BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

NON-PARALLEL VOICE CONVERSION

NON-PARALLEL VOICE CONVERSION

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **JAN BRUKNER**
AUTOR PRÁCE

SUPERVISOR **doc. Dr. Ing. JAN ČERNOCKÝ**
VEDOUCÍ PRÁCE

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2019/2020

Master's Thesis Specification

Student: **Brukner Jan, Bc.**
Programme: Information Technology Field of study: Computer Graphics and Multimedia
Title: **Non-Parallel Voice Conversion**
Category: Speech and Natural Language Processing
Assignment:

1. Get acquainted with techniques for non-parallel voice conversion.
2. Obtain internet quality data for training (eg. from YouTube, etc).
3. Implement (possibly with available toolkit(s)) an existing system and evaluate the results.
4. Propose improvements of the existing system or design a new one.
5. Implement modifications and evaluate the results.
6. Create a poster and/or short video presenting your work.

Recommended literature:

- according to supervisor's advice

Requirements for the semestral defence:

- Items 1 to 4 of the assignment.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Černocký Jan, doc. Dr. Ing.**
Consultant: Kinnunen Tomi, University of Eastern Finland
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: June 3, 2020
Approval date: November 1, 2019

Abstract

Voice conversion (VC) aims at converting the voice of source speaker to the voice of target speaker. It is popular in funny Internet videos but has also series of serious use cases, such as dubbing of audiovisual material and anonymization of voice (for example for witness protection). As it can serve for spoofing of voice identification systems, it is also an important tool for development spoofing detectors and counter-measures.

Training VC models has mainly been on parallel audios (ie. two speakers uttering the same text) and on high quality audio material. The goal of this thesis was to investigate developing VC on non-parallel data and with low quality signals, mainly from publicly available dataset VoxCeleb.

This work follows the state-of-the-art AutoVC architecture defined by Qian et al. It is based on neural network (NN) autoencoders, aiming to separate speech into content- and speaker-dependent embedding. The target speech is then obtained by replacing source speaker embedding by the target speaker one. We have improved Qian's architecture to process low-quality audio by experimenting with different speaker embeddings (d-vectors vs. x-vectors), introducing a speaker classifier from content embeddings in an adversarial setup, and tuning the size of content embeddings imposing an information bottleneck to the autoencoder. Also, we have defined another adversarial architecture by comparing original content embeddings with those obtained after the VC process.

The results of experiments prove that non-parallel VC on low-quality data is indeed doable. The resulting audios were not so good as in case of using high-quality ones, but the speaker verification results after spoofing by proposed system have clearly shown a shift of voice characteristics toward the target speakers.

Abstrakt

Cílem konverze hlasu (voice conversion, VC) je převést hlas zdrojového řečníka na hlas cílového řečníka. Technika je populární je u vtipných internetových videí, ale má také řadu seriózních využití, jako je dabování audiovizuálního materiálu a anonymizace hlasu (například pro ochranu svědků). Vzhledem k tomu, že může sloužit pro spoofing systémů identifikace hlasu, je také důležitým nástrojem pro vývoj detektorů spoofingu a protiopatření.

Modely VC byly dříve trénovány převážně na paralelních (tj. dva řečníci čtou stejný text) a na vysoce kvalitních audio materiálech. Cílem této práce bylo prozkoumat vývoj VC na neparalelních datech a na signálech nízké kvality, zejména z veřejně dostupné databáze VoxCeleb.

Práce vychází z moderní architektury AutoVC definované Qianem et al. Je založena na neurálních autoenkodérech, jejichž cílem je oddělit informace o obsahu a řečníkovi do samostatných nízkodimenzionálních vektorových reprezentací (embeddingů). Cílová řeč se potom získá nahrazením embeddingu zdrojového řečníka embeddingem cílového řečníka. Qianova architektura byla vylepšena pro zpracování audio nízké kvality experimentováním s různými embeddingy řečníků (d-vektory vs. x-vektory), zavedením klasifikátoru řečníka z obsahových embeddingů v adversariálním schématu trénování neuronových sítí a laděním velikosti obsahového embeddingu tak, že jsme definovali informační bottle-neck v příslušné neuronové síti. Definovali jsme také další adversariální architekturu, která porovnává původní obsahové embeddingy s embeddingy získanými ze zkonvertované řeči.

Výsledky experimentů prokazují, že neparalelní VC na nekvalitních datech je skutečně možná. Výsledná audia nebyla tak kvalitní případě "hi fi" vstupů, ale výsledky ověření řečníků po spoofingu výsledným systémem jasně ukázaly posun hlasových charakteristik směrem k cílovým řečníkům.

Keywords

voice conversion, speech processing, x-vector, d-vector, autoencoder, verification, spoofing, wavenet, neural networks

Klíčová slova

konverze hlasu, zpracování řeči, x-vektor, d-vektor, autoenkodér, verifikace, spoofing, wavenet, neuronové sítě

Reference

BRUKNER, Jan. *Non-Parallel Voice Conversion.* Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Dr. Ing. Jan Černocký

Rozšířený abstrakt

Cílem konverze hlasu (voice conversion, VC) je převést hlas zdrojového řečníka na hlas cílového řečníka. Technika je populární je u vtipných internetových videí, ale má také řadu seriózních využití, jako je dabování audiovizuálního materiálu a anonymizace hlasu (například pro ochranu svědků). Vzhledem k tomu, že může sloužit pro spoofing systémů identifikace hlasu, je také důležitým nástrojem pro vývoj detektorů spoofingu a protiopatření.

Modely VC byly dříve trénovány převážně na paralelních (tj. dva řečníci čtou stejný text) a na vysoce kvalitních audio materiálech. Cílem této práce bylo prozkoumat vývoj VC na neparalelních datech a na signálech nízké kvality, zejména z veřejně dostupné databáze VoxCeleb.

Práce nejdříve zadefinuje konverzi hlasu, rozdíl mezi paralelní a neparalelní konverzí a následne tzv. one-shot konverzí. One-shot konverze znamená, že mluvčí, na kterých je systém testován nebyli zahrnuti v trénovací sadě. Tato technika získává v posledním roce více pozornosti díky vývoji a snaze posouvat hranice v oblasti konverze hlasu.

Dále je popsána Voice Conversion Challenge (VCC). VCC je soutěž, ve které se snaží účastníci vytvořit co nejlepší systém pro konverzi hlasu. Se systémem, který vznikl z této práce jsme se jí také účastnili, právě použitím one-shot metody.

Před samotným jádrem jsou nejdříve zadefinovány vrstvy a aktivační funkce neuronových sítí, které jsou dále používány.

V hlavní části práce jsou popsány autoenkodéry jako neuronové sítě, které mohou rozdělit řeč na část reprezentující mluvčího a část reprezentující obsah, tzv. "Speaker disentanglement". Této techniky využívá více systémů a tři z nich jsou detailněji popsány: CycleVAE využívá modifikovaný autoenkodér – Variational Autoencoder. Následuje metoda, která využívá textové vstupy pro lepší zadefinování výstupu řečového enkodéru a zároveň využívá dva klasifikátory, jeden pro separaci pouze informace o mluvčím a druhý pro kontradiktní (adversarial) trénování obsahové části tak, aby v ní byl co nejvíce potlačený mluvčí.

Hlavní část práce vychází z moderní architektury AutoVC definované Qianem et al. Je založena opět na neurálních autoenkodérech. Tato metoda používá separátně vytrénovaný extraktor embeddingů mluvčího (vektor, který jej reprezentuje). Dekodér se podmiňuje tímto embeddingem, tak aby vytvořil opět nahrávku zrojového mluvčího. Díky tomu můžeme trénovat tento systém standardní objektivní funkcí na rekonstrukci původního vstupu.

Řeč cílového řečníka se potom získá nahrazením embeddingu zdrojového řečníka právě embeddingem cílovým. Qianova architektura byla vylepšena pro zpracování audio nízké kvality experimentováním s různými embeddingy řečníků (d-vektory vs. x-vektory). Dále jsme se inspirovali výše uvedenými metodami a modifikovali jsme tuto metodu zavedením klasifikátoru řečníka z obsahových embeddingů v adversariálním schématu trénování neuronových sítí a laděním velikosti obsahového embeddingu tak, že jsme definovali informační bottle-neck v příslušné neuronové síti. Definovali jsme také další adversariální architekturu, která porovnává původní obsahové embeddingy s embeddingy získanými ze zkonvertované řeči, tato metoda je zase inspirována způsobem trénování systému CycleVAE.

Důležitou součástí systémů pro konverzi hlasu je vokodér, který transformuje spektrální reprezentaci nahrávky zpět do podoby signálu. V práci jsou popsány dva vokodéry: dnes již téměř legendární WaveNet a poté Parallel WaveGAN. WaveNet byl vyvinutý v roce 2016 a získal si velkou popularitu, díky tomu, že dokáže generovat nahrávky ve velmi vysoké kvalitě. Nevýhodou WaveNetu je, že generování je časově velmi náročné. Tento problém řeší

druhý zmíněný vokodér Parallel WaveGAN, který dosahuje stejné kvality syntetizovaného hlasu, ale dokáže jej produkovat v reálném čase.

Jedním z cílů této práce je vyzkoušet konverzi hlasu na nekvalitních datech. V další části je zadefinován dataset, na kterým je systém trénován.

Výsledky experimentů prokazují, že neparalelní VC na nekvalitních datech je skutečně možná. Výsledná audia nebyla tak kvalitní případě “hi fi” vstupů, ale výsledky ověření řečníků po spoofingu výsledným systémem jasně ukázaly posun hlasových charakteristik směrem k cílovým řečníkům.

Non-Parallel Voice Conversion

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mr. Jan Černocký. The supplementary information was provided by Mr. Tomi Kinnunen. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Jan Brukner
June 10, 2020

Acknowledgements

I would like to thank to Jan "Honza" Černocký and Tomi Kinnunen from University of Eastern Finland for valuable advices and neverending motivation.

Contents

Chapter 1

Introduction

Voice conversion (VC) is one of speech processing fields closely related to speech synthesis, voice cloning or speaker identification. The goal of voice conversion is to transform speech of the source speaker to sound like it was uttered by target speaker while not altering linguistic content.

Artificial speech of desired speaker can be also generated using text to speech system trained on target speaker, but these systems are usually only trained for specific speaker and do not work on other speakers. This problem might be solved by using voice cloning techniques, but there is no control over target speakers prosody which is, in some cases, useful to preserve. Voice conversion does not have hard definition in terms of which parts of the speech to convert. For the purposes of voice conversion, we can split speech into linguistic content, timbre and prosody, where prosody means fundamental frequency and speaking rate. What is always transformed is timbre but even such a speaker specific component as fundamental frequency can be useful not to convert (for example in singing voice conversion). Similar case is with the speaking rate, which is sometimes useful to transform, but in some applications it is better to preserve source speakers rate, for example for dubbing purposes. Usually, quality of VC system is measured in terms of naturalness of resulting voice and similarity to the target speaker. But also other methods like spoofing might be used.

Voice conversion techniques can be used for spoofing speaker verification systems and to develop counter measures. Rising topic for a few last years is speaker privacy, where the goal is to hide speaker identity, voice conversion might serve as anonymization method.

1.1 Claims of this Thesis

In this thesis, I am focusing mainly on one-shot voice conversion, which means, that the system is able to convert voice from and to any desired speaker with only few seconds of audio samples. VC systems are usually developed and evaluated using clean audio data recorded in controlled environment without background noise. Second main task is to evaluate one of current VC systems on wild dataset and to examine quality degradation.

1.2 Scope of Chapters

Further in the thesis, the voice conversion field and it's variations are summarized with definition of parallel and non-parallel voice conversion, followed with description of Voice

Conversion Challenge. Next, a specific family of VC techniques, *speaker disentanglement*, is presented. Most of the VC systems transform only spectral representation of speech, therefore it is crucial to use high-quality vocoders to transform spectral representations into raw speech. Some of neural vocoders are described in the next chapter. Thesis follows with used datasets and experiments. It is concluded in the last chapter.

Chapter 2

Voice Conversion Overview

In this chapter, the difference between parallel and non-parallel VC is described, following with the types of non-parallel VC. Later, the Voice Conversion Challenge is introduced.

2.1 Parallel Voice Conversion

In parallel voice conversion, the voice conversion model (further denoted only as „model“) is trained using parallel dataset. That means that the same set of utterances from each speaker is present in the training dataset. Usually, training is done for each pair of source – target (S – T) speakers (so called one-to-one VC). Typical structure of parallel VC models is shown in figure 2.1. For parallel VC systems, some form of time alignment (e.g. Dynamic Time Warping) is typical. Time alignment allows direct mapping from source to target features and makes conversion of the prosody easier. These systems have slightly better similarity and naturalness results than the non-parallel ones, but this difference is getting less significant with current state-of-the-art models. The need of parallel dataset on the other hand is a huge disadvantage of this approach, which leaves it with very few real world applications.

2.2 Non-Parallel VC

Non-parallel VC, where parallel dataset is no longer needed, has more practical use. VC techniques using this configuration are being developed for past few years and they have achieved impressive results.

Non-parallel models have to learn mapping from one speaker to another without being able to align frames with the same content. Neural networks are almost exclusively used for this task and various models were adapted such as Generative Adversarial Networks, Autoencoders or Variational Autoencoders.

We can further split non-parallel VC systems into groups depending on their „level of generalization“:

- One-to-One: Similarly to the parallel VC, a special model is trained for each pair of speakers.
- Many-to-Many: One universal model can be used to perform conversion among various speakers from the training dataset.

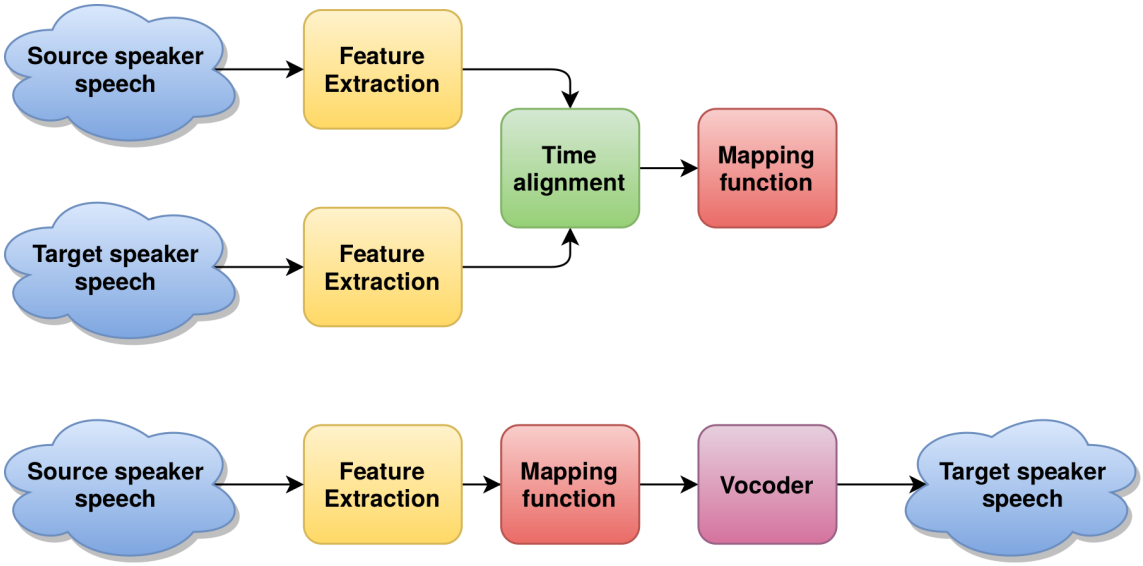


Figure 2.1: Scheme of basic parallel VC system. Training part is on the top and conversion part at the bottom. Mapping function can be Gaussian Mixture Model or Neural Network based model. Mapping function is unique for each source – target speaker pair → one-to-one conversion system.

