# 음향기기 오디오 검사 모델 (LSTM, CNN)

오디오 데이터(wav) 학습을 통한 자동 검사 모델 만들기


2023. 09. 20

박성현

## 분석 배경

❑ **문제 발생 장소** : 음향 기기 제조업체

❑ **문제 발생 공정** : 오디오 검사 공정 (노이즈, 무음 등 불량 검출)

❑ **문제 발생 내용** :

1) 현재 사람이 직접 청각으로 검사한다.

2) 최소 3개월 이상의 숙련공 필요 (검사 인력 한계에 의한 생산 Capa 문제),

3) 스피커를 귀에 대고 고주파음을 하루종일 듣다보면 검사자의 청력에 문제 생길 수 있음

4) 검사자의 실수로 불량을 양품으로 판정할 수 있음

❑ **전체 공정 흐름도**


## 분석 목적

➢ 양품과 불량의 오디오파일(wav) 학습을 통한 자동 검사 모델 개발

## 분석 효과

➢ 정상 소리와 불량 시료의 미세한 차이는 일반인은 구분하기 어려운 정도이나 모델 학습을 통해 자동검사 가능

➢ 검사자의 교육 기간이 필요 없고, 검사 시간에 제약이 없어 생산 Capa 증가

➢ 사람의 실수가 발생 되지 않아 불량 검출력 향상

# 학습 결과

## ❑ 결과 요약

- 가장 학습이 잘 되는 데이터셋 : STFT_dB (STFT의 진폭을 데시벨로 스케일링 한 데이터)
- 베스트 모델 : **LSTM 모델 (Long Short-Term Memory)**
- 베스트 모델 정확도 : **99.0%**

## ❑ 모델별 평가 결과

| DataSet | Data Structure | Model | Accuracy | Precision | Recall | F1 Score | AUC |
|---|---|---|---|---|---|---|---|
| **MFCC** | **2D** | **LSTM** | 94.9% | 77.8% | 93.3% | 84.9% | 98.7% |
| **STFT_dB** | **2D** | **LSTM** | 99.0% | 100.0% | 93.3% | 96.7% | 99.6% |
| **Waveform** | **1D** | **CNN** | 96.9% | 100.0% | 80.0% | 88.9% | 100.0% |

- RNN 모델의 Vanishing Gradient 문제를 개선한 LSTM 모델은 시계열 데이터 분석에 효과적이다.
- 오디오데이터는 시간에 따라 변하는 연속적인 신호로서 시계열 데이터이다.

## ❑ 모델 보완 필요

- 데이터 개수가 348개 (양품:55개, 불량:293개)로 모델을 학습시키기에는 부족함
- 학습 데이터 개수가 늘어나면 모델 정확도가 100% 에 가까워 질 것으로 예상

## ❑ 실제 공정에 적용 방법

- 자동 검사 설비를 초기에 100% 신뢰할 수 없기 때문에 기존 검사자와 검사를 병행하며 모델을 개선, 점차 검사자의 검사 비율을 줄여가는 방식으로 공정에 적용 가능

[실제 공정 적용 예시]

## 분석 환경

❑ **사용 언어** : Python3

❑ **사용 패키지** : numpy, pandas, matplotlib, sklearn, tensorflow, librosa

❑ **분석 환경** : [CPU] Apple M2, [RAM] 8GB, [GPU] T4 (colab)

## 학습 데이터셋 형태 및 개수

❑ **데이터 형태** : 비정형 오디오 파일(wav)

❑ **수집 장소** : 음향기기 제조사의 오디오 검사 공정

❑ **데이터 개수** : 348개 (양품:55개, 불량:293개)

