Speech Synthesis for Text-Based Editing of Audio Narration

Zeyu Jin

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Adam Finkelstein

June 2018

Abstract

Recorded audio narration plays a crucial role in many contexts including online lectures, documentaries, demo videos, podcasts, and radio. However, editing audio narration using conventional software typically involves many painstaking low-level manipulations. Some state of the art systems allow the editor to perform select, cut, copy and paste operations in the text transcript of the narration and apply the changes to the waveform accordingly. However such interfaces do not support the ability to synthesize new words not appearing in the transcript. While it is possible to build a high fidelity speech synthesizer based on samples of a new voice, to operate well they typically require a large amount of voice data as input as well as substantial manual annotation.

This thesis presents a speech synthesizer tailored for text-based editing of narrations. The basic idea is to synthesize the input word in a different voice using a standard pre-built speech synthesizer and then transform the voice to the desired voice using voice conversion. Unfortunately, conventional voice conversion does not produce synthesis with sufficient quality for the stated application. Hence, this thesis introduces new voice conversion techniques that synthesize words with high individuality and clarity. Three methods are proposed: the first approach is called CUTE, a data-driven voice conversion method based on frame-level unit selection and exemplar features. The second method called VoCo is built on CUTE but with several improvements that help the synthesized word blend more seamlessly into the context where it is inserted Both CUTE and VoCo select sequences of audio frames from the voice samples and stitch them together to approximate the voice being converted. The third method improves over VoCo with deep neural networks. It involves two networks: FFTNet generates high quality waveforms based on acoustic features, and TimbreNet transforms the acoustic feature of the generic synthesizer voice to that of a human voice.

Acknowledgements

First of all, I would like to thank my advisor, Adam Finkelstein, for his mentorship throughout my Ph.D. process. I am extremely lucky to have the opportunity to work with him. He has provided tremendous amount of inspiration and advise to my research and professional development. I would like thank him for the countless hours of he spent on polishing our papers and his kindness and patience during and outside paper deadline hours. No words can express my gratitude; I would like to dedicate this thesis to all the years we spend together and all the potato chips we consumed during our late meetings!

The next biggest thank goes to my Adobe Research mentor, Gautham Mysore for his persistent inspiration and guidance in the past three years. It is my greatest pleasure to work with him and he helped enter the world of speech processing and connect with awesome people in the field. Moreover, my thesis work, VoCo, is developed during my internship at Adobe Research and has generated surprising amount of interest. I would like to thank Gautham for the great opportunity to work at Adobe and dedicate this thesis to all the beers we had during ICASSP and Adobe happy hours!

I would also like to the Graphics Lab, especially Thomas Funkhouser, Szymon Rusinkiewicz, and Marshini Chetty for their valuable input and support to my thesis work. I am really lucky to work in the Graphics Lab despite I am an audio guy. I have learned a lot from PXIL lunch and everyone in the group. I also want to thank my collaborators and friends from Adobe Research and Princeton University. Thank you, Cynthia Lu and Stephen DiVerdi for all the support and advise in my projects! Thank you, Reid Oda and Katie Wolf for all the great time we had in my first three years! Thank you, my Graphics Lab friends Xinyi Fan, Ohad Fried, Fisher Yu, Maciej Halber, Linguang Zhang, Huiwen Chang, Riley Simmons-Edler, Nora Willett and Elena Sizikova for all the fun we had, from Mafia to Psyco. And

special thanks to my food buddies Zhixing Xu and Haoyu Zhang for your help in maintaining my weight.

Last and not least, I would like to thank my family for raising me and educating me to the person I am today.

This thesis supported in part by generous gift from Adobe Research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Text-based speech editing

Modern technology have vastly accelerated the spread of news and communication of knowledge. We are blessed with many forms of media to keep us entertained and informed: movies, podcast, YouTube videos, interactive online education, etc. And every single one of them involves recording of audio narrations. After narration is recorded, most of these applications require editing. Typical audio editing interfaces present a visualization of the audio such as waveform and/or spectrogram (Figure 1.1) and provide the user with standard select, cut, copy, paste and volume adjustment, which are applied to the waveform itself. Many also support advanced operation such time stretching, pitch bending and de-noising. Such tools often requires domain knowledge to operate; the interaction can be cumbersome, especially for non-experts.

Researchers have addressed this problem by aligning the waveform with a transcript of the narration, and providing an interface wherein the user can perform cut-copy-paste operations in the text of the transcript. Whittaker and Amento [98] show that this form of interface significantly reduces search and editing time, and is

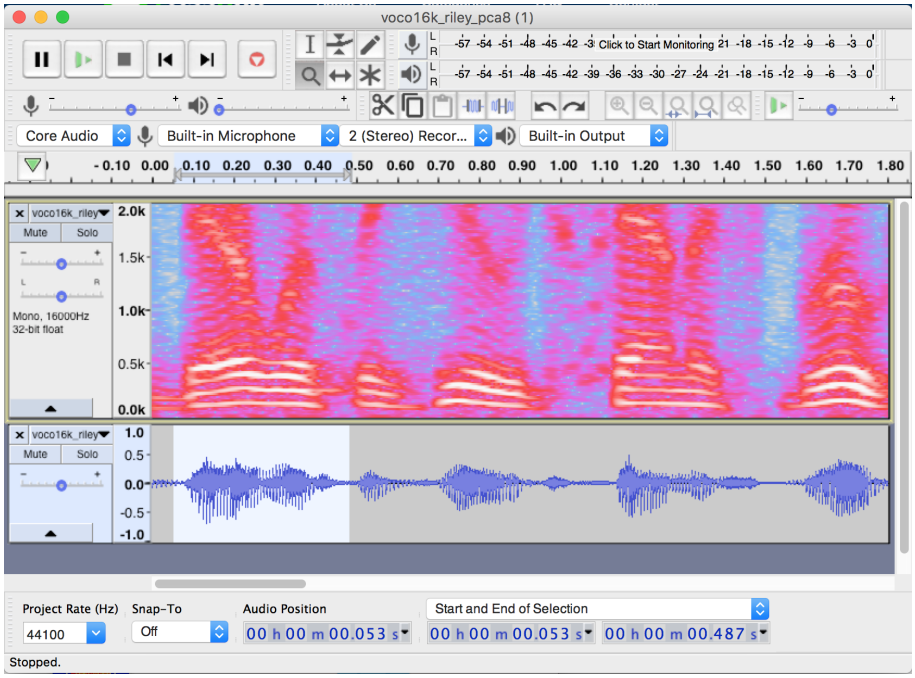


Figure 1.1: An example of conventional audio editing interface (Audacity)

preferred by users. State of the art video editing systems incorporate audio-aligned transcript editors to facilitate search and editing tasks [6, 14] (Figure 1.2).

While cut-copy-paste operations are well supported, one aspect is conspicuously missing from text-based audio editors: insertion. It is easy for a person to type a new word not appearing in the transcript, but it is not obvious how to synthesize the corresponding audio. Nevertheless, in many circumstances inserting a new word or phrase during editing would be useful, for example replacing a misspoken word or inserting an adjective for emphasis. Of course it is possible to record new audio of just the missing word, but to do so requires access to the original voice talent. Moreover, even when the the original narrator, microphone and acoustic environment are available for a new recording, it remains difficult to match the audio quality of an inserted word or phrase to the context around it. Thus the insertion is often evident in the edited audio, even while methods like that of Germain et al. [28] can ameliorate such artifacts. Regardless, just as it is easier to type for cut and paste, it is also easier to type for insertion or replacement rather than record new audio.

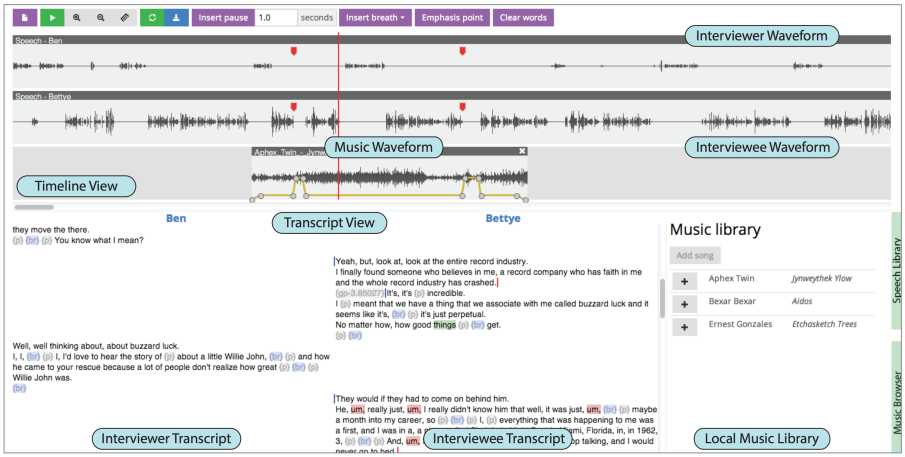


Figure 1.2: Script-based editing interface proposed by Rubin et al.. [69] This interface supports navigating conversational recording via the transcript as well as basic operations such as select, cut, copy and paste.

Thus, this dissertation focuses on synthesizing a word or short phrase to be inserted into a narration, based on typing. The input is an existing recording coupled with a transcript, as well as the text and location of the word to be inserted. With the proliferation of voice-based interfaces, there now exist many high-quality text-to-speech (TTS) synthesizers. The challenge is to automatically synthesize the inserted word in a voice that sounds seamless – as if it were uttered by the same person in the same recording as the rest of the narration (the *target* voice). While it is possible to customize a speech synthesizer for a specific target, conventional approaches (e.g., Acapela Group [1]) start from a large corpus of example recordings (e.g., 10 hours) and require many hours of human annotation (e.g., three months). Modern TTS methods can produce full sentences (or paragraphs) with both *clarity* (that is, free of noticeable artifacts like popping or muffling) and high *individuality* (that is, sounding like the target). Our goal is to synthesize a word or short phrase (as opposed to a full sentence) that reasonably matches the target voice in the context into which it is inserted, with clarity and individuality, but based on a much smaller corpus that lacks human annotation.

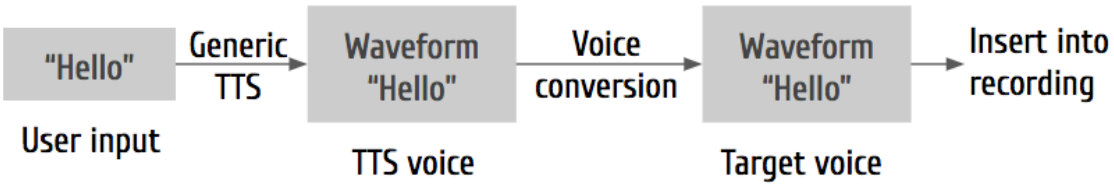


Figure 1.3: Basic idea of proposed word synthesis method. When a user types the word "hello", we first synthesize it using an off-the-shelf generic speech synthesizer and then transform it to the voice that we synthesize for using voice conversion technique. Then the synthesized word is inserted into the actual recording based on user input.

The key idea of our approach (Figure 1.3) is to synthesize the inserted word using a similar TTS voice (e.g., having correct gender) and then modify it to match the target using *voice conversion* (making an utterance by one person sound as if it was made by another). Voice conversion has been studied for decades, and the idea of using voice conversion for TTS was proposed by Kain and Macon [36] in the 1990s. However, only recently have voice conversion methods been able to achieve high individuality together with substantial clarity [47]. Unlike TTS, voice conversion requires significantly less data to operate (20–60 minutes). However, state-of-the-art voice conversion results remain inferior to that of TTS. This dissertation aims at devising new voice conversion methods that close the gap for text-based editing of audio narration.

1.2 Related work

1.2.1 Text-based speech editing

Audio editing tools such as Adobe Audition and Audacity allow experts to edit waveforms in a multi-track timeline interface. Audio manipulation is achieved by chaining low-level signal processing tools that require expertise and knowledge to use. However, such general-purposed editing tools typically do not have a specialized speech processing interface as they are unaware of the linguistic properties

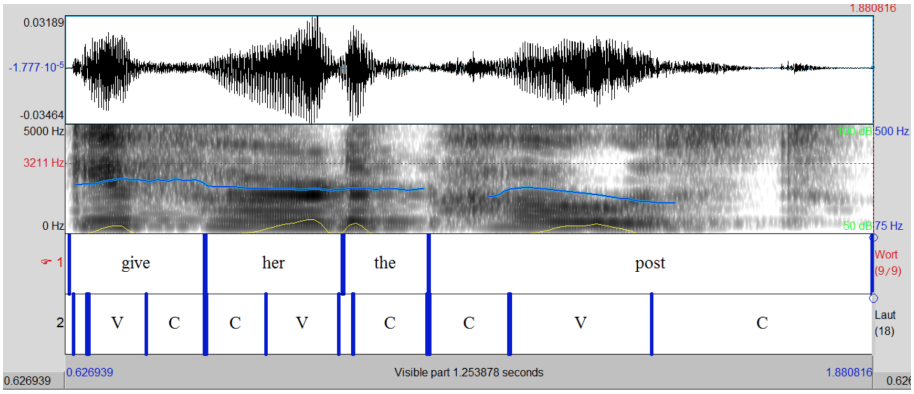


Figure 1.4: An example of specialized speech processing interface (Praat).

of speech signal. Therefore research-oriented editing tools such as Praat [10] and STRAIGHT [38] allow a user to view and manipulate acoustic and linguistic aspects of speech in a timeline where audio waveform is presented synchronously with annotations such as phonemes and pitch (Figure 1.4).

Based on similar design concept, recent research aims at connecting non-expert users with sophisticated speech processing tools using text and a middle layer: Whittaker and Amento presents a voicemail editing interface that boosts productivity with text-based navigation and word-level copy-and-paste. The idea of using a script to navigate a timeline is also employed by video authoring tools such as the work of Casares et. al. [14], and recent versions of commercial software such as Avid Media Composer. It is also adopted by video browsing and symmetrization tools such as Video Digest [63] and SceneSkim [62]. Text can also be used as audio editing medium: Berthouzoz et. al. created interview video editing tool that allows an editor to place cuts and pauses by cutting words and adding pauses in the transcript [6]. They also show "suitablility" scores as colors in the transcript to indicate suitable points to place cuts. Later, Rubin et. al. presented an editing tool for audio stories that support cut, copy and paste in text and selecting among different takes in the text [69]. However, none of existing work support insertion and replacement words that are not in the transcript, which is the focus of this thesis.

1.2.2 Text-to-speech synthesis

There are two main-stream text-to-speech synthesis techniques: unit selection and statistical parametric methods. Dating back to 1990s, unit selection-based methods first process audio samples into units (usually segmented by phonemes) and synthesize speech by selecting and concatenating units to match the text and desired prosody [31]. A general process is to select units that match the linguistic feature of the text input while maintaining smooth transition from the previous unit to the next. This is achieved by minimizing a cost function that can be optimized using Viterbi algorithm. Past research has evaluated different unit definition [41, 61], objective functions [53] and searching algorithms that provide better efficiency [8]. As a data-driven approach, the quality and naturalness depend heavily on data quantity and quality [7]. Expressive TTS systems are often built over large databases [22] that consist over 10 hours of voice samples, manually selected and refined. Though the unit annotation process can be automated using speech recognition techniques, human annotation is still required to refine mis-annotated units. Because of the high cost, unit-selection-based methods are incorporated in building high quality speech synthesizers for a small number of voices.

On the other hand, statistical parametric methods synthesize speech by generating acoustic features via a statistical models [91]. This statistical model represents a generative process that maps linguistic feature extracted from text to acoustic features, such as fundamental frequency (i.e.pitch or F0), and MFCC that represents timbre. It further relies on a Vocoder that extract acoustic features from waveform and resynthesize waveform based on generated acoustic features. The most popular method is based on HMM where the hidden variables represent linguistic features and the observation variables are acoustic features such as MFCC and F0 [102]. However, the result often contain muffled artifact caused by statistical model and unrealistic sounding Vocoder synthesis process. Later research has approached improving the

vocoder [70, 38], prosodic expressiveness [84] and HMM replacement such as deep neural networks [103]. One notable advantage of parametric speech synthesis is its higher resilience to user controls such as emotional strength [72]. It is also possible to train a generic HMM model across a range of speakers [82] and adapt to a specific speaker with small amount of recorded speech [101]. However, one persistent drawback of parameteric methods is audible artifact due to over-smoothed parameter trajectories [93]. The vocoder and the acoustic features defined by the vocoder introduces another layer of artifact as they are not perfect representation of human speech [99]. These problems stops parametric TTS to be used in word insertion and replacement as such artifacts are audible when blended with the original audio recording, while unit-selection-based TTS has much less artifact but requires large amount of data. To trade off between the benefits and problems associated with both methods, hybrid approaches are proposed. One can use HMM to inform unit selection targets [39] and costs [56]. However, such methods cannot reduce artifact completely.

With the rapid adoption of deep neural networks, researchers have been seeking end-to-end solutions to TTS, transforming linguistic notation or acoustic feature directly into waveforms without relying on a pre-defined vocoder. WaveNet is such neural network proposed by Deepmind that generates natural sounding speech sample by sample; the quality is unparalleled with parametric or unit-selection-based synthesis method and therefore has become the most favored method in modern TTS systems, for example Deep Voice [5, 4]. New applications have emerged, from converting letter to speech, to on-the-wild TTS system that mimic a voice with very limited samples. The WaveNet architecture is also adopted to solve other problems, such as voice conversion [42], voice enhancement [86], de-noising [67] and musical instrument synthesis [24]. Many of these efforts leverage WaveNet as a vocoder. Moreover, it has been demonstrated that WaveNet WaveNet converts Mel Ceptral and F0 back to waveform with unprecedented quality [73]. Unfortunately,

original WaveNet takes tens of minutes to synthesize a second of audio; fast synthesis techniques have been proposed to synthesize around 200 samples per second on a CPU [58], but it's still far from being real-time. On way to accelerate WaveNet scales down the model size so as to reach real-time with a high-end CPU, but at the cost of a substantial quality reduction (from 3.35 to 2.74 in the MOS test described in Section 3.5.2) [4].

1.2.3 Voice conversion

Voice conversion can be used to create new text-to-speech synthesis voices with relatively smaller data size [36]. Typical voice conversion methods represent voice signals as parametric source-filter models in which the source approximates the vocal cords that generate periodic or noise signals and the filter mimics the vocal tract that shapes the source signal into the sounds we hear [87]. Then these methods explicitly model a conversion function from one voice (source) to another (target) in the parametric space. Given a new source (query), they apply the conversion function and then re-synthesize actual signals from the converted parametric model. The most classical conversion function is based on mapping over Gaussian Mixture Models [90, 78]. Recent research has also explored probabilistic modeling [88], matrix factorization [2] and deep neural networks [15, 19]. These approaches achieved very low distortion in the parametric space but the output signal contains muffled artifacts possibly not captured by the distortion measurement or introduced when re-synthesizing from parametric representations of MFCC [19] or STRAIGHT [38]. The artifacts tend to be perceptible when inserting a converted word between words of an existing speech signal, and thus unsuitable for our word insertion application. More recent work adopts WaveNet to convert source acoustic features to target waveform direction [55] but the result sounds noisy and unstable for our application.

In another thread of voice conversion research, algorithms concatenate segments of a target speaker's voice to approximate the query; a method called **unit selection** is used to select these segments such that they are close to the source in content and have a smooth transition from one segment to the next. Recent approaches in this this thread differ in how units are defined and selected: for example Fujii et al. [27] segment speech into phonemes and concatenates these phonemes to form new content. Another thread of research performs frame-level unit selection (thereby increasing the number of units) [23] to overcome the shortage of voice samples. Wu et al. perform unit selection on exemplars [100], time-frequency speech segments that span multiple consecutive frames [65]. Although these methods produce words without muffled artifacts, they tend to have new problems with unnatural pitch contours and noise due to discontinuity between units. However, when concatenating less units and synthesizing short speech such as a word or phrase, these problems are less prominent.

1.3 Challenges

Both TTS and voice conversion are possible solutions for the word synthesis task within text-based speech editing. But the current state of either direction displays limitations: parametric TTS and voice conversion produces synthesis with robotic sounding artifact; unit-selection and deep learning based TTS produces more natural sounding audio but it requires much larger data (tens of hours) and computational power. Unit-selection-based voice conversion is likely to deliver less artifact and does not requires large amount of training data; but the current state of art still suffers from noises due discontinuity at the concatenation area. Realizing these limitations, this thesis focuses on improve the quality unit-selection-based voice conversion while seeking alternative solutions to deep-learning based TTS that require much less data (1 hour or less).

1.4 Thesis overview

This thesis builds on parallel voice conversion, in which two voices speaking the same sentences are provided as training data: the voice being converted is called source voice and the voice being converted to is called target voice. Both voices speak the same sentences in their own way and therefore the resulting speech recordings have different length for the same sentence. One common step of parallel-data voice conversion is to align parallel recordings frame by frame; each frame is represented using an acoustic feature such as MFCC and spectrogram. Conventional voice conversion maps the source frame's acoustic feature to its aligned target frame's acoustic feature using a statistical model. And finally, a vocoder [85] is used to synthesize waveform based on the converted target acoustic features. However, this approach has two major drawbacks: over-smoothing artifacts caused by over-simplification in the statistical model, and robotic sounding artifacts introduced by the vocoder. To solve this problem, this thesis presents three new methods: a data-driven voice conversion based on concatenating audio snippets of the target voice; a word synthesis algorithm that utilizes data-driven voice conversion and deep-learning-based method that learns the conversion function, the acoustic feature and the vocoder at the same time. Thus, this thesis is divided into three main parts:

Chapter 2 introduces CUTE, a data-driven voice conversion method. The name CUTE comes from four features of this voice conversion method: Concatenative, meaning the synthesis is the result of concatenating audio snippets found in the target voice; Unit-selection, meaning the target voice is organized into frames called units and audio snippets are determined by selecting among all the frames; Triphone-preselection, meaning the selection of frames is not based on all the frames but only the frames that corresponding to the phonemes that we aim to synthesize; and Exemplar, meaning the acoustic features we use are not acoustic feature of an individual frame

but rather $2k + 1$ frame features concatenated with a triangular weighting (called exemplar in speech recognition context [65]).

Chapter 3 introduces VoCo (short for Voice Conversion), a text-to-speech synthesizer dedicated to word insertion and replacement in text-based speech editing setting. When a new word is typed by a user, VoCo uses a existing TTS system to synthesize that word in a robot voice and perform data-driven voice conversion to synthesize the same word in the voice of the target speaker so that it can be seamlessly inserted into a real recording. VoCo also offers several advancements from the proposed CUTE method. The most notable improvement is called Range Selection algorithm that selects the starting and ending points of audio snippets directly via dynamic programming. This method is a replacement of unit selection algorithm that is conventionally used for speech synthesis and conversion. VoCo also introduces alternative syntheses based on range selection algorithm that allows a user to choose among different versions of synthesized words that varies in length and pitch contour. An advanced editing interface allows a user to further adjust the pitch and duration of each snippet or uses his/her own voice as input to modify the pitch and duration.

However, VoCo is unable to synthesize long passage and has a word synthesis success rate of 60% based on CMU ARCTIC dataset. Chapter 4 takes a different route and uses deep neural networks to perform voice conversion using. Two networks are introduced: the first is called FFTNet, a real-time neural vocoder that transforms acoustic features to waveform with high fidelity. The second is called TimbreNet that transforms the acoustic features of a source voice into the acoustic features of a target voice based on user-specified length and pitch contour. This method is called Deep-VoCo, short for Deep Voice Conversion.

Finally, Finally, Chapter 5 summarizes the work and proposes potential directions for future research.

This dissertation weaves together and elaborates on papers that were presented by this author at three conferences: IEEE ICASSP 2016 [33], and ACM SIGGRAPH 2017 [35], and IEEE ICASSP 2018 [34].

Chapter 2

CUTE: a data driven method for voice conversion

State-of-the art voice conversion methods re-synthesize voice from spectral representations such as MFCCs and STRAIGHT, thereby introducing muffled artifacts. We propose a method that circumvents this concern using concatenative synthesis coupled with exemplar-based unit selection. Given parallel speech from source and target speakers as well as a new query from the source, our method stitches together pieces of the target voice. It optimizes for three goals: matching the query, using long consecutive segments, and smooth transitions between the segments. To achieve these goals, we perform unit selection at the frame level and introduce triphone-based preselection that greatly reduces computation and enforces selection of long, contiguous pieces. Our experiments show that the proposed method has better quality than baseline methods, while preserving high individuality.

2.1 Background

The goal of voice conversion (VC) is to modify an audio recording containing the voice of one speaker (the *source*) so that the identity sounds like that of another speaker

(the *target*) without altering the speech content. Approaches to VC typically rely on a training set of parallel utterances spoken by both the source and target. State of the art **parametric methods** [83, 2] then explicitly model a conversion function mapping from the source to the target in some feature space such as MFCC [19] or STRAIGHT [37]. A new source utterance (the *query*) may be transformed into the feature space and then mapped through the conversion function to match the target. The output of such parametric methods must be re-synthesized from these features, and artifacts are inevitable since these feature spaces do not perfectly model human voice. Thus, the converted speech usually has a muffled effect [77] as a result of re-synthesis.

Voice conversion can be used in many applications such as generating speech synthesis voices with small samples [36] and bandwidth expansion [59]. In order to avoid artifacts due to re-synthesis, an alternative to the parametric approach relies on **unit selection**. The basic idea is to choose segments of the target speaker's training samples whose corresponding source samples sound like the query, while also seeking smooth transitions between neighboring segments. Modern text-to-speech synthesis systems [87] demonstrate that unit selection can generate high quality speech with high individuality, which is crucial for VC. These systems require very large training sets (many hours up to days) as well as substantial human annotation. Yet, in typical VC applications, we have a limited training set (such as one hour) and usually no manual effort.

Researchers have investigated several approaches for overcoming these limitations in the context of unit selection. For example, the method of Fujii et. al. [27] and Sndermann et. al. [80] relies on automatic speech recognition to generate phoneme level annotations for units. As noted by the former, segmentation errors among the phonemes yield significant artifacts. Another line of research reduces the need for

very large datasets by performing frame-level unit selection (thereby increasing the number of units) [23].

Wu et. al. further improve frame-level unit selection using exemplars [100], defined as time-frequency speech segments that span multiple consecutive frames [65]. We find that this method greatly reduces distortion from frame-based unit selection, but there remains a significant quality gap between this approach and phoneme-level unit selection where substantial user annotation is involved.

