Towards Model Compression for Deep Learning Based Speech Enhancement

Ke Tan and DeLiang Wang, *Fellow, IEEE*

***Abstract*—The use of deep neural networks (DNNs) has dramatically elevated the performance of speech enhancement over the last decade. However, to achieve strong enhancement performance typically requires a large DNN, which is both memory and computation consuming, making it difficult to deploy such speech enhancement systems on devices with limited hardware resources or in applications with strict latency requirements. In this study, we propose two compression pipelines to reduce the model size for DNN-based speech enhancement, which incorporates three different techniques: sparse regularization, iterative pruning and clustering-based quantization. We systematically investigate these techniques and evaluate the proposed compression pipelines. Experimental results demonstrate that our approach reduces the sizes of four different models by large margins without significantly sacrificing their enhancement performance. In addition, we find that the proposed approach performs well on speaker separation, which further demonstrates the effectiveness of the approach for compressing speech separation models.**

***Index Terms*—Model compression, sparse regularization, pruning, quantization, speech enhancement.**

I. Introduction

SPEECH enhancement aims to separate target speech from background noise. Inspired by the concept of time-frequency (T-F) masking in computational auditory scene analysis, speech enhancement has been formulated as supervised learning [45], [46]. In the past decade, many data-driven algorithms have been developed to address this problem, in which discriminative patterns within signals are learned from training data. The rapid rise in deep learning has tremendously benefited supervised speech enhancement [47]. Since deep learning became a dominant approach to speech enhancement in the research community, there has been increasing interest in deploying DNN-based enhancement systems for real-world applications and products (e.g. headphones). Due to the well-recognized over-parameterization property of DNNs [1], [5], however, to achieve satisfactory enhancement performance would require a large DNN, which can be both computationally intensive and memory consuming. It is difficult to deploy such DNNs in latency-sensitive applications or on resource-limited devices. Hence, it becomes an increasingly important problem to reduce memory and computation in DNNs for speech enhancement.

Various model compression techniques have been developed in the deep learning community, which can be broadly categorized into two classes [4]. The first class reduces the number of trainable parameters. A widely-used technique of this class is network pruning, which selects and removes the least important set of weights based on certain criteria [34]. Two pioneering works are optimal brain damage [23] and optimal brain surgeon [12], which leverage the Hessian matrix of the loss function to determine 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 regain the lost accuracy. Another effective technique is tensor decomposition, which reduces the redundancy by decomposing a large weight tensor into multiple smaller tensors based on the low-rankness of the weight tensor. Moreover, one can transfer the knowledge from a pretrained large model to a relatively small model, known as knowledge distillation [15]. Soft targets produced by the large DNN are used to guide the training of the smaller DNN. This approach has proven to be effective in classification tasks such as image classification [36] and speech recognition [2], [27]. Other related studies reduce the inference cost of DNNs by designing more parameter-efficient network architectures [16], [17], [52]. The second class of model compression techniques is network quantization, which reduces the bitwidth of weights, activations, or both. A simple method is to train DNNs with full precision and then directly quantize the learned weights, which was shown to significantly degrade the accuracy for relatively small DNNs [18], [22]. To compensate for the loss of accuracy, quantization-aware training was developed in [18], which incorporates simulated quantization effects during training. Furthermore, weight quantization can be performed by applying clustering to the trained weights [3], [10], [11], [19].

Over the past several years, increasing research efforts have been devoted to improving the inference efficiency of DNNs for speech enhancement. In [25], an integer-adder DNN was developed, where an integer-adder is used to implement floating-point multiplication. Evaluation results show that the integer-adder DNN yields comparable speech quality to a full-precision DNN with the same architecture, while more efficient in terms of both computation and memory. Ye *et al.* [50] iteratively prune a DNN for speech enhancement, where the importance of weights is

Manuscript received February 3, 2021; revised April 27, 2021; accepted May 14, 2021. Date of publication May 20, 2021; date of current version May 28, 2021. This work was supported in part by an National Institute on Deafness and Other Communication Disorders (NIDCD) under Grant R01 DC012048, and by Ohio Supercomputer Center. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Xiao-Lei Zhang *(Corresponding author: Ke Tan.)*

Ke Tan is with the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210-1277 USA (e-mail: tan.650@osu.edu).

DeLiang Wang is with the Department of Computer Science and Engineering, and the Center for Cognitive, and Brain Sciences, The Ohio State University, Columbus, OH 43210-1277 USA (e-mail: dwang@cse.ohio-state.edu).

Digital Object Identifier 10.1109/TASLP.2021.3082282

determined by simply comparing the absolute values of weights to a predefined threshold. The experimental results suggest that their pruning method can compress a feedforward DNN by a factor of roughly 2, without degrading the enhancement performance in terms of subjective intelligibility. In [49], Wu *et al.* used pruning and quantization techniques to compress a fully convolutional neural network (FCN) for time-domain speech enhancement. Their results show that these techniques can significantly reduce the size of the FCN without performance degradation. More recently, Fedorov *et al.* [6] performed pruning and integer quantization to compress recurrent neural networks (RNNs) for speech enhancement, which can reduce the RNN size to 37% with a 0.2 dB decrease in scale-invariant signal-to-noise ratio (SI-SNR).

Although DNN compression techniques have been extensively developed and investigated in other fields such as image processing, most of these techniques have been evaluated only on classification tasks. Given that DNN-based speech enhancement is usually treated as a regression task, it remains unclear for speech enhancement whether specific compression techniques are effective and how different techniques can be combined to achieve high compression rates. Furthermore, a generic compression pipeline would be desired due to the wide variety and fast evolution of speech enhancement models. With these considerations in mind, we recently developed two preliminary model compression pipelines for DNN-based speech enhancement [41]. The compression pipelines consist of sparse regularization, iterative pruning and clustering-based quantization. Sparse regularization imposes sparsity of weight tensors through DNN training, which leads to a higher pruning ratio without significantly sacrificing the enhancement performance. We train and prune the DNN alternately and iteratively, and subsequently apply k-means clustering based quantization to the remaining weights. We perform pruning and quantization both based on per-tensor sensitivity analyses, which would benefit the selection of pruning ratios and bitwidths if the weight distributions vary vastly between tensors. Building on [41], the present study additionally examines the effects of each individual technique and their combinations on different types of speech enhancement models, and further investigates the compression pipelines on speaker separation models. Specifically, we evaluate the compression pipelines on speech enhancement models with different designs, including DNN types, training targets and processing domains. Evaluation results show that the proposed approach substantially reduces the sizes of all these models, without significant performance degradation. In addition, we find that our approach performs well on two representative models for talker-independent speaker separation.

The rest of this paper is organized as follows. In Section II, we describe our proposed approach in detail. In Section III, we provide the experimental setup. Experimental results are presented and analyzed in Section IV. Section V concludes this paper.

