ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) | 978-1-7281-6327-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICASSP49357.2023.10095780

STRUCTURED PRUNING OF SELF-SUPERVISED PRE-TRAINED MODELS FOR SPEECH RECOGNITION AND UNDERSTANDING

Yifan Peng[2*], *Kwangyoun Kim*[1], *Felix Wu*[1], *Prashant Sridhar*[1], *Shinji Watanabe*[2]

[1]ASAPP Inc., Mountain View, CA, USA [2]Carnegie Mellon University, Pittsburgh, PA, USA

{yifanpen,swatanab}@andrew.cmu.edu {kkim,fwu,psridhar}@asapp.com

ABSTRACT

Self-supervised speech representation learning (SSL) has shown to be effective in various downstream tasks, but SSL models are usually large and slow. Model compression techniques such as pruning aim to reduce the model size and computation without degradation in accuracy. Prior studies focus on the pruning of Transformers; however, speech models not only utilize a stack of Transformer blocks, but also combine a frontend network based on multiple convolutional layers for low-level feature representation learning. This frontend has a small size but a heavy computational cost. In this work, we propose three task-specific structured pruning methods to deal with such heterogeneous networks. Experiments on LibriSpeech and SLURP show that the proposed method is more accurate than the original wav2vec2-base with 10% to 30% less computation, and is able to reduce the computation by 40% to 50% without any degradation.

Index Terms— Structured pruning, self-supervised models, speech recognition, spoken language understanding

1. INTRODUCTION

Self-supervised speech representation learning (SSL) has achieved great success in a variety of speech processing tasks [1, 2, 3, 4, 5, 6, 7]. However, SSL pre-trained models (e.g., wav2vec2 [8], HuBERT [9] and WavLM [10]) usually require large memory and high computational cost. Hence, it is difficult to deploy them in real-world applications. Recent studies have utilized model compression techniques to reduce the model size and computation without degradation in accuracy. One common approach is knowledge distillation [11], which trains a small student model with a pre-specified architecture to match the soft targets generated by a large pre-trained model. Distillation has shown to be effective in natural language processing (NLP) [12, 13] and speech processing [14, 15, 16, 17], but it usually performs general distillation using large amounts of unlabeled data before task-specific distillation or fine-tuning. This can make the training procedure computationally expensive.

Another compression technique is pruning, which extracts a compact and accurate subnetwork from the original model. Pruning has been widely used in computer vision (CV) [18, 19, 20, 21] and NLP [21, 22, 23, 24]. For speech models, [25, 26] prune recurrent neural networks (RNNs) for resource-limited applications. Another work [27] prunes deep neural networks (DNNs) based speech enhancement models using the sparse group lasso regularization [28]. These studies do not consider SSL models. PARP [29] is one of the first pruning methods designed for SSL speech models, which prunes individual weights based on magnitudes. Despite its good performance in low-resource automatic speech recognition (ASR), PARP is a type of unstructured pruning and thus cannot achieve an actual speedup without an efficient sparse matrix computation library, which is not usually available in many deployment scenarios. Another limitation is that PARP only prunes the Transformer layers while keeping the convolutional feature extractor (CNN) fixed. As discussed in [30], although the CNN has much fewer parameters than the Transformer, its computational cost is large and cannot simply be ignored. For example, in wav2vec2-base, the CNN has less than 5% of the total parameters but its computational cost is nearly 33% in terms of multiply-accumulate (MAC) operations for a 10-second audio.

In this work, we propose HJ-Pruning (Heterogeneous Joint Pruning) where both CNN and Transformer components are pruned jointly. We consider three variants: (a) *HJ-Pruning based on the overall model size* sets a single sparsity for the entire model size. (b) *HJ-Pruning based on separate model sizes* introduces a separate sparsity hyperparameter for each component which allows fine-grained tuning to find a trade-off between CNN and Transformer. (c) *HJ-Pruning based on the overall MACs* uses multiply–accumulate (MAC) operations as the sparsity criterion to find the best allocation of the computation budget across different components. We evaluate our methods in the ASR and spoken language understanding (SLU) tasks. Experiments on LibriSpeech and SLURP show that all HJ-Pruning methods outperform the previous Transformer-only pruning strategy. Our HJ-Pruning-MAC is more accurate than the original wav2vec2-base with 10% to 30% less computation, and is able to reduce the computation by 40% to 50% without any degradation.