We introduce a hybrid method called CUTE that leverages four major components described in various contexts in the literature:

- Concatenative synthesis
- Unit selection
- Triphone pre-selection
- Exemplar-based features

This combination offers a significant improvement over previous approaches such as phoneme-based and exemplar-based unit selection. The basic idea of our method is to concatenate segments from the target speaker's voice so that they match the query and form long consecutive segments, with smooth transitions between them. To have sufficient units and flexibility in choosing the segments, we define unit selection at the frame level. To enforce smooth transition, we compute features using exemplars, concatenated spectral representations of multiple consecutive frames. To better match the query prosody, we exchange the $F0$ contour of source and target voices so that the query $F0$ is transformed and matched to the target $F0$ directly. Finally, we introduce a triphone preselection method to sift the candidate list before unit selection, ensuring large consecutive segments to be preserved in the candidate list. To evaluate our method, we performed two listening tests with human subjects: an XAB preference test and an ABX individuality test. The results show that our

method offers significantly higher quality than two other unit selection based methods, and that high individuality is maintained. The contributions in this chapter are:

- Adapting exemplar-based unit selection framework for concatenative synthesis.
- Using phonetic information to improve exemplar-based unit selection for voice conversion.
- Experiments showing that the proposed method has better quality than baseline methods, while preserving high individuality.

2.2 Exemplar-based unit selection for concatenate synthesis

2.2.1 Pairing

Given the source speaker's frame sequence $X = \{x_1, x_2, ..., x_N\}$ in the time domain and the target speaker's frame sequence $Y = \{y_1, y_2, ..., y_M\}$, we first perform dynamic time warping (DTW) to align the source voice to the target in order to obtain frame-wise pairings $\mathcal{A} = \{(x_{n_1}, y_{m_1}), (x_{n_2}, y_{m_2}), ..., (x_{n_K}, y_{m_K})\}$ where for $n, m = 1, 2, ..., k$, $n_k \in [1..N]$ and $m_k \in [1..M]$. We then construct a mapping function $r(j) = Y_j$ where $Y_j = \{y | n_k = j, (x_{n_k}, y) \in \mathcal{A}\}$. This is the mapping we use to translate pairings to actual frames.

2.2.2 Exemplar feature extraction

Having multiple parallel frames, we define an *exemplar frame* as two parallel sequences of source and target frames with the central frame aligned. With a central frame

(x_{n_k}, y_{m_k}), an *exemplar frame* is defined as:

$$\begin{pmatrix} x_{n_k-t} & \dots & x_{n_k-1} & x_{n_k} & x_{n_k+1} & \dots & x_{n_k+t} \\ y_{m_k-t} & \dots & y_{m_k-1} & y_{m_k} & y_{m_k+1} & \dots & y_{m_k+t} \end{pmatrix} \tag{2.1}$$

We define our exemplar as a concatenation of the weighted features of the member frames. Suppose a parallel frame (x_{n_k}, y_{m_k}) has feature vector f_{n_k,m_k}, then an exemplar is:

$$[w_{-t} f_{n_k-t,m_k-t}, ..., w_0 f_{n_k,m_k}, ..., w_t f_{n_k+t,m_k+t}]^T \tag{2.2}$$

The weights $w_{-t:t}$ are used to attenuate the influence of non-central frames, emphasizing the central frame. A triangule-shaped window function may be used for weights:

$$w_i = ((t - |i| + 1)/(t + 1))^\beta \tag{2.3}$$

Here $\beta > 0$ is used to control the amount of attenuation - the higher the value, the less emphasis on the non-central frames. An *exemplar* has better temporal resolution than a large frame, which allows us to model transition between phonemes. Using exemplars in unit selection also improves smoothness because similar exemplars share similar contexts; when concatenating these frames using overlap-add, the overlapping region is likely to produce less artifacts.

In this work, we define the weight function as a triangular function ($\beta = 1$) and we use two sets of exemplars: (1) matching exemplars includes the MFCCs of the source frames concatenated with the target frames' F0 (in log space), denoted as $\{A_1, ..., A_K\}$; we use these exemplars to match the spectral envelope and prosody of the query. (2) concatenation exemplars are formed by the MFCCs and the F0 of the target frames, denoted as $\{B_1, ..., B_K\}$. These exemplars are used to enforce transition smoothness between the selected frames.

2.2.3 Pre-selection of candidates

The next several steps involve finding the exemplars that match the query. The first step is to construct exemplars from $\{q_1, .., q_S\}$ and calculate the exemplar features $\{Q_1, ..., Q_S\}$ using the matching feature descriptors we defined in section 2.2.2. Because the source $F0$s are in a different range of the target $F0$s, we adapted logarithm Gaussian normalized transformation [19] to convert query $F0$ contour to the target's:

$$\log p_{\text{conv}} = \mu_{\text{tgt}} + \frac{\sigma_{\text{tgt}}}{\sigma_{\text{scr}}}(\log p_{\text{scr}} - \mu_{\text{scr}}) \tag{2.4}$$

where μ_{tgt} and σ^2_{tgt} are the mean and variance of the target voice's log F0s; μ_{scr} and σ^2_{scr} are the mean and variance of the source voice's log F0s. p_{scr} and p_{conv} are the query and the converted F0s.

Then, for each query exemplar, we want to select one exemplar from the dataset that resembles the query and makes a smooth transition to the next exemplar. The Viterbi algorithm can be used for this purpose. However, it does not scale well with the number of states (exemplars) [26]. We pre-select the candidates per query exemplar in order to reduce run-time computation. One way to do this is to compute K nearest neighbors and choose the candidates that are closest to queries [100]. In this work, we found that pre-selecting exemplars that share the same phoneme as the query exemplar generates better result. We will present our approach in Section 2.3. Denote the candidates (indices of exemplars) for Q_k as $\{c_{k1}, c_{k2}, ..., c_{ku_k}\}$ where u_k is the number of candidates for the k-th query exemplar.

2.2.4 Optimization

The objective of matching is to select one candidate source frame per query so that the corresponding target frames when concatenated together speak the same word as the query and at the same time sound natural without artifacts, noise or unnatural

prosody. To translate that into an optimization scheme, we want to minimize the target cost $\mathcal{T}(q_s, A_{c_{s,j}})$ between the query q_s and a candidate exemplar with index $c_{s,j}$ and concatenation cost $\mathcal{C}_s(B_{c_{s-1,i}}, B_{c_{s,j}})$ between neighboring candidates with subscripts $c_{s-1,i}$ and $c_{s,j}$:

$$\begin{aligned} \min_{d_1,\ldots,d_S} & \mathcal{T}(Q_1, A_{c_{1,d_1}}) \\ & + \sum_{s=2}^{S} \mathcal{T}(Q_s, A_{c_{s,d_j}}) + \mathcal{C}_s(B_{c_{s-1,d_{j-1}}}, B_{c_{s,d_j}}) \end{aligned} \tag{2.5}$$

Note that different exemplar features are used for the matching and the concatenation costs. One difference between this scheme and previous unit selection method is that concatenation cost function varies over s. In other words, some statistics about the query q_s is used in calculating the cost function. Here are the equations we use to calculate costs

$$\mathcal{T}(Q, A) = \|Q - A\|_2 \tag{2.6}$$

$$\mathcal{C}_s(B_i, B_j) = d(B_i, B_j)\mathcal{P}(i, j)\mathcal{R}(E(q_s)) \tag{2.7}$$

In equation 2.7, $d(B_i, B_j)$ is defined as the Euclidean distance between the first $2t$ frames in exemplar B_i and the second last $2t$ frames from exemplar B_j, modeling the smoothness at the overlapping region between frames. The second term is defined as break costs, which prefers candidates that are originally neighbors; it also allows skips and repeats, adding more flexibility in the query length.

$$\mathcal{P}(i, j) = \begin{cases} 0, & j - i = 1 \\ c_s, & j - i = 2 \\ c_r, & j - i = 0 \\ c_b, & \text{else} \end{cases} \tag{2.8}$$

c_s, c_r and c_b are skip, repeat and break penalty multipliers. Through experiment, we found 10, 2, 2 are reasonable choices for c_b, c_r and c_s. The last term $\mathcal{R}(E(q_s))$ is penalty reduction considering the significance of the query. $E(q_s)$ extracts the features for significance and in our experiment we define it as energy times periodicity extracted using YIN [16] algorithm. It means we prefer breaks at silences and noisy parts such as background noise or unvoiced consonants. Mapping $\mathcal{R}$ normalizes the value to be in [0,1].

2.2.5 Translation and Concatenation

The last step is to convert exemplars to actual frames. Suppose we have selected candidates with indices $\{c_1, ..., c_s\}$, we first remove the repeats and skips and then translate them into target frames using the function $r(j) = Y_j$ defined in section 2.2.1. Then we concatenate elements in sequences $(Y_{c_1}, Y_{c_2}, ..., Y_{c_s})$ to obtain a series of frames. We obtain the final result using overlap-add [87].

2.3 Triphone preselection

We have observed two limitations in the above mentioned method: first, when the query contains unfamiliar prosody (strong emphasis, high or low $F0$s), the query exemplars might lack true nearest neighbors in the dataset and therefore are likely to be matched to target exemplars with incompatible phonemes. Secondly, if a different utterance of the same query word exists in the dataset, the above mentioned method fails to extract the whole word most of the time. We think these problems have to do with the fact that spectral envelope is not completely independent of prosody, especially at extreme $F0$s and emphasis. As a result, different utterances of the same word might have high gross spectral distances with each other, making several frames of the word not appearing in the K nearest neighbors. To encourage concatenating

long segments and reduce the negative effects of extreme prosody, we propose using phonetic information to preselect the candidates.

2.3.1 Pre-selection using phonemes

To obtain phoneme segmentation, we first translate the transcripts into phoneme sequences and then apply forced alignment to align phonemes to the target speaker's voice [74]. For each pair of source and target frames, we annotate it with a phoneme label, such as "AH" or "P". Then for each exemplar, we label it with its central frame's phoneme. In the pre-selection step, for each query exemplar, we include only the candidates that share the same phoneme.

Using phoneme pre-selection, we guarantee that if other utterances of the same query word exist in the dataset, their frames are included in the candidate lists. With a proper break cost, we can extract a full word from the target speech to match the query. This overcomes the issue faced by KNN pre-selection. Unit selection at the frame level with phoneme pre-selection also outperforms phoneme-level unit selection because it allows breaks in the middle of a phoneme, making it possible to form longer or shorter phones. This approach is also robust to phoneme segmentation errors since an incorrectly segmented frame is often dissimilar from the query and thus excluded in the optimization step. The drawback of this method however is that the candidate table gets very large for frequent phonemes such as vowels; because the complexity of Viterbi algorithm scales with the squared size of candidates, this method seems impractical for large datasets. Even if we select the K nearest candidates after phoneme preselection, there is still chance of omitting frames that potentially make up long sequences. Our solution to this drawback is to use Triphones to pre-select the candidates.

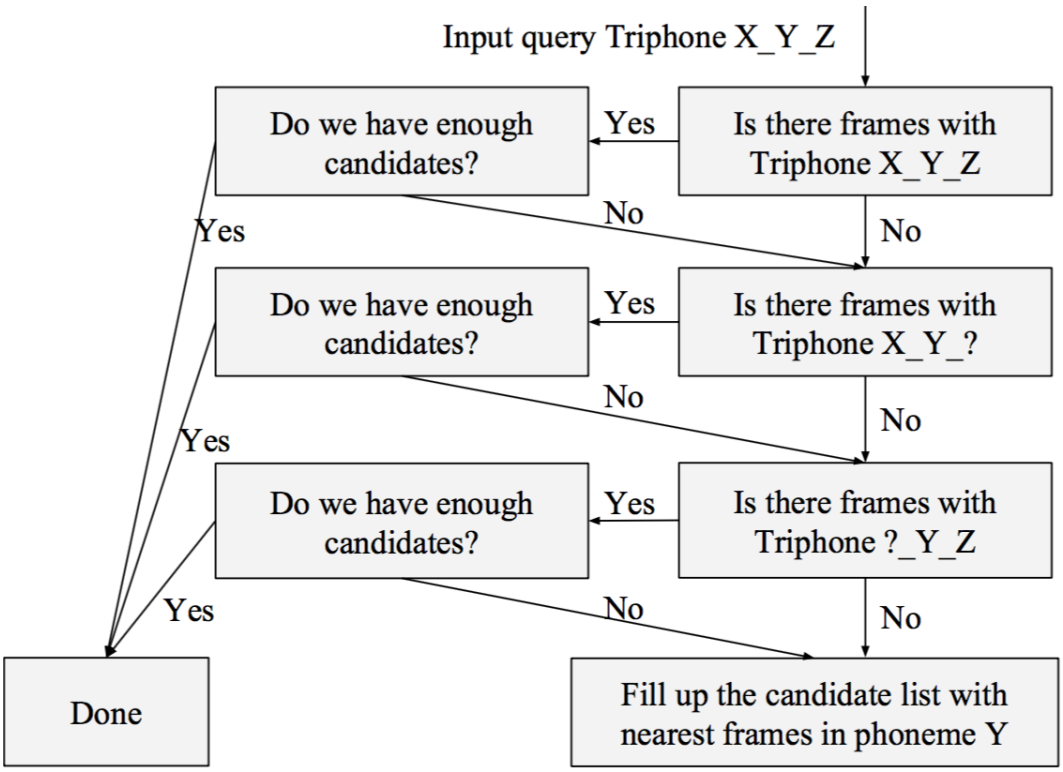


Figure 2.1: Decision graph for candidate preselection based on triphones

2.3.2 Decision graph for triphone pre-selection

A triphone is a three-phoneme sequence, composed of a center phoneme and its two immediate neighbors. In tasks such as speech recognition and text-to-speech synthesis, triphones are superior to phonemes as they capture contextual information [64]. To preselect exemplar candidates, we propose labeling frames with triphones instead of individual phones. Although the triphone vocabulary is large, the number of instances per triphone is small. Therefore, we adopt a simple scheme to deal with triphones nonexistent in the dataset: first, we look for candidates with exact triphones; if we cannot find enough candidates (e.g. less than 10), we look for candidates with similar diphones; and if we again fail to find sufficient candidates, we apply monophone-based preselection and only include frames that are closest to the query. One may also adopt decision trees used in speech recognition [30] to inform triphone selection procedure.

This preselection scheme has several merits: (1) The frames in the target dataset having exactly the same phoneme sequence with the query will be present in the candidate table. (2) Moreover, we can get away with only a few candidates per query frame - following the decision table, if we find a correct triphone instance, we include all candidate frames in that triphone; if not we look for diphones and include all of them if they exist. If no matching diphone is found, we can include only a small number of frames that has correct phoneme because there will be a break at this point anyway. In this way, the number of candidates can be kept small resulting in faster Viterbi computation. (3) Using the decision graph, the synthesis quality improves with increasing amounts of data: the larger the dataset, the more likely we have triphones in the candidate list.

2.4 Evaluation and Discussion

This section describes subjective and objective evaluations of our method, in comparison with two other baseline methods in the literature. We use CMU arctic corpus for these experiments [43]. We examined four conversion schemes based on four voices: DBL-male to RMS-male (m2m), DBL-male to SLT-female (m2f), SLT-female to RMS-male (f2m), and SLT-female to CLB-female (f2f). The training data is composed of 800 utterances, while 20 utterances are used for evaluation. Transcripts are aligned to utterances via forced alignment implemented in `p2fa-vislab`[1]. No manual segmentation or adjustment is performed in these experiments, so segmentation errors are prevalent.

We obtain frames using a 25ms analysis window and a 12.5ms hop size. To extract exemplar features, we use 24-coefficient MFCCs, excluding the 0th energy coefficient, and the YIN algorithm [16] for F0 detection. We compare our method (CUTE) with two other unit selection voice conversion methods:

[1] https://github.com/ucbvislab/p2fa-vislab

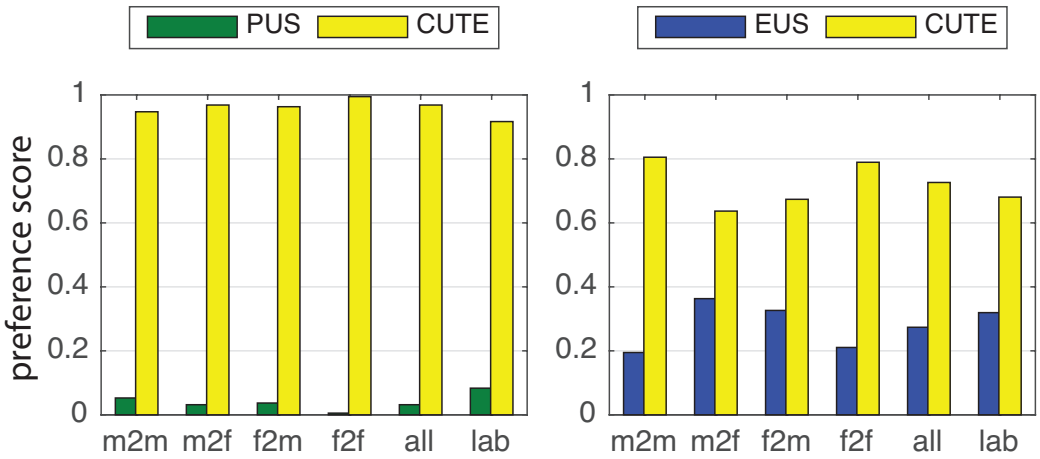


Figure 2.2: XAB test results. Our method (CUTE) has better quality than two baseline methods (PUS and EUS – similar to those found in the literature). Label 'all' refers to aggregating the four preceding gender settings, while 'lab' refers to a smaller experiment performed in the lab. Note that p-values ≈ 0 in all cases except for EUS-m2f, EUS-f2m, and EUS-lab with p-values $< 10^{-4}$.

1. Phoneme-level unit selection (PUS) [27]: We used the same phoneme segmentation to extract the units. Unlike the experiments in the original paper, we did not perform manual adjustment to phoneme segmentation. This is a baseline used to show that using frame-level concatenation and phonetic pre-selection can overcome segmentation problems.

2. Exemplar-based unit selection (EUS) [100]: This method is similar to our method without triphone-based preselection and penalty reduction terms in the Viterbi algorithms. In our experiments, we notice that noisy discontinuities are substantially reduced via triphone-based preselection.

2.4.1 Subjective Evaluation

To compare the quality of the proposed method to the two baseline methods, we conducted a subjective evaluation using an XAB test, as follows. We shuffle the order of the 20 utterances, of which 16 are used to compare methods, and the remaining four are used as a validation described below. For each utterance, we randomly select

one out of four voice conversion settings (m2m, m2f, f2m, and f2f) and one of the two baseline methods. The subject is shown three tracks: the reference track (X) followed by two other tracks (A&B) – CUTE track and the baseline method, in random order. A subject is asked to choose which sample (A or B) sounds more like the reference track X. In the four validation tests, the A&B tracks (in random order) are our method and a duplicate of the reference track X, which should easily be selected as the preferred track. We assume that when a subject fails any validation tests they are not paying careful attention, and we omit the data for all 20 utterances from that experiment. For load-balancing on conversion tasks and methods, we ensure that each assignment will contain all permutations of (conversion settings, method) twice and validation tests once for each conversion setting.

We recruited subjects on the Amazon Mechanical Turk (MTurk), a crowdsourcing platform that has become a common tool for perceptual and other human subjects experiments [52]. In each task (called a HIT) the subject provided the 20 XAB answers described above. Of 100 HITs we retain 95 valid results with 1520 (95×16) individual responses. As seen in Figure 2.2, our method (CUTE) is almost always preferred when comparing to phoneme-level unit selection (PUS), with p-values ≈ 0 in all four cases. The lower quality of PUS is possibly due to segmentation errors and insufficient units. CUTE is also strongly preferred over the exemplar-based unit selection method (EUS) for same-gender voice conversion. For cross-gender tests, the margin is smaller, but significant.

In order to corroborate these outcomes obtained on MTurk, we also ran a duplicate experiment with only a dozen subjects recruited in the research lab at Adobe. Of these, only 9 passed the validation test, insufficient data to break out the four separate gender settings. Nevertheless, the aggregate results (labeled 'lab' in the figure) were similar to the aggregate from MTurk (labeled 'all').

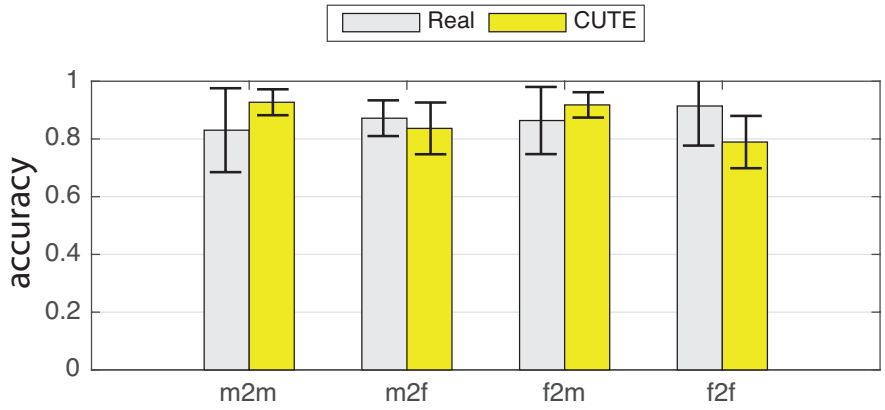


Figure 2.3: ABX individuality test results. The performance of subjects is compared between identifying voices for real samples and synthetic samples generated using the proposed method.

We hypothesize two reasons for the significant improvement in CUTE: first, CUTE overcomes phoneme segmentation errors by selecting new concatenation points: if the query has two phonemes and in the candidate table there is no consecutive paths between them, CUTE finds the best concatenation points within both phonemes that produces the least cost. The new cut points are not necessarily at the recognized boundaries that may deviate from the true boundary. Secondly, with triphone preselection, we can reduce noise of concatenating small segments by enforcing less breaks with a large break penalty. Triphone preselection ensures that these larger segments are phonetically consistent with the source. Note that having a large break penalty is not effective in pure frame-level unit selection as it over-smooths the distance and thus may select chunks that are not phonetically consistent.

We also conducted 100 ABX tests on MTurk, in order to test if our method preserves individuality claimed in the phoneme-level unit selection method [27]. In these tests, a subject is presented with Person A's and Person B's voice samples (same gender) and then an unknown sample X. The task is to choose which voice (A or B) is that of the sample X. In these tests, X can be either an actual voice sample from A or B, or generated using our proposed method (CUTE). As shown in Figure

2.3, the subjects have similar performance on generated samples and on real samples, meaning individuality is well preserved in CUTE.

The audio clips and results of the experiments described in this section may be found at `http://voxogram.com/CUTE/`.

2.4.2 Objective Evaluation

As an objective evaluation, we computed Mel-cepstral distortion (MCD) [88] to measure the distortion between the target voice sample and the converted samples. The results contradict the subjective evaluation results – MCDs of samples generated by our method are about 1dB higher than those generated by EUS even though the synthesized samples have better perceptual quality verified in the subjective listening tests. We hypothesize two explanations: (1) we use query F0s to control the output and therefore the prosody may vary from the target sample; as noted before, the spectral envelope may vary with prosody – a natural sounding sample can have high distance to other versions of it. (2) MCD is an over-smoothed measure of the spectral envelope, artifacts and noises are evened out. We found that having a low break cost would result in much lower MCD, but because of many breaks, the output sounds noisy.

2.5 Summary and Discussion of CUTE

We have proposed a concatenative voice conversion method based on exemplars and phonetic pre-selection of units. We defined two types of exemplars, target exemplar and concatenation exemplar, to allow controlling the prosody with source examples and enforce concatenation smoothness in the target samples. Using phoneme information to pre-select the phonemes, we ensure the longest possible phonetically correct segments to be used in concatenative synthesis. Experiments demonstrate that

our CUTE method has better quality than previous exemplar-based voice conversion methods and high individuality comparable to real samples. One limitation is that it still suffers from the lack-of-data problem faced by phoneme-based unit selection – sometimes the result is phonetically correct but unsatisfactory in prosody due to the lack of samples with appropriate F0s. It also relies on speech recognition for phoneme segmentation. Future work includes refining pre-selection to balance phonetic correctness and prosodic continuity.

Chapter 3

VoCo: Text-based Insertion and Replacement in Audio Narration

Editing audio narration using conventional software typically involves many painstaking low-level manipulations. Some state of the art systems allow the editor to work in a text transcript of the narration, and perform select, cut, copy and paste operations directly in the transcript; these operations are then automatically applied to the waveform in a straightforward manner. However, an obvious gap in the text-based interface is the ability to type new words not appearing in the transcript, for example inserting a new word for emphasis or replacing a misspoken word. While high-quality voice synthesizers exist today, the challenge is to synthesize the new word in a voice that matches the rest of the narration. This chapter presents a system that can synthesize a new word or short phrase such that it blends seamlessly in the context of the existing narration. Our approach is to use a text to speech synthesizer to say the word in a generic voice, and then use voice conversion to convert it into a voice that matches the narration. Offering a range of degrees of control to the editor, our interface supports fully automatic synthesis, selection among a candidate set of alternative pronunciations, fine control over edit placements and pitch profiles, and even

guidance by the editor's own voice. The chapter presents studies showing that the output of our method is preferred over baseline methods and often indistinguishable from the original voice.

3.1 Introduction

Recorded audio narration plays a crucial role in many scenarios including animation, computer games, demonstration videos, documentaries, and podcasts. After narration is recorded, most of these applications require editing. Typical audio editing interfaces present a visualization of the audio waveform and provide the user with standard select, cut, copy and paste operations (in addition to low-level operations like time stretching, pitch bending, or envelope adjustment), which are applied to the waveform itself. Such interfaces can be cumbersome, especially for non-experts. Researchers have addressed this problem by aligning the waveform with a transcript of the narration, and providing an interface wherein the user can perform cut-copy-paste operations in the text of the transcript. Whittaker and Amento [98] and Rubin et al. [69] show that this form of interface significantly reduces search and editing time, and is preferred by users. State of the art video editing systems incorporate audio-aligned transcript editors to facilitate search and editing tasks [6, 14].

While cut-copy-paste operations are supported, one aspect remains conspicuously missing from text-based audio editors: insertion. It is easy for a person to type a new word not appearing in the transcript, but it is not obvious how to synthesize the corresponding audio. Nevertheless, in many circumstances inserting a new word or phrase during editing would be useful, for example replacing a misspoken word or inserting an adjective for emphasis. It is possible to record new audio of just the missing word, but to do so requires access to the original voice talent. Moreover, even when the original narrator, microphone and acoustic environment are available

for a new recording, it remains difficult to match the audio quality of an inserted word or phrase to the context around it. Thus the insertion is often evident in the edited audio, even while methods like that of Germain et al. [28] can ameliorate such artifacts. Regardless, just as it is easier to type than to edit audio waveforms for cut and paste operations, it is also easier to type for insertion or replacement rather than record new audio.

