

인공지능 응용 term project

| 음성 위조 탐지 판별 연구

27조

이예진
12201934

이상혁
12201928

장동근
12191812

임위순
12181828

목차

1

주제

- 선정 이유
- 주제 설명

2

데이터 분석

- Deep voice
- ASVspoof 2019
- MFCC / Mel spectrogram

3

모델링

- Deep voice
 - ANN / CNN
- ASVspoof
 - Simple ANN / Simple CNN
 - Resnet101 / Custom model

4

모델 비교

- 성능 비교

5

결과 정리 및 요약

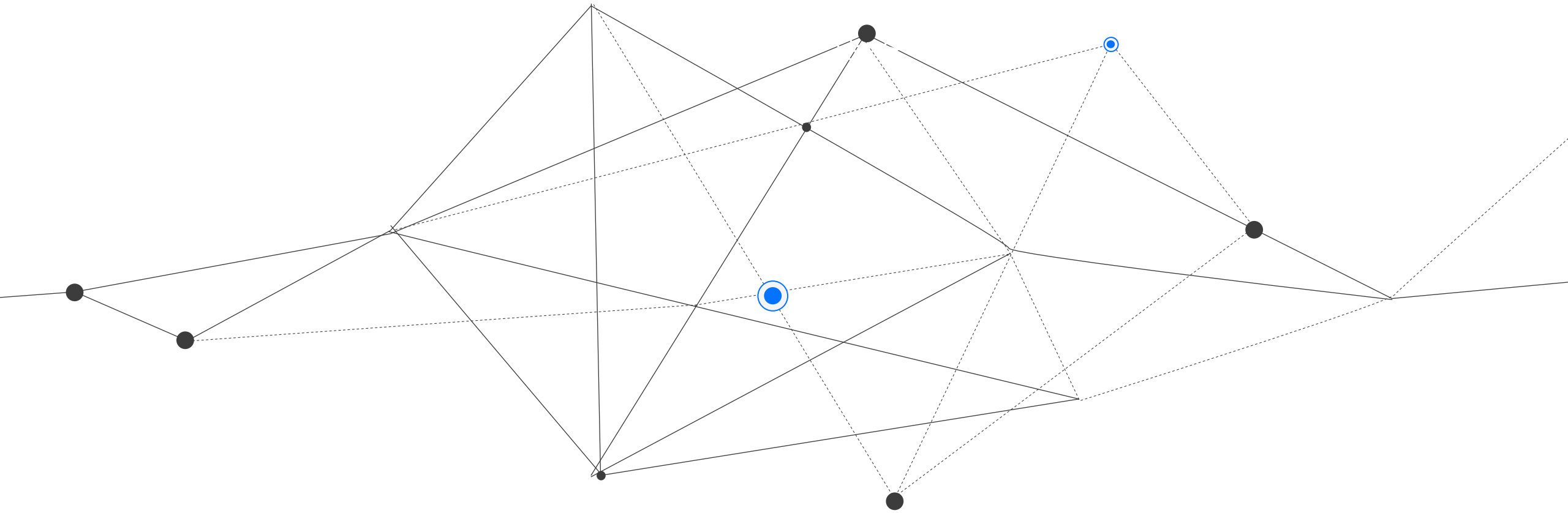
- 결과 분석
- case 적용

6


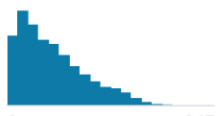
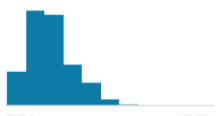
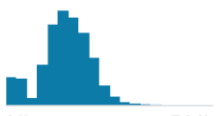
고찰

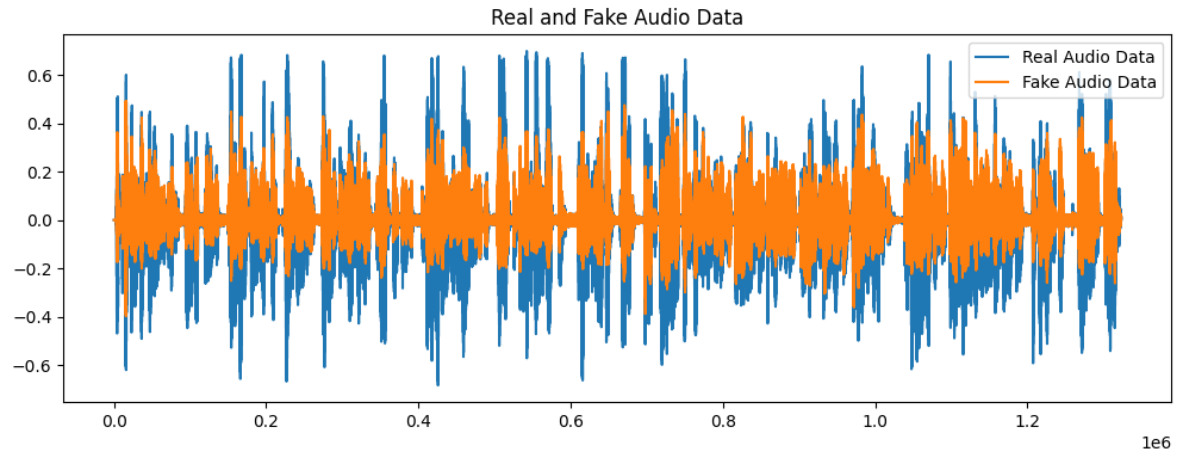
- 한계점
- 개선 방안 제안

음성 위조 탐지 판별 연구



Deep voice (DEEP-VOICE : Real-time Detection of AI-Generated Speech for DeepFake Voice Conversion)

# chroma_stft Chromagram	# rms Root Mean Square	# spectral_centroid Spectral Centroid	# spectral_bandwid... Spectral Bandwidth
			
0.338055	0.027948	2842.948867	4322.916759
0.443766	0.037838	2336.129597	3445.777044
0.302528	0.056578	2692.988386	2861.13318
0.319933	0.031504	2241.665382	3503.766175
0.420055	0.016158	2526.069123	3102.659519
0.44288	0.012317	3952.880304	3702.717829
0.453897	0.021782	4178.07215	3698.644769
0.474154	0.011107	3993.039753	3948.154333
0.60269	0.00097	3815.431438	3992.517515



원본 오디오 파일

8명의 유명인의 실제 음성 + Retrieval-based Voice Conversion을 통해 변환된 DeepFake 음성



추출된 특징 데이터("DATASET-balanced.csv")

1s 단위로 오디오 창에서 추출, 랜덤 샘플링으로 데이터 균형을 맞춘 데이터



ASVspoof 2019 (Automatic Speaker Verification(ASV)2019 Challenge's Dataset)

```
./ASVspoof2019_root
--> LA
    --> ASVspoof2019_LA_asv_protocols
    --> ASVspoof2019_LA_asv_scores
--> ASVspoof2019_LA_cm_protocols
    --> ASVspoof2019_LA_dev
    --> ASVspoof2019_LA_eval
--> ASVspoof2019_LA_train
--> README.LA.txt
--> PA
    --> ASVspoof2019_PA_asv_protocols
    --> ASVspoof2019_PA_asv_scores
--> ASVspoof2019_PA_cm_protocols
    --> ASVspoof2019_PA_dev
    --> ASVspoof2019_PA_eval
--> ASVspoof2019_PA_train
--> README.PA.txt
--> asvspoof2019_evaluation_plan.pdf
--> asvspoof2019_Interspeech2019_submission.pdf
--> README.txt
```

Table 3.1 Data description for asvspoof 2019 logical access (LA) dataset

Subsets	Male	Female	Bona fide	Spoof
Training set	8	12	2,580	22,800
Development set	8	12	2,548	22,296
Evaluation set	30	37	7,355	63,882

구성

- Voice Cloning Toolkit(VCTK2)를 기반으로 하는 원본 데이터
- Text To Speech(TTS), Voice Conversion(VC), Voice Synthesis(VS), Tacotron2, WaveNet 등의 음성 위조 알고리즘을 사용해 생성된 음성 데이터

샘플링

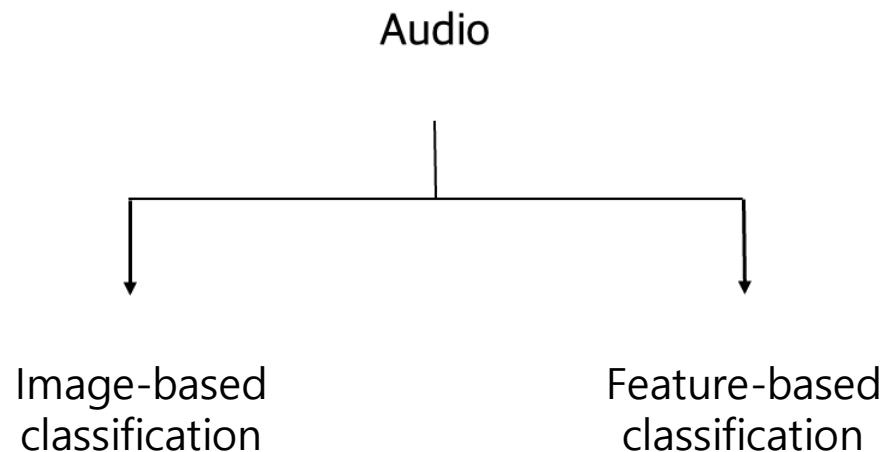
96kHz에서 녹음된 다중 화자 영어 음성 데이터베이스를 16kHz로 다운샘플링

