Audio Engineering Society

Convention Paper

Presented at the 144th Convention
2018 May 23 – 26, Milan, Italy

Deep Learning for Timbre Modification and Transfer: an Evaluation Study

Leonardo Gabrielli[1], Carmine E. Cella[2], Fabio Vesperini[1], Diego Droghini[1], Emanuele Principi[1], and Stefano Squartini[1]

[1] *Università Politecnica delle Marche, Ancona, Italy*
[2] *IRCAM, Paris*

Correspondence should be addressed to L. Gabrielli (l.gabrielli@univpm.it)

ABSTRACT

In the past years, several hybridization techniques have been proposed to synthesize novel audio content owing its properties from two audio sources. These algorithms, however, usually provide no feature learning, leaving the user, often intentionally, exploring parameters by trial-and-error. The introduction of machine learning algorithms in the music processing field calls for an investigation to seek for possible exploitation of their properties such as the ability to learn semantically meaningful features. In this first work, we adopt a Neural Network Autoencoder architecture and we enhance it to exploit temporal dependencies. In our experiments the architecture was able to modify the original timbre, resembling what it learned during the training phase, while preserving the pitch envelope from the input.

1 Introduction

The music signal processing literature is rich of algorithms for the synthesis of novel sounds from two or more different sound sources to allow creative and surprising effects and enhance sound design practice. A number of techniques are known to that extent, although terms and definitions may slightly vary: *analysis-resynthesis*, *cross-synthesis*, *vocoding*, *morphing* or *hybridization* are terms used widely in the literature. According to the systematization of Caetano *et al.* [1], the terms *hybridization* and *morphing* can be employed, respectively, to distinguish between algorithms that aim at taking different features from each source and merge them together and algorithms that interpolate each feature from the two sources to obtain a mixed signal.

Cross-synthesis is defined by Julius O. Smith as a sound processing algorithm where the spectral envelope of one sound is impressed on the flattened spectrum of another [2], and a similar proposition is reported in [3], additionally including vocoding as a synonym.

Analysis-resynthesis is a family of techniques allowing to describe timbre and manipulating it and is often used in morphing by interpolating synthesis parameters obtained from two sound sources. Typical examples include sinusoidal modelling [3] or time-frequency domain transforms [4].

Hybridization, instead, requires extracting specific features of a signal. The construction of new signals is done by imposing some of the features of a signal onto another. Features can be extracted at different abstraction levels. Low-level representations are generic, have very high dimensionality and often give only access to geometric transformations (translations, rotations, etc.) [5]. Mid-level representations are often related to perceptual concepts and allow for transformations on specific variables such as pitch [6]. Finally, abstract representations are more expressive, have a low dimensionality and relates to high level musical concepts [7].

With the present work we start investigating the use of Machine Learning (ML) for the goal of obtaining new morphing or hybridization algorithms. The motivation for using ML for sound hybridization or morphing comes from the interesting results of Gatys *et al.* [8] in the image processing field. In their works the authors showed how it is possible, using the latent space of a deep neural network, to transfer high level features from one image to another. The proposed algorithm allows for an effective machine-learned representation of artistic style from a template image that can be transferred to a *content* image by preserving its subject while altering its perceived qualities (shape, line, colour, texture and so forth). It is interesting to remark that the transferred features are not simple effects but real traits of the style of the image.

Other researchers and machine learning developers experimented with the same algorithms to test whether these were able to produce similar results with audio signals, and specifically, musical signals. Currently, a few attempts, some of which are unpublished but have open source code released, report that current methods are inadequate to learn and represent music features [9, 10, 11]. This is probably related to the different application domain: images and spectrograms are different in that the first are isotropic while the second are not, and the information contained in the frequency and temporal axes have different scales, different structure and repetition thereof.

To the best of our knowledge, it seems that the only successful result of ML in this field is that obtained by the Magenta team with WaveNet in [12], where interpolation in the *latent code* feature space extracted by the WaveNet network produces sound morphing. The WaveNet architecture projects raw audio signals to a feature space that is learned by the network. In [12] an encoding WaveNet stage is followed by an inverse decoding (i.e. resynthesis) stage. One downside of this approach is the prohibitive computational cost. Up to now, training the network is possible only for specialized hardware available only to the Magenta team researchers. Furthermore, no tests have been conducted on time-varying audio, but only on the dataset described in the paper (later discussed in Section 4).

In the present work we follow a bottom-up approach, starting from simple and well-known ML architectures of low computational cost. Specifically, our approach involves a Neural Network Autoencoder in the time-frequency domain, trained on a audio signal to learn some of its features. The trained Autoencoder is then employed to process a test signal. The expected result is that features from the training signal are retained and imposed onto the test signal, i.e. what would be expected from a timbre transfer algorithm.

