KoSpeech : Korean Speech Recognition

한국어 음성 인식

19010692 이준호

이론적 설명 : Deep Speech2 모델

Deep Speech 2 : 음성 인식을 위해 설계된 심층 신경망 기반의 모델

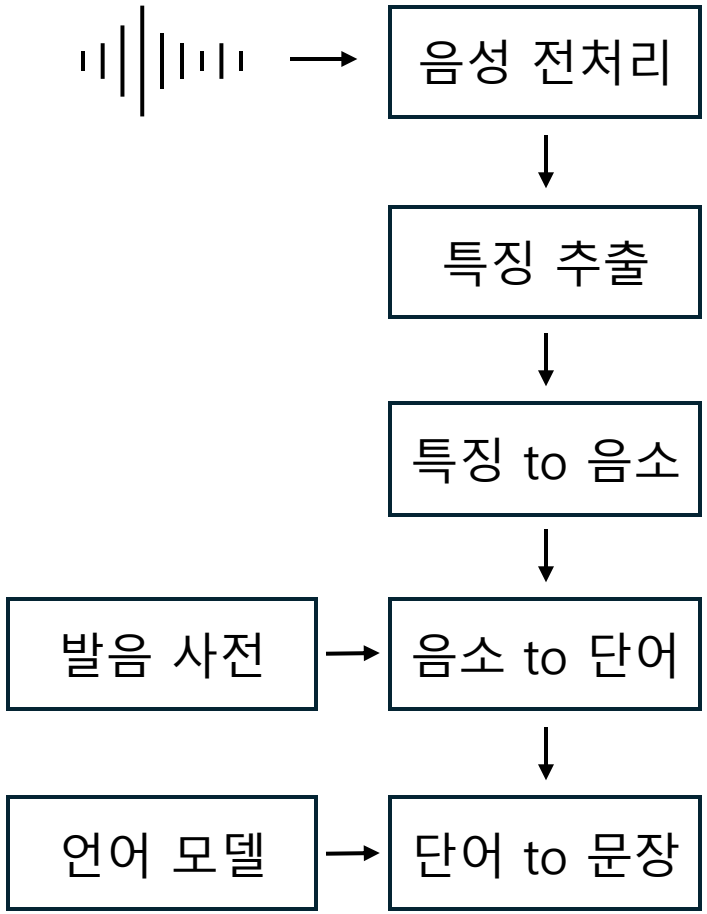
- ✓ End-to-End Speech Recognition으로 CTC를 사용하는 것이 특징
- ✓ 현재 **CER 14.81%** 에서 개선시키는 것을 목표
- ✓ 성능 개선 가능성이 가장 높다고 판단하여 해당 모델 선택

Feature	CER(%)
MFCC	17.31
Log Mel Spectrogram	15.79
Log Spectrogram	10.72
Filter Bank	***10.31***
CNN Extractor	**CER(%)**
Deep Speech 2	14.81
VGG Net	***10.31***
Attention Mechanism	**CER(%)**
Scaled Dot-Product Attention	14.81
Additive Attention	10.31
Location Aware Attention	13.52
Multi-Head Attention	***10.31***

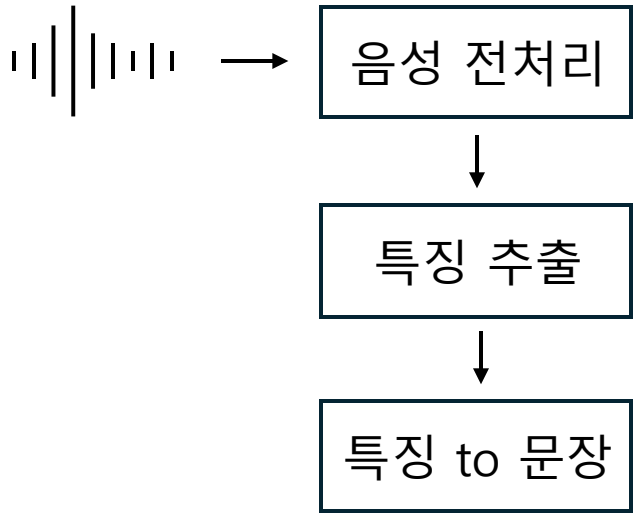
- https://www.sciencedirect.com/science/article/pii/S2665963821000026?ref=pdf_download&fr=RR-2&rr=8842f58d1b133519

이론적 설명 : End-to-End Speech Recognition

Deep Learning Boom 이전의 음성인식 방법

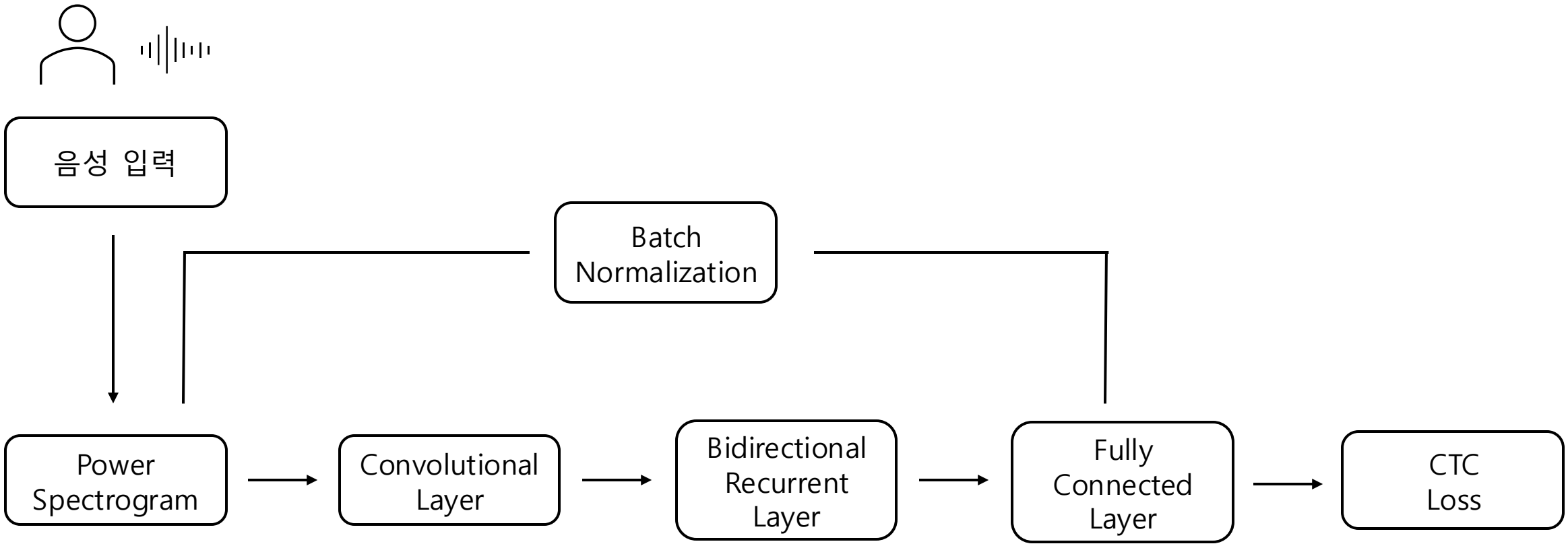


End-to-End 학습



✓ 입력과 출력을 전부 넣어서 문법과 발음까지 한꺼번에 모두 학습

이론적 설명 : Deep Speech 2 모델 구조

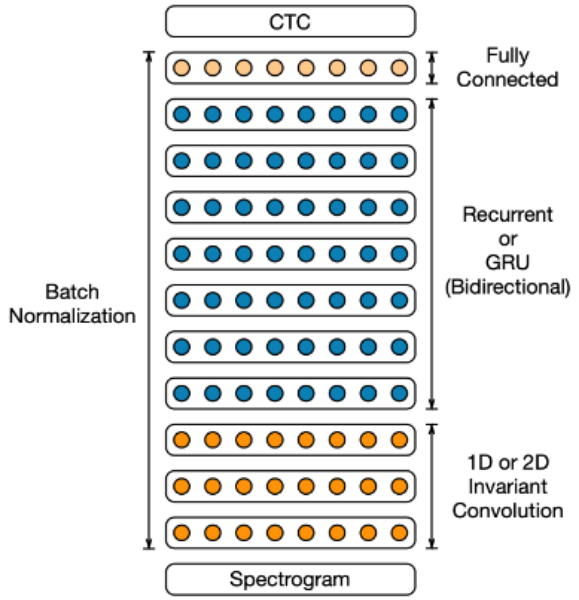


이론적 설명 : Deep Speech 2

- **Power Spectrogram**

  시간 – 주파수 영역에서 신호의 에너지를 시각적으로 표현

  음성 신호 분야에서 주로 사용



- **Convolution Layer**

  이미지와 비디오 분석하여 시각적 데이터를 처리

  시간과 주파수 두 개 축에 대해 동시에 컨볼루션 연산(2D conv, 3개 레이어)

