# Low-bit quantization and quantization-aware training for small-footprint keyword spotting

Yuriy Mishchenko
*Amazon.com*
Cambridge, USA
yuriym@amazon.com

Yusuf Goren
*Amazon.com*
Cambridge, USA
yggoren@amazon.com

Ming Sun
*Amazon.com*
Cambridge, USA
mingsun@amazon.com

Chris Beauchene
*Amazon.com*
Cambridge, USA
beauche@amazon.com

Spyros Matsoukas
*Amazon.com*
Cambridge, USA
matsouka@amazon.com

Oleg Rybakov
*Amazon.com*
Seattle, USA
rybakovo@amazon.com

Shiv Naga Prasad Vitaladevuni
*Amazon.com*
Cambridge, USA
shivnaga@amazon.com

***Abstract*—In this paper, we investigate novel quantization approaches to reduce memory and computational footprint of deep neural network (DNN) based keyword spotters (KWS). We propose a new method for KWS offline and online quantization, which we call dynamic quantization, where we quantize DNN weight matrices column-wise, using each column's exact individual min-max range, and the DNN layers' inputs and outputs are quantized for every input audio frame individually, using the exact min-max range of each input and output vector. We further apply a new quantization-aware training approach that allows us to incorporate quantization errors into KWS model during training. Together, these approaches allow us to significantly improve the performance of KWS in 4-bit and 8-bit quantized precision, achieving the end-to-end accuracy close to that of full precision models while reducing the models' on-device memory footprint by up to 80%.**

***Index Terms*—keyword spotting, quantization-aware training, small-footprint**

## I. Introduction

Keyword spotting is the task of detecting a keyword of interest in a continous audio stream. Keyword spotting has become an active area of research recently in view of its importance for modern voice assistant systems. Voice assistant-enabled devices, such as Amazon Echo or Google Home, only start streaming audio to the cloud when their keyword spotter (KWS) detects correct keyword. KWS systems are expected to provide high recall and extremely low false alarm rate, while continuously running on device under stringent CPU and memory constraints. For those reasons, development of accurate and small-footprint KWS has attracted much attention in the community recently [1], [2], [3], [4].

One type of small-footprint KWS used in the industry are based on single-stage feed-forward DNN and convolutional neural networks (CNN) [1], [5], [6]. In such KWS, the keyword posterior calculated by the DNN or CNN is smoothed with a sliding window, and the keyword event is triggered whenever the smoothed posterior exceeds a threshold. The tradeoff between false rejects and false accepts is performed by tuning the threshold. Typically trained in full precision, such KWS are always deployed on-device in a ”quantized” form. By quantization here we refer to the process of converting the weights of the full-precision DNN or CNN to an integer number format and also decoding the model at run-time by using only integer arithmetic.

A number of methods for quantizing DNN models have been pursued in research literature lately [7], [8], [9], [10], [11], [12], [13], [14], [15]. In the industry, a common approach for KWS quantization is post-training quantization, where the weights of the DNN layers are quantized at matrix-level, i.e. by mapping the floating-point range of the weights in a layer to a suitable integer interval based on the minimum and maximum weights of the entire layer, while the activation values at run-time are quantized based on fixed floating point ranges such as $0..1$, $-10..10$, etc.

In this paper, we propose 2 improvements that, while being a simple change to implement on top of the above industry standard, allow for significantly reducing the accuracy degradation of quantized KWS up to the point where 8- and 4-bit quantized KWS can perform similar to full-precision models. The first improvement is a different post-hoc quantization method, where the weights of DNN layers are quantized column-wise instead of matrix-wise. That is, each column of DNN layer is quantized to the column's exact min-max range. This is similar to channel-wise quantization in the literature [16] except that, instead of separately quantizing different ”channels” in the input, such as colors in an image, or CNN convolution channels, we quantize individually the columns of the DNN layers' weight matrices. Furthermore, the activation values at run-time are quantized separately for each forward-propagated input, based on the true min and max values of each activation vector during run-time. We call this approach *dynamic quantization* to distinguish it from the common approach of ”statically” quantizing the weight matrices to a single min-max range and activation vectors to predetermined ranges. Secondly, we implement a quantization-aware training approach for KWS, where quantization errors are injected into training by performing the forward-propagation pass using explicitly the quantized implementation at run-time. This allows us to further reduce the performance degradation after quantization up to a point, where we find that 4- and 8-bit quantized KWS can be trained successfully with minor loss of performance relative to full precision.

