EE200 Endsem Practical

Kumar Aditya (220559)

July 5, 2025

1 Question 1. Frequency mixer: 'Beauty and the Blur'

In the **first problem**, we were given the task of creating a **hybrid image** which is done by fusing two images together, one of which has been passed through a high-pass filter and the other through a low-pass filter. These filters are applied in the **frequency domain**. Thus, one image has its **low frequencies** suppressed which give the overall **shape & appearance** of the object present in the image while the other one has its **high frequencies** suppressed that contain finer details such as edges, etc.

Thus, the final **hybrid image** depicts two different objects depending upon the distance from which it is seen. The original work was done by **Oliva et al.** [1]
In this assignment, I have not followed the exact methodology given in that paper but used a much simpler approach using **Discrete Fourier Transform** only.

1.1 Basic Theory

As we know that **Fourier Transform** helps us to analyze any signal in the frequency domain. Although it may be easier to visualize the Fourier transform for the **1D signal** it may not be that intuitive for a **2D signal** such as an image. We can look at the figures attached below to get a better understanding of the **2D discrete Fourier transform**.

The corresponding **2D DFT** equations pair is:

Forward 2D DFT:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cdot e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

Inverse 2D DFT:

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) \cdot e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

where:

- $f(x,y)$: spatial-domain signal
- $F(u,v)$: frequency-domain representation
- M: length of the signal in the x-direction
- N: length of the signal in the y-direction

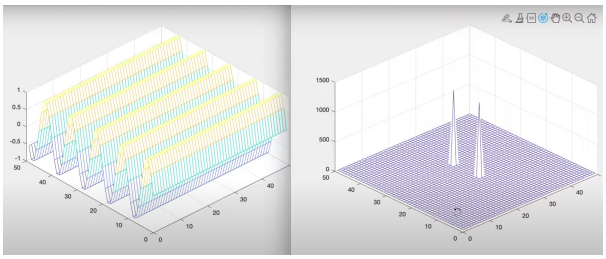

Figure 1: 2D Signal with variation in only y-dirn.

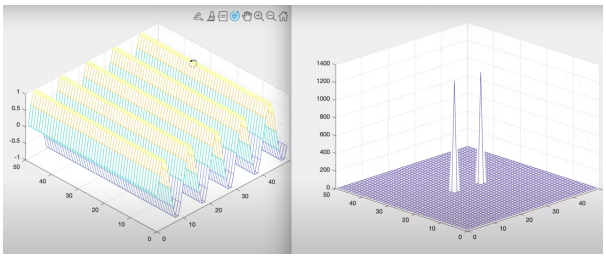

Figure 2: 2D Signal with variation in only x-dirn.

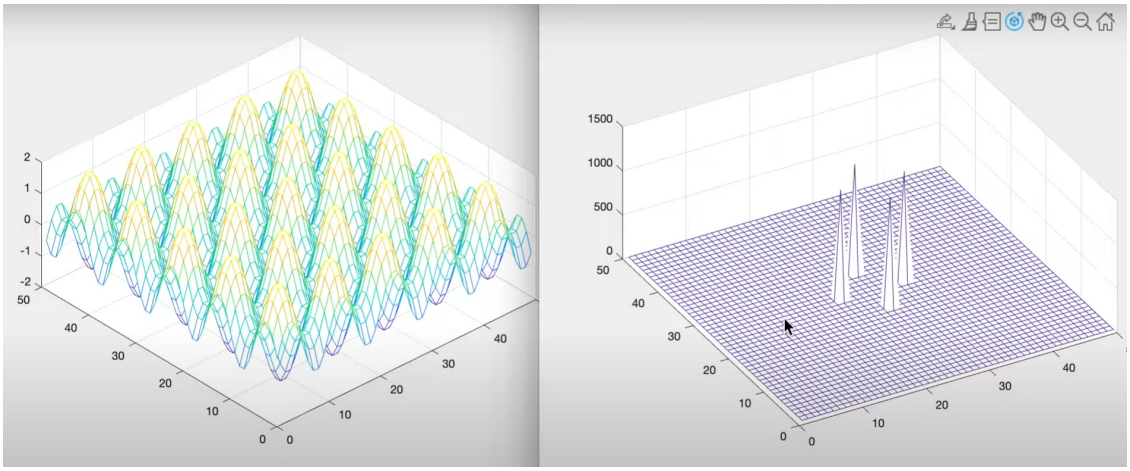

Figure 3: 2D Signal with variation in both x & y dirn.