2. BACKGROUND

2.1. Self-supervised pre-trained speech models

We evaluate the pruning algorithms mainly using wav2vec2 [8], but our proposed methods can be easily applied to other SSL models with as a similar architecture such as HuBERT [9] (see Sec. 4.5), SEW-D [30], and WavLM [10]. The wav2vec2-base model (pre-trained on Librispeech 960h [31]) consists of a convolutional feature extractor (CNN) and a Transformer [32] encoder. The CNN contains seven temporal convolutions with 512 channels and GeLU [33] activations. The Transformer encoder is a stack of 12 Transformer layers with a hidden dimension of 768 and 12 attention heads.

2.2. Structured pruning using L_0 regularization

We follow [22, 23, 34] to formulate the structured pruning task as a regularized learning problem, which aims to learn a sparse model. Let $f(\cdot;\boldsymbol{\theta})$ be a model with parameters $\boldsymbol{\theta} = \{\theta_j\}_{j=1}^{n}$, where each θ_j is a group of parameters (e.g., an attention head) and n is the number of groups. The pruning decisions are given by a set of binary

[*]Work done during an internship at ASAPP.

variables called *gates*: $\mathbf{z} = \{z_j\}_{j=1}^n$ where $z_j \in \{0, 1\}$. The model parameters after pruning are $\tilde{\boldsymbol{\theta}} = \{\tilde{\theta}_j\}_{j=1}^n$ such that $\tilde{\theta}_j = \theta_j z_j$. We usually sample gates from some distributions (e.g., Bernoulli) and update their parameters during training. Suppose the gates follow a distribution $q(\mathbf{z}; \boldsymbol{\alpha})$ with parameters $\boldsymbol{\alpha} = \{\alpha_j\}_{j=1}^n$, then our training objective is:

$$\min_{\boldsymbol{\theta}, \boldsymbol{\alpha}} \ \mathbb{E}_{\mathbf{z} \sim q} \left[\frac{1}{D} \sum_{i=1}^{D} \mathcal{L}(f(\mathbf{x}_i; \tilde{\boldsymbol{\theta}}), \mathbf{y}_i) + \lambda \|\tilde{\boldsymbol{\theta}}\|_0 \right], \tag{1}$$

where $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^D$ is the training data containing D samples, $\mathcal{L}$ is the training loss (i.e., CTC loss for ASR, cross entropy loss for SLU), and $\lambda > 0$ is a hyperparameter to control the sparsity. However, it is intractable to optimize Eq. (1) using gradient descent because the gates are discrete. Louizos et al. [34] propose a reparameterization trick to make the loss differentiable, which has been widely used in sparse model learning. Here we only introduce their final approach. Please refer to [34] for the derivation. Louizos et al. adopt the Hard Concrete Distribution [34] to model the gates $\mathbf{z}$:

$$\begin{gathered} \mathbf{u} \sim \mathcal{U}(0, 1), \ \ \mathbf{v}(\boldsymbol{\alpha}) = \sigma\left(\frac{1}{\beta}\left(\log \frac{\mathbf{u}}{1 - \mathbf{u}} + \log \boldsymbol{\alpha}\right)\right), \\ \bar{\mathbf{v}}(\boldsymbol{\alpha}) = (r - l) \cdot \mathbf{v}(\boldsymbol{\alpha}) + l, \ \ \mathbf{z} = \min(1, \max(0, \bar{\mathbf{v}}(\boldsymbol{\alpha}))), \end{gathered} \tag{2}$$

where $\mathcal{U}(0, 1)$ is a uniform distribution over the interval $[0, 1]$, $\sigma(\cdot)$ is the sigmoid function and β is a temperature constant. The actual parameters are $\boldsymbol{\alpha}$. $l < 0$ and $r > 0$ are two constants to stretch the output of sigmoid to $[l, r]$, which is finally rectified to $[0, 1]$. It is proven that the first term in Eq. (1) now becomes differentiable w.r.t. all parameters. We can write the second term in a closed-form expression based on the distribution of $\mathbf{z}$ shown in Eq. (2):

$$\mathbb{E}_{\mathbf{z}}\left[\|\tilde{\boldsymbol{\theta}}\|_0\right] = \sum_{j=1}^{n} P(z_j \neq 0) = \sum_{j=1}^{n} \sigma\left(\log \alpha_j - \beta \log \frac{-l}{r}\right), \tag{3}$$

which is also differentiable. $P(\cdot)$ denotes the probability.