Feature-based classification

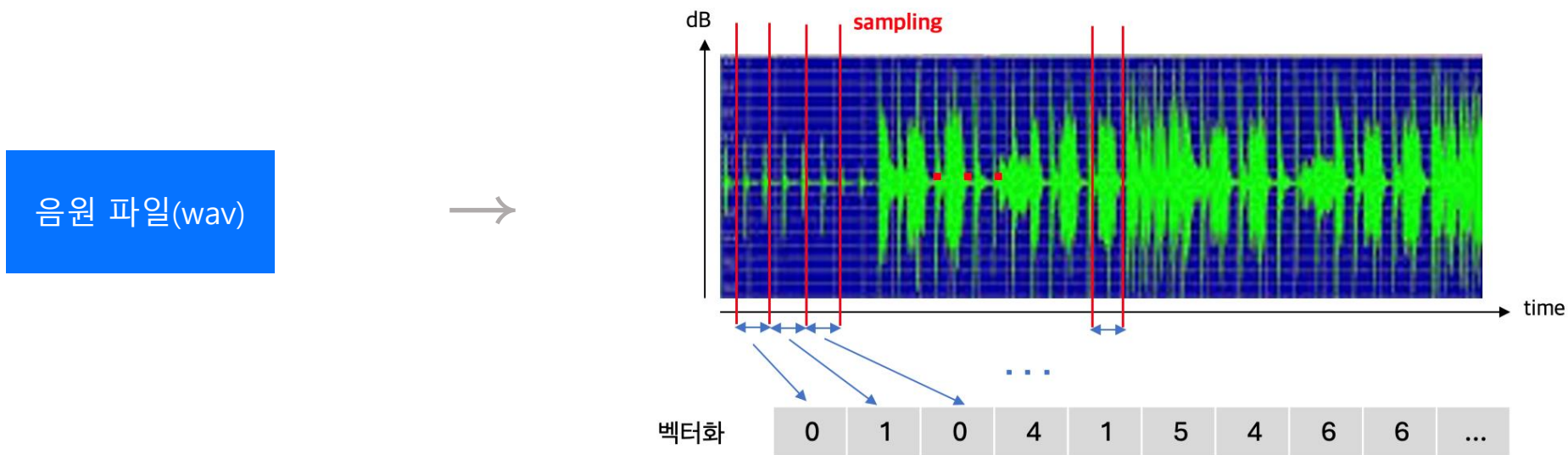
오디오 파일에서 얻은 음성데이터에서
Feature들을 추출하여 특징벡터화하여
이를 기반으로 학습하는 방법

Image-based classification

Mel-spectrogram으로 구성된
오디오 샘플을 적용하여 학습



음원 데이터를 time domain에서 그대로 벡터화하여 학습을 할 때의 단점



메모리 소모의 증가

샘플링 간격을 짧게 할수록
데이터의 길이, 차원이 증가
→ 학습의 효율성 감소, 메모리 소모 증가

불필요한 정보

Time domain의 파형은 amplitude와 phase를 모두 가짐
amplitude로 인식하는 생물학적 청각 특성을 고려 x

Image-based classification

feature-based classification

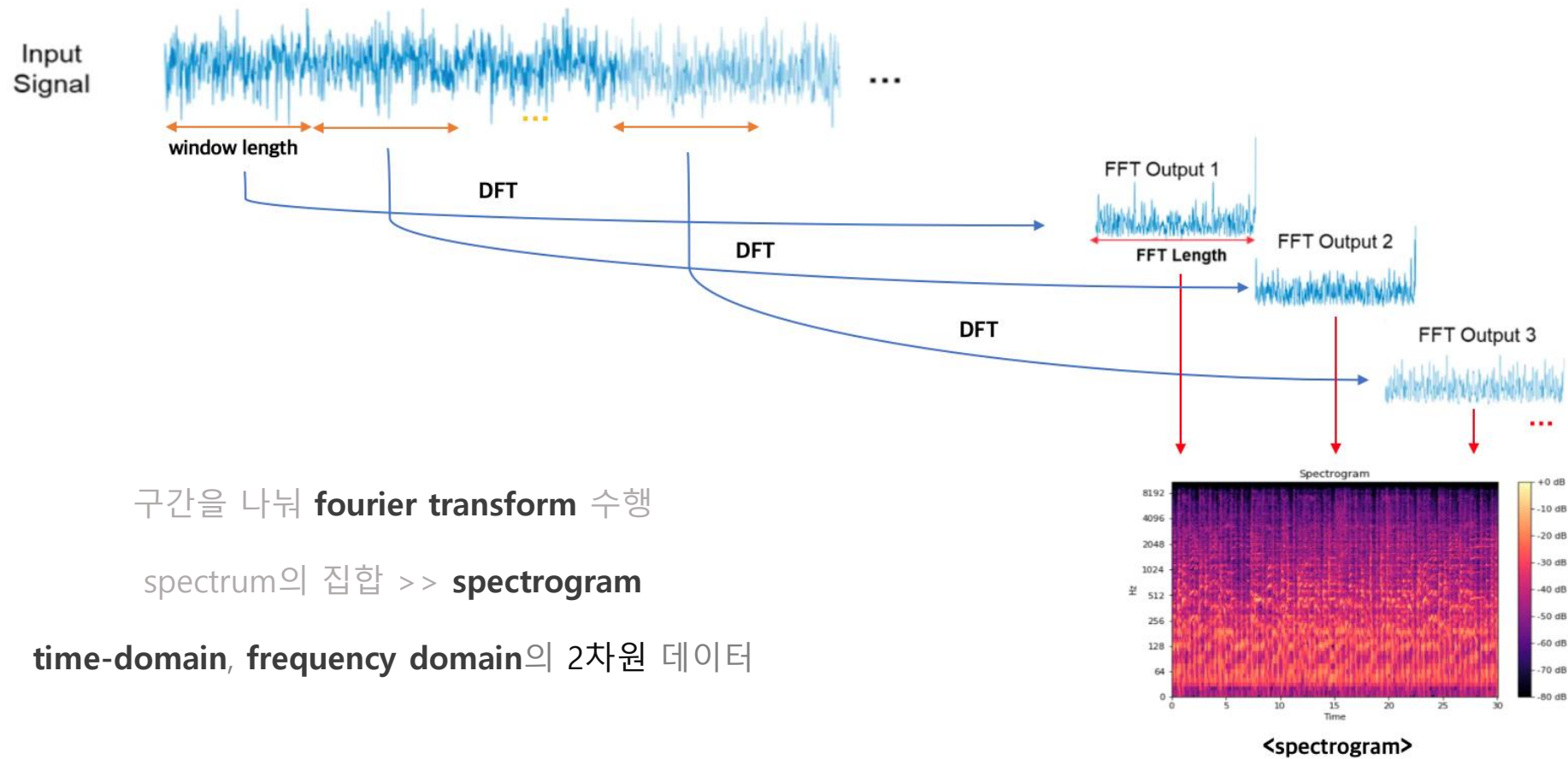
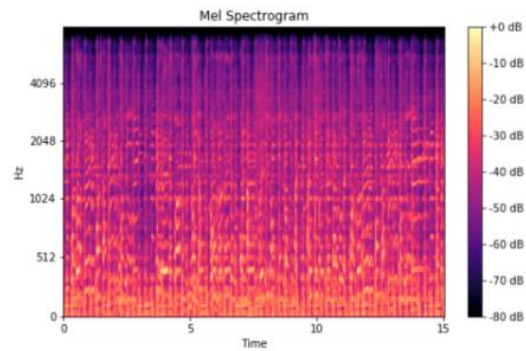


image-based classification

feature-based classification



Spectrogram(2D-image)

차원 축소



1D-array

DCT(Discrete Cosine Transform)