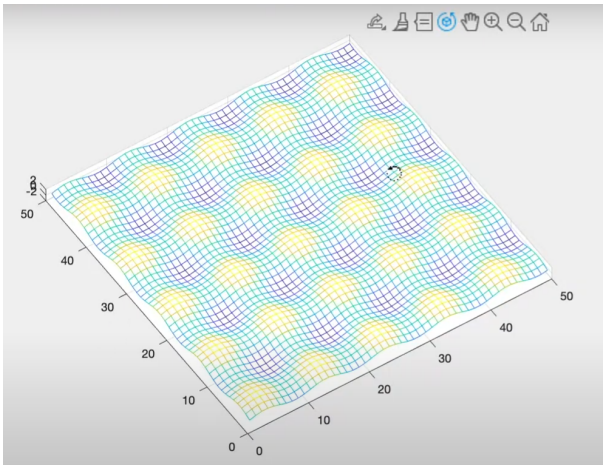

Figure 4: Visualization of an image as 2D signal

Ref: 2D Fourier Transform Explained with Examples

1.2 Methodology

- The first step often involves converting the given images to **gray-scale** but here, we were already provided with grayscale images but it contained **3 channels**. So, we need to convert it to **single-channel**. We can do the operation with **multi-channels** as well but in that case we need to find the FFT and do other operations on all the channesl present.

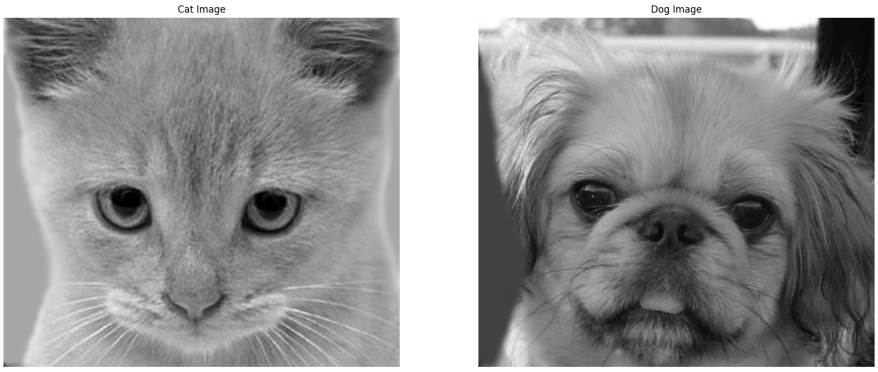

Figure 5: Single channel gray-scale image of Cat & Dog

- Then, I took the **Fast Fourier Transform** [2] of both the images using in-built **Numpy** functions.
- By deafault, the **Fourier Transform** which we get is not centred around **low frequencies** instead the low frequencies are present at the **top-right** corner. So, I first shifted the **Fourier Transform** which I got after which I got a better visualization.
- Since the Fourier Transform contained complex values, I took the magnitude of it and plotted that signal. I didn't get anything on directly plotting the magnitude because most of the magnitudes were of very **less order**. So, I then plotted the **Bode Magnitude plot** instead, which is a graphical representation of a system's frequency response which is basically a plot of:

  $$20 \log_{10} |H(e^{j\omega})|$$

  which shows the gain of the system in decibels (dB) as a function of angular frequency ω on a logarithmic scale.
- I then rotated the images by **90°** and found their Fourier Transform as well.
- Next step was to apply appropriate filter to both the images. I decided to apply the **high-pass filter** to the **cat image** and **low-pass filter** to the **dog image**.
- I used a **Gaussian filter** for this purpose. Since all the operations are happening in the **frequency domain**, we can directly multiply the filter with the Fourier Transform of the image to get the filtered result.

