Timbre Transfer and Interpolation Using a Conditional Convolutional beta-Variational Autoencoder

Tesi di Laurea Magistrale in
Music and Acoustic Engineering

Author: **Silvio Pol**

Student ID: 941793
Advisor: Prof. Massimiliano Zanoni
Co-advisors: Luca Comanducci
Academic Year: 2021-22

Abstract

Timbre Transfer techniques may find application in several different scenarios, particularly in music production environments. Having a tool that takes as input a signal of a recorded instrument and that gives as output the same recording but with a new timbre could be helpful to music producers.

In this thesis, after giving an overview of the existing techniques and methodologies that follow this goal, we propose a method which can effectively create a timbre space which permits to operate one to many timbre transfer. We do this by training a conditional convolutional beta-VAE architecture on a subset of the NSYNTH dataset build by us. The system takes as input a spectrogram of a note with a given timbre, namely its time-frequency representation, and outputs multiple spectrograms of the same note with different timbres. It does that by constructing a navigable conditioned latent space representation of timbres and automatically encoding the pitch information. Given the possibility to move inside the latent space, we perform timbre interpolation, namely the morphing between two timbres, generating new samples that go from a starting timbre to an ending one, exploring the timbral space between them. We evaluate our system from different perspectives. In particular, we establish a twofold evaluation system based on classification and perceptual ratings. The experimental results show that the model is capable of performing the timbre transfer task having the generated samples that match the ground truth ones and that the conditioned latent space creates automatically clusters based on timbre and pitch, giving the possibility to perform timbre interpolation by moving inside it.

Keywords: Deep Learning, Timbre Transfer, Timbre Interpolation, Variational Autoencoder, Latent Space, NSYNTH

Abstract in lingua italiana

Le tecniche di trasferimento di timbro possono trovare applicazione in diversi scenari, particolarmente negli ambienti di produzione musicale. Avere un sistema che prende in input un segnale di uno strumento registrato e che dà in output lo stesso segnale ma con un nuovo timbro può rivelarsi vantaggioso per i produttori musicali.

In questa tesi, dopo aver dato una panoramica sulle tecniche e metodologie esistenti che perseguono questo scopo, proponiamo un metodo che può efficacemente creare uno spazio timbrico che permette di realizzare trasferimento di timbro uno a molti. Lo facciamo allenando un'architettura beta-VAE convoluzionale e condizionata su un sottoinsieme del dataset NSYNTH costruito da noi. Il sistema prende in input uno spettrogramma di una nota con un certo timbro, ovvero la sua rappresentazione tempo-frequenza, e restituisce multipli spettrogrammi della stessa nota con timbri diversi. Lo fa costruendo uno spazio latente condizionato navigabile di timbri e codificando automaticamente l'informazione legata all'intonazione. Data la possibilità di muoversi all'interno dello spazio latente, realizziamo interpolazione timbrica, ovvero il passaggio tra due timbri, generando nuovi campioni che vanno da un timbro iniziale ad un timbro finale, esplorando lo spazio tra loro. Valutiamo il nostro sistema secondo molteplici prospettive. In particolare, abbiamo stabilito un duplice sistema di valutazione basato sulla classificazione e su test percettivi. I risultati evidenziano che il modello è in grado di svolgere il trasferimento di timbro avendo i campioni generati che coincidono con quelli reali e che lo spazio latente condizionato crea automaticamente dei cluster basati sul timbro e sul pitch, dando la possibilità di realizzare interpolazione di timbro muovendosi all'interno di esso.

Parole chiave: Deep Learning, Trasferimento di Timbro, Interpolazione di Timbro, Variational Autoencoder, Spazio Latente, NSYNTH

Contents

1 | Introduction

Artificial Intelligence (AI) has made impressive progress, thanks to the advent of Machine Learning (ML) and Deep Learning (DL) techniques, affecting almost every field, one among them being music. Being an art form, music cannot be 100% modeled by Deep Learning algorithms, however, astonishing results has been achieved. One of the challenges in the research area of modelling music is the one related to the concept of timbre. With the term "timbre", we refer to the perceptual qualities of a musical sound distinct from its amplitude and pitch [8].

Modeling timbre is a hard task: it is a difficult job to define a physical or mathematical model of timbre since it is a perceptual and subjective characteristic of sound. In the field of music production, for example, most of state-of-the-art musical sound libraries used by studio composers are still obtained with high quality records of real instruments. However, building a music sound library with this methodology can be a very expensive and time consuming task. For this reason, there is in fact a substantial body of research in timbre modelling and synthesis. A particular sub-field of research is the one regarding timbre transfer.

With "timbre transfer" is intended the task of, given an input signal, generate a new signal that maintains every characteristic of the input one, except for its timbre. Classic approaches [13, 24, 37, 39, 46, 51, 60] resort to the use of generative Deep Learning models such as Generative Adversarial Networks (GANs) [18] or Variational Autoencoders (VAEs) [32] or on the combination of Signal Processing techniques and Deep Learning architectures such as the DDSP framework [12].

GANs and VAEs are 2 typologies of Generative Deep Learning models. GANs are composed by two neural networks called *Generator* and *Discriminator*, one contesting the other in a zero-sum game. The *Generator* generates new data while the *Discriminator* classifies the output of the *Generator* as real or fake. VAEs are based on a usually symmetrical multi-layer network that compress input data into a low dimension latent space and then re-construct it inverting the compression procedure.

TimbreTron [24] is one of the first example of architectures performing high quality timbre transfer using Deep Learning models, inspired by Neural Style Transfer successes

[14, 29, 61] and built upon a GAN framework. An example of timbre transfer on percussive patterns extracted from polyphonic audio using a MelGAN-VC model [42] is given in [37] where the authors successfully managed to transfer the drum style from hip-hop to metal. In [60], timbre transfer is performed with a Variational Autoencoder, allowing composers to synthesize sounds using a latent space of audio that is constrained to the timbre space of the audio recordings in the training set. In [39] the focus is on learning disentangled representations of timbre and pitch exploiting a Gaussian Mixture Variational Autoencoder (GMVAE [10, 27, 33]) architecture.
In this thesis, after giving a deeper overview of the existing techniques and methodologies that pursue this goal, we propose a method which can effectively create a timbre space that permits to operate one to many timbre transfer i.e. the task of, given a signal characterized by its timbre, creating a system that outputs different versions of the signal with everything unchanged except for the timbre. We do this by training a conditional convolutional beta-variational autoencoder architecture on a subset of the NSYNTH dataset [11] built by us. Beta-VAE [23] is an extension of the VAE architecture that introduces an hyper-parameter which permits a better regularization of the latent space. By conditioning the input, we manage to build a regularized latent space, required to perform the timbre transfer task.
The system takes as input the module of the Short Time Fourier Transform, also known as spectrogram, of a signal with a given timbre associated to a specific note and outputs multiple spectrograms of the same note with different timbres. For this reason we opted for a convolutional network, suitable for use with image-like inputs. It does that by constructing a navigable conditioned latent space representation of timbres and automatically encoding the pitch information. Given the possibility to move inside the latent space, we extend the capabilities of the system by performing timbre interpolation, namely the morphing between two timbres, generating new samples that go from a starting timbre to an ending one, exploring the timbral space between them.
We evaluate our system from different perspectives. In particular, we establish a twofold evaluation system based on classification and perceptual ratings. The experimental results show that the model is capable of performing the timbre transfer task having the generated samples that match the ground truth ones and that the conditioned latent space creates automatically clusters based on timbre and pitch, giving the possibility to perform timbre interpolation by moving inside it.

This thesis is organized as follows. In Chapter 2 we provide a compact theoretical background of the main paradigms that are at the base of this research. We start with an overview on time-frequency representations of audio signals, then we recap Machine

Learning and Deep Learning techniques and architectures in order to help the reader with the comprehension of the subsequent parts of the thesis.
In Chapter 3 we expose the main state-of-the-art works concerning the timbre transfer and the timbre interpolation problems. We divide the chapter in sections associated to the architectures used in those works, firstly we present the Generative Adversarial Network (GAN) based works, then the Variational Autoencoder (VAE) based works and finally we present briefly the DDSP architecture that combines Deep Leaning and Signal Processing techniques.
In Chapter 4 we first formalize the timbre transfer and timbre interpolation tasks. After that we expose the system architecture of our choice, outlining and justifying our choices. We divide the subsection into Pre-Processing, Network Architecture and Post-Processing following the pipeline. Finally we dedicate a chapter for the modelling of interpolation we used.
In the former part of Chapter 5 we delineate our experimental setup along with a description of the dataset that we used for the task and we present the twofold evaluation method we adopted to asses the outcomes of the network. In the latter, we present the results of our experiments dividing them by task.
Finally, Chapter 6 is dedicated to conclusion and to possible future extension of this work.

2 | Theoretical Background

In this chapter we will present how a music signal can be represented in order to be interpreted by machines. Two domains come into play when talking about audio signals' informatic representation: time and frequency. Normally, in almost any audio representation we see in audio-related technologies such as SoundCloud or Whatsapp's vocal messages, only the time-domain signal is represented. What we see is the **soundwave**, the pressure wave generated by a signal, while the frequency information is discarded. In 1882, Jean Baptiste Joseph Fourier developed the now called **Fourier Transform** (FT) that links the time domain of a signal with the frequency domain. Since then, the Fourier Transform has been studied but most of all adapted for the digital domain: it is known as Discrete Fourier Transform (DFT) and it's roughly a sampled and quantized version of the Fourier Transform. This discretization of the Fourier Transform is indispensable since computers are, obviously, digital devices.

2.1. Short Time Fourier Transform

Thanks to the FT and the DFT, we can represent signals in the time and frequency domains. It is advantageous, by the way, to have a compact representation where we can see both domains: we want to see how the frequency components of the signal vary along time. This representation consists in an image with time progression on the x-axis and frequency evolution on the y-axis (or viceversa).

The Short Time Fourier Transform (STFT) is the most common time-frequency representation of audio signals. It consists of the calculation of the Discrete Fourier Transform (DFT) along windowed, fixed length portion of the time signal. Sliding a window function $w[h]$ along a given signal $x[k]$ with length N at every R time frames we obtain a series of windowed portions of the signals: $x_w[k] = h[k]x[k + tR],\ 0 \leq n \leq N - 1$ with $h, k,\ R \in \mathbb{N}$. The constant R is known as *hop size* or *hop length*. The window could be any finite signal used to chop the longer signal $x[n]$, although choosing a good $h[k]$ is crucial for the outcome of the time-frequency representation of the signal we want to obtain since there are many types of windows which serve different purposes and exhibit various properties.

The main properties of windows are observable when performing the FT, so investigating the windows themselves in the frequency domain. Three major properties are the Peak Sidelobe Amplitude, the width of the main lobe and the Roll off, each one impacting in a different way on the windowing operation [20]. Performing the DFT for each one of the windowed portions and "stacking" the results on a time axis, we end up with a 2-D representation of the signal, defined as:

$$X[t,f] = \sum_{k=0}^{N-1} x_w[k]e^{-j(2\pi/N)fk}, \qquad f = 0, ..., N-1, \tag{2.1}$$

having t representing time bins (or samples), f representing frequency bins and j the imaginary unit.
The result $X[t,f]$ of equation 2.1 is complex-valued and usually divided in *phase* and *magnitude* representation of the STFT, defined as $\angle X[t,f]$ and $|X[t,f]|$. The magnitude component of the STFT can finally be defined as a **spectrogram**. Squaring the magnitude spectrogram we obtain the *power spectrogram*, the most common representation used in computer science's studies.

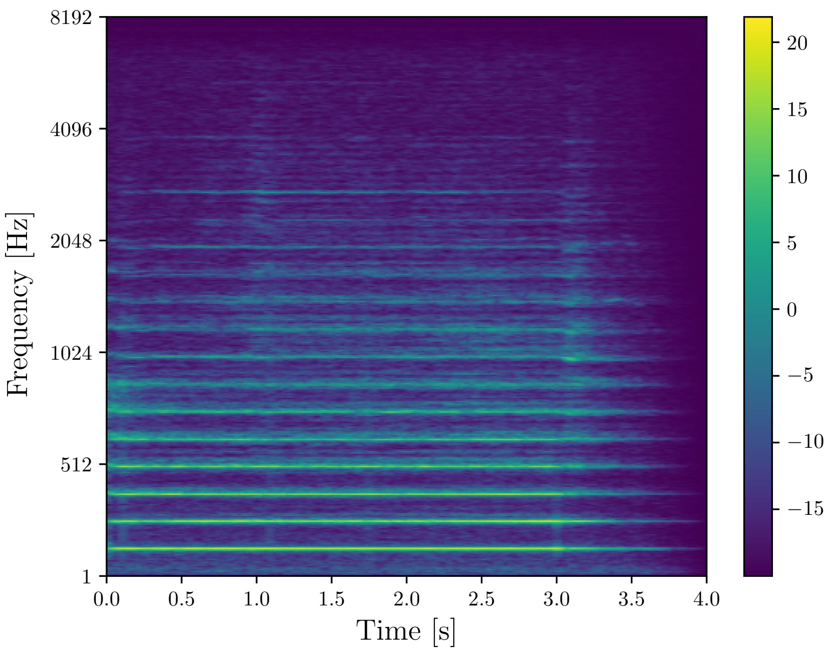


Figure 2.1: Example of Power Spectrogram.

2.2. Contant-Q Transform

The cochlea is the component of our inner ear responsible for converting vibrations into electrical signals to be delivered and processed by the brain. It is a spiral-shaped cavity with ≈ 30.000 hair cells which responds to audio frequency in a logarithmic fashion: resonance frequencies doubles every 3.5 mm. This observation leads to *perceptually motivated representations* of audio signals such as the Constant Q Transform (CQT). Instead of using a uniform frequency spacing like in the STFT, frequency bins are now distributed geometrically. This permits a representation of frequency that follows the equal temperament notation. Thus, a constant ratio Q between the central frequency of a band f_k, and the frequency resolution $(f_k - f_{k-1})$ for that band, is obtained with:

$$f_k = f_0 \cdot 2^{\frac{k}{b}}, \tag{2.2}$$

with f_0 as the first band central frequency and b the number of frequency bins per octave.

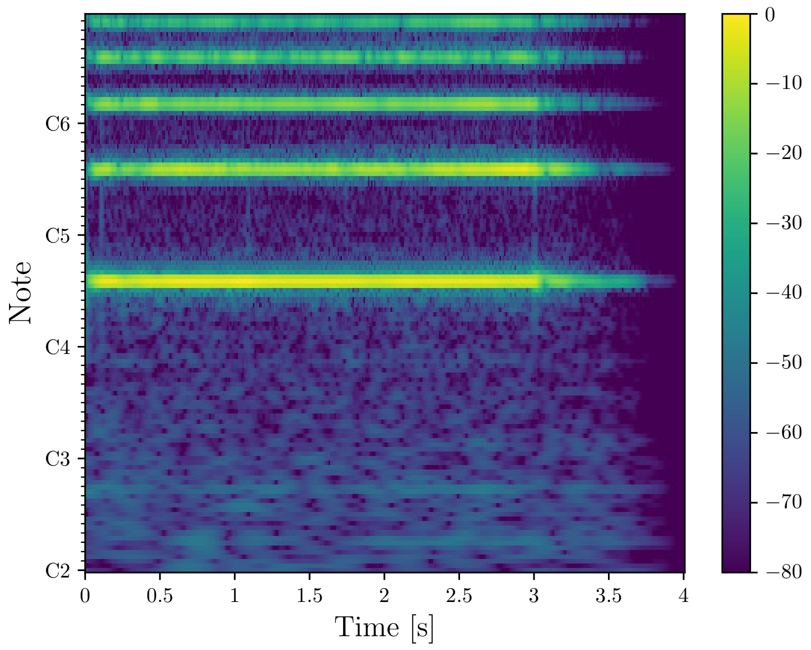


Figure 2.2: Example of Constant Q Transform. The frequency axis is now marked with notes instead of frequencies.

Then, the CQT can be defined as:

$$X[t,f] = \frac{1}{N[f]} \sum_{n=0}^{N[f]-1} W[f,k]x[k]e^{\frac{-j2\pi Qk}{N[f]}}, \qquad Q = \frac{f_k}{\delta f_k}, \;\; N[f] = Q\frac{f_s}{f_k}. \tag{2.3}$$

2.3. Mel Spectrogram

Mel-spectrograms are also a perceptually-based representation of audio signals. Differently from the CQT, the Mel Spectrogram follow another definition for the frequency axis based on the *Mel scale* that corresponds to an approximation of the psychological sensation of heights of a pure sinusoid.

