IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 7, 2022, accepted April 2, 2022, date of publication April 12, 2022, date of current version April 20, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3166939

Target-Aware Neural Architecture Search and Deployment for Keyword Spotting

PAOLA BUSIA[1], GIANFRANCO DERIU[1], LUCA RINELLI[2], CRISTINA CHESTA[2], LUIGI RAFFO[1], AND PAOLO MELONI[1]

[1]DIEE, University of Cagliari, 09123 Cagliari, Italy
[2]Concept BU, Santer Reply SpA, 10143 Torino, Italy

Corresponding author: Paolo Meloni (paolo.meloni@unica.it)

This work was supported by the European Union's Horizon 2020 Research and Innovation Program under Grant 780788.

ABSTRACT Keyword spotting (KWS) utilities have become increasingly popular on a wide range of mobile and home devices, representing a prolific application field for Convolutional Neural Networks (CNNs), which are commonly exploited to perform keyword classification. Addressing the challenges of targeting such resource-constrained platforms, requires a careful definition of the CNN architecture and the overall system implementation. These reasons have led to a growing need for design and optimization flows, able to intrinsically take into account the system's performance when ported on the target platform. In this work, we present a design methodology based on Neural Architecture Search, exploited to combine the exploration of the optimal network topology, the audio pre-processing scheme, and the data quantization policy. The proposed design flow includes target-awareness in the exploration loop, comparing the different design alternatives according to a model-based pre-evaluation of metrics like execution latency, memory footprint, and energy consumption, evaluated considering the application's execution on the target processing platform. We have tested our design flow to obtain target-specific CNNs for a resource-constrained commercial platform, the ST SensorTile. Considering two different application scenarios, enabling the comparison with the state-of-the-art of efficient CNN-based models for KWS, we have obtained up to a 1.8% accuracy improvement and a 40% footprint reduction in the most favorable case.

INDEX TERMS Keyword spotting, neural networks, neural architecture search.

I. INTRODUCTION

Convolutional Neural Networks (CNNs) are moving fast to the edge. The bandwidth and privacy issues of cloud-based execution have pushed for the deployment of lightweight, nevertheless very accurate, CNN implementations [1], [2] on mobile and power-efficient edge-processing platforms, thus improving the application's responsiveness and alleviating communication-related requirements. However, processing systems of such a kind are often resource-constrained and operate on limited energy budgets, requiring careful tailoring of the CNN and accurate choices during porting and deployment. The high number of design choices and optimization parameters at the disposal of the developers poses a need for design practices allowing for target-aware optimization on all of the design phases, through dedicated supporting design tools.

The associate editor coordinating the review of this manuscript and approving it for publication was Valentina E. Balas.

Along with computer vision, speech recognition is a common application field for CNNs, often configured as keyword spotting (KWS) in the edge domain. The KWS task focuses on the recognition of a limited set of words representing a simple speech recognition system [3], or working as wake words triggering the execution of more complex speech processing systems running on the cloud, as in smart home devices. When it comes to automated CNN design and optimization, KWS represents an extremely interesting case study, due to two main reasons:

1) the need for KWS is ubiquitous, thus it could require the classifying network to be deployed on a very wide range of processing platforms and hardware architectures, highlighting the importance of a precise target characterization within the design flow;
2) most of the approaches to CNN-based KWS envision fairly complex steps of pre-processing to be applied

to the input samples, opening an additional range of choices to the designer.

Thus, a CNN-based KWS system involves several design choices, often assigned to automated procedures, like Neural Architecture Search (NAS). In this work, we present a design flow, implemented with the ALOHA[1] tool flow, which is capable of efficiently synthesizing, training, quantizing, and deploying platform-specific KWS applications. The flow is designed to:

- *consider target-awareness*: we use a model-based evaluation tool to pre-estimate the effects of the different design choices on metrics measuring inference performance on the target processing platform, such as latency, footprint, or energy consumption. The obtained estimations are considered during the whole optimization process;
- *consider quantization* as one of the parameters to be explored and tuned during the optimization;
- enable the combined *cross-exploration of the data pre-processing and the CNN topology*, as these design choices influence each other, while both impact accuracy and demands for hardware resources.

We define two different methodologies for the usage of the tool flow. The first one implements a *fast selection* procedure, representing an efficient trade-off between the required design time and the quality of the results. The second one involves an intermediate characterization phase where the effects of quantization on the network model's accuracy are precisely evaluated. In the following, we refer to it as *accurate selection*. Considering as a target platform the microcontroller-based ST SensorTile, we have tested our CNN selection methodologies on two different sets of hardware-related constraints, enabling the comparison with the best works in the literature presenting CNN architectures for KWS [4], [5] obtained through NAS. We have obtained two application configurations outperforming CNN state of the art in the KWS field. In the first case, our approach improves the accuracy of the CNN model considered in [4], by 1.8% points, with 40% lower storage requirements. In the other case, we reach similar accuracy (0.14% improvement), with 14% fewer operations required, corresponding to a 5% latency reduction [5]. Our approach could be extended with some additional support to depthwise separable convolutions, which result in sensible performance improvement in [4]–[6]. To provide a brief outline of our work, Section II contains a brief introduction to NAS and its role in the design of efficient CNN architectures for KWS, while Section III describes the platform exploited as a target for our design. Section IV defines a KWS system and anticipates the choices involved in its design, while Section V presents the framework exploited for CNN exploration. Section VI describes in detail our selection procedure, in both the fast and accurate implementations. Finally, experimental results are presented in Section VII: the considered search space is defined in Section VII-A, while Section VII-B provides the motivation for the pre-processing scheme/ CNN topology combined exploration.

[1]The ALOHA project is available at https://www.aloha-h2020.eu/ and aims at developing a framework providing several tools for architecture-aware CNN exploration.

PAPER CONTRIBUTION SUMMARY

The main novel contribution of this paper can be summarized as follows:

- we introduce a tool flow composed of several search and evaluation utilities usable to compose automated target-aware optimization of CNNs;
- we propose two different selection procedures (*fast* and *accurate*) exploiting such tool flow for the KWS problem;
- we test such procedures on two use-cases corresponding to two different sets of target-related constraints, reaching results comparable to the state-of-the-art or improving it, in terms of accuracy, latency, or footprint.

II. RELATED WORK

The very recent work of [8] presents an overview of the KWS methods and techniques explored in the literature. The first systems were developed as large-vocabulary continuous speech recognition systems, mapping the audio into a sequence of probable phonetic units, but requiring high processing and storage capabilities to be executed, and thus not suitable for edge execution [9]. A more lightweight solution was represented by Hidden Markov Models (HMMs) and Gaussian Mixture Models (GMMs) [10]. Nowadays, Deep Neural Networks (DNNs) have replaced the alternatives, due to their flexibility in terms of complexity, functionality, and accuracy, becoming the most popular classification method for KWS tasks. Very recent works have also exploited DNNs for speaker identification [11], as well as CNNs applied to Mel-Frequency Cepstral Coefficients (MFCC) features for the detection of the speaker language [12]. However, some energy-efficient Machine Learning alternatives have been proposed. Among those is the work of [13], presenting a KWS system based on a Tsetlin Machine, a Finite State Machine exploiting propositional logic to perform classification, while the work of [14] investigates an approach based on Support Vector Machines (SVMs). The currently best performing models are compared in [15], typically referring to 12 classes classification problems on the common benchmark represented by the Google Speech Commands dataset [3]. All in all, they can be grouped into CNNs [16], Recurrent Neural Networks (RNNs) [17], and keyword Transformers [18].

Regardless of the model's affiliation with one or the other of the neural networks domains, efficient network design is a crucial issue when inference has to be executed on resource-constrained platforms. It is often the case for KWS tasks, which are always-on and commonly executed on power-constrained edge devices, whose available storage space is also strictly limited. Thus, recent works have proposed several workload-efficient network architectures,

TABLE 1. Comparison with state of the art works on NAS targeting KWS.

	Hardware metric	Quantization		Pre-processing		Exploration description
		Levels	Cross-exploration	Parameters	Cross-exploration	
[4]	*OPS footprint*	*8bit*	✗	*n. frames*	✓	✗
[5]	*OPS latency*	*up to 1bit*	✗	*not applicable*	*not applicable*	✓
[7]	*OPS*	✗	✗	✗	✗	✓
[6]	✗	✗	✗	✗	✗	✓
this work	*latency footprint energy*	*up to 4bit*	✓	*feature type n.features n. frames*	✓	✓

either handcrafted [19] or automatically obtained as the result of NAS processes [4]–[7]. In the following, we will focus on the latter, which are more closely related to our work.

