END-TO-END EQUALIZATION WITH CONVOLUTIONAL NEURAL NETWORKS

Marco A. Martínez Ramírez, Joshua D. Reiss

Centre for Digital Music,
Queen Mary University of London
London, United Kingdom
m.a.martinezramirez, joshua.reiss@qmul.ac.uk

ABSTRACT

This work aims to implement a novel deep learning architecture to perform audio processing in the context of matched equalization. Most existing methods for automatic and matched equalization show effective performance and their goal is to find a respective transfer function given a frequency response. Nevertheless, these procedures require a prior knowledge of the type of filters to be modeled. In addition, fixed filter bank architectures are required in automatic mixing contexts. Based on end-to-end convolutional neural networks, we introduce a general purpose architecture for equalization matching. Thus, by using an end-to-end learning approach, the model approximates the equalization target as a content-based transformation without directly finding the transfer function. The network learns how to process the audio directly in order to match the equalized target audio. We train the network through unsupervised and supervised learning procedures. We analyze what the model is actually learning and how the given task is accomplished. We show the model performing matched equalization for *shelving*, *peaking*, *lowpass* and *highpass* IIR and FIR equalizers.

1. INTRODUCTION

Equalization (EQ) is an audio effect widely used in the production and consumption of music. It consists of the modification of frequency content through positive or negative gains which change the harmonic and timbral characteristics of the audio. This is performed for different purposes, such as a corrective/technical filter to reduce masking or leakage within a mixing task, to modify the frequency response of a speaker system, or as an artistic or creative tool when recording a specific audio source.

An equalizer is normally implemented via a filter bank whose coefficients are obtained from the designed cut-off frequency f_0 and quality factor Q. In general, EQ is performed through an arbitrary boost or cut at a given f_0 and Q, and it can be applied in the time-domain and frequency-domain [1]. The filters can be classified into different classes such as *lowpass, highpass, peaking, and shelving*.

Taking into account that multiplying the spectrum of signals is the same as convolving their time-domain representation [2], filtering can be described by (1).

$$y(t) = x(t) * h(t) \rightarrow Y(k) = X(k) \cdot H(k) \tag{1}$$

Where h is the time-domain representation of the filter and x and y are the input and filtered signals respectively. H, X, and Y are the respective frequency-domain representations. In this manner, EQ can be achieved with time-domain convolutions, where the transfer function of the filter bank can be expressed through various signals in the time-domain and the equalized audio signal is obtained through the respective convolutions. Therefore, we investigate EQ as a time-domain convolution transformation, where the inherent content of the input and filtered signals can lead a convolutional neural network (CNN) to match a target frequency response.

Given an arbitrary EQ configuration, our task is to train a deep neural network to learn the specific transformation. In this way, an optimal filter bank decomposition and its latent representation are learned from the input data, and these are transformed and decoded to obtain an audio signal that matches the target. Thus, we explore whether the model can be used for EQ matching using an end-to-end architecture, where raw audio is both the input and the output of the system.

We train a model that matches an EQ objective without explicitly obtaining the parameters of the filters ($gain$, f_0 and Q). We show that a procedure based on convolutional and fully connected layers, via time-domain convolutions and latent-space modifications, can lead us to perform EQ matching or modeling. We analyze what the model is actually learning and use a relevant loss function in the time and frequency domains in order to achieve the equalizer task.

The rest of the paper is organized as follows. In Section 2 we summarize the relevant literature related to equalization matching and end-to-end learning. We formulate our problem in Section 3 and in Section 4 we present the methods. Sections 5, 6 and 7 present the obtained results, their analysis and conclusion respectively.

2. BACKGROUND

2.1. EQ Matching and Automatic Equalization

Several methods have been implemented in order to obtain the parameters of the filters or to match a specific frequency response. [3] provides a review of the different state-of-the-art approaches. These methods apply numerical optimization to find a transfer function that corresponds a given complex or magnitude frequency response. Most common techniques are based on the equation error method [4], the Yule-Walker algorithm [5], the Steiglitz-McBride method [6] and the frequency warped method [7].

Within an automatic mixing framework, [8, 9] explored multitrack EQ as a cross-adaptive audio effect, where the processing

of an individual track depends on the content of all the tracks involved, then, the gains of a five filter, first order, filter bank are obtained based on a perceptual loudness weighing.

Given the raw multitrack recording an the final mixture, [10] used least-squares optimization to estimate the gains and f_0 of FIR filters. [11] proposed a pitch tracking system to perform automatic EQ within a mastering task, where the selected pitches are considered as center frequencies for a set of second order peaking filters. [12] used least squares fitting to equalize an audio signal by using IIR filters with arbitrary frequency responses. A cross-adaptive EQ was implemented in [13], where center and cut-off frequencies of peaking and shelving filters were obtained through the minimization of spectral masking and source separation. Similarly, based on unmasking, [14] obtained the center frequencies and gains of peaking filters and [15] attains the gains of a six-band equalizer based on second-order IIR filters.

Based on an perceptual task, [16] proposed a method where the model is trained manually by the users and through nearest neighbor techniques the equalizer gains are obtained in order to match the training data. In a similar approach, [17, 18] investigated a model that associates the gain of each frequency band with the user's training data.

