**Pusan National University Computer Science and Engineering Technical Report 2022-09**

**기존 사투리 앱 증진**



김도영

김윤종

황승찬

지도교수 이도훈 교수님

목 차

[1. 서론 1](#_Toc115427785)

[1.1. 연구 배경 1](#_Toc115427786)

[1.2. 기존 문제점 1](#_Toc115427787)

[1.3. 연구 목표 2](#_Toc115427788)

[2. 연구 배경 3](#_Toc115427789)

[2.1. 연구 개발 환경 3](#_Toc115427790)

[2.2. 데이터 수집 4](#_Toc115427791)

[2.3. ATTENTION 5](#_Toc115427792)

[2.4. 사전학습 모델(Pretrained Model) 5](#_Toc115427793)

[2.5. STT(Speech-to-Text) 배경 지식, 문제점 5](#_Toc115427794)

[2.6. 웹서버 배포를 위한 React와 Nginx 사용 7](#_Toc115427795)

[2.7. Docker를 사용한 배포 관리 8](#_Toc115427796)

[2.8. React-Native로 iOS/Android 앱 동시 개발 8](#_Toc115427797)

[2.9. Expo을 활용한 앱 테스트 및 배포 8](#_Toc115427798)

[3. 연구 내용 9](#_Toc115427799)

[3.1. 경상도, 표준어 데이터셋 구축 및 번역모델 학습 9](#_Toc115427800)

[3.1.1. 모델 실험(transformer, BART) 9](#_Toc115427801)

[3.1.2. 데이터셋 실험 11](#_Toc115427802)

[3.2. STT , 서버 구축 12](#_Toc115427803)

[3.2.1. Google STT Speech Context 단어 추가 13](#_Toc115427804)

[3.2.2. No DataBase 단어 추가 13](#_Toc115427805)

[3.2.3. 중앙 서버 구축 16](#_Toc115427806)

[3.2.4. 번역 프로그램 모듈 연결 17](#_Toc115427807)

[3.2.5. 앱/웹 통신 18](#_Toc115427808)

[3.2.6. Docker로 React + Nginx 프로젝트 관리하기 18](#_Toc115427809)

[3.2.7. React로 경상도 사투리 번역사이트 개발 22](#_Toc115427810)

[3.2.8. React-Native로 iOS/Android 앱 개발 및 배포 23](#_Toc115427811)

[4. 결론 및 향후 연구 방향 26](#_Toc115427812)

[4.1. 결론 및 향후 연구 방향 26](#_Toc115427813)

[4.1.1. STT, 서버 파트 26](#_Toc115427814)

[4.1.2. 웹/앱 27](#_Toc115427815)

[4.1.3. 방언/표준어 번역 27](#_Toc115427816)

[4.2. 구성원별 역할 27](#_Toc115427817)

[4.3. 개발 일정 28](#_Toc115427818)

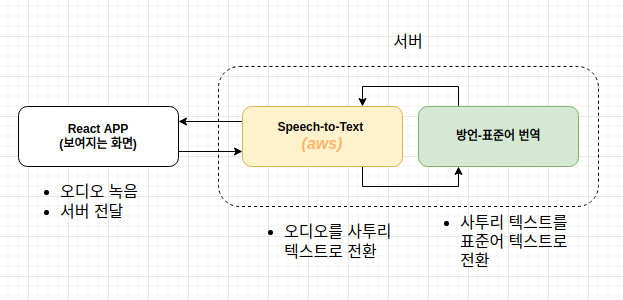
[5. 참고 문헌 28](#_Toc115427819)

# 서론

## 연구 배경

동일한 언어를 사용하는 나라에서도 지역마다 다른 언어 특성을 가지고 있다. 한국도 작은 나라이지만 지역마다 다른 특성의 사투리가 존재한다. 지역마다 같은 물건을 보고 다르게 이야기하기도 하고 같은 의미의 문장을 이야기하지만 한 음절도 같지 않은 경우가 있다. 아래 그림처럼 "경상도 사람은 말하지 못하는 문장", " 서울 사람이 못하는 경상도 문장" 과 같은 주제가 현재 SNS에서 큰 공감을 받고 있으며 이것을 통해 지금까지도 지역들 간의 언어의 벽이 크게 존재하는 것을 알 수 있다.

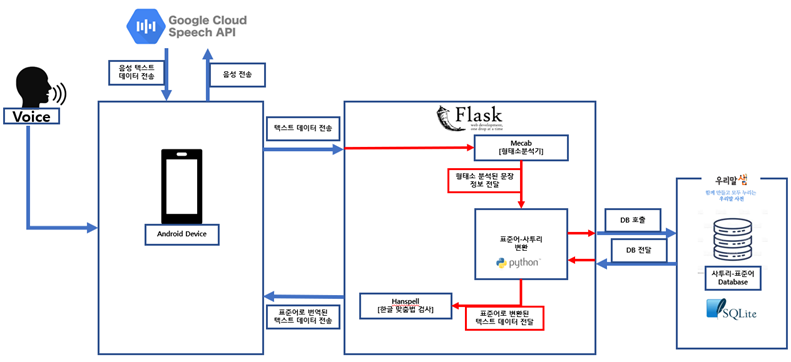
현재는 자연어처리분야에서 다양한 attention기법들이 발전해 있고 이것을 통해 구문론적, 형태론적, 의미론적으로 좋은 성과를 낼 수 있었다.  우리는 이런 도전에 영감을 받아 딥러닝을 적용시켜 경상도-표준어의 번역을 진행하여 우리들의 언어의 벽을 허물어보려고 한다.



