신호 및 시스템
프로젝트 결과 보고서

위급 음성 및 비명 감지를 통한
위험 상황 탐지 시스템

최종수정일: 2024년 12월 14일 (토)

작성자: 소프트웨어학부 소속 2023245100 박김한결
소프트웨어학부 소속 2022245117 황선준
소프트웨어학부 소속 2023245092 이우성

목차

1. 제목 및 목적

A. 제목

위급 음성 및 비명 감지 시스템 개발

B. 연구 배경

여성 안심 귀갓길은 한국에서 여성들의 안전한 귀가를 지원하기 위해 도입된 주요 치안 강화 정책 중 하나로, 특히 야간 시간대의 범죄 예방과 신속한 구조 요청을 목적으로 한다. 현재 대부분의 시스템은 귀갓길에 설치된 전봇대나 특정 지점에 마련된 비상벨을 통해 위급 상황을 신고하는 방식으로 운영되고 있다. 사용자가 비상벨 버튼을 누르면 즉각적으로 관제센터나 경찰에 경고 신호가 전달되어 긴급 대응이 이루어진다.

그러나 기존의 버튼 기반 비상벨 시스템은 실제 위급 상황에서 여러 가지 운영상의 한계를 드러내고 있다. 한겨레(2023)에 의하면, 여성안심 귀갓길에 비상벨이 아예 설치되지 않은 경우도 적지 않으며, 설치된 경우에도 관리가 제대로 이루어지지 않아 작동 여부가 불확실한 경우가 많다. 또한, 갑작스러운 위협 상황에서는 피해자가 비상벨의 위치를 인지하지 못하거나, 공포와 충격으로 인해 벨을 눌러 도움을 요청하는 것이 어려울 수 있다. 이러한 문제들로 인해 기존 비상벨 시스템은 예기치 못한 상황에 대한 실질적인 대응 수단으로 한계가 있다는 지적이 제기되고 있다.

이러한 문제점을 해결하기 위한 대안으로 음성 및 소리 분석 기술을 활용한 자동 신고 시스템이 주목받고 있다. 이 시스템은 비명, 도움을 요청하는 소리, 유리 깨지는 소리 등 일반적인 환경 소리와 구별되는 비정상적 소리를 실시간으로 감지하고 이를 자동으로 신고하는 방식이다. 인공지능(AI) 기반 음향 인식 알고리즘은 특정 소리 패턴을 학습하고 분류하여 위급 상황 발생 시 즉각적으로 관제센터에 알림을 보낼 수 있다.

음향 기반 위급 상황 감지 시스템은 사용자가 의도적으로 작동시킬 필요가 없다는 점에서 버튼 기반 시스템과 차별화된다. 즉, 위험을 인지하고 자연스럽게 발생하는 소리를 자동으로 포착하여 반응하기 때문에 더 신속하고 직관적인 대응이 가능하다. 이는 특히 어두운 골목길이나 인적이 드문 지역에서 범죄 예방과 조기 대응에 효과적이다.

C. 목적

이 연구는 여성 안심 귀갓길에서의 기존 비상벨 시스템의 한계를 보완하고, 위험 상황에서 더 직관적이고 신속하게 대응할 수 있는 안전 시스템을 구축하는 데 기여한다. 비명 소리와 같은 비정상적인 음향 신호를 자동으로 감지하여 위급 상황을 실시간으로 판단하는 비명 감지 시스템을 개발하고자 한다.

2. 프로젝트 내용

A. 프로젝트 개요

본 프로젝트는 비명 감지 시스템을 구축하기 위해 다양한 딥러닝 모델을 비교하고 최적의 구조를 도출하는 것을 목표로 한다. 주요 비교 대상 모델은 CNN-GRU(게이트 순환 유닛), CNN-LSTM(장단기 메모리 신경망), CNN+Transformer 모델이다. 이러한 모델의 성능과 안정성을 체계적으로 평가하여 응급 상황 탐지에 가장 적합한 구조를 설계하고자 한다.

모델 성능 비교 실험은 음성 데이터의 특성을 반영한 정량적 및 정성적 평가를 포함한다. 모델 평가 지표로는 정확도(Accuracy), 재현율(Recall), 정밀도(Precision), F1-score 등을 활용하며, 응답 속도와 처리 효율성 또한 주요 고려 대상이 된다.

학습 데이터로는 주로 Kaggle의 비명 데이터셋을 사용할 계획이다. 수집된 데이터는 품질과 노이즈 수준에 따라 분류하고, Fourier 변환(FFT)과 멜 스펙트로그램 변환을 적용하여 모델 학습에 적합한 형태로 전처리한다. 음성 신호는 주파수 및 시간 영역의 특징을 반영하여 모델의 학습 성능을 최적화한다.

B. 프로젝트 진행 과정

1. 데이터 수집 및 준비

본 프로젝트의 데이터 수집은 각 팀원별로 역할을 분담하여 진행하였다.
박김한결: 유튜브를 제외한 다양한 웹사이트 및 Kaggle과 같은 대형 공개 데이터베이스를 중심으로 비명 음성 관련 자료 탐색
이우성: 프로젝트 진행에 참고할 수 있도록 허준영 조교님께 자문
기타 팀원: 유튜브, 영화 등의 영상 매체에서 직접 비명 소리를 추출하고, 비교를 위한 일반 음향(대조군 음성) 자료를 수집
이렇게 해서 모인 데이터를 통합하여 학습에 이용했다.

본 연구에서는 다양한 출처로부터 확보한 비명 음성 데이터 중에서도, 신뢰성과 다양성을 갖춘 Kaggle 데이터셋을 중점적으로 활용하였다. 선택된 데이터셋은 다음과 같다.

Scream Dataset (Kaggle)

URL: https://www.kaggle.com/datasets/sanzidaakterarusha/scream-dataset/data
URL: https://www.kaggle.com/datasets/kimuliarnoldmuliisa/screams-dataset
(두 데이터셋의 많은 내용이 중복되어 있어 함께 기술한다.)
데이터셋 설명: 해당 데이터셋은 각종 장면 및 환경에서 포착된 비명 소리를 포함하며, 비명음 분류 모델 학습에 유용한 다양한 음향 특성을 제공한다.

Human Screaming Detection Dataset (Kaggle)

URL: https://www.kaggle.com/datasets/whats2000/human-screaming-detection-dataset
데이터셋 설명: 인간의 비명소리 검출에 초점을 맞춘 데이터셋으로, 비명 소리를 식별하기 위한 특징 추출 및 분류 알고리즘 개발에 유용한 오디오 샘플을 다수 포함하고 있다.

2. 데이터 전처리

데이터 전처리 파트의 모든 코드는 조장 황선준이 구현했다. 아래는 전체 코드 GitHub 링크와 작성한 코드에 대한 세부 설명이다.

2-1. 데이터 로더

코드 링크 :
https://github.com/Fourier-Frontier/Deliverables/blob/main/DataProcessing/dataloader.py

데이터 로더의 전체 코드를 요약하면, 오디오 파일을 탐색하고, Fourier 변환을 수행하여 주파수 데이터를 추출하며, 각각의 결과를 '.npy' 파일로 저장하는 오디오 처리 파이프라인이다.

세부 설명:

```
1  import os
2  import numpy as np
3  import librosa
4  import scipy
```

os: 파일 경로와 디렉터리를 관리할 수 있는 모듈
numpy: 수치 연산 라이브러리
librosa: 오디오 처리 라이브러리
scipy: 신호 처리 및 수학적 계산 라이브러리

```
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
TRAIN_DIR = os.path.join(BASE_DIR, 'train data')
TEST_DIR = os.path.join(BASE_DIR, 'test data')
RAWTRAIN_DIR = os.path.join(BASE_DIR, 'rawtrain')
RAWTEST_DIR = os.path.join(BASE_DIR, 'rawtest')
```

경로 설정 부분은 절대 경로 기반으로 데이터 디렉토리를 정의한다.

