Max Graf - 190120296

Deep Learning for Audio and Music: Assignment

End-to-End Raw Audio Instrument Resynthesis

Introduction

This report describes the research for and implementation of an end-to-end raw audio instrument re-synthesis system on the PyTorch platform. The aim of the project is to accurately model mappings from one set of instrument's timbral characteristics to another. In this case the two instruments are a sampled Steinway piano and an electronic instrument that imitates the sound of the Yamaha DX7 synthesiser (an electronic piano sound). Audio files for both instruments were automatically synthesised from MIDI data and subsequently used to train two separate models. A convolutional autoencoder is used to find the mappings, which are cleared of noise in the second step, using a modified U-Net [1] structure.

Data

Arguably the most important aspect in creating mappings from one type of data to another is alignment. In the case of raw audio data, it is essential that events occur at the same point in time for all instruments. For that reason, a synthesised dataset was chosen for training the model. Additionally, for reasons of simplicity, the current system is limited to resynthesis of *one* instrument only. That means that input data consist of songs that contain only one instrument. One type of music that fits that requirement is found in classical piano pieces. A collection of MIDI files of transcribed piano concertos from numerous composers was found online[1]. A selection of those was compiled and used to synthesise training, validation and test data.

The Fluidsynth[2] platform was used to synthesise the MIDI data into uncompressed **.wav* files. It provides a flexible command line environment that allows for straightforward configuration of synthesis parameters using the *SoundFont*[3] format. Links to the resources used for the piano and e-piano sounds are listed in the Appendix section. The files were synthesised with a sampling rate of 22050Hz.

To accelerate the training process, Mel spectrograms stretching 128 Mel bands were precomputed and stored for each audio file using the *librosa* toolchain. To keep the training times limited, all pieces were trimmed to a length of 30 seconds. This resulted in 94 datapoints (each 30 seconds long), which amounts to a total length of 47 minutes.

Models

The resynthesis process is structured into two parts, utilising two different models. First, a mapping between the two instruments is approximated using a fully convolutional autoencoder that is trained on the aforementioned spectrogram data. To improve the interim results, i.e. the output of the autoencoder, a U-Net architecture is used to reduce the artifacts created by the mapping process.

Fully Convolutional Autoencoder

Autoencoders in both their discrete and continuous variants have been successfully applied to a broad range of audio- and video-related machine learning tasks in the past [2] [3]. Hsu et al. described a method for speech (re-) synthesis in 2017 [4]. In their system, a variational

[1] http://www.piano-midi.de/midi_files.htm, accessed on 18.04.2020
[2] https://github.com/FluidSynth/fluidsynth, accessed on 18.04.2020
[3] http://freepats.zenvoid.org/sf2/sfspec24.pdf, accessed on 18.04.2020

convolutional autoencoder was trained on spectrogram data to construct a latent representation of different speakers' characteristics. Systematic sampling from this latent space was shown to produce predictable results. Their system provided the baseline for the proposed architecture here, based on the assumption that re-designation of vocal characteristics is similar to re-designation of timbral features in instruments, in that both processes require some form of representational mappings between frequency components.

The autoencoder here consists of six convolutional layers in total. Three two-dimensional convolution layers are used to encode the data into the latent space, leading to a final dimensionality of 256 by 1 by 645 values. A fully connected layer was omitted because no operations (such as sampling to create new datapoints) are to be carried out with the latent representations. Finally, to reproduce spectrogram outputs, this structure is mirrored and the standard convolution layers are replaced with transposed layers.

One notable feature is the nature of the first and last convolution layers. As described in the paper by Hsu et al., these layers operate with filter sizes of $(SpectrogramHeight \times 1)$ and 64 channels. This effectively leads to a compression of the full frequency range of one spectrogram frame into a 1 x 64 vector, and forces the network to learn that the data is not translational invariant w.r.t. the frequency axis, but only the time axis. Subsequent convolution layers then use stride values greater than 1 in order to perform down- and up-sampling respectively.

The L1 loss function was used to calculate loss values from output and target tensors, as the number of outliers in the training data is unknown. This forfeits some accuracy, but provides more flexibility for the variation in the test data.

U-Net

Although the U-Net topology was originally designed as a method for segmentation in image data [1], it has been successfully adapted to perform other tasks, such as image restoration [5] and denoising [6], as well. The model included here was adapted from a pre-existing implementation[4] of a system built for denoising grayscale images.

