IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 25 July 2024, accepted 18 August 2024, date of publication 4 September 2024, date of current version 13 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3454363

RESEARCH ARTICLE

Design and Implementation of Tiny Deep Neural Networks for Landing Pad Detection on UAVs

EDOARDO RAGUSA, (Member, IEEE), TOMMASO TACCIOLI, ALESSIO CANEPA, RODOLFO ZUNINO, AND PAOLO GASTALDO

DITEN, University of Genoa, 16145 Genoa, Italy

Corresponding author: Edoardo Ragusa (edoardo.ragusa@unige.it)

ABSTRACT This paper presents a design paradigm to implement convolutional neural networks (CNNs) on low-power commercial microcontrollers for the detection of landing pads in small-size drone applications. A neural architecture search (NAS) strategy generates and selects CNN architectures automatically; candidate networks are compared in terms of their computing costs and representation capabilities. The proposed NAS procedure adopts a teacher-student learning paradigm, in which the 'student' network should mimic the 'teacher's' intermediate representation. The associate selection strategy aims to attain an efficient feature representation that can take into account the peculiarities of the problem at hand. This approach allowed the generation of tiny networks capable of real-time execution on commercial microcontrollers (STM32 family). Experimental results confirmed that the resulting architectures could trade off generalization capabilities and computing costs effectively, and outperformed state-of-the-art solutions for landing-pad detection in small-size drones.

INDEX TERMS Embedded systems, landing pad detection, microcontrollers, tiny CNNs, UAVs.

I. INTRODUCTION

Landing pad detection is a crucial problem in the operation of Unmanned Aerial Vehicles (UAV)s [1], [2]. Detecting safe landing areas by using simple passive tags can be critical for many applications [3], [4], [5], [6], [7], [8], including smart farm monitoring, rescue operations, home delivery, and autonomous guidance.

The development of a framework for landing pad detection in small-size UAVs poses two major issues. The first issue concerns the localization of small, colored patterns within a possibly complex background. Even if the target has a basic pattern and is very visible, the operation environment can make the detection goal quite challenging. Secondly, small-size UAVs can typically only host compact, low-power computing platforms, featuring limited computational resources. Cloud-based approaches might represent a viable solution for some applications. Nonetheless, two issues may discourage their adoption: the assumption of a stable and low-latency connection and the energy consumption involved for data transmission. Onboard landing pad detection in particular is critical because, in case of lost connections, the capability of locating a safe landing area is pivotal.

The approach presented in this paper tackles landing-pad detection as a computer vision problem and exploits the effectiveness of deep neural networks (DNNs). The main contribution is a design strategy for the development of lightweight neural architectures, that can fit the operational constraints set by standard commercial low-power microcontrollers. In the envisioned scenario, a reliable landing-pad detector should a) spot the landing pad in medium/long-distances images (e.g., 8 meters) and then b) trigger fine-grained control operations entitled to manage UAV landing. Hence, one wants the inference (forward) phase of the detector to run on onboard edge devices without power-demanding hardware accelerators or dedicated chips like vision processing units or deep learning dedicated processors, as latency requirements are not tight in this phase. Accordingly, fine-grained control operations can be implemented via standard computer vision methods, which

The associate editor coordinating the review of this manuscript and approving it for publication was Mohammad Hossein Moaiyeri.

can meet tight real-time requirements as they do not rely on computationally-demanding artificial intelligence models. In this paper, the target platform is the STM32F746NG[1] micro-controller, which embeds an ARM Cortex-M7 32-bit RISC core.

The deployment of DNNs on embedded devices has become recently a crucial issue, especially in view of the considerable computational resources required by DNN-based computer vision. Thus, while custom architectures are being developed to limit computational costs and memory occupation, several strategies and solutions [9], [10], [11], [12] aim to deploy trained DNNs on resource-constrained devices. This includes, for instance, pruning, quantization, and weight-sharing techniques. In combination with these approaches, always more efficient low-power architecture such as PULP and GAP8 allows low latency and low power results [13], [14], [15]. However, nowadays, most commercial small-size drones implement standard microcontrollers that grant computing performance far away from the results obtained by these dedicated processors. When targeting target devices such as a family of microprocessors, post-training strategies strictly depend on technology-specific deployment tools. For example, quantization techniques take into account the processor architecture and instruction set. In critical applications, however, one also needs to consider the training phase and the associate design of custom, lightweight architectures.

This paper describes a novel design strategy to design effective tiny networks for landing pad detection. The strategy combines knowledge distillation and Neural Architecture Search (NAS). An evolutionary algorithm procedure spawns a generation of candidate architectures (the Students). The neural training makes each Student adhere to a reference feature representation, provided by the Teacher. Students compete in terms of both recognition accuracy and compliance with hardware constraints; the most promising device-compatible architecture is the seed for the next generation of Students. This cyclic process progressively converges to an effective, hardware-compatible neural system. The competing architectures, generated via NAS, include Bottleneck Residual Blocks [16] as the network core components. The lightweight, yet powerful DNN MobileNetV3 [16] architecture supports the Teacher model, thanks to its known effectiveness in computer vision applications.

The experimental session aimed to assess the design capability of effective architectures, applied to landing-pad detection supported by an STM32F746NG micro-controller. The tests involved a dataset of aerial-like images. Experimental outcomes confirmed that the design strategy compared favorably with state-of-the-art solutions, yet featured lower computing requirements.

The contribution of the paper can be summarized as follows:

- an automatic strategy for the design of lightweight DNNs for landing pad detection: the proposed approach adjusts the network architecture based on the hardware requirements of the target device and a reference representation, ensuring a high level of robustness for the generated detectors while meeting time and energy constraints;
- the deployment on ARM Cortex M of the hardware-aware DNN supporting landing pad detection proves the capability of the proposed approach to generate hardware-effective models with real-time performance on commercial microcontrollers;
- a DNN model running on low-power commercial microcontroller that can meet latency requirements for landing pad detection on UAV while maintaining the capability of extracting robust features typically obtained by larger models.

II. RELATED WORKS

A. DNNs FOR EMBEDDED DEVICES

Two main aspects support the effective implementation of DNNs on embedded devices: 1) the architecture of the deep network and 2) the deployment optimization tools. The following analysis briefly reviews optimization tools and focuses on techniques for the design of efficient architectures.

The architecture of a DNN sets the memory footprint and the number of floating point operations (FLOPs) for the inference phase. When targeting implementations on resource-constrained embedded devices, one faces a crucial structural choice. Relying on state-of-the-art architectures allows transfer learning and can therefore lighten the training process significantly; on the other hand, this might make it challenging to fit the constraints imposed by the target embedded device. Available solutions include pruning [17], weights compression [10], quantization [18], low-rank approximation [19] and quantization-aware training [20]. The effectiveness of each strategy, though, depends on both the deployment tools and the target device itself [21], [22].

Otherwise, one might consider the design of custom architectures. Toward that end, one usually exploits consistent methods that co-optimize accuracy and hardware constraints [23], [24], such as Knowledge distillation [25], Neural Architecture Search (NAS) [26], [27], and their combination [28], [29]. These approaches led to the design of efficient architectures for computer vision [30], [31], [32]. Three main features characterize Hardware-Aware (HW)-NAS strategies [24]. The search space is the first issue, spanning the set of eligible networks, i.e. the set of all admissible architectures. From among the several proposals in the literature, Mobile (M)NAS space is a popular solution [33] designed on the blocks of MobileNet, hence the resulting candidate architectures only include computationally efficient building blocks. Secondly, the actual search algorithm adopted is another crucial aspect, which considers the various hypotheses by selecting architectures from the

[1] https://www.st.com/en/microcontrollers-microprocessors/stm32f746ng.html

search space. Finally, the choice of an effective evaluation criteria is of paramount importance, since hardware devices can vary in latency performances even when they support the same number of FLOPS and parameters [26]. Furthermore, software implementations can heavily affect speed, as the same device can score quite differently when it relies on either general-purpose solutions (such as TfLite and TfMicro) or optimized libraries [34], [35]. For that reason, the MCUnet solution [23] integrates both the selection of the architecture and the setup of the computing layer in one optimization procedure, thus attaining excellent performances at the expense of a 'closed' system [36].

