DSP Programming Assignment

AI대학원 20231186 최유선

Programming Assignment #1 요구사항
[overlap-add method를 이용한 speech signal DFT-IDFT]

- input.wav file 불러오기
- 80개 샘플의 각 프레임에 대해 FFT를 수행하고 IFFT를 통해 신호를 재구성
- 100번째 프레임에서 magnitude spectrum 그리기
- time-domain에서 input signal과 ouput signal 간 difference 그리기

1) 라이브러리 불러오기

```
import torch
import torchaudio
import numpy as np
import matplotlib .pyplot as plt
```

- torch: 텐서 연산을 위한 Pytorch 라이브러리
- torchaudio: 오디오 처리를 위한 Pytorch 라이브러리
- numpy: 배열 연산을 위한 라이브러리
- matplotlib.pyplot: 시각화 라이브러리

2) 오디오 파일 불러오기

```
frame_size =80
waveform , sr =torchaudio .load('input.wav')
waveform =waveform [0]
```

torchaudio를 통해 input.wav 파일을 로드합니다. 불러들인 신호는 전체 18859개의 샘플, sampling rate는 48000입니다. 싱글 채널 오디오 처리를 하기 위해, waveform에서 첫 번째 채널만 가져옵니다.

2) overlap_add 함수 구현

```
def overlap_add (signal , frame_size ):
   frame_num = (len (signal ) +frame_size -1 ) //frame_size
   output_signal =torch .zeros(len (signal ) +frame_size )
   for i in range (frame_num ):
     start =i *frame_size
     end =start +frame_size
     if end < (i +1 ) *frame_size :
       frame =signal [start :]
     else :
       frame =signal [start :end ]

     # Zero-padding if the frame is smaller than frame_size
     if len (frame ) <frame_size :
       frame =torch .nn.functional.pad(frame , (0 , frame_size -len (frame )))
```

```
    spectrum =torch .fft.fft(frame )
    reconstructed_frame =torch .fft.ifft(spectrum ).real
    output_signal [start :end ] +=reconstructed_frame
  return output_signal [:len (signal )]
```

먼저, 전체 신호를 커버하는데 필요한 프레임 수를 계산하여 frame_num 변수에 저장합니다. output_signal을 0으로 초기화하여 overlap-add 과정을 수행할 수 있는 길이로 만들어줍니다. 각 프레임에 대해, 80 샘플 만큼 shift하면서 세그먼트를 추출하고 추출된 프레임이 포함하고 있는 샘플의 수가 80 보다 작다면, zero-padding하여 남은 샘플들을 0으로 채워줍니다. 프레임마다 FFT를 수행하고 IFFT를 통해 신호를 재구성한 것에서 실수 부분만을 가져옵니다. 재구성된 프레임을 output_signal에 합산하여 최종적으로 원래의 신호를 만들어냅니다.

3) plot_magnitude_spectrum 함수 구현

```
def plot_difference (input_signal , output_signal ):
  difference =input_signal -output_signal
  plt .figure (figsize =(10 ,4 ))
  plt .plot (difference )
  plt .title ("Difference between input and output signals")
  plt .xlabel ("Sample")
  plt .ylabel ("Difference")
  plt .show ()
```

torch.abs를 사용해 스펙트럼의 magnitude를 계산하고 maplotlib을 통해 magnitude spectrum을 시각화합니다.

