# Speaker Classification

## 화자 분류

이승백

# Table of contents

01

# 개발 배경

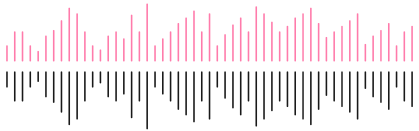
1:23

2:36

# 화자별 자동 자막 서비스를 요구하는 시장은 커진다.

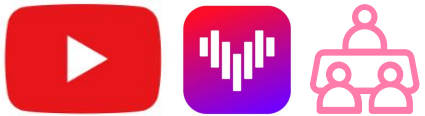
## Video

- 한국은 문맹률이 낮기 때문에 해외영상보다 자막이 많다.
- 영상에 자막을 넣으면 조회수가 늘어난다.



## Audio

- 음원 플랫폼이 오디오 영화·드라마 등을 자체 제작하며 시장 영역을 넓히고 있다.
- 오디오 영화·드라마는 여러 명의 성우가 개별 캐릭터를 맡아 연기한다.
- 책 읽어주는 서비스인 오디오북도 있다.



## Conference

- 자동 회의록 작성에 활용 가능하다.

**Speech to Text** 는 많은 양의 데이터 셋을 요구하므로, 프로젝트 기간에 유의미한 결과를 기대하기 어렵다.

그러므로, 음성데이터를 화자별로 구분하여 시간별로 어떤 화자가 말하는지 웹으로 구현했다.

02

# 데이터 분석

# 아날로그 신호를 디지털 신호로 전환하는 과정

## 표본화(Sampling)

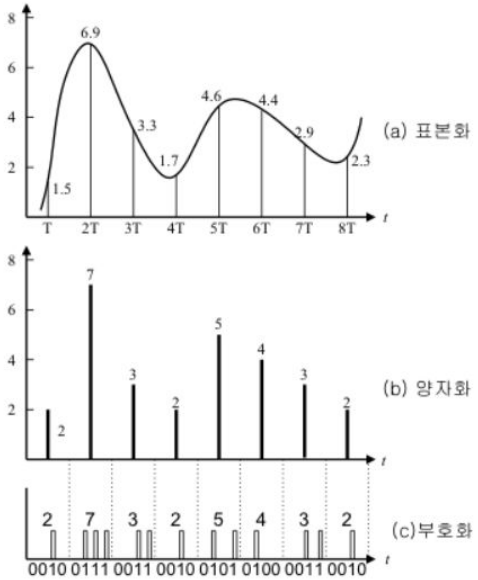
- 아날로그 신호에 일정 시간 간격으로 표본을 취하는 과정

## 양자화(Quantization)

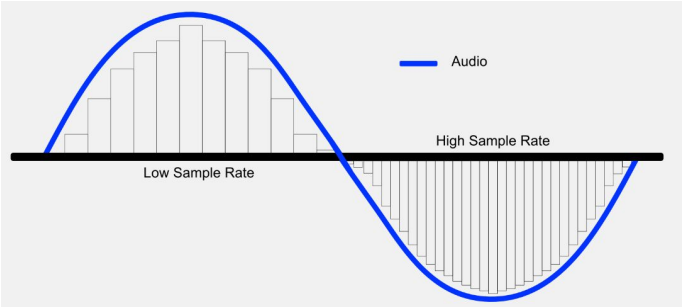
- 표본화된 각 점을 진폭에 따라 어느정도의 정밀도로 표현할 것인지 정하는 과정