Large-scale benchmarks offer a pre-computed evaluation of huge search spaces in the presence of different hardware setups. For example, the research presented in [37] compares 46k architectures deployed in 6 devices. A major issue in the NAS paradigm lies in the cost of architecture search; a popular approach consists in picking candidate networks from a unique 'super' network, which contains all possible architectures spanned by the search space at hand [38]. That strategy sharply reduces training time at the expense of a bias in the search procedure.

As compared with the approach presented in [39], the design strategy presented in this paper introduces a teacher network that simplifies the overall procedure, while measurements of FLOPS drive the architecture-selection process.

B. DNNs FOR THE NAVIGATION OF UAVs

Bringing intelligence to small-size UAVs is a challenging task due to the limited hardware resources [40]. Popular, cloud-based solutions delegate expensive computations to remote servers [41]; the literature offers cloud-based, object-detecting neural systems for UAVs [42], [43].

When one cannot rely on stable connections or is subject to low-latency inference timings, a platform featuring on-board intelligent behavior seems the preferred approach. Embedded solutions hosting hardware accelerators [44], such as the Jetson TX2 board [45], [46] or Raspberry SBCs, have been presented [47], but physical occupation, power consumption, and timing constraints still remain crucial issues in the overall design process. This is even more important in the presence of multiple smart sensors [48], [49].

Cameras represent the primary input sources to support the autonomous navigation of UAVs [50], [51], [52], [53], [54], also including the detection of a safe landing area. The method described in [55] and [47] applies conventional computer vision algorithms to extract relevant features and feed Deep Neural Networks (DNN) accordingly. In [56], a Convolutional Neural Network cooperates with a Kalman filter to control landing operations.

In the specific landing-pad detection context [57], the application domain and a simplified version of MobileNetV3 [16] allowed to overcome the (otherwise) computationally demanding requirements of the network. That research confirmed that the adoption of a segmentation network for the detection problem could sharply reduce computational costs.

Several approaches tackled landing-pad detection either by using a variety of additional sensors or by adopting specific landing pads. Tran et al. [58] proposed a design based on multiple markers. The method recently described in [59] relied on a custom landing pad containing specific RGB colors. That paper also reviewed a variety of landing systems based on traditional computer vision techniques, including edge detection [60], geometry-based template matching [61], and ArUco Markers [62], [63]. That comparative analysis confirmed that the use of DNNs for landing-pad detection could deserve further investigation.

III. AUTOMATED DESIGN OF EFFICIENT DNNs FOR LANDING-PAD DETECTION

The design approach presented in this paper yields a lightweight DNN architecture, which processes the frames captured by the UAV's onboard camera and prompts the coordinates of the landing pad in the image plane. The iterative design strategy embeds the Knowledge Distillation paradigm within a Neural Architecture Search process. The idea is to start from a reference ''Teacher'' model, which is indeed best performing but typically proves cumbersome to implement on low-performance microcontrollers. The method progressively searches for the best ''Student'' architecture that replicates the Teacher's behavior and, at the same time, satisfies computational constraints.

The search space is a refined version of MNAS. The overall process relies on a straightforward evolutionary algorithm, which combines simplicity and effectiveness. Empirical evidence justifies the adoption of FLOPS as the basic evaluation parameter because the target device exhibits a quasi-linear correlation between FLOPS and run-time inference timings [32]. This is due to the single computing core within the device, which forces a sequential flow of operations, making FLOPS a reliable estimator for this specific case.

A. KNOWLEDGE DISTILLATION

In knowledge distillation, a 'Student' architecture is optimized to replicate the behavior of a 'Teacher' model. In the target application, a Teacher's architecture includes a *backbone* to carry out feature extraction, and a *head*.

In the specific case of landing-pad detection, the head component processes those features and prompts the coordinates of the object of interest. The Teacher model looks for a simple pattern, and high-level semantic information (usually residing in the head section) is less critical. As a consequence, the intermediate layers in the backbone may already work out the relevant features. The overall strategy, therefore, is to select the Student that best replicates the behavior of the Teacher's backbone.

To optimize the weights of the Student one measures the discrepancy between the Teacher's and the Student's responses to the same input. The associate cost compares

the internal feature representations of the Teacher and of the Student at selected corresponding locations of the networks. The Teacher's backbone relies on a fixed architecture; one denotes, for the $i-th$ input, as $\mathcal{F}_{Ii}^{T}$ and $\mathcal{F}_{Hi}^{T}$ the representations at some intermediate level and at the higher level, respectively. Those values are compared with the corresponding feature values, $\mathcal{F}_{Ii}^{S}$ and $\mathcal{F}_{Hi}^{S}$ in the Student model. Figure 1 exemplifies the comparative approach.

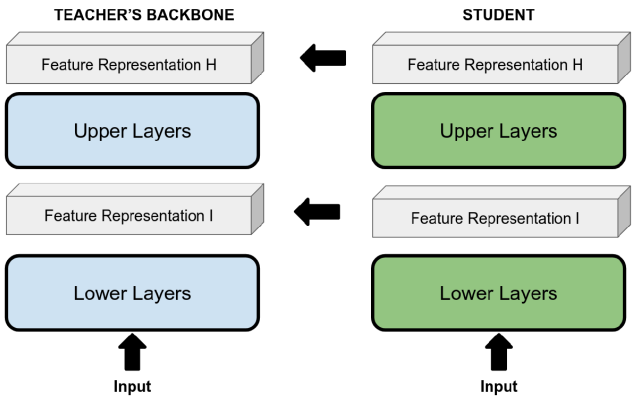


FIGURE 1. Block-wise representation of the CNNs with the features maps. On the left is the Teacher's backbone which produces two representations (tensors). On the right is the Student network that aims to provide the same feature sets, using different building blocks.

The following notation helps formalize the cost D:

- $\mathcal{X}_i \in R^{H\times W\times 3}$ is the tensor representation of the i-th input image (having size $H \times W$).
- $\mathcal{F}_{li} \in R^{H_l\times W_l\times C_l}$ is the tensor that holds the output feature representations at the l-th layer of the neural network, where H_l, W_l, C_l are the height, width and number of channels of the tensor, respectively.
- V$=\ \{(\mathcal{X}_i, y_i); i = 1, .., Z\}$ is the dataset, i.e., a collection of images and labels that are never involved in the training of both the Teacher and the Students.

The overall distillation loss, $\mathcal{L}_D$, is the expected discrepancy value, measured at the selected positions in the networks:

$$\mathcal{L}_D = A_D(|\mathcal{F}_{Ii}^{T} - \mathcal{F}_{Ii}^{S}| + |\mathcal{F}_{Hi}^{T} - \mathcal{F}_{Hi}^{S}|) \qquad (1)$$

where $A_D()$ denotes the average over a validation set. Figure 2 outlines the knowledge-distillation approach. The distillation loss function (1) makes the Student's intermediate representation match the Teacher model's backbone.