차원 축소

- correlation 삭제
- 독립적인 feature 추출

더 단순한 모델로 빠르고 효율적인 학습 가능

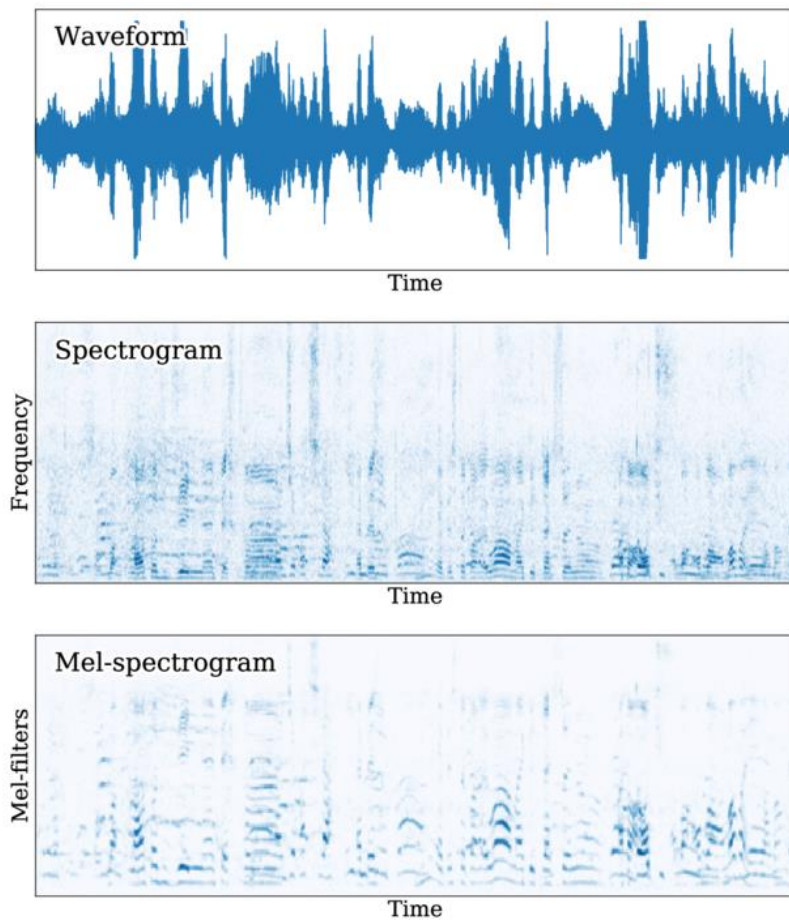
예) MFCC, LFCC

MFCC

Mel Spectrogram

LFCC

Chroma



● librosa 사용

```
fake_mfcc_librosa = librosa.feature.mfcc(y=fake_ad, sr=fake_sr)
```

● 직접 구현

```
def MFCC(y, sr, n_mfcc=13, n_fft=2048, hop_length=512, n_mels=128):  
    # STFT 계산  
    stft = np.abs(librosa.stft(y, n_fft=n_fft, hop_length=hop_length))**2  
    # 멜 필터 계산  
    mel_filter = librosa.filters.mel(sr=sr, n_fft=n_fft, n_mels=n_mels)  
    # 멜 스펙트로그램 계산  
    mel_spect = np.dot(mel_filter, stft)  
    # 로그 취하기  
    log_mel_spect = np.log(mel_spect + 1e-6)  
    # 로그 멜 스펙트로그램의 이산 코사인 변환 (DCT) 계산  
    mfcc = scipy.fftpack.dct(log_mel_spect, type=2, axis=0, norm='ortho')[:n_mfcc]  
    return mfcc
```

SFTF (Short Time Fourier Transform)을

바탕으로 한 **Spectrogram** 성분을

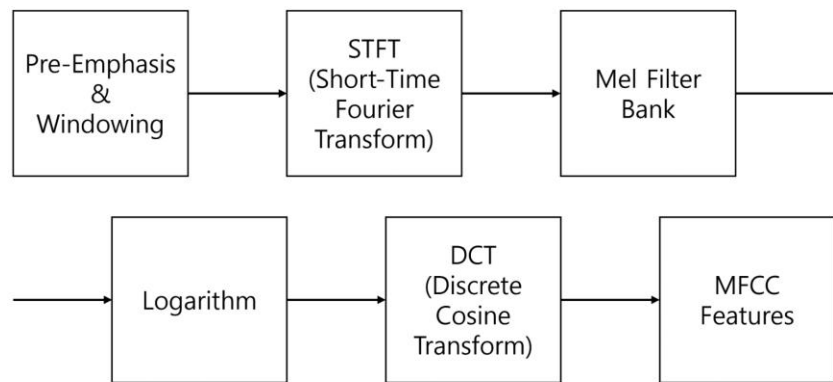
인간의 청각 특성을 잘 나타낼 수 있도록 **Mel Scale**로 변환

MFCC

Mel Spectrogram

LFCC

Chroma



● librosa 사용

```
fake_mel_spect = librosa.feature.melspectrogram(y=fake_ad, sr=fake_sr)
fake_mel_spect = librosa.power_to_db(fake_mel_spect, ref=np.max)
```

● 직접 구현

```
def melspectrogram(y, sr, n_fft=2048, hop_length=512, n_mels=128):
    # STFT 계산
    stft = np.abs(librosa.stft(y, n_fft=n_fft, hop_length=hop_length))**2
    # 멜 필터 계산
    mel_filter = librosa.filters.mel(sr=sr, n_fft=n_fft, n_mels=n_mels)
    # 멜 스펙트로그램 계산
    mel_spect = np.dot(mel_filter, stft)
    # 로그 파워 스펙트로그램으로 변환
    log_mel_spect = np.log(mel_spect + 1e-6)
    return log_mel_spect
```

음성 신호를 주파수 정보의 **고차원** 표현에서 더 의미 있는 **저차원**의 특징 공간으로 변환하는 기법

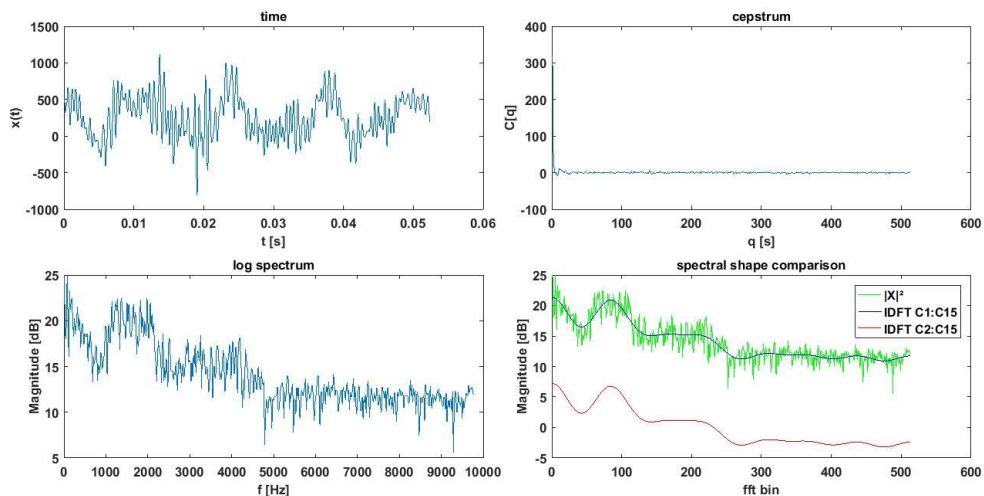
주파수 정보를 사람의 청각 특성에 더 가깝게 변환해 중요한 음성 특징을 추출하는 과정

MFCC

Mel Spectrogram

LFCC

Chroma



FFT를 통해 얻어낸 주파수 스펙트럼

→ 로그 변환하여 로그 스펙트럼을 얻고,

이를 역 푸리에 변환하여 cepstrum을 구하고,

선형 주파수 스케일로 분석해 주파수 성분을 추출

● librosa 사용

```
fake_lfcc_librosa = librosa.feature.lfcc(y=fake_ad, sr=fake_sr)
```

● 직접 구현

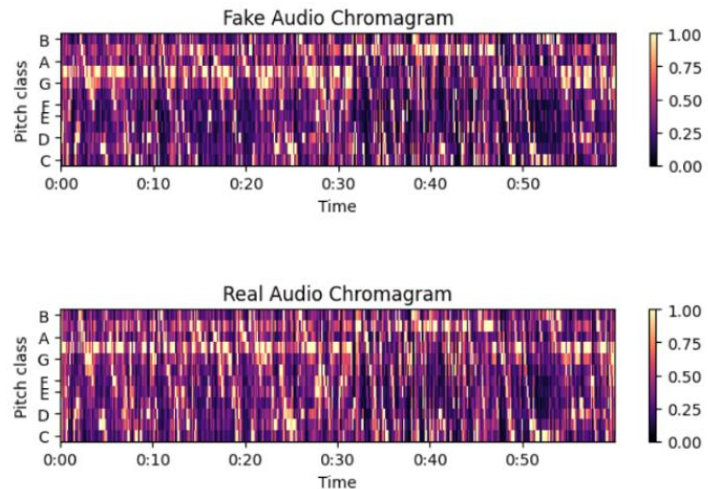
```
def LFCC(y, sr, n_fft=2048, hop_length=512, n_filter=40, n_lfcc=13):  
    # Short-Time Fourier Transform (STFT) 계산  
    stft = np.abs(librosa.stft(y, n_fft=n_fft, hop_length=hop_length))**2  
    # 선형 필터 계산  
    filter_banks = librosa.filters.mel(sr=sr, n_fft=n_fft, n_mels=n_filter, fmin=0,  
    filter_banks = filter_banks / np.max(filter_banks, axis=-1)[:, None] # 정규화  
    # 파워 스펙트럼에 필터뱅크 적용  
    energy = np.dot(filter_banks, stft)  
    # 로그 취하기  
    log_energy = np.log(energy + 1e-6)  
    # 로그 에너지의 이산 코사인 변환 (DCT) 계산  
    lfcc = scipy.fftpack.dct(log_energy, type=2, axis=0, norm='ortho')[:n_lfcc]  
    return lfcc
```