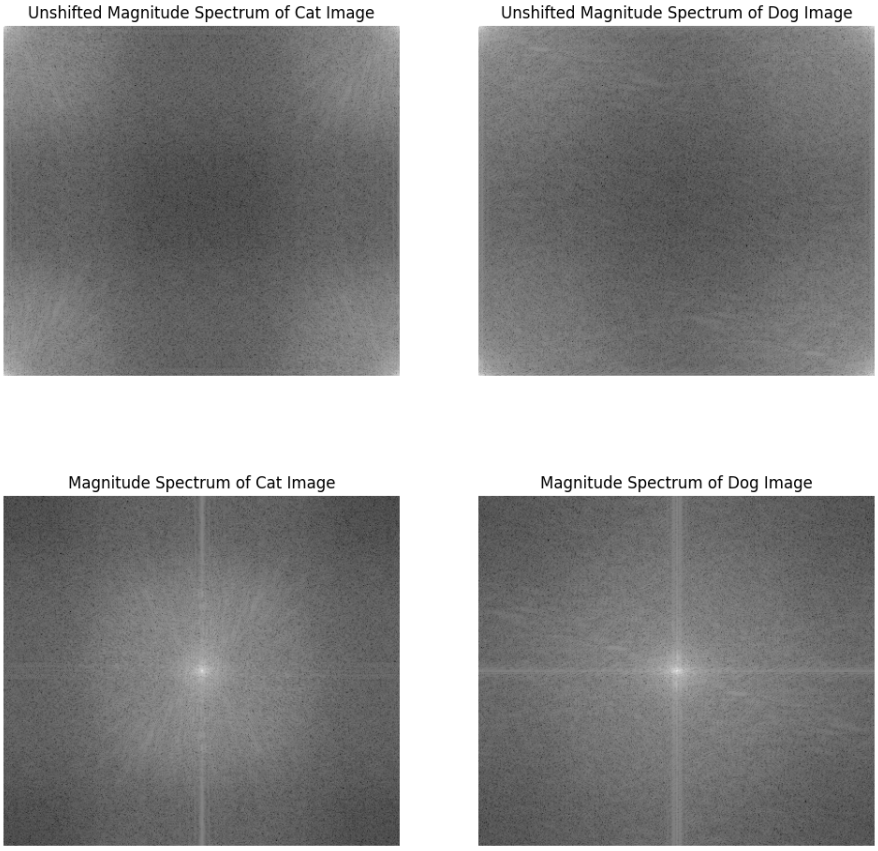

Figure 6: Magnitude Spectrum of FFT of the given images: (Top) Unshifted spectrum, (Bottom) Center-shifted spectrum

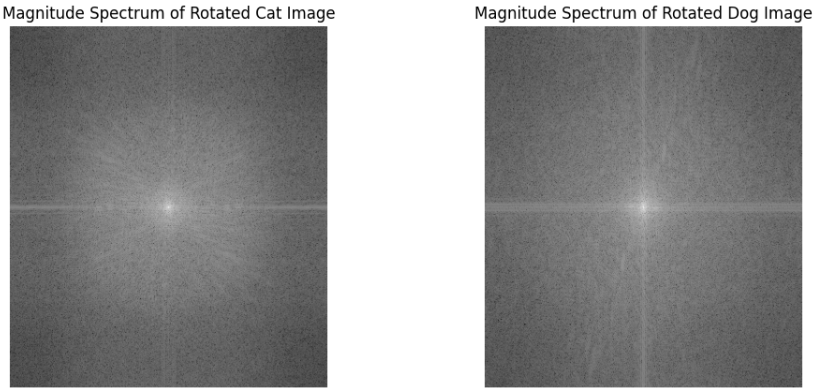

Figure 7: Rotated magnitude spectrum of the hybrid image

Gaussian Low-Pass Filter (GLPF):

$$H_{\text{low}}(u, v) = e^{-\frac{D(u,v)^2}{2\sigma^2}}$$

Gaussian High-Pass Filter (GHPF):

$$H_{\text{high}}(u, v) = 1 - e^{-\frac{D(u,v)^2}{2\sigma^2}}$$

where:

- $D(u, v) = \sqrt{(u - M/2)^2 + (v - N/2)^2}$: distance from the frequency center
- σ: standard deviation that controls the cut-off sharpness
- M, N: dimensions of the frequency domain (same as image size)
- $H(u, v)$: filter transfer function

- For **high-pass filter**, I chose the value of $\sigma_{high} = 5$ and for **low-pass filter**, I chose the value of $\sigma_{low} = 20$. I came to these values by manually fine-tuning them to get best result.
- After applying filters, I took **Inverse DFT** for both images to get our **two starting images** for the final hybrid image.

Figure 8: FFT-based filtered outputs of the hybrid image using Gaussian high-pass and low-pass filters

- Finally I fused both the images with different weights to get the final **hybrid image**. I chose $\boldsymbol{w}_{\text{low-freq}} = 0.6$ and $\boldsymbol{w}_{\text{high-freq}} = 0.4$ again by manually fine-tuning to get the best result.