This chapter introduces a method for synthesizing a word or short phrase to be inserted into a narration, based on typing. The input is an existing recording coupled with a transcript, as well as the text and location of the word to be inserted. With the proliferation of voice-based interfaces, there now exist many high-quality text-to-speech (TTS) synthesizers. The challenge in our problem is to automatically synthesize the inserted word in a voice that sounds seamless in context – as if it were uttered by the same person in the same recording as the rest of the narration (the *target* voice). While it is possible to customize a speech synthesizer for a specific target, conventional approaches (e.g., Acapela Group [1]) start from a large corpus of example recordings (e.g., 10 hours) and require many hours of human annotation. Our goal is to synthesize a word or short phrase (as opposed to a full sentence) that reasonably matches the target voice in the context into which it is inserted, but based on a much smaller corpus that lacks human annotation. Moreover, the synthesized voice should have *clarity* (be free of noticeable artifacts like popping or muffling) and *individuality* (sound like the target).

The key idea of our approach is to synthesize the inserted word using a similar TTS voice (e.g., having correct gender) and then modify it to match the target using *voice conversion* (making an utterance by one person sound as if it was made by another). Voice conversion has been studied for decades, and the idea of using voice conversion for TTS was proposed by Kain and Macon [36] in the 1990s. However, only recently have voice conversion methods been able to achieve high individuality together with

substantial clarity, for example the recent CUTE algorithm of Jin et al. [33]. Based on a triphone model of human voice, CUTE performs a dynamic programming optimization to find and assemble small snippets of the target voice from the corpus, such that when concatenated they resemble the TTS word. This approach has related ideas from computer graphics research, for example the *image analogies* method of Hertzmann et al. [29] and the *HelpingHand* approach of Lu et al. [48].

This chapter introduces a method called VoCo that builds on CUTE and makes several improvements suitable for our problem: (1) it improves synthesis quality by introducing *range selection* to replace frame-level unit selection; (2) to accelerate the optimization for use in our interactive application, it introduces a new two-stage optimization approach (dynamic programming for selecting phoneme sequences, followed by *range selection* to choose audio frames that match those phonemes); (3) it introduces the notion of *exchangeable triphones* to achieve clarity with a smaller corpus (20-40 minutes) than earlier methods; (4) it optimizes matching the context of the insertion; and (5) for cases where the default output is unsatisfactory in quality or prosody, it supports interfaces by which novices and/or experts can improve the results – by choosing among a variety of alternative versions of the synthesized word, adjusting the edit boundaries and pitch profiles of the concatenated audio clips, and even adjusting the synthesis using the editor's own voice.

We present studies showing that words synthesized by VoCo are perceived as more natural than those produced by baseline methods (TTS and CUTE). Moreover, the studies show that VoCo produces output that, more often than not, is indistinguishable from the voice of the target speaker when inserted into a sentence. These studies are conservative in the sense that the goal of being indistinguishable from the target voice (for careful listeners) is stronger than the goal of being plausible or acceptable in edited narration (for casual listeners). Finally, we describe another study wherein an expert audio engineer painstakingly assembles snippets of the corpus

to synthesize new words, and even this arduous process produces results that are audibly inferior those of VoCo.

3.2 Related Work

Our approach offers several connections to work in **computer graphics**. The optimization over triphones described in 3.4.3 builds on the seminal *Video Rewrite* work of Bregler et al. [11]. Likewise, research like that of Stone et al. [76] and Levine et al. [45] shows how natural gestures of animated characters can be driven by speech. Finally our approach is closely related to work that relies on optimization for example-based synthesis of textures over a scalar variable like time or arc-length [48, 49].

Audio editing tools such as Adobe Audition and Audacity allow experts to edit waveforms in a timeline interface. Audio manipulation is achieved by chaining low-level signal processing tools that require expertise and knowledge to use. However, such general-purpose editing tools typically do not have a specialized **speech processing** interface, as they are unaware of the linguistic properties of the signal. Therefore research-oriented speech editing tools such as Praat [10] and STRAIGHT [38] allow a user to manipulate phonetic and linguistic aspects of speech in a timeline where audio is presented synchronously with other properties such as phonemes and pitches. Similar to this idea, recent research in this scope aims at connecting average users with sophisticated speech processing tools using text and a middle layer: Whittaker and Amento presents a voicemail editing interface that boosts productivity with text-based navigation and word-level copy-and-paste. The idea of using script to navigate a timeline is also employed by video authoring tools such as the work of Casares et. al. [14], and recent versions of commercial software such as Avid Media Composer. It is also adopted by video browsing and symmetrization

tools such as Video Digest [63] and SceneSkim [62]. In speech editing, text is not only used for browsing but editing as well: Berthouzoz et al. [6] created interview video editing tool that allows an editor to place cuts and pauses by cutting words and adding pauses in the transcript. They also show "suitablility" scores as colors in the transcript to indicate suitable points to place cuts. Later, Rubin et al. [69] presented an editing tool for audio stories that support cut, copy and paste in text and selecting among different takes in the text. One important aspect of text-based speech editing, however, is missing from these approaches – word replacement and insertion, crucial for words that do not exist in the transcript. This is the main focus of our work.

Kain and Macon [36] showed how to create new TTS voices with relatively little data using **voice conversion**. Typical voice conversion methods represent voice signals as parametric source-filter models in which the source approximates the vocal cords that generate periodic or noise signals and the filter mimics the vocal tract that shapes the source signal into the sounds we hear [87]. Then these methods explicitly model a conversion function from one voice (source) to another (target) in the parametric space. Given a new source (query), they apply the conversion function and then re-synthesize actual signals from the converted parametric model. The most classical conversion function is based on mapping over Gaussian Mixture Models [90, 78]. Recent research has also explored other forms of probabilistic modeling [88], matrix factorization [2] and deep neural networks [15, 19]. These approaches achieved very low distortion in the parametric space but the output signal contains muffled artifacts possibly not captured by the distortion measurement or introduced when re-synthesizing from parametric representations of MFCC [19] or STRAIGHT [38]. The artifacts tend to be perceptible when inserting a converted word between words of an existing speech signal, and thus unsuitable for our word insertion application.

In another thread of voice conversion research, algorithms concatenate segments of a target speaker's voice to approximate the query; a method called **unit selection** is used to select these segments such that they are close to the source in content and have a smooth transition from one segment to the next [31]. Recent approaches in this this thread differ in how units are defined and selected: for example Fujii et al. [27] segment speech into phonemes and concatenate these phonemes to form new content. Another thread of research performs frame-level unit selection (thereby increasing the number of units) to overcome the shortage of voice samples [23]. Wu et al. [100] perform unit selection on exemplars, time-frequency speech segments that span multiple consecutive frames [65]. Although these methods produce words that are free from muffled artifacts, they tend to have new problems with unnatural pitch contours and noise due to discontinuities between units. The recent CUTE method Jin et al. [33] further reduces distortion by a hybrid approach using phoneme-level pre-selection and frame-level unit selection. Our method VoCo builds on CUTE and has several improvements for better performance in word insertion.

3.3 Editing Interface

Our goal is to make editing narrations as simple as text editing for both novices and experts, while providing a range of customization options for motivated and advanced editors. To achieve this goal, our interface inherits the basic cut-copy-paste functionality of the speech editor of Rubin et al. [69]. In addition our interface allows a user to type new words (via insert or replace) and then automatically synthesize and blend them into the audio context. A non-expert user can customize the synthesized words by selecting among a number of alternative results. More experienced editors can adjust the length, pitch and amplitude of the stitched audio snippets to further refine the synthesis.

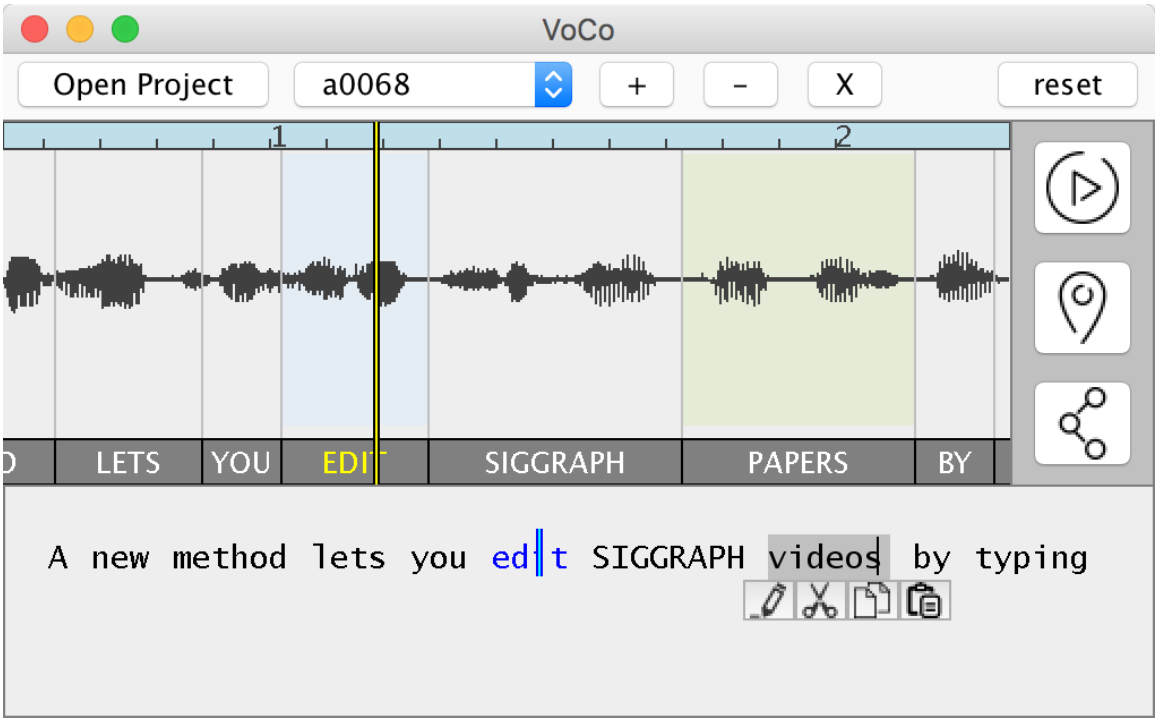


Figure 3.1: Main Editing Interface. An editor can cut, copy and paste text in the transcript in order to effect changes in the narration. Our main contribution is that the editor can also replace existing words or type new words not in the narration (in this case replacing *papers* with *videos*) and our system will synthesize a result that matches the voice of the speaker.

3.3.1 Text-based Editor

To start, a user selects an audio recording and its corresponding transcript. Our system aligns the loaded audio recording with the transcript and builds a customized voice synthesizer based on the voice characteristics of the selected recording (Section 4.1.2). Then, the main editing interface launches, shown in Figure 3.1. The upper part visualizes the audio waveform segmented by individual words. The lower part displays the transcript and provides a text pad in which users can perform edits. When playing back the audio, two cursors appear in synchronized locations in both the audio and the text. A user can move the cursor by clicking on either the waveform or the text. Basic operations including *delete, cut, copy* and *paste* are allowed in the text editor. These operations will be reflected immediately in the audio visualizer.

The primary contribution of this paper is that a user can also insert and replace words via typing; the synthesizer will then synthesize the new words in the voice of the person in the given recording and blend them into context as seamlessly as possible. An edit button will appear and can be clicked if the user chooses to customize the synthesis.

3.3.2 Alternative Syntheses

There are multiple ways to speak the same word, and therefore it is natural to provide a user with alternative synthesis results. As shown in Figure 3.2, the alternative results are organized in a list. Via radio buttons, the user can quickly listen to all the alternatives, and then optionally change modes (by a checkbox) to listen to how each alternative sounds in the context of the full sentence. Studies presented in 3.5 show that in general allowing the user to select among alternatives produces a more natural sounding result.

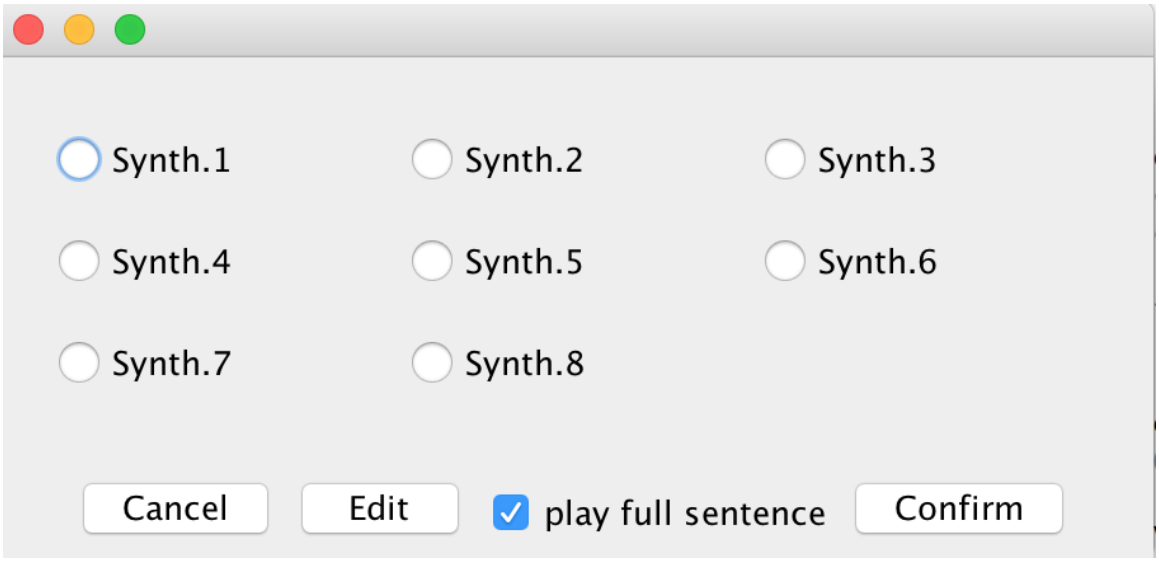


Figure 3.2: Interface presenting alternative results. When the editor is not satisfied with the default synthesis, this interface allows them to explore and choose among several alternative pronunciations.

3.3.3 Manual Editing

To allow users with audio editing skills to customize the synthesis result, we introduce two advanced editing options which can be applied sequentially. Recall that the synthesized word is created by stitching together snippets of words from the target voice corpus. By clicking on the *Edit* button in the *alternative syntheses* window, a user is presented with a new window containing multiple audio tracks that depict these snippets. In Figure 3.3, a user can adjust the boundaries of the snippets, change the lengths and find new stitching points. After previous changes are confirmed, a second window appears (Figure 3.4) that allows the user to adjust the pitch contour of each snippet (shown as a graph of dots) either by dragging handles or by choosing one of a pre-selected set of pitch profiles shown at the bottom of the window. This window also allows the editor to adjust the amplitude and/or duration of each snippet by dragging the top or sides of the box that surrounds it. Finally, as an alternative to manually adjusting the pitch and timing, a user can speak into a microphone to *demonstrate* how a target word should be spoken. The target word will then be re-synthesized taking into account the users' pitch and timing features. Studies presented in 3.5 show that manual editing can further improve on words selected via alternative syntheses.

3.4 Algorithms

Our word replacement and insertion tasks require synthesizing new audio based on typed text, a process often referred to as text-to-speech (TTS) synthesis. State of the art commercial TTS systems rely on a large corpus with a significant amount of manual annotation [93], neither of which is available in our setting. In order to build a *target TTS* synthesizer for an arbitrary target voice with a relatively small amount

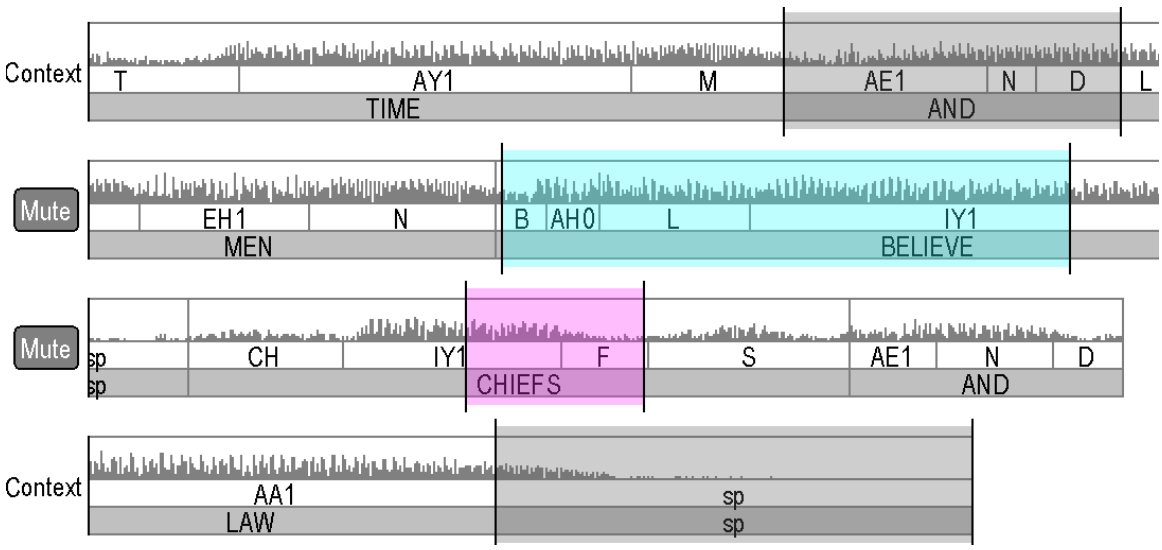


Figure 3.3: More advanced users can have fine-grain control over the boundaries of audio snippets selected by our algorithm.

of data and no manual annotation, we perform voice conversion with the following procedure:

1. **Corpus preparation** - We first align the target speaker's speech samples to the transcript using a *forced alignment* algorithm [74]. This method converts the transcript to phoneme sequences and then establishes the mapping between phonemes and speech samples in time. This step is also used by existing text-based speech editors such as that of Rubin et al. [69].

2. **TTS selection and preparation** - Given the *target* speaker's speech recording and transcript, we first choose an existing synthetic voice, the *source* TTS, which sounds similar to the target. This is done by sampling several sentences from the target speaker's narration, and synthesizing the same content in a variety of TTS voices. Then we calculate the acoustic distances between the target and each of the TTS voices using Mel-Cepstral Distortion (MCD) [44], a metric that measures spectral difference between two audio samples. We use the TTS voice with the lowest MCD distance to the target, as the source TTS voice.

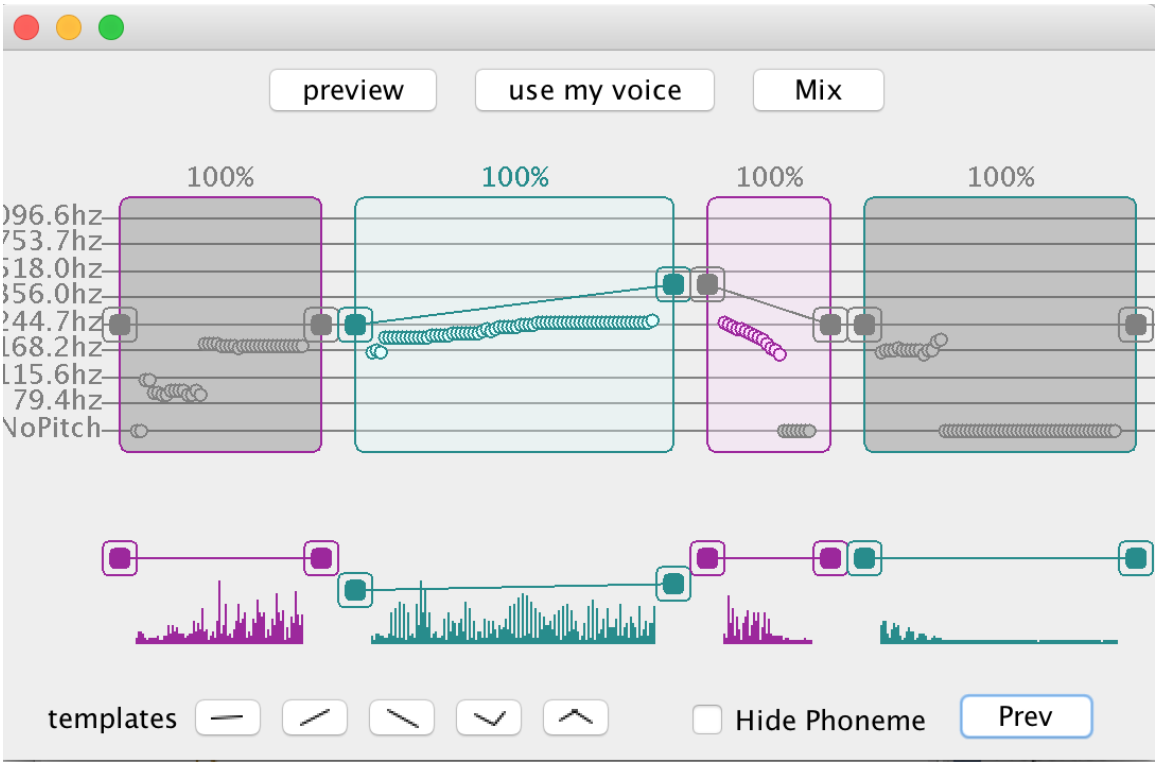


Figure 3.4: Advanced editors can manually adjust pitch profile, amplitude and snippet duration. Novice users can choose from a pre-defined set of pitch profiles (bottom), or record their own voice as an exemplar to control pitch and timing (top).

3. **Building a voice converter** - We build a voice converter based on parallel samples of the source and target voices. Since we have the target narration and transcript, we use the source TTS to synthesize the parallel speech in the source speaker's voice. We therefore do not need a pre-existing parallel speech corpus. Similar to the target voice, we segment the source voice into phonemes using *forced alignment*. The segmented audio and transcript (both source and target) serve as the training data for the proposed data-driven voice converter (Section 3.4.1 through 3.4.4).

4. **Synthesis and blending** - Given a new word, we synthesize the corresponding audio using the *source* TTS, transform the result using the trained voice converter to make it sound like the target speaker, and finally insert the synthesized word into the given context using cross fade.

3.4.1 CUTE voice conversion

In this section we describe CUTE, the voice conversion method recently introduced by Jin et al. [33]. As noted above, we use voice conversion to convert our source voice (produced by TTS) to match our target voice. CUTE is a data-driven voice conversion approach that uses the source voice as a query and searches for snippets of the target voice that as much as possible sound similar to the query, while also matching at the stitch boundaries. CUTE first segments both the source and target audio into small, equally-sized windows called *frames*. It then needs to find the optimal sequence of frames of the target voice in order to perform the synthesis. It does so via an optimization that minimizes two costs: matching cost between the selected target and the source frames and concatenation cost between adjacent frames. This optimization can be performed via dynamic programming, much as drawing gestures are stitched together in the HelpingHand approach of Lu et al. [48]. For interactive performance, CUTE relies on *triphone preselection,* which limits the search space to a small number of candidate frames, and is crucial in preserving synthesis quality. In particular, rather than using the KNN algorithm as in HelpingHand, CUTE selects candidates that are phonetically consistent with the query, meaning they share a similar sequence of triphones. 3.4.2–3.4.4 offer several quality and performance improvements over CUTE. Nevertheless, we briefly describe CUTE here as a baseline approach for voice conversion in our system:

1. **Frame-by-frame alignment**: First, we segment the audio into frames (in our implementation, the window size is 25ms and consecutive frames overlap by 5ms%). Next, we apply Dynamic Time Warping (DTW) to align the source frames with the target frames using a distance metric defined as the Euclidean distance between the corresponding MFCC features [54].

2. **Feature extraction and exemplars**: For both source and target voices, we extract the per-frame MFCC and F0 feature vectors. The F0 vectors representing the

fundamental frequency (also referred to as *pitch*) are normalized to the target voice's F0 range using a logarithmic Gaussian normalized transformation [88]. We concatenate the feature vectors of $2n + 1$ consecutive frames into exemplars [100]. These exemplars not only encode frequencies, but also temporal variation in frequencies.

Using the frame alignment from the first stage, we combine the source MFCC exemplars and target F0 exemplars into joint *matching exemplars*. Minimizing distance between the query and matching exemplars helps the synthesized voice sound like the query. Likewise, the target MFCC and F0 exemplars are combined into *concatenation exemplars*; minimizing the distance between neighboring concatenation exemplars encourages smoothness at stitch boundaries.

3. **Triphone preselection**: For performance, this step removes frames that are unlikely to be selected in the matching step. Based on phoneme segmentation, each frame is assigned a phoneme label. These labels are further converted to *triphones* – a phoneme and its two neighbors. For example, the triphones for the word "user" (phonemes: `Y UW1 Z ER0`) are

```
(st)_Y_UW1, Y_UW1_Z, UW1_Z_ER0, Z_ER0_(ed)
```

where `(st)` and `(ed)` label the beginning and the end of a sentence. For each query frame (source voice), frames (target voice) that share the same triphone label are selected as candidate matches. If there are no exact triphone matches, CUTE selects the frames in which either the first two or last two phonemes (called *diphones*) match with the query triphone. Likewise, if no diphones are available, candidates that only match the central phoneme (*monophone*) are selected. This way, frames that are most phonetically similar to the query are preserved for the next step.

5. **Matching**: In the speech synthesis literature, this matching step is called *unit selection* [87]. Based on the candidate table, the Viterbi algorithm [26] is used to select one candidate frame per query frame such that the sum of two cost functions are minimized. The first is *matching cost*, defined as the Euclidean

distance between the query exemplar and the candidate's matching exemplar. The second is *concatenation cost* designed to characterize the distortion introduced by concatenating two candidates. It is the product of three terms, *break cost*, *smoothness cost* and *significance cost*. The two cost functions together, aim to choose candidates that result in the least number of breaks and if there are breaks, they are in the least significant part of the synthesis and their transition is smooth.

6. **Concatenation**: Consider sequences of consecutive frames as audio snippets – we need to combine these snippets at their stitch points. We use the WSOLA algorithm of Roelands and Verhelst [68] to find the overlapping areas between adjacent snippets where the waveforms are most similar to one another, and blend the snippets together using a cross-fade.

We have implemented CUTE for our application, and as shown by experiments described in 3.5, the quality of the synthesized result is often satisfactory. However, CUTE has several drawbacks in our context. First, CUTE is slow whenever matching triphones are not found in the preselection stage. In that case CUTE falls back on diphones or monophones, of which there is a much larger candidate pool, making the candidate table very large. Second, the CUTE algorithm searches for snippets that match at internal stitch boundaries, but does not optimize for blending an inserted word into its context – sometimes leading to audible artifacts. Third, CUTE has difficulty with accents. Suppose the source and target speakers use different pronunciations (both of which are valid) for the same word in the same context. When synthesizing the same word, CUTE cannot emulate the target speaker's accent because of mismatched phonemes between the query and the candidates. Finally, CUTE sometimes fails to synthesize natural sounding prosody. This is especially the case when we have limited training data, as the selected segments might have pitch discontinuities at the stitch boundaries.

