Università
di Genova

PhD Program in Science and Technology for Electronic and Telecommunication Engineering

Searching Neural Architectures at the Edge

Andrea Mattia Garavagno

Thesis submitted for the degree of *Doctor of Philosophy* (XXXVIII cycle)

26th February 2026

Prof. Paolo Gastaldo	Tutor
Prof. Antonio Frisoli	Tutor
Prof. Maurizio Valle	Head of the PhD program

Università di Genova | **DITEN** DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA, ELETTRONICA E DELLE TELECOMUNICAZIONI

Searching Neural Architectures at the Edge

Andrea Mattia Garavagno

⋈

Committee:

Reviewers

Prof. Luca De Marchi, University of Bologna, Italy
Dr. Sergio Decherchi, Italian Institute of Technology, Italy

Tutors

Prof. Paolo Gastaldo, University of Genoa, Italy
Prof. Antonio Frisoli, Scuola Superiore Sant'Anna, Italy

University of Genoa

Andrea Mattia Garavagno

Searching Neural Architectures at the Edge

xix+150 p.

To Mom, Dad, and Tigro, my family.

Abstract

The present dissertation introduces a new approach towards the design of neural networks, in which an edge device automatically designs a neural architecture based on the locally collected data. The proposed approach enables a new paradigm of on-device, privacy-preserving neural network design, as sensitive data no longer needs to leave the local network.

The proposed approach has been developed thanks to three key contributions. The first contribution, ColabNAS, established the feasibility of low-cost Hardware-Aware Neural Architecture Search (HW-NAS) by utilizing free online GPU services. The second, NanoNAS, eliminated the GPU dependency, enabling the search to run on standard CPUs and single board computers by incorporating self-awareness of the search platform's memory. The culminating contribution, GatewayNAS, achieves fully autonomous HW-NAS on an embedded IoT gateway. This framework introduces a dual-level hardware awareness, adapting the search process to the constraints of both the target sensor node and the gateway's own time and energy budget. The outcomes of a thorough experimental session confirm that on the Visual Wake Words dataset, a TinyML benchmark, the proposed approach can achieve state-of-the-art results by exploiting a search procedure that runs in less than 10 hours on the Raspberry Pi Zero 2, a $15 edge device.

In conclusion, this work democratizes access to automatic neural network design, enables privacy-critical applications in sectors like healthcare and industry, and promotes a more sustainable, low-energy approach to AI development.

Keywords: Privacy-Preserving AI, Neural Architecture Search, Hardware-Aware, Edge AI, TinyML, Internet of Things, IoT, Industrial Internet of Things, Healthcare Internet of Things, Artificial Intelligence, Machine Learning.

Contents

Appendix

List of Figures

List of Tables

Notations

Notation	Definition
Core Architecture Parameters	
A or $A(k, c)$	A candidate neural network (NN) architecture.
k	The number of kernels in the first (or base) convolutional layer.
c	The number of cells (or building cells) stacked in the architecture.
x	The search variable, typically represented as the tuple (k, c).
n_c	The number of kernels used in the c-th cell.
vi	The input size (e.g., 50x50x3) of the neural network.
Performance / Objective Functions	
$f(x)$ or $\Gamma(A)$	The objective function to be maximized, typically representing the maximum validation accuracy of an architecture.
Edge Device (Target) Constraints	
$\phi_R(x)$ or $\phi_{RAM}(A)$	A function returning the peak RAM occupancy of architecture A on the target device (e.g., MCU).
$\phi_F(x)$ or $\phi_{Flash}(A)$	A function returning the Flash memory occupancy of architecture A on the target device.
$\phi_M(x)$ or $\phi_{MAC}(A)$	A function returning the number of Multiply-Accumulate (MACC) operations for architecture A.
ξ_R or ξ_{RAM}	The upper bound (limit) for peak RAM occupancy on the target device.
ξ_F or ξ_{Flash}	The upper bound (limit) for Flash memory occupancy on the target device.
ξ_M or ξ_{MAC}	The upper bound (limit) for MACC operations on the target device.

Notation	Definition
Gateway Device (Host) Constraints	
$\theta(x)$ or $\phi_{MEM}(A)$	A function returning the peak RAM occupancy required to *train* architecture A on the host (gateway) device.
$\phi_{Time}(S_\alpha)$	A function estimating the total execution time to run the search over the search space S_α.
$\phi_{Energy}(S_\alpha)$	A function estimating the total energy consumed to run the search over the search space S_α.
Θ_T or ξ_{MEM}	The upper bound (limit) for peak RAM occupancy on the host (gateway) device.
ξ_{Time}	The upper bound (limit) for the total search execution time.
ξ_{Energy}	The upper bound (limit) for the total search energy budget.
Search Space & Strategy	
Ω	The set of all valid (feasible) solutions; the feasible search space.
$\hat{S}_\alpha$	The extensive, uncropped search space, containing all architectures feasible under both edge and gateway memory constraints.
S_α	The final, operational search space, cropped from $\hat{S}_\alpha$ to respect time and energy budgets.
n_{ep}	The total number of epochs for a full training cycle.
n'_{ep}	The number of epochs used for a partial/early-stopping evaluation during the search.
$\bar{t}$	An upper bound for the time required to evaluate a single candidate architecture.
$\bar{e}$	An upper bound for the energy consumed to evaluate a single candidate architecture.
$\bar{w}$	The measured maximum power consumption of the gateway device.
β, γ	Internal parameters of the GatewayNAS search strategy (Algorithm 5.3) controlling the increment of k.
c^*	The optimal number of cells c found for a given, fixed k.
$\bar{k}$	A candidate value for k being explored during the search.

Acronyms

AI	Artificial Intelligence
AS	Affordance Segmentation
BO	Bayesian Optimisation
C-10	CIFAR-10 (Dataset)
CNN	Convolutional Neural Network
COCO	Common Objects in Context (Dataset)
CPU	Central Processing Unit
CWRU	Case Western Reserve University (Dataset)
DNAS	Differentiable Neural Architecture Search
EA	Evolutionary Aging / Evolutionary Algorithms
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
HIoT	Healthcare Internet of Things
HMI	Human-Machine Interface
HW-NAS	Hardware-Aware Neural Architecture Search
ICCE	International Conference on Consumer Electronics
IFD	Intelligent Fault Diagnosis
IIoT	Industrial Internet of Things
IoT	Internet of Things
L0	L010RBT6 (STM32 MCU)
L1	L151UCY6DTR (STM32 MCU)
L4	L412KBU3 (STM32 MCU)
MAC	Multiply-And-Accumulate
MCTS	Monte Carlo Tree Search
MCU	Microcontroller Unit / Microcontroller
ML	Machine Learning
NAS	Neural Architecture Search
NN	Neural Network
NRRP	National Recovery and Resilience Plan
OS	Operating System
PTQ	Post-Training Quantization
QAT	Quantization Aware Training

RAM	Random-Access Memory
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RPi 3	Raspberry Pi 3 Model B
RPi 4	Raspberry Pi 4 Model B
RPiZ 2	Raspberry Pi Zero 2
SAR	Synthetic Aperture Radar
SEAS	Harvard John A. Paulson School of Engineering and Applied Sciences
SoC	System-on-a-Chip
STM	STMicroelectronics
TL	Transfer Learning
VWW	Visual Wake Words (Dataset)

CHAPTER 1

Introduction

1.1 The Proliferation of Edge Intelligence

The last decade has witnessed a paradigm shift in computing, driven by the explosive growth of the Internet of Things (IoT) [2]. Billions of interconnected devices, from smart wearable systems to distributed industrial sensors, are now embedded in our daily environments, generating unprecedented volumes of data at the network's edge [3]. Processing this data locally, rather than relying on centralized cloud infrastructure, offers significant advantages in terms of latency, bandwidth, and, critically, data privacy. This shift has given rise to the field of Edge AI, where the goal is to imbue these resource-constrained devices with intelligent decision-making capabilities [4].

Convolutional Neural Networks (CNNs) have emerged as the cornerstone of modern AI, particularly for teleceptive sensing tasks involving computer vision [5]. Their ability to extract hierarchical features from raw sensor data has enabled breakthroughs in applications ranging from real-time object detection in autonomous systems to medical diagnostics in the Healthcare Internet of Things (HIoT) [6] and intelligent fault diagnosis in the Industrial Internet of Things (IIoT) [7]. However, a fundamental tension exists between the high computational and memory demands of deep neural networks and the stringent resource limitations of the hardware at the edge, typically low-power microcontrollers (MCUs) with only kilobytes of RAM and Flash memory. This mismatch presents a significant barrier to the widespread deployment of advanced AI on the vast majority of edge devices.

1.2 Hardware-Aware Neural Architecture Search

To bridge the gap between deep learning models and resource-constrained hardware, the field of Neural Architecture Search (NAS) has been instrumental [8]. NAS automates the laborious and intuition-driven process of designing neural network architectures. A

specialized subfield, Hardware-Aware Neural Architecture Search (HW-NAS), has become particularly relevant for edge applications. HW-NAS explicitly incorporates the hardware constraints of the target device, such as memory footprint, latency, and power consumption, directly into the architecture optimization process. This allows for the automatic generation of models that are not only accurate but also efficient and deployable on specific edge platforms.

Despite its promise, the practical application of HW-NAS has been severely limited by its own prohibitive resource requirements. Seminal works in the field required enormous computational budgets, with search procedures consuming tens of thousands of GPU hours [9]. While subsequent research has made significant strides in reducing this cost to hundreds of GPU hours [10], the process remains largely inaccessible to researchers, small enterprises, and developers without access to high-performance computing clusters. This high barrier to entry has hindered the democratization of efficient AI model design. Moreover, the reliance on powerful, often cloud-based, computing resources perpetuates the very privacy concerns that Edge AI aims to solve, as sensitive, domain-specific data must still be transmitted to a third party for the model design phase.

1.3 Thesis and Contributions

This dissertation posits that through the development of lightweight search spaces and novel, low-cost optimization strategies, it is possible to create HW-NAS frameworks that are not only computationally affordable but can be executed directly on resource-constrained edge hardware. This approach enables a new paradigm of fully on-device, privacy-preserving neural network design, where models are automatically tailored to the specific hardware and local data of an application, without any reliance on external cloud services.

The research presented herein follows a progressive path toward this goal, systematically reducing the computational footprint of HW-NAS until it can operate within the strict confines of an IoT gateway. The principal contributions are:

ColabNAS: The first contribution establishes the feasibility of low-cost HW-NAS. By employing a novel search strategy inspired by the principle of Occam's razor, it demonstrates that state-of-the-art, task-specific CNNs can be generated in a few hours using freely available online GPU services. This work significantly lowers

the barrier to entry for HW-NAS and proves that effective architecture design does not require massive computational resources.

NanoNAS: Building upon this foundation, the second contribution removes the dependency on GPUs entirely. It shows that for designing highly compact CNNs for ultra-low-power microcontrollers, the search procedure can be executed efficiently on a standard laptop CPU. This work introduces the concept of a self-aware search platform by incorporating constraints related to the memory of the device performing the search, moving HW-NAS from the cloud to local, general-purpose computers.

GatewayNAS: The final and culminating contribution realizes the ultimate goal of executing HW-NAS on an embedded IoT gateway. This framework introduces a dual-level hardware awareness, considering the constraints of both the target sensor node for inference and the gateway for the search process itself. A key innovation is an adaptive mechanism that tailors the search space to the gateway's available time and energy budget, enabling a complete, privacy-preserving workflow where local sensor data is used to generate custom, hardware-optimized models without ever leaving the premises.

Additionally, a novel automated pipeline for generating large-scale and high-quality binary classification datasets for Tiny Machine Learning (TinyML) is presented. It tackles the current limitations in size and quality of existing datasets and benchmarks for TinyML hindering the design of production-grade systems for Edge AI and therefore the adoption of HW-NAS. Intelligent multi-source label fusion, confidence-aware filtering, automated label correction, and systematic fine-grained benchmark generation are applied to existing ML datasets to generate datasets specifically designed for TinyML.

1.4 Dissertation Outline

The remainder of this dissertation is structured to detail this progressive journey toward on-device HW-NAS.

Chapter 2 provides a survey of previous work in the fields of Neural Architecture Search and its hardware-aware counterpart. It traces the evolution of search strategies, discusses key benchmarks, and highlights the existing limitations, particularly the high computational cost, that motivated this research.

Chapter 3 introduces ColabNAS, a lightweight HW-NAS framework designed to run on free online GPU services. This chapter details its novel derivative-free search strategy and demonstrates its effectiveness in producing compact, task-specific CNNs that outperform models derived from conventional techniques like Transfer Learning.

Chapter 4 presents NanoNAS, the next step in reducing the computational burden of HW-NAS. This chapter describes the methodology that enables the search process to run efficiently on a CPU, eliminating the need for specialized hardware and making HW-NAS accessible on commodity laptops.

Chapter 5 details GatewayNAS, the first HW-NAS framework capable of running on a resource-constrained IoT gateway. It introduces the adaptive mechanism for managing time and energy budgets and extends the methodology to time-series data, demonstrating state-of-the-art performance in an industrial fault diagnosis task.

Chapter 6 concludes the dissertation by summarizing the key contributions and their collective impact on democratizing AI hardware design, enabling privacy-preserving AI, and promoting a more sustainable approach to AI development. The chapter also discusses the limitations of the presented work and proposes promising directions for future research.

Additionally, **Appendix A** introduces Wake Vision, a novel automated pipeline for generating large-scale and high-quality binary classification datasets for TinyML, tackling the current limitations of existing datasets and benchmarks.

CHAPTER 2

Previous work

This chapter surveys the previous work in NAS. The discussion is organized into three parts. The first section introduces NAS from its inception, highlighting the main search strategies, the benchmarks that have shaped the field, and the key challenges that remain unresolved. The second section narrows the scope to HW-NAS, analyzing the strategies developed to incorporate hardware constraints into the search process, the benchmarks used to evaluate them, and the limitations that still hinder widespread adoption. Finally, the third section outlines the contributions of this dissertation to the HW-NAS literature and provides a roadmap for the remainder of the document.

2.1 Neural Architecture Search

The concept of NAS was first defined by Zoph et al. in [11], where a Recurrent Neural Network (RNN) was employed to automatically design the architecture of a NN. Reinforcement Learning (RL) was adopted to train the RNN with the aim of maximizing the validation accuracy of the resulting NN on the target task. Notably, the paper introduces the concept of a search space, which makes use of domain-specific knowledge to restrict the set of allowed operators for the design of the NN's architecture depending on the task to solve. State-of-the-art results have been obtained on the image classification dataset CIFAR-10 [12] and on the language modeling dataset Penn Treebank [13], rivaling human-made design.

2.1.1 Search Strategies

After the first experience of NAS [11], several ways of automatically searching the best neural architecture to solve a given task have been proposed in the literature, introducing the concept of a search strategy, i.e., the algorithm adopted to find the best NN for a given task. Table 2.1 summarizes the main search strategies adopted in the literature. Figure

Table 2.1: Main search strategies adopted by previous work

Work	Search Strategy
Zoph et al. [11]	Reinforcement Learning (RL)
Real et al. [14]	Evolutionary Aging (EA)
Brock et al. [15]	Automatic Weight Generation + RL
Liu et al. [16]	Differentiable Neural Architecture Search (DNAS)
Kandasamy et al. [17]	Bayesian Optimisation (BO)
Pham et al. [18]	Super-Network + RL
Bender et al. [19]	Super-Network + DNAS
Wang et al. [20]	Monte Carlo Tree Search (MCTS)
Li et al. [21]	Random Search

2.1 visualizes the main concepts behind NAS, the search space, represented by a set of possible neural architectures, and the search strategy represented by a guy questioning how to find the best neural architecture inside the search space.

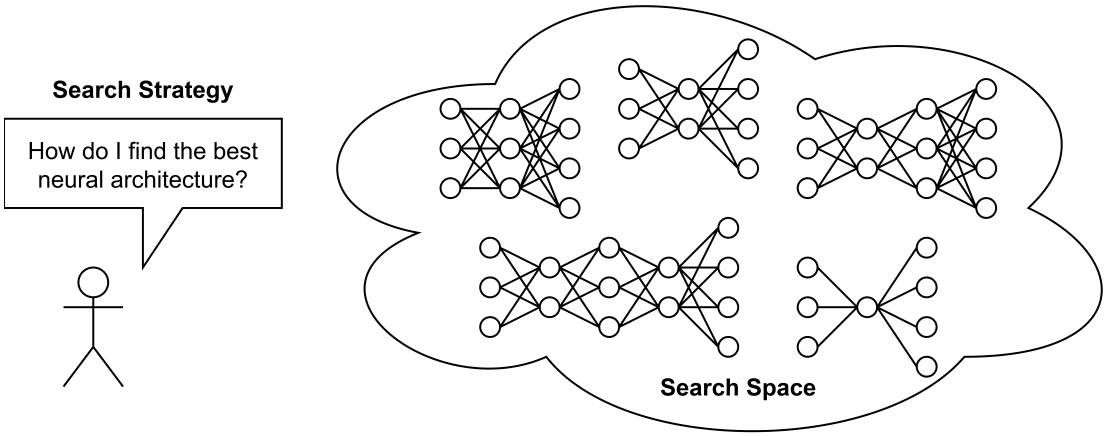


Figure 2.1: Visualization of the main concepts behind NAS: the search space, i.e., the set of possible neural architectures, and the search strategy, i.e., the method adopted to find the best neural architecture in the search space.

The first search strategy different from RL was proposed by Real et al. [14], who made use of Evolutionary Aging (EA) to automatically design Neural Networks (NNs). The concepts of mutation, selection, and reproduction were adapted to the NAS use case, leading to state-of-the-art models for the CIFAR-10 and CIFAR-100 [12] datasets through evolutionary mechanisms.

While NAS results were rivaling human-made designs, the high execution cost restricted the usage of such techniques to few and well-equipped entities. Trying to lower the search cost, Liu et al. [16] re-formulated the problem of finding the best NN inside the search space in a differentiable manner, allowing the use of gradient-based optimization methods for solving the NAS problem, introducing the term Differentiable Neural Architecture Search (DNAS). This innovation reduced the search cost while finding state-of-the-art NNs for CIFAR-10 [12], Imagenet [22], Penn Treebank [13], and WikiText-2 [23] datasets.

In the same year, Kandasamy et al. [17] proposed the usage of Bayesian Optimisation (BO) to solve the NAS problem in the classical non-differentiable formulation. The proposed Gaussian process showed broad applicability, obtaining state-of-the-art results on CIFAR-10 [12], blog feedback [24], indoor location [25], slice localization [26], naval propulsion [27], protein tertiary structure [28], and news popularity [29] datasets, while reducing the search cost when compared to previous methods.

The quest towards smaller search costs pushed Pham et al. [18] to introduce the concept of a super-network, a large NN containing all the candidate solutions inside it as sub-networks. The super-network required only one large training phase followed by a search phase for the best sub-network for solving the given task, done with RL. This allowed to reduce the search time at the cost of increasing hardware requirements for the training phase, which shifted from a multitude of small NNs to a single large NN. The proposed NAS obtained state-of-the-art results on the Penn Treebank [13] and on the CIFAR-10 [12] datasets.

A similar concept was pioneered by Brock et al. [15], who used a hyper-network to generate the weights of each candidate architecture explored by RL, requiring only the training of the hyper-network.

The same concept of a super-network has been adopted by Bender et al. [19] in conjunction with gradient-based optimization methods to reduce the search cost, introducing another flavor of DNAS. Generally, works training only one super-network and then searching for the best sub-network inside it fall under the umbrella of One-Shot NAS.

Another effort towards reducing the search cost led Wang et al. [20] to apply Monte Carlo Tree Search (MCTS) for finding the best NN for a given task, achieving state-of-the-art results on CIFAR-10 [12] and Imagenet [22] datasets.

2.1.2 Benchmarks

Still, the search cost was too high for democratizing the usage of NAS. Only a few private entities possessed the resources needed for running these algorithms, pushing out of the field the majority of researchers. For this reason, Ying et al. [30] introduced the concept of a NAS benchmark, a pre-trained search space that can be used by researchers to evaluate their search strategies without training any candidate architecture, opening this research topic to a broader audience. 423k unique convolutional architectures, trained multiple times on the CIFAR-10 [12], were proposed, producing over 5 million trained models available to researchers for evaluating their search strategies on a common ground. Later, the benchmark has been updated by Dong et al. [31], also including ImageNet-16-120 [32] and CIFAR-100 [12], besides the previous CIFAR-10 [12].

Criticizing the limited size of the search space included in the benchmark, Siems et al. [33] proposed the usage of surrogate models to estimate the final performance of a candidate architecture, avoiding the extensive training of the search space, thus allowing the usage of a larger search space.

2.1.3 Reproducibility issues

In one of the first reproducibility studies, Li et al. [21] showed the impact of the evaluation phase of the search strategy on the performance of the resulting NN, highlighting the need for listing all the technical details needed for allowing at least exact reproducibility of the experimental results presented in the papers, questioning how to evaluate broader reproducibility against different benchmarks given the high cost of running NAS.

Moreover, the same authors found that random search is a competitive baseline for NAS search strategies, and that a mix of random search and weight sharing obtains state-of-the-art NAS results on the Penn Treebank [13] dataset and a highly competitive result on CIFAR-10 [12], setting new standards for the research field of NAS.

2.2 Hardware-Aware Neural Architecture Search

Cai et al. [34] pioneered the concept of introducing hardware-defined metrics estimating the latency of the candidate into the NAS to obtain production-grade NNs ready to be deployed on different hardware, i.e., mobile, CPU, GPU, introducing for the first time the

Table 2.2: Main hardware-awareness strategies adopted by previous work

Work	Target	Hardware-Awareness	Methodology
Cai et al. [34]	Mobile	Latency	Modeled
Tan et al. [9]	Mobile	Latency	Measured for each model
Wu et al. [35]	Mobile	Latency	Measured for each operator
Cai et al. [36]	Mobile	Latency	Predicted using ML
Lin et al. [10]	IoT	Latency, Flash RAM	Computed at compile time
Banbury et al. [37]	IoT	Latency, Flash, RAM, Energy	Estimated through verified proxies

concept of HW-NAS. Table 2.2 summarizes the hardware-awareness strategies adopted by previous work.

Tan et al. [9] further developed the concept of hardware-awareness using direct measurements of latency in the search strategy. Specifically, latency measurements on real-world smartphone hardware have been incorporated into the search process to search for the NN with the best trade-offs between accuracy and latency. State-of-the-art results on the COCO [38] and on the ImageNet datasets [22] were achieved under established smartphone latency constraints using RL as a search strategy.

Instead, Wu et al. [35] proposed a novel gradient-based solution for the HW-NAS problem targeting mobile devices. Gradient descent has been used to find the best NN inside a pre-trained super-network, built evaluating each node on the target hardware, to gather knowledge about the latency. A latency-aware loss function has been used to include the latency information in the gradient descent. State-of-the-art results have been obtained on ImageNet [22] at an estimated search cost 420 times smaller than the previous work [9], for a total of 216 GPU-hours. Still, the whole procedure had to be repeated for each target hardware, which included the training of the super-network.

To overcome the latter problem, Cai et al. [36] decoupled the training phase of the super-network from the selection phase of the nodes inside it, to allow the specialization of the super-network after its training. Doing so, the super-network can be trained once and then specialized for several target hardware, just selecting the nodes that better fit the deployment needs. State-of-the-art results were obtained on ImageNet [22], surpassing the accuracy of hand-crafted mobile models like MobileNetv3 [39].

2.2.1 Towards TinyML

After the initial phase of HW-NAS, which mostly worked on obtaining production-grade NNs for mobile devices like smartphones, the focus shifted towards the domain of TinyML, i.e., electronic systems in the order of 100 kiB of RAM, which account for over 15 billion devices globally, roughly 4 times the number of mobile devices [40].

Lin et al. [10] introduced the constraints on the RAM and Flash available to the target hardware in the HW-NAS problem, focusing on Internet of Things (IoT) devices. Using One-Shot NAS, state-of-the-art results able to run on several resource-constrained hardware like STM32F412, STM32F746, STM32F765, and STM32H743 microcontrollers were provided for ImageNet [22], Visual Wake Words [41], and Speech Commands [42] datasets in 300 GPU hours. This is a reduction of 133× compared to the 40,000 GPU hours required by Tan et al. [9].

In the same framework of IoT, Banbury et al. [37] proposed a search space where the operation count of the candidate NN can be used as a valuable proxy towards model latency and energy consumption. This search space is explored using gradient descent. State-of-the-art NNs have been found for Visual Wake Words [41], Speech Commands [42], and MIMII [43] while targeting STMF446RE, STMF746ZG, STMF767ZI microcontrollers.

2.2.2 Benchmarks for TinyML

Even though benchmarks like CIFAR [12] and ImageNet [22] were broadly adopted by the NAS community, the TinyML community was lacking established benchmarks for guiding the development of the field [44]. In this scenario, Visual Wake Words [41], Speech Commands [42], CIFAR-10 [12], and ToyADMOS [45], which were already being partially adopted as benchmarks by the HW-NAS community, emerged as benchmarks from a joint effort between industry and academia, giving birth to the MLPerf Tiny, the first industry-standard benchmark suite for ultra-low-power tiny machine learning systems [46].

2.2.3 Limitations

Even though the search cost of HW-NAS has been reduced from 40,000 GPU hours to 300 GPU hours, factors such as high-end GPU accessibility, time availability, and the known difficulties in reproducing NAS results strongly limit the adoption of HW-NAS. In the end, it is more probable that a user adopts the resulting architectures listed in the paper rather than the HW-NAS itself.

Another element hindering the adoption of HW-NAS, and more in general of TinyML, is the lack of large and high-quality TinyML datasets which prevents the generation of high performance NNs for production-grade systems [44].

2.3 Main Contribution of This Work

The main contribution of this work is the introduction of a new generation of lightweight HW-NAS that can run at the edge. This is done with three key technical contributions:

The ideation of lightweight search spaces, specifically designed for ultra-low-power microcontrollers;

The design of custom derivative-free optimization methods to lower the search cost of the HW-NAS;

The introduction of hardware-awareness mechanisms regarding the platform executing the search procedure, which adapt the HW-NAS to the hardware features of the platform running it, enabling its execution on a variety of edge devices;

A lightweight HW-NAS able to run at the edge without any internet connection involved not only has the potential to increase its adoption, but also serves as an enabling technology in privacy sensitive applications. Once a user can run HW-NAS on their hardware, detached from the internet, they can safely design optimal NNs on their data, without sharing it with any third party, preventing the release of industrial secrets, biometric and healthcare data.

The remainder of the document is organized as follows. Chapter 3 presents ColabNAS, a lightweight HW-NAS designed to run on free-online GPU services, like Google Colab and Kaggle Kernel, democratizing for the first time the usage of HW-NAS. Instead, Chapter 4 introduces NanoNAS, a new iteration of lightweight HW-NAS that can run on laptops, without using a GPU. Finally, Chapter 5 introduces GatewayNAS, the first HW-NAS capable of running on IoT Gateways with as low as 512 MiB of RAM.

Additionally, Appendix A presents a novel automated pipeline for generating large-scale and high-quality binary classification datasets for TinyML, overtaking the limitations of current datasets and benchmarks.

CHAPTER 3

ColabNAS: Obtaining Lightweight Task-Specific Convolutional Neural Networks Using Free Online GPU Services

The contribution presented in this chapter has been previously published as a journal paper [47].

3.1 Introduction

The proliferation of next-generation smart wearable systems and distributed sensors is largely enabled by task-specific CNNs. The constrained scope of the problems these devices solve allows for the use of lightweight models. However, the development of an effective lightweight neural network from the ground up is a challenging endeavor. A specialized research field, known as HW-NAS, addresses this challenge by tailoring the architecture search to the specific resource constraints of the target hardware.

A significant body of work in recent years has focused on establishing standard benchmarks for tiny machine learning applications [44]. The Imagenet1k dataset [22] and the Visual Wake Words (VWW) dataset [41] have emerged as key benchmarks for evaluating deep learning techniques in computer vision on commodity hardware. Initial progress in this area was driven by manually designed lightweight architectures such as MobileNets [48, 49, 39], SqueezeNet [50], and ShuffleNets [51, 52]. Subsequently, research efforts shifted towards automating this design process using NAS techniques that incorporate hardware constraints like Flash and SRAM occupancy, giving rise to HW-NAS.

Several HW-NAS methods employ reinforcement learning to discover an optimal architecture within the given hardware constraints. In this paradigm, a controller generates candidate architectures and receives rewards based on their validation accuracy and hardware cost. Examples of this approach include MNASNet [9], FPNet [53], Codesign-NAS

[54], and the work by [55]. A common drawback of this strategy is the high computational cost, as each generated architecture typically requires a full training cycle. MNASNet, for instance, reported a search cost of 40,000 GPU hours to find a Pareto optimal solution by penalizing an objective function with hardware constraints.

To mitigate such high search costs, the concept of a supernetwork was introduced. A supernetwork is an over-parameterized model that contains all possible architectures within the search space as sub-networks sharing weights. This approach requires training only the single supernetwork, thereby significantly reducing the search time. Once the supernetwork is trained, techniques such as evolutionary algorithms [10, 36, 56] or gradient descent methods [35, 16, 37, 57, 58, 34] are used to identify the best-performing sub-network. Specifically, MCUNet comprises a two-part system: TinyNAS, an HW-NAS technique, and TinyEngine, a framework for executing deep learning on microcontrollers. Its search space is constructed with hardware constraints in mind, measuring Flash, SRAM, and latency directly on the target device. In contrast, Micronets targets microcontrollers by avoiding hardware-in-the-loop validation, instead calculating memory occupancy and using operation count as a proxy for latency. Although the search time is reduced, training a supernetwork still demands substantial computational resources.

Despite the advancements in HW-NAS, a prevalent method for designing lightweight task-specific CNNs remains Transfer Learning (TL) [59]. For instance, a 2022 study by [60] proposed a lightweight CNN for diagnosing COVID-19 from chest X-rays, using MobileNetV2 [49] as the model's backbone through a TL procedure. A similar methodology was applied by [61] to detect ships in synthetic aperture radar (SAR) images, employing MobileNetV2 as the backbone for a YOLOv4-LITE model. In the same year, TL was also leveraged by [62] to develop an Efficientnet-B1 [63] based model for detecting arc faults in photovoltaic systems, by [64] for a MobileNetV2 based model to count wheat heads, and by [65] for an EfficientNet-B0 [63] based model for early oral cancer detection. Furthermore, Ragusa et al. utilized MobileNets as a backbone for image polarity detection [66] and affordance detection [67] on embedded devices.

This overview highlights the extensive use of TL for designing lightweight networks. TL leverages knowledge from a pre-trained CNN to solve a new, similar problem, thus reducing the required training data and time. This is particularly useful when data or time for designing a model from scratch is limited. However, TL can be excessive for certain applications. Task-specific problems can often be addressed by significantly lighter CNNs. The models used in TL are typically trained on large-scale datasets like ImageNet1k [22], which may be unnecessarily complex for narrow applications where sufficient end-user

data is already available. In such scenarios, generating a CNN from scratch is a more direct approach. Yet, the high search cost of HW-NAS or the expense of manual design often outweighs the time savings of TL, which may explain its continued prevalence.

This chapter introduces ColabNAS, an affordable HW-NAS technique for producing lightweight, task-specific CNNs. It employs a novel derivative-free search strategy, inspired by Occam's razor, which achieves state-of-the-art results on the Visual Wake Word dataset [41] in only 3.1 GPU hours using free online GPU services like Google Colaboratory and Kaggle Kernel. The name ColabNAS emphasizes its ability to run on these free subscription services, making the technology accessible to a broader audience. This accessibility is critical for fostering the adoption of CNNs across a variety of application tasks, especially in the domain of embedded wearable and distributed devices, and for enabling a diverse population of end-users and researchers to implement them.

