

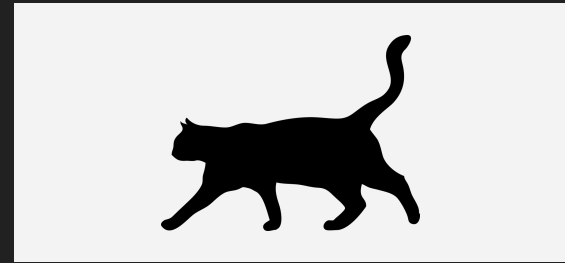
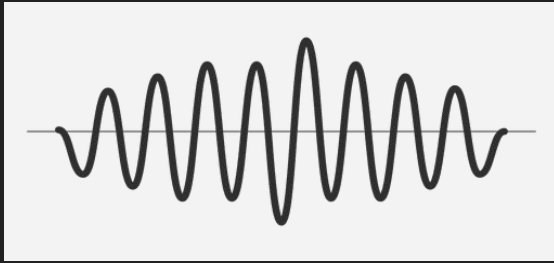
딥러닝으로 오디오 만나보기 (실전편)

Junwon Hwang

nuclear.github.io
nuclear1221@gmail.com

들어가며





시간

?



공간

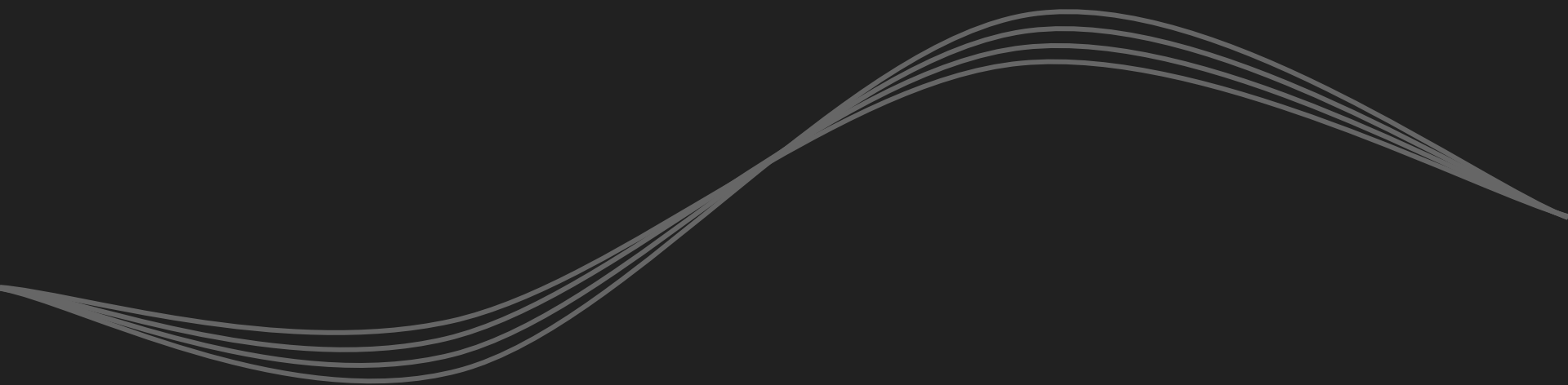


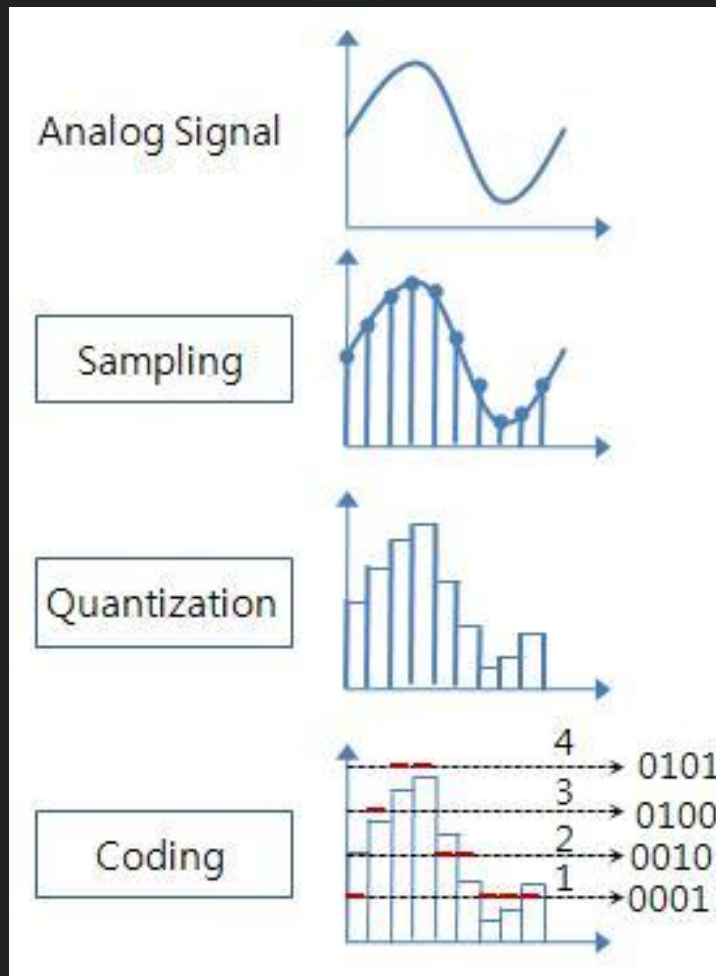
Audio vs Image
오디오와 이미지는 많이
다르다



익숙하지 않다 보니 시작이 어렵다
길라잡이가 있으면 좋겠다..!

오디오를 만져보자





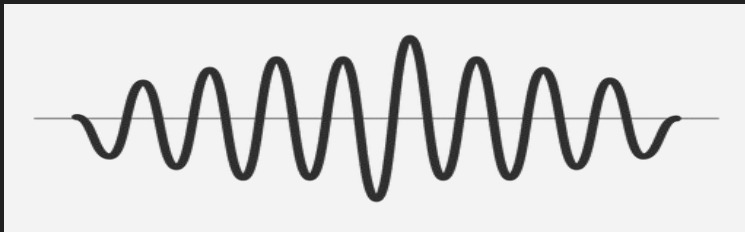
아날로그 신호 → 디지털 신호
“**Sampling**” 한다

Sampling Rate

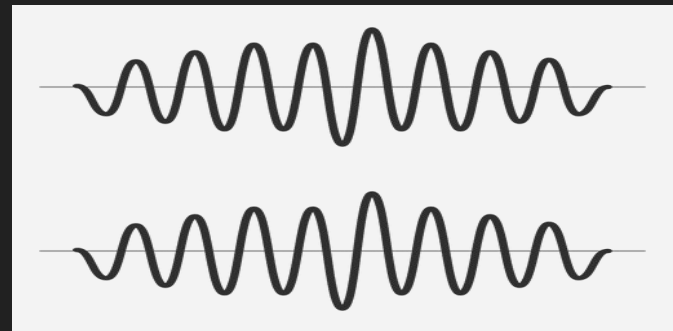
1초에 몇 개의 sample을 뽑아내는가?

44100Hz

사람의 청각 범위까지 잘 복원된다

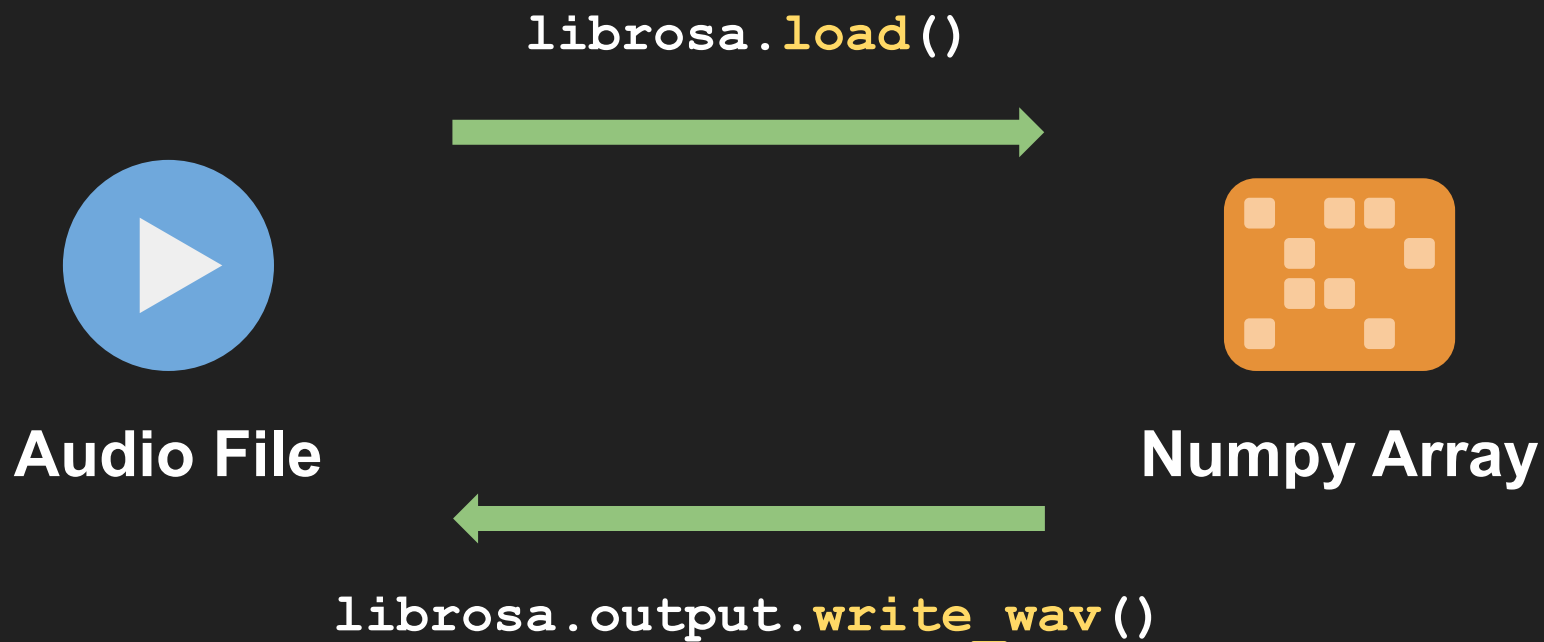


Mono 음성이 5초 길이면
(220500,)



Stereo 음성이 2초 길이면
(2, 88200)