MFCC

Mel Spectrogram

LFCC

Chroma



● librosa 사용

```
fake_chroma_librosa = librosa.feature.chroma_cqt(y=fake_ad, sr=fake_sr, bins_per_octave=36)
```

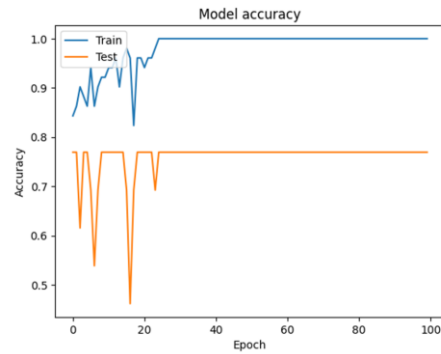
● 직접 구현

```
def CHROMA(y, sr, n_fft=2048, hop_length=512, n_chroma=12):  
    # STFT 계산  
    stft = np.abs(librosa.stft(y, n_fft=n_fft, hop_length=hop_length))  
    # 주파수 빈도에 해당하는 피치 클래스 빈도를 계산하는 크로마 필터 뱅크 생성  
    chroma_filter = librosa.filters.chroma(sr=sr, n_fft=n_fft, n_chroma=n_chroma)  
    # 크로마그램 계산  
    chroma = np.dot(chroma_filter, stft)  
    # 크로마 값을 정규화  
    chroma = chroma / chroma.sum(axis=0, keepdims=True)  
    return chroma
```

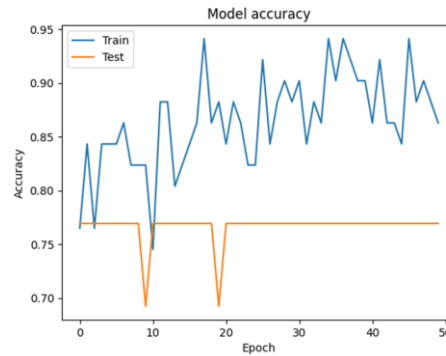
12개의 다른 **반음**(ex. C, C#, D, D#, E, F, F#, G, G#, A, A#, B) 각각의 강도를 나타내는,

한 옥타브(octave) 내의 모든 음표를 12개의 구간으로 나눈 **Chroma 벡터**를 계산

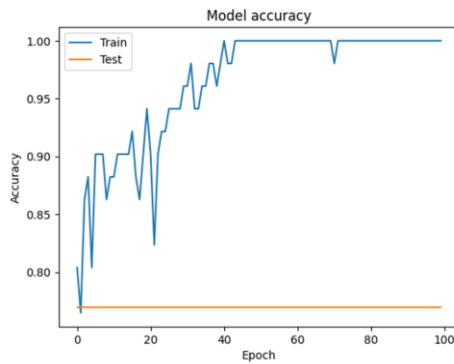
3. 문제 분석



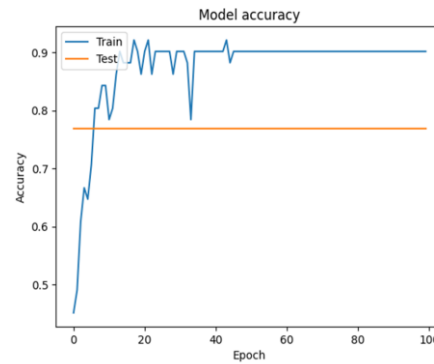
CNN



ANN + dropout



CNN



CNN + dropout

validation accuracy가 정체됨

원인?

" 데이터셋의 부족 "

fake : 53, real : 8

real test split(0.2) >> test data : 1



data split,
증강 60배

1. ANN 모델 – (1) 모델 정의

```
model = Sequential()
model.add(Dense(64, input_shape=input_shape))
model.add(Activation("relu"))

model.add(Dense(64))
model.add(Activation("relu"))

model.add(Dense(64))
model.add(Activation("relu"))

model.add(Dense(num_labels))
model.add(Activation("sigmoid"))

model.summary()
```

Model: "sequential_104"

Layer (type)	Output Shape	Param #
dense_415 (Dense)	(None, 64)	2624
activation_415 (Activation)	(None, 64)	0
dense_416 (Dense)	(None, 64)	4160
activation_416 (Activation)	(None, 64)	0
dense_417 (Dense)	(None, 64)	4160
activation_417 (Activation)	(None, 64)	0
dense_418 (Dense)	(None, 2)	130
activation_418 (Activation)	(None, 2)	0
Total params: 11074 (43.26 KB)		
Trainable params: 11074 (43.26 KB)		
Non-trainable params: 0 (0.00 Byte)		

Sequential 모델을 사용해
3개의 은닉층, 출력층으로 구성된 신경망을
정의

각 은닉층은 64개의 뉴런을 가지며,
활성화 함수로 **ReLU**를 사용

출력층은 num_labels만큼의 뉴런을 가지며,
활성화 함수로 **소프트맥스(softmax)**를 사용

1. ANN 모델 - (2) 콜백 설정 및 컴파일 결과

```
1 checkpoint_cb = keras.callbacks.ModelCheckpoint('ANN.keras',  
2                                             save_best_only=True)  
3 early_stopping_cb = keras.callbacks.EarlyStopping(patience=2,  
4                                             restore_best_weights=True)  
5  
6 model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics = 'accuracy')  
7 history = model.fit(xtrain, ytrain, epochs=50, batch_size=3, validation_data=(xtest, ytest), verbose=1,  
8 )
```

ModelCheckpoint 콜백

>> 최적의 모델을 저장

EarlyStopping 콜백

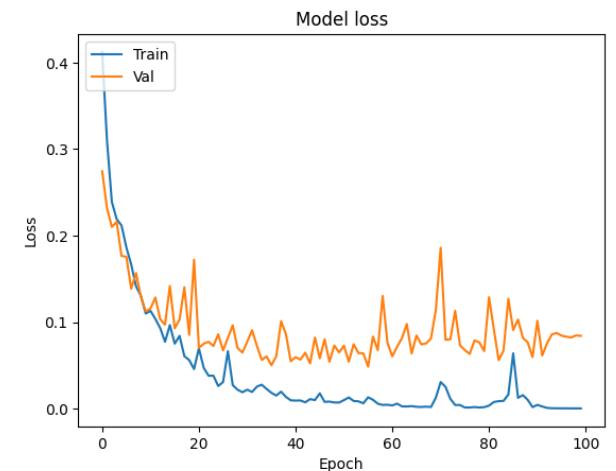
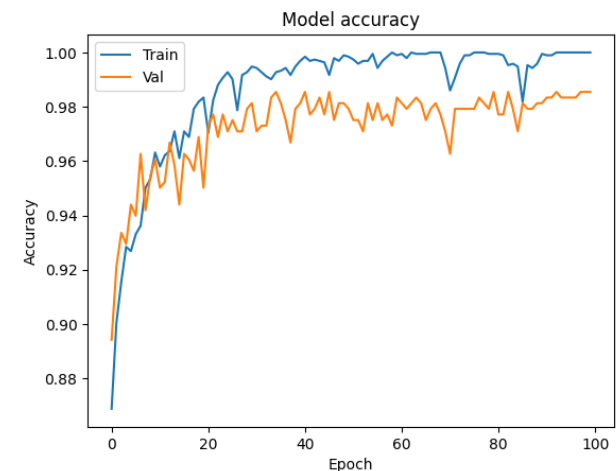
>> 검증 손실이 개선되지 않으면 학습을
조기에 종료

Adam 옵티마이저, 손실 함수

binary_crossentropy를 사용

>> 모델을 컴파일

19/19 [=====] - 0s 2ms/step - loss: 0.0135 - accuracy: 0.9934
Test Loss: 0.013452514074742794
Test Accuracy: 0.9933664798736572