Section 2 proposes a ML architecture to learn and transfer spectral features, Section 3 reports comparative methods used during the experiments of Section 4. Results are discussed in Section 5 and conclusions are drawn in Section 6.

2 Proposed Method

Neural Network Architecture

The proposed neural network architecture is designed to capture spectral characteristics of audio signals in the time and frequency domain. The architecture adopted to perform this task is an autoencoder, i.e., a neural network trained to copy its input to its output [13]. A properly trained autoencoder, however, does not perform a simple copy operation, but learns significant characteristics of the training data. This property has been exploited in several different task in the literature, such as novelty detection [14], dimensionality reduction [15], and speech enhancement [16]. Here, the capabilities of the autoencoder are used to transform the input data based on the characteristics of the training data. The rationale here is that the autoencoder will try to reconstruct the input based on the knowledge acquired during the training phase, thus transforming the input signal based on the characteristics of the training data. An overview of the proposed architecture is shown in Figure 1 for the sake of clarity, and its details will now be presented.

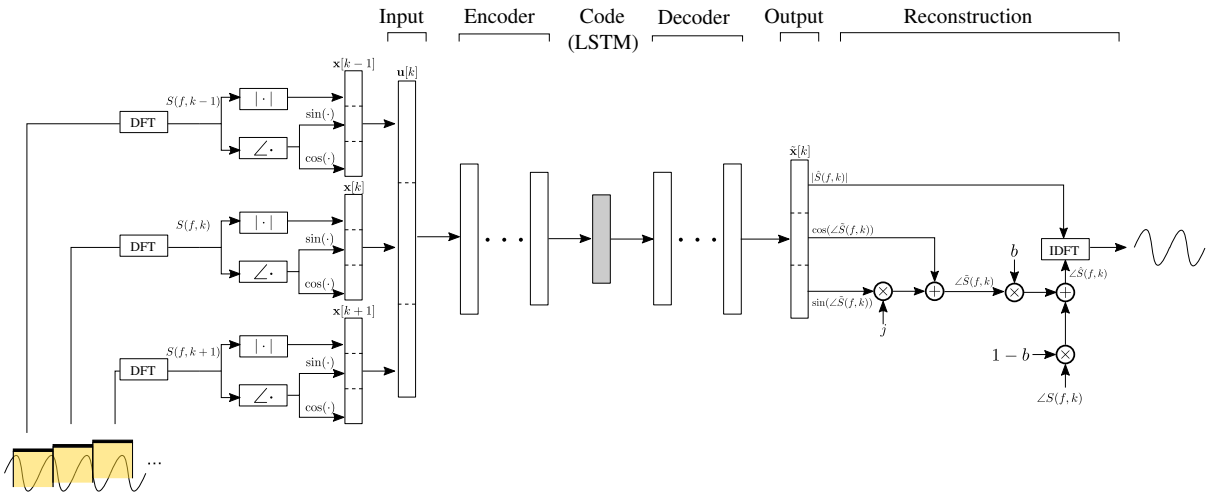


Fig. 1: Overview of the proposed architecture.

Denoting with $S(f,k)$ the Short-Time Fourier Transform (STFT) at frame k of an audio signal $s[n]$, the network is designed to accept an input vector $\mathbf{u}[k]$ composed of the STFT magnitude and phase of frames adjacent to the one being processed. The phase is expressed as $\sin(\angle S(f,k))$ and $\cos(\angle S(f,k))$ terms. Denoting with $\mathbf{x}[k]$ the vector

$$\mathbf{x}[k] = \begin{bmatrix} |S(f,k)| \\ \cos(\angle S(f,k)) \\ \sin(\angle S(f,k)) \end{bmatrix}, \tag{1}$$

the network input vector $\mathbf{u}[k]$ is given by:

$$\mathbf{u}[k] = \begin{bmatrix} \mathbf{x}[k-1] \\ \mathbf{x}[k] \\ \mathbf{x}[k+1] \end{bmatrix}. \tag{2}$$

Indicating with F the size of the FFT, the vectors $\mathbf{x}[k]$ and $\mathbf{u}[k]$ are respectively composed of $3F$ and $9F$ elements.

The encoding section of the network is composed of a stack of fully connected layers of gradually narrower size, reaching the inner hidden layer that yields the latent code of representation. Similarly, the decoding section is composed of a stack of fully connected layers, excepts for the fact that the layer size gradually grows from the inner hidden layer to the output layer.

