자연어 처리 실습 5

과목명 자연어 처리

담당교수 김낙현교수님

제출일 20211119

전공 컴퓨터전자시스템

학번 201904458

이름 이준용

한국외국어대학교

HANKUK UNIVERSITY OF FOREIGN STUDIES

```
import numpy as np
import pandas as pd
import re
import shutil
import os
import unicodedata
import urllib3
import zipfile
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
# 데이터 로드
```

```
http = urllib3.PoolManager()
url ='http://www.manythings.org/anki/fra-eng.zip'
filename = 'fra-eng.zip'
path = os.getcwd()
zipfilename = os.path.join(path, filename)
with http.request('GET', url, preload_content=False) as r, open(zipfilename, 'wb') as out_file:
    shutil.copyfileobj(r, out_file)

with zipfile.ZipFile(zipfilename, 'r') as zip_ref:
    zip_ref.extractall(path)
```

```
# 총 33,000개의 샘플을 사용할 예정
```

```
num_samples = 33000
```

```
# 전처리 함수 구현
```

```
def unicode_to_ascii(s):
  return ''.join(c for c in unicodedata.normalize('NFD', s)
      if unicodedata.category(c) != 'Mn')
```

```
def preprocess_sentence(sent):
    # 위에서 구현한 함수를 내부적으로 호출
    sent = unicode_to_ascii(sent.lower())

    # 단어와 구두점 사이에 공백을 만듭니다.
    # Ex) "he is a boy." => "he is a boy ."
    sent = re.sub(r"([?.!,¿])", r" \1", sent)

    # (a-z, A-Z, ".", "?", "!", ",") 이들을 제외하고는 전부 공백으로 변환합니다.
    sent = re.sub(r"[^a-zA-Z!.?]+", r" ", sent)

    sent = re.sub(r"\s+", " ", sent)
    return sent
```

```
# 구현한 전처리 함수 테스트 : 임의의 문장 입력
```

```
# 전처리 테스트
en_sent = u"Have you had dinner?"
fr_sent = u"Avez-vous déjà diné?"
print(preprocess_sentence(en_sent))
print(preprocess_sentence(fr_sent).encode('utf-8'))
```

```
have you had dinner ?
b'avez vous deja dine ?'
```

샘플 데이터를 불러와 모든 전처리를 수행하는 함수 구현

- teacher forcing을 위해 디코더의 입/출력 시퀀스를 따로 분리하여 저장
- 입출력 심볼 추가
- 총 3개의 데이터 셋 생성: 인코더의 입력, 디코더의 입력, 디코더의 실제값

```
def load_preprocessed_data():
    encoder_input, decoder_input, decoder_target = [], [], []

    with open("fra.txt", "r") as lines:
        for i, line in enumerate(lines):

            # source 데이터와 target 데이터 분리
            src_line, tar_line, _ = line.strip().split('₩t')

            # source 데이터 전처리
            src_line_input = [w for w in preprocess_sentence(src_line).split()]

            # target 데이터 전처리
            tar_line = preprocess_sentence(tar_line)
            tar_line_input = [w for w in ("<sos> " + tar_line).split()]
            tar_line_target = [w for w in (tar_line + " <eos>").split()]

            encoder_input.append(src_line_input)
            decoder_input.append(tar_line_input)
            decoder_target.append(tar_line_target)

            if i == num_samples - 1:
                break

    return encoder_input, decoder_input, decoder_target
```

```
# 데이터셋 출력하여 확인
```

```
sents_en_in, sents_fra_in, sents_fra_out = load_preprocessed_data()
print(sents_en_in[:5])
print(sents_fra_in[:5])
print(sents_fra_out[:5])
```

```
[['go', '.'], ['go', '.'], ['go', '.'], ['hi', '.'], ['hi', '.']]
[['<sos>', 'va', '!'], ['<sos>', 'marche', '.'], ['<sos>', 'bouge', '!'], ['<sos>', 'salut', '!'], ['<sos>', 's
[['va', '!', '<eos>'], ['marche', '.', '<eos>'], ['bouge', '!', '<eos>'], ['salut', '!', '<eos>'], ['salut', '.
```

