인공지능론 최종보고서

필기 한자 인식모델 구현 및 히라가나 변환

산업경영공학과
2018100922
이승건

Contents

주제 선정

일본 성씨 개수

약 12만 3000여 종

한자로 구성

음독, 훈독 등 다양한 발음법

일본 이름을 익히는 과정에서 어려움을 겪음

주제 선정

일본어 손글씨 학습

佐
藤

손글씨 인식

さとう

히라가나 변환

일본어 손글씨를 학습하여 손글씨 인식, 히라가나로 변환하는 모델

기술 수준 및 동향

OCR (Optical Character Recognition)

여러 검출 방법에 따른 기술 모델 구현 수준

Methods	Models
Bounding box regression	TextBoxes, TextBoxes++, DMPNet, SSTD, RRD, EAST, DeRPN
Part-based methods	SegLink, SegLink++
Segmentation-based methods	Mask TextSpotter, PSENet, TextSnake, Pixellink
Fast scene text detection method	TextBoxes, TextBoxes++, SegLink, RRD, EAST, DBNet, DENet++, CentripetalText(CT)

출처 Reproduced from [7].

필요한 상황 별 다양한 모델을 통해 정확도 향상 가능

왜곡된 글자 형태 처리 가능

데이터 수집방안

etlcdb

ETL문자 데이터베이스

http://etlcdb.db.aist.go.jp/?lang=ja

ETL8G

956 문자의 한자, 히라가나
손글씨 153916개 데이터

여러 손글씨가 포함된 사진과
그에 맞는 값 txt 파일로 구성

데이터 분리

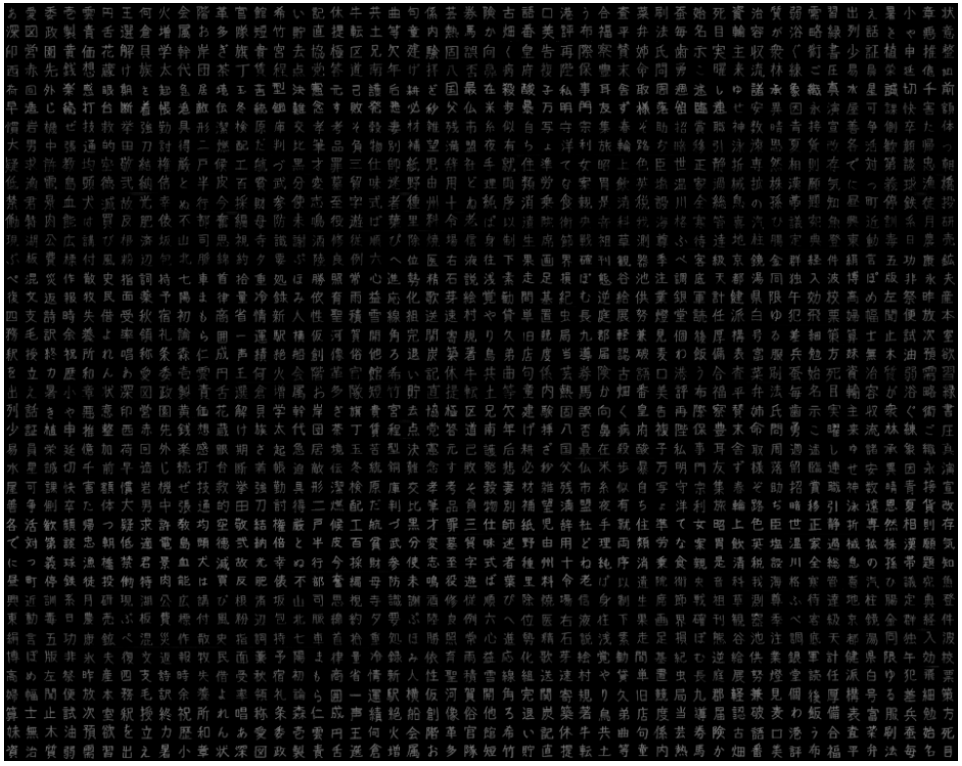
히라가나 데이터 제거 후
train : test
97791 : 44050
직접 분리하였음
Y는 히라가나 제거 후 범위 변경
(0~880)

데이터 추출

X = (153916, 127, 128)
Y = (153916,)
넘파이 배열로 추출
X는 127*128 문자 데이터
Y는 각 데이터를 구분하는 값
(0~955)

