END-TO-END KEYWORD SPOTTING USING NEURAL ARCHITECTURE SEARCH AND QUANTIZATION

David Peter *Wolfgang Roth* *Franz Pernkopf*

Graz University of Technology
Signal Processing and Speech Communication Laboratory
Graz, Austria

ABSTRACT

This paper introduces neural architecture search (NAS) for the automatic discovery of end-to-end keyword spotting (KWS) models for limited resource environments. We employ a differentiable NAS approach to optimize the structure of convolutional neural networks (CNNs) operating on raw audio waveforms. After a suitable KWS model is found with NAS, we conduct quantization of weights and activations to reduce the memory footprint. We conduct extensive experiments on the Google speech commands dataset. In particular, we compare our end-to-end models to mel-frequency cepstral coefficient (MFCC) based CNNs. For quantization, we compare fixed bit-width quantization and trained bit-width quantization. Using NAS only, we were able to obtain a highly efficient model with an accuracy of 95.55% using 75.7k parameters and 13.6M operations. Using trained bit-width quantization, the same model achieves a test accuracy of 93.76% while using on average only 2.91 bits per activation and 2.51 bits per weight.

Index Terms**—** keyword spotting, neural architecture search, quantization

1. INTRODUCTION

Automatic speech recognition (ASR) is becoming increasingly important for user interaction with everyday consumer devices. ASR systems are typically complex and computation-intensive, i.e. running ASR in always-on mode results in a steady high energy consumption. This is especially problematic for mobile devices whose batteries are drained quickly when running ASR permanently.

A common solution is to run a low-cost keyword spotting (KWS) system that is listening permanently only for a limited set of prespecified keywords. Upon detection of such a keyword, a full ASR system is triggered which then listens for a rich set of user commands. The requirements of a KWS system are: (i) The system should be resource-efficient to mitigate the aforementioned energy problem, (ii) it should run in real-time and, (iii) it should be accurate to maintain a high user-experience.

In KWS, deep neural networks (DNNs) have become the state-of-the-art. In [1], several models from the literature [2, 3, 4, 5] are evaluated on the Google speech commands dataset [6], a popular data set consisting of pre-segmented fixed-length recordings of isolated words. In [1], models are compared in terms of accuracy, memory requirements and number of operations (i.e. the number of multiplications and additions) per forward pass. To allow for easy deployment on microcontrollers they train their models using 32 bit float numbers and quantize the weights after training to 8 bit fixed-point numbers. In [7], DNNs are compressed using model compression techniques to enable neural speech enhancement on microcontrollers.

There are many aspects to consider when designing resource-efficient DNNs for KWS. In this paper, we will focus specifically on the following important aspects: (i) The DNN architecture, (ii) comparison between spectral MFCC features and raw audio processing as well as (iii) quantization of weights and activations. We introduce relevant works of these areas in the following paragraphs.

Recently, neural architecture search (NAS) became a popular technique to automate the design of DNN architectures. A NAS algorithm searches for the best performing DNN architecture within a given search space using appropriate search heuristics. Popular NAS approaches use concepts such as reinforcement learning [8], gradient based methods [9] or evolutionary methods [10] for exploring the search space. In the context of resource-efficient DNNs, NAS techniques have also been used to find DNN architectures that are specifically tailored to the underlying hardware [11, 12, 13, 14] by, for instance, additionally minimizing memory requirements, number of operations, or latency of the resulting model. Therefore, NAS techniques are well-suited for finding DNNs that run on mobile phones, embedded devices and microcontrollers. In the context of KWS, NAS has been used successfully in [15, 16] to find efficient and small DNN architectures.

KWS with DNNs is typically performed on hand-crafted speech features such as MFCCs that are extracted from raw audio waveforms. Extracting MFCCs involves computing the Fourier transform. However, performing the Fourier transform is computationally expensive and might exceed the capabilities of resource-constrained devices. Therefore, Ibrahim et al. [17] proposed to use simpler speech features derived in the time-domain. The features, referred to as Multi-Frame Shifted Time Similarity (MFSTS), are obtained by computing constrained lag autocorrelations on overlapping speech frames to form a 2D map. A temporal convolutional neural network (TCN) [18] is then used to classify keywords on MFSTSs.

