TP Final

Luciana Diaz Kralj
Agustina Sol Ortu
Paula Oseroff

4 de diciembre de 2024

Profesor: Daniel Jacoby

Introducción

En el idioma japonés, existen 3 abecedarios diferentes: hiragana, katakana y kanji.

El katakana y hiragana son considerados silabarios ya que la fonética de cada uno de los símbolos que los componen consta del sonido de una consonante y luego una vocal. Así, por ejemplo, el símbolo か se pronunca "ka".

En particular, el katakana suele ser utilizado para palabras extranjeras. Observando la palabra パソコン ("pasokon"), la misma está escrita en katakana, siendo un fonetismo de la palabra anglosajona "Personal Computer" acortada "Perso(nal) Comp". Por otro lado, los kanji son ni más ni menos que símbolos heredados del chino antigüo dotados de un significado en sí. Entonces, 卵 es el kanji para "huevo" para dar un ejemplo.

Finalmente, se encuentra el hiragana. Este nos permite saber la pronunciación de una palabra y es posible realizar un mapeo 1 a 1 con letras que generen los mismos sonidos que el español (al igual que como sucede con el katakana). En Japón, son utilizados para señales que todo el mundo debe poder identificar o para índicar la pronunciación de un kanji.

Como dato curioso, en el juego Pokémon, al iniciar una nueva partida, se puede seleccionar si se quiere que los dialogos se encuentren con una combinación de kanji y hiragana o que los mismos se encuentren únicamente con los hiragana correspondientes. Esto es porque los más pequeños no pueden leer kanji.

Bajo esta premisa, el siguiente trabajo busca identificar los hiragana que pueden aparecer en carteles y señales alrededor de Japón. No se busca una traducción ni un mapeo a nuestra fonética de los mismos, solo identificarlos y obtener el carácter unicode correspondiente. De todas maneras, el trabajo abre las puertas para tomar el resultado y realizar los experimentos mencionados.

Pipeline

El proceso de identificar los hiragana en un letrero se encuentra dividido en dos partes fundamentales: la separación de los hiragana que se encuentran en la imágen y la identificación en sí del kanji (predecir que kanji es cada uno).

```
import cv2
import keras
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
```

Modelo

Para el segundo apartado (la identificación del kanji), se entrenó un modelo de Red Convolucional a partir del dataset Kuzushiji-49 que contiene ejemplos de los 46 hiragana estándar y 3 hiragana antigüos manuscritos. Consta, en total, de 270.912 imágenes de 28x28 pixeles separadas en train y test.

```
batch_size = 128
num_classes = 49
epochs = 350

# input image dimensions
img_rows, img_cols = 28, 28

def load(f):
    return np.load(f)['arr_0']

# Load the data
x_train = load('k49-train-imgs.npz')
x_test = load('k49-test-imgs.npz')
y_train = load('k49-train-labels.npz')
y_test = load('k49-test-labels.npz')

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('{} train samples, {} test samples'.format(len(x_train),
len(x_test)))
```

```
232365 train samples, 38547 test samples
```

```
# Convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
```

```python
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
c:\Users\Usuario\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning:
Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Epoch 1/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 168s 92ms/step - accuracy: 0.0274 -
loss: 3.8830 - val_accuracy: 0.0766 - val_loss: 3.8312
Epoch 2/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 169s 93ms/step - accuracy: 0.0645 -
loss: 3.8052 - val_accuracy: 0.1164 - val_loss: 3.7331
Epoch 3/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 170s 94ms/step - accuracy: 0.1005 -
loss: 3.6881 - val_accuracy: 0.1625 - val_loss: 3.5785
Epoch 4/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 170s 94ms/step - accuracy: 0.1434 -
loss: 3.5165 - val_accuracy: 0.2137 - val_loss: 3.3836
Epoch 5/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 170s 94ms/step - accuracy: 0.1954 -
loss: 3.3180 - val_accuracy: 0.2584 - val_loss: 3.1889
Epoch 6/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 172s 94ms/step - accuracy: 0.2382 -
loss: 3.1313 - val_accuracy: 0.3002 - val_loss: 3.0201
Epoch 7/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 170s 94ms/step - accuracy: 0.2751 -
loss: 2.9697 - val_accuracy: 0.3402 - val_loss: 2.8784
Epoch 8/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 171s 94ms/step - accuracy: 0.3016 -
loss: 2.8448 - val_accuracy: 0.3688 - val_loss: 2.7655
```

```
Epoch 9/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 171s 94ms/step - accuracy: 0.3283 -
loss: 2.7348 - val_accuracy: 0.3869 - val_loss: 2.6743
Epoch 10/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 171s 94ms/step - accuracy: 0.3474 -