BASE_DIR: 현재 파일의 절대 경로를 정의
TRAIN_DIR, TEST_DIR: 학습 및 테스트 데이터 경로
RAWTRAIN_DIR, RAWTEST_DIR: 변환된 파일을 저장할 디렉토리.

```python
def find_audio_files(input_dir):
    audio_files = []
    for root, dirs, files in os.walk(input_dir):
        for file in files:
            if file.lower().endswith(('.wav', '.mp3', '.ogg')):
                file_path = os.path.join(root, file)
                audio_files.append(file_path)
                print(f"Audio file found: {file_path}")
    return audio_files
```

오디오 파일 검색 함수로 주어진 디렉토리와 모든 하위 폴더에서 오디오 파일(.wav, .mp3, .ogg)을 찾고, 오디오 파일의 경로를 수집하고 콘솔에 출력한다.
os.walk()는 디렉토리를 순회하며 파일과 폴더 구조를 탐색한다.

```python
def perform_fourier_transform_n_save_npy_file(file_path, output_path, FS=16000):
    try:
        print(f"Loading audio file: {file_path}")
        data, sample_rate = librosa.load(file_path, sr=None)
        if data is None or len(data) == 0:
            print(f"Failed to load audio (empty data): {file_path}")
            return None
        print(f"Audio loaded successfully: {file_path}, Sample rate: {sample_rate}, Data length: {len(data)}")

        if sample_rate != FS:
            data = librosa.resample(data, orig_sr=sample_rate, target_sr=FS)

        _, _, freq_data = scipy.signal.stft(data, fs=FS, window='rectangular', nperseg=512, noverlap=0, nfft=512)
        magnitude = np.abs(freq_data)  # 절댓값 계산

        np.save(output_path, magnitude)
    except Exception as e:
        print(f"Error during file processing: {file_path}, Error message: {str(e)}")
        return None
```

Fourier 변환 및 **.npy** 파일 저장 함수로 주어진 오디오 파일을 Fourier 변환 후 .npy 파일로 저장한다.

아래와 같은 방식으로 동작한다.

1. 파일 로딩: librosa.load()로 파일 로드.
2. 재샘플링: 샘플링 레이트가 다르면 librosa.resample()로 16000Hz로 변환.
3. **STFT** 변환: scipy.signal.stft()로 주파수 변환 수행.
4. 저장: np.save()로 변환된 데이터를 .npy 형식으로 저장.

```python
def process_audio_files(input_dir, output_dir):
    audio_files = find_audio_files(input_dir)
    for file_path in audio_files:
        relative_path = os.path.relpath(file_path, input_dir)
        output_path = os.path.join(output_dir, relative_path).replace(relative_path[-4:], '.npy')
        os.makedirs(os.path.dirname(output_path), exist_ok=True)

        perform_fourier_transform_n_save_npy_file(file_path, output_path)
```

오디오 파일 처리 파이프라인으로 주어진 디렉토리 내 모든 오디오 파일을 찾아 Fourier 변환을 수행하고 저장한다. 원본 경로를 기반으로 변환된 파일의 상대 경로 계산하고 디렉토리 구조를 유지한 채 .npy 파일을 생성한다.

```python
def main():
    os.makedirs(RAWTRAIN_DIR, exist_ok=True)
    os.makedirs(RAWTEST_DIR, exist_ok=True)

    print("Processing train data...")
    process_audio_files(TRAIN_DIR, RAWTRAIN_DIR)

    print("Processing test data...")
    process_audio_files(TEST_DIR, RAWTEST_DIR)

    print("All tasks completed successfully.")
```

메인 함수로 프로젝트를 초기화 및 오디오 파일 처리 작업을 실행한다.

os.makedirs(): 폴더 생성.
process_audio_files(): 학습 및 테스트 데이터 처리.
print(): 진행 상황을 콘솔에 출력.

```python
if __name__ == "__main__":
    main()
```

그리고 이 스크립트를 통해 **main()** 함수가 실행된다.

2-2. 오디오 데이터 분류 모델

코드 링크 **:**
https://github.com/Fourier-Frontier/Code/blob/main/modiver/CNN_Transformer.py

세부 설명 **:**
이 모델은 오디오 파일을 입력으로 받아 CNN과 Transformer 계층을 통해 위급상황시의 비명을 감지하는 것을 목표로 한다.

```
class AudioDataset(Dataset):
    def __init__(self, data_dir):
        self.chunk_frame = 10
        self.data, self.labels = self.load_data(data_dir)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        str_idx = random.randrange(0, len(self.data[idx]) - self.chunk_frame)
        data = self.data[idx][:, str_idx:str_idx + self.chunk_frame]
        return data, self.labels[idx]
```

데이터셋 클래스로 PyTorch 데이터셋 클래스를 상속하여 오디오 데이터를 관리한다.

__init__ 메서드: 데이터 경로에서 데이터를 로드하고 프레임 크기를 설정
__len__ 메서드: 데이터셋 크기를 반환
__getitem__ 메서드: 데이터와 레이블을 반환하며, 임의의 프레임을 잘라내어 모델에 입력

```
def load_data(self, data_dir):
    data = []
    labels = []

    for file_path in glob.glob(f"{data_dir}/*.npy", recursive=True):
        try:
            array = np.load(file_path)
            label = 1 if "scream" in file_path else 0
            data.append(array)
            labels.append(label)
        except Exception as e:
            print(f"Error loading file {file_path}: {e}")

    labels = np.array(labels)
    return data, labels
```

데이터 로딩 함수로 .npy 형식의 오디오 데이터를 로드하고 라벨을 설정한다.
glob.glob()을 사용해 모든 오디오 파일을 탐색한다.
파일 경로에 "scream"이 포함된 경우 라벨을 1로 설정하고, 그렇지 않으면 0으로 설정한다.

```python
class CNNTransformer(nn.Module):
    def __init__(self, input_size=257, num_classes=1):
        super(CNNTransformer, self).__init__()
        self.cnn = nn.Sequential(
            nn.Conv1d(input_size, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2),
            nn.Conv1d(128, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2)
        )
        self.transformer_layer = nn.TransformerEncoderLayer(d_model=32, nhead=2)
        self.transformer = nn.TransformerEncoder(self.transformer_layer, num_layers=1)
        self.fc = nn.Linear(32, num_classes)
```

CNN-Transformer 모델 클래스

다음과 같은 계층을 거친다.

CNN 계층: 오디오 데이터를 처리하며 특징을 추출.
Transformer 계층: 시퀀스 정보를 학습하여 더 나은 표현을 생성.
최종 선형 계층: 클래스 예측 수행.

모델의 순전파 메서드

```python
def forward(self, x):
    x = self.cnn(x)                    # CNN 계층 통과
    x = x.permute(2, 0, 1)             # Transformer 입력 형식으로 변환
    x = self.transformer(x)            # Transformer 계층 통과
    x = x.mean(dim=0)                  # 평균 풀링
    x = self.fc(x)                     # 분류 결과 도출
    return x
```

모델의 순전파 메서드다.