II. Algorithm Description

A. DNN-Based Speech Enhancement

In this study, we focus on DNN compression for monaural speech enhancement, although our approach is expected to apply to DNNs for multi-channel speech enhancement. Given a single-channel mixture y, the goal of monaural speech enhancement is to estimate target speech s. The mixture can be modeled as

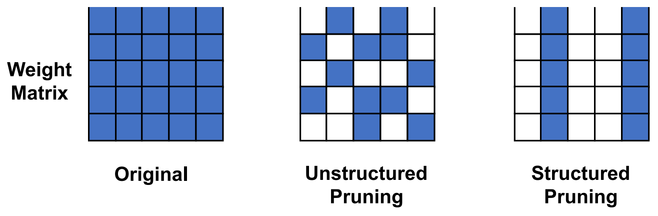


Fig. 1. (Color Online). Illustration of unstructured and structured pruning. White cells indicate the pruned weights, and blue cells the remaining weights.

$$y = s + v, \tag{1}$$

where v represents background noise. Thus DNN-based enhancement can be formulated as

$$z = \mathcal{F}_1(y), \tag{2}$$

$$\hat{x} = \mathcal{H}(z; \Theta), \tag{3}$$

$$\hat{s} = \mathcal{F}_2(\hat{x}, y), \tag{4}$$

where $\mathcal{F}_1$ and $\mathcal{F}_2$ denote transforms, and $\mathcal{H}$ the nonlinear mapping function represented by a DNN. For T-F domain enhancement, $\mathcal{F}_1$ and $\mathcal{F}_2$ can be short-time Fourier transform and waveform resynthesis, respectively. For time-domain enhancement, $\mathcal{F}_1$ and $\mathcal{F}_2$ can be segmentation and overlap-add, respectively. The symbol Θ denotes the set of all trainable parameters in the DNN, and $\hat{s}$ the estimated speech signal. Symbols z and $\hat{x}$ represent the input and output of the DNN, respectively. 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 is the training target.

B. Iterative Unstructured and Structured Pruning

A typical procedure of network pruning comprises three stages: (i) training a large DNN that achieves satisfactory performance, (ii) removing a specific set of weights in the trained DNN with a certain criterion, and (iii) fine-tuning the pruned DNN. One can view the removed weights as zero, and thus pruning leads to sparse weight tensors. The granularity of tensor sparsity impacts the efficiency of hardware architecture. Fine-grained sparsity is a type of sparsity patterns where individual weights are set to zero [23]. Such sparsity patterns are typically irregular, which makes it difficult to apply hardware acceleration [30]. This problem can be mitigated by imposing coarse-grained sparsity, of which the pattern is more regular. We investigate both unstructured and structured pruning. Specifically, unstructured pruning removes each individual weight separately, while structured pruning groups of weights, as illustrated in Fig. 1. For example, one can remove entire columns or rows of a weight matrix.

To perform structured pruning, we define the pruning granularity as follows. For convolutional/deconvolutional layers, we treat each kernel as a weight group for pruning. Specifically, each weight group for 2-D convolutional/deconvolutional layers is a matrix, and for 1-D convolutional/deconvolutional layers is a vector. 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. For example, a long short-term memory (LSTM) layer has eight weight matrices, four for the layer input and the others for the hidden state from the last time step, which correspond to four gates (i.e. input, forget, cell and output gates). In the implementation of LSTM, each group of weight matrices for the four gates is typically concatenated, which amounts to two larger matrices. We treat each column of these matrices as a weight group for pruning. Such pruning granularity would lead to reasonably high compression ratios and produce coarse-grained sparsity that is more hardware-friendly than fine-grained sparsity [30]. Note that we do not prune biases, as the number of biases is small relative to 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 $\Theta$ is the set of all nonzero
  trainable parameters in the DNN; (4) predefined
  tolerance value $\alpha_1$.
Output: Pruning ratio $\beta_l$ for weight tensor $\mathbf{W}_l$, $\forall l$.
 1: for each tensor $\mathbf{W}_l$ do
 2:   for $\beta$ in $\{0\%, 5\%, 10\%, \ldots, 90\%, 95\%, 100\%\}$ do
 3:     Let $\mathcal{U} \subseteq \mathcal{W}_l$ be the set of the $\beta(\%)$ of nonzero
        weights with the smallest absolute values in tensor
        $\mathbf{W}_l$;
 4:     $\mathcal{I}_\mathcal{U} \leftarrow \mathcal{L}(\mathcal{V}, \Theta | w = 0, \forall w \in \mathcal{U}) - \mathcal{L}(\mathcal{V}, \Theta)$;
 5:     if $\mathcal{I}_\mathcal{U} > \alpha_1$ then
 6:       $\beta_l \leftarrow \beta - 5\%$;
 7:       break
 8:     end if
 9:   end for
10:   if $\beta_l$ is not assigned any value then
11:     $\beta_l \leftarrow 100\%$;
12:   end if
13: end for
14: return $\beta_l$ for weight tensor $\mathbf{W}_l$, $\forall l$
```

For network pruning, the key issue is to define the pruning criterion, which determines the set of weights to be removed. To perform unstructured pruning, we define the saliency of a specific set $\mathcal{U}$ of weights as the increase in the error induced by removing them. Specifically, weight saliency is measured using a validation set $\mathcal{V}$:

$$\mathcal{I}_\mathcal{U} = \mathcal{L}(\mathcal{V}, \Theta | w = 0, \forall w \in \mathcal{U}) - \mathcal{L}(\mathcal{V}, \Theta). \tag{5}$$

Unlike [6], [49], [50], we conduct a per-tensor pruning sensitivity analysis to determine the pruning ratios for all weight tensors, following Algorithm 1. Subsequently, we perform unstructured pruning as per tensor-wise pruning ratio. The pruned DNN is then fine-tuned to recover the enhancement performance. We evaluate the fine-tuned DNN on the validation set by two metrics, i.e. short-time objective intelligibility (STOI) [39] and perceptual evaluation of speech quality (PESQ) [35]. Such pruning and fine-tuning operations are performed iteratively and alternately. We repeat this procedure until the number of pruned weights becomes trivial in an iteration or a significant decrease in STOI or PESQ is observed on the validation set. Note that both pruning and fine-tuning are performed on the entire network. For structured pruning, the weight group saliency is measured as

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

where $\mathcal{U}$ represents a set of weight groups. Similarly, we conduct a sensitivity analysis following Algorithm 2. Structured pruning and fine-tuning are then performed for multiple iterations. Note that the size of the parameter set Θ decreases after each pruning iteration.

Algorithm 2: Per-tensor sensitivity analysis for structured pruning

