School of Electronic Engineering
and Computer Science

Final Report

Programme of study:
BSc Computer Science

<u>Project Title:</u> Audio Inpainting with Machine Learning

Supervisor:
Dr Emmanouil Benetos

Student Name:
Yuanxu Han

Final Year
Undergraduate Project 2021/22

Queen Mary
University of London

Date: 04/05/2022

Abstract

The applications of audio signal processing have developed significantly during the last few years. Audio inpainting is one of the most important technologies based on its potentials of the participations in multiple areas such as music production, speech recognition, etc.

The currently available audio inpainting methods require high level of human participations, such as professional audio editing tools. Although, these methods are more controlled under supervision, they are less efficient and time consuming. Nonetheless, these approaches are designed for users who are professional on audio signal processing, and the quality of audio often depends on user's abilities and the usability of the tools, which creates inconsistency in the outcome.

In this project, I have implemented a program for audio bandwidth extension that does not require professional audio signal processing skills to operate. The model is implemented in Python 3.8 with librosa (audio signal processing library), sklearn (machine learning library) and MedleyDB (multitrack recording dataset). It uses Non-negative Matrix Factorisation (NMF) to recover the missing weights of higher frequencies on Mel-spectrograms, which performs well when the training data and the test data have high similarity in chord progressions, time signatures, melody, instruments, etc. This can potentially be useful for inpainting audios that have low sample rate.

After testing the model with various data under all possible situations, it is clear that a traditional machine learning approach for audio inpainting like NMF is less complex and more efficient. The construction of the model is simple, and training the model consumes less data compared to Deep Learning models like Neural Networks. However, the quality of outcome of this model is highly dependent on the choice of the training data, meaning that the model has limitations when inpainting large variety of audios. In this case, good amount of supervision and careful choices of training data are indispensable to guarantee high-quality outputs.

Contents

Chapter 1: **Introduction**

1.1 Background

Audio inpainting has become a major part of audio signal processing. It is often related to audio upsampling, audio super-resolution and bandwidth extension. Most of the previous research involves Deep Learning and Neural Networks, which are not approachable with limited computational resources, whereas traditional machine learning models are more tolerant with less powerful computational resources(e.g., computations can be done on CPUs only). As a trade-off, its accuracy is mostly dependent on the quality or quantity of training data, which can often be problematic during data pre-processing and training. In a user's view, such approaches are great for lightweight tasks, but they are hard to understand and execute without professional knowledge on audio signal processing.

Hence, it is crucial to implement a model for audio inpainting that require less computational resources, with basic operations been simplified, while the outcome still meets the expectations.

1.2 Problem Statement

The current proposed methods for audio inpainting are mainly designed for research purposes. This makes them inefficient in the real-life scenario, due to the limitations of the computers most users have and the high level of knowledge they require. These methods should be improved and abstracted further to adapt to a wider range of hardware and users, which can benefit a larger population with the demand of audio inpainting.

1.3 Aim

To implement a model based on the previous proposed methods with a traditional machine learning approach, which is easier to set up and use. The model should be able to recover the missing frequencies of the input, return the Mel-spectrograms of the training data and the input, and save the upsampled audio time series to a local path on user's machine.

1.4 Objectives

- Extract features of an audio signal with librosa in Python (e.g., time series, Mel-spectrograms)
- Implement a Python program that pre-processes the data, trains the model with the appropriate training data and upsamples the input audio.
- Abstract the operations for audio bandwidth extension into one method.
- Save the upsampled audio to a local path. Create Mel-spectrograms from both input and output audio signals to visualise the outcome.

1.5 Research Questions

Which proposed method for audio inpainting is more suitable for traditional machine learning models?

Which traditional machine learning model is more approachable for audio bandwidth extension?

Chapter 2: **Literature Review**

2.1 Concept Definitions

2.1.1 What is audio inpainting?

A framework of algorithms for restoring an audio signal to its original form. It usually involves audio click removal, audio declipping, audio packet loss concealment, audio bandwidth extension and local time-frequency restoration. This project will be focused on audio bandwidth extension only (Adler et al., 2011).

2.1.2 Definition of audio bandwidth extension

'To retrieve the energy in high-frequency bands from low-frequency contents' (Adler et al., 2011). The bandwidth of an audio signal depends on its sampling rate. If the frequencies exceed the sampling rate, aliasing artifacts can occur in the lower frequencies. This leads to the absence of high frequencies in an audio with a low sampling rate. In this project, the higher frequency bands are retrieved by upsampling a low sampling rate audio with traditional machine learning algorithms.

2.1.3 What is machine learning

Machine learning is a set of algorithms for computers to study a training dataset in order to classify the class of an object or predict continuous numeric values. This is usually referred as supervised learning. In this project, I have defined a regression problem since the aim of the project is to generate upsampled audio signals, which are continuous numeric values. Traditional machine learning methods such as non-negative factorisation will be ideal for this project.

2.1.4 Short-Time Fourier Transform and Mel-Spectrogram

The Short-Time Fourier Transform (STFT) computes the Discrete Fourier Transforms (DFT) over short overlapping windows to transform time series into the time-frequency domain (M. Müller, 2015). The STFT magnitudes can be plotted into a spectrogram, which is commonly used in audio signal processing approaches such as audio source separation. A Mel-spectrogram is a spectrogram that is mel-scaled, meaning that the frequencies are group in bins following a logarithmic scale rather than a linear scale. It is very useful in practice because humans perceive frequencies logarithmically (S. S. Stevens et al., 1937).

2.1.5 Griffin Lim algorithm

In this project, a spectrogram upsampling model that is inspired by a 'spectrogram inpainting model' (Ya-Liang et al.,2019) will be used, which involves inversing spectrograms to audio signals. To achieve this, I will be using the Griffin Lim algorithm. 'The algorithm randomly initialises phase estimates, and the alternates forward- and inverse-STFT operations' (D.W. Griffin and J.S. Lim, 1984). For Mel-spectrograms and MFCCs, they must be inversed to STFT magnitude matrices in order to perform the GL algorithm.

2.2 Audio Bandwidth Extension Models

2.2.1 Waveform Upsampling Model

This model is proposed in ‘Audio Super-Resolution Using Neural Nets’ (Volodymyr et al., 2017). The model uses raw waveforms as input, and directly generates upsampled waveforms. The model uses a bottleneck architecture, which is presented as a series of downsampling and upsampling blocks. Based on this architecture, the structure of the model is constructed as the following:

For each downsample block with target sample rate ***r***:

W: Original dataset. A set of raw waveforms sampled at 44.1kHz.

x: A vector of audio time series: ***w*** sampled at 2***r*** (***w E W***).

$\boldsymbol{x_r}$: ***x*** downsampled to ***r***.

The downsample blocks continue until ***r*** meets a specified sample rate (included in the hyperparameters).

For each upsample block with input sample rate ***r***:

$\boldsymbol{y_r}$: A vector of target audio time series: ***w*** sampled at 2***r*** (***w E W***).

The model ***m*** is updated following $\boldsymbol{m}(\boldsymbol{x_r}, \boldsymbol{y_r})$ for upsampling any raw waveforms from ***r*** to 2***r***.