- 단어집합 생성
- 정수 인코딩

```
tokenizer_en = Tokenizer(filters="", lower=False)
tokenizer_en.fit_on_texts(sents_en_in)
encoder_input = tokenizer_en.texts_to_sequences(sents_en_in)

tokenizer_fra = Tokenizer(filters="", lower=False)
tokenizer_fra.fit_on_texts(sents_fra_in)
tokenizer_fra.fit_on_texts(sents_fra_out)
decoder_input = tokenizer_fra.texts_to_sequences(sents_fra_in)
decoder_target = tokenizer_fra.texts_to_sequences(sents_fra_out)
```

- 패딩

```
encoder_input = pad_sequences(encoder_input, padding="post")
decoder_input = pad_sequences(decoder_input, padding="post")
decoder_target = pad_sequences(decoder_target, padding="post")
```

- 단어 집합의 크기 정의

```
src_vocab_size = len(tokenizer_en.word_index) + 1
tar_vocab_size = len(tokenizer_fra.word_index) + 1
print("영어 단어 집합의 크기 : {:d}, 프랑스어 단어 집합의 크기 : {:d}".format(src_vocab_size, tar_vocab_size))
```

```
영어 단어 집합의 크기 : 4606, 프랑스어 단어 집합의 크기 : 8107
```

- 단어로부터 정수를 얻는 딕셔너리와
- 정수로부터 단어를 얻는 딕셔너리를 각각 생성

```
src_to_index = tokenizer_en.word_index
index_to_src = tokenizer_en.index_word # 훈련 후 결과 비교할 때 사용

tar_to_index = tokenizer_fra.word_index # 훈련 후 예측 과정에서 사용
index_to_tar = tokenizer_fra.index_word # 훈련 후 결과 비교할 때 사용
```

테스트 데이터 분리

- 적절한 분포를 갖도록 데이터를 섞어 줌

```
indices = np.arange(encoder_input.shape[0])
np.random.shuffle(indices)
print(indices)
```

```
[13164 13996 13639 ... 22015 13010 10880]
```

```
# 순서가 섞인 정수 시퀀스 리스트 생성



# indices =[]
# for encoder_input in range(10000):
#     if(encoder_input%100 != 53):
#       continue
#     elif(encoder_input>=1054):
#       break
#     else:
#       indices = np.append(indices,encoder_input)
# indices = np.array(indices)
# np.random.shuffle(indices)
# print(indices)
```

```
# 이를 데이터셋의 순서로 지정해주면 샘플들이 기존 순서와 다른 순서로 섞임

encoder_input = encoder_input[indices]
decoder_input = decoder_input[indices]
decoder_target = decoder_target[indices]
```

```
# 확인

# encoder_input[10053]
```

```
# decoder_input[30997]
```

```
# decoder_target[30997]
```

- 훈련 데이터의 10%를 테스트 데이터로 분리

```
n_of_val = int(33000*0.1)
print(n_of_val)
```

```
3300
```

```
# 33,000개의 10%에 해당되는 3,300개의 데이터를 테스트 데이터로 사용

encoder_input_train = encoder_input[:-n_of_val]
decoder_input_train = decoder_input[:-n_of_val]
decoder_target_train = decoder_target[:-n_of_val]

encoder_input_test = encoder_input[-n_of_val:]
decoder_input_test = decoder_input[-n_of_val:]
decoder_target_test = decoder_target[-n_of_val:]
```

```
# 훈련 데이터와 테스트 데이터의 크기(shape)를 출력

print(encoder_input_train.shape)
print(decoder_input_train.shape)
print(decoder_target_train.shape)
print(encoder_input_test.shape)
print(decoder_input_test.shape)
print(decoder_target_test.shape)
```

```
(29700, 8)
(29700, 16)
(29700, 16)
(3300, 8)
(3300, 16)
(3300, 16)
```

2. 기계 번역기 만들기

```
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Masking
from tensorflow.keras.models import Model
```

```
# 임베딩 벡터와 LSTM의 은닉 상태의 크기를 50으로 고정

latent_dim = 50
```

