MODEL COMPRESSION FOR EFFICIENT MACHINE LEARNING INFERENCE

Sunwoo Kim

Submitted to the faculty of the University Graduate School

in partial fulfillment of the requirements

for the degree

Doctor of Philosophy

in the Luddy School of Informatics, Computing, and Engineering,

Indiana University

May, 2022

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Minje Kim, PhD

Fan Chen, PhD

Peter Todd, PhD

Christopher Raphael, PhD

04/28/2022

Sunwoo Kim

MODEL COMPRESSION FOR EFFICIENT MACHINE LEARNING INFERENCE

This dissertation presents model compression methods to facilitate the practicality of deep learning and machine learning frameworks for real-time applications. Starting from conventional compression techniques such as quantization to reduce bit-widths, we extend to developing novel and compact frameworks through a lossless compression approach.

We begin with an extreme network quantization algorithm to compress a floating-point deep neural network using single bit representations. The training is done in two rounds to preserve the model performance, first in a weight compressed real-valued network and then in a bitwise version with the same topology. The pretrained weights of the first round are used to initialize the weights of the bitwise network, where we redefine the feedforward procedure with bitwise values and operations. Only the bitwise network is used for deployment for test time inference, which not only makes it easier to put on small devices but also expedites the inference speed with bitwise arithmetic operations. For this study, we aim at compressing a recurrent neural network architecture for single-channel source separation. Applying extreme quantization on this type of network poses additional challenges due to its complex recurrent relations as quantization noise can accumulate over multiple time frames. We address this by proposing a more delicate solution to incrementally binarize the model parameters in order to minimize the potential loss that can occur from a sudden introduction of quantization. As the proposed binarization technique turns only a few randomly chosen parameters into their binary versions, it gives the network training procedure a chance to gently adapt to the partly quantized version of the network. It eventually achieves the full binarization by incrementally increasing the amount of binarization over the iterations.

Binarization can be extended to data compression to provide the same benefits of extreme compression rates and expedited inference speeds using supported algorithms and hardware. Similarly to binarizing model weights, we propose to compress the bitwidths of data down to binary form

with emphasis on minimizing loss of information. To this end, we introduce locality sensitive hash functions (LSH) to reduce the storage overhead while preserving the semantic similarity between the high-dimensional data points in the Euclidean space and binary codes. However, given the random nature of LSH projection vectors, a large bitstring is required to form discriminative hash codes that can guarantee high precision. In this dissertation, we propose to learn the locality sensitive hash functions using boosting theory to efficiently encode the underlying structure of data into hash codes. Our adaptive boosting algorithm learns simple logistic regressors as the weak learners. The algorithm differs from AdaBoost in the sense that the projections are trained to minimize the distances between the self-similarity matrix of the hash codes and that of the original data points, rather than the misclassification rate. We evaluate our discriminative hash codes on a source separation problem framed as a similarity search task. Upon training our hash functions, their binary classification results transform each data point into a bit string, on which simple bitwise operations calculate Hamming distance to find the nearest neighbors from the hashed dictionary.

Quantization and other model compression methods can achieve good compression rates, but they are applied as a post-training procedure that propagate noise and decrease generalization performance. Quantization-aware training helps to minimize the accuracy drop by simulating the low precision inference using the same floating point backpropagation, there is a limit to the amount of recovery from this fine-tuning procedure. Furthermore, quantized models demand dedicated hardware designs to support bit-level manipulation in memory and computation units to reap the benefits from model reduction.

We address this worsened generalization and hardware compatibility issue of model compression methods by improving compact models to outperform larger model counterparts as a form of lossless compression. The first approach is personalization, in which small models are fine-tuned to their test-time specificity. Personalized compact models are trained in original floating-point values without structural modifications, and do not require any specialized hardware. We aim at use-

cases for end-user devices in realistic settings where we often encounter only a few classes within a target domain that tend to reoccur in the specific environment. Hence, we postulate a small personalized model suffices to handle this focused subset of the original universal problem. Our goal in this test-time adaptation is to develop personalized speech enhancement model targeting edge-devices that can perform well for relevant users' voices and surrounding acoustics (e.g. a family-owned smart assistant device). One major challenge for personalization is a major data shortage issue due to recent privacy infringement and data leakage issues. Our goal in this test-time adaptation is to perform personalized speech enhancement without utilizing clean speech target of the test speaker using a knowledge distillation framework. We distill the denoising results from an overly large teacher model, and use them as the pseudo target to train the small student model. Experimental results show that the personalized models outperform larger non-personalized baseline models, demonstrating that personalization achieves model compression with no loss of denoising performance.

Finally, we propose another lossless approach using evolutionary algorithms to optimize compact generative adversarial networks. We coordinate the adversarial characteristics with a coevolutionary strategy and evolve a population of models to achieve high fitness corresponding to generative performance and training stability. Our framework exposes individuals to not only various but also fit and stronger adversaries per generation to learn robust and compact models for efficient and faster inference. The experimental results demonstrate generative models trained using the proposed coevolutionary strategy can produce small models capable of outperforming larger counterparts trained under the regular adversarial framework.

Minje Kim, PhD

Fan Chen, PhD

Peter Todd, PhD

Christopher Raphael, PhD

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Advances in deep learning have shown significant improvements for applications from various domains demonstrating human-level capabilities [1]. Deep neural networks (DNN) have long surpassed accuracies of conventional approaches and continue to improve each year. Recently, DNNs have achieved state-of-the-art results in applications within the audio domain such as speech separation and automatic speech recognition [2, 3]. Developments of various neural architecture design have enabled advanced audio signal processing: Recurrent neural networks (RNN) [4, 5] handling long term dependencies and time-domain audio separation networks (TasNet) [6] performing end-to-end source separation. Deep models have also shown superior performance in the field of computer vision, in applications such as image classification and synthesis [7, 8]. Generative adversarial networks (GAN) [8] have shown to excel at synthesizing and editing photorealistic images. Despite their impressive capabilities, modern deep neural architectures require significant space and computational complexity. For example, RNNs come with heavy memory consumption to store hidden states and cells [9], and GANs along with other convolutional architectures are computationally intensive, which also implies longer training time and latency [10, 11]. Recent neural network designs such as GPT-3 [12] report more than 100 billion parameters and Tutel[1] with 1.1 trillion parameters, and GauGAN consumes more than 250G MACs[2] per image [13].

To accommodate their expensive cost, DNN frameworks are typically deployed in the cloud where memory and computation constraints are not an issue. From an external location, large-scale computationally intensive training on huge data volumes can be done across distributed hardware.

[1] https://github.com/microsoft/tutel

[2] Multiply-Accumnulate Operations. Quantifies the computation cost. 1 MAC = 2 FLOPs

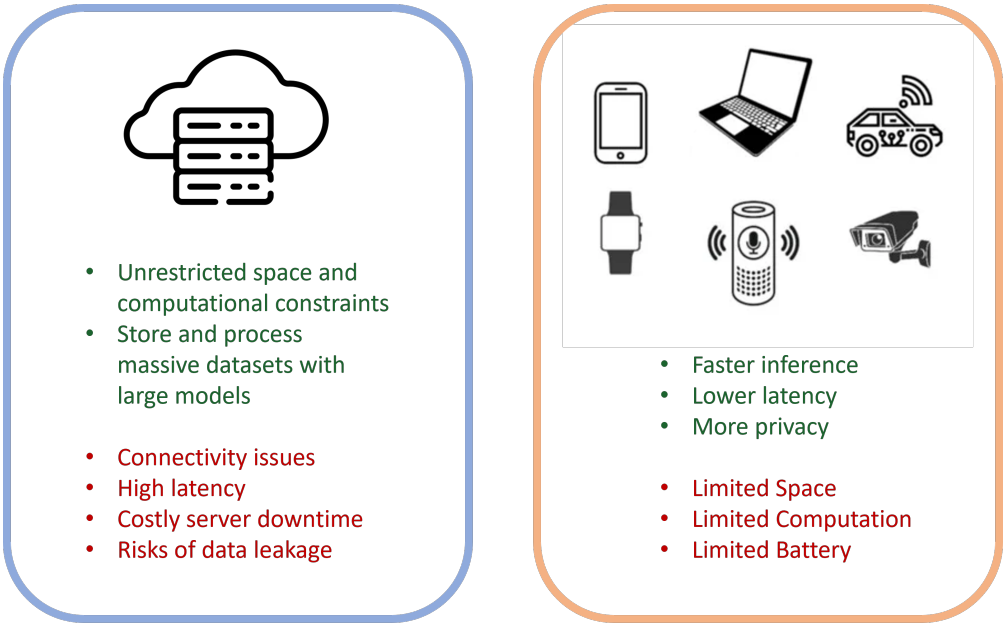


Figure 1.1: Tradeoffs between deploying models on cloud and edge-devices.

Despite the numerous benefits, cloud computing presents several drawbacks. As cloud computing networks are highly centralized, data gathered on the edge-devices need to transmitted back to the main servers for processing. Consistent connectivity with sufficient bandwidth from device to cloud is required, and latency from communication can be an issue and slow down the inference speed. There are also privacy concerns due to recent privacy infringement and data leakage issues, and users have grown wary of sharing sensitive information. For example, the advancement in DeepFake technology increased people's concern towards releasing personal information [14]. Ideally deep models would be deployed on edge devices that are close to the individual users instead of a remote server room. For DNNs placed on the edge of the network, inference will be computed using data generated from the same device; real-time edge-computing enables faster inferences and greatly reduced latency [15]. Since data stays on the device rather than being transmitted back to a central data center, edge computing also provides better data security. Furthermore, with the proliferation of Internet of Things (IoT) devices (e.g. wearables, smart assistants, phones, etc.), the demand for deploying DNNs on edge devices has been growing [16].

Although running DNNs on edge devices is appealing, their increasing complexity impedes edge-deployment and limits their practical usage. Embedded devices have small memory and weak battery, it is difficult to fit large models and also costly to perform heavy computations that will quickly drain the battery. For deep models to be widely applicable for real-time applications on edge-devices, it is paramount to compress the overall complexity. Consequently, research in model compression have been gaining interest to address the practicality of deep-learning architectures for real-time applications. The objective of model compression is to reduce redundancy in over-parametrized network architectures to obtain hardware-efficient models that enable fast and accurate inference with reduced model sizes. Some common modes of compression such as quantization and knowledge distillation have shown great promise in dramatically reducing the complexity of DNNs. To reduce redundancy in network weights, researchers proposed quantization to reduce the bit-depth of model parameters by replacing the floating-point weights of trained networks with a lower-precision representation [17, 18]. Pruning is a sparsification method that removes nodes or filters that are deemed relatively insignificant by zeroing out the values in the weight matrix [19]. Knowledge distillation (KD) uses a different approach to compression that distills the output of a larger and more complex network as a soft target for the training of a smaller and simpler network [20].

Model compression methods have shown great promise in dramatically reducing the model size and complexity. However, it is commonly expected to observe a certain level of performance drop after compression. Despite efforts try to minimize the drop in generalization performance, applying compression as a post-processing step often incurs noise that subsequently degrade generalization performance (e.g. quantization noise from reducing the bit-widths of model parameters) [18]. Fine-tuning operations such as quantization-aware training or iterative pruning aid in recovery from compression noise; however, it is not guaranteed that the models can achieve its previous performance. Another issue is compressed models (i.e., quantized or pruned) require specialized

hardware to support sparsity or bitwise storage and operations, which otherwise would not provide significant benefits of reduced complexity. A potential solution is to pursue lossless compression methods to produce compact models capable of outperforming larger and more complex models.

In this thesis, we help further bridge the gap between deep learning research and practical deployment for real-time applications through novel model compression methods. We begin with a solution to binarize an existing floating-point DNN models, specifically the recurrent neural network. Its complex recurrent relations leads to additional challenges as quantization noise can accumulate over multiple time frames. Hence, we address this by proposing a more delicate solution to incrementally binarize the parameters. With binary inputs, binarization not only reduces the model size but also expedites the inference speed with bitwise arithmetic operations. We extend binarization and apply extreme quantization on dataset features while preserving their discriminative properties. To extract higher-level features and better encode the underlying structure of data, we introduce learnable locality sensitive hash (LSH) functions that can preserve its semantic details. In addition, we address challenges facing model compression methods. We seek to overcome the performance drop and hardware compatibility issues of existing compression methods by implementing a personalization framework to adapt to the test-time conditions and improve their generalization performance. Finally, we present an evolutionary training approach to discover robust and compact generative adversarial networks.

1.2 Contribution and Thesis Outline

This thesis proposes techniques for model compression. The contributions of this thesis include:

- An extreme quantization technique using binarization to quantize a RNN model's inputs and outputs, weights, and also activations into binary representations, which significantly reduces the space and time complexity.
- An adaptive boosting approach to learning locality sensitive hash codes, called Boosted Local-

Method	Description	Advantages
Quantization	Reduce bit-widths of elements from a trained network	• Can fine-tune using *quantization-aware training* • Integer or bitwise arithmetic can lower latency
Pruning	Sparsification method that zeroes out insignificant elements	• Can fine-tune through *iterative pruning* • Structured removal can lower latency
Knowledge-distillation	Distill output from a larger network to train a more compact model	• Can personalize and produce high-accuracy students of compact model size
Evolutionary methods	Evolve solutions using evolutionary operators to maximize a fitness criteria	• Can evolve a population to reach a compact, and accurate network architecture

Figure 1.2: Overview of model compression methods

ity Sensitive Hashing (BLSH), that can best preserve the discriminative properties of dataset features using only binary representations.

- A personalization framework as a mode of model compression that adapts small compact models to test-time environments in a zero-shot manner without requiring any ground-truth target data.
- An evolutionary algorithm that stabilizes the training of generative adversarial networks and learn robust and compact generator and discriminator pairs.

Chapter 2 surveys related works in model compression and provides relevant background information for the applications and deep neural networks that we use in this thesis.

Chapter 3 describes a binarization technique to not only reduce the bit-width of RNN model parameters, but also accelerate its inference speed. Due to complex recurrent relations, compressed RNN models are susceptible to quantization noise accumulating over multiple time frames. To address this issue, we introduce a smooth binarization method to incrementally binarize the model parameters.

Chapter 4 extends binarization to significantly compress dataset features while preserving their discriminative properties. We introduce learnable locality sensitive hash functions to extract higher-level features and efficiently encode the underlying structure of data using 1-bit representations.

Chapter 5 presents a novel personalization framework to adapt compact models to test-time environments in a zero-shot manner without requiring any ground-truth target data. Instead, we employ a knowledge distillation method to generate pseudo targets from a larger teacher model from solely noisy inputs and distill it to fine-tune a compact student model.

Chapter 6 proposes an evolutionary algorithm to train and evolve generative adversarial networks. We make use of the adversarial aspect of the GANs and coordinate training with genetic algorithms to produce robust and compact models.

In **Chapter 7** we summarize the thesis and discuss the future work for network compression.

Chapter 2

Background

In this chapter, we cover background information for the model compression methods proposed in this dissertation. We first introduce previous work in model compression covering quantization, hashing, knowledge-distillation, and neural architecture search. Then we describe the applications we target and the neural network architectures that we used in our experiments.

2.1 Related Work

Research in model compression methods have been gaining interest to address the practicality of deep-learning architectures for real-time applications. In this section, We briefly review previous work that our research builds on. We cover relevant works in quantization to reduce the bit-width, knowledge distillation to distill larger teacher models onto smaller student models, hashing for data compression, and neural architecture search using genetic algorithms.

2.1.1 Quantization

Quantization is one of several popular compression techniques that reduces the bit-width of floating-point elements. This method uses low-precision representations to compress the model size and simplify operations for accelerated inference computations. Quantization achieves a good compression rate providing significantly reduced bandwidth and memory space. For example, quantizing from 32-bit floating point to 8-bit integers for weights and activations reduces model complexity fourfold. With supporting hardware, quantized networks can also benefit from faster integer computation that can improve inference speed during runtime. Implementations for quantization typically use fixed point quantization with uniform number of bits for model parameters.

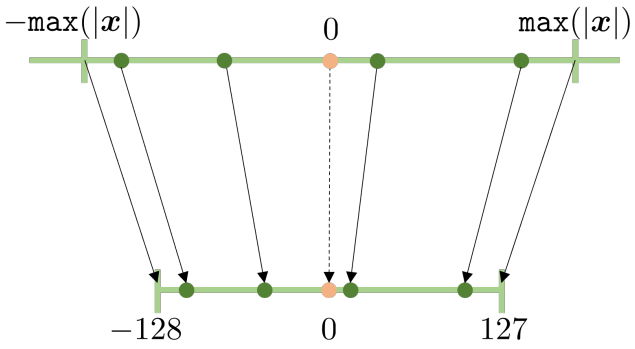


Figure 2.1: Illustration of uniform symmetric quantization with $N = 8$ bits.

Fixed-Point Quantization

Quantization applied on DNNs are applied on a pre-trained network as a post-processing procedure [17, 21]. Also known as post-training quantization, this process quantizes the bit-width of pre-trained model parameters, typically in FP32 numerical format, to lower precision. Applying fixed-point scalar quantization replaces the floating-point weights with N bit fixed-point representations. More precisely, the weights are rounded to one of 2^N possible evenly spaced bins. Given weight vector $\boldsymbol{w} \in \mathbb{R}^K$ of length K, we wish to quantize it using N bits. Fixed-point quantization function $q(\cdot)$ maps the i-th weight element $\boldsymbol{w}_i$ to its nearest bin $\overline{\boldsymbol{w}}_i \in \mathbb{Z}$ as

$$\overline{\boldsymbol{w}}_i = q(\boldsymbol{w}_i) = \left\lfloor \frac{\boldsymbol{w}_i}{s} \right\rfloor \tag{2.1}$$

where s is a real-valued scaling factor and floor operation is used to represent the rounding function. The scaling factor s divides the real values $\boldsymbol{w}$ into $b = 2^N$ number of partitions:

$$s = \frac{2 \cdot \texttt{max}(|\boldsymbol{w}|)}{2^b - 1} \tag{2.2}$$

Fig. 2.1 shows an example of using $N = 8$ bits that results in quantization ranges of $\overline{\boldsymbol{w}} \in [-128, 127]$. This technique, referred to as *uniform symmetric* quantization, is one of many modes of quantization and is widely adopted in practice for its efficiency and simplicity. Other modes such as *asymmetric* quantization adds a scalar quantization bias variable to Eq. 2.1 to offset the zero-point. Another mode is *non-uniform* quantization that better captures the distribution by assigning

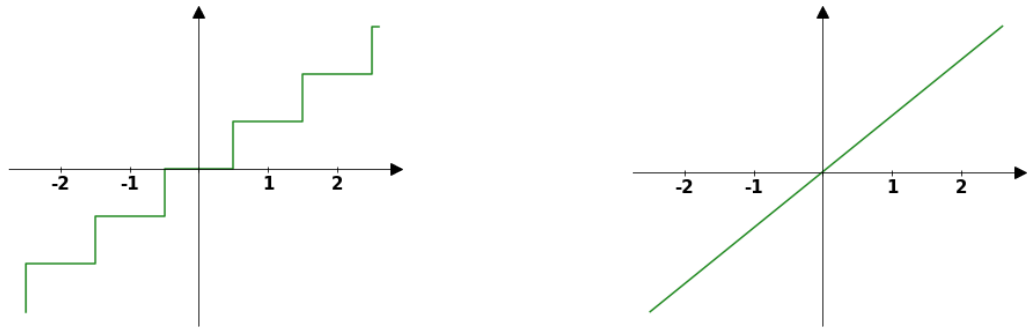
more bits to ranges of values that occur more often (e.g. bell-shaped weight distributions). Since real-valued weights are reduced to a much lower resolution, post-training quantization results in loss in accuracy [18, 22]. Models are not exposed to the quantization noise injected from the procedure, leading to suboptimal performance. To address this, quantization-aware training proposes to expose the network to quantization during training to make it robust to the noise.

Quantization-Aware Training

In order to recover from the incurred quantization loss, the low-precision model can be re-trained through quantization-aware training [23]. Throughout the fine-tuning process, a copy of the real-valued model is maintained. The model parameters are quantized at every feedforward pass, and its full precision weights are updated during backpropagation. This allows the weights to accumulate the small changes in the gradients without loss of precision. Ultimately, after fine-tuning, only the quantized weights are used for inference. One of the main challenges with fine-tuning a quantized model is that the discretization operator is non-differentiable. The rounding operation in Eq. 2.1 is a piece-wise flat function so the derivatives with respect to the input are zero almost everywhere $\partial\overline{\boldsymbol{w}}/\partial\boldsymbol{w} = 0$. From Fig. 2.2a, it can be observed that the gradients will be zero everywhere except at the quantization bin thresholds, in which the gradients would be an unstable infinite value. To resolve this issue, the gradients are approximated with a Straight-Through-Estimator, which essentially ignores the low-bit transformation and approximates it with an identity function [24].

$$\begin{aligned} \overline{\boldsymbol{w}} = q(\boldsymbol{w}) &\approx \boldsymbol{I}(\boldsymbol{w}) = \boldsymbol{w} \\ \frac{\partial\overline{\boldsymbol{w}}}{\partial\boldsymbol{w}} = \frac{\partial q(\boldsymbol{w})}{\partial\boldsymbol{w}} &\approx \frac{\partial\boldsymbol{I}(\boldsymbol{w})}{\partial\boldsymbol{w}} = 1 \end{aligned} \tag{2.3}$$

By approximating the q operator with the identity function, gradients can backpropagate through the network and smoothly update the full precision model to the quantization error. Despite its coarse approximation, STE have shown to work well in practice [25]. Quantization-aware training with STE has been successfully applied for various quantization applications even for binarization, an extreme network quantization.



(a) Sample quantization operator

(b) STE gradient approximation

Figure 2.2: Example of uniform symmetric quantization and its approximated identity gradient

Binarization: Extreme Network Quantization

Binarization quantizes values to a 1-bit representation, thereby drastically reducing the memory requirement by 32x (assuming 32 bit floating-point precision). Bitwise neural networks (BNN) transform all components of the framework, even the inputs, into their binary versions [26, 27]. Instead of assigning real value weights to quantization bins, the sign function is applied to acquire a set of bipolar binaries, as

$$\overline{\boldsymbol{w}}_i = \bar{\phi}(\boldsymbol{w}_i) \tag{2.4}$$

Here, $\bar{\phi}(x) = sgn(x) \in \{-1, +1\}$, where $sgn(x)$ denotes the sign function, and $\overline{\boldsymbol{w}} \in \mathbb{B}$ refers to the binarized model parameters. Since the conversion from real-values to bipolar binaries incurs severe quantization loss, one method to reduce this penalty is the concept of sparsity. Sparsity can be introduced to bitwise networks by converting the pretrained weights with smaller values to 0's [23, 28]. The threshold for determining the sparsity is calculated with a predefined boundary β.

The relaxed quantization process for a weight element w is:

$$\overline{\boldsymbol{w}}_i = \begin{cases} +1 \text{ if } \boldsymbol{w}_i > \beta \\ -1 \text{ if } \boldsymbol{w}_i < -\beta \\ 0 \quad \text{otherwise} \end{cases} \tag{2.5}$$

Another way to mitigate the quantization error is by multiplying a scaling factor μ to the bipolar-binarized weights, so that the quantized values approximate the original values more closely [29]. In addition, the sign function is further applied as the activation function to ensure that the intermediate representations are kept in bipolar binary form as well. During the pre-training phase the network is softly introduced to the binarization procedure by wrapping all weights and assigning all activations with a hyperbolic tangent,

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.6}$$

as a soft approximation of the sign function. Thus, all weights and intermediate outputs are constrained between -1 and $+1$, allowing a smooth transition to bitwise values.

Binarization with sparsity and ϕ-compressed weights still leads to severe accuracy degradation. To minimize the loss, quantization-aware training is applied similarly to fixed-point quantization except with subtle variations. The binarization function is applied during the feedforward, and the gradients are backpropagated through the full-precision network; the derivatives for binarized network weights are still approximated using an identity function. For the activations, the derivatives for the non-differentiable sign functions are approximated using that of the hyperbolic tangent function. Empirical results show that it works well for speech denoising and hand written digit recognition.

Not only does binarization reduce the size of the network, it also reduces the energy consumption. Bitwise neural networks (BNN) can estimate feedforward inferences using only bitwise operations (e.g. XNOR and bit counting), which further accelerates inference speed compared

to floating-point or integer arithmetic of real-valued or fixed-point quantized models respectively. However, this benefit only holds when all of the operands are in binary form. The computation savings for binary networks using real-value and binary inputs are vastly different, with speed up being 2× to 58× respectively for convolutional networks.

Applying binarization to the input data poses a similar challenge as to the model parameters, which is to minimize the loss of information. A solution to preserve the semantic similarity between real-valued and binary data has been explored in the form of locality sensitive hashing.

2.1.2 Hashing

Hashing based approaches have been explored as efficient solutions to compress and process huge volumes of datasets. The encoded dataset allows for effective similarity search, which aims to to find items similar to some given query.

Locality Sensitive Hashing

Given a collection of items represented as points in a high dimensional feature space (e.g., documents, images, audio files, etc.), a similarity search problem aims to find the nearest or most similar item in the dataset to a given query [30]. This problem has been crucial to several areas such as databases and data mining, information retrieval, signal processing, and machine learning to name a few. Given a set $\mathcal{S}$ of n database items, a naive solution to finding neighbors requires enumerating over all points and sorting them with respect to their similarity distance to the query. The running time of this approach is linear with the number of items and the complexity of the object. So when n is large and $\mathcal{S}$ is comprised of high-dimensional items requiring complex similarity function evaluations, this approach becomes prohibitively expensive.

For efficient large-scale search, approximate similarity search techniques can be used to sacrifice some accuracy in order to allow fast queries even for high-dimensional samples [31]. One of the most notable methods is Locality Sensitive Hashing (LSH) that retrieves items within c times the

optimal similarity threshold R in sub-linear query time.

LSH method hashes data points using hash functions that ensure the probability of collision is higher for objects that are close to each other in the original Euclidean space than those that are farther apart [32]. Formally, for any two points $p, q \in \mathcal{X}$ in metric space $(\mathcal{X}, d)$ for some distance metric d and LSH function h, LSH must satisfy

$$\begin{aligned} &\text{if } d(p,q) \leq R \text{ then } h(p) = h(q) \text{ with probability at least } P_1 \\ &\text{if } d(p,q) \geq cR \text{ then } h(p) = h(q) \text{ with probability at most } P_2 \end{aligned} \tag{2.7}$$

for distance threshold $R > 0$, approximation factor $c > 1$, and probabilities P_1 and P_2 where $P_1 > P_2$. This states that points closer to the query q within distance R will be more likely to collide with the query, while those further away have a smaller chance P_2 of collision. Furthermore, LSH hash functions must satisfy

$$d(p,q) < d(q,r) \Rightarrow |h(p) - h(q)| < |h(q) - h(r)| \tag{2.8}$$

for some point $r \in \mathcal{X}$. This states that the relative pairwise distance between the data points is preserved in the hashed space.

LSH uses random projection vectors to transform data points into low dimensional binary embeddings that can approximate complex distance functions using the Hamming similarity metric. This way, items similar in the Euclidean space maps to similar binary code bins. For a given query, the nearest neighbors are determined by hashing the query point and retrieving elements that are stored in the buckets containing that point. Given more random projections and longer code lengths, the Hamming distance between two codes asymptotically approaches the Euclidean distance between their corresponding data points. LSH provides an approach for embedding high-dimensional feature vectors to a low-dimensional Hamming space while retaining as much as possible. However, there are several disadvantages to the LSH approach. One main drawback being that LSH is a data-agnostic method and does not explore the data distribution when choosing hash functions.

Therefore, to preserve the geometry of high-dimensional data lots of hash bits are required, leading to inefficient hash codes.

Data-aware Hashing

Semantic hashing based methods are data-dependent techniques that design compact binary cods for a large dataset such that semantically similar documents are mapped to similar codes (within a short Hamming distance) [33]. By constraining the latent variables into binary representations, high-dimensional features are mapped into a low-dimensional Hamming space while preserving much of the semantic similarity structure of data. The data points are mapped into few bits that could then be reconstructed back to the original representation. Obtaining codes for previously unseen samples using semantic hashing is still a challenging task. Spectral hashing is another data-aware hashing based method that takes an unsupervised approach to learn similarity preserving hash codes [34]. Inspired by spectral clustering, it leverages machine learning to find optimal hash functions that could preserve similarity structures between data points. Both semantic and spectral hashing methods have shown significant improvements over LSH in terms of the number of bits required to find good similar items.

Binarization induces significant quantization loss to deep neural networks and data features, and we have proposed incremental and hashing-based methods to minimize the information lost from the extreme quantization procedure. Yet, these solutions are computationally intensive and time-consuming, and it would be ideal to compress without experiencing severe performance losses. Knowledge-distillation methods apply model compression from a different direction. Instead of compressing a large model, it improves the accuracy of a small model instead, at times exceeding the performance of much larger models.

2.1.3 Pruning

Pruning is a sparsification method that removes redundant connections and filters within the model. It can be divided into two categories: unstructured and structured. Unstructured pruning finds redundant network connections and removes those least significant weights. Through an iterative procedure of pruning followed by retraining the remaining elements, the model size and computation cost can be drastically lowered with minimal loss in accuracy [35, 36]. On the other hand, structured pruning finds the least significant filters or channels through a similar iterative procedure. Pruning structural components not only reduces the model size, but it also directly corresponds to the savings in computation [37, 38, 39]. Furthermore, instead of exhaustively searching for an optimal network topology, pruning can be adopted as an automatic neural architecture search framework [40, 41, 42]. The pruned architecture can help to give insight into designing better networks [43, 44, 45]. The resulting subnetwork can also be trained more efficiently given the thinner and sparser structure. The Lottery Ticket Hypothesis conjectures that by rewinding to its initial weights, the subnetwork can reach similar performances of the original model prior to pruning [46, 47, 48, 49].

2.1.4 Knowledge-Distillation

Knowledge-Distillation (KD) is a versatile method that uses a large model as a teacher to train a more compact student model [20]. Also known as a student-teacher framework, KD trains the student model by using the output distribution predicted by the teacher as soft labels along with the hard labels (e.g. one-hot vector). The distillation loss $\mathcal{L}_{KD}$ with teacher $\mathcal{T}$ and student $\mathcal{S}$ models is defined as

$$\mathcal{L}_{KD} = \mathcal{L}(\mathcal{T}(\boldsymbol{x}), \mathcal{S}(\boldsymbol{x})) \tag{2.9}$$

for some input $\boldsymbol{x}$ and predefined loss function $\mathcal{L}$ (e.g. cross-entropy loss) (Fig. 2.3). In the original definition, KD was formulated to capture *dark knowledge*, or the uncertainties of the teacher, encoded within the estimated output distribution. In classification tasks, the *dark knowledge* in

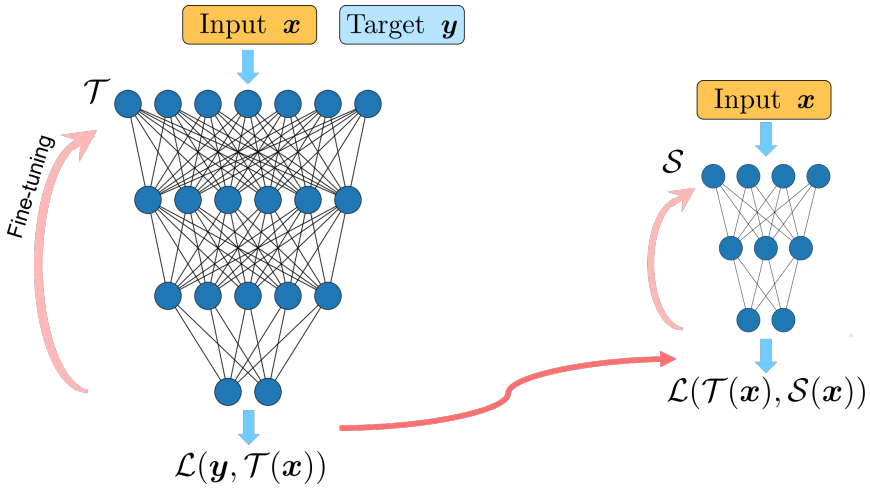


Figure 2.3: Illustration of knowledge-distillation using teacher $\mathcal{T}$ and student $\mathcal{S}$ models.

the form of the soft probability targets $\boldsymbol{q}_i$ can be computed as

$$\boldsymbol{q}_i = \frac{\texttt{exp}(\boldsymbol{z}_i/t)}{\sum_j \texttt{exp}(\boldsymbol{z}_j/t)} \tag{2.10}$$

where $\boldsymbol{z} = \mathcal{T}(\boldsymbol{x})$ and $\boldsymbol{z}_i$ denotes the estimated logit for class i and t is an additional temperature parameter to control the smoothness of the output probability distribution. Larger values of t produces a smoother distribution over output classes.

The student can be additionally trained with the distillation loss to minimize the loss between its estimates and both the soft and hard labels. Given input $\boldsymbol{x}$ with its associated ground-truth target $\boldsymbol{y}$,

$$\mathcal{L}_{KD} = \lambda \mathcal{L}(\mathcal{T}(\boldsymbol{x}), \mathcal{S}(\boldsymbol{x})) + (1 - \lambda)\mathcal{L}(\boldsymbol{y}, \mathcal{S}(\boldsymbol{x})) \tag{2.11}$$

with λ hyperparameter determining the tradeoff between two losses. To provide further guidance for the student, the student can be taught how the teacher has learned as well. For example, instead of distilling just the final outputs, the feature maps produced by intermediate layers of the teacher can also be included in the distillation loss by additionally minimizing the differences between the selected intermediate layers of both student and teacher models [50].

Furthermore, this framework shows flexibility as it allows structural differences between the teacher and student; even an ensemble of complex models can be assigned as the teacher from

which a compact student learns from various teachers simultaneously. KD has shown to be effective in training compact models in various tasks such as image classification and speech recognition. Students outperform baseline models with the same architecture by a large margin, demonstrating its success as a mode of model compression. Given the flexibility of the framework, KD can be combined with other compression approaches such as quantization by incorporating the distillation loss into the fine-tuning a smaller quantized student network [51, 52].

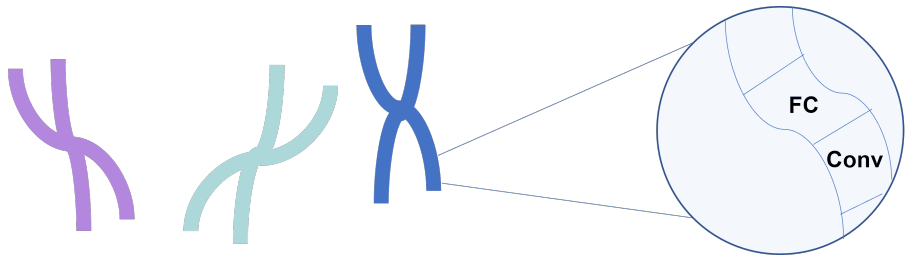
Although KD can enhance the performance of small models, designing their network architecture is another difficult task. The *best* structure can be approximated, but it is uncertain whether they have been initialized with the optimal compact network structure. Neural architecture search offers a solution for automating such design of deep neural networks and have shown to outperform random initialization strategies.

2.1.5 Evolutionary Computation

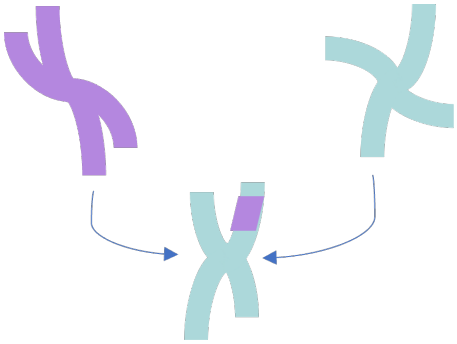
Evolutionary computation is an optimization technique motivated by Darwinian principle of evolution. It borrows from nature's problem solving strategies and aims to mimic its biological mechanism. GA contrasts with gradient descent-based optimization methods; it has been prominently utilized for non-differentiable tasks without smooth gradient manifolds and even for problems without known solutions.

Genetic Algorithms

One class of evolutionary algorithms, genetic algorithms (GA) [53]. Inspired by natural evolution, GA is a population-based optimization technique that models a solution as an individual in a population and produces more competent offspring solutions through numerous evolutionary cycles. The GA strategy uses an abstracted version of evolutionary processes to evolve solutions to given problems. Each algorithm operates on a population of artificial chromosomes that represent solutions to the problem and their fitness is measured by a predetermined performance criteria



(a) Chromosomes representing DNN solutions

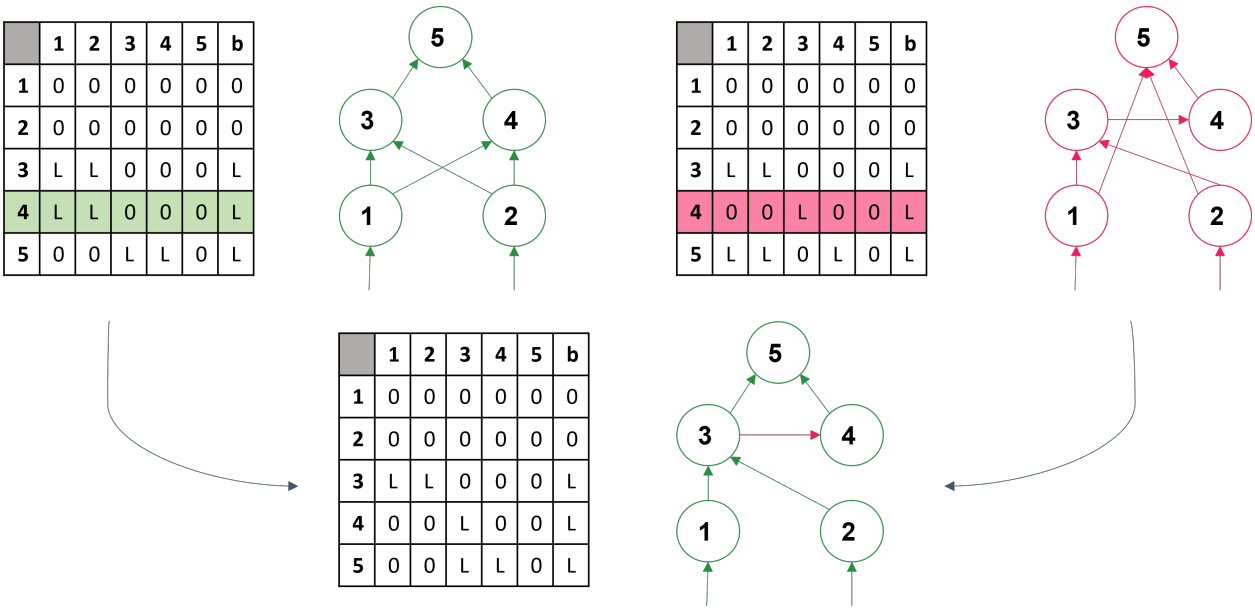


(b) Crossover operation

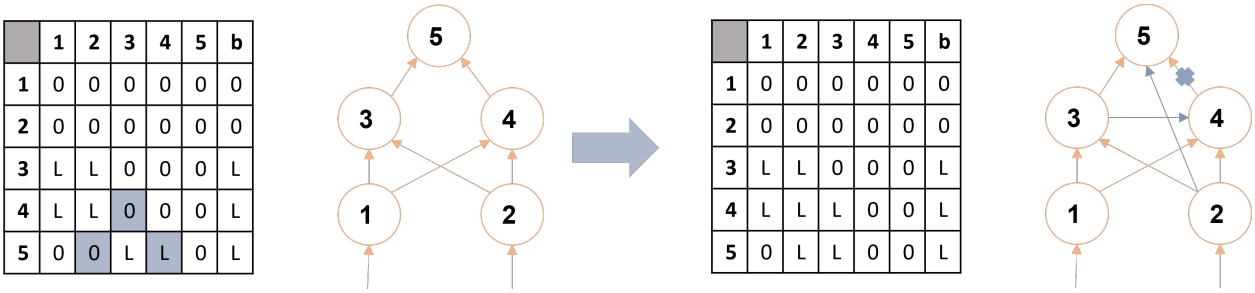
Figure 2.4: Sample DNN representations and genetic operators in GA optimization

(Fig. 2.4a). Genetic operators, inspired by natural selection and genetic variation, are used to mix one population of chromosomes with another and promote population diversity by a set of genetic operators, primarily crossover and mutations: crossovers exchange subparts of two chromosomes (Fig. 2.4b) and mutations randomly alter some values of an individual. GA optimization problems are framed to promote the *survival of the fittest* such that on average the fitter chromosomes should produce more offspring. Through repeated evolutionary cycles, or generations, a better successor population is produced until some stopping criterion.