CNN 처리 후: (batch, feature, seq_len)
변환 후: (seq_len, batch, feature)로 변경해 Transformer 입력 형식으로 변환한다.
Transformer 출력: 평균 풀링으로 최종 특징을 추출한다.

```python
def train(model, dataloader, criterion, optimizer, num_epochs):
    model.train()
    for epoch in range(num_epochs):
        total_loss = 0
        for data, labels in dataloader:
            data = torch.tensor(data, dtype=torch.float32)
            labels = torch.tensor(labels, dtype=torch.long)
            optimizer.zero_grad()
            outputs = model(data)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        print(f"Epoch [{epoch + 1}/{num_epochs}], Loss: {total_loss / len(dataloader):.4f}")
```

모델 학습 함수다.

역할: 모델 학습 루프 실행.
손실 계산: 교차 엔트로피 손실을 계산하고 역전파한다.
모델 업데이트: optimizer.step()으로 매개변수를 업데이트한다.

```python
# 데이터 로더 준비
batch_size = 32
data_dir = './dataset'

# 데이터셋 및 데이터로더 생성
dataset = AudioDataset(data_dir)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

# 모델 초기화
input_size = 257
num_classes = 1
num_epochs = 10
learning_rate = 0.001

model = CNNTransformer(input_size=input_size, num_classes=num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# 모델 학습 및 평가
train(model, dataloader, criterion, optimizer, num_epochs)
evaluate(model, dataloader)

# 모델 저장
torch.save(model.state_dict(), "cnn_transformer_model.pth")
print("Model saved as cnn_transformer_model.pth")
```

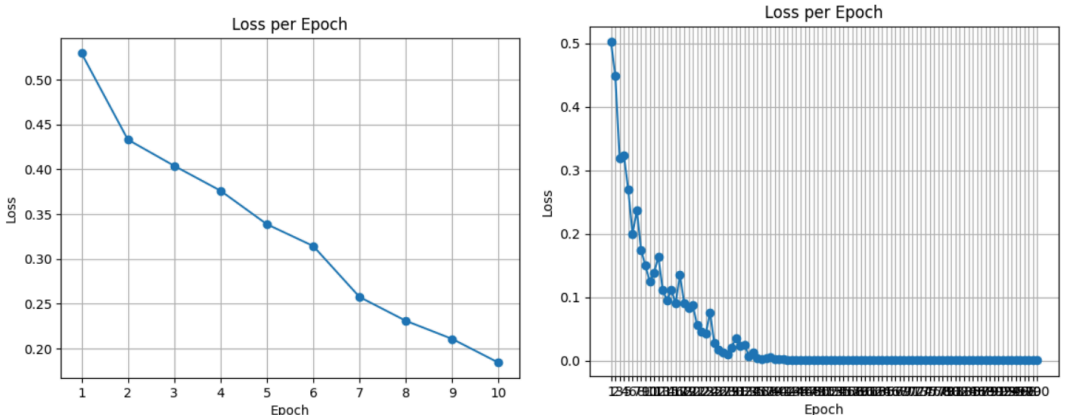
모델 실행 및 저장 코드로 CNN-Transformer 기반 오디오 분류 시스템을 구축하고. 오디오 데이터를 전처리하고 모델 학습 및 평가를 수행하며, 최종 모델은 파일로 저장된다.

3. 모델 설계 및 실험

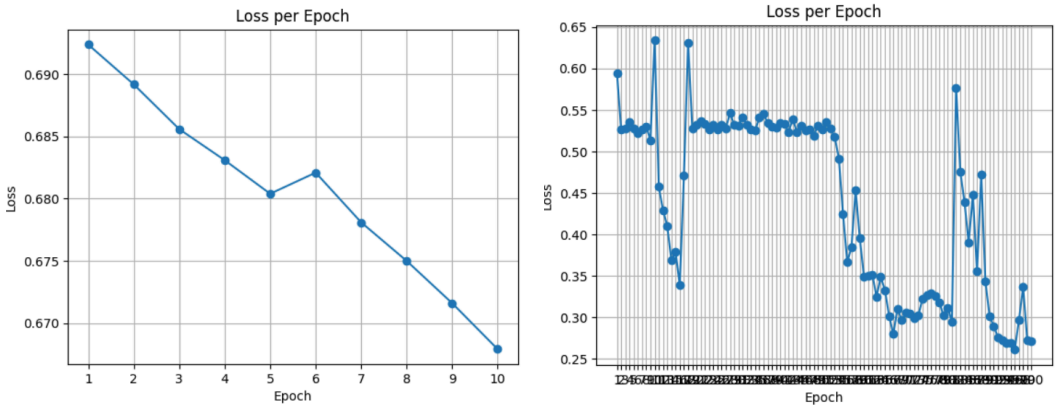
참고한 논문 "Development of scream detection system with large-scale scream dataset”(2024)을 바탕으로 비명 감지에 적합한 모델 세 개(CNN-GRU, CNN-LSTM, CNN+Transformer)를 선정하고 가장 성능이 좋은 모델을 검증하기 위해 실험을 진행했다.

CNN-GRU, CNN-LSTM, CNN+Transformer 구조를 비교했다. 각 모델의 초기 학습 속도, 손실 감소 추이, 안정성을 분석한 결과, CNN+Transformer가 가장 우수한 성능을 보였다.

좌측 그래프는 황서림 대학원생님께 피드백을 받기 이전의 결과고 우측은 피드백 이후의 결과다. 좌측 그래프를 확인하면, 피드백 이전에는 손실 값이 수렴하기 이전에 학습을 종료했고, 데이터 전처리 과정에서 리샘플링 단위가 너무 큰 문제가 있었다. 피드백 이후, 문제를 해결한 우측의 그래프를 확인하면 전반적인 정확도가 개선된 걸 확인할 수 있다.

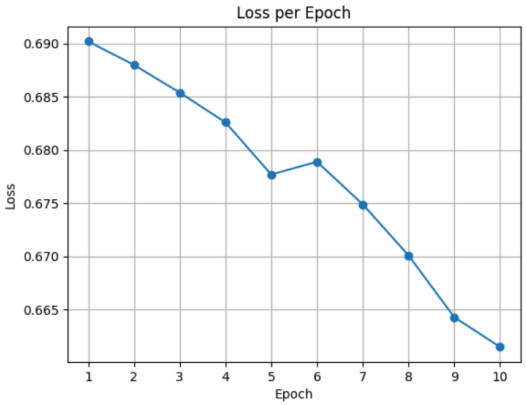


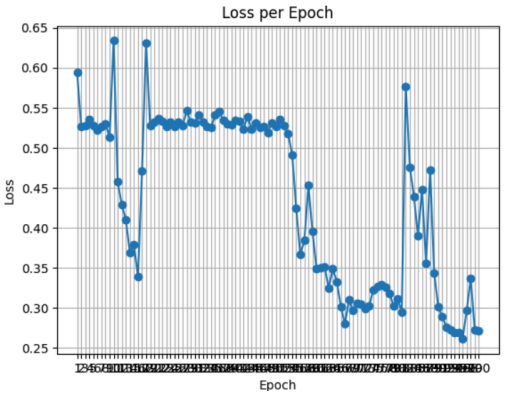
CNN+Transformer 모델은 초기 손실이 0.3577~0.5018 사이로 시작하였으며, 학습 초반 빠르게 손실이 감소하여 40 에포크 이후 안정화되었다. 최종적으로 손실 값은 0.0002~0.0003으로 매우 낮은 수준에 도달했으며, 정확도는 99.92%~100%로 기록되었다. 이는 CNN과 Transformer의 결합이 초기 수렴 속도와 학습 안정성에서 강점을 보이며, 변동성을 효과적으로 제어할 수 있음을 나타낸다.



