feature_exploration

September 29, 2020

1 Audio Features

At this point, selecting and creating meaningful representations of our data is probably more important (or at least requires more thought) than our ultimate architecture. This might be truer of speech than other domains. Images have a natural conversion into mathematical objects which fit nicely into neural networks (tensors of pixel intensities), but getting from some digital signal to a meaningful feature in speech might not be immediately obvious (it wasn't to me).

So, here we'll do a deep dive into what a *feature* is in the domain of speech.

These techniques are discussed in:
[1] Beigi, H. *Fundamentals of Speaker Recognition* , Springer: New York, NY, 2011; doi:10.1007/978-0-387-77592-0
[2] Fayek, H. *Speech Processing for Machine Learning* source
First let's load a random audio file from the `dev-clean` dataset in the LibriSpeech corpus,which I have downloaded from here. Alternatively, both PyTorch and Tensorflow have built-in helper classes to load it.

2 Setup

[122]:
```
import os
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import Audio
```

[123]:
```
URL = 'dev-clean'
DATA_DIR = f'/home/thomas/Dir/ccny/ccny-masters-thesis/raw-data/LibriSpeech/
 ↪{URL}'
speaker = os.listdir(DATA_DIR)[0]
chapter = os.listdir(os.path.join(DATA_DIR, speaker))[0]
sample = os.listdir(os.path.join(DATA_DIR, speaker, chapter))[0]
speaker, chapter, sample
```

[123]:
```
('6345', '93302', '6345-93302-0013.flac')
```

[124]:
```
fpath = os.path.join(DATA_DIR, speaker, chapter, sample)
fpath
```

[124]:
```
'/home/thomas/Dir/ccny/ccny-masters-thesis/raw-data/LibriSpeech/dev-
clean/6345/93302/6345-93302-0013.flac'
```

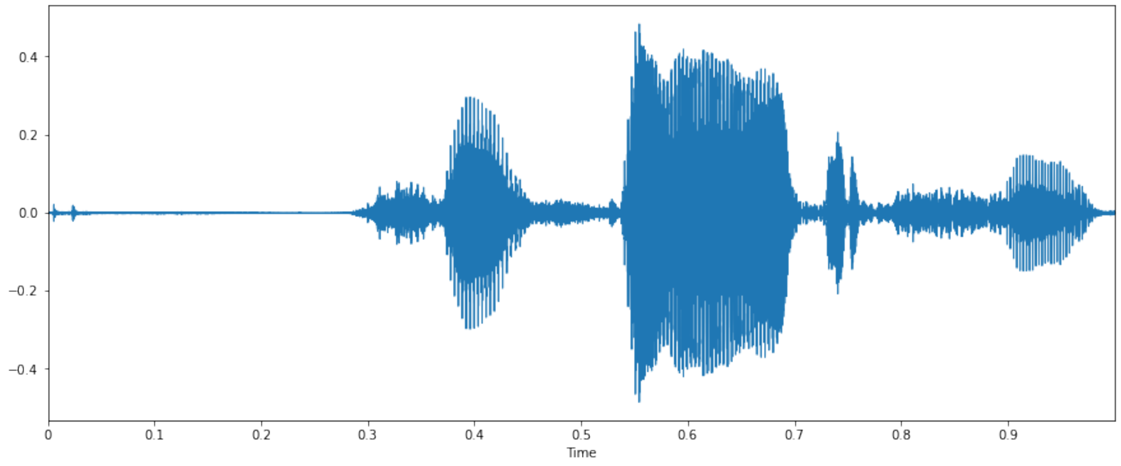
Now we can use the `librosa` python package to load our raw waveform.

[125]:
```
sr = 16000 # the Sample rate (Hertz) of our source audio
y, sr = librosa.load(fpath, sr=sr)
```

The time series returned by `librosa.load()` represents a quantized, digital, representation of the original analog signal of the individual reading one of the texts contained in the LibriSpeech dataset. It uses `audioread` to load a signed 16-bit integer which is then normalized to be bound `[-1,1]`. The resultant returned time series denotes the amplitude of the signal at each time step. Note that here we are experiencing information loss, because the original signal is analog (continuous) and we are representing it with discrete (though small) time steps.

Below, we'll visualize the first second as a raw waveform.

[126]:
```
plt.figure(figsize=(15,6))
_ = librosa.display.waveplot(y[:1*sr], sr=sr)
```



And this is what it sounds like…

[42]:
```
Audio(y[:1*sr], rate=sr)
```

[42]:
```
<IPython.lib.display.Audio object>
```

3 Data Cleaning

Before we begin to explore our possible feature mappings (frontends), representing these raw waveforms in way that retain information but are also meaningful with regards to the classification task at hand, we will need to perform some audio cleaning, to ensure we are getting the cleanest features possible.

3.1 Trim Boundary Silence

The first thing, simplest, thing to consider is silence. We do not want to ascribe a particular moment of silence (when a person is not speaking) to them. It does not make sense for a window of audio to be classifier as a speaker S if they did not actually speak in that window.

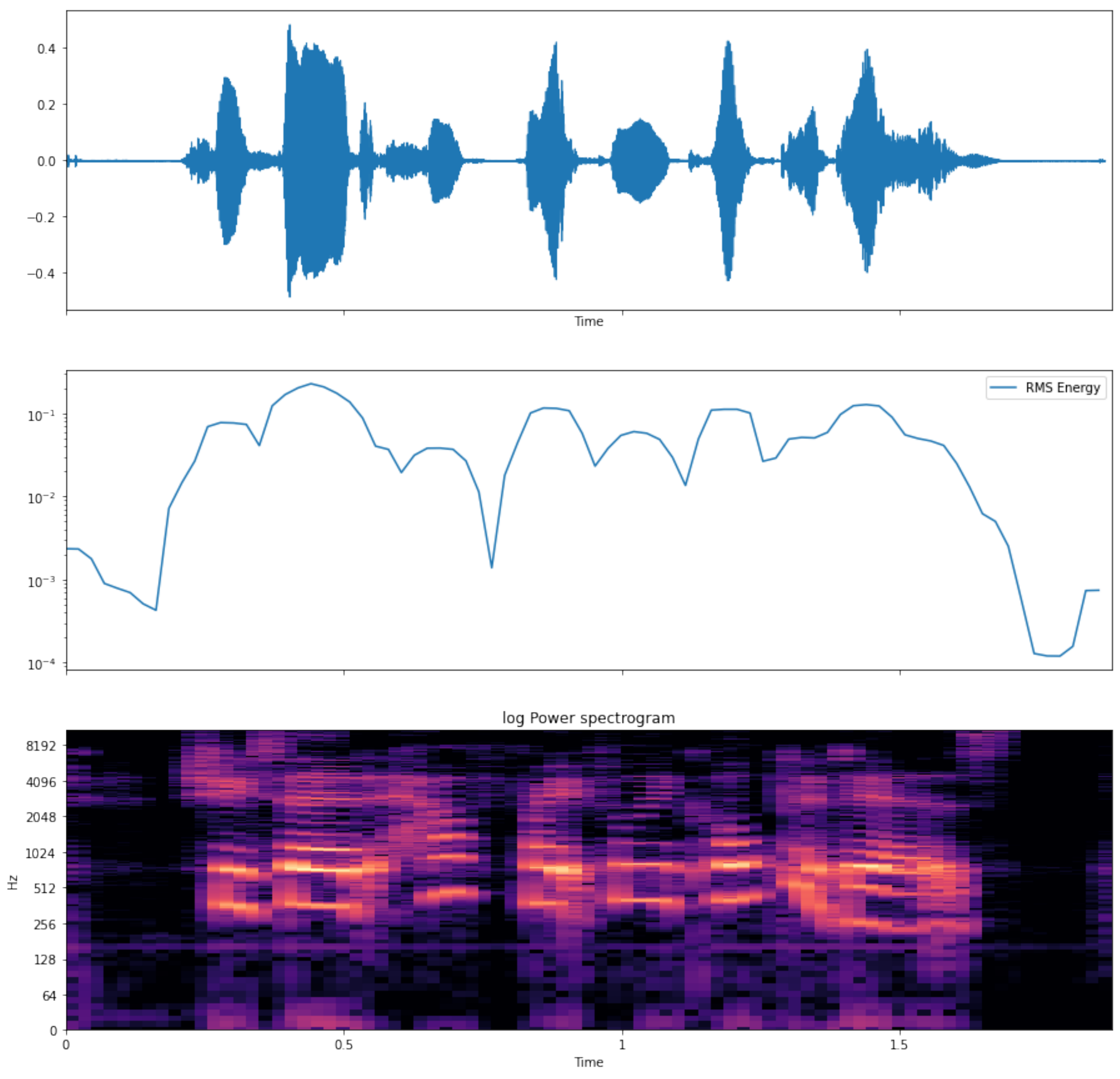
Here, using `librosa.effects.trim()` will perform threshold-based voice activity detection. Importantly, this method does not discriminate between just a loud noise (e.g. a trombone interrupting our speaker) and a person speaking. As long as the decibel threshold is reached, a frame is considered non-silent. Of course, in our controlled context this makes sense, but it is something to be aware of.