In the remainder of this section we introduce several improvements to CUTE that address these limitations: use of *Exchangeable Triphones* allows for alternative pronunciations. *Dynamic Triphone Preselection* further minimizes the search space while preserving meaningful candidates. *Range Selection* replaces the traditional Unit Selection described above with a faster method for selecting audio snippets directly. Finally we introduce an approach for generating *Alternative Synthesis Results* that all tend to sound reasonable but emphasize different characteristics of the target voice.

3.4.2 Exchangeable Triphones

Recall that when CUTE cannot find an exact triphone match, it falls back on diphones or monophones, which leads to a much larger candidate set containing worse matches, which impacts quality and performance. Here we introduce two types of *exchangeable triphones* that expand the triphone matching to include similar triphones that lead to only a slightly different pronunciation, while maintaining a much smaller search space than the diphone fallback.

Word-based Exchangeable Triphones come from a word-to-phoneme dictionary like the one used in the aforementioned forced alignment process [74]. Some of the triphones in a word are exchangeable because alternative pronunciations of the word exist. For example the word "this" has two pronunciations, `DH_IH1_S` and `DH_AH0_S`, in the dictionary. So when we search for one of those triphones in our corpus, we will accept the other as a valid match as well.

Accent Exchangeable Triphones are not defined in dictionaries, but rather are discovered via the phoneme alignment component of the DTW-based source to target alignment process outlined above in the first stage of the CUTE algorithm (Section 3.4.1). In that stage, we discover cases where the source tends to utter one triphone whereas the target utters a different triphone, when saying the *same* text. Figure 3.5 shows an example of such phoneme alignment. In this example (third

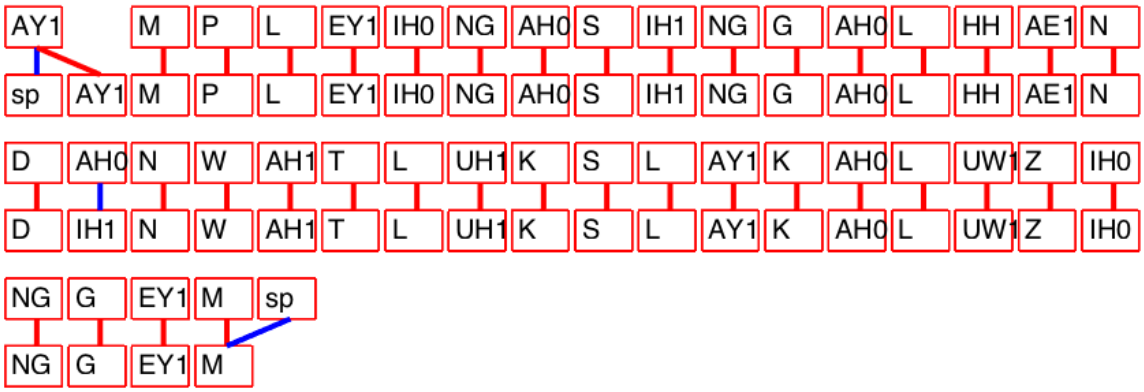


Figure 3.5: Alignment of two phoneme sequences (source above and target below). Red lines indicate a match whereas blue lines show mismatching phonemes (and thus suggest potential accent mappings).

and fourth rows), we find that `D_AH0_N` and `D_IH1_N` are exchangeable. Therefore, just as with word-based exchangeable triphones above, we allow matches between these triphones when searching the corpus. This allows for a target speaker with a particular accent to be better matched with the more generic source TTS voice.

3.4.3 Dynamic Triphone Preselection (DTP)

When a query triphone has no matches in the corpus, the CUTE algorithm instead fills the candidate table with diphones or monophones as described above. This is somewhat ameliorated by the exchangeable triphones introduced in the previous section. Nevertheless, the problem can still occur. Since the complexity of the Viterbi algorithm scales quadratically with the number of candidates, having a large number of inferior candidates (matching only the diphones and monophones parts) is undesirable. Moreover, not all diphones are necessary. When the matching step imposes a large break penalty for snippet boundaries, only segments with the smallest number of breaks tend to be selected. Therefore we can remove candidates that might result in larger number of breaks from the candidate table. Essentially what we do is perform a dynamic programming optimization at the phoneme level, to reduce the

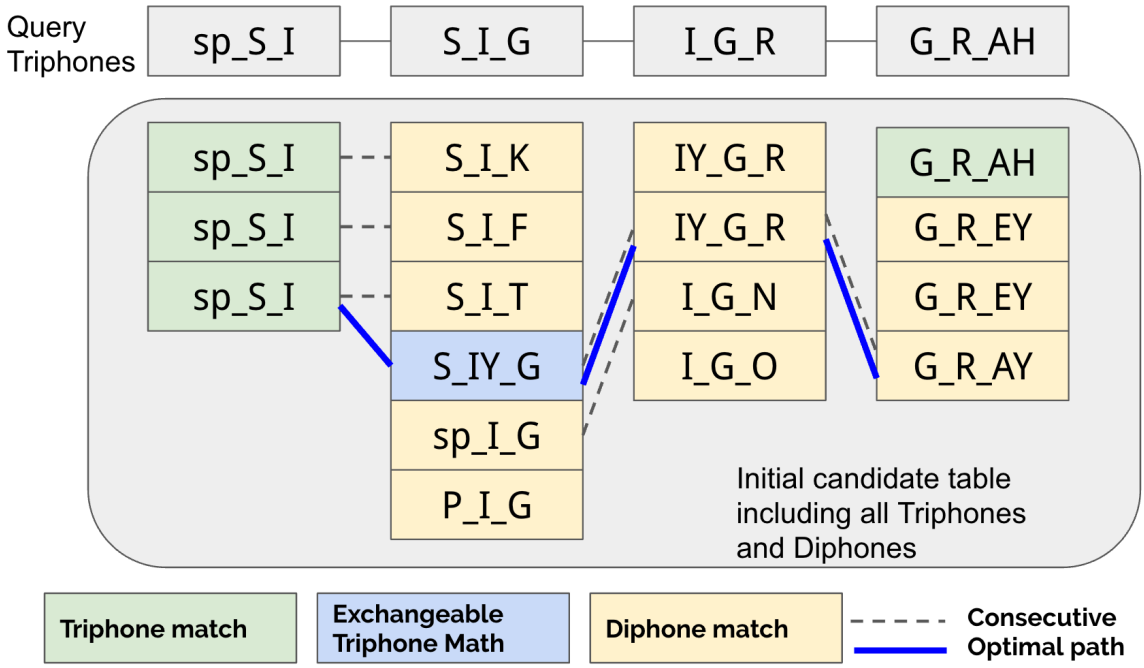


Figure 3.6: Dynamic triphone preselection. For each query triphone (top) we find a candidate set of good potential matches (columns below). Good paths through this set minimize differences from the query, number and severity of breaks, and contextual mismatches between neighboring triphones.

set of candidates to only those triphone sequences with the optimal number of breaks. This hugely accelerates the frame-level search that follows (Section 3.4.4).

To construct an optimally compact *frame candidate table*, we first perform Dynamic Triphone Preselection, or DTP, at the phoneme level to obtain a *triphone candidate table*. We call audio frames spanning a phoneme, a *phoneme segment*. One triphone can have multiple matching phoneme segments. The goal of DTP is to select a small set of segments per query triphone such that the sequences of frames that have the minimal number of breaks are contained. In other words, this method finds optimal paths through phoneme segments that contain minimal number of breaks and then merges them into the candidate table.

The method is illustrated in Figure 3.6. For each query triphone q, we first initialize its candidate table with all matching Triphones, with exchangeable triphones also included. Similar to the triphone preselection step, when exact matches are

unavailable, diphones and monophones are used instead. Between neighboring phonemes, some segments are consecutive (dotted links) in the original target speech and some are not. Non-consecutive phoneme segments will surely introduce a break in the matching step and thus should be minimized with priority. We also want to minimize the number of neighboring Triphone segments that do not match, e.g. `sp_S_I` and `P_I_G`, because they are likely to cause audible distortion when they are stitched together. Finally, we should also minimize the number of Diphones and Interchangeable Triphones because they are an approximation to the desired Triphone in the query. Putting all these criteria together, this problem is to minimize an energy function of matching cost (similarity between query Triphone and a candidate Triphone segment) and concatenation cost (whether there is a break and whether a segment matches its previous segment's Triphone). This is very similar to those in the matching step of CUTE and HelpingHand [48], and can be solved with the Viterbi Algorithm. Figure 3.6 shows an example optimal solution. Note that there could be multiple optimal paths, and all of them should be included. Finally, we retain only the triphone segments that are included in the optimal paths and omit the rest. The retained segments form the input to the *range selection* step that follows.

3.4.4 Range Selection

In the *matching* step of the CUTE algorithm, the preselected triphones are translated into corresponding frames and the optimal sequence of frames are selected using the unit selection. This section introduces a new method called *Range Selection*, to replace Unit Selection. Instead of selecting individual frames and indirectly encouraging continuity using a high concatenation cost, *Range Selection* selects ranges of consecutive frames directly by choosing their starting and ending boundaries so that the sequences sound similar to the query in terms of phonemes and pace and contain little distortion at stitching points (Figure 3.7).

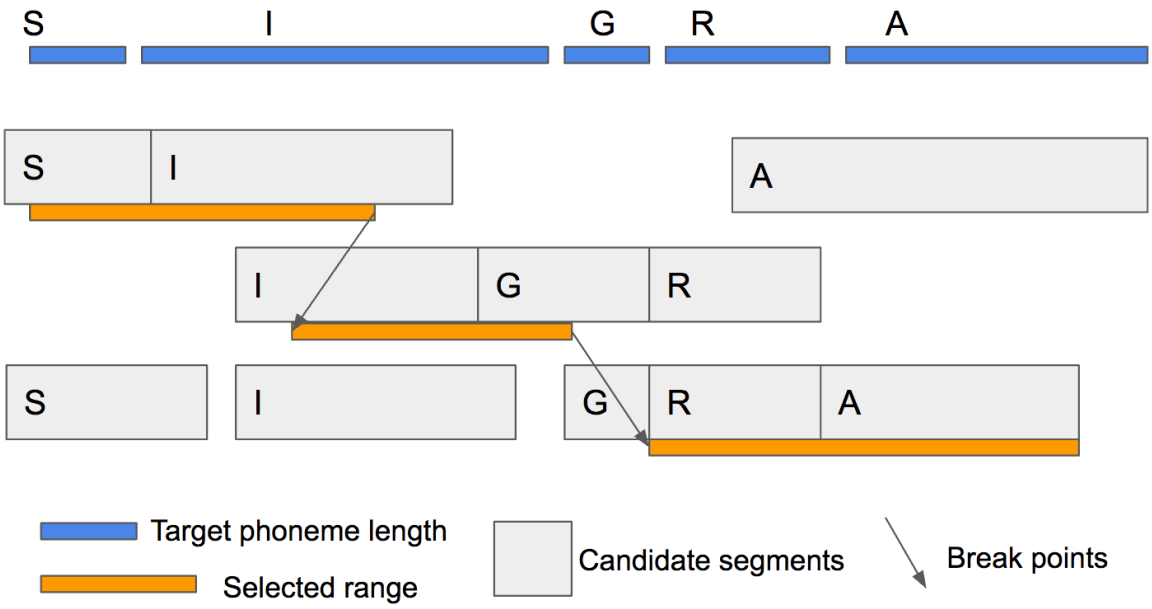


Figure 3.7: Range selection objective illustrated. The blue bars are the query phonemes with length proportional to their duration. The gray wider bar are candidate segments; consecutive segments are visually next to each other. The orange bar depicts one possible selection of the candidates while the arrows show where stitching occurs between the selected range of segments.

Selecting a range of frames instead of individual ones offers several advantages: (1) it allows the definition of similarity at the sequence level and thus more sophisticated similarity measures such as Mel-Cepstral Distortion can be used; (2) it is an order of magnitude faster than frame level unit selection – by a factor of average number of frames per phoneme; and (3) it includes duration of phonemes as part of the cost function, offering explicit controls on the pace of the resulting synthesis – it is superior to the skip and repeat costs in unit selection, which do not have a direct influence on the duration of the synthesized signal.

Based on pre-selected phoneme segments, *Range Selection* finds a small set of subsequences from those segments or "range", expressed by its starting and ending frame numbers. We use $\langle s, t \rangle$ to present a range from Frame s to Frame t. Since dynamic triphone preselection (Section 3.4.3) ensures that breaks will only occur once per phoneme, we can limit the number of ranges selected per phoneme to be at most two (e.g. Phoneme I in Figure 3.7). It means for each query phoneme, at most

two candidate segments are considered, and the ranges we select from them are their subsequences.

Since only one break is allowed per phoneme, the break will occur in one of the two locations: (Case 1) inside a phoneme, e.g., the break inside Phoneme I in Figure 3.7; and (Case 2) between two phonemes, e.g., the break between Phoneme G and R in Figure 3.7. In Case 1, the range selected must cover the beginning of phoneme (first candidate of Phoneme I) because otherwise it will introduce one more break, violating the rule. Then it will transition to either another candidate in the same phoneme (second candidate of Phoneme I) or to the next phoneme (phoneme G transitioning to R). Note that transitioning to another candidate of the same phoneme means a second range is selected and the second range must extend to the end of the phoneme (second candidate of Phoneme I); otherwise an extra break will occur. In Case 2, there should be only one range selected inside the phoneme because if there are two, one more break is introduced.

To optimize for similarity, smooth transition and pace, we define an objective function in the following way. Let R_{ij} be the j-th candidate segment for Phoneme i chosen by DTP. Each segment can be represented with two numbers, the beginning and ending frame indices, $R_{ij} = \langle b_{ij}, e_{ij} \rangle = \{b_{ij}, b_{ij}+1, ..., e_{ij}\}$. Let variable $\mathcal{R}_{ik}$ be the k-th *selected range* for phoneme i, where $\mathcal{R}_{ik} = \langle s_{ik}, t_{ik} \rangle$. Define the set of ranges selected for Phoneme i as $\mathcal{R}_i = \{\mathcal{R}_{ik} \quad | \quad k \in (1, K_i)\}$ where $K_i \in 1, 2$ is the number of ranges selected for Phoneme i. *Range Selection* minimizes the following function:

$$O_{rs} = \sum_{i=1}^{n} (\alpha S(q_i, \mathcal{R}_i) + \beta L(q_i, \mathcal{R}_i)) + \sum_{i=1}^{n} \mathcal{C}_i + \sum_{i=2}^{n} \mathcal{D}_i \tag{3.1}$$

where q_i is the i-th query phoneme; Functions S and L measure similarity cost and the duration cost between the query phoneme and the selected ranges $\mathcal{R}_i$. Their weights are controlled by a constant value *alpha* and *beta*. Functions $\mathcal{C}_i$ and $\mathcal{D}_i$ are two types

of concatenation costs that penalizes concatenating *ranges* that are dissimilar to one another at the boundaries [17]. $\mathcal{C}_i$ is used for a concatenation point in the middle of a segment (Case 1) and $\mathcal{D}_i$ for a concatenation point at the beginning (Case 2). Through our experiments, we found that balancing between concatenation, similarity and duration cost ($\alpha = 1$ and $\beta = 6$) is most likely to produce good results. The above optimization problem can be solved efficiently with dynamic programming, described in the next section.

3.4.5 Dynamic Programing for Range Selection

Range selection aims to find a small set of consecutive frames (*ranges*) that are similar to the query and then combine them together to obtain the final synthesis result. In this section, we present the algorithmic details and more precise mathematical definitions about *Range Selection*. To recap, a range , denoted as $\langle s, t \rangle$, is defined as a sequence of frames starting from index s to t. The j-th candidate segment of phoneme i is $R_{ij} = \langle b_{ij}, e_{ij} \rangle$. Our objective is to select ranges $\mathcal{R}_{ik} = \langle s_{ik}, t_{ik} \rangle$ that minimizes Equation 3.1. We denote the collection of ranges selected for i-th phoneme as $\mathcal{R}_i = \{\mathcal{R}_{ik} \quad | \quad k \in (1, K_i)\}$, assuming there are K_i ranges selected for that phoneme ($K_i \in \{1, 2\}$).

In Equation 3.1, the concatenation costs $\mathcal{C}_i$ and $\mathcal{D}_i$ are defined as follows:

$$\mathcal{C}_i = \begin{cases} 0 & \text{if } K_i = 1 \\ C(t_{i1}, s_{i2}) & \text{if } K_i = 2 \end{cases} \qquad \mathcal{D}_i = C(t_{i-1,K_{i-1}}, s_{i1})$$

where function $C(t, s)$ represents the distortion transitioning from Frame t to Frame s. In this section, we define $C(t, s)$ as the Euclidean distance between the exemplar feature of Frame t to that of Frame s. For other cost functions in Equation 3.1, we define similarity cost S as Mel-Cepstral Distortion and the duration cost $L(r_1, r_2)$ as the log of the ratio between the length of r_1 and the length of r_2.

Because we limit the number of *ranges* per phoneme to be at most two, there are two types for each phoneme (1) choose two ranges, one starts from phoneme boundary (we call it `Pre`) and the other ends at a phoneme boundary (we call it `Post`). (2) choose only 1 range, starting and ending in the same phoneme segment; we call its starting point `Start` and ending point `End`.

This allows us to use dynamic programming to solve the general optimization defined in equation 3.1 efficiently. Let n_i be the number of candidates for phoneme i. For a segment $\langle b_{ij}, e_{ij} \rangle_{j=1...n_i}$, the only valid *ranges* set for each query phoneme i should belong to one of the following two sets: $\texttt{Pre}(i) \times \texttt{Post}(i) \equiv \{\{r_{pre}, r_{post}\} | r_{pre} \in \texttt{Pre}(i), r_{post} \in \texttt{Post}(i)\}$ and $\texttt{Start}(i) \otimes \texttt{End}(i) \equiv \{< b, e > | j \in [n_i], b \in \texttt{Start}(i,j), e \in \texttt{End}(i,j), b < e\}$ where,

$$\begin{aligned}
\texttt{Pre}(i) &= \{\langle b_{ij}, b_{ij} + k \rangle | j \in [n_i], k \in [e_{ij} - b_{ij}]\} \\
\texttt{Post}(i) &= \{\langle b_{ij} + k, e_{ij} \rangle | j \in [n_i], k \in [e_{ij} - b_{ij}]\} \\
\texttt{Start}(i,j) &= \{b_{ij}, b_{ij} + 1, ..., e_{ij} - 1\} \\
\texttt{End}(i,j) &= \{b_{ij} + 1, b_{ij} + 2, ..., e_{ij}\} \\
\texttt{Start}(i) = \bigcup_{j=1}^{n_i} \texttt{Start}(i,j), &\quad \texttt{End}(i) = \bigcup_{j=1}^{n_i} \texttt{End}(i,j)
\end{aligned}$$

For example, if a segment contains frame 1 through 3, then the above four sets are: $\texttt{Pre} = \{\langle 1,2 \rangle, \langle 1,3 \rangle\}$, $\texttt{Post} = \{\langle 1,3 \rangle, \langle 2,3 \rangle\}$, $\texttt{Start} = \{1,2\}$ and $\texttt{End} = \{2,3\}$. And the valid *ranges* are: $\texttt{Pre} \times \texttt{Post} = \{\{\langle 1,2 \rangle, \langle 1,3 \rangle\}, \{\langle 1,2 \rangle, \langle 2,3 \rangle\}, \{\langle 1,3 \rangle, \langle 1,3 \rangle\}, \{\langle 1,3 \rangle, \langle 2,3 \rangle\}\}$ and

$\texttt{Start} \otimes \texttt{End} = \{\{\langle 1,2 \rangle\}, \{\langle 1,3 \rangle\}, \{\langle 2,3 \rangle\}\}$. Because we are looking at one phoneme here ($i = 1$), we omit i from the equation. We can assume these sets are ordered for simplicity.

Now the task becomes selecting a subset of these valid *ranges* to minimize the objective function. First we prepare a minimal cost table: for each phoneme i,

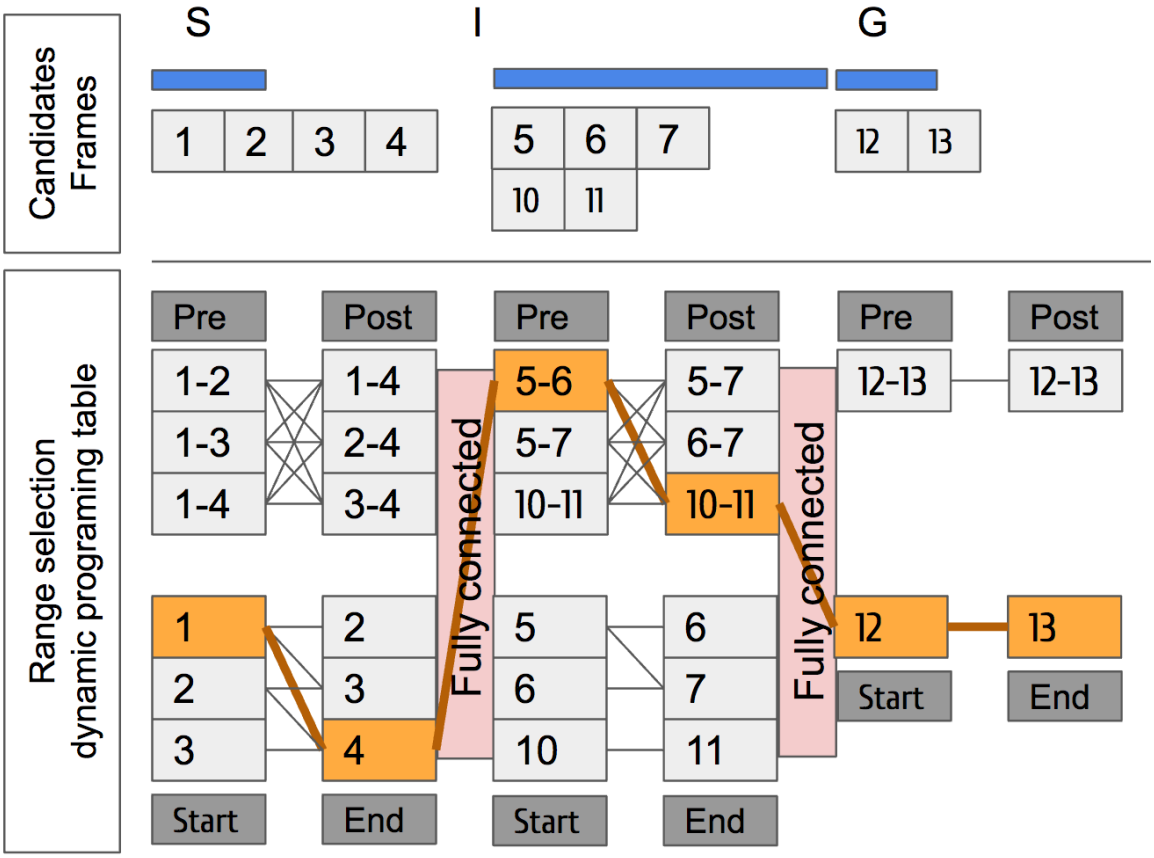


Figure 3.8: Range selection algorithm illustrated. Above: the frames selected by dynamic triphone preselection, numbered by frame indices. Below: the dynamic programming table, where each box shows the frame-range it represents; the connections between boxes show dynamic programming dependencies. The orange boxes and connections show an example solution, wherein frame ranges 1-4, 5-6, 10-11 and 12-13 are selected.

we create a note for each of the elements in the above 4 sets, $\texttt{Pre}(i)$, $\texttt{Post}(i)$, $\{\texttt{Start}(i,j)\}_j$ and $\{\texttt{End}(i,j)\}_j$. Let $F(\texttt{Pre}, i, j)$ be the corresponding frames of j-th element in set $\texttt{Pre}(i)$ and $M_{\texttt{pre},i,j}$ be its minimal cost. We do the same for `Post`, `Start` and `End`. Then we can derive the following dynamic programming algorithm that selects a transition path through the table that combines *ranges* to obtain a minimal cost M:

For `Start` and `Pre`, their preceding *ranges* are always from the previous phoneme. Therefore, the minimal cost M is defined to be the smallest of all their preceding *ranges*' minimal costs M plus concatenation cost C. If the *ranges* are consecutive,

the concatenation cost is 0. Here is the complete mathematical definition.

$$
\begin{aligned}
M_{\texttt{Pre},i,j} = \min(& \\
& \min_t \{M_{\texttt{Post},i-1,t} + C(F_{\texttt{Post},i-1,t}, F_{\texttt{Pre},i,j})\}, \\
& \min_t \{M_{\texttt{End},i-1,t} + C(F_{\texttt{End},i-1,t}, F_{\texttt{Pre},i,j})\}) \\
M_{\texttt{Start},i,j} = \min(& \\
& \min_t \{M_{\texttt{Post},i-1,t} + C(F_{\texttt{Post},i-1,t}, F_{\texttt{Start},i,j})\}, \\
& \min_t \{M_{\texttt{End},i-1,t} + C(F_{\texttt{post},i-1,t}, F_{\texttt{Start},i,j})\})
\end{aligned}
$$

For each `Post` *range*, when combined with a preceding `Pre` *range*, they form a valid range choice for the current phoneme. Then we can use the corresponding frames to determine the similarity cost and the duration cost. If `Pre` and `Post` are not consecutive, then there will be a concatenation cost. Therefore,

$$
\begin{aligned}
M_{\texttt{Post},i,j} = \min_t \{ & M_{\texttt{Pre},i,t} + \alpha S(q_i, \{F_{\texttt{Pre},i,t}, F_{\texttt{Post},i,j}\}) \\
& + \beta L(q_i, \{F_{\texttt{Pre},i,t}, F_{\texttt{Post},i,j}\}) \\
& + C(F_{\texttt{Pre},i,t}, F_{\texttt{Post},i,j})\}
\end{aligned}
$$

in which the definition of α, β, S and L follows the same definition used in Equation 1. Similarly, combining an `End` *range* with a `Start` *range* forms a valid *range* choice that defines the similarity and duration cost. Since there is only one *range* selected, there is no concatenation cost. Therefore,

$$
\begin{aligned}
M_{\texttt{End},i,j} = \min_t \{ & M_{\texttt{Start},i,t} + \alpha S(q_i, \{\langle F_{\texttt{Start},i,t}, F_{\texttt{End},i,j}\rangle\}) \\
& + \beta L(q_i, \{\langle F_{\texttt{Start},i,t}, F_{\texttt{End},i,j}\rangle\})\}
\end{aligned}
$$

Using back trace, we can extract the selected *ranges* that produce the minimal cost. Figure 3.8 shows an example selection result. When combined, the final selected frames form two consecutive: segments: 1-6 and 10-13.

Range Selection is more efficient than Unit selection. Suppose each query phoneme lasts m frames and has n candidates. Unit selection has a complexity of $O(mn^2)$ per phoneme with the Viterbi algorithm. In *Range selection*, however, the list of candidates per phoneme contain $O(n)$ rows and only 2 columns (Figure 3.8). Therefore, the complexity of *Range selection* per phoneme is $O(n^2)$. Because of the efficiency, we can mass produce selection results for all possible values of α and β, which leads to alternative Syntheses (Section 3.4.6).

