UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Comparison and Development of Practical Voice Conversion Models

Matthew Baas
20786379

Report submitted in partial fulfilment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at Stellenbosch University.

Supervisor: Dr H. Kamper

November 2020

Acknowledgements

This work would not have been possible without the compute resources provided by Ackerview, and Torsten Babl's help in procuring such resources – a massive thanks is extended to him. I would also like to thank Dr Herman Kamper for his amazing role as a supervisor and as the provider of several LaTeX report templates. Finally, I would also like to extend a thanks to those who help develop some of the libraries used in this project – namely those who help contribute to `librosa`, `pytorch`, and Nvidia's CUDA drivers. This work is only possible due to those who helped build up the software and hardware foundations necessary. A further thank you to all my friends and colleagues who gave valuable input and sensible suggestions to problems I faced along the way. And, above all, the love and support of my family was and is pivotal to my efforts.

Abstract

English

Voice conversion is the task of converting an utterance spoken by a source speaker so that it appears to be said by a different target speaker, while retaining the linguistic content of the utterance. However, current voice conversion systems fail to be practically useful in many scenarios due to their failure to satisfy all requirements necessary for practical use. Namely, they need to (i) be trainable without access to parallel data, (ii) work in a zero-shot setting where both the source and target speakers are unseen during training, (iii) retain quality when trained on little data and (iv) run in real time or faster.

Recent techniques fulfil some of these requirements, but not all. This report aims to profile recent voice conversion systems and extend recent voice conversion models to satisfy all four requirements. We specifically propose the StarGAN-ZSVC model, which extends recent generative adversarial network (GAN) based models to work in the zero-shot setting by conditioning it on a speaker embedding. We compare StarGAN-ZSVC against other models in a low-resource setting, performing both subjective and objective comparisons. Our comparisons confirm that no existing model satisfies all aforementioned conditions, while the proposed StarGAN-ZSVC does.

Afrikaans

Stemomskakeling behels die omskepping van 'n spraaksein wat deur 'n bronspreker geproduseer word sodat dit dit voorkom asof dit deur 'n ander teikenspreker gesê word, terwyl die taalinhoud van die oorspronklike sein behou word. Meeste bestaande stemomskakelingstelsels is nie gepas vir praktiese gebruik nie, omdat dit nie voldoen aan vier spesifieke vereistes nie. 'n Praktiese stelsel moet (i) afgerig kan word sonder parallelle data, (ii) gebruik kan word in 'n nulskootomgewing waar nie die bron- of teikenspreker gesien word gedurende afrigting nie, (iii) hoë kwaliteit behou wanneer dit op min data afgerig word, en (iv) intyds of vinniger kan hardloop.

Onlangse tegnieke voldoen aan sommige van hierdie vereistes, maar nie almal nie. Hierdie verslag het ten doel om die onlangse stemomskakelingstelsels te ondersoek en uit te brei om aan al vier vereistes te voldoen. Ons stel spesifiek die StarGAN-ZSVC model bekend, wat modelle gebasseer of generatiewe adversêre netwerke (GANe) uitbrei om in die nulskootopstelling te werk. Ons vergelyk StarGAN-ZSVC met ander stemomskakelingstegnieke in 'n laehulpbronomgewing. Ons eksperimente bevestig dat geen bestaande model aan al die bogenoemde vereistes voldoen nie, terwyl die voorgestelde StarGAN-ZSVC wel doen.

Contents

List of Figures

List of Tables

Nomenclature

Variables and functions

$\mathbf{s}$	A vector representing information uniquely associated with some speaker. $\mathbf{s}_{\text{src}}$ refers to a vector embedding of information associated with speaker `src`.
d	The scalar dimensionality of a speaker embedding vector $\mathbf{s}$.
$\|\|\Delta\mathbf{s}\|\|$	The Euclidean norm of the vector difference between two speaker embeddings, typically between a source and target speaker embedding $\|\|\mathbf{s}_{\text{src}} - \mathbf{s}_{\text{trg}}\|\|_2$.
m	The number of Mel-scale frequency bins in a Mel-scale spectrogram.
T	The total length of a time-domain speech utterance.
N	The total length of a discrete frequency-domain speech utterance, or more specifically the number of frames in a Mel-spectrogram.
$\mathcal{X}$	A time-domain speech utterance encoded as a scalar sequence of amplitudes. $\mathcal{X}_{\text{a}}$ refers to a speech utterance by speaker `a`.
X	A frequency-domain speech utterance, encoded as a sequence of Mel-scale spectrogram frames, each of dimension m. X_{a} refers to a speech utterance by a speaker `a`.
$\mathbf{x}_i$	A vector frame of a Mel-spectrogram X. I.e. $X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$.
$\mathcal{L}$	A scalar loss function computed using a neural network's outputs and possibly other data.
θ	An unspecified tuple of parameters representing the weights of a neural network.
α	The learning rate of a gradient-based optimizer.
$\mathbf{h}_i$	The hidden state vector of an RNN (LSTM or GRU) at sequence index i.
h	The scalar dimensionality of hidden state vectors $\mathbf{h}$.

Acronyms and abbreviations

DBLSTM	Deep bidirectional long short-term memory
DFT	Discrete Fourier transform
DNN	Deep neural network
DTW	Dynamic time warping
GAN	Generative adversarial network
GE2E	Generalized end-to-end (architecture)
GLU	Gated linear unit
GMM	Gaussian mixture model
GRU	Gated recurrent unit
GSP	Global sum pooling
HMM	hidden Markov model
LSTM	Long short-term memory
Mel-spectrogram	log Mel-scale short-time Fourier transform
MFCC	Mel-frequency cepstral coefficient
MOS	Mean opinion score
MSE	Mean square error
RNN	Recurrent neural network
SELU	Scaled exponential linear unit
StarGAN-VC	StarGAN - voice conversion
StarGAN-ZSVC	StarGAN - zero-shot voice conversion
STFT	Short-time Fourier transform
VC	Voice conversion

Chapter 1

Introduction

Voice conversion (VC) is a speech processing task where speech by a source speaker is transformed into speech that appears to be spoken by a desired target speaker while preserving linguistic content (which words are spoken). VC is a subfield of speech processing and has seen significant advances in recent years, in part due to the increasing research focus brought about by several VC competitions [1, 2]. Some techniques are beginning to achieve near human-level quality in conversion outputs. This progress builds off the advances in speech processing in the last decade with the rise of large deep neural networks (DNNs) [3] together with more recent techniques introduced in image-to-image translation models making use of generative adversarial networks (GANs) [4].

However, the majority of such progress has been confined to fairly academic settings which have limited usefulness in practice due to their failure to satisfy several requirements of practical VC systems, particularly in low-resource settings where there is limited training data. The work of this report attempts to address this shortfall by first formalizing the requirements for what it means for a VC system to be 'practical'. Then the report goes on to characterize several recent VC systems in terms of their performance and which requirements they fulfill. Upon finding that no systems fulfill all requirements, we go on to develop a new VC technique for real-time zero-shot voice conversion in low-resource settings and then perform a series of experiments to compare the performance of our proposed model to existing VC systems.

1.1. Motivation

A fast, human-level VC system has important applications in several industries. First, it can be utilized for the preservation of privacy and identity protection [5], or for voice mimicry and disguise [6]. Voice mimicry and alteration is most useful in the entertainment and gaming industry, where a company could use a single actor to record several roles. For example, a game with 100 non-player characters that have similar lines would need to hire 100 different voice actors to record each line, whereas if a practical VC system is available they could instead hire only a handful and use VC to generate unique dialog for the remaining characters, saving significant effort and expense.

VC can also be essential in addressing downstream natural language and speech processing problems in low-resource contexts where training data is limited: VC could be used to augment training data by converting the available utterances to novel speakers – effectively increasing the diversity of training data and improving the quality of the resulting systems. This entails the promise of increasing the performance and availability of downstream tasks like automatic speech recognition and text-to-speech for speakers of under-served and low-resource languages, which are acutely prevalent in South Africa.

Thus the motivation for this report is to provide a step toward realizing some of these potential applications of a practical VC system. In particular, this work aims to bridge the gap between the recent high performing techniques for VC and their proposed applications

by attempting to profile and overcome some of the key limitations that prevent their practical use.

1.2. Problem statement and objectives

This work aims to produce a *practical VC system*. Such a goal is under-specified and is thus expanded into two separate problems. The first of which is to define what it means for a VC system to be *practical*. Based on the proposed use cases of VC, we can derive four requirements that a practical VC system must satisfy: a practical VC system should

- be trainable without access to parallel data: the VC system should not need paired source and target utterances with the same linguistic content during training (expanded in Section 2.1). Parallel data is difficult to collect in general, and even more so for low-resource languages.
- work in zero-shot settings: the VC system should be able to perform VC for source and target speakers unseen during training. Without this requirement, a system would need to be retrained whenever speech from a new speaker is desired.
- retain reasonable quality output in low-resource settings: when trained on little data ($<$ 30 minutes of audio), the VC system should maintain its output quality.
- run at least in real-time: the system needs to be able to convert one second of audio in less than one second. For data augmentation in particular, having the system run as fast as possible is essential for it to be practical in the training of a downstream speech model.

The second problem is to design, develop, and test such a practical VC system. This report proposes such a practical VC system and confirms its effectiveness and practicality by comparing it to existing VC methods.

1.3. Scope and contributions

This report accomplishes two related tasks. First, it characterizes several VC techniques in terms of their performance and practicality (how well they satisfy the practical requirements defined in Section 1.2). Second, after realizing that no existing VC systems satisfy all the requirements, we develop a new VC technique by combining GAN methods from the StarGAN-VC & StarGAN-VC2 [7, 8] architectures with a speaker embedding technique introduced with the recent AutoVC model [9]. This new model is capable of performing zero-shot VC and is thus termed StarGAN-ZSVC.

StarGAN-ZSVC achieves zero-shot prediction by using a separate neural network to generate speaker embeddings for potentially unseen speakers; these embeddings are then used to condition the model at inference time. We go on to compare the VC techniques profiled to one another and to StarGAN-ZSVC, showing that our new method performs better than simple baseline models and similar or better than the recent AutoVC zero-shot voice conversion approach across a range of objective and subjective evaluation metrics. More specifically, we observe that it satisfies all the practicality objectives of Section 1.2 and gives similar or better performance in all zero-shot settings considered, and does so roughly 6 times faster than AutoVC. The proposed StarGAN-ZSVC system, as set out in this report, has also been described in a more condensed paper titled *StarGAN-ZSVC: Towards zero-shot voice conversion in low-resource contexts*. This paper has been accepted

for publication and presentation at the *South African Conference for Artificial Intelligence Research 2020* [10].

To truly assess the practicality of a VC system, however, would actually require us to perform the downstream activities mentioned in Section 1.1 and check whether the performance has improved or if the task is now possible. For example, one might train a low-resource automatic speech recognition (ASR) model with and without using StarGAN-ZSVC for data augmentation and compare its performance. Since this would drastically expand the scope of the project, this is not performed and we limit ourselves to developing and assessing VC systems in isolation from their downstream applications.

1.4. Ethical considerations

Developing a practical real-time VC system has the potential to assist with both positive and negative social actions. For example, the positive applications of VC for voice mimicry and disguise [6] could potentially also be misused to aid in identity fraud. Since a practical real-time VC system has not existed before, there is little research into the positive and negative ethical and social considerations associated with practical use of VC in society. We will assess and develop VC systems in isolation in the well established field of voice conversion, however we also note that additional research is required to assess ethical and social considerations enabled by downstream applications of VC systems.

1.5. Report overview

This report is structured as follows. In the next chapter, a literature review is conducted to introduce the relevant topics and notation when dealing with VC systems. This chapter continues by providing a description of domain techniques used with VC systems and various classes of VC models.

Then, Chapter 3 gives a detailed description of several traditional and recent VC systems and their capabilities. The chapter continues on with a motivation, description, and detailed design of our new proposed model StarGAN-ZSVC. Chapter 4 proceeds to discuss and define precisely how each model is prepared, trained, and compared to one another, together with details about the input dataset and preprocessing performed.

Finally, Chapter 5 presents and discusses the results of several objective and subjective evaluations performed on the set of models discussed in Chapter 3 to assess the performance of the considered models and confirm that the proposed StarGAN-ZSVC satisfies all practicality requirements defined in Section 1.2. The report is concluded in Chapter 6, providing a summary of results, conclusions, and possible avenues for future work.

Chapter 2

Literature Review

Before we can investigate prominent VC systems, we need to outline the background theory necessary to reason about VC and the techniques typical VC systems employ. This chapter begins with definitions of specific terms used in voice conversion, and continues on to outline the general form of a VC system. Then, a summary of various pieces of domain knowledge important for VC systems are given in Section 2.3. Finally, we outline the general classes of VC systems in terms of the methods they use in Section 2.4. The information in this chapter is laid out so that by Chapter 3, the idea behind each model should appear imminently clear to the reader.

2.1. Terminology

Within the VC literature one frequently encounters terms such as 'source speaker', 'one-to-one model', and 'parallel/non-parallel' data. These terms have evolved from the VC literature, with some only being introduced recently as model capabilities have improved. To ensure that information presented in the rest of this report is based on a solid foundation, we concretely define several of these terms based on their usage in the literature [11]:

- **Utterance**: the unit of data for a VC system referring to a single recording of a person speaking intelligible speech into a microphone. In practice these recordings are between 2-7 seconds and are encoded in digital format (e.g as a `.wav` file). In referring to an utterance, one may either be referring to the time-domain waveform or some processed representation of the waveform (such as a spectrogram), depending on context.
- **Source/target speaker**: in VC, the source speaker is the identity of the person speaking the source utterance, while the target speaker is the *desired* identity for the VC target. I.e. the VC system should aim to transform an utterance spoken by a source speaker into one recognized as being spoken by a desired target speaker.
- **Parallel and non-parallel data**: parallel data refers to *pairs of source-target utterances*, where the model designer has access to an utterance from both the source and target speaker containing the same linguistic content (spoken words). Non-parallel data is the converse, where each data item is an *individual utterance* by a source speaker with no such paired utterance by a target speaker.
- **One-to-one and many-to-many**: a VC model is a *one-to-one* model if it can only convert from speaker A to speaker B, while a VC model is *many-to-many* if it can convert an utterance spoken by any speaker in a set of source speakers to any speaker in a set of target speakers.
- **Zero-shot and few-shot**: a VC model has *zero-shot capability* if it can perform VC when no utterances from the source or target speaker are seen during training of the model, while a VC model has *few-shot capability* if it can perform VC when only a handful (typically less than 5) utterances from the source or target speaker are seen during training of the model.

2.2. Voice conversion systems

The ultimate goal of a VC system is to convert an input utterance by a source speaker into an output utterance by a target speaker. The particular methods used by different VC systems to achieve this vary widely, and there is very little standard notation within the existing literature. Despite this, nearly all VC systems share several common structural elements [11]. To allow us to easily discuss and compare between different VC models, we need to provide a consistent framework to view these system elements and define a standard notation. Concretely, most VC systems encountered can be generalized to the system diagram in Figure 2.1

As in Figure 2.1, a VC system consists of several parts. First, a source utterance $\mathcal{X}_{\text{src}}$ is parsed with a feature extraction operation to obtain some meaningful feature representation of the utterance, X_{src} (note the calligraphy form $\mathcal{X}_{\text{src}}$ indicates the waveform which is distinct from the feature representation X_{src}). An utterance $\mathcal{X}$ is defined as a sequence of scalar samples $\mathcal{X} = [x_1, x_2, ..., x_T]$ where each sample $x_i \in \mathbb{R}$ is the i'th amplitude sample of a T-sample waveform. In most VC systems the feature extractor is a kind of frequency domain transformation such that X_{src} contains the frequency information of the time-domain waveform $\mathcal{X}_{\text{src}}$ with $X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$ also a sequence of vectors, with each vector element $\mathbf{x}_i \in \mathbb{R}^m$ as the i'th frame of an N-frame spectrogram.

In parallel, the source and target speaker's identity are passed through some form of speaker information extraction, obtaining some kind of meaningful and numerical encoding of the source speaker, $\mathbf{s}_{\text{src}}$, and target speaker, $\mathbf{s}_{\text{trg}}$. This information can take various forms, but most forms can usually be expressed as some d-dimensional vector, representing for example a one-hot encoding of the source/target speaker index (from a list of known speakers). The extraction process for the source and target speakers need not be the same, but typically are [11].

The representations of the source and target speaker, together with the input utterance features X_{src}, are given to a certain mapping function which performs the actual conversion. It returns a feature representation $X_{\text{src}\to\text{trg}}$ that usually exists in the same domain as X_{src} [12]. Some form of reconstruction is then performed on this converted feature representation to yield the final time-domain output waveform $\mathcal{X}_{\text{src}\to\text{trg}}$.

Since this functional block takes the coding of an utterance $X_{\text{src}\to\text{trg}}$ and converts it to an actual voice waveform, it is known as a voice coder, or **vocoder**. The design and construction of vocoders typically falls outside of the scope of voice conversion and comprises an entire field of study in its own right [13–15]. VC systems typically design the mapping function and speaker information extraction process, and simply choose an

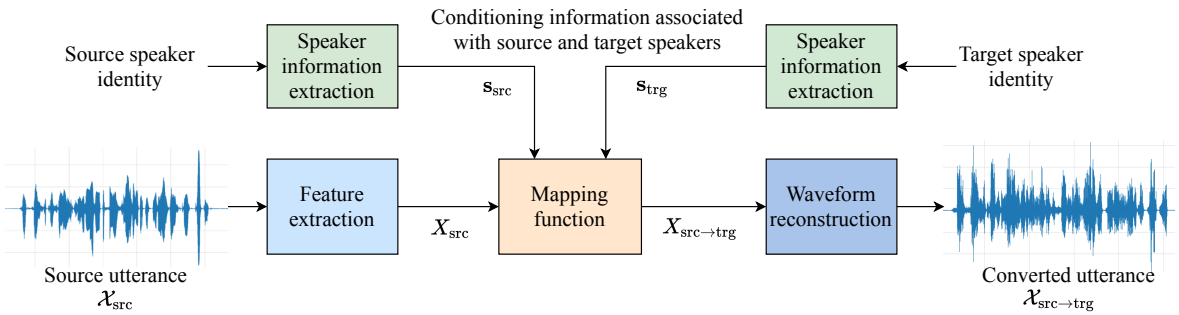


Figure 2.1: Generalized block diagram of a VC system. Individual VC implementations define the form and operation of each functional block.

existing method or system from the literature for feature extraction and vocoding [12].

As an example instance of the block diagram in Figure 2.1, consider a simple one-to-one model that just pitch shifts the input utterance from the (hard-coded) average pitch of the source speaker to the (hard-coded) average pitch of the target speaker's voice. In such a model, the feature extraction and waveform reconstruction are both identity operations. The mapping function is just a simple pitch shift operation which only considers the utterance input and ignores the conditioning speaker information because, being a one-to-one model, the source and target speaker information is already implicitly present in the model. Thus the speaker information extraction blocks are also irrelevant (and can be treated as any arbitrary operation) in such a one-to-one system.

2.3. Relevant methods and techniques

With the basic framework of VC systems established, we will now go on to introduce several techniques which are highly prevalent in the various functional blocks of Figure 2.1.

2.3.1. Speech representations

We start our discussion with the first block – extracting some feature representation X from a time-domain utterance $\mathcal{X}$. Aside from trivial representations, such as using an identity function as in the aforementioned example, feature extraction techniques in the literature make extensive use of the Fourier transform and frequency domain techniques [11].

Other parametric techniques attempt to build representations of speech by modelling the voice tract [16], or by modelling the continuous pitch of the utterance [17]. However, most modern VC systems extract features from the input speech by representing it in the frequency domain using a variant of the discrete Fourier transform (DFT) [12]. In particular, the current state-of-the-art VC models represent speech as Mel-scale spectrograms or Mel-frequency cepstral coefficients together with a pitch contour [1, 12].

Intuitively, the Mel-scale is an alternate non-linear frequency scale that intends to align with the human perception of frequency [18]. When constructing the short-time Fourier transform (STFT) using this scale we obtain a Mel-scale spectrogram. As a practical consideration, VC systems typically use the natural logarithm of the Mel-scale spectrogram magnitudes – often simply called the 'Mel-spectrogram' [12]. The details behind the Mel-spectrogram used in this project, as well as other typical speech representations, is given in Appendix C.

Since the field of speech representations is a well defined research area in its own right, we make the decision to use the *same speech representation* for all models we construct in this project – the Mel-spectrogram. Without this limitation, the scope would need to expand unacceptably to properly cover all well-known speech representation techniques. We make an exception for StarGAN-VC2 however, because it uses and is specifically designed to be used with the WORLD vocoder (the workings of which are beyond the scope of this project).

2.3.2. Deep neural networks

Now we proceed to the techniques used in the mapping function and vocoding blocks of Figure 2.1. Although non-statistical methods exist, such as a simple pitch shift for the mapping function, or the WORLD vocoder for the waveform reconstruction [19],

recent high-performing systems increasingly use deep neural networks (DNNs) for both the mapping function and vocoder [12].

As a basic formulation of neural networks, a DNN typically refers to a function $F(A;\theta)$ that operates on some defined input A (anything from a tuple to a tensor). The function is parametrized by a set of variables θ, which typically need to be optimized in some way to make the function / network output useful results. The functions can usually be expressed as a sequence of linear matrix operations paired with non-linear element-wise operations and reshaping operations. Many forms of such functions can actually be shown to, under certain conditions, approximate any arbitrary function with the Universal Approximation Theorem [20].

The process of optimizing these parameters θ is known as training the network. There exist several ways to train networks, but most VC systems are trained in a supervised fashion because the speaker identity of each utterance is typically known (although a target converted utterance is not present for non-parallel settings). To train the network, a scalar *loss* $\mathcal{L}$ is defined to be some function of the network output and any other variables or external data. The only requirement is that $\mathcal{L}$ should be a scalar, and the gradient with respect to each of the parameters θ should be well defined and smooth everywhere. The loss is defined so that a *lower loss* corresponds to a better performing network – i.e. to optimize the network, the loss should be minimized.

So long these conditions hold, the following method is iteratively repeated to train the network until the weights converge:

- A set of data items are sampled from the dataset. Again, the type of the data is left unspecified and can take on any form.
- The loss $\mathcal{L}$ is computed using the data items and functions from the DNN architecture.
- The gradient of this loss is taken with respect to the weights, computing $\nabla_\theta \mathcal{L}$.
- The network weights are then updated to lower the loss with the update equation: $\theta_{t+1} = \theta_t - \alpha \frac{\tilde{m}_1}{\sqrt{\tilde{m}_2}+\epsilon}$ where $\tilde{m}_1$ is an exponentially weighted running average of $\nabla_\theta \mathcal{L}$ and $\tilde{m}_2$ is an exponentially weighted running average of the square of each gradient $\nabla_\theta \mathcal{L}$. A small constant ϵ is added for numerical stability [21].

The update equation given in the final point is actually very general, and can cover a wide range of different known optimizers by setting the learning rate α, constant ϵ, and running average update coefficients. For example, setting the square gradient term $\tilde{m}_2$ to never update and remain zero, $\epsilon = 1$, and the gradient term $\tilde{m}_1$ to always be the latest value (exponential update coefficient of 1) yields the stochastic gradient descent algorithm!

Typically, most neural approaches for VC are trained using the Adam optimizer, which follows the update equation shown above with the use of the running average gradient and square gradient to help smooth training [21]. This method for training DNNs is very versatile and allows any operation to be used in the network – regardless of non-linearity or recurrent use of its output as an additional input – so long as it is smoothly differentiable.

2.3.3. Recurrent neural networks

A mel-spectrogram X can be treated both as a sequence of vectors or as a 2D matrix that can be viewed as an image. There are two layers most often used in DNN-based VC models – one to best deal with 2D images, and one to deal with sequential data. For dealing with 2D input images, we use convolutional layers where the coefficients of the discrete 2D convolution filter are treated as part of the weights θ.