Evolutionary approaches have been proposed to apply evolutionary algorithms for training neural networks and neural architecture search. Neural network architectures through human-design requires heuristics that can be biased, inefficient, and prone to error. Genetic algorithms has been previously explored to automate neural architecture search across the combinatorially immense space of complex network architectures. At each evolutionary cycle, the network architectures are



(a) Example crossover operation: a row between two parent matrices are swapped to produce an offspring.



(b) Example mutation operation: random genes are turned on or off

Figure 2.5: Illustration of crossover and mutation genetic operators on fully connected networks

initialized and evolved to produce an offspring population based on all networks' fitness values to produce models with more highly adapted network designs. This process can be combined with gradient descent optimization to modify the model weights so that it can perform a more accurate function approximation.

One of the network representation strategies is to represent the genes as individual connections such that it can capture the connectivity patterns precisely and deterministically without much intrusion from human design biases. The architecture of a network of N units can be represented

by a connectivity constraint matrix $\boldsymbol{C}^{N\times(N+1)}$ [54]. First N columns of matrix $\boldsymbol{C}$ specify the constraints on the connections between the N units and the final $(N+1)$-th column holds the biases of each unit. The i-th column of $\boldsymbol{C}$ represents the fan-out connections from unit i while the j-th row holds the fan-in connections to unit j. Each entry $\boldsymbol{C}_{i,j}$ holds either 0 for no connection or L for a learnable connection. An example of a connectivity constraint matrix $\boldsymbol{C}$ for a network with 5 units (with 2 input units and a single output unit) is shown in Fig. 2.5 with the corresponding architecture. The crossover genetic operator is implemented by selecting an i-th random row where $i = [1, N]$ and swaps all elements of $\boldsymbol{C}_{.,i}$ between two parents (Fig. 2.5a). The mutation operator randomly selects entries within $\boldsymbol{C}$ and randomly chooses a new constraint value (0 or L) with some pre-defined probability (Fig. 2.5b). After the genetic operators have been applied, the networks or more specifically the learnable connections are then trained using backpropagation, and resulting fit models are selected to proceed to the next generation. The performance measure for evaluating the fitness of networks is chosen to reflect the design criteria deemed important for the given application. With this specification scheme, empirical results showed not only that the generated network architecture could successfully solve various tasks but also discover unusual architectural designs that help speed up faster learning.

To go further, GA-based optimization has been successfully applied to provide automatic design of more complex network structures. Through a properly defined evolutionary optimization schema, the network architecture topology and weights along with its hyper parameters can be evolved to produce an optimal model. Neuroevolutions of Augmenting Topologies (NEAT) [55] demonstrated the network topology could be evolved from a simple to complex structure leading to increasingly sophisticated behavior. NEAT addresses the competing conventions problem (or permutations problem), a major problem deemed as the *Holy Grail* in neuroevolution, where there is more than one way to express a solution using a neural network. For example, genes from separate individuals but from the same network position could represent different intermediate features, or genes in

different positions could potentially represent the same features as well. Crossing over one neuron from a network to another will lose the functional interdependence and information encoded among neurons.

NEAT addresses this issue by tracking changes within the lineage of every gene in the system to allow network structures to increase in complexity. The NEAT approach starts from small simple models and adds onto the network topology over generations, while using historical markings on each new network structure that appears through mutation. A global innovation number is assigned to each new gene corresponding to its order of appearance over the generations and is inherited by its subsequent offsprings. The innovation numbers are used to compare between individuals and group similar individuals into separate species, in order to protect new and diverse topologies instead of allowing one type of network to dominate the population. The similarity threshold δ is measured by counting the number of excess E and disjoint D genes between two individuals:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} \tag{2.12}$$

for some coefficients c_1 and c_2 and number of genes in the larger individual N as a normalization factor. By keeping track of the historical origin of every gene in the population, the structural regularities of each gene can be differentiated and identified for speciation.

DeepNEAT is proposed as an extension of NEAT to evolve neural network topologies [56]. It follows the specifications of NEAT except the population of chromosomes are described with a graphical representation in which the nodes and edges are the layers and their connectivity patterns respectively. Each node can contain different types of layers and their descriptions such as convolutional or fully-connected and number of hidden units. The historical markings are assigned and tracked to compute their level of compatibility δ and determine the crossover operations.

2.2 Applications

In this section, we define the applications in the audio and computer vision domain. We target the audio source separation and image classification problems to assess our deep neural networks and proposed model compression algorithms.

2.2.1 Source Separation

We first define the single-channel speech separation problem. Given a monaural signal recorded in a noisy environment, we formulate the signal model as

$$\boldsymbol{x}[t] = \boldsymbol{s}[t] + \alpha \boldsymbol{n}[t] \tag{2.13}$$

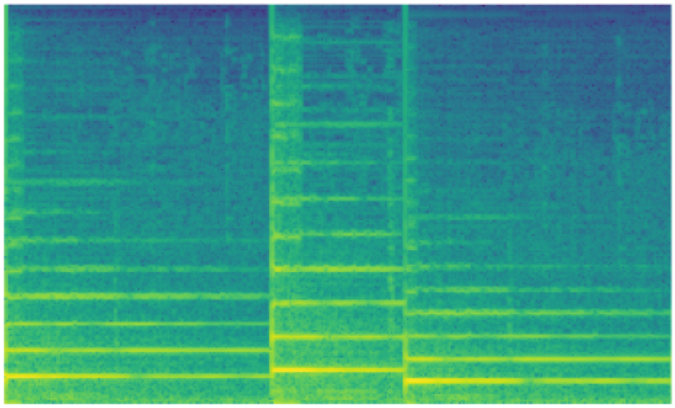
where $\boldsymbol{s}$, $\boldsymbol{n}$, and $\boldsymbol{x}$ denote speech source, background noise, function and noisy speech, respectively. The parameter α controls the signal-to-noise ratio (SNR) between the speech and interfering noise source. Our goal in this study is to separate the speech and noise sources from the corresponding noisy observation $\boldsymbol{x}$ and recover the clean version of the single-talker speech signal $\boldsymbol{s}$. In the following sections, we introduce the relevant data features and methods involved in our source separation applications.

Short-time Fourier Transform

Short-time Fourier transforms (STFT) are widely used in various applications to acquire 2D spectrogram representations in time-frequency (TF) domain from raw 1D time-domain signals. STFT spectrograms separate the signal into separate frequency components which make it easier to interpret than their time-domain counterparts. For example, in the spectrogram created from a audio signal of comprised of three consecutive piano notes (Fig. 2.6b), we can observe the harmonics displayed from the fundamental frequencies.

A Discrete Fourier Transform (DFT) operation is performed on the windowed segment, which

(a) Raw audio



(b) STFT

Figure 2.6: Raw audio and STFT spectrogram of three consecutive piano notes

measures the frequency content as a complex vector of DFT coefficients per frequency bin. Each DFT operation computes one frame of the spectrogram $\boldsymbol{X}$, and we denote the windowed t-th frame of the signal as $\boldsymbol{X}(t,.)$. Let $w(j)$ be a M-point window function defined such that $w(j) = 0$ for $j \notin \{0, \cdots, M-1\}$, which is shifted by a hop size H each time before being applied on the segment of the time-domain signal $\boldsymbol{x}$. We formulate the transformation as:

$$\boldsymbol{X}(t,f) = \sum_{j=0}^{M-1} \boldsymbol{x}(tH+j)\boldsymbol{w}(j)e^{\frac{-2\pi ijf}{M}} \tag{2.14}$$

where t is the frame index, f is the frequency bin and $i = \sqrt{-1}$. The frames are computed for all segments from the signal, which are then concatenated together to form the STFT spectrogram

matrix.

The length of the window, M, determines the trade-off between frequency and temporal resolution of the spectrogram. Wider windows allow for tighter frequency resolution such that close frequencies can be separated, but at the cost of temporal detail, losing clarity on the times when the frequencies change. And vice versa for narrow windows. One of the most commonly used is the Hann window function. This window function is given by

$$w(j) = \frac{1 + \cos(\frac{2\pi j}{M} - \pi)}{2} \tag{2.15}$$

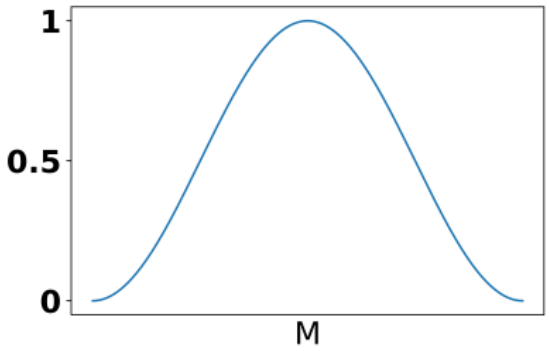
where $j = 0, \cdots, M-1$, and is illustrated in Fig. 2.7a. As shown in the figure, the Hann window places attenuating emphasis on points further away from the center. This window function is also referred to as the raised cosine because it can be interpreted as one period of a cosine scaled upwards such that the negative peaks equal 0.

From the complex STFT spectrogram, this transformation is invertible back to the time-domain. Due to their spectral qualities and invertibility, the STFT representation has been widely used in source separation applications for denoising and speech enhancement.

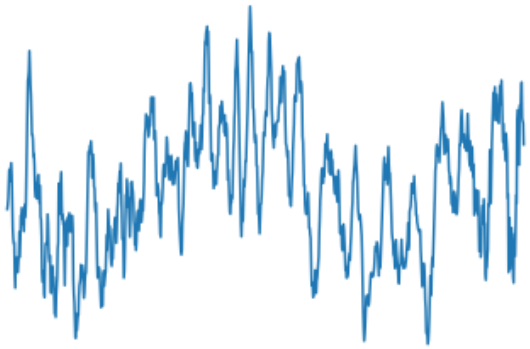
Mel

STFT applies a uniform window length, M, giving equal emphasis to all frequencies; however, the human ear does not perceive frequencies in a linear format. Rather, our ears are better at discerning small changes at low frequencies and have difficulty distinguishing closely spaced high frequencies. In more specific terms, the windows become narrower at higher frequencies and the spectra lose frequency resolution. STFT features do not model the nonlinearity of human audio perception; thus, it is not an accurate representation from a psychoacoustical perspective.

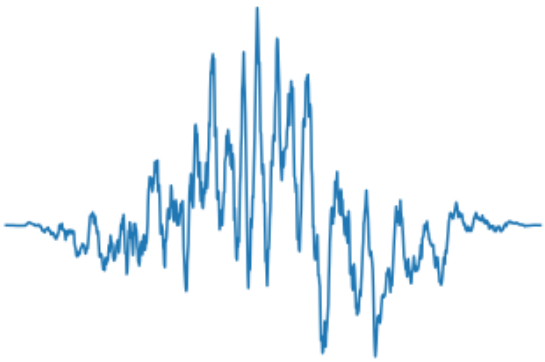
Works have been done to generate more perceptually motivated feature representations, one of which is the mel scale. The mel scale is a nonlinearly spaced set of filters that imitate the frequency



(a) Hann window

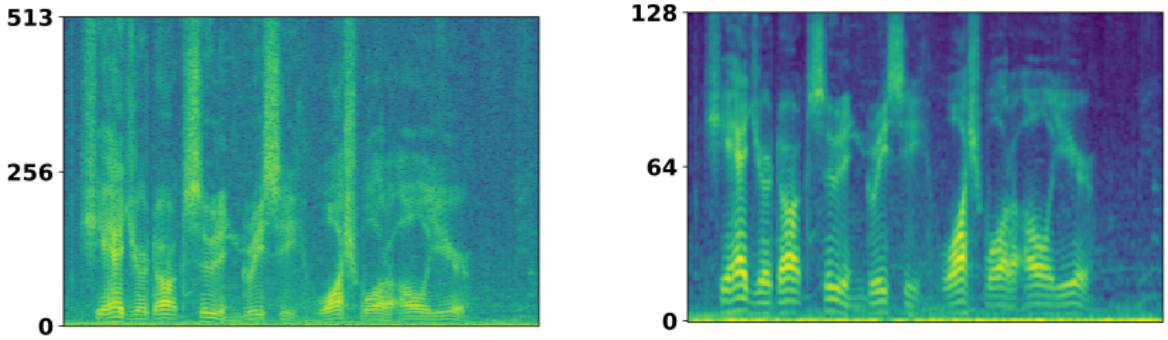


(b) Raw audio



(c) Windowed audio

Figure 2.7: Hann window applied on raw audio



(a) STFT spectrogram

(b) Mel spectrogram

Figure 2.8: STFT and corresponding Mel spectrogram

resolution of human hearing. It was measured by asking listeners to judge when a tunable pitch was perceived as half of that of a baseline tone. The results from the experiment were averaged to construct the pitch scale. This subjective scale measures pitches that are judged to be perceptually equal in distance from another [57, 58]. The conversion between frequency f into mels m can be formulated as

$$m = 2595 \log \left(1 + \frac{f}{700}\right) \tag{2.16}$$

It is apparent from Eqn. 2.16 that the perceptual scale aligns well with the prior of a nonlinear perceptual emphasis. By applying the mel scale on the power spectrogram of the STFT, the mel spectrogram can be constructed.

Not only is the mel spectrogram more perceptually motivated, it is also a compressed version of the STFT. It is an efficient 2D form that preserves the most perceptually important information, making it an optimized form for human auditory perception. Kernels trained in end-to-end configurations have also shown nonlinear frequencies that are similar to the mel scale, which confirms the rich representativeness of the mel features. Although trainable, the mel scale transformation is a preferred pre-processing step to extract rich non linear features because they are still not easily learned through data-driven approaches. Even with the compact and perceptually motivated qualities, one drawback of the mel transformation is it does not retain phase information, losing its

invertibility.

For this reason, for machine learning based source separation, the mel spectrogram has only been explored as the input to the model to estimate back to the STFT [59]. Denoising operations are performed to estimate the approximately clean mel spectrum first, followed by a super-resolution task back to the STFT from which is inverted back to audio. But the mel spectra has been a popular choice for other tasks, which rely on spectral analysis without the need for an inverse transformation.

Mask-based approach for source separation

Mask-based methods have been one of the dominating deep learning solutions for the source separation application. These solutions operate by estimating masks, either on the time-frequency (TF) representations [60, 61] or on the latent feature space [62, 2]. In this section, we focus on the former approach using STFT spectrograms.

Due to their spectral qualities and invertibility, the STFT representation has been widely used in source separation applications for denoising and speech enhancement. One of the source separation approaches formulate the problem as a binary time-frequency mask estimation, thereby modelling the task as an easier supervised binary classification job using the STFT magnitude as inputs and ideal binary masks (IBM) as targets. Masking based approaches have shown to perform better than direct spectral estimation due to the bounded targets making training easier compared to unbounded spectral targets.

IBMs are constructed from the premixed sources by classifying T-F bins as 1 where the magnitude of the desired source is stronger than that of the noise and 0 otherwise. The equation is written as follows

$$\boldsymbol{M}_{t,f} = \begin{cases} 1 & \text{if } |\boldsymbol{S}_{t,f}| > |\boldsymbol{N}_{t,f}| \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

where $\boldsymbol{S}$ is the magnitude spectrogram generated from the clean speech and $\boldsymbol{N}$ is from the noise.

Suppose $\boldsymbol{X}$ is the mixture spectrogram. Upon applying the mask onto a mixture spectrogram, $\boldsymbol{M} \odot \boldsymbol{X}$, only the bins with dominant clean sources remain; thereby, successfully removing noisy bins. The advantages of IBMs are their simplicity and resulting large speech intelligibility gains. But for their simplicity, binary masking typically produces residual noise in the denoised output.

Ideal ratio masks (IRM) are a softer version that are computed by taking the ratios of the sources to quantify the relative dominance over a T-F bin. IRMs outperform their binary counterpart in terms of objective intelligibility and quality metrics. This is likely because predicting ratio targets is less sensitive to estimation errors than predicting binary targets. The IRM formulation is given as

$$\boldsymbol{M}_{t,f} = \left(\frac{|\boldsymbol{S}_{t,f}^2|}{|\boldsymbol{S}_{t,f}^2| + |\boldsymbol{N}_{t,f}^2|} \right) \tag{2.18}$$

2.2.2 Image Classification

The image classification problem is one of the core problems in the computer vision domain [63]. The task consists of training a classifier to extract information from an input image to correctly predict its correspond label from a fixed set of categories. In the simplest supervised learning setup, a database consisting of a set of images with a label with a single category. Most notable solutions for image classification use convolutional neural networks for their success in learning hierarchies of spatially invariant features [64]. Solutions using convolution operations have not only achieved state-of-the-art accuracies for image classification tasks but also for applications in different domains as well.

2.3 Neural Network Architectures

Next, we give an overview of relevant neural network architectures, specifically recurrent neural networks (RNN), convolutional neural networks (CNN), and generative adversarial networks (GAN). These models were chosen for their widespread usage in various applications, and compatibility

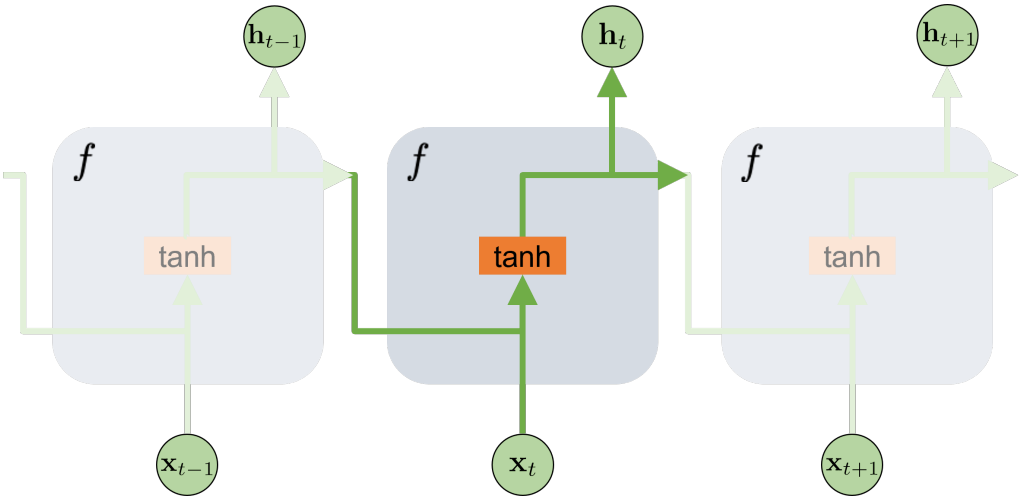


Figure 2.9: Basic RNN structure operating over an input sequence

with our compression algorithm.

2.3.1 Recurrent Neural Networks

Recurrent neural networks are a class of neural networks that have shown to be more effective in applications involving sequential or temporal data [65, 66]. The RNN is able to attain the superior performance in sequential tasks by utilizing a shared hidden state and gates within its hidden cells that guide the memory and learning over a sequence of inputs [4]. Also, the computational complexity of recurrent models are much lower compared to convolutional architectures. The recurrent relations take into account historical information of continuous inputs of various length while keeping the model size constant. Suppose the model is given an input sequence $\boldsymbol{x} \in \mathbb{R}^{T \times F}$ of length T and feature size F. In its basic form, RNN applies a function f on each input at timestep t as

$$\boldsymbol{h}(t) = f(\boldsymbol{h}(t-1), \boldsymbol{x}(t)) \tag{2.19}$$

where $\boldsymbol{h}(t)$ is the current hidden state. The most practical method to train RNNs is with truncated Backpropagation Through Time (BPTT) [67]. This bounded-history approximation method simplifies computation by limiting itself to a fixed scope of T timesteps [68]. Due to the com-

plex recurrent computations, RNNs are susceptible to a vanishing and explosive gradient problem that make it difficult to capture long term dependencies. Standard RNNs have shown difficulties in learning dependencies between samples larger than 5-10 steps [69]. Long Short-Term memory (LSTM) overcomes the vanishing gradient problem by enabling consistent gradient within the cells using a set of multiplicative gates. Gated Recurrent Unit (GRU) is another popular variant of RNNs that can address the same problems using a more efficient architecture.

2.3.2 Convolutional Neural Networks

A convolutional neural networks (CNN) are commonly used for classification and computer vision tasks. CNNs are composed of stacks of modules, primarily the convolution and pooling operation. The primary convolution operation extracts tiles from an input feature map and applies filters (or kernels) to compute an output feature map. For a single channel 2D input $\boldsymbol{X}$ and 2D kernel $\boldsymbol{W} \in \mathbb{R}^{(2k+1)\times(2k+1)}$, the convolution operator $\boldsymbol{Y} = \boldsymbol{W} * \boldsymbol{X}$ is formulated as

$$\boldsymbol{Y}[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \boldsymbol{W}[u,v]\boldsymbol{X}[i-u, j-v] \tag{2.20}$$

In a convolutional layer, the filters slide over the input feature map horizontally and vertically until all pixels have been visited. During training, the CNN learns the optimal values for the filter weights as to extract meaningful features (e.g. textures, edges, shapes) from the inputs. The number of filters determines the depth, or number of channels, of the output feature map. Using more filters or layers helps the CNN extract more features, but the convolution operation can quickly become computationally intensive. The pooling module downsamples the learned representations and is commonly placed in between convolutional layers. By reducing the resolution of the feature maps, pooling layers aim to preserve the spatial relationship between pixels and captures the local dependencies. This allows CNNs to achieve translational invariance and become robust to shifting positions of vital features regardless of their positions in an image.

2.3.3 Generative Adversarial Networks

Generative Adversarial Networks have gained in popularity in various domains recently due to their high quality outputs. Generative models trained with an adversarial framework have shown to be capable of synthesizing high resolution images, sequences of realistic text from diverse range of topics, and natural sounding time-domain waveforms [70, 71].

The ability to create high quality realistic samples originates from GAN's training framework in which two models compete in the form of a minimax optimization. Generative adversarial training framework is based on a two-player minimax game between two neural nets, generator $\boldsymbol{G}$ and discriminator $\boldsymbol{D}$ [8, 72]. Given a prior sampled from a noise distribution $\boldsymbol{z} \sim p_{\boldsymbol{z}}$, the generator synthesizes fake examples, $\boldsymbol{G}(\boldsymbol{z})$, whose distribution p_g attempts to be close to the ground-truth data distribution p_{data}. The discriminator network $\boldsymbol{D}$ evaluates an input $\boldsymbol{x}$ and outputs a value, $\boldsymbol{D}(\boldsymbol{x}) \in \mathbb{R}$, representing the probability of the input being sampled from the true data distribution. The discriminator model is trained to distinguish between real and fake samples while the generator tries to produce synthetic samples that can fool the discriminator to classify as being generated from a true probability distribution. The generator uses the discriminator's output as a loss function to estimate the probability distribution of real samples such that the two probability distribution of real and synthetic samples are drawn closer. Fooling the discriminator as an objective can be a better metric to guide generative models for producing realistic examples than existing hand engineered metrics that may produce artifacts, rendering synthesized samples unrealistic [73]. The discriminators can also be guided to learn the surface of the desired metrics themselves as a surrogate loss function [74]. In the original GAN formulation, the adversarial training process is formulated as a minimax game $\min_{G} \max_{D} L(\boldsymbol{D}, \boldsymbol{G})$ where

$$L(\boldsymbol{D}, \boldsymbol{G}) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log \boldsymbol{D}(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - \boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z})))] \tag{2.21}$$

in which the goal of the discriminator is to maximize the likelihood of predicting the correct label

to the inputs to maximize its reward $L(\boldsymbol{D}, \boldsymbol{G})$ while the generator aims to produce examples close to the true data manifold and minimize the Discriminator's reward.

Both $\boldsymbol{G}$ and $\boldsymbol{D}$ are differentiable networks and can be optimized using gradient-based updates to their parameters. In contrast to gradient-descent based optimization methods which seek to minimize a loss function until some convergence criteria is met, GANs seek to reach an equilibrium point between both models. The adversarial framework continuously optimizes both networks against each other, allowing generators to synthesize high quality outputs; however, the competitive optimization design introduces degenerate behaviors such as mode collapse and vanishing gradients that make training unstable. Since the parameters weights of both models are constantly changing, their optimization trajectory has to chase to the changing objective. These unstable learning dynamics have led GAN to be known to be notorious difficult to train. Advances have been made over the original GAN framework to stabilize training. Several GAN variants have been proposed (e.g. Wasserstein GANs) but their success has only been contained to specific problem settings and have not been universally applied [75].

Chapter 3

Bitwise Gated Recurrent Units

3.1 Introduction

In this section, we propose an extreme network compression method using binarization, where real-value parameters are reduced to 1-bit precision. Binarization has been previously explored as a method of network compression. BinaryConnect [23], binarized neural networks [76], trained ternary quantization [28], and Bitwise Neural Networks (BNN) [26] have implemented a binarized or ternarized neural network in bipolar binaries (with zeros in the ternarized case) for network compression. They emphasize that replacing real-valued operations with bitwise versions greatly reduces the network's complexity. In particular, the BNN training process is assisted by initializing the binarized network with pretrained weights. The weights are compressed in a real-valued network with the hyperbolic tangent activation function in order to better approximate their binary versions. Further quantization is performed in the BNN, where the inputs are quantized using Quantization-and-Disperson, which uses Lloyd-Max's quantization to convert each frequency magnitude of the noisy input spectrum into 4 bits with bipolar binary features [77].

In the domain of source separation, BNN's have been applied by predicting Ideal Binary Masks (IBM) as target outputs [27]. We extend this work on BNNs aim at compressing a recurrent neural network (RNN) architecture. This chapter proposes a Bitwise Gated Recurrent Unit (BGRU) network for the single-channel source separation task.

3.2 Proposed Incremental Binarization Algorithm

For a single feedforward step, the RNN requires multiple sets of weights and performs operations in Eq. 3.1 for T timesteps. With deeper RNNs, the computational cost rises rapidly in terms of K

and L. To mitigate this increased computation, we focus on the Gated Recurrent Unit (GRU) cells and quantize the feedforward procedure with binarized values and bitwise operations. The model we propose is a BGRU network that reduces network complexity by re-defining the originally real-valued inputs and outputs, weights, and operations in a bitwise fashion. By limiting the network to bipolar binary values, the space complexity of the network can be significantly reduced. In addition, all real-valued operations during the feedforward procedure can be replaced with bitwise logic, which further reduces both spatial and time complexity [76, 29].

3.2.1 Gated Recurrent Units

An efficient cell structure that is robust to the gradient vanishing problem is the GRU cell [78]. The computation within each GRU cell is:

$$\begin{aligned}
\boldsymbol{r}^{(l)}(t) &= \sigma\Big(\boldsymbol{W}_r^{(l)}\boldsymbol{X}^{(l-1)}(t) + \boldsymbol{U}_r^{(l)}\boldsymbol{h}^{(l)}(t-1)\Big) \\
\boldsymbol{z}^{(l)}(t) &= \sigma\Big(\boldsymbol{W}_z^{(l)}\boldsymbol{X}^{(l-1)}(t) + \boldsymbol{U}_z^{(l)}\boldsymbol{h}(t-1)\Big) \\
\tilde{\boldsymbol{h}}^{(l)}(t) &= \phi\Big(\boldsymbol{W}_h^{(l)}\boldsymbol{X}^{(l-1)}(t) + \boldsymbol{U}_h^{(l)}\big(\boldsymbol{r}^{(l)}(t) \odot \boldsymbol{h}^{(l)}(t-1)\big)\Big) \\
\boldsymbol{h}^{(l)}(t) &= \boldsymbol{z}^{(l)}(t) \odot \boldsymbol{h}^{(l)}(t-1) + (1 - \boldsymbol{z}^{(l)}(t)) \odot \tilde{\boldsymbol{h}}^{(l)}(t)
\end{aligned} \tag{3.1}$$

where $l = \{1, \ldots, L+1\}$ denotes the layer index, $t = \{1, \ldots, T\}$ is the time index, and $\odot$ is the elementwise multiplication operator. $\boldsymbol{r}_t, \boldsymbol{z}_t, \tilde{\boldsymbol{h}}_t$, and $\boldsymbol{h}_t$ are reset gate, update gate, candidate hidden state, and updated hidden state respectively all of dimension $\mathbb{R}^{K^{(l)}}$ with $K^{(l)}$ as the number of units at layer l. $\boldsymbol{W}_r^{(l)} \in \mathbb{R}^{K^{(l)} \times K^{(l-1)}}$ and $\boldsymbol{U}_r^{(l)} \in \mathbb{R}^{K^{(l)} \times K^{(l)}}$ are the weight matrices for the input $\boldsymbol{X}^{(l)}(t)$ and previous hidden state $\boldsymbol{h}^{(l)}(t-1)$ at the reset gate. Similarly, $\boldsymbol{W}_z$, $\boldsymbol{U}_z$, $\boldsymbol{W}_h$, and $\boldsymbol{U}_h$ are corresponding weights for the update gate and candidate state. The σ and ϕ refer to the logistic sigmoid and hyperbolic tangent activation functions. The bias term is omitted for simplicity. Note that $\boldsymbol{h}^{(l)}(t)$ is fed to the next layer as an input, $\boldsymbol{X}^{(l)}(t)$.

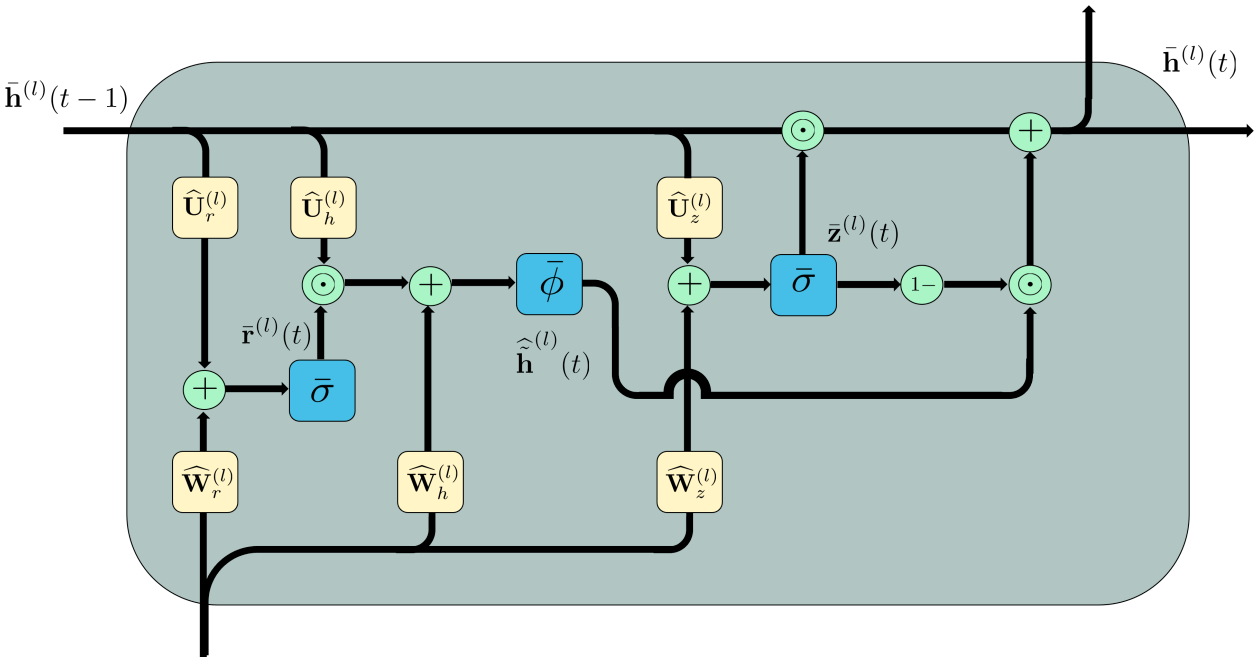


Figure 3.1: Illustration of the BGRU cell

3.2.2 Feedforward in BGRU

Notation and setup

For the following sections of the paper, we specify discrete variables with a bar notation, i.e. $\bar{x}$. Depending on the context, this could be a binary variable with 0 and 1 (e.g. gates), a bipolar binary variable with $+1$ and -1 (e.g. binarized hidden units), or a ternary variable (e.g. sparse bipolar binary weights). The binary versions of logistic sigmoid and hyperbolic tangent activation functions are:

$$\bar{\sigma}(x) = \frac{sgn(x)+1}{2} \in \{0,1\},\ \bar{\phi}(x) = sgn(x) \in \{-1,+1\} \tag{3.2}$$

respectively where $sgn(x)$ is a sign function [76, 26].

The feedforward procedure

In the BGRU, the feedforward process is defined as follows:

$$\begin{aligned}
\bar{\boldsymbol{r}}^{(l)}(t) &= \bar{\sigma}\Big(\bar{\boldsymbol{W}}_r^{(l)}\bar{\boldsymbol{X}}^{(l-1)}(t) + \bar{\boldsymbol{U}}_r^{(l)}\bar{\boldsymbol{h}}^{(l)}(t-1)\Big) \\
\bar{\boldsymbol{z}}^{(l)}(t) &= \bar{\sigma}\Big(\bar{\boldsymbol{W}}_z^{(l)}\bar{\boldsymbol{X}}^{(l-1)}(t) + \bar{\boldsymbol{U}}_z^{(l)}\bar{\boldsymbol{h}}^{(l)}(t-1)\Big) \\
\bar{\tilde{\boldsymbol{h}}}^{(l)}(t) &= \bar{\phi}\Big(\bar{\boldsymbol{W}}_h^{(l)}\bar{\boldsymbol{X}}^{(l-1)}(t) + \bar{\boldsymbol{U}}_h^{(l)}\big(\bar{\boldsymbol{r}}^{(l)}(t) \odot \bar{\boldsymbol{h}}^{(l)}(t-1)\big)\Big) \\
\bar{\boldsymbol{h}}^{(l)}(t) &= \bar{\boldsymbol{z}}^{(l)}(t) \odot \bar{\boldsymbol{h}}^{(l)}(t-1) + (1 - \bar{\boldsymbol{z}}^{(l)}(t)) \odot \bar{\tilde{\boldsymbol{h}}}^{(l)}(t)
\end{aligned} \tag{3.3}$$

The product between two binarized values (e.g. between the (i,j)-th element of $\bar{\boldsymbol{W}}_r^{(l)}$ and the j-th element of $\bar{\boldsymbol{X}}^{(l-1)}(t)$) is equivalent to the XNOR operations, a cheaper binary operation than the corresponding floating-point operation. Also, the use of sign functions $\bar{\phi}$ and a hard step function $\bar{\sigma}$ in place of the hyperbolic tangent and sigmoid functions also expedite the process because they can be usually implemented by a pop counter.

3.2.3 Scaled Sparsity and Bernoulli Masks

Scaled Sparsity Masks

While the BNN significantly reduces the space and time complexity of the network, the conversion from real-values to bipolar binaries inevitably produces quantization error. One method to lessen this penalty is to introduce sparsity to zero out a portion of pretrained weights with small values close to 0. A pre-determined threshold can be used to define the boundary determining the sparsity, as in Eq. 2.5. For example, Fig. 3.2a displays real-valued weights sampled from a GRU cell. The distribution of the real-valued weights are all relatively close to 0, and quantizing all values into bipolar binary values would be too intensive of a tranformation. This loss of precision can be addressed by introducing sparsity, but only to some degree (Fig. 3.2b). The non-zero portions are still closer to 0 and converting them to bipolar binaries would be a poor approximation. Another way to mitigate the quantization error is by multiplying a scaling factor μ to the bipolar-binarized weights, so that the quantized values approximate the original values more closely [28]. We compute μ as the mean of the non-zero values, which we then multiply with the remaining nonzero bipolar

values. Thus, the values are scaled down to the average value of the non-sparse portion, which is a better representative for the nonzero elements (Fig. 3.2c).