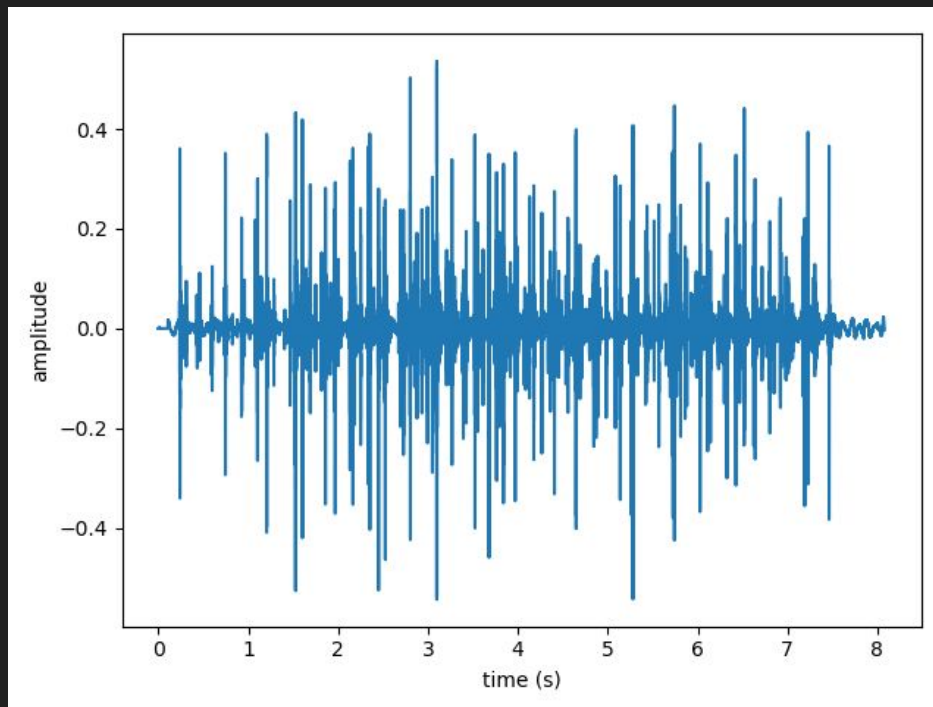
Python에서 음성을 다루려면?
“Librosa” 라이브러리를 쓴다



음성을 보고 듣자!

```
plt.plot()
```

```
librosa.display.waveplot()
```





음성을 보고 듣자!

`sounddevice.play()`

```
# because playing is non-blocking,  
# you must wait or program will be finished before playing audio  
  
sd.play(wave, sr)  
sd.wait()  
  
# or simply get blocked by as below  
  
sd.play(wave, sr, blocking=True)  
  
# for stereo audio  
# NOTE: sounddevice need data of shape (len, 2),  
# so we need to transpose axis  
  
wave, sr = librosa.load(audio_path, sample_rate, mono=False)  
wave = np.transpose(wave, [1, 0])  
  
sd.play(wave, sr, blocking=True)
```

`IPython.display.Audio()`

```
In [3]: # play audio by file path  
# in default, wave is loaded as stereo  
  
IPython.display.Audio(audio_path)  
  
Out[3]:   
  
In [4]: # play audio by Numpy array  
  
import librosa  
  
wave, sr = librosa.load(audio_path) # mono sound  
  
IPython.display.Audio(wave, rate=sr)  
  
Out[4]: 
```

Librosa와 채널 순서가 반대!
(44100, 2) 꼴로 넣어줘야 한다

오디오를 만들어보자



어떻게 오디오를 만들까?

직접 녹음한다

VS

컴퓨터에서 만든다

실제 오디오 파형을 가진다

...

녹음 환경 구성이 어렵다

추가적으로 장비가 있어야 한다

많은 양의 오디오를 구하기 힘들다

...

오디오 환경 구성이 쉽다

원하는 만큼 빠르게 만들 수 있다

음량 조절이나 편집이 쉽다

...

실제 오디오랑 많이 다를 수 있다

...

Numpy를 써서 만들어보자



`np.sin()`



`np.random.normal()`

Numpy를 써서 만들어보자

```
sr = 44100

time = 2.
freq = 440

# create continuous sine wave with frequency

samples = np.arange(time * sr) / sr
wave = np.sin(2 * np.pi * freq * samples)
```

Sampling Rate에 맞게 뽑아주면
원하는 길이의 음성을 만들 수 있다

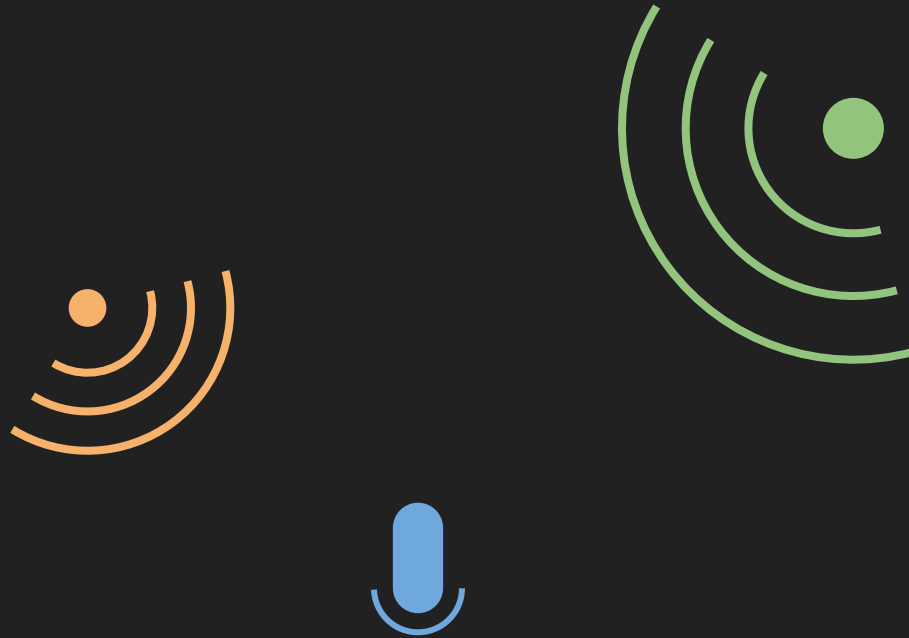
더 다양하게 만들어보자



두 음성을 단순히 더하면
합쳐진 새로운 음성을 구할 수 있다

실제 두 음성을 동시에 트는 것과는 다르다
잔향(Reverb) 이나 위치 등의 요소가 빠진다

더 다양하게 만들어보자



녹음 환경을 시뮬레이션해서 만들 수도 있다
OpenAL 같은 라이브러리를 사용해보자

컴퓨터로도 녹음해 보자

```
sounddevice.rec()
```

```
sounddevice.playrec()
```

```
sounddevice.Stream(callback=callback)
```

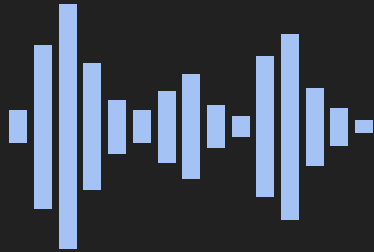
실시간으로 음성을 받아 처리할 수 있다

데모 프로그램이나 테스트 시 유용하다

오디오를 이미지로



무엇이 문제일까?



VS

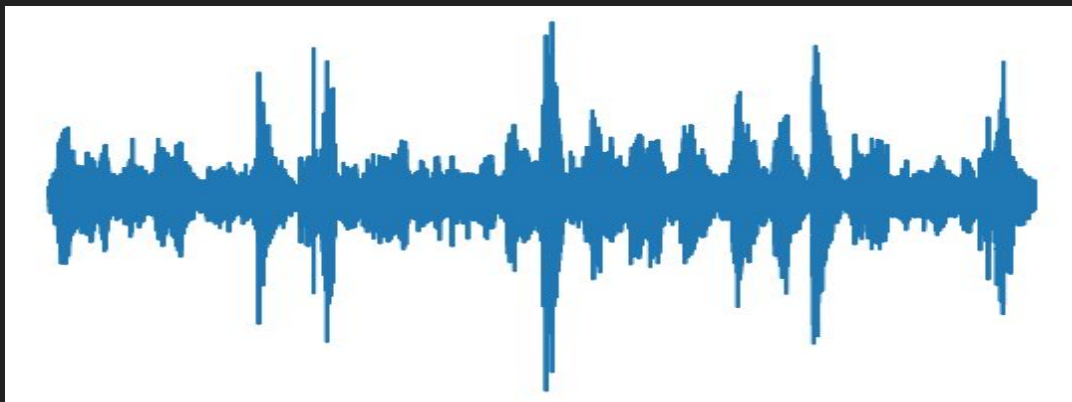
안녕하세요

44100Hz의 스테레오 1초 음성
1.3 MB

문자열 “안녕하세요”
10 Byte

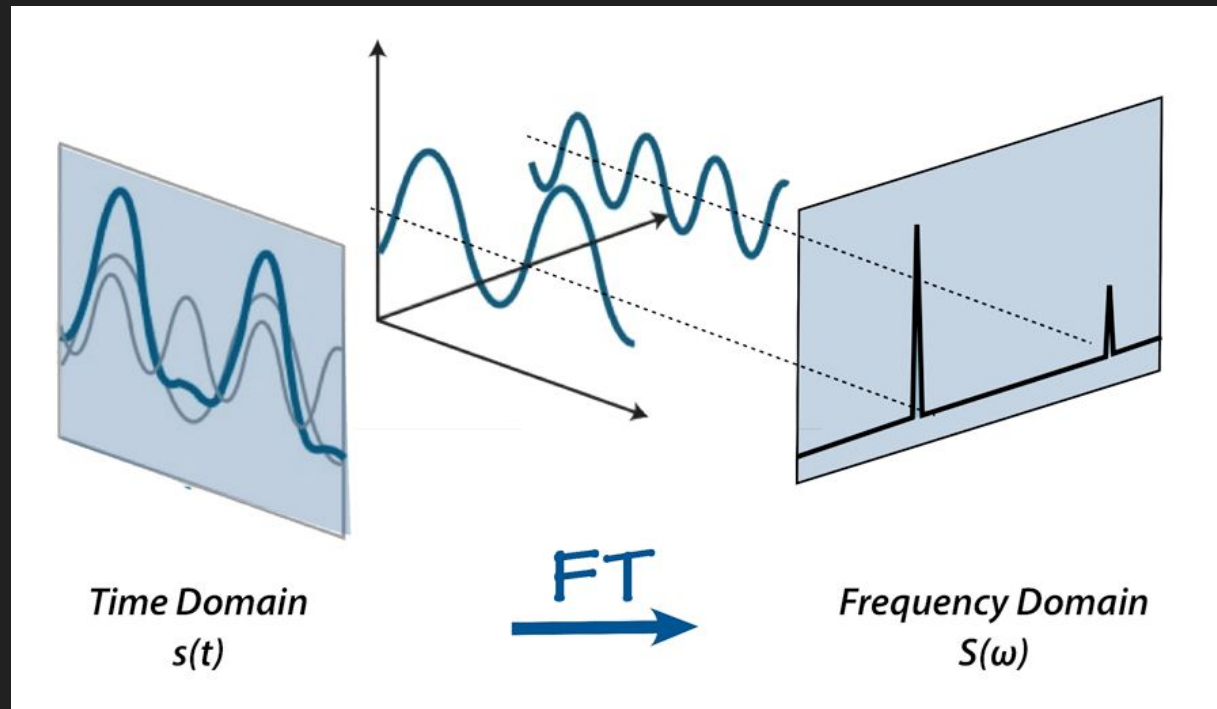
정보의 밀도가 낮다

무엇이 문제일까?