## 부호화(Coding)

- 표본화와 양자화를 거친 디지털 정보를 2진수로 표현하는 과정




< 디지털 전환 과정 >



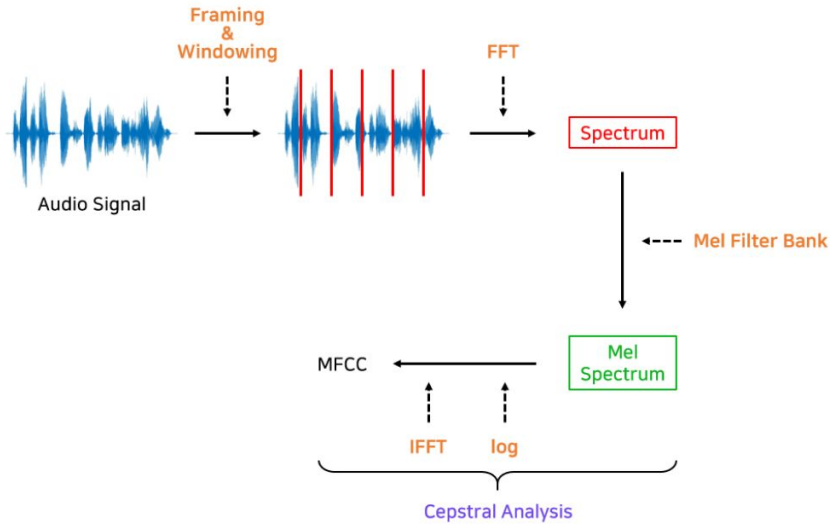

< Sample rate >

## Sample Rate

- 연속적 신호에서 얻어진 단위시간(주로 초)당 샘플링 횟수이다.
- 단위는 **Hz**를 사용하며 보통 **CD**에서는 **44.1kHz**의 **Sample Rate**를 사용한다.

# MFCC (Mel-Frequency Cepstral Coefficient)

- **MFCC**는 오디오 신호에서 추출할 수 있는 **feature**로, 소리의 고유한 특징을 나타내는 수치이다.
- 주로 음성 인식, 화자 인식, 음성 합성, 음악 장르 분류 등 오디오 도메인의 문제를 해결하는 데 사용된다.




< **MFCC**의 추출 과정 >

**MFCC** 의 기술적 이해

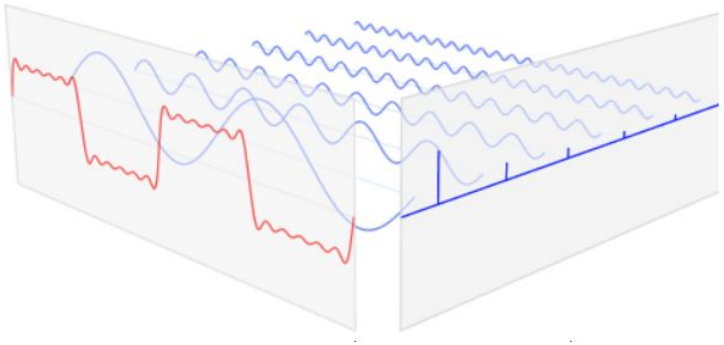
- Mel Spectrum(멜 스펙트럼)에서 Cepstral(켑스트럴) 분석을 통해 추출된 값

**MFCC** 의 추출 과정

1. 오디오 신호를 프레임별(보통 **20ms - 40ms**)로 나누어 **FFT**를 적용해 Spectrum을 구한다.
2. Spectrum에 **Mel Filter Bank**를 적용해 Mel Spectrum을 구한다.
3. Mel Spectrum에 Cepstral 분석을 적용해 **MFCC**를 구한다.

# FFT(Fast Fourier Transform : 고속 푸리에 변환)

- 신호를 주파수 성분으로 변환하는 알고리즘으로, 기존의 ***이산 푸리에 변환(DFT)***[1]을 더욱 빠르게 수행할 수 있도록 최적화한 알고리즘.
- 주파수(가로축)에 따른 음압(세로축)의 표현, 즉 주파수 영역(frequency domain)의 표현 (Spectrum)