데이터 수집방안

파이썬 코드를 통해 데이터 추출, 분류 구현



etlcdb 데이터베이스 - ETL8G 데이터

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(97791, 127, 128)
(97791,)
(44050, 127, 128)
(44050,)
```

```
len(np.unique(y_test))
```

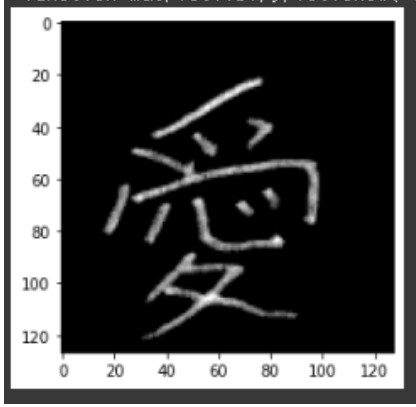
```
881
```

데이터 추출 및 분리

데이터 확인

```
first_image = x_train[0]
first_image = np.array(first_image, dtype = 'float')
#pixels = first_image.reshape((28,28))
plt.imshow(first_image, cmap = 'gray')
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```



적용 기법 및 모델

CNN 모델 코드 및 모델 설계

파라미터	값
num_classes	881
epochs	20
batch_size	128
learning_rate	0.001

```
with tf.device('/device:GPU:0'):
  model = tf.keras.Sequential()

  model.add(layers.Conv2D(16, (3, 3), activation='relu', input_shape = input_shape))
  model.add(layers.Conv2D(32, (3, 3), activation='relu'))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(Dropout(0.25))

  model.add(layers.Conv2D(64, (3, 3), activation='relu'))
  model.add(layers.MaxPooling2D((2, 2)))
  model.add(Dropout(0.25))

  model.add(layers.Conv2D(128, (3, 3), activation='relu'))
  model.add(Dropout(0.25))

  model.add(layers.Flatten())

  model.add(layers.Dense(2048, activation = 'relu'))
  model.add(Dropout(0.25))

  model.add(layers.Dense(1024, activation = 'relu'))

  model.add(layers.Dense(num_classes, activation = 'softmax'))

  model.compile(loss = tf.keras.losses.categorical_crossentropy,
                optimizer = tf.keras.optimizers.Adam(learning_rate = learning_rate),
                metrics = ['accuracy'])
  early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience = 5)
  hist = model.fit(x_train, y_train,
                   batch_size = batch_size, epochs = epochs,
                   validation_split = 0.2, callbacks = [early_stop]
                   )

  score = model.evaluate(x_test, y_test, verbose=1)

  print(' - test_loss:', score[0], ' - test_acc:', score[1])
  #model.save("cnn_japanese.h5")
```

적용 기법 및 모델

학습시간, 메모리 문제와 해결방안

문제점

- 학습 데이터 크기가 큼
  모델의 파라미터가 많음
- Colab에서
  메모리 문제로
  진행할 수 없음
- 많은 학습 시간 소요
  (한 epochs 당 5분)

해결방안

- 정규화를 진행하지 않고
  모델 학습 진행
- 작은 feature map에서
  커지는 방향으로 학습 진행
- 한 epochs 당
  1분 30초 정도로 학습 시간 단축

X 데이터의 정규화과정에서
많은 데이터 소모 확인

적용 기법 및 모델

CNN 모델 코드 및 모델 설계

기존 학습에 계속 발생하던
Overfitting 문제를
Dropout을 통해 해결

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 125, 126, 16)      160

 conv2d_1 (Conv2D)           (None, 123, 124, 32)      4640

 max_pooling2d (MaxPooling2D  (None, 61, 62, 32)       0
 )

 dropout (Dropout)           (None, 61, 62, 32)        0

 conv2d_2 (Conv2D)           (None, 59, 60, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 29, 30, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 29, 30, 64)        0

