종합설계 프로젝트 수행 보고서

프로젝트명	통화 데이터 기반 모바일 TTS 어플
팀번호	S3-2
문서제목	수행계획서( ) 2차발표 중간보고서( ) 3차발표 중간보고서( ) 학기말 발표 보고서( ) 최종결과보고서(O)

2020.12.03

팀원 : 이영수 (팀장)
윤영범
오민재

지도교수 : 노영주 교수 (인)

문서 수정 내역

작성일	대표작성자	버전(Revision)	수정내용	
2019.12.18	이영수(팀장)	1.0	수행계획서	최초작성
2020.03.01	윤영범	2.0	2차발표자료	설계서추가
2020.05.01	오민재	3.0	3차발표자료	시험결과추가
2020.06.24	윤영범	4.0	학기말 발표자료	서버&코드추가
		5.0	최종결과보고서	시험결과 수정

문서 구성

진행단계	프로젝트 계획서 발표	중간발표1 (2월)	중간발표2 (4월)	학기말발표 (6월)	최종발표 (10월)
기본양식	계획서 양식	계획서 양식	계획서 양식	계획서 양식	계획서 양식
포함되는 내용	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~3) 참고자료	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~4) 참고자료	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~5) 참고자료	Ⅰ. 서론 (1~6) Ⅱ. 본론 (1~7) 참고자료	Ⅰ Ⅱ Ⅲ

목 차

Ⅰ. 서론

1. 작품선정 배경 및 필요성

러닝 기술이 발전하면서 많은 사람들에게 알려지고 여러 분야에 도입되고 있다. 대표적으로 사용되고 있는 것이 텍스트를 음성으로 읽어내는 TTS(Text To Speech)이다. 구글과 네이버에서 현재 api를 제공하고 있다. 해당 서비스는 목소리가 한정되어 있다. 모델을 더 늘리기 위해서는 많은 데이터를 필요로 한다. 그러나 음성이라는 특징 때문에 데이터 수집에 많은 어려움이 있다. 따라서 데이터를 수집하기 위한 방법이 필요하다. 우리가 선택한 방법은 어플을 제작하고 사용자들에게 음성데이터를 받고 데이터가 일정 모이면 TTS 엔진으로 변환하여 제공하는 것이다. 결과적으로 개발자는 음성데이터를 모을 수 있고 사용자는 해당 엔진을 이용하여 여러 분야에 사용할 수 있다.

2. 기존 연구/기술 동향 분석

TTS 엔진 유료서비스로는 네이버 N-Voice, Voice Ware 가 존재하며 구글, 삼성, 애플의 스마트폰에 간단한 TTS 엔진이 내장되어있다. 다음은 같은 문장을 입력한 뒤 나온 결과를 분석한 표(표1)이다.

	목소리 설정	띄어쓰기 구분	한국어 제공	억양 관련
구글 TTS엔진 [1]	남/여	○	○	속도,음조 조절 가능
삼성 TTS엔진 [2]	남/여	○	○	x
아이폰 TTS 엔진[3]	여성 목소리	x	○	x
Voice Ware[4]	남/여 다수	o	o	변조 기능 제공
네이버 Clova Voice[5]	등록된 목소리	o	o	약간의 감정 추가
마이 보이스	개인 목소리	○	○	속도,음조 조절

TTS 프로그램 비교 표1

3. 개발 목표

통화 간에 사용자 목소리만 담을 수 있는 어플리케이션이 필요하다.
어플리케이션에서 얻은 데이터를 서버를 통해 데이터베이스에 저장한다.
머신러닝을 통해 얻어낸 데이터로 TTS 엔진을 만들어낸다.

4. 팀 역할 분담

이영수(팀장)	AWS 사용법 학습 및 서버 구축 서버와 어플리케이션 통신 Google Speech Api 사용법 러닝에 필요한 영상, 데이터 수집 FFmpeg 학습
윤영범	어플리케이션 탐색 및 개발 Tacotron 및 deepvoice 기법 학습, 적용 러닝에 필요한 영상, 데이터 수집 계획서, 보고서 제작
오민재	러닝 이론적 개념 학습 Tensorflow 사용법 학습 및 적용 발표자료 제작 머신러닝에 필요한 영상, 음성 데이터 수집 프로토타입 테스트

5. 개발 일정

항목	11,12	1	2	3	4	5	6	7	8,9월
요구사항 수집	→								
개발환경 조사 및 학습	→	→							
개발환경 구축		→	→						
TTS 개발		→	→	→	→				
서버 구축		→	→	→	→				
어플리케이션 개발				→	→				
개발 수정 및 데모 테스트						→	→	→	→
최종검토 및 발표									→

