Problem 1 overview: DFT and spectrograms

Pertinent readings for Problem #1:

The Pokemon theme song:

My recording was taken from an actual recording session by the original guy who sang the title song ! Go to:

https://www.youtube.com/watch?v=fCkeLBGSINs

A brief introduction to vocals and spectrograms:

This is a great webpage !! I would definitely take a look at this first before you tackle the readings in Cohen & Rangayyan below:

http://vocped.ianhowell.net/a-spectrogram-primer-for-singers/

Cohen – Analyzing neural time series data

(Download from Blackboard, under /Resources / Signals & systems texts)

Ch. 11: pages 111 – 127 (A reminder on DFTs)
Ch. 15: pages 195 - 201 (A gentle introduction to spectrograms
Fancy-pants name = short-time Fourier transforms)

Rangayyan – Biomedial signal analysis (2nd ed) :

(Download from Blackboard, under /Resources / Signals & systems texts)

Ch. 3: pages 113 – 137 (A reminder on DFTs)
Ch. 8: pages 478 - 486 (short-time Fourier transforms)

1) Spectrograms = Local DFT calculations, with a sliding window

If you are familiar with the concept of convolutions (ie. Calculations involving moving windows), then you will immediately find the concept of making spectrograms really easy ! To begin with, a spectrogram is a *pcolor* representation of the frequency contents within a dataset, where:

a) The horizontal axis = Time
b) The vertical axis = Frequency components of our data, in the vicinity of a given time point

In Figure 1, we have plotted a spectrogram of an excerpt from the Pokemon theme song ! You can download this sound file in the same directory as this homework:

$Sound\ file$: $Pokeman.wav$

$Total\ duration\ of\ sound$: $132\ seconds$

$Data\ point\ density$: $f_{sample} = 44{,}100\ Hz$ $\left(\begin{matrix}This\ means\ 1\ second\ worth\ of\ sound \\ is\ equal\ to\ 44{,}100\ data\ points\ !!\end{matrix}\right)$