`librosa` filters from the threshold by calculating the MSE for the waveform and then calling `core.power_to_db()` on it to filter from the `top_db` parameter.

« Explain why you picked 30 and what a decibel is »

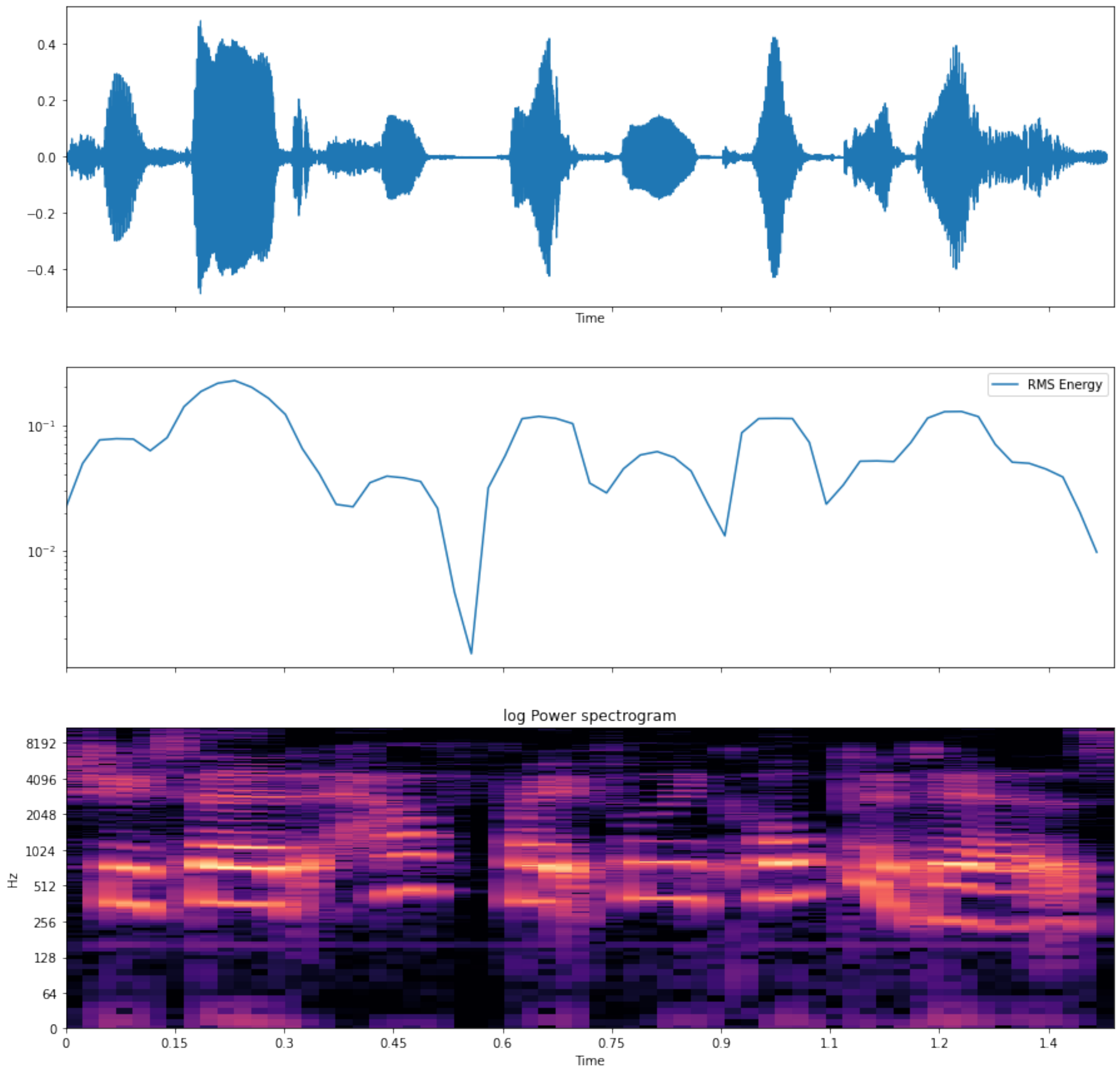
[183]:
```
FRAME_LENGTH = int(sr * .025) # 25 ms
HOP_LENGTH = int(sr * .01)    # 10 ms
```

[199]:
```
# add in the waveplot
fig, ax = plt.subplots(nrows=3, sharex=True, figsize=(15,15))

# raw waveplot
librosa.display.waveplot(y, ax=ax[0])

# Root mean square of raw signal
rms = librosa.feature.rms(y)
times = librosa.times_like(rms)
ax[1].semilogy(times, rms[0], label='RMS Energy')
ax[1].set(xticks=[])
ax[1].legend()
ax[1].label_outer()

# log power spectrogram
S, phase = librosa.magphase(librosa.stft(y))
librosa.display.specshow(librosa.amplitude_to_db(S, ref=np.max),
                         y_axis='log', x_axis='time', ax=ax[2])
ax[2].set(title='log Power spectrogram')

plt.show()
```



```
[94]: y_trim, (lower_idx, upper_idx) = librosa.effects.trim(y,␣
       ↪frame_length=FRAME_LENGTH, hop_length=HOP_LENGTH, top_db=30)
      S_orig = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max,␣
       ↪top_db=None)

      S_preemph = librosa.amplitude_to_db(np.abs(librosa.stft(y_filt)), ref=np.max,␣
       ↪top_
      # raw waveplot
      librosa.display.waveplot(y_trim, ax=ax[0])

      # Root mean square of raw signal
      rms = librosa.feature.rms(y_trim)
      times = librosa.times_like(rms)
      ax[1].semilogy(times, rms[0], label='RMS Energy')
```

```
ax[1].set(xticks=[])
ax[1].legend()
ax[1].label_outer()

# log power spectrogram
S, phase = librosa.magphase(librosa.stft(y_trim))
librosa.display.specshow(librosa.amplitude_to_db(S, ref=np.max),
                         y_axis='log', x_axis='time', ax=ax[2])
ax[2].set(title='log Power spectrogram')

plt.show()
```



[61]:
```
Audio(y_trim[:1*sr], rate=sr)
```

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

3.2 Pre-emphasis

In order to mimic the brain's processing of audio information by *emphasizing* a signal, we perform pre-emphasis. From [1], p. 154-155: > The cochlea utilizes a fine-tuning mechanism based on feedback from the brain that amplifies special frequencies. It is noted that the human ear can easily recognize these low energy reasons. Since we are designing an automatic speaker recognition system, we need to do something similar, to be able to utilize the important features embedded in higher frequencies such as fricatives, etc.

