Timbre modification using deep learning

Master thesis

Mattia Paterna

Aalborg University Copenhagen
Sound and Music Computing

This document has been designed in LaTeX. All the scripts have been implemented in *Python* programming language. *IPython* and *Jupyter Notebook* have been used to sketch part of the experimental setup. The simulations have been run in the *HPC* cluster at the *DCC computing center*, Danmarks Tekniske Universitet.

Sound and Music Computing
Aalborg University Copenhagen
http://www.aau.dk

Title:
Timbre modification using deep learning

Theme:
Master thesis

Project Period:
Spring Semester 2017

Project Group:
IX

Participant(s):
Mattia Paterna

Supervisor(s):
Hendrik Purwins

Copies: 1

Page Numbers: 53

Date of Completion:
July 14, 2017

Abstract:

This thesis introduces timbre transformations by means of deep learning. A set of convolutional autoencoders is created to deal with the task. Each structure uses *convolutional layers* as building blocks. First, a shallow architecture is used to perform the reconstruction of a series of piano notes and infer a set of optimal hyperparameters for the building blocks. Later, several architectures are deployed and compared in the attempt of transforming an input sound into a *target* sound. Doing so, two wind instrument sets are used. The input and the output of the deep structure are log-magnitude spectra of the audio signals. The Griffin-Lim algorithm is used for reconstructing phase information and generate an audio output using the outcome of the autoencoder. Results show that the convolutional autoencoder performs a fair job in the timbre transformation, especially when techniques, such as residual learning and dilation, are implemented. Moreover, constraints, such as sparsity, and regularisation helps in retrieving an optimal latent representation of the spectra.

Lyd og Musik
Aalborg Universitet København
http://www.aau.dk

Titel:
Timbre modification using deep learning

Tema:
Master Thesis

Projektperiode:
Efterårssemestret 2017

Projektgruppe:
IX

Deltager(e):
Mattia Paterna

Vejleder(e):
Hendrik Purwins

Oplagstal: 1

Sidetal: 53

Afleveringsdato:
14. juli 2017

Abstract:

This thesis introduces timbre transformations by means of deep learning. A set of convolutional autoencoders is created to deal with the task. Each structure uses *convolutional layers* as building blocks. First, a shallow architecture is used to perform the reconstruction of a series of piano notes and infer a set of optimal hyperparameters for the building blocks. Later, several architectures are deployed and compared in the attempt of transforming an input sound into a *target* sound. Doing so, two wind instrument sets are used. The input and the output of the deep structure are log-magnitude spectra of the audio signals. The Griffin-Lim algorithm is used for reconstructing phase information and generate an audio output using the outcome of the autoencoder. Results show that the convolutional autoencoder performs a fair job in the timbre transformation, especially when techniques, such as residual learning and dilation, are implemented. Moreover, constraints, such as sparsity, and regularisation helps in retrieving an optimal latent representation of the spectra.

Contents

Preface

This master thesis has been prepared in partial fulfilment of the requirement for the Master thesis in Sound and Music Computing, 4th semester in the Academic year 2016-2017

Aalborg University, July 14, 2017

Mattia Paterna
<mpater15@student.aau.dk>

Acknowledgement

I spent almost four months in doing this thesis, and some people deserve my gratitude. Some helped me in my technical issues, others helped me in keeping the sanity and in my everyday issues.

I would like to thank Corey Kereliuk, whom I met in Copenhagen at the very beginning and helped me in refining the very first draft of my idea. I would also like Paolo Galeone, who replied to many mails of mine with great details and exchanging knowledge about autoencoders that literally enlightened me.

Many thanks I give to Carmine-Emanuele Cella, who I've been pleased to know for some years. He pushed me further down this way and he helped me in learning how to think. His talks have been always valuable. Thanks to Hendrik Purwins, my supervisor, whom I spent almost one and a half year of supervision with. Undoubtedly, he has been the most important figure in my master and encouraged me in this steep way to the mathematical side of the music.

My thanks should I also give to DTU Computing Center and Sebastian Borchert, who carefully replied to my mails and helped me in running Tensorflow using the GPU clusters.

And then, I give my thanks to my friends, and classmates. Thank you to my ex workmates, who became good friends. Matteo, Adam, Devid. I spent lots of time, and lots of laughs with you. Thanks to Paolo and Roberta, who silently stood and listened to anything. I love your company.

And finally, my parents and my Gyulyuzar. To my parents: although you are quite far, your Sunday Skype made me feel home. If I'm here, it's undoubtedly because you always supported me. And to my Gyulyuzar: it's been beautiful and crazy working together on our theses, but now we shall rest a bit. You encourage me, no matter what. And when I wake up and see you, whatever the issue, I know that it will be a good day. I love you Gyulyuzar, I love you all.

And just a little dedication to my beloved dog, who I could not see once more in these two years and I could not unfortunately see for some time.

You are my sister and I love you. Ciao piccolina.

Chapter 1

Introduction

In the last decades, the use of interactive computer music systems has lead to performances in which musical notations and numerical representations of music can be directly converted to audio outcomes [39, 46]. Later, statistical models have proved to be able to merge both the processes of analysis and synthesis of music [12] and to interact with human performers in real time, *learning* from them [2].

Consequently, new ways of representing music, and audio, have emerged [36], in which properties of sounds, such as *timbre* and *pitch*, are generalised in the concept of *features*. Such representations aimed to be *hierarchical* and to exhibit a *multiscale* approach [15, 43]. In other words, they take into account several properties of sound which span through different levels of abstraction.

With the recent breakthrough of deep learning [28, 27], neural networks have been widely deployed for musical tasks, such as *annotation* [8] and *generation* [17]. Generative models have proved to successfully cope with the composition of meaningful musical structures from symbolic input [37, 10]. In the very recent time, another breakthrough appeared, with the introduction of a deep neural network that is capable of end-to-end audio generation [44]. At the same time, Google Brain has dedicated an entire project for the deployment of machine learning in the compelling creation of music[1].

Given the premise, a further exploration of neural network techniques for the generation of audio excerpts sounds particularly interesting. The generation increases in its interest, as the output of the generation is driven by the input musical characteristics. This thesis aims to give a contribution to this field.

In this chapter, I point out the importance of the representation in music through the latest advances in the field of *Music Information Retrieval*. Furthermore, I outline some of the recent techniques used for music generation. Then, I show how representation and generation can be merged in a deep learning paradigm and, from this, I state the research question for the thesis.

[1]`https://magenta.tensorflow.org/`

1.1 Representations in music

The need for representing music relates to early times, where no audio recording technology existed to recall music over temporal and spatial restriction. Thus, the creation of the musical score as a formal language [41] to depict a musical piece encoded in a high dimensionality [34]. The computer era has made possible the invention of various form of representation, among which MIDI is the most common symbolic representation. Nowadays, such representations are essential for any kind of musical generative system that makes use of computers.

There can be several ways of defining the properties of music, ranging from specific to abstract representations in technical systems, such as computers. Vinet introduced a distinction that goes further the traditional complementary way of representing music through a signal and symbolic representations, adding a physical and a knowledge representation [55]. Particularly, the latter has driven the last development in the field of Music Information Retrieval (MIR).

If considered stacked together, these representations span the whole processing of musical information by the brain, where the symbolic and the knowledge representation refer namely to the inference of musical features, e.g. *pitch* and *scales*, and the cognitive aspect, i.e. the creation of a formal language that carries out the description of musical phenomena [55].

1.1.1 Mid-level representations

However, converting audio signal to symbolic representations still lacks of accuracy. The concept of mid-level representations has been depicted in [18] to mitigate this gap and can be seen as a fixed mixture of both signal and symbolic level. In other words, the focus is on simple concepts that are more abstract than those related to a signal representation, such as the sinusoid. Mid-level representations can take place in the full network of representations described in [55], being these simple concepts in between the constraints imposed by lower and high levels.

1.1.2 Multiscale architectures

Recent developments in the MIR research have showed an increased interest in multiscale architectures [23], since music exhibits structures on many different time scales [15]. Indeed, if using frame-level features, i.e. extracted from short-time windows, no temporal information is retained. It can be therefore beneficial combine both low and high level aspect in a unique framework. Researcher have already explored them [15, 10], and some work shows architectures that are both *deep* and multiscale [36].

1.1.3 Deep learning for music content analysis

Recently, deep learning approaches have become of interest for feature learning and representation on audio classification tasks in supervised [43] and unsupervised [36] fashion. Kereliuk et al. [32] use deep learning for music genre recognition, since it is naturally suited to learn relevant abstractions for music content analysis. The key here is to build deep models that are capable of representing data at multiple level of abstraction, and to learn itself such representations. The layers in the deep learning model forms a hierarchy, where the innermost layer be the most *abstract* representation of the input data [14].

Representation learning The idea of moving from hand-crafted features to automatic feature learning is the key concept of *representation learning* [21]. In [31] the use of feature learning in MIR tasks have been advocated firstly, together with deep models. According to Goodfellow [21], learned representations yield to better performance minimising the human intervention. Furthermore, deep learning may solve the problem of extracting high-level features building them out of simpler and low-level representation. Hence, the reason for using a representation learning algorithm, such as the *autoencoder*.

1.2 Music generation

The last decades have seen an extensive application of computers in the generation and processing of musical content. An analytic model is usually followed by a synthetic model. In the middle, sound processing algorithms take place, depending on the aimed modifications, e.g. *pitch shifting*. The *phase vocoder* [16] is one of the most common techniques. However, computational methods for complex musical tasks, such as *style imitation* or *audio continuation*, have been far more difficult than initially imagined, leading to their replacement by hand-crafted algorithm [12].

1.2.1 The use of statistical models

In [12], Conklin stated that statistical model can be deployed for such complex tasks. Statistical models are created, which assign a probability to the musical objects considered, say *notes* or *excerpts*. In this scenario, generating music is equal to sampling from such models. In other words, statistical models provide the conditional probability distribution over a series of musical objects. In the case of *machine improvisation*, the distribution over a sequence of preceding events is then used for generating new sequences or computing the probability of a given one [2].

Markov models have been widely used for accomplishing the process of music generation [45, 37, 2, 10]. While the *Continuator* [45] and *OMax* [2] worked with MIDI sequences inside their architectures[2], in [37] and [10] audio events are directly processed and recombined in a generation of a new sequence. Particularly, the latter works show how audio events can be grouped by similarities using techniques, such as hierarchical clustering and Gaussian mixture models, that are applied to audio features specifically extracted from the original audio.

1.2.2 The use of deep learning

In recent years, many attempts have been done in using deep learning techniques for music generation. In [8], symbolic sequences of polyphonic music are modelled and generated. Particularly, the use of *recurrent neural networks* (RNN) have proved successful in generating sequences of notes and chords. Google Brain devoted an entire research project to advance the state-of-art in the deployment of machine learning in music, named Magenta. One of its most successful application has been a RNN that generates music from MIDI input sequences[3]. However, the representation is still limited to a high-dimensional level. Furthermore, RNNs showed much difficult to be trained.

Convolutional neural synthesizer *Wavenet* [44] has emerged last year, gaining the state-of-art performance in text-to-speech task. The peculiarity of such a model reside in the choice of using *dilated* convolutional neural networks only (no RNN is required) and *raw* audio waveforms as input, returning an audio waveform as output. A new breakthrough in the panorama of music generation has been achieved on May, 18th with the release of an instrument[4], which allows for sound morphing among several audio samples using Neural Audio Synthesis (NSynth) [19]. In other words, it takes several audio samples and gives the player the opportunity to create custom timbres modulating the amount of those selected sounds. Undoubtedly, this is quite a brand new synthetic model, which also works by finding an optimal latent representation of the original audio.

Hence, NSynth demonstrates that the analysis and the synthesis part can be merged together in a deep models where audio features are learnt directly by the network[5].

1.3 Objectives

On the basis of what I previously pointed out, in this thesis I aim to demonstrate that **it is feasible for a deep learning model to generate *target* output**

[2]Although, Assayag et al. developed a version of the system that is capable of working with monophonic acoustic instruments: Ofon.

[3]`https://magenta.tensorflow.org/2016/06/10/recurrent-neural-network-generation-tutorial`

[4]`https://magenta.tensorflow.org/nsynth-instrument`

[5]To be more precise, [19] and [44] make use of external conditioning to achieve the best results.

sounds, which are *others* than the sounds given as inputs to the model. Such a task appears different than the other tasks yet successfully deployed by means of WaveNet-based systems.

As a matter of fact, the NSynth application mentioned previously performs morphing among *compressed representations* of sounds, but it *does* not directly map the content of a given input sound to a targeted output sound. In other words, the network is trained to reconstruct an input sound at its best and the morphing is obtained by interpolating similar representations of different sounds at the innermost layer of the model. Conversely, I aim to create a model that, given an input sound, may successfully modify its characteristics and produce a different outcome.

To cope with limits of time and complexity, I concentrate on the *timbre* transformation between an input sound and a desired sound, namely a *target*. With the use of *convolutional autoencoders*, I aim to a reconstruction of the target sound, given the input sound. Given the nature of this task, the autoencoder should be capable of *learning* features from the input sound. Moreover, it should *map* such features in order to achieve the desired transformation.

