프로젝트	한국어 문서 생성 요약
파일명	abstract_summary.ipynb
용도	seq2seq 모델과 attention 모델을 활용한 생성 요약 모델 훈련

코드

```
import json
import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from bs4 import BeautifulSoup
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import urllib.request

# 토크나이저 선언
!pip install konlpy
from konlpy.tag import Komoran
komoran = Komoran()

# 단어를 입력받아 필수 형태소 부분만 추출
def tokenize(word):
  tokens = komoran.pos(word, join=True)
  tokened_word = ''.join([w.split('/')[0] for w in tokens if ('/NN'
in w or '/XR' in w or '/VA' in w or '/VV' in w or '/SN' in w )])
  return tokened_word


# 전처리 함수 선언
# 한글, 영어, 숫자만 가져오며, ()안 내용은 삭제
def preprocess_sentence(article):

    article = ''.join(re.findall('[ 가-힣a-zA-Z0-9()]',  article) )
    bracket = re.findall(r'\([^)]*\)', article )
    for i in bracket:
          article = article.replace(i,'')

    return article.strip(' ')
```

```
# 훈련에 사용될 train 데이터 불러오기
input_file_name = '/train.jsonl'

# train 데이터를 모델 학습에 필요한 전처리 과정을 거쳐 dataframe 형식으로 저장
with open(input_file_name, 'r', encoding = 'utf-8', newline = '') as
input_file:
    i = 0
    for line in input_file:
        line = json.loads(line)
        id_num, sents , summary_ori ,answer_index =
list(line.values())[1:]
        ori_sents = ' '.join([sents[s] for s in answer_index ])

        summary_3 = preprocess_sentence(ori_sents)
        tokenized = ' '.join([tokenize(w) for w in summary_3.split('
') ] )

        test1.loc[i] = [ori_sents,  tokenized , summary_ori ]
        i += 1
```

결과물

	text_ori	tokenized_text	summary_ori
0	당진시 문화관광과를 대상으로 하는 행정사무감사에서 당진시립합창단 관계자가 보낸 것으...	당진시 문화관광과 대상 하 행정사감사 당진시립합창단 관계자 보내 것 추정 문자 관련...	지난 6일 당진시의회 행정사무감사에서 '합창단이 소리를 작게 낼 것이니 알고 있으라...
1	미국 메이저리그(MLB)에서 활동하는 한국 선수들의 시즌 초반 희비가 엇갈리고 있다...	미국 메이저리그 활동 한국 선수 시즌 초반 희비 엇갈리 있 에인절스 최지만 맹활약...	LA 에인절스의 최지만이 맹활약을 하여 시즌 타율 0.250에서 0.313으로 올리...
2	16일 부평구와 협회 등에 따르면 부영공원 안에 있는 야구장을 구생활체육야구협회가 ...	16일 부평구 협회 등 따르 부영공원 안 있 야구장 생활체육야구협회 무상 사용 이...	16일 부평구와 협회 등에 따르면 부영공원 안에 있는 야구장을 구생활체육야구협회가 ...
3	대구·경북첨단의료산업진흥재단(이하 대구첨복재단) 의약생산센터는 주세제 특수제제인 세...	대구경북첨단의료산업진흥재단 의약생산센터 세제 특수제제 세포독성 항암주사제 품질관리기준...	대구·경북첨단의료산업진흥재단 의약생산센터는 약사법 시행규칙에서 정한 바에 따라 전용...
4	식품의약품안전처는 29일 여름철 어린이가 즐겨 마시는 탄산음료 282개와 혼합음료 ...	식품의약품안전처 29일 여름철 어린이 즐기 마시 탄산음료 282개 혼합음료 350개...	식품의약품안전처는 29일 어린이가 즐겨마시는 음료를 대상으로 영양성분을 조사한 결과,...
...	...	...	...
42798	칠곡북삼초등학교(교장 서금자)에서는 지난달 7일부터 18일까지 3-5학년 학생 중 ...	칠곡북삼초등학교 지난달 7일 18일 35학년 학생 중 희망학생 대상 2018 겨울 ...	지난달 7일부터 18일까지 칠곡북삼초등학교는 학생들에게 실제 영어 사용에 대한 자신...
42799	최근 5년간, LH공공임대 임차인이 사망했으나 상속되거나 반환되지 못한 임대보증금이...	최근 5년간 공공임대 임차인 사망 상속 반환 임대보증금 96여원 넘 것 나타나 1...	감상훈 자유한국당 의원은 최근 5년간 임차인의 사망으로 반환되지 못한 3천479가구...
42800	경기도의 'DMZ 155마일 걷기' 행사가 5일 오전 파주 임진각에서 열린 출정식을...	경기도 155마일 걷기 행사 5일 오전 파주 임진각 열리 출정식 시작 기나길 여정...	경기도는 DMZ의 의미와 평화에 관한 국민들의 인식을 새롭게 하고자 'DMZ 1...
42801	지난 14일 기준 휘발유 가격이 가장 저렴한 곳은 AJ토탈(주)당진주유소(신평)였으...	지나 14일 기준 휘발유 가격 저렴 곳 토탈당진주유소 가격 1488원 반면 휘발유...	지난 14일 기준 당진지역 휘발유 중 가장 저렴한 곳의 가격은 AJ토탈(주)당진주유...
42802	양승조 충남도지사는 12일 대기오염물질 배출 주요 사업장인 현대제철㈜ 당진공장을 찾...	양승조 충남도지사 12일 대기오염물질 배출 주요 사업장 현대제철 당진공장 찾 미세먼...	양승조 충남도지사는 12일 대기오염물질 배출 주요 사업장인 현대제철㈜ 당진공장에서 ...

