음성데이터 증대 기법

Input data : wave form

Input data : spectrogram

참고1. Environmental Corruption	참고2. Speech Augmentation	참고3. SpecAugmentation
1. Noise Mixing	1. Speed perturbation	1. Time warping
2. Reverberation	2. Time dropout	2. Frequency masking
3. Environmental Corruption Lobe	3. Frequency dropout	3. Time masking
	4. Clipping	
	5. Augmentation Lobe	

Clean data + 첨가

Clean data – 손상

Environmental Corruption Tutorial

1. clean 음성 신호 + 잡음 신호 첨가
2. 장점 : 다른 노이즈 및 잔향(메아리)을 사용함으로 다양한 방식으로 인위적으로 신호를 오염시킬 수 있음
   => 테스트 세트에서 네트워크를 더 잘 일반화 가능케 함
3. 종류
   a. Noise Mixing
   b. Reverberation
   c. Environmental Corruption Lobe (a + b)
4. $y[n] = x[n] * h[n] + n[n]$

   x[n] : 깨끗한 신호
   h[n] : 잔향 신호 (b.reverberation)
   n[n] : 잡음 (a.noise mixing)

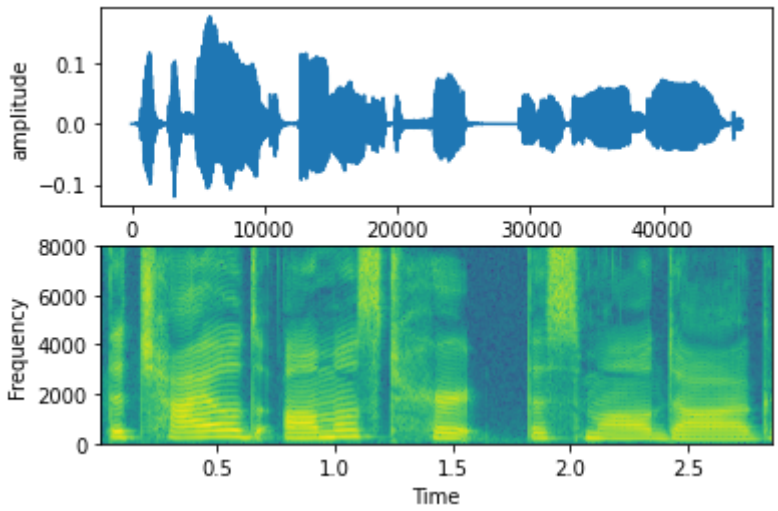
```
%%capture

#·speechbrain·:·pytorch 를 기반으로하는 종합적인 음성처리 툴킷
!pip install speechbrain

# noise mixing
!wget https://www.dropbox.com/s/vwv8xdr7l3b2tta/noise_sig.csv
!wget https://www.dropbox.com/s/aleer424jumcs08/noise2.wav
!wget https://www.dropbox.com/s/eoxxi2ezr8owk8a/noise3.wav
!wget https://www.dropbox.com/s/u8qyvuyie2op286/spk1_snt1.wav

# reverberation
!wget https://www.dropbox.com/s/pjnub2s5hql2vxs/rir1.wav
!wget https://www.dropbox.com/s/nyno6bqbmiy2rv8/rirs.csv
```

clean speech data 불러오기

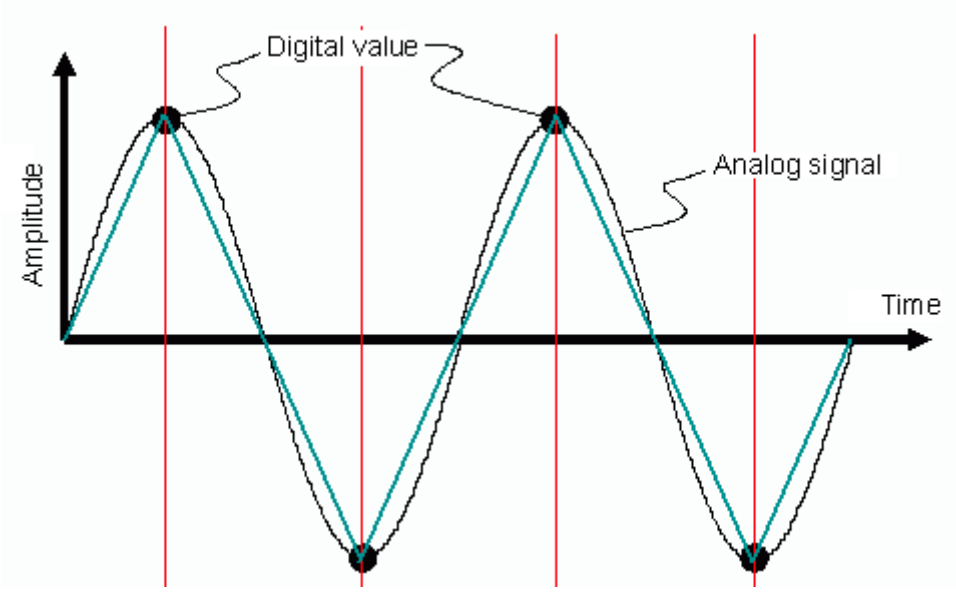
```
# 시각화 라이브러리
import matplotlib.pyplot as plt

# 데이터 로딩 및 데이터 전처리
# speechbrain 공식문서 : https://speechbrain.readthedocs.io/en/latest/index.html
from speechbrain.dataio.dataio import read_audio

# python 에 대한 대화형 인터페이스 제공
# https://ipython.org/ipython-doc/3/api/generated/IPython.display.html?highlight=audio#IPython.d
from IPython.display import Audio

```

```
# https://speechbrain.readthedocs.io/en/latest/API/speechbrain.dataio.dataio.html#speechbrain.da
clean = read_audio('spk1_snt1.wav').squeeze()
# print(clean)

# Plots
# https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot.html?highlight=subplot#mat
# subplot(nrows, ncols, plot_number) => nrows=2 ncols=1 plot_number=1
# => 2행 x 1열의 개념적 그리드에서 상단 플롯(즉, 첫 번째)을 나타내는 그림의 하위 축을 생성
plt.subplot(211)
plt.plot(clean)
plt.xlabel('Time')
plt.ylabel('amplitude')

plt.subplot(212)
# https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.specgram.html?highlight=specgram#m
plt.specgram(clean,Fs=16000)
# => clean data 의 spectrogram plotting
plt.xlabel('Time')
plt.ylabel('Frequency')

Audio(clean, rate=16000)
# rate : The sampling rate of the raw data
# sample rate : 소리 (아날로그 신호) -> 숫자 (디지털 신호)
# 16khz : 1초당 16000 번의 samples 를 기록

# https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.specgram.html
```

0:02 / 0:02



- 파형 : 아날로그 신호
- 샘플링한 점 부분 : 디지털신호



1. Additive Noise

1. (speechbrain.processing.speech_augmentation.AddNoise)로 오염시킬 수 있는 클래스를 설계함
2. 이 클래스로 노이즈 신호 목록을 항목화하는 csv 파일을 입력받음
3. 호출되면 AddNoise는 이 노이즈 컬렉션에서 샘플링함
4. 선택한 노이즈를 임의의 SNR(Signal-to-Nose Ratio)로 깨끗한 신호에 추가함

```
1 import pandas as pd
2
3 df = pd.read_csv('noise_sig.csv')
4 df
```

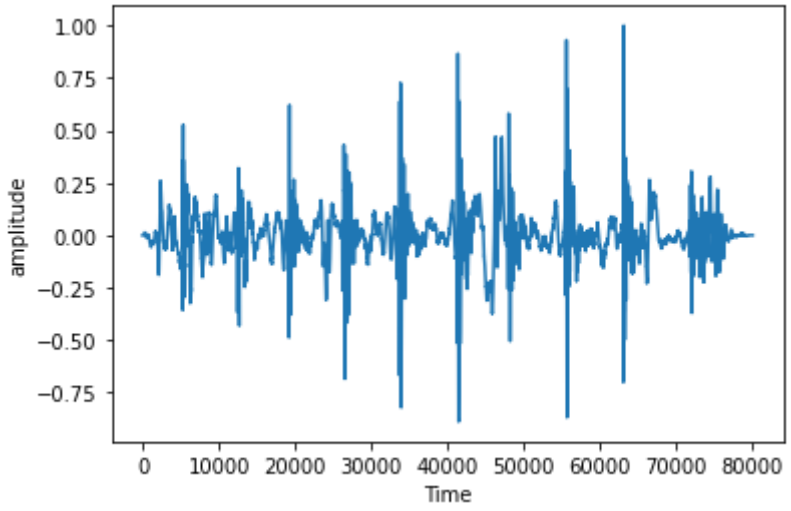
	ID	duration	wav	wav_format	wav_opts
0	noise2	5.0	noise2.wav	wav	NaN
1	noise3	1.0	noise3.wav	wav	NaN

```
1 sound_file ='noise2.wav'
2 Audio(sound_file)
```