However, hand-crafted features such as MFSTSs and MFCCs may not be optimal for KWS. Therefore recent works have proposed to directly feed the DNNs with raw audio waveforms. In [19], a CNN for speaker recognition is proposed that encourages to learn parameterized sinc functions as kernels in the first layer. This layer is referred to as SincConv layer. During training, a low and high cutoff frequency per kernel is determined. Therefore, a custom filter bank is derived by training the SincConv layer that is specifically tailored to the desired application. SincConvs have also been recently applied to KWS [20].

Quantization-aware training uses the straight-through estimator (STE) [21, 22] to approximate the gradient of piecewise constant quantizers by the non-zero gradient of some other function. The

STE has been used for example in the training of binarized neural networks (BNNs) [23] where the resolution of both weights and activations is reduced to binary values $\{-1, 1\}$. Another method for quantizing DNNs involves a Bayesian approach to learn weight distributions over discrete weights [24]. Recently, the STE has been used to also learn the weight and activation bit-widths (i.e. number of bits) during training [25]. We will refer to this type of training as *trained bit-width quantization*.

In this paper, we propose NAS for the automatic discovery of small and resource-efficient end-to-end models for KWS. We utilize the techniques from ProxylessNAS [11] to find suitable models. During NAS, we establish a tradeoff between the model accuracy and the number of operations. We compare our end-to-end KWS models obtained by NAS to MFCC based CNNs. Once the efficient full-precision end-to-end KWS model has been found, we compare two weight and activation quantization methods. Both methods perform quantization-aware training from scratch on the full-precision end-to-end KWS model. In the first method, the weight and activation bit-widths are fixed during training. In the second method, trained bit-width quantization is performed.

Our work specifically focuses on the classification of pre-segmented fixed-length recordings of isolated words using the Google speech commands dataset. Our contributions are the following:

- We apply NAS to obtain efficient end-to-end KWS models operating on raw audio waveforms instead of hand crafted features. Recent works such as [16, 26, 15] rely on hand crafted features such as MFCCs when performing NAS for KWS.
- We perform a thorough comparison between KWS on raw audio waveforms and KWS on MFCCs in terms of accuracy, number of operations and number of model parameters.
- We compare two quantization methods for weight and activation quantization. In particular, we perform fixed bit-width and trained bit-width quantization on end-to-end KWS models to further reduce the memory footprint.

The outline of the paper is as follows: In Section 2 we present our NAS configuration, the feature extraction using SincConvs and the weight quantization methods utilized in this paper. The experimental setup is shown in Section 3. In Section 4, we discuss the results of our experiments. Finally, Section 5 provides the conclusion.

2. METHODS

2.1. Neural Architecture Search

We aim to find well performing architectures for different computing regimes. To achieve this, we use multi-objective ProxylessNAS [11] to discover DNNs optimized for accuracy and number of operations. Note that this implicitly optimizes for the model size as well.

ProxylessNAS constructs an overparameterized model with multiple parallel candidate operations per layer as the base model. The overparameterized model is trained together with a set of architecture parameters specifying probabilities over the candidate operations. Once training has finished, for each layer the most probable candidate operation is selected.

Table 1 shows the overparameterized model used in this paper. It consists of five stages with two input stages (i) (ii), two intermediate stages (iii) (iv) and one output stage (v). Stages (i), (ii) and (v) are fixed whereas stages (iii) and (iv) are optimized using NAS. We use mobile inverted bottleneck convolutions (MBCs) [27] as our main building blocks in stages (iii) and (iv).

Table 1. NAS model used for KWS. K denotes the kernel size, S the stride, C the number of channels and L the number of layers per stage. Stages (i) and (ii) and (v) are fixed. For stage (iii) and (iv), the parameters e (expansion rate), k (kernel size) and whether an identity layer is selected or not is optimized using NAS.