For dealing with sequential data, recurrent neural networks (RNNs) are used. RNNs

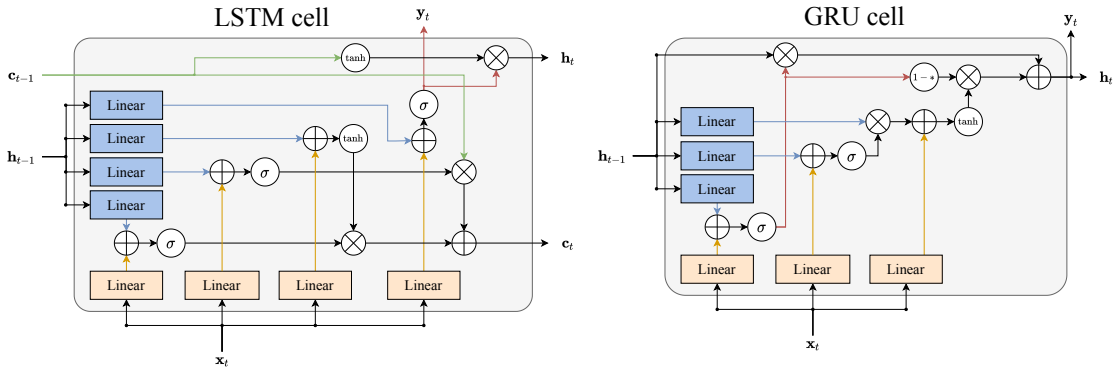


Figure 2.2: GRU and LSTM recurrent cell architectures, showing the operations performed for each item in a sequence [22, 23]. $\mathbf{c}$ is a cell state vector, while $\mathbf{h}$ denotes a hidden state vector. $\mathbf{x}$ and $\mathbf{y}$ denote the input and output vectors respectively. The GRU cell only has a hidden state and no cell state. All operations aside from the linear layers are applied element-wise to their inputs. σ refers to the sigmoid function, and $1 - *$ is the element-wise operation that maps each input value to one minus that input value.

refer to a category of network layers / functions that process a given sequence by iteratively performing some function on each item $\mathbf{x}_i$ in the sequence in order, whereby the function returns some output and some information for processing of the next item in the sequence. This latter piece of information is termed the *state* of the RNN at a particular point in a sequence, and acts both as an output of the previous step's function evaluation and an input to the next step's function evaluation. This *state* is initialized in some pre-defined way for the initial item in the sequence where no previous state exists.

The parameters of the function performed on each sequence item are included in the weights of the network θ. Although the function performed on each sequence item can be any arbitrary DNN, two network models are utilized extensively in existing literature: long-short term memory (LSTM) RNNs [23] and gated recurrent unit (GRU) RNNs [22]. The architectures of both these RNN layers are shown in Figure 2.2. As is seen, the GRU cell is slightly simpler with only three quarters of the number of linear layers as the LSTM cell, making it reasonably faster [22].

2.3.4. Common DNN layers

In addition to the recurrent layers just described, several other common sets of operations – typically collectively called a *layer* of a DNN – are used. In Figure 2.2, the `Linear` layers are the same as those covered in standard undergraduate machine learning courses, whereby $\texttt{Linear}(\mathbf{x}) = \mathbf{x}W^T + \mathbf{b}$ for weight matrix W and bias vector $\mathbf{b}$ – both W and b are included as part of the network parameters θ. The dimensions of the hidden state $\mathbf{h}$ and cell state $\mathbf{c}$ vectors are chosen as a design hyperparameter. This choice, together with the size of the input vector $\mathbf{x}$, determines the shape of each of the linear layer's weights. If the input to a linear layer is a tensor of rank greater than 1 (excluding a possible batch dimension), then the input is assumed to be flattened into a rank-1 tensor (a vector) before being fed into the linear layer.

Another common set of layers is that of *activation functions* – non-linear element-wise functions applied to an input tensor. Common activation functions include the sigmoid function $\sigma(x)$, hyperbolic tangent tanh, rectified linear units (ReLU) which clamps the minimum value of an input at zero [24], and scaled exponential linear units which maps

negative inputs to smaller and smaller positive inputs [25]. The precise details of each function are outside the scope of this report, but suffice to say they are typically chosen based on practical reasons relating to how gradients scale for weights early on in the computation graph of a network. Regardless of the choice of activation function throughout the network, so long as they are non-linear the DNN's generalization ability is guaranteed by the Universal Approximation Theorem [20].

Furthermore, a random layer called dropout has also gained traction in several DNN architectures due to its purported ability to reduce overfitting in DNN models. The dropout layer is straightforward: given an input tensor of arbitrary shape, randomly set each input to zero with some probability p (a hyperparameter set in model design), and scale the output tensor to have the same mean and standard deviation as the input tensor [26]. In some specific applications, the specific entries of a tensor set to zero are made to be in some way 'consistent'. For example, a dropout layer in a RNN might ensure that the same entry is dropped in all of the input vectors $\mathbf{x}_i$ in a particular training batch [27].

Embedding layers map an input one-hot vector to a specific d-dimensional real-valued vector, where each d-dimensional vectors is included as part of the trainable network parameters θ. They are used in some VC systems to get unique floating-point vector representations of a one-hot speaker vector $\mathbf{s}$.

One final layer that is used in this project is that of the `PixelShuffle` layer. This layer intuitively takes in (at least) a rank-3 tensor image, with a width, height, and channels dimension. The `PixelShuffle` layer then rearranges the input to upscale the width and height by some factor, while reducing the number of channels by the square of the upscaling factor. Without delving into the details, this effect is realized by performing a kind of 'sub-pixel' convolution with a stride equal to the inverse of the scaling factor [28]. It is used in vision models and some VC networks to upscale a set of features from a previous layer to bring the output closer to the desired width-height resolution, or in the case of VC systems operating on Mel-spectrograms, to the desired sequence length and number of Mel-frequency bins.

2.3.5. Generative adversarial networks

Several recent high-performing methods for VC utilize generative adversarial networks (GANs). Concretely, a GAN does not refer to a particular DNN layer or operation, but rather to a particular loss configuration whereby two (or more) networks iteratively compete to maximize the other network's losses as they minimize their own loss [29]. Since Goodfellow's introduction of GANs in 2014, a myriad of variants have arisen, each following a slightly different training paradigm to achieve various tasks such as image-to-image translation [4] and even vocoding [15]. Focusing on the prevalent GAN structures in VC, we will first outline the original GAN model, and then discuss two relevant extensions to GANs – StarGAN and LSGAN.

Original GAN

The original GAN model consists of two DNNs – a *generator* G and a *discriminator* D. The generator was proposed to, using some input, produce a *realistic* output from some dataset (for example, a picture of a zebra). Meanwhile, the discriminator is designed to detect fake generated data items from real ones (detecting a generated image of a zebra from an original photograph of a zebra). They are trained such that G's loss $\mathcal{L}_G$ attempts to *maximize* D's loss $\mathcal{L}_D$, while D minimizes its loss by attempting to maximize G's loss.

Algorithm 2.1: Procedure for training traditional GANs. The hyperparameters k and b represent the number of steps to optimize the discriminator D for each step taken to optimize the generator G [29], and the batch size, respectively.

$r \leftarrow 0$
for each training epoch **do**
  for iteration in epoch **do**
    if $r \mod k$ is 0 **then** ▷ Perform G optimization step
      Sample minibatch of G input data $\mathbf{z}_1, \mathbf{z}_2, ...\mathbf{z}_b$.
      Compute $D(G(\mathbf{z}_i))$ for each item in batch.
      Update G by performing SGD optimization step on average loss over batch:

$$\mathcal{L}_G = \log(1 - D(G(\mathbf{z}_i)) \tag{2.1}$$

    else ▷ Perform D optimization step
      Sample minibatch of target data $X_1, X_2...X_b$.
      Sample minibatch of G input data $\mathbf{z}_1, \mathbf{z}_2, ...\mathbf{z}_b$.
      Compute $D(G(\mathbf{z}_i))$ and $D(X_i)$ for each item in batch.
      Update D by performing SGD optimization step on average loss over batch:

$$\mathcal{L}_D = -\left[\log(D(X_i)) + \log(1 - D(G(\mathbf{z}_i))\right] \tag{2.2}$$

    end if
    $r \leftarrow r + 1$
  end for
end for

The two objectives of G and D are opposite, and a balance is needed to ensure one does not begin to dominate the other. This feature of GANs has lent them the reputation of being difficult to train [9]. The basic algorithm to train these networks is given in Algorithm 2.1. In the traditional GAN model, D attempts to predict the *probability* that a given input X has been generated by G.[1] Meanwhile, G is trained to generate data fitting some probability distribution when given a vector $\mathbf{z}$ sampled from a random prior noise distribution.

2.3.6. GAN extensions

Since Goodfellow's introduction of GANs, a family of extensions to the basic training loop of Algorithm 2.1 have arisen. One effective model that has seen applications in VC is that of StarGAN [4], introduced for image-to-image domain translation.

StarGAN

StarGAN changes the traditional GAN setup by positioning the generator to translate data from some source domain to some target domain, instead of translating a random noise sample to an output in a target domain. In the original StarGAN paper, this was proposed to translate across image domains where e.g. the generator G could take in images of horses and translate them to pictures of zebras [4]. Recent further extensions to StarGAN have extended this to translation of video sequences and generation of faces [30].

Concretely, StarGAN designs its generator G to take two inputs – an input in some domain X_{src} and an indication of the target domain to convert the input to $\mathbf{s}_{\text{trg}}$ (typically specified as a one-hot index vector from a fixed set of domains in the training set). It then

[1] This X given as input to D is indeed the same mel-spectrogram X from Figure 2.1 in the case of StarGAN-based VC systems dealt with in this report. However, in general, it could be a sample from a data distribution of arbitrary meaning and form.

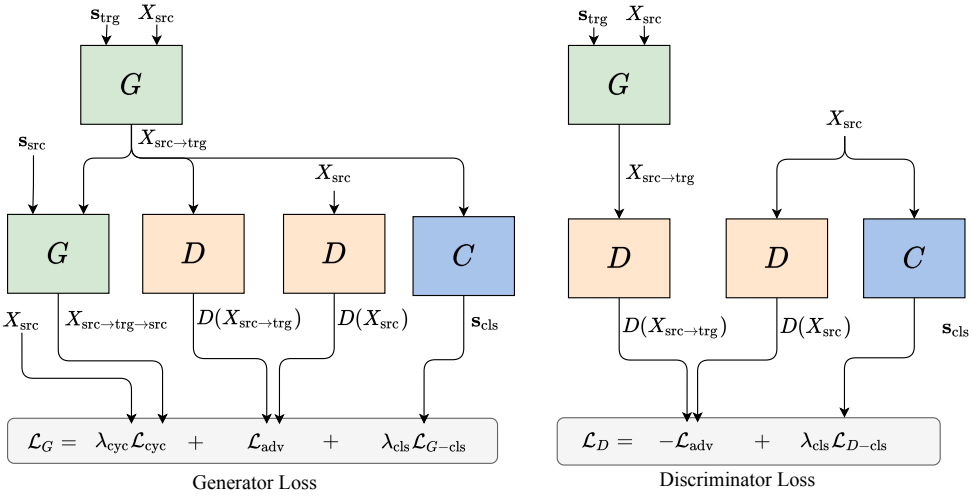


Figure 2.3: StarGAN training configuration [4], showing the various data flows necessary to compute each term in the generator loss (left) and discriminator loss (right) functions. The generator G is shown in green, while the discriminator D and classifier C are shown in orange and blue, respectively.

outputs a converted data item $X_{\text{src}\to\text{trg}}$. On the discriminator side, StarGAN introduces two separate networks – a classic discriminator D as seen for the traditional GAN, and a discriminative classifier C, which predicts the domain an input data item X belongs to by predicting a probability distribution over the training domains $\mathbf{s}_{\text{cls}}$. The classifier C typically shares most of its weights with D and is trained together with the D in the discriminator training step [4].[2]

To effectively train such a model, several additional terms are added to the generator and discriminator's loss functions. The computation paths to construct these terms are given in Figure 2.3. During a generator optimization step, the following loss terms are computed:

- **Cyclic loss term $\mathcal{L}_{\text{cyc}}$**: by feeding the converted output $X_{\text{src}\to\text{trg}}$ back into the generator G together with the one-hot representation of the original source domain $\mathbf{s}_{\text{src}}$, the generator returns the data item translated back into the source domain $X_{\text{src}\to\text{trg}\to\text{src}}$. Such a source-target-source mapping is known as a *cyclic mapping*. StarGAN adds a loss term

$$\mathcal{L}_{\text{cls}} = ||X_{\text{src}} - G(G(X_{\text{src}}, \mathbf{s}_{\text{trg}}), \mathbf{s}_{\text{src}})||_1 \tag{2.3}$$

  to the generator's loss $\mathcal{L}_G$ to force G to retain the semantic content independent of domain to ensure its representation in the target domain contains all the necessary information to reconstruct the original input, or as the original authors put it: it forces the generator to be *cyclically consistent*.
- **Adversarial loss term $\mathcal{L}_{\text{adv}}$**: the adversarial loss term is similar to that of the traditional GAN:

$$\mathcal{L}_{\text{adv}} = \log(D(X_{\text{src}})) + \log(1 - D(G(X_{\text{src}}, \mathbf{s}_{\text{trg}})) \tag{2.4}$$

[2]The notation used here again is consistent with that of the abstract VC system of Figure 2.1 – recall that the Mel-spectrogram X can also be treated as a 2D matrix or image and that utterances from a source and target speaker comprise unique domains. For StarGAN the domains are just image classes and the 2D input images X are of physical objects.

And the motivation is the same – since D attempts to predict the probability that a given input is generated by G, G's loss minimizes the term above to force G to trick the discriminator into predicting low probabilities for samples it generates. Note that, in Equation 2.4 above, the first term actually falls away in the derivative since it has no dependence on G's parameters θ_G. However it does play a role in the discriminator's adversarial loss term. I.e. it will impact the updates of θ_D – see below where the procedure fo the discriminator updates are described.

- **Classification loss $\mathcal{L}_{G-\text{cls}}$**: the converted output $X_{\text{src}\to\text{trg}}$ is fed into a DNN classifier to get a vector of probabilities over possible training domains $\mathbf{s}_{\text{cls}}$. This is compared to the actual target domain one-hot vector $\mathbf{s}_{\text{trg}}$ to form a classification loss:

$$\mathcal{L}_{G-\text{cls}} = -\log(C(G(X_{\text{src}}, \mathbf{s}_{\text{trg}})) \cdot \mathbf{s}_{\text{trg}}) = -\log(\mathbf{s}_{\text{cls}} \cdot \mathbf{s}_{\text{trg}}) \tag{2.5}$$

  By minimizing the this term with respect to G's parameters, G is forced to produce outputs which *maximize* the classifier's ability to correctly classify $X_{\text{src}\to\text{trg}}$ as belonging to the desired target domain.

Similarly for the discriminator's optimization step, we compute two terms for the loss. The first is the negative of the adversarial loss term of Equation 2.4 – it is used in the same way as in Algorithm 2.1 to train the discriminators parameters θ_D. The second term is one to train the classifier C, computed by feeding the original input data X_{src} into the classifier to get a vector of probabilities over the possible training domains. This vector is then compared to the one-hot vector representing the original source domain:

$$\mathcal{L}_{D-\text{cls}} = -\log(C(X_{\text{src}}) \cdot \mathbf{s}_{\text{src}}) = -\log(\mathbf{s}_{\text{cls}} \cdot \mathbf{s}_{\text{src}}) \tag{2.6}$$

The inquisitive reader might recognize this as the standard binary cross entropy loss for training a machine learning classifier! This term simply ensures that C is trained to act like a standard classifier model. C's parameters are typically partly shared with the discriminator D, and are optimized at the same time as the classic discriminator D in the training algorithm. Phrased differently, we treat C's parameters as part of θ_D.

Finally, the coefficients λ_{cls}, λ_{cyc} are scalar hyperparameters used to balance the importance of difference terms when training. In summary, StarGAN replaces the generator and discriminator's loss functions in the training Algorithm 2.1 with those shown in Figure 2.3. One aspect to note about the training configuration shown in Figure 2.3 is that a ground-truth target X_{trg} *is never used*. Thus, one does not need data items X_{src} and X_{trg} containing the same domain-independent semantic content to train the model – only X_{src} is needed. Recalling the definitions of parallel/non-parallel models from Section 2.1, this means that StarGAN is a non-parallel model because it can be trained with non-parallel data. This fact is exploited in non-parallel VC systems discussed in Chapter 3.

LSGAN

A second GAN modification used by modern VC algorithms is that of the *least-squares generative adversarial network* (LSGAN) [31]. Like StarGAN, it entails a change to the loss function of the generator and discriminator, but unlike StarGAN, the change is purely mathematical and does not require additional invocations of the models or additional loss terms. Simply put, LSGAN entails changing the adversarial loss term so that, instead of D predicting the probability of an output being generated by G, the discriminator predicts an *unbounded scalar*.

LSGAN defines an adversarial loss for the generator $\mathcal{L}_{G-\text{adv}}$ and discriminator $\mathcal{L}_{D-\text{adv}}$ consisting two hyperparameter scalar constants a and b. G's loss tries to push D's output for converted data items closer to a, while D's loss function tries to push D's output for converted items closer to b and its output for real data samples closer to a [31]. Concretely, the generator's adversarial loss is replaced with the LSGAN adversarial loss

$$\mathcal{L}_{G-\text{adv}} = [D(G(X_{\text{src}}, \mathbf{s}_{\text{src}})) - a]^2 \tag{2.7}$$

while the discriminator's adversarial loss is no longer just the negative of the generator's loss, but rather is now defined as

$$\mathcal{L}_{D-\text{adv}} = [D(G(X_{\text{src}}, \mathbf{s}_{\text{src}})) - b]^2 + [D(X_{\text{src}}) - a]^2 \tag{2.8}$$

Here we have used the same notation as in the StarGAN setup because the notation lends itself well to its further extension for VC discussed in Chapter 3, where X refers to an utterance and $\mathbf{s}$ refers to some representation of the (source or target) speaker identity. LSGAN purpose is to stabilize the training process of the GAN by punishing D more severely when it predicts values far from the desired constants a and b [31]. LSGAN, like other GAN extensions, can be utilized together with StarGAN. A particular GAN architecture can incorporate the LSGAN loss by simply replacing its adversarial loss terms for G and D with those of Equations 2.7 and 2.8 respectively.

2.4. Related work

We now proceed from the theory used in VC systems to the history and recent progress of VC models. The origins of VC is found in the field of digital signal processing. With the resurgence of neural networks in the past decade, recent VC models have splintered into several directions based on DNNs. Two popular approaches which have broken state-of-the-art results in the past few years have been methods based on autoencoders and those based on GANs. In this section we discuss recent trends in VC research and methods used, saving the specific model details for the implementations compared and analyzed in Chapter 3.

2.4.1. Traditional methods

Traditional VC models can typically be identified by their ability to only perform one-to-one conversion that requires either parallel or no training data. Most of the early VC models in the 1980's and 1990's treated speech as an explicit statistical model. Toda et. al. [32] and Stylianou et. al. [33] used Gaussian mixture models in the 1990's to achieve compelling results at the time by modelling a special relationship between the 'spectral envelope' (pitch) of the source and target speaker, using dynamic time warping (DTW) – a technique to align two sequences of vectors – to train a parallel VC system.

Some other models attempted to model the frequency warping effect of the human vocal tract [16]. Meanwhile, other traditional models attempted to model the specific frequency bands present (high amplitude in the STFT/Mel-spectrogram) in a speaker's speech (called formants) [34, 35]. In general, these traditional techniques can be classified into rules-based methods [16, 34, 35] and statistical models of speech [32, 33]. More recently, some DNN aproaches have attempted to use stacked LSTMs to perform one-to-one VC [36].

These traditional methods have fallen out of favor in the recent VC literature [1,12] due to their inability to handle many-to-many VC mappings. The field is currently dominated by neural network approaches, both for the vocoder and the mapping function of Figure 2.1. Note, however, that the training process of a neural network can be phrased in many circumstances as a statistical model (with DNN parameters θ) being optimized to fit some probability distribution (the samples of which comprise the dataset) by maximizing a maximum likelihood or proxy metric. This report, however, will continue to treat DNNs as pure function approximators, whereby the DNN is trained to approximate an 'ideal voice converter' that perfectly performs the VC task.[3]

2.4.2. Autoencoder methods

An autoencoder is a model (typically a neural network) consisting of two sub-modules – an *encoder* and a *decoder*. The encoder compresses the input X to some lower dimensional representation, while the decoder attempts to expand this to reconstruct the original X. The full autoencoder network is typically trained to reconstruct its input by minimizing the difference between the decoder's output and the encoder input. Since they are only trained to reconstruct inputs, they can be trained with non-parallel voice data.

With the introduction of neural networks in the last decade, several variants of autoencoders have arisen such as vector quantized and variational autoencoders [37]. Like with GANs, a handful of these extensions have been developed for use in VC. Kameoka et. al. [38] proposed a variational autoencoder with an auxiliary classifier to perform VC, while Tobing et. al. developed a cyclic autoencoder to achieve high-performing non-parallel VC [39] by optimizing reconstruction after a cyclic source-target-source mapping.

We will look in particular at the AutoVC model introduced by Qian et. al. [9] since it was the first VC model to achieve zero-shot VC and satisfies several of the practicality requirements of Chapter 1. It achieved this by introducing a non-trivial way to encode the desired source and target speaker information. AutoVC achieves this by utilizing a special speaker embedding network to generate a unique embedding for the source and target speaker $\mathbf{s}_{\text{src}}$ and $\mathbf{s}_{\text{trg}}$, instead of encoding the speaker information as trivial one-hot vectors or ignoring them outright as previous one-to-one VC models did.

2.4.3. GAN methods

With the success of GANs in image-to-image translation across domains, particularly StarGAN's ability to train with non-parallel data as shown in Section 2.3.6, several authors attempted to adapt these techniques for VC. Figure 2.4 shows how the many-to-many image translation task performed by existing GAN systems can be directly applied to VC. This direct application is possible because a Mel-spectrogram X_{src} can be treated as a 2D image, and different speaker identities as different 'image domains'. The methods, reasoning, and theoretical justifications for image domain translation GANs like StarGAN must then be valid for VC.

Kameoka et. al. applied the StarGAN technique directly as in Figure 2.4 to the VC task in [7] to achieve high performing results. They went on to further adapt the StarGAN technique to develop StarGAN-VC2 – a state-of-the-art many-to-many VC system [30]. Other authors have modified other GAN extensions to achieve VC, such as

[3] Multiple such functions may exist, as one can imagine multiple valid conversions for an input utterance. The DNN need only approximate one arbitrary valid 'ideal VC function'.

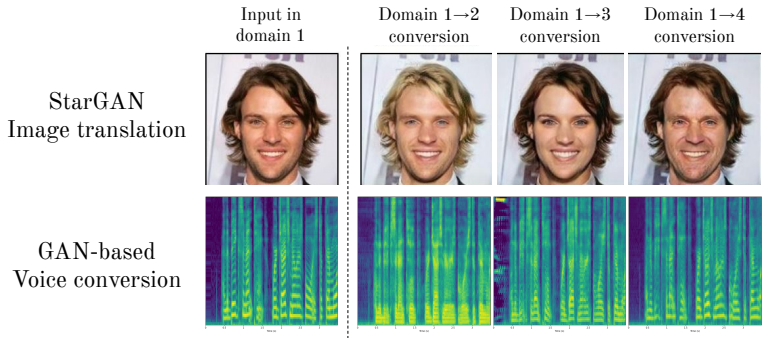


Figure 2.4: Equivalency of StarGAN image translation and VC tasks, showing the original many-to-many image domain translation results by StarGAN (top) [4] together with how many-to-many VC can be expressed as an equivalent image domain translation task (bottom). Each image in the bottom row is the Mel-spectrogram of a speech utterance, with the converted utterances produced using the StarGAN-ZSVC model introduced in Chapter 3.

the AdaGAN-VC system by Patel et. al. [40]. These GAN-based systems have seen success in recent VC competitions [1, 12], owing to their ability to perform both many-to-many conversion and ability to be trained with non-parallel data.

2.4.4. Vocoders

So far we have only described the mapping function of the full VC system (recall Figure 2.1). GANs and other model architectures have also played a role in vocoder systems. Some models, such as voice conversion based on modelling the vocal tract [16] or StarGAN-VC [7], use parametric algorithms like the WORLD vocoder [19] to convert a model's spectrogram output back to a time-domain waveform. Others use more recent neural vocoders, which can be further divided into autoregressive models, such as those of the WaveNet family [14] and the recent WaveGlow models [13], and non-autoregressive models, such as the GAN-based MelGAN [15].