 conv2d_3 (Conv2D)           (None, 27, 28, 128)       73856

 dropout_2 (Dropout)         (None, 27, 28, 128)       0

 flatten (Flatten)           (None, 96768)             0

 dense (Dense)               (None, 2048)              198182912

 dropout_3 (Dropout)         (None, 2048)              0

 dense_1 (Dense)             (None, 1024)              2098176

 dense_2 (Dense)             (None, 881)               903025

=================================================================
Total params: 201,281,265
Trainable params: 201,281,265
Non-trainable params: 0
_________________________________________________________________
```

적용 기법 및 모델

CNN 모델 학습 과정

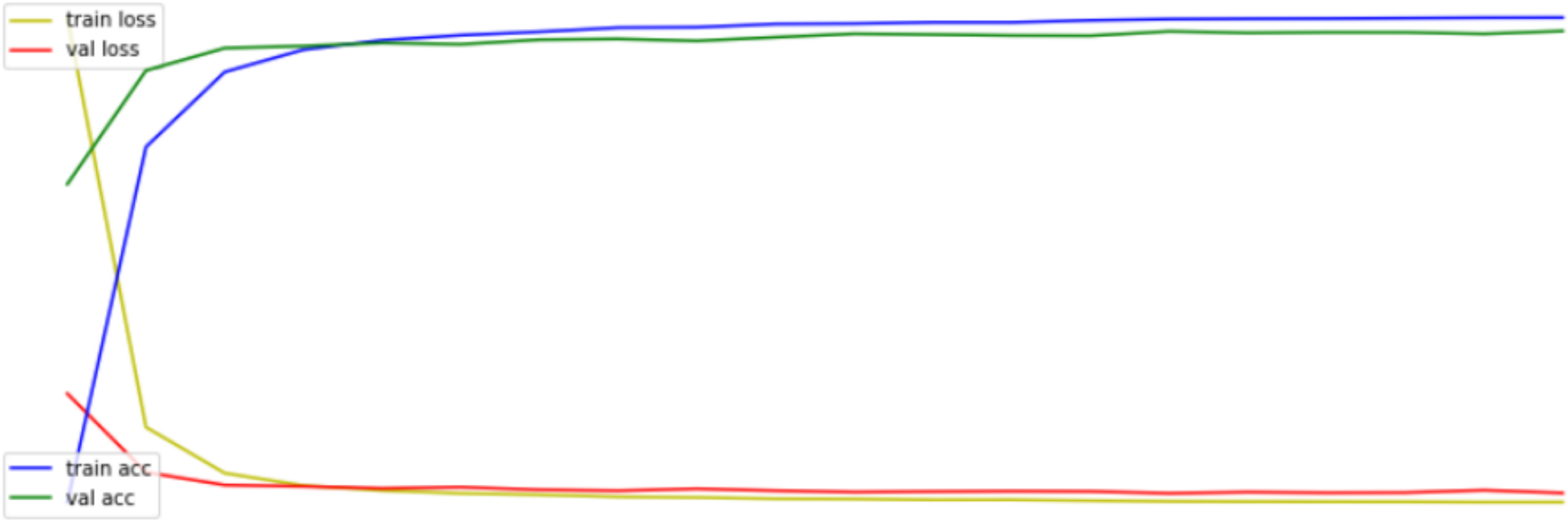
```
Epoch 1/20
612/612 [==============================] - 105s 157ms/step - loss: 4.9755 - accuracy: 0.1855 - val_loss: 1.1545 - val_accuracy: 0.7097
Epoch 2/20
612/612 [==============================] - 102s 167ms/step - loss: 0.8134 - accuracy: 0.7712 - val_loss: 0.3547 - val_accuracy: 0.8971
Epoch 3/20
612/612 [==============================] - 103s 169ms/step - loss: 0.3461 - accuracy: 0.8947 - val_loss: 0.2252 - val_accuracy: 0.9341
Epoch 4/20
612/612 [==============================] - 99s 162ms/step - loss: 0.2204 - accuracy: 0.9314 - val_loss: 0.2115 - val_accuracy: 0.9374
Epoch 5/20
612/612 [==============================] - 99s 162ms/step - loss: 0.1693 - accuracy: 0.9470 - val_loss: 0.1916 - val_accuracy: 0.9425
Epoch 6/20
612/612 [==============================] - 104s 169ms/step - loss: 0.1410 - accuracy: 0.9553 - val_loss: 0.2033 - val_accuracy: 0.9403
Epoch 7/20
612/612 [==============================] - 100s 163ms/step - loss: 0.1265 - accuracy: 0.9610 - val_loss: 0.1769 - val_accuracy: 0.9479
Epoch 8/20
612/612 [==============================] - 99s 162ms/step - loss: 0.1063 - accuracy: 0.9677 - val_loss: 0.1673 - val_accuracy: 0.9491
Epoch 9/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0996 - accuracy: 0.9684 - val_loss: 0.1871 - val_accuracy: 0.9459
Epoch 10/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0863 - accuracy: 0.9738 - val_loss: 0.1676 - val_accuracy: 0.9519
Epoch 11/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0835 - accuracy: 0.9744 - val_loss: 0.1534 - val_accuracy: 0.9577
Epoch 12/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0750 - accuracy: 0.9765 - val_loss: 0.1587 - val_accuracy: 0.9564
Epoch 13/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0754 - accuracy: 0.9764 - val_loss: 0.1628 - val_accuracy: 0.9549
Epoch 14/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0667 - accuracy: 0.9798 - val_loss: 0.1600 - val_accuracy: 0.9544
Epoch 15/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0595 - accuracy: 0.9818 - val_loss: 0.1416 - val_accuracy: 0.9617
Epoch 16/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0571 - accuracy: 0.9824 - val_loss: 0.1532 - val_accuracy: 0.9592
Epoch 17/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0558 - accuracy: 0.9827 - val_loss: 0.1468 - val_accuracy: 0.9601
Epoch 18/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0549 - accuracy: 0.9834 - val_loss: 0.1486 - val_accuracy: 0.9600
Epoch 19/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0516 - accuracy: 0.9842 - val_loss: 0.1722 - val_accuracy: 0.9576
Epoch 20/20
612/612 [==============================] - 99s 162ms/step - loss: 0.0518 - accuracy: 0.9844 - val_loss: 0.1445 - val_accuracy: 0.9622
1377/1377 [==============================] - 17s 12ms/step - loss: 0.1867 - accuracy: 0.9555
 - test_loss: 0.1867275983095169  - test_acc: 0.955459713935852
```

test	값
test_loss	0.1867
test_accuracy	0.9555

cnn_japanese_96_19.h5
모델 저장

적용 기법 및 모델

CNN 모델 학습 결과



결과

모델에 사용할 데이터

japanese_use_firstname

- 長谷川
- 田中
- 山本
- 中村
- 小林
- 山口
- 森
- 山下
- 行武

長.JPG

谷.JPG

川.JPG

결과

모델 사용하기 (use 파일)