A frequently used method is a differentiator (a single zero filter) with the following transfer function:

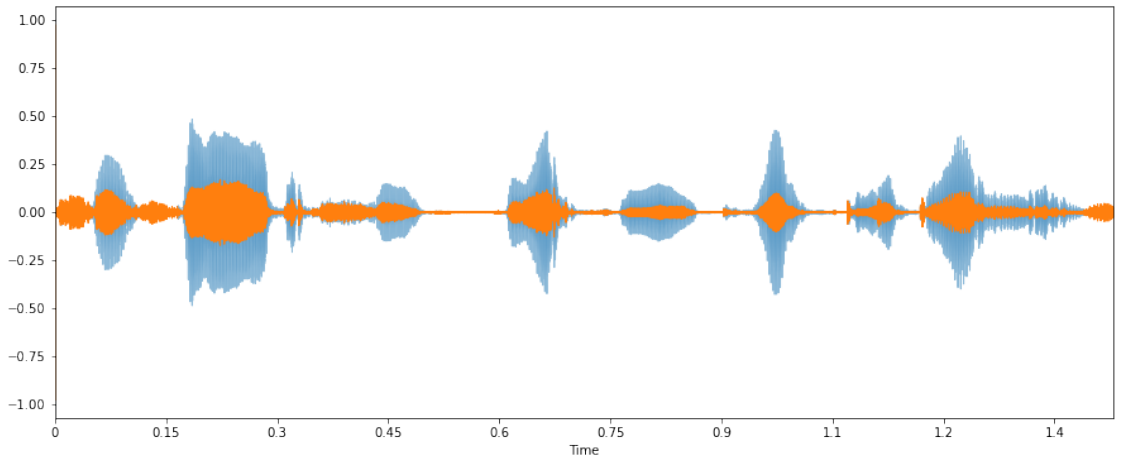
$$H_p(z) = 1 - \alpha z^{-1}$$

As `librosa.effects.preemphasis` implements it, given a signal value at t of a signal x, we receive our pre-emphasized signal y by computing:

$$y(t) = x(t) - \alpha x(t-1)$$

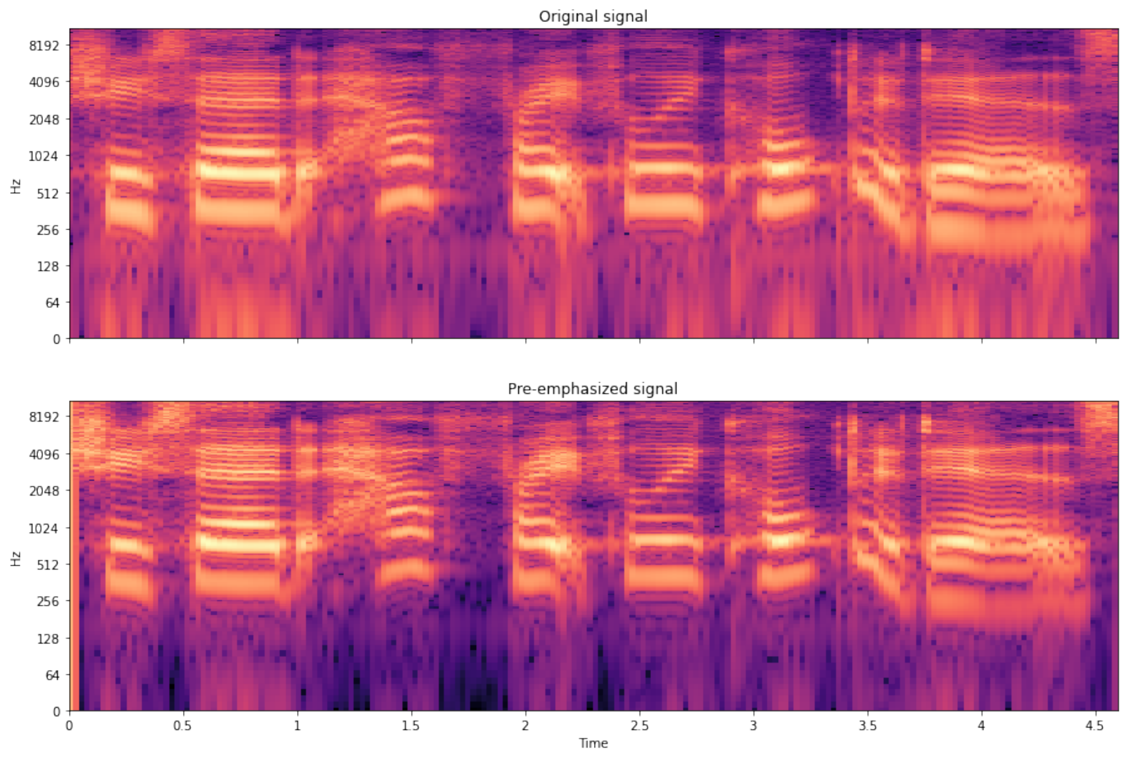
Typically a value of $[0.95, 0.97]$ has been used for α. The default in `librosa` is 0.97, given that it is the default used in the HTK implementation of an MFCC.

```
[100]: # fig, ax = plt.subplots(nrows=2, sharex=True, figsize=(15,15))
plt.figure(figsize=(15,6))
# raw waveplot
librosa.display.waveplot(y_trim, alpha=0.5)

# pre-emphasized
y_filt = librosa.effects.preemphasis(y_trim)
_ = librosa.display.waveplot(y_filt)
```



Now, we can visualize the resultant spectrograms from the original (silence-trimmed) signal and the pre-emphasized version. We should see the high frequency part of the spectrum become more prevalent.

[201]:
```
S_orig = librosa.amplitude_to_db(np.abs(librosa.stft(y_trim,␣
↪win_length=FRAME_LENGTH, hop_length=HOP_LENGTH)), ref=np.max, top_db=None)
S_preemph = librosa.amplitude_to_db(np.abs(librosa.stft(y_filt,␣
↪win_length=FRAME_LENGTH, hop_length=HOP_LENGTH)), ref=np.max, top_db=None)

fig, ax = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(15,10))
librosa.display.specshow(S_orig, y_axis='log', x_axis='time', ax=ax[0])
ax[0].set(title='Original signal')
ax[0].label_outer()
img = librosa.display.specshow(S_preemph, y_axis='log', x_axis='time', ax=ax[1])
_ = ax[1].set(title='Pre-emphasized signal')
```



[112]:
```
Audio(y_filt[:1*sr], rate=sr)
```

[112]:
```
<IPython.lib.display.Audio object>
```

We should note some have some have argued that pre-emphasis is an artifact of a previous era of speaker recognition prior to the advent of deeper neural networks, so we will continue our analysis with both `y_trim` and `y_filt`.