While the first approaches to NAS have mostly focused on accuracy-oriented automated design, especially targeting image classification tasks, [20]–[22], NAS has more recently evolved into a target-aware design process, where non-functional performance metrics [23] are taken into account during the optimization procedure. Furthermore, to cope with the size of the architecture design space, several approaches have focused on reducing the search time in NAS, e.g. exploiting one-shot training [24], or differentiable search [25]. Finally, to keep up with the wide adoption of reduced-precision CNNs, some automatic design flows also consider compression through quantization [26], as an optimization objective along with network topology selection.

Given the tight deployment constraints of KWS systems, NAS for optimal design has been widely adopted in this field. In Table 1, we list some of the most recent works presenting neural networks for KWS designed through NAS, establishing the state of the art. In column 1, we compare them in terms of the hardware performance metric which is considered during the exploration process. In columns 2 and 3, we describe how the quantization subject is addressed. In detail, we list the quantization policies explored and we report whether the quantization policy is cross-explored with the CNN topology. Similar information is reported in columns 4 and 5 for the pre-processing scheme. In column 4, referring to the most common speech features, Mel energies and MFCC, we report which of the design parameters are explored and whether the evaluation happens contextually, or on top of, the network topology selection. Finally, in column 6, we define whether the corresponding work provides methodological guidelines for the exploration. The listed works are briefly summarized in the following.

The work of [4] specifically targets microcontrollers embedding ARM Cortex-M Processors. Considering the design constraints posed by different sized systems, the authors present an exploration of different network architectures and operands, targeting three search spaces defined by Small, Medium, and Large size systems. The search for the optimal network architecture is guided by number-of-operations-(OPS) and footprint-based performance evaluations. A partial pre-processing scheme cross-exploration involves the number of frames composing the input spectrogram. However, all of the presented CNN models perform classification on spectrograms obtained through MFCC, considering 10 features and 49 time-frames, thus having 49×10 resolution. On the other hand, the quantization policy is not cross-explored, as compression to 8-bit representation is only applied to the selected architectures. The focus of the work is on the comparison among different network models, based on CNNs, DS-CNNs, RNNs, and DNNs, thus it does not provide a defined exploration method, either for the network topology or for the pre-processing scheme. The work presents the well-known state-of-the-art DS-CNN architectures, exploiting depth-wise separable convolutions and outperforming the corresponding CNN-based models. In the following, we compare with the CNN architecture selected for the 200kB - 20 MOPS Medium region, reaching 92.2% accuracy, obtained through 199.4 kB parameters and 17.3 MOPS.

In [5], the authors present a network architecture selected through differentiable NAS, where deployment performance is evaluated in terms of OPS. The inference is executed on raw audio files, through parameterized *sinc* functions learned as a first layer, known as SincConvs. Thus, the approach to audio processing can not be described in terms of Mel/MFCC parameters, and the corresponding columns 4 and 5 in Table 1 are not filled in. Quantization, up to 1bit precision, is only applied to the selected architectures, while it is not cross-explored with the network topology. The selected CNN-based model reaches 95.6% accuracy, having 75.7 kB parameters and 13.6 MOPS, while the quantized version uses up to 2.51 bits per weight and 2.91 bits per activation, reaching 93.76% accuracy.

In [7], the authors exploit NAS where deployment performance is evaluated in terms of OPS count. After the first fixed-budget training phase, a Pareto frontier of candidate points is selected for a second refinement process, where the training hyperparameters are evaluated through exploration. The work provides methodological guidelines for the exploration, which do not include cross-exploring the pre-processing scheme and quantization policy together with the CNN topology. The selected network reaches 95.1%

accuracy, exploiting pre-processing based on MFCC, resulting in 40 × 32 input resolution.

Finally, the work of [6] presents the architecture which is, to the best of our knowledge, the state-of-the-art one for 12 classes KWS, reaching 97.2% accuracy. The selected network topology exploits both depthwise separable and dilated convolutions, and is trained on 101 × 40 input pixels, obtained through MFCC based audio pre-processing. The CNN design is obtained through NAS, but the pre-processing scheme is pre-defined, and not subject to exploration. Likewise, the quantization policy is not explored.

To the best of our knowledge, as reported in Table 1, ours is the first approach to put in place a co-exploration of the optimal CNN topology, pre-processing scheme, and quantization level, based on hardware-aware performance evaluations, and to provide its step by step description. Our perspective is motivated by the great impact of the feature extraction and input resolution choice, on both the classification accuracy and the overall system performance, in terms of execution time and required storage resources. Furthermore, according to the current trend [27], we mean to extend to the KWS field the evaluation of the quantization policy co-exploration within NAS. Finally, we allow for hardware-aware performance evaluation. This is achieved through a latency prediction model of the target platform, allowing us to refine the OPS metric into a more accurate execution time estimation and consider immediate performance constraints expressed as a maximum allowed execution time.

The application of our proposed design procedures allows us to obtain CNN architectures reaching accuracy values competitive with the CNN state-of-the-art in the KWS field while being specifically tailored for the target platform, thanks to hardware-aware inference latency predictions. Our approach enables considering the specific hardware requirements in the design of all the most relevant parameters impacting the performance of the system, including the pre-processing scheme and quantization policy choice. As is further discussed in Section VII-B, different pairs of topology and preprocessing scheme can represent the optimal combination in different latency regions, thus it is beneficial to combine their exploration. The adoption of one-shot training and multiple parameters cross-exploration improves the efficiency of a NAS process in such an enhanced design space. When multiple pre-processing schemes are evaluated, the fast selection procedure enables an exploration time reduction, after the one-shot training, proportional to the number of schemes explored. The details of the exploration time analysis are given in Section VI-C.

III. TARGET PLATFORM

As an example target of our design-flow application, we consider a tiny microcontroller platform, developed by STMicroelectronics: the SensorTile. It is an IoT module, equipped with a digital microphone, and embedding an 80 MHz ARM Cortex-M4 32-bit low-power microcontroller, accessing a 96kB SRAM, and 1MB FLASH memories, posing strict storage constraints. The system architecture exploits a Real-Time lightweight Operating System (RTOS), providing support for multi-threading, and scheduling of the different application tasks on defined timings. For efficient CNN execution, we relied on the CMSIS-NN library [28], specifically developed to target this family of processors.

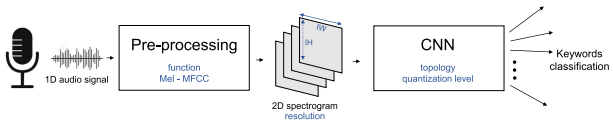


FIGURE 1. KWS system overview.

IV. OPTIMIZING A CNN FOR KWS: PROBLEM DEFINITION

Our approach aims at defining a set of tools and methodologies, for automating the optimization of KWS systems. A generic view of a KWS system detailing the composition of the software application is highlighted in Figure 1. Audio samples are received in input and are streamed to a pre-processing stage. In this paper, we assume as reference the pre-processing functionality which is more commonly used in literature, i.e. converting the monodimensional input stream into a bidimensional representation. We particularly consider Mel energies and MFCC, which are both representations of the power spectrum of the acquired audio over time. Such extracted 2D features are then sent in input to a CNN algorithm, which classifies the incoming data over a set of classes corresponding to the keywords to be spotted. We aim to set an optimal configuration of the application knobs available in such a system, maximizing the classification accuracy under the constraints defined considering the target processing platform and the required performance. More formally, the optimization process starts from the following list of inputs:

- dataset - set of pre-processed audio data, obtained through a certain combination of the pre-processing steps;
- platform description - defining the available storage and processing resources, and allowing to obtain hardware-related performance metrics for the examined CNN algorithm;
- constraints - limits to the maximum CNN memory footprint, and the maximum inference latency, depending on the platform's description and the application requirements on the final throughput;
- design space definition - set of CNN structures and operands to be explored.

As most of the state-of-the-art works, the classification task refers to the Google Speech Commands dataset [3], and involves 10 of the 35 classes provided: "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop" and "Go", plus the additional classes "silence" and "unknown".