1. ANN 모델 - (3) dropout 및 최적화

```
model = Sequential()
model.add(Dense(64, input_shape=input_shape))
model.add(Activation("relu"))
model.add(Dropout(0.2))

model.add(Dense(64))
model.add(Activation("relu"))
model.add(Dropout(0.2))

model.add(Dense(64))
model.add(Activation("relu"))
model.add(Dropout(0.2))

model.add(Dense(num_labels))
model.add(Activation("sigmoid"))

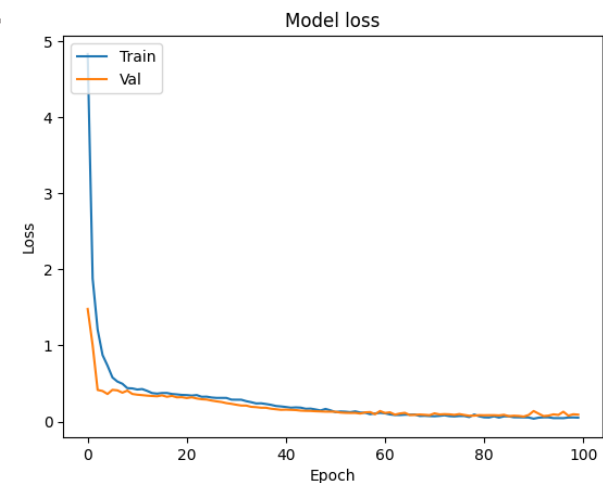
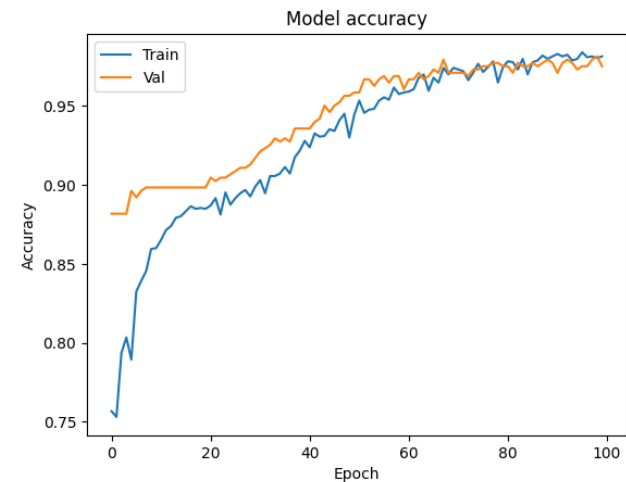
model.summary()
```

```
1 checkpoint_cb = keras.callbacks.ModelCheckpoint('ANN+dropout.keras',
2                                           save_best_only=True)
3 early_stopping_cb = keras.callbacks.EarlyStopping(patience=2,
4                                           restore_best_weights=True)
5
6 model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics = 'accuracy')
7 history = model.fit(xtrain, ytrain, epochs=50, batch_size=3, validation_data=(xtest, ytest), verbose=1,
8 )
```

Dropout(0.1)

>> 각 레이어에서 10%의 뉴런을
무작위로 제외하여 오버피팅을 방지함

19/19 [=====] - 0s 2ms/step - loss: 0.0378 - accuracy: 0.9967
Test Loss: 0.03782206028699875
Test Accuracy: 0.9867330193519592



2. CNN 모델 - (1) 모델 정의

```
x = np.array(data)
y = np.array(labels)

# 라벨 인코딩
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

xtrain, xtest, ytrain, ytest = train_test_split(x, y_encoded, test_size = 0.2)
print("훈련 세트의 크기:", xtrain.shape, ytrain.shape)
print("검증 세트의 크기:", xtest.shape, ytest.shape)

# 1D 합성곱을 사용하기 위해 입력 데이터를 3차원으로 변환 (samples, timesteps, features)
xtrain = xtrain.reshape(xtrain.shape[0], xtrain.shape[1], 1)
xtest = xtest.reshape(xtest.shape[0], xtest.shape[1], 1)

input_shape = xtrain.shape[1:]
num_labels = len(folders) # label의 개수는 folders에 있는 REAL과 FAKE 2개
```

데이터 전처리

train / test data set를 분리,
CNN 모델에 맞도록 3차원으로 변환해 입력

```
model = Sequential()
model.add(Conv1D(64, 3, padding='same', activation='relu', input_shape=input_shape))
model.add(MaxPooling1D(pool_size=2))

model.add(Conv1D(64, 3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))

model.add(Conv1D(64, 3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(num_labels))
model.add(Activation("softmax"))

model.summary()
```

CNN 모델 정의

1D Convolution 레이어 X 3
Max pooling 레이어 X 3

→

특징 추출

Flatten 레이어

→

1차원 변환

Dense 레이어
Dropout 레이어

→

비선형 변환, 오버피팅
방지

2. CNN 모델 - (2) 콜백 설정 및 컴파일 결과

```
1 checkpoint_cb = keras.callbacks.ModelCheckpoint('CNN.keras',  
2                                           save_best_only=True)  
3 early_stopping_cb = keras.callbacks.EarlyStopping(patience=2,  
4                                           restore_best_weights=True)  
5  
6 model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics = 'accuracy')  
7 history = model.fit(xtrain, ytrain, epochs=50, batch_size=3, validation_data=(xtest, ytest), verbose=1,  
8 )
```

ModelCheckpoint 콜백

>> 최적의 모델을 저장

EarlyStopping 콜백

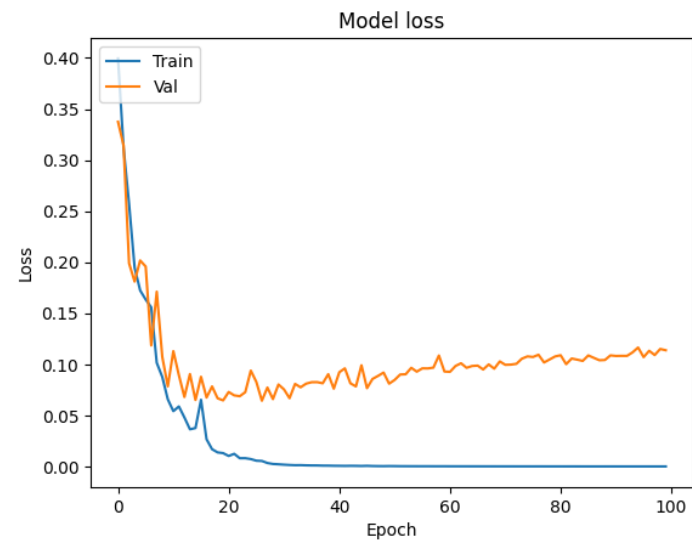
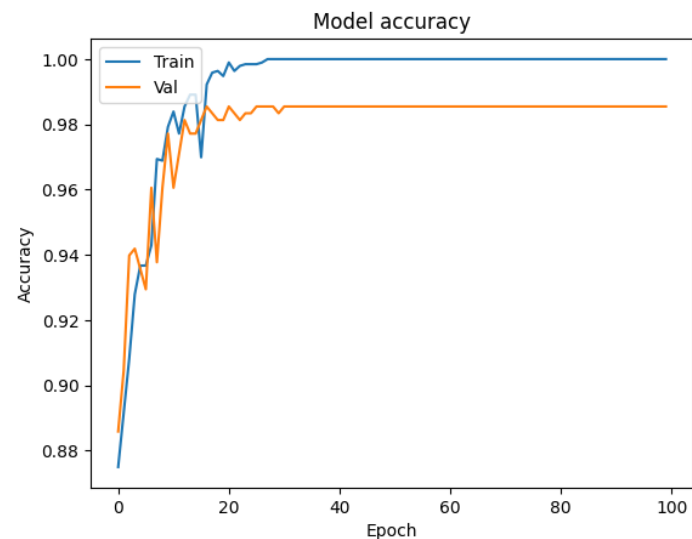
>> 검증 손실이 개선되지 않으면 학습을
조기에 종료

옵티마이저 **Adam**

손실 함수 **binary_crossentropy**

>> 모델을 컴파일

(앞과 동일)



2. CNN 모델 - (3) dropout 및 최적화

```
model = Sequential()
model.add(Conv1D(64, 3, padding='same', activation='relu', input_shape=input_shape))
model.add(MaxPooling1D(pool_size=2))

model.add(Conv1D(64, 3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))

model.add(Conv1D(64, 3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))

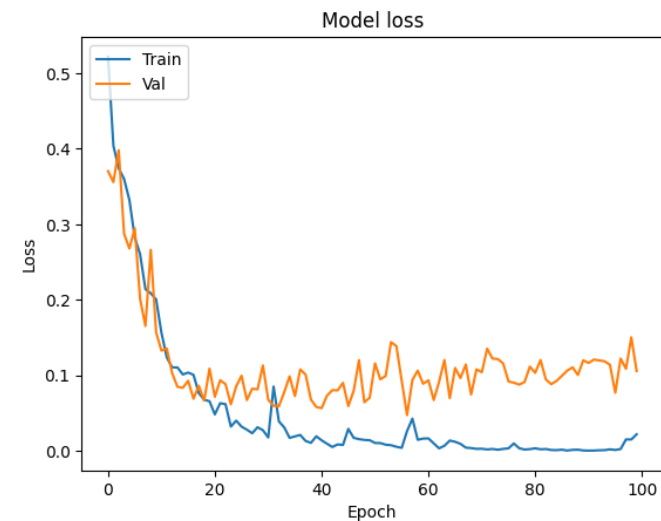
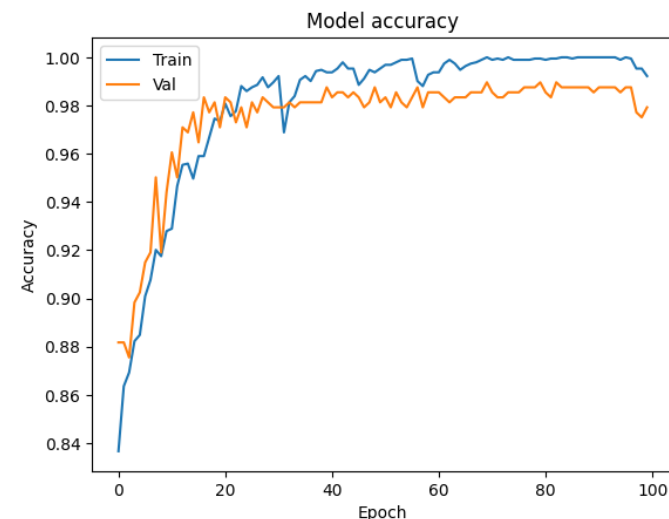
model.add(Dense(num_labels))
model.add(Activation("softmax"))
```

```
1 checkpoint_cb = keras.callbacks.ModelCheckpoint('CNN+dropout.keras',
2                                             save_best_only=True)
3 early_stopping_cb = keras.callbacks.EarlyStopping(patience=2,
4                                                    restore_best_weights=True)
5
6 model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics = 'accuracy')
7 history = model.fit(xtrain, ytrain, epochs=50, batch_size=3, validation_data=(xtest, ytest), verbose=1,
8                      )
```



