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
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

\[
y = s + v,
\]

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

\[
z = F_1(y),
\]

\[
\hat{x} = H(z; \Theta),
\]

\[
\hat{s} = F_2(\hat{x}, y),
\]

where \( F_1 \) and \( F_2 \) denote transforms, and \( H \) the nonlinear mapping function represented by a DNN. For T-F domain enhancement, \( F_1 \) and \( F_2 \) can be short-time Fourier transform and waveform resynthesis, respectively. For time-domain enhancement, \( F_1 \) and \( F_2 \) can be segmentation and overlap-add, respectively. The symbol \( \Theta \) 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 \( \Theta \) are trained to minimize a loss function \( L(x, \hat{x}) = L(x, H(F_1(y); \Theta)) \), 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.

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

$$I_{U_l} = \mathcal{L}(\mathcal{V}, \Theta | g = 0, \forall g \in U) - \mathcal{L}(\mathcal{V}, \Theta),$$

where $U$ 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 $\Theta$ 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 $\{0\% , 5\% , 10\% , \ldots , 90\% , 95\% , 99\% , 100\% \}$ do
3: Let $U \subseteq \mathcal{W}_l$ be the set of the $\beta$\% of nonzero weights with the smallest absolute values in tensor $\mathbf{W}_l$;
4: $I_{U_l} \leftarrow \mathcal{L}(\mathcal{V}, \Theta | \mathbf{w} = 0, \forall \mathbf{w} \in U) - \mathcal{L}(\mathcal{V}, \Theta)$;
5: if $I_{U_l} > \alpha_1$ 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 $\mathbf{W}_l, \forall l$
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 \( \ell_1 \) regularization, which penalizes the sum of absolute values of weights during training. Specifically, \( \ell_1 \) 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 \( \ell_1 \) regularizer can be written as

\[
\mathcal{R}_{\ell_1} = \frac{\lambda_1}{n(W)} \sum_{w \in W} |w|,
\]

where \( W \) is the set of all nonzero weights, and \( \lambda_1 \) 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]:

\[
\mathcal{R}_{\ell_2,1} = \frac{\lambda_2}{n(G)} \sum_{g \in G} \sqrt{p_g} \| g \|_2^2,
\]

where \( G \) is the set of all weight groups, and \( \lambda_2 \) 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 \( \ell_1 \) regularization [37], [38]:

\[
\mathcal{R}_{\text{SGL}} = \mathcal{R}_{\ell_1} + \mathcal{R}_{\ell_2,1} = \frac{\lambda_1}{n(W)} \sum_{w \in W} |w| + \frac{\lambda_2}{n(G)} \sum_{g \in G} \sqrt{p_g} \| g \|_2^2.
\]

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:

\[
\arg \min_{S_1, S_2, \ldots, S_K} \sum_{k=1}^{K} \sum_{w \in S_k} |w - \mu_k|^2,
\]

where \( \mu_k \) 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 \( w_{\text{min}} \) and \( w_{\text{max}} \) 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 \( 32K \) additional bits. Hence, the compression rate for quantization is calculated as

\[
r = \frac{32N}{N \log_2 K + 32K},
\]

where \( N \) denotes the number of nonzero weights in the tensor.

Algorithm 3: Per-tensor sensitivity analysis for quantization

\begin{algorithm}
\begin{algorithmic}[1]
\State \textbf{Input:} (1) Validation set \( \mathcal{V} \); (2) set \( \mathcal{W}_l \) of all nonzero weights in the \( l \)-th weight tensor \( W_l \), \( \forall l \); (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 \( \alpha_2 \).
\State \textbf{Output:} Number of clusters \( K_l \) for weight tensor \( W_l \), \( \forall l \).
\For {each tensor \( W_l \)}
\State \( K \leftarrow 1 \);
\While {true}
\State \( \bar{\mathcal{I}}_K \leftarrow \mathcal{L}(\mathcal{V}, \Theta) \) quantize \( w \) to \( \log_2 K \) bits, \( \forall w \in \mathcal{W}_l \) \( - \mathcal{L}(\mathcal{V}, \Theta) \);
\If {\( \bar{\mathcal{I}}_K < \alpha_2 \) or \( 2K > \log_2(\mathcal{W}_l) \)}
\State \( K_l \leftarrow K \);
\State \break
\EndIf
\EndWhile
\State \( K \leftarrow 2K \);
\EndFor
\Return \( K_l \) for weight tensor \( W_l \), \( \forall l \)
\end{algorithmic}
\end{algorithm}

