인공지능 응용 term project 보고서

음성 위조 탐지 판별 연구

인하대학교 정보통신공학과

12201934 이예진

12201928 이상혁

12191812 장동근

12181828 임위순

2024 – 06 - 08

목 차

1. 서론

AI의 핵심기술인 딥러닝과 '페이크 보이스(fake voice)'를 합친 신조어인 딥보이스(Deep Voice) 기술은 인공지능을 활용하여 사람의 목소리를 모방하거나 변조하는 기술을 말한다. 최근 몇 년간 이 기술은 크게 발전하여 실제 사람의 목소리와 구별하기 어려울 정도로 정교해졌다. 이러한 진보는 많은 긍정적인 가능성을 열었지만, 동시에 심각한 위험성도 내포하고 있다. 특히 보이스 피싱에 악용되어 가족이나 지인, 공공기관을 사칭하는 사례가 더욱 늘어날 것으로 예상되며, 손쉽게 음성 조작이 가능해진 만큼 앞으로 다양한 유형의 범죄가 증가할 가능성도 커질 것으로 예상된다.

따라서 본 프로젝트 다양한 딥러닝 모델을 기반으로 위조된 음성을 탐지하는 시스템들을 비교한다. 음성 데이터셋은 kaggle의 deep-voice, ASVspoof 2019 데이터 셋을 활용하였다. 이렇게 수집한 음성 데이터를 MFCC와 Mel-Spectrogram이라는 두 가지 음성 특징 추출 기법을 사용하여 음성 데이터를 각각 수치 데이터와 이미지 데이터로 변환하여 모델의 학습을 진행하였다.

2. 이론적 배경

2.1 데이터 세트

데이터 세트는 kaggle에서 제공하는 deep-voice 데이터 셋과 ASVspoof 2019 데이터 셋을 사용하였다. deep-voice 데이터 셋은 Barack Obama, Joe Biden 등 8명의 유명인의 실제 음성과 변환된 음성을 담고 있으며 변환된 음성들은 Retrieval-based Voice Conversion을 통해 변조되었다. 56개의 FAKE, 8개의 REAL 데이터로 구성되어 있다.

ASVspoof 2019 데이터 셋은 2019년도 ASVspoof Challenge에서 사용하기 위해 구성된 데이터로 Voice Cloning Toolkit(VCTK2)를 기반으로 생성된 표준 다중 화자의 음성 합성 원본 데이터와 이를 Text To Speech(TTS), Voice Conversion(VC), Voice Synthesis(VS), Tacotron2, WaveNet 등의 음성 위조 알고리즘을 사용해 위조한 위조 음성 데이터로 구성되어 있다. Training, Development, Evaluation set으로 구성되어 있으며, Training set에는 2,580개의 Bonafide, 22,800개의 Spoof, Development set에는 2548개의 Bonafide, 22,296개의 Spoof, Evaluation set에는 7,355개의 Bonafide, 63,882개의 Spoof로 이루어져 있다.

2.2 음성 특징 추출

딥러닝에서의 음성처리에서는 음성 자체를 그대로 학습에 이용하는 경우는 드물다. 음성의 샘플링을 더 잘게 할수록 데이터가 매우 높은 차원을 가지게 되며, time domain에서의 음성 데이터는 amplitude와 phase의 정보를 모두 가지고 있지만 인간의 귀는 amplitude만 인식하기 때문이다. 따라서 음성 특징 추출 기법을 사용해야 하는데, 본 프로젝트에서는 Mel-

spectrogram과 MFCC를 사용하였다.

2.2.1 Mel-spectrogram

Mel-spectrogram은 특히 음성 인식, 음악 분석과 같은 분야에서 널리 사용되는 오디오 신호를 시각화하고 분석하는 매우 유용한 도구다. Mel-spectrogram은 시간, 주파수, 인간의 청각 특성을 반영하는 정보를 결합하여 효과적으로 신호를 분석할 수 있다. Mel-spectrogram을 추출하는 과정은 Pre-Emphasis를 통해 고주파 성분을 강조하고 신호를 프레임으로 나누고 윈도우를 적용하는 Windowing을 거친 프레임에 대해 STFT를 바탕으로 얻은 각 주파수 spectrum에 Mel-filter bank를 적용하여 Mel-scale로 변환하고 logarithm한 결과를 쌓는다.