The final loss value obtained while training the autoencoder was below 5% of the range of the training data. However, the resulting sound was still imperfect. This has two main reasons: Firstly, by converting the frequency information from the original short-term Fourier transform representation to Mel bands, some accuracy is lost due to the coarse grouping of frequency bands on the Mel scale. This error can be minimised however, by using a non-negative least squares approximation algorithm. Secondly, due to the polyphonic nature of the training data and the variations in timing of combined notes, mappings from one set of frequency components to another contain a certain amount of noise. However, "clean" versions of the desired results are available for training (the synthesised e-piano files). Therefore, after training the autoencoder, all items from the original training and validation sets were used to generate a second set of training/validation data for the U-Net.

The U-Net model was trained to minimise the mean squared error between the generated data and the original, synthesised e-piano files. This is helpful, because the model learns to minimise exactly those deviations that were caused by the lossy autoencoder.

[4] https://github.com/n0obcoder/UNet-based-Denoising-Autoencoder-In-PyTorch

The final system then consists of a sequential combination of autoencoder and U-Net model, producing a maximally clean output given the restrictions imposed by down-sampling (due to the use of Mel spectrograms and a lossy autoencoder). Finally, the *librosa* implementation of the Griffin-Lim algorithm [7] is used to convert the output spectrogram back to raw audio.

Evaluation

Autoencoder

The autoencoder model was trained for 600 epochs. It was found empirically that the training and validation losses do not decrease significantly after that period. Figure 1 shows the development of the loss values of training and validation sets averaged over the number of batches over time.

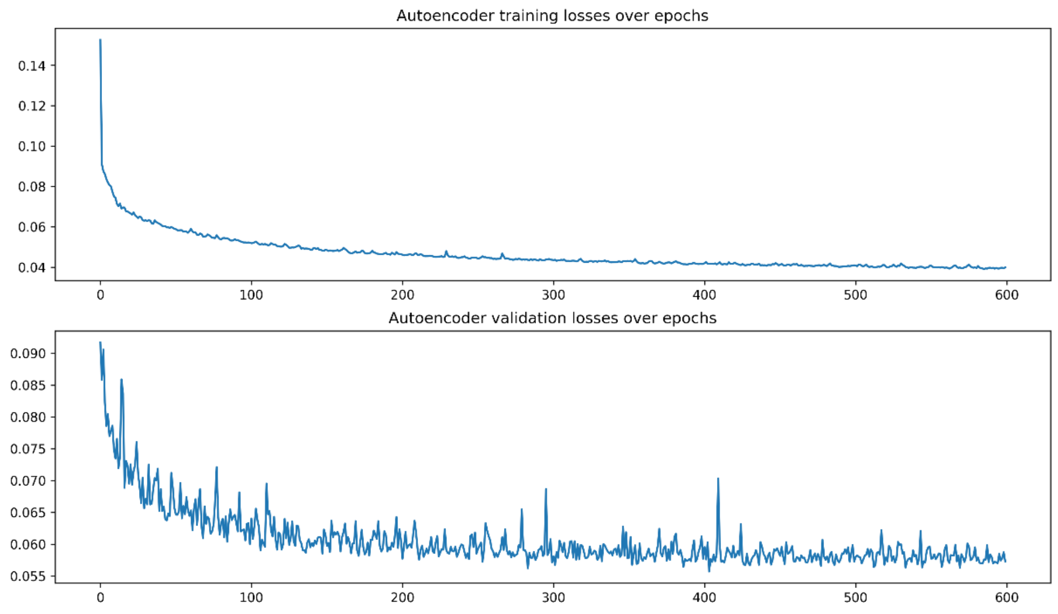

Figure 1: Development of loss values for training and validation sets

When comparing samples from the training set put through the model for evaluation at different points in training it becomes apparent that the model successfully learns to remap the frequency composites. Figures 2 and 3 demonstrate this visually. While the model produces considerable noise after 20 epochs of training, this error is minimised after 600 epochs. Figure 4 shows the spectrogram representation of the original piano piece. The comparison of the original piano piece with the synthesised version shows that while the timing of note onsets is identical in both versions, note durations and the combination of overtones is markedly different. The e-piano sound contains a larger amount of high-frequency energies and a distinct developmental pattern of sound over time. While the piano sounds have a relatively short attack time and a low sustain energy level, the e-piano sounds' onsets are slightly more gradual, and also tend to decay with a slower rate. This is reflected both visually and in the output audio files.