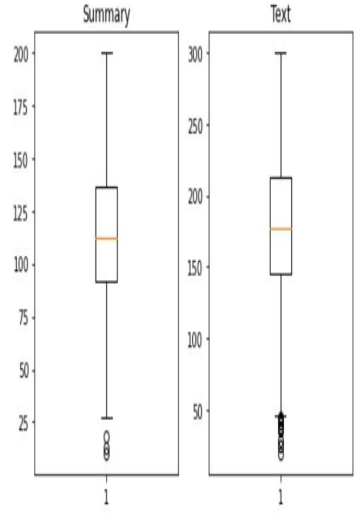
42803 rows × 3 columns

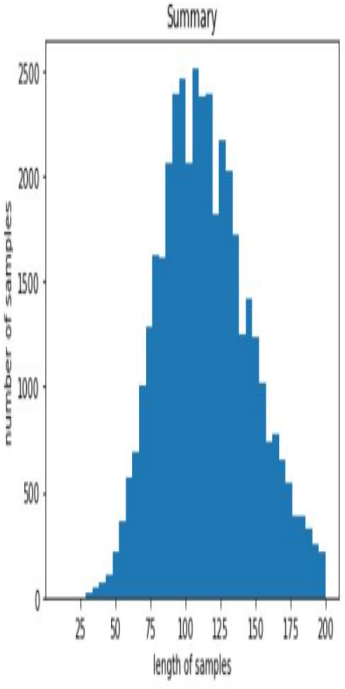
```
# Text와 Summary 길이 분포 확인
text_len = [len(s) for s in test1['tokenized_text']]
summary_len = [len(s) for s in test1['summary_ori']]

print('텍스트의 최소 길이 : {}'.format(np.min(text_len)))
print('텍스트의 최대 길이 : {}'.format(np.max(text_len)))
print('텍스트의 평균 길이 : {}'.format(np.mean(text_len)))
print('요약의 최소 길이 : {}'.format(np.min(summary_len)))
print('요약의 최대 길이 : {}'.format(np.max(summary_len)))
print('요약의 평균 길이 : {}'.format(np.mean(summary_len)))
```