The fine details of how vocoders work is beyond the scope of this project, since vocoding is a substantive research field in its own right. Suffice to say, the vocoder converts the Mel-spectrogram back to a time-domain waveform and each has its advantages and disadvantages. The autoregressive models like WaveNet [13] are typically of higher quality than non-autoregressive models [11], however this comes at a cost: because autoregressive models – by definition – must operate sequentially to progressively build up its output, they are slower than non-autoregressive models.

Sometimes this effect is massive. For example, WaveNet models can often take 10-30 seconds to convert one second of audio, however the conversion quality will be immaculate compared to MelGAN, which can convert a second of audio in only a few milliseconds on an Nvidia RTX 2070 graphics card. Other times it is less pronounced, with the autoregressive WaveGlow model only being 10x slower than the non-autoregressive models.

2.4.5. Speaker identification

A further interesting innovation introduced by AutoVC is the technique of using a specific *speaker embedding* model for the 'speaker information extraction' block of Figure 2.1. By using an explicit speaker information extraction model, AutoVC can achieve zero-shot VC

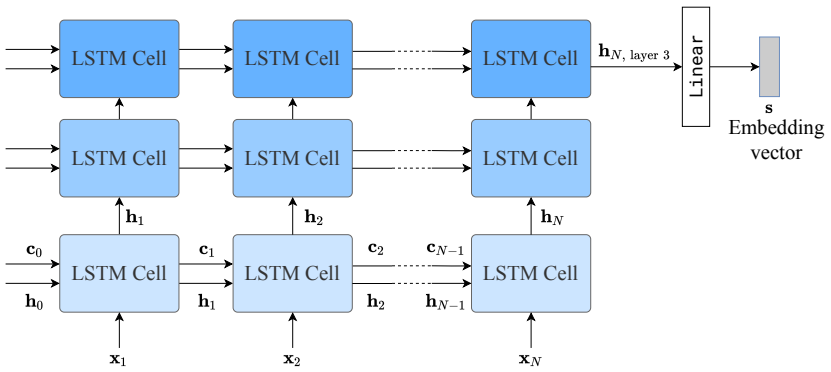


Figure 2.5: The 3-layer LSTM model used for speaker identification in the GE2E model. For visual clarity, the notation for the 2nd (middle) and 3rd (top) LSTM layers are omitted. All cell invocations with the same color share parameters. All hidden and cell states are initialized as zero vectors.

by using the model to obtain embeddings $\mathbf{s}$ for speakers unseen during training.[4] In fact, only two known models – aside from the StarGAN-ZSVC model developed in this project – can perform zero-shot VC (AutoVC [9] and the pre-print ConVoice VC model [41])) and both use the same speaker embedding idea to achieve zero-shot VC.

Concretely, the speaker embedding network is repurposed from an existing successful architecture for speaker identification. *Speaker identification* is the task of, given an input utterance X, determining the identity of the speaker of the utterance. Although speaker identification encompasses another research field itself, one model in particular is useful for VC because it achieves speaker identification by finding a fixed dimensional embedding $\mathbf{s}$ for an arbitrary speaker. This embedding can then be treated as the information vector $\mathbf{s}_{\text{src}}$ for the source speaker and $\mathbf{s}_{\text{trg}}$ for the target speaker in VC systems.

The embedding system is that introduced in the paper Generalized End-to-End (GE2E) Loss for Speaker Verification [42]. It consists of an embedding network and a special GE2E loss. The embedding network is a stacked 3-layer LSTM network which takes in a Mel-spectrogram X and returns a $d-$dimensional vector $\mathbf{s}$ – known as a $d-$vector in the speaker identification literature.

In such a stacked LSTM network, each frame of the Mel-spectrogram X is fed into a LSTM layer as the input vector $\mathbf{x}_i$ as described in Section 2.3.3. The data is then fed through the stacked LSTM as shown in Figure 2.5 which shows how a stacked LSTM operates by treating the hidden states of the first LSTM layer as a new sequence of inputs to a separate LSTM layer with different parameters. As an aside, some authors prefer to use the sequence if *outputs* $\mathbf{y}_1, \mathbf{y}_2, ...\mathbf{y}_N$ as the input to the next layer as opposed to the sequence of hidden states in Figure 2.5. Most implementations in software libraries – and funnily enough by implication in most recent papers using multi-layer RNN systems – use the hidden state sequence as input to the next RNN layer [27], thus we use it in our experiments here to be consistent with the major software libraries. The h-dimensional hidden state of the final layer is fed through a linear layer to project it to d-dimensions and then divided by its Euclidean norm, yielding the embedding vector $\mathbf{s}$.

The final piece of the puzzle is how such a GE2E network is trained such that the $\mathbf{s}$ vector represents information about the speaker of an input utterance. To achieve this,

[4] All models prior to AutoVC had either ignored this section of the VC system (as in one-to-one models), or used trivial encodings of speaker identity (such as a one-hot encoding, or fixed embedding for each training speaker).

the second part of the GE2E system is a special GE2E loss. Intuitively, the GE2E loss is made to maximize the cosine similarity between embeddings of utterances from the same speaker while at the same time attempting to minimize the cosine similarity between embeddings of utterances from other speakers.

Concretely, the embedding model takes in U utterances from K different speakers in each batch, generating UK embeddings. Using these embeddings, we compute the geometric *centroid* $\mathbf{c}_k$ over the embeddings for each speaker, giving us K centroids – each acting as an 'average embedding' vector that speaker. Then, a scaled cosine similarity is computed for each embedding/centroid pair $w \cdot \cos(\mathbf{s}, \mathbf{c}_k) + b$ where w and b are trainable parameters shared by all cosine similarity computations. Performing this yields UK^2 cosine similarity scores – one for each pairing of the UK embeddings and K centroids.

Finally, the scalar loss for each embedding $\mathbf{s}$ (belonging to speaker k) is defined as the sum of the negative cosine similarity to the k'th speaker centroid $\mathbf{c}_k$ and the log of the sum of K additional terms. These K terms are the natural exponentiation of the cosine similarity between $\mathbf{s}$ and each of the K centroids [42]. The first part of the loss function guides the model to maximize the cosine similarity for embeddings of utterances from the same speaker, while the second part forces the embeddings to be far apart from *all* centroids – and thus all other embeddings. The average of this loss is taken over all utterances in the batch for the optimization step.

Training such a network with this specially crafted loss function yields well-behaved embedding behavior [42]. This means that embeddings of utterances saying vastly different things – but by the same speaker – still yield embeddings very close to one another in $d-$dimensional space. It also means that utterances saying precisely the same thing but by different speakers yield embeddings far apart in $d-$dimensional space. Such functionality provides us with a way to obtain a unique vector associated with each speaker, simply by passing an arbitrary utterance from the desired speaker through the network.

2.5. Summary

This chapter began with a general description of VC systems and the terminology used to describe each part of the system. We then proceeded to describe the theory necessary to reason about and discuss common techniques used in the various blocks of a VC system. Finally, we provided a broad outline of recent efforts in VC in Section 2.4, covering several classes of VC models and their supporting vocoders and feature extractors.

Now, with this background in place, we can proceed to define concrete implementations of VC systems in the next chapter, and discuss how their construction defines their capabilities and limitations. In particular as we proceed, Figure 2.1 (which defines the general form of a VC system) will be crucial. All the models we examine in the next chapter will entail filling in the various blocks of the full VC system – from the mapping function / model to the speaker information representation – using different methods. In each case, using the background of this chapter should make the reasoning behind certain architecture choices appear trivial.

For example, with the theory behind the GE2E speaker verification model in the previous section, one can begin to see how achieving zero-shot VC is possible: an utterance from a speaker unseen during training can be fed into the speaker embedding network to yield a speaker embedding $\mathbf{s}$ which can be used to condition the VC mapping function to convert to/from the unseen speaker. Yet, before the introduction of the AutoVC model in 2019, no authors had made this connection which now seems almost obvious.

Chapter 3

Models

In this chapter we will describe six VC models – three baseline one-to-one models of increasing complexity, two state-of-the-art many-to-many VC models, and finally our own VC model design – StarGAN-ZSVC. For each model, the motivation behind including it in our analysis is given along with the structure of the model and the abstract form by which it is trained.

As mentioned in Section 2.2, each model will define the core mapping function and the way speaker information is represented. So long as the vocoder is sufficiently fast, the feature extraction and vocoding choices are not the limiting factors in determining whether a VC system satisfies the practicality requirements of Chatper 1. Thus we will define and use a standard feature extraction (using Mel-spectrograms) and vocoding process (using the WaveGlow vocoder) later in Chapter 4. We use these across all models to ensure a fair comparison.

3.1. One-to-one baseline models

The first set of models we consider are three one-to-one VC models. All the one-to-one models require parallel data, and are optimized by comparing the model output $X_{\text{src}\to\text{trg}}$ to the ground-truth target Mel-spectrogram X_{trg} using an L_1 loss (which we found to produce better results than the L_2 loss). In all one-to-one models, the full VC system ignores the speaker information because such information is implicit in the one-to-one mapping that the model is trained to perform.

3.1.1. Linear convolutional network

The first model we consider is a purely linear DNN with no activation functions. It is chosen as a simple baseline to ensure that the rest of the software framework is working and to serve as a sanity check that any designed VC system we make should outperform the linear baseline.

Treating the input Mel-spectrogram X_{src} as a 2D image, the linear model consists of 4 convolutional layers with output channels and kernel sizes of (200, 5x5), (200, 5x5), (100, 3x3), and (1, 3x3), respectively. The input Mel-spectrogram is fed sequentially through each of these convolutional layers to produce the output. The network architecture is is chosen based on experimentation and practical experience with DNNs to try get the optimal performance (on a validation set) with only a few linear layers.

In such a network, all inputs should come from a single source speaker and all ground-truth targets should be from a single target speaker – the network is optimized to only convert from speaker A to speaker B. Further, since the model consists of only a handful of linear convolutional layers, this network is incredibly fast to perform inference – helping it satisfy at least the speed requirement. The other practical requirements are disregarded since this network is designed as a simple baseline model to compare others against.

3.1.2. UNet-based network

A common architecture in image-to-image translation models is that of the UNet architecture, which consists of several convolutional layers that encode an input image down to a lower dimensional space, and then a series of transposed convolution or pixel shuffle layers which bring the lower dimensional data back to the original input dimensions. The input to set of convolutional layers on the downward / encoder path is also concatenated as an input to each corresponding set of pixel shuffle layers on the upward / decoder path in what are known as 'skip connections' [43].

Since the original UNet was proposed, several improvements have been suggested, and one UNet derivative that maintains very high performance on image translation tasks is that of Howard's DynamicUNet [44] based on XResNet. XResNet is a convolutional model intended for image classification, developed by implementing several improvements to a previous convolutional DNN model [45]. Howard's DynamicUNet arranges the layers of XResNet into the encoder of a UNet model, and then constructs an appropriate upward / decoder path using pixel shuffle and further convolution layers [44].

Again, as a one-to-one model it uses parallel training data and compares the output Mel-spectrogram (image) to the ground-truth X_{trg}. We chose to include this DynamicUNet model in our comparison to assess how well recent image-to-image translation systems perform when directly applied to VC without any adaptation or changes. The precise model architecture is rather large and cumbersome to enumerate, requiring significantly more background from the visual segmentation field to motivate certain layer choices and hyperparameters. Thus, since the goal of evaluating this model is to assess the 'out-of-the-box' performance of a high performing image-to-image translation model, only a superficial description of the model architecture is provided.

At a high level, the UNet model follows the form shown in Figure 3.1. with over 31 million parameters. Each block in the figure represents several layers of specially crafted sizes such that, in addition to yielding high performance on image-to-image translation tasks, the $m \times N$ input Mel-spectrogram X_{src} yields an output $X_{\text{src}\to\text{trg}}$ of the same shape $m \times N$. The various blocks are comprised of arrangements of convolutional layers, ReLU and sigmoid non-linearities, `PixelShuffle` layers, and a kind of image normalization layer

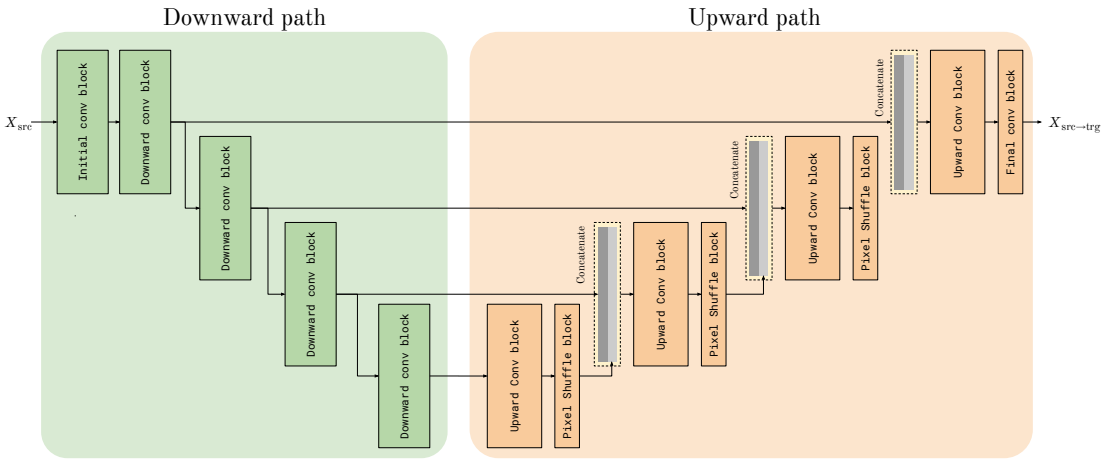


Figure 3.1: High level architecture of XResNet-based UNet constructed with the DynamicUNet method [44], showing the downward and upward paths and skip connections between each downward and upward set of layers. The full model architecture is defined precisely in the original DynamicUNet method by Howard [44].

known as `BatchNorm` [44].

Note that, due to the nature of the kind of layers present, the input spectrogram X can be of arbitrary length N. If instead the model included linear layers, then the input would need to be of fixed dimensions because the matrix multiplication of a linear layer requires the input to be of a fixed shape. This model serves as a more advanced one-to-one conversion baseline based on directly applying image-to-image mapping techniques without adaptation.

3.1.3. Deep bidirectional LSTM (DBLSTM)

We next consider a modification of the deep bidirectional LSTM (DBLSTM) model proposed by Sun et. al. in 2015 [36]. This model serves as one of the more advanced methods for one-to-one VC, and most work since 2015 has moved on to many-to-many systems [11] using the techniques outlines in Section 2.4. The modified DBLSTM model is described in Figure 3.2.

The model is strikingly similar to the LSTM model for speaker identification discussed in Section 2.4.5, with the DBLSTM model defined by four stacked bidirectional LSTM layers with a hidden state $\mathbf{h}$ size of 256 and a dropout of 0.3, followed by a final linear layer to project the output the LSTM output back to m dimensions (recall m is the number of bins in the Mel-spectrograms X). A bidirectional LSTM layer is identical to two traditional LSTM layers, except one receives the input sequence of Mel-spectrogram frames $\mathbf{x}_i$ in *reverse order*. The bidirectional LSTM layer is then completed by treating the hidden state as the *concatenation* of the hidden state $\mathbf{h}$ for the forward and backward LSTM layers [36]. Thus, the bidirectional LSTM layer acts exactly as the traditional unidirectional LSTM to external layers, with the exception that the each hidden state passed to the next layer is twice as long as before.

The original DBLSTM also had three important differences to our modified model. First, it used the STRAIGHT vocoder, which is similar to the WORLD vocoder discussed in Section 2.3.1 in that it represents speech as a 3-tuple of MFCCs, aperiodicity signal, and pitch contour. The second is that the DBLSTM model *only* attempted to convert the MFCCs of the source speaker to match those of the target speaker, and performed a linear

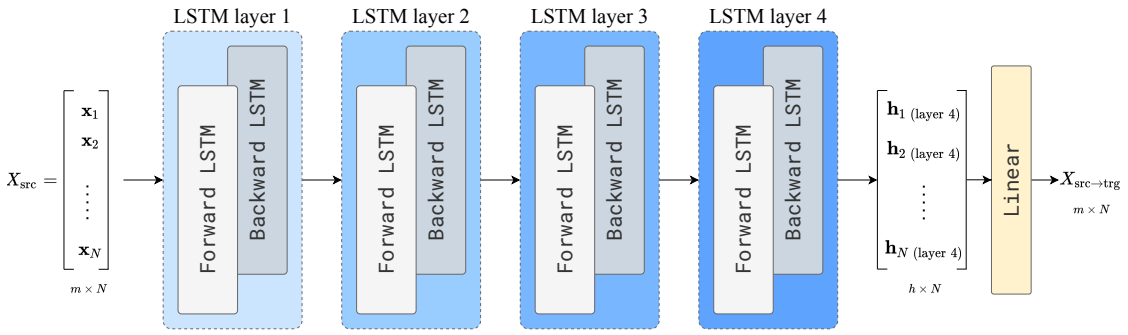


Figure 3.2: Architecture of the DBLSTM VC model adapted from the original Sun et. al. model [36]. Each LSTM layer has a hidden state size of $h = 256$. The model takes as input a sequence of m-dimensional vectors (Mel-spectrogram), and returns a sequence of hidden states at the final layer, which are each separately fed through the same linear layer to project them from $\mathbb{R}^h$ to $\mathbb{R}^m$. This sequence of m-dimensional output vectors comprise the frames of the converted Mel-spectrogram. The matrix sizes of the input and output are indicated below each variable.

conversion for the other two items in the tuple using the mean pitch and aperiodicity computed from utterances from the single target speaker. As mentioned at the start of the chapter, we opt to use a uniform vocoder and Mel-spectrogram feature extraction process to fairly compare the VC models in isolation. Thus our implementation uses the DBLSTM model to convert Mel-spectrograms instead of MFCCs, and the resulting Mel-spectrograms are synthesized to speech using the same WaveGlow vocoder as the other models.

The second important distinction is that the original DBLSTM uses a time alignment technique (DTW, discussed further in Chapter 4) between the source and target utterances to try help the model learn more effectively [36]. We opt to not use this, as we already use the DTW alignment technique to construct several of our metric comparisons in Chapter 5 and no other models considered use an alignment technique. The DTW alignment of the source spectrogram to the target spectrogram can also be viewed as part of the feature extraction block of Figure 2.1. So since we have already decided to use a consistent standard Mel-spectrogram feature extraction process, we do not use any alignment techniques.

Furthermore, in our version of DBLSTM dropout is applied to the sequence of hidden states passed between LSTM layers with the zeroing of elements done consistently for hidden states in the same sequence. We included this as it appeared to improve the performance on the validation set substantially. Again, since the DBLSTM is a one-to-one model, it does not use any explicit speaker information and is trained to minimize the L_1 difference between the output sequence $X_{\text{src}\to\text{trg}}$ and the ground-truth target utterance X_{trg}. The DBLSTM model is included to assess how a recent traditional (in the sense that it can only perform one-to-one conversion) model designed specifically for VC performs in terms of the practicality requirements of Chapter 1.

3.2. StarGAN-VC2

We now proceed to the many-to-many models which are much closer to being practical. The first model we look at is the many-to-many StarGAN-VC model introduced by Kaneko et. al. [7] and in particular its extension with StarGAN-VC2 [8]. The StarGAN-VC and StarGAN-VC2 models are beginning to achieve near human level results with StarGAN-VC2 being a frontrunner at the recent VCC 2020 competition [12]. StarGAN-VC was a direct application of StarGAN (as discussed in Section 2.3.6 and 2.4.3) to VC, while StarGAN-VC2 makes several modifications to tailor the StarGAN model for VC.

Unfortunately, like the DBLSTM model just discussed, both StarGAN-VC and StarGAN-VC2 use the WORLD feature extractor and vocoder [19] and represents an input utterance as a 3-tuple (as discussed in Section 2.3.1). The models only convert the MFCC from the source utterance to the target domain and performs a linear conversion – utilizing the average pitch statistics for utterances of the source and target speaker – for the other two elements of the 3-tuple of each input utterance.

The authors design the model to achieve high performance in a low-resource setting using this specific WORLD feature extraction and vocoder, and acknowledge that the performance drops substantially when the model is forced to convert the full 3-tuple of each utterance or a simple Mel-spectrogram [8]. However, this model is included in this project to assess a state-of-the-art GAN-based VC model. And since the performance of this model is strongly tied to the WORLD vocoder, we opt to make an exception and **retain the original WORLD feature extraction and vocoder in our experiments and analysis of this model**.

Since StarGAN-VC2 is a strict improvement of StarGAN-VC [8], we will only an-

alyze StarGAN-VC2. In particular, StarGAN-VC2 [8] extends upon StarGAN-VC [7] by training a single generator model to perform many-to-many voice conversion using speaker-dependent modulation factors in specially designed DNN layers. Concretely, StarGAN-VC2 follows precisely the same training regime and structure as the StarGAN model applied to VC discussed in Section 2.4.3 with the following changes applied:

- The model's generator $G(X_{\text{src}}, \mathbf{s}_{\text{src}}, \mathbf{s}_{\text{trg}})$ and discriminator $D(X, \mathbf{s}_{\text{src}}, \mathbf{s}_{\text{trg}})$ are conditioned on *both* the source and target speaker, unlike the StarGAN and StarGAN-VC models which are only conditioned on the *target* domain/speaker. Like with the original StarGAN, the source and target speaker identities are given as one-hot vectors, $\mathbf{s}_{\text{src}}$ and $\mathbf{s}_{\text{trg}}$, respectively. The input X_{src} is now a $36 \times N$ MFCC, as per the original paper.

- The discriminator $D(X, \mathbf{s}_{\text{src}}, \mathbf{s}_{\text{trg}})$ returns a scalar (instead of a probability), and the overall GAN is trained using LSGAN (see Section 2.3.6) for the adversarial loss terms. No separate discriminative classifier is used as was the case for StarGAN. Instead, the discriminator fills the role of the traditional GAN discriminator and the classifier by yielding a scalar indicating how confident it is that a spectrogram X was generated by G converting it from some specified source domain $\mathbf{s}_{\text{src}}$ to some target domain $\mathbf{s}_{\text{trg}}$.

- Specially designed conditional instance normalization (CIN) layers are added to allow speaker specific information to modulate convolutional layer outputs along the channels dimension [46].

- The architecture of the generator and discriminator are carefully designed using a 2-1-2D DNN generator architecture and a projection discriminator [47].

3.2.1. Loss function

Recalling the background on GANs, the generator is trained to force the discriminator's output when given converted spectrograms to be high, while the discriminator is trained to make its output low when given converted outputs and high when given original spectrograms.

To make the above changes above more formal, we define the loss function explicitly using the source-and-target conditional generator and discriminator. The generator G is trained to minimize the loss $\mathcal{L} = \lambda_{\text{id}}\mathcal{L}_{\text{id}} + \lambda_{\text{cyc}}\mathcal{L}_{\text{cyc}} + \mathcal{L}_{G-\text{adv}}$. The first term, $\mathcal{L}_{\text{id}}$ is an identity loss term added to the traditional StarGAN generator loss to stabilize training. It aims to minimize the difference between the input and output spectrogram when the model is made to keep the same speaker identity, i.e. convert from speaker A to speaker A. It is defined by the square L_2 loss:[1]

$$\mathcal{L}_{\text{id}} = ||G(X_{\text{src}}, \mathbf{s}_{\text{src}}, \mathbf{s}_{\text{src}}) - X_{\text{src}}||_2^2 \tag{3.1}$$

Next, the cyclic loss is added to ensure that StarGAN-VC2 remains cyclically consistent. This second loss term is nealy identical to the original cyclic loss term of Equation 2.3 and is motivated by the same reasons. Formally, the second term of the loss aims to minimize

[1]The original system used a pure L_2 loss, but we found a square L_2 loss to yield better results in our experiments.

the cyclic reconstruction error between the cyclic mapping and original spectrogram:

$$\mathcal{L}_{\text{cyc}} = ||X_{\text{src}} - G(G(X_{\text{src}}, \mathbf{s}_{\text{src}}, \mathbf{s}_{\text{trg}}), \mathbf{s}_{\text{trg}}, \mathbf{s}_{\text{src}})||_1 \tag{3.2}$$

Finally, the adversarial loss term $\mathcal{L}_{G-\text{adv}}$ is added based on the LSGAN loss as discussed in Section 2.4.3. Concretely, G's adversarial loss is defined as

$$\mathcal{L}_{G-\text{adv}} = \left(D(G(X_{\text{src}}, \mathbf{s}_{\text{src}}, \mathbf{s}_{\text{trg}}), \mathbf{s}_{\text{src}}, \mathbf{s}_{\text{trg}}) - a\right)^2. \tag{3.3}$$

while, the discriminator D is trained to minimize the corresponding LSGAN discriminator loss:

$$\mathcal{L}_{D-\text{adv}} = \left(D(G(X_{\text{src}}, \mathbf{s}_{\text{src}}, \mathbf{s}_{\text{trg}}), \mathbf{s}_{\text{src}}, \mathbf{s}_{\text{trg}}) - b\right)^2 + \left(D(X_{\text{src}}, \mathbf{s}_{\text{trg}}, \mathbf{s}_{\text{src}}) - a\right)^2 \tag{3.4}$$

For the original StarGAN-VC2, the authors set the scalar coefficients as $\lambda_{\text{id}} = 5$, and $\lambda_{\text{cyc}} = 10$. The original study [8] does not mention how a and b are set (despite these greatly affecting training); we treat them as hyperparameters and found $a = 1$, $b = 0$ to work well in our experiments.