In order to obtain optimal results, most automatic EQ implementations rely on fixed architectures of filter banks or require prior knowledge of the type of filters to be modeled. Therefore, we explore a general architecture capable of performing equalization matching given an arbitrary frequency response.

2.2. End-to-end learning

End-to-end learning corresponds to the integration of an entire problem as a single indivisible task that must be learned from *end-to-end*. The desired output is obtained from the input by learning directly from the data [19]. Deep learning architectures using this principle have experienced significant growth, since by learning directly from raw audio signals, the amount of required prior knowledge is reduced and the engineering effort is minimized [20].

Most audio applications are in the fields of music information retrieval, music recommendation, and music generation. [20, 21] explored CNNs to solve automatic tagging tasks. The networks autonomously learn features related to the frequency and phase of the raw waveforms, although architectures based on spectrograms still yielded better results. In [22] an end-to-end neural network is investigated for the transcription of polyphonic piano music. In the context of end-to-end supervised source separation, [23] proposed an adaptive autoencoder neural network capable of learning a latent representation from the raw waveform.

Likewise, [24, 25] proposed models that generate audio sample by sample without the need handcrafted features and [26] obtained a model capable of performing singing voice synthesis based on *Wavenet* [27] autoencoders.

End-to-end learning has not been implemented for audio effect processing, though recent work demonstrated the usefulness of deep learning applied to intelligent music production systems. [28, 29] explored deep neural networks (DNN) to perform source separation in order to remix the obtained stems and [30] used autoencoders to achieve automatic dynamic range compression for mastering applications. Furthermore, most implementations rely on the magnitude of different frequency representations (spectrogram, melspectogram, etc.), thus omitting the phase information. This is sometimes not ideal, since the task under study could also be based on phase transformations, and therefore would not be learned by the models.

3. PROBLEM FORMULATION

For a specific EQ configuration or arbitrary combination of filters, consider x and y the raw and equalized audio signals respectively. We train a CNN autoencoder which operates as a filter bank and produces a latent representation $\boldsymbol{Z}$ of the given task. One CNN layer can be described by:

$$\boldsymbol{X}_k = \sum_{i=0}^{N-1} \boldsymbol{X}_{k-1}(n-i) \cdot \boldsymbol{W}_k(i) \tag{2}$$

Where $\boldsymbol{X}_k$ represents the feature map of the k_{th} layer, N represents the size of the input feature map $\boldsymbol{X}_{k-1}$ or input frame x in the case of the first layer, and $\boldsymbol{W}_k$ is the kernel matrix with K filters. The latent representation $\boldsymbol{Z}$ is obtained after a designated number of convolutional and subsampling layers.

Thus, in order to obtain a $\hat{y}$ that matches the EQ target y, we implement a deep neural network to modify $\boldsymbol{Z}$ based on the EQ task. Finally, the decoder implements the deconvolution operation and reconstructs the time-domain signal by inverting the operations of the encoder. We train the whole network within an end-to-end learning framework and we minimize a suitable metric between the target and the output of the network.

Based on an EQ matching task, we expect the network to learn the relevant filters $\boldsymbol{W}_k$, latent representation $\boldsymbol{Z}$ and further manipulation. We attempt to find a general architecture that can serve as a matching equalizer based on an arbitrary time-invariant EQ target.

4. METHODS

4.1. Model

In order to implement the network, we followed a similar procedure as [23], although based entirely on the time-domain. The model can be divided into three parts: adaptive front-end, synthesis back-end and latent-space DNN. The model is depicted in Fig. 1.

4.1.1. Adaptive front-end

The adaptive front-end consist of a convolutional encoder. It contains two CNN layers, one pooling layer and one residual connection for the back-end. The front-end performs time-domain convolutions with the raw waveform in order to map it into a latent-space. It also generates a residual connection which facilitates the reconstruction of the audio signal by the back-end. This differs

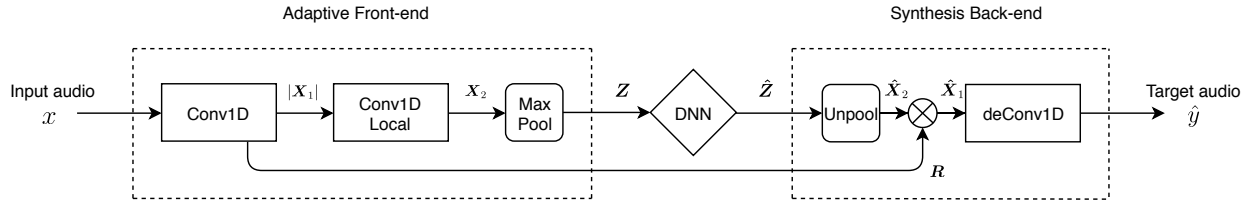

Figure 1: *Block diagram of the proposed model; adaptive front-end, synthesis back-end and latent-space DNN.*

from traditional autoencoders, where the complete input data is encoded into a latent-space, which causes each layer in the decoder to solely generate the complete desired output [31]. Furthermore, a full encoding approach such as [25, 27] will require very deep models, large data sets and difficult training procedures.