The remainder of the paper is organized as follows: Section II introduces our keyword spotting system and quantization-aware training approach. Experiments follow in Section III. Conclusions follow in Section IV.

## II. Keyword Spotting System and Quantization-Aware Training

The KWS system considered in this paper is a DNN with 50k or 250k parameters with linear-bottleneck architecture described in [17]. Our choice of feed-forward DNN comes from past effectiveness of this architecture for low-power keyword spotting, and it has been well optimized and serves as strong baseline [17]. We choose two DNN sizes to validate generic nature of our approach. Briefly, the DNN operates on 20-dimensional log mel filter-bank energy (LFBE) features calculated over 25ms frames with a 10ms frame shift, stacked in 620-dimensional input window with 20 frames left and 10 frames right context. The DNN has 6 hidden layers with dimensions of 39 and 128 (50k) or 87 and 400 (250k) units, as shown in Figure 1. The narrower layers are followed by linear activation function and the wider layers are followed by sigmoid activation. We call the pair of such layers SVD-bottleneck. Those bottlenecks play the role of SVD approximation of fully-connected layers in DNN [17], [18], [19], [20]. The output of the DNN is a softmax layer with 2 outputs representing the posterior distribution over input frames in {'keyword', 'non-keyword'}. The DNN is trained using cross-entropy loss.

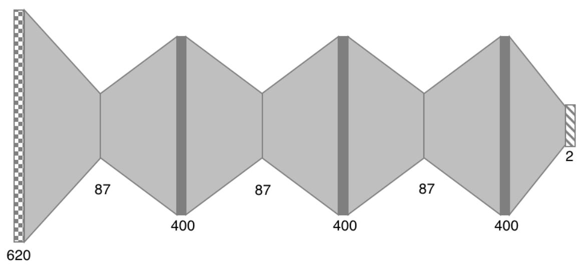

Fig. 1: The DNN architecture used in this work, illustrated using 250k DNN KWS. The DNN consists of 3 SVD-bottleneck pairs (light gray) comprised of layers of sizes 87 and 400 units, as indicated. The SVD-bottlenecks are sequences of 2 affine transforms with linear activation between them, followed by sigmoid activation (dark gray). The output is a softmax layer with 2 targets (diagonal stripes).

For KWS model at run-time, we use dynamic quantization (DQ) of the model's layers and all activation vectors. DQ uses column-wise shifts and scales for quantizing the DNN weight matrices. This allows for a more accurate quantization of the DNN than the single, matrix-level shift and scale-quantization used in existing deep learning software packages. More specifically, let $W^k = (w_{ij}^k)$ be the weight matrix of $k^{th}$ layer of the DNN, and let $\mathfrak{b}$ be the bit-precision we quantize to. We introduce column-wise quantization shifts and scales

$$\begin{aligned} \sigma_j^k &:= \frac{\max\limits_i w_{ij}^k - \min\limits_i w_{ij}^k}{2^{\mathfrak{b}} - 1} \\ \alpha_j^k &:= \max_i w_{ij}^k - (2^{b-1} - 1)\sigma_j^k \end{aligned} \tag{1}$$

and column-wise quantization transform

$$\widetilde{w}_{ij}^k = \text{round}\left(\frac{w_{ij}^k - \alpha_j^k}{\sigma_j^k}\right)\sigma_j^k + \alpha_j^k \tag{2}$$

where the column-wise shifted and rescalled weights of the original matrix satisfy

$$\frac{w_{ij}^k - \alpha_j^k}{\sigma_j^k} \in [-2^{\mathfrak{b}-1}, 2^{\mathfrak{b}-1}) \tag{3}$$