  활성화 함수로는 Clipped ReLU를 사용 (아웃풋 최대값을 20으로 제한)

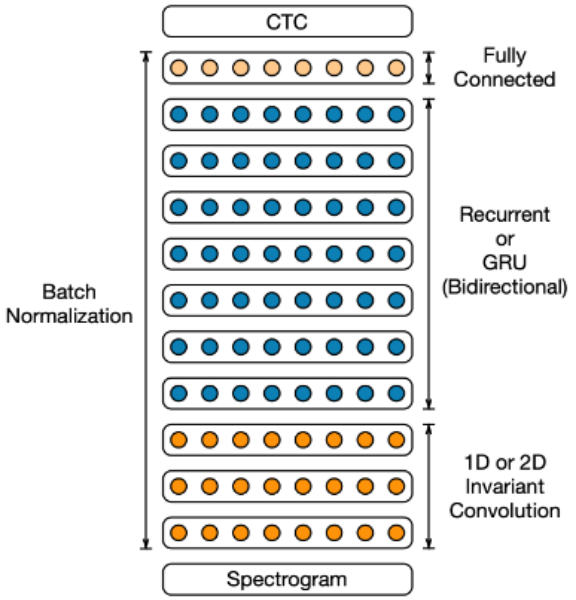
- https://www.sciencedirect.com/science/article/pii/S2665963821000026?ref=pdf_download&fr=RR-2&rr=8842f58d1b133519

이론적 설명 : Deep Speech 2

- **Bidirectional RNN Layer**

  양방향 RNN으로 시간 순서와 시간의 역순으로 처리하는 두 개의 별도 RNN 층을 사용 (Vanilla RNN, 7개 레이어)

  두 방향으로부터 출력이 통합되어 과거와 미래의 정보를 모두 반영



- **Fully Connected Layer**

  입력 특징들을 받아서 이를 기반으로 최종 출력을 예측 (1개 레이어)

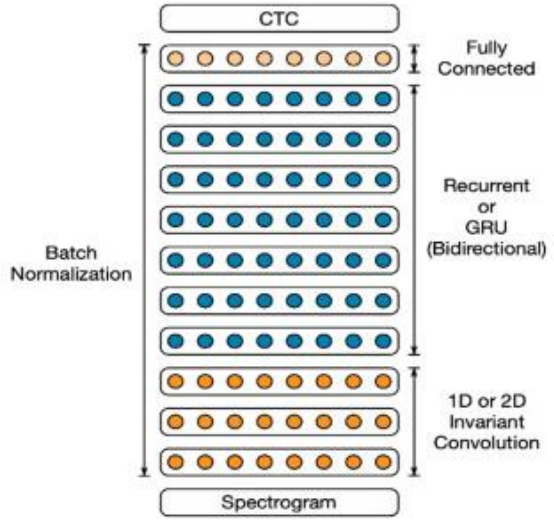
  모든 입력 뉴런이 다음 계층의 모든 뉴런과 연결되어 있는 구조

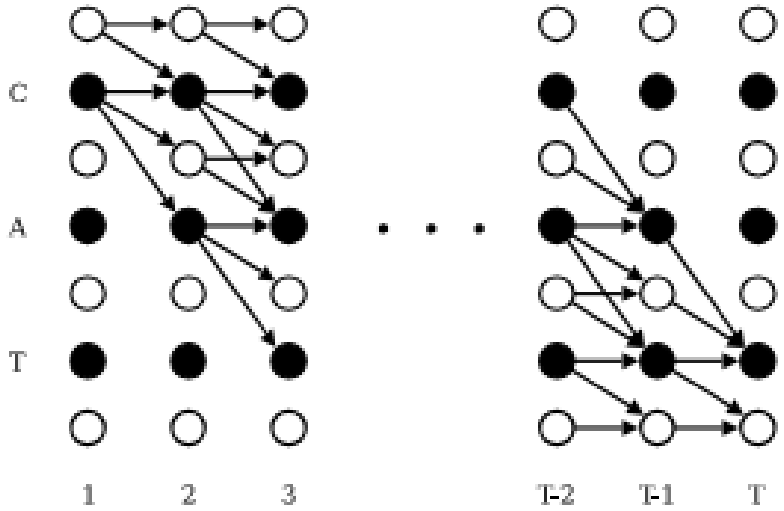
  활성화 함수로 SoftMax Layer를 통해 모델의 출력 값을 확률 분포 계산

- https://www.sciencedirect.com/science/article/pii/S2665963821000026?ref=pdf_download&fr=RR-2&rr=8842f58d1b133519

이론적 설명 : Loss Function

- **CTC** (Connectionist Temporal Classification)
- 입력 시퀀스 내의 각 시점에 대해 모든 가능한 출력 시퀀스의 확률을 계산하여 예측



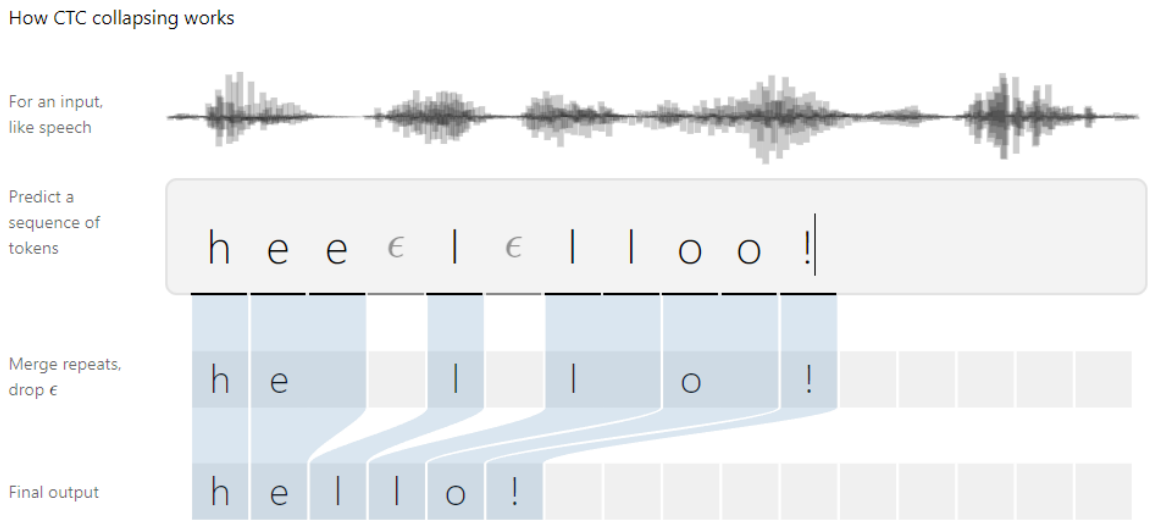


$$p(\mathbf{l}|\mathbf{x}) = \sum_{s=1}^{|\mathbf{l}'|} \frac{\alpha_t(s)\beta_t(s)}{y^t_{\mathbf{l}'_s}}.$$

✓ 모든 경우의 수를 계산하므로 시간이 오래 걸림 → Pytorch를 통해 해결

- https://www.sciencedirect.com/science/article/pii/S2665963821000026?ref=pdf_download&fr=RR-2&rr=8842f58d1b133519
- https://www.cs.toronto.edu/~graves/icml_2006.pdf

이론적 설명 : Loss Function



오디오 클립과 Transcript를 입력 받지만 어떤 단어의 Character가 Audio와 Alignment가 일치하는지 알 수 없음

Ex) 2초 후에 Hello를 말하나 곧바로 Hello를 말하나 Transcript는 동일하게 Hello

✓ **이와 달리 CTC는 Input과 Output의 가능한 Align을 모두 뽑아 계산하는 과정**

- https://distill.pub/2017/ctc/

이론적 설명 : Evaluation (성능 평가 지표)

- **CER** (Character Error Rate)
- 인식된 문자열(한 문자)과 정답 문자열(한 글자) 사이의 문자 오류 비율을 나타내는 지표

Character Error Rate (CER):

$$CER = \frac{\text{Number of incorrect characters}}{\text{Total number of characters in the reference text}} \times 100\%$$