3.2.2. Architecture

StarGAN-VC2 uses a specially designed 2-1-2D convolutional architecture for the generator, as well as a projection discriminator [47] which comprises of a convolutional network (to extract features) followed by an inner product with an embedding corresponding to the source/target speaker pair. For the generator, a new form of modulation-based conditional instance normalization was introduced in [8]. This allows the speaker identity (which is provided as a one-hot vector) to multiplicatively condition the channels of an input feature. According to [8], this special layer is a key component in achieving high performance in StarGAN-VC2.

Unfortunately, StarGAN-VC2 is not open source, and the original paper includes a rather nasty error in its model definition.[2] Without being certain that our implementation is the same as the original StarGAN-VC2, we make a best assumption interpretation from the original paper and construct the model as shown in Figure 3.3.

The network architecture requires a few remarks. The first is that both the generator and discriminator concatenate the d-dimensional one-hot embeddings for the source and target speaker together, and then one-hot encode the result. Since each one-hot speaker encoding $\mathbf{s}$ has d possible values, the one-hot encoding of the pair $(\mathbf{s}_{\text{src}}, \mathbf{s}_{\text{trg}})$ has d^2 possible values. Thus the output of the `One-hot encode` layers of Figure 3.3 have an output size of d^2, with a unique one-hot encoding for each source-target speaker pairing.

The second aspect to note is that the projection discriminator (based off of [47]) D performs a dot product of a 512-dimensional embedding with the output of a global sum pooling layer – a layer which simply returns the input tensor summed along the width and height dimension. This fixed-dimensional dot product means that the input size X must be fixed – and thus that any input utterance to D must be of fixed length (128 MFCC frames). However it does not hinder the ability of the VC model, as the generator G does not suffer from any fixed-dimension requirements and can convert inputs of arbitrary

[2] The reader might find it instructive to inspect Figure 3 of [8] and carefully review the channels and shapes of the final few generator layers. There is no way to consistently interpret the convolutional layer sizes that makes the model mathematically valid.

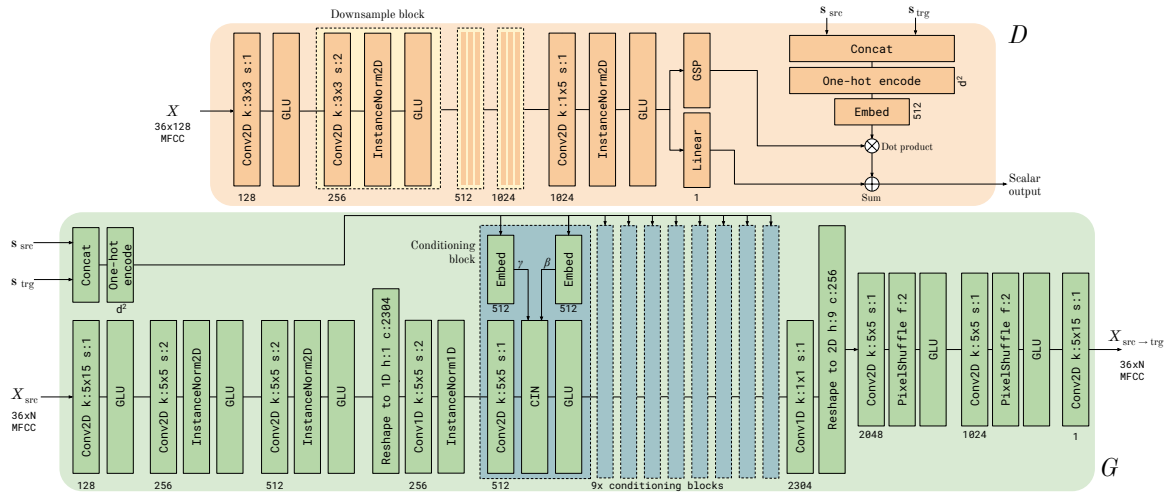


Figure 3.3: The generator G and discriminator D architectures of the StarGAN-VC2 model [8]. Within layers, `k` and `s` represent kernel size and stride (for convolutions), `f` is the scaling factor (for pixel shuffle), and `h` and `c` are the height and channels of the output (for reshape layers). A number alongside a layer indicates the number of output channels (for convolutions), or output units (for linear and embedding layers). GLU layers split the input tensor in half along the *channels* dimension. GSP, GLU, and Embed indicate global sum pooling, gated linear units, and embedding layers, respectively.

length. Thus, *during training*, all input utterances must be trimmed to only 128 MFCC frames while *during inference* (when only the generator is used), input utterances can be of arbitrary length N.

A further important part of the architecture is the 'Conditional block' of Figure 3.3. These conditional blocks are intended to provide the network with a way to modulate the channels of an input spectrogram, with modulation factors conditioned on the specific source and target speaker pairing. They utilize a convolutional layer followed by a modified CIN layer [46] and a gated linear unit (another kind of non-linear activation function) [48].

The modified CIN layer performs the following operation on an input feature vector $\mathbf{f}$:

$$\mathrm{CIN}(\mathbf{f}, \gamma, \beta) = \gamma \left(\frac{\mathbf{f} - \mu(\mathbf{f})}{\sigma(\mathbf{f})} \right) + \beta \tag{3.5}$$

where $\mu(\mathbf{f})$ and $\sigma(\mathbf{f})$ are respectively the scalar mean and standard deviation of vector $\mathbf{f}$, while scalars γ and β are obtained from two embedding vectors associated with the one-hot source-target speaker vector, as depicted in Figure 3.3. The above relation is computed separately for each channel when the input feature contains multiple channels.

3.3. Speaker embedding network

With StarGAN-VC2 and previous models mentioned, the speaker information was presented as either a one-hot embedding, implicitly built into the one-to-one model, or encoded using an embedding layer. All of these methods do not allow generating embeddings for unseen speakers – preventing any of the previous models in chapter from achieving zero-shot VC. AutoVC (discussed next) changed this by introducing a speaker embedding network as described in Section 2.4.5.

As mentioned in Section 2.4.5, the network is trained using a GE2E loss to convert an arbitrary length input Mel-spectrogram X into an embedding vector $\mathbf{s}$ that should be as similar as possible for utterances spoken by the same speaker, and far apart from embeddings of utterances spoken by other speakers. Thus, for use in the AutoVC and StarGAN-ZSVC models described next, we design a speaker embedding network similar to the original GE2E model of [42].

Formally, we define our speaker embedding model $E(X)$ as a 2-layer stacked GRU network. The data flows are the same as that of the stacked LSTM of Figure 2.5, except using GRU layers instead of LSTM layers, and with only 2 such layers instead of 3. Each layer has $h = 768$ hidden units – in line with the number of hidden units used by the speaker embedding network of AutoVC [9] – with a 0.3 dropout between layers which we found to give best results on the validation set. The final linear layer is made to project the final hidden state to $d = 256$ dimensions.

Recall this model is intended to generate speaker embeddings and is not a VC system on its own – it can simply act as the 'speaker information extraction' block of Figure 2.1. And the first model to recognize the usefulness of such an embedding network and implement it in a VC system was AutoVC, discussed next.

3.4. AutoVC

Zero-shot voice conversion was first introduced in 2019 with the AutoVC model [9], which remains one of only a handful of models that can perform zero-shot conversion. AutoVC uses an autoencoder with a specially designed bottleneck layer to force the network's encoder to only retain linguistic content in its encoded latent representation – represented as a sequence of vector similar to a Mel-spectrogram. AutoVC then uses a separate recurrent speaker embedding model as described in the previous section to extract a speaker embedding $\mathbf{s}$ from an input spectrogram. This embedding $\mathbf{s}$ is then used to supply the missing speaker identity information to the decoder which, together with the linguistic content from the encoder, allows the decoder to synthesize an output spectrogram for an unseen speaker.

3.4.1. Architecture

The original authors give a lengthy derivation and motivation for the layer size requirements to ensure that the encoder output only contains speaker-independent linguistic information. They also utilize some additional special DNN layers like batch normalization and a special downsample/upsample operation. Since the goals of including this model in analysis is to assess the practical usefulness of a state-of-the-art zero-shot VC technique, we treat the above two parts of the model as outside the scope of this project, and purely focus on its broad structure and performance in comparison to the other models.

Concretely, the architecture of the autoencoder is shown in Figure 3.4. The encoder and decoder consists of convolutional and LSTM recurrent layers which are carefully designed to ensure that no speaker identity information is present in the encoder output. The original AutoVC used a 2-layer stacked LSTM network for the speaker embedding model [9], while we opt for the 2-layer stacked GRU model described in Section 3.3 because of its improved speed over LSTM layers. In Figure 3.4 the outputs of LSTM (or bidirectional LSTM) layers are taken as the sequence of hidden state vectors from the final layer, concatenated

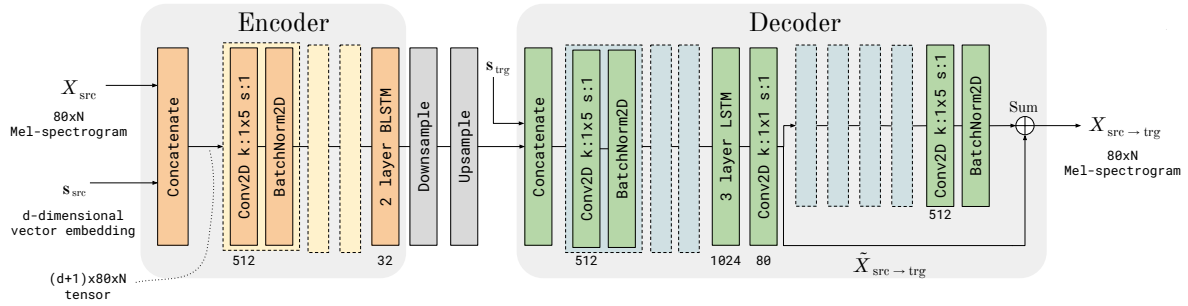


Figure 3.4: Architecture of the AutoVC model [9]. The encoder and decoder parts of the autoencoder are grouped as shown. Within layers, `k` and `s` represent kernel size and stride (for convolutions). A number alongside a layer indicates the number of output channels (for convolutions), output units (for linear layers), or hidden units (for LSTM layers). Groups of layers sharing identical structure are indicated by boxes of unique color with dashed outlines. BLSTM and BatchNorm indicate bidirectional LSTM and batch normalization layers, respectively.

to form a 2D image to feed to downstream convolution layers. The extensive use of LSTM layers causes the model to be fairly slow, as LSTM layers operate sequentially.

The first layer in the encoder concatenates the d-dimensional speaker embedding $\mathbf{s}_{\text{src}} = E(X_{\text{src}})$ to the $80 \times N$ (recall we use $m = 80$ as the number of Mel-frequency bins) input Mel-spectrogram by broadcasting the vector across the width and height of X_{src}. I.e. $\mathbf{s}_{\text{src}}$ is repeated along the width and height dimension to form a $d \times 80 \times N$ dimensional tensor. This vector is then concatenated with X_{src} to form a $(d+1) \times 80 \times N$ tensor to use for further convolution layers.

The `Upsample` and `Downsample` layers perform a special kind of sampling operation on the input 2D image tensor (or equivalently a sequence of hidden states) to produce an output vector sequence which is concatenated with the target speaker embedding vector $\mathbf{s}_{\text{trg}}$ in the same way as for the first layer – by broadcasting the speaker embedding along the width and height dimensions of the `Upsample` layer's output tensor.

In the decoder an intermediary value is labeled $\tilde{X}_{\text{src}\to\text{trg}}$. This is a value used in training, however at inference it remains unused and the converted spectrogram is $X_{\text{src}\to\text{trg}}$ as usual.

The original model utilized a WaveNet vocoder, yielding a very high quality but with extremely slow inference times (roughly 10-20 seconds to convert 1 second of audio). This made the model impractical in many applications, and is one of the reasons why we chose to standardize on using the WaveGlow vocoder in our implementation (as with the other models). The precise model definition in software is modified from the original author's source code [9].

3.4.2. Loss function

As with most autoencoders, during training the model is made to reconstruct its input by using the same source embedding $\mathbf{s}_{\text{src}}$ in the decoder to attempt to reconstruct the input $X_{\text{src}\to\text{src}}$. The model is thus only trained to reconstruct Mel-spectrograms by attempting to convert from speaker A to speaker A and ensuring the output is similar to the input. An additional term is added to ensure that the speaker identity is closely maintained in the reconstructed output.

More formally, the full encoder-decoder DNN model is trained to minimize three terms. The first term is a square L_2 reconstruction loss between the decoder output spectrogram

$X_{\text{src}\to\text{src}}$ and input spectrogram X_{src}, with the source speaker's encoding (from the speaker encoder) provided to both the encoder and decoder. The second term is the square L_2 reconstruction loss between the intermediary decoder term $\tilde{X}_{\text{src}\to\text{src}}$ and the input X_{src}. Adding this term gives a stronger gradient signal to the earlier layers and stabilizes training [9].

The third term is an L_1 loss between the speaker embedding of the decoder output $E(X_{\text{src}\to\text{src}})$ and the original speaker embedding $\mathbf{s}_{\text{src}} = E(X_{\text{src}})$. The full loss function used to optimize the parameters θ of the autoencoder model is thus:

$$\mathcal{L} = ||X_{\text{src}\to\text{src}} - X_{\text{src}}||_2^2 + \lambda_1||\tilde{X}_{\text{src}\to\text{src}} - X_{\text{src}}||_2^2 + \lambda_2||E(X_{\text{src}\to\text{src}}) - \mathbf{s}_{\text{src}}||_1 \tag{3.6}$$

where λ_1 and λ_2 are scalar constants to balance the importance of the different loss terms in training. In both the original paper and in this project, both values are set at 1 which we found to yield good results.

The speaker embedding model is assumed to be *pre-trained* and is not optimized along with autoencoder model. I.e. the 2-layer GRU speaker embedding model parameters are **not** treated as part of the AutoVC encoder-decoder parameters θ. In the original paper, the speaker embedding is trained on a very large (> 10 hours) corpus from several datasets [9]. For our experiments we also pre-train the encoder on a large set of external datasets.

Finally, note that as with StarGAN-VC and StarGAN-VC2, a corresponding parallel target utterance X_{trg} does not appear in any of the loss terms, allowing AutoVC to be trained with non-parallel data. Zero-shot inference is performed by using the speaker embedding model E to obtain embeddings for new utterances from unseen speakers, which is then provided to the decoder instead of the embedding corresponding to the source speaker, causing the decoder to return a converted output. We use this same idea of using an encoding network to obtain embeddings for unseen speakers in our new GAN-based approach, which we describe next.

3.5. A new approach: StarGAN-ZSVC

We use these building blocks for our new zero-shot approach. Concretely, the one-hot speaker vectors in StarGAN-VC2 are replaced with continuous embedding vectors obtained from a separate speaker embedding network (which can be applied to arbitrary speakers) as utilized by the AutoVC model. The motivation for using a speaker embedding network should now be straightforward: it enables a model to perform zero-shot VC, which is one of the requirements for the VC model to be practical.

However, lets make this motivation more concrete. While StarGAN-VC and StarGAN-VC2 allows training with non-parallel data and runs sufficiently fast, it is limited in its ability to only perform voice conversion for speakers seen during training: while parallel X_{src} and X_{trg} utterance pairs are not required, the model can only synthesize speech for target speaker identities (specified as one-hot vectors) seen during training. This could preclude the use of these models in many practical situations where zero-shot conversion is required between unseen speakers. Conversely, AutoVC allows for such zero-shot prediction and is trained on non-parallel data, but the original model uses a slow WaveNet vocoder, includes several recurrent layers which significantly slow down its speed, and its performance suffers when trained on very little data.

Combining the strengths of both of these methods, we propose the *StarGAN zero-shot*

voice conversion model – StarGAN-ZSVC. In the following sections we give the motivation for the various parts of the architecture needed to overcome each practical requirement defined in Chapter 1. In the following chapter we will compare our new model to the existing models from the literature previously described.

3.5.1. Overcoming the zero-shot barrier

To achieve voice conversion between multiple speakers, the original StarGAN-VC2 model creates an explicit embedding vector for each source-target speaker pairing, which is incorporated as part of the generator G and discriminator D. This requires that each source-target speaker pairing is seen during training so that the corresponding embedding exists and has been trained – prohibiting zero-shot voice conversion. To overcome this hurdle, we instead infer separate source and target speaker embeddings, $\mathbf{s}_{\text{src}}$ and $\mathbf{s}_{\text{trg}}$, using a speaker embedding network E as defined in Section 3.3.

The full StarGAN-ZSVC framework is shown in Figure 3.5 – note how each block roughly maps to one of the blocks of the abstract VC system in Figure 2.1. Utterances from unseen speakers (i.e. X_{src} and Y_{trg}) are fed to the speaker encoder E, yielding embeddings for these new speakers, which are then used to condition G and D, thereby enabling zero-shot conversion. The generator uses these embeddings to produce a converted Mel-spectrogram $X_{\text{src}\to\text{trg}}$ from a given source utterance's Mel-spectrogram X_{src}. And as mentioned at the start of this chapter, the model uses NVIDIA's WaveGlow [13] vocoder, which does not require any speaker information for vocoding and thus readily allows for zero-shot conversion.

3.5.2. Overcoming the speed barrier

The speed of the full voice conversion system during inference is bounded by (a) the speed of the generator G (recall the discriminator D is only used in training); (b) the speed of converting the utterance between time and frequency domains, consisting of the initial conversion from time-domain waveform to Mel-spectrogram and the speed of the vocoder;

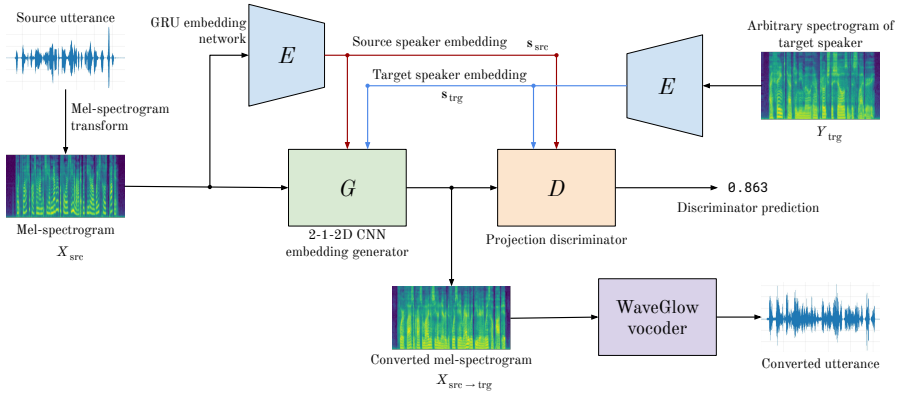


Figure 3.5: StarGAN-ZSVC system diagram. The speaker encoder network E and the WaveGlow vocoder are pretrained on large speech corpora, while the generator G and discriminator D are trained on a 9-minute subset of the VCC dataset. During inference, arbitrary utterances for the source and target speaker are used to obtain source and target speaker embeddings, $\mathbf{s}_{\text{src}}$ and $\mathbf{s}_{\text{trg}}$, which are used by both G and D to condition their outputs. Y_{trg} is the Mel-spectrogram of an arbitrary utterance from the target speaker.

and (c) the speed of the speaker encoder E. To ensure that the speed of the full system is at least real-time, each subsystem needs to be sufficiently faster than real-time.

(a) Generator speed.

For the generator G to be sufficiently fast, we design it to roughly follow StarGAN-VC2's architecture, only including convolution, linear, normalization, and upscaling layers as opposed to models using slower LSTM recurrent layer like those used in AutoVC [9]. By ensuring that the majority of layers are convolutions, we obtain significantly better-than real-time speeds for the generator – disussed further in Chapter 5.

(b) Vocoder and Mel-spectrogram speed.

Following from the background provided in Section 2.4.4, we opt for a reasonable middle-ground choice with the WaveGlow vocoder, which does have recurrent connections but does not use any recurrent layers with dense multiplications such as LSTM or GRU layers. We specifically use a pretrained WaveGlow network, as provided with the original paper [13]. And as mentioned in Chapter 2, vocoding is a wide field in its own right, and we simply use the vocoder model trained by the original authors and assign the details of the vocoder model as outside the scope of this project.

Furthermore, the speed of the Mel-spectrogram transformation for the input audio is well faster than real-time due to the efficient nature of the fast Fourier transform and the efficient nature of performing dot products with the Mel-frequency filters.

This choice of using the WaveGlow vocoder and Mel-spectrogram for the feature extraction and conversion is identical for all the models assessed (except for StarGAN-VC2, which specifically requires the WORLD vocoder) and is made using the same motivations in terms of speed and common use in the VC literature.

(c) Speaker encoder speed.

The majority of research efforts into obtaining speaker embeddings involve models using slower recurrent layers, often making these encoder networks the bottleneck. We also make use of a recurrent stacked-GRU network as our speaker embedding network E as per Section 3.3. However, we only need to obtain a single speaker embedding to perform any number of conversions involving that speaker. We therefore treat this as a preprocessing step where we apply E to a few arbitrary utterances from the target and source speakers, averaging the results to obtain target and source speaker embeddings, and use those same embeddings for all subsequent conversions.

And recall from the design of the speaker embedding network in Section 2.4.5 and 3.3, the speaker embeddings are $d = 256$ long vectors of unit length. If we were to use StarGAN-ZSVC downstream for data augmentation (where we want speech from novel speakers), we could then simply sample random unit-length vectors of this dimensionality to use with the generator. This way, each new batch of training data for downstream applications would transform the utterances into ones spoken by new, never before seen speakers – hopefully significantly improving the performance of these downstream applications.

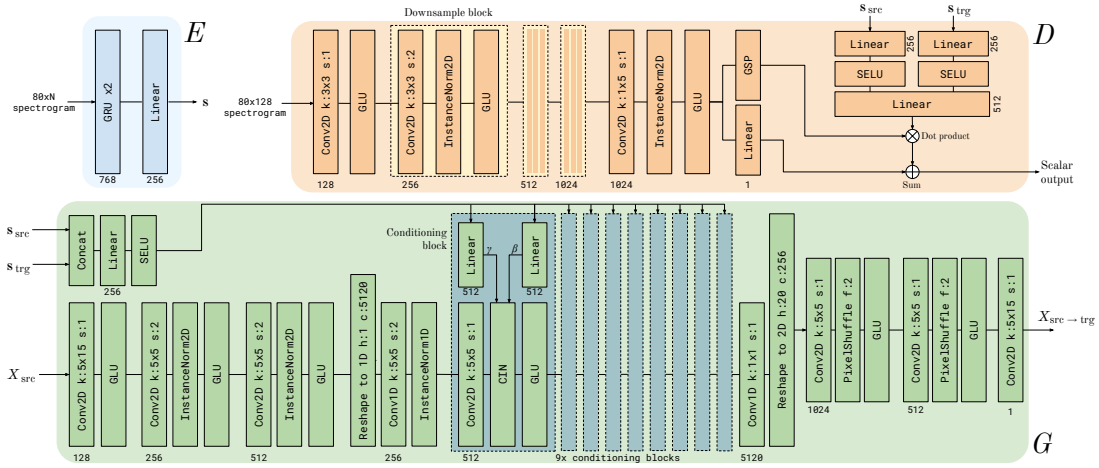


Figure 3.6: StarGAN-ZSVC network architecture. The speaker encoder E is the recurrent network defined in Section 3.3, while the generator G and discriminator D are modified versions from the original StarGAN-VC2 architecture. Within layers, `k` and `s` represent kernel size and stride (for convolutions), `f` is the scaling factor (for pixel shuffle), and `h` and `c` are the height and channels of the output (for reshape layers). A number alongside a layer indicates the number of output channels (for convolutions), or output units (for linear and GRU layers). GLU layers split the input tensor in half along the *channels* dimension. GSP, GLU, and SELU indicate global sum pooling, gated linear units, and scaled exponential linear units, respectively.

