Image Processing - Exercise 2

Shachar Cohen

Introduction

The goal of this exercise is to understand the significance and use of the STFT algorithm and of spectrogram analysis. More specifically, the exercise focuses on the creation and detection of audio watermarks. A successful watermark must be inaudible to human listeners while remaining detectable algorithmically. Achieving this required applying concepts from digital signal processing (DSP), particularly Fourier Transform, STFT algorithm and spectrogram presentation of frequency magnitudes over time.

According to Fourier's theorem, any periodic function can be represented as a weighted sum of sinusoidal components (sine and cosine functions) with different frequencies, amplitudes, and phases. This principle extends to audio signals, enabling us to decompose any audio input into a combination of sinusoidal functions.

To practically decompose the signal, we were expected to use the Fast Fourier Transform (FFT) algorithm. For a signal with N samples, FFT calculates the frequency components in the range 0 to $N-1$, but due to symmetry and periodicity, only the coefficients in the range 0 to $\frac{N}{2}$ are required to fully describe the signal. This property is crucial for understanding how a signal's spectrum is derived from its sampling rate: we learned that the human hearing spectrum spans from 50 Hz to 20,000 Hz. According to the Nyquist-Shannon theorem, the sample rate must be at least twice the bandwidth of the signal to avoid aliasing. Therefore, the sampling rate (SR) must be at least twice the maximum frequency of the signal to preserve the entire audible spectrum - at least 40,000 Hz.

Two of the main tools I used in this exercise are the STFT algorithm and spectrogram presentation of frequencies magnitudes over time. By using these tools, I could visualize and characterize the watermark added to the data. Specifically, I could add watermarks to only chosen time fragments in the original audio (ensuring memory efficiency) and evaluate their detectability.

Adding Watermarks

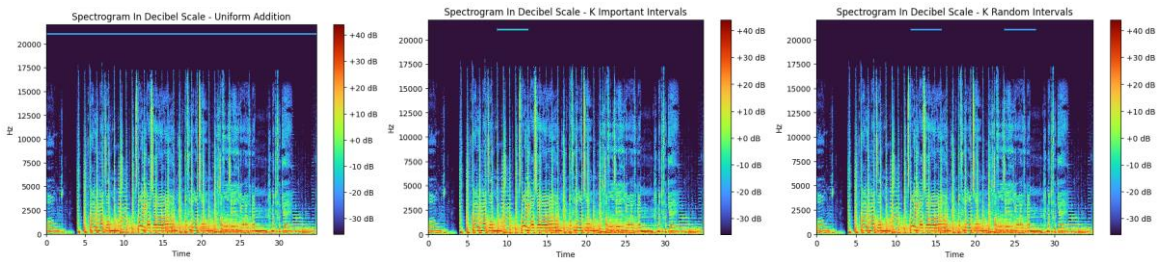
According to Wikipedia: "An audio watermark is a unique electronic identifier embedded in an audio signal, typically used to identify ownership of copyright… in a way that is difficult to remove... Watermarking has become increasingly important to enable copyright protection and ownership verification".
Since the exercise states that every audible watermark is bad, I concluded that a good watermark characteristics are:

1. Inaudible, meaning the STFT magnitude of any frequency lower than 20khz is zero at any point t in the audio data.
2. Unique, since it should be associated with a specific creator.
3. Detectable by an algorithm.
4. Efficient in memory and in time complexity.

I explored several options for a good watermark. First, I added a simple sinus function with a frequency of 21khz and amplitude of 10 to any point in the audio data. This is of course the simplest way to create a detectable inaudible watermark. However, it is not unique and not very efficient in memory, since it adds a value to every data point. Assuming the data contains N points and the value addition is of size M, we end up consuming additional $O(NM) \cong O(N)$ memory. One possible solution to the memory consumption problem is limiting the watermark addition to a set of K intervals of length L, thus reducing the additional memory consumed to $O(KLM) \cong O(1)$.

This leaves us with the question – what is the best choice of K intervals and length L? To remove a watermark, the audio data must be processed. That might damage the original signal, even if one removes only frequencies outside the human hearing range. Since the watermark should be hard to remove – we might want to choose the most important intervals in

the audio data. One reasonable choice method is taking the intervals that contain most of the information (e.g., highest sum of amplitudes). This choice method would take at least $O(N(K+F))$ additional time (F = the time it takes to calculate the importance value for every point). Another similar method is choosing K intervals randomly – assuming that if K and L are large enough, at expectation one of the intervals contain important information. This choice method would take $O(K)$ time. I chose the second method for its efficiency.