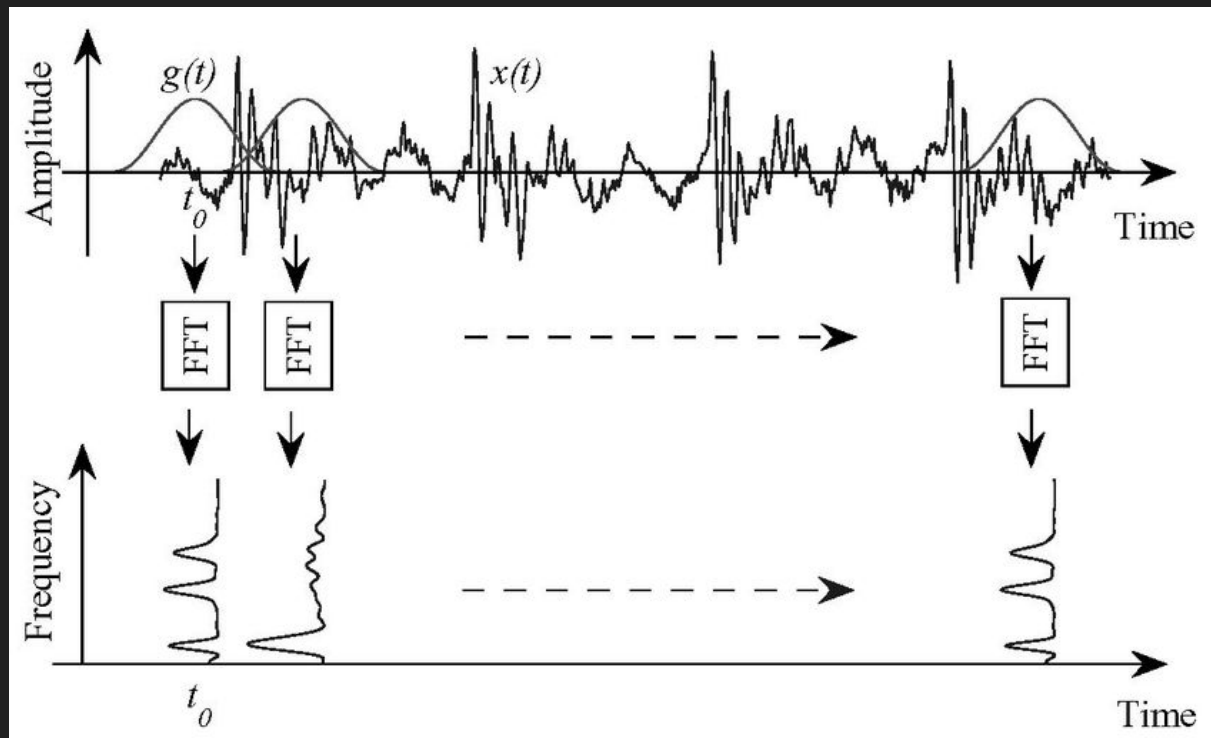


개? 고양이? 목소리?
파형만 보고 음성을 예측하기 어렵다

파형을 전부 그리지 말고
주파수 정보만 뽑아내자!

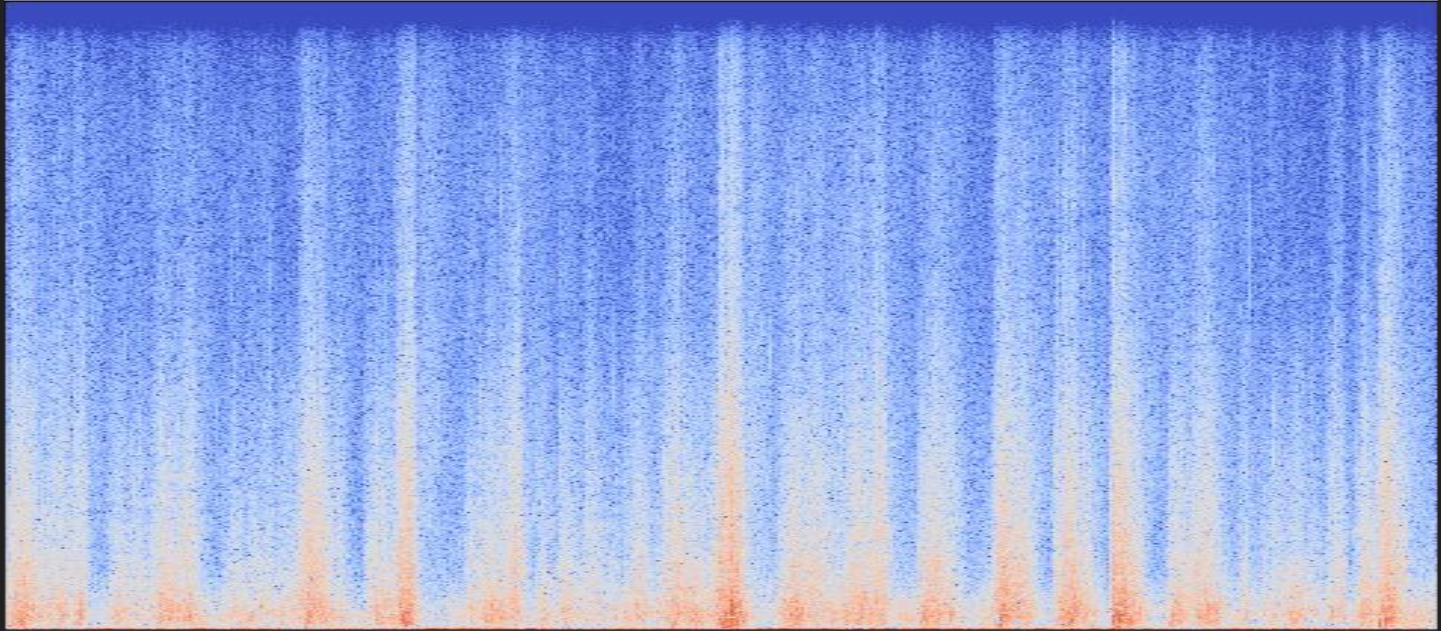


시간 정보가 사라지는데?



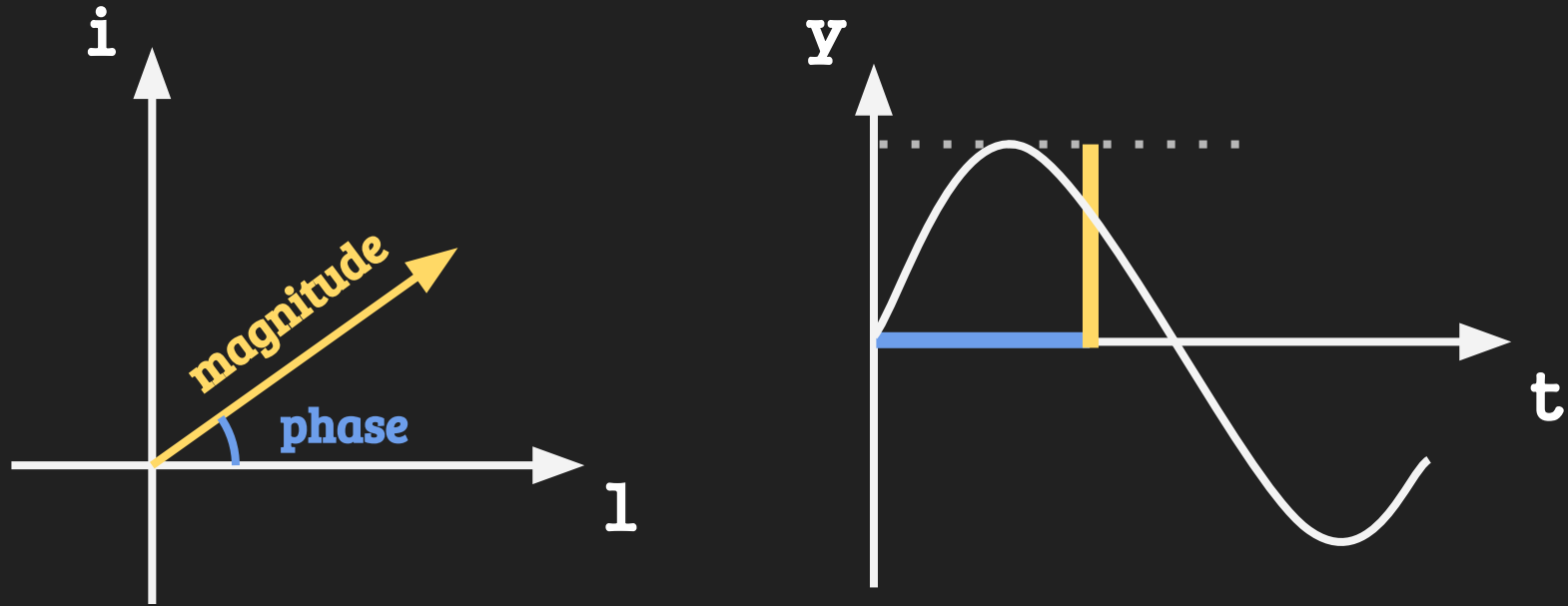
작은 시간 단위로 잘게 잘라서
각각 변환해보자

변환한 feature를 이어붙이면
“Spectrogram”이 된다



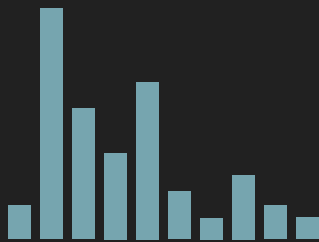
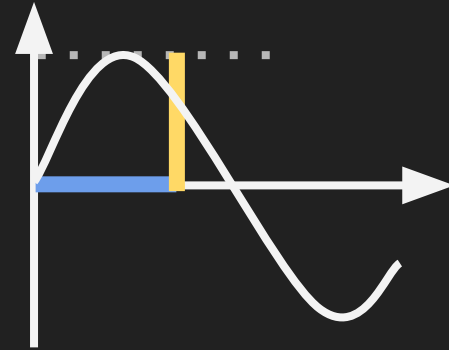
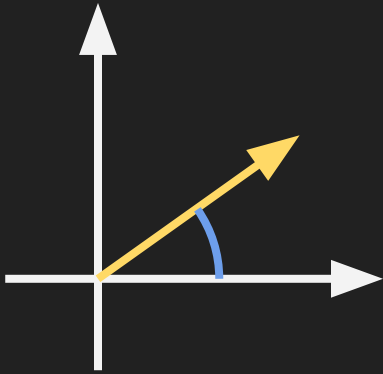
(주파수, 시간) 축을 갖는 2차원 데이터
이미지처럼 볼 수 있다

