COMPRESSING DEEP NEURAL NETWORKS FOR EFFICIENT SPEECH ENHANCEMENT

Ke Tan[1] *and DeLiang Wang*[1,2]

[1]Department of Computer Science and Engineering, The Ohio State University, USA
[2]Center for Cognitive and Brain Sciences, The Ohio State University, USA
{tan.650, wang.77}@osu.edu

ABSTRACT

The use of deep neural networks (DNNs) has dramatically improved the performance of speech enhancement in the past decade. However, a large DNN is typically required to achieve strong enhancement performance, and this kind of model is both computationally intensive and memory consuming. Hence it is difficult to deploy such DNNs on devices with limited hardware resources or in applications with strict latency requirements. In order to address this problem, we propose a model compression pipeline to reduce DNN size for speech enhancement, which is based on three kinds of techniques: sparse regularization, iterative pruning and clustering-based quantization. Evaluation results show that our approach substantially reduces the sizes of different DNNs without significantly affecting their enhancement performance. Moreover, we find that training and compressing a large DNN yields higher STOI and PESQ than directly training a small DNN that has a comparable size to the compressed DNN. This further suggests the benefits of using the proposed model compression approach.

Index Terms— model compression, sparse regularization, pruning, quantization, speech enhancement

1. INTRODUCTION

Speech enhancement is the task of separating target speech from background noise. Since it is formulated as supervised learning [1], many data-driven algorithms have been developed over the past decade, in which discriminative patterns within signals are learned from training data. In particular, the enhancement performance has been substantially advanced due to the use of deep learning [2, 3]. To achieve satisfactory enhancement performance would require a large DNN, which is both time and memory consuming. Thus it is difficult to deploy such DNNs in latency-sensitive applications or on resource-limited devices (e.g. headphones). As a result, reducing the cost of memory and computation in DNNs has become an increasingly important problem.

Various model compression techniques have been developed in recent years [4]. These techniques can be broadly categorized into two classes. The first class aims to reduce the number of trainable parameters. A widely-used technique is network pruning, which selects and removes the least important weights with a certain criterion [5]. It dates back to optimal brain damage [6], which utilizes the Taylor expansion to estimate the importance of each weight (i.e. weight saliency). The weights with the smallest saliency are pruned, and the remaining weights are fine-tuned to maintain the original accuracy. Another effective technique is tensor decomposition, which decomposes a large weight tensor into multiple smaller tensors by leveraging the low-rankness of the weight tensor. Furthermore, knowledge distillation was first developed by Hinton *et al.* [7] to guide the training of a relatively small DNN using soft targets produced by a pretrained large DNN, which has proven to be effective in classification tasks such as image classification [8] and speech recognition [9, 10]. Other related studies reduce network redundancy by designing more parameter-efficient architectures [11, 12, 13]. The second class of model compression techniques is network quantization, which reduces the number of bits representing each weight. A common method is to train DNNs with full precision and then directly quantize the resulting weights. This method tends to significantly degrade the accuracy for relatively small DNNs [14, 15]. In contrast, a more robust method, known as quantization-aware training, incorporates simulated quantization effects, which compensate for the loss of accuracy [14]. Moreover, one can perform quantization by applying k-means clustering to the learned weights [16, 17].

For DNN-based speech enhancement, there is increasing interest in reducing network redundancy and accelerating network inference. In [18], a tensor-train long short-term memory (LSTM) was developed for monaural speech enhancement, in which tensor-train decomposition [19] is applied to weight matrices. An integer-adder DNN was designed in [20], where floating-point multiplication is implemented using an integer-adder. Experimental results show that the integer-adder DNN produces comparable speech quality to a full-precision DNN with the same structure, but it is more computation- and memory-efficient. Ye *et al.* [21] iteratively prunes a DNN for speech enhancement. Their experimental results show that the pruning method can remove roughly 50% of the trainable parameters without degrading subjective intelligibility. More recently, Wu *et al.* [22] proposed to use pruning and quantization techniques to reduce the size of a fully convolutional neural network (FCN) for time-domain speech enhancement. The results suggest that these techniques can significantly increase the compactness of the FCN without affecting the enhancement performance.

