Image Processing - Exercise 2

Liel Amar, liel.amar, 211771993

Introduction

In this assignment, we were to create two algorithms which detect and cut out noises in given audio files. This is done through the usage of Fourier Transform and analyzing frequency graphs and spectrograms. More specifically - detecting out-of-place frequencies that do not match what we would expect in a clean audio file.

As for the differences between the two given types of noises, we hear that the first type of noise is a continuous, high-pitched (high-frequency) noise that is constantly playing at the same rate. However, the second type of noise is a changing noise that spreads across multiple frequencies with different magnitudes. It is rather a low-pitched (low-frequency) noise, a fact which helped me detect the noise. The distinction between the two different types of noise made it difficult to come up with a single algorithm to handle both cases, so instead, I decided to use two different algorithms.

Algorithm

The algorithm I used to detect the first type of noise is the following:

1. Importing the given audio file
2. Computing the Fourier Transform of the audio
3. Finding the frequency with the maximum magnitude
4. Taking the frequency from the previous step and its symmetric as the frequencies are mirrored, thus the symmetric frequency has the same, maximum, magnitude
5. Zeroing out the magnitude of the two frequencies
6. Computing the Inverse Fourier Transform
7. Saving/Returning the newly created audio file

To implement the above algorithm, I used three existing libraries, which are: *numpy*, *scipy*, and *matplotlib*. I chose these libraries specifically because either I have experience using them, or they were recommended to me by the course staff.

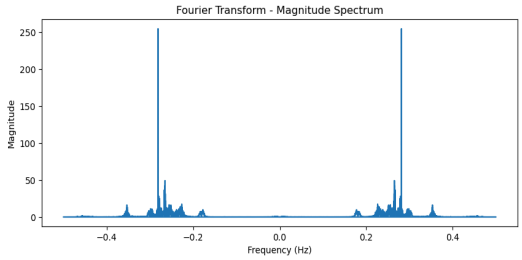
Let's describe how each step was implemented:

1. Using *scipy.io.wavfile*, and specifically the *#wav.read* function, I imported the audio file as a 1-dimensional list of length 9600, each representing a sample.

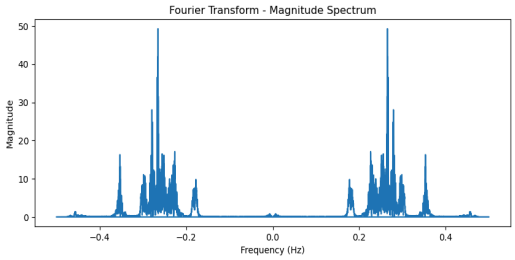
2. Then, I used *numpy.fft.fft* to compute the one-dimensional Fourier Transform of the said list. This resulted in a list of the same length (9600) such that each entry holds the matching Fourier coefficient (a complex number) for each frequency.
3. To find the index of the frequency with the maximum magnitude, I used *numpy.argmax* with the input being the absolute value of each entry in the list of Fourier coefficients. The absolute value is the magnitude of each frequency.
4. When we studied about Fourier Transform we were told that it is periodic and symmetric. Thus, besides the found index representing the maximum magnitude, I also wanted to grab the index of the symmetric frequency. This was done by taking the found index, and the symmetric index which is the length of the list, minus the found index (*len(fourier_coefficients) - found_index*).
5. Afterwards, I zeroed-out the magnitude of the two indexes, which previously had the highest magnitude.
6. To compute the Inverse Fourier Transform, I used *numpy.fft.ifft* and passed it the new Fourier coefficients.
7. Lastly, I return the result of IFT, which is the newly de-noised version of the audio.

As described above, I made a few choices based on the material learned in class. Mainly, zeroing-out both the found frequency and its symmetric-matching frequency.

A challenge I faced at the beginning was finding the frequency(ies) that causes the noise. I decided it would be best to plot out the frequencies and their magnitude, as well as the spectrogram of the audio.



(A) Fourier Coefficients before denoising



(B) Fourier Coefficients after denoising

Indeed, the plots helped me gather some insights about the data.
In *Figure A* we can see that there's one frequency with a huge spike in its magnitude, implying that this might be a frequency causing some of the noise. Moreover, when I looked at the spectrogram, I realized that the noise is continuous as I suspected, since there's a long line around the ~1125 frequency, throughout the whole time axis.
I decided to check out if the frequency with the largest magnitude in *Figure A* has any relations with the continuous frequency I could see in the spectrogram, and when I zeroed-out the magnitude of that frequency (and its negative, as Fourier Transform is symmetric), and plotted out the graphs once again, I saw a huge difference. I got a

much cleaner spectrogram, without the continuous, green line. This implies that the sound that was part of every window is now gone, and it makes perfect sense that this sound was actually the noise. As for the frequency graph - we could tell that the frequencies distribute much better now, and there isn't one frequency with a spike. After listening to the output file, we could immediately tell that the noise has been removed to a large extent.