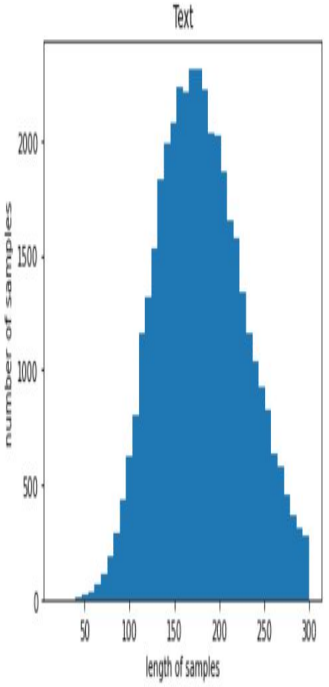
```
plt.subplot(1,2,1)
plt.boxplot(summary_len)
plt.title('Summary')
plt.subplot(1,2,2)
plt.boxplot(text_len)
plt.title('Text')
plt.tight_layout()
plt.show()

plt.title('Summary')
plt.hist(summary_len, bins=40)
plt.xlabel('length of samples')
plt.ylabel('number of samples')
plt.show()

plt.title('Text')
plt.hist(text_len, bins=40)
plt.xlabel('length of samples')
plt.ylabel('number of samples')
plt.show()
```

결과물

텍스트의 최소 길이 : 19
텍스트의 최대 길이 : 300
텍스트의 평균 길이 : 179.7237512242889
요약의 최소 길이 : 10
요약의 최대 길이 : 200
요약의 평균 길이 : 114.72154750244859







```
# 최대 길이를 정하고, 정해진 최대 길이보다 큰 샘플들은 제거
summary_max_len = 200
text_max_len = 300
test1 = test1[test1['tokenized_text'].apply(lambda x: len(x) <=
text_max_len)]
test1 = test1[test1['summary_ori'].apply(lambda x: len(x) <=
summary_max_len)]

# seq2seq 훈련을 위해 시작 토큰과 종료 토큰 추가
test1['decoder_input'] = test1['tokenized_text'].apply(lambda x :
'sostoken '+ x)
test1['decoder_target'] = test1['summary_ori'].apply(lambda x : x + '
eostoken')
test1.to_csv('/content/gdrive/My
Drive/extract_summary/create_test2.csv')

# 인코더의 입력, 디코더의 입력과 레이블 Numpy 타입으로 저장
encoder_input = np.array(test1['tokenized_text'])
decoder_input = np.array(test1['decoder_input'])
decoder_target = np.array(test1['decoder_target'])

# 크기와 형태가 같은 정수 시퀀스 만듦
indices = np.arange(encoder_input.shape[0])
np.random.seed(42)
np.random.shuffle(indices)

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

n_of_val = int(len(encoder_input) * 0.2)

# 테스트 데이터의 갯수를 이용해 전체 데이터를 양분
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:]
```

```
print("훈련 데이터의 갯수 :", len(encoder_input_train))
print("훈련 레이블의 갯수 :", len(decoder_input_train))
print("테스트 데이터의 갯수 :", len(encoder_input_test))
print("테스트 데이터의 갯수 :", len(decoder_input_test))
```

결과물

```
훈련 데이터의 갯수 : 32672
훈련 레이블의 갯수 : 32672
테스트 데이터의 갯수 : 8168
테스트 데이터의 갯수 : 8168
```

```
# 정수 인코딩
# 각각의 단어들한테 정수 값을 배정하고, 정수로 치환시킴
src_tokenizer = Tokenizer()
src_tokenizer.fit_on_texts(encoder_input_train)

# text 데이터에서 총 단어의 수와 빈도수가 낮은 단어들을 출력함
threshold = 5
total_cnt = len(src_tokenizer.word_index) # 단어의 수
rare_cnt = 0 # 등장 빈도수가 threshold보다 작은 단어의 개수를 카운트
total_freq = 0 # 훈련 데이터의 전체 단어 빈도수 총 합
rare_freq = 0 # 등장 빈도수가 threshold보다 작은 단어의 등장 빈도수의 총 합

# 단어와 빈도수의 쌍(pair)을 key와 value로 받는다.
for key, value in src_tokenizer.word_counts.items():
    total_freq = total_freq + value

    # 단어의 등장 빈도수가 threshold보다 작으면
    if(value < threshold):
        rare_cnt = rare_cnt + 1
        rare_freq = rare_freq + value

print('단어 집합(vocabulary)의 크기 :',total_cnt)
print('등장 빈도가 %s번 이하인 희귀 단어의 수: %s'%(threshold - 1,
rare_cnt))
print('단어 집합에서 희귀 단어를 제외시킬 경우의 단어 집합의 크기
%s'%(total_cnt - rare_cnt))
print("단어 집합에서 희귀 단어의 비율:", (rare_cnt / total_cnt)*100)
print("전체 등장 빈도에서 희귀 단어 등장 빈도 비율:", (rare_freq /
total_freq)*100)
```