- 모델 설계

```
# 인코더

encoder_inputs = Input(shape=(None,))
enc_emb =  Embedding(src_vocab_size, latent_dim)(encoder_inputs) # 임베딩 층
```

```
enc_masking = Masking(mask_value=0.0)(enc_emb) # 패딩 0은 연산에서 제외
encoder_lstm = LSTM(latent_dim, return_state=True) # 상태값 리턴을 위해 return_state는 True
encoder_outputs, state_h, state_c = encoder_lstm(enc_masking) # 은닉 상태와 셀 상태를 리턴
encoder_states = [state_h, state_c] # 인코더의 은닉 상태와 셀 상태를 저장
```

```
# 디코더

decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(tar_vocab_size, latent_dim) # 임베딩 층
dec_emb = dec_emb_layer(decoder_inputs) # 패딩 0은 연산에서 제외
dec_masking = Masking(mask_value=0.0)(dec_emb)

# 상태값 리턴을 위해 return_state는 True, 모든 시점에 대해서 단어를 예측하기 위해 return_sequences는 True
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)

# 인코더의 은닉 상태를 초기 은닉 상태(initial_state)로 사용
decoder_outputs, _, _ = decoder_lstm(dec_masking,
                                     initial_state=encoder_states)

# 모든 시점의 결과에 대해서 소프트맥스 함수를 사용한 출력층을 통해 단어 예측
decoder_dense = Dense(tar_vocab_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

