Name: Alam Ali (NetID: aa8007)

Course Name: Machine Learning (ECE-GY-6123)

Fall 2020 Semester Project

Submission Date: 10-December-2020

Title: DDSP (Differentiable Digital Signal Processing)

Introduction:

This project introduces Differentiable Digital Signal Processing (DDSP) library, which enables direct integration of classic signal processing elements with deep learning methods. Focusing on audio synthesis, we achieve high-fidelity generation without the need for large autoregressive models or adversarial losses, demonstrating that DDSP enables utilizing strong inductive biases without losing the expressive power of neural networks. We show that combining interpretable modules permits manipulation of each separate model component, with applications such as independent control of pitch and loudness, realistic extrapolation to pitches not seen during training, blind dereverberation of room acoustics, transfer of extracted room acoustics to new environments, and transformation of timbre between disparate sources. DDSP enables an interpretable and modular approach to generative modeling, without sacrificing the benefits of deep learning.

Basics of DDSP:

DDSP is a library of differentiable versions of common DSP functions (such as synthesizers, waveshapers, and filters). This allows these interpretable elements to be used as part of a deep learning model, especially as the output layers for audio generation.

How to get started:

First, follow the steps in the Installation section to install the DDSP package and its dependencies. DDSP modules can be used to generate and manipulate audio from neural network outputs as in this simple example:

Code:

```
import ddsp

# Get synthesizer parameters from a neural network.

outputs = network(inputs)

# Initialize signal processors.
```

```
additive = ddsp.synths.Additive()

# Generates audio from additive synthesizer.

audio = additive(outputs['amplitudes'],

                outputs['harmonic_distribution'],

                outputs['f0_hz'])
```

Neural Networks:

Neural networks are universal function approximators in the asymptotic limit, but their practical success is largely due to the use of strong structural priors such as convolution, recurrence and self-attention.

We will demonstrate that relatively small models employing DDSP components are capable of generating high-fidelity audio without autoregressive or adversarial losses. Furthermore, we show the interpretability and modularity of these models which enable the following functions:

1) Independent control over pitch and loudness during synthesis,

2) Realistic extrapolation to pitches not seen during training,

3) Blind dereverberation of audio through separate modelling of room acoustics,

4) Transfer of extracted room acoustics to new environments,

5) Timbre transfer between disparate sources, converting a singing voice into a violin.

DDSP Components:

We express core components as feedforward functions, allowing efficient implementation on parallel hardware such as GPUs and TPUs, and generation of samples during training. These components include oscillators, envelopes, and filters (linear-time-varying finite-impulse-response LTV-FIR). We have implemented further components (as future work) such as wavetable synthesizers and non-sinusoidal oscillators.

1) Harmonic Oscillator

2) Envelopes

3) Filter Design using frequency sampling method

4) Filtered noise synthesizer

5) Reverbation using long impulse responses

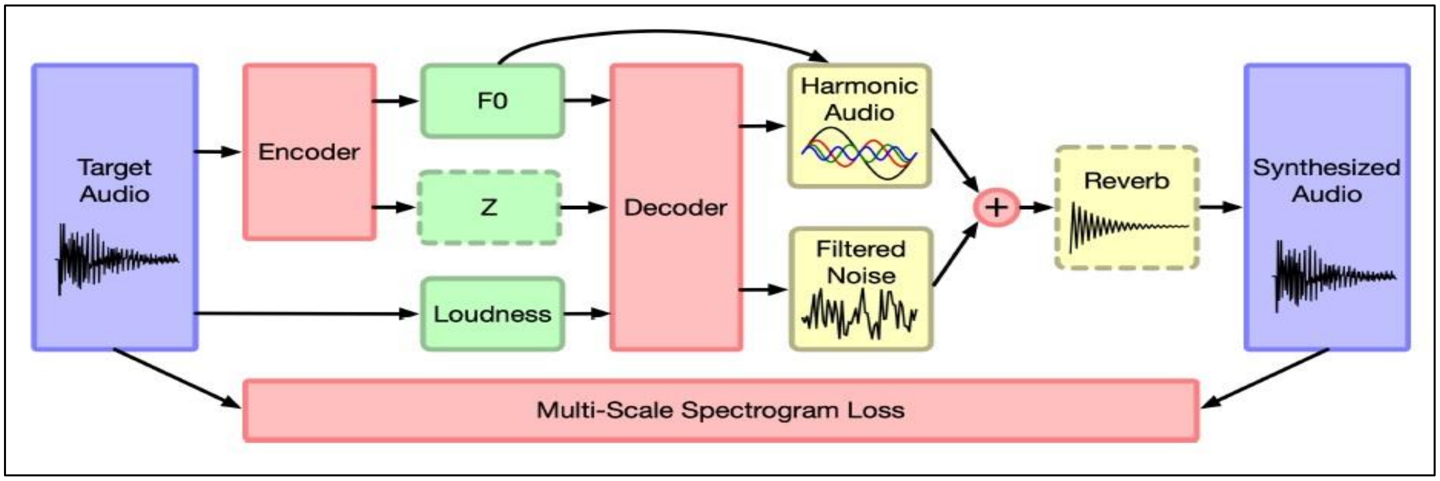


Fig 1: This figure shows an auto-encoder architecture. Red components are part of the neural network architecture, green components are the latent representation, and yellow components are deterministic synthesizers and effects.

Multi-scale Spectral Loss:

The primary objective of the auto-encoder is to minimize the reconstruction loss. However for audio waveforms, point-wise loss on the raw waveform is not ideal, as two perceptually identical audio samples may have distinct waveforms, and point-wise similar waveforms may sound very different. Instead, we use a multi-scale spectral loss which is similar to the multi-resolution spectral amplitude distance is defined as follows.

$$L_i = ||S_i - \hat{S}_i||_1 + ||\log S_i - \log \hat{S}_i||_1$$
$$L_{\text{reconstruction}} = \sum_i L_i$$

Spectral Loss is defined as the sum of the L1 difference between Si and Si(hat) and the L1 difference between log Si and log Si(hat). The loss (i) given by L (for all i) covers differences between the original and synthesized audios at different spatial-temporal resolutions.