## 기존 문제점

1.2.1 번역 문제점

이번 과제에 목표는 작년의 모델에 대한 성능을 높이는 것이었다. 작년 모델의 구조는 다음과 같았다.



출처 : 작년 보고서

작년 모델에서는 번역을 word level로 tokenize하여 DB에서 사투리 단어, 표준어 단어를 mapping하는 방식으로 진행하였다. 해당 방식으로 사용하게 된다면 부추 - 정구지 같은 단어 level의 번역은 가능하지만, 한 음절 단위로도 의미가 바뀌게 되는 사투리에 있어서 좋은 성능을 기대하기 어렵다.

1.2.2 UI 문제점



작년 UI, 출처 : 작년 보고서

작년 보고서의 안드로이드 어플은, 기기에 따라 화면의 UI가 온전히 유지되기 힘들었으며, 사용자가 이해하기 힘든 UI/UX로 이루어져있다. 사용자가 쉽게 앱과 웹을 사용할 수 있도록, UI/UX를 개선하였다.

## 연구 목표

본 졸업 과제는 웹 페이지와 안드로이드용 모바일 애플리케이션 개발을 통해 누구나 쉽게 경상도 사투리 번역 서비스를 이용할 수 있게 하고, 보다 쉽게 사투리를 접할 수 있게 함으로써 지역 간의 소통을 원활하게 하는 것을 목표로 한다.

이를 위해 App에서는 Google STT(Speech-To-Text) API를 통한 음성 인식 서비스를 제공하고, attention 기법들을 활용하여 딥러닝 모델을 바탕으로 번역 모델을 생성하고 파이썬 웹 프레임워크인 Flask로 구축된 간단한 Web Server를 통해 음성으로 입력된 경상도 사투리 문장을 표준어 문장으로 번역해주는 프로세스를 마련한다.

# 연구 배경

## 연구 개발 환경

(모바일) 스마트폰

|  |  |  |
| --- | --- | --- |
| Device | OS | feature |
| iphone 12 mini | ios 13 | React Expo |
| Galaxy 10 5G | Android 11 | React Expo |

(PC) SAMSUNG LAPTOP

|  |  |  |  |
| --- | --- | --- | --- |
| CPU | Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz | | |
| RAM | 8GB | OS | Windows 10 Home 64bit |

## 데이터 수집

데이터 수집은 AI-HUB에서 사용되는 데이터와 영화 대사의 text을 사용하였다. 이미 충분히 AI-HUB에 20만개 이상의 경상도-표준어 데이터 쌍이 있었지만 일상어를 그대로 가지고 온 데이터여서 방언 column과 표준어 column의 차이가 크지 않았다. 또한 수정을 해야 하는 부분도 제대로 수정이 되어있지 않은 상태였다. 아래 그림은 ai hub data를 random으로 21개 sampling한 것이다.



AI HUB 텍스트, 방언 데이터 셋 (좌) 방언 (우) 표준어

그래서 우리는 제대로 된 방언 번역을 제공하기 위해 표준어와 차이가 큰 방언을 많이 사용하는 데이터셋이 필요했다. 그래서 범죄와의 전쟁, 바람의 text를 다운받아 수작업으로 방언-표준어의 번역 데이터셋을 만들어 주었다.



범죄와의 전쟁 텍스트 파일

이렇게 우리가 사용할 수 있는 데이터는 총 ai hub(200,000), 영화 대사(3,000)개가 준비되어졌다.

## ATTENTION

2015년에 나온 바다나우 어텐션기법이 NLP에서 큰 성능을 보여준 후 정말 많은 어텐션 기법들이 등장하기 시작하였고 이 중 self-attention을 기반으로 trasnformer라는 모델이 만들어지게 된다. 현재 대부분의 NLP에서 사용되는 모델은 해당 transformer의 기반의 모델로 만들어지게 되었으며, 이는 번역에 있어서 하나의 단어만 보는 것이 아닌 단어 하나에 따른 모든 문장에서의 단어 가중치를 학습하게 된다. 즉, 번역에 있어 문장 전체를 보고 번역을 진행하게 되어 같은 모습의 단어라도 해당 의미를 해석하여 적절한 단어로 번역을 진행하게 해준다.

## 사전학습 모델(Pretrained Model)

앞서 언급된 transformer 이후 NLP에서 연산처리 속도가 매우 빨라지게 된다. 또한 하드웨어들의 성능향상으로 대량학습이 가능하게 되어 Vision에서 큰 효과를 입증하였던 사전학습 방식이 자연어처리에서도 사용할 수 있게 된다. 대표적인 모델로는 GPT, BERT들이 있으나 이들은 Encoder 또는 Decoder만 있는 모델로 번역에 적합한 모델이 아니며 번역에 적합한 사전학습 모델로는 최근에 나온 BART와 T5들이 있다.

## STT(Speech-to-Text) 배경 지식, 문제점

음성 인식 분야를 간단하게 설명하자면, 많은 언어들에 있어서 개별 적용이 필요한 사실상 “미개발 분야”라는 것이다. 음성 인식 기술을 가지고 있는 회사는 애플, 구글, 네이버, 카카오 등 자체 개발 인력이 풍부한 대기업이 대부분이며, 표준어에 국한된다. 사람의 발음을 사람이 인지하는 가장 작은 단위로 나눈다면 ‘음소’라는 기준이 있을 것이고, 같은 단어나 문장에서라도 음소는 다르게 표시되고, 같은 음소이더라도 파형은 수 만 가지를 가질 수 있다. 완벽하게 음소만으로 나타낼 수 있는 언어는 없으며, 언어-음소-파형의 대입 방식이 아닌, 언어-파형의 직접 대입으로 개발되는 것이 음성 인식 기술의 현재 진행형이다. 이것이 많은 언어에 있어서 개별 적용이 필요하다고 한 이유이다. 더 나아가 음성을 국제 음성 기호(음소)로 변환하는 API는 현재 없고, 음성인식 API의 원리 상 음소를 정확히 찾아내는 것이 아닌 비슷한 파형의 음성 배열에서 등장할 확률이 높은 단어를 통해 단어 배열을 결정하는 것이기 때문에 정확한 음소를 찾는 것이 크게 의미가 없다. 위와 같은 이유로 이번 연구에서는 가장 활용성이 open API를 사용해 구현하는 방법을 선택할 수 밖에 없었다.

