가정 내 위급상황 음성 인식 모델 개발

3조 2019540040 황태연, 2019920014 김상훈, 2020540023 이명규

1 프로젝트 목표 및 시나리오

본 프로젝트의 목표는 가정 내에서 발생하는 위급상황을 인지할 수 있는 음성 인식 모델을 개발하는 것이다. 만약 이러한 모델이 만들어진다면 다음과 같은 시나리오들을 생각해볼 수 있다. (시나리오의 자세한 설명은 발표자료 참고)

1) 1인 가구, 노인 가구의 실내 위급상황에 대한 즉각적 대처

2) 특정 사고 다발 예상 지역에 위급상황 대처 인프라 구축

3) 청각 장애인의 위험상황 인지 보조

2 데이터 전처리

프로젝트에 사용한 데이터는 AI Hub의 1) '위급상황 음성/음향 데이터'와 2) '소음 환경 음성인식 데이터'이다. 두 개의 데이터를 적절히 추출하여 14개의 위급상황, 5개의 비위급상황이 존재하도록 구성했으며, 총 66,514개(59.6GB)의 훈련 데이터 셋, 12,102개(10.9GB)의 테스트 데이터 셋을 구축할 수 있었다.

2.1 데이터 구조

데이터 구조는 세 부분('input features', 'labels', 'class')으로 구성되어 있다. 1) input features은 멜-스펙트로그램(Mel-spectrogram)으로, 음성의 시간, 주파수, 진폭에 대한 정보가 담겨있다. 2) labels은 음성의 텍스트 정보가 담겨있다. 3) class는 음성의 분류 번호이다. 1~14이면 위급상황, 15~19이면 비위급상황으로 분류된다.

2.2 텍스트 데이터 전처리

AI Hub의 위급상황 음성/음향 데이터는 텍스트 라벨링의 품질이 매우 낮다. 거의 대부분의 텍스트에 띄어쓰기와 맞춤법 오류가 존재하기 때문에 해당 데이터로는 음성 인식 모델을 학습시키기에 적합하지 않다고 판단했다. 이에 따라 49,000개의 위급상황 텍스트 데이터를 전수 조사하여 약 1,000 종류의 텍스트 분류가 있음을 확인하였고, 그 중 442개의 텍스트에 맞춤법 교정을 진행하였다.

```
# 맞춤법 교정 사전
correction_dict = {
    '도둑잡아': '도둑 잡아',
    '갇혔나봐': '갇혔나 봐',
    '짜증나게하지마': '짜증나게 하지 마',
    '내몸만지지마': '내 몸 만지지 마',
    '전기차단됐다': '전기 차단됐다',
    '전기 안되는데': '전기 안 되는데',
    '전기안되는데': '전기 안 되는데',
    '가스냄새가 나요': '가스 냄새가 나요',
    '유독가스마시지마': '유독가스 마시지 마'
}
```

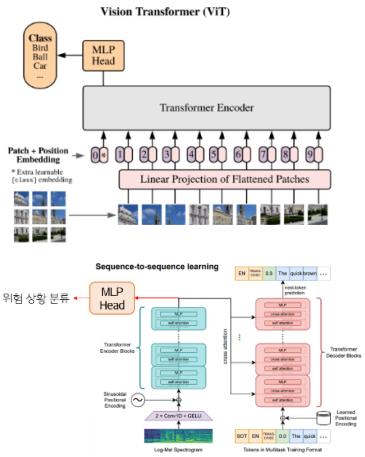
▲ [그림 1] 442개 텍스트의 맞춤법 교정 전처리

3 음성 인식 모델 구현

음성 인식 모델은 2023년 Open AI에서 발표한 Whisper 모델[1]을 이용하기로 하였다. 기존에는 2020년 Facebook에서 발표한 Wav2Vec2.0 모델[2]을 사용하기로 하였으나, 해당 모델의 한국어 음성 인식 정확도가 매우 낮은 편임을 확인하였다. 따라서 한국어 데이터를 대량으로 학습한 Whisper 모델이 더 적합하다고 판단했다.