The output layer of the network is composed of 3 groups of F neurons, so that the output vector is arranged as $\mathbf{x}[k]$. Each group is specialized to reconstruct a component of the complex $S(f,k)$ as:

$$\tilde{\mathbf{x}}[k] = \begin{bmatrix} \left|\tilde{S}(f,n)\right| \\ \cos(\angle \tilde{S}(f,k)) \\ \sin(\angle S(f,k)) \end{bmatrix}. \tag{3}$$

The ReLU activation function is only applied to the group related to the magnitude reconstruction, in order to constrain the output values to be positive. The tanh activation function is applied to all other neurons in the architecture, in order to allow the signals to assume values in the range $[-1,1]$.

Such a simple configuration is able to learn and reproduce an averaged spectrum, so that a dataset containing chromatic scales reproduces a signal that contains all musical notes at the same time. This is not sufficient in the current context, thus, temporal dependencies must be learned by the network. This can be obtained by using one or more recurrent layers as inner layer of the network. In particular, we used one hidden layer composed of Long-Short Term Memory block (LSTM). These blocks can efficiently exploit a long-range temporal context by means of connections between units which form directed cycles, and store state information in the cell.

A feature-wise batch normalization is applied to the output of each layer of the network in order to reduce the internal covariance shift and to better distribute the latent representations obtained during the network training procedure. In addition, the dropout technique was employed during the neural network training to prevent

overfitting and increase the generalization performance of the neural network in reconstructing the input signal.

Training is performed by using a dataset composed of audio signals characterized by the desired timbre. The network is trained to minimise the mean squared error between the estimated signal $\tilde{\mathbf{x}}[k]$ and the input signal $\mathbf{x}[k]$ by using the AdaDelta stochastic gradient-based optimization algorithm. It was chosen because it is well-suited for dealing with sparse data and is robust to different choices of model hyperparameters. Furthermore no manual tuning of learning rate is required.

Resynthesis

During the generation phase, a novel input is employed featuring a different instrument/timbre from the ones used during training. Special care must be taken in the inverse STFT processing in order to provide time-domain reconstruction without phase artefacts. Owing from works in cross-synthesis and spectral morphing [17], the predicted spectral magnitude and phase can be mixed with the magnitude and phase of the input signal. Denoting with $\hat{S}$ the DFT of the final reconstructed signal, the output magnitude and phase are obtained as:

$$|\hat{S}| = a|S| + (1-a)|\tilde{S}| + a_M\sqrt{|S|\cdot|\tilde{S}|}, \quad (4)$$

$$\angle\hat{S} = b\angle\tilde{S} + (1-b)\angle S, \quad (5)$$

where $0 < a, b, a_M < 1$ determine the proportion between the estimated and original signal components. In practice, the original magnitude information is not used, i.e., $|\hat{S}| = |\tilde{S}|$, while choosing b close to 0.5 allows to obtain a time domain signal with reduced artefacts. This is the choice followed for all reported experiments.

3 Comparative Methods

To provide a comparison to the proposed approach, the following methods for timbre transfer have been implemented to be tested with the same audio material. The implemented algorithms are:

- **spectral envelope hybridization**: in this case, the timbre transfer is performed by computing the spectral envelope of both signals, flattening the envelope of the target signal by deconvolution and then multiplying it by the source signal; in this context, the spectral envelope is based on the cepstrum as follows:

$$E = DFT(W_{LP}(DFT^{-1}(log(|DFT(s)|)))) \quad (6)$$

  where W_{LP} is a low-pass filter in the cepstral domain called *liftering* filter and S is a signal in time domain;

- **MFCC-based mosaicing**: this method performs timbre transferring by replacing the frames of the source signal by specific frames of a target dataset of sounds (that we call dictionary), where the match is done by some distance measure in a specific domain; in this context, we used a k-nn algorithm applied on the first 14 Mel-frequency cepstral coefficients (MFCC) of each frame [18].

For the MFCC-based approach, the length of the generated output sound is equal to the length of the input sound. For the other methods, instead, the generated output is as long as the longest between input and target, where the shorter one is repeated as necessary during the process.

The spectral envelope hybridization and the MFCC-based mosaicing methods do not require parameters and perform a full timbre transfer between the processed signals; on the other hand, the DFT morphing requires the setting of the interpolation parameters. In this context we decided to create the output sound by using the phases of the source signal and the amplitudes of both signals while keeping some bias on the target signal. To achieve this, we set the interpolation parameters as follows: $a = 0, b = 1, a_M = 1$.