❑ **데이터 용량** : 1.14 GB

❑ **Sample data 정보**

- 재생 시간 : 약 3초
- Sample rate : 192 kHz
- 용량 : 3.5MB

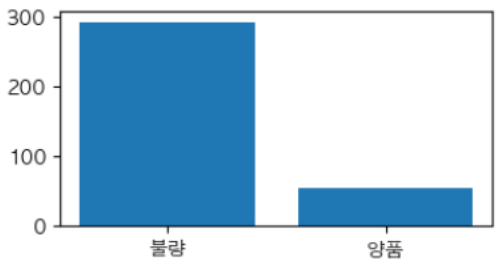
# 데이터 분석(EDA) 및 전처리

## ❑ 파일명 저장 및 라벨 클래스 균형 확인

```
# 파일명 저장, 개수 확인

import glob

abnorm_files = glob.glob('/Users/park/play/data/audio/abnorm/*.wav')
normal_files = glob.glob('/Users/park/play/data/audio/normal/*.wav')

all_files = abnorm_files + normal_files
print(f'전체 파일 개수 : {len(all_files)}')
print(f'불량품 개수 : {len(abnorm_files)}')
print(f'양품 개수 : {len(normal_files)}')

plt.figure(figsize=(4,2))
plt.bar(['불량', '양품'], [len(abnorm_files), len(normal_files)])
plt.show()
```

✓ 0.0s

```
전체 파일 개수 : 348
불량품 개수 : 293
양품 개수 : 55
```




## ❑ 파일 리스트 정렬, 라벨 리스트 만들기

```
# 모든 파일 리스트 정렬
all_files.sort()
all_files
```

✓ 0.0s

```
['/Users/park/play/data/audio/abnorm/abnorm_1.wav',
 '/Users/park/play/data/audio/abnorm/abnorm_10.wav',
 '/Users/park/play/data/audio/abnorm/abnorm_100.wav',
 '/Users/park/play/data/audio/abnorm/abnorm_101.wav',
 '/Users/park/play/data/audio/abnorm/abnorm_102.wav',
 '/Users/park/play/data/audio/abnorm/abnorm_103.wav',
 ...
 '/Users/park/play/data/audio/normal/normal_55.wav',
 '/Users/park/play/data/audio/normal/normal_6.wav',
 '/Users/park/play/data/audio/normal/normal_7.wav',
 '/Users/park/play/data/audio/normal/normal_8.wav',
 '/Users/park/play/data/audio/normal/normal_9.wav']
```

```
# 라벨 리스트 만들기
label_0 = np.array([0]*len(abnorm_files))
label_1 = np.array([1]*len(normal_files))
label = np.append(label_0, label_1)
```