Stage	Type	K	S	C	L
(i)	SincConv	400	160	1	1
(ii)	Conv	3x3	2, 2	10	1
(iii)	MBC[e] / Identity	[k]$\times$[k]	2, 2	20	3
(iv)	MBC[e] / Identity	[k]$\times$[k]	2, 2	40	3
(v)	Conv	1×1	1, 1	80	1
	Global Avg. Pooling	-	-	-	1
	Fully connected	-	-	-	1

MBCs have two learnable parameters, the expansion rate e and the size k of the quadratic $k\times k$ convolution kernel. MBCs consist of three separate convolutions, one 1×1 convolution followed by a depthwise-separable 3×3 convolution followed again by a 1×1 convolution. The first two convolutions apply batch normalization and ReLU activation functions. The third convolution only applies batch normalization. The first and third 1×1 convolution change the number of feature maps by the expansion rate factor of e and $1/e$ respectively. Stride (as stated in Table 1) is only applied to the first convolution of each stage.

During NAS, we allow MBCs with expansion rates $e \in \{1, 2, 3, 4, 5, 6\}$ and kernel sizes $k \in \{3, 5, 7\}$ for selection. We also include the zero operation which effectively results in an identity layer [13]. For blocks where the input feature map size is equal to the output feature map size we include skip connections.

2.2. Feature Extraction using SincConvs

SincNet [19] uses parameterized sinc functions as filters in the first layer. This layer performs a convolution of the raw audio input x with an arbitrary number of parameterized sinc functions called SincNet filters. For a single SincNet filter, the output y given the filter $g[n, \theta]$ parameterized by θ is simply

$$y[n] = x[n] * g[n, \theta]. \tag{1}$$

In SincNet, the choice for the filter function g is

$$g[n, f_1, f_2] = 2f_2 \operatorname{sinc}(2\pi f_2 n) - 2f_1 \operatorname{sinc}(2\pi f_1 n), \tag{2}$$

where the sinc function is defined as $\operatorname{sinc}(x) = \sin(x)/x$ and $\theta = (f_1, f_2)$. This choice of g can be seen as a bandpass filter in the frequency domain with f_1 and f_2 being the low and high cut-off frequencies of the bandpass filter. The magnitude of the bandpass filter in the frequency domain is therefore

$$G[f, f_1, f_2] = \operatorname{rect}\left(\frac{f}{2f_2}\right) - \operatorname{rect}\left(\frac{f}{2f_1}\right) \tag{3}$$

where $\operatorname{rect}(f)$ is the rectangular function defined as

$$\operatorname{rect}(f) = \begin{cases} 1 & \text{if } |f| \leq \frac{1}{2} \\ 0 & \text{if } |f| > \frac{1}{2} \end{cases}. \tag{4}$$

SincConv filters have a much smaller memory footprint than 1D-Convs where the filter kernel is fully learnable. To derive a single filter $g[n, f_1, f_2]$, only two parameters, the lower cutoff frequency f_1 and the upper cutoff frequency f_2 are needed, whereas for convolutions with arbitrary filters, the number of parameters to store is equal to the length of the filter.

During runtime however, SincConvs and 1D-Convs of similar length need the same amount of memory to store the filter kernels. SincConv filter kernels are precomputed once before runtime. However, the computational cost of computing the SincConv filter kernels can typically be neglected.

2.3. Weight and Activation Quantization

For the architecture discovered by NAS, we compare two quantization methods for quantization of weights and activations, namely fixed bit-width quantization and trained bit-width quantization.

We perform quantization-aware training using the quantization framework Brevitas [28]. We quantize weights and activations of all layers, except for the input layer.

In quantization-aware training, quantized tensors are obtained from real-valued auxiliary tensors by applying a quantization function Q. During backpropagation, the gradients of the auxiliary tensors are obtained using the STE to estimate the gradient of Q. The quantized tensors are typically integer numbers encoded to k bits. A factor α, called the dynamic range [29, 30, 31], is used to map the integer numbers of the quantized tensor to real-valued numbers. The scaling factor typically increases the performance quite substantially.