The algorithm I used to detect the second type of noise is the following:

1. Importing the given audio file
2. Computing the Short-Time Fourier Transform of the audio
3. If we let x be the number of windows, I loop over all windows between 3(x/10) and x, because it's possible to identify that the noise doesn't immediately start, but rather roughly around 1.5 seconds into the audio.
4. Finding all frequencies between 550 and 650, starting from around 1.5 seconds.
5. Zeroing out the magnitude of the found frequencies
6. Computing the Inverse Fourier Transform
7. Saving/Returning the newly created audio file

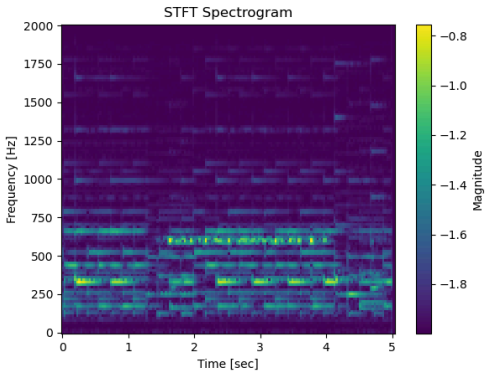
To implement the above algorithm, I used the same three libraries, for the same reasons. These libraries are: *numpy*, *scipy*, and *matplotlib*.

Let's describe how each step was implemented:

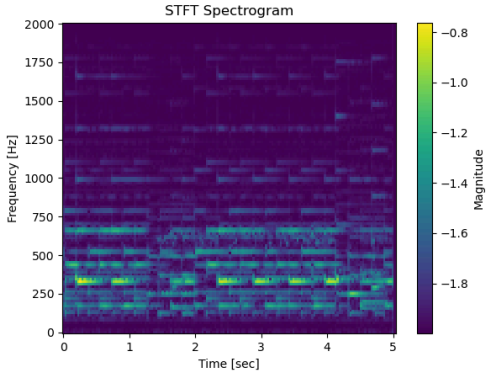
1. Using *scipy.io.wavfile*, and specifically the *#wav.read* function, I imported the audio file as a 1-dimensional list of length 9600, each representing a sample.
2. Then, I used *scipy.signal.stft* to compute the Short-Time Fourier Transform of the said list. This resulted in a list of frequencies, a list of times (windows) and a 2D list, representing the Fourier Coefficients of all frequencies in each window.
3. I used a good old for loop to loop over all windows that are relevant to our case.
4. In each iteration, I found all frequencies in the specific range (550-650).
5. Afterwards, I zeroed-out the magnitude of the found frequencies, which previously had caused the noise
6. To compute the Inverse Fourier Transform, I used *scipy.signal.istft* and passed it the new Fourier coefficients and the sample rate.
7. Lastly, I return the result of ISTFT - the newly de-noised version of the audio.

As described above, I used a threshold and two hyper-parameters, the former is used to determine the frequencies to zero out and the latter is to loop over frequencies that might be causing noise.
A challenge I faced at the beginning was finding the frequency(ies) that caused the noise. I decided it would be best to plot out the frequencies and their magnitude in a

spectrogram, and indeed, doing so helped me notice that there are frequencies that seem out of place as they don't have as big of a magnitude as other frequencies, and also they start exactly when the noise starts in the audio file (determined by listening to the audio file).



(E) Fourier Coefficients spectrogram before denoising



(F) Fourier Coefficients spectrogram after denoising

As can be seen above in *Figure E*, before denoising, I noticed that around 1.5 seconds into the audio, there are some high-frequencies with a very large magnitude (compared to other frequencies) starting to appear. This matched my suspicions, because when I listened to the file, indeed the noise started about this much after the beginning of the audio. I noticed that all of these frequencies are rather high (between 550 and 650), and these are the ones I zeroed-out in the algorithm specified above. After zeroing out these frequencies, looking at *Figure F*, it's obvious that the frequencies indeed look cleaner and match what we expect. When I outputted the new audio to a file and listened to it, I was able to tell it's much cleaner and the noise is almost non-existent - thus, confirming my suspicions of the cause of the noise.

Conclusion

To sum up all our findings and insights, we can conclude that we had to deal with two types of noises, and each had different characteristics. The first one was a continuous, high-pitched sound, which we were able to filter out pretty easily by finding the frequency matching that sounds and zeroing-out its magnitude. The second type of noise was tricker but not much harder. We had to determine which frequencies at which times might have caused the noise, and then filter these out as well.

The two outputs were cleaner than the input files we were given, and indeed, denoising seemed to work just as expected.

The only thing bothering me is the fact that the designed algorithms solve a very specific problem and are not modular and dynamic. They use hyper-parameters and insights that are specific to the given two types of noises, and they most likely won't work well on different, noisy audio files.