Since the Mel scale is based on psychoacoustic studies and relies on the outcome of listening experiments, it has several definitions. One of them is the one proposed by Stanley Smith Stevens, John Volkman and Edwin Newman [59]:

$$mel(f) = \begin{cases} f, & f < 1000\text{Hz} \\ 2595 \cdot \log(1 + \frac{f}{700}), & f \geq 1000\text{Hz}. \end{cases} \tag{2.4}$$

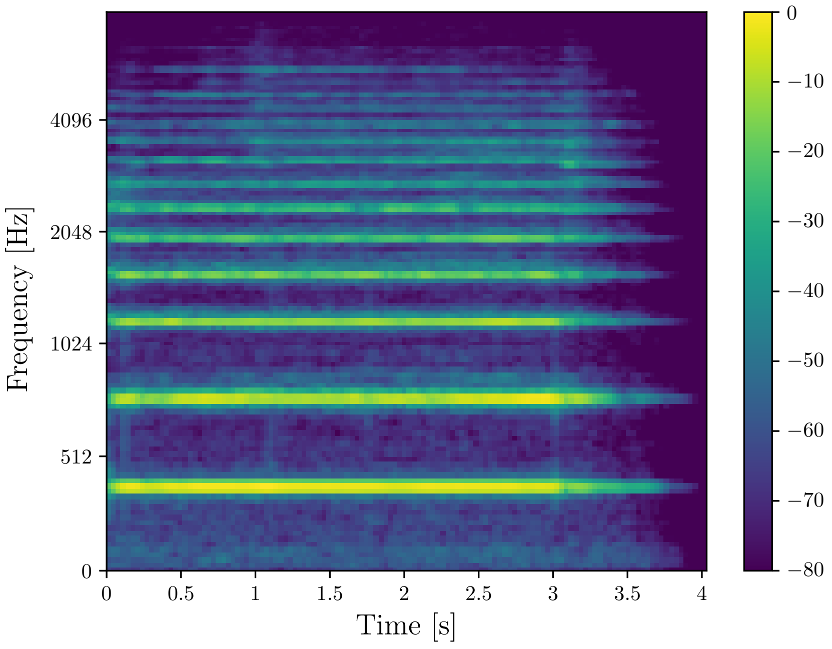


Figure 2.3: Example of Mel Spectrogram.

In practice, to compute the mel spectrogram of a signal, we compute the STFT and multiply the linear frequency axis by a bank of $N_{mel} \in \mathbb{N}$ triangular, 50% overlapping filters spaced apart uniformly one from another following the mel scale.

2.4. Machine Learning

Machine learning (ML) is a field of Artificial Intelligence based on the usage of statistics and algorithms with the aim of making machines learn structures and models from examples of data in an automatic fashion. The concept of *experience* is the crucial point of machine learning: with many data samples given as input, it is possible to extract a statistical model based on features, that could lead to a meaningful learning by the machine.

There are plenty tasks related to ML in audio and acoustic such as speech recognition, genre classification, automatic transcription, automatic music composition, etc.

The two main ML algorithmic paradigms are $classification$ and $regression$. Classification assigns each input to one of a given number of discrete categories. An example of classification task could be, given a set of classes of animals, classifying if the subject of an image is a cat or a dog or any of the animals of the classes. Regression, instead, finds a relationship between input variables with the intent of predicting a continuous value based on past observations. An example of regression is house pricing: we want to predict the price of a house given a set of its features, like location, size, etc.

The process of learning the statistical model through the analysis of the data constituting the dataset is called *training*.
In order to build a powerful model, it is very important to rely the training phase on high quality data that should be meaningful to solve the task. The optimal deployment situation is achieved when the data used to train and develop the model matches with the actual data encountered at the deployment stage.

Depending on the structure of the dataset but also on the type of task we want to solve, we have two macro categories of learning processes:

- **Supervised Learning:** when we have a *labeled* dataset. In the training phase, we give the model in input the data and its *label*, namely a value associated to the defining characteristic of each sample. For example, recovering the classification case suggested before, we train the model on images of animals whose labels will be "cat", "dog", "frog" or, more often, a numerical value associated to them such as "0", "1", "2" and so on. With such a labeled dataset, the computer will be able to associate the features of the images to the labels related to them, so when a new unknown input is presented to the system it will be able to associate its features to a class of animals and determine what animal is shown in the new image;

- **Unsupervised Learning:** when we have an *unlabeled* dataset. The model in this case can rely only of the features representing the data and from them it can learn useful, unknown representations. Unsupervised learning can be used to cluster the input data in classes on the bases of their statistical properties only or to build generative models, finding the sub-structure of data understanding how it is generated in terms of a probabilistic model.

Usually, the dataset is partitioned into three subsets, namely:

- **Training set:** the only part of the dataset used to actually learn the I/O mapping from data. The training set is the bigger partition so that the model can gather more information and generalize better;
- **Validation set:** a smaller set on which we monitor how the model performs while training;
- **Test set:** set used to evaluate the model performances after the end of the training phase. It is used to simulate a real-world application of the algorithm.

2.5. Deep Learning

Deep Learning (DL) is a branch of ML being now the state of the art for most of research and application tasks in scientific fields. DL is based on *layers* that embed increasingly meaningful representation of input data. This mechanism could be seen as an automatic feature extraction process: every layer will be responsible for modeling a hierarchical representation of input data with deeper layers describing higher-level features and the first ones describing lower-level features. And therein lies the power of deep learning: the layer's outputs do not necessarily make sense for humans but they do from the machine point of view. The layers and their connections constitute what we call *Artifical Neural Networks (ANN)*, so called because inspired by the structure of the human brain. Every layer in an ANN is characterised by a number of computing units called *neurons*. A neuron is a cell containing a real number. The most basic architecture using a single neuron is the *perceptron* [49] depicted in figure 2.4.

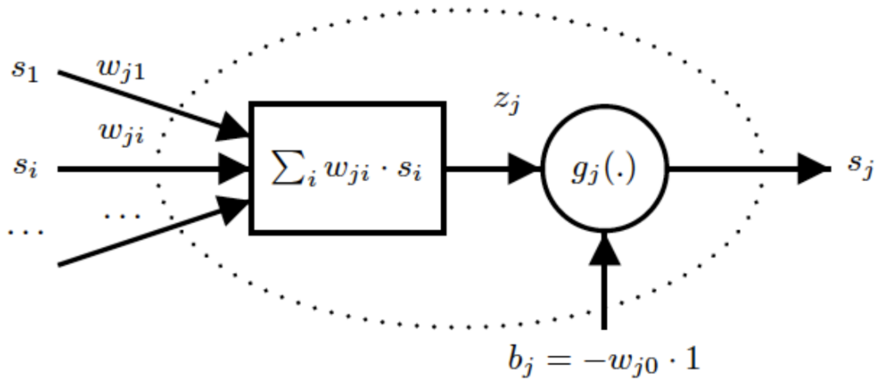


Figure 2.4: Graphical representation of the Perceptron.

The perceptron follows the feed-forward model: input are sent into the neuron, processed and result in an output. Given a generic neuron j and a set of N inputs s_i, the output s_j of the perceptron will be:

$$s_j = g_j(\sum_{i=1}^{N} w_{ij} s_i - b_j). \tag{2.5}$$

In equation 2.5 we introduce three unknown elements:

- g: is the *activation function*;
- $w \in \mathbb{R}$: is the *synaptic weight*;
- $b \in \mathbb{R}$: is the activation threshold or *bias*.

For a given perceptron, for every input s_i we have a weight w_{ij} that will be multiplied by the input itself, and one bias that will be subtracted from the sum of every of the input-weight multiplication just described.

The activation function g takes as input that and outputs a real number s_j. There are plenty activation functions to be used depending on the application. The most commonly used ones are:

- **Sigmoid:**

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.6}$$

- **Rectified Linear Unit (ReLU):**

$$R(x) = max(x, 0), \tag{2.7}$$

- **Hyperbolic Tangent:**

$$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{2.8}$$

- **Logistic:**

$$Log(x) = (1 + e^{-x})^{-}1, \tag{2.9}$$

- **Linear:**

$$L(x) = x, \tag{2.10}$$

with $x \in \mathbb{R}$. All the activation functions just presented, except for the Linear activation function, are non linear functions. This is essential for the network in order to learn non-linear structures in data when needed.

As in ML, the learning stage of the model is called training phase. During this phase, the whole training set is given as input to the network multiple times called *epochs*. Several epochs may be needed for a network to learn, depending obviously on the application. The aim of the network is now to build a statistical model based on the values of the weights **w** and biases **b** on which the network itself is constructed. The parameters of the network are updated after every epoch, minimizing a *loss function* $L(\hat{y}, y|w, b)$, basically the difference between ground truth outcome of the network ($\hat{y}$) and the real outcome (y) after a given epoch. The process of updating the parameters of the network is performed by *backpropagation* [21, 50]: the gradient of the loss function is computed and the error propagated back through the layers with the aim of finding the best arrangement of values of the network to have the best possible accuracy for the output.

One of the biggest issues we encounter dealing with ML and DL algorithms is the one of *overfitting*. This happens when we the performances of the model achieved in a test phase of the model aren't as good as the one achieved in the training phase. In other words, the model has learnt too well how to perform on the training set, focusing only on that specific assortment, but when challenged to perform on new, unseen, data it is not capable to have the same results. Opposed to that, the ability of the model to have comparable performance in training and testing is called *generalization*. Overfitting can be caused from various factors, one of the most common is the use of a network that has too many parameters. In that case we have too many values that will model the statistical structure of the training set by the network causing an over-adaptation, leading, in fact, to overfit. In this case, the solution to the problem could be quite straight forward: implement a simpler architecture. Other techniques have been discovered during the years to deal with overfitting, among them:

- **Dropout**: [57]: the idea of dropout is to randomly drop a certain percentage of neurons, along with their connections, of the network during the training phase. By doing that we ensure that the model will not over-adapt on the training set;
- **Early Stopping** [45]: this technique is based on the simple idea of stopping the epochs and save the model as soon as the validation loss stops decreasing (meaning that the network has finished learning and is starting to over adapt on new data). This is actually done after a predefined number of epochs after the validation loss does not improve with respect to the best recorded value;
- **Data Augmentation** [9]: performing data augmentation means to extend the size of the training set using and modifying its samples and adding these new examples to the training set. In audio dataset this could be done by performing time stretch or pitch shifting the audio file. Another possibility is to add noise to some samples;
- **Batch Normalization** [25]: this technique is applied to layers and performs normalization of the output for each training mini-batch. By doing this, we solve the problem of internal covariance shift also allowing faster learning rates;
- **Regularization** [17]: when we have an excessively complex model, we want, in a sense, penalize the flexibility of the model so it doesn't adapt too much to the training set. We can do it adding a regularization term to the loss function with the specific aim of penalizing flexibility. The loss function becomes:

  $$L_R = L + \gamma L_W \tag{2.11}$$

  with γ known as *regularization term* controlling how much the factor L_W, the regularization factor, impacts the total loss. There are different regularization factor, among them the *Lasso* and the *Ridge Regression*, with the aim of reducing the values of the parameters toward zero, in fact lowering down the model complexity.

2.5.1. MLP

When perceptrons are organised in layers, the feed-forward ANN connecting them is called *Multi Layer Perceptron*. When all neurons are connected with other being part of the previous and following layers, we have a *Fully Connected Neural Network* as shown in Fig. 2.5.

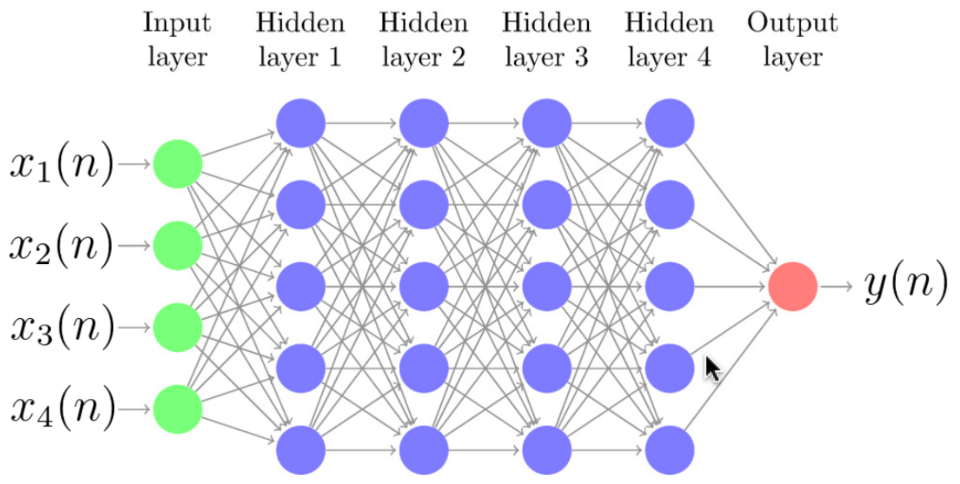


Figure 2.5: Fully Connected MLP network with input, four hidden layers with five neurons each and a mono-dimensional output layer. Image taken from [43]

.

The first layer of the network is called *input layer*, the last one *output layer* and those in between, if present, *hidden layers.*

2.5.2. CNN

Convolutional Neural Networks (CNNs) are feed forward neural networks designed to work with 2D input data (i.e. matrices) such as images. There are also 1D and multidimensional CNNs but we will present the 2D case since it is the one deployed in this work.

In audio engineering 2D CNNs are suited when we have time-frequency representation of signals like spectrograms. The architecture and the concept behind a CNN are similar to the ones of MLP: the network is still built on input, hidden and output layers. The difference is that the network is now working with matrices instead of vectors and performing convolution operations between input data and filters instead of multiplications between the output of the neurons. The convolution operation on a CNN can be formulated as:

$$y(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m)(j-n), \tag{2.12}$$

where $y(i,j)$ is the feature map, I and K are respectively the input and the filter matrices. We have three different types of layers forming the convolutional neural networks: *Convolutional Layers, Pooling Layers* and *Fully Connected Layers.*

Convolutional layers are responsible for the convolution operation that is performed between the input and the set of filters (whose parameters represent the learnable parameters of the network) called *kernels*. Each filter is convolved with the input along the x and y axes, ideally capturing a certain pattern of the input. These operations are performed following a certain configuration of parameters. The most important ones are the filter size, the depth that indicates the number of filter used in the layer and the horizontal and vertical stride that represent how far the filter moves from one position to the next position. We can imagine the filters being able to detect more complex pattern as we go deeper into the network: the first layers are able to detect general patterns and the deeper ones are ideally capable of detecting the details.

Pooling layers implement subsampling of a certain output of a convolutional layer so that deeper layers effectively integrate larger extent of data. There are different non linear functions performing different kinds of downsampling, the most common are *max pooling* (retaining only the max of a certain patch) and *average pooling* (computing the average).

Fully Connected Layers are the layers explained in section 2.5.1 when describing MLP. One fully connected layer is usually put at the end of the architecture to compute the output.

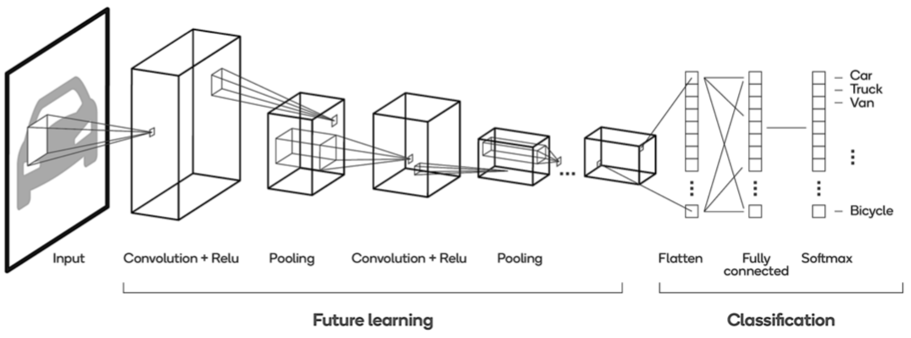


Figure 2.6: Example of CNN network used for classification

2.5.3. Autoencoders

Deep learning is not limited to classification and regression tasks: another important DL application regards the generation of new data created by exploiting*generative models*. Autoencoders, implementation of the classical encoder-decoder architecture, are one of the main models able to generate new data. Actually, the goal of the autoencoder is to reconstruct exactly the input data by getting its compression in the middle layer.

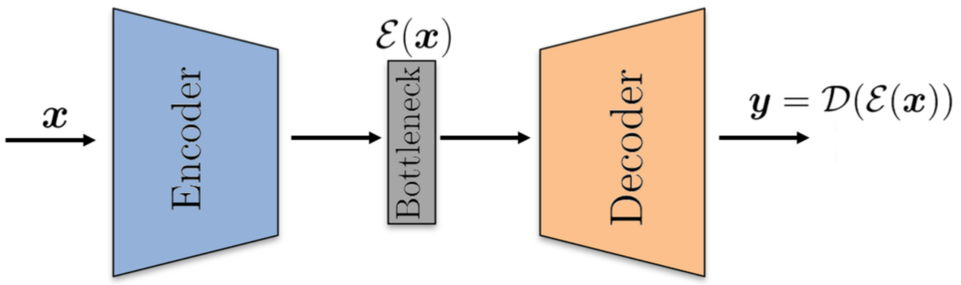

$\boldsymbol{x}$
Encoder
$\mathcal{E}(\boldsymbol{x})$
Bottleneck
Decoder
$\boldsymbol{y} = \mathcal{D}(\mathcal{E}(\boldsymbol{x}))$

Figure 2.7: Generic Autoencoder structure

The Autoencoder (AE) is based on the concept of reducing the dimensionality of a certain input and decode that compressed representation with the target of reconstructing the input data, all exploiting the neural network architecture. We have two main blocks constituting an autoencoder: the *Encoder* $\mathcal{E}$ and the *Decoder* $\mathcal{D}$, each one implemented with Fully Connected layers or Convolutional Layers, depending on the application. The role of the encoder is to encode the input $\mathbf{x}$ into a lower dimensional space called *latent space* while the role of the decoder is to take this latent space representation of $\mathbf{x}$ and reconstruct it in order to have $\hat{\mathbf{x}}$ that should mimic the input $\mathbf{x}$. Ideally, calling E and D the operations performed respectively from the encoder and the decoder $\hat{\mathbf{x}} = D(E(\mathbf{x})) = \mathbf{x}$. The learning process is carried once again by backpropagation, reducing a loss function also known as *reconstruction error* that consists on the difference (usually MSE) between the input and the output of the network. Autoencoders in their standard formulation are an example of unsupervised learning, since the network has only the aim of reconstructing the input regardless its semantic description. There are also examples of semi-supervised learning autoencoders whose latent space is regularized with labelling the input given to the network.

