Separation of drums from music signals

Yan Feng 281679

The goal of this project

As the music signal always contains the harmonic and percussive signals, it is hard to distinguish then when they simultaneously accrue in one audio material. In order to separate harmonic signal from percussive signal, I need to apply an algorithm that iteratively searches for such spectrograms that maximize the anisotropy of them and evaluate its performance.

The assumption based on this problem

It is known that the harmonic component includes the stable pitch and generates parallel ridges with smooth temporal envelopes, but percussive tone forms vertical ridge with a wide band spectral envelope.

Based on this principle, the diffusion-like process of harmonic and percussive energy distribution with a balance by complementary diffusion could be applied in this place. In my assumption, with the iterative number increases, the harmonic energy distribution diffuses horizontally and concentrates vertically on the spectrogram and percussive energy distribution has the contrary performance.

The basic solution to this problem

Since I need to implement the anisotropy properties of harmonic and percussive signals on the spectrum, the first step is to calculate the STFT of an input signal. And then apply the algorithm which executes the diffusion-like process of harmonic and percussive energy distribution. At last, transform the binarized results into waveform with inverse STFT. In all, the separated results would be obtained.

The exact implementation

To begin this project, the test material needs to be load and its data shall be transformed into correct data type “float64”.

The algorithm process could be divided into STFT transformation, initialization, iteration, binarization, ISTFT convention.

1. STFT transformation: When calculating the STFT of an input signal, the window length and type had been assigned as 1024 and hanning window as figure1.

```
#Apply the STFT function on the input data
def STFT(x,fs,winlen,han):
    f, t, X = stft(x, fs=fs, window=han, nperseg=winlen, noverlap=winlen / 2, nfft=winlen, detrend=False,
                   return_onesided=True, padded=True, axis=-1)
    return X
```

Fig. 1. STFT transformation

2. Initialization: To calculate a range-compressed version of the power spectrum by assigning the gamma as 0.3. And then initialize the harmonic and percussive components with the half of this compressed version as figure2.

```
#Intial the necessary parameters
y = 0.3
k=10
alpha=0.3
winlen_ms = 20.0  # milliseconds
winlen = int(2.0 ** np.ceil(np.log2(np.float(fs) * winlen_ms / 1000.0)))
han = signal.get_window('hann', int(winlen))
```

Fig. 2. Initialization

3. Iteration and binarization: To enhance the anisotropy of them, the iteration must be applied on them. In this way, update the harmonic and percussive components with the gradient which is the second order derivation of them. After the iterations, take the last results and binarize them. In this project, I had used “H_updated” to collect all updated harmonic components. And “P_updated” has the similar function. During the updating, the “gradient” function had been called to calculate the gradient of the specified index. And the binarization part is also included in the “update” function so that it could use the results after updating. The figure 3 illustrates the process of iteration and binarization and the figure 4 reveals the solution to the gradient.

```
#Update the harmonic and precussive compoents iteratively and binarize the seperation result
def update(H_updated,P_updated,k,alpha):
    [h_len,i_len]=np.shape(H_updated[0])
    #Execute the ieration
    for k_index in np.arange(k-1):
        H0= H_updated[k_index]
        P0 = P_updated[k_index]
        H1=np.zeros(np.shape(H0))
        P1=np.zeros(np.shape(P0))
        #Calculate the next elemnets with the gradient and previos one
        for h in np.arange(0, h_len-1):
            for i in np.arange(0, i_len-1):
                max=float(np.max([H0[h,i]+gradient(H0,P0,h,i,alpha),0]))
                w=float(W[h,i])
                H1[h,i]=np.minimum(max,w)
                P1[h,i]=W[h,i]-H1[h,i]
        H_updated.append(H1)
        P_updated.append(P1)

    #Take the last elements of the collections
    H_max_1 = H_updated[-1]
    P_max_1 = P_updated[-1]
    H_max = np.zeros(np.shape(H_max_1))
    P_max = np.zeros(np.shape(P_max_1))
    #Excute the binarization process
    for i in np.arange(h_len):
        for j in np.arange(i_len):
            if  H_max_1[i,j]<P_max_1[i,j]:
                H_max[i,j]=0
                P_max[i,j]=W[i,j]
            else:
                H_max[i, j] = W[i, j]
                P_max[i, j] =0
    return [H_max,P_max]
```

Fig. 3. Iteration and binarization

```
#Calculate the gradient of conrresponding parameters
def gradient(H0,P0,h,i,alpha):
    partH=H0[h,i-1]-2*H0[h,i]+H0[h,i+1]
    partP=P0[h-1,i]-2*P0[h,i]+P0[h+1,i]
    grad=alpha*partH/4-(1-alpha)*partP/4
    return grad
```

Fig. 4. Gradient

4. ISTFT convention: Convert the binarized results into waveform with inverse STFT with the same parameter as the STFT function as the figure 5.

```
#Apply the inverse STFT function
def iSTFT(X,fs,winlen,han):
    _, x = istft(X, fs=fs, window=han, nperseg=winlen, noverlap=winlen / 2, nfft=winlen, input_onesided=True)
    return x
```

Fig. 5. iSTFT transformation

By observing these through audio and image materials, I could achieve my goal.