6. 개발 환경

- Google Speech API

개발사 : Google

- FFmpeg

개발사 : FFmpeg팀
사용 개발 언어 : C
라이선스 : LGPLv2.1+
사용 버전 : 4.2

- Python

개발사 : Python SW 재단
사용 개발 언어: Python
라이선스 : 파이썬 sw 재단 라이선스
사용 버전 : 3.7

- TensorFlow

개발사 : Google
사용 개발 언어 : Python
라이선스 : Apache 2.0
OpenSource 라이선스
사용 버전 :1.13.0

- Andrioid Studio

개발사 : Google , JetBrains
사용 개발 언어 : 자바, 코틀린 c++
라이선스 : Apache 2.0

- Amazon Web Service

개발사 : Amazon
사용 개발 언어 : 자바, python, php

- Ubuntu

개발사 : Canonical
사용 개발 언어 : python
버전 : 16.04

7. Github 주소

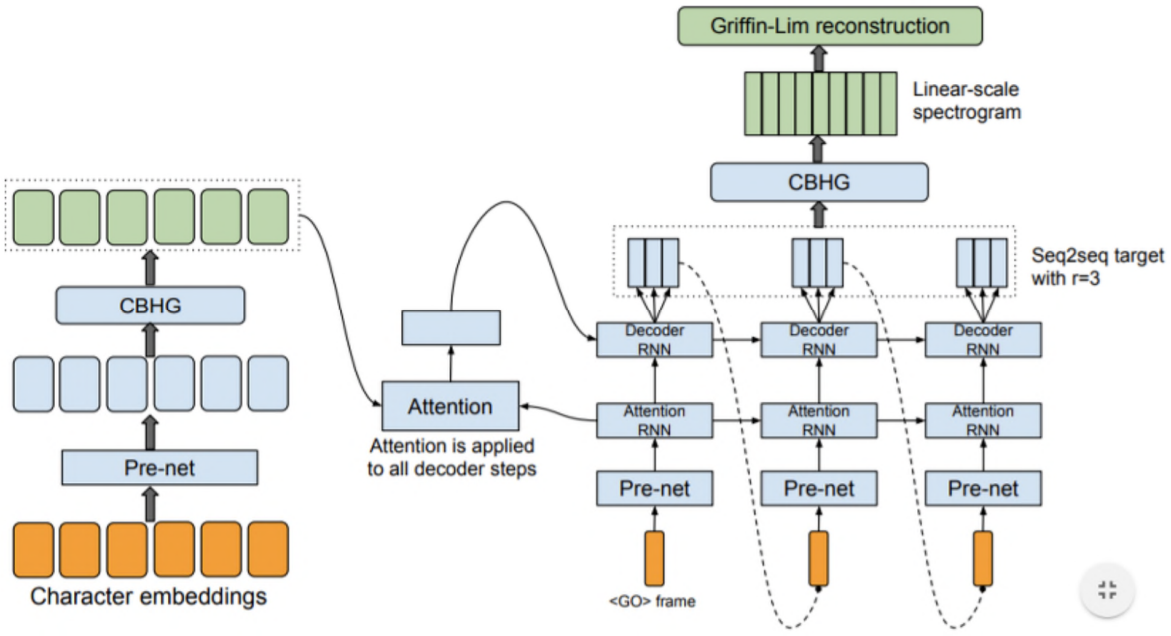
팀프로젝트 : https://github.com/minjae5/graduate_project.git
이영수 : whvh3056
오민재 : minjae5
윤영범 : 0beom

Ⅱ. 본론

1. 개발 내용

1.1 환경요소 Google Speech API 사용법 파악 및 응용 ·····진행
1.2 환경요소 Python 언어 공부 ·····진행
1.3 TensorFlow 사용법 공부 ·····진행
1.4 FFmepg를 통한 동영상 음성데이터 추출 시험 ·····진행
1.5 AWS 서버 구축 및 연결 ·····진행
1.6 Tacotron 모델 공부 ·····진행
1.7 Android Studio를 이용한 사용자 음성추출 어플 제작 ·····대기
1.8 어플과 AWS 서버와의 통신 ·····대기
1.9 AWS 서버 위 Tensorflow 환경구축과 러닝 실행 ·····불가능
1.10 Tacotron 러닝 학습 ·····진행
1.11 러닝을 통한 엔진 제작 ·····대기
1.12 엔진을 사용자에게 전송 ·····대기

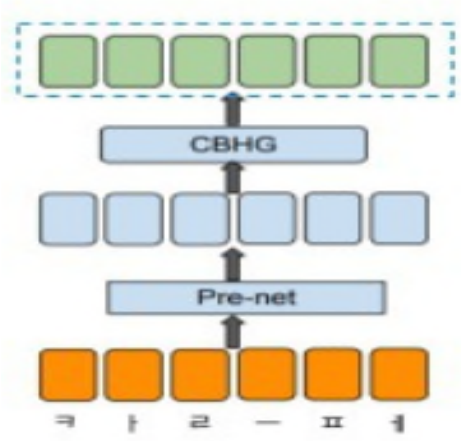
Tacotron 모델



Tacotron 기법 그림1[6]

Text-> Waveform 으로 변환시키는 것이 주 목적이다. 4가지 요소로 구성되어 있다.
문자열을 입력으로, 음성의 스펙트럼 특징벡터 열을 출력으로 지정하여 문자열이 스펙트럼으로 변환되는 중간 과정을 자동으로 학습한다. 이때 입력과 출력의 길이 차이가 발생하여, 어텐션이라는 신경망 층을 도입해 입력과 출력 사이의 매핑 관계를 학습한다.
학습 문장에 비해 입력 문장이 긴 경우 오차가 발생한다. 문장에 포함되지 않은 새로운 어휘, 단어 입력 시 품질이 왜곡된다.

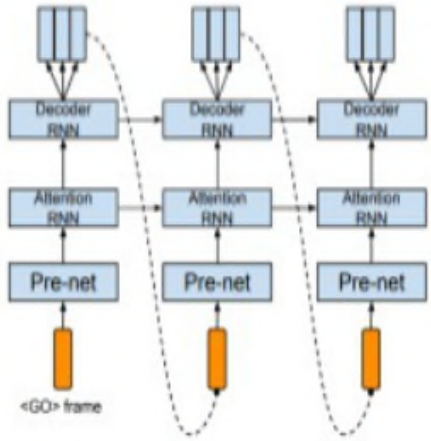
1) Encoder



Encoder 기법 그림2[6]

input(text) →
Pre-net(FC(Dense) - ReLU - Dropout - FC - ReLU - Dropout)
→ CBHG Network
텍스트 -> Encoder -> 텍스트 정보를 잘 나타내는 숫자