4 Experiments

Experimental Setup

The algorithm has been implemented in the Python programming language using the Keras deep learning libraries. All the experiments were performed on a CINECA Galileo computational node with Nvidia K80 accelerators.

The neural networks were trained with the Adadelta gradient descent algorithm and a learning rate equal to 1.0, $\rho = 0.95$, $\varepsilon = 10^{-6}$. The maximum number of epochs was set to 300 and an early stopping strategy on the validation set loss with 20 epochs of patience

was employed for regularization. Each training iteration involved a number of samples (i.e., batch size) between the 5% and the 10% of total samples present in the training set, depending on the amount of samples present on the latter.

The network weights were initialized with a random Gaussian distribution with mean $\mu = 0$ and standard deviation $\sigma = 0.1$, as it usually provides an acceptable initialization in our experience. Several network topologies were tested, varying the number of layers and units per layer. Indeed the number of layers of the encoder has been varied from 1 to 4 while the number of unit for each layer from 80 to 4096. The decoder part was mirrored with respect to the encoder. The number of LSTM layers has been varied from 0 (no LSTM layer) to 1 with a number of units from 80 to 1024.

Two datasets have been employed for the evaluation of the proposed approach. Dataset 1 is an internal dataset of recorded solo instrumental or vocal tracks. Each audio file consisted of 30-120 seconds of audio. The following instruments were considered: clean electric guitar (2 files), distorted electric guitar (3 files), synth pad (2 file), trumpet (3 files), electric piano (3 files), male voice (1 file), female voice (2 files). All the files are real recordings of solo performance and thus contains a rich content of notes, chords (for polyphonic instruments), legatos, glissandi, and other expressive effects. Each file is played on a single tonality.

Dataset 2 is built from a very large musical instrument dataset shared by the Magenta project team[1], containing single notes for eleven classes of musical instruments. In creating Dataset 2 we picked ten from the eleven classes, discarding the bass class because of its narrow coverage of the frequency spectrum, and randomly selected 3500 audio files for each class, to give each class the same dimensionality. This dataset is composed of short files containing a single note each, at different dynamic levels and pitches. The differences between the two datasets have been exploited to observe the role of the training data on the resulting output.

All files from Dataset 1 have been sub-sampled to 22050 Hz to reduce the computational cost, while the files from Dataset 2 are sampled at 16 kHz and were left as such. The STFT of the audio signals were computed with a 2048-points DFT, 50% overlap and window size calculated in order to reach 43 frames per second.

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

Table 1: Hyperparameters used for the experiment with training on the distorted guitar track of the song *Sweet Child O' Mine*.

Network layout	**Dropout**
Encoder: 1024, 808 LSTM: 808 Decoder: 808, 1024	input units to drop $p = 0.1$
Training epochs	**Optimiser parameters**
300, 20 patience Validation split = 10%	learning rate = 1 $\rho = 0.95$, $\varepsilon = 10^{-6}$
Batch Normalization: $\varepsilon = 10^{-6}$, $\mu = 0.9$	

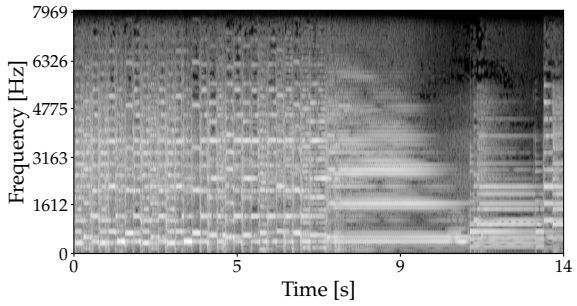
Audio samples are available at `https://gitlab.com/a3labPapers/CompanionFiles/tree/master/AES-XSynth`.

5 Results

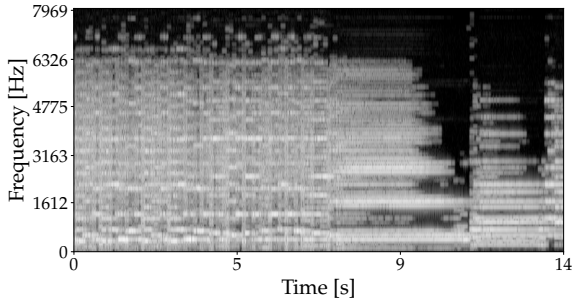
This section reports experiments with different audio sources as target and input conducted on Dataset 1 and 2 together with a qualitative analysis of the synthesized outputs. A first batch of experiments was conducted with Dataset 1 to tune the network hyperparameters. The outputs produced by autoencoders trained on each file in the dataset have been evaluated by analysis of their spectrograms and informal listening tests. The hyperparameters reported in Table 1 have been found to obtain acceptable results, which will be discussed shortly. Without considering the output layer and input layers, the network is composed of five layers: 2 layers respectively of 1024 and 808 units, one LSTM layer composed of 808 units, and 2 layers respectively of 808 and 1024 units.