- 정답

내	일	은	약	속	이	있	어	

- 인식

내	인		약	속	이	잇	어	요

$$CER = \frac{S + D + I}{N}$$

substitution = 2

deletion = 1

insertion = 1

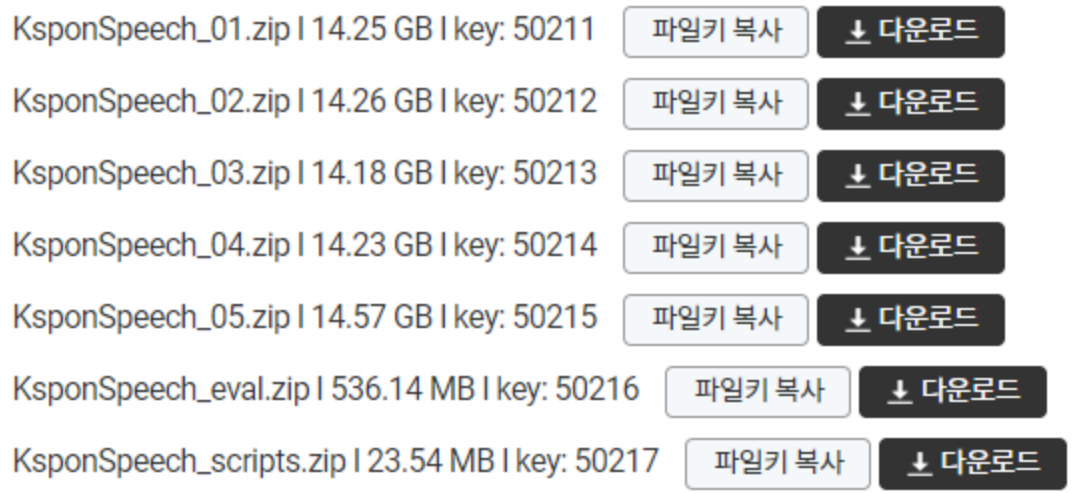
$$ED = \frac{2 + 1 + 1}{\text{Length of Ref.}} = \frac{4}{8} = 0.5$$

- https://medium.com/@tam.tamanna18/deciphering-accuracy-evaluation-metrics-in-nlp-and-ocr-a-comparison-of-character-error-rate-cer-e97e809be0c8

훈련 영상들 종류 및 다운로드 장소

데이터 : AIHUB에서 제공하는 KsponSpeech

https://www.aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100&aihubDataSe=realm&dataSetSn=123



- ✓ 대화형 음성인식 성능 개선을 위한 음향모델용 한국어 자유발화 음성데이터
- ✓ 조용한 환경에서 2000여명이 발성한 한국어 대화 음성 1,000 시간

- ✓ 총 622,545개의 PCM-TXT 파일로 구성

깃 허브 & 오픈 소스 코드

https://github.com/sooftware/kospeech

- Raw Data

"b/ 아/ 모+ 몬 소리야 (70%)/(칠 십 퍼센트) 확률이라니 n/"

- b/, n/, / .. 등의 잡음 레이블 삭제

"아/ 모+ 몬 소리야 (70%)/(칠 십 퍼센트) 확률이라니"

- 제공된 (철자전사)/(발음전사) 중 발음전사 사용

"아/ 모+ 몬 소리야 칠 십 퍼센트 확률이라니"

- 간투어 표현 등을 위해 사용된 '/', '*', '+' 등의 레이블 삭제

"아 모 몬 소리야 칠 십 퍼센트 확률이라니"

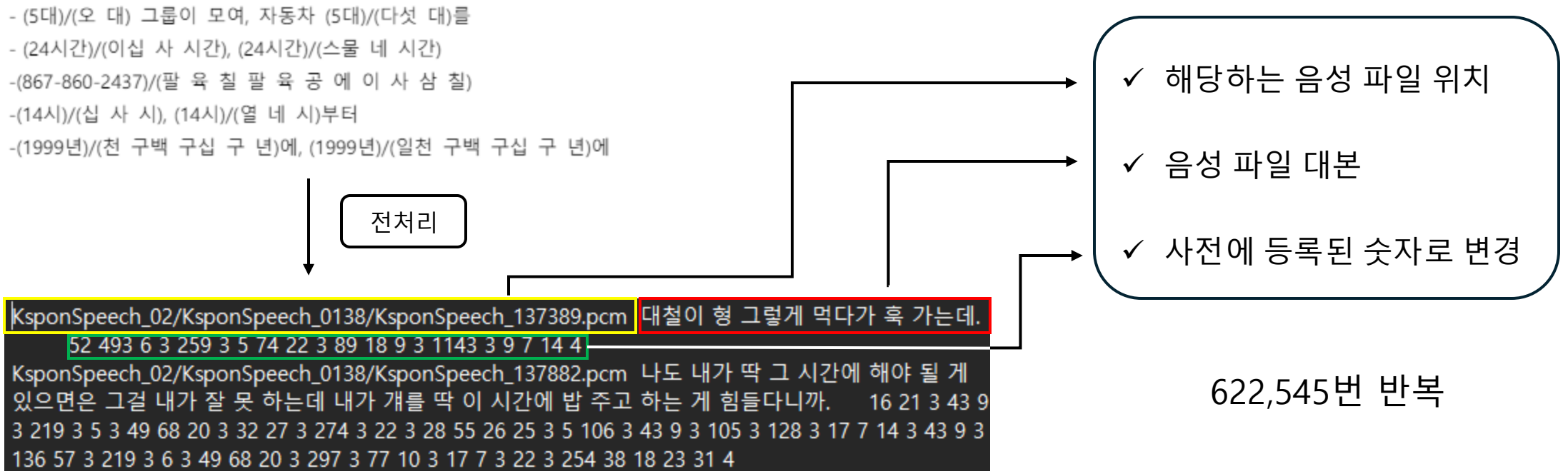
Supported Models

Acoustic Model	Notes
Deep Speech 2	2D-invariant convolution & RNN & CTC
Listen Attend Spell (LAS)	Attention based RNN sequence to sequence
Joint CTC-Attention LAS	Joint CTC-Attention LAS
RNN-Transducer	RNN Transducer
Speech Transformer	Convolutional extractor & transformer
Jasper	Fully convolutional & dense residual connection & CTC
Conformer	Convolution-augmented-Transformer

✓ 전처리 예시와 다양한 모델들 제공

전처리 과정

제공된 대본(Scripts)을 사전에 등록된 숫자로 변경



개선 방향 : Attention 적용

다양한 Attention 중 **Self-Attention** 과 **Cross-Attention**을 적용할 생각

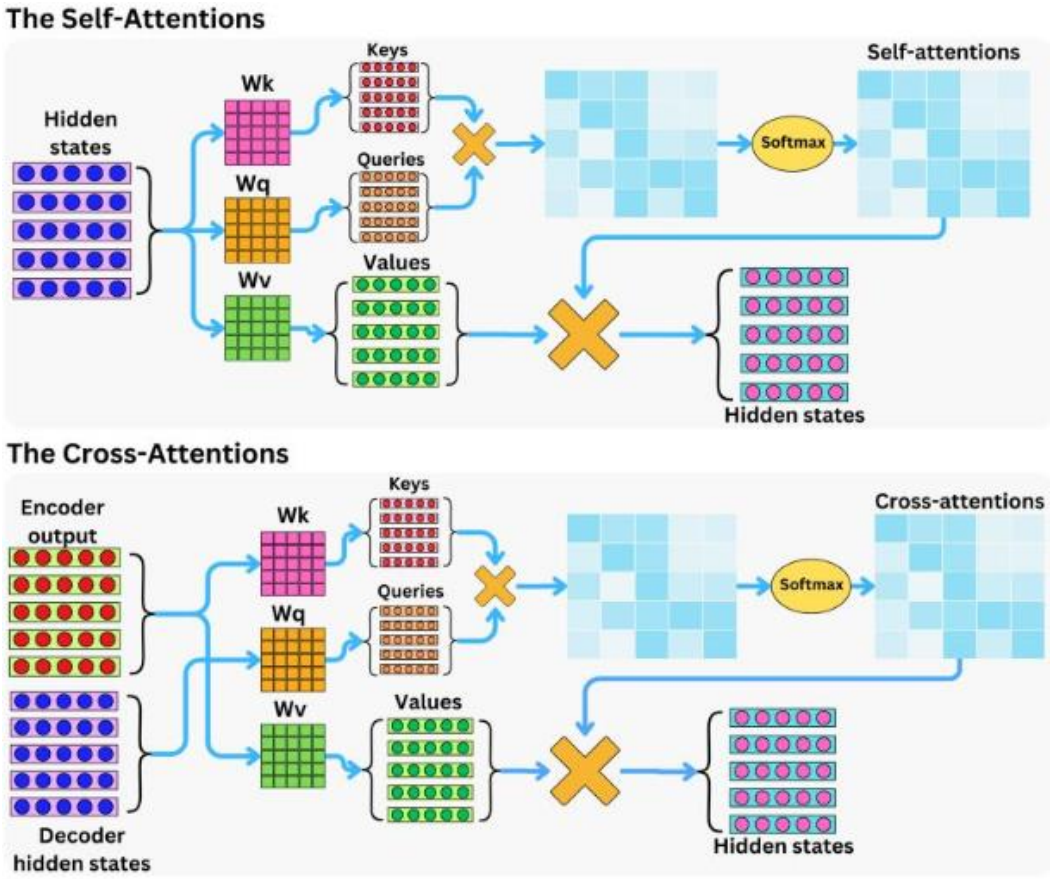
Self-Attention
시퀀스 내의 각 요소가 동일한 시퀀스 내의 다른 요소들과 관련 있는 지를 판단