```
# 모델 정의

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

- 훈련 실행

```
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy', metrics=['acc'])

# 현재 decoder_outputs의 경우에는 원-핫 인코딩을 하지 않은 상태이므로
# sparse_categorical_crossentropy를 사용
```

```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, None)]       0           []

 input_2 (InputLayer)           [(None, None)]       0           []

 embedding (Embedding)          (None, None, 50)     230300      ['input_1[0][0]']

 embedding_1 (Embedding)        (None, None, 50)     405350      ['input_2[0][0]']

 masking (Masking)              (None, None, 50)     0           ['embedding[0][0]']

 masking_1 (Masking)            (None, None, 50)     0           ['embedding_1[0][0]']

 lstm (LSTM)                    [(None, 50),         20200       ['masking[0][0]']
                                 (None, 50),
                                 (None, 50)]

 lstm_1 (LSTM)                  [(None, None, 50),   20200       ['masking_1[0][0]',
                                 (None, 50),                      'lstm[0][1]',
                                 (None, 50)]                      'lstm[0][2]']

 dense (Dense)                  (None, None, 8107)   413457      ['lstm_1[0][0]']

==================================================================================================
```

```
Total params: 1,089,507
Trainable params: 1,089,507
Non-trainable params: 0
```

```
model.fit(x = [encoder_input_train, decoder_input_train], y = decoder_target_train, ₩
          validation_data = ([encoder_input_test, decoder_input_test], decoder_target_test),
          batch_size = 128, epochs = 50)
```

```
233/233 [==============================] - 9s 38ms/step - loss: 0.8171 - acc: 0.8582 - val_loss: 0.9442 - va
Epoch 24/50
233/233 [==============================] - 9s 37ms/step - loss: 0.8055 - acc: 0.8601 - val_loss: 0.9374 - va
Epoch 25/50
233/233 [==============================] - 9s 37ms/step - loss: 0.7946 - acc: 0.8616 - val_loss: 0.9378 - va
Epoch 26/50
233/233 [==============================] - 9s 37ms/step - loss: 0.7844 - acc: 0.8632 - val_loss: 0.9097 - va
Epoch 27/50
233/233 [==============================] - 9s 37ms/step - loss: 0.7738 - acc: 0.8647 - val_loss: 0.9072 - va
Epoch 28/50
233/233 [==============================] - 9s 37ms/step - loss: 0.7644 - acc: 0.8662 - val_loss: 0.9075 - va
Epoch 29/50
233/233 [==============================] - 9s 38ms/step - loss: 0.7553 - acc: 0.8675 - val_loss: 0.8982 - va
Epoch 30/50
233/233 [==============================] - 9s 38ms/step - loss: 0.7465 - acc: 0.8694 - val_loss: 0.9021 - va

Epoch 31/50
233/233 [==============================] - 9s 38ms/step - loss: 0.7385 - acc: 0.8707 - val_loss: 0.8969 - va
Epoch 32/50
233/233 [==============================] - 9s 38ms/step - loss: 0.7311 - acc: 0.8725 - val_loss: 0.9022 - va
Epoch 33/50
233/233 [==============================] - 9s 38ms/step - loss: 0.7243 - acc: 0.8736 - val_loss: 0.8812 - va
Epoch 34/50
233/233 [==============================] - 9s 38ms/step - loss: 0.7181 - acc: 0.8748 - val_loss: 0.8892 - va
Epoch 35/50
233/233 [==============================] - 9s 38ms/step - loss: 0.7119 - acc: 0.8762 - val_loss: 0.8798 - va
Epoch 36/50
233/233 [==============================] - 9s 38ms/step - loss: 0.7060 - acc: 0.8774 - val_loss: 0.8881 - va
Epoch 37/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6998 - acc: 0.8787 - val_loss: 0.8885 - va
Epoch 38/50
233/233 [==============================] - 9s 37ms/step - loss: 0.6937 - acc: 0.8798 - val_loss: 0.8762 - va
Epoch 39/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6880 - acc: 0.8811 - val_loss: 0.8759 - va
Epoch 40/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6821 - acc: 0.8819 - val_loss: 0.8715 - va
Epoch 41/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6763 - acc: 0.8831 - val_loss: 0.8702 - va
Epoch 42/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6713 - acc: 0.8844 - val_loss: 0.8614 - va
Epoch 43/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6663 - acc: 0.8853 - val_loss: 0.8653 - va
Epoch 44/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6614 - acc: 0.8866 - val_loss: 0.8695 - va
Epoch 45/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6564 - acc: 0.8876 - val_loss: 0.8718 - va
Epoch 46/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6515 - acc: 0.8885 - val_loss: 0.8548 - va
Epoch 47/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6466 - acc: 0.8894 - val_loss: 0.8579 - va
Epoch 48/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6416 - acc: 0.8904 - val_loss: 0.8542 - va
Epoch 49/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6365 - acc: 0.8912 - val_loss: 0.8461 - va
Epoch 50/50
233/233 [==============================] - 9s 38ms/step - loss: 0.6319 - acc: 0.8922 - val_loss: 0.8517 - va
<keras.callbacks.History at 0x7f1d56ca1e90>
```

3. seq2seq 기계 번역기 동작시키기

```
# 테스트를 위해 모델 재설계

# 인코더
encoder_model = Model(encoder_inputs, encoder_states)

# 디코더
# 이전 시점의 상태를 보관할 텐서
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

# 훈련 때 사용했던 임베딩 층을 재사용
dec_emb2= dec_emb_layer(decoder_inputs)

# 다음 단어 예측을 위해 이전 시점의 상태를 현 시점의 초기 상태로 사용
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=decoder_states_inputs)
decoder_states2 = [state_h2, state_c2]

# 모든 시점에 대해서 단어 예측
decoder_outputs2 = decoder_dense(decoder_outputs2)
```

```
# 디코더 정의

decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs2] + decoder_states2)
```