Reconstruction Tests

Preliminary tests were conducted to ensure that the autoencoder is able to reconstruct an input signal if it is also used for training. The network is able to reconstruct sufficiently well the original file, although some compression is inherent to the compressive autoencoding process. Figure 2 shows the spectrogram of an electric guitar from Dataset 1 playing arpeggios and chords and its reconstruction. The compression is visible from the spectrograms especially at high frequency.



(a) Original electric guitar track (input and target).



(b) Reconstruction.

Fig. 2: Spectrograms from (a) an electric guitar track, (b) its reconstruction when using (a) as both target and input. The hyperparameters for (b) are reported in Table 1.

Dataset 1

These experiments were done with a female singing voice input file (from now on, in short FV1). This choice is motivated by the fact that the voice has subtle pitch variations (vibratos, glissandi, etc.) and the voice presents pitched, unpitched and unvoiced audio.

As an example of the results that can be obtained by the proposed architecture, some selected outputs are analysed in detail. A distorted guitar track from Dataset 1 (playing the tune in *Sweet Child O' Mine* by Guns N'Roses) is taken as the target for the proposed architecture with parameters shown in Table 1. The resulting output file is named R1 for short, and its spectrogram is shown in Figure 3(b). For comparison, the distorted guitar track has also been used for the MFCC-based mosaicing algorithm and for the spectral flattening algorithm. The input file used for all techniques is FV1. Its spectrogram is shown in Figure 3(a), followed by the spectrograms obtained by the other methods.

With the proposed approach, timbre is quite coherent from frame to frame if compared, for example, to the MFCC-based method, thus resulting in a more convincing output. In the MFCC hybridization, furthermore, it is also quite apparent that frames from the target signal appear from time to time in an inconsistent way exposing explicit features of the target (for example, a recognizable note in a riff). Finally, the spectral flattening method has a recognizable vocoder-like timbre, with the musical structure of the target file appearing together with its spectral content, which is an undesired feature in this context (see, for example, the arpeggios of the target song appearing in the spectrogram and chromagram approximately at 5 s on to the end).

Chromagrams from audio files shown in Figure 3 are shown in Figure 4, to compare the pitch trajectories and the presence of spurious chromatic components. The chromagrams obtained from the proposed architecture (Figure 4(b),(e)) show similar pitch trajectories to the input signal (Figure 4(a)). We observe that the network fails to follow the pitch of the input when it is outside the range learned during the training phase. This can be seen around second 2 in Figure 4(e), where the high pitch of the input file reaches a B4 which the network cannot match, due to the lack of notes above G#4 in the training set.

Dataset 2

The experiments with Dataset 2 were conducted by training an autoencoder for each of the ten instrument classes, thus greatly increasing the complexity of the problem. Each trained autoencoder was subsequently used to synthesize audio with FV1 as input, resulting in ten audio files of different timbre and similar pitch contour. As an example of the good pitch tracking capabilities of the proposed architecture, we report a DFT (4096 points) calculated for each file at a specific time interval, where the input file shows a pitched */i:/* phoneme at frequency 497.6 Hz (B4 + 13 cents), see Figure 5. Each DFT has different features (e.g., spectral slope, spectral centroid, presence of noise, etc.) but all have same pitch.

From the experiments with Dataset 2 we observed that the network cannot follow glissando, pitch bending and vibrato from the input because the dataset has no time-varying pitch to be learned, being composed of static single notes only. This is apparent by applying FV1

as input and results in lower-quality note transitions compared to the autoencoder trained on Dataset 1.

Judging timbre learning and transferring with this dataset is difficult because of the extremely large variety of tones in an instrument class and the large number of files to evaluate. The network cannot learn all the timbres of the different instruments in a class, but it learns instead an averaged spectrum of the whole class. It must be noted that we are not conducting supervised training in this work, thus, the network was not instructed with labels regarding the instrument type or any other property of the target sound that could help in clustering the timbre families inside a class. The bandwidth of the output is related to that of the class used for training with, e.g.,brass instruments having a wide spectrum and flutes having a reduced bandwidth when producing an output with the same input. We also noted that the autoencoder trained on vocal samples produces tones with a voice-like texture.