This chapter presents the details of the proposed technique and its experimental validation on five different classification problems using publicly available datasets. The results obtained on the Visual Wake Word dataset are compared with state-of-the-art HW-NAS methods. Online access to experimental data and code is provided as Google Colaboratory notebooks at this link. The code is also available on GitHub at the following link.

3.2 Methodology

This section details the proposed methodology for searching lightweight, task-specific CNNs. Any HW-NAS technique is characterized by three key components: the search space, the formulation of the corresponding optimization problem, and the strategy for searching for a solution.

3.2.1 Search Space and Problem Formulation

ColabNAS utilizes a cell-wise search space, which is constructed starting from a single two-dimensional convolutional layer. Subsequently, pairs of pooling and convolutional layers are stacked, forming a single cell, in a manner similar to the VGG16 architecture [68]. This process, illustrated in Figure 3.1, continues as long as the network's generalization capability improves.

The stopping criterion is inspired by the problem-solving principle known as Occam's razor, attributed to the English Franciscan friar William of Ockham, which states that "entities should not be multiplied beyond necessity".

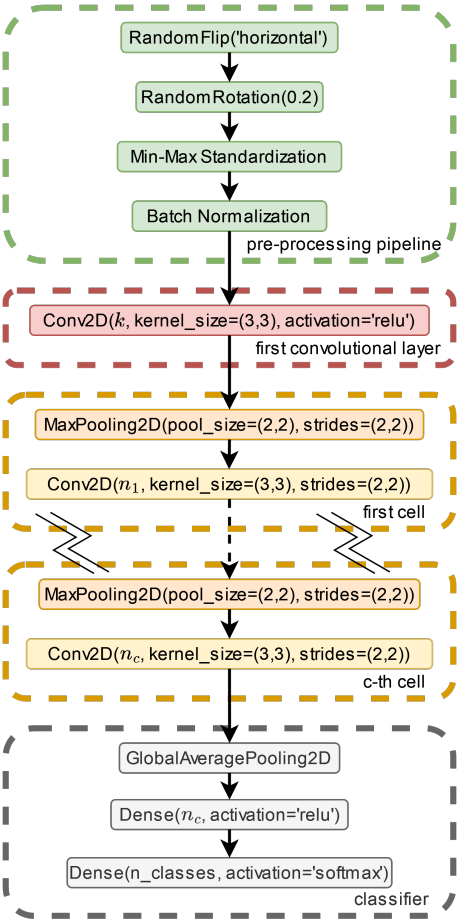


Figure 3.1: Detailed graphical representation of the generic network architecture. n_c represents the number of kernels used in the c-th cell. The dotted connection represents a generic number of added cells. The deep dense layer has a number of neurons equal to the kernels used in the last added cell.

The number of kernels for each added cell is determined by Equation 3.1, where k is the number of kernels in the first convolutional layer. This formulation is inspired by the VGG16 architecture [68], where each cell doubles the number of kernels relative to the previous one. In the proposed approach, however, this amplification is modulated at each cell to limit the growth of parameters.

$$n_c = \begin{cases} k & \text{if} \quad c = 0 \\ \left\lceil (2 - \sum_{i=1}^{c-1} 2^{-i}) \cdot n_{c-1} \right\rceil & \text{if} \quad c \geq 1 \end{cases} \tag{3.1}$$

The search space is constrained by the network's peak RAM occupancy, Flash occupancy, and the number of multiply-accumulate operations (MACCs), which serves as a rough estimate of latency as suggested by [37].

$$P1 : \begin{cases} \max f(x) \\ \phi_R(x) \leq \xi_R \\ \phi_F(x) \leq \xi_F \\ \phi_M(x) \leq \xi_M \\ \xi_R, \xi_F, \xi_M > 0 \end{cases} \tag{3.2}$$

The resulting optimization problem is formulated in Equation 3.2, where the function f returns the maximum validation accuracy achieved during training. The function ϕ_R gives the network's peak RAM occupancy, ϕ_F provides the network's Flash occupancy, and ϕ_M calculates the network's MACC count. These functions depend on the number of kernels in the first layer, k, the number of added cells, c, and the network's input size, s. Since the input size is considered fixed during the search, it is omitted from the formulation, defining the search variable as $x = (k, c)$. The values of $f(x)$ are obtained from training. The set of all valid solutions will be referred to as the feasible search space Ω.

3.2.2 Network Architecture Details

Inspired by VGG16 [68], convolutional layers with 3x3 kernels and zero padding were adopted to preserve the input size, along with pooling layers using a 2x2 receptive field and a (2, 2) stride.

The output of the convolutional base is reduced using a 2D Global Average Pooling operator to enhance the model's generalization capability [69]. Following this, a deep fully connected layer, with a neuron count equal to the number of kernels in the last convolutional layer, processes the reduced features. Finally, a single fully connected layer performs the classification of the extracted features.

A pre-processing pipeline is integrated into the network's architecture. It first applies min-max standardization to improve the convergence rate of gradient descent [70]. Next, it employs batch normalization to stabilize and accelerate training [71]. Data augmentation, including both horizontal flips and random rotations, is also applied. Figure 3.1 provides a graphical representation of the generic architecture that defines the search space.

3.2.3 Search Strategy

The search strategy explores the search space in a two-step process. First, for a given initial number of kernels in the first layer, it sequentially adds cells as long as the network's generalization capability improves, consistent with Occam's razor, or until hardware constraints are violated. Second, this process is repeated by adjusting the number of kernels in the first layer according to Equation 3.3.

$$k_j = \begin{cases} k_0 & \text{if} \quad j = 0 \\ 2 \cdot k_0 & \text{if} \quad j = 1 \\ 2 \cdot k_{j-1} & \text{if} \quad f(k_1^*, c_1^*) > f(k_0^*, c_0^*) \\ \frac{1}{2} \cdot k_{j-1} & \text{if} \quad f(k_1^*, c_1^*) \leq f(k_0^*, c_0^*) \end{cases} \tag{3.3}$$

If the selected hardware allows, the procedure is repeated with double the number of kernels. If the solution found in the second iteration, (k_1^*, c_1^*), surpasses the previous one, (k_0^*, c_0^*), the process continues in this manner until performance no longer improves or hardware limits are met. Otherwise, the process proceeds by halving the initial number of kernels until a performance degradation is observed. This approach again adheres to Occam's razor, as entities are not multiplied beyond necessity.

This search strategy can be viewed as a custom derivative-free method for solving the constrained optimization problem defined in Equation 3.2. Algorithm 1 outlines this interpretation.

The strategy performs an alternating search along two principal axes, as depicted in Figure 3.2. It first explores the axis corresponding to the addition of cells (direction $d = (0, 1)$) from a given starting point, using Algorithm 2. It then shifts the starting point along the axis for the number of kernels in the first layer (direction $d = (1, 0)$) and repeats the search. If permitted by the hardware, the number of kernels is doubled, effectively doubling the starting point x_0. If the new network found is superior, the algorithm continues to double the starting point until generalization no longer improves or hardware limits are breached. Otherwise, the algorithm proceeds by halving the initial number of kernels, i.e., the starting point x_0, until performance degrades or an infeasible point is encountered. The algorithm's final output is represented by $\widetilde{x_j}$.

Procedure 2 explores the axis of cell additions ($d = (0, 1)$) from a starting point x with a unitary step size ($t = 1$). In practice, it continues to add cells as long as the generalization capability increases and the new network remains feasible. For the sake of conciseness in

Algorithm 1 Search Strategy

Require: $x_0 \in \Omega$

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

$\epsilon \leftarrow 0.005$, $k_0 \leftarrow 4$, $x_0 \leftarrow (k_0, 0)$, $d \leftarrow (1, 0)$, $t_0 \leftarrow x_0$, $j \leftarrow 0$

$\widetilde{x_0}$ = EXPLORE_NUM_CELLS(x_0)

$x_1 = x_0 + t_0 \cdot d$

$\widetilde{x_1}$ = EXPLORE_NUM_CELLS(x_1)

if $f(\widetilde{x_1}) > f(\widetilde{x_0})$ **and** $x_1 \in \Omega$ **then**

  do

    j = j + 1

    $t_j = x_j$

    $x_{j+1} = x_j + t_j \cdot d$

    $\widetilde{x_{j+1}}$ = 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

    $t_j = -\frac{x_j}{2}$

    $x_{j+1} = x_j + t_j \cdot d$

    $\widetilde{x_{j+1}}$ = EXPLORE_NUM_CELLS(x_{j+1})

  while $f(\widetilde{x_{j+1}}) \geq f(\widetilde{x_j})$ **and** $x_{j+1} \in \Omega$

end if

Output: $\widetilde{x_j}$

Algorithm 2 EXPLORE_NUM_CELLS(x)

Require: $x \in \Omega$

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

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

do

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

  i = i + 1

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

return x_i

Algorithm 1, x_1 is explored regardless of its feasibility. For the same reason, the feasibility check for point x_{j+1} is performed after the exploration. A similar simplification occurs in Algorithm 2, where the network x_{i+1} is trained even if it is not feasible.

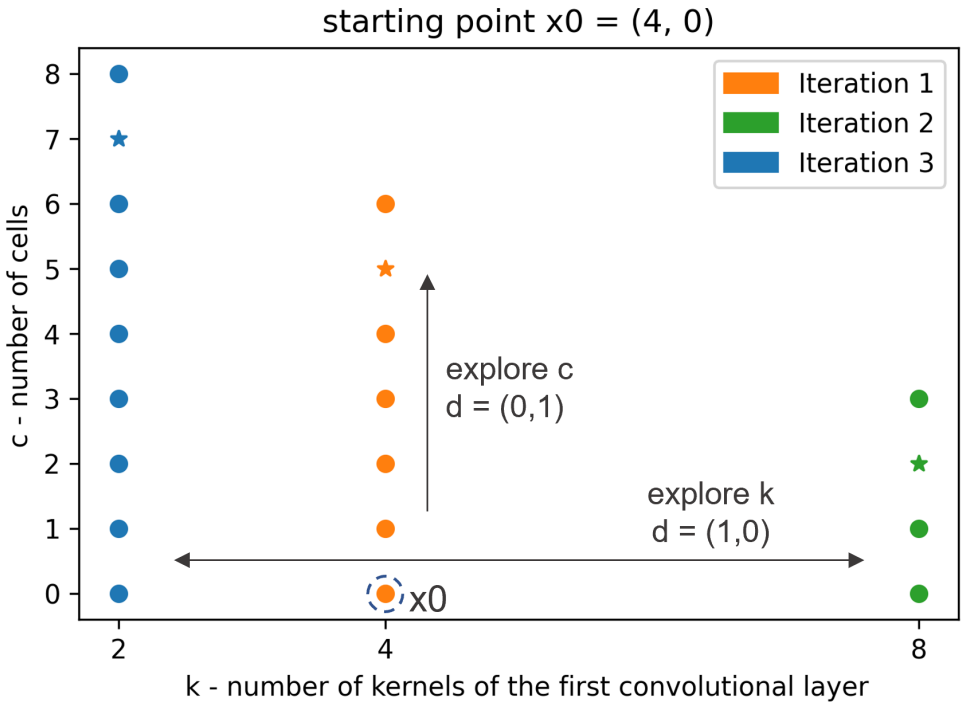


Figure 3.2: A figure showing the points explored by the algorithm during a sample run, in which the generalization capability does not increase during iteration 1 and worsens during iteration 2. The star marker represents the network with the highest generalization capability for each iteration. During each iteration, the points move along the c-axis. After each iteration, the starting point moves along the k-axis.

3.3 Experimental Validation

3.3.1 Experimental Setup

To evaluate the efficacy of the proposed technique, five task-specific classification problems were selected. First, the models obtained using ColabNAS are compared with those derived from applying TL, a common method for designing lightweight task-specific CNNs [59]. The objective of this comparison is to demonstrate that ColabNAS can produce substantially smaller models than TL, with only a minor trade-off in test set accuracy, within a reasonable timeframe. Subsequently, three different microcontrollers (MCUs) are used to showcase the ability of ColabNAS to adapt to various hardware platforms, particularly in low-RAM environments. Finally, a comparison with state-of-the-art HW-NAS methods is presented, based on the Visual Wake Word dataset, a standard TinyML benchmark [44].

Code, models, and datasets are made publicly available as Google Colaboratory notebooks at the link provided in Section 3.1. All computations were executed on Google Colaboratory using Tesla T4 GPUs, which are commonly offered to free subscription users at the time of this writing.

The five classification problems were chosen to provide a diverse set of common task-specific datasets, all publicly available, with sizes ranging from a few thousand to hundreds of thousands of images. This selection also ensures the reproducibility of the experiments on free-subscription online GPU platforms. For datasets where a test split was not provided, one was created using a 0.2 split, allocating 80% of the data for training and 20% for testing. A brief description of each task follows.

3.3.2 Melanoma Skin Cancer

The Melanoma Skin Cancer classification task involves discriminating between benign and malignant images of melanoma. The dataset consists of a training set with 9,605 images and a test set of 1,000 images. It is publicly available on Kaggle as the "Melanoma Skin Cancer Dataset of 10000 Images" [72].

3.3.2.1 Visual Wake Words

The Visual Wake Words classification task, introduced by [41], aims to distinguish between images with and without a human presence. The dataset is composed of a training set containing 115,000 images and a test set of 8,000 images, derived from the "minival ids" as done by [41].

3.3.2.2 Animals-3

The Animals-3 classification task requires discriminating among three animal species: horse, butterfly, and hen. The dataset contains 2,623 instances of horses, 2,112 of butterflies, and 3,098 of hens. It is a subset of the Animals-10 dataset, which is publicly available on Kaggle*.

*`https://www.kaggle.com/datasets/alessiocorrado99/animals10`. Accessed: 28th October 2025.

3.3.2.3 Flowers-4

The Flowers-4 classification task is to differentiate between four classes of flowers: dandelion, iris, tulip, and magnolia. The dataset includes 1,052 instances of dandelion, 1,054 of iris, 1,048 of tulip, and 1,048 of magnolia. It is a subset of the Flowers dataset, available on Kaggle[†].

3.3.2.4 MNIST

The MNIST classification task [73] involves distinguishing between ten handwritten digits (0 through 9). It is a subset of a larger dataset from NIST and comprises a training set of 60,000 images and a test set of 10,000 images.

3.3.3 Comparison with Networks from Transfer Learning

This section provides a comparison between ColabNAS and TL. For the TL approach, MobileNetV2 [49], with frozen weights pre-trained on ImageNet1k [22], was used as the feature extraction backbone. The extracted features were then compressed by a two-dimensional global average pooling layer and fed into a shallow classifier consisting of a single dense layer. The pre-processing pipeline was identical to that of ColabNAS, as described in Section 3.2. It applies horizontal flips and random rotations, followed by min-max standardization and batch normalization. These models were trained for 20 epochs with a learning rate of 10^{-3} and a batch size of 128. Subsequently, the trained models were fine-tuned for 10 epochs with the backbone unfrozen, using a learning rate of 10^{-5}.

In contrast, ColabNAS was executed using the characteristics of the TL-derived models (RAM occupancy, ROM occupancy, and MACC) as constraints. Candidate solutions were trained for 100 epochs with a learning rate of 10^{-3} and a batch size of 128. An input size of 224x224x3, as required by the pre-trained MobileNetV2, was used for all datasets. Results for the Visual Wake Words and MNIST datasets are not included in this comparison due to their large number of images, which would cause the process to exceed the time limits for free users on Google Colaboratory.

The comparison metrics are test accuracy, peak RAM occupancy, Flash occupancy, and the number of MACCs. Post-training quantization (PTQ) was applied to all models. The X-CUBE-AI software from ST Microelectronics was used to measure peak RAM

[†]`https://www.kaggle.com/datasets/l3llff/flowers`. Accessed: 28th October 2025.

and Flash occupancy, while the MACC count was computed using the Keras backend of TensorFlow.

3.3.3.1 Melanoma Skin Cancer

Table 3.1 presents the comparison for the Melanoma Skin Cancer classification task. In this case, ColabNAS surpassed the result from TL in all aspects. The network produced by ColabNAS is 3.1 percentage points more accurate, occupies 4.6 times less RAM, 35.2 times less Flash, and performs 6.8 times fewer MACCs compared to the TL result.

Table 3.1: Test accuracy, peak RAM occupancy, Flash occupancy, MACC, and search cost of the resulting models from TL and ColabNAS for the Melanoma Skin Cancer dataset.

	Test Acc. [%]	RAM [kiB]	Flash [kiB]	MACC [MM]	Search Cost [hh]:[mm]
TL	88	2,523	2,650	300	00:16
ColabNAS	91.1	547	75.2	44	03:52

3.3.3.2 Animals-3

The comparison for the Animals-3 classification task is shown in Table 3.2. Here, ColabNAS produced a model that occupies 2.6 times less RAM, 13.4 times less Flash, and performs 2 times fewer MACCs, while being 6 percentage points less accurate compared to the TL result.

Table 3.2: Test accuracy, peak RAM occupancy, Flash occupancy, MACC, and search cost of the resulting models from TL and ColabNAS for the Animals-3 dataset.

	Test Acc. [%]	RAM [kiB]	Flash [kiB]	MACC [MM]	Search Cost [hh]:[mm]
TL	99.2	2,523	2,653	300	00:11
ColabNAS	93.2	988	197.3	153	03:02

3.3.3.3 Flowers-4

Finally, Table 3.3 shows the comparison for the Flowers-4 classification task. For this dataset, ColabNAS generated a model that occupies 4.6 times less RAM, 44.4 times less Flash, and performs 6.8 times fewer MACCs, with an accuracy that is 5.4 percentage points lower than the TL result.

Table 3.3: Test accuracy, peak RAM occupancy, Flash occupancy, MACC, and search cost of the resulting models from TL and ColabNAS for the Flowers-4 dataset.

	Test Acc. [%]	RAM [kiB]	Flash [kiB]	MACC [MM]	Search Cost [hh]:[mm]
TL	99.2	2,523	2,653	300	00:07
ColabNAS	93.8	546	59.7	44	01:20

3.3.4 Evaluation of the Hardware-Aware Feature

This section evaluates the hardware-aware capabilities of ColabNAS by presenting results for different hardware targets. To emphasize the ability of ColabNAS to provide lightweight CNNs, three low-RAM STMicroelectronics (STM) MCUs from the Ultra-low Power series were selected: the L010RBT6 (L0), the L151UCY6DTR (L1), and the L412KBU3 (L4). The generated networks were evaluated using the STM32 X-Cube-AI software.

Table 3.4: 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 3.4 summarizes the key specifications of each MCU: available RAM, Flash, and CoreMark score, a benchmark for comparing the performance of commercial MCUs. The

values in this table establish the constraints for running ColabNAS on each target. RAM and Flash constraints were used as specified, while the MACC upper bound was set by multiplying the target's CoreMark score by 10^4 to permit a fair exploration of the search space. An input size of 50x50x3 was used to accommodate the constrained resources of the MCUs. The results for each dataset follow.

3.3.4.1 Melanoma Skin Cancer

Table 3.5 displays the resulting models for each target on the Melanoma Skin Cancer dataset. ColabNAS successfully provided a feasible model for each target, adapting to the available resources. As the target's resources increase, so does the size of the resulting model. The models show an average test accuracy drop of 1.8 percentage points as the target's resources decrease. The drop is most significant when moving from L1 (the medium target) to L0 (the small target).

Table 3.5: Test accuracy, RAM and Flash occupancy, MACC, and search cost of the resulting models for each target, indicated in its abbreviated form, 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	2,075	00:17

3.3.4.2 Visual Wake Words

Table 3.6 presents the models for each target on the Visual Wake Words dataset. ColabNAS generated a feasible model for each target, tailored to its resources. Larger targets yield larger models. A mean test accuracy drop of 4.2 percentage points is observed as resources are lowered. The most significant drop occurs between the L1 and L0 targets.

3.3.4.3 Animals-3

Table 3.7 shows the models for the Animals-3 dataset. ColabNAS was able to generate a feasible model for each target, with model size scaling with available resources. The

Table 3.6: Test accuracy, RAM and Flash occupancy, MACC, and search cost of the resulting models for each target, indicated in its abbreviated form, 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	2,086	2:47

models exhibit a mean test accuracy decrease of 8.65 percentage points with diminishing resources, with the largest drop again occurring between the L1 and L0 targets.

Table 3.7: Test accuracy, RAM and Flash occupancy, MACC, and search cost of the resulting models for each target, indicated in its abbreviated form, 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	2,086	00:09

3.3.4.4 Flowers-4

The results for the Flowers-4 dataset are in Table 3.8. As observed previously, ColabNAS produced a feasible, resource-adapted model for each target. The models show a mean test accuracy drop of 6.05 percentage points as resources are reduced. In this case, the most significant drop is between the L4 (large target) and L1 (medium target) devices.

3.3.4.5 MNIST

Table 3.9 shows the models for the MNIST dataset. Feasible models were produced for each target, adapting to the available resources. The models exhibit a mean test accuracy

Table 3.8: Test accuracy, RAM and Flash occupancy, MACC, and search cost of the resulting models for each target, indicated in its abbreviated form, 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	2,075	00:10

drop of 4.9 percentage points with decreasing resources, with the largest drop between the L1 and L0 targets.

Table 3.9: Test accuracy, RAM and Flash occupancy, MACC, and search cost of the resulting models for each target, indicated in its abbreviated form, 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	2,087	01:29

3.3.4.6 Final Considerations

The experiments demonstrate that ColabNAS can successfully adapt to targets with low RAM availability, providing feasible solutions tailored to the available resources. The characteristics of the resulting models appear to be more dependent on the target's hardware than on the dataset. For a chosen target, the model's RAM and MACC counts remain nearly constant across the different datasets explored, whereas the Flash occupancy shows greater variation. The search time appears to be proportional to the number of images in the dataset, given a fixed input size. By further reducing the input size, it should be possible to obtain networks with even smaller RAM footprints, likely at the cost of reduced test accuracies.

3.3.5 Comparison with State-of-the-Art HW-NAS

This section compares ColabNAS with two leading HW-NAS techniques for CNNs: MicroNets by [37] and MCUNet by [10]. The comparison is conducted on the Visual Wake Word dataset [41], a standard benchmark for TinyML models [44]. The Imagenet1k benchmark [22] is not included due to its general-purpose nature, which is incompatible with the task-specific focus of the presented method.

The model with the lowest RAM occupation was selected from both competing projects[‡]. Subsequently, ColabNAS was run on the Visual Wake Word dataset using the same hardware target as Micronets, which had the smallest RAM among the targets of the two projects: the STM32F446RE MCU, featuring 128 kiB of RAM, 512 kiB of Flash, and a CoreMark score of 608.

Table 3.10: Test accuracy, RAM and Flash occupancy, latency, and input size for the chosen models from Micronets, MCUNet, and ColabNAS 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	64x64x3
Micronets	76.8	70.5	273.81	1.15	50x50x3
ColabNAS	77.6	31.5	20.83	0.432	50x50x3

Table 3.10 summarizes the comparison results, which are also visualized in Figure 3.3. All models are in TFLite format and are fully quantized for 8-bit inference. Test accuracy, RAM occupancy, and Flash occupancy are compared. Since there is no direct method to measure MACCs using the TFLite API, these values are not presented. To compare execution times, a latency value was measured using the IPython magic command "%timeit" with the TFLite interpreter invocation. The input tensor contained random

[‡]For MicroNets, the "vww2_50_50_INT8.tflite" model was downloaded from the ARM GitHub page `https://github.com/ARM-software/ML-zoo/blob/master/models/visual_wake_words/micronet_vww2/tflite_int8/vww2_50_50_INT8.tflite`. Accessed: 28th October 2025. For MCUNet, the "mcunet-10fps-vww" model was downloaded from the project's web page `https://hanlab18.mit.edu/projects/tinyml/mcunet/release/mcunet-10fps_vww.tflite`. Accessed: 28th October 2025.

data. All measurements were performed within the same Google Colaboratory session on a dual-core Intel(R) Xeon(R) CPU at 2.20GHz, without GPU involvement.

The ColabNAS model shows improvements in all metrics compared to the one from Micronets. However, the solution proposed by MCUNet remains more accurate. It achieves 9.8 percentage points higher accuracy than the ColabNAS solution, while occupying 5.35 times more RAM, 25.47 times more Flash, and being 5 times slower.

3.3.5.1 Search Costs Comparison

According to [10], MCUNet required 300 GPU hours to produce its architecture. In contrast, MicroNets [37] does not report the time spent on architecture search. However, the authors state that they run DNAS for 200 epochs to find it. DNAS, as described by [16], uses 1.5 GPU days to find a network on the CIFAR-10 dataset (this does not include the selection cost of 1 GPU day or the final evaluation cost of 1.5 GPU days). In that case, DNAS was run for 100 epochs on a dataset of 60,000 32x32 images. Given that the Visual Wake Word training set consists of 107,954 images, the input volume adopted is 50x50x1, and the number of epochs is doubled, the time spent by MicroNets is presumably higher than that spent on CIFAR-10. Therefore, ColabNAS is considerably faster than both MCUNet and MicroNets.

3.4 Conclusion

This chapter presented ColabNAS, an affordable HW-NAS technique for designing task-specific CNNs that are suitable for low-RAM MCUs. Its novel search strategy, inspired by Occam's razor, features a low search cost. It achieves state-of-the-art results on the Visual Wake Word dataset in just 3.1 GPU hours, improving upon the solution found by Micronets in all measured aspects.

The low search cost enables its execution on free online services such as Kaggle Kernel or Google Colaboratory, eliminating the need for a high-end GPU. This can be particularly beneficial for end-users and researchers aiming to apply lightweight CNNs to custom classification problems.

The technique also provides task-specific lightweight CNNs that, on average, occupy 3.9 times less RAM, 31 times less Flash, and have 5.2 fewer MACs than those obtained by applying TL and fine-tuning based on MobileNetV2 for the same task, which is one of the most common techniques in research. This is achieved at the cost of an average reduction

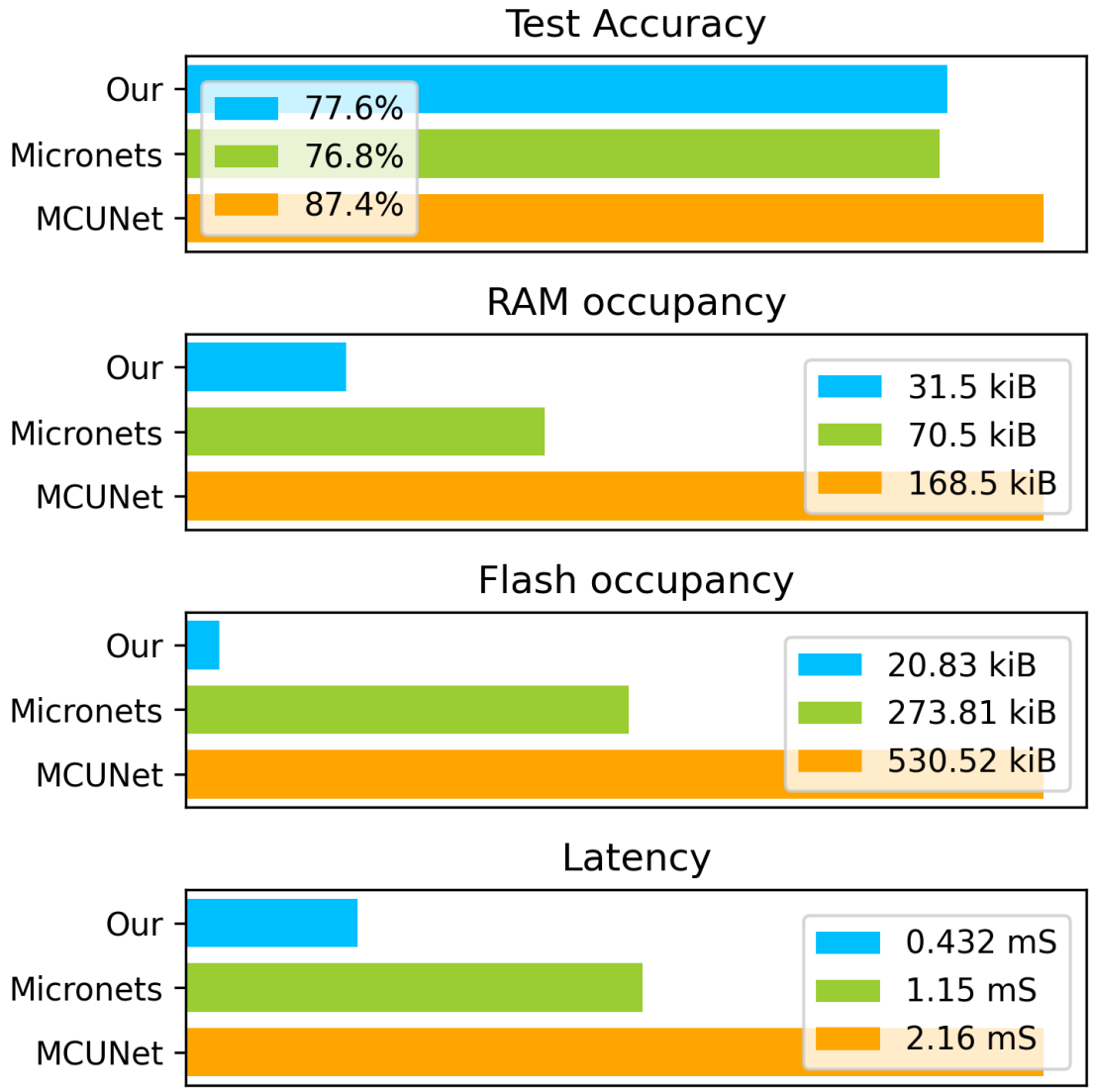


Figure 3.3: Graphical representation of test accuracy, RAM and Flash occupancy, latency, and input size for the chosen models from Micronets, MCUNet, and ColabNAS for the Visual Wake Words dataset.

of 2.8 percentage points in accuracy. Such a trade-off is well-suited for deployment in the growing field of wearable and distributed devices with embedded electronics. Furthermore, reducing the computational cost of the final application translates to more efficient power consumption. In a world where artificial intelligence is becoming increasingly pervasive, reducing energy consumption is a fundamental step towards a more sustainable future.

CHAPTER 4

NanoNAS: Automatically Design Convolutional Neural Networks for Ultra-Low-Power Microcontrollers on Your Laptop

The contribution presented in this chapter has been previously published as a journal paper [74] which expanded the work previously published as a conference paper [75].

4.1 Introduction

Sensing nodes can achieve significant benefits from real-time inference procedures supported by machine learning algorithms. For instance, teleceptive sensing, which involves sensing without physical contact through technologies like optical sensors or radars, relies heavily on Convolutional Neural Networks (CNNs) [76]. However, CNNs introduce substantial computing costs and complex training procedures, characteristics often incompatible with sensing nodes that must operate under severe energy constraints [77]. Hardware-Aware Neural Architecture Search (HW-NAS) presents a compelling solution by offering an automatic procedure to generate and train suitable CNNs that adhere to the predefined constraints of the physical device.

In recent years, significant efforts have been directed toward implementing CNNs on devices with limited computational capabilities. This led first to manually designed networks for mobile devices [39, 52, 63] and subsequently to the rise of HW-NAS [9, 35], which automates the design process by considering the target hardware's available resources, such as RAM, Flash memory, or FLOPS. However, state-of-the-art HW-NAS methods, including μNAS [78], MCUNet [10], and Micronets [37], have primarily targeted high-performance microcontroller units (MCUs). Table 4.1 highlights the significant

Table 4.1: Hardware features of the high-end MCUs targeted by state-of-the-art HW-NAS, versus the MCUs targeted by this work [1].

