Real-time Speech Enhancement on Mobile Phones

William Zhao Mark Jabbour Ian Lee
MIT
wzhao6, mjabbour, ianl@mit.edu

Abstract

We present a monaural (single-microphone) speech enhancement tool for Android devices that can filter background and non-speaker noises in real-time on phone CPU. We improve on a previous model for speech enhancement on laptops. The original model uses an U-Net architecture with two LSTM layers. We tested and applied optimizations techniques including uniform pruning, sensitivity-aware pruning, and linear quantization to the model to minimize model size and latency.[1].[2] Our mobile-deployed speech enhancement tool is capable of filtering background noise and maintains similar performance as the original model, while achieving real-time performance on mobile platforms. We evaluate our models' performances on two standard metrics for speech quality and report their real-time-factors (RTF) on an Android device. Lastly, we publish our proposed speech denoiser model for Android 6.0 and above, and open-source all of our research and deployment code. [3]. Finally, we publish a demo of our Android app. [4]

1 Introduction

With the advent of work virtualization and telecommunication, millions are now dependent on virtual call and conferencing tools for their day-to-day work. Maximizing speech quality, or the task of speech enhancement, has thus become a topic of paramount interest for most, from regular phone and virtual conferencing software users, patients relying on hearing aids, to researchers interested in automated speech recognition (ASR) (1). Normally, the goal of speech enhancement is to filter out interfering background noise, such as vehicles passing by, animal barking, or babbled noise from a crowd of speakers, with the additional expectation of performing noise filtering with minimal lag, to reach the effect of real-time enhanced audio.

To achieve this, deep learning approaches are preferred over traditional denoising methods due to their superior performance in generating enhanced audio with overall higher audio quality, especially when non-Gaussian noise is present (2; 3). Given the need for low-latency speech enhancement and the promising nature of deep learning methods, we developed a real-time speech enhancement mobile app, which relies on an optimized version of the labtop-CPU-enabled DEMUCS model (4). To enable deployment on mobile platform, we applied sensitivity-based pruning and quantization to DEMUCS to minimize the size burden of the model while preserving the overall model architecture.

In evaluating performance, two standard objective metrics are employed to measure the enhancement quality generated from our optimized DEMUCS model. We performed experiments on both labtop CPU and physical Android devices for testing latency of the deployed model. Our results suggest

[1]Code for the optimized model is available at `https://github.com/uraniumCoder/denoiser`

[2]More code is available at `https://drive.google.com/drive/folders/1vZNuj1mNISckZ4f782tysrtEfsdj7-GW?usp=sharing`

[3]Code for deployed app is available at `https://github.com/ianlee-tyl/noiseOut`

[4]A demo of the Android app is available at `https://youtube.com/shorts/egA3BmFw00E?feature=share`

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

that the optimized model extends the possibility of deploying real-time speech enhancement software on end devices, specifically Android mobile phones.

2 Background

2.1 Problem setting

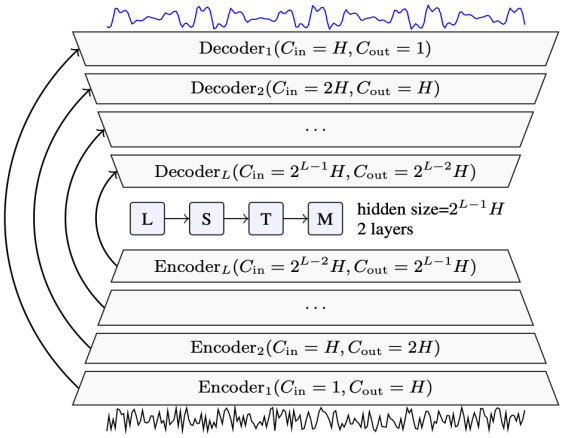

Figure 1: Overview of DEMUCS architecture: with U-net skip connections and encoder-decoder structure. Input noisy audio signal is converted to cleaned audio signal on the top.

In the original model, the goal is to find an enhancement function f such that $f(\boldsymbol{x}) \approx \boldsymbol{y}$, where $\boldsymbol{x}$ is an audio signal corrupted by a background signal $\boldsymbol{n}$, and $\boldsymbol{y}$ is the corresponding cleaned audio signal after processing (4). Specifically, $\boldsymbol{x} = \boldsymbol{y} + \boldsymbol{n}$. We set the enhancement function f to be the DEMUCS model similar to the original model, but with optimizations applied.

