EXPLORATIONS OF SINGING VOICE SYNTHESIS USING DDSP

Juan ALONSO (jalons19@student.aau.dk) [1] and **Cumhur ERKUT** (cer@create.aau.dk) (0000-0003-0750-1919) [1]

[1] *Sound and Music Computing, Department of Architecture, Design, and Media Technology*, **Aalborg University**, A.C. Meyers Vænge, Copenhagen, DK-2450 Denmark

ABSTRACT

Machine learning based singing voice models require large datasets and lengthy training times. In this work we present a lightweight architecture, based on the Differentiable Digital Signal Processing (DDSP) library, that is able to output song-like utterances conditioned only on pitch and amplitude, after twelve hours of training using small datasets of unprocessed audio. The results are promising, as both the melody and the singer's voice are recognizable. In addition, we explore the unused latent-z vector in DDSP to improve the lyrics. Furthermore, we present two zero-configuration tools to train new models, including our experimental models. Our results indicate that the latent-z improves both the identification of the singer as well as the comprehension of the lyrics.

1. INTRODUCTION

Human voice is one of the oldest musical instruments [1]. Before the Deep Learning era [2], high-quality singing synthesis was carried out either by the spectral models [3], based on perception, or the physical models [4], based on production and articulation. Combining the spectral models with deep learning, the Differentiable Digital Signal Processing [5] (DDSP) library by Google's Magenta team became a powerful toolkit for audio-related machine learning. The first published examples of DDSP were focused on timbre transfer from monophonic instruments.

In this paper we present the DDSP architecture and apply it to a more complex, expressive instrument: the human voice. We check the suitability of the DDSP for singing voice synthesis. By constructing small size databases, experimenting with the model parameters and configurations, and by training the resulting models only for about twelve hours, we obtain singing-like outputs, which clearly resemble to original singers / speakers. However, the lyrics are incomprehensible because we don't have a language model. When we condition the model on the MFCC of the original audio, the results improve, promising intelligibility. Our contribution also enhances the documentation of the library and provides two zero-configuration notebooks for experimentation.

This paper is organized as follows. In Sec 2, we introduce the context of neural singing synthesis. Next we provide a detailed look at the DDSP architecture for timbre transfer. In Sec. 4, we introduce our experiments together with the data sets and model configurations, and the improved results we have obtained by adding the latent-z vector. In the next section, we discuss our observations. We finally draw our conclusions and indicate areas of further research.

2. BACKGROUND

In neural singing synthesis, the Deep Neural Network (DNN) receives a list of pitches and the lyrics as input, and outputs a signal modeling a specific voice. It is a problem closely related to the speech synthesis, however more challenging because of more diverse sets of pitches and intensities, different vowel durations, and other attributes specific to singing.

Gómez et al. [6] revised many data-driven deep learning models for singing synthesis. A thoughtful remark in that paper is that the black-box characteristics of deep learning models make it very difficult to gain knowledge related to the acoustics and expressiveness of singing. Even if deep learning techniques are general and can learn from almost any arbitrary corpus, it is necessary to advocate for explainable models to break the black-box paradigm.

The DDSP library is a set of tools released by Google's Magenta team. DDSP is set to bring explainability and modularity in neural audio synthesis [5]. The idea behind DDSP is "*to combine the interpretable structure of classical DSP elements (such as filters, oscillators, reverberation, envelopes...) with the expressivity of deep learning.*" [1] To avoid a common misunderstanding, DDSP is not an architecture *per se*, but a set of signal-processing tools that can be incorporated into modern automatic differentiation software. Many examples of DDSP relate to singing input, therefore we provide an overview in this section.

Several papers extended the DDSP specifically for speech and singing synthesis [7–9]. In [7], the model is conditioned on the phoneme level using an Automatic Speech Recognition (ASR) system to extract the phonemes of the training set and use them as additional conditioning data. In [8], the authors synthesize spoken speech using the DDSP architecture with a model conditioned on mel-spectrograms, instead of using raw audio. The loss function is also adapted to use mel-spectrograms trying to mimic the way human perception works. In [9], the authors propose using MIDI data modified by an LSTM network, to obtain continuous pitch and loudness contours that will be fed into the DDSP architecture.

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

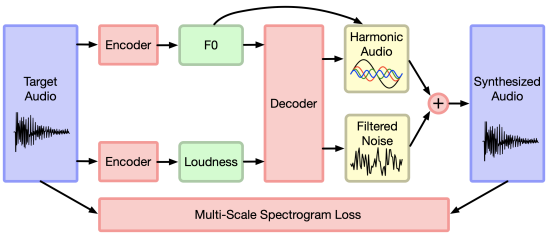


Figure 1. Timbre transfer architecture used in this work. Adapted from [5].