1.3 Results

- One of the **key observation** was that on rotating the image, the corresponding **Fourier Transform** also gets rotated.
- Since we're multiplying the Fourier Transform of input image (input 2D signal) with **Gaussian filter**, it would be the corresponding **transfer function** H($e^{j\omega}$) as well. The corresponding plof of the **Gaussian filter/transfer function** for both the images are attaches below:

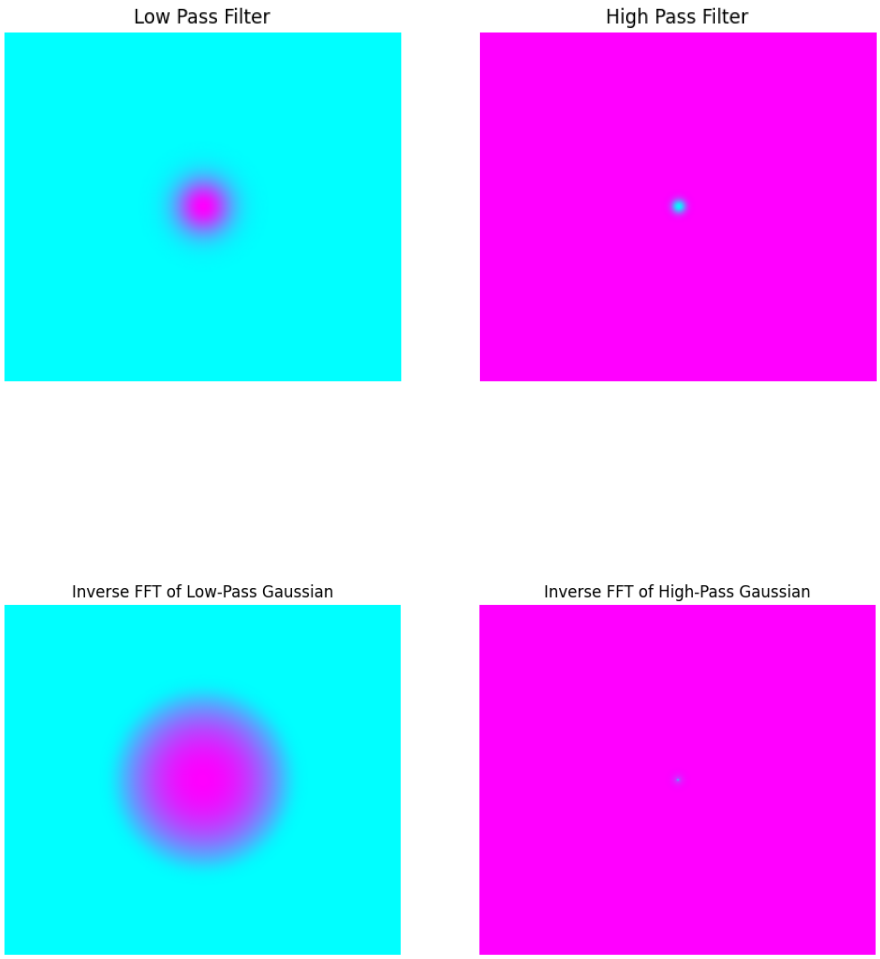

Figure 9: Top: Gaussian filters in the frequency domain. Bottom: Corresponding inverse Fourier Transforms in the spatial domain.

- The final **hybrid image** is:

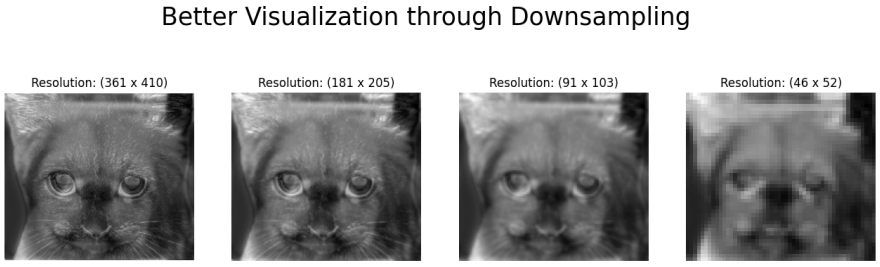

Figure 10: Top: Original hybrid image combining low- and high-frequency content. Bottom: Downsampled hybrid image showing resolution-aware perception.

2 Question 2. Frequency de-mixer: 'Unwanted Solo'

In the **second problem**, we were given an **audio signal with noise** and we were required to denoise the audio signal using **Fourier Transform** and associated principles.

2.1 Basic Theory

The basic theroy remain same as in the case of **2D Signal**.

The corresponding **1D DFT** equations pair is:

Forward 1D DFT:

$$X(k) = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi}{N}kn}$$

Inverse 1D DFT:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \cdot e^{j\frac{2\pi}{N}kn}$$

where:

- $x(n)$: discrete input signal in the time domain
- $X(k)$: Discrete Fourier Transform (frequency-domain representation)
- N: total number of samples
- k: frequency index (ranging from 0 to $N-1$)
- n: time index (ranging from 0 to $N-1$)

In case of audio signal, one important terminology is **sampling rate**. In real life, audio signals are continuous but when they're recorded they have to be discretized by taking samples only at some fixed duration. The inverse of **that fixed duration** is called **sampling rate**. For example, if an audio signal has **sr = 100Hz**, it means that **100 samples** are being taken **per second**. Most of the time, the sampling rate is given with the audio itself.