This end-to-end learning of timbre transformation makes use of *log-magnitude* spectra as inputs for the autoencoder. Being a magnitude spectrum a time-frequency representation, they retain much of the audio original information and, at the same time, can be considered as sound *images*. Doing so, I expect that the use of convolutional neural networks be favourable to the task since much of their advance has interested the field of computer vision. On the other hand, I have to deal with the loss of phase information when moving from the signal representation. This implies the use of an algorithm for reconstructing the phase when moving back to audio.

Finally, I aim to the generation of an arrangement based on such an input sound, that is a new musical object with specific musical characteristics, of which the timbre is one. This generation is therefore *content*-based, and requires the comprehension of all the multiple levels of representations that define the input. Given the promising works described in the foregoing chapter, I argue that a deep structure can extract information enough to create a content-dependent arrangement.

1.4 Outline

In this chapter, I outlined the premises which this thesis relies on, stating the research question this thesis addresses.

In chapter 2, I provide an in-depth description of the theoretical background that I used in this work. Particular emphasis is given on the *autoencoder* paradigm and on its deployment together with *neural networks*.

In chapter 3, I outline the experimental framework used in this thesis. I give a description of the architectures, explaining and motivating the choice of techniques, such as *non-linearities*, *learning rate methods* and *optimisation*. I then describe the experiment and give detail about the training specifics. I finally provide results and discuss the main outcomes. During the discussion, I unfold them using the theory and literature provided.

In chapter 4, I give a conclusion to this thesis showing the impact of the results in the field of research and reflecting on possible limitations that I encountered during this work, and on the further development of the topic.

Chapter 2

Theoretical background

This chapter aims to provide a theoretical explanation of some of the techniques implemented in this work, describing *deep* architectures and narrows down to a particular set of them, the *convolutional autoencoders*.

2.1 Deep architectures

This work mostly concentrates on the use of deep architectures. As described in [5] a deep architecture allows for the composition of *non-linearities*, which can optimally describe complex functions.

Traditionally, a distinction is made between *shallow* and *deep* architectures. A shallow architecture presents only two levels of data-dependent computational elements [5], i.e. there are no intermediate stages between the input and the output[1]. The main problem that lies in a shallow architecture is the inefficiency in terms of computational units, i.e. the pieces used to represent the input. In other words, the representation can require a number of pieces that grows exponentially with the complexity of the input function, thus leading to the problem of the *curse* of dimensionality.

On the other hand, a deep architecture is made of several layers stacked on top of each other [38] in a way such that the output of one layer is used as the input of the successive one. Each layer is generally associated with a *non-linearity*, i.e. a mathematical function that is able to display a wide range of variations of the input function [6]. The term *variations* here is meant to describe all the set of variables that can be observed from the input and that are often related by *unknown* statistical relationship [6], being the input to the learning system a high-dimensional entity. Thus, one of the major aims of deep architectures is to discover such relationships with little human effort, or none if possible.

[1]Although in [6] a definition of shallow is generalised to a whole bunch of learning algorithms with a number of levels between 1 and 3.

The complexity lies in the number of possible intermediate representations that span between the *raw* input, e.g. pixels of an image, and the *feature* that has to be inferred from it, e.g. a category. Advancing from one representation to the other the level of *abstraction* rises, that is the connection to the input gets remote. In a deep architecture, a given input is represented at multiple levels of abstraction in which features at a higher level rely on the composition of lower level features. In other words, *feature hierarchies* are learnt automatically and the mapping from the input to the output is directly inferred from data with no human hand-crafted features dependency [6].

Deep architectures are not a novel concept in the field of neural network, since the belief of an efficient modelling of complex relationship by means of the composition of several levels of non linearity has prospected almost 30 years ago [26]. A recent revival of interest is due to the discovery of approaches [27, 28, 5] that could end the issue of learning deep structure parameters. Among them, the *autoencoder* is one of such successful approaches.

2.2 The autoencoder paradigm

The idea of autoencoders has been circulating in the field of neural network for decades [21]. The appearance of autoencoders dates back to 1980s to address the problem of backpropagation *without a teacher* [3]. From this, autoencoders have provided one of the fundamental paradigm in the field of unsupervised learning.

Traditionally, autoencoders have proved to successfully perform dimensionality reduction tasks [28, 52]. Particularly, in DeMers' seminal work [13] a method for creating *non-linear encoder-decoder* structures is presented as a non-linear counterpart of the Principal Component Analysis. In a sense, the need for deep, i.e. multi-layer, autoencoders came in an early stage to improve yet existing classic dimensionality reduction techniques. A quite common practice results in training the autoencoder and discard the decoder part, so to have a structure capable of build a new dataset with a lower dimensionality.

A first proposal on autoencoder-based deep architectures has been made in [5] with the purpose of facilitating the training of the deep architectures described in 2.1. After the breakthrough of Hinton et al. [27] which gave researchers a first successful method to train deep neural network, new algorithms for deep architecture were proposed with respect to the idea of *training the intermediate level using unsupervised learning* [6]. In this context, autoencoders have been mainly used to initialise *deep supervised* feedforward neural network for specific tasks, such as regression and classification [53, 5] to increase the performance stability.

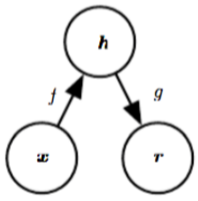

Figure 2.1: The general structure of an autoencoder, mapping an input x to an output through an internal representation, or code. Image from [21].

2.2.1 Definition

In [57], an autoencoder is described as

> a network [that] uses a set of recognition weights to convert an input vector into a code vector [and] then uses a set of generative weights to convert the code vector into an approximate reconstruction of the input vector.

In other words, a first stage maps the input to a lower-dimensional space (the *encoder*) and a second stage expand the code in the attempt to reproduce the original data (the *decoder*). The code vector is generally the result of the innermost layer, which is often referred as the *bottleneck* layer because of its reduced number of units. Several architectures can be used to build an autoencoder, thus leading to a more general definition of an *encoder-decoder paradigm* [38, 54].

More formally, an autoencoder typically consists of two parts: an *encoder* function $h = f(x)$ and a *decoder* function that performs the reconstruction $r = g(h)$ of a given input $x \in \mathcal{R}^d$. The internal representation h contains the *code*, i.e. the latent representation of the raw input. Figure 2.1 shows a general structure.

2.2.2 Network constraints

At a first glance, an autoencoder seems to learn the set $g(f(x))$, that means it learns an approximation of the identity function. One way of making the model be unable to learn the copy is to place *constraints* on the network. The model is thus forced to learn which aspects to prioritise and therefore learn useful properties of data, which allow for a proper *reconstruction* of it.

Hidden unit limitation The simplest yet effective limitation of an autoencoder is to constrain the *bottleneck* layer h to have smaller dimension than the input. Such an autoencoder is called *undercomplete*. Conversely, an *overcomplete* autoencoder lets the hidden code be *at least* not smaller than the input [21]. This way, the network is forced to learn a compressed representation of the input such that the algorithm can discover patterns and correlations within the input features. The theoretical scenario is an autoencoder with a one-dimensional code i learnt from each training example x_i and a decoder that maps back to the specific values. The learning process occurs by minimising a loss function

$$L(x, g(f(x))) \tag{2.1}$$

where $g(f(x))$ is penalised for being dissimilar from the input x [21].

Regularisation If the encoder and decoder have a large number of units no useful about data structure is learnt [21]. That applies also to overcomplete autoencoders. However, a method exists which gives the autoencoder the ability to learn information although no strict limit to the model capacity is imposed. Such a method is called *regularisation* and leads to the *regularised* autoencoders.

Such autoencoders make use of a loss function that allows the model to have properties, such as the *sparsity* of the representation.

2.2.3 Sparse autoencoders

A *sparse* autoencoder is basically an autoencoder whose training criterion involves a so-called *sparsity penalty* on the latent representation $\Omega(h)$ added to the reconstruction error given by the loss function:

$$L(x, g(h)) + \Omega(h) \tag{2.2}$$

where $h = f(x)$ is the encoder and $g(h)$ is the decoder output.

2.3 Autoencoder neural networks

Because of the non-linearities and the feedworfard *layer-wise* structure, it is common for the encoder and decoder to be parametric functions, such as neural networks. Doing so, they can be differentiable with respect to a distance function (the *mean squared error*, for instance), so that the parameters of the encoding/decoding functions can optimally minimise the reconstruction loss function using traditional neural network techniques, such as gradient descent.

Given the premise, an *autoencoder neural network* can be defined as an unsupervised learning algorithm that makes use of backpropagation algorithm and set the target (output) be equal to the input [42]. The simplest autoencoder neural network has a MLP-like (multi-layer perceptron) structure, with the difference that the output of the autoencoder has the same cardinality of the input.

This leads to a further definition of the autoencoder model, e.g. [38, 54]. The *encoder* is the transformation of the input vector $\mathbf{x}$ into a latent representation using a deterministic mapping $f_\theta(x)$ followed by a non-linearity:

$$\mathbf{h} = f_\theta(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.3}$$

where σ is the non-linearity and the parameter set is $\theta = (\mathbf{W},\mathbf{b})$. The matrix $\mathbf{W}$ is defined as the set of *dxd′ weights* and $\mathbf{b}$ is generally a d'-dimensional vector of biases.

The *code* provided in the innermost layer is then mapped back to a reconstruction vector $\mathbf{r}$ by a *reverse* mapping $g_{\theta'}(\mathbf{h})$:

$$\mathbf{r} = g_{\theta'}(\mathbf{h}) = \sigma_2(\mathbf{W'}\mathbf{h} + \mathbf{b'}) \tag{2.4}$$

where the mapping $g_{\theta'}(\mathbf{h})$ is referred to as the *decoder*, the non-linearity may be optional and the parameter set it $\theta' = (\mathbf{W},\mathbf{b})$. If we stack the equations all together, a formula for a standard autoencoder can be defined:

$$\boxed{f(\mathbf{x}) = \sigma_2(\mathbf{b'} + \mathbf{W'}\sigma(\mathbf{b} + \mathbf{W}\mathbf{x}))} \tag{2.5}$$

The two parameter sets are sometimes constrained to be $W' = W^T$, that means the weight matrices are tied. Doing so, both functions are sharing the same weights. If starting with random and tied weights in the two networks, they can be trained together [28]. During the training, a specific code $\mathbf{h}_i$ is created for each training pattern $\mathbf{x}_i$ and then the code is mapped to the specific reconstruction $\mathbf{r}_i$. The parameters are optimised by means of an appropriate *loss* function that is minimised over the unlabeled training set $\{x_1, x_2, ..., x_n\}$ [38, 42]. The required gradients are obtained by means of the *chain rule* to backpropagate through the decoder first, and the encoder then [28].

2.3.1 Single-layer autoencoders

The simplest architectures for autoencoders require just a single hidden layer between the data and the code and are called *shallow* autoencoders. In short, the total number of layers in the neural network are three, which are symmetrical with respect to the hidden one. As mentioned before, such feedforward forms recall closely the multi-layer perceptron (MLP) [38] (fig. 2.2). This is the essential structure of a deep learning model and is basically a mathematical function that maps an input to an output [21], being this function a composition of many simpler functions. Each of those simple functions provides a representation of the input. The mapping defined by the feedforward neural network is $\mathbf{y} = f(\mathbf{x}; \theta)$. The aim of the network is to learn optimal values of the parameter set θ so to yield to the best function approximation and minimise the loss function.

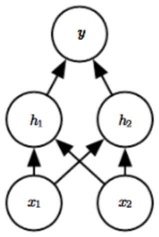

Figure 2.2: The MLP structure. Single-layer autoencoders are very similar except for the output layer, which has same dimensionality of the input. Image from [21].

2.3.2 Sparsity

Sparse representations have received much attention recently. The key is to impose on the autoencoder different constraints than that of a tight capacity [54]. That allows even for overcomplete structures. Some motivations for which to adopt sparse representations include a loosely *brain-inspired* neural activity, where most of the neurons are inactive.

If we think an autoencoder layer to be made of several *units*, i.e. the neurons, each unit has its associated *activation* value. Therefore, the sparsity constraint forces the code to have only few non-zero units while most of them are zero most of the time[2] [9]. More formally, if the average activation of a single unit is calculated:

$$\hat{\Omega}_j = \frac{1}{n}\sum_{i=1}^{n} a_j x_i \tag{2.6}$$

where a_j is the activation of the hidden unit j over the training set $\{x_1, x_2, ..., x_n\}$, and a sparsity parameter Ω is set, the sparsity constraint:

$$\hat{\Omega}_j = \Omega \tag{2.7}$$

keeps most of the hidden unit activation near to 0. To practically achieve this, a sparsity penalty is added as depicted in 2.2.3. As well explained in [42], the penalty term is often based on the concept of Kullback-Leibler (KL) divergence. For this case two distributions are considered, the former with mean Ω and the latter with mean $\hat{\Omega}_j$. Such a built penalty function has the property that $KL(\Omega \parallel \hat{\Omega}_j) \approx 0$ when the means equal each other.