2.3 One-Shot Voice Conversion

Above mentioned many-to-many systems consider only conversion among speakers in the training dataset. In order to step up generalization of VC systems, one-shot voice conversion was introduced. One-shot means, that as little as one utterance of the source or target speaker is required to perform transformation from one speaker to another. Some authors call this type of VC as „Zero-shot“ conversion (e.g. [14]), but I decided to follow „One-shot“ notation, because at least one utterance of each speaker is needed.

It is also worth to mention, that with this level of generalization, VC field is slowly approaching the speaker identification, where models are always tested against speakers outside of the training dataset.

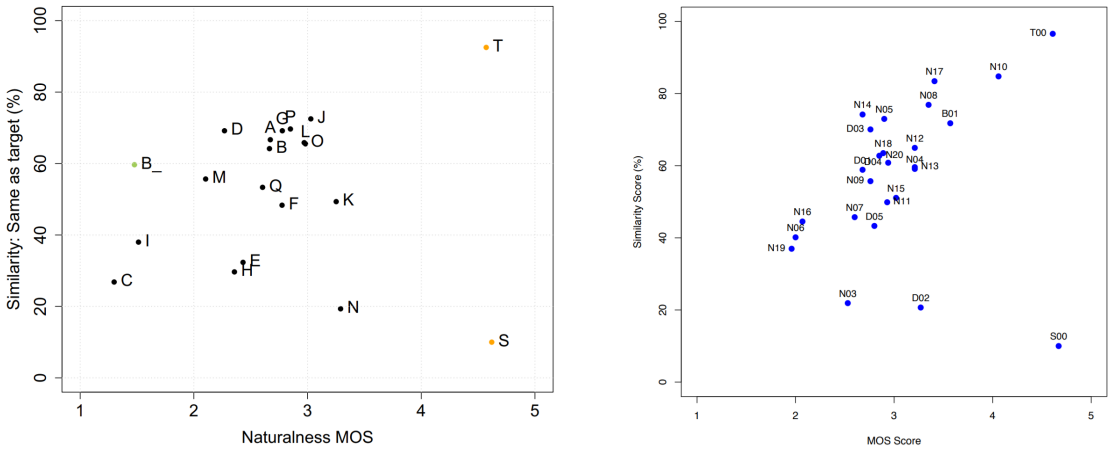
2.4 Voice Conversion Challenge

Voice Conversion Challenge (VCC) is a series of events promoting development of new VC techniques and providing comparison of submitted VC systems using identical data and evaluation methods. Two challenges were organized in years 2016 [20] and 2018 [5], with the 2020 evaluation currently in progress.

In the first challenge, there was only one task: to create VC system using parallel dataset. The second challenge consisted of two tasks: Mandatory „Hub“ task using parallel data and optional „Spoke“ with non-parallel data. This year (2020), the third VCC takes place, where no parallel conversion is done at all. It once again consists of two tasks. In the first one, the goal is to develop non-parallel VC system, but in the second one, VC settings are cross-lingual – samples of the target speakers are uttered in different language than the source speakers. Selected target languages are Finnish, German and Mandarin.

Both past challenges used two subjective metrics for evaluation (exact conditions are described in [20] or [5]):

- *Naturalness*: Subjects evaluated naturalness of converted speech samples on standard Mean Opinion Score (MOS) scale from 1 to 5. Original samples were included for reference.
- *Similarity*: Subjects were asked to compare two recordings and decide whether they were produced by the same speaker. Subjects were also supposed to ignore distortion and artefacts in converted speech.



(a) VCC16 results. S and T denote original source and target speech samples [20].

(b) VCC18 Hub task results. S00 and T00 denote original source and target speech samples [5].

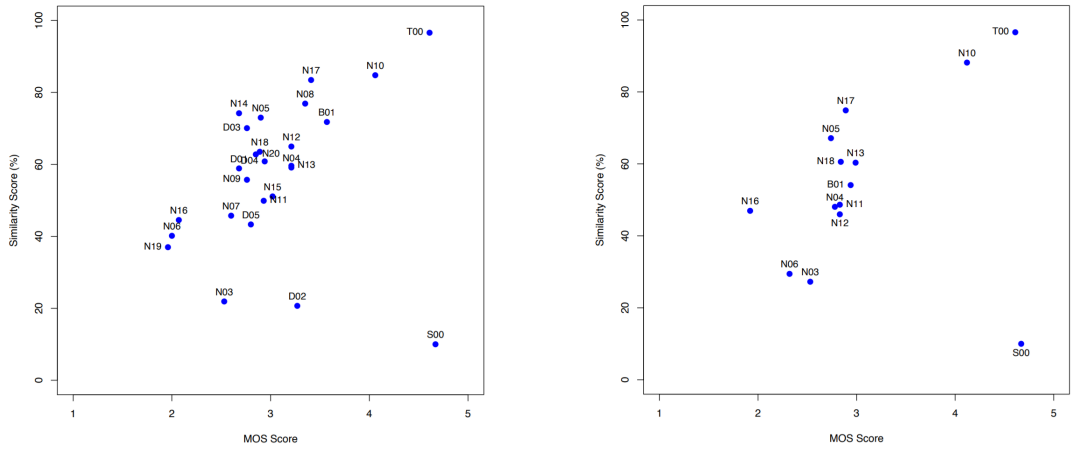
Figure 2.2: Comparison among systems submitted to VCC16 and VCC18. While the best systems from VCC16 did not pass 3.5 naturalness MOS and 80 % similarity, the best systems from VCC18 passed both values and in terms of MOS, even 4.0. Parallel dataset was used in both tasks.

There is a huge improvement between systems from year 2016 and 2018 in terms of both naturalness and similarity as seen in figure 2.2. The main difference is using neural networks as model and developing better quality vocoders such as *WaveNet* [21]. Also, from comparison of Hub and Spoke tasks in figure 2.3, we can deduce, that even though non-parallel VC is harder, it is possible to obtain the same or even better results compared to parallel VC.

2.4.1 Baseline Systems

The baseline system in the first challenge was a modified voice conversion system from FestVox toolkit[1], It used fundamental frequency and Mel-cepstrum as the input, parallel utterances were synchronized using dynamic time warping (DTW) and the mapping was done using joint Gaussian Mixture Model (GMM) trained with Expectation-Maximization (EM) algorithm. F0 was converted using global mean and variance of the target speaker. Converted Mel-cepstral coefficients and F0 were then synthetized to waveform using FestVox's vocoder.

[1] http://festvox.org/



(a) VCC18 Hub task results. S00 and T00 denote original source and target speech samples [5].

(b) VCC18 Spoke task results. S00 and T00 denote original source and target speech samples [5].

Figure 2.3: Comparison of results of Hub and Spoke task. Most systems achieved worse results in non-parallel settings except for the best system "N10", which has a slightly better results in Spoke task.

Sprocket [3] was used as baseline system for Hub task in VCC18. It can perform conversion in three modes; the first one uses similar principle as above described system from FestVox, described in detail in [19]; the second approach uses vocoder-free method for timbre conversion, where *differential GMM (DiffGMM)* trained filter is applied directly to the waveform without F0 conversion and the third one combines DiffGMM approach with F0 transformation.

The second baseline for VCC18 was Merlin toolkit [25], which is again supporting only parallel conversion and uses deep neural network (DNN) for spectral feature mapping.

There are again two baseline systems developed for VCC20. The first one uses Automatic Speech Recognition system to extract content of the speech utterance and then synthesizes target speaker's speech using text-to-speech model. It is part of ESPnet toolkit[2]. The second one is CycleVAE [18] and it is further described in section 4.2 as it is more related to the core of this work.

[2]https://github.com/espnet/espnet

Chapter 3

Neural Networks

In this chapter, layers and activation functions of used neural networks (NN) are described. Only layers, that are further referred to in the thesis are mentioned. Objective functions are described for each network separately in the text.

Notation of equations is adapted from Pytorch documentation[1] [11].

3.1 Layers

Numerous different layers are used in NNs. Some of them, that are used further in this thesis are defined in this section. In the following equations, data vector $\mathbf{x} = [x_1, \ldots, x_D]^\top$ Vector of biases $\mathbf{b}_{layer}$ is explicitly added.

- *Linear layer*, sometimes called fully connected or dense, consists only of multiplication of input data with weights and summing them together. Weight matrix has dimensions $D \times N$, where N is number of cells in layer.

$$f_L(\mathbf{x}; \mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}_L \tag{3.1}$$

$$f_L : \mathbb{R}^D \longrightarrow \mathbb{R}^N, \tag{3.2}$$

  where $\mathbf{x} = [x_1, \ldots, x_D]^\top$ is input vector, and $\mathbf{b}_L$ is D dimensional vector of biases.

- *Convolutional layer* applies convolution instead of multiplication. In speech processing, 1D convolutions along frequency domain are usually used. Kernel size is $k = (2n + 1)\,,\ n \in \mathbb{N}_0$.

$$f_C(\mathbf{x}; \mathbf{W}) = \mathbf{b}_C + \sum_{i=1}^{D} \sum_{j=-(k-1)/2}^{(k-1)/2} \mathbf{W}(\cdot, j)\, \mathbf{x}(i + j) \tag{3.3}$$

$$f_C : \mathbb{R}^D \to \mathbb{R}^N \tag{3.4}$$

  Note, that operation used in equation 3.3 is actualy a cross-correlation instead of convolution. The only difference between those two operations is kernel flip. Kernel values are learnt during training, therefore there is no difference between outputs. To avoid filtering outside of the input vector or reducing output size, (zero) padding of size $p = (k - 1)/2$ is usually used.

[1] https://pytorch.org/docs/stable/

- *Batch Normalization* is used to speed up training of the network. It normalizes features across the batch. In speech processing, 1D version is used with statistics computed per-dimension.

$$f_{batchnorm}(\mathbf{x}) = \frac{\mathbf{x} - \mathbb{E}(\mathbf{x})}{\sigma(\mathbf{x})} \times \gamma + \beta, \tag{3.5}$$

  where γ and β are learnable parameters

- *Long Short Term Memory (LSTM) layer* is a variation of the Recurrent neural layer. Recurrent neural networks (RNN) process input in sequence and they have two inputs, one regular and the second one is from the previous – hidden – step. Both inputs have they own set of weights $\mathbf{W_{i,\_}}$ and $\mathbf{W_{h,\_}}$ respectively. RNN's output at time t is computed as $h_{(t)} = \tanh\left(\mathbf{W}_{i,n}\mathbf{x}_t + b_{i,n} + \mathbf{W}_{h,n}h_{(t-1)} + b_{h,n}\right)$. $\mathbf{x}$ is again a feature vector, and subscript t is added in order to denote precisely its time. Standard RNN have problem, that past inputs loose quickly influence on the output (the network forgets). These issues are addressed by LSTM, where cell state is introduced. Cell state c_t is a hidden feature of the cell, it is passed to the next step with only residual modifications. This leads to a possibility of keeping long time context in the network. Scheme of LSTM cell is in figure[2] 3.1.

$$\begin{aligned}
i_t &= \sigma\left(\mathbf{W}_{i,i}\mathbf{x}_t + b_{i,i} + \mathbf{W}_{h,i}h_{t-1} + b_{h,i}\right)\\
f_t &= \sigma\left(\mathbf{W}_{i,f}\mathbf{x}_t + b_{i,f} + \mathbf{W}_{h,f}h_{t-1} + b_{h,f}\right)\\
g_t &= \tanh\left(\mathbf{W}_{i,g}\mathbf{x}_t + b_{i,g} + \mathbf{W}_{h,g}h_{t-1} + b_{h,g}\right)\\
o_t &= \sigma\left(\mathbf{W}_{i,o}\mathbf{x}_t + b_{i,o} + \mathbf{W}_{h,o}h_{t-1} + b_{h,o}\right)\\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t\\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}$$

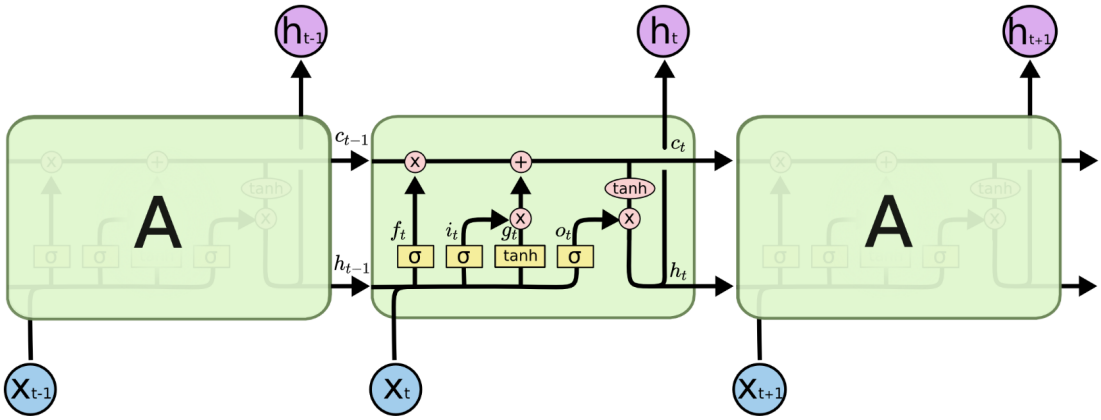


Figure 3.1: LSTM cell scheme.

[2]Picture is taken from Christopher Olah's blog https://colah.github.io/posts/2015-08-Understanding-LSTMs/

3.2 Activation Functions

Stack of only linear layers would result only in a linear transformation of the input. To break the linearity, activation function needs to be used. For example, LSTM layer has these activations built inside, but other layers, like linear of convolutional, need to use them explicitly.