Spectrogram의 pixel 값은 복소수다



진폭을 나타내는 **Magnitude**와
위상을 나타내는 **Phase**로 이루어진다

Spectrogram과 Wave는 서로 변환이 가능하다



`librosa.istft()`

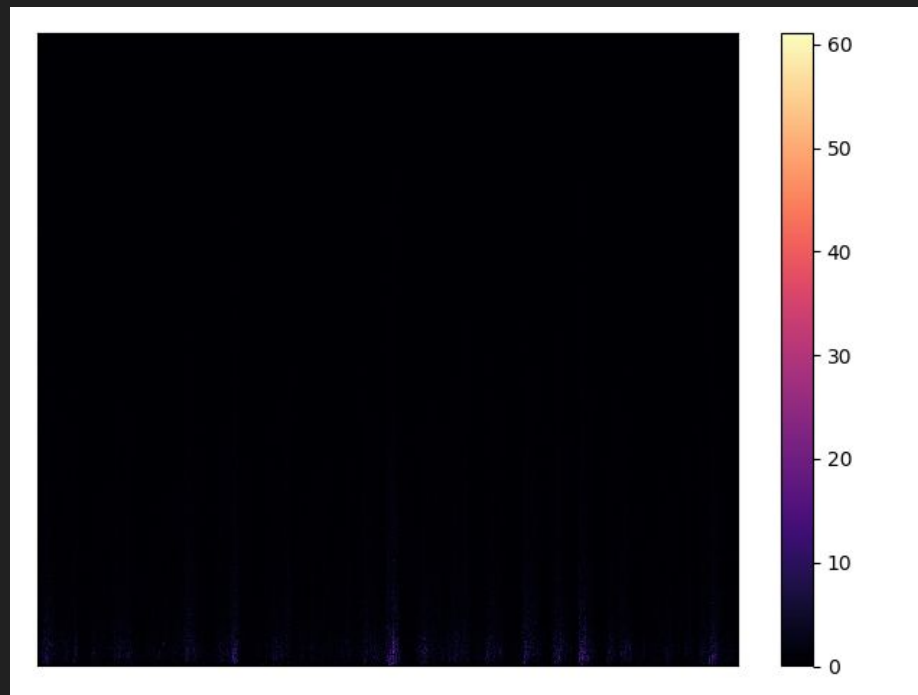


`librosa.stft()`



Spectrogram을 관찰해보자!

```
librosa.display.specshow()
```

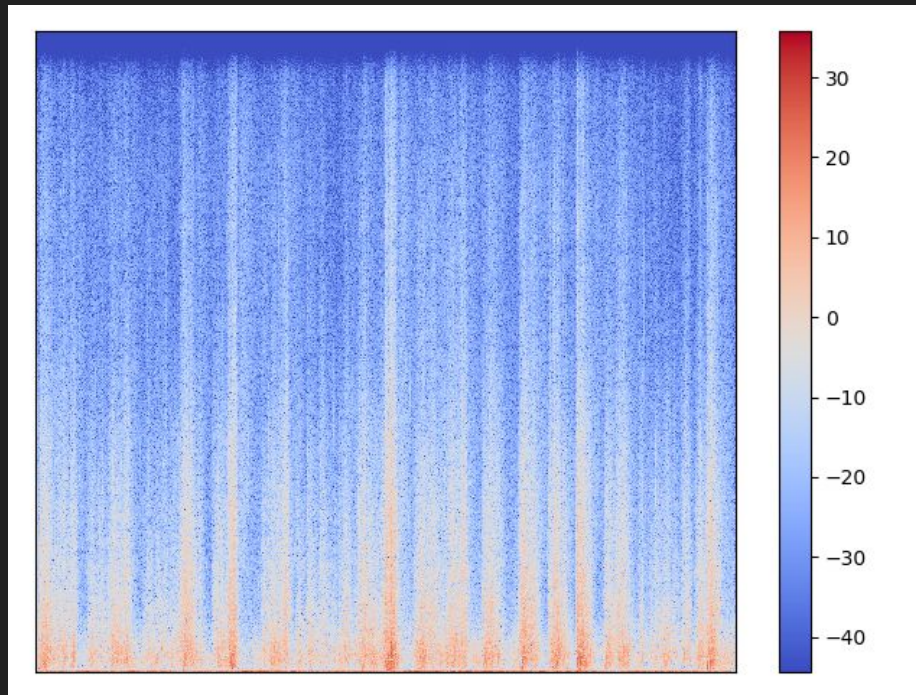


...?

Spectrogram은 “복소수”

`np.abs()`

`librosa.amplitude_to_db()`



Magnitude를 볼 수 있다
이미지처럼 학습에 그대로 쓸 수 있다

여기서 잠깐!

Amplitude to dB 가 뭘까?

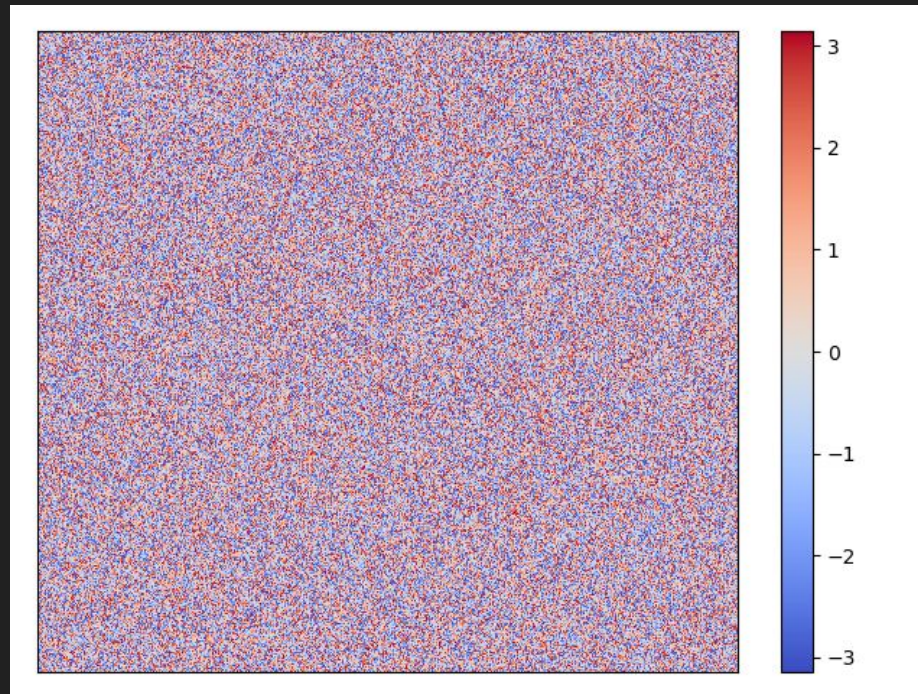
```
librosa.amplitude_to_db()  
librosa.power_to_db()
```

$$dB = 10 \times \log \frac{P_1}{P_0} = 10 \times \log \frac{V_1^2}{V_0^2}$$

전압 or 전력을 상대적 세기로 바꿔준다
분석하기에도 더 편해진다

Phase도 그려볼 수 있다

```
librosa.display.specshow(np.angle())
```



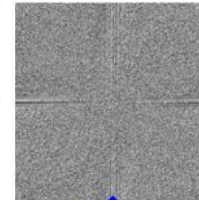
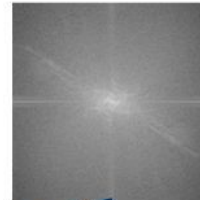
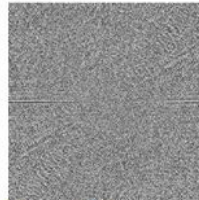
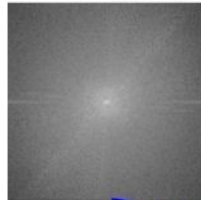
그냥 노이즈처럼 보이는데...
Phase를 쓰는 의미가 있을까?

Phase Carries More Information

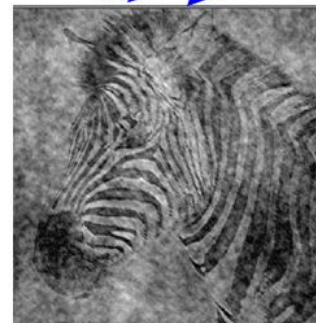
Raw
Images:



Magnitude
and
Phase:



Reconstruct
(inverse FFT)
mixing the
magnitude and
phase images



Phase "Wins"

Phase가 더 많은 정보를 담는다
하지만 다루기 쉽지 않다...

Magnitude와 Phase를 구해보자

함수 두 개로 쉽게 할 수 있다

```
wave, sr = librosa.load(audio_path, sample_rate)

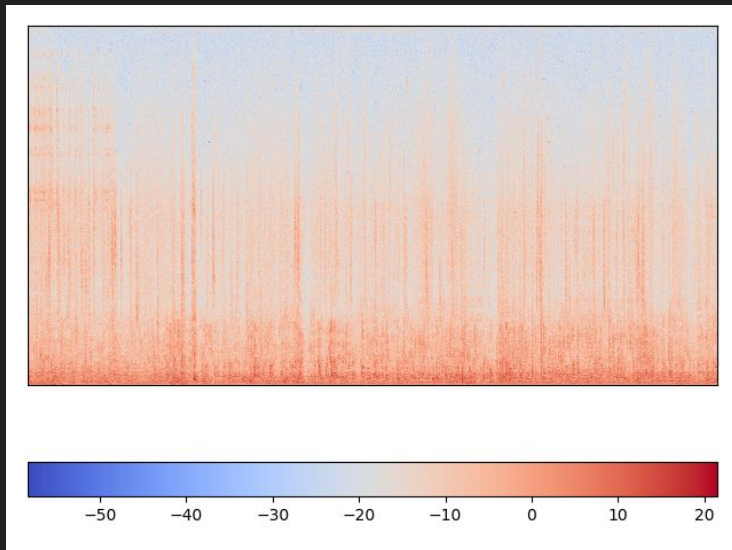
spec = librosa.stft(wave)

# to get magnitude & phase, get absolute & angle of complex value

mag = np.abs(spec)
phase = np.angle(spec)
```