The latent space of the autoencoder is a powerful representation of the dataset: every input data is represented as a point in the latent space but sampling other points in that space and decoding them can lead to the generation of new, different data, still based on the characteristics of the input dataset.

2.5.4. Variational Autoencoders

Variational Autoencoders (VAEs) [34] are an extension of Autoencoders in which each sample of the input data is mapped into a Gaussian probability distribution rather than into a simple point in the latent space. The architecture is still based on *Encoder* and *Decoder* networks working together. Given $\mathbf{x}$ as input sample, the encoder network is designed to generate a set of parameters for the distribution $q(\mathbf{z}|\mathbf{x})$ with $\mathbf{z}$ defined as a latent space variable, i.e. how the input $\mathbf{x}$ is represented at the end of the encoder network. The decoder, on the other hand, generates a set of parameters for $p(\mathbf{x}|\mathbf{z})$, that represent the conditional distribution of the input $\mathbf{x}$ given the latent space variable $\mathbf{z}$. The objective of the VAE architecture is the one of learning the parameters of the network so that the encoder distribution $q(\mathbf{z}|\mathbf{x})$ becomes consistent with the posterior $p(\mathbf{z}|\mathbf{x})$. We can infer the posterior $p(\mathbf{z}|\mathbf{x})$ by applying the Bayes theorem, knowing that $p(\mathbf{z}|\mathbf{x}) \propto p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$. To model $q(\mathbf{z}|\mathbf{x})$, $p(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z})$, we can assume them as Gaussian distributions:

- $q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}(\mathbf{x}), diag(\boldsymbol{\sigma}^2(\mathbf{x})))$,
- $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}(\mathbf{z}), diag(\boldsymbol{\sigma}^2(\mathbf{z})))$,
- $p(z) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$,

with $\boldsymbol{\mu}(\mathbf{x})$, $\boldsymbol{\sigma}^2(\mathbf{x})$ the output of the encoder network and $\boldsymbol{\mu}(\mathbf{z})$, $\boldsymbol{\sigma}^2(\mathbf{z})$ the outputs of the decoder network. Therefore, the encoder is now trained to return a mean $\boldsymbol{\mu}(\mathbf{x})$ and a variance vector $\boldsymbol{\sigma}^2(\mathbf{x})$ for each input sample and in order to have a more regular latent space, the distributions returned by the encoder are forced to be close to *normal distributions*. For this reason, the loss function needs an additional term that minimized the difference between $q(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{z})$, beside the reconstruction error, that will push the two distributions to be consistent and will push the distribution calculated from the input data to have mean $= 0$ and standard deviation $= 1$. This term is commonly referred as *KL loss* since it is based on the Kullback-Leibler divergence. The "full" loss function of a VAE can be formulated as:

$$LOSS = L_R + L_{KL} = ||\hat{\mathbf{y}} - \mathbf{y}|| + KL[q(\mathbf{z}|\mathbf{x}), p(\mathbf{x}|\mathbf{z})], \tag{2.13}$$

where L_R is defined as *Reconstruction Loss*, namely the difference among the output of the decoder $\hat{\mathbf{y}}$ and the ground truth output $\mathbf{y}$, most of the time based on Mean Square Error (MSE) or cross entropy.

The regularization term on the loss function ensures two properties:

1. **continuity:** two close points in the latent space should not give completely different contents when decoded, as could happen in a non-variational Autoencoder;
2. **completeness:** for a chosen distribution, a point sampled from the latent space should give always meaningful contents.

This regularization prevent the model to encode data far apart in the latent space and encourage the returned distributions to overlap.

The decoder will sample a point from that Gaussian distribution and reconstruct the input by using the reparametrization trick [34]. Defining the reparametrization as:

$$\mathbf{z} = \boldsymbol{\mu}(\mathbf{x}) + \boldsymbol{\sigma}(x) \odot \boldsymbol{\epsilon}, \tag{2.14}$$

having $\boldsymbol{\epsilon} \approx \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ and with $\odot$ indicating the component-wise multiplication, it is possible to replace the sampling operation from $q(\mathbf{z}|\mathbf{x})$ to sampling $\boldsymbol{\epsilon}$. This renders the computation of the gradient possible, allowing a gradient path through a non-stochastic node.

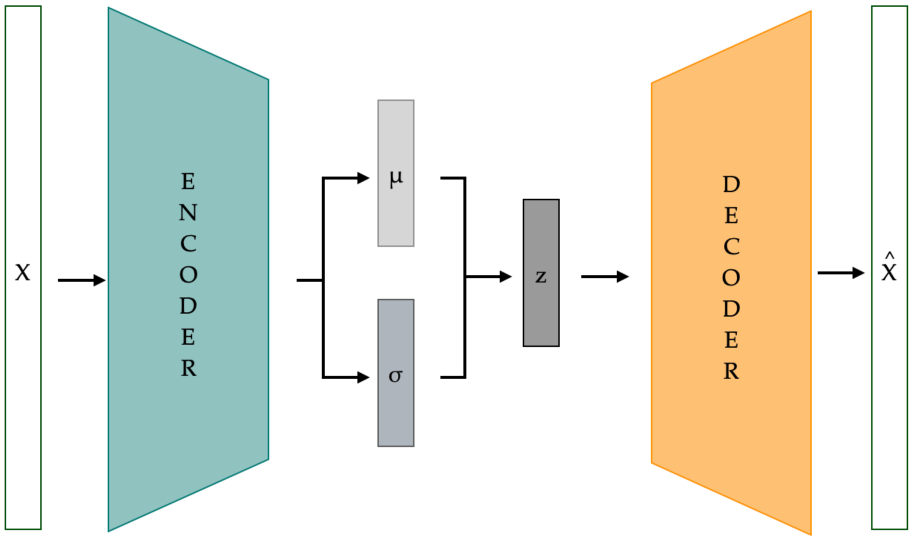


Figure 2.8: Generic Variational Autoencoder structure

2.5.5. Generative Adversarial Networks

Generative Adversarial Networks (GANs) [18] are another big family of generative models based on Deep Learning. A GAN is composed by two neural networks called *Generator* and *Discriminator*, one contesting the other in a zero-sum game. The generator is trained to generate new data following a certain probability distribution from random noise, sculpturing it, to form output similar to the training set as shown in figure 2.9. The output of the generator is compared with the real data given in same training set. The discriminator is trained to compare the output of the network with the real data, classifying the output of the network as "real" or "fake". The two models are trained together in an adversarial, zero-sum game in order to fool each other: the generator learns to produce more and more realistic examples to fool the discriminator that, going forward with the training, will be more capable to discriminate between real or fake examples. The training process finishes when the discriminator is fooled about half the time, meaning the generator is generating plausible examples.

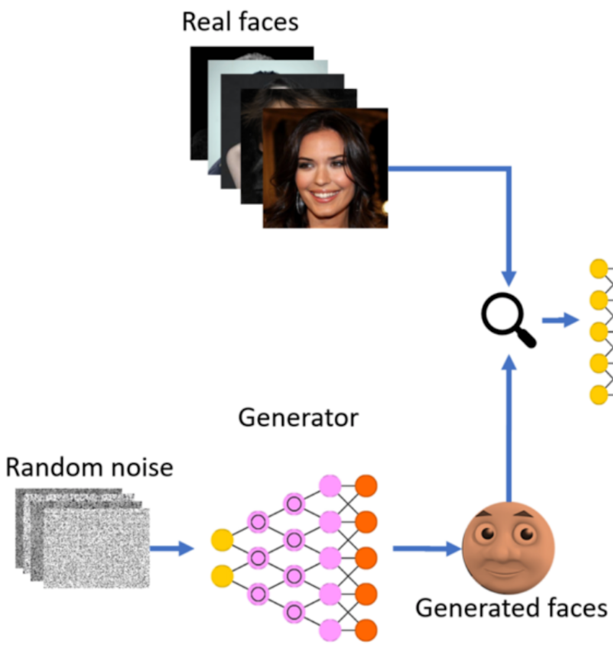


Figure 2.9: GAN architecture for face generation. Image taken from `https://www.pngwing.com/en/free-png-xflrf`

2.6. Conclusive Remarks

In this chapter we made a brief summary regarding the most used time-frequency representations of audio signals; we gave a summary of Machine Learning paradigms and categories, ending in Deep Learning models description, categorization and usage along with theoretical description.
In the next chapter we will present the state of the art concerning timbre transfer methods with Deep Learning.

3 | State of The Art

Generative models based on Deep Learning embody a consistent part of the latest areas of research. As computational power has increased and thanks to the spread of big, organized datasets, deep learning algorithms have become more and more popular, showing promising results and achievements in various research fields. In the field of audio engineering, Deep Learning models are often combined with Signal Processing algorithms to investigate sound/music, wanting to extract the most meaningful representation possible to exploit for a given task. In the generative art field, one example is Neural Style Transfer [14, 28]: a CNN-based technique used for rendering a content image in different styles as depicted in figure 3.1.

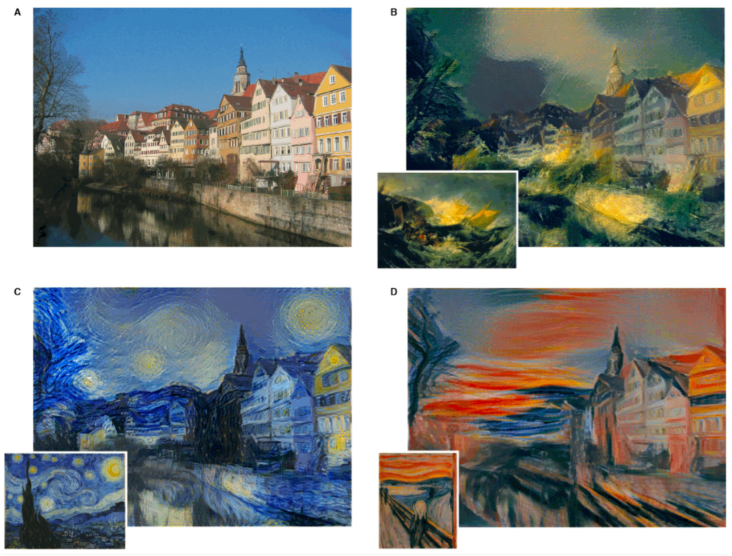


Figure 3.1: Neural Style Transfer: the landscape (content) is preserved, while the style is changed with famous paintings of William Turner, Vincent Van Gogh, and Edvard Munch. Image taken from [15].

Given an image, this technique is then capable of discerning its *content* from its *style*.

In this work we want to emulate this concept, applying it to the music domain. What could represent the content and the style when talking about music? In order to answer this question we have to introduce three concepts: *amplitude, pitch* and *timbre*:

1. **amplitude** of a sound wave is the measure of the height of the wave. The amplitude of a sound wave can be defined as the loudness or the amount of maximum displacement of vibrating particles of the medium from their mean position when the sound is produced;

2. **pitch** is a perceptual property of sounds that allows their ordering on a frequency-related scale [35], or more commonly, pitch is the quality that makes it possible to judge sounds as "higher" and "lower" in the sense associated with musical melodies [44];

3. **timbre** refers to the perceptual qualities of a musical sound distinct from its amplitude and pitch. It is timbre that allows a listener to distinguish between a guitar and a violin both producing a concert C note [8].

That being said, performing Style Transfer in audio/music fields could translate into performing **Timbre Transfer**, meaning that the timbre represent the style while the amplitude and the pitch the content. Another example of Style Transfer applied to music is the one proposed in [16] where the authors focus the work on music genre transfer.

Timbre depends on a myriad of factors: the acoustic properties of the source, the environment in which the sound is propagating, the non-linear dependency on its loudness, in the case of instruments, also how the performer is playing, and the list goes on. Even if there is a substantial body of research in timbre modelling and synthesis [6, 48, 55], most of state-of-the-art musical sound libraries used by studio composers are still obtained with high quality records of real instruments [24]. This procedure can be however expensive and time consuming, leading to open research prospects in the field, especially in the timbre transfer sub-field.

In this chapter we will present the current state-of-the-art works on Timbre Transfer techniques based on Deep Learning, briefly describing some of the most remarkable works.

3.1. Timbre Transfer using Generative Models

Modeling timbre is a hard task: being defined as "that attribute of auditory sensation which enables a listener to judge that two nonidentical sounds, similarly presented and having the same loudness and pitch, are dissimilar" [54]. That is clearly a perceptual definition so it is a difficult job to define a physical or mathematical model of timbre.

3.1.1. Timbre Transfer using Generative Adversarial Networks

TimbreTron [24] is one of the first example of architectures performing high quality timbre transfer using Deep Learning models. It was inspired by Neural Style Transfer successes [14, 29, 61] adapted in order to work with spectrograms. One of the challenges to face when dealing with spectrograms is the one of re-synthesis, i.e. passing from a spectrogram representation to an actual audio signal. To do that we need both the magnitude and phase information of the STFT and Griffin Lim Algorithm GLA can produce undesirable artifacts [52]. To overcome this issue, the authors convert the generated spectrograms to a waveform using a conditional WaveNet synthesizer [62]. The timbre transfer part is performed using an architecture called *CycleGAN* [7], a variation of GAN for unsupervised domain transfer able to learn a mapping between two domains without paired data. It works by learning two generators mapping $F : \mathcal{X} \mapsto \mathcal{Y}$ and $G : \mathcal{Y} \mapsto \mathcal{X}$ and two discriminators $D_{\mathcal{X}} : \mathcal{X} \mapsto [0, 1]$ and $D_{\mathcal{Y}} : \mathcal{Y} \mapsto [0, 1]$ with $\mathcal{X}$ and $\mathcal{Y}$ representation of image-like data belonging to two different classes represented by the numeric labels 0 and 1. The loss function consists of an adversarial loss combined with a a cycle consistency constraint which forces it to preserve the structure of the input.

Another timbre transfer architecture based on GAN is presented in [37] where the authors focus on the task of transferring the style of percussive patterns extracted from polyphonic audio using a MelGAN-VC model [42] trained on music genres. The MelGAN-VC architecture is the Mel-Spectrogram adaptation of Transformation Vector Learning GAN (TraVeLGAN) [3]. It adds a siamese network to the generator and the discriminator and trains to preserve vector arithmetic between points in the latent space of the siamese network so it can preserve semantic information. The experiment of this work consists in transferring the drum style from hip-hop to metal. The model is trained on a subset of GTZAN dataset [1] which contains 100 30-seconds long tracks for 10 music genres. Only hip-hop and metal folders are retained and used to train the network, after source separation has been performed by Spleeter [22] to extract only drum tracks.

Generative Adversarial Networks are powerful generative architectures but tend to be difficult to train. They can suffer the following major problems [4]:

- non-convergence: the model parameters oscillate, destabilize and never converge;
- mode collapse: the generator collapses resulting in limited varieties of samples;
- diminished gradient: the discriminator gets too successful not providing enough information for the generator to make progress;

[1] `https://www.tensorflow.org/datasets/catalog/gtzan`

- unbalance between the generator and discriminator causing overfitting;
- highly sensitive to the hyperparameter selection.

These problems are reflected in a lack of a well organized latent space. Since the study on how a well-trained GAN is able to encode different semantics inside the latent space is still in progress [53], the motion inside the latent space generated by a GAN can be hard to handle and interpret, leading to poor generative results and difficulties in the case of application where we need to move fluidly between latent space samples.

3.1.2. Timbre Transfer using VAEs

As explained in section 2.5.4, Variational Autoencoders are specifically designed to have a regularised latent space to avoid overfitting and to ensure that the latent space has good properties that enable generative process. This architecture fits well for timbre transfer tasks, but especially for the timbre interpolation task which would be that of changing smoothly the timbre of an audio signal going from a specific timbre to another, discovering the intermediate timbres. Thanks to their structure based on encoding input data into a lower dimension space, it is possible to understand the structure beneath the encoded data. In the 2D and 3D case, it is actually possible to visualize samples inside the latent space, clustering depending on their properties.

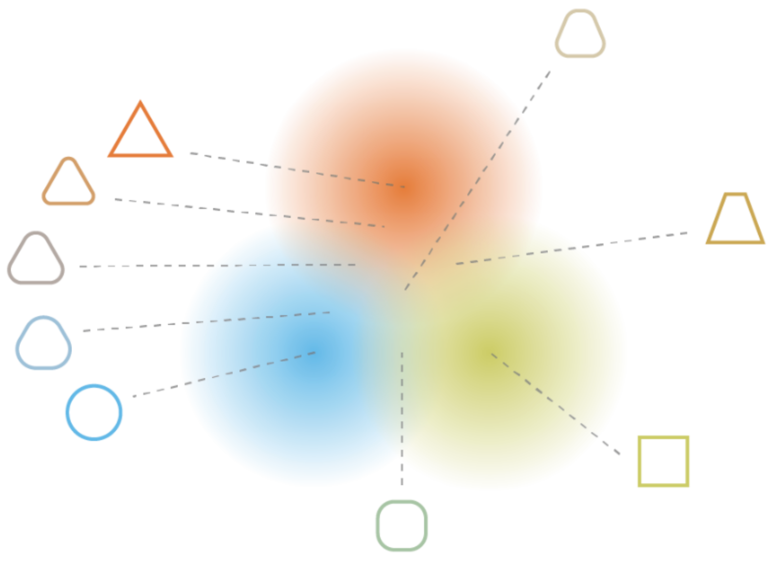


Figure 3.2: VAE 2D latent space. Regularisation tends to create a "gradient" over the information encoded in the latent space. Image taken from `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`.