Now we can train a sparse model using Eq. (1). However, it is difficult to exactly control the pruned model size [22, 23]. Instead of adding a regularizer $\lambda\|\tilde{\boldsymbol{\theta}}\|_0$, prior studies [22, 23] suggest optimizing the training loss subject to an explicit equality constraint on sparsity:

$$\min_{\boldsymbol{\theta}, \boldsymbol{\alpha}} \ \mathbb{E}_{\mathbf{z}} \left[\frac{1}{D} \sum_{i=1}^{D} \mathcal{L}(f(\mathbf{x}_i; \tilde{\boldsymbol{\theta}}), \mathbf{y}_i) \right] \ \text{s.t.} \ s(\boldsymbol{\alpha}) = t, \tag{4}$$

where $s(\boldsymbol{\alpha})$ is the current sparsity and t is a pre-specified target sparsity. The sparsity is defined as the percentage of parameters that are pruned. Similar to Eq. (3), given the current parameters $\boldsymbol{\alpha}$, we can calculate the expected number of nonzero gates in every module of the model. Recall that each gate is associated with a group of parameters. Hence, we know the expected number of parameters that are still kept, which further gives us the sparsity $s(\boldsymbol{\alpha})$. Eq. (4) can be rewritten as an adversarial game according to the augmented Lagrangian method [22]:

$$\max_{\boldsymbol{\lambda}} \min_{\boldsymbol{\theta}, \boldsymbol{\alpha}} \ \mathbb{E}_{\mathbf{z}} \left[\frac{1}{D} \sum_{i=1}^{D} \mathcal{L}(f(\mathbf{x}_i; \tilde{\boldsymbol{\theta}}), \mathbf{y}_i) \right] + g(\boldsymbol{\lambda}, \boldsymbol{\alpha}), \tag{5}$$

$$g(\boldsymbol{\lambda}, \boldsymbol{\alpha}) = \lambda_1 (s(\boldsymbol{\alpha}) - t) + \lambda_2 (s(\boldsymbol{\alpha}) - t)^2, \tag{6}$$

where $\lambda_1, \lambda_2 \in \mathbb{R}$ are two Lagrange multipliers that are jointly trained with other parameters. Once the game reaches equilibrium, the equality constraint will be satisfied. Hence, we can precisely control the sparsity of the pruned model. To facilitate training, we linearly increase the target sparsity t from zero to the desired value.

2.3. Structured pruning of Transformer layers

A Transformer [32] layer consists of a multi-head self-attention (MHA) block and a position-wise feed-forward network (FFN). We consider three pruning units, i.e., attention heads (12 per layer), intermediate size of FFN (3072 per layer), and the model's hidden size (768). We define a gate for each pruning unit. Given an input sequence $\mathbf{X} \in \mathbb{R}^{T \times d}$ of length T and feature size d, the MHA and FFN at a particular layer are the following:

$$\mathrm{MHA}(\mathbf{X}) = \sum_{k=1}^{h} (z_k^{\text{head}} \cdot \mathrm{ATT}(\mathbf{X}; \mathbf{W}_k^{\text{att}})), \tag{7}$$

$$\mathrm{FFN}(\mathbf{X}) = \mathrm{GeLU}(\mathbf{X}\mathbf{W}_1^{\text{ffn}}) \cdot \mathrm{diag}(\mathbf{z}^{\text{int}}) \cdot \mathbf{W}_2^{\text{ffn}}, \tag{8}$$

where $\mathrm{ATT}(\cdot; \mathbf{W}_k^{\text{att}})$ denotes the k-th attention head parameterized by $\mathbf{W}_k^{\text{att}}$, and z_k^{head} is a scalar gate. There are h heads in total. $\mathbf{z}^{\text{int}}$ is a d^{int}-dimensional gate for the FFN intermediate size. $\mathrm{diag}(\cdot)$ creates a diagonal matrix with its argument vector on the diagonal. GeLU is an activation [33]. FFN has two linear layers $\mathbf{W}_1^{\text{ffn}} \in \mathbb{R}^{d \times d^{\text{int}}}$, $\mathbf{W}_2^{\text{ffn}} \in \mathbb{R}^{d^{\text{int}} \times d}$. Each Transformer layer has its own gates and their parameters are independent. For the main hidden size, we define a gate $\mathbf{z}^{\text{hid}}$ of size d and share it across layers as in [23].

3. PROPOSED METHODS

3.1. Joint pruning based on the model size

As introduced in Sec. 1, the convolutional feature extractor (CNN) in SSL models is small in size but heavy in computation. To optimize the overall computation, we propose to jointly prune the CNN and Transformer. We have introduced the pruning units for Transformer in Sec. 2.3. For CNN, we prune convolution channels by introducing gate variables for every channel in every CNN layer, i.e., each output channel is multiplied with a gate. To train the model using Eq. (5), we need to define the model sparsity $s(\boldsymbol{\alpha})$. Our first proposed method is **HJ-Pruning-Size** (HJ-Pruning based on the overall model size), which can be viewed as a direct extension from prior work [22, 23]. Specifically, given the current distribution parameters $\boldsymbol{\alpha}$, we can calculate the probability of each gate being nonzero (i.e., the corresponding module is kept) as in Eq. (3). We then know the current sizes of all modules, including the model's hidden size, CNN channels, attention heads, and FFN intermediate sizes. Based on these sizes, we can compute the percentage of parameters that are pruned, which is the overall size sparsity $s_{\text{size}}^{\text{all}}(\boldsymbol{\alpha})$.