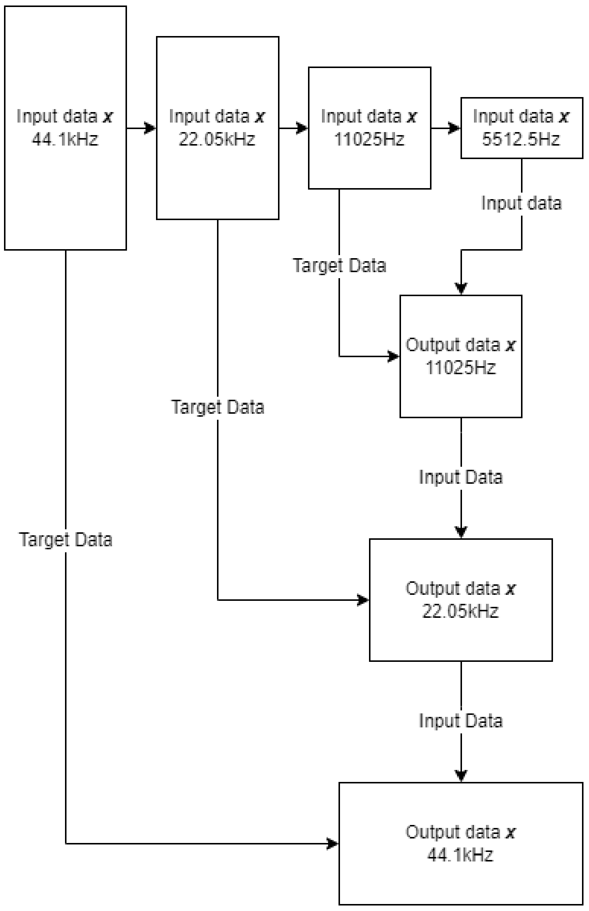


Figure 1 Waveform Upsampling Model Construction

Signals retrieve using this model is consistent since the model is trained with ground truth data. The downside of this solution is its poor accuracy because the predictions are approximated between linear vectors (time series), but not 2D matrices. The waveform upsampling model is only efficient when the input audio is under the same category as the training dataset (e.g., machine trained with speech audio can only upsample speech audio in a good quality, but not classical music audio). The training dataset will always underfits categories it does not contain.

2.2.2 Spectrogram Upsampling Model

For this model, instead of processing time series, the audio is first transformed into Mel-spectrograms (Sun and Mazumder, 2013). The structure of the model is constructed as the following:

Apply bandwidth extension to test data at target sample rate $\boldsymbol{r}$:

$\boldsymbol{y_r}$: A spectrogram of the training data sampled at $\boldsymbol{r}$.

$\boldsymbol{x}$: A spectrogram of the test data, which its sample rate is lower than $\boldsymbol{r}$.

$\boldsymbol{y_r}$ and $\boldsymbol{x}$ are split into weights ($\boldsymbol{w_y}$, $\boldsymbol{w_x}$) and features ($\boldsymbol{f_y}$, $\boldsymbol{f_x}$), and then reconstructed by calculating the matrix multiplication of $\boldsymbol{w_y}$ and $\boldsymbol{f_x}$ to generate the inpainted audio

This model retrieves missing frequencies from a spectrogram that was generated from a signal with a lower sampling rate. The spectrogram is then inversed to audio signals using GL algorithm. It is more accurate on predicting missing frequencies, and it is less limited on upsampling a wider range of categories of audio. A serious problem this model has is that a lot of approximations are applied during feature extractions and spectrogram-audio conversion, which can cause the absence of some signals and create inconsistencies in the output.

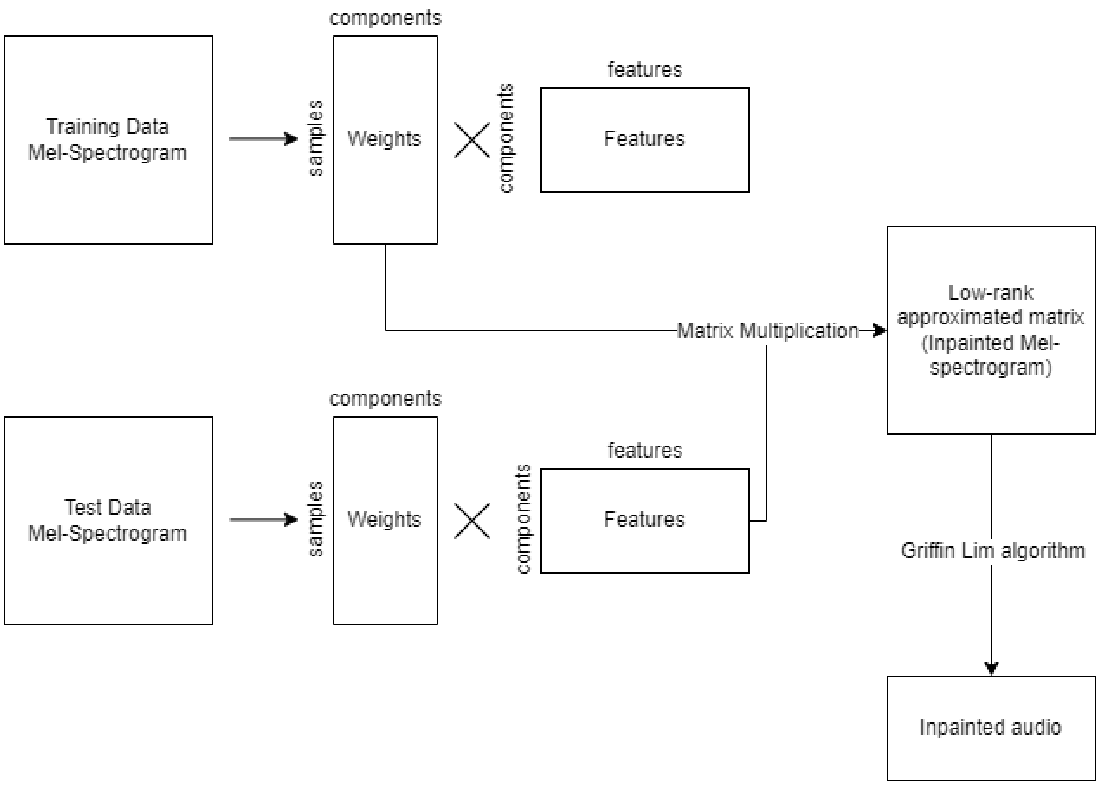


Figure 2 Spectrogram Upsampling Model Construction

2.3 Traditional Machine Learning Models

2.3.1 Support Vector Regression

A supervised learning model that finds a hyperplane in n-dimensional space based on the training dataset. A hyperplane is a linear regression of features and expected outcomes in the n-dimensional space, that is used to predict continuous numeric values (Chih-Chung et al., 2021). This is useful for inpainting audio time series.

2.3.2 Random Forest Regression

Random forest model consists of many decision trees, which the trees are created based on some random sampled subsets of the training dataset (Breiman, 2001). This model can vary on user's preferences, meaning that it is more controlled and efficient with the right adjustments in the relative sizes of the datasets.

2.3.3 Non-negative Matrix Factorisation

NMF is a method that decompose a matrix ***V*** into ***V* ≈ *WH*** (Lee and Seung, 1999). An inner dimension ***r*** of ***W*** and ***H*** is specified for factorising ***V*** where ***r* < min(*m*, *n*)**. Since NMF cannot be computed analytically, ***r*** should be given by minimising the error between ***V*** and ***WH***. Matrix ***W*** contains the abstracted weights of ***V*** and ***H*** contains features of the components in ***W***. This can be used for analysing spectrograms, which is helpful for audio bandwidth extension.