CNN-GRU 모델은 초기 손실이 0.5783~0.6956 사이로 다소 높았으며, 데이터의 양이 적어서, 학습 도중 손실 변동성이 크고 안정화되기까지 시간이 오래 걸렸다. 최종 손실 값은

0.0387~0.5110, 정확도는 49.88%~77.60%로, 데이터 특성에 따라 성능 편차가 큰 모습을 보였다.





CNN-LSTM 모델은 초기 손실이 0.5947~0.6982로 시작되었고, 학습 데이터의 양이 적어서 손실 감소 추이가 일관되지 않은 경향을 보였다. 학습 후반부에서 손실이 완만히 감소하며 안정화되었으나, 일부 모델은 마지막 에포크까지 손실이 안정적으로 낮아지지 않았다. 최종 손실 값은 0.2717~0.5752, 정확도는 65.06%~87.02%로 비교적 높은 수준이었으나, CNN+Transformer보다 학습 안정성이 낮았다.

4. 실험 결과 요약

피드백 전후 수렴 에포크 경향

데이터 전처리 단계에서 황서림 대학원생님의 의견을 반영하여, 모델이 충분히 수렴할 때까지 epoch 수를 늘렸다.(10->40) ,
그리고 데이터 전처리 과정에서 리샘플링 단위를 짧게 조정했다.

모델	이전 수렴 에포크	이후 수렴 에포크
CNN + Transformer	10+	40
CNN-GRU	10+	80
CNN-LSTM	10+	90

***** 피드백 이전 모델은 **loss** 값이 수렴하기 전에 학습을 종료함
* 너무 많은 학습을 진행하는 것도 모델의 성능에 악영향을 미친다. 그래서 수렴 에포크를 정해 필요 이상의 학습을 제한했다.피드백 전후 정확도 비교

모델	이전 정확도	이후 정확도
CNN + Transformer	97.56%	99.92%
CNN-GRU	58.54%	77.60%
CNN-LSTM	60.86%	87.02%

***** 세 모델 모두 피드백 이후 정확도가 향상된 걸 확인할 수 있다.
***** 데이터 양이 적어서 CNN + Transformer를 제외한 다른 모델에서는 정확도가 낮게 나왔다.

최종 모델별 수렴 값

모델	수렴 손실 값	수렴 에포크
CNN + Transformer	0.0002	40
CNN-GRU	0.0387	80

CNN-LSTM	0.2717	90

* 결과적으로 **CNN + Transformer** 모델이 가장 우수한 성능을 보였다.

5. 시스템 개발 및 테스트

시스템 개발 및 테스트 파트는 조원인 이우성이 담당했다.

실시간 예측 및 감지 시스템

딥러닝 모델(CNN+Transformer)과 Google Speech-to-Text AP를 이용하여 실시간으로 음성을 녹음하고, 특정 키워드 및 비명을 탐지하여 위험 상황을 판단하는 시스템이다.

코드 링크:
https://github.com/Fourier-Frontier/Deliverables/blob/main/analysis_code/predictScreaming.py

세부 설명:

```
# 탐지할 키워드 정의
keywords = ["help", "please help", "help me", "도와주어", "도와주세요"]
```

API 를 이용하여 감지할 텍스트를 정의한다. API에 대한 내용은 아래에 후술하였다.

```
# JSON 키 파일 경로 설정
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "/Users/iuseong/Downloads/signal & system/methodical-tea-442914-r5-3dca061d585f.json"

# Google Speech-to-Text 클라이언트 초기화
client = speech.SpeechClient()
```

Google Speech-to-Text API를 활용하여 녹음된 음성을 텍스트로 변환하고, 특정 키워드("help", "도와주세요" 등)가 포함되어 있는지 탐지한다.

```
# 음성 녹음 함수
def record_audio(duration=2, sampling_rate=44100):
    print("Recording...")
    audio_data = sd.rec(int(duration * sampling_rate), samplerate=sampling_rate, channels=1, dtype='float32')
    sd.wait()
    file_path = "recorded_audio.wav"
    sf.write(file_path, audio_data, sampling_rate)
    print("Recording saved.")
    return file_path, sampling_rate
```

음성 녹음 함수다.
sounddevice.rec를 통해 지정된 duration 동안 녹음하고 샘플링 레이트를 설정한다.
결과적으로 녹음 데이터를 WAV로 저장하고 저장된 파일 경로와 샘플링 레이트를 반환한다.

```python
# Google Speech-to-Text API 함수
def transcribe_audio(file_path):
    try:
        with open(file_path, "rb") as audio_file:
            content = audio_file.read()

        audio = speech.RecognitionAudio(content=content)
        config = speech.RecognitionConfig(
            encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,
            sample_rate_hertz=44100,
            language_code="en-US",  # 기본 언어
            alternative_language_codes=["ko-KR"]  # 보조 언어 설정
        )

        response = client.recognize(config=config, audio=audio)
        transcribed_text = " ".join([result.alternatives[0].transcript for result in response.results])
        return transcribed_text

    except Exception as e:
        print(f"Error during transcription: {e}")
        return ""  # 빈 문자열 반환
```

텍스트 탐지 함수로 **Google Speech-to-Text API**를 사용하여 .wav 파일을 텍스트로 변환한다.
예외가 발생하면 빈 문자열을 반환한다.

API 요청 생성:
speech.RecognitionAudio로 음성 데이터를 API가 이해할 수 있는 포맷으로 변환.
speech.RecognitionConfig로 샘플링 레이트, 언어 설정.

```python
# 키워드 탐지 함수
def detect_keywords(text):
    print(f"Transcribed text: {text}")
    for keyword in keywords:
        if keyword.lower() in text.lower():
            print(f"Keyword detected: {keyword}")
            return True
    print("No matching keyword found.")
    return False
```

키워드 탐지 함수로 텍스트에서 특정 키워드(예: "help")를 탐지한다.
변환된 텍스트를 출력.
키워드 탐지시 True 탐지 실패시 False를 반환한다.

```python
# CNN+Transformer 모델 정의
class CNNTransformer(nn.Module):
    def __init__(self, input_size=257, num_classes=1):
        super(CNNTransformer, self).__init__()
        self.cnn = nn.Sequential(
            nn.Conv1d(input_size, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2),
            nn.Conv1d(128, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2)
        )
        self.transformer_layer = nn.TransformerEncoderLayer(d_model=32, nhead=2)
        self.transformer = nn.TransformerEncoder(self.transformer_layer, num_layers=1)
        self.fc = nn.Linear(32, num_classes)

    def forward(self, x):
        x = self.cnn(x)
        x = x.permute(2, 0, 1)
        x = self.transformer(x)
        x = x.mean(dim=0)
        x = self.fc(x)
        return x
```

CNN+Transformer 모델로 음성 데이터를 입력받아 특정 클래스(예: 비명 여부)를 예측한다.