The input layer has 128 one-dimensional filters of size 64. Based on (2), the operation performed by the first layer can be described by (3).

$$\boldsymbol{X}_1 = x * \boldsymbol{W}_1 \tag{3}$$

$$\boldsymbol{R} = \boldsymbol{X}_1 \tag{4}$$

Where $\boldsymbol{R}$ is the matrix of the residual connection, $\boldsymbol{X}_1$ is the feature map or frequency decomposition matrix after the input signal x is convolved with the kernel matrix $\boldsymbol{W}_1$. The first layer is followed by the *absolute value* as non-linear activation function.

The second layer has 128 one-dimensional filters of size 128 and each filter is locally connected. This means we follow a filter bank architecture by having unshared weights in the second layer since each filter is only applied to its corresponding row in $|\boldsymbol{X}_1|$. The filters in this layer are larger due to convolving $|\boldsymbol{X}_1|$ with suitable averaging filters $\boldsymbol{W}_2$ could lead the model to learn smoother representations [23], such as envelopes. This layer is followed by the *softplus* non-linearity.

$$\boldsymbol{X}_2 = \mathit{softplus}(|\boldsymbol{X}_1| * \boldsymbol{W}_2) \tag{5}$$

Where $\boldsymbol{X}_2$ is the second feature map obtained after the local convolution with $\boldsymbol{W}_2$, the kernel matrix of the second layer.

The latent-space representation $\boldsymbol{Z}$ is achieved by the *max-pooling* operation. This pooling function consists of a moving window of size 16 applied over $\boldsymbol{X}_2$ and the maximum value within that window correspond to the output. Also, the positions in time of the maximum values are stored and used by the decoder.

4.1.2. Synthesis back-end

In order to invert the operations performed by the front-end, the decoder consists of one CNN layer and one unpooling layer. Since the *max-pooling* function is non-invertible, the inverse can be approximated by recording the locations of the maximum values in each pooling window [32] and only upsampling $\boldsymbol{Z}$ at these time indices. Thus the discrete approximation $\hat{\boldsymbol{X}}_2$ is obtained.

The approximation $\hat{\boldsymbol{X}}_1$ of matrix $\boldsymbol{X}_1$ is obtained through the element-wise multiplication of the residual $\boldsymbol{R}$ and $\hat{\boldsymbol{X}}_2$.

$$\hat{\boldsymbol{X}}_\mathbf{1} = \boldsymbol{R} \cdot \hat{\boldsymbol{X}}_2 \tag{6}$$

Depending on whether $\boldsymbol{Z}$ has been modified or not, (6) can be seen as a sampling or transformation of $\boldsymbol{X}_\mathbf{1}$.

The final layer corresponds to the deconvolution operation, which can be implemented by transposing the first layer transform. This layer is not trainable since its kernels are transposed versions of $\boldsymbol{W}_1$. In this way, the synthesis layer reconstructs the audio signal in the same manner the front-end decomposed it.

$$\hat{y}(t) = \hat{\boldsymbol{X}}_\mathbf{1} * \boldsymbol{W}_1^T \tag{7}$$

All convolutions are along the time dimension and all strides are of unit value. This means, during convolution, we move the filters one sample at a time.

4.1.3. Latent-space deep neural network

The latent-space DNN contains two layers, which are based on locally connected and fully connected dense layers respectively. Thus, following the filter bank architecture, the first layer applies a different dense layer to each row of the matrix $\boldsymbol{Z}$. Each of the locally connected dense layers has 64 hidden units and is followed by the *softplus* activation function. The second layer consists of a fully connected neural network of 64 hidden units, which is applied in each row of the output matrix from the first layer. It is also followed by the *softplus* activation function.

The output of the max pooling operation $\boldsymbol{Z}$ corresponds to an optimal latent representation of the input audio given the EQ task. The DNN is trained to modify this matrix, thus, a new latent representation $\hat{\boldsymbol{Z}}$ is fed into the synthesis back-end in order to reconstruct an audio signal that matches the target task.

4.2. Training

The training of the model is performed in two steps. The first step is to train both the adaptive front-end and the synthesis back-end for an unsupervised learning task. This can be considered as a pre-training of the autoencoder since the model showed better results than when only trained with the second training step. The second

step consists of an end-to-end supervised learning task based on a given EQ target.

During the pretraining only the weights $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are optimized and both the raw audio x and equalized audio y are used as input and target functions. This means the model is being prepared to reconstruct the input and target data in order to have a better fitting when training for the EQ task. Once the convolutional autoencoder is trained, the latent-space DNN is incorporated in the model. Hence, the second training procedure consists in using as objectives of the model x and y as input and target respectively. During the end-to-end learning, all the weights of the convolutional and dense layers are updated. This is done independently for each EQ task.

The loss function to be minimized is based in time and frequency and described by (8).

$$loss = kl(Y, \hat{Y}) + mse(Y, \hat{Y}) + mae(y, \hat{y}) \tag{8}$$

Where *kl* is the normalized Kullback-Leibler divergence, the mean squared error is *mse*, and *mae* is the mean absolute error. Y and $\hat{Y}$ are the frequency magnitude of the target and output respectively, and y and $\hat{y}$ their respective waveforms. We use a 1024-point Fourier transform (FFT) in order to obtain Y and $\hat{Y}$, which we extract on the GPU using *Kapre* [33].

We selected a more specialized loss function since by introducing spectral terms in a frequency related task, such as EQ, fewer training iterations were required. In both training procedures the input and target audio is windowed by a *hanning* function into frames of 1024 samples with hop size of 64 samples. The batch size consisted of the total number of frames per audio sample and 100 iterations were carried out in each training step. *Adam* is used as optimizer.

