ATIAM 2022 - ML Project
Can you speak like a violin?
A multi-generator approach to timbre transfer

Sarah Nabi[1] Axel Chemla–Romeu-Santos[1]
[1] Institut de Recherche et Coordination Acoustique Musique (IRCAM)
UMPC - CNRS UMR 9912 - 1, Place Igor Stravinsky, F-75004 Paris
{nabi, chemla}@ircam.fr

November 2022

Abstract

In recent years, deep generative models have achieved impressive results in generating high-quality audio samples that reproduce the properties of a given training set distribution. These recent advances have transformed deep generative models into promising tools for musical creation leading to several applications such as *timbre transfer*. The goal is to transform the timbre of a source audio from one instrument to another target timbre while preserving the other core features such as the loudness, pitch, and rhythm. This can be particularly useful for composers who want to quickly experiment the sound of different instruments on a same melody. To achieve this, existing methods mostly rely on *latent-based generative models* such as *Generative Adversarial Networks* (GAN) [6] and *Variational Auto-Encoders* (VAE) [8] which can provide a trainable analysis-synthesis framework reconstructing audio samples through a *latent* representation that captures high-level signal features.

In this project, we will leverage *domain translation* techniques and extend the many-to-many voice conversion approach proposed by Albadawy et al. [1] to voice and instrumental timbre transfer. Hence, we will combine VAEs with GANs using a single encoder and multiple target domain generators on mel-spectrograms, in order to obtain meaningful representations of the source audio signal and generate realistic audio samples across multiple target domains related to different instruments and speakers. To ensure each generator only captures the timbre features associated to its target domain and preserves the other core features from the source domain, we will regularize the latent space with a *cycle-consistency* loss so that one input sample shares the same latent representation across all target domains.

1 Introduction

In recent years, neural audio synthesis has become an actively research topic offering new creative prospects in terms of composition and sound design. Generative models such as *Differentiable Digital Signal Processing* (DDSP) [5], GANSynth [4] and *Real-time Audio Variational auto-Encoder* (RAVE) [3], have achieved impressive results in generating high-quality audio samples that reproduce the properties of a given training set. To achieve this goal, existing methods mostly leverage *Generative Adversarial Networks* (GAN) [6] and *Variational Auto-Encoders* (VAE) [8] which can provide a trainable analysis-synthesis framework reconstructing audio samples through a *latent* representation that captures high-level signal features.

Advances in deep learning techniques for audio generation have yielded a wide range of new musical applications such as timbre transfer between instruments which aims at modifying the timbre of an audio sample while preserving all its other features. This can be particularly useful for voice anonymisation, music production and data augmentation for instance. Most timbre transfer approaches draw inspiration from *neural style transfer* and *domain translation* techniques initially introduced in the image field. A notable prior work on this topic is the *UNsupervised Image-to-image Translation* (UNIT) model proposed by Liu et al. [10] for image-to-image translation from one domain to another in an unsupervised manner. It takes a probabilistic modeling perspective by making the underlying assumption of a shared latent space between the two domains. This means that two potentially matching samples should have the same latent representation. The translation between both domains is achieved by partially weight-shared VAEs. The learning is carried out through an adversarial criterion in both directions with a *cycle consistency loss* initially introduced in the CycleGAN model [15]. Although the latent representation can enhance high-level features, this method only performs *one-to-one domain translation*. In the audio domain, the *Universal Music Translation* (UMT) model [12] has extended this work to perform *many-to-many* timbre transfer by leveraging a WaveNet [13] encoder and separate decoders for each domains to learn a shared representation of the raw waveform.

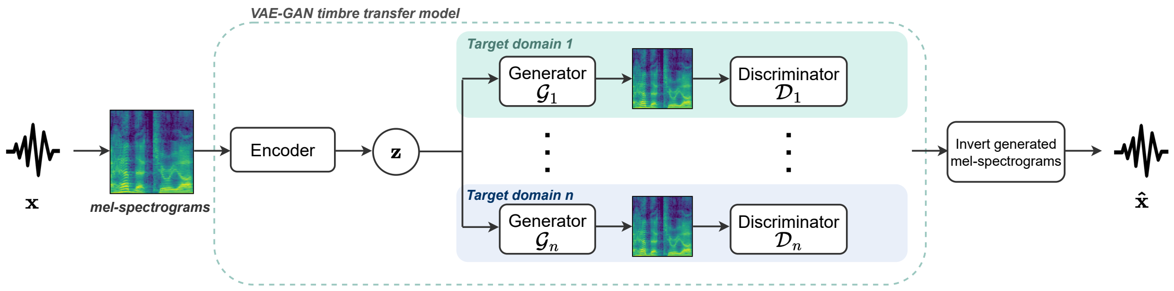

VAE-GAN timbre transfer model
Target domain 1
Generator $\mathcal{G}_1$
Discriminator $\mathcal{D}_1$
Target domain n
Generator $\mathcal{G}_n$
Discriminator $\mathcal{D}_n$
$\mathbf{x}$
mel-spectrograms
Encoder
$\mathbf{z}$
Invert generated mel-spectrograms
$\hat{\mathbf{x}}$

Figure 1: Overview of the proposed approach combining VAEs with GANs to perform *many-to-many* timbre transfer on mel-spectrograms. The model is composed of a single *encoder* and a *generator-discriminator* per target domain.