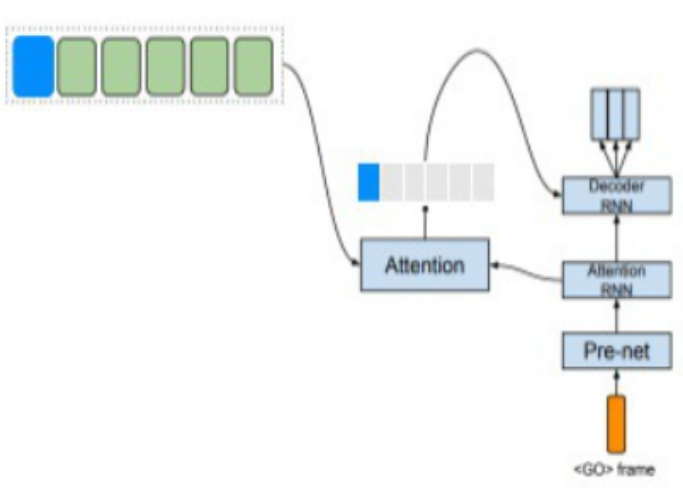
2) Decoder



Decoder 기법 그림3[6]

스펙트로그램 -> Decoder-> 스펙트로그램
스펙트로그램 : 음성을 숫자로 표현하는 여러 가지 방법 중 하나로, RNN 방식의 반복으로 n 개의 스펙트로그램 리스트 획득

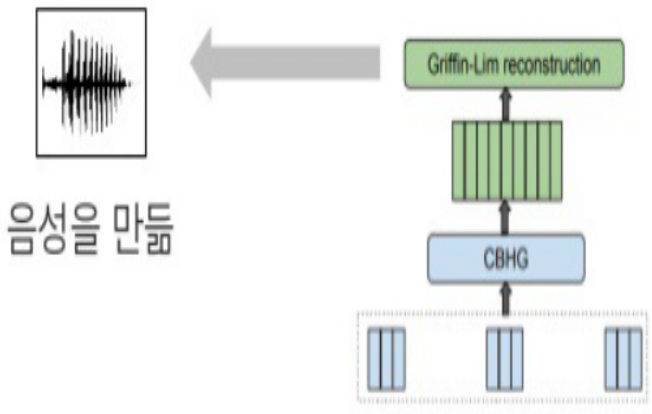
3) Attention



Attention 기법 그림4[6]

인코더의 출력으로, 텍스트 임베딩한 결과물이다.
텍스트 임베딩과 디코더를 잘 합쳐주는 모델이 어텐셜이다.
음성이 들어오면 첫 번째 단어에 집중하고 후에 집중할 위치를 계산하여 러닝한다.

4) Vocoder



Vovoder 기법 그림5[6]

스펙트로그램을 입력으로 받아서 음성으로 변환한다.
스펙트로그램 -> CBHG->그리핀 림 알고리즘-[7]>음성

2. 문제 및 해결방안

2.1 Tactron 모델 샘플링 진행 중 불완전한 코드 임의 수정이 필요하다.

->수정 진행 중

2.2 통화데이터의 음질 개선이 필요하다.

->골든 웨이브를 이용한 음질 개선

2.3 윈도우 기반으로 리눅스 환경 진행 힘들다.

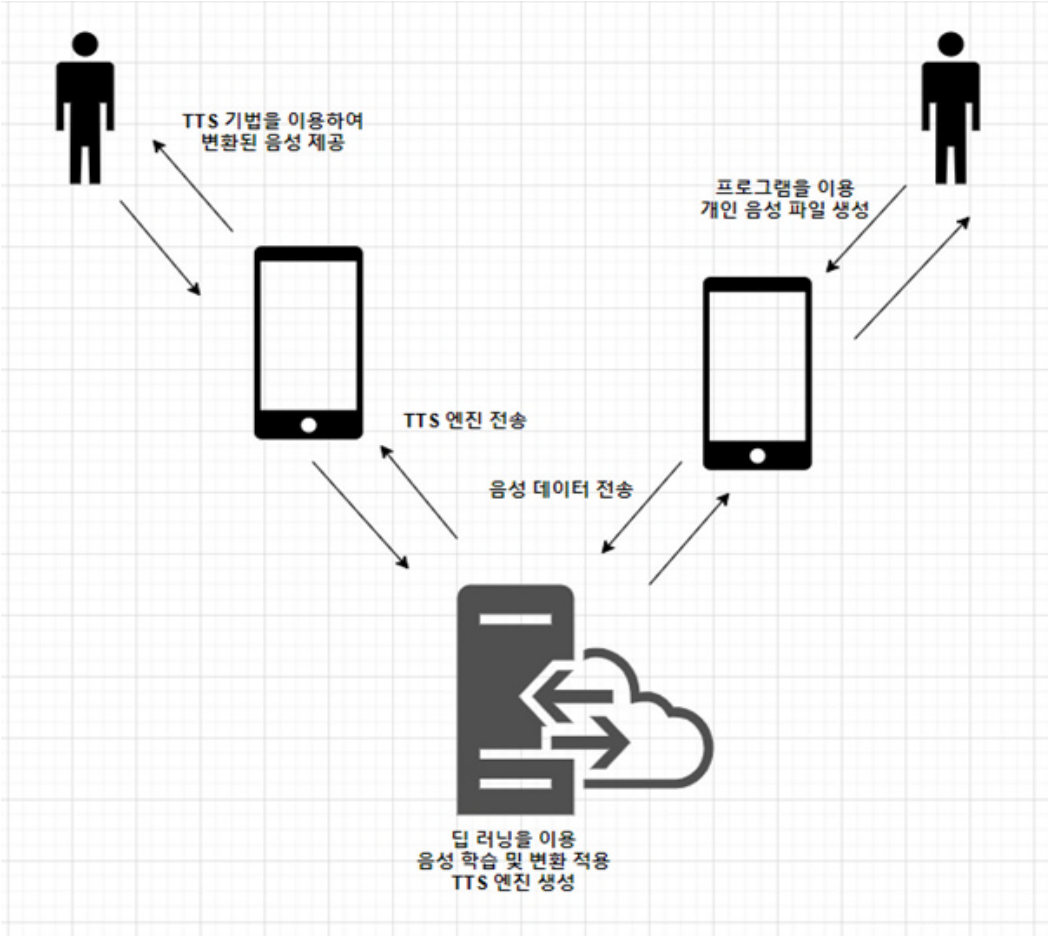
-> 리눅스 운영체제 설치, 멀티 부팅 이용.