CNN 계층:
주파수 데이터를 처리하여 로컬 특징을 추출. Conv1d와 MaxPool1d를 사용.
Transformer 계층:
CNN 결과를 시퀀스 형태로 변환하여 Transformer 입력으로 사용.
시퀀스 정보를 학습하여 더 풍부한 표현 생성.
Fully Connected Layer:
최종 출력으로 클래스 확률(예: 비명 여부) 계산.

```python
# Screaming 여부 판단 함수 (결과 시각화 포함)
def process_audio_file(file_path, sampling_rate, model, threshold=0.5):
    print(f"Loading audio file: {file_path}")
    try:
        audio_data, sr = sf.read(file_path)
        if sr != sampling_rate:
            print(f"Resampling from {sr} to {sampling_rate}")
            audio_data = librosa.resample(audio_data, orig_sr=sr, target_sr=sampling_rate)

        audio_data = audio_data / np.max(np.abs(audio_data))
        magnitude = np.abs(stft(audio_data, fs=sampling_rate, nperseg=512, noverlap=256)[2])
        magnitude = magnitude.T
        input_data = torch.tensor(magnitude, dtype=torch.float32).unsqueeze(0).permute(0, 2, 1)

        with torch.no_grad():
            output = model(input_data)
            predictions = torch.sigmoid(output).numpy()
            print(f"Model predictions (sigmoid values): {predictions}")

        is_screaming = np.any(predictions > threshold)
        print(f"Is screaming detected (any > {threshold}): {is_screaming}")

        # 결과 시각화
        plt.figure(figsize=(12, 8))

        # 오디오 신호 플롯
        plt.subplot(2, 1, 1)
        plt.plot(audio_data[:sampling_rate])
        plt.title("Audio Signal")
        plt.xlim(0, len(audio_data))
        plt.ylim(-1, 1)

        return is_screaming

    except Exception as e:
        print(f"Error during screaming detection: {e}")
        return False
```

Screaming 여부 판단 함수로 녹음된 오디오 파일에서 비명 여부를 판단하며 아래와 같은 과정을 통해 동작한다.

오디오 데이터를 읽고, 필요 시 샘플링 레이트 변환.
데이터를 정규화하고, 주파수 데이터를 STFT로 계산.
STFT 결과를 모델에 적합한 텐서 형식으로 변환.

모델 추론:
모델 출력에서 sigmoid 값을 계산하여 비명 여부 판단.
임계값(threshold=0.5) 이상이면 비명으로 판단.

결과 시각화:
녹음된 신호를 그래프로 출력.

```python
# WAV 파일 재생 함수
def play_siren(wav_file_path):
    try:
        wave_obj = sa.WaveObject.from_wave_file(wav_file_path)
        play_obj = wave_obj.play()
        play_obj.wait_done()  # 재생이 끝날 때까지 대기
    except Exception as e:
        print(f"Error playing siren sound: {e}")
```

WAV파일 재생 함수로 경고음 파일(.wav)을 읽고 재생한다.

```python
# 메인 함수
if __name__ == "__main__":
    try:
        # 모델 초기화
        model = CNNTransformer(input_size=257, num_classes=1)
        model.load_state_dict(torch.load("cnn_transformer_model.pth"))
        model.eval()

        # 음성 녹음
        recorded_audio_path, sampling_rate = record_audio(duration=2, sampling_rate=44100)

        # Screaming 여부 판단
        s_bool = process_audio_file(recorded_audio_path, sampling_rate, model)
        print(f"s_bool (Screaming Detected): {int(s_bool)}")

        # 도움 요청 키워드 탐지
        transcribed_text = transcribe_audio(recorded_audio_path)
        h_bool = detect_keywords(transcribed_text)
        print(f"h_bool (Help Detected): {int(h_bool)}")

        # 최종 결과 출력
        print(f"s_bool: {int(s_bool)}, h_bool: {int(h_bool)}")

        # 위험상황 출력 및 경고음 재생
        if s_bool or h_bool:
            print("위험상황")
        else:
            print("안전")

    except KeyboardInterrupt:
        print("Process interrupted by user.")
    except Exception as e:
        print(f"Unexpected error: {e}")
```

main 함수에서는 전체 워크플로를 실행한다.

테스트

테스트 영상은 아래 링크에서 확인 가능하다.

테스트 영상 링크: https://www.youtube.com/watch?v=9sI7QYvPGFU
이 영상에서는 시스템의 성능을 검증하기 위해 다양한 상황을 재현하여 테스트를 진행했다.. 구체적으로, "help"와 같은 도움 요청 음성, 비명 소리, 그리고 단순히 큰 소리를 비교하여 시스템이 각각의 상황을 어떻게 인식하고 구별하는지 확인했다.

C. 프로젝트 결과 및 의의

이번 프로젝트는 비명 감지 시스템 개발의 전 과정을 철저히 수행하며, 데이터 전처리, 다양한 모델 설계 및 비교, 최적 구조 선정, 성능 검증 등의 단계를 통해 실질적인 성과를 거두었다. 특히, 딥러닝 모델을 기반으로 한 음향 인식 시스템의 가능성을 실험적으로 입증하며, 향후 발전 가능성을 제시했다.

---

1. 높은 정확도와 모델 안정성

CNN+Transformer 기반 비명 감지 시스템은 최종적으로 99.92%의 높은 정확도와 0.0002의 낮은 손실 값을 기록하며 가장 우수한 성능을 보였다. 초기 학습에서는 데이터의 제한과 전처리 문제로 인해 손실 값이 불안정하게 변화하는 모습을 보였으나, 리샘플링 조정과 충분한 에포크 설정을 통해 안정적인 학습이 가능해졌다. 이를 통해 비명과 일반 소리를 효과적으로 구별할 수 있는 모델을 구현하였다.

2. 데이터 처리와 모델 설계의 기여

이번 프로젝트는 데이터 처리와 모델 설계 과정에서 명확한 비교와 분석을 수행했다. Kaggle 및 기타 출처의 데이터를 활용하여 적절히 정리하고, Fourier 변환과 같은 전처리 기법을 적용해 데이터의 특징을 모델이 잘 학습할 수 있도록 준비했다. CNN, CNN-GRU, CNN-LSTM, CNN+Transformer 모델을 비교한 결과, CNN+Transformer 모델이 안정성과 정확도 면에서 가장 적합하다는 결론을 도출하였다. 이를 통해 향후 유사한 프로젝트에서 참고할 수 있는 결과를 제시하였다.

3. 응급 상황 감지 시스템과 확장 가능성

이번 시스템은 비상벨을 직접 작동할 수 없는 상황에서도 비정상적 소리를 자동으로 감지할 수 있다는 점에서 기존 안전 시스템의 한계를 보완한다. 이를 통해 어두운 골목길이나 사람의 왕래가 드문 지역에서 위험을 감지하고 즉각적으로 대응할 가능성을 보여주었다. 향후 다양한 환경과 음향 데이터를 추가로 확보하고, 비명 외에 다른 비정상적 소리(예: 유리 깨지는 소리, 교통사고 소리 등)를 감지하는 기능을 추가함으로써 응급 상황 탐지 시스템의 범용성을 확대할 수 있는 가능성을 열어두었다.

결론적으로, 이번 프로젝트는 딥러닝 기반의 음향 분석 기술이 응급 상황 감지에 유용하게 적용될 수 있음을 실험적으로 입증하였다. 비록 데이터 수집과 학습 환경에서의 한계가 있었지만, 이를 극복하고 높은 성능을 갖춘 모델을 구현했다는 점에서 의미가 있다. 이는 실제 환경에서의 응용 가능성을 검토할 수 있는 기초 자료로 활용될 수 있을 것이다.

3. 고찰

1. 모델 선택 과정

CNN+Transformer 모델은 다른 모델(CNN-GRU, CNN-LSTM)과 비교하여 다음과 같은 우수성을 보였다.

CNN-GRU 모델: 초기 손실이 상대적으로 높고 학습 변동성이 크며, 수렴 속도가 느려 최종 손실 값과 정확도가 불안정했다. CNN-GRU 모델의 최종 정확도는 49.88%~77.60%로 큰 편차를 보였으며, 실용적인 안정성을 확보하기 어려웠다.