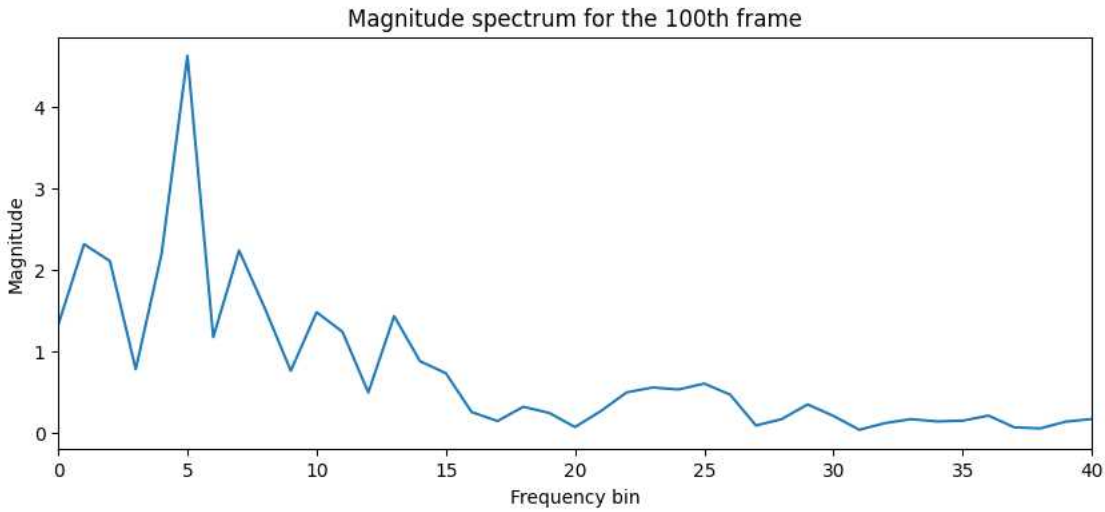
4) plot_difference 함수 구현

```
def plot_difference (input_signal , output_signal ):
  difference =input_signal -output_signal
  plt .figure (figsize =(10 ,4 ))
  plt .plot (difference )
  plt .title ("Difference between input and output signals")
  plt .xlabel ("Sample")
  plt .ylabel ("Difference")
  plt .show ()
```

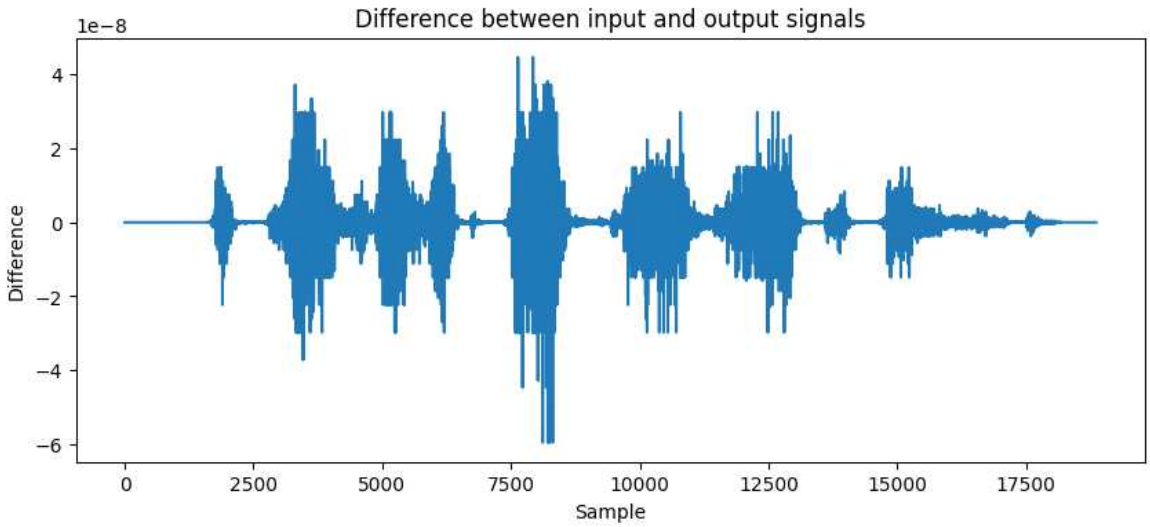
시간 영역에서 입력 신호와 출력 신호의 차이를 시각화합니다.

```
# overlap-add method
output_signal =overlap_add (waveform , frame_size )

# Plot of magnitude spectrum for the 100th frame
start =99 *frame_size
end =start +frame_size
ff =waveform [start :end ]
plot_magnitude_spectrum (ff )

# Plot of the difference between input and output (time domain) signals
plot_difference (waveform.squeeze(0) , output_signal )
# Save output file
torchaudio .save("output.wav", output_signal , sr )
```

앞서 구현한 overlap_add 함수를 통해, 재구성된 출력 신호를 얻을 수 있습니다. 재구성한 신호의 100번째 프레임은 아래와 같습니다.



