IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 1 May 2023, accepted 18 May 2023, date of publication 26 May 2023, date of current version 6 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3280477

RESEARCH ARTICLE

Evaluating Robustness to Noise and Compression of Deep Neural Networks for Keyword Spotting

PEDRO H. PEREIRA, WESLEY BECCARO, AND MIGUEL A. RAMÍREZ, (Life Senior Member, IEEE)
Department of Electronic Systems Engineering, Escola Politécnica, University of São Paulo, São Paulo 05508-010, Brazil
Corresponding author: Miguel A. Ramírez (maramire@usp.br)

This work was supported in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) under Grant CAPES-PROEX 0472/2019, and in part by Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) under Grant 2018/26455-8 and Grant 2022/10909-5.

ABSTRACT Keyword Spotting (KWS) has been the subject of research in recent years given the increase of embedded systems for command recognition such as Alexa, Google Home, and Siri. Performance, model size, processing time, and robustness to noise are fundamental in these systems. Furthermore, applications in embedded systems demand computationally efficient models that can be implemented in current technology. In this work, an approach for keyword recognition is evaluated using three deep learning models namely LeNet-5, SqueezeNet, and EfficientNet-B0. We evaluate transfer learning, pruning and quantization strategies in training and test using noisy and clean speech signals. In addition, compression techniques such as pruning and quantization were assessed in terms of the size reduction of the model footprint and the accuracy obtained in each case. Using the Google's Speech Commands dataset and additive babble noise signal, our keyword recognition approach achieves an accuracy of 94.6% using an unstructured pruning of 80% of the parameters of the original SqueezeNet network with a reduction of 70% in the model size.

INDEX TERMS Speech recognition, machine learning algorithms, speech analysis, spectral analysis, pruning, quantization, keyword spotting.

I. INTRODUCTION

Voice commands are becoming a natural way to interact with consumer electronic devices [1], [2]. Systems with speech command recognition such as Amazon's Alexa, Apple's Siri, and Google's Assistant are examples of this popularity. These smart devices often use some embedded system (e.g., microcontrollers [3], microprocessors, field-programmable gate arrays, or dedicated devices [4]) with limited resources, making the implementation of speech recognition algorithms dependent on hardware limitations [5].

Typically, microcontrollers, the cheapest approach, have small memory capacity (i.e., a few kilobytes) and require energy-saving strategies since, in most cases, these edge devices are always active and they are generally powered by batteries. Additionally, they must have low latency and real-time response not to harm the user experience. There are also security and privacy concerns when audio is continuously streamed to cloud services, reinforcing the need for speech recognition, in some cases, to be performed locally thereby reducing internet data usage.

Given the hardware limitations, there is a need to obtain low-complexity solutions [6], [7], in other words, models that are computationally efficient and implementable in real time [8]. A major obstacle in porting a deep learning model to an edge device is its high memory consumption, which must be decreased in order to fit within the device. This may be brought about with a reduction in the number of parameters, which, as a positive side effect, increases energy efficiency by requiring fewer computing operations. Furthermore, the assessment of the robustness of keyword recognition models in noisy environments is in high demand [9], [10], [11], [12]. In this work, we investigate three deep learning models regarding their robustness to detect keywords with and

The associate editor coordinating the review of this manuscript and approving it for publication was Mounim A. El Yacoubi.

without the addition of babble noise. Considering the trade-off between complexity and detection performance, we also evaluate neural network compression techniques using as reference the best compromise model obtained. This leads to contributions to the deployment of deep keyword spotting models on edge devices by means of parameter pruning and quantization methods.

The novelty of this work can be summarized in three main contributions. First, in order to exploit the robustness to noise of deep keyword spotting models, a strategy for training and testing models by combining inputs with and without noise is proposed, mapping the behavior of the models in clean/noisy environments and the influence of the data in the overall accuracy of each model. Second, a methodology for analysis of the results obtained with pruning and quantization methods was deployed, which makes it straightforward to evaluate how these compression techniques impact the classification and efficiency metrics of the models and how feasible they are. Both methods can be used by embedded systems developers to assess and compare models before the final implementation. Finally, we demonstrate the advantages of SqueezeNet architecture [13] compared to the LeNet-5 [14] and EfficientNet [15] models based on the performance analysis carried out.

Addressing the issues raised above, this article is structured as follows: Section II presents theoretical aspects of KWS and speech representation. The section briefly describes the Convolutional Neural Networks (CNN) and also exemplifies some state-of-art deep learning models detailing the LeNet-5 and two architectures that have shown benefits in small footprint networks, SqueezeNet and EfficientNet-B0. Section III qualitatively describes the pruning and quantization techniques. Sections IV and V detail and discuss the experimental setup and the results obtained, respectively. Finally, Section VI highlights the findings of this paper.

II. SPEECH REPRESENTATION FOR DEEP KEYWORD SPOTTING SYSTEMS

A. KEYWORD SPOTTING STRATEGIES

The Dynamic Time Warping (DTW) [16] algorithm was one of the first strategies for speech recognition. DTW leads to an optimal alignment between time series under local and global constraints composed of a forward pass that computes a global distortion and an optional backward pass that determines the warping function [17]. Another important technique for continuous speech recognition was introduced with Hidden Markov Models (HMM) [18], where the state for a model and its sequence of states are hidden, but during training the state sequence is estimated along with the state transition probabilities and the observation probabilities for each state in the sequence. When testing an utterance, its sequence of acoustic features, taken as observations, is used for determining how likely it is to have been generated by each trained model and the more likely one is selected.

In recent years, improvements in automatic speech recognition (ASR) have been achieved thanks to the evolution of deep learning techniques [19], motivated by the increase in computational capacity as well as in available data, allowing the implementation of complex algorithms. Speech recognition systems have followed the evolution of machine learning and recently deep neural networks have become an attractive choice for ASR and KWS architectures due to their superior accuracy compared to traditional speech processing and classifier algorithms. Among the most recent deep neural networks for the command recognition task, the convolutional neural networks (CNN) and the recurrent neural networks (RNN) stand out.

KWS algorithms are focused on identifying a small set of pre-defined keywords, being, from the viewpoint of statistical classification, a reduced task when compared to ASR [20]. Several works in the literature seek to improve KWS, both in different treatments in the signal pre-processing stage and/or in improvements in the models. Different databases, implementation methods, and performance metrics can be found in the literature.

One of the first papers [21] developed in Google's laboratories employed neural networks for KWS and achieved an improvement of 45% in relation to previous models that use HMM. In a later paper, Sainath et al. [22] presented a KWS strategy using CNN obtaining an improvement between 27% to 44% in false reject rate when compared to models based on deep neural networks.

Tucker et al. [23] achieved significant progress in reducing false alarms without increasing processing and memory resources in KWS systems using the knowledge distillation technique, which is a process of transferring knowledge from a large model to a smaller one.

Zhang et al. [24] presented a study evaluating several neural network architectures oriented to memory usage, number of parameters, and operations per second. The Depthwise Separable Convolutional Neural Network (DS-CNN) architecture reached an accuracy of 95.4% in a model with 189.2 kB and 19.8 Million Operations Per Second (MOPS), a 10% better in performance when compared to simple neural networks with the same number of parameters, considering the Google's Speech Commands dataset [25], a widely used dataset for KWS performance benchmarking.

Majumdar and Ginsburg [26] proposed an end-to-end deep neural network architecture for efficient speech command recognition composed of 1D convolutional layers, batch-normalization layers, Rectified Linear Unit (ReLU) activation functions, and dropout. The model presents state-of-the-art accuracy on Google's Speech Commands dataset with fewer parameters than models with similar accuracy. By using intensive data augmentation with auxiliary background noise during training, the authors demonstrated that the model can be very robust for recognition even with background noise.

More recently, Mittermaier et al. [27] employed end-to-end deep models using as input the raw speech waveforms,

reaching an accuracy of 96.4% with only 62 k parameters using Sinc-convolution in CNN. Unlike other papers, the authors did not employ classical time-frequency representations as the input of the learning models.

B. SPEECH REPRESENTATION AND FEATURE EXTRACTION

Although deep learning models can directly employ the raw signals as input [27], the strategy of performing feature extraction for subsequent classification (i.e., feature engineering) plays a crucial role in the overall performance of the system since KWS algorithms are very sensitive to the quality of the feature representation [20]. Many techniques are available in the literature and one of the more frequent ones is based on time-frequency representations. Speech signals are non-stationary. By transforming short segments of the speech signal into the frequency domain, we highlight the varying spectral information over time without blurring its evolution if properly segmented. In a typical approach, the Short-Time Fourier Transform (STFT) of the signal is calculated as

$$X_{\hat{n}}\left(e^{j\hat{\omega}}\right) = \sum_{m=-\infty}^{\infty} x[m]w[\hat{n}-m]e^{-j\hat{\omega}m}, \tag{1}$$

where $\hat{n}$ is the discrete time index that denotes the window position and $\hat{\omega}$ corresponds to the analysis frequency. The usual implementation of the STFT is a sequence of Discrete Time Fourier Transforms (DTFT) of the signal $x_{\hat{n}}[m] = x[m]w[\hat{n}-m]$, that is, a sampled version of the DTFT of the amplitude-weighted signal, $x[m]$, multiplied by the sliding window $w[\hat{n}-m]$ [28].