Evaluation

When the different numbers of iteration had been applied, the results could be observed clearly.

The figure 6 is taken at iteration number is 2 and the corresponding SNR is 311.189dB.

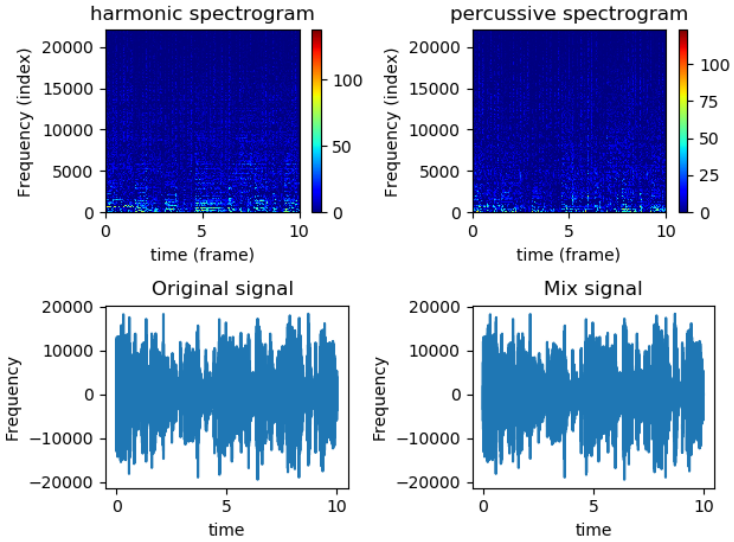

Fig. 6. Frequency and time domain visualization when it is the second iteration

The figure 7 is taken at iteration number is 5 and the corresponding SNR is 311.169 dB.

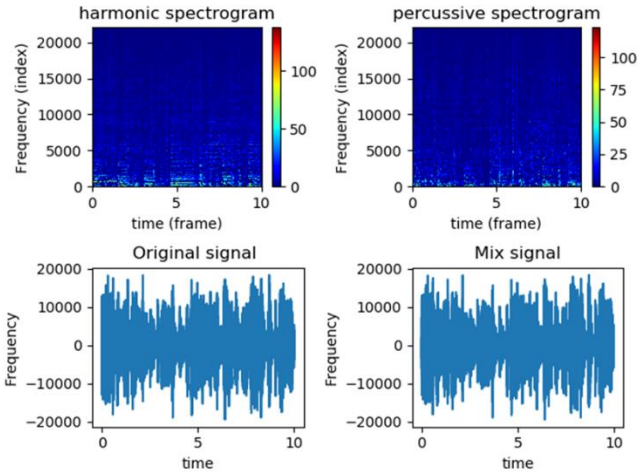

Fig. 7. Frequency and time domain visualization when it is the fifth iteration

The figure 8 is taken at iteration number is 10 and the corresponding SNR is 311.155dB.

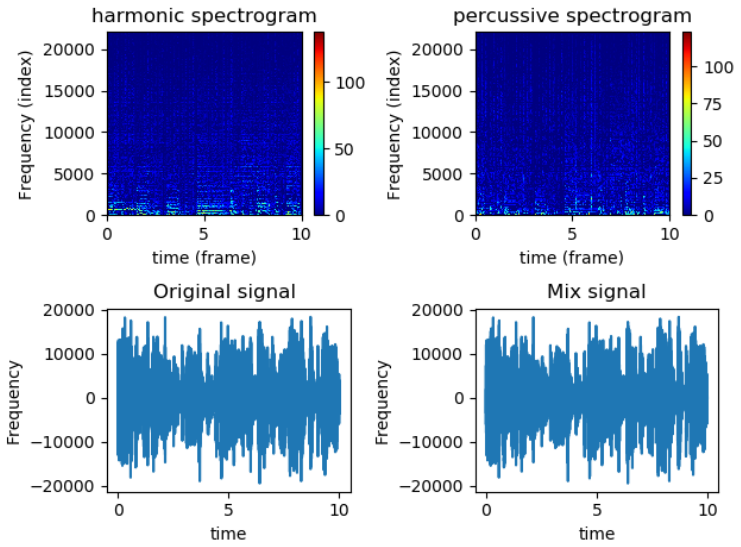

Fig. 8. Frequency and time domain visualization when it is the tenth iteration

Related question:

- What kind of audio material is the algorithm limited to and why?

The first kind of audio material which should be limited is the stereo signals. It needs to be processed the both channels separately. However, this algorithm works only for mono signal.

Another situation should be considered is the audio material includes a lot of the pitch-varying pitches, such as the bass drum and the singing voice. They would lead to the wrong result while using this algorithm.

- How should the separation quality be measured and assessed?

The first method is to use the energy radio of each track included in harmonic and percussive signal and draw a figure to observe the performance as it is described in the paper .

The second method I used in this experience is to compare the SNR. If its value is larger, the result is more precise.

The third choice is to compare the harmonic and percussive signal spectrums. And evaluate them according to their properties. Harmonic components should contain stable tones that produce parallel ridges with smooth time envelopes, but the pulse tones form a vertical ridge with the broadband spectral envelope.

The last method is to listen the final wav files. Although this approach is subjective, it is effective when the modification had been made,