0:05 / 0:05

```
1 sound_file2_1 = read_audio('noise2.wav').squeeze()
2 plt.plot(sound_file2_1)
3 plt.xlabel('Time')
4 plt.ylabel('amplitude')
```

```
Text(0, 0.5, 'amplitude')
```

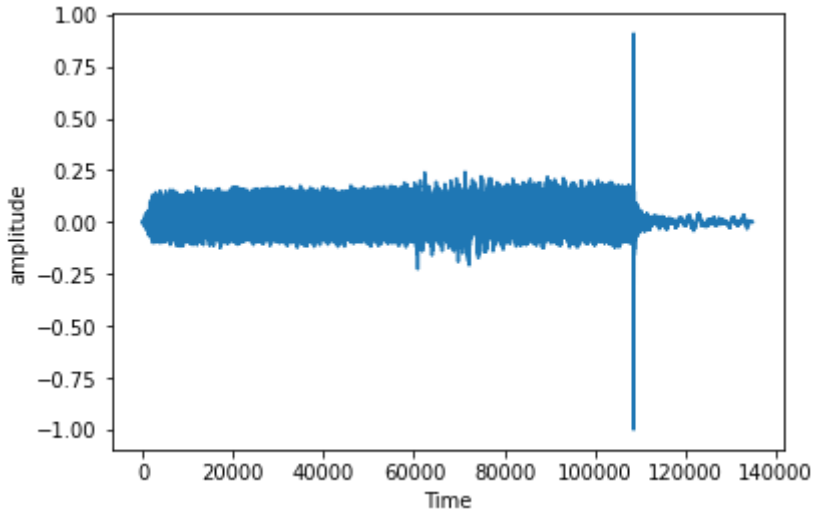



```
1 sound_file3 ='noise3.wav'
2 Audio(sound_file3)
```

0:08 / 0:08

```
1 sound_file3_1 = read_audio('noise3.wav').squeeze()
2 plt.plot(sound_file3_1)
3 plt.xlabel('Time')
4 plt.ylabel('amplitude')
```

```
Text(0, 0.5, 'amplitude')
```

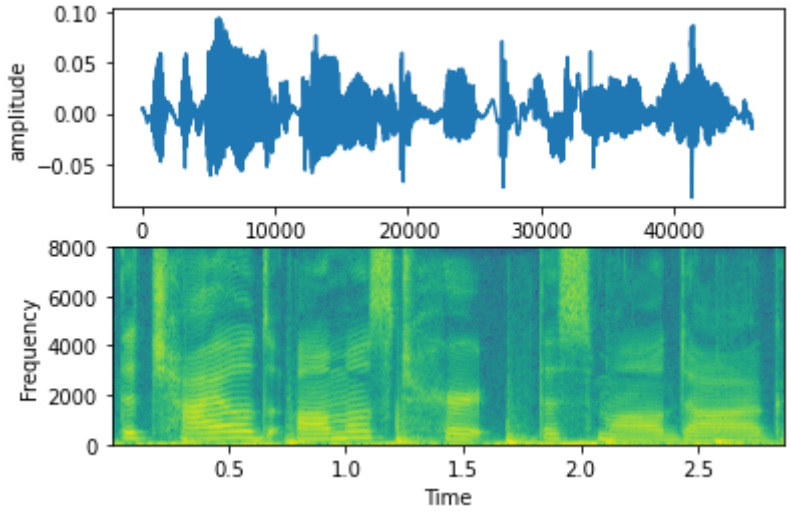



```
1 import torch
2
3 # 음성신호를 노이즈로 오염시키기 위해 필요한 모듈 import
4 # https://speechbrain.readthedocs.io/en/latest/API/speechbrain.processing.speech_augmentation.ht
5 from speechbrain.processing.speech_augmentation import AddNoise
6
7 # 1.(speechbrain.processing.speech_augmentation.AddNoise)로 오염시킬 수 있는 클래스생성
8 # 2. 이 클래스는 노이즈 신호 csv 파일을 입력받음.
9 # 3. 호출되면 AddNoise는 이 노이즈 컬렉션에서 샘플링함
```

```
noisifier = AddNoise('noise_sig.csv', snr_low=-10, snr_high=10)
lengths = torch.ones(1) # 단일 차원 배치 모양 텐서
# 4. 선택한 노이즈를 임의의 SNR(Signal-to-Nose Ratio)로 깨끗한 신호에 추가.
noisy = noisifier(clean.unsqueeze(0), lengths)
print(noisy)


# snr_low  : (int)혼합 비율의 데시벨 하한값
# snr_high  : (int)혼합 비율의 데시벨 상한값
# 데시벨(deci bell) : 소리의 측정단위 ; int -> 10^int 배
# snr_high, snr_low 를 이용해서 노이즈의 양을 조절 가능!

# Plots
# 시간 - 진폭 파형으로 표현
plt.subplot(211)
plt.plot(noisy.squeeze())
plt.xlabel('Time')
plt.ylabel('amplitude')

# 시간 - 주파수 스펙토그램으로 표현
plt.subplot(212)
plt.specgram(noisy.squeeze(),Fs=16000)
plt.xlabel('Time')
plt.ylabel('Frequency')

Audio(noisy.squeeze(0), rate=16000)
```

```
tensor([[ 0.0050,  0.0050,  0.0049,  ..., -0.0141, -0.0140, -0.0142]])
```

0:02 / 0:02

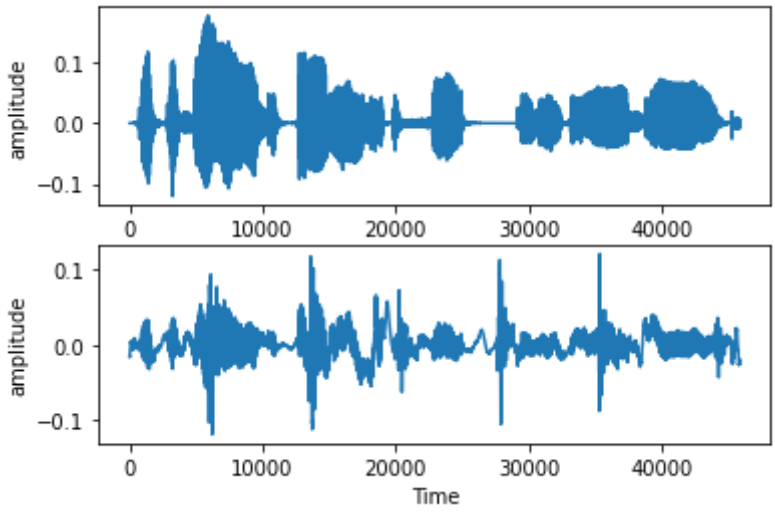


clean data vs noise를 첨가한 data

```
# clean data 파형
plt.subplot(211)
plt.plot(clean.squeeze())
plt.ylabel('amplitude')

# noise data 파형
```