2.1 DDSP Overview

Phase alignment poses a problem when generating audio in deep learning; that problem is present when frames are used, either in the time- or the frequency-domain. An autoregressive model does not present this problem, but instead is harder to train due to the amount of data needed, and due to the interplay between audio perception, wave shape and loss. In fact, two wave shapes with very different losses can be perceived as sounding exactly the same.

One possible solution is the use of audio oscillators or vocoders that perform analysis and synthesis. In analysis, interpretable parameters such as f0 or loudness are extracted, and in synthesis the generative algorithm uses these parameters to construct the synthetic sound. DDSP disentangles them by a series of classical Digital Signal Processing (DSP) modules as feed-forward functions allowing, efficient implementation on GPUs.

3. DDSP AND TIMBRE TRANSFER

The architecture used in this work is based on the "solo_instrument" setup proposed for timbre transfer in [5] and shown in Fig 1. The network is presented an audio file. f0 and loudness are extracted and fed to the decoder, which produces the parameters for a harmonic synthesizer and for a subtractive synthesizer. The audio from both synthesizers is combined and presented as the final output. During the training phase, the loss is computed using different resolution spectrograms from the original and the resulting signal. In the following, we describe the modules used.

The **encoder** transforms the incoming audio into latent vectors, in this case, two interpretable features: the fundamental frequency and the perceived loudness of the monophonic input audio. The DDSP library does not require a specific method to generate the **f0** and **loudness** vector, but expects arrays of float values, sampled at 250 Hz.

f0 can be generated synthetically, but the DDSP library includes examples using CREPE [10] and DDSP-inv [11]. The f0 latent vector is fed directly to the additive synthesizer. This allows to disentangle the fundamental frequency and facilitates the model to respond to frequencies unseen during the training.

Loudness can also be generated synthetically, but the DDSP library includes a utility function to compute the perceptual loudness of the audio, applying A-weighting curves to the power spectrogram.

The **decoder** (Figure 2, top) is an RNN that receives the latent vectors (f0 and loudness) and outputs the control parameters required by the synthesizers: the amplitudes of the harmonics, and the transfer function for the FIR filter. The RNN is fairly generic, as the DDSP authors emphasize that the quality of the results comes from the DDSP modules, not from the complexity of the neural network.

The latent vectors are preprocessed by a Multilayer Perceptron (MLP) (Fig. 2, bottom), which comprises a block of three layers (Dense, Normalization and ReLU layers) repeated three times. The output of the MLP is connected to a 512-cell GRU layer which is connected to a second MLP with the same structure and finally passed to two dense layers that will provide the parameters for the synthesizers.

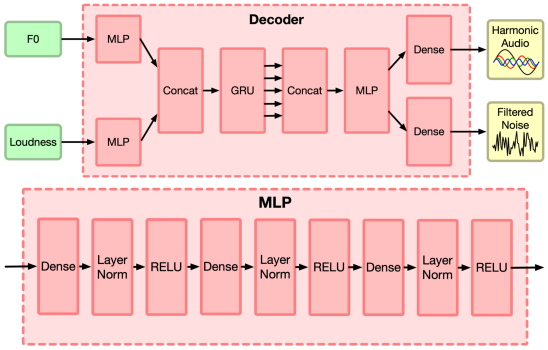


Figure 2. Decoder architecture (top) and MLP structure (bottom). Adapted from [5].

To synthesize audio, the DDSP library uses **Spectral Modelling Synthesis** (SMS), a technique proposed by Serra and Smith [3], where the sound is modeled as two components: an additive or harmonic synthesizer, where a series of sinusoids is combined, and a subtractive synthesizer where the residual component is modeled as filtered noise. The DDSP library implements a differentiable version of the SMS, with an additional constraint: all the frequencies are integer multiples of f0. The expressiveness of this technique is a consequence of the high number of parameters needed. With the default configuration (60 harmonics, 65 noise magnitudes and 1 amplitude), sampled at 250Hz, 1 second of audio yields (60+65+1)*250 = 31,500 dimensions vs 16,000 audio samples.

The **additive (harmonic) synthesizer** models the harmonic part of the signal $x(n)$ as a sum of K sinusoids, with amplitudes $A_k(n)$ and instantaneous phases $\phi_k(n)$.

$$x(n) = \sum_{k=1}^{K} A_k(n) \sin(\phi_k(n)). \quad (1)$$

We can further expand $A_k(n)$ into a global amplitude $A(n)$ and a normalized distribution $c(n)$ of amplitudes for the harmonic components, so that $A_k(n) = A(n)c_k(n)$. Then Equation (1) can be rewritten as

$$x(n) = A(n) \sum_{k=1}^{K} c_k(n) \sin(\phi_k(n)), \quad (2)$$

The instantaneous phase is described as

$$\phi_k(n) = 2\pi \sum_{m=0}^{n} k f_0(m) + \phi_{0,k} \quad (3)$$

where $\phi_{0,k}$ is the initial phase for the harmonic k.