Mel-filter bank는 인간의 청각 특성을 반영하여 주파수를 Mel-scale로 변환하고 non-linear 특성을 반영하는 logarithm한 결과들을 쌓으면 Mel-spectrogram이 된다.

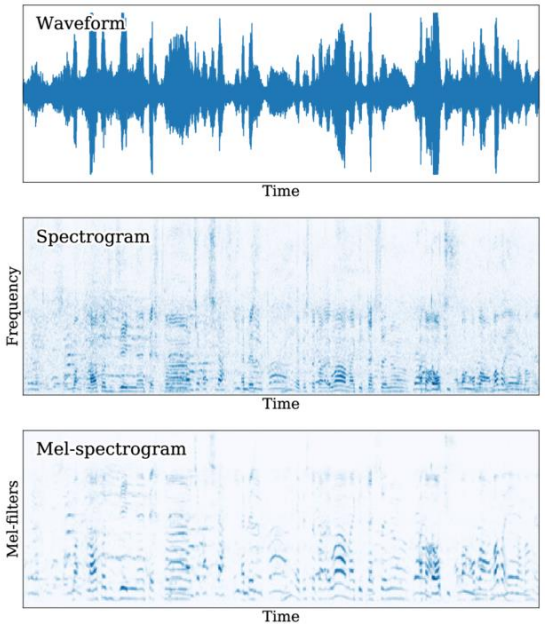


그림 1 Spectrogram과 Mel-spectrogram

2.2.2 MFCC

MFCC(Mel Frequency Cepstral Coefficients)는 음성 신호의 중요한 특징을 추출하는 데 사용된다. 2차원 Mel-spectrogram 정보를 1차원 정보로 변환하여 효율적으로 분석할 수 있도록 하는 feature vector이다. MFCC를 추출하는 과정은 Mel-spectrogram을 구하는 과정을 진행한 뒤 DCT 연산을 수행하여 주파수끼리의 상관관계가 De-correlate된 MFCC 특징 값을 추출한다. 일반적으로 12~13개의 계수를 사용한다.

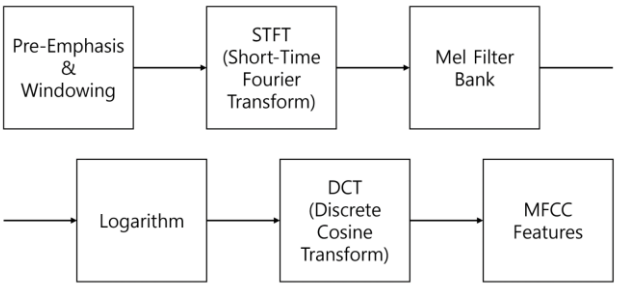


그림 2 MFCC 추출 과정

3. 모델링

다양한 딥러닝 모델들을 비교하기 위해 deep-voice 데이터 세트를 이용하여 커스텀한 CNN, ANN 모델의 학습 결과 Val accuracy가 일정한 모습을 보였다. 이유는 데이터셋 부족이었는데 deep-voice 데이터는 fake: 53, real: 8로 test set로 8:2 split을 하면 test set에서 real data가 1개가 되기 때문에 항상 같은 결과가 나온 것이다. 그래서 deep-voice 데이터 세트를 이용할 때는 5초 단위로 나누어 데이터를 증강시켜 총 60배 정도 증가시켜 사용했다.

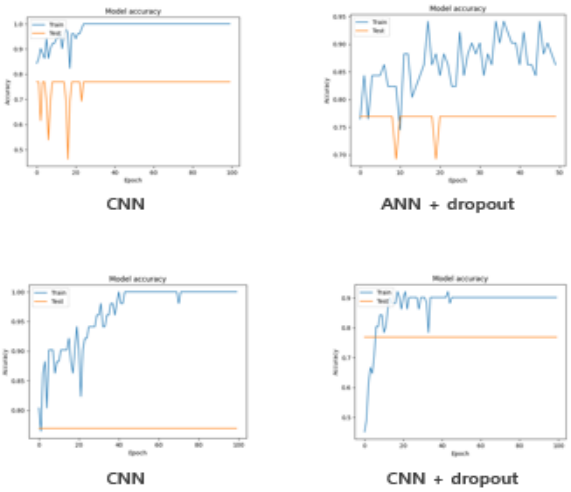


그림 3 데이터 증강 전의 학습 결과(정확도)