Typically used activation functions are:

- Rectified Linear Unit – $\mathrm{ReLU}(x) = max(0, x)$ and its modified version $\mathrm{LeakyReLU}(x) = max(cx, x)$ where $c \in \langle 0, 1 \rangle$
- Logistic sigmoid – $\sigma(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent – $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- Softmax is usually used at the end of network together with cross-entropy loss. It rescales input vector so that it sums up to one and can thus serve as probabilities in categorical distribution:

$$\mathrm{softmax}(\mathbf{x}_i; \mathbf{x}) = \frac{\exp(\mathbf{x}_i)}{\sum_{j=1}^{D} \exp(\mathbf{x}_j)} \tag{3.6}$$

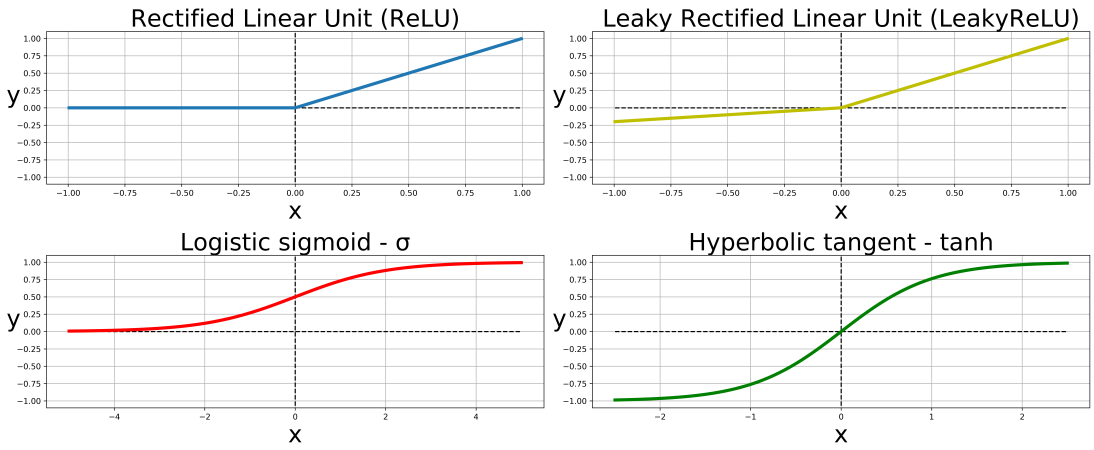


Figure 3.2: Activation functions.

Chapter 4

Speaker Disentanglement for Voice Conversion

In general, disentanglement refers to a method of separating compound object into preferably disjoint features.

In the voice conversion, *Speaker Disentanglement* is a technique of separating speech into speaker and linguistic (content) dependent representations that can be later reconstructed back to speech. To enforce this separation, some form of dimensionality reduction needs to be done on the input speech data.

In this chapter, the concept of autoencoder architecture and a way to use it for speaker disentanglement is described. Later, some VC methods, which are using autoencoders and speaker disentanglement, are described.

4.1 Autoencoder Architecture

The autoencoders are neural network architectures, that consist of two parts, *encoder* E and *decoder* D. The task of the encoder is to reduce dimensionality of the input data $\mathbf{X}$ and create information bottleneck.

$$\mathbf{z} = E(\mathbf{X}) \tag{4.1}$$

On the other hand, decoder's task is to re-create input data from the output of the encoder – often called latent features – with as small information loss as possible.

$$\hat{\mathbf{X}} = D(\mathbf{z}) \tag{4.2}$$

Typically autoencoders are trained using mean squared error loss function to achieve data reconstruction.

$$\mathcal{L}_{AE} = \mathbb{E}\left[\|\hat{\mathbf{X}} - \mathbf{X}\|_2^2\right] \tag{4.3}$$

In order to create different output or to reconstruct more precise output, decoder can be *conditioned* on some external information. This information can be varying from simple binary flag up to the outputs of the whole neural network. Both approaches are used in the voice conversion. This architecture can be modified by adding more encoders, each one can be trained to extract specific part of the input data. Typically two encoders are used in voice conversion systems. One for extracting speaker-dependent information E_S and the other one for the content E_C. Speaker-dependent information is usually in form of a embedding (one vector, that uniquely describes speaker). Input of the decoder are latent

features from E_C and it is further conditioned on speaker embedding. Equation 4.2 can be now rewritten into the following:

$$\hat{\mathbf{X}}_{\mathbf{s},\mathbf{u}} = D\left(E_C\left(\mathbf{X}_{s,u}\right), E_S\left(\mathbf{X}_s\right)\right), \tag{4.4}$$

where s and u denote speaker and utterance respectively. $\mathbf{X}_s$ is any utterance of the speaker u.

To perform voice conversion, we can simply replace speaker embedding $E_S(\mathbf{X}_s)$ with embedding of desired target speaker $E_S(\mathbf{X}_t)$.

$$\hat{\mathbf{X}}_{\mathbf{t},\mathbf{u}} = D\left(E_C\left(\mathbf{X}_{s,u}\right), E_S\left(\mathbf{X}_t\right)\right) \tag{4.5}$$

4.1.1 Speaker Encoder

Speaker Encoder can be either trained together with whole VC system or use an already pre-trained system for extracting some form of speaker embedding such as x-vectors [17] or d-vectors [23]. Often, the assumption about embedding is, that it stays constant for any utterance of the same speaker. This assumption is false as seen in the figure 4.1, but mean vector of several embeddings can be taken as a representative and used further for the VC system training.

4.1.2 Content Encoder

Content Encoder's task is to derive *speaker independent* information from the input speech. Phrase *speaker independent* is important as even though we assume, that the output is some form of text representation (it can be called „text embeddings"), there is more information „hidden" in latent features such as prosody or background noise.

There are several proposals how to force E_C to remove speaker information from the utterance, naMely:

- Create bottleneck narrow enough, that all speaker information in reconstructed speech is taken from speaker embedding [14].
- Use additional speaker classifier to classify latent features and train E_C against it [27].
- Slightly different approach is to use separately trained *automatic speech recognition* system to extract the text information in form of text embeddings

4.1.3 Decoder

The decoder creates desired speaker's spectrogram using latent features from E_C and target speaker's embedding as the only information about speaker.

It is worth to notice, that the decoder has a similar functionality as *text-to-speech* (TTS) models, which take text as the input data (vs. latent features) and are trained on a specific speaker or as *voice cloning* systems, which, again, take text as input, but also use speaker embedding to generate target speaker's voice.

Decoder's architecture is usually similar to some TTS model as they have similar purpose – to produce spectrogram. Both [14] and [27] used slightly modified architecture of Tacotron 2 [16].

Figure 4.1: t-SNE [6] visualization of speaker embeddings. Shown embeddings are x-vectors of subset of speakers from VoxCeleb dataset.

Postnet

Both systems described further in Sections 4.3 and 4.4 use *post-network* at the end of the decoder. It was proposed in Tacotron 2 [16] to improve Mel-spectrogram's fine details. Postnet's architecture is show further in table 4.1. It predicts residual of the Mel-spectrogram and adds it to the output of the decoder. Comparison of Mel-spectrograms with and without postnet can be seen in Figure 4.2.

4.1.4 Variational Autoencoders

VAEs are generative models, which use similar architecture as autoencoders. Generative model means that it can create new unseen data. In VAEs, it is done by sampling from the latent space. In order to be able to generate meaningful data, the latent space needs to be regular, which means that sampling the slightly different features result into slightly different meaning as visualised in Figure 4.3. Basic autoencoders do not guarantee this property, thus cannot be regarded as generative models.

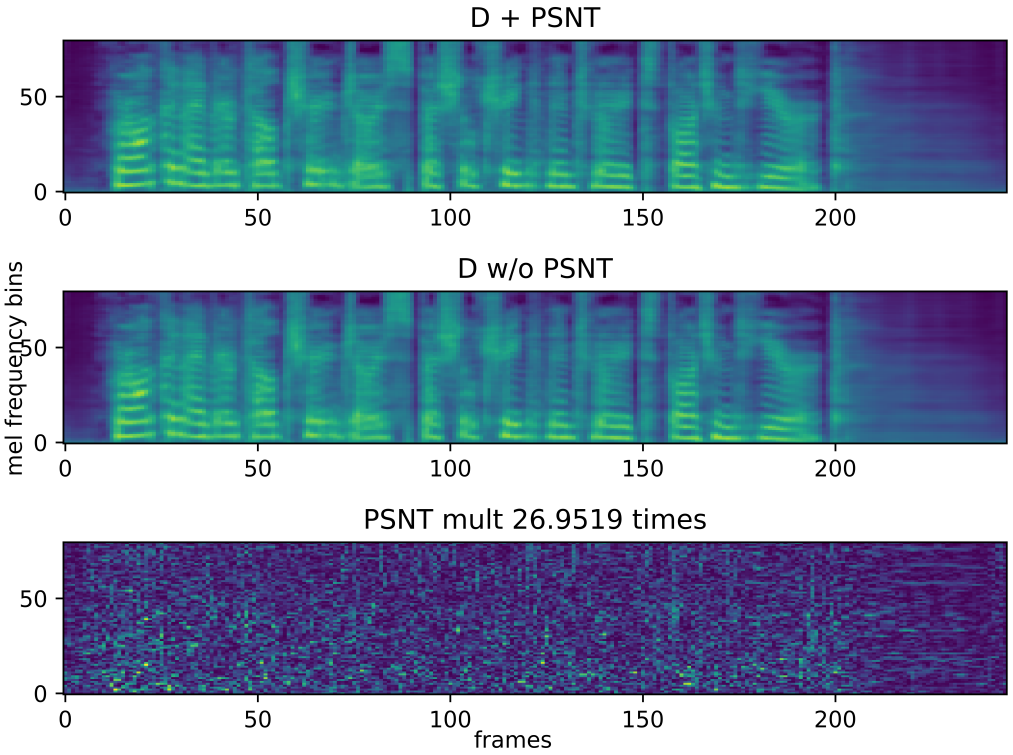


Figure 4.2: Comparison of the decoder (D) output with added postnet (PSNT) residual in the first plot, only decoder outputs without postnet output in the second plot, and finally only postnet's residual multiplied by $\frac{1}{max(\mathbf{P}(x))}$ to get values between 0 and 1.

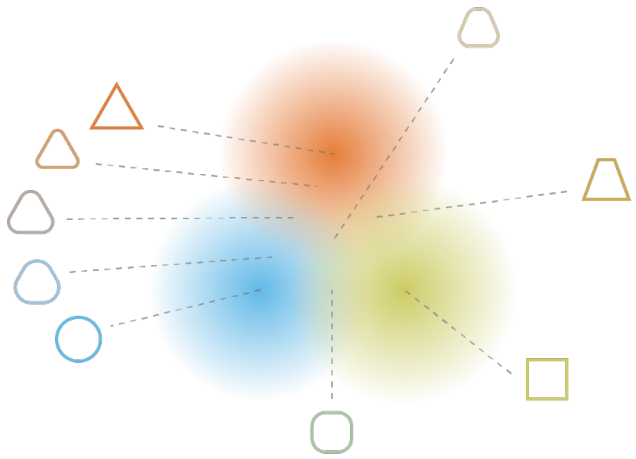


Figure 4.3: Visualization of regular latent space (underlaying circles) and its decoded representations (shapes). Picture is from [15].

VAE's encoder does not output a sample in the latent space, but a latent probability distribution instead. Latent feature is then sampled from that distribution and decoded:

$$p(\mathbf{z}|\mathbf{x}) = E(\mathbf{x}) \tag{4.6}$$

$$\mathbf{z} \sim p(\mathbf{z}|\mathbf{x}) \tag{4.7}$$

$$\hat{\mathbf{x}} = D(\mathbf{z}) \tag{4.8}$$

Furthermore, $p(\mathbf{z}|\mathbf{x})$ is normal distribution. To achieve regular latent space, KL-divergence term in loss function is introduced to push latent space towards normal distribution [15]:

$$\mathcal{L}_{VAE} = \mathbb{E}\left[\|\hat{\mathbf{X}} - \mathbf{X}\|_2^2\right] - D_{KL}\left(p(\mathbf{z}|\mathbf{x})\,||\mathcal{N}(0, \mathbf{I})\right) \tag{4.9}$$

KL-divergence

KL-divergence (short for Kullback-Leibler divergence) is a term, that measures difference between two probability distributions $p(x)$ and $q(x)$. It is defined for continuous distributions as:

$$D_{KL}(p||q) = \int_{\mathcal{X}} p(x) \log\left(\frac{p(x)}{q(x)}\right) \tag{4.10}$$

KL-divergence is not commutative operation $D_{KL}(p||q) \neq D_{KL}(q||p)$ and it has a close form solution for D_{KL} between diagonal multivariate Gaussian distribution and multivariate standard normal distribution $\mathcal{N}(0, \mathbf{I})$[1]:

$$D_{KL}\left(\mathcal{N}\left([\mu_1, \ldots, \mu_D]^T, diag(\sigma_1^2, \ldots, \sigma_D^2)\right)||\mathcal{N}(0, \mathbf{I})\right) = \frac{1}{D}\sum_{d=1}^{D}\left(\sigma_d^2 + \mu_d^2 - \ln\sigma_d^2 - 1\right) \tag{4.11}$$

4.2 CycleVAE VC

CycleVAE VC[2] [18] uses cyclic variational autoencoder to perform voice conversion, it is also one of the baseline systems in Voice Conversion Challenge 2020. It creates one-to-one VC system.

The cycle can be broken into two parts: in the first part, input features of the source speaker are converted to the target speaker and also back to themselves; in the second part converted target features are converted back to the source speaker as shown in figure 4.4.

Used features are Mel-spectrogram and fundamental frequency extracted using WORLD toolkit [8]. f_0 is transformed linearly in log-domain using mean and variance of the source and target speakers pitch:

$$f_{0_{tar}} = \frac{\sigma_{tar}}{\sigma_{src}}\left(f_{0_{src}} - \mu_{src}\right) + \mu_{tar} \tag{4.12}$$

CycleVAE creates separate model for each speaker pair, therefore there is no need to use embeddings with speaker information. We can replace it by simply using a binary flag. Authors also state, that for many-to-many conversion, one-hot vector can be used to determine desired target speaker. However, in order to perform one-shot VC, additional speaker information (and separately extracted) must be provided instead of using one-hot vector.