3.4.6 Alternative Syntheses

While the default range selection method uses predefined α and β values in our experiments, alternative syntheses can be produced by using different combinations of α and β. Since α is the weight of similarity, the higher the value of α, the closer to the query the synthesis sounds will be, in both pitch and timbre. A higher α is more likely to lead to stitching distortion while a lower α is likely to lead to a smoother synthesis result since it is less restricted in terms of pitch and timbre. Similarly, higher β makes the duration of the synthesis closer to the query while lower β is less restrictive in terms of duration. Since range selection is effecient, we can quickly produce a grid of results, varying α and β in log space (Figure 3.9).

Though there are many of cells in the grid, only a handful sound unique. We group synthesis results using the same phoneme segments and color-code the grid by group. Inside each group, we retrieve a representative result by choosing the one with the minimal number of breaks. The most representative nodes in Figure 3.9 are marked with dots. These comprise the alternatives presented to the user, as shown in Figure 3.2.

3.5 Experiments and Results

This section describes some synthesis results, as well as several studies wherein we asked subjects to evaluate the quality of these results. The audio files used in these experiments can be found at our project web page: `http://gfx.cs.princeton.edu/pubs/Jin_2017_VTI/`

3.5.1 Synthesis Examples

In this section we show some example results from our speech synthesis algorithm to demonstrate how our proposed method achieves the goals of contextual smoothness

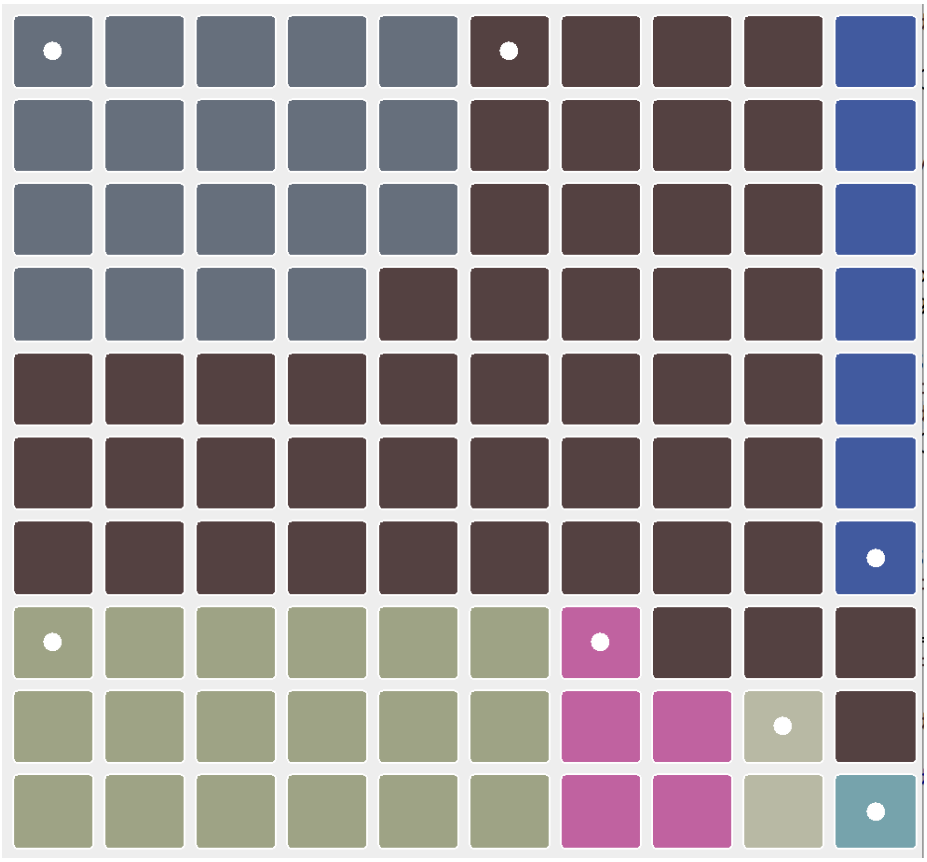


Figure 3.9: Alternative syntheses. A grid of results produced by varying values of α and β for range selection. Each cell represents one alternative, and they are color coded such that similar results have the same color.

and having as few breaks as possible. The CMU Arctic dataset [43] is used in these experiments.

Figure 3.10 shows a first example, where we synthesize the word "purchase" and insert into a recording where it is surrounded by silence. Multiple copies of the word "purchase" exist in the corpus, but because they are all chained with neighboring context words they would be unsuitable for simply copying into a silent context. Instead, our algorithm finds a different combination of audio pieces where `PUR` is from "pursue" (preceded by silence) and `ASE` is from the ending of "this" (followed by silence). Between those fragments, the middle part of "speeches" connects them. Note that because of exchangeable triphones, `Z` and `S` are allowed to be matched.

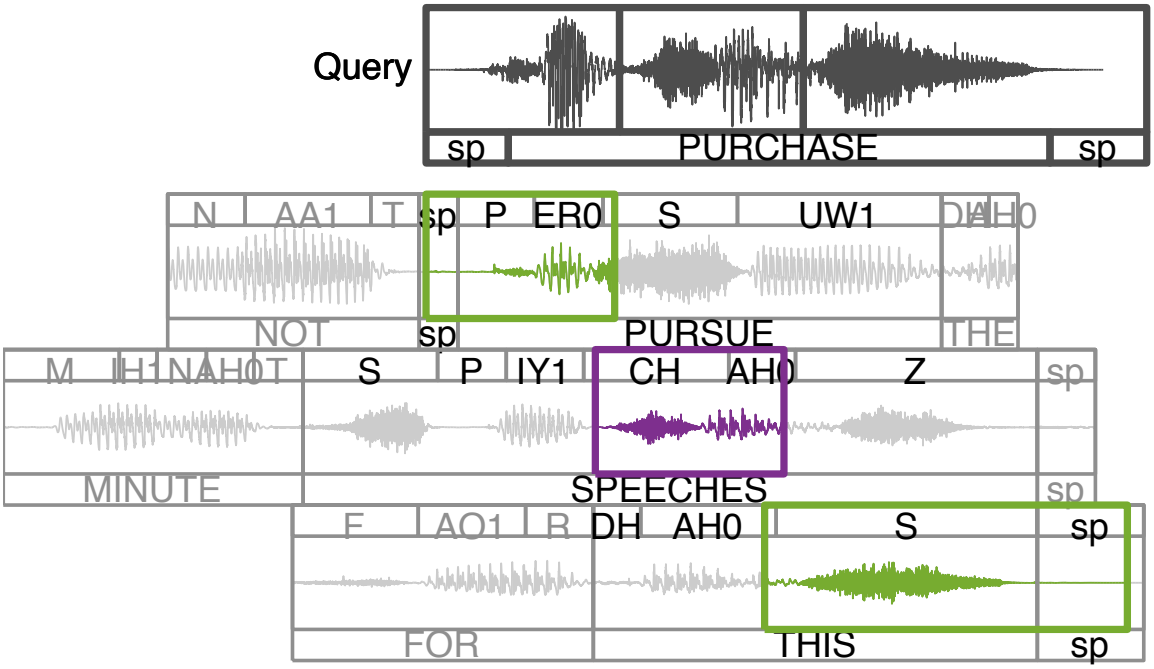


Figure 3.10: Synthesis example: the word "purchase" placed in a silent context (not audibly linked to neighboring words). The query is above, segmented by where breaks occur in the synthesis result. The text beneath it shows the spans of words. The following three chunks of waveform show the snippets of audio that are concatenated during synthesis (green, purple, green). For visualization, the query waveform is stretched or compressed to match the length of chunks below.

The next example is a homophone. The query is "prophet" without context. The result we obtained has no break at all. It is one solid piece of the word "profit". This demonstrates our approach not only finds exact word matches if they exist, it also finds homophones if they fit. The two lines below this paragraph show the query and the piece(s) used to synthesize that query. The first row is the query's phoneme decomposition. Then the row below it shows information about a piece, including sentence ID, frame numbers, phonemes and words contained in this piece.

```
Query: P R AA1 F AH0 T              (prophet)
u0518: (334-407) P|R|AA1|F|AH0|T    (profit)
```

The third example is alternative synthesis. When putting the word "poison" between "the" and "of the strange vegetation", we obtain alternative combinations of pieces that have different prosody and styles:

```
Query: P OY1 Z AH0 N                (poison)
COMBINATION 1:
u0277: (359-448) P|OY1|Z|AH0|N      (poisonous)
COMBINATION 2:
u0277: (359-402) P|OY1              (poisonous)
u0412: (107-120) Z                  (is)
u0519: (243-277) Z|AH0|N            (is in)
COMBINATION 3:
u0141: (309-354) P|OY1              (pointing)
u0020: (073-121) Z|AH0|N            (clubs and)
```

Although the second alternative combination has the largest number of breaks, it sounds most natural within the context. It is selected in the sampling method because it has competitive pitch continuity with other alternative syntheses. Also note that we select segments across words if there is no silence detected between them.

3.5.2 Mean Opinion Score Test

We conducted two experiments in Technical Turk to evaluate our methods. The first one is a Mean Opinion Score (MOS) test [51] that asks subjects to rate the quality of the inserted synthetic words. The second one is an identification test where subjects will tell whether they think a sentence has been edited or not. Four voices, two male and two female, from the CMU Arctic dataset, are used to create test samples. We used the first 800 utterances (40 minutes) for training and the remaining 332 utterances for word replacement. We randomly choose 44 words that spans 4 to 11 phonemes from 44 different sentences. For each sentence, we remove the chosen word, synthesize the word and insert it in the same location to create a recording with one word altered. We synthesize the same word using various other methods for comparison. There are 6 conditions where sentences are made:

- **Synth.** We use the source TTS voice to synthesize the word and put into context.
- **CUTE.** Based on our word synthesis framework, we use CUTE instead of VoCo for the voice conversion.
- **Auto.** Our method using pre-defined α and β values in range selection.
- **Choose.** We manually choose one synthesis from a number of alternatives (up to 16), if it improves on Auto above.
- **Edit.** We use the editing interface to further refine the synthesis, if it improves on Auto/Choose.
- **Real.** the actual recording without modification.

In each MOS test, we present a subject with a recording drawn randomly from 44 sentences and 6 conditions above. They are informed that each sentence has one word

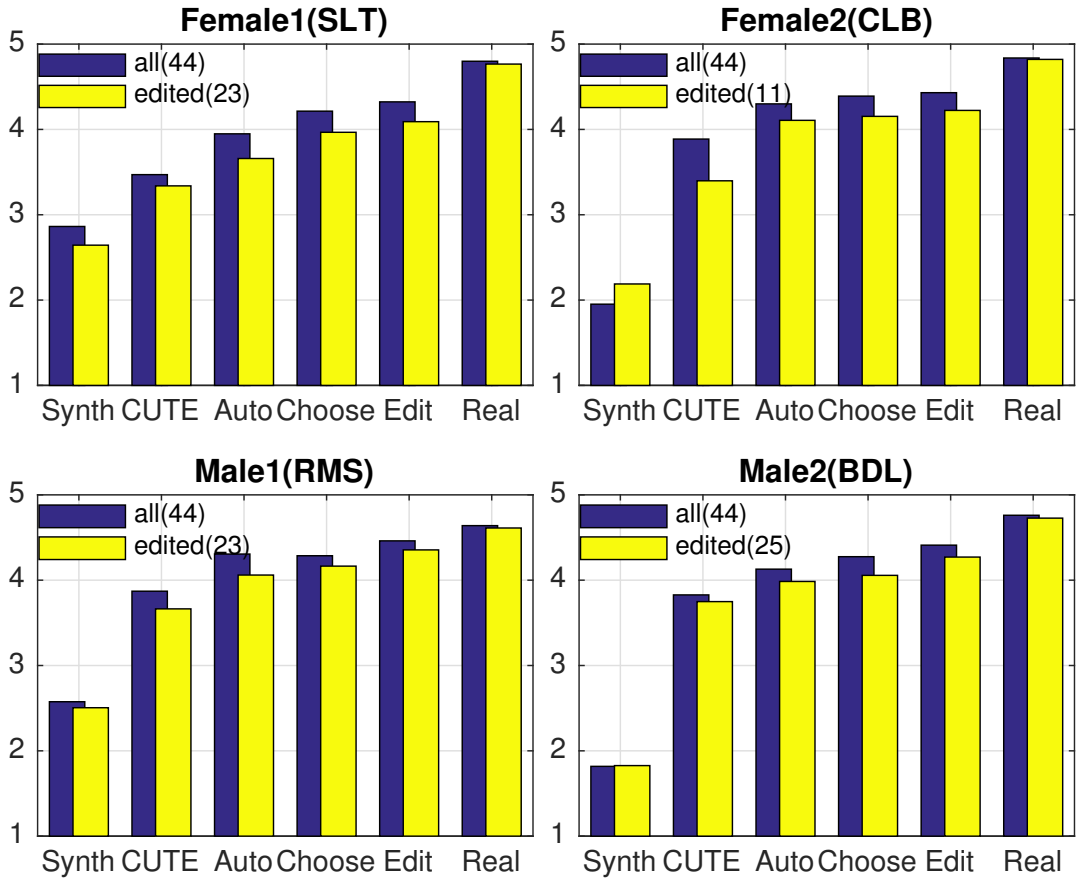


Figure 3.11: These mean opinion score tests show that VoCo synthesis results are generally perceived as higher quality than those of baseline methods, scored on a Likert scale from 1=*bad* to 5=*excellent*.

corrupted but restored by a computer program. They are asked to listen and rate the quality of the restored sentence on a Likert scale: 1 = bad (very annoying), 2 = poor (annoying), 3 = fair (slightly annoying), 4 = good (perceptible but not annoying) and 5 = excellent (imperceptible, almost real). They can play the recording multiple times.

On Amazon Mechanical Turk, we launched 400 HITs, 100 per voice. At the beginning of each HIT, we use a warmup question to get the subject familiar to the voice they are about to hear, where they are asked to choose a word heard in a long sentence. In each HIT, a subject is presented with 32 different sentences in which 24 of them are made of 4 instances from the above 6 conditions. From a held-out set

of sentences, we add 4 more instances of the "Real" condition and 4 more cases of badly edited "Synth" condition to validate that the subject is paying attention and not guessing randomly. For the data to be retained, the subject may make at most one mistake on these validation tests, by either rating < 3 on "Real" examples or > 3 on "Synth" examples. The sentences and conditions are globally load-balanced using a database, so that we cover every sentence-condition combination a similar number of times.

In the end, we collected 365 valid HITs (out of 400 issued). For each condition per voice, we average all the ratings to produce the bar plot shown in Figure 4.8. The blue bars show average rating of all sentences while the yellow bars only include sentences that were manually edited (because it improved on the automatic method). From the blue bars, we can see that the MOS is ascending in the order of Synth, CUTE, Auto, Choose, Edit and Real. The average scores for Auto, Choose and Edit are both over "Good" and they significantly outperform CUTE that resides in the range of "fair". For the male voice "RMS" specifically, the rating is very close to "Real" example. From the results on Edited examples, we can see all methods generally have reduced rating because these examples are hard to synthesize and thus require editing. We also notice that "Edit" consistently outperform "Choose", which indicates that manual editing is useful in improving quality for some sentences.

3.5.3 Identification Test

In the Identification test, a subject is presented one sentence drawn from the six conditions described above (Synth, CUTE, Auto, Choose, Edit and Real) and informed that there may or may not be one word edited using a computer. The task is to identify if the recording is original or edited. In this experiment, we released 440 HITs on Mechanical Turk, 110 for each voice. These HITs incorporate the same warm-up question as in the MOS test, as well as the same procedure relying on held-

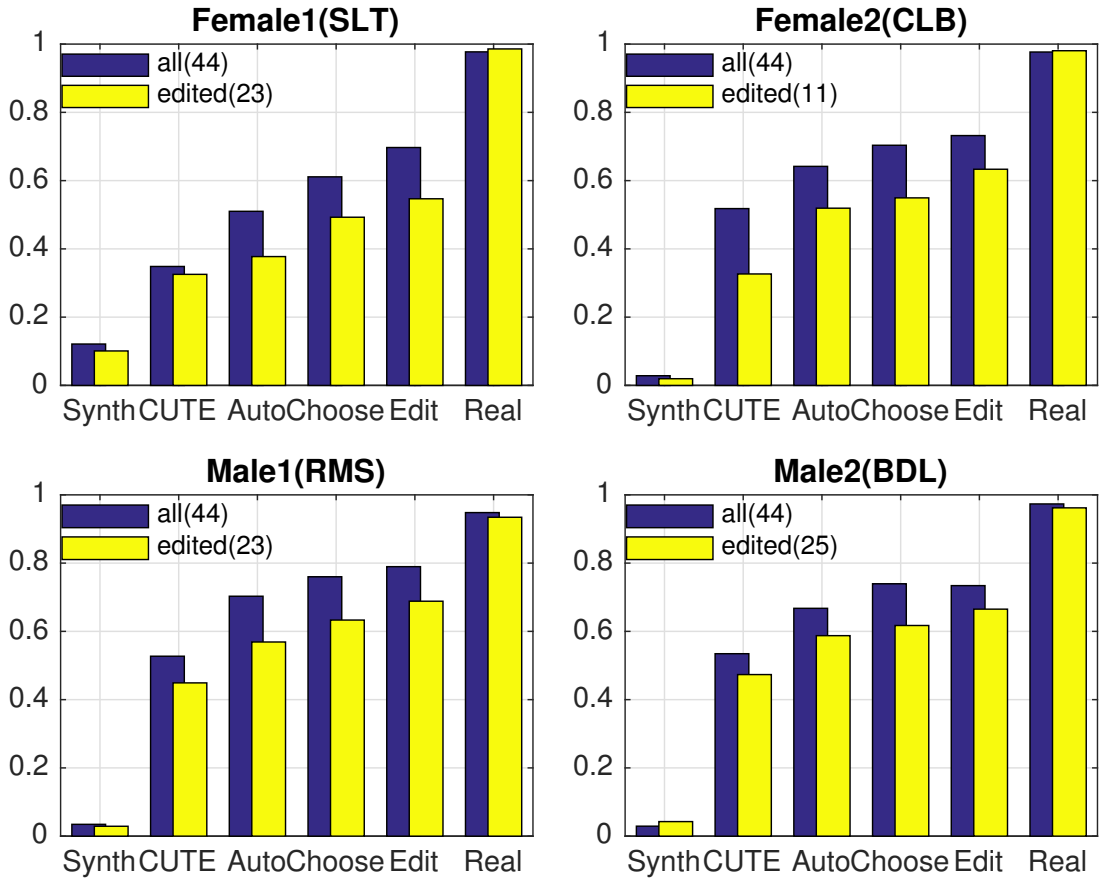


Figure 3.12: Identification tests show that VoCo synthesis results are more likely identified as original recordings than other baseline methods, shown here as a fraction of people who identify the recordings as original.

out sentences for validation with at most one mistake allowed. Each HIT contains 24 actual test sentences drawn randomly from the six conditions, together with 8 validation sentences. The sentences and conditions are load-balanced such that no sentence is repeated during a HIT, and globally all pairs of sentence and condition are covered roughly uniformly.

For this experiment we collected 391 valid HITs in total (out of 440 issued). For each method in each voice, we compute the percentage of subjects who think the recordings they heard are original. As with the MOS tests, we show both the results for all sentences as well as just the edited sentences in Figure 4.10. From the blue bars (all sentences), we observe that the likelihood of a synthesis being identified as original

ascends in the order: Synth, CUTE, Auto, Choose, Edit. The chance that Auto is confused with an actual recording is on average 57% for the female voices and 68% for the male voices which is significantly higher than those of CUTE (43% and 53%). Based on Fisher's exact test, Auto improves on CUTE with $p << 1 \times 10^{-16}$, indicating that our fully-automatic approach performs significantly better than a state of the art method for this task. Likewise Choose improves over Auto with $p < 2 \times 10^{-5}$, meaning that choosing among alternatives improves the results, at least in some cases. We also find that Edit improves on Choose ($p < 0.027$ using Fisher exact test), with significance even though manual editing is used only in difficult cases where choosing among alternatives does not yield adequate results.

For the Edit condition, the chance of being confused with actual recordings is 71% for female voices and 76% for male voices. This performance gap between female and male voices is reduced with alternative syntheses and manual editing. Note, however, that this gap is not observed at all in the MOS test, where our method performs equally well on female and male voices. We speculate that this difference is observed because males voices generally sound noisier than female voices, and the noise obscures discontinuities introduced by piecing together audio snippets. Evidence supporting this hypothesis is revealed in Figure 3.13. The probability of a synthesized word being identified as real is plotted as a function of the number of breaks points in synthesizing a word using the fully automatic method. For female voices there is a strong descending trend whereas the trend is less strong for male voices.

3.5.4 Human synthesis

VoCo makes it possible for non-experts to insert and replace words in audio narration by typing. But does it make editing easier and/or generate better results for audio experts? To answer these questions, we asked two audio experts to manually create new words and insert them into a sentence using software tools with which they were

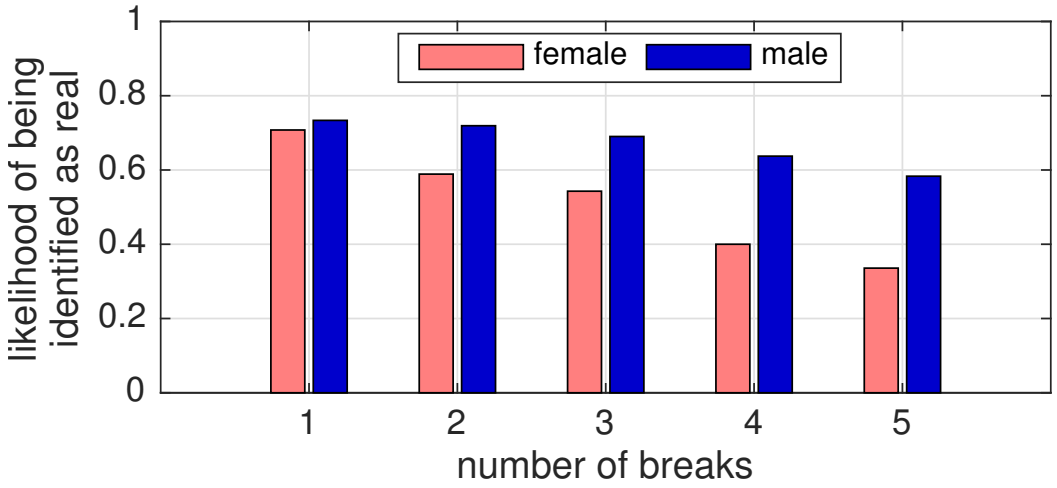


Figure 3.13: Identification test results as a function of number of breaks in the synthesis. For female voices, the result degrades significantly as the number of breaks increase while in male voices, the trend is less strong.

familiar. One expert chose to work with the open-source audio editor Audacity[1] while the other used commercial application Adobe Audition.[2] Four words – *mentioned, director, benefit* and *television* – were were picked for the task because they contain non-trivial numbers of phonemes and there were sufficient triphones to synthesize them in the corpus. The experts were presented with 80 seconds of recordings, selected so as to ensure that there were abundant material to create those specific words. We used only 80 seconds of recordings to narrow the search space and thereby save human search time, thus measuring an upper bound of human performance. The actual human performance on a real corpus that contains tens of minutes of recordings should be considerable worse than what is demonstrated in our experiment. For comparison, we also synthesized these words using VoCo and the Synth voice as a baseline, each based on the same corpus.

We clocked the time the experts spent to create each new word: Expert 1 spent 15-27 minutes and Expert 2 spent 30-36 minutes per word. For comparison, VoCo only takes about one minute to process the 80-second corpus, and less than a second

[1]http://www.audacityteam.org/
[2]http://www.adobe.com/products/audition.html

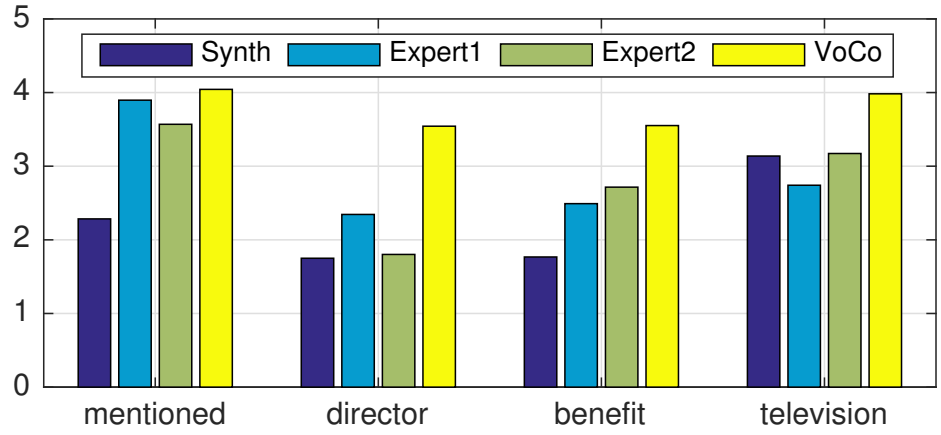


Figure 3.14: Mean opinion scores for experts and algorithms replacing words in sentences with synthesized words: *mentioned, director, benefit* and *television*. VoCo outperforms the experts, while Synth provides a baseline.

to synthesize each word. We also performed an MOS test on Mechanical Turk for the manually-edited words as well as two automatic methods – VoCo and Synth. There are 16 unique tests in total (4 words crossed with 4 conditions), and each unique test was shown twice per HIT (which contained 32 tests in total, presented in random order). Since the subject answered each question twice, we were able to reject data from subjects whose answers were inconsistent (more than 2 points apart on a question). Thus we collected 48 valid HITs, the results of which are shown in Figure 3.14. From the MOS scores, we can see that VoCo outperforms human editing in all four cases. Also the results of the generic synthesizer were considered no better than those of the experts, since they are in a voice that's slightly different from the voice used in the sentence. This study suggests that VoCo makes word insertion more efficient and effective than manual editing by audio experts.

3.6 Summary and Discussion of VoCo

This chapter describes an algorithm that supports text-based editing of audio narrations by augmenting the relatively straightforward cut-copy-paste operations

with the more challenging operations insert and replace. Our studies suggest that when synthesized words are inserted in the context of a spoken sentence, the modified sentence is often perceived as indistinguishable from other sentences spoken in the same voice. With alternative syntheses and manual editing, the "authenticity" of the synthesized word is further strengthened. Moreover, even in cases where subjects can recognize the sentence as edited, it would still have acceptable quality for many applications. In cases where a new word is desired (or needed), the only alternative approaches are to re-record the audio, or to recruit an expert audio engineer to painstakingly assemble the word (more than 20 minutes per word on average, according to our study, to yield words that were of inferior quality to those produced by VoCo). Therefore, we believe the proposed system offers a valuable improvement over the state of the art.

