EHB 328E MLSP
PROJECT PROGRESS REPORT

VOICE CONVERSION

By Using Generative Adversarial Networks

mlsp2020/mlsp-voice-conversion (github.com)

GROUP MEMBERS

Selahaddin HONİ
040160046

İmran Çağla EYÜBOĞLU
040160230

İsmail Melik TÜRKER
040160240

CONTENT

Fall, 2020

Introduction to Voice Conversion (VC): [1]

Aim of this project is manipulating and generating some famous person's voice by using voice conversion (VC) techniques. Main motivation behind VC is extracting the feature of source voice and regenerate an output to target form (as another target voice or text etc.). One of the most common application of VC is speech to text or text to speech conversion. In our study we will try to use VC techniques and generate a famous person's voice. Some of these techniques are Codebook mapping, Vector Quantization, Dynamic Frequency Warping, Hidden Markov Models, Gaussian Mixture Models and Artificial Neural Networks.

After Ezzine and Frikha compare the techniques applied on voice conversion in their paper, they state as a conclusion that "*The perceptual tests shown that techniques based on neural network provide better output quality even in the presence of noise once the number of layers increases in the neural network. More precisely, to obtain greater synthesis accuracy we should essentially get a deeper non-linear architecture with lots of hidden layers."* Therefore, use of neural networks is chosen to realize our project desire.

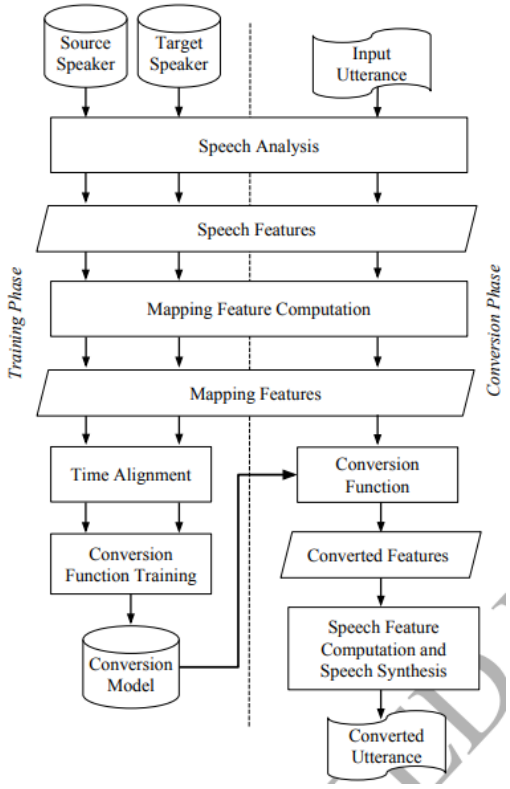


Figure 1: A scheme of VC process [2]

VC basically consist of two phases such as training phase and conversion phase [1].

- In the training phase it is required to collect data from source and target speakers. Then, collected data is evaluated and its' distinctive features are extracted and formulated by the system.
- The conversion phase utilizes this formulation to resemble source voice to target voice.

There are three stages in training phase: acoustic analysis, alignment and model training.

- Acoustic analysis acquires some distinctive features of source and target voices in order to define characteristics.
- Alignment stage detects resembling elements in both source and target speakers.
- Training step creates a model to convert source to target speakers. Also, we can say that conversion algorithm is involved in this stage.

Introduction to Generative Adversarial Network:

GAN is simply a neural network type that consists of two models: generator and discriminator. Generator is named 'the artist' while the discriminator 'the art critic' in a post of *Tensorflow* tutorials. *"During training, the* generator *progressively becomes better at creating images that look real, while the* discriminator *becomes better at telling them apart. The process reaches equilibrium when the* discriminator *can no longer distinguish real images from fakes."* [3]

Because we are beginner in Generative Adversarial Network's training process, we tried to follow how a Deep Convolutional GAN is designed step by step in a tutorial project which is generating hand-written digits by training MNIST dataset.

Interactive Python notebook file is uploaded to the project *GitHub* page. However; result figures are shown in this paper to give the main idea.

The figure given below is the first attempt of generator to generate a fake hand-written digit image from a noise. Of course, the discriminator is going to label this generated image as fake.

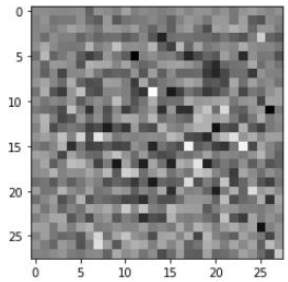


Figure 2 Non-trained model output

After epochs of training, the generator improves the output images for given inputs and here are the new results. They are more likely to be real digits.

