임베디드 소프트웨어: 음성인식 학습 및 모델 평가 과제

Park Jonghyeon
컴퓨터정보통신공학과
전남대학교 공과대학
jonghyeon@jnu.ac.kr

***Abstract*—텐서플로우 튜토리얼 문서의 <간단한 오디오 인식: 키워드 인식>을 이용하여 오디오 인식 모델을 학습하고, 실제 목소리를 사용하여 성능을 평가하였습니다.**

I. 모델 정의 및 학습

실험은 튜토리얼 문서 상의 모델을 그대로 사용하였으므로, 구조가 변화하지 않았습니다.

Model: "sequential"

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 32, 32, 1)	0
normalization (Normalization)	(None, 32, 32, 1)	3
conv2d (Conv2D)	(None, 30, 30, 32)	320
conv2d_1 (Conv2D)	(None, 28, 28, 64)	18,496
max_pooling2d (MaxPooling2D)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 128)	1,605,760
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 8)	1,032

그 대신 에폭을 10에서 100으로 변경하였습니다. 이로 인해 발생하는 학습 시간의 증가는 배치 사이즈를 64에서 128로 조정하고, 튜토리얼 문서의 <미니 음성 명령 데이터 세트 가져오기> 섹션의 작은 규모의 음성 명령 데이터 세트를 학습에 사용하여 대응하였습니다.

II. 성능 실험

이렇게 학습시킨 모델의 성능 실험은 제출자 본인의 목소리를 사용하여 평가하는 것으로 수행하였습니다. 제출자는 기말 팀 프로젝트를 위해 마이크 입력 코드를 미리 준비하고 있었으므로, 편의를 위해 이를 사용하여 실험을 수행하였습니다.

실험은 실시간으로 `no`를 녹음하여 모델에 그 녹음 데이터를 분류하도록 하였습니다. 모델이 녹음을 마치면 평가한 데이터를 개별적으로 저장합니다. 이러한 과정을 30회 반복하였습니다.

III. 실험 결과

모델은 음성이 전반적으로 `no`보다는 `down`과 `yes`에 가깝다고 평가하고 있습니다. 5번 시도와 같이 `no`를 정확하게 추정한 경우도 있으나, 대부분의 경우 `down`, `yes`과 비슷

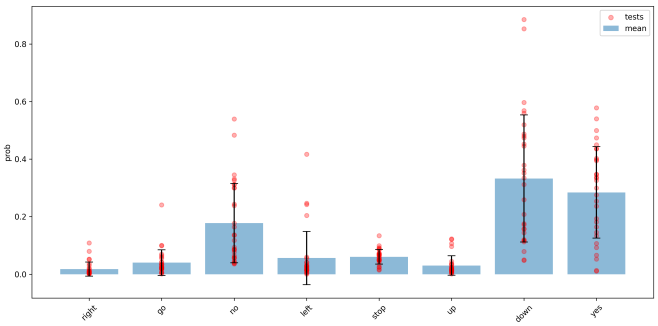

Fig. 1: 30회의 시도 결과

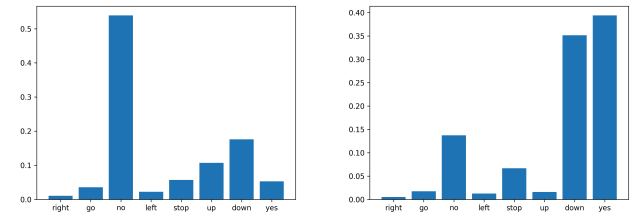

Fig. 2: 시도 #5의 추정 결과(좌) 및 시도 #21의 추정 결과(우)

한 확률이거나 이 두 단어일 확률이 더 높다고 평가하고 있습니다.

이러한 낮은 성능을 보이는 이유 중 하나로, 학습에 사용한 음성 데이터는 원어민의 음성 데이터를 그대로 사용했기 때문으로 추정됩니다.

제출자의 영어 발음법은 영어 원어민 수준을 모사할 수 없으므로, 원어민과 발음에 있어 큰 차이를 보입니다. 사람이 듣기에 같은 형식의 발음이라도 디지털 데이터로 인코딩하고 양자화하는 과정에서 전혀 다른 데이터로 저장되었을 수 있습니다.

유효한 방법인지는 검증되지 않았으나, 학습 과정에서 원어민의 음성 데이터에 노이즈를 넣는 시도로 개선을 시도할 수 있을 것 같습니다.