2.4 NMF with Beta-Divergence

2.4.1 Beta-Divergence

The divergence is the measure of a dissimilarity between two probability distributions (J. Olaya and C. Otman, 2021). The Beta-Divergence is used for updating ***H*** and ***W*** to derive the low-rank approximated matrix for ***V*** (C. Févotte and J. Idier, 2011). The three most popular cost functions in Lee and Seung's literature are Euclidean distance (Euc), Kullback Leibler divergence (KL) and Itakura-Saito divergence (IS):

$$d_{Euc}(x|y) = \frac{1}{2}(x-y)^2$$

$$d_{KL}(x|y) = xlog(\frac{x}{y}) - x + y$$

$$d_{IS}(x|y) = \frac{x}{y} - log(\frac{x}{y}) - 1$$

Figure 3 The three Beta-Divergence

2.4.2 NMF Solvers

In the sklearn.decomposition.NMF class, there are two available solvers: the Coordinate Descent solver (CD) and the Multiplicative Update solver (MU). The CD solver is based on the idea of minimising the factorisation error in one direction at a time, and solve this optimisation problem in a loop. The MU solver is the more standard and simpler algorithm provided by Lee and Seung:

$$\boldsymbol{W \leftarrow W \cdot (VH^T) / (WHH^T)}$$

$$\boldsymbol{H \leftarrow H \cdot (W^TV) / (W^TWH)}$$

2.5 Evaluation Metrics

2.5.1 Loss of Waveform

The L1 loss (Mean Absolute Error) measures the loss by aligning the target waveform and the predicted waveform, finding their absolute differences at each point and average them. The L2 loss (Mean Squared Error) is similar to L1 loss, but the absolute differences are squared before calculating the mean (Evan Radkoff, 2021).

With the L2 loss, the Error-to-Signal Ratio (ESR) (Steinmetz et al., 2020) can be derived as:

$$\ell_{\text{ESR}}(\hat{y}, y) = \frac{\sum_{i=0}^{N-1} |\hat{y}_i - y_i|^2}{\sum_{i=0}^{N-1} |y_i|^2}$$

Figure 4 Error-to-Signal Ratio formula

The ESR gives a clear illustration of the level of error in a signal, which can be a good metric for evaluating the perceptual quality of the predicted outcome.

2.5.2 Spectral Loss

For models that work with spectrograms instead of time series, it is necessary to visualise the differences between spectrograms (usually log magnitude spectrograms). This can be achieved by applying L1 and L2 loss functions to the ground truth and predicted spectrograms, which can be formulate as the log magnitude loss function:

$$\left|\left|log(|STFT(y)| + \epsilon) - log(|STFT(\hat{y})| + \epsilon)\right|\right|_\ell$$

Figure 5 Log magnitude loss function

If Mel-spectrograms are used instead of log magnitude spectrograms, the above loss function can be adopted by converting STFT magnitudes to Mel-spectrograms.

Chapter 3: **Design and Implementation**

3.1 Proposed Solution

3.1.1 Choice of ML Model and Audio Bandwidth Extension Method

Based on my research, most of the current audio bandwidth extension methods use Deep Learning models to inpaint spectrograms rather than time series, and they use traditional ML models for finding non-negative low-rank approximating matrices, such that the divergence between training data and test data is minimised. Since the computations are related to matrices, it is more suitable to use the spectrogram upsampling model, as Mel-spectrograms are represented in 2D numpy arrays in Python. Hence, NMF is the most ideal traditional ML model for this task. In conclusion, I will build a Spectrogram Upsampling Model using NMF with Multiplicative Update Rule and Kullback Leibler Beta-Divergence, and test the spectral losses between the input signal, the ground truth and the inpainted signal.

3.1.2 Preparation

For this project, I will be using the MedleyDB (R. Bittner et al., 2014) as my training dataset, because it includes a large variety of audio, and there are more than enough audio files under each category for training and testing the model.

The test data is a randomly selected audio that has a duration of 10 seconds and sample rate of 2756.25hz ($44100/2^4$). This is to simulate the loss of higher frequencies caused by low sample rate. The training data for each test data will be the original audio of the test data in 44100hz, a recording in the same genre with similar structure, and another recording in a different genre. For testing, the ground truth will be the original audio of the test data, and the spectral losses between the test data, the inpainted audio and the ground truth will be calculated to evaluate the accuracy of the model.

3.1.3 Model Construction

In order to perform matrix multiplications, the test data has to be resampled at 44100hz (notice that no additional information will be brought to the test data). Both test data and training data will be transformed into Mel-spectrograms at 44100hz, and the weights of the training data ($\boldsymbol{w_y}$) and the features of the test data ($\boldsymbol{f_x}$) will be derived by applying NMF on both Mel-spectrograms (following the Multiplicative Update Rule, with Kullback Leibler Beta-Divergence). $\boldsymbol{w_y}$ and $\boldsymbol{f_x}$ are then multiplied together to get the low-rank approximated matrix of the Mel-spectrogram of the training data.

3.2 Implementation

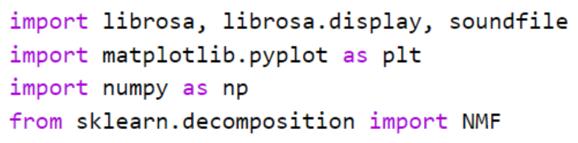
3.2.1 Import Necessary Python libraries and methods

librosa: Transform audio signals into time series and Mel-spectrograms.

matplotlib.pyplot, **librosa.display**: Manage plots and display Mel-spectrograms.

sklearn.decomposition.NMF: sklearn class for Non-negative Matrix Factorisation.

numpy: Support the ndarray datatype for time series and Mel-spectrograms

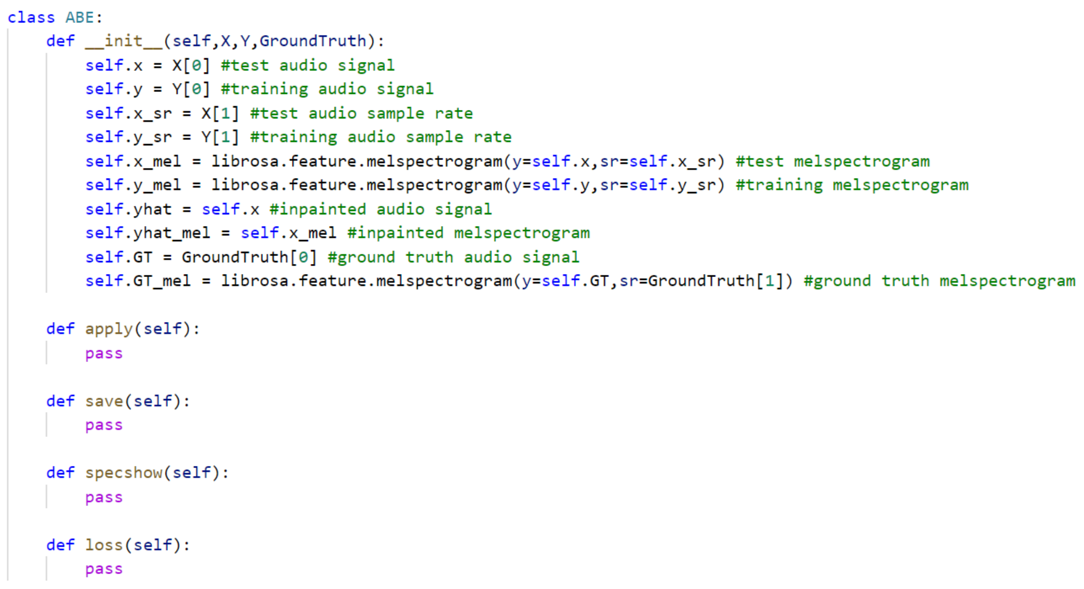


```
import librosa, librosa.display, soundfile
import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import NMF
```