B. NEURAL ARCHITECTURE SEARCH

The NAS process gathers a set of lightweight, candidate architectures that should replace the Teacher's backbone. For its definition, one should consider the search space, the search strategy, and the selection criteria.

The search space is in fact vast, as one (mild) constraint simply admits any architecture that includes a pair of main blocks (i.e., lower and upper layers, respectively). To shrink

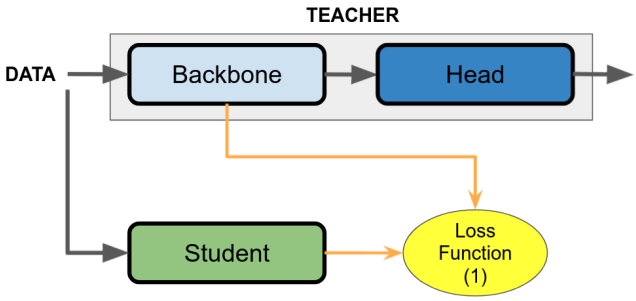


FIGURE 2. Proposed teacher/student learning schemata.

that space, the design strategy only considers a sequential combination of parametric building blocks, made by single-branch neural networks. This conventional setup [26], however, cannot by itself support an extensive search, since typical building blocks involve up to 6 parameters.

The search strategy selects the best Student architecture and embeds a basic evolutionary algorithm (Figure 3). Given a 'parent' architecture, the process yields a set of N 'child' networks by applying random mutations to the parent. Admissible mutations include changes in the parameters of the blocks or in the number of blocks itself. Child architectures all undergo a distillation-based training procedure. The selection criteria highlights the best Student from among the N children. The selected candidate now plays as the new parent architecture, and the whole selection strategy iterates until a stop condition is fulfilled (that condition will be detailed in the following).

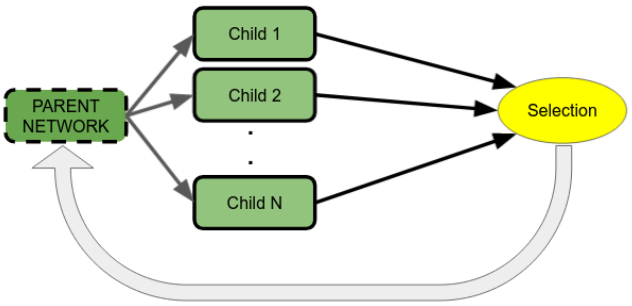


FIGURE 3. Block scheme of the evolutionary algorithm.

The adoption of a straightforward evolutionary algorithm stems from several reasons. First, the training of a child network is fast as lightweight models are involved. Secondly, an evolutionary algorithm supports any kind of mutation in the network architecture and is not subject to the optimization procedure. In addition, random mutations allow a wider, unbiased exploration of the search space as compared with other NAS approaches [64]. Fourth, the evolutionary algorithm can admit a non-differentiable cost criterion for the child-comparison task. This is a major advantage when considering that formalizing hardware constraints explicitly

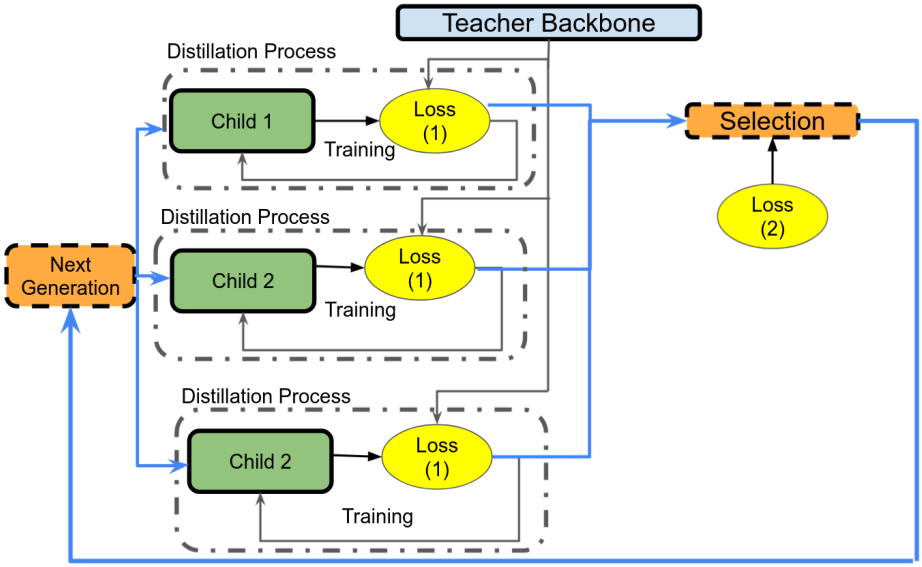


FIGURE 4. The integrated design strategy.

may prove quite difficult: for instance, the relationship between the network architecture and the corresponding RAM occupation might also depend on the optimization tool used to deploy the overall architecture itself. Finally, an independent selection process can indirectly help the children-training procedure. Common practice suggests that child networks are trained for a limited number of epochs, mostly to speed up the search procedure; widening the search area in the space of architectures can integrate the basic weight-adjustment process. Eventually, a large set of strategies has been proposed to speed up search procedures [65], which could be useful for increasing the number of tested models or reducing the time required for the selection process.

The selection criteria is the last issue to consider to define the overall NAS process. It relies on a cost function that integrates the Teacher-Student representation mismatch and the associate computational cost, measured in Floating Point Operations per Second (FLOPS). The resulting overall cost function, S, for the $n-th$ model is therefore written as:

$$S_n = \mathcal{L}_{Dn}, +\beta C_n \tag{2}$$

where $\mathcal{L}_{Dn}$ measures the discrepancy between the Teacher and the n-th candidate Student as per (1), C_n is the computational cost in FLOPS associated with that candidate, and β is a parameter that rules the relative weights of the two terms.

Measuring the computational cost in FLOPS might appear, in general, insufficient to characterize the hardware awareness of a Student model. In the present context, however, it is a significant indicator when considering the architectures of basic micro-controllers, which support limited or null parallel computation.

C. INTEGRATED NEURAL ARCHITECTURE SEARCH WITH KNOWLEDGE DISTILLATION

A cyclic process combines the paradigms discussed above, and supports the progressive performance-driven neural architecture search; the overall approach is depicted in figure 4 and evolves as follows:

1) The next generation block generates a set of candidate children networks introducing a mutation into a parent architecture;
2) In compliance with the Knowledge-Distillation paradigm, the set of candidate Student architectures are trained (as per function 1) to approximate the Teacher's behavior; in figure 4 the dashed boxes identify the training procedures of the children;
3) The trained children are compared, by also taking into account the computational complexity of each candidate as per expression 2 by the selection block;
4) The resulting 'best' child architecture is used to spawn the next generation of candidates as per step 1, which undergoes the same selection process.

For simplicity, the stopping condition in the above cyclic procedure just relies on a preset number of iterations. The procedure yields a lightweight backbone that can work out similar features with respect to those extracted by a large network. The backbone is finally retrained in an end-to-end fashion together with a head on the target landing-pad detection problem.

IV. DEPLOYMENT OF THE LANDING PAD DETECTOR

A. EDGE DEVICES

The reference platform for the deployment of the landing pad detector is the STM32F746NG microcontroller, which features an ARM® Cortex-M7 32-bit RISC core. The microcontroller unit (MCU) operates at up to 216 MHz and

includes a single-precision floating point unit that supports all ARM® single-precision data-processing instructions. It holds 1 Mbyte of Flash memory and 320 Kbytes of SRAM. The device also supports external memory access, which can be efficiently used via a Flexible Memory Controller (FMC) and either standard or advanced communication channels. The device extends on an area of 13.15mm × 13.15mm. To cope with memory requirements, an 8 Mbyte SDRAM was added.