4.3. Dataset

The raw audio x is obtained from the *Salamander Grand Piano V3* dataset[1], which consists of a *Yamaha C5* grand piano sampled in minor thirds from the lowest *A* note and with 16 velocity layers for each note. The dataset is augmented by pitch shifting each note until all the available semitones of the piano are obtained. This gives us a total of 1440 samples. The piano notes are downsampled to 16 kHz and trimmed to 4 seconds. The test and validation subsets correspond to 10% of the dataset and contain a musical note (*B*) not present in the training subset.

The EQ targets y are obtained by applying the filters described in Table 1.

Table 1: *Filter parameters of the EQ targets.*

EQ	filter type	order	gain (dB)	f_0 (Hz)	Q
$shelving$	IIR	2	10	500	0.707
$peaking$	IIR	2	10	500	0.707
$lowpass$	FIR	50	0	500	.
$highpass$	FIR	50	0	500	.

[1] CC BY

5. RESULTS

The unsupervised and supervised learning steps were performed for each type of EQ target. Then, the models were tested with samples from the test dataset.

Fig. 2 shows various visualizations from the front-end and back-end of the autoencoder after the unsupervised training procedure. Fig. 2a displays the waveform and frequency magnitude of a test frame x of 1024 samples and its respective reconstruction $\hat{x}$. The weights of the first convolutional layer $\boldsymbol{W}_1$ can be seen in Fig. 2b, where the first 32 filters are shown.

Consequently, in order to obtain $\hat{x}$, different plots from the front-end, latent-space and back-end are shown in Figs. 2c-2e. The results of (3) can be seen in Fig. 2c where the first 32 rows of $\boldsymbol{X}_1$ are displayed. Fig. 2d presents their latent-space representation $\boldsymbol{Z}$, which is obtained through the second convolutional and subsampling layers. Fig. 2e shows $\hat{\boldsymbol{X}}_1$, which is the result of (6) and the input to the deconvolution layer, the prior step to obtain the output frame $\hat{x}$.

Following the pretraining of the autoencoder, the model is trained through an end-to-end supervised learning method. For each EQ task, Fig. 3 shows the results of selected samples from the test dataset. For a specific frame of 1024 samples, the input, target and output waveforms as well as their FFT magnitudes are displayed. The power spectrogram of the respective 4-second samples is also shown. Finally, together with the input and the target, the complete reconstructed output waveform of a shelving EQ task is presented in Fig. 4.

The performance of the models, and their respective losses (8) in time and frequency can be seen in Table 2.

Table 2: *Evaluation of the models with the test datasets. Loss values for each EQ task.*

EQ	*kl*	*mse*	*mae*	*loss*
$shelving$	0.021845	0.007764	0.002474	0.032083
$peaking$	0.022038	0.007847	0.002521	0.032406
$lowpass$	0.025365	0.005345	0.002710	0.033420
$highpass$	0.021463	0.000951	0.001293	0.023708

6. ANALYSIS

6.1. Adaptive front-end and back-end

From the results of the encoder and decoder, a comparison between the input and output waveforms, as well as their FFT magnitude (see Fig. 2a), it can be seen the model manages to reconstruct the input frame almost perfectly. There are minor differences between the magnitudes of the lower and higher frequencies, but it is worth mentioning that the network achieves this by optimizing only two convolutional layers.

During the first training step, the model learns the $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ weight matrices with 128 filters each. These filters correspond to the optimal weights of the autoencoder for the decomposition and reconstruction of the training data. As expected, from