Figure 6 Imported libraries

3.2.2 Define Audio Bandwidth Extension Class

The Audio Bandwidth Extension class will take in three parameters (‘X’, ‘Y’ and ‘GroundTruth’), where ‘X’ is a tuple of test data time series and test data sample rate, ‘Y’ is a tuple of training data time series and training data sample rate, and ‘GroundTruth’ is a tuple of ground truth data time series and ground truth sample rate. The class constructor will store the time series, sample rates and Mel-spectrograms for test data, training data, ground truth and inpainted audio. The class should provide methods for inpainting the test audio (‘apply’), saving the inpainted audio to local paths (‘save’), displaying spectral differences between Mel-spectrograms (‘specshow’), and calculating spectral and waveform loss (‘loss’).



```
class ABE:
    def __init__(self,X,Y,GroundTruth):
        self.x = X[0] #test audio signal
        self.y = Y[0] #training audio signal
        self.x_sr = X[1] #test audio sample rate
        self.y_sr = Y[1] #training audio sample rate
        self.x_mel = librosa.feature.melspectrogram(y=self.x,sr=self.x_sr) #test melspectrogram
        self.y_mel = librosa.feature.melspectrogram(y=self.y,sr=self.y_sr) #training melspectrogram
        self.yhat = self.x #inpainted audio signal
        self.yhat_mel = self.x_mel #inpainted melspectrogram
        self.GT = GroundTruth[0] #ground truth audio signal
        self.GT_mel = librosa.feature.melspectrogram(y=self.GT,sr=GroundTruth[1]) #ground truth melspectrogram

    def apply(self):
        pass

    def save(self):
        pass

    def specshow(self):
        pass

    def loss(self):
        pass
```

Figure 7 ‘ABE’ Class Definition

3.2.3 ABE.apply

The ‘apply’ method has a parameter ‘mode’ for deciding which machine learning model will be used. Since the project is aimed to adopt NMF with Beta-Divergence for the Spectrogram Upsampling model, it is set to ‘NMF’ by default. This parameter allows future extensions like Convolutional Neural Networks to be added to the method to provide more options. If the mode specified by the user is not supported, it will warn the user and tell them what the available options are. If the mode is ‘NMF’, it will resample the test data at the sample rate of the training data (44100hz), and create a new Mel-spectrogram with the higher sample rate for the test data. Then, the NMF model from the sklearn.decomposition.NMF class will be defined with the following parameters:

Init=’random’: Initialise the procedure with non-negative random matrices.

random_state=1: Specify the randomisation with integer.

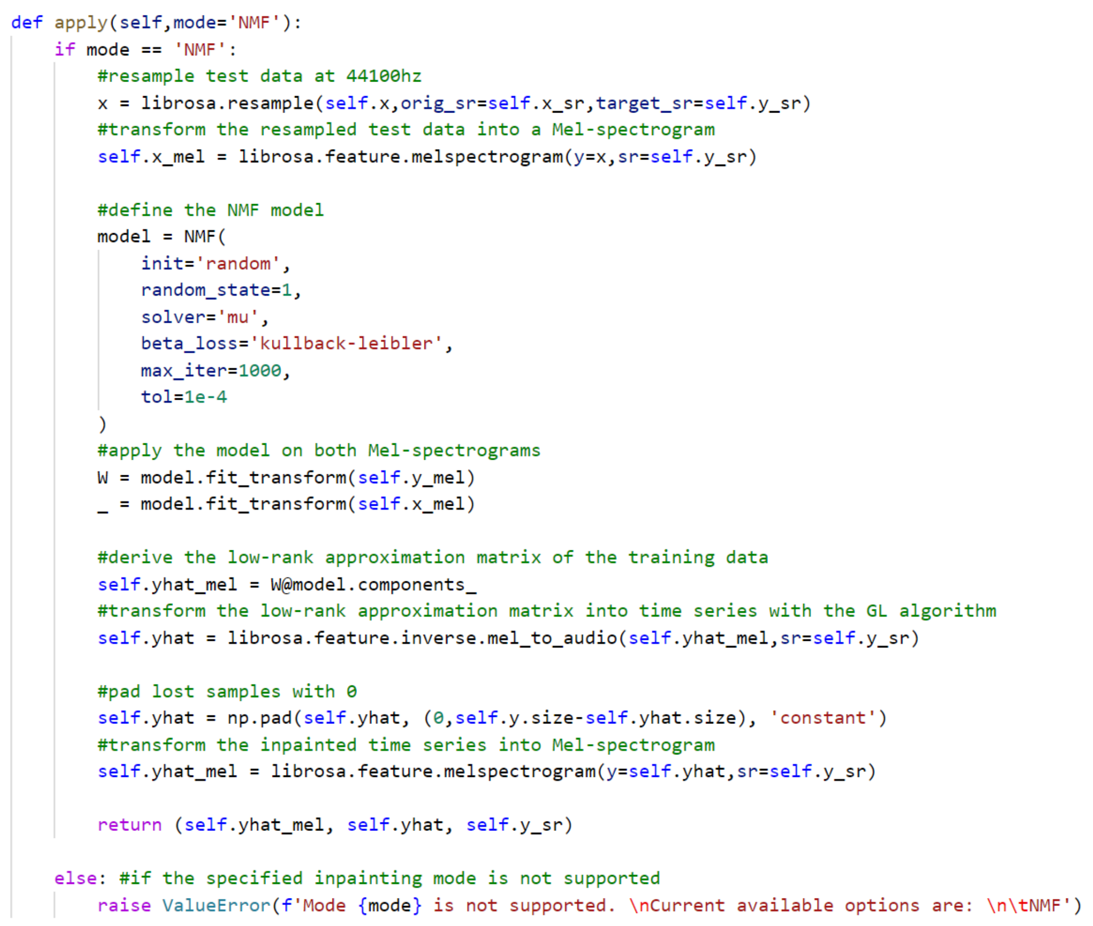
solver: Choose Multiplicative Update solver for this NMF model.

beta_loss=’kullback-leibler’: Use Kullback Leibler Beta-Divergence.

max_iter=1000: Set the maximum number of iterations to 1000 before timing out.

tol=1e-4: Set the tolerance of the stopping condition to 10^{-4}.

After the initialisation, the Mel-spectrogram of the training data will be fit to the NMF model and transformed into two matrices, and the weights will be returned (the features are stored in **model.components_**). The Mel-spectrogram of the test data will also be fit and transformed to extract its features. In order to derive the low-rank approximation of the training data, a matrix multiplication will be performed between the weights of the training data and the features of the test data. Then, the low-rank approximated matrix (inpainted Mel-spectrogram) will be transformed into time series using **librosa.feature.inverse.mel_to_audio**, which the Mel-spectrogram is transformed to STFT magnitudes first, and the STFT magnitude is transformed to audio using the Griffin Lim algorithm. Lastly, due to the approximations in the model, there can be some sample loss that can lead to a shorter duration of the inpainted audio. Although the aural difference is not significant, it can be problematic for calculating waveform and spectral losses since the sizes of the time series and Mel-spectrogram matrices of the ground truth and the inpainted audio have to be the same. Therefore, the lost samples will be padded with zeros, and a new Mel-spectrogram will be created for the inpainted audio. The method will then return the Mel-spectrogram, the inpainted audio and its sample rate as a tuple.