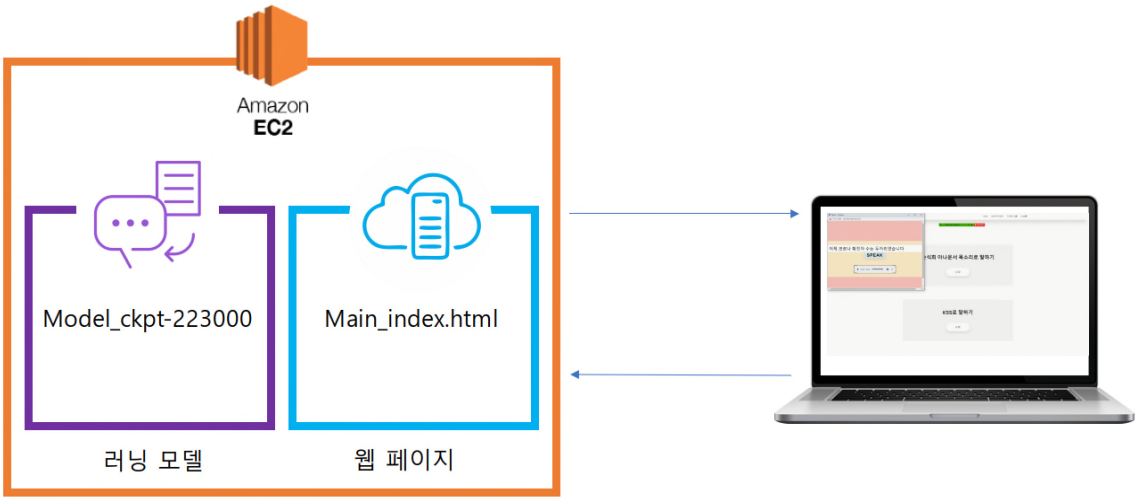
2.4 기본적인 통화 녹음기능 실행 시 상대의 목소리도 같이 저장되는 문제가 있다.

-> 통화 진행 시 추가적인 녹음기능 필요.

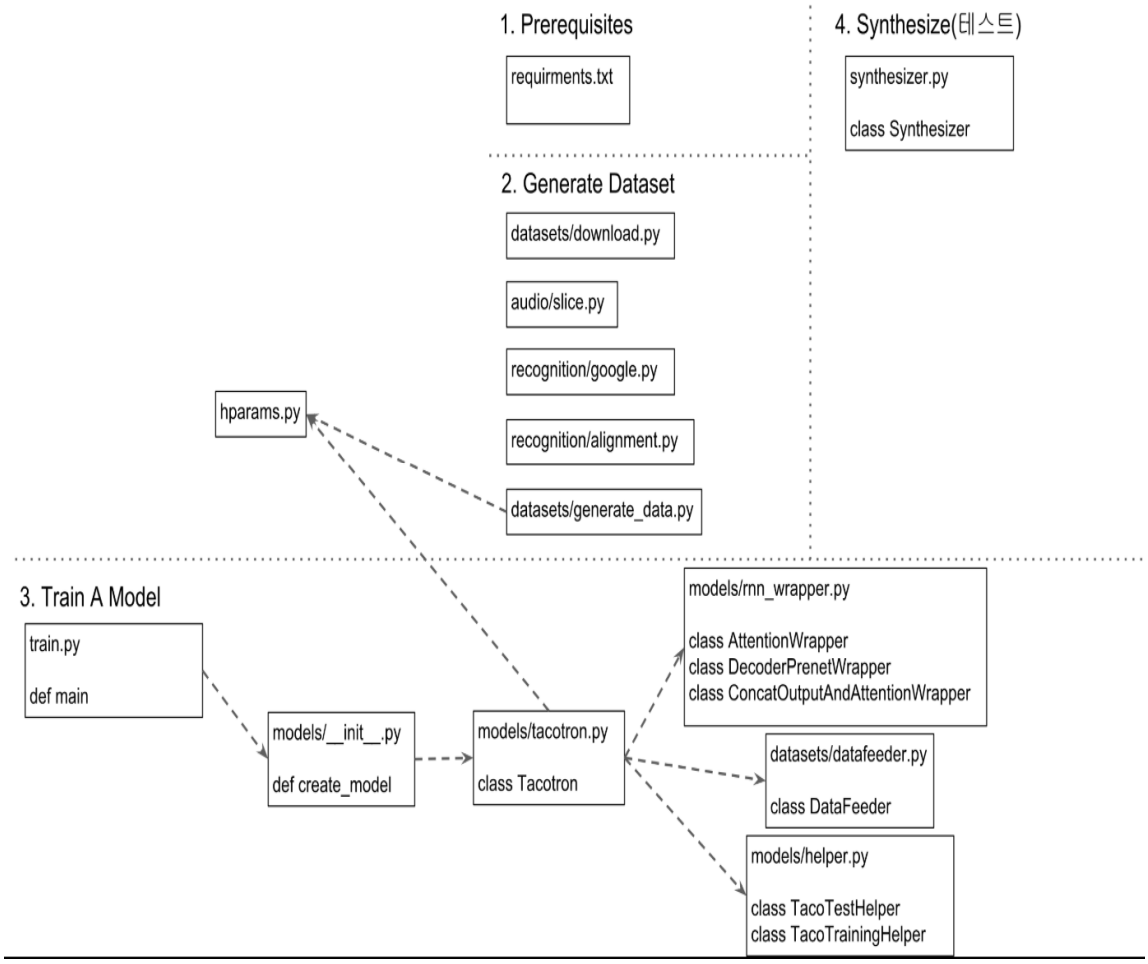
3. 시험 시나리오

클라이언트의 통화 시작 시 모바일 TTS 어플리케이션이 실행되어 통화 음성 녹음 이때 음성은 상대방의 음성을 제외하고 클라이언트의 음성만 녹음 되어야한다.
녹음된 음성 데이터 파일을 서버로 전송시킨 뒤 해당 서버에서 음성데이터 기반 러닝. 러닝을 통해 TTS 엔진 생성이 되어 완성된 엔진을 클라이언트에게 전송.
-> 모바일 어플리케이션에서 웹페이지로 변경