round$(\cdot)$ denotes the standard integer rounding function. The reasoning behind column-wise quantization is that the form of matrix-vector multiplication during affine transform in forward propagation allows using separate scales for each column-row multiplication. At the same time, when quantization range, or equivalently - the scale, is chosen on per-column basis using the min and the max values of each column of the affine transform matrix independently, such range can be chosen more tightly, therefore allowing for better representation of the original floating point weights in quantized form. When the quantization scale for a weight matrix is chosen on the matrix-wide max and min weights, many columns may be suboptimally quantized due to quantization range or scale for them being unnecessarily large (i.e. define by the entire layer's min and max weight). Note that it is also possible to use stochastic rounding in Eq.(2), such as described in [21]. Thus, all weight matrices are quantized using the above prescription and stored on-device using $\mathfrak{b}$-bit integer form identified by Eq.(3).

The input and output vectors of all layers are furthermore quantized at run-time on per input frame-basis. That is, those vectors are quantized using the DQ approach where the shifts and scales are chosen individually for each input/output vector during each forward propagation pass. The use of DQ both for DNN weights and inputs/outputs quantization allows us to significantly improve accuracy of the quantized KWS at run-time over the "static" quantization, while requiring minimal algorithmic changes. Also, DQ does not require prior knowledge about the distribution of input data or intermediate activation values, such as otherwise needed for choosing the quantization scales of layers' inputs and outputs (cf. TensorRT implementation [22]).

On the software implementation side, we use hardware-specific SIMD operations to accelerate and parallelize quantized multiplications, whereas both the quantization transforms and multiplications can be expressed in terms of float scaling and integer dot-products. This quantization scheme is computationally efficient and does not suffer from packing and unpacking due to bucketing [23], [16]. In contrast with other low-bit quantization methods [9], [10], [13], we do not need weight sampling at quantization level and therefore the resulting representation is closer to the original algebraically. This leads to the computation of matrix multiplications requiring less auxiliary operations. Our software implementation of DQ forward propagation additionally computes the sums of columns in floating precision, further optimizing the number of element-wise multiplications necessary for run-time decoding. More precisely, given the quantized input and weights values, the cross terms of the quantized approximation of the product $X^T \cdot W$ can be expressed as the sum of products of quantized integer values from $X$ and $W$ and the column-wise sums over the original $X$ and $W$. Therefore, storing those sums directly allows us to not only speed up the product calculation but also makes it more precise, since the sums can be computed in floating point.

Further improvements of quantized KWS accuracy can be achieved by using quantization-aware training (QAT) [20], [24], [25]. We implement a QAT approach where we inject quantization errors into DNN components $C$ during training by using quantized forward propagation for all training inputs, as shown in Figure 2. This allows the quantization errors to be transfered to the loss function $\mathcal{L}$ during back-propagation, which is evaluated in full precision but using the quantized forward activation values $\mathbb{Q}[C]$. QAT is used during final several training epochs to allow the model to tune its performance with respect to the quantization errors, in order to improve parity with full precision. A salient point of our approach is that the forward propagation during training is performed by using the exact version of the quantized model and arithmetic as exists on device during run-time. This is different, e.g., from [8], where the model is considered as floating point during the forward pass. Therefore, in our approach, weights quantization and inputs/outputs quantization are performed to match the exact quantized behavior on device, independent of the batching. Our algorithm does not use stochastic perturbation [7].

## III. Experimental Results

The keyword 'Alexa' is chosen for our experimentation. We use an in-house 500 hrs corpus of diverse far-field speech data, split in proportion 99:1 for training and dev. Given this amount of data, 1% dev dataset serves well for validation purposes. We do not use regularization considering that the amount of data is large relative to the model size here. We use a similar far-field speech 100 hrs dataset for evaluation of final KWS models. We evaluate the models using end-to-end Detection Error Tradeoff (DET) curves, which describe the models' miss rate vs. false alarm rate (FAR). Miss rate is the fraction of keywords missed by KWS during evaluation. False alarm rate is the rate of false keywords detected per a given number of true keywords. We also calculate the Area Under Curve (AUC) for the DET curves. In this paper, absolute numbers of false alarm and miss rates have been obscured due to confidentiality.

