POLITECNICO DI TORINO
Repository ISTITUZIONALE

Deep Neural Network Architecture Search for Accurate Visual Pose Estimation aboard Nano-UAVs

Original
Deep Neural Network Architecture Search for Accurate Visual Pose Estimation aboard Nano-UAVs / Cereda, E., Crupi, L., Risso, M., Burrello, A., Benini, L., Giusti, A., Jahier Pagliari, D., Palossi, D.. - ELETTRONICO. - (2023), pp. 6065-6071. (2023 IEEE International Conference on Robotics and Automation (ICRA) London (UK) 29 May 2023 - 02 June 2023) [10.1109/ICRA48891.2023.10160369].

Availability:
This version is available at: 11583/2980543 since: 2023-07-20T07:25:06Z

Publisher:
IEEE

Published
DOI:10.1109/ICRA48891.2023.10160369

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

08 November 2026

This paper has been accepted for publication in the IEEE ICRA 2023 conference
©2023 IEEE.

Deep Neural Network Architecture Search for Accurate Visual Pose Estimation aboard Nano-UAVs

E. Cereda[1], L. Crupi[2], M. Risso[2], A. Burrello[3], L. Benini[34], A. Giusti[1], D. Jahier Pagliari[2], and D. Palossi[14]

arXiv:2303.01931v1 [cs.RO] 3 Mar 2023

***Abstract*— Miniaturized autonomous unmanned aerial vehicles (UAVs) are an emerging and trending topic. With their form factor as big as the palm of one hand, they can reach spots otherwise inaccessible to bigger robots and safely operate in human surroundings. The simple electronics aboard such robots (sub-100 mW) make them particularly cheap and attractive but pose significant challenges in enabling onboard sophisticated intelligence. In this work, we leverage a novel neural architecture search (NAS) technique to automatically identify several Pareto-optimal convolutional neural networks (CNNs) for a visual pose estimation task. Our work demonstrates how real-life and field-tested robotics applications can concretely leverage NAS technologies to automatically and efficiently optimize CNNs for the specific hardware constraints of small UAVs. We deploy several NAS-optimized CNNs and run them in closed-loop aboard a 27-g Crazyflie nano-UAV equipped with a parallel ultra-low power System-on-Chip. Our results improve the State-of-the-Art by reducing the in-field control error of 32% while achieving a real-time onboard inference-rate of ∼10 Hz@10 mW and ∼50 Hz@90 mW.**

SUPPLEMENTARY VIDEO MATERIAL

In-field tests: `https://youtu.be/dVCSckvcg8`.

I. INTRODUCTION

Nano-sized unmanned aerial vehicles (UAVs) are gaining significant momentum due to their reduced form factor (sub-10 cm) and weight (sub-40 g), which allows them to fulfill sensitive missions, such as in GPS-denied narrow spaces and human proximity. Moreover, the simplified sensory, mechanical, and computational sub-systems available aboard these platforms make them particularly cheap and attractive compared to their bigger counterparts. However, making them fully autonomous, i.e., no external/remote infrastructure or computation, is still challenged by their simplified electronics, e.g., sub-100 mW compute power, which is 1-2 orders of magnitude less than mobile phones' processors.

These constraints hinder the adoption of standard algorithmic pipelines, which often rely on memory/compute-intensive sophisticated planning/localization/mapping algorithms, and heavy pre-trained perception convolutional neural networks (CNNs). Therefore, running these methods aboard nano-UAVs is unfeasible. In this context, tiny CNNs increasingly define the State-of-the-Art (SoA) of autonomous nano-drones [1], [2]. First, they can be trained to operate on data from very limited sensors, such as ultra-low power, and tiny cameras with poor resolution/dynamic range. Second, CNNs have predictable and fixed computational and memory requirements at inference time. Third, the system designer can tune such requirements to fit the available resources on the robot by defining a suitable network architecture.

Roboticists integrating CNNs in larger robots can often overlook the third aspect, relying on standard SoA architectures such as ResNet50 [3] that easily fit in the available resources. In contrast, thoroughly defining a suitable custom CNN is crucial when developing nano-class robot systems. The choice of the neural network architecture determines whether the model can run on the robot and directly impacts two critical parameters that affect the robot's behavior: *i*) prediction performance and *ii*) real-time throughput.

For many years, the only approach to fulfill the requirements posed by this complex optimization scenario with contrasting objectives (i.e., obtaining accurate yet deployable CNNs) was to resort to a manual, tedious, error-prone, and time-consuming iterative hyper-parameters tuning based on heuristics and rules-of-thumb. Nowadays, the *go-to* approach to perform such optimization is based upon the neural architecture search (NAS) paradigm [4], [5]. NAS tools enable an automatic exploration over an arbitrarily large search space of different network topologies and hyper-parameters settings. Furthermore, many novel NAS approaches can directly optimize complex cost functions by combining different objectives [5], [6], [7], such as regression performance and the network's computational complexity.