CNN-LSTM 모델: 수렴 속도와 학습 안정성이 CNN-GRU보다 나았지만, 손실 변동 폭이 여전히 크고 학습 후반부에서도 성능이 완전히 수렴하지 않았다. 최종 정확도는 65.06%~87.02%로 비교적 높았지만, 모델 간 성능 편차가 커 CNN-Transformer에 비해 신뢰성이 낮았다.

CNN-Transformer 모델: 초기 학습 속도가 빠르고, 손실 값이 안정적으로 감소하며 정확도 또한 99.92%~100%로 높은 성능을 기록했다. 손실 변동 폭이 작아 실질적인 응급 상황 탐지 시스템에 적합한 안정성을 보였다.

2. 모델 수렴 최적화 과정

프로젝트 초기, 학습 속도를 고려해 10 에포크(epoch)에서 모델 학습을 종료했으나, 데이터가 충분히 수렴하지 않은 상태에서 종료되어 정확도가 낮았다.
CNN+Transformer 모델의 초기 정확도는 97.56%로 높았지만, 손실 변동성이 커 학습 결과가 불안정했다. 대학원생 멘토와의 피드백을 통해 적절한 수렴 에포크를 확인하는 과정이 추가되었다. 테스트 결과, 40 에포크에서 손실이 0.0002로 안정화되었고, 최종 정확도도 99.92%로 향상되었다.
이 과정에서 데이터의 손실 추이를 관찰하며 적절한 수렴 지점을 판단하는 방법을 배우는 등, 프로젝트 수행 과정에서 모델 최적화의 중요성을 경험했다.

3.적은 데이터 양

가장 큰 문제는 데이터의 양이 적다는 점이었다. 이는 데이터 수집이 가능한 소스가 제한적이었기 때문에 발생한 한계이다. 특히, 비명 소리나 특정 상황에서 발생하는 소리를 수집하기 위해서는 실제 상황을 재현하거나 공개된 데이터셋에 의존해야 했는데, 이 과정에서 다양한 샘플을 확보하는 데 어려움이 있었다. 이러한 데이터 부족은 결과적으로 높은 정확도를 확보하는 데 제약으로 작용하였다.

이러한 한계를 보완하기 위해 Google Speech API를 활용하여 음성 데이터를 추가로 확보하고 처리함으로써, 더 다양한 샘플을 학습 데이터에 포함할 수 있도록 하였다.

4. 데이터 및 학습 조건의 제약

소리 데이터의 특성과 전처리 과정에서 발생한 제약이 있었다. 데이터셋에는 1,000개 이상의 음성 파일이 포함되어 있었으며, 이 중 비명 소리 이외에도 음악, 환경 소음 등 다양한 소리가 섞여 있었다. 특히, 하나의 데이터 파일에 **3**초 음악 소리 → **2**초 비명 소리 → **4**초 음악 소리와 같은 형태로 비명 소리가 특정 구간에만 포함된 경우, 이를 정밀하게 전처리하여 비명 소리만 분리하는 작업은 시간과 리소스의 한계로 인해 현실적으로 불가능했다.

결과적으로, 모든 데이터를 세밀하게 전처리하기보다는 일정 시간 동안 소리를 측정한 후, 그 **10**초 간의 구간에 비명 소리가 포함되었는지를 확인하는 방식으로 대체하였다. 이 방식은 실시간성에서 다소 손해를 보지만, 제한된 시간 안에 실질적인 결과를 도출하기 위해 실효성을 확보하는 현실적인 해결책이었다.

이 접근법은 단기적인 해결책으로는 효과적이었으나, 다음과 같은 한계를 내포한다:

1. 실시간성의 손실: 특정 구간(10초)의 데이터를 분석하는 방식은 실시간 응답성을 보장하지 못한다. 이는 응급 상황에서 즉각적인 반응이 중요한 애플리케이션에서는 제한적인 활용 가능성을 의미한다.
2. 정확도 저하의 가능성: 비명 소리가 매우 짧거나 소음이 비명과 유사한 경우, 잘못된 판단이 이루어질 가능성이 있다.

5. 아두이노 연계 보류

이번 프로젝트에서는 아두이노 키트를 활용해 시스템을 구동하려는 시도를 진행했으나, 여러 가지 기술적 어려움에 직면했다.

먼저, 아두이노 마이크에서 지나치게 많은 잡음이 발생하여 음질 확보가 어려웠다.
다음으로, 아두이노로 녹음한 음성 데이터를 컴퓨터로 전송하는 과정에 대한 이해가 부족했다.
이를 해결하기 위해 아두이노 여러 대를 병행 사용하거나 추가적인 음성 녹음 부품을 구입해 실험을 진행했으나, 전송 과정에서 음질 저하와 실시간 데이터 전송 구현의 난항이 지속되었다.
이러한 문제는 현재 단계에서 해결이 어려운 부분으로 판단되었으며, 황서림 대학원생님도 당장 적용하기에는 기술적 제약이 크다는 의견을 주셔서 이를 받아들여 해당 방식을 보류하게 되었다.

4. 결론

CNN+Transformer 기반 비명 감지 시스템은 응급 상황 감지의 높은 정확도와 안정성을 입증했다. 프로젝트를 통해 음성 분석 모델의 설계와 구현에 대한 심층적인 이해를 높였으며, 실시간 응용 가능성을 강화하기 위한 추가적인 연구 필요성을 확인하였다.
본 프로젝트는 CNN, CNN-GRU, CNN-LSTM, CNN+Transformer 모델을 비교 분석하여 음향 기반 응급 상황 감지 시스템의 최적 구조를 도출하는 데 성공하였다. CNN+Transformer 모델은 신속한 학습 수렴과 높은 예측 정확도를 보이며, 99.92%의 모델 정확도를 보였다. 이로써 CNN과 Transformer의 결합이 음성 데이터의 시퀀스 및 지역적 패턴을 효과적으로 학습할 수 있음을 확인할 수 있었다.
모델 성능 평가 결과, CNN-GRU와 CNN-LSTM은 학습 초기 변동성이 크고 안정화까지 시간이 오래 걸렸으며, CNN+Transformer는 안정적인 손실 감소와 높은 정확도를 유지하며 우수한 결과를 보였다. 이를 통해 음향 신호 분석에서 시계열 정보와 로컬 패턴 학습이 중요한 요소임을 재확인할 수 있었다.
또한, 실시간 음성 인식 및 감지 시스템은 Google Speech-to-Text API와 결합해 특정 키워드를 탐지할 수 있는 기능을 포함하고 있으며, 이로 인해 사용자 인터페이스와 시스템 응답 속도가 개선되었다. 실시간 테스트 결과 비명을 비롯한 특정 음성 신호 탐지에서 정확도와 반응 속도가 기대 수준을 충족하였다.
프로젝트 진행 중 주요 한계는 데이터 양과 품질의 제한, 아두이노 기반 하드웨어 연계의 어려움이었다. 다양한 환경에서 발생하는 비명 소리를 충분히 수집하지 못했으며, 특정 데이터는 전처리 과정에서 소음을 제거하는 데 어려움을 겪었다. 데이터 수집 범위를 확대하고, 노이즈 제거 알고리즘을 고도화함으로써 이러한 한계를 극복할 수 있을 것이다.
또한, 아두이노 기반 실시간 녹음 시스템 구현은 하드웨어 성능 및 소음 필터링 문제로 인해 프로젝트에 포함되지 못하였으나, 향후 연구의 중요한 개선 요소로 남아 있다.
결론적으로, 본 연구는 딥러닝과 음성 인식 기술을 결합하여 응급 상황 감지 시스템의 개발 가능성을 입증했으며, 향후 더 많은 데이터와 개선된 하드웨어 연계를 통해 공공 안전 분야에서 실질적인 활용 가능성을 기대할 수 있다. 이를 통해 공공장소, 스마트 홈, 범죄 예방 시스템 등 다양한 분야에서 음성 기반 감지 시스템의 적용 가능성을 확대할 수 있을 것이다. 이러한 성과는 미래의 응급 상황 대응 시스템 설계와 개발에 중요한 기술적, 학술적 기여를 할 것으로 보인다.