To reconstruct the additive signal at audio sample rate, f0 is upsampled using bilinear interpolation, and the amplitude and harmonic distribution are smoothed using an overlapping Hamming window centered on each frame.

The **subtractive synthesizer** models the residual component, the difference between the original signal and the signal from the additive synthesizer. If we assume that the residual component is stochastic, it can be modeled as filtered white noise, using a time-varying filter. The filter is applied in the frequency-domain, to avoid phase distortion. For every frame l, the network outputs H_l, the frequency-domain transfer function of the FIR filter. The convolution is applied as a multiplication in the frequency-domain (Eq. 4), and then an Inverse Discrete Fourier Transform (IDFT) is applied to recover the filtered signal (Eq. 5):

$$Y_l = H_l DFT(x_l), \quad (4)$$

$$y_l = IDFT(Y_l). \quad (5)$$

Training the autoencoder means finding a set of parameters for the synthesizers that minimize the reconstruction loss i.e., minimize the difference between the output and input signals. A sample-wise loss measure is not recommended, as two similar waveforms can be perceived as having a very different sound. The spectral loss L –the difference between the spectrograms of the input (S) and output ($\hat{S}$) signals– is perceptually better, but there is a compromise between time and frequency.

To solve this problem, a multi-scale spectral loss is defined. Instead of using a single pair of spectrograms $(S, \hat{S})$, the loss is defined as the sum of different spectral losses L_i where i is the FFT size. Moreover, the linear magnitude losses are sensitive to the peaks, whereas logarithmic magnitude losses are sensitive to the quiet regions of the signals. Therefore the loss is defined as $L = \sum_i L_i$, with $i \in (2048, 1024, 512, 256, 128, 64)$, where

$$L_i = ||S_i - \hat{S}_i||_1 + ||log(S_i) - log(\hat{S}_i)||_1 \quad (6)$$

4. EXPERIMENTS AND RESULTS

Our first experiment [2] is a simple test to check if the system is correctly set up. The second experiment tries to determine if the model is able to learn single from single- and multiple-voice datasets. The third experiment explores the effect of training the model using the same dataset, with different sets of parameters for the spectral synthesizer. The last experiment introduces a encoded representation of the Mel Frequency Cepstrum Coefficients. Tables 1, 2 and 3 describe the datasets we used.

[2] Our notebooks and configuration files are available at `https://github.com/juanalonso/DDSP-singing-experiments`. Example results (audio and spectrograms) are at `https://juanalonso.github.io/DDSP-singing-experiments/`.

Name	Speaker / Singer	Gender	Type	Language
alba	Alba	Female	Spoken	Spa
mrallsop	Scott Allsop	Male	Spoken	Eng
eva	Eva Páez	Female	Sung	Eng, Spa
belen	Belén Chanes	Female	Sung	Spa
servando	Servando Carballar	Male	Sung	Spa
voices2	belen + eva	Female	Sung	Eng, Spa
voices3	belen + eva + servando	Mixed	Sung	Eng, Spa
felipe_vi	King Felipe VI	Male	Spoken	Spa

Table 1. Dataset metadata.

Name	Source
alba	Scrapped from `https://albalearning.com/`
mrallsop	Scrapped from `https://tinyurl.com/mrallsop`
eva	Provided by singer
belen	Provided by singer
servando	Provided by singer
voices2	Provided by singers
voices3	Provided by singers
felipe_vi	Scrapped from `https://tinyurl.com/felipe-vi`

Table 2. Dataset sources.

To test the model, we used a fragment of 'Over the rainbow' as sung by Lamtharn (Hanoi) Hantrakul as the original audio presented to the model. This melody is also used in the DDSP [5] examples.

4.1 Pre-evaluation

A quick test of the system has been carried out to reproduce the timbre transfer experiments while checking the validity of our setup. In this test we have used the same material as in the original paper [3] , generating the dataset and training the model for 30k steps using the standard configuration file `solo_instrument.gin` (60 sine waves, 65 noise magnitudes, reverb and batch size of 16).

The estimated f0 is transposed up two octaves, to better match the pitch range of the violin, and the loudness is attenuated 20dB. Two reconstructions are produced, the first one with no additional information, and the second one using masking and statistical amplitude adjustment. These reconstructions are shown in Fig. 3.