The ARM® Cortex-M7 is a member of the energy-efficient Cortex-M processor family. Cortex-M targets low-power applications featuring reduced clock frequencies (up to a few hundreds of MHz) and supporting the indexing of small-size memories (up to a few MB). Hence, the deployment of DNN-based computer-vision solutions on such processors represented a challenge. ST's software tool to optimize artificial neural networks on STM32 (STM32 X-Cube-AI) only supports the deployment of very tiny architectures; this also holds true in the case of TensorFlow Lite for microcontrollers.

In the experimental session (Sec. V) the landing pad detector was also deployed on a processor of the ARM Cortex-A family, to provide a baseline for comparison. The processors of the A (Application level) family feature high clock frequencies and support the indexing of large memories in the range of GB. Those processors are typically hosted in microcomputers and smartphones. The specific device used for the experiments was the 1.5 GHz quad-core Arm® Cortex®-A72 CPU that supports the Raspberry Pi 4.

B. HARDWARE-AWARE LANDING PAD DETECTOR

To cope with the tight constraints imposed by ARM® Cortex-M7 MCUs the Teacher model was inherited from [57]. In [57], the detection task was suitably approached as a pixel-wise problem. The network's output was a mask that marked the pixels belonging to the landing pad. That approach allowed us to rely on the lightweight LR-ASPP head for image segmentation and to avoid the popular, yet computationally hungry single-shot detectors (SSDs). Eventually, applying simple heuristics on the output mask provided the coordinates of the landing pad [57].

This architecture was selected as a reference for the Teacher model as the empirical evidence suggested that the network can detect landing pads with high accuracy while restraining computing requirements [57]. The ablation study presented in [57] showed that, given the small set of bottleneck layers, the squeeze and excite layers could be removed without significant loss in accuracy. The landing pad detector was implemented with a network with 60,612 parameters that required 1.289 GFlops to process an input image of size 320 × 320 pixels. In the approach presented here, the candidate Student models only involved Bottleneck Residual Blocks, which admitted different settings for the number of filters, the kernel size, the expansion value, the stride, and the non-linear component. The loss function 2 drove the distillation process.

The severe constraints set by the STM32F746NG microcontroller imposed to customize also the segmentation head of the final detector, which in principle should inherit the LR-ASSP architecture. This simple segmentation head involved a few convolutional layers and upper sampling layers. The overall architecture is presented in Fig. 5: green blocks refer to the input tensors coming from the backbone, while the red block refers to the output mask. Input images have a size of 320 × 320. From a computational viewpoint, the scheme highlights three main bottlenecks:

- The number of filters in the lowest convolutional layers (marked in red); this quantity not only sets the number of parameters and operations for the two layers but also impacts the size of tensors in the following layer.
- The size of the output mask (marked in yellow), which in the original implementation halves the size of the input image. In principle, one can use a smaller output mask to cut the number of operations.
- The size of the input image, which heavily affects the number of floating point operations.

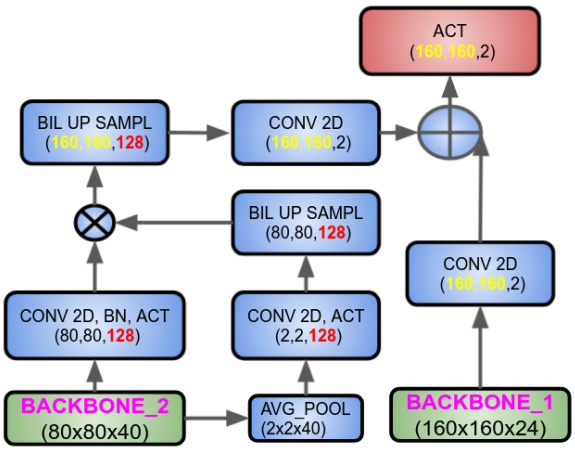


FIGURE 5. Block scheme of the teacher segmentation head [30].

In the proposed implementation, input images held 160 × 160 pixels; the number of filters was set to 8 (instead of 128), as the landing pad had a simple shape. The size of the output mask was four times smaller than the size of the input image.

V. EXPERIMENTS

The experimental dataset included 21 videos of landing pads and featured a total of 29,415 frames. The videos were grabbed at two different heights, approximately 4 and 8 meters from the rooftop of two buildings using a GoPro camera. The dataset covered the same three landing pads tested in [57], two in color (orange and blue), and one in grayscale, all with a black 'H' mark printed on a standard A4 sheet (sample images of the landing pads are available in [57]). A collection of 9,200 frames (drawn from 9 of those videos) formed the training sets. The images from the remaining 12 videos composed the test set.

The labeling procedure was divided into two phases. First, a proposed mask was generated using computer vision algorithms and large-scale networks. Then, the obtained labels were reviewed and corrected by the authors when necessary.

A. DISTILLATION

The distillation procedure was implemented by using Keras. The Teacher network was trained on the dataset for 20 epochs. The candidate children networks were trained for 4 epochs, following the early stopping paradigm to speed up NAS procedure [33]; batch size was set to 4, and the learning rate was 1e-4. Every pool of child networks included 6 candidates; each candidate stemmed from two mutations of the parent network. The distillation procedure was performed considering images having size 320×320; the original segmentation head of the teacher networks relied on a 160×160 segmentation mask. The number of filters was 128 as per the original implementation.

TABLE 1. Architecture summary of the first parent architecture in the NAS procedure.

Input size	Kernel size	Filters	Expsize	Act.	stride
320x320x3	3x3	24	72	Relu	2
160x160x24	5x5	40	96	h-swish	2

The first parent network featured a tiny architecture holding a pair of Bottleneck Residual Blocks. Table 1 gives the settings of the two layers; the columns give, for each layer, the size of the input tensor, the kernel size, the number of filters, the expansion factor (Expsize), the activation function, and the stride. Using a tiny architecture at the beginning of the procedure is a popular approach called *hot start* [66]. In practice, the search process is expected to progressively increase the complexity of the architecture while targeting a higher accuracy.

The parameter β trades off the quantities D_n and C_n in the cost function 2. Those terms typically differ significantly in their orders of magnitude: D_n varies in the order of units, whereas C_n may range from millions to billions. In the experiments, β was set using the following equation:

$$\beta = \frac{D_0}{C_0}\beta_E \tag{3}$$

where β_E sets the actual trade-off between the two terms in eq. 2, D_0 and C_0 are two normalization terms: when $\beta = \frac{D_0}{C_0}$ the contributions of the two terms are equal. The values D_0 and C_0 are the respective cost terms in 2 worked out for the Teacher architecture computed after 4 epochs of training, i.e. the training epochs used for Students architectures.

Two distillation experiments were conducted for as many different settings of the parameter β_E in the loss function 2:

- Balanced configuration ($\beta_E = 1$). In this experiment, the stopping criterion for the distillation process was set to 50 iterations. Therefore, a total of 300 children architectures were evaluated. Eventually, the selected Student network included three Bottleneck Residual Blocks; the number of parameters of the resulting network composed of backbone and segmentation head (LR-ASSP) was 39,012, while an inference required 648 MFlops. Table 2 gives all the details about the selected architecture (denoted as *BAL_BB*) by adopting the same format of Table 1. In practice, starting from a parent network with two blocks, a block was added to find a trade-off between the discrepancy D and the computation cost C (since $\beta_E = 1$).
- Small architecture ($\beta_E = 10$). This experiment privileged the minimization of the computation component, C. The number of iterations in the distillation process was set to 100 because the child networks were smaller in size and the training phases were faster. A total of 600 architectures were evaluated. Again, the selected Student network involved three Bottleneck Residual Blocks; in this case, the number of parameters of the resulting network composed of backbone and segmentation head (LR-ASSP) amounted to 23,232 and one inference step required 283 MFlops. Table 3 gives the architecture details and adopts the same format as Table 1. Notably, the first block uses a kernel of size 1×1, while in general low-level kernels have larger sizes in standard architectures [67]. In the following, this architecture will be referenced to as *SMALL_BB*.