loss: 2.6518 - val_accuracy: 0.4013 - val_loss: 2.5951
Epoch 11/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 171s 94ms/step - accuracy: 0.3611 -
loss: 2.5816 - val_accuracy: 0.4131 - val_loss: 2.5299
Epoch 12/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 172s 94ms/step - accuracy: 0.3784 -
loss: 2.5076 - val_accuracy: 0.4217 - val_loss: 2.4726
Epoch 13/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 172s 94ms/step - accuracy: 0.3903 -
loss: 2.4524 - val_accuracy: 0.4314 - val_loss: 2.4200
Epoch 14/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 172s 94ms/step - accuracy: 0.4000 -
loss: 2.4102 - val_accuracy: 0.4393 - val_loss: 2.3752
Epoch 15/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4081 -
loss: 2.3729 - val_accuracy: 0.4475 - val_loss: 2.3343
Epoch 16/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 172s 95ms/step - accuracy: 0.4169 -
loss: 2.3308 - val_accuracy: 0.4552 - val_loss: 2.2960
Epoch 17/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 172s 95ms/step - accuracy: 0.4239 -
loss: 2.2976 - val_accuracy: 0.4607 - val_loss: 2.2627
Epoch 18/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4311 -
loss: 2.2682 - val_accuracy: 0.4669 - val_loss: 2.2322
Epoch 19/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.4366 -
loss: 2.2374 - val_accuracy: 0.4724 - val_loss: 2.2043
Epoch 20/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4442 -
loss: 2.2041 - val_accuracy: 0.4785 - val_loss: 2.1786
Epoch 21/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 172s 95ms/step - accuracy: 0.4504 -
loss: 2.1869 - val_accuracy: 0.4826 - val_loss: 2.1537
Epoch 22/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4574 -
loss: 2.1599 - val_accuracy: 0.4872 - val_loss: 2.1306
Epoch 23/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4605 -
loss: 2.1406 - val_accuracy: 0.4912 - val_loss: 2.1095
Epoch 24/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4655 -
loss: 2.1201 - val_accuracy: 0.4951 - val_loss: 2.0895
Epoch 25/350
```

```
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4704 -
loss: 2.0931 - val_accuracy: 0.4990 - val_loss: 2.0711
Epoch 26/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4716 -
loss: 2.0813 - val_accuracy: 0.5016 - val_loss: 2.0541
Epoch 27/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4773 -
loss: 2.0614 - val_accuracy: 0.5055 - val_loss: 2.0377
Epoch 28/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4815 -
loss: 2.0469 - val_accuracy: 0.5100 - val_loss: 2.0213
Epoch 29/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.4868 -
loss: 2.0297 - val_accuracy: 0.5130 - val_loss: 2.0047
Epoch 30/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4879 -
loss: 2.0202 - val_accuracy: 0.5160 - val_loss: 1.9902
Epoch 31/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4937 -
loss: 1.9997 - val_accuracy: 0.5189 - val_loss: 1.9768
Epoch 32/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.4968 -
loss: 1.9862 - val_accuracy: 0.5225 - val_loss: 1.9624
Epoch 33/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5000 -
loss: 1.9740 - val_accuracy: 0.5254 - val_loss: 1.9488
Epoch 34/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5009 -
loss: 1.9583 - val_accuracy: 0.5283 - val_loss: 1.9380
Epoch 35/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5074 -
loss: 1.9443 - val_accuracy: 0.5313 - val_loss: 1.9241
Epoch 36/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5095 -
loss: 1.9266 - val_accuracy: 0.5339 - val_loss: 1.9128
Epoch 37/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5120 -
loss: 1.9197 - val_accuracy: 0.5368 - val_loss: 1.9009
Epoch 38/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5149 -
loss: 1.9054 - val_accuracy: 0.5397 - val_loss: 1.8908
Epoch 39/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5180 -
loss: 1.9007 - val_accuracy: 0.5421 - val_loss: 1.8803
Epoch 40/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5206 -
loss: 1.8841 - val_accuracy: 0.5436 - val_loss: 1.8708
Epoch 41/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5214 -
```

```
loss: 1.8815 - val_accuracy: 0.5460 - val_loss: 1.8595
Epoch 42/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5250 -
loss: 1.8669 - val_accuracy: 0.5476 - val_loss: 1.8500
Epoch 43/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5281 -
loss: 1.8490 - val_accuracy: 0.5496 - val_loss: 1.8410
Epoch 44/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5326 -
loss: 1.8432 - val_accuracy: 0.5520 - val_loss: 1.8296
Epoch 45/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5338 -
loss: 1.8274 - val_accuracy: 0.5530 - val_loss: 1.8220
Epoch 46/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5349 -