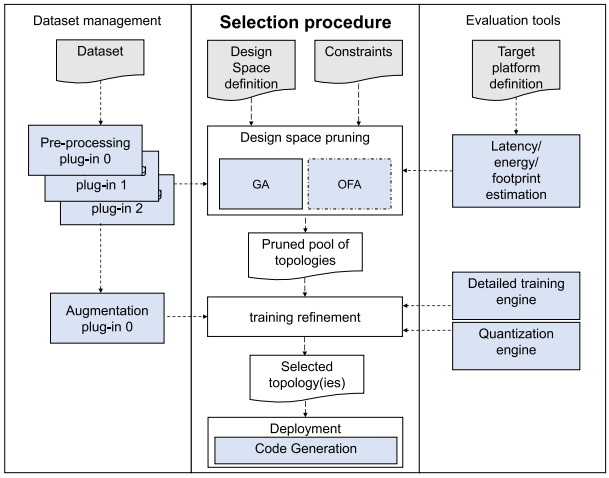


FIGURE 2. Selection procedure for target-oriented CNN design exploiting the ALOHA framework.

Considering this input, we need to implement a Design Space Exploration that analyzes and compares different *design points*, corresponding to different configurations of the following design parameters:

- CNN topology;
- feature-extracting function to be used during pre-processing;
- resolution of the features produced by the pre-processing;
- CNN quantization level.

As noticeable, all the possible design points, corresponding to the available permutations of such parameters, define a very vast design space that, in general, cannot be fully explored.

Thus more complex procedures, based on NAS strategies, are needed to obtain (near-) optimal results in a reasonable time.

V. ALOHA CNN DESIGN FLOW

According to the NAS approach, the ALOHA framework allows addressing CNN design as an iterative selection process. An overview of the framework organization is shown in Figure 2. The core of the design flow is a *Selection Procedure*, which aims to define and deploy the optimized CNN, based on a set of design constraints and the definition of the reference design space. The procedure is composed of several sub-steps that iteratively refine the CNN topology selection until the identification of the optimal candidate for the target platform. A *Dataset management* utility takes in input the reference dataset defining the task to be implemented by the CNN and applies pre-processing and data-level transformations to feed the different training actions within the procedure. Moreover, the *Selection Procedure* is served by a set of *Evaluation tools*, which can be used to assess the metrics needed to compare the design points with each other.

In the following, we outline in more detail the features of the tool flow components.

A. DATASET MANAGEMENT UTILITY

As depicted in the left column, *Dataset management*, the ALOHA tool flow provides the possibility to customize the data pre-processing operations according to the application's requirements [29]. In detail, each transformation to be applied to the data is described as a plug-in, i.e. a pre-processing operator that can be treated independently and arbitrarily connected to others, to compose a *preprocessing pipeline*. The available operands can be set to be applied on the overall input set, at sample level, or batch level. In this paper, for KWS classification, they include MFCC and Mel feature-extracting functions, representing the *pre-processing plug-ins*, but also several *augmentation plug-ins*, like random time shifts and random noise addition, as well as random pitch and random speed. Whether these transformations are applied, and at which point of the selection procedure, can be defined by the user.

B. SELECTION PROCEDURE

The *selection procedure* starts with the definition of a reference *Design Space*, establishing the set of operands and topologies to be considered for exploration. Furthermore, the selection needs to account for possible design *Constraints*, posed by the performance requirements of the application, or by the resources available on the target hardware. Given this set of inputs, the first stage of the procedure involves a *Design space pruning* step. A search strategy surfs through the initially defined design space, to identify a reduced pool of eligible near-optimal CNN topologies. To this aim, it uses a *Genetic Algorithm* (GA) that iteratively refines evolving populations of CNNs, according to their comparison. During this phase, considering the big number of networks to be compared, the accuracy of the design points is assessed using a *Once-for-All* (OFA) one-shot training utility [24]. Moreover, the metrics related to the execution on the target hardware are estimated using a target-aware *Latency/ energy/ footprint evaluation* tool.

At this point, the resulting pruned pool of topologies can be further examined through a refinement phase, precisely assessing the accuracy and footprint of the deployable network through *detailed training* and *quantization*. Finally, one or more CNN architectures can be selected for *Deployment*.

1) THE OFA TRAINING

The one-shot training facilitates energy-efficient NAS evaluation cycles. This external utility [24] allows describing the search space as a single SuperNetwork, obtaining, through a single training process, all the configurations to be evaluated as possible subnetworks. After the teacher network has been trained for a configurable number of training epochs, all of the desired configurations can be optimized through a Progressive Shrinking procedure. To prevent the training of one subnetwork from interfering with the others, the optimization starts from the network models having the highest number of parameters, and finally adjusts the accuracy of the smallest

ones, which profit by weight sharing. For example, starting from a CNN exploiting only 5×5 filtering kernels, the kernel size is made elastic by additionally training their 3×3 subsets. The same can be done with the network's depth and width. As further discussed in Section VII-A, this allows us to train a huge number of candidate networks, in a reasonable amount of time.

As soon as the SuperNetwork is trained, the accuracy of each design point can be assessed by just repeating a validation pass over the validation set, which is significantly shorter than training each design point from scratch.

2) THE GENETIC ALGORITHM

The GA allows searching across the search space, by iteratively selecting and updating the population of design points satisfying the search criteria. The candidate networks are admitted into the eligible population according to performance evaluation based on accuracy, latency, memory, and energy estimations. After one generation of design points has been evaluated, the new one is obtained as the composition of the most accurate networks evaluated up to that point, and of new network models obtained by randomly changing the parameters of those most promising points. The possible mutations considered in this work, based on the flexibility of the procedure, and the structure of the SuperNetwork defining the search space, involve the specific pre-processing pipeline, the input and kernel size, the number of convolutional layers, and their width. Thus, the GA evolves towards the selection of the most suitable candidate points, by optimizing the validation accuracy within the defined constraints.

C. EVALUATION TOOLS

The *Evaluation tools*, exploited during the pruning and refinement phases, are listed in the right column.

1) THE LATENCY ESTIMATION TOOL

The *Latency estimation tool* provides hardware performance metrics based on the evaluated network's parameters and the target platform specifications [30]. The inference time on the SensorTile is evaluated through a simple Roofline-based [31] model. The model, shown in Figure 3, considers two distinct performance roofs, assessed through benchmarking, for convolutional and fully connected operands: in the first case, the maximum achievable performance is 0.64 ops/cycle, while in the latter it is limited to 0.3 ops/cycle. The network's latency is obtained as the sum of each layer's execution time, evaluated as the ratio between the number of operations required by the layer, and the corresponding achievable performance. We assessed the average estimation error of around 25%, by comparing the resulting estimations with on-hardware latency measurements, on a set of 450 common convolutional layers, and 60 fully connected layers.

The memory footprint is evaluated considering the selected precision for the network's parameters, and assuming a double buffer mechanism for the activations. Finally, the tool can be easily extended to provide energy consumption estimations, although we do not consider energy among the hardware metrics in this work.

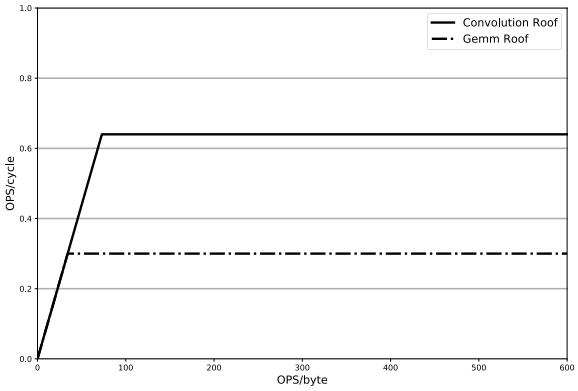


FIGURE 3. Roofline model of the SensorTile platform, representing peak performance roofs for convolution and gemm execution.

2) THE DETAILED TRAINING ENGINE

The *Detailed training engine* works on ONNX (Open Neural Network Exchange) model descriptions [32] and provides the possibility to explore the training hyper-parameters and handle multiple input formats for different use-case configurations, supported by the Dataset management plug-ins. It also allows applying transfer learning techniques to pre-trained models, further improving their accuracy [33]. Thanks to the possibility to exploit data augmentation techniques, the Detailed engine allows reaching higher accuracy values than the one-shot training, although it requires a longer time.