0.0s

- 불량 파일이 앞 쪽에 나열,
- 양품 파일이 뒷 쪽에 나열.
- 불량과 양품의 개수만큼 라벨 0과 1 만들기

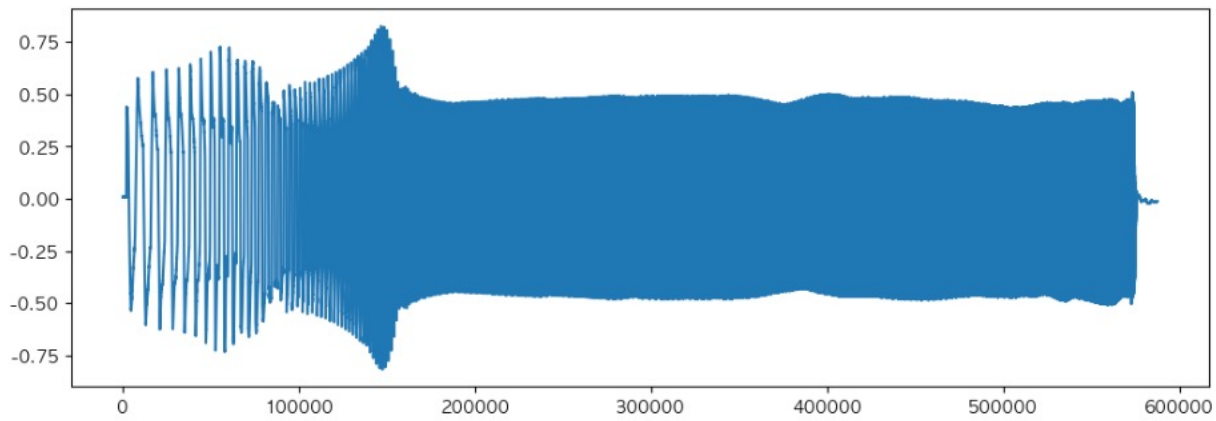
# 데이터 분석(EDA) 및 전처리

## ❑ 원본 오디오 파일을 Waveform 형식으로 변환, 시각화

```
# waveform 파일로 변환하기 (1D data)
# - mono 오디오 형식으로 처리함 (stereo 두 채널의 평균값)
# - -1~1 사이의 값으로 반환
sample_data = normal_files[0] # 데이터 구조를 확인하기 위한 샘플 데이터
wave, sr = librosa.load(sample_data, sr=sample_rate)
print(f'sample rate : {sr}') # 1초당 샘플링 되는 횟수
print(f'waveform shape : {wave.shape}')

# waveform 시각화
# - x : sampling rate의 샘플 번호
# - y : 진폭 (주파수 아님, 주파수를 보고싶다면 stft, mfcc로 변환)
plt.figure(figsize=(12,4))
plt.plot(wave)
plt.show()
```

✓ 0.2s MagicPyt

```
sample rate : 192000
waveform shape : (587536,)
```




- X축 : 시간에 따른 Sample (1초당 192000개 이므로 이 파일은 3초가 약간 넘는다)
- Y축 : 진폭 (Stereo 두 채널의 평균값으로 -1 ~ 1 사이의 값으로 변환 됨)

- Waveform은 1D 데이터로 1D Convolution layer를 사용하여 학습 할 수 있음

# 데이터 분석(EDA) 및 전처리

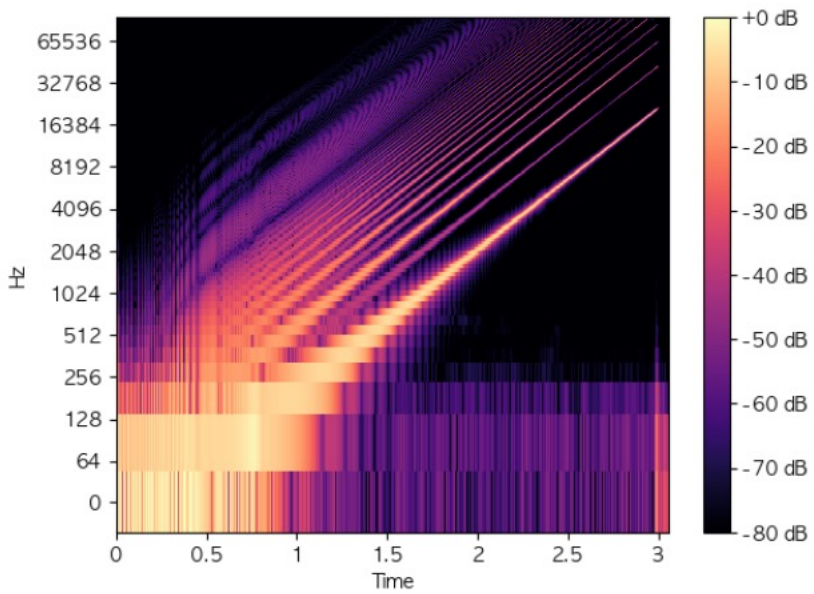
## ❑ 원본 오디오 파일을 STFT 형식으로 변환, 시각화 (Sort Time Fourier Transform)

```
# stft(wavetogram)은 복소수 형태 (실수+허수)
# - 시각화를 위해서는 np.abs()처리 -> specshow() 사용 시각화
# - (1025, 1148) : 1025개의 주파수 bin, 1148개의 시간 frame
# -> stft를 구하는 목적 : 시간에 따른 주파수 성분의 변화를 분석, 시각화 하기 위함
stft = librosa.stft(wave)
```

```
# librosa.amplitude_to_db() : stft의 진폭을 데시벨로 변환(스케일링)
# -> 인간의 청각 특성과 맞추기 위해

stft_db = librosa.amplitude_to_db(np.abs(stft), ref=np.max)
print(stft_db.shape)
librosa.display.specshow(stft_db, sr=sample_rate, x_axis='time', y_axis='log')
plt.colorbar(format="%+2.0f dB")
plt.show()
```