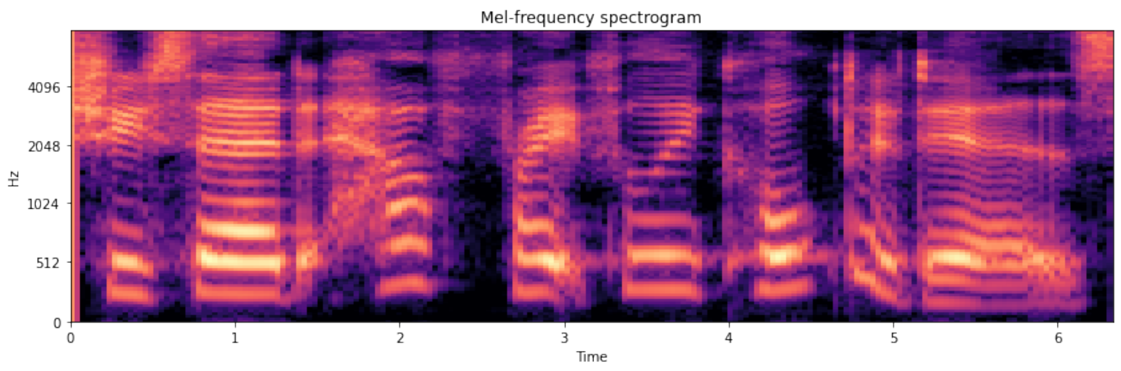
4 Final features

4.1 Spectrogram

The spectrogram is a visual representation of the spectrum of frequencies of a signal as they vary over time.

[202]:
```
S = librosa.feature.melspectrogram(y_filt, sr=sr, n_fft=2048,␣
↪hop_length=HOP_LENGTH, win_length=FRAME_LENGTH)
```

[204]:
```
fig, ax = plt.subplots(figsize=(14,4))
S_dB = librosa.power_to_db(S, ref=np.max)
img = librosa.display.specshow(S_dB, x_axis='time',y_axis='mel',␣
↪sr=sr,fmax=8000, ax=ax)
_ = ax.set(title='Mel-frequency spectrogram')
```



5 DIY

This section contains more of a DIY / roll-your-own approach following [2].

5.1 Utterances

The original waveform contains an entire sentence and several seconds of audio from our speaker.

[117]:
```
f'{y.size / sr} seconds of audio'
# show the utterance
```

[117]:
```
'2.575 seconds of audio'
```

We can, potentially, focus our energies more by restricting our investigation to the actual "utterances" (single word or syllables) of a sample. We can hear in our original that multiple words are spoken, with intermittent pauses.

Here, we might split that sample into its constituent parts, using `librosa.effects.split`.

« Meaning of `top_db` here too »

```
[164]: splits = librosa.effects.split(y_filt, top_db=20)
       splits
```

```
[164]: array([[    0, 11776],
              [12800, 19456],
              [20480, 23040],
              [23552, 31520]])
```

```
[169]: split = 3
       Audio(y[splits[split][0]:splits[split][1]], rate=sr)
```

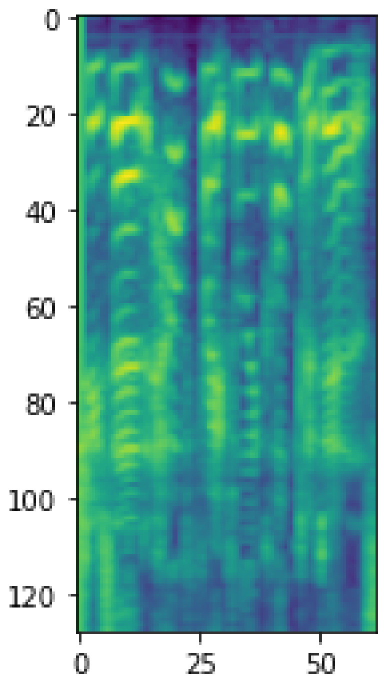
```
[169]: <IPython.lib.display.Audio object>
```

```
[ ]: # show waveforms we've received
```

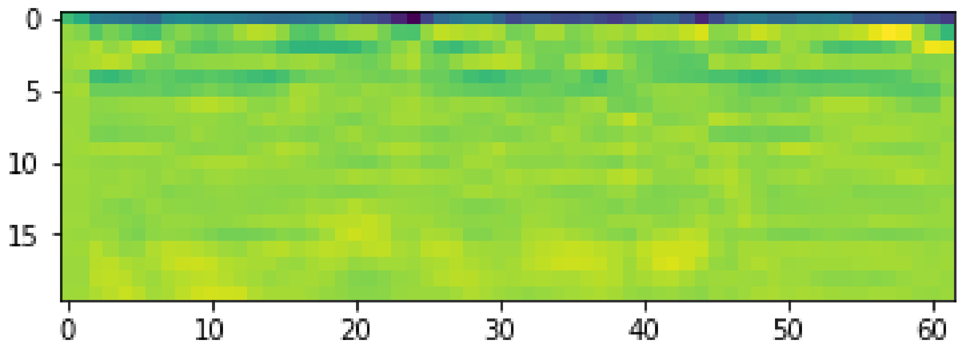
```
[160]: # visualize all the splits; and draw spectrograms, what not
```

```
[171]: plt.figure(figsize=(2,20))
       s_split = librosa.feature.melspectrogram(y_filt, sr=sr)
       print(s_split.shape)
       _ = plt.imshow( np.log( s_split + 1e-8) ) #librosa.power_to_db(s_split, ref=np.
        ↪max))
```

```
(128, 62)
```



[179]:
```
_ = plt.imshow(librosa.feature.mfcc(y_filt))
```



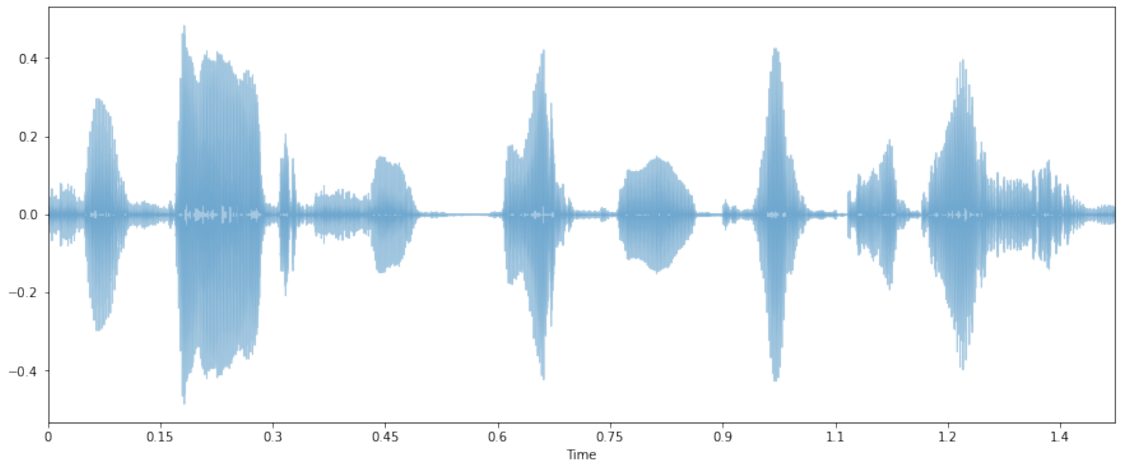
5.2 Mean Normalization

« Claims that using mean normalization achieves a similar result, investigate » http://isl.anthropomatik.kit.edu/pdf/Nguyen2018a.pdf

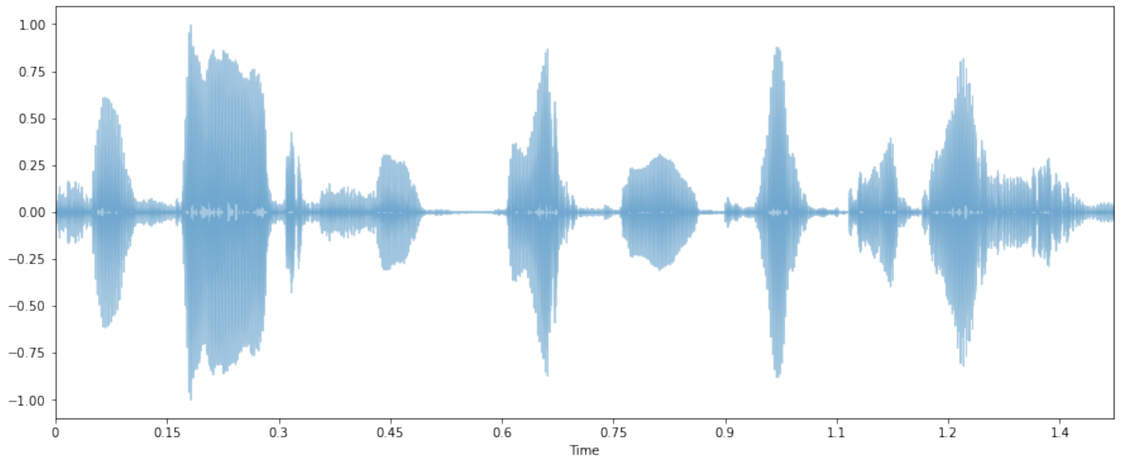
You have to think about how this works in a *multiple* speaker environment. Do you need the **global** mean and variance? How does this work to distinguish speakers, i.e. to make them separable?