Moreover, a coefficient β is sometimes added to control the weight of the sparse penalty, thus its influence on the loss function. The equation can be rewritten as

$$L(\mathbf{x}, g(\mathbf{h})) + \beta\Omega(h)KL(\Omega \parallel \hat{\Omega}_j) \tag{2.8}$$

[2]The assumption is that a *sigmoid* activation function is used, or a *ReLU*. If a *tanh* activation function is used, the neuron happens to be inactive if its output value is close to -1.

where both the loss function and $\hat{\Omega}_j$ depend on the parameter set $\theta' = \mathbf{W},\mathbf{b}$.

2.3.3 Other constraints

As described previously in this section, if the hidden layer has large capacity the autoencoder may easily learn the identity function. To avoid this, a *tied* autoencoder can be used, which has the formula

$$f(\mathbf{x}) = \sigma_2(\mathbf{b'} + \mathbf{W}^T\sigma(\mathbf{b} + \mathbf{W}\mathbf{x})) \tag{2.9}$$

and eliminates several degree of freedom in the autoencoder.

Additionally, a recent technique called *dropout* [49] has proved to give improvements. Dropout is a stochastic regularisation technique that applies to the activation functions. It basically drops units *randomly* from the neural network during the training, along with incoming and outcoming connections, preventing units from co-adapting too much. A fixed probability can be chosen. Furthermore, it can be seen as a technique to improve the *sparsity* property of the network. Dropout prevents overfitting and, due to the random nature of the process, it also leads to a combination of *thinned* neural network architectures [49], which are sampled and trained.

2.4 Convolutional autoencoders

2.4.1 Convolutional neural networks

Convolutional neural networks (CNNs) are a specialised kind of feedworward network [21] whose use is intensely deployed in the field of computer vision since the advent of *AlexNet* [35]. CNNs are very efficient in processing grid-like data, such as time series[3] and images. The name itself explains their main feature, i.e. they perform *convolution* instead of general matrix multiplication in their layers.

The use of convolution in deep learning can find a dual reason, namely an ease in the computation process of a convolutional layer compared to a dense one and its capacity to filter a part of the input signal and return a function of it. That is, convolutional networks exhibits sparse connectivity [21] by making the filters much smaller than the input. The kernel size reduction gives in turn detection of *local* features, i.e. that depend on subregions of the input [7]. This reduces memory requirements and improve computational efficiency [21].

CNNs are hierarchical models whose architecture consists of three basic building blocks, as shown in fig. 2.3: *convolution* layer, non-linearity and *pooling* layer [38]. The convolution operation incorporates in the neural network the concepts

[3]A time series can be considered as a 1D grid of samples, instead of pixels [21].

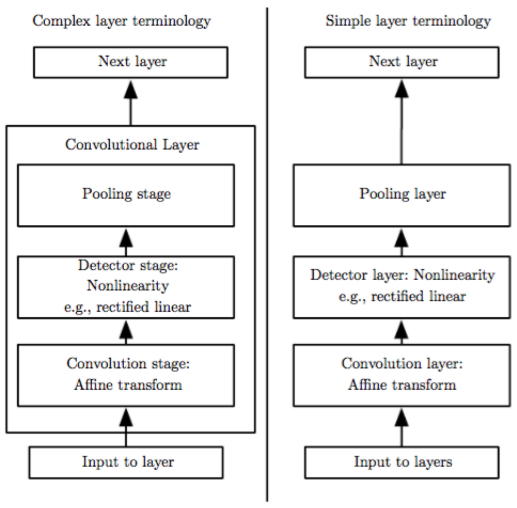
Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

Figure 2.3: The building blocks of a CNN layer. Two sets of terminology have been used, the former referring to a convolutional network as the sum of convolution, non-linearity and pooling, while the latter referring to a sum of simple layers. Image from [21].

of *local receptive fields* and *weight sharing*, while the pooling operation involves *subsampling* [7]. That is because all the units in a specific *feature map* share the same weight values specified in the kernel.

2.4.1.1 Convolution layer

The first building block performs convolution to produce a set of linear activations, which flow through the chosen non-linear activation function. The combination of this two blocks is referred to as the *detector* stage [21]. The result is an output representing a feature detected in different locations of the image. In other words, the unit will react if recognising a feature represented by a particular filter in a *local* portion of the input. This is performed for several filters, resulting in several feature maps, each represented by a specific set of weights.

The activation of the feature maps will not be changed if the input is shifted, for instance. This provides a *shift-invariant* property to the convolutional layer [7].

2.4.1.2 Pooling layer

The output of the convolutional layer forms the input of the pooling layer. Basically, it just replaces the output of the convolution layer with a statistic that sums nearby outputs returning only one value. The *summary* statistic can pick the largest value (*max-pooling*) or perform an average (*L2 pooling*) with the addition of a scaling *adaptive* weight [7]. The purpose of this process is to add *invariance* to small translations and distortions of the input [7, 21], making the feature map quite insensitive to small changes. In other words, since the hidden representation becomes translation-invariant, it is capable of an improved selectivity as the unit is fired when a strong correspondence between the feature and the input field over a region of interest is obtained. Each of these representation will be a distinct *input channel* [29] for the next convolutional layer.

2.4.1.3 Parameters

A discrete convolution has some additional parameters that may be set specifically depending on the addressed task.

Stride The *stride* determines the distance between the boundaries of the receptive fields of nearby units [29]. As an example, it refers to the number of pixels to skip when performing the successive convolution step in an image. Doing so, the receptive fields can be either consecutive or overlapped. A large stride leads to fewer units in the set of linear activation.

Receptive field The choice of the receptive field is also crucial for a meaningful feature map. In [35] a *11x11* with stride 4 receptive fields are deployed, while in [48] overlapped *3x3* receptive fields are used throughout the whole net (with stride 1). Moreover, [48] suggests the deployment of 1 x 1 convolutional layers in a *very* deep convolutional network to increase the non-linearity decision without affecting the receptive field, i.e. it is just a linear projection onto a space of same dimensionality with the addition of a non-linear function.

Zero padding The concept of *padding* comes from the fundamentals of audio signal processing. In [58] this technique is used to increase the frequency resolution for spectrum analysis. In short, a bunch of zeros are added to a windowed signal in order to apply a *Fast Fourier Transform* (FFT) whose size M is larger than the size of the window N. An analogous is deployed in the convolutional networks in order to control the kernel width and the size of the output independently [21]. If zero padding was not possible, the kernel would be forced to convolve only with subregions that entirely contain it. Conversely, if the input is zero padded the spatial extent of the network will not be *shrinked* at each stage.

The most common zero-padding techniques applied to CNNs allow a *valid* convolution [21], where no zero padding is used whatsoever, and a *same* convolution, where the zero padding is used, so that the spatial resolution is preserved after the convolution [48].

Lastly, the number of weights in a convolutional layer is smaller compared to a fully-connected one, e.g. the MLP. This is due to the use of such local receptive fields, which imposes the constraint that the weights be *tied*. In other words, the elements of a kernel are not used once, but they are applied at every position of the input instead. This returns in decreased storage requirements and more efficient convolutions [21].

2.4.2 Convolutional autoencoders

An autoencoder only made of fully-connected layers is not able to describe spatial properties of a multidimensional input, for instance an image. They also introduce redundancy since all the network spans the entire input [38], i.e. it learns *global* features.

Conversely, convolutional autoencoders (CAEs) make use of the convolution operator to share every set of weights at all position in the input, making the features be *local*.

2.4.2.1 Definition

The convolutional autoencoder architecture is similar to the one described in 2.3 with the main difference that the weights, i.e. the kernel matrix, are shared [38]. The latent representation of the k-*th* feature map for a input $\mathbf{x}$ is

$$h^k(\mathbf{x}) = \sigma(b^k + \mathbf{W}^k * \mathbf{x}) \tag{2.10}$$

where σ is the activation function, $*$ denotes the convolution operation and the feature map k has parameter set $\theta = \mathbf{W}, b$ with a unique bias for a whole map. The input $\mathbf{x}$ here is considered *mono-channel*, i.e. it is not a volume. Examples of mono-channel inputs are greyscale images and spectrograms.

As for a convolutional network, a CAE may have several kernels that are convolved with the input to specialise on different features on the whole input [38]. Therefore, the resulting hidden layer has the formula

$$\mathbf{H}(\mathbf{x}) = \sum_{k \in H} \sigma(b_k + \mathbf{W}_k * \mathbf{x}) \tag{2.11}$$

where $\mathbf{H}$ identifies the group of feature maps. The number of filter is referred as to a *hyper-parameter* to be tuned in the definition of the convolutional layer. Every convolution is wrapped by a non-linear activation function σ in a way such that during the training, the network learns to represent the input by combining non-linearities.

CAEs are fully convolutional networks, therefore they are capable of reconstructing the input by using convolution. The reconstruction is given by

$$\mathbf{r}(\mathbf{h}) = \sigma(\sum_{k \in H} b'_k + \mathbf{W'}_k * \mathbf{h}_k) \tag{2.12}$$

where a single bias b' spans the whole channel. The spatial resolution of the latent map is determined by applying a specific set of parameters, as explained in 2.4.1.3.

Since the number of parameters does not depend on the size of the input, CAEs are able to scale to high-dimensional inputs since each feature map always requires the same amount of weights and biases [38].

2.4.2.2 Loss function and gradient descent

Having the input and output dimensions equal to each other, it is possible to minimise a loss function that relates them, such as the *mean square error* (MSE):

$$L(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (x_i - r_i)^2 \tag{2.13}$$

where $\mathbf{x}$ represents the input vector and $\mathbf{r}$ the reconstruction. As for a standard network, the backpropagation algorithm is applied to compute the gradient with respect to the parameter set θ. The weights are then updated using, for instance, gradient descent (SGD) [7]

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \eta \nabla L(\theta) \tag{2.14}$$

where the parameter η is the *learning rate* and it is set to be a small positive value, i.e. $\eta > 0$.

In gradient descent, the weight parameters are iteratively updated in the direction of the negative gradient (hence the minus) of the loss function in order to minimise the objective. After an update, the gradient is re-evaluated with respect to the $\mathbf{w}$ vector and the process is repeated until a minimum is reached [7].

An *online* version of the gradient descent has proved useful when training neural networks on large datasets [7]. Unlike gradient descent, the gradient at each iteration is not computed on the whole dataset, but a single data point is processed at a time. Doing so, such a process is less memory expensive and may converge faster because of the efficient redundancy handling. This is called *stochastic* gradient descent.

Chapter 3

Experimental setup

This chapter provides the description of the structures deployed in this work, together with the configurations of their hyperparameters. I motivate the choice of the learning rate method and the model settings. Finally, I compare and discuss the results of the predictions for the different models.

3.1 Overview

Two models have been tested for two different tasks, namely the reconstruction of a given *log-magnitude* spectra and the transformation of an *input* single note given a *target* single note. A general overview for both tasks is given below. Further details about the technicalities of the dataset creation are provided later in the chapter.

3.1.1 The *reconstruction* task

As baseline for arguing that the autoencoder be able to learn and extract a latent representation of the input, a *reconstruction* task has been configured. Doing so, I created a custom dataset of roughly 4.000 *log-magnitude* spectra of duration 1 sec from the *MusicNet* [51] dataset of classical recordings[1].

However, after several training processes with architectures of different depth poorly accurate results had been achieved. Therefore, I decided to use spectra of limited complexity to cope with this task. Recording of single piano notes have been chosen, which are freely provided by the *Electronic Music Studios* of University of Iowa[2]. Such samples cover the whole range of the instrument, together with four different dynamic ranges (*pp*, *mf*, *f*, *ff*) in an overall number of 192 samples. I then

[1]http://homes.cs.washington.edu/ thickstn/musicnet.html

[2]http://theremin.music.uiowa.edu/MIS.html for further details regarding recording and instrument specifics.

deployed the yet made dataset with architectures of increased depth to minimise the reconstruction error.

3.1.2 The *transformation* task

The task described above is not of particular interest, given that in [19] the same task has been achieved with a similar structure. However, it can be seen as a way of finding the optimal set of *hyperparameters* for convolutional layers. These are then used as building blocks for more complex tasks with a similar input, such as the log-magnitude spectrum of an audio segment.

In this thesis, I attempted to actuate sound *transformations*. I chose to concentrate on *timbre* transformation, that is the perceived sound quality of a sound independent of pitch and loudness. Thus, timbre may be referred as a characteristic for distinguishing different types of sound production, and different instruments.

It is therefore possible to set an input sound for the architecture and, as output, a sound *other* than the input. The latter is called *target* sound, which is the timbre the architecture aims to reproduce. For the sake of simplicity, two small sets of two instrument single notes, a flute and a *Bb* clarinet have been chosen, each of them containing the same number of notes (37 in total), ranging the same interval (from B3 to G6) and with the same dynamic range, that is *ff*. Both sets have been arranged in a way such that for a specific note of the input sound corresponds the same note of the *target* sound.

Note on the terms The use of the terms *parameters* and *hyperparameters* follows the terminology given in [4]. The term *parameters* refers to the weights and their update with respect to the analytic gradient, thus the values within the architecture. The term *hyperparameters* refers both to the *bells and whistles* [4] involved in the learning algorithms for deep learning and to the different option that a layer exhibits, such as *learning rate methods*, *initialisation techniques*, *constraints* and *batch size*.