However, Sec. 4.2 shows that this approach does not work well in practice, because the CNN has much fewer parameters than the Transformer. If we simply set an overall sparsity, parameters will be pruned mainly from Transformer. To solve this problem, we propose the second method, i.e., **HJ-Pruning-SepSize** (HJ-Pruning based on separate model sizes). We calculate the size sparsity separately for CNN ($s_{\text{size}}^{\text{cnn}}(\boldsymbol{\alpha})$) and Transformer ($s_{\text{size}}^{\text{trans}}(\boldsymbol{\alpha})$). We also specify separate target sparsities $t_{\text{size}}^{\text{cnn}}, t_{\text{size}}^{\text{trans}}$ and extend Eq. (6) to have two terms:

$$\begin{aligned} g_{\text{size}} = {} & \lambda_1^{\text{cnn}} (s_{\text{size}}^{\text{cnn}}(\boldsymbol{\alpha}) - t_{\text{size}}^{\text{cnn}}) + \lambda_2^{\text{cnn}} (s_{\text{size}}^{\text{cnn}}(\boldsymbol{\alpha}) - t_{\text{size}}^{\text{cnn}})^2 \\ & + \lambda_1^{\text{trans}} (s_{\text{size}}^{\text{trans}}(\boldsymbol{\alpha}) - t_{\text{size}}^{\text{trans}}) + \lambda_2^{\text{trans}} (s_{\text{size}}^{\text{trans}}(\boldsymbol{\alpha}) - t_{\text{size}}^{\text{trans}})^2. \end{aligned} \tag{9}$$

As shown in Sec. 4.2, this method achieves strong performance. However, it requires careful tuning of the separate target sparsities. We always need to search over the two sparsities to meet a particular budget, which is computationally expensive.

3.2. Joint pruning based on the MACs

The third method we propose is **HJ-Pruning-MAC** (HJ-Pruning based on the overall MACs). Unlike prior methods, we prune the entire model to directly meet a computational budget measured by MACs. We follow the formulas used in the DeepSpeed flops profiler to calculate MACs. [1] For an input sequence of length T and hidden size d, the MACs for each MHA and FFN block are as follows:

$$\text{MAC}^{\text{mha}} = 4Thdd^{\text{head}} + 2T^2hd^{\text{head}}, \tag{10}$$

$$\text{MAC}^{\text{ffn}} = 2Tdd^{\text{int}}, \tag{11}$$

where h is the number of attention heads and d^{head} is the size per head. d^{int} denotes the intermediate size of FFN. The MACs of a 1-D convolution can be computed by

$$\text{MAC}^{\text{cnn}} = T^{\text{out}}C^{\text{out}}C^{\text{in}}K, \tag{12}$$

where T^{out} is the output length and K is the kernel size. C^{in} and C^{out} are the input and output channels, respectively. Note that $h, d, d^{\text{int}}, C^{\text{in}}, C^{\text{out}}$ are calculated from the current gate distributions (similar to Eq. (3)). They are differentiable functions of $\boldsymbol{\alpha}$. We define the percentage of MACs that are pruned as the MACs-based sparsity $s^{\text{all}}_{\text{macs}}(\boldsymbol{\alpha})$. [2] It is differentiable w.r.t. parameters $\boldsymbol{\alpha}$. Hence, we can still train the model using Eq. (5).

4. EXPERIMENTS

4.1. Experimental setup

We focus on task-specific structured pruning of SSL speech models. We mainly prune wav2vec2-base, but we also show that our methods can be directly applied to HuBERT-base in Sec. 4.5. We conduct experiments using PyTorch [35] and HuggingFace's transformers [36]. Our implementation of the basic pruning algorithm is based on prior work in NLP [23]. For each task, we add a linear layer on top of the pre-trained SSL model and fine-tune the entire model to obtain an unpruned model. Then, we prune this fine-tuned model to reach a specific sparsity using Eq. (5). We employ an AdamW optimizer and a linear learning rate scheduler for all experiments.