FFT 결과로 나온 spectrum은 대칭성을 갖기 때문에, 프레임 길이가 80일 때, 주파수 성분은 절반인 40까지만 보면 됩니다.



Programming Assignment #2 요구사항

[overlap-add /overlap-save method를 이용한 speech signal DFT-LPF-IDFT]

- input.wav file 불러오기
- LPF는 moving average filter, i.e., $h[n]=1, 0 \le n \le 10\,(P=11)$

1) overlap-add method의 경우,

- Frame shift는 80 샘플 $(L=80)$
- 80 샘플마다, 128 point FFT 수행 $(N=128)$
- filter에 FFT한 값과 곱하기
- 128 point IFFT 하여 time-domain 신호로 재구성

2) overlap-save method의 경우,

- Frame shift는 80 샘플 $(L-P+1=80 \rightarrow L=90)$
- 80개 샘플마다, 입력의 90개 샘플에 대한 128 point FFT 수행 $(N=128)$
- filter에 FFT한 값과 곱하기
- 128 point IFFT 하여 time-domain 신호로 재구성

공통

- 각 method에 대해, 필터링 이전 입력 신호의 100번째 프레임의 magnitude spectrum 그리기
- overlap-add / overlap-save method 출력 결과 비교하기

1) overlap_add2 함수 구현

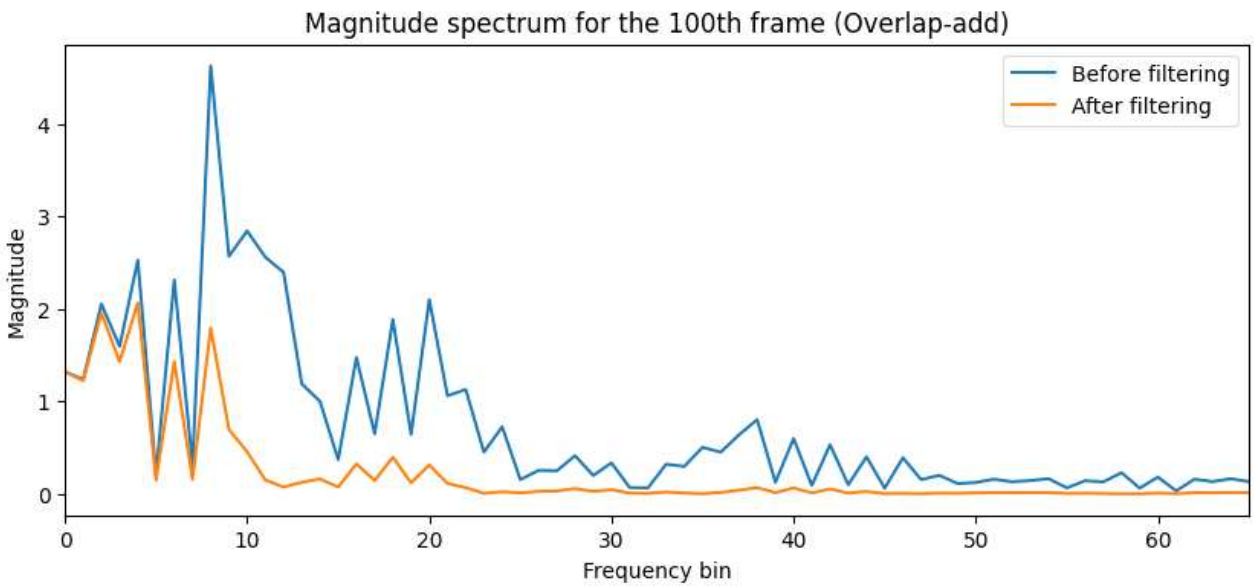
```
def overlap_add2 (signal , LPF , frame_shift , fft_point , overlap_length ):
   output_length1 =waveform .size(0 ) +overlap
   output_signal1 =torch .zeros (output_length1 )
   h_fft =torch .fft .fft(h , N )
   for i in range (0 , waveform .size(0 ), L ):
     frame =waveform [i :i +L ]
     frame_fft =torch .fft .fft(frame , N )
     filtered_frame_fft =frame_fft *h_fft

     if i ==99 *80 :
       before_filtering1 =frame_fft
       after_filtering1 =filtered_frame_fft
     filtered_frame =torch .fft .ifft(filtered_frame_fft )
     frame_length =min (N , output_signal1 .size (0 ) -i )
     output_signal1 [i :i +frame_length ] +=filtered_frame [:frame_length ].real

   return output_signal1 [:waveform .size(0 )], before_filtering1 , after_filtering1
```