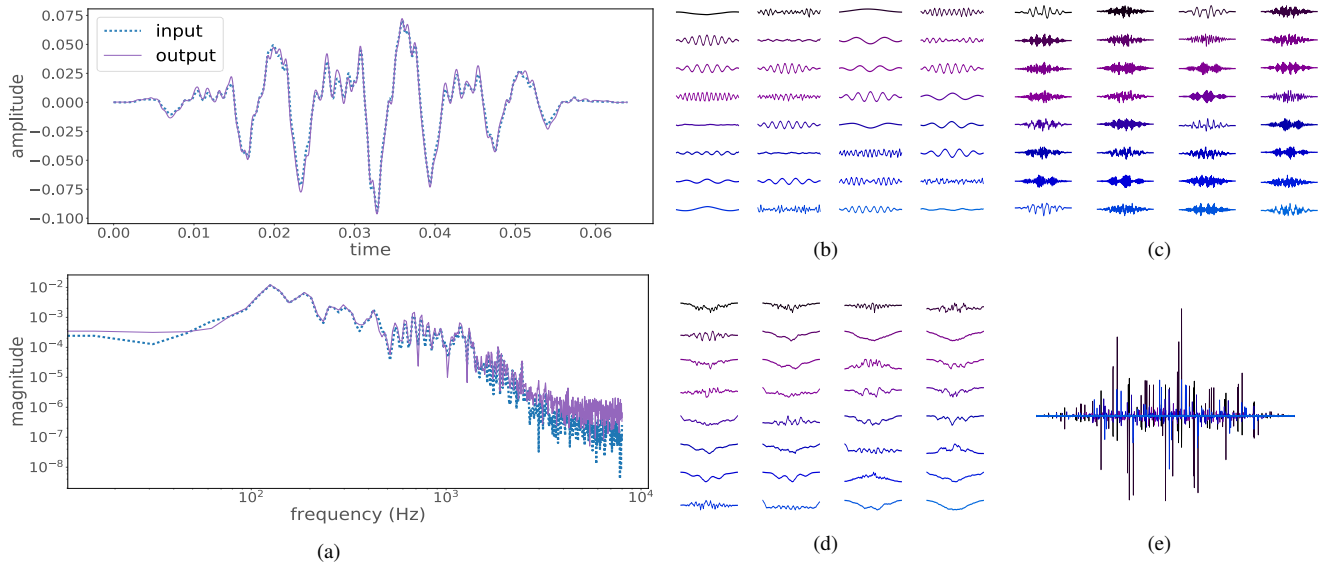

Figure 2: *Various plots from the front-end and back-end with the test dataset after the unsupervised learning step. 2a) Input (x) and output ($\hat{x}$) frames of 1024 samples and their respective FFT magnitude. 2b) First 32 filters ($\boldsymbol{W}_1$) of the first convolutional layer. 2c) First 32 rows of $\boldsymbol{X}_1$, resulting matrix of the convolution between the kernels $\boldsymbol{W}_1$ and the input frame x. 2d) Latent-space representation that is being encoded by the front-end. First 32 rows of $\boldsymbol{Z}$. 2e) Result of the element-wise multiplication between the residual $\boldsymbol{R}$ and the output of the unpooling layer. This is the input to the deconvolution layer prior to obtaining the output frame $\hat{x}(t)$. Vertical axes in 2b)-2e) are unitless and horizontal axes correspond to time.*

the $\boldsymbol{W}_1$ kernels shown in Fig. 2b, it can be observed the filters represent sinusoids and distributions of different frequencies. Also, upon examination of all the weights, we find some redundancy between the filters. This can be improved by adding kernel or activity regularizations, such as the L_1 or L_2 norm regularizes. In addition, some learned weights follow the *hanning* window shape, which makes sense given that all the input frames were windowed.

From the feature map matrix $\boldsymbol{X}_1$ (see Fig. 2c), the filters $\boldsymbol{W}_1$ are actively acting as a filter bank or frequency selectors, since $\boldsymbol{X}_1$ correspond to the decomposition of the input data into different frequencies. Since this is also the residual matrix $\boldsymbol{R}$, the resulting features consist of the required frequencies from the input data in order to be reconstructed by the back-end and encoded by rest of the front-end.

The second convolutional layer is acting as a smoothing layer, since $\boldsymbol{X}_2$ correspond to positive and negatives envelopes from $\boldsymbol{X}_1$. This is due to the learned averaging filters and the absolute and softplus activation functions. The subsampled version $\boldsymbol{Z}$ is presented in Fig. 2d, where different types envelopes are evident. Therefore, the autoencoder is learning a latent-space representation based on the envelopes of selected frequencies.

Taking into account that the result of the unpooling layer $\hat{\boldsymbol{X}}_2$ corresponds to the values of $\boldsymbol{Z}$ at the time positions registered by the max-pooling layer and padded with zeros between each maximum value. The element-multiplication of $\hat{\boldsymbol{X}}_2$ with $\boldsymbol{R}$ generates a discrete version of the latter, which indicates the amplitudes and positions in time that the deconvolution layer should use to reconstruct the input signal (see Fig. 2e). Thus, convolving $\hat{\boldsymbol{X}}_1$ with $\boldsymbol{W}_1^T$ generates the output frame presented in Fig. 2a.

The front-end and back-end manage to reconstruct the test piano notes with a loss value (8) of 0.104. Adding a simple latent-space neural network or increasing the number of filters in the convolutional layers would improve the results significantly. Also, since the training was performed with a hop size of 64 samples, an ideal unit sample hop size would decrease the loss value, although the training time will increase notably. Given that the unsupervised learning task only acts as pretraining step, and that the autoencoder has a relative small number of trainable parameters (24832) we consider these results to be satisfactory.

6.2. EQ task

Table 2 shows that the model performed well on each EQ task. To provide a reference, the mean loss value between the inputs and targets of the *shelving* testing samples is 1.21. The *kl* is fairly uniform across the four types of equalizers, with a minor increase for the *lowpass* EQ. The same can be said about the *mse* and *mae* with the exception of a significant decrease for the *highpass* EQ. Therefore, loss function values were minimal and the model is capable of matching the most common types of EQ, whether these are based on FIR or IIR filters.

