Assignment 4

Batuhan Faik Derinbay derinbay18@itu.edu.tr
150180705

Question 1

- In the first part of the code "SilentKnight.wav" audio file is read. Then, according to its maximum FFT value, a y axis limit for the frames is selected. This is solely for visual reasons and is not required. Afterwards, the framerate per second (FPS) is selected and the window length is calculated accordingly.

Read Audio and Set Variables

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal
from scipy.io import wavfile
from scipy.signal import savgol_filter
from IPython.display import Audio
import scipy.fftpack as fftpack
import os
import shutil

sampling_freq, silent_knight = wavfile.read("HW4_audio/SilentKnight.wav")
silent_knight_L = silent_knight[:,0]; silent_knight_R = silent_knight[:,1]

silent_knight_L_norm = silent_knight_L/max(abs(silent_knight_L))
silent_knight_fft = fftpack.fft(silent_knight_L_norm)
silent_knight_fft_magn = np.abs(silent_knight_fft)
silent_knight_fft_magn = 10 * np.log(silent_knight_fft_magn + np.finfo(float).eps)

FPS = 30
y_limit = 3*max(silent_knight_fft_magn)/5
window_length = sampling_freq // FPS
```

- In the second part of the code, a temporary directory is created for output frames. Then, for each interval of the input signal, FFT is applied. After FFT, a smoothing Savitzky–Golay filter is applied for better looking outputs. Finally, each frame is saved using built-in functions of the ‘matplotlib‘ library. As an example, one of the frames can be seen in Figure 1. This sample frame can be reproduced by setting the index variable ‘idx = 10‘.

Draw and Save Each Frame

```
if os.path.exists('temp'):
    shutil.rmtree('temp')
os.makedirs('temp')

# Frames of the YouTube video are rendered using 300 DPI
DPI = 50    # Change this for faster processing times
data_length = silent_knight.shape[0]//window_length

for idx in range(data_length):
    window_freqs = fftpack.fftfreq(len(silent_knight_L[idx*↩
        window_length:(idx+1)*window_length]))*window_length

    silent_knight_fft_window = fftpack.fft(↩
        silent_knight_L_norm[idx*window_length:(idx+1)*↩
        window_length])
    silent_knight_fft_magn_window = np.abs(↩
        silent_knight_fft_window)
    silent_knight_fft_magn_window = 10 * np.log(↩
        silent_knight_fft_magn_window + np.finfo(float).eps)
    silent_knight_fft_magn_window = savgol_filter(↩
        silent_knight_fft_magn_window, 101, 4)

    plt.figure(figsize=(10, 5))
    plt.axis('off')
    plt.ylim(0, y_limit)
    plt.axhspan(0, y_limit, facecolor='#0f0e2b', alpha=1)
    plt.plot(window_freqs, silent_knight_fft_magn_window, ↩
        color='#d18d44', linewidth='4')
    plt.savefig('temp/{0:05}.png'.format(idx), format='png', ↩
        bbox_inches='tight', dpi=DPI)
    plt.close()
```

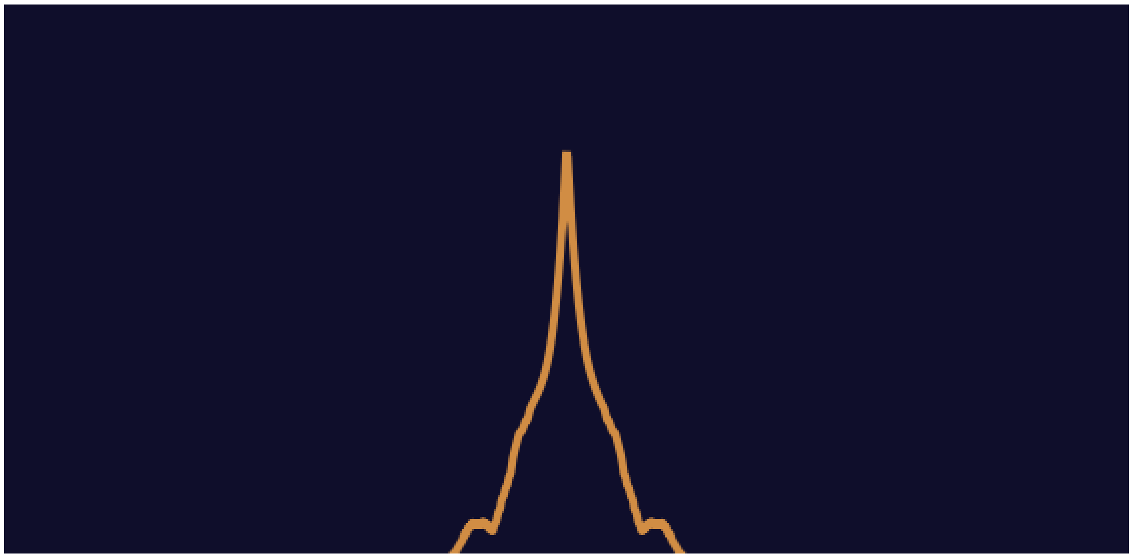

Figure 1: Sample Frame

- In the third and the final part of the code, directories of temporary frames are read and appended in a list. Then using built-in functions of the ‘moviepy‘ library, a video clip using the image sequences and an audio clip using the previously imported ”SilentKnight.wav” is exported as ”SilentKnightVisualized.mp4”. Output video of the question 1 can be found in the same directory with the filename ”SilentKnightVisualized.mp4” after running the provided code snippets. Moreover, the output video with the filename ”SilentKnightVisualized_300dpi.mp4” can also be found in the submission files. 300 DPI exported images have better quality overall but takes a long time to export. Therefore it is provided with the submitted homework files and can be viewed on YouTube through the following address: youtu.be/nQsbC5ZfIDg.

Export Video