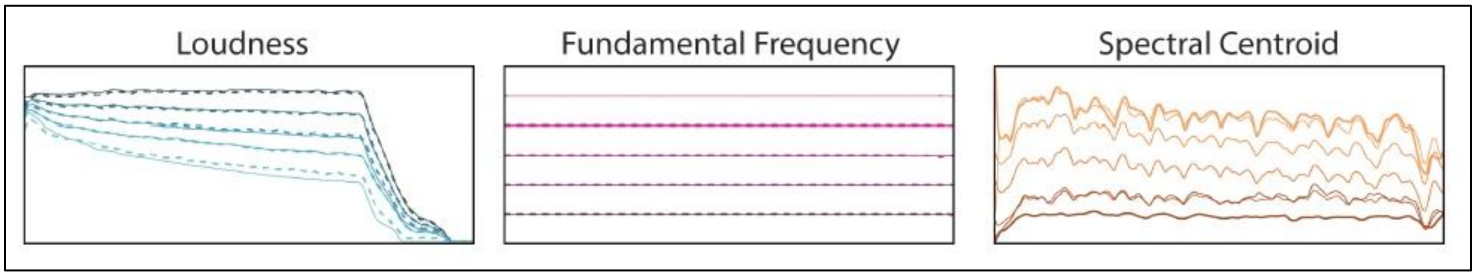
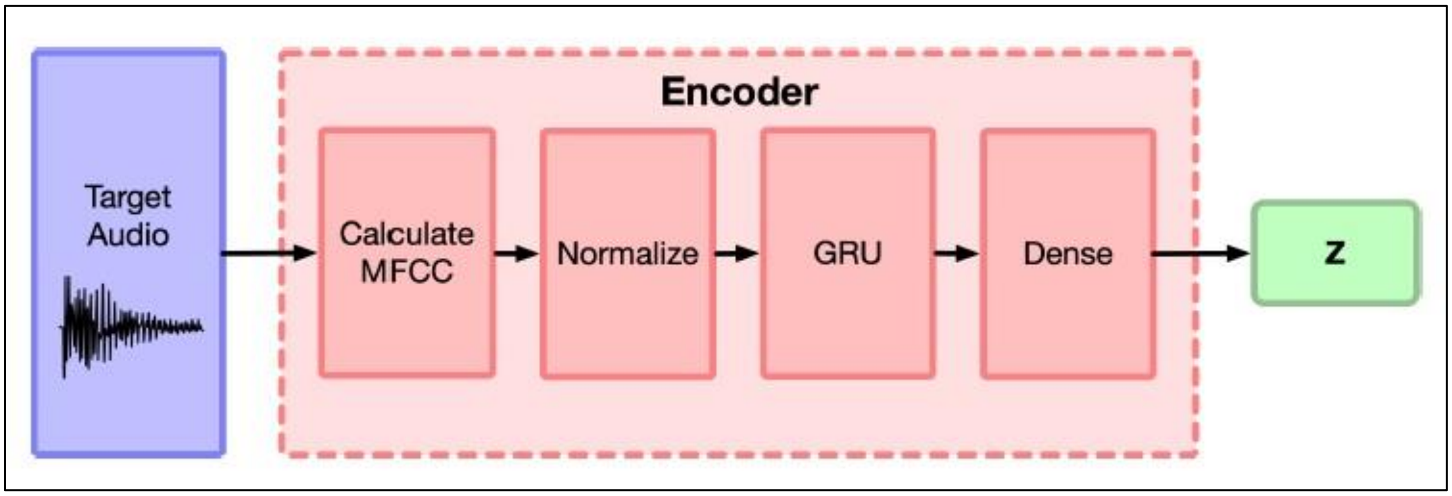


Fig 2: Separate interpolations over loudness, pitch, and timbre.

Figure 2 shows the conditioning features (solid lines) are extracted from two notes and linearly mixed (dark to light coloring). The features of the resynthesized audio (dashed lines) closely follow the conditioning. On the right, the latent vectors known as z(t), are interpolated, and the spectral centroid of resulting audio (thin solid lines) smoothly varies between the original samples (dark solid lines).

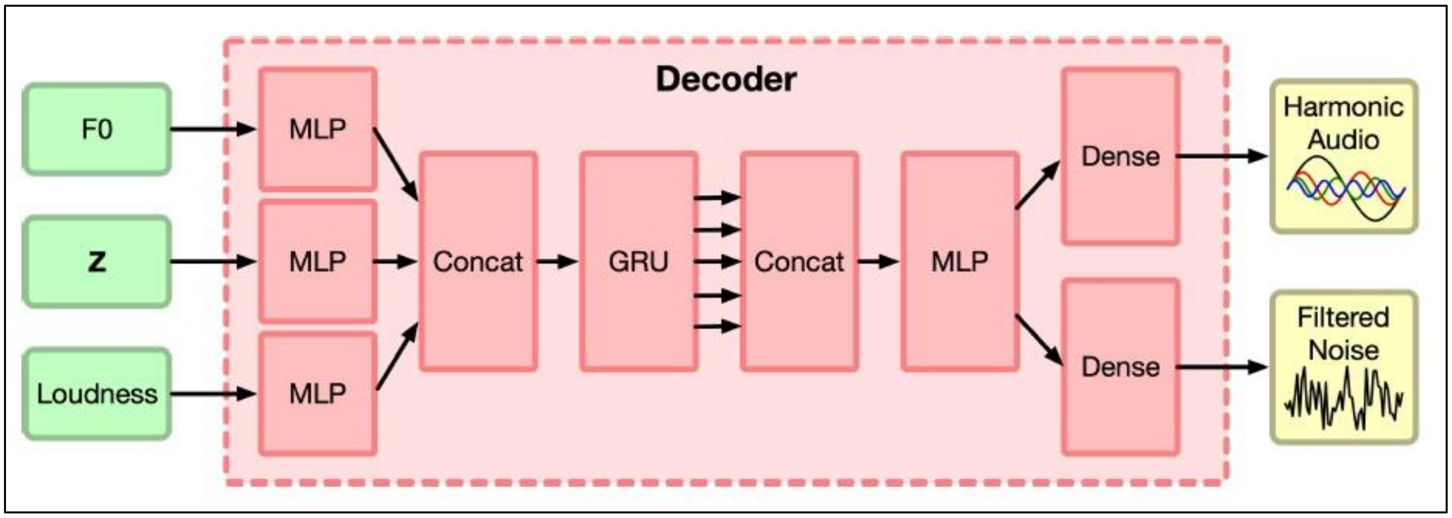
Encoder:

The model has three encoders, f-encoder that outputs fundamental frequency f(t), l-encoder that outputs loudness l(t), and a z-encoder that outputs residual vector z(t).