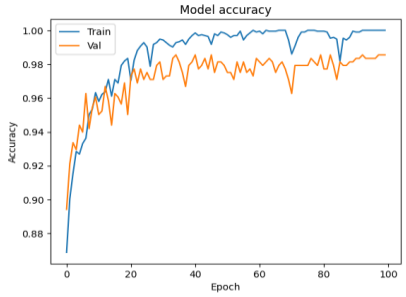
3.1 ANN, DeepVoice

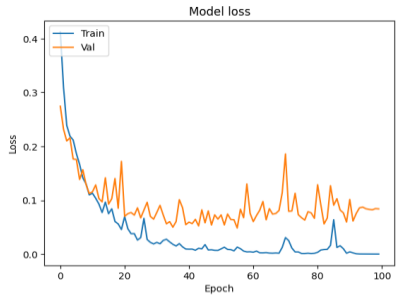
첫번째로, DeepVoice 데이터셋과 ANN모델을 사용하여 모델을 구성하였다. Keras의 Sequential 모델을 사용하여 3개의 은닉층, 출력층으로 구성된 신경망을 정의하였다. 데이터셋에서 MFCC를 추

출한 특징값이 입력으로 주어진다. 각 은닉층은 64개의 뉴런을 갖고 있으며, 활성화 함수로는 ReLU를 사용하였다. 출력층은 2개의 뉴런을 가지고 활성화 함수로 sigmoid를 사용하여 출력에서 최종 클래스 분류를 수행하였으며, Adam optimizer와 binary cross entropy loss를 사용하여 모델의 가중치를 update하였다. 학습 결과 훈련 데이터와 검증 데이터 모두 일정 수준 이상의 높은 정확도를 유지하였으며 loss도 훈련 데이터와 검증 데이터가 모두 감소하나 약 40 epoch부터는 미세하게 과적합 현상을 보였다.

```
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()
```

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






```
checkpoint_cb = keras.callbacks.ModelCheckpoint('ANN.keras',
                                                save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=2,
                                                  restore_best_weights=True)

model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics  = 'accuracy')
history = model.fit(xtrain, ytrain, epochs=50, batch_size=3, validation_data=(xtest, ytest), verbose=1,
                    )
```

그림 4 ANN 모델과 학습 결과

성능 향상을 위해 모델의 가중치를 저장하는 ModelCheckpoint 콜백, 일정 임계값 이상의 성능 향상을 보이지 않으면 중단하는 EarlyStopping 콜백을 정의하였으나 콜백 없이도 학습이 원만하게 수행이 된다고 판단하여 사용하지는 않았다.

여기서 모델을 좀 더 개선하기 위해 각 layer에 Dropout layer를 추가해주었다. Dropout Layer는 활성화 함수를 통해 얻은 중간 출력 값을 무작위로 삭제하여 overfitting을 방지해준다. 이 모델은 0.2의 Dropout 비율을 설정해주었다. 학습 결과 훈련 Dropout의 영향으로 초기에는 학습 데이터보다 검증 데이터의 정확도가 더 높게 측정되는 underfitting 현상이 발생했으나 epoch이 진행됨에 따라 이러한 문제가 해결되었으며, loss에서의 overfitting은 사라지고 검증 데이터의 loss가 안

정화되는 모습을 보였다.