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
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 $\ell_1$ 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 × 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:

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

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 2048 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

<table>
<thead>
<tr>
<th>Noise</th>
<th>SNR</th>
<th>STOI (%)</th>
<th>PESQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-5 dB</td>
<td>0 dB</td>
<td>5 dB</td>
</tr>
<tr>
<td>Mixture</td>
<td>BAB</td>
<td>CAP</td>
<td>BAB</td>
</tr>
<tr>
<td>FDNNU</td>
<td>58.49</td>
<td>57.22</td>
<td>78.18</td>
</tr>
<tr>
<td>FDNNC1</td>
<td>58.35</td>
<td>57.31</td>
<td>78.24</td>
</tr>
<tr>
<td>FDNNC2</td>
<td>58.32</td>
<td>57.28</td>
<td>78.26</td>
</tr>
<tr>
<td>LSTMC1</td>
<td>76.79</td>
<td>75.09</td>
<td>78.10</td>
</tr>
<tr>
<td>LSTMC2</td>
<td>76.76</td>
<td>75.07</td>
<td>78.11</td>
</tr>
<tr>
<td>GCRNC1</td>
<td>77.52</td>
<td>75.91</td>
<td>78.46</td>
</tr>
<tr>
<td>GCRNC2</td>
<td>77.52</td>
<td>75.91</td>
<td>78.46</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Metric</th>
<th>SNR</th>
<th>STOI (%)</th>
<th>PESQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-5 dB</td>
<td>0 dB</td>
<td>5 dB</td>
</tr>
<tr>
<td>Mixture</td>
<td>74.80</td>
<td>74.19</td>
<td>90.98</td>
</tr>
<tr>
<td>FDNNU</td>
<td>80.94</td>
<td>80.52</td>
<td>93.03</td>
</tr>
<tr>
<td>FDNNC1</td>
<td>80.27</td>
<td>80.46</td>
<td>92.59</td>
</tr>
<tr>
<td>FDNNC2</td>
<td>79.65</td>
<td>79.35</td>
<td>92.55</td>
</tr>
<tr>
<td>LSTMC1</td>
<td>87.46</td>
<td>86.30</td>
<td>95.44</td>
</tr>
<tr>
<td>LSTMC2</td>
<td>87.73</td>
<td>86.30</td>
<td>95.67</td>
</tr>
<tr>
<td>GCRNC1</td>
<td>87.77</td>
<td>86.72</td>
<td>95.55</td>
</tr>
<tr>
<td>GCRNC2</td>
<td>87.90</td>
<td>86.76</td>
<td>96.69</td>
</tr>
<tr>
<td>GCRNC3</td>
<td>90.43</td>
<td>90.51</td>
<td>96.03</td>
</tr>
<tr>
<td>GCRNC4</td>
<td>90.90</td>
<td>90.73</td>
<td>96.63</td>
</tr>
</tbody>
</table>

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 C1 and C2 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 LSTM model size from 115.27 MB to 2.49 MB and 9.97 MB, corresponding to compression rates of $46 \times$ and $12 \times$, respectively. Note that both LSTMC1 and LSTMC2 provide similar STOI and PESQ levels as the LSTMU for all the three SNRs.

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 (“SP-SQUARE”). 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 C1 achieves higher compression rates than C2 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 presents the number of multiply-accumulate (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.

<table>
<thead>
<tr>
<th>Model</th>
<th># MACs</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDNN</td>
<td>3.63 G</td>
<td>100.00%</td>
</tr>
<tr>
<td>FDNNc1</td>
<td>0.05 G</td>
<td>1.28%</td>
</tr>
<tr>
<td>FDNNc2</td>
<td>0.35 G</td>
<td>9.65%</td>
</tr>
<tr>
<td>LSTM</td>
<td>12.13 G</td>
<td>100.00%</td>
</tr>
<tr>
<td>LSTMc1</td>
<td>1.20 G</td>
<td>9.91%</td>
</tr>
<tr>
<td>LSTMc2</td>
<td>4.79 G</td>
<td>39.50%</td>
</tr>
<tr>
<td>TCNN</td>
<td>4.80 G</td>
<td>100.00%</td>
</tr>
<tr>
<td>TCNnc1</td>
<td>1.76 G</td>
<td>36.57%</td>
</tr>
<tr>
<td>TCNnc2</td>
<td>2.57 G</td>
<td>53.43%</td>
</tr>
<tr>
<td>GCRN</td>
<td>9.61 G</td>
<td>100.00%</td>
</tr>
<tr>
<td>GCRNc1</td>
<td>2.36 G</td>
<td>24.58%</td>
</tr>
<tr>
<td>GCRNc2</td>
<td>5.24 G</td>
<td>54.57%</td>
</tr>
</tbody>
</table>