3.2 Models

All the networks deployed in the experimental setup are *convolutional* autoencoder (see chapter 2 for further theoretical details). In order to match the architecture with the required task, different configurations have been realised, which differ mainly in the number of *convolutional layers* used both for the *encoding* and the *decoding* part, and in their displacement (e.g. parallel branches, use of pooling layers etc.).

I built the architecture using the *Keras* library [11], which is a high-level library for Deep Learning that runs on top of Google Brain's *TensorFlow* [1]. The key concept in both interfaces is *modularity*: a model is seen as a series of modules, that is the *nodes*. Each node has its own input and output (ranging from zero to multiple) and represents the instance for an operation to be computed. The sum of all nodes gives a *graph*, in which values in shape of *tensors* flow along. The graph represents the whole dataflow computation which takes place inside the architecture. The computation makes use of CPU or external GPU accelerators when possible. Finally, the use of such interfaces in open-source programming language, such as Python or C++, allows for accessing more and more increasing open-source community

3.2.1 Design choices

Such models share some common overall design choices for sizing the architecture and choosing the hyperparameters for the layers and the learning process.

3.2.1.1 Initialisation

As a general rule of thumbs, the weights should be initialised as small non-zero values to avoid the undesirable case where the network does not learn. A crucial point is that the network should not *explode* during the backpropagation process, that is the neuron being saturated because of an initial too large weight. However, the weights should not be identically zero and, in a more general sense, they should be *random* and *unique*. That is because equal weights will be updated by the same gradient, thus there is no source of asymmetry [4].

The concept of *symmetry breaking*, together with the one of *variance calibration*, are important for a successful learning process. The recommended heuristic is to initialise the weights as a random sampling from a *standard* Gaussian distribution and scale them according to the square root of the *fan-in*, i.e. the number of inputs to a hidden unit[3], as described in the work of Glorot et al [20]. This, particularly, should help the gradient to flow at a faster rate.

I used the initialisation derived from the work of He et al. [25], which is specifically suggested for ReLU neurons in a feedforward network. In short, the variance of the network should be $2.0/n$, where n be the number of input units in the weight tensor.

[3]In a CNN, the number of fan-in has to take into account the number of features maps (the *filters*) together with the receptive field. The formula to compute this value is:
`fan_in = feature_maps * receptive_field_height * receptive_field_width`.
(source: http://deeplearning.net/tutorial/lenet.html, last visited May, 26th.)

3.2.1.2 Constraints

As described in 2.3.2, I implemented a sparsity constraint as an independent Keras regulariser class[4]. The parameters are set according [42], that is Ω be 0.01 and β be 0.5. Equation 2.8 in 2.3.2 describes the constraint. The use of a sparsity-induced penalty is suggested in [4] and is applied in deep belief network for specific audio tasks in [36].

Moreover, the introduction of *dropout* before the convolution layer complements the other methods. For further details see 2.3.3 In the experiments, I set the *rate*, e.g. the percentage of the neurons being excluded in the sampling neural network, being smaller than the value suggested in [35, 49] ($0.1 - 0.2$ in range $[0, 1]$ instead of 0.5). The experiment showed that a larger amount of dropout rate let the structures quite inefficient when training, that is the network does not converge and the loss function is far from being minimised.

The use of dropout seems particularly necessary because of the limited training data. As explained in [49], such a technique may control to some extent the tendency of a convolutional autoencoder to easily learn the identity function especially if small kernels are set.

3.2.1.3 Regularisation

Regularisation is a way of penalising some measure of complexity of the model. Together with the constraints, regularisation techniques may prevent *overfitting*. That is, we force the model to assume a particular set of parameters using added penalties. The concept of regularisation should be intended as a way to take over every single weight, while the sparsity is conversely applied to the whole network and concerns the values *after* the non-linearities.

I therefore used the **L2** regularisation, which is perhaps the most common in the neural networks. It can be implemented by adding one more penalty factor to the objective function, which is the *squared magnitude* of all the parameters. In other words, a term $\frac{1}{2}\lambda w^2$ is added for the set of weights, where λ be the regularisation *strength* (and, in practice, the parameter to change when implementing such a technique in a library, such as TensorFlow or Keras). The term $\frac{1}{2}$ serves as a normalisation so that the gradient of this term with respect to the parameter w is simply λw instead of $2\lambda w$. The L2 regularisation can be interpreted as a way of favouring smooth and diffuse weight vectors penalising non-homogeneous values. From this, the network will tend to use all of its inputs.

The superiority of L2 over L1 depends whether the task is to extract explicit features selection, since L1 makes the weight vector become sparse during the

[4]Keras has its own regulariser as well as the Kullbach-Leibler divergence function, but does not bring itself the concept of sparsity.

optimisation. Later in 3.4, L2 has proved to help in minimising the reconstruction error function better than L1 for the reconstruction task.

3.2.1.4 Receptive field

In 2.4.1.3 I addressed the importance in the choice of the receptive field. I chose in this project small spatial filter mostly of size *3x3* and *5x5*. Intuitively, small filters allows for extracting more powerful features of the input with a relatively small parameter cost. If stacking several convolutional layer made of tiny kernels, the latent representation will also take advantage of the repetition of non-linearities. A drawback in such a design choice can be the demand for more constraints, as pointed out before, and a larger amount of memory to compute the backpropagation.

Such small filters, in conjunction with unit stride and zero padding (see 2.4.1.3 for details), make the convolutional layer retain the size of the input so that only the pooling layers will be in charge of downsampling the spatial dimensions. Moreover, the use of zero padding permits that the information at the edges of the input be kept for longer time.

Finally, as motivated later in 3.2.2, I used *dilated* convolutions with several dilation factors in the effort to improve and expand the receptive field maintaining small kernels.

3.2.1.5 Activation function

As a general suggestion, the activation function should be chosen depending on the range of the target values. In other words, if an activation function has a range of [0,1], such as the sigmoid, all the target values should lie within that range.

As deployed in [19], I used *Rectified Linear Units* (ReLUs) for each layer in each structure[5]. The choice reflects several advantages of using such a non-linearity and *"is essential for state-of-the-art neural network"* [25].

First, the results of Krizhevsky et al. in the *ImageNet* seminal paper [35] show that ReLus accelerate the convergence of stochastic gradient descent. Second, ReLU takes a single value of the convolution and performs the function $f(x) = max(0, x)$. That means, very inexpensive operations are involved if compared with the exponential of the sigmoid and it can be implemented as thresholding a matrix of activation values to zero.

Moreover, as the computation of the input log-magnitude spectra forces all the values be non-negative because of the $log(x + 1)$, I expect the target values be non-negative. Therefore, I did not deployed the *Leaky ReLU* as [19], which allows for small negative values above a fixed value α. In this case, the choice of ReLU is

[5]Although they have used an improved version as explained below.

also convenient since it does not require input normalisation in the avoidance of saturation.

On the other hand, ReLU shows some fragility during the training, especially if the learning rate is high such that the weight update will not permit for a specific unit to be fired on another datapoint. LeakyReLU and the *parametric* ReLU (PReLU) introduced in [25] attempts to fix the problem allowing for a slope in the negative region, which can be a parameter of the neuron.

Finally, the choice of rectifying non-linearities, such as *ReLU* has a strong impact on the sparsity obtained and has proved to be successful [4].

3.2.1.6 Loss function

As explained in 2.3.2, the objective function is made by a regularisation loss part and a data loss part. In 2.4.2.2, I introduced the *mean squared error* as a measure to be minimised during the training process. In short, the *data loss* measures how much the reconstruction differs from the given network input.

When training using mini-batches, the loss is evaluated on individual batches during the forward pass, thus the squared error for an autoencoder prediction for a batch of examples is the squared average of the error for each example, which is computed across all dimensions. As a general advice, the loss function is increasing as the regularisation strength are increased.

Moreover, a relationship can be found between the loss function behaviour and the batch size. A common heuristic is to avoid too *noisy* functions increasing the batch size (the lower the batch size, the more noise can be noticed).

Such a loss function is suggested in [54] and in [21] as a way of penalising the output for being dissimilar from the input.

3.2.1.7 Learning rate methods

The *learning rate* undoubtedly plays a crucial point in a successful training. The learning rate, often referred as α, is hyperparameter of the optimisation technique for parameters update. In short, the backpropagation algorithm computes the analytic gradient and then such gradients are used to perform parameter updates given a specific optimisation technique, such as stochastic gradient descent (SGD) or Adam.

Optimisation techniques As described in 2.4.2.2, SGD is the simplest form of update. I initially used a modified form of the *vanilla* SGD, which includes the *Nesterov Momentum*[6]. I then made use of **Adam** [33], which has the great advantage to be *adaptive*. Both are algorithms for first-order gradient-based optimisation, with

[6]The theory behind the Nesterov accelerated gradient (NAG) is beyond the scope of the writing. Further details can be found in Sutskever's PhD dissertation [50].

the former manipulating the learning rate globally and for all the parameters and the latter adapting the learning rate in a *per-parameter* way. The utilised value in this work are the recommended ones in the [33] ($\epsilon = 10^{-8}, \beta_1 = 0.9, \beta_2 = 0.999, \alpha = 10^{-3}$).

Rate annealing, or rate *decay* The learning rate can be interpreted as a way of controlling the amount of *energy* in the system given by the gradients. Too much energy leads in the weight vectors *bouncing* unable to settle down into deeper part in the loss function. Conversely, if the learning rate is too *low*, so will be the learning process and the system will not reach further minima.

A common heuristic is to *anneal* the learning rate over time, i.e. to make it decay at the right pace. Different type of implementing the learning rate decay are possible, such as *step* decay and *exponential* decay. The SGD implements the former, while Adam the latter.

3.2.2 The architectures

I deployed two main architectures in this project. As said before, both of them make use of *convolutional autoencoders* as building blocks, with the main differences between such structures lying in the *depth* and in the use of advanced techniques, such as *dilation* [56] and *residual* framework [24]. The way these techniques have been implemented slightly differ from the original proposals. For instance, in [24] He suggests the use of a *shortcut* connection approximately every two layer of weights. As shown later in figures, four different basic structures have been tested in training processes, which present different combinations of residual connections and dilation within the convolutional layers. Here is a short description:

- a *shallow* convolutional autoencoder is used for the reconstruction task. That is, only a hidden convolutional layer takes place between the input and the output layer without the hidden layer being reshaped by means of max-pooling operation. In other words, all the layers share the same dimensions. As explained later in 3.3, increased levels of depth have been implemented and tested, leading to a *stacked convolutional autoencoder*, in the attempt to minimise the reconstruction error. No residual connections or use of dilation is involved in such a structure, which is shown in figure 3.1.

- a shallow convolutional autoencoder with *residual* connections in the encoding part is used in the very first part of the transformation task to assess whether it may help in the task deployment. Results show that a significant improvement in minimising the reconstruction error is achieved when implementing *shortcut* connections in the *encoding* part .Figure 3.3 shows the structure.

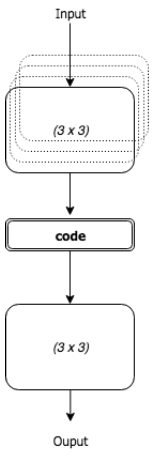

Figure 3.1: The shallow convolutional autoencoder structure.

- a shallow convolutional autoencoder that makes use of *dilation* is also used in the transformation task. Dilation has proved to help in accounting for multi-scale information [56] and for this specific case it is used to get an insight of the time-frequency representations given different *timescale* and *frequency* range. The use of dilation is deployed both in the *encoding* and in the *decoding* part. The basic structure is depicted in figure 3.2.
- finally, a convolutional autoencoder that makes use of *residual* connections and *dilation* factors is used for the transformation task. The residual connection takes place in the encoding part, while the dilation is involved in both parts. Moreover, a *max-pooling* layer is used to add invariance to small translations (see 2.4.1.2 for details) and the encoding part presents two convolutional autoencoder to benefit of a more powerful reconstruction. The structure is shown in figure 3.4

The use of *residual* connections The concept of *deep residual learning* [24] has emerged in the last couple of years leading to a new *de-facto* state-of-art for image classification. In the work of He et al., the use of residual learning is motivated by the difficulty of training deeper and deeper neural network. In short, deep models lead to higher training error [24].

To apply the concept of residual network, *shortcut connections* need to be introduced in the structure. The shortcut connections simply perform identity mapping (i.e. a *linear* activation function), and their outputs are added to the outputs of the stacked layers after the non-linearity [24]. Identity shortcut connections add neither extra parameter nor computational complexity since they can easily be seen as convolution with unity filters (*1x1*) initialised to 1. After the adding operation,

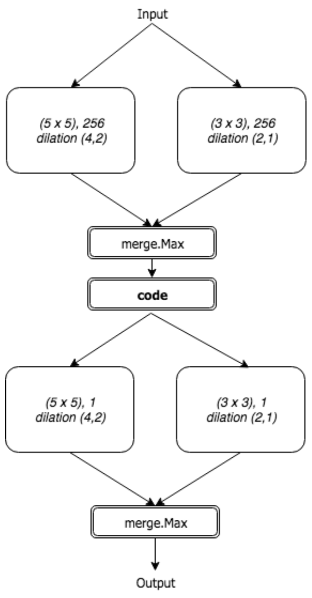

Figure 3.2: The shallow convolutional autoencoder with *dilation* factors.

a new non-linearity (in this case ReLU) is triggered, as shown in figure 3.3.