3.5.3. Architecture

With the previous considerations in mind, we design the generator G, discriminator D, and encoder network E, as shown in Figure 3.6. The generator and discriminator are adapted from StarGAN-VC2 [8], while the speaker encoder is adapted from the original model proposed for speaker identification [42] as defined in Section 3.3. For G, we use the 2-1-2D generator from StarGAN-VC2 with a modified central set of layers, denoted by the *Conditional Block* in the figure.

These conditional blocks are intended to provide the network with a way to modulate the channels of an input spectrogram, with modulation factors conditioned on the specific source and target speaker pairing. They utilize a convolutional layer followed by the same CIN layer used by StarGAN-VC2 [46] and a gated linear unit [48]. However now, instead of using an embedding layer to convert the one-hot encoding of the source-target speaker pair into the CIN modulation constants γ and β as in StarGAN-VC2, we instead directly concatenate the d-dimensional source and target speaker embeddings and feed the resulting $2d$-dimensional vector through several linear layers and non-linearities to map them to γ and β in each conditioning block as shown in Figure 3.6.

For the discriminator, the source and target speaker embeddings are also fed through several linear layers and activation functions to multiply with the pooled output of D's main branch (as opposed to the embedding layers used in StarGAN-VC2).

The generator and discriminator layer sizes are also slightly changed from StarGAN-VC2 to accommodate $m = 80$ dimensional Mel-spectrograms instead of the 36-dimensional MFCCs used by the WORLD vocoder in StarGAN-VC2. And, as in StarGAN-VC2, the discriminator of StarGAN-ZSVC requires a fixed 128-frame Mel-spectrogram input due to the dot product operation in Figure 3.6. However, the generator G and speaker encoder E can still take in utterances of arbitrary length.

3.5.4. Loss function

StarGAN-ZSVC follows the same training procedure as StarGAN-VC2, using precisely the same loss terms and formulation as given in Section 3.2. The motivation for the loss is the same as the general motivation behind the original StarGAN described in Section 2.3.6 and the StarGAN-VC2 specific terms introduced in Section 3.2. In our experiments, using the same loss function and coefficients λ as for StarGAN-VC2 yielded good results.

3.6. Summary

In Section 3.1, we defined the structure of three baseline models which can only perform one-to-one VC, while in Sections 3.2 and 3.4 we defined the structure of two recent state-of-the-art methods for many-to-many VC. Then, combining the strengths of StarGAN-VC2 and AutoVC we defined our new model – StarGAN-ZSVC in Section 3.5.

In addition to outlining the core mapping function of each model, we also defined the speaker embedding model used by the AutoVC and StarGAN-ZSVC systems in Section 3.3. In the next chapter, we will concretely define how each model is defined and trained in software, toward the ultimate goal of assessing how well each one satisfies the four practicality requirements of Chapter 1.

Chapter 4

Experimental Setup

In this chapter we define how we analyze and evaluate each model described in the previous chapter. In particular, we start with a description of the dataset and how it is used. We then continue to give a description of the software and tooling used in the training and evaluation of each model, followed by a concrete specification of the vocoding and feature extraction procedures used. Finally, we give an outline of training framework for each model, and a definition of the various metrics we use to compare them in the results chapter.

4.1. Dataset

4.1.1. VC models

We compare StarGAN-ZSVC to the other voice conversion models using the voice conversion challenge (VCC) 2018 dataset [1], which contains parallel recordings (utterances) of native English speakers from the United States.

The VCC 2018 dataset was recorded from 8 speakers, each speaking 81 utterances from the same transcript. 4 speakers are used for training and 4 for testing. To emulate a low-resource setting (recall good performance in low-resource settings is one of the practicality requirements), we use a 9-minute subset of the VCC 2018 training dataset for the many-to-many models. This corresponds to 90% of the utterances from two female (F) and two male (M) speakers (SF1, SF2, SM1, and SM2). This setup is in line with existing evaluations on VCC 2018 [8], allows for all combinations of inter- and intra-gender conversions, and allows for zero-shot evaluation on the 4 remaining unseen speakers.

In contrast to many-to-many models, the baseline models of Section 3.1 only allow for one-to-one conversions, i.e. they are trained on parallel data and can only convert from seen speaker A to seen speaker B. We therefore train the baseline models on a single

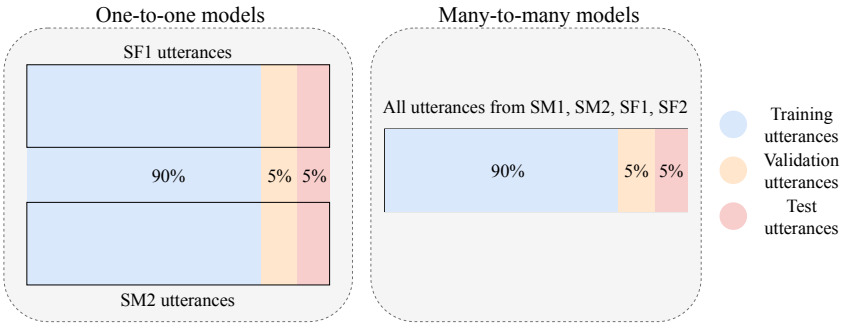


Figure 4.1: VCC 2018 training-validation-test splits. One-to-one models utilize 90% of parallel utterance *pairs* between SF1 and SM2, while many-to-many models utilize 90% of all *individual* utterances from SF1, SF2, SM1, SM2 for training. The precise split is made using Fastai's `RandomSplitter` function using a random seed of 6 [49].

source-target speaker mapping (from SF1 to SM2), using 90% of the parallel training utterances for this speaker pair. Figure 4.1 shows the training-validation-test split for the two classes of models.

Importantly, recall from the loss definitions of the many-to-many models (StarGAN-VC2, AutoVC, StarGAN-ZSVC) in Chapter 3 that the parallel target utterance X_{trg} is not present – i.e. the models can be trained with non-parallel data. However, the one-to-one baseline models do require parallel data, and thus the dataset splits are different in Figure 4.1 where each sample for the one-to-one models is a parallel pair of utterances from SF1 and SM2. Meanwhile, each sample for the many-to-many models is only an individual utterance from a speaker.

This makes comparison difficult since the precise utterances the one-to-one and many-to-many models are trained on are different, however we try overcome this in the next chapter by providing two sets of results for the many-to-many models in seen-to-seen conversion: one where they are evaluated on the test utterances of the one-to-one models (which might have been seen during training), and another set of evaluations on all utterances from the many-to-many speaker's test dataset (covering all 4 seen speakers).

4.1.2. Speaker embedding model

The speaker embedding model of Section 3.3 is pretrained on a large external corpus. Namely, it is trained on 90% of the utterances from a combined dataset consisting of the VCTK [50], VCC 2018 [1], LibriSpeech [51], and the English CommonVoice 2020-06 datasets[1]. This data amounts to over 30 hours of training data and is decidedly *not* low-resource. However, in a practical situation it can be trained on a massive external corpus and still work well for out of domain speakers – as shown in the original work [42] and our experiments. In fact, in our experiments we found that after training it on only the combined English dataset above, it was still able to effectively separate embeddings from Afrikaans utterances from native Afrikaans speakers in the NCHLT Afrikaans dataset [52].

The embedding model is trained with the Adam optimizer [21] for 8 epochs with 8 speakers per batch, and 6 utterances per speaker in each batch following the loss function outlined in Section 2.4.5. We start with a learning rate of 4×10^{-4} and adjust it downwards each epoch to 3×10^{-7} in the final epoch. We use a similar training process defined in Section 4.4 adapted for the speaker identification task appropriate for the GE2E model. Finally, when training all the other VC models, embeddings for the VCC 2018 speakers are precomputed by taking the average embedding over 4 arbitrary utterances for each speaker.

4.2. Software and tooling

All experiments are performed in the Python programming language using Jupyter notebooks for ease of experimentation. Primarily three large software packages are used in the training and evaluation of each model – the Pytorch software package for various neural network related functionality; the librosa audio library for audio manipulation and feature extraction; and the Fastai library for various data loading and training utilities. Detailed descriptions and motivations for the use of each software technology is given in Appendix D.

[1] Available under a CC-0 license at https://commonvoice.mozilla.org/en/datasets

4.3. Vocoder and feature extraction

As a first step, we use the librosa library to resample all audio in the dataset to a standard 22.05kHz sampling rate. In our experiments we use log Mel-scale spectrograms (often just called Mel-spectrograms), with one exception. Concretely, the Mel-scale spectrograms use $m = 80$ Mel-frequency bins and are clamped such that any value below 10^{-5} is fixed to 10^{-5}. The STFT used to construct the spectrograms have a window and hop length of 1024 and 256 samples, respectively. The natural logarithm is then taken to obtain the Mel-spectrogram.

Some of the models using these Mel-spectrograms further scale each Mel-spectrogram to roughly be in the range $(-1, 1)$ by assuming the minimum Mel-spectrogram value to be -11.52 and scaling the mean and standard deviation by half this value. I.e. given a value from a log Mel-scale spectrogram z, we scale it: $\frac{z+5.76}{5.76}$. The AutoVC and one-to-one models perform this scaling proces, while the StarGAN-VC2 and StarGAN-ZSVC do not. We found this setup to give the best results on the validation set of each model.

The vocoder we opt for is the WaveGlow vocoder by NVIDIA due to its reasonably high performance while remaining sufficiently fast. The same WaveGlow vocoder is used to produce output waveforms for all networks (with the same exception for StarGAN-VC2) to ensure a fair comparison. The WaveGlow model is pretrained on a large external corpus, as provided by the original paper [13]. However, since most models use Mel-spectrogram inputs and produce Mel-spectrogram outputs, we rather perform all objective comparisons on the spectrograms of each utterance (instead of waveforms), in order to minimize the confounding effect from the vocoder.

Our WaveGlow implementation takes approximately 240 ms to produce one second of vocoded audio (taking a spectrogram as input). For the full voice conversion system to be faster than real-time, this means that the combined inference speed of the remaining sub-networks needs to be well under 700ms/s, or preferably significantly faster if used for data augmentation.

The one exception mentioned earlier is the StarGAN-VC2 model, which is specially designed to only operate on MFCCs using the WORLD feature extractor and vocoder [8]. While all the other models use the Mel-spectrogram and WaveGlow setup defined above, StarGAN-VC2 uses the WORLD vocoder setup described in Section 3.2. Thus for this model we use the WORLD feature extractor and vocoder with a 36-dimensional MFCC. To compare to the other models, we convert the vocoded waveform outputs of StarGAN-VC2 back into log Mel-scale spectrograms. The WORLD vocoder is somewhat faster to the WaveGlow vocoder, taking roughly 10 ms to vocode a second of audio.

4.4. Training framework

Training is performed using the Fastai framework using the standard method for training DNNs outlined in Chapter 2, where the DNN optimized is the one that performs the central mapping function of the full VC system – i.e. we assume the speaker embedding network and vocoder models to be pretrained as specified in Section 4.1.2 & 4.3. It consists of three main processes: loading the data, defining the model and loss function, and running the training loop. Each step differs slightly between the one-to-one and many-to-many models, however the general procedure is given in the following subsections. Each model is trained by running various Python cells inside a Jupyter notebook environment.

4.4.1. Data loading

This step occurs once at the beginning of training, and its objective is to yield a Python data loader object that can be repeatedly called to return minibatches of training data for a given model. Since each utterance only needs to be converted to Mel-spectrogram (or MFCC for StarGAN-VC2) once, it is done as a preprocessing step before any training commences using the method defined in the previous section. Each Mel-spectrogram is then saved to a solid-state drive (SSD) and loaded as needed during training.

We implement the Mel-spectrogram transform as a modified version of the Mel-spectrogram transform provided by librosa and another implementation by the authors of the WaveGlow vocoder [13, 53], adapted to work for the VCC 2018 dataset with our 22.05kHz sampling rate.

To construct the desired data loader, the Fastai Data Blocks software interface is used [49]. In it, we define a dataset object which can be invoked with a file url to return a single sample of training data. Then, hyperparameters for the dataloader are set up (such as batch size, source file paths, train-valid-test splits) and transforms defined for collating several samples together into a minibatch.

The primary transform applied is that of random segment sampling. Concretely, during training, for each batch we randomly sample a k-frame sequence from each Mel-spectrogram, where k is randomly sampled from multiples of 32 between 96 to 320 (inclusive). This is done for all models to make them robust to utterance length. An exception is made for StarGAN-ZSVC and StarGAN-VC2, which both use a discriminator that requires fixed-size inputs. This leads to slightly worse performance for the GAN-based systems for long or silence-padded sequences. Thus, when performing comparisons, we will only compare on non-silent frames of the target spectrogram (discussed further in Section 4.5).

This full data loading process is summarized in Algorithm 4.2, which shows the process called to construct a single batch of training data. Fastai then uses this procedure for obtaining a single batch to construct a full data loader object which automatically performs all the other standard tasks required by a dataloader. For example, it automatically shuffles the order of training data items once they have been exhausted (i.e. once an epoch has finished) and keeps track of the train-validation-test splits.

4.4.2. Model and loss definition

Once the data object has been defined, the second step to produce a trained model is to instantiate a specific VC model instance – i.e. to create the DNN architecture of the particular VC mapping model under consideration in GPU memory. This is done also using the Pytorch and Fastai libraries in a straightforward process following the model and loss definitions of the previous chapter.

First, an instance of the DNN model (defined in a Python script) is created in the Jupyter notebook using the specific hyperparameters associated with each model in Chapter 3. Second, the loss function for the specific model is defined as a python function. Third, the model instance, loss function, and data loader object are used to instantiate a Fastai `Learner` object. This `Learner` has all the required knowledge to perform the training loop, and contains other utilities for saving models and logging training statistics.

Algorithm 4.2: Procedure to load a batch of training data for a model, given a list of b filenames where b is the batch size. The model's specific speaker representation and spectrogram form is used where appropriate (e.g. one-hot vs speaker embedding, or Mel-spectrogram vs MFCC).

```
Initialize empty arrays of source and target spectrograms
Initialize empty arrays of source and target speaker representations
Sample k from multiples of 32 between 96 and 320 (inclusive).
for filename in filenames do
    Sample target speaker from all training speakers and load its representation.
    Load filename and parallel target speaker spectrogram into memory.
    Sample k frames from both spectrograms at a random start frame.
    Scale spectrograms according to Section 4.3.
    Parse source speaker from filename and load its representation.
    Append source & target spectrograms & speaker representations to respective arrays.
end for
Stack all source & target spectrograms together into single tensors.
Stack all source & target speaker representations into single tensors.
Move all tensors to GPU memory.
if model is one-to-one then
    return (source spectrogram tensor, target spectrogram tensor)
else
    return (source spectrogram tensor, source speaker tensor, target speaker tensor)
end if
```

4.4.3. Training loop

We now use the `Learner` object to train each VC model. Two similar processes are followed for the non-GAN methods and the GAN-based methods (StarGAN-VC2 and StarGAN-ZSVC). The precise details of the two training processes are detailed in Appendix D.2. Intuitively, for the non-GAN methods we follow the process outlined in Section 2.3.2 using the Adam optimizer with typical settings and a learning rate determined by a special technique developed by Smith [54]. Similarly, for GAN-based methods follow the procedure outlined in Section 2.3.6, adapted for the StarGAN loss function. An Adam optimizer is again used but with a reduced momentum. Several special techniques are employed to ensure the GAN training remains stable. These techniques, along with more detail about the precise steps in the training loop are given in Appendix D.2.

4.4.4. Failed efforts

To get to the precise training setup defined in the previous few sections required a significant amount of investigation and trail and error of various techniques. Ultimately the final training configuration used was the outcome of a long series of experiments with different training techniques and settings. A summarized list of the various techniques used to try train each model, along with which ones were successful and which failed along with our hypothesis for such success/failure is given in Appendix E.

4.5. Objective evaluation

We compare converted output spectrograms to their ground-truth target spectrograms on the test dataset using several objective metrics. To account for different speaking rates, we first undo the scaling done on the Mel-spectrograms as described in Section 4.3 and then use dynamic time warping (DTW) to align the converted spectrogram to the target

spectrogram, and then perform comparisons over non-silent regions of the *target* utterance. Non-silent regions are defined as those 80-dimensional spectrogram frames where the mean vector element value is greater than -10dB (or simply -10 if comparing values of the log Mel-scale spectrogram).

For each trained model, the process to obtain objective comparisons is as follows: (i) converted spectrograms are obtained on the test set for each model, then (ii) any scaling done for spectrograms of a particular model as per Section 4.3 is reversed, and finally (iii) the converted spectrograms are aligned to the target spectrograms with DTW and the frames were the target spectrogram is silent are ignored. Any metrics are then computed on these aligned log Mel-scale spectrograms. The objective metrics computed aim to evaluate how well each model performs the VC task in a low resource context (requirement 3 of Section 1.2), or to measure each models speed (requirement 4 of Section 1.2).

For StarGAN-VC2 which produces MFCCs, we use the WORLD vocoder to generate converted waveforms $\mathcal{X}_{\text{src}\to\text{trg}}$ and then perform the Mel-spectrogram transform on them to allow comparison to the spectrograms generated by the other models.

4.5.1. Dynamic time warping

Since a converted utterance might not be spoken at the same rate or with the same pace as the target utterance, we first align the converted Mel-spectrogram $X_{\text{src}\to\text{trg}}$ to the target spectrogram X_{trg} using dynamic time warping (DTW).

The precise details of the technique are beyond the scope of this project, and we simply use it as a practical method to compare converted spectrograms to the ground truth spectrograms. Intuitively, DTW aims to find a warping path (list of mappings of indices between the converted spectrogram and target spectrogram frames) by computing a distance metric between each pair of converted-target spectrogram frames and using these distances to form a so-called 'cumulative distance matrix' [55]. This cumulative distance matrix is then traversed according to a set of rules to find the optimal warping path, which may be non-linear.

We use librosa's DTW implementation [53] to obtain this optimal warping path and then map each frame of the converted spectrogram to its associated frame index of the target spectrogram.

4.5.2. Mean absolute error

The first metric we use to compare $X_{\text{src}\to\text{trg}}$ and X_{trg} is the mean absolute error (MAE) between the two. To calculate the MAE between the two Mel-spectrograms we phrase both as a 2D matrix and take the element-wise absolute difference between the two. We then take the mean absolute difference to arrive at the MAE, only including the Mel-spectrogram frames from non-silent frames.

This gives a simple numerical representation of how far apart the generated spectrogram is from being an ideal conversion, however this comparison is not perfect as multiple valid conversions could exist and the ground truth X_{trg} represents only one such possibility. Such a comparison also penalizes slight pitch differences harshly, however it remains a reasonable first-glance metric of performance.

4.5.3. Cosine distance

To help overcome some of the shortcomings of a pure MAE comparison, we also compute a *cosine similarity* by finding the cosine distance between each 80-dimensional source/target frame pair of the Mel-spectrograms and then computing the mean cosine distance over all non-silent frames (after DTW alignment). This metric, which we denote as $\cos(\theta)$, gives an additional measure of conversion quality.

This measurement is included in comparisons because its shortcomings in measuring the semantic quality of conversions does not completely overlap with that of the MAE measurements. This means that if both MAE and $\cos(\theta)$ metrics are better for a particular model, we can be more confident that it is indeed producing higher quality conversions.

4.5.4. Embedding difference

The previous two metrics primarily looked at the quality of conversion in hopes of measuring how natural the converted spectrogram sounds. However, that is only one aspect of VC. A high performing VC system also needs to ensure that the converted spectrogram sounds as if it were spoken by the target speaker. I.e. the converted utterance must be both realistic human speech and interpreted as being spoken by the desired target speaker.

To address this second aspect of VC performance, we introduce an embedding difference measurement $||\Delta\mathbf{s}||$. This difference is the Euclidean norm of the vector difference between two speaker embeddings. For comparison of VC quality, we compare the speaker embeddings of the converted and target utterance. I.e. $||\Delta\mathbf{s}|| = ||\mathbf{s}_{\text{src}\to\text{trg}} - \mathbf{s}_{\text{trg}}||_2$, where $\mathbf{s}_{\text{src}\to\text{trg}} = E(X_{\text{src}\to\text{trg}})$ using the trained speaker embedding model of Section 4.1.2.

This metric gives us a *smaller value* when the converted utterance's speaker identity is close to the desired target speaker identity, and a *larger value* when the converted utterance sounds like it is spoken by any other speaker. Thus this measurement gives an objective way to measure how close the converted speaker identity matches the target speaker identity and thereby helps profile the performance of a VC model.

4.5.5. Speed

Finally, to measure the speed of a VC model we simply record the time it takes to convert one second of audio. In all measurements of speed, we measure the time taken to convert several different utterances and then divide the total time taken by the total length of the utterances, yielding a measurement of number of seconds required to convert one second of audio. This metric is typically expressed in milliseconds (of converted audio) per second, or ms/s.

Note, however, that this measurement is only of the core VC model and does not include the compute time of the speaker embedding model (since that may be done as a preprocessing step in practical applications), the vocoder (which takes 240ms to vocode one second of audio as discussed in Section 4.3), and feature extraction (which is a near instant FFT combined with a filtering operation taking less than 10ms) steps.

4.6. Subjective evaluation

Noting that many of the objective metrics have substantial inadequacies, we also perform perceptual experiments to evaluate the quality/performance of the conversions for each

model. For these experiments, we recruited 12 proficient English speaking adults to listen to 100 utterances each (totalling 1200 utterances evaluated) and rate their naturalness (how realistic they sound) on a scale from 1 to 5, 1 being 'bad' and 5 being 'excellent'. The 100 utterances are comprised of several subsets of data to assess several different models in different settings (for example, 8 utterances are from AutoVC evaluated when converting from seen speakers to unseen speakers).

The mean score for each utterance grouped by subset is then averaged to yield a mean opinion score (MOS) where the larger the score, the more natural sounding the waveforms are interpreted to be by human listeners. The perceptual experiments were performed using a Flask server running on Google Cloud App Engine and using JourneyApps Oxide to record the responses. These technology choices were chosen for their ease of use and low setup time compared to alternatives. The server ensures that each listener receives the utterances in random order with randomized utterance names to make the comparison as fair and consistent as possible.

We also evaluate the trained GE2E speaker embedding model by visually inspecting how well utterances of different speakers are separated when the embeddings are projected onto a two dimensional image using the t-SNE dimensionality reduction technique [56]. The details of the t-SNE algorithm is beyond the scope of the project, and it is used purely to obtain a reasonable visualization of the $d = 256$ dimensional speaker embeddings $\mathbf{s}$ to allow for visual confirmation that the speaker embedding network is working correctly.

4.7. Summary

With the various models defined in Chapter 3, we proceeded in this chapter to concretely define the setup used to conduct our experiments. In the following chapter we will utilize this setup to conduct several experiments to test how well each model satisfies the practicality requirements of Chapter 1.

In particular, this chapter looked at how the low-resource dataset is configured in Section 4.1. We then proceeded to describe the software implementation details for the dataset and how spectrogram and vocoding process are performed. Finally we looked at how each model is trained in Section 4.4 and then concluded with the various metrics that will be used extensively in the experiments of the next chapter.

Chapter 5

Experiment Results

With the models and experimental setup defined, we now proceed to enumerate the various experiments we ran on each model and the results of such experiments in our quest to profile the extent each model satisfies the four practicality requirements of Chapter 1. Concretely, the ability to perform zero-shot conversion and be trainable with non-parallel data are determined by each individual model architecture as defined in Chapter 3.