loss: 1.8222 - val_accuracy: 0.5553 - val_loss: 1.8124
Epoch 47/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5371 -
loss: 1.8176 - val_accuracy: 0.5571 - val_loss: 1.8035
Epoch 48/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5387 -
loss: 1.8048 - val_accuracy: 0.5593 - val_loss: 1.7931
Epoch 49/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5416 -
loss: 1.7891 - val_accuracy: 0.5611 - val_loss: 1.7846
Epoch 50/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5447 -
loss: 1.7809 - val_accuracy: 0.5623 - val_loss: 1.7765
Epoch 51/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5465 -
loss: 1.7744 - val_accuracy: 0.5645 - val_loss: 1.7681
Epoch 52/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5491 -
loss: 1.7614 - val_accuracy: 0.5665 - val_loss: 1.7595
Epoch 53/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5523 -
loss: 1.7558 - val_accuracy: 0.5682 - val_loss: 1.7505
Epoch 54/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5565 -
loss: 1.7379 - val_accuracy: 0.5699 - val_loss: 1.7433
Epoch 55/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5586 -
loss: 1.7251 - val_accuracy: 0.5716 - val_loss: 1.7355
Epoch 56/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5571 -
loss: 1.7256 - val_accuracy: 0.5731 - val_loss: 1.7269
Epoch 57/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.5618 -
loss: 1.7099 - val_accuracy: 0.5756 - val_loss: 1.7186
```

```
Epoch 58/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5641 -
loss: 1.7008 - val_accuracy: 0.5767 - val_loss: 1.7111
Epoch 59/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5641 -
loss: 1.7015 - val_accuracy: 0.5783 - val_loss: 1.7043
Epoch 60/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5669 -
loss: 1.6911 - val_accuracy: 0.5801 - val_loss: 1.6961
Epoch 61/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 173s 95ms/step - accuracy: 0.5677 -
loss: 1.6833 - val_accuracy: 0.5821 - val_loss: 1.6885
Epoch 62/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5719 -
loss: 1.6756 - val_accuracy: 0.5838 - val_loss: 1.6804
Epoch 63/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5716 -
loss: 1.6723 - val_accuracy: 0.5851 - val_loss: 1.6737
Epoch 64/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.5751 -
loss: 1.6596 - val_accuracy: 0.5869 - val_loss: 1.6657
Epoch 65/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5759 -
loss: 1.6472 - val_accuracy: 0.5879 - val_loss: 1.6574
Epoch 66/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5775 -
loss: 1.6376 - val_accuracy: 0.5895 - val_loss: 1.6510
Epoch 67/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.5823 -
loss: 1.6272 - val_accuracy: 0.5920 - val_loss: 1.6434
Epoch 68/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5832 -
loss: 1.6239 - val_accuracy: 0.5929 - val_loss: 1.6365
Epoch 69/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.5856 -
loss: 1.6141 - val_accuracy: 0.5946 - val_loss: 1.6296
Epoch 70/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5862 -
loss: 1.6056 - val_accuracy: 0.5963 - val_loss: 1.6215
Epoch 71/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5911 -
loss: 1.5934 - val_accuracy: 0.5986 - val_loss: 1.6149
Epoch 72/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.5918 -
loss: 1.5871 - val_accuracy: 0.5995 - val_loss: 1.6074
Epoch 73/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.5932 -
loss: 1.5810 - val_accuracy: 0.6010 - val_loss: 1.6001
Epoch 74/350
```

```
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.5957 -
loss: 1.5711 - val_accuracy: 0.6028 - val_loss: 1.5928
Epoch 75/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.5963 -
loss: 1.5686 - val_accuracy: 0.6044 - val_loss: 1.5871
Epoch 76/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.5994 -
loss: 1.5593 - val_accuracy: 0.6060 - val_loss: 1.5793
Epoch 77/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6004 -
loss: 1.5523 - val_accuracy: 0.6074 - val_loss: 1.5723
Epoch 78/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.6034 -
loss: 1.5429 - val_accuracy: 0.6092 - val_loss: 1.5657
Epoch 79/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6050 -
loss: 1.5321 - val_accuracy: 0.6108 - val_loss: 1.5603
Epoch 80/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6078 -
loss: 1.5190 - val_accuracy: 0.6122 - val_loss: 1.5524
Epoch 81/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 97ms/step - accuracy: 0.6085 -
loss: 1.5204 - val_accuracy: 0.6140 - val_loss: 1.5453
Epoch 82/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6084 -
loss: 1.5159 - val_accuracy: 0.6150 - val_loss: 1.5392
Epoch 83/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6123 -
loss: 1.5034 - val_accuracy: 0.6172 - val_loss: 1.5325