Pitch Tracking Accuracy

We conducted more systematic tests to quantify the pitch tracking accuracy of the architecture. These tests were conducted with Dataset 2 because of the precise pitch of its content and its stability over time, allowing for an easier pitch accuracy evaluation. Five audio files were generated systematically from a MIDI file containing 20 random notes. Each audio file was generated from a different digital instruments using sampling synthesis and employing equal temperament and 440.0 Hz tuning. Using the 5 files as input to the 10 networks, resulted in 50 outputs for a total of 1000 notes.

Pitch accuracy was evaluated by employing a peak picking algorithm in the frequency domain [2], using a thresholded parabolically-interpolated STFT[2]. For each note, the accuracy was evaluated considering 128 pitch classes (those defined by the MIDI protocol), plus one unpitched class (i.e.,the output shows no clear pitch information, despite the input content which was always pitched). A pitch tolerance of ± 50 cents and an octave tolerance of ± 1 were allowed. Only 12% of the 1000 notes lost the original pitch, showing a good reliability of the network in retaining the original pitch of the input.

[2]https://librosa.github.io/librosa/generated/librosa.core.piptrack.html

Discussion

Overall, some properties of the proposed method are summarized.

- The output pitch follows quite closely the pitch of the input signal if the training set contains pitched audio in the same range as the input, however, the network is not able to generalize above or below the pitch range learned from the target file.
- For multi-pitched inputs (e.g., containing chords) the network is usually able to generalize providing a harmonization similar to the input.
- The timbre of the output signal resembles that of the training set, if the latter is sufficiently homogeneous (e.g., solo instrument from a track or separate notes from a specific instrument).
- Unpitched frames in the input audio are mapped to unpitched frames in the output if the training set contains unpitched material.
- Spectral continuity has intermediate quality between *k*-nn approaches and whitening approaches, i.e., frames do not change abruptly, thanks to the input context and the LSTM layer, but may still have deviations on a larger time basis. Previous experiments without context and the LSTM layer resulted in an output subject to abrupt variations from frame to frame similar to mosaicing.
- The algorithm does not guarantee that the frame energy of the input is transferred to the output. If the training set does contain sufficient levels of dynamic to learn there is a higher chance that energy is preserved, but it is not always guaranteed. The shortcoming of this is the generation of outlier frames when the input is silent or of low energy; this can be addressed by applying the input frame energy to the output by conventional signal processing techniques, but some constraints may be applied to allow the network learning energy preservation.

The proposed architecture is also able to perform morphing in the frequency domain, according to equations 4-5.

6 Conclusion

This work discussed how machine learning can be applied to the synthesis of novel sounds from existing sources, i.e. how it can provide a new class of *transfer* algorithms that has different properties with respect to conventional algorithms for morphing, hybridization, etc. A neural autoencoder has been proposed to the task and tested with different settings. Salient properties are the ability to transfer the timbre of a training set to an input file, while preserving its pitch. Pitch preservation has been proved to be reasonably good with systematic tests, while timbre transfer still needs improvement, especially with a large corpus of input signals.

Compared to dictionary-based algorithms, such as k-nn matching on MFCC, it proves to be more robust in term of frame-to-frame coherence, because of its improved generalization properties and possibly more relevant frame mapping.

Nonetheless, the timbre transferring capabilities of the algorithm are not systematically evaluated because it is very sensitive to the features of its input and target signals and an evaluation framework is missing. More research work is required to make the network robust to changes in frame energy, spectral bandwidth, time decay and pitch range of the target and input audio, in order to increase the intelligibility and the quality of the output signal. Furthermore, the network must be able to generalise for pitch ranges not learned from the target signal, and must be more expressive in order to learn different timbres in a large dataset.

Future works will be devoted to obtain improved results by employing established convolutional techniques. Recent trends in end-to-end convolutional neural networks may bring improved timbre learning at an acceptable computational cost [19, 20]. More research effort is needed and, arguably, quantitative evaluation of the results to this extent is required. Timbre classification, following state of the art algorithms (see, e.g., [21]) may be regarded as a logical choice, avoiding the arbitrary selection of metrics, distances or perceptually-motivated measures.

Acknowledgments

We acknowledge the CINECA award under the ISCRA initiative, for the availability of high performance computing resources and support.

References

[1] Caetano, M. and Osaka, N., "A formal evaluation framework for sound morphing," in *International Computer Music Conference*, 2012.

[2] Smith, J. O., *Spectral Audio Signal Processing*, Stanford University, CCRMA, 2010, (online book, last viewed 9/3/2011).

[3] Zölzer, U., *DAFX-Digital Audio Effects*, John Wiley and Sons, 2011.