[12]:
```
y_mean_norm = y - np.mean(y)
y_max_norm = y / np.max(y)
```

[16]:
```
plt.figure(figsize=(15,6))
_ = librosa.display.waveplot(y_mean_norm, alpha=0.4)
# _ = librosa.display.waveplot(y, alpha=0.1)
```



[18]:
```
plt.figure(figsize=(15,6))
_ = librosa.display.waveplot(y_max_norm, alpha=0.4)
```



[20]:
```
Audio(y / np.max(y), rate=sr)
```

[20]:
```
<IPython.lib.display.Audio object>
```

5.2.1 Framing

« How is the task of speech recognition different from speaker verification in this regard? »

```
[21]: y.shape
```

```
[21]: (31360,)
```

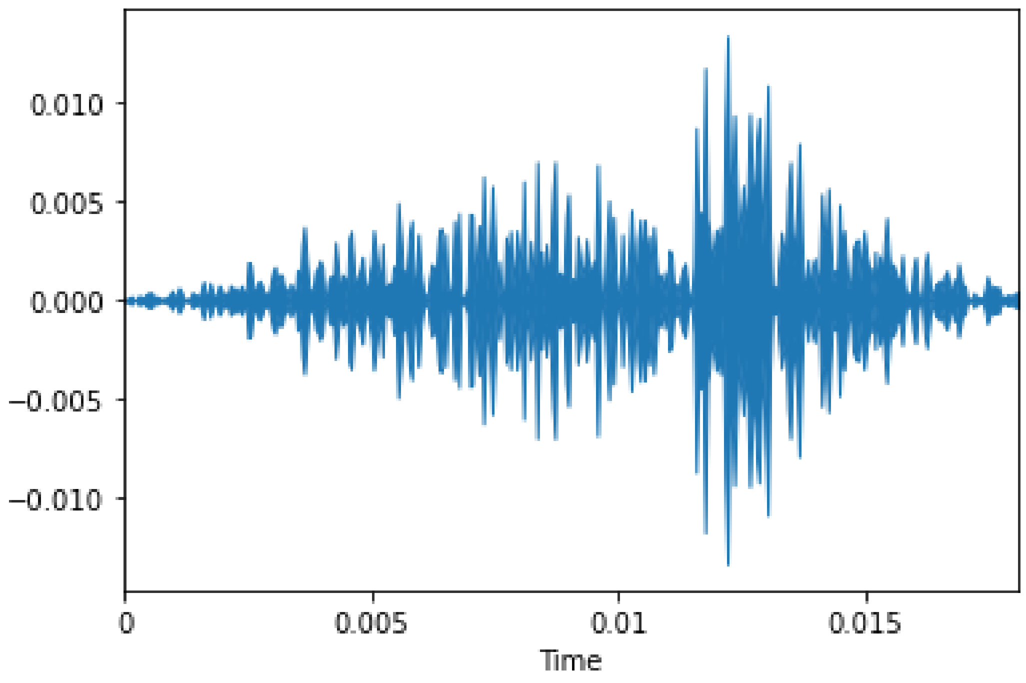
```
[22]: frame_length = int(sr * .025) # 25ms
      hop_length = int(sr * .01) # 10ms
      # returns numpy array [num_windows x frame_length] with axis=0 arg
      y_framed = librosa.util.frame(y, frame_length=frame_length,␣
       ↪hop_length=hop_length, axis=0)
      y_filt_framed = librosa.util.frame(y_filt, frame_length=frame_length,␣
       ↪hop_length=hop_length, axis=0)
      y_framed.shape
```

```
[22]: (194, 400)
```

5.2.2 Windowing

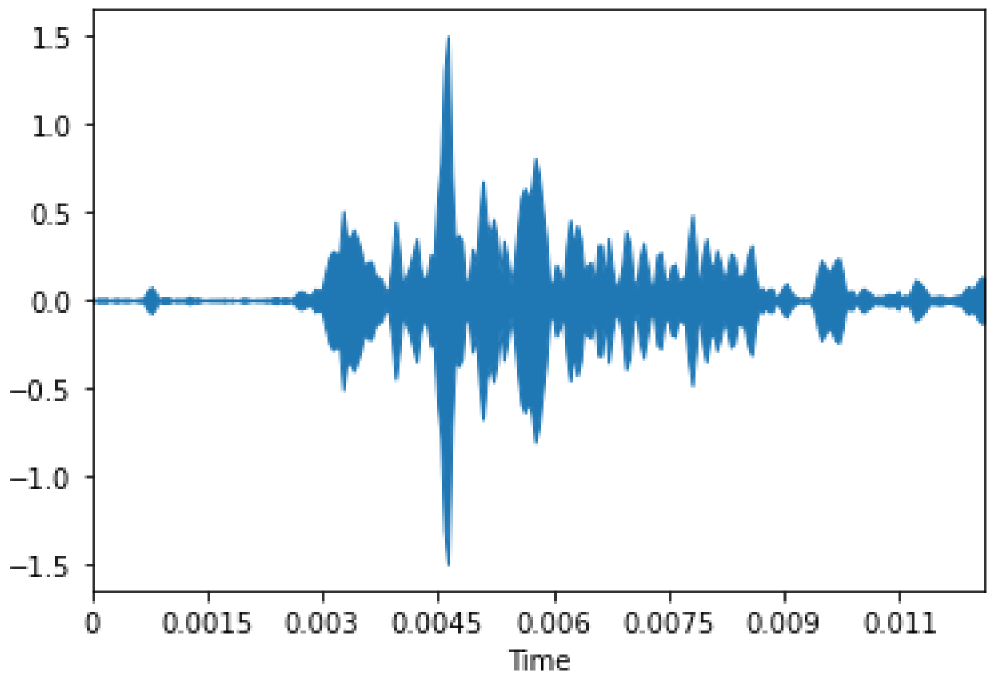
```
[23]: y_hamming = y_framed * np.hamming(frame_length)
      y_filt_hamming = y_filt_framed * np.hamming(frame_length)
      y_hanning = y_framed * np.hanning(frame_length)
      y_filt_hanning = y_filt_framed * np.hanning(frame_length)
```

```
[24]: _ = librosa.display.waveplot(y_hamming[100])
```