Loss of this network is based on the standard VAE loss, with exception, that mean absolute error is used instead of mean squared error. Loss for one cycle is then implemented

[1] https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence#Examples also used in CycleVAE's [18] implementation, https://github.com/patrickltobing/cyclevae-vc/

[2] Implementation is available on https://github.com/patrickltobing/cyclevae-vc

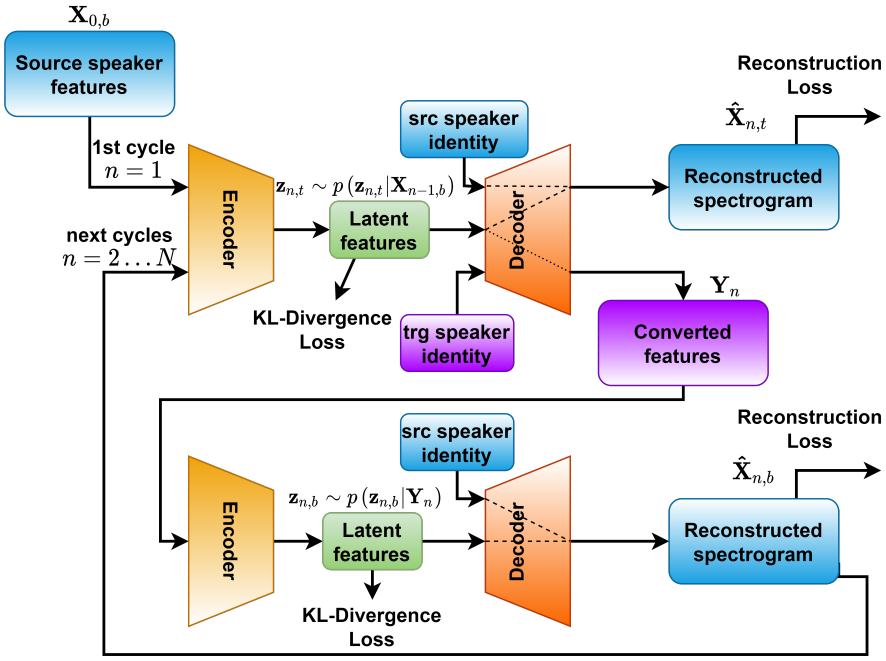


Figure 4.4: Scheme of CycleVAE VC training process. Top part of the figure is simple VAE voice conversion and whole graph shows one cycle of CycleVAE during training phase. Picture is heavily inspired by [18].

as follows:

$$\begin{aligned} \mathcal{L}_{Cycle,n} = &\mathbb{E}\left[\|\hat{\mathbf{X}}_{n,t} - \mathbf{X}_{0,b}\|\right] \\ &+\mathbb{E}\left[\|\hat{\mathbf{X}}_{n,b} - \mathbf{X}_{0,b}\|\right] \\ &-D_{KL}\left(p\left(\mathbf{z}_{n,t}|\mathbf{X}_{n-1,b}\right)||N\left(0,\mathbf{I}\right)\right) \\ &-D_{KL}\left(p\left(\mathbf{z}_{n,b}|\mathbf{Y}_{n}\right)||N\left(0,\mathbf{I}\right)\right) \end{aligned} \tag{4.13}$$

All symbols follow the notation from figure 4.4. Full loss is then constructed as sum over all N cycles:

$$\mathcal{L}_{CycleVAE} = \sum_{n=1}^{N} \mathcal{L}_{Cycle,n} \tag{4.14}$$

4.3 Non-Parallel Sequence-to-Sequence Voice Conversion

In this method[3], proposed in [27], phonetic transcriptions $\mathbf{T} = [t_1, ...t_M]$ are used together with acoustic features. Furthermore, auxiliary speaker classifier is used to enforce better disentanglement. The model is trained on both auto-encoding and text-to-speech task. Whole structure of this system is shown in figure 4.5.

Five modules are used in training or conversion process:

[3]Implemenation is available on https://github.com/jxzhanggg/nonparaSeq2seqVC_code

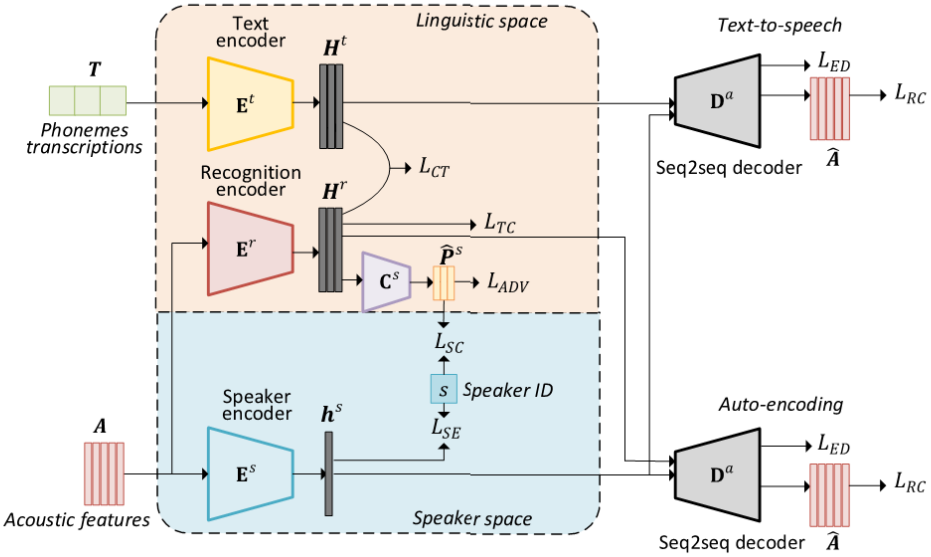


Figure 4.5: The architecture of Sequence-to-Sequence VC model. Image is taken from [27].

- Speaker encoder E_S is used for extracting speaker embedding $\mathbf{e}_s$ from acoustic features. It consists of 2 BLSTM layers with 128 output cells followed by average pooling and one fully connected layer with *tanh* activation.
- Text encoder E_T extracts vector of text embeddings $\mathbf{H}^t = \left[\mathbf{h}_1^t, ..., \mathbf{h}_n^t\right]$ from phoneme transcription of the input speech.
- Recognition encoder E_R is trained to extract the same sequence of embeddings $\mathbf{H}^r = [\mathbf{h}_1^r, ..., \mathbf{h}_n^r]$ as E_T except it takes acoustic features as the input. This is the „content encoder“ when following the terminology from above.
- Auxiliary classifier C_S predicts speaker identity from $\mathbf{H}^r$ and is used to eliminate remaining speaker dependent information in $\mathbf{H}^r$.
- Decoder D_A recovers the acoustic feature sequence from $\mathbf{H}^r$ and $\mathbf{e}_s$ for auto-encoding or $\mathbf{H}^t$ for text-to-speech task (both are used during training). Decoder predicts two frames at each time step. Architecture of the decoder is similar to Tacotron 2 [16].

Several Loss functions are used for training:

1. Loss for phoneme sequence classification compares $\mathbf{H}^r$ with original phoneme sequence using *Cross Entropy*.

$$\mathcal{L}_{TC} = \mathbb{E}\left[\mathrm{CE}(\mathbf{t}_n, \mathrm{softmax}(\mathbf{W}\mathbf{h}_n^r))\right], \tag{4.15}$$

   where $\mathbf{W}$ are trainable parameters of the softmax layer.

2. C_S module is trained again with cross entropy with $\mathbf{p}^s$ as true speaker labels and $\hat{\mathbf{p}}_n^s$ as predicted speaker probability.

$$\mathcal{L}_{CS} = \mathbb{E}\left[\mathrm{CE}\left(\mathbf{p}^s, \hat{\mathbf{p}}_n^s\right)\right] \tag{4.16}$$

3. Linguistic embedding similarity loss is introduced to increase similarity between $\mathbf{h}^r$ and $\mathbf{h}^t$ when $m = n$ and reduce similarity for $m \neq n$.

$$\mathcal{L}_{CT} = \sum_{m=1,n=1}^{N,N} \mathbb{I}_{mn} d_{mn} + (1 - \mathbb{I}_{mn}) \max(d_{mn}, 0) \tag{4.17}$$

$$d_{mn} = \left\| \frac{\mathbf{h}_m^r}{\|\mathbf{h}_m^r\|_2} - \frac{\mathbf{h}_n^t}{\|\mathbf{h}_n^t\|_2} \right\|_2^2, \tag{4.18}$$

where $\mathbb{I}$ is indicator matrix and $\mathbb{I}_{mn} = 1$ when $m = n$ and 0 otherwise. d_{mn} is distance between $\mathbf{h}_m^r$ and $\mathbf{h}_n^t$.

4. Adversarial loss is used for recognition encoder training. It compares C_S's predicted probabilities $\hat{\mathbf{p}}_n^s$ with uniform distribution $\mathbf{e}_n = \frac{1}{N_{spkrs}}$. This loss function forces recognition encoder to produce text embeddings with no information about speaker.

$$\mathcal{L}_{ADV} = \mathbb{E}\left[\|\mathbf{e} - \hat{\mathbf{p}}_n^s\|_2^2\right] \tag{4.19}$$

5. Speaker encoder is trained using cross entropy loss.

$$\mathcal{L}_{SE} = \mathrm{CE}\left(\mathbf{p}^{\mathbf{s}}, \mathrm{softmax}\left(\mathbf{V}\mathbf{h}^{\mathbf{s}}\right)\right), \tag{4.20}$$

where $\mathbf{V}$ are trainable parameters of the softmax layer.

6. Decoder is trained on reconstruction loss. It is using absolute error instead of squared error.

$$\mathcal{L}_{RC} = \mathbb{E}\left[\left\|\hat{\mathbf{X}} - \mathbf{X}\right\|_1\right] \tag{4.21}$$

7. To end generation of the speech sample during conversion, hidden state of LSTM layer in decoder is projected into scalar value using a linear layer and sigmoid activation $\hat{f}_{end}$. Cross-entropy is used to guess the correct ending frame.

$$\mathcal{L}_{ED} = CE\left(f_{end}, \hat{f}_{end}\right) \tag{4.22}$$

Text-to-speech and self-reconstruction is alternating iteration by iteration.

Training of this model consists of two stages. During pre-training stage large amount of speakers is present in training data. In the fine-tuning stage, specific pair of source-target speaker is trained. Speaker encoder E_S is also discarded in fine-tuning stage and replaced with averaged embedding for each speaker.

Resulting Mel-spectrogram is transformed into waveform using WaveNet vocoder[21].

The model is able to perform only one-to-one VC under these conditions, however, authors claim that many-to-many VC is possible by simply using more speakers in pre-training stage.

4.4 AutoVC

AutoVC[4] [14] is one of the many-to-many VC systems. It uses regular autoencoder and conditions decoder on the speaker embeddings. This model is also one of the first VC systems, that is capable of one-shot conversion (called zero-shot in the paper).

[4]Implemenation is available on https://github.com/auspicious3000/autovc

AutoVC achieves disentanglement by designing bottleneck narrow enough to exclude speaker information but wide enough to preserve the content. All this is done without any adversarial training.

Scheme of the full network is shown in figure 4.6 and its architecture in table 4.1.

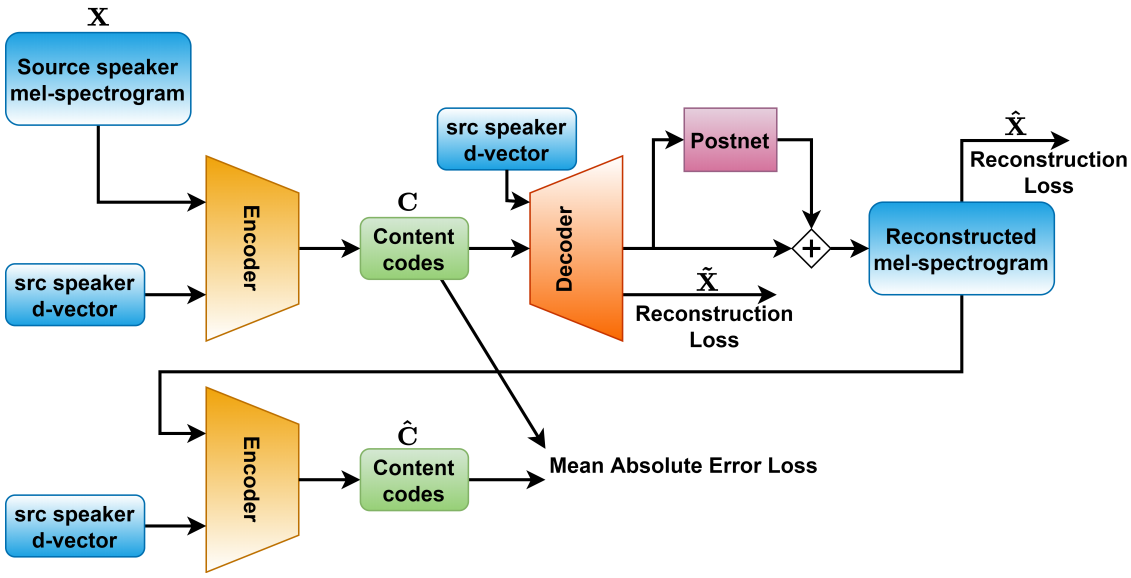


Figure 4.6: Scheme of AutoVC autoencoder.

4.4.1 AutoVC Structure

AutoVC autoencoder has the following structure.

Content encoder E_C extracts content codes ($\mathbf{C}$) from acoustic features. Codes do not contain only linguistic but all speaker independent information. Input of the E_C is Mel-spectrogram concatenated with separately extracted speaker embedding of the source speaker. Supplied embedding should allow E_C to learn faster, what information to discard. Output is vector of content embeddings reduced in both time and channel dimensions. Let $\mathbf{c}_{\rightarrow}(t)$ and $\mathbf{c}_{\leftarrow}(t)$ be BLSTM outputs in forward and backward way respectively at time frame t. Both are d_{neck} dimensional vectors. Downsampling rate is d_f. Final content codes are extracted as:

$$\mathbf{C} = \left[\begin{pmatrix} \mathbf{c}_{\rightarrow}(d_f - 1) \\ \mathbf{c}_{\leftarrow}(0) \end{pmatrix}, \begin{pmatrix} \mathbf{c}_{\rightarrow}((i+1)\, d_f - 1) \\ \mathbf{c}_{\leftarrow}(i d_f) \end{pmatrix}, \ldots, \begin{pmatrix} \mathbf{c}_{\rightarrow}\left(\left(\frac{T}{d_f} + 1\right) d_f - 1\right) \\ \mathbf{c}_{\leftarrow}\left(\frac{T}{d_f} d_f\right) \end{pmatrix} \right] \tag{4.23}$$

Decoder D recreates Mel-spectrogram from content encoder and supplied speaker embedding of the target speaker concatenated together. As mentioned in section 4.1.3, architecture is inspired by Tacotron 2 [16] decoder model. First, it upsamples content codes by *copying* them into the same temporal resolution as was the original Mel-spectrogram, then target speaker embedding is concatenated to each frame (original speaker during training). To further improve reconstruction, postnet is used and its outputs are added to the decoder output.

WaveNet conditioned on Mel-spectrograms is used for waveform generation.

Table 4.1: Architecture of the AutoVC's encoder.

Content Encoder

	Layer	Misc	Output
	Input	-	$(80 + 256) \times T$
$3 \times$	*Conv1D*	kernel = 5	$512 \times T$
	BatchNorm1D	-	-
	ReLU	-	-
$2\times$	*BLSTM*	-	$d_{neck} \times T$
	Downsample	-	$2d_{neck} \times \frac{T}{d_f}$

Decoder

	Layer	Misc	Output
	Input	-	$(2d_{neck} + 256) \times \frac{T}{d_f}$
	Upsample	copying	$(2d_{neck} + 256) \times T$
	LSTM	-	$512 \times T$
$3 \times$	*Conv1D*	kernel = 5	$512 \times T$
	BatchNorm1D	-	-
	ReLU	-	-
$2\times$	*LSTM*	-	$1024 \times T$
	Linear	-	$80 \times T$

Postnet

	Layer	Misc	Output
	Input	-	$80 \times T$
$4 \times$	*Conv1D*	kernel = 5	$512 \times T$
	BatchNorm1D	-	-
	tanh	-	-
	Conv1D	kernel = 5	$80 \times T$

Table 4.2: Architecture of the Speaker Encoder.

