Interpolation Based Neural Audio Synthesis using Convolutional Autoencoders

Benedikt Langer, BSc

MASTERARBEIT

eingereicht am

Fachhochschul-Masterstudiengang

Mobile Computing

in Hagenberg

im Juni 2023

Advisor:

FH-Prof. DI Stephan Selinger
Alexander Palmanshofer, BSc MSc

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere. This printed copy is identical to the submitted electronic version.

Hagenberg, June 27, 2023

Benedikt Langer, BSc

Contents

Declaration iv

Preface vii

Abstract viii

Kurzfassung ix

1 Introduction 1

2 Related Works 2
2.1 Neural Audio Synthesis . . . 2
2.2 Audio Style Transfer . . . 7
2.3 Image Style Transfer . . . 11

3 Approach 12
3.1 Motivation . . . 12
3.2 Overview . . . 13
3.3 Pre-processing . . . 14
3.3.1 Spectrograms and STFT . . . 15
3.4 ML-Model . . . 17
3.4.1 Neural Networks - Introduction . . . 18
3.4.2 Convolutional Neural Networks . . . 18
3.4.3 Autoencoder . . . 19
3.4.4 Optimizer . . . 22
3.5 Synthesis of novel sounds . . . 23
3.5.1 Interpolation in latent space . . . 23
3.6 Post Processing . . . 24
3.7 Dataset . . . 25
3.7.1 NSynth Dataset . . . 26

4 Experiment 27
4.1 Implementation Environment . . . 28
4.2 Training . . . 28
4.2.1 Training configuration . . . 28
4.2.2 Training Execution . . . 29

4.3 Initial Experiments 30
4.3.1 Whole Spectrograms as Input 30
4.3.2 Spectrograms of signalframes as Input 31
4.4 Experiments single frequency vectors 33
4.4.1 Experiments for Synthesis 35
4.5 Experiments with slices of spectrograms 37
4.6 Experiments with mel-scale 38
4.7 Measurement of results 38

5 Results 39

6 Discussion/Evaluation 40

7 Conclusion 41

8 Future Work 42

A Technical Details 43

B Supplementary Materials 44
B.1 PDF Files 44
B.2 Media Files 44
B.3 Online Sources (PDF Captures) 44

C Questionnaire 45

D LaTeX Source Code 46

References 47
Literature 47
Software 49
Online sources 49

Preface

Abstract

This should be a 1-page (maximum) summary of your work in English.

Kurzfassung

An dieser Stelle steht eine Zusammenfassung der Arbeit, Umfang max. 1 Seite. ...

Chapter 1

Introduction

Chapter 2

Related Works

There already exist a few good approaches around the thematic of neural style transfer or generating audios using neural networks, that present rather good solutions. Some of these studies have proven, that with neural networks it is possible to generate synthesized audio up to a certain quality. Those approaches can get categorized into different areas, as their principle and methodology differ in certain ways. As this field is related to the technique of image style transfer, a lot of works apply those methods to audio, and respective audio spectrograms and thus, call it explicitly audio style transfer. Secondly because those solutions, are defining a combination specifically of content and style. This topic will get discussed in more detail in section 2.2. All methods which do not include this principle of content and style, can get categorized to the technique of neural audio synthesis or simply just audio synthesis (see 2.1). Those methods incorporate mostly autoencoder networks.

2.1 Neural Audio Synthesis

Neural audio synthesis is the field of creating or synthesizing novel sounds with the help of neural networks. The approach is similar and related to the field of audio style transfer. As described before, approaches in this domain differ in certain ways to neural style transfer. As a major difference, with respect to neural audio synthesis, no content or style sound is specified, which means, that for the creation of novel sounds, two sound sources are used equally. While audio style transfer also gets frequently applied on whole audio samples or musical pieces, in audio synthesis the focus is more directed on the application for single notes. With a special look onto autoencoder networks, neural audio synthesis also includes the tasks of learning important sound features for compression and recreation of the input data. How different approaches are designed and which machine learning techniques and which results could be obtained, will be described as follows.

Probably one of the most prominent solutions, in the field of neural audio synthesis, comes from *Engel et al.*. [6] With their work “Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders” they have proposed a system that is capable of synthesizing audio as well as interpolating/morphing encoded audio data of two instruments

to create new audio. In addition, there also exists a publicly available dataset called “NSynth” that contains a large scale of high quality musical notes. The dataset has been applied for training of the model in the course of this specific project. In their work regarding the synthesis, *Engel et al.* developed and compared two different approaches with two different kind of networks. Basically they have a similar structure, as they are both designed as autoencoders. However they do accept different formats of audio-data and thus have different components. While the one kind of network operates on time-continuous data the other one is trained on the spectral representation of audio samples. Throughout the work, the second technology using spectrograms is referenced as Baseline Model. This one consists of convolutional layers, with leaky ReLU activation, except the last layers in encoder and decoder. Furthermore BatchNormalization is applied as well. This spectral autoencoder gets compared to the a so called “WaveNet-style Autoencoders”, that are trained on continuous time signal. The autoencoder-structure enables learning efficient encodings of the music data, which are representing essential features from the original audio.
In order to create new sounds they take the encoded data from the embeddding space of two instruments and interpolate them linearly. In addition they applied the decoder part to reconstruct audio data. As a result they were able to create new sounds with the characteristics of two different audio signals. Comparing the performance of the two different networks used, they found the WaveNet-style autoencoder to be advantageous. This was proven both by the error scores for reconstructing the audios or auditory quality and by quantitative comparison with a pitch and quality classifier model. Nevertheless it was also concluded that the spectral baseline model has a strong performance.
The results regarding the WaveNet-style Autoencoder can be explored via Engel’s online AI-Experiment called “Sound Maker”.[1].

In further publications and approaches, *Engel* continued the research on neural audio synthesis by applying other network structures for this purpose. With respect to generative adversarial networks (GANs) and recurrent neural networks (RNNs), two more works have been published in the field of neural audio synthesis. [5, 10] Similar to their work concerning WaveNet-style and convolutional autoencoder, they conducted experiments in (re)synthesizing audios, as an example interpolation of extracted features has been done. Eventually the discuessed works, show a suitability for audio synthesis and also highlight a major speedup in the computation of synthesized audio samples.

The work by *Natsiou et al.* does not explicitly mention the term neural audio synthesis in its title, but deals with it throughout the article. [20] In their work they do a research on the reconstruction capacity of (stacked) convolutional autoencoders in terms of log-mel-spectrograms and carry out experiments on different configurations. In their experiments they evaluate the effectiveness of autoencoders in terms of neural audio synthesis whereas also feasible improvements through additional techniques are measured. As well they mention, that their work makes an exploration on musical timbre compression. Here the synthesis gets specifically referred to timbre synthesis. As audio spectrograms exist with different scales, this approach uses, in contrast to others, the

[1] “Sound Maker” https://magenta.tensorflow.org/nsynth-instrument

log-mel scale. This scale proves to be beneficial, as it already captures the most significant properties with the effect of consuming less memory and computational power. For the training the autors used the NSynth-Dataset proposed by *Engel et al.* [6], whereas just a sub-sample, containing samples of different instruments of one single pitch, was considered. The model(s) that were used throughout their experiments, followed the general structure of a (stacked) convolutional autoencoder network, which is composed mainly of 2D convolutional layers. For experimental reasons, additional layers and techniques such as pooling layers, fully connected layers, dropout, kernel regularization got applied (added/removed).
In order to measure the results of their experiments they applied error metrics such as root mean squared error (RMSE) and structural similarity index (SSIM). As a result these metrics cannot accurately say anything about the quality. Because of this reason, they also introduced a precision and recall score and combined it in a F1_score. In order to generate sounds from the spectrograms, they were utilizing the preserved phase information, unless there was no modification of the embedding. In the latter case, the Griffin Lim phase estimation algorithm was applied, as no phase information is present. Regarding the results that could be obtained by reconstructing spectrograms (without modification in latent space), some interesting findings could be extracted. To their surprise, by reducing the size of the latent space, they found out that the smaller it is, more accurate spectrograms with a smoother distribution could be generated. Also, in some cases where kernel regularization got applied, the spectrograms were more accurate, while with dropout layers no improvement could be achieved. The use of (max) pooling also resulted in a more accurate time-frequency resolution with less noise than with just convolution layers. Finally, after the removal of the fully connected layers it showed that the quality was significant better, as spatial information got preserved better.

Regarding audio synthesis, *Colonel et al.* published a few works, where they investigated the suitability of autoencoder networks in connection with audio synthesis. [2, 3, 17] Starting in 2017 they proposed an autoencoder based audio synthesis through compression and reconstruction of audio spectrograms. [17] In contrast to the before mentioned approaches, this one is based on fully connected layers without convolutions. Also a self-made dataset was created, with their own synthesizer. In contrast to to e.G. the NSynth dataset, this one contains polyphonic notes and thus more complex harmonies. During the experiment they trained different parameterized networks, where they vary the depth and width of the network and its layers as well as the activation functions and different optimizers. As error metric in this work the mean squared error (MSE) was used.
Comparing these scores regarding networks of one or two hidden layers on each side show, the network using the Adam optimizer worked out best in contrast to using Momentum as optimizer. These networks, based on sigmoid activation functions worked best when less compression is applied. The concept of 4 hidden layers, showed having a mix of ReLU and sigmoid activation functions worked out best. Additionally by applying regularization methods such as dropout and l2 penalty, the latter proved to be better, as the results where of better auditory quality. Further results showed that sigmoid activations led to fuller sound than with ReLU. Furthermore by using bias terms, it could be observed that noise was present in the results. As a consequence, despite of

the better convergence, they chose to let them out. The authors concluded, that using a network with 4 hidden layers and a composition of sigmoid and ReLU worked out best also in terms of auditory quality.

Another work by *Colonel et al.* was proposed in 2018, which actually states an improvement of the method, described in their previous work from 2017. [2] Those improvements contain the use of a phase reconstruction method not used before, which allows a direct activation of the latent space. For an improved model convergence, the autoencoder was designed asymmetrically, via input augmentation. This means that they padded the input magnitude data with different permutations, being first or second order difference or mel-frequency cepstral coefficients (MFCC). Whereas in the previous work only MSE was contemplated as error metric, here, a comparison of several cost functions was used. Within the cost functions, the mean absolute error (MAE), as well as the spectral convergence cost function (SC) with L2 penalty, was considered. The penalization of the total spectral power, proves to be advantageous as the power in the output is more accurate as compared to others. In comparison to their work from 2017, they also left out additional bias terms, but decided to just use ReLU-activations instead of a mixture with sigmoid.
Eventually improvements to their previous results could be achieved regarding the additional methods, that were applied. Concerning the augmentation of the input data, a significant improvement regarding score could be reached, whereas augmenting with first order difference outperformed all other. With a look onto the generated sound, it could be observed, that by padding with the MFCCs a different sound palette was present. In another comparison to their baseline, *Colonel et al.* introduced the possibility to omit the encoder part of the network. This directly enables the activation of the innermost 8 neuron layer while the decoder can generate novel sounds. As no phase information was present, estimations were done via a method called real-time phase gradient heap integration, which enables the generation of a playable sound. In addition to this work, the authors implemented a small program including a GUI, where it is possible to directly interact and activate the innermost neurons (eight control values in latent space) to generate new sounds.

In a more recent work, *Colonel et al.* implemented and compared autoencoder networks with different topologies regarding their performances for musical timbre generation. [3] This work already utilizes findings and methodologies from previous works. Based on a study from 2018, they implemented a mechanism to directly activate and control the latent space of a trained autoencoder with a graphical tool. They found out that this technique proved to be difficult in terms of controlling the latent space. To overcome this issue and improve the work, they added chroma-based input augmentation to improve the reconstruction performance in this approach. The chroma-values are based on the 12 note (western) scale to represent the dominant note present in an audio sample. Besides this type of input augmentation, they also implemented a so-called skip connection, where the latent space gets conditioned with the chroma-value. In this work the chroma-values get represented via a one-hot encoded representation for each training sample, where the maximum value is set to one while all others to zero. Consequently these one-hot encoded chroma representations tell the note played

in a single-note audio. With this technique the authors could shape the timbre around a specific note class. For the networks topologies, they varied the size of the “bottleneck”-layer (8, 3 or 2 neurons), the activation functions, the input augmentation, the use of the chroma skip connection, as well as different datasets. It should be mentioned that the authors trained and experimented with the self-generated dataset from their previous works, containing five octaves of notes, a one octave subset of it and a separate violin note dataset.
As a result, the network with an eight neuron bottleneck, with the chroma-based input augmentation worked out best. Thus, for the rest of the experiments *Colonel et al.* were using this technique. In case of the two neuron bottleneck network, the sigmoid activation functions without skip connection worked out best for the one octave dataset. The skip connection turned out to work best for the violin dataset (sigmoid and two neurons). Finally in the case of three neurons also the variant with the skip connection worked out best for both datasets. By analyzing the latent spaces some interesting observations could be made for the sake of audio synthesis. The authors applied a clustering method to see the distribution of the values in the latent space concerning their note and timbre. Using sigmoid activations turned out to bind the values in the range of (0,1) as well as distributing the values in a more uniform manner. Also the skip connections led to a denser representation. By taking this as an advantage, and moving forward with just sigmoid activations, sampling of the latent space (with a mesh grid e.g. 350x350 for two neurons bottleneck) was done to generate a new timbre. In combination with setting the additional chroma conditioning vector to a given note class, the decoder generates the timbre that matches the chroma vector and thus, the desired note is present in the output sample.