In [60], timbre transfer is performed with a Variational Autoencoder. The work aims at helping composers by utilizing an abstract *latent timbre space*, generated by training an unsupervised VAE with a set of audio recording, allowing composers to synthesize sounds using a latent space of audio that is constrained to the timbre space of the audio recordings in the training set. The framework consists of three modules: calculation of the CQT, latent audio frame space generation with VAE and inverse synthesis using CQT-based magnitude spectrogram generated by the decoder of the VAE.

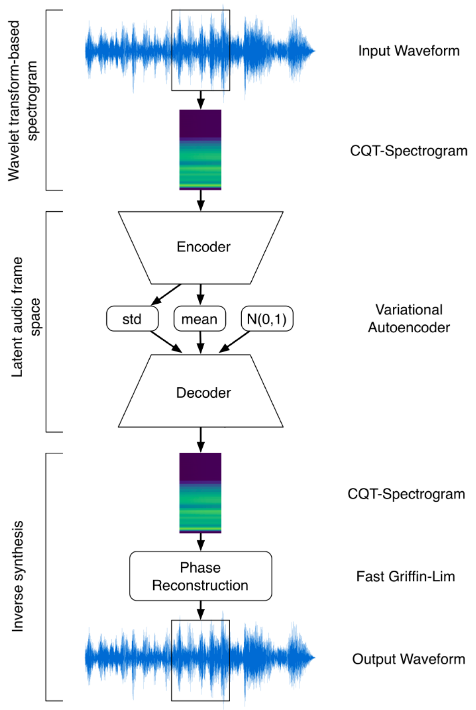


Figure 3.3: Latent Timbre Synthesis framework [60].

A further step towards an even more regularized space can be taken with conditioning the learning phase, with an architecture called *Conditional Variational Autoencoder* (CVAE) [56]. CVAE is a conditional directed graphical model whose input observations modulate the prior on Gaussian latent variables that generate the outputs, trained to maximize the conditional log-likelihood. In other words, we are regularizing the conditional probabilities of both the encoder and the decoder by adding a posterior variable.

CAESYNTH [46] is an audio synthesizer based on a conditional autoencoder. The variable conditioning the autoencoder is the accuracy in timbre classification. In fact, the regularization is characterized by two terms: an adversarial term to address pitch disentanglement and a classification term aiming to uniformly distribute the timbre quality. The conditioning is carried out by concatenating the desired one-hot pitch embedding along with the inferred latent code. The general workflow consists still of the calculation of the training set spectrograms, feed them to the model, sample the variables from the latent space, generate the new spectrograms and re-synthesize, recover back the phase with the conventional Griffin-Lim Algorithm.

Another example of regularization of the latent space is given in [13]. In this work Esling et. al. introduce a specific regularization that allows to enforce any given similarity distances onto latent spaces. Given the continuity of these spaces, they also study how audio descriptors behave along latent dimensions showing that even if they have a non-linear topology, they follow a locally smooth evolution. The authors introduced a method for descriptor-based synthesis showing that they can control the descriptors of an instrument while keeping its timbre structure. The regularization is, in this case, carried by human dissimilarity ratings between pairs of audio samples inside a set of instruments collected in five independent perceptual ratings studies [5, 19, 26, 36, 40] and organised by applying Multi Dimensional Scaling (MDS), leading to timbre spaces which exhibit the perceptual similarity between different instruments. They bridge timbre perception analysis and perceptually-relevant audio synthesis by regularizing the learning of VAE latent spaces so that they match the perceptual distances collected from the five timbre studies, showing that perceptually-regularized latent spaces are simultaneously coherent with perceptual ratings, while being able to synthesize high-quality audio distributions.

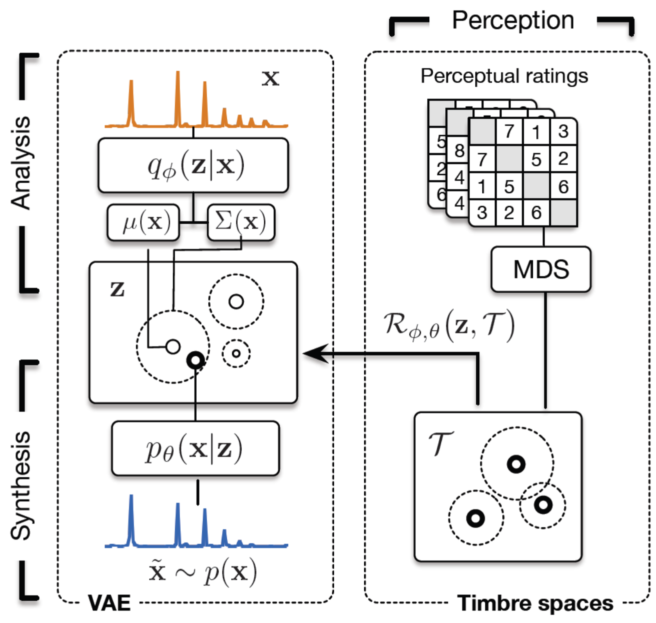

Perception
Analysis
Synthesis
$\mathbf{x}$
$q_\phi(\mathbf{z}|\mathbf{x})$
$\mu(\mathbf{x})$
$\Sigma(\mathbf{x})$
$\mathbf{z}$
$p_\theta(\mathbf{x}|\mathbf{z})$
$\tilde{\mathbf{x}} \sim p(\mathbf{x})$
VAE
Perceptual ratings
MDS
$\mathcal{R}_{\phi,\theta}(\mathbf{z},\mathcal{T})$
$\mathcal{T}$
Timbre spaces

Figure 3.4: Architecture proposed in [13].

A slightly different architecture is used in [39] where the focus is on learning disentangled representations of timbre and pitch. The main structure is still based on a Variation Autoencoder, with Gaussian mixture latent distributions (GMVAE [10, 27, 33]). The difference between regular VAEs and GMVAEs is that the former prior distribution's common choice $p(\mathbf{z})$ is an *isotropic Gaussian*, encouraging each dimension of the latent variables to capture an independent factor of variation from the data resulting in a disentangled representation [23], not allowing for multi-modal representations while the latter extends the prior to a mixture of Gaussians assuming the observed data are generated by first determining the mode from which it was generated. This is needed for learn separate latent distributions for timbre and pitch through a model composed of two separate encoders for pitch and timbre and a shared decoder as shown in figure 3.5.

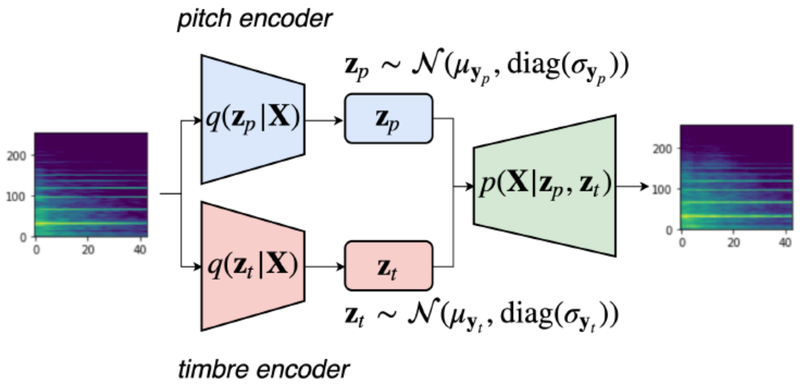

pitch encoder
$\mathbf{z}_p \sim \mathcal{N}(\mu_{\mathbf{y}_p}, \mathrm{diag}(\sigma_{\mathbf{y}_p}))$
$q(\mathbf{z}_p|\mathbf{X})$
$\mathbf{z}_p$
$p(\mathbf{X}|\mathbf{z}_p, \mathbf{z}_t)$
$q(\mathbf{z}_t|\mathbf{X})$
$\mathbf{z}_t$
$\mathbf{z}_t \sim \mathcal{N}(\mu_{\mathbf{y}_t}, \mathrm{diag}(\sigma_{\mathbf{y}_t}))$
timbre encoder

Figure 3.5: Separate pitch and timbre encoders sharing the decoder.

The model proposed in [51] adopts a VAE-GAN approach. It is motivated by UNIT [38], a VAE-GAN model that performs neural style transfer in both directions namely going from the style of the source image to the one of the target image and viceversa. For each direction they use an encoder-decoder section that follows the reconstructive objective of VAEs but the whole path follows the adversarial objective of GANs. The VAE component motivates content persistence while the GAN one the style transfer. The work follows in the footsteps of [2] that is a VAE-GAN architecture used for voice conversion, reworked to be used for timbre transfer between instruments. The model involves one universal encoder that leads to a shared residual block for decoding at a superficial level and multiple decoder-discriminator pairs specific to target domains. The overall loss function is composed of four parts:

$$L = L_{GAN} + L_{VAE} + L_{CC} + L_{Latent}, \tag{3.1}$$

where L_{GAN} is the adversarial loss, L_{VAE} the variational encoding loss, L_{CC} the cyclic consistency loss and L_{Latent} the latent loss between a pair of latent means $\boldsymbol{\mu}_A$ and $\boldsymbol{\mu}_B$ from different source mel-spectrograms A and B.

3.2. DDSP

Differentiable Digital Signal Processing (DDSP) [12] combines Deep Learning methods with classic Digital Signal Processing algorithms, having different uses such as blind dereverberation of audio through separate modeling of room acoustics, transfer of extracted room acoustics to new environments and, of course, timbre transfer between disparate sources.

What differentiates DDSP from other methods is the synthesis part of the architecture. While previous models aim at reconstructing the spectrogram from which is possible to compute the audio, DDSP is based on DSP components (expressed as feed-forward functions, allowing efficient implementation on parallel hardware) such as *oscillators, envelopes* and *filters*, whose parameters are learned during the training phase of the model. The neural network, in fact, learns the parameters needed for these components to perform the re-synthesis part of the algorithm. The architecture shown in figure 3.6 is still an autoencoder one: three encoders (loudness encoder $l(t)$, fundamental frequency encoder $f(t)$ and timbre encoder $z(t)$) encode the input audio information. Now the decoder no longer has the task of directly reconstruct the audio, it rather learns the parameter for the additive and filtered noise synthesizers. The reconstruction loss between the synthesized audio and the original one is minimized, in the form of a *Multi-Scale Spectral Loss* defined as:

$$L_i = ||\mathbf{S}_i - \hat{\mathbf{S}}_i||_1 + \alpha || \log \mathbf{S}_i - \log \hat{\mathbf{S}}_i||_1, \tag{3.2}$$

where $\mathbf{S}_i$ and $\hat{\mathbf{S}}_i$ represent respectively the spectrograms of the original and re-synthesized audio and α a weighting term.

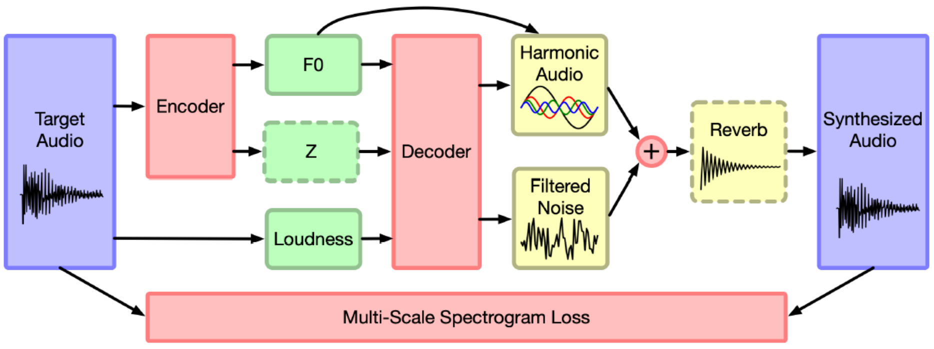


Figure 3.6: DDSP architecture [12].

3.3. Conclusive Remarks

In this chapter we first described the first examples of Neural Style Transfer for images, the starting point for extending the concept in order to be applied for music timbre transfer based on spectrograms. Then we gave a qualitative definition of *amplitude, pitch* and *timbre*, fundamental concepts for our work. Afterwards we have presented some of the latest architectures used in the literature for the timbre transfer task, starting with GAN architectures, passing through VAEs and Conditional VAEs ending with the alternative DDSP framework. In the next chapter we will present our method, explaining structure and details of the architecture.

4 | Proposed Method

In this chapter we will first formalize the problems we're going to address namely the timbre transfer problem and the timbre interpolation problem. Next, we will present our system architecture, analyzing in depth the components that structure the model, presenting the choices made and explaining how the training phase of the network works. In those sections we will present all the techniques adopted in order to build a system able to perform an end to end model, starting from the pre-processing phase where we transform and prepare the input audio files, moving on to the network itself that is the core of the model, ending up into the post-processing phase that takes the output of the network and transform back it into the generated audio files. Lastly we will present the timbre interpolation technique adopted in this work.

4.1. Problem Statement

Before starting with the actual problem statement, it's important to define some terminology and concepts that we will use throughout this section.
The first important concept is the one of **timbral class**: with that, we intend the timbre associated to the audio file. Examples of timbre classes are *guitar acoustic, organ electronic*. In chapter 5 we'll present the timbre classes used in this work with technical details.
The second important concept is the one of **target timbre**: when performing timbre transfer, we want to reach a specific timbre for the output of the network that, from now on will be called target timbre.
In the context of one-to-many timbre transfer, we have multiple timbral classes functioning as target timbres. Precisely, from now on, we will have $n \in \mathbb{N}$ target timbral classes.

The problem we want to tackle in this thesis is twofold: firstly, we want to perform one-to-many timbre transfer from notes of a given timbre class and secondly we want to perform timbre interpolation between the notes generated performing the timbre transfer task, exploiting the properties of the latent space derived during the training phase.

Let us first explicitly define the one-to-many timbre transfer problem. Given a discrete sound signal $\mathbf{x}^t$ characterized by a specific timbral class t, we calculate its time-frequency representation $\mathbf{S}^t$:

$$\mathbf{S}^t = |STFT(\mathbf{x}^t)|. \tag{4.1}$$

Considering the network as a function f that takes $\mathbf{S}^t$ as input, the output will be:

$$\mathbf{S}^{t_1}, \dots, \mathbf{S}^{t_n} = f(\mathbf{S}^t), \tag{4.2}$$

where the apexes $t, t_1, ..., t_n$ represent a discrete pre-defined set of timbre classes T, with, as explained before, cardinality n. The set of target timbral classes T is dataset and application dependent. These representations are transformed back using Griffin Lim Algorithm (GLA), ending up in a set of discrete audio signals:

$$\mathbf{x}^{t_1}, \dots, \mathbf{x}^{t_n} = GLA(\mathbf{S}^{t_1}, \dots, \mathbf{S}^{t_n}), \tag{4.3}$$

each one representing the audio signal $\mathbf{x}^t$ with only the timbral characteristic varied.

As anticipated, we also explore the possibility of performing interpolation between different timbres. While the interpolation task is not directly modeled during the training procedure, it is possible thanks to the latent space ordering. With interpolation we mean the generation of new notes that perform a gradual *morphing* between two given notes belonging to two different timbral classes: given a couple of generated signals $\mathbf{x}^{t_a}$ and $\mathbf{x}^{t_b}$ with $a, b \in T$ and $a \neq b$, the system will produce $m \in \mathbb{N}$ *timbre interpolations* between $\mathbf{x}^{t_a}$ and $\mathbf{x}^{t_b}$, with m adjustable. The first and the last of the m outputs will be, by convention, exactly $\mathbf{x}^{t_a}$ and $\mathbf{x}^{t_b}$ that can be defined as the **starting point** and the **ending point** of the interpolation. For example, having $\mathbf{x}^{t_a}$ a generic sample with the timbre class a, $\mathbf{x}^{t_b}$ a generic sample with the timbre class b and $m = 3$, we will have a single intermediate generated sample between the starting point $\mathbf{x}^{t_a}$ and the ending point $\mathbf{x}^{t_b}$ that will have a timbre that is a mixture of the class a and the class b revealing, indeed, a new timbre. A graphic example with $m = 5$ is depicted in figure 4.1.

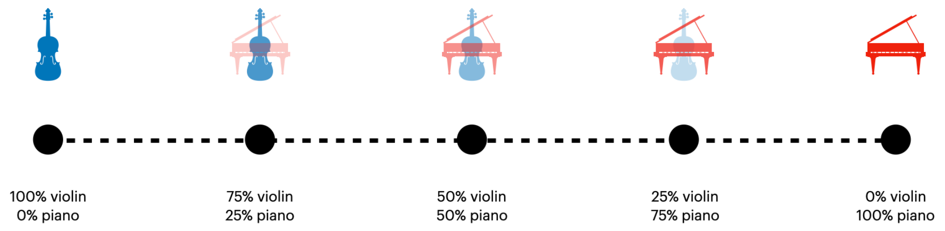


Figure 4.1: Schematic representation of interpolation between violin and piano with $m = 5$.