```
from tensorflow.python.keras.models import load_model

model = load_model("/content/gdrive/MyDrive/인공지능론/텀프/cnn_japanese_96_19.h5")
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 125, 126, 16)      160
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 123, 124, 32)      4640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 61, 62, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 61, 62, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 59, 60, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 29, 30, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 29, 30, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 27, 28, 128)       73856
_________________________________________________________________
dropout_2 (Dropout)          (None, 27, 28, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 96768)             0
_________________________________________________________________
dense (Dense)                (None, 2048)              198182912
_________________________________________________________________
dropout_3 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dense_2 (Dense)              (None, 881)               903025
=================================================================
Total params: 201,281,265
Trainable params: 201,281,265
Non-trainable params: 0
_________________________________________________________________
```

결과

데이터 예측하기 (use 파일)

file_names = os.listdir(subdir) 과정에서
파일 순서 변경되는 문제 발생
-> 폴더 이름과 인덱스 대조하여 해결

```
import os
dir = "/content/gdrive/MyDrive/인공지능론/텀프/japanese_use_firstname"
subdir_names = os.listdir(dir)
X_test = []
y_test = []
ignore_data = []
predict_data = []
#subdir_name = 폴더 단위
for subdir_name in subdir_names:
  count = 0
  subdir = dir +"/" + subdir_name
  predict_data_add = []
  #file names 순서 오류 해결
  file_names = os.listdir(subdir)
  file_names_sort = []
  for k in subdir_name:
    if len(file_names) != 0:
      index_sort = [i for i in range(len(file_names)) if k in file_names[i]]
      file_names_sort.append(file_names[index_sort[0]])
  #file_name = 파일단위
  for file_name in file_names_sort:
    if len(np.where(y_train_char ==  subdir_name[count])[0]) == 0:
      ignore_data.append(subdir_name[count])
    else:
      path = subdir +"/" + file_name
      print('filename:', file_name)
      arr = jpg_image_to_array(path, 128, 127)
      arr_1c = image_array_to_1channel(arr)
      print(arr_1c.shape)
      if len(X_test) == 0:
        X_test = [arr_1c]
        y_test = int(y_train[np.where(y_train_char ==  subdir_name[count])[0][0]])
        predict_data_add.append(subdir_name[count])
        print(subdir_name[count])
      else:
        X_test = np.concatenate((X_test, [arr_1c]))
        y_test = np.append(y_test, int(y_train[np.where(y_train_char ==  subdir_name[count])[0][0]]))
        predict_data_add.append(subdir_name[count])
        print(subdir_name[count])
    count += 1
  if predict_data_add != []:
    predict_data.append(predict_data_add)
```

결과

데이터 예측 결과 (use 파일)