Although model compression techniques have been extensively developed in other fields such as image processing, it remains unclear for speech enhancement whether specific techniques are effective and how different techniques can be combined to achieve high compression rates. In this study, we propose a generic model compression pipeline for DNN-based speech enhancement, which consists of sparse regularization, iterative pruning and clustering-based quantization. The sparse regularization imposes sparsity of weight tensors, which allows for a higher pruning ratio under the constraint that the enhancement performance is not significantly degraded. With a sparsity-inducing regularizer, we train and prune the model alternately and iteratively, which significantly reduces the number of weights. Subsequently, we apply clustering-based quan-

This research was supported in part by an NIDCD grant (R01 DC012048), and the Ohio Supercomputer Center. The authors would like to thank A. Pandey for providing his implementation of TCNN.

tization to the remaining weights. Both pruning and quantization are performed based on a per-tensor sensitivity analysis, which can be beneficial if the weight distributions vary greatly between tensors. We evaluate the compression pipeline on different DNN architectures with different training targets. The results show that the proposed approach substantially reduces the sizes of all these models, without significantly sacrificing the enhancement performance.

The rest of this paper is organized as follows. We provide a detailed description of our approach in Section 2. The experimental setup and evaluation results are presented in Section 3. Section 4 concludes this paper.

2. SYSTEM DESCRIPTION

2.1. DNN-based speech enhancement

In this study, we focus on DNN compression for monaural speech enhancement, although our method can be easily applied to DNN-based multi-channel speech enhancement. The goal of monaural speech enhancement is to estimate target speech s given a single-microphone mixture y. Hence, DNN-based enhancement can be formulated as

$$z = \mathcal{F}_1(y), \tag{1}$$

$$\hat{x} = \mathcal{H}(z;\Theta), \tag{2}$$

$$\hat{s} = \mathcal{F}_2(\hat{x}, y), \tag{3}$$

where $\mathcal{F}_1$ and $\mathcal{F}_2$ are predefined transformation functions, and $\mathcal{H}$ the mapping function defined by a DNN. In the time-frequency domain, for example, $\mathcal{F}_1$ and $\mathcal{F}_2$ can be the short-time Fourier transform and waveform resynthesis, respectively. The symbol Θ denotes the set of all trainable parameters in the DNN, and $\hat{s}$ the enhanced speech signal. The parameters Θ are trained to minimize a loss function $\mathcal{L}(x, \hat{x}) = \mathcal{L}(x, \mathcal{H}(\mathcal{F}_1(y);\Theta))$, where x represents the training target.

2.2. Sparse regularization and iterative pruning

The granularity of sparsity affects the efficiency of hardware architecture. Fine-grained sparsity is a type of sparsity patterns where individual weights are masked as zero [6]. These sparsity patterns are typically irregular, which makes it difficult to apply hardware acceleration [23]. This problem can be alleviated by inducing coarse-grained sparsity, of which the pattern is more regular. In this study, we examine both unstructured and structured pruning. Specifically, unstructured pruning (or fine-grained pruning) removes each individual weight separately, while structured pruning (or coarse-grained pruning) limits sparsity to higher-level structure of weight tensors. For example, one can prune entire columns or rows of a weight matrix.

To perform structured pruning, we define the pruning granularity as follow. For a 2-D convolutional or deconvolutional layer, the weight tensor has a shape of $C_1 \times C_2 \times K_1 \times K_2$, where C_1 and C_2 represent the output and input channel dimensions respectively, and K_1 and K_2 the shapes of convolution kernels. Each kernel (i.e. a $K_1 \times K_2$ matrix) is treated as a weight group for pruning. For a 1-D convolutional or deconvolutional layer, the weights compose a 3-D tensor of shape $C_1 \times C_2 \times K$, where K denotes the kernel size. We treat each length-K vector as a weight group for pruning. For both recurrent layers and fully connected layers, each weight tensor is a matrix, of which each column is treated as a weight group for pruning. Take, for example, an LSTM layer. It has eight weight matrices, four for the layer input and the others for the hidden state from the last time step, corresponding to four different gates (i.e. input, forget, cell and output gates). Each group of four weight matrices is concatenated into a larger matrix and treated as a weight group for pruning. Note that we do not prune biases, as the number of biases is far smaller than that of weights.