In this project, you will focus on the frequency domain and extend the many-to-many voice conversion approach proposed by Albadawy et al. [1] to timbre transfer between voices and instruments. As depicted in Figure 1, you will implement and train a VAE-GAN model with a single encoder and multiple target domain decoders on mel-spectrograms to transform the input sample from a source domain to different target domains. You will further improve the model performance by adding a *cycle consistency loss* and a *latent space loss* on the VAE encoder features embeddings as proposed in the original paper [1] to enforce the encoder to eliminate the specific timbre features from the latent space so that one input sample share the same latent vector across all target domains

2 Getting started

In this project we will use PyTorch [1], a versatile machine-learning Python framework that you will use to code your VAE-GAN-based model for timbre transfer. First, you will implement and train a VAE and a GAN on a simpler problem: digits generation. Therefore, you will use the MNIST dataset which is composed of 70,000 examples of hand written digits from 0 to 9, and which is often used as a playground to test new methods. We suggest that you write your code in a *modular* way so that you can reuse it for the rest of the project.

Exercise 0: Setup your environment and learn Pytorch

This exercise only includes basic programming skills that you will need for the following work.

1. Create the github repository that will host your wonderful code.
2. Create a virtual environment with Python 3.9. We advise you to use the `conda` or `miniconda` package manager to set your dependecies properly [2]. This is not mandatory, but can save you a lot of time in case of problems. It is also advised you structure your code as a *package* [3]. For example, you can start with the following structure:

```
timbre-transfer/
 config/   # directory to store the models config .yaml template files
 data/     # directory to store the data in local /!\ DO NOT COMMIT /!\
 docs/     # for github pages content
```

[1]https://pytorch.org/
[2]https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html
[3]https://docs.python-guide.org/writing/structure/

```
models/  # directory to store checkpoints in local /!\ DO NOT COMMIT /!\
notebooks/  # jupyter notebooks for data exploration and models analysis
README.md   # documentation
requirements.txt   # python project dependencies with versions
src/
   timbre_transfer/  # main package
       __init__.py
       datasets/       # data preprocessing and dataloader functions
          __init__.py
       helpers/        # global utility functions
          __init__.py
       models/         # models architecture defined as class objects
          __init__.py
tests/   # tests package with unit tests
```

3. Install PyTorch in your virtual environment. Many tutorial are available on their website [4] and it is strongly advised to become familiar with its use before continuing the project. Follow the basic tutorials and learn how it works `https://pytorch.org/tutorials/beginner/basics/intro.html`. Learn how to implement basic models using `torch.nn` module.

4. Install and read about the support libraries for audio signal processing: `librosa` [5], `torchaudio` [6] and `tifresi` [7].

5. Install and learn `tensorboard` [8] in order to monitor your trainings.

Exercise 1: Build a simple VAE

You will now code your own convolutional VAE on the MNIST dataset. You will first look at the main article's implementation [8], and then implement it on Pytorch guided by this nice tutorial : `https://keras.io/examples/generative/vae/` (programmed in Keras otherwise it's too simple).

1. Based on the tutorial or another source you will find, develop your very own VAE.

2. Train your model on the MNIST dataset. As the output is binary, formulate what should be its loss function. We strongly recommend you to use tensorboard in order to monitor your trainings (losses, reconstruction, latent space plotting).

3. Compare your model's outcomes to the VAE results from [8].

4. Implement the warm-up procedure for the ELBO (the regularization coefficient β varying from 0 to 1 [7]) during N epochs. What is it made for?.

5. Once the model has achieved a reasonable reconstruction quality, you can generate new data by sampling the latent space: decode random points, perform linear and/or spherical interpolations between two points.

Exercise 2: Build a simple GAN

As *Generative Adversarial Networks* (GAN) [6] are difficult to implement and train, you will start with the simpler adversarial digits generation problem using the MNIST dataset. You will start by implementing a simple convolutional GAN composed of a *generator* and a *discriminator*. Although you are free to propose your own architectures, we suggest that you draw inspiration from the DCGAN model [14] which will be used in the final timbre transfer model. First, take a look at the original paper and then implement it in `Pytorch`. To help you, you can rely on the tutorial

[4] `https://pytorch.org/tutorials/`
[5] `https://librosa.org/doc/latest/index.html`
[6] `https://pytorch.org/audio/stable/index.html`
[7] `https://github.com/andimarafioti/tifresi`
[8] `https://pytorch.org/docs/stable/tensorboard.html`

`https://bit.ly/3wuLAwS` (implemented in `Keras` otherwise it is too simple) which will give you a good intuition of the system.