TABLE 2. Balanced architecture distilled with the proposed method.

Input size	Kernel size	Filters	Expsize	Act.	stride
320×320x3	2x2	24	72	Relu	2
160×160x24	5x5	46	96	Relu	2
80×80x46	5x5	40	128	Relu	1

TABLE 3. Small architecture distilled with the proposed method.

Input size	Kernel size	Filters	Expsize	Act.	stride
320×320x3	1x1	24	10	Relu	2
160×160X24	5x5	40	40	Relu	2
80×80x40	6x6	40	56	Relu	1

B. GENERALIZATION PERFORMANCE OF THE LANDING PAD DETECTOR

The experiment assessed the generalization performance of the architectures embedding the two Student networks discussed above. In total, four architectures for landing pad detection were tested:

1) *BALANCED*: this architecture stacked the original segmentation head (LR-ASSP, as per Fig. 5) on the BAL_BB backbone.
2) *SMALL*: this architecture stacked the original segmentation head (LR-ASSP) on the SMALL_BB backbone.
3) *STM32*: this architecture stacked the customized version of the segmentation head described in Sec. IV on the SMALL_BB backbone.

4) *STM32_TINY*: this architecture stacked the customized version of the segmentation head described in Sec. IV on a customized version of the SMALL_BB backbone, where the expansion factor has been divided by 2. In this way, the number of floating point operations has been reduced without changing the size of the tensors propagated through the architecture.

The input size for the two latter architectures was set to 160 × 160 pixels instead of 320 × 320 pixels. The images of size 160 × 160 were obtained by zooming the original images maintaining the definition of the target landing pad in the image. All the networks were trained for 20 epochs; performance analysis involved a test set of 20,215 images never used in the training process.

The four networks were compared with three recent baselines suitable for solving the same task. The first baseline was the architecture adopted as the Teacher model [57]. The second baseline was the network proposed in [47], which supports recognition and classification of the landing pad on an edge device embedded in the drone. Finally, the third baseline was the SSD-lite MobileNetV3 architecture, i.e., a general-purpose object detector that suits an edge paradigm ([42], [43]). The Teacher network was trained using the aforementioned settings. The detection method proposed in [47] was replicated following the setup of the original paper. The SSD-lite MobileNetV3 architecture pre-trained on coco-dataset was fine-tuned for 20 epochs with a learning rate $10^{-}3$ and ADAM optimizer. Hyper-parameters were set using a subset of the training data as a validation set.

The landing pad detectors supported by the four architectures described above extracted a probability mask, i.e., a mask containing the probability of being a landing pad for each pixel. The mask was converted into a box using a blob search algorithm. In general, a mask can generate multiple boxes. Here, only the largest box was considered. This setup corresponded to a worst-case analysis.

Figure 6 gives the results of the experiments involving the seven different implementations of the landing pad detector. On the x-axis the plot gives the confidence threshold, i.e., the minimum probability level for a pixel to be classified as a landing pad. The y-axis gives the percentage of true positives (TPs), i.e., images where the Intersection over Union (IoU) between prediction and ground truth was higher than 0. Threshold 0 was set because landing pads cover a small portion of the image therefore even small values of IoU identify valid detections. The plot compares the performance of the four architectures described above with those achieved by three baselines. In this plot, the two baseline methods that did not exploit a pixel-wise classification are characterized by a constant TP percentage.

The results confirmed that a general-purpose object detector with backbones for general-purpose computer vision represents a valuable option for high-accuracy detections. Similarly, traditional computer vision pipelines, even being more efficient, feature lower generalization capabilities. The teacher network [57] is a suitable option to extract

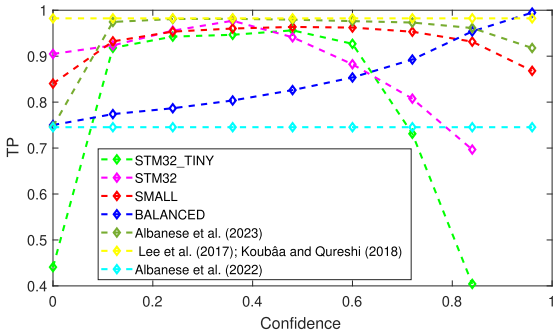


FIGURE 6. Generalization performance of the architectures under analysis: NSE [57], NAS-1 tiny architecture, NAS-2 balanced architecture.

effective features, for intermediate confidence values. The four distilled networks show two trends. The BALANCED network improves monotonically its performance when the confidence threshold grows. With high confidence levels, the network becomes the most accurate among all the predictions. The results suggest that using a smaller set of parameters acted as a regularizer. The three versions based on the small backbone exhibit a similar trend. Interestingly, the difference with respect to the teacher becomes very high for low or high confidence levels, but for intermediate values, the drop in accuracy remains low.

Figure 7 shows the standard precision-recall plot obtained by varying the confidence levels. An image was considered a false negative (FN) when the detector did not identify any landing pad and was classified as a false positive (FP) when IoU was zero, i.e., the network identified the landing pad in the wrong position. In this plot, the x-axis gives the recall while the y-axis gives the precision. The chart focuses on a specific portion of the plot, i.e., precision and recall greater than 0.95. As in fig. 6, the plot compares the performance of the architectures under test with those achieved by baselines. The baseline [47] is not visible in the plot because it scored a Precision/Recall value always lower than (0.95, 0.95).

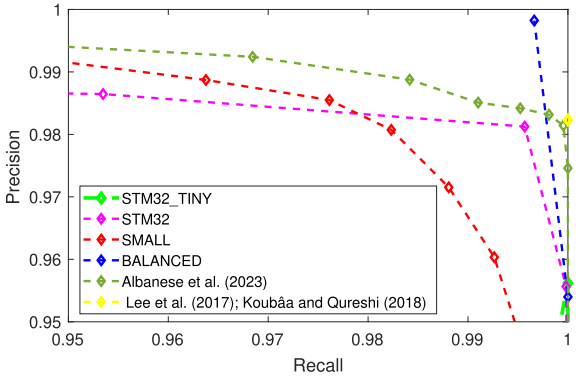


FIGURE 7. Generalization performance: precision-recall analysis.

The results confirm that all the solutions based on deep neural networks yield a high level of precision and recall for at least one confidence value, confirming the suitability of the

proposed methodology to detect landing pads from medium distances.

C. COMPUTATIONAL PERFORMANCE

The computational performance of the four proposed implementations of a landing pad detector has been evaluated by using as reference the Cortex A and Cortex M platforms, as anticipated in Sec. IV.

Two commercial tools supported the deployment of the architectures on the two platforms. The deployment on RaspberryPi 4 (featuring a Cortex A core) was obtained using the TFLite suit.

The deployment on the STM32 microcontroller (featuring a Cortex M core) was performed in two steps. First, the network was converted via TFLite; then, the STM32 X-Cube-AI suit was exploited to optimize the model. Eventually, the memory indexing was tuned to use the external memory, when necessary, to host the tensors during the propagation along the layers of the network. Data representation was set to 32-bit. All the measurements were performed using the STM32 design suite utility for testing.

