, design and deployment require domain-specific skills and specialized computing resources. We proposed an HW-NAS for ultra-low-power MCUs supported by a lightweight optimization procedure. This NAS proved able to fit the constraints of different target devices while providing state-of-the-art CNNs on the VWW dataset. In addition, the search procedure can run in a few hours without requiring GPUs. Experiments indeed showed that the HW-NAS can even run on a Raspberry Pi 4.

A few aspects need further investigation. First, large datasets may represent an issue, in particular, if the search procedure should run on

an embedded system. Second, as the proposed HW-NAS is designed for tiny CNNs, it may not achieve state-of-the-art results when the target platforms belong to high-performance MCUs. In this case, traditional NAS or handcrafted architectures may provide valuable solutions in terms of generalization performance.

## ACKNOWLEDGMENT

Project funded under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.1 - Call for tender No. 1049 published on Sept 14, 2022 by the Italian Ministry of University and Research (MUR) funded by the European Union - NextGenerationEU. Project Title "LEARN - muLtimodal Edge computing-bAased weaRable exoskeletons for assistance in daily life" - CUP D53D23016200001- Grant Assignment Decree No. 1181 adopted on July 27, 2023 by MUR.

## REFERENCES

- [1] E. Ragusa, S. Dosen, R. Zunino, and P. Gastaldo, "Affordance segmentation using tiny networks for sensing systems in wearable robotic devices," *IEEE Sensors J.*, vol. 23, no. 19, pp. 23916–23926, Oct. 2023.
- [2] J. S. Gidon, J. Borah, S. Sahoo, S. Majumdar, and M. Fujita, "Bi-directional LSTM model for accurate and real-time landslide detection: A case study in Mawiongim, Meghalaya, India," *IEEE Internet Things J.*, vol. 11, no. 3, pp. 3792–3800, Feb. 2024.
- [3] C. Banbury et al., "MicroNets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," in *Proc. Mach. Learn. Syst.*, 2021, pp. 517–532.
- [4] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUnet: Tiny deep learning on IoT devices," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2020, pp. 11711–11722.
- [5] E. Liberis, Ł. Dudziak, and N. D. Lane, "μNAS: Constrained neural architecture search for microcontrollers," in *Proc. 1st Workshop Mach. Learn. Syst.*, 2021, pp. 70–79.
- [6] J. Borah et al., "AiCareBreath: IoT enabled location invariant novel unified model for predicting air pollutants to avoid related respiratory disease," *IEEE Internet Things J.*, vol. 11, no. 8, pp. 14625–14633, Apr. 2024.
- [7] A. M. Garavagno, D. Leonardi, and A. Frisoli, "ColabNAS: Obtaining lightweight task-specific convolutional neural networks following Occam's razor," *Future Gener. Comput. Syst.*, vol. 152, pp. 152–159, 2024.
- [8] A. M. Garavagno, E. Ragusa, A. Frisoli, and P. Gastaldo, "A hardware-aware neural architecture search algorithm targeting low-end microcontrollers," in *Proc. 18th Conf. Ph.D. Res. Microelectronics Electron.*, 2023, pp. 281–284.
- [9] A. M. Garavagno, E. Ragusa, A. Frisoli, and P. Gastaldo, "Running hardware-aware neural architecture search on embedded devices under 512MB of RAM," in *Proc. IEEE Int. Conf. Consum. Electron.*, 2024, pp. 1–2.
- [10] A. Howard et al., "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1314–1324.
- [11] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 116–131.
- [12] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.
- [13] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2815–2823.
- [14] B. Wu et al., "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 10726–10734.
- [15] "32-bit arm cortex MCUS," Accessed: Feb. 2, 2024. [Online]. Available: <https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>
- [16] R. David et al., "Tensorflow lite micro: Embedded machine learning for tinyml systems," in *Proc. Mach. Learn. Syst.*, 2021, pp. 800–811.
- [17] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual wake words dataset," 2019, *arXiv:1906.05721*.
- [18] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: <http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf>
- [19] M. H. Javid, "Melanoma skin cancer dataset of 10000 images," 2022. Accessed: Feb. 2, 2024. [Online]. Available: <https://www.kaggle.com/dsv/3376422>
- [20] D. P. Kingma and J. Ba, "ADAM: A method for stochastic optimization," 2014, *arXiv:1412.6980*.