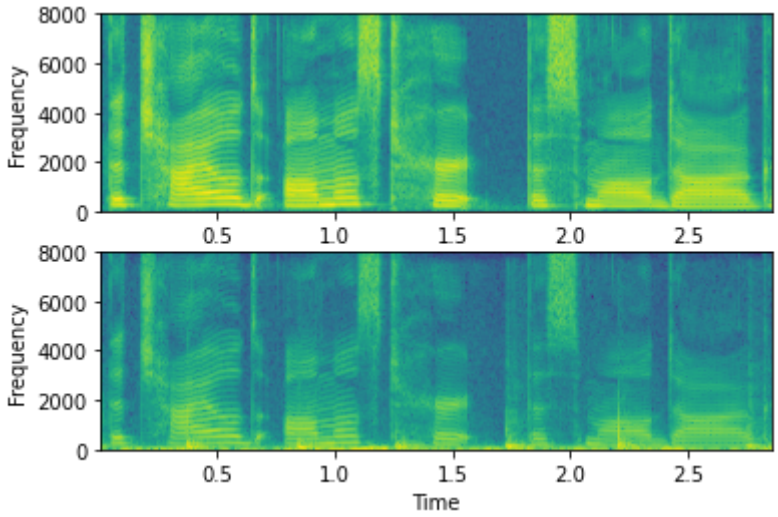
```
plt.subplot(212)
plt.plot(noisy.squeeze())
plt.xlabel('Time')
plt.ylabel('amplitude')
```

```
Text(0, 0.5, 'amplitude')
```




```
# clean data spectogram
plt.subplot(211)
plt.specgram(clean,Fs=16000)
plt.ylabel('Frequency')

# noise data 스펙토그램으로 표현
plt.subplot(212)
plt.specgram(noisy.squeeze(),Fs=16000)
plt.xlabel('Time')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```




```
Audio(clean, rate=16000)
```

0:02 / 0:02

```
1 Audio(noisy.squeeze(0), rate=16000)
```

0:02 / 0:02

1. SNR의 샘플링 범위를 정의하는 snr_low 및 snr_high 매개변수로 노이즈의 양 조정가능
2. 길이 벡터는 길이가 다른 신호의 병렬 배치를 처리할 수 있기 때문에 필요함. 길이 벡터는 배치를 구성하는 각 문장에 대한 상대 길이를 포함. (예: 두 가지 예의 경우 lenght=[0.8 1.0]을 가질 수 있음. 여기서 1.0은 배치에서 가장 긴 문장의 길이)

2. Reverberation

=> 다중 경로 전파 : **잔향** (메아리) 잔향은 시간 및 주파수 영역에서 신호를 "부드럽게" 만드는 복잡한 잡음.

방에 말할 때 우리의 음성은 벽, 바닥, 천장 및 음향 환경 내의 물체에 의해 **여러 번 반사**됨.
결과적으로 원거리 마이크에 의해 녹음된 최종 신호에는 **원래 신호의 지연된 복제본이 여러 개 포함** 됨.
이러한 모든 복제본은 **서로 간섭하고 음성 신호의 명료도에 상당한 영향을 미침**.

주어진 방 내에서 소스와 수신기 사이의 잔향은 임펄스 응답으로 모델링 됨.

```
1 sound_file4 ='rir1.wav'
2 Audio(sound_file4)
```

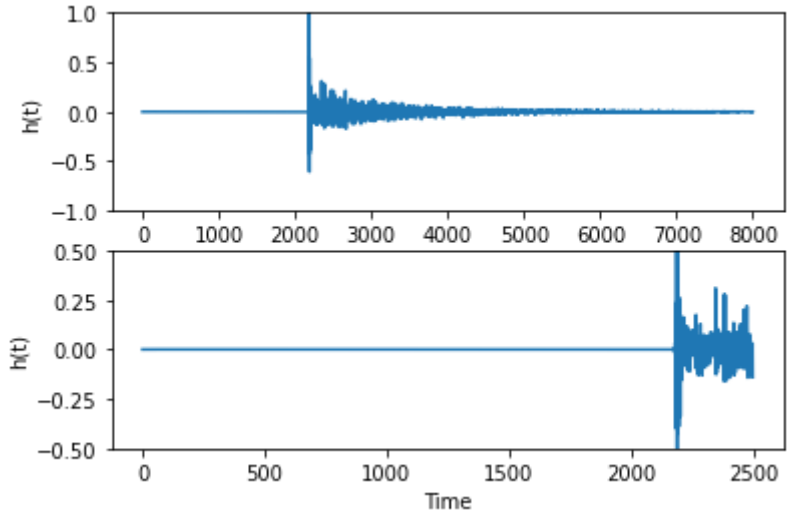
0:00 / 0:01

```
1 sound_file4_1 = read_audio('rir1.wav').squeeze()
2 plt.plot(sound_file4_1)
3 plt.xlabel('Time')
4 plt.ylabel('amplitude')
```

```
Text(0, 0.5, 'amplitude')
```


```
rir = read_audio('rir1.wav')

# Impulse response : h(t)
# t = 0 일때, 입력이 들어간 순간

plt.subplot(211)
plt.plot(rir[0:8000])
plt.xlabel('Time')
plt.ylabel('h(t)')
plt.ylim(-1,1)


# Zoom on early reflections
plt.subplot(212)
plt.plot(rir[0:2500])
plt.xlabel('Time')
plt.ylabel('h(t)')
plt.ylim(-0.5,0.5)
```

```
(-0.5, 0.5)
```



임펄스 응답은 소리가 소스에서 수신기로 이동할 때 겪는 변화를 보여줌.
임펄스 응답의 각 피크는 수신기에 도달하는 복제본에 해당.
첫 번째 피크는 직접 경로이며, 그런 다음 벽, 천장, 바닥에서 1차 반사를 볼 수 있음(두번째 그림 참조)

전반적으로 임펄스 응답은 기하급수적인 감쇠를 따름. 이 감쇠는 잔향 시간이 낮은 건조한 방에서 더 빠르고, 크고 비어 있는 환경에서는 더 느림.

잔향은 깨끗한 신호와 임펄스 응답 사이에 컨볼루션을 수행하여 추가됨. ->x[n]∗h[n]

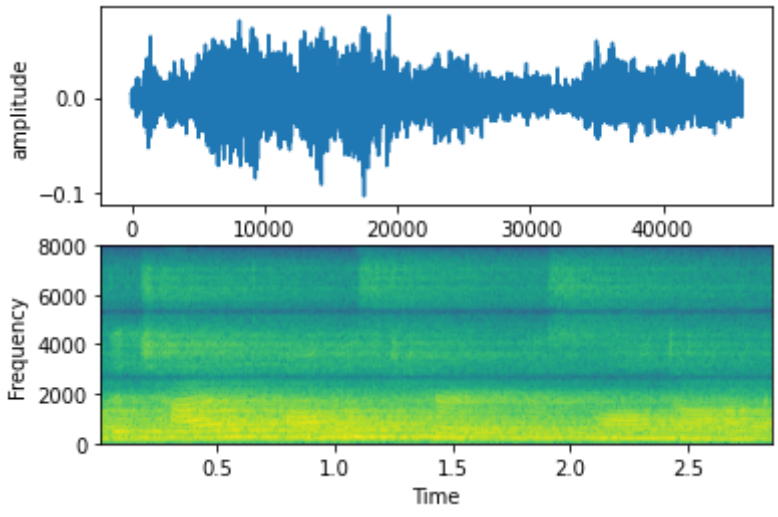
```
import pandas as pd

```

```
df2 = pd.read_csv('rirs.csv')
df2
```

	ID	duration	wav	wav_format	wav_opts
0	rir1	1.0	rir1.wav	wav	NaN

```
# SpeechBrain에서 이 작업은 speechbrain.processing.speech_augmentation.AddReverb에 의해 수행됨.
# https://speechbrain.readthedocs.io/en/0.5.7/API/speechbrain.processing.speech_augmentation.htm
from speechbrain.processing.speech_augmentation import AddReverb

# rir_scale_factor ( float ) - 주어진 임펄스 응답을 압축하거나 확장한다.
# 잔향의 양은 rir_scale_factor 매개변수에 의해 제어
# 0 < scale_factor < 1이면 임펄스 응답이 압축되고(더 적은 잔향), scale_factor > 1이면 확장됨(더
# 호출되면 AddRev는 주어진 csv 파일에서 임펄스 응답을 샘플링함.
reverb = AddReverb('rirs.csv', rir_scale_factor=6)
# rir_scale_factor 을 변화시키며 확인 가능
reverbed = reverb(clean, torch.ones(1))

# Plots
plt.subplot(211)
plt.plot(reverbed.squeeze())
plt.xlabel('Time')
plt.ylabel('amplitude')


plt.subplot(212)
plt.specgram(reverbed.squeeze(),Fs=16000)
plt.xlabel('Time')
plt.ylabel('Frequency')