The discrete STFT, $\tilde{X}_{\hat{n}}[k]$, computed using the Discrete Fourier Transform (DFT), is a sequence of Fourier transforms evaluated on a finite set of discrete frequencies $\omega_k = 2\pi k/N$ as

$$\begin{aligned}\tilde{X}_{\hat{n}}\left(e^{j(2\pi k/N)}\right) &= e^{-j(2\pi k\hat{n}/N)}\sum_{m=0}^{L-1} x[\hat{n}+m]w[-m]e^{-j(2\pi k/N)m} \\ &= \tilde{X}_{\hat{n}}[k], \quad k = 0, 1, \ldots, N-1,\end{aligned} \tag{2}$$

where L is the length of a non-causal window such that $w[-m] \neq 0$ only in $0 \leq m \leq L-1$ and $L \leq N$ [28].

The visualization of STFT is often performed via its spectrogram, which is an intensity plot of STFT magnitude over time, normally represented with a logarithmic amplitude unit.

Additionally, Mel spectrogram and Mel-Frequency Cepstral Coefficients (MFCC) are time-frequency representations based on the STFT that include aspects of human auditory perception [29].

In Mel spectrogram, a set of band-pass filters known as Mel-filter bank are applied to the power spectrum of the speech signal. The Mel-scale aims to mimic the non-linear human ear perception of sound, by being more discriminative at lower frequencies and less at higher frequencies. Considering $X_{\hat{n}}[k]$ the DFT of the $\hat{n}^{th}$ frame, the Mel-spectrum is defined as

$$\mathrm{MF}_{\hat{n}}[r] = \frac{1}{A_r}\sum_{k=L_r}^{U_r}\left|V_r[k]X_{\hat{n}}[k]\right|^2, \tag{3}$$

where $V_r[k]$ is the weighting function for the r^{th} filter ranging from DFT index L_r to U_r and $A_r = \sum_{k=L_r}^{U_r}\left|V_r[k]\right|^2$ is a normalizing factor for the r^{th} Mel-filter, considering $r = 1, 2, \ldots, R$, being R the number of Mel bands [28].

The MFCC is a representation of the short-term power spectrum of the speech signal, based on a linear cosine transform of the log power spectrum on a Mel scale of frequency, calculated as

$$\mathrm{mfcc}_{\hat{n}}[m] = \frac{1}{R}\sum_{r=1}^{R}\log(\mathrm{MF}_{\hat{n}}[r])\cos\left[\frac{2\pi}{R}\left(r+\frac{1}{2}\right)m\right], \tag{4}$$

where the $\mathrm{mfcc}_{\hat{n}}[m]$ is computed varying m from 1 to the total number of MFCC chosen (usually about 10-25).

In this work, we focus on MFCC and Mel spectrogram as inputs of the deep learning models since both representations have proven their efficiency in speech-based classification tasks and are still the most used features for KWS [20], [30].

C. DEEP NEURAL NETWORKS: LeNet-5, SqueezeNet, AND EfficientNet

Much of the recent research involving deep neural networks focuses on increasing accuracy in computer vision datasets by changing and adding new blocks in the structure of the model, such as GoogleLeNet [31], DenseNet [32], Vision Transformer [33], and VGG networks [34]. Another set of new architectures has the main objective of obtaining an architecture with a reduced number of parameters in addition to considering competitiveness related to classification evaluation metrics. That is the case of the architectures SqueezeNet [13] and EfficientNet [15].

In general, the initial layers of deep learning models are implemented using CNN, a specialized kind of neural network for processing data that has a known grid-like topology [35]. CNN are invariant to translation, scaling, and elastic deformations, being widely applied in pattern recognition tasks. LeNet-5 convolutional neural network is one of the earliest models based on this approach that has a simple and effective architecture [14].

The LeNet-5 model consists of 3 convolutional layers, 2 subsampling layers, and 2 fully connected layers. The architecture was originally conceived for image recognition (i.e., handwritten digit recognition) and achieved great success becoming a reference for deep learning models. In this work, the LeNet-5 is used as baseline.

The SqueezeNet architecture, in general, employs three main strategies. The first one is to replace filters (i.e., kernels) 3×3 with filters 1×1, as they have 9 times fewer parameters than a filter 3×3. The second one is based on decreasing the number of input channels, in order to keep a small total number of parameters in a CNN. It is important not only to decrease the size of the filters 3×3 (first strategy) but

also to decrease the number of input channels using compression layers (squeeze layers). The third strategy is based on late subsampling, at the end of the network, so that the convolution layers have large activation maps that are directly controlled by the size of the input data (e.g., images of size 256×256 pixels) and by the choice of layers in the which downsampling occurs in the architecture [13].

Most commonly, subsampling is performed into CNN architectures by setting the stride greater than 1 in some of the convolution layers or via pooling. If the first three layers of the network have large step sizes, most layers will have small activation maps. On the other hand, if most layers in the network have a step equal to 1, and steps greater than 1 are concentrated at the end of the network, then many layers in the network will have large activation maps. The intuition is that large activation maps can lead to greater classification accuracy.

Another architecture that has shown good results in pattern recognition tasks is the EfficientNet. The EfficientNet models, proposed by Tan and Le [15], achieve greater accuracy and better efficiency compared to other classic CNN architectures, reducing the number of parameters and the number of Floating-point Operations per Second (FLOPS), with high accuracy in classic computer vision datasets such as ImageNet. The authors evaluated strategies for model scaling and demonstrated, supported by empirical evidence, that scaling only one dimension (e.g., width) quickly stagnates accuracy gains. However, coupling this with an increase in the number of layers (i.e., depth) or input resolution enhances the predictive capability of the models. This implies that network sizing to increase accuracy must consider a combination of the three dimensions, employing a scaling method that uniformly modifies all depth, width, and resolution dimensions of the neural network layers using a compound coefficient. For example, if the spatial resolution of the input image is increased, the number of convolutional layers should also be increased so that the receptive field is large enough to encompass the entire image which now contains more pixels. In this way, one strategy is resizing the architecture together by a compound factor, as proposed by the EfficientNet architecture.

III. PRUNING AND QUANTIZATION TECHNIQUES

Neural network pruning is a technique that removes redundant parameters or neurons that do not contribute significantly to the performance of the model. Typically, this condition occurs when a sizable subset of the weights and biases are zero or close to zero. Besides, the networks can be retrained after being pruned. This approach offers the possibility to escape from local minima and further improve accuracy [36]. Therefore, pruning is an important strategy for reducing computational complexity due to the decrease of the network size [37] and allows, for example, neural networks to be deployed in environments such as embedded systems.

Recently, new network pruning techniques have been proposed [38], [39]. Modern pruning techniques can be classified into dynamic and static. Dynamic pruning determines at runtime which layers, channels, or neurons will not participate in further activity. Static pruning is a network optimization technique that removes neurons, offline, from the network after training and before inference. In this work, static pruning was evaluated, since there is greater support for it by the PyTorch library.

Static pruning can be classified into structured and unstructured pruning. Unstructured pruning refers to pruning individual parameters, weights and biases in linear layers and/or filters in convolution layers. The idea behind unstructured pruning is that it is possible to prune parameters without affecting their corresponding structure, hence the name unstructured pruning. As an alternative to unstructured pruning, structured pruning removes entire structures of parameters. This does not imply that the full parameter set must be removed, but, for example, in linear weights, it removes entire rows or columns, or in convolution layers, complete filters. An additional classification, pruning can be applied per layer (local) or in multiple/all layers (global).

Static pruning usually has three parts. The first one is the selection of the parameters to be pruned. The second is the method of pruning neurons, and, finally, the optional fine-tuning or retraining. Random selection and magnitude-based prune are amongst the most popular methods for finding prunable weights (pruning criterion). In magnitude-based prune, the least weight parameters are pruned based on L_n-norm, for example. For unstructured pruning, the criterion of the L_1-norm achieves the smallest specified number or ratio of parameters to be pruned. Otherwise, in structured mode, the same applies in relation to the number of structures with the lowest L_1-norm instead of parameters.

An additional step in the pruning technique is to retrain the neural network once pruned. Retraining can improve the performance of the pruned network to achieve comparable accuracy to the original network, but it requires significant time and computing resources [37].

Another technique to reduce the size of deep learning models is quantization. Generally, in the context of neural networks, quantization works by mapping the weights, biases, and filters from single-precision floating-point format (*fp32*) to low bit-width formats, such as signed integer numbers stored with 8 bits (*int8*), thus reducing the total size of the model and the inference time [40]. The built-in PyTorch library uses the *fp32* as default number representation for tensor and *qint8*, *quint8*, or *qint32* for quantization representation. Quantized tensors store quantized data in integer format (*int8*, *uint8*, or *int32*) and quantization parameters like scale and zero-point.