Algorithm 1 Per-tensor sensitivity analysis for unstructured pruning

Input: (1) Validation set $\mathcal{V}$; (2) set $\mathcal{W}_l$ of all nonzero weights in the l-th weight tensor $\mathbf{W}_l$, $\forall l$; (3) loss function $\mathcal{L}(\mathcal{V}, \Theta)$, where Θ is the set of all nonzero trainable parameters in the DNN; (4) predefined tolerance value α_1.

Output: Pruning ratio β_l for weight tensor $\mathbf{W}_l$, $\forall l$.

```
for each tensor W_l do
    for β in {0%, 5%, 10%, ..., 90%, 95%, 100%} do
        Let U ⊆ W_l be the set of the β(%) of nonzero weights with the smallest absolute values in tensor W_l;
        I_U ← L(V, Θ|w = 0, ∀w ∈ U) − L(V, Θ);
        if I_U > α_1 then
            β_l ← β − 5%;
            break
        end if
    end for
    if β_l is not assigned any value then
        β_l ← 100%;
    end if
end for
return β_l for weight tensor W_l, ∀l
```

To achieve a higher compression rate, we propose to use sparsity-inducing regularization during training. Specifically, we perform ℓ_1 regularization for unstructured pruning:

$$\mathcal{R}_{\ell_1} = \frac{\lambda_1}{n(\mathcal{W})} \sum_{w \in \mathcal{W}} |w|, \tag{4}$$

where $\mathcal{W}$ is the set of all nonzero weights, and λ_1 a predefined weighting factor. The function $n(\cdot)$ calculates the cardinality of a set. Thus the new loss function can be written as $\mathcal{L}_{\ell_1} = \mathcal{L} + \mathcal{R}_{\ell_1}$. To impose group-level sparsity for structured pruning, group sparse regularization [24] is performed during training. We propose to use the following sparse group lasso (SGL) penalty:

$$\mathcal{R}_{\text{SGL}} = \frac{\lambda_1}{n(\mathcal{W})} \sum_{w \in \mathcal{W}} |w| + \frac{\lambda_2}{n(\mathcal{G})} \sum_{\mathbf{g} \in \mathcal{G}} \sqrt{p_{\mathbf{g}}} \left\|\mathbf{g}\right\|_2, \tag{5}$$

where $\mathcal{G}$ is the set of all weight groups, and $\left\|\cdot\right\|_2$ the ℓ_2 norm. The symbol $p_{\mathbf{g}}$ represents the number of weights in each weight group $\mathbf{g}$. Such a regularizer combines both ℓ_1 and $\ell_{2,1}$ norms with two weighting factors λ_1 and λ_2, which encourages group-level sparsity of the weight tensors [25]. The corresponding loss function can be written as $\mathcal{L}_{\text{SGL}} = \mathcal{L} + \mathcal{R}_{\text{SGL}}$.

The saliency of a specific set $\mathcal{U}$ of weights can be measured as the increase in the error incurred by setting them to zero. A validation set $\mathcal{V}$ is used for measuring weight saliency:

$$\mathcal{I}_{\mathcal{U}} = \mathcal{L}(\mathcal{V}, \Theta | w = 0, \forall w \in \mathcal{U}) - \mathcal{L}(\mathcal{V}, \Theta). \tag{6}$$

We conduct a per-tensor pruning sensitivity analysis to determine the pruning ratios that should be applied to particular weight tensors, following Algorithm 1. Unstructured pruning is then performed as per the tensor-wise pruning ratios. Subsequently, we fine-tune the pruned DNN to regain the lost performance. The fine-tuned DNN is evaluated on the validation set by two metrics, i.e. short-time objective intelligibility (STOI) [26] and perceptual evaluation of speech

quality (PESQ) [27]. This procedure is repeated until the number of pruned weights becomes trivial in an iteration or a significant degradation of STOI or PESQ is observed on the validation set. Note that the parameter set Θ becomes smaller after each pruning iteration. Similarly, one can conduct sensitivity analysis for structured pruning following the same procedures in Algorithm 1, except that the weight group saliency is measured as

$$\mathcal{I}_{\mathcal{U}} = \mathcal{L}(\mathcal{V}, \Theta | \mathbf{g} = \mathbf{0}, \forall \mathbf{g} \in \mathcal{U}) - \mathcal{L}(\mathcal{V}, \Theta), \tag{7}$$

where $\mathcal{U}$ represents a set of weight groups. Structured pruning and fine-tuning are then performed alternately and iteratively.