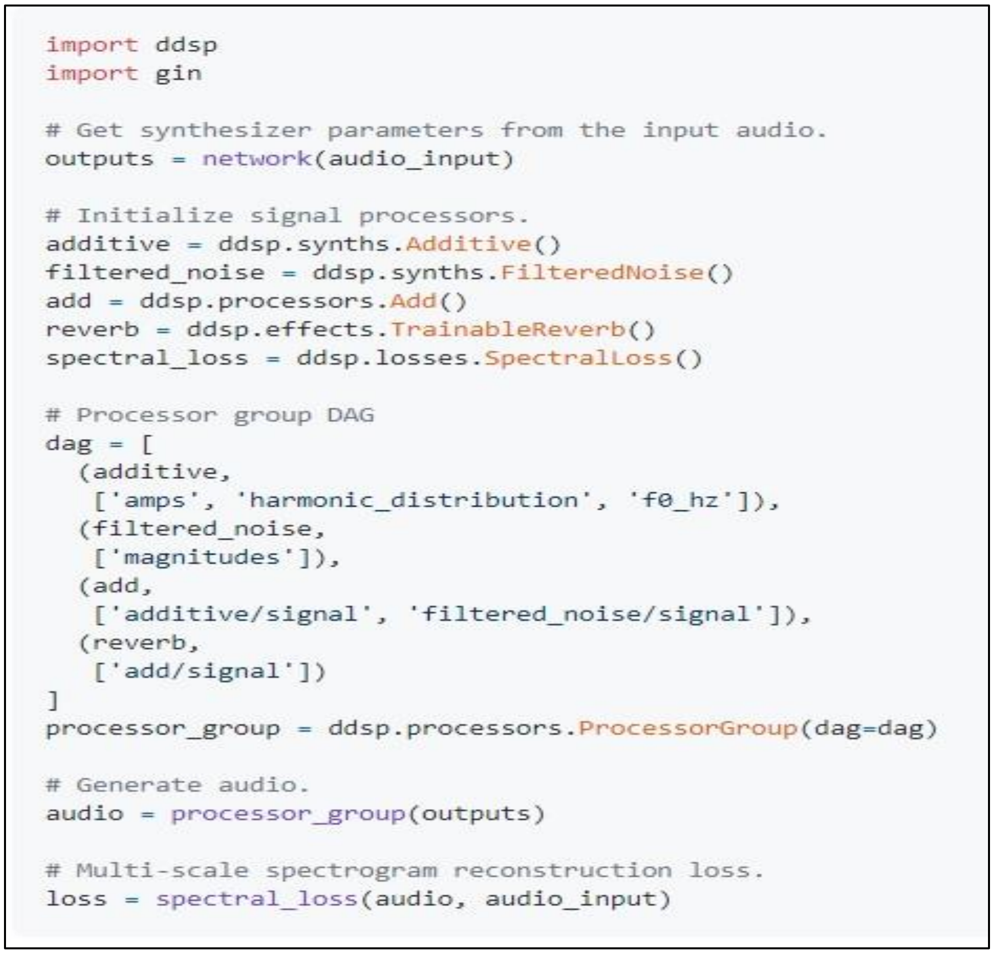


Decoder:

Decoder input is the latent tuple (f(t); l(t); z(t)). Its outputs are the parameters required by the synthesizers. For example, in the case of the harmonic synthesizer and filtered noise synthesizer setup, the decoder outputs a(t) (amplitudes of the harmonics) for the harmonic synthesizer and H (transfer function of the FIR filter) for the filtered noise synthesizer.



Processor Group: A Processor Group allows specified Directed Acyclic Graph (DAG) of processors.



```
import ddsp
import gin

# Get synthesizer parameters from the input audio.
outputs = network(audio_input)

# Initialize signal processors.
additive = ddsp.synths.Additive()
filtered_noise = ddsp.synths.FilteredNoise()
add = ddsp.processors.Add()
reverb = ddsp.effects.TrainableReverb()
spectral_loss = ddsp.losses.SpectralLoss()

# Processor group DAG
dag = [
  (additive,
   ['amps', 'harmonic_distribution', 'f0_hz']),
  (filtered_noise,
   ['magnitudes']),
  (add,
   ['additive/signal', 'filtered_noise/signal']),
  (reverb,
   ['add/signal'])
]
processor_group = ddsp.processors.ProcessorGroup(dag=dag)

# Generate audio.
audio = processor_group(outputs)

# Multi-scale spectrogram reconstruction loss.
loss = spectral_loss(audio, audio_input)
```

Some End Notes:

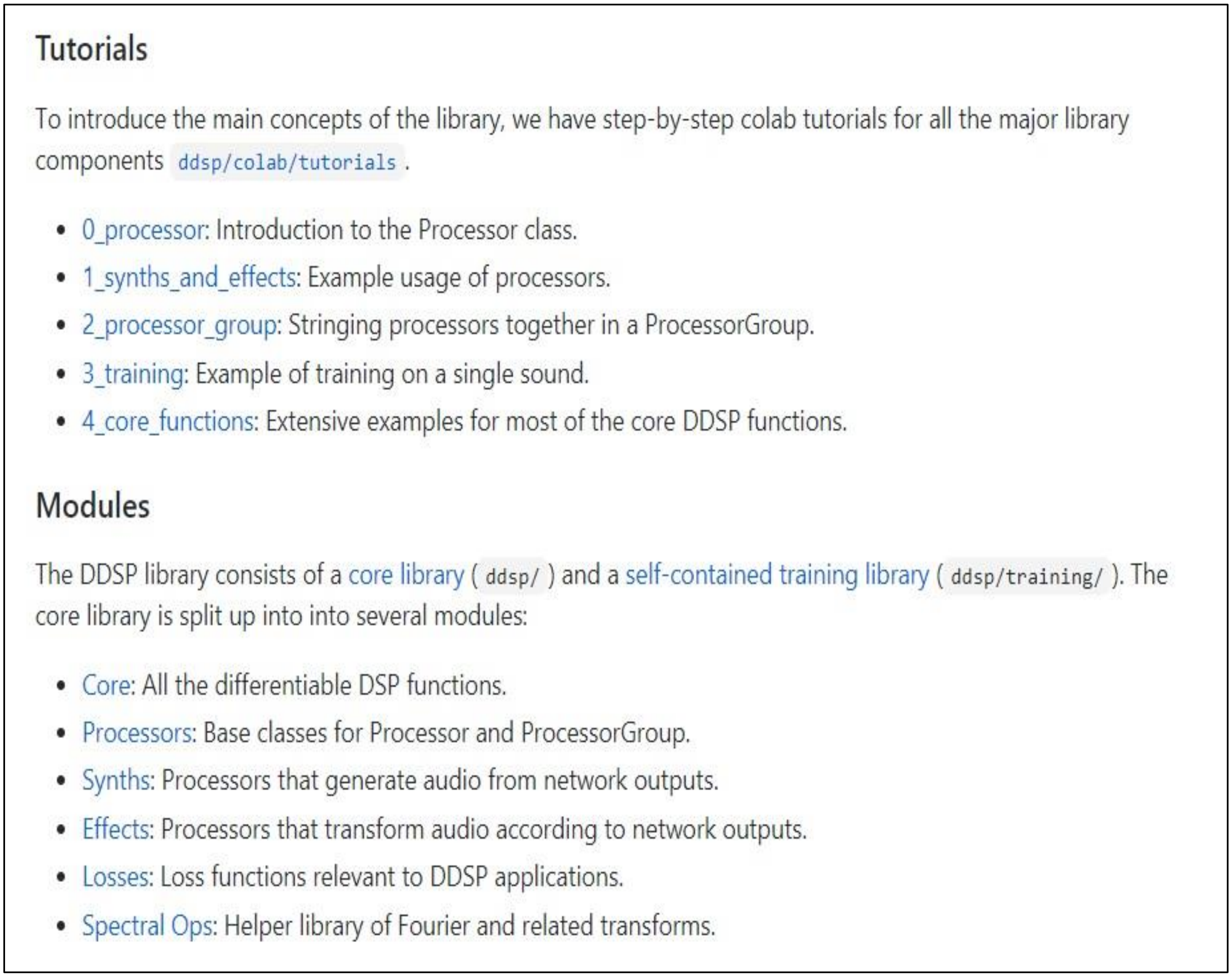
Sinusoidal models:

While unconstrained sinusoidal oscillator banks are strictly more expressive than harmonic oscillators, we restricted ourselves to harmonic synthesizers for the time being to focus on the problem domain. However, this is not a fundamental limitation of the technique and perturbations such as inharmonicity can also be incorporated to handle phenomena such as stiff strings.

Regularization:

The modular structure of the synthesizers also makes it possible to define additional losses in terms of different synthesizer outputs and parameters. For example, SNR loss to penalize outputs with too much noise if we know the training data consists of mostly clean data. We have not experimented too much with such engineered losses, but we believe they can make training more efficient, even though such engineering methods deviates from the end-to-end training paradigm.

The DDSP library consists of a core library (DDSP) and a self-contained training library (ddsp/training/). The core library is split up into several modules which are given below. Modules are Core, Processors, Synths, Effects, Losses and Spectral Ops.



Tutorials

To introduce the main concepts of the library, we have step-by-step colab tutorials for all the major library components `ddsp/colab/tutorials` .

- 0_processor: Introduction to the Processor class.
- 1_synths_and_effects: Example usage of processors.
- 2_processor_group: Stringing processors together in a ProcessorGroup.
- 3_training: Example of training on a single sound.
- 4_core_functions: Extensive examples for most of the core DDSP functions.

Modules

The DDSP library consists of a core library (`ddsp/`) and a self-contained training library (`ddsp/training/`). The core library is split up into into several modules:

- Core: All the differentiable DSP functions.
- Processors: Base classes for Processor and ProcessorGroup.
- Synths: Processors that generate audio from network outputs.
- Effects: Processors that transform audio according to network outputs.
- Losses: Loss functions relevant to DDSP applications.
- Spectral Ops: Helper library of Fourier and related transforms.

Results:

1) High-fidelity synthesis

2) Independent control of loudness and pitch

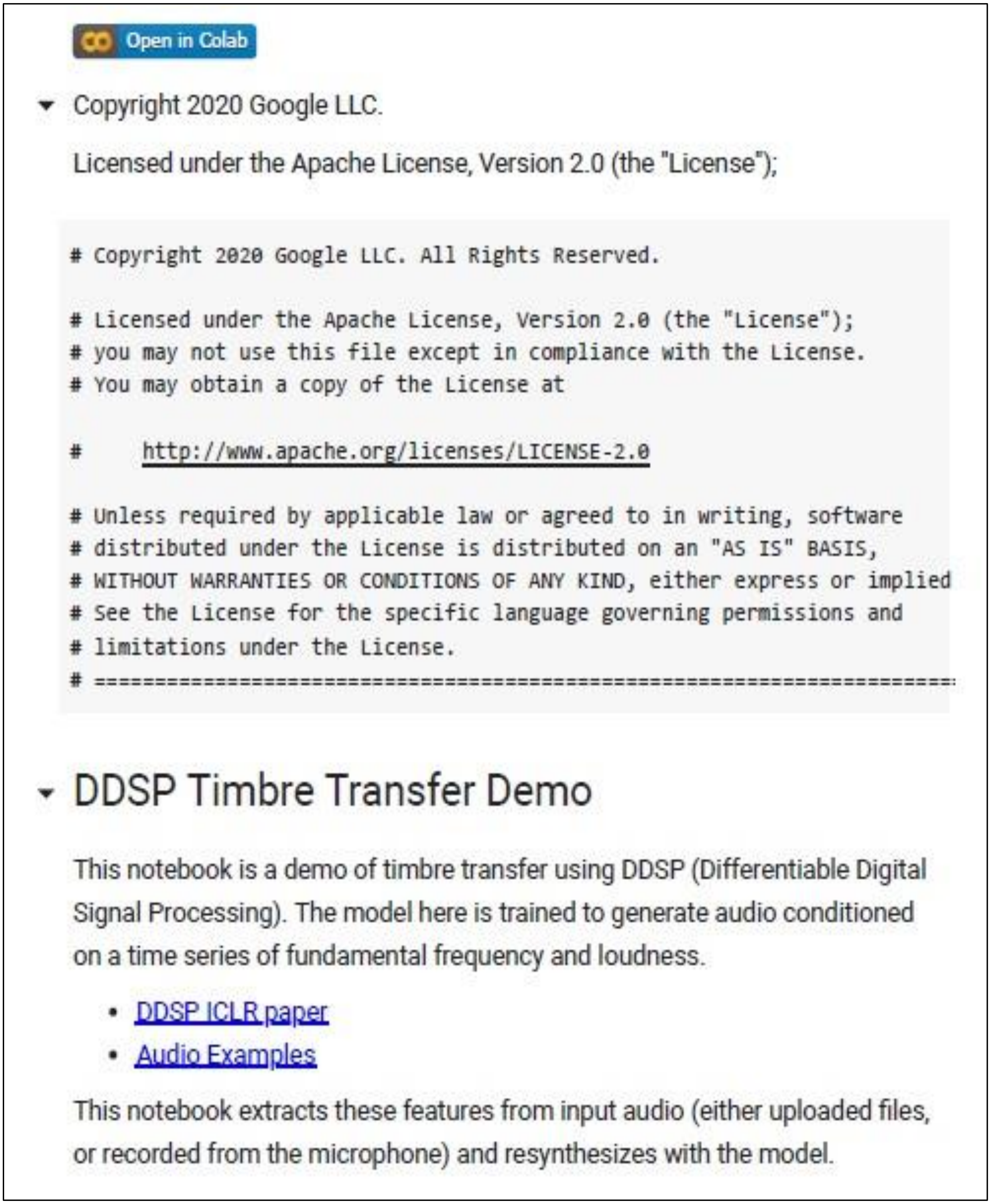
3) Dereverberation and acoustic transfer

