cpp_filter_bank

November 10, 2020

[1]:
```
import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt

from librosa.core.time_frequency import fft_frequencies, mel_frequencies
```

0.1 librosa

[2]:
```
n_mels = 40
n_fft = 512
sr = 16000
norm = 1
```

[3]:
```
weights = np.zeros((n_mels, int(1 + n_fft // 2)))
weights.shape
```

[3]:
```
(40, 257)
```

[4]:
```
fftfreqs = fft_frequencies(sr=sr, n_fft=n_fft)
fftfreqs.shape
```

[4]:
```
(257,)
```

[5]:
```
mel_f = mel_frequencies(n_mels + 2, fmin=0.0, fmax=sr/2, htk=False)
mel_f.shape
```

[5]:
```
(42,)
```

[6]:
```
fdiff = np.diff(mel_f)
fdiff.shape
```

[6]:
```
(41,)
```

[7]:
```
ramps = np.subtract.outer(mel_f, fftfreqs)
ramps.shape
```

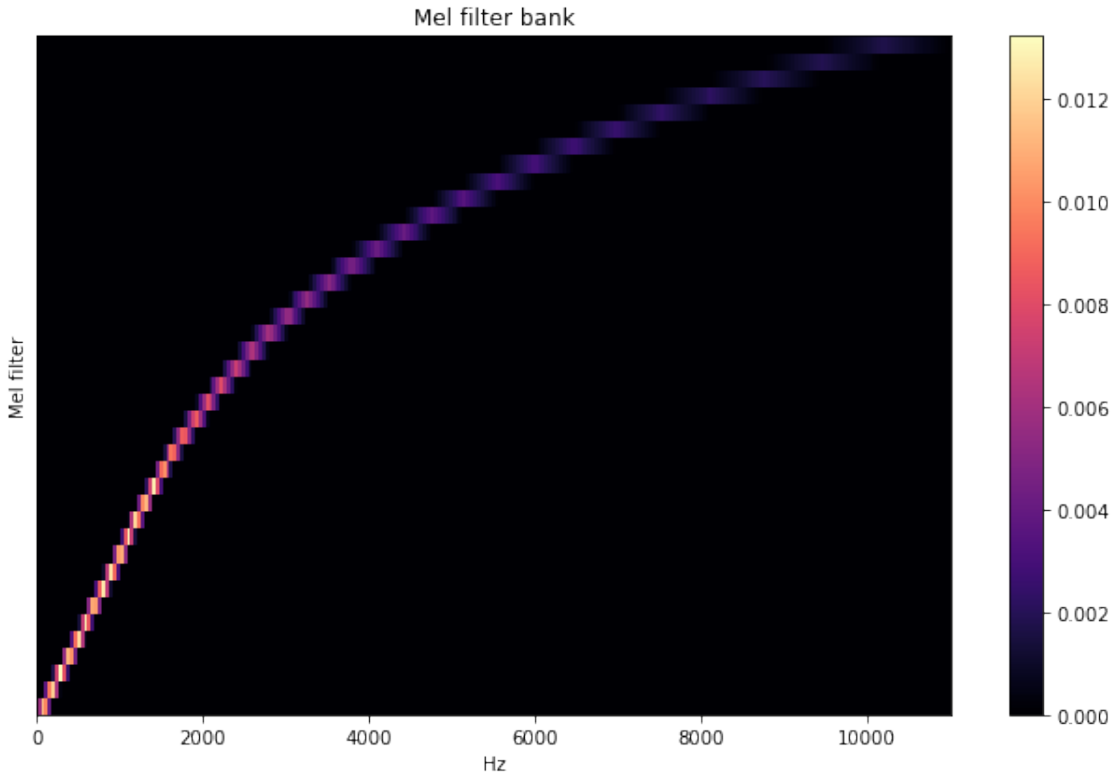
[7]: (42, 257)

[8]:
```
for i in range(n_mels):
    # lower and upper slopes for all bins
    lower = -ramps[i] / fdiff[i]
    upper = ramps[i+2] / fdiff[i+1]

    # .. then intersect them with each other and zero
    weights[i] = np.maximum(0, np.minimum(lower, upper))

if norm == 1:
    # Slaney-style mel is scaled to be approx constant energy per channel
    enorm = 2.0 / (mel_f[2:n_mels+2] - mel_f[:n_mels])
    weights *= enorm[:, np.newaxis]
```

[9]:
```
weights.shape
```

[9]: (40, 257)

[10]:
```
plt.figure(figsize=(9,6))
librosa.display.specshow(weights, x_axis='linear')
plt.ylabel('Mel filter')
plt.title('Mel filter bank')
plt.colorbar()
plt.tight_layout()
plt.show()
```