The selection between unstructured and structured pruning depends upon the accessibility of hardware acceleration for the underlying device. For devices on which acceleration is inaccessible, unstructured pruning would be the better choice, as it typically achieves higher compression rates than structured pruning under the constraint that the enhancement performance is not significantly degraded. For devices with accelerators, it would be better to use structured pruning.

2.3. Clustering-based quantization

To further compress the pruned DNN, we propose to use clustering-based quantization [16, 17]. Specifically, we partition the weights in each tensor into K clusters $S_1, S_2, \ldots, S_K$ via k-means clustering:

$$\underset{S_1, S_2, \ldots, S_K}{\arg\min} \sum_{k=1}^{K} \sum_{w \in S_k} |w - \mu_k|^2, \tag{8}$$

where μ_k is the centroid of cluster S_k. Prior to applying the k-means algorithm, the cluster centroids are initialized with K values evenly spaced over the interval $[w_{\min}, w_{\max}]$, where $w_{\min}$ and $w_{\max}$ are the minimum and maximum values of the weight tensor, respectively. Once the clustering algorithm converges, all the weights that fall into the same cluster are reassigned the value of the corresponding cluster centroid. Thus the original weights are approximated by these cluster centroids. Such a weight sharing mechanism reduces the number of effective weights needed to be stored.

A codebook is created to store the values of cluster centroids for each weight tensor, in which each weight is tied to the corresponding cluster index. During inference, the value of each weight is looked up in the codebook. An example of the clustering-based quantization is illustrated in Fig. 1. Each weight value is quantized to $\log_2 K$ bits, which are needed to store the corresponding cluster index. Assuming that the original weights are represented as 32-bit floating-point numbers, $32K$ additional bits are needed to store the codebook. Therefore, the compression rate can be calculated as

$$r = \frac{32N}{N \log_2 K + 32K}, \tag{9}$$

where N is the number of nonzero weights in the tensor. Note that only nonzero weights are subject to clustering and weight sharing.

To determine the number of clusters that should be used for each weight tensor, we conduct a per-tensor sensitivity analysis for quantization following Algorithm 2. Subsequently, we perform the clustering-based quantization following the analysis results. Unlike [17] in which the same number of clusters is used for all tensors, our method allows for quantizing each tensor using different numbers of bits, which is beneficial if the importance and sizes of weight tensors vary vastly.

Algorithm 2 Per-tensor sensitivity analysis for quantization

Input: (1) Validation set $\mathcal{V}$; (2) set $\mathcal{W}_l$ of all nonzero weights in the l-th weight tensor $\mathbf{W}_l$, $\forall l$; (3) loss function $\mathcal{L}(\mathcal{V}, \Theta)$, where Θ is the set of all nonzero trainable parameters in the DNN; (4) predefined tolerance value α_2.

Output: Number of clusters K_l for weight tensor $\mathbf{W}_l$, $\forall l$.

```
for each tensor W_l do
    K ← 1;
    while true do
        I_K ← L(V, Θ| quantize w to log2 K bits, ∀w ∈ W_l) − L(V, Θ);
        if I_K < α2 or 2K > n(W_l) then
            K_l ← K;
            break
        end if
    end while
    K ← 2K;
end for
return K_l for weight tensor W_l, ∀l
```

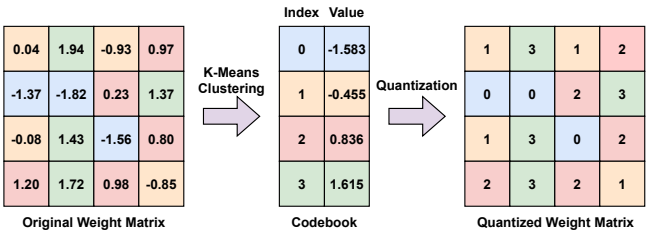

Fig. 1. Illustration of clustering-based quantization.