- 테스트 과정에서의 동작을 위한 decode_sequence 함수를 구현

```
def decode_sequence(input_seq):
    # 입력으로부터 인코더의 상태를 얻음
    states_value = encoder_model.predict(input_seq)

    # <SOS>에 해당하는 정수 생성
    target_seq = np.zeros((1,1))
    target_seq[0, 0] = tar_to_index['<sos>']

    stop_condition = False
    decoded_sentence = ''

    # stop_condition이 True가 될 때까지 루프 반복
    # 구현의 간소화를 위해서 이 함수는 배치 크기를 1로 가정합니다.
    while not stop_condition:
        # 이점 시점의 상태 states_value를 현 시점의 초기 상태로 사용
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # 예측 결과를 단어로 변환
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = index_to_tar[sampled_token_index]

         # 현재 시점의 예측 단어를 예측 문장에 추가
        decoded_sentence += ' '+sampled_char

        # <eos>에 도달하거나 정해진 길이를 넘으면 중단.
        if (sampled_char == '<eos>' or
           len(decoded_sentence) > 50):
            stop_condition = True

        # 현재 시점의 예측 결과를 다음 시점의 입력으로 사용하기 위해 저장
        target_seq = np.zeros((1,1))
```

```
        target_seq[0, 0] = sampled_token_index

        # 현재 시점의 상태를 다음 시점의 상태로 사용하기 위해 저장
        states_value = [h, c]

    return decoded_sentence
```

- 결과 확인을 위한 함수 구현

```
# 원문의 정수 시퀀스를 텍스트 시퀀스로 변환
def seq2src(input_seq):
    temp=''
    for i in input_seq:
        if(i!=0):
            temp = temp + index_to_src[i]+' '
    return temp

# 번역문의 정수 시퀀스를 텍스트 시퀀스로 변환
def seq2tar(input_seq):
    temp=''
    for i in input_seq:
        if((i!=0 and i!=tar_to_index['<sos>']) and i!=tar_to_index['<eos>']):
            temp = temp + index_to_tar[i] + ' '
    return temp
```

- 훈련 데이터에 대해서 임의로 선택한 인덱스의 샘플의 결과를 출력

```
for seq_index in [158,258,358,458,558,658,658,758,858,958,1058]:
  input_seq = encoder_input_train[seq_index: seq_index + 1]
  decoded_sentence = decode_sequence(input_seq)

  print("원문 : ",seq2src(encoder_input_train[seq_index]))
  print("번역문 :",seq2tar(decoder_input_train[seq_index]))
  print("예측문 :",decoded_sentence[:-5])
  print("₩n")
```

```
원문 :  i got a suntan .
번역문 : je suis halee .
예측문 :  je suis me suis .


원문 :  we have a deal .
번역문 : nous avons etabli un accord .
예측문 :  nous avons un plan .


원문 :  i can see that now .
번역문 : je m en rends compte a l heure actuelle .
예측문 :  je peux faire maintenant .


원문 :  that feels amazing .
번역문 : ce qu on ressent est incroyable !
예측문 :  ca va va ?


원문 :  have a nice trip .
번역문 : bon voyage .
예측문 :  bon voyage !


원문 :  i bought a red tie .
번역문 : j ai achete une cravate rouge .
```

```
예측문 :  j ai achete une voiture .


원문 :  i bought a red tie .
번역문 : j ai achete une cravate rouge .
예측문 :  j ai achete une voiture .


원문 :  you re worried .
번역문 : vous etes inquiets .
예측문 :  tu es pas .


원문 :  you stay here .
번역문 : tu restes ici .
예측문 :  tu t en ici .


원문 :  i left you a note .
번역문 : je vous laissai une note .
예측문 :  je vous ai laisse un coup de nous .


원문 :  i m being used .
번역문 : on m utilise .
예측문 :  j ai ete en train de faire .
```