2.2 DEMUCS architecture and objective

A visual description of the DEMUCS architecture used in (5) is shown in Figure 1. It consists of an encoder and decoder made of 1d convolution layers. These convolution layers are connected with skip connections as in U-Net. Between its encoder and decoder is a 2-layer unidirectional LSTM layer to process the latent audio encoding. In this work, we improve on the smallest pretrained DEMUCS model provided by (4), which contains 5 layers of convolutions with kernel size 7 and stride 4, and 48 channels in the first layer. It also up-samples the audio 4-fold with sinc interpolation as a prepossessing step, and down-samples the outputs correspondingly. Both the input and output of this model uses a single channel for monaural (single-microphone) audio.

For the training objective function, the original model uses L1 loss with a multi-resolution STFT loss. We adhere to the original model and optimizes using the same objective function as in (4).

2.3 DEMUCS evaluation

In the original model, both objective and subjective metrics were used to evaluate performance. In particular, the objective measures we followed are: (i) PESQ: Perceptual evaluation of speech quality, which ranges from 0.5 to 4.5, with higher value as a better score, and (ii) Short-Time Objective Intelligibility (STOI), from 0 to 100 due to their wide acceptance for objective audio quality measurement.

3 Methods

3.1 Profiling

Our model parameters total 72 MB in size, which would likely exhaust an older phone's memory hierarchy. When profiled on an Intel(R) Xeon(R) CPU while denoising a 15s audio sample, outr model spends around 60% of its time applying convolutions, and 30% applying LSTMs. This is likey due to the deep encoder decoder structure. We further profiled the model for longer (60s) and shorter (40ms) audio samples, and there were no significant changes in the latency ratios. As a result, we prioritize reducing our model size and speeding up the convolutions layers.

3.2 Pruning

We prune our model to decrease the latency and model size convolution layers. We use channel-based pruning in order to take advantage of existing efficient hardware acceleration for convolutional layers on mobile devices.

Unlike previous work which focused on pruning models for classification tasks, our model architecture contains skip connections due to the due to the U-architecture. We analyzed the connections around each skip connections in Figure 2. As shown, the skip connections are input to two convolutions layer (1 encoder and 1 decoder) and combines outputs of two layers as well. As we're using importance pruning on input channels, we need to balance the magnitude information from both output convolution layers.

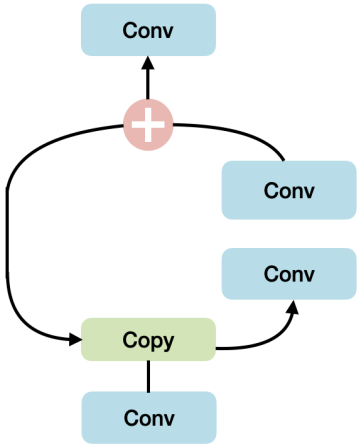

Figure 2: Overview of the U-arch skip connections that lend itself useful for optimization.

Recall that the importance of a channel is usually given by its L_2 norm $I_c = \sqrt{\sum_c w^2}$ (8). We tested two methods to combine the importance of a channel when it is an input to two layers.

1. **Averaging:** Average the importance of the channel in both tensors.

$$I = \frac{I_{encoder} + I_{decoder}}{2}$$

2. **Normalization:** We observe that the DEMUCS architecture contains no batch-normalization layers, so the magnitude of the model parameters are not standardized. To resolve this, we normalize the importance of each convolution layer by its L_2 norm, then average the normalized importance to get a more accurate measure.

$$I = \frac{\frac{I_{encoder}}{|I_{encoder}|} + \frac{I_{decoder}}{|I_{decoder}|}}{2}$$

After pruning, we fine-tune the remaining weights using 10 hours of the DNS dataset (6).

3.2.1 Pruning Sensitivity Scan

We compared two different methods for choosing which layers to prune. First, we considered pruning all channels uniformly, which lead to worse performance. We then conducted a sensitivity scan over all layers and used the PESQ

3.3 K-means Quantization

K-means quantization works by clustering the weights of each tensor into 2^b clusters using k-means clustering, where b is the desired number of bits. The parameter weights are replaced by their cluster id and is accompanied by a codebook mapping each cluster's id to its centroid. While k-means quantization significantly decreases our model size (9), it's impact on latency is less significant since the MAC operations must still be done with floats, which is must slower than int8 operations for phone CPUs.

3.4 Linear Quantization

Linear Quantization replaces floating point arithmetic with integer arithmetic. In general, integer arithmetic tend to be faster and have less memory footprint floating point arithmetic on edge devices such as mobile phones (10).