A comparative work on autoencoders, in terms of music sound modeling, has been published by *Roche et al.* in 2019. [25] In this work they implemented four different types of autoencoder networks, that have been compared in terms of audio synthesis. Similar to the other techniques described earlier, this one also orientates itself on the principle of an autoencoder, to project the input data to a low-dimensional space, from which input can be (re)synthesized. In the described experiments, the proposed autoencoder networks consist of (shallow) autoencoers (AEs), deep autoencoders (DAEs), recurrent autoencoders (LSTM-AEs) and variational autoencoders (VAEs) which all got compared to principal component analysis (PCA) as baseline. As sound data for training and experimenting, they used a subsample of 10,000 different random selected notes from the public available NSynth dataset. The networks that where implemented got trained on the normalized log-magnitude spectra of those samples. Regarding the structure or the depth of the different networks, the researchers used two and three layers for the DAE on each side. In the case of the VAE just one version with two layers, and one version of the LSTM-AE with one layer on each side was applied. Regarding the size of the output from the encoder (latent space), they experimented with different values in a range from 4 to 100. The conducted experiments consist of an resynthesis-analysis where the reconstruction error (RMSE in dB) of the different methods got compared. Additionally to the RMSE so called PEMO-Q scores were introduced to calculate the objective measures of perceptual audio quality.
The results showed to their surprise, that PCA outperformed the shallow autoencoder

network. Continuing with DAEs, the reconstruction performed almost 20% better than the shallow AE, having an encoding size of 12 and 16. Also the error decreased faster when the dimension of the latent space was decreased. Even better results with over 23% improvement compared to PCA could be achieved by using LSTM-AEs which brought them to the concolusion, that it is feasible to use more complex architectures. The fact that more compression and thus a small latent space can be generated, is even more important for sound-synthesis. In comparison the reconstuction error from the VAE was lying between the one of the DAE and shallow AE/PCA. As the size of the latent space influences the reconstruction error, it can be stated, that the bigger the size, the lower the error. Interestingly PCA outperforms all models having an encoding size of 100. In addition to the RMSE score, the perceptual audio quality got measured with te PEMO-Q score. The results are comparable to those with RMSE, with just the LSTM-AE having a slightly lower score as compared to RMSE. In this context it was also investigated how the latent space values can be used to be controlled by musicians, and thus, the correlation between those values has been calculated. Averaged over all samples per model, it was shown that the values from LSTM have the most correlation while VAE has the worst. Having less correlation makes VAE the better candidate in terms of using the latent values as control values for synthesis (less redundancy and clear perceptual meaning). In terms of audio synthesis, including the latent space variables, *Roche et al.* also demonstrated how it could be applied for sound interpolation like in the Work of *Engel et al.*. For this task they selected the latent space vectors of two sounds with different characteristics and linearly interpolated each value. By decoding and in addition, applying the inverse STFT and Griffin Lim, new interesting sounds could be generated.

2.2 Audio Style Transfer

The below discussed works, all orientate themselves on the techniques of image style transfer. As those techniques have a significant impact on the development of audio style transfer algorithms, two important works get discussed in section 2.3. Applying the method of image style transfer to audio also means, as audio is a time-continuous signal, that it has to be brought into a similar shape, which will be done mostly by generating spectrograms out of signals. As for image style transfer, a content and a style picture is needed, this principle also gets applied to audio style transfer. In image style transfer, the style (e.g. brush strokes, colors) and content of an other image (e.g. contours, scenery) get combined, to form a new stylised image. [7] This means that in the output image, the content image looks painted with a certain "style". Mapping this principle to the audio domain, this means, that there has to be a specific content sound (sample) that gets stylized with a certain style of a sound (e.g. style of a specific instrument). In the image domain it is difficult to distinguish content from style, whereas it is a bigger question regarding audio, that appears in the different approaches. Most authors define the style as a musical instruments' timbre or even a musical genre. Alongside, the content might get defined as global music structure with rhythmic constraints. [9] Those scientific questions also might be influenced if whole audio samples/musical pieces might be taken to get stylised or just some single notes from an instruments. Furthermore if speech is considered as audio data, style and content differently defined as well. Here, style could be e.g. the emotion of the voice or the speakers identity and content the spoken words in an sample. The following works show different solutions specific to the problem of Audio Style transfer in which they also get compared and assessed.

One approach that applies this principle, is the solution proposed by *Ramani et al.* in 2018. [24] In their study they developed a neural network that is constructed as an convolutional autoencoder. Here they described officially their system as audio style transfer algorithm. In this case the process of generating an audio containing characteristics of two audio signals is slightly different as in the work of *Engel et al.*. They worked with two networks, namely a transformation network and a loss network. This architecture and methodology is especially inspired by the neural style transfer algorithm by *Johnson et al.*. Both networks have the same structure and composition of layers. The loss network is trained to compress input spectrograms to lower dimensions, which means that the encoder part learns to preserve the high level features of the input. In addition, the decoder learns to reconstruct a spectrogram similar to the input of the network from the encoded data. For the training of the transformation network, the pre-trained weights of the loss network are used which speeds up the learning process. This means just optimization towards low level features/style has to be made. The trained transformation network, is then able to transform an input spectrogram into a stylised spectrogram. The loss network is subsequently used to calculate the style-loss but also content-loss between the respective spectrograms and the output from the transformation network. This loss gets minimized by back-propagation to the transformation network. Through this procedure it is possible to pass a single spectrogram through the transformation network. In the following the network outputs a new spectrogram containing the characteristics of itself (content) but also of one other style audio. Due to its architecture it also performed really fast and could be used for real-time use.

Verma et al. presented a new machine learning technique for the purpose of generating novel sounds, in their paper in 2018. [26] In this approach they tried to apply the method for artistic image style transfer, to audio and they specifically mention the approach proposed by *Gatys et al.*[7] (see section 2.3). Unlike *Gatys'* image style transfer approach, they adapted and trained an AlexNet architecture on the classification of audio-samples. This kind of network is a so called convolutional neural network, where the audio gets converted into spectrograms, which can be seen as grey-scale images. An important aspect is, in this work they used the log-magnitude data of the STFT output. It also should be mentioned, that they adapted the network towards a receptive size (kernel) of 3x3 instead of the larger ones in the original network, as it retains the resolution of the audio. Similarly to the image domain, the stylised output image gets initialized with random noise. Thus an input spectrogram consisting of a gaussian noise signal is utilized. This one gets iteratively optimized by minimizing the content, but also style loss via back-propagation. In the end this process creates a spectrogram combining the content of one audio with the style of a different audio sample. Additionally they found out that including additional loss terms for temporal and frequency energy envelopes, helped to improve the quality, as otherwise temporal dynamics would not get incorporated. For their experiments they imposed the style of a tuning fork onto a harp sound and also transferred the style of a violin sound onto a sample of a singing voice. In this way they developed a novel method for achieving cross-synthesis by using image style transfer methods.

More work was published by *Liu et al.* [19] exploring the application of technologies given from the image domain for "mixing audio". This means, that this approch focuses on using audio as spectrograms. As the previous study solely focused on the technique by *Gatys et al.* this experimient explores two more approaches. While one is inspired by *Johnson et al.*, a convolutional autoencoder coupled with a VGG classification network, the other one uses an approach with (cycle)GAN (Generative Adversarial Network). In this work they call Gatys' approach specifically "slow transfer", as the iterative computation from gaussian noise was proven to be really slow. Different to the previous work by *Verma et al.*, the authors adapted a VGG network (1 input channel in first layer instead of 3) for the "slow transfer" method. This approach has also been used in *Gatys'* image style transfer. The transfer process is also similar to the previous work, as *Liu* uses a spectrogram initialized as gaussian noise in order to iteratively minimize the content loss in the higher layers, and the style loss in the lower layers. Setting this transfer process as a baseline model, they also adapted a faster style transfer method by coupling the VGG network with a convolutional autencoder network. The purpose of this network is to take the content spectrogram as input and in the following outputting a spectrogram containing also the style features of a style spectrogram. Comparing the described work to other approaches, this proves to be similar to the one of *Ramani et al.* having a transformation network. The only difference is the second network as here they are using a VGG classification network and no second autoencoder. Having the output of the autoencoder network (also called generative network) this one serves as the initial spectrogram on which the content and style loss gets computed in the VGG network. The gradient descent then gets applied to the autoencoder network, resulting after few iterations, in a stylized spectrogram. The researchers have proven that their approach is faster than the one with gaussian noise. As mentioned before, for the third experiment they adapted a cycleGAN, which accepts audio spectrograms instead of images. In the image domain, this kind of network is able to apply style transfer to only a portion of the input images. Secondly the application of this method generates two images, as transfer is done in both directions, which means in case of audio that two new sounds get calculated.
As another result, this study showed that their approach generate the results in a shorter amount of time. For the purpose of evaluation, they listen to the outcome, but also apply objective mechanisms like visual assessment of spectrograms, consistency tests with classification and examination of signal clusters. Eventually the authors state that with the baseline approach features like the harmonic is not clear and high frequencies get discarded and that the faster transfer emphasizes on lower frequencies but is missing out on beginnings of the notes. With cycleGAN also the lower frequencies get emphasized while higher ones get discarded. The listenable results of each approach are provided online.[2]

[2] https://www.xuehaoliu.com/audio-show

As the above mentioned approaches are working on single notes, the experiment of *Grinstein et al.* has been implemented for whole audio samples [9]. Within their work they were adapting several other approaches, with neural networks from the image domain. Besides of neural networks, they also implemented a handcrafted sound texture model which got compared to the neural approaches. The latter one is composed of three sound processing steps, that in combination emulate the human auditory system. Taking a closer look on their work, especially on the neural networks, it can be said that they differ from existing ones in several ways. On the one hand, they do not use a random noise spectrogram, but use the content spectrogram which then gets stylised through their methods. On the other hand many audio style transfer approaches, explicit are computing the result with a combined loss function, that incorporates both style and content loss. *Grinstein et al.* do not make use of this concept, as they already initialize the future stylized spectrogam with the content spectrogram, like mentioned previously. On this target spectrogram, just the style loss gets optimized, as the content is already present. To mention here, they proved this method to have compelling results, as the global structure of the content sound gets preserved.
In detail the neural network-based approach the authors investigated the use of three different network architectures for the purpose of audio style transfer. Concerning all three network types, they minimized the style loss on the content sound respective spectrogram. This style loss is equally computed as in *Gatys'* image style transfer approach. It gets calculated by minimizing the error to a "style sound or spectrogram", at specific layers in the network that extract the style. Via back-propagation the loss gets minimized again at each layer, which results in a stylised content sound or spectrogram, after a few iterations. This workflow was applied to all three different network types and compared in the end. As first network they used a VGG-19 network like *Gatys*, where the input spectrogram was replicated three times in order to mach the input shape (RGB-like). By averaging all three channels in the end, they were able to obtain the final stylised spectrogram. The second network is called SoundNet, which is a convolutional network learned on unlabeled videos including sounds. This type of network operates on the raw waveform where no generation of spectrograms has to be done in advance. Finally a wide-shallow-random network was investigated with audio spectrograms consisting of just one-layer CNN (like in the work of *Ulyanov and Lebedev* [28]). As the fourth and last method they made trials with a handcrafted sound texture model that emulates the human auditory system. Even it is no neural network, it consists of three layers doing cochlear filtering, envelop extraction with compressive non-linearity and modulation filtering.
Eventually the authors came to the following experimental conclusions: While using the VGG network no meaningful results could be obtained due to the noisiness. In comparison the SoundNet yielded more relevant results despite also containing some noise. Surprisingly the shallow random network performed best together with the sound texture model. For a better understanding, the results were provided online.[3]

The work of *Ulyanov and Lebedev* has to be mentioned here, as they are often referred to be one of the first, that explored transfer algorithms for audio. [28] In fact,

[3] https://egrinstein.github.io/2017/10/25/ast.html

they took the architecture used in image style transfer by Gatys et al. and adapted it for the use on audio spectrograms. Rather then seeing the spectrogram as a picture, with the dimensions of frequency x time, they took the freuquency values as channels for the CNN. The network itself was designed as a shallow network (1-layer) using 1D-Convolutions with random weights. To obtain a final spectrogram containing content and style, an optimization is made on random noise, to minimize the loss values to a style and a content spectrogram. Instead of applying it on single notes, this approach also uses longer samples or music snippets.