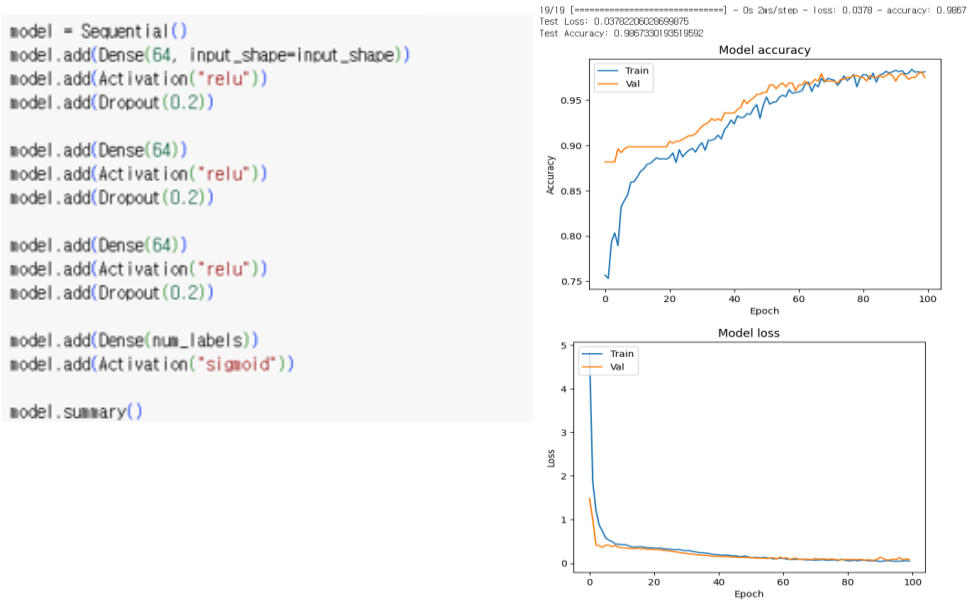
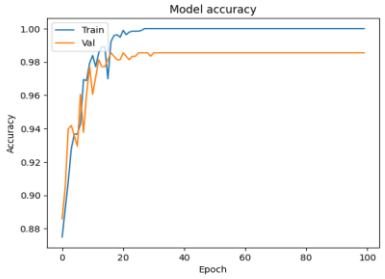


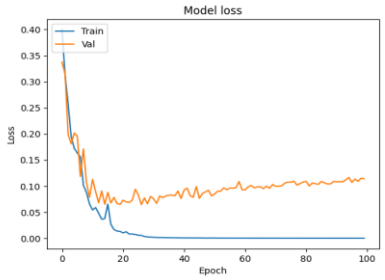
그림 5 Dropout 레이어 추가한 ANN 모델

3.2 CNN, DeepVoice

두번째로 같은 DeepVoice 데이터의 MFCC를 이용하여 CNN 모델을 구성하였다. CNN 모델에 입력할 수 있도록 samples, timesteps, features의 3차원 형태로 변환하여 데이터를 전처리를 해준 뒤, 1차원 컨볼루션 레이어(Conv1D)와 맥스 풀링 레이어(MaxPooling1D)를 3번 적용하는데, 각 컨볼루션 레이어는 64개의 필터와 relu 활성화 함수, 크기 3의 커널을 사용하여 입력 데이터를 처리했고, 맥스 풀링 레이어는 풀링 크기 2로 다운샘플링하여 특징 맵의 크기를 줄였다. 그 다음에는 Flatten 레이어로 2D 특징 맵을 1D 벡터로 변환하여 2개의 Dense layer를 거친다. 추가적으로 과적합을 방지하기위해 0.3 비율의 Dropout 레이어를 추가하였다.

```
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()
```



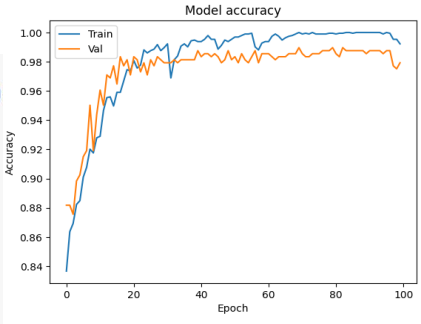



```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)  # 원하는 학습률로 설정

model.compile(optimizer='adam', loss = 'sparse_categorical_crossentropy', metrics  = 'accuracy')
history = model.fit(xtrain, ytrain, epochs=100, batch_size=3, validation_data=(xtest, ytest), verbose=1,)
                    #callbacks=[early_stopping_cb])
```

그림 6 CNN 모델과 학습 결과

학습 결과 훈련 데이터와 검증 데이터의 정확도가 모두 안정적으로 높았으나 또 다시 loss에서 과적합 현상이 발생하여 Dropout(0.5)를 추가하여 손실을 안정적으로 만들어주었다.

```
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"))
```