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

differences between these high-end MCUs and the ultra-low-power line commonly found in battery-operated sensing nodes.

Furthermore, existing frameworks generally involve resource-intensive optimization procedures. A search that demands hundreds of hours of GPU computing is often impractical, for instance, in a gateway device that must produce custom CNNs for IoT edge nodes using local data without cloud access. For broader application, two qualities are essential: 1) the capacity to generate tiny CNNs compliant with the much tighter constraints of ultra-low-power MCUs, and 2) a limited computational cost for the NAS procedure itself.

Recent studies [47, 75, 79] have demonstrated that NAS techniques can be effectively utilized to design CNNs that comply with the stringent constraints imposed by this class of microcontrollers, such as in near-sensor computing [80]. These works have shown that the constraints of the ultra-low-power line necessitate a specialized approach that deviates from conventional HW-NAS techniques and that the search procedure can be executed without needing GPUs.

This chapter provides additional insights into the HW-NAS approach first proposed in [75]. It aims to demonstrate empirically that a lightweight search procedure can effectively adapt to the hardware constraints of the target platform, generating custom tiny CNNs capable of real-time execution. Furthermore, it shows that the search process itself can be executed on embedded systems by incorporating a memory constraint on the device performing the NAS into the optimization problem. The generated CNNs achieved state-of-the-art results in human-recognition tasks on the Visual Wake Word dataset. The

entire HW-NAS procedure was executed on a laptop with 16 GB of RAM in under 4 CPU hours*.

4.1.1 Lightweight HW-NAS Overview

A HW-NAS is characterized by three key components: the search space, the optimization problem, and the search strategy. The HW-NAS presented in [75] utilizes a constrained cell-wise search space, which establishes the foundation for achieving two primary goals: 1) the capability to generate CNNs that fit within strict resource constraints of the target hardware, and 2) the flexibility to appropriately manage the computational cost of the search process. The first goal is a critical requirement for a HW-NAS designed for ultra-low-power MCUs. The second goal is essential for a HW-NAS intended to run on embedded systems.

In the adopted search space, a candidate solution is derived from an established structure organized as follows [75]:

1. a pre-processing pipeline that performs min-max standardization and batch normalization on the input data;
2. a convolutional layer with k kernels;
3. c cells;
4. a classifier composed of a global average pooling layer followed by a dropout layer, which feeds into a final layer with softmax activation and a number of neurons equal to the number of classes.

Here, a $cell$ is a sequence of three layers: a 2-dimensional max pooling layer, a batch normalization layer, and a convolutional layer. Each max pooling layer halves the input resolution using a 2x2 receptive field with a (2,2) stride. The convolutional layers employ (3,3) kernels with a (1,1) stride and ReLU activation. The number of kernels n_c used in the c-th cell is dependent on k according to the following rule:

$$n_c = \begin{cases} k & \text{if } c = 0 \\ \left\lceil (2 - \sum_{i=1}^{c-1} 2^{-i}) \cdot n_{c-1} \right\rceil & \text{if } c \geq 1 \end{cases} \tag{4.1}$$

*code available at `https://github.com/AndreaMattiaGaravagno/NanoNAS`.

Consequently, in this search space, admissible solutions can be described by two parameters: k, which defines the number of kernels in the first convolutional layer, and c, which defines the number of cells.

4.2 Methodology

The proposed lightweight HW-NAS inherits its overall design from the work presented in [75]. It employs enhanced versions of both the optimization problem and the search strategy. The objective is to combine the ability to generate tiny CNNs that can run in real-time on sensing nodes with the capability of executing the NAS itself on an embedded system, which might serve as a central unit in a sensor network.

4.2.1 The Optimization Problem

HW-NAS aims to find the optimal neural architecture for a target hardware within a given search space $\mathcal{S}_S$. This is formulated as a constrained optimization problem where the objective function describes the metric used to evaluate candidate solutions (i.e., CNNs), and the constraints define the boundaries of the search. In this problem formulation, a candidate CNN is evaluated by training it and assessing its validation accuracy. The constraints for a candidate CNN are the available RAM, Flash memory, and multiply-and-accumulate (MAC) operations on the target hardware that will host the inference phase. The number of MAC operations is used as a reliable proxy for latency [37]. Additionally, a specific constraint is introduced to limit the memory available for the training phase on the device hosting the search procedure.

This leads to the problem formulation P:

$$P : \begin{cases} \max f(x) \\ \phi_R(x) \leq \xi_R, \phi_F(x) \leq \xi_F, \phi_M(x) \leq \xi_M, \theta(x) \leq \Theta_T \\ \xi_R, \xi_F, \xi_M, \Theta_T > 0 \end{cases} \tag{4.2}$$

where $x = (k, c)$. The function f returns the validation accuracy. The parameters ξ_R, ξ_F, and ξ_M represent the upper bounds for RAM usage, Flash usage, and MAC operations on the target hardware, respectively. Θ_T sets the bound on RAM usage for the search procedure. Thus, the function ϕ_R returns the CNN's RAM occupancy, ϕ_F returns its Flash occupancy, and ϕ_M returns the number of MAC operations required by the CNN, while

θ returns the RAM occupancy on the device hosting the NAS. These quantities depend on the number of kernels k in the first convolutional layer, the number of cells c, the size of the network's input vi, and the adopted platform, in this case TF Lite Micro [81]. The search variable $x = (k, c)$ does not include vi, which is a fixed quantity defined by the user.

4.2.2 The Search Strategy

The optimization problem P is solved using a derivative-free technique, as $f(x)$ is not differentiable [75]. This procedure compares candidate networks by assessing their validation score $f(x)$, which in principle requires a full training procedure for each candidate CNN. Training, however, is a computationally demanding step. The cost of the NAS itself is roughly proportional to the number of training procedures needed to implement the search strategy. Therefore, a NAS designed to run on resource-constrained devices requires a custom approach.

Algorithm 3 formalizes the proposed search strategy. It relies on three key factors: 1) the search space inherently generates tiny CNNs as candidate solutions; 2) early stopping is used to limit the computational cost of the training process (detailed further in Section 4.2.3); and 3) a specific constraint is included to account for resource availability on the platform executing the NAS. The search begins with $k = 1$ kernel in the first layer. For a given k, the algorithm increments the number of cells c until the corresponding $CNN(k, c)$ first, satisfies the constraints imposed by ξ_R, ξ_F, and ξ_M, and second, scores a validation accuracy $f(k, c)$ greater than that achieved by $CNN(k, c - 1)$. This process is then repeated with $k + 1$. If the best CNN found with $k + 1$ outperforms the best CNN from the k iteration, the search continues by further incrementing k. Otherwise, the search terminates.

If the training of a network $CNN(k, c)$ cannot be completed due to limited resources on the computing system running the NAS, its associated accuracy $f(k, c)$ is set to 0. This enables the search strategy to operate effectively even on resource-constrained embedded systems.

4.2.3 The Early Stopping Strategy

CNNs are trained with gradient descent techniques that require multiple applications of the backpropagation algorithm. This work focuses on tiny networks with a small set