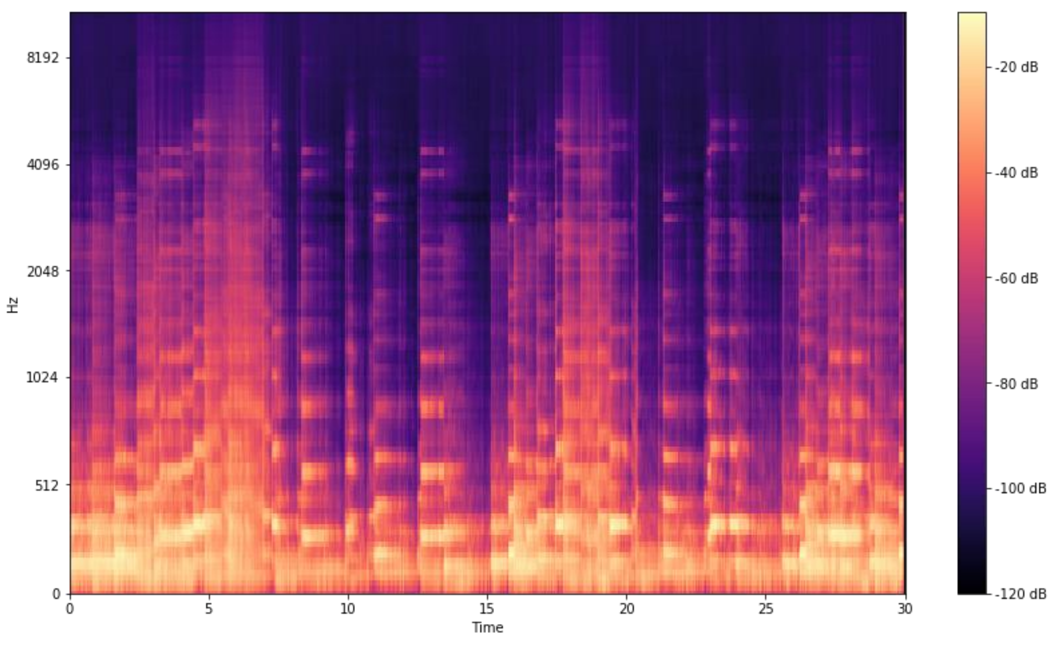

Figure 2: Autoencoder output for "chpn_ op7_ 1.wav" after 20 epochs of training

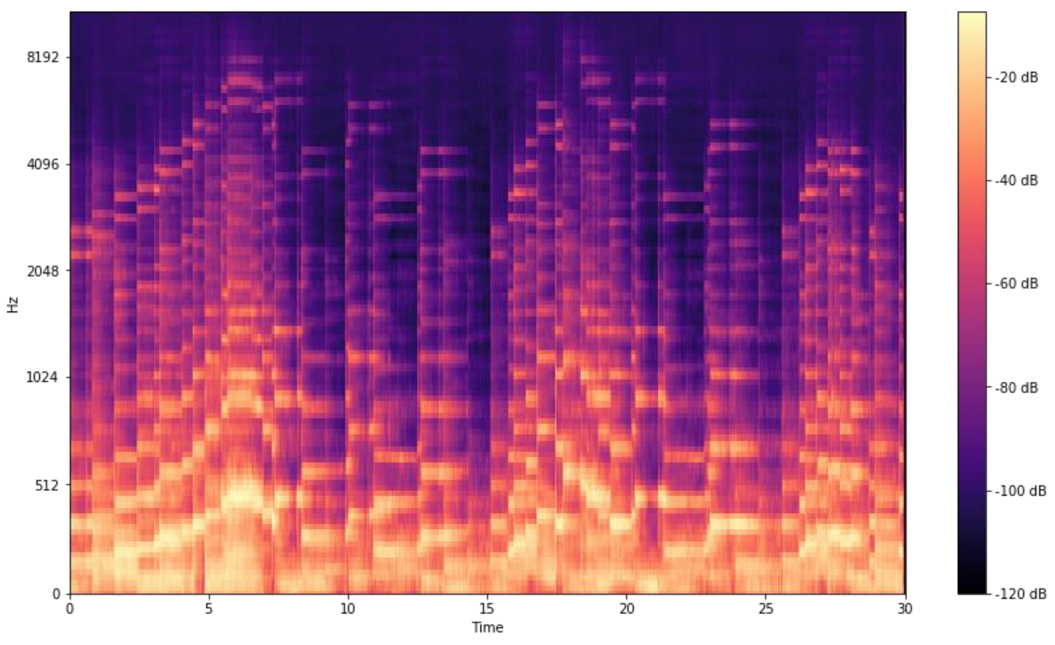

Figure 3: Autoencoder output "chpn_ op7_ 1.wav" after 600 epochs of training

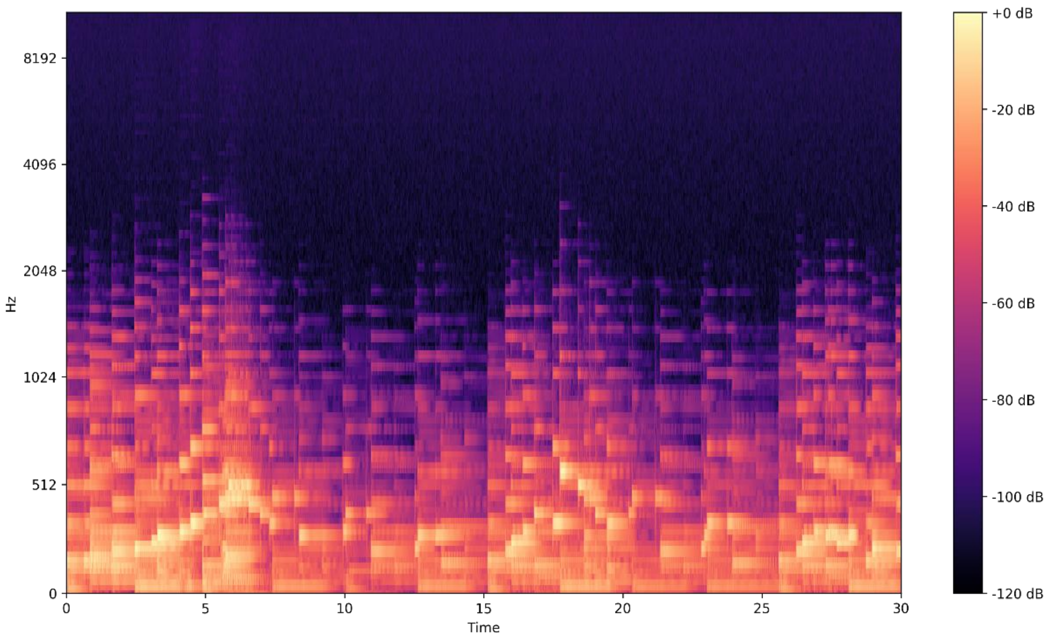

Figure 4: Original piano version of the sample "chpn_ op7_ 1.wav"

U-Net

The U-Net model was trained for 50 epochs. Figure 5 shows the development of the loss values for the training and validations datasets respectively. While the training loss values continually decrease, the validation set values persistently fluctuate. This fluctuation can be explained by the relatively limited amount of training and validation data items. It seems that a larger dataset might be required for training in order to persistently reduce the validation set loss. However, due to the small value of the final losses (expressed in the range [0...1]) this fluctuation is negligible here.