In this work, we exploit a novel computationally efficient NAS technique [7], able to generate Pareto-optimal CNN architectures in the accuracy vs. model size, to optimize a vision-based human pose estimation task. First, we contribute by enhancing the functionalities of an existing NAS engine, which is needed to explore two different seed CNNs: PULP-Frontnet [2] and MobileNetv1 [8]. Then, we thoroughly analyze and deploy multiple CNNs on our target nano-drone robotic platform. Ultimately, the delivered CNNs improve the SoA baseline [2] at least for one metric among size (up to $5.6\times$ smaller), speed (up to $1.5\times$ faster), and accuracy (up to 32% lower horizontal displacement error). The improvement in accuracy is finally confirmed by a challenging in-field testing setup, with a *never-seen-before* environment.

This work was partially supported by the Secure Systems Research Center (SSRC) of the UAE Technology Innovation Institute (TII) and the Swiss National Science Foundation (SNSF) through the NCCR Robotics.

[1]E. Cereda, A. Giusti, and D. Palossi are with the Dalle Molle Institute for Artificial Intelligence, USI and SUPSI, Lugano, 6962, Switzerland `name.surname@idsia.ch`

[2]L. Crupi, M. Risso, and D. Jahier Pagliari are with the Department of Control and Computer Engineering, Politecnico di Torino, Turin, 10129, Italy `name.surname@polito.it`

[3]A. Burrello and L. Benini are with the Department of Electrical, Electronic, and Information Engineering, University of Bologna, Bologna, 40136, Italy `name.surname@unibo.it`

[4]L. Benini and D. Palossi are also with the Integrated Systems Laboratory, ETH Zürich, Zürich, 8092, Switzerland

II. Related work

Neural architecture search. NAS tools assist designers in the design phase of DNNs by automatically exploring a large space of architectures defined as combinations of different layers and/or hyper-parameters. On constrained platforms, these tools usually minimize an objective function that depends both on task accuracy and non-functional cost metrics (e.g., memory footprint, latency, or energy consumption). Early NAS algorithms were based on *evolutionary algorithms* (EA) [4] and reinforcement learning (RL) [5]. These methods can explore arbitrary search spaces, and optimize any cost function by iteratively sampling a network, training it to convergence to evaluate performance, and then using this information to drive the following sampling.

However, their extreme flexibility implies poor scalability, due to their computational requirement, i.e., thousands of GPU hours, even for simple tasks. Instead, differentiable NAS (DNAS) [9], [6] has been proposed to mitigate this issue. Early DNASes exploit *supernets*, i.e., networks whose layers' outputs are obtained as a weighted sum of multiple alternative sub-layers (e.g., different types of convolution) applied to the same input [9]. The weights assigned to each alternative are optimized *during training*, based on an accuracy/complexity-dependent loss, and the final network is obtained by selecting, for each layer, the alternative associated with the largest weight at the end of the search. Compared to EA/RL approaches, DNAS trades some flexibility in the definition of the optimization target, which must be differentiable, in exchange for a search process that requires a *single training*. However, the convergency of a supernet is *i*) not trivial, and *ii*) still expensive (in terms of GPU memory), since multiple alternatives are instantiated for each layer.

Mask-based DNAS are a further step toward lightweight architecture search. They replace the supernet with a single-path DNN, usually referred to as *seed* [10], [11], [7], and their search space is composed of sub-architectures obtained from the seed *by subtraction* (e.g., eliminating some channels in each convolution). In practice, these sub-networks are simulated at training time using *trainable masks*, which prune part of the seed. As in supernet DNASes, the masks are optimized during training, but since the seed is much smaller than a supernet, the time and memory overhead compared to regular DNN training is minimal. On the one hand, mask-based DNASes can only produce networks derived from the seed; on the other, this allows for a much more fine-grained search space exploration. For instance, they can easily generate convolutions with an arbitrary number of channels (e.g., 17, 25, or 31 for a 32-channel seed): prohibitive with a supernet approach [7].

Nano-drones. Deploying CNN models for autonomous navigation on a nano-drone introduces severe sensorial, computational, and memory constraints due to limited form factor, payload, and energy availability. Solutions built around commercial off-the-shelf (COTS) microcontroller units (MCUs) [12], [13] can afford only minimal neural architectures to achieve real-time performance, such as [13] with 27 kMAC (multiply-accumulate operations) per frame at 100 Hz. These approaches are thus suitable only for low-dimensional input signals, not for processing camera images. More advanced visual approaches [2], [14] have been enabled by careful hardware-software co-design, exploiting general-purpose ultra-low-power multi-core System-on-Chip (SoC) [15], integrated software deployment pipelines [16], [17] and manually-tuned CNN architectures. These technological breakthroughs enabled PULP-Frontnet [2], a model with 14.7 MMAC (3 order of magnitude larger than [13]), to achieve an inference rate of 48 Hz while consuming 96 mW, aboard a COTS Crazyflie 2.1 nano-drone.