The model achieved the best results during the *highpass* task, which could be an indication of the frequency distribution among the training data. Since only piano notes where used, and most spectral energy of acoustic pianos is within 250 Hz - 1 kHz with higher frequencies responsible for the perceived timbral quality of the notes [34]. Thus, having a 500 Hz cut-off frequency could sig-

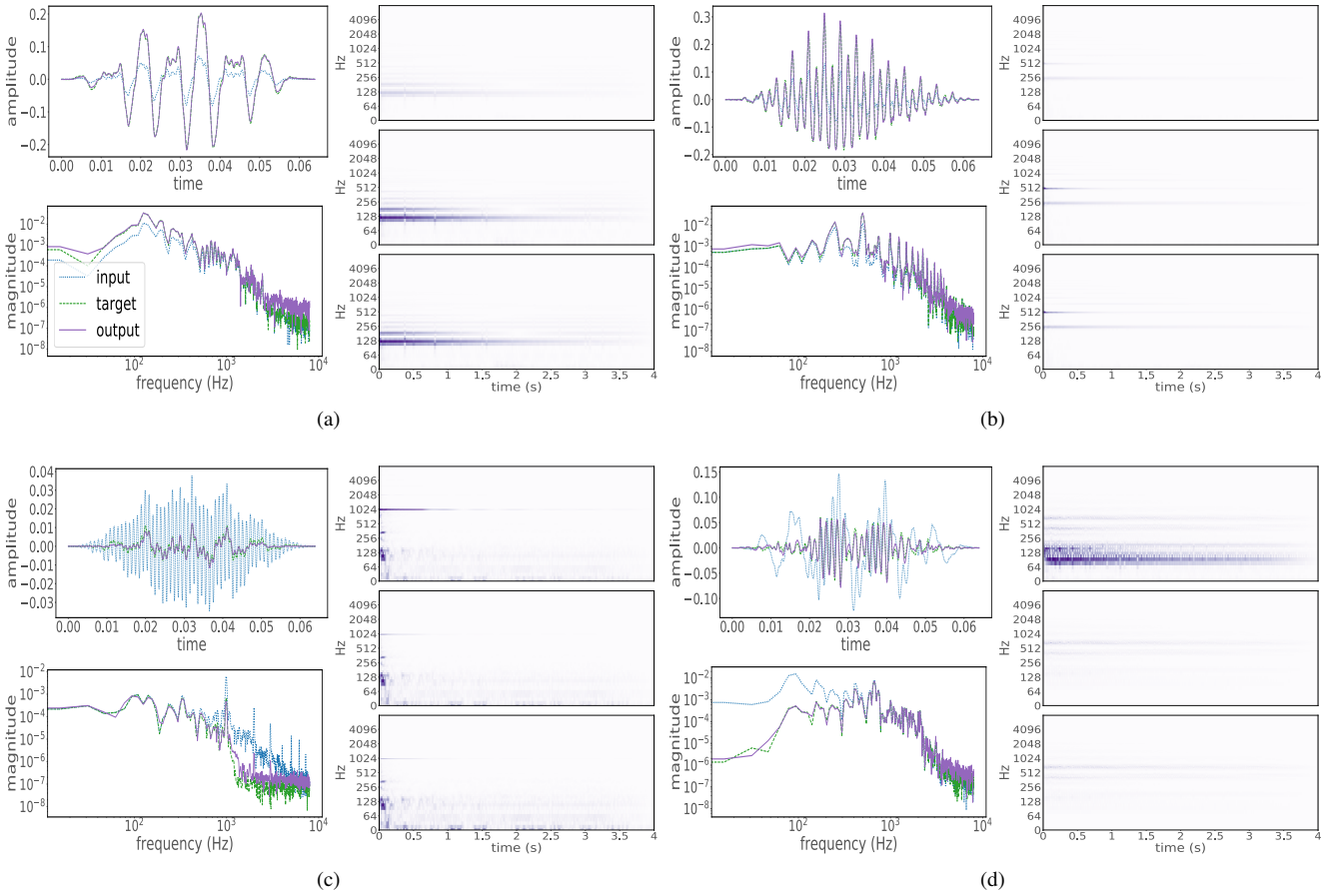

Figure 3: *Results with the test dataset for the following EQ tasks: 3a) shelving, 3b) peaking, 3c) lowpass and 3d) highpass. In 3a)-3d), the input, target and output frames of 1024 samples are shown in waveforms and their respective FFT magnitudes. In addition, for each EQ task and from top to bottom: input, target and output power spectrograms of the 4-second test samples are displayed. Color intensity represents higher energy.*

nify that the model effectively filters out the lower-end of the piano notes by efficiently learning the filters for this task. The slightly worse performance for the *lowpass* task could be further explored by adding kernel regularizations on the CNN layers.

Fig. 3 confirms the correct EQ matching for the different types of equalizers. The spectral and waveform comparison between input, target and output shows how accurate the model is at reconstructing an audio signal that matches the EQ task. For individual frames and complete piano notes, the different types of EQ are evident from the FFT magnitude and power spectogram respectively.

For the *shelving* EQ in Fig. 3a, the effect of the equalizer can be seen in the target and output spectral plots. The power spectrogram shows how the spectral energy was boosted for frequencies lower than 500 Hz. From the FFT magnitude it can be noticed a minor deviation in the lower-end of the target, where there is a boost increment around 20 Hz. This could indicate a weak generalization around these frequencies, which could be improved by using a loss function with higher resolution in the lower-end [7].

The *peaking* equalizer can be seen in Fig. 3b. The selective boost at 500 Hz is notorious both in the FFT magnitude and in the power spectrogram. There is a minor boost in the lower-end which is a consequence of the reasons discussed above. Overall the results indicate a significant fitting for the *peaking* EQ task. Accordingly, the model is able to match EQ tasks based on *peaking* and *shelving* IIR filters.