노드 64*3 -> 32*2개로 감축, Dense 레이어 추가, Dropout(0.5)

>> 오버피팅을 방지함



1.(1) Simple ANN + Mel 모델 - 모델 정의

```
class SimpleANN(nn.Module):
    def __init__(self):
        super(SimpleANN, self).__init__()
        self.flatten = nn.Flatten()
        # 입력 데이터의 총 크기: 1*64*188 = 12032
        self.linear1 = nn.Linear(12032, 1024) # 첫 번째 선형 레이어
        self.relu = nn.ReLU() # 활성화 함수
        self.linear2 = nn.Linear(1024, 512) # 두 번째 선형 레이어
        self.linear3 = nn.Linear(512, 256) # 세 번째 선형 레이어
        self.linear4 = nn.Linear(256, 1) # 출력 레이어
        self.sigmoid = nn.Sigmoid() # 출력 활성화 함수

    def forward(self, x):
        x = self.flatten(x)
        x = self.linear1(x)
        x = self.relu(x)
        x = self.linear2(x)
        x = self.relu(x)
        x = self.linear3(x)
        x = self.relu(x)
        x = self.linear4(x)
        x = self.sigmoid(x) # 이진 분류를 위한 시그모이드 함수
        return x
```

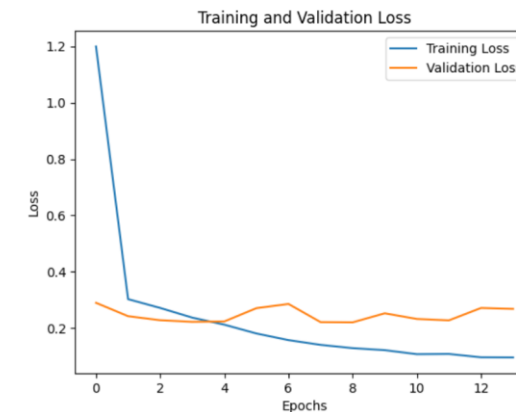
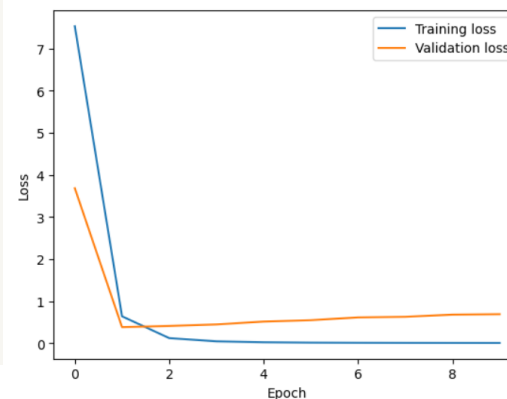
옵티마이저 - Adam

손실 함수 - binary_crossentropy

>> 모델을 컴파일

1.(2) Simple ANN + Dropout + MFCC - 모델 정의

```
# 모델 정의
model = Sequential([
    Dense(128, activation='relu', input_shape=(40,)), # 입력층
    Dropout(0.5), # 드롭아웃 레이어 추가
    Dense(64, activation='relu'), # 은닉층
    Dropout(0.5), # 드롭아웃 레이어 추가
    Dense(1, activation='sigmoid') # 출력층 (이진 분류)
])
```



1.(3) Simple ANN vs Dropout + MFCC 모델 비교

모델	SimpleANN	SimpleANN + Dropout + MFCC
주요 특징	<ul style="list-style-type: none"> •완전 연결된 레이어로 구성됨 •ReLU 활성화 함수 사용 •이진 분류를 위해 Sigmoid 함수 사용 	<ul style="list-style-type: none"> •Dropout을 추가하여 과적합 방지 " "
적합한 데이터	<ul style="list-style-type: none"> •단순한 구조의 입력 데이터 •이미지나 시퀀스가 아닌 데이터 	<ul style="list-style-type: none"> •이미지나 시퀀스가 아닌 데이터
구조	Flatten -> Linear(12032, 1024) -> ReLU -> Linear(1024, 512) -> ReLU -> Linear(512, 256) -> ReLU -> Linear(256, 1) -> Sigmoid	Dense(128) -> Dropout(0.5) -> Dense(64) -> Dropout(0.5) -> Dense(1) -> Sigmoid
파라미터 수	약 12,350,721	약 12,416
특징	간단한 완전 연결 구조, ReLU 활성화 함수 사용	Dropout으로 과적합 방지

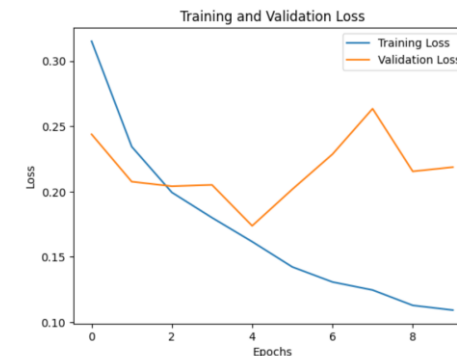
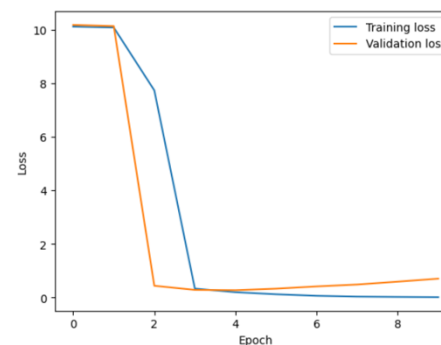
2.(1) Simple CNN 모델 – 모델 정의

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        # 첫 번째 Convolutional layer
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5, stride=1, padding=2) # 출력 크기: 32x64x188
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0) # 출력 크기: 32x32x94 (Pooling)
        # 두 번째 Convolutional layer
        self.conv2 = nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2) # 출력 크기: 64x32x94
        # 두 번째 Pooling을 거치면 64x16x47
        # Fully connected layers
        self.fc1 = nn.Linear(64 * 16 * 47, 1024) # 첫 번째 완전 연결 레이어
        self.fc2 = nn.Linear(1024, 512) # 두 번째 완전 연결 레이어
        self.fc3 = nn.Linear(512, 256) # 세 번째 완전 연결 레이어
        self.fc4 = nn.Linear(256, 1) # 최종 출력 레이어
        self.sigmoid = nn.Sigmoid() # 출력 활성화 함수(이진 분류)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x))) # 첫 번째 Convolutional + Pooling
        x = self.pool(F.relu(self.conv2(x))) # 두 번째 Convolutional + Pooling
        x = x.view(-1, 64 * 16 * 47) # Flatten
        x = F.relu(self.fc1(x)) # 첫 번째 Fully connected layer
        x = F.relu(self.fc2(x)) # 두 번째 Fully connected layer
        x = F.relu(self.fc3(x)) # 세 번째 Fully connected layer
        x = self.fc4(x) # 최종 출력 레이어
        x = self.sigmoid(x) # 시그모이드 활성화 함수
        return x
```

2.(2) Simple CNN 모델 + Dropout – 모델 정의