PyTorch provides three different quantization algorithms, which differ mainly in what moment they determine the quantization operation. The algorithms are: dynamic quantization, static quantization, also called Post-Training Quantization (PTQ), and Quantization-Aware Training (QAT). Although there are other quantization techniques proposed in the literature, this work focuses in particular on PTQ.

The function that maps values from floating-point to integer space is known as mapping function. A commonly used mapping function is the nonlinear transformation given by $Q(r) = \text{round}(\frac{r}{S} + Z)$, where r is the input, and the quantization parameters S and Z are, respectively, the scaling factor and the zero-point. S can be computed as the ratio of input range to the output range $S = \frac{\beta-\alpha}{\beta_q-\alpha_q}$, where $[\alpha, \beta]$ is the interval where permissible inputs lie and $[\alpha_q, \beta_q]$ is the interval in quantized output space that it is mapped to. For example, considering an 8-bit quantization, the output range is $\beta_q - \alpha_q = (2^8 - 1)$. The quantization parameter Z ensures that a 0 in the input space maps perfectly to a 0 in the quantized space by calculating $Z = -(\frac{\alpha}{S} - \alpha_q)$.

The process of choosing the input clipping endpoints is known as calibration. The simplest technique is to record the running minimum and maximum values and assign them to α and β, respectively. There are other alternatives to find these parameters, such as minimization of the Kullback-Leibler divergence, mean-square-error minimization, or percentiles over the input interval based on the histogram [37]. In PyTorch, the module *Observer* collects statistics on the input values and calibrates the parameters S and Z. Different calibrations result in different quantized outputs, and typically it is necessary to verify empirically which one works better for the particular application [40].

In dynamic quantization, the weights and biases of the neural network are quantized once and are held fixed thereafter, while the quantization of the activation step occurs dynamically along the inference process, with small adjustments in the scale factor made based on the observed input values until the conversion is approximately optimal. Basically, this method multiplies the input values by a scale factor, then rounds the result to the nearest integer number.

PTQ works by tuning the quantization algorithm on a set of test data after the initial model training is complete. This additional inference process is not used to adjust the model, rather it just adapts the parameters of the quantization algorithm. This is much more complex than dynamic quantization, requiring an additional pass over the dataset to work. In contrast, it is more accurate than static quantization and gives the algorithm the opportunity to calibrate using real data at once, rather than having to do it all at runtime.

In QAT, all weights and activations are falsely quantized during the forward and backward training steps (i.e., float values are rounded to *int8* values, but all calculations still are made with floating point numbers). Thus, all weight adjustments during training are performed mindfully of the fact that the model will eventually be quantized. In other words, the quantization and dequantization simulate the quantization loss and add it to the training loss during fine-tuning, making the network more resilient to quantization. In general, QAT is able to better preserve accuracy and can be an alternative when PTQ is not enough [41].

Based on good results of PTQ in CNN presented in [42], we conducted our experiments using this algorithm.

IV. EXPERIMENTAL SETUP

This work involves 8 of the 35 classes provided by the Google's Speech Commands dataset [25], specifically the words: "cat", "stop", "dog", "bird", "go", "yes", "no", and "up". In order to analyze the robustness of the speech command classifier, segments of babble noise from NOISEX-92 database [43] were added to the clean speech files. Noise, an undesirable component in the message in a communication process, is often present in the environment with virtual assistants. In special, babble noise is a signal made up of the voices of other people nearby and is one of the most challenging interferences for speech systems, due to its temporal structure and its similarity with the desired target speech [44].

Preliminary results show that the accuracy obtained with Signal-to-Noise Ratio (SNR) levels greater than 10 dB tend to deliver mostly accurate identifications. Otherwise, SNR levels lower than 5 dB impose a hard noisy condition, degrading the keyword recognition even by an attentive listener. To assess the impact of noise in the speech command classifier, the models were evaluated with clean speech signals and at two levels of SNR equal to 5 and 10 dB, respectively, as shown in Fig. 1. Training and test strategies were also evaluated by including inputs with and without babble noise.

The dataset was split into training and test subsets considering the proportion of 80% and 20%, respectively. Speech files were sampled at a rate of 16 kHz. The speech signals were segmented in frames of 20 ms overlapped by 50% (i.e., 10 ms). Each short time frame was windowed by a Hamming window function. After this, the Mel spectrogram and the 22 coefficients of MFCC of the speech signal for each frame were calculated. The use of around 20 MFCC is common in ASR, although 10-13 coefficients are often considered sufficient to encode speech signals [45]. In addition, for recognition tasks, the delta (i.e., first-order derivative) and delta-delta (i.e., second-order derivative) MFCC provide additional information about the speech temporal dynamics; however, increasing the computational cost. The preprocessing and feature extraction were done in Python with librosa [46], SciPy, and PyTorch libraries.

The audio files have different durations and generate Mel Spectrogram and MFCC outputs with different dimensions. Thus, it was necessary to standardize the network input data for each set of parameters. A padding with zeros in the original signal was considered, having as default dimension the longest duration of the training set, which is about 1 second.

Three neural networks based on the architectures of LeNet-5, SqueezeNet, and EfficientNet-B0 were evaluated. EficientNet-B0 is the base model of the EfficientNet family and presents a mobile-sized architecture. The LeNet-5 model was trained from scratch. For SqueezeNet and EfficientNet, we performed fine-tuning of pre-trained models available in the PyTorch library. This transfer learning strategy saves training time and circumvents the need for lots of data. All models were trained for 50 epochs with a batch size equal

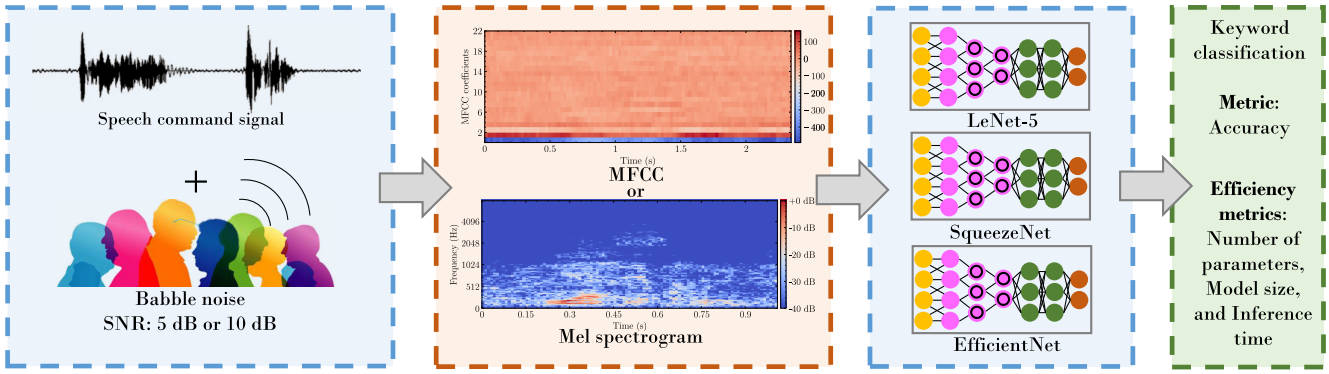

FIGURE 1. A general pipeline to assess the robustness to noise of the deep learning models: 1) babble noise is added to the speech signal, 2) MFCC or Mel spectrogram are extracted from the clean/noisy speech signal, 3) each model is trained and tested individually with one of the time-frequency representation combining inputs with and without babble noise, and 4) the 8 different classes are evaluated in terms of overall accuracy.

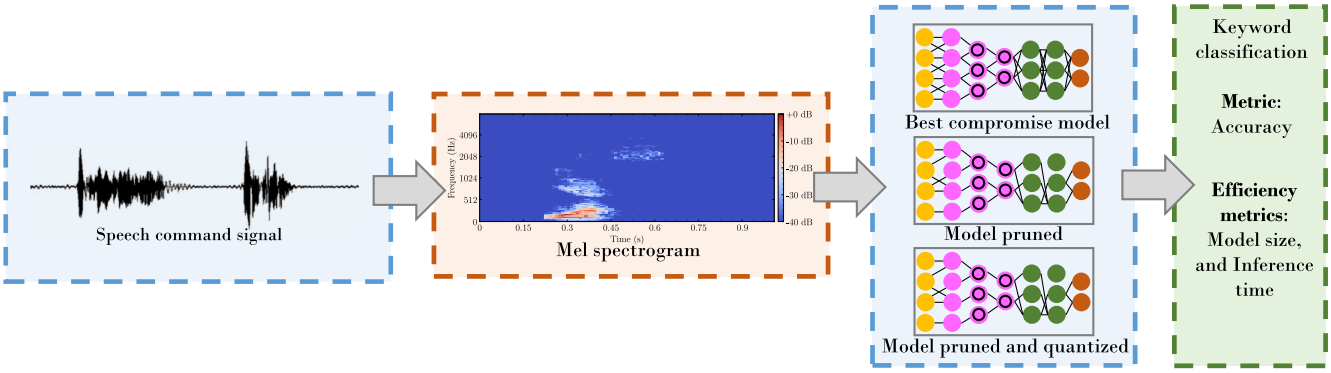

FIGURE 2. Steps to evaluate the pruning and quantization techniques: 1) the original speech signal is transformed into a Mel spectrogram, 2) the best compromise model (i.e., SqueezeNet) was compared with its pruned and quantized versions in terms of overall accuracy, size, and inference time.

to 128 using the Adam optimizer with learning rate equal to 0.001. Besides, modifications were made in the dimensions of the input and output layers to adapt the models to the time-frequency representations and the numbers of classes (i.e., 8).