결과물

```
단어 집합(vocabulary)의 크기 : 145520
등장 빈도가 4번 이하인 희귀 단어의 수: 120254
단어 집합에서 희귀 단어를 제외시킬 경우의 단어 집합의 크기 25266
단어 집합에서 희귀 단어의 비율: 82.63743815283122
전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 11.038235847920905
```

```
src_vocab = 25266 # 단어 집합의 크기 설정
src_tokenizer = Tokenizer(num_words = src_vocab) # 단어 집합의 크기를 으로 제한
src_tokenizer.fit_on_texts(encoder_input_train) # 단어 집합 재생성

# texts_to_sequences()는 생성된 단어 집합에 기반하여 입력으로 주어진 텍스트 데이터의 단어들을 모두 정수로 변환하는 정수 인코딩을 수행
# 텍스트 시퀀스를 정수 시퀀스로 변환
encoder_input_train =
src_tokenizer.texts_to_sequences(encoder_input_train)
encoder_input_test =
src_tokenizer.texts_to_sequences(encoder_input_test)



# summary 데이터에서 총 단어의 수와 빈도수가 낮은 단어들을 출력함
tar_tokenizer = Tokenizer()
tar_tokenizer.fit_on_texts(decoder_input_train)

threshold = 5
total_cnt = len(tar_tokenizer.word_index) # 단어의 수
rare_cnt = 0 # 등장 빈도수가 threshold보다 작은 단어의 개수를 카운트
total_freq = 0 # 훈련 데이터의 전체 단어 빈도수 총 합
rare_freq = 0 # 등장 빈도수가 threshold보다 작은 단어의 등장 빈도수의 총 합

# 단어와 빈도수의 쌍(pair)을 key와 value로 받는다.
for key, value in tar_tokenizer.word_counts.items():
    total_freq = total_freq + value

    # 단어의 등장 빈도수가 threshold보다 작으면
    if(value < threshold):
        rare_cnt = rare_cnt + 1
        rare_freq = rare_freq + value

print('단어 집합(vocabulary)의 크기 :',total_cnt)
```

```
print('등장 빈도가 %s번 이하인 희귀 단어의 수: %s'%(threshold - 1,
rare_cnt))
print('단어 집합에서 희귀 단어를 제외시킬 경우의 단어 집합의 크기
%s'%(total_cnt - rare_cnt))
print("단어 집합에서 희귀 단어의 비율:", (rare_cnt / total_cnt)*100)
print("전체 등장 빈도에서 희귀 단어 등장 빈도 비율:", (rare_freq /
total_freq)*100)
```