[4] Gabrielli, L. and Squartini, S., "Ibrida: A New DWT-Domain Sound Hybridization Tool," in *45th AES International Conference*, Audio Engineering Society, 2012.

[5] Mallat, S., "Understanding deep convolutional networks," *Phil. Trans. R. Soc. A*, 374(2065), p. 20150203, 2016.

[6] Lostanlen, V. and Cella, C.-E., "Deep convolutional networks on the pitch spiral for musical instrument recognition," *arXiv preprint arXiv:1605.06644*, 2016.

[7] Cella, C. E., *On symbolic representations of music*, Ph.D. thesis, Università degli Studi di Bologna, 2011.

[8] Gatys, L. A., Ecker, A. S., and Bethge, M., "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414–2423, 2016.

[9] Foote, D., Yang, D., and Rohaninejad, M., "Do Androids Dream of Electric Beats?" Available online: `https://audiostyletransfer.wordpress.com/`, 2016, accessed on 24 October 2017.

[10] Ulyanov, D. and Lebedev, V., "Audio texture synthesis and style transfer," Available online: `https://dmitryulyanov.github.io/audio-texture-synthesis-\and-style-transfer/`, 2016, accessed on 24 October 2017.

[11] Geng, S., *Music Style Classification And Transformation Using Convolutional Neural Network*, Ph.D. thesis, University of Miami, 2016.

[12] Engel, J., Resnick, C., Roberts, A., Dieleman, S., Eck, D., Simonyan, K., and Norouzi, M., "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders," *arXiv*, 2017.

[13] Goodfellow, I., Bengio, Y., and Courville, A., "Autoencoders," in *Deep Learning*, chapter 14, pp. 502–525, MIT Press, 2016, `http://www.deeplearningbook.org`.

[14] Principi, E., Vesperini, F., Squartini, S., and Piazza, F., "Acoustic Novelty Detection with Adversarial Autoencoders," in *Proc. of the International Joint Conference on Neural Networks (IJCNN)*, pp. 3324–3330, Anchorage, AK, USA, 2017.

[15] Hinton, G. E. and Salakhutdinov, R. R., "Reducing the dimensionality of data with neural networks," *Science*, 313(5786), pp. 504–507, 2006.

[16] Lu, X., Tsao, Y., Matsuda, S., and Hori, C., "Speech enhancement based on deep denoising autoencoder." in *Proc. of Interspeech*, pp. 436–440, Lyon, France, 2013.

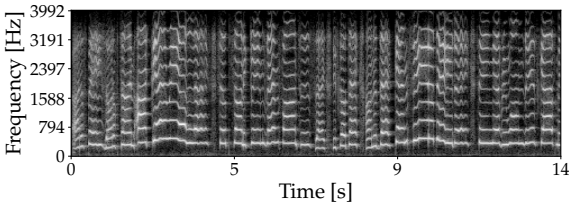
[17] Cella, C. E. and Burred, J. J., "Advanced Sound Hybridizations by Means of the Theory of SoundTypes," in *International Computer Music Conference*, 2013.

[18] Burred, J. J., "A framework for music analysis/resynthesis based on matrix factorization," in *International Computer Music Conference*, 2014.

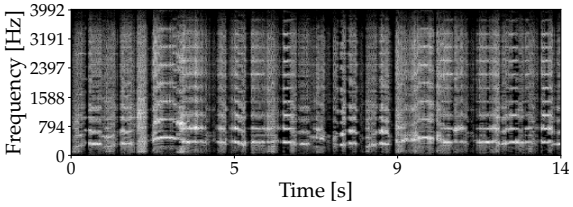
[19] Pons, J. and Serra, X., "Designing efficient architectures for modeling temporal features with convolutional neural networks," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2017.

[20] Lee, J., Park, J., Kim, K. L., and Nam, J., "Sample-level Deep Convolutional Neural Networks for Music Auto-tagging Using Raw Waveforms," *arXiv preprint arXiv:1703.01789*, 2017.

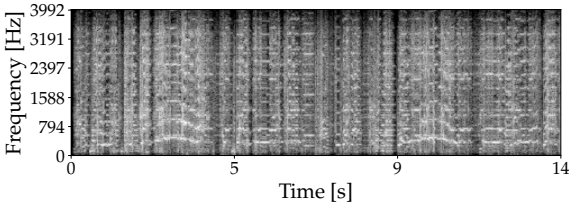
[21] Pons, J., Slizovskaia, O., Gong, R., Gómez, E., and Serra, X., "Timbre Analysis of Music Audio Signals with Convolutional Neural Networks," *25th European Signal Processing Conference, EUSIPCO*, 2017.