< **Fourier Transform** : 푸리에 변환 >

***1)*** *이산 푸리에 변환* ***(DFT, Discrete Fourier Transform)***

- 시간영역의 ***이산 신호***[2]에 대한 푸리에 변환

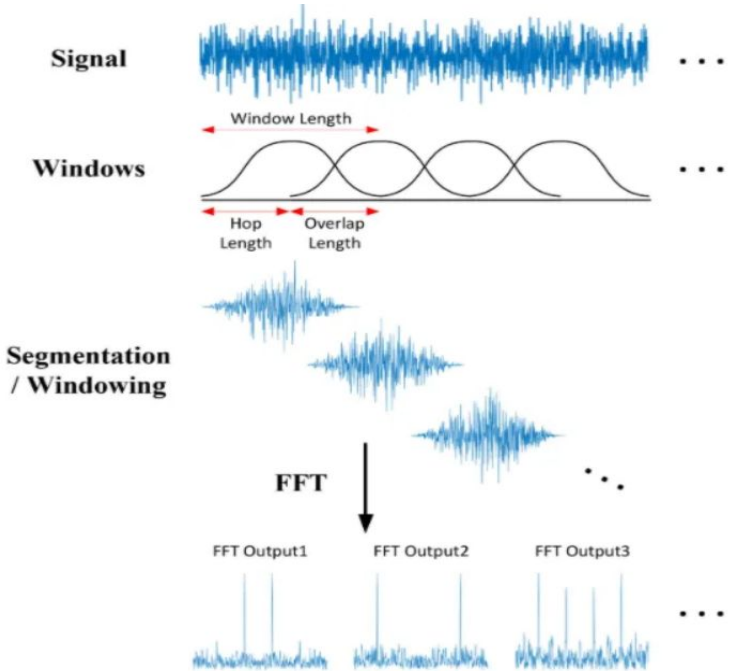
***2)*** *이산신호* ***(Discrete)***

- 특정한 시각에서만 띄엄띄엄 값을 갖는 신호
- 이산신호는 연속신호를 샘플링한 신호이고, 디지털 신호는 양자화까지한 신호

# STFT (Short Time Fourier Transform, 국소 푸리에 변환)

- 시간에 따라 변화하는 **Speech signal**의 경우 **Fourier Transform**은 그 특성을 잘 표현하지 못하는 경우가 많아서

  시간별로 짧게 잘라**(window)**서 **Fourier Transform**을 적용하는 것이다




### Window function

- **각각의 프레임에 특정 함수를 적용해 경계를 스무딩하는 기법**

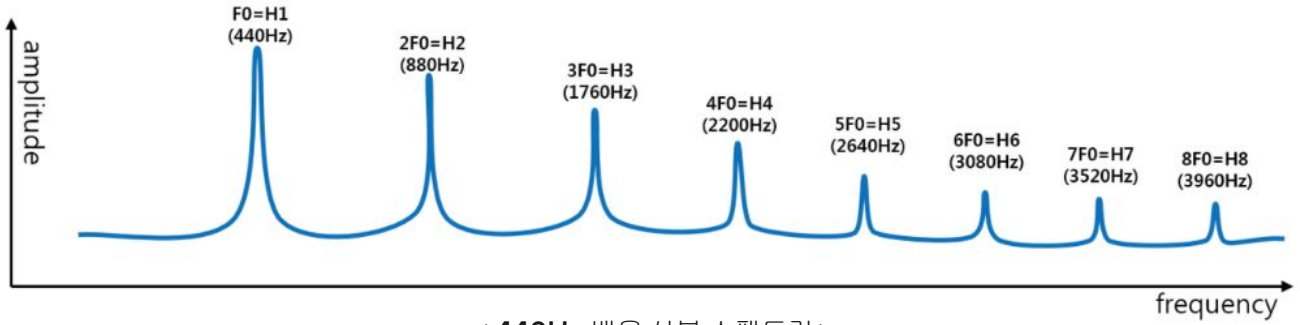
### Hop Length

- **Frame offset**으로 각각의 프레임이 얼마나 겹치게 할지 정할 수 있다.

# 배음(harmonics) 구조

- 악기 소리나 사람의 음성은 한 가지의 주파수만으로 구성되지 않고, 일반적으로 배음 구조를 가지고 있다.
- 기본 주파수(fundamental frequency)와 함께 기본 주파수의 정수배인 배음(harmonics)들로 구성된다.
- 배음 구조는 악기나 성대의 구조에 따라 달라지며 배음 구조의 차이가 음색의 차이를 만든다.
- 예를 들어 우리가 피아노 건반에서 **4**옥타브 '라'**(440Hz)** 음을 연주했다면 그 소리는 기본 주파수인 **440Hz**뿐만 아니라 그 정수배인 **880Hz**, 그리고 그다음 배음들까지 포함하고 있다.