3) THE QUANTIZATION ENGINE

The *Quantization engine* uses the NEMO (NEural Minimization for pytOrch) framework [34]. It is based on PACT (Parameterized Clipping Activations) quantization [35], which replaces ReLU activations with a clipping function between 0 and a maximum value, defined by the desired number of representation bits. The network model resulting from such procedure presents quantized convolutional weights, while Batch Normalization (BN) and ReLU operators along the architecture are replaced respectively with a series of Mul/Add and Mul/Div/Clip operators, allowing for intermediate quantized representations.

D. DEPLOYMENT

Finally, efficient and fast deployment is automated thanks to a Python script for target-oriented *Code generation*. The Python script generates a software application in C language, that uses generic operators for NN and pre-processing functions, which are linked to the specific target-compliant implementation. Based on the network model in ONNX format, the tool provides the source code for inference execution and appropriate parameters arrangement for the target library. To briefly describe generation targeting SensorTile, inference execution exploits CMSIS-NN library, modified to allow BN and ReLU (or their quantized replacements) folding with the

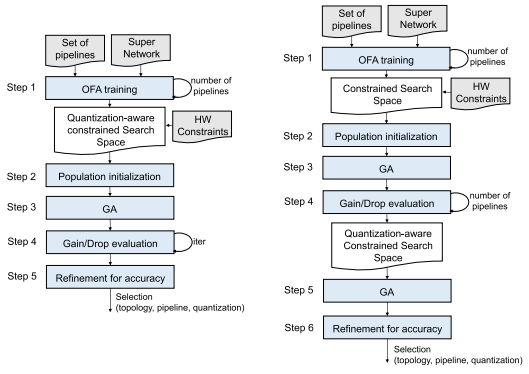


FIGURE 4. Design steps to be performed according to the a) fast and b) accurate implementations of the CNN selection procedure.

convolution operations, and to efficiently handle the quantized models produced by the quantization tool.

VI. CNN SELECTION PROCEDURE IMPLEMENTATION

In the following, we present an efficient CNN selection procedure, representing our proposed design solution exploiting the tools presented in Section V. We developed it to provide an efficient co-exploration strategy, where both the network topology and the pre-processing scheme are treated as random variables by the evolution search. The motivation for such an approach is further explored in Section VII-B. We first describe a fast and simple implementation, resulting in an architecture selection that ensures high accuracy and efficient performance, while requiring a limited design time. Finally, we present a more complex and accurate version, aiming at improving the selection quality with more detailed performance evaluations.

A. FAST IMPLEMENTATION

Figure 4a shows an overview of the simple version of the network selection procedure, whose detailed implementation is described in Algorithm 1. It can consider a set PP of pre-processing pipelines, exploiting different feature extracting functions and resulting spectrum resolutions. Furthermore, it accepts a set of hardware constraints, setting a maximum limit for the network's memory footprint and execution time, based on the target platform specification.

In Step 1, the SuperNetwork architecture, SN, is one-shot trained exploiting the OFA utility [24]. The search space to be explored is obtained through the Progressive Shrinking procedure as the set N of all its possible sub-networks. If a set of hardware constraints is provided, the search space is restricted into one, denoted as $N*_c$, enforcing the memory and latency restrictions. All the design points whose storage requirements or inference latency exceed such constraints are discarded, but first different levels of quantization are evaluated, reducing the precision of either weights or activations up to 4bit representation.

In Step 3, the GA is executed. We consider populations made up of 100 network models, evolving to explore $G_1 = 20$ generations of design points, compliant with the hardware constraints. Each generation preserves the 25 most accurate network architectures of the previous one, while the new candidate design points are obtained through random mutation (50) and parameter crossover (25) of such best design points. The possible mutations involve the network's depth, resolution, and pre-processing features, as well as layer-wise kernel size, and channel width. The performance evaluation in this step is based on the one-shot training accuracy.

The selection of the optimal model is handled in Step 4. The simplest design choice coincides with the selection of the most accurate design point in the last generation explored. Otherwise, a more in-depth analysis can be performed, repeating a sequence of evaluation actions for a desired number of iterations, denoted as *iter* in Algorithm 1:

- the last generation A_G is ordered based on the predicted classification accuracy (during the first iteration, such value matches the one-shot accuracy);
- the most accurate CNN architecture is selected for 100 epochs Detailed training, where data augmentation is applied to reduce the overfitting effect;
- the CNN architecture is quantized according to the selected quantization policy, and retrained for 100 epochs to reduce the accuracy drop;
- retraining gain and quantization drop are evaluated and exploited to improve the predicted accuracy of the architectures in A_G.

The value of *iter* can be defined by the user, based on the effort and compute time that he is willing to dedicate to the selection flow. The case $iter = 1$ falls into the simple selection of the most accurate design point in the last generation, while higher values of *iter* may require the definition of an update-policy for the accuracy gain resulting from Detailed training and the quantization drop. Different schemes can be exploited. To provide an example, gain and drop can be updated by: 1) considering the values evaluated for the architecture with the closest memory footprint; 2) considering an average of the available values; 3) considering the last evaluated value. After the last iteration, Step 5 performs a final refinement on the selected architecture. This refinement step can be preceded by a hyperparameters exploration, to evaluate the configuration of learning rate and batch size resulting in the highest accuracy.

B. ACCURATE IMPLEMENTATION

One possible flaw of the previously described procedure derives from neglecting the effects of quantization on the models' accuracy during the evolutionary search. Such effects are only considered during the final selection process.

As an alternative, we developed a more accurate version, depicted in Figure 4b and described in Algorithm 2.

Algorithm 1 Fast CNN Architecture Selection

Input: $PP(H, W, AudioProc)$, SN, $hw(Mem, Texe)$
Result: CNN architecture a
Step 1. *OFA training*;
for $i \in [1, p]$ **do**
  OFA_train(SN, PP_i);
$N = \{N(PP_1), \ldots, N(PP_p)\}$;
Step 2. *Population initialization*;
$N*_c = \{n_i |$ (Mem*(n_i),Texe(n_i)) $< hw(Mem, Texe)\}$;
$A_1 = \{n_1, \ldots, n_{100}\}$ with $n_i \in N*_c$;
Step 3. *GA in HW-aware search space* $N*_c$;
for $i \in [1, G]$ **do**
  Evolution_Search(A_i, $N*_c$);
A_G;
Step 4. *Quantization drop evaluation*;
for $i \in [1, iter]$ **do**
  Order(A_G, Accuracy);
  Detailed_train(best(A_G));
  $g(PP_i)$ = Evaluate_Gain;
  Quantization((n, quant));
  $d(PP_i)$ = Evaluate_Drop;
  Adjust_accuracy(A_G, $g(PP_i)$, $d(PP_i)$);
$a = best(A_G)$
Step 5. *Refinement for Accuracy*;
return a

In this case, the GA in Step 3 explores a search space N_c which only admits the networks that fit in the memory constraint when using 8bit representation, which has little or no effect on the model's accuracy and is the most common deployment precision of the target library, CMSIS. This first run of the evolutionary search is a preliminary step, exploited to choose adequate design points to explore the achievable gain, resulting from detailed training introducing data augmentation, and the possible drop connected to quantization. To this aim, we select from A_{G_1} the CNN architecture having the biggest footprint, among the network models within one percent accuracy point from the most accurate one. This choice follows the general assumption that a network with a higher number of parameters can benefit more from the training procedure. The analysis conducted in this step is exploited as a prediction model for the networks requiring quantization to fit the memory constraint and be included in the search.

Therefore, in Step 4, the retraining gain and quantization drop are evaluated separately for each of the pre-processing pipelines. To reduce the training time on the Detailed engine, which is critical to this implementation of the selection procedure, we exploit a *static augmentation* of the training dataset. Multiple copies of the dataset, enforcing different random levels of data augmentation, are created, and alternatively processed at each training epoch. Such a static augmented dataset is saved and made available for the successive training procedures, reducing the impact of the pre-processing operations on the training time. We found that such a solution does not impact the final accuracy.