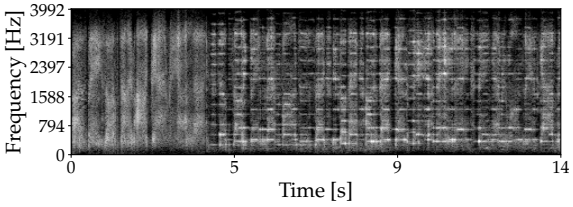
(a) Female voice FV1 (input).



(b) The proposed approach.

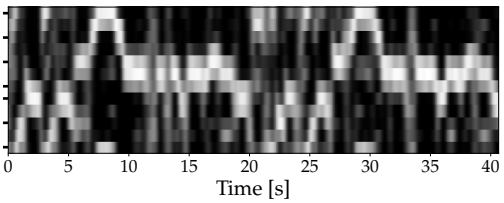


(c) k-nn matching techinque.

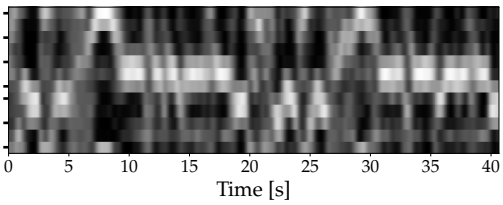


(d) Spectral flattening technique.

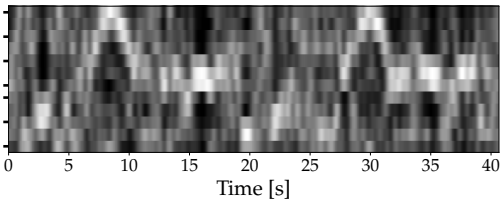
Fig. 3: Spectrograms from the input female voice FV1 (a), the proposed approach and the comparative methods(c-d). The hyperparameters for (b) are reported in Table 1.



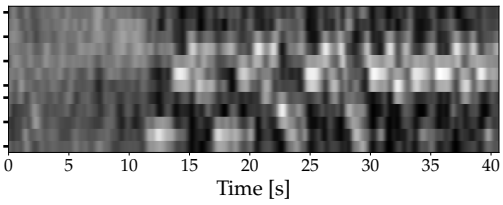
(a) Female voice FV1 (input).



(b) The proposed approach.



(c) k-nn matching technique.



(d) Spectral flattening technique.

Fig. 4: Chromagrams from (a) FV1, (b) to (d) different timbre transfer approaches using a distorted guitar track as a target and FV1 as input. Vertical axis shows the 12 notes (ticks correspond to natural notes).

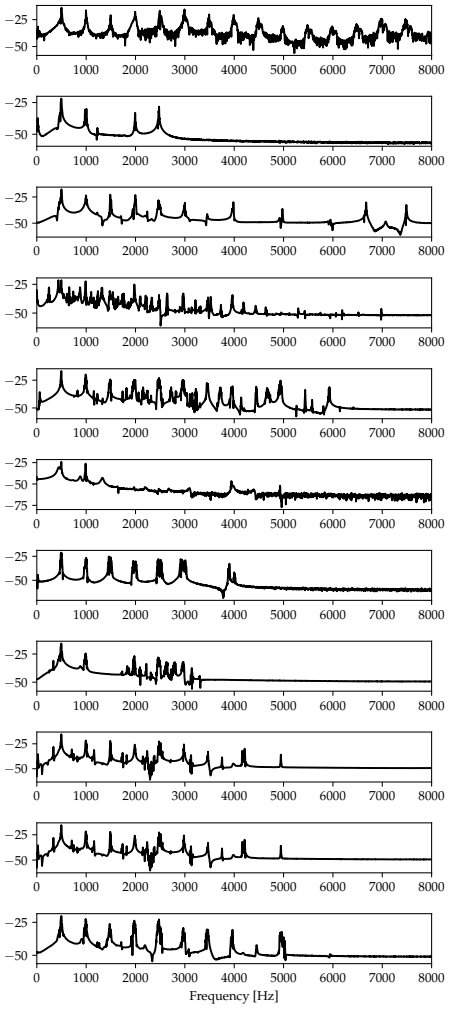


Fig. 5: Windowed DFTs taken from each instrument class output with FV1 as input, at the position where a pitched /i:/ phoneme takes places in FV1. Each DFT shows similar pitch, although timbres differ. Instrument classes, top to bottom: brass, flute, guitar, keyboard, mallet, organ, reed, string, synth-lead, vocal.