In this problem, I have used an another transform called **Short Time Fourier Transform (STFT)** as well with the usual **1D DFT**. STFT is an important technique to analyse those signals whose **frequency content** change over time. Unlike the regular Fourier Transform (which gives frequency content over the entire signal), the STFT computes the frequency spectrum of local segments (windows) of the signal.

Discrete Short-Time Fourier Transform (STFT):

$$\text{STFT}_x(m, \omega) = \sum_{n=-\infty}^{\infty} x[n] \cdot w[n-m] \cdot e^{-j\omega n}$$

Discrete Inverse STFT:

$$x[n] = \sum_{m} w[n - mR] \cdot \Re \left\{ \frac{1}{N} \sum_{k=0}^{N-1} X_m[k] \cdot e^{j \frac{2\pi}{N} kn} \right\}$$

where:

- $x[n]$: Original discrete-time signal
- $w[n - m]$: Window function (e.g., Hann or Hamming), centered at time index m
- m: Frame index (window position in time)
- R: Hop size (stride between successive windows)
- N: FFT size (window length)
- ω: Angular frequency (in radians/sample)
- $X_m[k]$: Discrete Fourier Transform of the m^{th} windowed segment
- $\text{STFT}_x(m, \omega)$: Frequency content of the signal at time m and frequency ω

2.2 Pre-processing

- First, I read the given audio file and its sampling rate using the **Librosa** library in Python.
- Then, I plotted the signal waveform using **Matplotlib** after normalizing the audio magnitudes.
- Then, I found the **FFT** (Fig. 12), **STFT** (Fig. 11) & **Power Spectral Density** (Fig. 13) of the given signal and plotted them as well.

2.3 Methodology

- After carefully observing the **PSD plot** and **FFT plot**, we can see that there are many peaks between the frequencies range from **1000Hz** to **6000Hz**.
- This hints towards the fact that most of the noise might be contained in these frequencies.
- So, we need to apply a **bandstop filter** which will suppress this frequency range as possible.
- I used **Butterworth filter**[3] for this purpose as it is the most close-to-ideal filter available and is directly applied to the signal itself in its time domain.
- We can set the **order** for this filter which basically controls how steep the transition from **passband** to **stopband** will be.
- For this task, I chose **order = 6** with **lower-cutoff = 1000Hz** and **upper-cutoff = 6000Hz**.
- After applying the filter, I once again plotted the **PSD**, **FFT** and **STFT** for the filtered signal to draw comparison from the original given signal and see the effects of filtering.