결과물

```
단어 집합(vocabulary)의 크기 : 145521
등장 빈도가 4번 이하인 희귀 단어의 수: 120254
단어 집합에서 희귀 단어를 제외시킬 경우의 단어 집합의 크기 25267
단어 집합에서 희귀 단어의 비율: 82.63687027989087
전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 10.819994347913175
```

```
tar_vocab = 25267
tar_tokenizer = Tokenizer(num_words = tar_vocab)

tar_tokenizer.fit_on_texts(decoder_input_train)
tar_tokenizer.fit_on_texts(decoder_target_train)

# 텍스트 시퀀스를 정수 시퀀스로 변환
decoder_input_train =
tar_tokenizer.texts_to_sequences(decoder_input_train)
decoder_target_train =
tar_tokenizer.texts_to_sequences(decoder_target_train)
decoder_input_test =
tar_tokenizer.texts_to_sequences(decoder_input_test)
decoder_target_test =
tar_tokenizer.texts_to_sequences(decoder_target_test)

# 빈 샘플 제거
drop_train = [index for index, sentence in
enumerate(decoder_input_train) if len(sentence) == 1]
drop_test = [index for index, sentence in
enumerate(decoder_input_test) if len(sentence) == 1]

print("삭제할 훈련 데이터의 갯수 : ", len(drop_train))
print("삭제할 테스트 데이터의 갯수 : ", len(drop_test))

encoder_input_train = np.delete(encoder_input_train, drop_train,
```

```
axis=0)
decoder_input_train = np.delete(decoder_input_train, drop_train,
axis=0)
decoder_target_train = np.delete(decoder_target_train, drop_train,
axis=0)

encoder_input_test = np.delete(encoder_input_test, drop_test, axis=0)
decoder_input_test = np.delete(decoder_input_test, drop_test, axis=0)
decoder_target_test = np.delete(decoder_target_test, drop_test,
axis=0)


# 딥러닝 모델 입력을 위한 데이터 패딩 작업

encoder_input_train = pad_sequences(encoder_input_train,
maxlen=text_max_len, padding='post')
encoder_input_test = pad_sequences(encoder_input_test, maxlen =
text_max_len, padding='post')

decoder_input_train = pad_sequences(decoder_input_train, maxlen =
summary_max_len, padding='post')
decoder_input_test = pad_sequences(decoder_input_test, maxlen =
summary_max_len, padding='post')

decoder_target_train = pad_sequences(decoder_target_train, maxlen =
summary_max_len, padding='post')
decoder_target_test = pad_sequences(decoder_target_test, maxlen =
summary_max_len, padding='post')


encoder_input_train = pad_sequences(encoder_input_train,
padding='post')
encoder_input_test = pad_sequences(encoder_input_test,
padding='post')

decoder_input_train = pad_sequences(decoder_input_train,
padding='post')
decoder_input_test = pad_sequences(decoder_input_test,
padding='post')

decoder_target_train = pad_sequences(decoder_target_train,
padding='post')
```

```
decoder_target_test = pad_sequences(decoder_target_test,
padding='post')

# seq2seq + Attention으로 요약 모델 생성 및 훈련

from tensorflow.keras.layers import Input, LSTM, Embedding, Dense,
Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import tensorflow as tf

# 인코더를 설계. 인코더는 LSTM 층을 3개 쌓음
embedding_dim = 128
hidden_size = 256

# 인코더
encoder_inputs = Input(shape=(text_max_len,))
# encoder_inputs = Input(shape=(None,))

# 인코더의 임베딩 층
enc_emb = Embedding(src_vocab, embedding_dim)(encoder_inputs)
# enc_emb = Embedding(2500, embedding_dim)(encoder_inputs)

# 인코더의 LSTM 1
encoder_lstm1 = LSTM(hidden_size, return_sequences=True,
return_state=True ,dropout = 0.4, recurrent_dropout = 0.4)
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

# 인코더의 LSTM 2
encoder_lstm2 = LSTM(hidden_size, return_sequences=True,
return_state=True, dropout=0.4, recurrent_dropout=0.4)
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

# 인코더의 LSTM 3
encoder_lstm3 = LSTM(hidden_size, return_state=True,
return_sequences=True, dropout=0.4, recurrent_dropout=0.4)
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

# 디코더를 설계, 단, 출력층은 제외하고 설계
# (초기 상태(initial_state)를 인코더의 상태로 주어야 하므로)
# 디코더
decoder_inputs = Input(shape=(None,))
```

```
# 디코더의 임베딩 층
dec_emb_layer = Embedding(tar_vocab, embedding_dim)
# dec_emb_layer = Embedding(2500, embedding_dim)
dec_emb = dec_emb_layer(decoder_inputs)

# 디코더의 LSTM
decoder_lstm = LSTM(hidden_size, return_sequences = True,
return_state = True, dropout = 0.4, recurrent_dropout=0.2)
decoder_outputs, _, _ = decoder_lstm(dec_emb, initial_state =
[state_h, state_c])

# 디코더 출력층 설계
# 디코더 출력층
decoder_softmax_layer = Dense(tar_vocab, activation = 'softmax')
# decoder_softmax_layer = Dense(2500, activation = 'softmax')
decoder_softmax_outputs = decoder_softmax_layer(decoder_outputs)

# 모델 정의
model = Model([encoder_inputs, decoder_inputs],
decoder_softmax_outputs)
model.summary()
```