2.3 Image Style Transfer

In the previous sections it has been written about neural audio synthesis as well as neural audio style transfer. A lot of these works especially those proposing solutions for neural audio style transfer, took their inspirations from the image domain. For this reason this section makes a short excursion on relevant image style transfer works of *Gatys et al.* as well as *Johnson et al.*. [7, 15] *Gatys et al.* were the first to implement a system of neural style transfer applied on visual data. By using a convolutional network trained on object recognition and localization (VGG-19) in images, they were able to extract the texture as style but also the content of an image. They found that especially from higher layers in the network, high-level features of the images, can get reconstructed. This includes objects and their arrangements in the scene, without the exact pixel information, which will be used as content representation further on. Using a special feature space for texture synthesis, the style of a content image can get extracted by using the feature responses at certain layers in the network. By combining these two principles, respective style losses and content loss can get computed which will be used for the style transfer. The generation of the target image starts by initialization of a random noise image, on which those losses get minimized by using gradient descent.
Johnson et al. developed an improved image style mechanism on the basis of the former methodology, that particularly shows improvements regarding computational speed. For the computation of content and style losses they use a VGG network with 16 layers and pre-trained on image classification. Here they added a special transformation network, that is designed as autoencoder. This one takes a target image as input (content image) and synthesizes an image on which the style and content loss gets calculated in the VGG network instead of a random noise. Back-propagation was performed just in the transformation network, while the VGG network stayed fixed. As a consequence the transformation network produced a stylised image after training. By comparing Johnson's work to the method by *Gatys et al.*, it shows a significant improvement regarding computational speed but also yields promising results.
Previously described image domain methods significantly inspired the development of audio style transfer algorithms, which are presented in section 2.2. Finally, it should be noted that works by [24] and [19] adapted and applied the method of *Johnson et al.* while all other in section 2.2 mainly used the methodology proposed by *Gatys et al.*.

Chapter 3

Approach

In the last section a few approaches have been outlined and discussed, that have successfully implemented methods regarding the creation of audios/music with a neural network approach. As seen, these have been mainly categorized in neural audio synthesis and neural audio style transfer. This work is mainly influenced from the area of neural audio synthesis, and can be categorized as such, as the methodology and workflow is strongly related to those works. Nevertheless regarding certain components, it has also its influence from the style transfer methods, despite not defining a specific content or style audio respective loss functions.

This chapter will therefore dive into the methodology and exact workflow of this works' solution, to the problem that also will help to derive the answers to the defined research questions. First an Overview/Motivation should provide the reader with the intended idea and an overview of the applied methods, to get a general understanding of the idea (see section 3.1. Later on the single steps and components that are needed, to reach the desired functionalities, are getting in detail, starting with the pre processing. Further on the ML-Model (neural network) will get described, as well as the step that is done to synthesize new sounds. Further on, the required steps for (re)generating a listenable audio as well as an description of the used dataset for training and also all experiments conducted later on (see chapter 4).

3.1 Motivation

Like mentioned in the beginning of this thesis, this work aims to explore the possibilities of machine learning techniques such as neural networks, to apply in the audio domain for sound generation. This idea is mainly inspired by the idea of taking two distinct audio sources and mixing their characteristics in order to generate a new sound. As seen in the previous chapter, this idea is strongly related to the image domain, where the "synthesis" of a new pictures based on two source images, is commonly known as image style transfer (point 2.3). This technique, having a content image to be stylised with a certain style from another image, would mean for the application in the audio, to have a style sound to be transferred onto a content or target sound. Such methods are specifically known as audio style transfer and can either be applied to single notes or also whole audio samples or songs. Having the principle of content and style this would mean, that of one sound the global structure and rhythmical components get preserved while imposing style (e.g. the timbre) on it to generate audios. The details to these approaches, have already been outlined in the previous chapter, when describing some existing work around this topic.

Neural audio synthesis is another method for neural sound generation, which does not apply the principles of style and content audio. In the previous chapter, some insights could be gained, how neural audio synthesis can look like, as well as how it can be achieved using different methods and neural networks. Most of those methods, were showing promising results, either concerning the auditory quality but also the possibilities that arise in experimenting and designing sounds. Those methods were applying most of the time so called autoencoder networks, that can be used for dimensionality reduction of input data, as they have a so called "bottleneck" in the middle. [12] Because of this structure, the compressed data in this "bottleneck", can be seen as a representation for essential features that either can be combined/interpolated or directly synthesized. To generate synthesized audio, the solutions described in section 2.1 took advantage of the "decompressing part" of the network to generate in order audio data. The exact workflow and methodologies for sound creation, have already been mentioned in the chapter related works (see chapter 2, point 2.1).

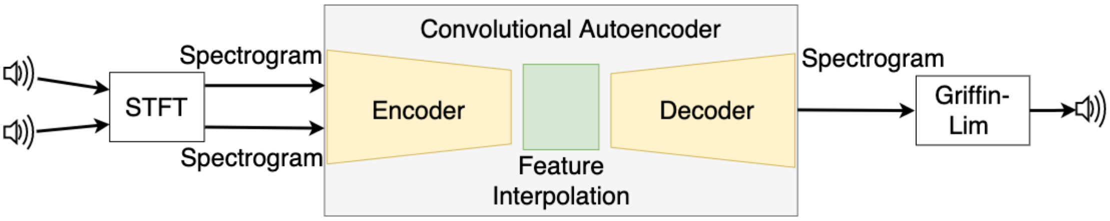
Out of those methodologies, when having the idea of using two instruments' characteristics, to generate audio, the approach of *Engel et al.* [6] using convolutional and WaveNet-style autoencoders yielded the most promising and interesting results. This can be said especially in terms of output quality but also concerning its implementation and reproducability. With a provided interactive web application, the results of this solutions can be explored, whereas different sounds can be mixed based with a certain ratio. The results in the web application are based on the WaveNet-style autoencoder but according to the scientific article, the convolutional (baseline) yet also provides strong results. Implementing an approach with a WaveNet-style network would also go beyond the scope, not at least as the computational costs would be too high. As also some audio style transfer methods, especially the approach by *Ramani et al.* [24], are using convolutional autoencoders, this kind of network was chosen to be preferable, to be applied in this work. How (convolutional) neural networks work, especially concerning autoencoders, will get described later on. This gets done with special respect, on how those functionalities help to carry out neural audio synthesis, but also to gain general knowledge and a better understanding.

3.2 Overview

Based on the motivation and existing approaches, this work aims to propose a system, that uses a convolutional autoencoder network, for the task of neural audio synthesis. This systems' goal is to take two distinct audio samples as input, whereas the significant features, of those get extracted and interpolated, to (re)generate a novel sound in the end. In figure 3.1 the general workflow of the toolchain is depicted in order to get an understanding, how this system is built up.

Starting on the very left, two audios are taken and have to be brought into a suitable

Figure 3.1: Overview of the proposed solution



representation for this type of network. As audio is in its raw form a time-continuous signal and the input for convolutional networks are of a different shape (e.g. images) some pre-processing has to be done. In this case the short-term Fourier transform (STFT) is applied in order to generate a spectrogram, that shows the frequency spectra over the time. The frequency spectra contains on the one hand the magnitude (power) of the frequencies but also the phase information. For this purpose, only the magnitude data gets used, as recent publications stated that it contains the most descriptive data of an audio (spectrogram). The autoencoder model then takes the magnitude data as input, from which a compressed representation with the essential features gets generated by the lefthand (encoder) side. Having those features of two different audio samples, those get linearly interpolated, to generate one feature vector representing the “mixed” features of two instruments. This new vector, gets passed through the righthand (decoder) side of the network, which regenerates again spectral magnitude data of the same dimension as the input. In order to obtain a “playable” audio sound, it gets transformed back into time-domain with the Griffin-Lim algorithm [8] or the inverse short-term Fourier transform (ISTFT). The latter will be applied if there was no interpolation in the embedded space, as the phase information can be reused. Corresponding terminologies as well as a detailed insight into each step and its functionalities are given down below in the following points.

3.3 Pre-processing

Pre-processing is the task of preparing raw data for a specific purpose. Moreover it is an important component of machine learning techniques, with respect to training neural networks. Deciding which pre-processing technique(s) to use on the one hand depends heavily on the type of ML-problem that has to be solved or even the training method that is chosen, but of course also on the type of data itself. As written before, for this work a neural network consisting of convolutional layers, has been chosen to be applied to the problem of neural audio synthesis. As convolutional neural networks are known for image processing tasks, they also can be applied for audio data, which has already been outlined in recent works in this field (see chapter 2). In contrast to image data which most of the time has a 3D shape (width x length x RGB-colors), raw audio has a different structure in its data representation, as it is a time-continuous signal (1D-shape). To in order bring the audio data in a similar shape, it has to undergo some pre-processing steps. Some representations of audio that have a image-like shape and that got proven beneficial regarding neural audio synthesis, include e.g. log-magnitude spectrograms or Mel-spectrograms but also chromagrams and Constant-Q Transform as stated by [1]. Taking recent works into account, this work chooses to use the first representation as this one also got used more frequently and had promising results. As for comparison in the experimental part of this work, also the use of Mel-spectrograms will get assessed and discussed concerning the synthesis task and the performance of the neural network.

As the practical part of the project to this thesis is implemented in python, a special library was used for the pre-processing part. For this part the library librosa [27] was used, as it provides practical functions for audio processing, that were considered as useful for this work. Special to mention here are the functionalities as calculating spectrograms (STFT) but also transforming spectral data back into time domain to generate playable audio data (ISTFT, Griffin-Lim).

3.3.1 Spectrograms and STFT

Spectrograms represent a 2D-representation of an time-continuous signal, which essentially shows the presence of certain frequency bands over time. Like previously said, there exist different forms of spectrograms e.G. log-magnitude and log-mel spectrograms. Especially speaking of the log-magnitude spectrogram whose calculation is based on the short-time Fourier transform (STFT) and thus on the Fourier transform. The Fourier transform, takes a frame of N values of an (audio) signal and transforms it from the time domain into the frequency domain. Generally said, that the bigger the frame, the better is the frequency resolution. What this means in terms of calculating the spectrogram, will get outlined shortly. Whats also important to mention at this point is, that the result of the Fourier transform, consists of an array of N complex numbers, which are mirrored in the middle. Every complex number in this array stands for a so called frequency bin in the signal. The real part of these numbers would represent the power/magnitude of this “bin” and the imaginary part gives information about the phase. Coming back to the frequency resolution, this for example means that when taking a one-second signal with a sampling rate SR and performing the Fourier transform with length $N = SR$ on it, this would yield an array of N values. The first value in the result depicts the signal’s offset whereas all values from 1 to N/2 are the frequency bins with a resolution of 1Hz per bin. This means that each of this bin shows the magnitude and also phase of each frequency from 1 to N/2 Hz. The ongoing values in the result, show the same values except they are mirrored, as they depict the negative frequencies. Because of this behaviour, the second part can be omitted for further use. Now these values just show the frequency spectra of one time frame and do not incorporate more information about the change. To overcome this shortcomming, the Fourier transform can get applied to a series of frames of the signal in order to obtain multiple frequency spectras over time that get depicted as a spectrogram.