```
Input: (1) Validation set $\mathcal{V}$; (2) set $\mathcal{G}_l$ of all nonzero
  weight groups in the $l$-th weight tensor $\mathbf{W}_l$, $\forall l$; (3) loss
  function $\mathcal{L}(\mathcal{V}, \Theta)$, where $\Theta$ is the set of all nonzero
  trainable parameters in the DNN; (4) predefined
  tolerance value $\alpha_1$.
Output: Pruning ratio $\beta_l$ for weight tensor $\mathbf{W}_l$, $\forall l$.
 1: for each tensor $\mathbf{W}_l$ do
 2:   for $\beta$ in $\{0\%, 5\%, 10\%, \ldots, 90\%, 95\%, 100\%\}$ do
 3:     Let $\mathcal{U} \subseteq \mathcal{G}_l$ be the set of the $\beta(\%)$ of nonzero
        weight groups with the smallest $\ell_1$ norms in tensor
        $\mathbf{W}_l$;
 4:     $\mathcal{I}_\mathcal{U} \leftarrow \mathcal{L}(\mathcal{V}, \Theta | \mathbf{g} = \mathbf{0}, \forall \mathbf{g} \in \mathcal{U}) - \mathcal{L}(\mathcal{V}, \Theta)$;
 5:     if $\mathcal{I}_\mathcal{U} > \alpha_1$ then
 6:       $\beta_l \leftarrow \beta - 5\%$;
 7:       break
 8:     end if
 9:   end for
10:   if $\beta_l$ is not assigned any value then
11:     $\beta_l \leftarrow 100\%$;
12:   end if
13: end for
14: return $\beta_l$ for weight tensor $\mathbf{W}_l$, $\forall l$
```

Our method is beneficial in two aspects. First, some existing pruning methods (e.g. [50]) use a common threshold to differentiate unimportant weights from the others for all layers throughout the DNN. This can greatly limit the pruning of the more redundant layers or over-prune the less redundant layers, particularly if the importance of layers varies significantly. Such a problem can be alleviated by our sensitivity analysis. Second, we perform pruning and fine-tuning iteratively, and evaluate the resulting model on speech enhancement metrics (STOI and PESQ) for each iteration. This can substantially reduce the risk of over-pruning and the corresponding unrecoverable performance degradation, even when the importance of a weight is not strongly correlated with its magnitude [31].

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

C. Sparse Regularization

To increase pruning ratios without performance degradation, we propose to use sparse regularization during training and fine-tuning. A principal way to impose weight-level sparsity is ℓ_1 regularization, which penalizes the sum of absolute values of weights during training. Specifically, ℓ_1 regularization encourages less important weights to become zero, reducing the resulting performance degradation. Hence this may result in higher pruning ratios with our pruning criterion. The ℓ_1 regularizer can be written as

$$\mathcal{R}_{\ell_1} = \frac{\lambda_1}{n(\mathcal{W})} \sum_{w \in \mathcal{W}} |w|, \tag{7}$$

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 is $\mathcal{L}_{\ell_1} = \mathcal{L} + \mathcal{R}_{\ell_1}$.

Group-level sparsity can be induced by a group lasso penalty [7]:

$$\mathcal{R}_{\ell_{2,1}} = \frac{\lambda_2}{n(\mathcal{G})} \sum_{\mathbf{g} \in \mathcal{G}} \sqrt{p_{\mathbf{g}}} \, \|\mathbf{g}\|_2 \,, \tag{8}$$

where $\mathcal{G}$ is the set of all weight groups, and $\|\cdot\|_2$ the ℓ_2 norm. Symbol $p_{\mathbf{g}}$ represents the number of weights in each weight group $\mathbf{g}$, and λ_2 a weighting factor. With such a penalty, all weights in a group are simultaneously either encouraged to be zero, or not. An extended version is sparse group lasso (SGL), which further imposes sparsity on the non-sparse groups by additionally incorporating ℓ_1 regularization [37], [38]:

$$\begin{aligned} \mathcal{R}_{\text{SGL}} &= \mathcal{R}_{\ell_1} + \mathcal{R}_{\ell_{2,1}} \\ &= \frac{\lambda_1}{n(\mathcal{W})} \sum_{w \in \mathcal{W}} |w| + \frac{\lambda_2}{n(\mathcal{G})} \sum_{\mathbf{g} \in \mathcal{G}} \sqrt{p_{\mathbf{g}}} \, \|\mathbf{g}\|_2 \,. \end{aligned} \tag{9}$$

The corresponding loss function is $\mathcal{L}_{\text{SGL}} = \mathcal{L} + \mathcal{R}_{\text{SGL}}$. Based on different pruning granularities, we adopt $\mathcal{L}_{\ell_1}$ for unstructured pruning and $\mathcal{L}_{\text{SGL}}$ for structured pruning.

D. Clustering-Based Quantization

To further compress the pruned DNN, we propose to use clustering-based quantization [10], [11]. Specifically, the weights in each tensor are partitioned into K clusters $S_1, S_2, \ldots, S_K$ through k-means clustering:

$$\underset{S_1, S_2, \ldots, S_K}{\arg\min} \sum_{k=1}^{K} \sum_{w \in S_k} |w - \mu_k|^2, \tag{10}$$

where μ_k is the centroid of cluster S_k. Following [11], we initialize the cluster centroids with K values evenly spaced over the interval $[w_{\text{min}}, w_{\text{max}}]$ prior to performing k-means clustering, where w_{min} and w_{max} represent the minimum and maximum values of the weight tensor, respectively. Once the clustering algorithm converges, we reset all the weights that fall into the same cluster to the value of the corresponding centroid. Thus the original weights are approximated by these cluster centroids. Such a weight sharing mechanism substantially reduces the number of effective weight values that need to be stored. Each weight can be represented as a cluster index. Note that only nonzero weights are subject to clustering and weight sharing.

We create a codebook to store the values of the cluster centroids for each weight tensor, in which each nonzero weight is tied to the corresponding cluster index. During inference, the value of each weight is looked up in the codebook. Fig. 2 illustrates clustering-based quantization. Specifically, we quantize each weight value to $\log_2 K$ bits. In other words, it requires $\log_2 K$ bits to store the corresponding cluster index. Assuming that the original weights are 32-bit floating-point numbers, to store the codebook needs $32\,K$ additional bits. Hence, the compression rate for quantization is calculated as

$$r = \frac{32\,N}{N \log_2 K + 32\,K}, \tag{11}$$

where N denotes the number of nonzero weights in the tensor.

Algorithm 3: 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 $\Theta$ is the set of all nonzero
  trainable parameters in the DNN; (4) predefined
  tolerance value $\alpha_2$.
Output: Number of clusters $K_l$ for weight tensor $\mathbf{W}_l$, $\forall l$.
 1: for each tensor $\mathbf{W}_l$ do
 2:   $K \leftarrow 1$;
 3:   while true do
 4:     $\mathcal{I}_K \leftarrow \mathcal{L}(\mathcal{V}, \Theta|$ quantize $w$ to $\log_2 K$ bits, $\forall w \in$
        $\mathcal{W}_l) - \mathcal{L}(\mathcal{V}, \Theta)$;
 5:     if $\mathcal{I}_K < \alpha_2$ or $2\,K > n(\mathcal{W}_l)$ then
 6:       $K_l \leftarrow K$;
 7:       break
 8:     end if
 9:   end while
10:   $K \leftarrow 2\,K$;
11: end for
12: return $K_l$ for weight tensor $\mathbf{W}_l$, $\forall l$
```

A common issue in quantization techniques is how to maintain the performance of DNNs. For clustering-based quantization, selecting an appropriate value of K is critical for achieving this goal. Given that the number of nonzero weights may vary greatly between weight tensors, we propose to conduct a per-tensor sensitivity analysis for quantization following Algorithm 3. The idea is to gradually increase the number of clusters for each weight tensor and measure the corresponding increase in the validation loss. The results of this sensitivity analysis are used to quantize weights in each weight tensor. Unlike [11] in which the same number of clusters is used for all weight tensors, our method allows for quantizing each tensor using

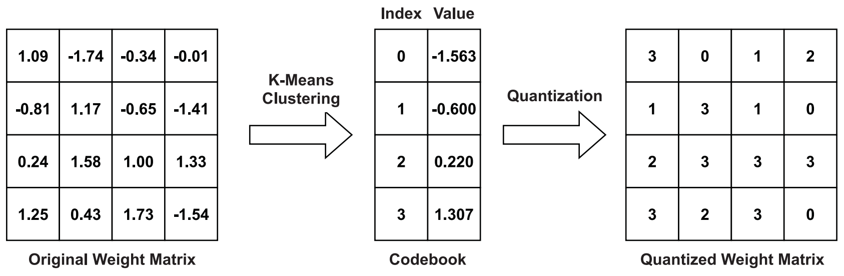


Fig. 2. Illustration of clustering-based quantization.