In both cases, a standard 32-bit floating point representation was considered. This choice allows for a fair comparison between the two devices that use different computing engines. Indeed, the documentation of the two SDKs highlights different management of the quantized models based on the selected quantization strategy and layers involved in the network architecture, which would bias the measured performance. Ultimately, the measured performance can be considered a worst-case analysis for both devices, given that quantization can improve performance in terms of latency.

Table 4 summarizes the outcome of the experiments. The first two columns (in red) give the network name and the input size (in pixels), respectively. The third and fourth columns (in yellow) show the computational features of the network that are independent of the implementation, i.e., the FLOPS and the number of parameters. Columns from fifth to seventh (in blue) refer to the implementation on the STM32 microcontroller. These columns give the latency in seconds (per frame), the amount of flash memory required to store the parameters, and the peak of RAM usage. Finally, the last column (in green) refers to the implementation on RaspberryPi. This column shows the latency in milliseconds (per frame). Since the available memory in the RaspberryPi platform was abundant for the networks involved in the experiment, the table does not provide any information about memory usage.

The table compares the four proposed implementations (BALANCED, SMALL, STM32, STM32_TINY) with the model proposed in [57]. In addition, as a reference, the last row gives the FLOPS and the number of parameters of the SSD architecture for object detection. SSD was not deployed on the devices; the table refers to the original implementation available in Tensorflow v1 object detection API model zoo.[2]

[2]https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md

Eventually the deployment would have led to biased results due to the different implementation. In addition, one can see that both the number of FLOPS and parameters are larger than the other network involved in the experiments therefore hardware requirements are likely to be significantly larger.

The proposed architectures can save parameters and FLOPS operations (yellow indicators). The STM32TINY network is the smallest among the proposed ones, it uses one-tenth of the parameters of the Teacher model and around 0.08 of the FLOPS for both the input sizes. As expected RaspberryPi can support real-time landing pad detection using all the proposed architectures: even the balanced model, with the largest input size, is executed at 3 FPS without dedicated accelerators. The STM32 is the hardest benchmark. The number of FLOPS and the peak RAM usage are a direct function of the input size. Memory is set by the size of the largest tensor propagated inside the architecture. The lowest latency, as expected, has been achieved by STM32TINY, which completes the inference phase in 1.5 seconds. This is a noteworthy result for a computer vision task on a commercial microcontroller. The well-known recent work of Banbury et al. [36] designed DNNs for real-time inference on MCUs. The software release contained tiny object detector for ARM Cortex A but Cortex M was not supported.

At the end of the pipeline proposed in [47] a CNN classifies the landing pad. This model was not included in the table because it refers to a different pipeline. MobileNetV2 was one of the architectures tested in [47] for the classification stage. With input size 100×100 it has 3538K parameters and uses 156 MFLOPS for a single inference, proving less efficient than the proposed models

The frame rate on the STM32 was lower than 1 FPS which is incompatible with requirements imposed by some fine-grained control operations. Nonetheless, in the envisioned scenario, the low-power core equipped with the STM32 microcontroller monitors the ground in long-distance images. After the detection of a landing pad, the fine-grained control operations required to manage the UAV landing could be handled by a hardware accelerator activated only in that phase. Overall, such a solution can meet real-time requirements while limiting energy consumption, as UAV landing can be managed via a high-performance unit only for the time required to complete the task.

STM declares a peak power consumption of 345 mW for STM32F746NG. As the STM32TINY architecture scores an inference time of 1.5 s, the estimated energy is 0.518 J. In [47] land-pad detection was deployed on a RaspberryPi with a dedicated hardware accelerator. The reported energy consumption was 0.246 J at the expense of an average power consumption of 4.92 W. Therefore, the proposed implementation on MCU shows a modest increment in energy consumption (2x) and a large cutback in peak power consumption (14x). This result is interesting if one considers that a larger power peak forces the use of dedicated power supplies on the AUV.

TABLE 4. Hardware measures.

Model Name	Input Size	FLOPS (M)	PAR.S (K)	STM32 Lat. (s)	Flash (KiB)	Ram (MiB)	RASP Lat. (ms)
BALANCED	320	564	33	NaN	121.57	29.54	305.36
	160	141	33	8.4	121.57	7.4	106.5
SMALL	320	283	23	NaN	83.81	17.19	189.2
	160	70	23	4.2	83.81	4.3	58.4
STM32	320	203	12	11.2	44.44	7.52	135.3
	160	50	12	2.7	44.44	1.88	34.2
STM32 TINY	320	110	6	6.3	22.12	5.57	108.2
	160	27	6	1.5	22.12	1.39	32.5
Teacher [57]	320	1289	60	NaN	216.27	21.3	453.1
	160	322	60	17.5	216.27	5.22	164.9
SSD [42], [43]	320	3758	2760	-	-	-	-

VI. CONCLUSION

The paper presented a design strategy for landing pad detection on ARM cortex-based microprocessors. The strategy uses an automatic procedure to extract the neural network architecture balancing, in the best way, hardware requirements and generalization performance. The results confirmed the suitability of the proposed approach for real-time deployment on STM32 microcontrollers. Possible future work includes: 1) the use of quantization-aware training strategies directly in the NAS procedure to enhance the trade-off between hardware efficiency and the generalization performance of the models; 2) the exploration of faster search procedures to speed up architecture selection, which currently requires significant GPU time; 3) testing and/or adapting the proposed methodology for network generation to new applications that require a good level of abstraction on tiny architectures.

REFERENCES

[1] C. G. Grlj, N. Krznar, and M. Pranjić, "A decade of UAV docking stations: A brief overview of mobile and fixed landing platforms," *Drones*, vol. 6, no. 1, p. 17, Jan. 2022.

[2] J. Uddin, M. F. Wadud, R. Ashrafi, M. G. R. Alam, and M. K. Rhaman, "Landing with confidence: The role of digital twin in UAV precision landing," in *Proc. 10th Int. Conf. Recent Adv. Air Space Technol. (RAST)*, Jun. 2023, pp. 1–6.

[3] F. V. Benitez, A. Rutherford, J. Van Hilst, E. Babcock, M. Schlief, and L. Markussen, "A vision-based approach to autonomous landing of an eVTOL aircraft in GPS-denied environments," in *Proc. IEEE/AIAA 42nd Digit. Avionics Syst. Conf. (DASC)*, Oct. 2023, pp. 1–6.

[4] E. Miccio, P. Veneruso, R. Opromolla, G. Fasano, C. Tiana, and G. Gentile, "AI-powered vision-aided navigation and ground obstacles detection for UAM approach and landing," in *Proc. IEEE/AIAA 42nd Digit. Avionics Syst. Conf. (DASC)*, Oct. 2023, pp. 1–10.

[5] A. Hafeez, M. A. Husain, S. P. Singh, A. Chauhan, M. T. Khan, N. Kumar, A. Chauhan, and S. K. Soni, "Implementation of drone technology for farm monitoring & pesticide spraying: A review," *Inf. Process. Agricult.*, vol. 10, no. 2, pp. 192–203, Jun. 2023.

[6] L. Xing, X. Fan, Y. Dong, Z. Xiong, L. Xing, Y. Yang, H. Bai, and C. Zhou, "Multi-UAV cooperative system for search and rescue based on YOLOv5," *Int. J. Disaster Risk Reduction*, vol. 76, Jun. 2022, Art. no. 102972.