Scaled sparsity can be implemented in a form of a mask that is directly applied onto the weights. The scaled sparsity mask is a two-in-one solution to introduce both scaling parameters and sparsity into weights during the binarization process. The scaled sparsity mask $\boldsymbol{B}$ is created using a predefined sparsity parameter $0 < \rho < 1$. First, we find a per-layer cutoff value β and the scaling parameter μ that meet the following equations:

$$\begin{aligned} &\mathcal{I} = \{(i,j) : |\boldsymbol{W}_{i,j}^{(l)}| > \beta\}, \quad |\mathcal{I}| = K^{(l-1)}K^{(l)}\rho \\ &\mu = \frac{1}{|\mathcal{I}|} \sum_{(i,j)\in\mathcal{I}} |\boldsymbol{W}_{i,j}^{(l)}|, \end{aligned} \tag{3.4}$$

where $\mathcal{I}$ is the set of weight indices whose absolute values are larger than the cutoff value and $|\mathcal{I}|$ denotes the number of such weights. Therefore, for a given sparsity value ρ, we first sort the weights in their absolute values and then find the cutoff that results in $\mathcal{I}$ with the predefined size. Using β and μ, we set the mask elements as follows:

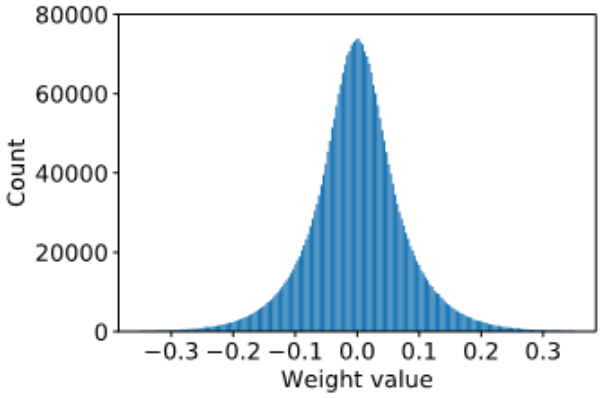
$$\boldsymbol{B}_{i,j} = \begin{cases} \mu \ \text{ if } \ |\boldsymbol{W}_{i,j}| > \beta \\ 0 \quad \text{otherwise} \end{cases} \tag{3.5}$$

The created mask is applied on weights $\boldsymbol{W}$ to create the scaled and sparse binarized matrix $\widehat{\boldsymbol{W}}$:

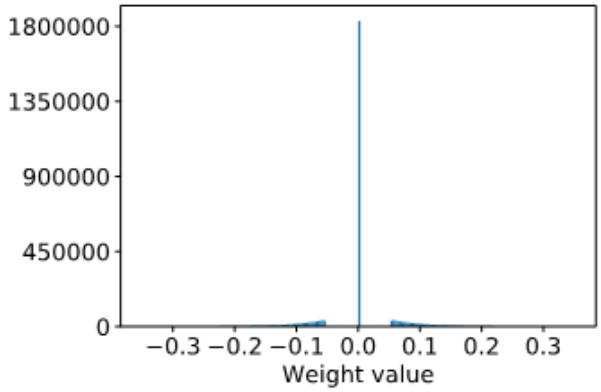
$$\widehat{\boldsymbol{W}} = \bar{\phi}(\boldsymbol{W}) \odot \boldsymbol{B} \tag{3.6}$$

Bernoulli Binarization Masks

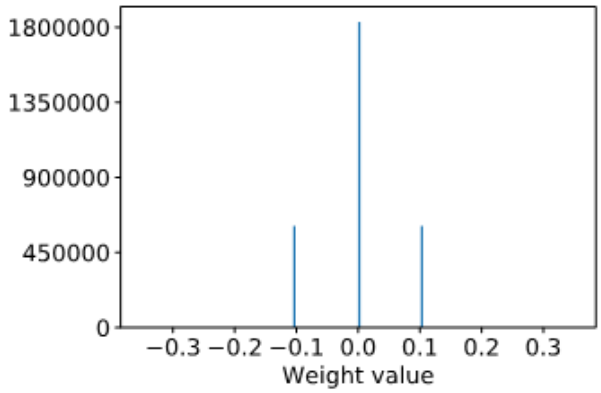
To further alleviate the heavy quantization loss from transforming real-valued weights into bipolar binaries, the weights are converted into binary values through a gentle training procedure. We introduce an incremental binarization method to minimize the potential loss that can occur from a sudden introduction of quantization. We setup another type of mask as a random Bernoulli matrix $\boldsymbol{C}$ with a parameter $0 < \pi < 1$ as the amount of binarization. Each element of $\boldsymbol{C}$ is initialized by



(a) Real-valued GRU weights



(b) With desired sparsity $\rho = 0.8$



(c) With scaled sparsity

Figure 3.2: Visualization of scaled sparsity applied on real-valued weights within a GRU cell

drawing a random binary number from a Bernoulli distribution according to the probability value given in π as $\boldsymbol{C}_{i,j} = \text{Bernoulli}(p = \pi)$. The value of π is initially chosen as a small value (e.g. 0.1 for 10% binarization) and gradually increased up to 1.0, which means the network is completely binarized. Both masks are applied on weights $\boldsymbol{W}$ to create the partly binarized matrix $\widehat{\boldsymbol{W}}$:

$$\widehat{\boldsymbol{W}} = \big(\bar{\phi}(\boldsymbol{W}) \odot \boldsymbol{B}\big) \odot \boldsymbol{C} + \phi(\boldsymbol{W}) \odot (1 - \boldsymbol{C}). \tag{3.7}$$

The purpose of $\boldsymbol{B}$ with μ values is to lessen the quantization error from the binarization. The $\bar{\phi}$, a sign function, will transform all values into bipolar binary values, which would be too intensive of a transformation because the distribution of the first round weights are all relatively close to 0. Thus, by multiplying the remaining nonzero bipolar values after applying sparsity with μ, the values are scaled down to the average value of the non-sparse portion, which is a better representative for the nonzero elements. Note that feedforward is still bitwise thanks to the symmetry of $\boldsymbol{B}$ and by skipping zeros.

The Bernoulli mask $\boldsymbol{C}$ enables a gradual transition from real-valued weights and operations to bitwise versions. This mask is applied on the bitwise and real-valued elements in a complementary way to control the proportion of binarization in the network. $\widehat{\boldsymbol{W}}$ in (3.7) is binarized only partly with the proportion set by π.

$\boldsymbol{C}$ is used to control the binarization of the other network elements such as gates and hidden units, too. Suppose we define the bitwise candidate hidden units $\bar{\tilde{\boldsymbol{h}}}$ as

$$\bar{\tilde{\boldsymbol{h}}}^{(l)}(t) = \bar{\phi}\Big(\bar{\boldsymbol{W}}_h^{(l)} \bar{\boldsymbol{X}}^{(l-1)}(t) + \bar{\boldsymbol{U}}_h^{(l)}\big(\bar{\boldsymbol{r}}^{(l)}(t) \odot \bar{\boldsymbol{h}}^{(l)}(t-1)\big)\Big) \tag{3.8}$$

which is equivalent to that of Eq. 3.1 except with the sign function. Then its activations are performed as:

$$\widehat{\bar{\tilde{\boldsymbol{h}}}} = \bar{\tilde{\boldsymbol{h}}} \odot \boldsymbol{C} + \tilde{\mathbf{h}} \odot (1 - \boldsymbol{C}), \tag{3.9}$$

The gates are also partially binarized in this way. The mask $\boldsymbol{C}$ is generated at each iteration for the weights as in (3.7) and then at each timestep for the activation functions of GRU cells as in

(3.9). This ensures that the gradients of the bitwise terms are evenly distributed gradually for all weights at each levels of π. Without even distribution, certain elements of the graph that do not participate in the bitwise procedure begin focusing on compensating for the quantization loss from the other bitwise elements. This needs to be avoided since as π is increased to 1.0 these elements need to be quantized eventually.

Using the Bernoulli mask, the proposed binarization technique turns only a few randomly chosen parameters into their binary versions, thereby giving the network training procedure a chance to gently adapt to the partly quantized version of the network. It eventually achieves the full binarization by incrementally increasing the amount of binarization over the iterations.

3.2.4 Training BGRU Networks

The objective is to accept binarized mixture signal inputs and predict the corresponding IBMs. There are various approaches for input binarization. In this study, we use Quantization-and-Dispersion (QaD) using Lloyd-Max's quantization to convert individual TF coefficients from STFT magnitude inputs $\boldsymbol{X} \in \mathbb{R}^{T \times F}$ into 4 bits to produce its corresponding binary variant $\bar{\boldsymbol{X}} \in \mathbb{R}^{T \times 4F}$. [77] (Fig. 3.3). The target outputs are bipolar binary IBM predictions which are later converted to 0's and 1's for the source separation task. We follow the typical two-round training scheme from BNNs [27].

First round: Pretraining ϕ-compressed weights

The GRU network is first initialized with real-valued weights and then trained on quantized binary inputs. As in [27], during training the weights are wrapped with the hyperbolic tangent activation function, ϕ. This has the effect of bounding the range of weights between -1 and $+1$ as well as regularization. In the second round, the sign function, $\bar{\phi}$ is applied on the weights instead, hence the first round network can be perceived as its softer version. For example, the feedforward procedure

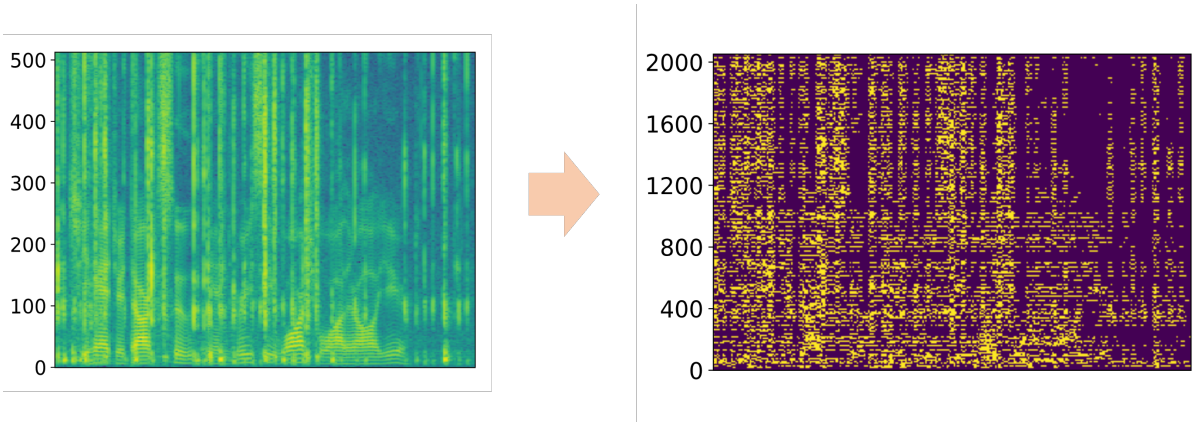


Figure 3.3: Illustration of input binarization on sample STFT magnitude spectrogram using QaD

in (3.1) for only the hidden candidate state at layer l and timestep t becomes:

$$\tilde{\boldsymbol{h}}^{(l)}(t) = \phi\Big(\phi\big(\boldsymbol{W}_h^{(l)}\big)\bar{\boldsymbol{X}}^{(l-1)}(t) + \phi\big(\boldsymbol{U}_h^{(l)}\big)\big(\bar{\boldsymbol{r}}^{(l)}(t) \odot \bar{\boldsymbol{h}}^{(l)}(t-1)\big)\Big) \tag{3.10}$$

The ϕ-compressed weights are applied similarly for the reset and update gates.

Backpropagation: With the introduction of ϕ on the weight matrices, the derivative with respect to ϕ is added onto the backpropagation due to the chain rule. For example, the gradients for (3.10) are computed as:

$$\begin{aligned}
\boldsymbol{\delta}_{\tilde{\boldsymbol{h}}}(t) &= \boldsymbol{\delta}^{(l)}(t) \odot (1 - \boldsymbol{z}(t)) \odot \big(1 - \tilde{\boldsymbol{h}}^{(l)}(t)^2\big) \\
\nabla \boldsymbol{W}_h^{(l)} &= \Big(\sum_{t=0}^{T} \boldsymbol{\delta}_{\tilde{\boldsymbol{h}}}(t) \cdot \big(\bar{\boldsymbol{X}}^{(l-1)}(t)\big)^\top\Big) \odot \Big(1 - \phi^2\big(\boldsymbol{W}_h^{(l)}\big)\Big) \\
\nabla \boldsymbol{U}_h^{(l)} &= \Big(\sum_{t=1}^{T} \boldsymbol{\delta}_{\tilde{\boldsymbol{h}}}(t) \cdot \big(\bar{\boldsymbol{r}}^{(l)}(t) \odot \bar{\boldsymbol{h}}^{(l)}(t-1)\big)\Big) \odot \Big(1 - \phi^2\big(\boldsymbol{U}_h^{(l)}\big)\Big)
\end{aligned} \tag{3.11}$$

where $\boldsymbol{\delta}^{(l)}(t)$ is the backpropagation error for the training sample at layer l and timestep t. The gradients are similarly defined for the weights in the gates.

Second round: BGRU

The BGRU network is initialized with the real-valued weights from the first round, which are pretrained to be optimal for the source separation task. The real-valued weights are saved for the backpropagation step and used to construct bitwise weights for the feedforward procedure using

both the mean-scaled sparsity mask $\boldsymbol{B}$ and Bernoulli mask $\boldsymbol{C}$. The bitwise activation functions, $\bar{\sigma}$ and $\bar{\phi}$ are applied during the feedforward as well. Again as an example, with the introduction of the masks and bitwise functions, the feedforward step for the hidden candidate state becomes:

$$\begin{aligned}
\widehat{\boldsymbol{W}}_h^{(l)} &= (\bar{\phi}(\boldsymbol{W}_h^{(l)}) \odot \boldsymbol{B}) \odot \boldsymbol{C} + \phi(\boldsymbol{W}_h^{(l)}) \odot (1 - \boldsymbol{C}) \\
\widehat{\boldsymbol{U}}_h^{(l)} &= (\bar{\phi}(\boldsymbol{U}_h^{(l)}) \odot \boldsymbol{B}) \odot \boldsymbol{C} + \phi(\boldsymbol{U}_h^{(l)}) \odot (1 - \boldsymbol{C}) \\
\boldsymbol{V} &= \widehat{\boldsymbol{W}}_h^{(l)} \bar{\boldsymbol{X}}^{(l-1)}(t) + \widehat{\boldsymbol{U}}_h^{(l)} \big(\bar{\boldsymbol{r}}^{(l)}(t) \odot \bar{\boldsymbol{h}}^{(l)}(t-1)\big) \\
\widehat{\tilde{\boldsymbol{h}}}^{(l)}(t) &= \bar{\phi}(\boldsymbol{V}) \odot \boldsymbol{C} + \phi(\boldsymbol{V}) \odot (1 - \boldsymbol{C})
\end{aligned} \tag{3.12}$$

where $\boldsymbol{V}$ is an intermediary term. The Bernoulli parameter π is incremented gradually to determine $\boldsymbol{C}$ until the network is completely binarized at $\pi = 1.0$.

Backpropagation: To deal with the non-differentiable nature of the optimization (e.g., sign function) the derivatives of non-differentiable activation functions are relaxed overwritten with the derivatives of their soft counterparts such that $\bar{\phi}' = \phi'$ and $\bar{\sigma}' = \sigma'$. This simplifies the gradients for (3.12). The gradients are computed as (3.11) with an additional factor for the masks which are:

$$\begin{aligned}
\nabla \boldsymbol{W}_h^{(l)} &= \nabla \boldsymbol{W}_h^{(l)} \odot (\boldsymbol{B} \odot \boldsymbol{C} + (1 - \boldsymbol{C})) \\
\nabla \boldsymbol{U}_h^{(l)} &= \nabla \boldsymbol{U}_h^{(l)} \odot (\boldsymbol{B} \odot \boldsymbol{C} + (1 - \boldsymbol{C}))
\end{aligned} \tag{3.13}$$

The gradients are computed similarly for the gates. The calculations in (3.13) show that the network is the same as the first round network except with the addition of masking factors. Only the real-valued weights are updated with the gradients during training.

3.3 Experiments

For the experiment, we randomly subsample 12 speakers for training and 4 speakers for testing from the TIMIT corpus [79]. For both subsamples, we select half of the speakers as male and the other half as female. There are 10 short utterances per speaker recorded with a 16kHz sampling rate. Each utterances are mixed with 10 different non-stationary noise signals with 0 dB Signal-to-Noise Ratio (SDR), namely {birds, casino, cicadas, computer keyboard, eating chips, frogs, jungle,

machine guns, motorcycles, ocean} [80]. In total, we have 227,580 training examples and 81,770 test examples from 1,200 and 400 mixed utterances, respectively. We apply a Short-Time Fourier Transform (STFT) with a Hann window of 1024 and hop size of 256. To quantize the spectra into bipolar binaries, we apply a 4-bit QaD procedure and convert them into $n \times (4 \times 513)$ dimension matrices. These vectors are used as inputs to the BGRU systems. The truncated BPTT length used was $T = 50$. We found $\rho = 0.8$ to perform well in our experiment. We used the Adam optimizer for both first and second rounds with the same beta parameters, $\beta_1 = 0.4$ and $\beta_2 = 0.9$. Minibatch size is set as 10 for 10 mixed utterances constructed from 1 clean signal mixed with the 10 noise signals. We train two types of networks that predict the IBMs with respect to the noisy quantized input:

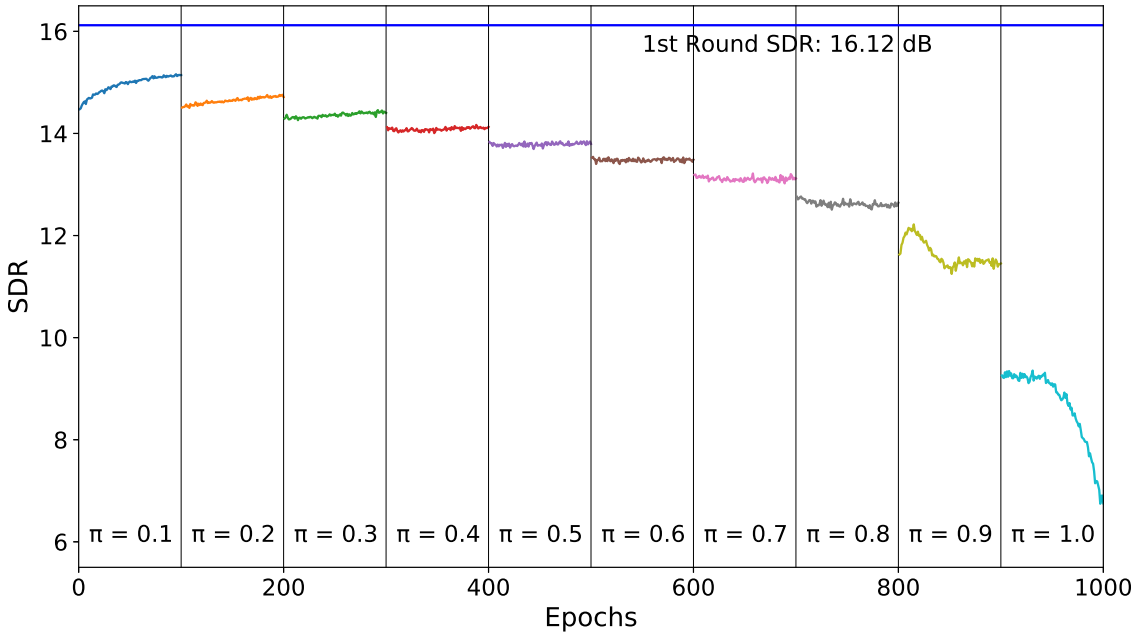
- *Baseline with binary input*: The baseline network is constructed with a single GRU layer with $K = 1024$ units. The inputs to the network are 4×513 dimension 4-bit QaD vectors and predicted outputs are 513 dimension IBMs. We use the first round training algorithm to train the baseline network. For regularization, we apply dropout rate of 0.05 for the input layer and 0.2 for the GRU layers.

- *The proposed BGRU*: We initialize the weights with the pretrained weights and use the second round training algorithm to train the BGRU network. We increase the π parameter by 0.1 starting from 0.1 to 1.0. The learning rates are reduced for each increase in π.

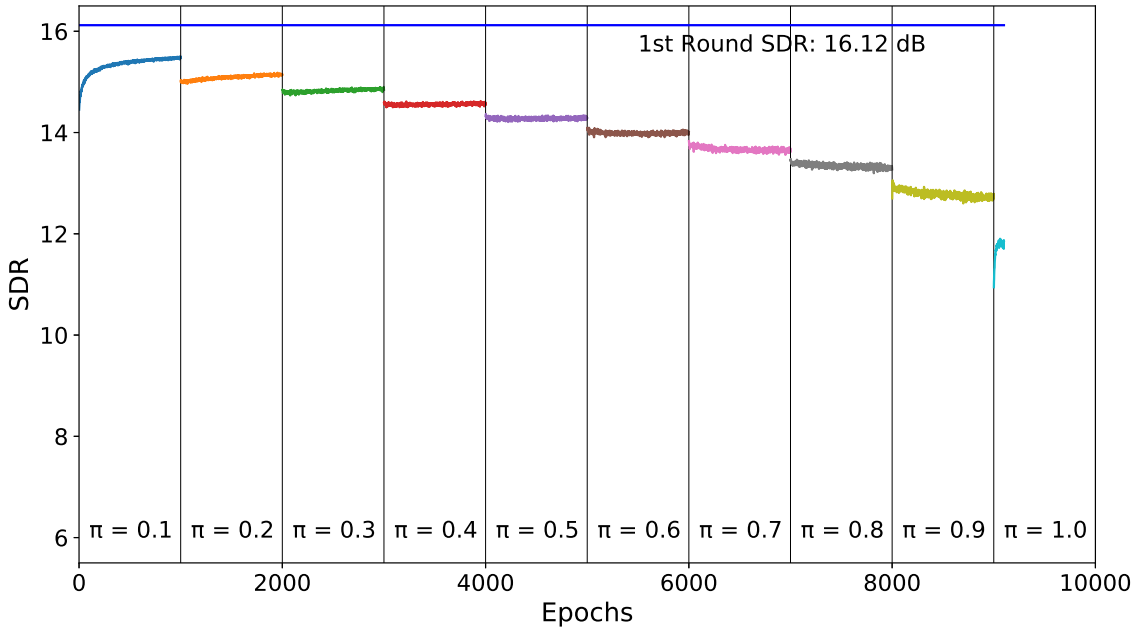
3.4 Discussion

Table 3.1 shows results for the BGRU along with other systems for comparison. The metrics displayed are Signal-to-Distortion Ratio (SDR) [81] and Short-Time Objective Intelligibility (STOI) [82]. At each increase in π, there is a distinct drop in SDR and STOI due to the loss in information as we increase the number of elements undergoing binarization.

Since the initial weights transferred from the first round are optimal, we restrict the weights from



(a) Results from 100 epochs for each π



(b) Results from 1000 epochs for $\pi < 1.0$ and 100 epochs for $\pi = 1.0$

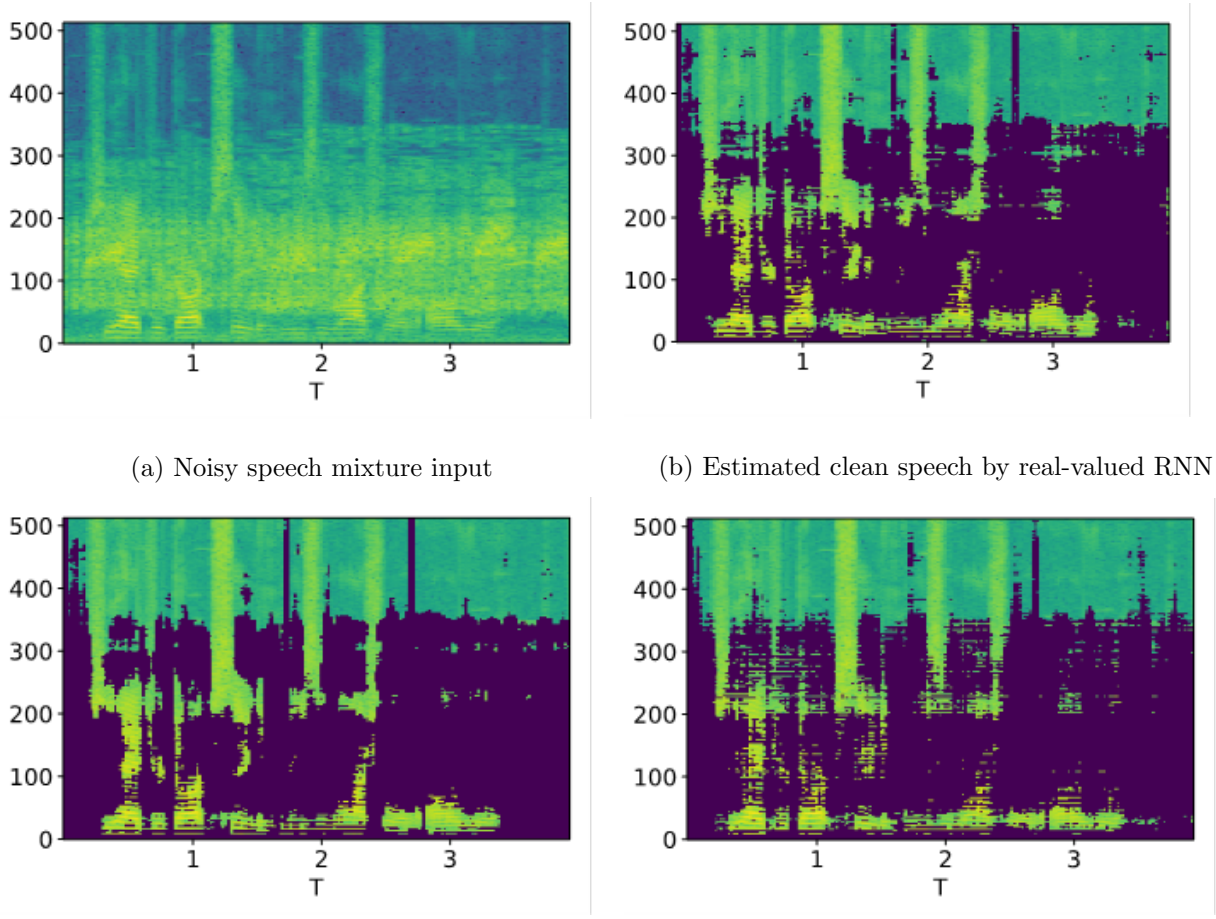
Figure 3.4: Second round testing results on incremental levels of π. Figures a and b show the effects of running different number of iterations.

Table 3.1: Speech denoising performance of the proposed BGRU-based source separation model compared to FCN, BNN, and GRU networks

Systems		Topology	SDR	STOI
FCN with original input		1024×2	10.17	0.7880
		2048×2	10.57	0.8060
FCN with binary input		1024×2	9.80	0.7790
		2048×2	10.11	0.7946
BNN		1024×2	9.35	0.7819
		2048×2	9.82	0.7861
GRU with binary input		1024×1	16.12	0.9459
BGRU	π=0.1	1024×1	15.50	0.9393
	π=0.2		15.17	0.9361
	π=0.3		14.90	0.9324
	π=0.4		14.58	0.9292
	π=0.5		14.32	0.9252
	π=0.6		14.02	0.9217
	π=0.7		13.66	0.9174
	π=0.8		13.30	0.9104
	π=0.9		12.70	0.9019
	π=1.0		11.76	0.8740

updating too drastically by dampening the learning rate at each increase in π. We did not observe substantial difference from reducing the learning rate before $\pi = 0.8$, however the performance becomes sensitive as the rate of binarization nears 1. In Figure 3.4b it can be seen that from $\pi = 0.8$ the performance begins to decrease more than during previous π values.

The BGRU network is trained for an extended number of iterations so it propagates the corrections and adjusts to the quantization injected into the network. We trained 1000 epochs for each π values except at $\pi = 1.0$. Figure 3.4b shows that this many iterations is not always beneficial



(a) Noisy speech mixture input

(b) Estimated clean speech by real-valued RNN

(c) Denoised output from partial BGRU with $\pi = 0.5$

(d) Denoised output from BGRU with $\pi = 1.0$

Figure 3.5: Effects of incremental binarization portrayed on speech denoising task

within the same session with a fixed π, because SDR improvement becomes stagnant and even starts to drop. However, in this way the network can prevent a greater drop in performance at the next increase in π. At $\pi = 1.0$, we only train for 100 epochs and perform early stopping because the network is less robust and degrades in performance after more than 100 epochs. Also, since the network has finished training for the source separation task at $\pi = 1.0$, further training is unnecessary. On the contrary, Figure 3.4a shows that training for less number of iterations, e.g. 100 epochs, produces a greater drop at each increment of π.

Fig. 3.5 illustrates the effects of incremental binarization. From the figures, we can observe that the binarization process injects significant amount of quantization noise during the incremental

procedure. Given a noisy mixture input STFT magnitude spectrogram (Fig. 3.5a), we compute the baseline results using a real-valued pre-trained RNN (Fig. 3.5b). A snapshot at $\pi = 0.5$ during incremental binarization training can be observed in Fig. 3.5c. The resulting denoised output from the partially binarized model appear less sparse and smoother compared to the estimate from the real-valued version, which indicates binarization resulting in loss of granular details that represent clean speech elements. This can be observed in the higher frequency regions ($f > 350$) especially around $T = [0.5, 1]$ and also in frequency bins $f = [100, 200]$ around $T = 1.5$. Next, the fully binarized version at $\pi = 1.0$ is shown in Fig. 3.5d. First, we can observe that the model is susceptible to making incorrect predictions in the higher frequency regions. For instance, at $T \sim .5$ the portions retained by the real-valued model have been removed, which indicates removal of clean speech regions. Moreover, the denoised result from the fully binarized model shows horizontal streaks dispersed around granular output regions from the real-valued model. Specifically, around frequency regions $f = [100, 200]$ at $T \sim 1.5$, the horizontal streaks render the region resembling a vague mosaic of the corresponding output regions produced by the real-valued model.

The drop in performance from a real-valued network to a bitwise version is quite comparable between a FCN with BNN and GRU with BGRU. The loss is much greater in the BGRU network (16.12 dB to 11.76 dB SDR) than in the case of BNN (10.11 dB to 9.82 dB SDR). Yet, the performance of a single-layer fully bitwise BGRU network with 1024 units (11.76 dB SDR and 0.8740 STOI) is still greater than that of a double-layer BNN with 2048 units (9.82 dB SDR and 0.7861 STOI), and also greater than that of a unquantized double-layer FCN with real-valued inputs and 2048 units (10.57 dB SDR and 0.8060 STOI). We discuss the space complexity of the BGRU network compared to a FCN and BNN. Considering that a GRU layer contains 3 sets of weights, the single layer BGRU network contains $3 \times (1024 \times 1)$ number of weights. This number is still less than a FCN or BNN of topology 2048×2. We introduced a real-valued scaling factor μ, but it reduces down to bipolar binaries once training is done, so it does not add additional costs.

In the future, we plan to extend the network structure to deeper ones. Also, more scheduled annealing of the π values is another option to investigate.

3.5 Use-cases

Although BNNs are capable of achieving faster inference through bitwise operations, they require specialized hardware to reap these benefits. Researchers have implemented BNNs on supported hardware such as FPGAs and ASICs that allow customized designs adjusted to the network structure [83, 84]. Provided dedicated hardware that can support bitwise model architectures and operations, the proposed BGRU can maximize model reduction benefits with respect to spatial and computational complexity. Furthermore, BNNs have also been implemented on open-source libraries that use C/C++ backend capable of utilizing bit-type data operations [85]. Experiments have shown significant speedups for inference on the CPU device, which indicates another potential use-case for scenarios without GPU resources available.

3.6 Conclusion

This chapter introduced an efficient method to reduce the computational and spatial complexity of the GRU network for the source separation problem while maintaining high performance results. We proposed an incremental binarization procedure to binarize a RNN with GRU cells. The training is done in two rounds, first in a weight compressed network and then in an incrementally bitwise version with the same topology. The pretrained weights of the first round are used to initialize the weights of the bitwise network. For the BGRU cells, we redefined the feedforward procedure with bitwise values and operations. Due to the sensitivity in training the BGRU network, the bitwise feedforward pass is performed gently using two types of masks that determine the level of sparsity and rate of binarization.

With 4-bit QaD quantized input magnitude spectra and IBM targets, the BGRU at full bi-

narization performs well for the speech denoising job with a minimal computational cost. Our experiments show that the proposed BGRU method produces source separation results greater than that of a real-valued fully connected network, with 11-12 dB mean SDR. A fully binarized BGRU still outperforms a Bitwise Neural Network by 1-2 dB even with less number of layers.

Chapter 4

Boosted Locality Sensitive Hashing

4.1 Introduction

In this chapter, we take another route to lightweight source separation by redefining the problem as a $\boldsymbol{K}$-nearest neighborhood ($\boldsymbol{K}$NN) search task: for a given test mixture, the separation is done by finding the nearest mixture spectra in the training set, and consequently their corresponding ideal binary mask (IBM) vectors.

The complexity of the search process poses as a major obstacle as it linearly increases with the size of the training data and the frequency dimension of the spectrum. We expedite this tedious process by converting the query and database spectra into a hash code to exploit bitwise matching operations during the search process. To this end, we start from locality sensitive hashing (LSH), which is to construct hash functions such that similar data points are more probable to collide in the hashed space, or, in other words, more similar in terms of Hamming distance [30, 32]. With increasing number of bits, the Hamming distance between binary hash codes will asymptotically approach the Euclidean distance between pairs of data.

While simple and effective, the random projection-based nature of the LSH process is not trainable, thus limiting its performance when one uses it for a specific problem. We propose a learnable, but still projection-based hash function, Boosted LSH (BLSH), so that the separation is done in the binary space learned in a data-driven way [86]. BLSH reduce the redundancy in the randomly generated LSH codes by relaxing the independence assumption among the projection vectors and learn them sequentially in a *boosting* manner such that they complement one another, an idea shown in search applications [87, 88].

4.1.1 Baseline 1: Direct Spectral Matching

We assume that if two mixture spectra are similar, they consist of similar sources. Hence, their IBMs must be similar, too. The $\boldsymbol{K}$NN search-based source separation requires a large reference dictionary of training examples, whose IBMs are known ahead of time. For a given test example, the separation algorithm searches for only $\boldsymbol{K}$NN to the query spectra to infer its mask.

Let $\boldsymbol{H} \in \mathbb{R}^{D \times T}$ be the normalized feature vectors from T frames of training mixture examples, e.g., noisy speech spectra. T can potentially be a very large number as it exponentially grows with the number of sources. Out of many potential choices, we are interested in STFT and mel-spectra as the feature vectors. For example, if $\boldsymbol{H}$ is from magnitudes of STFT on the training mixture signals, D corresponds to the number of Fourier transform's subbands F, i.e., $D = F$, while for mel-spectra $D < F$.

Columns of $\boldsymbol{H}$ are normalized with their L_2 norm. We also prepare their corresponding IBM vectors, $\boldsymbol{Y} \in \{0, 1\}^{F \times T}$, whose dimension F matches that of STFT, regardless whether we use STFT or mel spectra as input. For a test mixture spectrum, $\bar{\boldsymbol{x}} \in \mathbb{C}^F$, our goal is to estimate a denoising mask, $\hat{\boldsymbol{y}} \in \mathbb{R}^F$, to recover the source by masking, $\hat{\boldsymbol{y}} \odot \bar{\boldsymbol{x}}$. While masking is applied to the complex STFT spectrum $\bar{\boldsymbol{x}}$, the $\boldsymbol{K}$NN search can be done in the D-dimensional feature space $\boldsymbol{x} \in \mathbb{R}^D$, e.g., mel-spectra or full Fourier coefficients.

Algorithm 1 describes the $\boldsymbol{K}$NN source separation procedure. We use notation $\mathcal{S}$ as the affinity function, which denotes the cosine similarity function with a subscript: $\mathcal{S}_{\cos}$. For a given pair of L_2-normalized D-dimensional feature vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, it is defined as an inner product:

$$\mathcal{S}_{\cos}(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b} \tag{4.1}$$

For each frame $\boldsymbol{x}$ in the test-time mixture signal, we find the $\boldsymbol{K}$ closest frames in the reference database (line 4 to 9), which form the set of indices of $\boldsymbol{K}$NN, $\mathcal{N} = \{\tau_1, \tau_2, \cdots, \tau_{\boldsymbol{K}}\}$. Using them, we find the corresponding IBM vectors from $\boldsymbol{Y}$ and take their average (line 13).

In this proposed separation framework, finding $\boldsymbol{K}$ frames that *best* approximate the query

Algorithm 1 $\boldsymbol{K}$NN source separation

Input: $\boldsymbol{x}$, $\boldsymbol{H}$, $\boldsymbol{Y}$

▷ A test mixture vector, the dictionary, and IBMs

Output: $\hat{\boldsymbol{y}}$ ▷ A denoising mask vector

Initialize an empty set $\mathcal{N} = \varnothing$ and $\mathcal{A}_{\min} = 0$

for $t \leftarrow 1$ to T **do**

  if $|\mathcal{N}| < \boldsymbol{K}$ **then**

    Add t to $\mathcal{N}$

    Update $\mathcal{A}_{\min} \leftarrow \min_{k \in \mathcal{N}} \mathcal{S}_{\cos}(\boldsymbol{x}, \boldsymbol{H}_{:,k})$

  else if $\mathcal{S}_{\cos}(\boldsymbol{x}, \boldsymbol{H}_{:,t}) > \mathcal{A}_{\min}$ **then**

    Replace $\arg\min_{k \in \mathcal{N}} \mathcal{S}_{\cos}(\boldsymbol{x}, \boldsymbol{H}_{:,k})$ in $\mathcal{N}$ with t

    Update $\mathcal{A}_{\min} \leftarrow \min_{k \in \mathcal{N}} \mathcal{S}_{\cos}(\boldsymbol{x}, \boldsymbol{H}_{:,k})$

  end if

end for

return $\hat{\boldsymbol{y}} \leftarrow \frac{1}{\boldsymbol{K}} \sum_{k \in \mathcal{N}} \boldsymbol{Y}_{:,k}$

ensures quality source separation. Although it is well known that simple Euclidean or cosine distance cannot represent the semantic similarity between high-dimensional data points, preserving the local similarity in the feature space and the SSM can lead to successful manifold learning [89, 90, 91]. In this regard, our separation method also requires the reference database to be sufficiently large to represent the manifold of the data distribution. However, a large T is burdensome not only for storage but also during the test time since the distance computation between $\boldsymbol{x}$ and $\boldsymbol{H}$ (floating point inner product) is costly.

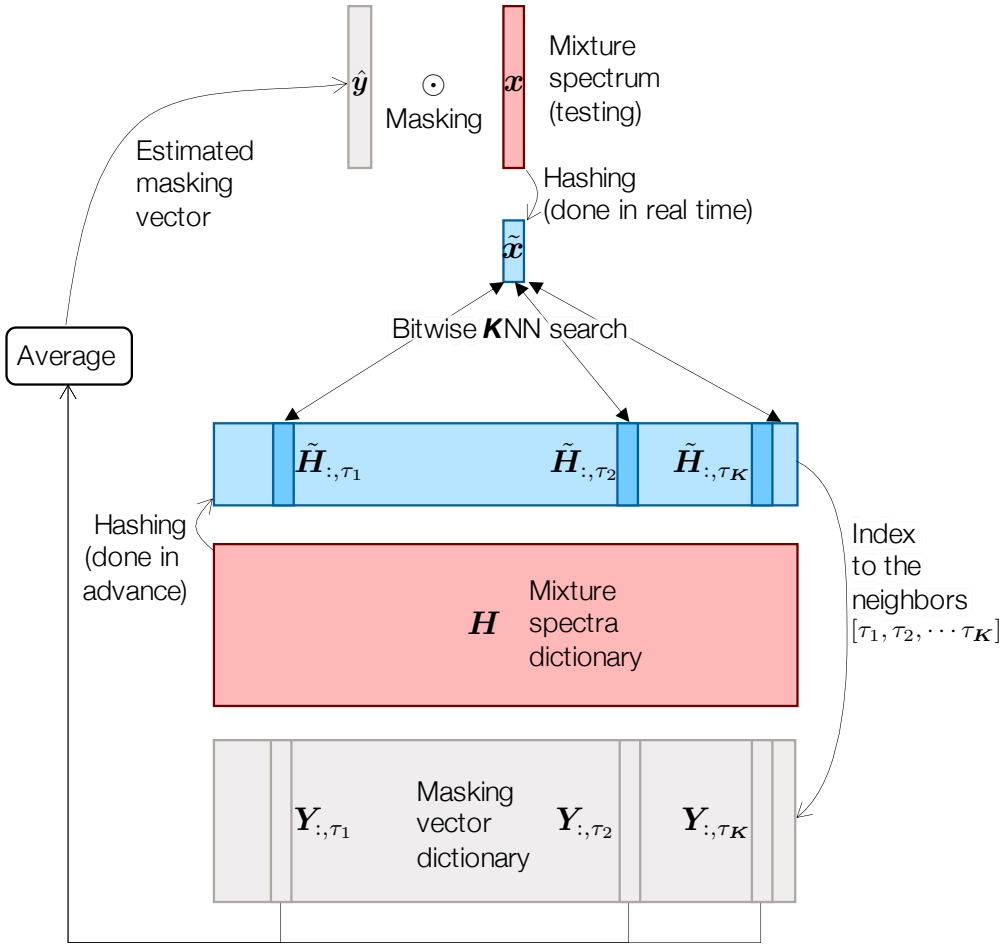


Figure 4.1: The $\boldsymbol{K}$NN-based source separation process using hash codes.