IV. 부록

A. 실험 결과

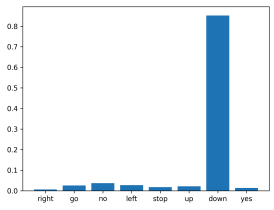

Fig. 3: 시도 #1의 추정 결과

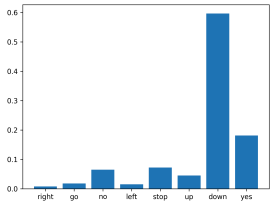

Fig. 4: 시도 #2의 추정 결과

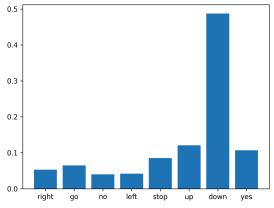

Fig. 5: 시도 #3의 추정 결과

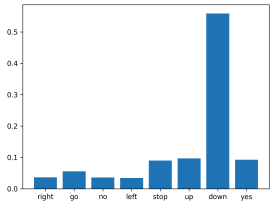

Fig. 6: 시도 #4의 추정 결과

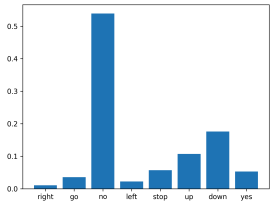

Fig. 7: 시도 #5의 추정 결과

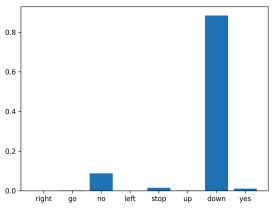

Fig. 8: 시도 #6의 추정 결과

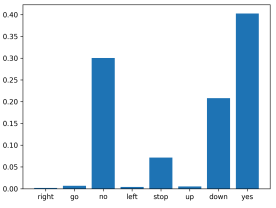

Fig. 9: 시도 #7의 추정 결과

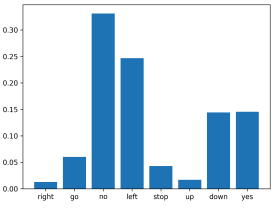

Fig. 10: 시도 #8의 추정 결과

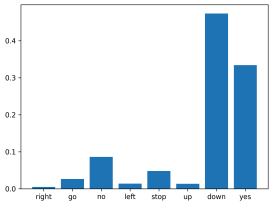

Fig. 11: 시도 #9의 추정 결과

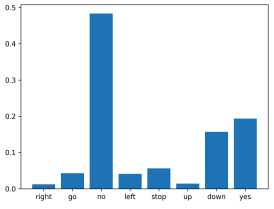

Fig. 12: 시도 #10의 추정 결과

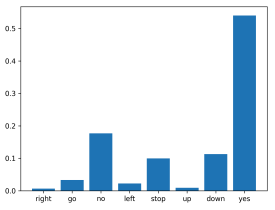

Fig. 13: 시도 #11의 추정 결과

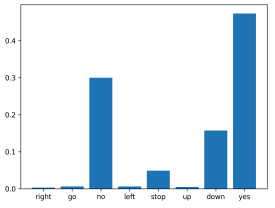

Fig. 14: 시도 #12의 추정 결과

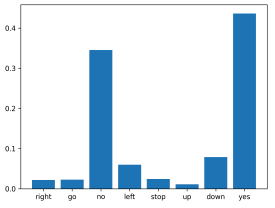

Fig. 15: 시도 #13의 추정 결과

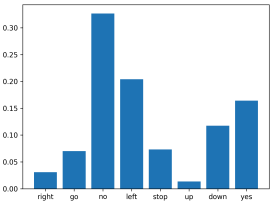

Fig. 16: 시도 #14의 추정 결과

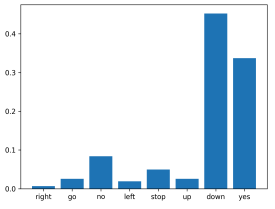

Fig. 17: 시도 #15의 추정 결과

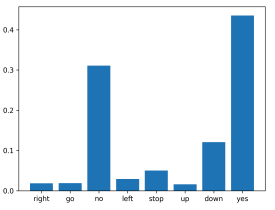

Fig. 18: 시도 #16의 추정 결과

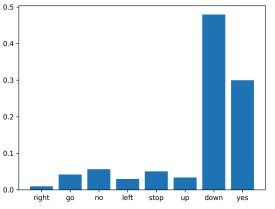

Fig. 19: 시도 #17의 추정 결과

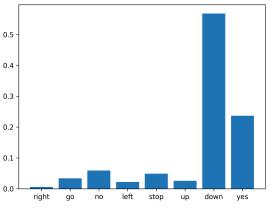

Fig. 20: 시도 #18의 추정 결과

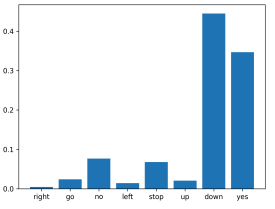

Fig. 21: 시도 #19의 추정 결과

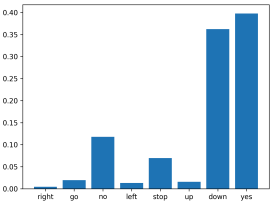

Fig. 22: 시도 #20의 추정 결과

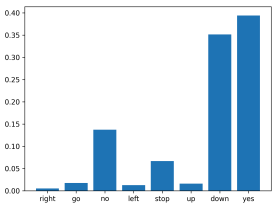

Fig. 23: 시도 #21의 추정 결과

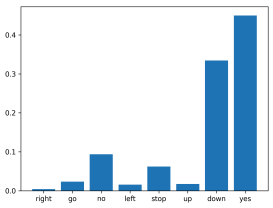

Fig. 28: 시도 #26의 추정 결과

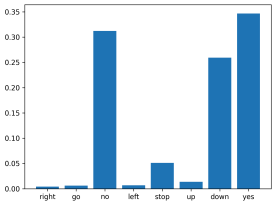

Fig. 24: 시도 #22의 추정 결과

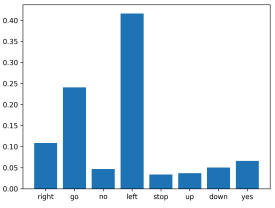

Fig. 29: 시도 #27의 추정 결과

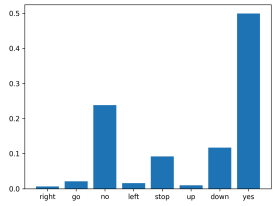

Fig. 25: 시도 #23의 추정 결과

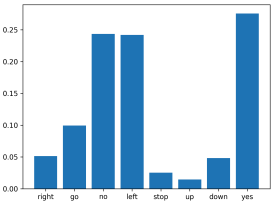

Fig. 30: 시도 #28의 추정 결과

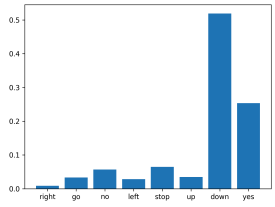

Fig. 26: 시도 #24의 추정 결과

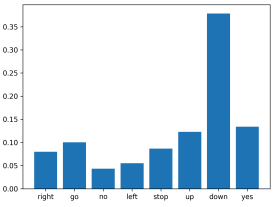

Fig. 31: 시도 #29의 추정 결과

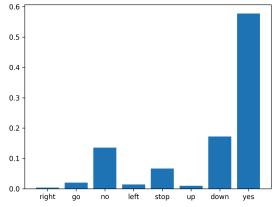

Fig. 27: 시도 #25의 추정 결과

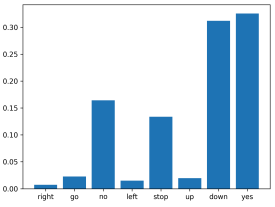

Fig. 32: 시도 #30의 추정 결과

B. 소스코드

a) *train.py*:

```
import os
import pathlib
from shutil import move

import numpy as np
import seaborn as sns
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import models

DATASET_PATH = 'data/mini_speech_commands'
class Config:
   seed = int(42)
   batch_size = 128