5. 참고 문헌

1. Kim, Y., Jang, D., Lee, J. (2024). "Development of scream detection system with large-scale scream dataset," *Proc. of IEEE ICTC 2024*, Seoul, Korea. IEEE, pp. 590-593.
URL: https://ictc.org/media?key=site/ictc2024/abs/D3-2.pdf

우리 연구에서는 논문 "Development of scream detection system with large-scale scream dataset"을 참고했다. 해당 논문에서는 11,921개의 대규모 비명 데이터를 수집하여 CNN, GRU, LSTM, Transformer 등을 포함한 다양한 모델의 성능을 비교하였다. 특히, CNN과 Transformer를 결합한 모델이 97.64%의 정확도와 0.9822의 F1 Score를 기록하며 가장 우수한 성능을 보였음을 입증하였다.

이 논문의 연구 결과는 우리가 CNN-LSTM, CNN-GRU, CNN+Transformer 세 가지 모델을 기준으로 실험을 진행하고 최종적으로 CNN+Transformer 구조를 선택하는 데 중요한 참고자료로 작용하였다. Transformer는 시퀀스 데이터 간의 복잡한 관계를 학습하며, CNN과 결합하여 지역적 패턴과 전역적 관계를 모두 효과적으로 파악할 수 있다. 이를 바탕으로, 본 연구에서도 비명 감지에 최적화된 모델 설계를 목표로 삼았다.

2. 곽진산, 박지영 (2023). "보안등 하나 없는 안심귀갓길 459곳…“오히려 피해 다녀요”," 한겨레.
URL: https://www.hani.co.kr/arti/society/society_general/1093643.html

해당 기사에서는 안심귀갓길의 보안등 미설치와 관리 부실 문제, 시설물의 부족과 훼손, 안심벨의 식별 어려움 등을 지적하며, 이러한 문제들이 범죄 예방 효과를 제한하고 있다는 점을 강조하고 있다. 특히, 시설물이 제대로 관리되지 않거나 주요 시설물이 아예 설치되지 않은 경우가 많아, 실제로 범죄 상황에서 예방 및 대응에 한계가 있음이 드러났다.

6. 참여자 역할 및 협업 방식

A. 참여자 역할

학과	학번	이름	역할
소프트웨어학부	2022245117	황선준	팀리더, 딥러닝 모델 개발, 데이터 전처리, 전체적인 피드백, 보고서 피드백 + 작성 보조
소프트웨어학부	2022245097	김현서	PPT제작,발표준비
소프트웨어학부	2023245100	박김한결	결과 시각화, 보고서 작성
소프트웨어학부	2023245092	이우성	모델을 이용한 실시간 시스템 개발, 서기
소프트웨어학부	2023245043	최상혁	ppt 제작, 발표

B. 협업 방식

효율적인 협업을 위해 체계적인 회의 진행 방식과 자료 관리를 통해 프로젝트를 운영하였다. 다음은 협업 과정에서 채택한 주요 회의 진행 방식과 관리 방안이다.

효율적인 협업을 위해 체계적이고 명확한 회의 진행 방식을 도입하여 프로젝트를 운영하였다.

1. 정기 주간 회의 및 기록 관리
    - 프로젝트의 일관성을 유지하고 진행 상황을 점검하기 위해 매주 수요일 오후 1시에 정기적으로 주간 회의를 진행했다.
    - 회의에서는 팀원들이 지난주에 수행한 작업을 공유하고, 다음 주 동안 각자 진행해야 할 활동과 역할을 구체적으로 설정하였다. 이를 통해 팀원 간 책임 분담을 명확히 하고 작업의 연속성을 유지하였다.
    - 모든 회의에서 회의록 작성을 필수적으로 진행하여, 논의된 주요 내용, 결정된 사항, 팀원별 활동 계획을 기록하고 공유하였다
2. 진행 상황 및 자료 관리
    - 프로젝트의 진행 상황과 자료를 체계적으로 관리하기 위해 GitHub Organization(https://github.com/Fourier-Frontier)을 적극 활용했다.
    - 모든 프로젝트 자료는 아래와 같이 GitHub에 체계적으로 저장했다:
        - 전체 소스 코드: Fourier-Frontier/Code

- 데이터셋: Fourier-Frontier/Audio-Files
- 회의록: Fourier-Frontier/Meeting Minutes Collection
- 최종 결과물: Fourier-Frontier/Deliverables

7. 소감

황선준

이번 프로젝트를 마무리하면서 우리 팀의 뛰어난 팀워크와 노력에 대해 깊은 감사를 표하고 싶습니다. 프로젝트 초반부터 지금까지 함께해 온 모든 팀원들이 각자의 역할을 충실히 수행하고, 서로를 이해하며 협력하는 모습을 통해 우리는 높은 성과를 이룰 수 있었습니다. 이 과정을 되돌아보며 우리의 팀워크가 어떻게 발전했는지를 다시금 되새겨 보고자 합니다.

처음 프로젝트를 시작할 때는 다소 막막하게 느껴졌던 부분도 있었지만, 우리 팀은 빠르게 조직적이고 체계적인 접근 방식을 확립했습니다. 프로젝트 계획을 수립하고 역할을 분담할 때, 모든 팀원이 자신의 의견을 적극적으로 제시하고 경청하면서 최적의 방향을 도출했습니다. 이러한 상호 존중과 열린 소통은 우리가 어려운 문제들을 해결하고 예상치 못한 도전 과제에도 유연하게 대응할 수 있는 기반이 되었습니다.

특히, 프로젝트 진행 중 가장 어려웠던 문제 중 하나는 예상보다 복잡했던 기술적 난제들이었습니다. 그러나 팀원 모두가 지치지 않고 해결 방안을 탐색하며 지식을 공유한 덕분에 결국 문제를 해결할 수 있었습니다. 이 과정에서 우리는 기술적 역량뿐만 아니라 문제 해결을 위한 협업과 집단 지성의 중요성을 깊이 깨달을 수 있었습니다.

또한, 주어진 일정 내에 목표를 달성하기 위해 몇몇 팀원은 자발적으로 추가적인 시간을 투자하고, 다른 팀원들의 부담을 덜어 주기 위해 나서는 모습도 자주 볼 수 있었습니다. 이러한 헌신과 책임감은 프로젝트를 성공으로 이끄는 중요한 원동력이었습니다.

결국, 우리는 다른 팀들보다 더 좋은 결과를 내며 프로젝트를 성공적으로 마무리할 수 있었습니다. 이는 결코 운이 아니라, 팀원 개개인의 노력과 팀 전체의 협력 덕분이라고 확신합니다. 각자가 가진 장점을 최대한 발휘하고, 약점을 보완해 주며, 지속적인 피드백을 통해 스스로 발전해 나가는 팀의 모습은 정말 자랑스럽습니다.

이번 프로젝트를 통해 얻은 경험은 단순히 결과물 이상의 의미를 지닙니다. 우리 팀이 보여준 헌신, 협력, 그리고 문제 해결 능력은 앞으로의 프로젝트나 도전에서도 큰 자산이 될 것입니다. 모두가 한마음으로 목표를 향해 달려왔던 시간을 기억하며, 함께한 모든 순간에 대해 깊은 감사의 마음을 전합니다.