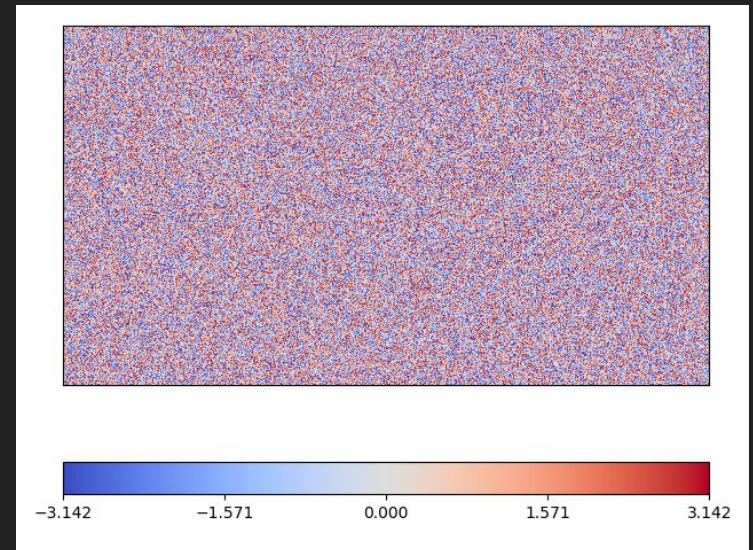
Magnitude

`np.abs()`

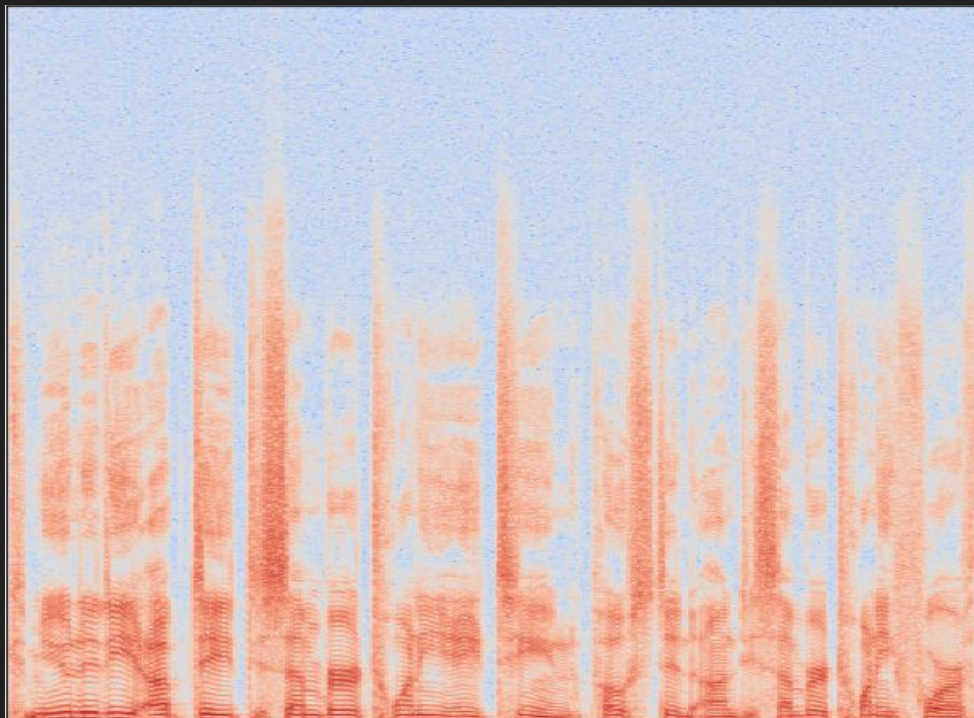


Phase

`np.angle()`



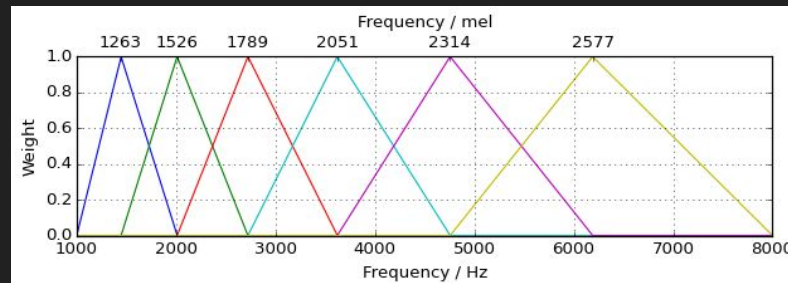
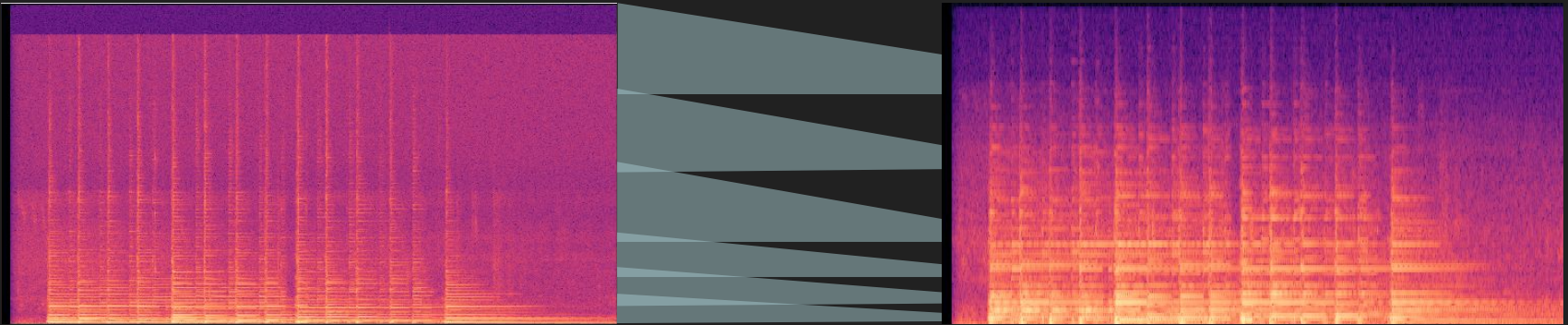
Spectrogram을 보면 소리를 알 수 있다
주파수 축이 생겨나 그림이 되었다



고주파 쪽에 배경이 많다
목소리 부분만 집중해서 볼 수
없을까?

사람의 청력은 특정 주파수들에 맞춰져
있다

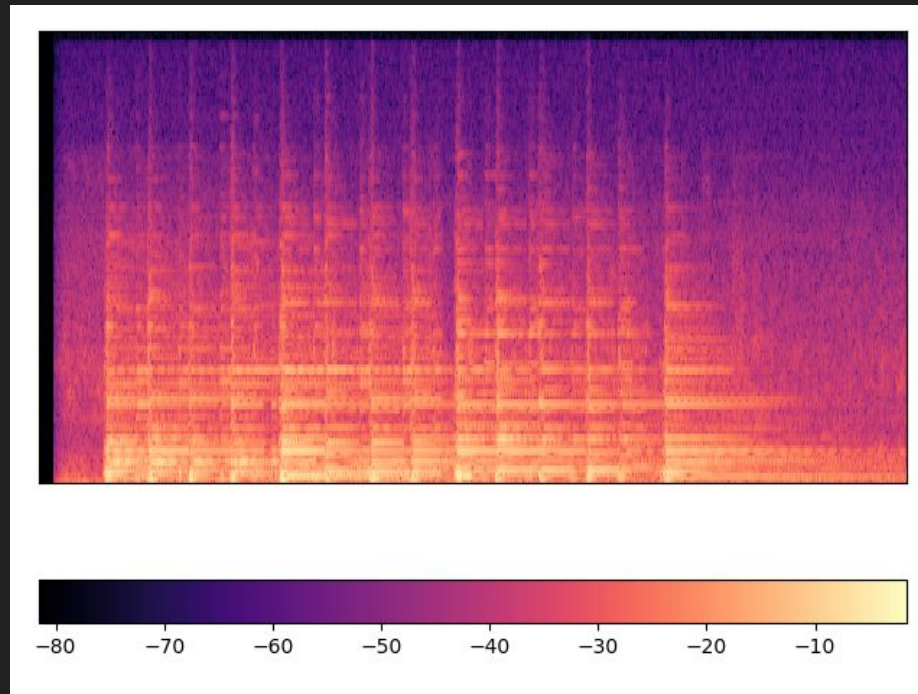
달팽이관 구조를 보면 확인할 수 있다



달팽이관 역할을 하는 필터를 씌운다
이 이미지를 “Mel Spectrogram” 이라
한다

Mel Spectrogram은 “실수”
다른 처리 없이 그대로 사용할 수 있다

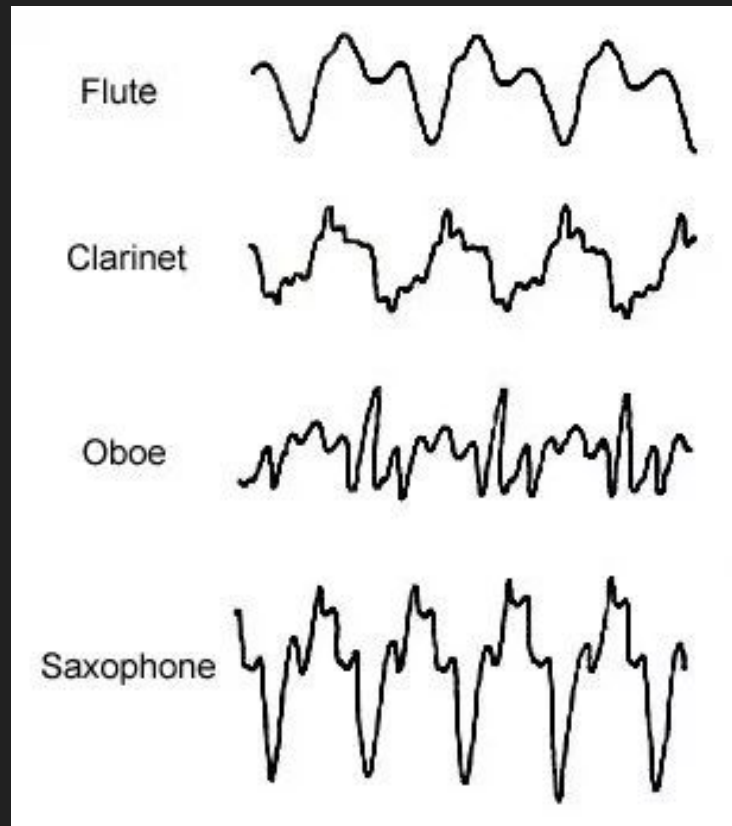
```
librosa.feature.melspectrogram()
```



특정 주파수들이 강조되고 변형되었기 때문에
파형으로 복구할 때 완벽하지 않다

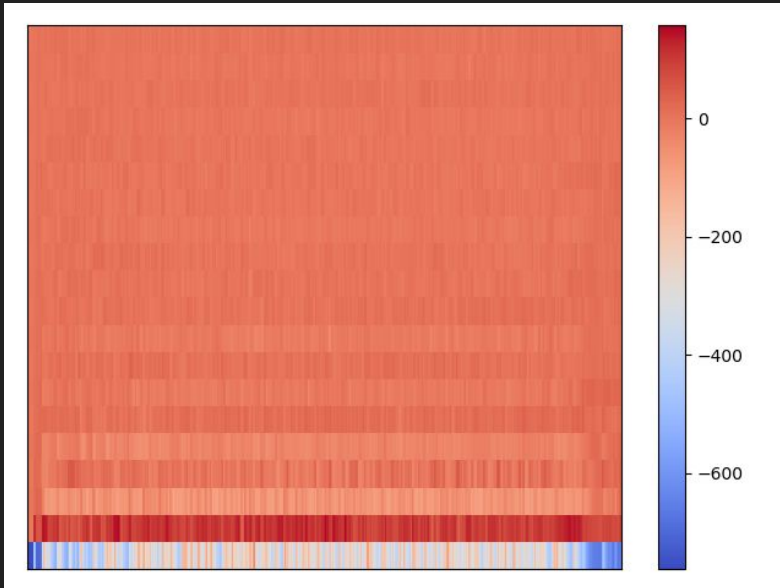
음성의 맵시를 구분하고 싶을 때는?