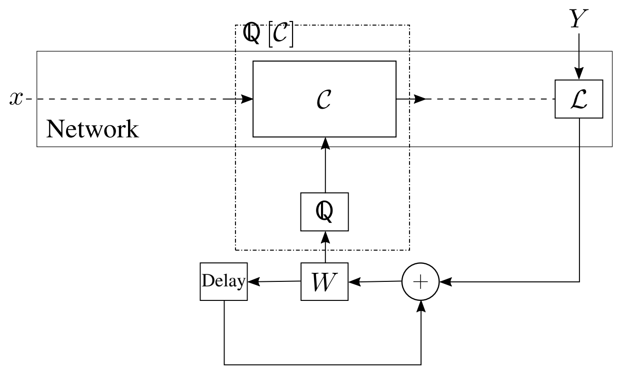

Fig. 2: Quantization-aware training used in this work. The forward propagation during training using examples $x$ and labels $Y$ is evaluated using the quantized DNN components $\mathbb{Q}[C]$, i.e. by using dynamically quantized weights $\mathbb{Q}[W]$ and all inputs and outputs as well as integer CPU or GPU instructions. This allows the quantization errors to be transferred to the loss function $\mathcal{L}$ and ultimately to the weights $W$ during back-propagation performed in floating point.

Instead, we report those metrics up to a multiplicative constant. The false alarm range considered in our experiments is aligned with a low value range that can be considered for production deployment purposes.

We train the models using GPU-based distributed training described in [26]. Our training recipe leverages transfer learning using ASR initialization and multi-task learning based on using keyword and ASR training targets simultaneously [27]. Together with long training runs, those allow producing strong baseline KWS models.

The training is organized into sequence of 3 steps:

1) We use transfer learning to train a small ASR DNN with 3 hidden layers of 128 (for 50k KWS) or 400 (for 250k KWS) units. This DNN is randomly initialized and trained using the full set of ASR triphone-targets generated by a large, production-grade ASR system from our dataset. The model is trained for 12 epochs using the exponential learning rate decay schedule below.
2) We use multi-task learning to train the KWS DNN using the ASR DNN from step 1. We add a fully-connected keyword branch to the ASR DNN and train that 2-branch model for 20 epochs with the keyword and the ASR targets together.
3) We introduce SVD-bottlenecks into the fully-connected DNN layers in order to reduce the model's size by about 50%, and train the final model with the SVD-bottlenecks using the multi-task learning from step 2 for 20 more epochs.

In the end of the training, the keyword targets of the DNN are used for the KWS model while the ASR branch is discarded. In all steps, we use exponential learning rate decay with initial learning rate of 0.000125 and decay factor of 2 for the first 5 epochs and 1.2 for all remaining epochs.

We investigate 16-bit, 8-bit, hybrid 4-8 bit, and 4-bit quantizations for quantized on-device KWS models. For 16-bit, 8-bit, and 4-bit quantization all layers are quantized using indicated bit-width. For 4-8 bit quantization, the first 2 layers comprising the first SVD-bottleneck in Figure 1 and the 2nd layers of all subsequent bottlenecks are quantized as 8 bits, while other layers are quantized as 4 bits. The logic behind this strategy is that the first hidden and all 2nd bottleneck-layers receive non-squeezed inputs with potentially large dynamic range. At the same time, the layers following sigmoid activation units have their inputs restricted to $[0, 1]$ and may allow for lower-bit quantization. We test DQ and QAT in those models and compare results with the baseline comprising 16- and 8-bit KWS models quantized statically post-training, as is common in the industry [20], [28].