Thus this chapter aims to determine to what extent each model satisfies the final two requirements – measuring the VC model's performance on a low resource dataset and measuring its inference speed. The dataset described in Section 4.1 comprises just under 9 minutes of recorded audio – very low-resource context indeed. Since all models in this chapter are trained on this small dataset, all performance metrics computed give an indication of low-resource performance. Furthermore, the speed of each model at inference is independent of the training data, and such speed measurements will be valid in all contexts on the same hardware.

This chapter beings with experiments confirming the efficacy of the speaker embedding model. This is then followed by a set of objective and subjective evaluations of each model. Finally, a performance summary of each model is given.

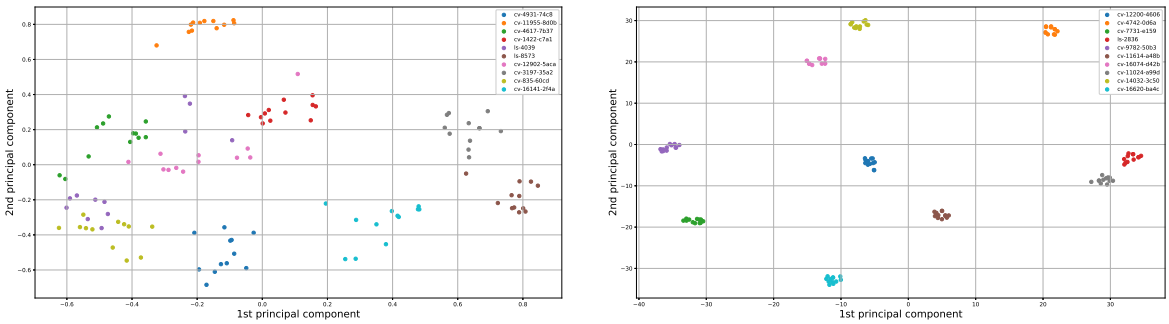
5.1. Speaker embeddings

Before evaluating each VC model, we need to assess whether the speaker embedding model is successful in its task. A low loss on the validation and test set has little semantic meaning – how precisely a lower value for the loss of Section 2.4.5 translates to better embeddings is not clear.

Thus instead we opt for a visual confirmation that the trained model is performing successfully. Recall from Section 4.1 that the speaker embedding model is trained on a very large combined corpus over several datasets. So, to visually assess its performance we obtain the speaker embeddings for 12 utterances for each of 8 unseen speakers from the test set of speakers. We then project these 12×8 embeddings onto a 2-dimensional plane and plot the projected embedding of each utterance on a scatter plot.

The result of such a projection is shown in Figure 5.1 which plots the projections from utterances in the first two batches taken from the (shuffled) test set. Two different projection techniques are shown for two different batches of test data to give multiple perspectives. First, principal component analysis (PCA) is used to project the embeddings onto the first two principal components. This embedding technique is not particularly sophisticated and, as we can observe from the figure, shows some overlap between utterances by different speakers. However, overall from Figure 5.1a we observe that embeddings from utterances by different speakers are separated very well and that embeddings of utterances from the same speaker are close to one another.

The t-SNE projection in Figure 5.1b strongly supports this. It shows that the speaker embedding model is performing very well – strongly grouping embeddings of different



(a) PCA projection **(b)** t-SNE projection

Figure 5.1: 2D projections of a batch of utterance embeddings from several speakers unseen during the training of the speaker embedding model. Each utterance embedding is colored by speaker. Two projections are given: (a) the PCA projection, and (b) the t-SNE projection.

utterances from the same speaker and doing the opposite for utterances by different speakers. For the t-SNE projection we use `sklearn`'s implementation with a perplexity of 10, keeping all other hyperparameters as their default values [57].

Recall that both projections are on utterances by speakers never seen during training of the embedding model. And, since the embedding model seems to exhibit very good performance and behavior in generating speaker embeddings, we conclude that it will be successful and useful in obtaining speaker embeddings for use in zero-shot VC. Finally, we measured the time taken to obtain one speaker embedding given an input utterance of 2.1 seconds. Such an embedding required (on average) 13.1 ms to compute. This is sufficiently fast for nearly all VC applications, especially considering that obtaining the speaker embedding may be done once as a preprocessing step in most VC applications.

5.2. Objective evaluation

We perform two sets of experiments. First we perform an evaluation on seen speakers, where we compare StarGAN-ZSVC to all other models to obtain an indication of both speed and performance in the traditional seen-to-seen VC task. We then compare StarGAN-ZSVC with AutoVC for zero-shot voice conversion (since they are the only two models capable of zero-shot conversion), looking at both the output and cyclic reconstruction error.

5.2.1. Seen-to-seen conversion

In the first set of comparisons, we evaluate the performance metrics defined in Section 4.5 for test utterances where other utterances from both the source and target speaker have been seen during training. I.e., while the models have not been trained on these exact test utterances, they have seen the speakers during training. There is, however, a problem in directly comparing the one-to-one models (traditional baselines) to the many-to-many models (AutoVC, StarGAN-VC2, and StarGAN-ZSVC). The one-to-one models are trained on parallel data, always taking in utterances from one speaker as input (VCC2SF1 in our case) and always producing output from a different target speaker (VCCSM2).

In contrast, the many-to-many models are trained without access to parallel data, taking in input utterances from several speakers (4 speakers, including VCC2SF1 and VCCSM2 in our case, as explained in Section 4.1). This means that the one-to-one and

many-to-many models observe very different amounts of data. Moreover, while the data for both the one-to-one and many-to-many models are divided into the train-test-validation split of Section 4.1, the same exact splits aren't used in both setups; this is because the former requires parallel utterances, and the split is therefore across utterance *pairs* and not individual utterances.

Experiment

To address this, we evaluate the many-to-many models in two settings: on the exact same test utterances as those from the test split of the one-to-one models, as well as on all possible source/target speaker utterance pairs where the source utterance is in the test utterances for the 4 seen training speakers. In the former case, it could happen that the many-to-many model actually observes one of the test utterances during training. Nevertheless, reporting scores for both settings allows for a meaningful comparison. The results of this evaluation on seen speakers are given in Table 5.1. Note that the speed metric only records the speed of the core VC model and does not include the speed of the vocoder or waveform feature extraction.

Discussion

The results indicate that AutoVC appears to be the best in this evaluation on seen speakers. In both settings it has the lowest MAE and $||\Delta\mathbf{s}||$ metrics in addition to having the highest cosine similarity. This means that the generated spectrograms from AutoVC are closer to the true target spectrograms than other models. However, this comes at a computational cost: the linear, StarGAN-VC2, and StarGAN-ZSVC models are much faster than the models relying on recurrent layers like DBLSTM and AutoVC.

In terms of speed, StarGAN-ZSVC and the linear baseline model are the fastest. It is interesting to note that StarGAN-ZSVC is significantly faster than StarGAN-VC2 despite the model architectures appearing so similar. The exact reasoning for this is not entirely clear, however we suspect the reasoning is due to minor architecture optimizations added to StarGAN-ZSVC over StarGAN-VC2.

Table 5.1: Objective evaluation results when converting between speakers where both the source and target speaker are seen during training. For all metrics aside from cosine similarity ($\cos(\theta)$), lower is better. The first many-to-many model entries correspond to evaluations on the one-to-one test utterances, while the starred many-to-many entries correspond to metrics computed when using the test utterances from all seen training speakers for the many-to-many models.

Model	MAE	$\cos(\theta)$	$\|\|\Delta\mathbf{s}\|\|$	Speed (ms/s)
Linear	1.277	0.980	0.863	**0.151**
DBLSTM	1.329	0.982	0.510	12.5
UNet	1.370	0.980	0.550	101
StarGAN-VC2	1.172	0.981	1.035	6.44
AutoVC	**0.993**	**0.987**	**0.257**	11.0
StarGAN-ZSVC	1.092	0.977	0.508	1.88
StarGAN-VC2*	1.180	0.981	0.992	6.44
AutoVC*	**1.000**	**0.987**	**0.321**	11.0
StarGAN-ZSVC*	1.008	0.983	**0.321**	**1.88**

For example, the instance normalization layers of StarGAN-VC2 do not purely use the mean and standard deviation of an incoming tensor, but rather utilize a running average of the mean and standard deviation. These means and standard deviations entail a substantial additional number of parameters for the model and need to be updated whenever the model is called – slowing it down. Other smaller optimizations include removing the superfluous bias term added in convolution layers which are followed by an instance normalization layer (since the instance normalization layer already includes a trainable bias).

Returning to performance, StarGAN-VC2 and StarGAN-ZSVC are fairly close to AutoVC in terms of the metrics. StarGAN-VC2 fails somewhat on the $||\Delta\mathbf{s}||$ metric likely because it never sees the embeddings generated by the embedding network, requiring it to learn its own embeddings. This requirement along with the many-to-many nature of StarGAN-VC2 causes it to seemingly learn how to convert between a 'generic male voice' and a 'generic female voice' as opposed to more finely differentiated speaker identities of SM1, SM2, SF1, SF2. So the speaker embedding difference is large since the speaker embedding network still strongly differentiates embeddings for speakers of the same gender that sound similar (Figure 5.1).

5.2.2. Zero-shot conversion

We now proceed to compare StarGAN-ZSVC with AutoVC in zero-shot settings, since these are the only two models capable of performing zero-shot VC. In these zero-shot experiments we will not only profile the standard speaker A→B conversion, but also the same metrics evaluated on the *cyclic reconstruction* of utterances. That is, we perform the cyclic mapping (as defined in Section 2.3.6) whereby an utterance from a source speaker can be converted to a target speaker and then back to the source speaker. We can use this cyclic reconstruction of the source spectrogram as another source for evaluation, where we compare how well the original spectrogram is reconstructed when performing this cyclical mapping (i.e. we compare $X_{\text{src}\to\text{trg}\to\text{src}}$ to the ground-truth X_{src}).

Experiment

Results for zero-shot conversion are shown in Table 5.2. It includes evaluations on three different experiments, for each possible zero-shot source/target speaker pairing. Concretely, we compute the metrics of Section 4.5 between the converted output and ground-truth target utterance (standard prediction metrics) and between the cyclic mapping and original source utterance (reconstruction metrics) for three cases.

The first case is the seen-to-unseen, where the source speaker has been seen during training but the target speaker is unseen during training. Similarly, a separate set of metrics is computed for conversions from unseen speakers to speakers seen during training (unseen-to-seen) and for pure zero-shot conversion where both the source and target speaker are unseen during training (unseen-to-unseen).

Discussion

The performance for AutoVC and StarGAN-ZSVC are similar on most metrics for the unseen-to-seen case. But for the seen-to-unseen case and the unseen-to-unseen case StarGAN-ZSVC achieves both better prediction and reconstruction scores. Both models,

Table 5.2: Objective evaluation results for zero-shot voice conversion for AutoVC and StarGAN-ZSVC. The *prediction* metrics compare the predicted output to the ground-truth target, while the *reconstruction* metrics compare the cyclic reconstruction $X_{\text{src}\to\text{trg}\to\text{src}}$ with the original source spectrogram. For all metrics asidie from cosine similarity ($\cos(\theta)$), lower is better.

		Prediction			Reconstruction		
Setting	Model	MAE	$\cos(\theta)$	$\|\|\Delta\mathbf{s}\|\|$	MAE	$\cos(\theta)$	$\|\|\Delta\mathbf{s}\|\|$
Seen-to-unseen	AutoVC	1.246	**0.982**	0.742	1.178	0.976	0.392
	StarGAN-ZSVC	**1.030**	**0.982**	**0.705**	**0.197**	**0.997**	**0.124**
Unseen-to-seen	AutoVC	1.014	**0.986**	**0.328**	1.201	0.975	**0.753**
	StarGAN-ZSVC	**0.974**	0.985	0.380	**0.921**	**0.986**	0.760
Unseen-to-unseen	AutoVC	1.238	0.981	0.746	1.340	0.968	0.827
	StarGAN-ZSVC	**1.079**	**0.982**	**0.742**	**0.921**	**0.986**	**0.760**

however, have worse scores on the zero-shot case. This is to be expected, since the zero-shot VC task is harder than the traditional one where the speakers are seen during training.

From the table, AutoVC appears to not be able to generalize well to unseen speakers when trained on little data, with its performance converting *to* unseen speakers greatly reduced. This observation will be repeated in the next section and is one of the failings of AutoVC – its zero-shot performance degrades significantly in a low-resource context. StarGAN-ZSVC does not appear to suffer such degradation in the zero-shot case and outperforms AutoVC in most zero-shot applications on the objective metrics evaluated. This, coupled with its fast inference speed (Table 5.1), enables it to be used efficiently and effectively for downstream data augmentation purposes. Most importantly, StarGAN-ZSVC's retention of good performance in the zero-shot setting indicates that it satisfies the third practicality requirement of Section 1.2 moderately better than AutoVC.

5.3. Subjective evaluation

The objective evaluations performed in the previous section have shortcomings since they not perfectly represent the conversion quality as perceived by humans. Thus we perform a series of subjective evaluation experiments according to the setup defined in Section 4.6. The 1200 utterance evaluations by listeners are comprised of 14 subsets of data which are used to evaluate the performances of different models in different situations. The breakdown of the subsets included in the survey are given in Appendix F.

In addition to the MOS values calculated from the subjective evaluation, we also strongly encourage the reader to both observe sample converted spectrograms (in Appendix F) themselves and listen to converted samples (at https://rf5.github.io/skripsie/about) to further inform their assessment of the relative performance of each model. In the following two sections we give the results of the MOS in two related sets of experiments.

5.3.1. Seen-to-seen conversion

Evaluating the MOS for all ratings from the subsets of Appendix F involving seen-to-seen conversion, we obtain the scores given in Figure 5.2. The figure requires several comments. First, our measured MOS values for AutoVC, StarGAN-VC2, and DBLSTM

are substantially lower than those reported by the original authors [8, 9, 36], despite using similar MOS evaluation procedures.

The reasons for this are different for each work. For AutoVC and DBLSTM, the original work uses substantially more data to train each model. Our work shows their performance on a very low-resource 9-minute training set, hence the lower MOS score. Additionally, both authors leave out substantial detail in their training procedure and hyperparameters which makes re-implementation difficult and imperfect. This problem is exasperated for StarGAN-VC2, where the authors provide no source code and the original paper has an error in the model definition (as described in Section 3.2). These model and training ambiguities of the original model ultimately caused our implementation to have a substantially lower MOS than the original authors.

However, with the MOS values in our evaluation in Figure 5.2 we can see a clear trend. The proposed StarGAN-ZSVC model outperforms all the other models by a substantial margin, indicating it has better low-resource performance than all other models analyzed. AutoVC and StarGAN-VC2 have similar scores which are better than the simple one-to-one baseline models, which is in line with their performance rankings from the literature. The DBLSTM performs slightly better than the UNet and linear model, which performed the worst.

To put the values into context, several raw source utterances and vocoded source utterances were included in the analysis, yielding a MOS of 4.86 and 4.33 respectively. Thus, since all the models (aside from StarGAN-VC2) use the same vocoder, 4.33 represents an upper bound on the MOS for any model. In light of this, the StarGAN-ZSVC MOS of 2.69 yields a marginally convincing (but decidedly not perfect) voice conversion output. Sample converted spectrograms for each model is also included in Appendix F to provide additional perspective. In the Figures of Appendix F we again observe that the converted spectrograms of StarGAN-ZSVC appear more 'natural' (similar in texture to the target utterance) than other models, supporting the conclusions of the MOS values in Figure 5.2.

In summary, the data indicates that (at least on the traditional seen-to-seen VC task) StarGAN-ZSVC satisfies practicality requirement 3 of Section 1.2 better than all other models analyzed.

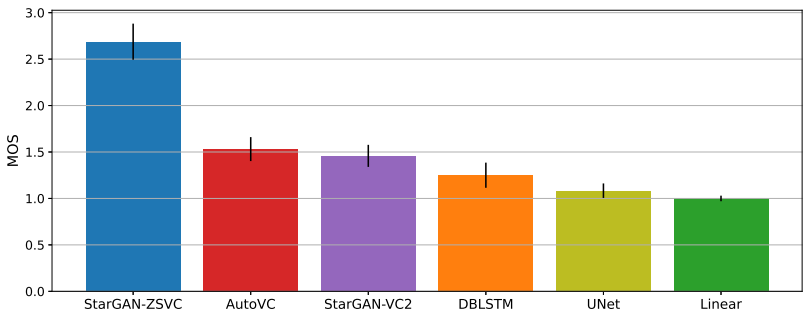


Figure 5.2: Mean opinion score for seen-to-seen conversions from each model with 95% confidence intervals shown, as evaluated from listening tests by 12 proficient English speakers.

5.3.2. Zero-shot conversion

To assess whether the improved performance of StarGAN-ZSVC is retained in the zero-shot setting, and to provide further evidence for the results of Section 5.2.2, we perform a series of zero-shot listener tests. In particular, we find the MOS values for StarGAN-ZSVC and AutoVC (the only models capable of zero-shot inference) in all zero-shot settings according to the subsets outlined in Appendix F. The results are shown in Figure 5.3. The MOS values in this plot are directly comparable to that of Figure 5.2 because they form part of the same listening test as defined in Section 4.6.

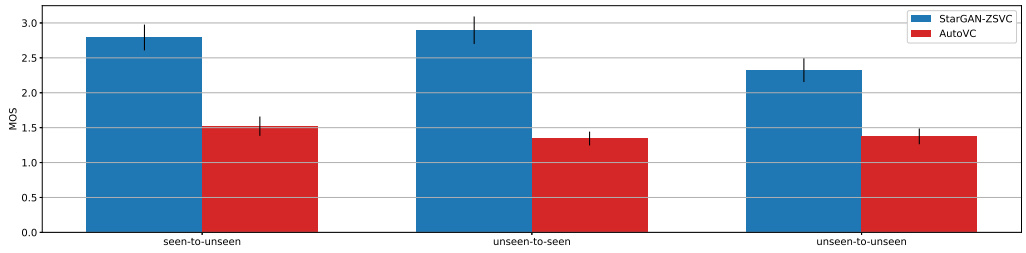


Figure 5.3: Mean opinion score for zero-shot conversions from StarGAN-ZSVC (blue) and AutoVC (red) with 95% confidence intervals shown, as evaluated from listening tests by 12 proficient English speakers.

The results support our previous objective metric evaluation and the seen-to-seen subjective comparison in the previous subjection – StarGAN-ZSVC retains reasonable performance in the zero-shot setting and is moderately better than AutoVC in all zero-shot settings considered. This is supported by the zero-shot converted spectrograms from each model in Appendix F (Figures F.7 & F.8), where we can clearly see that StarGAN-ZSVC's converted spectrograms are structurally more realistic and closer to the target utterance.

In Figure 5.3 we observe that AutoVC takes a marginal hit in performance in all three zero-shot cases (comparing it to its seen-to-seen MOS in Figure 5.2), while StarGAN-ZSVC appears to retain its performance for seen-to-unseen and unseen-to-seen conversion but decreases in performance considerably in the harder unseen-to-unseen zero-shot case. StarGAN-ZSVC's unseen-to-unseen MOS is, however, still higher than the scores of any model in the seen-to-seen VC task of Figure 5.2, indicating that its performance is still considerable even in the hardest zero-shot task.

In summary, these subjective evaluations agree largely with the objective evaluations of the previous section, with StarGAN-ZSVC appearing to perform better than the other models by an even greater margin in the subjective evaluation. Thus we can confidently state that StarGAN-ZSVC better satisfies practicality requirement 3 better than the other models considered.

5.4. Summary

In this chapter we performed several objective and subjective evaluations on the models of Chapter 3 using the experimental setup defined in Chapter 4. We performed two sets of evaluations – an objective evaluation using metrics computed on the converted spectrograms and another set of subjective listening tests. In both cases, it was evident that the designed StarGAN-ZSVC and AutoVC were the top performing models, with StarGAN-ZSVC performing the best in zero-shot conversion.

This chapter primarily evaluated the *speed* of models and their *performance/quality in a low-resource setting* – two of the practicality requirements of Chapter 1. Meanwhile, the model architectures of Chapter 3 determine their limitations in terms of their ability to be *trained with non-parallel* data and their *ability to perform zero-shot VC* – the other two practicality requirements of Chapter 1. Summarizing the results of the experiments and design limitations, we qualitatively describe how well each model satisfies the practicality requirements in Table 5.3.

Table 5.3: Summary of the extent to which each model satisfies the four practicality requirements of Chapter 1, showing a qualitative description of how well each model satisfies each requirement.

Model	Non-parallel trainable	Zero-shot capable	Performance in low-resource setting	Speed
Linear	No	No	Very poor	Very fast
UNet	No	No	Poor	Very slow
DBLSTM	No	No	Marginally poor	Marginally slow
StarGAN-VC2	Yes	No	Moderate	Moderate
AutoVC	Yes	Yes	Moderate	Marginally slow
StarGAN-ZSVC	Yes	Yes	Good	Fast

Chapter 6

Summary and Conclusion

6.1. Problem and objective

In this project we investigated several different techniques for voice conversion and attempted to adapt recent VC systems to work better in practical settings. The aim in performing these two tasks is to help enable the downstream applications of a practical VC system, primarily for its use in data augmentation for other natural language processing tasks or for use in the entertainment and gaming industry. These tasks are not readily performed by existing systems because they fail to satisfy certain requirements needed for practical use in downstream applications. These requirements are defined in Section 1.2, and require that a VC system is fast, trainable with non-parallel data, high performing in low-resource contexts, and is able to perform zero-shot VC.

6.2. Literature and model design

To address these two related tasks, we first provided a background of the relevant VC theory and terminology along with descriptions of related work in the VC literature in Chapter 2. We then proceeded to identify and concretely define five existing models from the literature that covered a range of sophistication and neural network techniques. Upon realizing that none of the existing techniques satisfy all practicality requirements we adapted the StarGAN-VC2 by using a speaker encoder introduced with the AutoVC model to generate speaker embeddings which are used to condition the generator and discriminator network on the desired source and target speakers. The resulting model, StarGAN-ZSVC, can perform zero-shot inference and is trainable with non-parallel data.

6.3. Experiments and comparisons

With concrete specifications of these five models defined, we proceeded to build a series of experiments to assess how well each satisfies the practicality requirements. This is done using a carefully designed dataset, training setup, and feature extraction and vocoding detailed in Chapter 4, whereby each model is trained in a low-resource setting and experiments are defined to assess its performance in a low-resource setting and its inference speed. The other two practicality requirements are determined exclusively by the VC model design.

In a series of objective and subjective experiments comparing StarGAN-ZSVC to the other five models from the literature, we found that no model perfectly satisfied all practicality requirements perfectly. However, in both comparisons in the traditional seen-to-seen VC task and in the zero-shot comparisons to AutoVC, we demonstrated

that StarGAN-ZSVC scores better or similar objective and subjective metrics to all other models considered. AutoVC and StarGAN-VC2 performed worse than StarGAN-ZSVC, but better than the other baseline models considered. Using these results, as well as the model definitions of Chapter 3 we summarized how well each model satisfies the practical requirements in Table 5.3 – indicating StarGAN-ZSVC was correctly designed and built to satisfy the requirements defined in Chapter 1 and be useful in practical applications. To our knowledge this is the only model to satisfy these four practicality requirements to the extent shown in Chapter 5.

6.4. Future work

With the StarGAN-ZSVC model designed and trained, and the other literature models profiled, there are several further avenues of research and engineering that open up:

- The StarGAN-ZSVC model and training method can be used to build and implement software packages and pipelines for downstream applications, such as a real-time voice changer for streaming software or integration with game engines such as Unity.
- Realizing the promise of data augmentation, the StarGAN-ZSVC and other models can be incorporated into existing programming libraries and packages to provide integrated data augmentation for utterances when training other speech processing systems.
- Further investigating the application of data augmentation, as discussed in Section 1.1, it would be valuable to train downstream speech and natural languages processing algorithms and compare the performance with and without using StarGAN-ZSVC (or other VC models) for data augmentation.
- Increasing focus has been placed on transformer-based networks for various domain translation tasks in vision and natural language processing recently, and it may be valuable to consider designing a transformer-based VC system to assess the transformer's applicability for VC.
- The comparisons in this project only covered a very low-resource setting. It will be valuable to scale the training to a much larger dataset and compare the performance of the models again to assess how the performance of each model changes with dataset size.
- The models chosen were all adapted from DNN-based models. Our comparison can be expanded to include other types of models which do not require training and are specifically tailored to work well in low-resource settings.