Figure 3 Trained model output

Conversion Specified Network: CycleGAN

CycleGANs are popular in style transferring applications due to its convenient architecture to capture the characteristics of both source and target datasets. [4]

Again, a tutorial project is investigated which implements the original paper of CycleGAN. [5]

To be honest, it is a little bit hard to understand the whole mathematical concept behind this architecture, e.g. loss functions for both forward and backward cycle consistency. Most probably, we will not write the codes from scratch; instead, use open-source project files to train our dataset.

This figure given below is cropped from StarGAN-VC paper. [6] It explains the training procedure better by visually. Generator G, tries to generate a transform from given source input to target voice in our project. Yet, F generator gets input as generated voice from G and gives back to G with the conversion of target characteristics. By the way, discriminators decide to generated voices real or fake.

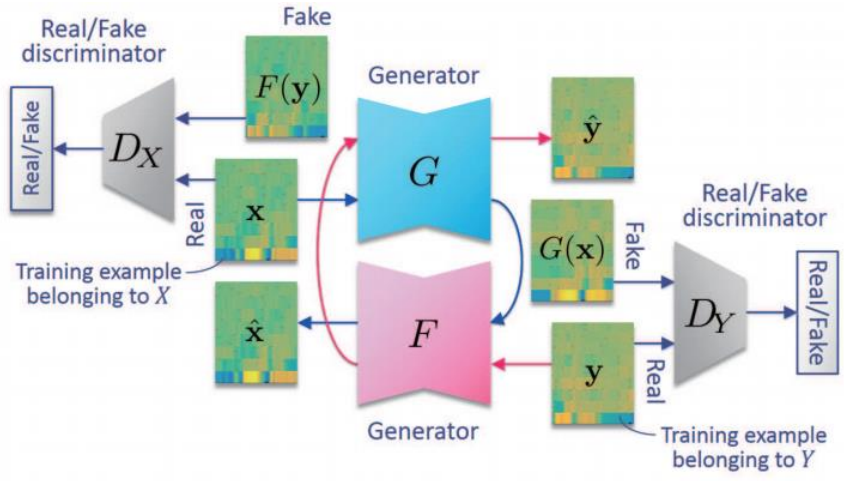


Figure 4: Training process of CycleGAN [6]

Dataset

It is selected the Text-to-Speech voice as source voice so that we can give input our model as text file with a tricky way. A Python API script is written for this purpose; we have ability to construct a connection to Google voice with following simple commands:

```
# Text-to-Speech
# This script generates a voice for given text input
from gtts import gTTS

text_en = 'A voice demo for source speaker'
text_tr = 'Kaynak konuşmacı için ses denemesi'

tts_en = gTTS(text_en, lang='en')
tts_tr = gTTS(text_tr, lang='tr')

tts_en.save('source/source_female_google_en.mp3')
tts_tr.save('source/source_female_google_tr.mp3')
```

Source: Google Text-to-Speech API is used to generate the source speaker voice.

Target: Female voice (Ece Uner) (2 min) is chosen for now; it is possible to enlarge

On the other hand, in a previous work of Takuhiro Kaneko et.al. they also used 5 minutes of training speech. [7]

Audio Tensors in Tensorflow Library

A jupyter notebook is written to learn how to import audio files into Tensorflow environment as tensors, operate basic dimension conversions, generate waveforms and spectrograms.

In the previous section, a TTS voice is generated which is “Kaynak konuşmacı için ses denemesi” in Turkish. This speech is examined:

Audio file imported as an AudioIOTensor; we need to convert it to tf.Tensor() object

```
import tensorflow as tf
import tensorflow_io as tfio
audio = tfio.audio.AudioIOTensor('source/source_female_google_tr.mp3')
print(audio)
```

```
<AudioIOTensor: shape=[80064     1], dtype=<dtype: 'float32'>, rate=24000>
```

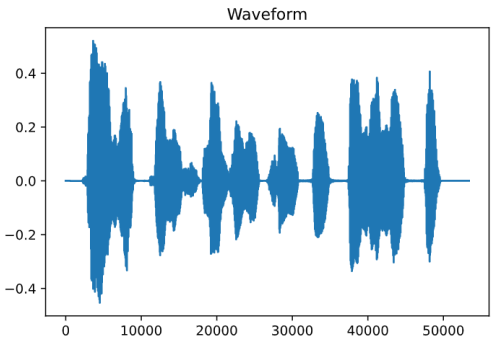
Removing the last dimension by squeeze() command changed the shape. Also resampling and slicing reduce the data samples.

```
# remove last dimension
audio_tensor = tf.squeeze(audio[:], axis=[-1])
print("Dimension Eliminated \n",audio_tensor)
# resample
audio_tensor = tfio.audio.resample(audio_tensor, 24000, 16000, name=None)
print("After Resampling \n",audio_tensor)
# slice
audio_tensor = audio_tensor[:]
print("Sliced \n",audio_tensor)
```

```
Dimension Eliminated
 tf.Tensor(
[ 0.0000000e+00  2.3279690e-12 -1.2419072e-12 ... -1.9600179e-07
 -1.5530611e-07  1.9855221e-08], shape=(80064,), dtype=float32)
After Resampling
 tf.Tensor(
[ 0.0000000e+00 -2.4585088e-17  7.6834465e-17 ...  7.4304275e-08
  2.5449455e-07 -1.6132046e-07], shape=(53376,), dtype=float32)
Sliced
 tf.Tensor(
[ 0.0000000e+00 -2.4585088e-17  7.6834465e-17 ...  7.4304275e-08
  2.5449455e-07 -1.6132046e-07], shape=(53376,), dtype=float32)
```