TABLE I: Relative AUC and model sizes after quantization for the 50k DNN KWS. Lower AUC values indicate better KWS performance. Full precision model AUC is normalized to 1.0, and AUC for other quantization schemes are normalized against full precision model's AUC. Model sizes are normalized against the full precision model taken to have size 1.0.

| | **16 bit** | **8 bit** | **4-8 bit** | **4 bit** |
|---|---|---|---|---|
| Baseline | 1.000 | 1.093 | - | - |
| DQ | 1.000 | 1.009 | 1.168 | 2.216 |
| Size | 0.65 | 0.35 | 0.32 | 0.20 |

TABLE II: AUC improvements of quantized 50k KWS performance with QAT. Lower AUC indicates better KWS performance.

| | **4-8 Bit** | **4 Bit** |
|---|---|---|
| **DQ** | 1.168 | 2.216 |
| **DQ + QAT** | 1.021 | 1.410 |
| Degradation recovered | 87.4% | 66.3% |

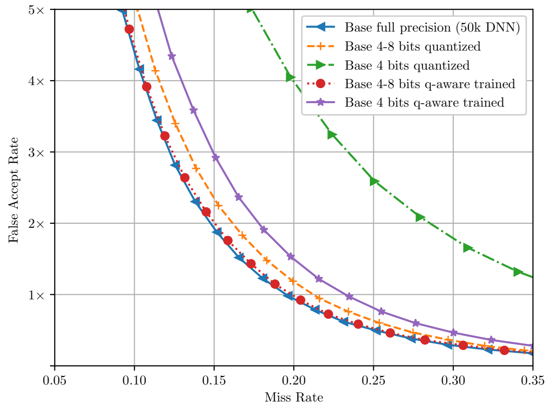

Fig. 3: DET for full-precision, DQ, and DQ+QAT 50k DNN KWS model. The DET curves for 16-bit and 8-bit quantized-models are not shown due to them not being significantly different from the full-precision model.

The results of our experimentation with the 50k KWS are shown in Table I. In the case of 16-bit quantization, both the baseline and the DQ models show practically no differences relative to the full precision model. However, with 8 bits quantization we observe 1% performance degradation in terms of the model's AUC when using DQ vs. about 10% AUC degradation if the model was quantized using baseline approach. KWS models with 4-8 bit hybrid and 4-bit quantization are worse with 16.8% and 121.6% AUC increase, respectively. However, using the baseline quantization in those regimes we find that those quantized models saturate at 0 or 1 output, for which reason we do not show AUC for baseline quantization in those regimes in Tables I and III. Therefore, DQ allows us to decrease the quantization-related performance loss by 90% relative to the baseline quantization approach. We also obtain functional, albeit worse, KWS models in 4-bit precision, where we could not do so with the baseline quantization.

We are able to further improve the performance of quantized models by using our QAT approach. Those results are shown in Table II. We do not pursue QAT for 16- and 8-bit quantization considering that DQ essentially recovered the full-precision models' performance there to within 1%. In the case of 4-8 bit quantization, our QAT approach allows us to recover close to 90% of the lost accuracy. The resulting model has only 2% worse AUC than the full-precision model, at 70% smaller memory footprint. In the case of 4-bit quantized model, we recover close to 66% of AUC with QAT, whereas the final model has 40% worse AUC relative to the full-precision model at 80% smaller size.

Our experimentation with larger, 250k, KWS model shows essentially similar results. Those are shown in Tables III and IV, and Figure 4. The impact of QAT is greater in this case, likely because of larger model size. With DQ and QAT, we are able to completely remove the performance degradation in 4-8 bit quantized model, achieving the same performance with the full-precision model at 70% smaller size. For 4-bit quantization, we achieve reasonable 19% AUC degradation with 82% model size reduction. We observe slightly better performance in DQ + QAT model than the floating-point model, which may be due to regularizing effect of quantization on training.

## IV. Conclusions

We present results of our work where we propose a new dynamic quantization and quantization-aware training approach for reducing quantization-related errors, especially of interest for ultra-low power keyword spotting applications. Our method significantly reduces the model size and possibly computational cost of KWS while maintaining performance on par with full-precision models. Our results indicate that using our DQ and QAT approaches, 4-8 bit hybrid quantized KWS models can be built maintaining the performance of full-precision models while reducing the memory footprint of the models by 70% compared to the full-precision models. For 4-bit quantized KWS, our approach significantly reduces the performance gap while providing up to 80% model memory footprint reduction.