ASR: The 100-hour clean set of LibriSpeech [31] is utilized. In Sec. 4.3, the Tedlium [37] test set is used as out-of-domain data to demonstrate the robustness of structured pruning. The training loss is CTC [38]. We fine-tune a pre-trained model for 25 epochs and prune for 30 epochs with a learning rate of 1.5e-4 and a batch size of 64. The target sparsity is linearly increased to the desired value during the first 5 epochs. The learning rate of $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$ is selected from $\{0.02, 0.05\}$. The pruned model is fine-tuned for another 10 epochs with a learning rate of 5e-5. The learning rate warmup steps are 3k, 3k, and 1k for training, pruning, and fine-tuning, respectively.

SLU: The SLURP corpus [39] is used for intent classification. A pre-trained SSL model is fine-tuned for 50 epochs and pruned for 50 epochs with a learning rate of 1e-4 and a batch size of 80. The final fine-tuning has 10 epochs with a learning rate of 1e-5. The learning rate warmup is performed for 4k, 4k, 1k steps for training, pruning, and fine-tuning, respectively. Other configs are the same as ASR.

[1] `https://github.com/microsoft/DeepSpeed`

[2] The computation of MACs also depends on the sequence length T, because MHA has quadratic complexity w.r.t. T. We use 10 seconds to compute MACs in our experiments. This is a "virtual" length used only for computing MACs. We do not modify any training utterances.

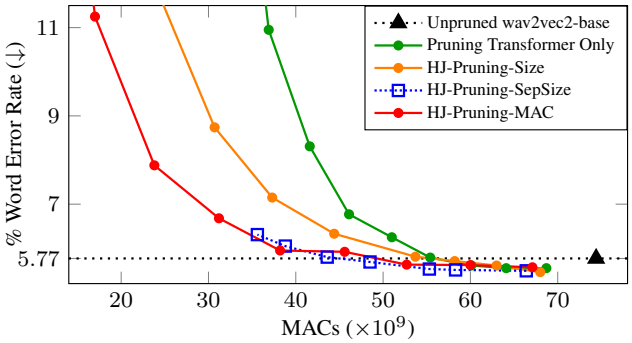


Fig. 1: Word Error Rate (%) vs. Multiply-Accumulate Operations on LibriSpeech test-clean. Our proposed HJ-Pruning methods consistently outperform the baseline.

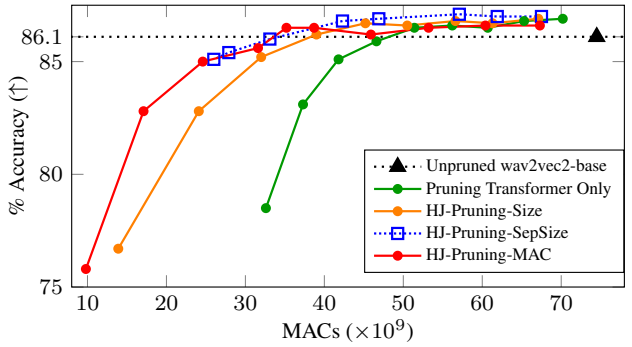
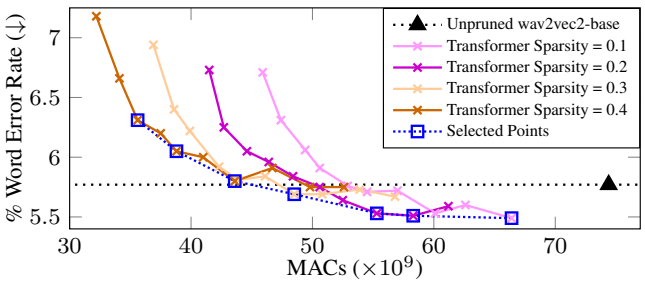


Fig. 2: Intent Classification Accuracy (%) vs. Multiply-Accumulate Operations on the SLURP test set. Our proposed HJ-Pruning methods consistently outperform the baseline.