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 $\ell_1$ 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 $\ell_1$ 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. FDNNs, LSTM, TCNN and GCRNs. 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 FDNNs and LSTM is reset to 200 and 320, respectively. For TCNNs, 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 GCRNs, 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 FDNNs, LSTMs, TCNNs and GCRNs 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 FDNNp, LSTMp, TCNNp and GCRNp. Table IV compares the STOI and PESQ results produced by these models. We observe that FDNNp, LSTMp, TCNNp and GCRNp produce significantly higher STOI and PESQ than FDNNs, LSTMs, TCNNs and GCRNs, 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 GCRNp. The other (denoted as GCRNp*) 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 GCRNp has the exactly same compression rate as GCRNp. 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 $\alpha_1$ 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. FDNNQ, LSTMQ, TCNNQ and GCRNQ. 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
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 5x.

### 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

---

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 \(\ell_1\) regularization for unstructured pruning and group sparse regularization for structured pruning.

---

**TABLE IV**

| Traditional SI NER | 0 dB | 5 dB | 0 dB | 5 dB | # Param.
|--------------------|------|------|------|------|-----------
| **STOI (%)**      |      |      |      |      |           |
| Noise             | 0.55 | 0.75 | 0.75 | 0.55 |           |
| Mixture           | 0.55 | 0.75 | 0.75 | 0.55 |           |
| **PESQ**          |      |      |      |      |           |
| Noise             | 1.60 | 1.60 | 1.60 | 1.60 |           |
| Mixture           | 1.60 | 1.60 | 1.60 | 1.60 |           |

**TABLE V**

<table>
<thead>
<tr>
<th>Model Size</th>
<th>Compression Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>34.54 MB</td>
<td>1x</td>
</tr>
<tr>
<td>5.90 MB</td>
<td>6x</td>
</tr>
<tr>
<td>115.27 MB</td>
<td>1x</td>
</tr>
<tr>
<td>21.42 MB</td>
<td>5x</td>
</tr>
<tr>
<td>19.23 MB</td>
<td>1x</td>
</tr>
<tr>
<td>2.84 MB</td>
<td>7x</td>
</tr>
<tr>
<td>37.27 MB</td>
<td>1x</td>
</tr>
<tr>
<td>1.11 MB</td>
<td>34x</td>
</tr>
</tbody>
</table>
Fig. 7. Comparison between the proposed pruning method and a method that uses a common pruning threshold for all weight tensors.

TABLE VI

<table>
<thead>
<tr>
<th>Method</th>
<th>ESTOI (%)</th>
<th>PESQ</th>
<th>SI-SNR (dB)</th>
<th>SDR (dB)</th>
<th>Model Size</th>
<th>Compression Rate</th>
<th># Pruning Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixture</td>
<td>56.22</td>
<td>2.02</td>
<td>0.00</td>
<td>0.15</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>uPIT-LSTM</td>
<td>17.31</td>
<td>2.41</td>
<td>7.22</td>
<td>7.84</td>
<td>250.46 MB</td>
<td>1×</td>
<td>-</td>
</tr>
<tr>
<td>uPIT-LSTM_C1</td>
<td>72.19</td>
<td>2.45</td>
<td>7.34</td>
<td>7.96</td>
<td>2.50 MB</td>
<td>100×</td>
<td>3</td>
</tr>
<tr>
<td>uPIT-LSTM_C2</td>
<td>72.37</td>
<td>2.45</td>
<td>7.36</td>
<td>7.98</td>
<td>16.43 MB</td>
<td>15×</td>
<td>5</td>
</tr>
<tr>
<td>TasNet</td>
<td>81.61</td>
<td>2.71</td>
<td>10.08</td>
<td>10.57</td>
<td>19.27 MB</td>
<td>1×</td>
<td>-</td>
</tr>
<tr>
<td>TasNetC1</td>
<td>79.83</td>
<td>2.68</td>
<td>9.89</td>
<td>10.10</td>
<td>0.66 MB</td>
<td>29×</td>
<td>2</td>
</tr>
<tr>
<td>TasNetC2</td>
<td>79.52</td>
<td>2.68</td>
<td>9.78</td>
<td>10.00</td>
<td>0.77 MB</td>
<td>25×</td>
<td>1</td>
</tr>
</tbody>
</table>

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