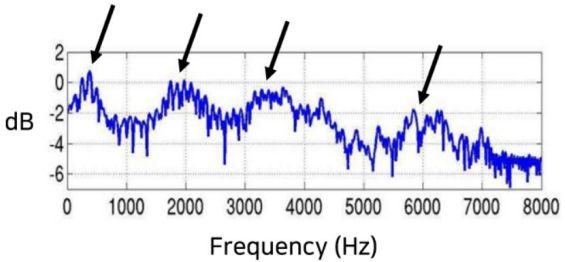
**※ 즉, 배음 구조를 유추해낼 수 있다면 소리의 고유한 특징을 찾아낼 수 있다.**



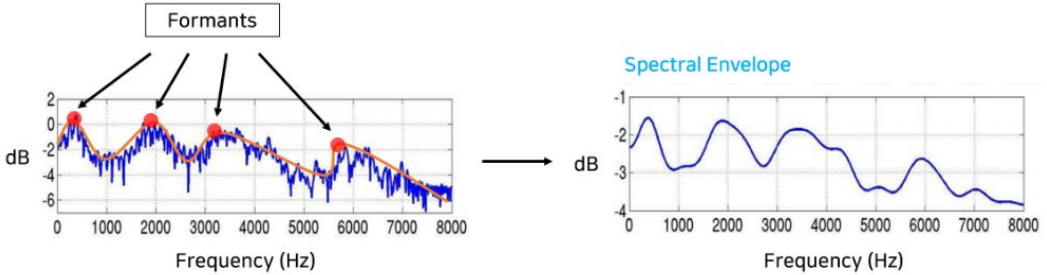

< **440Hz** 배음 성분 스펙트럼 >

# 캡스트럼 분석(Cepstral Analysis)

- 신호의 스펙트럼 표현을 기반으로 하는 음성 분석 방법
- 스펙트럼상에서 배음(harmonics)의 정도를 로그푸리에변환을통해 하나의 정점(peak)으로 표시해준다.




< Spectrum >

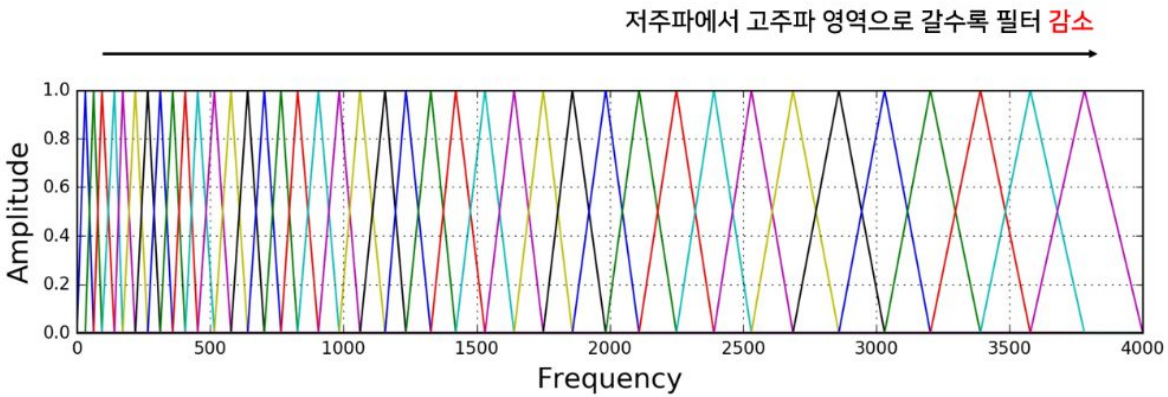



< 스펙트럼 포락선(Spectral Envelope) 분리 >

- 화살표로 지목된 피크(peak)들은 신호에서 지배적인 주파수 영역을 가리킨다.
- 이 피크들을 포먼트(Formants)라하며, 소리가 공명되는 특정 주파수 대역이다.
- 포먼트는 배음(harmonics)과 만나 소리를 풍성하게 혹은 선명하게 만드는 필터 역할을 한다.
- **MFCC**는 둘을 분리하는 과정에서 도출된다.
- 이때 사용하는 수학과 알고리즘이 **log**와 **IFFT(Inverse FFT : 역 고속 푸리에 변환)**이다.