```
# 모델 정의
model = Sequential([
    Conv1D(128, 3, activation='relu', input_shape=(x_train.shape[1], 1)),
    MaxPooling1D(2),
    Dropout(0.5),
    Conv1D(64, 3, activation='relu'),
    MaxPooling1D(2),
    Dropout(0.5),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
```



2.(3) Simple CNN vs Dropout 모델 비교

모델	SimpleCNN	SimpleCNN + Dropout + MFCC
주요 특징	<ul style="list-style-type: none"> •합성곱 레이어와 풀링 레이어 사용 •완전 연결된 레이어로 분류 •이미지 처리에 특히 효과적 	<ul style="list-style-type: none"> •합성곱 레이어와 풀링 레이어 사용 •Dropout을 추가하여 과적합 방지
적합한 데이터	<ul style="list-style-type: none"> •이미지 데이터 •공간적 특징 추출이 필요한 경우 	<ul style="list-style-type: none"> •공간적 특징 추출이 필요한 경우
구조	Conv2d(1, 32) -> ReLU -> MaxPool2d -> Conv2d(32, 64) -> ReLU -> MaxPool2d -> Linear(64 * 16 * 47, 1024) -> ReLU -> Linear(1024, 512) -> ReLU -> Linear(512, 256) -> Linear(256, 1) -> Sigmoid	Conv1D(128) -> ReLU -> MaxPooling1D -> Dropout(0.5) -> Conv1D(64) -> ReLU -> MaxPooling1D -> Dropout(0.5) -> Flatten -> Dense(64) -> ReLU -> Dense(1) -> Sigmoid
파라미터 수	약 7,177,857	약 52,289
특징	Convolutional 레이어와 Max Pooling 사용, ReLU 활성화 함수 사용	Dropout으로 과적합 방지, ReLU 활성화 함수 사용

3.(1) Custom Model (Resnet101 기반) – 모델 정의

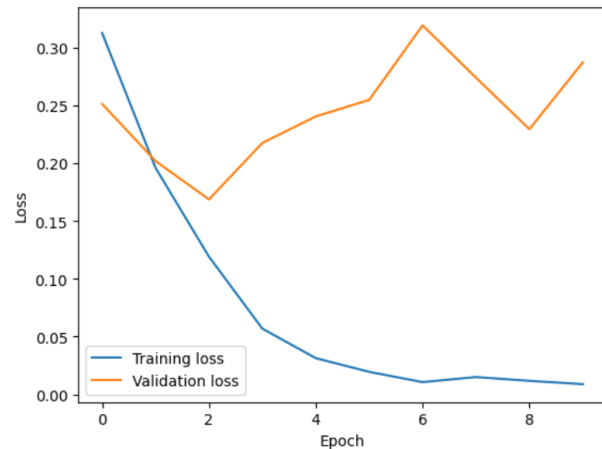
```
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        # ResNet101 모델사전 훈련된 가중치 사용하며, 입력 채널 수를 1
        self.model = timm.create_model('resnet101', pretrained=True, in_chans=1)

        # 모델의 파라미터 중 일부를 고정
        for i, (name, param) in enumerate(list(self.model.named_parameters()))[:39]:
            param.requires_grad = False

        # ResNet101 모델의 마지막 두 개의 레이어를 제외한 나머지 레이어들을 features로 정의
        self.features = nn.Sequential(*list(self.model.children())[:-2])

        # 새로운 레이어를 정의. AdaptiveAvgPool2d를 통해 출력의 공간 크기를 (1, 1)로 조정, Flatten 레이어를 사용하여 2D 텐서를 1D로
        # fully connected 레이어와 시그모이드 활성화 함수로 이루어진 레이어를 정의
        self.custom_layers = nn.Sequential(
            nn.AdaptiveAvgPool2d(1), # 출력의 공간 크기를 (1, 1)로 조정
            nn.Flatten(), # 2D 텐서를 1D로 평탄화
            nn.Linear(self.model.num_features, 1), # fully connected 레이어를 정의
            nn.Sigmoid() # 시그모이드 활성화 함수를 적용하여 이진 분류를 수행
        )

    def forward(self, inputs):
        # 입력 데이터를 ResNet101의 특성 추출기 부분을 통과
        x = self.features(inputs)
        # 특성 추출된 데이터를 새로운 레이어(custom_layers)를 통과시켜 최종 출력을 계산
        x = self.custom_layers(x)
        return x
```



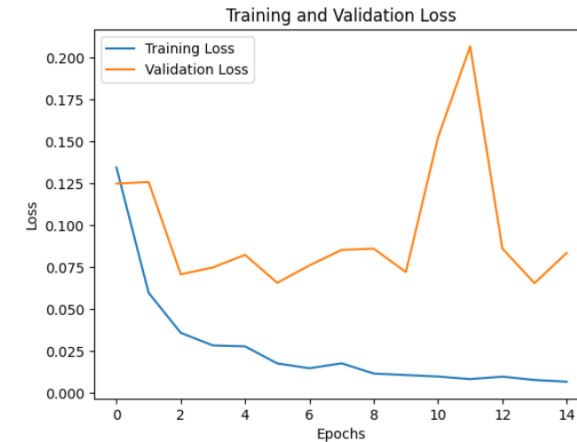
3.(2) Custom Model (Resnet202 기반) – 모델 정의

```
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = timm.create_model('resnet200d.ra2_in1k', pretrained = True,
            in_chans = 1)
        for i, (name, param) in enumerate(list(self.model.named_parameters()))\
            [0:39]):
            param.requires_grad = False

        self.features = nn.Sequential(*list(self.model.children())[:-2])

        self.custom_layers = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(self.model.num_features, 1),
            nn.Sigmoid()
        )

    def forward(self, inputs):
        x = self.features(inputs)
        x = self.custom_layers(x)
        return x
```



3.(3) LSTM – 모델 정의

```
# 모델 정의
model = Sequential([
    LSTM(128, input_shape=(x_train.shape[1], 1)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
```



```
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        # ResNet101 모델사전 훈련된 가중치 사용하며, 입력 채널 수를 1
        self.model = timm.create_model('resnet101', pretrained=True, in_chans=1)

        # 모델의 파라미터 중 일부를 고정
        for i, (name, param) in enumerate(list(self.model.named_parameters())[:39]):
            param.requires_grad = False

        # ResNet101 모델의 마지막 두 개의 레이어를 제외한 나머지 레이어들을 features로 정의
        self.features = nn.Sequential(*list(self.model.children())[:-2])

        # 새로운 레이어를 정의. AdaptiveAvgPool2d를 통해 출력의 공간 크기를 (1, 1)로 조정, Flatten 레이어를 사용하여 2D 텐서를 1D로
        # fully connected 레이어와 시그모이드 활성화 함수로 이루어진 레이어를 정의
        self.custom_layers = nn.Sequential(
            nn.AdaptiveAvgPool2d(1), # 출력의 공간 크기를 (1, 1)로 조정
            nn.Flatten(), # 2D 텐서를 1D로 평탄화
            nn.Linear(self.model.num_features, 1), # fully connected 레이어를 정의
            nn.Sigmoid() # 시그모이드 활성화 함수를 적용하여 이진 분류를 수행
        )

    def forward(self, inputs):
        # 입력 데이터를 ResNet101의 특성 추출기 부분을 통과
        x = self.features(inputs)
        # 특성 추출된 데이터를 새로운 레이어(custom_layers)를 통과시켜 최종 출력을 계산
        x = self.custom_layers(x)
        return x
```

3.(4) Custom Model (Resnet101 기반) vs Custom Model (Resnet202 기반) vs LSTM 모델 비교

모델	Custom Model (Resnet101 기반)	Custom Model (Resnet202 기반)	LSTM
주요 특징	<ul style="list-style-type: none"> •ResNet101 아키텍처 기반 •전이 학습을 통해 성능 향상 가능 •이미지 특성 추출을 위해 사전 훈련된 모델 사용 	<ul style="list-style-type: none"> ResNet202 아키텍처 기반 전이 학습을 통해 성능 향상 가능 	<ul style="list-style-type: none"> 시계열 데이터에 적합 LSTM 레이어 사용
적합한 데이터	<ul style="list-style-type: none"> •이미지 데이터 •대규모 이미지 데이터, 딥러닝 태스크에 적합 	<ul style="list-style-type: none"> •이미지 데이터 •대규모 이미지 데이터, 딥러닝 태스크에 적합 	<ul style="list-style-type: none"> •시계열 데이터 •시계열 데이터, 연속적인 특징 추출 필요
구조	ResNet101의 특성 추출기 부분 -> AdaptiveAvgPool2d -> Flatten -> Linear -> Sigmoid	ResNet202의 특성 추출기 부분 -> AdaptiveAvgPool2d -> Flatten -> Linear -> Sigmoid	LSTM(128) -> Dropout(0.5) -> Dense(64) -> Sigmoid
파라미터 수	약 42,497,969	약 41,498,841	약 93,057
특징	사전 훈련된 ResNet101 아키텍처 사용, 전이 학습에 적합	사전 훈련된 ResNet202 아키텍처 사용, 전이 학습에 적합	LSTM을 사용하여 시계열 데이터에 적합

점수 산정 방식

● ROC Curve

$$TNR = \frac{\text{예측 : Negative, 실제 : Negative}}{\text{실제 : Negative}} = \frac{TN}{FP + TN}$$

$$FPR = \frac{\text{예측 : Positive, 실제 : Negative}}{\text{실제 : Negative}} = \frac{FP}{FP + TN} = 1 - \text{Specificity (TNR)}$$

$$TPR = \frac{\text{예측 : Positive, 실제 : Positive}}{\text{실제 : Positive}} = \frac{TP}{FN + TP} = \text{Recall (= Sensitivity)}$$

● 코드