Algorithm 3 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 (4.2) AND Δ > 0
    return c, best
end procedure
```

of parameters. Consequently, their representation capability is limited, and standard approaches to accelerate training convergence cannot be exploited. It is well-known, however, that early stopping can act as a regularization technique, since the loss function generally decreases dramatically in the first few epochs of training. Therefore, a coarse version of early stopping can be used to find a trade-off between the computational cost of the search strategy and the ability to identify an effective configuration for the pair (k, c). This solution is viable because the primary goal is not to obtain a precise estimation of f itself, but rather to adopt a reliable criterion for deciding if one CNN is more promising than another.

If a full training cycle consists of n_{ep} epochs and the performance estimation is performed after only n'_{ep} epochs, the speedup is approximately n_{ep}/n'_{ep}. This approach notably impacts only the time and energy consumption, without affecting memory occupation. Naturally, when $n_{ep} \gg n'_{ep}$, the speedup is greater, but the risk of incorrect selections also increases.

The proposed HW-NAS thus utilizes early stopping to reduce the computational cost of the search procedure while simultaneously addressing issues like overfitting and local

minima. Section 4.3.3 will empirically demonstrate that training can be stopped after very few epochs without compromising the effectiveness of the NAS.

4.3 Experimental Validation

The experiments were conducted on three datasets: Visual Wake Words (VWW)[41], Cifar-10 (C-10) [12], and Melanoma Skin Cancer [72]. VWW and C-10 are standard benchmarks for Tiny Visual ML. The target platforms are from the STM32 ultra-low-power line: L010RBT6 (hereafter L0), L151UCY6DTR (L1), and L412KBU3 (L4). The constraints in (4.2) were configured according to Table 4.1; the MAC upper bound was derived by rescaling the CoreMark score by a factor of 10^4. An early stopping criterion of $n'_{ep} = 3$ was used in the search strategy. The CNN selected by the NAS was subsequently trained for 100 epochs with a batch size of 128 and a learning rate of 10^{-2} using the Adam optimizer [82]. The validation set comprised 10% of the original training set. Generalization performance was evaluated on public test sets.

Runtime RAM and Flash usage were measured 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 equipped with an 11th Gen Intel®Core™i7-11370H CPU @ 3.30GHz, 16 GB of RAM, and a 512 GB SSD.

4.3.1 Evaluation of the Hardware-Aware Feature

The initial experiment was designed to demonstrate that the proposed HW-NAS can successfully adapt the generated CNNs to the target hardware, even for ultra-low-power MCUs. Table 4.2 summarizes the results of this experiment. The table is organized into four sections. The first section specifies the MCUs. The second provides the pair (k, c) that characterizes the selected CNN and the temporal cost of the NAS search procedure. The third section details the features of the selected CNN: RAM occupancy, Flash occupancy, and MAC operations. The final section reports the performance of the CNN in terms of test accuracy and latency (inference time per image). Latency was measured using the STM32Cube.AI runtime on the L4 platform in balanced mode.

The proposed HW-NAS successfully adapted the architecture of the generated CNNs to each specific platform. The search cost increased with the hardware resources of the target device, a consequence of the expanded search space. The search costs reported in Table 4.2 represent a worst-case scenario, as they were obtained without GPU acceleration.

Table 4.2: Performance of the proposed HW-NAS on target MCUs.

MCU	NAS		Tiny CNN			Performance	
STM32	**Arch. (k,c)**	**Cost [hh]:[mm]**	**RAM [kiB]**	**Flash [kiB]**	**MAC [MM]**	**Acc. [%]**	**Lat. [mS]**
Visual Wake Words (vi = 50x50)							
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 (vi = 32x32)							
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 (vi = 50x50)							
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

To test its portability, the experiment targeting the L4 MCU on the VWW dataset was repeated by executing the NAS on a Raspberry Pi 4 with 4 GB of RAM; the search cost was 34 hours and 29 minutes. This result confirms that the proposed HW-NAS can be executed successfully on resource-constrained devices.

As anticipated, larger CNNs were generated when the L4 MCU was the target. The generalization performance of the selected CNNs scaled with their size. Overall, the proposed method effectively manages very strict constraints by balancing generalization performance with available hardware resources.

4.3.2 Comparison with State-of-the-Art Methods

Table 4.3 provides a comparison on the VWW dataset with three state-of-the-art HW-NAS frameworks for MCUs: MCUNet [10], Micronets [37], and ColabNAS [47]. The first two focus on high-performance MCUs, whereas ColabNAS targets ultra-low-power MCUs. For MCUNet and Micronets, the comparison involves the smallest CNN presented in their respective studies. In the case of MCUNet, the target device was an STM32F412

Table 4.3: Comparison with existing HW-NAS on VWW dataset.

Model	Acc [%]	RAM [kiB]	Flash [kiB]	MACC [MM]	Search time [hh]:[mm]
Proposal	77	28.5	23.7	1.3	3:17 CPU / 1:08 GPU
ColabNAS [47]	77.6	31.5	20.83	2	7:09 CPU / 2:28 GPU
Micronets [37]	76.8	70.5	273.8	3.3	n.a GPU
MCUNet [10]	87.4	168.5	530.5	6	> 300:00 GPU

[10], while for Micronets and ColabNAS, the target was an STM32F446RE [37]. For the proposed HW-NAS, the table reports the CNN selected when L4 was the target platform.

The table reveals that the CNN selected by the proposed HW-NAS achieved an accuracy on the VWW dataset comparable to that of the CNNs selected by Micronets [37] and ColabNAS [47]. However, the CNNs from these latter two frameworks require more RAM, more Flash memory, and more MAC operations. While the CNN from MCUNet [10] achieved the highest accuracy, it cannot be hosted on an ultra-low-power MCU like the L4. It is important to note that Lin et al. [10] demonstrated that a well-known lightweight CNN for image classification, MobileNetV2, requires at least 256 kB of RAM to attain satisfactory accuracy on VWW. Such a memory footprint is not available on ultra-low-power MCUs.

The proposed HW-NAS proved to be the most efficient in terms of search cost. This outcome confirms its ability to select effective architectures using a lightweight search strategy.

4.3.3 Analysis of the Early Stopping Criterion

The early stopping criterion is the primary factor contributing to the algorithm's low search cost. However, such an approach could potentially compromise the ability to find an effective CNN. Figure 4.1 illustrates the probability of selecting the exact same CNN that would be identified using 100 epochs when applying the early stopping criterion with 3, 15, and 75 epochs, respectively. A training duration of 100 epochs was used as a reference, as this setup almost always guarantees convergence to a stable local minimum [47]. The plot in Fig. 4.1 displays the number of epochs on the x-axis and the empirically assessed probability on the y-axis.

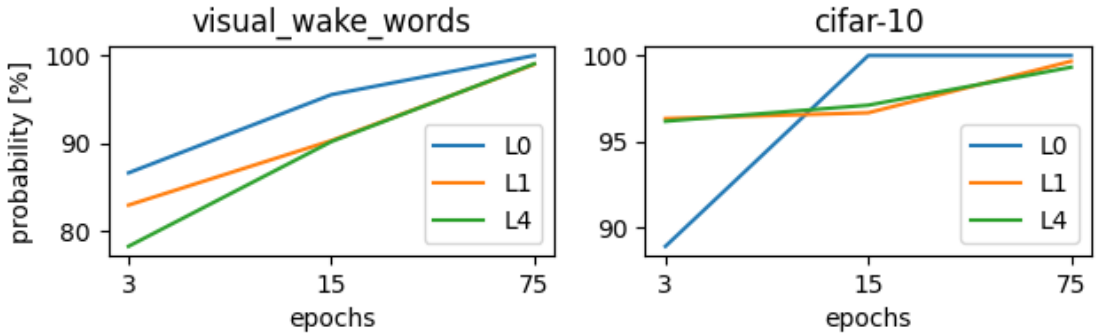


Figure 4.1: Probability of selecting the best CNN based on the number of epochs used during the search.

The experiment confirms that even when using as few as 3 epochs, the proposed HW-NAS selects the most promising candidate, which is the CNN that would have been chosen with 100 epochs, with a high probability (> 78%). This is a significant result, considering that the search time scales linearly with the number of epochs. This provides a theoretically estimated speed-up of 100/3. However, the overhead from model initialization and dataset loading procedures, which must be repeated before each training, can substantially reduce this speed-up, especially with large datasets.

4.4 Conclusion

Many applications can benefit from the integration of deep learning into sensing nodes. However, the design and deployment of these solutions demand domain-specific skills and specialized computing resources. A Hardware-Aware Neural Architecture Search (HW-NAS) for ultra-low-power MCUs, supported by a lightweight optimization procedure, was proposed. This NAS demonstrated its ability to fit the constraints of different target devices while delivering state-of-the-art CNNs on the Visual Wake Words dataset. Furthermore, the search procedure can be completed in a few hours without requiring GPUs. Experiments confirmed that the HW-NAS can even run on a resource-constrained device like a Raspberry Pi 4.

Several aspects warrant further investigation. First, large datasets could present a challenge, particularly if the search procedure is intended to run on an embedded system. Second, since the proposed HW-NAS is designed for tiny CNNs, it may not achieve state-of-the-art results when targeting high-performance MCUs. In such cases, traditional

NAS methods or hand-crafted architectures may offer superior 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. 1409 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-bAsed weaRable exoskeletoNs for assistance in daily life" – CUP: D53D23016200001, J53D23014090001 - Grant Assignment Decree No. 1383 adopted on September 01, 2023 by the Italian Ministry of University and Research (MUR).

CHAPTER 5

GatewayNAS: Automatic Design of Neural Networks at the Edge

The contribution presented in this chapter has been previously published as a journal paper [83] which expanded the work previously published as a conference paper [79].

5.1 Introduction

The application of NNs is widespread in the domain of the IoT [84]. A common practice in this field is to augment training datasets with data gathered from available sensors to enhance generalization performance. However, certain data types are inherently sensitive, such as healthcare records [6], industrial process data [85], and biometric information [86]. Concerns over privacy often restrict some organizations from leveraging the advantages of machine learning (ML), as they are either unable or unwilling to transmit their data to cloud services for training or automated NN design [87].

To address this challenge and extend the accessibility of NNs to privacy-sensitive scenarios, a method is presented for the automatic design of NNs for sensor nodes, executed directly on IoT gateways using locally collected data. This approach ensures that the entire design process can be confined within the local network, eliminating the need for cloud services and thereby preserving data privacy. In the context of the Healthcare Internet of Things (HIoT), this technique can deliver personalized healthcare services by creating customized, hardware-friendly NNs for individual patients [88, 7]. Similarly, the Industrial Internet of Things (IIoT) can gain from this methodology. For instance, it can facilitate the creation of specialized NNs for intelligent fault diagnosis (IFD) in manufacturing equipment or for quality control, without the associated risk of data leakage that could expose proprietary industrial information [89].

A conceptual overview of the proposed scenario is illustrated in Figure 5.1. Initially, data from sensor nodes are aggregated by a local gateway. Subsequently, the gateway employs the proposed method to design a unique NN for each connected sensor node,

accommodating the possibility that each node possesses different hardware resources. Finally, the resulting NNs are deployed onto their respective sensor nodes. This entire workflow can be executed without transferring data to any external cloud infrastructure, thus safeguarding privacy.

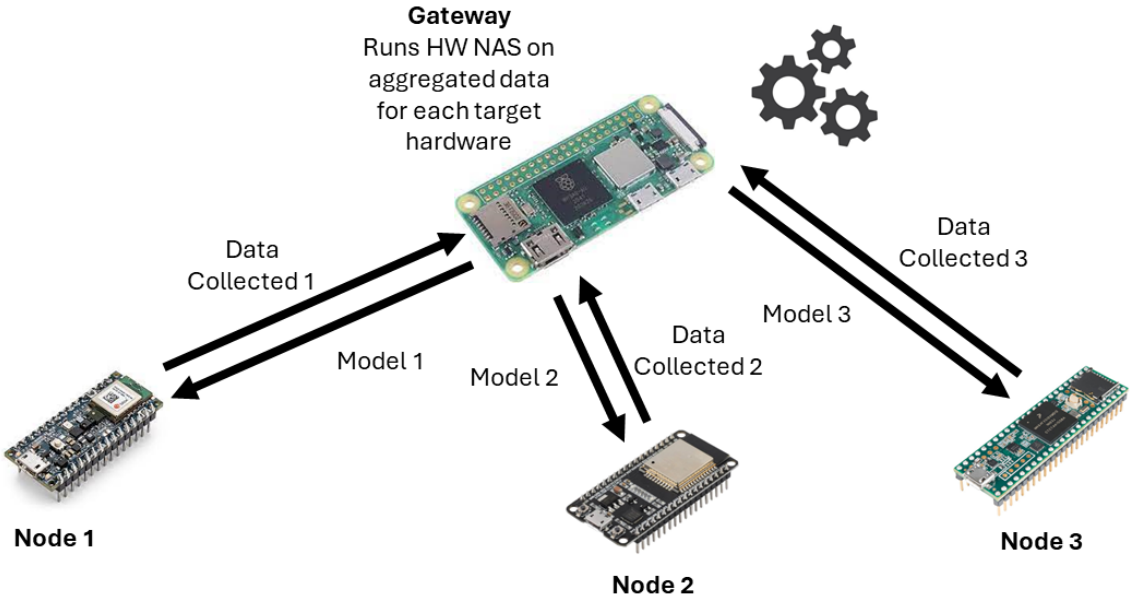


Figure 5.1: Automatic design of neural architectures at the edge, where Hardware-Aware Neural Architecture Search (HW-NAS) runs on an IoT gateway using locally collected data.

The proposed technique utilizes Hardware-Aware Neural Architecture Search (HW-NAS) for the automated design of NNs for sensor nodes. This approach has gained traction in IoT applications [90, 91, 20]. A key challenge in the state of the art is the computational expense of the search procedure itself. While the primary objective of conventional HW-NAS is to produce NNs that conform to the resource limitations of the target inference hardware, the computational resources of the platform executing the search are typically assumed to be unrestricted [92]. This assumption becomes a significant bottleneck when dealing with resource-constrained devices like IoT gateways [93, 94]. To avoid transmitting data to external computational resources, the search procedure must be designed to operate on the available IoT gateway.

The main difficulties in executing HW-NAS on edge platforms are a) managing limited computational power and b) coping with scarce memory. To overcome these obstacles, a derivative-free search strategy is combined with a regular search space. The structured nature of the search space simplifies the evaluation phase by using architectures that are

hardware-friendly, meaning they lack complex branching structures. This design choice reduces memory and computational demands during the backpropagation phase. The derivative-free search strategy further lessens memory and computation requirements compared to derivative-based algorithms.

Historically, HW-NAS has been a resource-intensive process requiring substantial computing infrastructure. For instance, Mnasnet [9] demanded 40,000 GPU hours. Subsequent efforts, such as MCUNet [10], reduced this to 300 GPU hours. More recently, specialized techniques that combine derivative-free optimization with custom-designed search spaces have emerged to accommodate the constraints of ultra-low-power hardware. In [47], the search cost was lowered to 4 GPU hours, and later work by Garavagno et al. eliminated the need for a GPU entirely [75]. Two recent studies have demonstrated the feasibility of running the HW-NAS process itself on an embedded system [79, 74] by imposing a constraint on the memory available to the search platform. Table 5.1 provides an overview of state-of-the-art HW-NAS methods, categorized by the computing resources required for the search. This work introduces an adaptive capability, allowing the search to be tailored to the gateway's available time and energy.

The proposed approach safeguards the training data, parameters, and architecture of the resulting neural network. This offers a more secure alternative to Federated NAS, which requires sharing candidate parameters with an external server to determine the optimal architecture [95]. While placing the server within the local network could also preserve parameter privacy, it would render federated learning techniques unnecessary; the sensor nodes would already be transmitting sensitive data within a secured local environment.

Other methodologies exist for adapting deep learning models to deployment environments. Dynamic Neural Networks can alter their structure and parameters in response to varying inputs during inference, potentially enhancing generalization and interpretability [96]. However, a persistent gap between their theoretical potential and practical efficiency makes them less suitable for current computing hardware [97]. Domain adaptation techniques transfer knowledge from a source domain to a target domain with a different data distribution [98] and have been used to improve NN generalization in IoT applications with limited data [99, 100, 101, 102]. Nevertheless, these methods still place the burden of designing a hardware-compatible neural architecture on the user.

This work builds upon the foundation laid in [79], advancing the concept of running HW-NAS on constrained devices. The primary contributions of this chapter are:

A new search strategy is introduced that achieves state-of-the-art performance on the Visual Wake Words dataset, while decreasing the search time from 4 days to under 10 hours on a Raspberry Pi Zero 2.

An adaptation mechanism is developed that adjusts the search space based on the energy and time constraints of the IoT gateway, enabling HW-NAS execution even with a limited energy budget or a restricted execution timeframe.

The capability to handle time-series data is incorporated, yielding state-of-the-art results on the CWRU dataset for Intelligent Fault Diagnosis, with a search time of just 1 hours and 52 minutes on a Raspberry Pi 4.

The proposed HW-NAS is released as open-source software, designed for Linux-based embedded devices, available at `https://github.com/AndreaMattiaGaravagno/GatewayNAS`. Accessed: 28th October 2025.

The experimental evaluation involved three distinct single-board computers commonly used as IoT gateways and three different ultra-low-power microcontroller units (MCUs) representing sensor nodes. The experiments confirmed that the proposed HW-NAS could successfully adhere to constraints on execution time and energy consumption while generating architectures that delivered state-of-the-art performance on the selected benchmarks.

The remainder of this chapter is structured as follows. Section 5.2 provides a brief introduction to HW-NAS fundamentals and discusses the specific challenges of running it on resource-constrained devices. Section 5.3 details the proposed approach for designing an HW-NAS suitable for the IoT environment. The experimental setup is described in Section 5.4.1, and the results are presented in Section 5.4.2. Finally, Section 5.5 offers concluding remarks.

5.2 Background

5.2.1 Hardware-Aware Neural Architecture Search

HW-NAS automates the discovery of an optimal neural architecture for a specific task on a target hardware platform. This process is typically formulated as an **optimization problem**, subject to the resource constraints of the target hardware. The objective function defines the metrics used for selecting the best architecture from a set of candidates, which constitutes the **evaluation process** [103]. Equation (5.1) presents a typical formulation

Table 5.1: Target execution platform of state-of-the-art HW-NAS.

Work	GPU	CPU	Embedded	
			Static	Adaptive
Mnasnet [9]	x			
MCUNet [10]	x			
Micronets [37]	x			
ColabNAS [47]	x			
NanoNAS v1 [75]		x		
NanoNAS v2 [74]			x	
Proposal				x

of this optimization problem, where the goal is to find the architecture A that maximizes validation accuracy, $\max \Gamma(A)$, while adhering to a series of constraints. These constraints include limits on RAM usage (ξ_{RAM}), Flash memory (ξ_{Flash}), and the number of multiply-and-accumulate (MAC) instructions (ξ_{MAC}). The terms $\phi_{RAM}(A)$, $\phi_{Flash}(A)$, and $\phi_{MAC}(A)$ represent the RAM usage, Flash memory footprint, and MAC count of a candidate architecture A on the deployment device, respectively.

$$\begin{cases} \max \Gamma(A) \\ \phi_{RAM}(A) \leq \xi_{RAM}, \phi_{Flash}(A) \leq \xi_{Flash} \\ \phi_{MAC}(A) \leq \xi_{MAC} \\ \xi_{RAM}, \xi_{Flash}, \xi_{MAC} > 0 \end{cases} \tag{5.1}$$

The set of all possible candidate solutions is known as the **search space**. It is defined by a set of fixed architectural rules that determine the permissible NN structures. These rules can operate at different levels of granularity: they can apply to individual layers (layer-wise search space), groups of layers known as cells (cell-based search space), or a combination of both (hierarchical search space) [103].

The optimization technique used to solve the problem, referred to as the **search strategy**, dictates how the search space is explored to find the optimal neural architecture [103]. Early methods employed techniques such as Reinforcement Learning (RL) [9], Evolution-

ary Algorithms (EA) [36], Random Search [21], and Bayesian Optimization (BO) [104], which are well-suited to the non-smooth nature of the HW-NAS problem. Subsequently, a new formulation of HW-NAS [16] enabled the use of gradient-based techniques [34, 35]. However, many of these search strategies are computationally intensive [103], sometimes requiring tens of thousands of hours on specialized hardware [10]. In recent years, new approaches have been developed to reduce the search cost by employing custom search strategies [105, 75, 79].

5.2.2 Running HW-NAS on Resource Constrained Devices

Executing HW-NAS on embedded devices presents significant challenges due to their limited resources. Two recent studies [79, 74] have demonstrated that the search cost can be substantially reduced by using a custom derivative-free optimization technique and a cell-based search space. In both cases, the HW-NAS was tailored to generate NNs for deployment on ultra-low-power computing platforms.

The search space adopted in [79] constructs small Convolutional Neural Networks (CNNs) by stacking four distinct cell types: a pre-processing cell, a base cell, a building cell, and a classifier cell. The pre-processing cell applies min-max standardization to improve the convergence rate of gradient descent [70]. The base cell consists of a single convolutional layer with k kernels, serving as the foundation upon which building cells are stacked. Each building cell first reduces the feature resolution by half using a max pooling operator, then applies a convolution followed by batch normalization to enhance training stability and speed [71]. A rectified linear unit activation function then generates the new feature maps. The number of kernels n_c in a convolutional layer is determined by k and the number of previously stacked cells c, calculated as $n_c = n_{c-1} + 2^{1-c} n_{c-1}$, with $n_0 = k$ [79]. This formula is inspired by VGG16 [68], but instead of doubling the number of kernels after each block, the increment is gradually reduced to conserve resources.

Finally, the classifier cell reduces the extracted features using global average pooling to enhance the model's generalization capabilities [69]. A fully connected layer then produces the class predictions, employing a softmax activation function and a number of neurons equal to the number of classes. All convolutional kernels use a 3×3 receptive field.

Consequently, within this search space, neural architectures are formed by stacking building cells on top of the base cell. Each architecture begins with a pre-processing cell and concludes with a classifier cell. Any such architecture can be uniquely identified by

the tuple (k, c), where k represents the number of kernels in the base cell and c is the number of building cells stacked upon it.

To explicitly address the limitations of the device performing the HW-NAS, a custom constraint was introduced in [79] and [74]. This constraint imposes a limit on the memory available for the training phase on the host device. As a result, candidate architectures that exceed this memory limit are deemed unfeasible, even if they could be deployed on the target hardware.

5.3 Methodology

The central feature of the proposed HW-NAS is its ability to be "hardware aware" at two levels: the edge (the device where the NN will be deployed) and the gateway (the device where the NAS will run). To achieve this, the HW-NAS incorporates six distinct constraints:

At the edge: A candidate solution, denoted as an architecture $A(k, c)$, must comply with constraints on the available RAM (ξ_{RAM}), Flash memory (ξ_{Flash}), and MAC instructions (ξ_{MAC}).

At the gateway: The entire process of identifying the optimal architecture must adhere to constraints on available memory (ξ_{MEM}), execution time (ξ_{Time}), and the total energy budget (ξ_{Energy}).

The objective is to maintain the largest possible search space, defined by the number of candidate solutions, while respecting the constraints at the gateway level. The proposed approach generates the set of admissible solutions, S_α, through a two-step procedure. First, it selects architectures $A(k, c)$ that 1) are deployable on the edge device and 2) can be trained on the gateway without exceeding the memory usage constraint ($\phi_{RAM}(A) \leq \xi_{RAM}$), as shown in Equation (5.2).

$$\begin{cases} \forall A \in S_\alpha \\ \phi_{RAM}(A) \leq \xi_{RAM}, \phi_{Flash}(A) \leq \xi_{Flash} \\ \phi_{MAC}(A) \leq \xi_{MAC}, \phi_{MEM}(A) \leq \xi_{MEM} \\ \xi_{RAM}, \xi_{Flash}, \xi_{MAC}, \xi_{MEM} > 0 \end{cases} \tag{5.2}$$

Next, a cropping process is applied to reduce the size of S_α if necessary. The search must also satisfy the constraints on execution time and energy budget, which are estimated

by the functions ϕ_{Time} and ϕ_{Energy} based on the candidate search space. Therefore, the number of solutions in S_α should be maximized, yet remain small enough to allow the gateway to complete the search without violating the constraints ξ_{Time} and ξ_{Energy}. The final set S_α is determined by solving the problem in Equation (5.3).

$$\begin{cases} \max |S_\alpha| \\ \phi_{Time}(S_\alpha) \leq \xi_{Time}, \phi_{Energy}(S_\alpha) \leq \xi_{Energy} \\ \xi_{Time}, \xi_{Energy} > 0 \end{cases} \tag{5.3}$$

Once the set S_α is established, the search strategy can be executed upon it. The following sections provide details on the algorithms used to generate S_α (Section 5.3.1) and the search strategy itself (Section 5.3.2).

5.3.1 Generating the Search Space On The Gateway

The final search space S_α is derived from a two-step process. In the initial step, Algorithm 4 constructs the set $\hat{S}_\alpha$, which serves as a superset of S_α. This set is intended to contain all admissible architectures $A(k, c)$ based on the deployment target, as defined by Equation (5.2). The algorithm begins with the smallest possible candidate, $(k = 1, c = 0)$, and sequentially evaluates candidate architectures $(1, c)$, adding them to $\hat{S}_\alpha$ until an unfeasible architecture is encountered (the while loop at line 6). An architecture $A(k, c)$ is considered unfeasible if it violates any of the constraints in Equation (5.2). The algorithm then increments k (the repeat loop at line 4) and repeats the exploration along the c parameter. This process continues until an architecture $A(k, 0)$ is found to be unfeasible, which serves as the stopping criterion (line 11).

After $\hat{S}_\alpha$ is computed, a cropping process may begin. This is necessary because the gateway must operate within the execution time and energy budget constraints when performing the NAS, as stipulated by Equation (5.3). In essence, both of these quantities are dependent on the number of architectures in the search space, as each one must be evaluated to identify the best CNN. This implies that each architecture requires training. Therefore, it cannot be assumed that the search space can be as large as $\hat{S}_\alpha$. Initially, S_α is set equal to $\hat{S}_\alpha$. The cropping strategy, formalized in Algorithm 5, then systematically reduces the size of S_α to create a search space that allows the gateway to meet its constraints.

First, an upper bound $\bar{t}$ is established for the time required to evaluate a single candidate architecture. This is estimated by selecting the architecture with the most trainable

Algorithm 4 The algorithm that generates the extensive search space $\hat{S}_\alpha$

Require: $\xi_{MEM}, \xi_{RAM}, \xi_{Flash}, \xi_{MAC}$
Ensure: $\hat{S}_\alpha$
procedure SEARCH SPACE - CREATION
  $k \leftarrow 1$
  $\hat{S}_\alpha \leftarrow \emptyset$
  repeat ▷ Outer loop
    $c \leftarrow 0$
    while $A(k, c)$ is feasible **do** ▷ Inner loop
      $\hat{S}_\alpha \leftarrow \hat{S}_\alpha \cup \{(k, c)\}$
      $c \leftarrow c + 1$
    end while
    $k \leftarrow k + 1$
  until $A(k, 0)$ is not feasible ▷ Stopping criterion
  return $\hat{S}_\alpha$
end procedure

Algorithm 5 The algorithm that obtains the final search space S_α by cropping $\hat{S}_\alpha$.

Require: $\hat{S}_\alpha, \xi_{Time}, \xi_{Energy}, \bar{w}$
Ensure: S_α
procedure SEARCH SPACE - CROPPING
  $A^* \leftarrow$ architecture in $\hat{S}_\alpha$ with max parameters
  $\bar{t} \leftarrow$ eval_time(A^*) ▷ Time upper bound
  $\bar{e} \leftarrow \bar{t}\bar{w}$ ▷ Energy upper bound
  $k \leftarrow 1$
  $S_\alpha \leftarrow \emptyset$
  $stop \leftarrow$ false
  repeat
    $c \leftarrow 0$
    while $(k, c) \in \hat{S}_\alpha$ and not $stop$ **do**
      if $|S_\alpha|\bar{t} \leq \xi_{Time}$ and $|S_\alpha|\bar{e} \leq \xi_{Energy}$ **then**
        $S_\alpha \leftarrow S_\alpha \cup \{A(k, c)\}$
      else
        $stop \leftarrow$ true
      end if
      $c \leftarrow c + 1$
    end while
    $k \leftarrow k + 1$
  until $stop$ or no more architectures with current k in $\hat{S}_\alpha$
  return S_α
end procedure

parameters from $\hat{S}_\alpha$ and empirically measuring its evaluation time (training and validation) on the gateway. The upper bound on energy consumption, $\bar{e}$, is then calculated by multiplying $\bar{t}$ by the gateway's maximum power consumption, $\bar{w}$, which is provided as an input.

Next, starting with the smallest solution in $\hat{S}_\alpha$, $(k = 1, c = 0)$, the algorithm iteratively adds architectures to S_α in the same order they were added to $\hat{S}_\alpha$. This process terminates when the size of S_α reaches a threshold where it is estimated that the gateway would be unable to complete the NAS while satisfying the constraints on ξ_{Time} and ξ_{Energy}.

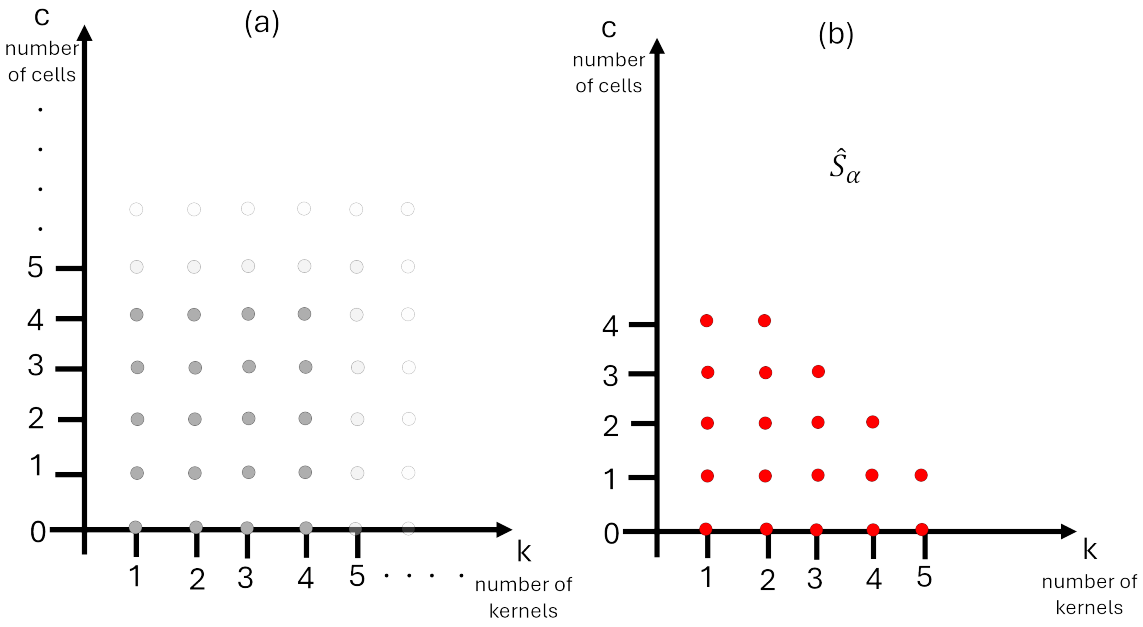
Figure 5.2 illustrates the process of generating S_α. It begins with a hypothetical search space of all possible architectures $A(k, c)$ (Fig. 5.2a). Algorithm 4 then creates an extensive, but provisional, search space $\hat{S}_\alpha$ (Fig. 5.2b). Finally, Algorithm 5 produces the final, operational search space S_α by cropping $\hat{S}_\alpha$ (Fig. 5.2c).

5.3.2 Search Strategy

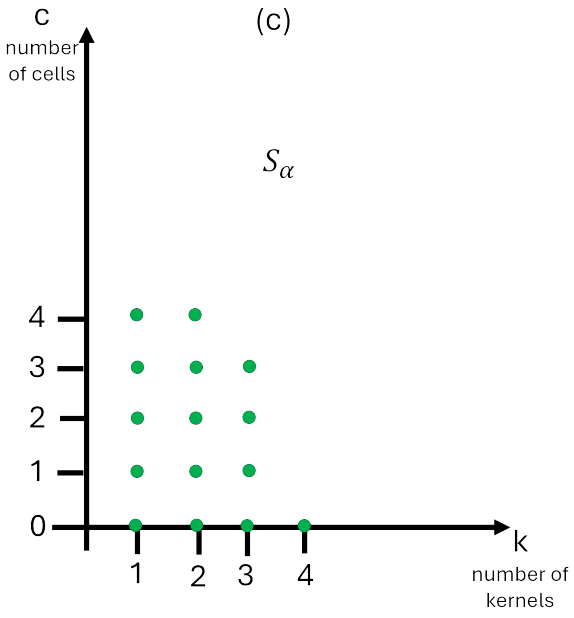
The proposed search strategy is a bi-level, derivative-free optimization technique. It aims to find the best neural architecture within the search space described in Section 5.2.2 for a given classification task on a specified microcontroller, with a preference for smaller solutions. The inner level of the technique (Algorithm 6, while loop in line 7) determines the optimal number of building cells to stack for a given number of kernels in the base cell, by adjusting the parameter c. The outer level (Algorithm 6, repeat-until loop in line 5) modifies the parameter k to identify the best combination of k and c.

Specifically, the inner level identifies c^*, the optimal value of c for a fixed $\bar{k}$, by incrementally adding building cells until the solution is no longer feasible. The outer level proposes a new candidate $\bar{k}$ by adding a variable increment, $\lfloor k2^{-\beta} \rfloor$ (with β initialized to zero), to the previously confirmed value k, and then invokes the inner level. If the newly found candidate solution is superior to the previously confirmed one, it is confirmed, and the algorithm proceeds. If not, β is incremented by one, reducing the variable increment, and a new $\bar{k}$ is proposed. Eventually, the increment becomes zero, causing the newly proposed value to be the same as the last confirmed one. When this condition is met, the algorithm terminates and returns the last confirmed candidate. The evaluation metric used is generalization capability, measured as validation accuracy on a validation set.

Figure 5.3 illustrates a potential execution of the search strategy on the search plane. In this visualization, parameter changes that result in a better candidate are indicated by green arrows, while those leading to a worse candidate are marked with red arrows.



(a) Hypothetical search space **(b) Extensive search space**



(c) Actual search space

Figure 5.2: Example of the process that leads to the search space S_α

Algorithm 6 Proposed search strategy.

procedure PROPOSED OPTIMIZATION TECHNIQUE
  $k \leftarrow 1, c \leftarrow 0$ ▷ Starting point
  $\beta \leftarrow 0, \gamma \leftarrow 0$
  $\bar{k} \leftarrow k$
  repeat ▷ Outer loop
    $\bar{c} \leftarrow 0, c^* \leftarrow 0$
    while $(\bar{k}, \bar{c}+1) \in S_\alpha$ **do** ▷ Inner loop
      if $f(\bar{k}, \bar{c}+1) > f(\bar{k}, c^*)$ **then**
        $c^* \leftarrow \bar{c}+1$ ▷ Update of best c
      end if
      $\bar{c} \leftarrow \bar{c}+1$
    end while
    if $f(\bar{k}, c^*) > f(k, c)$ **then**
      $k \leftarrow \bar{k}, c \leftarrow c^*$ ▷ Candidate confirmation
    else
      $\gamma \leftarrow 1$ ▷ Start k increment reduction
    end if
    $\beta \leftarrow \beta + \gamma$ ▷ Optional k increment reduction
    $\bar{k} \leftarrow k + \lfloor 2^{-\beta} k \rfloor$ ▷ Variable increment of k
  until $\lfloor 2^{-\beta} k \rfloor = 0$ ▷ Stopping criterion
end procedure

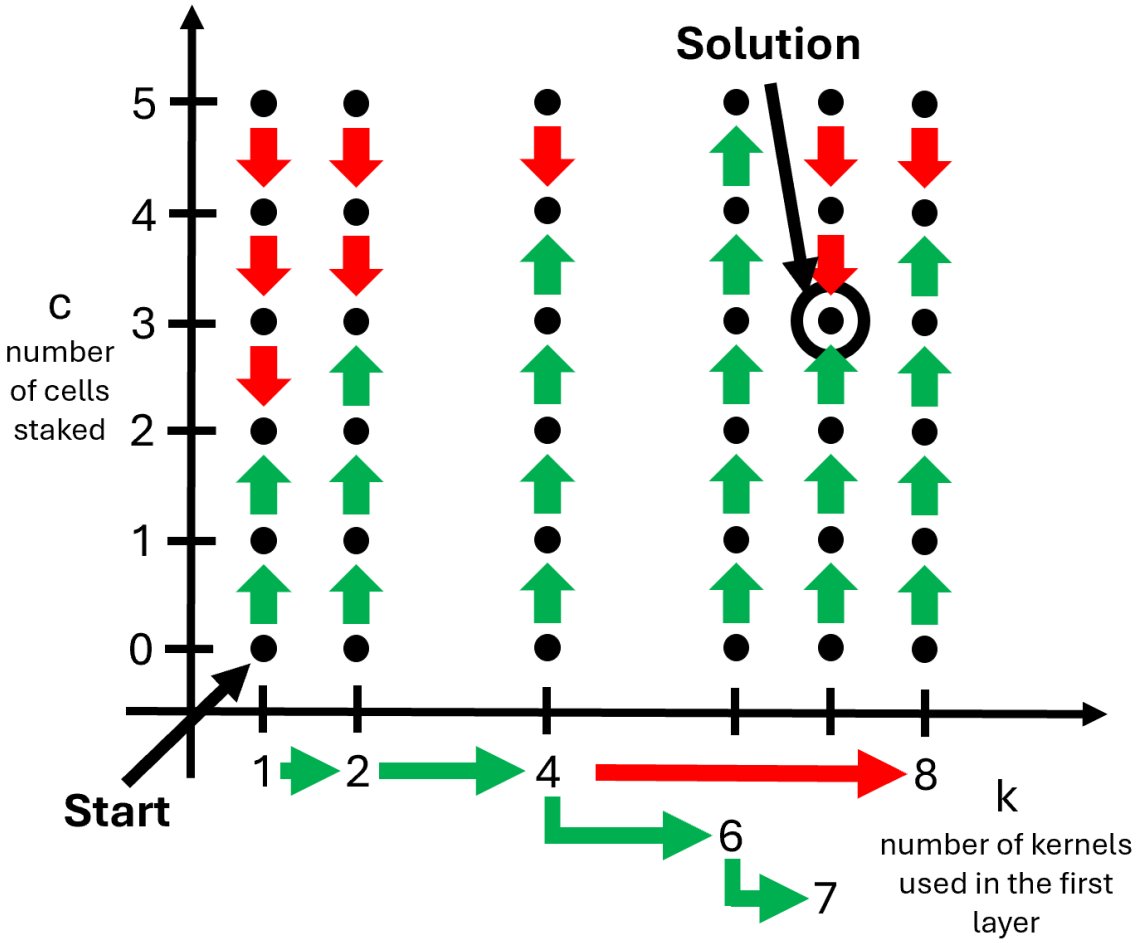


Figure 5.3: A possible run of the search strategy plotted on the search plane.

The search begins at the smallest admissible solution, $A_0 = A(k = 1, c = 0)$, and explores the corresponding c-axis to find the first candidate, $A_1 = A(\bar{k} = 1, c^* = 2)$. Since A_1 performs better than A_0, it is confirmed. The search then proceeds to explore the k-axis by doubling the confirmed k value, resulting in $\bar{k} = 2$. The c-axis is explored again, yielding the second candidate, $A_2 = A(\bar{k} = 2, c^* = 3)$. This new candidate is also confirmed as it outperforms A_1, and the transition along the k-axis is marked with a green horizontal arrow.

The algorithm continues in this manner until performance along the k-axis ceases to improve, which occurs in this example at $\bar{k} = 8$, indicated by a red arrow. The corresponding candidate, $A_4 = A(\bar{k} = 8, c^* = 4)$, is not confirmed, leaving $A_3 = A(k = 4, c = 4)$ as the confirmed candidate.

This halt in performance improvement triggers the k-increment reduction by setting $\gamma = 1$. A new k value, $\bar{k} = 6$, is then proposed. The corresponding candidate found by exploring the c-axis, $A_5 = A(\bar{k} = 6, c^* = 5)$, performs better than the previously confirmed candidate A_3 and is therefore confirmed, marked with a horizontal green arrow.

The next candidate found, $A_6 = A(k = 7, c = 3)$, improves upon the last confirmed solution and is also marked with a green arrow. The final k value proposed is 7, which is identical to the last confirmed solution because the variable increment has reached zero. At this point, the algorithm terminates, returning A_6 as the final result.

Even though the choice of solving a set of simpler optimization problems instead of a complex one to reduce the computational burden recalls dynamic programming, the absence of overlap in the set, i.e., the absence of a task repeated multiple times with the same inputs and always giving the same output (the training of a neural network is not deterministic and the algorithm does not ask to train the same candidate a lot of times), does not allow effective memoization of solutions preempting a formalization within that framework [106]. Instead, the feature of making the locally optimal choice at each step, i.e., choosing the number of building cells c locally maximizing the validation accuracy at each step, makes the proposed search strategy fall under the umbrella of greedy optimization [107].

5.4 Experimental Validation

5.4.1 Experimental Setup

This chapter details the results of four distinct experiments:

The first experiment shows that the proposed HW-NAS can generate compact neural network architectures customized for ultra-low-power microcontrollers with diverse hardware specifications.

The second experiment compares the proposed HW-NAS against state-of-the-art methods, demonstrating its ability to produce cutting-edge tiny CNNs optimized for ultra-low-power computing platforms.

The third experiment establishes that the proposed HW-NAS can effectively adapt the cost of its search procedure to meet the execution time and energy budget constraints of an embedded device.

The fourth experiment validates that the proposed HW-NAS can operate on different embedded devices.

Gateways The proposed HW-NAS was executed on three different single-board computers that can function as IoT gateways: the Raspberry Pi 4 (RPi 4), Raspberry Pi 3 (RPi 3), and Raspberry Pi Zero 2 (RPiZ 2). Table 5.2 summarizes the key specifications of these boards, including SoC details, available RAM, and measured peak power consumption. The maximum power consumption of each device was evaluated using the "stress-ng" utility from Debian. The command line “stress-ng –cpu 0 –cpu-method all –vm 0 –vm-bytes 100% –vm-method all –verify -t 1m -v” was used to force full utilization of all CPU cores and main memory. A power meter connected directly to the power supply measured the maximum power consumption and was also used to monitor energy consumption and execution time during the HW-NAS process.