[3] Movements II, III, IV, VI and VIII from the Bach Violin Partita no.1 BWM 1002, as played by John Garner at `https://musopen.org/music/13574-violin-partita-no-1-bwv-1002/`

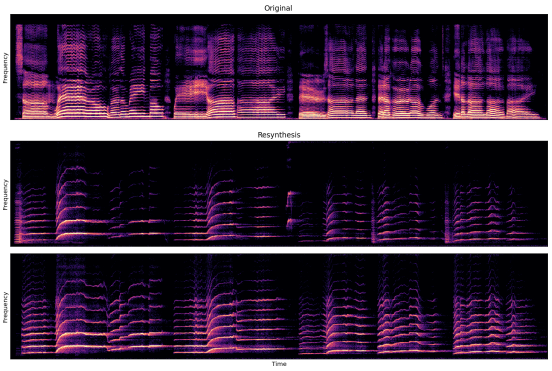


Figure 3. Over the rainbow as sung by Lamtharn (Hanoi) Hantrakul (top) and reconstructed by our version of the violin model (+2oct, -20dB), using no statistical information (middle) and with statistical information (bottom).

Name	Avg. pitch	Duration	Active audio
alba	53.76	1:24:29	0:53:28
mrallsop	48.42	0:25:41	0:23:51
eva	64.10	0:18:34	0:16:23
belen	62.39	0:10:16	0:06:24
servando	55.08	0:11:02	0:08:31
voices2	63.59	0:28:50	0:22:47
voices3	61.75	0:39:52	0:31:17
felipe_vi	50.93	0:13:32	0:08:35

Table 3. Audio properties of the datasets. Active audio is the duration after removing parts of the original file where audio is lower than -52dB for more than 200ms. This is only a reference value and it is not used for training.

The violin model works as expected. The resulting audio is equivalent to the audio produced by the original model, which is trained for 20k extra steps with a batch size of 32.

4.2 Different datasets trained on the same parameters

4.2.1 Single voice model

To generate the models for singing voices, we are using seven datasets (see Table 1), obtained from speakers and singers, both male and female, in different languages. No audio source separation software is used. The datasets eva, belen and servando are raw vocal tracks recorded by professional singers and have been kindly provided by the performers upon request of the authors. These tracks were not specifically recorded for this work, they were previously recorded for existing and upcoming music records. Two additional datasets have been created by combining the audio files from the eva and belen datasets (voices2) and the eva, belen and servando datasets (voices3). The rest of the datasets (mrallsop, alba and felipe_vi) has been scrapped form the web. Files longer than five minutes have been split into three to four-minute chunks. Other than that, the original audio has not been transformed in any way, keeping all the original characteristics: silences, different volume levels, background noise, etc.

The models are trained for 40k steps each, using the notebook `01_train`. Each model has been trained using `singing.gin`, a modified version of the standard configuration file. The losses after training (Fig. 4, top) are all in the range [4.987, 5.437] as recommended in [5]. There are no significant differences between the losses in the spoken and sung datasets. The servando model, whose loss is considerably higher, is an exception. The only apparent difference with the rest of datasets is that servando's source audio is hard-clipped / compressed at 0dB, with a smaller dynamic range than the other voices, which present a wider range of amplitudes and compression values.

Name	Gender	Type	Loss
servando	male	sung lyrics	6.984
belen	female	sung lyrics	5.114
eva	female	sung lyrics	4.987
mrallsop	male	spoken speech	5.437
alba	female	spoken speech	5.142
voices2	female	sung lyrics	5.415
voices3	mixed	sung lyrics	6.143

Table 4. List of datasets used for training the model and loss value after 40k steps.

The output is generated by executing notebook `02_run`. The pitch shift is chosen manually by comparing the mean MIDI pitch of the dataset with the mean MIDI pitch of the melody. The loudness shift is handpicked after comparing different settings. The threshold and quiet parameters are adjusted manually depending on how much noise bleeds into the silent parts of the original audio. The values chosen for each example are shown in the audio web page.

4.2.2 Multiple voices model

The model **voices2** combines the source audio from belen and eva, both female singers. It is trained for 40k steps. The loss, after training is 5.415, higher than the loss of both of the datasets (belen=5.114, eva=4.987), as shown in Fig. 4, bottom. This result is expected, as we are training the model with the same number of steps as the single voice models but, in this case, the model needs to learn two different timbres, with different loudness and slightly different MIDI mean pitches (belen=62.39, eva=64.10).

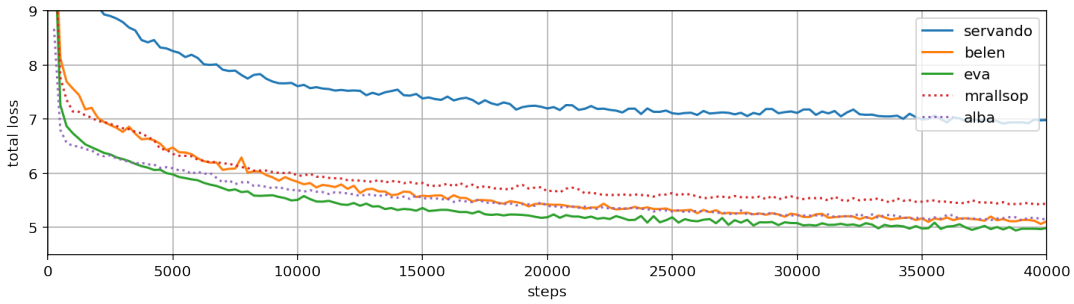
When the model is used, the resulting audio is a combination of the individual voices. Depending on the f0 and loudness, the model outputs a succession of fragments by the original singers. The transition is smooth: only one voice is perceived at a concrete frame.