4.1.2 Baseline 2: LSH with Random Projections

We can reduce the storage overhead and expedite Algorithm 1 using hashed spectra and the Hamming similarity between them. For this second baseline, we follow the basic LSH setup that applies random projections to the data points followed by a step function (i.e., the sign function) as the hash function.

We define L random projection vectors as $\boldsymbol{P} \in \mathbb{R}^{L \times D}$ that are randomly initialized and fixed throughout the experiment. The l-th projection using $\boldsymbol{P}_{l,:}$ defines the l-th bit in the codes:

$$\tilde{\boldsymbol{H}}_{l,:} = \text{sign}(\boldsymbol{P}_{l,:}\boldsymbol{H} + c_l) \tag{4.2}$$

where the inner product between $\boldsymbol{P}_{l,:}$ and $\boldsymbol{H}$ is summed with a bias term c_l. Applying the same $\boldsymbol{P}$

onto $\boldsymbol{H}$ and $\boldsymbol{x}$, we obtain T training spectra's hash codes each of which is a bipolar binary bitstring, i.e, $\tilde{\boldsymbol{H}} \in \{-1, +1\}^{L \times T}$, and one for the test spectrum, $\tilde{\boldsymbol{x}} \in \{-1, +1\}^{L}$, respectively. Accordingly, Hamming similarity also replaces the similarity function by counting the number of matching bits in the pair of binary feature vectors. For a given pair of binary feature vectors $\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}} \in \{0, 1\}^{L}$, the Hamming similarity is defined as:

$$\mathcal{S}_{\text{Ham}}(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}}) = \sum_{l} \mathcal{I}(\tilde{a}_l, \tilde{b}_l)/L, \tag{4.3}$$

where $\mathcal{I}(\tilde{a}_l, \tilde{b}_l) = 1$ iff $\tilde{a}_l = \tilde{b}_l$, otherwise $\mathcal{I}(\tilde{a}_l, \tilde{b}_l) = 0$. In our application, we assume bipolar binaries and simplify the expression above as follows:

$$\mathcal{S}_{\text{Ham}}(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}}) = \tilde{\boldsymbol{a}}^{\top} \tilde{\boldsymbol{b}}/L, \quad \text{if } \tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}} \in \{-1, +1\}^{L} \tag{4.4}$$

Given large enough L, we can expect from the hash codes is that the Hamming similarity of an originally similar pair should be more likely to be high than that of a dissimilar pair [32]. Suppose there is a similar pair in the original feature space, e.g., $\mathcal{S}_{\cos}(\boldsymbol{x}, \boldsymbol{H}_{:,i}) > \rho$, where ρ stands for a threshold. Then, a dissimilar pair can be also defined similarly: $\mathcal{S}_{\cos}(\boldsymbol{x}, \boldsymbol{H}_{:,j}) \leq \rho$. Their corresponding LSH codes are discriminative, if they fulfill the following inequality:

$$p\big(\mathcal{S}_{\text{Ham}}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{H}}_{:,i}) = 1\big) > p\big(\mathcal{S}_{\text{Ham}}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{H}}_{:,j}) = 1\big). \tag{4.5}$$

In other words, a successful LSH hashing step guarantees a higher chance of the originally similar pair colliding each other in the same bucket than a dissimilar pair.

Fig. 4.1 overviews the separation process of *Baseline 2*. First, the system prepares the hash codes of all mixture spectra dictionary $\tilde{\boldsymbol{H}}$. Then, it applies the same hashing process (i.e., the same projection matrix $\boldsymbol{P}$) to the test-time mixture spectrum to acquire its hash code $\tilde{\boldsymbol{x}}$. The $\boldsymbol{K}$NN search follows to find the similar training spectra, but the search is performed in the hash code space $\tilde{\boldsymbol{H}}$ instead of the original feature space $\boldsymbol{H}$. Using the found $\boldsymbol{K}$NN entries $\mathcal{N} = \{\tau_1, \tau_2, \cdots, \tau_{\boldsymbol{K}}\}$, the system infers the masking vector. Note that it is similar to *Baseline 1* described in 1, except that the search is done based on the simpler-to-compute Hamming similarity.

The downside of the $\boldsymbol{K}$NN search task performed in a binary feature space $\tilde{\boldsymbol{H}}$ is that it may result in a suboptimal search compared to $\boldsymbol{H}$. However, the upside is that the binary comparison can be fast and efficient since it can be performed using efficient bitwise operations (e.g., bitwise AND and pop counting) with supporting hardware. Furthermore, if the size of the hash table L is much smaller than the original input features, more items can be loaded into memory, resulting in disk I/O cost reduction.

Since the $\boldsymbol{K}$NN search will be on the hash codes, the search performance depends on the quality of the hash function as to how well it preserves the original similarity after hashing. However, when it comes to the LSH method, the lackluster quality of the codes originates from the data-blind nature of random projections [92, 93]. Instead of attempting to learn the best projection vectors from data, the representativeness of the hash codes relies greatly on the randomness in the projection vectors: during the initial construction, each hash function, i.e., the projection, is chosen independently and uniformly at random. Hence, a large L is required to form discriminative hash codes that can guarantee high precision. This is detrimental in terms of query time, computational cost of projecting the query to hash codes, and storage overhead from the large number of projections, even though they result in binary codes.

4.2 Boosted LSH training algorithm

The proposed BLSH algorithm addresses the inefficiency of the LSH algorithm by *learning* each projection vector. With the boosting concept, we learn the projections in an incremental fashion, i.e., one by one in the order of their significance. In this way, each projection improves the total representativeness of the code by trying to fix the preceding projections' mistakes in approximating the original similarity function. As a result, we can expect a hash code whose bits are ordered based on their contribution to the search performance, adding scalability to the entire system. For example, if a too large L is not affordable, instead of re-initializing the entire projection table $\boldsymbol{P}$,

one can trim the end of the existing table. To summarize, the goals of the BLSH training algorithm are as follows:

- Compactness: By using the boosting concept, we aim at achieving higher representation power using shorter bitstrings in comparison with ordinary LSH codes.
- Scalability: Out of the total L bits, the system can choose to utilize only the first few bits because the bits are ordered based on their significance. Hence, the system can adapt to different computational environments with greater scalability.
- Efficiency: The inference on the proposed hashing process should still be affordable (i.e., a single-layer neural network) for efficient test-time hashing.

In this section, we present our learnable LSH algorithm for source separation. We extend LSH-based $\boldsymbol{K}$NN search with approximating SSMs, boosting, and kernel functions.

4.2.1 Binary Approximations of Similarity Matrices

Rather than employing a large number of independent random projections, we aim to *learn* similarity-preserving hash functions that better fit the data distribution. Our goal of learning to hash is to preserve the features' discriminative properties as shown in the literature [94, 93, 95, 96]. Among them, deep learning models [94, 96] have been proposed to learn data-dependent hash codes as well; nonetheless, the computational complexity of multilayered neural networks is restrictive in hashing applications that operate under a tight resource budget.

The proposed BLSH algorithm is still based on the projection and sign function as in the LSH algorithm. However, instead of relying on random projections, we see each projection as a weak classifier and learn its model parameters, i.e., the projection vector, during training. Here, we will use the same formulation defined in eq. (4.2), while the projection vector $\boldsymbol{P}_{l,:}$ and bias b_l are trainable parameters. Note that the non-differentiable sign function results in an intractable Dirac delta function during the gradient descent computation. We circumvent this issue by employing

the derivative of the hyperbolic tangent function as a surrogate of the delta function:

$$\text{sgn}'(x) \approx \tanh'(x) = 1 - \tanh^2(x). \tag{4.6}$$

During training the projection vectors are directed to minimize the discrepancies between the pairwise affinity relationships among the original features in $\boldsymbol{H}$ and those we construct from their corresponding hash codes $\tilde{\boldsymbol{H}}$. We express the loss function in terms of the dissimilarity between the target self-similarity matrices (SSM) and its binary estimation. We denote $\boldsymbol{S} \in \mathbb{R}^{T \times T}$ for the target SSM,

$$\boldsymbol{S}_{i,j} = \mathcal{S}_{\cos}(\boldsymbol{H}_{:,i}, \boldsymbol{H}_{:,j}) \tag{4.7}$$

where $i, j \leq T$. Suppose that the optimization process is learning the l-th projection $\boldsymbol{P}_{l,:}$, which result in $\tilde{\boldsymbol{H}}_{l,:}$ bipolar binary hash codes. We construct the binary SSM $\tilde{\boldsymbol{S}}$ as

$$\tilde{\boldsymbol{S}}_{i,j} = \frac{\tilde{\boldsymbol{H}}_{l,i} \cdot \tilde{\boldsymbol{H}}_{l,j} + 1}{2} \tag{4.8}$$

where we shift and scale such that $\tilde{\boldsymbol{S}} \in \{0, 1\}^{T \times T}$. Then, our objective is to minimize the dissimilarities:

$$\sum_{i,j} \mathcal{D}\Big(\tilde{\boldsymbol{S}}_{i,j} \| \boldsymbol{S}_{i,j}\Big) \tag{4.9}$$

The objective also depends on the choice of the distance metric $\mathcal{D}$, which is cross-entropy in our case. With this objective the learned binary hash codes can be more compact and representative than the ones obtained from random projections as in eq. (4.2). There can be potentially many different solutions to this optimization problem, such as solving this optimization directly for the set of projection vectors $\boldsymbol{P}$ or spectral hashing that learns the hash codes directly with no assumed projection process [93]. We present our proposed method in the following subsection.

4.2.2 Boosted LSH

A major drawback of LSH is the independence between the random hyperplanes. Hence, LSH tends to result in redundancy in number of projections, i.e., a large L value, to build discriminative

Algorithm 2 BLSH training

Input: $\boldsymbol{S} \in \mathbb{R}^{T \times T}$ ▷ Target SSM

Output: $\boldsymbol{P} \in \mathbb{R}^{L \times D}$ ▷ Set of projections

$\boldsymbol{W} \in \mathbb{R}^{T \times T} \leftarrow$ uniform vector of $\frac{1}{T \times T}$

$\boldsymbol{P} \in \mathbb{R}^{L \times D} \leftarrow$ random numbers

$\boldsymbol{\beta} \in \mathbb{R}^{L} \leftarrow 0$

for $l \leftarrow 1$ to L **do**

  $\boldsymbol{P}_{l,:} \leftarrow \arg\min_{\boldsymbol{P}_{l,:}} \sum_{i,j} \mathcal{D}\big(\tilde{\boldsymbol{S}}_{i,j} \| \boldsymbol{S}_{i,j}\big) \odot \boldsymbol{W}_{i,j}^{(l)}$

  $\varepsilon_l = \sum_{i,j} \mathcal{D}\big(\tilde{\boldsymbol{S}}_{i,j} \| \boldsymbol{S}_{i,j}\big) \odot \boldsymbol{W}_{i,j}^{(l)}$

  $\beta_l \leftarrow \ln((1-\varepsilon_l)/\varepsilon_l)$

  $\boldsymbol{W}_{i,j}^{(l+1)} = \boldsymbol{W}_{i,j}^{(l)} \exp\Big(\beta_l \sum_{i,j} \mathcal{D}\big(\tilde{\boldsymbol{S}}_{i,j} \| \boldsymbol{S}_{i,j}\big)\Big)$

end for

return $\boldsymbol{P}$

hash codes. As we saw in Sec. 4.2.1, we resolve this issue by turning LSH into a learnable hashing process, which minimizes the gap between SSM and BSSM (eq. (4.9)). In this way, the hash codes behave similarly to the already-known discriminative features $\boldsymbol{H}$. A remaining issue though is that the BSSM approximation has to pre-define the number of projections L, within which there is no straightforward order of significance. Moreover, if L is set to be too large, some projections do not contribute to the discrimination, leading to a cost in test-time inference, while they increase the bit depth of the code.

We address this issue by proposing a *boosted* LSH algorithm, where each projection is learned one by one, in the order of their significance. Hence, one can rely more on the initially learned projections than the later added ones. This property is designed to handle the test-time use case, where the application chooses to use only a small number of most significant projections, i.e., the *first few* bits from the code to save the processing time and resources.

To this end, we reformulate AdaBoost, one of the most widely studied adaptive boosting strategy which was originally proposed for classification [97]. It learns efficient weak learners, e.g., a linear classifier, in an additive manner: each new weak learner complements those learned in previous rounds and infers something new about the data. The set of weak learners constructs an adaptive basis function model, whose weighted sum makes the strong final prediction. In such a weighted sum, the complementary nature tends to give larger weights to the earlier learners than the later added ones.

The AdaBoost framework aligns with our applicational goal, because we also need such an order of importance between the projections. Moreover, since each of the hash code bits is a result of a weak classifier, i.e., linear combination followed by a sign function, the hashing process is similar to the weighted sum of weak learners in AdaBoost. The biggest difference between AdaBoost and BLSH is that BLSH does not use misclassification as the loss. Instead, it aims to learn weak learners whose binary decision results can improve the speech enhancement performance.

Consequently, we propose to adjust the loss function and the boosting algorithm accordingly. Here, since our speech enhancement is based on the matching process between test and training examples, we once again seek the binary feature space where matching results are similar to the ones in the original feature space. Hence, our training objective is to approximate the pairwise relationship between the known discriminative features $\boldsymbol{H}$ using the binary features $\tilde{\boldsymbol{H}}$. It leads to the total error function defined as follows:

$$\sum_{i,j} \mathcal{D}\Big(\sum_{l=1}^{L} \beta_l \tilde{\boldsymbol{S}}_{i,j} \| \ \boldsymbol{S}_{i,j} \Big), \tag{4.10}$$

where β_l denotes the weight of the l-th weak learner to the total estimation and the l-th weak learner is defined by the l-th projection $\tilde{\boldsymbol{H}}_{:,i} = \boldsymbol{P}_{l,:}\boldsymbol{H}_{:,i} + b_l$ that constructs the shifted and scaled BSSM $\tilde{\boldsymbol{S}}_{i,j}$ (eq. 4.8).

In AdaBoost, when each weak learner is trained, it focuses more on the previously misclassified examples. This mechanism is implemented by a weighting scheme on the samples: those misclas-

sified training samples receive an exponentially large weight, so that the next weak learner pays more attention to them. Likewise, the per-sample weights in AdaBoost are defined over all training samples.

On the other hand, in BLSH for speech denoising, the optimization process involves T^2 pairwise relationships even though it begins with T training examples. It is because our goal is to preserve the cross-sample correlation rather than a per-sample property. Hence, we redesign AdaBoost's sample weighting scheme by defining a *per-pair* symmetric weight matrix $\boldsymbol{W} \in \mathbb{R}^{T \times T}$. It is initialized uniformly, but then updated after adding every projection. The weight matrix is designed to exponentially increase its values when a pair (i, j) is incorrectly represented by BSSM compared to the target SSM as follows:

$$\boldsymbol{W}_{i,j}^{(l+1)} = \boldsymbol{W}_{i,j}^{(l)} \exp\left(\beta_l \sum_{i,j} \mathcal{D}\Big(\tilde{\boldsymbol{S}}_{i,j} \| \boldsymbol{S}_{i,j}\Big)\right), \tag{4.11}$$

where the superscript (l) denotes the per-pair weight matrix associated with the l-th weak learner.

As in AdaBoost, the effect of this update rule is to exaggerate the importance of BSSM cells that failed to reconstruct the target SSM. The $(l+1)$-th weak learner can then use these weights to concentrate on *hard-to-approximate* pairs of examples. This approach pursues a complementary $(l+1)$-th projection, thus overall rapidly increasing the approximation quality in (4.10) with relatively smaller L projections. The final boosted objective for the l-th projection is formulated as

$$\varepsilon_l = \sum_{i,j} \mathcal{D}\Big(\tilde{\boldsymbol{S}}_{i,j} \| \boldsymbol{S}_{i,j}\Big) \odot \boldsymbol{W}_{i,j}^{(l)}. \tag{4.12}$$

Under this boosted objective, the first *few* weak learners result in binary features that approximate the majority of the original feature similarity, thereby dramatically reducing the overall storage of projections, computation from projecting elements, and the length of hashed bit strings. Given the learned l-th projection and per-pair weights, we obtain the projection weights as

$$\beta_l = \ln \frac{1 - \varepsilon_l}{\varepsilon_l}. \tag{4.13}$$

Although the cross-entropy loss $\mathcal{D}$ is relatively small, we clip $\varepsilon_l = \min(\varepsilon_l, 1)$ prior to computing eq. 4.13.

Algorithm 2 summarizes the BLSH procedure introduced in this section. Fig. 4.2 shows the complementary nature of the projections and their convergence behavior. After learning the projection matrix $\boldsymbol{P}$, the rest of the test-time source separation process is the same as described in Sec. 4.1.2.

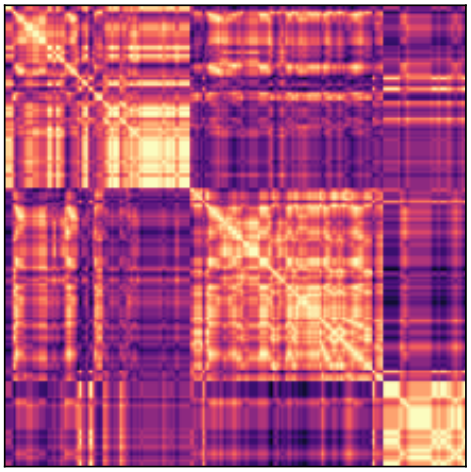
4.2.3 Kernel Functions as Similarity Measure

Deep neural networks can undoubtedly produce highly discriminative data-dependent hash codes via a series of nonlinear transformations. For example, in semantic hashing, autoencoders are trained to learn codes in its bottleneck layer [94]. However, the deep architecture tends to require heavy inference computation as opposed to BLSH. Meanwhile, it is also true that the projection-based LSH process corresponds only to a shallow neural network limiting its performance.

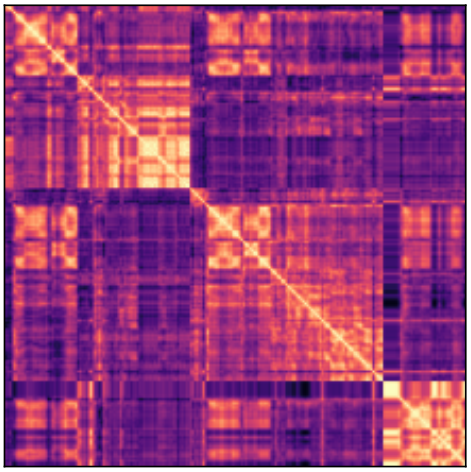
In this section we employ a nonlinear kernel to produce the target SSM so that the proposed BLSH algorithm learns from a potentially very complex pair-wise relationship. Based on the learning objective defined in eq. (4.9), the relative performance of the learned hash functions significantly depends on the definition of ground-truth targets (e.g., distance metric) [93]. Hence, the quality of the feature vectors used to construct the target SSM is critical. For example, if $\boldsymbol{H}$ stands for the raw spectrogram of STFT magnitudes, we could derive a more abstract feature via a nonlinear feature transform function $\phi(\cdot)$. Although the transform function can be a tractable one, such as a neural network encoder, in this section we assume that it can be intractable. By using the kernel trick, we implement our new objective by computing the pairwise similarity through the kernel function:

$$\mathcal{G}\big(\boldsymbol{H}_{:,i}, \boldsymbol{H}_{:,j}\big) = \boldsymbol{\phi}(\boldsymbol{H}_{:,i})^\top \boldsymbol{\phi}(\boldsymbol{H}_{:,j}). \tag{4.14}$$

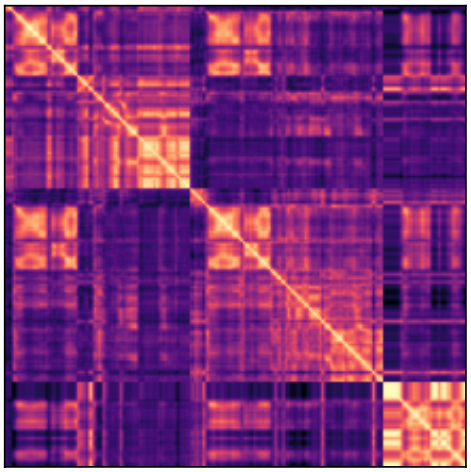
For example, the previous formulation eq. (4.9) can be considered as the case with an identity



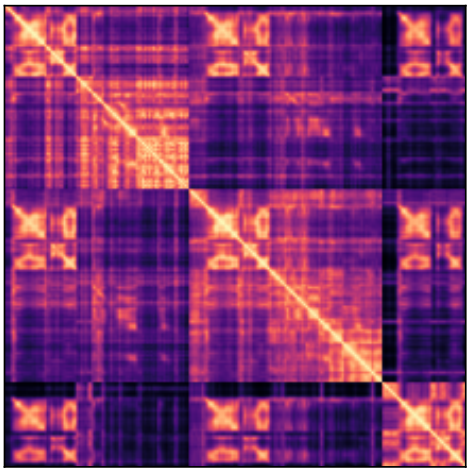
(a) $L = 10$



(b) $L = 50$



(c) $L = 300$



(d) Ground Truth SSM

Figure 4.2: Self-affinity matrices of varying L hash codes and original time-frequency bins. (a) $L = 10$ (b) $L = 50$ (c) $L = 300$ (d) Ground-truth SSM.

mapping function $\boldsymbol{\phi} = \boldsymbol{I}$ that forms a linear kernel. Among various kernels, we adopt the radial basis function (RBF) to create pairwise similarity embedded in high dimensional spaces [98]. An RBF kernel is defined on our normalized input features as follows:

$$\mathcal{G}\big(\boldsymbol{H}_{:,i}, \boldsymbol{H}_{:,j}\big) = C \exp\left(\frac{\boldsymbol{H}_{:,i}^{\top}\boldsymbol{H}_{:,j}}{\sigma^2}\right) \tag{4.15}$$

where $C = \exp(-1/\sigma^2)$ and $\mathcal{G}\big(\boldsymbol{H}_{:,i}, \boldsymbol{H}_{:,j}\big) \in [0, 1]$ for normalized features. The hyperparameter σ^2 controls the width of the Gaussian-shaped decision boundary, which determines the locality of the similarity measure defined in the feature space [99]. Since RBF kernels can be represented as an implicit sum over an infinite sequence of polynomials, it nonlinearly defines an infinitely high-dimensional feature space ($\mathbb{R}^n \rightarrow \mathbb{R}^{\infty}$). Note that with the tractable RBF kernel function in eq. (4.15) one can avoid the intractability involved in the infinitely high-dimensional feature transformation.

Using the kernel trick, we train another version of our hash functions. This time, the objective is to minimize the dissimilarity between the predicted BSSM and the RBF kernel defined between pairwise training feature vectors $\boldsymbol{H}$. With a shortened notation for the kernel matrix $\boldsymbol{G}_{i,j} = \mathcal{G}\big(\boldsymbol{H}_{:,i}, \boldsymbol{H}_{:,j}\big)$ the loss function simply replaces the target SSM $\boldsymbol{S}$ of eq. (4.10)-(4.12) with $\boldsymbol{G}$.

Note that the nonlinearity of the target SSM $\boldsymbol{G}$ can be adjusted using the hyperparameter σ^2: the smaller the value, the more local the distance in the feature space. Eventually, if $\boldsymbol{G}$ is strictly defined with local pairs, $\boldsymbol{G}$ introduces more nonlinearity. Likewise, σ^2 allows for flexibility in determining the target complexity of the system. We will investigate the impact of different choices of σ^2 in the experiment section.

4.3 Experiments

4.3.1 Datasets

To investigate the effectiveness of the proposed framework for speech enhancement, we evaluate our model on the single-channel speech denoising setup. A training set consisting of 10 hours of noisy utterances and their corresponding clean speech signals. The clean utterances are from the randomly selected but gender-balanced 160 speakers from the train fold of the TIMIT corpus. They are mixed with different non-stationary noise signals with 0 dB signal-to-noise ratio (SNR), namely {birds, casino, cicadas, computer keyboard, eating chips, frogs, jungle, machine guns, motorcycles, ocean} [100]. Since there are 10 short utterances per speaker, it amounts to 1,600 utterances recorded with a 16kHz sampling rate. The projections are learned on the training set and the output hash codes are saved as the dictionary.

One hour of evaluation data set was mixed from 110 unseen speakers in the TIMIT test set and unseen noise sources from the DEMAND database, which consists of domestic, office, indoor public places, transportation, nature, and street (**open mixture set**) [101]. The test set mixtures are with a 0 dB SNR and gender-balanced as well.

We apply a short-time Fourier transform (STFT) with a Hann window of 1024 samples and a hop size of 256. Optionally, mel spectrograms are created using 128 mel-scale filters. During training, the whole mixture speech was segmented into a minibatch with a length of 1000 frames, such that the self-similarity matrices and the kernel matrices are of a reasonable size.

For evaluation, we calculate the signal-to-distortion ratio improvement (ΔSDR) [81]. We also report results in terms of model size, scale-invariant SNR ratio improvement (ΔSI-SNR) [102], perceptual evaluation of speech quality (PESQ) with values ranging from -0.5 to 4.5 [103], and extended short-time intelligibility (ESTOI) [104].

4.3.2 Models

In our experiments we use a few different setups to provide a comprehensive understanding of the proposed model. First, we differentiate the models based on the type of the input features $\boldsymbol{x}$ as follows:

- $\texttt{IN}_{\texttt{STFT}}$: The 513-dimensional magnitude coefficients of STFT.
- $\texttt{IN}_{\texttt{mel}}$: The 128-dimensional mel spectra.

In addition, we also denote the models based on what they are targeting:

- `SSM`: This is the case of using the regular SSM but its behavior varies depending on the input representation, i.e., `STFT` or `mel`.
- $\texttt{RBF}_{\sigma^2}$: The case when RBF kernels are used as target. To compute RBF kernels, we only use STFT as input to discern the effects of nonlinearity introduced by the RBF kernel. The RBF kernels vary depending on the choice of σ^2.

We differentiate our hashing-based models based on two important hyperparameters. First, the number of projections L defines the bitdepth of the binary hash codes. Our goal is to achieve a better performance using the proposed BLSH algorithm than the random projection-based LSH method if L is the same. Second, the size of the training dictionary matters. Although in theory we can use the entire training examples and convert them into the hash codes, using these T examples can increase the complexity of the $\boldsymbol{K}$NN search. Instead, we investigate a subsampling option, where we use only a proportion of T training examples, i.e., $\rho = 0.01$ or 0.1. Consequently, we also denote the two hashing mechanisms distinctively:

- $\texttt{LSH}_{L,\rho}$: The ordinary LSH-based method defined with L projections and ρT training examples.
- $\texttt{BLSH}_{L,\rho}$: Ditto, except that the hash codes are learned via the proposed BLSH algorithm.
- $\texttt{KNN}_{\rho}$: This one is a $\boldsymbol{K}$NN-based system, but using raw input features, `STFT` or `mel`, instead of the hash codes. Hence, it is not elaborated with the choice of L.

In all three $\boldsymbol{K}$NN methods, we fixed the neighborhood size to $\boldsymbol{K} = 10$.

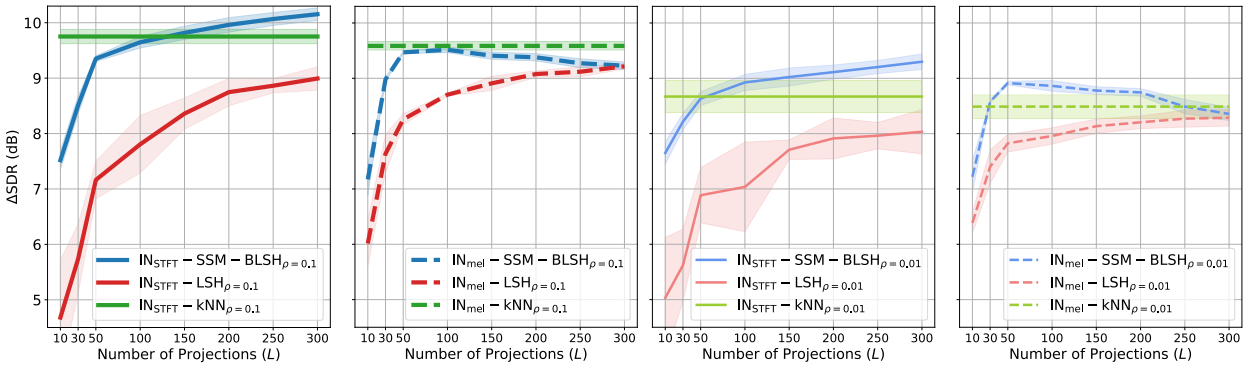
For example, $\texttt{IN}_{\texttt{STFT}}\texttt{-RBF}_{\sigma^2=0.1}\texttt{-BLSH}_{L=300,\rho=0.1}$ denotes a $\boldsymbol{K}$NN-based model that uses hash codes learned from the proposed BLSH method. It uses only 10% of the training data for the $\boldsymbol{K}$NN search and the bitdepth is 300. When BLSH algorithm learns the projection it targets a RBF kernel with $\sigma^2 = 0.1$. The algorithm operates on the magnitudes of STFT spectra. Or, $\texttt{IN}_{\texttt{mel}}\texttt{-KNN}_{\rho=0.01}$ is a model that performs $\boldsymbol{K}$NN search directly on the mel spectra. For cases applying to all L or both ρ values, we remove the subscripts from `BLSH` for brevity.

Finally, three neural network models are trained for comparison.

- `FC`: We train fully-connected (FC) network with two or three hidden layers. The number of hidden units ranges from 32 to $1,024$.
- `BiLSTM`: A recurrent neural network using bidirectional long short-term memory (BiLSTM) [105] cells are employed. We use two BiLSTM layers by varying the number of memory cells and hidden units from 32 to $1,024$.

Both models are trained with STFT magnitude spectrogram inputs and IBM targets. The logistic activation function was applied on the outputs of both models. For the `FC` network, batch normalization [106] and ReLU activation functions [107] were applied to the intermediate outputs. The learning rate was set as 1×10^{-4} for both `FC` and `BiLSTM` models. Adam optimizer was used for both models [108].

In addition, we also present another end-to-end model architecture that performs speech enhancement in the time domain as a reference. Conv-TasNet is a fully convolutional model that performs the masking operation in the latent space, achieving state-of-the-art performance in speech separation benchmarks [62]. While there have been a few other models that followed up showing meaningful improvement on speech separation [109, 2, 110, 111], we adopted the Conv-TasNet model as a stable reference. To train Conv-TasNet, we employed a much larger dataset consisting of Librispeech's training fold [112] and the MUSAN dataset's Free Sound corpus [113] by mixing them with a range of mixing ratio from -5 to 10 dB. Note that this training set is deliberately chosen to



(a) $\texttt{IN}_{\texttt{STFT}}$ at $\rho = 0.1$ (b) $\texttt{IN}_{\texttt{mel}}$ at $\rho = 0.1$ (c) $\texttt{IN}_{\texttt{STFT}}$ at $\rho = 0.01$ (d) $\texttt{IN}_{\texttt{mel}}$ at $\rho = 0.01$

Figure 4.3: Average and standard deviations (shades) of ΔSDR for various ***K***NN systems with respect to the number of projections L under different input features and ρ values. The KNN baseline is expressed as a solid horizontal line since it is not dependent on L. BLSH systems were trained on SSM targets.

maximize the performance of the Conv-TasNet architecture, while the other models mentioned so far have been trained on a much smaller dataset.

4.4 Evaluation Results and Discussion

In this section, we compare the effects of different ***K***NN systems, relative subset size ρ, bitdepth L, and the type of input features and training targets. Experimental results are reported using the evaluation data set (open mixture set).

4.4.1 Analysis of the denoising systems on STFT features

From Fig. 4.3a, we can observe that the proposed BLSH system clearly outperforms the LSH version in terms of average SDR improvements for both mel and STFT input features and different choices of ρ and L. With a sufficiently large L, $\texttt{IN}_{\texttt{STFT}}\texttt{-SSM-BLSH}_{L\approx 100,\rho=0.1}$ can reach the $\texttt{IN}_{\texttt{STFT}}\texttt{-KNN}_{\rho=0.1}$'s performance. For all L values, the performance of $\texttt{IN}_{\texttt{STFT}}\texttt{-SSM-BLSH}_{\rho=0.1}$ is consistently higher than the random LSH version, $\texttt{IN}_{\texttt{STFT}}\texttt{-LSH}_{\rho=0.1}$. This demonstrates that the boosting mechanism provide

more discriminative features compared to its counterpart using random hash codes.

It is also promising that the BLSH hash codes surpass the performance of the raw Fourier coefficients, while LSH hash codes fall short. It should be noted that the $\texttt{IN}_{\texttt{STFT}}\texttt{-KNN}_{\rho=0.1}$ baseline conducts the $\boldsymbol{K}$NN search on high-dimensional floating-point spectrum coefficients (i.e., $D = F = 513$), whereas the BLSH systems are based on the bitwise Hamming distance between L-bit hash codes.

4.4.2 Analysis of the denoising systems on mel features

We can observe from Fig. 4.3b that $\texttt{IN}_{\texttt{mel}}\texttt{-LSH}$ achieves better performance compared to $\texttt{IN}_{\texttt{STFT}}\texttt{-LSH}$, demonstrating the effectiveness of mel features to construct discriminative hash codes even when a small bitdepth L is allowed (i.e., less than $L = 150$). The mel features' improved performance is expected because mel scaling performs a nonlinear feature transform, a logarithmic scaling process that aligns better to human auditory perception. This nonlinear feature transform adds additional representativeness to the LSH methods that rely solely on randomly initialized projections.

However, the mel transformation is a lossy dimension reduction method that condenses multiple subbands into one. Although perceptually motivated, mel scaling eliminates vital information. This fact is reflected in Fig. 4.3b. Our proposed boosting-based framework, $\texttt{IN}_{\texttt{mel}}\texttt{-SSM-BLSH}_{\rho=0.1}$, performs best when using only up to a handful of hash codes (e.g., $L = 50$). The scores saturate after $L = 50$ with no further improvements and rather exhibit overfitting behavior. Hence, we believe that the proposed BLSH algorithm learns the majority of features using approximately the first $L = 50$ weak learners. This result contrasts with the BLSH results on STFT features in Fig. 4.3a, where more hyperplanes learned via the BLSH algorithm keep improving the denoising performance.

Hence, the experimental results suggest that the BLSH method on the raw STFT features is more useful than the LSH counterparts: given the same bitdepth, BLSH hash codes provide better

discrimination, and consequently, better separation. Mel features give boosts to the LSH models, but BLSH on STFT still outperforms it significantly.

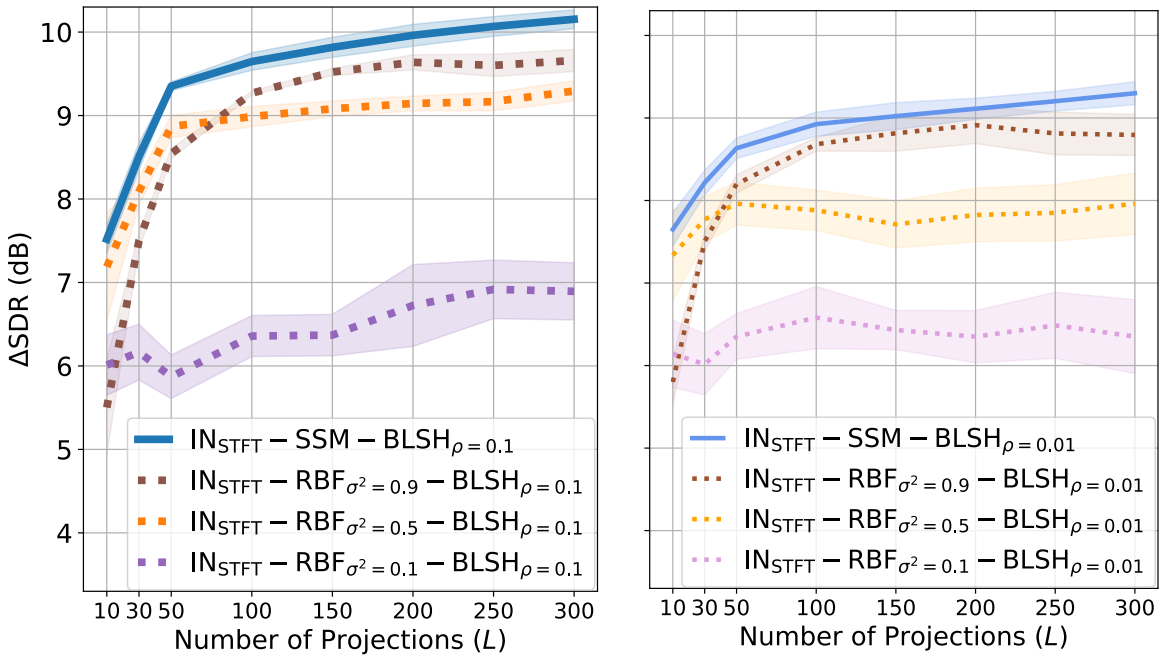
4.4.3 The impact of subsampling the training set

In Fig. 4.3c and 4.3d, we observe a global performance drop of all three $\boldsymbol{K}$NN systems. First, the baseline $\texttt{IN}_{\texttt{STFT}}\texttt{-kNN}_{\rho=0.01}$ exhibits more than 1dB loss by reducing the training set from 10% to 1% of the original size. While the BLSH systems also show decreased performance, they are more robust to the subsampling process. Random projections of LSH show a noticeable sensitivity to the randomness in the subsampling procedure: the standard deviation is significantly larger especially in the $\rho = 0.01$ case, whereas that of BLSH remains tighter. Also, now the BLSH systems start to exceed the baseline when $L = 50$. Note that when $L = 50$, $\texttt{IN}_{\texttt{mel}}\texttt{-SSM-BLSH}_{L=50,\rho=0.01}$ shows slightly better performance than $\texttt{IN}_{\texttt{STFT}}\texttt{-SSM-BLSH}_{L=50,\rho=0.01}$, showcasing mel spectrum's usefulness in this extreme condition. Overall, a similar trend to the $\rho = 0.1$ cases is retained: we still prefer the BLSH model trained from STFT features the best.