As the speech command classes are balanced, the overall accuracy of the test subset was used as a comparison metric between models [47]. Considering the limitations of the embedded systems, the MFCC and Mel spectrogram were tested separately, obtaining accuracy metrics for each input. We have also evaluated the inference time of the models under comparison.

The best compromise model obtained from the classification tests was employed in the analysis of pruning and quantization techniques. To assess the effect of pruning, the technique of unstructured pruning, which removes weights and biases in a pointwise manner, and the technique of structured pruning, which removes weights in groups (e.g., removing entire filters at once), were evaluated. In both cases, the parameters of convolutional and linear layers were considered. In addition, to analyze the behavior of quantization, PTQ was evaluated for the selected model as well as all the pruned networks obtained previously. In this analysis, we evaluated the accuracy and file size for all the models pruned and pruned and quantized. The inference time was estimated for some determined conditions. Fig. 2 schematically shows the tests carried out.

It is worth mentioning that the pruning functions in the PyTorch library work just like a weight mask, assigning zero to the position of the pruned parameter and one to the unpruned one. Thus, there are no memory savings associated with using only pruning techniques. Some accelerators in hardware and/or software explore sparse neural networks to improve inference-time [48], [49]. However, this occurs without further contributions to reducing the size of the models. A viable alternative to evaluate the pruning techniques is applying algorithms that can efficiently compress the sparse matrices originating from the pruning process. An example of a compression algorithm is *gzip*, which is based on the DEFLATE [50], LZ77 [51], and Huffman coding [52] algorithms. Therefore, even if the pruned neurons are not actually removed, we can deal with the theoretical estimation of the models' size improvement. Besides, some algorithms are capable to optimize pruned neural networks by extracting the remaining structure from a pruned model, and removing all the zeroed-out neurons from the network, obtaining a simplified model [53]. In this work, *gzip* compression was applied to the SqueezeNet architecture before and after global pruning.

Experiments have been conducted on a Google Colab Pro virtual machine that provides 16 GB of RAM, an Intel Xeon CPU 2.20 GHz processor with 2 cores, and a Tesla T4 GPU accelerator with 16 GB.

V. RESULTS AND DISCUSSION

A. KEYWORD CLASSIFICATION AND EVALUATION OF ROBUSTNESS

Figure 3 shows an example of Mel spectrogram and MFCC without and with babble noise for an utterance of the keyword ''stop'' by a male speaker. Evaluating the SNR condition of 5 dB, it can be seen the decrease of gaps and/or dips in the Mel spectrogram and high intensities in regions outside the utterance making recognition of the keyword difficult. In addition, as demonstrated in [44], babble noise and speech share the same acoustic space when described using MFCC spectral features. This fact can be seen in the first coefficients of the MFCC in both conditions (5 and 10 dB SNR), which can make the distinction between speech versus babble more difficult.

Table 1 shows the classification results based on MFCC input. The LeNet-5 architecture, used here as a baseline, presents lower values of accuracy compared with the other networks in all combinations performed, also highlighting the significant decrease in accuracy when raising the additive noise level for the same combination of training and test subsets, evidencing the sensitivity of this model when dealing with noisy signals. It is also possible to observe that the SqueezeNet and EfficientNet architectures present similar accuracies in the noise addition combinations, with reduced loss of accuracy when raising the level of the SNR, highlighting the robustness of these deep networks when handling noisy speech signals. In the experiments carried out with MFCC, the EfficientNet network, trained and tested with additive noise for an SNR at 5 dB, presented the best performance with an accuracy of 89.8%.

Table 2 presents the results of classification referring to the Mel Spectrogram as input. The LeNet-5 architecture still presents the lowest results in all combinations performed. However, when comparing them with previous MFCC results, it is possible to observe an increase in accuracy for all combinations of training and test subsets. This observation is also valid for the SqueezeNet and EfficientNet architectures, which in addition to presenting similar results, it was also possible to observe the robustness of these deep models in terms of adding babble noise at different intensities when trained and tested with another time-frequency representation. In absolute terms, the noiseless training and test scenario on the SqueezeNet network performed better, with 94.9% accuracy. By way of comparison, the three models with highest accuracy scores obtained with the Google Speech Commands V2 12 (i.e., the reduced dataset for 12 commands) are respectively the Broadcasting-residual network (BC-ResNet-8) with accuracy equal to 98.7% [54], the KWT-3 model, based on Transformer architecture, with 98.56% [55], and the Wav2KWS model with 98.5% [56]. These models use different strategies of speech representation (e.g., the latent representation based on wav2vec 2.0 [57] employed as input in the Wav2KWS model) and not all have size and/or memory restrictions as is the case of KWT-3 and Wav2KWS models. In addition, Peter et al. [58] compared an end-to-end keyword spotting model with its trained bit-width quantized version obtaining respectively accuracy equal to 95.55% and 93.76% for a dataset with 10 commands.

In the condition of training without noise and testing with noise, both the SqueezeNet and the EfficientNet models did not show a significant decrease in terms of accuracy for MFCC and the Mel Spectrogram inputs, demonstrating that these models are robust in the presence of babble noise. On the other hand, the LeNet-5 model showed considerable degradation in the accuracy results.

The results of the combination of training with noise and testing without noise demonstrated the generalization capacity of the trained models since the models were able to retain features from the keywords even under noisy conditions. Furthermore, it intuits the possibility of using babble noise as data augmentation.

Table 3 presents a comparative analysis of the models regarding the number of parameters, inference time, and the size of networks considering as input the Mel Spectrogram (i.e., models with greater number of weights and biases).

The SqueezeNet architecture presents good results in terms of the number of parameters, model size, and accuracy, obtaining 94.9% in accuracy with only 727 k parameters and size of the model equal to 2.9 MB. Although the number of parameters of LeNet-5 is smaller when compared to the other models, the LeNet-5 presented low accuracy. The accuracy of SqueezeNet is slightly greater than the accuracy of the EfficientNet, 94.5%. Although the number of parameters of the EfficientNet is 6.87 times greater than the SqueezeNet, the inference time for both models is similar. This is expected since the models of the EfficientNet family are designed to focus on increasing accuracy performance while minimizing the overall number of operations required [15], [59]. Considering file size and accuracy as decision-making metrics for the choice of deep keyword spotting models, among the selected one, SqueezeNet can be considered the best compromise model.

Figure 4 illustrates the confusion matrix of the SqueezeNet architecture trained and tested without babble noise and with Mel Spectrogram as input. In this condition, the accuracy achieved 94.9%. From the confusion matrix, the keywords ''up'', ''yes'', and ''cat'' present the best values of true positive rate (TPR), 97.4%, 96.4%, and 96.4%, respectively. Otherwise, the keywords ''stop'', ''dog'', ''bird'', ''go'', and ''no'' present a TPR of 95.9%, 91.4%, 96.2%, 93.5%, and 93.6%, respectively. The SqueezeNet model presented incorrect predictions in ''cat'' and ''up'', ''dog'' and ''stop'', and ''go'' and ''no'', this latter pair with very similar-sounding words. This same result can also be verified for the other