앞으로도 지금과 같은 멋진 팀워크를 유지하며 더 큰 성과를 이루어 나가길 기대합니다. 여러분과 함께 일할 수 있어서 정말 영광이었고, 이 소중한 경험을 통해 더욱 성장할 수 있었습니다. 모두 고생 많으셨습니다. 감사합니다.

박김한결

이번 협업 과정에서 보고서 작성 전반을 총괄하며 팀의 기획안 작성에도 중요한 역할을 맡아 노력하였다. 초기 단계에서는 프로젝트의 전체적인 방향성을 설정하고 기획안을 정리하며, 자료를 구조화하는 데 집중하여 팀의 목표를 명확히 하고 작업의 효율성을 크게 높였다. 이러한 초기 기여를 통해 프로젝트의 기틀을 마련하고, 팀이 효과적으로 진행할 수 있는 기반을 마련했다는 점에서 의미 있는 성과를 거두었다.

그러나 후반부로 접어들면서 개인 일정 관리에 미흡했던 점이 드러나면서 팀원들과 약속한 보고서 완성 일정을 몇 차례나 뒤로 미뤘다. 이로 인해 팀원들에게 불편을 끼치고 협업 과정에 부담을 주게 되었지만, 다행히 제출 기간에 여유가 있어 프로젝트 완성도에는 큰 영향을 미치지 않았다. 원활한 커뮤니케이션을 통해 빠르게 결과물을 완성했기 때문이다. 이러한 경험은 팀원 간 소통과 약속의 중요성을 깊이 깨닫는 계기가 되었다.

향후에는 약속을 하기 전 충분히 검토하여 이번과 같은 불상사가 발생하는 일을 최소화하고 서로에게 부담을 주지 않는 선에서 적절한 커뮤니케이션을 통해 효율적이고 원활한 협업을 이뤄내는 데 주력할 것이다.

이우성

이번 학기 진행한 프로젝트는 제게 정말 값진 경험이었습니다. 제가 맡은 역할은 팀원들이 만든 비명 예측 딥러닝 모델을 실시간 음성 감지 시스템과 결합하는 것이었습니다. 구글 Speech to Text API를 사용해 음성을 실시간으로 텍스트로 변환하고, 이를 딥러닝 모델과 연결하여 비명 또는 구호 키워드를 탐지하는 시스템을 구현하는 작업이었습니다. 또한 프로젝트 진행 중 회의에서 서기 역할을 맡아 회의록을 작성하고, 팀원들의 의견을 정리하여 프로젝트가 원활히 진행될 수 있도록 돕는 일도 맡았습니다.
이번 프로젝트는 제가 처음으로 맡아본 실전 프로젝트였기 때문에 많은 것들을 배우고 경험할 수 있는 기회였습니다. 처음에는 여러 가지 어려움이 있었지만, 팀원들이 서로 협력하고 아이디어를 나누며 문제를 해결해 나가는 과정에서 많은 것을 배웠습니다. 특히, 비명 감지라는 주제를 다루다 보니, 현실적인 문제를 해결하는 데 필요한 기술적인 접근을 배우는 데 큰 도움이 되었고, 프로젝트를 통해 저의 신호 처리 및 시스템 구축에 대한 이해도 많이 향상되었습니다.
모든 팀원들이 각자 맡은 역할을 충실히 수행하며 열정을 가지고 프로젝트를 완수한 점이 매우 인상 깊었습니다. 각 팀원이 보여준 뛰어난 기술력과 협업 정신 덕분에 저희 팀은 예상보다 빠르고 원활하게 시스템을 구현할 수 있었습니다. 각자의 강점을 발휘하며 유기적으로 협력한 덕분에 프로젝트가 성공적으로 마무리될 수 있었고, 그 과정에서 저 역시 많이 성장할 수 있었습니다.
이번 프로젝트를 통해 배운 것은 단순히 기술적인 지식뿐만 아니라, 팀워크의 중요성, 문제 해결을 위한 창의적 접근 방법, 그리고 끈기와 열정이 얼마나 중요한지 깨닫는 계기가 되었습니다. 처음에는 막막하고 어려운 점이 많았지만, 팀원들의 도움과 협력 덕분에 모든 것이 원활하게 진행될 수 있었고, 그 결과는 매우 만족스러웠습니다.
마지막으로, 이번 프로젝트를 함께 해준 모든 팀원들에게 진심으로 감사의 말을 전하고

싶습니다. 프로젝트가 성공적으로 마무리될 수 있었던 것은 각자의 노력과 헌신 덕분입니다. 팀원들의 열정과 성실함이 없었다면 불가능했을 일이라고 생각합니다. 정말 고맙고, 여러분과 함께한 시간이 매우 소중하고 의미 있었습니다.
이 프로젝트를 통해 얻은 경험과 지식을 바탕으로 앞으로 더 많은 도전에 임할 준비가 되었고, 함께한 팀원들과의 추억도 오래도록 기억에 남을 것입니다. 다시 한번, 모든 팀원들에게 진심으로 감사의 말씀을 전하며, 앞으로도 더 많은 성과를 이루어 나가기를 기대합니다.

김현서

이번 프로젝트를 진행하며 학업적으로나 팀워크 부분에서 많은 것을 배웠다. 능력적으로 뛰어난 팀원들을 만나게 된 것은 너무나 행운이었다고 생각하고 분위기 또한 좋아서 프로젝트 큰 어려움 없이 잘 진행되어서 좋은 경험이 되었다. 개인으로서 능력이 많이 부족해서 팀 활동에 민폐를 끼치지 않을까 걱정을 했지만, 잘 이끌어주고 역할 배분을 잘해준 조장과 조원들에게 감사함을 많이 느끼며 부족한 능력 안에서 할수있는 일을 찾으며 참여한것이 개인적으로 성장하는데에 많은 도움이 될수있었다. 이번 프로젝트에서의 경험은 학생의 입장에서 앞으로도 있을 수많은 프로젝트를 진행하는 데에 단단한 초석이 될것같다.

최상혁

이번 프로젝트를 통해 한 단계 더 성장할 수 있었던 소중한 경험을 하게 되어 매우 뜻깊었습니다. 프로젝트 결과물에 대해 스스로도 만족할 만큼, 실제로 적용 가능한 시스템을 만들어낸 점에서 자부심을 느낍니다. 초기 목표였던 단순 시스템 구축을 넘어서, 더 큰 스케일로 확장할 수 있는 가능성을 확인하게 됐다고 생각됩니다. 이 점이 특히 인상 깊었습니다.

프로젝트를 진행하며 팀워크가 정말 좋았다고 생각합니다. 각 조원이 자신의 역할을 성실히 수행해주었고, 서로의 의견을 존중하며 조화롭게 협력할 수 있었습니다. 이러한 팀워크 덕분에 높은 완성도의 결과물을 만들어낼 수 있었다고 생각합니다.

또한, 프로젝트의 성공적인 마무리를 위해 함께 노력해준 모든 조원들에게 진심으로 감사의 말씀을 전하고 싶습니다. 팀원들의 열정과 헌신 덕분에 이번 프로젝트가 단순한 과제를 넘어 실제적인 가치를 지닌 시스템으로 완성될 수 있었습니다.

앞으로 이번 프로젝트에서 얻은 경험과 성과를 발판 삼아, 더 나은 시스템을 개발하고 실제 적용 사례를 만들어나가고 싶습니다. 이번 경험을 통해 많은 것을 배웠고, 앞으로도 지속적으로 성장해나가고자합니다.