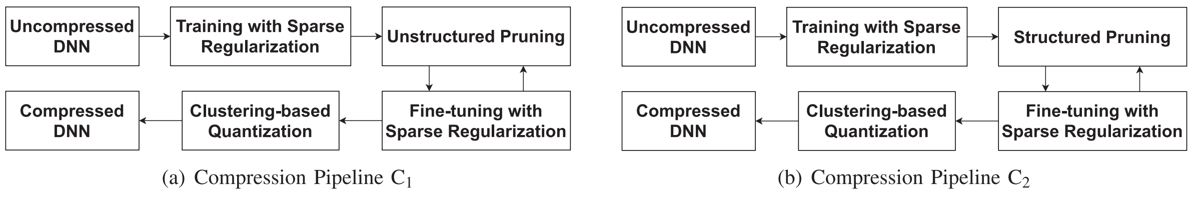

Uncompressed DNN
Training with Sparse Regularization
Unstructured Pruning
Fine-tuning with Sparse Regularization
Clustering-based Quantization
Compressed DNN
(a) Compression Pipeline C1
Uncompressed DNN
Training with Sparse Regularization
Structured Pruning
Fine-tuning with Sparse Regularization
Clustering-based Quantization
Compressed DNN
(b) Compression Pipeline C2

Fig. 3. Illustration of the proposed compression pipelines.

different numbers of bits, which potentially leads to higher compression rates.

Thus we can derive two compression pipelines by combining sparse regularization, iterative pruning and clustering-based quantization, as illustrated in Fig. 3. In the compression pipeline depicted in Fig. 3(a), we apply ℓ_1 regularization and unstructured pruning. In the other pipeline (see Fig. 3(b)), we apply group sparse regularization (see Eq. (9)) and structured pruning.

III. Experimental Setup

A. Data Preparation

In our experiments, we use the training set of the WSJ0 dataset [8] for evaluation, which contains 12 776 utterances from 101 speakers. These speakers are split into three groups, which include 89, 6 and 6 speakers for training, validation and testing, respectively. More specifically, the speaker groups for validation and testing include 3 males and 3 females. We use 10 000 noises from a sound effect library[1] for training, and a factory noise from the NOISEX-92 dataset [43] for validation. To create test sets, we use two highly nonstationary noises, i.e. babble ("BAB") and cafeteria ("CAF"), from an Auditec CD.[2]

Our training set includes 320 000 mixtures, and its total duration is roughly 600 hours. To create a training mixture, we mix a randomly sampled training utterance with a random segment from the 10 000 training noises. The signal-to-noise ratio (SNR) is randomly sampled between -5 and 0 dB. Following the same procedure, we create a validation set consisting of 846 mixtures. A test set including 846 mixtures is created for each of the two noises and each of three SNRs, i.e. -5, 0 and 5 dB.

In this study, all signals are sampled at 16 kHz. Each noisy mixture is rescaled by a factor such that the root mean square of the mixture waveform is 1. We use the same factor to rescale the corresponding target speech waveform. A 20-ms Hamming window is utilized to produce a set of time frames, with a 50% overlap between adjacent frames. We apply a 320-point (16 kHz $\times$ 20 ms) discrete Fourier transform to each frame, which yields 161-dimensional one-sided spectra.

B. Speech Enhancement Models

To systematically investigate the proposed model compression pipelines, we use the following four models for monaural speech enhancement, which have different designs including DNN types, training targets and processing domains.

1) Feedforward DNN: The first model is a feedforward DNN (FDNN), which has three hidden layers with 2048 units in each layer. We use the ideal ratio mask [48] as the training target:

$$\mathrm{IRM}(m,f) = \sqrt{\frac{|S(m,f)|^2}{|S(m,f)|^2 + |N(m,f)|^2}}, \qquad (12)$$

where $|S(m,f)|^2$ and $|N(m,f)|^2$ represent speech energy and noise energy within the T-F unit at time frame m and frequency bin f, respectively. The magnitude spectrogram is used as the FDNN input.

2) Lstm: The second is a recurrent LSTM model that performs spectral mapping in the magnitude domain. It has four LSTM hidden layers with 1024 units in each layer, and the output layer is a fully-connected layer followed by rectified linear activation function [9].

3) Temporal Convolutional Neural Network: The third model is a temporal convolutional neural network (TCNN) developed in a recent study [32]. The TCNN is a fully convolutional neural network, which directly maps from noisy speech to clean speech in the time domain.

4) Gated Convolutional Recurrent Network: The fourth is a newly-developed gated convolutional recurrent network (GCRN) [40]. The GCRN has an encoder-decoder architecture, which incorporates convolutional layers and recurrent layers. It is trained to perform complex spectral mapping, where the real

[1][Online]. Available: https://www.sound-ideas.com
[2][Online]: Available: http://www.auditec.com

TABLE I
COMPARISONS BETWEEN UNCOMPRESSED AND COMPRESSED MODELS

Metric	STOI (%)						PESQ						Model Size	Compression Rate	# Pruning Iterations
SNR	-5 dB		0 dB		5 dB		-5 dB		0 dB		5 dB				
Noise	BAB	CAF	BAB	CAF	BAB	CAF	BAB	CAF	BAB	CAF	BAB	CAF			
Mixture	58.49	57.22	78.44	78.83	87.21	87.29	1.56	1.46	1.82	1.77	2.12	2.12	-	-	-
FDNN_U	64.35	65.38	78.44	78.83	87.21	87.29	1.60	1.69	2.04	2.13	2.43	2.51	34.54 MB	1×	-
FDNN_{C_1}	62.69	64.46	77.02	77.99	86.36	86.63	1.60	1.67	2.01	2.11	2.39	2.49	0.10 MB	343×	5
FDNN_{C_2}	63.72	63.81	77.11	77.82	86.10	86.37	1.59	1.65	1.98	2.09	2.37	2.46	0.55 MB	63×	3
LSTM_U	76.39	75.09	87.10	85.83	92.34	91.73	1.98	2.01	2.48	2.45	2.86	2.82	115.27 MB	1×	-
LSTM_{C_1}	76.76	74.90	87.31	85.92	92.60	91.89	1.96	1.99	2.47	2.44	2.85	2.82	2.49 MB	46×	5
LSTM_{C_2}	77.52	74.91	87.38	85.88	92.46	91.73	2.01	2.01	2.49	2.46	2.86	2.83	9.97 MB	12×	4
TCNN_U	81.08	78.44	90.51	88.92	94.31	93.60	2.06	2.01	2.56	2.47	2.90	2.82	19.28 MB	1×	-
TCNN_{C_1}	80.10	77.39	89.77	88.41	93.69	93.03	2.00	1.97	2.50	2.44	2.83	2.78	0.94 MB	21×	3
TCNN_{C_2}	80.33	77.34	89.80	88.35	93.82	93.12	2.03	1.97	2.53	2.44	2.87	2.80	0.91 MB	21×	2
GCRN_U	82.38	79.68	91.16	89.70	94.74	94.12	2.17	2.10	2.70	2.59	3.05	2.97	37.27 MB	1×	-
GCRN_{C_1}	82.12	79.19	90.96	89.34	94.62	93.89	2.19	2.10	2.70	2.59	3.06	2.98	1.11 MB	34×	5
GCRN_{C_2}	82.55	79.55	91.07	89.56	94.69	94.01	2.20	2.10	2.71	2.60	3.07	2.98	4.11 MB	9×	5

and imaginary spectrograms of clean speech are estimated from those of noisy speech.