구글 STT API, 네이버 CLOVA CRS, 카카오 Speech API 비교 분석

|  |  |  |  |
| --- | --- | --- | --- |
|  | 구글  STT(Speech-to-Text) | 네이버  CRS | 카카오톡  Speech API |
| 가격 | 60분까지 무료  60분 이상 진행시  15초당 0.006USD | 유료(15초당 4원) | 무료(이전엔 유료였음) |
| 제공형태 | Android SDK  ios SDK  REST API | Android SDK  ios SDK | Android SDK  ios SDK  REST API |
| 사용자 지정 단어 추가 가능여부 | O | X | X |

**[표 1] 음성인식 API 비교표**

텍스트이(가) 표시된 사진

자동 생성된 설명

**[그림 1] 카카오 Speech API 사이트 설명**

이번 연구에서 음성 인식 파트의 주 목적은 사용자의 음성을 사투리 텍스트로 바꾸는 것이다. 본인은 2022년 기준 이용 가능한 open API를 사용하거나, 음소를 추출 가능한 프로그램을 처음부터 만드는 두 가지 측면에서 접근하였다.

카카오와 네이버는 한국 표준어에 특화되어 있지만, 카카오 Speech API 설명 예시처럼 사투리는 지원하지 않는다. 특히 네이버는 Rest API를 지원하지 않아 웹 개발이 어렵다. 반면, 구글의 Speech API는 사용자 지정 단어 (Speech Context)라는 것을 지원하여 표준어에 있는 단어(문장) 이외에도 커스텀이 가능하며, 사투리 문장을 만들어내야 하는 목표 특성 상 구글 API를 선택하게 되었다.

구글 STT(Speech-to-Text) API speech Context

텍스트이(가) 표시된 사진

자동 생성된 설명[1]

음성 인식 프로그램에서 임의의 음성을 문장으로 만드는 원리는 위의 논문 내용의 이론을 따른다.한 음원의 파형 배열은 여러 음소로 나타낼 수 있고, 한 문장 배열 또한 여러 음원으로 나타날 수 있다. 따라서 한 음성 데이터를 한 문장으로 특정하기 위해서는 음소 뿐만 아니라 다른 요소를 더 필요로 한다. 특정 단어가 함께 나타날 확률 p(w)이다. 구글 STT API의 사용자 지정 단어의 원리는 특정 단어로 해석될 확률을 높이는 것으로 단어를 추가한 것과 같은 효과를 내는 것이다. 이것이 구글 STT API를 선택한 가장 큰 이유이다.

그러나 구글 API 또한 표준어 기반이다. 때문에 확률을 강화한다고 하더라도, 사투리 단어가 구글의 DB에 존재하지 않으면, 효과가 없었다. 구글 API는 자동 학습을 하기 때문에 같은 음성과 단어를 꾸준히 업로드하면 자체 추가가 되는 모습도 보였으나, 개발자 임의로 언제든지 추가할 수 있는 시스템이 아니라 학습될 때까지 기다릴 수도 없는 상황이다. 이것이 가장 큰 문제점으로, 이것을 인지한 시점에서 음성인식 모듈을 자체 개발할 생각도 했지만, 위에서 설명한 이유로 불가피했다.

## 웹서버 배포를 위한 React와 Nginx 사용

nginx는 웹 서버 소프트웨어로, 가벼움과 높은 성능을 목표로 한다. Nginx는 요청에 응답하기 위해 비동기 이벤트 기반 구조를 가진다. 이러한 구조는 서버에 많은 부하가 생길 경우의 성능을 예측하기 쉽게 해준다. nginx의 핵심은 **가벼움과 높은 성능**이다. Apache에 비하여 스레드를 많이 사용하지 않기 때문에 CPU 소모도가 상대적으로 낮다. 적은 수의 스레드로 효율적인 일처리가 가능하기에 많은 접속자들이 와도 대응을 할 수 있다.

## Docker를 사용한 배포 관리

컨테이너는 애플리케이션을 환경에 상관없이 실행하는 기술이다. 예를 들어, 우분투 OS를 설치하려 해도 도커는 명령어로 실행하면 어느 환경이든 상관 없이 다음 명령어를 사용하여 실행할 수 있다. 서버 제작과정에 견고함과 유연성, 편리함이 좋고 다른 사람이 만든 서버를 소프트웨어 사용하듯이 바로 사용할 수 있는 장점이 있다. 또한 여러 대에 배포할 수 있는 확장성을 가진다.

## React-Native로 iOS/Android 앱 동시 개발

리액트 네이티브는 페이스북에서 만든 오픈소스 모바일 애플리케이션 프레임워크이다. JavaScript로 개발이 가능하며 하나의 프로그래밍 언어로 IOS와 안드로이드 모바일 앱을 동시에 개발할 수 있는 크로스 플랫폼이다.