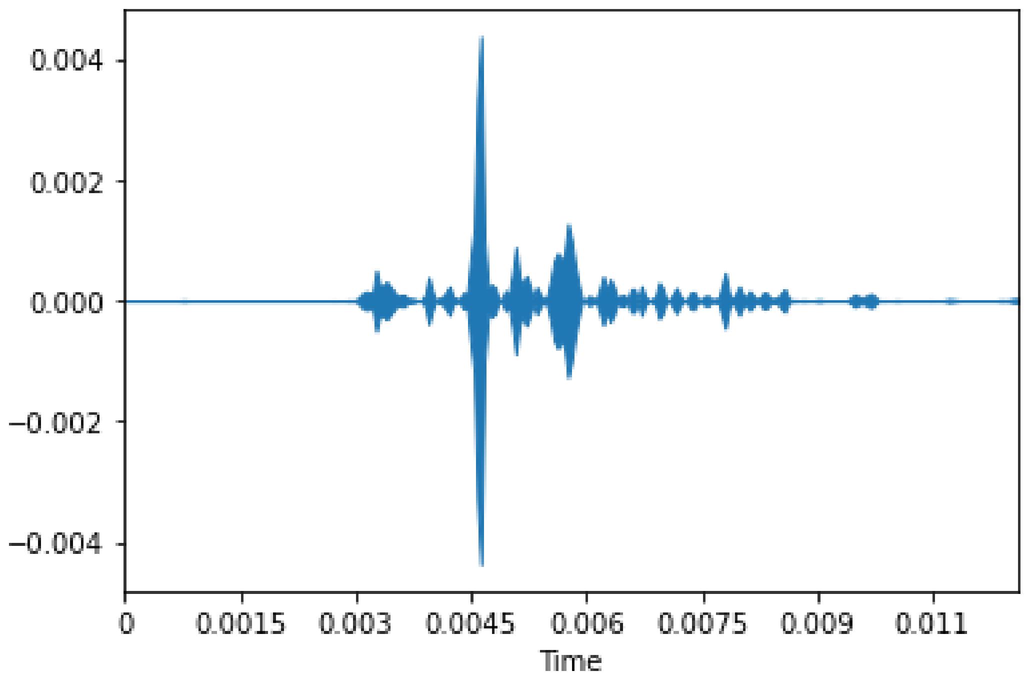


5.2.3 Fourier Transform

[26]:
```
# Compute the one-dimensional discrete Fourier Transform for real input.

# This function computes the one-dimensional n-point discrete Fourier Transform␣
↪(DFT)
# of a real-valued array by means of an efficient algorithm called the Fast␣
↪Fourier Transform (FFT).

nfft = 512
magnitude_frames = np.absolute(np.fft.rfft(y_hamming, nfft))
# how to display these?
_ =librosa.display.waveplot(magnitude_frames[0])
```



[27]:
```
power_frames = ((nfft ** -1) * (magnitude_frames ** 2))
_ = librosa.display.waveplot(power_frames[0])
```



5.2.4 Filter Banks

« What's the relationship between this and a bandpass filter (SincNet mentions this)? » « Are these learnable as SincNet claims? Do they have implications for speaker separation? »

```
[28]: low_mel_freq = 0
      high_mel_freq = (2595 * np.log10(1 + (sr/2)/700)) # converts Hz to Mel (it's a␣
       ↪new space?)
      low_mel_freq, high_mel_freq
```

```
[28]: (0, 2840.023046708319)
```

```
[31]: nfilters = 128
      mel_points = np.linspace(low_mel_freq, high_mel_freq, nfilters+2) # add 2 for␣
       ↪the bounds
      mel_points
```

```
[31]: array([   0.        ,   22.01568253,   44.03136507,   66.0470476 ,
               88.06273013,  110.07841266,  132.0940952 ,  154.10977773,
              176.12546026,  198.14114279,  220.15682533,  242.17250786,
              264.18819039,  286.20387292,  308.21955546,  330.23523799,
              352.25092052,  374.26660305,  396.28228559,  418.29796812,
              440.31365065,  462.32933319,  484.34501572,  506.36069825,
              528.37638078,  550.39206332,  572.40774585,  594.42342838,
              616.43911091,  638.45479345,  660.47047598,  682.48615851,
              704.50184104,  726.51752358,  748.53320611,  770.54888864,
              792.56457117,  814.58025371,  836.59593624,  858.61161877,
              880.6273013 ,  902.64298384,  924.65866637,  946.6743489 ,
              968.69003144,  990.70571397, 1012.7213965 , 1034.73707903,
             1056.75276157, 1078.7684441 , 1100.78412663, 1122.79980916,
             1144.8154917 , 1166.83117423, 1188.84685676, 1210.86253929,
             1232.87822183, 1254.89390436, 1276.90958689, 1298.92526942,
             1320.94095196, 1342.95663449, 1364.97231702, 1386.98799956,
             1409.00368209, 1431.01936462, 1453.03504715, 1475.05072969,
             1497.06641222, 1519.08209475, 1541.09777728, 1563.11345982,
             1585.12914235, 1607.14482488, 1629.16050741, 1651.17618995,
             1673.19187248, 1695.20755501, 1717.22323754, 1739.23892008,
             1761.25460261, 1783.27028514, 1805.28596768, 1827.30165021,
             1849.31733274, 1871.33301527, 1893.34869781, 1915.36438034,
             1937.38006287, 1959.3957454 , 1981.41142794, 2003.42711047,
             2025.442793  , 2047.45847553, 2069.47415807, 2091.4898406 ,
             2113.50552313, 2135.52120566, 2157.5368882 , 2179.55257073,
             2201.56825326, 2223.58393579, 2245.59961833, 2267.61530086,
             2289.63098339, 2311.64666593, 2333.66234846, 2355.67803099,
             2377.69371352, 2399.70939606, 2421.72507859, 2443.74076112,
             2465.75644365, 2487.77212619, 2509.78780872, 2531.80349125,
             2553.81917378, 2575.83485632, 2597.85053885, 2619.86622138,
             2641.88190391, 2663.89758645, 2685.91326898, 2707.92895151,
             2729.94463405, 2751.96031658, 2773.97599911, 2795.99168164,
             2818.00736418, 2840.02304671])
```

[32]:
```
# convert those mel points back to Hz with the inverse
hz_points = (700 * (10**(mel_points / 2595) - 1))
hz_points
```

[32]:
```
array([   0.        ,   13.80884544,   27.8900969 ,   42.24912811,
         56.89141881,   71.82255684,   87.04824026,  102.57427955,
        118.40659981,  134.55124302,  151.01437035,  167.8022645 ,
        184.92133214,  202.37810629,  220.17924886,  238.33155318,
        256.8419466 ,  275.7174931 ,  294.96539604,  314.59300086,
        334.60779791,  355.01742531,  375.82967183,  397.05247991,
        418.69394868,  440.76233702,  463.26606673,  486.21372576,
        509.61407148,  533.47603399,  557.80871957,  582.62141415,
        607.92358682,  633.72489348,  660.0351805 ,  686.86448851,
        714.22305618,  742.12132418,  770.56993915,  799.57975772,
        829.16185072,  859.32750737,  890.08823957,  921.45578634,
        953.44211827,  986.05944206, 1019.32020527, 1053.23710095,
       1087.8230726 , 1123.09131901, 1159.05529936, 1195.72873836,
       1233.12563143, 1271.26025009, 1310.14714741, 1349.80116351,
       1390.2374313 , 1431.4713822 , 1473.51875203, 1516.39558705,
       1560.11825005, 1604.70342661, 1650.16813148, 1696.52971504,
       1743.80586994, 1792.01463787, 1841.1744164 , 1891.30396606,
       1942.42241743, 1994.54927851, 2047.70444211, 2101.90819348,
       2157.18121804, 2213.54460924, 2271.01987666, 2329.62895421,
       2389.39420846, 2450.33844722, 2512.48492823, 2575.85736803,
       2640.47995101, 2706.37733865, 2773.57467891, 2842.09761588,
       2911.97229947, 2983.22539551, 3055.88409582, 3129.97612864,
       3205.52976922, 3282.57385058, 3361.13777453, 3441.25152289,
       3522.94566891, 3606.25138898, 3691.20047451, 3777.82534402,
       3866.15905558, 3956.23531938, 4048.08851062, 4141.7536826 ,
       4237.26658013, 4334.66365315, 4433.98207064, 4535.2597348 ,
       4638.53529555, 4743.84816524, 4851.23853371, 4960.7473836 ,
       5072.41650604, 5186.28851655, 5302.40687134, 5420.81588386,
       5541.56074175, 5664.68752404, 5790.24321876, 5918.27574089,
       6048.8339506 , 6181.96767194, 6317.72771182, 6456.16587943,
       6597.33500599, 6741.2889649 , 6888.08269234, 7037.77220819,
       7190.41463746, 7346.06823204, 7504.79239294, 7666.647693  ,
       7831.69589994, 8000.        ])
```