Nevertheless, there are several limitations in this approach that suggest areas for future work. First, our ultimate goal is to synthesize words with such consistently high quality that they will be rarely or never recognizable as edited. Our approach relies on MFCC and pitch features for determining compatible areas of waveform to be stitched together. We believe the factor that could improve the quality of our synthesized words would be to design or find features that were optimized specifically for our problem.

Second, our method relies on a hand-crafted energy function to select an "optimal" synthesis result, but this function is not well correlated with human perception. We believe it may be possible to design a more suitable energy function for our problem via a data-driven approach that may even be voice-specific.

Finally, our method can currently only synthesize a word or short phrase. This limitation mainly stems from the challenge of synthesizing a voice with natural prosody over longer sequences of words. Since the prosody of generic TTS systems is generally unnatural, our approach inherits this defect. One idea we would like to

investigate is the use of the audio editor's voice as the query, instead of the generic TTS voice. The editor could enunciate the query with a particular prosody, and this would guide the synthesized result. The main challenge is that in this scenario there is no aligned synthetic voice to guide the voice conversion process towards the target voice. However it may be possible by applying many-to-one voice conversion to the editor's voice [89]. An alternate approach might build on the recent WaveNet work of van den Oord et al. [57] which shows that deep learning can effectively model human prosody by learning directly from audio samples.

Chapter 4

Deep-VoCo: word synthesis based on deep neural networks

Like every other data-driven speech synthesis technique such as the MOS test described in Section 3.5.2., the quality of VoCo depends largely on the quality of data and data processing. Experiments shown in Section 3.5 displays 60% success rate on the CMU ARCTIC dataset; the synthesis quality deteriorite as the number of snippets goes up. Therefore the proposed VoCo method does not scale to long sentence synthesis where more breaks between snippets are required. There are multiple cause to this imperfection: first, certain triphones or phonemes may be scarce in the training data; the synthesis quality may drop when no matching triphones are available. Second, for each triphone, there may be limited variation of pitch contours and duration, resulting in overly long or short synthesis and inconsistent pitch at the concatenation boundary. Thirdly, VoCo is based on parallel voice conversion that heavily relies on good alignment between the source (TTS voice) and the target (human voice that we want to synthesize). Alignment error in frame alignment and forced alignment described in Section 4.1.2 will impact synthesis quality as the incompatible snippets may be selected as a result. Finally, VoCo uses an acoustic

distance metric that is far from being consistent to human hearing. Although pre-selection methods are proposed to overcome this problem, Range selection is still based on similarity and concatenation costs that are defined using these acoustic features. Therefore, this chapter explores an alternative direction to speech synthesis with the help of neural networks.

As stated in Section 1.2.2, parametric speech synthesis suffers from oversmoothing and muffled artifact; this is caused over-simplification of statistical models and the vocoder, a processor that synthesizes waveform from acoustic features such as MFCC and F0. Recently, a neural network based synthesizer called WaveNet is introduced by DeepMind [57]; unlike traditional speech synthesis based on hand crafted feature and vocoding process, WaveNet generates waveform directly from linguistic features such as phoneme sequence and F0. Recent work demonstrates that WaveNet can be used as a vocoder and the resulting synthesis quality is highly preferred over other existing numeric vocoders such as STRAIGHT and WORLD. WaveNet opens a gate to high fidelity speech synthesis and experiments show that the synthesis result is very close to real human speech making them hard to tell from real recordings [73]. However, the proposed method only works for Google's own voice samples; and it requires large amount of data supply (26 hours) and computation, making it unsuitable for VoCo application. As the third part of my thesis work, I aim to explore alternative neural network structures that synthesize high-fidelity speech with small amount of voice samples (less than 1 hour) and acceptable computational complexity comparable to that of VoCo. The proposed method has two parts: a neural vocoder that turns acoustic features into waveform in real time with the same quality as WaveNet; and a voice conversion network that transform a TTS voice to the target voice based on less than 1 hour of training data. The former is called FFTNet, introduced in Section 4.1 and the latter is called TimbreNet introduced in 4.2. Section 4.3 evaluates the

vocoder first and then evaluates both network jointly with the word replacement task conducted in Section 2.4.

4.1 FFTNet: a real-time speaker-dependent neural vocoder

We introduce FFTNet, a deep learning approach synthesizing audio waveforms. Our approach builds on the recent WaveNet project, which showed that it was possible to synthesize a natural sounding audio waveform directly from a deep convolutional neural network. FFTNet offers two improvements over WaveNet. First it is substantially faster, allowing for real-time synthesis of audio waveforms. Second, when used as a vocoder, the resulting speech sounds more natural, as measured via a "mean opinion score" test.

4.1.1 Introduction

The WaveNet project introduced a deep learning architecture capable of synthesizing realistic sounding human speech based on linguistic features and F0 pitch [57]. Though it was surprising to many researchers that a plausible waveform could be synthesized directly as the output of a convolutional neural network, it has subsequently been confirmed by many follow-on projects. The approach has many applications, including the classical text-to-speech (TTS) problem [21]. While WaveNet and others initially addressed TTS starting from *linguistic features,* ensuing work showed that speech could be synthesized directly from input *text* [97, 75]. The approach has also been adapted to other problems, including voice conversion [66, 42], speech enhancement [67], and musical instrument synthesis [57, 24].

Despite the impressive quality of the synthesized waveform, the WaveNet approach still suffers from several drawbacks: it requires substantial training corpus (roughly

30 hours), the synthesis process is slow (40 minutes to produce a second of audio), and the result contains audible noise. Recent work by Tamamori et al. [85] showed that WaveNet could also be used as a vocoder [20], which generates a waveform from *acoustic features*. Working from acoustic features, the training process is effective with a substantially smaller corpus (roughly one hour) [9] while still producing higher quality speech than baseline vocoders like MLSA [32]. Several research efforts have addressed the problem of computational cost. Paine et al. [58] introduce an algorithmic improvement for the same architecture called Fast WaveNet, which can synthesize a second of audio in a roughly a minute. The Deep Voice approach of Arik et al. [5] is able to achieve real-time synthesis by reducing the WaveNet model size significantly, but at the expense of noticeably reduced voice quality. Concurrent with our work, the WaveNet team also achieved huge performance gains by introducing a new deep learning network architecture that leverages parallelism on a GPU cluster [95].

This chapter introduces FFTNet, an alternative deep learning architecture, coupled with several improved techniques for training and synthesis. WaveNet downsamples audio via dilated convolution in a process that resembles wavelet analysis. In contrast the FFTNet architecture resembles the classical Fast Fourier Transform (FFT) [18], and uses substantially fewer parameters than the analogous WaveNet model. FFTNet models produce audio more quickly ($> 70\times$ faster) than the Fast WaveNet formulation [58], thereby enabling real-time synthesis on a single CPU. Moreover, we show that when used as a vocoder, FFTNet produces higher quality synthetic voices, as measured by a MOS test described in Section 3.5.2. We also show that the FFTNet training and synthesis techniques can improve the original WaveNet approach such that the quality of the synthesized voice is on par with that of the FFTNet architecture (albeit much slower to synthesize). Finally, we note that

the FFTNet architecture could also be leveraged in a variety of other deep learning problems such as classification tasks and autoencoders.

4.1.2 Method

Similar to WaveNet, our method generates waveforms one sample at a time based on previously generated samples and an auxiliary condition, but has a simpler architecture. In this section, we first briefly introduce WaveNet, then the proposed architecture called FFTNet, and finally a set of training and synthesis techniques essential in building a high quality FFTNet vocoder.

WaveNet vocoder

WaveNet [57] is a neural network architecture that has been used in audio synthesis to predict one audio sample at a time based on previously generated samples and auxiliary conditions, such as a sequence of phonemes and fundamental frequencies ($F0$). The prediction is based on the posterior distribution of sample values quantized using μ-law. When the auxiliary conditions are acoustic features, such as Mel Cepstral Coefficients (MCC) and pitch, WaveNet can be used as a Vocoder, which generates a waveform from these features. It makes no assumption on how speech is generated and has been shown to generate significantly more natural and clear speech than conventional vocoders.

The basic building block of WaveNet is dilated causal convolution [94]: given signal f, dilation d and kernel k, dilated convolution at time step t is defined as $(k *_d f)_t = \sum_i k_i f_{t-di}$. If we replace the multiplication between k_i and f_{t-di} with 1×1 convolution (`conv1x1`), we get a layer of 1-D dilated convolutional neural network, or DCNN. WaveNet is constructed by stacking a series of DCNN layers with exponentially increasing dilation factors for each subsequent layer: $2^0, 2^1, 2^2, ...2^n$. At

each layer, a gated activation structure is used:

$$z = \tanh(W_f *_d x + V_f *_d h) \odot \sigma(W_g *_d x + V_g *_d h)$$

where x is the output from the previous layer and serves as the input for the current layer; W_f and W_g are convolution kernels for the filters and the gates; V_f and V_g represent `conv1x1` on the auxiliary conditions h (F0 and MCC); and $\odot$ is element-wise dot product. The final output of a layer is obtained by another `conv1x1` on z. The original WaveNet proposed to use skip connections – an additional `conv1x1` is applied to z to obtain a new output s; then s of all layers are summed together to become a skip output. To further increase nonlinearity, more `conv1x1` and ReLU layers are applied on top of the skip output and finally a softmax layer is used to produce the posterior distribution of quantized sample values.

With dilated convolution, a n-layer network has a receptive field of 2^n meaning as many as 2^n previous samples can influence the synthesis of the current sample, which leads to superior synthesis quality. However, since WaveNet synthesizes one sample at a time, to generate one second of audio sampled at 16kHz, the causal dilated network needs to be applied 16,000 times. Faster methods have been proposed [58], which can produce 200 samples per second, but the performance is still far from real-time on personal computers. We aim to design a vocoder with a simpler (thus faster) yet equally powerful architecture.

FFTNet architecture

Highlighting the nodes that influence the prediction of the new sample, we see a reversed binary tree structure as shown in Figure 1. This dilated convolution structure resembles wavelet analysis - in each step filtering is followed by down-sampling. This observation inspired us to explore an alternative structure to wavelet based

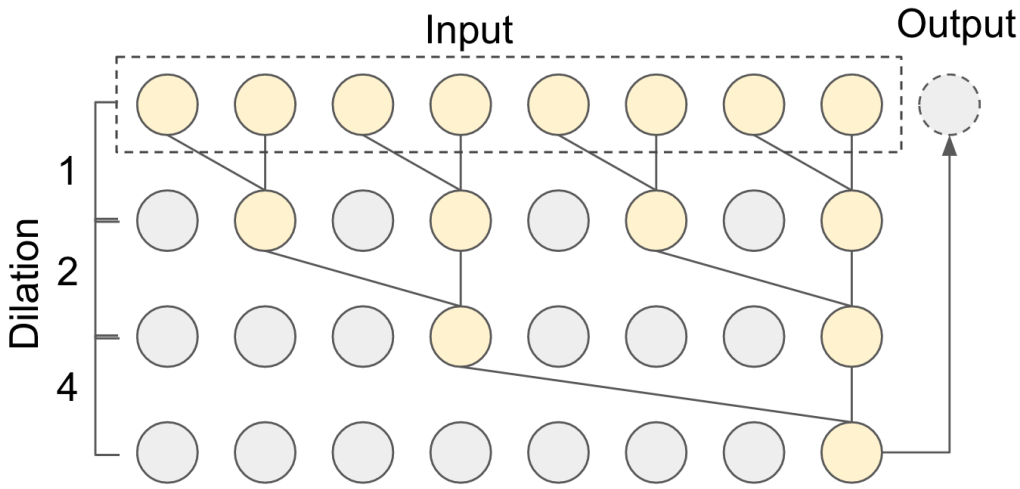


Figure 4.1: Dilated convolution in WaveNet

on the Cooley-Tukey Fast Fourier Transform (FFT) [18]. Given an input sequence $x_1, x_2, ..., x_n$, FFT computes the k-th frequency component f_k from time-domain series $x_0, ..., x_{N-1}$ by

$$f_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i n k/N} = \sum_{n=0}^{N/2-1} x_{2n} e^{-2\pi i (2n) k/N} + \sum_{n=0}^{N/2-1} x_{2n+1} e^{-2\pi i (2n+1) k/N}$$

Denote $\sum_{n=0}^{N-1} x_n e^{-2\pi i n k/N}$ as $f(n, N)$ and the above equation can be simplified as

$$\begin{aligned} f(n, N) &= f(2n, N/2) + f(2n+1, N/2) \\ &= f(4n, N/4) + f(4n+1, N/4) \\ &+ f(4n+2, N/4) + f(4n+3, N/4) = ... \end{aligned}$$

One can think of x_n as a node with K channels (e.g. 256 quantization channels) and the term $e^{-2\pi i (2n) k/N}$ as a transformation function, then each term $f(n, N) = f(2n, N/2) + f(2n+1, N/2)$ is analogous to applying a transformation to previous nodes x_{2n} and x_{2n+1} and summing up their results. If we use `conv1x1` as the

transformation, we can create a network that resembles the FFT as shown in Figure 2. In other words, given input $x_{0:N}$ defined as a 1D series $(x_0, x_1, ..., x_{N-1})$, each FFTNet layer clips the input into two halves ($x_L = x_{0:N/2}$ and $x_R = x_{N/2:N}$), performs separate `conv1x1` transformation to each half and then sums up the results:

$$z = W_L * x_L + W_R * x_R \tag{4.1}$$

where W_L and W_R are `conv1x1` weights for x_L and x_R. Instead of using a gated activation structure, FFTNet uses a simple ReLU followed by a `conv1x1` to produce inputs for the next layer, namely $x = \texttt{ReLU}(\texttt{conv1x1}(\texttt{ReLU}(z)))$, which reduces computation cost. Stacking n layers will give an input size of 2^n. The auxiliary conditions are transformed by `conv1x1` and then added to z. i.e.,

$$z = (W_L * x_L + W_R * x_R) + (V_L * h_L + V_R * h_R) \tag{4.2}$$

where h_L and h_R are the two halves of the condition vector h and V_L and V_R are `conv1x1` weights. Note that if the condition information is stationary along the time axis, one may use $V * h_N$ instead of $(V_L * h_L + V_R * h_R)$.

To use FFTNet as a vocoder, we define h_t as F0 and MCC features at time t. To generate the current sample x_t, we use the previously generated samples $x_{t-N:t}$ and auxiliary condition $h_{t-N+1:t+1}$ (shifted forward by 1) as the network input. In our experiments, the auxiliary condition is obtained as follows: first we take an analysis window of size 400 every 160 samples. Then we extract the MCC and F0 features for each overlapping window. For the h_t corresponding to the window centers, we assign the computed MCC and F0 values (26 dimensions in total). For the h_t that are not located at the window centers, we linearly interpolate their values based on the assigned h_t in the last step.

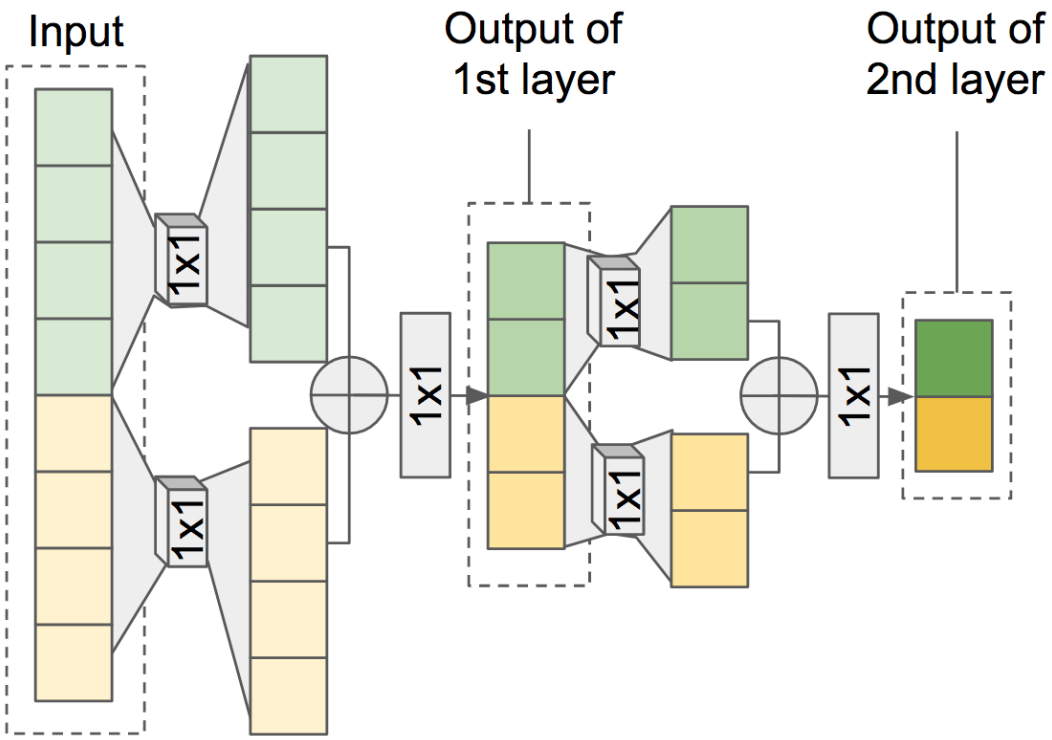


Figure 4.2: FFTNet architecture: given size-8 inputs, they are first divided into two halves; each passed through a different 1×1 convolution layer and then summed together. The summed size-4 output will pass through ReLU and then another 1×1 convolution and ReLU before repeating the the same operation.

Finally, FFTNet uses a fully connected layer followed by a softmax layer (size 1 with $K = 256$ channels) as the last two layers to produce the posterior distribution of the new sample's quantized values. To determine the final value of the current sample, one can use $argmax$ or perform random sampling from the posterior distribution, but we propose an alternative strategy in Section 2.3.2.

4.1.3 Training Techniques

In this section, we introduce a set of techniques that are essential for training an FFTNet to produce results of similar quality to WaveNet. Our experiments indicate that these techniques can also improve WaveNet synthesis quality.

Zero padding

WaveNet uses zero-padding during dilated convolution. The same idea can be applied to FFTNet. Given a sequence of length M, the input $x_{1:M}$ ($M > N$) is shifted to the right by N samples with zero padding. The N padded zeros are denoted as $x_{-N:0}$ where $\forall j < 0, x_j = \mathbf{0}$. Equation 1 becomes

$$z_{0:M} = W_L * x_{-N:M-N} + W_R * x_{0:M}$$

Our experiments show that without zero padding, the network tends to produces noise or gets stuck (outputting zeros) when the inputs are nearly silent. Zero-padding during training allows the network to generalize to partial input. We recommend using training sequences of length between $2N$ and $3N$ so that a significant number (33% - 50%) of training samples are partial sequences. In this work, we apply zero-padding to both WaveNet and FFTNet in our experiment.

Conditional sampling

Both WaveNet and FFTNet are classification networks - the last softmax layer produces a posterior distribution over 256 categories. Like every classifier, the prediction error comes from two sources: training error and true error. True error mainly corresponds to noise, and resides in the unvoiced parts of the signal. To synthesize noise, we rely on the network to learn the noise's distribution by the output posterior distribution on which we use random sampling to obtain the sample's value. Training error comes from the model itself; and the prediction strategy that gives the minimal training error is $argmax$. However, $argmax$ is not suitable for simulating signals that contain true noise, since it always chooses the center of a noise distribution leading to zero noise in the synthesis output. Instead of using $argmax$ universally, we propose to use different prediction strategies for unvoiced and voiced

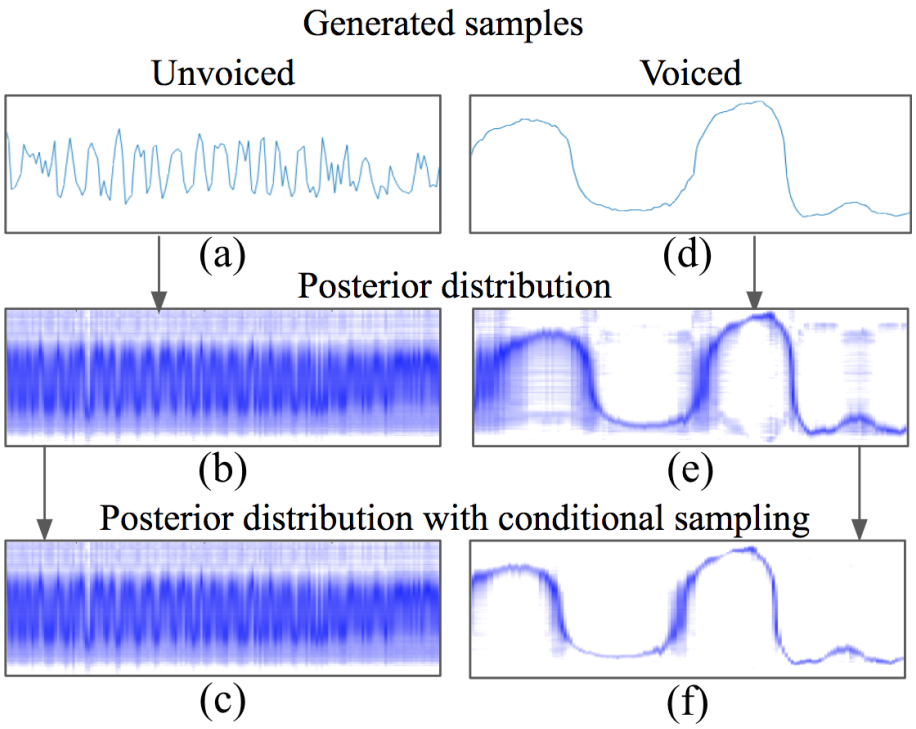


Figure 4.3: Conditional sampling: WaveNet and FFTNet model noise by matching the posterior distribution to the noise's distribution (b). Therefore to generate noise (a), we randomly sample from the posterior distribution (b). For periodic signals (d), we double the log of the posterior distribution (e), to obtain a cleaner distribution (f); for aperiodic signals, the posterior distribution remains the same (c).

sounds (Figure 3). For unvoiced sounds, we randomly sample from the posterior distribution; and for voiced sounds, we take the normalized logits (the input values before softmax), multiply it by a constant $c > 1$ and pass it through the softmax layer to obtain a posterior distribution where random sampling is performed. In this way, the posterior distribution will look steeper while the original noise distribution is preserved. In this work, we use $c = 2$.

Injected noise

Due to the training error, the synthesized samples always contain some amount of noise; during synthesis, the network will generate samples that get noisier over time. They serve as network input to generate the next sample, adding more and more randomness to the network. When the noise builds up, the output sample might drift

leading to clicking artifacts. To avoid such drift, the network needs to be robust to noisy input samples. We achieve this by injecting random noise to the input $x_{0:M}$ during training. We determine the amount of noise to inject into the input based on the amount of noise the network is likely to produce. In this work, we observe that the prediction is often one category (out of 256) higher or lower than the ground-truth category and thus, we inject Gaussian noise centered at 0 with a standard deviation of 1/256 (based on 8 bit quantization).

Post-synthesis denoising

Our experiments show that the injected noise eliminates the clicking artifact almost perfectly for FFTNet but introduces a small amount of random noise to voiced samples. Therefore, we apply a spectral subtraction noise reduction [46] to reduce the injected noise for the voiced samples. The reduction is proportional to the amount of noise injected during training. It is possible to apply noise reduction to the unvoiced samples as well, but it may result in artifacts. Therefore, we reduce the amount of noise reduction applied to the unvoiced samples by half.

4.2 TimbreNet: voice conversion based on RNN and bottlenecked FFTNet

This section introduces TimbreNet, a voice conversion method based on recurrent neural networks. Given acoustic features of a source voice (in this thesis, an off-the-shelf speech synthesizer), TimbreNet generates the acoustic features of a target voice (real human voice that we want to synthesize) according to pre-defined phoneme duration. The generated acoustic feature can be further combined with F0 to synthesize waveform using FFTNet proposed in the last section. The pre-defined

phoneme duration and F0 can be learned from data [96], extracted from source voice using normalization [88], or defined by user.

TimbreNet consists of an encoder net that encode acoustic features of source voice into a sequence of hidden states, and a decoder net that maps these hidden states to target acoustic features. Unlike sequence-to-sequence models [81], the decoder net does not determine total duration based on a "terminate" token; rather, it uses a predefined frame-level mapping from the source frames to target frames as a local attention mechanism and fixed total duration. The attention mechanism is called implicit alignment in this context, which is akin to local-p attention mechanism [50]. Since this proposed method avoids training sequence-to-sequence model, it is quick to train (a matter of hours) and has real-time performance at inference time. Section 4.2.1 introduces the notion of implicit alignment, Section 4.2.2 introduces its architecture and Section 4.2.3 shows how to combine it with FFTNet to produce the best performance.

4.2.1 Implicit alignment

Traditional parallel voice conversion is based on frame-level alignment: given the source and target voice speaking the same utterances, acoustic features are extracted frame-by-frame and then aligned using dynamic time warping [54]. The assumption is that for each target frame, there is one and only one source frame that contains the corresponding acoustic content; and most parametric voice conversion method is based on inferring the acoustic feature of a target frame from its aligned source frame's features, utilizing a statistical model learned from training data. However, this assumption is an over-simplification due to three problems: first, parallel speech is imperfect. In case when the target has missing phonemes (linked words) and when the source has additional silences, the target frames will have to map to source frames that do not have the matching acoustic content. Second, DTW used in previous works is

based on L2 distance of MFCCs, a feature that in many cases inconsistent with human hearing. This problem may result in alignment errors, introducing incompatible source and target pairs. Lastly, the same phoneme may be spoken differently by different speakers, while frames within these phonemes should be aligned, but often than not they will cause alignment errors once again due to the problem of MFCC distance metrics. When alignment error occurs, the synthesized target frame may sound different or not as sharp. This is potential cause of oversmoothing problem in most parametric voice conversion and noises heard in data driven voice conversion.

To rethink about frame-by-frame alignment, let's first consider the scenario when a human speaker repeating the word "hello" in a unfamiliar foreign language after hearing a native speaker saying that word. Often the person listen the entire word, memorize the sound component of the word (for example phonemes) and repeat the word using what he/she own way of repeating those sounds components. If the word he/she is learning is long, it is also likely that he/she will ask the native speaker to speak syllable by syllable and repeat each syllable in the same manner. The short word example is in analogous to a sequence-to-sequence model, where the source voice is encoded and memorized before it is decoded using target's way of speaking. The second case is as if the target voice is generated by part under the influence of the corresponding part from the source voice. It is no longer a frame-by-frame alignment; the determination of a target frame feature is influenced by a sequence of source frames and its context as well.