3.1 Whisper 모델의 구조 변형: MLP Head

기존의 Whisper 모델은 STT(Speech-to-Text) 기능만을 수행한다. 그러나 본 프로젝트는 단순히 음성을 텍스트로 변환하는 것이 아니라, 음성과 비음성 데이터가 혼합된 음원으로부터 위급상황을 분류하는 것이다. 이러한 목적을 달성하기 위해 BERT[3]와 ViT(Vision Transformer)[4] 논문을 참고하여 Whisper의 Encoder 뒤에 MLP Head를 추가함으로써 Whisper 모델이 STT 작업뿐만 아니라 위급상황 분류까지 수행하도록 구현하였다. 이 부분이 본 프로젝트의 가장 핵심적인 구현 아이디어이다. 위 아이디어를 구체화시켜서 코드로 구현하면 다음과 같다.



[그림 2] (위) ViT 모델의 구조 (아래) Whisper 모델의 구조 변형 ▶

```python
whisper_model.config.forced_decoder_ids = processor.get_decoder_prompt_ids(language="korean", task="transcribe")
class CustomModel(nn.Module):
    def __init__(self, whisper_model=whisper_model):
        super().__init__()
        self.encoder = whisper_model.model.encoder
        self.decoder = whisper_model.model.decoder
        self.proj_out = whisper_model.proj_out

        # ViT(Vision Transformer)의 MLP Head에서 아이디어를 가져온다.
        self.MLPHead = nn.Sequential(
            nn.Linear(in_features=512, out_features=3072, bias=True),
            nn.Linear(in_features=3072, out_features=19, bias=True)
        )

    def forward(self, inputs):
        input_features = inputs['input_features'].to(device)
        audio_features = self.encoder(input_features)

        dec_input_ids = inputs['dec_input_ids'].to(device)
        dec_input_ids[dec_input_ids == -100] = tokenizer.pad_token_id # Critical Point!!! -100이 아닌 pad로 입력해야 한다.
        out = self.decoder(input_ids=dec_input_ids, encoder_hidden_states=audio_features[0])
        out = self.proj_out(out[0])

        logits = self.MLPHead(audio_features.last_hidden_state) # classification

        return (out, logits)
```

▲ [그림 3] MLP Head를 추가한 Whisper 모델의 변형 구현 코드

따라서 이 모델의 Encoder 부분에서는 19가지의 위급/비위급상황 분류를 수행하도록 하고, Decoder 부분에서는 음성의 텍스트를 생성함으로써 위급상황에 해당하는 텍스트가 포함되어 있는지 확인한다.

3.2 Whisper Model Fine-tuning

Whisper Model을 변형한 Custom Model([그림 3] 참고)은 Hugging Face의 Transformers 라이브러리에서 제공하고 있는 Pre-trained Whisper-base(parameters: 74M) 모델을 이용한다. 이는 음성 인식 성능은 다소 떨어지지만, 상당히 빠른 추론 시간(GPU 사용 시 약 0.5초)을 보인다.

[그림 3]을 보면 기존의 pretrained model의 Encoder와 Decoder를 분리해서 새롭게 만든 Custom Model의 Encoder와 Decoder로 사용함으로써 학습된 파라미터들을 그대로 활용한다는 것을 알 수 있다. 이는 Whisper Model을 Fine-tuning하는 것으로 해석할 수 있다. Whisper Model을 분리해서 사용하기에 내부의 데이터 토큰을 알맞게 조절해줄 필요가 있었으며, 약 300 lines 코드로 이를 구현하였다.

다음은 Whisper 모델의 변형인 Custom Model을 학습시킬 때 사용한 하이퍼파라미터이다. Text Loss와 Label Loss를 구할 때에는 모두 Cross Entropy Loss를 사용하며, Class Loss는 MLP Head의 output length 1,500개 모두 정답 class와 비교한다. (앞으로는 Custom Model을 Whisper Model이라고 언급한다.)