Table 5.2: Hardware features of the embedded devices targeted in this work.

Embedded Device Raspberry Pi	**SoC Broadcom**	**RAM [GiB]**	**Power Consumption [W]**
4 Model B	BCM2711	4	5.6
3 Model B	BCM2837	1	4.3
Zero 2	BCM2710A1	0.5	2.8

Sensor Nodes Three different microcontrollers from the ST Microelectronics ultra-low-power product line were chosen as the computing units for the sensor nodes: the

L010RBT6, U083RCT6, and L412KBU3. Table 5.3 presents the available RAM, Flash memory, and CoreMark score for each microcontroller.

Table 5.3: Hardware features of the MCUs targeted in this work.

MCU STM32	RAM [kiB]	Flash [kiB]	CoreMark score
ultra-low-power line			
L010RBT6	20	128	75
U083RCT6	32	256	134
L412KBU3	40	128	273

NAS The proposed HW-NAS includes a standalone process to monitor execution time and estimate energy consumption in real-time. This backup mechanism prevents the HW-NAS from continuing if either the execution time limit or the energy budget is exceeded. This is crucial because non-real-time Operating Systems (OS) like Debian or Windows, which are common on IoT gateways, can introduce unpredictable system overheads that are difficult to estimate. Consequently, the search space S_α produced by Algorithm 5 might be too large to fully explore within the constraints ξ_{Time} and ξ_{Energy}. Therefore, the software implementation is designed to output an architecture even if a complete exploration of the search space S_α is not possible.

Hyper-parameters The maximum validation accuracy of a candidate architecture is determined after three epochs of training, using a validation split of 30% of the training data, a batch size of 16, and a learning rate of 1×10^{-3}. The Adam optimizer [108] is employed for this evaluation. For time series data, architectures were evaluated after fifty epochs of training.

Deployment Both Quantization Aware Training (QAT) and Post Training Quantization (PTQ) are used to produce 8-bit quantized models. Memory characteristics, including Flash and RAM occupancy, were measured using the STM utility "stm32tflm," which

is part of the X-CUBE-AI software package. This tool provides precise measurements for running the resulting architecture on STM32 microcontrollers with the TFLite Micro runtime. Additionally, the proposed method calculates MAC operations and architectural parameters, represented as (k, c).

The architectures selected by the proposed strategy are subsequently trained for 100 epochs (500 for time series), with a validation split of 10%, a batch size of 128, and a learning rate of 1×10^{-2}. The Adam optimizer is used, and validation-based checkpointing is implemented. The training data is augmented through random horizontal flips and random rotations in the range $[-\frac{2}{5}\pi, \frac{2}{5}\pi]$.

The reported latency is measured on the NUCLEO-L432KC board using the 'validation on target' feature of X-CUBE-AI. X-CUBE-AI version 8.1.0 and STM32CubeIDE version 1.17.0 were used to program the boards.

Datasets Four datasets were used for the experiments:

The Visual Wake Words dataset [41], a tinyML benchmark [46], contains 123,000 images for detecting the presence of a person.

The CIFAR-10 dataset, another tinyML benchmark [46], consists of 60,000 32x32 color images across 10 classes, with 6,000 images per class.

The Melanoma Skin Cancer dataset [109], representing an HIoT application, includes 10,000 images for classifying samples as malignant or benign.

The Case Western Reserve University (CWRU) dataset [110], for an IIoT application, comprises accelerometer data from rolling bearings with various induced faults.

The test accuracy of the final architectures is evaluated on the public test set of the respective dataset. For the Visual Wake Words dataset, only 10% of the training split was used for the HW-NAS process, while the entire training split was used for the final training of the resulting architecture.

5.4.2 Results

5.4.2.1 Adaptability to ultra-low-power microcontrollers

The first experiment is designed to demonstrate that the proposed HW-NAS is capable of generating tailored architectures for different ultra-low-power computing platforms, which

are representative of sensor nodes in IoT applications. For this experiment, it is assumed that the HW-NAS can be executed on a standard computing platform, meaning there are no constraints on its available resources.

Table 5.4 displays the results, organized into three subtables corresponding to each dataset. Each subtable details, for a given target microcontroller, the configuration of the selected architecture, the hardware requirements of the deployed model (RAM, Flash memory, and MAC operations), the test accuracy, and the latency for a single inference.

Table 5.4: Performance of the proposed HW-NAS when targeting ultra-low-power computing platforms.

Target MCU STM32	**Arch. (k,c)**	**RAM [kiB]**	**Flash [kiB]**	**MAC [MM]**	**Test Acc. [%]**	**Lat. [ms]**
Visual Wake Words (50x50)						
L010RBT6	(3,4)	19	10.8	0.4	71	42
U083RCT6	(5,5)	24.5	22.7	0.9	75.2	63.2
L412KBU3	(8,3)	31	18.8	2	78.3	79.1
Melanoma Skin Cancer (50x50)						
L010RBT6	(3,5)	18.5	8.1	0.4	84.2	43.2
U083RCT6	(6,4)	26.5	20.4	1.3	88.5	67.3
L412KBU3	(9,4)	34	35.7	2.6	90	129.8
CIFAR10 (32x32)						
L010RBT6	(6,5)	14	28.7	0.5	63.3	31.6
U083RCT6	(9,4)	17	36	1.1	67.1	60
L412KBU3	(13,4)	21.5	65.2	2.1	70.9	97.5

The experimental results confirm that the characteristics of the selected architectures scale with the resources available on the target platforms. As the resource capacity increases, the HW-NAS selects architectures that demand more RAM, Flash memory, and MAC operations. This increase in resource utilization corresponds to an improvement in test accuracy, along with an expected increase in latency.

5.4.2.2 State of the art comparison

The proposed method is benchmarked against four state-of-the-art NAS techniques: MCUNet [10], Micronets [37], ColabNAS [47], and NanoNAS [74]. The first two were

designed for high-performance MCUs, whereas the latter two were developed for ultra-low-power MCUs. The comparison is performed using the Visual Wake Words dataset [41], a standard benchmark for tiny machine learning visual applications [44].

To ensure a fair comparison, Table 5.5 includes the smallest models selected by NAS methods targeting high-performance MCUs [10, 37] and the largest models selected by NAS methods targeting ultra-low-power MCUs [47, 74]. For the method proposed in this work, the largest architecture from Section 5.4.2.1 was chosen. The table presents the test accuracy, RAM usage, Flash memory footprint, and MAC operations for each NAS.

The results indicate that MCUNet produces the network with the highest test accuracy, but it is also the largest architecture in terms of RAM, Flash, and MAC operations. In contrast, the proposed method achieves the second-highest test accuracy while having the lowest Flash memory usage and the second-lowest RAM and MAC operation counts. ColabNAS places third in test accuracy, with larger RAM and Flash requirements. NanoNAS is fourth in accuracy, featuring the lowest RAM occupancy and MAC operations. Micronets ranks last, delivering the lowest test accuracy with an architecture only slightly smaller than MCUNet's in terms of RAM and Flash.

Table 5.5: Comparison with existing HW-NAS on VWW dataset.

Work	**Acc [%]**	**RAM [kiB]**	**Flash [kiB]**	**MAC [MM]**
MCUNet [10]	87.4	168.5	530.5	6
Micronets [37]	76.8	70.5	273.8	3.3
ColabNAS [47]	77.6	31.5	20.83	2
NanoNAS [74]	77	28.5	23.7	1.3
Proposal	78.3	31	18.8	2

5.4.2.3 Running HW-NAS on energy and time constrained devices

This experiment is designed to show that the proposed HW-NAS employs an adaptive strategy that can conform to a given resource budget. A Raspberry Pi Zero 2 was chosen as the gateway for this experiment, with an STM32L412KBU3 serving as the sensor node.

Table 5.6: Performance of the proposed HW-NAS when running on Raspberry Pi Zero 2 with different energy/time budgets, with STM32L412KBU3 as a target.

NAS Details				Resulting Architecture				
Budget [Wh][time]	**Cost [Wh][time]**	S_α **[%]**	**Expl. [%]**	**RAM [kiB]**	**Flash [kiB]**	**MAC [MM]**	**Acc. [%]**	**Lat. [ms]**
16.5 - 9:51	16.5 - 9:51	100	51	28.5	16.1	1.6	77.8	87.4
11.0 - 6:34	11.0 - 6:30	33	98	21.5	13.1	0.7	73.1	43.3
5.50 - 3:17	5.41 - 3:17	15	95	18.5	7.2	0.1	66	24.6

First, the extensive search space $\hat{S}_\alpha$ for the STM32L412KBU3 was generated using Algorithm 4. Then, the energy and time budgets required by the Raspberry Pi Zero 2 to process the full search space ($S_\alpha = \hat{S}_\alpha$) were empirically determined. It was found that an energy budget of 16.5 Wh and a time budget of 9 hours and 51 minutes (hh:mm) were sufficient to avoid any cropping of $\hat{S}_\alpha$ (as per Algorithm 5).

Table 5.6 presents the results of three experiments where the proposed HW-NAS was run on the Raspberry Pi Zero 2 with different energy and time budgets. The first row shows the outcome with the full budget of 16.5 Wh and 9:51. The second column reports the actual cost of the search procedure, the third column shows the final size of S_α as a percentage of $\hat{S}_\alpha$, and the fourth column indicates the percentage of S_α that was explored by the search strategy in Algorithm 6. The second part of the table provides details of the generated architecture: RAM, Flash, MACs, test accuracy on the VWW dataset, and latency on the STM32L412KBU3. The subsequent rows show the results with the budget reduced to two-thirds (11.0 Wh, 6:34) and one-third (5.50 Wh, 3:17) of the maximum.

The table confirms that, as expected, only the full budget of 16.5 Wh and 9:51 allowed the search procedure to operate on the entire search space $\hat{S}_\alpha$ without cropping, resulting in S_α being 100% of $\hat{S}_\alpha$. The search strategy explored 51% of this space to find the optimal architecture. In the other two experiments, the exploration was halted due to resource depletion. In the second experiment, the search was stopped when the energy threshold was reached, with four minutes of time remaining. The search space S_α was reduced to 33% of $\hat{S}_\alpha$, and due to its smaller size, the strategy was able to explore 98% of it. In the final experiment, the search was stopped when the time limit was reached. Here, S_α was only 15% of $\hat{S}_\alpha$, and the strategy explored 95% of this much smaller space.

Naturally, the three experiments produced three different architectures. The footprint and latency of the selected architecture decrease as the budget for running the HW-NAS

is reduced. Correspondingly, the generalization performance also deteriorates. Overall, these results demonstrate that the proposed HW-NAS can effectively adapt to the available budget by adjusting the portion of the search space it explores.

5.4.2.4 Effect of different platform on HW-NAS execution

This experiment is intended to show that the proposed HW-NAS can adapt its search strategy to the characteristics of the specific gateway. For this purpose, three different devices were used as gateways: Raspberry Pi Zero 2, Raspberry Pi 3 Model B, and Raspberry Pi 4 Model B. The STM32L412KBU3 was again used as the sensor node.

Table 5.7 shows the results obtained by running the HW-NAS on the three gateways with an identical energy and time budget: 16.5 Wh and 9:51. As established previously, this budget is sufficient for the Raspberry Pi Zero 2 to explore the full search space $\hat{S}_\alpha$ defined by the STM32L412KBU3.

The table details for each experiment: the gateway used, the actual cost of the search, the final size of S_α as a percentage of $\hat{S}_\alpha$, and the percentage of S_α that was explored. The second part of the table provides information on the resulting architecture, including its resource usage, accuracy, and latency.

The results reveal that when the Raspberry Pi 3 was used as the gateway, the search process terminated after 5 hours and 4 minutes due to energy depletion. The Raspberry Pi 3 is generally less energy-efficient than the Raspberry Pi Zero 2. As a result, the HW-NAS procedure had to crop $\hat{S}_\alpha$, with the final S_α being 38% of the extensive search space. In contrast, the HW-NAS was able to complete its run on the Raspberry Pi 4 without exhausting the budget. In this case, S_α was 100% of the extensive search space, requiring no cropping. On the Raspberry Pi 4, the HW-NAS explored 57% of $\hat{S}_\alpha$ in 3 hours and 25 minutes, whereas the Raspberry Pi Zero required 9 hours and 51 minutes to explore 51% of the same space.

These findings demonstrate that the proposed HW-NAS effectively adapts to different execution platforms. It can adjust the size of S_α based on the available budget and the gateway's characteristics. Furthermore, the search strategy (Algorithm 6) is capable of adequately exploring both small and large search spaces.

5.4.2.5 Time Series

This experiment aims to demonstrate that the proposed technique can also be effectively applied to time-series data, achieving state-of-the-art results on the CWRU dataset [110].

Table 5.7: Performance of the proposed HW-NAS with STM32L412KBU3 as a target platform, maintaining a fixed resource budget while varying the execution platforms

RPi	NAS Details			Resulting Architecture				
	Cost [Wh][time]	S_α **[%]**	**Expl. [%]**	**RAM [kiB]**	**Flash [kiB]**	**MAC [MM]**	**Acc. [%]**	**Lat. [ms]**
Z2	16.5 - 9:51	100	51	28.5	16.1	1.6	77.8	87.4
3	16.5 - 5:04	38	98	21	9.5	0.6	72.7	44.5
4	14.2 - 3:25	100	57	35	52.1	2.6	75.9	125.7

A Raspberry Pi 4 was used as the gateway and an STM32-L010RBT6 as the target sensor node. The algorithm was run without any energy or time constraints. The resulting network was compared with the state-of-the-art work from Chen et al. [111].

Table 5.8: Resulting Architecture VS Reference Architecture

Work	Search Cost		Resulting Architecture					
	Energy [Wh]	**Time [hh]:[mm]**	**(k, c)**	**RAM [kiB]**	**Flash [kiB]**	**MAC [MM]**	**Test Acc. [%]**	**Latency [ms]**
Proposal	6.4	1:52	(6, 4)	13.5	12.9	0.6	99.5	34
[111]	n/a	n/a	n/a	66.5	163.4	0.2	99.3	38.2

Table 5.8 presents the results of this experiment. It compares the neural network generated by the proposed HW-NAS with the one used in [111]. The table includes details on RAM and Flash usage, MAC operations, test accuracy on the CWRU dataset, and latency as measured on the STM32-L010RBT6 device.

The HW-NAS successfully produced a tiny CNN that requires 4.9 times less RAM and 12.7 times less Flash memory than the architecture in [111]. Remarkably, this compact CNN achieved a higher test accuracy while also reducing latency. The resulting architecture is capable of running on an STM32-L010RBT6, an ultra-low-power MCU, making it well-suited for IIoT sensor nodes.

The search procedure was conducted on a Raspberry Pi 4 and completed in 1 hour and 52 minutes, with an energy consumption of 6.4 Wh. The average CPU usage per core was 67.9%, and the average RAM usage was 593 MB (out of 4 GB available). These

results indicate that the proposed method is compatible with existing services running on deployed gateways in IoT networks.

5.5 Conclusion

With the proposed method, NNs can be automatically designed at the edge, ensuring that data never leaves the collection site. This capability enables the use of NNs in privacy-sensitive applications. For instance, customized and hardware-friendly NNs can be created for personalized healthcare within the HIoT framework, keeping patient data securely within the hospital. Similarly, it can be applied to IFD and quality control in industrial plants under the IIoT framework, preserving valuable industrial secrets that could otherwise be revealed by the data.

An extensive experimental campaign has demonstrated that the proposed HW-NAS a) is capable of running on embedded devices that are commonly used as gateway nodes in IoT networks and b) delivers state-of-the-art results when targeting ultra-low-power MCUs as the computing platform for sensor nodes.

To the best of the authors' knowledge, the existing literature does not offer other approaches for designing an HW-NAS that allows the search process to run on an embedded device. The sole exception is a recent publication by the authors of this work [74]. However, the HW-NAS presented here improves upon the method in [74] by incorporating the ability to dynamically adapt to the gateway's characteristics, thereby enabling its practical use in diverse IoT networks. This approach may further stimulate the development of lightweight HW-NAS methods that can operate on resource-constrained devices, opening up new application domains. Future research could, for example, explore training-free evaluation techniques to further reduce the search cost.

Acknowledgment

Project funded under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.1 - Call for tender No. 1409 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-bAsed weaRable exoskeletoNs for assistance in daily life" – CUP: D53D23016200001, J53D23014090001

- Grant Assignment Decree No. 1383 adopted on September 01, 2023 by the Italian Ministry of University and Research (MUR).

CHAPTER 6

General Conclusion

This dissertation addressed a critical bottleneck in the widespread adoption of deep learning at the edge: the high computational cost and resource requirements of Hardware-Aware Neural Architecture Search (HW-NAS). The central thesis was that by designing lightweight search spaces and novel, low-cost search strategies, it is possible to create HW-NAS frameworks that are not only accessible to a broader range of users but can also be executed directly on resource-constrained edge devices. This approach enables a new paradigm of privacy-preserving, on-device neural network design, tailored specifically for the hardware and data unique to each application.

6.1 Summary of Contributions

The research presented in this dissertation demonstrated a clear, progressive path toward fully autonomous, on-device HW-NAS. The journey began by democratizing access to HW-NAS and systematically reducing its computational footprint until it could operate within the stringent constraints of an IoT gateway.

The first major contribution, **ColabNAS**, established the feasibility of low-cost HW-NAS. By leveraging a search strategy inspired by Occam's razor, it was shown that state-of-the-art, task-specific Convolutional Neural Networks (CNNs) could be generated in a matter of hours using freely available online GPU services. This work significantly lowered the barrier to entry for researchers and practitioners, proving that effective HW-NAS did not necessitate vast computational resources. Furthermore, ColabNAS demonstrated its ability to produce models that were significantly more compact than those derived from common techniques like Transfer Learning, making it a valuable tool for developing lightweight applications.

Building on this foundation, the second contribution, **NanoNAS**, took the crucial step of eliminating the dependency on GPUs entirely. This work demonstrated that for the design of highly compact CNNs targeting ultra-low-power microcontrollers, the search

procedure could be successfully executed on a standard laptop CPU. By incorporating a constraint on the memory of the device performing the search, NanoNAS introduced the concept of a self-aware search platform. The effectiveness of a coarse early-stopping criterion was empirically validated, proving that the search cost could be dramatically reduced without compromising the ability to identify optimal architectures. This contribution moved HW-NAS from the cloud to local, general-purpose hardware.

The final and culminating contribution, **GatewayNAS**, realized the ultimate goal of this dissertation: executing HW-NAS on an embedded IoT gateway. This framework introduced a novel, dual-level hardware awareness, considering the constraints of both the target sensor node for inference and the gateway for the search process itself. A key innovation was an adaptive mechanism capable of tailoring the search space to the gateway's available time and energy budget, ensuring that the design process could be completed under real-world operational constraints. The capability of GatewayNAS was extended to time-series data, achieving state-of-the-art results in an industrial fault diagnosis task. This work enables a complete, privacy-preserving workflow where sensor data is used locally to generate custom, hardware-optimized neural networks without ever leaving the premises.

6.2 Broader Impact

The contributions of this dissertation have several significant implications for the field of embedded AI and the Internet of Things.

Democratization of AI Hardware Design. By drastically lowering the computational and financial costs associated with HW-NAS, this work makes the automated design of efficient neural networks accessible to a wider audience, including small enterprises, academic labs, and individual developers who lack access to large-scale computing clusters.

Enabling Privacy-Preserving AI. The ability to run the entire NAS process on a local gateway is a critical enabler for applications involving sensitive data, such as in healthcare (HIoT) and industrial settings (IIoT). This eliminates the need to transfer private patient records, biometric data, or proprietary industrial process data to third-party cloud services, addressing major privacy and security concerns that currently hinder AI adoption in these sectors.

Towards Sustainable AI. The methodologies developed in this thesis contribute to the growing field of Green AI. By creating highly efficient, task-specific models and utilizing

low-power hardware for the design process itself, this research promotes a more sustainable approach to AI development and deployment. Reducing the energy consumption of both the final inference application and the design process is a fundamental step toward mitigating the environmental impact of artificial intelligence.

6.3 Limitations and Future Directions

While this dissertation successfully demonstrated the viability of lightweight, on-device HW-NAS, there are limitations that present opportunities for future research.

First, the proposed methods are specifically tailored for generating tiny neural networks for ultra-low-power microcontrollers. While they achieve state-of-the-art performance within this domain, they may not produce the most accurate models when targeting high-performance hardware with more generous resource budgets. In such cases, traditional, more computationally intensive NAS methods might still yield superior results in terms of generalization performance. Future work could explore hybrid approaches that scale the complexity of the search strategy based on the available resources of the search platform.

Second, the efficiency of the search process, particularly on embedded gateways, could be challenged by very large datasets. Although the use of partial datasets for the search was effective, managing data loading and preprocessing on memory-constrained devices remains a potential bottleneck. Research into more efficient data handling techniques or the integration of data-subset selection strategies could further enhance the practicality of on-device NAS.

Finally, the evaluation of each candidate architecture currently relies on a brief training period. While shown to be effective, this step still constitutes the most time-consuming part of the search. A promising avenue for future investigation is the exploration of training-free evaluation techniques. By developing proxy metrics that can accurately predict a candidate architecture's final performance without any training, the search cost could be reduced by orders of magnitude, enabling near-instantaneous design of neural networks at the extreme edge.

Furthermore, while this work prioritizes accuracy and efficiency, the "black box" nature of the resulting CNNs warrants further investigation, particularly for sensitive applications like healthcare (e.g., Melanoma detection). Future work should incorporate Explainable AI (XAI) techniques, such as Grad-CAM [112] or Layer-wise Relevance Propagation (LRP) [113], to ensure the models are learning robust, domain-relevant features rather

than relying on shortcut learning [114] or artifacts. Verifying that the networks leverage genuine pathological features rather than spurious correlations is essential for establishing clinical trust in automatically designed architectures.

Appendix

APPENDIX A

Wake Vision: A Tailored Dataset and Benchmark Suite for TinyML Computer Vision Applications

The contribution presented in this appendix has been previously published as an arXiv preprint [115]. The work was part of my fellowship at the Harvard John A. Paulson School of Engineering and Applied Sciences. I principally curated the benchmark results about TinyML models, and the competitions.

A Introduction

Ultra-low power machine learning (TinyML) has emerged as an important technology to enable efficient ML deployments by co-locating models with sensors on microcontrollers (MCUs) [44, 116, 46, 81]. Although this approach dramatically reduces energy consumption and deployment costs, it imposes severe constraints that fundamentally differ from standard machine learning scenarios. For instance, the models must often fit within hundreds of kilobytes of memory, nearly four orders of magnitude less than common model sizes on smartphones[37]. These extreme hardware-imposed constraints necessitate compact and efficient TinyML models, rendering traditional ML benchmarks and datasets unsuitable. For example, ImageNet [22] models must output a thousand classes, which is infeasible on MCUs and even smartphone models struggle to support [41].

Existing TinyML datasets such as VWW [41] and Google Speech Commands [42] target simpler applications, requiring support for binary classification and tens of classes, respectively. While these datasets are valuable, their limited scale and label error rates make them unsuitable for pushing the frontier of TinyML research and training production-grade TinyML models. This creates a challenge where the development of robust TinyML systems is hindered by existing datasets being either too complex for tiny architectures or too limited in scale and diversity. However, this challenge extends beyond simply

collecting more data; it requires systematic methodologies for rapidly creating, curating, and evaluating datasets that meet TinyML's unique constraints at scale.

To address this gap, an automated dataset generation pipeline is presented. This pipeline is designed for creating and benchmarking large-scale binary classification datasets suitable for TinyML and incorporates several algorithmic innovations: intelligent fusion of multiple label sources, confidence-aware filtering strategies, automated label correction techniques, and systematic generation of fine-grained benchmark suites. TinyML dataset automation is not merely a convenience but a necessity. Manual labeling approaches are prohibitively expensive (a manual validation of 70K images, for example, cost $7,000, which would extrapolate to $600,000 for a 6M image dataset) and suffer from inconsistency across labelers. The presented dataset generation pipeline derives datasets from Open Images v7 [117, 118], which enables created datasets to be licensed under a permissive CC-BY 4.0 license to facilitate broad adoption.

The effectiveness of this pipeline is demonstrated through the creation of Wake Vision, a large-scale open-source binary classification dataset. Wake Vision validates the described methodology using person detection, which serves as the quintessential vision use case for TinyML [46]. It is an ideal testbed because it enables several important applications ranging from occupancy detection [119] and smart HVAC/lighting [120] to acting as an always-on 'wake model' in larger ML systems [121]. As shown in Table A.1, Wake Vision advances the state-of-the-art TinyML datasets by providing almost 6M images, close to 100× more than the current leading dataset VWW [41], while achieving up to a 6.6% accuracy improvement.

Wake Vision is used to derive insights about TinyML-specific design principles: small models benefit more from smaller, high-quality training sets compared to noisier, larger sets. However, if designed carefully, a two-stage training strategy (pre-training on Wake Vision Large followed by fine-tuning on Wake Vision Quality) can leverage both scale and quality to achieve optimal performance (C.1). Furthermore, to enable rigorous evaluation of real-world capabilities, a comprehensive suite of five fine-grained benchmarks was developed (B.3). This benchmarking framework addresses a gap in current TinyML assessment methods, revealing that Wake Vision trained models outperform VWW trained models on 13 of 16 fine-grained sets, demonstrating robust performance beyond aggregate metrics alone.

Beyond person detection, it is shown that the dataset generation pipeline generalizes to all 9.6K trainable classes provided by Open Images v7, enabling the systematic creation of more production-grade datasets across diverse TinyML applications (B.5). Evidence

is provided with two additional domains for TinyML deployments: bird detection for smart nests [122] and automated feeders [123] (12× more images with label error rates improving from 6.4% to 4.8%), and car detection for parking monitoring [124] and blind-spot systems [125] (27× more images with error rates improving from 6.6% to 0.6%). Additionally, public challenges are hosted for long-term, community-driven enhancements to Wake Vision. The dataset, benchmarks, code, and models, along with past and upcoming challenges, can be found at `https://wakevision.ai`.

A.1 Background and Related Work

This section reviews the state of the art in dataset generation for TinyML, the role of person detection in such applications, and existing datasets used for this task.

A.1.1 Generating Binary Classification Datasets for TinyML

Automatically generated datasets can significantly lower costs associated with deploying TinyML systems. Prior to this work, the primary way to generate binary classification datasets for TinyML applications was through the code used to create VWW [41], which can be applied to other MS-COCO [38] classes. While the same capability is offered for Open Images, Wake Vision goes beyond dataset generation, offering customizable curation properties comprising of label correction, fine-grained benchmark suite generation, and community-driven enhancements through competitions, addressing the whole dataset life-cycle.

A.1.2 Person Detection in TinyML Applications

Person detection has emerged as the canonical computer vision use case for TinyML systems [44, 46]. In the context of resource-constrained computing, TinyML systems demand tasks that carefully balance computational feasibility with real-world utility. Person detection represents an optimal compromise. It is sufficiently lightweight to be implemented on embedded microcontrollers while maintaining enough discriminative power for practical applications. In particular, person detection serves as a critical activation mechanism in resource-constrained systems. These lightweight models support always-on deployment on low-powered MCUs to selectively "wake" power-intensive sensors, processors, and larger, more capable ML models upon detecting human presence [121]. This architectural pattern enables substantial energy savings while preserving privacy through on-device

Table A.1: A comparison of Wake Vision for person detection and image classification datasets.

Dataset	Total Images	# of Person Images Train	Validation	Test	Fine-Grained Filtering	Suitable for TinyML
Ours – WV (Quality)	**1,322,574**	**624,115**	**9,291**	**27,881**	✓	✓
Ours – WV (Large)	**5,760,428**	**2,880,214**	–	–	✗	✓
Visual Wake Words [41]	123,287	36,000	3,926	19,107	✗	✓
CIFAR-100 [12]	60,000	2,500	–	500	✗	✗
PASCAL VOC 2012 [127]	11,530	1,994	2,093	–	✗	✗

processing, and supports diverse applications ranging from occupancy detection [119], smart HVAC systems [120], or poacher detection [126].

A.1.3 Existing Person Detection Datasets

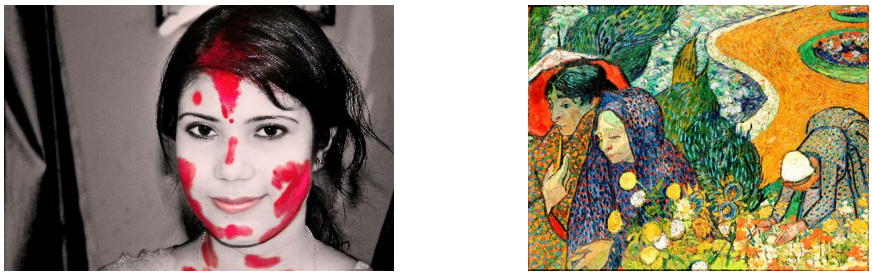
In the TinyML computer vision domain, the VWW dataset [41, 81, 10, 37, 46] has established itself as the de facto standard. Before Wake Vision, it was the only open-source dataset specifically designed for person detection with direct commercial applicability. However, VWW faces significant limitations: its small size and indirect access requirements (requiring regeneration from MS-COCO [38]) restrict its accessibility and utility.

Although other data sets contain person labels within general image classification collections, such as Cifar-100 [12] and PASCAL Visual Object Classes [127], these present significant drawbacks for TinyML applications. Specifically, their inadequate representation of the open "no-person" class can lead to poor perceived performance of TinyML models [41]. The Wake Vision dataset addresses these limitations by providing nearly two orders of magnitude more images than any existing public, permissively licensed dataset (A.1). Furthermore, it distinguishes itself as the only person detection dataset offering a fine-grained benchmark suite and official distribution through popular services like TFDS and Hugging Face Datasets.

B Dataset and Benchmark Generation

With Wake Vision, a new dataset for TinyML person detection is introduced which is two orders of magnitude larger than prior datasets. The dataset's size, quality, and detailed metadata enable new avenues of TinyML research. A.1 illustrates example images in the dataset. This section presents the methodologies for dataset generation and curation, with

(a) Older Person **(b) Near Person** **(c) Bright Image**



(d) Female Person **(e) Depicted Person**

Figure A.1: Sample images from the Wake Vision fine-grained benchmark suite (for person detection)

a focus on person detection, including label generation, data filtering, error correction, and community-driven enhancements. It also discusses the usability and generalization capabilities of the dataset pipeline in comparison with VWW.

B.1 Label Generation

A large person detection dataset is indispensable for TinyML research. However, manually labeling millions of images would be prohibitively expensive for a nascent field like TinyML. Therefore, Wake Vision is bootstrapped using existing large-scale data efforts.

The base label in Wake Vision is a binary person/non-person label. These labels are derived from Open Images [117, 118], which contain both image-level and bounding box labels of 9.6K trainable classes and 600 objects, respectively. Image-level labels are unlocalized, describing objects present in an image while bounding box labels localize an object by four coordinates. The bounding box label classes are hierarchically structured so that one class can be a subcategory or a part of another class. For example a "Woman" is a subcategory of a "Person," and a "Human Hand" is a part of "Person." A comprehensive list of Open Images labels related to Wake Vision can be found in P.

Users can adapt Wake Vision to meet the needs of specific use cases using the many configuration options available in the binary classification dataset generation pipeline. For example, a configuration option exists to change whether artistic depictions of humans are labeled as non-persons or excluded from the dataset. Reference is made to M for details about these configuration options.

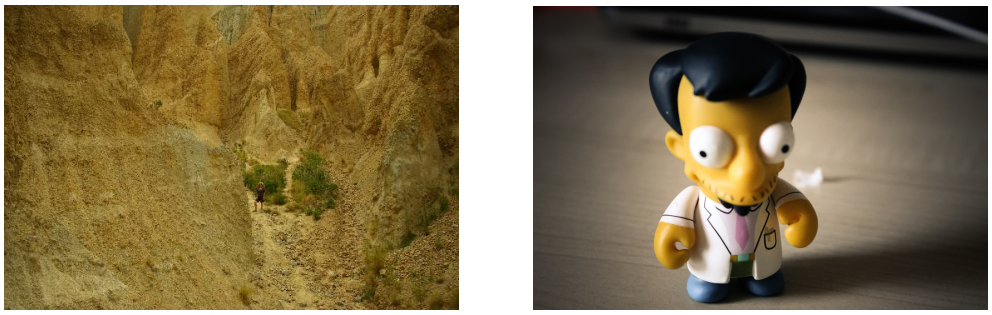
It is observed that bounding box labels are generally more accurate and provide more information for data filtering while being less numerous. The Open Images training set has ~9 million images with image-level person labels, but only ~1.7 million images with a person bounding box. Consequently, this presents a trade-off between more data (image-level labels) or higher-quality labels (bounding box labels). In response, two Wake Vision training sets are provided: Wake Vision (Large), labeled via Open Images image-level labels, and Wake Vision (Quality), a smaller set labeled via Open Images bounding boxes (Table A.1). Since the Open Images validation and test sets are fully labeled with higher-quality bounding box labels, the Wake Vision validation and test sets are derived from the bounding box labels in these Open Images splits.