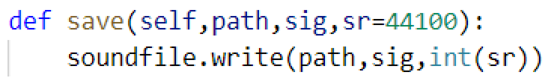


```
def apply(self,mode='NMF'):
    if mode == 'NMF':
        #resample test data at 44100hz
        x = librosa.resample(self.x,orig_sr=self.x_sr,target_sr=self.y_sr)
        #transform the resampled test data into a Mel-spectrogram
        self.x_mel = librosa.feature.melspectrogram(y=x,sr=self.y_sr)

        #define the NMF model
        model = NMF(
            init='random',
            random_state=1,
            solver='mu',
            beta_loss='kullback-leibler',
            max_iter=1000,
            tol=1e-4
        )
        #apply the model on both Mel-spectrograms
        W = model.fit_transform(self.y_mel)
        _ = model.fit_transform(self.x_mel)

        #derive the low-rank approximation matrix of the training data
        self.yhat_mel = W@model.components_
        #transform the low-rank approximation matrix into time series with the GL algorithm
        self.yhat = librosa.feature.inverse.mel_to_audio(self.yhat_mel,sr=self.y_sr)

        #pad lost samples with 0
        self.yhat = np.pad(self.yhat, (0,self.y.size-self.yhat.size), 'constant')
        #transform the inpainted time series into Mel-spectrogram
        self.yhat_mel = librosa.feature.melspectrogram(y=self.yhat,sr=self.y_sr)

        return (self.yhat_mel, self.yhat, self.y_sr)

    else: #if the specified inpainting mode is not supported
        raise ValueError(f'Mode {mode} is not supported. \nCurrent available options are: \n\tNMF')
```

Figure 8 ‘ABE.apply’ Method

3.2.4 ABE.save

The ‘save’ method saves an audio to a given local path on the user’s machine. It takes in the path, the signal (time series) and the sample rate of the audio as parameters and use **soundfile.write** to write the audio to the specified path.

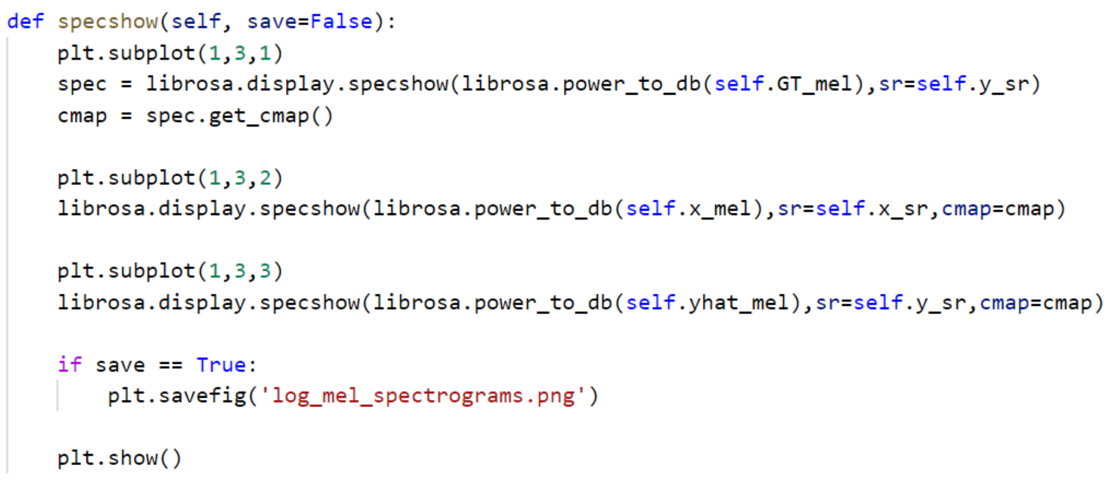


```
def save(self,path,sig,sr=44100):
    soundfile.write(path,sig,int(sr))
```

Figure 9 ‘ABE.save’ Method

3.2.5 ABE.specshow

The ‘specshow’ method plots the log-Mel-spectrogram of the ground truth, the test data and the inpainted audio using matplotlib. First, it creates a subplot using **plt.subplot** for each audio (the three integers represent the number of rows, the number of columns and the indices). Then, Mel-spectrogram is transformed to using **librosa.power_to_db** and plotted with **librosa.display.specshow**. The reason why the Mel-spectrograms are transformed to log-Mel-spectrograms is because by converting a power spectrogram to decibel (dB) units, the spectrograms will have higher contrasts, which makes them easier to compare with each other. Finally, if the parameter ‘save’ is set to ‘True’, the spectrograms will be saved to ‘log_mel_spectrograms.png’ using **plt.savefig**, and displayed with **plt.show**.

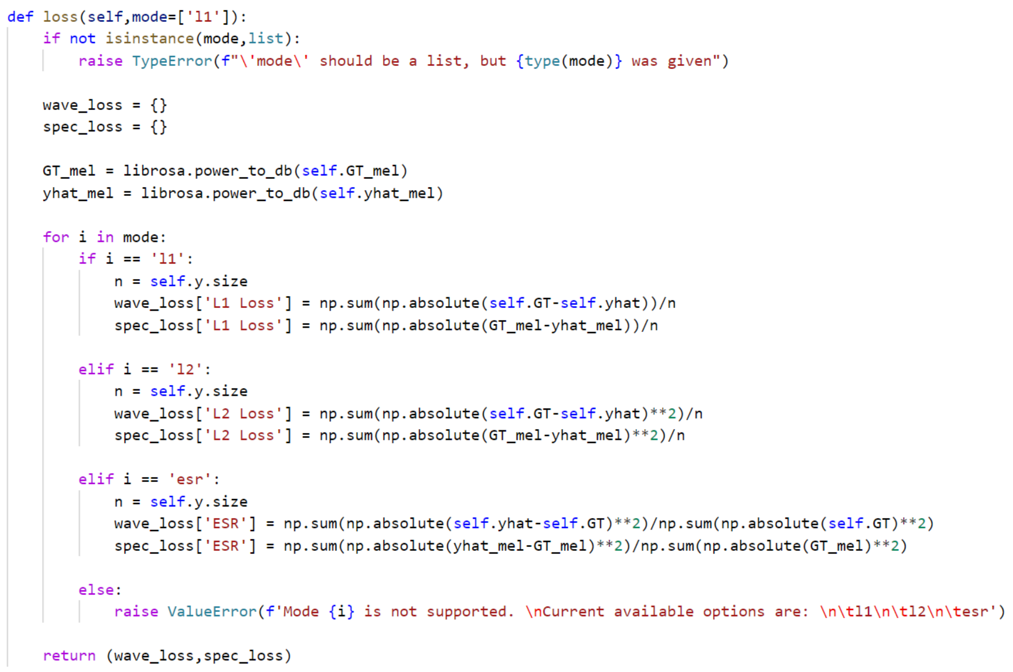


```python
def specshow(self, save=False):
    plt.subplot(1,3,1)
    spec = librosa.display.specshow(librosa.power_to_db(self.GT_mel),sr=self.y_sr)
    cmap = spec.get_cmap()

    plt.subplot(1,3,2)
    librosa.display.specshow(librosa.power_to_db(self.x_mel),sr=self.x_sr,cmap=cmap)

    plt.subplot(1,3,3)
    librosa.display.specshow(librosa.power_to_db(self.yhat_mel),sr=self.y_sr,cmap=cmap)

    if save == True:
        plt.savefig('log_mel_spectrograms.png')

    plt.show()
```