Cross-Attention
입력 시퀀스와 출력 시퀀스 간의 상호 관계를 판단 → 입력 음성과 출력 텍스트 관계 판단

Global-Attention
전체 입력을 고려하므로 특정 관계를 더 세밀하게 파악하지 못해 적절하지 않다고 판단

Local-Attention
입력 시퀀스의 일부 만을 고려하여 제한된 범위 내에서만 동작하므로 적절하지 않다고 판단

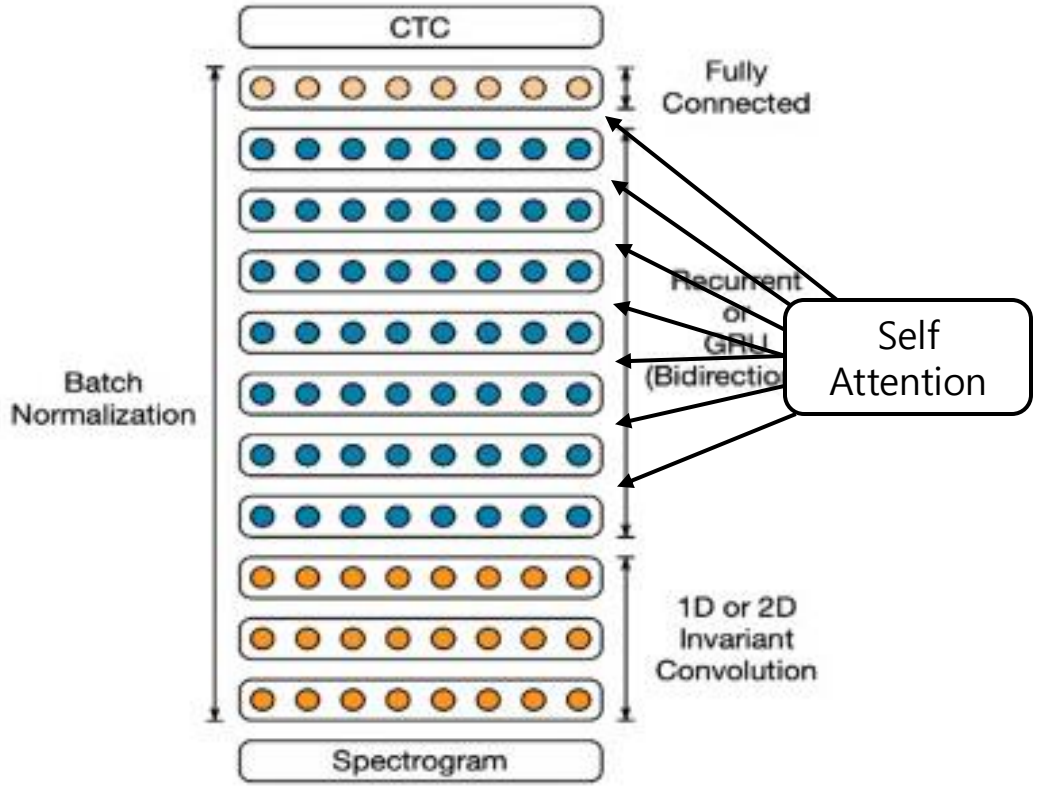
개선 방향 : Self-Attention, Cross-Attention



Self-Attention	Cross-Attention
동일한 Attention Mechanism을 사용	
둘 다 시퀀스 내의 컨텍스트 정보를 활용	
하나의 시퀀스 내 요소들 사이의 상호작용을 모델링	두 개의 다른 시퀀스 사이의 상호작용을 모델링

- https://www.linkedin.com/posts/damienbenveniste_machinelearning-datascience-artificialintelligence-activity-7136039194711257089-fqBN

개선 방향 : Self-Attention



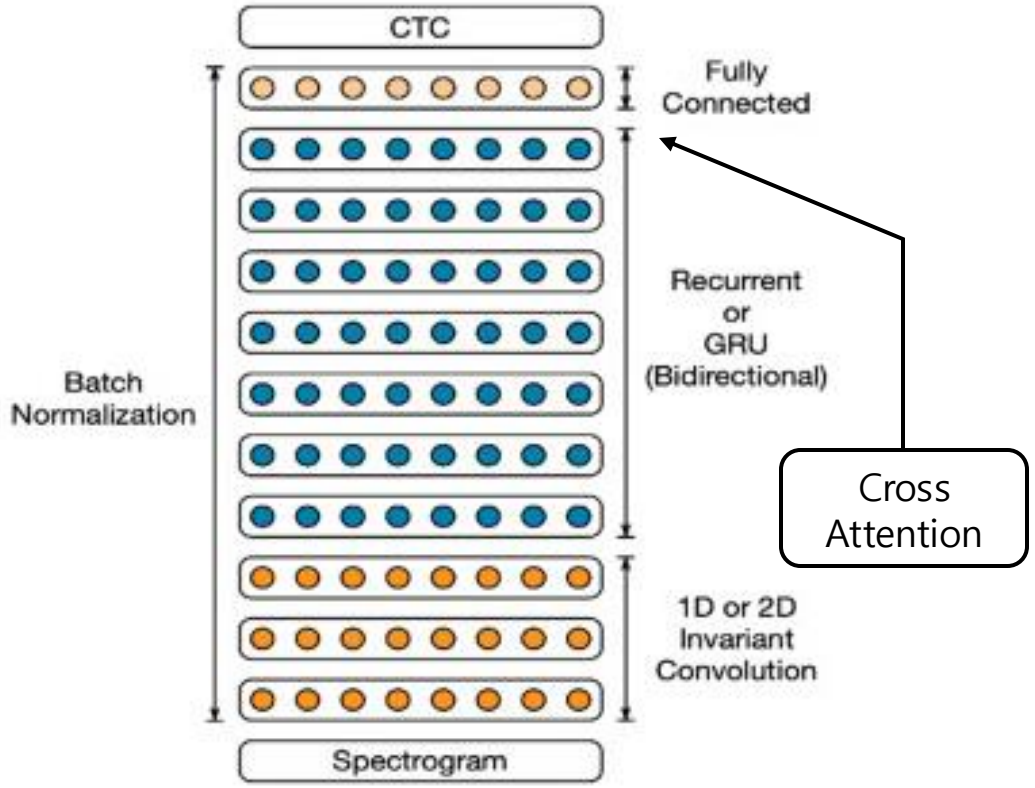
DeepSpeech2 모델에는 각 시퀀스 요소의 중요도를 고려하는 요소가 없으며 기울기 소실 문제

Self-Attention은 시퀀스 내 모든 요소 간의 직접적인 상호작용으로 다른 요소와의 관계를 동시에 학습

✓ Self-Attention을 추가하여 기울기 소실문제를 완화 시키고 이는 모델의 성능을 향상 시킬 것으로 판단

- https://www.sciencedirect.com/science/article/pii/S2665963821000026?ref=pdf_download&fr=RR-2&rr=8842f58d1b133519

개선 방향 : Cross-Attention



DeepSpeech2 모델은 시퀀스 내의 시간적 특성에 중점을 두지만 입력과 출력 간의 직접적인 상호 작용 X

Cross-Attention을 추가함으로써 입력 오디오 시퀀스와 출력 텍스트 시퀀스 간의 상호 관계를 고려 가능

✓ 음성과 텍스트 간의 관계를 더욱 명확하게 학습해 음성인식의 정확도를 높이는데 효과적일 것

- https://www.sciencedirect.com/science/article/pii/S2665963821000026?ref=pdf_download&fr=RR-2&rr=8842f58d1b133519

코드 세부 설명 : transcript 생성

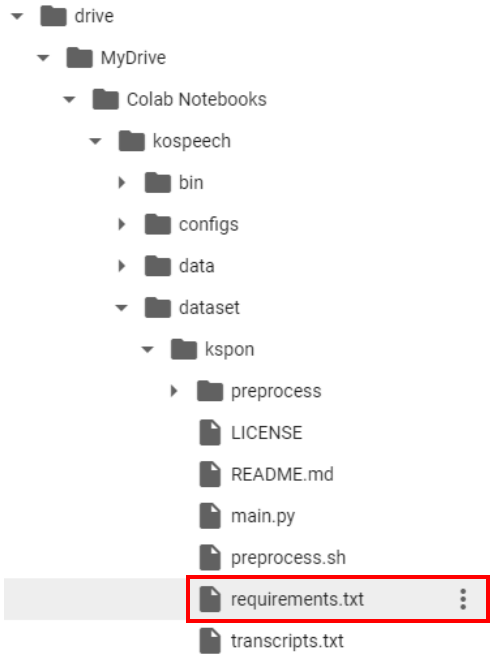
```
!git clone https://github.com/sooftware/kospeech.git
```