AutoVC's Speaker Encoder (d-vector system)

	Layer	Misc	Output
	Input	-	$80 \times T$
$3\times$	*LSTM*	only last output in time domain is considered	768×1
	Linear projection	-	256
	L_2 normalization	-	256

4.4.2 Speaker Encoder

Speaker encoder E_S in AutoVC system is network extracting deep neural embeddings called d-vectors [23]. Architecture of the E_S is similar to the previous method, except that for AutoVC, the speaker encoder is trained separately.

Loss function used for training is Generalized end-to-end loss function [23]. This loss was designed specially for embedding extraction and it pulls embeddings from the same speaker towards their centroid and pushes them further away from centroids of all other speakers. This is consistent with assumption that embeddings are constant throughout various utterances of the same speaker.

Centroid is computed as mean of embeddings $[\mathbf{e}_{k1}, \dots \mathbf{e}_{kM}]$ from one speaker k :

$$\mathbf{c}_k = \frac{1}{M} \sum_{m=1}^{M} \mathbf{e}_{k,m} \tag{4.24}$$

Training batch is constructed containing N different speakers and each speaker has M utterances. Then similarity matrix is created as scaled cosine distance (equation 6.2) between each utterance and each centroid.

$$S_{ji,k} = w \cos\left(\mathbf{e}_{ji}, \mathbf{c}_k\right) + b, \tag{4.25}$$

where j is index of speaker and i is index of the utterance. To compute centroid for similarity matrix when $j = k$, embedding $\mathbf{e_{k,m=i}}$ is excluded to increase stability during training.

$$\mathbf{c}_k^{(-i)} = \frac{1}{M} \sum_{m=1, m\neq i}^{M} \mathbf{e}_{k,m} \tag{4.26}$$

Similarity matrix is then designed as follows:

$$S_{ji,k} = \begin{cases} w \cos\left(\mathbf{e}_{ji}, \mathbf{c}_k^{(-i)}\right) + b & k = j \\ w \cos\left(\mathbf{e}_{ji}, \mathbf{c}_k\right) + b & k \neq j \end{cases} \tag{4.27}$$

Final loss function for speaker encoder is then defined as:

$$\mathcal{L}(\mathbf{e}_{ji}) = -S_{ji,j} + \log \sum_{k=1}^{N} \exp(S_{ji,k}) \tag{4.28}$$

4.4.3 Training

AutoVC's autoencoder is trained only on self-reconstruction error and content code reconstruction error in an almost unsupervised way (speaker encoder was still trained with supervision).

$$\mathcal{L}_{PSNT} = \mathbb{E}\left[\|\hat{\mathbf{X}} - \mathbf{X}\|_2^2\right], \tag{4.29}$$

$$\mathcal{L}_{D} = \mathbb{E}\left[\|\tilde{\mathbf{X}} - \mathbf{X}\|_2^2\right], \tag{4.30}$$

$$\mathcal{L}_{CD} = \mathbb{E}\left[\|E_S\left(\hat{\mathbf{X}}\right) - \mathbf{C}\|_1\right], \tag{4.31}$$

$$\mathcal{L} = \mathcal{L}_{PSNT} + \mathcal{L}_D + \mathcal{L}_E, \tag{4.32}$$

where $\tilde{\mathbf{X}}$ is reconstructed Mel-spectrogram *before* postnet outputs are added.

Chapter 5

Vocoders in Voice Conversion

VC systems usually produce only converted (Mel-)spectrogram which can be used directly in some cases for example as augmented data for training of some other network, but usually it needs to be further processed in order to obtain waveform.

In traditional VC systems, high-quality parametric vocoders such as STRAIGHT [1] or WORLD [8] have been used for feature extraction and synthesis. Some systems are vocoder-free like sprocket [3] (designs only filter applied directly on waveform). Modern VC systems usually use neural vocoders such as WaveNet or Parallel WaveGAN, which is also used in VCC20 baselines.

5.1 WaveNet

WaveNet [21] is one of the state-of-the-art neural vocoders which produces speech with quality almost similar to the recorded one.

WaveNet models joint probability density function of the signal samples as product of conditional probabilities of previous samples.

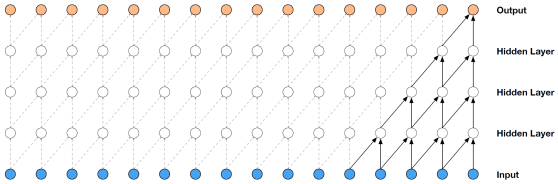
$$p\left(\mathbf{x}\right) = \prod_{t=1}^{T} p\left(x_t | x_1, x_2, \ldots, x_{t-1}\right) \tag{5.1}$$

To consider only past frames and not future ones, causal convolution layers are used (figure 5.1a). Causal convolution works similarly to the regular convolution, except it computes output only from previous samples, i. e. this is the convolution, that is used when processing online signal, seen in figure. To generate speech, large temporal context is needed. That means, that large number of convolutional layers would be needed. To increase temporal context, dilated causal convolution layers are used (figure 5.1b). Dilated convolution achieves larger perception field while using the same kernel size k by skipping some inputs with dilation factor d.

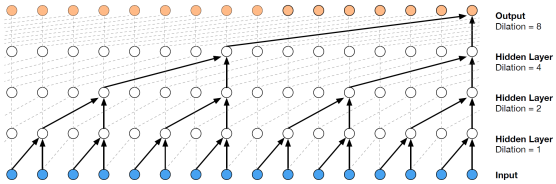
$$f\left(\mathbf{x}; \mathbf{W}\right) = \mathbf{b} + \sum_{i=1}^{D} \sum_{j=0}^{k \cdot d - 1} \mathbf{W}\left(\cdot, k + j\right) \mathbf{x}\left(i + dj\right) \tag{5.2}$$

In WaveNet, dilation is doubled for each layer up to the limit and then reset to 1.

When considering speech to be stored in 16 bit integer for each sample, the last softmax layer, which predicts the next timestep, would need to compute $2^{16} = 65536$ probabilities.



(a) Visualization of causal convolutional layers.



(b) Visualization of dilated causal convolutional layers.

Figure 5.1: Comparison between causal convolution layers and dilated causal convolution layers. Both stacks use 5 layers with kernel size 2. The regular one has temporal context 5 and the dilated one 16, while using the same number of parameters. Advantage of larger context is preserved with regular convolutional layers. Both pictures are from [21].

To make it easier to train, the speech is quantized using μ-law:

$$f_\mu(x) = sgn(x)\frac{\ln(1+\mu|x|)}{\ln 1+\mu} \tag{5.3}$$

Where sgn is a sign function, $\mu = 255$ and $-1 < x < 1$.

WaveNet is constructed with residual blocks built in stacks. Visualization of residual block is in figure 5.2. In each block, input is passed through dilated convolution layer.

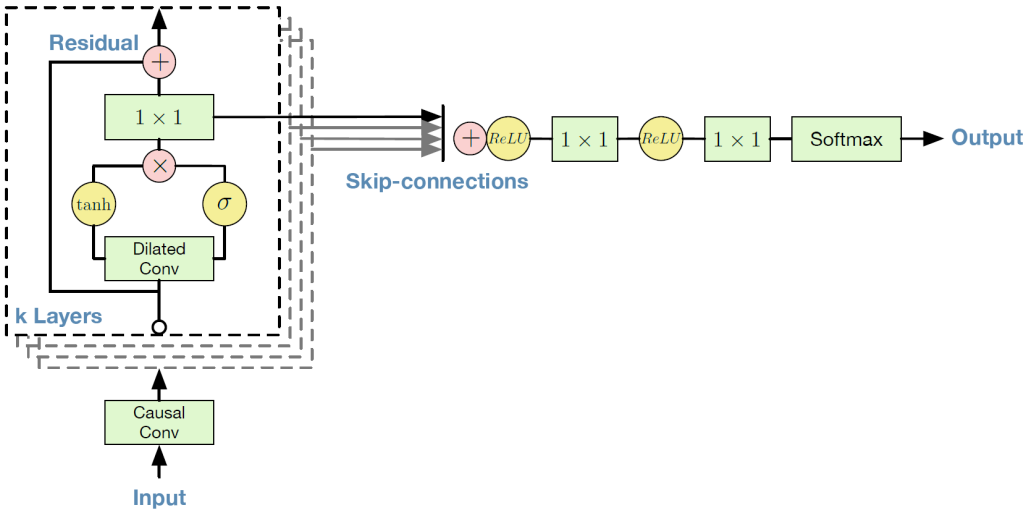


Figure 5.2: Structure of residual block in WaveNet. Picture is from [21].

Then, a non-linearity is applied:

$$\mathbf{z} = \tanh(\mathbf{W}_k\mathbf{x}) \odot \sigma(\mathbf{W}_k\mathbf{x}), \tag{5.4}$$

with $\odot$ as element-wise multiplication and $\mathbf{W}$ as weights of dilated convolution layer in k-th residual block. After non-linearity, 1x1 convolution is applied and output is either added to the input of the block. Output of the residual block serves as one of the inputs for the next block in stack. Each stack processes the whole signal.

Input is passed through causal convolution layer before being fed to the residual blocks.

Output of each stack is also (before adding the input) saved as *skip-connection*. Once all residual blocks are processed, skip-connections are summed together and then ReLU,

1x1 convolution, ReLU and one more 1x1 convolution are applied. Finally, softmax layer predicts correct quantized sample.

5.1.1 Conditional WaveNet

WaveNet, in general, is a generative model, i.e. it creates new samples from previous ones. To produce more customized outputs, WaveNet can be conditioned globally or locally on additional input. Local conditioning on Mel-spectrogram is used, when using WaveNet as a vocoder.

Input Mel-spectrogram needs to be first upsampled to match output signal in time domain. To do that, interpolation and 2d convolution are stacked into layers. In each stack, Mel-spectrogram is stretched by certain scale and product of the scales must match hop-size of Discrete Fourier transform used for spectrogram extraction.

Upsampled Mel-spectrogram $\mathbf{s}$ is then passed through 1d convolutional layer and summed together with outputs of dilated convolutional layer in residual block:

$$\mathbf{z} = \tanh\left(\mathbf{W}_k\mathbf{x} + \mathbf{V}_k\mathbf{s}\right) \odot \sigma\left(\mathbf{W}_k\mathbf{x} + \mathbf{V}_k\mathbf{s}\right), \tag{5.5}$$

where $\mathbf{V}_k$ are weights of convolutional layer for the conditioning input. When using local conditioning, input to the WaveNet can be just vector of zeros. Speech waveform is then created from Mel-spectrogram and speech samples generated in previous steps.

5.2 Parallel WaveGAN

Parallel WaveGAN uses Generative Adversarial Network (GAN) scheme together with multi-resolution short-time Fourier transform (STFT) auxiliary loss [26].

Scheme of Parallel WaveGAN training process and its architecture is shown in figure 5.3

GANs are generative models, which use two separate neural networks. Generator G produces samples $\hat{\mathbf{x}}$ with the target to deceive a Discriminator D. The discriminator is trained to classify ground-truth samples $\mathbf{x}$ as *real* and the samples from generator as *fake*:

$$\mathcal{L}_{D(G,D)} = \mathbb{E}_{\mathbf{x}\sim p_{data}}\left[\left(1 - D\left(\mathbf{x}\right)\right)^2\right] + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}\left[D\left(G\left(\mathbf{z}\right)\right)^2\right] \tag{5.6}$$

Adversarial loss function for the generator is following:

$$\mathcal{L}_{adv}\left(G, D\right) = \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,I)}\left[\left(1 - D\left(G\left(z\right)\right)\right)^2\right] \tag{5.7}$$

Note the power of two on the subtraction term, standard GANs do not have it, PWG uses least squares GAN modification proposed in [7].

Auxiliary (STFT) loss improves the stability and efficiency of the adversarial training process.

$$\mathcal{L}_s\left(G\right) = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}),\mathbf{x}\sim p_{data}}\left[\mathcal{L}_{sc}\left(\mathbf{x}, \hat{\mathbf{x}}\right) + \mathcal{L}_{mag}\left(\mathbf{x}, \hat{\mathbf{x}}\right)\right], \tag{5.8}$$

$$\mathcal{L}_{sc}\left(\mathbf{x}, \hat{\mathbf{x}}\right) = \frac{\| \left|\mathrm{STFT}(\mathbf{x})\right| - \left|\mathrm{STFT}(\hat{\mathbf{x}})\right| \|_F}{\| \left|\mathrm{STFT}(\mathbf{x})\right| \|_F}, \tag{5.9}$$

$$\mathcal{L}_{mag}\left(\mathbf{x}, \hat{\mathbf{x}}\right) = \frac{1}{N}\| \log\left|\mathrm{STFT}\left(\mathbf{x}\right)\right| - \log\left|\mathrm{STFT}\left(\hat{\mathbf{x}}\right)\right| \|_1, \tag{5.10}$$

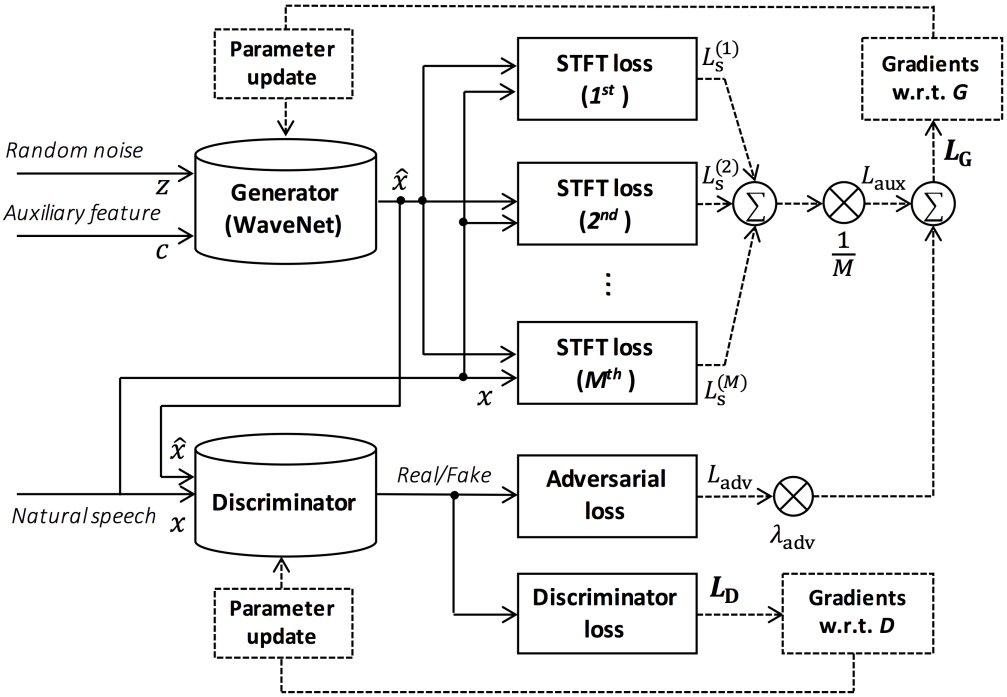


Figure 5.3: Scheme of ParallelWaveGAN. Image is taken from [26]

where $\|\cdot\|_F$ and $\|\cdot\|_1$ denote Frobenius and $L1$ norms, respectively.

$$\|A\|_F = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N}|a_{i,j}|^2} \tag{5.11}$$

$$\|A\|_1 = \sum_{i=1}^{M}\sum_{j=1}^{N}|a_{i,j}| \tag{5.12}$$

$|\mathrm{STFT}(\cdot)|$ and N denote the STFT magnitudes and number of elements in the magnitude, respectively. Multiple STFT configurations (window size, number of FFT point and frame shift) are used in training, in order not to get over-fitted to only one STFT setting and to better learn time-frequency characteristics of speech (short window gives better time resolution and wide window more precise frequencies). Complete auxiliary loss is defined as:

$$\mathcal{L}_{aux}(G) = \frac{1}{M}\sum_{m=1}^{M}\mathcal{L}_s^{(m)}(G), \tag{5.13}$$

where M is number of STFT configurations. Complete loss function is then linear combination of multi-resolution STFT loss and adversarial loss:

$$\mathcal{L}_G(G,D) = \mathcal{L}_{aux}(G) + \lambda_{adv}\mathcal{L}_{adv}(G,D), \tag{5.14}$$

with λ_{adv} as balancing parameter between two losses.