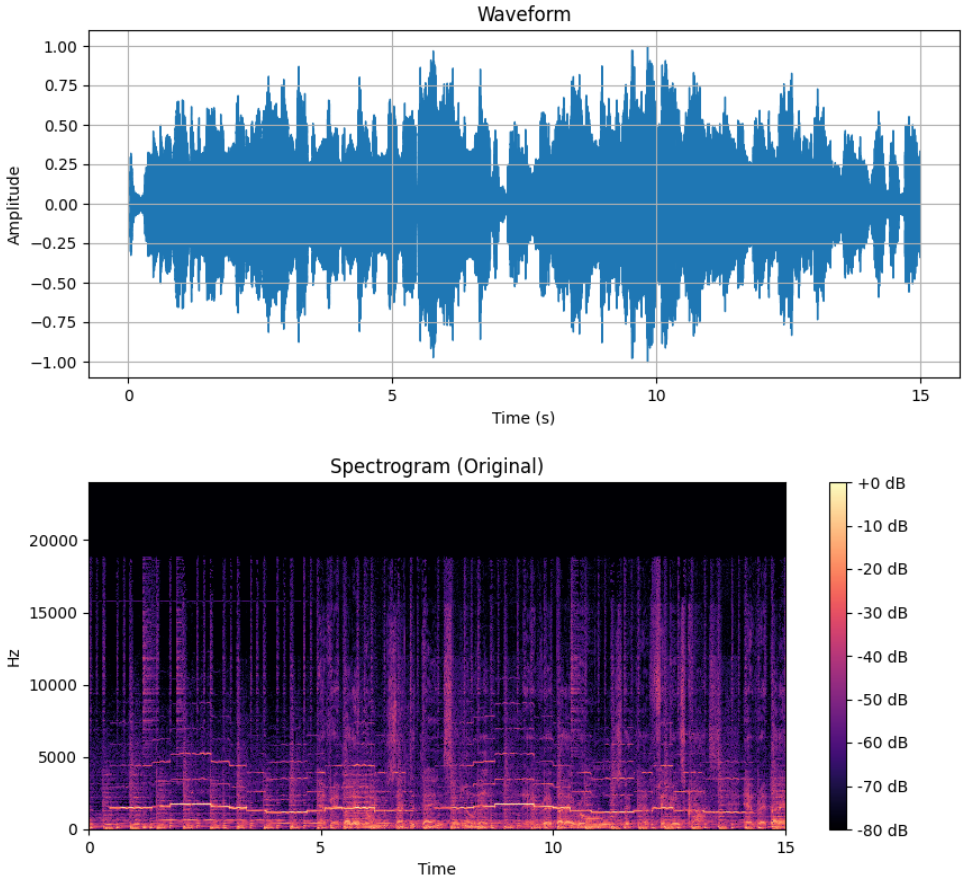

Figure 11: (Top): Audio waveform (Bottom): Spectrogram of the audio signal

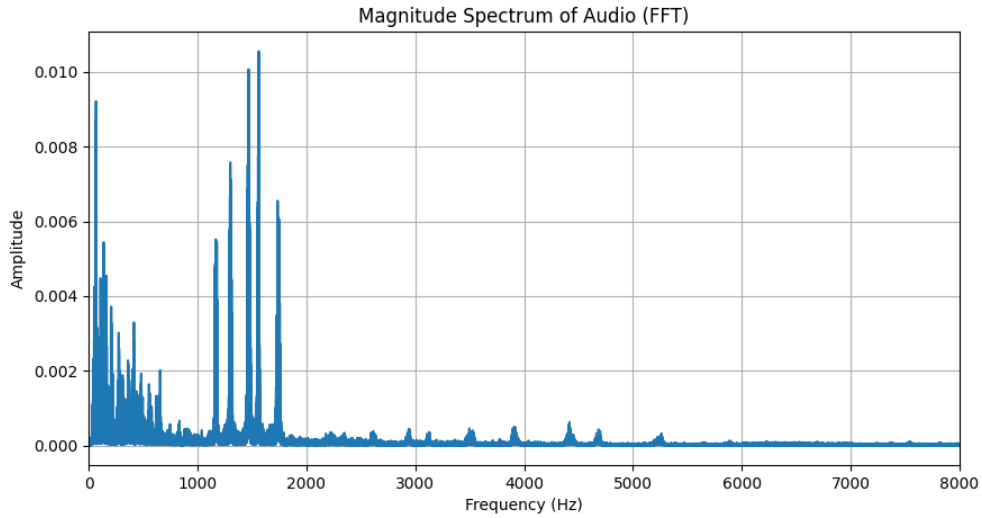

Figure 12: Magnitude spectrum from FFT

2.4 Results

- After filtering, most of the noise from the audio signal was suppressed.
- From the **PSD plot** & **FFT plot** also, it can be seen that much of the frequencies in the stopband range defined above were suppressed nearly completely.
- The final plots for the audio signal after applying filter can be found below.
- The final cleaned audio file can be found here.
- The **Python** notebook can be find at this link.

References

[1] A. Oliva, A. Torralba, and P. G. Schyns, "Hybrid images," in *ACM Transactions on Graphics (TOG)*, vol. 25, 2006, pp. 527–532.

[2] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965,
Fast Fourier Transform is basically an algorithm to find Fourier Transform in an efficient manner using computers.

[3] S. Butterworth, "On the theory of filter amplifiers," *Wireless Engineer*, vol. 7, no. 6, pp. 536–541, 1930,
This foundational paper introduces the Butterworth filter, known for its maximally flat frequency response in the passband, making it ideal for smooth filtering applications.