Likewise, the *lowpass* and *highpass* EQ targets were correctly accomplished. Fig. 3c-3d show the cut of frequencies higher than 500 Hz for the *lowpass* and the opposite for the *highpass*. As discussed, it can be seen the model performs the best for the *highpass* EQ task, obtaining a highly accurate matching between target and output in both time and frequency domains.

The model was trained in a frame-by-frame basis and the input frames were windowed. So the model learned the windowing procedure and the output frames followed the *hanning* shape. Therefore, in order to reconstruct the complete audio signal (see Fig. 4), no further windowing was needed. The overlapping procedure was carried out by applying a gain in order to ensure a Constant

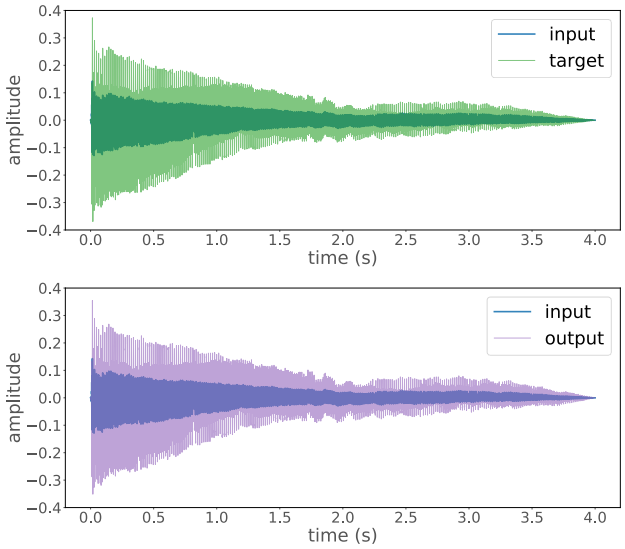

Figure 4: *For a test sample of the shelving EQ task, complete waveform reconstruction of the output and comparison with the input and target. See Fig. 3a for the power spectogram of these waveforms.*

Overlap-Add [35], which is specific to the type of window and hop size.

7. CONCLUSION

In this work, we proposed a novel deep learning architecture capable of performing an audio processing task such as EQ matching. To achieve this, based on the universal approximation capabilities of neural networks, we explored a convolutional adaptive front-end and back-end together with a latent-space deep neural network. Thus, we introduced a general purpose architecture for EQ matching able to model different types of equalizers and filters.

We showed the model matching *shelving*, *peaking*, *lowpass* and *highpass* IIR and FIR equalizers. For each EQ task the model was trained via unsupervised and supervised learning procedures. The latter corresponded to an end-to-end learning approach, which presents and advantage towards common methods of automatic EQ since no prior knowledge of the type of filters nor fixed filter bank architecture is required. Accordingly, the proposed model approximated the target as a content-based transformation without using or obtaining filter parameters. Therefore, the model learned an optimal filter bank decomposition and latent representation from the training data, and correspondingly, how to modify it in order to obtain an audio signal that matches the EQ task.

Possible applications for this architecture are within the fields of automatic mixing and audio effect modeling. For example, style-learning of a specific sound engineer could be explored, where the model is trained with several tracks equalized by the engineer and finds a generalization from the engineer's EQ practices. Also, automatic EQ for a specific instrument across one or several genres could be analyzed and implemented by the model.

Our implementation can serve as a baseline model for deep learning architectures in the context of audio processing. Linear transformations within a mixing task could be easily achieved. As future work, the exploration of recurrent or recursive neural networks or adaptive activation functions can improve the capabilities of the network to model much more complex audio effects. In this case, transformations involving temporal dependencies such as compression or different modulation effects, as well as complicated distortion effects, could be implemented.

A further exploration of the latent-space DNN, or deeper convolutional layers within the encoder and decoder could improve the results of the model. As well as regularizers and loss functions based on frequency wrappers. Also, since training on piano semitones provides only a sparse sampling of the frequency dimension, the generalization capability of the model should be extended for much more complex audio signals, such as noise, human voice or non-musical sounds. Therefore, a further exploration with a less homogeneous dataset together with an analysis of the type of filters learned by the model could benefit the design of a general architecture for modeling audio effects.

Finally, it is worth noting the immense benefit that generative music could obtain from deep learning architectures for intelligent music production. Our implementation could be used in the field of deep neural networks applied to generative music and automatic mixing production systems.

Acknowledgments

This work has been possible thanks to the computational resources by "Fusing Audio and Semantic Technologies for Intelligent Music Production and Consumption" (FAST IMPACt) EPSRC Grant EP/L019981/1.

8. REFERENCES

[1] Vincent Verfaille, U. Zölzer, and Daniel Arfib, "Adaptive digital audio effects (A-DAFx): A new class of sound transformations," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 14, no. 5, pp. 1817–1831, 2006.

[2] Pedro D. Pestana, *Automatic mixing systems using adaptive digital audio effects*, Ph.D. thesis, Universidade Católica Portuguesa, 2013.

[3] Vesa Välimäki and Joshua D. Reiss, "All about audio equalization: Solutions and frontiers," *Applied Sciences*, vol. 6, no. 5, pp. 129, 2016.

[4] Julius Orion Smith, *Introduction to digital filters: with audio applications*, vol. 2, Julius Smith, 2007.