# Mel Spectrum

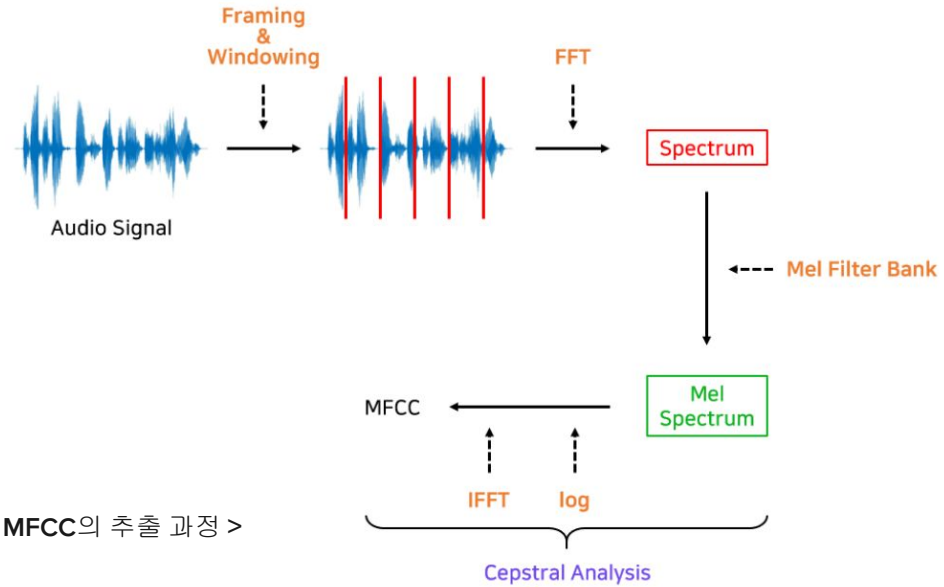
- **MFCC**는 **Spectrum**이 아닌 **Mel Spectrum**에서 **Cepstral** 분석으로 추출한다.
- 사람의 청각기관은 고주파수**(high frequency)** 보다 저주파수**(low frequency)** 대역에서 더 민감하다.
- 사람의 이런 특성을 반영해 물리적인 주파수와 실제 사람이 인식하는 주파수의 관계를 표현한 것이 **Mel Scale**(멜 스케일)이다.
- 이 **Mel Scale**에 기반한 **Filter Bank**를 **Spectrum**에 적용하여 도출해낸 것이 **Mel Spectrum**이다.
- **Mel Scale**은 **Filter Bank**를 나눌 때 어떤 간격으로 나눠야 하는지 알려주는 역할을 한다.




< Mel Scale Filter >

# MFCC (Mel-Frequency Cepstral Coefficient) 정리

- 배음 구조를 유추해낼 수 있다면 소리의 고유한 특징을 찾아낼 수 있다.
- **MFCC**는 오디오 신호에서 **Mel Spectrum**(멜 스펙트럼)에서 **Cepstral**(켑스트럴) 분석을 통해 배음의 구조를 알 수 있다.




< **MFCC**의 추출 과정 >

03

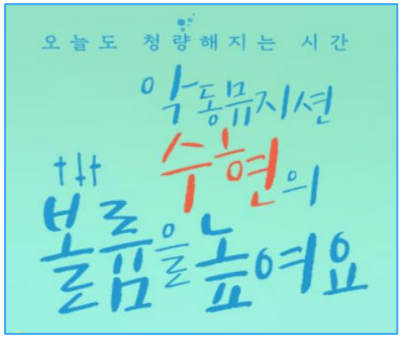
# 구현 방법

# 데이터 수집

- 최대한 음질이 비슷하고, 특정인물의 목소리가 주로 나오는 오디오 파일 수집 **(10명)**
- 오디오 편집 툴을 이용하여 배경음악과 다른 사람과 겹치는 음성 제거
- 파이썬 오디오 라이브러리**(Scipy)**를 이용하여 공백에 해당하는 부분 제거 후, **WAV** 파일로 저장

  **WAV** 파일은 파형 오디오 파일 형식, 압축되지 않은 무손실 오디오 파일

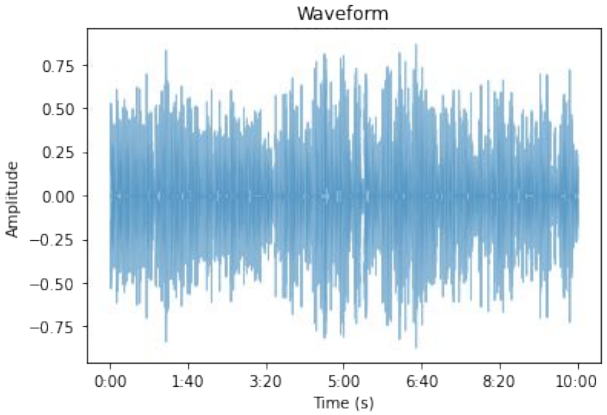





# 데이터 전처리 1

- **Sample rate**을 **16,000** 으로 **Data Load (**사람의 목소리는 대부분 **16,000 Hz** 안에 포함**)**
- 앞에서 수집 및 편집한 **10**명의 음성데이터를 **10**분**(600**초**)**로 분리
- **shape (9,600,000, )** 데이터 **10**개 생성