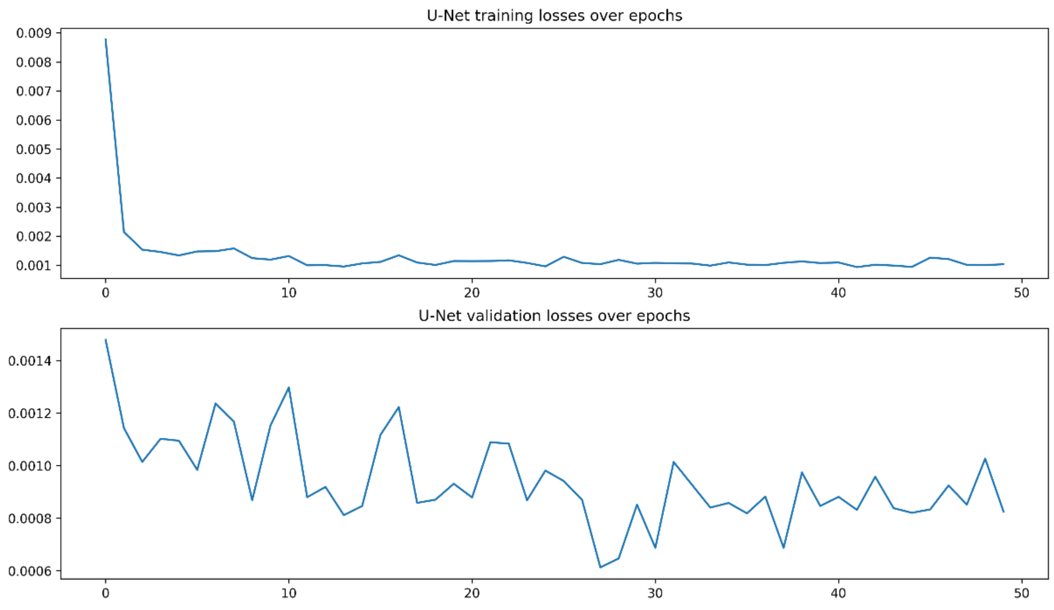

Figure 5: Development of loss values for training and validation sets

To demonstrate the role that the U-Net model plays in the system, two samples from the training data were taken before and after the postprocessing is applied. They are included in the *results* folder (*train_ sample_ after_ autoencoder.wav* and *train_ sample_final.wav*). Figures 6 and 7 show their respective spectrograms. While the noise reduction is visually most apparent in the middle and high frequency ranges, the main audible difference is located in the lower frequency bands. The application of the U-Net drastically reduces the "rumble" in the sound and produces cleaner final outputs.

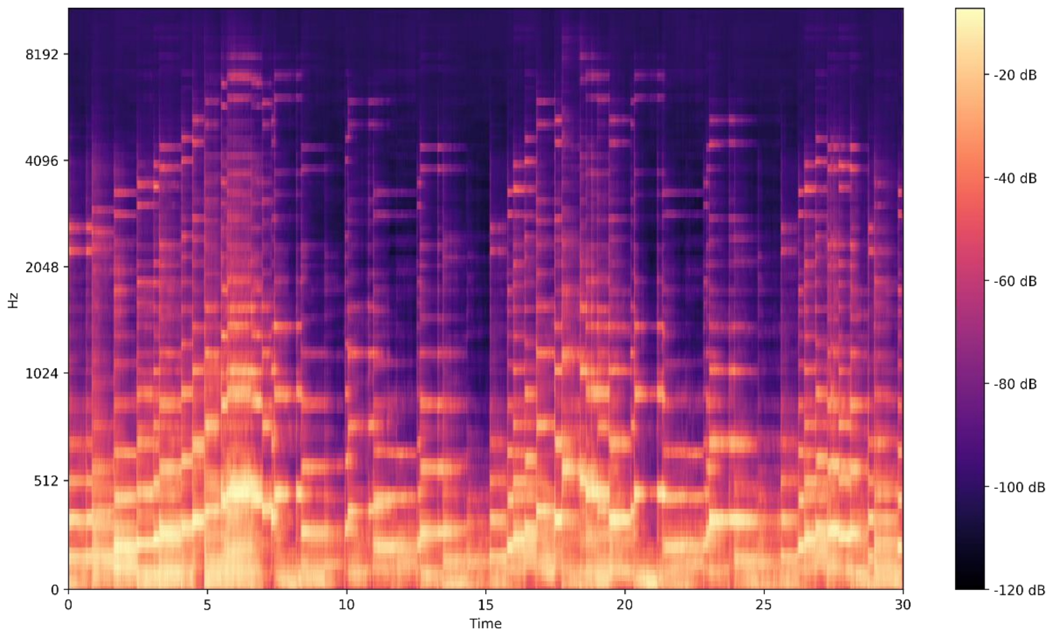

Figure 6: Spectrogram of the autoencoder output for "chpn_ op7_ 1.wav"