순서대로, 필터링할 signal (waveform), LPF (h), frame_shift (L), fft_point (N), overlap_length (overlap)을 매개변수로 받는 함수를 구현하였습니다. overlap_add2는 80 샘플만큼 shift하면서 프레임을 추출하고 각 프레임마다 128 point FFT를 수행합니다. 이때의 중첩되는 프레임 길이는 128-80=48입니다. LPF에 FFT한 값을 각 프레임마다 곱해주고 IFFT 해주면, time-domain 상 필터링된 프레임을 구할 수 있게 됩니다. 100번째 프레임의 magnitude spectrum은 필터링 전후 프레임을 각각 나타내도록,

before_filtering1, after_filtering1 변수에 값을 지정해주었습니다. 신호를 재구성할 때, 마지막 프레임이 출력 신호의 끝을 초과하지 않도록 하는 작업이 필요하기 때문에, frame_length가 현재 프레임에서 output_signal1의 길이까지 남은 길이와 기존 프레임 크기 (128) 중 작은 값을 선택하도록 설계하였습니다. 이후, 각 프레임을 합산하여 신호를 재구성합니다.

```
waveform , sr =torchaudio .load('input.wav')
waveform =waveform [0 ]

P =11
h =torch .ones (P ) /P
L =80
N =128
overlap =N -L

output2_1 =overlap_add2 (waveform , h , L , N , overlap )[0 ]
torchaudio .save("output2_1.wav", output2_1 .unsqueeze (0 ), sr )

# Plot of magnitude spectrum for the 100th frame
Y1 =abs (before_filtering1)
Y2 =abs (after_filtering1)
plt .figure (figsize =(10 ,4 ))
plt .plot (Y1 , label ='Before filtering')
plt .plot (Y2 , label ='After filtering')
plt .xlim (0 ,65 )
plt .title ("Magnitude spectrum for the 100th frame (Overlap-add)")
plt .xlabel ("Frequency bin")
plt .ylabel ("Magnitude")
plt .legend ()
plt .show ()
```

overlap_add2 함수를 통해, 재구성한 신호의 100번째 magnitude spectrum은 아래와 같습니다. lowpass filter로 인해, 필터링 전후, 저주파 대역의 magnitude가 크게 줄어든 것을 확인할 수 있습니다.