For quantized weights, we select α to be the maximal absolute value of the auxiliary weight tensor. For quantized activations, we select α differently depending on the bit width. Binary (i.e. 1 bit) quantization of activations is performed using a constant scaling factor of $\alpha = 1$. When the activation bit-width is larger or equal to 2 bits, the scaling factor α is declared as a trainable parameter that is optimized using a gradient based approach.

For trained bit-width quantization we optimized the following loss function

$$L = L_{CE} + \lambda_w \cdot B_w + \lambda_a \cdot B_a \tag{5}$$

where L_{CE} is the cross-entropy loss, λ_w, λ_a are hyperparameters and B_w, B_a are the average weight and activation bit-width of the model respectively. For our experiments, the hyperparameters were selected based on a grid search with values 0, 0.02, 0.04, 0.06, 0.08 for both λ_w and λ_a. We selected $\lambda_w = 0.04$ and $\lambda_a = 0.04$ to obtain a good tradeoff between the model size and accuracy. Note that in Brevitas, trained bit-width quantization is limited to bit-widths larger or equal to 2. However, Brevitas is currently under active development and future versions may allow 1 bit activations and weights for trained bit-width quantization.

3. EXPERIMENTAL SETUP

3.1. Dataset

We use the first version of the Google speech commands dataset [6]. It consists of 65,000 1-second long audio files sampled with 16 bit at 16 kHz sampling frequency. Every audio file contains one utterance of an English word spoken by one person. The words are grouped into 30 different classes. We follow the procedure of [32] and use the following 10 classes ”Yes”, ”No”, ”Up”, ”Down”, ”Left”, ”Right”, ”On”, ”Off”, ”Stop” and ”Go” from the dataset. Likewise, we also include an ”unknown” class which is a blend of randomly selected samples from the remaining 20 classes. Furthermore, a ”silence” class is added. The ”silence” class is artificially generated and consists of 1-second audio files containing a random slice of audio from a randomly selected noise sample provided by the Google Speech commands dataset. We follow the procedure of [32] and perform data augmentation by applying a random time shift and adding background noise to the raw audio waveforms.

3.2. Feature Extraction

End-to-end KWS models presented in this paper do not need any hand-crafted feature extraction since they directly classify keywords from the raw audio waveforms. However, we will later compare end-to-end KWS models with models that use MFCCs as input. For models using MFCCs as input, the raw audio waveforms are first filtered with a low pass filter $f_{\text{low}} = 4\text{kHz}$ and a highpass filter $f_{\text{high}} = 40\text{Hz}$. We then extract between 10 and 30 MFCCs per 40ms frame with a stride length of 20ms. The number of MFCCs is varied in the experiments. We use a large frame size of 40ms and large stride of 20ms to keep the number of frames in the time dimension small. This in turn reduces the spatial size of the feature maps that are supplied to our models and therefore reduces the number of operations needed to compute a prediction. Before performing classification on raw audio waveforms, we select a window length of 25ms and a hop length of 10ms to split up the raw audio waveform into frames.

4. EXPERIMENTS

4.1. KWS from Raw Audio Waveforms using NAS

We compare our end-to-end KWS models to models using MFCCs [15]. MFCC models consist of three stages (c.f. Table 1 in [15]) where stage (i) is a 5×11 convolution and stage (iii) is a 1×1 convolution followed by global average pooling and a single fully connected layer. Stage (ii) consists of 12 MBC layers that are optimized using ProxylessNAS. A tradeoff between the model size and accuracy is established by varying the regularization parameter $\beta \in \{0, 1, 2, 4, 8, 16\}$ to obtain MFCC models of different sizes (for details on β see [15])

For end-to-end KWS models, we introduce a width multiplier m that scales the number of channels of the model by a factor of $m = 1, m = 0.5$ or $m = 2$. The width multiplier does not effect the SincConv layer at the input. For the SincConv layer, the number of filters was set to 40, 60, and 80. After a width-multiplier and the number of SincConv filters is selected, NAS is performed to select the layers in stage (iii) and (iv) of our NAS model (c.f. Table 1). Again, a tradeoff between the model size and accuracy is established by varying the regularization parameter $\beta \in \{0, 1, 2, 4, 8, 16\}$ to obtain end-to-end KWS models of different sizes. To assess the difference between SincConvs and 1D-Convs, we also performed NAS using models with 1D-Conv instead of SincConv. However, here we only select a width-multiplier of $m = 1$.