Recall that linear quantization associates each tensor with numbers z and s representing its mean and range. Then each real number r in this tensor is approximated with $s(q - z)$ where q is an integer. In particular $s = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$, and $z = \frac{r_{\max}}{s} - q_{\max}$. We use $q_{\max} = 127$ and $q_{\min} = -128$ for int8 quantization (10).

While the above can always be applied on weights, applying it on activation requires calibration the range of the activation. The most effecient way to do this, would be to make the ranges static and learn them from samples as done in static quantization. We decided not to pursue this experiment for two reasons:

1. In recurrent networks such as LSTM, the activation can exhibit outlier behavior overtime, making the range of activation very difficult to pre-determine.
2. We could not find a suitable static quantization library for both of LSTM and Conv1D layers. Even if we implemented static quantization manually, we would ultimately observe less speedup on mobile devices due to lack of hardware support.

3.4.1 Dynamic quantization

In dynamic quantization, the dynamic range of activation is continually adjusted after deployment based on previous observed values. This reduces the boost in performance. We focus on quantizing the LSTM layers since we did not prune it earlier to minimize the model size. (Additionally, we found no framework supporting dynamic quantization on Conv1D layers).

4 Experiments

4.1 Setup

We evaluate the model's fidelity on the Valentini dataset (7), which (4) also used to benchmark their paper on. However, we opted to fine-tune our models using the DNS dataset (6) to avoid overfitting during fine-tuning. We further deployed our models on an Mi 11 Ultra android device using TF-Lite to measure its latency. Similar to (4), we measured the latency of denoising a 10 second recording to compute the Real Time Factor (RTF), which is the ratio of the model latency over the input duration.

4.2 Pruning Results

The results of comparing method 1 and 2 for skip connections is shown in Figure 3. We compared the sensitivity curves for each of the 4 skip connections using each method. We observe that pruning using method 2 significantly outperformed method 1 for almost all layers and pruning ratios.

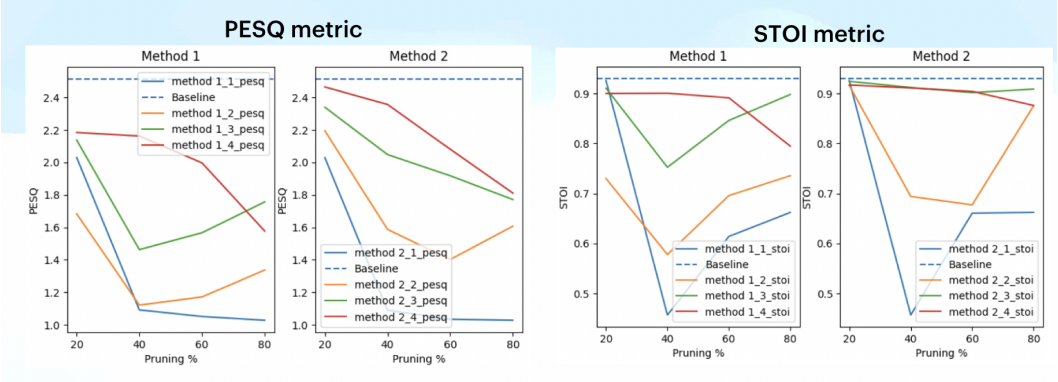

Figure 3: Comparison of method 1 and method 2 for pruning skip connections

Our sensitivity scan analysis are shown in Figure 4. We observe that the inner skip layers are less sensitive to pruning than outer skip layers, and that the innermost encoder and decoder layers are less sensitive as well. We think this is due to DEMUC's design, where there are more channels in the inner layers than outer layers. As we used two metrics, PESQ and STOI, to measure our model's performance, we considered 3 different thresholds for pruning:

1. PESQ drop is <5%
2. PESQ drop is <10%
3. STOI drop is <3%

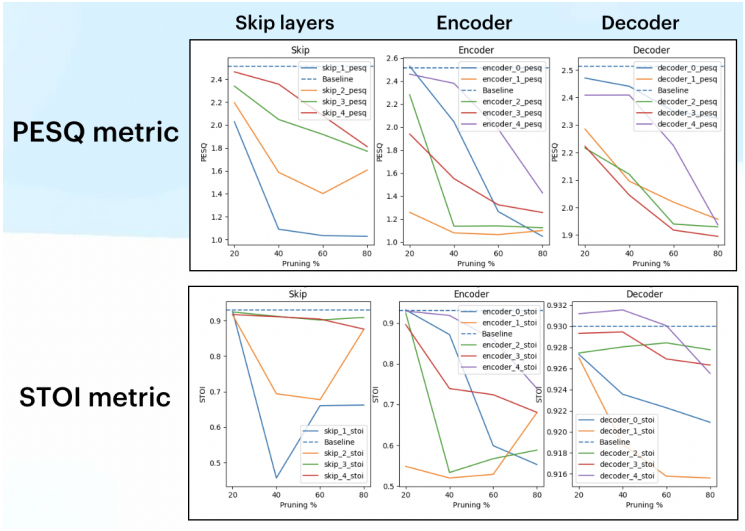

Figure 4: Sensitivity scan results

The final results of these pruning thresholds, as well as uniform pruning, after fine-tuning is shown in Figure 5. We use the "PESQ drop <5%" for deployment and measure its latency improvement as it provided the best tradeoff between model size and accuracy.

4.3 Quantization Results

Table 1: Performance across the different experiments

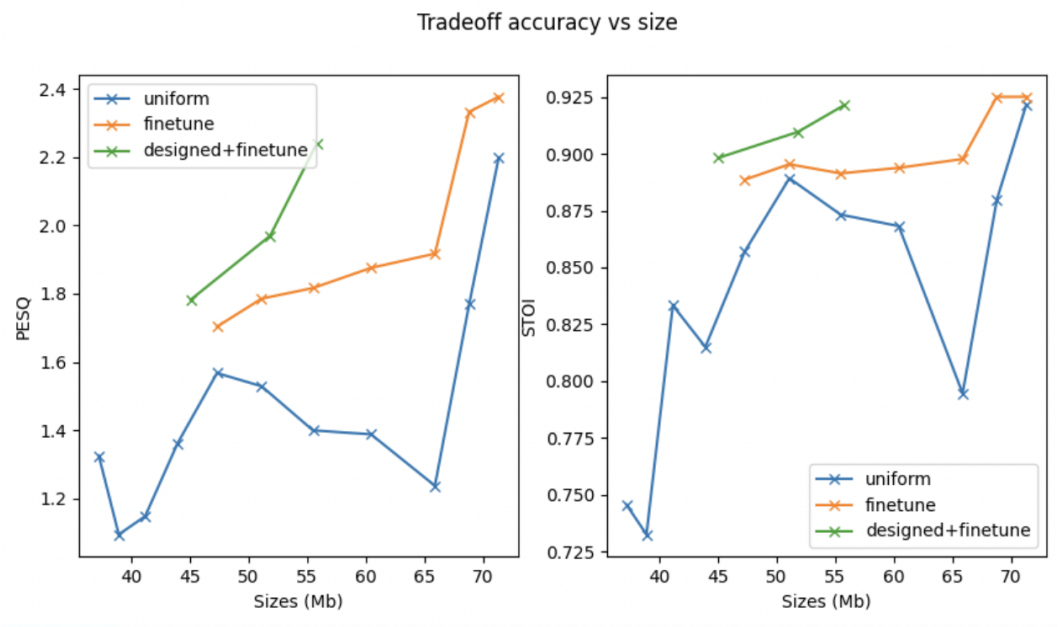

Figure 5: Final pruning results comparing sensitivity scan and uniform pruning, after fine-tuning

Optimizations	Size (MB)	STOI	PESQ	RTF
Baseline	72	0.93	2.514	0.233
Dynamic quantization	27	0.916	2.28	0.170
K-means quantization 4bit	4.6	0.92	2.2	N/A
Pruning	55.8	.92	2.25	0.222
Dynamic quantization + Pruning	18.5	0.914	2.03	0.160

The results for K-means quantization is shown in Figure 6. Although we obtained an 80% reduction in model size using K-means quantization while still maintaining a reasonable accuracy, we were unable to measure its latency during deployment due lack of TF-Lite support.

The results for integer quantization and the final pruned models are shown in 4.3. We notice that our pruning and quantization affect different parts of the model, so there effects can almost be perfectly separated. Pruning results in around .1 reduction in RTF, and 16 MB reduction in size. While quantization results in a .5 reduction in RTF, and around 40 MB reduction.

Additionally, we notice that even though dynamic quantization was not applied to Conv1d layers, it resulted in drastic reduction in the size of the model as around 40 MB was in other layers (mostly in the LSTM). Furthermore, even though the bulk of the latency (around 60%) was in the non-quantized Conv1d layers , the 40% that was quantized was accelerated up to 2-3 times.