Bibliography

[1] J. Lorenzo-Trueba, J. Yamagishi, T. Toda, D. Saito, F. Villavicencio, T. Kinnunen, and Z. Ling, "The Voice Conversion Challenge 2018: Promoting Development of Parallel and Nonparallel Methods ," in *Odyssey Speaker and Language Recognition Workshop*, 2018.

[2] M. Versteegh, R. Thiollière, T. Schatz, X.-N. Cao Kam, X. Anguera, A. Jansen, and E. Dupoux, "The Zero Resource Speech Challenge 2015," in *Interspeech*, 2015.

[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, 2012.

[4] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation," in *IEEE CVPR*, 2018.

[5] B. M. Lal Srivastava, N. Vauquier, M. Sahidullah, A. Bellet, M. Tommasi, and E. Vincent, "Evaluating Voice Conversion-Based Privacy Protection against Informed Attackers," in *ICASSP*, 2020.

[6] C. Huang, Y. Y. Lin, H. Lee, and L. Lee, "Defending Your Voice: Adversarial Attack on Voice Conversion," *arXiv e-prints*, p. arXiv:2005.08781, 2020.

[7] H. Kameoka, T. Kaneko, K. Tanaka, and N. Hojo, "StarGAN-VC: Non-parallel many-to-many voice conversion using star generative adversarial networks," in *IEEE SLT Workshop*, 2018.

[8] T. Kaneko, H. Kameoka, K. Tanaka, and N. Hojo, "StarGAN-VC2: Rethinking Conditional Methods for StarGAN-Based Voice Conversion," in *Interspeech*, 2019.

[9] K. Qian, Y. Zhang, S. Chang, X. Yang, and M. Hasegawa-Johnson, "AutoVC: Zero-shot voice style transfer with only autoencoder loss," in *PMLR*, 2019.

[10] M. Baas and H. Kamper, "StarGAN-ZSVC: Towards zero-shot voice conversion in low-resource contexts," in *SACAIR*, 2020.

[11] B. Sisman, J. Yamagishi, S. King, and H. Li, "An Overview of Voice Conversion and its Challenges: From Statistical Modeling to Deep Learning," *arXiv e-prints*, p. arXiv:2008.03648, 2020.

[12] Y. Zhao, W.-C. Huang, X. Tian, J. Yamagishi, R. K. Das, T. Kinnunen, Z. Ling, and T. Toda, "Voice Conversion Challenge 2020: Intra-lingual semi-parallel and cross-lingual voice conversion," *arXiv e-prints*, p. arXiv:2008.12527, 2020.

[13] R. Prenger, R. Valle, and B. Catanzaro, "Waveglow: A Flow-based Generative Network for Speech Synthesis," in *IEEE ICASSP*, 2019.

[14] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," *arXiv e-prints*, p. arXiv:1609.03499, 2016.

[15] K. Kumar, R. Kumar, T. de Boissiere, L. Gestin, W. Z. Teoh, J. Sotelo, A. de Brébisson, Y. Bengio, and A. C. Courville, "MelGAN: Generative adversarial networks for conditional waveform synthesis," in *NeurIPS*, 2019.

[16] D. Suundermann, G. Strecha, A. Bonafonte, H. Höge, and H. Ney, "Evaluation of vtln-based voice conversion for embedded speech synthesis." in *Interspeech*, 2005.

[17] E. Moulines and F. Charpentier, "Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones," *Speech Communication*, vol. 9, no. 5-6, pp. 453–467, 1990.

[18] 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.

[19] M. Morise, F. Yokomori, and K. Ozawa, "WORLD: A Vocoder-Based High-Quality Speech Synthesis System for Real-Time Applications," *IEICE Transactions on Information and Systems*, vol. E99.D, no. 7, p. 1877–1884, 2016.

[20] B. C. Csáji *et al.*, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, no. 48, p. 7, 2001.

[21] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv e-prints*, p. arXiv:1412.6980, 2014.

[22] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," *EMNLP*, 2014.

[23] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural Computation*, vol. 9, pp. 1735–80, 12 1997.

[24] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, 2000.

[25] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-Normalizing Neural Networks," in *NeurIPS*, 2017.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *NeurIPS*. Curran Associates, Inc., 2019. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[28] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network," in *IEEE CVPR*, 2016.

[29] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *NeurIPS*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014.

[30] Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha, "StarGAN v2: Diverse Image Synthesis for Multiple Domains," in *IEEE CVPR*, 2020.

[31] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley, "Least Squares Generative Adversarial Networks," *ICCV*, 2017.

[32] T. Toda, A. W. Black, and K. Tokuda, "Voice Conversion Based on Maximum-Likelihood Estimation of Spectral Parameter Trajectory," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 8, pp. 2222–2235, 2007.

[33] Y. Stylianou, O. Cappe, and E. Moulines, "Continuous probabilistic transform for voice conversion," *IEEE Transactions on Speech and Audio Processing*, vol. 6, no. 2, pp. 131–142, 1998.

[34] Z.-W. Shuang, R. Bakis, S. Shechtman, D. Chazan, and Y. Qin, "Frequency warping based on mapping formant parameters," in *Interspeech*, 2006.

[35] D. Erro and A. Moreno, "Weighted frequency warping for voice conversion," in *Interspeech*, 2007.

[36] L. Sun, S. Kang, K. Li, and H. Meng, "Voice conversion using deep Bidirectional Long Short-Term Memory based Recurrent Neural Networks," in *IEEE ICASSP*, 2015.

[37] J. Chorowski, R. J. Weiss, S. Bengio, and A. van den Oord, "Unsupervised speech representation learning using WaveNet autoencoders," *arXiv e-prints*, p. arXiv:1901.08810, 2019.

[38] H. Kameoka, T. Kaneko, K. Tanaka, and N. Hojo, "ACVAE-VC: Non-Parallel Voice Conversion With Auxiliary Classifier Variational Autoencoder," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 9, p. 1432–1443, 2019.

[39] P. L. Tobing, Y.-C. Wu, T. Hayashi, K. Kobayashi, and T. Toda, "Non-Parallel Voice Conversion with Cyclic Variational Autoencoder," in *Interspeech*, 2019.

[40] M. Patel, M. Purohit, M. Parmar, N. J. Shah, and H. A. Patil, "AdaGAN: Adaptive GAN for Many-to-Many Non-Parallel Voice Conversion," 2020. [Online]. Available: https://openreview.net/forum?id=HJlk-eHFwH

[41] Y. Rebryk and S. Beliaev, "ConVoice: Real-Time Zero-Shot Voice Style Transfer with Convolutional Network," *arXiv e-prints*, 2020.

[42] L. Wan, Q. Wang, A. Papir, and I. L. Moreno, "Generalized End-to-End Loss for Speaker Verification," *ICASSP*, 2018.

[43] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *MICCAI*. Springer International Publishing, 2015.

[44] J. Howard and S. Gugger, "DynamicUnet: create a U-Net from a given architecture," 2020, Last accessed 8 Aug 2020. [Online]. Available: https://docs.fast.ai/vision.models.unet#DynamicUnet

[45] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of Tricks for Image Classification with Convolutional Neural Networks," in *IEEE CVPR*, 2019.

[46] V. Dumoulin, J. Shlens, and M. Kudlur, "A learned representation for artistic style," in *ICLR*, 2017.

[47] T. Miyato and M. Koyama, "cGANs with Projection Discriminator," in *ICLR*, 2018.

[48] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language Modeling with Gated Convolutional Networks," in *PMLR*, D. Precup and Y. W. Teh, Eds., 2017.

[49] J. Howard and S. Gugger. (2020) fastai. Last accessed 20 Aug 2020. [Online]. Available: https://docs.fast.ai/

[50] C. Veaux, J. Yamagishi, and K. Macdonald. (2017) CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit. Last accessed 1 Sep 2020. [Online]. Available: http://homepages.inf.ed.ac.uk/jyamagis/page3/page58/page58.html

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

[52] W. D. Basson and M. H. Davel, ""Category-Based Phoneme-to-Grapheme Transliteration"," in *Interspeech*, 2013.

[53] B. McFee, C. Raffel, D. Liang, D. Ellis, M. Mcvicar, E. Battenberg, and O. Nieto, "librosa: Audio and Music Signal Analysis in Python," in *SciPy*, 2015.

[54] L. N. Smith, "Cyclical Learning Rates for Training Neural Networks," in *IEEE WACV*, 2017.

[55] H. Sakoe and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.

[56] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[58] L. N. Smith and N. Topin, "Super-convergence: very fast training of neural networks using large learning rates," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, SPIE. SPIE, 2019.

Appendix A

Project Planning Schedule

Table A.1: Project planning schedule, showing rough date outlines planned for each stage of the project. Dates are given in YYYY-MM-dd format.

Time span		Planned progress
2020-04-05 2020-05-11	to	Planning and literature review. Learn and become familiar with deep learning and speech processing software packages (Pytorch, librosa, Fastai). Acquire hardware and software prerequisites (Linux, RTX 2070 GPU).
2020-05-11 2020-06-29	to	Get simple baseline vocoders and feature extraction blocks working – complete Mel-spectrogram and WORLD vocoders and feature extractions, and test them to make sure it is working. Complete GE2E speaker embedding model design, implementation, and training.
2020-06-29 2020-07-21	to	Complete design, implementation, and initial training of all models. Generate preliminary results for each model, develop the training procedure and code pipeline.
2020-07-21 2020-07-28	to	Complete initial consultation and improvement of models, data, and training pipeline. Test each part of the software setup and remove bugs.
2020-07-28 2020-8-17	to	Start write-up of introduction of report. Retrain all models with the previous tests and bug fixes implemented. Begin design of metrics and evaluations to perform on each model. Generate initial results on validation dataset.
2020-08-17 2020-08-31	to	Further consultation and improvement of testing and evaluation pipeline. Sanity check results and debug and optimize the testing pipeline and evaluation metrics. Complete first pass on literature review chapter in report.
2020-08-31 2020-09-21	to	Create, code, and host the subjective listening tests for additional comparison metrics between models. Complete the model design chapter of report. Perform initial review and consultation with supervisor about first two chapters.
2020-09-21 2020-10-12	to	Tabulate and compile all final metrics on test set. Generate all relevant figures for test set. Complete the experimental setup chapter of the report.
2020-10-12 2020-10-20	to	Complete results and conclusion chapter of report. Complete auxiliary report requirements (outcomes compliance, formatting...).
2020-10-20 2020-11-04	to	Consultation and review of report with supervisor; fix all spelling errors and ensure report maintains good flow. Setup final demonstration website with generated audio samples.

Appendix B

Outcomes Compliance

Table B.1: ECSA outcomes compliance, indicating the content in the report which demonstrates compliance with each ECSA learning outcome for this project.

ECSA outcome	Compliance
ELO 1. Problem solving:	This project aimed to compare and deliver a practical voice conversion system. This outcome is satisfied because: (i) The problem of developing a 'practical' VC system is under-specified and required us to concretely specify and define what entails a practical VC system (Section 1.2). (ii) The models profiled in Sections 3.1, 3.4, 3.2 are all well beyond the scope of engineering codes and standards, and (iii) the solution developed in Section 3.5 both draws on several complex models (one of which is even erroneously defined by the original authors!) to construct a working VC system.
ELO 2. Application of scientific and engineering knowledge:	The knowledge of statistics, machine learning, and data science all played a pivotal role in both defining each model in Chapter 3, defining our own StarGAN-ZSVC in Section 3.5, and motivating the loss function and training choices described for each model and in the literature review in Chapter 2. Mathematical and engineering sciences also is evident in reasoning about and designing various comparison metrics in Section 4.5 and 4.6, particularly with the introduction of the embedding norm difference $\|\|\Delta\mathbf{s}\|\|$ in Section 4.5.4.
ELO 3. Engineering Design:	Procedural design and synthesis of components is used in describing, defining, and programming the complex problem of utterance feature extraction and vocoding procedures (given in Section 2.3.1, 3.3, and 4.3) by combining existing techniques from the literature and the electronic engineering discipline. Non-procedural design is used extensively in both designing the various models (StarGAN-ZSVC and StarGAN-VC2 in particular in Sections 3.2 & 3.5), and in the numerous techniques investigated to attempt to train and compare each model (a complex engineering problem), as expanded in Chapter 4 and Appendix E.
ELO 4. Investigations, experiments and data analysis:	The project required construction of several metrics to evaluate and compare between models, given in Section 4.5 and 4.6. These metrics and the entire experimental setup in Chapter 4 provides strong evidence for competence in designing and conducting investigations and experiments to compare each VC model. Selected knowledge in the literature was used to construct the subjective evaluation metrics of Section 4.6, and the various objective metrics are based on knowledge from the electronic engineering subfield of machine learning and data science. Substantial investigation into the appropriate literature was conducted in Section 2.4.
ELO 5. Engineering methods, skills and tools, including Information Technology:	The extensive use of various machine learning software packages (Pytorch, Fastai...) to define, train, and compare each model is evident throughout Chapter 4 and Chapter 5. To obtain the various results of Chapter 5 required the significant use of sophisticated, deep learning specific software packages. Further web development software packages were used in Section 4.6 (Flask, Google Cloud Compute) to conduct the subjective listening tests as part of the investigation and results in Chapter 5.
ELO 6. Professional and technical communication:	This outcome is demonstrated by this report in its entirety, the oral presentation, and poster presentation (to be presented). The methods of providing information in this report include conventional methods such as plots (e.g. Figure 5.1), system diagrams (e.g. Figure 3.5), and tables (e.g. Table 5.1). All of these provide evidence that demonstrates competence in communicating with engineering audiences and the community at large.
ELO 8. Individual work:	All work, coding, and design decisions presented in this project are done alone, with the sole advice from supervisor who was consulted on a weekly basis. The literature review of Chapter 2, like the model design of Chapter 3, training and implementation in Chapter 4, and actual experiments conducted in Chapter 5 are all performed personally by myself (M. Baas).
ELO 9. Independent Learning Ability:	Independent learning ability is demonstrated by the literature review of Chapter 2, which demonstrates independent learning of several advanced topics outside the scope of undergraduate engineering – from deep learning in Section 2.3.2 to GANs (Section 2.3.6), and speech representations (Section 2.3.1. Further learning is demonstrated by the acquirement of skills in training, designing, and comparing complex DNN models in software using Pytorch, as demonstrated by the experimental setup of Chapter 4 and results in Chapter 5. Ethical considerations associated with the project are also given in Section 1.4.

Appendix C

Speech representations supplement

C.1. The Mel-scale

The Mel-scale is an alternate scale of frequency (as opposed to the Hertz scale) which is defined such that equal distances on the Mel-scale are interpreted as perceptually equal distances by human listeners [18]. Although no unique formula exists to convert frequency in Hertz to the Mel-scale, they all exhibit a logarithmic relationship. For all of our implementations we use the equation defined by the audio processing library librosa [53]:

$$f_{\text{mels}} = 2595 \log_{10}(1 + \frac{f_{\text{Hz}}}{700}) \tag{C.1}$$

where f_{mels} is the frequency value on the Mel-scale, and f_{Hz} is the original frequency in Hertz.

C.2. The Mel-spectrogram

Using the Mel-scale, we can obtain a representation X of an input waveform $\mathcal{X}$ by first performing a Short-time Fourier transform (STFT) of the input waveform. Next, to convert it to the Mel-scale, we define a scalar number of Mel-frequency bins m. For the work performed in this project, we define $m = 80$ in all our experiments. Now, we compute m uniformly separated frequencies on the Mel-scale over some finite frequency range $[f_{\text{min}}, f_{\text{max}}]$. Typically $f_{\text{min}} = 0\,\text{Hz}$ and f_{max} is set at just below half the sampling frequency to ensure that the computed Mel-scale frequency bins cover all the meaningful information present in the original STFT (in accordance with the Nyquist sampling theorem).

Next, since these Mel-scale frequency bins do not directly map to the frequency bins of the original frequency bins of the STFT, a filtering operation is performed whereby, using these m uniform Mel-scale frequencies, Mel frequency i serves as the center frequency of a triangular filter that spans from the previous Mel frequency $i - 1$ to the following frequency $i + 1$. The height of each triangular filter is set to have a maximum amplitude of 1 at its center frequency, and then further normalized by dividing the magnitude of the filter by the width of the triangle (in Hertz). Performing such a process yields a set of discrete triangular filters as shown in Figure C.1. The hyperparameters of the Mel filters and STFT given in Figure C.1 are the ones used for the experiments in this project, chosen based on the sampling rate of the VCC dataset [1] (discussed in Chapter 4).

To obtain the Mel-scale spectrogram, the dot product is taken between each of the m filters and each frame of the STFT *magnitudes* – the phase is discarded in this process and only the magnitude of each DFT point is used. The scalar output from each dot product corresponding to each of the m filters applied to a single frame are then concatenated together to form one $m-$dimensional frame of the output Mel-scale spectrogram, or simply Mel-spectrogram [53].

Finally, as a practical consideration, the values of these Mel-spectrograms can become

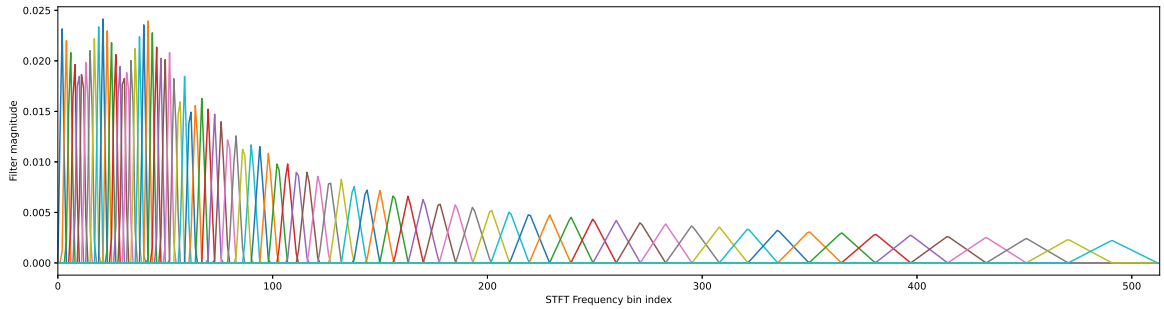


Figure C.1: Impulse response of Mel-scale discrete triangular filters for $m = 80$ uniformly spaced Mel-scale frequency bins between 0 Hz and 8000 Hz. Each of the 80 discrete filters is constructed to be applied to each frame of a STFT utilizing a 1024-point DFT. Each triangular filter is shown in a separate color, indicating the magnitude of the triangular filter for each DFT index.

large enough in magnitude to cause instability for some machine learning methods. Thus, typically VC systems prefer to use the *log* Mel-spectrogram, which is simply the (possibly natural) logarithm of the Mel-spectrogram magnitudes [12].

C.3. Other frequency-domain representations

As mentioned, there are several other possible ways to represent the frequency content of speech. For example, some VC systems use Mel-frequency cepstral coefficients (MFCCs) which consists of further processing a Mel-spectrogram using a discrete cosine transform [53]. Meanwhile, others use the WORLD feature extractor which represents an utterance as a 3-tuple of pitch, aperiodicity, and spectral envelope sequences [19].

Appendix D

Training software and setup supplement

This appendix describes detailed supplementary information for how each model is trained in software, and the precise method used during data loading, model definition, and training.

D.1. Software and tooling detail

The first is the Pytorch machine learning framework, which provides a useful Python interface to define DNNs, manipulate tensors of arbitrary rank, and train DNNs by automatically computing the loss gradients with respect to arbitrary parameters by carefully recording the computation path leading to the loss (or any other variable the user wishes to take the derivative off) in a process known as backpropagation [27]. We use Pytorch to define each model, and to manually perform various operations on tensors.

The second library used is the librosa audio processing library, which is a fairly standard library in the literature for extracting features from and processing audio. We use it to aid in constructing and displaying Mel-spectrograms [53].

The third major library used is the Fastai deep learning framework [49]. It is a library that sits atop Pytorch to offer additional functionality, such as various data transform, loading, and training utilities. We use it to build the Python constructs which parses and loads each batch of data to the models, and to perform the actual training loop.

Furthermore, training and experimentation is performed and data is loaded in Jupyter notebooks, which provide a rich interactive IDE experience for editing interpreted languages such as Python. After pieces of code were verified to work in a Jupyter notebook, they are mostly offloaded to python files to be imported when needed.

For hardware, all training and speed measurements are performed on a single NVIDIA RTX 2070 SUPER graphics processing unit (GPU). This is done because NVIDIA offers special hardware libraries to Pytorch that drastically speeds up tensor computations on GPUs compared to standard central processing units (CPUs).

D.2. Training loop detail

D.2.1. Non-GAN methods

The process to train each non-GAN model is nearly identical: we follow the abstract four-step training loop defined in Section 2.3.2 with a batch size of 8 for all models execpt AutoVC, which requires a smaller batch size of 4 as per the author's experiments [9]. The optimizer (weight update equation in the final step) is defined as the Adam optimizer [21] whereby the running gradient average $\tilde{m}_1$ updates with momentum 0.9, and the square gradient running average $\tilde{m}_2$ updates with momentum 0.99.

The stability constant is set at $\epsilon = 10^{-5}$, and we also add an L_2 weight penalty with coefficient $\lambda = 0.01$ to help regularize the model [27]. The precise details of this weight decay are similar to those used for logistic regression when a prior over the weight is assumed, however the details of how this weight decay interfaces with the standard Adam update step is beyond the scope of this project.

The last aspect of training that needs clarification is the learning rate and duration of training. For this, we use the method developed by Smith [54] and expended by Smith in [58]. In this method we follow a special procedure to determine a good learning rate α before we train the network. Formally we run several batches through the network with a exponentially growing learning rates from some minimum α_{min} (10^{-7} in our experiments), computing the loss and updating the weights each step. The loss is plotted against the learning rate, and this process is continued until the loss begins to diverge.

The optimal loss is then read off from the plot at the point where the loss decreases the fastest. The weights are then reset to their value before this process, and standard training proceeds using this determined learning rate. To train each non-GAN model, we perform this process to obtain a good α, train for several hundred epochs using a one-cycle learning rate scheduler based on this α as defined by Smith [58]. We then repeat these steps until the loss on the validation set converges.

The theoretical motivations provided by Smith for finding a good learning rate and the details of the one-cycle scheduler are beyond the scope of this project. They were chosen after a lengthy period of trial and error among several methods, whereby we found these methods by Smith (and functionality provided in Fastai) consistently yielded good results in our experiments.

D.2.2. GAN-based methods

Training StarGAN-VC2 and StarGAN-ZSVC followed instead the training loop given in Algorithm 2.1 with the StarGAN adaptation defined in Section 2.3.6 and using a batch size of 8. The Adam optimizer used is the same as for the non-GAN methods, except with the square gradient update momentum set at 0.5 to ensure that the generator or discriminator never starts to overpower the other.

Instead of using the method by Smith for training, for the GAN models we simply train for several thousand epochs at a learning rate of $\alpha = 2 \times 10^{-4}$ until the validation loss begins to increase. However, GANs are known to be somewhat difficult to train and we needed to make several special additions to the training process to produce good results:

- The gradients ∇_{θ_G} and ∇_{θ_D} are clipped to have a maximum L_2 norm of 1.
- The discriminator's learning rate is made to always be half of the generator's learning rate.
- The number k from Algorithm 2.1 is updated every several hundred epochs to ensure that the discriminator's loss is always roughly a factor of 10 lower than the adversarial term of the generator's loss ($L_{G-\text{adv}}$).
- A dropout layer is added to the input of the discriminator with probability 0.3 after the first 3000 epochs (if added sooner it causes artifacts and destabilizes training, if not added the generator begins to experience mode collapse).

For the LSGAN constants we set $a = 1$ and $b = 0$. For the loss coefficients we set $\lambda_{\text{cyc}} = 10$, $\lambda_{\text{id}} = 5$, with λ_{id} being adjusted exponentially downwards during training until $\lambda_{\text{id}} = 0.01$ by epoch 4000. We found these settings to yield good results.

Appendix E

Experiment hurdles

Herewith we provide a list of experiments attempting to use various techniques or implement various changes to successfully train each VC model.

Table E.1: Attempts used to overcome hurdles in training StarGAN-VC2 (and thus StarGAN-ZSVC) successfully, along with the outcome and explanation of each experiment.