Algorithm 2 Accurate CNN Architecture Selection

Input: $PP(H, W, AudioProc)$, SN, $hw(Mem, Texe)$
Result: CNN architecture a
Step 1. *OFA train*;
for $i \in [1, p]$ **do**
  OFA_train(SN, PP_i);
$N = \{N(PP_1), \ldots, N(PP_p)\}$;
Step 2. *Population initialization*;
$N_c = \{n_i |$ (Mem(n_i),Texe(n_i)) $< hw(Mem, Texe)\}$;
$A_1 = \{n_1, \ldots, n_{100}\}$ with $n_i \in N_c$;
Step 3. *GA in HW-aware search space* N_c;
for $i \in [1, G_1]$ **do**
  Evolution_Search(A_i, N_c);
A_{G_1};
Step 4. *Quantization drop evaluation*;
$D = \{n_{PP_1}, \ldots, n_{PP_p})\}$ where n_{PP_i} has biggest footprint in A_{G_1};
for $n \in D$ **do**
  Detailed_train(n);
  $g(PP_i)$ = Evaluate_Gain;
  Quantization(n);
  $d(PP_i)$ = Evaluate_Drop =$\{d_{x8w8}, d_{x4w8}, d_{x8w4}, d_{x4w4}\}$;
$N*_c = \{n_i |$ (Mem*(n_i),Texe(n_i)) $< hw(Mem, Texe)\}$;
$A'_1 = A_{G_1}$;
Step 5. *GA in HW aware Search Space* $N*_c$;
for $i \in [1, G_2]$ **do**
  Adjust_accuracy(A'_i, $g(PP)$, $d(PP)$);
  Evolution_Search(A_i, $N*_c$);
$a = best(A'_{G_2})$;
Step 6. *Refinement for Accuracy*;
return a

In Step 5, a second run of the GA is performed, starting from the last generation A_{G_1}, produced in Step 3, and including in the new search space, $N*_c$, the possibility to perform quantization up to 4 bits. At this point, the ranking of the architectures takes into account the effects of quantization on their accuracy, as evaluated in Step 4.

After $G_2 = 20$ generations, the most accurate model, associated to its pre-processing and quantization scheme, is chosen as the optimal selection, and is retrained and refined, in Step 6, to further improve its accuracy.

C. SELECTION TIME

We analyze here the required exploration time for the described selection procedures. Table 2 lists the operations performed according to the fast and accurate implementations and provides a general estimate of their execution time, based on measurements performed on NVIDIA Tesla T4, exploited for the one-shot training, and on NVIDIA Tesla P100. For each table entry, we report the corresponding time values, which depend on the search parameters, more specifically on the number of different preprocessing pipelines ($|PP|$) and quantization levels (Q) considered, and on the number of refinement steps performed until selection ($iter$). As can be derived from the Table, the time required for the quantization

TABLE 2. Step by step required exploration time for the accurate and fast selection procedure, where the OFA training is executed on NVIDIA Tesla T4, while the GA exploration and the detailed training are executed on NVIDIA Tesla P100.

Operation	Step		Execution Time	
	Accurate Selection	Fast Selection	Accurate Selection	Fast Selection
OFA train	1	1	2h 30× $\|PP\|$	2h 30× $\|PP\|$
GA	3	3	6h	6h
Gain eval	4	4	1h 30× $\|PP\|$	1h 30 × $iter$
Drop eval	4	4	2h× $Q \times \|PP\|$	2h × $iter$
GA	5	-	6h	-
Refinement	6	5	5h	5h

drop evaluation in the fast implementation does not scale with the number of pipelines and quantization levels explored, since it is only performed a selected number of times, *iter*. For the use-cases presented in the following, where $|PP| = 6$ and $Q = 4$, the Gain/Drop evaluation requires 51 hours in the accurate implementation, against 3h 30 needed in the fast one (with an *iter* choice of 1). Furthermore, we also mean to emphasize the substantial savings delivered by the CNN topology/ pre-processing co-exploration. A separate evaluation would in effect require repeating the topology GA search on multiple design spaces, as many times as is the number of pipelines considered, or assuming in advance a given scheme, neglecting such an important design variable. This would require 36 hours of GA exploration, against the 6 hours needed by the fast implementation. Thus, the fast implementation allows a factor *PP* reduction of the required exploration time after the one-shot training.

VII. EXPERIMENTAL RESULTS

The selection procedures described in the previous section were exploited to design optimal CNN architectures in two different scenarios, resulting from the hardware constraints selection. We referred to the state-of-the-art for the constraints definition [4], [5], in order to enable a direct comparison with the literature dealing with NAS targeting KWS applications. As anticipated in Section III, the search targets the ST SensorTile, and enforces a latency and memory constraint, exploiting the latency prediction model described in Section V.

A. SEARCH SPACE DEFINITION

The search space explores CNN architectures composed as indicated in Table 3. Each network presents either 1 or 2 convolutional stages, separated by a MaxPooling layer. The possible feature size within each stage is defined in column 5, while column 6 reports the stage's maximum depth. The number of convolutional layers in each stage ranges from 1 to *Max Depth*. Possible channel width values are listed in column 3, while considered kernel sizes are listed in column 4. Channel width and kernel size values are set independently for each convolutional layer. All the network configurations present a final fully connected stage.

Six pre-processing pipelines, described in Section VII-B, are considered, thus the training process at Step 1 of Algortihms 1 and 2 results in a set of over 330000 CNNs

TABLE 3. Parameters of the CNN architectures which constitute the Design Space for NAS targeting SensorTile.

Stage	Operator	Output Features	Kernel Size	Input Size	Max Depth
0	Conv	16/32/64	3x3/5x5	40x32/32x16 16x8	1
1	Conv	16/32/64	3x3/5x5	20x16/16x8 8x4	5

available for exploration, corresponding to all the combination of parameters in Table 3 for the six pre-processing schemes.

B. PRELIMINARY PRE-PROCESSING EXPLORATION

Given the different choices of CNN input dimensions exploited in the literature, we started our exploration process by comparing different feature extracting functions and resulting spectrogram's resolutions, to preliminary estimate the usefulness of the pre-processing choice's adaptation to different targets. Figure 5 shows the output of an evolution search conducted on the search space defined in Section VII-A. The search strategy does not enforce any hardware constraint, but it is hardware-aware and evolves by optimizing the design points to be Pareto optimal in terms of classification accuracy and required execution time on the target platform, based on the *latency evaluation tool* described in Section V. We repeated the search process on six distinct search spaces, each corresponding to a pre-processing scheme choice, exploiting either Mel-spectrogram or MFCC as feature extracting functions and resulting in 16×8, 32×16, or 40×32 input resolution. The dots in the figure represent the Pareto optimal design points, obtained after 20 generations. Each curve is defined by the input resolution and selected feature extracting function. As can be derived from the plot, the overall Pareto front would be made up of design points exploiting different pre-processing schemes, showing their impact both on the network's performance and on its classification accuracy. For example, optimal points in the left region are trained on 16×8 Mel-spectrograms, while in the rightmost region of the plot the higher accuracy values are reached thanks to 40×32 spectrograms.

Furthermore, Table 4 reports the execution time of online pre-processing, measured on the target platform. The measured values do not include the evaluation time of the constant parameters (e.g. the coefficients of the Mel filtering banks, and the DCT matrix), which can be computed once at first execution, and memorized for the successive iterations of the audio processing. Such results show how the overall system's performance is impacted by this design choice, affecting the overall latency of the KWS task.

Thus, we consider the pre-processing selection to require careful evaluation as a part of the system design process, and consider in the following its co-exploration with the topology and quantization scheme.

C. USE-CASE 1

As summarized in Table 5, considering as a reference the Medium size region defined in [4], we performed a CNN

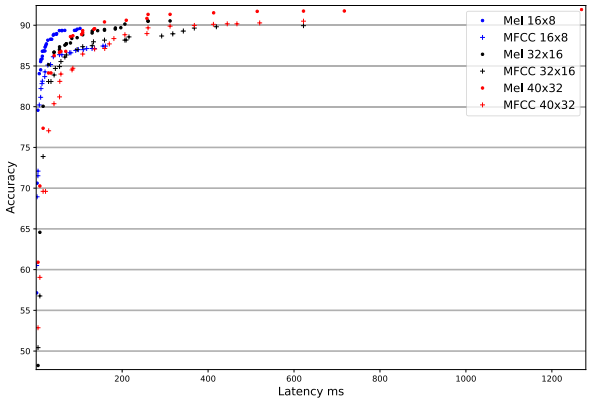


FIGURE 5. Comparison of the Pareto optimal design points resulting from NAS based on distinct evolution searches considering pre-processing schemes based on Mel and MFCC, with input resolution of 16 × 8, 32 × 16 and 40 × 32.