음의 높낮이나 크기는 궁금하지 않다

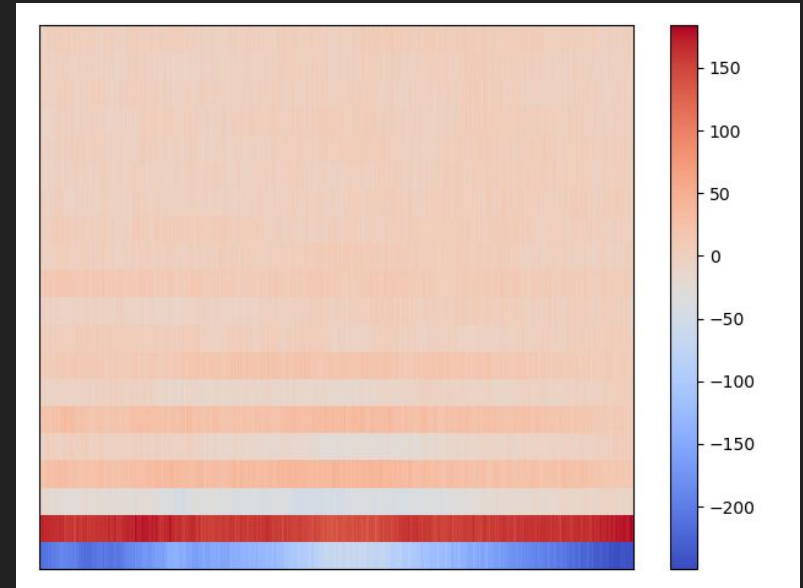


MFCC를 사용한다

Mel Frequency Cepstral Coefficient



박수



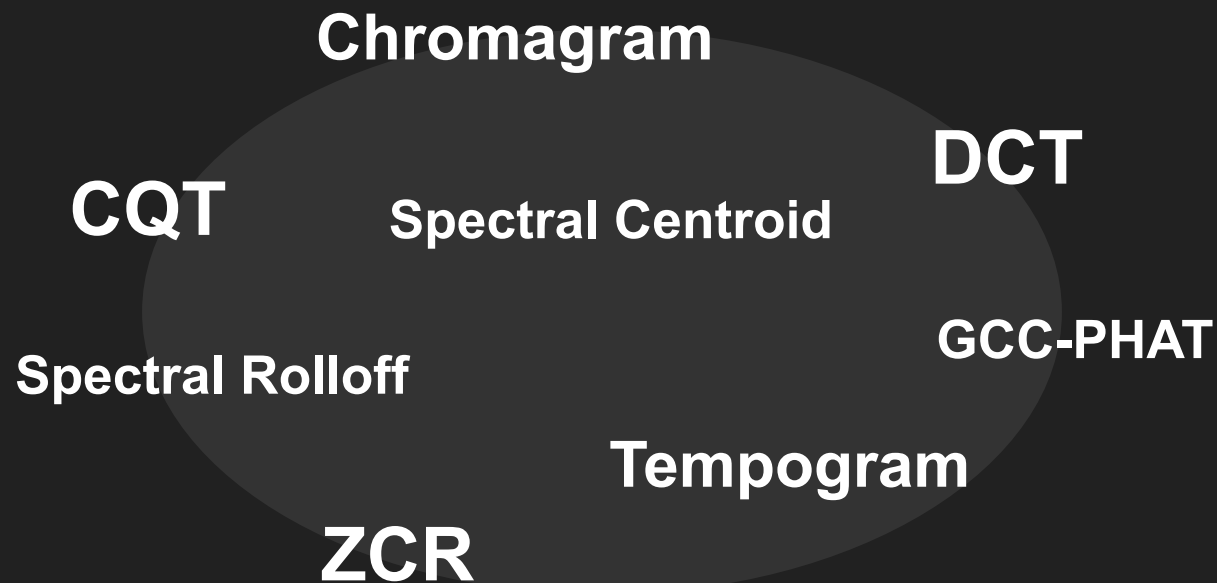
오토바이

음성 인식, 악기나 장르 구분 등에 널리
쓰인다

음정이 중요할 때는 잘 쓰지 않는다

이 외에도 Feature 종류가 엄청
많다

“scipy.signal” 라이브러리도 참고해보자



여러가지 전처리

0 1 1 1 0 1 1 0 0 1 0
1 0 1 0 1 0 0 1 0
1 0 0 0 1 1 1 1
1 0 1 1 1 1 1 1
1 0 1 0 0 1 0 1
1 0 1 0 0 1 0 1

이런 경험 한번쯤은...

Music 1



소리가 작네...

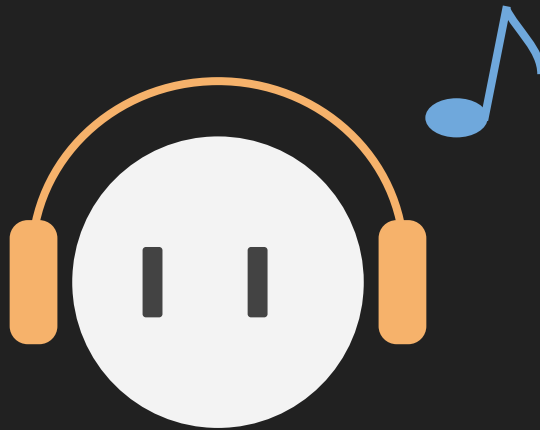


Volume Up

Music 2



일정한 크기의 음성을 듣고 싶다면
정규화가 필요하다



But **How?**

크게 두 가지 방법이 있다

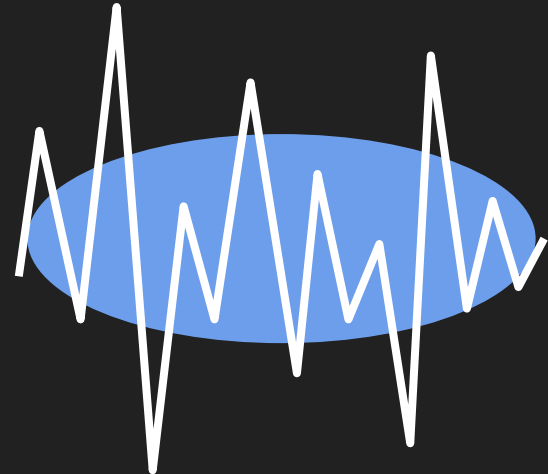
Peak Normalization

최대값에 맞춰서

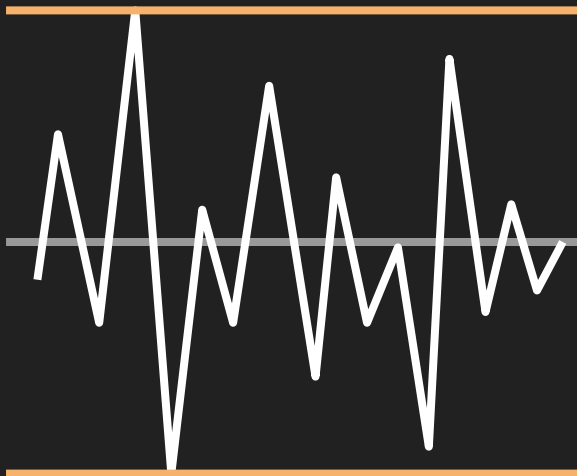


Loudness Normalization

전체 음량에 맞춰서



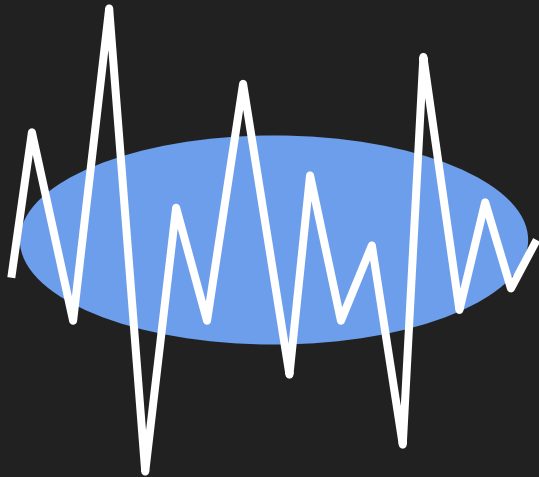
Peak Normalization



쉽게 쓸 수 있다
최대 진폭에 맞게 조절이 가능하다

튀는 노이즈가 있을 때 좋지 않다

Loudness Normalization

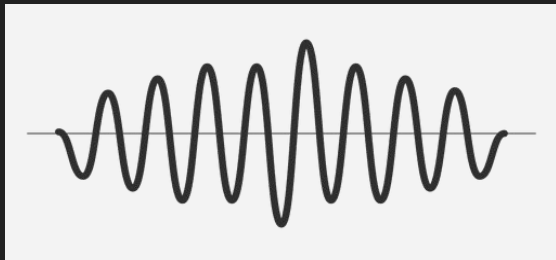
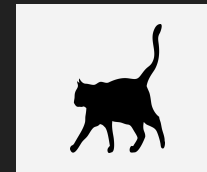
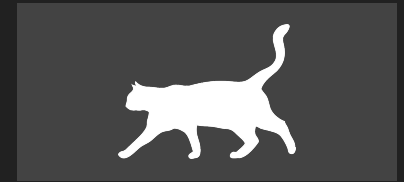
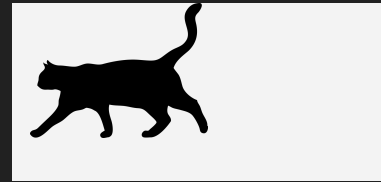
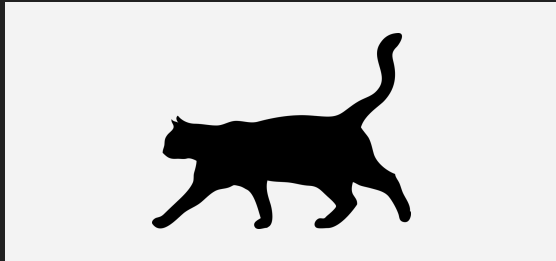


음량을 맞춰줄 수 있다
파형에 크게 구애받지 않는다

최대 제한을 넘어가 버릴 수 있다

두 방법 모두 단점이 있다
압축과 제한으로 단점을 완화한다

더 많은 데이터가 필요할 때
이미지처럼 쉽게 늘릴 수 있을까?



?

이미지에 다양한 효과를 주듯이
음성도 여러 효과를 줄 수 있다

Pitch

Play speed

Relative volume of sources

Shift / Delay

High / Low freq filter

Reverb / Echo

Fade in / out

Noise / Distort

...