3. EVALUATION AND ANALYSIS

3.1. Data preparation

In our experiments, we use the training set of the WSJ0 dataset [28] as the speech corpus, which includes 12776 utterances from 101 speakers. We split these speakers into three groups (i.e. 89, 6 and 6 speakers) for training, validation and test sets, respectively. To create a training mixture, we mix a randomly selected training utterance with a random cut from 10,000 noises in a sound effect library available at https://www.sound-ideas.com. The signal-to-noise ratio (SNR) is randomly sampled between -5 and 0 dB. We create 320,000 mixtures for training. Following the same procedure, we create a validation set including 846 mixtures using a factory noise from the NOISEX-92 dataset [29]. For testing, we use two highly nonstationary noises, i.e. babble ("BAB") and cafeteria ("CAF"), from an Auditec CD available at http://www.auditec.com. We create a test set consisting of 846 mixtures for each of the two noises and each of three SNRs, i.e. -5, 0 and 5 dB.

3.2. DNNs for speech enhancement

To systematically examine our proposed model compression pipeline, we use the following four models for monaural speech enhancement, which have different network architectures and are trained with different targets. The first is a feedforward DNN (FDNN), which has three hidden layers with 2048 units in each layer. The ideal ratio mask is used as the training target. The second is an LSTM model that performs spectral mapping. It has four LSTM hidden layers with 1024 units in each layer, and the output layer consists

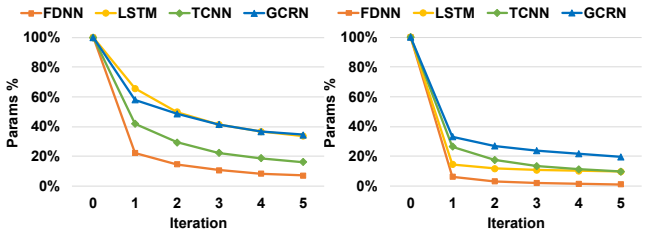

Fig. 2. The percent of the original number of trainable parameters at different pruning iterations. (a). Without, and (b). With sparse regularization.

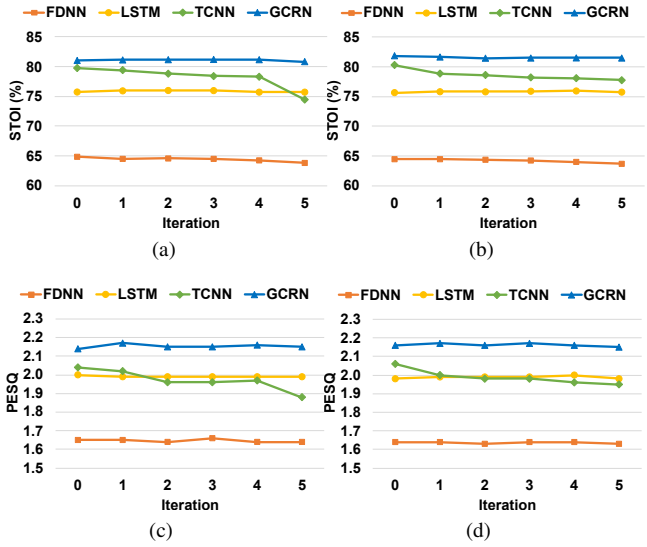

Fig. 3. STOI and PESQ scores for -5 dB SNR at different pruning iterations. (a)&(c). Without, and (b)&(d). With sparse regularization.

of a fully-connected layer followed by rectified linear activation function. The third is the temporal convolutional neural network (TCNN) developed in [30], which is a fully convolutional neural network for time-domain speech enhancement. The fourth is the gated convolutional recurrent network (GCRN) developed in [31], which is trained to learn a complex spectral mapping. We train all the models following the configurations in [31]. The validation set is used for both selecting the best model among different epochs and performing the sensitivity analysis for pruning and quantization.