TABLE 4. Measured execution time for the considered preprocessing schemes on ST SensorTile.

	Pre-processing time
Mel 16x8	46 ms
MFCC 16x8	48 ms
Mel 32x16	94 ms
MFCC 32x16	98 ms
Mel 40x32	120 ms
MFCC 40x32	132 ms

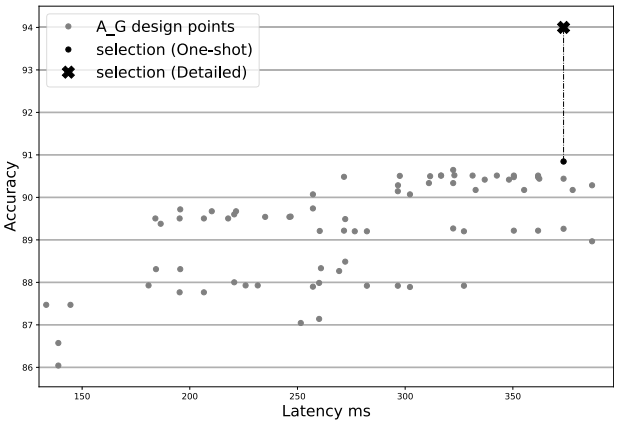


FIGURE 6. Fast selection output in 200kB - 20 MOPS search space. The selected model is highlighted, and its final accuracy upon detailed training is reported.

TABLE 5. Summary of search parameters for NAS targeting use-case 1.

Design Space	*Constraints*	*Constraints Reference*
as defined in Table 3	MOPS: 20 Memory 200kB	[4], referred to as Medium region
	Latency: 390ms	obtained based on model in Figure 3

topology/ pre-processing scheme exploration, constrained by a maximum memory footprint of 200 kB. The number of operations is limited to 20 MOPS, corresponding to a maximum latency of 390 ms. Figure 6 reports the output of the fast selection process. The most accurate design points within the pruned pool of CNNs selected through the GA run are depicted as bullets and placed based on their one-shot

TABLE 6. Performance metrics summary for the selected design point and the reference state of the art network, in the 200kB-20MOPS region, expressed as percentage of the constraints value.

Network model	*Accuracy*	*Latency*	*MOPS*	*Memory*
CNN M [4]	92.2%	86.9%	86.5%	99.7%
fast selection	94%	95.8%	86.8%	60%

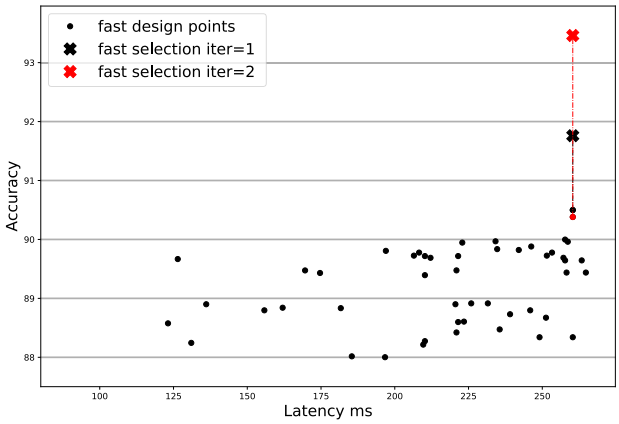
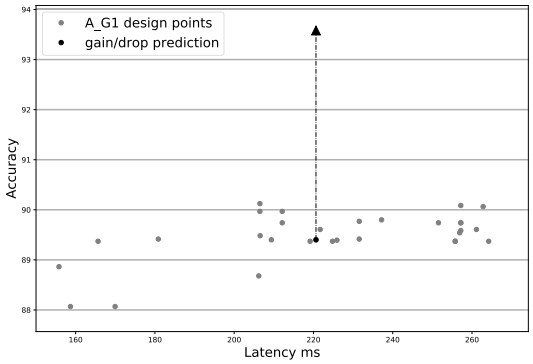


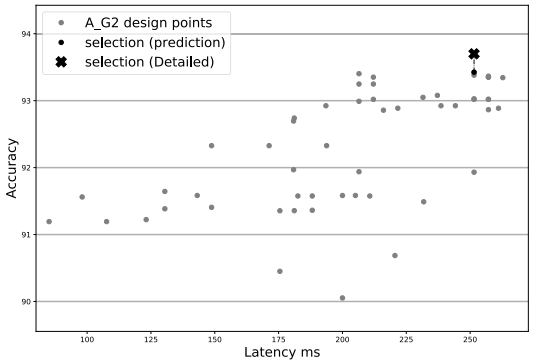
FIGURE 7. CNN architecture fast selection output in 75.7kB - 13.6 MOPS search space. The network models selected based on different choices for the *iter* value are highlighted.

training accuracy and estimated latency. Considering an *iter* value equal to 1, we select for refinement the network having the highest one-shot accuracy, highlighted in the plot. The selection output includes the pre-processing and quantization scheme associated with the CNN topology: 8bit representation for both weights and activations, and Mel-based pre-processing, resulting in 32 × 16 input spectrograms. The selected network model is finally retrained on the Detailed engine for 100 epochs, exploiting data augmentation through random shifts and random noise addition, and then quantized according to the directions resulting from the evolution search. Based on our hyper-parameters exploration, the training exploits a learning rate value $lr = 0.025$, batch size $bs = 16$, and SGD optimizer.

The co-exploration approach allows us to improve the efficiency of the design process, since, as shown in Figure 5, the pre-processing scheme's impact on performance is deeply connected with the search constraints, and consequently to the CNN architecture to be deployed. Thus performing a dedicated preliminary analysis is not only time-consuming, as anticipated in section VI-C, but also very complex, especially when considering multiple constraints, like the inference time and the network's storage requirements. As shown in Figure 9, the selected architecture reaches 94% accuracy, improving the reported state-of-the-art CNN model [4] by up to 1.8% with 40% less storage space required for weights and activations, while the number of OPS is increased by 0.3%, thus requiring a 10% higher latency. The results summary is reported in Table 6. The exploration process requires around 30 hours, considering 15 hours of one-shot training executed on NVIDIA Tesla T4 GPU, while the evolution search and the Detailed training were executed on NVIDIA Tesla P100 GPU.



(a) Pareto plot of the pruned design space after Step 3. The m exploited for the accuracy gain/drop evaluation, on pipeline ba on Mel with 32x16 input resolution, is highlighted.



(b) Pareto plot of the pruned design space after Step 5. The comparison between the predicted and training accuracy on the final selection is highlighted.

FIGURE 8. Accurate selection procedure in 75.7kB - 13.6 MOPS search space.

TABLE 7. Summary of search parameters for NAS targeting use-case 2.

Design Space	*Constraints*	*Constraints Reference*
as defined in Table 3	MOPS: 13.6 Memory 75.7kB	[5], parameters of the selected network
	Latency: 265ms	obtained based on model in Figure 3

D. USE-CASE 2

Table 7 reports the search parameters for a NAS process performed considering as a reference the work of [5]. We defined a constrained search space limited by the number of operations and the storage requirements stated for the state of the art network: 75.7 kB and 13.6 MOPS. Such an architecture achieves 95.55% accuracy, while its quantized version, exploiting mixed data representation (2.91 bits to represent activations and 2.51 bits to represent weights) reaches 93.76% accuracy.

Figure 7 reports the output of the fast selection procedure. We report two possible selections, respectively corresponding to $iter = 1$ and $iter = 2$ values. As can be derived from the plot, the second one results in a higher accuracy after the detailed training and quantization process and is referenced as the fast selection in the following. However, considering the accuracy drop connected to the selected quantization

TABLE 8. Performance metrics summary for the selected design point and the reference state of the art network, in the 75.7kB-13.6MOPS region, expressed as percentage of the constraint value.