Figure 1 shows the performance of SincConv and MFCC models. For better visibility, we only include models on the Pareto frontier. We also include 1D-Conv models although none of the models contributes to the Pareto frontier. The number of operations of a model corresponds to the circle area. The number of parameters of SincConv models is the number of parameters needed to store the models before performing classification. During classification, the cutoff frequencies of the filters are used to compute the kernels of the SincConvs, which in turn slightly increases the number of pa-

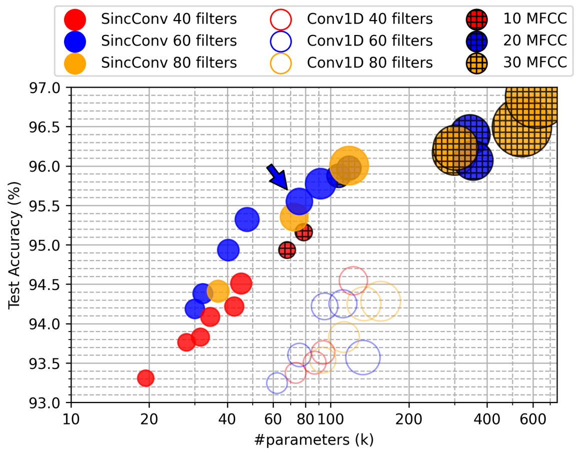

Fig. 1. Test accuracy versus number of parameters of KWS models obtained using NAS. The number of operations corresponds to the circle area. The model marked with an arrow is quantized in Section 4.2.

rameters. We can observe that 1D-Conv models perform worse than SincConv models with regard to test accuracy, number of operations and number of parameters. This indicates that using parameterized sinc functions instead of fully learnable filter kernels provide a substantial benefit in the performance of end-to-end KWS models. We can also observe that SincConv models need less parameters than MFCC models to achieve the same test accuracy. However, the number of operations is slightly larger in SincConv models.

Using NAS only, we were able to obtain efficient models. We highlighted one model from Figure 1 (marked with a blue arrow) that will be quantized in Section 4.2. This model achieves an accuracy of 95.55% using only 75.7k parameters and 13.6M operations.

4.2. Weight and Activation Quantization

After NAS is performed and a suitable full-precision end-to-end KWS model is found, we quantize the weights and activations to further reduce the memory footprint of the model. We select the model marked with a blue arrow from Figure 1. We first perform fixed bit-width quantization for the weights and activations. The results are visualized in Figure 2.

We observe that the most notable impact on performance is encountered when the activation bit-width is reduced to 2 or even 1 bit. When the activation bit-width is 4 bits, a slight performance impact is observed. However, when using 8 bit activations, the performance is similar to the full precision model irrespective of the weight bit-width.

We also perform trained bit-width quantization on the same model. However, now the weight and activation bit-width are optimized together with model parameters using backpropagation. The weight and activation bit-widths per layer after training are visualized in Figure 3. For MBC blocks, we report the average per-weight and per-activation bit width over the three convolutions.