```
# Get the directories of images
images = [os.path.join('temp',f) for f in os.listdir('temp') ↩
    if os.path.isfile(os.path.join('temp', f))]
images = sorted(images)

import moviepy.editor as mpy

clip = mpy.ImageSequenceClip(images, fps=FPS).set_audio(mpy.↩
    AudioFileClip("HW4_audio/SilentKnight.wav"))
clip.write_videofile('SilentKnightVisualized.mp4')
shutil.rmtree('temp')
```

Question 2

- In the first part of the code ”aphex_twin_equation.wav” audio file is read and normalized.

Read Audio File and Normalize

```
sampling_freq, aphex_twin = wavfile.read("HW4_audio/aphex_twin_equation.wav")
aphex_twin_L = aphex_twin[:,0]; aphex_twin_R = aphex_twin[:,1]
aphex_twin_L_norm = aphex_twin_L/max(abs(aphex_twin_L))
```

- In the second part of the code window size and stride length for the short-time fourier transform (STFT) are set and an overlap ratio of 50% is chosen. Then, FFTs of the input signal are taken with the lenght of windows size. After each FFT, the window is shifted the amount of stride length to the next time frame. In the end, logarithm operation is used for better representation. It should also be mentioned that using the machine epsilon in the logarithm operation causes the output spectrogram to be highly distorted. Therefore a variable epsilon, set to 0.0001 is used instead.

STFT of the Input Signal

```
epsilon = 0.0001
audio_duration = aphex_twin_L.shape[0]//sampling_freq
window_size = 4096
stride = 2048
window_psec = sampling_freq//stride
aphex_twin_output = np.empty([audio_duration*window_psec, window_size])

for idx in range(aphex_twin_output.shape[0]):
    aphex_twin_output[idx] = np.abs(fftpack.fft(aphex_twin_L_norm[idx*stride:idx*stride+window_size]))
aphex_twin_output = 10 * np.log(aphex_twin_output + epsilon)
```

- In the third and the final part of the code, output of the STFT operation is plotted using the ‘pcolormesh‘ function. The resulting plot of the Aphex Twin signal, and the hidden face is given in Figure 2.

Plot the Hidden Face

```
fig = plt.figure(figsize=(12,9))
plt.pcolormesh(np.transpose(aphex_twin_output)[:↩
    aphex_twin_output.shape[1]//2])
plt.yscale("symlog")
plt.axis('off')
plt.title("STFT of Aphex Twin and The Hidden Face")
plt.show()
```

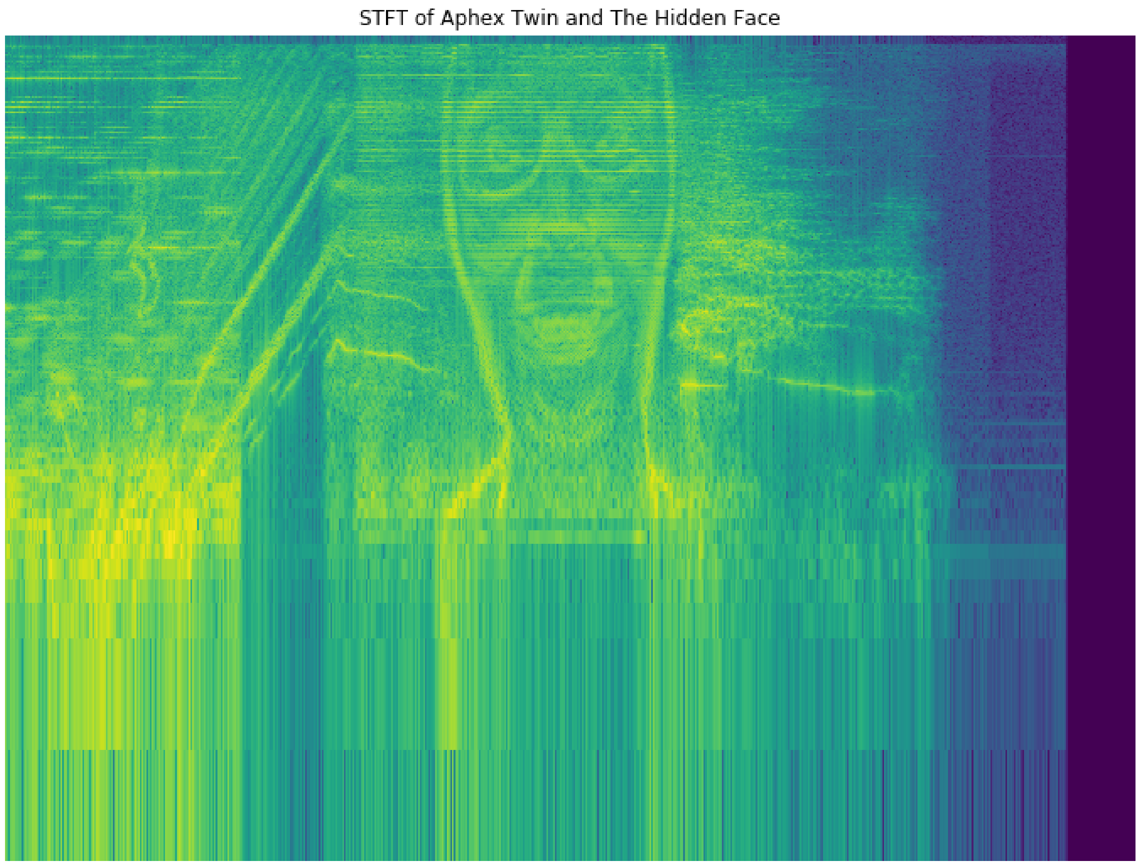

Figure 2: STFT of Aphex Twin and the Hidden Face