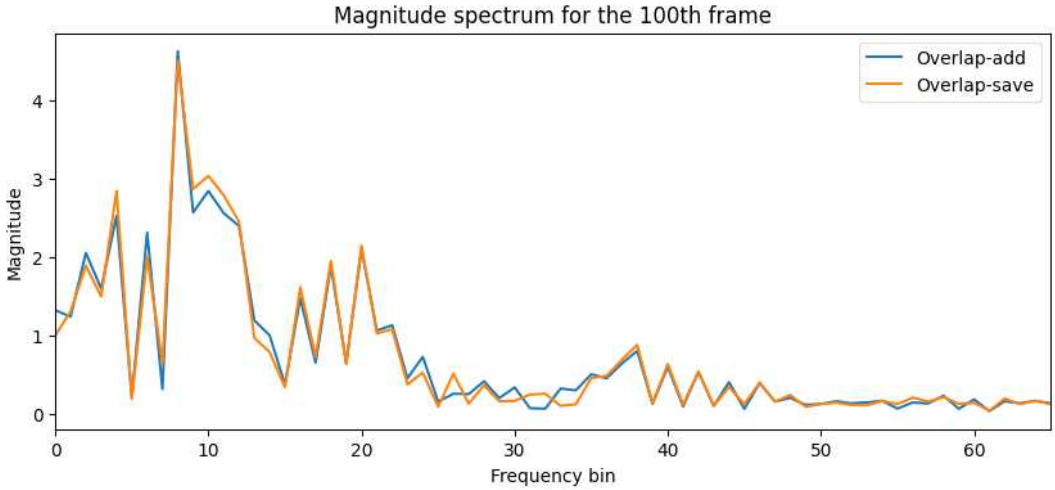


2) overlap_save 함수 구현

```
def overlap_save (signal , LPF , frame_size , fft_point , frame_shift ):
   h_fft =torch .fft .fft(h , n =N )
   num_frames = (waveform .size(0 ) -L +frame_shift ) //frame_shift
   output_signal2 =torch .zeros (waveform .size(0 ))
   for i in range (num_frames ):
     start_idx =i *frame_shift
     end_idx =start_idx +L

     if end_idx >waveform .size(0 ):
       break

     frame =waveform [start_idx :end_idx ]
     frame_padded =torch .cat ([frame , torch .zeros (N -L )])

     frame_fft =torch .fft .fft(frame_padded , n =N )
     filtered_frame_fft =frame_fft *h_fft
     filtered_padded =torch .fft .ifft(filtered_frame_fft , n =N ).real

     output_signal2 [start_idx :start_idx +frame_shift ] +=filtered_padded [P -1 :L ]
     if i ==99 :
       before_filtering2 =frame_fft
       after_filtering2 =filtered_frame_fft
   return output_signal2 [:waveform .size(0 )], before_filtering2 , after_filtering2
```

오디오 데이터에서 처리할 프레임 수를 계산하여 num_frames에 넣어주고 필터링된 신호를 저장할 텐서로 output_signal2를 초기화합니다. start_idx, end_idx에 현재 프레임의 시작 및 끝 인덱스를 계산해서 넣어주고, 프레임이 오디오 데이터의 끝에 다다르면 for문을 종료합니다. 현재 프레임을 추출해서 frame에 저장하고 frame_padded에 zero-padding을 추가합니다. 각 프레임에 FFT를 수행하고 lowpass filter를 곱한 뒤, IFFT를 사용하여 필터링된 프레임을 filtered_padded에 저장합니다.

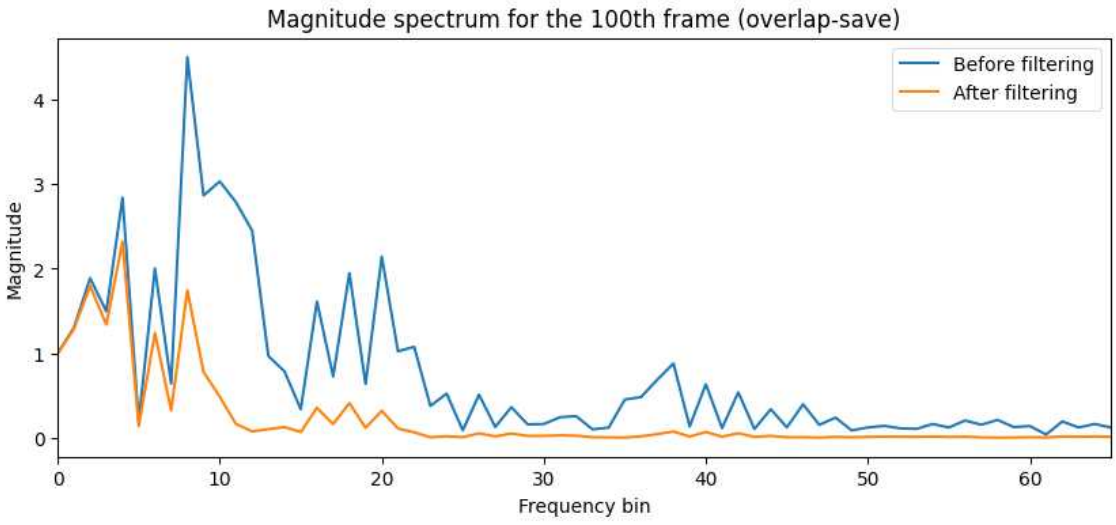
```
waveform , sr =torchaudio .load('input.wav')
waveform =waveform [0 ]

P =11
h =torch .ones (P ) /P
L =90
N =128
shift =80
output2_2 =overlap_save (waveform , h , L , N , shift )[0 ]
torchaudio .save('output2_2.wav', output_signal2.unsqueeze(0 ), sr )

Y3 =abs (before_filtering2)
Y4 =abs (after_filtering2)
plt .figure (figsize =(10 ,4 ))
plt .plot (Y3 , label ='Before filtering')
plt .plot (Y4 , label ='After filtering')
plt .xlim (0 ,65 )
plt .title ("Magnitude spectrum for the 100th frame (overlap-save)")
plt .xlabel ("Frequency bin")
plt .ylabel ("Magnitude")
plt .legend ()
plt .show ()
```