  **row = time x sample rate = 600 x 16,000 = 9,600,000**




< 음성 데이터 시각화 >

```
김이나    shape : (9600000,)
노홍철    shape : (9600000,)
박명수    shape : (9600000,)
아이유    shape : (9600000,)
유인나    shape : (9600000,)
유재석    shape : (9600000,)
이수현    shape : (9600000,)
정준하    shape : (9600000,)
정형돈    shape : (9600000,)
하하      shape : (9600000,)
```

< code 및 데이터 shape >

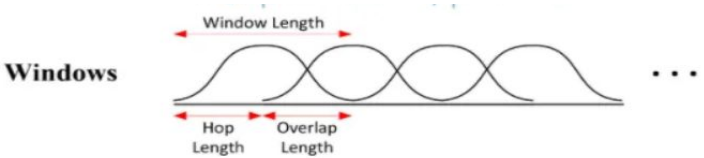
```python
import glob
import os.path
import librosa

dataset = []
labelset = {"유재석":0, "박명수":1, "정형돈":2, "노홍철":3, "하하":4,
            "정준하":5, "유인나":6, "아이유":7, "김이나":8, "이수현":9}
labelset_re = {v:k for k,v in labelset.items()}

global sample_rate
sample_rate = 16000

files = glob.glob("./data/edit_voice_only/*")
for x in files:
    file_name = os.path.basename(x)
    if file_name.startswith("edited_"):
        y, sr = librosa.load(x, sr=sample_rate)

        # 10분 (600초) 로 데이터 쪼개기
        y = y[:9600000]

        # label
        speaker = os.path.splitext(file_name)[0].replace("edited_", "")
        label = labelset[speaker]

        dataset.append([y, label])

for i in range(len(dataset)):
    print(f"{labelset_re[dataset[i][1]]}\t shape : {dataset[i][0].shape}")
```

# 데이터 전처리 2

- 0.5초(500ms)당 **MFCC**를 사용하여 특징벡터 **40**개 추출
- **shape (12000, 51, 40)** 데이터 생성 **(row** 당 **0.5**초에 해당**)**

## Argument

- **y** : audio data
- **sr** : sampling rate
- **n_mfcc** : return 될 mfcc의 개수를 정해주는 파라미터.
- **n_fft** : frame의 length를 결정하는 파라미터.
- **hop_length** : 윈도우 길이.