✓ 0.3s

```
(1025, 1148)
```




- Waveform -> STFT -> STFT_dB 형식으로 변환
- STFT : 시간에 따른 주파수 성분의 특성 변화를 분석 할 수 있음
- STFT_dB : STFT의 진폭을 데시벨로 변환(스케일링) – LSTM 학습에 사용

- 인간의 청각은 주파수에 대해 로그 스케일로 반응 (주파수 차이를 절대적인 헤르츠 값보다는 상대적인 비율로 인식)
- 이러한 인간의 청각 특성을 반영하기 위해 주파수 축을 로그 스케일로 표시
- 로그 스케일에서는 낮은 주파수 대역이 더 넓게 표시되고, 높은 주파수 대역은 더 좁게 표시

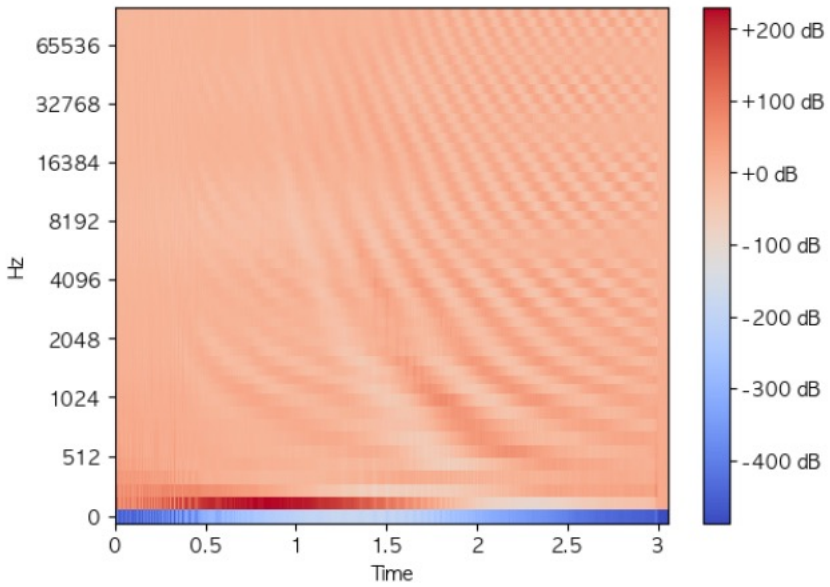
# 데이터 분석(EDA) 및 전처리

## ❑ 원본 오디오 파일을 MFCC 형식으로 변환, 시각화 (Mel-Frequency Cepstral Cefficients)

```
# MFCC는 음성 및 오디오 데이터의 복잡성을 줄이면서도 중요한 특성을 효과적으로 캡처하여,
# 다양한 오디오 처리 작업에 적합한 특징 벡터를 제공
# Mel 스케일링 : 인간의 청각 특성을 반영
# log 스케일링 : 주로 수학적 특성 또는 데이터 분포를 조정하기 위해 사용
mfcc = librosa.feature.mfcc(y=wave, sr=sample_rate, n_mfcc=50)
print(mfcc.shape)
librosa.display.specshow(mfcc, sr=sample_rate, x_axis='time', y_axis='mel')
plt.colorbar(format="%+2.0f dB")
plt.show()
```

✓ 0.2s

```
(50, 1148)
```




- MFCC는 오디오 신호의 특징을 n_mfcc 개수만큼 추출 - LSTM 모델 학습에 사용

- STFT의 shape : (1025, 1148)
  - 1025개의 주파수 구간, 1148개의 time step
- MFCC의 shape : (50, 1148) n_mfcc=50
  - 50개의 MFCC 계수, 1148개의 time step