Audio(reverbed.squeeze(0), rate=16000)

```

0:02 / 0:02




```
Audio(clean, rate=16000)
```

0:02 / 0:02

3. Environmental Corruption Lobe (1 + 2)

소음(noise) + 잔향(reverberation) 특정 확률로 결합되어 활성화.

클래스 내에서 손상 작업 순서 = 먼저 잔향을 도입 한 후, 노이즈가 추가.
y[n]=x[n]∗h[n]+n[n]
이 클래스는 임펄스 응답 및 open-rir이라는 노이즈 시퀀스의 오픈 소스 데이터 세트를 자동으로 다운로드하고 샘플링을 수행.

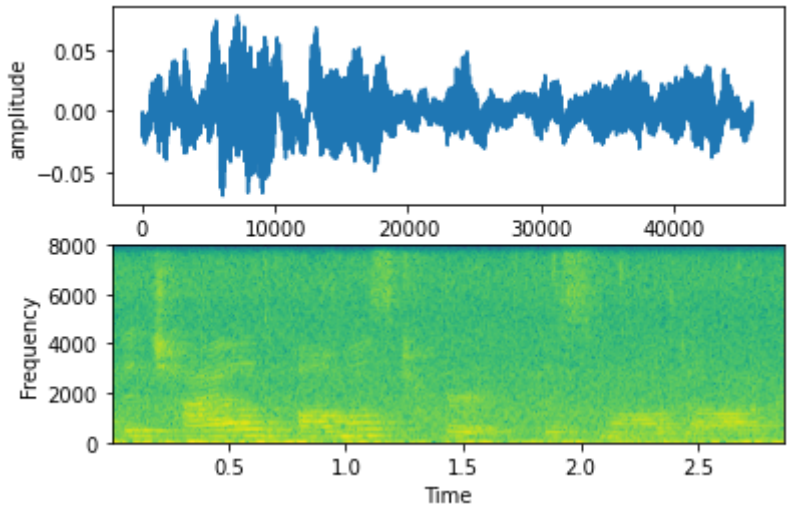
```
# SpeechBrain에는 Speechbrain.lobes.augment.EnvCorrupt에 TimeDomainSpecAugment라는 클래스이용.
# https://speechbrain.readthedocs.io/en/0.5.7/API/speechbrain.lobes.augment.html#speechbrain.lob
from speechbrain.lobes.augment import EnvCorrupt

# 이 클래스는 임펄스 응답 및 open-rir이라는 노이즈 시퀀스의 오픈 소스 데이터 세트를 자동으로 다운
# open rir 다운로드 및 백업에 몇 분 정도 소요됨
corrupter = EnvCorrupt(openrir_folder='.')
noise_rev = corrupter(clean.unsqueeze(0), torch.ones(1))
#=> 왜곡된 파형을 반환

# Plots
plt.subplot(211)
plt.plot(noise_rev.squeeze())
plt.xlabel('Time')
plt.ylabel('amplitude')

plt.subplot(212)
plt.specgram(noise_rev.squeeze(),Fs=16000)
plt.xlabel('Time')
plt.ylabel('Frequency')

Audio(noise_rev.squeeze(0), rate=16000)
```

```
./rirs_noises.zip exists. Skipping download
```

0:02 / 0:02

```
# 함수를 다시 호출하여 신호가 다른 방식으로 오염되게 함
## 즉석 오염을 구현하고, 서로 다른 입력에 서로 다른 데이터를 생성해 낼 수 있음
noise_rev = corrupter(clean.unsqueeze(0), torch.ones(1))

# Plots
plt.subplot(211)
plt.plot(noise_rev.squeeze())
plt.xlabel('Time')
plt.ylabel('amplitude')

plt.subplot(212)
plt.specgram(noise_rev.squeeze(),Fs=16000)
plt.xlabel('Time')
plt.ylabel('Frequency')

Audio(noise_rev.squeeze(0), rate=16000)
```

0:02 / 0:02



이 외에도 음성 증강을 수행하는 데 사용할 수 있는 다른 유형의 왜곡이 존재함 (예: 속도 변경, 시간 드롭아웃, 주파수 드롭아웃, 클리핑)
=> speech augmentation tutorial 에 설명되어있음

References

[1] M. Ravanelli, P. Svaizer, M. Omologo, "Realistic Multi-Microphone Data Simulation for Distant Speech Recognition", in Proceedings of Interspeech 2016 ArXiv

[2] M. Ravanelli, M. Omologo, "Contaminated speech training methods for robust DNN-HMM distant speech recognition", in Proceedings of INTERSPEECH 2015. ArXiv

[3] M. Ravanelli, M. Omologo, "On the selection of the impulse responses for distant-speech recognition based on contaminated speech training", in Proceedings of INTERSPEECH 2014. ArXiv

[4] M. Ravanelli, A. Sosi, P. Svaizer, M.Omologo, "Impulse response estimation for robust speech recognition in a reverberant environment", in Proceeding of the European Signal Processing Conference, EUSIPCO 2012. ArXiv

About SpeechBrain

- Website: https://speechbrain.github.io/
- Code: https://github.com/speechbrain/speechbrain/
- HuggingFace: https://huggingface.co/speechbrain/

Citing SpeechBrain

Please, cite SpeechBrain if you use it for your research or business.

```
@misc{speechbrain,
  title={{SpeechBrain}: A General-Purpose Speech Toolkit},
  author={Mirco Ravanelli and Titouan Parcollet and Peter Plantinga and Aku Rouhe and Samuele Cor
  year={2021},
  eprint={2106.04624},
  archivePrefix={arXiv},
  primaryClass={eess.AS},
  note={arXiv:2106.04624}
}
```

speech augmentation : **원래 음성 신호를 인위적으로 손상**시켜 네트워크에 새로운 신호를 처리하고 있다는 환상을 제공.
장점 : 신경망이 일반화를 개선하여 테스트 데이터에서 더 나은 성능을 달성하는 데 도움이 되는 강력한 정규화 장치의 역할을 함.

SpeechBrain 이 지원하는 기능들

1. speed perturbation (속도섭동)
2. time dropout
3. frequency dropout
4. clipping
5. augmentation lobe (1 + 2 + 3)

```
# 증강 기술은 speechbrain.processing.speech_augmentation에서 구현됨

# speechbrain 설치
%%capture
!pip install speechbrain
```

```
# 음성신호 다운로드

%%capture
!wget https://www.dropbox.com/s/u8qyvuyie2op286/spk1_snt1.wav
```

1. Speed Perturbation

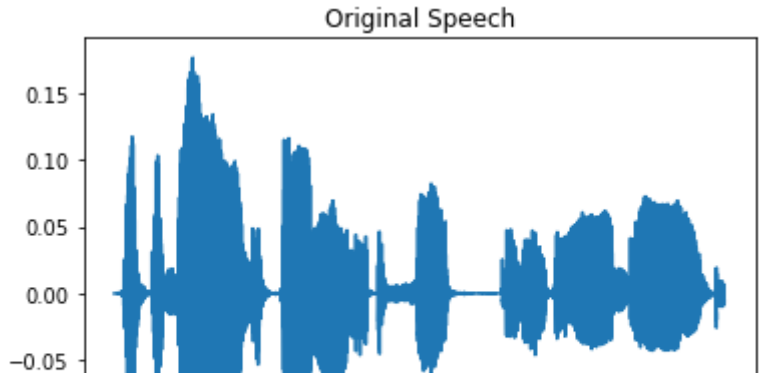
저장이 완료되었습니다.

호를 원래 신호와 약간 다른 샘플링 속도로 다시 샘플링함.

원래 신호보다 약간 "빠르거나" "느리게" 들리는 음성 신호.

말하기 속도뿐만 아니라 음높이 및 포먼트와 같은 스피커 특성에도 영향을 줌.