4.4.4 The impact of using RBF kernels as the training targets

In addition to the SSMs computed by the mel or STFT features, we also evaluate the validity of nonlinear kernels as a learning target of our BLSH algorithm. Considering the nonlinearity introduced during mel scaling, we construct RBF kernels from the raw STFT features to adequately measure the impact of the level of nonlinearity controlled by the kernel width parameter, σ^2.

Fig. 4.4a compares the SSM-based BLSH results $\texttt{IN}_{\texttt{STFT}}\texttt{-SSM-BLSH}_{\rho=0.1}$ with three BLSH systems targeting RBF kernels with different kernel widths, $\sigma^2 = 0.1$, 0.5, and 0.9. We observe that RBF-based systems exhibit saturation in performance and are unable to catch up to $\texttt{IN}_{\texttt{STFT}}\texttt{-SSM-BLSH}_{\rho=0.1}$. Specifically, wider kernel widths, $\sigma^2 = 0.9$ and 0.5, provide reasonable performance. However, they still fall short compared to $\texttt{IN}_{\texttt{STFT}}\texttt{-SSM-BLSH}_{\rho=0.1}$, indicating that the RBF kernels used here are either too linear to introduce additional distinction between codes or the

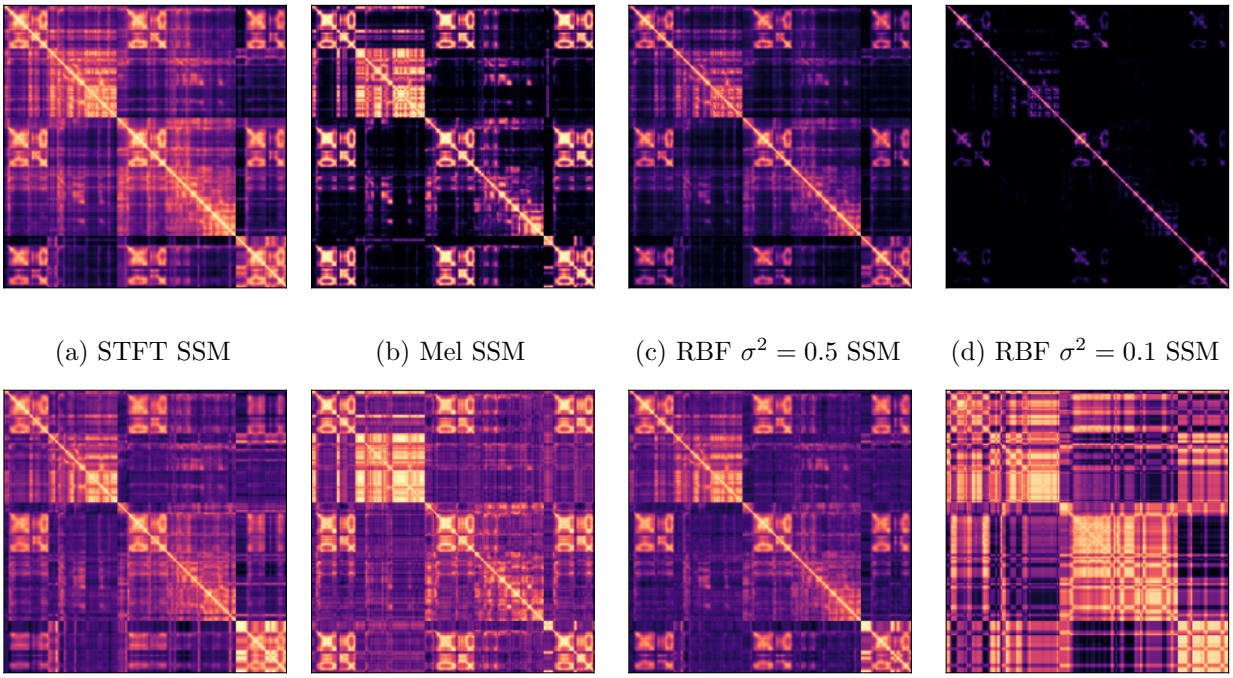


(a) $\rho = 0.1$ (b) $\rho = 0.01$

Figure 4.4: Average and standard deviations of ΔSDR for BLSH systems trained on `SSM` and `RBF` targets under varying σ^2 and ρ values. Only the STFT feature was used for each of the systems.

nonlinear relation is far from perceptual similarity between audio spectra. Furthermore, the most extreme case, $\texttt{IN}_{\texttt{STFT}}\texttt{-RBF}_{\sigma^2=0.1}\texttt{-BLSH}$, rarely shows improvement even during the earlier phases of training. This is due to the narrow width of the kernel which only emphasizes elements that are relatively close and harshly devalues the scores of distant frames to near zero values. Smaller σ^2 allows us to focus on local features, thus being more nonlinear. However, it removes too much information and restricts the perceptrons from learning any more features. Similar trends are observed when the subsampling rate is reduced to $\rho = 0.01$ in Fig. 4.4b, with even less relevant results from the mid-sized kernel ($\sigma^2 = 0.5$).

Fig. 4.5 gives a more detailed view to the varying relationship between BSSM and SSM depending on how the target SSM is constructed. From BSSM reconstructions, first we can see that the linear kernel computed from STFT coefficients is easy for the BLSH algorithm to reconstruct



(a) STFT SSM (b) Mel SSM (c) RBF $\sigma^2 = 0.5$ SSM (d) RBF $\sigma^2 = 0.1$ SSM

(e) STFT BSSM (f) Mel BSSM (g) RBF $\sigma^2 = 0.5$ BSSM (h) RBF $\sigma^2 = 0.1$ BSSM

Figure 4.5: Self-affinity matrices and their estimations from binary codes for varying transformations. The binary outputs shown are constructed as the weighted sum of BSSMs created from all $L = 300$ projections. (a) STFT SSM (b) Mel SSM (c) RBF $\sigma^2 = 0.5$ SSM (d) RBF $\sigma^2 = 0.1$ SSM (e) STFT BSSM (f) Mel BSSM (g) RBF $\sigma^2 = 0.5$ BSSM (h) RBF $\sigma^2 = 0.1$ BSSM.

(Fig. 4.5a vs. 4.5e), which is also the best denoising solution we have achieved so far. Meanwhile, the mel SSM target shown in Fig. 4.5b exhibits more contrast than Fig. 4.5a, indicating more locality introduced in the similarity between mel spectra. BSSM mimics this relationship in Fig. 4.5f, while it fails to catch up with the drastically dissimilar pairs: dark pixels in Fig. 4.5b are not well represented in the BSSM reconstruction.

The RBF kernels also introduce similar locality to the target SSMs. While with $\sigma^2 = 0.5$ the RBF kernel behaves as an SSM with a certain level of nonlinearity (Fig. 4.5c), the most narrow kernel width $\sigma^2 = 0.1$ suppresses most of the non-local pairwise similarities. As a result, $\texttt{IN}_{\texttt{STFT}}\texttt{-RBF}_{\sigma^2=0.1}\texttt{-BLSH}$ (Fig. 4.5h) is distinctly different from its target (Fig. 4.5d), demonstrating

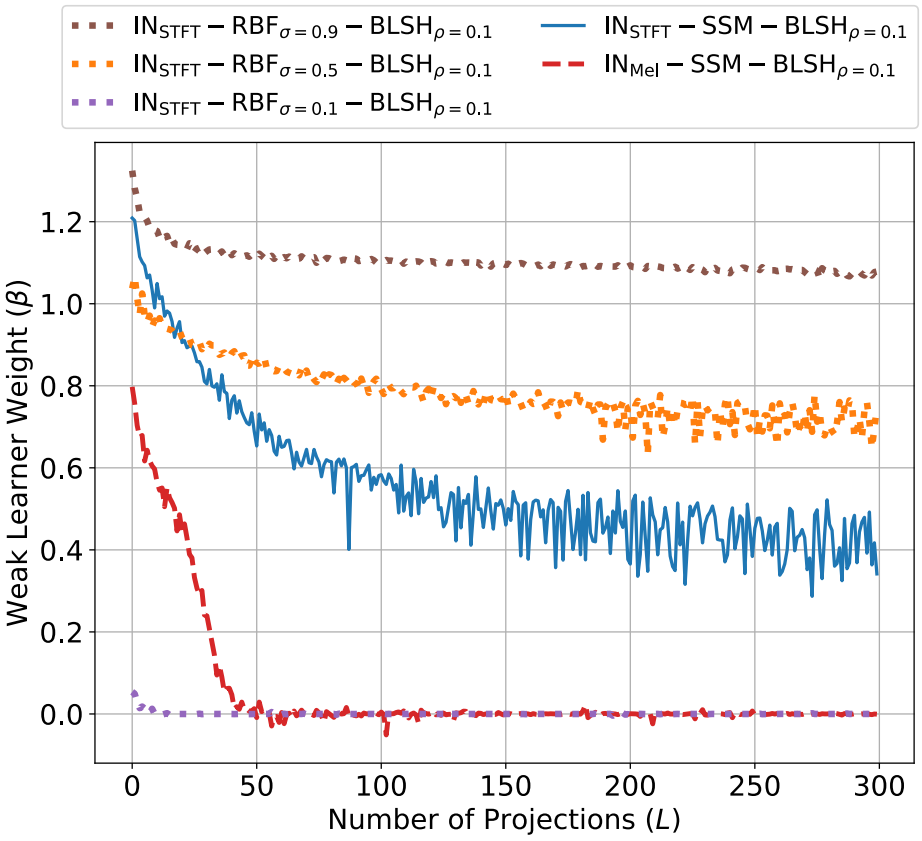


Figure 4.6: Weak learner weights (β) for BLSH systems trained on varying features and kernels

the difficulty for binary codes to approximate the sparsity in the target SSM. This is reflected by the respective denoising performances (e.g., $\texttt{IN}_{\texttt{STFT}}\texttt{-RBF}_{\sigma^2=0.1}\texttt{-BLSH}$ shows relatively low improvements for increasing L). We believe that it is because the RBF kernels do not properly represent the perceptual relationship between audio spectra.

4.4.5 The learning behavior of the BLSH algorithm

In our BLSH algorithm, similarly to the ordinary boosting method, the β values represent relative importance of each weak learner (i.e., the projection) in the final ensemble. It is common to see them diminish as more weak learners are added because the initial weak learners are more important than the later learned ones.

Fig. 4.6 indeed shows the decreasing β values in various BLSH learning processes. First, we

note here that the β values learned from the mel spectra $\texttt{IN}_{\texttt{mel}}\texttt{-SSM-BLSH}_{\rho=0.1}$ decays very fast in the initial few rounds and then stagnates, while the STFT graph from $\texttt{IN}_{\texttt{STFT}}\texttt{-SSM-BLSH}_{\rho=0.1}$ shows more steady decrease. Hence, this comparison explains the saturating or overfitting behavior of the BLSH algorithm on the mel spectrum input shown in Fig. 4.3b and 4.3d.

From Fig. 4.6, it is also noticeable that targeting too localized RBF kernels is detrimental. When $\sigma^2 = 0.1$, for example, the β values quickly drop and become insignificant. SSMs created using RBF kernels with small kernel width σ^2 are mostly comprised of near-zero values (e.g., Fig. 4.5 (d)), and most of the information can be learned using only a few weak learners. For these SSM targets, adding on more weak learners do not contribute to the final boosted solution, which is depicted by $\beta \to 0$ values for increasing L. The diminishing β values show that the newly added weak learners no longer learn significant information and the hyperplanes are not discriminative.

4.4.6 Comparison among *K*NN systems

Table 4.1 shows the model size and performance of all three ***K***NN systems: KNN, LSH, and BLSH. We narrow our discussion to BLSH systems trained on linear SSM targets. Model sizes are computed as the total memory occupied by the training dictionary and hash functions. The KNN baselines show the enormous memory occupied by the dictionary. For example, at $\rho = 0.1$ subsampling rate, $\texttt{IN}_{\texttt{STFT}}\texttt{-KNN}_{\rho=0.1}$ requires $\sim$0.5GB. $\texttt{IN}_{\texttt{mel}}\texttt{-KNN}_{\rho=0.1}$ requires less since the number of subbands for mel-spectra is lower than STFT's.

LSH methods show significant reduction in dictionary size with minimal storage overhead from the random projections, although at the cost of drop in performance. Across various subsampling rates ρ, LSH methods show a distinct drop in performance, especially in terms of SDR and SI-SNR, compared to its KNN counterparts using the same input features.

Our proposed BLSH method consistently outperforms the LSH approaches and also KNN on the STFT magnitude features. $\texttt{IN}_{\texttt{STFT}}\texttt{-SSM-BLSH}_{L=300,\rho=0.1}$ achieves better scores than $\texttt{IN}_{\texttt{STFT}}\texttt{-KNN}_{\rho=0.1}$

Table 4.1: Model size, average ΔSDR, ΔSI-SNR, PESQ and ESTOI for various systems evaluated on open mixture set. Hashing-based ***K***NN systems are reported using optimal L parameters, and their model size is expressed as a sum of reference database size and model size. `FC` and `BiLSTM` models are reported using relevant parameters N_l and N_h, the number of layers and hidden units respectively, in place of D and L. Similarly Conv-TasNet model is reported using N_r and N_b denoting the number of repeats and blocks respectively. We used the same parameters as reported in the Conv-TasNet (CTN) paper. The model sizes are reported using single precision (i.e. 32 bits per parameter) for both ***K***NN and deep learning systems.

System	Feature	ρ	L	Model Size (MB)	ΔSDR	ΔSI-SNR	PESQ	ESTOI
Mixture	-	-	-	-	0.00	0.00	1.51	81.97
IBM	-	-	-	-	20.38	19.98	2.93	94.58
`KNN`	STFT	0.1	-	503.55	9.77	8.76	1.80	73.74
	Mel	0.1	-	125.64	9.61	8.12	1.73	74.60
	STFT	0.01	-	50.36	8.65	7.58	1.76	71.38
	Mel	0.01	-	16.79	8.36	6.85	1.66	71.20
`LSH`	STFT	0.1	300	9.20 + 0.62	8.97	7.96	1.79	72.50
	Mel	0.1	300	1.53 + 0.15	9.23	7.72	1.73	73.60
	STFT	0.01	300	0.92 + 0.62	7.95	6.90	1.76	71.24
	Mel	0.01	300	0.15 + 0.15	8.25	6.71	1.68	71.20
`BLSH`	STFT	0.1	300	9.20 + 0.62	10.18	9.10	1.81	74.98
	Mel	0.1	50	1.53 + 0.03	9.47	7.84	1.84	76.58
	STFT	0.01	300	0.92 + 0.62	9.31	8.15	1.73	72.56
	Mel	0.01	50	0.15 + 0.03	8.92	7.25	1.80	75.78
		N_l	N_h					
`FC`	STFT	3	1024	12.61	13.10	12.34	1.64	82.32
			256	1.58	12.92	12.08	1.74	82.10
			64	0.30	12.19	11.28	1.55	80.40
			32	0.14	11.74	10.51	1.59	79.64
		2	1024	8.41	10.29	9.85	1.68	78.46
			256	1.32	10.25	9.71	1.59	77.06
			64	0.28	10.01	9.35	1.53	74.53
			32	0.14	9.89	9.07	1.48	74.18
`BiLSTM`		2	1024	155.36	14.74	14.24	1.87	86.74
			256	13.68	14.18	13.62	1.86	85.79
			64	1.85	13.50	12.91	1.83	84.69
			32	0.79	13.06	12.21	1.81	82.89
		N_r	N_b					
CTN	Raw	3	8	19.94	18.48	18.19	2.45	92.47

across all metrics. As discussed previously in Sec. 4.4.1, this demonstrates the better representativeness of the learned hash codes than the raw Fourier coefficients or the basic LSH codes. Furthermore, our BLSH methods using mel-spectra features can save more memory than LSH by requiring smaller L. $\texttt{IN}_{\texttt{mel}}\texttt{-SSM-BLSH}_{L=50}$ achieves higher scores with fewer hash functions than $\texttt{IN}_{\texttt{mel}}\texttt{-LSH}_{L=300}$ for both $\rho = 0.1$ and 0.01 cases.

Next, we evaluate the computational complexity of the aforementioned $\boldsymbol{K}$NN systems and highlight the efficiency of BLSH models. First, the baseline KNN search procedure requires a linear scan of all real-valued feature vectors in $\boldsymbol{H}$, giving $O(QDT)$, where Q stands for the floating-point precision (e.g., $Q = 64$ for double precision). This procedure is restrictive since T needs to be large for quality source separation and the distance computation (i.e. floating-point inner product) is costly, which is the reason for pursuing an expedited approach to using hash codes.

Since the LSH baseline also uses Algorithm 1, the dependency to T remains the same. However, we can reduce the complexity from $O(QDT)$ to $O(LT)$ if $L < QD$. The procedure can be significantly accelerated with supporting hardware as the Hamming similarity calculation is done through bitwise operations. Also, the size of the hash table is designed to be much smaller than the original input features ($L < QD$) and hence can be loaded into memory, resulting the disk I/O cost reduction.

Our proposed BLSH framework does not change the run-time complexity $O(LT)$ of baseline 2. Nevertheless, BLSH can outperform LSH with smaller L thanks to the boosting mechanism (e.g., $\texttt{IN}_{\texttt{mel}}\texttt{-SSM-BLSH}_{L=50,\rho=0.1}$ scores higher than any LSH systems using $L = 300$). In conclusion, BLSH models can achieve higher accuracy and more efficient memory usage than both baseline systems, computationally heavy KNN and data-blind LSH approaches.

4.4.7 Comparison against deep neural networks (DNN)

Although our goal in this work is not to compete with the state-of-the-art DNN's performance, as our solution is more about maximizing the utility of the binary models, here we present some denoising results from various DNN models to provide a reference point.

In Table 4.1, we also compare the BLSH system with deep neural networks, a fully-connected network (FC) and a bi-directional long short-term memory (BiLSTM) network using varying number of hidden units. We denote N_l and N_h as the number of hidden layers and units respectively. We also report the performance of one of the state-of-the-art source separation models, Conv-TasNet. We use N_r and N_b to denote the number of *repeats* and *blocks*, which defines the internal separator module's architecture in Conv-TasNet; we used the default values as reported in the paper.

First, it is obvious that neural networks with complex structure (e.g., BiLSTM and Conv-TasNet) easily outperform BLSH models by all evaluation metrics. We observe that Conv-TasNet can achieve better performance than the largest BiLSTM while using significantly smaller architecture (19.94 vs 155.36 MB). Given that these models are either designed to learn from a long sequence (i.e., BiLSTM) or capable of processing a relatively long segment of input signal (i.e., 1 second input for Conv-TasNet), comparison to BLSH's frame-by-frame processing results is inadequate. Meanwhile, it also signifies that the proposed BLSH methods do not scale up to provide state-of-the-art performance. Our aim in designing the BLSH method was to maximize its performance in the hashing-based denoising paradigm.

Given the drastic difference in model sizes, a more reasonable comparison would be between BLSH and the small neural network models. For example, BLSH outperforms FC models of various architectures and performs similarly to BiLSTM models at least in terms of PESQ. Furthermore, BLSH can achieve comparable ESTOI scores to small FC models. For example, the ESTOI score of $\texttt{IN}_{\texttt{mel}}\texttt{-SSM-BLSH}_{L=50,\rho=0.01}$ with 0.18MB of parameters is better than the 2×64 FC model with 0.28MB (75.78% vs 74.18%).

From the perspective of a neural network, the BLSH or LSH methods can be seen as a shallow neural network with only one hidden layer, whose activations are binarized. Hence, it is expected that BLSH does not compete with the properly trained DNNs, especially in terms of SDR or SI-SDR. Still, we believe that BLSH may work as a reasonable solution. For example, in comparison to the similar-sized FC networks, BLSH loses only about 7.6% ($\texttt{IN}_{\texttt{mel}}\texttt{-SSM-BLSH}_{L=50,\rho=0.1}$'s 9.47dB SDR improvement vs. FC 2×256's 10.25dB) or 9.8% ($\texttt{IN}_{\texttt{mel}}\texttt{-SSM-BLSH}_{L=50,\rho=0.01}$'s 8.92dB SDR improvement vs. FC 2×32's 9.89dB) of the denoising performance. Considering that these DNN models are operating with floating-point values, a hardware support that benefits from bitwise operations can favor BLSH, as it clearly outperforms the other $\boldsymbol{K}$NN-based baselines, `KNN` and `LSH`. In addition, BLSH provides ordered binary representations based on their contribution, which adds scalability to the system as another merit.

4.4.8 Use-cases

The use-cases for the BLSH framework is similar to previous section's BGRU (Sec. 3.5) as both methods utilize bitwise storage and inference. One distinct difference between our proposed BLSH model with deep learning methods and BNNs is in achieving higher PESQ scores. In a scenario where resource is constrained yet perceptual quality is a priority, it would be advantageous to employ a BLSH model. For example, $L = 50$ on a $\rho = 0.01$ dictionary of mel features ($\texttt{IN}_{\texttt{mel}}\texttt{-SSM-BLSH}_{L=50,\rho=0.01}$) totals 0.18MB for both model and dictionary size, and achieves PESQ scores close to those reported for BiLSTM models.

4.5 Conclusion

In this chapter, we proposed a data-driven hashing algorithm for the source separation problem using nearest neighbor search. Under this setup, the $\boldsymbol{K}$-nearest matching frames in the dictionary to the test frame are found, and a denoising mask estimated by the average of associated IBMs.

To implement a lightweight solution, we utilize locality sensitive hash functions to transform the query and database spectra into binary hash codes. In doing so, memory space is reduced and search process expedited using bitwise matching operations. We compress the framework further through a learnable approach to hashing using the boosting theory. The hash functions are learned sequentially as linear classifiers in a boosting manner such that they complement one another. Our learning objective is set to minimize the discrepancy between the self-similarity of real-valued spectra and hash codes. Hence, we preserve the original similarity between the hash codes, so a test query can be quickly matched from the database through the bitwise $\boldsymbol{K}$NN search. Our framework trains binary classifiers sequentially under a boosting paradigm, culminating to a better approximation of the original self-similarity matrix with shorter hash strings. With learned projections and expedited inference using bitwise operations, our method offers a lightweight solution to the speech enhancement problem.

To study the level of nonlinearity the learned hash codes achieves, we employed RBF kernels as the self-similarity targets and taught the weak learners to learn more complex pairwise relationships. Since the features are expected to be more discriminant through the nonlinear transformation implied by the RBF kernel, it is expected that the BLSH code learned from it may encode a certain level of nonlinear similarity. While the BLSH algorithm does achieve reasonable performance by targeting the RBF kernels, we also found that the data-blind nature of RBF kernel itself has only a narrow capacity in capturing perceptual relationship among audio spectra.

Evaluation results demonstrate that the $\boldsymbol{K}$NN performance with the learned hash codes is better than with random codes from locality sensitive hashing and even raw Fourier coefficients or mel-scaled spectra. This shows that the hash codes learned through the proposed BLSH method function as more discriminative features than the real-valued features. Our proposed framework is not comparable to comprehensive deep learning models in terms of denoising quality, but it showed promising results compared against a very small neural network with similar model size to the

BLSH dictionary.

Chapter 5

Personalization: Zero Shot Test-Time Adaptation

5.1 Introduction

In this chapter, we propose a novel personalization framework to adapt compact models to test time environments and improve their speech enhancement performance in noisy and reverberant conditions. We aim at use-cases for end-user devices in realistic speech enhancement (SE) settings where we often encounter only a few speakers and noise types that tend to reoccur in the specific acoustic environment. DNN solutions are typically with a large model capacity and trained from a large training set, so they generalize well to various test-time conditions (e.g. wide ranges of different speakers, noises, and signal-to-noise ratios of the added noise). We postulate a small personalized model suffices to handle this focused subset of the original universal speech enhancement problem. Our study addresses a major data shortage issue for personalization: although the goal is to learn from a specific user's speech signals and the test time environment, the target clean speech data is not available for model training due to privacy-related concerns and technical difficulty of recording clean and dry voice signals.

Our goal in this test-time adaptation is to utilize no clean speech target of the test speaker, thus fulfilling the requirement for zero-shot learning. To complement the lack of clean speech, we employ the knowledge distillation framework: we distill the more advanced denoising results from an overly large teacher model, and use them as the pseudo target to train the small student model. This zero-shot learning procedure circumvents the process of collecting users' clean speech, a process that users are reluctant to comply due to privacy concerns and technical difficulty of recording clean voice. Experiments on various test-time conditions show that the proposed personalization method can significantly improve the compact models' performance during the test time. Furthermore,

since the personalized models outperform larger non-personalized baseline models, we claim that personalization achieves model compression with no loss of denoising performance.

5.1.1 Personalization for Model Compression

Model compression techniques such as quantization require a fine-tuning procedure to compensate for the injected noise (e.g. quantization-aware training). The performance drop from these methods can be attributed to their *context-agnostic* approach as they do not utilize the specificity of the test time context. Instead, they tend to seek a general-purpose compression technique that overall works reasonably well in various real-world test conditions. As a result, it is commonly expected to observe a certain level of performance drop after compression.

We aim at developing a *context-aware* DNN compression method for SE. We envision that a compressed model can reduce its run-time complexity without losing its performance if it focuses on a particular test environment. We contrast the proposed concept and the ordinary DNN-based SE models, which are typically designed as general-purpose frameworks with a large architecture. A DNN's large capacity is fully utilized when it is trained on a large training set, generalizing well to unseen test time conditions, e.g., different speakers, noise sources, signal-to-noise ratios (SNR), and room acoustics. In some practical use cases though, it suffices for the enhancement model to perform well only for the specific test time context. For instance, a family-owned smart assistant device sitting in the living room needs to perform well only for the family members' voices and their home acoustics, but not necessarily for the other situations. Compared to the general-purpose SE model, *the generalist*, our context-aware compression method can allow a model to adapt to the specific speakers and their acoustic context, overcoming the generalization losses. We call this kind of context-aware SE models *personalized* speech enhancement (PSE) systems. Since the test time context is assumed to contain small variability, a small personalized model can even outperform larger and more complex universal generalist models, demonstrating personalization as a form of

model compression.

5.1.2 Zero-Shot Learning for Domain Adaptation

The proposed personalized SE models achieve context-awareness by reducing the domain mismatch between the training and test datasets. The topic of domain adaptation has been an active area of research in machine learning. One common procedure for domain transfer is regularizing the differences between the learned representations of source and target datasets. It has been applied for emotion, speech, and speaker recognition [114, 115]. However, these applications rely on ample target data, which cannot be assumed if the target problem is narrowly defined as in our PSE cases. Few-shot adaptation can be a solution, as it requires only a small amount of ground-truth signal [116]. However, it can be challenging to obtain ground-truth user information due to recent privacy infringement, data leakage issues. For example, the advancement in DeepFake technology increased people's concern towards releasing personal information [14]. Meanwhile, with user compliance, the user enrollment phase can obtain trigger phrases from the users, but there is no guarantee that these recordings are clean enough to be used as the target of PSE.

In contrast to aforementioned approaches, zero-shot learning is a data-free solution suitable for training tasks where no additional labeled data is available [117, 118]. In the context of personalization, a zero-shot approach means that it does not require test users' clean speech data or their home acoustic environment, while its goal is still to adapt to the test time specificity. Zero-shot learning is an active research topic for classification tasks, where test time labels are typically inferred by computing auxiliary information [119, 120]. Similarly, zero-shot learning in the speech and audio classification applications extracts semantic properties or articulatory distribution to obtain labels during test time [121, 122].

5.2 The Proposed KD-Based Zero-Shot Personalization Algorithm

In this section, we present a zero-shot learning approach to personalization for joint dereverberation and denoising based on the knowledge distillation (KD) framework [20]. As a zero-shot learning method, it does not ask for clean ground-truth signals from the user, while it still aims at enhancing noisy reverberant mixtures. Since its goal is to train a small specialist model for a particular user's speech and recording environment, it qualifies as a personalization method. For zero-shot learning approaches implemented via student-teacher strategies, it is common to use data synthesis techniques through generative adversarial frameworks, where the generator generates fake samples [123, 124, 125]. Pseudo samples can be also synthesized using activation or output statistics of trained teacher models [126, 127], among many others [128, 129]. Instead of including an intermediate data synthesis step, our proposed model directly uses the teacher model's outputs as if they were ground-truth targets. Previous works have successfully applied KD to develop compact SE systems [130, 131, 132, 133]. Under the KD framework, the basic assumption is that the teacher model's large computational capacity guarantees the generalization goal. We extend this concept to a novel zero-shot learning approach for *personalized* SE. Since the teacher model works well in most test time environments, we consider its excellent SE results as if they were the target clean speech from the student model's perspective. That way, we can turn any noisy and reverberant test signals into labeled training examples by passing them through the teacher model. In this process, the teacher model remains as a generalist model, while the student model can use the teacher's generalization power to learn from the test time input signals, fulfilling the zero-shot learning condition.

Using a KD learning paradigm enables us to leverage noisy unlabeled data and obtain their corresponding soft targets generated by the teacher model. In this paper, we focus on domain adaptation when we have large unlabeled target-domain dataset and assume noisy speech data of a target test-time user to be more widely available as opposed to their clean labeled data. Under

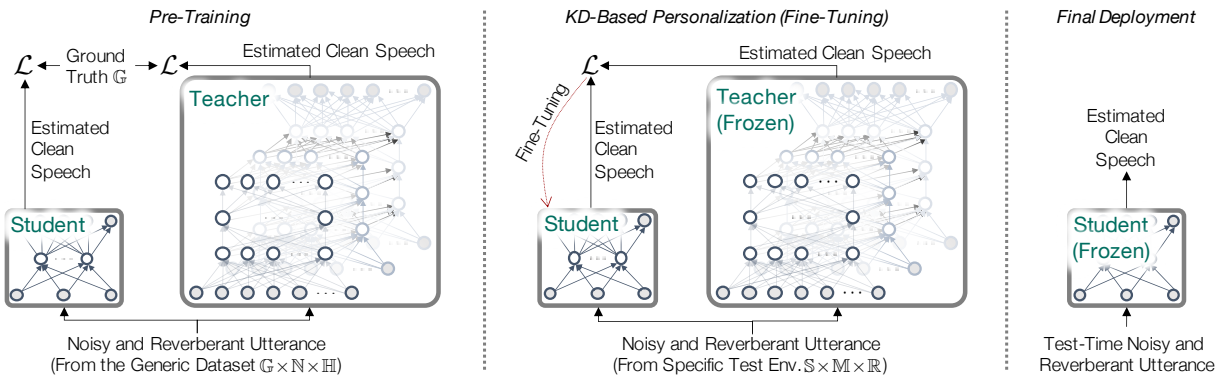


Figure 5.1: An overview of the proposed KD-based PSE process. KD-based fine-tuning learns from the target test environment: the estimated clean speech by the student model is compared against the result from a larger teacher model, whose discrepancy is used to fine-tune the student model. The zero-shot framework enables test time adaptation.

this assumption, we approach the PSE problem with a self-supervised method where corresponding pseudo-targets are generated from large amounts of unpaired noisy speech data [134, 135, 136, 137].

We implement the proposed ZSL-based PSE via KD. Our goal is to fine-tune a compact student model during the test time, so it adapts to the unseen test speaker and environment. KD plays a key role in our ZSL framework, as its teacher model provides a *pseudo* target for the student model to learn from, while the target clean speech of the test-time noisy utterance is absent. We claim that the proposed PSE method will be helpful when the system needs to deal with the peculiarity of the test-time conditions. This kind of flexibility will be also advantageous for SE models if the system has to be frequently relocated to different test environments. Figure 5.1 describes the KD-based PSE process that can fine-tune the student model during the test time.

5.2.1 Training Teacher Models

First, we train the teacher model $\mathcal{T}(\cdot)$ using a large-scale dataset consisting of dry speech sources and various noise signals. Here, the teacher model $\mathcal{T}(\cdot)$ is defined with a large model architecture, so

it can properly approximate the complex general-purpose joint speech denoising and dereveberation function. Once trained, $\mathcal{T}(\cdot)$ is frozen and *not* fine-tuned, assuming that its SE performance as a generalist meets the quality standard in most test cases. Another assumption is that it is too complex for the given test time user device to perform real-time SE inference tasks.

To train the teacher models, we use generic training datasets. The formulation of the training dataset is as follows. The clean speech utterances are taken from a large corpus containing many speakers, $\boldsymbol{s} \in \mathbb{G}$; the noise recordings are also from a large dataset containing various noise types, $\boldsymbol{n} \in \mathbb{N}$; the RIRs are similarly from a large collection recorded in various rooms, $\boldsymbol{h} \in \mathbb{H}$. We use them to synthesize the noisy signals $\boldsymbol{x}$ as input (Eq. (2.13)).

Hence, the goal of the teacher model is to denoise $\boldsymbol{x}$, so the model can estimate the waveforms $\hat{\boldsymbol{s}}$ that closely approximate the target clean speech, i.e., $\boldsymbol{s} \approx \hat{\boldsymbol{s}} \leftarrow \mathcal{T}(\boldsymbol{x})$. The optimization on $\mathcal{T}(\cdot)$ reduces the loss between the target utterance $\boldsymbol{s}$ and reconstruction $\hat{\boldsymbol{s}}$, i.e., $\arg\min_{\boldsymbol{\Theta}_\mathcal{T}} \mathcal{L}(\boldsymbol{s}||\mathcal{T}(\boldsymbol{x};\boldsymbol{\Theta}_\mathcal{T}))$, where $\boldsymbol{\Theta}_\mathcal{T}$ denotes the trainable parameters of the teacher model. Note that the training process for the teacher models correspond to the typical supervised learning method for general-purpose SE. Detailed model and optimization descriptions are provided in Sec. 5.3.2.

5.2.2 Pre-Training Student Models

Our student models $\mathcal{S}(\cdot)$ are pre-trained in a similar way to the teacher models, i.e., by updating its own model parameters $\arg\min_{\boldsymbol{\Theta}_\mathcal{S}} \mathcal{L}(\boldsymbol{s}||\mathcal{S}(\boldsymbol{x};\boldsymbol{\Theta}_\mathcal{S}))$ using the same generic datasets, $\mathbb{G}$, $\mathbb{N}$, and $\mathbb{H}$. However, its small capacity hinders it from generalizing well to the unseen test conditions. Hence, we argue that further improvement is required for these student models to meet the quality requirement. We introduce the KD-based test time personalization algorithm in Sec. 5.2.3 which is designed to reduce the performance gap between $\mathcal{T}(\cdot)$ and $\mathcal{S}(\cdot)$. In this regard, the purpose of pre-training $\mathcal{S}(\cdot)$ is to prepare the student model better than a random initialization, primed for the next fine-tuning step. Further details on model and training are also given in Sec. 5.3.2.

5.2.3 Test time Personalization

During the test time, we assume that the enhancement system is exposed to mixture signals composed of clean speech utterances from the test speaker, $\boldsymbol{s} \in \mathbb{S}$, and background noise sources, $\boldsymbol{n} \in \mathbb{M}$. Note that we differentiate these test sets from the training sets, i.e., $\mathbb{G} \neq \mathbb{S}, \mathbb{N} \neq \mathbb{M}$. Meanwhile, we also assume that the noisy speech signals defined by the combination of the training sets $\mathbb{G} \times \mathbb{N}$ are representative enough to encompass the test time variations, i.e., $\mathbb{S} \times \mathbb{M} \subseteq \mathbb{G} \times \mathbb{N}$. Hence, we postulate that if a teacher model is large enough it can serve as an unbiased solution to the denoising problem. Meanwhile, a small student model is also of our interest if it is small enough for the resource-constrained edge device. However, it may be too biased to generalize well to the test time SE task due to its small model capacity.

Given these assumptions, we propose a personalization framework that can adapt to a new environment without requiring test user's ground-truth clean speech samples or any other auxiliary information of the speakers and acoustic scene. Since we formulate the proposed personalization method as a fine-tuning process, we begin with a compact student model, $\mathcal{S}(\cdot)$, pre-trained in a context-agnostic manner as in Sec. 5.2.2. To fine-tune the student model, its enhancement result from denoising, $\hat{\boldsymbol{s}}_{\mathcal{S}}$, must be compared against the target to compute the loss and perform backpropagation. However, since we assume the target is not available, we use the pseudo target computed from the teacher model.

This process falls in the category of the student-teacher framework in which a student model is optimized using a teacher model's prediction [20]. In the context of personalized speech enhancement, we employ a large pre-trained teacher model $\mathcal{T}(\cdot)$ whose predicted clean utterance serves as the target to compute the student model's loss. Both student and teacher models are initialized with pre-trained generic enhancement models as discussed in Sec. 5.2.1 and 5.2.2, respectively. During the test time, the student model is optimized as: $\arg\min_{\boldsymbol{\Theta}_{\mathcal{S}}} \mathcal{L}(\hat{\boldsymbol{s}}_{\mathcal{T}} || \tilde{\mathcal{S}}(\boldsymbol{y}; \boldsymbol{\Theta}_{\tilde{\mathcal{S}}}))$, where $\hat{\boldsymbol{s}}_{\mathcal{T}}$ is the estimates of clean speech signals obtained from the teacher model and $\boldsymbol{\Theta}_{\tilde{\mathcal{S}}}$ are trainable param-

eters of the student model. We distinguish this fine-tuned student model $\tilde{\mathcal{S}}(\cdot)$ from the pre-trained one $\mathcal{S}(\cdot)$ from now on.

The teacher's estimate $\hat{\boldsymbol{s}}_{\mathcal{T}}$ is only an approximation of the ground-truth target $\boldsymbol{s}$, and can contain artifacts from denoising [138]. However, under a zero-shot PSE setup, we assume having these synthesized pseudo targets is better than nothing. Hence, the performance of the fine-tuning results depends on the quality of $\hat{\boldsymbol{s}}_{\mathcal{T}}$. To this end, we employ relatively large models that surely outperform the student models on the test signals, i.e., $\mathcal{L}(\boldsymbol{s}||\hat{\boldsymbol{s}}_{\mathcal{T}}) < \mathcal{L}(\boldsymbol{s}||\hat{\boldsymbol{s}}_{\mathcal{S}})$. Given its large capacity, the teacher model generalizes better to unseen inputs compared to the student model. Thus, we hypothesize that the student still learns from these imperfect targets and adapts to the test environment.