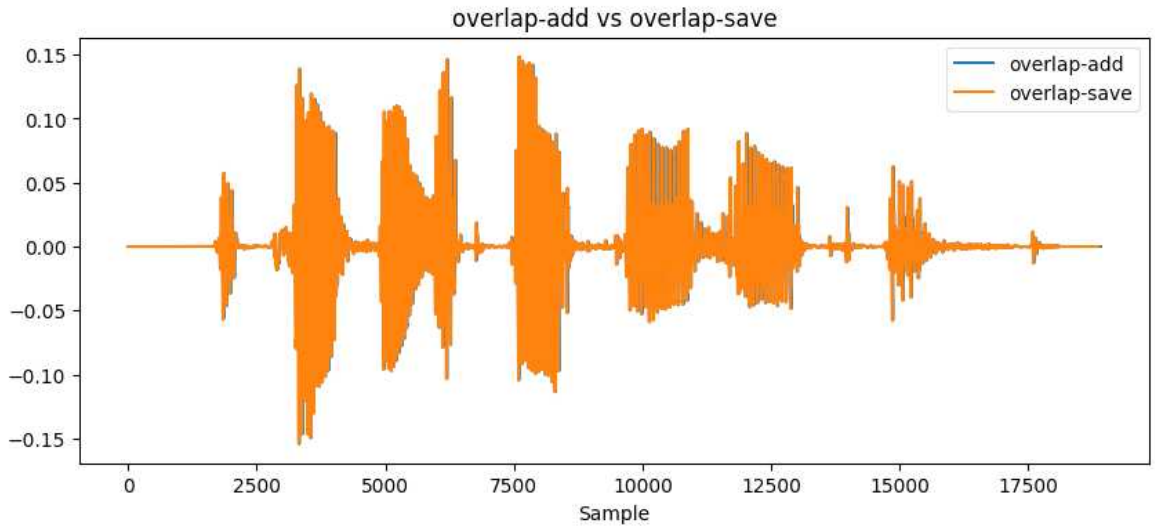
overlap_save 함수를 통해, 재구성한 신호의 100번째 magnitude spectrum은 아래와 같습니다. 이 방법 역시도 lowpass filter로 인해, 필터링 전후, 저주파 대역의 magnitude가 크게 줄어든 것을 볼 수 있습니다.

```
plt .figure (figsize =(10 ,4 ))
plt .plot (Y1 , label ='Overlap-add')
plt .plot (Y3 , label ='Overlap-save')
plt .xlim (0 ,65 )
plt .title ("Magnitude spectrum for the 100th frame")
plt .xlabel ("Frequency bin")
plt .ylabel ("Magnitude")
plt .legend ()
plt .show ()
```

lowpass filtering 전, overlap-add / overlap-save 방법을 frequency-domain의 magnitude spectrum 상에서 비교한 결과입니다. 이론적으로, 두 방법은 frequency-domain에서 같은 필터링 연산을 수행하기 때문에 동일한 결과를 얻어야 합니다. 하지만, 프레임 세그먼트화, 프레임 경계 처리, FFT/IFFT 연산 오차, zero-padding 처리 등 구현 방식이 조금 다르기 때문에, 100번째 프레임에서의 magnitude spectrum에서 조금 차이가 있는 걸 확인할 수 있습니다.