Hyper parameters	Batch size	Learning rate	Warmup steps	Num epochs	Eval steps
Whisper Model	8	1e-5(0.00001)	2000	3	1000

▲ [표 1] Whisper Model(Custom Model)의 Hyper parameters

3.3 CNN Model 구축

현재 사용하고자 하는 위급상황 데이터는 비음성 데이터와 음성 데이터의 혼합으로 구성되어 있다. 비음성 데이터에는 화재, 붕괴, 낙상 소리 등과 같은 정보가 있으며, 음성 데이터에는 '살려주세요!'와 같이 사람이 위급상황에서 발화하는 목소리 정보가 있다. 이에 따라 위급상황을 분류할 때에는 비음성 신호와 음성 신호를 동시에 활용할 필요가 있는데, Whisper 모델은 모델의 특성상 텍스트를 생성해내는 일에 최적화되어 있기에 '비음성 신호'보다는 '음성 신호'에 집중할 가능성이 높다. 따라서 '비음성 신호'를 주로 보고 판단할 수 있는 CNN Model을 도입하기로 하였다. (4.2절 참고)

기존 선행연구[5]를 참고하면 CNN Model은 간소화된 모델도 위급상황을 잘 분류해 내는 것으로 알려져 있다. 다음과 같이 간단하게 CNN Model을 구축하기로 하였다.

```
class MyNet(nn.Module):
    def __init__(self) -> None:
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 8, kernel_size=3, stride=(1, 2), padding=(1, 1)), # (?, 1, 80, 3000) -> (?, 8, 80, 1500)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)) # (?, 8, 80, 1500) -> (?, 8, 40, 750)
        self.conv2 = nn.Sequential(
            nn.Conv2d(8, 16, kernel_size=3, stride=(1, 2), padding=(1, 2)), # (?, 8, 40, 750) -> (?, 16, 40, 376)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)) # (?, 16, 40, 376) -> (?, 16, 20, 188)
        self.conv3 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=3, stride=(1, 2), padding=(1, 1)), # (?, 16, 20, 188) -> (?, 32, 20, 94)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)) # (?, 32, 20, 94) -> (?, 32, 10, 47)
        self.conv4 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, stride=(1, 2), padding=(1, 2)), # (?, 32, 10, 47) -> (?, 64, 10, 24)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)) # (?, 64, 10, 24) -> (?, 64, 5, 12)
        self.conv5 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=(1, 2), padding=(1, 1)), # (?, 64, 5, 12) -> (?, 128, 5, 6)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(1, 2), stride=(1, 2))) # (?, 128, 5, 6) -> (?, 128, 5, 3)

        self.fc1 = nn.Linear(128*5*3, 128, bias=True) # (?, 256, 5, 3) -> (?, 128)
        nn.init.kaiming_normal_(self.fc1.weight, nonlinearity='relu') # He Initialization
        self.linear1 = nn.Sequential(
            self.fc1,
            nn.ReLU(),
            nn.Dropout(p=0.2)) # 20% dropout

        self.fc2 = nn.Linear(128, 64, bias=True) # (?, 128) -> (?, 64)
        nn.init.kaiming_normal_(self.fc2.weight, nonlinearity='relu')
        self.linear2 = nn.Sequential(
            self.fc2,
            nn.ReLU(),
            nn.Dropout(p=0.2)) # 20% dropout

        self.fc = nn.Linear(64, 19, bias=True) # (?, 64), (?, 19)
```

▲ [그림 4] Convolution Layer 5층과 Fully Connected Layer 3층으로 구성된 CNN Model

3.4 앙상블(Ensemble) 모델 구축

1) Whisper Encoder에서는 음성 신호와 비음성 신호를 종합적으로, 2) Whisper Decoder에서는 음성 신호(텍스트 정보) 위주로, 3) CNN Model은 비음성 신호 위주로 위급상황을 분류하는 것으로 예상된다. 세 모델의 결과를 종합하여 Majority Voting(앙상블 기법 중 하나, 가장 많은 표를 얻은 분류를 택하는 기법)으로 위급상황을 분류하고자 한다.