페이스북의 리액트에서 파생되었기 때문에 그 방식을 그대로 사용했다. React는 Component 기반으로 기능들을 분리해 재사용할 수 있는 구조로 되어 있다. 또한 기존에 리액트로 개발된 사이트가 있다면 그 사이트의 Component 코드를 그대로 사용할 수 있기 때문에 재사용 가능하면 생산성은 더욱 올라간다.

또한 하이브리드 앱의 특징으로 두 개의 OS인 Android와 IOS 모두 한 번에 개발할 수 있기 때문에 개발을 효율적으로 할 수 있다.

# 연구 내용

## 경상도, 표준어 데이터셋 구축 및 번역모델 학습

번역파트에서는 딥러닝을 활용한 번역을 진행하였다. 번역에서 좋은 성능을 보여줬던 trasnformer, bart 등의 다양한 모델을 학습시켜 실험하여 가장 좋은 모델을 사용하였으며, 데이터셋 역시 영화 text 데이터를 추가하여, 기존 aihub 데이터셋 20만개와 영화 text 데이터 5000개를 클래스 불균형을 완화하는 oversampling 기법을 통해 비율을 맞춰 사용하였다.

또한 해당 실험에서는 방언이 잘 번역되는지에 대한 평가를 진행하는데, 사투리의 특성상 우리가 원하는 답과 틀려도 옳은 문장이 될 수 있기 때문에 (ex. 니 뭐하노 -> 너 뭐해, 너 뭐하니) 평가를 사람이 직접하는 정성평가로 진행한다.

데이터셋 실험에서는 공평한 실험을 위해 train에서 학습하지 않은 영화의 대사로 실험을 진행하였다.

### 모델 실험(transformer, BART)

현존하는 NLP에서 가장 인기 많은 두 모델을 꼽으라고 하면 당연 GPT와 BERT가 언급될 것이다. 하지만 이 두 모델은 encoder 또는 decoder만 존재하기 때문에 번역에 적합한 모델은 아니며, 번역에 적합한 모델이 되기 위해서는 인코더, 디코더 두 개 모두가 필요했다. BART 또한 GPT와 BERT 처럼 사적학습 모델로 GPT의 사전학습 방식의 디코더, BERT 변형 모델들의 사전학습 방식의 인코더가 합쳐진 모델이다. 그래서 번역이 가능한 모델이며 해당 과제에서는 한국어를 사전학습한 KoBART모델을 사용하였다.

transformer는 self attention 기법을 사용하는 모델로 등장했을 당시 이전의 모델들과 압도적인 성능차이를 보여 준 모델이다. 이를 통해 예측한 문장은 다음과 같다.





틀린 부분이 충분히 있지만, BASE MODEL로 채택하기에 충분한 가능성을 보여주었다.

위의 transformer 모델의 사용 후에 사전학습 모델 기반의 번역을 하면 더 좋은 성능이 나올 것이라 예상하여 번역에서 사용될 수 있는 사전 학습 모델인 KoBART를 사용하였다. 정확히는 번역이 아닌 Text Style Transfer의 방식으로 사투리를 표준어로 바꾸려는 시도를 해보았으며 결과는 다음과 같았다.



"니 뭐라카노" 가 "니니 뭐라카노" 로 번역된 것을 알 수 있었는데, 이는 사전학습 모델의 단점인 OOV(Out Of Vocabulary) 때문에 나타난 것으로 보였다. 기본적으로 컴퓨터가 인식시키기 위해서는 우리는 언어를 그대로 넣어줄 수 없어 언어를 숫자로 바꾸는 작업을 진행하여야 하는데, KoBART는 표준어를 토대로 사전학습이 진행되었기 때문에 사투리 단어를 제대로 인식을 하지 못하였다. 사투리를 인식하지 못하는 것은 학습시에도 제대로 하지 못했을 것이며 이는 큰 성능 하락의 요인이 된다.

BART에 대한 방식이 큰 성과를 내지 못했고 경상도 사투리를 대량으로 사전학습한 모델이 있지 않기에 transformer를 basemodel로 잡고 성능을 늘리는 방법을 생각했다.

### 데이터셋 실험

처음 모델의 학습은 ai hub데이터로만 진행되었다. 하지만 ai hub 데이터셋 만을 이용하여 큰 성과를 얻어내기 어려웠다. 좋은 데이터셋을 구하기 위해 직접 데이터셋을 구축하였으며, 표준어와 같은 의미이지만 많은 변화가 있는 문장의 쌍일수록 경상도 사투리를 잘 학습시킬 것으로 예상하였기 때문에 경상도 사투리가 두드러지는 영화들을 골라 데이터셋을 구축하였다.

영화는 바람, 범죄와의 전쟁의 대사 text본을 구하여 사투리 데이터를 얻어 왔으며 이에 대응되는 표준어 데이터는 수작업으로 진행하였다. 세개의 영화에서 2500개의 데이터를 얻어 냈으며 이를 oversampling 하여 어느 정도 개수를 맞춰 실험을 진행하였다.

처음 ai hub 데이터 20만개와 영화 데이터 5천개를 단순하게만 합쳤을 때는 큰 변화를 느끼지 못하여 ai hub 데이터 중 표준어와 방언 데이터가 달라진, 즉 동일한 의미이지만 다르게 쓰이는 문장쌍 2만개와 표준어와 방언 데이터가 같은 데이터 3만개를 random으로 sampling하여 총 5만개의 ai hub 데이터를 사용하였다.





최종적으로 우리가 만든 번역 모델이 현재 시중에 사용되는 다른 모델과 얼마나 성능차이가 나는지 실험하기 위해 NLP 스타트업 기업인 Tunib의 모델과 비교하는 실험을 진행하였다.