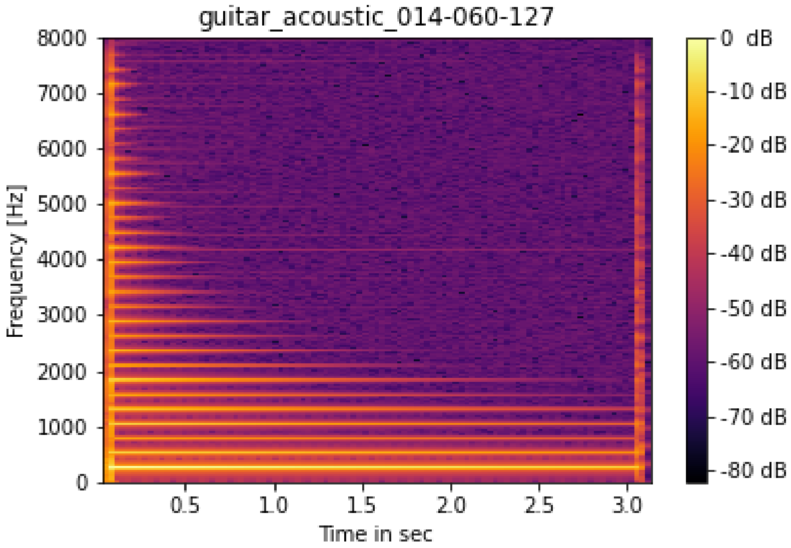
The calculation of multiple frequency spectras over time is done via the so called short-time Fourier transform. This form of calculation is widely used for pre-processing of audio data for ML-Tasks (for example see chapter 2). When applying this transform, a few parameters have to be considered, as those influence the result but also the quality for the later workflow. One of the the most important parameters is `n_fft` as it specifies the actual length of the signal-frame, on which the FFT (fast Fourier transform) gets applied to. This parameter therefore influences the frequency- but also time-resolution in the final spectrogram. To be mentioned regarding the official Librosa documentation, this should be a value of a power of two, as it speeds up the computation of the FFT behind. Another important parameter would be the `hop_length`, which defines how much audio values are between the beginning of the first and the following frame. This means that when defining this parameter to `n_fft/2` this would yield in a 50% overlap of the following frame. Modifying this parameter, would mean to either increase or decrease the overlap and also the amount of time columns in the result, as more overlapping frames occur. The overlap of the frames is also coherent with the chosen window function for the STFT. As every time when the FFT gets applied to a frame, this one gets multiplied with a so called “window”. Multiplying a signal frame with a window has to be done, as the FFT assumes, that the transformed signal is periodic (repeating itself infinitely). [11] This gets problematic when the input signal does contain frequencies that may not directly fall into a frequency bin, due to the FFT’s frequency resolution. Due to the assumed cyclic continuation the Fourier transform will ’think’ that there is a discontinuity and will spread therefore the power over all the spectrum. There exist multiple window functions such as “Hann”, “Hamming”, “Blackman”, etc. which start at (almost) zero, rises to a maximum in the middle but falls again to (almost) zero at the end (symmetric). Multiplying the signal frame with such a window function helps to overcome this issue as it removes the discontinuity. On which window to chose, depends on the use-case of the application, whereas throughout this work a “Hann”-window was chosen all the time. Coming back to the relation with the window overlap, if no overlap would be used a lot of information of the signal would get lost. This is because when multiplying the signal frames with the window functions, this would bring very little or zero values at the beginning and end of the frame. [11] When having overlapping frames this issue would get corrected. Important here is again the amount of overlap as this is dependent on the window and its wideness. Using the “Hann”-window a common value for the overlap would be 50% which was also considered throughout this work. This value is also beneficial later on, when performing the inverse transformation, back into time domain, but more on that in section 3.6. Beside of these parameters some more exist, for example for specifying the padding of the signal whereas in this work a constant padding on both sides of the signal has been used, which is also the default setting.

Now having the knowledge of the STFT and its parameters, it can be applied onto a signal, to generate a spectrogram. For example by using an audio sample with a sample rate of 16 kHz and applying the STFT with a `n_fft` value of 1024 and a `hop-length` of 512 this would result in a spectrogram with a frequency resolution of 15,625 Hz and time resolution of 64 ms. As explained before, the values of the result, consist of complex numbers which contain the magnitude but also the phase at each frequency bin. By setting this result absolute, or calling the function `librosa.magphase(spectrogramm)` the real magnitude data can be obtained, whereas the latter also retrieves phase information in a separate vector. The magnitude here displays the energy values of the spectrogram, whereas for further processing and also to be better displayable those get converted into a dB-scale. [1] This function also takes a reference value that gets set to 0 dB, which in

[1] normally the magnitude would need to get squared to obtain the power, but in this case magnitude

this case will be the maximum value of the magnitude spectrum. As for post-processing when converting the dB-scaled data back into energy, also a reference value is needed, this one gets preserved, in order to get the same scaling as in the original input. Finally when having the log-mag spectrograms in dB, those were considered for the training of the neural network afterwards. An example of a log-mag spectrogram can be seen in figure 3.2. As also the phase information was obtained when calculating the magnitude data, this one was also preserved next to the energy reference value for the recreation of signals, as it is needed there (this will mainly affect the recreation of single samples, without interpolation in embedded space, but more on that later on).

Figure 3.2: STFT log-mag spectrogram of a guitar note



This section describes the general workflow of the pre-processing from taking a signal and converting it into a spectral representation. This workflow is a basis on which different experiments with different parameterization (size of n_fft, etc.) of the calculation of the spectrograms but also with additional steps (log-mel scale, additional framing, etc.) are being made. Those steps will get mentioned later on in chapter 4 when describing the experimental part of the thesis.

3.4 ML-Model

The main or core component of every machine learning project is of course the model itself, as it achieves the main task of prediction or inference to a given problem. Those models exist as different technologies that perform regression tasks or even classification tasks. Dependent on the use-case, but also the kind of data that is present, different models are better suited or not. To count technologies, there exist the KNN-Algorithm, Decision-Trees, Random Forests, Support-Vector-Machines (SVM) but also neural net-

without squaring was taken

works which can be applied in a variety of usecases. Especially the latter, the neural networks, are able to achieve a variety of different tasks, as they are highly adaptive regarding their topology, used layers, but also the size and shape. Those variety of different tasks spread across different domains, so also for images and audio.

3.4.1 Neural Networks - Introduction

Generally said a neueral network can be seen as a graph of connected nodes with numeric values that can achieve transformations between patterns using message-passing algorithms. [16] Those nodes are commonly structured in layers, where there especially exist certain nodes or even layers that are seen as input nodes/layers and some as output nodes/layers. Between the input and the output there can also exist so called hidden layers, expanding the depth of the network. The links between the nodes, that get also called neurons, are connected via links, that are parameterized with weights, that get optimized using learning algorithms. Each neuron receives its weighted input (activities) of its connected predecessors, which get converted into a single output that gets broadcast to all its connected successors. [13] The latter involves a so called input-output function which is also commonly known as activation function (e.G. ReLU, Sigmoid, Softmax ...). Important to know, is that the weights on the connections define how much this value influences the input of the connected node. When a neural network gets trained to a specific problem (e.g. classifying certain images), using predetermined training data, the output of the neural network gets compared with the desired one, resulting in a certain error (metric). To minimize this error, the weights in the network get adapted by back-propagating the error through all the layers, to the beginning. On this way it changes the influence of certain connections and therefore the overall outcome. This procedure gets repeated on all training data over several iterations, until the error gets low to produce the desired output. The initialization of the network and its weights are often random, which also means that every training run starts and progressses differently.

Depending on the problem to solve / what's the desired output, neural networks can be trained using labeled data (supervised) but also just by minimizing a cost function (unsupervised).[21] More details on the learning will get mentioned later on when explaining the model itself.

3.4.2 Convolutional Neural Networks

In this approach, due to the promising usage in existing solutions, a convolutional neural network has been chosen as model. Convolutional neural networks are a type of networks, that get primarily used for tasks in the image-domain for example to recognize patterns in pictures or classification, but also like seen in chapter 2 for image style transfer. They have the advantage over traditional neural networks, that they can cope with the dimensionality of pictures (width x height x colors/depth). The layers containing convolutional nodes have as learnable parameters, so called kernels. [21] Those kernels, if taking a 2D-Convolutional layer, are normally small in width and height (e.g. 3x3) but span the whole depth (channels) of the input (in the case of RGB-pictures depth of three). Those kernels calculate the skalar-product for each value contained in the kernel and the input map. This yields, having e.g. a 3x3 kernel operating on a 3x3 field, in a single value. This value is the weighted sum of the kernel's values and from those of the input vector. This operation will be applied to each 3x3 field along the spatial dimension of the input, resulting in a smaller activation map. One can also apply padding around the input, to preserve the dimensionality. Furthermore a stride can be defined, which defines how much these convoluted fields overlap, as using a bigger stride would result in a smaller overlap. Having also a smaller overlap would result in a much smaller activation map. For example taking a 7x7 input, by applying a 3x3 kernel with no strides and also no padding, this would yield an output field of 5x5. With a padding of 1x1 the output would be of the same size. Finally if the stride would be 2, the output field would be of 3x3. Through training of the network and back-propagating the error, those values in the kernel get adapted in order to learn certain/important features or patterns on which a classification or pattern recognition can be made (easier). As it got here described for 2D-convolutions, depending on the input dimensionality it can be also be applied as 1D-convolutions or even 3D-convolution.

With the knowledge, that such convolutions are successful on image data, those can be also applied on audio provided in the shape of a spectrogram. As described in the previous section (see 3.3) spectrograms can be described in a "picture-like" shape having the dimensions of frequency x time with a depth of one. Speaking of that, the spectrogram can be seen as a grey-scale image. The energy in different frequency bands over time with its variations, can be seen as "recognizable" patterns. Taking a 2D-convolution, the kernel (e.g. 3x3) takes a 3x3 frame of frequency x time which results in one value. Summed up, this results in an activation map smaller or equal (if zero padding is used) than the input. After training on different samples and iterating several times, those activation map would contain the most significant features or characteristics of the spectrogram for example when training on classification. Depending on the chosen hyper parameter for the amount of output channels (depth), the resulting activation map can be of depth one or even deeper.

As mentioned before, each neuron in a neural network has an activation function, which is also the case for convolutional neural networks. As there exist different kinds of activation functions, for this approach mostly ReLU (Rectified Linenar Unit) activation functions got used but also LeakyReLU. Additional to this activation function, Batch-Normalization gets applied after each convolutional and ReLU non-linearity component. The choice is mainly based on already existing approaches like from *Ramani et al.*[24] and *Engel et al.*[6], as it was proven to yield promising results in combination. According to *Ioffe ad Szegedy* [14] applying Batch Normalization also improves the training speed (number of iterations/epochs) and enables use much higher learning rates. Furthermore it acts a regularizer, so that overfitting-reducing technologies, such as Dropout, can be omitted.

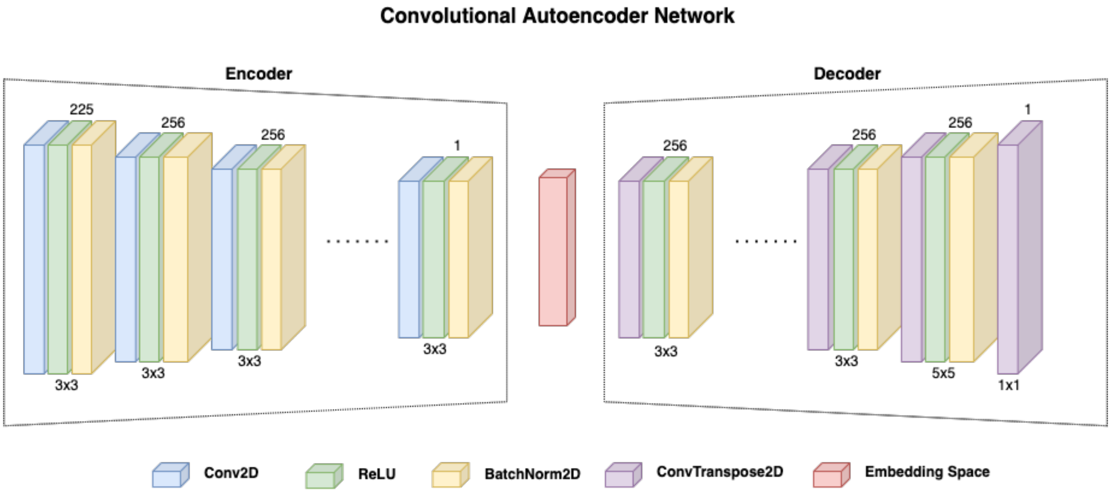
3.4.3 Autoencoder

Neural networks exist in different shapes and compositions, depending on desired work it should fulfill. Whereas some networks e.g. for classification of a given input, might reduce the width of the layers towards the end, some are designed to have a so called "bottle neck". This means, that this kind of network, contains a smaller central layer then the input, but at the end again a bigger layer (eventually same size of input layer).

[12] Those networks are called autoencoder, in which the first part of the network (until the smaller central layer) gets called “encoder”. The second part beginning at the small central layer, that is getting bigger towards the end, gets called “decoder”. These parts are called this way, as for the encoder part, it “encodes” the high-dimensional input data to a low-dimensional representation (output of small central layer). The counterpart is therefore called “decoder” as it decodes the low-dimensional data, to bring it again to a higher-dimensional representation (mostly same as input). The lower dimensional output of the small central layer, can be named differently, as for example “code”, what *Hinton et al.* does when describing the work of autoencoders in his publication [12]. Some other common names would be e.g. latent space, embedding space, encoding or embedded data. This principle of dimensionality reduction, therefore means, that the encoder part extracts the most important or characteristic data, from which the decoder is able to reconstruct the input data.