1. Based on the tutorial or another source you will find, implement your very own GAN in Pytorch.
2. Train your model on the MNIST dataset using the Binary Cross Entropy (BCE) loss used in the orginal paper [6] and analyze the results.
3. Even though suitable for unsupervised learning problems, GANs often suffer from training instabilities mainly caused by the *vanishing gradient* and *mode collapse* issues. To address these, several training strategies have been proposed to replace the original BCE adversarial criterion such as the Least Square GAN (LSGAN) [11] approach used in the voice conversion model [1].
   Train your DCGAN model with the LSGAN loss on the MNIST dataset.
4. Compare both the original GAN and LSGAN training strategies in term of stability and sample quality.
5. BONUS - If you want, you can also compare your model with the other two well-known training strategies introduced in the Hinge GAN [9] and Wasserstein GAN [2] models.
6. Compare the results of your MNIST VAE and your MNIST GAN. What do you think are the advantages and disadvantages of each method ?

3 Spectral VAE

Keep your MNIST VAE aside and now adapt your VAE in order to learn the spectral content of classical instruments.

We will rely on the *NSynth*[9] dataset composed of one-shot instrumental notes. It contains 305979 musical notes with unique pitch, timbre and envelope. The sounds were collected from 1006 instruments from commercial sample libraries and are annotated based on their source (acoustic, electronic or synthetic). It consists of four-second monophonic 16kHz audio snippets. The NSynth dataset is usually used as a benchmark and entry point into audio machine learning. To keep it simple, we will only use the *string* instruments family which is composed of 20594 four-second audio samples in order to generate monophonic sounds.

Exercise 3: Data processing

A mandatory first step for adequate learning is to pre-process the data into known ranges and properties. Indeed, contrary to images where we directly give raw information to the model, we usually rely on specific representations. Here, we will focus on mel-spectrograms.

First, we extract the magnitude spectrograms from the raw waveforms. To do so, you can use a *Short-Time Fourier Transform* (STFT) or a *Discrete Gabor Tansform* (DGT). Then, we convert these spectrograms in the mel scale (using a mel filter bank) to obtain mel-spectrograms. To ease the training, we additionally constrain the range of values in the mel-spectrograms by applying a *log1p* transform ($f(x) = log(1 + x)$). We can then train our model on these representations. To retrieve the audio signals from the generated spectrograms, we must invert these transformations. To invert the spectrogram, you can use the *Griffin-Lim* algorithm or the *Phase Gradient Heap Integration* (PGHI) algorithm which is more efficient.

1. Based on the `torchaudio` and `tifresi` packages, implement the data processing transformations in a class `AudioTransform` with two methods:
   - `def forward(wav, **kwargs)`: compute mel-spectrograms from raw waveforms,
   - `def invert(spec, **kwargs)`: invert mel-spectrograms and return waveforms.
2. Implement the dataloader to train your model using the `AudioTransform` class.
   NB: Do not forget to normalize your spectrograms!

[9] `https://magenta.tensorflow.org/datasets/nsynth`

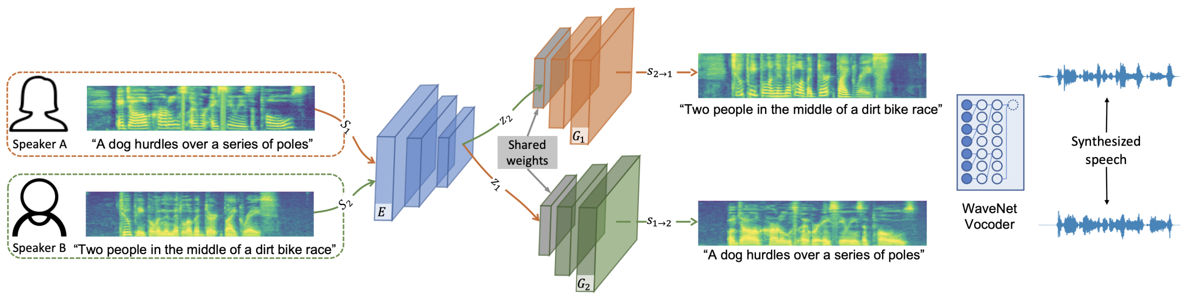
Exercise 4: Implement a Spectral VAE on mel-spectrograms

You will now adapt your VAE to generate mel-spectrograms using the *NSynth-string* dataset.