5.2.4 Interpretation from a Manifold Assumption

By training under the SE criterion, models learn to produce latent representations that are robust to corruptions in input data and are useful for recovering the clean speech. Successfully learned latent representations are discriminative and can capture useful structure and variations in the input distribution as discussed in the context of denoising autoencoders [139]. We interpret the process of SE using the manifold assumption: high dimensional speech data lie on a low dimensional manifold [140]. Under this interpretation, corrupt examples are mapped away from the manifold of clean inputs, and SE models try to project these examples back onto the manifold (Fig. 5.2). During training, SE models focus on learning the underlying manifold and try to minimize the geodesic distance between the points and the learned manifold, thereby mapping corrupt examples $\boldsymbol{x}$ back to (approximately) uncorrupted $\hat{\boldsymbol{s}}$. The farther away $\boldsymbol{x}$ is from the manifold, the more corrupt the example, and the model takes bigger steps to reach the manifold.

We expect large complex models to better approximate the manifold given its larger architecture and thus being able to achieve higher generalization performances. On the contrary, smaller models

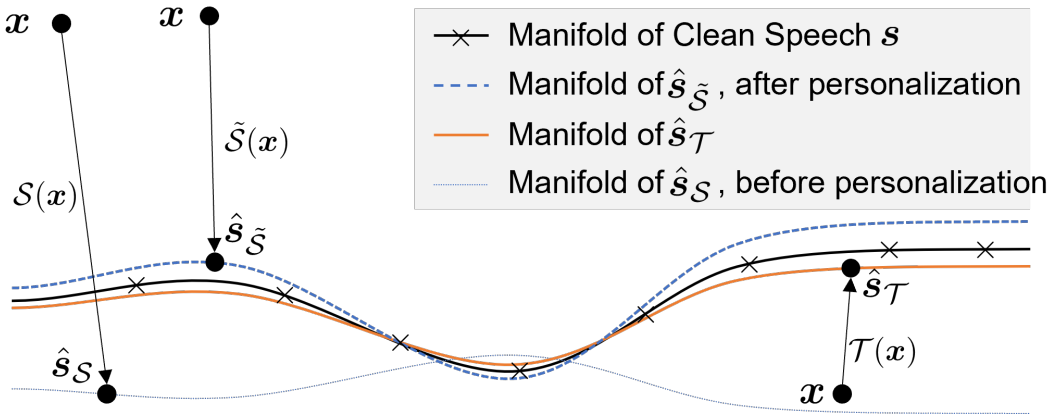


Figure 5.2: Manifold learning perspective of SE. By having the manifold of the ground-truth clean speech samples $\boldsymbol{s}$ as the target, the goal SE is to learn a non-linear feature transform (e.g., a neural network) that maps the corrupt input $\boldsymbol{x}$ back to the manifold. SE models $\mathcal{T}$ and $\mathcal{S}$ do this conversion but only to their own estimates of the estimated manifold, respectively. After personalization, the student model $\mathcal{S}$ improves to a better version $\tilde{\mathcal{S}}$, whose manifold estimate is more closer to the original than the pre-trained student's.

are likely to learn a poorly defined manifold leading to mediocre SE results (the red dotted line in Fig. 5.2). The aim of our proposed personalization framework is to distill the pseudo-manifold determined by $\hat{\boldsymbol{s}}_{\mathcal{T}}$ to the smaller student models to help better approximate the manifold. Also, under our assumption that test time environments will contain smaller subset of sources (e.g. speakers, noises and room variation), the models would only need to learn the manifold of those subsets. By doing so, student models fine-tuned under our framework will be able to better define and map points $\boldsymbol{x}$ closer to the test time manifold approximated by $\hat{\boldsymbol{s}}_{\tilde{\mathcal{S}}}$ (the red dash in Fig. 5.2).

5.3 Experiments

5.3.1 Datasets

For pre-training, we used clean speech recordings from the LibriSpeech corpus [112], and noise recordings from the MUSAN [113] and ESC50 dataset [141]. For RIRs, we used publicly avail-

	Corpus		Notation
Speech	Librispeech [112]	`train-clean-360`	$\mathbb{G}$
		`test-clean`	$\mathbb{S}_{ft/va/te}$
Noise	MUSAN [113]		$\mathbb{N}$
	ESC50 [141]		$\mathbb{M}_{ft/va/te}$
RIR	AIR [142], PORI [143], RWCP [144]		$\mathbb{H}$
	BUT [145], REVERB [146]		$\mathbb{R}_{ft/va/te}$

Table 5.1: Corpus and notation of speech, noise and RIR datasets used during pre-training and personalization.

able recordings downloaded using Kaldi scripts[1]. The RIR data sources consist of the Aachen Impulse Response Database [142], PORI concert hall impulse responses [143], and RWCP Sound Scene Database in Real Acoustical Environments [144]. We used Librispeech's `train-clean-360`, MUSAN's `free-sound` and the collective RIR data for training, which we denote as $\mathbb{G}$, $\mathbb{N}$ and $\mathbb{H}$ respectively. A comprehensive summary of RIR datasets including information on RT60, number of rooms, microphone to loudspeaker distance can be found in [143] and [145]. This exposes the generalist models to up to 251 speakers, 843 noise recordings, and 334 RIRs during training. The noisy mixtures are obtained by adding the noise to speech signals at random input SNR levels uniformly chosen between -5 and 10 dB.

For fine-tuning, or zero-shot PSE, we used 44 speakers from Librispeech's `test-clean` and noise from the ESC-50 dataset for environmental sound classification with 50 different noise types from 5 categories consisting of animals, natural and water soundscape, nonspeech human sounds, interior-domestic sounds, and exterior-urban sounds. For RIRs, we used 5 rooms from BUT Speech@FIT Reverb Database (BUT) [145] and real rooms from the Reverb 2014 Challenge (RVB) dataset [146] for a total of 11 rooms. Each room in the BUT dataset contains 31 microphones and 5 source

[1] `https://github.com/kaldi-asr/kaldi/blob/master/egs/aspire/s5/local/multi_condition/rirs/`

positions in average. RIRs were measured for each speaker position using exponential sine sweep method. For RVB dataset, there are 3 types of rooms (small, medium and large) and 2 types of microphone placement (near and far). RIRs are collected from 2 microphone angles per room. Further information on RIR datasets can be found in Table 1 of [145].

We synthesize $K = 44$ unique test time environments, each of which consists of a test speaker, a noise source, and a RIR configuration defined by the location of the speaker and microphone. In particular, given a test environment index $k \in \{1, \ldots, K\}$, we sample clean utterances from the k-th speaker $\mathbb{S}^{(k)}$, convolve it with k-th room's RIR $\mathbb{R}^{(k)}$ and add noises from k-th noise type $\mathbb{M}^{(k)}$. For each test environment, $\mathbb{S}^{(k)}$ are split into separate sets for fine-tuning and evaluation: the partitions are approximately 5, 1, and 1 minutes of clean speech, which we denote by $\mathbb{S}^{(k)}_{\text{ft}}$, $\mathbb{S}^{(k)}_{\text{va}}$ and $\mathbb{S}^{(k)}_{\text{te}}$. The noise and RIR samples are prepared similarly and partitioned into three separate sets. We synthesize noisy and reverberant input signals by combining $\mathbb{S}^{(k)}_{\text{ft}}$, $\mathbb{M}^{(k)}_{\text{ft}}$, and $\mathbb{R}^{(k)}_{\text{ft}}$. Having them as input, the student model is fine-tuned via the KD process, where the teacher model's denoising results are used as the pseudo target. In other words, the student model for generic SE is first deployed to the device and is personalized to the user's specificity using 5 minutes of noisy and reverberant recordings of the test environment. Then, $\mathbb{S}^{(k)}_{\text{va}}$, $\mathbb{M}^{(k)}_{\text{va}}$, and $\mathbb{R}^{(k)}_{\text{va}}$ are used to validate the student model during fine-tuning, mainly to prevent overfitting. Note that this does not mean that the PSE algorithm needs clean speech for validation: the validation process still relies on the teacher's estimate of clean speech as the target to compute the validation loss. Hence, early stopping is still conducted in a zero-shot manner. We report our PSE models' final performance using the test sets, $\mathbb{S}^{(k)}_{\text{te}}$, $\mathbb{M}^{(k)}_{\text{te}}$, and $\mathbb{R}^{(k)}_{\text{te}}$, for which we do compute the final enhancement performance by comparing to the ground-truth clean speech signals $\mathbb{S}^{(k)}_{\text{te}}$.

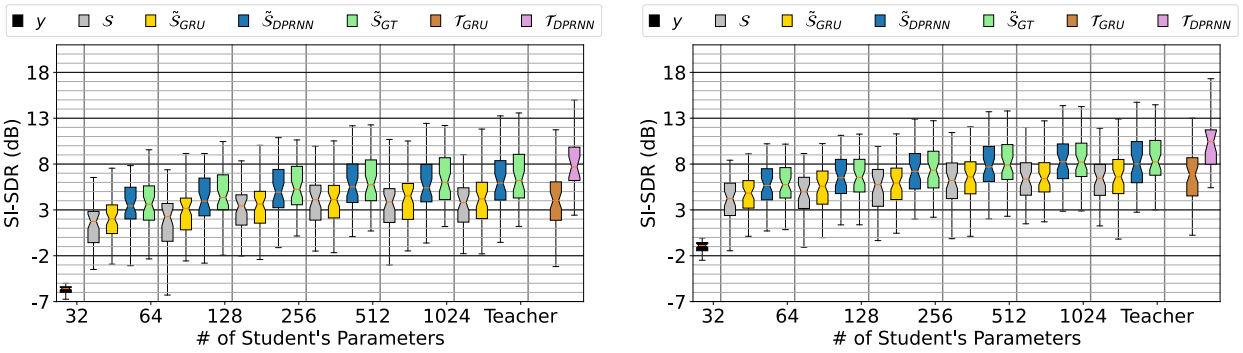
When we simulate various test conditions, the noise and speech sources are mixed under four different input SNR levels (i.e. -5 dB, 0 dB, 5 dB and 10 dB). All speech and noise audio files are loaded at 16 kHz sampling rate and standardized to have unit-variance.

5.3.2 Models

Our students models are based on the uni-directional gated recurrent unit (GRU) architecture [147]. The simple recurrent architecture was deliberately chosen as a simple yet effective baseline due to their success in sequential and temporal processing tasks [148, 149, 150]. Also, the computational complexity of recurrent models are much lower compared to convolutional architectures. We use frequency-domain representations obtained through the short-time Fourier transform (STFT) as inputs to the enhancement models. STFT is with a Hann windowed frame of 1024 samples and a hop size of 256 samples. A dense layer transforms the GRU's output into complex ideal ratio masks [151]. The denoising mask is applied element-wise to the mixture spectrogram, then transformed back to the time-domain signal $\hat{\boldsymbol{s}}$ through inverse STFT. We use negative scale-invariant signal-to-noise ratio (SI-SNR) as the loss function [152]. While the GRU architecture for the student models is fixed with two hidden layers, we vary their hidden units from 32 to 1024 to verify the impact of personalization on the different architectural choices.

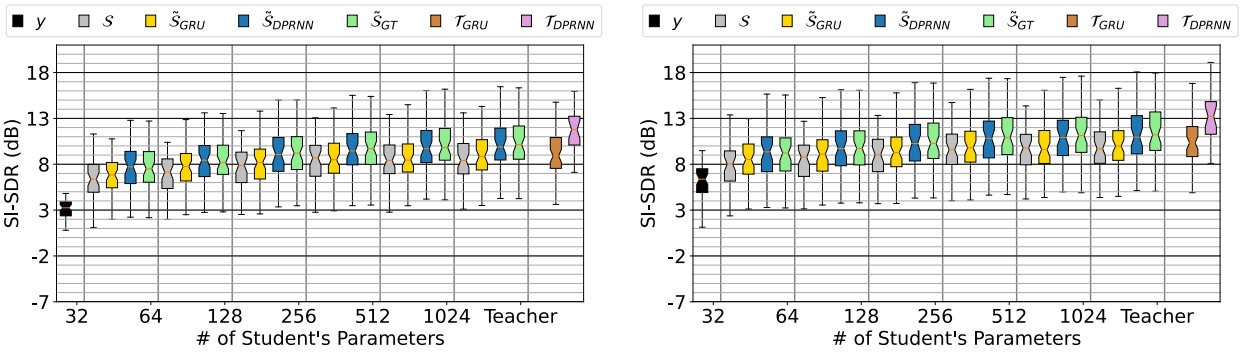
Meanwhile, as for the teacher model, we employ two different network architectures. First, we use a 3×1024 GRU architecture, which is large enough to outperform the students. In addition, we also employ Dual-Path RNN (DPRNN) [153] as an alternative teacher model. DPRNN was chosen because of its higher performance and smaller model size compared to other time-domain models such as Conv-TasNet [62]. More advanced transformer based models such as Dual-Path Transformer (DPTNet) [111] report higher performance with comparative size to DPRNN in speech separation tasks, but we found empirically that the DPRNN performs better for our dereverberation and denoising task.

Indeed, the DPRNN teacher outperforms the GRU teacher due to its structural advantage. Hence, we contrast the impact of the two teacher models on the PSE performance after the KD-based fine-tuning process. The DPRNN model is configured using implementation available in Asteroid's source separation toolkit [154]. Same architecture as reported in [153] is adopted (i.e. 6



(a) -5 dB input SNR

(b) 0 dB input SNR



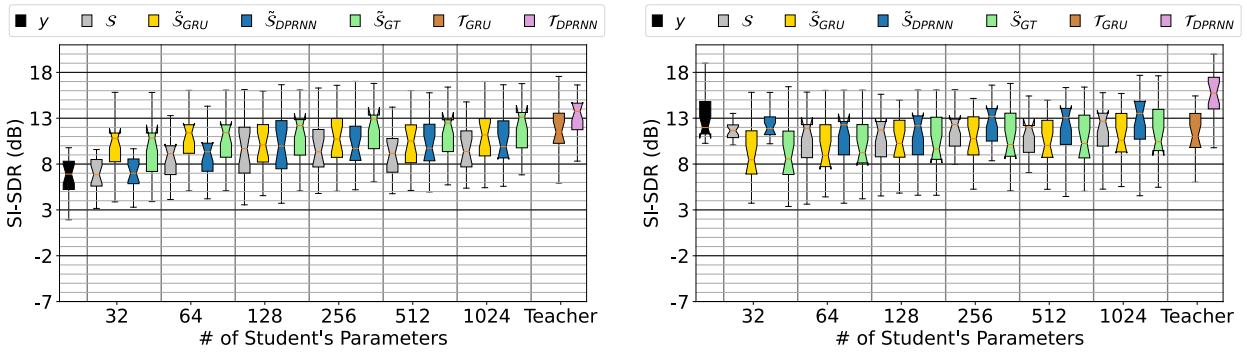
(c) 5 dB input SNR

(d) 10 dB input SNR

Figure 5.3: Comparison of joint dereverberation and denoising performances from pre-trained generalists against personalized specialists under various input SNR levels. Student models are initialized as 2-layered GRU generalists. Teacher models are provided as references.

repeats), while we trained it with our single-speaker SE setup rather than the original speech separation task. The model architectures, their respective number of parameters, and the multiplier-accumulator (MAC) operation counts are shown in Table 5.3. Note that DPRNN is not the largest model but it requires extensive MAC operations.

Here, we introduce new notations to distinguish the two teacher model architectures: $\mathcal{T}_{\text{GRU}}$ and $\mathcal{T}_{\text{DPRNN}}$. In addition, we also denote the fine-tuned students models differently from the pre-trained initial model $\mathcal{S}$ and add the subscript to indicate what it learns from: $\tilde{\mathcal{S}}_{\text{GRU}}$ and $\tilde{\mathcal{S}}_{\text{DPRNN}}$,



(a) Lower SI-SDR($\boldsymbol{y}, \boldsymbol{s}$)

(b) Upper SI-SDR($\boldsymbol{y}, \boldsymbol{s}$)

Figure 5.4: Dereverberation performances of pre-trained generalist and personalized specialists on reverberant inputs without additional background noise (i.e., dereverberation-only experiments). The results are shown as two separate cases, where SI-SDR of input reverberant signals are low for half of the environments ($K = 22$) as shown in Fig. 5.4a and high for other half in Fig. 5.4b.

respectively. We include a student model fine-tuned on the ground-truth targets as a baseline and denote it as $\tilde{\mathcal{S}}_{\mathrm{GT}}$. Summary of notations for pre-trained and fine-tuned models are listed in Table 5.2. Note that the systems denoted with tilde, $\tilde{\mathcal{S}}_{\mathrm{GRU}}$, $\tilde{\mathcal{S}}_{\mathrm{DPRNN}}$, and $\tilde{\mathcal{S}}_{\mathrm{GT}}$, represent personalized student models, while the generalist models, $\mathcal{S}$, $\mathcal{T}_{\mathrm{GRU}}$, and $\mathcal{T}_{\mathrm{DPRNN}}$, are discussed to report the performance of either the teacher models or the baseline.

The Adam optimizer [108] was used with learning rate of 1×10^{-4} for pre-training and 1×10^{-5} for fine-tuning.

5.4 Discussion

5.4.1 Overall performance analysis

The box plots in Fig. 5.3 show the enhancement performances of various models under $K = 44$ environments synthesized with different noise level conditions. The results are shown for pre-trained and fine-tuned student models as well as for teacher models as reference. Figures from 5.3a

Table 5.2: Notations for pre-trained and fine-tuned models.

Notation	Description
$\mathcal{T}_{\text{GRU}}$	GRU teacher model, trained from generic datasets and frozen
$\mathcal{T}_{\text{DPRNN}}$	DPRNN teacher model, trained from generic datasets and frozen
$\mathcal{S}$	Initial student model pre-trained from generic datasets
$\tilde{\mathcal{S}}_{\text{GRU}}$	Student model fine-tuned on $\mathcal{T}_{\text{GRU}}$'s test output
$\tilde{\mathcal{S}}_{\text{DPRNN}}$	Student model fine-tuned on $\mathcal{T}_{\text{DPRNN}}$'s test output
$\tilde{\mathcal{S}}_{\text{GT}}$	Student model fine-tuned on the test time ground-truth targets

to 5.3d show comprehensive denoising and dereverberation results across cases with severe (-5dB) to moderate (+10dB) test time SNR levels. We see that the overall performance decreases due to the additive background noise. While their final input SNR values are controlled by varying the loudness of the test noise sources $\mathbb{M}_{\text{te}}^{(k)}$, the speech sources are also degraded by the reverberation defined by the test time RIR set $\mathbb{R}_{\text{te}}^{(k)}$. In these figures, we observed that our proposed personalization framework improves dereverberation and denoising performances of pre-trained student models under all noise and room conditions, i.e., $\tilde{\mathcal{S}}_{\text{GRU}}$ and $\tilde{\mathcal{S}}_{\text{DPRNN}}$ results are always better than the $\mathcal{S}$ results on average if their model complexity is the same. From these results, we can infer that personalization helps significantly improve the joint dereverberation and denoising performance for all student architectures.

In addition, we also observe that the personalized models learned from the DPRNN teacher, $\tilde{\mathcal{S}}_{\text{DPRNN}}$, always outperform their corresponding ones fine-tuned using the GRU teacher, $\tilde{\mathcal{S}}_{\text{GRU}}$. Since each fine-tuned student models stem from the same pre-trained GRU model, this shows that the fine-tuned performance depends on the quality of the teacher model. It is also noticeable that the structural discrepancy between the student and teacher, i.e., $\tilde{\mathcal{S}}_{\text{GRU}}$ (a GRU) and $\mathcal{T}_{\text{DPRNN}}$ (a DPRNN), is not an issue. It implies that the proposed framework can potentially employ various advanced teacher models as the deep learning research improves the state of the art in the future.

We also notice that $\tilde{\mathcal{S}}_{\text{DPRNN}}$ can catch up to $\tilde{\mathcal{S}}_{\text{GT}}$'s performance, which is only marginally

Table 5.3: Complexity of student and teacher models in MACs and number of parameters. MACs are computed given 1-second inputs.

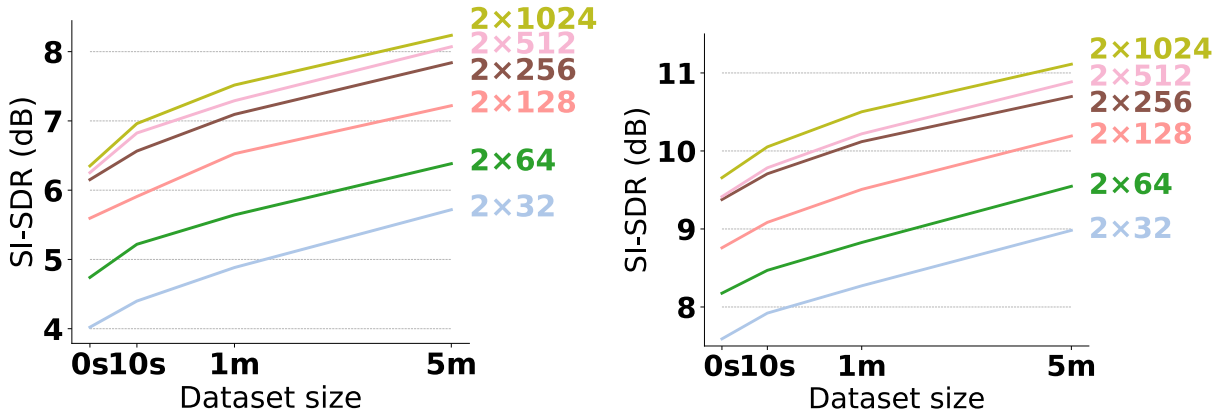
Models		MACs (G)	Param. (M)
Student	GRU (2×32)	0.006	0.09
	GRU (2×64)	0.013	0.20
	GRU (2×128)	0.030	0.48
	GRU (2×256)	0.079	1.25
	GRU (2×512)	0.232	3.68
	GRU (2×1024)	0.762	12.08
Teacher	GRU (3×1024)	1.159	18.37
	DPRNN [153]	15.238	3.63

better. This suggests that fine-tuning on imperfect pseudo-targets generated by $\tilde{\mathcal{T}}_{\text{DPRNN}}$ has almost the same benefits as when ground-truth targets are utilized. Given the student models' small architecture and mediocre generalization capacity, our personalization procedure can fine-tune the student model to its optimal performance, but by relying only on the teacher's SE results.

After personalization, the small student models consistently show significant improvements on their pre-training-based initialization. Hence, it verifies that our personalization framework is a model compression method, if we compare the improved PSE models to those pre-trained generalist models. Indeed, a smaller personalized model can compete with a large generalist, e.g. 2×32 $\tilde{\mathcal{S}}_{\text{DPRNN}}$ vs. 2×1024 $\mathcal{S}$ for -5 dB input SNR as in Fig. 5.3a. According to Table 5.3, a personalized 2×32 specialist saves 11.99M parameters and 756M MACs compared to a 2×1024 generalist (for 1-second inputs), which is more than 99% reduction in terms of spatial and arithmetic complexity. Hence, even if further compression methods, e.g., 8-bit quantization, are always available to the 2×1024 $\mathcal{S}$ model, it will still most likely be more complex than 2×32 $\tilde{\mathcal{S}}_{\text{DPRNN}}$. Furthermore, applying

compression on larger models will subsequently lower their performance depending on the type and amount of compression. On the contrary, the 2×32 $\tilde{\mathcal{S}}_{\text{DPRNN}}$ outperforms the 2×1024 $\mathcal{S}$ even after its 99% reduction of complexity. This demonstrates that our framework works as a mode of *lossless model compression*. Hence, we argue that it is more advantageous to personalize the models instead of increasing generalists' computational capacity for better generalization capabilities. In addition, since PSE shows improved performances in both denoising and dereverberation tasks in various unique test environments, our personalization framework can be seen as a genuine *adaptive system* that specializes not only in each individual user, but in the specific noise source and reverberant condition of the test time environment.

Fig. 5.4 show the SE performance of various models on the reverberation-only input signals, i.e., with no additive background noise. It gives a separate view to the proposed PSE method's dereverberation performance from the joint denoising and dereverberation setup. In addition, the dereverberation results are shown separately as two cases, in which one half of the environments ($K = 22$) are with low input SI-SDR (Fig. 5.4a) and the other with high input SI-SDR (Fig. 5.4b). For the lower SI-SDR cases, the results in Fig. 5.4a show that our framework can successfully personalize to different room acoustics. Contrary to joint dereverberation and denoising results, the improvements for $\tilde{\mathcal{S}}_{\text{DPRNN}}$ are minimal compared to those of $\tilde{\mathcal{S}}_{\text{GRU}}$. This trend is not observed in Fig. 5.4b that reports results from upper SI-SDR cases. First, the generalist models $\mathcal{S}$ worsen the sound quality. Since the SI-SDR of the reverberant inputs were already high, the pre-trained generalists must have injected artifacts that significantly decrease the quality of the signal. This is also evident in the $\tilde{\mathcal{S}}_{\text{GT}}$ baselines where the model is fine-tuned on ground-truth anechoic targets. Hence, the goal of personalization in these cases is to be able to mitigate this performance degradation of the processed signals. Indeed, $\tilde{\mathcal{S}}_{\text{DPRNN}}$ show improved performance over the initial worsened estimates by the pre-trained models, demonstrating the KD framework's consistent capacity to produce pseudo-labels for fine-tuning student models.



(a) 0 dB input SNR

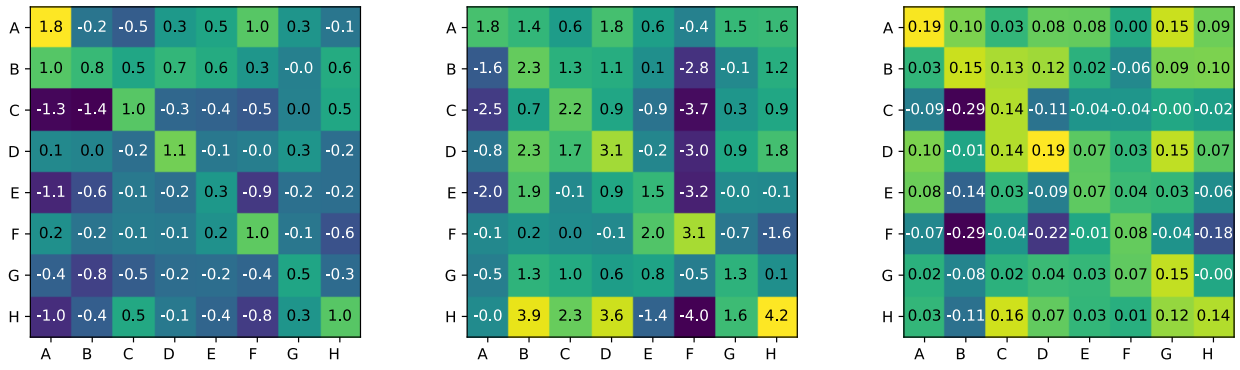
(b) 10 dB input SNR

Figure 5.5: Relative improvements in SI-SDR from different dataset sizes for fine-tuning on various input SNR levels. Student models are 2-layered GRU fine-tuned on DPRNN teacher models ($\tilde{\mathcal{S}}_{\text{DPRNN}}$).

5.4.2 Evaluation of personalization using varying amounts of fine-tuning datasets

Our proposed personalization showed great improvements under 5 minutes of fine-tuning data, which is noisy and reverberant speech recorded from the same acoustic scene during test time. However, we cannot assume this amount of data to be readily available for realistic scenarios. Hence, we test our framework on varying amounts of noisy reverberant mixtures as well, i.e., 10 seconds and 1 minute. Fig. 5.5 shows the average SI-SDR improvements from using varying lengths of noisy input data across all K environments. We only use the DPRNN teacher's estimates for fine-tuning since we have observed its effectiveness from the previous section. For brevity, we test on 0 dB and 10 dB input SNR cases.

As expected, the length of available datasets for fine-tuning is correlated with the test time performance. This experiment illustrates a realistic use-case where initially 5 minutes of noisy data will not be directly available, but rather 10 seconds and later 1 minute of test time signals will be gradually collected over time in a realistic data collection scenario. From both figures, we can observe that smaller personalized models can still outperform a larger generalist even with less



(a) Δ SI-SDR (b) Δ STOI (c) Δ PESQ

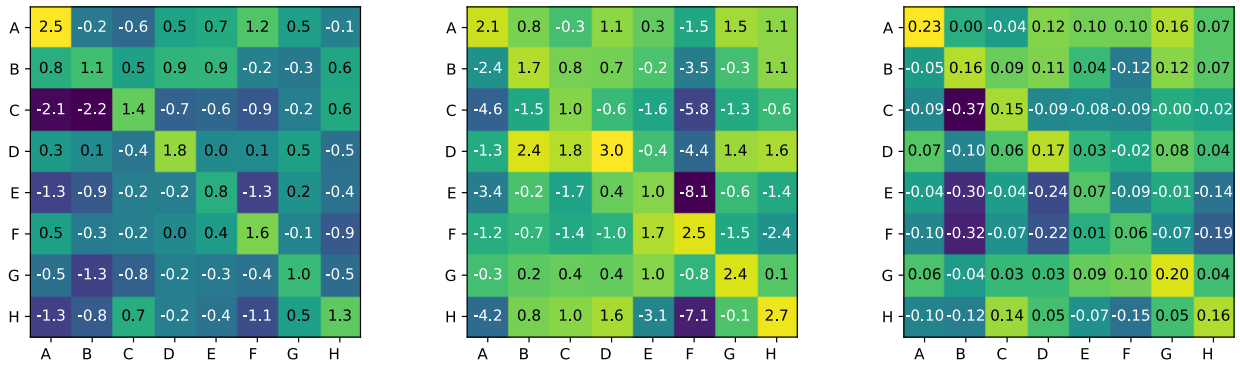
Figure 5.6: Joint denoising and dereverberation results of 2×64 $\tilde{\mathcal{S}}_{\text{DPRNN}}$ on different environments with 0 dB input SNR. The objective scores are measured using ground-truth anechoic targets as the reference.

fine-tuning data. For example, in Fig. 5.5a, 2×256 $\tilde{\mathcal{S}}_{\text{DPRNN}}$ personalized on only 10 seconds of data can outperform the largest generalist.

5.4.3 PSE models' generalization performance on unseen speakers, noises, and room RIRs

Personalization could potentially worsen the generalization performance if a model fine-tuned on a specific test environment must generalize to other unseen test environments comprised of unseen speakers, noise types, or room conditions. The performance degradation is mainly due to the *catastrophic forgetting* phenomenon [155]: fine-tuning on the target test time environment changes the weights that were initially pre-trained on the general-purpose training set. This can be problematic if the model is relocated or the surrounding is changed (e.g., furniture rearrangement or new additions to room such as draping that could alter the acoustics).

We examine this behavior by challenging an already personalized student with a different unseen environment. For this experiment, we design $K = 8$ different environments with balanced speaker gender, noise class and various room dimensions. Details of the configurations can be found in



(a) Δ SI-SDR (b) Δ STOI (c) Δ PESQ

Figure 5.7: Joint denoising and dereverberation results of 2×64 $\tilde{\mathcal{S}}_{\text{DPRNN}}$ on different environments with 0 dB input SNR. In this time, all scores are measured using the teacher model's outputs as the reference.

Table 5.4. The student models are personalized to each k-th room, using noisy reverberant signals generated from $\mathbb{S}_{ft}^{(k)}$, $\mathbb{M}_{ft}^{(k)}$, and $\mathbb{R}_{ft}^{(k)}$ for K different personalized student models in total. The fine-tuned models are then evaluated on each j-th room using set-aside unseen datasets $\mathbb{S}_{te}^{(j)}$, $\mathbb{M}_{te}^{(j)}$, and $\mathbb{R}_{te}^{(j)}$ taken from the same $K = 8$ configurations, i.e., $j \in \{1 \cdots K\}$. Thus, $k = j$ is the desired personalization setup, while $j \neq k$ represents the k-th PSE model challenged to work on the j-th environment. 2×64 RNN student and DPRNN teacher was used to produce the figure. Additive background noise were scaled to 0 dB input SNR.

In Fig. 5.6, we show the relative differences between the pre-trained generalist and personalized student models evaluated on all $K = 8$ environments. We report the result using SI-SDR, short-time objective intelligibility (STOI)[2] [156], and perceptual evaluation of speech quality (PESQ)[3] [103] scores for an in-depth evaluation on personalization and its following effects on other environments. Negative values in the cells indicate performance degradation incurred from personalization. Each j-th cell in the k-th row corresponds to the performances of the model personalized on k-th

[2] `https://github.com/mpariente/pystoi`

[3] `https://github.com/vBaiCai/python-pesq`

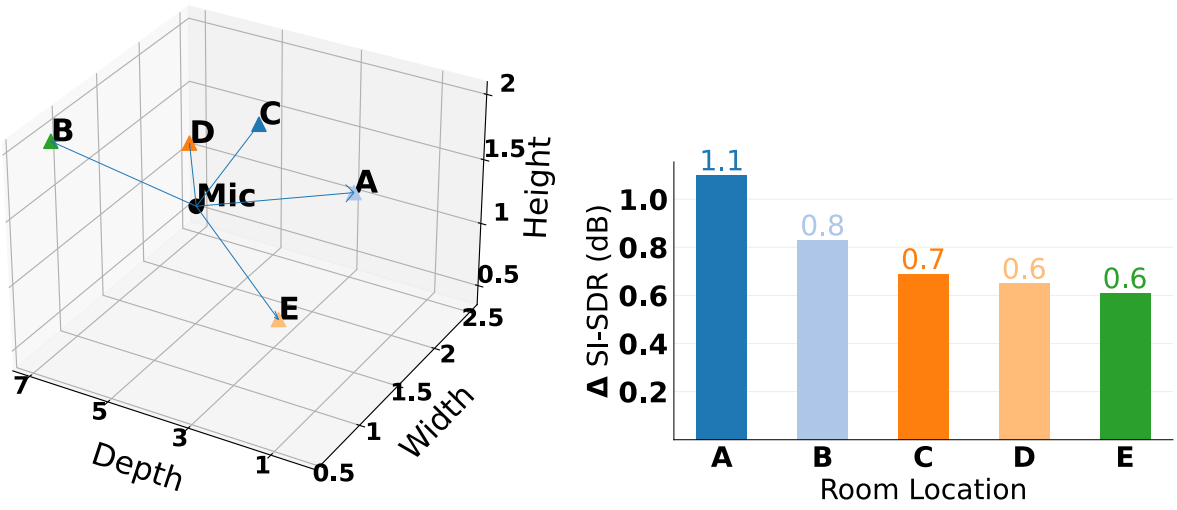
Table 5.4: Descriptions of different rooms configurations in testset for generalization. BUT Reverb Database and Reverb 2014 Challenge datasets are abbreviated as BUT and RVB respectively.

Config. ID	Speaker ID (Gender)	Noise	Room	RIR
A	260 (M)	Crying baby	Q301	BUT
B	1221 (F)	Rooster	E112	BUT
C	1995 (M)	Crackling Fire	CR2	BUT
D	3575 (F)	Car Horn	R112	BUT
E	908 (M)	Sea Waves	Small-Far	RVB
F	1320 (F)	Clapping	Medium-Near	RVB
G	2830 (M)	Crickets	Medium-Far	RVB
H	4992 (F)	Train	Large-Far	RVB

environment and evaluated on the j-th environment. For example, the first row corresponds to the performances of the student-model fine-tuned in environment "A" evaluated on all K configurations.

Unsurprisingly, the diagonal axes generally show highest improvements since those cells represent evaluation results of student models personalized on the same environment. This supports the main argument of our proposed framework. On the other hand, there are under-performing cases such as models fine-tuned on "E" performing poorly on "A" (-1.1 dB ΔSI-SDR) and "F" (-0.9 dB ΔSI-SDR). Interestingly, the inverse relationship does not always hold. Although the model personalized on "B" generalize well to "A" (1.0 dB ΔSI-SDR), the model fine-tuned on "A" does not for "B" (-0.2 dB ΔSI-SDR). These results show that personalizing on one condition can incur negative effects when the model has to generalize to another environment.

Although this reveals a weakness of the proposed framework, this problem can be addressed



(a) Locations of speakers and microphone

(b) Average Δ SI-SDR per source location

Figure 5.8: Denoising and dereverberation evaluation results from fine-tuning in room L212 from location "A" and generalizing to unseen locations.

with a simple solution, by *resetting* or *re-adjusting* the model when sudden worsened performance is detected. However, it is not straightforward to detect such a performance drop during the test time. As a remedy, we propose to compare the PSE result to the teacher model's estimate as an indirect way to evaluate the speech quality. It is because there are no ground-truth test time data available. In Fig. 5.7, we show the results from a same experimental setup with Fig. 5.6, but with the scores computed from using the teacher's estimates as the reference, i.e., the pseudo targets. We see that the SI-SDR, STOI and PESQ scores measured against teacher's estimates (Fig. 5.7) are different from those measured against the ground-truth targets (Fig. 5.6). However, the scores measured using teacher's outputs are still close approximates to the ground-truth metrics. This showcases another merit of using the teacher's estimates. We can reliably use these pseudo metrics to estimate a model's test time performance and decide to reset back to the pre-trained generalist version or to initiate a fine-tuning process to adjust the model to the new test environment.

Figures 5.6b and 5.6c also show that the relative differences in SI-SDR, STOI and PESQ are not

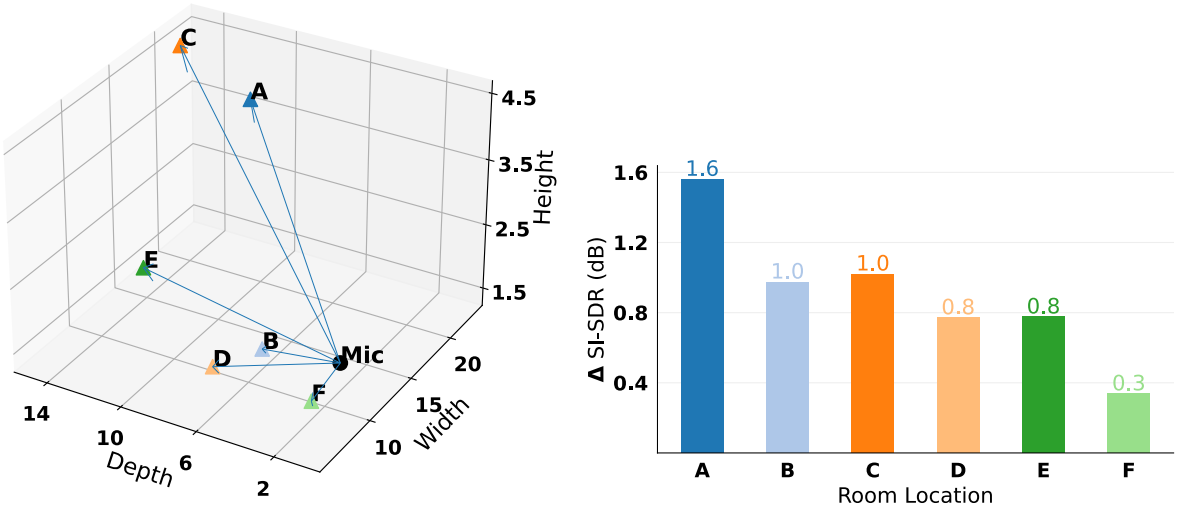
Table 5.5: Dimensions and source-mic distance of Room L212.