The use of *dilation* Dilated convolutions support exponential expansion of the receptive field maintaining the same resolution [56]. The use of dilation makes sense in the context of creating a *multi-scale* representation of the same input without the need for analysing rescaled version.

Moreover, musical sounds can be described as made of two distinct parts, the former characterised by the main modes of vibrations (e.g. harmonics and overtones) and the latter that contains non-sinusoidal energy components and perturbations due to the excitation process (often referred as *residual noise*) [47]. It also has to be considered that the *temporal envelope* of the sound largely contributes in the timbre recognition [30]. That means, the architecture should attempt to extract as much of temporal information as possible for a better timbre transformation.

3.3 Experiments

3.3.1 Datasets

As said earlier in the chapter, the experiments make use of two different sets of data according to the task. To accomplish the reconstruction task, a piano sets of 192 notes has been used, while for the transformation task two sets of 37 notes each of flute and clarinet have been deployed.

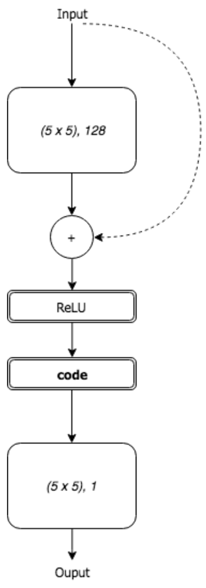

Figure 3.3: The shallow convolutional autoencoder with *residual* connections for the encoding part.

The choice of such sets reflects the aim of finding some baseline for the experimental setup. A piano note has a clear transient followed by a release part and, given its percussive nature, it stabilises its harmonic content in a quite fast pace[7].

On the other hand, I chose flute and clarinet because of some common characteristics. First, they belong to the same family of instruments, that is the *woodwinds*. Second, they share a similar range (D4-C7 for the flute and E3-C7 for the clarinet[8] Third, they share similar way of producing sound. Nevertheless, the stream of air is directly focused inside the flute cylindrical tube, while in the clarinet air is focused into a mouthpiece which then causes the reed to vibrate.

From audio file to spectra All the notes come in *.aif* audio file format. The passage from such a format to *log-magnitude* spectra has been deployed creating a script in Python that makes use of the *librosa* library [40] for the audio processing computations.

First, the *Short-Time Fourier Transform* (STFT) is computed on the audio file *downsampled* to 22kHz. Then, the log-magnitude spectra of the complex spectrum is computed using the formula $log(|S|+1)$, where $|S|$ be the module of the spectrum. To get the best results, I used a FFT size of 1024 with 50% of overlap.

The choice of adding 1 is for numerical optimisation. As it turns out for single

[7]However, the very beginning of the tone may result quite problematic if not so accurate phase information is used.

[8]of which the lower register until C4, that is the *chalumeau* register, has been discarded because of its particular sound qualities.

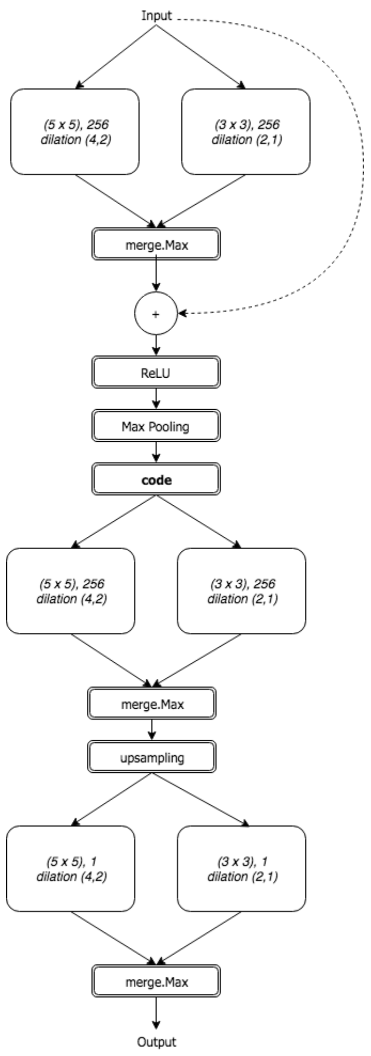

Figure 3.4: The convolutional autoencoder with *dilation* factors and *residual* connection in the encoding part. This structure has proved to be the most successful in the transformation part. Experiments involve the use of increased depth and different number of filters. The two parallel dilation blocks allows different receptive fields over the same input.

note spectra to have most of their values close to zero, when taking the logarithm of a very small value some *underflow* issue may be experienced[9]. Using this formula a non-negative real-valued is guaranteed, which is optimal when using an activation function, such as ReLU. Moreover, it is also useful in order to avoid any `nan` value that undermines the correct computation of the stochastic gradient descent.

Finally, the spectra are stacked in a tensor of [training_data x freq_bins x timesteps] dimension, which is successively used during the training process.

3.3.2 Infrastructure

Such deep structures demands for machines with high computational power, way beyond the normal capabilities of personal laptops. Deep learning involves huge amount of matrix multiplications, convolutions and other operations which can be massively parallelised and thus sped up on GPUs. Indeed, a GPU architecture can fetch much more memory at once being bandwidth optimised.

Therefore, to cope with the need of prolonged computations, I accessed the DCC (DTU Computing Centre)[10], which provides several configurations (i.e. *nodes*) of stacked GPUs.

3.3.3 Training

Figure 3.5 shows the schema for the experiments. As the *transformation* task is the one with more emphasis on, several training with different models and hyperparameters have been devoted to it. As mentioned above, the *reconstruction task* is meant to provide the optimal set of hyperparameters[11].

In the *reconstruction* task, 4 convolutional autoencoders of increased depth both for the *encoding* and the *decoding* part are fed with the piano set. All these models use a learning rate of 1e-3 with decay rate of 1e-6 and are trained with Adam optimiser. Particularly, the shallow model has been trained with different *batch size* values (4,8) and with different *kernel regularisation* methods (L1, L2). The number of channels for each convolutional layer is 256, each of size (3,3). The models with increased depth have been trained in a *layer-wise* fashion [54, 5], i.e. only one layer is trained at a time. After the layer training, a new layer is stacked on top of it. The weights for the trained layer are kept, while the decoding part is newly trained. The *transformation* task moves from the shallow model depicted in 3.1 (it performed better, as described later in 3.4) and deploys 3 new models that make use of dilated convolution and/or residual connections.

[9]The term *underflow* refers to the problem of performing an operation that leads to a number that is smaller than the smallest magnitude non-zero number.

[10]http://www.hpc.dtu.dk/

[11]Herein, the terms *structure*, *network* and *models* are interchangeable, as well as the terms *filters*, *channels* and *kernels*.

The *dilated* convolutional autoencoder is a shallow network which has 256 channels in the hidden layer. Each filter can be either of size (3,3) with dilation (2,1), or size (5,5) with dilation (4,2). Moreover, the model has been trained using batch sizes of 2 and 4. The *residual* convolutional autoencoder is also a shallow network, in which the *shorcut* connection adds the first convolutional layer to the identity mapping of the input, leading to the *code* (see 3.3 for details). This model has 256 channels of size (3,3). The two aforementioned models use a learning rate of 1e-3 with decay rate of 1e-6 and are trained with sthochastic gradient descent with Adam optimiser.

Finally, the last model combines both *dilation* and *residual* connections. It has been trained with different batch size values (2, 4), channels per convolutional layer (64, 128, 256 and 512). The size of the channels, together with their dilation factor are the same as the previous models. All the models use learning rates ranging from 1e-4 to 1e-6 with decay rates ranging from 1e-6 to 1e-8.

All the models in the experiments have been trained using approximately 85% of the amount of datapoints in the dataset as training part, and 15% as a validation part. They also have been trained for 100 epochs each.

In order to provide an audio outcome of both the reconstruction and the transformation, the outcomes of such models are used to performs *inverse* (*i*STFT), together with the reconstructed phase information using a well established iterative technique [22]. In short, the Griffin-Lim algorithm iterates through the STST-*i*STFT in order to provide more and more accurate phase information given a magnitude spectrum. The phase estimates vector is initially set to random values in range $[-\pi, \pi]$ and is updated with newer estimates of the angle of the complex part at each new STFT. I set a number of 100 iterations for the algorithm, with the STFT and the *i*STFT using the parameters given in 3.3.1.

3.4 Results and discussion

I provide both graphical and supplemental audio examples since magnitude spectrograms are sometimes hard to evaluate and, although they may appear quite similar to the eye, they can correspond to completely different audio excerpt given the many aspect of an audio file embedded in such a representation. Moreover, the audio examples[12] give a good insight on the effectiveness of the phase reconstruction. A list of the images for the reconstructed spectra can be found in Appendix A. Table 3.1 shows the results.

[12]The audio reconstruction, the plots and the spectra can be found in the folder *results* in my Github repository for this master thesis: `https://github.com/inspiralpatterns/master/tree/master/SMC%2010`

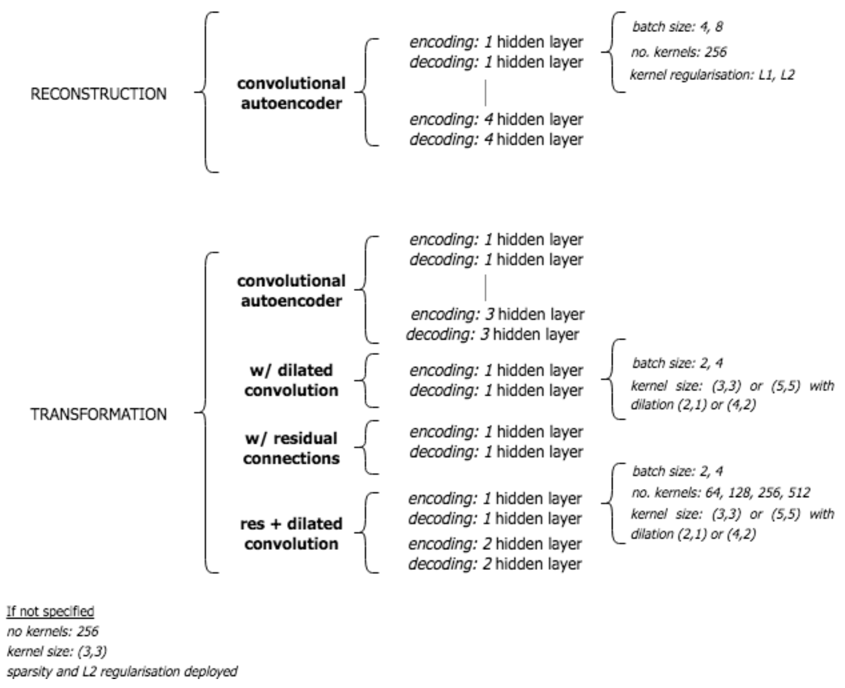

Figure 3.5: Schema of the experiments for both tasks.

Model	Loss	Reconstruction error
	Reconstruction	
shallow (batch=4)	5.232e-3	3.215e-4
shallow (batch=8)	4.47e-3	2.043e-4
2 layer deep	8.756e-3	5.237e-4
3 layer deep	1.339e-2	1.418e-3
shallow+l1	7.759e-1	1.424e-3
	Transformation	
base	2.028e-1	4.697e-2
2 layer deep	1.062e-1	4.876e-2
3 layer deep	1.379e-1	5.292e-2
dilated	2.199e-1	4.698e-2
w/residual	2.022	5.210e-2
dil+res	1.355	*3.417e-2*
2 layer deep	3.505	*3.825e-2*

Table 3.1: Results for the tasks. It can be noticed how the use of dilation together with residual connection helps improving the model performance and stability. As a matter of fact, both models involving such techniques show the best results. However, it can be noticed how the validation loss values are relatively high compared to the baseline model. This may lead to a longer training process for *fine-tuning* the parameters. It is also worth noticing that both the validation loss and the reconstruction error values are larger for the *transformation* task, representing the increased level of complexity. Each value represents the mean over the last 10 epochs.

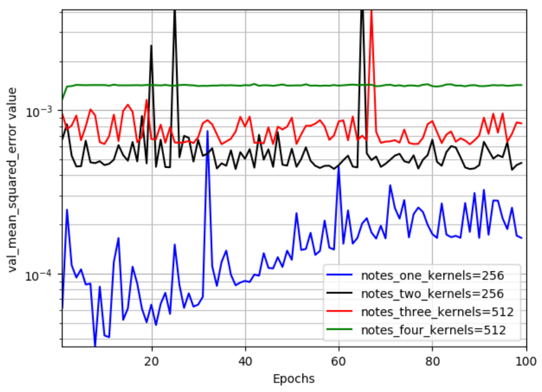

Figure 3.6: Comparison of the reconstruction error for the models in the *reconstruction* task. The figure shows how the error for the validation task increases as the depth of the model increases. The number 256 and 512 refer to the number of channel in the level of depth, e.g. the model 3 layer deep has 256 channels in the first two layers, and 512 in the third. The y axis value are in log scale.