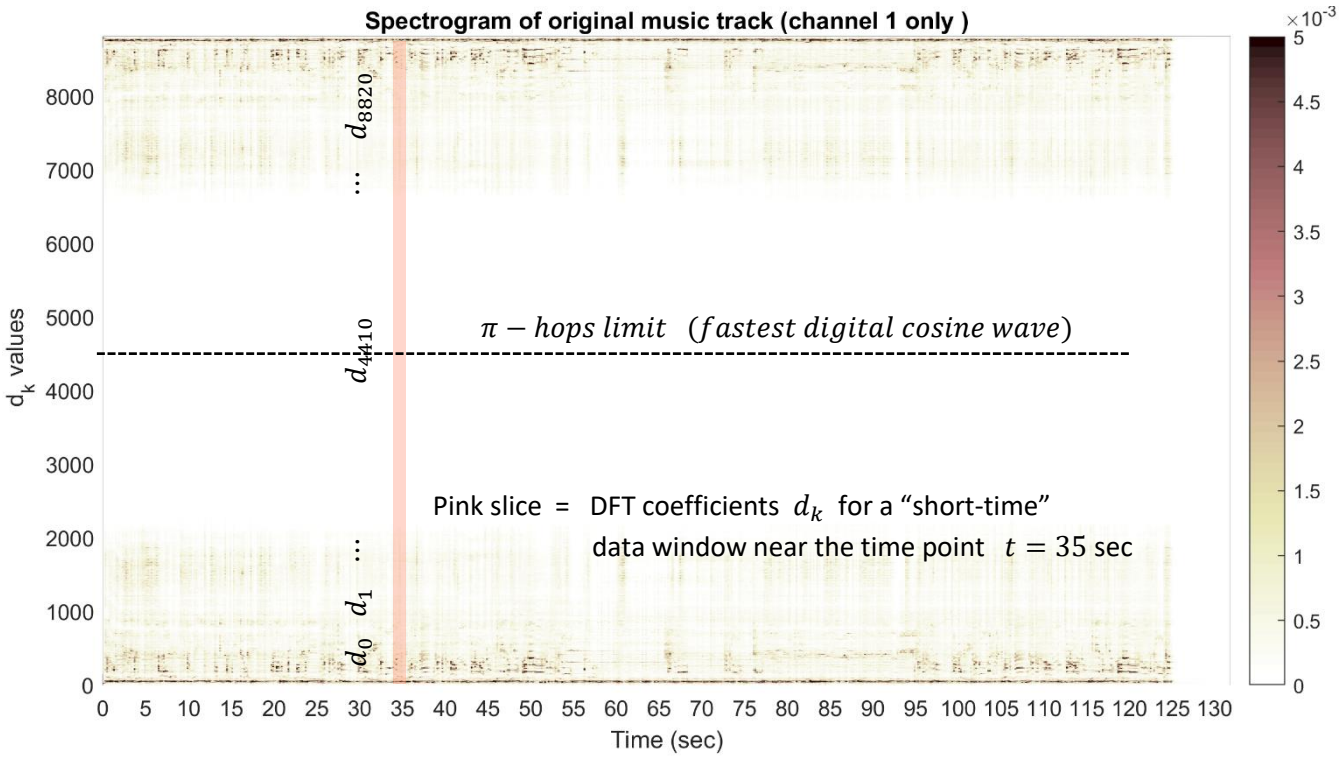


Figure 1: A spectrogram of *pokeman.wav*, where the frequency content (vertical axis) within lots of "short-time" window slices our data are plotted agains the time point of interest.

In this exercise, you will try to code your own "non-time-overlapping" spectrogram ! The non-overlapping spectrogram will be much easier to code and analyze than the standard "time-overlapped" spectrogram that most signal processing software use. Let's check out the basic steps !

Step 1: Pick a window width and use it to extract a section of data

Usually, your window with should be a fraction of your sampling frequency f_{sample}. . As an example:

If you decide to take chunks of …		*Your window length*
1 *second worth of data*	$\longrightarrow L =$	44,100 *data points* $= f_{sample}$ $(when\ f_{sample} = 44.1\ kHz)$
0.5 *second worth of data*	$\longrightarrow L =$	22,050 $= f_{sample}/2$
0.2 *second worth of data*	$\longrightarrow L =$	8820 $= f_{sample}/5$
⋮	⋮	

For the Pokeman spectrogram in Figure 1, we'll go with a smaller data window of:

$$L = 8820 = f_{sample}/5 \quad \left(\begin{array}{c} We\ will\ analyze\ samll\ chunks\ of\ data \\ at\ every\ 0.2\ seconds \end{array}\right)$$

Now, starting from the 1st sound data point y_0, we will use this window length and extract 0.2 sec worth of data points. This is the pink region depicted in Figure 2.

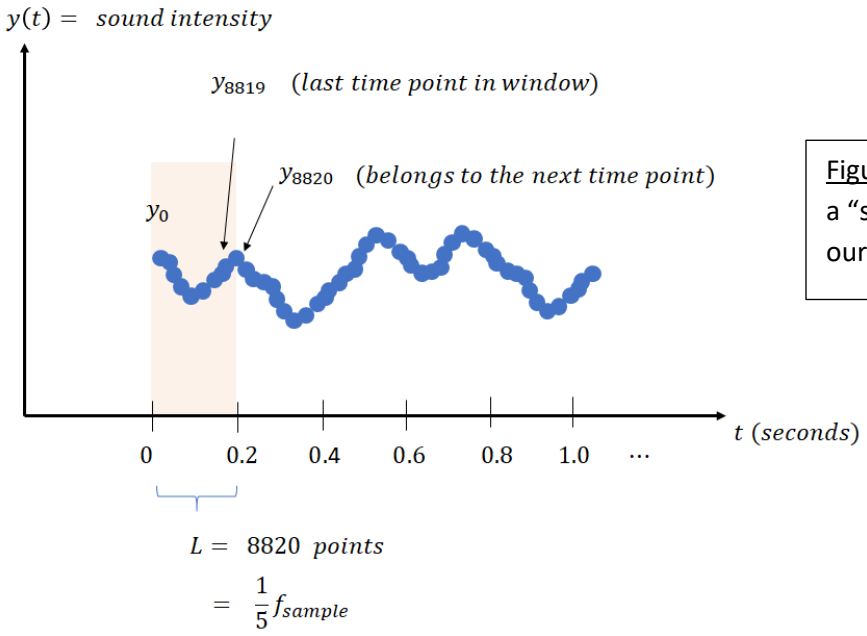


Figure 2: Using our window to extract a "short-time" windowed data from our sound file

Step 2: Compute and store the DFT coefficients d_k for your windowed data

Now, all we have to do is to take the DFT of our windowed data. By taking the DFT of the pink region, we are basically asking whether there are interesting (or statistically-important) periodicity within that 0.2 second worth of sound data.