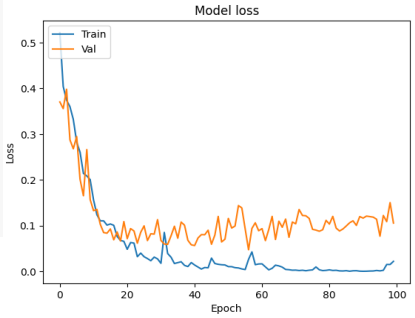


그림 7 Dropout 레이어 추가한 CNN 모델

3.3 ANN, ASVspoof2019

ASVspoof2019 데이터셋과 ANN모델을 사용하여 모델링을 하였다. 특징값은 Mel-spectrogram을 사용하였으며, Flatten을 사용해 입력 데이터를 1차원 벡터로 평탄화한 뒤 1024, 512, 256,1의 뉴런을 가지는 4개의 Dense 레이어 층을 구성하였다. 각 레이어 층 후에는 ReLU 활성화 함수가 적용되며, 마지막 층에서는 Sigmoid 함수를 사용하여 출력을 0과 1 사이의 값으로 조정하고 이진 분류를 수행하였다.

아래는 반대로 MFCC를 추출하여 Simple ANN 모델에 Dropout까지 더한 모델이다. Sequential 모델을 사용하여 3개의 Dense layer와 2개의 0.5 Dropout layer, 마지막 출력 활성화 layer는 sigmoid layer로 구성하였고, Adam 옵티마이저와 손실 함수로는 binary cross entropy를 사용하였다. 그래프를 보면 두 모델 모두 초기에 큰 손실 감소를 보였으며, MFCC의 경우 검증 손실이 안정적으로 낮은 수준을 유지하였으나 Mel-spectrogram의 경우 약간의 과적합 현상을 보였는데, 이미지 데이터를 평탄화함으로써 인접 픽셀 간의 연관성이 중요한 이미지의 특성을 고려하지 못하여 그런 것으로 예상된다.

```python
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
```

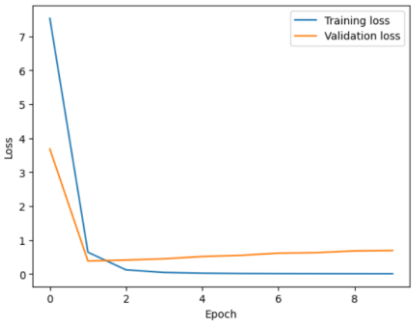


그림 8 ANN 모델과 학습 결과(Mel-spectrogram)

```python
# 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)
])
```

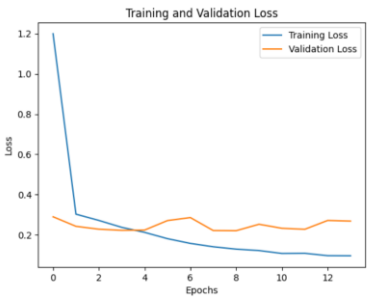


그림 9 ANN 모델과 학습 결과(MFCC)

3.4 CNN, ASVspoof2019

동일한 데이터셋 ASVspoof2019와 2D CNN 모델을 이용한 모델로, 데이터를 이미지화 하기 위해 Mel-spectrogram을 사용하였다. 두 개의 합성곱 레이어와 풀링 레이어를 연결하여 이미지 데이터의 특성을 추출하였다. Fully-connected layer에서는 먼저 64x 47x47 크기의 특성 맵을 1024 크기의 벡터로 변환 후, 1024, 512, 256 3개의 Dense layer로 특징을 추출하였다. 각 Linear 층 후에는 ReLU 활성화 함수, 마지막 층에서는 Sigmoid 함수를 사용하였다.

아래는 MFCC를 이용하여 1D CNN 모델에 Dropout을 더한 모델이다. Sequential 모델을 사용하여 2개의 합성곱 레이어, 풀링 레이어, Dropout 레이어를 연결하여 특징을 추출한 뒤 Flatten 레이어를 거쳐 데이터를 1차원으로 평탄화하고 Sigmoid 함수를 사용하여 분류를 수행하였다.