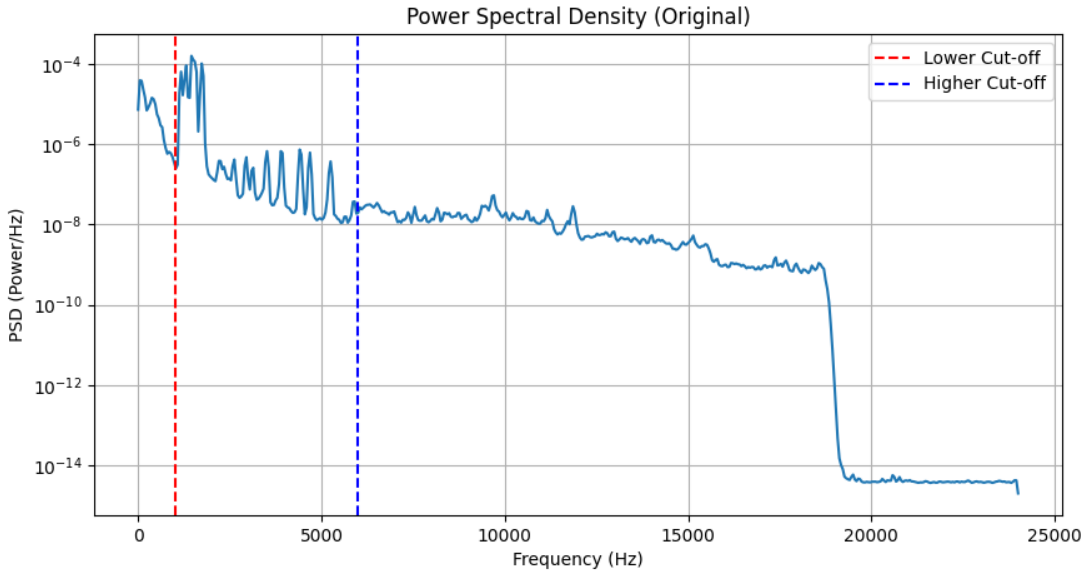

Figure 13: Power Spectral Density of the original signal

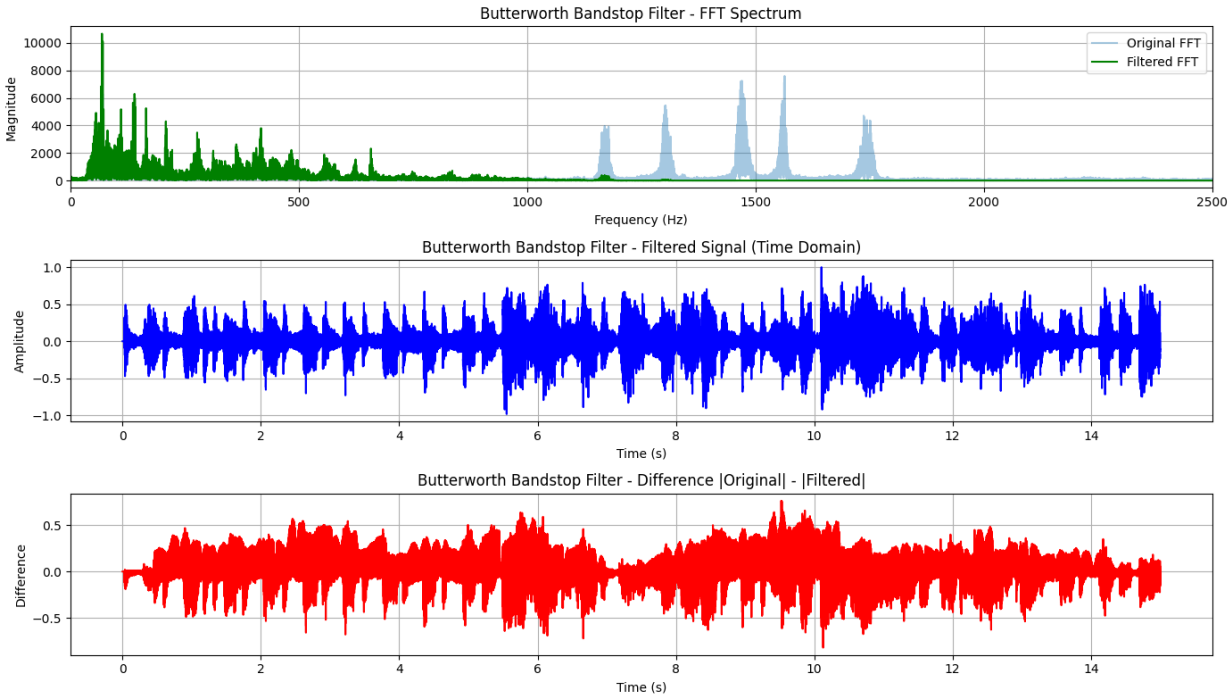

Figure 14: Final Waveform after applying filter