Epoch 84/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6153 -
loss: 1.4946 - val_accuracy: 0.6189 - val_loss: 1.5249
Epoch 85/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 174s 96ms/step - accuracy: 0.6152 -
loss: 1.4844 - val_accuracy: 0.6202 - val_loss: 1.5194
Epoch 86/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6163 -
loss: 1.4858 - val_accuracy: 0.6216 - val_loss: 1.5128
Epoch 87/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6206 -
loss: 1.4756 - val_accuracy: 0.6238 - val_loss: 1.5068
Epoch 88/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6177 -
loss: 1.4709 - val_accuracy: 0.6254 - val_loss: 1.4990
Epoch 89/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6224 -
loss: 1.4586 - val_accuracy: 0.6270 - val_loss: 1.4929
Epoch 90/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6235 -
```

```
loss: 1.4525 - val_accuracy: 0.6282 - val_loss: 1.4866
Epoch 91/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6276 -
loss: 1.4457 - val_accuracy: 0.6298 - val_loss: 1.4800
Epoch 92/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6290 -
loss: 1.4339 - val_accuracy: 0.6317 - val_loss: 1.4744
Epoch 93/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6315 -
loss: 1.4257 - val_accuracy: 0.6327 - val_loss: 1.4672
Epoch 94/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 178s 98ms/step - accuracy: 0.6311 -
loss: 1.4291 - val_accuracy: 0.6350 - val_loss: 1.4597
Epoch 95/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 181s 100ms/step - accuracy: 0.6339 -
loss: 1.4163 - val_accuracy: 0.6361 - val_loss: 1.4545
Epoch 96/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 183s 101ms/step - accuracy: 0.6385 -
loss: 1.4043 - val_accuracy: 0.6376 - val_loss: 1.4492
Epoch 97/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 185s 102ms/step - accuracy: 0.6345 -
loss: 1.4049 - val_accuracy: 0.6393 - val_loss: 1.4425
Epoch 98/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 184s 102ms/step - accuracy: 0.6369 -
loss: 1.3995 - val_accuracy: 0.6405 - val_loss: 1.4371
Epoch 99/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 182s 100ms/step - accuracy: 0.6398 -
loss: 1.3910 - val_accuracy: 0.6420 - val_loss: 1.4295
Epoch 100/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6437 -
loss: 1.3829 - val_accuracy: 0.6435 - val_loss: 1.4244
Epoch 101/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6438 -
loss: 1.3780 - val_accuracy: 0.6447 - val_loss: 1.4193
Epoch 102/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6450 -
loss: 1.3659 - val_accuracy: 0.6465 - val_loss: 1.4121
Epoch 103/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6464 -
loss: 1.3641 - val_accuracy: 0.6472 - val_loss: 1.4068
Epoch 104/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6466 -
loss: 1.3592 - val_accuracy: 0.6488 - val_loss: 1.4011
Epoch 105/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6508 -
loss: 1.3442 - val_accuracy: 0.6511 - val_loss: 1.3948
Epoch 106/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6514 -
loss: 1.3450 - val_accuracy: 0.6518 - val_loss: 1.3896
```

```
Epoch 107/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6522 -
loss: 1.3385 - val_accuracy: 0.6543 - val_loss: 1.3839
Epoch 108/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6546 -
loss: 1.3277 - val_accuracy: 0.6552 - val_loss: 1.3778
Epoch 109/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 97ms/step - accuracy: 0.6568 -
loss: 1.3202 - val_accuracy: 0.6564 - val_loss: 1.3722
Epoch 110/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6578 -
loss: 1.3174 - val_accuracy: 0.6577 - val_loss: 1.3656
Epoch 111/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6610 -
loss: 1.3093 - val_accuracy: 0.6590 - val_loss: 1.3606
Epoch 112/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6600 -
loss: 1.3065 - val_accuracy: 0.6599 - val_loss: 1.3559
Epoch 113/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6622 -
loss: 1.3002 - val_accuracy: 0.6613 - val_loss: 1.3500
Epoch 114/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6640 -
loss: 1.2910 - val_accuracy: 0.6621 - val_loss: 1.3446
Epoch 115/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6643 -
loss: 1.2857 - val_accuracy: 0.6638 - val_loss: 1.3401
Epoch 116/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 96ms/step - accuracy: 0.6667 -
loss: 1.2803 - val_accuracy: 0.6648 - val_loss: 1.3346
Epoch 117/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 97ms/step - accuracy: 0.6670 -
loss: 1.2775 - val_accuracy: 0.6662 - val_loss: 1.3292
Epoch 118/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 97ms/step - accuracy: 0.6701 -
loss: 1.2692 - val_accuracy: 0.6678 - val_loss: 1.3233
Epoch 119/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 97ms/step - accuracy: 0.6704 -
