Problem Set #4

Atahan ÖZER

BLG 354 SIGNAL AND SYSTEMS Spring 2019-2020

Istanbul Technical University

June 15th

Audio Visualization

In this part of the code required packages are imported, audio is read from file and split into channels.

```
from scipy.io.wavfile import read
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from scipy import fft
import math
import  cv2
from moviepy.editor import *
import moviepy.editor as mpe


fs, song = read('SilentKnight.wav') # data reading
channel1 = song[:,1]
channel1 = channel1/max(abs(channel1))
```

After reading the data fps is chosen and according to fps, new sampling rate and total number of intervals calculated. All intervals must have same number of sample therefore a helper pad function is defined and data is padded.

```python
def padder(data,size): # helper function to pad the windows
    new = np.zeros(int(size))
    new[0:len(data)]= data
    return new

fps= 10
interval_sample = int(fs/fps)
interval_number = math.ceil(len(channel1)/interval_sample)
new_rate = interval_number*interval_sample
```

In this part magnitudes are calculated using fft transform. In order to have a better representation magnitudes are log scaled such as $10 * log10(x_{ft} + c)$. Here c is taken as 1 because visualization look better with it however normally it should be about 0.0001. One frame is plotted in order to check the procedure.

```python
magnitudes = np.zeros((interval_sample,interval_number))
for i in range(magnitudes.shape[1]):
    magnitudes[:,i]=abs(np.fft.fft(padded[interval_sample*i:interval_sample*(i+1)]))

magnitudes= 10*np.log10( magnitudes + 0.001)
freq = np.fft.fftfreq(interval_sample,d= 1/fs)
plt.plot(magnitudes[:interval_sample//2,200])
```

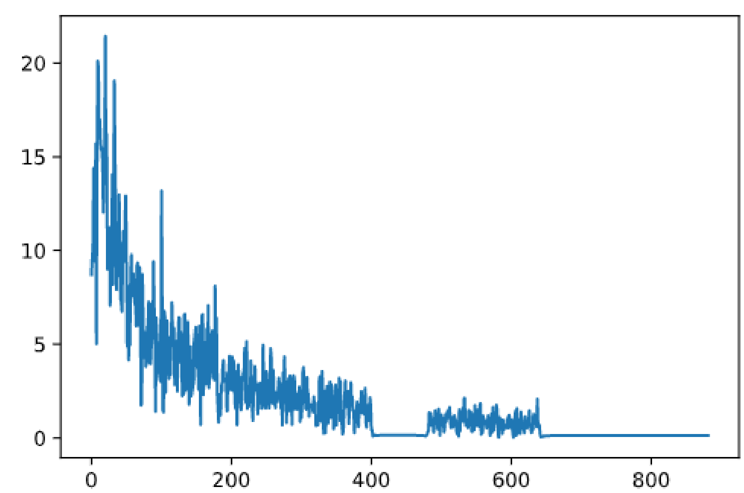

Figure 1: Sample frame

Since plots are ready, they need to be rendered and saved in a list so that a video can be made using those plots.

```

fig, ax = plt.subplots(nrows=1, ncols = 1, figsize =(10,6))
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_ylim(top=80)
imagelist = []

for i in range(interval_number):
    ax.plot(magnitudes[:interval_sample//2:,i])
    plt.savefig("snap.png",format="png")
    ax.clear()
    X = cv2.imread("snap.png")
    X =cv2.cvtColor(X,cv2.COLOR_BGR2RGB)
    imagelist.append(X)
```

In the final part frames are written into a video and music is added by using moviepy library.

```

clip = ImageSequenceClip(imagelist,fps = fps)
clip.write_videofile("part1video.mp4", codec = "mpeg4")


my_clip = mpe.VideoFileClip("part1video.mp4")
audio =  mpe.AudioFileClip("SilentKnight.wav")
final = my_clip.set_audio(audio)
final.write_videofile("total10_1.mp4")
```

The Spectogram

In this part of the code required packages are imported, audio is read from file and split into channels.

```
from scipy.io.wavfile import read
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from scipy import fft
import math

fs, song = read('aphex_twin_equation.wav')
channel1 = song[:,1]
channel1 = channel1/max(abs(channel1))
```