이미지에 다양한 효과를 주듯이
음성도 여러 효과를 줄 수 있다

`pysndfx.AudioEffectsChain()`

```
wave, sr = librosa.load(audio_path, sample_rate)

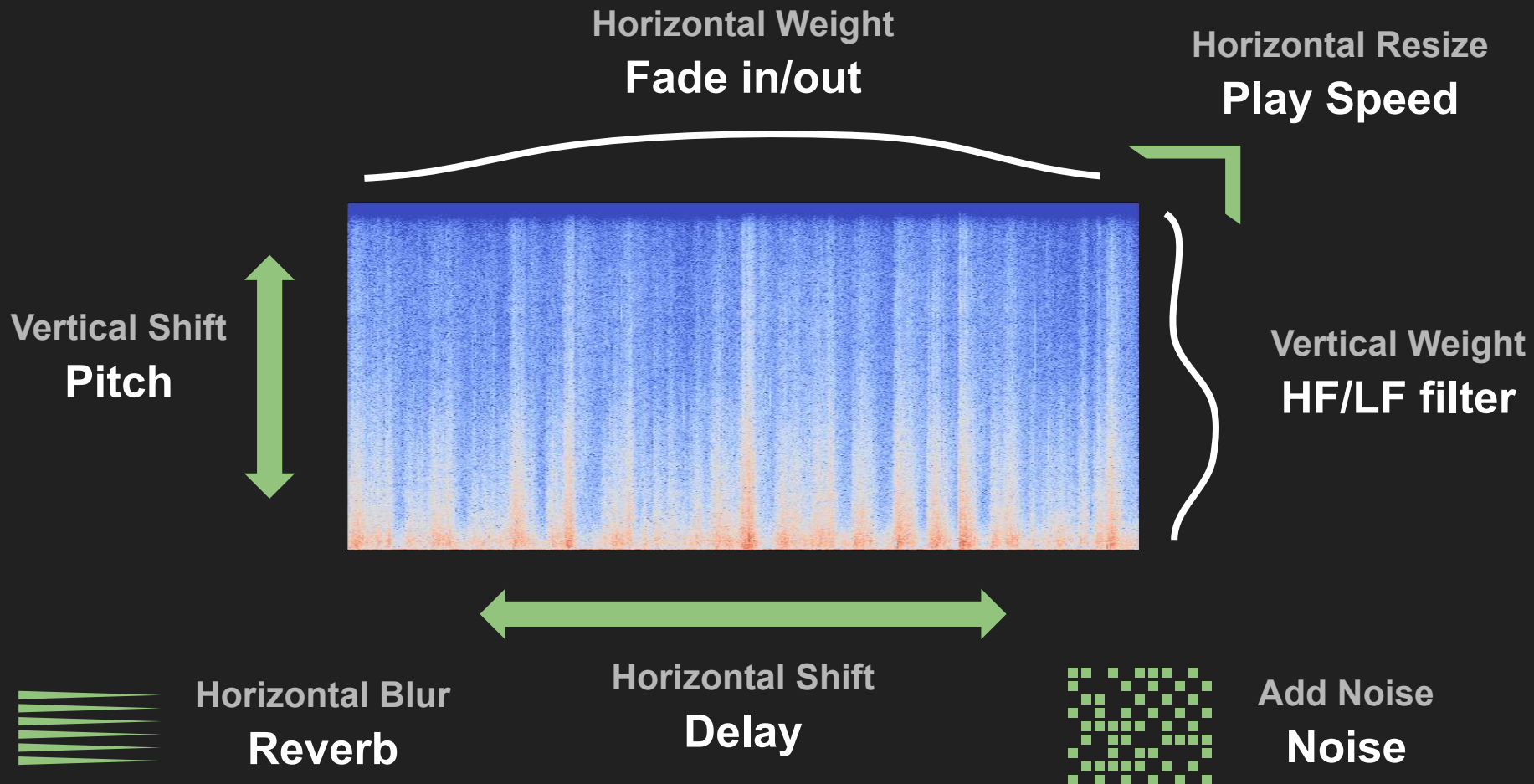
fx = (
    AudioEffectsChain()
    .highshelf()
    .delay()
    .phaser()
    .reverb()
    .lowshelf()
)

result = fx(wave)

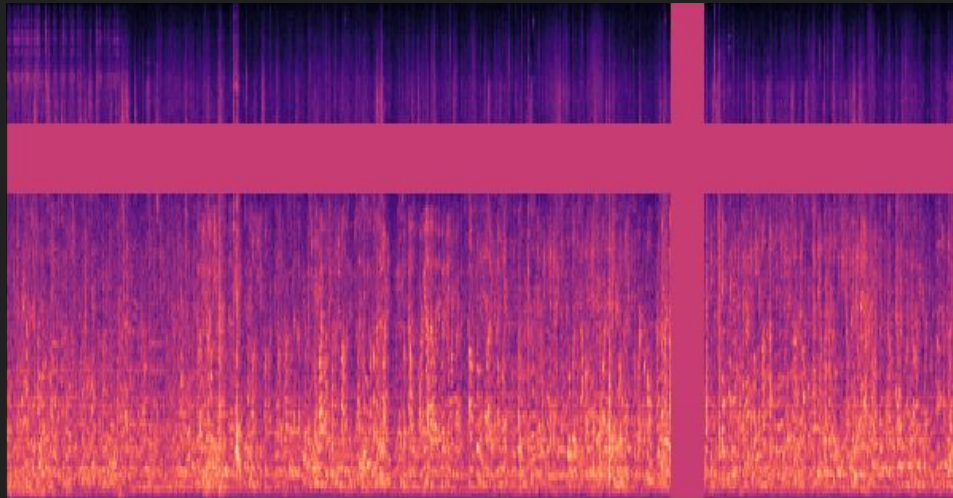
sd.play(result, sr, blocking=True)
```

Pysndfx를 사용해 간단히 해볼 수 있다
SoX 라이브러리를 설치해야한다!

음성도 이미지로 볼 수 있으니
이미지에서 하듯이 해보자



더 새로운 방법?
SpecAugment



단순히 가로줄과 세로줄만 **Masking**해도
데이터 증강 효과를 낼 수 있다

GPU를 활용해 보자



librosa 등 대다수 라이브러리는 CPU 기반
전처리가 **Bottleneck**이 될 수
있다

어떻게 해결할까?

멀티프로세싱

미리 전처리하기

또는

GPU에서 전처리하기

tf.signal

Module: tf.signal

Functions

`dct(...)` : Computes the 1D [Discrete Cosine Transform (DCT)][dct] of `input`.

`fft(...)` : Fast Fourier transform.

`fft2d(...)` : 2D fast Fourier transform.

`fft3d(...)` : 3D fast Fourier transform.

`frame(...)` : Expands `signal`'s `axis` dimension into frames of `frame_length`.

`hamming_window(...)` : Generate a [Hamming][hamming] window.

`hann_window(...)` : Generate a [Hann window][hann].

`idct(...)` : Computes the 1D [Inverse Discrete Cosine Transform (DCT)][idct] of `input`.

`ifft(...)` : Inverse fast Fourier transform.

`ifft2d(...)` : Inverse 2D fast Fourier transform.

`ifft3d(...)` : Inverse 3D fast Fourier transform.

`inverse_stft(...)` : Computes the inverse [Short-time Fourier Transform][stft] of `stfts`.

`inverse_stft_window_fn(...)` : Generates a window function that can be used in `inverse_stft`.

`irfft(...)` : Inverse real-valued fast Fourier transform.

`irfft2d(...)` : Inverse 2D real-valued fast Fourier transform.

`irfft3d(...)` : Inverse 3D real-valued fast Fourier transform.

`linear_to_mel_weight_matrix(...)` : Returns a matrix to warp linear scale spectrograms to the [mel scale][mel].

`mfccs_from_log_mel_spectrograms(...)` : Computes [MFCCs][mfcc] of `log_mel_spectrograms`.

`overlap_and_add(...)` : Reconstructs a signal from a framed representation.

`rfft(...)` : Real-valued fast Fourier transform.

`rfft2d(...)` : 2D real-valued fast Fourier transform.

`rfft3d(...)` : 3D real-valued fast Fourier transform.

`stft(...)` : Computes the [Short-time Fourier Transform][stft] of `signals`.

STFT / MFCC 등을 GPU에서 구할 수
있다

Keras에서도 활용할 수 있다

kapre

kapre

Keras Audio Preprocessors. Written by Keunwoo Choi.

Why bother to save STFT/melspectrograms to your storage? Just do it on-the-fly on-GPU.

How demanding is the computation? [Check out this paper!](#)

Contents

- [News](#)
- [Installation](#)
- [Usage](#)
- [One-shot example](#)
- [How to cite](#)
- [API Documentation](#)
 - [time_frequency.Spectrogram](#)
 - [time_frequency.Melspectrogram](#)
 - [utils.AmplitudeToDB](#)
 - [utils.Normalization2D](#)
 - [filterbank.Filterbank](#)
 - [augmentation.AdditiveNoise](#)

Keras Layer처럼 사용할 수 있다
자주 쓰이는 **feature**들을 쉽게 사용해보자

torchaudio

TORCHAUDIO

The `torchaudio` package consists of I/O, popular datasets and common audio transformations.

Package Reference

- `torchaudio.sox_effects`
- `torchaudio.datasets`
 - VCTK
 - YESNO
- `torchaudio.kaldi_io`
 - Vectors
 - Matrices
- `torchaudio.transforms`
- `torchaudio.legacy`

데이터셋 / 음성 효과 / 라이브러리 호환
아직 기능은 적지만 앞으로 활용하기 좋을 듯하다

torchaudio-contrib

torchaudio-contrib

Goal: To propose audio processing Pytorch codes with nice and easy-to-use APIs and functionality.

👉 This should be seen as a community based proposal and the basis for a discussion we should have inside the pytorch audio user community. Everyone should be welcome to join and discuss.

Our motivation is:

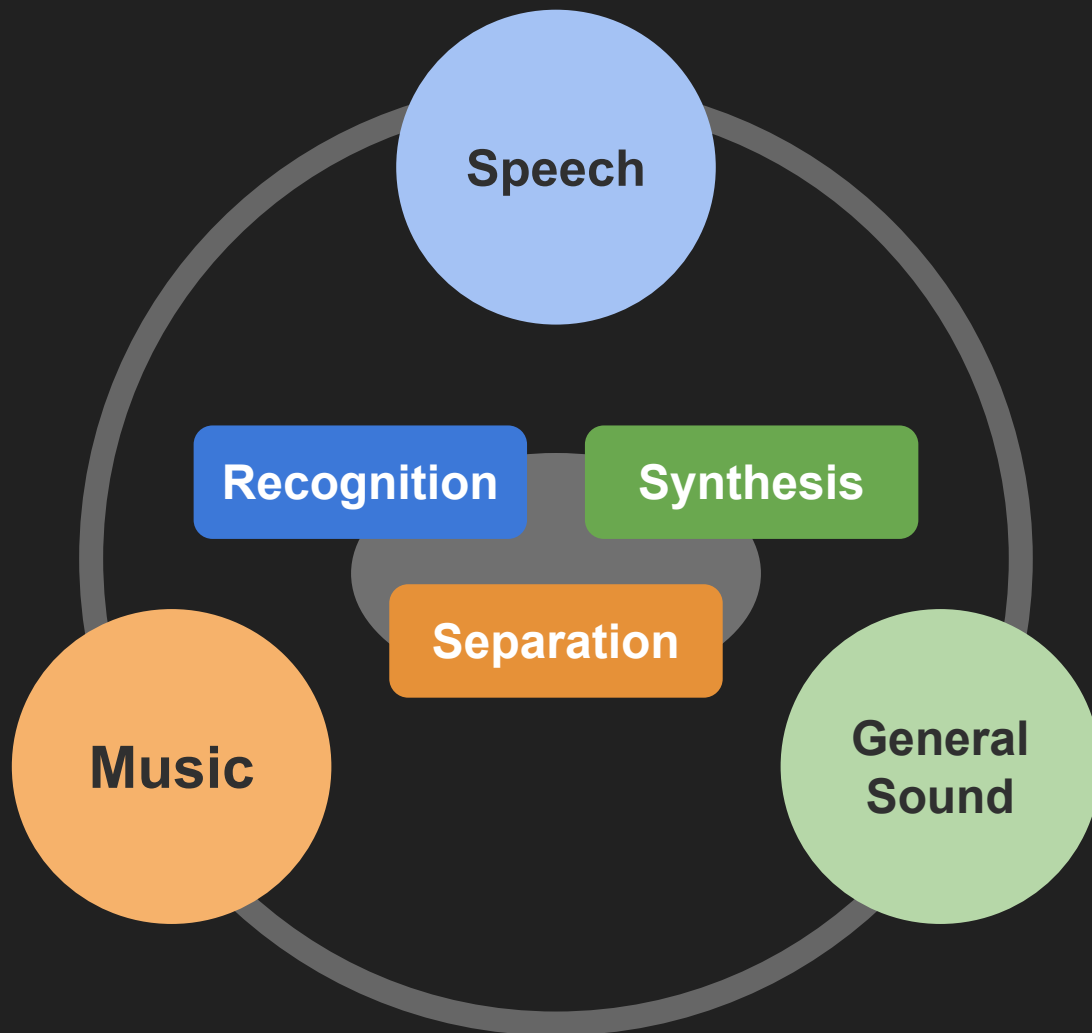
- API design: Clear, readable names for class/functions/arguments, sensible default values, and shapes.
 - Reference: [librosa](#) (audio and MIR on Numpy), [kapre](#) (audio on Keras), [pytorch/audio](#) (audio on Pytorch)
- Fast processing on GPU
- Methodology: Both layer and functional
 - Layers (`nn.Module`) for reusability and easier use
 - and identical implementation with `Functionals`
- Simple installation
- Multi-channel support