그래프를 보면 두 모델 모두 초기에는 손실이 감소하나, 1D CNN 모델의 경우 Dropout을 적용했음에도 불구하고 과적합이 되는 모습을 보였는데, 이 이유는 MFCC의 차원에 비해 모델이 너무 복잡해서 그런 것으로 예상된다. 따라서 Dropout을 추가하는 것보다는 모델을 단순화하면 더 좋은 결과를 낼 것으로 예상된다.

```python
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
```

그림 10 2D CNN 모델과 학습 결과(Mel-spectrogram)

```python
# 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')
])
```

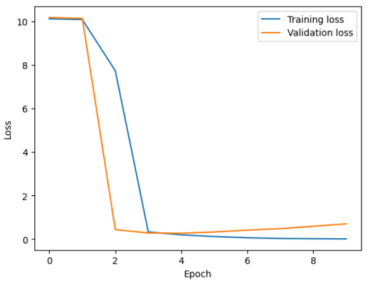


그림 11 1D CNN 모델과 학습 결과(MFCC)

3.5 ResNet, ASVspoof2019

ResNet101 기반 커스텀 모델은 ResNet101 아키텍처를 기반으로 하며, 주로 복잡한 이미지 데이터에서 효과적인 특성 추출을 목표로, 사전 훈련된 가중치를 활용하여 빠르게 학습할 수 있도록 설계된 모델이다. ResNet202 기반 커스텀 모델은 더 깊은 레이어 구조를 가지며, ResNet202를 활용해 더욱 정교한 이미지 분석이 가능하다. 이 두 모델은 추가적인 커스텀 레이어를 포함하여 특정 작업에 더 적합하도록 조정할 수 있다. 먼저 커스텀 모델 1은 ResNet101의 사전 훈련된 모델을 기반으로, 기존의 layer에 Adaptive AvgPool2d, Flatten, Linear 완전 연결 레이어를 더해 특성을 추출하는 과정을 구현하였다. 아래의 모델은 ResNet202의 사전 훈련된 모델을 기반으로, ResNet202의 기본 구조에 이어서 커스텀 레이어를 추가하여 더 깊은 특성을 추출하도록 구현하였다. 또한 Dropout 레이어를 추가해 과적합을 방지하는 동시에 일반화 능력을 높였다.

두 모델의 훈련 결과 그래프를 살펴보면 ResNet101 기반, ResNet202 기반 커스텀 모델 모두 손실의 변동이 조금씩은 있으나 전반적으로 각각 0.2, 0.1근처의 매우 작은 손실 값을 유지하면서 안정적인 학습 경향을 갖는 것을 볼 수 있다

.

```python
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
```

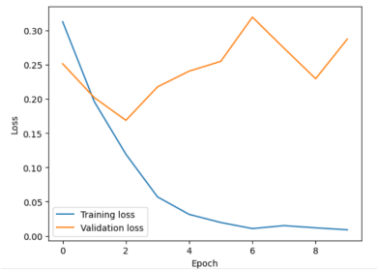


그림 12 ResNet101 기반 커스텀 모델

```python
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
```

그림 13 ResNet101 기반 커스텀 모델

3.6 LSTM, ASVspoof2019

LSTM 모델은 시계열 데이터나 순차적 데이터 처리에 최적화된 구조를 가진다. LSTM 모델은 Keras의 Sequential 모델로 구성했으며, 128개의 유닛을 가지며 Dropout rate는 0.5이다. 이 레이어는 입력 데이터의 시간적 특성을 포착하는 데 중요한 역할을 한다. Dropout 레이어 후에는 두 개의 Dense 레이어를 연결하여 특징을 추출하였다. 그래프를 보면 초기 에폭에서 손실이 급격히 감소하며, 검증 손실도 0.1~0.2의 매우 낮은 손실을 유지하는 것으로 보아 학습이 안정적으로 진행되었음을 알 수 있다.

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

그림 14 LSTM 기반 커스텀 모델

3.7 결과 분석

ROC curve는 거짓일 때 참으로 판단할 확률인 FPR(False Position Rate)과 참일 때 참일 확률인 TPR(True Position Rate)의 비례관계를 나타낸 곡선으로, 두 class의 확률분포 사이에 threshold를 어느 지점으로 설정하느냐에 따라 점이 plot되는 위치가 달라지며 두 클래스 분포의 교집합이 작아질수록 ROC 커브는 좌상단에 더 가까워지게 된다.