Solution attempt	Outcome	Suspected explanation
Original paper specifies Adam momentum as 0.5, but unclear if it is momentum for gradient and square gradient term. Attempt to set both momenta to 0.5.	Failure	Model fails to train at all, must be only gradient momentum.
Change target speaker sampling to assign lower probabilities to source speaker so that speaker A $\rightarrow$ A mapping is less common.	Little effect	A $\rightarrow$ A mapping must already be sufficiently optimized by the identity and cyclic loss terms.
Experiment with changing loss function coefficients to make each term (numerically) roughly the same size.	Failure	Model appears to need the cyclic loss term to have much higher magnitude, probably because its gradients travel backwards much farther and appear to vanish more.
Change how CIN weights initialized to get $\gamma = 1$, $\beta = 1$ so as to initialize with identity mapping.	Failure	Does not improve training, perhaps the random initialization is good because it provides richer gradients.
Train generator with only cyclic and identity loss terms to check software framework.	Success	The generator does indeed learn to successfully perform identity mappings (speaker A $\rightarrow$ A conversios), and thus software framework likely works.
To get D to work for arbitrary sized inputs, change GSP layer for a traditional sum or average pooling over channels.	Failure	Sum pooling fails as tensor mean changes based on length of utterance, average pooling fails as long silences drastically affects tensor mean.
Attempt to switch between training D and G using an adaptive GAN switcher [49], which switches between D and G as their loss decreases to a fixed point.	Failure	The training process for the GAN is very non-linear, and sometimes both D and G's loss will increase by a factor of 3 yet produce better results. So setting a fixed loss point to switch between optimizing D and G did not work.
Pretrain G using non-adversarial loss terms to speed up training.	Little effect	Does not significantly speed up training, likely as pretraining does not help G learn actual A$\rightarrow$B speaker mappings.
Try bound D output with a non-linearity such as sigmoid or softplus to aid it learn to output near LSGAN a and b.	Failure	It appears to not learn at all now. Perhaps forcing it to output small values without non-linearity helps regularize the magnitude of the weights.
Apply gradient clipping to G and D.	Success	Improves training stability.
Scale gradient of embedding layers in StarGAN-VC2 by frequency of embedding use.	Failure	Appears to not produce a significant change, if not slightly worsening output and increasing number of parameters.
Add weight decay to training.	Marginal improvement	Slight improvement from increased regularization, we suspect that D already is strongly regularized by forcing its unbounded output to be small.
Use different configurations with instance normalization layers recording a running average of statistics.	Mixed	Appears to help at inference, but fails during training as the statistics of a poorly trained cyclical mapping is far from that of a real Mel-spectrogram.
Initialize StarGAN-VC2 embedding layers with Fastai's truncated normal initialization.	Success	Drastically improves speed of training, due to lots of underlying theory about DNN initialization that is beyond the scope of this project.
Completely drop the $\mathcal{L}_{\text{id}}$ term after 3000 epochs as per original paper.	Failure	Model begins to fail at identity mappings after the loss term is removed, rather keep it but with lower coefficient as training proceeds.
Use Adam with zero momentum as in other popular GAN implementations.	Little effect	Seemed to increase stability slightly, but slowed down convergence as well.

Table E.2: Various techniques used to overcome the hurdle of training the speaker embedding model successfully, along with the outcome and explanation of each experiment.

Solution attempt	Outcome	Suspected explanation
Use initial model defined in AutoVC with LSTMs	Marginal failure	Model is rather slow using LSTMs, but still largely works.
Attempt to add various forms of dropout as in Fastai's AWD LSTM [49] implementation (weight dropout, cell state dropout...)	Little effect	With the large amount of data (roughly 2 hours per epoch) available to train the GE2E model, adding all the dropout appears to just slow down training slightly.
Switch to GRU cells instead of LSTM cells.	Success	Performance similar and speed improved moderately.
Attempt to scale the input according to Section 4.3.	Failure	Reduced performance, possibly because GRU cells use substantial amounts of sigmoid non-linearities which already bounds values between zero and one.
Attempt to directly perform VC by freezing trained GE2E model and treating an input spectrogram X_{src} as the trainable parameters θ, and directly attempt to minimize the loss $\|\|E(X_{\text{src}}) - \mathbf{s}_{\text{trg}}\|\|_2$ by optimizing the actual values of the spectrogram.	Failure	In the end the learnt spectrogram becomes purely an adversarial attack on E to make $E(X)$ very close to a desired speaker embedding without changing the spectrogram in a semantically meaningful way.
Attempt to perform the same direct optimization described in the previous entry but with other loss functions between $E(X)$ and the desired $\mathbf{s}_{\text{trg}}$, such as KL divergence.	Failure	Still no meaningful results, likely still performing an adversarial attack on $E(X)$. Some anti-adversarial attack measures might be required to make this technique work.

Table E.3: Tooling, hardware, and software issues encountered and attempted solutions, along with outcomes and suspected explanation for each outcome.

Solution attempt	Outcome	Suspected explanation
Use the WORLD vocoder for experiments since it does not require training (good for low-resource).	Failure	By representing utterances as 3-tuples, it requires conversion of 3 separate signals, which is non-trivial and all existing systems utilize the mean pitch information of target speakers – not a variable obtainable in a zero-shot setting.
We encountered `CUDNN_STATUS_BAD_PARAM` errors at arbitrary times during training when loading a batch. Upon investigation, the SATA cable between the SSD and motherboard was found to have one pin corrupted. Replace SATA cable.	Success	No further arbitrary CUDNN errors when loading batches.
We encountered `CUDNN_STATUS_BAD_PARAM` errors at arbitrary times when gradients were backpropogated through an LSTM or GRU layer whose parameters were not being trained. Upon investigation, it was hypothesised that this was a low-level Windows issue with how it interfaces with CUDA. Solution: switch to Ubuntu Linux.	Success	No further CUDA errors during training.

Table E.4: AutoVC training issues encountered and attempted solutions, along with outcomes and suspected explanation for each outcome.

Solution attempt	Outcome	Suspected explanation
Attempt to train model with initial WaveNet vocoder to try replicate exact results of authors.	Failure	WaveNet vocoder is very slow (15s to vocode 1s of audio).
Training appeared to be failing, so we applied scaling to the input as defined in Section 4.3.	Success	Training more smooth as is often the case when scaling input.

Table E.5: StarGAN-ZSVC training issues encountered and attempted solutions, along with outcomes and suspected explanation for each outcome.

Solution attempt	Outcome	Suspected explanation
The model apperas to experience mode collapse after a long time of training. Try add dropout to the discriminator input spectrogram.	Failure	The discriminator fails to ever train properly, and introduces weird artifacts into the generated results. The likely cause is that the mean tensor value changes slightly, which affects the discriminator output through the global sum pooling layer.
Use dropout to input of D, but only add the dropout after several thousand epochs once both D and G are reasonably good.	Success	No mode collapse or artifacts arise during training.
Try out different non-linearities for the layers operating on the speaker embeddings (SELU, ReLU, Softplus...)	Mixed	SELU yielded the best results on validation set (faster training and slightly better results).
Add additional loss term akin to AutoVC, minimizing L_1 distance between $E(X_{\text{src}\to\text{trg}})$ and $\mathbf{s}_{\text{trg}}$.	Mixed	Model appears to be training slightly better and results look promising, but adding this loss term slows down training too much to be feasable, thus we discard this change.
Try utilize different pooling layers instead of global sum pooling in discriminator to allow arbitrary length inputs.	Failure	All other pooling types causes the dot product to yield high variance outputs when utterances are of varying length. Thus rather stick with global sum pooling and fixed-size spectrograms.
Use different scales for learning rate (such as making the discriminator's learning rate 4x the generator's learning rate).	Mixed	Most choices immediately cause training to diverge, and ultimately setting the discriminator's learning rate to half the generator's learning rate worked consistently well.

Appendix F

Subjective evaluation data

This appendix includes configuration information for the subjective evaluation survey and converted sample spectrograms for each model to provide the reader with an additional visual means to assess each model's performance.

F.1. Subjective evaluation survey breakdown

Each listener listens to 100 utterances given in random order with randomized filenames. The 100 utterances are comprised of the following:

- 8 random true source utterances (purely giving them the raw waveform $\mathcal{X}$).
- 8 random vocoded source utterances (taking the spectrogram of a true utterance X and then using the WaveGlow vocoder to convert it back to a waveform $\mathcal{X}$).
- 8 random converted utterances from StarGAN-ZSVC's seen-to-seen test set.
- 8 random converted utterances from StarGAN-ZSVC's seen-to-unseen test set.
- 8 random converted utterances from StarGAN-ZSVC's unseen-to-seen test set.
- 8 random converted utterances from StarGAN-ZSVC's unseen-to-unseen test set.
- 8 random converted utterances from AutoVC's seen-to-seen test set.
- 8 random converted utterances from AutoVC's seen-to-unseen test set.
- 8 random converted utterances from AutoVC's unseen-to-seen test set.
- 8 random converted utterances from AutoVC's unseen-to-unseen test set.
- 8 random converted utterances for StarGAN-VC2's seen-to-seen test set.
- all 4 converted utterances for the linear model's test set (recall these must convert from SF1 to SM2).
- all 4 converted utterances for the DBLSTM model's test set.
- all 4 converted utterances for the UNet model's test set.

This setup allows for all possible seen/unseen comparisons and provides a baseline MOS for both raw and vocoded source waveforms from the dataset.

F.2. Converted spectrogram samples

This section provides a handful of converted sample (log Mel-scale) spectrograms for each model, along with the ground-truth target spectrogram and source spectrogram. In all of the following figures each row corresponds to one VC sample, with the input spectrogram in the left column, ground-truth target in the middle column, and the converted spectrogram for the model in question in the right column.

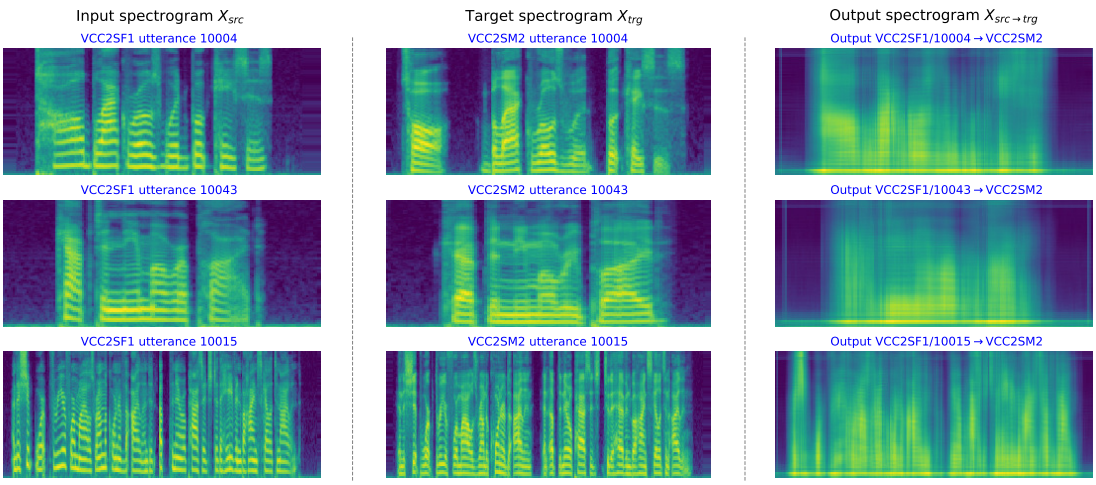


Figure F.1: Linear model conversion samples for one-to-one conversion from speaker SF1 to speaker SM2.

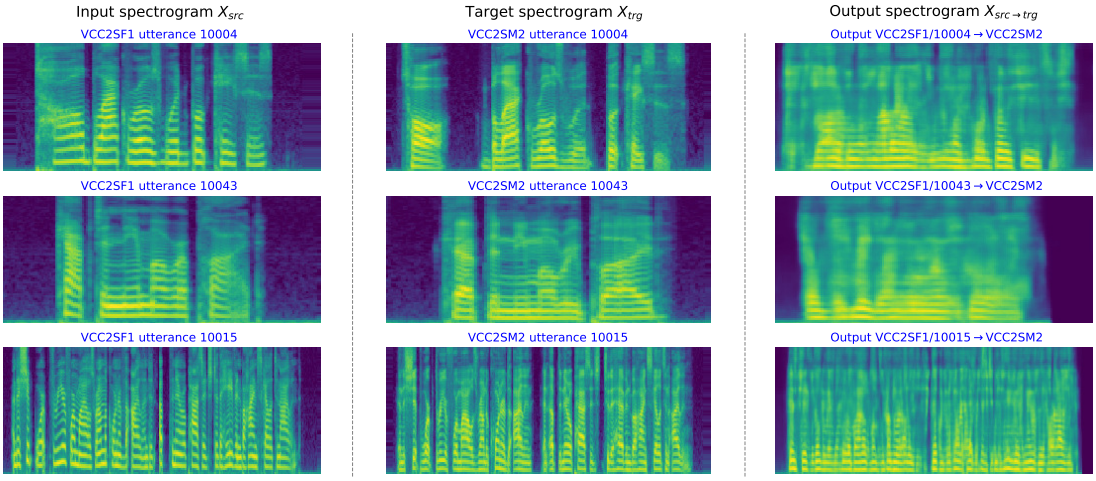


Figure F.2: UNet model conversion samples for one-to-one conversion from speaker SF1 to speaker SM2.

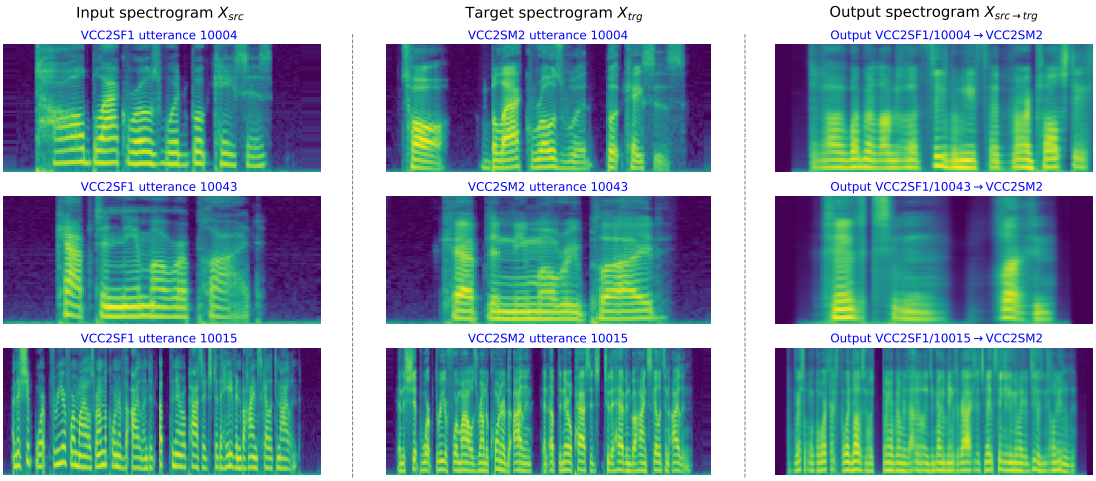


Figure F.3: DBLSTM conversion samples for one-to-one conversion from speaker SF1 to speaker SM2.

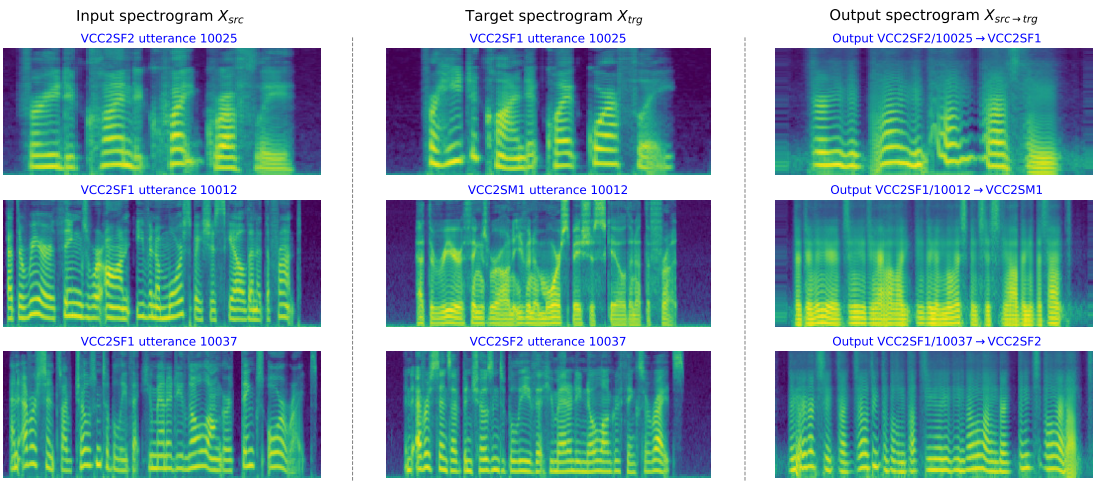


Figure F.4: StarGAN-VC2 conversion samples for seen-to-seen conversion task from arbitrary training speakers to arbitrary training speakers.

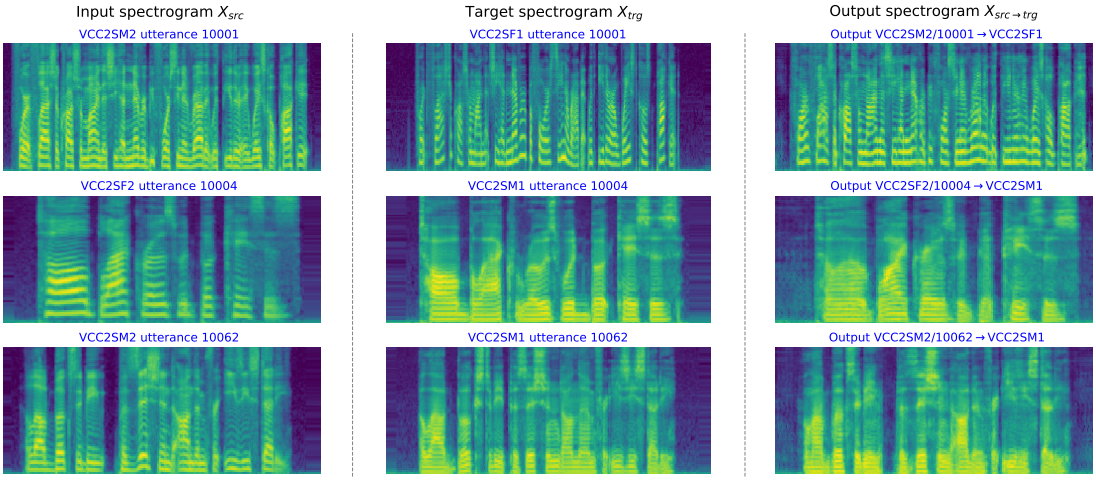


Figure F.5: StarGAN-ZSVC conversion samples for seen-to-seen conversion task from arbitrary training speakers to arbitrary training speakers.

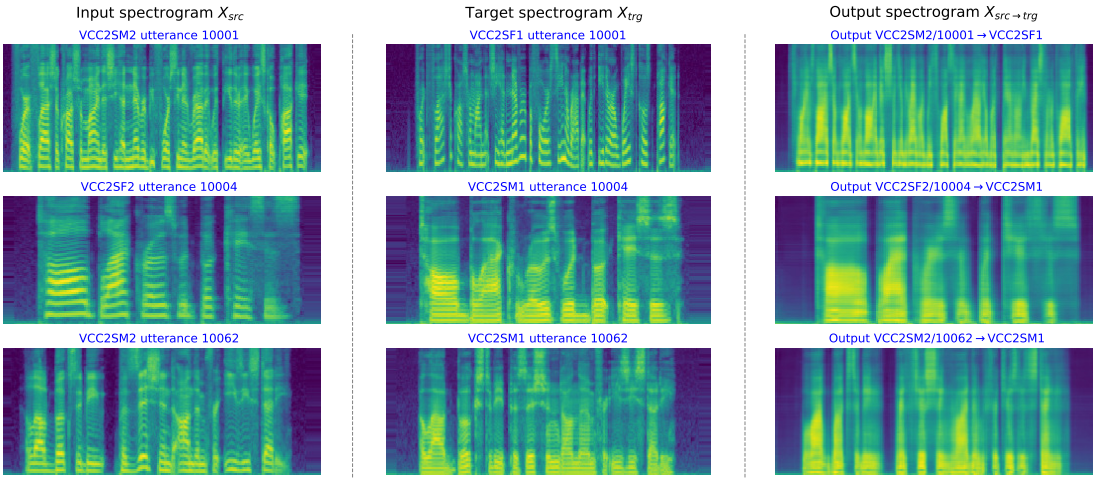


Figure F.6: AutoVC converksion samples for seen-to-seen conversion task from arbitrary training speakers to arbitrary training speakers.

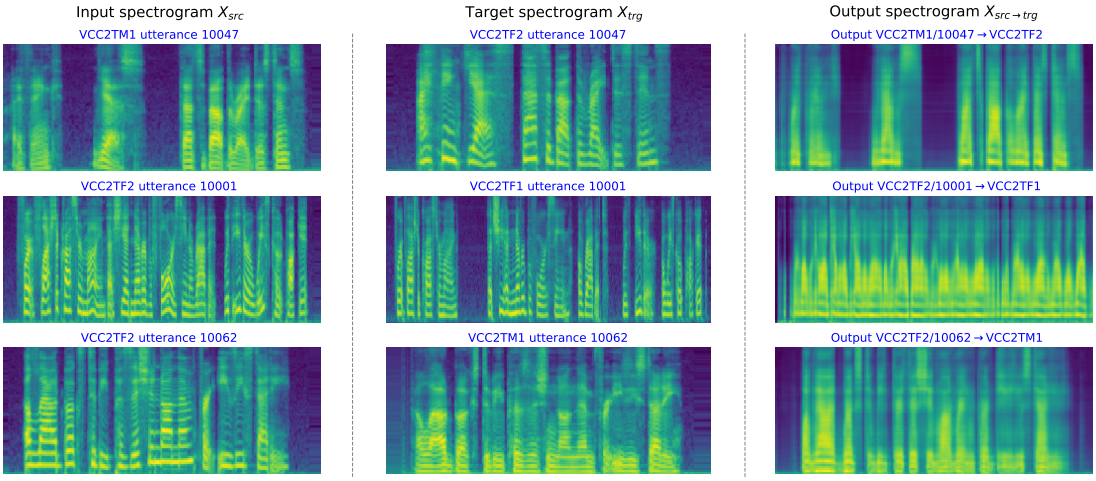


Figure F.7: AutoVC conversion samples for the unseen-to-unseen zero-shot conversion task.

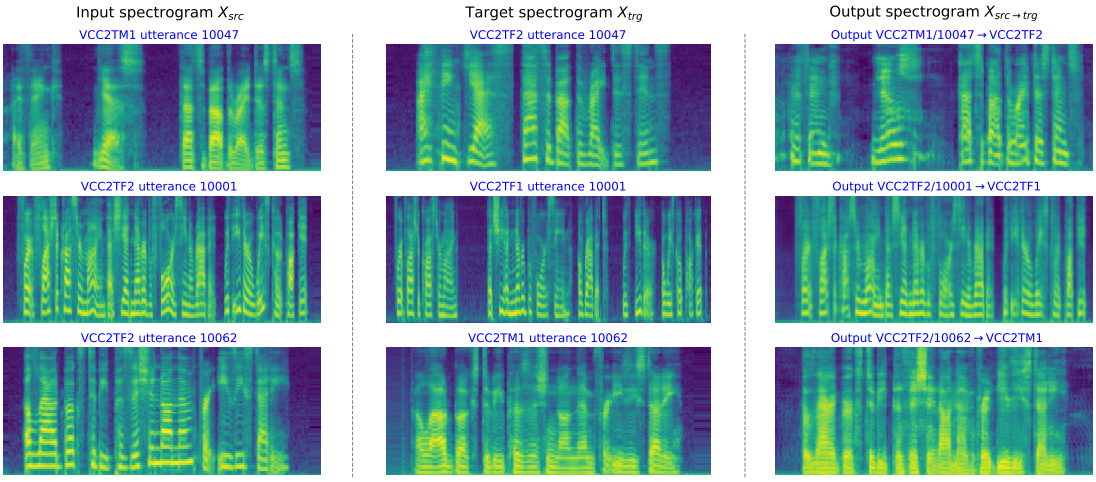


Figure F.8: StarGAN-ZSVC conversion samples for the unseen-to-unseen zero-shot conversion task.