주로 쓰는 **feature**들을 사용할 수 있다
nn.Module 꼴로 쉽게 사용할 수 있다

다뤄보기 좋은
데이터셋과 도구들



오디오에도 여러 분야가 있다
Sub Domain과 Task 별로 나뉘볼 수 있다



어떤 데이터셋이 있을까?

간단한 데이터셋 Spoken Digit Datasets

Free Spoken Digit Dataset (FSDD)

DOI [10.5281/zenodo.1342401](https://doi.org/10.5281/zenodo.1342401)

A simple audio/speech dataset consisting of recordings of spoken digits in `wav` files at 8kHz. The recordings are trimmed so that they have near minimal silence at the beginnings and ends.

FSDD is an open dataset, which means it will grow over time as data is contributed. In order to enable reproducibility and accurate citation the dataset is versioned using Zenodo DOI as well as `git tags`.

Current status

- 4 speakers
- 2,000 recordings (50 of each digit per speaker)
- English pronunciations

목소리나 발화 데이터셋은?

LibriSpeech

1000시간의 영어 오디오북 발화 데이터셋 (60GB)

VoxCeleb

발화자 6000명의 영상-음성 데이터셋

The Spoken Wikipedia Corpora

영어 / 독일어 / 네덜란드어 위키 문서 발화 데이터셋 (38GB)

음악 관련 데이터셋은?

Free Music Archive

고음질의 음악 분석용 데이터셋 (1TB)

Million Song Dataset

현대 곡 음성과 feature 데이터셋 (280GB)

The NSynth Dataset

1000개의 악기 별 음표 데이터셋

일반적인 음성의 데이터셋은?

AudioSet

5800시간의 Youtube 영상 분류 데이터셋

Freesound Datasets (FSD)

클라우드소싱 기반의 일반 음성 데이터셋

Urban Sound

10종류의 도심 속 소리 분류 데이터셋 (6GB)

DCASE

음성 감지와 분류를 위한 대회 및 데이터셋

한국어 음성은 어떻게 구하지?

ETRI 데이터셋을 써보자!

음성 학습데이터

한국어 및 영어 음성인식 기술을 개발하기 위한 과학기술정보통신부의 R&D인 “언어학습을 위한 자유발화형 음성대화처리 원천기술 개발” 과제에서는 음성대화 인터페이스 및 외국어교육을 위한 다양한 음성인식 기술 개발과 관련 데이터를 구축하고 있습니다.

과제를 수행하면서 구축한 약 300명 규모의 음향모델 훈련을 위한 학습데이터를 배포하여 관련 연구 분야에 도움이 되고자 합니다.

아래의 표와 같이 잡음 및 입력 환경, 비원어민 음성인식을 고려하여 구분된 학습데이터의 다운로드를 제공합니다.

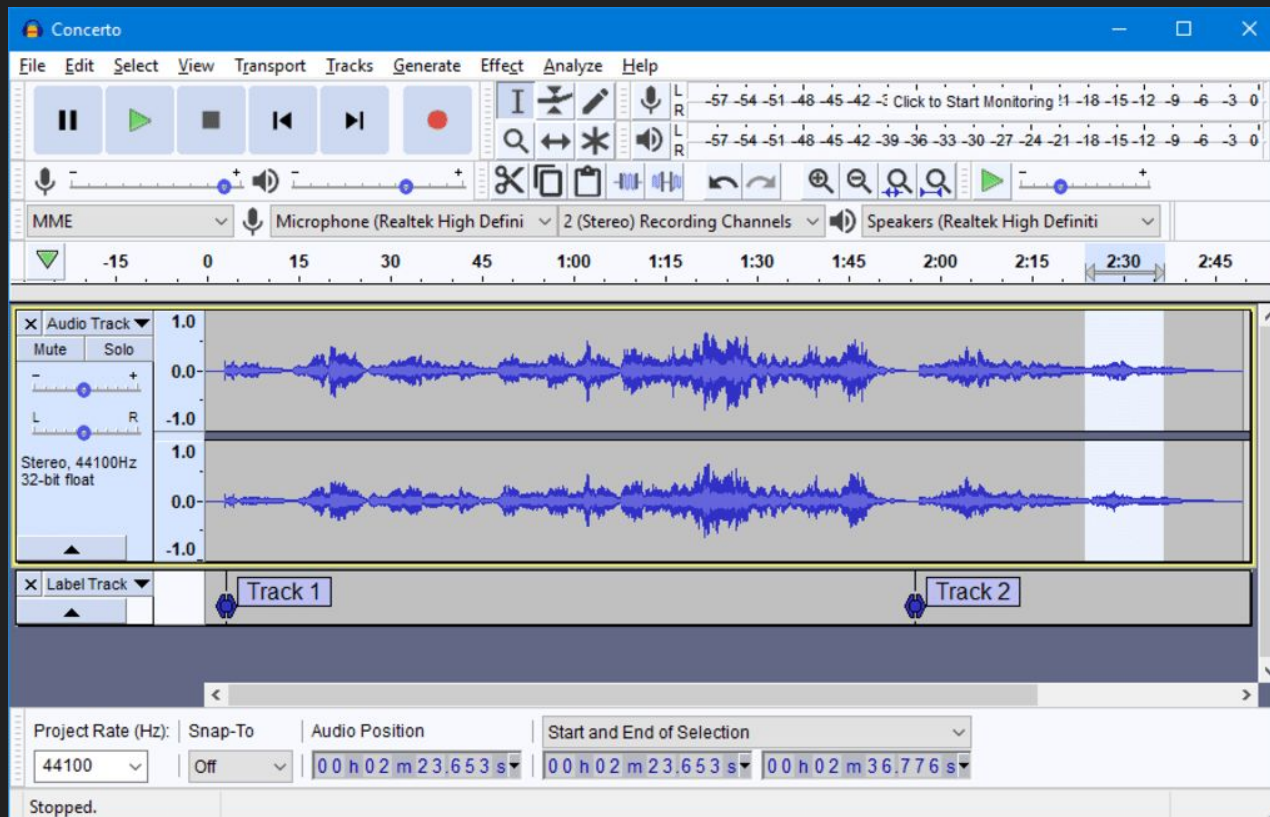
적용 포맷은 공히 16kHz 샘플링 주파수로 녹음된 16-비트 선형(linear) PCM 파일이며, 부가 정보로서 파일별로 단어 단위의 전사 정보 및 발성자의 성별 정보가 제공됩니다.

| 데이터 명 | 구축 방법 및 환경 | 분량(전체 발화수) |
|---------------------------------------|--|-----------------------------|
| 다채널 잡음처리 기술 개발 및 평가용 데이터 | · 8채널 어레이 마이크 이용 (1M거리 녹음) · 사무실 잡음 환경 | 50명 * 100발화 * 8마이크 (40,208) |
| 텔레매틱스 목적지 인식을 위한 음향모델 적응용 음성 데이터 | · 정지 및 주행 중인 자동차 환경: 정차중, 60Km/h, 100Km/h의 3가지 주행 상황 · 네비게이션 단말기 적용 | 50명 * 120발화 (6,000) |
| 잡음처리 및 음성검출을 위한 스마트폰 환경 연속어 음성 데이터 | · 아이폰4, 갤럭시S2 사용 · 조용한 사무실 환경 | 50명 * 100발화 * 2환경 (10,000) |
| 한국어 및 영어 음향모델 훈련용 음성 데이터 | · 한국어 : 한국어 자연어 발성 문장 · 영어 : 한국인이 발성한 영어 문장 | 한/영 각 50명 * 100발화 (10,000) |
| 음성인터페이스 개발을 위한 어린이 음성 데이터 | · 어린이 음향모델 훈련용 · 초등 1~4학년 대상 녹음 · 아이폰5, 갤럭시S4, 마이크 동시 녹음 | 50명 * 100발화 * 3환경 (16,200) |

이 외에도 유용한 도구가 많다

Audacity

무료 음성 편집 / 분석 도구



이 외에도 유용한 도구가 많다

SoX

OpenAL

Scipy.signal

PyAudio

Pydub

Audiogen

Pysndfx

...

PPT에 나온 코드가 궁금하다면?

nuxlear/keras-audio

부족한 내용은 채워나가는 중입니다 :)

Reference

최근우님 블로그 <http://keunwoochoi.blogspot.com/>

dB(데시벨) 이란?

<http://audio-probe.com/documentation/db%EB%9E%80-%EB%AC%B4%EC%97%87%EC%9D%B8%EA%B0%80/>

오디오 feature extraction <https://deepmi.me/deeplearning/19071/>

MFCC 설명 <https://m.blog.naver.com/mylogic/220988857132>

Librosa Documentation <https://librosa.github.io/librosa/index.html>

audio normalization https://en.wikipedia.org/wiki/Audio_normalization

specaugment with pytorch https://github.com/zcaceres/spec_augment

Magnitude vs Phase <https://slideplayer.com/slide/6923643/> (8:00)

Audio Datasets <https://towardsdatascience.com/a-data-lakes-worth-of-audio-datasets-b45b88cd4ad>

Awesome - python for scientific audio

<https://project-awesome.org/faroit/awesome-python-scientific-audio?fbclid=IwAR0KFPMRmc6taIp2RjT6cB2lvjrJNg4nGM7B3TPf6zKlIZ5McSc0YbnzEVs#read-write>

QnA