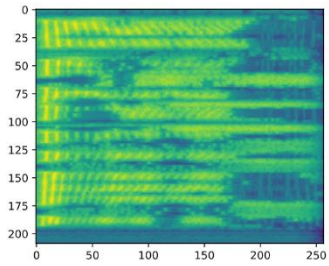
If data type is not float32; then data type casting should be applied. It is more convenient to use tensors in float domain.

```
import matplotlib.pyplot as plt
# audio_tensor = tf.cast(audio_tensor, tf.float32) / 32768.0
plt.figure()
plt.plot(audio_tensor.numpy())
plt.title("Waveform")
```



```
# Convert to spectrogram
spectrogram = tfio.experimental.audio.spectrogram(
    audio_tensor, nfft=512, window=512, stride=256)

plt.figure()
plt.imshow(tf.math.log(spectrogram).numpy())
```

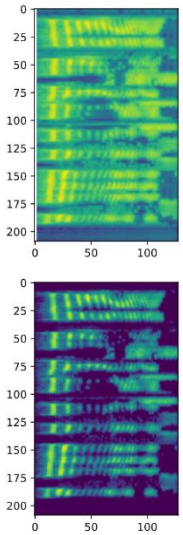



```
# Convert to mel-spectrogram
mel_spectrogram = tfio.experimental.audio.melscale(
    spectrogram, rate=16000, mels=128, fmin=0, fmax=8000)

plt.figure()
plt.imshow(tf.math.log(mel_spectrogram).numpy())

# Convert to db scale mel-spectrogram
dbscale_mel_spectrogram = tfio.experimental.audio.dbscale(
    mel_spectrogram, top_db=80)

plt.figure()
plt.imshow(dbscale_mel_spectrogram.numpy())
```



Open-Source Projects for Voice Conversion

As explained before, writing the codes from scratch is too hard and not necessary. Therefore, it is required to reference not only papers, but also project files.

Takuhiro Kaneko and Hirokazu Kameoka are researchers in NTT Communication Science Laboratories in NTT Corporation. Their works (CycleGAN-VC, CycleGAN-VC2, CycleGAN-V3, STARGAN-VC, STARGAN-VC2) are excellent for the future of our project.

GitHub repos are linked:

CycleGAN-VC: Voice Converter Using CycleGAN and Non-Parallel Data [8]

StarGAN-VC: Non-parallel many-to-many voice conversion with star generative adversarial networks [9]

References

[1] Ezzine, Frikha (2017). *A Comparative Study of Voice Conversion Techniques: A review*

[2] Mohammedi, Kain (2017). *An Overview of Voice Conversion Systems*

[3] Tensorflow Tutorials, *What are GANs?*

Retrieved from https://www.tensorflow.org/tutorials/generative/dcgan

[4] Tensorflow Tutorials, *CycleGAN*

Retrieved from https://www.tensorflow.org/tutorials/generative/cyclegan

[5] J. Zhu, T. Park, Isola and Efros (2020). *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*

[6] Kameoka, Kaneko, Tanaka and Hojo (2018). *StarGAN-VC: Non-Parallel Many-to-Many Voice Conversion With Star Generative Adversarial Networks*

[7] Kaneko, Kameoka, Tanaka and Hojo (2020). *CycleGAN-VC3: Examining and Improving CycleGAN-VCs for Mel-spectrogram Conversion*

Retrieved from http://www.kecl.ntt.co.jp/people/kaneko.takuhiro/projects/cyclegan-vc2/index.html

[8] GitHub repo: CycleGAN-VC

Retrieved from leimao/Voice_Converter_CycleGAN: Voice Converter Using CycleGAN and Non-Parallel Data (github.com)

[8] GitHub repo: StarGAN-VC

Retrieved from liusongxiang/StarGAN-Voice-Conversion: This is a pytorch implementation of the paper: StarGAN-VC: Non-parallel many-to-many voice conversion with star generative adversarial networks (github.com)