4.2. System Architecture

In this section we will present an overview of the system architecture.
Since, as we will deepen, the timbre interpolation task is derived from the timbre transfer task, in the first three subsections we will analyze the model built to perform the timbre transfer task and we dedicate the last subsection to explain the timbre interpolation method.
We start from the input audio signals from which we extract the spectrograms, performing the STFT. We normalize them and feed them to the network that outputs, for every normalized input spectrogram, n normalized generated spectrograms, with $n \in \mathbb{N}$ target timbral classes. Each of the generated spectrograms is de-normalized and the last step consists in performing GLA in order to synthesize the new generated audio signals. The end to end representation of the system is depicted in figure 4.2.

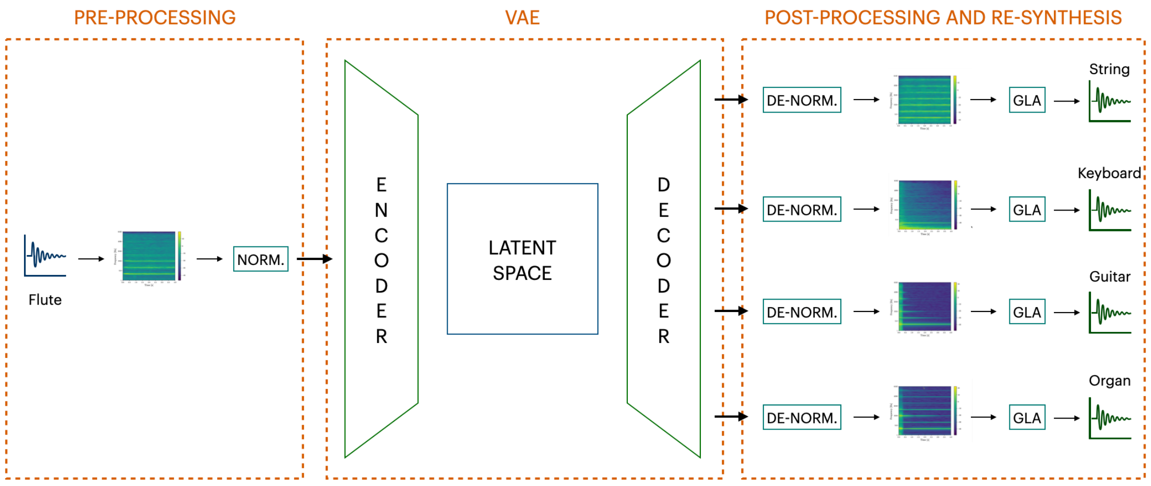


Figure 4.2: End to end representation of the system.

4.2.1. Pre-Processing

The pre-processing section in our work has two main purposes:

1. transform the audio signals in a truthful, high-quality representation of them;
2. make a transformation of the audio signals so that they are compatible with what the network requires as input in order to perform the training phase.

VAE architectures are designed for reconstruction and generation. In this work we want to perform timbre transfer, maintaining the pitch information. Given that, it is essential for this work that the output of the network can be transformed back in order to get an acoustically pleasant signal resembling the one of the dataset. The choice of the audio representation turns out to be crucial to have good results. In Chapter 2 we have seen that the STFT is the basic form of time-frequency representation while the CQT and the Mel spectrogram are perceptually based representation. It would then appear that the CQT and the Mel spectrogram are the best representation for our task. However, these two representations have each one a problem:

- the CQT is based on the concept of notes (one of the parameters of the CQT is the "bins per octave"). But we know that sometimes music signals do not follow exactly the perfect pitch representing the notes. One example could be the bending technique used by a guitarist: the performer bend one string of the instrument in order to achieve a different pitch from the normal one. If we have a slow bend and use the CQT as audio representation, we won't have the same number of bins for all the mid frequencies discovered while bending;

- the Mel-spectrogram representation perform a multiplication of the frequency axis with a bank of triangular filters. By doing that we are actually losing some information along the frequency axis: this is not feasible in our experiments since the re-synthesis of the new audio files will be in fact lossy.

For this reason we decided to use the STFT as our time-frequency representation of choice. Precisely we calculated the **log spectrogram** called $STFT_{dB}$ (since it follows the dB standard) from the magnitude of the STFT with the following formula:

$$STFT_{dB} = 10 \cdot \log_{10}(STFT_{mag} + 10^{-5}). \tag{4.4}$$

The 10^{-5} factor is included into the formula because to avoid $\log_{10}(0)$ issues.

Another important operation performed in the pre-processing phase is the one of **normalization**. Data normalization refers to the organization of data to appear similar across all dimensions.

In our work, we perform a 0-1 range min-max normalization meaning that for each set (train set and validation set), we extract the maximum and the minimum values between all log spectrograms and we re-scale all the log spectrograms of the aforementioned set so that every single value is between 0 (that correspond to the minimum value) and 1 (that correspond to the maximum value). Formally, given $\mathcal{S} = \{\mathbf{S}_1, \ldots, \mathbf{S}_M\}$ one of the set between training set, validation set and test set containing all M log spectrograms of all timbre classes, we extract the scalar $min(\mathcal{S}) = min(\{\mathbf{S}_1, \ldots, \mathbf{S}_M\})$ and the scalar $max(\mathcal{S}) = max(\{\mathbf{S}_1, ..., \mathbf{S}_M\})$ that are the minumum and maximum values between all log spectrograms. Given a spectrogram $\mathbf{x}_i$, its normalized version $N(\mathbf{S}_i) \in \mathcal{S}$ will be:

$$N(\mathbf{x}_i) = \frac{\mathbf{x}_i - min(S)}{max(S) - min(S)}. \tag{4.5}$$

Data normalization has different benefits that are needed for these reasons:

- large values given as input to the network lead to large weights inside the layers of the network. It is beneficial for the network to have small weights since a model with large weight values is often unstable, meaning that it may suffer from poor performance during learning and sensitivity to input values resulting in higher generalization error. Large weights lead to sharp transitions in the node functions and thus big changes in output for small changes in the inputs [47];

- most of the times, datasets contain features that highly vary in magnitudes. This could be a problem because most of the machine learning algorithms use distances between two data points in their computations and the features with high magnitudes will weight more than the ones with small ones. Normalization brings all the features to the same, small level of magnitude making the learning phase more regular.

4.2.2. Network Architecture

The architecture can be defined as a conditional convolutional beta-VAE.

The beta-VAE architecture [23] adds the adjustable parameter $\beta \in \mathbb{R}$ to the VAE loss. The beta-VAE loss can be formalized as:

$$LOSS = L_R + \beta \cdot L_{KL} = ||\hat{\mathbf{y}} - \mathbf{y}|| + \beta \cdot KL[q(\mathbf{z}|\mathbf{x}), p(\mathbf{x}|\mathbf{z})]. \tag{4.6}$$

It has been proven that beta-VAE with appropriately tuned $\beta > 1$ qualitatively outperforms VAE ($\beta = 1$). The parameter β can be considered as a weighting parameter between the reconstruction loss and the KL loss and finding the perfect balance between those losses is a delicate exercise. β can be considered as an hyper-parameter of the network

In this section we will break down the structure of the model used for training. The model can be split into encoder and decoder, both reported respectively in tables 4.1 and 4.2, together with the full autoencoder architecture reported in table 4.3. The end to end graphic representation of the network is depicted in Fig. 4.3.

Layer name	Output shape	# params	Depth	Size	Stride
encoder_input	$(512 \times 256 \times 1)$	0	-	-	-
encoder_cond_input	$(512 \times 256 \times 1)$	0	-	-	-
encoder_concat_input	$(512 \times 256 \times 2)$	0	-	-	-
encoder_conv_1	$(256 \times 128 \times 16)$	304	16	(3×3)	(2×2)
encoder_relu_1	$(256 \times 128 \times 16)$	0	-	-	-
encoder_batch_norm_1	$(256 \times 128 \times 16)$	64	-	-	-
encoder_conv_2	$(128 \times 64 \times 32)$	4640	32	(3×3)	(2×2)
encoder_relu_2	$(128 \times 64 \times 32)$	0	-	-	-
encoder_batch_norm_2	$(128 \times 64 \times 32)$	128	-	-	-
encoder_conv_3	$(64 \times 32 \times 64)$	18496	64	(3×3)	(2×2)
encoder_relu_3	$(64 \times 32 \times 64)$	0	-	-	-
encoder_batch_norm_3	$(64 \times 32 \times 64)$	256	-	-	-
encoder_conv_4	$(32 \times 16 \times 128)$	73856	128	(3×3)	(2×2)
encoder_relu_4	$(32 \times 16 \times 128)$	0	-	-	-
encoder_batch_norm_4	$(32 \times 16 \times 128)$	512	-	-	-
encoder_conv_5	$(16 \times 16 \times 256)$	295168	256	(3×3)	(2×1)
encoder_relu_5	$(16 \times 16 \times 256)$	0	-	-	-
encoder_batch_norm_5	$(16 \times 16 \times 256)$	1024	-	-	-
flatten	(65536)	0	-	-	-
mu	(64)	4194368	-	-	-
log_variance	(64)	4194368	-	-	-
encoder_output	(64)	0	-	-	-
decoder_cond_input	(4)	0	-	-	-
bottleneck_concat	(68)	0	-	-	-

Table 4.1: Encoder architecture.

Layer name	**Output shape**	**# params**	**Depth**	**Size**	**Stride**
`decoder_input`	(68)	0	-	-	-
`decoder_dense`	(65536)	4521984	-	-	-
`reshape`	$(16 \times 16 \times 256)$	0	-	-	-
`decoder_conv_trans_1`	$(32 \times 16 \times 256)$	590080	256	(3×3)	(2×1)
`decoder_relu_1`	$(32 \times 16 \times 256)$	0	-	-	-
`decoder_batch_norm_1`	$(32 \times 16 \times 256)$	1024	-	-	-
`decoder_conv_trans_2`	$(64 \times 32 \times 128)$	295040	128	(3×3)	(2×2)
`decoder_relu_2`	$(64 \times 32 \times 128)$	0	-	-	-
`decoder_batch_norm_2`	$(64 \times 32 \times 128)$	512	-	-	-
`decoder_conv_trans_3`	$(128 \times 64 \times 64)$	73792	64	(3×3)	(2×2)
`decoder_relu_3`	$(128 \times 64 \times 64)$	0	-	-	-
`decoder_batch_norm_3`	$(128 \times 64 \times 64)$	256	-	-	-
`decoder_conv_trans_4`	$(256 \times 128 \times 32)$	18464	32	(3×3)	(2×2)
`decoder_relu_4`	$(256 \times 128 \times 32)$	0	-	-	-
`decoder_batch_norm_4`	$(256 \times 128 \times 32)$	128	-	-	-
`decoder_conv_trans_5`	$(512 \times 256 \times 1)$	289	16	(3×3)	(2×2)
`sigmoid_layer`	$(512 \times 256 \times 1)$	0	-	-	-

Table 4.2: Decoder architecture.

Layer name	**Output shape**	**# params**
`encoder_input`	$(512 \times 256 \times 1)$	0
`encoder_cond_input`	$(512 \times 256 \times 1)$	0
`decoder_cond_input`	(4)	0
`encoder`	(68)	8783184
`decoder`	$(512 \times 256 \times 1)$	5501569

Table 4.3: Full Variational Autoencoder architecture.

The network can be defined as a function f:

$$\mathbf{S}^c_{F\times T} = f(\mathbf{S}_{F\times T}, \mathbf{C}^c_{F\times T}, \mathbf{h}^c) \tag{4.7}$$

where $\mathbf{S}_{F\times T}$ is the input spectrogram with T the total number of time frames and F the total number of frequency bins, $\mathbf{C}^c_{F\times T}$ is the conditioning matrix concatenated on the channel axis with $\mathbf{S}_{F\times T}$, $\mathbf{h}^c$ is the one-hot vector concatenated with the input of the decoder and $\mathbf{S}^c_{F\times T}$ is the generated spectrogram with the new timbre. Each entry $\mathbf{C}^c_{f\times t}$ with $t = \{1, \ldots, T\}$ and $f = \{1, \ldots, F\}$ of the conditioning matrix has the same natural value $c \in \{0, \ldots, n-1\}$ associated to a specific timbral class and the vector $\mathbf{h}^c$ is the one-hot representation of that value.

Tables 4.1 and 4.2 can be conceptually divided into sections. Starting from the encoder, we have a first section representing the inputs of the model. The `encoder_input` layer is the one associated to the normalized log spectrograms that we feed the network. The layer after is `encoder_cond_input` and has the same shape of `encoder_input`. The third layer `encoder_concat_input` performs concatenation between the spectrograms in the first layer and the matrices in the second. The concatenation happens along the third axis, also called the *channel axis*. The second and the third layer are responsible for the first of the two mechanism that render the VAE conditional.
After the concatenation, we have five convolutional blocks. Each one is composed by a convolutional layer followed by ReLU activation and a batch normalization layer.
After the fifth convolutional we have a flatten layer that brings the dimensionality of the output of the layers from three to one.
After the flatten layer, we have two parallel layers that constitute the latent space of the network: the `mu` layer and the `log_variance` layer with output shape $= D$ that is the *dimensionality* of the latent space. The bottleneck of the network is defined by the `encoder_output` layer. This layer performs the reparametrization trick [34] that is needed since the process of sampling from a distribution that is parameterized by a model is not differentiable. Rewriting our implementation of how we parameterize our Gaussian sampling, we allow a gradient path through a non-stochastic node as explained in section 2.5.4.

The output of the encoder is then concatenated with the one-hot encoding vector as said in equation 4.7.

The first layer of the decoder is a dense one followed by a reshape layer that re-establishes the dimension of the output from one to three so we can perform the transposed convolutional operations.

The structure of the transposed convolutional blocks of the decoder is symmetrical to the encoder, except for the last block that doesn't have the ReLU activation layer and the batch normalization layer.

The last layer of the decoder is followed by sigmoid activation.

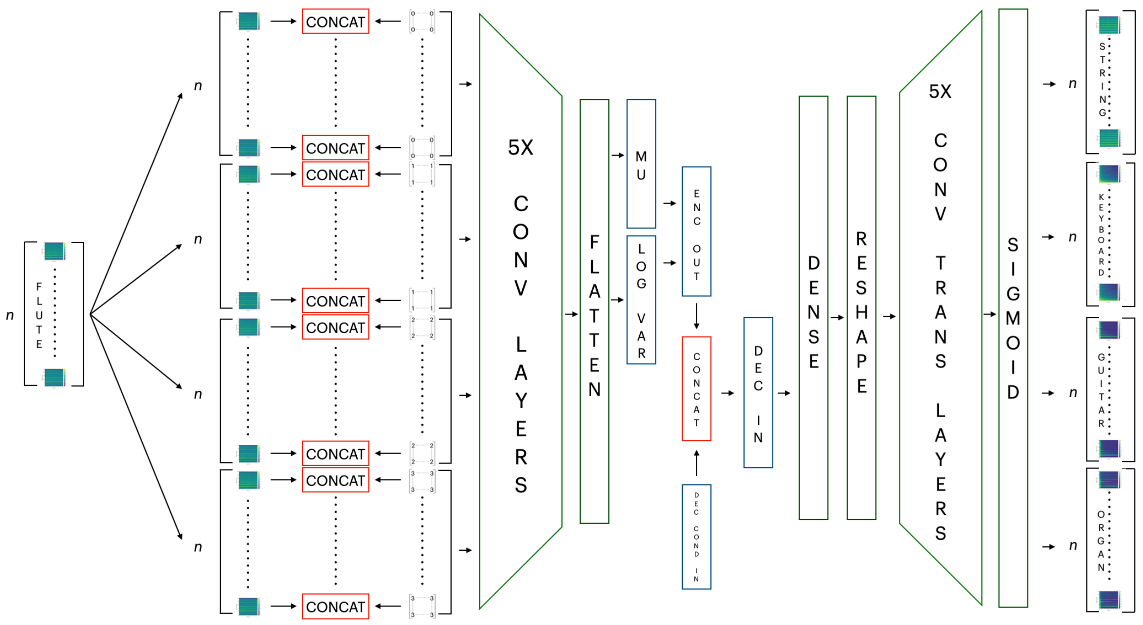


Figure 4.3: End to end graphic representation of the network.ti

4.2.3. Post-Processing

For each normalized log spectrogram given as input, the network produces n normalized log spectrograms, one for each target timbre. The post-processing part of the pipeline has the purpose of taking these normalized log spectrograms and transform them ending up in audio signals. Therefore the post-processing operations occurs only in the **inference** phase, namely when the training phase is finished and new data (i.e. the test set) is presented to the model. Once the post-processing section is done, it is possible to listen to the audio files and proceed to the evaluation of the model.

The first part of the post-processing phase is the de-normalization of the output of the network. We have to perform the inverse of formula 4.5. Given a generated spectrogram $\hat{S}_i^t$ with target timbre t and considering $\mathcal{S}$ the same set explained in the pre-processing section 4.2.1, we perform de-normalization as:

$$D(\hat{\mathbf{S}}_i^t) = \hat{\mathbf{S}}_i^t \cdot (max(\mathcal{S}) - min(\mathcal{S})) + min(\mathcal{S}), \tag{4.8}$$

where i is the index of the generated normalized log spectrogram that ranges than from 1 to M. This leads to a de-normalization process that follows the full range of values.