The model **voices3** combines the source audio from belen, eva and servando. It is trained for 40k steps: the model must learn three timbres, and one of them is more different from the other two (servando, loss=6.984, MIDI mean pitch=55.08) The peculiarities of the servando dataset penalize the training, and thus the model presents a higher loss than voices2.

Fig. 4, bottom, shows that the loss of voices3 (6.143) is lower than the loss of the servando model (6.984). We attribute this effect to an imbalanced dataset: the duration of servando's source audio is 11 minutes and 02 seconds, whereas belen's and eva's source audio combined is 28 minutes and 50 seconds.

In this experiment, the voice mixing capabilities of the model are more pronounced than in the previous experiment. The mean MIDI pitch of the example song is 51.30. Considering that the mean MIDI pitches of the servando, belen and eva datasets on Table 4 are, respectively 55.08, 62.39 and 64.10, we can expect that when rendering the audio, the model will generate a mix of the nearest-pitched voices. This is the case: when using the example fragment without transposing, the resulting melody is a mix of servando's and belen's voice. If the example fragment is transposed an octave higher (MIDI pitch of 63.30) the resulting melody is a mix of belen's and eva's voice. To demonstrate this effect, six examples have been uploaded to the audio page, using different sets of preprocessing parameters.

(a) Five independent datasets: dotted lines represent speech datasets.



(b) Hybrid dataset, with voices2 and voices3.

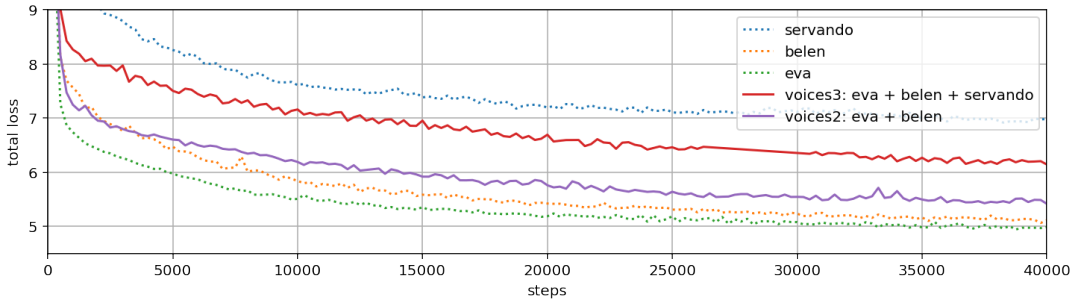


Figure 4. Spectral loss for the different datasets

Use statistics	Yes
Mask threshold	1
Quiet	20
Autotune	0
Octave shift	+1
Loudness shift	-10dB

Table 5. f0 and loudness preprocessing parameters.

4.3 Single dataset, different spectral parameters

In this experiment, we want to understand how the parameters of the spectral synthesizer affect the learning process and the results. We will be using the eva dataset –since its model has the lowest loss after training– to train the model using a range of spectral configurations.

We chose three values for the harmonic component (20, 60 and 100 sinusoidal waves). 60 is the default value provided in the configuration files. 100 is the maximum value a model can be trained without getting out of memory errors, and 20 is the symmetrical lower bound ($60 - (100 - 60)$). For the noise component we chose 10, 35 and 65 noise magnitudes, 65 being the default value.

Nine models are generated, one for each combination of harmonic components and noise magnitudes. Each model is trained for 20k steps, using the same configuration file we used in the previous sections (`singing.gin`). f0 and loudness are preprocessed using the handpicked parameters from Section 4.2.1 and shown in Table 5.

As can be observed in Fig. 5, to decrease the spectral loss, the amount of noise magnitudes is more relevant than the number of harmonic components. Perceptually, more models and tests are needed. On an informal test where three users with no musical training were presented pairs of the snippet 'way up high' (seconds 7 to 11 in the original audio) rendered with different parameters, there was no agreement on which "sounded better". The only exception was snippet h:20 n:10, where the subjects remarked it was the worst sounding of the pair. All subjects commented on listening fatigue due to being exposed to the same melody.

Observing the spectrograms of the reconstructions in Fig. 6, the examples with the lowest number of noise magnitudes ($n = 10$) show the model trying to reconstruct the high frequency noise with the harmonic model (faint sinusoids in the spectrograms).