Generators architecture is similar to the WaveNet with some modifications. It does not use causal convolutions but regular ones. Input to the WaveNet is not a vector of zeros but noise sampled from Gaussian distribution.

Discriminator consists of ten 1-D convolution layers with leaky ReLU activation function. Input features for training are normalized Mel-spectrograms. More implementation details are in [26].

Parallel WaveGAN can generate speech with similar quality to the WaveNet, but it's trained faster[1] and its inference time is faster than real-time.

[1] Around 3 days on Nvidia TITAN V according to freely available implementation https://github.com/k2kobayashi/ParallelWaveGAN.

Chapter 6

Data and Metrics

Datasets used for training VC systems are described in this chapter. Later, metrics used in VC are introduced.

6.1 Datasets

It was difficult to obtain data for Voice conversion in the past due to need of parallel set of utterances. Available free datasets are for example MOCHA-TIMIT [24] or Device and Produced Speech (DAPS) [9]. Subset of DAPS dataset was also used for training and evaluation in VCC16 and VCC18 (see section 2.4).

Non-parallel systems are usually using clean speech datasets, such as VCTK [22]. Other popular dataset for training and testing VC systems is VCC18's Spoke (non-parallel) task dataset. Using this dataset is handy for direct comparison with the systems from the challenge, however, subjective evaluations are depending on the testers and on specific design of the test, e.g. how many samples are rated by each evaluator or what is their order.

6.1.1 VCTK

The VCTK [22] dataset was designed for the Voice Cloning Toolkit created by The Centre for Speech Technology Research at the University of Edinburgh. It consists of 109 speakers and each speaker reads about 400 sentences and text transcriptions are provided. All utterances are recorded in hemi-anechoic chamber with identical settings. Sampling frequency is 48 kHz with 16 bits. Each speaker reads different set of sentences from the newspapers and each speaker reads elicitation paragraph[1] and rainbow passage[2], which aim to identify speaker's accent.

6.1.2 VoxCeleb dataset

VoxCeleb [10] consists of audio samples downloaded from YouTube videos. Two versions are available VoxCeleb1 and VoxCeleb2 with over 1000 speakers in the first one and almost 6000 in the second one. There is not that strong emphasis on the number of speakers for training of VC systems. However, independent speaker encoder only benefits from larger number of speakers in training dataset.

[1] http://accent.gmu.edu/howto.php
[2] https://www.dialectsarchive.com/the-rainbow-passage

VoxCeleb was used only to obtain easily reproducible set of utterances. 30 speakers with the most speech data from VoxCeleb2 test dataset were picked. These utterances were processed with Phonexia's speech quality estimator `sqestim`[3], which computes signal-to-noise ratio in dB based comparison of gamma vs. Gaussian distribution. Only utterances with SNR higher than 15 *dB* were picked and 7 speakers were completely discarded. This results into 23 speakers with number of utterances varying from 135 to 421 with mean 244 and median 228. This sub-dataset is further denoted as *VoxCeleb23*.

6.2 Metrics

There is no standardized evaluation process for VC systems like in other fields of speech processing. Generally, metrics can be divided into two: subjective and objective.

6.2.1 Subjective Metrics

Subjective metrics are using human listeners. Two properties of VC systems are usually tested: naturalness of synthesized voice and similarity to the target speaker. The most popular naturalness metric is MOS where testing subject rate each utterance on scale from 1 to 5. Original samples need to be included for reference.

To measure similarity, subjects rate pairs of utterances, whether they are from different or same speaker. It is difficult to design this test properly: converted and original samples need to be randomized, also human listeners cannot process large number of utterances, therefore it is expensive to perform large scale subjective testing.

6.2.2 Objective Metrics

One of frequently used objective evaluation metric is Mel Cepstral Distortion, introduced for VC in [19]. This metric compares D dimensional converted Mel-cepstral coefficients $mc^{(\hat{x})}$ with those of the target speaker $mc^{(x)}$:

$$MC_{CD} = \frac{10}{\ln 10}\sqrt{2\sum_{d=1}^{D}\left(mc_d^{(\hat{x})} - mc_d^{(x)}\right)^2} \tag{6.1}$$

This metric can be used only on parallel utterances, which need to be synchronized at first (e.g. DTW), therefore it is not suitable for purposes of this thesis.

6.2.3 Testing VC with Speaker Verification and Spoofing

Other way to measure effectiveness of conversion is from spoofing perspective. First, standard speaker verification test is conducted with two types of *trials*. Target trial contains two different utterances of the same speaker and non-target trial uses different speakers. Finally, to measure quality of verification system, equal error rate (EER) is computed (error rate with the same probability of miss and false alarm). Then one utterance of each non-target trial is converted to the speaker of the second one and the same test is performed.

To test anonymization properties of VC system, the same test can be performed, but instead of converting utterance from non-target trial, one utterance from target trial is converted to a different speaker.

[3] https://www.phonexia.com/en/product/speech-quality-estimation

Table 6.1: Comparison of equal error rate on VoxCeleb 1 test dataset.

Method	**EER** [%]
cosine similarity	9.7
cosine similarity + LDA	3.9
PLDA	3.1

In both approaches, the expected outcome is increased EER.

Further in thesis, speaker verification/spoofing tests are used with trial list adopted from VoxCeleb1. To compare two utterances, DNN embedding x-vector (described in section 7.2.2) is computed from each one. Then, linear discriminant analysis (LDA) is used to reduce dimensionality of x-vectors from 512 to 100 and cosine similarity is used as distance metric. Cosine similarity between two vectors is computed as dot product divided by length of each vector:

$$\cos(\mathbf{x}_1, \mathbf{x}_2) = \cos(\Phi) = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\|_2 \|\mathbf{x}_2\|_2} \tag{6.2}$$

Probabilistic linear discriminant analysis (PLDA) might be used to further improve EER, but described method is sufficient for the task. X-vector system is trained on VoxCeleb 1 and 2 as well as PLDA[4]. LDA was trained on VoxCeleb 1 train set. Table 6.1 shows, that LDA and PLDA results are comparable and the main focus is on comparing results..

[4] https://kaldi-asr.org/models/m7

Chapter 7

Experiments

The main task of this thesis is to examine the effectiveness of current Voice Conversion systems on datasets, which do not consist of studio quality samples. For this taskm VoxCeleb dataset was selected.

Later in this chapter some changes to AutoVC system (section 4.4), which should improve its conversion quality on VoxCeleb dataset and overall are proposed.

7.1 Feature extraction

Used acoustic features are Mel-spectrograms. Mel-spectrogram in general is created by applying Discrete Fourier Transform (DFT) to speech frames and converting the result to Mel-scale.

Used feature extraction is from AutoVC's implementation.

1. Waveform is loaded with sampling frequency 16 kHz.
2. Mean of the raw signal is subtracted.
3. Raw signal is filtered with a highpass filter.
4. Filtered signal is augmented with random noise:

$$y = 0.96y + \text{rand}\,(-0.5, 0.5) \cdot 10^{-6} \tag{7.1}$$

5. Fast Fourier transform is applied on windows 64 ms long with 16 ms shift. Hann window function is used. Then absolute value from the spectrogram is taken.
6. Finally, spectrogram is multiplied by 80 Mel-bases and resulting Mel-spectrogram is converted to $[dB]$.

7.2 Embeddings

Crucial component of AutoVC is speaker encoder and produced speaker embeddings. The most important property of embeddings is to be utterance-invariant and to separate speakers.

7.2.1 D-vector embeddings

Available AutoVC's speaker encoder (d-vector system), described in section 4.4.2, was trained on combination of VoxCeleb 1 and LibriSpeech[1] datasets. It turns out, that d-vectors lost their separating properties when applied on VoxCeleb23 dataset, as seen in figure 7.1.

Figure 7.1: t-SNE vizualization of d-vector embeddings extracted from VoxCeleb dataset. Even though there are some visible clusters, most speakers cannot be properly separated from the rest. A boundary is visible in the middle, which separates female (left) and male (right) speakers.

7.2.2 X-vector embeddings

X-vectors defined by Snyder et al. in [17] are one of the deep neural embeddings that are widely used for *speaker identification* (SID) task.

The architecture of the x-vector network is in table 7.1. The network consists of five time delay neural layers. Time delay layer is a linear layer, which considers also inputs from time context. For input sequence of features $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]^\top$ and context $\{t-2, t, t+2\}$, feature vectors are concatenated into: $[\mathbf{x}_{t-2}, \mathbf{x}_t, \mathbf{x}_{t+2}]^\top$. From this point, it is a standard linear layer. Statistics pooling layer computes mean and standard deviation of *frame5* layer for each dimension across whole time domain (2 statistics for each of 1500 dimensions result into 3000 output dimensions).

[1] http://www.openslr.org/12

Table 7.1: Architecture of the x-vector system. Square brackets mean, that all frames in interval are used, while compound brackets mean, that only frames on specified indices were used. ReLU activations are used after each layer. Table is adapted from [17].

X-vector system

Layer	Layer Context	Total Context	Input x Output
frame1	$[t-2, t, t+2]$	5	120×512
frame2	$\{t-2, t, t+2\}$	9	1536×512
frame3	$\{t-3, t, t+3\}$	15	1536×512
frame4	t	15	512×512
frame5	t	15	512×1500
stats pooling	$[0, T)$	T	$1500T \times 3000$
segment6	$\{0\}$	T	3000×512
segment7	$\{0\}$	T	512×512
softmax	$\{0\}$	T	$512 \times N$

X-vector network is trained using multiclass entropy loss function:

$$\mathcal{L} = -\sum_{n=1}^{N}\sum_{k=1}^{K} \mathbb{I}_{nk} ln\left(P\left(spkr_k|\mathbf{x}_{1:T}^{(n)}\right)\right), \tag{7.2}$$

where $\mathbb{I}$ is indicator matrix with ones when segment n is uttered by speaker k and with $P\left(spkr_k|\mathbf{x}_{1:T}^{(n)}\right)$ as output of the *softmax* layer.

Embeddings of size 512 are extracted from the *segment6* before non-linearity is applied.

X-vector system is also implemented in open-source Kaldi speech recognition toolkit [12], where example model trained on VoxCeleb 1 & 2 dataset is available.

X-vectors use Mel-Frequency Cepstral Coefficients as features. Used configuration is from Kaldi VoxCeleb v2 recipe[2].

Speaker embeddings extracted using this model nicely separate speakers in the dataset as seen in figure 7.2, therefore they might be used with AutoVC system. Question still remains whether using GE2E loss function (loss used for d-vector system, section 4.4.2) would improve x-vectors separability properties or not.

X-vectors with AutoVC

As the authors of AutoVC claim, any speaker embeddings, that are sufficiently invariant can be used for the decoder conditioning. From above, x-vectors should be good candidate to replace d-vectors without any significant changes.

The last step in the speaker encoder network in AutoVC was L_2 normalization, this needs to be performed also on the x-vectors, otherwise training doesn't converge and $\mathcal{L}_E$ tends to 0 after a few thousand iterations, while other parts of the loss stays the same. Besides normalization, there is one more structural difference between d-vectors and x-vectors: their dimensionality. I experimented with dimensionality reduction using Linear Discriminant Analysis (LDA) and taking only first 256 dimensions from x-vector. From informal subjective evaluation, results with reduced x-vectors were slightly worse, therefore regular x-vectors were used.

[2] https://github.com/kaldi-asr/kaldi/tree/master/egs/voxceleb/v2

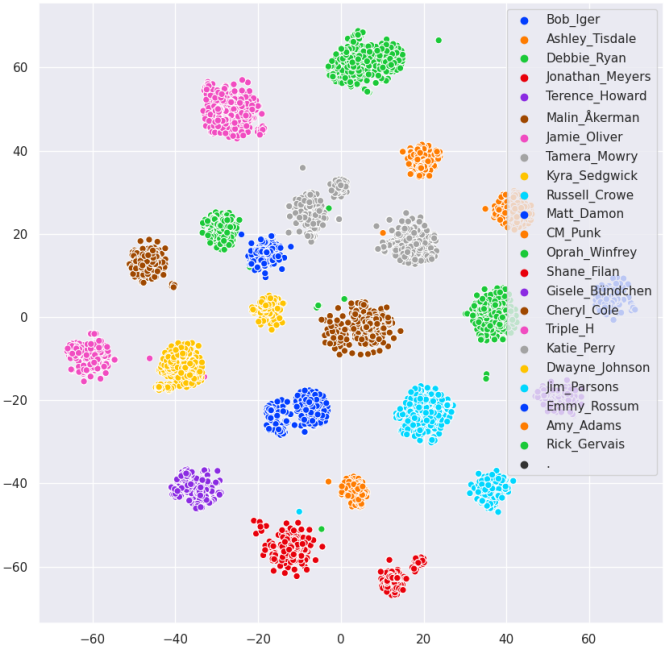


Figure 7.2: t-SNE visualization of x-vector embeddings extracted from VoxCeleb dataset. Speakers are now well separated.

7.3 Improving Disentanglement of Speaker and Content Information

As described in Section 4.4, AutoVC's disentanglement properties come from carefully designed bottleneck size. When the bottleneck is too narrow, resulting speech looses naturalness and intelligibility significantly. On the other hand, too wide bottleneck decreases level of disentanglement so that resulting speech contains too much information from the source speaker. To address this issue, two different approaches are tested. One with adversarial speaker classifier as used in method described in section 4.3, another approach is inspired by CycleVAE's training process.

Hypothesis is, that both approaches should reduce amount of source speaker information in the bottleneck content codes as well as to allow to increase the size of the bottleneck, therefore improve speech quality.

7.3.1 Auxiliary speaker classifier

In this modification, AutoVC's original architecture, shown in figure 4.6, is preserved, but a simple speaker classifier is used to classify content codes (where speaker information should be suppressed) in an adversarial training setup. Modified version is shown in figure 7.3.