After the de-normalization, we end up having a set of de-normalized log spectrograms. Before proceeding to re-synthesis we must bring back the de-normalized log spectrograms to the form of $STFT_{mag}$ by inverting formula 4.4 with:

$$STFT_{mag} = 10^{\frac{STFT_{dB}}{10}} - 10^{-5}). \tag{4.9}$$

As anticipated, we rely on Griffin-Lim Algorithm to perform the inversion of the STFT having only the magnitude information. We have seen that Convolutional Neural Networks are able to detect patterns in images with the goal of classification, detection or generation. Looking at the plot of the magnitude and the phase of a STFT as shown in figure 4.4, we can understand why the phase is usually discarded: its information is way more articulated than the one contained in the magnitude representation and it is very difficult for a machine to extract meaningful information and learn from it, especially with limited computational resources.

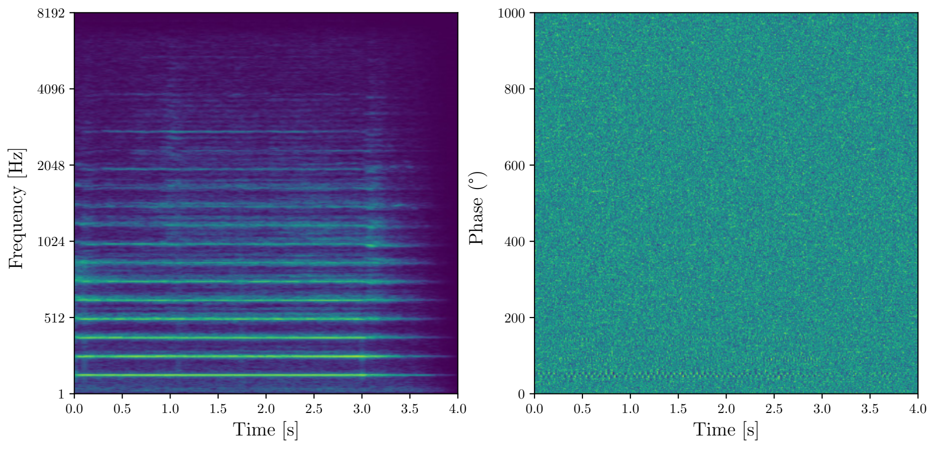


Figure 4.4: Magnitude and Phase of the same signal

When phase information is discarded, the *Griffin-Lim Algorithm* (GLA) is used. The GLA is a phase reconstruction method based on the redundancy of the short-time Fourier transform. This algorithm expects to recover a complex-valued spectrogram, which is consistent and maintains the given amplitude, by the usage of a projection procedure.

The phase estimation is not always optimal and the estimation of the phase is time consuming due to the iterative nature of the algorithm. These factors could result in a bottleneck on the audio quality reconstruction and on the speed of the algorithm in inference phase.

4.3. Interpolation

In the mathematical field of numerical analysis, interpolation is about the estimation of a method that aims to find new data points based on the range of a discrete set of known data points [58].

Our architecture is built to map input audio samples into a D dimensional latent space. Each input audio sample that goes trough the pre-processing and the conditioned training phase is represented by n points in the latent space that can be called **embeddings**. Formally, in the context of neural networks, embeddings are continuous vector representations of discrete variables. Embeddings are learned in the training phase and have the characteristic of being low-dimensional representations of input data. The purpose of the embedding process is the one of converting high-dimensional data to low-dimensional data in the form of a vector in such a way that the two are semantically similar. In our case, the formation of the embeddings has a slightly different course: we are not directly converting high dimensional data into a smaller space, like it happens when using a VAE with the sole aim of reconstructing the input data. Our architecture is built to perform timbre transfer, meaning that the output of the network are actually different from the input. That's where the concatenation operation between the normalized log spectrograms and the conditioning matrices actually renders our model conditional. Thanks to that, the encoder is able to map the same normalized log spectrogram into n different points in the latent space that the decoder will transform into n normalized log spectrograms with the corresponding target timbres. Formally, given an input sample $\mathbf{x}_i^t$ with timbral class t, it is mapped into n embeddings $\mathbf{e}_i^{t_1}, \ldots, \mathbf{e}_i^{t_n}$. Each one of the embeddings is actually represented by a point in the D dimensional latent space constituted by the output of the `mu` layer and by another point in the D dimensional latent space constituted by the output of the `log_var` layer.

The embeddings $\mathbf{e}_i^{t_1}, \ldots, \mathbf{e}_i^{t_n}$ when concatenated with the corresponding one-hot vector, decoded and post-processed will result, as said before, in audio samples with the target timbres. It is possible for us to sample new points belonging into the latent space that aren't necessarily associated to the input of the networks and that will result in spectrograms that follow the distribution of the latent space itself.

By doing that, we define new points in the latent space. It is also possible, from those points, to perform Algorithm 4.1 again, ending up in newer points representing a mixture of mixture of timbres. In the next chapter we will explicate and show the results of what just explained.

Algorithm 4.1 Interpolation of embeddings

```
1: $m$ = # of interpolations
2: $ratios$ = array of $m$ equally space float numbers $\in [0, 1]$
3: $embeddings = []$
4: $v1$ = starting embedding
5: $v1$ = ending embedding
6: for $i = 0$ to $m - 1$ do
7:     $v = (1.0 - ratio[i]) \times v1 + ratio[i] \times v2$
8:     append $v$ to $embeddings$
9: end for
10: return $embeddings$
```

4.4. Conclusive Remarks

In this chapter, after the problem statement section where we presented and formalize the objective of this thesis, we presented the method and the building blocks for timbre transfer and timbre interpolation developed in this work. We illustrated the Pre-Processing phase, justifying the technical choices taken. We then showed the Network Architecture describing its various layers and after that we reported the Post-Processing phase, responsible for all the operation to be done after the training phase in order to listen to the generated samples. Lastly, we formalized the interpolation method, giving a definition of embeddings and showing the algorithm used to perform the task.

5 | Experiments and Results

In this chapter we will present the results of our experiments aimed at demonstrating the effectiveness of the timbre transfer technique proposed in this thesis along with the description of the dataset, the technical details about our experimental setup and the evaluation methods.
First we will present the hyper parameters related to the training procedure of the proposed beta-VAE architecture. To explain that, we will go through our experimental setup and we will describe how the dataset has been set up for the tasks of timbre transfer. We will then illustrate the latent space topology together with explanatory images. Before presenting the actual results, we will define the evaluation methods that have been chosen to assess the outcomes of the system. There will be two different designated methodologies: the first one will rely on a classifier while the second one on perceptual evaluations. Finally we will show the results obtained in conjunction with observations and conclusions about the system.

5.1. Experimental setup and Dataset

The dimension of the STFT computed over the audio signals and fed into the network is of 512 × 256. The chosen parameters for the calculation of the STFT are $N\_fft =$ 1024 samples, $hop\ length = 128$ samples and $window\ length = 1024$ samples. This configuration has been chosen as a trade-off between the quality of the inversion and the quantity of data generated from the transform. Each of the audio signals in the dataset has length $l = 4s$ and it's sampled at $F_s = 16.000$kHz . Performing the $STFT$ with the listed parameters leads to a $F \times T$ frequency-time representation, having $F = \frac{N\_fft}{2} = 512$ the number of frequency bands discarding the negative ones and $T = \frac{l \cdot F_s}{hop\ length} = 500$ the number of time frames. For computational reasons, we considered the first 2.048s of the signal, ending up in a 512 × 256 frequency-time representation of it. Only $STFT_{mag}$, i.e. the magnitude information of the $STFT$, has been retained. We discarded the phase information since we rely on GLA to reconstruct it as explained in section 4.2.3.
In the re-synthesis step, we uses GLA with $hop\ size = 128$ (the same used for the

extraction of the STFT) and the number of iterations to infer the phase information has been set to 32.

The pre-processing and post-processing phases have been performed with the functions for feature extraction and re-synthesis provided by the *Librosa* v0.8.0 library [41] while the network presented in section 4.2.2 was developed using the *TensorFlow* v2.3.0 DL framework [1].

With regard to the network, we empirically found the best arrangement of parameters for our network with *learning rate* = 0.0001, *batch size* = 64, *Adam* optimizer [31], *latent dimension* $D = 64$ and the parameter β has been set to 2. Having *latent dimension* $D = 64$, it is possible to compress the data in an optimal way, resulting in a representation of the input that is not too small (that could cause an irreversible compression) but neither too big (that could end up in a sparse representation of the input data which is difficult to regularize). We imposed an early stopping on the training procedure when the validation loss did not decrease for more than 20 epochs saving the model with the best validation loss. The network converged at epoch 396 and the duration of the training has been around 90 minutes. The experiments have been performed using a workstation equipped with one Intel® Xeon E5-2630 v2 (12 Cores @2.6GHz), RAM 128 GiB, and two Quadro P6000 (3840 CUDA Cores @1530MHz), 24GiB, running Ubuntu 20.04.2. LTS.

The dataset is a subset of **NSYNTH dataset** [11], a large-scale and high-quality dataset of annotated musical notes. The NSYNTH dataset contains a total of 305,979 musical notes, each with a unique pitch, timbre, and envelope. Each sample is four seconds long, monophonic, sampled at 16 kHz and is generated from one over 1,006 instruments taken from commercial sample libraries. Each note has been held for the first three seconds and allowed to decay for the final second. Following the MIDI notation, the range of the pitches goes from 21 to 108 although not all instruments cover the full range, resulting in an average of 65.4 pitches per instrument and the velocities can have five values (25, 50, 75, 100, 127). Each note is saved inside the dataset with a name describing its qualities. The first part of the name represent the instrument *family* and the *source*. The *family* is the macro category of instruments contained into the dataset. There are 11 instrument families, namely *bass*, *brass*, *flute*, *guitar*, *keyboard*, *mallet*, *organ*, *reed*, *string*, *synth lead* and *vocal*. The *source* represents the method of sound production for the note's instrument and can be "acoustic" (recorded from instruments that produce sound through acoustic means), "electronic" (recorded from instruments that produce sound using electronic circuitry) or "synthetic" (recorded from synthesizers). The second part of the name is a triplet of sets of three digits, the first one associated to the specific instrument inside the family used for recording the sample, the second representing the pitch and the third the velocity.

In our work we used a subset of the aforementioned dataset in order to perform the specific case of one-to-four timbre transfer. As input we used the *flute acoustic* class and as output the *string acoustic*, *keyboard acoustic*, *guitar acoustic* and *organ electronic* classes.
In order to build our dataset, we performed a fine analysis of the samples provided by the NSYNTH dataset. The first step was the one of discarding all samples that did not satisfy common sense acoustic standards: some signals, especially in the bottom low and top high pitch ranges, were or inaudible or so noisy that they were not associative to the instrument class. We also noticed that a few specific instruments inside some families were mis-labelled in term of pitches having the pitch label associated to the wrong octave with respect to the rest of the dataset. These samples has been discarded as well. A dataset split policy of 80-10-10 was used with 708 samples for training, 88 for validation and 88 for test for each timbral class. Indeed, the total number of samples in the train set is 2832, 352 for validation and 352 for test. The input dataset consists in the same 708 samples of flutes concatenated with themselves four times. Following the notation proposed for the concatenating matrices and vectors in section 4.2.2 we have:

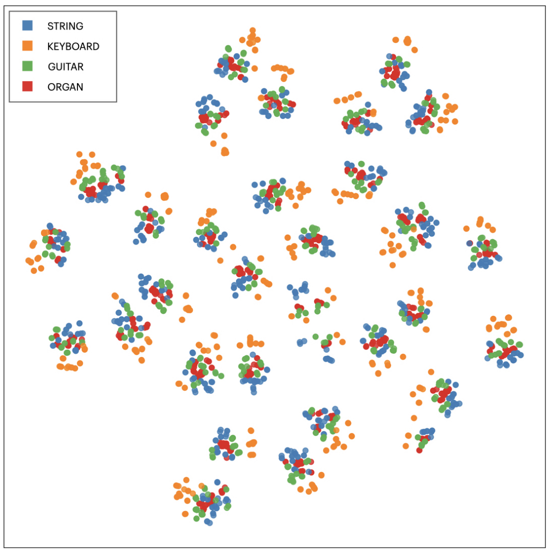
- $c = 0 \rightarrow \mathbf{C}^0_{512\times256}$ and $\mathbf{h}^0 = [1, 0, 0, 0]^T$ for the inputs whose timbre will be transferred from *flute acoustic* to *string acoustic*;
- $c = 1 \rightarrow \mathbf{C}^1_{512\times256}$ and $\mathbf{h}^1 = [0, 1, 0, 0]^T$ for the inputs whose timbre will be transferred from *flute acoustic* to *keyboard acoustic*;
- $c = 2 \rightarrow \mathbf{C}^2_{512\times256}$ and $\mathbf{h}^2 = [0, 0, 1, 0]^T$ for the inputs whose timbre will be transferred from *flute acoustic* to *guitar acoustic*;
- $c = 3 \rightarrow \mathbf{C}^3_{512\times256}$ and $\mathbf{h}^3 = [0, 0, 0, 1]^T$ for the inputs whose timbre will be transferred from *flute acoustic* to *organ electronic.*

In the training phase the input-output spectrogram couples have the same pitch value enabling the automatic clustering of both timbre and pitch. In order to do that we built the dataset so it has the same number of samples for each pitch, that, in our case, can have values that goes from 68 to 100, always following the MIDI notation.

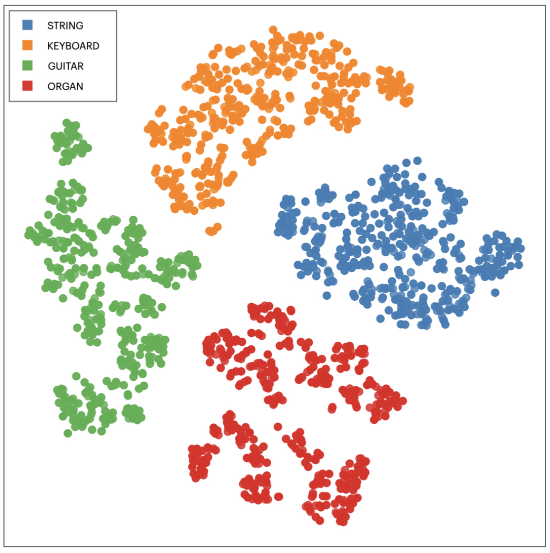
5.2. Latent Space Topology

As a preliminary result, we bring a visual inspection of the latent space of the trained architecture. The latent space is actually bipartite: VAE architectures [32] map the input data into a latent space consisting of two 1-D layers, representing the encoded normal

distributions, so the encoder is trained to return the mean ($\boldsymbol{\mu}$) and the covariance ($\boldsymbol{\sigma}$) that describe these Gaussians.
To visualize the latent space, we used t-SNE dimensionality reduction [63] on the 64-D $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ vectors representing the training set embeddings, respectively shown in Fig. 5.1a and Fig. 5.1b. As we can see, a peculiar form of clustering happens. In the t-SNE representation of the $\boldsymbol{\mu}$ latent vectors there are 33 clusters representing the 33 pitches ranging from 68 to 100 used in the training set. In the t-SNE representation of the $\boldsymbol{\sigma}$ latent vectors, we can clearly identify 4 clusters associated with the four timbre classes *string*, *keyboard*, *guitar* and *organ*. From this, we infer that the architecture perform automatically in the latent space a pitch clustering in the `mu` layer and a timbre clustering associated to the `log_variance` layer. This result enables to navigate the latent space in both dimensions, namely moving in a *timbre space* by sampling the $\boldsymbol{\sigma}$ latent space but also moving in a *pitch space* by sampling the $\boldsymbol{\mu}$ latent space having conditioned only the timbre information in the training phase.



(a) $\boldsymbol{\mu}$ latent space



(b) $\boldsymbol{\sigma}$ latent space

Figure 5.1: Latent space visualization obtained with t-SNE dimensionality reduction.

One of the applications possible thanks to the topology of the latent space is the one of building **timbral grids**. By performing timbre transfer on a given flute normalized log spectrogram $\mathbf{S}^f$, we obtain 4 generated normalized log spectrograms with the new timbres $\mathbf{S}^s$, $\mathbf{S}^k$, $\mathbf{S}^g$ and $\mathbf{S}^o$. By performing timbre interpolation between those points, defined as:

- $\mathbf{I}_{s\rightarrow k}$: interpolation sequence that starts from *string* sample $\mathbf{S}^s$ and ends up in *keyboard* sample $\mathbf{S}^k$;
- $\mathbf{I}_{k\rightarrow g}$: interpolation sequence that starts from *keyboard* sample $\mathbf{S}^k$ and ends up in

guitar sample $\mathbf{S}^g$;

- $\mathbf{I}_{g\to o}$: interpolation sequence that starts from *guitar* sample $\mathbf{S}^g$ and ends up in *organ* sample $\mathbf{S}^o$;
- $\mathbf{I}_{o\to s}$: interpolation sequence that starts from *organ* sample $\mathbf{S}^o$ and ends up in *string* sample $\mathbf{S}^s$;

and concatenating them, we can obtain a "circular" sequence of generated samples that explores all the intermediate interpolation points having new timbres between each couple of instruments' timbres, forming a timbral path between all target timbre classes.
A graphical representation of a timbral path obtained with $m = 5$ is shown in figure 5.2 where we arrange the spectrograms in the form of a square with vertexes indicating the starting points of the interpolations.
Note that each interpolation sequence is symmetrical, namely given two timbral classes a and b, the interpolations $\mathbf{I}_{a\to b}$ and $\mathbf{I}_{b\to a}$ lead to the same interpolation points.