Location ID	Location [m×m×m]	Source-Mic Distance [m]
A	0.41×1.13×1.98	4.87
B	6.96×0.77×1.98	2.11
C	5.34×2.48×1.39	0.90
D	4.72×1.32×1.88	0.87
E	3.21×1.67×0.46	2.12

Table 5.6: Dimensions and source-mic distance of Room D105.

Location ID	Location [m×m×m]	Source-Mic Distance [m]
A	11.93×22.93×3.63	13.76
B	5.23×9.90×1.82	4.18
C	14.79×20.79×4.47	14.76
D	6.01×6.06×1.97	7.85
E	9.65×6.22×3.12	10.01
F	0.70×5.48×2.02	7.86

always correlated to one another. Cells with high relative SI-SDR improvements do not necessarily show improvements in intelligibility (e.g., model personalized on "B" evaluated on "A"). This could be due to the loss function defined by ΔSI-SDR to optimize the student-models during the fine-tuning process. A better optimization objective could be explored to prevent such differences. Nonetheless, personalization on intended environments generally shows large improvements without significant performance degradation.



(a) Locations of speakers and microphone

(b) Average Δ SI-SDR per speaker location

Figure 5.9: Denoising and dereverberation evaluation results from fine-tuning in room D105 from location "A" and generalizing to unseen locations.

5.4.4 Generalization performance to unseen locations within a same room

Next, we evaluate on a scenario in which a student model is personalized on a single location of a room and tested on unseen positions within the same room. So far, our experiments utilized all RIRs $\mathbb{R}_{ft}^{(k)}$ from the k-th room to construct the the test time fine-tuning dataset. It was to make the PSE model robust to the variations of RIR filters, which vary vastly depending on the microphone-speaker distance, vicinity to walls or corners, occluding objects, and other factors [157]. In this subsection, we fine-tune a student model using noisy reverberant data generated using a RIR signal from a specified location i within the same k-th room, $\mathbb{R}_{i}^{(k)}$. As with other experiments, utterances from a single user $\mathbb{S}_{ft}^{(k)}$ and one noise type $\mathbb{M}_{ft}^{(k)}$ are used for fine-tuning. For evaluation, we select RIRs from unseen speaker locations within the same room, $\mathbb{R}_{j}^{(k)}$ where $j \neq i$. Unseen speech samples from $\mathbb{S}_{te}^{(k)}$ and noise sources $\mathbb{M}_{te}^{(k)}$ are used to generate the noisy reverberant evaluation set. For brevity, we experiment on 2×64 student models and 0 dB input SNR for additive background

noise.

Two rooms from the BUT Reverb Database were selected for this experiment: a small office (L212) and a large conference room (D105).

Fig. 5.8a shows several speaker positions from room L212 used in this experiment. We fine-tune on noisy reverberant speech from an arbitrary position "A" and evaluate on other locations of the room. We experiment using the same room but on multiple different speakers and noises described in Table 5.4. Fig. 5.8b shows the average generalization results. Despite the changes in location of the test time speech source, fine-tuning on one location of a room can improve dereverberation and denoising results for unseen locations within the same room at least to some degree. The same behavior can be seen in Fig. 5.9 for a much larger room, D105, although the generalization power drops drastically when the unseen location is too different from the one used for fine-tuning, e.g., as in "F". This demonstrates that a stationary personalized device is capable of performing robust speech enhancement on a non-stationary user within the same room, as opposed to suffering from drastic changes in entire room geometry (Sec. 5.4.3).

5.4.5 Use-case scenario for the personalization framework

We emphasize our target use cases in which the joint dereverberation and denoising models are deployed onto edge devices with constrained computing resources, i.e., memory, power, processing cycles, etc. Our proposed framework operates in use case scenarios where no clean ground-truth target speech are available, and compensate for the missing targets via student-teacher framework. During test time, in the background, the noisy signals are sent over to a cloud server. The teacher model and a copy of the student model are envisioned to be placed externally on the cloud server, where the actual fine-tuning operations are conducted in a location where memory and computation are not restrictive. When deploying our framework, we ultimately only use the student model on the device. The student models can be frequently updated on the server side and transferred to

replace the one in the user device.

Frequent cloud communication can seem intrusive and thus unattractive for users. In that case, another use-case for our framework would be where both models are deployed on the device and the fine-tuning procedure is conducted during device idle time. Although model size is no longer reduced, computational complexity can still be lowered from using the personalized student for efficient inference during test time.

Alternatively, we can also deploy our framework onto a smart home ecosystem in which communication occurs internally between local devices. The fine-tuning could take place on a larger capacity home hub device, from which a personalized student model can be deployed to a smaller edge-device (e.g., smart glasses or air pods).

5.5 Conclusion

In this chapter, we proposed a zero-shot knowledge distillation approach to personalizing speech enhancement models for joint dereveberation and denoising. Our goal was to adapt a small model to dynamically changing test time SE environment instead of employing a large generalist model, which can be too heavy for embedded systems. In doing so, we exploited widely available noisy mixtures during test time rather than leveraging ground-truth targets or any extra information of the acoustic environment, which are rarely available in the real-world use cases. To improve the usability of the corrupt examples found in the test scene, our framework synthesizes pseudo-targets by executing a superior-quality SE routine on an overly complex teacher model. We suggest that this knowledge distillation-based personalization can be performed on a regular basis or when a significant change is detected in the test time acoustic scene. Since this fine-tuning task can be performed either in the cloud or when the device is idle, we envision that it is not burdensome for the device.

Evaluation results demonstrate that the student model's performance greatly improves on spe-

cific test time speakers and acoustic environments. The improvements were consistent under various noise and room conditions. Furthermore, the improvements can be seen regardless of model size or the amount of fine-tuning data available: the fine-tuned performance is dependent on the amount of data, but this does not pose a serious limitation on our framework as we can observe improvements even with minimal data. It is also noticeable that the architectural difference between the student and teacher models does not impact the personalization process. Therefore, we expect that our proposed framework can benefit from advancements in the future deep learning-based speech enhancement research. Since our small personalized student model can give superior performances to large generalist models, we claim that the knowledge distillation-based fine-tuning method provides another mode of model compression that does not sacrifice performance.

While fine-tuning on specific environments can harm the generalization on other unseen scenes, the teacher's estimates can again be utilized to gauge the change of environments. A decision can be made to reset the student's parameters back to its pre-trained value, followed by another personalization procedure for further adaptation. Also, our study shows that models personalized on one location can still show improved generalization on unseen locations within the same room, demonstrating robustness to non-stationary sources.

Chapter 6

Evolutionary Optimization for Generative Adversarial Networks

6.1 Introduction

In this chapter, we propose an evolutionary computation strategy, GA-GAN, to utilize genetic algorithms (GA) to optimize generative adversarial networks (GAN). An optimization technique motivated by Darwinian principle of evolution, GA represents models as genes in nature and optimize using operators inspired from natural genetic variation and selection [53]. GA evolves population of models through an abstracted version of evolutionary processes to find solutions to given problems. Each solution is designed as an individual in the population with a corresponding fitness value determining its chances of reproduction. The better (or higher) the fitness, the more descendants they are able to create. Over a series of evolutionary cycles, the fitness of the entire population in subsequent generations is expected to be better on average than that of the previous generation. In the proposed GA-GAN framework, we aim to train our generator and discriminator populations under a coevolutionary training algorithm. Our objective is to increase GAN models' exposure to various adversaries and obtain robust models with efficient compact architectures.

Since GANs have been introduced, they have taken place as the state-of-the-art generative frameworks in various applications, especially in computer vision [8]. Although GANs have made significant advances in generating realistic samples, training these models still remains a challenging task due to their unstable training dynamics causing issues such as mode collapse and vanishing gradients [158]. One potential solution to stabilize training is to model GANs as a coevolutionary problem. As opposed to the regular GAN training, in a coevolutionary optimization framework, populations of discriminators and generators are initialized to compete against each other [159].

The GAN framework is compatible with GA-based evolutionary optimization, or more specif-

ically the crossover operation, as the generator or discriminator from one GAN model can be switched with another. Several methods have been applied for evolutionary computation for GANs using this crossover strategy. A notable work is Pareto GAN [160] that crosses over individual models over GAN pairs and subsequently applies mutation by altering the network structure. Given two GAN individuals $\boldsymbol{P}_1 = (\boldsymbol{G}_1, \boldsymbol{D}_1)$ and $\boldsymbol{P}_2 = (\boldsymbol{G}_2, \boldsymbol{D}_2)$, two offsprings were created as $\boldsymbol{O}_1 = (\boldsymbol{G}_1, \boldsymbol{D}_2)$ and $\boldsymbol{O}_2 = (\boldsymbol{G}_2, \boldsymbol{D}_1)$; hence, the *integrity* of each network is preserved through crossover. Subsequent mutation operation randomly changes the number of hidden units, adds or remotes a random hidden layer from the network, or randomly changes the weight initialization scheme. Lipizzaner [159] is another example that trains adversarial populations against another by crossing over individual GAN models. In particular, Lipizzaner uses a spatial distributed competitive coevolutionary algorithm that spreads GAN populations on a grid and performs genetic operators using a neighborhood communication scheme. Crossover operations are locally performed on neighboring cells within the grid-based model distribution to interchange adversarial pairs. Mutation is executed to modify and search for training hyperparameters that are then used for gradient learning to optimize the newly formed GAN pairs. Coevolutionary Generative Adversarial Networks (COEGAN) [161] incorporates the historical tracking idea from NEAT (Sec. 2.1.5) to enforce a speciation mechanism within each generator and discriminator populations. In this work, only mutation operators were used to add or remove layers and the crossover operator was not performed as it was reported that this operation brought too much instability into the ecosystem.

There are several issues with the aforementioned evolutionary strategies. First, there are no enforced requirements on finding compact architectures. Without a specific mechanism promoting smaller architectures, larger networks tend to eventually dominate the population. Another issue is the use of disruptive mutation operations that drastically change the network architecture (e.g., modifying the number of hidden units and adding or removing layers). Without connections to preserve the network structure, such altercations can significantly alter the network functionality.

For example, adding a layer in the middle of a network and then training the modified network will change the hierarchies of latent features learned by the network unless perhaps a skip connection is added before and after the newly added layer. Moreover, in COEGAN the crossover operation is neglected due to the introduced instability. In the proposed GA-GAN framework, we seek to address these issues of previous works; we aim to promote population diversity by crossing over adversarial pairs and inject mutation to amplify the variations within newly generated offspring populations to produce robust and compact GAN models.

6.2 Proposed Evolutionary Optimization Method for GANs

In this section, we present the GA-GAN framework, a genetic algorithm (GA) based optimization method for generative adversarial networks (GAN).

We first initialize a population of GAN models with each generator and discriminator pair representing an individual. We initially setup the pool with a diverse population to expand the search space of possible model pair combinations. We define an i-th individual $\boldsymbol{P}_i$ consists of a generator and discriminator pair as in $\boldsymbol{P}_i = (\boldsymbol{G}_i, \boldsymbol{D}_i)$. The generator and discriminators within the same pair are initially designed with the same architecture; the j-th generator model $\boldsymbol{G}_j$, is structurally equivalent to $\boldsymbol{D}_j$. We initialize each GAN model with a different network structure using different number of feature map sizes. We denote the feature map size for the generator and discriminator as $N_{\boldsymbol{G}}$ and $N_{\boldsymbol{D}}$. We also tie the learning rate hyperparameter associated to each generator or discriminator such that the model and its corresponding hyperparameters are transmitted together in subsequent crossover operations. The reason being the learning rate (or number of iterations) is an essential hyperparameter for training GANs [162]. By including hyperparamters as part of a generator or discriminator, we aim to search for optimal combinations of learning rates and model architectures. We denote the learning rate of the GAN pairs as $\boldsymbol{\lambda}_j = (\lambda_{j_{\boldsymbol{G}}}, \lambda_{j_{\boldsymbol{D}}})$ where $\lambda_{j_{\boldsymbol{G}}}$ and $\lambda_{j_{\boldsymbol{D}}}$ are learning rates for $\boldsymbol{G}_j$ and $\boldsymbol{D}_j$ respectively. The reason for two separate learning rates is

to discover the good hyper parameters for individual adversarial model. A common heuristic is to initialize $\lambda_{j_{\boldsymbol{G}}}$ with a larger value, yet this is uncertain as these units can be application-specific.

The initial and also the subsequent populations are trained using the GAN objective function as defined in Eq. 2.21:

$$L(\boldsymbol{D}, \boldsymbol{G}) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log \boldsymbol{D}(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - \boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z})))] \tag{6.1}$$

as a minimax optimization $\min_{G} \max_{D} L(\boldsymbol{D}, \boldsymbol{G})$. The discriminator and generator models are updated using $\mathcal{L}_{\boldsymbol{D}}$ and $\mathcal{L}_{\boldsymbol{G}}$ loss functions defined as

$$\mathcal{L}_{\boldsymbol{D}} = \mathbb{E}[\log \boldsymbol{D}(\boldsymbol{x}) + \log(1 - \boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z})))] \tag{6.2}$$

$$\mathcal{L}_{\boldsymbol{G}} = \mathbb{E}[\log(\boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z})))] \tag{6.3}$$

with $\mathcal{L}_{\boldsymbol{G}}$ being the non-saturating version of the loss function. Once the initial population has been trained, we proceed with the evolutionary algorithm. Each evolutionary cycle consists of selecting fit individuals from both populations, crossing over to produce the next population, and applying mutation to induce population variety. Through a series of evolutionary cycles, we aim to give each model more exposure to different opponents to promote their adversarial robustness. The weights of an individual's $\boldsymbol{G}$ and $\boldsymbol{D}$ models are transferred to the next offspring during the crossover process, which are then further trained in the successive generation. Hence, weights of fit models are transferred through generations, and the intermediary solutions contribute to the final solution. Under this setup, the weights are further fine-tuned at every generation. The learning rates are adjusted and explored for the continuous fine-tuning procedure through minor perturbations using the mutation operator. The genetic operators are defined in further detail in the following sections. The overall procedure is illustrated in Fig. 6.1.

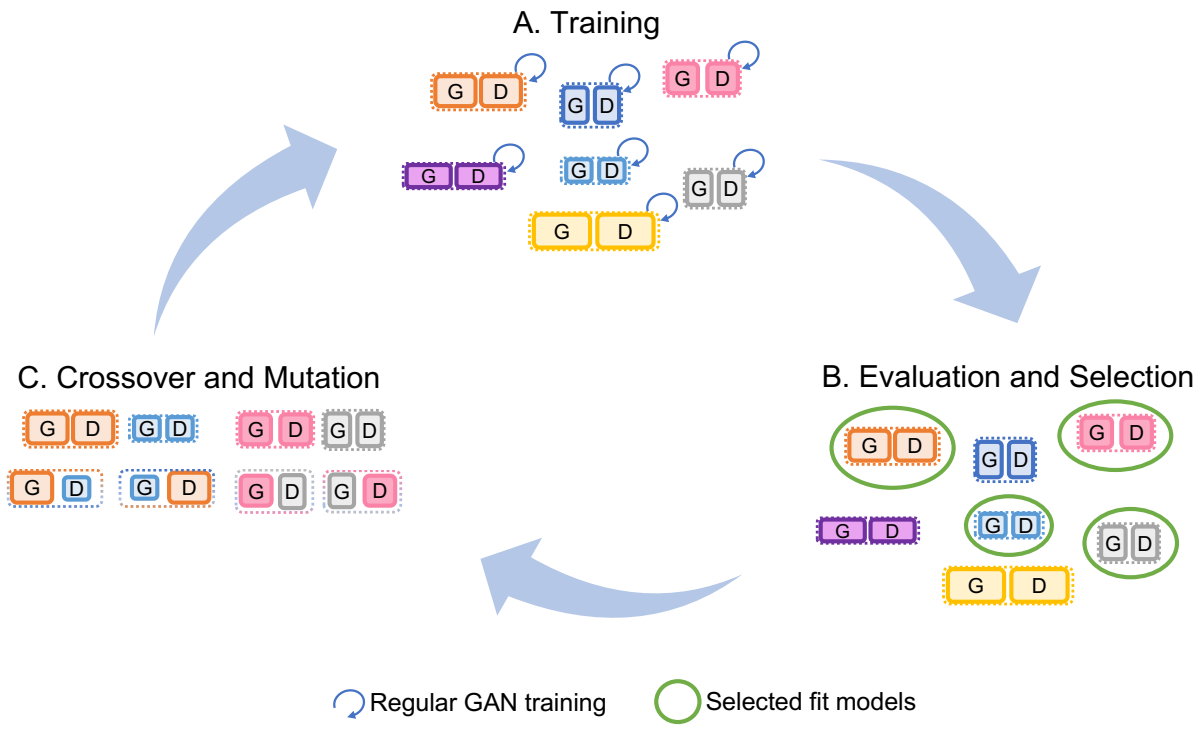


Figure 6.1: The figure illustrates the mechanism of evolutionary optimization for generative adversarial networks. A) At each generation cycle, a population of GAN individuals are trained and placed for the next cycle. B) Each individual is evaluated according to a predtermined fitness function, which then determines its rate of selection for the next evolutionary cycle. C) Selected models proceed to crossover models from generator and discriminator pairs to produce offsprings for the next generation. Our mutation operator consists of perturbing the learning rate of corresponding models by a small constant value.

6.2.1 Fitness

The fitness of the models are assessed by a predetermined performance metric that we denote as f. For our image classification problem, we assess the image quality of the synthesized examples using the Fretchet Inception Distance (FID) [162], which is commonly used as the standard measurement to assess the generative quality of GANs. This objective metric measures the difference between distributions of intermediate features of separate image data samples. Given real image $\boldsymbol{x}$ and generated sample $\boldsymbol{g}$, the FID is computed as:

$$\text{FID}(\boldsymbol{x}, \boldsymbol{g}) = ||\mu_{\boldsymbol{x}} - \mu_{\boldsymbol{g}}||_2^2 + Tr(\Sigma_{\boldsymbol{x}} + \Sigma_{\boldsymbol{g}} - 2\sqrt{\Sigma_{\boldsymbol{x}}\Sigma_{\boldsymbol{g}}}) \tag{6.4}$$

where $(\mu_{\boldsymbol{x}}, \Sigma_{\boldsymbol{x}})$ and $(\mu_{\boldsymbol{g}}, \Sigma_{\boldsymbol{g}})$ are the mean and covariance of the real and generated data distributions respectively. A subset of the real and fake images created by the generator is sampled to compute the FID score. In [163], they found that the FID score is a good representation of diversity and quality than other metrics, such as the Inception Score [164]. Moreover, the FID is sensitive to disturbances in the images and coincides with human perceptual judgment. We evaluated all individuals' FID scores after a fixed number of 1000 iterations which was arbitrarily set to reach near-convergence while also reducing the computation time. We assign the FID score as the fitness for both GAN model pairs to push the population pool into producing stronger generators with respect to this metric. Although the FID score is evaluated only on the synthesized images from the generator, we assign it as the fitness for the whole individual (both generator and discriminator). since the discriminator also contributes to improving the quality of the generator through the adversarial training procedure. Other related works such as COEGAN also used the FID metric as the overall individual fitness, which was shown to be more effective than using a separate fitness metric for the discriminator (e.g., discriminator loss $\mathcal{L}_{\boldsymbol{D}}$).

6.2.2 Selection

We use a tournament selection strategy for selecting the parents that will move onto produce the next generation of individuals [162]. Specifically, we use binary tournament selection that pools two individuals from the population and selects the fitter individual [165]. The binary tournament strategy only compares using local fitness values of two selected individuals, yet it incurs low computational complexity which is advantageous when executing extensive GAN training for all individuals over multiple generations.

Speciation

When pairing two individuals we divide the population of generators and discriminators into separate species based on the model size. During selection, only similar sized models are compared to enforce fair representation from all individuals especially for small models that will have an unfair disadvantage when paired with a disproportionately larger model. Each model is assigned a partner and duplicate pairing is allowed where one model can be assigned to more than one partner. Otherwise, the imposed size restriction leaves some models with no matching compatible partners. Specifically, two models $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$ are selected and their fitness compared only if $N_{\boldsymbol{G}_i} = scale \times N_{\boldsymbol{G}_j}$ where $scale \in \{\frac{1}{2}, 0, 2\}$. For example, given $\boldsymbol{G}_i$ with $N_{\boldsymbol{G}_i} = 32$, it will only be paired with $\boldsymbol{G}_j$ of $N_{\boldsymbol{G}_j} = \{16, 32, 64\}$.

Population Control

Tournament selection typically removes the worse of the two individuals, which halves the population size when executed over all the individuals. The subsequent crossover operation doubles the population and thereby maintains the population size to be constant throughout the generations. However, this removal is not imposed for duplicate pairing such that all individuals are paired. Without elimination, the population could potentially double every cycle and grow at an unsustainable exponential rate. To control such unrestricted population growth, models only participate

in maximum number of two consecutive generations and are removed afterwards. A side benefit from setting a maximum number of generations is it also prevents a local optimum solution from dominating the population. In addition, the duplicate pairing strategy contributes to control the population size as some fit models can be selected over all its other pairs. A locally most fit model reduces the chances of the unfit models from proceeding further. If those unfit models are also worse than all its other pairs, then they will be eliminated.

6.2.3 Crossover

Given two parents $\boldsymbol{P}_i = (\boldsymbol{G}_i, \boldsymbol{D}_i)$ and $\boldsymbol{P}_k = (\boldsymbol{G}_k, \boldsymbol{D}_k)$ from the selection process, we perform crossover to produce two offsprings $\boldsymbol{O}_i = (\boldsymbol{G}_i, \boldsymbol{D}_k)$ and $\boldsymbol{O}_k = (\boldsymbol{G}_k, \boldsymbol{D}_i)$ by swapping one of the models from the GAN pair. Additionally, their respective hyperparameters (i.e. learning rates) are also transmitted along with the model as well. Given $\boldsymbol{\lambda}_i = (\lambda_{i_{\boldsymbol{G}}}, \lambda_{i_{\boldsymbol{D}}})$ and $\boldsymbol{\lambda}_k = (\lambda_{k_{\boldsymbol{G}}}, \lambda_{k_{\boldsymbol{D}}})$ from two parents, we set the learning rates of the two offsprings as $\hat{\boldsymbol{\lambda}}_i = (\lambda_{i_{\boldsymbol{G}}}, \lambda_{k_{\boldsymbol{D}}})$ and $\hat{\boldsymbol{\lambda}}_k = (\lambda_{k_{\boldsymbol{G}}}, \lambda_{i_{\boldsymbol{D}}})$ respectively. Through this selection process, each model in a GAN pair is exposed to a new adversary at every generation.

6.2.4 Mutation

To promote variation in our population, we apply mutation on individual GAN models. Our mutation operator consists of perturbing the learning rate hyperparameter by a small constant value $\epsilon = 1e^{-5}$ such that $\boldsymbol{\lambda}_i \leftarrow (\lambda_{i_{\boldsymbol{G}}} \pm \epsilon, \lambda_{i_{\boldsymbol{D}}} \pm \epsilon)$ as done in [159]. The model architecture used for both the generator and discriminator is fixed and defined at initialization.

6.3 Experiments

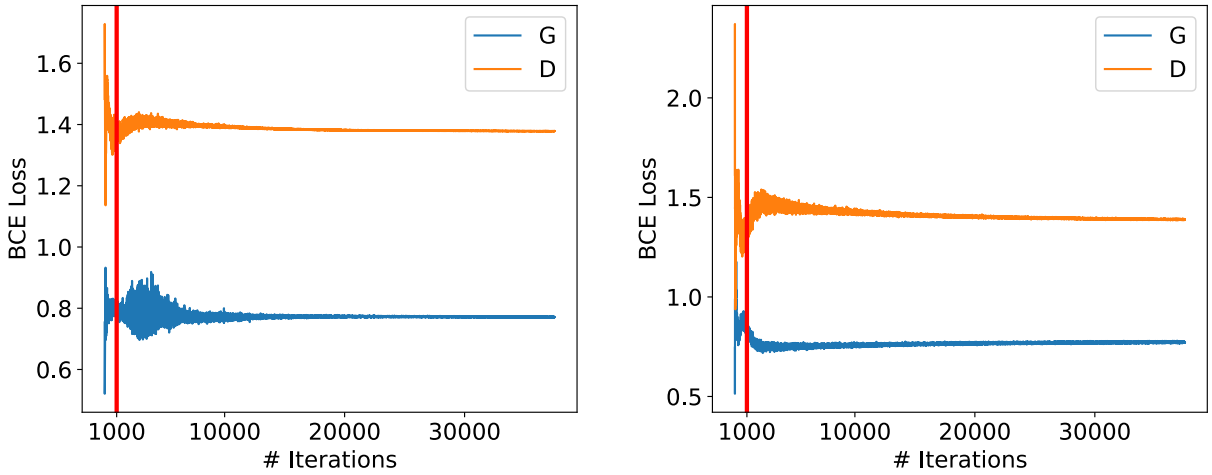
6.3.1 Datasets

We first begin our experiments on the image classification task on the Fashion MNIST [166] dataset to work on images from a more complex data distribution than the well known MNIST handwritten digit dataset [167]. For the Fashion MNIST dataset, we partition the provided training set into separate training and validation sets by a 80:20 ratio. We use the dataset as provided by TorchVision [1]. The database contains 60,000 training images and 10,000 testing images. Each sample is a 28×28 grayscale image, associated with labels from 10 classes: t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. We use the default shape of Fashion MNIST images.

6.3.2 Models

We use a smaller variant of the Deep Convolutional Generative Adversarial Network (DCGAN) [168]. The generator $\boldsymbol{G}$ takes as input a noise vector $\boldsymbol{z} \in \mathbb{R}^K$ and synthesizes an image $\boldsymbol{G}(\boldsymbol{z}) \in \mathbb{R}^{1\times28\times28}$. The generator model consists of 4 transpose convolutional layers, batch normalization layers, and ReLU activations except for the last layer, which is followed by tanh activation. We set the number of intermediate channels $N_{\boldsymbol{G}}$ to be uniform for all layers. We set $K = 100$ for our experiments. The discriminator $\boldsymbol{D}$ takes an input image $\boldsymbol{x} \in \mathbb{R}^{1\times28\times28}$ and outputs a probability $\boldsymbol{D}(\boldsymbol{x}) \in [0, 1]$ determining whether it is real or fake. The discriminator model consists of 4 convolutional layers, batch normalization layers, and leaky ReLU activations except for the last layer, which is followed by sigmoid activation. We also set the number of intermediate channels $N_{\boldsymbol{D}}$ to be uniform. We initialize the models with different feature map sizes $N_{\boldsymbol{G}}, N_{\boldsymbol{D}} \in [16, 32, 64, 128, 256, 512]$ and learning rate pairs $\boldsymbol{\lambda_j} = (\lambda_{j_{\boldsymbol{G}}}, \lambda_{j_{\boldsymbol{D}}}) \in [(1e^{-4}, 5e^{-4}), (2e^{-4}, 2e^{-4}), (5e^{-4}, 1e^{-4})]$ for generator and discriminator model pairs. We iterate over 1000 batches with 128 samples per batch for a single

[1] https://pytorch.org/vision/main/modules/torchvision/datasets/mnist.html



(a) Training losses of $N = 64$

(b) Training losses of $N = 256$

Figure 6.2: Training results from generator's and discriminator's binary cross entropy losses from the initial population. Vertical red bars indicate the stopping checkpoint of the individuals that proceed to the following generation.

generation cycle and run for a total of 10 generations.

The motivation for initializing varying learning rate pairs is to discover the ideal values for our application. Existing heuristics claim to train the discriminator less since it performs a simpler task than the generator. While the generator synthesizes realistic samples from random noise, the discriminator performs a binary classification task. The original GAN formulation requires updating the generator for at least 5 iterations for every update on the discriminator [8]. This has also been implemented by using a higher learning rate for generator [162]. However, it is uncertain if this rule holds for all cases as hyperparameters can be application-specific.

The combination of sizes of feature maps and learning rates gives a total 18 models for our initial population. After the initial training phase, we use the trained GAN model pairs for the evolutionary training procedure.

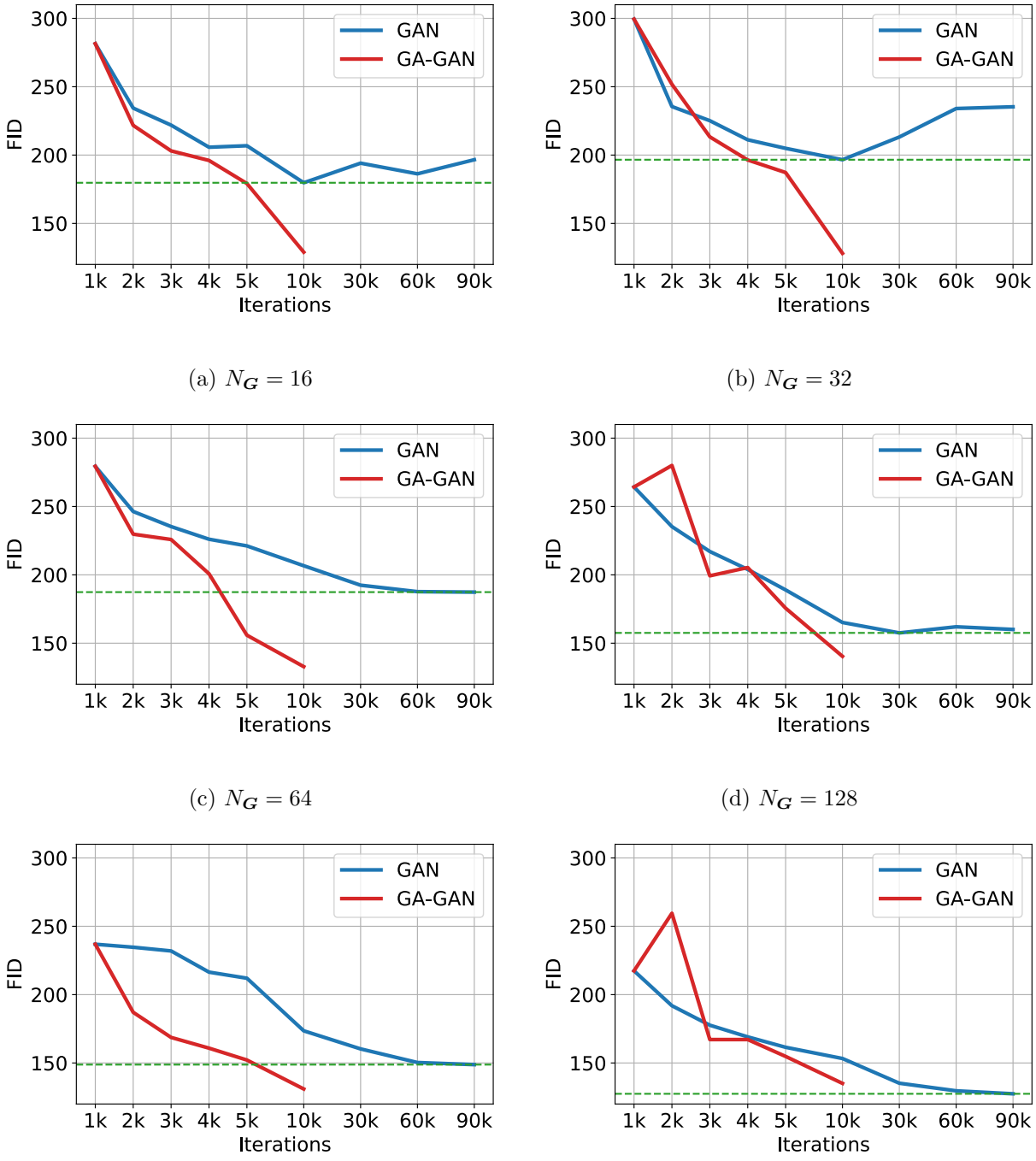
Table 6.1: Model sizes of generators and discriminators with various feature map sizes N

N	Model Size (MB)	
	G	D
16	0.13	0.03
32	0.31	0.11
64	0.82	0.42
128	2.47	1.66
256	8.21	6.59
512	29.54	26.29

6.4 Results

We begin our discussion with the results from the first generation training on the initial population. Fig. 6.2 shows example training results from GAN models with $N_{\boldsymbol{G}} = 64$ and $N_{\boldsymbol{G}} = 256$. Generator and discriminator models in both figures appear to reach near convergence. Although further training could be beneficial, the models stopped learning at 1000 iterations (red vertical bar) for training consistency for each generation.

Fig. 6.3 shows the Frechet Inception Distance (FID) of the best generators trained under regular GAN and GA-GAN settings. At each subplot, the $N_{\boldsymbol{G}}$ values are the same for generators under the regular GAN and GA-GAN setting. At initialization, there are a total 3 individuals with the same $N_{\boldsymbol{G}}$ and $N_{\boldsymbol{D}}$ values (from using 3 different learning rate pairs for each $N_{\boldsymbol{G}}$), and the initial population pool contains total 18 individuals consisting of 3 individuals from each $N_{\boldsymbol{G}}$ value. We plot the FID scores of the best models; hence, both GAN and GA-GAN report the same initial FID values. For the GA-GAN setting, after the first generation (i.e., 1k iterations), the crossover operations assign each model with a different discriminator such that it is likely that $N_{\boldsymbol{G}} \neq N_{\boldsymbol{D}}$.



(a) $N_{\boldsymbol{G}} = 16$

(b) $N_{\boldsymbol{G}} = 32$

(c) $N_{\boldsymbol{G}} = 64$

(d) $N_{\boldsymbol{G}} = 128$

(e) $N_{\boldsymbol{G}} = 256$

(f) $N_{\boldsymbol{G}} = 512$

Figure 6.3: FID fitness scores of models trained under regular GAN and GA-GAN frameworks on the Fashion MNIST dataset. The dotted green line represents the minimum FID score achieved by the regular GAN baseline. Each subplot represents the scores of generator models grouped by the same $N_{\boldsymbol{G}}$ values. $N_{\boldsymbol{D}}$ can differ for each generator model under the GA-GAN setting.

So each of the subplots report the best FID scores of generators with the specified $N_{\boldsymbol{G}}$ that can be paired with a different $N_{\boldsymbol{D}}$ at each generation. For the regular GAN setting, the subplots report the performance improvements of the same GAN model that is continuously trained from the first generation.

For most generator feature map sizes $N_{\boldsymbol{G}}$, we can see that the FID for GAGAN reaches lower scores faster than its regular counterpart with fewer iterations. Also, for all feature map sizes $N_{\boldsymbol{G}}$, our GA-GAN models trained for 10 generations can achieve FID scores better than a regular GAN with $N_{\boldsymbol{G}} = 256$. For example, the GA-GAN with $N_{\boldsymbol{G}} = 16$ (Fig. 6.3a) can outperform a larger regular GAN with $N_{\boldsymbol{G}} = 256$ (Fig. 6.3e). Table 6.1 lists the respective sizes for both generator and discriminator models with varying $N_{\boldsymbol{G}}$. The $N_{\boldsymbol{G}} = 16$ generator (0.13MB) is approximately 98.5% smaller than the $N_{\boldsymbol{G}} = 256$ model (8.21MB).

For the largest GAN model architecture with $N_{\boldsymbol{G}} = 512$ (Fig. 6.3f), the regular version reaches lower FID levels (f=127.26) than its GA-GAN counterpart. The worse GA-GAN performance for $N_{\boldsymbol{G}} = 512$ could be contributed to the experimental setup in which the $N_{\boldsymbol{G}} = 512$ generators are only assigned a discriminator with the same or lower $N_{\boldsymbol{G}}$ values. Since they do not meet a *stronger* adversary, we do not see much improvement. Another reason is larger models require more training iterations. In Fig. 6.3d and 6.3f we observe erratic improvements with a spike at the second generation (i.e. 2000 iterations). This could be attributed to a small initial population pool to progress into the second generation, and can be alleviated by enlarging the number of individuals or by repeating more executions and taking the average of the results.

Fig. 6.4 illustrates real and synthesized samples of the Fashion MNIST dataset generated by the best generators trained under regular GAN and GA-GAN frameworks. First, we compare the samples produced by the regular GAN with real images. Samples in Fig. 6.4b are generated by the largest and highest performing regular GAN model with $N_{\boldsymbol{G}} = 512$. Compared to the real samples in Fig. 6.4a, the synthesized samples do not appear perfect and contain some artifacts. Since this

(a) Real samples

(b) GAN: $N_{\boldsymbol{G}} = 512$

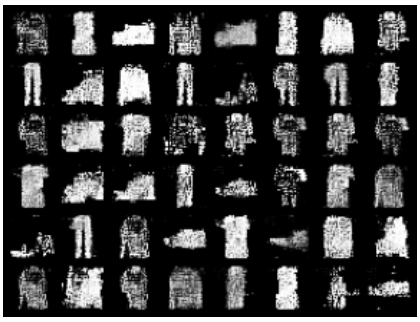
(c) GA-GAN: $N_{\boldsymbol{G}} = 16$

(d) GA-GAN: $N_{\boldsymbol{G}} = 32$

(e) GA-GAN: $N_{\boldsymbol{G}} = 64$

(f) GA-GAN: $N_{\boldsymbol{G}} = 256$

Figure 6.4: Samples generated by regular GAN and GA-GAN trained on the Fashion MNIST dataset



(a) GAN: $N_{\boldsymbol{G}} = 32$

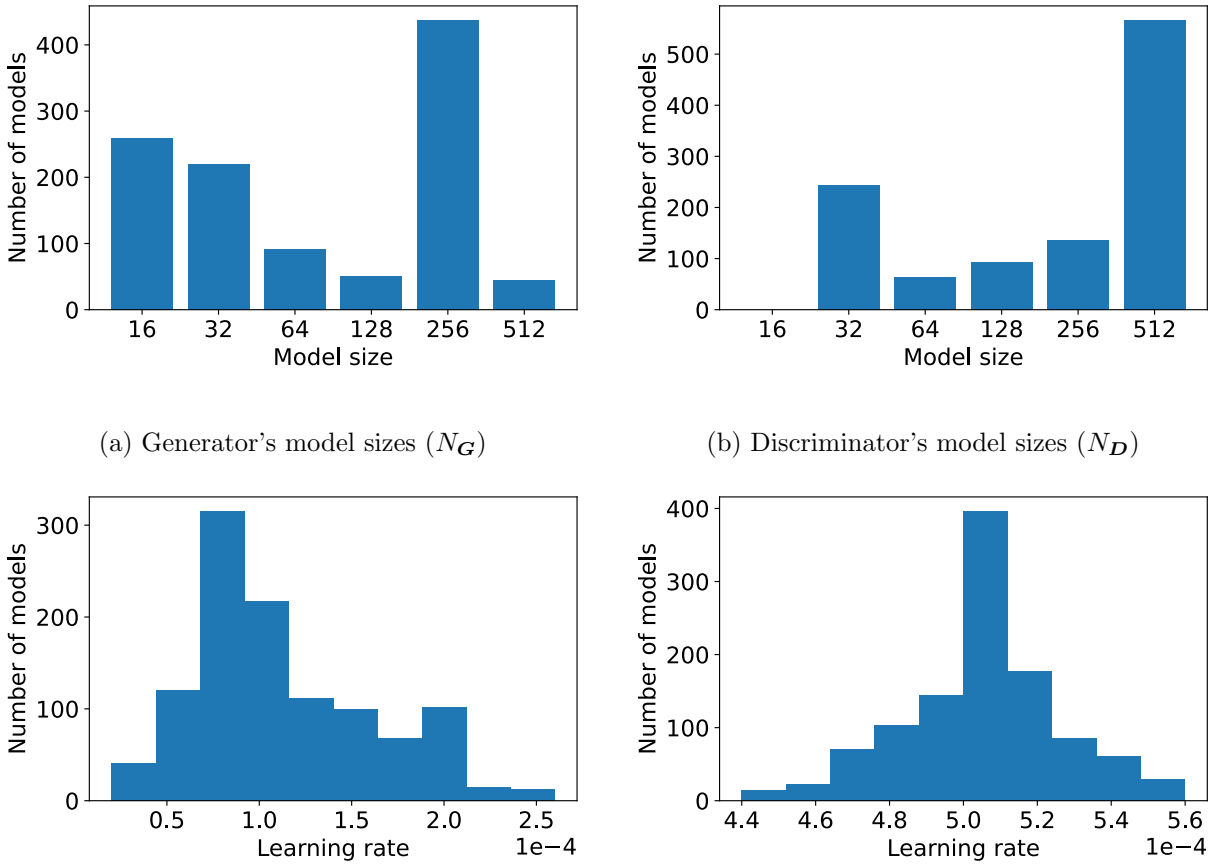
(b) GA-GAN: $N_{\boldsymbol{G}} = 32$

Figure 6.5: Comparison between samples generated by regular GAN and GA-GAN with $N_{\boldsymbol{G}} = 32$ on the Fashion MNIST dataset

model achieves the best FID scores, we recognize it and its generated samples as the upper bound for the experiment. Fig. 6.4c through 6.4f contain images from the best GA-GAN models after 10 generations. Although the samples are imperfect, the samples generated by the GA-GAN model with $N_{\boldsymbol{G}} = 16$ produces high quality samples (Fig. 6.4c) and is comparable to those produced by the regular GAN with $N_{\boldsymbol{G}} = 512$ (Fig. 6.4b). Next, we make a close comparison between a regular GAN and GA-based GAN for $N_{\boldsymbol{G}} = 32$ in Fig. 6.5. From the figures, we can observe that the samples produced by a regular GAN model (Fig. 6.5a) shows a grainy texture while those from GA-GAN are smoother, demonstrating the benefits of our GA-GAN framework.