# 데이터 분석(EDA) 및 전처리

## ❑ 전체 파일을 4가지 형식으로 변환하여 리스트 만들기 (모델 학습에 사용)

(Waveform, STFT, STFT_dB, MFCC)

```
# wave_list 만들기
wave_list = []
for file in all_files:
    wave, sr = librosa.load(file, sr=sample_rate)
    wave_list.append(wave)

wave_list = np.array(wave_list)
print(wave_list.shape)
```

✓ 4.5s

Waveform type : wave_list

```
(348, 587536)
```

```
# stft_list, db_list 만들기
stft_list = []
db_list = []

for file in all_files:
    wave, sr = librosa.load(file, sr=sample_rate)
    stft = librosa.stft(wave)
    stft_list.append(stft)
    stft_db = librosa.amplitude_to_db(np.abs(stft), ref=np.max)
    db_list.append(stft_db)

stft_list = np.array(stft_list)
db_list = np.array(db_list)
print(stft_list.shape)
print(db_list.shape)
```

✓ 58.4s

STFT type : stft_list

STFT_dB type : db_list

```
(348, 1025, 1148)
(348, 1025, 1148)
```

```
# mfcc_list 만들기
mfcc_list = []

for file in all_files:
    wave, sr = librosa.load(file, sr=sample_rate)
    mfcc = librosa.feature.mfcc(y=wave, sr=sample_rate, n_mfcc=50)
    mfcc_list.append(mfcc)

mfcc_list = np.array(mfcc_list)
print(mfcc_list.shape)
```

✓ 12.3s

MFCC type : mfcc_list

```
(348, 50, 1148)
```

# 학습 모델 구축 및 예측

## ❑ 데이터셋 분할 (train, test, val)

- 4개의 데이터셋 중 학습에 사용할 데이터셋 선택
- MFCC : 스케일링 필요 없음 (스케일링 하면 조기 과적합 발생, 데이터 크기가 STFT에 비해 작음)
- STFT_dB : 스케일링 필요 (스케일링 안하면 학습이 안됨. 데이터 크기와 값의 범위가 크기 때문)
- STFT : 복소수 형태로 학습은 안되나, STFT_dB 형태로 변환하여 학습 가능
- Waveform : 1D 형태로 CNN 으로 학습 가능

```
# X = mfcc_list
X = db_list
# X = stft_list # (복소수 형태는 학습에 제한. db_list 형태로 변환하여 학습)
# X = wave_list
y = label

# 데이터 스케일링 (mfcc_list:불필요(조기 과적합 발생), db_list:필요)
samples, features, times = X.shape
X_trans = X.reshape(samples, -1)
scaler = StandardScaler()
X_scaled_trans = scaler.fit_transform(X_trans)
X = X_scaled_trans.reshape(samples, features, times)

print(f'X.shape : {X.shape}')
print(f'y.shape : {y.shape}')
print(f'클래스 0, 1 : {np.bincount(y)}')
print(f'클래스 1 비중 : {np.bincount(y)[1] / np.bincount(y)[0]:.2f}\n')

# train, test 데이터셋 분할 후 train에서 다시 val 데이터셋을 분할 (라벨 클래스 비율 고정)
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.28, stratify=y)
X_train, X_val, y_train, y_val = \
    train_test_split(X_train, y_train, test_size=0.28, stratify=y_train)
```

4개의 데이터셋 중 선택하여 학습에 사용

✓ 9.6s

```
X.shape : (348, 1025, 1148)
y.shape : (348,)
클래스 0, 1 : [293  55]
클래스 1 비중 : 0.19
```

- Train, test 분할 시 비율 0.72 : 0.28
- Train, val 분할 시 비율 0.72 : 0.28

- 라벨 클래스 0과 1의 비율은 약 8:2
- 데이터셋 분할 시 라벨 클래스의 비율이 유지되도록 stratify 인자로 고정

# 학습 모델 구축 및 예측

## ❑ MFCC 데이터셋 학습 및 예측 – LSTM 모델 (정확도 : 94.9%)

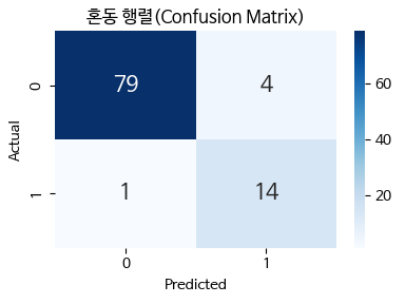
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam