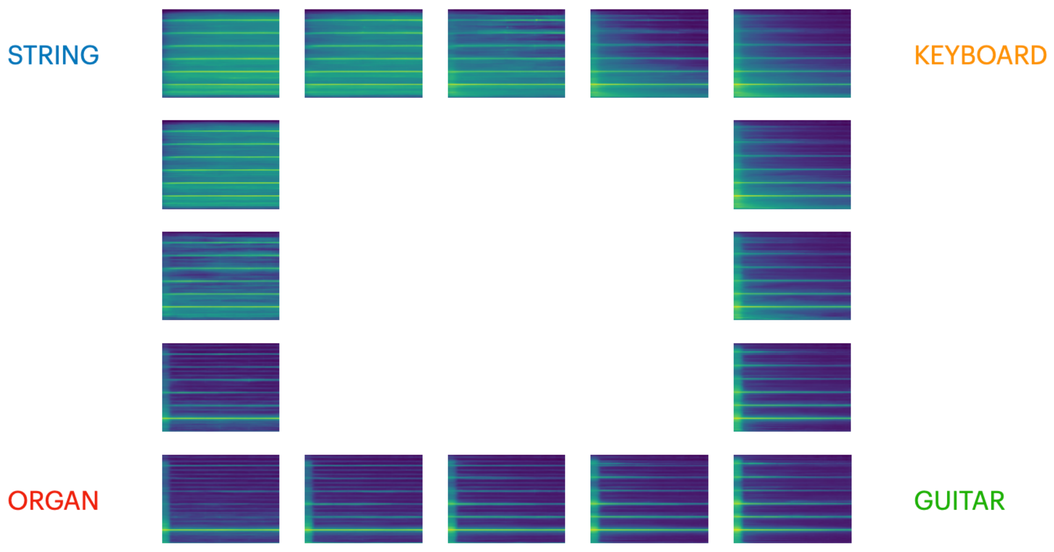


Figure 5.2: Interpolation path between predictions generated performing timbre transfer task, represented by the vertexes of the square.

We can generate new interpolation samples by saving the embeddings of the interpolation points calculated before and performing again interpolation between them by applying algorithm 4.1 to the new embeddings. By doing this we achieve what we call interpolation grids on the output of the network. By doing that we can generate samples with timbres that are a blend between all the output timbres of the network. In figure 5.3 we expose an interpolation grid of the outcomes of the network when given as input a flute sample with pitch 81.

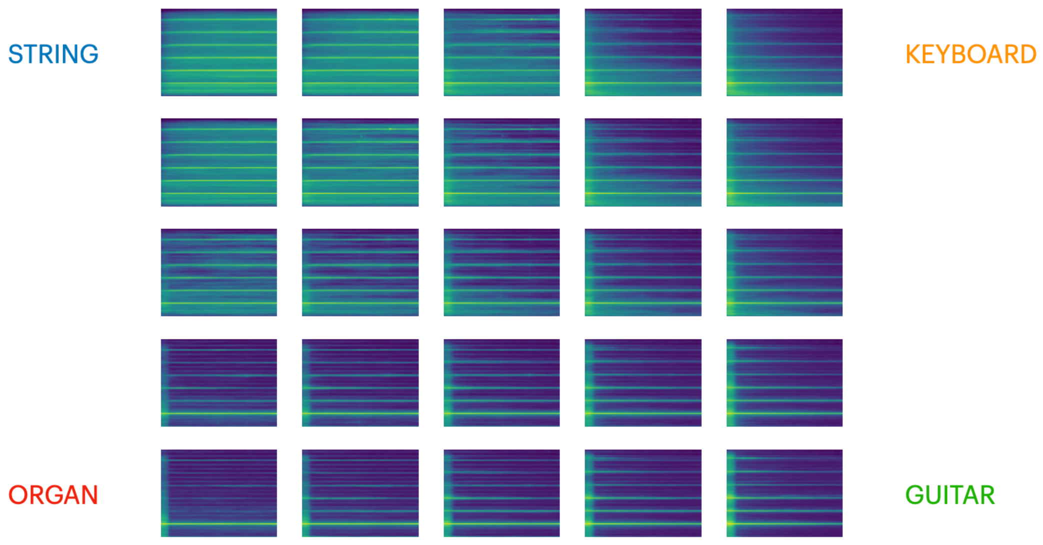


Figure 5.3: Interpolation grid between predictions.

5.3. Evaluation Methods

In this section we will present the methodologies to evaluate the outcomes of our model. The main aspects to be evaluated are:

- the quality of the output audio per se, namely the capability of the system to synthesize new samples that are acoustically pleasant and that resemble the ones present in the dataset;
- the capacity of the model to build a regularized latent space whose internal exploration leads to meaningful outcomes.

In order to do that, a twofold system evaluation method has been set that combines an objective assessment based on a timbral classifier and a perceptual one based on subjective ratings gathered with a questionnaire. The perceptual evaluation is needed since the timbre of a sound is primarily a perceptual characteristic that can be judged in its integrity only by a human response.

5.3.1. Evaluation by Classification

The timbre classifier has the following objective: given as input a normalized log spectrogram of an audio sample, it will classify it as belonging to one over a set of timbral classes, namely what we defined as target timbres. In our specific experiment, we will have as classes: *string*, *keyboard*, *guitar* and *organ*.

Considering the classifier as a function C that takes as input a normalized log spectrogram $\mathbf{S}$, it will output its timbral class, namely $t \in 0, 1, 2, 3$ where we mapped the four output classes of choice as $0 \rightarrow$ *String*, $1 \rightarrow$ *Keyboard*, $2 \rightarrow$ *Guitar* and $3 \rightarrow$ *Organ*:

$$t = C(\mathbf{S}). \tag{5.1}$$

Formally, the model is a supervised classifier based on a CNN. The model has been trained on the same dataset used for timbre transfer task with the sole difference that we used only the samples of the output classes, namely *string acoustic*, *keyboard acoustic*, *guitar acoustic* and *organ electronic*. The model has been trained for 20 epochs reaching a train accuracy of 0.9979 and a validation accuracy of 1. The perfect validation accuracy is due to the smallness of the validation set, consisting of only 88 samples, and it is used only as benchmark to validate the capabilities of the classifier. The structure of the classifier is exposed in table 5.1.

Layer type	**Output shape**	**# params**
`Input`	$(512 \times 256 \times 1)$	0
`Conv2D`	$(256 \times 128 \times 32)$	320
`MaxPooling2D`	$(128 \times 64 \times 32)$	0
`Conv2D`	$(128 \times 64 \times 64)$	18496
`MaxPooling2D`	$(64 \times 32 \times 64)$	0
`Flatten`	131072	0
`Dense`	128	16777344
`Dropout`	128	0
`Dense`	1	516

Table 5.1: Timbre classifier architecture

This evaluation method is used to analyze the capacity of the network to generate new samples that follow the condition given as input. Each generated normalized log spectrogram is labelled based on its conditioning and passed to the classifier to be evaluated.

The classifier can be extended in its functionality by adding a `Softmax` layer that, for each normalized log spectrogram given as input, outputs a vector of four probabilities of the input to belong to each of the four target timbral classes. Formally, defining p_0 as the probability of the normalized log spectrogram to belong to the timbral class *String*, p_1 as the probability to belong to the timbral class *Keyboard*, p_2 as the probability to belong to the timbral class *Guitar*, p_3 as the probability to belong to the timbral class *Organ* and defining the extended classifier as a function P that takes as input a normalized log spectrogram $\mathbf{S}$, the output can be defined with the equation 5.2:

$$[p_0, p_1, p_2, p_3]^T = P(\mathbf{S}). \tag{5.2}$$

This vector can be used to evaluate the samples generated during the timbre interpolation task. Ideally given a starting timbre t_a and an ending timbre t_b and having $m \in \mathbb{N}$ ordered generated interpolation going from timbre t_a to timbre t_b, the values of the probabilities of belonging to the class associated to the timbre t_a should follow a linear decay from 1 to 0 when analyzing the interpolation points in order. Vice versa, the values of the probabilities of belonging to the class associated to the timbre t_b should follow a linear ascend from 0 to 1.

5.3.2. Evaluation by Perceptual Tests

Subjects. Forty-three subjects, ranging from 24 to 36 years participated to the perceptual evaluation with a mean age of 26.3. This group included 33 males, 9 females and a subject that didn't want to express the gender. Despite no particular musical background was required to do the test, the subjects declared to have different musical backgrounds: 34.9% of them have a "low-level" backgound meaning that they have no experience of playing an instrument nor composing music. 48.8% have a "medium-level" background having played one or more instrument or composed music in their lives, not having doing it on a professional level and not having a music related degree. Finally 16.3% have "high-level" background, meaning that they have a conservatory degree in one or more instrument or in musical composition or they have played at a professional level.

Questionnaire. The questionnaire was divided in two parts. In the first one, the subject is made to listen to 20 (5 strings, 5 keyboards, 5 guitars and 5 organs) couples of samples consisting each on a ground truth audio sample directly taken from the dataset, followed

by 2 seconds of silence and a generated sample with the same pitch and belonging to the same timbral class. The subject is asked to rate the similarity of the two audio samples on a five point Likert scale [30] where the value 1 corresponds to "not similar" and 5 to "identical".
In the second one, we perceptually evaluate timbre interpolation. The subject is made to listen to 12 triplets of generated samples. Each triplet consists of three generated samples: the first one representing the starting point, the second one the interpolation and the third one the ending point. Each sample is spaced one second apart from the others. Since in our experiment we perform interpolation having $m = 5$, we obtain five interpolation points where the first one is exactly the starting point of the interpolation and the last one is the ending point. We denominated the interpolation points as 1, 2, 3, 4 and 5 where the point 1 represents the starting point, point 5 the ending point and the points 2, 3, 4 the points in between in the latent space, calculated following the Algorithm 4.1.
We asked the subject if the timbre of the sample chronologically in the middle (the second one), that represent the actual interpolation points between the starting point and the ending point of the interpolation, is more similar to the timbre of the first sample or to the timbre of the last one in a Likert scale that ranges from 1 to 5 where 1 means "the timbre is identical to the one of the first sample" and 5 means "the timbre is identical to the one of the last sample". The first 3 triplets represent, in order, point 2, point 3 and point 4 of the interpolation that goes from *keyboard* samples to the corresponding *string* ones. Following this pattern, triplets 4, 5 and 6 represent the interpolations from *keyboard* to *guitar*, triplets 7, 8 and 9 from *guitar* to *organ* and finally triplets 10, 11 and 12 from *string* to *organ*.

Procedure. The experimental session had, for each of the two parts of the questionnaire, a familiarization phase. For the first part, the subject is made to listen to 8 examples of couple of ground truth-generated notes, similar to the ones he/she would be listening in the actual part one of the test. For the second part, the subject is made to listen to 4 examples of triplets of generated samples, similar to the ones he/she would be listening in the actual part two of the test. The questionnaire has been hosted by Google Forms [1]. A disclaimer was presented at the beginning of the test, recommending the subjects to use a good sound system since the quality of the audio is a central factor of the study.

[1]`https://docs.google.com/forms/d/1YdG0VSc6kMY8NTuv_XBkl3liMYpBOWZ50uwkRhSf0hI/edit`

5.4. Results

In this section we describe the results obtained from the experimental setup just explained. We first present the timbre transfer results with a confusion matrix representing the outcomes of the classifier and with a box-plot associated to the scores gathered in the perceptual test.
Then we present the results on the timbre interpolation task in the form of graphics showing probabilities of belonging to the starting point class and to the ending point class for each of the mid-way points constituting the interpolation and finally with a box-plot with the perceptual scores grouped by interpolation point.

5.4.1. Timbre Transfer

The first evaluation of the timbre transfer outcomes obtained in our experiment happens via classification. As explained in section 5.3.1, we trained a classifier to discern between the 4 target timbre classes used in the experiment. Figure 5.4 shows the results of the classification on the test set outcomes of the network in a form of confusion matrix. As we can see, we obtained perfect classification for the classes *string*, *keyboard* and *guitar* and a single misclassified sample for the *organ* class, interpreted as a *guitar* sample by the network. This result indicates that the outcomes the network are properly generated, meaning that the model has learned to discern between timbral classes and that the conditioning of the input leads to the expected timbral outcomes.

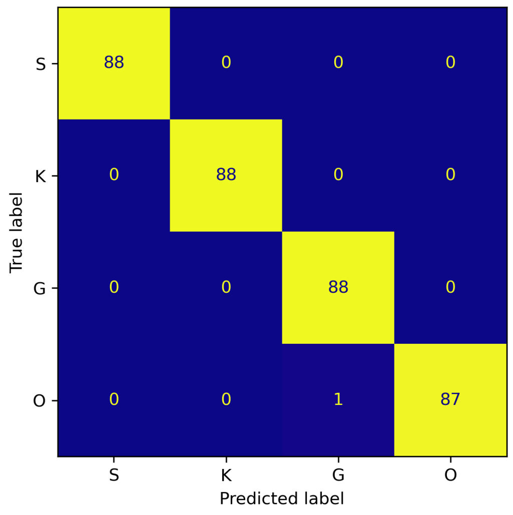


Figure 5.4: Confusion matrix on timbre transfer predictions. S → *String*, K → *Keyboard*, G → *Guitar*, O → *Organ*.

The first part of the perceptual test is dedicated to the evaluation of timbre transfer, asking the subjects how the generated and ground truth samples' timbre sound alike. The results have been aggregated by timbral class, in order to inspect the quality of the audio reconstruction depending on the class. As we can see from the box-plots shown in Figure 5.5, we have quite uniform results, with the *keyboard* class reaching the best score. Since the *string* class is actually composed by recordings of different instruments (violins, violas, cellos) played with different techniques such as legato, détaché and staccato, the architecture has a harder time modeling the class, justifying the lowest score of the class. The white triangle represent the mean score obtained for each class.

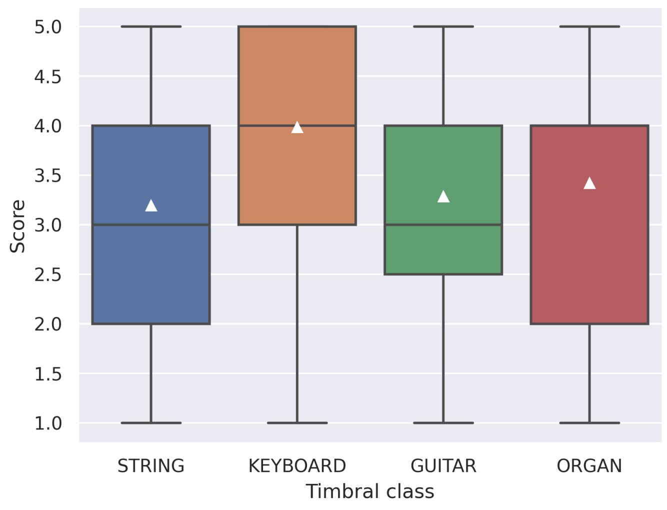


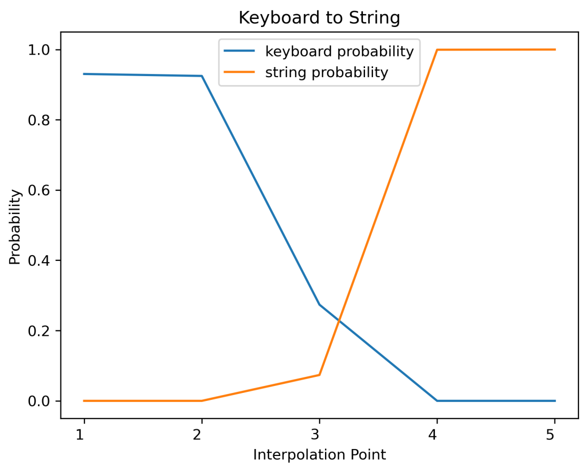
Figure 5.5: Perceptual ratings for timbre transfer task.

Both the objective evaluation based on classification and the perceptual test scores confirm that the model is capable of performing one-to-many timbre transfer. The confusion matrix shown in Fig. 5.4 expounds the classification on the direct outcomes of the network namely the frequency-time representation of the actual audio samples. What is evaluated performing classification is indeed the capability of the model to generate normalized log spectrograms belonging to a precise timbral class. On the contrary, the perceptual test measures the outcomes in a qualitative way: the subject is asked to rate the similarity of timbre between two samples belonging to the same timbral class, one "real" and the other one generated. In this case we are actually evaluating the quality of the generated samples as perceptually similar to the ground truth one that is clearly a different task from the one carried by the classifier since the subjects are not asked to discern among timbre classes. The combination of these two evaluation methodologies is essential for a proper discussion of the results: by investigating two complementary system evaluations we can have a full perspective on the ability of the model to perform timbre transfer.