In this part window size and step size are choosen. Spectogram matrix is calculated depending on those values however padding is required in order to have same size of windows. Same padding function is used from the previous part.

```
window_size = 2048
step = 256
segment_no  = int(35*fs/step)
spectogram = np.empty([segment_no,window_size])

for i in range(spectogram.shape[0]):
    try:
      spectogram[i] = np.abs(fft(channel1[i*step:i*step+window_size]))
    except: # if sizes do not match
      padded = padder(channel1[i*step:i*step+window_size],window_size)
      spectogram[i] = np.abs(fft(padded))
```

Finally spectogram is log scaled and plotted using pcolormesh function. Since fft yields a symetric matrix only first half is used for plotting.It is observed that resolution is directly proportional to window size and reversely proportional to step size.

```
spectogram  = 10*np.log10(spectogram+0.01)
first_half = spectogram[:,:window_size//2]
fig, ax = plt.subplots(figsize=(10,6))

ax.set_yscale('symlog')
ax.pcolormesh(np.transpose(first_half))
plt.show()
```

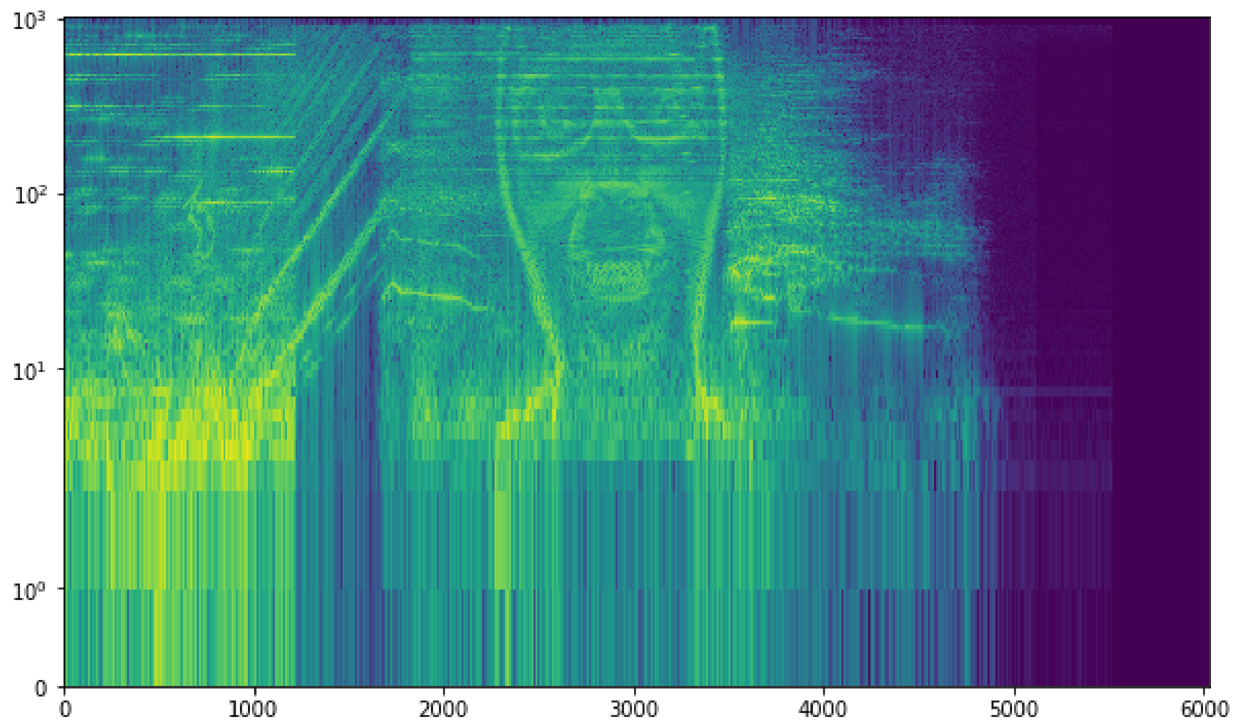

Figure 2: Spectogram