4. 상세설계



타코트론 기반 TTS 모델 그림 7

데이터셋 생성

1. audio.silecce -audio_pattern “음성파일명”으로 음성간 묵을을 기준으로 파일을 분할
2. Recognition.google -audio_pattern “음성파일명”을 이용하여 음성파일에 대한 txt 추출
3. Recogniton.alignment -recognition_path “대본 파일 명“ -score_threshold= “스코어 퍼센트(소수점)" 을 이용하여 실제 대본과 구글 추출 텍스트를 비교 일정 스코어 이상의 내용을 선별하여 json파일로 만들어 낸다.
4. Datasets/generate_data를 이용하여 json 파일을 이용하여 러닝 데이터 셋을 만들어 낸다.

러닝

1. Hparams파일을 이용하여 모델 스타일, 임베딩 크기, 인코더, 디코더, 어텐셜와 같은 파라미터를 학습할 데이터에 따라 조정 가능하다.
2. Train.py 를 통해 학습을 수행한다.
3. Tensorboard의 로그 분석을 통해 러닝을 상태를 확인 할 수있다.

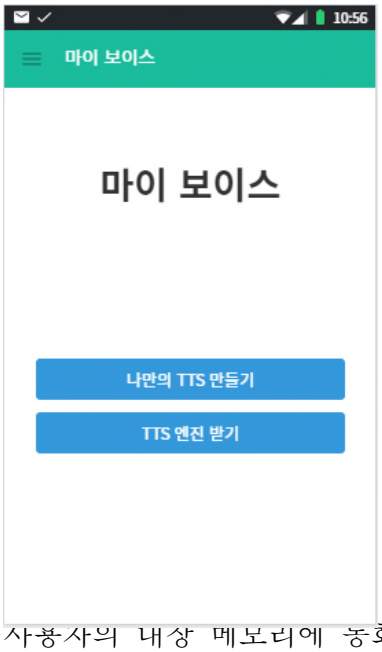
음성 합성

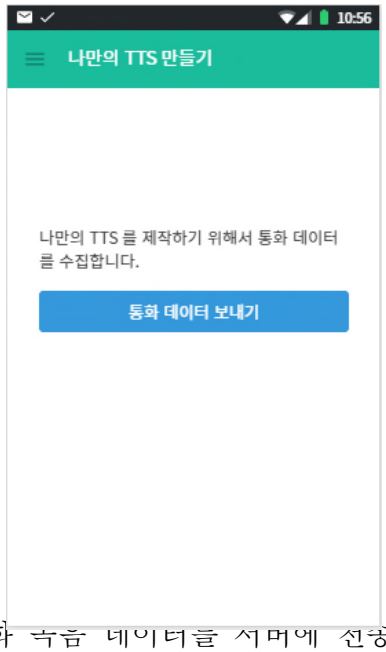
1. Synthesizer -load_path “러닝으로 생성된 로그파일 명“ -text “음성으로 변환할 텍스트" 로 음성을 만들어 낸다.

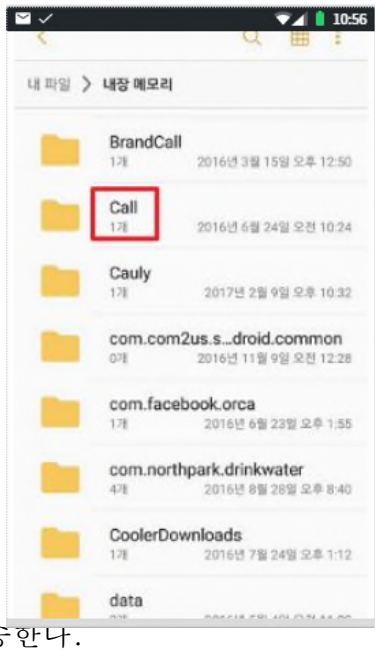
다중 화자 러닝

1. Hparams의 model_type을 single에서 deep voice 로 바꾸어 주어 다중화자 학습으로 바꾸어준다.
2. Train -data_path “학습할 데이터셋”, “기존에 학습된 데이터셋”을 이용하여 적은 양의 데이터 셋으로도 single학습보다 좋은 결과를 가져올 수 있다.

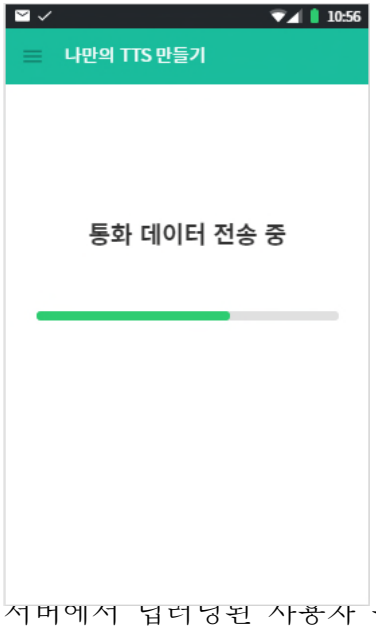
5. 프로토타입 구현







사용자의 내장 메모리에 통화 녹음 데이터를 서버에 전송한다.