[7] L. P. Osco, J. M. Junior, A. P. M. Ramos, L. A. de Castro Jorge, S. N. Fatholahi, J. de Andrade Silva, E. T. Matsubara, H. Pistori, W. N. Gonçalves, and J. Li, "A review on deep learning in uav remote sensing," *Int. J. Appl. Earth Observ. Geoinf.*, vol. 102, Apr. 2021, Art. no. 102456.

[8] A. Arishi, K. Krishnan, and M. Arishi, "Machine learning approach for truck-drones based last-mile delivery in the era of industry 4.0," *Eng. Appl. Artif. Intell.*, vol. 116, Nov. 2022, Art. no. 105439.

[9] A. Burrello, M. Risso, B. A. Motetti, E. Macii, L. Benini, and D. J. Pagliari, "Enhancing neural architecture search with multiple hardware constraints for deep learning model deployment on tiny IoT devices," *IEEE Trans. Emerg. Topics Comput.*, pp. 1–15, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10278089

[10] E.-H. Yang, H. Amer, and Y. Jiang, "Compression helps deep learning in image classification," *Entropy*, vol. 23, no. 7, p. 881, Jul. 2021.

[11] S. S. Saha, S. S. Sandha, and M. Srivastava, "Machine learning for microcontroller-class hardware: A review," *IEEE Sensors J.*, vol. 22, no. 22, pp. 21362–21390, Nov. 2022.

[12] A. Canepa, "Application-aware optimization of artificial intelligence for deployment on resource constrained devices," Ph.D. dissertation, DITEN, Univ. Genoa, Genoa, Italy, 2023. [Online]. Available: https://iris.unige.it/handle/11567/1109034

[13] R. J. Bouwmeester, F. Paredes-Vallés, and G. C. H. E. de Croon, "NanoFlowNet: Real-time dense optical flow on a nano quadcopter," 2022, *arXiv:2209.06918*.

[14] S. Bonato, S. C. Lambertenghi, E. Cereda, A. Giusti, and D. Palossi, "Ultra-low power deep learning-based monocular relative localization onboard nano-quadrotors," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 3411–3417.

[15] L. Lamberti, V. Niculescu, M. Barcis, 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 *Proc. IEEE 4th Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2022, pp. 287–290.

[16] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314–1324.

[17] S. Ye, X. Feng, T. Zhang, X. Ma, S. Lin, Z. Li, K. Xu, W. Wen, S. Liu, J. Tang, M. Fardad, X. Lin, Y. Liu, and Y. Wang, "Progressive DNN compression: A key to achieve ultra-high weight pruning and quantization rates using ADMM," 2019, *arXiv:1903.09769*.

[18] E. Park, J. Ahn, and S. Yoo, "Weighted-entropy-based quantization for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7197–7205.

[19] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, *arXiv:1511.06530*.

[20] Y. Nahshan, B. Chmiel, C. Baskin, E. Zheltonozhskii, R. Banner, A. M. Bronstein, and A. Mendelson, "Loss aware post-training quantization," *Mach. Learn.*, vol. 110, nos. 11–12, pp. 3245–3262, Dec. 2021.

[21] A. Jain, S. Bhattacharya, M. Masuda, V. Sharma, and Y. Wang, "Efficient execution of quantized deep learning models: A compiler approach," 2020, *arXiv:2006.10226*.

[22] L. Capogrosso, F. Cunico, D. S. Cheng, F. Fummi, and M. Cristani, "A machine learning-oriented survey on tiny machine learning," 2023, *arXiv:2309.11932*.

[23] J. Lin, W. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proc. 34th Conf. Neural Inf. Process. Syst.*, 2020, pp. 11711–11722.

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

[25] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789–1819, Jun. 2021.

[26] L. L. Zhang, Y. Yang, Y. Jiang, W. Zhu, and Y. Liu, "Fast hardware-aware neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 692–693.

[27] A. M. Garavagno, E. Ragusa, A. Frisoli, and P. Gastaldo, "Running hardware-aware neural architecture search on embedded devices under 512MB of RAM," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2024, pp. 1–2.

[28] M. Kang, J. Mun, and B. Han, "Towards Oracle knowledge distillation with neural architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, 2020, pp. 4404–4411.

[29] F. Boutros, P. Siebke, M. Klemt, N. Damer, F. Kirchbuchner, and A. Kuijper, "PocketNet: Extreme lightweight face recognition network using neural architecture search and multistep knowledge distillation," *IEEE Access*, vol. 10, pp. 46823–46833, 2022.

[30] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," 2019, *arXiv:1905.02244*.

[31] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[32] E. Ragusa, S. Dosen, R. Zunino, and P. Gastaldo, "Affordance segmentation using tiny networks for sensing systems in wearable robotic devices," *IEEE Sensors J.*, vol. 23, no. 19, pp. 23916–23926, Oct. 2023.

[33] 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 *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2815–2823.

[34] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm Cortex-M CPUs," 2018, *arXiv:1801.06601*.

[35] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, L. Gan, G. Yang, and D. Qian, "The deep learning compiler: A comprehensive survey," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 708–727, Mar. 2021.

[36] 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," in *Proc. Mach. Learn. Syst.*, vol. 3, 2021, pp. 517–532.

[37] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, and Y. Lin, "HW-NAS-bench: Hardware-Aware neural architecture search benchmark," 2021, *arXiv:2103.10584*.

[38] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 544–560.

[39] A. Shaw, D. Hunter, F. Landola, and S. Sidhu, "SqueezeNAS: Fast neural architecture search for faster semantic segmentation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019. [Online]. Available: https://openaccess.thecvf.com/content_ICCVW_2019/html/NeurArch/Shaw_SqueezeNAS_Fast_Neural_Architecture_Search_for_Faster_Semantic_Segmentation_ICCVW_2019_paper.html

[40] S. Y. Choi and D. Cha, "Unmanned aerial vehicles using machine learning for autonomous flight; state-of-the-art," *Adv. Robot.*, vol. 33, no. 6, pp. 265–277, Mar. 2019.

[41] L. Abualigah, A. Diabat, P. Sumari, and A. H. Gandomi, "Applications, deployments, and integration of Internet of Drones (IoD): A review," *IEEE Sensors J.*, vol. 21, no. 22, pp. 25532–25546, Nov. 2021.

[42] J. Lee, J. Wang, D. Crandall, S. Šabanovic, and G. Fox, "Real-time, cloud-based object detection for unmanned aerial vehicles," in *Proc. 1st IEEE Int. Conf. Robot. Comput. (IRC)*, Apr. 2017, pp. 36–43.

[43] A. Koubaa and B. Qureshi, "DroneTrack: Cloud-based real-time object tracking using unmanned aerial vehicles over the internet," *IEEE Access*, vol. 6, pp. 13810–13824, 2018.

[44] L. Santoro, A. Albanese, M. Canova, M. Rossa, D. Fontanelli, and D. Brunelli, "A plug-and-play TinyML-based vision system for drone automatic landing," in *Proc. IEEE Int. Workshop Metrol. Ind. 4.0 IoT*, Jun. 2023, pp. 293–298.

[45] S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 2539–2544, Jul. 2018.

[46] N. Tijtgat, W. Van Ranst, B. Volckaert, T. Goedeme, and F. De Turck, "Embedded real-time object detection for a UAV warning system," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops (ICCVW)*, Oct. 2017, pp. 2110–2118.

[47] A. Albanese, M. Nardello, and D. Brunelli, "Low-power deep learning edge computing platform for resource constrained lightweight compact UAVs," *Sustain. Comput., Informat. Syst.*, vol. 34, Apr. 2022, Art. no. 100725.