loss: 1.2688 - val_accuracy: 0.6686 - val_loss: 1.3198
Epoch 120/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6718 -
loss: 1.2624 - val_accuracy: 0.6694 - val_loss: 1.3150
Epoch 121/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 97ms/step - accuracy: 0.6735 -
loss: 1.2542 - val_accuracy: 0.6706 - val_loss: 1.3089
Epoch 122/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6726 -
loss: 1.2575 - val_accuracy: 0.6721 - val_loss: 1.3044
Epoch 123/350
```

```
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6760 -
loss: 1.2436 - val_accuracy: 0.6731 - val_loss: 1.2992
Epoch 124/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6763 -
loss: 1.2393 - val_accuracy: 0.6740 - val_loss: 1.2946
Epoch 125/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6776 -
loss: 1.2320 - val_accuracy: 0.6752 - val_loss: 1.2903
Epoch 126/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6781 -
loss: 1.2368 - val_accuracy: 0.6765 - val_loss: 1.2851
Epoch 127/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6811 -
loss: 1.2223 - val_accuracy: 0.6778 - val_loss: 1.2812
Epoch 128/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6805 -
loss: 1.2238 - val_accuracy: 0.6792 - val_loss: 1.2764
Epoch 129/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6841 -
loss: 1.2135 - val_accuracy: 0.6798 - val_loss: 1.2721
Epoch 130/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6849 -
loss: 1.2123 - val_accuracy: 0.6806 - val_loss: 1.2685
Epoch 131/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6863 -
loss: 1.2047 - val_accuracy: 0.6816 - val_loss: 1.2633
Epoch 132/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6866 -
loss: 1.1982 - val_accuracy: 0.6831 - val_loss: 1.2581
Epoch 133/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 175s 97ms/step - accuracy: 0.6870 -
loss: 1.1978 - val_accuracy: 0.6842 - val_loss: 1.2543
Epoch 134/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6890 -
loss: 1.1938 - val_accuracy: 0.6851 - val_loss: 1.2505
Epoch 135/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6884 -
loss: 1.1890 - val_accuracy: 0.6864 - val_loss: 1.2458
Epoch 136/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6883 -
loss: 1.1887 - val_accuracy: 0.6870 - val_loss: 1.2422
Epoch 137/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6904 -
loss: 1.1821 - val_accuracy: 0.6886 - val_loss: 1.2374
Epoch 138/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6939 -
loss: 1.1712 - val_accuracy: 0.6892 - val_loss: 1.2343
Epoch 139/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6940 -
```

```
loss: 1.1662 - val_accuracy: 0.6904 - val_loss: 1.2302
Epoch 140/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6942 -
loss: 1.1670 - val_accuracy: 0.6915 - val_loss: 1.2261
Epoch 141/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6969 -
loss: 1.1609 - val_accuracy: 0.6921 - val_loss: 1.2217
Epoch 142/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6962 -
loss: 1.1616 - val_accuracy: 0.6928 - val_loss: 1.2187
Epoch 143/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6979 -
loss: 1.1571 - val_accuracy: 0.6943 - val_loss: 1.2146
Epoch 144/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.6991 -
loss: 1.1486 - val_accuracy: 0.6952 - val_loss: 1.2104
Epoch 145/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7006 -
loss: 1.1438 - val_accuracy: 0.6958 - val_loss: 1.2064
Epoch 146/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 178s 98ms/step - accuracy: 0.7015 -
loss: 1.1455 - val_accuracy: 0.6973 - val_loss: 1.2021
Epoch 147/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7024 -
loss: 1.1380 - val_accuracy: 0.6979 - val_loss: 1.1987
Epoch 148/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7046 -
loss: 1.1333 - val_accuracy: 0.6985 - val_loss: 1.1949
Epoch 149/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7037 -
loss: 1.1295 - val_accuracy: 0.6994 - val_loss: 1.1922
Epoch 150/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7030 -
loss: 1.1285 - val_accuracy: 0.7003 - val_loss: 1.1886
Epoch 151/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7048 -
loss: 1.1206 - val_accuracy: 0.7011 - val_loss: 1.1846
Epoch 152/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 201s 96ms/step - accuracy: 0.7063 -
loss: 1.1232 - val_accuracy: 0.7015 - val_loss: 1.1813
Epoch 153/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 178s 98ms/step - accuracy: 0.7070 -
loss: 1.1140 - val_accuracy: 0.7030 - val_loss: 1.1777
Epoch 154/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 179s 98ms/step - accuracy: 0.7076 -
loss: 1.1132 - val_accuracy: 0.7037 - val_loss: 1.1741
Epoch 155/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7091 -