   validation_split = .25
   epochs = 100


tf.random.set_seed(Config.seed)
np.random.seed(Config.seed)

data_dir = pathlib.Path(DATASET_PATH)
if not data_dir.exists():
   tf.keras.utils.get_file(
      'mini_speech_commands.zip',
      origin="http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_
commands.zip",
      extract=True,
      cache_dir='.', cache_subdir='data')
            move('data/mini_speech_commands_extracted/mini_speech_commands',   'data/
mini_speech_commands')


commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = commands[commands != 'README.md']


train_ds, val_ds = tf.keras.utils.audio_dataset_from_directory(
   directory=data_dir,
   batch_size=Config.batch_size,
   validation_split=Config.validation_split,
   seed=Config.seed,
   output_sequence_length=16000,
   subset='both')

label_names = np.array(train_ds.class_names)


def squeeze(audio, labels):
   audio = tf.squeeze(audio, axis=-1)
   return audio, labels

train_ds = train_ds.map(squeeze, tf.data.AUTOTUNE)
val_ds = val_ds.map(squeeze, tf.data.AUTOTUNE)


test_ds = val_ds.shard(num_shards=2, index=0)
val_ds = val_ds.shard(num_shards=2, index=1)
```

```
def get_spectrogram(waveform):
  # Convert the waveform to a spectrogram via a STFT.
  spectrogram = tf.signal.stft(
    waveform, frame_length=255, frame_step=128)
  # Obtain the magnitude of the STFT.
  spectrogram = tf.abs(spectrogram)
  # Add a `channels` dimension, so that the spectrogram can be used
  # as image-like input data with convolution layers (which expect
  # shape (`batch_size`, `height`, `width`, `channels`).
  spectrogram = spectrogram[..., tf.newaxis]
  return spectrogram


def make_spec_ds(ds):
  return ds.map(
    map_func=lambda audio,label: (get_spectrogram(audio), label),
    num_parallel_calls=tf.data.AUTOTUNE)


train_spectrogram_ds = make_spec_ds(train_ds)
val_spectrogram_ds = make_spec_ds(val_ds)
test_spectrogram_ds = make_spec_ds(test_ds)


train_spectrogram_ds
= train_spectrogram_ds.cache().shuffle(10000).prefetch(tf.data.AUTOTUNE)
val_spectrogram_ds = val_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
test_spectrogram_ds = test_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)

for sample_spectograms, sample_spectogram_labels in train_spectrogram_ds.take(1):
  break