(좌) 튜닙 모델, (우) 우리 모델

tunib에서의 모델이 조금 더 좋은 성능을 가지고 있었지만 충분히 뒤쳐지지 않는 성능을 볼 수 있었다.

## STT , 서버 구축

STT(Speech-to-Text)파트에서는 사용자 입력 음성의 사투리 텍스트 변환을 목표로 하여 개발했다. 또한 APP/WEB 과 번역 프로그램의 중간 매개체의 역할을 하는 파트 특성 상 서버 구축 역할도 병행하게 되었다. 개발 언어는 번역 프로그램과 동일한 Python을 사용했으며, 서버 구축에 있어 flask 라이브러리, html 소스 코드를 활용하였다.

### Google STT Speech Context 단어 추가

가장 기본적으로 사투리 음성을 인식시키기 위한 방법으로, 구글 DB에 사투리 단어가 존재하는 경우도 존재했기 때문에, Speech Context 기능을 통해 단어의 인식 확률을 강화하는 것을 먼저 해볼 수 있었다.

f = open("Saturi\_file.csv", 'r', encoding='utf-8-sig')

Saturi\_word = csv.reader(f)

f.close

#사투리 단어 우선도 추가

speech\_contexts = []

scw\_dict = {}

for row in Saturi\_word:

phrases = row[0]

boost = (int)(row[1])

if not boost in scw\_dict:

scw\_dict[boost] = list()

scw\_dict[boost].append(phrases)

for key in scw\_dict:

speech\_contexts\_element = {"phrases": scw\_dict[key], "boost": key}

speech\_contexts.append(speech\_contexts\_element)

(생략)

config = {

"speech\_contexts": speech\_contexts,

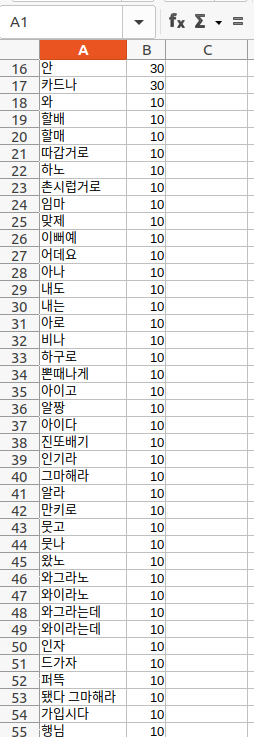
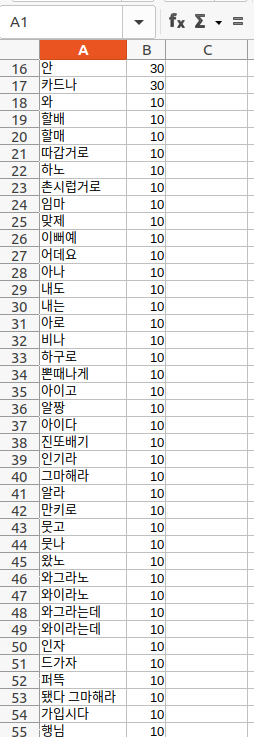
"sample\_rate\_hertz": 44100,

"audio\_channel\_count" : 2,

"language\_code": 'ko-KR',

"encoding": speech.RecognitionConfig.AudioEncoding.LINEAR16,

}



(A열에는 단어가, B열에는 강화(boost)정도가 들어간다.)

CSV 형식으로 단어와 확률 강화(boost) 정도를 지정했고, 이것을 API 입력 인자에 포함하여 요청하게 되면, 구글 DB에 있는 경우에 한해서 단순한 문장에서는 효과를 보였다.다만, DB에 없는 경우나 발화 문장이 너무 길어지는 경우 STT API가 문맥을 분석하지 못해 단어를 인식하지 못하거나 확률을 강화하는 효과가 없어지기도 했다.

### No DataBase 단어 추가

구글 DB에 없어 잘못된 단어로 인식하는 문장의 경우에 대해서 원래 의도한 문장으로 2차 수정하는 방식으로 앞 선 문제점을 해결하고자 했다. [2]

#구글 데이터 베이스에 없는 단어 처리

NoDBWord = []

for row in No\_DB\_word\_convert:

NoDBWord.append(row)

for key in scw\_dict:

speech\_contexts\_element = {"phrases": scw\_dict[key], "boost": key}

speech\_contexts.append(speech\_contexts\_element)

(생략)

#서버 웹 반환 코드 내용 일부

converted\_text = original\_text

converted\_text = converted\_text.replace(' ', ' ')

print(converted\_text)

for words in NoDBWord:

converted\_text = converted\_text.replace(words[0], words[1])

print("Converted text:" + converted\_text)

translated\_text = translate\_module(converted\_text)



(A열에는 잘못 출력된 문장 케이스가, B열에는 원래 의도했던 문장이 들어간다.)