TABLE III: Relative AUC and model sizes after quantization for the 250k DNN KWS. Notation is as in Table I.

| | **16 bit** | **8 bit** | **4-8 bit** | **4 bit** |
|---|---|---|---|---|
| Baseline | 1.000 | 1.079 | - | - |
| DQ | 0.999 | 1.013 | 1.040 | 2.630 |
| Size | 0.55 | 0.34 | 0.30 | 0.18 |

TABLE IV: AUC improvements of quantized 250k KWS performance with QAT. Lower AUC indicates better KWS performance.

| | **4-8 Bit** | **4 Bit** |
|---|---|---|
| **DQ** | 1.0404 | 2.6298 |
| **DQ + QAT** | 0.9509 | 1.1910 |
| Degradation recovered | 221.5% | 88.3% |

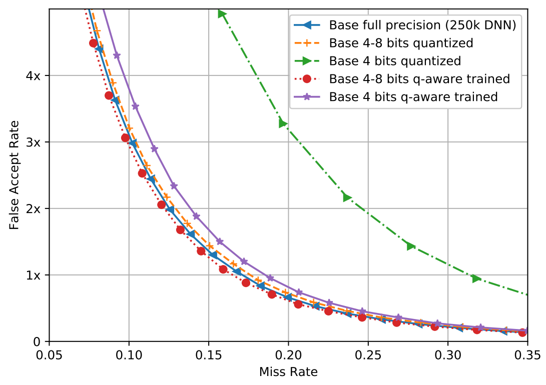

Fig. 4: DET for full-precision, DQ, and DQ+QAT 250k model. The DET curves for 16-bit and 8-bit quantized-models are not shown due to them not being significantly different from the full-precision model.

Investigated quantization schemes provide both computational cost and storage benefit. Quantization to 8-bit and 4-bit weights allows reducing model storage size on device by 65% to 80%. 8-bit, 4-8 bit, and 4-bit quantization allows also for computational cost reduction and latency improvements of KWS. For example, 4-bit integer multiplications can be performed in principle 4x faster than 8-bit integer multiplication, and 8-bit int multiplication can be performed up to 16x faster than floating point. However, such latency reduction numbers strongly depend on hardware, in particular presence of math accelerators such as floating point co-processors (typical in modern CPU) as well as hardware-specific optimizations such as SIMD instructions for parallelizing 8-bit integer multiplications using word-registers. Taking advantage of 4-bit computational cost reductions requires hardware that can support similar SIMD parallelization of 4-bit integer multiplications, such as specialized DNN accelerators with support for 4-bit arithmetic. Due to those not available to us at this time and other hardware and software factors influencing actual computational speedup in quantized models, it is difficult to talk substantively about specific numbers in that regard. Therefore, we choose here to not discuss such numbers and focus on the memory footprint reduction instead.

We introduce several novel techniques in this work. Our DQ algorithm uses column-wise DNN weight matrix quantization, where each column is quantized with its own shift and scale determined based on the range of the values in that column. Column-wise DNN weights quantization is different from what has been used in the literature and in particular from DNN quantization implemented in most common deep learning packages such as TensorRT. Our DQ algorithm also quantizes inputs and outputs of each DNN layer to their true min-max ranges during run-time. While relatively simple change on top of existing quantization implementations, DQ allows us to reduce the quantization-related performance degradation of quantized DNN KWS by up to 90% for 8-bit post-training quantized KWS. Our QAT algorithm uses DQ to quantize the DNN and its inputs and outputs during forward pass during training in the manner matching the behavior on device, and thus inject quantization errors into model training. This allows us to achieve further significant accuracy gains with 4-bit quantized KWS, allowing hybrid 4-8 bit quantized models with performance matching full-precision models and fully 4-bit quantized models with moderate performance loss.