1. Design and train your Spectral VAE on the *NSynth-string* dataset, finding the architecture that provides the best reconstruction.
2. Experiment with latent space exploration: decode random points, preform linear and/or spherical interpolations between two points.

4 Timbre transfer model

You will now build the timbre transfer model on top of your Spectral VAE relying on the voice conversion approach [1], depicted in figure 2. The goal will be to perform timbre transfer between two different target domains of your choice. We will provide you with a large scale audio dataset composed of different instruments and voice recordings splitted into target domains. Among these, you will choose two target domains that you wish to use for the project.



Speaker A
"A dog hurdles over a series of poles"
Speaker B
"Two people in the middle of a dirt bike race"
s_1
s_2
E
z_2
z_1
Shared weights
$\mathcal{G}_1$
$\mathcal{G}_2$
$s_{2\to1}$
$s_{1\to2}$
"Two people in the middle of a dirt bike race"
"A dog hurdles over a series of poles"
WaveNet Vocoder
Synthesized speech

Figure 2: The overall procedure of the voice conversion model proposed by AlBadawy et al. [1]. The *encoder* $\mathcal{E}$ generates features embedding for a given input speaker. Then, one of the *generators* ($\mathcal{G}_1$ or $\mathcal{G}_2$) decodes the latent representation in order to produce a mel-spectrogram output for the target speaker. Eventually, a WaveNet-based vocoder is used to generate the speech in the time domain.

Exercise 5: two-domains timbre transfer

First, take a look at the main article [1] to understand the training strategy and each loss functions. The overall objective loss function $\mathcal{L}$ is composed of four individual loss functions such that

$$\mathcal{L} = \mathcal{L}_{VAE} + \mathcal{L}_{GAN} + \mathcal{L}_{CC} + \mathcal{L}_{latent}. \tag{1}$$

We will build the model gradually by focusing, firstly, on the combination of spectral VAEs and GANs for two domains and, secondly, on the regularization of the latent space to ensure the same latent representation is shared between the two domains.

1. Update your dataloader so that it returns the spectrograms along with its source domain for each batch. Extend your spectral VAE by creating one *decoder* for each target domain that will be refered to as *generators* $\mathcal{G}_1$ or $\mathcal{G}_2$.
2. Using your DCGAN model from *Exercise 2* with the LSGAN loss formulation, add the *discriminators* $\mathcal{D}_1$ or $\mathcal{D}_2$ for each target domain. Update the loss of your model so that it optimizes the following objective loss: $\mathcal{L} = \mathcal{L}_{VAE} + \mathcal{L}_{GAN}$. Train your model and see how it performs.
3. Based on the voice conversion [1] and the CycleGAN [15] papers, implement the *cycle consistency loss* $\mathcal{L}_{CC}$.
4. Based on the voice conversion paper [1], implement the latent loss $\mathcal{L}_{latent}$.
5. Train your model with the complete objective loss defined in equation 1.

6. BONUS - Improve the architecture of your model and add the residual blocks mentioned in the paper [1] to the *encoder* and each *generator*. The generators share the first residual block as a pre-processing step of the latent vector $\mathbf{z}$. Train your model with this new architecture.
7. BONUS - Evaluate the performance of your model using the *Frechet Audio Distance* (FAD).

(BONUS) - Exercise 6: multi-domains timbre transfer

Generalize your model so that it can arbitrarily take n target domains. Compare the quality of the generated samples and analyze/explore the latent space when increasing the number of generators.

Exercise 7: Play with your model

Choose audio samples from different source domains and, encode and decode these into each target domain. Generate new samples by sampling the latent space. Explore the latent space through interpolations. Select and save some generated examples. (You can also create a github page if you want!)

References

[1] Ehab A AlBadawy and Siwei Lyu. Voice conversion using speech-to-speech neuro-style transfer. In *Interspeech*, pages 4726–4730, 2020.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. arxiv e-prints, page. *arXiv preprint arXiv:1701.07875*, 1(2):3, 2017.

[3] Antoine Caillon and Philippe Esling. Rave: A variational autoencoder for fast and high-quality neural audio synthesis. *arXiv preprint arXiv:2111.05011*, 2021.

[4] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *arXiv preprint arXiv:1902.08710*, 2019.

[5] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. Ddsp: Differentiable digital signal processing. *arXiv preprint arXiv:2001.04643*, 2020.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[7] Irina Higgins, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew M Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[9] Jae Hyun Lim and Jong Chul Ye. Geometric gan. *arXiv preprint arXiv:1705.02894*, 2017.

[10] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. *Advances in neural information processing systems*, 30, 2017.

[11] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[12] Noam Mor, Lior Wolf, Adam Polyak, and Yaniv Taigman. A universal music translation network. *arXiv preprint arXiv:1805.07848*, 2018.

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

[14] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[15] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.