The trained bit-width model visualized in Figure 3 needs on average 2.91 bits for quantized activations and 2.51 bits for quantized weights and achieves a test accuracy of 93.76%. Compared to a fixed bit-width model with 2 bit activations and 2 bit weights with a test accuracy of 90.35%, the trained bit-width model outperforms the fixed bit-width model by 3.41% while using only slightly more bits

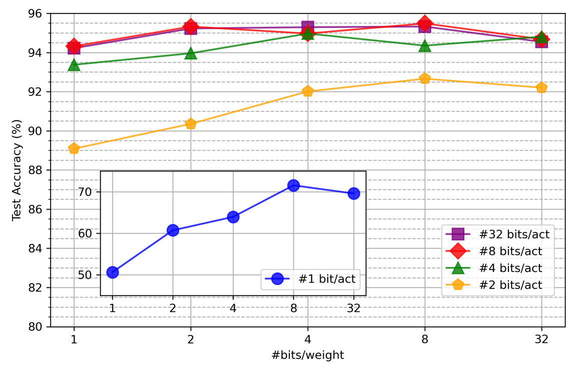

Fig. 2. Test accuracy versus weight bit-width versus activation bit-width of an end-to-end KWS model using SincConvs. The model was trained from scratch using quantization-aware training and fixed bit-widths.

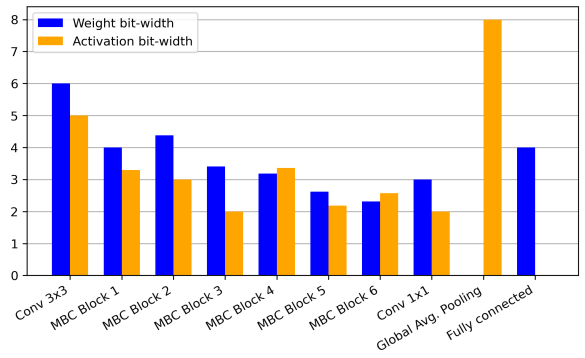

Fig. 3. Weight and activation bit-widths of an end-to-end KWS model using SincConvs. The model was trained from scratch using quantization-aware training and trained bit-widths.

on average. However, hardware implementation of trained bit-width quantization is more difficult.

We also observe that layers at the input and the output need more bits than intermediate layers. This result is in line with the vast literature where it is common practice to leave the input and output layers at full precision.

5. CONCLUSIONS

Resource-efficient DNNs are the key components in modern keyword spotting (KWS) systems. We used neural architecture search (NAS) to obtain efficient end-to-end convolutional neural networks (CNNs) for KWS. Our end-to-end KWS models utilize a SincConv at the input layer to perform classification on raw audio waveforms. To make our results comparable, we performed NAS on the Google speech commands dataset. We compared our end-to-end KWS models to mel-frequency cepstral coefficient (MFCC) based CNNs. We also compared two weight and activation quantization methods that help to further reduce the memory footprint. By establishing a trade-off between the model accuracy and the model size, we show that trained bit-width quantization can be used to obtain more competitive models than simply using fixed bit-width quantization.

6. REFERENCES

[1] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello Edge: Keyword Spotting on Microcontrollers," *CoRR*, vol. abs/1711.07128, 2017.

[2] G. Chen, C. Parada, and G. Heigold, "Small-footprint Keyword Spotting using Deep Neural Networks," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 4087–4091.

[3] T. N. Sainath and C. Parada, "Convolutional Neural Networks for Small-footprint Keyword Spotting," in *Annual Conference of the International Speech Communication Association (ISCA)*, 2015, pp. 1478–1482.

[4] S. Ö. Arik, M. Kliegl, R. Child, J. Hestness, A. Gibiansky, C. Fougner, R. Prenger, and A. Coates, "Convolutional Recurrent Neural Networks for Small-Footprint Keyword Spotting," in *Annual Conference of the International Speech Communication Association (ISCA)*, 2017, pp. 1606–1610.

[5] M. Sun, A. Raju, G. Tucker, S. Panchapagesan, G. Fu, A. Mandal, S. Matsoukas, N. Strom, and S. Vitaladevuni, "Max-pooling Loss Training of Long Short-term Memory Networks for Small-footprint Keyword Spotting," in *Spoken Language Technology Workshop (SLT)*, 2016, pp. 474–480.

[6] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *CoRR*, vol. abs/1804.03209, 2018.