We introduce novel quantized DNN KWS architecture in this work, where different components of DNN are quantized to different bit-widths to take into account different dynamic ranges encountered by different components at run-time. Typically, in the literature DNN quantization is performed using the same bit-width for all DNN components. Our mixed 4-8 bit quantized architecture emerges superior in our experiments to both fully 8-bit and 4-bit quantized models, by allowing quantized models of size smaller than with the standard 8-bit quantization while retaining performance of full-precision models. For fully 4-bit quantized models, we reduce performance gap down to 20-40% of the full-precision models with the memory footprint of 20% relative to the full-precision models.

## References

[1] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4087–4091.

[2] M. Sun, V. Nagaraja, B. Hoffmeister, and S. Vitaladevuni, "Model shrinking for embedded keyword spotting." in *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*. IEEE, 2015, pp. 369–374.

[3] Q. He, G. W. Wornell, and W. Ma, "An adaptive multi-band system for low power voice command recognition," in *Proc. Interspeech 2016*, 2016, pp. 1888–1892.

[4] 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 IEEE*. IEEE, 2016, pp. 474–480.

[5] P. Nakkiran, R. Alvarez, R. Prabhavalkar, and C. Parada, "Compressing deep neural networks using a rank-constrained topology." in *Proc. Interspeech 2015*, 2015, pp. 1473–1477.

[6] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting." in *Proc. Interspeech 2015*, 2015, pp. 1478–1482.

[7] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," *arXiv preprint arXiv:1802.04680*, 2018.

[8] D. Lee and B. Kim, "Retraining-based iterative weight quantization for deep neural networks," *arXiv preprint arXiv:1805.11233*, 2018.

[9] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[10] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[11] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[12] L. Hou, Q. Yao, and J. T. Kwok, "Loss-aware binarization of deep networks," *arXiv preprint arXiv:1611.01600*, 2016.

[13] C. Leng, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with ADMM," *arXiv preprint arXiv:1707.09870*, 2017.

[14] R. Alvarez, R. Prabhavalkar, and A. Bakhtin, "On the efficient representation and execution of deep acoustic models," in *Proc. Interspeech 2016*, 2016, pp. 2746–2750.

[15] R. Takeda, N. Kanda, and N. Nukaga, "Boundary contraction training for acoustic models based on discrete deep neural networks," in *Proc. Interspeech 2014*, 2014, pp. 1063–1067.

[16] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[17] M. Sun, D. Snyder, Y. Gao, V. Nagaraja, M. Rodehorst, S. Panchapagesan, N. Ström, S. Matsoukas, and S. Vitaladevuni, "Compressed time delay neural network for small-footprint keyword spotting," in *Proc. Interspeech 2017*, 2017, pp. 3607–3611.

[18] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6655–6659.

[19] A. Senior and X. Lei, "Fine context, low-rank, softplus deep neural networks for mobile speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 7644–7648.

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

[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015, pp. 1737–1746.

[22] S. Migacz, "8-bit inference with TensorRT," http://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf, 2017.

[23] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *arXiv preprint arXiv:1802.05668*, 2018.

[24] B. Shi, M. Sun, C.-C. Kao, V. Rozgic, S. Matsoukas, and C. Wang, "Compression of acoustic event detection models with low-rank matrix factorization and quantization training," in *NeurIPS 2018 workshop on Compact Deep Neural Networks with industrial applications*, 2018.

[25] ——, "Compression of acoustic event detection models with quantized distillation," in *Proc. Interspeech 2019*, 2019, pp. 3639–3643.

[26] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[27] S. Panchapagesan, M. Sun, A. Khare, S. Matsoukas, A. Mandal, B. Hoffmeister, and S. Vitaladevuni, "Multi-task learning and weighted cross-entropy for DNN-based keyword spotting," in *Proc. Interspeech 2016*, 2016, pp. 760–764.

[28] A. Gruenstein, R. Alvarez, C. Thornton, and M. Ghodrat, "A cascade architecture for keyword spotting on mobile devices," *arXiv preprint arXiv:1712.03603*, 2018.