Figure 10 ‘ABE.specshow’ Method

3.2.6 ABE.loss

The ‘loss’ method calculates the different waveform and spectral losses based on user’s preference. It takes in a parameter ‘mode’, which is a list of available loss functions the user wants to apply on the data. Currently, it supports three loss functions, the L1 loss, the L2 loss and the Error-to-Signal Ratio. First, it will check the type of the parameter ‘mode’ to ensure that it is a Python list object. If it is not a list, a type error will be raised to warn the user that they have inputted the wrong parameter. It then defines ‘wave_loss’ and ‘spec_loss’ as dictionaries to store the different losses of waveforms and spectrograms. As mentioned in the **Evaluation Metrics** section, the spectral loss uses dB as the unit for spectrograms, so the Mel-spectrograms of the ground truth and inpainted audio will be transformed using **librosa.power_to_db**. After the above preparation, the method will iterate through ‘mode’, and calculate the losses according to the items of the list and store the results in ‘wave_loss’ and ‘spec_loss’. Finally, it will return ‘wave_loss’ and ‘spec_loss’ as a tuple.



```python
def loss(self,mode=['l1']):
    if not isinstance(mode,list):
        raise TypeError(f"\'mode\' should be a list, but {type(mode)} was given")

    wave_loss = {}
    spec_loss = {}

    GT_mel = librosa.power_to_db(self.GT_mel)
    yhat_mel = librosa.power_to_db(self.yhat_mel)

    for i in mode:
        if i == 'l1':
            n = self.y.size
            wave_loss['L1 Loss'] = np.sum(np.absolute(self.GT-self.yhat))/n
            spec_loss['L1 Loss'] = np.sum(np.absolute(GT_mel-yhat_mel))/n

        elif i == 'l2':
            n = self.y.size
            wave_loss['L2 Loss'] = np.sum(np.absolute(self.GT-self.yhat)**2)/n
            spec_loss['L2 Loss'] = np.sum(np.absolute(GT_mel-yhat_mel)**2)/n

        elif i == 'esr':
            n = self.y.size
            wave_loss['ESR'] = np.sum(np.absolute(self.yhat-self.GT)**2)/np.sum(np.absolute(self.GT)**2)
            spec_loss['ESR'] = np.sum(np.absolute(yhat_mel-GT_mel)**2)/np.sum(np.absolute(GT_mel)**2)

        else:
            raise ValueError(f'Mode {i} is not supported. \nCurrent available options are: \n\tl1\n\tl2\n\tesr')

    return (wave_loss,spec_loss)
```

Figure 11 ‘ABE.loss’ Method

Chapter 4: **Testing and Evaluation**

4.1 Testing

4.1.1 Testing Method

In order to test the program and evaluate the outcome, I have built a test class. It randomly samples an audio as the test data from the MedleyDB, and takes the ground truth, an audio with the highest spectral similarity (the lowest ESR) and an audio with the lowest spectral similarity (the highest ESR) to simulate three different inpainting conditions for testing the limitation of this model. It provides a method called ‘run’, which takes in the specified training data index (0 means ground truth, 1 means highest similarity, and 2 means lowest similarity) as the parameter and applies the model to the data to derive the inpainted audio and various losses. It also displays the spectrograms and saves them to the ABE directory as a ‘.png’ file.

4.1.2 test Class Construction

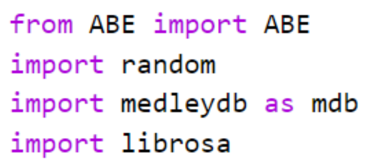
First of all, the necessary modules and libraries are imported:

ABE: The audio bandwidth extension model.

random: For selecting a random sample from the MedleyDB with a random integer index.

medleydb: For getting the paths of all multitrack mix ‘.wav’ files from the MedleyDB.

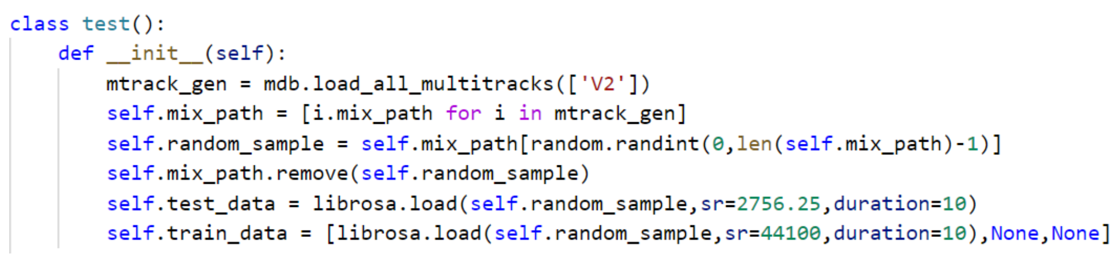
librosa: For loading audio into time series.



```
from ABE import ABE
import random
import medleydb as mdb
import librosa
```

Figure 12 Imported Libraries

In the class constructor, the paths of all multitrack mix audio files are stored in ‘self.mix_path’ for finding the test data and the suitable training data. The test data path is randomly sampled from the mix path list, then it is removed from the list (this is for finding an audio with the highest spectral similarity other than the test data, therefore the test data has to be excluded). The test data and the training data(ground truth) is loaded at 2756.25hz and 44100hz using the randomly selected path. The ground truth is stored at index 0 of a list, and the next 2 indices of the list is set to ‘None’, which will be replaced by other training data later.

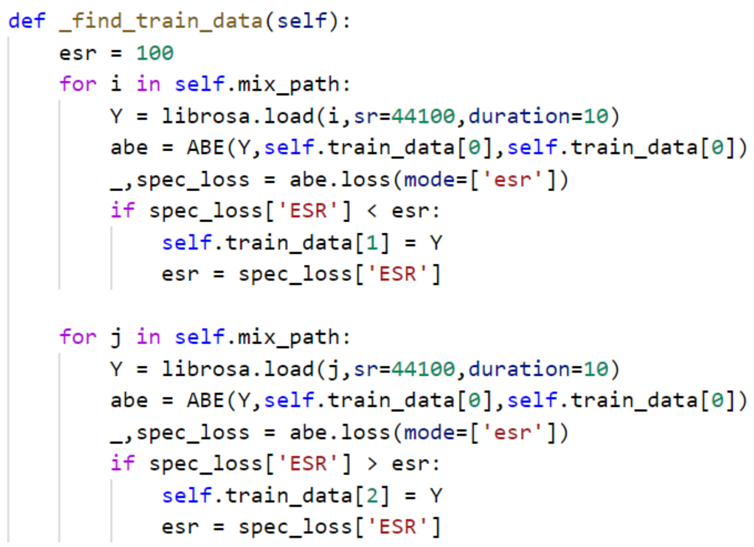


```
class test():
    def __init__(self):
        mtrack_gen = mdb.load_all_multitracks(['V2'])
        self.mix_path = [i.mix_path for i in mtrack_gen]
        self.random_sample = self.mix_path[random.randint(0,len(self.mix_path)-1)]
        self.mix_path.remove(self.random_sample)
        self.test_data = librosa.load(self.random_sample,sr=2756.25,duration=10)
        self.train_data = [librosa.load(self.random_sample,sr=44100,duration=10),None,None]
```

Figure 13 Class Constructor

4.1.3 test._find_train_data

This method is used for finding the training data with the highest spectral similarity and the training data with the lowest spectral similarity. It uses ESR to determine the spectral loss between the training data and the ground truth. The expected ESR is defined as ‘esr = 100’ at first (a value bigger than 1.0), and the method will loop through the MedleyDB to calculate the esr between each audio and the ground truth. Each time a lower ESR is found, the ‘temporary most similar audio’ will be replaced by the current audio and the ‘esr’ will be updated. After the first loop, the method will find any audio that has a ESR higher than the current ‘esr’ in a second loop, which is similar to the first loop.