Hence, we introduce the concept of implicit alignment: for each target frame, it is influenced by a collection of source frames or none of the source frames (silences). The term "influenced" means there exists a function that determine which frames in the source would influence the generation of a target frame and another function that infers the target frame based on the selected source frames. The former function is akin to attention mechanism used in sequence-to-sequence models [50]. The second

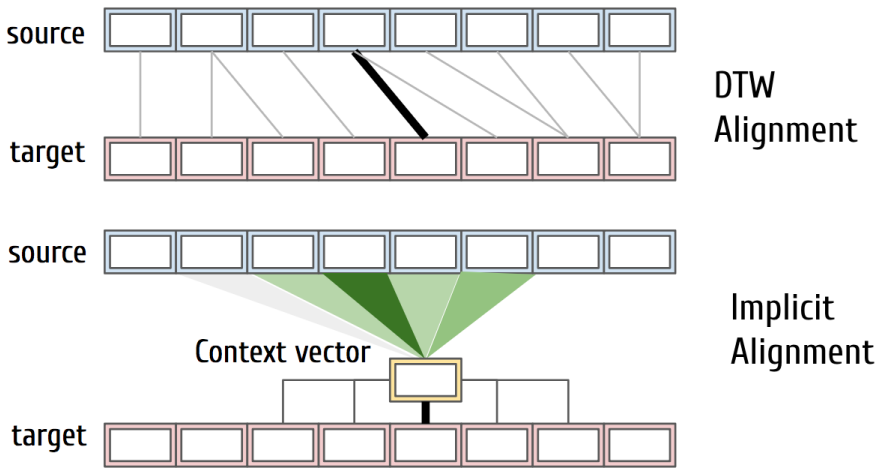


Figure 4.4: Demonstration of implicit alignment: conventional frame-to-frame alignment is one-to-one correspondence in non-decreasing manner computed by dynamic time warping algorithm. Implicit alignment maps multiple source frames to a target frame. The generation of a target frame's acoustic feature is based on a summarization of these source frames (called context vector) and neighbor target frames.

function should take the context feature (summarization from the source frame based on attention mechanism) to generate the target frame with the consideration of the target frame's context. This idea is illustrated in Figure 4.4. Section 4.2.2 propose a network that learns implicit alignment and the mapping jointly using recurrent neural networks.

4.2.2 Architecture

TimbreNet is consisted of two recurrent neural networks, an encoder net and a decoder net. The encoder net first transform the input source acoustic feature into a higher-dimensional embedding space using 1×1 convolution (`1x1conv` for short) and then encode the sequence using an Bi-directional LSTM. The hidden layers of LSTM at each time step is regarded as the output of the encoder net, denoted as $\{H_1, H_2, ..., H_n\}$. The decoder net starts with a Bi-directional LSTM that has the same number of time step as the number of frames of the target.

Denote the hidden layers of this LSTM as $\{D_1, D_2, ..., D_m\}$ As fore-mentioned in Section 4.2.1, each target frame D_j is generated under the influence of a collection of source frames; this relationship is reflected using a weighting vector $\{a_1, a_2, ..., a_n\}$ over the hidden layers of the encoder net. The input to each time step of the decoder net is decided by this weighting - a linear combination of encoder net's hidden layers: $z_j = \sum_{i=1}^{n} a_i H_i$ while each a_i is determined by a function evaluating the interaction between H_i and the hidden layer of the previous time step D_{j-1}, for example $a_i = W\text{concat}(H_i, D_{j-1})$ where `concat` concatenates two vectors and W is a learnable vector (same dimensionality as concat(H_i, D_{j-1}) transposed). z_j is called a context vector. This formulation is exactly global attention mechanism used in sequence to sequence models. The hidden layers of bi-directional LSTM is further connected to an uni-directional LSTM which is followed by several convolutional layers. In this work, a 5×1 convolutional layer and a `1x1conv` are used to generate the acoustic features of the target voice.

However, the global attention mechanism mentioned above is computationally heavy and often outputs a weighting vector $\{a_1, a_2, ..., a_n\}$ that has energy all over the input sequence. Therefore we adopt the idea of local-p attention mechanism [50] in this work utilizing the frame-to-frame alignment used in traditional voice conversion. Instead of computing the weight vector across all encoder net's layers, we only compute for those that are around a key frame based on this frame-to-frame alignment. The precise steps are as follows: first, extract 12-coefficient MFCC features for source and target and transform them into exemplars as in Section 2.2.2. Then, perform forced alignment on source and target and align phonemes using the method proposed in Section 3.4.2. Define a distance metric between source and target frame as the L2 distance of MFCC exemplars added with a phoneme matching cost. If the phoneme of the target frame aligns with the phoneme of the source frame, no cost is added, otherwise, a large amount of cost is added. Finally, we use dynamic

time warping (Section 2.2.1) to compute the best pairing based on this metric. My experiment shows that computing frame alignment this way produces better result than naively perform time warping on the L2 of 12-coefficient MFCC features directly.

Now we have a sequence of pairing in which one target frame can be associated with multiple source frames that are consecutive. To further produce one-to-one mapping, we define the center frame of all associated source frames as the one single source frame aligned to the target frame. The implicit alignment is a span of source frames surrounding this center frame. In this work where frames are extracted every 160 samples, a radius of 6 frames is used. This means 13 frames are considered in the implicit alignment. This length is often longer than a phoneme, making the alignment more robust to alignment errors. When computing context vector z_j for target frame j which is aligned with to frame i in the source, only $\{a_{j-6}, ..., a_{j+6}\}$ are considered. Namely $z_j = \sum_{t=-6}^{6} a_{i+t} H_{i+t}$. The entire TimbreNet architecture is illustrated in Figure 4.5.

4.2.3 Bottlenecked FFTNet

The last piece of puzzle is determining which acoustic feature to use for TimbreNet. One could adopt the mel-spectrogram features that were used in TacoTron [73] but our experiment shows poor result using one hour of training data. TimbreNet seems to overfit significantly even if we use high drop rate (50%) with very small hidden layer size (64). However, the performance gets much better if using 24-coefficient MGC with energy [92]. This means the network will generate Timbre of the sound while F0 is to be determined using a separate process such as another RNN mapping phoneme to pitch contour [96].

Although using MGC as acoustic feature achieves significantly better result than mel-spectrogram, the synthesis result (using FFTNet built from MGC+F0) still suffer from audible oversmoothing artifact and noise component that is stronger than using

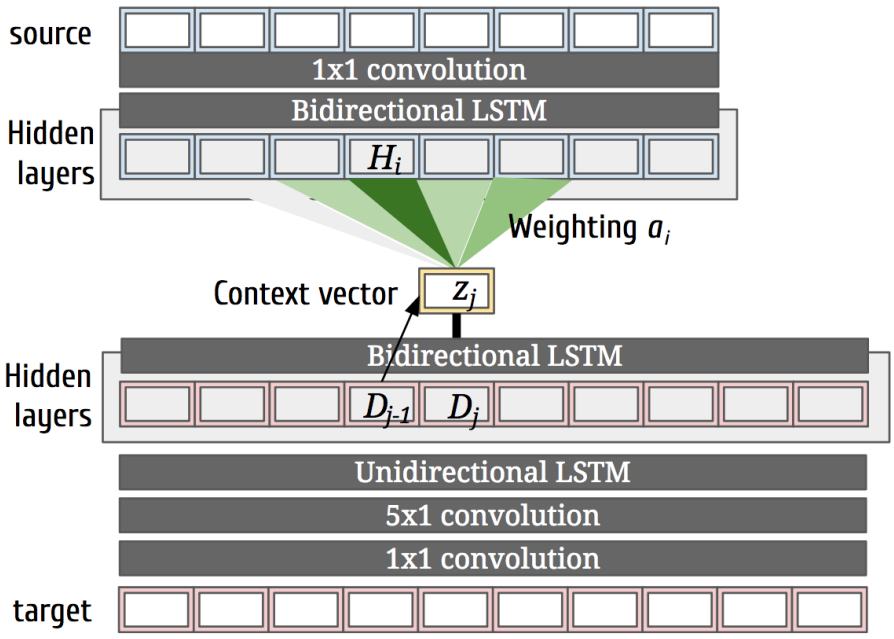


Figure 4.5: TimbreNet illustrated: the input (source acoustic features) are passed through `1x1conv` and then a bidirectional LSTM. For each target frame, a context vector is produced based on the implicit alignment. The context vector is served as input to another Bidirectional LSTM followed by a uni-directional LSTM and several convolutional layers that outputs the final acoustic features for the target frames.

the true MGC+F0 extracted from real recordings. Figure 4.6 shows an example generated MGC compared to the true MGC. I hypothesize two main causes of oversmoothing problem: first, MGC feature is based on log-spectra, an estimator that is inconsistent [71] resulting in noisy MGC estimation; secondly, MGC still contains fair amount of pitch information and when F0 and the pitch content of MGC are incompatible, FFTNet tends to generate "flatter" posterior distribution, resulting more noises. Therefore, I hope to develop a new acoustic feature that is independent from F0 and has less noise than MGC. One way is to use lower-order MGC with artificial oversmoothing artifact injected during FFTNet training; however, this method requires tuning for the best oversmoothing artifact simulation for different dataset. The final solution however learns a new acoustic feature by simultaneously learning the feature andFFTNet for specific dataset.

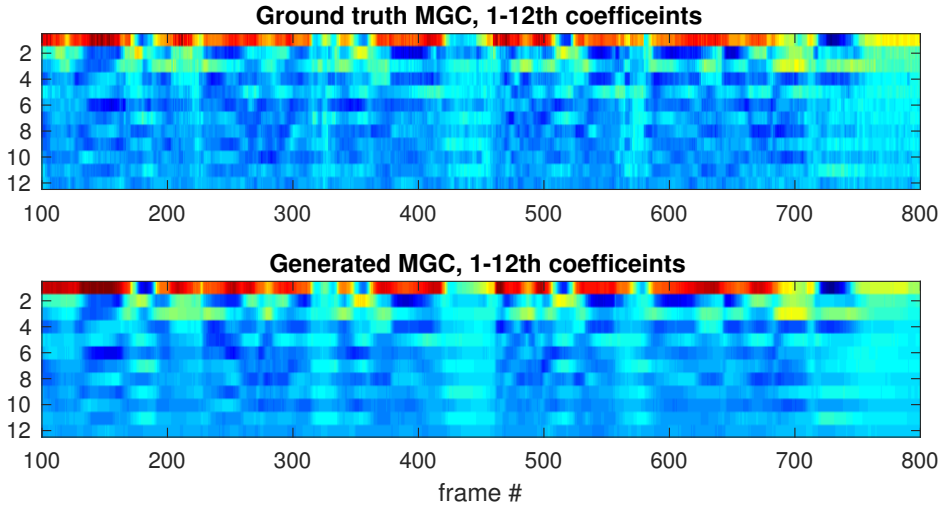


Figure 4.6: TimbreNet generated MGCs compared to ground true MGCs. Visually, the generated MGC does not reproduce the details that true MGCs have.

This is called a bottlenecked FFTNet. Instead of mixing the axillary condition directly into FFTNet layers based on Equation 4.1, we first transform the condition using convolutional neural networks. The first layer is a 1×5 convolution increase the channel of input from 24 (dimension of MGCs without the energy term) to a much higher number (e.g. 64). The resulting tensor is passed through a ReLU and then a 1×1 convolution reducing the channel number to a much lower one (e.g. 6). This 6-dimensional representation of MGC is called a bottleneck feature. Finally, the energy term is concatenated to the bottleneck feature, resulting in 7 dimensions. Then this 7-dimensional vector is upsampled and used as axillary condition for FFTNet. This feature transformation network and the FFTNet are trained jointly. Because of the low dimensional nature of this feature, containing F0 information is unfavorable because F0 is already observed in another axillary condition. To preserve as much of information about MGC as possible, this low dimensional feature should be rid of all F0 information. The first convolution also serves are a smoother to make the resulting acoustic feature more stable locally. Figure 4.7 shows a comparison between the original MGC and the learned bottleneck feature.

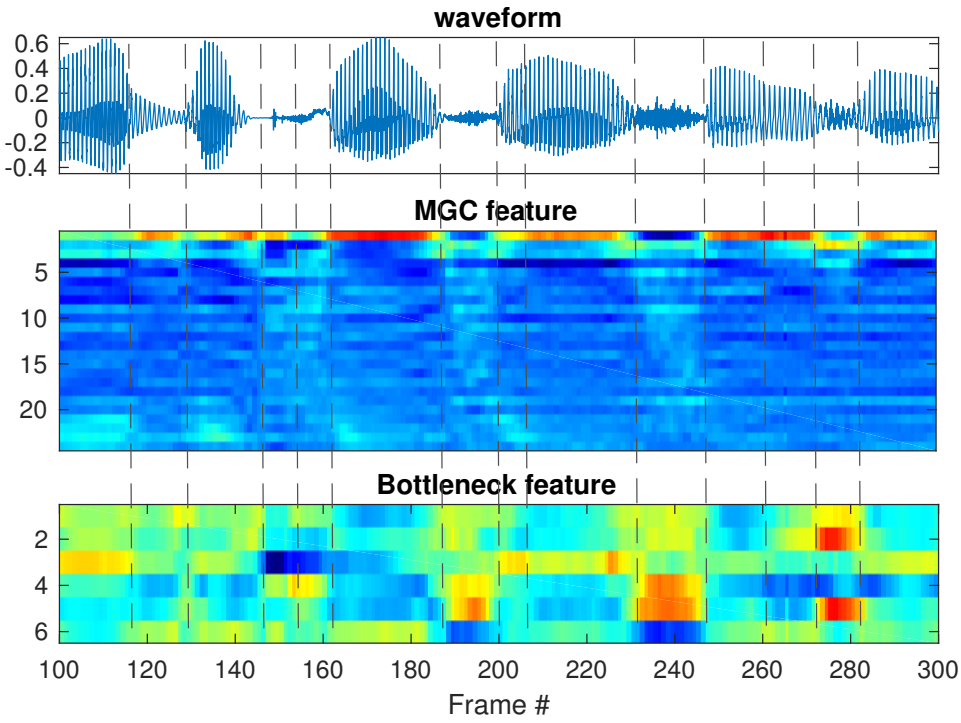


Figure 4.7: A comparison among the waveform, the original MGC feature (without the energy component) and the learned bottleneck feature. The dotted lines show the phoneme boundaries. Visually the bottleneck feature does not have the variation that MGC feature has and corresponds strongly with phoneme boundaries.

Finally, during TimbreNet training, we replace the target voice's acoustic feature with the bottleneck feature. The generated bottleneck feature looks much closer to the true bottleneck feature than the generate MGC to the true MGC. Auditorily, the resulting synthesis has much less oversmoothing artifact and less noise as well. We demonstrate our result in Section 4.3.

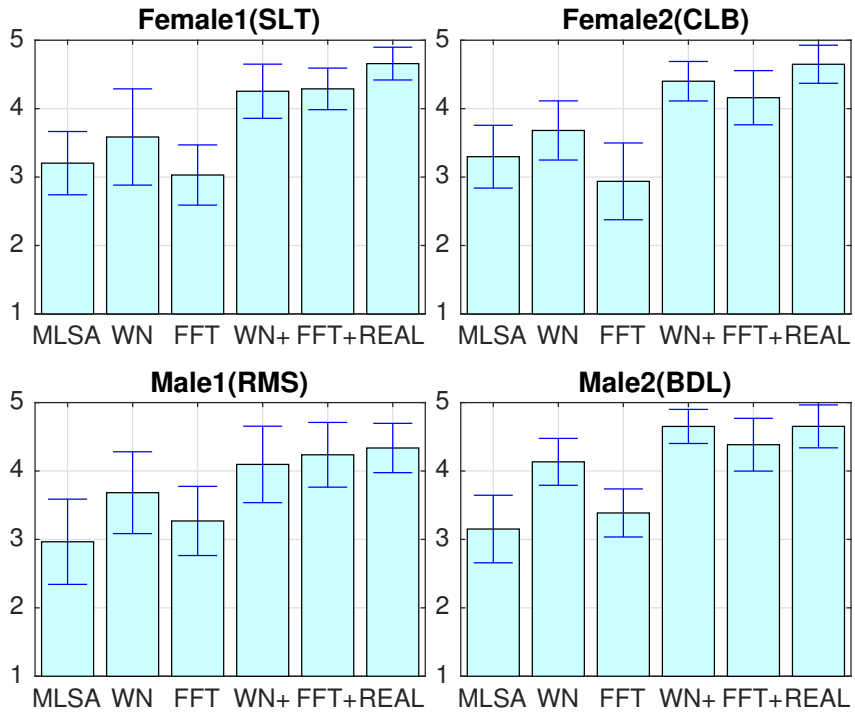
4.3 Evaluation

4.3.1 FFTNet

To evaluate our algorithm, we conduct both a subjective listening test using mean-opinion score (MOS) described in Section 3.5.2 to measure synthesis quality, and an objective test to measure spectral and cepstral distortion.

Experimental setup

Four voices, two male (BDL,RMS) and two female (SLT,CLB), from the CMU Arctic dataset [43] are used in our experiments. The first 1032 utterances (out of 1132) are used for training and the remaining are used for evaluation. The waveforms are quantized to 256 categorical values based on μ-law. 25-coefficient Mel Cepstral Coefficients (with energy) and F0 are extracted from the original samples. We build 4 networks for each voice, 2 WaveNets and 2 FFTNets. For each type of network, we used two training strategies: Strategy one is with zero padding but without using the other techniques in Section 2.3; Strategy two applies all techniques in Section 2.3. The WaveNet we implemented contains two stacks of 10-layer dilated convolution ($d = 2^0, 2^1, ..., 2^9$) with 256 dilation and 128 skip channels. The total receptive field is 2048 samples. We experimented with different numbers of channels and found the current configuration the best for the vocoder. The FFTNet implementation contains 11 FFT-layers with 256 channels, which also has a receptive field of 2048. Note that this FFTNet has less than 1M parameters and with proper caching [58], the computation cost for generating one second of audio (16kHz) is only around 16GFLOPs; it means a modern CPU can generate audio samples in real-time. In each training step, a minibatch of 5$\times$ 5000-sample sequences are fed to the network, optimized by the Adam algorithm [40] with a training rate 0.001. We set the variance of injected noise to be 1/256. In each minibatch, all sequences come from different



	MLSA	WN	FFT	WN+	FFT+	Real
MOS	3.16	3.77	3.16	4.35	4.27	4.57

Figure 4.8: Mean opinion score (MOS) test results show that the proposed method `FFT+` improves synthesis quality over the original WaveNet `WN`, and that our training/synthesis techniques (`+`) improve both original WaveNet and naive FFTNet (`FFT`). The bottom table shows the average MOS across four voices.

utterances; we trained WaveNet by 200,000 steps and FFTNet by 100,000 steps to ensure convergence.

In our implementations, synthesis using FFTNet is more than 70 times faster than Fast WaveNet [58], requiring only 0.81 second to generate one second of audio on a laptop CPU (2.5 GHz Intel Core i7). The audio clips and results of the experiments described in this section may be found at our project web page.[1]

We conducted a Mean Opinion Score (MOS) test described in 3.5.2 that asks subjects to rate the quality of the synthetic utterances. Our subjects were recruited via Amazon Mechanical Turk (AMT), a micro-task platform popular for crowdsourcing experiments [12]. We recruited participants from the United States who have an ap-

[1] `http://gfx.cs.princeton.edu/pubs/Jin_2018_FAR`

proval rate over 90% to ensure the reliability of the study results. We also designed a validation test to ensure that subjects are paying attention and rejected those who fail on those tests. AMT has been used for conducting MOS tests in various domains [25, 33, 35]. It has been shown to be effective for various kinds of listening tests [13, 60] and can provide a large number of diverse participants in a short amount of time. We evaluated six conditions for each utterance:

MLSA	MLSA filter
WN	WaveNet with only zero-padding
FFTN	FFTNet with only zero-padding
WN+	WaveNet with techniques in Section 2.3
FFTN+	FFTNet with techniques in Section 2.3
Real	the actual recording

In each task (called a *HIT*), a subject is presented with 32 different sentences in which 24 of them are made of 4 instances from each of the above 6 conditions. From a held-out set of sentences, we add 4 more instances of the "Real" condition and 4 more cases of badly edited "Fake" (3 bit A-law encoded) condition to validate that the subject is paying attention and not guessing randomly. For the data to be retained, the subject may make at most one mistake on these validation tests, by either rating < 3 on "Real" examples or > 3 on "Fake" examples. We launched 480 HITs (120 per voice) and retained 446 after validation.

4.8 shows a bar chart for the MOS test with the error bars indicating standard deviation across utterances. The proposed training technique improves both WaveNet and FFTNet significantly with an ANOVA test p-value less than 10^{-9} for both networks. The proposed network `FFT+` also improves on `WN` with a p-value of $< 10^{-20}$. Both `WN+` and `FFT+` have significant overlap with the real examples in MOS scores. The proposed method `FFT+` has a slightly lower MOS than WaveNet `WN+` (with an

	MCD (dB)				RMSE (dB)			
voice	slt	clb	rms	bdl	slt	clb	rms	bdl
mlsa	2.76	3.03	3.62	3.28	8.05	9.14	8.80	8.25
WN	4.47	4.04	4.60	3.05	9.71	9.65	9.38	8.29
WN+	4.57	4.13	4.41	3.28	9.80	8.95	9.74	8.67
FFT	5.24	5.07	4.82	4.23	10.39	9.77	10.33	10.13
FFT+	4.73	4.69	4.41	3.82	9.88	9.58	9.89	9.64

Table 4.1: Comparison of distortion between natural speech and synthetic speech based on Average MCD and RMS

insignificant p-value); but it is significantly faster, as noted above. It is also worth noting that `FFT` has similar quality to the baseline method `MLSA` (insignificant p-value) due to noisy artifacts; this implies the training and synthesis techniques are essential to make FFTNet work well.

We evaluated the distortion between the original and the synthesized speech using RMSE and MCD [85]. RMSE measures frequency domain difference between two signals; and MCD measures the difference in the cepstral domain, which reflects whether the synthesized speech can capture the characteristics of the original speech. Both measurements are in dB. The result is shown in Table 1.

The result shows that MLSA tends to preserve most of the cepstral and spectral structure while the MOS test puts it in a significantly lower tier as it generates audible over-smoothing artifacts. The techniques introduced in Section 2.3 do not reduce distortion in WaveNet but they significantly improve FFTNet in both metrics. Also, note that the WaveNet with the proposed techniques performs significantly better in subjective evaluation than the one without. This result also contradicts MCD and RMSE.

4.3.2 DeepVoCo

To evaluate DeepVoCo, we conducted MOS test and identification test as described in Section 3.5. We selected two voices from CMU ARCTIC dataset, one male (BDL)

and one female (SLT). For each voice, we first build an FFTNet with bottlenecked feature with a dimensionality of 8. The initial acoustic feature used to train the bottlenecked feature is 25-coefficient MGC. F0 is regarded as a second auxiliary condition in addition to the bottlenecked feature. The network is trained using the techniques described in Section 4.1.3 for 100K steps (batch size of 6×6000) using Adam optimizer (0.0005 learning rate). The injected has a max amplitude of 0.003 and is sampled using a uniform distribution. The receptive field is set to 2048 and the channels in FFTNet is set to 256.

To create training data for TimbreNet, we take the first 800 utterances (30 minutes) for each voice and convert their MGC features into bottlenecked features based on the learned network. Then we use the same synthesizer used in VoCo to produce the source utterances and extracted their 25-coefficient MGC features and F0. Then we compute alignment using the method described in Section **??**. Finally we train a TimbreNet to convert source voice's MGC features into the target's bottlenecked features. For each minibatch, we only use one parallel sequence at a time because every time the network is structure is different based on implicit alignment. Therefore, a small learning rate (1e-5) is used and we train the network 10K iterations. To synthesize a word for word replacement, we use the source synthesizer to speak the entire sentence with the new word in it and convert it to the target voice using FFTNet. Note that the alignment we use to generate new samples is uniform alignment, meaning each source frame is mapped the a target frame at the same time stamp. The source voice's F0 is also transferred to the target using the pitch normalization function used in Section 2.2.3. We generate the entire sentence using FFTNet and cut out the word from the synthesized sentence and insert it into the recording being edited.

We conducted the experiments on Mechanical turk using the same setting as we used in VoCo: in MOS test, for each sentence we remove and replace a word with

a synthesized word and ask a user to rate the quality of the synthesized word on a scale of one to five. In identification test, a user is presented with a sentence with a word replaced in the same way as described above or an actual recording; the task is to identify if the sentence contains a synthesized word or not. Both experiments are conducted in the same way as in Section 3.5 except that the conditions we compare are different, listed below:

- **Synth.** We use the source TTS voice to synthesize the word and put into context.
- **CUTE.** Based on VoCo synthesis framework, we use CUTE as the voice conversion method.
- **VoCo.** VoCo method using pre-defined α and β values in range selection.
- **Deep.** Deep VoCo method described above.
- **Real.** the actual recording without modification.

For each voice we collected valid 120 HITs for both tests, totaling 480 samples per condition. The MOS test result is shown in Figure 4.9, and identification test result is shown in Figure **??**.

From the result we can see that DeepVoCo improves over VoCo both male and female voices, pushing the average MOS to 4.3 and identification test score above 70%. It means words synthesized by DeepVoCo are confused for real recordings at a probability of 70% or more for CMU ARCTIC dataset. Auditorily, words synthesized by DeepVoCo do not have the "click" like sound that VoCo has. The examples that failed the identification test are those that contain slight amount of noise, which we theorize is caused by uncommon combination between the F0 and the generated bottlenecked features. Using an alternative F0 contour generated by external process or user input other than transferring from the source may fix the problem. It also

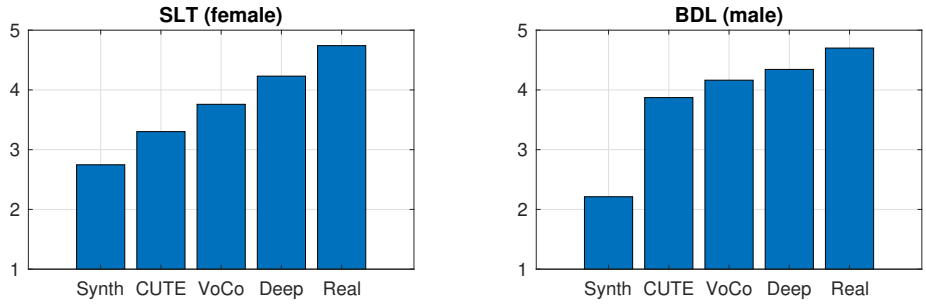


Figure 4.9: These mean opinion score tests show that DeepVoCo synthesis results are generally perceived as higher quality than those of baseline methods, scored on a Likert scale from 1=*bad* to 5=*excellent*.