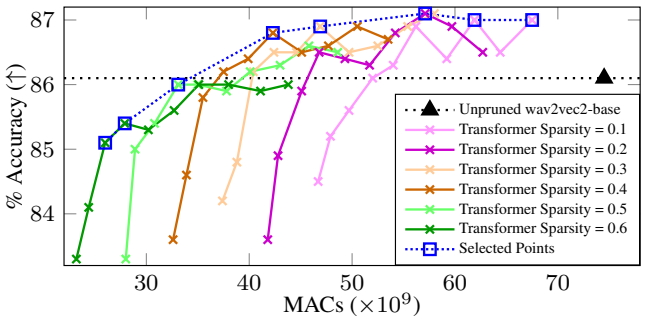
4.2. Main results

Fig. 1 compares various pruning methods for LibriSpeech ASR. The unpruned model has good performance (5.77% WER) but is computationally expensive (74.4 GMACs). At a low sparsity (>55 GMACs), all pruned models achieve similar WERs which are even better than the original result, because the pruning target can regularize the training. As the sparsity increases, the baseline method which only prunes Transformer drastically degrades. Our proposed three algorithms which jointly prune CNN and Transformer consistently outperform the baseline by a large margin. We can reduce over 40% of the total computation without degradation in WER. HJ-Pruning-MAC has similar performance with HJ-Pruning-SepSize, both outperforming HJ-Pruning-Size. This is because the CNN has much fewer parameters than Transformer. If we simply set an overall size sparsity, the pruned parameters are mainly from Transformer, while CNN still has high computational overhead. To prune them based on separate sizes (Eq. (9)), we have to search for the best combination of the two target sparsities. This model selection procedure is presented in Fig. 3a, where we perform a grid search and select the Pareto frontiers. This requires much more computation than the other methods. Hence, the HJ-Pruning-MAC is probably the best method in terms of performance and complexity.

Fig. 2 shows the results of intent classification on SLURP. The overall trend is very similar to that of ASR. Our joint pruning methods outperform the baseline by a large margin, especially at a high sparsity (low MACs). HJ-Pruning-SepSize is comparable with HJ-Pruning-MAC, but again, it requires a grid search over the two target sparsities as shown in Fig. 3b. This high complexity hinders its usage in practice. Compared to ASR, we can achieve a higher compression rate (over 55%) without loss in accuracy. This is probably because



(a) Word Error Rates (%) on LibriSpeech test-clean.



(b) Intent Classification Accuracy (%) on the SLURP test set.

Fig. 3: Model selection for HJ-Pruning-SepSize. As described in Sec. 3.1, we perform grid search over the Transformer sparsity (0.1 to 0.4/0.6) and CNN sparsity (0.1 to 0.95). The Pareto frontiers are shown in blue, which are also presented in Fig. 1 and Fig. 2.

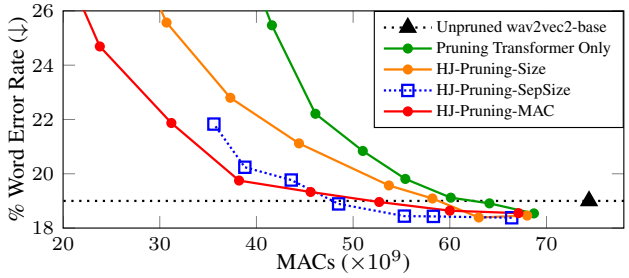


Fig. 4: Robustness analysis. All models are trained on LibriSpeech 100h and then directly evaluated on the **out-of-domain Tedlium test set**. The trend is similar to that of the in-domain evaluation in Fig. 1.

the classification task is easier and thus requires less information than the sequence-to-sequence task.

4.3. Robustness of structured pruning

To investigate the robustness of the proposed structured pruning methods, we test the ASR models using an *out-of-domain* corpus, TED-LIUM [37]. Note that these models are trained only with LibriSpeech data. As shown in Fig. 4, again, our joint pruning methods consistently outperform the baseline, and the trend is very similar to that of the in-domain evaluation (see Fig. 1). This demonstrates that our pruning methods are robust.

4.4. Architectures of pruned models

Fig. 5 shows the remaining CNN channels, attention heads and FFN intermediate sizes after HJ-Pruning-MAC. The target sparsity ranges from 10% to 40%. For CNN, the sequence length gradually decreases due to downsampling. The first few layers have higher computational cost, so they tend to be pruned more. For MHA and FFN,

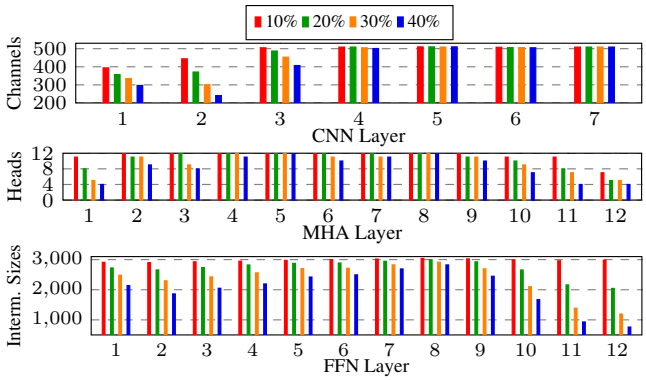


Fig. 5: ASR model architectures after HJ-Pruning-MAC. The target sparsity ranges from 10% to 40%.