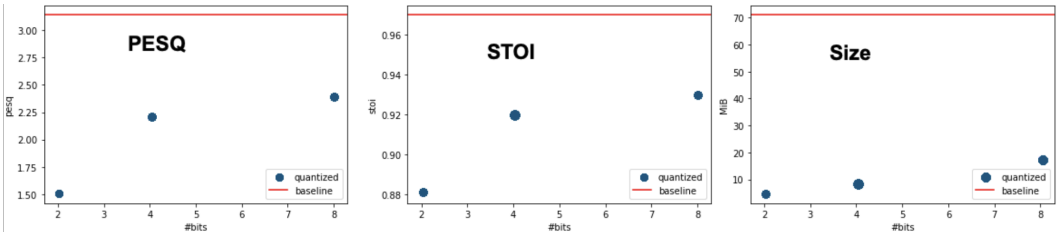

Figure 6: K-means quantization experimental results in terms of performance and model size

5 Conclusion and Future Works

5.1 Conclusion

We performed model size optimization on the DEMUCS model (4) and was able to reduce the model size by 75%, and reduce Real Time Factor (RTF) by 30% without having a noticeable drop in accuracy. Resulting in a lighter-weight, efficient real-time speech enhancer that is deployable on mobile devices. We also find linear quantization to be the most effective optimization method for our use case. Finally, we applied sensitivity-based pruning for pruning u-architectures, which are common in models that perform audio signal processing. Overall, our primary contributions consist of applying a series of optimization on a labtop-CPU-enabled speech enhancement model, and we extended it possibility to be deployed on mobile platform.

5.2 Future Works

5.2.1 More extensive quantization

As we mentioned earlier, we could quantize the Conv1d layers due to lack of support in the frameworks. However, we could find a theoretical reason for this. Seeing the great impact quantization had on LSTMs suggests that we can reduce our RTF to around .1 or below if we had managed to quantize those layers

5.2.2 Mobile-denoiser

We have applied optimization techniques on a network that was developed to run on CPUs in order to adapt it on phones. However, given more time and training resources, we could copy some of the lessons learned in mobile net to find a more mobile friendly architecture:

1. Replace convolutions with depth-wise convolutions.
2. Replace all activations with ReLUs.
3. Reduce the number of GLUs, maybe try to find other more quantization friendly gates that do not include attention functions. A first step might be replacing the attention part of the layer with a ReLU6.

Acknowledgments and Disclosure of Funding

The authors would like to thank 6.S965 TinyML and Efficient Deep Learning Computing teaching staffs for their unlimited support for the paper report and the project. We would also like to acknowledge platforms like Google Colab and Android Studio, which helped us test our model performance. Lastly, the authors would like to acknowledge Pytorch, Tensorflow Lite, and Alibaba for their Tiny Nueral Network framework, which helped in converting the optimized model for deployment [5].

[5] Model conversion to TFLite can be found here: `https://github.com/alibaba/TinyNeuralNetwork`

References

[1] C. Zorila, C. Boeddeker, R. Doddipatla, and R. Haeb-Umbach, “An investigation into the effectiveness of enhancement in asr training and test for chime-5 dinner party transcription,” preprint arXiv:1909.12208, 2019.

[2] S. Pascual, A. Bonafonte, and J. Serra, “Segan: Speech enhance- ment generative adversarial network,” preprint arXiv:1703.09452, 2017.

[3] H. Phan et al., “Improving gans for speech enhancement,” preprint arXiv:2001.05532, 2020.

[4] A. Dfossez et al., "Real time speech enhancement in the waveform domain," preprint arXiv:2006.12847, 2020.

[5] A. Dfossez et al., “Music source separation in the waveform do- main,” preprint arXiv:1911.13254, 2019.

[6] C. K. A. Reddy et al., “The interspeech 2020 deep noise sup- pression challenge: Datasets, subjective speech quality and test- ing framework,” 2020.

[7] C. Valentini-Botinhao. "Noisy speech database for training speech enhancement algorithms and tts models," 2017.

[8] S. Hans et al., "Learning Both Weights and Connections for Efficient Neural Network," NeurlPS, 2015.

[9] S. Hans et al., "Deep Compression," ICLR, 2016.

[10] B. Jacob et al., "Quantization and Training of Neural Networks fro Efficient Integer-Arithmetic-Only Inference," CVPR, 2018.