Authors of AutoVC [14] state, that bottleneck tuning is more effective, than adversarial speaker classifier, while comparing AutoVC with other baseline method, that uses

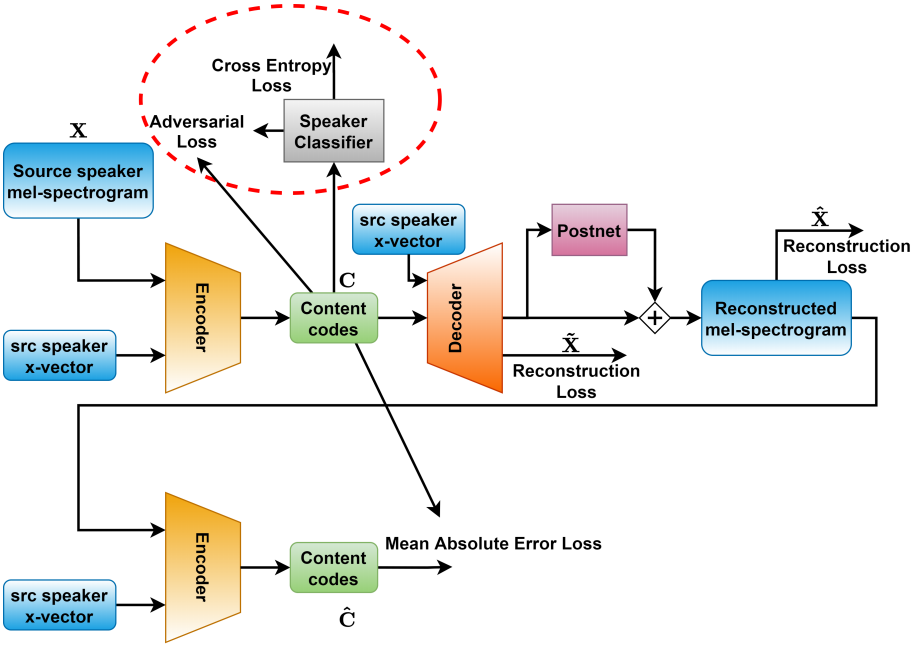


Figure 7.3: Scheme of AutoVC with auxiliary speaker classifier in red oval.

Table 7.2: Architecture of the Speaker Encoder.

Speaker Classifier			
	Layer	**Misc**	**Output**
	Input	-	$2d_{neck} \times \frac{T}{d_f}$
$3 \times$ {	*Conv1d*	$k = 1$	$2d_{neck} \times \frac{T}{d_f}$
	BatchNorm	-	-
	LeakyReLU	-	-
	Linear	-	$N_{spkrs} \times \frac{T}{d_f}$

adversarial classifier: "This result shows that bottleneck dimension tuning on speaker disentanglement is more effective than the more sophisticated adversarial training."

Although AutoVC has better results, than mentioned baseline, the authors did not try to merge both approaches. It is quite time consuming to find proper size of the bottleneck and this approach might allow to use slightly larger bottleneck size and discard remaining source speaker information with adversarial speaker classifier. This might also improve naturalness of the reconstructed speech for the same reasons.

For this experiment, the same classifier as in the system described in section 4.3 was used, also with the same loss functions from equations 4.16 and 4.19. The architecture of the speaker classifier is detailed in table 7.2, it classifies speaker for each frame in the utterance. If this classifier were used as standalone system for speaker identification, it probably wouldn't be very good, because of the lack of the temporal context. However, it should be sufficient for this task.

Complete loss function for this system is the following:

$$\mathcal{L}_{SC} = CE(\mathbf{C}, \mathbf{s}) \tag{7.3}$$

$$\mathcal{L}_{AE} = \mathcal{L}_{PSNT} + \mathcal{L}_D + \mathcal{L}_E + \lambda_{adv}\mathcal{L}_{ADV}, \tag{7.4}$$

where $\mathcal{L}_{SC}$ is loss for speaker classifier and L_{AE} for autoencoder. $\mathcal{L}_{ADV}$ is adversarial loss, it has weight $\lambda_{adv} = 0.5$ to keep emphasis on the original autoencoder training.

Training of this network is divided into two alternating stages. In the first stage, parameters of the classifier are frozen and only autoencoder's weights are updated. In the second stage, it is the other way around, i.e. autoencoder's weights are frozen and only the classifier is updated. The changing period was set to $100k$ iterations with $8:2$ ratio in favor of autoencoder. Evolution of the accuracy in training process is shown in figure 7.4.

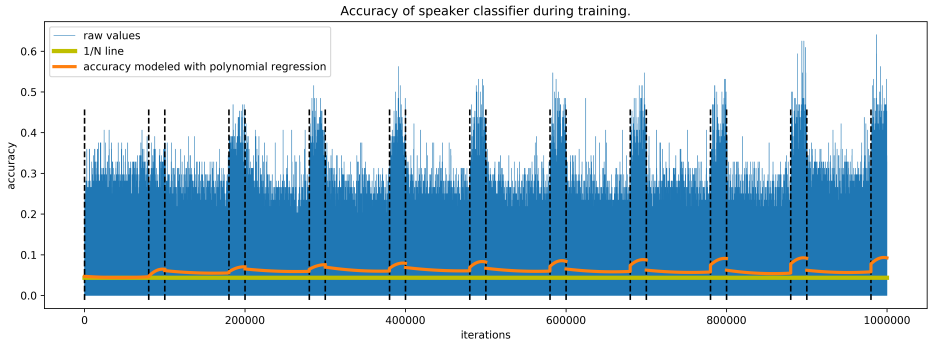


Figure 7.4: Plot of accuracy of the speaker classifier during training process. Yellow line shows desired probability of $\frac{1}{N}$.

7.3.2 Bottleneck consistency training

The second tested change is partially inspired by CycleVAE (section 4.2). The architecture of the AutoVC system stays the same, but the training process is modified. In the original AutoVC, three terms are present in the loss function: $\mathcal{L}_{PSNT}$ and $\mathcal{L}_D$ are for self-reconstruction and the last:

$$\mathcal{L}_E = \mathbb{E}\left\|E\left(\hat{\mathbf{X}}\right) - \mathbf{C}\right\|_1$$

tries to keep content codes $\mathbf{C}$ the same. The second set of content codes is generated from the reconstructed Mel-spectrogram. Proposed modification is to extract target content codes from converted features of a different speaker. For better idea, the modification is visualized in figure 7.5 and the modified part of the loss function is shown in full format:

$$\mathcal{L}_{E_{mod}} = \mathbb{E}\left\|E\left(D\left(E\left(\mathbf{X}\right), \mathbf{e}_{rnd}\right)\right) - E\left(\mathbf{X}\right)\right\|_1. \tag{7.5}$$

Otherwise, the loss is the same as in original AutoVC (equation 4.29). The decoder is conditioned on random speakers embedding $\mathbf{e}_{rnd}$, in order not to learn conversion from one specific speaker to another.

The motivation for this change is, again, to force the autoencoder to better disentangle speaker information with possibility of keeping a bit larger bottleneck. Down side of this approach is, that one more pass through autoencoder is required and it increases training time.

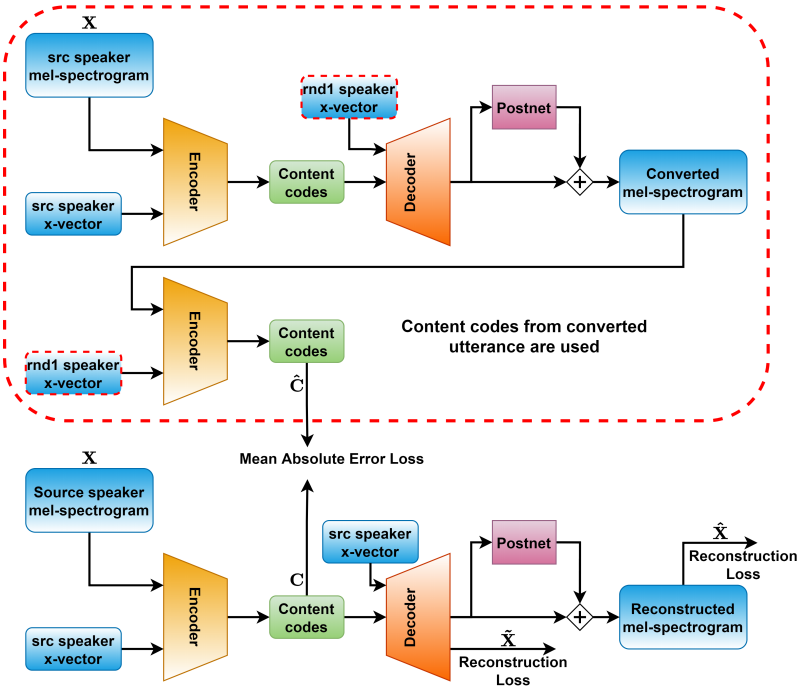


Figure 7.5: Scheme of AutoVC's modified training process. Extracted content codes are now compared with those from different speaker.

Table 7.3: Comparison of equal error rate on VoxCeleb 1 test dataset.

Name	Train dataset	Data size	No. iterations
PWG_VOX	VoxCeleb2 train	158 GB	$880k$
PWG_VCTK	VCTK (whole)	15 GB	1000k

7.4 Training and evaluation

In this section, configurations of trained networks are described. Later, the evaluation results are presented and discussed.

7.4.1 Vocoders

First, to compare speech samples, vocoder needs to be set up. AutoVC's authors published a WaveNet model pretrained on VCTK dataset[3]. This model was used for initial testing, but later, with growing size of the tests, WaveNet's inference speed became insufficient (generating 3 s of speech takes around 5 minutes). Parallel WaveGAN (PWG) networks were trained using data described in table 7.3. For training, the configuration from VCTK voc1 recipe in PWG repository[4] was used. Feature extraction was changed to the same one as in AutoVC, this changes also frame shift (hop-size) in STFT, therefore upsample scales needed to be reconfigured (for detailed explanation see section 5.1.1). Used values were $[4, 4, 4, 4]$. Parallel WaveGAN uses Rectified Adam optimizer [4] with default parameters.

[3] https://github.com/auspicious3000/autovc
[4] https://github.com/kan-bayashi/ParallelWaveGAN/tree/master/egs/vctk/voc1

Training of each system took around 6 days on Nvidia GeForce RTX 2080 Ti graphics card, which was also used for some other task during that time. PWG's inference time is faster than real-time with real-time factor of around 0.13. Real-time factor is defined as time required to generate one second of speech. Even though PWG uses the same feature extraction algorithm as AutoVC, PWG's features are normalized for training, therefore converted Mel-spectrogram needs to be normalized as well (subtract mean and divide by standard deviation).

7.4.2 AutoVC training

In order to train AutoVC on VoxCeleb data, bottleneck dimensionality needs to be estimated. 50 speakers from VCTK dataset were selected (further denoted as VCTK50 dataset) to find bottleneck size of AutoVC system. Estimating bottleneck size directly on VoxCeleb data would be much harder, given that results are uncertain.

In the original AutoVC implementation, bottleneck has size 32×32 ($d_{neck} \times d_f$; following notation from section 4.4). However, these settings did not work for VCTK50 dataset even though original AutoVC is evaluated also on VCTK subset (20 speakers). Reason behind this might be the use of x-vectors instead of d-vectors. Smaller sizes of d_{neck} always led to worse quality of speech, therefore smaller downsampling factor d_f is used[5]. AutoVC is trained using fixed speaker embedding for each speaker.

Resulting speech became intelligible with bottleneck size 32×8, using x-vectors as speaker embeddings and PWG_VCTK as vocoder. System with these settings was also submitted to the Voice Conversion Challenge 2020 in completely one-shot setting.

AutoVC system is trained using Adam optimizer [2] with learning rate set to 10^{-4}, other parameters are default. Training time of the original method is around 1 day. When using speaker classifier, the time rises to 1.5 days and with modified training, it reaches 2 days. Its inference speed is slightly faster than real-time.

Further, following notation holds:

- vanilla – original AutoVC implementation
- spk – AutoVC with speaker classifier
- targets – AutoVC with modified training process

For each method, systems with bottlenecks 32×8 and 32×4 were trained on VoxCeleb23 dataset. PWG_VOX was used as vocoder. The bigger[6] 32×4 bottleneck is chosen to test, whether proposed modifications are less sensitive to the bigger bottleneck.

Note, that all of these systems had first encounter with source or target speaker during inference time through one supplied x-vector.

From listening of a few random samples, resulting speech is sometimes unintelligible, but it highly depends on the quality of original utterance. Also, fundamental frequency is not explicitly converted in AutoVC method and it sometimes fluctuates between source and target speaker. Quian et. al. already published a follow up work, that addresses this isssue [13], but it is not covered in this thesis.

[5] Samples of converted speech using various bottleneck sizes are available on https://lemlak.github.io/VoiceConversion/#vcc20-tests-32x8

[6] This is a bit counter-intuitive, the second parameter is downsampling factor d_f, which affects time resolution, with smaller d_f more information gets through the bottleneck, therefore bottleneck is bigger. Downsampling process is defined in equation 4.23

Table 7.4: Evaluation of spoofing tests. Top table shows results using LDA reduction, bottom table shows results without LDA. Miss rate is computed using threshold of the original system. The best results in each are bold.

Name	Miss rate [%]	EER [%]	Threshold
original	3.91	3.91	0.331800
vanilla 32×4	7.37	5.33	0.352123
vanilla 32×8	7.28	5.40	0.352022
spk 32×4	7.23	5.31	0.351022
spk 32×8	**9.13**	5.9279	0.359745
targets 32×4	7.55	5.42	0.353497
targets 32×8	8.81	5.77	0.358251
original	9.70	9.70	0.672519
vanilla 32×4	23.64	14.62	0.702462
vanilla 32×8	20.59	13.75	0.698117
spk 32×4	24.81	15.11	0.705208
spk 32×8	25.76	15.28	0.706139
targets 32×4	21.99	14.26	0.700860
targets 32×8	**27.45**	15.47	0.707319

7.4.3 Evaluation

Spoofing and anonymization tests using cosine similarity as metric were conducted. Trials for verification were taken from VoxCeleb1 list. There is a total of 37720 tests with the same amount of target and nontarget trials.

Spoofing

In spoofing task, utterance of one speaker was converted to the second speaker for each nontarget trial. The goal is to increase false acceptance (miss) rate. Miss rate is computed as number of false accepted trials divided by total number of nontarget trials. Results of spoofing task are shown in table 7.4. The best system with LDA is `spk` 32×8 with miss rate increased by more than 5 % absolute. For setting without LDA, the best system is `targets` 32×8 with miss rate increased by more than 17 % absolute and all methods were more successful overall. This shows, that even simple LDA helps to create a more robust speaker verification systems.

Interestingly, modified methods have worse results with larger bottleneck 32×4 and only original `vanilla` method benefits from it. This might imply, that the bottleneck settings were still sub-optimal.