In the related works chapter (2) a few works have made use of this principle to extract “characteristic” features for audio data and synthesize audio from it, by altering them. This knowledge encouraged this work to also make use of this principle to synthesize audio by using autoencoders. Especially when having the extracted audio features as encodings, those can be modified (easier), to in order synthesize novel sounds from them using the decoder part of the network. Combined with the knowledge to apply convolutional networks to audio spectrograms and the advantages of autoencoers to extract features from the input, the model in this work is designed as convolutional autoencoder. Also the works of *Ramani et al.* or *Engel et al.* made use of these kind of network to generate novel sounds. Figure 3.3 shows the basic autoencoder network structure which is used throughout this thesis work. To be mentioned, the whole implementation of the neural network for this work has been achieved with the deep learning library PyTorch.[22]

Figure 3.3: Basic autoencoder structure used throughout this thesis



In this figure, the depicted autoencoder, gives an insight of which components and layers it is composed. The amount of layers as well as the parameters shown in this sketch, are just an example, as throughout the experimental part, those get varied. Therefore, the dotted lines in this figure, act as a placeholder for possible more layers on each side in the network. The numbers above the individual layers, represent the amount of output channels (depth) and those on the bottom, represent the kernel size. Some paragraphs before, it has been mentioned, that in this approach the convolutional layer gets equipped with ReLU activation function and BatchNormalization. This also gets visualized in this figure, where each layer is shown as combination of sub-layers that incorporate those three components. Speaking of the composition of the layers, it can be seen in the decoder part, that instead of a convolutional layer, a convolutional transpose layer is used. That is because of the nature of convolutions, as their output is always smaller or equal sized, which has been mentioned under point 3.4.2. As the decoder part generates output that is bigger than its input, the layers have to perform upsampling. In convolutional nets this is typically done via the convolutional transpose layer.

This transposed convolution is not the reverse operation of a convolution, but it is more a operation to recover the shape from the convolutions input. [4] Taking the convolution example from above, by taking as input a 5x5 field and applying the transposed convolution, with a kernel of 3x3 this would result in a field in 7x7. The equivalent of this operation would be a convolution with the same kernel, on an input of 5x5, with 2x2 padding (padded input = 9x9), as this would result in a 7x7 output too. When in the convolution, striding was applied, this also works for the transposed convolution. Again when having the 3x3 input, by applying a 3x3 kernel with a stride of 2, this again results in a 7x7 output field. To be mentioned, the striding parameter for the transposed convolutions, defines how much zeros are added between the values of the input. This means, when taking the previous example, that the 3x3 input gets one column respective one row of zeros inserted after each value to result in a 5x5 field.

With this knowledge, convolutional transpose layers are best suited to be in the decoder part of convolutional autoencoder networks. Coming back to the architecture of the convolutional autoencoder in figure 3.3, those convolutional transpose sublayers are also coupled with ReLU activations and BatchNormalization, with an exception to the last layer.

Having this kind of autoencoder for this approach and the idea to extract features of spectrograms using convolutions, this models task is to encode and decode spectral audio data. The network is therefore configured, to produce an output of the same dimensionality as the input of the encoder. Like mentioned above, the output should be a reconstruction of the input, that gets inferred, by decoding the extracted features in the embedded space. To achieve this goal of reconstructing the input, the network has to be trained, through minimizing a specific error. In some cases also maximization is desired, but is dependent on the error score and the desired outcome. In the case of an autoencoder, this works by comparing the output of the decoder with the input of the encoder, by calculating the difference. Generally said, depending on the outcome and the goal of a training, there exist different metrics like mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE) but also more specific formulas. Throughout the literature, those error or error functions are also often called cost (functions) or loss (functions). For the experimental part of this work, the choice has been made, to use MSE as error metric, as it has been used successfully by some existing works.

To minimize this error an optimization of the network has to be made, which will be done through back-propagating the error score. Through this step, the parameters (weights, bias, convolutional kernels, etc.) get adapted according the error, which in the best case, improves the score and thus the output of the network. In this work the output of the network are reconstructions of spectral data for example slices of spectrogram. As having the encoder-decoder structure, those output data get reconstructed through the decoder, which takes the encoder output as input. As mentioned before, this generated data of the encoder, is a compressed vector, consisting of the most significant extracted from the input. Depending on the configuration of the network, especially the encoder part, this vector also called encoding, can be of different size. This is because, when using more convolutional layers with strides, the output of each layer gets more downsampled, which results in a smaller encoding. As a consequence, the decoder part has to learn to regenerate spectral data from this encoded vector. This vector therefore is a compressed representation of the models input, which contains the most significant characteristics, of this specific spectrogram and further on, of the sound. All in all regarding the learning, it can be said, that the encoder learns to extract essential features, from which the decoder learns to regenerate in the best way, the input.

Altering those encodings, has therefore the consequence, that the output of the decoder is different, and thus will be a novel, synthesized sound. More on how it should get altered, gets discussed later on. Choosing the right compression, is also an important part, that influences the outcome and thus the quality of the desired model output. Too much compression, may lead to the fact, that the decoder has too less information to infer the desired spectral output, which in order results in poorer quality of the resulting audio. On the other hand having, less compression, results in embeddings being not significantly smaller then the input, containing less important data too (e.g. noise). It also may gets more difficult to alter those encoded vectors. Throughout this work, different amounts of compression have been applied in the experimental part, which get discussed later, including the impacts and observations made on that. Knowing those properties and behaviour of the autoencoder model, this strengthens the idea, to apply autoencoders for the use of neural audio synthesis.

3.4.4 Optimizer

Coming back to the training process, where the network gets optimized, in order to minimize a certain error function and as consequence helps the network to converge. For this optimization, different strategies exist, where hyper-parameters such as learning-rate or weight-decay play an essential role. Those optimizations are on a large scale, stochastic gradient-based techniques, to which algorithms such as stochastic gradient descent (SGD) or Adam can be counted. Those algorithms influence and improve the convergence which means to find a minimal error. Throughout this implementation the Adam optimizer [18] has been chosen, as it is used widely in recent publications where promising results, could be achieved. Also regarding the training process in this work, Adam optimizer was proven to be advantageous, in contrast to SGD. The parameters that have been found to have the most impact on the optimization process during training, are the previous mentioned learning-rate and weight-decay. To be mentioned, the learning-rate specifies how "fast" the model actually learns while weight-decay works as a penalty for the weights optimization to prevent overfitting. If the learning rate is chosen rather big, then the network learns faster, but because of its big steps or jumps, it could miss out the optimal solution in the solution space. Also it could happen, that when optimum is found, that it "jumps" out again. In this case a smaller learning rate would be desirable, as it makes smaller steps. Choosing it too low, would end up in a slow training where also large areas of the solution space are not visited. The latter leads to get stuck in a local minimum. Therefore it is important to choose the right size of learning rate, but this issue depends also on the problem size and type of network. In this work different learning rates have been applied throughout the experimental part where also different findings could be made, but this will be shown and discussed later on in this thesis.

With this knowledge, it can be stated, that a high learning rate, could be advantageous at the beginning of the training process, in order to rather find a global optimum and explore the solution space. In order to prevent, to jump out of minimum, a smaller learning rate would be desirable later on in the training. For this case, there exist some mechanisms to decrease the learning rate, later on in the training, especially when detecting "oscillating" due to a too large learning rate.

In this works' implementation, for the start of the training a specific starting learning rate has been set, while throughout the training, it gets adapted, when no more optimization and eventual oscillation gets detected.

3.5 Synthesis of novel sounds

Having now covered, the important properties of the pre-processing but also of the applied machine learning model, this section explains the methodology to synthesize a novel sounds. In the chapter 2 where related works got discussed, an insight could be gained, on how different works tried to synthesize audio with their neural networks. For example *Colonel et al.*[2, 3, 17] suggested in their works, to synthesize novel sounds, through directly activating the innermost layer. This means after training, for the sound creation process, the encoder part gets omitted. For example in the case where they had a network with 8 neurons at the encoders bottleneck, 8 different values could get defined, within a certain range. The decoder part then created, based on its training on recreating spectrograms, spectral data that got converted back to time domain to form a synthesized playable sound.

3.5.1 Interpolation in latent space

Having this as one possibility, to use in particular autoencoders as tool for audio synthesis, there also exist some more interesting approaches. One of those got applied by *Engel et al.* [6] but also *Roche et al.* [25] where they also made use of the latent space encodings. Contrary to omitting the encoder part, those works aimed to utilize the whole network, as no direct activation of the innermost layer is considered. Instead the authors proposed to take the encoded values of two different audio samples (possibly two distinct instrument) and combine them via linear interpolation. The interpolation process in order yields a vector of interpolated values. This new vector, can be seen as a modified encoding, which gets processed by the decoder part, resulting in spectrogram-like vector. The result of this process is then a synthesized spectrogram, that aims to contain features of the two input sounds.

As the latter methodology corresponds the most with the initial idea, to synthesize audio based on the characteristics of two instruments (e.g. guitar and synthesizer), this method was chosen to be implemented to carry out experiments on the creation of novel sounds.

To explain this method in more detail, the encoded vectors of two audio samples serve as basis. Then each of this vector is taken, to interpolate a value that lies on a linear line between the value at a given index in one vector with the corresponding value of the same index in the second vector. To mention at this point, those vectors are of the same length. The result then is a new array containing the interpolated values of those two vectors. This procedure gets repeated for each encoded output of one sample with the corresponding output of the second sample. Knowing the fact, that those encoded values represent the most significant features and probably the characteristics of the note, the result can be seen as a combination of those characteristics. Decoding those, will then end up in having a spectrogram, containing the characteristics of both instruments. How this procedure and its results look like, with the actual experiments, gets shown later on in this work.

3.6 Post Processing

It has been discussed, that regarding neural audio synthesis, the audio data can appear in different shapes. As there exist approaches, that focus on time-domain signal like the WaveNet-style autoencoder from *Engel et al.*, there also exist those who operate on spectrograms using convolutional networks. As mentioned before, this work emphasizes on the use of a convolutional autoencoder, that takes spectrograms or spectral data as input. In the case of this autoencoder, the output is of the same shape as the input and therefore also a spectrogram. To generate again a playabele or listenable audio, this one has to get converted back into time domain. There exist many different methods, to achieve this, while this also depends heavily on the data that is present. As stated in section 3.3, when spectrograms are calculated via the STFT, the output vectors are complex valued. To be mentioned, that without modification or further utilization, via the inverse STFT (ISTFT), this result vector can get converted back into time domain without loss. For achieving this task, it is necessary that the magnitude but also the phase information has to be present, combined in a complex number. As this autoencoder just operates on the magnitude data of spectrograms, there is no phase information present in the output of the network. At this point, multiple ideas can be applied, depending of what is done throughout the process. This means, that when there is no modification in latent space, i.e. value interpolation, the original phase information can be reused, to in order apply the ISTFT. As mentioned previously the phase information gets preserved for exactly this case.

In the second case, where modification steps are performed, like here the interpolating of two sounds' embeddings, there is no phase information present, that can be used. To overcome this issue, there exist techniques that can approximate the phase information. One of those techniques, which is also probably one of the most prominent ones, is the Griffin-Lim [8] algorithm that tries to estimate the time-domain signal based on just the magnitude data. With this algorithm the phase gets randomly initialized, and with alternating forward- and inverse STFT, estimated. For the calculations of the audio signal, again the python library *librosa* [27] has been used, as it provides the inverse STFT but also implements the Griffin-Lim algorithm. According to the documentation of *librosa*, a so called “fast” Griffin-Lim algorithm is applied, which got developed by *Perraudin et al.* [23]. The difference here is, that this one utilizes a additional momentum parameter which helps to accelerate the convergence of the estimation.

Some further note here, as when the pre-processing steps (see section 3.3 have been examined, that some certain parameters have to be taken care of. These are especially, the `n_fft`, `hop_length` but also the `window`. It has been mentioned, that e.g. the `hop_length` was chosen to be half of the `n_fft`, to ensure a 50% overlap of the STFT frames. Furthermore the “Hann”-window was chosen, to be multiplied with the signal frames, to avoid discontinuities. Again, those also appear in the inverse STFT as well as Griffin-Lim. In order to obtain the best result, it is important, to apply the same values with those parameters. As the inverse Fourer transform gets applied to each vector in the spectrogram, this results in single time-domain frames that have the length of `n_fft` (original length). With the knowledge of the window and the overlap, those single frames, get “overlap-added”, which results again in the full length audio sample.

Before applying the the inverse calculations on the auteoncoders output, it has to be considered, that the autoencoder works on the db-scaled magnitude. As a consequence the output therefore is also db-scaled. To apply the inverse calculations, the output gets back into energy. For this case a reference value, has been preserved from the preprocessing stage, to in order obtain the (almost) same scaling again in the output signal. For the experimental part some more steps also got applied, like scaling the energy according to the average energy present in the original spectrograms. This step should also correct and improve the resulting sounds. More on that in chapter Experiments (4), but also later on when examining and discussing the results.