After taking our "*short-time discrete Fourier transform*," we will store the d_k coefficients in the "$t = 0$" time slot of a brand-new storage matrix. These processes are depicted in Figure 3.

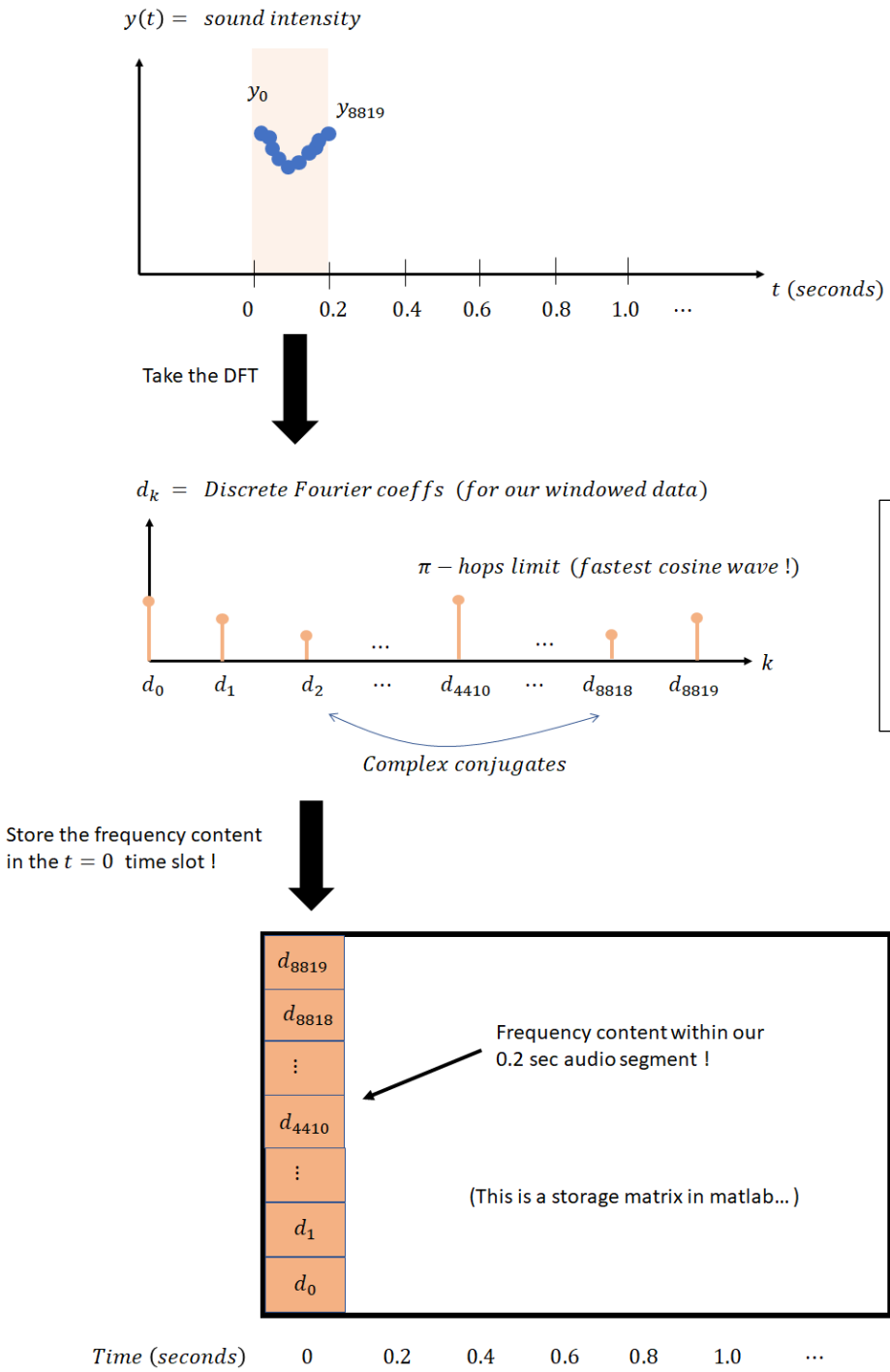


Figure 3: We take the *short-time discrete Fourier transform* of our windowed data, and then, we'll save the frequency information in a storage matrix

Storage matrix

Step 3: Slide your window to the right…. and do it again (*non-overlapping* fashion)

In a simple, non-overlapping version of the spectrogram, we will repeat the short-time DFT process by sliding your window towards the right with a full slide width of L. This will sure that our short-time DFT data sets are non-overlapping between window frames.