For unstructured pruning, the initial value of λ_1 (see Eq. (4)) is empirically set to 0.1, 10, 0.02 and 1 for FDNN, LSTM, TCNN and GCRN, respectively. For structured pruning, the same initial values of λ_1 are used, and the initial value of λ_2 (see Eq. (5)) is set to 0.0005, 0.005, 0.02 and 0.05 for FDNN, LSTM, TCNN and GCRN, respectively. Both λ_1 and λ_2 decay by 10% every pruning iteration. The tolerance values (α_1, α_2) for sensitivity analysis (see Algorithms 1 and 2) are empirically set to (0.003, 0.0005), (0.03, 0.01), (0.0005, 0.00005) and (0.02, 0.005) for FDNN, LSTM, TCNN and GCRN, respectively.

3.3. Evaluation results

We first investigate the effects of sparsity-inducing regularization. Fig. 2 shows the percent of the original number of trainable parameters, with or without ℓ_1 regularization (see Eq. (4)) for unstructured pruning. We can observe that the use of regularization leads to higher compression rates for all the four DNNs. For example, the compression rate for GCRN after 5 pruning iterations can be increased from 2.9× to 5.1× by using sparse regularization over not using it. Fig. 3 shows the corresponding STOI and PESQ results at -5 dB SNR, which suggests that our proposed pruning method does not significantly degrade the enhancement performance, except that the STOI and PESQ scores produced by TCNN slightly decrease as it is pruned for more iterations.

We additionally train two small DNNs, i.e. FDNN_B and LSTM_B. They have the same structures as the FDNN and LSTM described in Section 3.2, except that there are only 200 and 320 units in each hidden layer of FDNN_B and LSTM_B, respectively. Thus FDNN_B and LSTM_B have similar model sizes to the original FDNN and LSTM pruned for 5 iterations, which we denote as FDNN_P and LSTM_P in Table 1. It can be observed that FDNN_P and LSTM_P produce higher STOI and PESQ than FDNN_B and LSTM_B, which demonstrates the advantage of training and pruning a large DNN over directly training a small DNN.

Table 2 presents the STOI and PESQ results for uncompressed models (FDNN_U, LSTM_U, TCNN_U and GCRN_U) and compressed models (FDNN_C, LSTM_C, TCNN_C, GCRN_C and GCRN_C-SP) with regularization, pruning (for 5 iterations) and quantization. Note that all the compressed models are subject to unstructured pruning, except that GCRN_C-SP is subject to structured pruning. We can see that our proposed compression pipeline leads to a high compression rate (denoted as "CR" in Table 2) without significantly sacrificing the enhancement performance. Moreover, unstructured pruning yields a higher compression rate than structured pruning, although producing less regular sparsity patterns.

Table 1. Comparisons between pruned models and comparably-sized unpruned models.

Metric	STOI (%)			PESQ			# Param.
SNR	-5 dB	0 dB	5 dB	-5 dB	0 dB	5 dB	
Mixture	57.86	70.14	81.48	1.51	1.80	2.12	-
FDNN_P	63.72	77.63	86.59	1.63	2.06	2.44	1.15 M
FDNN_B	62.89	76.25	85.54	1.58	1.98	2.35	1.45 M
LSTM_P	75.76	86.62	92.25	1.98	2.46	2.85	2.93 M
LSTM_B	72.25	83.98	90.31	1.85	2.31	2.68	3.14 M

Table 2. Comparisons between uncompressed and compressed models.

Metric	STOI (%)			PESQ			Storage	CR
SNR	-5 dB	0 dB	5 dB	-5 dB	0 dB	5 dB		
Mixture	57.86	70.14	81.48	1.51	1.80	2.12	-	-
FDNN_U	64.87	78.64	87.25	1.65	2.09	2.47	34.54 MB	1×
FDNN_C	63.58	77.51	86.50	1.64	2.06	2.44	0.10 MB	343×
LSTM_U	75.74	86.47	92.04	2.00	2.47	2.84	115.27 MB	1×
LSTM_C	75.83	86.62	92.25	1.98	2.46	2.84	2.49 MB	46×
TCNN_U	79.76	89.72	93.96	2.04	2.52	2.86	19.28 MB	1×
TCNN_C	77.77	88.46	93.26	1.95	2.44	2.79	0.56 MB	34×
GCRN_U	81.03	90.43	94.43	2.14	2.65	3.01	37.27 MB	1×
GCRN_C	80.66	90.15	94.26	2.15	2.65	3.02	1.11 MB	34×
GCRN_C-SP	81.05	90.32	94.35	2.15	2.66	3.03	4.11 MB	9×