Spoofing studies In this subsection, the first task – spoofing with LDA – is closer examined. There is total of 18860 nontarget trials out of which 3804 (20.17 %) trials were spoofed by at least one system, when *excluding* trials falsely accepted by original verification system. Out of those 3804, 217 trials from 131 speaker pairs (considering $S_1 \rightarrow S_2$ and $S_2 \rightarrow S_1$ as one speaker pair) were spoofed by all the systems[7]. These utterances seem to be mostly intelligible, but they would be rather low on MOS scale.

[7] A few of these samples are in https://lemlak.github.io/VoiceConversion/#samples-from-systems-trained-on-voxceleb23-dataset

Table 7.5: Evaluation of anonymization tests. Top table shows results with using LDA reduction, bottom table shows results without LDA. The best results in each are bold. The last column shows adjusted results without „self-reconstruction error“.

Name	False alarm rate [%]	EER [%]	Threshold	Adj. FAR [%]
original	3.91	3.91	0.331800	-
vanilla 32×4	85.68	36.00	0.085723	24.49
vanilla 32×8	**93.83**	43.32	0.044691	16.53
spk 32×4	88.32	37.85	0.075743	**24.74**
spk 32×8	91.99	42.93	0.047043	22.93
targets 32×4	91.21	39.69	0.065066	24.60
targets 32×8	93.82	43.63	0.043012	17.08
original	9.70	9.70	0.672519	-
vanilla 32×4	75.99	41.78	0.465801	**39.81**
vanilla 32×8	**85.88**	47.36	0.415702	26.39
spk 32×4	79.81	42.28	0.462104	38.86
spk 32×8	83.82	46.77	0.421578	34.23
targets 32×4	82.81	44.27	0.444932	36.71
targets 32×8	84.70	46.13	0.427971	32.09

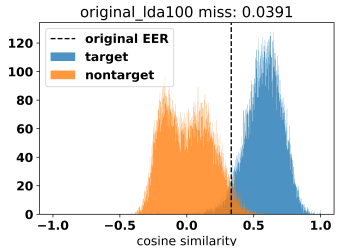
Table 7.6: Evaluation of self-reconstructions tests. Top table shows results with using LDA reduction, bottom table shows results without LDA. The best results in each are bold.

	Name	False alarm rate [%]	EER [%]	Threshold
w/ LDA	original	3.91	3.91	0.331800
	vanilla 32×4	**56.19**	18.64	0.185851
	vanilla 32×8	77.30	27.60	0.133703
	spk 32×4	63.58	19.55	0.179817
	spk 32×8	69.06	23.91	0.153849
	targets 32×4	66.61	21.29	0.168811
	targets 32×8	76.74	24.19	0.152333
w/o LDA	original	9.70	9.70	0.672519
	vanilla 32×4	**36.18**	18.64	0.616193
	vanilla 32×8	59.49	27.85	0.559989
	spk 32×4	40.95	19.88	0.609124
	spk 32×8	49.59	23.83	0.584766
	targets 32×4	46.10	21.17	0.600744
	targets 32×8	52.61	23.15	0.588615

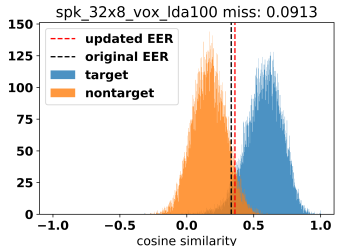
Anonymization

The second task was to measure anonymization. This time, one utterance from each target trial was converted to random different speaker. The goal is to increase false alarm rate, falsely rejected trials divided by number of target trials. All methods achieved high score in this task, the question is, whether this result is a reflection of good anonymization or low quality speech sample, which wouldn't be accepted for any speaker. The highest score was achieved by `vanilla` 32×8 system both with and without LDA. Results are shown in table 7.5.

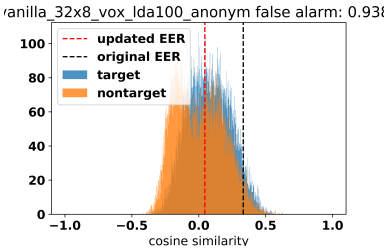
Self-reconstruction To address the question about speech quality, test with self-reconstruction was conducted. Self-reconstruction means, that speaker is converted back to him/herself. While in anonymization, we were looking for system with the highest false alarm rate, this time, smaller false alarm rate is better. There would be no change in ideal case. We can see in table 7.6, that the best self-reconstruction is achieved by system `vanilla` 32×4. Given rather small miss rate, this system achieved, it is probably due to the fact, that there is too much speaker information left in the bottleneck (this is also the largest bottleneck). When self-reconstruction false alarm rate is subtracted from anonymization results, system `vanilla` 32×8 is now on the opposite side. This adjusted false alarm rate should represent anonymization better, without influence of the global error of the voice conversion. Adjusted results are in table 7.5 in the last column. Figure 7.6 shows histograms[8] of cosine similarities for the best methods.



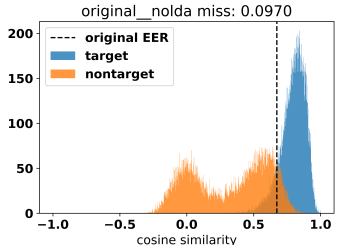
(a) Original X-vector system.



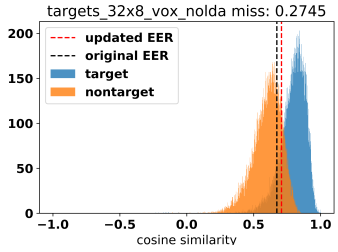
(b) System with the highest miss rate on spoofing test. Spk 32×8.



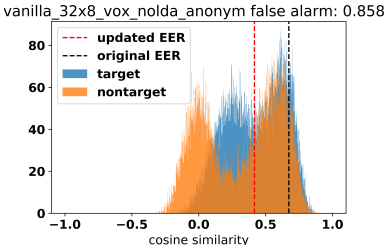
(c) System with the highest false alarm rate on anonymization test. Vanilla 32×8



(d) Original X-vector system.



(e) System with the highest miss rate on spoofing test. Targets 32×8.



(f) System with the highest false alarm rate on anonymization. Vanilla 32×8.

Figure 7.6: Histograms showing distribution of target and nontarget trials for different tasks. Top histograms are with LDA and bottom ones are without LDA. Black dotted line denotes original threshold and the red one threshold for EER on modified data.

[8] All histograms are available at https://lemlak.github.io/VoiceConversion/#spoofing-and-anonymization-histograms

Chapter 8

Conclusion

The voice conversion using speaker disentanglement was studied in this thesis. It follows the state-of-the-art AutoVC architecture defined by Qian et al.

The emphasis is on one-shot voice conversion, where conversion is tested on speakers outside of training data and on conversion on dataset with low-quality data, VoxCeleb. Objective evaluations on speaker verification/spoofing were conducted with aim to fool verification system by either impersonating the target speaker or hiding speakers identity.

Two modifications of baseline AutoVC system were proposed: one using adversarial speaker classifier and the other one based on comparing original content embeddings with those obtained after the conversion process. Furthermore, more robust speaker embeddings – x-vectors – were used to ensure one-shot properties of the VC system.

The results show, that both proposed modifications outperform regular AutoVC in terms of spoofing. The best created conversion system was able to increase miss rate by 5 % absolute when using Linear Discriminant Analysis and by 17 % absolute without it. This proves, that resulting speech was shifted towards target speaker even under these hard conditions.

Output speech quality was not evaluated in formal listening test, however, it highly depends on specific utterance. Background noise or low quality microphone can cause converted utterance to be completely unintelligible while cleaner recordings produce reasonably well converted samples.

8.1 Future works

In my opinion, AutoVC framework is promising concept of voice conversion, mainly due to simplicity of training, without need any text transcriptions, but it definitely needs to include conversion of fundamental frequency.

Auxiliary speaker classifier might be further tested with different architectures. Used implementation classifies each frame and it might be interesting to use some recurrent layer. Also, different methods of training might be used, without freezing the parameters. Combination of both proposed improvement methods can be also tested.

8.1.1 Follow-up works

Speech is does not consist only of speaker information and content information. It can be separated to more parts: fundamental frequency and intonation, speaking rate or even

emotions are stored in speech. All these parts might be extracted and played with to further improve our understanding of speech generation and perception.

Bibliography

[1] Kawahara, H., Masuda Katsuse, I. and De Cheveigne, A. 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*. Elsevier. 1999, vol. 27, 3-4, p. 187–207.

[2] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *ArXiv preprint arXiv:1412.6980*. 2014.

[3] Kobayashi, K. and Toda, T. Sprocket: Open-Source Voice Conversion Software. In: *Odyssey*. 2018, p. 203–210. Available at: https://github.com/k2kobayashi/sprocket.

[4] Liu, L., Jiang, H., He, P., Chen, W., Liu, X. et al. On the variance of the adaptive learning rate and beyond. *ArXiv preprint arXiv:1908.03265*. 2019.

[5] Lorenzo Trueba, J., Yamagishi, J., Toda, T., Saito, D., Villavicencio, F. et al. The voice conversion challenge 2018: promoting development of parallel and nonparallel methods. In: *Proc. Odyssey 2018*. 2018, p. 195–2026.

[6] Maaten, L. v. d. and Hinton, G. Visualizing data using t-SNE. *Journal of machine learning research*. 2008, vol. 9, Nov, p. 2579–2605.

[7] Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z. et al. Least squares generative adversarial networks. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, p. 2794–2802.

[8] Morise, M., Yokomori, F. and Ozawa, K. WORLD: a vocoder-based high-quality speech synthesis system for real-time applications. *IEICE TRANSACTIONS on Information and Systems*. The Institute of Electronics, Information and Communication Engineers. 2016, vol. 99, no. 7, p. 1877–1884.

[9] Mysore, G. J. Can we Automatically Transform Speech Recorded on Common Consumer Devices in Real-World Environments into Professional Production Quality Speech?—A Dataset, Insights, and Challenges. *IEEE Signal Processing Letters*. 2015, vol. 22, no. 8, p. 1006–1010.

[10] Nagrani, A., Chung, J. S., Xie, W. and Zisserman, A. VoxCeleb: Large-scale Speaker Verification in the Wild. *Computer Speech & Language*. october 2019, vol. 60, p. 101027.

[11] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H.,

Larochelle, H., Beygelzimer, A., Buc, F. d'Alché, Fox, E. et al., ed. *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, p. 8024–8035. Available at: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[12] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O. et al. The Kaldi speech recognition toolkit. In: *IEEE 2011 workshop on automatic speech recognition and understanding*. 2011.

[13] Qian, K., Jin, Z., Hasegawa Johnson, M. and Mysore, G. J. F0-consistent many-to-many non-parallel voice conversion via conditional autoencoder. In: IEEE. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, p. 6284–6288.

[14] Qian, K., Zhang, Y., Chang, S., Yang, X. and Hasegawa Johnson, M. AUTOVC: Zero-shot voice style transfer with only autoencoder loss. *ArXiv preprint arXiv:1905.05879*. 2019.

[15] Rocca, J. *Understanding Variational Autoencoders (VAEs)*. September 2019. [online]. Available at: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.

[16] Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N. et al. Natural TTS Synthesis by Conditioning Wavenet on MEL Spectrogram Predictions. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. April 2018, p. 4779–4783. ISSN 2379-190X.

[17] Snyder, D., Garcia-Romero, D., Sell, G., Povey, D. and Khudanpur, S. X-Vectors: Robust DNN Embeddings for Speaker Recognition. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, p. 5329–5333.

[18] Tobing, P. L., Wu, Y.-C., Hayashi, T., Kobayashi, K. and Toda, T. Non-Parallel Voice Conversion with Cyclic Variational Autoencoder. In: *Proc. Interspeech 2019*. 2019, p. 674–678. Available at: http://dx.doi.org/10.21437/Interspeech.2019-2307.

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

[20] Toda, T., Chen, L.-H., Saito, D., Villavicencio, F., Wester, M. et al. The Voice Conversion Challenge 2016. In: *Proc. INTERSPEECH*. 2016, p. 1632–1636.

[21] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O. et al. WaveNet: A Generative Model for Raw Audio. *ArXiv e-prints*. Sep 2016, p. arXiv:1609.03499.

[22] Veaux, C., Yamagishi, J., MacDonald, K. et al. Superseded-cstr vctk corpus: English multi-speaker corpus for cstr voice cloning toolkit. University of Edinburgh. The Centre for Speech Technology Research (CSTR). 2016.

[23] Wan, L., Wang, Q., Papir, A. and Moreno, I. L. Generalized end-to-end loss for speaker verification. In: IEEE. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, p. 4879–4883.

[24] Wrench, A. *The MOCHA-TIMIT Articulatory Database*. November 1999. [online]. Available at: http://www.cstr.ed.ac.uk/research/projects/artic/mocha.html.

[25] Wu, Z., Watts, O. and King, S. Merlin: An Open Source Neural Network Speech Synthesis System. In: *9th ISCA Speech Synthesis Workshop*. 2016, p. 202–207. Available at: http://dx.doi.org/10.21437/SSW.2016-33.

[26] Yamamoto, R., Song, E. and Kim, J.-M. Parallel WaveGAN: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram. In: IEEE. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, p. 6199–6203.

[27] Zhang, J., Ling, Z. and Dai, L. Non-Parallel Sequence-to-Sequence Voice Conversion With Disentangled Linguistic and Speaker Representations. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. 2020, vol. 28, p. 540–552. ISSN 2329-9304.

Appendices

Appendix A

Cookbook

A.1 Libraries and Code

Here are listed libraries used for training and experiments with AutoVC. Other versions might also work.

- Python >=3.6
- Pytorch 1.3 for neural network model, datasets, etc.
- CUDA 10.0
- TensorFlow 1.14.0, used only for hyperparameters in contrib module. This module which was deprecated from version 2.0
- NumPy 1.18.3
- SciPy 1.4.1
- scikit-learn 0.22.1
- librosa 0.7.2

AutoVC code was used from their authors.

Kaldi-asr was used to extract x-vectors in following way. After `wav.scp`, `spk2utt` and `utt2spk` are created[1] MFCC features are created and then vad decisions. With data prepared, x-vectors can be extracted using pretrained model[2].

To extract mel-spectrograms, script `make_spect.py` is ran in folder containing `wavs` directory with folders with utterances divided by speakers. Mel-spectrograms are generated into `spmel` directory, again with the utterances divided into folders by speakers. Now corresponding embeddings are copied into the speaker folders, one embedding for each utterance. Script `make_metadata` creates training file in `spmel` directory. `main.py` script is used for network training.

To convert samples, script `convert_cycle.py` can be used. It takes as parameters list of utterances to convert in format `<wav> <srcid> <trgid>`, map of embeddings that contains speakers in the list and trained network model.

[1] https://kaldi-asr.org/doc/data_prep.html
[2] https://kaldi-asr.org/models/m7

A.2 Media Content

Root folder contains directories:

- src: source codes of AutoVC, modifications and ParallelWaveGAN
- models: trained neural networks
- images: histograms from verification
- samples: sample converted utterances
- webpage: source codes for webpage https://lemlak.github.io/VoiceConversion/, together with samples and images (duplicates)
- misc: contains map of xvectors, example list of utterances and trained lda model