input_shape = sample_spectograms.shape[1:]
num_labels = len(commands)



class ExportModel(tf.Module):
  def __init__(self, model):
    self.model = model

    # Accept either a string-filename or a batch of waveforms.
    # YOu could add additional signatures for a single wave, or a ragged-batch.
    self.__call__.get_concrete_function(
      x=tf.TensorSpec(shape=(), dtype=tf.string))
    self.__call__.get_concrete_function(
    x=tf.TensorSpec(shape=[None, 16000], dtype=tf.float32))


  @tf.function
  def __call__(self, x):
    # If they pass a string, load the file and decode it.
    if x.dtype == tf.string:
      x = tf.io.read_file(x)
      x, _ = tf.audio.decode_wav(x, desired_channels=1, desired_samples=16000,)
      x = tf.squeeze(x, axis=-1)
      x = x[tf.newaxis, :]

    x = get_spectrogram(x)
    result = self.model(x, training=False)

    class_ids = tf.argmax(result, axis=-1)
    class_names = tf.gather(label_names, class_ids)
    return {
```

```
        'predictions': result,
        'class_ids': class_ids,
        'class_names': class_names
      }



# Instantiate the `tf.keras.layers.Normalization` layer.
norm_layer = layers.Normalization()
# Fit the state of the layer to the spectrograms
# with `Normalization.adapt`.
norm_layer.adapt(data=train_spectrogram_ds.map(map_func=lambda spec, label: spec))

model = models.Sequential([
   layers.Input(shape=input_shape),
   # Downsample the input.
   layers.Resizing(32, 32),
   # Normalize.
   norm_layer,
   layers.Conv2D(32, 3, activation='relu'),
   layers.Conv2D(64, 3, activation='relu'),
   layers.MaxPooling2D(),
   layers.Dropout(0.25),
   layers.Flatten(),
   layers.Dense(128, activation='relu'),
   layers.Dropout(0.5),
   layers.Dense(num_labels),
])

def main():
   model.compile(
      optimizer=tf.keras.optimizers.Adam(),
      loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
      metrics=['accuracy'],
   )


   EPOCHS = Config.epochs
   history = model.fit(
      train_spectrogram_ds,
      validation_data=val_spectrogram_ds,
      epochs=EPOCHS,
      callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=2),
   )

   export = ExportModel(model)
   tf.saved_model.save(export, "saved")

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

b) *run.py*:

```
import pyaudio
from array import array

import numpy as np
from scipy import signal
class AudioHandler:
   def __init__(self):
      self.CHUNK = 1024
      self.FORMAT = pyaudio.paInt16
      self.CHANNELS = 1
      self.RATE = 44100
      self.SILENCE_THRESHOLD = 1000  # Adjust this value based on your mic
```

```
    self.SILENCE_CHUNKS = 30  # How many chunks of silence before stopping

def record_until_silence(self):
    p = pyaudio.PyAudio()
    stream = p.open(format=self.FORMAT,
                    channels=self.CHANNELS,
                    rate=self.RATE,
                    input=True,
                    frames_per_buffer=self.CHUNK)

    print("Listening... Speak now!")

    frames = []
    silent_chunks = 0
    is_speaking = False

    while True:
        data = stream.read(self.CHUNK)
        array_data = array('h', data)
        max_volume = max(abs(vol) for vol in array_data)

        # If sound is above threshold, record
        if max_volume > self.SILENCE_THRESHOLD:
            is_speaking = True
            silent_chunks = 0
            frames.append(data)

        # If we were speaking but now there's silence
        elif is_speaking:
            frames.append(data)
            silent_chunks += 1

            # If enough silence, stop recording
            if silent_chunks > self.SILENCE_CHUNKS:
                break

    print("Done recording")

    stream.stop_stream()
    stream.close()
    p.terminate()

    return frames

def frames_to_waveform(self, frames):
    # Convert frames to numpy array
    audio_data = np.frombuffer(b''.join(frames), dtype=np.int16)

    # Normalize to [-1, 1] float
    audio_float = audio_data.astype(np.float32) / 32767.0

    # Resample from 44100 to 16000
    samples = int(len(audio_float) * 16000 / self.RATE)
    resampled = signal.resample(audio_float, samples)

    # Pad/trim to exactly 16000 samples
    if len(resampled) > 16000:
        resampled = resampled[:16000]
    else:
        resampled = np.pad(resampled, (0, 16000 - len(resampled)))

    return tf.convert_to_tensor(resampled, dtype=tf.float32)
```

```
import tensorflow as tf

def record_and_pred():
   audio_handler = AudioHandler()
   frames = audio_handler.record_until_silence()
   waveform = audio_handler.frames_to_waveform(frames)

   imported = tf.saved_model.load("saved")
   res = imported(waveform[tf.newaxis, :])

   print(res)

   from train import commands
   import matplotlib.pyplot as plt
   plt.bar(commands, tf.nn.softmax(res['predictions'])[0])
   plt.show()

from os import mkdir
from train import commands
import matplotlib.pyplot as plt
import pathlib
def manual_human_testing():
   audio_handler = AudioHandler()
   imported = tf.saved_model.load("saved")
   runout = pathlib.Path('run.out')
   epoch = 30
   if not runout.exists():
      mkdir(runout)
   results = []
   for i in range(1, epoch+1):
      frames = audio_handler.record_until_silence()
      waveform = audio_handler.frames_to_waveform(frames)
      res = imported(waveform[tf.newaxis, :])
      _sftmx = tf.nn.softmax(res['predictions'])[0]
      plt.bar(commands, _sftmx)
      plt.savefig(f'run.out/{i}.png', dpi=300)
      plt.close()
      results.append(_sftmx)
      print('saved')
   results_array = np.array(results)
   means = np.mean(results_array, axis=0)
   stds = np.std(results_array, axis=0)

   plt.figure(figsize=(12, 6))
   x = np.arange(len(commands))

   # Plot bars with error bars
   plt.bar(x, means, yerr=stds, capsize=5, alpha=0.5, label='mean')

   # Plot individual points
   for result in results_array:
      plt.scatter(x, result, color='red', alpha=0.3, s=30)

   plt.scatter([], [], color='red', alpha=0.3, label='tests')

   plt.xticks(x, commands, rotation=45)
   plt.ylabel('prob')
   plt.legend()
   plt.tight_layout()
   plt.savefig('run.out/summary.png', dpi=300)
   plt.close()

import argparse
```

```
def parse_args():
   parser = argparse.ArgumentParser(description='Audio classification tool')
   parser.add_argument('--test', action='store_true')

   return parser.parse_args()

def main():
   args = parse_args()

   if args.test:
      manual_human_testing()
   else:
      record_and_pred()

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