An open research question still revolves around CNN's complexity/memory reduction, as answering it would pave the way toward *i*) better energy utilization on battery-limited devices, and *ii*) enabling multi-tasking systems, yet not reached on nano-drones. Recent works [18], [19] have aimed at reducing CNNs' memory footprint with minimal degradation in accuracy, achieving up to $27\times$ reductions in computation with just 4% lower accuracy [19], but still at the cost of extensive manual fine-tuning of the architectures. In this work, we leverage NAS techniques to efficiently and automatically cope with this problem in a robotic domain.

Human pose estimation. We consider the robotic problem of monocular relative pose estimation of a human subject from an autonomous nano-drone. Despite the remarkable accuracy of SoA computer vision approaches [20], [21], [22], they are still far from the reach of nano-drones due to the required amount of computational resources [22] ($\sim$10s GMAC). To date, the PULP-Frontnet CNN [2] represents one of the few examples of a human pose estimation task fully executed aboard a Crazyflie nano-drone in real-time. While complex CNNs manage to estimate entire skeletons [21] or even dense 3D mesh representations [22] of the person, PULP-Frontnet minimizes its prediction to a 3D point in space (x, y, z) and a rotation angle w.r.t. the gravity z-axis (ϕ). MobileNetv1 [8] is another SoA CNN vastly used for vision-based classification tasks, which has also been proven accurate (e.g., 68.2% top-1 accuracy on ImageNet) when streamlined down to the same power envelope allotted on our nano-drone [23], [17]. For these reasons, we choose PULP-Frontnet and MobileNetv1 as seed models for our NAS, and we will refer to the PULP-Frontnet model as the SoA baseline (F_{SoA}) for the in-field comparison.

III. Methodology

A. Seed deep neural networks

In this work, we use a mask-based DNAS tool to optimize three seed models: one based on the shallow PULP-Frontnet architecture, and two based on a deeper MobileNetv1. PULP-Frontnet is composed by 6 convolutional layers requiring 304 k parameters and up to 14.7 MMAC. The design of the network was first introduced in [2] specifically for deployment on a nano-drone. On the other hand, MobileNetv1 [8] is a deeper network formed by 27 different convolutional layers, which mostly differ from the PULP-Frontnet by

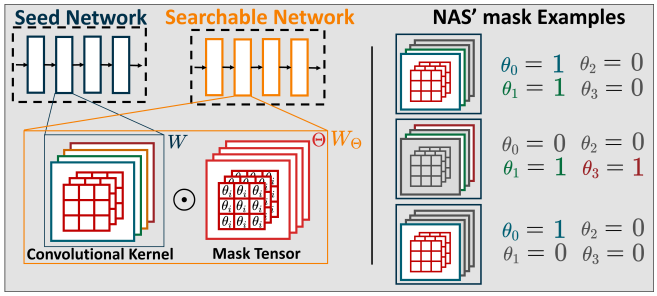

Fig. 1. Left: the proposed masking scheme. Right: three possible outcomes of the channel search process on a four-filters layer. $\theta_0 = \theta_1 = 1$ and $\theta_2 = \theta_3 = 0$ means that the first two channels are kept alive, while the second two are removed. Only the second and last channels are kept alive in the second example, and only the first channel in the last one.

the separable depthwise and pointwise convolutional layers instead of traditional ones. Our two MobileNetv1 seed networks vary by width multiplier, a hyperparameter that controls the size of the feature maps throughout the model: we consider width multipliers of 1.0 ($M^{1.0}$) and 0.25 ($M^{0.25}$), corresponding to respectively 3.21 M and 204 k parameters.

B. Network architecture search

We optimize the network architecture using a mask-based DNAS tool, namely *pruning in time* (PIT) [7], whose scope is to search for smaller and more lightweight networks which hold almost the same accuracy as the seed models. PIT shows low memory and execution time overheads (only ∼30% of the training time), several orders of magnitude lower than other NAS engines [4], [5], [9]. Additionally, the availability of good reference architectures for our task that can be used as *seed* fits well with a mask-based approach, which efficiently explores different sub-networks contained within the seed.

The NAS we propose performs a fine-grained search of the number of output channels in all convolutional layers of a 2D CNN. The approach is derived from the channel search scheme originally described in [7] for 1D Temporal Convolutional Networks. Figure 1 summarizes our NAS functionality: for each convolutional layer in the seed, its weight tensor W, with C_{out} output channels, is isolated, and the correspondent searchable tensor W_Θ is built as:

$$W_\Theta = W \odot \mathcal{H}(\theta) \tag{1}$$

where $\odot$ is the Hadamard product, θ is a trainable mask vector of C_{out} elements, in which each component θ_i represents the mask for a specific channel, and $\mathcal{H}$ is the Heaviside step function used to binarize θ. Depending on the binarized values of $\mathcal{H}(\theta_i)$ the correspondent i-th output channel of W may be kept alive ($\mathcal{H}(\theta_i) = 1$) or removed ($\mathcal{H}(\theta_i) = 0$). Therefore PIT finds models which only vary in the number of channels w.r.t. the seed networks. Compared to its original version proposed in [7], we also added support for *jointly exploring* the channels of pointwise and depthwise layers, the two main layers in depth-separable convolutional blocks and basic components of MobileNet architectures. To do so, we used shared masks to select the number of channels of these two layers, since the output channels of a depthwise convolution are completely determined by the preceding pointwise layer.