4. CONCLUSION

We have developed a new pipeline to compress DNNs for speech enhancement. The pipeline consists of three techniques: sparse regularization, iterative pruning and clustering-based quantization. Our experimental results show that the proposed pipeline substantially reduces the sizes of four different DNNs for speech enhancement without significantly degrading the enhancement performance. In addition, we find that training and compressing a large DNN achieves better enhancement results than directly training a small DNN that has a comparable size to the compressed DNN, which suggests the benefits of using our model compression approach.

5. REFERENCES

[1] D. L. Wang, "On ideal binary mask as the computational goal of auditory scene analysis," in *Speech Separation by Humans and Machines*, pp. 181–197. Springer, 2005.

[2] Y. Wang and D. L. Wang, "Towards scaling up classification-based speech separation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 7, pp. 1381–1390, 2013.

[3] D. L. Wang and J. Chen, "Supervised speech separation based on deep learning: An overview," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 10, pp. 1702–1726, 2018.

[4] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[5] R. Reed, "Pruning algorithms-a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[6] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, 1990, pp. 598–605.

[7] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[8] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "FitNets: Hints for thin deep nets," in *International Conference on Learning Representations*, 2015.

[9] Y. Chebotar and A. Waters, "Distilling knowledge from ensembles of neural networks for speech recognition.," in *Interspeech*, 2016, pp. 3439–3443.

[10] L. Lu, M. Guo, and S. Renals, "Knowledge distillation for small-footprint highway networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2017, pp. 4820–4824.

[11] 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," *arXiv preprint arXiv:1602.07360*, 2016.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[13] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[14] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

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

[16] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[17] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *International Conference on Learning Representations*, 2016.

[18] S. Samui, I. Chakrabarti, and S. K. Ghosh, "Tensor-train long short-term memory for monaural speech enhancement," *arXiv preprint arXiv:1812.10095*, 2018.

[19] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[20] Y.-C. Lin, Y.-T. Hsu, S.-W. Fu, Y. Tsao, and T.-W. Kuo, "Ia-net: Acceleration and compression of speech enhancement using integer-adder deep neural network.," in *Interspeech*, 2019, pp. 1801–1805.

[21] F. Ye, Y. Tsao, and F. Chen, "Subjective feedback-based neural network pruning for speech enhancement," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 2019, pp. 673–677.

[22] J.-Y. Wu, C. Yu, S.-W. Fu, C.-T. Liu, S.-Y. Chien, and Y. Tsao, "Increasing compactness of deep learning based speech enhancement models with parameter pruning and quantization techniques," *IEEE Signal Processing Letters*, vol. 26, no. 12, pp. 1887–1891, 2019.

[23] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the granularity of sparsity in convolutional neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 13–20.

[24] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini, "Group sparse regularization for deep neural networks," *Neurocomputing*, vol. 241, pp. 81–89, 2017.

[25] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani, "A sparse-group lasso," *Journal of Computational and Graphical Statistics*, vol. 22, no. 2, pp. 231–245, 2013.

[26] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen, "An algorithm for intelligibility prediction of time–frequency weighted noisy speech," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 7, pp. 2125–2136, 2011.

[27] A. W. Rix, J. G. Beerends, M. P. Hollier, and A. P. Hekstra, "Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (Cat. No. 01CH37221)*. IEEE, 2001, vol. 2, pp. 749–752.

[28] J. Garofolo, D. Graff, D. Paul, and D. Pallett, "CSR-I (WSJ0) complete LDC93S6A," *Web Download. Philadelphia: Linguistic Data Consortium*, vol. 83, 1993.

[29] A. Varga and H. J. 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 Communication*, vol. 12, no. 3, pp. 247–251, 1993.

[30] A. Pandey and D. L. Wang, "TCNN: Temporal convolutional neural network for real-time speech enhancement in the time domain," in *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2019, pp. 6875–6879.

[31] K. Tan and D. L. Wang, "Learning complex spectral mapping with gated convolutional recurrent networks for monaural speech enhancement," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 380–390, 2019.