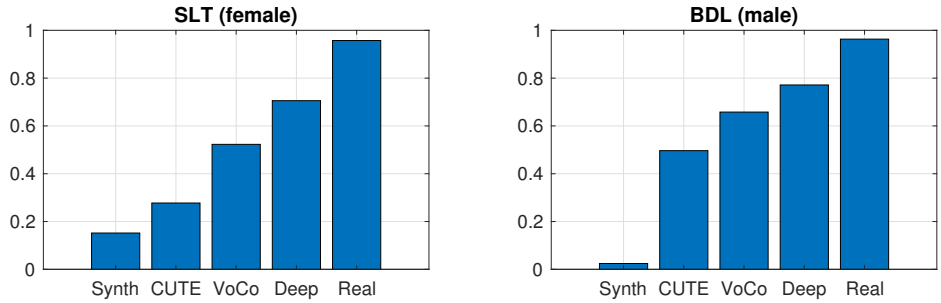


Figure 4.10: Identification tests show DeepVoCo synthesis results are more likely identified as original recordings than other baseline methods, shown here as a fraction of people who identify the recordings as original.

worth noting that DeepVoCo closes the quality gap between female and male voice because it is not a concatenative method and therefore free from artifact caused by stitching audio snippets of different pitch contour.

4.4 Summary and Discussion of Deep-VoCo

This chapter introduces FFTNet, a neural network that generates waveforms one sample at a time based on previously generated samples and axillary conditions. Unlike WaveNet, FFTNet uses a simple architecture mimicking the Fast Fourier Transform (FFT) that makes it possible to generate audio samples in real-time. We also propose a set of training and synthesis techniques that improve the synthesis quality of both FFTNet and WaveNet. Experiments demonstrate that when used as

a vocoder, FFTNet generates higher-quality speech than the original WaveNet, and on par with WaveNet as improved by our training and synthesis techniques.

This chapter then introduces TimbreNet, a voice conversion neural network that transforms a source voice's acoustic features into a target voice's acoustic features. It is based on two recurrent neural networks, encoder net and decoder net, bridged by implicit alignment, similar to attention mechnism used in sequence-to-sequence models. Because transforming MGC directly results in oversmoothing artifact, this chapter proposed learning a new bottleneck feature jointly with FFTNet and use this feature as the target acoustic feature instead. Experiment shows that TimbreNet is preferred over VoCo method in word replacement task.

Chapter 5

Conclusion

5.1 Summary

This thesis aims at enabling word synthesis in text-based speech editing with text-to-speech synthesis (TTS); while existing high fedelity TTS requires tens of hours of training data, this work takes a different route by using a generic TTS to synthesize a robotic version of the input text and then utilizing voice conversion to transform the robotic voice into the desired human voice. While existing voice conversion method contains artifact, my thesis work focuses on devising new voice conversion techniques so that the synthesized word can be seamlessly inserted or replaced with existing words in an actual recording. In particular, three methods are proposed: the first is called CUTE, a data-driven voice conversion method based on frame-level unit selection and exemplar features; experiments show that CUTE is preferred over other unit-selection based voice conversion method in parallel voice conversion task. The second method is called VoCo, a TTS system built on CUTE with several improvement including a unit-selection replacement called range selection algorithm that performs stabler and faster than unit selection framework. Mechanical turk experiments show that about 60% of the time, the synthesized word is confused as

real recorded words based on CMU ARCTIC dataset. To close further improve the synthesis quality and potentially make it possible to synthesize long sentences, the third method, called Deep-VoCo, uses deep learning to perform voice conversion in the VoCo TTS system. This method involves two novel neural networks: FFTNet, a neural vocoder that generates high quality waveform samples from acoustic features such as MCC and F0; and TimbreNet, a encoder-decoder network that transforms the generic TTS voice into the desired human voice in their acoustic feature space. This thesis shows FFTNet, the acoustic feature used in voice conversion and TimbreNet can be effectly trained jointly with less than one hour of data; and the resulting TTS system can be potentially used to synthesize long sentences as well.

5.2 Limitations and Future Work

There are several limitations in this thesis that suggest areas for future work. First, our ultimate goal is to synthesize passages that are longer than just a word or short phrase, with such consistently high quality that they are rarely recognizable as edited. This requires that the synthetic voice is not only sounds like the target person in timbre but in prosody (pitch and duration) as well. This is particularly challenging problem in general. The proposed deep-learning-based approach is able to synthesize sentences, but it relies on pre-defined prosody. Our current approach uses the source speaker's prosody which is unaware of the target speaker's prosodic characteristics. Therefore, an immediate continuation of this research is to explore prosody modeling. There has been success in modeling prosody using RNN [96] and implicitly modeling F0 together with timbre using mel-spectrogram features [97] based on large data (20hours+). Unfortunately, our experiments show limited result over one hour data. Possible solution may lie in the area of adaptation, i.e.learning a generic model based

on a collection of speaker data and adapt the learned model to new speaker via fine tuning.

Another issue with deepVoCo has to do with sampling rate. Although we demonstrated real-time and high-quality performance of FFTNet at 16KHz sample rate, most of the audio product is at 44.1KHz or more. Unfortunately, at such sample rate FFTNet tend to require about 10 times longer to train and substantial increase in model size, resulting in laggy performance. Future work in FFTNet may consider band completion techniques that upsamples 16K speech signal to 44.1K with a process that's more efficient than sample by sample.

One advantage of VoCo over DeepVoCo is its short pre-processing time. Given one hour of audio, VoCo takes about 15 minutes using a CPU to ready the data for word synthesis while DeepVoCo requires a GPU to train the networks. In our experiment, FFTNet requires 12 hours to train and TimbreNet requires 3 hours based on A NVidia 1080 GTX GPU. Therefore, it is beneficial to develop new training techniques that shorten the training time for new voices. One way is to use adaptation described above. Another way is to build a speaker-independent FFTNet based on large population using speaker encoding [3].

Although the proposed methods in this thesis are primarily designed for text-based editing of audio narrations, they can be applied to generic TTS and voice conversion as well. The proposed neural network can be re-targeted to many other applications, including:

Many-to-one voice conversion: with the help of generic speech recognition it is possible to turn deepVoCo into many-to-one voice conversion by replacing the current acoustic feature with a new feature consistent with posteriorgrams [79]. As a result, any voice recognizable by the stated speech recognition engine can be converted into a common voice without training. It is useful in many applications, such as voice over, voice disguise, and for entertainment purposes.

Voice beautification: instead of transforming the entire voice, another application is to modify certain aspects of a voice to sound like another, such as adding accent and making a voice sound thicker and heavier. One application is to beautify a voice by learning from a collection of professional voices and convert a voice of a non-professional to sound more like them.

Singing synthesis: it is also possible to add singing to a TTS voice. Since FFTNet is a general purpose speaker-dependent vocoder, one can train it over singing samples and use the F0 (pitch) as axillary conditions together with the bottleneck feature. TimbreNet on the other hand, can take in F0, duration and phonemes as input to generate the bottleneck feature for FFTNet. With this architecture, a user create singing voice by providing a music score and align text (broken down into phonemes) to notes.

Bibliography

[1] Acapela Group. `http://www.acapela-group.com`, 2016. Accessed: 2016-04-10.

[2] Ryo Aihara, Toru Nakashika, Tetsuya Takiguchi, and Yasuo Ariki. Voice conversion based on non-negative matrix factorization using phoneme-categorized dictionary. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2014)*, 2014.

[3] S. O. Arik, J. Chen, K. Peng, W. Ping, and Y. Zhou. Neural Voice Cloning with a Few Samples. *ArXiv e-prints*, February 2018.

[4] Sercan Arik, Gregory Diamos, Andrew Gibiansky, John Miller, Kainan Peng, Wei Ping, Jonathan Raiman, and Yanqi Zhou. Deep voice 2: Multi-speaker neural text-to-speech. *arXiv preprint arXiv:1705.08947*, 2017.

[5] Sercan O Arik, Mike Chrzanowski, Adam Coates, Gregory Diamos, Andrew Gibiansky, Yongguo Kang, Xian Li, John Miller, Jonathan Raiman, Shubho Sengupta, et al. Deep voice: Real-time neural text-to-speech. *arXiv preprint arXiv:1702.07825*, 2017.

[6] Floraine Berthouzoz, Wilmot Li, and Maneesh Agrawala. Tools for placing cuts and transitions in interview video. *ACM Trans. on Graphics (TOG)*, 31(4):67, 2012.

[7] Alan W Black. Unit selection and emotional speech. In *Eighth European Conference on Speech Communication and Technology*, 2003.

[8] Alan W Black and Paul A Taylor. Automatically clustering similar units for unit selection in speech synthesis. 1997.

[9] Alan W Black, Heiga Zen, and Keiichi Tokuda. Statistical parametric speech synthesis. In *Proc. ICASSP*, volume 4, pages IV–1229, 2007.

[10] Paulus Petrus Gerardus Boersma et al. Praat, a system for doing phonetics by computer. *Glot international*, 5, 2002.

[11] Christoph Bregler, Michele Covell, and Malcolm Slaney. Video rewrite: Driving visual speech with audio. In *Proceedings of the 24th Annual Conference on*

Computer Graphics and Interactive Techniques, SIGGRAPH '97, pages 353–360, 1997.

[12] Michael Buhrmester, Tracy Kwang, and Samuel D Gosling. Amazon's mechanical turk: A new source of inexpensive, yet high-quality, data? *Perspectives on psychological science*, 6(1):3–5, 2011.

[13] Mark Cartwright, Bryan Pardo, Gautham J Mysore, and Matt Hoffman. Fast and easy crowdsourced perceptual audio evaluation. In *Proc. ICASSP*, pages 619–623, 2016.

[14] Juan Casares, A Chris Long, Brad A Myers, Rishi Bhatnagar, Scott M Stevens, Laura Dabbish, Dan Yocum, and Albert Corbett. Simplifying video editing using metadata. In *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques*, pages 157–166. ACM, 2002.

[15] Ling-Hui Chen, Zhen-Hua Ling, Li-Juan Liu, and Li-Rong Dai. Voice conversion using deep neural networks with layer-wise generative training. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, 22(12):1859–1872, 2014.

[16] A. Cheveigné and H. Kawahara. Yin, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111(4):1917–1930, 2002.

[17] Alistair D Conkie and Stephen Isard. Optimal coupling of diphones. In *Progress in speech synthesis*, pages 293–304. Springer, 1997.

[18] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[19] Srinivas Desai, E Veera Raghavendra, B Yegnanarayana, Alan W , and Kishore Prahallad. Voice conversion using artificial neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2009)*, 2009.

[20] Homer Dudley. The vocoder – electrical re-creation of speech. *Journal of the Society of Motion Picture Engineers*, 34(3):272–278, 1940.

[21] Thierry Dutoit. *An introduction to text-to-speech synthesis*, volume 3. Springer Science & Business Media, 1997.

[22] Thierry Dutoit. *Corpus-Based Speech Synthesis*, pages 437–456. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[23] Thierry Dutoit, Andre Holzapfel, Matthieu Jottrand, Alexis Moinet, J Prez, and Yannis Stylianou. Towards a voice conversion system based on frame selection. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2007)*, 2007.

[24] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders. *arXiv preprint arXiv:1704.01279*, 2017.

[25] Maxine Eskenazi, Gina-Anne Levow, Helen Meng, Gabriel Parent, and David Suendermann. *Crowdsourcing for speech processing: Applications to data collection, transcription and assessment*. John Wiley & Sons, 2013.

[26] G David Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[27] Kei Fujii, Jun Okawa, and Kaori Suigetsu. High-individuality voice conversion based on concatenative speech synthesis. *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering*, 1(11):1617 – 1622, 2007.

[28] François G. Germain, Gautham J. Mysore, and Takako Fujioka. Equalization matching of speech recordings in real-world environments. In *41st IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP 2016)*, 2016.

[29] Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless, and David H Salesin. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 327–340. ACM, 2001.

[30] H. W. Hon and K. F. Lee. Cmu robust vocabulary-independent speech recognition system. In *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, pages 889–892. IEEE, 1991.

[31] Andrew J Hunt and Alan W Black. Unit selection in a concatenative speech synthesis system using a large speech database. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 1996)*, pages 373–376, 1996.

[32] Satoshi Imai, Kazuo Sumita, and Chieko Furuichi. Mel log spectrum approximation (mlsa) filter for speech synthesis. *Electronics and Communications in Japan (Part I: Communications)*, 66(2):10–18, 1983.

[33] Zeyu Jin, Adam Finkelstein, Stephen DiVerdi, Jingwan Lu, and Gautham J Mysore. Cute: A concatenative method for voice conversion using exemplar-based unit selection. In *Proc. ICASSP*, pages 5660–5664, 2016.

[34] Zeyu Jin, Adam Finkelstein, Gautham J. Mysore, and Jingwan Lu. Fftnet: a real-time speaker-dependent neural vocoder. In *Proc. ICASSP*, 2018.

[35] Zeyu Jin, Gautham J. Mysore, Stephen Diverdi, Jingwan Lu, and Adam Finkelstein. Voco: Text-based insertion and replacement in audio narration. *ACM Trans. Graph.*, 36(4):96:1–96:13, July 2017.

[36] Alexander Kain and Michael W Macon. Spectral voice conversion for text-to-speech synthesis. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 1998)*, pages 285–288, 1998.

[37] Hideki Kawahara, Ikuyo Masuda-Katsuse, and Alain de Cheveigné. Restructuring speech representations using a pitch-adaptive time-frequency smoothing and an instantaneous-frequency-based f0 extraction: Possible role of a repetitive structure in sounds. *Speech Communication*, 27:187–207, 1999.

[38] Hideki Kawahara, Masanori Morise, Toru Takahashi, Ryuichi Nisimura, Toshio Irino, and Hideki Banno. Tandem-straight: A temporally stable power spectral representation for periodic signals and applications to interference-free spectrum, f0, and aperiodicity estimation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2008)*, pages 3933–3936, 2008.

[39] Hisashi Kawai, Tomoki Toda, Jinfu Ni, Minoru Tsuzaki, and Keiichi Tokuda. Ximera: A new tts from atr based on corpus-based technologies. In *Fifth ISCA Workshop on Speech Synthesis*, 2004.

[40] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[41] S P Kishore and Alan W Black. Unit size in unit selection speech synthesis. In *IN PROC. EUROSPEECH 2003*, pages 1317–1320, 2003.

[42] Kazuhiro Kobayashi, Tomoki Hayashi, Akira Tamamori, and Tomoki Toda. Statistical voice conversion with wavenet-based waveform generation. *Proc. Interspeech 2017*, pages 1138–1142, 2017.

[43] John Kominek and Alan W Black. The cmu arctic speech databases. In *Fifth ISCA Workshop on Speech Synthesis*, 2004.

[44] Robert F. Kubichek. Mel-cepstral distance measure for objective speech quality assessment. In *Proceedings of IEEE Pacific Rim Conference on Communications Computers and Signal Processing*, pages 125–128, 1993.

[45] Sergey Levine, Christian Theobalt, and Vladlen Koltun. Real-time prosody-driven synthesis of body language. *ACM Trans. Graph.*, 28(5):172:1–172:10, December 2009.

[46] Philipos C Loizou. *Speech enhancement: theory and practice*. CRC press, 2013.

[47] Jaime Lorenzo-Trueba, Junichi Yamagishi, Tomoki Toda, Daisuke Saito, Fernando Villavicencio, Tomi Kinnunen, and Zhenhua Ling. The voice conversion challenge 2018: Promoting development of parallel and nonparallel methods. *arXiv preprint arXiv:1804.04262*, 2018.

[48] Jingwan Lu, Fisher Yu, Adam Finkelstein, and Stephen DiVerdi. Helpinghand: Example-based stroke stylization. *ACM Trans. Graph.*, 31(4):46:1–46:10, July 2012.

[49] Michal Lukáč, Jakub Fišer, Jean-Charles Bazin, Ondřej Jamriška, Alexander Sorkine-Hornung, and Daniel Sýkora. Painting by feature: Texture boundaries for example-based image creation. *ACM Trans. Graph.*, 32(4):116:1–116:8, July 2013.

[50] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[51] Anderson F Machado and Marcelo Queiroz. Voice conversion: A critical survey. *Proc. Sound and Music Computing (SMC)*, pages 1–8, 2010.

[52] Winter Mason and Siddharth Suri. Conducting behavioral research on Amazons Mechanical Turk. *Behavior research methods*, 44(1):1–23, 2012.

[53] Thomas Merritt, Robert AJ Clark, Zhizheng Wu, Junichi Yamagishi, and Simon King. Deep neural network-guided unit selection synthesis. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5145–5149. IEEE, 2016.

[54] Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques. *arXiv preprint arXiv:1003.4083*, 2010.

[55] Jumpei Niwa, Takenori Yoshimura, Kei Hashimoto, Keiichiro Oura, Yoshihiko Nankaku, and Keiichi Tokuda. Statistical voice conversion based on wavenet. In *Proc. ICASSP*, 2018.

[56] Tadashi Okubo, Ryo Mochizuki, and Tetsunori Kobayashi. Hybrid voice conversion of unit selection and generation using prosody dependent hmm. *IEICE transactions on information and systems*, 89(11):2775–2782, 2006.

[57] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[58] Tom Le Paine, Pooya Khorrami, Shiyu Chang, Yang Zhang, Prajit Ramachandran, Mark A Hasegawa-Johnson, and Thomas S Huang. Fast wavenet generation algorithm. *arXiv preprint arXiv:1611.09482*, 2016.

[59] Kun-Youl Park and Hyung Soon Kim. Narrowband to wideband conversion of speech using gmm based transformation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2000)*, volume 3, pages 1843–1846 vol.3, 2000.

[60] Jeanne Parson, Daniela Braga, Michael Tjalve, and Jieun Oh. Evaluating voice quality and speech synthesis using crowdsourcing. In *International Conference on Text, Speech and Dialogue*, pages 233–240. Springer, 2013.

[61] H. A. Patil, T. B. Patel, N. J. Shah, H. B. Sailor, R. Krishnan, G. R. Kasthuri, T. Nagarajan, L. Christina, N. Kumar, V. Raghavendra, S. P. Kishore, S. R. M. Prasanna, N. Adiga, S. R. Singh, K. Anand, P. Kumar, B. C. Singh, S. L. Binil Kumar, T. G. Bhadran, T. Sajini, A. Saha, T. Basu, K. S. Rao, N. P. Narendra, A. K. Sao, R. Kumar, P. Talukdar, P. Acharyaa, S. Chandra, S. Lata, and H. A. Murthy. A syllable-based framework for unit selection synthesis in 13 indian languages. In *Oriental COCOSDA held jointly with 2013 Conference on Asian Spoken Language Research and Evaluation (O-COCOSDA/CASLRE), 2013 International Conference*, pages 1–8, Nov 2013.

[62] Amy Pavel, Dan B. Goldman, Björn Hartmann, and Maneesh Agrawala. Sceneskim: Searching and browsing movies using synchronized captions, scripts and plot summaries. In *Proceedings of the 28th annual ACM symposium on User interface software and technology (UIST 2015)*, pages 181–190, 2015.

[63] Amy Pavel, Björn Hartmann, and Maneesh Agrawala. Video digests: A browsable, skimmable format for informational lecture videos. In *Proceedings of the 27th annual ACM symposium on User interface software and technology (UIST 2014)*, pages 573–582, 2014.

[64] Lawrence Rabiner and Ronald Schafer. *Theory and Applications of Digital Speech Processing*. Prentice Hall Press, 2010.

[65] Bhiksha Raj, Tuomas Virtanen, Sourish Chaudhuri, and Rita Singh. Non-negative matrix factorization based compensation of music for automatic speech recognition. In *Interspeech 2010*, pages 717–720, 2010.

[66] Miguel Varela Ramos. Voice conversion with deep learning. *Tecnico Lisboa Masters thesis*, 2016.

[67] Dario Rethage, Jordi Pons, and Xavier Serra. A wavenet for speech denoising. *CoRR*, abs/1706.07162, 2017.

[68] Marc Roelands and Werner Verhelst. Waveform similarity based overlap-add (wsola) for time-scale modification of speech: structures and evaluation. In *EUROSPEECH 1993*, pages 337–340, 1993.

[69] Steve Rubin, Floraine Berthouzoz, Gautham J. Mysore, Wilmot Li, and Maneesh Agrawala. Content-based tools for editing audio stories. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST 2013, pages 113–122, 2013.

[70] KIM Sang-Jin and HAHN Minsoo. Two-band excitation for hmm-based speech synthesis. *IEICE TRANSACTIONS on Information and Systems*, 90(1):378–381, 2007.

[71] L.L. Scharf and C. Demeure. *Statistical Signal Processing: Detection, Estimation, and Time Series Analysis*. Addison-Wesley series in electrical and computer engineering. Addison-Wesley Publishing Company, 1991.

[72] Marc Schröder. Expressive speech synthesis: Past, present, and possible futures. In *Affective information processing*, pages 111–126. Springer, 2009.

[73] Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, RJ Skerry-Ryan, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. *arXiv preprint arXiv:1712.05884*, 2017.

[74] Kåre Sjölander. An hmm-based system for automatic segmentation and alignment of speech. In *Proceedings of Fonetik 2003*, pages 93–96, 2003.

[75] Jose Sotelo, Soroush Mehri, Kundan Kumar, Joao Felipe Santos, Kyle Kastner, Aaron Courville, and Yoshua Bengio. Char2wav: End-to-end speech synthesis. In *ICLR 2017 workshop*, 2017.

[76] Matthew Stone, Doug DeCarlo, Insuk Oh, Christian Rodriguez, Adrian Stere, Alyssa Lees, and Chris Bregler. Speaking with hands: Creating animated conversational characters from recordings of human performance. *ACM Trans. Graph.*, 23(3):506–513, August 2004.

[77] Yannis Stylianou. Voice transformation: A survey. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2009)*, pages 3585–3588, 2009.

[78] Yannis Stylianou, Olivier Cappé, and Eric Moulines. Continuous probabilistic transform for voice conversion. *IEEE Transactions on Speech and Audio Processing*, 6(2):131–142, 1998.

[79] Lifa Sun, Kun Li, Hao Wang, Shiyin Kang, and Helen Meng. Phonetic posteriorgrams for many-to-one voice conversion without parallel data training. In *Multimedia and Expo (ICME), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

[80] David Sundermann, Harald Hoge, Antonio Bonafonte, Hermann Ney, Alan Black, and Shri Narayanan. Text-independent voice conversion based on unit selection. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2006)*, May 2006.

[81] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[82] Makoto Tachibana, Shinsuke Izawa, Takashi Nose, and Takao Kobayashi. Speaker and style adaptation using average voice model for style control in

hmm-based speech synthesis. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4633–4636. IEEE, 2008.

[83] Shinnosuke Takamichi, Tomoki Toda, Alan W Black, and Satoshi Nakamura. Modulation spectrum-constrained trajectory training algorithm for gmm-based voice conversion. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2015)*, 2015.

[84] NOSE Takashi, Junichi Yamagishi, Takashi Masuko, and Takao Kobayashi. A style control technique for hmm-based expressive speech synthesis. *IEICE TRANSACTIONS on Information and Systems*, 90(9):1406–1413, 2007.

[85] Akira Tamamori, Tomoki Hayashi, Kazuhiro Kobayashi, Kazuya Takeda, and Tomoki Toda. Speaker-dependent wavenet vocoder. In *Proceedings of Interspeech*, pages 1118–1122, 2017.

[86] Ke Tan, Jitong Chen, and DeLiang Wang. Gated residual networks with dilated convolutions for supervised speech separation. In *Proc. ICASSP*, 2018.

[87] Paul Taylor. *Text-to-Speech Synthesis*. Cambridge University Press, 2009.

[88] Tomoki Toda, Alan W Black, and Keiichi Tokuda. Voice conversion based on maximum-likelihood estimation of spectral parameter trajectory. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(8):2222–2235, 2007.

[89] Tomoki Toda, Yamato Ohtani, and Kiyohiro Shikano. One-to-many and many-to-one voice conversion based on eigenvoices. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2007)*, pages IV–1249, 2007.

[90] Tomoki Toda, Hiroshi Saruwatari, and Kiyohiro Shikano. Voice conversion algorithm based on gaussian mixture model with dynamic frequency warping of straight spectrum. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001)*, pages 841–844, 2001.

[91] K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura. Speech parameter generation algorithms for hmm-based speech synthesis. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, volume 3, pages 1315–1318 vol.3, 2000.

[92] Keiichi Tokuda, Takao Kobayashi, Takashi Masuko, and Satoshi Imai. Mel-generalized cepstral analysis-a unified approach to speech spectral estimation. In *Third International Conference on Spoken Language Processing*, 1994.

[93] Keiichi Tokuda, Yoshihiko Nankaku, Tomoki Toda, Heiga Zen, Junichi Yamagishi, and Keiichiro Oura. Speech synthesis based on hidden markov models. *Proceedings of the IEEE*, 101(5):1234–1252, May 2013.

[94] Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Koray Kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with pixelcnn decoders. In *NIPS*, pages 4790–4798, 2016.

[95] Aäron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C. Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel wavenet: Fast high-fidelity speech synthesis. *CoRR*, abs/1711.10433, 2017.

[96] Xin Wang, Shinji Takaki, and Junichi Yamagishi. An rnn-based quantized f0 model with multi-tier feedback links for text-to-speech synthesis. *Proc. Interspeech 2017*, pages 1059–1063, 2017.

[97] Yuxuan Wang, R. J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc V. Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. Tacotron: A fully end-to-end text-to-speech synthesis model. *CoRR*, abs/1703.10135, 2017.

[98] Steve Whittaker and Brian Amento. Semantic speech editing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI 2004, pages 527–534, 2004.

[99] Zhizheng Wu, Cassia Valentini-Botinhao, Oliver Watts, and Simon King. Deep neural networks employing multi-task learning and stacked bottleneck features for speech synthesis. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4460–4464. IEEE, 2015.

[100] Zhizheng Wu, Tuomas Virtanen, Tomi Kinnunen, Engsiong Chng, and Haizhou Li. Exemplar-based unit selection for voice conversion utilizing temporal information. In *INTERSPEECH 2013*, pages 3057–3061, 2013.

[101] Junichi Yamagishi, Takao Kobayashi, Yuji Nakano, Katsumi Ogata, and Juri Isogai. Analysis of speaker adaptation algorithms for hmm-based speech synthesis and a constrained smaplr adaptation algorithm. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(1):66–83, 2009.

[102] Takayoshi YoshimuraÝ, Keiichi TokudaÝ, Takashi MasukoÝÝ, Takao KobayashiÝÝ, and Tadashi KitamuraÝ. Simultaneous modeling of spectrum, pitch and duration in hmm-based speech synthesis.

[103] Heiga Ze, Andrew Senior, and Mike Schuster. Statistical parametric speech synthesis using deep neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7962–7966. IEEE, 2013.