For TCNN and GCRN, we use the same network hyperparameters in [32] and [40]. Note that all these four DNNs are causal. We choose causal DNNs to avoid unacceptable latency, in line with the need to compress DNNs.

C. Training Details and Sensitivity Analysis Configurations

We train the models on 4-second segments using the AMSGrad optimizer [33], with a minibatch size of 16. The learning rate is initialized to 0.001 and decays by 98% every two epochs. The mean squared error is used as objective function, which is an average over T-F units (for FDNN, LSTM and GCRN) or time samples (for TCNN). We use the validation set for both selecting the best model among different epochs and performing sensitivity analyses for pruning and quantization.

For unstructured pruning, the initial value of λ_1 (see Eq. (7)) 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. (9)) is set to 0.0005, 0.005, 0.02 and 0.05 for FDNN, LSTM, TCNN and GCRN, respectively. With these values, the orders of magnitude of $\mathcal{R}_{\ell_1}$ and $\mathcal{R}_{\ell_{2,1}}$ are almost the same, and one order of magnitude smaller than $\mathcal{L}$. Both λ_1 and λ_2 decay by 10% every pruning iteration. The tolerance values (α_1, α_2) for sensitivity analyses (see Algorithms 1, 2 and 3) are empirically set to (0.003, 0.0005), (0.03, 0.01), (0.0002, 0.00 005) and (0.02, 0.005) for FDNN, LSTM, TCNN and GCRN, respectively.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

A. Evaluation of the Proposed Compression Pipelines

Comprehensive comparisons between uncompressed and compressed models are shown in Table I. The subscript U indicates the uncompressed models, and C_1 and C_2 the compressed models by our proposed compression pipelines illustrated in Figs. 3(a) and 3(b), respectively. The STOI and PESQ scores represent the averages over the test examples in each test condition. We observe that the proposed compression pipelines result in slight or no performance degradation for all four models, in terms of STOI and PESQ. Take, for example, the LSTM model. The two pipelines compress the the LSTM model size from 115.27 MB to 2.49 MB and 9.97 MB, corresponding to compression rates of 46× and 12×, respectively. Note that both LSTM_{C_1} and LSTM_{C_2} produce similar STOI and PESQ to LSTM_U for all the three SNRs.

TABLE II
AVERAGE STOI AND PESQ RESULTS PRODUCED BY UNCOMPRESSED AND COMPRESSED MODELS ON FOUR ADDITIONAL NOISES

Metric	STOI (%)			PESQ		
SNR	-5 dB	0 dB	5 dB	-5 dB	0 dB	5 dB
Mixture	74.80	84.19	90.98	1.93	2.27	2.63
FDNN_U	80.94	88.52	93.03	2.24	2.63	2.97
FDNN_{C_1}	80.27	87.96	92.53	2.21	2.60	2.95
FDNN_{C_2}	79.65	87.55	92.25	2.16	2.56	2.91
LSTM_U	87.46	92.60	95.44	2.59	2.94	3.24
LSTM_{C_1}	87.73	92.83	95.67	2.59	2.95	3.26
LSTM_{C_2}	87.77	92.72	95.55	2.60	2.95	3.25
TCNN_U	90.11	94.22	96.26	2.53	2.89	3.16
TCNN_{C_1}	89.04	93.50	95.72	2.48	2.82	3.09
TCNN_{C_2}	89.13	93.60	95.77	2.51	2.86	3.13
GCRN_U	90.87	94.76	96.69	2.74	3.07	3.33
GCRN_{C_1}	90.43	94.51	96.53	2.71	3.07	3.33
GCRN_{C_2}	90.90	94.73	96.63	2.76	3.10	3.37

The effectiveness of the compression pipelines is further demonstrated by the results in Table II, in which four additional noises from the Diverse Environments Multichannel Acoustic Noise Database (DEMAND) [42] are used for testing. The four noises were recorded in four different environments, i.e. a city park ("NPARK"), a subway station ("PSTATION"), a meeting room ("OMEETING") and a public town square ("SPSQUARE"). The STOI and PESQ scores in Table II represent the averages over the four noises. We can see that our approach induces slight or no degradation in the model performance on these noises.

In addition, Table I suggests that C_1 achieves higher compression rates than C_2 for FDNN, LSTM and GCRN. It is likely because unstructured pruning uses smaller pruning granularity than structured pruning, which allows for less regular sparsity patterns and higher sparsity in weight tensors. Hence unstructured pruning is less constrained than structured pruning, typically leading to higher pruning ratios. For TCNN, the two pipelines yield similar compression rates. An interpretation is that structured pruning can achieve similar compression ratios to unstructured pruning for fully convolutional neural networks, consistent with [30].

TABLE III
NUMBER OF MAC OPERATIONS FOR UNCOMPRESSED AND COMPRESSED MODELS TO PROCESS A 4-SECOND NOISY MIXTURE. "PERCENT" DENOTES THE PERCENT OF THE ORIGINAL NUMBER OF MAC OPERATIONS

Model	# MACs	Percent
FDNN_U	3.63 G	100.00%
FDNN_{C_1}	0.05 G	1.28%
FDNN_{C_2}	0.35 G	9.65%
LSTM_U	12.13 G	100.00%
LSTM_{C_1}	1.20 G	9.91%
LSTM_{C_2}	4.79 G	39.50%
TCNN_U	4.80 G	100.00%
TCNN_{C_1}	1.76 G	36.57%
TCNN_{C_2}	2.57 G	53.43%
GCRN_U	9.61 G	100.00%
GCRN_{C_1}	2.36 G	24.58%
GCRN_{C_2}	5.24 G	54.57%

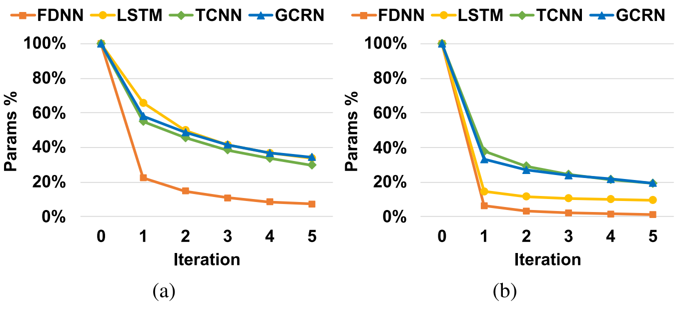


Fig. 4. (Color Online). The percent of the original number of trainable parameters at different pruning iterations. (a). Without, and (b). With sparse regularization. Note that unstructured pruning is performed.

Table III presents the number of multiply-accumulate (MAC) operations for uncompressed and compressed models to process a 4-second noisy mixture. We can observe that our approach significantly reduces the number of MAC operations for all the four models, demonstrating that the computational complexity is also reduced by the proposed compression pipelines.

B. Effects of Sparse Regularization and Iterative Pruning

We now investigate the effects of sparse regularization and iterative pruning. Fig. 4 presents the percent of the original number of trainable parameters, with or without ℓ_1 regularization (see Eq. (7)) for unstructured pruning. As shown in Fig. 4, the models can be incrementally compressed through iterative pruning. For example, the percent of the original number of trainable parameters in TCNN decreases to 55% after one pruning iteration and to 30% after five pruning iterations, without sparse regularization.