[7] I. Fedorov, M. Stamenovic, C. Jensen, L. Yang, A. Mandell, Y. Gan, M. Mattina, and P. N. Whatmough, "TinyLSTMs: Efficient Neural Speech Enhancement for Hearing Aids," in *Interspeech (IS)*, 2020, pp. 4054–4058.

[8] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient Neural Architecture Search via Parameter Sharing," in *International Conference on Machine Learning (ICML)*, 2018, pp. 4092–4101.

[9] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," in *International Conference on Learning Representations (ICLR)*, 2019.

[10] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical Representations for Efficient Architecture Search," in *International Conference on Learning Representations (ICLR)*, 2018.

[11] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware," in *International Conference on Learning Representations (ICLR)*, 2019.

[12] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2820–2828.

[13] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, "SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[14] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers," in *Proceedings of Machine Learning and Systems (MLSys)*, 2021, pp. 517–532.

[15] D. Peter, W. Roth, and F. Pernkopf, "Resource-efficient DNNs for Keyword Spotting using Neural Architecture Search and Quantization," *International Conference on Pattern Recognition (ICPR)*, 2020.

[16] T. Mo, Y. Yu, M. Salameh, D. Niu, and S. Jui, "Neural Architecture Search for Keyword Spotting," in *Interspeech (IS)*, 2020, pp. 1982–1986.

[17] E. A. Ibrahim, J. Huisken, H. Fatemi, and J. P. de Gyvez, "Keyword Spotting using Time-Domain Features in a Temporal Convolutional Network," in *22nd Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 313–319.

[18] C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, "State-of-the-Art Speech Recognition with Sequence-to-Sequence Models," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 4774–4778.

[19] M. Ravanelli and Y. Bengio, "Speaker Recognition from Raw Waveform with SincNet," in *IEEE Spoken Language Technology Workshop (SLT)*, 2018, pp. 1021–1028.

[20] S. Mittermaier, L. Kürzinger, B. Waschneck, and G. Rigoll, "Small-Footprint Keyword Spotting on Raw Audio Data with Sinc-Convolutions," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 7454–7458.

[21] G. Hinton, "Neural Networks for Machine Learning," *Coursera, video lectures*, 2012.

[22] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," *CoRR*, vol. abs/1308.3432, 2013.

[23] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 4107–4115.

[24] W. Roth, G. Schindler, H. Fröning, and F. Pernkopf, "Training Discrete-Valued Neural Networks with Sign Activations Using Weight Distributions," in *European Conference on Machine Learning (ECML)*, 2019, pp. 382–398.

[25] S. Uhlich, L. Mauch, F. Cardinaux, K. Yoshiyama, J. A. García, S. Tiedemann, T. Kemp, and A. Nakamura, "Mixed Precision DNNs: All you need is a good parametrization," in *International Conference on Learning Representations (ICLR)*, 2020.

[26] B. Zhang, W. Li, Q. Li, W. Zhuang, X. Chu, and Y. Wang, "Autokws: Keyword spotting with differentiable architecture search," *CoRR*, vol. abs/2009.03658, 2020.

[27] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[28] A. Pappalardo, "Xilinx/brevitas," *URL: https://github.com/Xilinx/brevitas*.

[29] S. Uhlich, L. Mauch, K. Yoshiyama, F. Cardinaux, J. A. García, S. Tiedemann, T. Kemp, and A. Nakamura, "Differentiable Quantization of Deep Neural Networks," *CoRR*, vol. abs/1905.11452, 2019.

[30] S. R. Jain, A. Gural, M. Wu, and C. Dick, "Trained Quantization Thresholds for Accurate and Efficient Fixed-Point Inference of Deep Neural Networks," in *Proceedings of Machine Learning and Systems (MLSys)*, 2020.

[31] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned Step Size quantization," in *International Conference on Learning Representations (ICLR)*, 2020.

[32] R. Tang and J. Lin, "Honk: A PyTorch Reimplementation of Convolutional Neural Networks for Keyword Spotting," *CoRR*, vol. abs/1710.06554, 2017.