The obtained network is inserted in a normal training loop where W and θ are trained in conjunction to solve the following optimization problem:

$$\min_{W,\theta} \mathcal{L}(W;\theta) + \lambda \mathcal{R}(\theta) \tag{2}$$

where $\mathcal{L}$ is the task-specific loss function (i.e., the same loss function as the seed CNNs) and $\mathcal{R}$ is a regularization loss term representing a cost metric to be optimized (e.g., n. of parameters, n. of operations, etc.). λ is the so-called *regularization strength*, a hand-tuned scalar term used to balance the two losses. Once loss terms of Eq.2 are defined, λ represents the main knob upon which the designer can act to drive the search towards more accurate or less costly networks. For our use case, we considered $\lambda \in [5 \cdot 10^{-11} : 5 \cdot 10^{-5}]$. As regularization loss $\mathcal{R}$, we used a differentiable estimate of the number of parameters of the network as a function of the NAS mask values. In this way, we can bias the exploration phase towards architectures that are both small and accurate.

C. System design

Robotic platform. Our target robotic platform is the Bitcraze Crazyflie 2.1[1], a 27 g nano-quadrotor. This drone features a main STM32 MCU in charge of basic functionalities, such as state estimation and control loops. In our in-field deployment, it is extended with a 5 g commercial AI-deck companion board [1]. The AI-deck features an additional MCU: the GreenWaves Technologies GAP8, which embodies the parallel ultra-low power paradigm [15] through a RISC-V-based multi-core SoC. These two processors communicate via a bidirectional UART interface. The GAP8 is designed with two power domains: a single-core *fabric controller* (FC) that orchestrates the interaction with external memories/sensors and offloads computationally intensive kernels on a second 8-core *cluster* (CL) domain. The SoC's memory hierarchy relies on 64 kB of low-latency L1 memory shared among all cluster cores and 512 kB of L2 memory within the FC domain. The GAP8 also features two DMA engines to efficiently automate data transfers from/to all memories and external peripherals, such as 8 MB off-chip DRAM and a QVGA monochrome camera, both available on the AI-deck. However, it provides neither data caches nor hardware floating-point units, dictating explicit data management and the adoption of integer-quantized arithmetic, respectively.

Deployment tools. To fulfill the platform's requirements, we exploit two tools to *i*) train integer-only neural networks and *ii*) automatically and optimally generate C code for the target network. The first tool, the open-source NEMO library [16], based on the PyTorch framework, is used to train the network in three sequential steps. First, NEMO

[1] https://www.bitcraze.io/products/crazyflie-2-1/

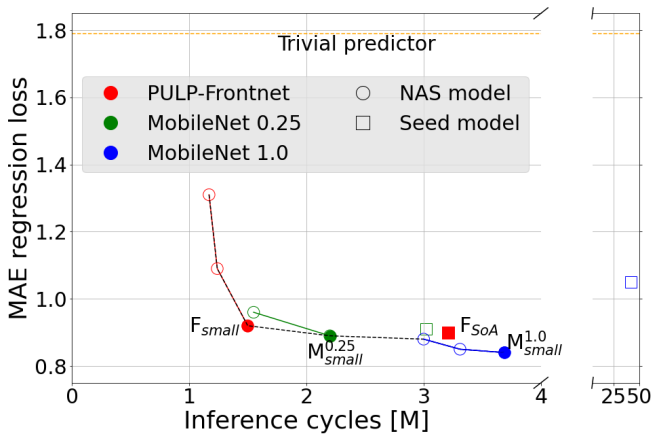

Fig. 2. Pareto curves of the networks extracted from the NAS in the clock cycles vs. MAE space (lower is better).

trains a *full-precision* floating-point network to minimize the sum of the L1 loss for the pose estimation vector (x, y, z, ϕ). We use the SGD optimizer with a *lr* of 0.001 over 100 epochs, selecting at the end the model which achieved the lowest validation loss. After, we convert this model into a *fake-quantized* network. In this stage, weights and activations are still represented as float32 values, but their magnitude can assume only a discrete (256 for 8-bits quantization) set of values. Given the support offered by the optimized kernels for GAP8, we use linear per-layer quantization and PACT [24] in this fine-tuning step. We choose to use 8-bit quantization since it is natively supported by the SIMD operations in the RISC-V ISA extensions, which allow the execution of four int8 MACs within a single cycle. To maximize the accuracy of the network, we initialize the fake-quantized network with the floating point weights, and we perform 100 additional train epochs with the same parameters. The final step is the creation of the *integer deployable* network. Compared to the fake-quantized network, all the tensors are represented by integer numbers and a float scale factor. Tensors $\mathbf{t}$ are approximated as

$$\mathbf{t} \approx \varepsilon_{\mathbf{t}} \cdot \mathbf{t}^* \ , \tag{3}$$

where $\mathbf{t}$ is the fake-quantized tensor, $\mathbf{t}^*$ is the integer-only tensor, and $\varepsilon_{\mathbf{t}}$ is the scale floating point factor. Therefore, the network can run entirely in the integer domain by multiplying and accumulating only integer values.