Moreover, it can be observed that the use of sparse regularization results in higher compression rates for all the four models. For example, the compression rate achieved by pruning GCRN for five iterations can be increased from 2.9× to 5.1× by applying ℓ_1 regularization. The corresponding STOI and PESQ results at -5 dB SNR are shown in Fig. 5, which suggests that our proposed pruning method does not significantly degrade the enhancement performance. To further investigate the effects of sparse regularization on pruning, we show the pruning ratios for different layers in FDNN after one pruning iteration in Fig. 6. We can see that sparse regularization increases the pruning ratios for all FDNN layers. These induced increases are significantly larger for structured pruning than unstructured pruning, which further demonstrates the effectiveness of group sparse regularization upon structured pruning.

We additionally train four relatively small DNNs, i.e. FDNN_S, LSTM_S, TCNN_S and GCRN_S. All of them have the same structures as the FDNN, LSTM, TCNN and GCRN described in Section III-B, except that the layer widths or the network depths are reduced. Specifically, the number of units in each hidden layer of FDNN_S and LSTM_S is reset to 200 and 320, respectively. For TCNN_S, the number of output channels in the middle layer of each residual block is reduced from 512 to 256, and the number of dilation blocks from 3 to 2. For GCRN_S, the number of output channels is reset to 64 and 128 for the fourth and the fifth gated blocks in the encoder, respectively. The number of output channels in the first gated block in each decoder is reduced from 128 to 64. We make these adjustments such that FDNN_S, LSTM_S, TCNN_S and GCRN_S have comparable model sizes to the original FDNN, LSTM, TCNN and GCRN pruned for 5, 5, 3 and 5 iterations, respectively. We denote these pruned models as FDNN_P, LSTM_P, TCNN_P and GCRN_P. Table IV compares the STOI and PESQ results produced by these models. We observe that FDNN_P, LSTM_P, TCNN_P and GCRN_P produce significantly higher STOI and PESQ than FDNN_S, LSTM_S, TCNN_S and GCRN_S, respectively. This demonstrates the advantage of training and pruning a large redundant DNN over directly training a relatively small DNN, consistent with [13], [24], [28], [51].

We now compare our proposed pruning method based on per-tensor sensitivity analyses with a method that uses a common threshold to determine the weights to prune for all weight tensors in a DNN. Such a strategy was adopted in many existing methods (e.g. [50]). Specifically, we compare the STOI and PESQ scores produced by two different pruned GCRNs. One is unstructurally pruned based on the results of Algorithm 1, denoted as GCRN_{P_1}. The other (denoted as GCRN_{P_2}) is pruned by removing the weights with absolute values smaller than a threshold, which is the same for all weight tensors. The value of this threshold is carefully selected such that GCRN_{P_2} has the exactly same compression rate as GCRN_{P_1}. Both GCRNs are pruned for only one iteration, and then fine-tuned. We use different values (0.02, 0.04, 0.08, 0.16 and 0.32) of the tolerance α_1 to obtain different compression rates. The STOI and PESQ results are shown in Fig. 7, and they suggest that our proposed approach yields higher STOI and PESQ. This demonstrates the advantage of per-tensor sensitivity analyses over the alternative method that uses a common pruning threshold.

C. Effects of Clustering-Based Quantization

To investigate the effects of clustering-based quantization, we directly quantize the weights of the original uncompressed models (without pruning), which amounts to four quantized models, i.e. FDNN_Q, LSTM_Q, TCNN_Q and GCRN_Q. The comparison between uncompressed and quantized models is shown in Table V. It can be seen that our proposed quantization method substantially reduces the model sizes without degrading the

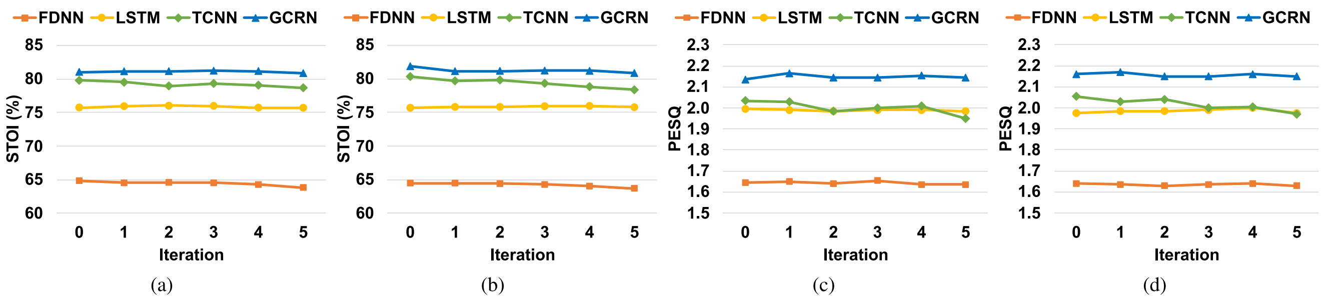


Fig. 5. (Color Online). STOI and PESQ scores for -5 dB SNR at different pruning iterations. (a)&(c). Without, and (b)&(d). With sparse regularization. Note that unstructured pruning is performed.

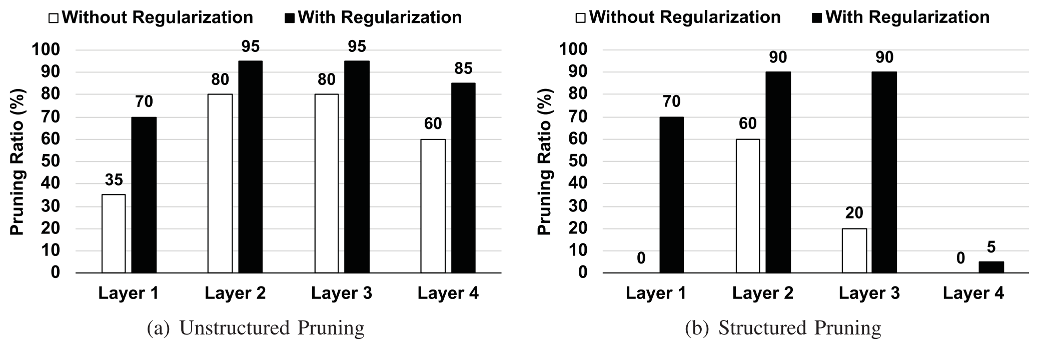


Fig. 6. Pruning ratios for different layers in FDNN after one pruning iteration. We apply ℓ_1 regularization for unstructured pruning and group sparse regularization for structured pruning.