time-domain에서 비교하면, 육안으로 봤을 때, overlap-add와 overlap-save가 거의 일치하는 것을 확인할 수 있습니다.



Programming Assignment #3 요구사항

[windowing과 overlap-add method를 이용한 speech signal DFT-LPF-IDFT]

- input.wav file 불러오기
- LPF는 moving average filter, i.e., $h[n] = 1, 0 \le n \le 10\,(P = 11)$

overlap-add method 사용

- Frame shift는 80 샘플 $(L = 80)$
- 분석에 128개 샘플 사용; 첫 24개 샘플은 이전 프레임의 마지막 24 샘플; 80 샘플은 현재 프레임에서 가져옴; 마지막 24 샘플은 0임
- window: 0-23, 총 24개 샘플은 48 point Hann window 첫 절반; 그 다음 56 샘플은 1.0; 마지막 24 샘플은 48 point Hann window의 나머지 절반
- 128개 샘플에 대해 128 point FFT 수행
- 필터의 FFT와 곱함
- 128 point IFFT를 계산하여 time-domain 신호를 재구성함

```
import scipy .signal
def overlap_add3 (signal , LPF , window , frame_shift , fft_point ):
   h_fft =torch .fft.fft(h , n =N )
   num_frames = (waveform .size(0 ) +L -1 ) //L
   output_signal_length =L *num_frames + (N -L )
   output_signal3 =torch .zeros(output_signal_length )
    padded_signal =torch .cat([torch .zeros(24 ), waveform , torch .zeros(N - (waveform
.size(0 ) %L ) -24 )])
   for i in range (num_frames ):
     start_idx =i *80
     end_idx =start_idx +N
```

```
    frame =padded_signal [start_idx :end_idx ]
    if frame .size(0 ) <N :
      frame =torch .cat([frame , torch .zeros(N -frame .size(0 ))])
    windowed_frame =frame *window
    frame_fft =torch .fft.fft(windowed_frame , n =N )
    filtered_fft =frame_fft *h_fft
    filtered_frame =torch .fft.ifft(filtered_fft ).real

    if i ==99 :
      before_filtering3 =torch .abs(frame_fft )
    output_signal3 [start_idx :start_idx +N ] +=filtered_frame
  return output_signal3 [24 :24 +waveform .size(0 )], before_filtering3
```