4 모델의 평가 방법

4.1 정확도(Accuracy)와 CER(Character Error Rate)

Whisper Encoder와 CNN Model의 성능은 1) 19가지의 위급상황 분류에 대한 정확도와 2) 위급/비위급상황 이진 분류의 정확도로 평가한다. 특히 19가지의 위급상황 분류는 혼동행렬(Confusion Matrix)을 통해 시각화를 한다. 그리고 Whisper Decoder의 성능은 1) 위급/비위급상황 이진 분류의 정확도와 2) CER(Character Error Rate)로 평가한다. CER은 예측 텍스트와 정답 텍스트 사이의 문자 오류율을 의미하는 지표로, 다음 식을 따른다.

$$CER = \frac{S+D+I}{N}$$ (S: 대체 오류, D: 삭제 오류, I: 삽입 오류, N: 정답 문자 개수)

4.2 LRP(Layer-wise Relevance Propagation)

XAI(Explainable AI)를 이용하면 모델이 예측할 때 input의 어느 부분을 얼마나 참고했는지 확인할 수 있어서 모델의 설명력을 높여줄 수 있다는 장점이 있다. 그래서 본 프로젝트의 CNN Model에 XAI를 적용하기 위해 LRP(Layer-wise Relevance Propagation) 기법을 도입하였다. LRP는 딥러닝 모델의 결과로부

터 각 은닉층을 역으로 추적해가며 입력 이미지의 각 원소의 기여도를 계산한다. LRP를 통해 1) CNN Model이 실제로 비음성 신호를 주로 참고했는지 여부와 2) 각 분류마다 어떤 주파수 영역을 참고했는지 확인해볼 수 있다. 해당 구현은 [6]의 github를 참고하였다.

Whisper에도 XAI를 적용해보기 위해 XAI for Transformers[7] 논문을 읽어보았으나, 현실적으로 시간적 한계로 구현에 어려움이 있을 것 같다고 판단하여 적용하지는 않았다.

5 모델 실험 결과

5.1 Whisper Model

Text Loss는 Whisper 모델의 Decoder에서 텍스트를 생성할 때 발생하는 손실(loss)이고 Label Loss는 Whisper 모델의 Encoder에서 위급상황을 분류해낼 때 발생하는 손실이다. 하나의 Whisper 모델이 두 작업을 동시에 수행해야 하기 때문에 두 가지 Loss가 동시에 발생한다. 그래서 두 Loss의 비율을 어떻게 설정하는지에 따라 학습 결과가 달라질 수 있다. 상대적으로 Label Loss가 Text Loss보다 크기 때문에, Text Loss : Label Loss 비율을 각각 1:1(Model 1), 10:1(Model 2), 100:1(Model 3)로 두고 실험을 진행하였다. 그 결과는 다음과 같다.

Model (Text Loss : Label Loss)	Model 1 (1:1)	Model 2 (10:1)	Model 3 (100:1)
Encoder Accuracy (분류 정확도)	76.05%	**98.40%**	99.36%
Decoder CER (Text 문자 오류율)	17.22	**19.21**	70.41

▲ [표 2] Text Loss와 Label Loss 비율에 따른 Whisper Model 실험 결과 (검증 데이터)

Model 1은 CER이 높지만 Accuracy가 매우 낮고, Model 3는 Accuracy가 높지만 CER이 매우 나쁘다. 따라서 Model 2로 선정하기로 하였다.

5.2 CNN Model

CNN Model도 각각 은닉층 수, 학습률(learning rate), Scheduler 등을 조절하면서 실험해보았다. Model 5~7의 정보만 요약하면 다음과 같다. (Model 1~4는 2차 발표자료 16페이지 참고)

Model	Convolution layers	Learning rate	Scheduler	Accuracy
Model 5	3	1e-4	X	92.04%
Model 6	3	1e-3	ReduceLROnPlateau	94.89%
Model 7	5	1e-3	ReduceLROnPlateau	**96.24%**

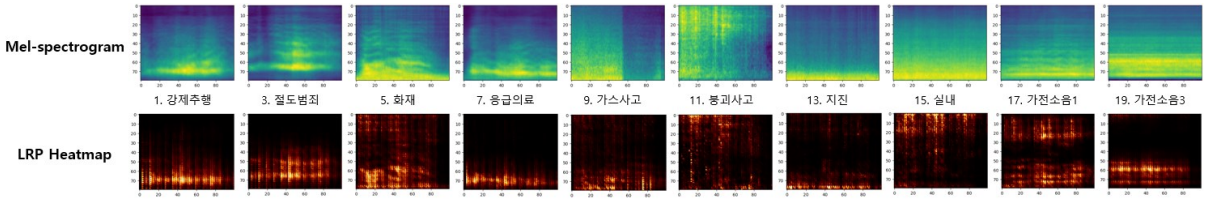
▲ [표 3] CNN Model 실험 결과 (검증 데이터)