The CNN deployment is based on the open-source PULP-NN library [25], which encompasses optimized 8-bits fixed-point arithmetic kernels. In particular, it exploits the eight general-purpose cores of the CLUSTER of GAP8 SoC and the ISA-specific optimizations to maximize kernels' performance. Given a theoretical maximum of 32 MAC/cycle (4 MACs can be executed thanks to the int8 SIMD for each of the 8 cores), PULP-NN is able to reach a peak of 15.6 MAC/cycle for squared-size images. Note that the library peak is obtained by perfectly re-using data in the register file, reducing the number of necessary loads and stores per each MAC. If no data is stored in the register file for data re-use, two loads would be required for MAC inputs and one store to save the result in memory, reducing the theoretical peak to 8 MAC/cycle. On the other hand, the PULP-NN kernels consider data to be stored in the shared L1 64 kB memory, relegating their applicability to small layers.

TABLE I
TEST SET EXPERIMENT RESULTS

Network	MAE				R2 score			
	x	y	z	ϕ	x	y	z	ϕ
Trivial	0.46	0.61	0.17	0.55	0.0	0.0	0.0	0.0
$\mathbf{F}_{SoA}$ **[2]**	0.19	0.18	0.09	0.44	0.80	0.65	0.55	0.26
$\mathbf{F}_{small}$	0.20	0.20	0.09	0.44	0.79	0.65	0.51	0.24
$\mathbf{M}^{0.25}_{small}$	**0.17**	0.20	0.08	0.44	**0.84**	0.63	0.62	0.22
$\mathbf{M}^{1.0}_{small}$	0.17	**0.18**	**0.07**	**0.42**	0.83	**0.67**	**0.62**	**0.28**

To cope with this constraint, we employ a second open-source tool, DORY [17]. This tool automatically produces template-based C code, which wraps the PULP-NN kernels, managing the different levels of memories of GAP8 (i.e., L1, L2, and the external RAM) and orchestrating the tensors' movements. In particular, DORY exploits a tiling approach to separate layers into small nodes whose tensors can fit the L1 memory of the system. Thanks to this, the kernels can be directly executed in these nodes. DORY produces indeed the C routines, which i) execute the kernels of the small nodes with tensors stored in L1, and ii) double-buffer the movements of tensors from L2 to L1, to always have data ready for kernel execution. Notice that since the DMA is not blocking, the calls to the kernels are always overlapped with the asynchronous DMA calls.

Closed-loop control. We deploy our models as part of the same closed-loop control system from [2], which allows to maintain the drone at a desired 3D pose in front of the person. Four main components are involved: *i)* the CNN model outputs a point-in-time pose estimate from a single image, *ii)* a Kalman filter produces smooth temporally-consistent sequences of poses, and *iii)* a velocity controller computes the drone's desired pose from the subject's current pose and generates velocity setpoints to bring the drone to the desired pose. *iv)* the Crazyflie's low-level control, responsible for motor actuation and stabilization. We adopt a Kalman filter decoupled between the model's four outputs, by assuming diagonal process and observation noise covariance matrices. The velocity controller is also decoupled between linear velocity control, whose goal is to reach the specified target position, and angular velocity control, which keeps the subject centered in the image frame.

IV. RESULTS

A. NAS Pareto analysis

In Figure 2, we show and analyze the architectures found by our NAS algorithm. We compare on one axis the inference latency (number of clock cycles), whereas, on the other, we report the mean absolute error (MAE), expressed as the sum of L1 errors on (x, y, z, ϕ) between the networks predicted

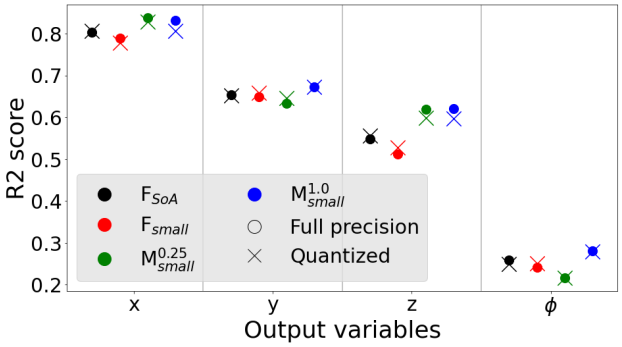

Fig. 3. R2 score of the four deployed models (higher is better).