Spectrograms of the watermarks I explored

To make my good watermark unique, instead of adding a simple wave in higher frequencies, I created a triangle shape in the spectrogram. For the bad watermark, I chose to add a high-amplitude low-frequency (<10khz) wave to every point in the audio data. This choice is bad for several reasons: The first and the most important one is that it is audible, since the frequency of the wave is low. Frequencies lower than 10 khz (~40dBhz) are audible to most humans (except for ones with severe hearing loss). Therefore, adding a low frequency wave creates a significant noise in the audio which damages its quality. Second, it is not memory-efficient – high amplitude means M (the addition to every point) is relatively large and added to all the data points (+O(MN) memory).

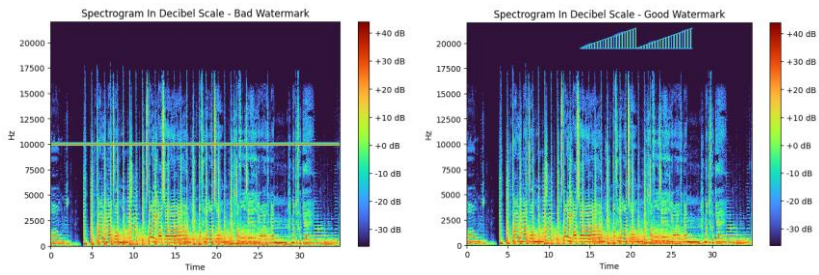
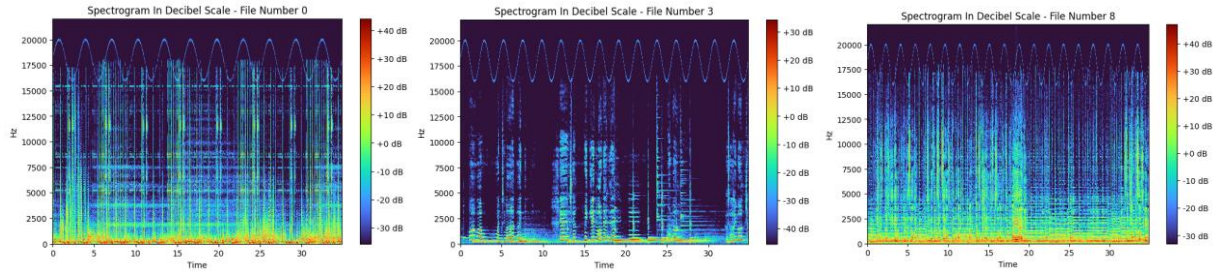


Image watermark – *"A digital watermark is called robust with respect to transformations if the embedded information may be detected reliably from the marked signal, even if degraded by any number of transformations. Typical image degradations are JPEG compression, rotation, cropping, additive noise, and quantization"* (Wikipedia). The main issues that should be taken into account when creating an image watermark are: (1) Minimal damage to the original image, (2) hard to remove by cropping, kernel filtering, or any other transformation (3) uniqueness, (4) detectability. Manipulating the image in the Fourier domain could be beneficial here as well[1] —high frequencies in the Fourier domain correspond to the fine details of the image. Therefore, adding a low-amplitude watermark to (relatively but not too-) high frequencies in the image might be beneficial, as it doesn't significantly distort the image while still remaining detectable. This kind of manipulation should also be robust to most image degradation, and especially to the typical degradations mentioned above.

Classifying watermarks

To detect the watermarks, I initially visualized the distribution of different frequencies over time using the STFT algorithm and generated a spectrogram for each file. I observed that the watermark leaves a sinusoidal imprint in the spectrogram of the audio file, as shown below

[1] https://www.sciencedirect.com/science/article/abs/pii/S003040262030396X



The spectrograms clearly show that all the added watermarks fall within identical the frequency range of approximately 16,250 – 20,000 Hz. All the watermarks seem to approximately have the same magnitude (same color at the spectrogram). Denote f_1 as the minimal frequency in the range, f_2 as the maximal frequency in the range, n the frequency of the added watermark, T the phase. Let x be a data point in the audio file. The watermark added to the spectrogram is of the following form:

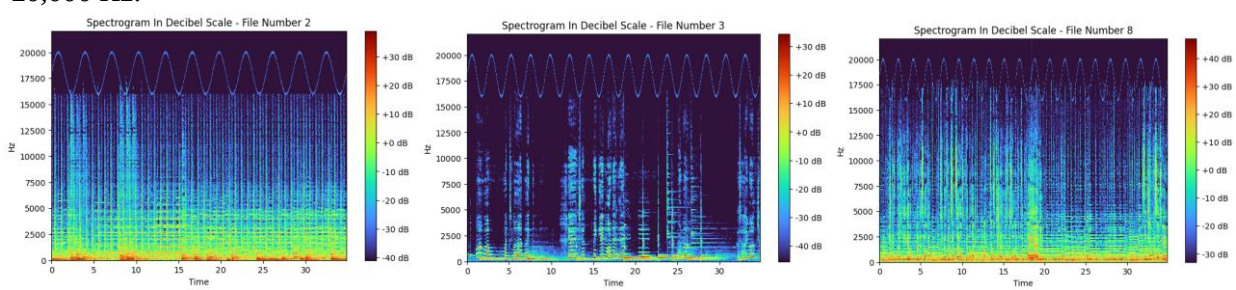
$$F(x) = |f_2 - f_1| \cdot sin(2\pi x \cdot n + T) \ + \ f_1$$

Therefore, the watermark added to the original audio at point x is of the following form:

$$W(x) = A \cdot sin(2\pi x \cdot F(x) + w)$$

Where A equivalent to the magnitude of the frequency $F(x)$ in the spectrogram before log-scaling. I assumed w constant since we were told the function is simple.

The spectrograms reveal clearly that the files can be divided into three different classes: 0-2, 3-5, 6-8, each class is characterized by a different frequency n. I each of these groups, I chose one file that contains the least noise in the range of 16,250 – 20,000 Hz:



The files I chose from each class

To approximate f_1, f_2, A I used the following algorithm:

1. Find a spectrogram that contains an interval with one peak and one through in the range $(16,\ 20)$ kHz that seems clear of noises (e.g., file number 3, seconds 27-30).
2. Let S(x, f) be the value of the spectrogram at point x and frequency f. Compute $G(f,x) = \ S(x,f) - \ S(x,f-1)$ using convolution. Find the values $f \in (16,\ 20)$ kHz the interval that satisfy $G(f,x) > \varepsilon$ for some x in the interval. Set $\hat{f}_1, \hat{f}_2$ to be the frequencies that satisfy the condition and maximize $\hat{f}_2 - \hat{f}_1$.
3. Set A to be the magnitude of $\hat{f}_2$ at the peak in the interval.

To approximate n, T for each class, I used the following algorithm:

1. Assume f_2, f_1 are known. Let S(x, f) be the value of the spectrogram at point x and frequency f with respect to the audio file with the least noisy spectrogram at the range (f_1, f_2) in the current class.
2. Set $D(x) = S(x, f_2) - S(x-1, f_2)$ (can be computed by convolution). Set $\hat{n}$ to the most frequent distance between two consecutive points that satisfy $D(x) > 0$.
3. Note that if $T = 0$ then the first peak of $F(x)$ should be at $\frac{n}{4}$. Let y be the first (smallest) data points that satisfy $D(x) > 0$. Set $\hat{T} = y - \frac{n}{4}$.

Approximation of w: Notice that if the only signal of frequency $F(x)$ at point x derives from the watermark function, then the phase of $F(x)$ must be w. Therefore, we a find "clean" watermark point at one of the spectrograms and compute its phase using the STFT algorithm.
The values I received are:

$$A \cong 1e-06$$
$$f_1 \cong 16kHz, \qquad f_2 \cong 20kHz$$

1. Files 0-2: $n = \frac{\#peaks}{len-audio} \cong \frac{11}{1323008} \cong \frac{11}{1{,}300{,}000}, T \cong 255$
2. Files 3-5: $n = \frac{\#peaks}{len-audio} \cong \frac{15}{1321728} \cong \frac{15}{1{,}300{,}000}, T \cong 255$
3. Files 6-8: $n = \frac{\#peaks}{len-audio} \cong \frac{19}{1323008} \cong \frac{19}{1{,}300{,}000}, T \cong 511$

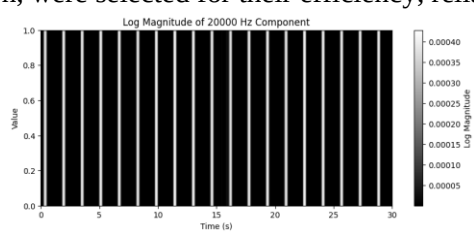
With respect to time t (and not data point x), F(t) in each of the classes comes out as approximately:

1. $|f_2 - f_1| sin(2\pi t \cdot \frac{11}{30}) + f_1$
2. $|f_2 - f_1| sin(2t \cdot \frac{15}{30}) + f_1$
3. $|f_2 - f_1| sin(2\pi t \cdot \frac{19}{30}) + f_1$