loss: 1.1083 - val_accuracy: 0.7039 - val_loss: 1.1709
```

```
Epoch 156/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7114 -
loss: 1.1014 - val_accuracy: 0.7051 - val_loss: 1.1676
Epoch 157/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7122 -
loss: 1.0975 - val_accuracy: 0.7057 - val_loss: 1.1639
Epoch 158/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7127 -
loss: 1.0969 - val_accuracy: 0.7065 - val_loss: 1.1615
Epoch 159/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7136 -
loss: 1.0935 - val_accuracy: 0.7076 - val_loss: 1.1578
Epoch 160/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7135 -
loss: 1.0847 - val_accuracy: 0.7076 - val_loss: 1.1554
Epoch 161/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7125 -
loss: 1.0919 - val_accuracy: 0.7090 - val_loss: 1.1514
Epoch 162/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7163 -
loss: 1.0795 - val_accuracy: 0.7102 - val_loss: 1.1482
Epoch 163/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7170 -
loss: 1.0785 - val_accuracy: 0.7102 - val_loss: 1.1456
Epoch 164/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7157 -
loss: 1.0762 - val_accuracy: 0.7117 - val_loss: 1.1420
Epoch 165/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7182 -
loss: 1.0775 - val_accuracy: 0.7116 - val_loss: 1.1400
Epoch 166/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7199 -
loss: 1.0678 - val_accuracy: 0.7124 - val_loss: 1.1367
Epoch 167/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 98ms/step - accuracy: 0.7190 -
loss: 1.0682 - val_accuracy: 0.7132 - val_loss: 1.1335
Epoch 168/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7190 -
loss: 1.0692 - val_accuracy: 0.7139 - val_loss: 1.1299
Epoch 169/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7210 -
loss: 1.0618 - val_accuracy: 0.7146 - val_loss: 1.1273
Epoch 170/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7201 -
loss: 1.0694 - val_accuracy: 0.7150 - val_loss: 1.1243
Epoch 171/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7224 -
loss: 1.0515 - val_accuracy: 0.7158 - val_loss: 1.1214
Epoch 172/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7245 -
```

```
loss: 1.0467 - val_accuracy: 0.7162 - val_loss: 1.1189
Epoch 173/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7233 -
loss: 1.0519 - val_accuracy: 0.7172 - val_loss: 1.1157
Epoch 174/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7253 -
loss: 1.0455 - val_accuracy: 0.7178 - val_loss: 1.1135
Epoch 175/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7255 -
loss: 1.0467 - val_accuracy: 0.7191 - val_loss: 1.1099
Epoch 176/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 98ms/step - accuracy: 0.7263 -
loss: 1.0413 - val_accuracy: 0.7191 - val_loss: 1.1076
Epoch 177/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7268 -
loss: 1.0383 - val_accuracy: 0.7199 - val_loss: 1.1048
Epoch 178/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7259 -
loss: 1.0394 - val_accuracy: 0.7204 - val_loss: 1.1024
Epoch 179/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7276 -
loss: 1.0335 - val_accuracy: 0.7214 - val_loss: 1.0994
Epoch 180/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7281 -
loss: 1.0306 - val_accuracy: 0.7215 - val_loss: 1.0971
Epoch 181/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7288 -
loss: 1.0271 - val_accuracy: 0.7225 - val_loss: 1.0945
Epoch 182/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 98ms/step - accuracy: 0.7271 -
loss: 1.0351 - val_accuracy: 0.7229 - val_loss: 1.0926
Epoch 183/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7299 -
loss: 1.0282 - val_accuracy: 0.7234 - val_loss: 1.0892
Epoch 184/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7292 -
loss: 1.0267 - val_accuracy: 0.7244 - val_loss: 1.0869
Epoch 185/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7308 -
loss: 1.0200 - val_accuracy: 0.7254 - val_loss: 1.0843
Epoch 186/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 98ms/step - accuracy: 0.7339 -
loss: 1.0140 - val_accuracy: 0.7253 - val_loss: 1.0816
Epoch 187/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 180s 99ms/step - accuracy: 0.7328 -
loss: 1.0130 - val_accuracy: 0.7255 - val_loss: 1.0795
Epoch 188/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 184s 102ms/step - accuracy: 0.7333 -
loss: 1.0112 - val_accuracy: 0.7262 - val_loss: 1.0774
```

```
Epoch 189/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 98ms/step - accuracy: 0.7336 -
loss: 1.0119 - val_accuracy: 0.7269 - val_loss: 1.0743
Epoch 190/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 98ms/step - accuracy: 0.7356 -
loss: 1.0018 - val_accuracy: 0.7270 - val_loss: 1.0724
Epoch 191/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 178s 98ms/step - accuracy: 0.7348 -