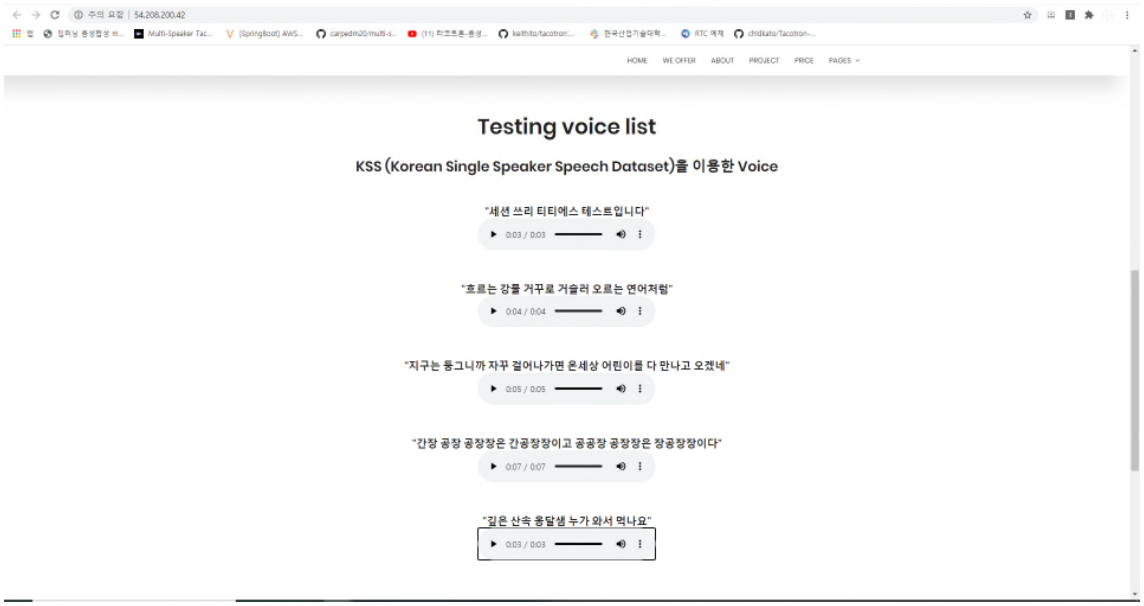


서버에서 업로딩된 사용자 음성의 TTS 엔진 데이터를 수신한다.
사용자는 수신이 완료되면 기기의 설정에서 TTS 종류를 선택해 변경한다.

6. 테스트 / 시험

http://54.208.200.42/

(개인 aws서버라 불가피하게 접속이 불가할 수도 있다.)



클라이언트가 접할 페이지(뷰)로 기존 학습 완료된 목소리를 이용한 샘플 텍스트를 청취할 수 있는 페이지를 확인할 수 있다.
Aws ec2생성한 뒤 apache로 서버 연결을 하였다.
소스는 Filezila로 업로드하였다.

7. Coding & Demo

코드는 크게 전처리(preprocess), 훈련(train), 실행, 합성(eval) 세부분으로 나누었다.

<preeprocess.py>

```
import numpy as np
import argparse, os, re
from concurrent.futures import ProcessPoolExecutor
from multiprocessing import cpu_count
from tqdm import tqdm
from hparams import hparams
from util import audio


text_name = 'transcript.v.1.2.txt'
filters = "([.,!?])"


def preprocess_kss(args):
  in_dir = os.path.join(args.base_dir, 'kss')
  out_dir = os.path.join(args.base_dir, args.output)
  os.makedirs(out_dir, exist_ok=True)
  metadata = build_from_path(in_dir, out_dir, args.num_workers, tqdm=tqdm)
  write_metadata(metadata, out_dir)


def build_from_path(in_dir, out_dir, num_workers=1, tqdm=lambda x: x):
  executor = ProcessPoolExecutor(max_workers=num_workers)
```

```
  futures = []
  index = 1
  with open(os.path.join(in_dir, text_name), encoding='utf-8') as f:
    for line in f:
      parts = line.strip().split('|')
      wav_path = os.path.join(in_dir, parts[0])
      text = parts[3]
      text = re.sub(re.compile(filters), '', text)
      futures.append(executor.submit(_process_utterance, out_dir, index, wav_path, text))
      index += 1
  return [future.result() for future in tqdm(futures)]


def _process_utterance(out_dir, index, wav_path, text):
  wav, _ = audio.load_wav(wav_path)

  spectrogram = audio.spectrogram(wav).astype(np.float32)  # (1025, frame)
  n_frames = spectrogram.shape[1]

  mel_spectrogram = audio.melspectrogram(wav).astype(np.float32)  # (80, frame)

  spectrogram_filename = 'kss-spec-%05d.npy' % index
  mel_filename = 'kss-mel-%05d.npy' % index
  np.save(os.path.join(out_dir, spectrogram_filename), spectrogram.T, allow_pickle=False)  # (frame, 1025)
  np.save(os.path.join(out_dir, mel_filename), mel_spectrogram.T, allow_pickle=False)  # (frame, 80)

  return (spectrogram_filename, mel_filename, n_frames, text)


def write_metadata(metadata, out_dir):
  with open(os.path.join(out_dir, 'train.txt'), 'w', encoding='utf-8') as f:
    for m in metadata:
      f.write('|'.join([str(x) for x in m]) + '₩n')
  frames = sum([m[2] for m in metadata])
  hours = frames * hparams.frame_shift_ms / (3600 * 1000)
  print('₩rote %d utterances, %d frames (%.2f hours)' % (len(metadata), frames, hours))


def main():
  parser = argparse.ArgumentParser()
  parser.add_argument('--base_dir', default='./')
  parser.add_argument('--output', default='training')
  parser.add_argument('--num_workers', type=int, default=cpu_count())
  args = parser.parse_args()
  preprocess_kss(args)


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

전처리 파일로 학습에 사용할 데이터를 만들기 위해 preprocess를 실행한다. 대본 파일의 경로와 대본 txt 파일명을 기재하여 경로 설정을 완료하고 데이터 생성 프로세스를 시작한다. 실행하게 되면, training폴더에 데이터와 train.txt가 생성된

다.

<train.py>