values and the ground truth. The *trivial predictor*, i.e., a network that always predicts each output as its mean value in the test set, represents a lower bound to the models' MAE. From our three seed network architectures, the NAS search discovers eight new models, most of which lie on the global Pareto front in the space of MAE vs. the number of cycles. The found architectures range from 1.27M to 3.69M execution cycles, with corresponding MAE values from 1.31 to 0.84. In detail, PULP-Frontnet models occupy the left-most section of the Pareto curve, being very lightweight but less accurate. The middle is populated by models derived from the MobileNet 0.25× seed. Finally, the most accurate architectures are derived from MobileNet 1.0×. This seed architecture is too big to fit in the GAP8's L2 memory. However, our NAS algorithm can shrink it enough to find great solutions: absolute top-performing models that only increase latency by 15% compared to F_{SoA} [2].

From the global Pareto front, we select four models to deploy and further analyze. The first is F_{SoA}, which corresponds to the original PULP-Frontnet, the current state-of-the-art, and our baseline. The F_{small} architecture is the fastest model, still achieving a MAE roughly equivalent to F_{SoA}. $M^{1.0}_{small}$ is the most accurate architecture found by the NAS, but also the most expensive in terms of latency. $M^{0.25}_{small}$ represents the most balanced trade-off between the two metrics, with both better than or equivalent to F_{SoA}. In the following sections, we further analyze the selected architectures' performance on the test set and their behavior in the field in a closed-loop system.

B. Regression performance

In Table I and Figure 3, we break down the models' regression performance in the four output variables, (x, y, z, ϕ). For each output, in addition to the MAE values, we report the coefficient of determination R^2, a standard adimensional metric that represents the fraction of variance in the target variable explained by the model[2]. Compared to the MAE, the R^2 score quantifies the quality of a regressor independently of the target variable's variance and is, therefore, better suited for comparing regression performance between different variables. Performance on all outputs shows trends consistent with those on the aggregated loss, confirming $M^{1.0}_{small}$ as the top performing on all outputs except for x, where $M^{0.25}_{small}$ performs slightly better. In Figure 3, we see that all models can predict x and y best while ϕ is the most difficult to estimate, matching the findings of background work.

[2] $R^2 = 1 - \sum_i (y_i - \hat{y}_i)^2 / \sum_i (y_i - \bar{y})^2$ with y_i the ground-truth output and $\hat{y}_i$ the model prediction for each test sample i, $\bar{y}$ the mean of ground-truth outputs. An $R2 = 1.0$ corresponds to a perfect regressor, while the trivial regressor achieves $R2 = 0.0$.

TABLE II
COMPUTATION AND MEMORY FOOTPRINT FOR INFERENCE ON ONE FRAME (**F**: PULP-FRONTNET, **M**: MOBILENET).

Network	Params	MAC	Cycles	Memory	P [mW]	T [fps]
$\mathbf{F}_{SoA}$ [2]	304 k	14.7 M	3.2 M	499 kB	92.2	45.3
$\mathbf{F}_{small}$	44 k	7.6 M	1.5 M	231 kB	81.3	71.6
$\mathbf{M}^{0.25}_{small}$	65 k	7.4 M	2.2 M	591 kB	86.9	51.2
$\mathbf{M}^{1.0}_{small}$	54 k	12.4 M	3.7 M	415 kB	88.3	32.7

C. Onboard performance assessment

We assess the onboard performance of the CNNs selected in the previous Section IV-A. To nail down the computational, power, and memory requirements, we profiled our models, running them on the GAP8 SoC and using a RocketLogger data logger [26] (64 ksps). For these experiments, we test three SoC operating points: minimum power consumption with FC@25 MHz CL@25 MHz VDD@1 V, most energy-efficient with FC@25 MHz CL@75 MHz VDD@1 V, and maximum performance with FC@250 MHz CL@170 MHz VDD@1.2 V, as shown for the PULP-Frontnet baseline [2]. Table II summarizes the analysis on the four CNNs, showing that NAS models significantly reduce parameters (up to -85% w.r.t. F_{SoA}), MACs, clock cycles, and memory (all three around -50%). Power and throughput figures for the operative point used in the in-field experiments (max. performance) are also reported. Figure 4 shows the relation between power and throughput, across the three operative points. We see that the two smaller models $M^{0.25}_{small}$ and F_{small} improve upon the baseline F_{SoA} in both regards, resulting in higher energy efficiency. Figure 5 breaks down the power usage of the entire system. Crazyflie electronics plus the AI-deck cost 4.8% of the total budget, almost saturating the power that can be allocated to sensing and computing [27]. As the GAP8 accounts for 24% of that, there is a clear benefit in best optimizing its workload.