```
# 원본 데이터 불러오기
import matplotlib.pyplot as plt
from speechbrain.dataio.dataio import read_audio

signal = read_audio('spk1_snt1.wav')

plt.figure(1)
plt.title("Original Speech")
plt.plot(signal)
plt.show()
```




```
# original speech

from IPython.display import Audio
Audio('spk1_snt1.wav')
```

0:02 / 0:02

```
# 속도 섭동 초기화

# https://speechbrain.readthedocs.io/en/0.5.7/API/speechbrain.processing.speech_augmentation.htm
from speechbrain.processing.speech_augmentation import SpeedPerturb

# 속도 섭동 인스턴스를 생성
perturbator = SpeedPerturb(orig_freq=16000, speeds=[120], perturb_prob=1.0)

# orig_freq : 원래 신호의 주파수
# speeds : 속도
# perturb_prob : 배치가 속도 교란될 가능성
```

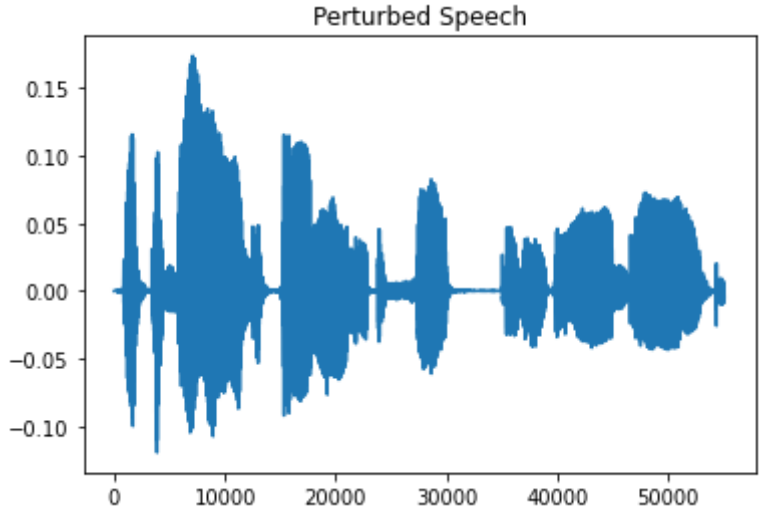
저장이 완료되었습니다.

- orig_freq: 원래 신호의 샘플링 주파수
- speegs: 원래 신호의 백분율로 신호를 변경해야 하는 모든 속도를 포함하는 목록 (즉, 속도=[100]은 원래 신호를 변경하지 않음).
  더 많은 값(예: 속도=[90, 100, 110, 120])을 추가하면 속도는 지정된 값 중에서 무작위로 선택됨.)
- perturb_prob: 배치가 교란될 가능성

```
clean = signal.unsqueeze(0) # [batch, time]
perturbed = perturbator(clean)
print('clean : ', clean)
print('perturbed : ', perturbed)

plt.figure(1)
plt.title("Perturbed Speech")
plt.plot(perturbed.squeeze())
plt.show()

```

```
print('clean.shape : ',clean.shape)
print('perturbed.shape : ',perturbed.shape)

Audio(perturbed,rate=16000)

```

```
clean :  tensor([[-9.1553e-05, -9.1553e-05, -9.1553e-05,  ..., -5.3711e-03,
         -5.1880e-03, -5.3711e-03]])
perturbed :  tensor([[-9.0999e-05, -9.5231e-05, -9.1362e-05,  ..., -4.9132e-03,
         -5.7586e-03, -4.6385e-03]])
```




```
clean.shape :  torch.Size([1, 45920])
perturbed.shape :  torch.Size([1, 55104])
```

0:03 / 0:03

```
Audio('spk1_snt1.wav')
```

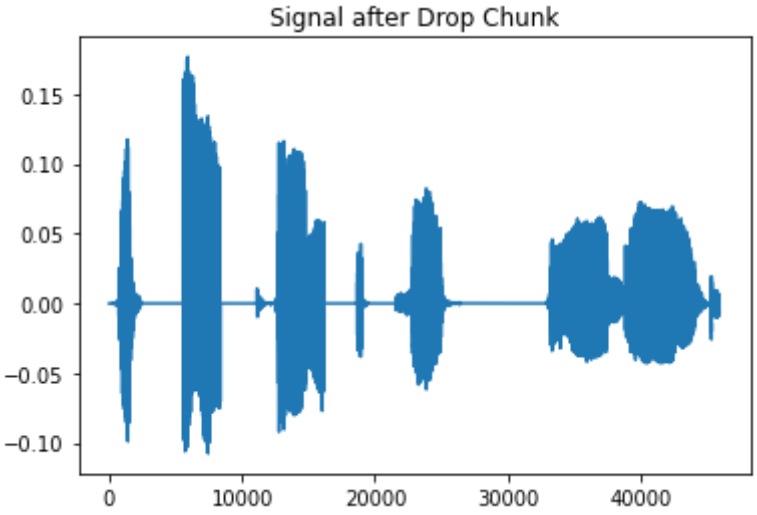
저장이 완료되었습니다.

2. Time dropout

time dropout은 원본 파형의 일부 임의 길이(간격)를 0으로 바꾼 것 (입력 신호의 일부를 삭제 하는 것)
신경망이 신호의 일부가 누락된 경우(time 채널 누락된 경우)에도 우수한 성능을 제공할 수 있어야 한다는 이론에서 나옴

```
import torch
# https://speechbrain.readthedocs.io/en/0.5.7/API/speechbrain.processing.speech_augmentation.htm
from speechbrain.processing.speech_augmentation import DropChunk

dropper = DropChunk(drop_length_low=2000, drop_length_high=3000, drop_count_low=5, drop_count_hi
# drop_length_low 및 drop_length_high , 0의 임의 청크의 최대 및 최소 길이를 결정
# drop_count_low 및 drop_count_high, 원래 신호에 추가할 임의의 청크 수에 영향을 줌
```

```
 8
 9 # 길이 벡터는 길이가 다른 신호의 병렬 배치를 처리가능.
10 # 길이 벡터는 배치를 구성하는 각 문장에 대한 상대 길이를 포함함
11 # (예: 두 배치의 경우 lenght=[0.8 1.0]을 가질 수 있음, 1.0은 배치에서 가장 긴 문장의 길이).
12 # 이 경우 단일 문장으로 구성된 배치가 있으므로 상대 길이는 length=[1.0].
13 length = torch.ones(1)
14 dropped_signal = dropper(clean, length)
15
16 plt.figure(1)
17 plt.title("Signal after Drop Chunk")
18 plt.plot(dropped_signal.squeeze())
19 plt.show()
20
21 Audio(dropped_signal,rate=16000)
22
```



0:02 / 0:02

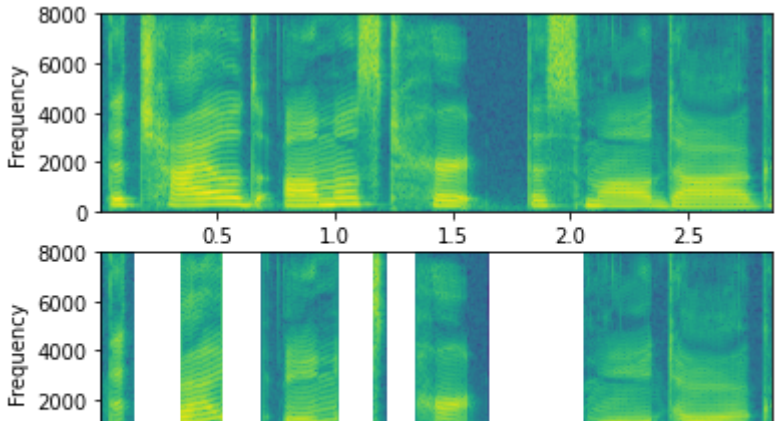
저장이 완료되었습니다.