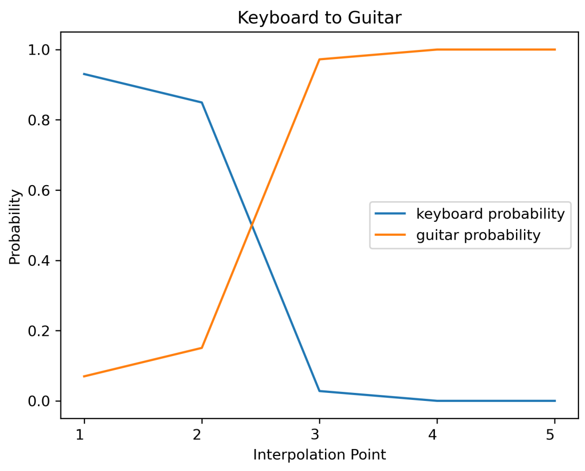
5.4.2. Timbre Interpolation

The first evaluation of the timbre interpolation is also obtained via classification. This time, we get the probability of the outcomes of the network to belong to each one of the 4 timbral classes for each interpolation. With $m = 5$, as explained in section 5.3.2 we get 5 interpolation points called 1, 2, 3, 4 and 5.
To visualize the results, given a generic interpolation that goes from timbre t_a to timbre t_b, we set up a graphic with the evolution of probabilities of the interpolation points (represented on the x-axis, where the left end represent point 1 and the right end point 5) to belong to the starting timbre t_a (blue line) and the ending timbre t_b (orange line). We show these plots for all the interpolations aggregated in Figure 5.6. From these plot we infer the fact that, while the point 3 is averagely in the middle between the two classes, the points 2 and 4 tend to be classified with a strong confidence as belonging respectively to the starting timbre class and to the ending one. This could be happening because of the low generalization capabilities of the classifier that, since it is trained on our dataset, does not have great potential with new unseen data. For this reason also we decided to do the perceptual tests.

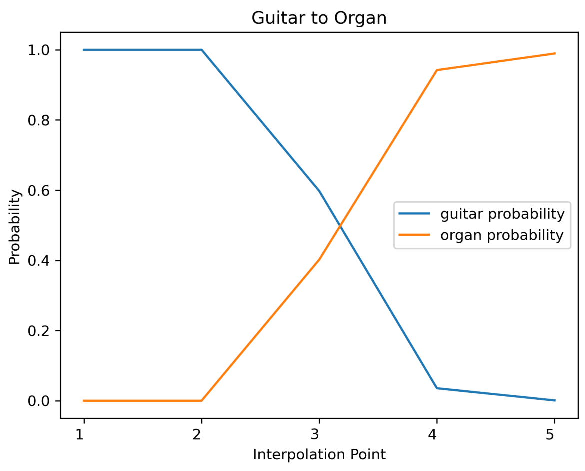
The quality of the timbre interpolations has been evaluated in the second part of the perceptual test where we asked the subject to rate the similarity, in terms of timbre, between a generated interpolation point and its correspondent starting point and ending point of the interpolation. In this case, the results has been aggregated for all points 2, 3 and 4. Figure 5.7 shows the box plots with the results. The scores reveal that perceptually the interpolation works, having the interpolating points matching the scores given by the subjects. Ideally the interpolation point number should match the score since the number of interpolation points matches with the score range both numerically and in meaning. For example all points 2 should have a score of 2 since that score represents the right spot in the interpolation, being closer to point 1 than point 5 of the interpolation. We can notice that the point three has a mean score slightly under 3, while point 2 and point 4 have values respectively near the starting timbre (score $= 1$) and the ending timbre (score $= 5$). A possible explanation for this fact is derived from the topology of the latent space: two points with different timbres are well separated in the latent space, as depicted in Figure 5.1b but the path going from one to the other could lead to a non-smooth evolution.



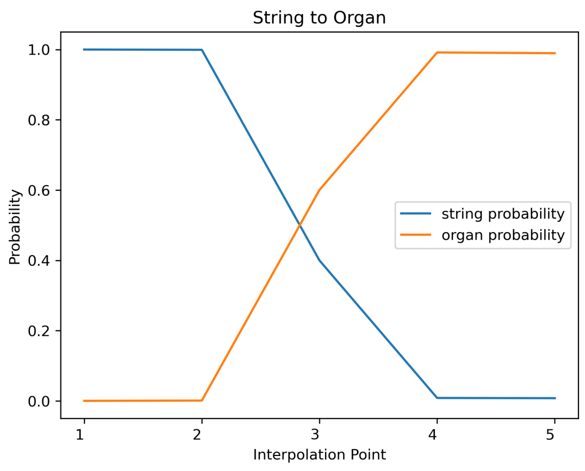
(a) Keyboard to String



(b) Keyboard to Guitar



(c) Guitar to Organ



(d) String to Organ

Figure 5.6: Classification on interpolations

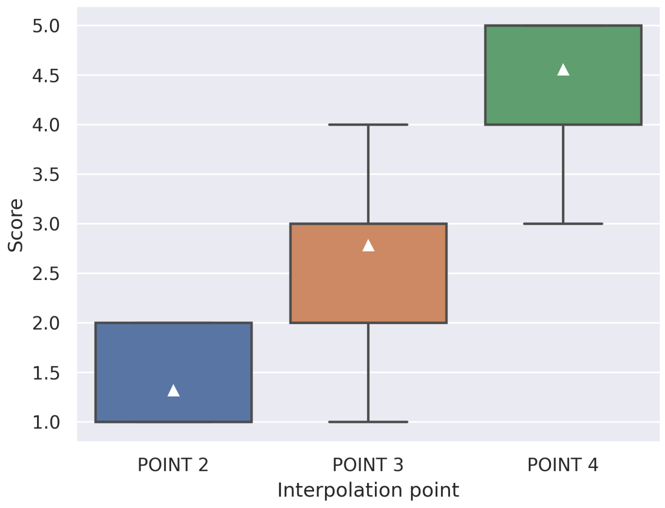


Figure 5.7: Perceptual ratings for timbre transfer task.

Also in the case of timbre interpolation task, the combination of objective and perceptual evaluation is required. The obtained results from these two evaluation methods are clearly correlated. In both cases the results demonstrate that the system is capable of performing timbre interpolation with certain limitations. When we performed the evaluation based on the classifier, we noticed that the interpolation points near to the extremes of the interpolations i.e. points 2 and 4 were strongly classified as belonging respectively to the starting point and to the ending point of the interpolation. For this reason we set up a perceptual test that, in fact, confirms the outcomes of the classifier with less accentuated scores: points 2 and 4 are still near respectively to points 1 and 5 but with less intensity with respect to the classifier's results. This clarifies the functioning of the classifier that probably is unable to classify unseen data, except when the normalized log spectrogram is a clear morphing among two given classes, which happens when taking into exam point 3, the mid point of the interpolation.

5.5. Conclusive Remarks

In this chapter we first presented the experimental setup of the work giving all the technical details of the implementation, starting from the time-frequency representation of the signal we adopted, passing through the network parameters and ending on the hardware and software technologies used to host the experiments. After that we presented the dataset we built for the task, describing it and mentioning its origin. In the section after, we outlined the latent space topology with the help of t-SNE dimensionality reduction, showing the clustering capabilities of our model in both pitch and timbre latent spaces. Evaluation methods are described immediately afterwards, divided by methodology. We exposed the design of the classifier we used and we explained how we trained it. We then delineated the specifics of the perceptual test we adopted. Finally we presented the results on both timbre transfer and timbre interpolation.

6 | Conclusions and future developments

In this thesis we have developed a Deep Learning based method able to perform one-to-many timbre transfer on music signal representing single notes, creating a navigable latent space of features that permits timbre interpolation among a set of pre-defined timbral classes. We used the Deep Learning paradigm following the state-of-the-art works making use of generative models. VAE architectures have proven to be effective in accomplishing the timbre transfer task [13, 39, 46, 56, 60] and can be exploited for the creation of a latent space that represents input samples in terms of timbral characteristics.
More specifically, we adopted a Conditional Convolutional beta-VAE architecture. The conditioning of the input is useful in the training phase in order to build a regularized latent space required to perform both timbre transfer and timbre interpolation tasks. It is useful also in the inference phase, where we can specify the model which timbral class we want as output of the system when executing timbre transfer. We opted for a 2D Convolutional architecture since we work with spectrograms, i.e. time-frequency representation of signals.
To test the methodology, we used a subset of the NSYNTH dataset, accurately built by us by removing unwanted, noisy samples and normalizing the classes with regard to number of samples per pitch which is the same for each timbral class. Our dataset consists in one single timbral class as input, namely *flute acoustic* and 4 output timbral classes: *string acoustic*, *keyboard acoustic*, *guitar acoustic* and *organ electronic*. Operatively, in order to test the proposed method, we performed, one-to-four timbre transfer and timbre interpolation between the four output classes.
We evaluated the outcomes of the system for both timbre transfer and timbre interpolation tasks with a twofold evaluation method. A first objective assessment employs a timbre classifier that, in the case of the timbre transfer task, classifies the outcomes of the network as belonging to one over the 4 output timbral classes used, while, in the case of timbre interpolation task, outputs a four value vector with the probabilities of belonging to each of the four output timbral classes for each generated spectrogram. The second

evaluation method relies on perceptual tests. Forty-three subjects were asked to evaluate the outcomes of the network as follows: for the timbre transfer task we asked the subject to rate on a 5 points scale the quality of a signal over which timbre transfer is performed with respect to its ground-truth counterpart having the same timbre and pitch. For the timbre interpolation task, we asked to rate a generated interpolation sample's timbre as more similar to the starting timbre or the ending timbre of the interpolation on a 5 points scale.
The results show that the timbre transfer task reaches high quality outcomes. The classification on the test set outcomes reaches remarkable outcomes and the scores gathered with the perceptual tests fulfill our expectations. The timbre interpolation task also reaches quite satisfactory results: the output probabilities follow the wanted evolution along the interpolation points and the same happens with the perceptual test scores.

We can still apply some improvements to the proposed methodology. A bigger dataset with more samples and more timbral classes could enhance the properties of the latent space. More samples would increase generalization capabilities of the network, while having more timbral classes could lead to more possible movements inside the latent space, leading to more interpolated timbres to discover. Another possible improvement could be the one of implementing new interpolation techniques such as parabolic interpolation or spherical interpolation in addition to the one used by us that is linear. New interpolation techniques could lead to a smoother evolution of the interpolation points or generally to new outcomes exploring differently the latent space.

There are several future developments of the work here presented. First of all, the extension of this work for longer samples. This development of the system could lead to an actual useful software/plugin to be used by musical producers that could record a sample with a given instrument and then change its timbre or perform a timbre interpolation with the help of our model. Another possible development could be the adaptation of the system to perform timbre transfer between human voices in the field of deepfake generation. Also it would be interesting to apply this method to different music signals such as unvoiced sounds, for example drum samples and validate the outcomes with new different metrics.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

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

[3] M. Amodio and S. Krishnaswamy. Travelgan: Image-to-image translation by transformation vector learning. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, pages 8983–8992, 2019.

[4] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[5] L. Carol. Why is musical timbre so hard to.

[6] J. M. Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the audio engineering society*, 21(7):526–534, 1973.

[7] C. Chu, A. Zhmoginov, and M. Sandler. Cyclegan, a master of steganography. *arXiv preprint arXiv:1712.02950*, 2017.

[8] J. T. Colonel and S. Keene. Conditioning autoencoder latent spaces for real-time timbre interpolation and synthesis. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.

[9] X. Cui, V. Goel, and B. Kingsbury. Data augmentation for deep neural network acoustic modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(9):1469–1477, 2015.

[10] N. Dilokthanakul, P. A. Mediano, M. Garnelo, M. C. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

[11] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders, 2017.

[12] J. Engel, L. Hantrakul, C. Gu, and A. Roberts. Ddsp: Differentiable digital signal processing. *arXiv preprint arXiv:2001.04643*, 2020.

[13] P. Esling, A. Bitton, et al. Generative timbre spaces: regularizing variational autoencoders with perceptual metrics. *arXiv preprint arXiv:1805.08501*, 2018.

[14] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[15] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[16] J. Gillick, A. Roberts, J. Engel, D. Eck, and D. Bamman. Learning to groove with inverse sequence transformations. In *International Conference on Machine Learning*, pages 2269–2279. PMLR, 2019.

[17] F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural computation*, 7(2):219–269, 1995.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[19] J. M. Grey. Multidimensional perceptual scaling of musical timbres. *the Journal of the Acoustical Society of America*, 61(5):1270–1277, 1977.

[20] F. J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[21] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[22] R. Hennequin, A. Khlif, F. Voituret, and M. Moussallam. Spleeter: a fast and efficient music source separation tool with pre-trained models. *Journal of Open Source Software*, 5(50):2154, 2020.

[23] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[24] S. Huang, Q. Li, C. Anil, X. Bao, S. Oore, and R. B. Grosse. Timbretron: A wavenet (cyclegan (cqt (audio))) pipeline for musical timbre transfer. *arXiv preprint arXiv:1811.09620*, 2018.

[25] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[26] P. Iverson and C. L. Krumhansl. Isolating the dynamic attributes of musical timbrea. *The Journal of the acoustical society of America*, 94(5):2595–2603, 1993.

[27] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.

[28] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song. Neural style transfer: A review. *IEEE transactions on visualization and computer graphics*, 26(11):3365–3385, 2019.

[29] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[30] A. Joshi, S. Kale, S. Chandel, and D. K. Pal. Likert scale: Explored and explained. *British journal of applied science & technology*, 7(4):396, 2015.

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

[32] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[33] D. P. Kingma, S. Mohamed, D. Jimenez Rezende, and M. Welling. Semi-supervised learning with deep generative models. *Advances in neural information processing systems*, 27, 2014.

[34] D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015.

[35] A. Klapuri. Introduction to music transcription. In *Signal processing methods for music transcription*, pages 3–20. Springer, 2006.

[36] S. Lakatos. A common perceptual space for harmonic and percussive timbres. *Perception & psychophysics*, 62(7):1426–1439, 2000.

[37] K. J. Lee. *Computer evaluation of musical timbre transfer on drum tracks.* PhD thesis, 2021.

[38] M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised image-to-image translation networks. *Advances in neural information processing systems*, 30, 2017.

[39] Y.-J. Luo, K. Agres, and D. Herremans. Learning disentangled representations of timbre and pitch for musical instrument sounds using gaussian mixture variational autoencoders. *arXiv preprint arXiv:1906.08152*, 2019.

[40] S. McAdams, S. Winsberg, S. Donnadieu, G. De Soete, and J. Krimphoff. Perceptual scaling of synthesized musical timbres: Common dimensions, specificities, and latent subject classes. *Psychological research*, 58(3):177–192, 1995.

[41] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, pages 18–25. Citeseer, 2015.

[42] M. Pasini. Melgan-vc: Voice conversion and audio style transfer on arbitrarily long samples using spectrograms. *arXiv preprint arXiv:1910.03713*, 2019.

[43] M. Pérez-Enciso and L. Zingaretti. A guide for using deep learning for complex trait genomic prediction. genes (basel) 10: 553, 2019.

[44] C. J. Plack, A. J. Oxenham, and R. R. Fay. *Pitch: neural coding and perception*, volume 24. Springer Science & Business Media, 2006.

[45] L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[46] A. V. Puche and S. Lee. Caesynth: Real-time timbre interpolation and pitch control with conditional autoencoders. In *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2021.

[47] R. Reed and R. J. MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks.* Mit Press, 1999.

[48] J.-C. Risset and D. L. Wessel. Exploration of timbre by analysis and synthesis. In *The psychology of music*, pages 113–169. Elsevier, 1999.

[49] F. Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[51] R. Sammut Bonnici, C. Saitis, and M. Benning. Timbre transfer with variational auto encoding and cycle-consistent adversarial networks. *arXiv e-prints*, pages arXiv–2109, 2021.

[52] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan, et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4779–4783. IEEE, 2018.

[53] Y. Shen, J. Gu, X. Tang, and B. Zhou. Interpreting the latent space of gans for semantic face editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9243–9252, 2020.

[54] K. Siedenburg and S. McAdams. Four distinctions for the auditory “wastebasket” of timbre1. *Frontiers in psychology*, page 1747, 2017.

[55] J. O. Smith. Physical audio signal processing for virtual musical instruments and digital audio effects. *online book at http://ccrma. stanford. edu/jos/pasp/, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University*, 2010.

[56] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[57] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[58] J. F. Steffensen. *Interpolation.* Courier Corporation, 2006.

[59] S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *The journal of the acoustical society of america*, 8 (3):185–190, 1937.

[60] K. Tatar, D. Bisig, and P. Pasquier. Latent timbre synthesis. *Neural Computing and Applications*, 33(1):67–84, 2021.

[61] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, volume 1, page 4, 2016.

[62] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, 125:2, 2016.

[63] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

A | Appendix A

Appendix containing the number of pitches for train, validation and test set.

Pitch	Number of samples
068	23
069	20
070	22
071	22
072	22
073	23
074	17
075	23
076	18
077	26
078	28
079	23
080	31
081	25
082	24
083	31
084	24
085	22
086	23
087	22
088	22
089	22
090	21
091	23
092	23
093	24
094	23
095	23
096	24
097	11
098	8
099	9
100	7

Table A.1: Number of pitches for each timbral class of the train set

Pitch	Number of samples
068	3
069	3
070	3
071	3
072	3
073	3
074	3
075	3
076	3
077	3
078	3
079	3
080	3
081	3
082	3
083	3
084	3
085	3
086	3
087	3
088	4
089	4
090	3
091	3
092	3
093	4
094	3
095	2
096	2
097	0
098	0
099	0
100	0

Table A.2: Number of pitches for each timbral class of the validation set

Pitch	Number of samples
068	3
069	3
070	3
071	3
072	3
073	3
074	3
075	3
076	3
077	3
078	3
079	3
080	3
081	3
082	3
083	3
084	3
085	3
086	3
087	3
088	3
089	3
090	2
091	2
092	2
093	2
094	2
095	2
096	2
097	2
098	2
099	2
100	2

Table A.3: Number of pitches for each timbral class of the test set

List of Figures

List of Tables