** Note: Most of time, you might want to have some overlaps because the frequency content between different windows might change drastically. You can read the discussions in Cohen, pages 199 – 200 (noted on the 1st page of this PDF) to gain more insights into this.

If you stare at Figure 4 for a while, you may realize that our window will "slide out of frame" when it gets to the right-edge of our dataset. This is where we need to create our favorite *ghost nodes* to compensate for this anomaly !

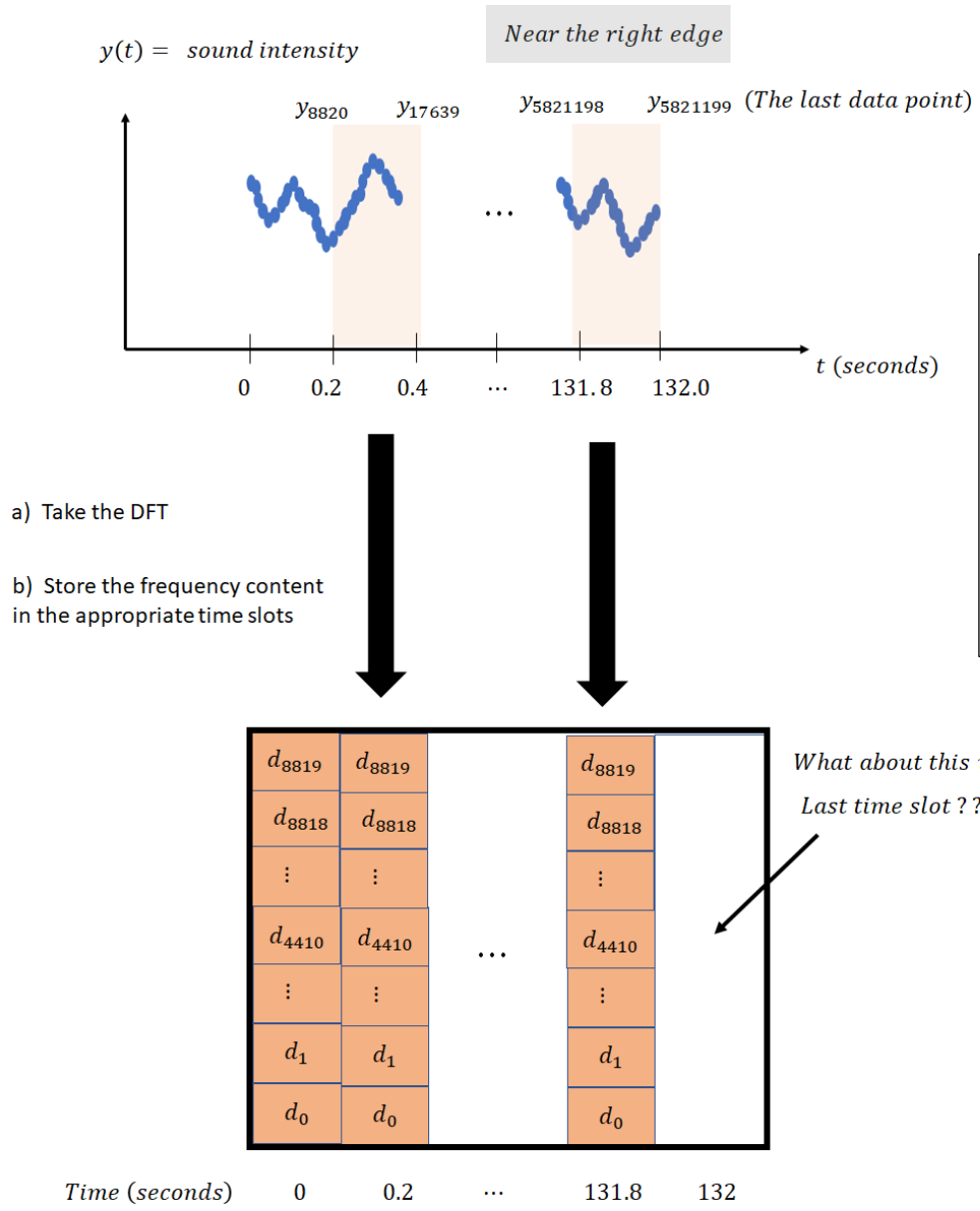


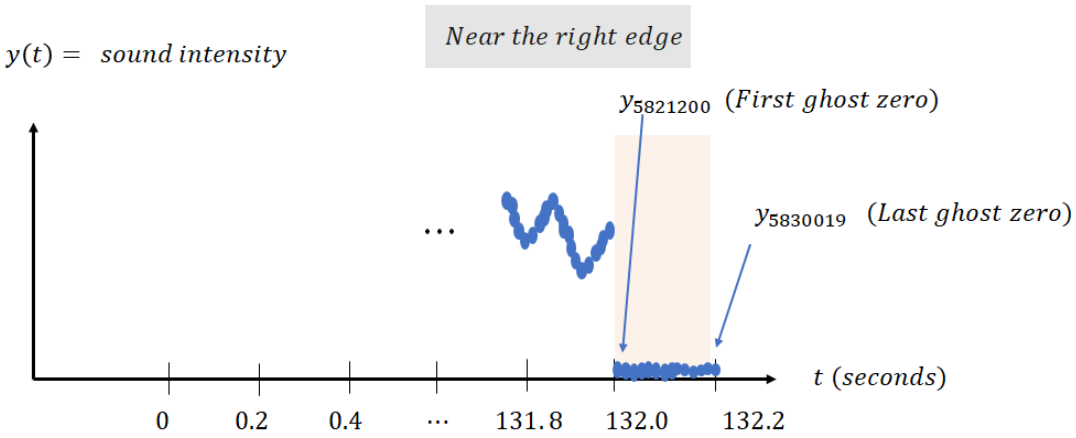
Figure 4: Non-overlapping spectrogram, where our sliding window will never contain overlapping DFT information between adjacent data frames.

We also need to do something to take care of the *right-edge time slot* in our storage matrix…

Step 4: Add "ghost zeros" at the right edge, and slide one more time….. !

At the right-most edge, if your sliding window has a with of L, it is customary to add L – worth of *ghost zero nodes* at the right edge to compensate for the "missing" last time slot in our DFT storage matrix. Once you do that, then you can safely slide your window towards the right one more time, get the last set of d_k's, and then save your coefficients in the last slot of your storage matrix.

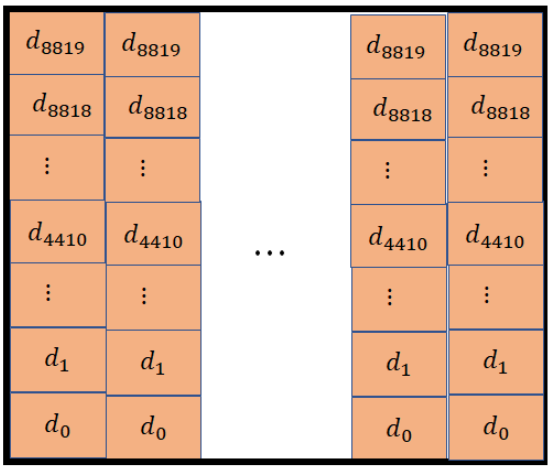
Then… congratulations – you've just created your first non-overlapping spectrogram ! =)



a) Take the DFT

b) Store the frequency content in the appropriate time slots

This entire slot will all be zeros …!



$\pi - hops\ limit$ $(fastest\ digital\ cosine)$

Slowest sine / cosine waves allowed within our 0.2 second window

DC component

Time (seconds) 0 0.2 ⋯ 131.8 132

Figure 5: We can compensate for the very last $t = 132$ sec slot by adding ghost zeros at the right-edge of our sound file.

The resulting storage matrix is our non-overlapping spectrogram !

2) Spectrograms = Great for analyzing sound files

You might ask: What are we gonna do with a spectrogram of our Pokemon theme song ? To begin with, the low-hanging fruit application of spectrograms is to identify different sound features in your *.wav file. For instance, we might be able to identify:

a) The main singer
b) Guitar riffs
c) Backup singers
d) Drums and other instruments, etc…

Once you've identify the individual sources of sounds, each of them may have their own frequency signatures…. And you guessed it – we might be able to perform digital filtering to remove or enhance these components !! I've plotted one zoomed-in section of the spectrogram for *Pokeman.wav* in Figure 6 below, where I have annotated some features of the sound file itself. The fully-annotated spectrogram can be found on Blackboard.