Figure 3.8 and 3.10 show the best results for each task. From the experiments, the single hidden layer convolutional autoencoder (figure 3.1) seems performing a better reconstruction (see figure 3.6 for a comparison between the models), while the convolutional autoencoder with dilation and residual connections depicted in figure 3.4 is capable of a fair transformation between the two instrument timbres.

3.4.1 The reconstruction task

At a first glance, figure 3.7 shows the learning rate of 1e-3 be optimal for the single hidden layer model (figure 3.1) since the learning process exhibits an exponential form shape[13]. However, the validation loss function shows some undesirable *peaky* behaviour, which may call for an increased batch size. Furthermore, I trained the model using a batch size of size 8, i.e. double as much the previous training. Figure 3.8 shows little improvement, as the validation loss function exhibits similar peaks. It is also worth noticing how the learning process requires more iterations to lower and stabilise as the batch size is increasing.

Being the gap between the training and validation error relatively small, the model seems having low overfitting[14]. However, in both training processes the reconstruction error for the training and the validation set moves upwards over the last iterations. This may be explained by two possible scenarios.

In the former, the few amount of data is responsible for it. Given the dimen-

[13]All plots have y axis values in the *log* scale

[14]In the evaluation of the results, the course notes for Stanford CS231 class taught by Andrej Karpathy have proved really useful. The link to the note referring to this specific argument is `http://cs231n.github.io/neural-networks-3/loss`, page visited on May, 23rd

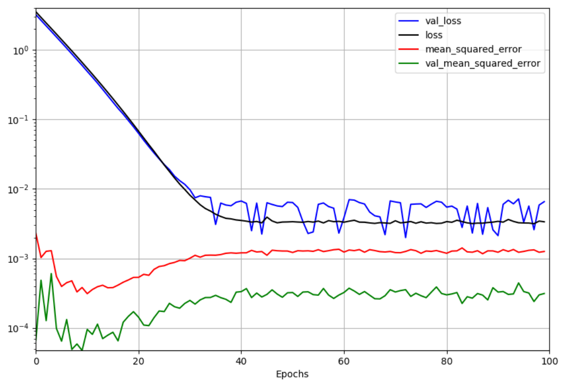

Figure 3.7: Measures for the shallow convolutional autoencoder for the *reconstruction* task with batch size of 4. The y axis value are in log scale.

sions of the piano sets (180 notes for training and 12 for validation), this option may justify such a behaviour. Moreover, the differences in the dimensions (*time* vs *frequency*[15]) of a representation such as the spectrogram, make more difficult the use of augmentation techniques, such as *rotation*, *flipping* or *shearing*, be applied[16].

In the latter, the reduced capacity of the model leads to the reconstruction error being not able to lower at consecutive iterations. This accounts for a model improvement, which is built on the shallow convolutional autoencoder. As described in 3.3.3, up to 4 convolutional layers[17] have been stacked in a way such that a new training process used the previous weights, so that only the new layers be trained by scratch and the model be effectively trained. The idea is to mimic the structure described in [19], which yields to fair results in the reconstruction of a single tone.

As shown in figure 3.6, at each new stacked convolutional layer the values for the reconstruction error slightly increase. The model is 4 layers deep with 2 layers having 256 channels and the deepest layers having 512 channel. Similar behaviour is presented when increasing the depth in the transformation task. I tackle this issue later in 3.4.3.

Finally, I trained the shallow model using L1 regularisation. Figure 3.9 shows that such a regularisation does not help in minimising the reconstruction error function. Indeed, the function seems not decreasing over the iterations. If analysing the loss function for the model with L1 regularisation, a steep slope at the very be-

[15]Observation inspired from Sander Dieleman's blog.
Here the blog entry: `http://benanne.github.io/2014/08/05/spotify-cnns.html`

[16]Here for an example of improved performance of deep learning using *brightness augmentation*: `goo.gl/UTmhzZ` (shortened link), page visited on May, 28th.

[17]The term *layer* is intended as the whole process of convolution + max-pooling, that is technically split in two consecutive layers.

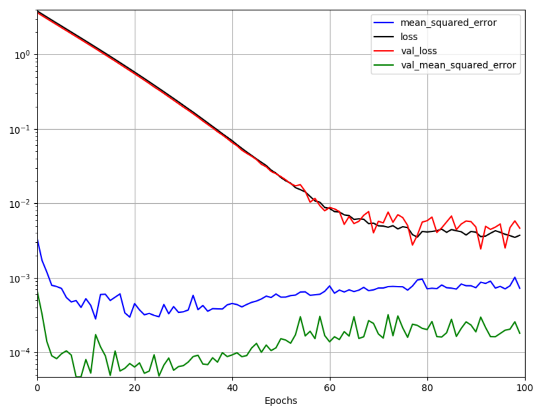

Figure 3.8: Measures for the shallow convolutional autoencoder for the *reconstruction* task using an increased batch size of 8. The y axis value are in log scale.
It is interesting to notice how the values for the reconstruction errors at the end of the iterations slightly move upwards.

ginning of the training can be noticed, which is followed by a relatively stable curve. As pointed out before, both the loss and the reconstruction functions might be a sign of a too high learning rate if L1 regularisation is used.

Another interpretation lies in the choice of L1 with other constraints, such as dropout and sparsity. If working with penalties that make the hidden units partly inactive during the learning process, the network will have only fewer non-zero activation values. L2 regularisation works in a way such that large values are punished because of the *square* operation. Conversely, L1 allows for sparse values and this may result in an even more *sparse* layer. Moreover, since L1 does not really penalise large values [18], they may be allowed during the parameter update with the consequence of such value not be used anymore during the training, hence the decreased performance.

3.4.2 The transformation task

Figure 3.10 shows the validation reconstruction error for the models described in 3.2.2. It can be noticed how the model that makes use both of residual connection and dilation (see 3.4 achieved the best result in minimising the error, obtaining roughly a 30% relative improvement on the task.

[18]If considering the vectors $a = (0.5, 0.5)$ and $b = (-1, 0)$, the L1 norm is $\| a_1 \| = |0.5| + |0.5| = 1$ and $\| b_1 \| = |-1| + |0| = 1$, while the L2 norm is $\| a_2 \| = \sqrt{0.5^2 + 0.5^2} = 1/\sqrt{2}$ and $\| b_2 \| = \sqrt{(-1)^2 + 0^2} = 1$. The two vectors are equivalent for the L1 norm, but different with respect to the L2 norm. Therefore, the L2 norm will penalise more vector b.
(source: `https://goo.gl/gMPSCh` (shortened url), page visited May,27th.

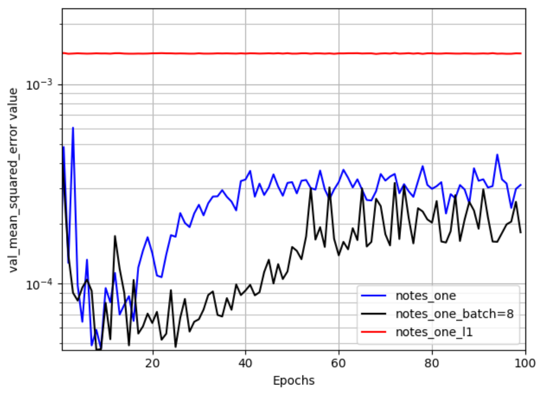

Figure 3.9: Validation reconstruction error for the models with L1 and L2 regularisation. Both the model with L2 regularisation performs a fair job in minimising the error, while the L1 regularisation is not capable of lower it. The y axis value are in log scale.

Two observations are of much importance. First of all, the learning rate for this model (and for the baseline models with either residual connection or dilation) had to be lowered to 1e-4. A motivation for this can be found in figure 3.11. When using the optimal learning rate of the reconstruction task, the validation reconstruction error exhibits a quite large jump around 30 epochs and is not able to converge then. That may be a piece of evidence that such a learning rate be too high and it does not lead to a good optimisation. Conversely, the models with lowered learning rate seem capable of constantly decreasing the reconstruction error, and in a smoother way.

Secondly, the reconstruction error function for the model with the best result still exhibits the same *peaky* behaviour encountered in the reconstruction technique. However, increasing the batch size has not been possible for this model, since every try encountered memory allocation problem with the process being aborted by the infrastructure. This can be due mainly to the increased amount of memory required when using dilation and residual. To give an example, each convolutional layer in the model in figure 3.4 needs 3 parallel paths, of which two for the dilation plus one to make the shortcut connection possible. That is, roughly sixfold need for memory if compared with the baseline.

Moreover, it is also possible noticing how the chosen learning rate seems optimal for the task. However, figure 3.10 shows that the model could benefit from a scheduler for the learning rate, such as the one described in [19], sec. 2.3. For the model in figure 3.3, I lowered the learning rate from 1e-4 to 1e-5 and the decay rate from 1e-6 to 1e-9. This adjustment has proved to be beneficial for the reconstruction error function, as it constantly decreased over all the iterations.

Finally, it is also worth noticing how the use of residual connection mitigates

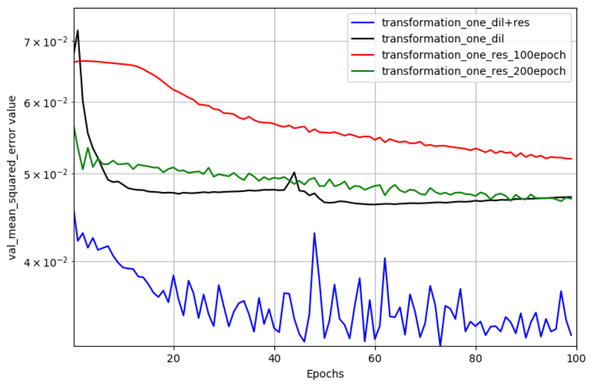

Figure 3.10: Comparison validation reconstruction error for different tested models in the *transformation* task. The y axis value are in log scale.

the effects of the small amount of data used in the training processes shown in figure 3.8, whereas the only use of dilation does not help. However, figure 3.10 shows how the dilation may help in speeding up the optimisation since it takes almost double as much the time for the model with only residual connections to get to the same reconstruction error for the validation set. Moreover, it is worth using dilation of different values in conjunction with 0-dilated filters, provided by the shortcut connection, to better merge spatial information. From this, the urge to include both techniques to increase the model capacity.

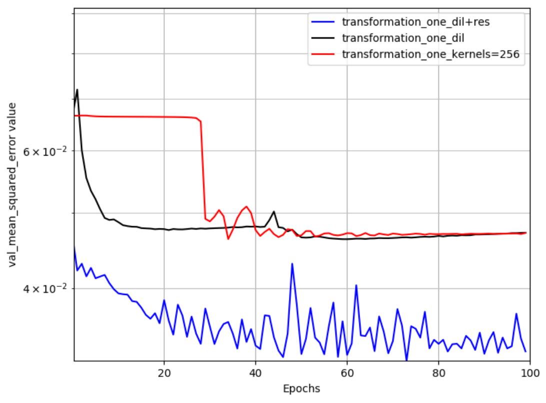

Figure 3.11: Comparison between three different models for the *transformation* task. The baseline autoencoder with 256 channels was trained with learning rate of 1e-3 while the other with learning rate lowered to 1e-4. As consequence, the latter learning rate has proved to be optimal for this task, avoiding the *bouncing* behaviour of the validation reconstruction error for the baseline model. The y axis value are in log scale.

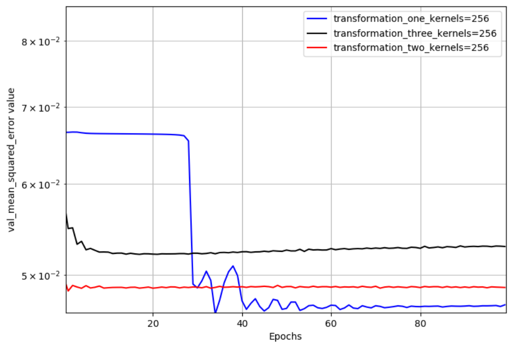

Figure 3.12: Comparison of validation reconstruction error for the baseline with increased depth in the *transformation* task. The y axis value are in log scale.

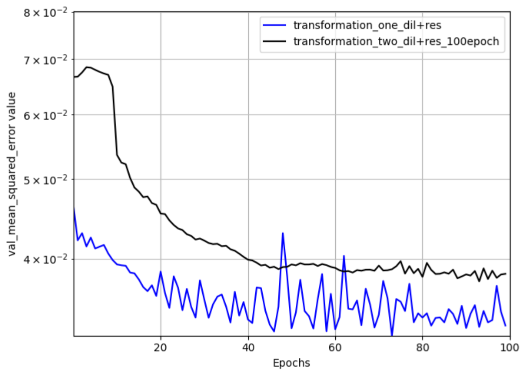

Figure 3.13: Comparison of validation reconstruction error for the full model with increased depth in the *transformation* task. Here *full* refers to the use of both residual and dilation in the network. However, the deeper the model the larger the reconstruction error. The y axis value are in log scale.

3.4.3 Further discussion on results

Figures 3.6, 3.12 and 3.13 show a common behaviour: all the models share a decreasing in the validation reconstruction error function as the depth of the structure is increasing. A corresponding decreasing in the model performance can be noticed both if plotting the reconstructed spectrograms (Appendix A contains the plot of reconstructed spectra for each model deployed in each task) and if listening to the audio excerpts.

