prelab5

February 27, 2023

[9]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io.wavfile import read, write
from scipy import signal
from numpy.fft import fft, ifft, rfft
from IPython.display import Audio
```

[10]:
```
Fs_1, data_1 = read('test_audio.wav')
data_1 = data_1[:,0]
Audio(data_1, rate=Fs_1)
```

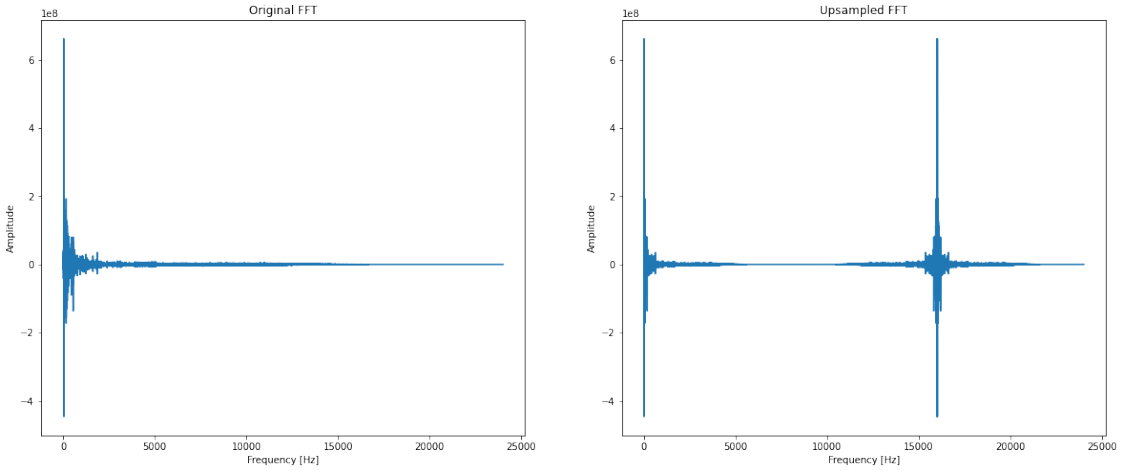
[10]:
```
<IPython.lib.display.Audio object>
```

[11]:
```
upsampled_3 = np.zeros(len(data_1)*3)

for i in range(len(upsampled_3)):
    if ((i%3) == 0):
        upsampled_3[i] = data_1[i//3]

Audio(upsampled_3, rate=Fs_1)
```

[11]:
```
<IPython.lib.display.Audio object>
```

[12]:
```
fft_og = rfft(data_1)
freq_og = np.linspace(0, Fs_1/2, len(fft_og))
fft_upsampled = rfft(upsampled_3)
freq_upsampled = np.linspace(0, Fs_1/2, len(fft_upsampled))
plt.figure(figsize=(20,8))
plt.subplot(121)
plt.plot(freq_og, fft_og)
plt.title("Original FFT")
plt.ylabel("Amplitude")
plt.xlabel("Frequency [Hz]")
plt.subplot(122)
plt.plot(freq_upsampled, fft_upsampled)
plt.title("Upsampled FFT")
plt.ylabel("Amplitude")
plt.xlabel("Frequency [Hz]")
```

[12]: Text(0.5, 0, 'Frequency [Hz]')



What is the relationship between the original signal's FFT and the upsampled signal's FFT? Upsampling leads to a more compressed version of the original signal meaning that the pitch becomes much lower.

How do we preserve the original information without introducing aliasing? After we upsample the signal, we must use a low pass filter on the new signal with a range of $-\frac{\pi}{M} \leq LPF \leq \frac{\pi}{M}$ where M is the factor by which you are upsampling. That would be the range at least for the STFT. In the discrete fourier transform domain, the range would be $-\frac{Fs}{2M} \leq LPF \leq \frac{Fs}{2M}$ where Fs is the sampling rate.

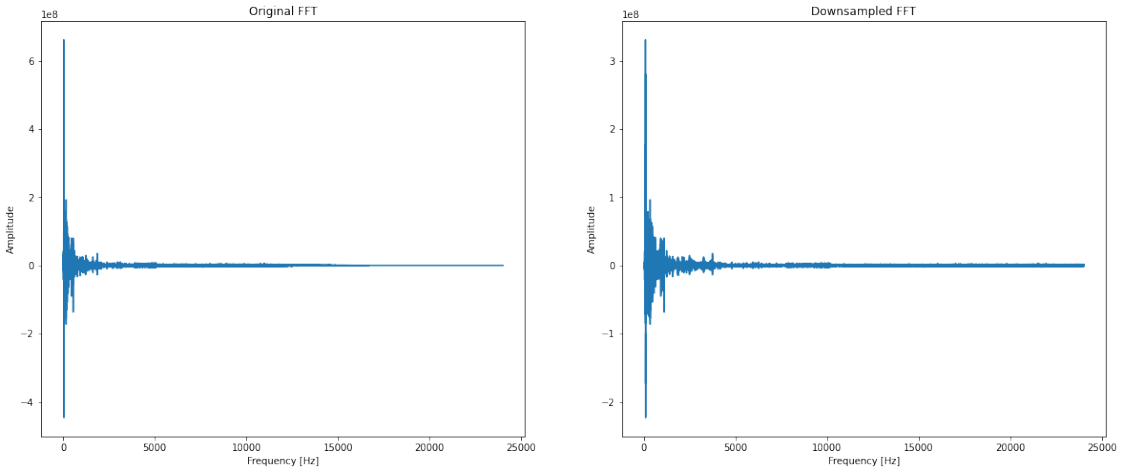
[13]:
```
downsampled_2 = data_1[::2]