TABLE IV
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		
Noise	BAB	CAF	BAB	CAF	BAB	CAF	BAB	CAF	BAB	CAF	BAB	CAF	-
Mixture	58.49	57.22	78.44	78.83	87.21	87.29	1.56	1.46	1.82	1.77	2.12	2.12	-
FDNN$_P$	63.41	64.52	77.16	78.10	86.44	86.74	1.60	1.66	2.01	2.10	2.39	2.49	1.15 M
FDNN$_S$	61.47	62.31	75.88	76.61	85.38	85.70	1.57	1.58	1.94	2.01	2.30	2.40	1.45 M
LSTM$_P$	76.73	74.79	87.32	85.91	92.60	91.90	1.97	2.00	2.47	2.44	2.86	2.83	2.93 M
LSTM$_S$	73.06	71.43	84.67	83.28	90.66	89.96	1.83	1.87	2.32	2.30	2.69	2.66	3.14 M
TCNN$_P$	80.93	77.71	90.12	88.59	94.10	93.34	2.03	1.97	2.52	2.45	2.88	2.82	1.24 M
TCNN$_S$	78.78	75.94	88.95	87.58	93.19	92.55	1.91	1.85	2.44	2.36	2.82	2.76	1.85 M
GCRN$_P$	82.97	80.00	91.37	89.81	94.88	94.17	2.20	2.09	2.71	2.59	3.06	2.97	1.91 M
GCRN$_S$	80.81	78.27	90.13	88.88	94.30	93.71	2.10	2.06	2.62	2.55	3.00	2.95	2.61 M

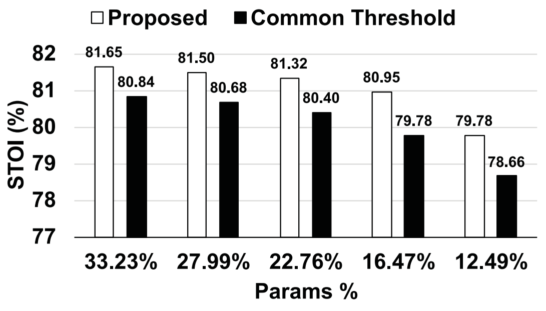
TABLE V
COMPARISONS BETWEEN UNCOMPRESSED AND QUANTIZED MODELS

Metric	STOI (%)						PESQ						Model Size	Compression Rate
SNR	-5 dB		0 dB		5 dB		-5 dB		0 dB		5 dB			
Noise	BAB	CAF	BAB	CAF	BAB	CAF	BAB	CAF	BAB	CAF	BAB	CAF		
Mixture	58.49	57.22	78.44	78.83	87.21	87.29	1.56	1.46	1.82	1.77	2.12	2.12	-	-
FDNN$_U$	64.35	65.38	78.44	78.83	87.21	87.29	1.60	1.69	2.04	2.13	2.43	2.51	34.54 MB	1×
FDNN$_Q$	64.27	65.04	78.09	78.65	87.04	86.99	1.61	1.68	2.03	2.13	2.43	2.51	5.50 MB	6×
LSTM$_U$	76.39	75.09	87.10	85.83	92.34	91.73	1.98	2.01	2.48	2.45	2.86	2.82	115.27 MB	1×
LSTM$_Q$	76.43	75.02	86.97	85.81	92.25	91.64	1.98	2.01	2.48	2.45	2.86	2.82	21.42 MB	5×
TCNN$_U$	81.08	78.44	90.51	88.92	94.31	93.60	2.06	2.01	2.56	2.47	2.90	2.82	19.28 MB	1×
TCNN$_Q$	80.44	77.35	90.07	88.47	94.09	93.29	2.05	2.00	2.54	2.46	2.90	2.81	2.84 MB	7×
GCRN$_U$	82.38	79.68	91.16	89.70	94.74	94.12	2.17	2.10	2.70	2.59	3.05	2.97	37.27 MB	1×
GCRN$_Q$	82.12	79.19	90.96	89.34	94.62	93.89	2.19	2.10	2.70	2.59	3.06	2.98	1.11 MB	34×

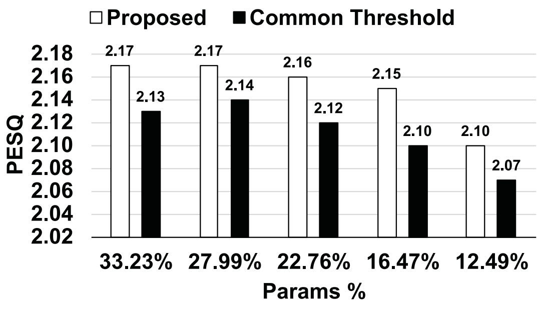
enhancement performance. For example, the differences between STOI and PESQ scores produced by LSTM$_U$ and LSTM$_Q$ are smaller than 0.2% and 0.01, respectively, for all the three SNRs. Through clustering-based quantization, the LSTM model is compressed from 115.27 MB to 21.42 MB, corresponding to a compression rate of 5×.

D. Evaluation on Speaker Separation

This section evaluates the proposed compression pipelines on multi-talker speaker separation. Specifically, we select TasNet [29] and an LSTM model based on utterance-level permutation invariant training (uPIT) [21] as representative talker-independent separation methods to apply our compression. We



(a)



(b)

Fig. 7. Comparison between the proposed pruning method and a method that uses a common pruning threshold for all weight tensors.

TABLE VI
COMPARISONS BETWEEN UNCOMPRESSED AND COMPRESSED MODELS FOR TALKER-INDEPENDENT SPEAKER SEPARATION

Metric	ESTOI (%)	PESQ	SI-SNR (dB)	SDR (dB)	Model Size	Compression Rate	# Pruning Iterations
Mixture	56.22	2.02	0.00	0.15	-	-	-
uPIT-LSTM$_U$	71.21	2.41	7.22	7.84	250.46 MB	1×	-
uPIT-LSTM$_{C_1}$	72.19	2.45	7.34	7.96	2.50 MB	100×	3
uPIT-LSTM$_{C_2}$	72.37	2.45	7.36	7.98	16.43 MB	15×	5
TasNet$_U$	81.61	2.71	10.08	10.57	19.27 MB	1×	-
TasNet$_{C_1}$	79.83	2.68	9.89	10.10	0.66 MB	29×	2
TasNet$_{C_2}$	79.52	2.68	9.78	10.00	0.77 MB	25×	1

use the same causal network configurations for both TasNet and uPIT-LSTM as in [29] and [21], respectively. The models are evaluated on the widely-used WSJ0-2mix dataset [8], [14], which contains 20 000, 5000 and 3000 mixtures in the training, validation and test sets, respectively. The sampling frequency is set to 8 kHz as in [21] and [29]. Following [26], we use extended short-time objective intelligibility (ESTOI) [20], PESQ, SI-SNR [29] and signal-to-distortion ratio (SDR) [44], to measure speaker separation performance. Other configurations are the same as Section III-C.

The speaker separation results are presented in Table VI, in terms of the four metrics. We can see that our proposed approach significantly compresses both models while maintaining the separation performance. For example, pipeline C_1 compresses the LSTM model from 250.46 MB to 2.50 MB, without reduction in any of the four performance metrics. This further demonstrates the effectiveness of our approach on speech separation models. In addition, pipeline C_1 yields higher compression rates than pipeline C_2 for uPIT-LSTM, while the two pipelines achieve comparable compression rates for TasNet, which is a fully convolutional neural network. This is consistent with our findings for compressing speech enhancement models (see Section IV-A).

V. CONCLUSION

In this study, we have proposed two new pipelines to compress DNNs for speech enhancement. The proposed pipelines incorporate three different techniques: sparse regularization, iterative pruning and clustering-based quantization. We systematically investigate these techniques on different types of speech enhancement models. Our experimental results show that the proposed pipelines substantially reduce the sizes of four different DNNs for speech enhancement, without significant performance degradation. In addition, structured pruning yields similar compression rates to unstructured pruning for fully convolutional neural networks, while unstructured pruning achieves significantly higher compression rates for other types of DNNs. We also find that training and pruning an over-parameterized DNN achieves better enhancement results than directly training a small DNN that has a comparable size to the pruned DNN. Moreover, our approach works well on two representative speaker separation models, which further suggests the capacity of our pipelines for compressing speech separation models.