결과물

```
Model: "functional_5"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 300)]        0
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 300, 128)     3234048     input_3[0][0]
__________________________________________________________________________________________________
lstm_4 (LSTM)                   [(None, 300, 256), ( 394240      embedding_2[0][0]
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
lstm_5 (LSTM)                   [(None, 300, 256), ( 525312      lstm_4[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, None, 128)    3234176     input_4[0][0]
__________________________________________________________________________________________________
lstm_6 (LSTM)                   [(None, 300, 256), ( 525312      lstm_5[0][0]
__________________________________________________________________________________________________
lstm_7 (LSTM)                   [(None, None, 256),  394240      embedding_3[0][0]
                                                                 lstm_6[0][1]
                                                                 lstm_6[0][2]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, None, 25267)  6493619     lstm_7[0][0]
==================================================================================================
Total params: 14,800,947
Trainable params: 14,800,947
Non-trainable params: 0
```

```
# Attention 모델이 결합된 모델 생성
# 어텐션 함수
urllib.request.urlretrieve("https://raw.githubusercontent.com/thushv8
```

```
9/attention_keras/master/src/layers/attention.py",
filename="attention.py")
from attention import AttentionLayer

# 어텐션 메커니즘을 이용 디코더의 출력층 설계
# 어텐션 층(어텐션 함수)
attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs,
decoder_outputs])

# 어텐션의 결과와 디코더의 hidden state들을 연결
decoder_concat_input = Concatenate(axis = -1,
name='concat_layer')([decoder_outputs, attn_out])

# 디코더의 출력층
decoder_softmax_layer = Dense(tar_vocab, activation='softmax')
# decoder_softmax_layer = Dense(2500, activation='softmax')
decoder_softmax_outputs = decoder_softmax_layer(decoder_concat_input)

# 모델 정의
model = Model([encoder_inputs, decoder_inputs],
decoder_softmax_outputs)
model.summary()
```

결과물

```
Model: "functional_7"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 300)]        0
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 300, 128)     3234048     input_3[0][0]
__________________________________________________________________________________________________
lstm_4 (LSTM)                   [(None, 300, 256), ( 394240      embedding_2[0][0]
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
lstm_5 (LSTM)                   [(None, 300, 256), ( 525312      lstm_4[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, None, 128)    3234176     input_4[0][0]
__________________________________________________________________________________________________
lstm_6 (LSTM)                   [(None, 300, 256), ( 525312      lstm_5[0][0]
__________________________________________________________________________________________________
lstm_7 (LSTM)                   [(None, None, 256),  394240      embedding_3[0][0]
                                                                 lstm_6[0][1]
                                                                 lstm_6[0][2]
__________________________________________________________________________________________________
attention_layer (AttentionLayer ((None, None, 256),  131328      lstm_6[0][0]
                                                                 lstm_7[0][0]
__________________________________________________________________________________________________
concat_layer (Concatenate)      (None, None, 512)    0           lstm_7[0][0]
                                                                 attention_layer[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, None, 25267)  12961971    concat_layer[0][0]
==================================================================================================
Total params: 21,400,627
Trainable params: 21,400,627
```

```
model.compile(optimizer='rmsprop',
loss='sparse_categorical_crossentropy')

# 조기 종료 조건을 설정하고 모델을 학습
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience = 2)
# 모델을 저장
save_model =
tf.keras.callbacks.ModelCheckpoint(filepath='/content/gdrive/My
Drive/extract_summary/model.{epoch:02d}-{val_loss:.2f}.h5',
save_best_only=False)
save_model_best =
tf.keras.callbacks.ModelCheckpoint(filepath='/content/gdrive/My
Drive/extract_summary/model_best.h5', save_best_only=True)

history = 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 = 16, callbacks=[es, save_model, save_model_best
```

```
], epochs = 30)
```

결과물

```
Epoch 1/10
2042/2042 [==============================] - 4090s 2s/step - loss: 0.7764 - val_loss: 0.7584
Epoch 2/10
2042/2042 [==============================] - 4095s 2s/step - loss: 0.7694 - val_loss: 0.7533
Epoch 3/10
2042/2042 [==============================] - 4065s 2s/step - loss: 0.7655 - val_loss: 0.7569
Epoch 4/10
2042/2042 [==============================] - 4067s 2s/step - loss: 0.7612 - val_loss: 0.7617
```