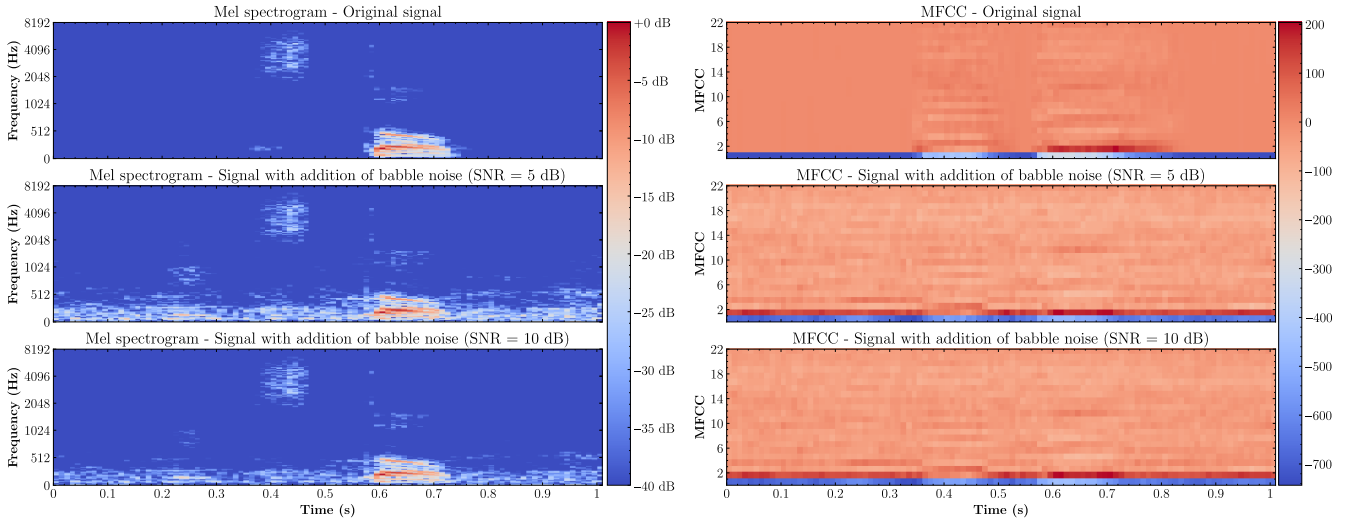

FIGURE 3. Representations of the Mel spectrogram and MFCC without and with babble noise (SNR equal to 5 and 10 dB) for an utterance of the keyword "stop" by a male speaker.

TABLE 1. Classification accuracy for training and test data with and without babble noise, considering the MFCC as input of the models. The column "No noise/No noise" presents the accuracy of the model trained and tested without babble noise, whereas, for example, the column "10 dB SNR/No noise" describes the accuracy of the models trained on data at SNR level of 10 dB and tested on data without babble noise.

Training/ Test / Deep learning model	No noise/ No noise	No noise/ 5 dB SNR	No noise/ 10 dB SNR	5 dB SNR/ No noise	10 dB SNR/ No noise	5 dB SNR/ 5 dB SNR	10 dB SNR/ 10 dB SNR
LeNet-5	78.5%	65.5%	62.7%	52.3%	48.6%	72.3%	67.6%
SqueezeNet	83.4%	84.2%	83.9%	82.8%	81.3%	88.8%	86.5%
EfficientNet-B0	87.1%	85.4%	85.7%	89.2%	87.3%	89.8%	89.1%

TABLE 2. Classification accuracy for training and test data with and without babble noise, considering the Mel spectrogram as input of the models. The description of the columns follows as presented in Table 1.

Training/ Test / Deep learning model	No noise/ No noise	No noise/ 5 dB SNR	No noise/ 10 dB SNR	5 dB SNR/ No noise	10 dB SNR/ No noise	5 dB SNR/ 5 dB SNR	10 dB SNR/ 10 dB SNR
LeNet-5	76.5%	66.4%	62.2%	65.7%	64.7%	78.7%	66.8%
SqueezeNet	94.9%	91.5%	90.1%	90.8%	89.7%	92.1%	91.9%
EfficientNet-B0	94.5%	90.7%	89.3%	92.1%	91.2%	91.3%	90.1%

models and conditions assessed and depicts a typical limitation of the classification models.

Considering that the best compromise results were obtained with SqueezeNet, the pruning and quantization techniques were evaluated using this model.

B. COMPRESSION OF THE SQUEEZENET MODEL

In general, deep neural networks have a large number of weights and biases with values very close to zero. In these cases, these connections contribute little to the model's inference. Fig. 5 exemplifies the frequency distribution of the weights and biases of the SqueezeNet architecture before and after the pruning (unstructured pruning of 80% of parameters). From the total number of weights and biases (i.e., 726,280), 723,627 are between −0.3 and 0.3, that is, 0.3% of the parameters have absolute values greater than 0.3. The largest and smallest values found are 4.44542 and −1.4677583, respectively.

The first pruning technique applied was the structure one, using *LnStructured* method from PyTorch, which prune entire (currently unpruned) channels in a tensor based on their L_1-norm. As pointed out in Section IV, after pruning, a fine adjustment of the network was performed, retraining the model for additional 5 epochs.

Table 4 presents the number of non-zero parameters, the size of the model (after compressed by the *gzip* algorithm), and the results of accuracy for tests varying pruning from 0% (original model) to 95%. By pruning up to 50% of the network channels, the accuracy after the fine adjustment undergoes a small deterioration of about 1.4% when compared to the network without pruning. When pruning 60% of the parameters, however, the accuracy drops dramatically to 15.6%. At the same time, it was possible to observe that approximately half of the non-zero parameters preserved the prediction condition satisfactorily.

The second pruning technique applied was the unstructured one, using *L1unstructured* method from PyTorch.

TABLE 3. Comparative analysis of the models LeNet-5, SqueezeNet, and EfficientNet-B0 in terms of number of parameters, size (MB), inference time, and accuracy, considering Mel spectrogram as input of the models and training and test datasets without babble noise. Inference time (mean ± standard deviation) obtained with the input of one sample and models loaded only in the Central Processing Unit (CPU).

Deep learning model	Number of parameters	File size	Inference time (ms)	Accuracy
LeNet-5	∼680 k	2.7 MB	0.96 ± 0.22	76.5%
SqueezeNet	∼727 k	2.9 MB	26.89 ± 3.72	94.9%
EfficientNet-B0	∼5 M	21.0 MB	23.99 ± 3.38	94.5%

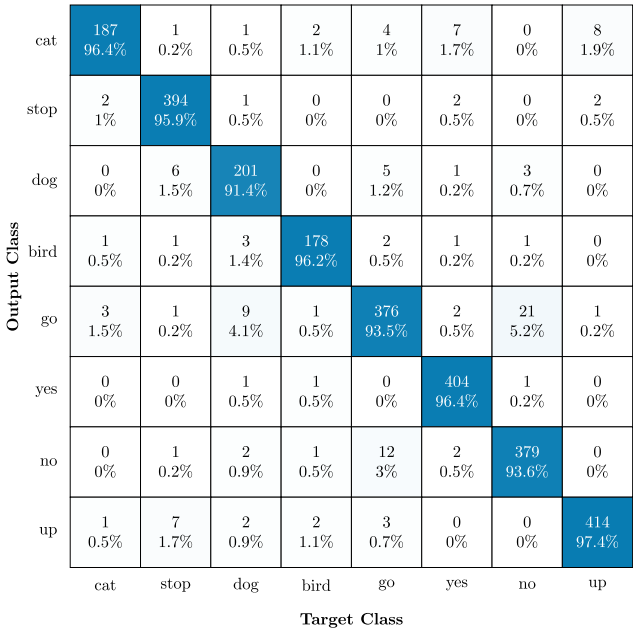

FIGURE 4. Confusion matrix with absolute values and relative percentages to evaluate the performance of the SqueezeNet model (best compromise model) trained and tested without babble noise and with Mel Spectrogram as input. The rows hereby indicate the output class, corresponding to the SqueezeNet output, whereas the columns indicate the target (i.e., real class).

This pruning technique operates on the units of each tensor zeroing out those with the lowest L_1-norm. With a similar approach to the previous one, experiments varying pruning from 0% (original model) to 95% in the proportion of parameters were considered. As previously described, a fine adjustment of the network was performed after pruning, retraining the model for additional 5 epochs. Table 5 presents the number of non-zero parameters of the model, the size of the compressed model, and the accuracy obtained.

Pruning up to 80% of the network parameters causes a small deterioration in the accuracy of the model of around 0.4% when compared to the network without pruning. It is interesting to highlight an increase in the accuracy in cases of pruning rate equal to 10%, 20%, 30%, and 50% reaching more than 95%, suggesting an improvement in the generalization capacity of these models, however, this fact requires further experiments to validate this hypothesis. When pruning 90% of the parameters, the accuracy drops by about 7.2% when compared to the original model. Thus, one can observe that 10% of the parameters, about 75 k non-zero weights and biases preserve the network prediction satisfactorily.

TABLE 4. Structured pruning of the SqueezeNet model. Number of non-zero parameters and accuracy of the model after pruning entire layers and channels based on their L_1-norm. The original model is indicated as 0%. For the pruned conditions, accuracy is the mean achieved across 5 different trained models.

Channels pruned	Number of non-zero parameters	File size (*gzip*)	Accuracy after fine-tuning
0%	727 k	2.6 MB	94.9%
10%	655 k	2.4 MB	94.4%
20%	580 k	2.2 MB	95.4%
30%	510 k	2.0 MB	94.7%
40%	436 k	1.8 MB	94.6%
50%	365 k	1.5 MB	93.5%
60%	293 k	1.3 MB	15.6%
70%	218 k	1.1 MB	15.2%
80%	148 k	0.79 MB	15.3%
90%	73 k	0.36 MB	15.2%
95%	37 k	0.36 MB	15.2%

TABLE 5. Unstructured pruning of the SqueezeNet model. Number of non-zero parameters and accuracy of the model after pruning units of weights and biases by zeroing out the ones with the lowest L_1-norm. The original model is indicated as 0%. For the pruned conditions, accuracy is the mean achieved across 5 different trained models.