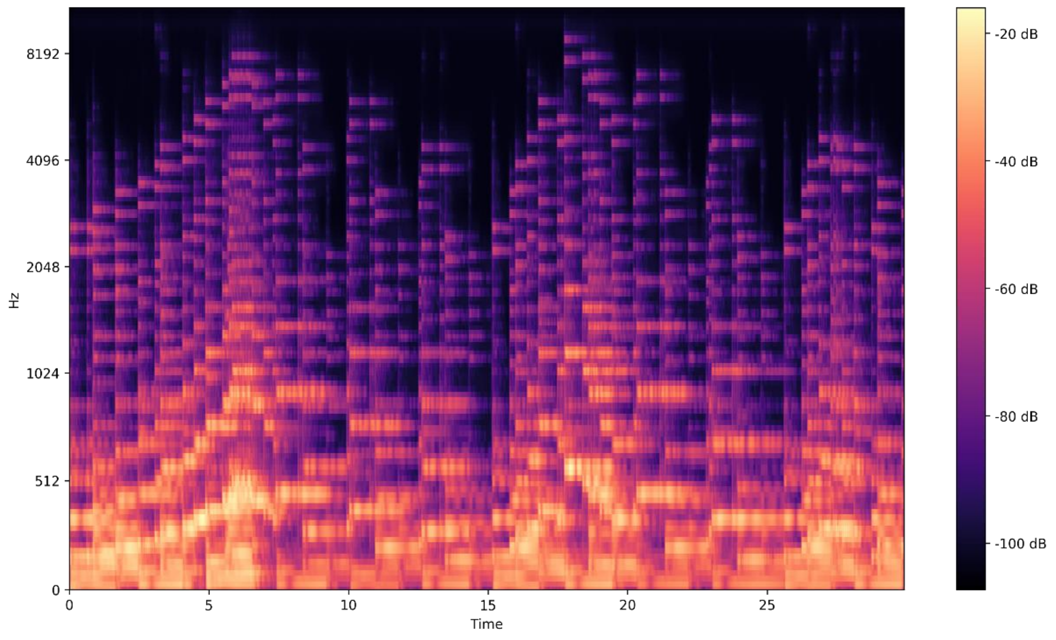

Figure 7: Final U-Net output when given the autoencoder output for "chpn_ op7_ 1.wav"

Application to audio file

In addition to testing the system on a separate set of synthesised test data, a real-world piano recording was chosen to assess the performance of the models (Laurens Goedhart performing

Debussy's *"Clair de lune"* in the *Suite Bergamasque*). The two main differences here are that the piece was recorded using a real microphone, and that the timbral characteristics of the piano it was performed on differ from those of the SoundFont used to train the models.

Figure 8 and 9 show the spectrogram of the original piano recording and the spectrogram of the output of the autoencoder respectively.

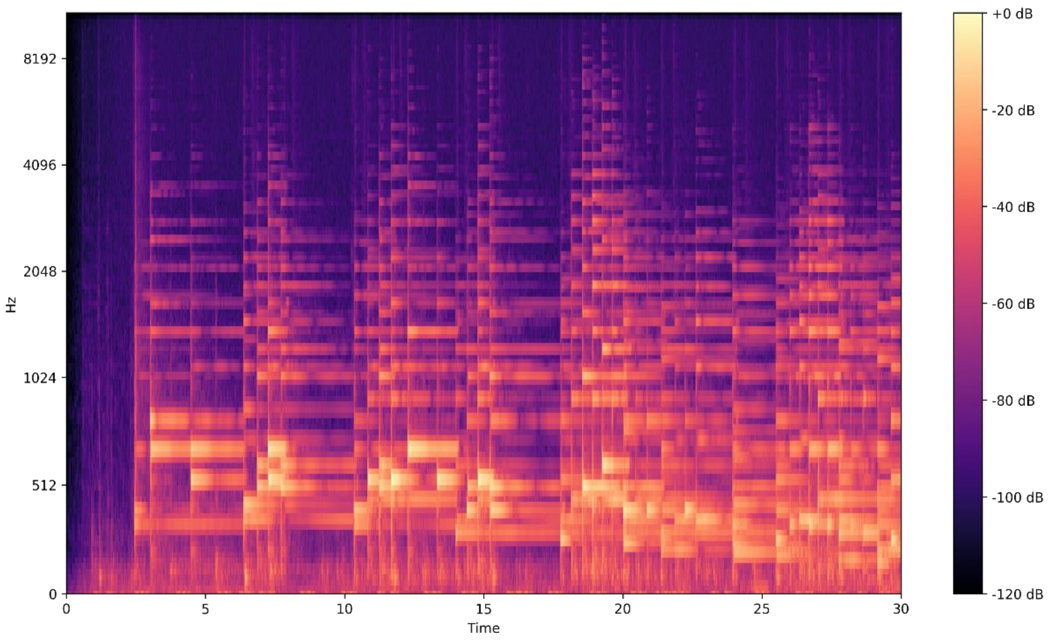

Figure 8: Spectrogram of the original piano recoring of "Clair de Lune"