drops = 0.12 # 과적합 방지를 위해 drop
model = Sequential()
model.add(Dropout(drops)) # LSTM layer에 input 되기 전 drop
model.add(LSTM(64, input_shape=(X_train.shape[1], X_train.shape[2]),
               return_sequences=True))
# 추가 lstm 레이어가 있을 때는 return_sequencese 설정이 있어야하고, 추가 레이어 없으면 삭제해야함
model.add(LSTM(16))
model.add(Dense(2, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=Adam(learning_rate=0.00005), # 학습률 조정
              metrics=['sparse_categorical_accuracy'])

callbacks = [ModelCheckpoint('best_model.h5', save_best_only=True,
             monitor='val_loss', mode='min')] # val_loss 값 기준으로 베스트 모델 저장

# LSTM 모델 학습
history = model.fit(X_train, y_train, # 학습데이터, 라벨
                    batch_size=4, # batch size
                    validation_data=(X_val, y_val), # 이미 만들어진 검증 데이터
                    epochs=5, # 학습 반복 횟수
                    callbacks=callbacks) # val_loss 값이 낮아질 때만 모델 저장

# 모델 평가
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

## 학습데이터, 검증데이터의 정확도, loss 그래프 그리기
plt.figure(figsize=(10, 5))
plt.plot(history.history['sparse_categorical_accuracy'], label='Training accuracy')
plt.plot(history.history['val_sparse_categorical_accuracy'], label='Validation accuracy')
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.ylabel('Accuracy, Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()

# 모델 구조 / 파라미터 개수 확인
model.summary()
```

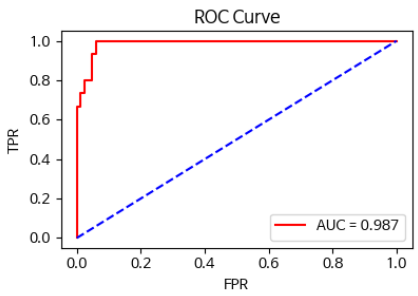
# 학습 모델 구축 및 예측

## ❑ MFCC 데이터셋 학습 및 예측 – LSTM 모델 (정확도 : 94.9%)

[Metrics]

- Accuracy: 94.90%
- Precision: 77.78%
- Recall: 93.33%
- F1 Score: 84.85%
- AUC: 98.71%



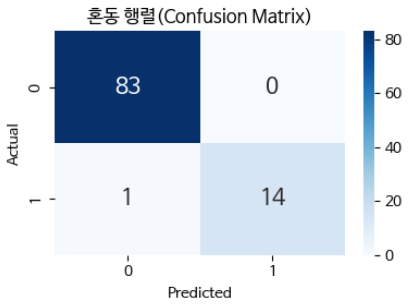

```
# 클래스별 예측 확률, 예측 값
probs = model.predict(X_test)[:, 1] # 예측 결과가 클래스 1에 속할 확률
y_pred = np.argmax(model.predict(X_test), axis=1) # 예측 값

# 평가지표 계산
accuracy = accuracy_score(y_test, y_pred) # 정확도
precision = precision_score(y_test, y_pred) # 정밀도: 양성이라고 예측한것중 실제 양성의 비율
recall = recall_score(y_test, y_pred) # 재현율(민감도): 실제 양성 중 양성으로 예측된 비율
f1 = f1_score(y_test, y_pred) # f1: 2 * (정밀도*재현율)/(정밀도+재현율)
auc_score = roc_auc_score(y_test, probs)  # AUC 계산 (ROC 곡선 아래의 면적)

# 평가지표 출력
print(f"Metrics")
print(f" - Accuracy: {round(accuracy*100,2)}%")
print(f" - Precision: {round(precision*100,2)}%")
print(f" - Recall: {round(recall*100,2)}%")
print(f" - F1 Score: {round(f1*100,2)}%")
print(f" - AUC: {round(auc_score*100,2)}%")