3.7 Dataset

As machine-learning models including neural networks, have to get trained, in order to deliver accurate results, data is needed on which it should get trained on. To get compelling results, it is not only important to have an appropriate model configuration or pre-processing chain. The choice of an appropriate dataset therefore is also of high significance. As seen in related works, datasets can either be self-generated or taken from a public available data-source. As in the case of this work, the model operates on audio data, a dataset of musical notes is desirable. Generating sufficient data, on ones own, is a task that would take a significant amount of time. Not only as this dataset preferably should contain a large amount of audio samples, those also should be highly divers such as different instrument sources or different pitches. Not only is it an advantage for the training of the neural network to have lot of samples and diversity, as it helps to improve the learning process and generalization within the neural networks. Moreover it is also advantageous for this kind of work, as when having many different instrument sources and available notes, more interesting combinations with regard to audio synthesis can be made.

3.7.1 NSynth Dataset

Exactly for this kind of approach a big dataset consisting of musical notes called "NSynth", has been made publicly available by *Engel et al.*. [6] This dataset consists of a total of 306 043 musical notes that have a unique pitch, timbre but also envelope and has been created for the idea of neural audio synthesis. This amount of musical notes incorporate monophonic audio snippets, sampled at a rate of 16kHz, of 1006 different instruments. Every note is of a specific pitch, ranging over every note of a standard midi piano (21-108). This results in having 88 different pitched notes, in the best case, as not every instrument is capable of producing all different pitches. The average amount of pitches is therefore, according to the scientific publication, 65.4 per instrument. With more detail, each audio sample belongs to a certain instrument family which could for example a keyboard, guitar, organ, bass, brass and so on. Further on they can be distinguished of their source, being either produced acoustically, electronically or synthetically. All these specifications make this dataset, highly attractive for this kind of work, as well is it publicly available[2] for free and already successfully used for neural audio synthesis. Regarding the use for machine learning tasks, on their website, they provide the dataset already, splited up into a training, validation and test set which do not overlap at all. To specify, the training set consists of 289 205, the validation set of 12 678 and the test set of 4 096 examples. For this purpose, it has been chosen, to take these splits as they are, for the training, validation and finally testing stage.

[2] https://magenta.tensorflow.org/datasets/nsynth

Chapter 4

Experiment

This section describes the experiments conducted in order to be able, to answer the defined research questions. In the previous chapter (3) the general methodology used technologies got described, to gain an understanding of the implementation and its components. Based on this knowledge and implementation, some experiments got conducted, that provide interesting results and meaningful insights into this approach of neural audio synthesis. For the purpose of this thesis, some research questions have been defined, for which the following experiments, deliver the answers, which get discussed later on when showing the results.

Those research question are defined as the following:

- **Is it possible to create novel sounds based on the characteristics of two instruments, by using ML technologies such as convolutional neural networks?**
- How do the pre processing steps influence the quality of the training, but also the quality of the output?
- How does the configuration/composition of a neural network influence the quality of the output?
- What are the information that are learnt from the neural network?
- By using audio spectrograms, are they suited best to achieve this task?

In the last section, when describing the different technologies, it has been mentioned, that they can be parameterized in order to influence the outcome of this work. Furthermore some additional steps can be introduced that have a significant impact on the result. This section therefore describes, how the proposed methods were utilized, in order to answer the above mentioned questions. Furthermore those experiments, should deliver some interesting insights, on how different configurations influence the workflow but also the final result being a synthesized audio. Those experiments span almost every stage from the pre-processing until post-processing.

4.1 Implementation Environment

In the previous chapter it has already been mentioned, that this approach is developed in python using specific libraries. To shortly mention, for pre- and post-processing the python audio-library *librosa*[27] has been chosen, as it provides all necessary functionalities that are needed for the approach and experiment. For all steps regarding the neural network model such as configuration, training, inference etc., *PyTorch*[22] has been utilized. The project though has been implemented and applied on two different machines, depending on the task that has to be done. Generally speaking, the toolchain compromised of all stages, has been developed on a local machine running python, except the training itself.

Speaking of that, the training has been mainly performed on a remote "jupyter-notebook" that has access to high-performance GPU resources. Not at least, as in the case of training a convolutional neural network, this is a rather time and computational power-consuming task. Of course, this not only depends on the kind and complexity of the network, but also on the amount of data that is used for the training. Furthermore using GPU-acceleration means to significantly have more computation power and speed, as its applying parallelism. Using the local machine, just the CPU could be utilized for training, which would mean that training is done sequentially and thus significantly slower while the local machine has to be awake constantly. For the training on the remote instance, the pre-processing also gets done there as the data is directly loaded there. As just the training is performed remotely, all other steps, including the evaluation towards audio (re)synthesis having the trained model, gets again done locally. Not at least, as no time-consuming tasks have to be made, but also as its more convenient, as the remote service is not always accessible.

4.2 Training

The training, as mentioned previously, gets performed on a remote "jupyter notebook"-service with access to a GPU. As outlined in the previous chapter, for the whole experiments, the NSynth data set proposed by *Engel et al.*[6], gets used. As is well known this dataset is already split into a training, validation and test part. For the training on the remote notebook, the training and validation data set will therefore be used, which in order also get pre-processed there. As a side note, in the beginning of the project, the training was just held locally, with a small subset of the already small test set (mostly of one instrument). This was just done to make a low-level proof that the autoencoder model can produce meaningful results.

4.2.1 Training configuration

To take a closer look onto the training process, this one consists of several important stages and components. First of all the PyTorch-model, defined as a class, gets initialized. As an metric is needed, to measure the error of the output, the mean squared error (MSE) gets utilized. This error metric calculates the difference between all values of the desired and actual output, squares them, and takes the average over all. Furthermore to optimize the network, as explained before, the Adam optimizer gets applied, in which the (starting) learning rate, but also the weight decay gets defined. The right learning rate depends here heavily on the amount of training data but also complexity of model. In the case of this work, this means it is in the range between $1e-5$ to $1e-7$. In chapter 3 it got also mentioned, that a technique to minimize the the learning rate, during the training process gets utilized. This function called `torch.optim.lr_scheduler.ReduceLROnPlateau(...)` reduces the learning rate, by a given factor, if within a certain patience period (epochs) no optimization gets detected. This method, improves the training process as further convergence can be achieved. Of course for the training process, the pre-processed training and validation data set has to be loaded. To mentioned, the pre-processed data gets calculated and stored on disk, in advance, to not always have to run through it. for the training it therefore gets loaded and brought into the desired shape for the corresponding model. This shape gets varied throughout the experiments, to observe its impact on the training process but also quality of the output. As this also depends on the chosen network, this gets described later on in this chapter. For the sake of training, this gets done with the training dataset, as well as for the validation dataset, as after each epoch the model gets evaluated on a held out validation set, to check the error on never seen data. The data in the right shape, has to be converted to a tensor and in further notice, to a (custom) dataset object. Finally a so called "DataLoader" has to be initialized, either for the training but also for the validation. In this DataLoader the batch size can be specified, whitch is an important parameter regarding the training. Throughout a few batch sizes, have been tried out, whereas in the end, the prefered batch size was 32. This means that the input data, gets portioned in equally sized chunks of 32 tensors, that get fed into the network at once. Further on it can be set, that after each training epoch, the dataset gets shuffled. Setting this to true, the samples in the batches are in a different order and constellation. Otherwise, the batches consist always of the same data in the same order. Throughout the experiments, this setting was proven to be advantageous regarding the convergence of the model, as the error could be more minimized (see chapter 5).

4.2.2 Training Execution

Having the configured dataloaders, which are also iterables, it is possible to iterate over the batches of the dataset. Before the training begins, a number of epochs has to be set, which defines how often the training should be performed on the whole dataset. In advance it cannot be said, how many epochs are needed, therefore a preferably big number gets chosen e.g. 1000. To clarify, the training never runs until the final epochs, as it get stopped at a point, where the result is sufficient (more on that shortly). In each iteration, the corresponding batch of data will get input into the network with a forward pass which calculates an output. This output gets compared to the desired value, which is in this case the same as the input, and further on the error gets calculated. Further on the gradients get computed and optimization via parameter update is done. The loss value gets added up, and after all batches are done, the average loss gets computed, to show the progress.

After all train batches have ran through and optimization is done, the model gets validated using the held out validation set. This one is also equally batched, and runs through the same process, except there is no optimization. Here there error is just calculated to see, how the model performs on data that is not used for training and therfore never seen before by the network. This technique helps to prevent to overfit the training data, which would be signalized through an increasing validation error despite training error gets smaller. Having the validation error after each epoch, this value gets used for the learning rate scheduler to decrease the learning rate if the error does not decrease in a certain period.

To ensure to have a sufficient trained network, the error scores get observed periodically. If the validation score does not improve more, i.e. the network convergence stagnates, and is sufficient, the training gets stopped. Important to know that after each training iteration, which is also called epoch, the state of the model gets saved, for further use. As also the scores for each model state is known, it is therefore possible to determine the best model, having the lowest MSE-score on the validation data. This one then gets further tested and analyzed towards the applicability for audio (re)synthesis. Those further steps, which do not involve training, as well as the pre-processing regarding the evaluation data, get performed locally.

4.3 Initial Experiments

In the beginning of this project, initial experiments, were done in order to make a very basic proof of concept implementation. Like mentioned before, those implementations, where entirely held on the local machine, not at least as there was no access to GPU-accelerated training. This was also possible, due to just taking a small subset of the test dataset. Of this test dataset, samples of one instrument (*keyboard_synthetic*) were taken out, and considered for test-wise training.

4.3.1 Whole Spectrograms as Input

Pre-processing

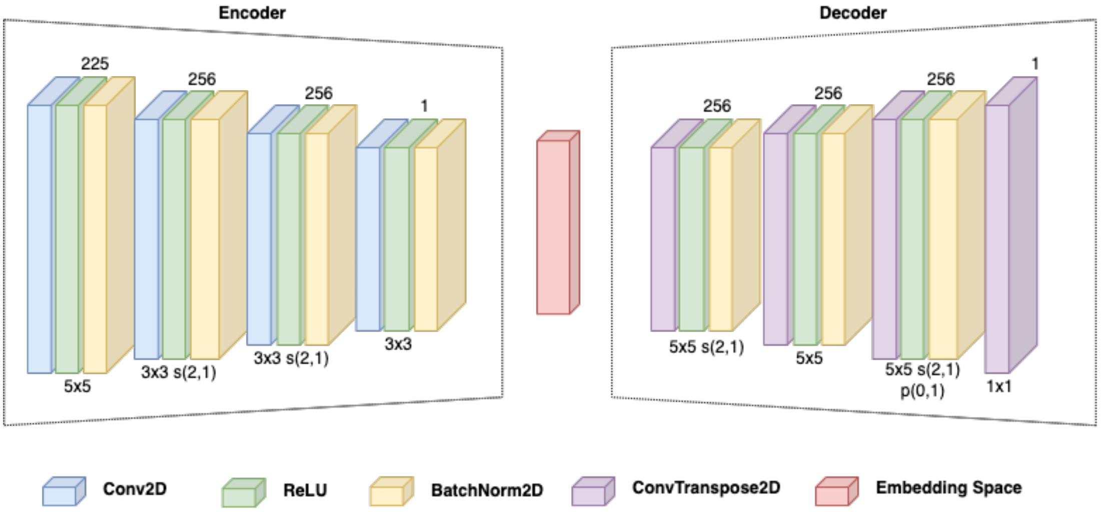
For the first experiments, all the samples of the subset, have been converted to log-magnitude spectrograms as a whole. As already known, those samples have all the same length of 4 seconds. Some samples are padded with zeros, as those do not contain audio data over the full length. As parameters for the STFT `n_fft` of 512 and `hop_length` of 256 got chosen. This configuration results in spectrograms having 257 frequency bins with a resolution of 31,25 Hz. Regarding the time-resolution it can be said, that each frequency vector represents 16 milliseconds of the original signal with a 50% overlap. This procedure resulted in spectrograms with a dimension of 257x250.

Model and training

As the spectrograms can be seen as grey-scale images, and thus 2D-Convolutions got applied, a third dimension got added resulting in 257x250x1 spectrograms. Finally to form a dataset ready for training, all the spectrograms of the mentioned subset got concatenated to a 4D-array, which was converted into a tensor and subsequently into a dataset. The following training, was performed on 80% of this subset, whereas the other 20% were used for evaluation. Regarding the duration of the training, this was held for a short time (20 epochs).