```
1 Audio( spk1_snt1.wav )
```

0:02 / 0:02

```
 1 # Let's plot the two spectrograms
 2 plt.subplot(211)
 3 plt.specgram(clean.squeeze(),Fs=16000)
 4 plt.xlabel('Time')
 5 plt.ylabel('Frequency')
 6
 7 plt.subplot(212)
 8 plt.specgram(dropped_signal.squeeze(),Fs=16000)
 9 plt.xlabel('Time')
10 plt.ylabel('Frequency')
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/axes/_axes.py:7592: RuntimeWarning: divide
  Z = 10. * np.log10(spec)
Text(0, 0.5, 'Frequency')
```



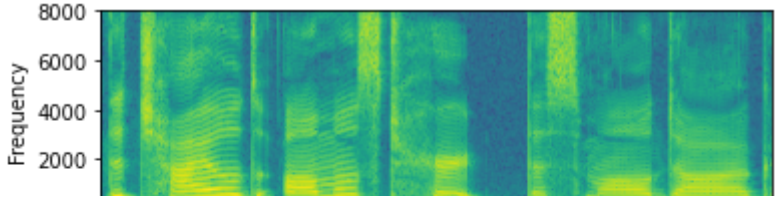
3. Frequency Dropout

주파수 드롭아웃 시간 영역에서 0을 추가하는 대신 Drop Freq는 주파수 영역에서 0을 추가.
이것은 무작위로 선택된 대역 정지 필터로 원래 신호를 필터링하여 달성가능함.
time dropout과 유사하게 일부 주파수 채널이 누락된 경우에도 신경망이 잘 작동해야 한다는 원리 이용.

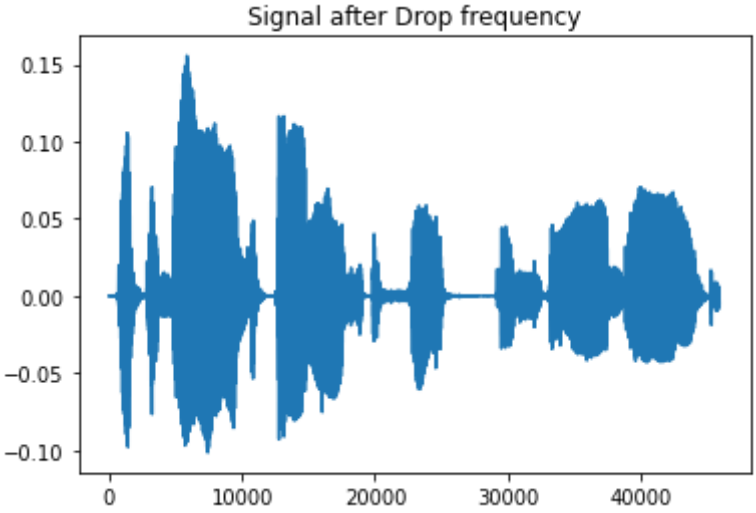
```
from speechbrain.processing.speech_augmentation import DropFreq

dropper2 = DropFreq(drop_count_low=5, drop_count_high=8)
dropped_signal_freq = dropper2(clean)

# 주파수 강하의 양 제어 매개변수
# drop_count_low / drop_count_high, 드롭할 주파수 대역 수에 영향을 줌
                                    롭될 수 있는 최소 및 최대 주파수에 해당
                                    해당

# Let's plot the two spectrograms
plt.subplot(211)
plt.specgram(clean.squeeze(),Fs=16000)
plt.xlabel('Time')
plt.ylabel('Frequency')

plt.subplot(212)
plt.specgram(dropped_signal_freq.squeeze(),Fs=16000)
plt.xlabel('Time')
plt.ylabel('Frequency')
```

저장이 완료되었습니다.

```
Text(0, 0.5, 'Frequency')
```




```
plt.figure(1)
plt.title("Signal after Drop frequency")
plt.plot(dropped_signal_freq.squeeze())
plt.show()

Audio(dropped_signal_freq,rate=16000)
```



0:02 / 0:02

저장이 완료되었습니다.

0:02 / 0:02

4. clipping

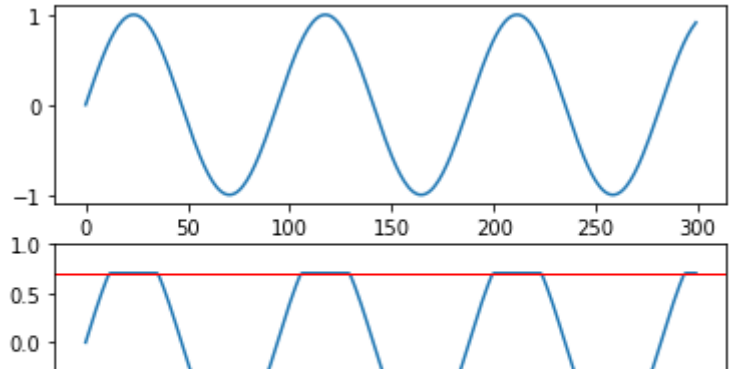
음성 신호에서 일부 정보를 제거하는 또 다른 방법은 클리핑을 추가하는 방법이 있음.
신호의 최대 절대 진폭을 고정하는 비선형 왜곡의 한 형태 -> 포화 효과 추가.

```
# 입력 텐서를 클램핑하여 오디오 클리핑을 모방하는 함수
# => 신호의 최대 절대 진폭을 고정!

import torch
# https://speechbrain.readthedocs.io/en/0.5.7/API/speechbrain.processing.speech_augmentation.htm
from speechbrain.processing.speech_augmentation import DoClip

# 클리핑 양은 신호가 클램핑되는 하한 및 상한 임계값을 설정하는 매개변수 clip_low 및 clip_high로
```

```
# clip_low ( float ) - 신호를 클리핑할 진폭의 하한.
# clip_high ( float ) - 신호를 클리핑할 진폭의 상한.
clipper = DoClip(clip_low=0.7, clip_high=0.7)

# https://pytorch.org/docs/stable/generated/torch.sin.html?highlight=sin#torch.sin
# https://pytorch.org/docs/stable/generated/torch.linspace.html?highlight=linspace
sinusoid = torch.sin(torch.linspace(0,20, 300))
clipped_signal = clipper(sinusoid.unsqueeze(0))

#  plots
plt.figure(1)
plt.subplot(211)
plt.plot(sinusoid)
plt.xlabel('Time')

plt.subplot(212)
plt.plot(clipped_signal.squeeze())
plt.xlabel('Time')
plt.ylim(-1,1)
# y = 0.7, -0.7 긋기
plt.axhline(y=0.7, color='r', linewidth=1)
plt.axhline(y=-0.7, color='r', linewidth=1)


# freq domain
plt.figure(2)
plt.subplot(211)
plt.specgram(sinusoid,Fs=16000)
plt.xlabel('Time')
plt.ylabel('Frequency')

plt.subplot(212)
                                                    ),Fs=16000)
plt.ylabel('Frequency')
```

저장이 완료되었습니다.

```
Text(0, 0.5, 'Frequency')
```



5. Augmentation Lobe

speed perturbation, time dropout, frequency dropout 은 종종 특정 확률로 결합되어 사용됨
SpeechBrain의 Speechbrain.lobes.augment에 TimeDomainSpecAugment 클래스 이용

```
 1 # speechbrain.lobes.augment에서 이러한 기술을 결합하고 무작위로 활성화하는 인터페이스를 제공
 2 # https://speechbrain.readthedocs.io/en/0.5.7/API/speechbrain.lobes.augment.html#speechbrain.lob
 3 from speechbrain.lobes.augment import TimeDomainSpecAugment
 4
 5 do_augment = TimeDomainSpecAugment(speeds=[80, 110, 120],
 6                                    perturb_prob=1.0,
 7                                    drop_freq_prob=1.0,
 8                                    drop_chunk_prob=1.0,
 9                                    drop_chunk_length_low=1000,
10                                    drop_chunk_length_high=3000)
11
12 length = torch.ones(1)
13 augmented_signal = do_augment(clean, length)
```

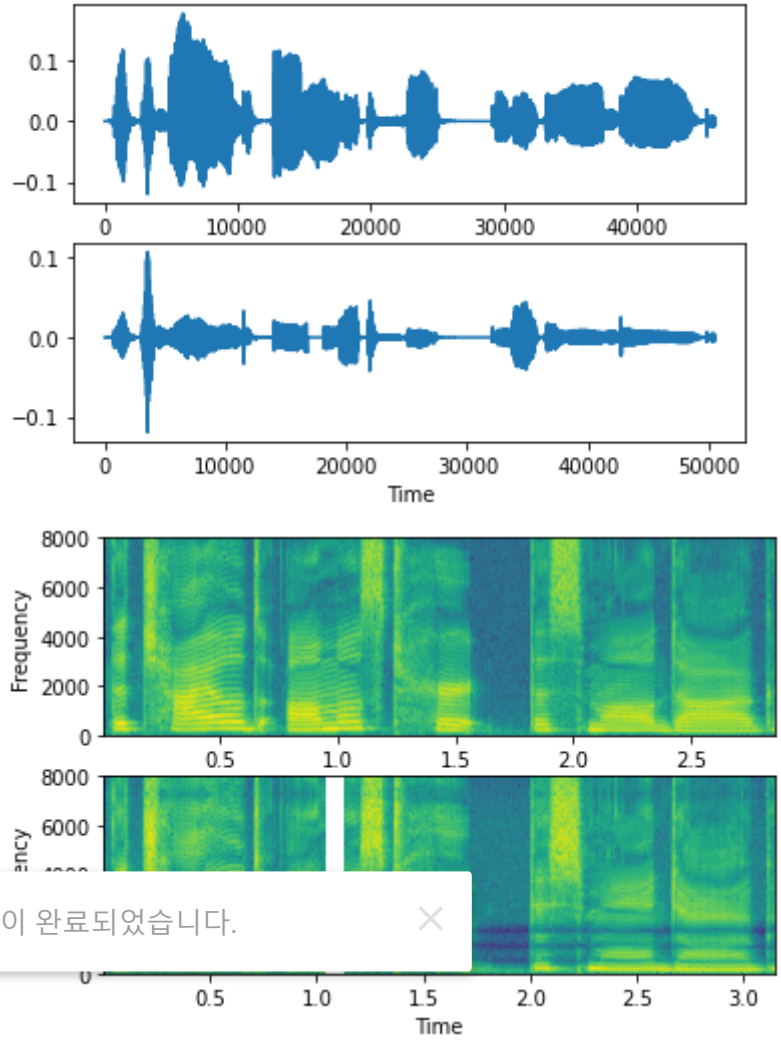
저장이 완료되었습니다.