Parameters pruned	Number of non-zero parameters	File size (*gzip*)	Accuracy after fine-tuning
0%	727 k	2.6 MB	94.9%
10%	654 k	2.5 MB	95.1%
20%	581 k	2.3 MB	95.0%
30%	509 k	2.1 MB	95.3%
40%	437 k	1.8 MB	94.7%
50%	364 k	1.6 MB	95.4%
60%	292 k	1.4 MB	94.8%
70%	220 k	1.1 MB	94.5%
80%	147 k	0.84 MB	94.6%
90%	75 k	0.39 MB	87.7%
95%	39 k	0.39 MB	21.3%

Otherwise, the accuracy dropped out to 21.3% when 95% of the parameters were pruned, demonstrating the limits of this procedure. Fig. 6(b) comparatively illustrates the accuracy results with the structured and unstructured pruning approaches.

Figure 6(a) presents the results of the *gzip* compression applied in the SqueezeNet model after unstructured and structured pruning. A linear behavior between the percentage of pruning and the model size obtained with *gzip* compression can be observed for both unstructured and structured pruning techniques from 0% (i.e., SqueezeNet model without pruning and after compression) to 90%. The compressed models with structured pruning are a little smaller, but in the pruning conditions of 40% and 70%, they presented equivalent sizes. For the case of structured pruning of 50%, the size obtained was 1.5 MB, that is, a reduction of 42.3% when compared

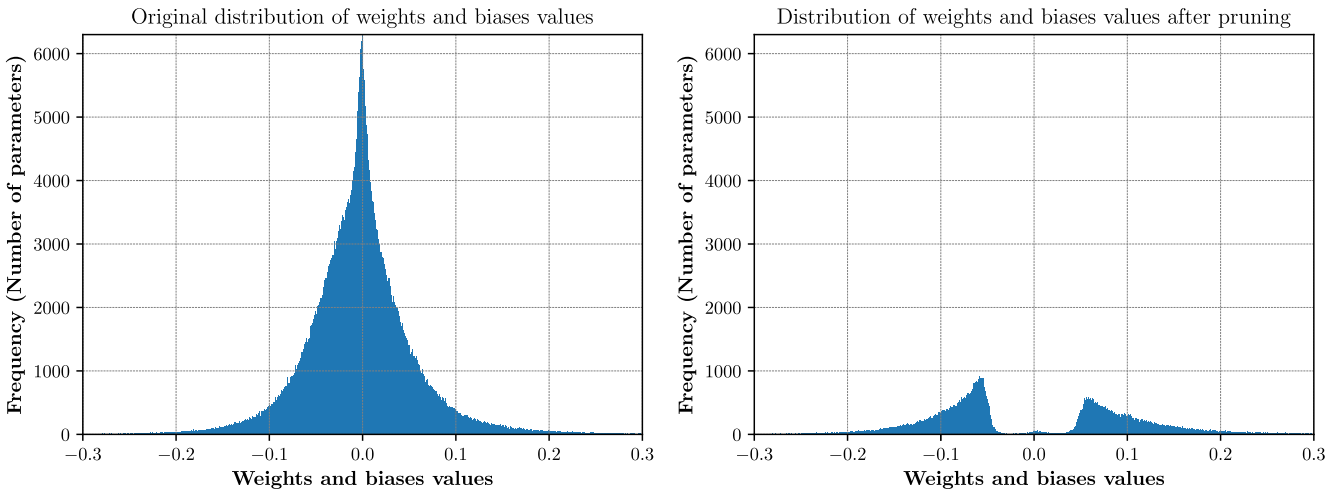

FIGURE 5. Histograms of the weights and biases obtained after training the SqueezeNet network (original distribution of weights and biases values) and after pruning of 80% of parameters (unstructured pruning and fine-tuning). Histograms constructed using a number of bins equal to 1000 in the range of −0.3 to 0.3.

with the 2.6 MB of the original model, described in Fig. 6 with 0% of pruning. For the case of unstructured pruning of 80%, the size obtained was 0.84 MB, that is, a reduction of 67.7% when compared with the 2.6 MB of the original model. It is worth mentioning that the original size of the model is 2.9 MB and the size obtained after *gzip* is equal to 2.6 MB.

To analyze the behavior of SqueezeNet with quantization, the technique of PTQ was evaluated. As previously described, PTQ involves not only converting the parameters from *fp32* to *int8* but also using the training data in batches on the network and computing the distributions resulting from the different activations. Specifically, this is done by inserting observer modules at different points that record this data. These distributions are used to determine how different activations should be quantized during inference. A simple technique would simply be to divide the entire range of activations into 256 levels. This additional step allows passing quantized values between operations instead of converting these values to *float*—and then back to *int*—between each operation, resulting in a significant speed up. Three observers provided in PyTorch were evaluated: the *MinMaxObserver*, which computes the quantization parameters based on tensor min/max statistics; the *MovingAverageMinMaxObserver*, based on the moving averages of min/max of the incoming tensors; and *HistogramObserver*, that searches for the min/max values ensuring the minimization of the quantization error with respect to the floating point model. Empirically, the *HistogramObserver* showed better results in preliminary tests, and, therefore, was employed in the experiments.

Tables 6 and 7 present the comparison of model size and accuracy after applying the quantization in structured and unstructured pruned models, respectively. There is a considerable reduction of 73.4% in the size of the models (original to quantized) with the use of quantization as pointed out in Fig. 7(a). However, for a pruning rate of 30%, accuracy decreases considerably from 94.9% to 76.4% in the structured pruning and from 94.9% to 73.4% in the unstructured pruning (i.e., a reduction of approximately 19.5% and 22.6%, respectively) as can also be seen in Fig. 7(b). Comparing the results obtained in Tables 5 and 7, for an unstructured pruning with a rate equal to 80% it was possible to achieve an accuracy of 94.6%, even though, after quantization, the accuracy drops to 57.3%, demonstrating the limitations of the quantization. In general, the accuracy was negatively affected by the quantization.

TABLE 6. File size (original and after *gzip* compression) and accuracy of the structured pruned SqueezeNet before and after the PTQ. For the quantized and pruned conditions, accuracy is the mean achieved across 5 different trained models.

SqueezeNet model	File size	File size (*gzip*)	Accuracy
0% Original (*fp32*)	2.9 MB	2.6 MB	94.9%
Quantized (0% pruned)	0.77 MB	0.56 MB	76.4%
Quantized (10% pruned)	0.77 MB	0.56 MB	73.6%
Quantized (20% pruned)	0.77 MB	0.54 MB	68.7%
Quantized (30% pruned)	0.77 MB	0.51 MB	68.4%
Quantized (40% pruned)	0.77 MB	0.47 MB	54.6%
Quantized (50% pruned)	0.77 MB	0.43 MB	57.7%
Quantized (60% pruned)	0.77 MB	0.38 MB	15.3%
Quantized (70% pruned)	0.77 MB	0.31 MB	15.6%
Quantized (80% pruned)	0.77 MB	0.24 MB	15.4%
Quantized (90% pruned)	0.77 MB	0.10 MB	15.2%
Quantized (95% pruned)	0.77 MB	0.09 MB	15.2%

Figure 7(a) also describes how the size of the models after compressed by the *gzip* algorithm monotonically decreases achieving, for example, 0.09 MB for the case of quantization of the SqueezeNet model structured pruned with a rate equal 95%.

Different from Fig. 6(b), in which the accuracy remained approximately constant up to a pruning rate of 90% for the case of unstructured pruning and up to 50% for structured pruning, in Fig. 7(b), for both pruning methods, there is a decrease in accuracy for all the quantized and pruned models when compared to the original model only quantized (i.e., quantized and 0% pruned).