As a first remark, it can be noticed how an increased depth leads to a reconstruction error function which does not move upwards. This may be used as a proof that the shallow models may be improved in its capacity. However, none of them for the *deep* models achieves values as good as the validation function ones for the shallow model. If inspecting the spectrogram plots, it is worth pointing out how depth introduces a *blurred* character, thus reconstructed audio from deeper models tends to sound more as a series of burst of coloured noise, rather then single notes. A clear tendency in this scenario is that low fundamental frequency notes (e.g. A2) gets more blurred, and more distorted. This is probably due to the displacement of the harmonics, which are closer to each other than in a high-frequency note. As consequence, the whole phase reconstruction process is biased, leading to poor reconstructions.

As a second remark, the values in the reconstructed spectrograms are undoubtedly squashed close to zero. As an example, the initial matrix of piano notes in range [0, 2.5] is crushed in the output matrix in range [0., 0.025]. Moreover, both the audio excerpt and the spectrograms are dissimilar at all, resulting in a severe distortion of the original information.

A motivation may reside in the choice of non-linearity together with the L2 regularisation. If we take a look closer, the L2 may force the value being smaller and smaller, while the ReLU acts as a linear mapping in the positive domain. Since only non-negative values lie in the input, this may lead to this behaviour. Nevertheless, L1 regularisation does not help in allowing for larger parameters. There could be several ways of coping with such an issue. As first attempt, the learning rate could be lowered at each new layer insertion. Unfortunately, if running a training process for 50 epochs with learning rate of respectively 1e-4, 1e-5 and 1e-6, the reconstruction error for both the training and the validation set does not decrease. That means, the learning rate is too small and the network does not learn. In addition, for the last value the loss function is not able to converge.

Another way could be to apply preprocessing to the dataset. For instance, zero-centering the data and allowing for the layer output to take small negative value using an activation function, such as *leaky* ReLU, may help in preserving the original range (Engel et al. applied successfully *leaky* ReLU to their baseline autoencoder in [19]). Another way could be to reduce the amount of sparsity, although it can lead to the network to learn the identity function and simply *copy* the input.

If inspecting in the model deployed for the transformation task, the influence of this behaviour on the validation reconstruction error function seems mitigated. Two possible reason may explain such a reduced influence. First of all, the use of residual connections allows for a stronger *preservation* of the input range (the input is summed to the post non-linearity layer outputs). Secondly, merge operations are applied after parallel dilated convolution, which retain the maximum values over all the feature maps. Their combination may prevent the loss of range in the output. Nonetheless, if listening the recordings a *noisy* component is added making the notes with lower fundamental frequency a bit undistinguished in their pitch and timbre contour.

Chapter 4

Conclusion

In this thesis, I aimed to create the meaningful basis for the generation of audio excerpt given an audio input signal. In the attempt to do so, I deployed a series of convolutional autoencoders that take log-magnitude spectra as input. Such neural networks have been tested on two different tasks: *reconstruction* and *transformation*. With the former, I tuned the parameter of the building blocks, namely the *convolutional layers*. In the latter, I applied such parameter and I refined the structure, introducing recent techniques, such as residual learning and dilation, to cope with the increased complexity of the task.

I then used the resulting output spectra to get an estimate of the original phase information by means of the Griffin-Lim algorithm, and eventually generate an audio excerpt entirely build on *non-original* information. Results show a fair job of the convolutional autoencoder for both tasks, with emphasis on the latter where the introduced techniques led to a significant improvement in the reconstruction error, and in the quality of the generated audio.

At the end of this work, the objectives of the thesis have not been successfully achieved. That said, the work exhibits decent results in the yet-born field of *neural synthesis*, which had its first breakthrough with *WaveNet* last year. At the time of writing, a first concrete musical application with *Nsynth*[1] has emerged for only some weeks.

I still have no proof that an arrangement based on an input sound can be created using deep learning techniques. However, it has been demonstrated that such techniques may be capable of learning sound *transformations*, which is one of the numerous aspects that the aims of this thesis involve.

Several limitations in this work may have significantly constrained the quality of my results and need to be addressed into detail.

[1]Here for the Google's *magenta* blog entry: `https://magenta.tensorflow.org/nsynth`

Deep learning, and particularly its musical application, is a field of research that has recently experienced a vast growth. That means, it has been evolving at a fast pace and sometimes may lack of theoretical foundation for decisions, such as the choice of the architecture and the hyperparameters for the model deployed. The broad choice in literature may provide with heuristics, but in many instances they refer to particular tasks involving a specific field, such as the one of computer vision. Furthermore, deep learning brings time and resource limitations. Training networks may require days, and some experiments may be bounded because of computational constraints, as in the case of this work. Lastly, such an active field of research brings implicitly many alternatives, which are worth being examined.

Nonetheless, I believe that this work contains results that may be used as a promising starting point for a further investigation. Currently, the Magenta team has been dealing with similar tasks and the effectiveness of its structure has been undoubtedly demonstrated, as referred in the 1.2.2.

However, the bases are quite different. With Nsynth, the Magenta team aims to give a contribution that moves from the traditional *hard-tuned* synthesizer paradigm to one which gives intuitively controls over timbre and sound manipulation. That is, the neural synthesizer is given the main task to reconstruct and interpolate eventually between the samples, creating new sounds. I differently aim to the use of neural networks for a re-arrangement paradigm using yet existing sounds.

Finally, broader research must be carried out to achieve a quality that may lead to any practical application. First, the quality of the reconstruction needs significant improvements, thus the need for testing new techniques for phase reconstruction. The phase reconstruction is however affected by the quality of the spectra reconstruction, which has to increase. The urge to create a more powerful model is of much importance as well.

At the end of this work I believe that a combination of *classical* audio signal processing techniques and deep learning techniques should be deployed when dealing with the numerous characteristics of a temporal sequence. Perhaps long time could pass before a concrete musical application be possible. Nevertheless, I strongly believe that it is worth applying machine intelligence to allow new way of expressiveness of which the performer could benefit.

Appendix A

Reconstructed log-magnitude spectra

In this appendix, I include all the images of the reconstructed spectra. The images are divided according to the task, either *reconstruction* or *transformation*. For each task, I show the *original* spectrum, that is the result of the computed Short-Time Fourier Transform over the original audio file, and the *reconstructions* for each deployed model. Moreover, the *transformation* task shows a *target* spectrum, that is the sound whose timbre the transformation aims for.

A.1 The reconstruction task

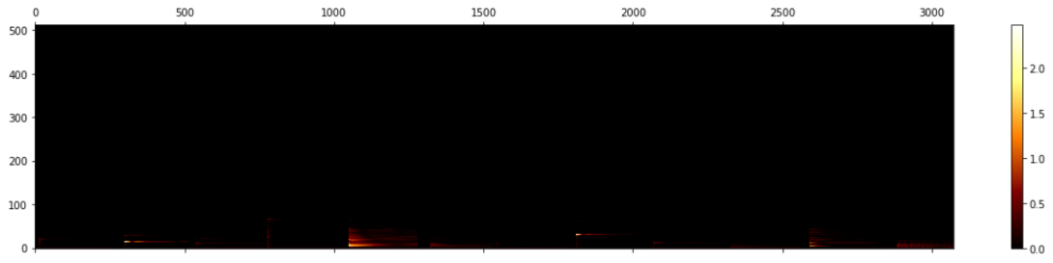

Figure A.1: The log-magnitude spectrum for the original audio sample (a sequence of piano notes).

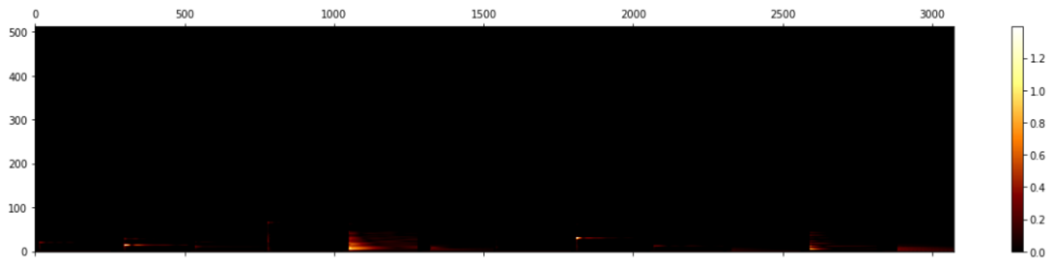

Figure A.2: The log-magnitude spectrum for the reconstruction using a shallow convolutional autoencoder.

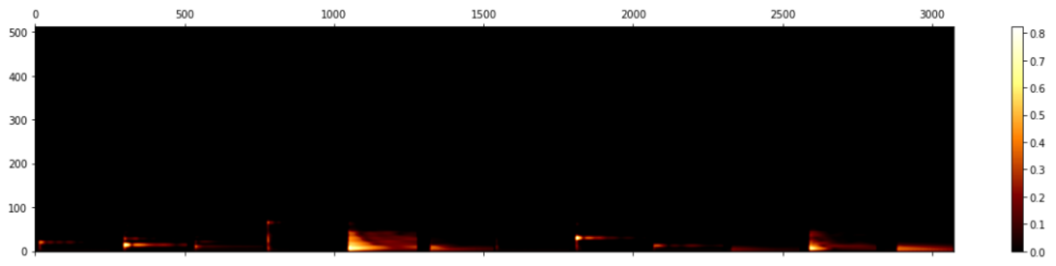

Figure A.3: The log-magnitude spectrum for the reconstruction using a convolutional autoencoder 2 layer deep both for the encoding and the decoding.

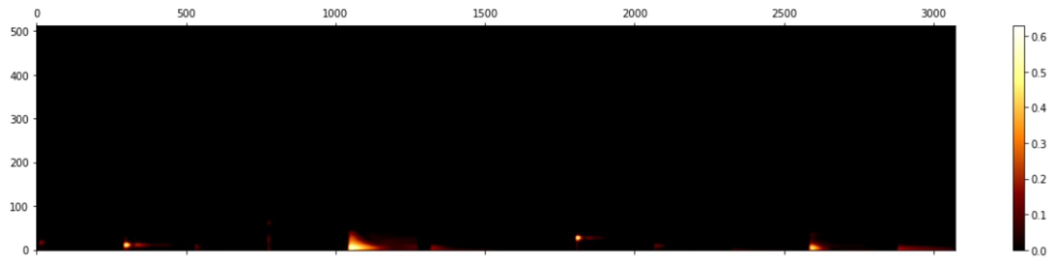

Figure A.4: The log-magnitude spectrum for the reconstruction using a convolutional autoencoder 3 layer deep both for the encoding and the decoding. It is noticeable how the magnitude range is decreasing and the notes get blurred in their contour.

A.2 The transformation task

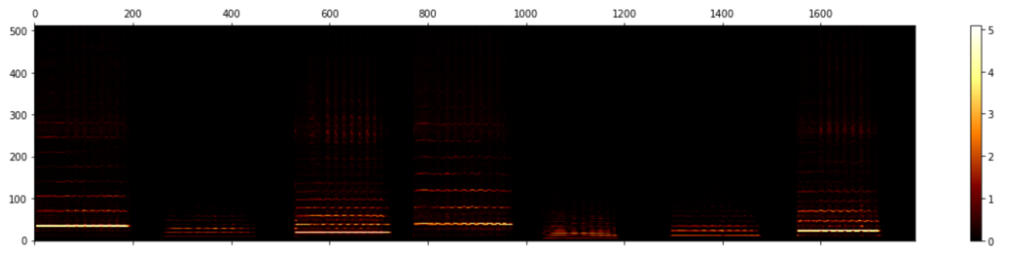

Figure A.5: The log-magnitude spectrum for the original audio sample (a sequence of flute notes).

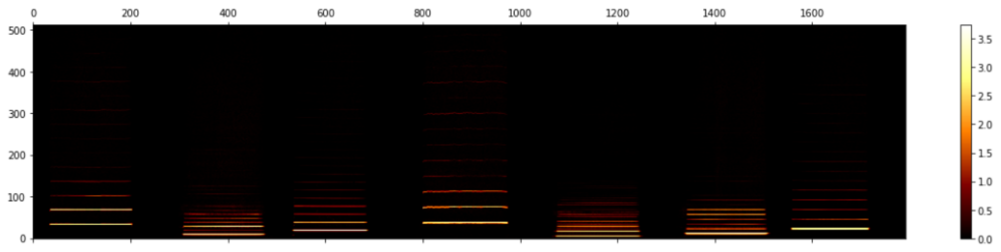

Figure A.6: The log-magnitude spectrum for the *target* audio sample (a sequence of clarinet notes).