Wake Vision (Large) Training Set Image-level labels in Open Images include a confidence property, which represents how certain it is that a label is correct. This confidence ranges from 0 to 1 for machine-generated labels and is strictly either 0 or 1 for human-verified labels. See B.2 for more information. Low-confidence machine-labeled person images are excluded from Wake Vision.

Wake Vision (Quality) Training Set Bounding box labels in Open Images, in contrast to image-level labels, are all verified and localized by humans, minimizing false positive labels. Bounding box labels can be used to calculate an approximation of the image area taken up by a person. Area is used as a proxy for the distance of a person to the camera, and by default far away persons are excluded, i.e., those taking up less than 5% of the image (see A.2a for an example). Finally, bounding box attributes include an artistic depiction flag (an example is provided in A.2b). By default, Wake Vision does not consider a depiction a person. See H for an illustration of the bounding box filtering process.

B.2 Label Correction

Label errors are a challenge in computer vision that limit progress and obscure true performance [128, 129]. This is especially prevalent in large datasets that use complex



(a) An example of a person far away

(b) An example of a depicted person

Figure A.2: Examples of challenging outlier images

labeling pipelines to scale while managing costs. Recognizing the importance of label quality, measures are taken to estimate and improve the accuracy of Wake Vision's labels.

As Wake Vision's labels are derived from Open Images v7; errors in these labels are inherited by Wake Vision. All image-level labels in the Open Images dataset originate from machine-generated candidate labels. Many of these labels are later verified by human annotators, and a subset of them are given bounding boxes and additional annotations.

While the machine-generated label phase aims to identify objects efficiently, any instances missed during this initial step are unlikely to be captured in downstream phases. Consequently, the Open Images dataset may contain numerous false negative labels, referring to images where an object is present but lacks the corresponding label. Therefore, additional measures are necessary to identify and correct such labeling omissions. Given Wake Vision's size, manually correcting label errors across the entire dataset becomes an arduous undertaking.

Therefore, priority is given to labeling the Wake Vision validation and test sets so that the dataset can be used to accurately evaluate a model's performance. The Scale Rapid tool from Scale AI (`https://scale.com/`) is used to relabel the Wake Vision validation and test sets. Details about the relabelling process, including instructions given and total cost, can be found in R. With the ground truth established in the validation and test sets, Wake Vision becomes an ideal foundation for research on automated data cleaning techniques [130]. An initial exploration of these methods is performed in S.

The estimated label error rate after label corrections is reported versus VWW in Table A.2. These estimates are all based on a random subset of 500 samples from each

Table A.2: Label error rate of VWW and Wake Vision after label correction. The Wake Vision Train (Quality) and Train (Large) have different error rates due to the different label sources. The Wake Vision Validation and Test sets have a lower error rate due to manual relabeling.

Dataset	Wake Vision Set	Label Error
Visual Wake Words	Train, Val, & Test	7.8%
Wake Vision	Train (Large)	15.2%
Wake Vision	Train (Quality)	6.8%
Wake Vision	Val & Test	2.2%

dataset, manually evaluated by the authors. The Wake Vision validation and test set is considerably lower than that of the VWW dataset top-level error rate.

B.3 Fine-grained Benchmark Suite

While Wake Vision provides a substantial improvement in scale and quality over existing datasets, it is recognized that overall test set performance alone may not capture the nuanced challenges of real-world deployment for TinyML applications. Internet-sourced images typically exhibit a distinct bias towards well-composed photographs; they are often well-lit, properly framed, and feature clearly visible subjects. This stands in stark contrast to the challenging conditions where TinyML person detection systems are typically deployed, such as varying lighting conditions, extreme viewing distances, or partial occlusions [131].

To bridge this gap between benchmark performance and practical utility, Wake Vision is augmented with a comprehensive fine-grained benchmark suite. This suite evaluates model robustness across challenging real-world scenarios where traditional accuracy metrics might mask significant failure modes. The benchmarks assess performance across critical dimensions, including lighting conditions, subject distance, and demographic attributes, enabling developers to identify potential biases or limitations during the design phase rather than after deployment. By providing these targeted evaluation sets, a more nuanced analysis of model behavior beyond aggregate accuracy metrics is enabled.

The suite consists of five fine-grained benchmark sets, three of which are applicable to any dataset created by the dataset generation pipeline, while two are specific to Wake Vi-

Table A.3: Accuracy on the Wake Vision and VWW test sets by models trained on the VWW, Wake Vision (Quality) and Wake Vision (Combined) training sets.

		Train		
		VWW	Wake Vision (Quality)	Wake Vision (Combined)
Test	VWW	88.33±0.29%	**88.59±0.17%**	**89.34±0.02%**
	Wake Vision	83.79±0.23%	**84.89±0.11%**	**85.72±0.04%**

sion. The three generally applicable fine-grained benchmarks include Distance, Lighting, and Depictions, whereas the Wake Vision-specific criteria include Perceived Gender and Perceived Age. J details all the adopted criteria. Each of the five fine-grained benchmarks has been picked based on a combination of its relevance to TinyML use cases and the availability of requisite metadata to generate the sets. Each benchmark set is a subset of the respective validation or test set filtered based on the criteria under test. The benchmark determines whether a model is sufficiently accurate in the planned deployment setting. For example, a model designer may make different design choices for a use case where the subject is close to the camera and well-lit compared to the inverse setting. I contains a case study that shows how fine-grained evaluation sets can be used to design robust models.

B.4 Community Accessibility and Involvement

Wake Vision and its fine-grained benchmark suite are available through TensorFlow Datasets [132] and HuggingFace Datasets [133] to enable easy access and use by the community. The images and labels are rehosted to ensure the dataset will not shrink due to dead links. The rehosted labels are generated according to the default dataset configuration, further described in M. Most datasets remain static after the initial release. To continuously improve Wake Vision, a partnership with the Edge AI Foundation [134] facilitates community involvement through recurrent competitions (Sec. C.5). In this vision, the iterative enhancements brought by the community are integrated into the dataset, leading to a continuous improvement cycle.

B.5 Generating Binary Image Classification Datasets for TinyML

Beyond person detection, many other TinyML applications can benefit from large-scale binary classification datasets. For example, bird detection enables smart nest systems [122] and automated feeding [123], while vehicle detection enables automated parking space

Table A.4: Accuracy on test set of a MobileNetV2-0.25 trained on image-level and bounding box labels, with and without KD. A MobileNetV2-1.0 model trained on the bounding box set is the teacher.

Training set	Label Source	Training Set Size	Accuracy	Distilled Accuracy
Wake Vision (Large)	Image-Level	5,760,428	80.80±0.18%	85.00±0.20%
Wake Vision (Quality)	Bounding Box	1,248,230	84.89±0.11%	85.17±0.18%
Wake Vision (Combined)	Both	5,760,428	85.72±0.04%	-

occupancy monitoring [124] and blind spot monitoring [125]. It is demonstrated that the binary dataset generation pipeline is adaptable for constructing image classification datasets for these and other classes of interest among the 9.6K trainable classes or 600 boxable objects present in Open Images v7 [117, 118].

Constructing binary classification datasets using this pipeline yields significantly larger datasets compared to the prior VWW [41] methodology applied to the same class. For the bird category, the dataset generated via the pipeline contains 12× more images than VWW, with a decrease in label error rate from 6.4% to 4.8%. Similarly, for car detection, 27× more images are provided with a decrease from 6.6% to 0.6% in label error rate. Further details are available in W.

C Results

A comprehensive evaluation of Wake Vision is presented through five complementary analyses. First, the two training sets are compared to derive insights on best practices when training. Second, it is shown that Wake Vision is an effective drop-in replacement for Visual Wake Words by conducting cross-evaluation studies using identical MobileNetV2 models. Third, Wake Vision's benefits are validated across diverse model architectures used in the TinyML community, from MCUNet to ColabNAS families. Fourth, fine-grained benchmark results are presented across demographics, environmental conditions, and visual contexts to assess model robustness in real-world scenarios. Fifth, the results of the first Wake Vision Challenge and its resulting community-driven improvements to the Wake Vision dataset are presented. Through these analyses, it is demonstrated that Wake Vision not only improves upon VWW's performance but also enables a more thorough evaluation of TinyML computer vision models.

C.1 Training Set Evaluation

To evaluate the performance of the two Wake Vision training sets, identical models are trained on each set. MobileNetV2 [49] models with a width modifier of 0.25 are used, training for 200,000 steps on 224x224x3 images using AdamW [135], a learning rate of 2e-3 with cosine decay and a weight decay of 4e-6. After training, each model's performance is compared on the common test set.

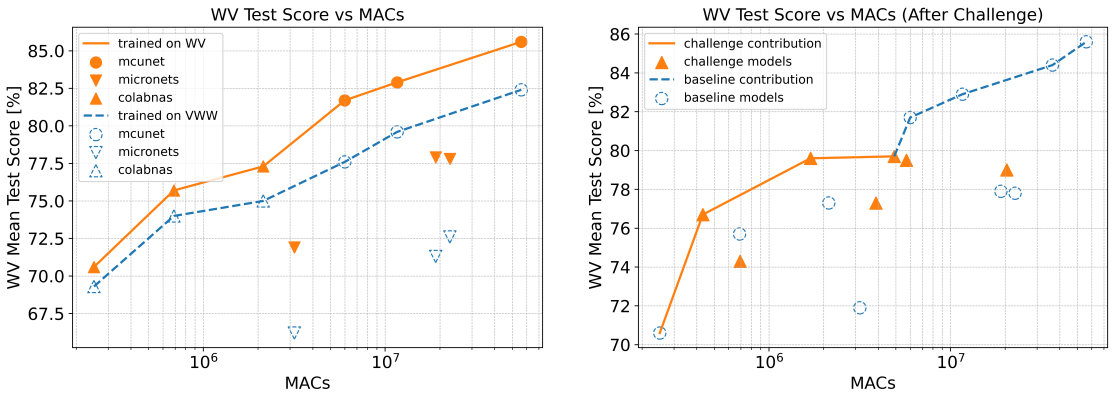
The result of this comparison is shown in A.4. The Wake Vision (Quality) training set outperforms the Wake Vision (Large) training set by 4.09% test accuracy, indicating that label quality is more important than quantity in this evaluation setting. The gap between the two is reduced to just 0.17% when training exclusively on soft labels from a teacher model (MobileNetV2-1.0) trained on bounding box labels. A further study investigates how training set size and quality impact accuracy across model sizes, observing that smaller models are more sensitive to errors in the training set, dropping accuracy by a delta of 0.8% compared to larger models (G).

Using the two training sets in unison to train a model, where Wake Vision (Large) acts as a pre-training set and Wake Vision (Quality) acts as a fine-tuning set, achieves the best performance at a mean top-level test accuracy of 85.72%. The two training sets together provide a foundation for further research on data-centric TinyML [136, 137].

C.2 Wake Vision & VWW Cross Evaluation

A key challenge in advancing TinyML research is ensuring that improvements can be readily adopted by the community without requiring significant modifications to existing workflows or architectures. While novel datasets can offer better quality or scale, their practical impact is limited if they require substantial changes to model architectures, training pipelines, or deployment processes. Therefore, Wake Vision was designed to serve as a drop-in replacement for VWW.

To validate this drop-in capability, a cross-evaluation study is performed. Two identical MobileNetv2 [49] models are trained according to the same training recipe as in C.1. One model uses VWW's training set, and the other uses Wake Vision's Quality training set. After training these identical models using the respective datasets' training sets with the same recipe for an equal number of steps, their performance is evaluated on the corresponding test sets.



(a) Initial Pareto frontier for Wake Vision vs VWW

(b) Advanced Pareto frontier by challenge submissions

Figure A.3: The Wake Vision Challenge submissions advanced the initial Pareto frontiers

A.3 shows the result of the cross-evaluation. The Wake Vision trained model outperforms the VWW trained model with a 0.26% improvement on VWW's own test set, which indicates that Wake Vision is a direct improvement over VWW, not simply a domain shift.

Also, a 1.1% improvement over VWW is achieved on Wake Vision's Test set, indicating that the new test set is more challenging. A.3 also shows the performance of the Wake Vision (Combined) model described in B.1. This model achieves 0.75% improvement over the Wake Vision (Quality) trained model on the VWW test set and 0.83% improvement on the Wake Vision test set. The Wake Vision (Combined) model is trained for longer than the VWW and Wake Vision (Quality) models but shows the value of using the training sets in unison even on the VWW test set.

C.3 Comprehensive Model Architecture Evaluation

To show that the benefits of using Wake Vision over VWW are not limited to MobileNetv2 architectures, several other models are also trained on both training sets, Wake Vision (Quality) and VWW, and their performance is compared. For this experiment, a collection of models commonly used in the TinyML community is used, such as models from the MCUNet [10] and Micronets [37] families. Furthermore, models from ColabNAS [47] are included to provide evaluation results on even smaller models.

Each model is trained on each dataset using the Adam optimizer with a learning rate of 1e-3 for 243,800 steps at a batch size of 512, equivalent to 100 epochs on the Wake Vision (Quality) training set. In addition, random horizontal flips, rotations, and 8-bit

quantization-aware training are applied. The checkpoint with the best validation score is used for testing. In this comparison, each trained model's performance is cross-evaluated on the test sets of both VWW and Wake Vision. Results on the Wake Vision test set can be found in A.3a, while further details and the results for the VWW test set can be found in E.

For the majority of models, the one trained on the Wake Vision (Quality) set achieves a higher accuracy on both the VWW and Wake Vision test set, which supports the claim that Wake Vision is a direct improvement over VWW. At the extreme, for one Micronets [37] model (middle right of A.3a), switching the training set to Wake Vision increased the test accuracy by as much as 6.6%. The only instance where Wake Vision isn't a complete improvement over VWW is in A.4 in E, where small models perform better when trained on the same dataset they are tested on (i.e., trained on VWW and tested on VWW), likely due to ultra-low-capacity models failing to generalize past the slight domain shift between the VWW and Wake Vision test sets.

Accuracy and performance metrics, such as latency or model size, are in constant tension in TinyML models. Therefore, any boost in accuracy achieved through Wake Vision can also be traded off for more efficient architectures (in some cases 3X fewer MACs) while preserving test accuracy.

C.4 Fine-Grained Benchmark Evaluation

Aggregate metrics like accuracy or F1-score provide high-level insight but can mask critical failure modes and biases that emerge in real-world scenarios. In response, fine-grained benchmarks are included in Wake Vision (Section B.3) across demographics, environmental conditions, and visual contexts. This is particularly important for person detection models deployed in real-world applications, where failures on certain subgroups or conditions could adversely affect fairness and safety.

In Table A.5, the models from C.2, trained on Wake Vision and VWW, are compared across various scenarios to understand their relative strengths and limitations. The Wake Vision model exhibits superior robustness across the challenging settings exercised by the benchmarking suite. For instance, on the "Depictions" benchmark, which evaluates performance on images containing persons, non-person objects, or no depictions, the Wake Vision model achieves an F1 score of 0.71 for person depictions, outperforming the VWW model's 0.66. The Wake Vision model also showcases improved performance on the "Age" benchmark, with F1 scores of 0.94, 0.91, and 0.94 for young, middle, and older

Table A.5: Wake Vision Fine-Grained Benchmark Suite. The samples in each set and the average F1 score across three Wake Vision models and three VWW models on each benchmark are reported.

		Gender			Age				Distance			Lighting			Depictions		
		Female	Male	Unknown	Young	Middle	Older	Unknown	Near	Medium	Far	Dark	Normal	Bright	Person	Non-Person	No Depiction
Size	Val	684	1310	1612	275	2133	90	1299	5457	2213	398	3255	14315	1012	356	352	8583
	Test	2157	3918	4940	884	6595	276	3837	16333	6876	1140	9420	43010	3332	978	1101	25802
F1	Wake Vision	**0.93**	**0.91**	0.77	**0.94**	**0.91**	**0.94**	0.71	**0.91**	**0.85**	0.59	**0.85**	**0.82**	**0.82**	**0.71**	**0.86**	**0.87**
	VWW	0.89	0.88	**0.78**	0.90	0.88	0.89	**0.74**	0.89	0.84	**0.67**	0.81	**0.82**	**0.82**	0.66	0.82	0.85

individuals, surpassing the VWW model's scores of 0.90, 0.88, and 0.89, respectively. This highlights the dataset's effectiveness in enhancing model robustness for detecting people across various age groups, particularly the elderly demographic. In addition, the Wake Vision model demonstrates resilience to challenging lighting conditions, achieving F1 scores of 0.85 in dark lighting scenarios, compared to the VWW's 0.81.

Notably, a VWW model only significantly outperforms a Wake Vision model on the far distance benchmark at a score of 0.67 and 0.59, respectively. During dataset generation,VWW filters out persons smaller than 0.5% of the image, versus 5% for Wake Vision. Therefore, VWW contains images with significantly smaller persons. Experiments were conducted adopting a default of 0.5% for Wake Vision, however this disproportionately decreased the performance of models in other settings.

Collectively, these results underscore the significance of Wake Vision's large-scale, diverse data and fine-grained benchmarks, enabling the development of more robust and reliable TinyML person detection models across a wide range of realistic and real-world scenarios.

C.5 The Wake Vision Flywheel: Community-Driven Continuous Improvement

Unlike most datasets which remain static, Wake Vision employs a community-driven flywheel where each competition generates improvements that fuel greater participation and innovation in subsequent rounds, thereby advancing TinyML research and commercial applications. A partnership exists with the Edge AI Foundation [134] to host Wake Vision competitions. The inaugural competition in early 2025 featured two tracks, one focused

on developing high-performance, resource-efficient models, and the other on enhancing the Wake Vision person detection dataset. The event garnered significant support from the TinyML community, with 48 participants across both tracks [138].

Successful challenge outcomes included an advanced pareto frontier (A.3b), containing 3 new models (orange), and a technique to enhance the quality of the Wake Vision large split [139], which in experiments led to a reduction in the label error rate from 15.2% to 9.8%. Further details can be found in X. A new competition is currently being launched [140] to further community-led advancements to Wake Vision, and the best results will be integrated in a new version of the dataset. This competition-driven advancement exemplifies the vision of continuous dataset improvement.

D Ethical Considerations and Limitations

Several ethical considerations are acknowledged. *Potential Misuse:* The dataset and benchmark suite are designed to advance TinyML research and applications. Intended benefits include occupancy detection systems that conserve energy by turning off the lights/HVAC/TV when a person leaves, and privacy preservation through on-device inference, avoiding cloud processing of camera data. It is acknowledged that person detection technology can be misappropriated for harmful purposes such as weapons targeting or mass surveillance systems. *Data Rights and Privacy:* Images are sourced from Flickr through Open Images under the CC-BY 2.0 license. Some images may have been uploaded without proper distribution rights, or may contain potentially identifying biometric information. *Fairness and Representation:* The benchmark suite is designed to evaluate model fairness and robustness across different demographics and conditions, ensuring person detection systems work equitably for all populations. However, demographic benchmarks can be misused towards classifying gender and age without consent, potentially enabling privacy violations or discriminatory practices.

An automated binary dataset generation pipeline was introduced that addresses a fundamental challenge in TinyML: the systematic creation of large-scale, high-quality datasets tailored to diverse deployment constraints. The approach combines intelligent multi-source label fusion, confidence-aware filtering, automated label correction, and systematic benchmark generation; innovations necessitated by the prohibitive cost and inconsistency of manual approaches for TinyML's bespoke dataset requirements. Through Wake Vision, a large-scale binary classification dataset demonstrating the methodology

via person detection, a 100× scale improvement and 6.6% accuracy gains were achieved over existing benchmarks. More importantly, critical TinyML-specific insights were uncovered: smaller models exhibit greater sensitivity to label quality than their large-scale counterparts, and optimal performance requires carefully orchestrated two-stage training that balances scale and quality. The fine-grained evaluation framework revealed performance disparities across demographic and environmental conditions that aggregate metrics fail to capture. The generality of the approach extends beyond person detection, successfully creating production-grade datasets for bird and car detection with similar improvements. By enabling systematic dataset creation across Open Images' 20K classes, this work provides a methodological foundation that can accelerate TinyML research across diverse application domains. All resources are open-sourced under CC-BY 4.0 license to maximize community impact.

E Additional Benchmark Results

Table A.6 provides input shape, Flash, RAM, and MACs for each model presented in section C.2 as well as the mean and standard deviation of test accuracies for both Wake Vision and VWW datasets. RAM and Flash have been measured using the "stm32tflm" utility of X-CUBE-AI 8.1.0, whereas MACs with "stm32ai". Links pointing to the original models used for training are also present for each model family involved in the experiment.

Figure A.4 shows cross-evaluation results on the Visual Wake Words test set. The majority of models trained on the Wake Vision dataset obtain better test accuracy on the VWW than the ones trained on the VWW. Only the smallest models do not achieve a higher test accuracy when trained on the Wake Vision, likely due to the lack of enough resources to overcome the slight domain shift between the two datasets.

F Code Repository

The code used to generate the Wake Vision dataset is available at the following GitHub repository: `https://github.com/harvard-edge/Wake_Vision`

This repo contains the code to generate Wake Vision and the benchmark suite, as well as the code to train and evaluate models. This code is sufficient to reproduce all results in the paper.

Table A.6: Raw data of reconstructed models.

Model .tflite	RAM [kiB]	Flash [kiB]	MACs [MM]	Test Set	Training Set WV (Qual.)	Training Set VWW
MobileNetV2 [49]						
MobileNetV2_0.25	1,244.5	410.55	36,453,732	**WV**	84.9 ± 0.11	83.8 ± 0.23
				VWW	88.6 ± 0.17	88.3 ± 0.29
MCUNet [10]						
10fps_vww	168.5	533.84	5,998,334	**WV**	81.7 ± 0.28	77.6 ± 0.31
				VWW	81 ± 0.12	80.8 ± 0.1
5fps_vww	226.5	624.55	11,645,502	**WV**	82.9 ± 0.29	79.6 ± 0.26
				VWW	83.2 ± 0.57	82.6 ± 0.16
320kb-1mb_vww	393	923.76	56,022,934	**WV**	85.6 ± 0.34	82.4 ± 0.62
				VWW	86.5 ± 1.01	86 ± 0.25
Micronets [37]						
vww2_50_50	71.50	225.54	3,167,382	**WV**	71.9 ± 0.67	66.2 ± 0.88
				VWW	70.8 ± 0.68	70.6 ± 0.64
vww3_128_128	137.50	463.73	22,690,291	**WV**	77.8 ± 0.56	72.6 ± 1.01
				VWW	78.3 ± 0.91	77.6 ± 1.13
vww4_128_128	123.50	417.03	18,963,302	**WV**	77.9 ± 0.6	71.3 ± 1.03
				VWW	78.4 ± 0.51	76.5 ± 0.37
ColabNAS [47]						
k_2_c_3	18.5	7.66	250,256	**WV**	70.6 ± 0.96	69.3 ± 0.97
				VWW	65.6 ± 0.66	70.7 ± 0.08
k_4_c_5	22	18.49	688,790	**WV**	75.7 ± 0.18	74 ± 0.23
				VWW	69.9 ± 0.26	75.5 ± 0, 64
k_8_c_5	32.5	44.56	2,135,476	**WV**	77.3 ± 0.37	75 ± 0.15
				VWW	73 ± 0.91	77.3 ± 0.57

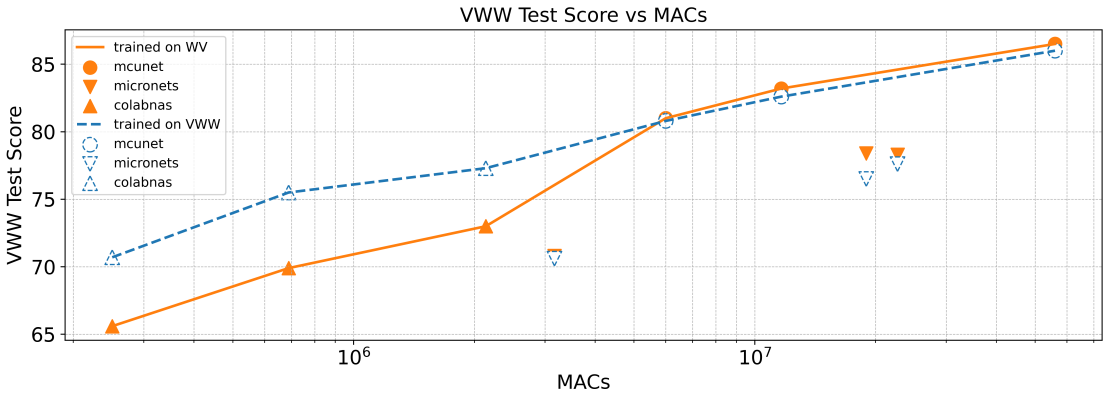


Figure A.4: Cross-evaluation results on the Visual Wake Words test set

G Impact of Dataset Quality & Size in TinyML

Understanding the relationship between dataset quality and size is important for training efficient models in real-world scenarios. While gathering large amounts of data has become easier, ensuring high-label quality remains costly and time-consuming. This creates an important trade-off: is it better to have a smaller dataset with high-quality labels or a larger dataset with more label noise? The answer likely depends on the capacity of the model and the specific task.

We investigate the impact of dataset quality and size on Resnet [141] style models of varying capacity. We measure quality by the approximate rate of label errors. The Wake Vision (Quality) training set has an estimated error rate of around 7%. We simulate higher error rate datasets (15% and 30%) by flipping the binary label with a certain probability. Appendix G.1 gives the derivation for the flip probability. We also sweep the dataset size by taking a slice of Wake Vision (Quality). Each model is trained for 50,000 steps on 224x224x3 images using AdamW [135], a learning rate of 0.001 with a cosine schedule and a weight decay of 0.004.

A.5 shows that smaller models (leftmost figure) are more sensitive to higher error rates than large models, similar to prior findings that model capacity can help memorize and ignore errors and outliers [142, 143, 144]. The lowest accuracy drop in the smallest model is 1. 3%, going from the base train (quality) error rate of around 7% to 15% compared to only a 0.5% accuracy loss for the largest model. In contrast, large models benefit more from big datasets and start to overfit on the smaller slices of the training data [145]. We also see the large slice of the 15% error rate dataset outperforming a smaller slice of the 7% error rate dataset.

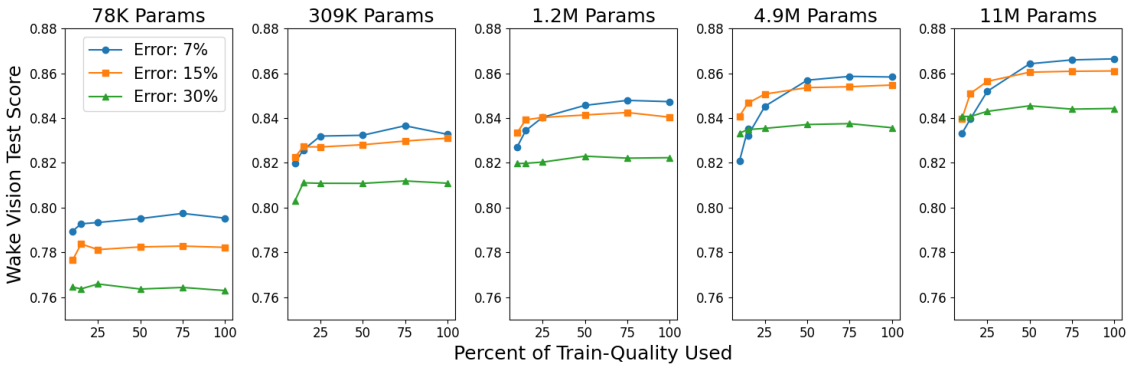


Figure A.5: Impact of dataset error rate and size on models of varying capacity. The y-axis is the Wake Vision test accuracy and the x-axis is the percentage of Wake Vision Train (Quality) used in training. The error % indicates the expected rate of incorrectly labeled samples.

G.1 Inducing Label Errors

The goal is to make a single pass through the dataset and flip the labels of a binary classification dataset such that the expected label error rate is d. There is an underlying rate of label errors e. If we flip one of these underlying errors, we correct it, thereby inadvertently decreasing the label error rate. After flipping labels with a probability of p we can claim that the likelihood of a single label being correct is the probability that we flipped the label and it was originally an error plus the probability that we didn't flip the label and it was originally correct: $1-d = p*e+(1-p)(1-e)$. Then solving for p gives $p = (e-d)/(2e-1)$. A current flaw of this method is that the injected label errors are not consistent between epochs, which would likely be less destructive to a model's accuracy since the same errors are not reinforced each epoch. This could also potentially explain why large models in Fig. A.5 don't overfit on training data with higher label errors, as the inconsistent label noise has a regularizing effect.

H Flowchart of Bounding Box Filtering Process

A.6 illustrates the filtering process for Wake Vision when using Open Images' bounding box labels as the label source.

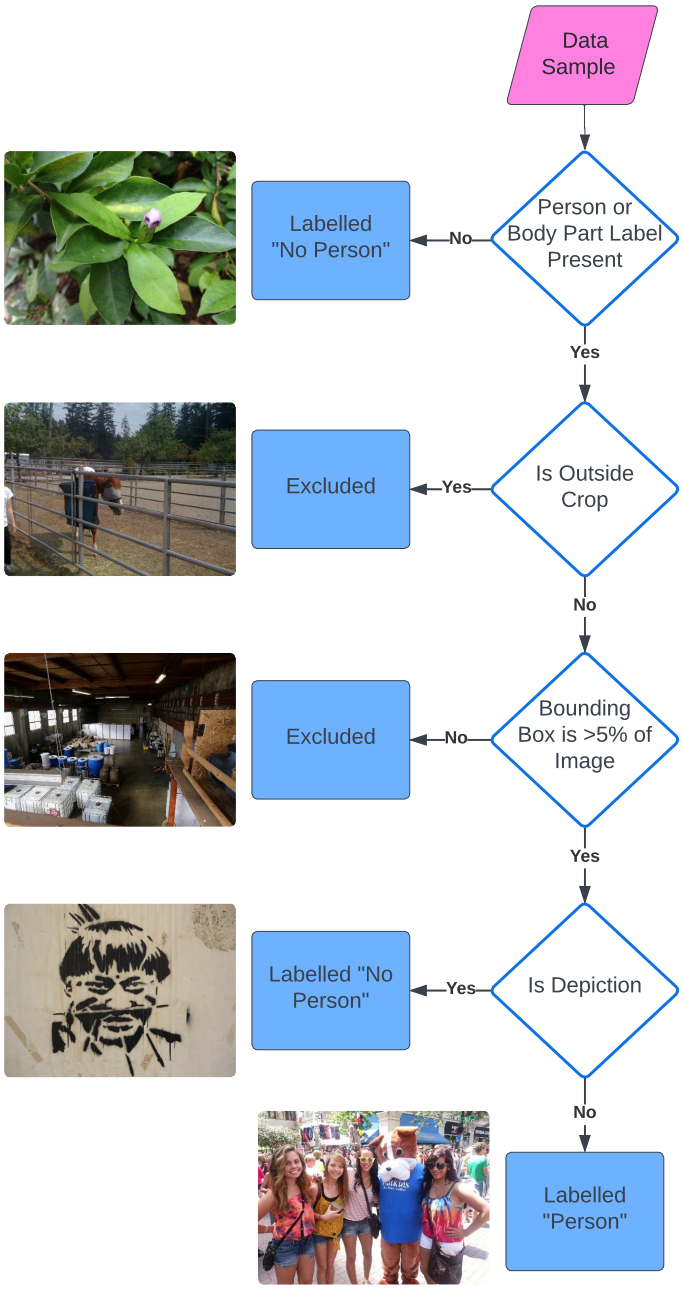


Figure A.6: A flowchart of the filtering of an image according to default Bounding Box filtering rules.

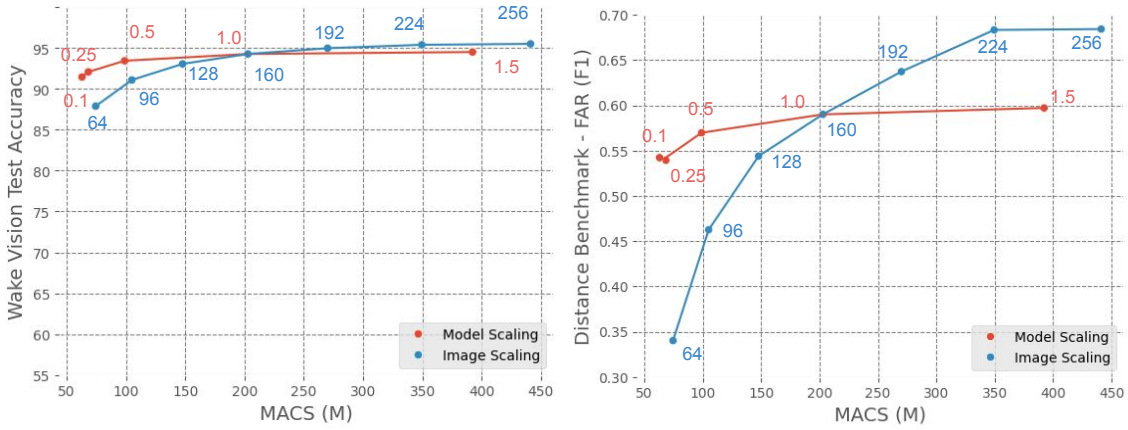


Figure A.7: Effects of scaling the image size vs. the model width on Wake Vision test accuracy and the Distance-Far benchmark set. The Distance-Far F1 score (right) is far more sensitive to changes in the image size than the overall test accuracy (left). Image sizes refer to square image resolutions (e.g. 96x96) and model size refers to the MobileNetV2 width multiplier. When not specified the image size is 160 and the width multiplier is 1.0.

I Model Design Case Study

Basic test set performance can often misrepresent a model's performance, given the typical domain shift between images scraped from the internet and real-world use cases. This issue is exacerbated when ML practitioners must trade off accuracy for model performance and size, which is necessary for TinyML use cases. A design decision might have seemingly little impact on the test accuracy but may destroy real-world performance depending on the deployment environment. For example, a person detection system may only operate in dark lighting conditions, but the test dataset has an insignificant number of dark samples; therefore, the test accuracy will not reflect the real-world accuracy.

The benchmark suite enables more holistic analysis during the design phase. To show this use we perform a series of scaling experiments employing typical TinyML compression techniques and identifying under which circumstances these techniques are appropriate. While these results can inform ML practitioners, our intention is to demonstrate the usefulness of the benchmark suite.

I.1 Image Size vs. Model Width Scaling

We train two series of MobileNetV2 models: one series that sweeps the input image size [64-256] and one that sweeps the width multiplier of a model [0.1-1.5]. We then benchmark these models on the Wake Vision test set as well as the far distance benchmark.

We plot these results against the number of multiply-accumulate (MAC) operations in the model as a proxy for on-device latency [37].

The results in Table A.7 (left) show that when looking exclusively at the high-level metric (i.e., test accuracy), scaling the input image size has a similar impact as scaling the model size. However, as shown in Figure A.7 (right), when we consider only samples where the person is far away from the camera (i.e., the distance benchmark), we observe a much more significant impact when scaling the image size. In the case of distant subjects, the image size becomes the bottleneck.

These findings suggest that for ML developers targeting use cases where the subject is likely to be far from the camera, prioritizing larger input image sizes over wider models may be more beneficial. However, this critical design consideration could be obscured when solely relying on high-level metrics like overall test accuracy. The distance benchmark in Wake Vision effectively unveils the disproportionate impact of image size on model performance for distant subjects, enabling more informed decision-making during model optimization.

I.2 Quantization

Quantization is a crucial technique for deploying efficient TinyML models, offering substantial benefits in terms of reduced latency, memory footprint, and model size. However, prior work has suggested that quantization can disproportionately impact the performance of models on underrepresented subsets of data [146]. To assess the implications of quantization in the context of person detection, we investigate the impacts of int8 quantization on a model's benchmark results across Wake Vision's fine-grained benchmarks.

Our findings show negligible degradation in performance across all benchmarks ($\pm$0.004 F1) when employing int8 quantization, even on outlier sets. This result contradicts the previously observed disproportionate impact of quantization on underrepresented subsets. We speculate that person detection may be a relatively simple task, potentially explaining why we do not observe this specific property of quantization in our experiments. Given the negligible performance degradation and the substantial latency, memory, and model size benefits of quantization, we conclude that quantization is a win for person detection.

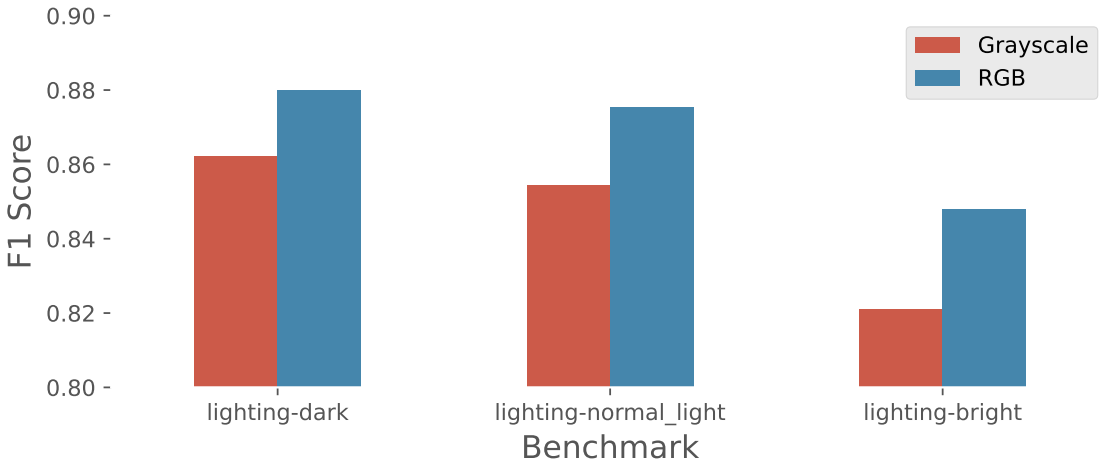


Figure A.8: Impact of grayscale input images on the lighting benchmarks: dark, normal light, and bright. Models that use grayscale input images are more sensitive to bright lighting conditions than RGB.

I.3 Grayscale

Converting a model's input image channels to grayscale from RGB is a commonly employed optimization in the TinyML field [37] as it can substantially reduce a model's memory consumption. We observed, however, that the grayscale optimization disproportionately impacts images on the brighter end of the spectrum as illustrated in Figure A.8. This further demonstrates the importance of fine-grained analysis, as some real-world deployment environments might be far brighter than the average Wake Vision test sample.

J Fine Grained Benchmark Suite

This section details the fine grained benchmark suite proposed.

Perceived Gender and Age Underrepresented sub-groups typically constitute a small portion of a generic test set; therefore, top-line metrics often obscure a bias in a model until it is deployed [147]. This benchmark, generated from the Open Images miap extended fairness labels [148], evaluates a model separately on demographics that are underrepresented in the underlying dataset distribution, identifying bias. These labels are based on perceived gender and age representation and are not necessarily representative.

Distance This tests how the distance of people in images impacts model performance. If a person detection system is intended to recognize subjects at great distances, the system's performance on the faraway dataset will be more informative than its performance on the top-level test set. We created three data sets based on the percentage of the image the subject bounding box covers. The three sets are near (>60%), at a medium distance (10-60%), and far away (<10%).

Lighting The lighting data sets test the performance of ML systems in different lighting conditions. In scenarios like security monitoring, outdoor robotics, or augmented reality applications, models must be robust to varying lighting conditions, including low-light environments. We create three fine-grained datasets of this type for dark, normal, and bright lighting conditions, respectively. We quantify lighting conditions by the average pixel values of images in greyscale, a simple but effective method for distinguishing lighting conditions [149]. We define low as an average pixel value less than 85, normal as between 85 and 170, and bright lighting conditions as greater than 170.

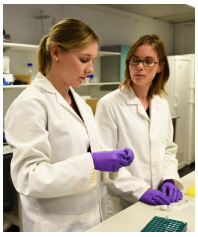
Depictions A particularly challenging task for a person detection model is to correctly reject depictions of people. In many use cases, a person detection model should not trigger on a depiction. For example, a room occupancy detector could incorrectly identify a painting on the wall as a person. This benchmark measures a model's accuracy on three related sets of non-person samples: depictions of people, depictions that are not of people, and images that do not contain a depiction of any kind. Depictions of people can range from photo-realistic to crude stick figures.

A.9 gives example images for each of the benchmarks in the suite.

K Open Image Label Distribution

Wake Vision is derived from Open Images V7, therefore the diversity of subjects in the images should follow the distribution of labels in Open Images. The Label distribution of Open images can be found here: `https://storage.googleapis.com/openimages/web/factsfigures_v7.html#statistics`

Gender



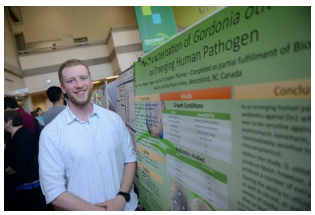
Female

Male

Gender Unknown