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

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

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

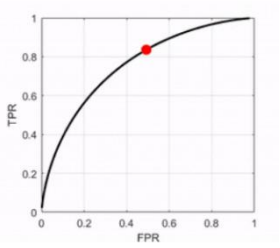


그림 15 ROC curve

ROC curve의 아래 면적을 AUC(Area Under Curve)라고 하며, 이 면적은 FPR 대비 TPR의 비율을 의미한다. AUC가 1에 가까울수록 두 클래스의 분류가 잘 일어나며 따라서 이 면적을 가지고 score를 계산할 수 있다.

이 ROC score를 기반으로 평가한 결과는 전반적으로 pre-training 모델이 커스텀 모델에 비해 우세한 성능을 보였다. DeepVoice 데이터 셋의 경우 Random Forest와 XGBoost가 98.7, 99.3으로 높은 성능을 보였으며, ASVspoof2019 데이터셋을 이용했을 때 ResNet 모델이 Mel Spectrogram과 MFCC 데이터 모두에서 뛰어난 성능을 보여주며 특히 ResNet202 모델이 가장 효과적이었다. LSTM 모델도 MFCC 데이터에서 높은 성능을 나타냈으며, Simple 모델에 Dropout을 추가한 경우, 일부에서는 성능 향상을 보이지만 일관성은 떨어질 수 있다는 결과를 얻었다.

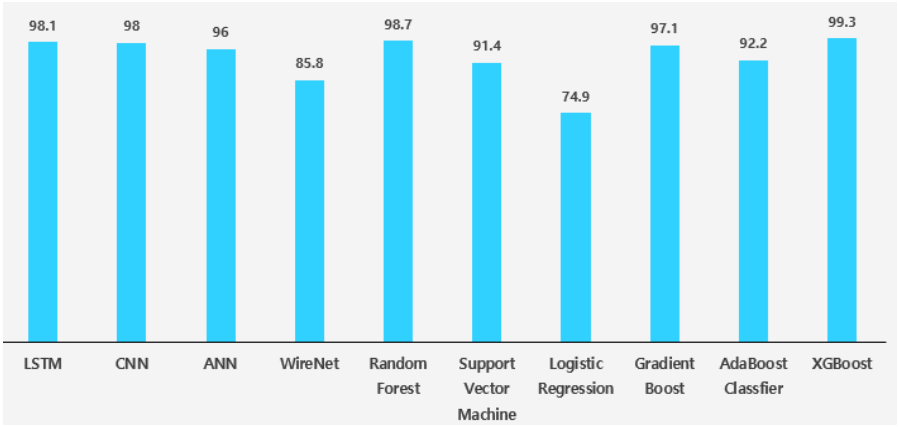


그림 16 모델들의 성능 평가(DeepVoice)

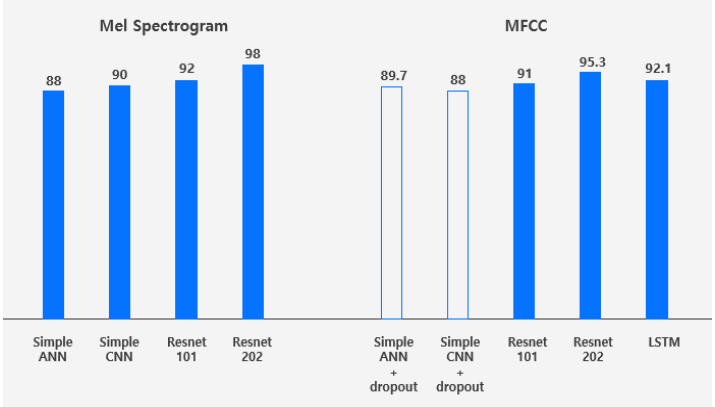


그림 17 모델들의 성능 평가(ASVspoof2019)

또한 마지막으로 실제 가수 Charile puth의 노래인 hero와 목소리를 통해 ai로 생성한 노래인 퀸카를 특징추출을 진행하고 테스트한 결과 그림 18과 같이 진짜와 가짜를 잘 구분하는 모습을 확인할 수 있다.