4) Timbre transfer

Conclusions:

The DDSP library fuses classical DSP with deep learning, providing the ability to take advantage of strong inductive biases without losing the expressive power of neural networks and end-to-end learning. We encourage contributions from domain experts and look forward to expanding the scope of the DDSP library to a wide range of future applications.

The code for Timbre Transfer Demo through microphone and violin as output is implemented in Co-lab below:

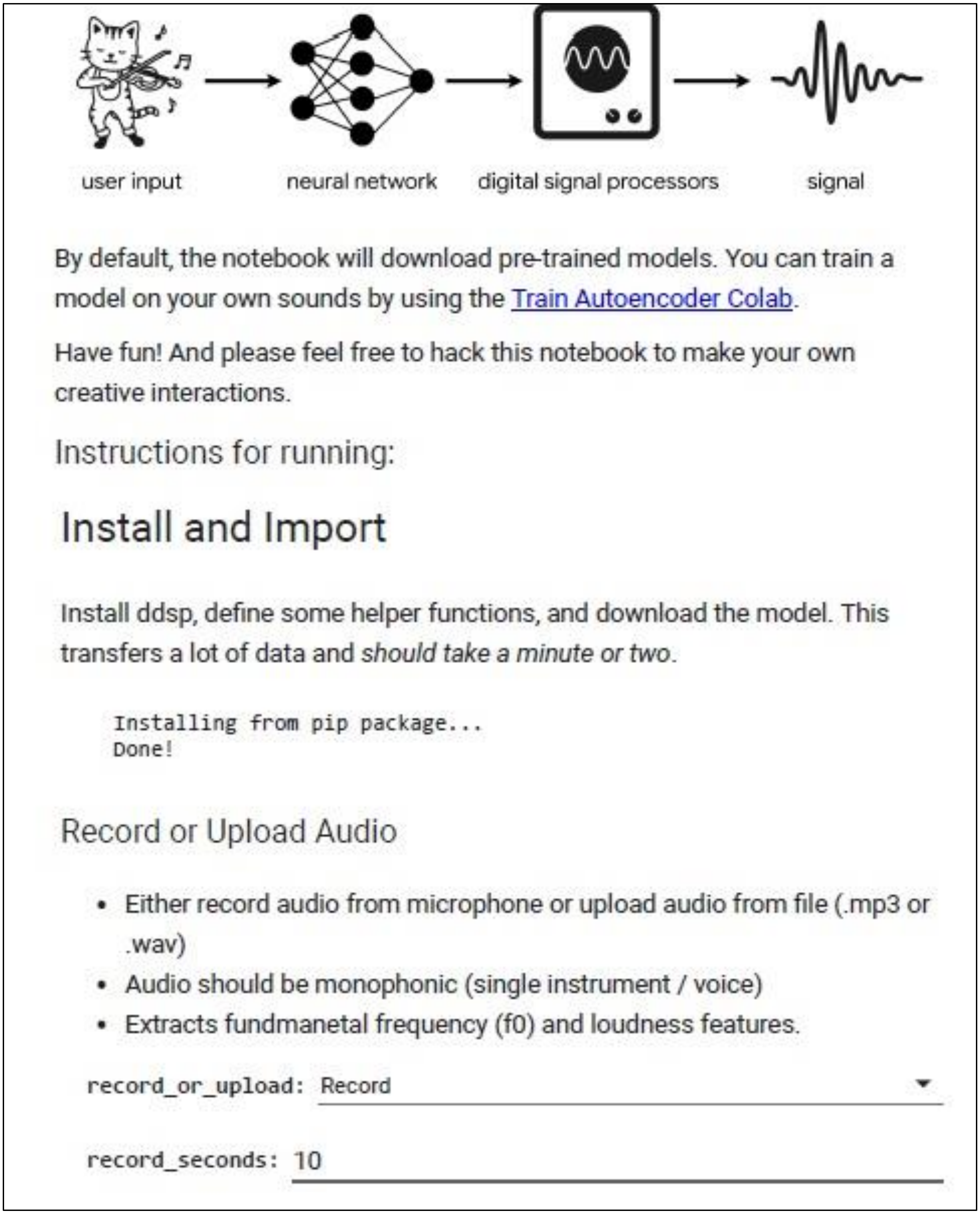


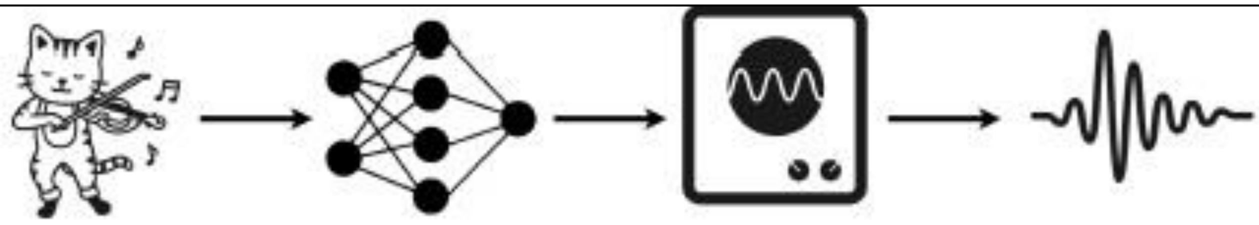
DDSP Timbre Transfer Demo

This notebook is a demo of timbre transfer using DDSP (Differentiable Digital Signal Processing). The model here is trained to generate audio conditioned on a time series of fundamental frequency and loudness.

- DDSP ICLR paper
- Audio Examples

This notebook extracts these features from input audio (either uploaded files, or recorded from the microphone) and resynthesizes with the model.