✓ 깃 허브 코드 클론

```
cd "/content/drive/My Drive/Colab Notebooks/kospeech/dataset/kspon"
```

```
pip install -r requirements.txt
```

✓ requirements.txt 설치




```
!python main.py --dataset_path "/content/drive/My Drive/Colab Notebooks/ksponSpeech" --vocab_dest "/content/drive/My Drive/Colab Notebooks/kospeech"
--savepath "/content/drive/My Drive/Colab Notebooks/ksponSpeech" --output_unit "character" --preprocess_mode 'phonetic'
```

✓ 데이터 셋 경로, vocab 위치 경로, 저장 위치와 원하는 전처리 방식 선택

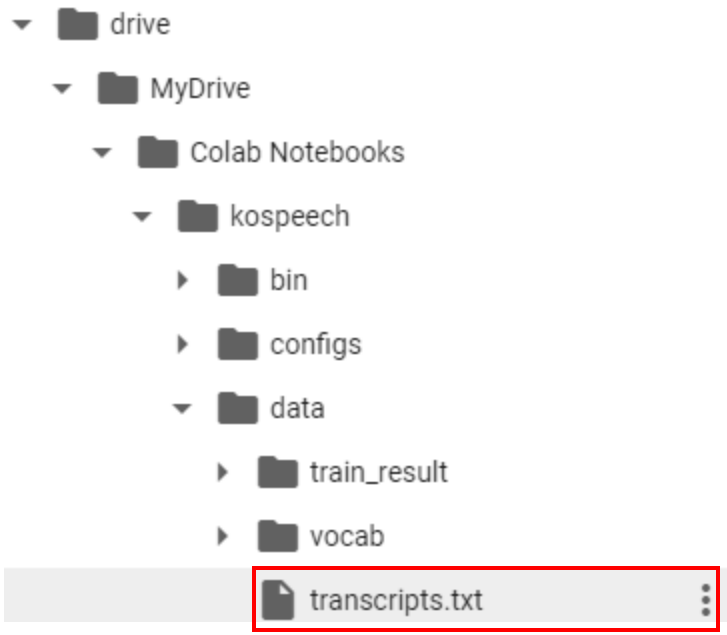
코드 세부 설명

```
from google.colab import drive
drive.mount('/content/drive')
```

✓ Google drive 마운트

```
!pip install hydra-core --upgrade
!pip install librosa
!pip install astropy
!pip install Levenshtein
!pip install torchaudio
```

✓ 사전 라이브러리 설치



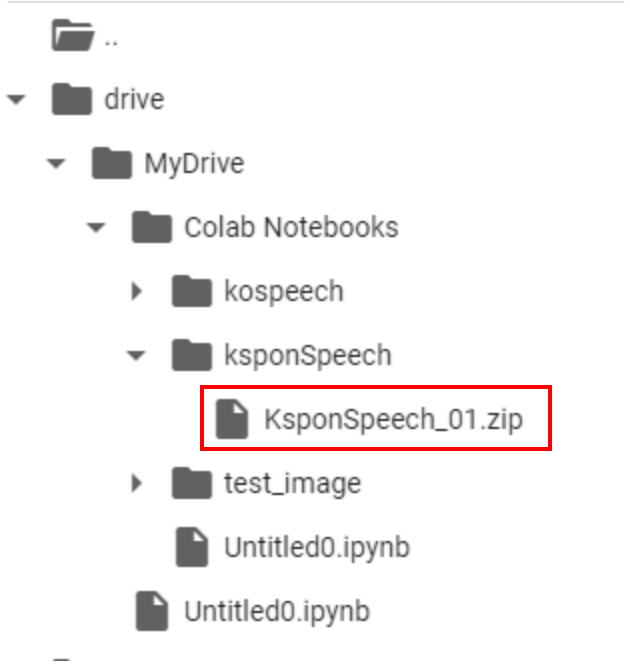

```
file_path = "/content/drive/My Drive/Colab Notebooks/kospeech/data/transcripts.txt"

count = 0

with open(file_path, 'r', encoding='utf-8') as file:
    for line in file:
        count += 1

print(f'총 데이터 개수: {count}')
# kospeech/data/dataloder.py 에서 split_dataset train, validation 개수 수정
# 9500/500
```

✓ transcripts.txt 경로 설정 후 총 데이터 개수 측정

✓ Dataloder.py 파일에서 Train과 Validation 개수 설정

코드 세부 설명 : main.py 학습

```
import zipfile
import os

base_file_path = "/content/drive/My Drive/Colab Notebooks/ksponSpeech/KsponSpeech_01.zip"
extract_to = "/content/Extracted_Files"

if not os.path.exists(extract_to):
    os.makedirs(extract_to)

with zipfile.ZipFile(base_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_to)
```



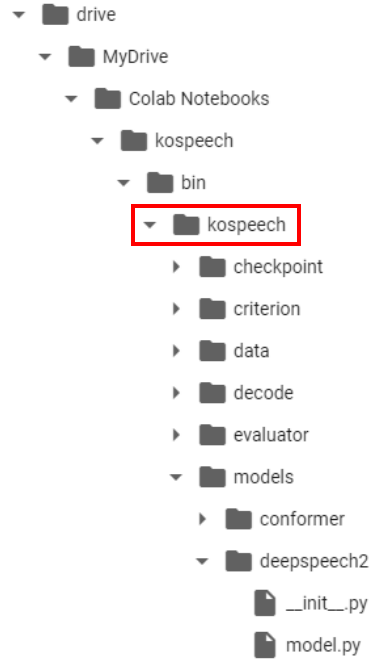
- ✓ 압축한 데이터셋 압축 해제
- ✓ KsponSpeech_01의 상위 폴더 10개만 사용

```
!python "/content/drive/My Drive/Colab Notebooks/kospeech/bin/main.py" model=ds2 train=ds2_train train.dataset_path="/content/Extracted_Files"
train.transcripts_path="/content/drive/My Drive/Colab Notebooks/kospeech/data/transcripts.txt"
```

- ✓ Main.py 파일의 경로와 압축 해제한 폴더 경로 설정
- ✓ 참고하는 transcripts.txt 파일 결로 설정 후 학습 시작

코드 세부 설명 : main.py

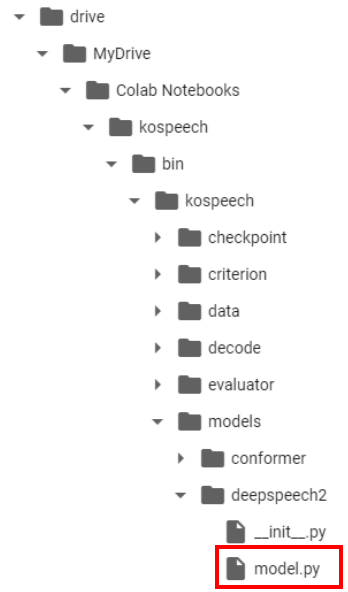
```
KSPONSPEECH_VOCAB_PATH = '/content/drive/My Drive/Colab Notebooks/kospeech/data/vocab/kspon_sentencepiece.vocab'
KSPONSPEECH_SP_MODEL_PATH = '/content/drive/My Drive/Colab Notebooks/kospeech/data/vocab/kspon_sentencepiece.model'
LIBRISPEECH_VOCAB_PATH = '/content/drive/My Drive/Colab Notebooks/kospeech/data/vocab/tokenizer.vocab'
LIBRISPEECH_TOKENIZER_PATH = '/content/drive/My Drive/Colab Notebooks/kospeech/data/vocab/tokenizer.model'


def train(config: DictConfig) -> nn.DataParallel:
    random.seed(config.train.seed)
    torch.manual_seed(config.train.seed)
    torch.cuda.manual_seed_all(config.train.seed)
    device = check_envirionment(config.train.use_cuda)
    if hasattr(config.train, "num_threads") and int(config.train.num_threads) > 0:
        torch.set_num_threads(config.train.num_threads)

    vocab = KsponSpeechVocabulary(
    f'/content/drive/My Drive/Colab Notebooks/kospeech/data/vocab/aihub_{config.train.output_unit}_vocabs.csv',
    output_unit=config.train.output_unit,
    )
```