4.4 Adding the latent-z

For our last experiment, we expand the `singing.gin` configuration (Fig. 7) to include a time-varying encoding of the Mel Frequency Cepstral Coefficients (MFCC) to understand how this additional encoding affects the model's output. This representation is specially well suited for the human voice, as it mimics the non-linearity of human sound perception and helps model the timbre [12]. The MFCC are encoded using a normalization layer and a 512-unit GRU, reducing the dimensionality from 30 coefficients to a 16-dimensional latent vector, at 125 frames per second and then upsampled to 250 frames, to match the sampling rate of the f0 and loudness vectors. To decode this vector, the same MLP architecture shown in Fig. 2, bottom, is used, and concatenated with the preprocessed f0 and loudness vectors.

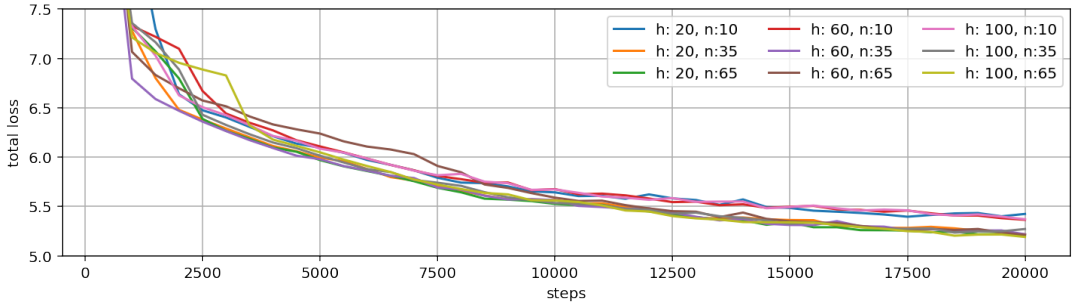


Figure 5. Total (spectral) loss for the eva dataset, using different parameters for the spectral synthesizer. h is the number of harmonic components; n is the number of noise magnitudes.

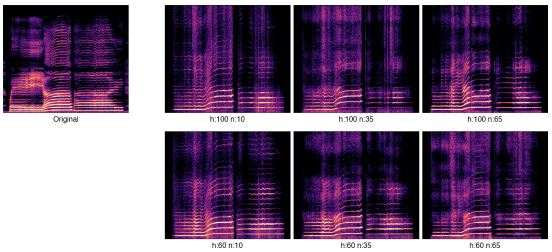


Figure 6. Spectrogram of the original audio ('way up high', seconds 7-11). left column, and spectrograms of the model's output with different spectral parameters.

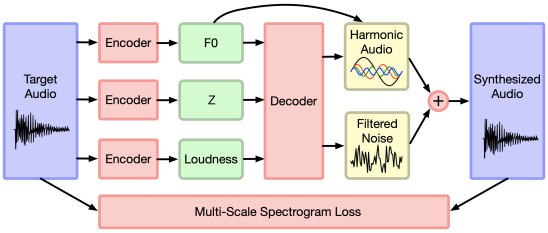


Figure 7. Timbre transfer architecture with latent-z. Adapted from [5].

To generate the z models, we used the belen and mrallsop datasets and a new dataset, felipe_vi, extracted from Felipe VI's, King of Spain, 2020 Christmas speech (male spoken voice, Spanish) As with the previous datasets, felipe_vi is used "as is", without any kind of preprocessing. A new configuration file (`singing_z.gin`) is also used, and it is available at the GitHub repository. This configuration file inherits all the values from `singing.gin` and includes the z encoder.

The models are trained for 25k steps. As shown in Figure 8, with the additional encoding, the loss function increases (from 5.310 to 6.322 in the case of the belen dataset and from 5.634 to 8.011 in the mrallsop dataset) This is an expected behavior, as the model now needs to fit additional parameters. The loss value for the felipe_vi_z model is low enough (5.514) to be in the range of the non-z models.

5. RESULTS AND DISCUSSION

The architecture proposed in [5] is powerful enough to perform timbre transfer and to produce surprising results even if the dataset is small (10-15 minutes of audio). The training, using the Colab environment, is fast and it works out of the box, with the GPU infrastructure ready. All the required libraries (CREPE, TensorFlow, Apache Beam. etc.) are available without version conflicts.

In singing voice synthesis, we challenged the model (small datasets, unprocessed audio, etc.), but the quality of the results surprised us. Of course, in no way the output of the model is going to be mistaken for a human, but the model's ability to produce such good results with no additional conditioning is very promising and opens several avenues for exploration, research and creative usage.

With the addition of the z-encoder, the quality of produced audio is increased, becoming almost intelligible. The right vowels start appearing, and the babbling effect is reduced substantially. The resulting timbre is halfway between the instrument timbre and the original timbre.

This architecture makes very difficult to estimate the performance of a model. As we have noticed, the training loss of the servando model is quite high, compared with the rest, but when analyzing the dataset, nothing stands out as the cause of this value. Similar datasets (speaking, male voice) such as felipe_vi_z and mrallsop_z present very different losses (5.514 versus 8.011 respectively), but the quality of the resulting audio is comparable.