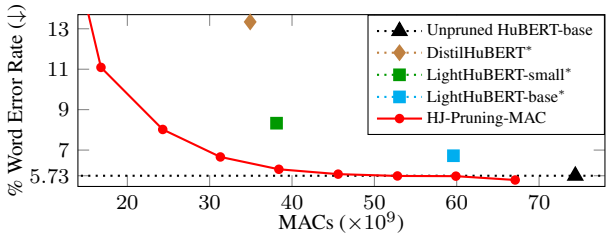


Fig. 6: Results of pruning HuBERT-base on LibriSpeech test-clean. * WERs from SUPERB [1]. See Sec. 4.5 for discussions.

the upper layers are pruned the most, indicating that upper layers are more redundant. Prior studies had similar observations by analyzing the self-attention patterns in speech encoders [40, 41, 42]. The overall trend is also consistent with a prior work in NLP [23].

4.5. Comparison with other compression methods

As introduced in Sec. 2.1, HJ-Pruning can be directly applied to other SSL models. In Fig. 6, we prune the HuBERT-base model based on the overall MACs for ASR. The performance is similar to that of the wav2vec2. We also include other compressed models for comparison, including DistilHuBERT [15] and LightHuBERT [16]. Note that these results are not really comparable due to: (1) Their WERs are from SUPERB [1], which combines a frozen SSL model with another learnable RNN. We also tried to replace the RNN with a single linear layer and fine-tune the entire model (same as our setting), but the performance was clearly worse. (2) Their compressed models are initially distilled using the 960h unlabeled LibriSpeech data and then fine-tuned on the 100h labeled data, but our task-specific pruning *only* utilizes the 100h data. This comparison shows that our task-specific pruning method is highly effective.

5. CONCLUSION

In this paper, we propose HJ-Pruning to jointly prune heterogeneous components of SSL speech models, which achieves strong performance-efficiency tradeoffs compared to several baselines. At a small sparsity (0.1 to 0.3), HJ-Pruning improves the wav2vec2 baseline while being faster. Depending on the task, HJ-Pruning saves 40% or 50% MACs while maintaining the performance of wav2vec2. HJ-Pruning is a general method that can be applied to most of speech SSL models such as HuBERT. In the future, we plan to explore the application of HJ-Pruning on encoder-decoder SSL models [43] and other SLU tasks [44, 5].

6. REFERENCES

[1] S. w. Yang, P.-H. Chi, Y.-S. Chuang, et al., "SUPERB: Speech Processing Universal PERformance Benchmark," in *Proc. Interspeech*, 2021.

[2] A. Mohamed, H.-y. Lee, L. Borgholt, et al., "Self-Supervised Speech Representation Learning: A Review," *arXiv:2205.10643*, 2022.

[3] X. Chang, T. Maekaku, P. Guo, et al., "An exploration of self-supervised pretrained representations for end-to-end speech recognition," in *Proc. ASRU*, 2021.

[4] Z. Huang, S. Watanabe, S.-w. Yang, et al., "Investigating Self-Supervised Learning for Speech Enhancement and Separation," in *Proc. ICASSP*, 2022.

[5] S. Shon, A. Pasad, F. Wu, et al., "SLUE: New Benchmark Tasks For Spoken Language Understanding Evaluation on Natural Speech," in *Proc. ICASSP*, 2022.

[6] S. Arora, S. Dalmia, P. Denisov, et al., "ESPnet-SLU: Advancing Spoken Language Understanding Through ESPnet," in *Proc. ICASSP*, 2022.

[7] Y. Peng, S. Arora, Y. Higuchi, et al., "A Study on the Integration of Pre-trained SSL, ASR, LM and SLU Models for Spoken Language Understanding," in *Proc. SLT*, 2022.

[8] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," in *Proc. NeurIPS*, 2020.

[9] W.-N. Hsu, B. Bolte, Y.-H. H. Tsai, et al., "HuBERT: Self-supervised speech representation learning by masked prediction of hidden units," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 29, pp. 3451–3460, 2021.

[10] S. Chen, C. Wang, Z. Chen, et al., "WavLM: Large-scale self-supervised pre-training for full stack speech processing," *IEEE Journal of Selected Topics in Signal Processing*, 2022.

[11] G. Hinton, O. Vinyals, J. Dean, et al., "Distilling the knowledge in a neural network," *arXiv:1503.02531*, 2015.

[12] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," *arXiv:1910.01108*, 2019.

[13] X. Jiao, Y. Yin, et al., "TinyBERT: Distilling BERT for natural language understanding," in *Findings of EMNLP*, 2020.

[14] Z. Peng, A. Budhkar, I. Tuil, et al., "Shrinking bigfoot: Reducing wav2vec 2.0 footprint," in *SustaiNLP*, 2021.