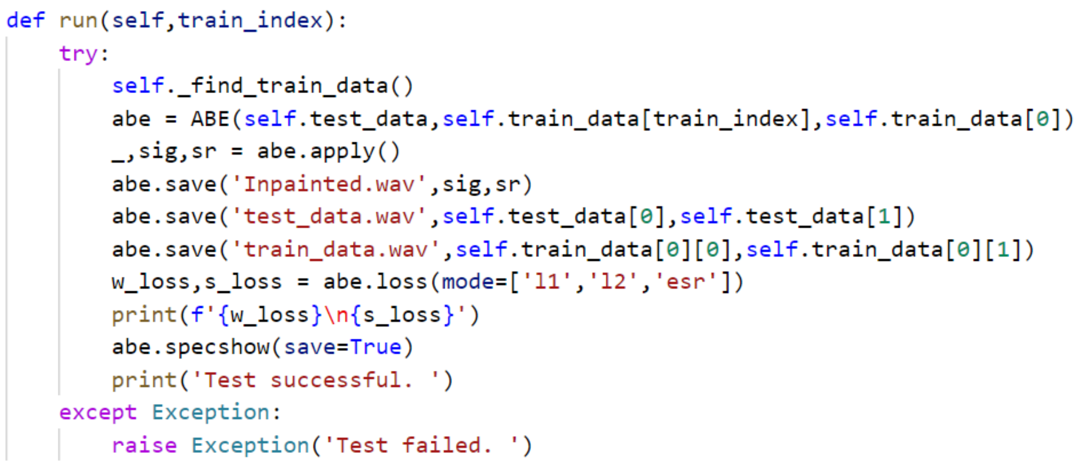


```
def _find_train_data(self):
    esr = 100
    for i in self.mix_path:
        Y = librosa.load(i,sr=44100,duration=10)
        abe = ABE(Y,self.train_data[0],self.train_data[0])
        _,spec_loss = abe.loss(mode=['esr'])
        if spec_loss['ESR'] < esr:
            self.train_data[1] = Y
            esr = spec_loss['ESR']

    for j in self.mix_path:
        Y = librosa.load(j,sr=44100,duration=10)
        abe = ABE(Y,self.train_data[0],self.train_data[0])
        _,spec_loss = abe.loss(mode=['esr'])
        if spec_loss['ESR'] > esr:
            self.train_data[2] = Y
            esr = spec_loss['ESR']
```

Figure 14 ‘test._find_train_data’ Method

4.1.4 test.run

This method is used to evaluate the quality of the model. If any function throws an exception, the test will fail, and the user will be warned. The testing includes preparing the training data, applying the model to test and training data, saving the audio to local paths, calculating different losses and plot log-Mel-spectrograms. It has a ‘train_index’ parameter for the user to specify what training data is used for this test. The ‘train_index’ accepts integers from 0 to 2, which the integers map the ground truth, the audio with highest spectral similarity and the audio with lowest spectral similarity.



```
def run(self,train_index):
    try:
        self._find_train_data()
        abe = ABE(self.test_data,self.train_data[train_index],self.train_data[0])
        _,sig,sr = abe.apply()
        abe.save('Inpainted.wav',sig,sr)
        abe.save('test_data.wav',self.test_data[0],self.test_data[1])
        abe.save('train_data.wav',self.train_data[0][0],self.train_data[0][1])
        w_loss,s_loss = abe.loss(mode=['l1','l2','esr'])
        print(f'{w_loss}\n{s_loss}')
        abe.specshow(save=True)
        print('Test successful. ')
    except Exception:
        raise Exception('Test failed. ')
```

Figure 15 'test.run' Method

4.1.5 Test results

1. **test.run(0)**: When using the ground truth as the training data, most of the missing high frequencies are retrieved:

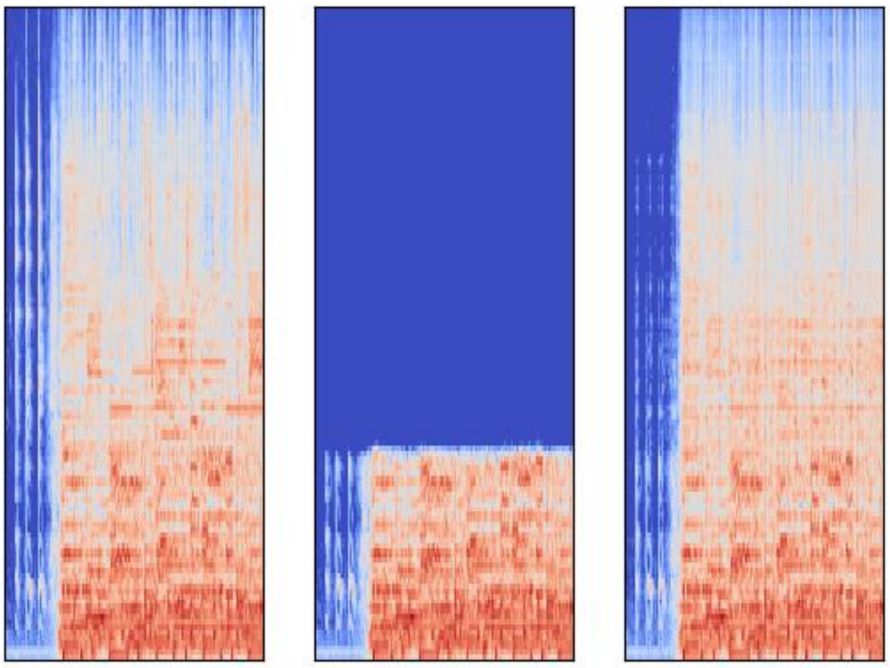


Figure 16 Ground truth (left), test data (mid) and inpainted audio (right)

The waveform and spectral losses, including L1 loss, L2 loss and ESR:

Waveform losses:

L1 Loss: 0.2509917800453515

L2 Loss: 0.13260821464002268

ESR: 1.9174937

Spectral losses:

L1 Loss: 1.027579365079365

L2 Loss: 11.905048752834468

ESR: 0.10944084

2. **test.run(1)**: When using the audio with the highest spectral similarity as the training data, some of the missing high frequencies are retrieved from the lower frequencies, but the inpainted audio is very distorted compared to the ground truth:

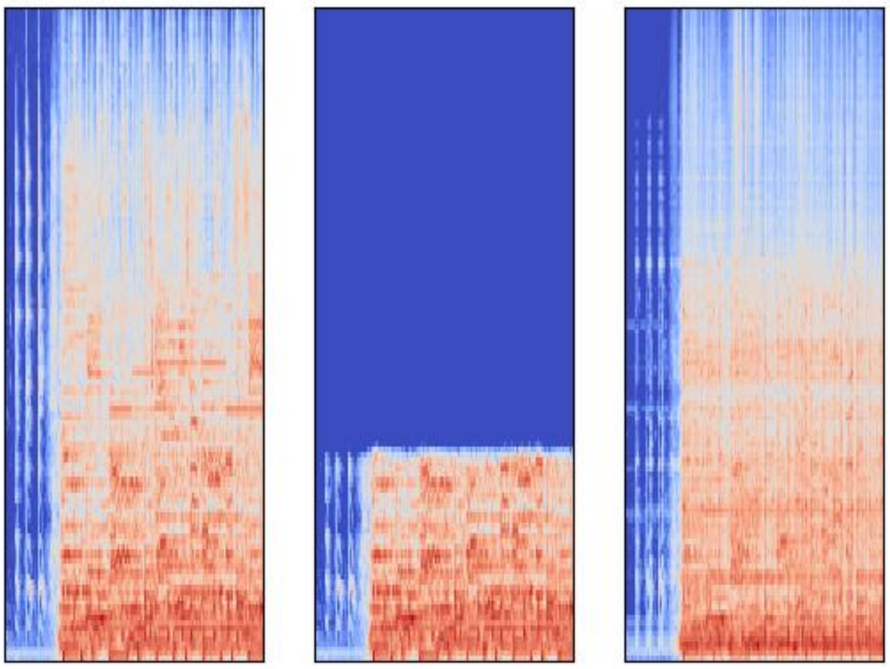


Figure 17 Ground truth (left), test data (mid) and inpainted audio (right)

The waveform and spectral losses, including L1 loss, L2 loss and ESR:

Waveform losses:

L1 Loss: 0.23005842545351474

L2 Loss: 0.10949203691893424

ESR: 1.5832375

Spectral losses:

L1 Loss: 1.742889314058957

L2 Loss: 20.51640136054422

ESR: 0.18860336

3. **test.run(2)**: When using the audio with the lowest spectral similarity as the training data, although partial high frequencies are restored, the lower frequencies are completely different from the ground truth:

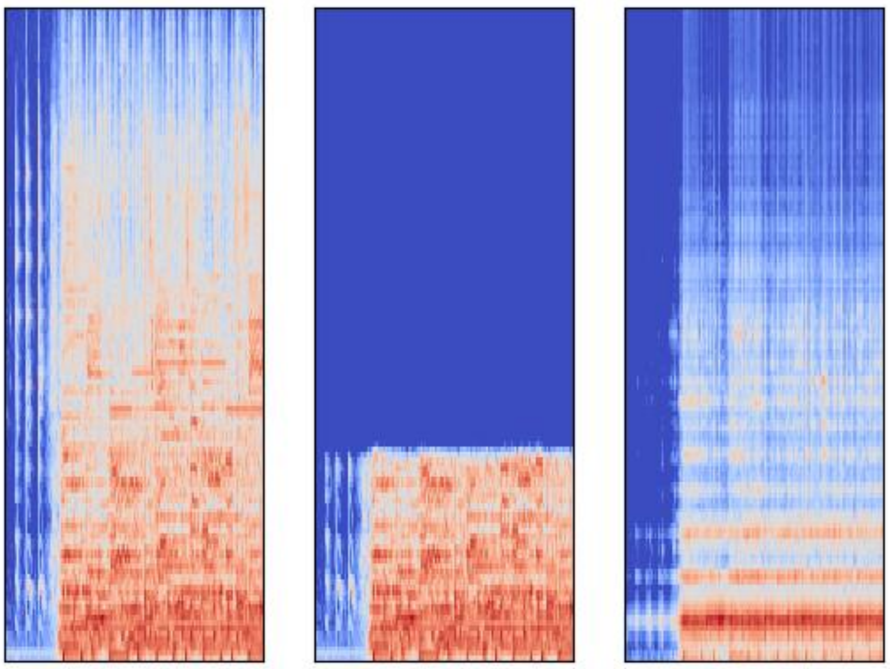


Figure 18 Ground truth (left), test data (mid) and inpainted audio (right)

The waveform and spectral losses, including L1 loss, L2 loss and ESR:

Waveform losses:

L1 Loss: 0.18436631944444445

L2 Loss: 0.06911524766156463

ESR: 0.99939555

Spectral losses:

L1 Loss: 9.472717120181406

L2 Loss: 392.9858684807256

ESR: 3.612644

Chapter 5: **Conclusion**

Based on the above testing, my spectrogram audio bandwidth extension model with NMF is able to restore the missing higher frequencies of an audio that is sampled at a low sample rate. The tests shows that the higher spectral similarity the training data has compared to the test data, the smaller the spectral Error-to-Signal ratio between the ground truth and the inpainted audio will get, which proves that the quality of the outcome is highly dependent on the choice of training data. On the other hand, the model can only be trained with pairs of test data and training data, and it cannot be applied to other audio. This means that it does not learn from previous training data. Hence, the accuracy of the model cannot be improved by fitting larger dataset, and it can only be optimised by updating the algorithms.

Compared to other audio bandwidth extension methods that use Deep Learning models, my model can meet its limitations when facing generalised datasets, and artifacts are introduced due to approximation errors during the process. This model can be further improved by adopting Deep Learning models such as Convolutional Neural Networks and other audio inpainting algorithms like Audio Source Separation to resolve the generalisation issue and reduce the amount of artifacts in the inpainted audio.

References

Amir Adler, Valentin Emiya, Maria Jafari, Michael Elad, Rémi Gribonval, et al., Audio Inpainting. IEEE Transactions on Audio, Speech and Language Processing, Institute of Electrical and Electronics Engineers, 2012, 20 (3), pp.922 - 932. 10.1109/TASL.2011.2168211. inria-00577079.

Ya-Liang Chang, Kuan-Ying Lee, Po-Yu Wu, Hung-yi Lee, Winston Hsu, et al., ArXiv:1911.06476 15 Nov 2019.

D. Griffin and Jae Lim, "Signal estimation from modified short-time Fourier transform," ICASSP '83. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1983, pp. 804-807, doi: 10.1109/ICASSP.1983.1172092.

Volodymyr Kuleshov, S. Zayd Enam, Stefano Ermon, et al., arXiv:1708.00853v1 2 Aug 2017.

Chih-Chung Chang and Chih-Jen Lin, “LIBSVM: A Library for Support Vector Machines”, 20 Jan 2021.

Breiman, “Random Forests”, Machine Learning, 45(1), 5-32, 2001.

Lee, Daniel D., and H. Sebastian Seung. "Learning the parts of objects by non-negative matrix factorization." *Nature* 401.6755 (1999): 788-791.

D. L. Sun and R. Mazumder, "Non-negative matrix completion for bandwidth extension: A convex optimization approach," *2013 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2013, pp. 1-6, doi: 10.1109/MLSP.2013.6661924.

R. Bittner, J. Salamon, M. Tierney, M. Mauch, C. Cannam and J. P. Bello, "MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research", in 15th International Society for Music Information Retrieval Conference, Taipei, Taiwan, Oct. 2014.

J. Olaya and C. Otman, "Beta-divergence for Nonnegative Matrix Factorization," 2021 International Conference on Digital Age & Technological Advances for Sustainable Development (ICDATA), 2021, pp. 15-22, doi: 10.1109/ICDATA52997.2021.00013.

Evan Radkoff, "Loss Functions in Audio ML", https://www.soundsandwords.io/audio-loss-functions/, 6 Sep 2021

Steinmetz, Christian J., and Joshua D. Reiss. "auraloss: Audio focused loss functions in PyTorch." *Digital music research network one-day workshop (DMRN+ 15)*. 2020.

Cédric Févotte and Jérôme Idier, arXiv:1010.1763v3, 8 Mar 2011.

Meinard Müller, “Fundamentals of Music Processing”, Berlin, Germany: Springer Verlag, 2015.

Stanley Smith Stevens, John Volkmann & Edwin B. Newman, "A scale for the measurement of the psychological magnitude pitch". Journal of the Acoustical Society of America, 1937.