## 모델 혼동행렬과 ROC Curve 그리기
fig, axs = plt.subplots(1, 2, figsize=(8,3))

# 혼동 행렬(Confusion Matrix)
confusion = confusion_matrix(y_test, y_pred)
# 히트맵으로 시각화
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues",
            xticklabels=['0', '1'], yticklabels=['0', '1'],
            annot_kws={"size": 15}, ax=axs[0])
axs[0].set_ylabel('Actual')
axs[0].set_xlabel('Predicted')
axs[0].set_title('혼동 행렬(Confusion Matrix)')

# ROC 그래프 그리기
fpr, tpr, thresholds = roc_curve(y_test, probs) #fpr, tpr, thresholds 값 변수에 할당
axs[1].plot(fpr, tpr, color='red', label=f'AUC = {auc_score:.3f}') # ROC Curve 그리기
axs[1].plot([0, 1], [0, 1], color='blue', linestyle='--')
axs[1].set_title('ROC Curve')
axs[1].set_xlabel('FPR')
axs[1].set_ylabel('TPR')
axs[1].legend(loc="lower right")
```

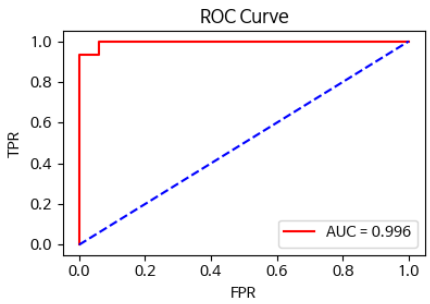
# 학습 모델 구축 및 예측

## ❑ STFT_dB 데이터셋 학습 및 예측 – LSTM 모델 (정확도 : 98.98%)

[Metrics]

- Accuracy: 98.98%
- Precision: 100.0%
- Recall: 93.33%
- F1 Score: 96.55%
- AUC: 99.60%




- 학습 및 평가 코드는 MFCC 와 동일하므로 생략

# 학습 모델 구축 및 예측

## ❑ Waveform 데이터셋 학습 및 예측 – CNN 모델 (정확도 : 96.94%)

```
## CNN 학습 및 예측 결과

# CNN 학습을 위해 데이터 shape 변환
X_train_exp = np.expand_dims(X_train, -1) # X_train 배열의 마지막 차원에 새로운 축을 추가
X_test_exp = np.expand_dims(X_test, -1) # X_test 배열의 마지막 차원에 새로운 축을 추가

# 하이퍼파라미터 설정
epoch = 5 # 학습 반복 횟수
input_dim = X_train.shape[1]
act = 'LeakyReLU' # 활성화 함수
opt = 'adam' # optimizer
filters = 1 # Conv1D Layer의 filter수
batch = 4 # batch size
kernel = 3 # filter의 kernel size
drops = 0 # drop 비율 - drop 안시키는게 loss가 더 잘나오고, 과적합 없음

# 모델 정의(구조)
def make_cnn_model():
    model = Sequential()
    # Conv1D Layer
    model.add(Conv1D(filters=filters, kernel_size=kernel, activation=act,
                     padding='same', input_shape=(input_dim, 1)))
    model.add(BatchNormalization()) # batch 데이터의 분포를 정규화
    model.add(MaxPooling1D(pool_size=16))  # MaxPooling1D Layer

    # Flatten Layer
    model.add(Flatten()) # filter에 의해 만들어진 다차원의 데이터를 1차원으로 변환
    model.add(BatchNormalization()) # batch 데이터의 분포를 정규화
    model.add(Dropout(drops))

    # Dense Layer
    model.add(Dense(16, activation=act)) # Dens Layer 노드 수, 활성함수 설정
    model.add(BatchNormalization())# batch 데이터의 분포를 정규화