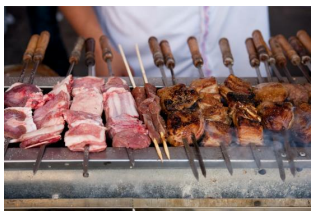
Age

Young



Middle

Older



Age Unknown

Distance

Near

Medium

Far

Lighting

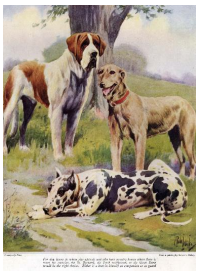
Dark

Normal

Bright

Depictions

Person



Non-Person

No Depiction

Figure A.9: Images from each fine grained benchmark dataset

L Dataset Access and Organization

Wake Vision is available online at the Harvard Dataverse, via TFDS and Hugging Face Datasets. The version on Harvard Dataverse has been assigned the following DOI: `https://doi.org/10.7910/DVN/1HOPXC` More information can be found on the `https://wakevision.ai` webpage.

The dataset is organized as a set of compressed tar files containing images and a series of label CSVs. The label CSVs are organized such that the file name of the image is the identifying index. CSVs for all dataset splits include the person and depiction label. The Validation and Test label CSVs also have flags that denote a sample's inclusion into a fine-grain benchmark set (e.g. Distance-Near). This structure makes it easy to access just the required data without requiring a full download. It also ensures the dataset can be easily updated as new versions are introduced.

M Label Generation Details

Person Labels. The most straightforward Open Images label classes to label as a person in Wake Vision are the "person" label and its subcategories (Configurable, but defaults listed in P). All of these are relabelled as persons in Wake Vision. These labels are present as both image-level labels and bounding box labels.

We furthermore inspect the image-level label classes for synonyms and umbrella terms for all the person related labels. This search resulted in an additional six person related label classes. These person related label classes will only be used in the image level label configuration.

Lastly, the image level person labels in Open Images have an associated confidence score. This score ranges from 0-10. Labels that have been verified by humans to be absent from an image have a score of 0, while labels that have been verified present by humans have a score of 10. Purely machine-generated labels have a fractional confidence score that is generally >= 5 [117, 118]. By default, we only respect labels that have a minimum confidence of 7. Labels below this threshold are ignored.

Person Body Part Labels. Body parts are more challenging to relabel, as it is dependent on the use case whether a body part should be considered a person.

For example a camera that detects whether a person is inside a room to decide if the light should be switched on would want to consider body parts as a person, as this will keep the lights on even when the person is only partly in the camera frame. For waking up electronics, however, it may not make sense to consider body parts as persons. This could, e.g., mean that a computer would turn on when detecting a foot.

To cater to both use cases we include a flag in our open-source dataset creation code to set whether body parts should be considered persons. By default we consider body parts as persons.

Depictions. Open Images bounding box labels contain metadata about whether an object is a depiction, e.g., a painting or a photograph of a person. This presents the challenge of how to handle depictions. While most use cases would not consider a depiction a person, it could make sense to either exclude them to make training easier, or include them as non-persons to make a model resistant to seeing depictions when deployed.

In line with how we handle body parts, we therefore include a flag for our open-source dataset creation code to set whether depictions should be excluded or considered non-persons. By default depictions are considered non-persons.

Bounding Box Size. The VWW dataset only considered a COCO person to be a person if the bounding box around the person took up at least 0.5% of the image [41]. If the person took up less than 0.5% of the image, the image was excluded from the dataset. For Wake Vision, we observed a decrease in performance at the 0.5% limit. We theorize that the cause of this decrease is that a person becomes indistinguishable at such a low percentage of already small images. Therefore we adopted a 5% threshold for the Wake Vision dataset instead. For different requirements, users can change a configuration parameter in our open-source dataset creation code.

N Standardized Evaluation

VWW has no standardized way to evaluate performance on the dataset. This makes it challenging to compare works based on the dataset, since performance difference can come down to choices outside the contribution of the work. e.g., two works could be

contributing with model improvements, but use different pre-processing pipelines that skews results.

To allow for both data-centric and model-centric improvements, we provide a standard model for data-centric contributions, and a standard pre-processing pipeline for model-centric contributions. Both types of contributions are expected to use accuracy as the primary metric for the overall test and validation set, and F1-score as the primary metric for the fine grained benchmarks introduced in B.3.

Therefore, for data-centric contributions we propose to use a Mobilenet v2 model with a width modifier* of 0.25 [49]. For model-centric contributions we propose the following pre-processing pipeline:

1. Cast image pixel value datatype from 8-bit integers into 32-bit floating points
2. Resize image such that the shortest side matches the model input size
3. Perform a center crop on the image such that the longest size matches the model input size
4. Normalize pixel values to between -1 and 1 sample wise
5. Use image tensor as input features and person label as target feature

O Open Images Download

The full Open Images v7 dataset is not hosted by the dataset authors. Rather it is provided as a collection of flickr image URL and their associated labels. As an unfortunate result of this, the dataset is not static over time as image owners can delete their images from the flickr platform. As a result we were only able to download a subset of the original Open Images v7 dataset as shown in A.7.

P Person Label Classes

In our default configuration, we consider the following Open Images v7 labels to be a person for the Wake Vision dataset:

Person

*Also known as alpha.

Table A.7: Number of images downloaded from Open Images v7. Download occurred between the 28th of November to the 5th of December

	Train	Validation	Test
Downloaded	7,936,979	36,406	109,305
Errors	1,055,669	5,214	16,131

Woman (Subcategory of *Person*)

Man (Subcategory of *Person*)

Girl (Subcategory of *Person*)

Boy (Subcategory of *Person*)

Human body (Part of *Person*)

Human face (Part of *Person*)

Human head (Part of *Person*)

Human (Person synonym - Only in Image Level Label Configuration)

Female person (Woman synonym - Only in Image Level Label Configuration)

Male person (Man synonym - Only in Image Level Label Configuration)

Child (Umbrella term for *Girl* & *Boy* - Only in Image Level Label Configuration)

Adolescent (Umbrella term for *Girl* & *Boy* - Only in Image Level Label Configuration)

Youth (Umbrella term for *Girl* & *Boy* - Only in Image Level Label Configuration)

The following labels are considered body parts and are labelled according to the body part configuration:

Human eye (Part of *Person*)

Skull (Part of *Person*)

Table A.8: Amount of images in the Wake Vision dataset

	Person Images	Non-Person Images	Excluded
Wake Vision (Large) Training Dataset	2,880,214	2,880,214	2,176,551
Wake Vision (Quality) Training Dataset	624,115	624,115	6,688,749
Validation Dataset	9,291	9,291	17,824
Test Dataset	27,881	27,881	53,543

Human mouth (Part of *Person*)

Human ear (Part of *Person*)

Human nose (Part of *Person*)

Human hair (Part of *Person*)

Human hand (Part of *Person*)

Human foot (Part of *Person*)

Human arm (Part of *Person*)

Human leg (Part of *Person*)

Beard (Part of *Person*)

Q Wake Vision Dataset Size

See A.8.

R Manual Label Correction

We use the Scale Rapid tool from Scale AI to relabel the Wake Vision validation and test sets. Figure A.10 shows a screenshot of the labeling menu presented to the human labelers. The labelers were instructed to label an image as a "person" if a person was present anywhere in the image and "no person" if no visible person was present. The labelers also indicated if the image contained a depiction of a person or a "picture of a picture/reflection", which we use for metadata.

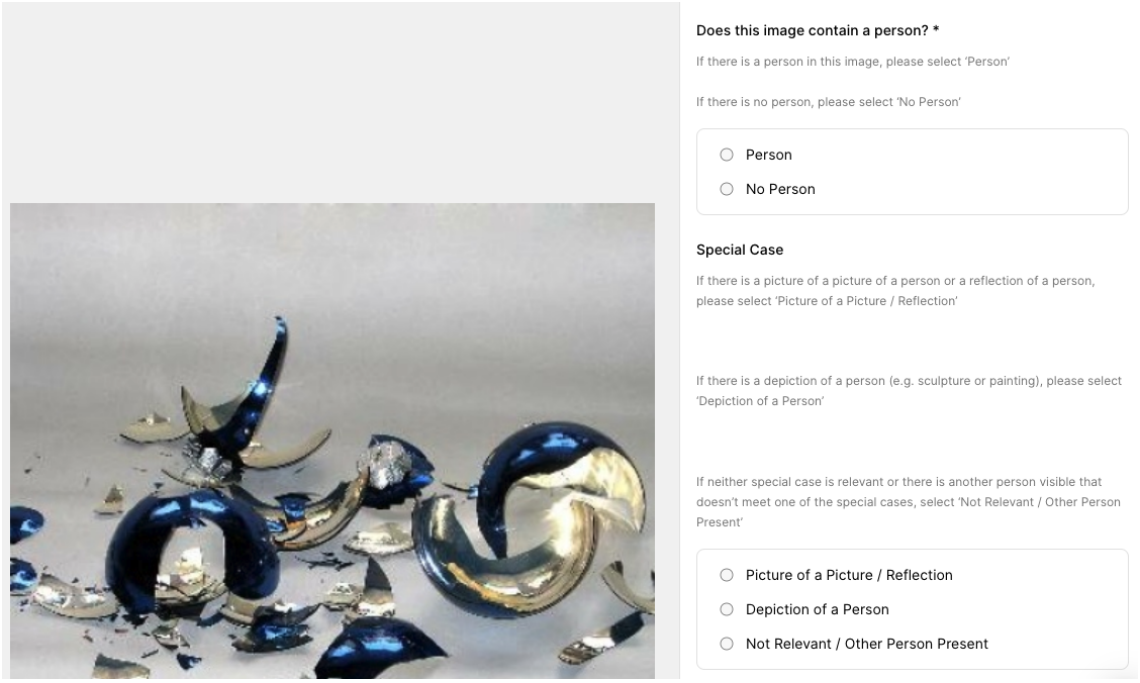


Figure A.10: A screenshot of the labeling menu used to manually label the validation and test sets.

The cost per image was $0.10 USD and is set based on Scale's pricing structure. Each image is labeled by 3 different labelers to form a consensus. The total cost of labeling was $7089.8 USD. The authors do not know the hourly rate paid to each labeler, but Scale lists $18/hr on job postings at the time of writing. For context on the difficulty of the task, the authors were able to average around 500 images per hour at a reasonable pace when estimating error rates.

S Automatic Label Correction

To address the challenge of correcting Wake Vision validation and test labels we initially employed the Confident Learning technique [130] to intelligently identify potential label errors. We selected Confident Learning as prior work has demonstrated its capability to find label errors in large datasets [128]. Confident Learning flagged suspected mislabeled instances, which we inspected and corrected through a manual verification process.

As shown in Table A.9, the confident learning process identified a large amount of possible label errors in Wake Vision's Validation and Test sets. However, only between

Table A.9: Amount of label errors identified and corrected using Confident Learning.

Dataset Split	Total Size	Suggested Errors	Corrected Errors	Suggestion Accuracy
Validation	18,582	632	81	12.82%
Test	51,282	1672	267	15.97%

12 and 16% of these possible label errors were legitimate errors. We corrected these identified label errors in the Wake Vision validation and test sets.

Given this low acceptance rate of label issues identified by Confident Learning, we concluded that we could not automate label cleaning to a point where no human in the loop is needed. This made this strategy too human-intensive to be applied to the much larger training sets.

To further correct label errors in the final Wake Vision Validation and Test sets we employed the Scale AI platform to crowd-source manual label corrections. These label corrections are described in B.2 and deprecate the automatic label corrections described in this section.

T Author Statement

The authors of this paper hereby confirm that we bear all responsibility in case of violation of rights, etc, and confirm the data license to be CC BY 4.0 for our labels, and that all images in the dataset have been uploaded to Flickr with a CC BY 2.0 license. Note: while the Open Images authors tried to identify images that are licensed under a Creative Commons Attribution license, we make no representations or warranties regarding the license status of each image and users should verify the license for each image themselves.

U Hosting, License and Maintenance Plan

The dataset is hosted on HuggingFace Datasets. Additionally, the dataset is available through TensorFlow datasets and can be regenerated from Open Images using our open-

source filtering code. The Edge AI Foundation will assume responsibility for hosting and maintaining the dataset long-term.

Wake Vision's labels and metadata are licensed under a CC BY 4.0 license. Images in the dataset have all been uploaded to Flickr with a CC BY 2.0 license. Note: while the Open Images authors tried to identify images that are licensed under a Creative Commons Attribution license, we make no representations or warranties regarding the license status of each image and users should verify the license for each image themselves.

V DOI and Croissant URLs

The version of Wake Vision on Harvard Dataverse has been assigned the following DOI: `https://doi.org/10.7910/DVN/1HOPXC` The following two croissant files describe the Wake Vision (Quality, Validation, and Test) and Wake Vision (Large) respectively: `https://huggingface.co/api/datasets/Harvard-Edge/Wake-Vision/croissant` `https://huggingface.co/api/datasets/Harvard-Edge/Wake-Vision-Train-Large/croissant`

W Car and Bird Datasets

Table A.10 compares the number of images as well as the label error rate for the car and bird datasets, when generated using both our methodology and the VWW one. All datasets are balanced, i.e., have the same number of images for the target and background class. Label error rates have been manually computed on a subset of 500 randomly selected images for each generated dataset. Examples of positive class images from the Wake Vision test splits are shown in A.11

X Wake Vision Challenge Results

Submissions to the model-centric track have been trained using the same recipe adopted in C.3. Table A.11 provides input shape, Flash, RAM, and MACs for each model presented in C.5 as well as the mean and standard deviation of test accuracies for the Wake Vision dataset. RAM and Flash have been measured using the "stm32tflm" utility of X-CUBE-AI 8.1.0, whereas MACs with "stm32ai".

Table A.10: Raw data of Car and Bird datasets.

Methodology	# of Images	Label Error Rate
	Car	
Wake Vision (Ours)	157,668	4.8%
VWW [41]	12,902	6.4%
	Bird	
Wake Vision (Ours)	31,642	0.6%
VWW [41]	1,148	6.6%

Table A.11: Raw data of model-centric track.

Model .tflite	input volume	RAM [kiB]	Flash [kiB]	MACs [MM]	Test Acc [%]
anas-benalla	(50,50,3)	48.5	104.55	4,899,292	79.7 ± 0.28
samy	(80,80,3)	73.5	34.55	5,718,064	79.5 ± 0.61
benx13	(48,48,3)	63.5	94.95	1,693,474	79.6 ± 0.22
cezar	(50,50,3)	245.5	57.23	20,435,868	79.6 ± 0.42
mohammad_hallaq	(80,80,3)	129.5	128.32	3,887,331	77.3 ± 0.5
ymac	(50,50,3)	34.5	27.77	431,985	76.7 ± 0.51
apighetti	(50,50,3)	48.5	278.36	693,818	74.3 ± 74.3

Table A.12 shows test accuracy, precision, and recall of a MobileNetV2 having a width modifier of 0.25, trained on the original Wake Vision large split and on the same split enhanced with the technique proposed by the winner of the data-centric track [139]. The training recipe is the same reported in C.3.

(a) Wake Vision "Car" samples **(b) Wake Vision "Bird" samples**

Figure A.11: Positive class examples from the test splits of our Wake Vision generated datasets for "Car" and "Bird" objects, demonstrating our ability to easily prototype large-scale TinyML vision datasets (B.5)

Y Training Resources

A desktop computer featuring a 13th Gen Intel® Core™ i9-13900K × 32, a 32 GB of RAM, and an NVIDIA GeForce RTX 4080 with 16 GB of VRAM has been employed to train the TinyML models.

Z Datasheet for Wake Vision

This document is based on *Datasheets for Datasets* by Gebru *et al.* [150]. Please see the most updated version here.

MOTIVATION

For what purpose was the dataset created? Was there a specific task in mind? Was there a specific gap that needed to be filled? Please provide a description.

Table A.12: Main improvement of the data-centric track.

Metrics	Train Large Split	
	Original	Enhanced
MobileNetV2_0.25		
Test Accuracy	82.3 ± 0.2	87.1 ± 0.81
Precision	86.4 ± 1.49	87.6 ± 1.78
Recall	76.7 ± 1.83	86.5 ± 1.32

Wake Vision was created as a prototype for a new generation of datasets for TinyML. Where previous TinyML datasets were small due to limited resources, Wake Vision is a production grade dataset consisting of almost 6 million images and with an extensively cleaned validation and test set. Wake Vision specifically targets the binary person classification task. Such a large dataset enables TinyML researchers to explore questions previously impossible. For example, the size of the Wake Vision dataset enables new research into the tradeoff that often exist between dataset size and sample quality.

Who created this dataset (e.g., which team, research group) and on behalf of which entity (e.g., company, institution, organization)?

The authors of this paper created the dataset (Harvard Edge Computing Lab, Technical University of Denmark, and Useful Sensors).

What support was needed to make this dataset? (e.g.who funded the creation of the dataset? If there is an associated grant, provide the name of the grantor and the grant name and number, or if it was supported by a company or government agency, give those details.)

The Google TPU Research Cloud program partially supported this work via cloud computing credits. The PhD stipend of Emil Njor was supported by the Innovation

Fund Denmark DIREC project (9142-00001B). Emil Njor's external research stay was furthermore supported by Fulbright Denmark, Stibo Fonden, Thomas. B. Thriges Fond, Otto Mønsteds Fond and Kaj og Hermilla Ostenfelds Fond. This work was also support by the National Science Foundation (NSF) and the Semiconductor Research Corporation (SRC).

Any other comments?
No additional comments

COMPOSITION

What do the instances that comprise the dataset represent (e.g., documents, photos, people, countries)? Are there multiple types of instances (e.g., movies, users, and ratings; people and interactions between them; nodes and edges)? Please provide a description.

The dataset is composed of images, labels, and metadata. The images are sourced from Open Images. The labels are binary, with '1' indicating there is a person somewhere in the image and '0' indicating that there is not a person in the image. The metadata is used to form the benchmark suite where we can evaluate models in specific challenging settings. Each image has a binary variable for each fine-grained benchmark where a '1' indicates that the image belongs to the fine-grained benchmark set.

How many instances are there in total (of each type, if appropriate)?

There are over 6 million labeled images in the dataset.

Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set? If the dataset is a sample, then what is the larger set? Is the sample representative of the larger set (e.g., geographic coverage)? If so, please describe how this representativeness was validated/verified. If it is not representative of the larger set, please describe why not (e.g., to cover a more diverse range of instances, because instances were withheld or unavailable).

The dataset uses a large subset of the Open Images dataset (6 million out of a possible 9 million images). We use a subset in order to have a balanced number of person samples and non-person samples. We also filter the dataset for quality. The filtering removes subjects that are too far and outside of a center crop. Open Images sources images from Flickr that are licensed as CC-BY 2.0. This is likely not a fully representative sample of the overall world population and likely skews heavily towards a European or North American population. See the Open Images paper [117] for more information.

What data does each instance consist of? "Raw" data (e.g., unprocessed text or images) or features? In either case, please provide a description.

The dataset is composed of images, labels, and metadata.

Is there a label or target associated with each instance? If so, please provide a description.

Yes, every image has a binary label to indicate if a person is visible in the image. Images in the some dataset splits include more metadata labels, e.g., about distance of subject or whether the subject is a depiction.

Is any information missing from individual instances? If so, please provide a description, explaining why this information is missing (e.g., because it was unavailable). This does not include intentionally removed information, but might include, e.g., redacted text.

The training set does not contain the metadata used for the fine grained benchmark suite. The dataset contains two training sets, one of higher quality and one larger. The higher quality dataset and other sets contain information about depictions and body parts. The large training set does not contain this information.

Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)? If so, please describe how these relationships are made explicit.

No

Are there recommended data splits (e.g., training, development/validation, testing)? If so, please provide a description of these splits, explaining the rationale behind them.

Yes there are separate splits for training, validation, and testing. These splits are generated from equivalent splits in Open Images. The dataset contains two training splits, one focusing on quality of samples and one on number of samples.

Are there any errors, sources of noise, or redundancies in the dataset? If so, please provide a description.

Yes there are label errors in the dataset due to the labeling process. We discuss this at length in the paper.

Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)? If it links to or relies on external resources, a) are there guarantees that they will exist, and remain constant, over time; b) are there official archival versions of the complete dataset (i.e., including the external resources as they existed at the time the dataset was created); c) are there any restrictions (e.g., licenses, fees) associated with any of the external resources that might apply to a future user? Please provide descriptions of all external resources and any restrictions associated with them, as well as links or other access points, as appropriate.

The dataset is self-contained. We re-host the images, labels, and metadata. The images may be subject to different licenses than our labels and metadata. All images have originally been hosted on Flickr under a permissive CC BY license.

Does the dataset contain data that might be considered confidential (e.g., data that is protected by legal privilege or by doctor-patient confidentiality, data that includes the content of individuals' non-public communications)? If so, please provide a description.

No, the images were posted online with the CC-BY 2.0 license. It is possible an image was posted without the individual's consent. While we tried to identify images that are licensed under a Creative Commons Attribution license, we make no representations or warranties regarding the license status of each image, and a user should verify the license for each image themselves.

Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? If so, please describe why.

No, not to our knowledge. Due to the size of the dataset we have not had the resources to manually verify this.

Does the dataset relate to people? If not, you may skip the remaining questions in this section.

Yes, the dataset contains images of people.

Does the dataset identify any subpopulations (e.g., by age, gender)? If so, please describe how these subpopulations are identified and provide a description of their respective distributions within the dataset.

Yes the benchmark suite contains subsets that focus on perceived age and gender. These labels have been sourced from the More Inclusive Annotations of People [148]. The purpose of these benchmarks are to test the fairness of a trained model across these subpopulation and we do not condone using these labels to train a model. The size of these sets is listed in Table A.5.

Is it possible to identify individuals (i.e., one or more natural persons), either directly or indirectly (i.e., in combination with other data) from the dataset? If so, please describe how.

It could be possible to identify a person by their face or other visible characteristics in the image. We do not share any other personal information besides the images.

Does the dataset contain data that might be considered sensitive in any way (e.g., data that reveals racial or ethnic origins, sexual orientations, religious beliefs, political opinions or union memberships, or locations; financial or health data; biometric or genetic data; forms of government identification, such as social security numbers; criminal history)? If so, please provide a description.

No, not to our knowledge. Some personal or sensitive information may be visible in the images. However, the images were shared publicly under a CC-BY 2.0 license. We do not include any personal information beyond the images.

Any other comments?

The images used were published publicly under a CC-BY 2.0 license and are already used in the Open Images [118] dataset and its derivatives. The authors of Open images tried to identify images that are licensed under a Creative Commons Attribution license but make no representations or warranties regarding the license status of each image and a user should verify the license for each image themselves.

COLLECTION

How was the data associated with each instance acquired? Was the data directly observable (e.g., raw text, movie ratings), reported by subjects (e.g., survey responses), or indirectly inferred/derived from other data (e.g., part-of-speech tags, model-based guesses for age or language)? If data was reported by subjects or indirectly inferred/derived from other data, was the data validated/verified? If so, please describe how.

The dataset is derived from Open Images [118], which in turn acquired its images from Flickr. Section B.1 describes the process.

Over what timeframe was the data collected? Does this timeframe match the creation timeframe of the data associated with the instances (e.g., recent crawl of old news articles)? If not, please describe the timeframe in which the data associated with the instances was created. Finally, list when the dataset was first published.

The original data collection was conducted in Noember of 2015. The Open Images paper [117] details the image acquisition process.

What mechanisms or procedures were used to collect the data (e.g., hardware apparatus or sensor, manual human curation, software program, software API)? How were these mechanisms or procedures validated?

The images were sources from Flickr in an automated pipeline. The original labeling pipeline was a combination of automated labeling and manual verification and labeling. The Open Images paper details this process [117]. We detail our process for relabeling and filtering the dataset in Section B.1.

What was the resource cost of collecting the data? (e.g. what were the required computational resources, and the associated financial costs, and energy consumption - estimate the carbon footprint. See Strubell *et al.*[151] for approaches in this area.)

We used cloud TPU credits provided by the Google Cloud TRC program. The primary source of required resources was model training to evaluate the dataset and the storage and bandwidth required to process and upload the dataset to hosting locations. The total size of the dataset is approximately 2 TB. To the authors knowledge, there is no public information about the power draw of TPUs. This matches the conclusions drawn by Strubell *et al.*[151]. This makes it impossible to disclose the energy consumption and estimated carbon footprint of the project.

We used paid crowdsource labelers to relabel the validation and test sets. The total cost of labeling was approximately $7,000.

If the dataset is a sample from a larger set, what was the sampling strategy (e.g., deterministic, probabilistic with specific sampling probabilities)?

The dataset is a large subset of the total Open Images dataset. The subset is generated by a deterministic filtering process described in Section B.1.

Who was involved in the data collection process (e.g., students, crowdworkers, contractors) and how were they compensated (e.g., how much were crowdworkers

paid)?

We used paid crowdsource labelers to relabel the validation and test sets. The total cost of labeling was approximately $7,000. We used the Scale Rapid labeling service. A total of 70,898 images were labeled at a cost of $0.1 per image and $0.033 per individual label. Appendix B.2 gives additional detail.

Were any ethical review processes conducted (e.g., by an institutional review board)? If so, please provide a description of these review processes, including the outcomes, as well as a link or other access point to any supporting documentation.

No

Does the dataset relate to people? If not, you may skip the remainder of the questions in this section.

Yes the dataset contains images of people.

Did you collect the data from the individuals in question directly, or obtain it via third parties or other sources (e.g., websites)?

We obtained the images via the Open Images dataset [117] which originally sourced its images from Flickr [152].

Were the individuals in question notified about the data collection? If so, please describe (or show with screenshots or other information) how notice was provided, and provide a link or other access point to, or otherwise reproduce, the exact language of the notification itself.

No the individuals were not notified to our knowledge. The images were posted publicly under the CC-BY 2.0 license.

Did the individuals in question consent to the collection and use of their data? If so, please describe (or show with screenshots or other information) how consent was requested and provided, and provide a link or other access point to, or otherwise reproduce,

the exact language to which the individuals consented.

Yes, the images were posted publicly under the CC-BY 2.0 license. It is possible that the images were uploaded by a third party without the consent of the individual in the image.

If consent was obtained, were the consenting individuals provided with a mechanism to revoke their consent in the future or for certain uses? If so, please provide a description, as well as a link or other access point to the mechanism (if appropriate)

No we do not have a mechanism for an individual the revoke their consent. As with Open Images, we make no representations or warranties regarding the license status of each image, and a user should verify the license for each image themselves via the Flickr link in the metadata. Note that it is impossible to revoke a CC BY license after publication [153].

Has an analysis of the potential impact of the dataset and its use on data subjects (e.g., a data protection impact analysis)been conducted? If so, please provide a description of this analysis, including the outcomes, as well as a link or other access point to any supporting documentation.

No formal analysis has been conducted. We describe the ethical considerations of the dataset in Section D.

Any other comments?

No additional comments.

PREPROCESSING / CLEANING / LABELING

Was any preprocessing/cleaning/labeling of the data done(e.g.,discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)? If so, please provide a description. If not, you may skip the remainder of the questions in this section.

Yes we filtered the data as shown in Figure A.6.

Was the "raw" data saved in addition to the preprocessed/cleaned/labeled data (e.g., to support unanticipated future uses)? If so, please provide a link or other access point to the "raw" data.

The "raw" data is accessible in the Open Images dataset.

Is the software used to preprocess/clean/label the instances available? If so, please provide a link or other access point.

Yes, we open-source our filtering code. `https://github.com/harvard-edge/Wake_Vision`

Any other comments?

No additional comments

USES

Has the dataset been used for any tasks already? If so, please provide a description.

The dataset has been used for the experiments reported in the paper.

Is there a repository that links to any or all papers or systems that use the dataset? If so, please provide a link or other access point.

The usage of the dataset can be tracked via the citations of this paper. We plan to host competitions using the dataset and will track participants.

What (other) tasks could the dataset be used for?

The dataset can be used as a benchmark for TinyML models and as the basis for data-centric AI research given the prevalence of label errors in the training set and the cleaned nature of the validation and test sets.

Is there anything about the composition of the dataset or the way it was collected and preprocessed/cleaned/labeled that might impact future uses? For example, is there anything that a future user might need to know to avoid uses that could result in unfair treatment of individuals or groups (e.g., stereotyping, quality of service issues) or other undesirable harms (e.g., financial harms, legal risks) If so, please provide a description. Is there anything a future user could do to mitigate these undesirable harms?

Yes, the estimated error rates are reported in Table A.2. We also discuss the high likelihood of false negatives in Wake Vision's training sets in Section B.2. These errors are due to the labeling and filtering process. We mitigate the impact of these errors by manually labeling the validation and test sets as well as study their impact for TinyML models in Section G.

Are there tasks for which the dataset should not be used? If so, please provide a description.

The perceived age and gender benchmarks should not be used to train a gender or age classifier.

Any other comments?

No additional comments

DISTRIBUTION

Will the dataset be distributed to third parties outside of the entity (e.g., company, institution, organization) on behalf of which the dataset was created? If so, please provide a description.

The dataset is hosted on Harvard Dataverse and HuggingFace Datasets. Additionally, the dataset is available through TensorFlow datasets and can be regenerated from Open Images with our open source filtering code. We plan for the TinyML foundation (`https://www.tinyml.org/`) to assume responsibility for hosting and maintaining the dataset long-term.

How will the dataset will be distributed (e.g., tarball on website, API, GitHub)? Does the dataset have a digital object identifier (DOI)?

Yes, the dataset is hosted on Harvard Dataverse and HuggingFace Datasets. Additionally, the dataset is available through TensorFlow datasets and can be regenerated from Open Images with our open source filtering code.

When will the dataset be distributed?

The Dataset is publicly accessible at the time of writing.

Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)? If so, please describe this license and/or ToU, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms or ToU, as well as any fees associated with these restrictions.

The dataset is published under the CC-BY 4.0 license.

Have any third parties imposed IP-based or other restrictions on the data associated with the instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms, as well as any fees associated with these restrictions.

No

Do any export controls or other regulatory restrictions apply to the dataset or to individual instances? If so, please describe these restrictions, and provide a link or

other access point to, or otherwise reproduce, any supporting documentation.

No

Any other comments?

No additional comments.

MAINTENANCE

Who is supporting/hosting/maintaining the dataset?

The dataset is hosted on HuggingFace Datasets. Additionally, the dataset is available through TensorFlow datasets and can be regenerated from Open Images with our open-source filtering code. The Edge AI Foundation will assume responsibility for hosting and maintaining the dataset long-term.

How can the owner/curator/manager of the dataset be contacted (e.g., email address)?
At the time of writing, the authors can be contacted at cbanbury@g.harvard.edu. Once maintenance is transferred to the Edge AI Foundation, there will be a point of contact there.

Is there an erratum? If so, please provide a link or other access point.

No

Will the dataset be updated (e.g., to correct labeling errors, add new instances, delete instances)? If so, please describe how often, by whom, and how updates will be communicated to users (e.g., mailing list, GitHub)?

The dataset may be updated if any problematic instances are discovered or the label errors are corrected. In this case, we will issue a new version of the dataset to ensure past results are contextualized under the version of the dataset that was used.

If the dataset relates to people, are there applicable limits on the retention of the data associated with the instances (e.g., were individuals in question told that their data would be retained for a fixed period of time and then deleted)? If so, please describe these limits and explain how they will be enforced.

There is no limit.

Will older versions of the dataset continue to be supported/hosted/maintained? If so, please describe how. If not, please describe how its obsolescence will be communicated to users.