5.1 Problems

5.1.1 Babbling and stuttering

We are forcing the model to recreate sung lyrics. The model needs to learn the timbre of the voice, how to extrapolate previously unheard pitches and the flow of the language. The current architecture manages to extract the timbre and to correlate f0 and loudness with sounds, but it lacks the ability to learn the sequences of phonemes that constitute speech. Even with more comprehensive datasets, where all the possible combinations of phonemes and pitches could be present, without additional conditioning (phonetic, text, etc.) the model will try to make up what to say, and the produced audio will be just a better-quality

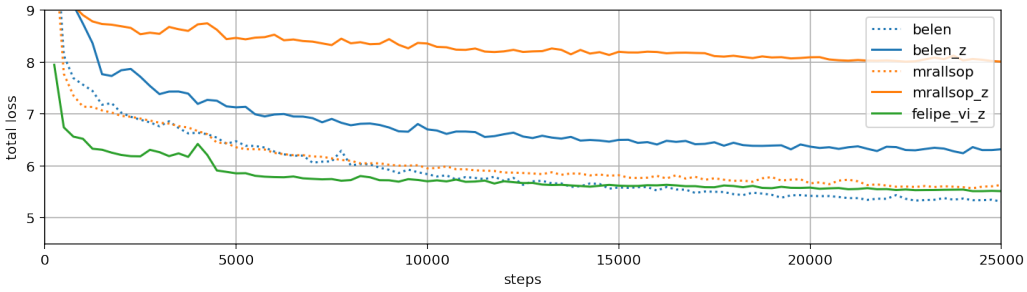


Figure 8. Spectral loss for the three z models. The dotted lines are included for reference, and represent loss function for the two related non-z models.

version of this stuttering and nonsensical babbling.

During the development of this work, the audio has been presented to several listeners who knew the original singers (belen, servando) and they all found the results unsettling, due to this kind of babbling. They recognized the speaker, but the babbling and stuttering were compared to listening to a person having suffered a stroke that impaired the language centers of the brain.

5.1.2 Silence

If the dataset does not include silences (for example, the dataset used to train the violin model) the resulting model has difficulties trying to recreate them and will resort to generate some very low notes. This can be mitigated by adding some transitions to and from silence and by fine tuning the preprocessing parameters, which right now it is a manual process dependent on the input audio and the model. The datasets used in this work do not present this problem, since the original material includes pauses and therefore, the network can recreate possible silences.

The example on the audio page shows this phenomenon particularly well. On the one hand, the original audio, a staccato synthesizer riff, is played as legato. On the other hand, the silence that occurs at seconds 3 and 9 is reinterpreted as a pair of low-pitched tones. Even tweaking the preprocessing parameters, we can mitigate the low tones, but not suppress them.

5.1.3 Pitch artifacts

The accuracy of the output pitches depends on the f0 estimation. If the estimation made by the CREPE model is wrong, the output will include wrong notes. We have detected several cases where CREPE estimates a very high pitch with high confidence, so the preprocessor cannot mask it. In those cases, the resulting audio will include a squeal, a quick glissando to the estimated high pitch.

To avoid this, we can substitute CREPE for another algorithm, or use a symbolic notation, such as MIDI, to generate the f0 vector. In that case, we risk having a monotonous voice, and we would need to add some modulation (e.g., amplitude or frequency modulation) to make it more natural sounding.

5.1.4 DDSP maturity

Although the possibilities of these libraries are immense, exploring them is a challenging task for two major reasons. The first one is the lack of information about the most complex processes, how some of its modules work and how to modify the workflow to add new functionalities. Despite open-sourcing the code in GitHub, the tutorials and demos barely scratch the surface. The second problem is that, because the libraries are still under active development, some of the updates are missing information about the release changes and are not as stable as expected. These, however are expected in any open source library.

6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The DDSP library opens up a lot of possibilities for audio synthesis. The work presented here allows us to get a better understanding on how the DDSP library works, especially when used for timbre transfer. It achieves two goals:

1. **Test the validity of the DDSP architecture to generate a singing voice.** The tests carried out on the architecture have used unfavorable data (no preprocessing, background noises, small datasets, etc.), and even so, the network could generate audio similar to the human voice, with enough features to be recognized as belonging to a specific singer.

2. **Create an easy-to-use environment to facilitate model training and timbre transfer to end users**. The notebooks provided will help the SMC community to ease the learning curve of this architecture and get familiar with the advantages and nuisances of the library. Since the GitHub repository includes some of the models used in this work, curious users can just interact with them, without needing to create their own datasets. Also, as per today (March 2021) our models are compatible with the ones made available by Google (flute, violin...) so they can be used interchangeably.