- ✓ Main.py 내부 경로 코랩 환경에 맞도록 수정
- ✓ Kospeech 폴더를 bin 내부로 위치 변경

코드 세부 설명 : deepSpeech2 모델 model.py

```python
class BNReluRNN(nn.Module):
    supported_rnns = {
        'lstm': nn.LSTM,
        'gru': nn.GRU,
        'rnn': nn.RNN,
    }

    def __init__(
            self,
            input_size: int,  # size of input
            hidden_state_dim: int = 512,  # dimension of RNN`s hidden state
            rnn_type: str = 'gru',  # type of RNN cell
            bidirectional: bool = True,  # if True, becomes a bidirectional rnn
            dropout_p: float = 0.1,  # dropout probability
    ):
        super(BNReluRNN, self).__init__()
        self.hidden_state_dim = hidden_state_dim
        self.batch_norm = nn.BatchNorm1d(input_size)
        rnn_cell = self.supported_rnns[rnn_type]
        self.rnn = rnn_cell(
            input_size=input_size,
            hidden_size=hidden_state_dim,
            num_layers=1,
            bias=True,
            batch_first=True,
            dropout=dropout_p,
            bidirectional=bidirectional,
        )

    def forward(self, inputs: Tensor, input_lengths: Tensor):
        total_length = inputs.size(0)

        inputs = F.relu(self.batch_norm(inputs.transpose(1, 2)))
        inputs = inputs.transpose(1, 2)

        outputs = nn.utils.rnn.pack_padded_sequence(inputs, input_lengths.cpu())
        outputs, hidden_states = self.rnn(outputs)
        outputs, _ = nn.utils.rnn.pad_packed_sequence(outputs, total_length=total_length)

        return outputs
```



은닉 상태 크기(hidden_state_dim)는 512로 설정

양방향 RNN을 활성화하고 Dropout 비율은 0.1로 설정

배치 정규화(batch_norm)는 ReLU 활성화 함수를 사용

패딩된 입력 시퀀스를 PackedSequence 형태로 변환 후 RNN 층에 입력 → RNN의 outputs를 원래 크기의 시퀀스로 복원

코드 세부 설명 : deepSpeech2 모델 model.py

```python
class DeepSpeech2(EncoderModel):
    def __init__(
        self,
        input_dim: int,
        num_classes: int,
        rnn_type='gru',
        num_rnn_layers: int = 5,
        rnn_hidden_dim: int = 512,
        dropout_p: float = 0.1,
        bidirectional: bool = True,
        activation: str = 'hardtanh',
        device: torch.device = 'cuda',
    ):
        super(DeepSpeech2, self).__init__()
        self.device = device
        self.conv = DeepSpeech2Extractor(input_dim, activation=activation)
        self.rnn_layers = nn.ModuleList()
        rnn_output_size = rnn_hidden_dim << 1 if bidirectional else rnn_hidden_dim

        for idx in range(num_rnn_layers):
            self.rnn_layers.append(
                BNReluRNN(
                    input_size=self.conv.get_output_dim() if idx == 0 else rnn_output_size,
                    hidden_state_dim=rnn_hidden_dim,
                    rnn_type=rnn_type,
                    bidirectional=bidirectional,
                    dropout_p=dropout_p,
                )
            )

        self.fc = nn.Sequential(
            nn.LayerNorm(rnn_output_size),
            Linear(rnn_output_size, num_classes, bias=False),
        )

    def forward(self, inputs: Tensor, input_lengths: Tensor) -> Tuple[Tensor, Tensor]:
        outputs, output_lengths = self.conv(inputs, input_lengths)
        outputs = outputs.permute(1, 0, 2).contiguous()

        for rnn_layer in self.rnn_layers:
            outputs = rnn_layer(outputs, output_lengths)

        outputs = self.fc(outputs.transpose(0, 1)).log_softmax(dim=-1)

        return outputs, output_lengths
```

RNN 레이어 수는 5, 히든 레이어 크기는 512로 설정

Dropout 비율은 0.1로 설정

활성화 함수로 hardtanh를 사용

Device 환경을 cuda로 사용

RNN 레이어의 출력을 LayerNorm과 Linear로 Fully connected Layer를 구성

출력은 log_softmax 함수를 사용

코드 세부 설명 : deepSpeech2 모델 + self-attention

직접 구현한 코드

```python
class SelfAttention(nn.Module):
    def __init__(self, input_size, num_heads, dropout_p):
        super(SelfAttention, self).__init__()
        self.num_heads = num_heads
        self.dropout = nn.Dropout(dropout_p)

        self.query_layer = nn.Linear(input_size, input_size)
        self.key_layer = nn.Linear(input_size, input_size)
        self.value_layer = nn.Linear(input_size, input_size)

        self.output_layer = nn.Linear(input_size, input_size)

    def forward(self, inputs):
        batch_size, seq_len, input_size = inputs.size()

        query = self.query_layer(inputs).view(batch_size, seq_len, self.num_heads, input_size // self.num_heads)
        key = self.key_layer(inputs).view(batch_size, seq_len, self.num_heads, input_size // self.num_heads)
        value = self.value_layer(inputs).view(batch_size, seq_len, self.num_heads, input_size // self.num_heads)

        attention_scores = torch.matmul(query, key.transpose(-2, -1)) / (input_size // self.num_heads) ** 0.5
        attention_weights = F.softmax(attention_scores, dim=-1)

        attended_values = torch.matmul(attention_weights, value)
        attended_values = attended_values.view(batch_size, seq_len, input_size)

        output = self.output_layer(attended_values)
        output = self.dropout(output)

        return output
```

기존과 비슷한 구조로 input_size, num_heads, dropout 비율을 받아 초기화

query, key, value 변환을 위한 3개의 선형 변환 레이어 생성

query와 key의 내적을 계산하여 attention_scores를 계산

attention_scores 에 softmax 함수를 적용하여 attention_weights를 계산

attention_weights 와 value의 행렬 곱을 계산하여 attended_value 값을 얻음

코드 세부 설명 : deepSpeech2 모델 + self-attention

직접 구현한 코드

```python
class DeepSpeech2(EncoderModel):
    def __init__(
            self,
            input_dim: int,
            num_classes: int,
            rnn_type='gru',
            num_rnn_layers: int = 5,
            rnn_hidden_dim: int = 512,
            dropout_p: float = 0.1,
            bidirectional: bool = True,
            activation: str = 'hardtanh',
            num_attention_heads: int = 8,
            device: torch.device = 'cuda',
    ):
        super(DeepSpeech2, self).__init__()
        self.device = device
        self.conv = DeepSpeech2Extractor(input_dim, activation=activation)
        self.rnn_layers = nn.ModuleList()
        self.attention_layers = nn.ModuleList()
        rnn_output_size = rnn_hidden_dim << 1 if bidirectional else rnn_hidden_dim

        for idx in range(num_rnn_layers):
            self.rnn_layers.append(
                BNReluRNN(
                    input_size=self.conv.get_output_dim() if idx == 0 else rnn_output_size,
                    hidden_state_dim=rnn_hidden_dim,
                    rnn_type=rnn_type,
                    bidirectional=bidirectional,
                    dropout_p=dropout_p,
                )
            )
            self.attention_layers.append(
                SelfAttention(
                    input_size=rnn_output_size,
                    num_heads=num_attention_heads,
                    dropout_p=dropout_p,
                )
            )

        self.fc = nn.Sequential(
            nn.LayerNorm(rnn_output_size),
            Linear(rnn_output_size, num_classes, bias=False),
        )
```

Num_attention_heads를 8로 입력 시퀀스의 관계를 학습하는 헤드 수를 설정

Input_size는 RNN 층의 output_size와 동일, 미리 정의해둔 attention_heads 수와 dropout 비율을 설정하는 Attention_layers 추가하여 적용

나머지 코드는 기존 DeepSpeech2 코드와 동일

코드 세부 설명 : deepSpeech2 모델 + self-attention

직접 구현한 코드

```python
def forward(self, inputs: Tensor, input_lengths: Tensor) -> Tuple[Tensor, Tensor]:
    outputs, output_lengths = self.conv(inputs, input_lengths)
    outputs = outputs.permute(1, 0, 2).contiguous()

    for rnn_layer in self.rnn_layers:
        outputs = rnn_layer(outputs, output_lengths)

    outputs = self.fc(outputs.transpose(0, 1)).log_softmax(dim=-1)

    return outputs, output_lengths
```



```python
def forward(self, inputs: Tensor, input_lengths: Tensor) -> Tuple[Tensor, Tensor]:
    outputs, output_lengths = self.conv(inputs, input_lengths)
    outputs = outputs.permute(1, 0, 2).contiguous()

    for idx, rnn_layer in enumerate(self.rnn_layers):
        outputs = rnn_layer(outputs, output_lengths)
        outputs = self.attention_layers[idx](outputs)

    outputs = self.fc(outputs.transpose(0, 1)).log_softmax(dim=-1)

    return outputs, output_lengths
```