loss: 1.0039 - val_accuracy: 0.7277 - val_loss: 1.0692
Epoch 192/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 98ms/step - accuracy: 0.7369 -
loss: 0.9958 - val_accuracy: 0.7282 - val_loss: 1.0671
Epoch 193/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 178s 98ms/step - accuracy: 0.7351 -
loss: 1.0049 - val_accuracy: 0.7282 - val_loss: 1.0657
Epoch 194/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7369 -
loss: 0.9966 - val_accuracy: 0.7294 - val_loss: 1.0626
Epoch 195/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7378 -
loss: 0.9942 - val_accuracy: 0.7297 - val_loss: 1.0608
Epoch 196/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 98ms/step - accuracy: 0.7373 -
loss: 0.9927 - val_accuracy: 0.7312 - val_loss: 1.0579
Epoch 197/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 178s 98ms/step - accuracy: 0.7399 -
loss: 0.9906 - val_accuracy: 0.7309 - val_loss: 1.0557
Epoch 198/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 178s 98ms/step - accuracy: 0.7398 -
loss: 0.9845 - val_accuracy: 0.7318 - val_loss: 1.0538
Epoch 199/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 98ms/step - accuracy: 0.7401 -
loss: 0.9796 - val_accuracy: 0.7324 - val_loss: 1.0508
Epoch 200/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 98ms/step - accuracy: 0.7405 -
loss: 0.9826 - val_accuracy: 0.7327 - val_loss: 1.0490
Epoch 201/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 178s 98ms/step - accuracy: 0.7423 -
loss: 0.9766 - val_accuracy: 0.7332 - val_loss: 1.0468
Epoch 202/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 178s 98ms/step - accuracy: 0.7425 -
loss: 0.9754 - val_accuracy: 0.7340 - val_loss: 1.0449
Epoch 203/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 187s 103ms/step - accuracy: 0.7416 -
loss: 0.9752 - val_accuracy: 0.7340 - val_loss: 1.0434
Epoch 204/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 210s 115ms/step - accuracy: 0.7410 -
loss: 0.9763 - val_accuracy: 0.7348 - val_loss: 1.0402
Epoch 205/350
```

```
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 207s 114ms/step - accuracy: 0.7428 -
loss: 0.9708 - val_accuracy: 0.7356 - val_loss: 1.0384
Epoch 206/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 198s 109ms/step - accuracy: 0.7429 -
loss: 0.9702 - val_accuracy: 0.7357 - val_loss: 1.0368
Epoch 207/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 194s 107ms/step - accuracy: 0.7438 -
loss: 0.9687 - val_accuracy: 0.7363 - val_loss: 1.0343
Epoch 208/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 187s 103ms/step - accuracy: 0.7472 -
loss: 0.9558 - val_accuracy: 0.7367 - val_loss: 1.0322
Epoch 209/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 189s 104ms/step - accuracy: 0.7454 -
loss: 0.9635 - val_accuracy: 0.7374 - val_loss: 1.0298
Epoch 210/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 188s 103ms/step - accuracy: 0.7453 -
loss: 0.9652 - val_accuracy: 0.7380 - val_loss: 1.0275
Epoch 211/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 186s 102ms/step - accuracy: 0.7456 -
loss: 0.9560 - val_accuracy: 0.7382 - val_loss: 1.0264
Epoch 212/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 186s 102ms/step - accuracy: 0.7452 -
loss: 0.9636 - val_accuracy: 0.7381 - val_loss: 1.0244
Epoch 213/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 186s 102ms/step - accuracy: 0.7478 -
loss: 0.9556 - val_accuracy: 0.7388 - val_loss: 1.0222
Epoch 214/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 203s 112ms/step - accuracy: 0.7457 -
loss: 0.9582 - val_accuracy: 0.7395 - val_loss: 1.0201
Epoch 215/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 229s 126ms/step - accuracy: 0.7471 -
loss: 0.9502 - val_accuracy: 0.7400 - val_loss: 1.0185
Epoch 216/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 242s 133ms/step - accuracy: 0.7495 -
loss: 0.9523 - val_accuracy: 0.7405 - val_loss: 1.0167
Epoch 217/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 236s 130ms/step - accuracy: 0.7498 -
loss: 0.9516 - val_accuracy: 0.7408 - val_loss: 1.0146
Epoch 218/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 238s 131ms/step - accuracy: 0.7508 -
loss: 0.9400 - val_accuracy: 0.7409 - val_loss: 1.0129
Epoch 219/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 244s 134ms/step - accuracy: 0.7501 -
loss: 0.9423 - val_accuracy: 0.7414 - val_loss: 1.0111
Epoch 220/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 239s 131ms/step - accuracy: 0.7493 -
loss: 0.9464 - val_accuracy: 0.7422 - val_loss: 1.0089
Epoch 221/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 239s 132ms/step - accuracy: 0.7525 -
```

```
loss: 0.9389 - val_accuracy: 0.7425 - val_loss: 1.0068
Epoch 222/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 239s 132ms/step - accuracy: 0.7505 -