Audio(downsampled_2, rate=Fs_1)
```

[13]: <IPython.lib.display.Audio object>

[14]:
```
fft_og = rfft(data_1)
freq_og = np.linspace(0, Fs_1/2, len(fft_og))
fft_downsampled = rfft(downsampled_2)
freq_downsampled = np.linspace(0, Fs_1/2, len(fft_downsampled))
plt.figure(figsize=(20,8))
plt.subplot(121)
plt.plot(freq_og, fft_og)
plt.title("Original FFT")
plt.ylabel("Amplitude")
plt.xlabel("Frequency [Hz]")
plt.subplot(122)
plt.plot(freq_downsampled, fft_downsampled)
plt.title("Downsampled FFT")
```

```
plt.ylabel("Amplitude")
plt.xlabel("Frequency [Hz]")
```

[14]:
```
Text(0.5, 0, 'Frequency [Hz]')
```



What is the relationship between the original signal's FFT and the downsampled signal's FFT? Downsampling stretches the original signal, leading to higher pitches in the signal.

How do we preserve the original information without introducing aliasing? Before downsampling is done, you must use a low pass filter on the signal with a range of $-\frac{\pi}{L} \leq LPF \leq \frac{\pi}{L}$ where L is the factor by which you are downsampling. That would be the range at least for the STFT. In the discrete fourier transform domain, the range would be $-\frac{Fs}{2L} \leq LPF \leq \frac{Fs}{2L}$ where Fs is the sampling rate.

[15]:
```
Fs, data = read('test_audio.wav')
data = data[:, 0]

up_ratio_1 = 4
down_ratio_1 = 2
up_ratio_2 = 2
down_ratio_2 = 2
up_ratio_3 = 2
down_ratio_3 = 4

output_1 = signal.resample_poly(data, up_ratio_1, down_ratio_1)
output_2 = signal.resample_poly(data, up_ratio_2, down_ratio_2)
output_3 = signal.resample_poly(data, up_ratio_3, down_ratio_3)
```

[16]:
```
Audio(output_1, rate=Fs)
```

[16]:
```
<IPython.lib.display.Audio object>
```

[17]:
```
Audio(output_2, rate=Fs)
```

[17]:
```
<IPython.lib.display.Audio object>
```

[18]:
```
Audio(output_3, rate=Fs)
```

[18]:
```
<IPython.lib.display.Audio object>
```

What does the resulting audio sound like? Be specific in your response. How do the vocal characteristics of the singer change? When the ratio is 1, the audio sounds almost unchanged. When the ratio is 2 though, the pitch seems a lot deeper than before. And when the ratio is 1/2, the pitch becomes much higher than the original.