[33]:
```
bins = np.floor((nfft + 1) * hz_points / sr)
bins
```

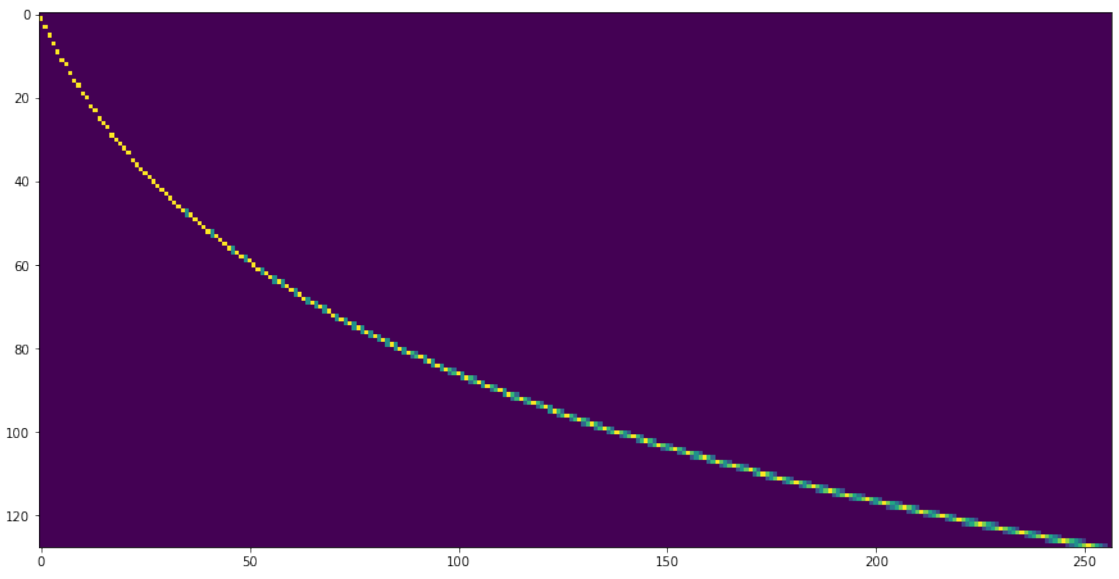
[33]:
```
array([  0.,   0.,   0.,   1.,   1.,   2.,   2.,   3.,   3.,   4.,   4.,
         5.,   5.,   6.,   7.,   7.,   8.,   8.,   9.,  10.,  10.,  11.,
        12.,   12.,   13.,   14.,   14.,   15.,   16.,   17.,   17.,   18.,   19.,
        20.,   21.,   22.,   22.,   23.,   24.,   25.,   26.,   27.,   28.,   29.,
        30.,   31.,   32.,   33.,   34.,   36.,   37.,   38.,   39.,   40.,   42.,
        43.,   44.,   45.,   47.,   48.,   50.,   51.,   52.,   54.,   55.,   57.,
        59.,   60.,   62.,   63.,   65.,   67.,   69.,   70.,   72.,   74.,   76.,
        78.,   80.,   82.,   84.,   86.,   88.,   91.,   93.,   95.,   97.,  100.,
       102.,  105.,  107.,  110.,  112.,  115.,  118.,  121.,  123.,  126.,  129.,
       132.,  135.,  138.,  142.,  145.,  148.,  152.,  155.,  159.,  162.,  166.,
       170.,  173.,  177.,  181.,  185.,  189.,  193.,  198.,  202.,  207.,  211.,
       216.,  220.,  225.,  230.,  235.,  240.,  245.,  251.,  256.])
```

[34]:
```
fbank = np.zeros((nfilters, int(np.floor(nfft / 2 + 1))))

for m in range(1, nfilters + 1):
    f_m_minus = int(bins[m-1])
    f_m = int(bins[m])
    f_m_plus = int(bins[m+1])

    for k in range(f_m_minus, f_m):
        fbank[m-1, k] = (k - bins[m-1]) / (bins[m] - bins[m-1])
    for k in range(f_m, f_m_plus):
        fbank[m-1, k] = (bins[m+1] - k) / (bins[m+1] - bins[m])
```

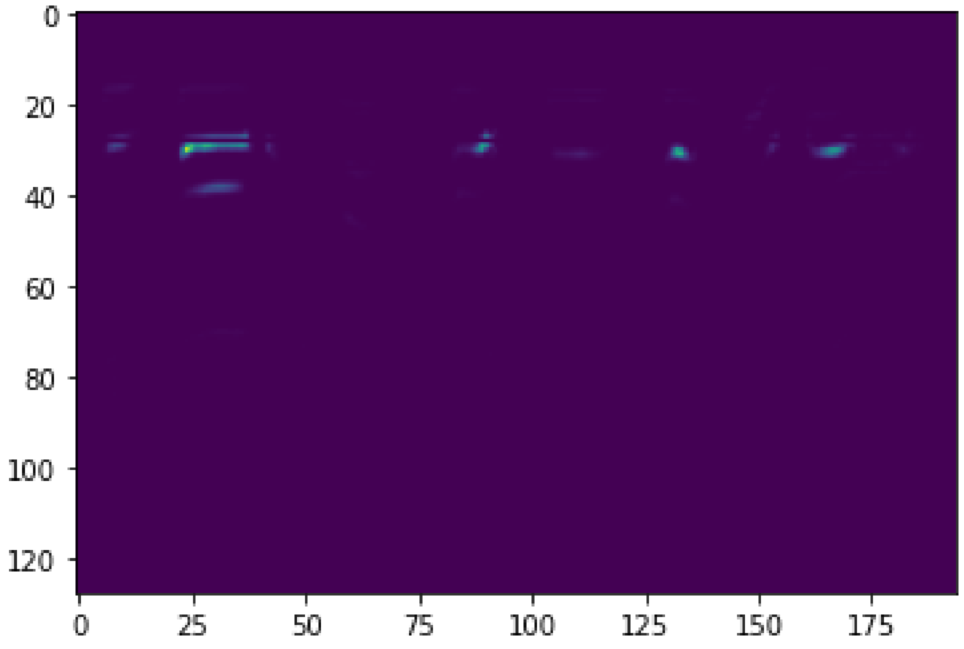
[35]:
```
fbank.shape
```

[35]:
```
(128, 257)
```

[36]:
```
# how to visualize these?
plt.figure(figsize=(15,15))
_ = plt.imshow(fbank)
```