Yes, we will continue to host older versions of the dataset. This is an inherent feature of the Harvard Dataverse.

If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so? If so, please provide a description. Will these contributions be validated/verified? If so, please describe how. If not, why not? Is there a process for communicating/distributing these contributions to other users? If so, please provide a description.

Yes, the dataset is licensed permissively, and the code used to create Wake Vision is open source.

Any other comments?

No additional comments

Acknowledgements

First of all, I would like to thank Giuliano Donzellini and Gianfranco Coletti, two mentors and now friends, who inspired me to pursue an academic career. It's because of you two that I'm here, reaching this important goal.

Then, I would like to thank my supervisors Paolo Gastaldo and Antonio Frisoli for creating this unique opportunity to pursue my PhD in the two realities that made me the student I am, the University of Genoa and the Scuola Superiore Sant'Anna of Pisa. You opened the doors of your labs to me and gave me the freedom to pursue my own research ideas, which is the thing I value the most about this experience. I use this occasion to also thank Edoardo Ragusa and Daniele Leonardis for their precious suggestions about doing research and pursuing an academic career, and the other folks working in the two labs, who always made me feel welcome.

A special thanks goes to Vijay Janapa Reddi who answered my email and opened the doors of his lab to me, where I met amazing folks working on exciting projects! It was a pleasure spending time with you inside and outside the lab! Thanks to this experience I also had the opportunity to experience the vibrant environment of Cambridge (US) and Boston, full of interesting people from all across the world. Mark, Andy, Emma, Jeff, Radhika, Arya, Ike, Jason J., Jason Y., Zishen, Shvetank, Jessica and Jennifer from the lab; Ted, Cedric, Emily, Silvana and Analisa from Exeter Park 3; Inka from the best B&B; Jack, Luca, Sarah, Nicola, Magdalena and Giulia, and all the other people I met contributed to make this time of my life unforgettable.

Not to mention my lifelong friends Lapo, Matteo, and Riccardo; Marta and Martina who took a plane towards Cambridge (US), even in the middle of uncertainties with the US border; the friends from university Marianna, Maurica, Stefano, Luca, Mattia, Davide, Angelica and Annalisa; and my friends from high school Vittoria, Lorenzo, Camilla, Elia, Giorgia and Melanie, even though our lives do not allow us to see each other as often as we used to, it's still the same when we get all together!

Last but certainly not least, I would like to thank my girlfriend Giada, who remained alongside me, even though we had to travel a lot in these years between Pisa, Genoa, and even Cambridge (US) to see each other!

Publications, Projects, Teaching and Awards

Publication record

1. International Journal Papers

[1] A. M. Garavagno, D. Leonardis and A. Frisoli, "ColabNAS: Obtaining lightweight task-specific convolutional neural networks following Occam's razor", in Future Generation Computer Systems, vol. 152, pp. 152-159, March 2024, doi: https://doi.org/10.1016/j.future.2023.11.003.

[2] A. M. Garavagno, E. Ragusa, A. Frisoli and P. Gastaldo, "An Affordable Hardware-Aware Neural Architecture Search for Deploying Convolutional Neural Networks on Ultra-Low-Power Computing Platforms", in IEEE Sensors Letters, vol. 8, no. 5, pp. 1-4, May 2024, Art no. 6004604, doi: https://doi.org/10.1109/LSENS.2024.3387056.

[3] A. M. Garavagno, E. Ragusa, A. Frisoli and P. Gastaldo, "Searching Neural Architectures for Sensor Nodes on IoT Gateways", in IEEE Internet of Things Journal, vol. 12, no. 21, pp. 44492-44501, November 2025, doi: https://doi.org/10.1109/JIOT.2025.3581442.

2. International Conference Papers

[4] A. M. Garavagno, E. Ragusa, A. Frisoli and P. Gastaldo, "A hardware-aware neural architecture search algorithm targeting low-end microcontrollers", 2023 18th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Valencia, Spain, 2023, pp. 281-284, doi: https://doi.org/10.1109/PRIME58259.2023.10161736.

[5] A. M. Garavagno, E. Ragusa, A. Frisoli and P. Gastaldo, "Running hardware-aware neural architecture search on embedded devices under 512MB of RAM", 2024 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 2024, pp. 1-2, doi: https://doi.org/10.1109/ICCE59016.2024.10444268.

[6] A. M. Garavagno, E. Ragusa, P. Gastaldo, A. Frisoli and C. Loconsole, "On-Device Neural Architecture Search", 2025 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 2025, pp. 1-4, doi: https://doi.org/10.1109/ICCE63647.2025.10929801.

3. National Conference Papers

[7] A. M. Garavagno, E. Ragusa, R. Zunino, A. Frisoli and P. Gastaldo, "Supporting in-Sensor Computing with Hardware-Aware Neural Architecture Search", 55th Annual Meeting of the Italian Electronics Society (SIE), Genoa, Italy, 2024, Lecture Notes in Electrical Engineering, vol 1263. Springer, doi: https://doi.org/10.1007/978-3-031-71518-1_48.

4. ArXiv Preprints

[8] C. Banbury*, E. Njor*, A. M. Garavagno*, M. Mazumder, M. Stewart, P. Warden, M. Kudlur, N. Jeffries, X. Fafoutis and V. J. Reddi, "Wake vision: A tailored dataset and benchmark suite for tinyml computer vision applications", 2024, arXiv preprint, doi: https://doi.org/10.48550/arXiv.2405.00892. *These authors contributed equally to this work.

5. Blog Posts

[9] C. Banbury, E. Njor, A. M. Garavagno and V. J. Reddi, "Introducing Wake Vision: A High-Quality, Large-Scale Dataset for TinyML Computer Vision Applications", 2024, TensorFlow Blog, url: `https://blog.tensorflow.org/2024/12/introducing-wake-vision-new-dataset-for-person-detection-in-tinyml.html`. Accessed: 28th October 2025.

Participation in Research Projects

Research project 1, Project Title "LEARN – muLtimodal Edge computing-bAsed weaRable exoskeletoNs for assistance in daily life" – Project funded under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.1 – Call for tender No. 1409 published on Sept 14, 2022 by the Italian Ministry of University and Research (MUR) funded by the European Union – NextGenerationEU – CUP: D53D23016190001, D53D23016200001, J53D23014090001 – Grant Assignment Decree No. 1383 adopted on September 01, 2023 by the Italian Ministry of University and Research (MUR).

Description of the research project 1: LEARN focuses on assistive devices that in the shape of wearable robot systems can empower or restore motor functions in healthy workers or in patients with motor impairments. The main goal is to exploit Machine Learning (ML) as enabling technology for the development of the next generation of such assistive devices. To this purpose, the project strongly relies on interdisciplinary research. The three research units involved in LEARN bring their expertise on the different topics covered by the research project: machine learning, embedded systems, edge computing, computer vision, robotic systems for human-machine interaction, wearable robotics.

The major outcome of the project is the development of novel, advanced wearable exoskeleton prototypes for motor assistance, where embedded ML plays a core role in significantly increasing controllability and usability by the end users. The advanced features to be added to wearable exoskeletons are: higher controllability by damaged or limited efferent control signals from the user, operational awareness, and shared autonomy between the user and the device to increase usability in more complex manipulation tasks.

The integration of ML and edge computing is indeed a main factor in LEARN. Advanced wearable exoskeletons take advantage of two technological innovations in this area. First, the deployment of computer vision for Affordance Segmentation (AS) on edge devices, Second, the development of a low-power control system based on embedded ML for classification of sEMG and biometric signals. The first research activity involves the design of a novel RGB-D (i.e., RGB + Depth) AS system adopting lightweight DL architectures. As a major result, the assistive device is equipped with sensors and cognitive-like processing that enable semi-autonomous actions (e.g., automatic preshaping). The second research activity exploits shallow networks and lifelong ML to deploy on a microcontroller a control system that can both detect the motion intent relying on biometric signals and improve its performance day after day by adapting to the user.

The project addresses two different application scenarios: medical assistive devices and wearable devices in the industrial and service fields. In the former scenario, exoskeletons actively support movements of the patient. Here, embedded ML enables semi-autonomous operations that can significantly improve usability of the device, diminishing the complexity of the communication channel between the user and the device in terms of number of degrees of freedom and continuous versus event-based control. In the latter scenario, exoskeletons and wearable assistive devices support operator's movements, hence significantly diminishing the burden applied to the body. In this case, embedded ML enhances dual modality (transparency versus modulated assistance), which strongly impacts on the final usability of the device and its acceptability by the end user.

Activity: the candidate contributed to the second research activity carrying out activity A2.1 sEMG and biometric signal-based HMI development (mo. 1-18), where a novel HMI has been developed based on the concept of running HW-NAS at the edge for automatically designing tailor made HMIs based on a small set of labeled data acquired on the fly, and activity A2.2 Integration of the HMI with the wearable system (mo. 13-18), where the developed HMI has been integrated in an assistive hand exoskeleton, specifically designed for the project, and tested on healthy subjects to validate the entire system.

Attendance to workshops

Biological, Chemical and Medical Sensors and Applications in the IEEE Sensors Conference 2024, Kobe, Japan, 20-23 October 2024, presenting the poster number 7298: Garavagno, A. M., Ragusa, E., Frisoli, A., & Gastaldo, P. Running hardware-aware neural architecture search on embedded devices under 512MB of RAM.

Teaching and other activities

Teaching support activities for the courses:

> “Architecture and Programming of Electronic Systems” (Italian: “Architetture e Programmazione dei Sistemi Elettronici”), course code 108695, academic year 2024-2025;
>
> “Electronic Systems Programming” (Italian: “Programmazione dei Sistemi Elettronici”), course code 108694, academic year 2023-2024;
>
> “Electronic Systems Architecture” (Italian: “Architettura dei Sistemi Elettronici”), course code 72303, academic year 2023–2024;
>
> “Computer Science and Telematics Laboratory” (Italian: “Laboratorio di Informatica e Telematica”), course code 94721, academic year 2022-2023;

Co-supervised 5 Master / Bachelor’s Thesis at the University of Genoa,

> “Soft-core RISC V: Implementation of the M instruction set for integer multiplication and division” (Italian: “Soft-core RISC V: implementazione del set di istruzioni M per moltiplicazioni e divisioni intere”), Moreno Tomaghelli, October 2025;
>
> “Microcontroller implementation of a monodimensional convolutional neural network (1D-CNN) for rolling-element bearing fault diagnosis” (Italian: “Implementazione su microcontrollore di una rete neurale convoluzionale monodimensionale per il riconoscimento di guasti su cuscinetti a sfera”), Simone Tami, December 2024;
>
> “32-bit RISC-V soft-core design for FPGAs” (Italian: “Progetto di soft-core RISC-V a 32 bit per FPGA”), Andrea Moretto, September 2024;
>
> “Training neural networks on resource-constrained programmable processors” (Italian: “Addestramento di reti neurali su processori programmabili con risorse limitate”), Matteo Caroleo, July 2024;
>
> “RISC-V microprocessor design on the Deeds platform” (Italian: “Progetto di microprocessore RISC-V sulla piattaforma Deeds”), Giacomo Manzotti, December 2023.

Visiting student at the Harvard John A. Paulson School Of Engineering And Applied Sciences (SEAS), under the supervision of Prof. Vijay Janapa Reddi.

Awards and scientific recognitions

Fellow of the Harvard John A. Paulson School of Engineering and Applied Sciences (SEAS), 2025;

Member of the technical committee “Machine learning, Deep learning and AI in Consumer Electronics (MDA)” of the IEEE Consumer Technology Society (IEEE CTSoc), 2025;

Best Session Presentation Award at the 2024 IEEE International Conference on Consumer Electronics (ICCE 2024);

Invited Talk “A hardware-aware neural architecture search algorithm targeting ultra-low-power microcontrollers”, presented at the tinyML Global Talks, Edge AI Foundation (ex tinyML Foundation), 2023;

Invited Lecture “Fault diagnosis in inspection using robots”, presented at the “PSL lectures on Machine learning for physics and engineering”, PSL University, Paris, March 6th-10th 2023.

Bibliography

[1] 32-bit arm cortex mcus. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] Statista, "Number of internet of things (iot) connected devices worldwide from 2019 to 2030," Statista Research Department, 2023, accessed: 29th October 2025. [Online]. Available: https://www.statista.com/statistics/1194682/iot-connected-devices-vertically/

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[5] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 12, pp. 6999–7019, 2021.

[6] M. N. Bhuiyan, M. M. Rahman, M. M. Billah, and D. Saha, "Internet of things (iot): A review of its enabling technologies in healthcare applications, standards protocols, security, and market opportunities," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 474–10 498, 2021.

[7] G. Aceto, V. Persico, and A. Pescapé, "Industry 4.0 and health: Internet of things, big data, and cloud computing for healthcare 4.0," *Journal of Industrial Information Integration*, vol. 18, p. 100129, 2020.

[8] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "Hardware-aware neural architecture search: Survey and taxonomy." in *IJCAI*, vol. 2021, 2021, pp. 4322–4329.

[9] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2820–2828.

[10] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han *et al.*, "Mcunet: Tiny deep learning on iot devices," *Advances in neural information processing systems*, vol. 33, pp. 11 711–11 722, 2020.

[11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[12] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[13] M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," *Using Large Corpora*, vol. 273, p. 31, 1994.

[14] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International conference on machine learning*. PMLR, 2017, pp. 2902–2911.

[15] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017.

[16] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[17] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," *Advances in neural information processing systems*, vol. 31, 2018.

[18] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International conference on machine learning*. PMLR, 2018, pp. 4095–4104.

[19] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *International conference on machine learning*. PMLR, 2018, pp. 550–559.

[20] L. Wang, Y. Zhao, Y. Jinnai, Y. Tian, and R. Fonseca, "Neural architecture search using deep neural networks and monte carlo tree search," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 06, 2020, pp. 9983–9991.

[21] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Uncertainty in artificial intelligence*. PMLR, 2020, pp. 367–377.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[23] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *International Conference on Learning Representations*, 2017.

[24] K. Buza, "Feedback prediction for blogs," in *Data analysis, machine learning and knowledge discovery*. Springer, 2013, pp. 145–152.

[25] J. Torres-Sospedra, R. Montoliu, A. Martínez-Usó, J. P. Avariento, T. J. Arnau, M. Benedito-Bordonau, and J. Huerta, "Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems," in *2014 international conference on indoor positioning and indoor navigation (IPIN)*. IEEE, 2014, pp. 261–270.

[26] F. Graf, H.-P. Kriegel, M. Schubert, S. Pölsterl, and A. Cavallaro, "2d image registration in ct images using radial image descriptors," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2011, pp. 607–614.

[27] A. Coraddu, L. Oneto, A. Ghio, S. Savio, D. Anguita, and M. Figari, "Machine learning approaches for improving condition-based maintenance of naval propulsion plants," *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, vol. 230, no. 1, pp. 136–153, 2016.

[28] P. S. Rana, "Physicochemical properties of protein tertiary structure data set," *UCI Machine Learning Repository*, 2013.

[29] K. Fernandes, P. Vinagre, and P. Cortez, "A proactive intelligent decision support system for predicting the popularity of online news," in *Portuguese conference on artificial intelligence*. Springer, 2015, pp. 535–546.

[30] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," in *International conference on machine learning*. PMLR, 2019, pp. 7105–7114.

[31] X. Dong and Y. Yang, "Nas-bench-201: Extending the scope of reproducible neural architecture search," *arXiv preprint arXiv:2001.00326*, 2020.

[32] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," *arXiv preprint arXiv:1707.08819*, 2017.

[33] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter, "Nas-bench-301 and the case for surrogate benchmarks for neural architecture search," *arXiv preprint arXiv:2008.09777*, vol. 4, p. 14, 2020.

[34] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[35] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and 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. 10 734–10 742.

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

[37] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *Proceedings of machine learning and systems*, vol. 3, pp. 517–532, 2021.

[38] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[39] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

[40] V. J. Reddi, B. Plancher, S. Kennedy, L. Moroney, P. Warden, A. Agarwal, C. Banbury, M. Banzi, M. Bennett, B. Brown *et al.*, "Widening access to applied machine learning with tinyml," *arXiv preprint arXiv:2106.04008*, 2021.

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

[42] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[43] H. Purohit, R. Tanabe, K. Ichige, T. Endo, Y. Nikaido, K. Suefusa, and Y. Kawaguchi, "Mimii dataset: Sound dataset for malfunctioning industrial machine investigation and inspection," *arXiv preprint arXiv:1909.09347*, 2019.

[44] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov *et al.*, "Benchmarking tinyml systems: Challenges and direction," *arXiv preprint arXiv:2003.04821*, 2020.

[45] Y. Koizumi, S. Saito, H. Uematsu, N. Harada, and K. Imoto, "Toyadmos: A dataset of miniature-machine operating sounds for anomalous sound detection," in *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, 2019, pp. 313–317.

[46] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau *et al.*, "Mlperf tiny benchmark," *arXiv preprint arXiv:2106.07597*, 2021.

[47] A. M. Garavagno, D. Leonardis, and A. Frisoli, "Colabnas: Obtaining lightweight task-specific convolutional neural networks following occam's razor," *Future Generation Computer Systems*, vol. 152, pp. 152–159, 2024.

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

[49] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

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

[51] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[52] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 116–131.

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

[54] M. S. Abdelfattah, Ł. Dudziak, T. Chau, R. Lee, H. Kim, and 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.

[55] L. Yang, Z. Yan, M. Li, H. Kwon, L. Lai, T. Krishna, V. Chandra, W. Jiang, and 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.

[56] A. Marchisio, A. Massa, V. Mrazek, B. Bussolino, M. Martina, and 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.

[57] X. Zheng, C. Yang, S. Zhang, Y. Wang, B. Zhang, Y. Wu, Y. Wu, L. Shao, and R. Ji, "Ddpnas: Efficient neural architecture search via dynamic distribution pruning," *International Journal of Computer Vision*, vol. 131, no. 5, pp. 1234–1249, 2023.

[58] S. Li, Y. Mao, F. Zhang, D. Wang, and G. Zhong, "Dlw-nas: differentiable light-weight neural architecture search," *Cognitive Computation*, vol. 15, no. 2, pp. 429–439, 2023.

[59] M. Iman, H. R. Arabnia, and K. Rasheed, "A review of deep transfer learning and recent advancements," *Technologies*, vol. 11, no. 2, p. 40, 2023.

[60] T. Sanida, A. Sideris, D. Tsiktsiris, and M. Dasygenis, "Lightweight neural network for covid-19 detection from chest x-ray images implemented on an embedded system," *Technologies*, vol. 10, no. 2, p. 37, 2022.

[61] S. Liu, W. Kong, X. Chen, M. Xu, M. Yasir, L. Zhao, and J. Li, "Multi-scale ship detection algorithm based on a lightweight neural network for spaceborne sar images," *Remote sensing*, vol. 14, no. 5, p. 1149, 2022.

[62] Y. Wang, C. Bai, X. Qian, W. Liu, C. Zhu, and L. Ge, "A dc series arc fault detection method based on a lightweight convolutional neural network used in photovoltaic system," *Energies*, vol. 15, no. 8, p. 2877, 2022.

[63] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[64] S. Khaki, N. Safaei, H. Pham, and L. Wang, "Wheatnet: A lightweight convolutional neural network for high-throughput image-based wheat head detection and counting," *Neurocomputing*, vol. 489, pp. 78–89, 2022.

[65] F. Jubair, O. Al-karadsheh, D. Malamos, S. Al Mahdi, Y. Saad, and Y. Hassona, "A novel lightweight deep convolutional neural network for early detection of oral cancer," *Oral Diseases*, vol. 28, no. 4, pp. 1123–1130, 2022.

[66] E. Ragusa, T. Apicella, C. Gianoglio, R. Zunino, and P. Gastaldo, "Design and deployment of an image polarity detector with visual attention," *Cognitive Computation*, vol. 14, no. 1, pp. 261–273, 2022.

[67] E. Ragusa, C. Gianoglio, S. Dosen, and P. Gastaldo, "Hardware-aware affordance detection for application in portable embedded systems," *IEEE Access*, vol. 9, pp. 123 178–123 193, 2021.

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

[69] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[70] M. Shanker, M. Y. Hu, and M. S. Hung, "Effect of data standardization on neural network training," *Omega*, vol. 24, no. 4, pp. 385–397, 1996.

[71] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. pmlr, 2015, pp. 448–456.

[72] M. H. Javid, "Melanoma skin cancer dataset of 10000 images," 2022. [Online]. Available: https://www.kaggle.com/dsv/3376422

[73] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[74] A. M. Garavagno, E. Ragusa, A. Frisoli, and P. Gastaldo, "An affordable hardware-aware neural architecture search for deploying convolutional neural networks on ultra-low-power computing platforms," *IEEE Sensors Letters*, 2024.

[75] ——, "A hardware-aware neural architecture search algorithm targeting low-end microcontrollers," in *18th Conference on Ph. D Research in Microelectronics and Electronics (PRIME)*. IEEE, 2023, pp. 281–284.

[76] E. Ragusa, S. Dosen, R. Zunino, and P. Gastaldo, "Affordance segmentation using tiny networks for sensing systems in wearable robotic devices," *IEEE Sensors Journal*, vol. 23, no. 19, pp. 23 916–23 926, 2023.

[77] J. S. Gidon, J. Borah, S. Sahoo, S. Majumdar, and M. Fujita, "Bidirectional lstm model for accurate and real-time landslide detection: A case study in mawiongrim, meghalaya, india," *IEEE Internet of Things Journal*, vol. 11, no. 3, pp. 3792–3800, 2023.

[78] E. Liberis, Ł. Dudziak, and N. D. Lane, "μnas: Constrained neural architecture search for microcontrollers," in *Proc. of the 1st Workshop on Machine Learning and Systems*, 2021, pp. 70–79.

[79] A. M. Garavagno, E. Ragusa, A. Frisoli, and P. Gastaldo, "Running hardware-aware neural architecture search on embedded devices under 512mb of ram," in *2024 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2024, pp. 1–2.

[80] J. Borah, S. Kumar, N. Kumar, M. S. M. Nadzir, M. G. Cayetano, H. Ghayvat, S. Majumdar, and N. Kumar, "Aicarebreath: Iot-enabled location-invariant novel unified model for predicting air pollutants to avoid related respiratory disease," *IEEE Internet of Things Journal*, vol. 11, no. 8, pp. 14 625–14 633, 2023.

[81] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj *et al.*, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Proc. of Machine Learning and Systems*, vol. 3, pp. 800–811, 2021.

[82] D. Kinga, J. B. Adam *et al.*, "A method for stochastic optimization," in *International conference on learning representations (ICLR)*, vol. 5, no. 6. California;, 2015.

[83] A. M. Garavagno, E. Ragusa, A. Frisoli, and P. Gastaldo, "Searching neural architectures for sensor nodes on iot gateways," *IEEE Internet of Things Journal*, vol. 12, no. 21, pp. 44 492–44 501, 2025.

[84] G. Dong, M. Tang, Z. Wang, J. Gao, S. Guo, L. Cai, R. Gutierrez, B. Campbel, L. E. Barnes, and M. Boukhechba, "Graph neural networks in iot: A survey," *ACM Transactions on Sensor Networks*, vol. 19, no. 2, pp. 1–50, 2023.

[85] J. Chen, K. Wen, J. Xia, R. Huang, Z. Chen, and W. Li, "Knowledge embedded autoencoder network for harmonic drive fault diagnosis under few-shot industrial scenarios," *IEEE Internet of Things Journal*, 2024.

[86] S.-C. Wu, P.-L. Hung, and A. L. Swindlehurst, "Ecg biometric recognition: unlinkability, irreversibility, and security," *IEEE Internet of Things Journal*, vol. 8, no. 1, pp. 487–500, 2020.

[87] C. Li and B. Palanisamy, "Privacy in internet of things: From principles to technologies," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 488–505, 2018.

[88] H. Habibzadeh, K. Dinesh, O. R. Shishvan, A. Boggio-Dandry, G. Sharma, and T. Soyata, "A survey of healthcare internet of things (hiot): A clinical perspective," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 53–71, 2019.

[89] M. Serror, S. Hack, M. Henze, M. Schuba, and K. Wehrle, "Challenges and opportunities in securing the industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 5, pp. 2985–2996, 2020.

[90] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[91] C. Zhang, X. Yuan, Q. Zhang, G. Zhu, L. Cheng, and N. Zhang, "Toward tailored models on private aiot devices: Federated direct neural architecture search," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17 309–17 322, 2022.

[92] K. T. Chitty-Venkata and A. K. Somani, "Neural architecture search survey: A hardware perspective," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–36, 2022.

[93] K. Adhinugraha, W. Rahayu, T. Hara, and D. Taniar, "On internet-of-things (iot) gateway coverage expansion," *Future Generation Computer Systems*, vol. 107, pp. 578–587, 2020.

[94] H. Lu, M. Du, X. He, K. Qian, J. Chen, Y. Sun, and K. Wang, "An adaptive neural architecture search design for collaborative edge-cloud computing," *IEEE Network*, vol. 35, no. 5, pp. 83–89, 2021.

[95] H. Zhu and Y. Jin, "Real-time federated evolutionary neural architecture search," *IEEE transactions on evolutionary computation*, vol. 26, no. 2, pp. 364–378, 2021.

[96] J. Skarding, B. Gabrys, and K. Musial, "Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey," *iEEE Access*, vol. 9, pp. 79 143–79 168, 2021.

[97] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7436–7456, 2021.

[98] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[99] J. Yang, H. Zou, Y. Zhou, and L. Xie, "Learning gestures from wifi: A siamese recurrent convolutional architecture," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 763–10 772, 2019.

[100] H. Zou, J. Yang, Y. Zhou, L. Xie, and C. J. Spanos, "Robust wifi-enabled device-free gesture recognition via unsupervised adversarial domain adaptation," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2018, pp. 1–8.

[101] X. Yu, F. Dong, B. Xia, S. Yang, E. Ding, and W. Yu, "An intelligent fault diagnosis scheme for rotating machinery based on supervised domain adaptation with manifold embedding," *IEEE Internet of Things Journal*, vol. 10, no. 1, pp. 953–972, 2022.

[102] G. Santamato, A. M. Garavagno, M. Solazzi, and A. Frisoli, "Leveraging systems' non-linearity to tackle the scarcity of data in the design of intelligent fault diagnosis systems," *Nonlinear Dynamics*, vol. 112, no. 18, pp. 16 153–16 166, 2024.

[103] H. Benmeziane, K. E. Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "A comprehensive survey on hardware-aware neural architecture search," *arXiv preprint arXiv:2101.09336*, 2021.

[104] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[105] L. L. Zhang, Y. Yang, Y. Jiang, W. Zhu, and Y. Liu, "Fast hardware-aware neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 692–693.

[106] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[107] J. Erickson, *Algorithms*, 2023.

[108] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[109] Melanoma skin cancer dataset. [Online]. Available: https://www.kaggle.com/datasets/hasnainjaved/melanoma-skin-cancer-dataset-of-10000-images

[110] W. A. Smith and R. B. Randall, "Rolling element bearing diagnostics using the case western reserve university data: A benchmark study," *Mechanical systems and signal processing*, vol. 64, pp. 100–131, 2015.

[111] C.-C. Chen, Z. Liu, G. Yang, C.-C. Wu, and Q. Ye, "An improved fault diagnosis using 1d-convolutional neural network model," *Electronics*, vol. 10, no. 1, p. 59, 2020.

[112] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[113] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller, "Layer-wise relevance propagation: an overview," *Explainable AI: interpreting, explaining and visualizing deep learning*, pp. 193–209, 2019.

[114] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann, "Shortcut learning in deep neural networks," *Nature Machine Intelligence*, vol. 2, no. 11, pp. 665–673, 2020.

[115] C. Banbury, E. Njor, A. M. Garavagno, M. Mazumder, M. Stewart, P. Warden, M. Kudlur, N. Jeffries, X. Fafoutis, and V. J. Reddi, "Wake vision: A tailored dataset and benchmark suite for tinyml computer vision applications," *arXiv preprint arXiv:2405.00892*, 2024.

[116] P. Warden and D. Situnayake, *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[117] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *IJCV*, 2020.

[118] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, S. Kamali, M. Malloci, J. Pont-Tuset, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy, "Openimages: A public dataset for large-scale multi-label and multi-class image classification." *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017.

[119] M. Piechocki, M. Kraft, T. Pajchrowski, P. Aszkowski, and D. Pieczynski, "Efficient people counting in thermal images: The benchmark of resource-constrained hardware," *IEEE Access*, vol. 10, pp. 124 835–124 847, 2022.

[120] A. Zacharia, D. Zacharia, A. Karras, C. Karras, I. Giannoukou, K. C. Giotopoulos, and S. Sioutas, "An intelligent microprocessor integrating tinyml in smart hotels for rapid accident prevention," in *2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. IEEE, 2022, pp. 1–7.

[121] G. Heitz, S. Gould, A. Saxena, and D. Koller, "Cascaded classification models: Combining models for holistic scene understanding," *Advances in neural information processing systems*, vol. 21, 2008.

[122] O. Debauche, S. Mahmoudi, A. Marzak, P. Manneback, F. Lebeau *et al.*, "Smart nest box: Iot based nest monitoring in artificial cavities," in *2020 3rd International Conference on Advanced Communication Technologies and Networking (CommNet)*. IEEE, 2020, pp. 1–7.

[123] C. Gerhardson and C.-H. Min, "Design and implementation of an iot wireless sensor network for bird feeder monitoring," in *2022 IEEE 13th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2022, pp. 0124–0128.

[124] H. Bura, N. Lin, N. Kumar, S. Malekar, S. Nagaraj, and K. Liu, "An edge based smart parking solution using camera networks and deep learning," in *2018 IEEE international conference on cognitive computing (ICCC)*. IEEE, 2018, pp. 17–24.

[125] Y. Shen and W. Q. Yan, "Blind spot monitoring using deep learning," in *2018 International Conference on Image and Vision Computing New Zealand (IVCNZ)*. IEEE, 2018, pp. 1–5.

[126] K. E. Doull, C. Chalmers, P. Fergus, S. Longmore, A. K. Piel, and S. A. Wich, "An evaluation of the factors affecting 'poacher'detection with drones and the efficacy of machine-learning for detection," *Sensors*, vol. 21, no. 12, p. 4074, 2021.

[127] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

[128] C. G. Northcutt, A. Athalye, and J. Mueller, "Pervasive label errors in test sets destabilize machine learning benchmarks," *arXiv preprint arXiv:2103.14749*, 2021.

[129] L. Beyer, O. J. Hénaff, A. Kolesnikov, X. Zhai, and A. v. d. Oord, "Are we done with imagenet?" *arXiv preprint arXiv:2006.07159*, 2020.

[130] C. Northcutt, L. Jiang, and I. Chuang, "Confident learning: Estimating uncertainty in dataset labels," *Journal of Artificial Intelligence Research*, vol. 70, pp. 1373–1411, 2021.

[131] Plumerai, "Great tinyml needs high-quality data | plumerai blog," https://blog.plumerai.com/2021/08/tinyml-data/, August 2021, (Accessed: 29th October 2025).

[132] "TensorFlow Datasets, a collection of ready-to-use datasets," https://www.tensorflow.org/datasets, 2024.

[133] Q. Lhoest, A. V. del Moral, Y. Jernite, A. Thakur, P. von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall *et al.*, "Datasets: A community library for natural language processing," *arXiv preprint arXiv:2109.02846*, 2021.

[134] "Edge ai foundation," https://www.edgeaifoundation.org/, 2025, accessed: 29th October 2025.

[135] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[136] M. Mazumder, C. Banbury, X. Yao, B. Karlaš, W. Gaviria Rojas, S. Diamos, G. Diamos, L. He, A. Parrish, H. R. Kirk *et al.*, "Dataperf: Benchmarks for data-centric ai development," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[137] E. Njor, J. Madsen, and X. Fafoutis, "Data aware neural architecture search," *arXiv preprint arXiv:2304.01821*, 2023.

[138] "Challenge edge: Wake vision," https://edgeai.modelnova.ai/challenges/details/challenge-edge:-wake-vision, 2025, accessed: 29th October 2025.

[139] K. Bedioui, "Wake vision challenge data-centric track kais bedioui submission," https://github.com/kais-bedioui/Wake_Vision_Challenge_Data_Centric_Track, 2025, accessed: 29th October 2025.

[140] "Challenge edge: Wake vision," https://edgeai.modelnova.ai/challenges/details/edge-ai-challenge:-wake-vision-2, 2025, accessed: 29th October 2025.

[141] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[142] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[143] V. Feldman, "Does learning require memorization? a short tale about a long tail," in *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, 2020, pp. 954–959.

[144] S. Goyal, P. Maini, Z. C. Lipton, A. Raghunathan, and J. Z. Kolter, "Scaling laws for data filtering–data curation cannot be compute agnostic," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 22 702–22 711.

[145] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[146] S. Hooker, N. Moorosi, G. Clark, S. Bengio, and E. Denton, "Characterising bias in compressed models," *arXiv preprint arXiv:2010.03058*, 2020.

[147] S. Hooker, A. Courville, G. Clark, Y. Dauphin, and A. Frome, "What do compressed deep neural networks forget?" *arXiv preprint arXiv:1911.05248*, 2019.

[148] C. Schumann, S. Ricco, U. Prabhu, V. Ferrari, and C. R. Pantofaru, "A step toward more inclusive people annotations for fairness," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (AIES)*, 2021.

[149] W. Zhang, H. Li, and Z. Wang, "Research on different illumination image classification method," in *2017 2nd International Conference on Automation, Mechanical Control and Computational Engineering (AMCCE 2017)*. Atlantis Press, 2017, pp. 574–581.

[150] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. D. Iii, and K. Crawford, "Datasheets for datasets," *Communications of the ACM*, vol. 64, no. 12, pp. 86–92, 2021.

[151] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *arXiv preprint arXiv:1906.02243*, 2019.

[152] Flickr, https://flickr.com/, 2024, accessed: 29th October 2025.

[153] R. U. Library, "Creative commons - copyright - libguides at roskilde university library," https://libguides.ruc.dk/copyright/cc, 2024.

Searching Neural Architectures at the Edge

Andrea Mattia Garavagno