[48] M. H. M. Ghazali and W. Rahiman, "Vibration-based fault detection in drone using artificial intelligence," *IEEE Sensors J.*, vol. 22, no. 9, pp. 8439–8448, May 2022.

[49] S. Gupta, P. K. Rai, A. Kumar, P. K. Yalavarthy, and L. R. Cenkeramaddi, "Target classification by mmWave FMCW radars using machine learning on range-angle images," *IEEE Sensors J.*, vol. 21, no. 18, pp. 19993–20001, Sep. 2021.

[50] S. Li, M. M. O. I. Ozo, C. De Wagter, and G. C. H. E. de Croon, "Autonomous drone race: A computationally efficient vision-based navigation and control strategy," *Robot. Auto. Syst.*, vol. 133, Nov. 2020, Art. no. 103621.

[51] H. E. Mohamadi, N. Kara, and M. Lagha, "Heuristic-driven strategy for boosting aerial photography with multi-UAV-aided Internet-of-Things platforms," *Eng. Appl. Artif. Intell.*, vol. 112, Jun. 2022, Art. no. 104854.

[52] B. Endale, A. Tullu, H. Shi, and B.-S. Kang, "Robust approach to supervised deep neural network training for real-time object classification in cluttered indoor environment," *Appl. Sci.*, vol. 11, no. 15, p. 7148, Aug. 2021.

[53] J. Fan, X. Chen, Y. Wang, and X. Chen, "UAV trajectory planning in cluttered environments based on PF-RRT* algorithm with goal-biased strategy," *Eng. Appl. Artif. Intell.*, vol. 114, Sep. 2022, Art. no. 105182.

[54] C.-L. Fan, "Advancements in image recognition for urban land use: Multi-scale CNN extraction," in *Proc. 10th Int. Conf. Electr. Eng., Comput. Sci. Informat. (EECSI)*, Sep. 2023, pp. 261–267.

[55] M. Paszkuta, J. Rosner, D. Peszor, M. Szender, M. Wojciechowska, K. Wojciechowski, and J. P. Nowacki, "UAV on-board emergency safe landing spot detection system combining classical and deep learning-based segmentation methods," in *Proc. Asian Conf. Intell. Inf. Database Syst.* Cham, Switzerland: Springer, 2021, pp. 467–478.

[56] P. Mathur, Y. Jangir, and N. Goveas, "A generalized Kalman filter augmented deep-learning based approach for autonomous landing in MAVs," in *Proc. Int. Symp. Asian Control Assoc. Intell. Robot. Ind. Autom. (IRIA)*, Sep. 2021, pp. 1–6.

[57] A. Albanese, T. Taccioli, T. Apicella, D. Brunelli, and E. Ragusa, "Design and deployment of an efficient landing pad detector," in *Proc. Int. Conf. Syst.-Integr. Intell.* Cham, Switzerland: Springer, 2023, pp. 137–147.

[58] H. D. Tran, T. H. Tran, Q. D. Nguyen, D. A. Ngo, D. N. Duong, J. P. Pestana, and A. Q. Nguyen, "A multiple marker design for precision and redundant visual landing in drone delivery," in *Proc. 12th Int. Conf. Control, Autom. Inf. Sci. (ICCAIS)*, Nov. 2023, pp. 127–132.

[59] J. A. García-Pulido, G. Pajares, and S. Dormido, "UAV landing platform recognition using cognitive computation combining geometric analysis and computer vision techniques," *Cognit. Comput.*, vol. 15, no. 2, pp. 392–412, Mar. 2023.

[60] M. S. Ruiz, A. M. P. Vargas, and V. R. Cano, "Detection and tracking of a landing platform for aerial robotics applications," in *Proc. IEEE 2nd Colombian Conf. Robot. Autom. (CCRA)*, Nov. 2018, pp. 1–6.

[61] P. H. Nguyen, K. W. Kim, Y. W. Lee, and K. R. Park, "Remote marker-based tracking for UAV landing using visible-light camera sensor," *Sensors*, vol. 17, no. 9, p. 1987, Aug. 2017.

[62] L. Chen, C. Liu, S. Guo, and H. Qian, "Vision-guided UAV landing on a swaying ocean platform in simulation," in *Proc. IEEE Int. Conf. Real-time Comput. Robot. (RCAR)*, Jul. 2023, pp. 456–462.

[63] Y. He, Z. Zeng, Z. Li, and T. Deng, "A new vision-based method of autonomous landing for UAVs," in *Proc. 9th Int. Conf. Electr. Eng., Control Robot. (EECR)*, Feb. 2023, pp. 1–6.

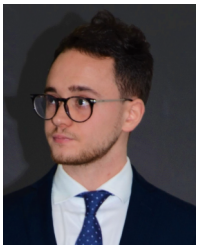
[64] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.

[65] I. Salehin, M. S. Islam, P. Saha, S. M. Noman, A. Tuni, M. M. Hasan, and M. A. Baten, "AutoML: A systematic review on automated machine learning with neural architecture search," *J. Inf. Intell.*, vol. 2, no. 1, pp. 52–81, Jan. 2024.

[66] W. Jiang, L. Yang, S. Dasgupta, J. Hu, and Y. Shi, "Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 4154–4165, Nov. 2020.

[67] A. Géron, *Hands-on Machine Learning With Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Sebastopol, CA, USA: O'Reilly Media, 2019.

EDOARDO RAGUSA (Member, IEEE) received the master's degree (cum laude) in electronic engineering and the Ph.D. degree in electronic engineering from the University of Genoa, Genoa, Italy, in 2015 and 2018, respectively. He is currently a Researcher with DITEN, University of Genoa, where he teaches digital systems electronics and machine learning. He has co-authored more than 50 refereed papers in international journals and conferences. His research interests include machine learning in resource-constrained devices, convolutional neural networks, and deep learning applications.



TOMMASO TACCIOLI was born in Genova, Italy, in 1997. He received the bachelor's and master's degrees in electronic engineering from the University of Genoa, Genova, in 2019 and 2022, respectively. In 2022, he joined the Nanobiointeractions and Nanodiagnostic Laboratories, Istituto Italiano di Tecnologia (IIT), Genova, as a fellow. Collaborating with the SEA Laboratory, University of Genoa, he worked on the development of computer vision algorithms and neural networks for smart sensing in POC devices.

ALESSIO CANEPA received the master's degree in electronic engineering from Turin University, Italy, in 2015, and the Ph.D. degree in electronic engineering from the University of Genoa, Genoa, Italy, in 2023. His research interests include machine learning and pattern recognition for embedded devices.

RODOLFO ZUNINO received the Laurea degree (cum laude) in electronic engineering from Genoa University, in 1985. From 1986 to 1995, he was a Research Consultant with the Department of Biophysical and Electronic Engineering (DIBE), Genoa University. He is currently a Full Professor with DITEN. He has co-authored more than 260 refereed papers in international journals and conferences. He has participated in the scientific committees of several international conferences on neural networks and their applications. His research interests include efficient models for data representation and learning, intelligent electronic systems for neural networks, intelligent systems for security, and advanced methods for multimedia data processing. He has chaired the master course (ii lev) in cyber security at the University of Genova.

PAOLO GASTALDO received the Laurea degree in electronic engineering and the Ph.D. degree in space sciences and engineering from the University of Genoa, Italy, in 1998 and 2004, respectively. He is currently an Associate Professor with DITEN, University of Genoa, where he teaches computer architectures and sensors. His research interests include embedded machine learning, computational intelligence, embedded systems for advanced signal processing in robotics and prosthetics, and cybersecurity.

Open Access funding provided by 'Università degli Studi di Genova' within the CRUI CARE Agreement