D. In-field experimental evaluation

We further validate the proposed networks in a closed-loop, in-field experiment. We ask a human subject to follow a predefined path, while the drone is tasked to stay in front of them at eye level, at a distance of $\Delta = 1.3\,\mathrm{m}$. For consistency, we adopt the same test setup proposed by [2], with a 50 s path composed of 8 phases of increasing difficulty for the model (walking straight along different directions, walking along a curve, and rotating in place). To ensure repeatability between different runs of the experiment, we ask the subject to completely ignore the drone's behavior and instead synchronize each step to the beats of a metronome.

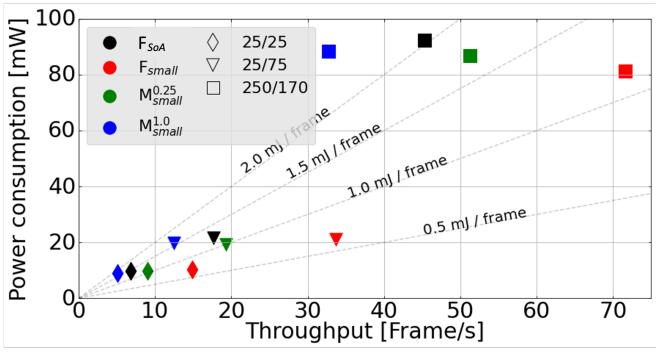

Fig. 4. Throughput vs. power consumption for the four models at various operative points.

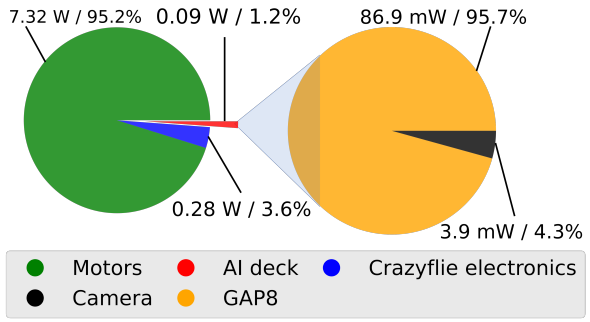

Fig. 5. Nano-drone's power breakdown, running the $M^{0.25}_{small}$ model.

The experiment is performed with both a subject and an environment outside of our training set, stressing the models' generalization capabilities. We perform four experiment runs for each of the four models, plus one baseline run in which the drone is controlled using perfect information about the subject's pose from a mocap system, a total of 17 test flights.

Table III summarizes our results in this experiment. We provide quantitative measures of three aspects of the system's performance: overall path completion, inference accuracy, and control accuracy. We measure path completion for each model in terms of total flight time over the four runs and mean percentage of path completed, interrupting a run as soon as the person completely leaves the camera field of view. In the challenging never-seen-before environment used for this experiment, the two PULP-Frontnet models struggle to complete the path (especially F_{small}, which reaches only 35% on average), while the two MobileNet models consistently maintain the tracking until the end. We mark metrics corresponding to incomplete runs with an asterisk because they cannot be directly compared with other runs.

Inference accuracy then evaluates the models in isolation, measuring their ability to correctly estimate the subject's position w.r.t. to the drone. As in the reference work, here we do not consider the z component because the target height is approximately constant in our task. We report the MAE, to allow direct comparison with offline performance on the test set reported in Section IV-B. As expected, absolute performance decreases significantly due to the different environments. At the same time, we see that the best in-field regression performance comes from $M^{0.25}_{small}$, instead of $M^{1.0}_{small}$ as on the test set. One explanation is the lower number of parameters in the former model, which makes it less prone to overfitting and thus generalize better.

TABLE III
IN-FIELD EXPERIMENT RESULTS

Network	Flight time [s]	Completed path [%]	MAE			Mean pose error	
			x	y	ϕ	e_{xy} [m]	e_θ [rad]
Mocap	165	100	0.0	0.0	0.0	0.18	0.21
F_{SoA} [2]	140	85	0.33*	0.12*	0.77*	0.72*	0.78*
F_{small}	58	35	0.81*	0.53*	0.55*	0.65*	0.42*
$M^{0.25}_{small}$	**165**	**100**	**0.25**	**0.11**	**0.52**	**0.49**	**0.59**
$M^{1.0}_{small}$	**165**	**100**	0.31	0.13	0.52	0.58	**0.59**

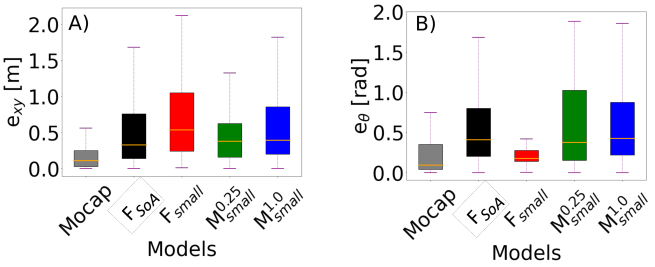

Fig. 6. In-field control errors distribution (lower is better). Boxplot whiskers mark the 5^{th} and 95^{th} percentile of data.