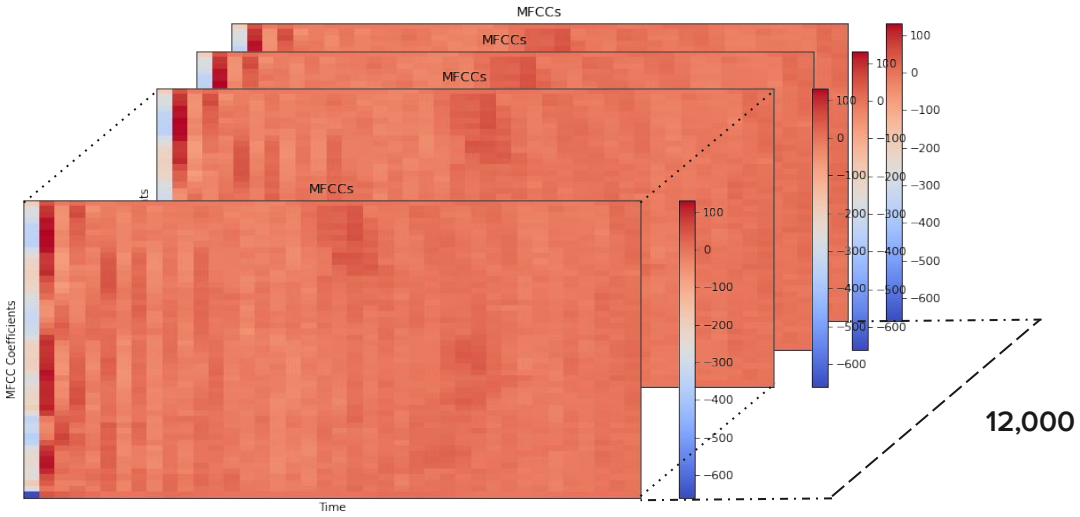

< Windows >

```python
def preprocess_dataset(df):
    mfccs = []
    labels = []
    for index, row in df.iterrows():
        for i in range(0, 1200):
            start = int(i*(sample_rate/2))
            end = int((i+1)*(sample_rate/2))
            extracted_features = librosa.feature.mfcc(y=row['data'][start:end],
                                                      sr=sample_rate,
                                                      n_mfcc=40,
                                                      hop_length=int(sample_rate*0.01),
                                                      n_fft=int(sample_rate*0.02)).T
            mfccs.append(extracted_features)
            labels.append( row['label'] )

    return np.array(mfccs), np.array(labels)

mfccs, labels = preprocess_dataset(df)

print(mfccs.shape)
print(labels.shape)
```

```
(12000, 51, 40)
(12000,)
```

< code 및 데이터 shape >

# 데이터 구성




( 51, 40 )

< Data set >

# 합성곱 신경망 (CNN : Convolutional Neural Network) 구현

- Train set, Validation set, Test set을 6:2:2로 데이터를 분할 및 reshape
  - Train set : (7200, 51, 40, 1)
  - Validation set : (2400, 51, 40, 1)
  - Test set : (2400, 51, 40, 1)
- Convolution Layer 3개, Pooling Layer 2개를 쌓아 특성 추출을 하고, Fully Connected Layer에 연결하여 분류
- Loss Function은 'categorical_crossentropy', Optimizer는 'rmsprop'
- Epochs는 300으로 딥러닝한 결과
  - Train set : 99.76 %
  - Validation set : 90.91 %
  - Test set : 90.62 %

```
훈련셋    :  0.02911156415939331 0.9976388812065125
검증셋    :  5.025374412536621 0.909166693687439
테스트셋 :  5.889211654663086 0.90625
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_4 (InputLayer) | [(None, 51, 40, 1)] | 0 |
| conv2d_12 (Conv2D) | (None, 50, 39, 10) | 50 |
| max_pooling2d_9 (MaxPooling 2D) | (None, 25, 19, 10) | 0 |
| conv2d_13 (Conv2D) | (None, 24, 18, 100) | 4100 |
| max_pooling2d_10 (MaxPoolin g2D) | (None, 12, 9, 100) | 0 |
| conv2d_14 (Conv2D) | (None, 11, 8, 200) | 80200 |
| max_pooling2d_11 (MaxPoolin g2D) | (None, 5, 4, 200) | 0 |
| conv2d_15 (Conv2D) | (None, 4, 3, 300) | 240300 |
| flatten_3 (Flatten) | (None, 3600) | 0 |
| dense_9 (Dense) | (None, 64) | 230464 |
| dense_10 (Dense) | (None, 32) | 2080 |
| dense_11 (Dense) | (None, 10) | 330 |

# 04

# 웹 구현

# Flask를 이용한 웹 구현

0:00 / 0:47

05

# 문제점 및 개선 방안

# 문제점 및 개선 방안

## 문제점

- 학습데이터에 지나치게 최적화된 과대적합이 생겼다.
- 입술소리, 쩝소리, 숨소리 등의 잡음과 "아", "어", “그” 등의 간투어에서 예측이 잘 안되는 모습이 보인다.
- 겹치는 소리가 있는 경우, 예측률이 떨어진다.

## 개선방안

- 학습데이터 수를 높인다.
- **sampling rate, mfcc** 특성 개수, **frame length, hop length** 등의 인수**(argument)**의 최적화한다.
- 시계열 예측 모델을 병행하여 훈련한다.
- 문장 중간에 예측 결과가 달라지지 않도록 **Speech to Text** 를 통해 텍스트와 병행하여 훈련한다.
- 비지도학습인 **Speaker Diarization**을 통해 성능을 높일 수 있다.

# Thanks!

Do you have any questions?