연구 후반에 추가한 내용으로, 1 대 1 수정을 하는 방식이라, 지속적인 추가가 필요하고, 별도의 알고리즘을 구축한 것이 아니라 정확도도 낮을 것이다. 현재는 의도하여 추가한 문장에 대해서만 인식하는 것을 볼 수 있다.

|  |
| --- |
| “text: 아지매 정구지는 어데서 팝니까“ |
| “text: 그라이께 내가 가마이 좀 있어라 안 카더나 “ |
| “text: 니 와 우리 할배 눈에 따갑거로 하노“ |
| “text: 촌시럽거로 뭔 두부고 내가 교도소 갔다 왔나 인마“ |
| “text:내가 세상에서 제일 이쁜거 맞제 “ |
| “text: 어데요 백설 공주가 세상에서 제일 이뻐예“ |
| “text: 무슨 마음으로 확인하는지 알아“ |
| “text: 니를 믿어라 내도 믿고 우리한테는 그거밖에 없다“ |
| “text: 아직도 내가 아로 비나“ |
| “text: 그전에 벌초라도 함 하구로“ |
| “text: 내는 니처럼 가오 잡고 뽄대나게 살고 싶은 놈도 아이고“ |
| “text: 니 같은 놈이 알짤거리가 까불고 넘볼 사람이 아이다“ |
| “text: 이게 진뽀배기 부산 국밥 인기라“ |
| “text: 됐다 그마해라“ |
| “text: 알라 오줌 만키로“ |
| “text: 니 와 그라는데 불만 있나“ |
| “text: 밥 뭇나 뭐 한다고 안 뭇고 왔노“ |
| “text: 인자 좀 드가자 퍼뜩“ |

텍스트이(가) 표시된 사진

자동 생성된 설명다음은 작년 논문에서 시험한 문장에 대한 음성 녹음 결과 텍스트다

‘눈을’, ‘눈에’ 와 같이 발음이 비슷한 경우 문맥을 통해 파악해야 하기 때문에 수정이 어렵고 구글 STT API에서 가 많으며, ‘알아(표준어), ‘아나(사투리)’와 같이 표준어에 복수의 의미가 있는 경우에는 표준어로 인식되어도 차후 수정이 어렵다.

### 중앙 서버 구축

@app.route('/')

def home\_page():

return render\_template('upload.html')

@app.route('/text\_only\_test/<string:input\_text>')

def text\_only\_test(input\_text):

#input\_text = request.args.get('input\_text')

translated\_text = main\_module(input\_text)

return jsonify(text=input\_text, text2=translated\_text)

@app.route('/file\_upload', methods= ['GET', 'POST'])

def upload\_audio():

global port\_adress

if request.method == 'POST':

#파일 저장

f = request.files['file']

#저장할 경로 + 파일명

file\_root = "./uploads/"+secure\_filename(f.filename)

f.save(file\_root)

#저장된 오디오 파일 stt api 적용

original\_text = stt\_func(file\_root)

print("original text:" + original\_text)

(생략)

translated\_text = translate\_module(converted\_text)

#나온 텍스트를 return

return jsonify(text=converted\_text, text2=translated\_text)

elif request.method == 'GET':

return render\_template('upload.html')

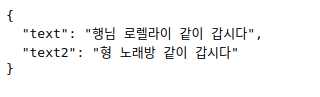
텍스트이(가) 표시된 사진

자동 생성된 설명

(음성 파일을 업로드 하는 서버, 사이트 화면)

텍스트이(가) 표시된 사진

자동 생성된 설명(음성 인식 결과창 1)

(음성 인식 결과창 2)

중앙 서버에서는 2가지 기능을 제공한다.

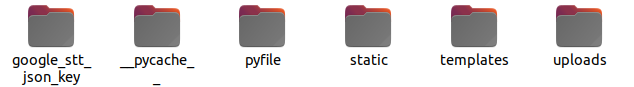
1. 음성인식 번역 기능

http://(서버 주소)/file\_upload 에 POST로 접속하여, AudioEncoding.LINEAR16, 44100hz 형식으로 인코딩된 오디오 파일을 넣으면 JSON Type으로 text(입력한 사투리 문장), text2(번역된 표준어 문장)을 반환 한다.

1. 텍스트 번역 기능

http://(서버 주소)/text\_only\_test/문자열 에 GET으로 접속하여. 문자열 부분에 사투리 문장을 파라미터로 넣으면, 음성 인식 과정 없이 JSON Type으로 text(입력한 사투리 문장), text2(번역된 표준어 문장)을 반환 한다.

번외로 단순히 http://(서버 주소)/ 로 접속한 경우에는 음성인식 번역 기능의 오디오 업로드 화면으로 들어간다.



업로드된 음성 파일은 서버 PC의 uploads에 임시 보관된다.

### 번역 프로그램 모듈 연결

from config import myConfig

from transformer import build\_model, do\_predict

def translate\_module(str):

## 모델 불러오기

vocab, config = myConfig()

model = build\_model(config)

model.load\_weights("wish\_translation.hdf5")

### input msg 주면 예측

input\_msg = str

result = do\_predict(model, 20, input\_msg)

print(f"output > {result}")

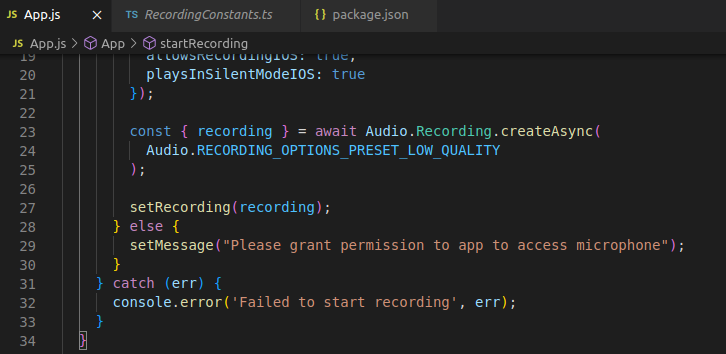
return result

번역 프로그램에 대한 자세한 설명은 위 연구 내용 3.1 ‘번역 파트’에서 확인할 수 있다.