To also mention the model configuration, this consists of 4 convolutional layers in the encoder including ReLU activation and batch normalization. The decoder part has therefore also 4 layers, which contain respective convolutional-transpose layers with also ReLU and batch normalization except the very last layer. Another important detail is, because of the shape of the input, that the number of input channels has to be 1. Throughout the network, this parameter gets varied, whereas in the first layer an expansion is being made to 225, subsequently in the second to 256 output channels. To the innermost layer it gets again reduced to 100. In the decoder part, again an expansion is being made, whereas at the end it gets reduced to 1 channel again, in order to match the input size. The next graphic (fig 4.1) shows the autoencoder that is used for the initial experiments. To be mentioned the kernel-size, strides (s) and padding (p) here is already the configuration for the succeeding experiment. The channels, the general structure as well as amount layers is the same for the next experiment.

Figure 4.1: Initial 2D-convolutional autoencoder



Testing/Evaluation

Regarding the evaluation of this first model, it has been tested out on the remaining 20% of the small subset. These first experiments, consisted just of evaluating the outcome of the decoder, by passing single spectrograms through the network. By this, the ability of the network to recreate audio spectrograms got proven, which further on serves as a base for the next experiments. Regarding the final outcoming sounds, the original preserved phase information was reused to recreate audios.

4.3.2 Spectrograms of signalframes as Input

In the previous initial experiment, the samples from the NSynth dataset have been taken as a whole for the experiment. It has been mentioned, that all samples are 4 seconds long, but some contain padding in order to come to the 4 seconds. As those zero-paddings are then also part of the trained that, this could affect the behaviour but also the outcome of the model. If those zero-paddings would be left out, this leads to unequal long samples, which brings the problem with it, that they cannot be used for the model. Not used at all, as the network has a fixed size of neurons at the input, which means that the input has to be in a fixed shape. Furthermore this also means, that having a fixed size of 4 seconds, the input always has to be of 4 seconds, which is not desirable. Not at least, if the system should be used in real-time applications for audio synthesis, one cannot wait to have 4 seconds of a signal, to perform audio synthesis with it. Therefore it would be desirable to perform audio synthesis on smaller “frames or chunks” of an audio signal, respective spectrogram.

Pre-processing

This leads, to the idea to take chunks or frames of audio data that get transformed into distinct spectrograms of same size. For a start the length of those frames, gets set to 500 ms. Furthermore it got chosen, that those frames are not consecutive, but have an overlap of 50%. This should preserve the continuity of the signal [1]. Additionally those frames get multiplied with a window function, like it is used in the STFT. Similar to the window function used in the STFT, here a Hann-window is used. Again as those trimmed signals all are differently long, they have to get padded to a multiple of the frame size respective hop-length. This ensures to have equally long chunks of the signal. As of the windowed frames, the first and last frame don’t have overlapping parts at the beginning respective end. To overcome this issue, additional zeros get added there to form one frame on each side, to get there also an overlap. In combination, having the framed and windowed signal chunks, by overlapping each again with 50% and adding the values, this would yield the original signal again. This gets especially helpful when reconstructing the final signal in the end.

Having those framed and windowed signal chunks, the STFT gets applied on those, having the same configuration as in the first experiment. This then leads to have multiple spectrograms for the length of 500 ms with again a frequency resolution of 31,25 Hz. Again for the training and testing, additional to the log-mag data, the phase information, reference value and name of the sample including a number to identify the frame.

Model and training

The configuration of the model is rather similar to the one used in the first setting. It has the same amount of layers on each side, despite different strides, but also different kernel-sizes got applied. Again this one has been trained on 80% of the *keyboard_synthetic* test dataset samples. Of course the significant difference here is, that now the input data are not whole spectrograms but overlapping frames. This also means, that the amount of data has increased. When collecting the spectrograms for the dataset, the names get shuffled, in order to not have the same order. In contrast, the single frames, don’t get shuffled as well not during training. Again the training has been performed over 20 epochs on the local machine.

[1] not sure if right

Testing/Evaluation

For the purpose of evaluating the test score this has been done with the remaining 20% of the samples. To evaluate the ability of the autoencoder to reconstruct spectrograms, the whole samples are used including those from the training. Here the spectrograms are provided in the order as they appear. Having all reconstructed spectrograms, the inverse STFT got applied with the preserved phase information. Resulting in the frames corresponding to every single input note, those got overlapped and added (in their right order). By this procedure the signal with the original length could be obtained and further on evaluated auditorily. For results see chapter 5.

A step for interpolating two different sources has not been investigated here, up to these experiments. Also the embedded space also has not been evaluated so far but will be subject of further experiments.

4.4 Experiments single frequency vectors

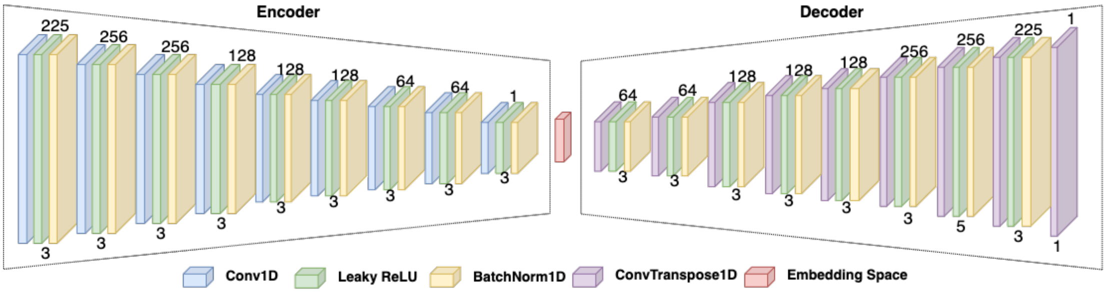
The above mentioned experiments, were a prove regarding the ability of convolutional autoencoders to recreate audio spectrograms. From this point on the experiments were done using the whole training dataset and also include audio synthesis. In the previous experiment the model is trained on frames of audio data which are 500ms long. Having the idea to synthesise audio with real-time input, this would mean that always a signal frame of 500ms has to be present, in order to have an input for the model. Therefore it is desirable to have an input that is as short as possible. In important note at this point is therefore, that spectrograms consist of frequency x time data. Each vector of the spectrogram along the time axis therefore represents a short frame of time.

As mentioned before, when pre-processing is done on 16kHz sampled data with an n_fft of 512, respective hop-length of 256, a time resolution of 16 ms is therefore present. Therefore the idea is to take those single frequency vectors, as input for the model. The shape therefore will be frequency x channels which in the case of the previous parameters is 256 x 1. With this idea, also the silence at the end of the signals can be omitted as just the frequency domain gets used for the models input. This therefore enables to take samples of different length in time. Throughout the experiment the value for the n_fft got increased to 1024, as an increase of the models performance could be achieved (more on that in chapter 5). Increasing this parameter therefore means to decrease the time resolution, but increase the amount of frequency bins and thus having a better frequency resolution. Resulting in a number of 513 frequency bins (15,625 Hz/bin) and time resolution of 32ms per vector.

Neural network

As a consequence this means, that also a different model has to be used, as 2D-convolutions are no more suited. Therefore a model has been designed that uses 1D-convolutions. 1D-convolutions are performing the same calculations, with the difference of having just a 1-dimensional kernel. This one dimensional kernel therefore operates on the frequency axis and tries to extract important features. Equally to the 2D-convolutions they also apply the principle of channels. Regarding the channels, those also get expanded but then subsequently until the innermost layer reduced to 1 in order to form a single dimensional vector. Until the end again those channels get expanded but reduced again to 1 at the end to have the same shape as the input vector. In the following graphic (figure 4.2), the structure and configuration of this autoencoder is depicted.

Figure 4.2: Deep 1D-convolutional autoencoder



Here it can be seen, that in contrast to the one used in the initial experiment (see section 4.3 figure 4.1), leaky ReLU is used instead of normal ReLU. The use of the Leaky ReLU was also documented, in the work of *Engel et al.* [6] where it was used as activation function in the convolutional autoencoder. In the beginning for this experiment, normal ReLUs were used, but later on leaky ReLUs got applied and proven beneficial regarding model performance. This model also does not include, strides in the convolutional layers, which means, that the input does not get significantly downsampled towards the “bottleneck”. One more notable difference is the depth of the network, as it has on each side 9 layers, resulting in a 18 layer network. Regarding the sublayers, also batch normalization gets applied, whereas it is also 1-dimensional, just as the convolutional layer. With all this configuration, the size of the embedding therefore would be 495x1x1. Throughout the experiment with 1D-convolutional networks, of course different configurations have been made, but this one worked out best. The results using this network can be seen later on in chapter 5.

Training

The training process for this experiment, differs in some major points from the previous ones. The main difference lies in the shape of the data. This can be concluded as here just the single frequency vectors, are used for this network and therefore the network itself has 1D-Convolutions as discussed before. Furthermore, this experiment was originally trained locally with the small subset like before, but later on access to a GPU-accelerated instance was made available. This GPU-instance, as discussed at the beginning of this chapter, enables to train complex ML-models with a large amount of data, over a long time. With this possibility, the training dataset can be utilized as this consists of several thousands of audio samples and would be too big to train locally. This does not mean, that the whole dataset was used all the time for training on this instance. With the progress of the project, several trainings with different amounts of data, could be made. Those are to be mentioned:

- **Single Instrument**

- **Multiple Instruments (>=2)**
- **All Instruments**

Regarding the training performance, some interesting findings and observations could be made which get mentioned when showing the results.

As here single frequency vectors are used, the process of creating a tensor and subsequently the dataset for the dataloader, is different. Here all spectrograms of the pre-defined and pre-processed dataset, were used, and the single frequency vectors get concatenated to form one big 3D array in the shape of amount x channels x frequency. To note the amount of channels here is also one in the input data. When experimenting with the training and its performance, also different strategies regarding shuffling got used. First on just the names of all samples got shuffled resulting in the instruments being mixed, but the frequency vectors were in the same order as in the spectrogram. This stays the same as after each epoch the data doesn't get shuffled. One more strategy that was found more successful, was also to shuffle the whole generated array, resulting in the frequency vectors of all instruments are mixed. Furthermore shuffling was also done after each epoch. The same strategies got also applied to the validation dataset, as from this point on also the validation dataset was considered. Not at least as more computational power was present from this point on. The most time, a batch-size of 32 was being used.

Like discussed in chapter 3, for the training also a validation step is needed. This validation was made using the validation dataset, which does not intersect with the training dataset. The validation process takes place in each epoch, after the training has been performed. This includes calculating of the train loss, back-propagating the error and optimizing the network. With this step, the network gets validated on data that has not been seen before, to prevent the network from overfitting. Introducing the validation step, also the scheduler to adapt the learning rate during the training gets applied here. Taking into account, that the network is rather complex and a huge amount of data has been used, the learning rate also has to be chosen adequately. Throughout this experiments, a small learning rate of $1e-7$ was chosen, as this led to a more stable training and better convergence.

Testing

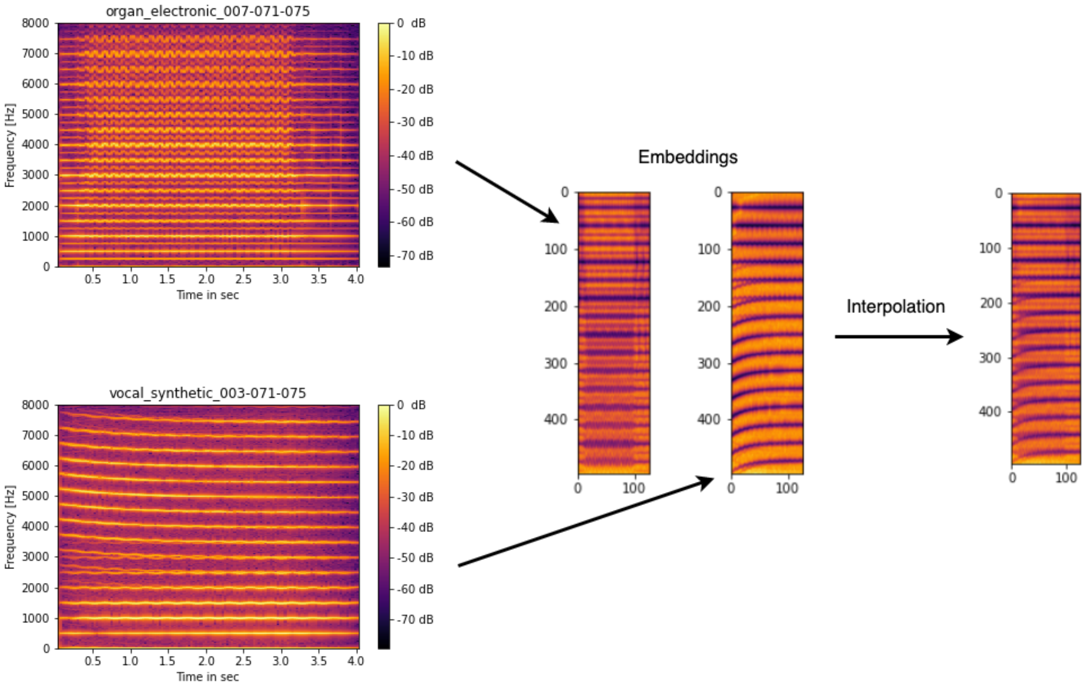
As described above, after each training, the current model state gets saved, those can be used for the testing and evaluation regarding synthesis. As having numerous states that got saved, the one with the lowest error score on the validation set gets taken for further steps. For the steps around testing and evaluation, the held-out test dataset gets used. This stage has been configured, to either take the whole test-dataset, a specific pitch, or a specific instrument source. By this technique its made possible to get the scores and therefore performance for certain instruments or pitch.