By default, the notebook will download pre-trained models. You can train a model on your own sounds by using the Train Autoencoder Colab.

Have fun! And please feel free to hack this notebook to make your own creative interactions.

Instructions for running:

Install and Import

Install ddsp, define some helper functions, and download the model. This transfers a lot of data and *should take a minute or two*.

```
Installing from pip package...
Done!
```

Record or Upload Audio

- Either record audio from microphone or upload audio from file (.mp3 or .wav)
- Audio should be monophonic (single instrument / voice)
- Extracts fundmanetal frequency (f0) and loudness features.

record_or_upload: Record

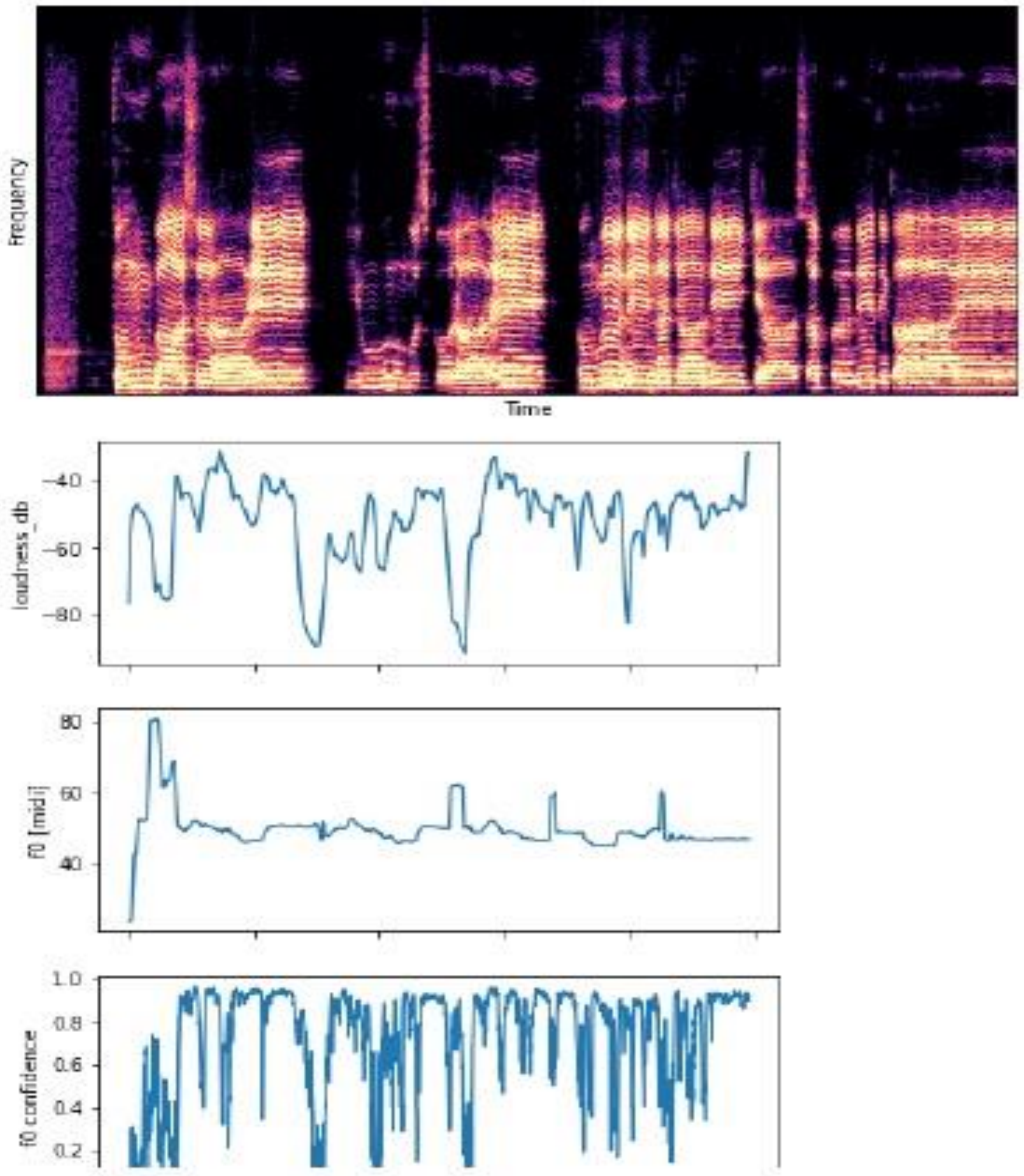
record_seconds: 10

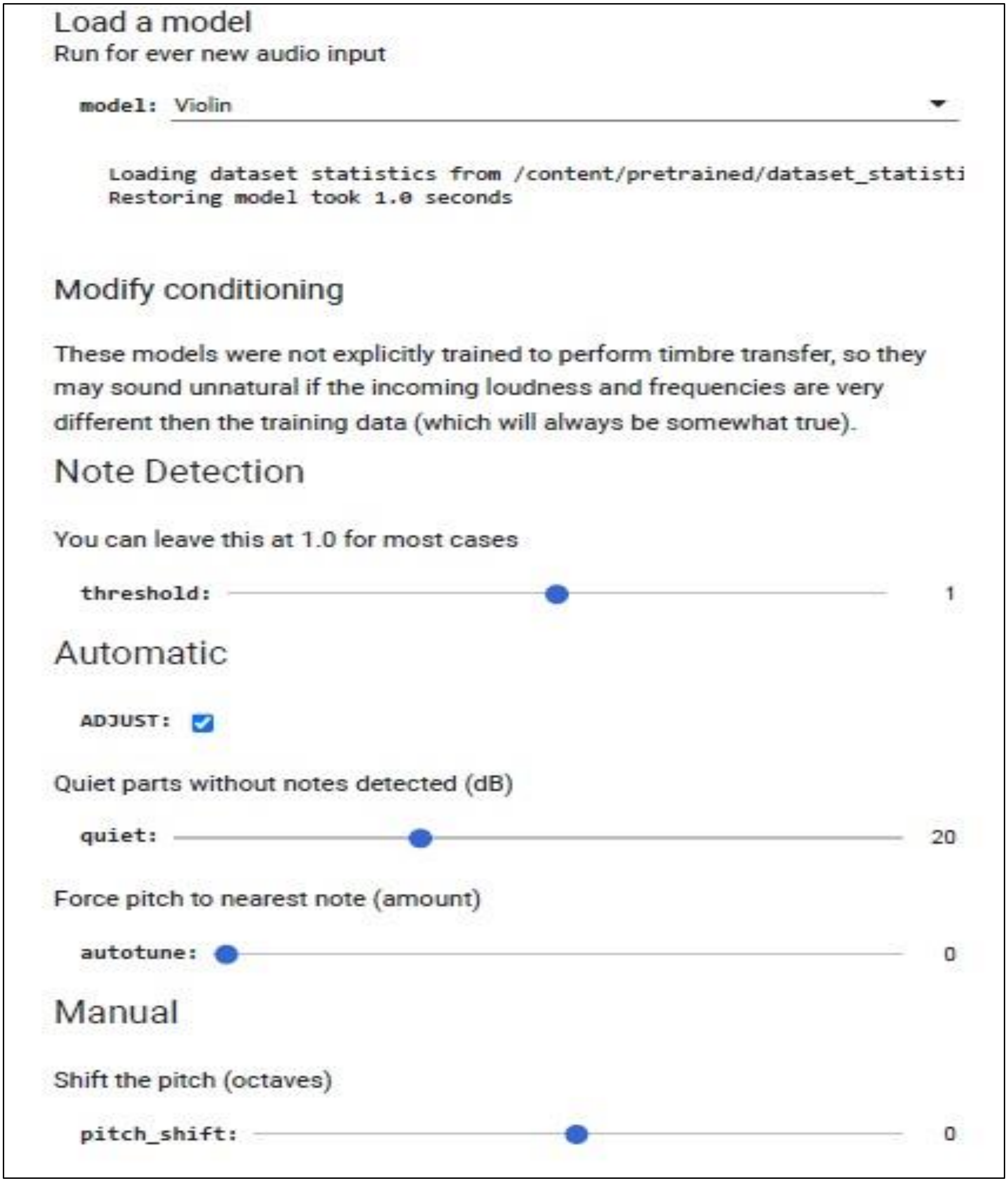
```
Starting recording for 10 seconds...