Play the sound file and see if you can follow the song on the spectrogram !

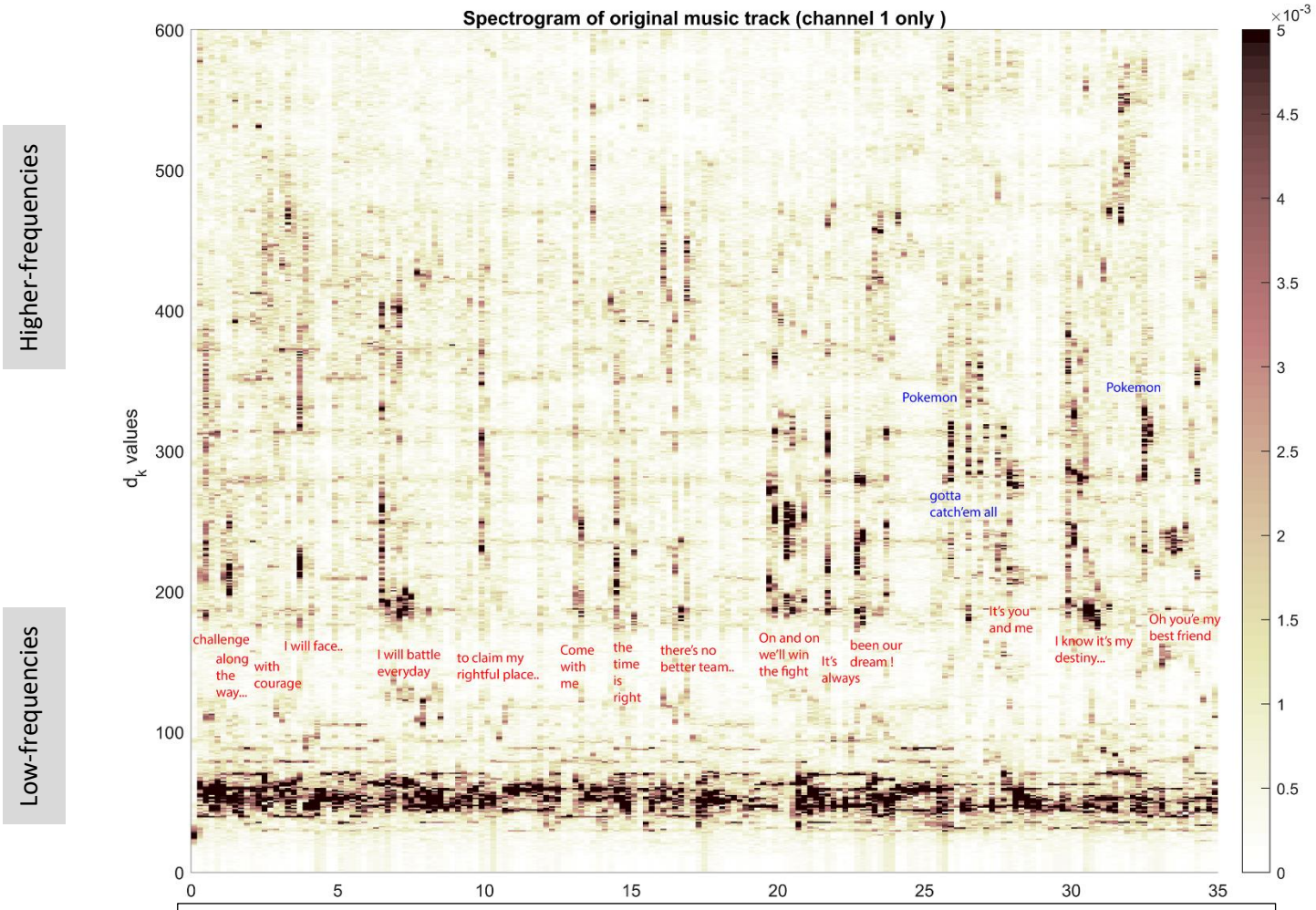


Figure 6: An excerpt of the spectrogram for *Pokeman.wav,* where one can analyze the frequency content of your sound file as a function of time. The vertical axis are the DFT coefficients for each small slice of windowed data.