0.2 C++

Now, we do the same but in C++,

```
float * mel_filters(int nfilter, int sr = SIGNAL_RATE) {
  float low_freq = 0.0;
  float high_freq = (2595 * std::log10(1 + (sr/2)/ 700.0f));
  float step = (high_freq - low_freq) / (nfilter+1);

  float * filters = new float[nfilter+2];
  filters[0] = 0.0f;
  for (int i = 1; i < nfilter+2; i++) {
      filters[i] = filters[i-1] + step;
  }
  return filters;
}

void mel_to_hz(float x[], int size) {
  for (int i = 0; i < size; i++ ) {
    x[i] = (700 * (std::pow(10, x[i] / 2595.0f) - 1));
  }
};
```

```
float ** filter_bank(int n_mels, int sr = 16000, int n_fft = 512) {

  // mel scale
  float * filts = mel_filters(n_mels, sr);
  mel_to_hz(filts, n_mels+2);

  // difference between mel steps
  float fdiff[n_mels+1];
  for (int i = 0; i < n_mels + 1; i++) {
    fdiff[i] = filts[i+1] - filts[i];
  }

  // FFT frequencies
  float fft_freqs[1 + n_fft / 2];
  float fft_freq_step = (sr * 1.0 / 2) / (n_fft / 2);
  float fft_freq = 0.0;
  for (int i = 0; i < 1 + n_fft / 2; i++) {
    fft_freqs[i] = fft_freq;
    fft_freq += fft_freq_step;
  }

  // outer subtraction: filts - fft_freqs
  float ramps[n_mels+2][1 + n_fft/2];
  for (int i = 0; i < n_mels+2; i++) {
    for (int j = 0; j < 1 + n_fft/2; j++) {
      ramps[i][j] = filts[i] - fft_freqs[j];
    }
  }

  // now build our filter bank matrix
  float ** weights = new float*[n_mels];

  for (int i = 0; i < n_mels; i++) {

    float * w = new float[1+n_fft/2];
    for (int j = 0; j < 1 + n_fft/2; j++) {
      float lower = -1.0 * ramps[i][j] / fdiff[i];
      float upper = ramps[i+2][j] / fdiff[i+1];
      float bound = lower < upper ? lower : upper;
      w[j] = 0.0 > bound ? 0.0 : bound;
    }

    weights[i] = w;
  }
  // Slaney normalize
  float enorm;
  for (int i = 0; i < n_mels; i++) {
```

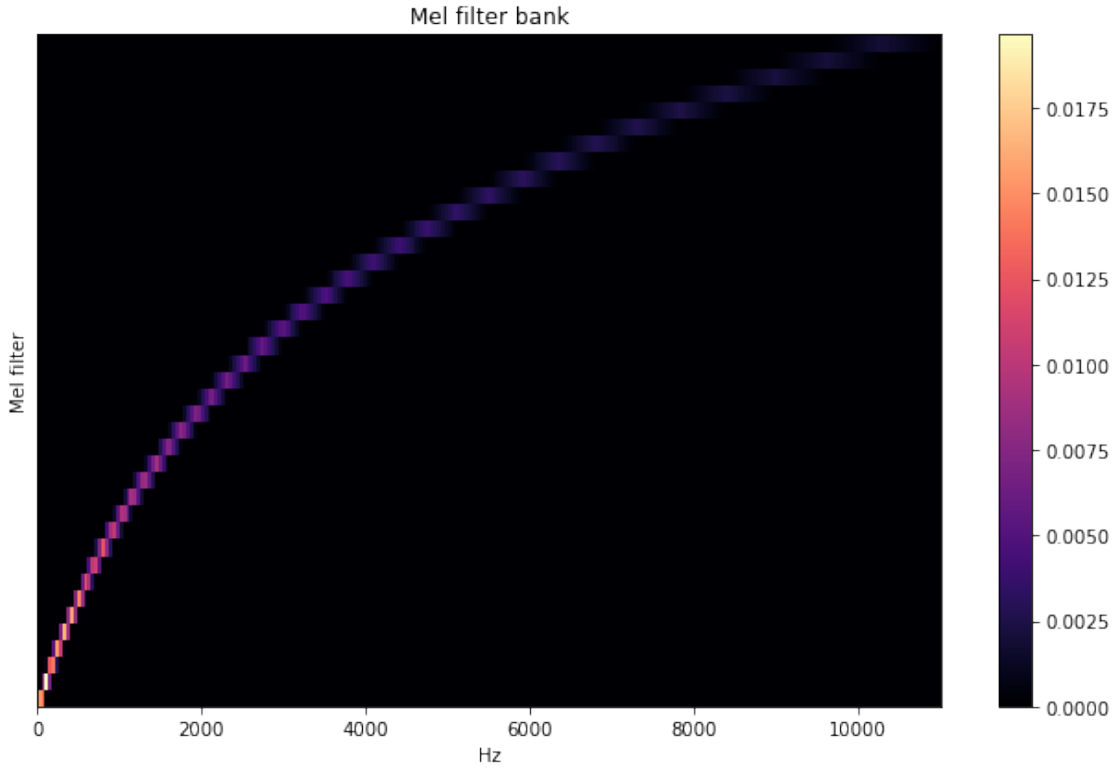
```
      enorm = 2.0 / (filts[i+2] - filts[i]);
      for (int j = 0; j < 1 + n_fft/2; j++) {
        weights[i][j] *= enorm;
      }
    }

    return weights;
  }
```

[14]:
```
def convert_cpp_file(path):
    frames = []
    with open(path, 'r') as f:
        for i, line in enumerate(f):
            window = line.rstrip().split(',')[:-1]
            frames.append([ float(val) for val in window ])
    return np.array(frames)
```

[15]:
```
fb = convert_cpp_file("/home/thomas/Dir/ccny/ccny-masters-thesis/cpp/out/
↪arduino/filter_bank.txt")
fb.shape
```

[15]:
```
(40, 257)
```

[16]:
```
plt.figure(figsize=(9,6))
librosa.display.specshow(fb, x_axis='linear')
plt.ylabel('Mel filter')
plt.title('Mel filter bank')
plt.colorbar()
plt.tight_layout()
plt.show()
```


Mel filter bank
Mel filter
Hz
0
2000
4000
6000
8000
10000
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000