From the figures, we can observe a good representation of various classes in the Fashion MNIST dataset displayed by GA-GAN based models. Common problems such as mode collapse and vanishing gradient that surround GAN training do not appear in the executions of GA-GAN. Since models that suffer from these issues perform worse than others (i.e. higher FID), they are eventually not selected by the evolutionary algorithm and are prevented from persisting these problems through further generations. Furthermore, by setting FID as the fitness measure, the training process with a diverse population of generators and discriminators further increase variability in the

models compared to a regular GAN. This increased variation contributes to a stronger population and training algorithm, preventing mode collapse and vanishing gradient issues.



(a) Generator's model sizes ($N_{\boldsymbol{G}}$)

(b) Discriminator's model sizes ($N_{\boldsymbol{D}}$)

(c) Generator's learning rates ($\lambda_{\boldsymbol{G}}$)

(d) Discriminator's learning rates ($\lambda_{\boldsymbol{D}}$)

Figure 6.6: Final model size and learning rate distribution of generator and discriminator models after 10 generation cycles.

Fig. 6.6 shows the distribution of model sizes and learning rates for generator and discriminator models after the last generation cycle. While only few of the largest generator models remain after the final evolutionary cycle, none of the smallest discriminator models survive and only the largest individuals dominate the population. Furthermore, the learning rates of the discriminator population is on average larger than the generator's. Starting from an initial population with varying learning rates, the distribution of learning rates converges close to $\lambda_{\boldsymbol{G}}, \lambda_{\boldsymbol{D}} = (1e^{-4}, 5e^{-4})$.

6.5 Discussion

At the end of the evolutionary process, our GA framework produces small generators paired against larger discriminators with higher learning rates. However, this does not necessarily indicate that initializing generators against a stronger discriminator with larger learning rates is beneficial for training GAN models for this specific application. Because the small generators were tasked with producing samples realistic enough to fool various discriminators, they were able to achieve good FID scores. The same results would not be reached without the multi-adversarial training strategy.

Furthermore, under the proposed GA-based evolutionary process, individual models are continuously trained against stronger adversaries after each cycle. Since both generator and discriminator models are fine-tuned on a more challenging objective, this creates an increasingly competitive environment in which the models are pushed to improve; otherwise, they would not be selected to proceed to the next generation. So, the average performance of the population is expected to improve or at least stay the same at each generation. The speciation technique also prevents individuals from being paired against models that are either relatively too small (or weak) or too large (or strong). Additionally, the proposed GA-GAN approach searches for the corresponding learning rates of newly formed GAN models after each generation. Since each offspring is paired with a new adversary, the training objective changes and the learning rates need to be adjusted to for fine-tuning. It is uncertain how these values should be changed, but this is addressed in our GA-based framework through various genetic operators (e.g., perturbed learning rates with mutation).

Comparatively, a random sampling approach for pairing generators and discriminators in the population would similarly continuously train and expose models to multiple adversaries. However, without a mechanism to select fit models, this method is prone to pairing models against poorly matched adversaries. When trained against a significantly weaker opponent, neither the generator nor discriminator model will improve since the adversary would be easy to fool or the generated

samples would be easily classified as fake respectively. Meanwhile, being paired against a significantly stronger opponent would impair learning due to the known vanishing gradient issue for the generator. Training against less than ideal opponents can potentially harm the models and degrade the overall performance of the population. Moreover, it is uncertain how the learning rates should be adjusted under this approach.

To go further, an exhaustive mix-and-match pairing strategy over multiple generations would face the same issue since there is still no mechanism to filter out the poorly trained models. Also, this strategy would require a computational cost of $|\boldsymbol{P}|^2$ at every generation since each model is exhaustively paired against all available opponents; whereas, our proposed GA-GAN framework (and also random sampling) would cost $|\boldsymbol{P}|$ per cycle. The evolutionary process using GA provides a more efficient and effective optimization scheme for an iterative multi-adversarial training procedure that is not attainable from using a random or exhaustive approach.

It is possible to apply heuristics to optimize model pairing and learning rate adjustments to design a similar iterative multi-adversarial training strategy. An example would be to pair each generator with a slightly larger discriminator and decrease/increase the learning rates of the generator/discriminator after every generation. However, there can be numerous heuristics and biases, which would need to be all tested and could be flawed and fail to produce the desired results. The proposed GA-based approach alleviates this issue and automates the search and pairing process for the learning rates and models respectively, which showed promising results in our experiments.

Nonetheless, GA-GAN uses its own set of heuristics. One instance is the speciation mechanism involved in the selection operator that restricts the models to be paired with similar sized individuals. Furthermore, both approaches need the learning rate adjustments for fine-tuning to be determined with human knowledge and intervention (i.e., manual adjustments in GAN vs. mutation rate in GA-GAN). Both approaches apply their own respective heuristics, and the advantages and disadvantages of these methods would need to be considered.

6.6 Use-cases

As mentioned in the previous application on personalization (Sec. 5.4.5), smaller high-performing models have an advantage of not requiring specialized hardware. Rather, the model structure is simplified with reduced parameters, easing resource requirements for deployment. In addition, since models with different sizes are included during initialization, GA-GAN produces robust models of various sizes by the final evolutionary cycle. This permits users to flexibly select the model of their choice depending on their desired performance and resource constraint; thus, adding on scalability as an extra benefit of this framework.

6.7 Conclusion

In this chapter, we proposed an evolutionary training strategy for generative adversarial networks (GAN). We presented a genetic algorithm (GA) based framework, GA-GAN, that makes use of adversarial characteristics of GAN to coordinate with a coevolutionary environment. The GA-based evolutionary process creates a sequentially more competitive environment in which the models are not only exposed to multiple adversaries but also to stronger opponents at each evolutionary cycle. At every generation, the generative models are trained to synthesize samples of high enough quality to fool different stronger discriminators, which has an additional benefit of preventing them from converging to a narrow range of sample data distribution. Genetic operators, specifically mutation, adjust the learning rates of individual offspring to facilitate fine-tuning on a new objective. The proposed GA-GAN approach reliably and efficiently selects fit adversaries that are well-matched in terms of inference performance (i.e., classification and synthesis quality of generators and discriminators respectively). From our experimental results, we can observe that GA-GAN is a faster converging and better performing training solution than a regular GAN for most $N_{\boldsymbol{G}}$ cases. Starting from a population initialized with varying model sizes and learning rates, the generator pool inclined towards smaller models with lower learning rates while being paired with stronger discrim-

inators with larger learning rates. These smaller models trained displayed no evidence of vanishing gradient and mode collapse, and also showed capable of outperforming larger models trained under the regular GAN training scheme; thus, showcasing the benefits of the GA-GAN framework.

Chapter 7

Conclusion

Recent advances in deep learning models have shown superior performance to traditional machine learning and signal processing methods in many applications. However, the growing model size and computational complexity renders them difficult to deploy onto resource-constrained devices. Hence, model compression methods have been gaining interest to facilitate the practicality of deep-learning architectures in real-time applications. In this dissertation, we introduce various modes of model compression to facilitate efficient deployment through various model compression methods.

In Chapter 2, we proposed a BGRU network for the single-channel source separation task. We focus on the GRU cells and quantize the feedforward procedure with binarized values and bitwise operations. The real-valued weights are pretrained and transferred to the bitwise network, which are then incrementally binarized to minimize the potential loss that can occur from a sudden introduction of quantization. Although quantization methods can achieve high compression rates, it can induce a significant amount of quantization noise (especially with binarization) and it is difficult to restore the drop in performance from further fine-tuning procedures (i.e., quantization-aware training). Although we attempted to minimize the loss in performance using an incremental binarization procedure in Chapter 2, our experiments still showed drastic drops in performance from 16.12 dB to 11.76 dB in SDR and 94.6% to 87.4% in STOI (Table 3.1). Even after significant amounts of fine-tuning, the bitwise model still suffered from a 4.36dB and 7.2% drop in SDR and STOI. In addition, quantized models, especially for bitwise neural networks, demand specialized hardware designs to support bitwise storage and operations, which otherwise cannot provide the maximum benefits from binarization.

We extend binarization to explore solutions that can best preserve the discriminative properties of dataset features using only binary representations. In Chapter 3, we introduce learnable locality

sensitive hash functions to extract higher-level features and better encode the underlying structure of audio data in the form of STFT spectrograms into bitstrings. We proposed an adaptive boosting approach to learn locality sensitive hash functions as the weak learners. Once trained, their binary classification results transform each spectrum of test noisy speech into a bit string. Our data-driven approach learned locality sensitive hash codes that could preserve the original similarity between the hash codes and serve as more discriminative features than the raw Fourier coefficients. Compared to the highest performing real-valued configurations, the best BLSH systems scored 0.41 dB higher in SDR ($\texttt{IN}_{\texttt{STFT}}\texttt{-SSM-BLSH}_{L=300,\rho=0.1}$ vs. $\texttt{IN}_{\texttt{STFT}}\texttt{-KNN}_{\rho=0.1}$) and 1.98% higher in ESTOI ($\texttt{IN}_{\texttt{mel}}\texttt{-SSM-BLSH}_{L=50,\rho=0.1}$ vs. $\texttt{IN}_{\texttt{mel}}\texttt{-KNN}_{\rho=0.1}$). Using a data-driven approach, we were able to not only overcome the performance drops incurred from compression but also to outperform the real-valued methods.

In Chapter 4, instead of applying compression as a post-training and fine-tuning procedure on large complex models trained on generic training sets, we implement a personalization framework to adapt a compact denoising model to the test-time specificity. We utilize no clean speech target of the test speaker in this test-time adaptation procedure. We employ the knowledge distillation framework where we distill the more advanced denoising results from an overly large teacher model, and use them as the pseudo target to train the small student model. Through our experiments, we demonstrated that our small personalized student model can compete against much larger generalist models. For instance, our 2×32 $\tilde{\mathcal{S}}_{\text{DPRNN}}$ performed similarly to a 2×1024 $\mathcal{S}$ for -5 dB input SNR setting, which corresponds to a 99% reduction in terms of spatial and arithmetic complexity with no sacrifice in performance. Furthermore, personalized compact models not only outperform larger general-purpose models, but also do not require dedicated hardware since only the model structure is reduced.

Then, in chapter 5 we proposed an evolutionary strategy targeting GANs to train small models capable of outperforming larger and more complex models trained under a regular GAN framework.

GANs have shown promising efforts in estimating the target data distribution; however, its unstable training (i.e. mode collapse, vanishing gradients, uncertain architecture and hyperparameters) prevents its widespread adoption to various domains and applications. In this study, we proposed a solution utilizing genetic algorithms to stabilize the training of GANs and additionally learn robust compact GAN models. By coordinating the adversarial characteristics of GANs with a coevolutionary strategy, our GA-GAN framework demonstrated to be capable of producing small yet high-performing models that can be used for efficient and faster inference during run-time. Small models trained under the evolutionary process were able to outperform larger models trained under the regular GAN framework. For instance, a $N_{\boldsymbol{G}} = 16$ generator (0.13MB) trained under GA-GAN can outperform a larger $N_{\boldsymbol{G}} = 256$ model from regular GAN (8.21MB), which is close to 98.5% reduction in model size. Moreover, as in the knowledge-distillation framework, models trained via GA-GAN have an additional benefit of not requiring specialized hardware for deployment.

In summary, this dissertation proposes to bridge the gap between research and practical deployment for real-time applications. We proposed several methods to facilitate efficient deployment through various model compression methods. We started from extreme quantization to drastically reduce the bit-widths of data and model parameters, and then addressed worsened generalization and hardware compatibility issues entailing compression. By utilizing personalization and genetic algorithms, we were able to train small yet competitive models capable of outperforming larger counterparts; thereby, achieving model compression with no loss in performance.

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] C. Subakan, M. Ravanelli, S. Cornell, M. Bronzi, and J. Zhong, "Attention is all you need in speech separation," in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2021, pp. 21–25.

[3] A. Gulati, J. Qin, C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, and Y. Wu, "Conformer: Convolution-augmented transformer for speech recognition," in *Proc. of the Annual Conference of the International Speech Communication Association (Interspeech)*, 2020.

[4] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[5] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. of the International Conference on Learning Representations (ICLR)*, 2015.

[6] Y. Luo and N. Mesgarani, "Tasnet: time-domain audio separation network for real-time, single-channel speech separation," in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2018.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[9] U. Thakker, G. Dasika, J. Beu, and M. Mattina, "Measuring scheduling efficiency of rnns for nlp applications," *arXiv preprint arXiv:1904.03302*, 2019.

[10] M. Li, J. Lin, Y. Ding, Z. Liu, J. Zhu, and S. Han, "Gan compression: Efficient architectures for interactive conditional gans," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5284–5294.

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

[12] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, and A. Askell, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[13] T. Park, M. Liu, T. Wang, and J. Zhu, "Semantic image synthesis with spatially-adaptive normalization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2337–2346.

[14] J. Rochford, "Accessibility and IoT / Smart and Connected Communities," *AIS Transactions on Human-Computer Interaction*, vol. 11, no. 4, pp. 253–263, 2019.

[15] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[16] J. Lin, W. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," *arXiv preprint arXiv:2007.10319*, 2020.

[17] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[18] B. Jacob, S. S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[19] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient ConvNets," *arXiv preprint arXiv:1608.08710*, 2016.

[20] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[21] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Proceedings of the Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

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

[23] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 3105–3113.

[24] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[25] Y. Bai, Y. Wang, and E. Liberty, "Proxquant: Quantized neural networks via proximal operators," *arXiv preprint arXiv:1810.00861*, 2018.

[26] M. Kim and P. Smaragdis, "Bitwise neural networks," in *International Conference on Machine Learning (ICML) Workshop on Resource-Efficient Machine Learning*, Jul 2015.

[27] ——, "Bitwise neural networks for efficient single-channel source separation," in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2018.

[28] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[29] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proc. of the European Conference on Computer Vision (ECCV)*, 2016, pp. 525–542.

[30] P. Indyk and R. Motwani, "Approximate nearest neighbor – towards removing the curse of dimensionality," in *Proc. of the Annual ACM Symposium on Theory of Computing (STOC)*, 1998, pp. 604–613.

[31] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 891–923, 1998.

[32] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 2004, pp. 253–262.

[33] R. Salakhutdinov and G. Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, vol. 50, no. 7, pp. 969 – 978, 2009.

[34] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 1753–1760.

[35] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[36] S. Han, H. Mao, and W. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[37] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[38] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.

[39] G. Huang, S. Liu, L. Van der Maaten, and K. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2752–2761.

[40] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K. T. Cheng, and J. Sun, "Metapruning: Meta learning for automatic neural network channel pruning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 3296–3305.

[41] A. Noy, N. Nayman, T. Ridnik, N. Zamir, S. Doveh, I. Friedman, R. Giryes, and L. Zelnik, "ASAP: Architecture search, anneal and prune," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 493–503.

[42] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "APQ: Joint search for network architecture, pruning and quantization policy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2078–2087.

[43] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[44] X. Li, Y. Zhou, Z. Pan, and J. Feng, "Partial order pruning: for best speed/accuracy trade-off in neural architecture search," in *Proceedings of the IEEE Conference on computer vision and pattern recognition*, 2019, pp. 9145–9153.

[45] X. Dong and Y. Yang, "Network pruning via transformable architecture search," in *Advances in Neural Information Processing Systems*, 2019, pp. 760–771.

[46] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[47] E. Malach, G. Yehudai, S. Shalev-Shwartz, and O. Shamir, "Proving the lottery ticket hypothesis: Pruning is all you need," *arXiv preprint arXiv:2002.00585*, 2020.

[48] M. Paganini and J. Forde, "On iterative neural network pruning, reinitialization, and the similarity of masks," *arXiv preprint arXiv:2001.05050*, 2020.

[49] H. You, C. Li, P. Xu, Y. Fu, Y. Wang, X. Chen, R. Baraniuk, Z. Wang, and Y. Lin, "Drawing early-bird tickets: Towards more efficient training of deep networks," *arXiv preprint arXiv:1909.11957*, 2019.

[50] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[51] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. of the International Conference on Learning Representations (ICLR)*, 2016.

[52] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," *arXiv preprint arXiv:1711.05852*, 2017.

[53] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[54] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, 1989, p. 379–384.

[55] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[56] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, and N. Duffy, "Evolving deep neural networks," in *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 2019, pp. 293–312.

[57] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.

[58] B. C. Moore, *An introduction to the psychology of hearing*. Brill, 2012.

[59] J. Lee, H.-S. Choi, C.-B. Jeon, J. Koo, and K. Lee, "Adversarially trained end-to-end korean singing voice synthesis system," *arXiv preprint arXiv:1908.01919*, 2019.

[60] A. Narayanan and D. L. Wang, "Ideal ratio mask estimation using deep neural networks for robust speech recognition," in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2013.

[61] D. S. Williamson, Y. Wang, and D. Wang, "Complex ratio masking for monaural speech separation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 3, pp. 483–492, March 2016.

[62] Y. Luo and N. Mesgarani, "Conv-TasNet: Surpassing ideal time–frequency magnitude masking for speech separation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 8, pp. 1256–1266, 2019.

[63] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[64] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.

[65] H. Erdogan, J. R. Hershey, S. Watanabe, and J. Le Roux, "Phase-sensitive and recognition-boosted speech separation using deep recurrent neural networks," in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2015.

[66] F. Weninger, H. Erdogan, S. Watanabe, E. Vincent, J. Le Roux, J. R. Hershey, and B. Schuller, "Speech Enhancement with LSTM Recurrent Neural Networks and its Application to Noise-Robust ASR," in *Proc. of the International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, Aug. 2015.

[67] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural computation*, vol. 2, no. 4, pp. 490–501, 1990.

[68] I. Sutskever, *Training recurrent neural networks*. University of Toronto Toronto, Canada, 2013.

[69] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[70] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8110–8119.

[71] L. Chen, S. Dai, C. Tao, H. Zhang, Z. Gan, D. Shen, Y. Zhang, G. Wang, R. Zhang, and L. Carin, "Adversarial text generation via feature-mover's distance," in *Advances in Neural Information Processing Systems*, 2018, pp. 4666–4677.

[72] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 5767–5777.

[73] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[74] S.-W. Fu, C.-F. Liao, Y. Tsao, and S.-D. Lin, "Metricgan: Generative adversarial networks based black-box metric scores optimization for speech enhancement," *arXiv preprint arXiv:1905.04874*, 2019.

[75] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. of the International Conference on Machine Learning (ICML)*, 2017, pp. 214–223.

[76] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 4107–4115.

[77] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, Mar 1982.

[78] K. Cho, B. V. Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine

Translation," in *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[79] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, and V. Zue, "TIMIT acoustic-phonetic continuous speech corpus," *Linguistic Data Consortium, Philadelphia*, 1993.

[80] Z. Duan, G. J. Mysore, and P. Smaragdis, "Online PLCA for real-time semi-supervised source separation," in *Proc. of the International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, 2012, pp. 34–41.

[81] E. Vincent, R. Gribonval, and C. Févotte, "Performance measurement in blind audio source separation," *IEEE transactions on audio, speech, and language processing*, vol. 14, no. 4, pp. 1462–1469, 2006.

[82] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen, "A short-time objective intelligibility measure for time-frequency weighted noisy speech," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*. IEEE, 2010, pp. 4214–4217.

[83] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 15–24.

[84] H. Yang, M. Fritzsche, C. Bartz, and C. Meinel, "Bmxnet: An open-source binary neural network implementation based on mxnet," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1209–1212.

[85] J. Zhang, Y. Pan, T. Yao, H. Zhao, and T. Mei, "dabnn: A super fast inference framework for binary neural networks on arm devices," in *Proceedings of the 27th ACM international conference on multimedia*, 2019, pp. 2272–2275.

[86] Z. Li, H. Ning, L. Cao, T. Zhang, Y. Gong, and T. S. Huang, "Learning to search efficiently in high dimensions," in *Advances in Neural Information Processing Systems*, 2011, pp. 1710–1718.

[87] X. Liu, C. Deng, Y. Mu, and Z. Li, "Boosting complementary hash tables for fast nearest neighbor search," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[88] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu, "Complementary hashing for approximate nearest neighbor search," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 1631–1638.

[89] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 14. MIT Press, 2002.

[90] M. Balasubramanian, E. L. Schwartz, J. B. Tenenbaum, V. de Silva, and J. C. Langford, "The Isomap algorithm and topological stability," *Science*, vol. 295, January 2002.

[91] S. T. Roweis and L. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, pp. 2323–2326, 2000.

[92] R. Salakhutdinov and G. Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, vol. 50, no. 7, pp. 969–978, 2009.

[93] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in neural information processing systems*, 2009, pp. 1753–1760.

[94] R. R. Salakhutdinov and G. E. Hinton, "Semantic hashing," in *In SIGIR workshop on Information Retrieval and applications of Graphical Models*, 2007.

[95] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-taught hashing for fast similarity search," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, 2010, pp. 18–25.

[96] H. Liu, R. Wang, S. Shan, and X. Chen, "Deep supervised hashing for fast image retrieval," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2064–2072.

[97] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *icml*, vol. 96. Citeseer, 1996, pp. 148–156.

[98] J.-P. Vert, K. Tsuda, and B. Schölkopf, "A primer on kernel methods," *Kernel methods in computational biology*, vol. 47, pp. 35–70, 2004.

[99] Q. Chang, Q. Chen, and X. Wang, "Scaling gaussian rbf kernel width to improve svm classification," in *2005 International Conference on Neural Networks and Brain*, vol. 1. IEEE, 2005, pp. 19–22.

[100] Z. Duan, G. J. Mysore, and P. Smaragdis, "Online PLCA for real-time semi-supervised source separation," in *International Conference on Latent Variable Analysis and Signal Separation*. Springer, 2012, pp. 34–41.

[101] J. Thiemann, N. Ito, and E. Vincent, "The diverse environments multi-channel acoustic noise database (demand): A database of multichannel environmental noise recordings," in *Proceedings of Meetings on Acoustics ICA2013*, vol. 19, no. 1. ASA, 2013, p. 035081.

[102] J. Le Roux, S. Wisdom, H. Erdogan, and J. R. Hershey, "Sdr–half-baked or well done?" in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 626–630.

[103] 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 *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*, vol. 2. IEEE, 2001, pp. 749–752.

[104] J. Jensen and C. H. Taal, "An algorithm for predicting the intelligibility of speech masked by modulated noise maskers," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 11, pp. 2009–2022, 2016.

[105] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[106] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. of the International Conference on Machine Learning (ICML)*, 2015.

[107] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[108] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. of the International Conference on Learning Representations (ICLR)*, 2015.

[109] Y. Luo, Z. Chen, and T. Yoshioka, "Dual-path RNN: efficient long sequence modeling for time-domain single-channel speech separation," in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2020.

[110] N. Zeghidour and D. Grangier, "Wavesplit: End-to-End Speech Separation by Speaker Clustering," *arXiv preprint arXiv:2002.08933*, 2020.

[111] J. Chen, Q. Mao, and D. Liu, "Dual-path transformer network: Direct context-aware modeling for end-to-end monaural speech separation," in *Proc. of the Annual Conference of the International Speech Communication Association (Interspeech)*, 2020.

[112] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2015, pp. 5206–5210.

[113] D. Snyder, G. Chen, and D. Povey, "MUSAN: A Music, Speech, and Noise Corpus," *arXiv preprint arXiv:1510.08484*, 2015.

[114] J. Deng, Z. Zhang, F. Eyben, and B. Schuller, "Autoencoder-based unsupervised domain adaptation for speech emotion recognition," *IEEE Signal Processing Letters*, vol. 21, no. 9, pp. 1068–1072, 2014.

[115] S. Sun, B. Zhang, L. Xie, and Y. Zhang, "An unsupervised deep domain adaptation approach for robust speech recognition," *Neurocomputing*, vol. 257, pp. 79–87, 2017.

[116] A. Sivaraman and M. Kim, "Self-Supervised Learning for Personalized Speech Enhancement," *arXiv preprint arXiv:2011.03426*, 2021.

[117] W. Wang, V. W. Zheng, H. Yu, and C. Miao, "A survey of zero-shot learning: Settings, methods, and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–37, 2019.

[118] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, "Zero-Shot Learning – A Comprehensive Evaluation of the Good, the Bad and the Ugly," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 9, pp. 2251–2265, 2018.

[119] H. Larochelle, D. Erhan, and Y. Bengio, "Zero-data learning of new tasks," in *Proc. of the AAAI National Conference on Artificial Intelligence (AAAI)*, vol. 1, no. 2, 2008, p. 3.

[120] M. Palatucci, D. Pomerleau, G. Hinton, and T. M. Mitchell, "Zero-shot learning with semantic output codes," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[121] Y. N. Dauphin, G. Tur, D. Hakkani-Tür, and L. Heck, "Zero-shot learning for semantic utterance classification," *Proc. of the International Conference on Learning Representations (ICLR)*, 2013.

[122] J. Choi, J. Lee, J. Park, and J. Nam, "Zero-shot learning for audio-based music classification and tagging," *Proc. of the International Society for Music Information Retrieval Conference (ISMIR)*, 2019.

[123] H. Chen, Y. Wang, C. Xu, Z. Yang, C. Liu, B. Shi, C. Xu, C. Xu, and Q. Tian, "Data-free learning of student networks," in *Proc. of the International Conference on Computer Vision (ICCV)*, 2019, pp. 3514–3522.

[124] P. Micaelli and A. Storkey, "Zero-shot knowledge transfer via adversarial belief matching," in *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[125] J. Ye, Y. Ji, X. Wang, X. Gao, and M. Song, "Data-free knowledge amalgamation via group-stack dual-GAN," in *Proc. of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 12 516–12 525.

[126] R. G. Lopes, S. Fenu, and T. Starner, "Data-free knowledge distillation for deep neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[127] G. K. Nayak, K. R. Mopuri, V. Shaj, R. V. Babu, and A. Chakraborty, "Zero-shot knowledge distillation in deep networks," in *Proc. of the International Conference on Machine Learning (ICML)*, 2019, pp. 4743–4751.

[128] G. K. Nayak, K. R. Mopuri, and A. Chakraborty, "Effectiveness of arbitrary transfer sets for data-free knowledge distillation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 1430–1438.

[129] A. Chawla, H. Yin, P. Molchanov, and J. Alvarez, "Data-free knowledge distillation for object detection," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 3289–3298.

[130] X. Hao, S. Wen, X. Su, Y. Liu, G. Gao, and X. Li, "Sub-band knowledge distillation framework for speech enhancement," in *Proc. of the Annual Conference of the International Speech Communication Association (Interspeech)*, 2020.

[131] S. Nakaoka, L. Li, S. Inoue, and S. Makino, "Teacher-student learning for low-latency online speech enhancement using wave-u-net," in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 2021, pp. 661–665.

[132] Y.-H. Tu, J. Du, and C.-H. Lee, "Speech enhancement based on teacher–student deep learning using improved speech presence probability for noise-robust speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 12, pp. 2080–2091, 2019.

[133] A. S. Subramanian, S.-J. Chen, and S. Watanabe, "Student-teacher learning for blstm mask-based speech enhancement," *arXiv preprint arXiv:1803.10013*, 2018.

[134] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised Visual Representation Learning by Context Prediction," in *Proc. of the International Conference on Computer Vision (ICCV)*, 2015.

[135] S. Watanabe, T. Hori, J. L. Roux, and J. R. Hershey, "Student-Teacher Network Learning with Enhanced Features," in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 2017, pp. 5275–5279.

[136] V. Manohar, P. Ghahremani, D. Povey, and S. Khudanpur, "A teacher-student learning approach for unsupervised domain adaptation of sequence-trained asr models," in *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2018, pp. 250–257.

[137] Z. Zhang, Y. Song, J. Zhang, I. V. McLoughlin, and L. Dai, "Semi-supervised end-to-end asr via teacher-student learning with conditional posterior distribution." in *Proc. of the Annual Conference of the International Speech Communication Association (Interspeech)*, 2020, pp. 3580–3584.

[138] Y. Xu, J. Du, L.-R. Dai, and C.-H. Lee, "An Experimental Study on Speech Enhancement Based on Deep Neural Networks," *IEEE Signal Processing Letters*, vol. 21, no. 1, pp. 65–68, 2014.

[139] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, Dec. 2010.

[140] O. Chapelle, B. Schölkopf, and A.Zien, *Semi-Supervised Learning*. MIT Press, 2006.

[141] K. J. Piczak, "Esc: Dataset for environmental sound classification," in *Proceedings of the 23rd ACM international conference on Multimedia*, 2015, pp. 1015–1018.

[142] M. Jeub, M. Schafer, and P. Vary, "A binaural room impulse response database for the evaluation of dereverberation algorithms," in *2009 16th International Conference on Digital Signal Processing*. IEEE, 2009, pp. 1–5.

[143] J. Merimaa, T. Peltonen, and T. Lokki, "Concert hall impulse responses pori, finland: Reference," *Tech. Rep*, 2005.

[144] S. Nakamura, K. Hiyane, F. Asano, T. Nishiura, and T. Yamada, "Acoustical sound database in real environments for sound scene understanding and hands-free speech recognition," in *International Conference on Language Resources and Evaluation*, 2000, pp. 965–968.

[145] I. Szöke, M. Skácel, L. Mošner, J. Paliesek, and J. Černockỳ, "Building and evaluation of a real room impulse response dataset," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 4, pp. 863–876, 2019.

[146] K. Kinoshita, M. Delcroix, T. Yoshioka, T. Nakatani, E. Habets, R. Haeb-Umbach, V. Leutnant, A. Sehr, W. Kellermann, R. Maas, and et. al, "The reverb challenge: A common evaluation framework for dereverberation and recognition of reverberant speech," in *Proc. of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, 2013, pp. 1–4.

[147] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[148] D. Wang, X. Liu, and S. C. Ahalt, "On temporal generalization of simple recurrent networks," *Neural Networks*, vol. 9, no. 7, pp. 1099–1118, 1996.

[149] T. Hayashi, S. Watanabe, T. Toda, T. Hori, J. L. Roux, and K. Takeda, "Duration-controlled lstm for polyphonic sound event detection," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 11, pp. 2059–2070, 2017.

[150] Z. Chen, Y. Luo, and N. Mesgarani, "Deep attractor network for single-microphone speaker separation," in *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, 2017, pp. 246–250.

[151] D. Williamson, W. Y, and D. Wang, "Complex ratio masking for monaural speech separation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 3, pp. 483–492, 2015.

[152] J. Le Roux, S. Wisdom, H. Erdogan, and J. R. Hershey, "SDR – half-baked or well done?" in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2019.

[153] Y. Luo, Z. Chen, and T. Yoshioka, "Dual-path rnn: efficient long sequence modeling for time-domain single-channel speech separation," in *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 2020, pp. 46–50.

[154] M. Pariente, S. Cornell, J. Cosentino, S. Sivasankaran, E. Tzinis, J. Heitkaemper, M. Olvera, F.-R. Stöter, M. Hu, J. M. Martín-Doñas, D. Ditter, A. Frank, A. Deleforge, and E. Vincent, "Asteroid: The PyTorch-Based Audio Source Separation Toolkit for Researchers," in *Proc. of the Annual Conference of the International Speech Communication Association (Interspeech)*, 2020.

[155] R. M. French, "Catastrophic forgetting in connectionist networks," *TRENDS in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

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

[157] B. G. Shinn-Cunningham, N. Kopco, , and T. J. Martin, "Localizing nearby sound sources in a classroom: Binaural room impulse responses," *Journal of the Acoustical Society of America*, vol. 117, no. 5, pp. 3100–3115, 2005.

[158] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in neural information processing systems*, 2017, pp. 5767–5777.

[159] T. Schmiedlechner, I. Z. Yong, A. Al-Dujaili, E. Hemberg, and U.-M. O'Reilly, "Lipizzaner: a system that scales robust generative adversarial network training," *arXiv preprint arXiv:1811.12843*, 2018.

[160] U. Garciarena, R. Santana, and A. Mendiburu, "Evolved gans for generating pareto set approximations," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, pp. 434–441.

[161] V. Costa, N. Lourenço, and P. Machado, "Coevolution of generative adversarial networks," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 2019, pp. 473–487.

[162] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[163] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are gans created equal? a large-scale study," *Advances in neural information processing systems*, vol. 31, 2018.

[164] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[165] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms*. Elsevier, 1991, vol. 1, pp. 69–93.

[166] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[167] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[168] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

Sunwoo **Kim**

700 N. Woodlawn Ave. Luddy Hall, Bloomington, IN, 47404

kimsunw@iu.edu | www.kimsunwoo.com | sunwookimiub | Sunwoo Kim

Summary

PhD candidate in Intelligent Systems Engineering department in Luddy School of Informatics, Computing, and Engineering at Indiana University Bloomington. I investigate efficient machine learning and deep learning solutions for audio processing problems with my advisor Prof. Minje Kim. My current research focuses on novel modes of compression for source separation applications through personalization and scalable learning for practical deployment onto resource-constrained devices.

Positions Held

Indiana University — *Bloomington, IN*

Research Assistant — *Dec 2018 - Present*

- Funded by NSF project "A Portable and Intelligent Testing System for Power-efficient and Accurate Foodborne Pathogen Detection"
- Researching efficient machine learning and deep learning models through various modes of model compression

Amazon Lab126 — *Sunnyvale, CA (Remote)*

Applied Scientist Intern — *Jun. 2021 - Sep 2021*

- Group: Audio Technology
- Supervisors: Mrudula Athi, Guangji Shi, Trausti Kristjansson
- Project: Personalized speech dereverberation

Amazon Lab126 — *Sunnyvale, CA (Remote)*

Applied Scientist Intern — *May. 2020 - Aug 2020*

- Group: Audio Technology
- Supervisors: Yuzhou Liu, Krishna Kamath, Trausti Kristjansson
- Project: Model compression for speech enhancement

Qualcomm — *San Diego, CA*

Interim Engineering Intern — *May. 2019 - Aug 2019*

- Group: Audio Algorithms
- Supervisors: Shuhua Zhang, Laehoon Kim
- Project: Non-linear echo cancellation

Indiana University — *Bloomington, IN*

Teaching Assistant — *Aug. 2016 - Dec 2018*

- Deep Learning Systems [Spring 2018]
- Machine Learning for Signal Processing [Fall 2017, Fall 2018]
- Software Engineering [Fall 2016, Spring 2017]

National Center for Supercomputing Applications — *Urbana, IL*

CyberGIS SPIN Intern — *May. 2015 - May 2016*

- Parallel Terrain Analysis and predictive ecosystem mapping

Publications

BLOOM-Net: Blockwise Optimization for Masking Networks Toward Scalable and Efficient Speech Enhancement — *Singapore*

Sunwoo Kim, Minje Kim — *May 22-27, 2022*

International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Test-Time Adaptation Toward Personalized Speech Enhancement: Zero-Shot Learning with Knowledge Distillation — *New Paltz, NY (Remote)*

Sunwoo Kim, Minje Kim — *Oct 17-20, 2021*

IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)

Personalized Speech Enhancement through Self-Supervised Data Augmentation and Purification — *Brno, Czech Republic (Remote)*

Aswin Sivaraman, **Sunwoo Kim**, Minje Kim — *Aug 30 - Sep 3, 2021*

Interspeech

Boosted Locality Sensitive Hashing: Discriminative Binary Codes For Source Separation — *Barcelona, Spain (Remote)*

Sunwoo Kim, Haici Yang, Minje Kim (Nominated for the Best Student Paper Award) — *May 4-8, 2020*

International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Incremental Binarization On Recurrent Neural Networks For Single-Channel Source Separation — *Brighton, UK*

Sunwoo Kim, Mrinmoy Maity, Minje Kim — *May 12-17, 2019*

International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Education

Indiana University — *Bloomington, IN*

Ph.D. Candidate in Intelligent Systems Engineering — *Aug 2016 - May 2022 (expected)*

Advised by professor **Minje Kim**

University of Illinois at Urbana-Champaign — *Urbana, IL*

B.S. in Physics — *May 2016*

Academic Services

IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Conference Reviewer — *2020, 2021, 2022*

ACM SIGGRAPH

Reviewer — *2022*

European Signal Processing Conference (EUSIPCO)

Conference Reviewer — *2022*

IEEE Transactions on Audio, Speech and Language Proceedings (TASLP)

Journal Reviewer — *2021*

European Association for Signal Processing (EURASIP) Journal on Audio, Speech, and Music Processing

Journal Reviewer — *2019, 2020, 2021*