```
from sklearn.metrics import roc_curve

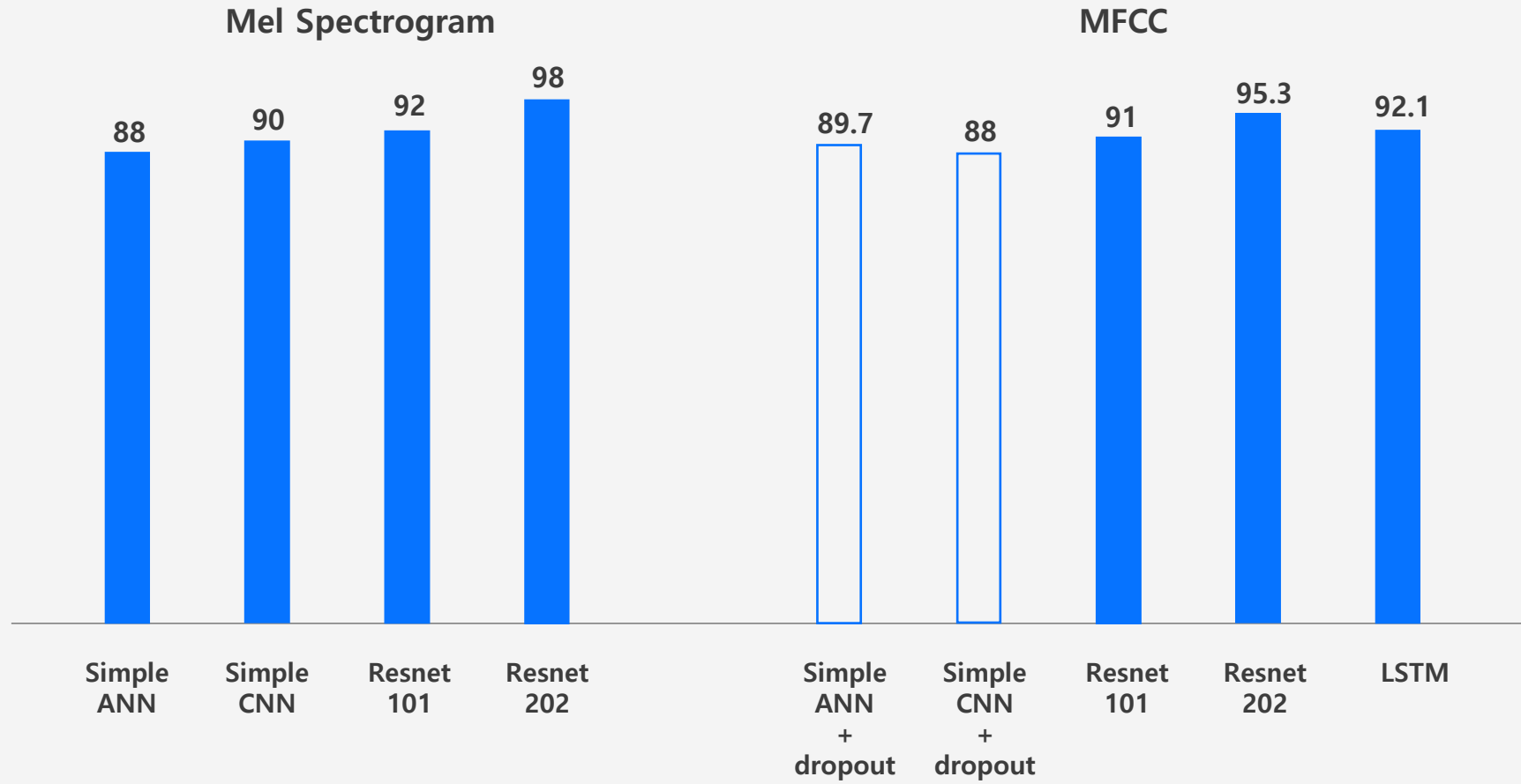
# 레이블 값이 1일때의 예측 확률을 추출
pred_proba_class1 = lr_clf.predict_proba(X_test)[:, 1]

fprs , tprs , thresholds = roc_curve(y_test, pred_proba_class1)
# 반환된 임계값 배열에서 샘플로 데이터를 추출하되, 임계값을 5 Step으로 추출.
# thresholds[0]은 max(예측확률)+1로 임의 설정됨. 이를 제외하기 위해 np.arange는 1부터 시작
thr_index = np.arange(1, thresholds.shape[0], 5)
print('샘플 추출을 위한 임계값 배열의 index:', thr_index)
print('샘플 index로 추출한 임계값: ', np.round(thresholds[thr_index], 2))

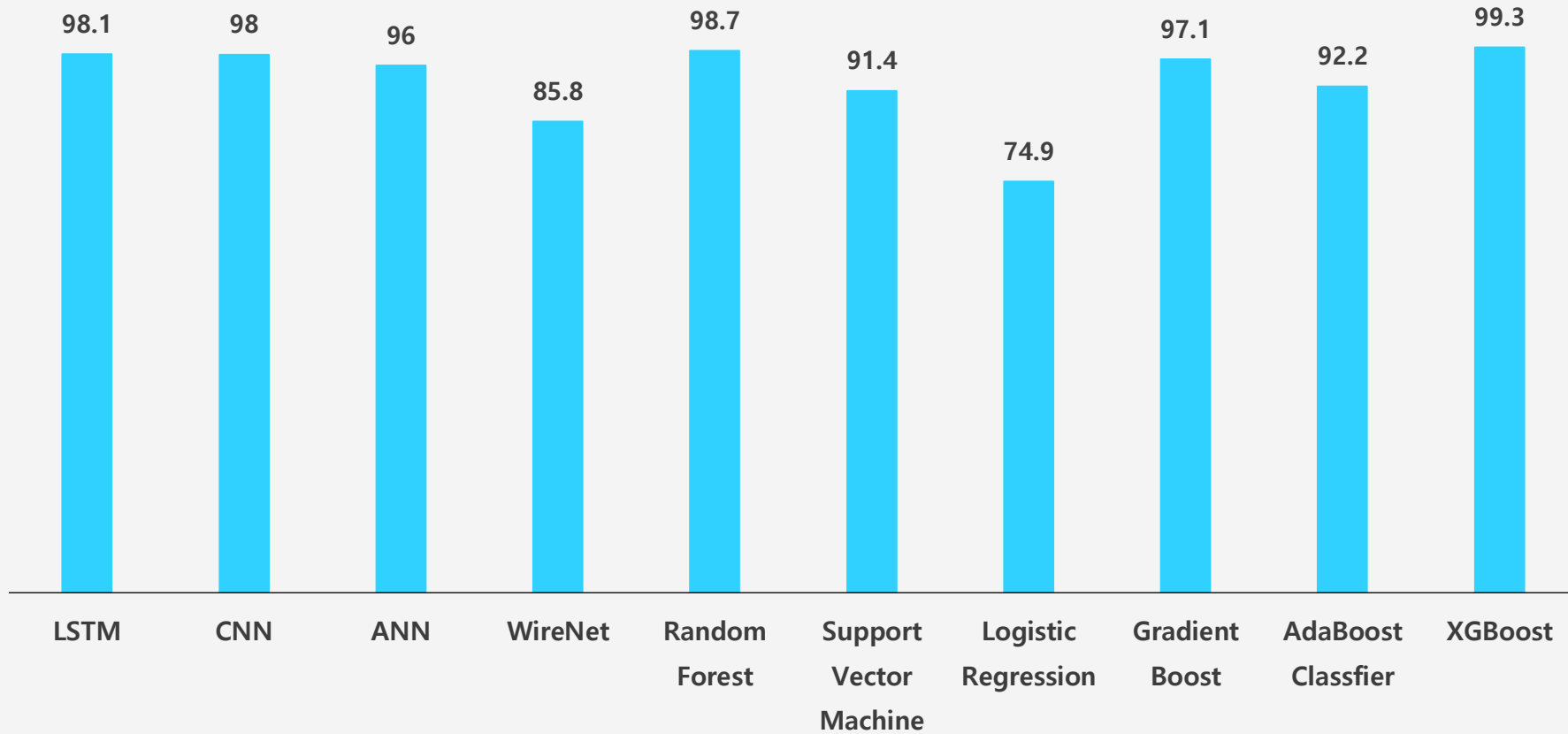
# 5 step 단위로 추출된 임계값에 따른 FPR, TPR 값
print('샘플 임계값별 FPR: ', np.round(fprs[thr_index], 3))
print('샘플 임계값별 TPR: ', np.round(tprs[thr_index], 3))
```

분석

Y : ROC_SCORE



추가사항 | Deep voice set 데이터 증강 후



한계점

데이터셋의 부족

배경지식 부족

개발환경의 한계

개선 방안 및 활용

음성 인증 시스템
보호

보이스피싱 범죄
탐지

보안 서비스 활용

새로운 특징추출법
및 알고리즘

모델 앙상블

VIT 사용

Reference

- 음성 위조 탐지를 위한 2단계 학습 모형 연구
[Journal of the Korean Data & <http://dx.doi.org/10.7465/jkdi.2023.34.2.203> Information Science Society]
- 합성된 음성과 발화 음성에 대한 차이 연구 Audio deepfake detection 기술 개발
(survey:<https://arxiv.org/pdf/2308.14970.pdf>)
- Deepfake Audio Detection via MFCC Features Using Machine Learning
(<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9996362>)
- Mel-Spectrogram과 MFCC를 이용한 딥러닝 기반 딥보이스 탐지시스템 개발에 관한 연구
(The Transactions P of the Korean Institute of Electrical Engineers KIEEP Vol. 72P, No. 03, p.186-192)
- <https://velog.io/@gangjoo/ML-%ED%8F%89%EA%B0%80-F1-Score%EC%99%80-ROC-AUC>