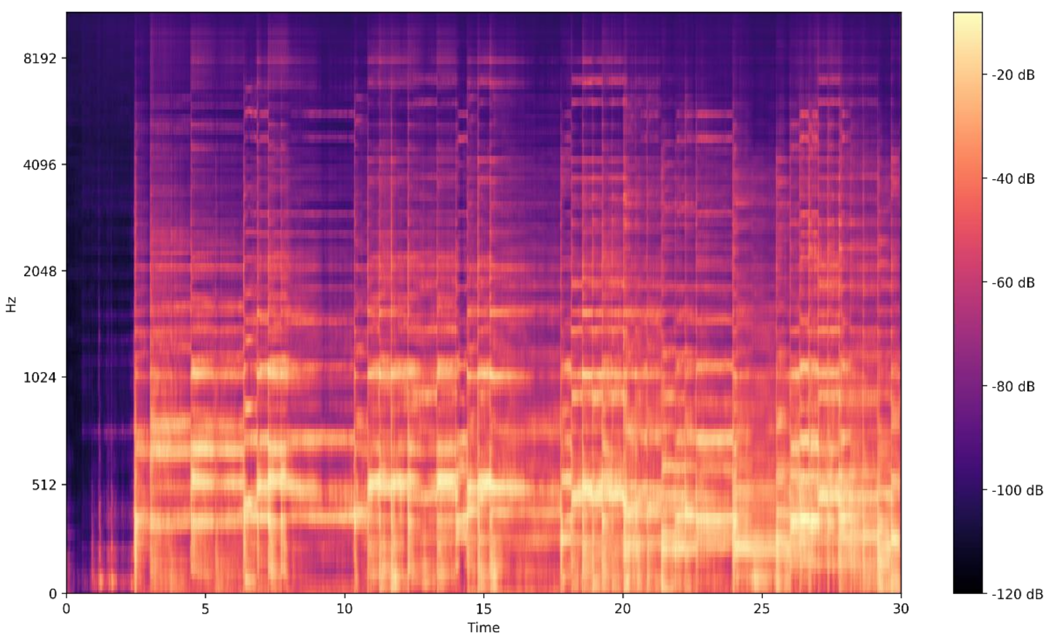

Figure 9: Spectrogram of the autoencoder output for the "Clair de Lune" recording

When comparing the two spectrograms, the remapping of the overtones is clearly visible. However, due to the unknown timbre of the piano used for the recording, the notes in the model's output are not as consistent as they should be. In addition, several frequencies are not accurately remapped, but instead distributed over several bands, which leads to a slightly atonal sound. The resulting file is included in the *results* folder as *clair_de_lune_after_autoencoder.wav*. Despite the noise and imperfect remapping, the timbre of the resulting file can still clearly be identified as an electronic piano, albeit with some phase distortion.

To correct the noise the autoencoder generated, its output is the used as input for the U-Net model. Figure 10 shows the result of the U-Net's operation. As discussed above, the most prominent visual difference in the resulting spectrogram is the reduction of noise in the higher frequency bands (above 1000Hz). The main audible effect however is noise reduction in the lower bands, which leads to a cleaner sound in the output. The final output is included in the *results* folder as well (*clair_de_lune_final.wav*).

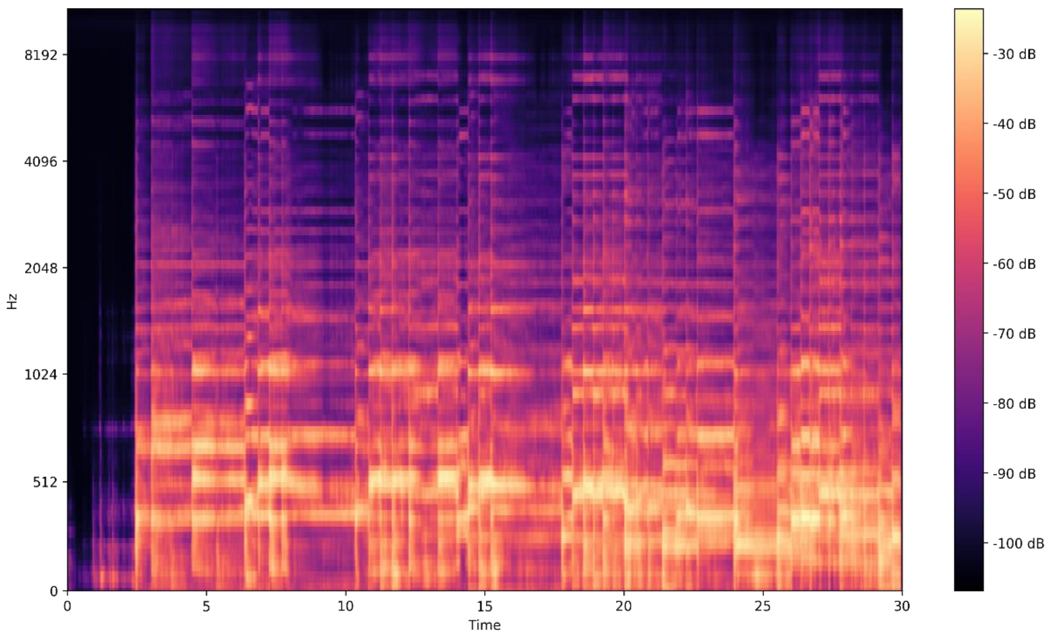

Figure 10: Final output of the system (after denoising U-Net is applied)

For comparison, the outputs of a file from the test set (an excerpt of a Mozart's Rondo Alla Turca) unknown to the model but synthesised with the same technique as the training data, are also included in the *results* folder (*mz_331_3_after_autoencoder.wav* and *mz_331_3_final.wav*).