```
y_test
```

```
array([608, 577, 515, 633, 593, 316, 788, 593, 558, 430, 861, 316, 260,
       459, 316,  57, 275, 744])
```

실제 값

```
y_pred = model.predict_classes(X_test)
y_pred
```

```
array([608, 577, 515, 581, 593, 316, 788, 593, 736, 430, 861, 316, 260,
       459, 316,  57, 736, 744])
```

예측 값

인덱스를
한자로 변경

```
# 실제 값
predict_data
```

```
[['長', '谷', '川'],
 ['田', '中'],
 ['山', '本'],
 ['中', '村'],
 ['小', '林'],
 ['山', '口'],
 ['森'],
 ['山', '下'],
 ['行', '武']]
```

```
#예측 값
y_pred_real
```

```
[array(['長', '谷', '川'], dtype='<U32'),
 array(['団', '中'], dtype='<U32'),
 array(['山', '本'], dtype='<U32'),
 array(['中', '付'], dtype='<U32'),
 array(['小', '林'], dtype='<U32'),
 array(['山', '口'], dtype='<U32'),
 array(['森'], dtype='<U32'),
 array(['山', '下'], dtype='<U32'),
 array(['付', '武'], dtype='<U32')]
```

결과

데이터 예측 평가 (use 파일)

```
from tensorflow.keras import utils
y_binary = utils.to_categorical(y_test, 881)
print(y_binary)

score = model.evaluate(X_test, y_binary)
print('test_loss: ', score[0])
print('test_ac: ', score[1])
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
1/1 [==============================] - 0s 418ms/step - loss: 1.3544 - accuracy: 0.8333
test_loss:  1.3544195890426636
test_ac:  0.8333333134651184
```

test	값
test_loss	1.3544
test_accuracy	0.8333

田 ----> 団
村 ----> 付
行 ----> 付

모양이 비슷한 다른 한자로 잘못된 예측이 발생함

결과

히라가나 변환 (use 파일)

일본 성씨 빈도 5000위 데이터

	name	reading	num	difficult
0	佐藤	さとう	1894000	0.96
1	鈴木	すずき	1809000	0.99
2	高橋	たかはし	1425000	1.20
3	田中	たなか|でんちゅう	1346000	0.98
4	伊藤	いとう	1084000	1.43
...	...	...	...	...
4995	治田	はるだ	2100	300.00
4996	岡上	おかうえ	2100	324.32
4997	韮沢	にらさわ	2100	501.90
4998	行武	ゆきたけ|ゆくたけ	2100	543.21
4999	井戸川	いどかわ	2100	6947.37

5000 rows × 4 columns

히라가나 변환 진행

```
for i in y_pred_real:
  string_join = ''.join(i)
  hiragana = df['reading'].values[df['name'] == string_join]
  if hiragana.size == 0:
    print(string_join, '=', "학습 결과가 올바르지 않습니다.")
  else:
    if '|' in hiragana[0]:
      index = hiragana[0].find('|')
      print(string_join, '=', hiragana[0][:index], '또는', hiragana[0][index+1:], '입니다.')
    else:
      print(string_join, '=', hiragana[0], '입니다.')
```

```
長谷川 = はせがわ 입니다.
団中 = 학습 결과가 올바르지 않습니다.
山本 = やまもと 입니다.
中付 = 학습 결과가 올바르지 않습니다.
小林 = こばやし 입니다.
山口 = やまぐち 입니다.
森 = もり 입니다.
山下 = やました 입니다.
付武 = 학습 결과가 올바르지 않습니다.
```

논의

1. 메모리 문제로 한정된 모델 내 학습 진행함
2. 정규화 진행하지 못해서 학습 향상에 어려움을 겪음
3. 한자 881개 데이터셋
   -> 실생활에 사용하기에는 부족함
4. 한글 번역까지 진행하고 싶었지만
   시간 관계 상 진행하지 못함
5. 휴리스틱으로 파라미터를
   설정하는 것에 어려움을 느낌

감사합니다