Finished recording!

Extracting audio features...

      0:09 / 0:09

Audio features took 2.2 seconds
```





Load a model
Run for ever new audio input

model: Violin

```
Loading dataset statistics from /content/pretrained/dataset_statisti
Restoring model took 1.0 seconds
```

Modify conditioning

These models were not explicitly trained to perform timbre transfer, so they may sound unnatural if the incoming loudness and frequencies are very different then the training data (which will always be somewhat true).

Note Detection

You can leave this at 1.0 for most cases

threshold: 1

Automatic

ADJUST: ✓

Quiet parts without notes detected (dB)

quiet: 20

Force pitch to nearest note (amount)

autotune: 0

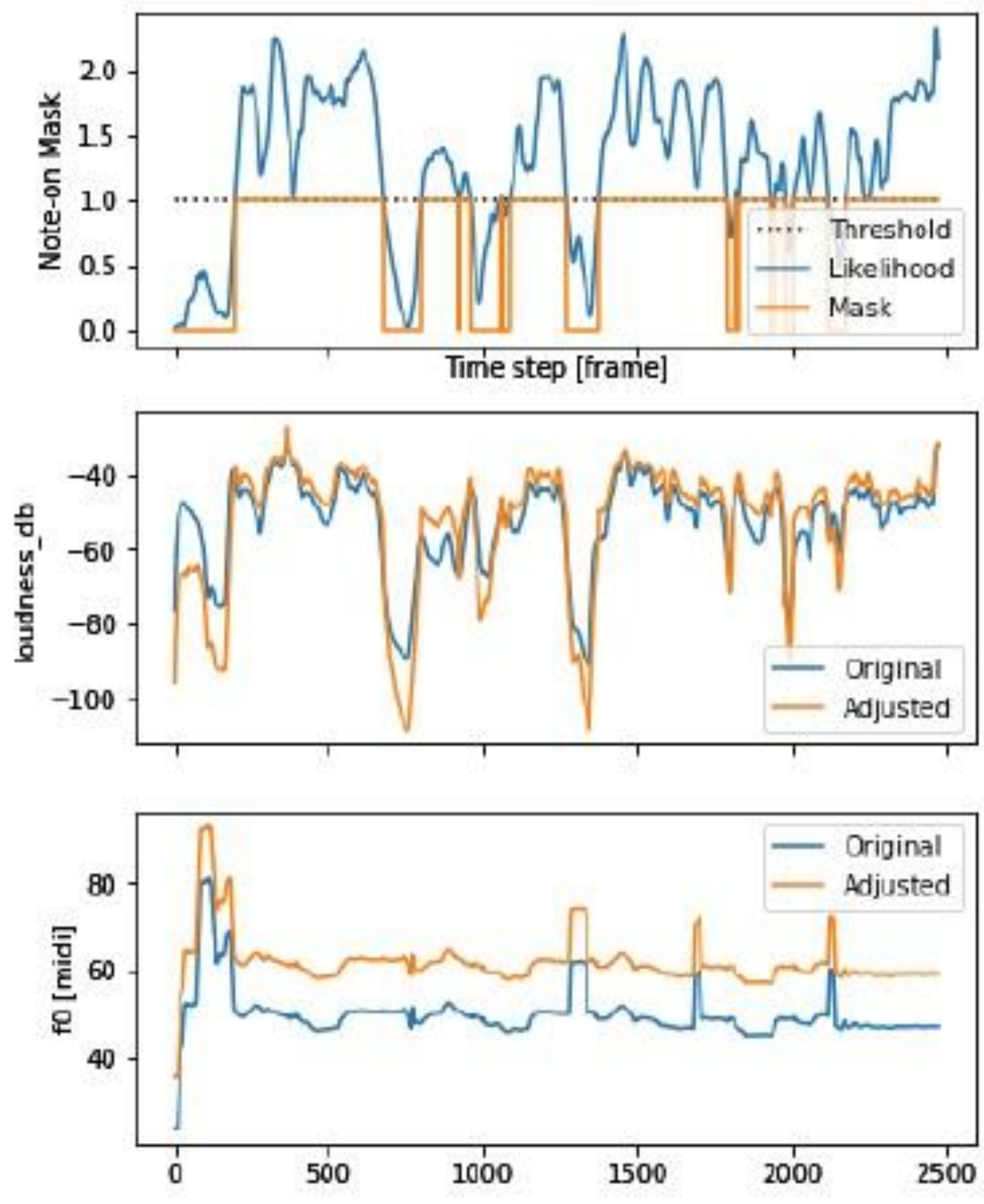
Manual

Shift the pitch (octaves)

pitch_shift: 0

Adjsut the overall loudness (dB)

loudness_shift:



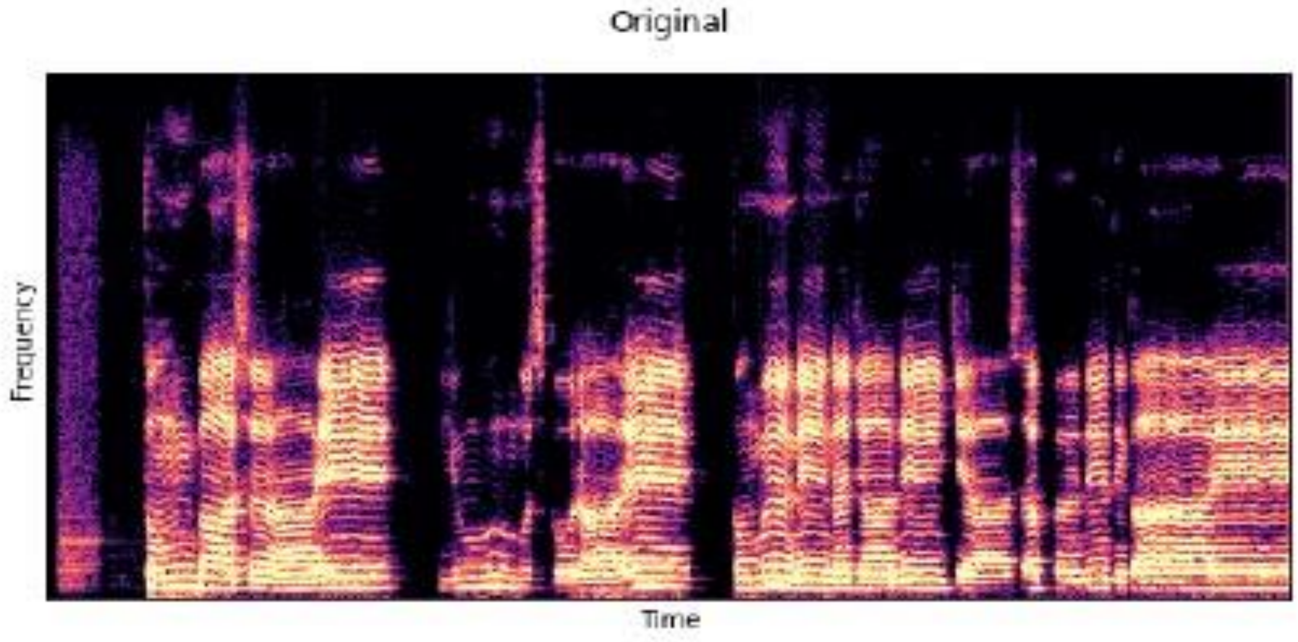
Resynthesize Audio

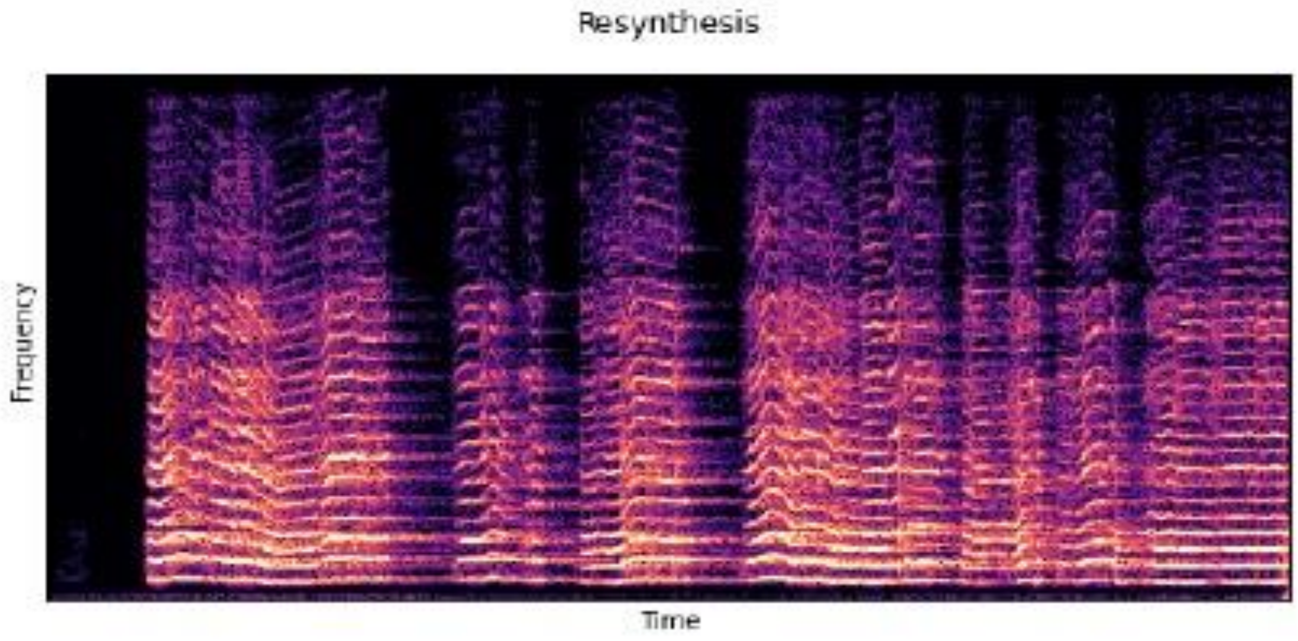
```
Prediction took 0.5 seconds
Original

      0:09 / 0:09

Resynthesis

      0:09 / 0:09
```





References:

[1] https://openreview.net/forum?id=B1x1ma4tDr,

[2] https://github.com/magenta/ddsp,

[3]https://colab.research.google.com/github/magenta/ddsp/blob/master/ddsp/colab/demos/timbre_transfer.ipynb#scrollTo=Bvp6GWqtfVCW,

[4] James L Flanagan. Speech analysis synthesis and perception, volume 3. Springer Science & Business Media, 2013.

[5] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. Crepe: A convolutional representation for pitch estimation. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 161–165. IEEE, 2018.

[6] Patrick A Naylor and Nikolay D Gaubitch. Speech dereverberation. Springer Science & Business Media, 2010.

[7] Frederic E Theunissen and Julie E Elie. Neural processing of natural sounds. Nature Reviews Neuroscience, 15(6):355, 2014.

[8] James W Beauchamp. Analysis, synthesis, and perception of musical sounds. Springer, 2007.