관련 코드는 그림 19와 같다.

```
예측 결과: 가짜 (Fake)
예측 값: 0.021316364407539368
[[0.02131636]]
```

```
예측 결과: 진짜 (Real)
예측 값: 0.07330457866191864
[[0.07330458]]
```

그림 18 실제노래와 AI노래를 입력하여 테스트한 결과

```python
# 오디오 파일 로드
audio_file_path = "charlie puth - hero.m4a"
# audio_file_path = "charlie puth - queencard.m4a"
audio, sr = librosa.load(audio_file_path)

# MFCC 추출
mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=10)

target_shape = (17, 10)

if mfccs.shape[1] > target_shape[0]:
    mfccs = mfccs[:, :target_shape[0]]
else:
    pad_width = target_shape[0] - mfccs.shape[1]
    mfccs = np.pad(mfccs, ((0, 0), (0, pad_width)), mode='constant')

mfccs = mfccs.reshape((1, target_shape[0], target_shape[1]))

predictions = model.predict(mfccs)

threshold = 0.5
prediction_value = predictions[0][0]

if prediction_value < threshold:
    print("예측 결과: 진짜 (Real)")
else:
    print("예측 결과: 가짜 (Fake)")

print(f"예측 값: {prediction_value}")
print(predictions)
```

그림 19 Test code

4. 결론

이상으로, 변조된 음성을 탐지하기 위해 2개의 데이터셋과 여러 모델을 이용하여 음성 위조 탐지 시스템을 구현하고 그 성능을 비교 분석해보았다. 실험 결과는 상당히 좋은 성과를 보여주었으며, 전반적으로 복잡한 모델을 사용할 때 성능이 좋은 경향을 보였다. 특히 ResNet202와 같은 pre-training 모델과 Random Forest, XGBoost와 같은 부스팅 모델의 경우 각 98, 98.7, 99.3으로 성능이 가장 좋게 나타났다. 이를 통해 본 프로젝트에서 높은 성능을 위해서는 복잡한 모델이 효과적임

을 느꼈고 증강된 데이터 셋에서의 결과가 더 좋았기 때문에 좋은 데이터 전처리의 중요성을 알게 되었다.

하지만 프로젝트를 진행하며 여러 한계점을 느꼈다. 첫번째로 데이터셋의 부족이다. 실제 현장에서 발생하는 다양한 음성 위조 케이스가 아닌 ASVspoof의 경우 대화용으로 제작된 세트로 학습된 모델에 실제 케이스의 데이터를 넣었을 때 정확도가 떨어지고 영어기반의 데이터이기에 언어가 변경됨에 따라 적응도가 떨어지는 모습을 볼 수 있었다.

두번째로 음성 신호처리에 대한 배경지식의 한계다. 음성 신호처리에 대한 배경지식이 충분하지 못하여 데이터 전처리와 모델링 과정에서 최적의 방법 및 다양한 방법을 고려하지 못했다.

세번째로 개발 환경의 한계다. 코랩 환경에서 개발을 진행하다보니 큰 데이터셋을 처리하거나 복잡한 모델을 훈련시키는 과정에서 메모리 부족과 같은 문제가 생겼다.

더 쾌적한 개발환경에서 더 많은 데이터셋과 다양한 모델을 사용한 추가 연구를 통해 더욱 정교하고 신뢰도가 높은 음성 위조 탐지 시스템을 개발할 수 있을 것이다. 최근 인공 지능 기술의 급속한 발전으로 인해 음성 합성과 변조 기술이 크게 향상된 상황에서 보이스피싱과 같은 범죄를 식별하고 막는데 큰 도움이 될 것이며 은행, 금융 기관과 같이 보안이 중요한 분야에서 안정성과 신뢰성을 높이는 데 크게 기여할 것으로 기대된다.

References

[1] 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]

[2] 합성된 음성과 발화 음성에 대한 차이 연구 Audio deepfake detection 기술 개발
(survey:https://arxiv.org/pdf/2308.14970.pdf)

[3] Deepfake Audio Detection via MFCC Features Using Machine Learning
(https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9996362)

[4] 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)

[5] https://velog.io/@gangjoo/ML-%ED%8F%89%EA%B0%80-F1-Score%EC%99%80-ROC-AUC

코드 자료 및 관련 내용

https://github.com/lexxsh/Deepfake-Audio