```
17 plt.subplot(211)
18 plt.plot(clean.squeeze())
19 plt.xlabel('Time')
20
21 plt.subplot(212)
22 plt.plot(augmented_signal.squeeze())
23 plt.xlabel('Time')
24
25 # freq domain
26 plt.figure(2)
27 plt.subplot(211)
28 plt.specgram(clean.squeeze(),Fs=16000)
29 plt.xlabel('Time')
30 plt.ylabel('Frequency')
31
32 plt.subplot(212)
33 plt.specgram(augmented_signal.squeeze(),Fs=16000)
34 plt.xlabel('Time')
35 plt.ylabel('Frequency')
36
```

```
37 Audio(augmented_signal,rate=16000)
38
39
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/axes/_axes.py:7592: RuntimeWarning: divide
  Z = 10. * np.log10(spec)
```

0:03 / 0:03



저장이 완료되었습니다.

- TimeDomainSpecAugment 매개변수

매개변수:

- **perturb_prob** (*float from 0 to 1*) – 배치에 속도 섭동이 적용될 확률.
- **drop_freq_prob** (*float from 0 to 1*) – 배치에서 빈도가 떨어질 확률.
- **drop_chunk_prob** (*float from 0 to 1*) – 배치에서 청크가 삭제될 확률.
- **속도** (*int 목록*) – 각 배치를 교란하는 데 사용할 다양한 속도 집합입니다. 보다 `speechbrain.processing.speech_augmentation.SpeedPerturb`
- **sample_rate** (*int*) – 입력 파형의 샘플링 속도.
- **drop_freq_count_low** (*int*) – 삭제할 수 있는 가장 낮은 주파수 수입니다.
- **drop_freq_count_high** (*int*) – 삭제할 수 있는 최대 주파수 수.
- **drop_chunk_count_low** (*int*) – 삭제할 수 있는 최소 청크 수입니다.
- **drop_chunk_count_high** (*int*) – 삭제할 수 있는 최대 청크 수입니다.
- **drop_chunk_length_low** (*int*) – 삭제할 수 있는 최소 청크 길이입니다.
- **drop_chunk_length_high** (*int*) – 삭제할 수 있는 최대 청크 길이입니다.
- **drop_chunk_noise_factor** (*float*) – 발언의 평균 진폭을 기준으로 삽입된 백색 잡음의 크기를 조정하는 데 사용되는 잡음 요인입니다. 기본값 0(노이즈가 삽입되지 않음).

SpeechBrain에서는 종종 즉석 speech augmentation을 수행함
실제로, 각 epoch에서 생성된 모든 훈련 배치를 다른 방식으로 증가시킴.
장점 : speech augmentation은 GPU에서 매우 빠르게 구현할 수 있음
모든 훈련 배치에 대해 즉석에서 수행해도 훈련 루프 속도가 크게 느려지지는 않음.

References

저장이 완료되었습니다.

ng, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, Quoc V. Le, *SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition*, Proc. Interspeech 2019, ArXiv

[2] Mirco Ravanelli, Jianyuan Zhong, Santiago Pascual, Pawel Swietojanski, Joao Monteiro, Jan Trmal, Yoshua Bengio: *Multi-Task Self-Supervised Learning for Robust Speech Recognition*. Proc. of ICASSP 2020 ArXiv

About SpeechBrain

- Website: https://speechbrain.github.io/
- Code: https://github.com/speechbrain/speechbrain/
- HuggingFace: https://huggingface.co/speechbrain/

Citing SpeechBrain

Please, cite SpeechBrain if you use it for your research or business.