Finally, control accuracy evaluates the whole system's precision in tracking the subject as it moves along the path. We compare the drone's actual pose against the desired pose, measuring two errors: the absolute position error e_{xy} (i.e., the distance between the two poses in the horizontal plane) and the absolute angular error e_θ (i.e., the difference in orientation). $M^{0.25}_{small}$ is the best on both metrics, confirming itself as the best-performing in-field. In Figure 6, $M^{0.25}_{small}$ is the model with the lowest variance in absolute position error, further explaining its better performance. In addition, visually inspecting the drone's behavior shows that the $M^{0.25}_{small}$ model is noticeably more accurate than the baseline F_{SoA}. To complement our results, we provide a supplementary video of the four model's behavior in the in-field experiments.

V. Conclusion

This system paper presents a practical use case on NAS technologies applied for a challenging robotic visual perception human pose estimation task on nano-drones. Starting from two seed CNNs (i.e., PULP-Frontnet and MobileNetv1), we select four Pareto-optimal models to be deployed aboard a resource-constrained (i.e., sub-100 mW) nano-quadrotor. We assess the capabilities of the CNNs with a thorough analysis: from their regression performance on a disjoint test set, an onboard performance evaluation (power consumption and throughput), down to an in-field closed-loop test in a *never-seen-before* environment. Our best model improves the SoA by reducing the in-field control error of 32% with a real-time inference rate of ~50 Hz@90 mW.

REFERENCES

[1] D. Palossi, F. Conti, and L. Benini, "An open source and open hardware deep learning-powered visual navigation engine for autonomous nano-uavs," in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2019, pp. 604–611.

[2] D. Palossi, N. Zimmerman, A. Burrello, F. Conti, H. Muller, L. M. Gambardella, L. Benini, A. Giusti, and J. Guzzi, "Fully onboard ai-powered human-drone pose estimation on ultralow-power autonomous flying nano-uavs," *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 1913–1929, 2022.

[3] 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 (CVPR)*, June 2016.

[4] 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.

[5] 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.

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

[7] M. Risso, A. Burrello, F. Conti, L. Lamberti, Y. Chen, L. Benini, E. Macii, M. Poncino, and D. J. Pagliari, "Lightweight neural architecture search for temporal convolutional networks at the edge," *IEEE Transactions on Computers*, 2022.

[8] 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.

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

[10] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1586–1595.

[11] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen, *et al.*, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 965–12 974.

[12] G. Shi, W. Hönig, Y. Yue, and S.-J. Chung, "Neural-swarm: Decentralized close-proximity multirotor control using learned interactions," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 3241–3247.

[13] W. Zhao, J. Panerati, and A. P. Schoellig, "Learning-based bias correction for time difference of arrival ultra-wideband localization of resource-constrained mobile robots," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3639–3646, 2021.

[14] D. Palossi, F. Conti, and L. Benini, "An open source and open hardware deep learning-powered visual navigation engine for autonomous nano-UAVs," in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2019, pp. 604–611.

[15] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, Oct 2017.

[16] F. Conti, "Technical report: Nemo dnn quantization for deployment model," *arXiv preprint arXiv:2004.05930*, 2020.

[17] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus," *IEEE Transactions on Computers*, pp. 1–1, 2021.

[18] M. Navardi, A. Shiri, E. Humes, N. R. Waytowich, and T. Mohsenin, "An optimization framework for efficient vision-based autonomous drone navigation," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2022, pp. 304–307.

[19] L. Lamberti, V. Niculescu, M. Barciś, L. Bellone, E. Natalizio, L. Benini, and D. Palossi, "Tiny-pulp-dronets: Squeezing neural networks for faster and lighter inference on multi-tasking autonomous nano-drones," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2022, pp. 287–290.

[20] X. Sun, B. Xiao, F. Wei, S. Liang, and Y. Wei, "Integral human pose regression," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[21] D. C. Luvizon, D. Picard, and H. Tabia, "2d/3d pose estimation and action recognition using multitask deep learning," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5137–5146.

[22] R. A. Güler, N. Neverova, and I. Kokkinos, "Densepose: Dense human pose estimation in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[23] A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, "Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 871–875, 2020.

[24] J. Choi, S. Venkataramani, V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and efficient 2-bit quantized neural networks," in *Proceedings of the 2nd SysML Conference*, vol. 2019, 2019.

[25] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "Pulp-nn: A computing library for quantized neural network inference at the edge on risc-v based parallel ultra low power clusters," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2019, pp. 33–36.

[26] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele, "Rocketlogger: Mobile power logger for prototyping iot devices: Demo abstract," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, 2016, pp. 288–289.

[27] R. Wood, B. Finio, M. Karpelson, K. Ma, N. Pérez-Arancibia, P. Sreetharan, H. Tanaka, and J. Whitney, "Progress on 'pico' air vehicles," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1292–1302, 2012.