```
import tensorflow as tf
import argparse, math, os, time, traceback
from hparams import hparams
from models import create_model
from models.datafeeder import DataFeeder
from util import audio, infolog, plot, ValueWindow
from util.text import sequence_to_text


log = infolog.log


def add_stats(model):
  with tf.variable_scope('stats') as scope:
    tf.summary.histogram('linear_outputs', model.linear_outputs)
    tf.summary.histogram('linear_targets', model.linear_targets)
    tf.summary.histogram('mel_outputs', model.mel_outputs)
    tf.summary.histogram('mel_targets', model.mel_targets)
    tf.summary.scalar('loss_mel', model.mel_loss)
    tf.summary.scalar('loss_linear', model.linear_loss)
    tf.summary.scalar('learning_rate', model.learning_rate)
    tf.summary.scalar('loss', model.loss)
    gradient_norms = [tf.norm(grad) for grad in model.gradients]
    tf.summary.histogram('gradient_norm', gradient_norms)
    tf.summary.scalar('max_gradient_norm', tf.reduce_max(gradient_norms))
    return tf.summary.merge_all()


def train(log_dir, args):
  checkpoint_path = os.path.join(log_dir, 'model.ckpt')
  input_path = os.path.join(args.base_dir, args.input)

  coord = tf.train.Coordinator()
  with tf.variable_scope('datafeeder') as scope:
    feeder = DataFeeder(coord, input_path, hparams)

  global_step = tf.Variable(0, name='global_step', trainable=False)
  with tf.variable_scope('model') as scope:
    model = create_model(args.model, hparams)
    model.initialize(feeder.inputs, feeder.input_lengths, feeder.mel_targets, feeder.linear_targets)
    model.add_loss()
    model.add_optimizer(global_step)
    stats = add_stats(model)

  # Bookkeeping
  time_window = ValueWindow(100)
  loss_window = ValueWindow(100)
  saver = tf.train.Saver(max_to_keep=5, keep_checkpoint_every_n_hours=2)

  # Train
  with tf.Session() as sess:
```

```
    try:
      summary_writer = tf.summary.FileWriter(log_dir, sess.graph)
      sess.run(tf.global_variables_initializer())

      if args.restore_step:
        restore_path = '%s-%d' % (checkpoint_path, args.restore_step)
        saver.restore(sess, restore_path)
        log('Resuming from checkpoint: %s' % (restore_path), slack=True)

      feeder.start_in_session(sess)

      while not coord.should_stop():
        start_time = time.time()
        step, loss, opt, mel_loss, linear_loss = ₩
          sess.run([global_step, model.loss, model.optimize, model.mel_loss, model.linear_loss])
        time_window.append(time.time() - start_time)
        loss_window.append(loss)
        message = 'Step %-7d [%.03f sec/step, loss=%.05f, avg_loss=%.05f, mel_loss=%.5f,
linear_loss=%.5f]' % (
          step, time_window.average, loss, loss_window.average, mel_loss, linear_loss)
        log(message, slack=(step % args.checkpoint_interval == 0))

        if step % args.summary_interval == 0:
          log('Writing summary at step: %d' % step)
          summary_writer.add_summary(sess.run(stats), step)

        if step % args.checkpoint_interval == 0:
          log('Saving checkpoint to: %s-%d' % (checkpoint_path, step))
          saver.save(sess, checkpoint_path, global_step=step)
          log('Saving audio and alignment...')
          input_seq, spectrogram, alignment = sess.run([
            model.inputs[0], model.linear_outputs[0], model.alignments[0]])
          waveform = audio.inv_spectrogram(spectrogram.T)
          audio.save_wav(waveform, os.path.join(log_dir, 'step-%d-audio.wav' % step))
          input_seq = sequence_to_text(input_seq)
          plot.plot_alignment(alignment, os.path.join(log_dir, 'step-%d-align.png' % step), input_seq,
            info='%s, step=%d, loss=%.5f' % (args.model, step, loss), istrain=1)
          log('Input: %s' % input_seq)

    except Exception as e:
      log('Exiting due to exception: %s' % e, slack=True)
      traceback.print_exc()
      coord.request_stop(e)


def main():
  parser = argparse.ArgumentParser()
  parser.add_argument('--base_dir', default='./')
  parser.add_argument('--input', default='training/train.txt')
  parser.add_argument('--model', default='tacotron')
  parser.add_argument('--restore_step', type=int)
  parser.add_argument('--summary_interval', type=int, default=100)
  parser.add_argument('--checkpoint_interval', type=int, default=1000)
  args = parser.parse_args()
  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

```
  run_name = args.model
  log_dir = os.path.join(args.base_dir, 'logs-%s' % run_name)
  os.makedirs(log_dir, exist_ok=True)
  infolog.init(os.path.join(log_dir, 'train.log'), run_name)
  train(log_dir, args)


if __name__ == '__main__':
  main()
```

학습단계의 파일로, 1000step 마다 학습결과에 해당하는 train, test 샘플 wav파일 그래프 파일인 png 확장 파일이 생성된다. 또한 실제 음성합성에 이용할 수 있는 chkpt파일이 생성된다. 또한 학습 진행과정에서 학습 상태는 tesorboard로도 확인이 가능하다. 팀에서 학습을 해본 결과 Step이 100,000 정도가 넘어가야 실제 합성을 해 볼 수 있다는걸 확인할 수 있었다.