Details of Implementation

The system's design principles adhere to those of standard Python libraries. Each component is located in a separate file, with additional scripts to simplify the execution of the program. The `MIDI2Audio.py` script contains methods to synthesise **.wav* files from MIDI data (*NB: To successfully run the script, FluidSynth must be installed on the operating system and accessible via the command line*). Synthesised files are stored in the `data/piano` and `data/synth` directories respectively.

The `DatasetCreator.py` script provides methods for converting the raw **.wav* files to data structures in accordance with the system's specifications. The generated files are stored in the `data/generated` directory.

The `AEDataset.py` and `UNetDataset.py` files are adaptions of PyTorch's built in dataset class, used by the autoencoder and the U-Net model respectively. They contain the functionality to access previously synthesised and stored data points.

The autoencoder module is located in the `Autoencoder.py` file. A wrapper class (`AEModel.py`) was created to streamline the training and generation processes. An executable script for training and evaluating the autoencoder is provided in `AEMain.py`.

The U-Net model is defined in the `UNet.py` file, with an executable script for training and testing the model located in `UNetMain.py`. Helper functions are defined in `Util.py` and static configuration as well as hyperparameters for the system are stored in `Hyperparameters.py`.

Finally, a main executable file that loads the two models from their saved state and provides a `pipeline()` method that feeds any given **.wav* file through both models is located in `Main.py`. In addition to the executable files, Jupyter Notebook files for evaluating the models separately, and both models combined, are included as well. The pretrained models were serialised and are stored in the `ae.torch` and `unet.torch` files respectively.

Conclusion

A description of the theory and implementation of an end-to-end audio resynthesis system were given and its operation on synthetic and real data was assessed and discussed. While the approach yields promising results with relatively little data, its performance is strongly dependent on the timbral specifics of the training data. Future work could extend the training process to several different types of piano sounds to extend the models' flexibility to deal with unknown data.

References

[1] Ronneberger, Olaf & Fischer, Philipp & Brox, Thomas. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. LNCS. 9351. 234-241. 10.1007/978-3-319-24574-4_28.

[2] lu, Xugang & Tsao, Yu & Matsuda, Shigeki & Hori, C.. (2013). Speech enhancement based on deep denoising Auto-Encoder. Proc. Interspeech. 436-440.

[3] Kiran, Bangalore & Thomas, Dilip & Parakkal, Ranjith. (2018). An Overview of Deep Learning Based Methods for Unsupervised and Semi-Supervised Anomaly Detection in Videos. Journal of Imaging. 4. 10.3390/jimaging4020036.

[4] Hsu, Wei-Ning & Zhang, Yu & Glass, James. (2017). Learning Latent Representations for Speech Generation and Transformation. 1273-1277. 10.21437/Interspeech.2017-349.

[5] Mao, Xiao-Jiao & Shen, Chunhua & Yang, Yu-Bin. (2016). Image Restoration Using Convolutional Auto-encoders with Symmetric Skip Connections.

[6] Liu, Ding & Wen, Bihan & Liu, Xianming & Huang, Thomas. (2017). When Image Denoising Meets High-Level Vision Tasks: A Deep Learning Approach.

[7] Griffin, Daniel & Lim, Jae. (1983). Signal estimation from modified short-time Fourier transform. IEEE Transactions on Acoustics, Speech, and Signal Processing. ASSP-32. 804 - 807. 10.1109/ICASSP.1983.1172092.

Appendix

Git

The project is available in the public domain on GitHub:
https://github.com/maxgraf96/DLAM_Assignment

Resources

The piano SoundFont was retrieved from
http://soundfonts.gonet.biz/search.php?goto=1&name=Steinway%20Grand%20D, accessed on 18.04.2020

The electronic piano SoundFont was retrieved from
https://www.flstudiomusic.com/2010/03/jr-soundfonts.html, accessed on 18.04.2020

The main audio test file was retrieved from:
https://commons.wikimedia.org/wiki/File:Clair_de_lune_(Claude_Debussy)_Suite_bergamasque.ogg, accessed on 19.04.2020