[5] Benjamin Friedlander and Boaz Porat, "The modified yule-walker method of arma spectral estimation," *IEEE Transactions on Aerospace and Electronic Systems*, , no. 2, pp. 158–173, 1984.

[6] Leland B Jackson, "Frequency-domain steiglitz-mcbride method for least-squares iir filter design, arma modeling, and periodogram smoothing," *IEEE Signal Processing Letters*, vol. 15, pp. 49–52, 2008.

[7] Aki Härmä et al., "Frequency-warped signal processing for audio applications," *Journal of the Audio Engineering Society*, vol. 48, no. 11, pp. 1011–1031, 2000.

[8] Enrique Perez-Gonzalez and Joshua D. Reiss, "Automatic equalization of multichannel audio using cross-adaptive methods," in *127th Audio Engineering Society Convention*, 2009.

[9] Enrique Perez-Gonzalez and Joshua D. Reiss, "Automatic mixing," *DAFX: Digital Audio Effects, Second Edition*, pp. 523–549, 2011.

[10] Daniele Barchiesi and Joshua D. Reiss, "Reverse engineering of a mix," *Journal of the Audio Engineering Society*, vol. 58, no. 7/8, pp. 563–576, 2010.

[11] Stylianos I. Mimilakis et al., "Automated tonal balance enhancement for audio mastering applications," in *134th Audio Engineering Society Convention*, 2013.

[12] Zheng Ma et al., "Implementation of an intelligent equalization tool using yule-walker for music mixing and mastering," in *Audio Engineering Society Convention 134*. Audio Engineering Society, 2013.

[13] Daniel Matz, Estefanía Cano, and Jakob Abeßer, "New sonorities for early jazz recordings using sound source separation and automatic mixing tools.," in *16th International Society for Music Information Retrieval Conference (ISMIR)*, 2015.

[14] Sina Hafezi and Joshua D. Reiss, "Autonomous multitrack equalization based on masking reduction," *Journal of the Audio Engineering Society*, vol. 63, no. 5, pp. 312–323, 2015.

[15] David Ronan et al., "Automatic minimisation of masking in multitrack audio using subgroups," *IEEE Transactions on Audio, Speech, and Language processing*, 2018.

[16] Dale Reed, "A perceptual assistant to do sound equalization," in *5th International Conference on Intelligent User Interfaces*. ACM, 2000, pp. 212–218.

[17] Andrew T Sabin and Bryan Pardo, "A method for rapid personalization of audio equalization parameters," in *17th ACM International Conference on Multimedia*, 2009.

[18] Bryan Pardo, David Little, and Darren Gergle, "Building a personalized audio equalizer interface with transfer learning and active learning," in *2nd International ACM Workshop on Music Information Retrieval with User-Centered and Multimodal Strategies*, 2012.

[19] Urs Muller et al., "Off-road obstacle avoidance through end-to-end learning," in *Advances in neural information processing systems*, 2006.

[20] Sander Dieleman and Benjamin Schrauwen, "End-to-end learning for music audio," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014.

[21] Jordi Pons et al., "End-to-end learning for music audio tagging at scale," in *31st Conference on Neural Information Processing Systems (NIPS)*, 2017.

[22] Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon, "An end-to-end neural network for polyphonic piano music transcription," *IEEE/ACM Transactions on Audio, Speech and Language Processing*, vol. 24, no. 5, pp. 927–939, 2016.

[23] Shrikant Venkataramani, Jonah Casebeer, and Paris Smaragdis, "Adaptive front-ends for end-to-end source separation," in *31st Conference on Neural Information Processing Systems (NIPS)*, 2017.

[24] Soroush Mehri et al., "Samplernn: An unconditional end-to-end neural audio generation model," in *5th International Conference on Learning Representations (ICLR)*, 2017.

[25] Jesse Engel et al., "Neural audio synthesis of musical notes with wavenet autoencoders," in *34th International Conference on Machine Learning*, 2017.

[26] Merlijn Blaauw and Jordi Bonada, "A neural parametric singing synthesizer," in *Interspeech 2017*.

[27] Aaron van den Oord et al., "Wavenet: A generative model for raw audio," *CoRR abs/1609.03499*, 2016.

[28] Gerard Roma et al., "Music remixing and upmixing using source separation," in *2nd AES Workshop on Intelligent Music Production*, 2016.

[29] Stylianos I. Mimilakis et al., "New sonorities for jazz recordings: Separation and mixing using deep neural networks," in *2nd AES Workshop on Intelligent Music Production*, 2016.

[30] Stylianos I. Mimilakis et al., "Deep neural networks for dynamic range compression in mastering applications," in *140th Audio Engineering Society Convention*, 2016.

[31] Kaiming He et al., "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[32] Matthew D Zeiler and Rob Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision*, 2014.

[33] Keunwoo Choi, Deokjin Joo, and Juho Kim, "Kapre: On-gpu audio preprocessing layers for a quick implementation of deep neural network models with keras," in *34th International Conference on Machine Learning*, 2017.

[34] David M Koenig, *Spectral analysis of musical sounds with emphasis on the piano*, OUP Oxford, 2014.

[35] Jérôme Antoni and Johan Schoukens, "A comprehensive study of the bias and variance of frequency-response-function measurements: Optimal window selection and overlapping strategies," *Automatica*, vol. 43, no. 10, pp. 1723–1736, 2007.