<u>Implementation of classification algorithm:</u>

1. Apply STFT on the audio file using scipy.signal algorithm. Take the magnitudes of the frequencies at each time point (abs(stft)).
2. Extract the row equivalent to 20kHz (my implementation).
3. Apply a gaussian kernel to reduce noises using scipy.ndimage.gaussian_filter1d using a hyper parameter sigma=5, which makes the blurring significant.
4. Use scipy.signal.find_peaks with width = 70, which is used to filter peaks based on their width (i.e., the horizontal distance of the peak in terms of the number of samples), and prominence of 1e-5. For our problem properties, 70 is a reasonable width (doesn't include two peaks, yet not too narrow). 1e-5 prominence works the best with the given data since the values are relatively small.
5. If the number of peaks is 11, classify as class 1. 15 – class 2. 19 – class 3.

The chosen libraries and methods, such as scipy.signal for STFT, scipy.ndimage for Gaussian filtering, and scipy.signal.find_peaks for peak detection, were selected for their efficiency, reliability, and domain-specific suitability.



Visualization of the 20khz log-magnitude values after applying the gaussian kernel

<u>Removing the watermark:</u>

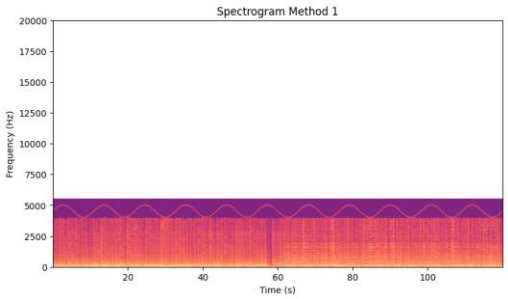
To remove the watermark while retaining most of the original information, I would apply noise reduction methods discussed in class, specifically targeting the areas where the watermark was likely embedded.

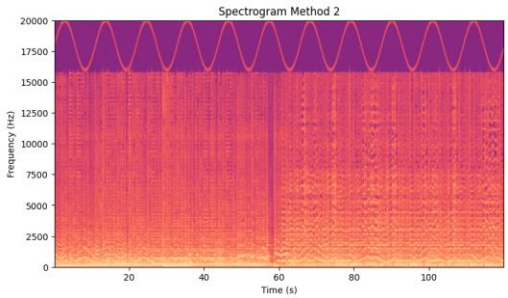
I would begin by approximating the data points (x,f) [data, frequency] where the watermark was added, based on the previous analysis. Using a similar **peak detection approach** as in the classification algorithm, I would also estimate the **maximum vertical width of the watermark** — that is, the maximum number of consecutive (x,f) points where the watermark signal was applied. Denote this variable by W. For each suspected watermark point, I would then replace the magnitude at (x,f) with the first quartile of a vertical kernel at width 2W. This operation effectively assigns a value close to the median of the surrounding region, excluding the watermark itself, thereby minimizing the watermark's impact while preserving the original signal.

Determining speedup method

From both perceptual and technical standpoints, the differences between the audio files are significant. Perceptually, the first audio file sounds distorted and slower, while the second audio file is clearer and less distorted.

To examine the technical differences, I applied the Short-Time Fourier Transform (STFT) and generated spectrograms for both files. The spectrogram of the first file appears “compressed” compared to the second — it displays the same patterns but within a lower frequency range. This suggests that the first audio file was sampled at a lower rate, as indicated by the Nyquist-Shannon theorem. Specifically, while the second spectrogram spans 0-20 kHz, the first spans only 0-5 kHz, indicating that the first audio file was sampled at a rate that is 4 times lower. That means that the first file was slowed down 4 times in the time domain, while the other file was slowed down 4 times in the frequency domain. Slowing down in the frequency domain means “stretching” the frames of Fourier transform in the STFT – spanning the signal (frequencies) in each datapoint x over 4 datapoints. The frequency content of each frame remains unchanged. The phases of the frequency components are adjusted to ensure smooth transitions between frames, preserving the pitch of the original signal.
Slow-down factor: 4. Method 1: time domain, Method 2: frequency domain.





Conclusion

In this exercise, I explored the creation, detection, and removal of audio watermarks using digital signal processing techniques. By leveraging the Short-Time Fourier Transform (STFT) and spectrogram analysis, I successfully embedded both "good" and "bad" watermarks and developed a classification algorithm to detect them based on their frequency characteristics. The exercise highlighted the importance of Fourier analysis and the Nyquist-Shannon theorem. For watermark removal, I applied noise reduction methods and quartile-based filtering to target specific high-frequency regions while preserving the integrity of the original audio. This experience provided valuable insights into balancing detectability, inaudibility, and computational efficiency in audio watermarking.