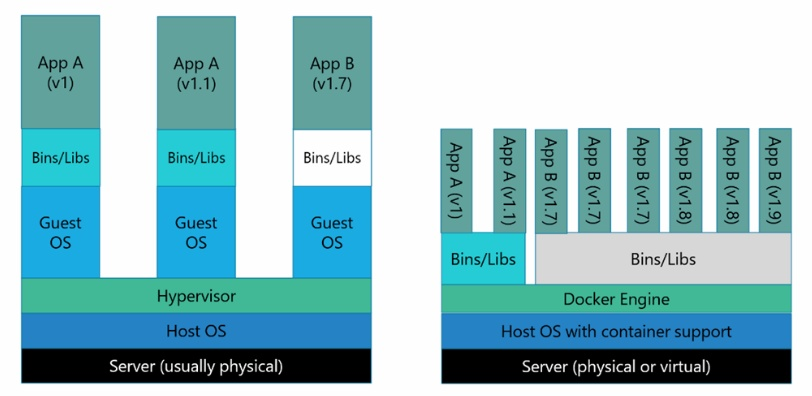
모듈 연결은 별도의 main() 함수에서 돌아가던 번역 프로그램을 translate\_module(str)라는 이름의 함수로 만들어 STT\_server.py에 연결하였다.

### 앱/웹 통신

앱/웹 통신은 앞서 설정한 중앙서버 내용 그대로 앱/웹에서 요청만 하면 되기에 큰 문제가 없지만, 요청하는 형식이 정해져 있고, 특히 오디오 형식이 고정되어 있기 때문에 오디오 인코딩을 잘 모른다면 헷갈리기 쉽다. 다음은 React Native에서 요청 시 인코딩 형식이다. (44100hz, low\_quality)



### Docker로 React + Nginx 프로젝트 관리하기



VM 가상화 플랫폼 vs Docker 가상화 플랫폼

VM같은 경우엔 Host OS 위에 가상화를 시키기 위한 Hypervisor 엔진 그리고 그 위에Guest OS를 올려 사용한다. 이는 가상화된 하드웨어 위에 OS가 올라가는 형태로 거의 완벽하게 Host와 분리된다고 봐도 무방하다. 반면에 컨테이너 기반 가상화는 Docker 엔진 위에 Application 실행에 필요한 바이너리만 올라가게 된다. OS 가상화를 보면 Host OS와 완전히 분리되는 장점은 있지만 OS위에 OS를 올리기 때문에 무겁고 느릴수 밖에 없다. 하지만 컨테이너 기반 가상화는 Host OS 그리고 Docker 엔진 위에서 바로 동작하며 Host의 커널을 공유한다. 커널을 공유하게 되면 io처리가 쉽게 되어 성능의 효율을 높일 수 있다.

Docker를 사용하면 코드를 더 빨리 전달하고, 애플리케이션 운영을 표준화하고, 코드를 원활하게 이동할 수 있다. 또한 리소스 사용률도 높여 효율적으로 관리할 수 있다.

Docker는 코드를 실행하는 표준 방식을 제공한다. 또한 컨테이너를 위한 운영 체제이다.

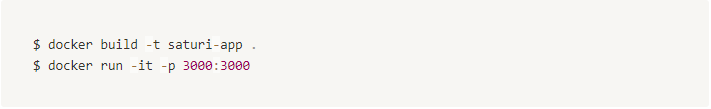
가상 머신이 서버 하드웨어를 가상화하는 방식과 비슷하게 컨테이너는 서버 운영 체제를 가상화한다. Docker는 각 서버에 설치되며 컨테이너를 구축, 시작, 또는 중단할 수 있는 간단한 명령어를 제공한다.

Docker를 사용하면 코드를 더 빨리 전달하고, 애플리케이션 운영을 표준화하고, 코드를 원활하게 이동할 수 있다. 또한 리소스 사용률도 높여 효율적으로 관리할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Dockerfile 생성



Docker Image 생성 후, Container 만들고 실행

텍스트이(가) 표시된 사진

자동 생성된 설명

Dockerfile-prod 생성

텍스트이(가) 표시된 사진

자동 생성된 설명

default.conf 생성

텍스트이(가) 표시된 사진

자동 생성된 설명

Dockerfile-prod 빌드 후, 80포트로 실행

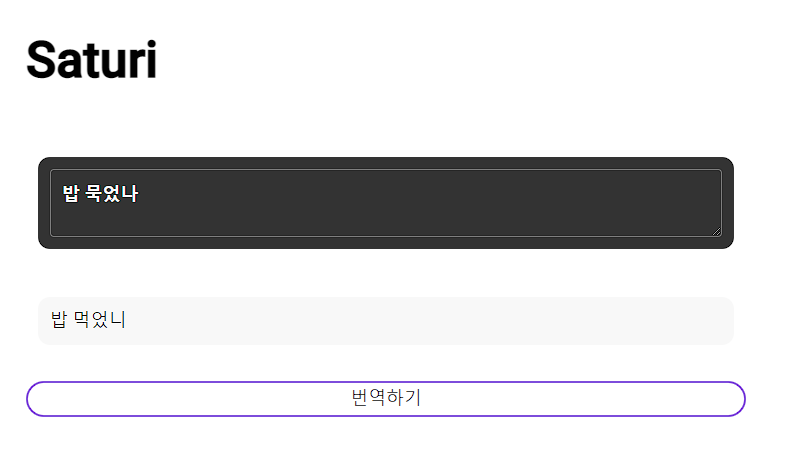
### React로 경상도 사투리 번역사이트 개발

웹의 경우, 연구실 서버가 Https를 지원하지 않아, 음성 녹음 권한에 있었다. 따라서 음성 API를 거치지 않고, 직접 번역 모델과 연결하여, 경상도 사투리 텍스트를 표준어 텍스트를 번역해주는 웹을 개발 및 배포하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

입력한 경상도 사투리의 텍스트를 받아, 표준어 번역 API로 전송 후, 표준어 결과값을 돌려 받는다.