<eval.py>

```
import numpy as np
import tensorflow as tf
import os, re, io, argparse
from jamo import hangul_to_jamo
from hparams import hparams
from librosa import effects
from models import create_model
from util.text import text_to_sequence, sequence_to_text
from util import audio, plot


class Synthesizer:
  def load(self, checkpoint_path, model_name='tacotron'):
    print('Constructing model: %s' % model_name)
    inputs = tf.placeholder(tf.int32, [1, None], 'inputs')
    input_lengths = tf.placeholder(tf.int32, [1], 'input_lengths')
    with tf.variable_scope('model') as scope:
      self.model = create_model(model_name, hparams)
      self.model.initialize(inputs, input_lengths)
      self.wav_output = audio.inv_spectrogram_tensorflow(self.model.linear_outputs[0])
      self.alignments = self.model.alignments[0]
      self.inputs = self.model.inputs[0]

    print('Loading checkpoint: %s' % checkpoint_path)
    self.session = tf.Session()
    self.session.run(tf.global_variables_initializer())
    saver = tf.train.Saver()
    saver.restore(self.session, checkpoint_path)


  def synthesize(self, text, base_path, idx):
    seq = text_to_sequence(text)
    feed_dict = {
      self.model.inputs: [np.asarray(seq, dtype=np.int32)],
      self.model.input_lengths: np.asarray([len(seq)], dtype=np.int32)
```

```
    }
    input_seq, wav, alignment = self.session.run([self.inputs, self.wav_output, self.alignments],
feed_dict=feed_dict)
    wav = audio.inv_preemphasis(wav)
    wav = wav[:audio.find_endpoint(wav)]
    out = io.BytesIO()
    audio.save_wav(wav, out)
    input_seq = sequence_to_text(input_seq)
    plot.plot_alignment(alignment, '%s-%d-align.png' % (base_path, idx), input_seq)
    return out.getvalue()


def get_output_base_path(checkpoint_path):
  base_dir = os.path.dirname(checkpoint_path)
  m = re.compile(r'.*?₩.ckpt₩-([0-9]+)').match(checkpoint_path)
  name = 'eval-%d' % int(m.group(1)) if m else 'eval'
  return os.path.join(base_dir, name)


def run_eval(args):
  synth = Synthesizer()
  synth.load(args.checkpoint)
  base_path = get_output_base_path(args.checkpoint)
  for i, text in enumerate(sentences):
    jamo = ''.join(list(hangul_to_jamo(text)))
    path = '%s-%d.wav' % (base_path, i)
    print('Synthesizing: %s' % path)
    with open(path, 'wb') as f:
      f.write(synth.synthesize(jamo, base_path, i))


def main():
  parser = argparse.ArgumentParser()
  parser.add_argument('--checkpoint', required=True)
  args = parser.parse_args()
  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
  run_eval(args)


if __name__ == '__main__':
  main()
```

train으로 만들어진 ckpt파일을 이용해서 모델 설정을 하고 음성파일을 만들어내는 부분이다.

Ⅲ. 결론

1. 연구 결과

KSS 모델의 경우, 음성 데이터의 양은 4GB로 10만 스텝 정도의 러닝을, 손석희 아나운서 모델의 경우 40GB의 음성데이터로 30만 스텝 정도의 러닝을 진행하였다. 데이터 양으로 보았을 때 손석희 아나운서 모델이 좀 더 자연스러울 것으로 예상했으나 결과적으로는 KSS모델이 좀 더 자연스럽고 깔끔한 음성을 청취할 수 있었다.

KSS의 경우 음질이 깔끔히 선별된 음성이기 때문에 spectrogram구축에 문제없이 학습이 잘 된 것으로 알 수 있었다. 그러나, 손석희 아나운서 목소리의 경우에는 뉴스룸 프로그램에서 뉴스 음성 자체를 따온 거싱기에 주변의 잡음이나 노랫소리가 같이 존재 하였기에 학습 모델링 구축이 잘못된 방향으로 이루어 졌을 것으로 해석할 수 있다. 또한 RNN 모델의 특성상 긴 문장에 대해서는 모델 형성에 어려움이 있기 때문에 길이가 긴 뉴스 대본이 영향을 미쳤음을 알 수 있었다. 본연구는 결과적으로 개인의 목소리의 경우도 많은 양으로 모델링을 하는 것이 아닌 모델링을 명확하게 구축할 수 있는 방향으로 나아간다면 적은 데이터 양으로도 좋은 모델을 만들 수 있음을 보여준다. 음성데이터를 이용하여 TTS를 만들기 위해서는 많은 양의 데이터보다 음질이 좋은 데이터가 더 효율적이라는 것을 알 수 있었다. 그러나, 본 연구의 두 모델의 경우 일반인의 목소리가 아닌 뉴스 아나운서와 처음부터 정제된 음성데이터였다. 일반인의 목소리를 생각해보면 4GB 정도의 데이터를 모으는 것이 쉽지 않을 것이다. 통화데이터를 오랫동안 쌓으면 가능할지라도 상대의 목소리가 같이 녹음되는 점과 소리의 음질을 생각했을 때 원하는결과가 나오지 않을 확률이 높다. 이를 보완하기 위해 기존에 잘 만들어진 모델의 spectrogram의 데이터를 새로 만들어질 모델의 Decoder 러닝에 적용시켜 적은 데이터 양과 시간으로도 좋은 결과를 낼 수 있어야 일반 개인의 목소리도 TTS 모델로 만들 수 있을 것이라 위 프로젝트를 진행하며 알 수 있었다.

2. 작품제작 소요 목록

Amazon Web Service
Amazon Machine Image - Deep Learning AMI(Ubuntu 16.04) Version 37.0
Instance - g3s.xlarge
SpotInstance 서버 대여 2020.06.01 ~ 2020.12.04

[참고 자료]

[1] Google TTS: https://cloud.google.com/text-to-speech/?hl=ko
[2] Samsung TTS; https://www.sammobile.com/apk/samsung-text-to-speech-engine
[3] I-phone TTS; https://www.pcmag.com/how-to/how-to-use-the-iphone-text-to-speech-feature
[4] Voiceware; http://www.voiceware.co.kr/kor/product/product1.php
[5] NaverClover; https://clova.ai/voice
[6] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, Rif A. Saurous
Tacotron: Towards End-to-End Speech Synthesis, Interspeech 2017, 6 April 2017; https://arxiv.org/abs/1703.10135v2
[7] Griffin-lim-algorithm; https://paperswithcode.com/method/griffin-lim-algorithm