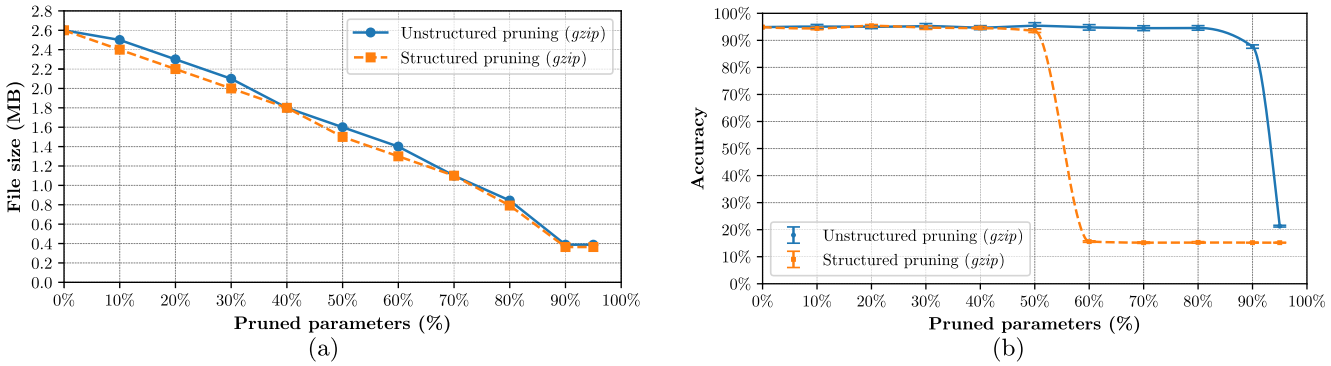

FIGURE 6. Evaluation of (a) File size (*gzip*) and (b) Accuracy *versus* percentage of pruned parameters of the SqueezeNet model obtained after the unstructured (solid blue line) and structured (dashed orange line) pruning. Error bars are within one standard deviation around the mean.

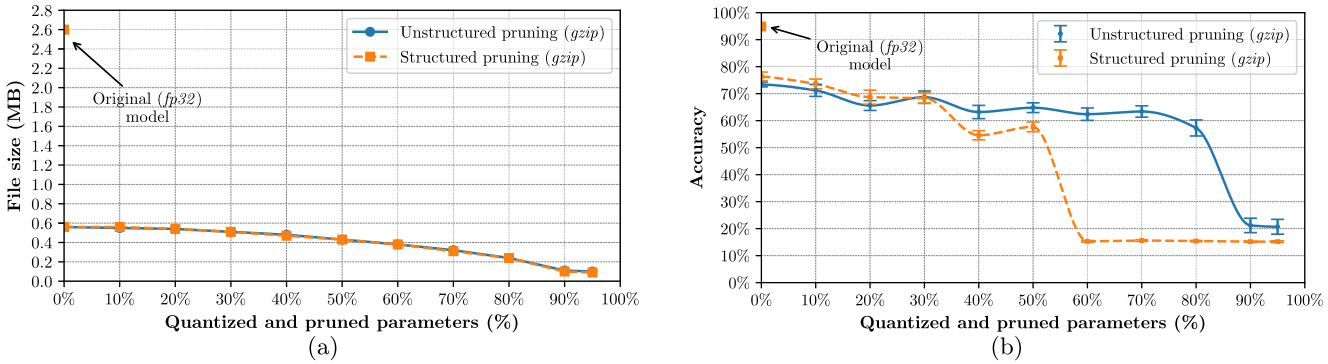

FIGURE 7. Evaluation of File size (*gzip*) and (b) Accuracy *versus* percentage of quantized and pruned parameters of the SqueezeNet model obtained after the unstructured (solid blue line) and structured (dashed orange line) pruning. By way of comparison, the results of file size and accuracy obtained with the original model are also indicated at 0%. Error bars are within one standard deviation around the mean.

TABLE 7. File size (original and after *gzip* compression) and accuracy of the unstructured pruned SqueezeNet before and after the PTQ. For the quantized and pruned conditions, accuracy is the mean achieved across 5 different trained models.

SqueezeNet model	File size	File size (*gzip*)	Accuracy
0% Original (*fp32*)	2.9 MB	2.6 MB	94.9%
Quantized (0% pruned)	0.77 MB	0.56 MB	73.4%
Quantized (10% pruned)	0.77 MB	0.55 MB	71.2%
Quantized (20% pruned)	0.77 MB	0.54 MB	65.6%
Quantized (30% pruned)	0.77 MB	0.51 MB	68.7%
Quantized (40% pruned)	0.77 MB	0.48 MB	63.2%
Quantized (50% pruned)	0.77 MB	0.43 MB	64.8%
Quantized (60% pruned)	0.77 MB	0.38 MB	62.4%
Quantized (70% pruned)	0.77 MB	0.32 MB	63.4%
Quantized (80% pruned)	0.77 MB	0.24 MB	57.3%
Quantized (90% pruned)	0.77 MB	0.11 MB	21.2%
Quantized (95% pruned)	0.77 MB	0.10 MB	20.7%

Table 8 presents the inference time for the original SqueezeNet, 80% unstructured pruned SqueezeNet, and 80% unstructured pruned SqueezeNet with additional PTQ. The selection of these models was not arbitrary. They represent approximately the values computed for the other pruning and quantization cases tested. In summary, due to caveats of the pruning functions of the PyTorch library, the same inference times are obtained for other pruning rates (approximately 24 ms) and other quantization and pruned conditions (approximately 9 ms). Based on the results described in the

TABLE 8. Inference time (s) measured for the original (dense) SqueezeNet model, unstructured pruned (80%) SqueezeNet, and unstructured pruned (80%) SqueezeNet with PTQ. Inference time (mean ± standard deviation) obtained with the input of one sample and models loaded only in the CPU.

SqueezeNet model	Inference time (ms)
Original (*fp32*)	26.89 ± 3.72
80% pruned	24.47 ± 1.46
Quantized (80% pruned)	9.12 ± 1.19

Table 8, there is a small improvement in inference time with the pruned model when compared to the original one, due to the strategy of assigning zeros to the parameters of the model performed by PyTorch. Otherwise, PTQ speeds up the inference time by 66% when compared to the original model, demonstrating a tradeoff of the quantization algorithm since, while the quantization decreases the model size and the inference time, the accuracy is partially affected.

VI. CONCLUSION

Voice-controlled devices are dependent on models that are implementable in real time and work even in noisy environments. This work presents initially an evaluation of the robustness to noise of the LeNet-5, SqueezeNet, and EfficientNet-B0 models, applied in the KWS task. We have tested 8 classes from Google's Speech Commands dataset with additive babble noise at SNR levels of 5 and 10 dB. Experiments have also been conducted on clean speech.

SqueezeNet and EfficientNet-B0 models did not show a significant decrease in accuracy in the presence of babble noise, demonstrating that these models preserve the ability to recognize keywords even in noisy environments. Based on the models and conditions presented throughout the article, the results indicate the SqueezeNet architecture as the best compromise model, which combined good accuracy with a small number of parameters (i.e., size). Based on this model, we have assessed some pruning and quantization techniques. As demonstrated with the SqueezeNet architecture, a reduction of 80% of the parameters by unstructured pruning technique did not affect significantly the accuracy of the model, decreasing from 94.9% to 94.6%. Although PTQ improved the performance of SqueezeNet in terms of size and inference time, the algorithm negatively affected the performance of the model in terms of accuracy, which suggests a limitation of the quantization strategy. Therefore, as shown in the results, the pruning technique allows greater compression of the models along with insignificant loss of accuracy. However, this approach is still limited to the few hardware and/or software resources for optimization and simplification of models or algorithms to speed up the calculation of sparse matrices. On the other hand, PTQ significantly reduces the size of the models, however, adversely affecting their accuracy. Strategies for optimizing the calculation of integer and/or rational fixed-point parameters have been widely explored in digital signal processors, which makes the quantization technique a possible strategy for embedded systems. Finally, both techniques still require more research to boost the use of deep learning models in embedded systems.

REFERENCES

[1] S. Leem, I. Yoo, and D. Yook, "Multitask learning of deep neural network-based keyword spotting for IoT devices," *IEEE Trans. Consum. Electron.*, vol. 65, no. 2, pp. 188–194, May 2019.

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

[3] G. Cerutti, L. Cavigelli, R. Andri, M. Magno, E. Farella, and L. Benini, "Sub-mW keyword spotting on an MCU: Analog binary feature extraction and binary neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 5, pp. 2002–2012, May 2022.

[4] W. Shan, M. Yang, T. Wang, Y. Lu, H. Cai, L. Zhu, J. Xu, C. Wu, L. Shi, and J. Yang, "A 510-nW wake-up keyword-spotting chip using serial-FFT-based MFCC and binarized depthwise separable CNN in 28-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 56, no. 1, pp. 151–164, Jan. 2021.

[5] Y. Gong, Y. Li, X. Ding, H. Yang, Z. Zhang, X. Zhang, W. Ge, Z. Wang, and B. Liu, "QCNN inspired reconfigurable keyword spotting processor with hybrid data-weight reuse methods," *IEEE Access*, vol. 8, pp. 205878–205893, 2020.

[6] T. Heittola, A. Mesaros, and T. Virtanen, "Acoustic scene classification in DCASE 2020 challenge: Generalization across devices and low complexity solutions," 2020, *arXiv:2005.14623*.

[7] I. Martín-Morató, F. Paissan, A. Ancilotto, T. Heittola, A. Mesaros, E. Farella, A. Brutti, and T. Virtanen, "Low-complexity acoustic scene classification in DCASE 2022 challenge," 2022, *arXiv:2206.03835*.

[8] C. Bucilua, R. Caruana, and A. Niculescu-Mizil, "Model Compression," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. New York, NY, USA: Association for Computing Machinery, 2006, pp. 535–541.