Forward의 RNN for 루프 안에 Self-attention 층도 추가하여 RNN과 Self-attention이 반복하여 작동하도록 설정

처리중인 RNN 인덱스(idx)로 self.attention_layers[idx]에 사용되어, 각 RNN 레이어에 대응되는 Attention 레이어를 선택할 수 있도록 설정

실험 결과

KsponSpeech_01.zip I 14.25 GB I key: 50211

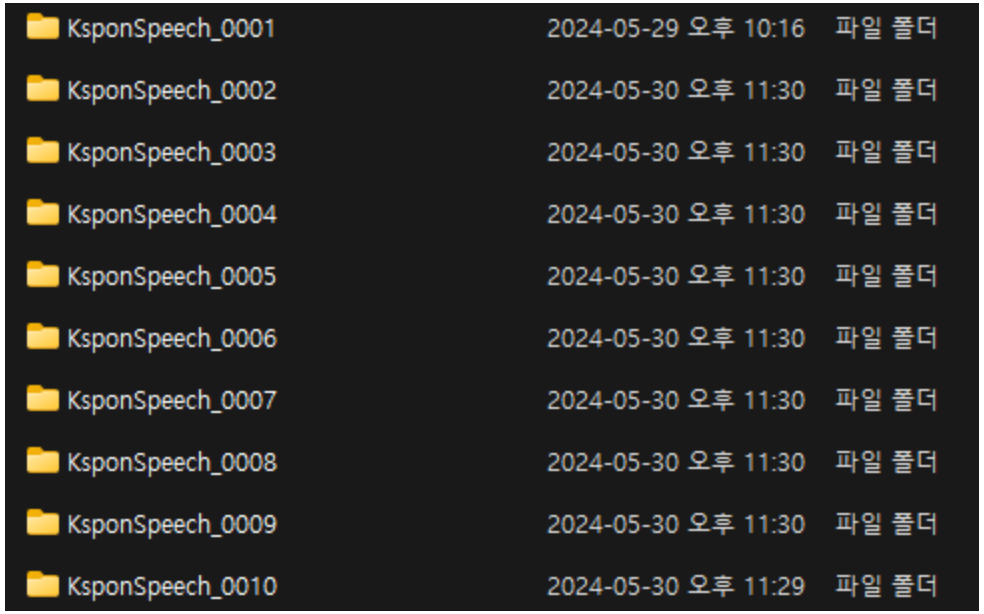
KsponSpeech_02.zip I 14.26 GB I key: 50212

KsponSpeech_03.zip I 14.18 GB I key: 50213

KsponSpeech_04.zip I 14.23 GB I key: 50214

KsponSpeech_05.zip I 14.57 GB I key: 50215

→ 100개 중 10개 선택 →



전체 데이터를 1epoch 학습시키는데 Colab T4 GPU 기준 10시간이 걸림

너무 오래 걸리고 Colab Pro 컴퓨팅 단위도 부족해 기간 내에 불가능 할거라 판단

10개의 폴더만 선택해서 진행하기로 결정

↓

폴더 하나당 1000개의 데이터

실험 결과

DeepSpeech2

```
loss : 0.0002898787083455852,
cer : 0.9293723719046877
```

```
Epoch 19 (Training) Loss 0.0003 CER 0.9291
```

DeepSpeech2 + Self-Attention

```
loss : 0.0003463518313753482 4,0.9932413275350512,
cer : 0.0003463335419658779,0.9932273509597914
```

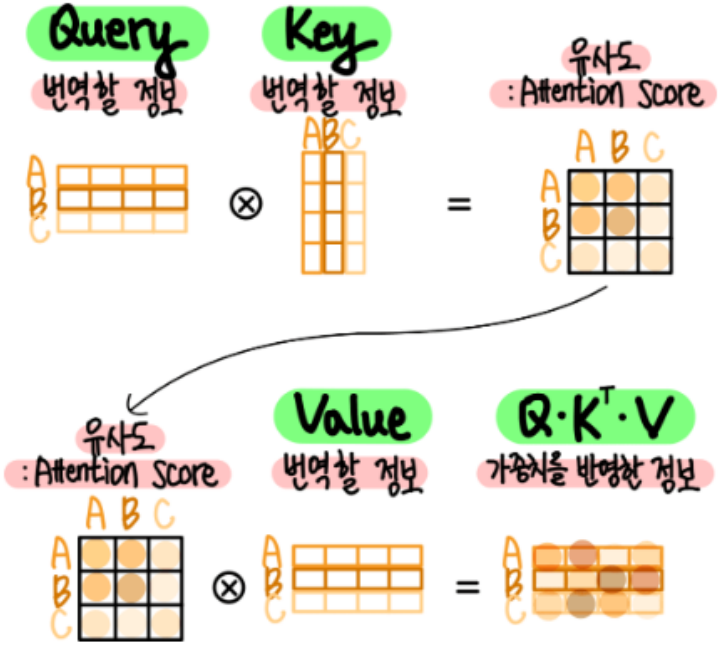
```
Epoch 19 (Training) Loss 0.0003 CER 0.9932
```

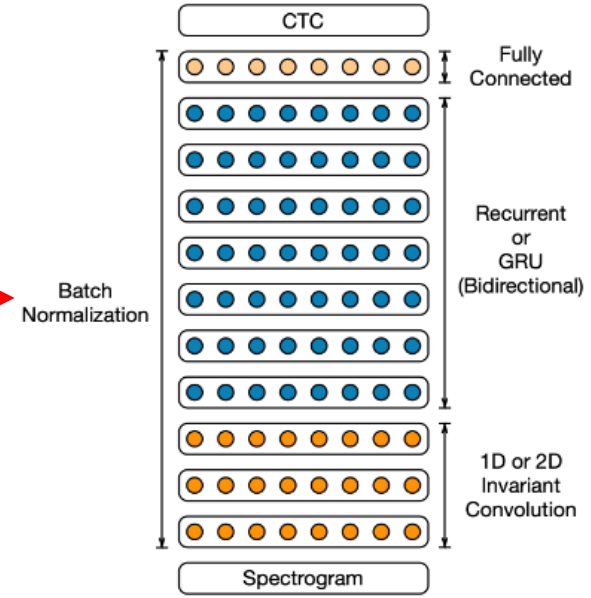
두 모델 모두 총 20 Epoch로 학습을 끝냄

둘 다 Loss는 0.0003으로 동일했지만 CER 값에서 미세한 차이가 존재

기존 DeepSpeech2 모델은 0.9291인 반면 Self-Attention을 추가한 모델은 0.9932로 약 0.641 증가함

분석 : 예상한 기대 효과





Self-Attention의 유사도를 통해 음성 신호 내의 토큰 간 관계를 잘 해석해 성능이 좋아질 것이라고 생각했음

Transformer와 같은 다른 모델에서 Self-Attention으로 장기 의존성 문제를 해결하는 경우가 많아 긴 음성 데이터에 대해서 정확도가 상승할 것이라 기대함

- https://ffighting.net/deep-learning-basic/%EB%94%A5%EB%9F%AC%EB%8B%9D-%ED%95%B5%EC%8B%AC-%EA%B0%9C%EB%85%90/attention-and-self-attention-in-deep-learning/
- https://www.sciencedirect.com/science/article/pii/S2665963821000026?ref=pdf_download&fr=RR-2&rr=8842f58d1b133519