[15] H.-J. Chang, S.-w. Yang, and H.-y. Lee, "DistilHuBERT: Speech representation learning by layer-wise distillation of hidden-unit BERT," in *Proc. ICASSP*, 2022.

[16] R. Wang, Q. Bai, J. Ao, et al., "LightHuBERT: Lightweight and Configurable Speech Representation Learning with Once-for-All Hidden-Unit BERT," in *Proc. Interspeech*, 2022.

[17] Y. Lee, K. Jang, J. Goo, et al., "FitHuBERT: Going Thinner and Deeper for Knowledge Distillation of Speech Self-Supervised Models," in *Proc. Interspeech*, 2022.

[18] S. Han, J. Pool, et al., "Learning both weights and connections for efficient neural network," in *Proc. NeurIPS*, 2015.

[19] H. Li, A. Kadav, I. Durdanovic, et al., "Pruning Filters for Efficient ConvNets," in *Proc. ICLR*, 2017.

[20] Z. Liu, J. Li, Z. Shen, et al., "Learning Efficient Convolutional Networks Through Network Slimming," in *Proc. ICCV*, 2017.

[21] Q. Zhang, S. Zuo, C. Liang, et al., "PLATON: Pruning large transformer models with upper confidence bound of weight importance," in *Proc. ICML*, 2022.

[22] Z. Wang, J. Wohlwend, and T. Lei, "Structured Pruning of Large Language Models," in *Proc. EMNLP*, 2020.

[23] M. Xia, Z. Zhong, and D. Chen, "Structured Pruning Learns Compact and Accurate Models," in *Proc. ACL*, 2022.

[24] C. Liang, S. Zuo, M. Chen, et al., "Super tickets in pre-trained language models: From model compression to improving generalization," in *Proc. ACL*, 2021.

[25] P. Dong, S. Wang, W. Niu, et al., "Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition," in *ACM/IEEE Design Automation Conference (DAC)*, 2020.

[26] S. Wang, P. Lin, R. Hu, et al., "Acceleration of LSTM With Structured Pruning Method on FPGA," *IEEE Access*, 2019.

[27] K. Tan and D.L. Wang, "Compressing Deep Neural Networks for Efficient Speech Enhancement," in *Proc. ICASSP*, 2021.

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

[29] C.-I J. Lai, Y. Zhang, A. H. Liu, et al., "PARP: Prune, Adjust and Re-Prune for Self-Supervised Speech Recognition," in *Proc. NeurIPS*, 2021.

[30] F. Wu, K. Kim, J. Pan, et al., "Performance-Efficiency Trade-offs in Unsupervised Pre-training for Speech Recognition," in *Proc. ICASSP*, 2022.

[31] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. ICASSP*, 2015.

[32] A. Vaswani, N. Shazeer, N. Parmar, et al., "Attention is all you need," in *Proc. NeurIPS*, 2017.

[33] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," *arXiv:1606.08415*, 2016.

[34] C. Louizos, M. Welling, and D. P. Kingma, "Learning Sparse Neural Networks through L0 Regularization," in *ICLR*, 2018.

[35] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," *Proc. NeurIPS*, 2019.

[36] T. Wolf et al., "Huggingface's transformers: State-of-the-art natural language processing," *arXiv:1910.03771*, 2019.

[37] A. Rousseau et al., "TED-LIUM: an automatic speech recognition dedicated corpus.," in *Proc. LREC*, 2012.

[38] A. Graves, S. Fernández, F. Gomez, et al., "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proc. ICML*, 2006.

[39] E. Bastianelli, A. Vanzo, P. Swietojanski, and V. Rieser, "SLURP: A Spoken Language Understanding Resource Package," in *Proc. EMNLP*, 2020.

[40] S. Zhang, E. Loweimi, P. Bell, and S. Renals, "On the usefulness of self-attention for automatic speech recognition with transformers," in *Proc. SLT*, 2021.

[41] Y. Peng, S. Dalmia, I. Lane, and S. Watanabe, "Branchformer: Parallel MLP-attention architectures to capture local and global context for speech recognition and understanding," in *Proc. ICML*, 2022.

[42] T. Maekaku, Y. Fujita, Y. Peng, and S. Watanabe, "Attention Weight Smoothing Using Prior Distributions for Transformer-Based End-to-End ASR," in *Proc. Interspeech*, 2022.

[43] F. Wu, K. Kim, S. Watanabe, et al., "Wav2seq: Pre-training speech-to-text encoder-decoder models using pseudo languages," *arXiv:2205.01086*, 2022.

[44] L. Lugosch, M. Ravanelli, P. Ignoto, et al., "Speech model pre-training for end-to-end spoken language understanding," in *Interspeech*, 2019.