[9] D. Kim, K. Ko, D. K. Han, and H. Ko, "Discriminatory and orthogonal feature learning for noise robust keyword spotting," *IEEE Signal Process. Lett.*, vol. 29, pp. 1913–1917, 2022.

[10] I. López-Espejo, Z. Tan, and J. Jensen, "A novel loss function and training strategy for noise-robust keyword spotting," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 29, pp. 2254–2266, 2021.

[11] A. Mohanty, A. Frischknecht, C. Gerum, and O. Bringmann, "Behavior of keyword spotting networks under noisy conditions," in *Artificial Neural Networks and Machine Learning ICANN*, I. Farkaš, P. Masulli, S. Otte, and S. Wermter, Eds. Cham, Switzerland: Springer, 2021, pp. 369–378.

[12] H.-J. Park, P. Zhu, I. L. Moreno, and N. Subrahmanya, "Noisy student-teacher training for robust keyword spotting," in *Proc. Interspeech*, Aug. 2021, pp. 331–335.

[13] 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*.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

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

[16] H. Sakoe and S. Chiba, "Dynamic-programming approach to continuous speech recognition," in *Proc. Int. Congr. Acoust.*, 1971, pp. 65–69.

[17] C. Kim and K.-D. Seo, "Robust DTW-based recognition algorithm for hand-held consumer devices," *IEEE Trans. Consum. Electron.*, vol. 51, no. 2, pp. 699–709, May 2005.

[18] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.

[19] P. Busia, G. Deriu, L. Rinelli, C. Chesta, L. Raffo, and P. Meloni, "Target-aware neural architecture search and deployment for keyword spotting," *IEEE Access*, vol. 10, pp. 40687–40700, 2022.

[20] D. B. de Souza, K. J. Bakri, F. d. S. Ferreira, and J. Inacio, "Multitaper-mel spectrograms for keyword spotting," *IEEE Signal Process. Lett.*, vol. 29, pp. 2028–2032, 2022.

[21] 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.

[22] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proc. Interspeech*, Sep. 2015, pp. 1–5.

[23] G. Tucker, M. Wu, M. Sun, S. Panchapagesan, G. Fu, and S. Vitaladevuni, "Model compression applied to small-footprint keyword spotting," in *Proc. Interspeech*, Sep. 2016, pp. 1878–1882.

[24] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," 2018, *arXiv:1711.07128*.

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

[26] S. Majumdar and B. Ginsburg, "MatchboxNet: 1D time-channel separable convolutional neural network architecture for speech commands recognition," 2020, *arXiv:2004.08531*.

[27] S. Mittermaier, L. Kürzinger, B. Waschneck, and G. Rigoll, "Small-footprint keyword spotting on raw audio data with sinc-convolutions," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 7454–7458.

[28] R. L. Rabiner and W. R. Schafer, *Theory and Applications of Digital Speech Processing*. Upper Saddle River, NJ, USA: Pearson, 2011.

[29] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *J. Acoust. Soc. Amer.*, vol. 8, no. 3, pp. 185–190, Jan. 1937.

[30] Z. K. Abdul and A. K. Al-Talabani, "Mel frequency cepstral coefficient and its applications: A review," *IEEE Access*, vol. 10, pp. 122136–122158, 2022.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[32] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2261–2269.

[33] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16×16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.

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

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[36] B. Choi, J.-H. Lee, and D.-H. Kim, "Solving local minima problem with large number of hidden nodes on two-layered feed-forward artificial neural networks," *Neurocomputing*, vol. 71, nos. 16–18, pp. 3640–3643, Oct. 2008.
[37] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, Oct. 2021.
[38] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning," Mar. 2020, *arXiv:2003.03033*.
[39] A. Gholami, S. Kim, Z. Dong, Z. Yao, W. Michael Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," Jun. 2021, *arXiv:2103.13630*.
[40] S. Subramanian. (Feb. 8, 2022). Practical Quantization in PyTorch. PyTorch. Accessed: May 30, 2023. [Online]. Available: https://pytorch.org/blog/quantization-in-practice/
[41] R. Krishnamoorthi, J. Reed, M. Ni, C. Gottbrath, and S. Weidman. (Mar. 26, 2020). Introduction to Quantization on PyTorch. PyTorch. Accessed: May 30, 2023. [Online]. Available: https://pytorch.org/blog/introduction-to-quantization-on-pytorch/
[42] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*.
[43] A. Varga and H. J. M. Steeneken, "Assessment for automatic speech recognition: II. NOISEX-92: A database and an experiment to study the effect of additive noise on speech recognition systems," *Speech Commun.*, vol. 12, no. 3, pp. 247–251, Jul. 1993.
[44] N. Krishnamurthy and J. H. L. Hansen, "Babble noise: Modeling, analysis, and applications," *IEEE Trans. Audio, Speech, Language Process.*, vol. 17, no. 7, pp. 1394–1407, Sep. 2009.
[45] A. Hagen, D. A. Connors, and B. L. Pellom, "The analysis and design of architecture systems for speech recognition on modern handheld-computing devices," in *Proc. 1st IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, 2003, pp. 65–70.
[46] B. McFee, C. Raffel, D. Liang, D. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "Librosa: Audio and music signal analysis in Python," in *Proc. 14th Python Sci. Conf.*, 2015, pp. 18–25.
[47] A. Tharwat, "Classification assessment methods," *Appl. Comput. Informat.*, vol. 17, no. 1, pp. 168–192, Jan. 2021.
[48] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 9, pp. 1953–1965, Sep. 2020.
[49] X. Zhou, Z. Du, S. Zhang, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Addressing sparsity in deep neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 10, pp. 1858–1871, Oct. 2019.
[50] P. Deutsch, *DEFLATE Compressed Data Format Specification Version 1.3*, document RFC 1951, 1996, pp. 1–17.
[51] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. IT-23, no. 3, pp. 337–343, May 1977.
[52] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.
[53] A. Bragagnolo and C. A. Barbano, "Simplify: A Python library for optimizing pruned neural networks," *SoftwareX*, vol. 17, Jan. 2022, Art. no. 100907.
[54] B. Kim, S. Chang, J. Lee, and D. Sung, "Broadcasted residual learning for efficient keyword spotting," in *Proc. Interspeech*, Aug. 2021, pp. 4538–4542.
[55] 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.
[56] D. Seo, H. Oh, and Y. Jung, "Wav2KWS: Transfer learning from speech representations for keyword spotting," *IEEE Access*, vol. 9, pp. 80682–80691, 2021.
[57] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "Wav2Vec 2.0: A framework for self-supervised learning of speech representations," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, Red Hook, NY, USA: Curran Associates, 2020, pp. 2449–2460.
[58] D. Peter, W. Roth, and F. Pernkopf, "End-to-end keyword spotting using neural architecture search and quantization," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2022, pp. 3423–3427.
[59] D. Masters, A. Labatie, Z. Eaton-Rosen, and C. Luschi, "Making efficient-Net more efficient: Exploring batch-independent normalization, group convolutions and reduced resolution training," 2021, *arXiv:2106.03640*.

PEDRO H. PEREIRA received the B.Sc. degree in electronics engineering from the São Carlos School of Engineering, University of São Paulo, Brazil, in 2017, with a sandwich year at the University of Limerick, Ireland.

He was a Data Scientist with IBM from 2017 to 2022. He is currently an Associate Researcher with the Escola Politécnica, University of São Paulo, and a Lead Data Scientist with Dell Technologies. His research interests include machine learning, digital signal processing, and embedded systems.

WESLEY BECCARO received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from Escola Politécnica, University of São Paulo, in 2008, 2012, and 2017, respectively.

His research interests include digital signal processing, machine learning, instrumentation, embedded systems, and fuel qualification. He has authored or coauthored over 50 journals and conference articles in these areas.

MIGUEL A. RAMÍREZ (Life Senior Member, IEEE) received the B.S. degree in electronics engineering from Instituto Tecnológico de Aeronáutica, Brazil, in 1980, the degree in electronic design engineering from the Philips International Institute, The Netherlands, in 1981, and the M.S. and Ph.D. degrees in electrical engineering and the Habilitation degree in signal processing from the University of São Paulo, Brazil, in 1992, 1997, and 2006, respectively.

He was the Engineering Development Group Leader for Interactive Voice Response Systems (IVRs) with Itautec Informática, Brazil, from 1982 to 1990. In 2008, he was a Visiting Researcher in time-frequency speech analysis and coding with the Royal Institute of Technology, Sweden. He is currently an Associate Professor with Escola Politécnica, University of São Paulo, where he is also a member of the Signal Processing Laboratory. He has authored or coauthored four book chapters and over 70 journals and conference papers in these areas. His research interests include the applications of novel signal processing and machine learning algorithms to signal compression and prediction, speech analysis, coding and recognition, speaker identification, and audio analysis and coding. He is a member of the Brazilian Telecommunications Society (SBrT).