분석 : 실패 원인 예상 1

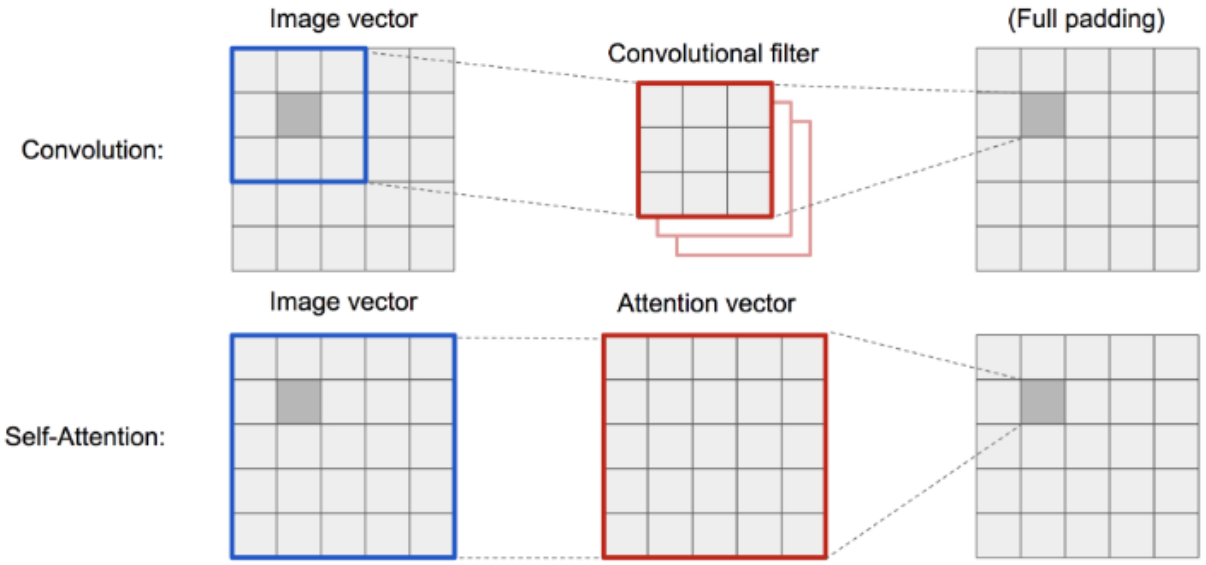
```
DataParallel(
  (module): DeepSpeech2(
    (conv): DeepSpeech2Extractor(
      (activation): Hardtanh(min_val=0, max_val=20, inplace=True)
      (conv): MaskCNN(
        (sequential): Sequential(
          (0): Conv2d(1, 32, kernel_size=(41, 11), stride=(2, 2), padding=(20, 5), bias=False)
          (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): Hardtanh(min_val=0, max_val=20, inplace=True)
          (3): Conv2d(32, 32, kernel_size=(21, 11), stride=(2, 1), padding=(10, 5), bias=False)
          (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (5): Hardtanh(min_val=0, max_val=20, inplace=True)
        )
      )
    )
    (rnn_layers): ModuleList(
      (0): BNReluRNN(
        (batch_norm): BatchNorm1d(640, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (rnn): GRU(640, 1024, batch_first=True, dropout=0.3, bidirectional=True)
      )
      (1-2): 2 x BNReluRNN(
        (batch_norm): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (rnn): GRU(2048, 1024, batch_first=True, dropout=0.3, bidirectional=True)
      )
    )
    (attention_layers): ModuleList(
      (0-2): 3 x SelfAttention(
        (dropout): Dropout(p=0.3, inplace=False)
        (query_layer): Linear(in_features=2048, out_features=2048, bias=True)
        (key_layer): Linear(in_features=2048, out_features=2048, bias=True)
        (value_layer): Linear(in_features=2048, out_features=2048, bias=True)
        (output_layer): Linear(in_features=2048, out_features=2048, bias=True)
      )
    )
    (fc): Sequential(
      (0): LayerNorm((2048,), eps=1e-05, elementwise_affine=True)
      (1): Linear(
        (linear): Linear(in_features=2048, out_features=2001, bias=False)
      )
    )
  )
```

해당 구조를 보면 RNN 레이어 3개, Self-Attention 레이어 3개 만 사용

기존 RNN 레이어 7개를 사용한 것에 비하면 적은 RNN 레이어를 사용해 자세한 학습이 부족했다고 생각

✓ **개선 방법** → 기존과 비슷한 수의 RNN 레이어를 사용하고 Self-Attention 레이어를 추가

분석 : 실패 원인 예상 2



음성 인식 문제는 시간적 순서와 문맥적 순서가 중요

그러나 모든 입력 간의 관계를 동등하게 고려하는 Self-Attention은 이를 효과적으로 구분하지 못 했을 가능성이 존재

✓ **개선 방법** → 시간과 문맥 순서를 고려하는 위치 인코딩 (Positional Encoding Attention)이나 시간 가중치 (Temporal weighted Attention) 을 사용하면 더 효과적일 거라 생각

- https://lilianweng.github.io/posts/2018-06-24-attention/

분석 : 실패 원인 예상 3

```yaml
# trainer
num_epochs: 20
batch_size: 32
save_result_every: 1000
checkpoint_every: 5000
print_every: 10
mode: 'train'
seed: 777
resume: false

# device
num_workers: 4
use_cuda: True

# optim
optimizer: 'adam'
init_lr: 1e-06
final_lr: 1e-06
peak_lr: 1e-04
init_lr_scale: 0.01
final_lr_scale: 0.05
max_grad_norm: 400
warmup_steps: 400
weight_decay: 1e-05
reduction: 'mean'
```

← kospeech/configs/train/ds2_train.yaml

```python
def __init__(
        self,
        input_dim: int,
        num_classes: int,
        rnn_type='gru',
        num_rnn_layers: int = 5,
        rnn_hidden_dim: int = 512,
        dropout_p: float = 0.1,
        bidirectional: bool = True,
        activation: str = 'hardtanh',
        num_attention_heads: int = 8,
        device: torch.device = 'cuda',
):
```

↑

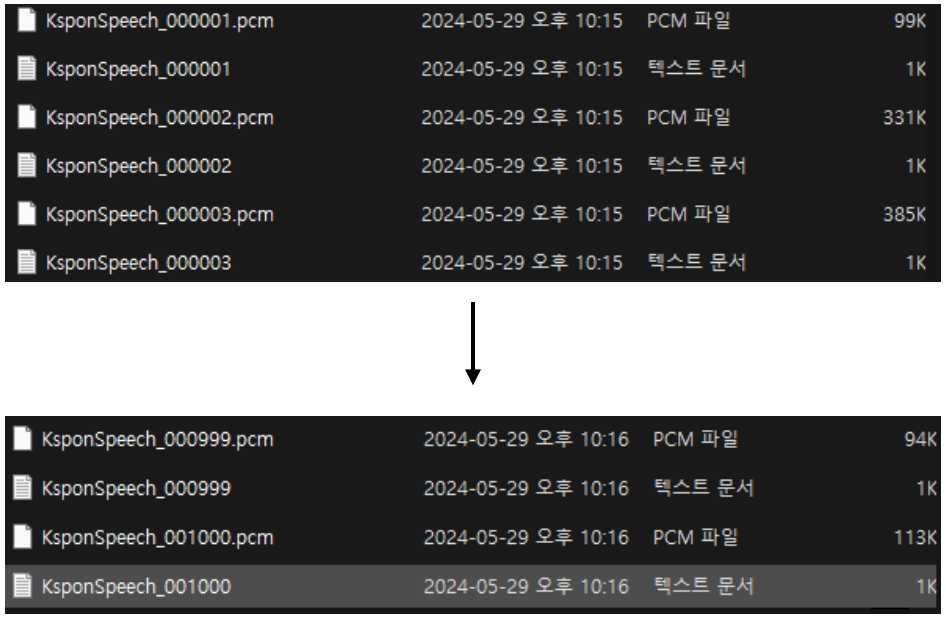
kospeech/bin/kospeech/models/
deepspeech2/model.py

하이퍼 파라미터의 최적화 문제

최적화 단계를 거치지 않고 기존 하이퍼 파라미터를 그대로 가져다 사용해 학습한 것이 문제가 되었을 가능성이 있음

✓ **개선 방법** → 최적화 알고리즘(Grid Search, Bayesian Optimization)을 사용한 후 학습 개선

분석 : 실패 원인 예상 4



1000 * 10

데이터 셋의 크기가 현저히 작았다는 점

총 622,545개의 데이터 중에 10,000개의 데이터 셋만 사용해 학습

데이터 셋의 크기가 작으면 모델이 데이터 내부의 패턴을 충분히 파악하기 어려웠을 가능성 있음

✓ **개선 방법** → 기존 DeepSpeech2와 동일하게 모든 데이터 셋을 학습시키거나 더 많은 양의 데이터 셋 학습이 필요

결론 및 아이디어의 의미

DeepSpeech2 모델에 Transformer의 핵심 기술인 Self-Attention 메커니즘을 적용하여 음성 인식 성능을 개선할거라 생각했음

입력 시퀀스 내의 모든 토큰들 간의 상호 관계를 학습하여 기존 RNN 기반 모델에 비해 문맥 정보를 더 효과적으로 활용 할 것이라 기대

↓

그러나 학습 데이터셋의 부족, 시간 부족, GPU 자원 부족 등 다양한 제약 조건으로 인해 결과적으로 모델 개선에 실패

더 적절한 Attention 메커니즘이나 하이퍼 파라미터 설정을 통해 더 나은 결과를 얻을 수 있었을 것으로 생각됨

또한 많은 양의 데이터셋과 충분한 시간, 자원이 확보된다면 Self-Attention 메커니즘을 통해 DeepSpeech2 모델의 성능 향상이 가능할 것으로 기대됨

↓

✓ 데이터셋, 시간, 자원 등의 제약 조건을 사전에 충분히 고려하지 못한 점이 부족했다고 생각

✓ 이번 프로젝트를 통해 개선 과정에서 고려해야 할 다양한 요소들을 파악하게 되어 도움이 되었음