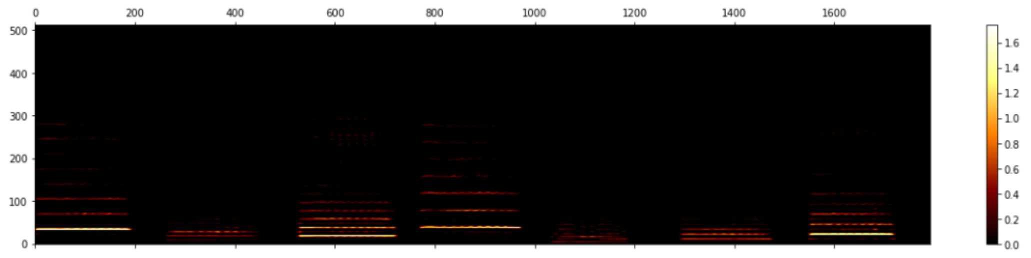

Figure A.7: The log-magnitude spectrum for the transformation using the baseline convolutional autoencoder depicted in the *reconstruction* task.

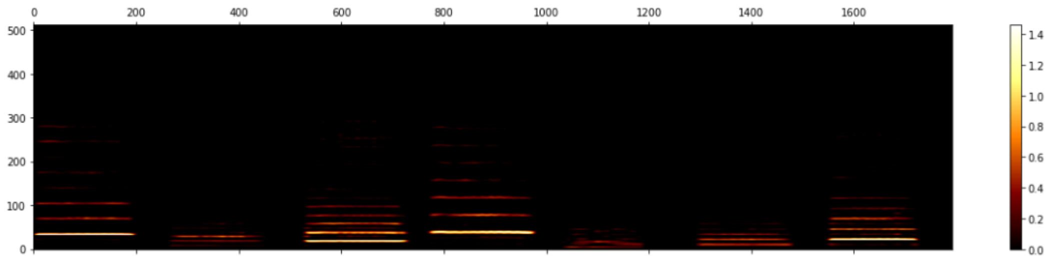

Figure A.8: The log-magnitude spectrum for the transformation using the baseline convolutional autoencoder 2 layers deep both for the encoding and the decoding.

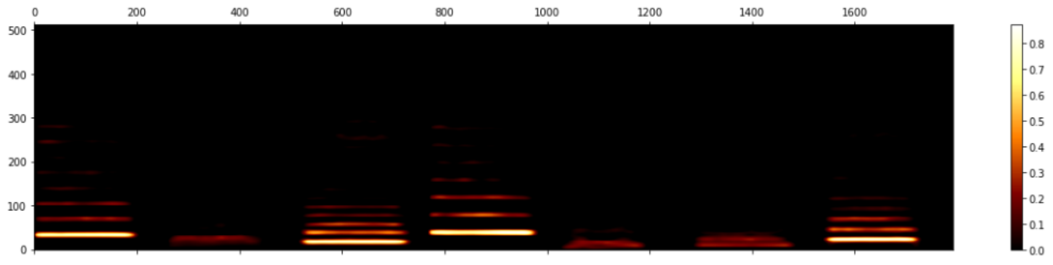

Figure A.9: The log-magnitude spectrum for the transformation using the baseline convolutional autoencoder 3 layers deep both for the encoding and the decoding. It is worth noticing how the note contours get blurred, the vibrato is no longer present but the harmonic displacement still resembles the one of the flute sequence.

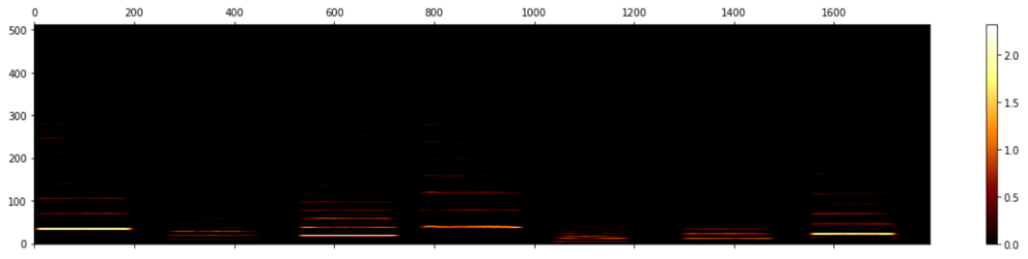

Figure A.10: The log-magnitude spectrum for the transformation using the *dilated* convolutional autoencoder. The note contours do not differ much from the baseline result, but the original magnitude scale is better preserved.

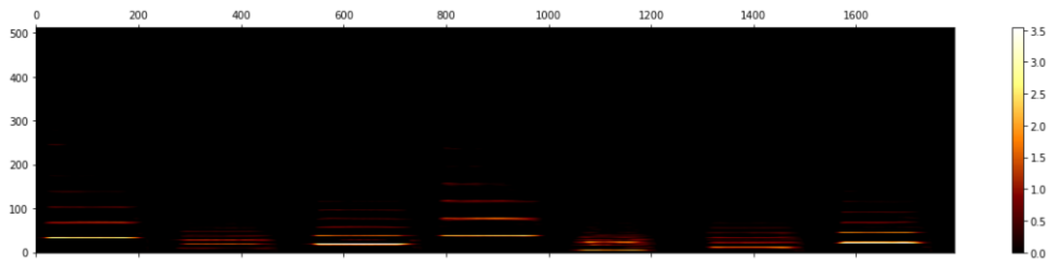

Figure A.11: The log-magnitude spectrum for the transformation using the *dilated* convolutional autoencoder with residual connections.

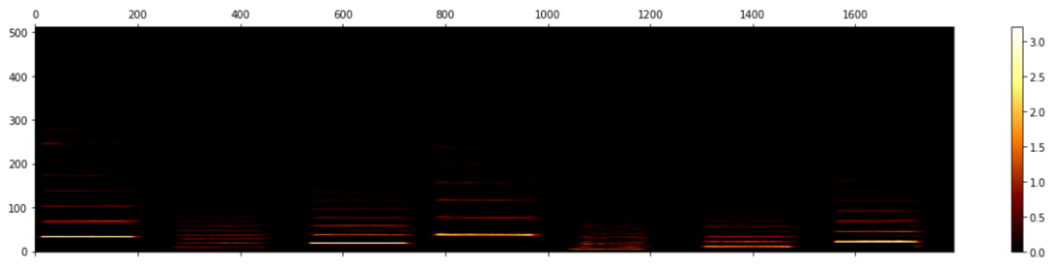

Figure A.12: The log-magnitude spectrum for the transformation using the *dilated* convolutional autoencoder with residual connections 2 layers deep both for the encoding and the decoding. It is interesting how the use of residual connections seem saving both the magnitude scale and preventing the note contours from being blurred as much as in the baseline model. I can therefore assert that such techniques improve the overall performance, leading to a fair timbre transformation. This image finally shows the best result for the task.

Bibliography

[1] Martín Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Gérard Assayag et al. "Omax brothers: a dynamic yopology of agents for improvization learning". In: *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*. ACM. 2006, pp. 125–132.

[3] Pierre Baldi. "Autoencoders, unsupervised learning, and deep architectures." In: *ICML unsupervised and transfer learning* 27.37-50 (2012), p. 1.

[4] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[5] Yoshua Bengio et al. "Greedy layer-wise training of deep networks". In: *Advances in neural information processing systems* 19 (2007), p. 153.

[6] Yoshua Bengio et al. "Learning deep architectures for AI". In: *Foundations and trends® in Machine Learning* 2.1 (2009), pp. 1–127.

[7] Christopher M Bishop. "Pattern recognition". In: *Machine Learning* 128 (2006), pp. 1–58.

[8] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription". In: *arXiv preprint arXiv:1206.6392* (2012).

[9] Y-lan Boureau, Yann L Cun, et al. "Sparse feature learning for deep belief networks". In: *Advances in neural information processing systems*. 2008, pp. 1185–1192.

[10] Carmine-Emanuele Cella. "Sound-types: a new framework for symbolic sound analysis and synthesis." In: *ICMC*. 2011.

[11] François Chollet. *Keras*. 2015.

[12] Darrell Conklin. "Music generation from statistical models". In: London: AISB Societ, 2003.

[13] David DeMers and GW Cottrell. "n–linear dimensionality reduction". In: *Adv. Neural Inform. Process. Sys* 5 (1993), pp. 580–587.

[14] Sander Dieleman. "Learning feature hierarchies for musical audio signals". PhD thesis. Ghent University, 2015.

[15] Sander Dieleman and Benjamin Schrauwen. "Multiscale approaches to music audio feature learning". In: *14th International Society for Music Information Retrieval Conference (ISMIR-2013)*. Pontifícia Universidade Católica do Paraná. 2013, pp. 116–121.

[16] Mark Dolson. "The phase vocoder: A tutorial". In: *Computer Music Journal* 10.4 (1986), pp. 14–27.

[17] Douglas Eck and Juergen Schmidhuber. "A first look at music composition using lstm recurrent neural networks". In: *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale* 103 (2002).

[18] Daniel Patrick Whittlesey Ellis and David Felix Rosenthal. *Mid-level representations for computational auditory scene analysis*. Perceptual Computing Section, Media Laboratory, Massachusetts Institute of Technology, 1995.

[19] Jesse Engel et al. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders". In: *arXiv preprint arXiv:1704.01279* (2017).

[20] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Aistats*. Vol. 9. 2010, pp. 249–256.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[22] Daniel Griffin and Jae Lim. "Signal estimation from modified short-time Fourier transform". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32.2 (1984), pp. 236–243.

[23] Philippe Hamel, Yoshua Bengio, and Douglas Eck. "Building Musically-relevant Audio Features through Multiple Timescale Representations." In: *ISMIR*. 2012, pp. 553–558.

[24] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[25] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[26] Geoffrey E Hinton. "Connectionist learning procedures". In: *Artificial intelligence* 40.1-3 (1989), pp. 185–234.

[27] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[28] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507.

[29] Geoffrey E Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

[30] Adrianus JM Houtsma. "Pitch and timbre: Definition, meaning and use". In: *Journal of New Music Research* 26.2 (1997), pp. 104–115.

[31] Eric J Humphrey, Juan Pablo Bello, and Yann LeCun. "Moving Beyond Feature Design: Deep Architectures and Automatic Feature Learning in Music Informatics." In: *ISMIR*. Citeseer. 2012, pp. 403–408.

[32] Corey Kereliuk, Bob L Sturm, and Jan Larsen. "Deep learning and music adversaries". In: *IEEE Transactions on Multimedia* 17.11 (2015), pp. 2059–2071.

[33] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[34] Tetsuro Kitahara. "Mid-level representations of musical audio signals for music information retrieval". In: *Advances in Music Information Retrieval*. Springer, 2010, pp. 65–91.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[36] Honglak Lee et al. "Unsupervised feature learning for audio classification using convolutional deep belief networks". In: *Advances in neural information processing systems*. 2009, pp. 1096–1104.

[37] Marco Marchini and Hendrik Purwins. "Unsupervised generation of percussion sound sequences from a sound example". In: *Sound and Music Computing Conference*. Vol. 220. 2010.

[38] Jonathan Masci et al. "Stacked convolutional auto-encoders for hierarchical feature extraction". In: *Artificial Neural Networks and Machine Learning–ICANN 2011* (2011), pp. 52–59.

[39] Max V Mathews et al. *The technology of computer music*. Vol. 5. 6. MIT press Cambridge, 1969.

[40] Brian McFee et al. "librosa: Audio and music signal analysis in python". In: *Proceedings of the 14th python in science conference*. 2015.

[41] Meinard Müller. *Information retrieval for music and motion*. Vol. 2. Springer, 2007.

[42] Andrew Ng. “Sparse autoencoder”. In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19.

[43] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. “Deep content-based music recommendation”. In: *Advances in neural information processing systems*. 2013, pp. 2643–2651.

[44] Aäron van den Oord et al. “Wavenet: A generative model for raw audio”. In: *CoRR abs/1609.03499* (2016).

[45] Francois Pachet. “The continuator: Musical interaction with style”. In: *Journal of New Music Research* 32.3 (2003), pp. 333–341.

[46] Jean-Claude Risset and Scott Van Duyne. “Real-time performance interaction with a computer-controlled acoustic piano”. In: *Computer music journal* 20.1 (1996), pp. 62–75.

[47] Curtis Roads et al. *Musical signal processing*. Routledge, 2013.

[48] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[49] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting.” In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[50] Ilya Sutskever. “Training recurrent neural networks”. PhD thesis. University of Toronto, 2013.

[51] John Thickstun, Zaid Harchaoui, and Sham Kakade. “Learning Features of Music from Scratch”. In: *arXiv preprint arXiv:1611.09827* (2016).

[52] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. “Dimensionality reduction: a comparative”. In: *J Mach Learn Res* 10 (2009), pp. 66–71.

[53] Pascal Vincent et al. “Extracting and composing robust features with denoising autoencoders”. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1096–1103.

[54] Pascal Vincent et al. “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion”. In: *Journal of Machine Learning Research* 11.Dec (2010), pp. 3371–3408.

[55] Hugues Vinet. “The representation levels of music information”. In: *International Symposium on Computer Music Modeling and Retrieval*. Springer. 2003, pp. 193–209.

[56] Fisher Yu and Vladlen Koltun. “Multi-scale context aggregation by dilated convolutions”. In: *arXiv preprint arXiv:1511.07122* (2015).

[57] Richard S Zemel. “Autoencoders, minimum description length and Helmholtz free energy”. In: NIPS. 1994.

[58] U Zölder. *DAFX: Digital Audio Effects*. 2002.