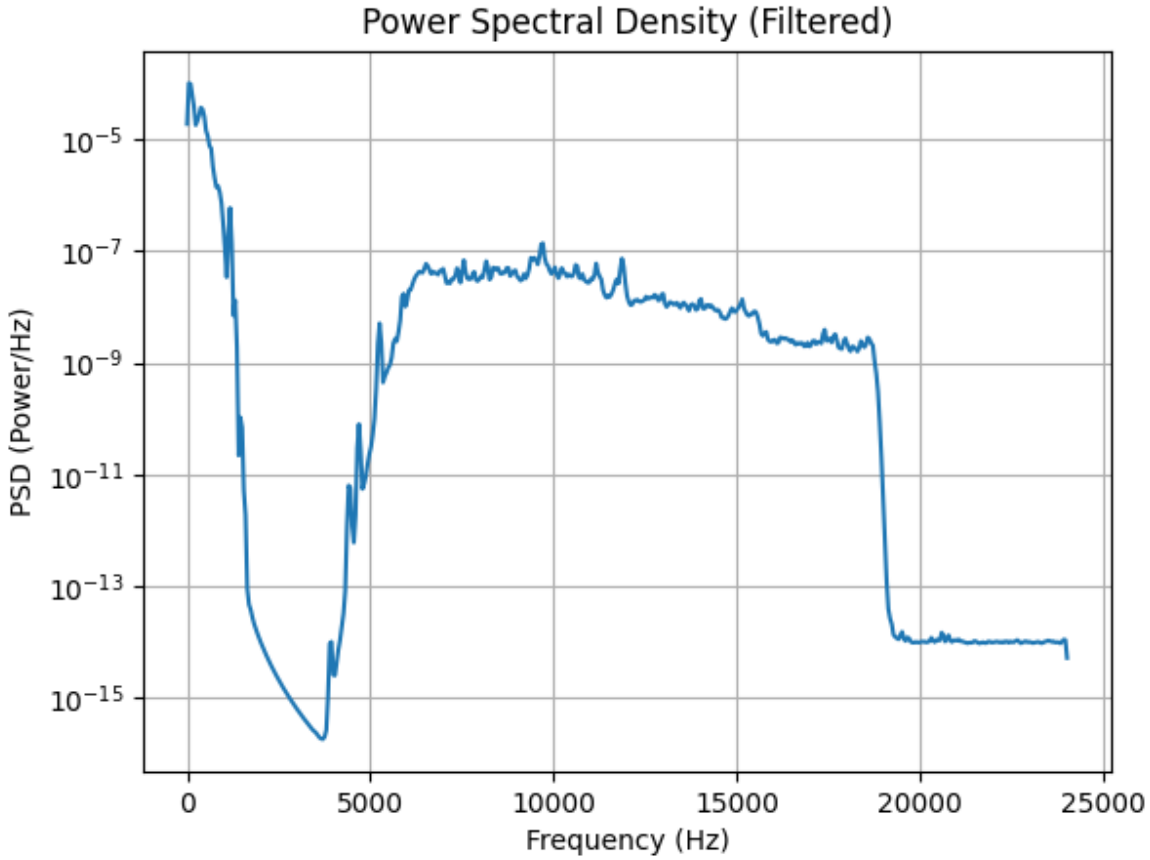

Figure 15: Power Spectral Density after applying filter

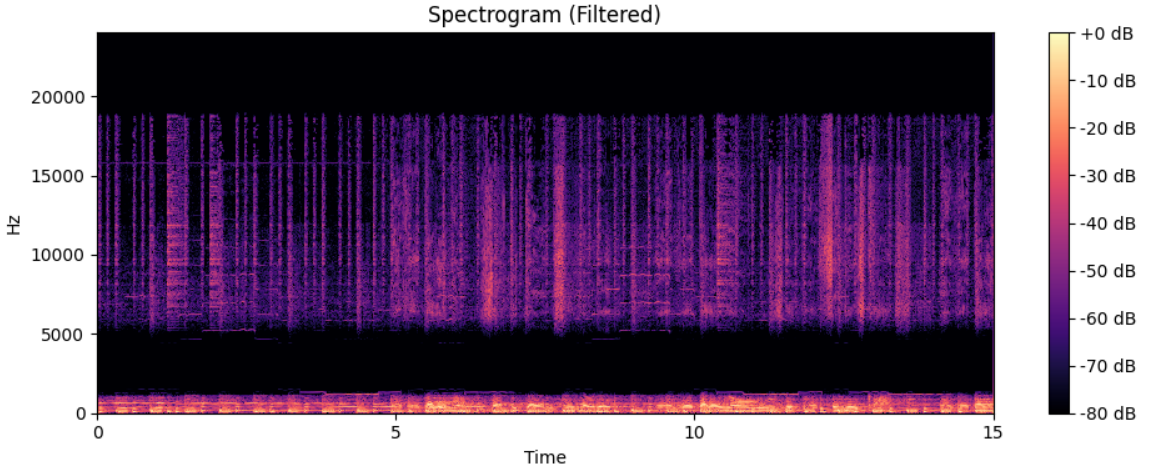

Figure 16: Spectogram after applying filter