웹 사이트 스크린샷

### React-Native로 iOS/Android 앱 개발 및 배포

리액트 네이티브는 Javascript 기반 프로그래밍 언어로 IOS와 안드로이드 앱을 한 번에 개발할 수 있는 크로스 플랫폼이다. 리액트 네이티브는 네이티브 브릿지(Native Bridge)를 통하여 스레드(Native Thread)와 통신하기 때문에, 웹앱과 다르게 성능에 최적화되어 있다.

리액트는 컴포넌트 기반으로 기능을 분리하여 재사용할 수 있고, IOS와 안드로이드 앱을 동시에 개발할 수 있기 때문에 개발을 효율적으로 할 수 있다.

Expo는 리액트 네이티브를 베이스로 한 것으로 앱을 간편하게 빌드, 배포를 할 수 있도록 도와주는 프레임워크이다. Expo는 바닐라 리액트 네이티브로 앱을 개발할 때 필요한 초기 환경 설정과 빌드를 쉽게 해주는 장점이 있다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

휴대폰의 음성 녹음 권한을 받고, 녹음을 시작하는 코드이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

음성 API에 사투리가 녹음된 파일을 전송하면, 표준어로 번역된 텍스트 결과값을 받는다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

앱 스크린샷

마이크 버튼을 누르고 사투리를 녹음하면, 일정 시간 뒤에 녹음한 사투리와 번역된 표준어 결과가 앱 화면에 뜨게 된다.

# 결론 및 향후 연구 방향

## 결론 및 향후 연구 방향

### STT, 서버 파트

먼저 STT 프로그램을 자체적으로 구현하지 못한다는 것이 이번 연구에서 가장 걸림돌이 되는 부분이었다고 생각한다. 현재 대기업들에서 구현한 STT API들도 표준어 기반이고, 영어권이라면 음성 소스가 많고, 개발이 많이 되지만, 한국어 환경의 음성 인식 기능은 현재도 그리 많지 않다. 본인이 파형 분석에 대한 지식이 있었다면 좋았을 것이라는 것과 만약 있었다 하더라도, 그 많은 사투리 음성 소스를 어디서 구할 것인가 하는 문제점이 없지 않았다. AI 허브 등의 사이트가 제공하는 데이터 베이스는 음원이 애매하고, 단어 간의 구별이나 문법적인 오류 등 때문에 한 언어를 구체화 하기에는 문제가 많다고 생각되었다. 향후 연구 방향은 음성 인식 프로그램을 자체 개발 할 수 있는가가 우선 될 것이다.

### 웹/앱

현재는 경상도 사투리만, 표준어로 번역해주는 기능 뿐입니다. 그리고 다른 번역기 서비스와 유사한 메인 화면을 가지고 있습니다. 경상도의 사투리 번역의 정확도를 좀 더 높이고, 타 지역의 사투리 모델까지 개발이 된다면, 전국의 사투리를 유용하게 번역해주는 서비스가 될 것입니다.

### 방언/표준어 번역

영어의 경우 트위터에 사용되는 문장들을 사전학습을 시킨 모델이 있을 정도로 여러 방면의 사전학습 모델들이 존재한다. 하지만 아직 한국에서는 표준어를 사전학습 시킨 모델만 있어 다양성 있는 도전을 하기가 어렵다. 또한 질 높은 데이터셋을 만들기가 어려웠으며 이는 차 후 기회가 된다면 서비스를 제공하면서 유저들이 사용한 데이터를 바탕으로 모델을 파인튜닝하는 방식으로 모델을 학습하는 방법을 사용하면 완화할 수 있을 것이다.

## 구성원별 역할

|  |  |
| --- | --- |
| **이름** | **역할** |
| 김도영 | * 앱/웹 UI/UX 디자인 및 설계 * Docker 사용, 웹 개발 및 배포 * iOS/Android 앱 개발 및 배포 |
| 김윤종 | * 경상도-표준어 모델 학습 * 영화 데이터셋 구축 * 딥러닝 모델 모듈화 |
| 황승찬 | * STT(Speech-To-Text) API 구축 * 데이터 처리 및 반환을 위한 중앙 서버 구축 (HTTP 통신을 통한 JSON Type 데이터 요청 및 전송) * 다른 파트와 연결(앱/웹 통신, 번역 프로그램 모듈화) |

## 개발 일정

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7월 | | | | 8월 | | | | 9월 | | | |
| 1주 | 2주 | 3주 | 4주 | 1주 | 2주 | 3주 | 4주 | 1주 | 2주 | 3주 | 4주 |
| STT API | | |  |  |  |  |  |  |  |  |  |
|  | stem analyzer | | |  |  |  |  |  |  |  |  |
|  |  | XML Parsing | | |  |  |  |  |  |  |  |
|  |  |  | Database | | |  |  |  |  |  |  |
|  |  |  |  | Web Server(Flask) | | | |  |  |  |  |
|  |  |  |  | Translation Function | | | |  |  |  |  |
|  |  |  |  |  | Android Application | | | | | |  |
|  |  |  |  |  |  |  | Test and Deburg | | | |  |
|  |  |  |  |  |  |  |  | Presentation & Report | | | |

# 참고 문헌

|  |
| --- |
| * [1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014. * [2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in neural information processing systems, pp. 5998–6008, 2017.   [3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4171– 4186, Minneapolis, Minnesota, June 2019.   * [4] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. Online. Association for Computational Linguistics. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7871–7880, 2020. * [5] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. Technical report, OpenAI. 2018. |