5.3 LRP를 통한 분석

CNN Model에 대해 LRP를 이용하여 시각화하면 CNN Model이 위급상황을 분류할 때 대체적으로 비음성 신호를 활용했음을 알 수 있다. 대표적으로 다음과 같은 예에서는 앞부분에서 벽이 붕괴하는 소리가 들리고 뒷부분에서 사람의 구조 요청 소리가 들리지만, CNN Model은 앞부분만을 참고하였다.

▲ [그림 5] 붕괴사고 위급상황의 LRP 시각화 예

또한 LRP Heatmap(입력 데이터의 기여도를 시각화한 그림)을 통해 각 상황에서 어느 주파수 범위를 주로 살펴봤는지 확인해보았다. 다음은 각 상황의 100장에 해당하는 Mel-spectrogram과 LRP Heatmap의 평균들을 보여주고 있다. 가로축은 1초에서 2초 사이의 시각, 세로축은 로그 스케일의 주파수이다. 위부분일수록 고주파에 해당한다고 볼 수 있다. 이를 통해 해당 모델이 Mel-spectrogram의 특징을 잘 잡아내고 있으며, 특히 '15. 실내'의 경우 Mel-spectrogram의 역관계를 주로 살펴보았음을 알 수 있다.



▲ [그림 6] 각 상황에서의 Mel-spectrogram의 평균과 LRP Heatmap의 평균

5.4 Ensemble Model

각 모델의 성능과 Ensemble Model의 성능은 다음과 같다. 위급상황을 99.93%로 이진 분류할 수 있다.

Model	CNN Model	Whisper Enc.	Whisper Dec.	Ensemble
19가지 위급상황 분류 Accuracy	94.61%	**97.40%**	-	-
위급/비위급상황 이진분류 Accuracy	99.60%	99.85%	96.21%	**99.93%**

6 프로젝트 한계점 및 보완 과제

본 프로젝트의 한계점에는 1) 데이터의 품질 한계, 2) 시간적 한계, 3) 컴퓨팅 자원의 한계 등이 있다. 데이터는 인위적인 위험상황으로 구성되어 있으므로 실제 상황에서 낮은 품질을 보일 수 있다. 또한 시간과 컴퓨팅 자원의 한계로 많은 실험을 해보는 데에 어려움이 있었다.

대표적인 보완 과제는 '시각별로 위급상황 판단하는 모델'을 구현하는 것이다. 현재 MLP Head는 위급상황을 시각별로 찾을 수 있도록 구현되어 있으나, 컴퓨팅 자원의 한계로 위급상황 구간이 파악된 데이터를 구축하지 못하였다. 만약 데이터를 적절히 구축하고 Whisper Encoder의 구조를 적절히 변경하면 시각별로 위급상황을 판단하도록 개선이 가능할 것이다.

7 참고문헌

[1] Radford, Alec, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. "Robust speech recognition via large-scale weak supervision." In International Conference on Machine Learning, pp. 28492-28518. PMLR, 2023.

[2] Baevski, Alexei, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. "wav2vec 2.0: A framework for self-supervised learning of speech representations." Advances in neural information processing systems 33 (2020): 12449-12460.

[3] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[4] Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).

[5] 김현돈(Hyun-Don Kim). "청각장애인을 위한 인공지능 헤드폰 설계." 산업기술연구논문지 (JITR) 27.4 (2022): 21-26.

[6] " PyTorchRelevancePropagation," GitHub, https://github.com/kaifishr/PyTorchRelevancePropagation

[7] Ali, Ameen, Thomas Schnake, Oliver Eberle, Grégoire Montavon, Klaus-Robert Müller, and Lior Wolf. "XAI for transformers: Better explanations through conservative propagation." In International Conference on Machine Learning, pp. 435-451. PMLR, 2022.