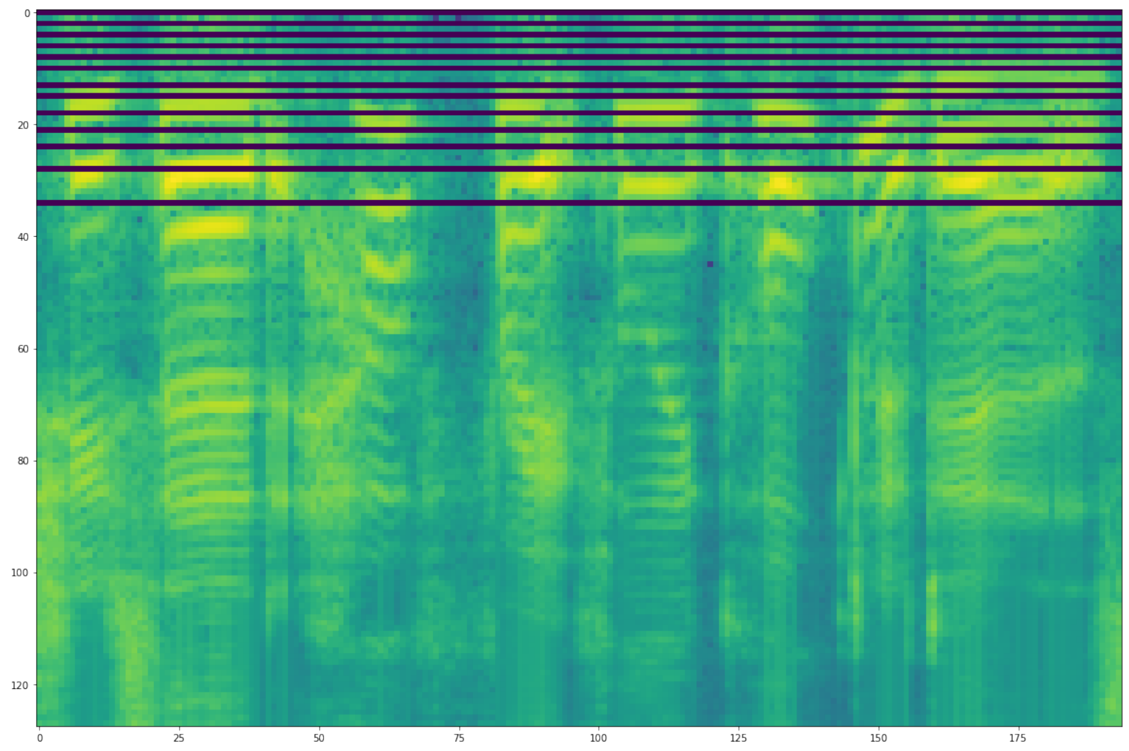


```
[37]: # now apply it
      filter_banks = np.dot(power_frames, fbank.T)
      _ = plt.imshow(filter_banks.T)
```



```
[38]: # convert to dB (why is this useful?)
      filter_banks = np.where(filter_banks == 0, np.finfo(float).eps, filter_banks)
      filter_banks = 20 * np.log10(filter_banks)
```

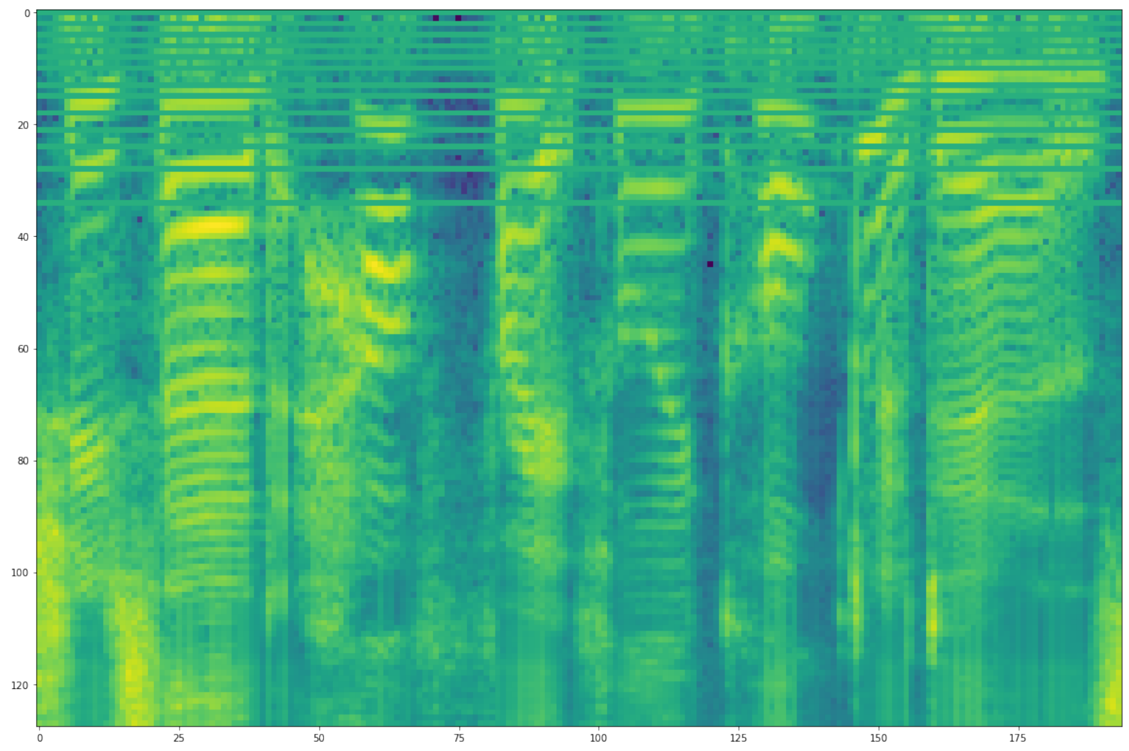
```
[39]: plt.figure(figsize=(20,20))
      _ = plt.imshow(filter_banks.T)
```



Mean Normalization

```
[40]: filter_banks_normed = filter_banks - np.mean(filter_banks, axis=0) + 1e-8
```

```
[41]: plt.figure(figsize=(20,20))
      _ = plt.imshow(filter_banks_normed.T)
```

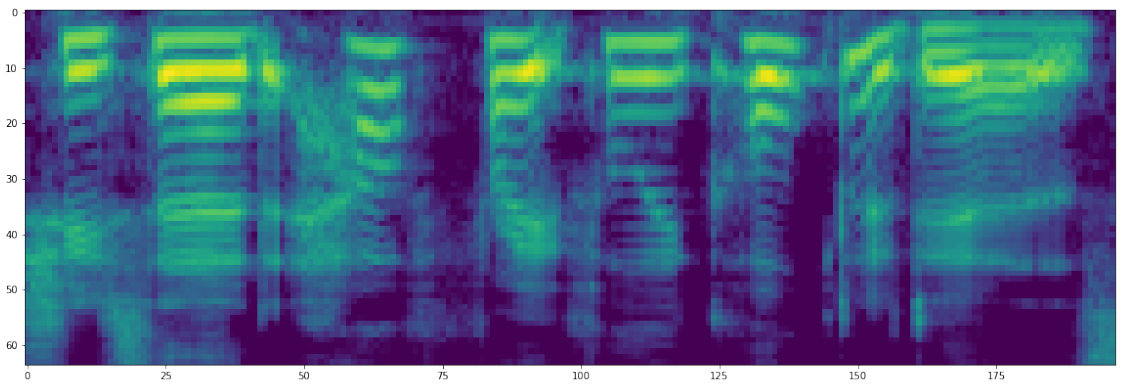


5.3 Getting there with Librosa

« Compare pre-emphasis & not along with mean & max & rms normalization »

[66]:
```
s = librosa.feature.melspectrogram(y_mean_norm, sr=sr, n_fft=nfft,␣
 ↪hop_length=hop_length, win_length=frame_length, n_mels=64)
```

[67]:
```
plt.figure(figsize=(20,7))
# _ = plt.imshow(np.log10(s))
_ = plt.imshow(librosa.power_to_db(s, ref=np.max))
# _ = librosa.display.specshow(librosa.power_to_db(s, ref=np.max),␣
 ↪y_axis='mel', x_axis='time', sr=sr)
```


0
10
20
30
40
50
60
0
25
50
75
100
125
150
175