Network model	*Accuracy*	*Latency*	*MOPS*	*Memory*
Full precision [5]	95.55%	100%	100%	100%
Quantized [5]	93.76%	100%	100%	7.8%
accurate selection	93.9%	94.9%	85.9%	95.5%
fast selection	93.46%	98.2%	89.7%	80.2%

level, we also performed the accurate selection procedure for this use case. The corresponding results are described in Figure 8. In detail, Figure 8a represents the Pareto plot of the most accurate design points belonging to the design space after the GA run, performed as Step 3 of Algorithm 2. They are depicted as grey bullets and placed according to their one-shot training accuracy, and their estimated latency. We highlight in the plot the design point corresponding to the network model exploited for the quantization drop evaluation, in Step 4. Its accuracy projection after the detailed training is also reported. The detailed training at this step was performed for 100 epochs exploiting $lr = 0.025$ and $bs = 16$. Although only one design point is depicted in the Figure, corresponding to the pre-processing pipeline based on Mel resulting in 32×16 input resolution, the same gain/drop evaluation is conducted for each of the pipelines considered. The output of the last GA run, corresponding to Step 5, is reported in Figure 8b. In this case, the design points are placed according to their predicted accuracy, considering the evaluated gain/drop corrections. The resulting selection, having the highest predicted accuracy, is highlighted on the plot. The predicted accuracy of the selected point is compared to the one really achieved with the detailed training.

The CNN architectures selected as the result of both the fast and accurate procedures are trained on Mel spectrograms of 32×16 resolution, while different quantization policies are suggested: the fast implementation results in 8bit representation for the activations, and 4bit representation for weights, whereas the accurate one results in 8bit quantization for all datatypes. Figure 10 and Table 8 report the comparison with the CNN proposed in [5]. The refined accurate selection, after additional 100 refinement training epochs in Step 6, results in an architecture reaching an accuracy 0.14% higher than the one of the quantized version of the reference state-of-the-art architecture, although having higher storage requirements, while neither of the selection procedures allows achieving the accuracy of the full precision model.

Anyway, the fast procedure allows selecting an architecture with an accuracy value only 0.34% points lower than the accurate one, exploiting only 37% of the required exploration time.

E. SELECTION'S QUALITY ASSESSMENT

To evaluate the quality of both selection procedures, we consider one of the exploration trials performed for the use-case 1 evaluation and report the detailed accuracy values for all of the considered design points, obtained through a full training exploration. Figure 11 reports the results of

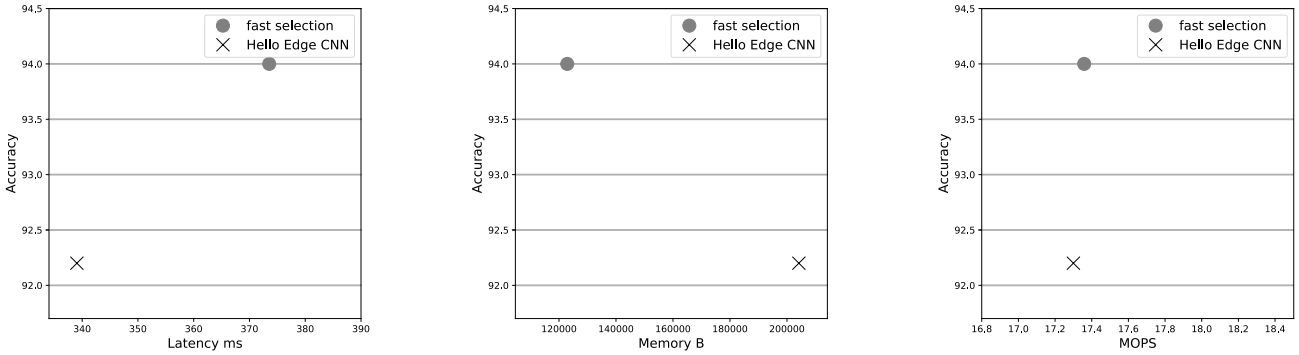


FIGURE 9. Comparison with CNN state of the art in 200kB 20 MOPS region [4].

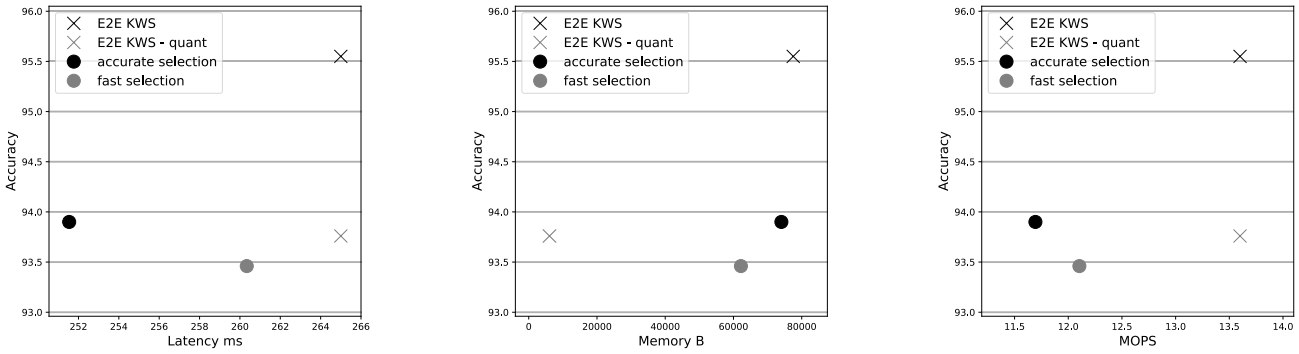


FIGURE 10. Comparison with state of the art end to end KWS network [5].

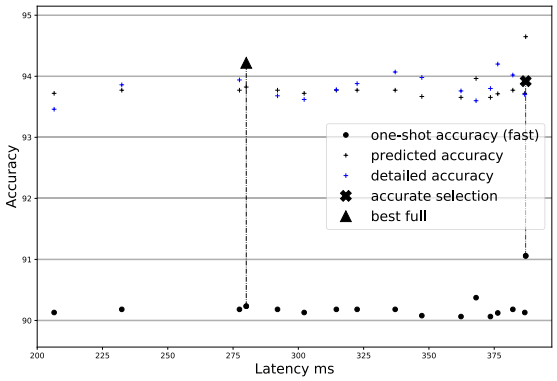


FIGURE 11. CNN architecture accurate selection output in 200kB - 20 MOPS search space. For each design point, the comparison among the predicted accuracy values and the ones achieved after detailed training is reported. The model selected by the accurate procedure is highlighted, as well as the optimal one based on the results of the full exploration.

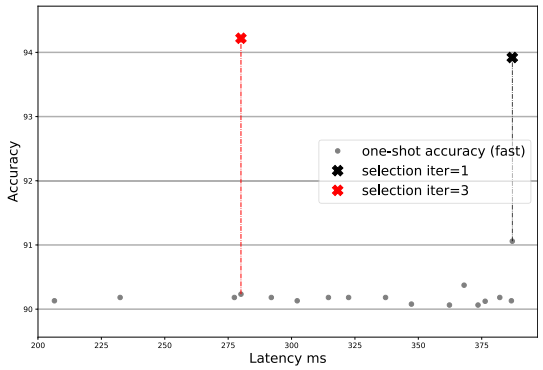


FIGURE 12. CNN architecture fast selection output in 200kB - 20 MOPS search space. The highlighted models represent the final selection resulting from an *iter* value of 1, and of 3, corresponding to the optimal one evaluated through the full exploration.

such extended exploration, showing, for each of the design points, the accuracy after one-shot training, as well as its predicted and real value upon detailed training exploiting data augmentation. As can be derived from the plot, the drop/gain prediction can be applied with sufficiently precise results. However, due to some inaccuracy of the one-shot evaluation, both the fast and accurate procedures result in the selection of a design point that does not improve as much as it is expected with the detailed training. Since this is not captured by the gain/drop evaluation, not even the accurate selection matches the overall best architecture, highlighted in the plot, instead, it requires 38% higher inference time, and has 0.3% lower accuracy. However, as shown in Figure 12, referring to the fast implementation, a value of $iter = 3$ would be sufficient to find the optimal solution. In this case, the required processing time would slightly increase to 45% of the accurate implementation one, thus still allowing for significant savings.