4.4.1 Experiments for Synthesis

Having this kind of network that got trained on the training dataset, to reconstruct single frequency vectors, some experiments have been conducted for generating audio. As during the training the network learned how to reconstruct frequency vectors, one experiment is to examine the quality of the output for a single non-modified audio. Those are similar to the ones conducted, in first experiments above. Also by reconstructing just single notes those, the phase information was present, which enabled to apply the ISTFT. For a comparison also the Griffin-Lim algorithm got utilized.

As the main objective of this work is to examine the capability of creating novel sounds, from this point on in the project the interpolation step got introduced. With the interpolation step the encoded features in the embedded space of two instruments, get taken and value-wise interpolated. As it is known by this point, the encoder part of this network takes as input a vector of 513x1 and creates a lower-dimensional representation of it with the size of 495x1. Those representations, can be seen as the essential features, and got considered for the interpolation task. For this task, the frequency vectors of two instruments of probably the same pitch get passed through the network. Here the data does not get shuffled, as its important to produce the values in the same order as they come from the spectrograms. The output of the encoder for each instrument, gets concatenated to a 2D-array 495 x N where N is the number of encoded frequency vectors. To get a better idea of this concept, the next graphic (figure 4.3) shows two spectrograms with the corresponding accumulated output of the encoder.

Figure 4.3: Input spectrograms with embeddings and interpolated embedding



Having those two representations, interpolation gets performed. Like discussed in chapter 3, along the x-axis each output vector of one sample gets interpolated with the corresponding output vector of the second sample. This process is also shown in figure 4.3 where the result of the interpolation process is shown. This process got applied equally for each subsequent experiments as the encoder output is always of the same shape, but more on that later on. This interpolated vector gets subsequently passed through the decoer part which forms again frequency vectors of 513x1 which get accumulated to final spectrogram. This spectrogram then depicts the result of audio synthesis combining features of two different instruments.

Further on this spectrogram gets converted back to audio domain. As in this case no phase information is present, the Griffin-Lim algorithm [8] for phase estimation gets applied. By listening to the final sound, it should be possible to hear the characteristics of both instruments combined in this sound.

4.5 Experiments with slices of spectrograms

Based on the results and insights that could be gained with the previous experiments (see chapter 5) some more experiments were made. Not at least, to examine how different representations of the input data and as a consequence different model configurations influence the task of neural audio synthesis. For this case the following experiments in contrast work with 2D-representations as input. As 2D data there already have been the the initial experiments with whole spectrograms but also spectrograms based on frames of the audio signal. As a difference for the following experiments, the spectrograms have been taken, but sliced into overlapping chunks. The spectrograms taken in this experiment, got generated with the same configurations, as in the previous experiment (`n_fft=1024`, `hop_length=512`). The idea to use chunks of spectrograms as input, arose to improve the models performance with regard to synthesis and recreation of spectrograms. As by using single frequency vectors, there is just the freuqency information present and no information about its change over time. As a theory to also incorporate the temporal axis as input, important information about the temporal frequency changes could be captured. Those frequency changes could deliver more important characteristics of the samples that could be extracted. As not the whole spectrograms should be used as input, chunks of them are favorable, to keep the input as small as possible. Therefore the choice has been made to take three consecutive frequency vectors to form a frame with the shape of (513x3). Throughout the experiments it has been tried out to either take the frames subsequently or overlapping, leading to the preference in taking the latter. The overlap has chosen to be always 2 vectors from the preceding frame.

For demonstration, let there be a spectrogrogram $spec = [v_0, v_1, v_2, v_3, v_4, v_5, ...v_n]$ of length n where v_i is the frequency vector at index i. This results in an array of $frames = [[v_1, v_2, v_3], [v_2, v_3, v_4], [v_3, v_4, v_5]...[v_{n-3}, v_{n-2}, v_{n-1}], [v_{n-2}, v_{n-1}, v_n]]$. The idea behind this overlap is to capture every change or time-pattern in the source spectrogram. This technique also has the advantage that all the silence of the source spectrograms can be cut away, allowing to use differently long spectrograms as input. A positive side-effect here is also, to gain more amount of data, on which the model can be trained on. As the models output are also overlapping frames, to reconstruct the final spectrogram and audio, this has also to be considered, but more on that later on. Again when creating the tensor and dataset, the spectrograms get taken in a random order. At first the resulting frames of the training set, don't get shuffled. Later on during the training, the frames in the batches, which again are of size 32, get shuffled like in previous examples. The same happens to the validation dataset, while it also gets shuffled initially but also after each epoch. An important note is that for the training and validation the whole dataset got used all the time.

Neural Network

As again input data in a 2D-shape was used for these experiments, the model again has to be adapted. Instead of 1D-convolutions and 1D-batch-normalization again, 2D-convolutions and 2D-batch-normalization has been used. Also in contrast to the previous network, a normal ReLU activation has been used. These experiments incorporate different model configurations whereas a special focus has been given to the amount of striding and thus input compression. Here it should evaluated, how more compression and thus a smaller latent space, influences the quality of the decoder output and further on of the generated audio. Both in terms of single note reconstruction and interpolation based synthesis. The following figure shows, the basic network structure for this kind of experiments.

Analysis of the embedded space

4.6 Experiments with mel-scale

4.7 Measurement of results

Chapter 5

Results

Chapter 6

Discussion/Evaluation

Chapter 7

Conclusion

Chapter 8

Future Work

Appendix A

Technical Details

Appendix B

Supplementary Materials

List of supplementary data submitted to the degree-granting institution for archival storage (in ZIP format).

B.1 PDF Files

Path: /

thesis.pdf	Master/Bachelor thesis (complete document)

B.2 Media Files

Path: /media

*.ai, *.pdf	Adobe Illustrator files
*.jpg, *.png	raster images
*.mp3	audio files
*.mp4	video files

B.3 Online Sources (PDF Captures)

Path: /online-sources

Reliquienschrein-Wikipedia.pdf	**WikiReliquienschrein2022**

Appendix C

Questionnaire

Appendix D

LaTeX Source Code

References

Literature

[1] Keunwoo Choi et al. *A Tutorial on Deep Learning for Music Information Retrieval.* 2018. arXiv: 1709.04396 [cs.CV] (cit. on p. 15).

[2] Joseph Colonel, Christopher Curro, and Sam Keene. *Autoencoding Neural Networks as Musical Audio Synthesizers.* 2018. eprint: 2004.13172 (eess.AS) (cit. on pp. 4, 5, 23).

[3] Joseph T Colonel and Sam Keene. "Conditioning Autoencoder Latent Spaces for Real-Time Timbre Interpolation and Synthesis". In: *2020 International Joint Conference on Neural Networks (IJCNN).* 2020, pp. 1–7. DOI: 10.1109/IJCNN48605.2020.9207666 (cit. on pp. 4, 5, 23).

[4] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning.* 2018. arXiv: 1603.07285 [stat.ML] (cit. on p. 21).

[5] Jesse H. Engel et al. "GANSynth: Adversarial Neural Audio Synthesis". *CoRR* abs/1902.08710 (2019). arXiv: 1902.08710. URL: http://arxiv.org/abs/1902.08710 (cit. on p. 3).

[6] Jesse H. Engel et al. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders". *CoRR* abs/1704.01279 (2017). arXiv: 1704.01279. URL: http://arxiv.org/abs/1704.01279 (cit. on pp. 2, 4, 13, 19, 23, 26, 28, 34).

[7] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. "Image Style Transfer Using Convolutional Neural Networks". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016, pp. 2414–2423. DOI: 10.1109/CVPR.2016.265 (cit. on pp. 7, 8, 11).

[8] D. Griffin and Jae Lim. "Signal estimation from modified short-time Fourier transform". *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32.2 (1984), pp. 236–243. DOI: 10.1109/TASSP.1984.1164317 (cit. on pp. 14, 25, 37).

[9] Eric Grinstein et al. "Audio Style Transfer". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2018, pp. 586–590. DOI: 10.1109/ICASSP.2018.8461711 (cit. on pp. 7, 10).

[10] Lamtharn Hantrakul et al. "Fast and Flexible Neural Audio Synthesis." In: *ISMIR.* 2019, pp. 524–530 (cit. on p. 3).

[11] Gerhard Heinzel, Albrecht Rüdiger, and Roland Schilling. “Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new at-top windows” (2002) (cit. on p. 16).

[12] G. E. Hinton and R. R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks”. *Science* 313.5786 (2006), pp. 504–507. DOI: 10.1126/science.1127647. eprint: https://www.science.org/doi/pdf/10.1126/science.1127647 (cit. on pp. 13, 20).

[13] Geoffrey E Hinton. “How neural networks learn from experience”. *Scientific American* 267.3 (1992), pp. 144–151 (cit. on p. 18).

[14] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* 2015. arXiv: 1502.03167 [cs.LG] (cit. on p. 19).

[15] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. “Perceptual losses for real-time style transfer and super-resolution”. In: *European conference on computer vision.* Springer. 2016, pp. 694–711 (cit. on p. 11).

[16] Michael I. Jordan and Christopher M. Bishop. “Neural Networks”. *ACM Comput. Surv.* 28.1 (Mar. 1996), pp. 73–75. DOI: 10.1145/234313.234348 (cit. on p. 18).

[17] joseph colonel joseph, christopher curro christopher, and sam keene sam. “improving neural net auto encoders for music synthesis”. *journal of the audio engineering society* (Oct. 2017) (cit. on pp. 4, 23).

[18] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. DOI: 10.48550/ARXIV.1412.6980 (cit. on p. 22).

[19] Xuehao Liu, Sarah Delany, and Susan Mckeever. “Sound Transformation: Applying Image Neural Style Transfer Networks to Audio Spectograms”. In: Aug. 2019, pp. 330–341. DOI: 10.1007/978-3-030-29891-3_29 (cit. on pp. 9, 11).

[20] Anastasia Natsiou, Luca Longo, and Sean O’Leary. *An investigation of the reconstruction capacity of stacked convolutional autoencoders for log-mel-spectrograms.* 2023. DOI: 10.48550/ARXIV.2301.07665 (cit. on p. 3).

[21] Keiron O’Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks.* 2015. arXiv: 1511.08458 [cs.NE] (cit. on p. 18).

[22] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* 2019. arXiv: 1912.01703 [cs.LG] (cit. on pp. 20, 28).

[23] Nathanaël Perraudin, Peter Balazs, and Peter L. Søndergaard. “A fast Griffin-Lim algorithm”. In: *2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics.* 2013, pp. 1–4. DOI: 10.1109/WASPAA.2013.6701851 (cit. on p. 25).

[24] Dhruv Ramani et al. “Autoencoder Based Architecture For Fast & Real Time Audio Style Transfer”. *CoRR* abs/1812.07159 (2018). arXiv: 1812.07159. URL: http://arxiv.org/abs/1812.07159 (cit. on pp. 8, 11, 13, 19).

[25] Fanny Roche et al. *Autoencoders for music sound modeling: a comparison of linear, shallow, deep, recurrent and variational models.* 2019. arXiv: 1806.04096 [eess.AS] (cit. on pp. 6, 23).

[26] Prateek Verma and Julius O Smith. “Neural style transfer for audio spectograms”. *arXiv preprint arXiv:1801.01589* (2018) (cit. on p. 8).

Software

[27] Brian McFee et al. *librosa/librosa: 0.9.1.* Version 0.9.1. Feb. 2022. DOI: 10.5281/zenodo.6097378 (cit. on pp. 15, 25, 28).

Online sources

[28] Dmitry Ulyanov and Vadim Lebedev. *Audio texture synthesis and style transfer.* 2016. URL: https://dmitryulyanov.github.io/audio-texture-synthesis-and-style-transfer (visited on 03/14/2023) (cit. on p. 10).

Check Final Print Size

— Check final print size! —

width = 100mm
height = 50mm

— Remove this page after printing! —