6.2 Future Work

In order to create a more refined model which is capable of synthesizing much realistic utterances with lyrics replication (and thus avoiding the gibberish / stuttering effect) additional work must be done in the following areas:

Conditioning: As noted in Section 5.1.1, the phonetic output is made up by the model, without any reference to real speech. The current architecture does not condition the output in any other data than pitch and loudness, missing additional information present in sung lyrics. To get the nuances of human singing and model the lyrics, we need to include additional conditioning on the language level, for example, the phonetic conditioning proposed in [7].

Use representations more suitable to voice synthesis: The default architecture proposed in the DDSP is generic, designed for monophonic musical instruments. Using mel-spectrograms as proposed in [8], instead of using raw audio or by postprocessing the harmonic and the noise components of the transformed audio to balance the voiced and unvoiced parts of the speech [7], results could be improved.

Use synthesizers more suitable to voice modeling: As stated previously, by using Spectral Modelling Synthesis, we get a very expressive synthesizer at the expense of producing twice as much data per second as the sampled audio. However, other synthesizers can provide a more compact representation, resulting in a smaller model which will be faster to train and run. The authors are currently working on implementing both an AM and a 2-operator FM differentiable synthesizer. These simple synthesizers will provide us a better understanding of the capabilities and nuances of a differentiable synth, its performance, how to integrate them in the existing toolchain, and how to modify the model architecture to fit different synthesizers.

Preprocessing of f0: Even if the model is able to transfer the timbre perfectly, following the "Garbage in, garbage out" concept, the quality of the output will be affected by the quality of the latent vectors. If the pitch estimation is not accurate, the resulting audio will present pitch artifacts. A quick solution can be to extract f0 from MIDI. While the resulting f0 is going to be precise, it is going to lack expressiveness. Solutions as the one proposed in [9] can add expressiveness to the MIDI data.

Explore the creative possibilities of the model: The creative possibilities offered by the DDSP architecture are immense, either with low fidelity, glitchy results as explored in this work, or with more realistic results by applying additional conditioning. Some of the possibilities are pitch- and time-shifting, lyric translation, voice morphing, change of singing style (e.g., to and from opera, pop, blues), tremolo and vibrato removal or addition, to name just a few. Working with clean data and synthesizing singing in different (or fictional) languages would be also interesting for the future.

Acknowledgments

The authors would like to thank the following singers for their generous collaboration providing the audio files: Servando Carballar from Aviador Dro (`https://www.instagram.com/el_aviador_dro/`), Belén Chanes from Lkan (`https://www.instagram.com/lkan_oficial/`) and Eva Páez (`http://evapaez.es/`).

7. REFERENCES

[1] J. Sundberg, "The singing voice," in *The Oxford handbook of voice perception*, S. Frühholz and P. Belin, Eds. Oxford University Press, 2018, ch. 6.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] X. Serra and J. Smith, "Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition," *Computer Music Journal*, vol. 14, no. 4, pp. 12–24, 1990.

[4] P. R. Cook, "Singing voice synthesis: History, current work, and future directions," *Computer Music J.*, vol. 2, no. 4, 1996.

[5] J. Engel, L. H. Hantrakul, C. Gu, and A. Roberts, "DDSP: Differentiable digital signal processing," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1x1ma4tDr

[6] E. Gómez, M. Blaauw, J. Bonada, P. Chandna, and H. Cuesta, "Deep learning for singing processing: Achievements, challenges and impact on singers and listeners," *arXiv preprint arXiv:1807.03046*, 2018.

[7] G.-M. Hutter, "Timbre transfer on singing voices," Master's thesis, ETH Zurich, 2020.

[8] G. Fabbro, V. Golkov, T. Kemp, and D. Cremers, "Speech synthesis and control using differentiable DSP," *arXiv preprint arXiv:2010.15084*, 2020.

[9] N. Jonason, B. Sturm, and C. Thomé, "The control-synthesis approach for making expressive and controllable neural music synthesizers," in *2020 AI Music Creativity Conference*, 2020.

[10] J. W. Kim, J. Salamon, P. Li, and J. P. Bello, "Crepe: A convolutional representation for pitch estimation," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 161–165.

[11] J. Engel, R. Swavely, L. H. Hantrakul, A. Roberts, and C. Hawthorne, "Self-supervised pitch detection with inverse audio synthesis," in *International Conference on Machine Learning, Self-supervised Audio and Speech Workshop*, 2020. [Online]. Available: https://openreview.net/forum?id=RlVTYWhsky7

[12] J. I. Godino-Llorente, P. Gomez-Vilda, and M. Blanco-Velasco, "Dimensionality reduction of a pathological voice quality assessment system based on gaussian mixture models and short-term cepstral parameters," *IEEE transactions on biomedical engineering*, vol. 53, no. 10, pp. 1943–1953, 2006.