VIII. CONCLUSION

We presented an efficient CNN design procedure, combining NAS and quantization, for target-oriented optimal network selection, through the co-exploration of the CNN topology, and the pre-processing and quantization schemes. We also provide a more accurate exploration procedure, responding to the need of accounting for the quantization's effect on the

accuracy in order to define a proper ranking of the considered architectures. We tested both the accurate and fast implementations of the proposed procedure, considering network design targeting a KWS task running on the ST SensorTile, and considering two different use-cases posing different latency and memory constraints. Through an automated and flexible design procedure, we obtained CNN architectures that can be compared to the CNN state of the art in the KWS field, providing up to a 1.8% accuracy improvement and a 40% footprint reduction. As a further future development, this approach could be extended to other applications fields requiring similar design evaluations based on hardware and performance constraints, and grow to include more detailed performance models.

REFERENCES

[1] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

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

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

[4] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," 2017, *arXiv:1711.07128*.

[5] D. Peter, W. Roth, and F. Pernkopf, "End-to-end keyword spotting using neural architecture search and quantization," 2021, *arXiv:2104.06666*.

[6] T. Mo, Y. Yu, M. Salameh, D. Niu, and S. Jui, "Neural architecture search for keyword spotting," in *Proc. Interspeech*, Oct. 2020, pp. 1982–1986.

[7] A. Anderson, J. Su, R. Dahyot, and D. Gregg, "Performance-oriented neural architecture search," in *Proc. Int. Conf. High Perform. Comput. Simul. (HPCS)*, Jul. 2019, pp. 177–184.

[8] I. Lopez-Espejo, Z.-H. Tan, J. H. L. Hansen, and J. Jensen, "Deep spoken keyword spotting: An overview," *IEEE Access*, vol. 10, pp. 4169–4199, 2022.

[9] D. Can and M. Saraclar, "Lattice indexing for spoken term detection," *IEEE Trans. Audio, Speech, Language Process.*, vol. 19, no. 8, pp. 2338–2347, Nov. 2011.

[10] J. G. Wilpon, L. G. Miller, and P. Modi, "Improvements and applications for key word recognition using hidden Markov modeling techniques," in *Proc. Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, vol. 1, Apr. 1991, pp. 309–312.

[11] S. Ahmed, Z. A. Abbood, H. M. Farhan, B. T. Yasen, M. R. Ahmed, and A. D. Duru, "Speaker identification model based on deep neural networks," *Iraqi J. Comput. Sci. Math.*, vol. 3, no. 1, pp. 108–114, Jan. 2022.

[12] F. M. Rammo and M. N. Al-Hamdani, "Detecting the speaker language using CNN deep learning algorithm," *Iraqi J. Comput. Sci. Math.*, vol. 3, no. 1, pp. 43–52, Jan. 2022.

[13] J. Lei, T. Rahman, R. Shafik, A. Wheeldon, A. Yakovlev, O.-C. Granmo, F. Kawsar, and A. Mathur, "Low-power audio keyword spotting using tsetlin machines," *J. Low Power Electron. Appl.*, vol. 11, no. 2, p. 18, Apr. 2021. [Online]. Available: https://www.mdpi.com/2079-9268/11/2/18

[14] J. Sangeetha and S. Jothilakshmi, "A novel spoken keyword spotting system using support vector machine," *Eng. Appl. Artif. Intell.*, vol. 36, pp. 287–293, Nov. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197614001821

[15] *Keyword Spotting on Google Speech Commands*. Accessed: Feb. 1, 2022. [Online]. Available: https://paperswithcode.com/sota/keyword-spotting-on-google-speech-commands

[16] B. Kim, S. Chang, J. Lee, and D. Sung, "Broadcasted residual learning for efficient keyword spotting," in *Proc. Interspeech*, Aug. 2021, pp. 4538–4542.

[17] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming keyword spotting on mobile devices," in *Proc. Interspeech*, Oct. 2020, pp. 2277–2281.

[18] A. Berg, M. O'Connor, and M. T. Cruz, "Keyword transformer: A self-attention model for keyword spotting," in *Proc. Interspeech*, Aug. 2021, pp. 4249–4253.

[19] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2014, pp. 4087–4091.

[20] 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.

[21] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, Jul. 2019, vol. 33, no. 1, pp. 4780–4789. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/4405

[22] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-598.html

[23] 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*.

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

[25] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX

[26] S. Han, H. Mao, and W. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, 2016.

[27] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "APQ: Joint search for network architecture, pruning and quantization policy," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2075–2084.

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

[29] C. Chesta and L. Rinelli, "Modular approach to data preprocessing in Aloha and application to a smart industry use case," 2021, *arXiv:2102.01349*.

[30] P. Busia, S. Minakova, T. Stefanov, L. Raffo, and P. Meloni, "ALOHA: A unified platform-aware evaluation method for CNNs execution on heterogeneous systems at the edge," *IEEE Access*, vol. 9, pp. 133289–133308, 2021.

[31] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009, doi: 10.1145/1498765.1498785.

[32] A. Tools Community. *Open Neural Network Exchange (ONNX)*. Accessed: Oct. 27, 2021. [Online]. Available: https://onnx.ai/

[33] P. Meloni, D. Loi, P. Busia, G. Deriu, A. D. Pimentel, D. Sapra, T. Stefanov, S. Minakova, F. Conti, L. Benini, M. Pintor, B. Biggio, B. Moser, N. Shepeleva, N. Fragoulis, I. Theodorakopoulos, M. Masin, and F. Palumbo, "Optimization and deployment of CNNs at the edge: The Aloha experience," in *Proc. 16th ACM Int. Conf. Comput. Frontiers*, New York, NY, USA, Apr. 2019, pp. 326–332, doi: 10.1145/3310273.3323435.

[34] F. Conti, "Technical report: NEMO DNN quantization for deployment model," 2020, *arXiv:2004.05930*.

[35] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks," 2018, *arXiv:1805.06085*.

PAOLA BUSIA received the B.S. and M.S. degrees in electronics engineering from the University of Cagliari, Italy, in 2017 and 2019, respectively, where she is currently pursuing the Ph.D. degree in electronic and computer engineering. Her research interests include the optimization and deployment of CNNs on resource-constrained systems.

GIANFRANCO DERIU received the degree in electronics engineering from the Università degli Studi di Cagliari, in 2015. He has been a Research Assistant with the Department of Electrical and Electronics Engineering (DIEE), Università degli Studi di Cagliari, since October 2015. He is currently working on reconfigurable hardware acceleration of CNNs for artificial vision focusing on deep networks. In ALOHA, he is responsible for the overall tool flow integration and testing. His research interests include image recognition and CNNs.

LUCA RINELLI received the B.Sc. degree in computer engineering and the M.Sc. degree in software engineering from the Politecnico di Torino. He is currently a Software Engineer and the Project Manager, tasked with software development, integration, architecture design, and team coordination. In Santer Reply SpA, he has been working on international projects in the fields of embedded, HMI, AI, and security, since 2019. He has worked on the ALOHA Project on the smart industry pilot and the plugins architecture.

CRISTINA CHESTA received the M.Sc. degree in electronic engineering and the Ph.D. degree in informatics and system engineering from the Politecnico di Torino. She is currently a Researcher and the Project Manager over multiple projects, with responsibility over team coordination and software architecture design, development, and integration. She was a Visiting Scientist with the Bell Laboratories, NJ, USA, in 1998, and worked for ten years with the Motorola Global Software Group as a Software Architect and a Project Lead in different research and development projects related to human–computer interaction. In Santer Reply SpA, she has been working in international projects in the ICT domain, since 2009, and has been a Project Coordinator of HyVar and PersonAAL Projects and the responsible for the smart industry pilot in ALOHA Project.

LUIGI RAFFO joined the Department of Electrical and Electronic Engineering, University of Cagliari, Italy, in 1994, where he has been a Full Professor in electronics, since 2006. He teaches courses on system design, digital, and analog electronics design and processor architectures. Since 2012, he has been a Rector's delegate for International Research Projects. He was a Coordinator of the Course of Studies in Biomedical Engineering, from 2006 to 2012 and from 2017 to 2018. His research interests include the field of the study, design, development of systems and micro-systems for applications where high performance, high efficiency, and low power are required. In such a field, he is the author of more than 200 scientific papers.

PAOLO MELONI has been an Assistant Professor with the University of Cagliari, since 2012. He is the author of a significant track of international research papers. He teaches advanced embedded systems with the University of Cagliari and is currently a Scientific Coordinator of the ALOHA (www.aloha-h2020.eu) H2020 Project. His research interests include the development of advanced digital systems, on the application-driven design and programming of multicore on-chip architectures, and FPGAs.