- 테스트 데이터에 대해서 임의로 선택한 인덱스의 샘플의 결과를 출력

```
for seq_index in [158,258,358,458,558,658,658,758,858,958,1058]:
  input_seq = encoder_input_test[seq_index: seq_index + 1]
  decoded_sentence = decode_sequence(input_seq)

  print("원문 : ",seq2src(encoder_input_test[seq_index]))
  print("번역문 :",seq2tar(decoder_input_test[seq_index]))
  print("예측문 :",decoded_sentence[:-5])
  print("Wn")
```

```
원문 :  they voted .
번역문 : ils ont vote .
예측문 :  ils ont chose .


원문 :  we re both crazy .
번역문 : nous sommes tous deux fous .
예측문 :  nous sommes tous deux ?


원문 :  the soup is thick .
번역문 : la soupe est epaisse .
예측문 :  le est la porte .


원문 :  he s gone senile .
번역문 : il est devenu senile .
예측문 :  il est devenu parti .


원문 :  i hate that movie .
번역문 : je deteste ce film .
예측문 :  je deteste ce que je l aime .


원문 :  it was disgusting .
번역문 : c etait repugnant .
예측문 :  c etait fantastique .
```

```
원문 :  it was disgusting .
번역문 : c etait repugnant .
예측문 :  c etait fantastique .


원문 :  i ache all over .
번역문 : mon corps me fait mal .
예측문 :  j ai dit .


원문 :  i feel normal .
번역문 : je me sens normal .
예측문 :  je me sens a vin .


원문 :  you guys wait here .
번역문 : attendez ici .
예측문 :  tu as l argent .


원문 :  your cat is fat .
번역문 : votre chatte est grosse .
예측문 :  mon chat est ce chien .
```

전체 모델 저장하기

model.save 메서드를 호출하여 모델의 구조, 가중치, 훈련 설정을 하나의 파일/폴더에 저장합니다. 모델을 저장하기 때문에 원본 파이썬 코드*가 없어도 사용할 수 있습니다. 옵티마이저 상태가 복원되므로 정확히 중지한 시점에서 다시 훈련을 시작할 수 있습니다.

전체 모델은 두 가지 다른 파일 형식(SavedModel 및 HDF5)으로 저장할 수 있습니다. 전체 모델을 저장하는 기능은 매우 유용합니다. TensorFlow.js로 모델을 로드한 다음 웹 브라우저에서 모델을 훈련하고 실행할 수 있습니다(Saved Model, HDF5). 또는 모바일 장치에 맞도록 변환한 다음 TensorFlow Lite를 사용하여 실행할 수 있습니다(Saved Model, HDF5).

SavedModel 포맷

SavedModel 형식은 모델을 직렬화하는 또 다른 방법입니다. 이 형식으로 저장된 모델은 tf.keras.models.load_model을 사용하여 복원할 수 있으며 TensorFlow Serving과 호환됩니다.

새로운 모델 객체를 만들고 훈련합니다

model = create_model() model.fit(train_images, train_labels, epochs=5)

SavedModel로 전체 모델을 저장합니다

!mkdir -p saved_model model.save('saved_model/my_model')

SavedModel 형식은 protobuf 바이너리와 TensorFlow 체크포인트를 포함하는 디렉토리입니다. 저장된 모델 디렉토리를 검사합니다.

my_model 디렉토리

ls saved_model

assests 폴더, saved_model.pb, variables 폴더

ls saved_model/my_model

저장된 모델로부터 새로운 케라스 모델을 로드합니다:

new_model = tf.keras.models.load_model('saved_model/my_model')

```
# 모델 구조를 확인합니다

new_model.summary()
```

복원된 모델은 원본 모델과 동일한 매개변수로 컴파일되어 있습니다. 이 모델을 평가하고 예측에 사용해 보죠

```
# 복원된 모델을 평가합니다

loss, acc = new_model.evaluate(test_images, test_labels, verbose=2) print('복원된 모델의 정확도: {:5.2f}%'.format(100*acc))

print(new_model.predict(test_images).shape)
```

HDF5 파일로 저장하기

```
# 새로운 모델 객체를 만들고 훈련합니다

model = create_model() model.fit(train_images, train_labels, epochs=5)

# 전체 모델을 HDF5 파일로 저장합니다 # '.h5' 확장자는 이 모델이 HDF5로 저장되었다는 것을 나타냅니다

model.save('my_model.h5')

# 가중치와 옵티마이저를 포함하여 정확히 동일한 모델을 다시 생성합니다

new_model = tf.keras.models.load_model('my_model.h5')

# 모델 구조를 출력합니다 new_model.summary()

# 정확도를 확인해 보겠습니다

loss, acc = new_model.evaluate(test_images, test_labels, verbose=2) print('복원된 모델의 정확도: {:5.2f}%'.format(100*acc))
```

✓ 4초 오전 12:14에 완료됨