    # Output Layer
    model.add(Dense(2, activation='softmax')) # 2개 클래스의 확률을 출력
    return model

cnn_model = make_cnn_model() # 모델 초기화

# 모델 컴파일
cnn_model.compile(optimizer=opt, # optimizer: 학습 최적화 알고리즘
                  loss='sparse_categorical_crossentropy', # 사용할 손실 함수
                  metrics=['sparse_categorical_accuracy']) # 모델의 평가 지표

# monitor 지표를 기준으로 베스트 모델을 저장
callbacks = [ModelCheckpoint('best_model.h5', save_best_only=True,
                             monitor='val_loss', mode='min')]

history = cnn_model.fit(X_train_exp, y_train, # 학습데이터, 라벨
                        batch_size=batch, # batch 크기 지정
                        validation_data=(X_val, y_val), # 검증용 데이터의 비율 지정
                        epochs=epoch, # 학습 반복 횟수
                        callbacks=callbacks) # 검증 정확도가 올라갈때만 모델 저장
```

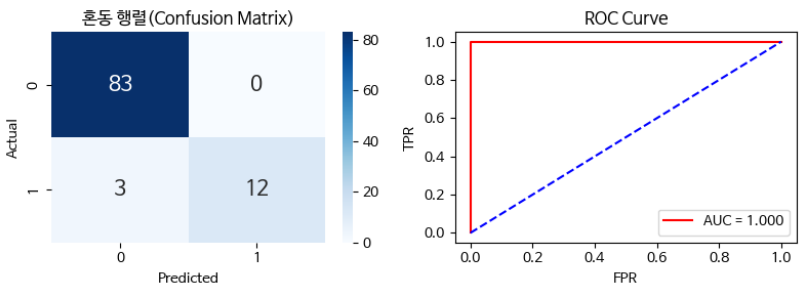
# 학습 모델 구축 및 예측

## ❑ Waveform 데이터셋 학습 및 예측 – CNN 모델 (정확도 : 96.94%)

```
# 훈련/검증 정확도 최대값과 해당 인덱스를 변수에 할당
maX_train_accuracy = max(history.history['sparse_categorical_accuracy'])
max_val_accuracy = max(history.history['val_sparse_categorical_accuracy'])
max_tra_acc_idx = np.argmax(history.history['sparse_categorical_accuracy']) + 1
max_val_acc_idx = np.argmax(history.history['val_sparse_categorical_accuracy']) + 1

# 훈련/검증 loss 최소값과 해당 인덱스를 변수에 할당
min_train_loss = min(history.history['loss'])
min_val_loss = min(history.history['val_loss'])
min_tra_loss_idx = np.argmin(history.history['loss']) + 1
min_val_loss_idx = np.argmin(history.history['val_loss']) + 1

# 훈련/검증 정확도 최대값과 해당 인덱스 출력, loss 최소값과 해당 인덱스 출력
print(f'Max train acc : {max_tra_acc_idx}_epoch_{maX_train_accuracy}')
print(f'Max val acc: {max_val_acc_idx}_epoch_{max_val_accuracy}')
print(f'Min train loss : {min_tra_loss_idx}_epoch_{min_train_loss}')
print(f'Min val loss: {min_val_loss_idx}_epoch_{min_val_loss}')

#  베스트 모델로 test dataset 평가 (베스트 모델은 callbacks에 설정 됨)
model = load_model('best_model.h5')
loss_cnn, acc_cnn = model.evaluate(X_test_exp, y_test)
print('test acc ', acc_cnn) # 정확도 출력
print('test loss ', loss_cnn) # loss 출력

## 학습데이터, 검증데이터의 정확도, loss 그래프 그리기
plt.figure(figsize=(10, 5))
plt.plot(history.history['sparse_categorical_accuracy'], label='Training accuracy')
plt.plot(history.history['val_sparse_categorical_accuracy'], label='Validation accuracy')
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.ylabel('Accuracy, Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()
```

[Metrics]

- Accuracy: 96.94%
- Precision: 100.0%
- Recall: 80.00%
- F1 Score: 88.89%
- AUC: 100.0%



# 학습 완료

❑ 학습 결과 요약은 2페이지 참조