ACKNOWLEDGMENT

The authors would like to thank Ashutosh Pandey for providing his implementation of TCNN.

REFERENCES

[1] L. J. Ba and R. Caruana, “Do deep nets really need to be deep?,” in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, vol. 2, 2014, pp. 2654–2662,.
[2] Y. Chebotar and A. Waters, “Distilling knowledge from ensembles of neural networks for speech recognition,” in *Proc. Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, 2016, pp. 3439–3443.
[3] Y. Chen, T. Guan, and C. Wang, “Approximate nearest neighbor search by residual vector quantization,” *Sensors*, vol. 10, no. 12, pp. 11259–11273, 2010.
[4] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, “Model compression and hardware acceleration for neural networks: A comprehensive survey,” *Proc. IEEE IRE*, vol. 108, no. 4, pp. 485–532, Apr. 2020.
[5] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, vol. 1, 2014, pp. 1269–1277,.
[6] I. Fedorov *et al.*, “TinyLSTMs: Efficient neural speech enhancement for hearing aids,” in *INTERSPEECH*, 2020, pp. 4054–4058.
[7] J. Friedman, T. Hastie, and R. Tibshirani, “A. note on the group lasso and a sparse group lasso,” 2010, *arXiv:1001.0736*.
[8] J. Garofolo, D. Graff, D. Paul, and D. Pallett, “CSR-I (WSJ0) Complete LDC93S6A,” *Web Download. Philadelphia: Linguistic Data Consortium*, vol. 83, 1993.
[9] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proc. 14th Int. Conf. Artif. Intell. Statist.. JMLR Workshop*, 2011, pp. 315–323.

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

[11] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” in *Proc. Int. Conf. Learn. Representations*, 2015.

[12] B. Hassibi and D. G. Stork, “Second order derivatives for network pruning: Optimal brain surgeon,” in *Proc. Adv. Neural Inf. Process. Syst.*, 1993, pp. 164–171.

[13] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 1389–1397.

[14] J. R. Hershey, Z. Chen, J. Le Roux, and S. Watanabe, “Deep clustering: Discriminative embeddings for segmentation and separation,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2016, pp. 31–35.

[15] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” in *Proc. Int. Conf. Neural Inf. Process. Syst. Deep Learn. Representation Learn. Workshop*, 2015.

[16] A. G. Howard *et al.*, “MobileNets: Efficient convolutional neural networks for mobile vision applications. 2017, *arXiv:1704.04861*.

[17] 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 <5 MB model size”. 2016, *arXiv:1602.07360*.

[18] B. Jacob *et al.*, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.

[19] H. Jegou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2010.

[20] J. Jensen and C. H. Taal, “An algorithm for predicting the intelligibility of speech masked by modulated noise maskers,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 24, no. 11, pp. 2009–2022, Nov. 2016.

[21] M. Kolbæk, D. Yu, Z.-H. Tan, and J. Jensen, “Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 25, no. 10, pp. 1901–1913, Oct. 2017.

[22] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” 2018, *arXiv:1806.08342*.

[23] Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 598–605.

[24] J. Lin, Y. Rao, J. Lu, and J. Zhou, “Runtime neural pruning,” in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2178–2188.

[25] 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.

[26] Y. Liu and D. L. Wang, “Divide and conquer: A deep CASA approach to talker-independent monaural speaker separation,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 27, no. 12, pp. 2092–2102, Dec. 2019.

[27] L. Lu, M. Guo, and S. Renals, “Knowledge distillation for small-footprint highway networks,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2017, pp. 4820–4824.

[28] J.-H. Luo, J. Wu, and W. Lin, “ThiNet: A filter level pruning method for deep neural network compression,” in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 5058–5066.

[29] Y. Luo and N. Mesgarani, “Conv-TasNet: Surpassing ideal time-frequency magnitude masking for speech separation,” *IEEE/ACM Trans Audio, Speech, Lang. Process.*, vol. 27, no. 8, pp. 1256–1266, Aug. 2019.

[30] H. Mao *et al.*, “Exploring the granularity of sparsity in convolutional neural networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2017, pp. 13–20.

[31] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, “Importance estimation for neural network pruning,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 11264–11272.

[32] A. Pandey and D. L. Wang, “TCNN: Temporal convolutional neural network for real-time speech enhancement in the time domain,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2019, pp. 6875–6879.

[33] S. J. Reddi, S. Kale, and S. Kumar, “On the convergence of adam and beyond,” in *Proc. Int. Conf. Learn. Representations*, 2018.

[34] R. Reed, “Pruning algorithms-a survey,” *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 740–747, Sep. 1993.

[35] 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 *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (Cat. No 01CH37221)*, vol. 2, 2001, pp. 749–752.

[36] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, “FitNets: Hints for thin deep nets,” in *Int. Conf. Learn. Representations*, 2015.

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

[38] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani, “A. sparse-group lasso,” *J. Comput. Graphical Statist.*, vol. 22, no. 2, pp. 231–245, 2013.

[39] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen, “An algorithm for intelligibility prediction of time-frequency weighted noisy speech,” *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 19, no. 7, pp. 2125–2136, Sep. 2011.

[40] K. Tan and D. L. Wang, “Learning complex spectral mapping with gated convolutional recurrent networks for monaural speech enhancement,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 28, pp. 380–390, 2020.

[41] K. Tan and D. L. Wang, “Compressing deep neural networks for efficient speech enhancement,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.* 2021, pp. 8358–8362.

[42] J. Thiemann, N. Ito, and E. Vincent, “The diverse environments multi-channel acoustic noise database: A database of multichannel environmental noise recordings,” *J. Acoust. Soc. Amer.*, vol. 133, no. 5, pp. 3591–3591, 2013.

[43] 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 Commun.*, vol. 12, no. 3, pp. 247–251, 1993.

[44] E. Vincent, R. Gribonval, and C. Févotte, “Performance measurement in blind audio source separation,” *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 14, no. 4, pp. 1462–1469, Jul. 2006.

[45] D. L. Wang, “On ideal binary mask as the computational goal of auditory scene analysis,” in P. Divenyi, ed., *Speech Separation by Humans Machines*. Springer, 2005, pp. 181–197.

[46] D. L. Wang and G. J. Brown, editors. *Computational Auditory Scene Analysis: Principles, Algorithms, and Applications*. Hoboken, NJ, USA, Wiley, 2006.

[47] D. L. Wang and J. Chen, “Supervised speech separation based on deep learning: An overview,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 26, no. 10, pp. 1702–1726, Oct. 2018.

[48] Y. Wang, A. Narayanan, and D. L. Wang, “On training targets for supervised speech separation,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 22, no. 12, pp. 1849–1858, Dec. 2014.

[49] 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 Process. Lett.*, vol. 26, no. 12, pp. 1887–1891, Dec. 2019.

[50] F. Ye, Y. Tsao, and F. Chen, “Subjective feedback-based neural network pruning for speech enhancement,” in *Proc. IEEE Asia-Pacific Signal Inf. Process. Assoc. Annu. Summit Conf.*, 2019, pp. 673–677.

[51] R. Yu *et al.*, “NISP: Pruning networks using neuron importance score propagation,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 9194–9203.

[52] X. Zhang, X. Zhou, M. Lin, and J. Sun, “ShuffleNet: An extremely efficient convolutional neural network for mobile devices,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.