```
@misc{speechbrain,
  title={{SpeechBrain}: A General-Purpose Speech Toolkit},
  author={Mirco Ravanelli and Titouan Parcollet and Peter Plantinga and Aku Rouhe and Samuele Cor
  year={2021},
  eprint={2106.04624},
  archivePrefix={arXiv},
  primaryClass={eess.AS},
  note={arXiv:2106.04624}
}
```

저장이 완료되었습니다.

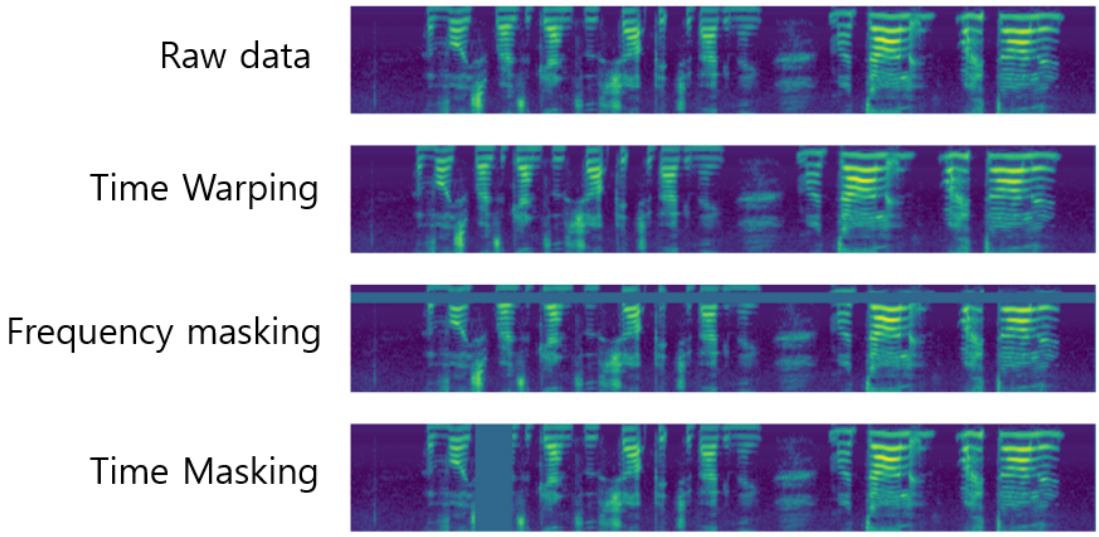
✓ 2초 오후 5:32에 완료됨

SpecAugment 관련 논문

(자동 음성 인식을 위한 간단한 데이터 증강 방법)

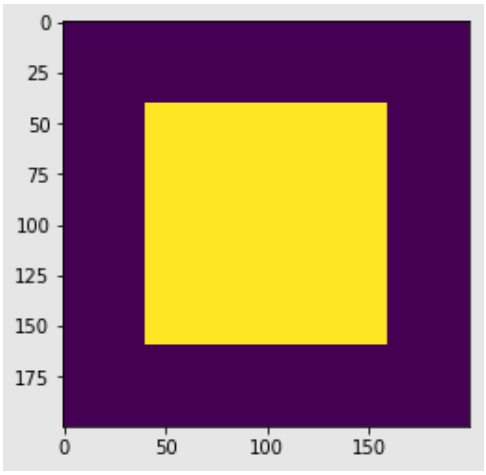
- input data : log mel spectogram (ex. LibriSpeech, SwitchBoard)
- data augmentation : 1. time warping, 2. frequency masking 3. time masking
- network : listen, attend and spell (LAS Network Architectures) , learning rate 를 조정하고, 모델 성능을 높이기 위해 음성인식 모델과 language model 도 함께 사용

Time warping의 경우 왼쪽 부분은 줄어들고 오른쪽 부분은 늘어난걸 볼 수 있습니다. Frequency maksing과 Time masking의 경우 데이터에 초록색 줄이 나타는 것을 볼 수 있습니다.

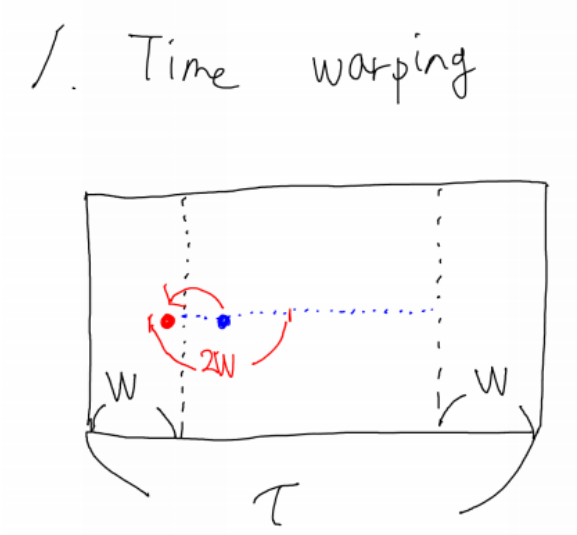


1. time warping

기본적으로 tensorflow에 있는 sparse image warp 함수를 사용합니다. sparse image warp 함수는 하나의 point를 다른 point로 옮기는 함수입니다. 아래 그림에서 사각형의 왼쪽 중심을 오른쪽으로 조금 움직인 예시입니다. (파란점 → 빨간점)



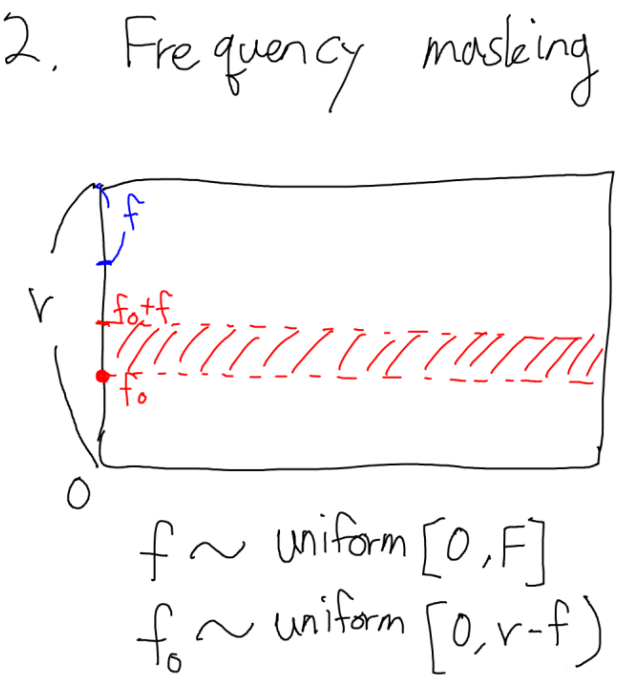
Time warping에서는 하이퍼 파라미터 W가 주어 집니다. 전체 이미지에서 양쪽으로 W만큼 줄인 후 가로 중앙선(파란선)에서 랜덤하게 하나의 점(파란점)을 고릅니다. 파란점이 출발 point가 됩니다. 이후 파란점을 기준으로 2W 범위 내에서 랜덤하게 하나의 점(빨간점)을 고릅니다. 빨간점이 도착 point가 됩니다. 이를 sparse image warp 함수에 파라미터로 사용합니다.



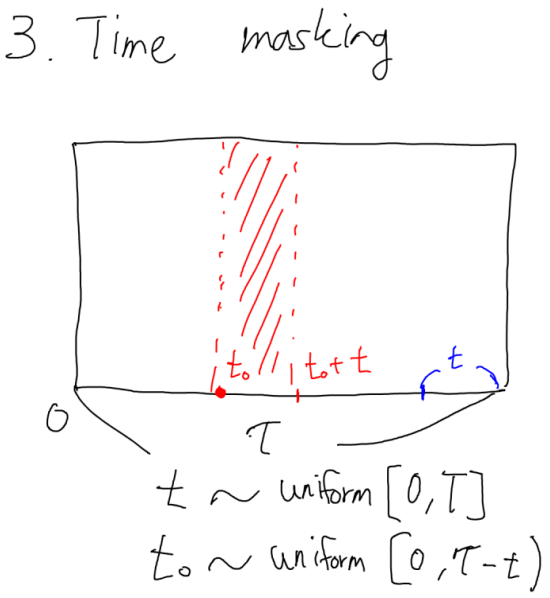
2. masking

masking의 경우 Frequency, Time 모두 방법은 동일합니다. Frequency로 예를 들면, 먼저 하이퍼 파라미터 F가 주어집니다. 이후 [0,F] 범위에서 랜덤하게 f 값을 뽑습니다. 이 값이 masking할 양이 됩니다. 이후 [0, r-f)에서 랜덤하게 f0 값을 뽑고, [f0, f0+f) 만큼 0으로 masking 합니다. Time masking의 경우도 Frequency와 동일합니다.

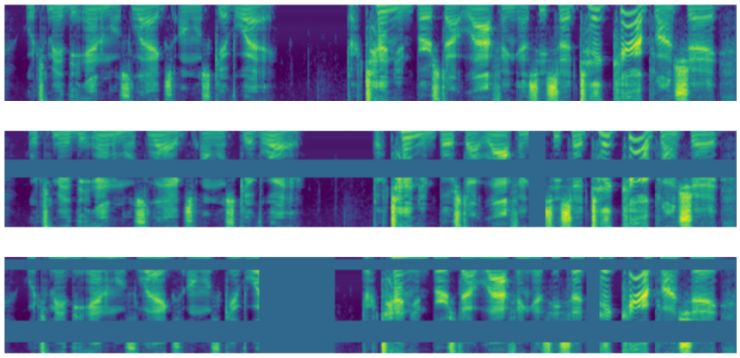
2-1. frequency masking : spectogram 에서 어느 주파수 영역에 대한 모든 시간을 0으로 처리



2-2. time masking : spectogram 에서 어느 시간대 영역에 대한 모든 주파수를 0으로 처리



데이터에 따라 다른 하이퍼 파라미터를 사용합니다. LibriSpeech basic (LB), LibriSpeech double (LD)



출처 - https://arxiv.org/abs/1904.08779 LB, LD 예시

Policy	W	F	m_F	T	p	m_T
None	0	0	-	0	-	-
LB	80	27	1	100	1.0	1
LD	80	27	2	100	1.0	2
SM	40	15	2	70	0.2	2
SS	40	27	2	70	0.2	2

왼쪽 표에서 mF와 mT는 masking의 개수를 뜻하고, p는 time masking이 2개 이상일 때 겹칠 수 있는 최대 길이 입니다.

Model

LAS Network Architectures

SpecAugment로 data augmentation을 한 후, Listen, Attend and Spell(LAS) 네트워크를 사용합니다

- https://arxiv.org/pdf/1508.01211.pdf

Learning Rate Schedules

수월한 학습을 위해 learning rate를 조정합니다. ramp-up, noise, exponentially decay 세 단계를 거칩니다. Sr까지 ramp-up을 하고, Snoise에서 weight에 noise를 주고, [Si, Sf] 구간에서 exponentially decay를 합니다.

$$\text{B(asic):}\ (s_r, s_{\text{noise}}, s_i, s_f) = (0.5\text{k}, 10\text{k}, 20\text{k}, 80\text{k})$$
$$\text{D(ouble):}\ (s_r, s_{\text{noise}}, s_i, s_f) = (1\text{k}, 20\text{k}, 40\text{k}, 160\text{k})$$

$$\text{L(ong):}\ (s_r, s_{\text{noise}}, s_i, s_f) = (1\text{k}, 20\text{k}, 140\text{k}, 320\text{k})$$

Shallow Fusion with Language Models

성능을 높이기 위해 음성인식 뿐 아니라 Language Model도 함께 사용합니다. 아래와 같이 LAS 네트워크 output과 Language Model의 output을 더하여 사용합니다. 논문에서 람다 값은 0.35로 설정합니다.

$$\mathbf{y}^* = \underset{\mathbf{y}}{\text{argmax}} \left(\log P(\mathbf{y}|\mathbf{x}) + \lambda \log P_{LM}(\mathbf{y}) \right) , \qquad (1)$$