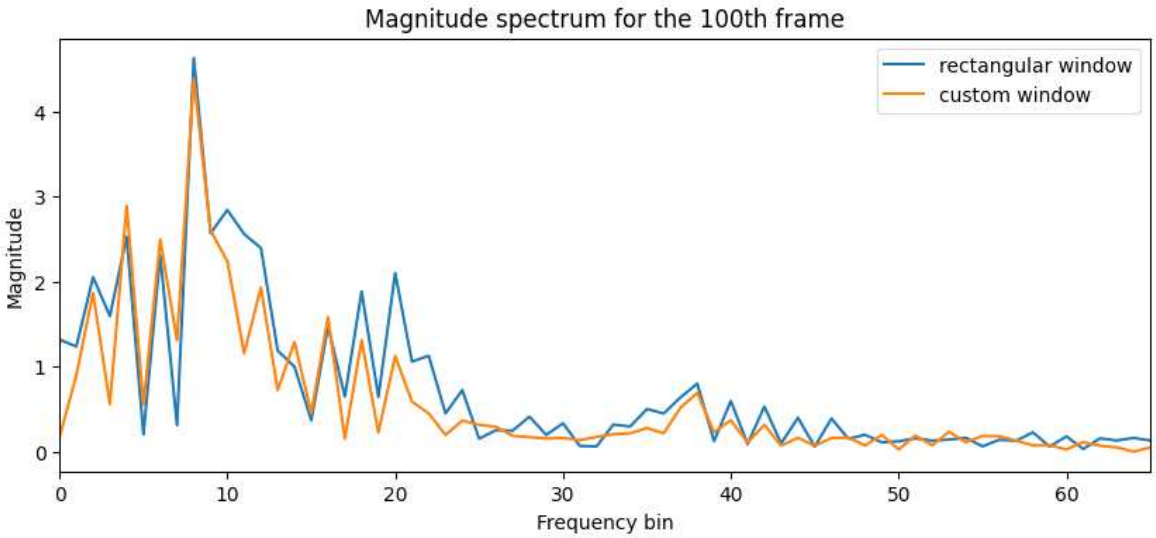
frame shift가 80이라고 주어졌기 때문에, 필요한 프레임 수와 출력 신호의 총 길이를 계산하여 변수에 저장해주었습니다. 신호에 앞뒤로 패딩을 추가함으로써, 필터링 과정에서 프레임 경계를 스무딩하기 위해, padded_signal 변수를 생성하여 앞쪽 24개 샘플에는 0값을, 뒤쪽 24개 샘플에는 프레임 크기에 맞춰 추가적인 0값을 추가하도록 설계하였습니다. 이후, 각 프레임에 대해 윈도우를 적용하고 FFT를 수행한 다음 필터의 FFT와 곱해 filtered_fft 값을 얻어주었습니다. 이를 IFFT 해주어, time-domain으로 변환하고 결과를 합산하여 앞의 0-23번째 샘플을 제외한 필터링된 신호를 반환하였습니다.

```
waveform , sr =torchaudio .load('input.wav')
waveform =waveform [0 ]
P =11
h =torch .ones(P ) /P
L =80
N =128
# Custom window
hann_window =scipy .signal.windows.hann(48 )
window =torch .cat([
  torch .tensor(hann_window [:24 ]),
  torch .ones(56 ),
  torch .tensor(hann_window [24 :]),
  torch .zeros(128 -104 )
])
output3 , before_filtering3 =overlap_add3 (waveform , h , window , L , N )
torchaudio .save('output3.wav', output3 .unsqueeze(0 ), sr )
```

custom window는 앞부분 24개 샘플과 뒷부분 24개 샘플에서 hann window로 설정하였고, 가운데는 길이 56인 rectangular window로 설정하였습니다. 마지막으로 총 길이가 128이 되도록 남은 128-104=24개 샘플에 0값을 넣어주었습니다.

```
Y5 =abs (before_filtering3 )
plt .figure (figsize =(10 ,4 ))
plt .plot (Y1 , label ='rectangular window')
plt .plot (Y5 , label ='custom window')
plt .xlim (0 ,65 )
plt .title ("Magnitude spectrum for the 100th frame")
plt .xlabel ("Frequency bin")
plt .ylabel ("Magnitude")
plt .legend ()
plt .show ()
```

앞에서 구현했던 rectangular window를 적용한 overlap-add 방법과 custom window를 적용한 overlap-add 방법을 100번째 프레임에서의 magnitude spectrum 상에서 비교한 결과입니다.



rectangular window만 쓴 전자와 달리, custom window를 쓴 후자의 경우, 설계 과정에서 hann window를 포함하고 있었기 때문에, main lobe의 gain 값이 상대적으로 조금 떨어지지만 side lobe의 attenuation이 더 큰 것을 확인할 수 있습니다.

```
plt .figure (figsize =(10 ,4 ))
plt .plot (output2_1 , label ="rectangular window")
plt .plot (output3 , label ="custom window")
plt .title ("rectangular window vs custom window")
plt .xlabel ("Sample")
plt .legend ()
plt .show ()
```

time-domain에서 비교하면, 육안으로 봤을 때, 어떤 window를 쓰든 눈에 띄는 차이가 없는 것을 확인할 수 있습니다.