loss: 0.9400 - val_accuracy: 0.7426 - val_loss: 1.0053
Epoch 223/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 263s 145ms/step - accuracy: 0.7516 -
loss: 0.9364 - val_accuracy: 0.7434 - val_loss: 1.0030
Epoch 224/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 318s 142ms/step - accuracy: 0.7522 -
loss: 0.9303 - val_accuracy: 0.7439 - val_loss: 1.0008
Epoch 225/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 263s 145ms/step - accuracy: 0.7530 -
loss: 0.9291 - val_accuracy: 0.7443 - val_loss: 0.9991
Epoch 226/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 260s 143ms/step - accuracy: 0.7535 -
loss: 0.9278 - val_accuracy: 0.7446 - val_loss: 0.9977
Epoch 227/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 243s 134ms/step - accuracy: 0.7517 -
loss: 0.9296 - val_accuracy: 0.7451 - val_loss: 0.9963
Epoch 228/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 249s 137ms/step - accuracy: 0.7532 -
loss: 0.9291 - val_accuracy: 0.7456 - val_loss: 0.9943
Epoch 229/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 248s 137ms/step - accuracy: 0.7541 -
loss: 0.9237 - val_accuracy: 0.7457 - val_loss: 0.9925
Epoch 230/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 241s 133ms/step - accuracy: 0.7559 -
loss: 0.9199 - val_accuracy: 0.7459 - val_loss: 0.9907
Epoch 231/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 242s 133ms/step - accuracy: 0.7573 -
loss: 0.9135 - val_accuracy: 0.7464 - val_loss: 0.9896
Epoch 232/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 261s 143ms/step - accuracy: 0.7558 -
loss: 0.9211 - val_accuracy: 0.7469 - val_loss: 0.9875
Epoch 233/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 257s 142ms/step - accuracy: 0.7568 -
loss: 0.9139 - val_accuracy: 0.7473 - val_loss: 0.9860
Epoch 234/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 285s 157ms/step - accuracy: 0.7581 -
loss: 0.9153 - val_accuracy: 0.7477 - val_loss: 0.9840
Epoch 235/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 289s 159ms/step - accuracy: 0.7575 -
loss: 0.9159 - val_accuracy: 0.7484 - val_loss: 0.9825
Epoch 236/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 282s 155ms/step - accuracy: 0.7574 -
loss: 0.9107 - val_accuracy: 0.7487 - val_loss: 0.9810
Epoch 237/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 262s 144ms/step - accuracy: 0.7588 -
loss: 0.9101 - val_accuracy: 0.7489 - val_loss: 0.9795
```

```
Epoch 238/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 244s 134ms/step - accuracy: 0.7565 -
loss: 0.9128 - val_accuracy: 0.7494 - val_loss: 0.9772
Epoch 239/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 232s 128ms/step - accuracy: 0.7605 -
loss: 0.9026 - val_accuracy: 0.7501 - val_loss: 0.9753
Epoch 240/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 273s 150ms/step - accuracy: 0.7592 -
loss: 0.9109 - val_accuracy: 0.7499 - val_loss: 0.9746
Epoch 241/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 268s 148ms/step - accuracy: 0.7604 -
loss: 0.9044 - val_accuracy: 0.7507 - val_loss: 0.9726
Epoch 242/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 265s 146ms/step - accuracy: 0.7615 -
loss: 0.9025 - val_accuracy: 0.7506 - val_loss: 0.9710
Epoch 243/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 228s 125ms/step - accuracy: 0.7610 -
loss: 0.8987 - val_accuracy: 0.7516 - val_loss: 0.9693
Epoch 244/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 189s 104ms/step - accuracy: 0.7608 -
loss: 0.9011 - val_accuracy: 0.7517 - val_loss: 0.9676
Epoch 245/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 195s 108ms/step - accuracy: 0.7621 -
loss: 0.8965 - val_accuracy: 0.7521 - val_loss: 0.9661
Epoch 246/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 196s 108ms/step - accuracy: 0.7623 -
loss: 0.8969 - val_accuracy: 0.7525 - val_loss: 0.9645
Epoch 247/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 199s 110ms/step - accuracy: 0.7630 -
loss: 0.8920 - val_accuracy: 0.7528 - val_loss: 0.9632
Epoch 248/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 198s 109ms/step - accuracy: 0.7620 -
loss: 0.8925 - val_accuracy: 0.7532 - val_loss: 0.9617
Epoch 249/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 197s 108ms/step - accuracy: 0.7659 -
loss: 0.8872 - val_accuracy: 0.7534 - val_loss: 0.9602
Epoch 250/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 195s 107ms/step - accuracy: 0.7622 -
loss: 0.8938 - val_accuracy: 0.7539 - val_loss: 0.9586
Epoch 251/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 196s 108ms/step - accuracy: 0.7641 -
loss: 0.8856 - val_accuracy: 0.7543 - val_loss: 0.9574
Epoch 252/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 195s 108ms/step - accuracy: 0.7657 -
loss: 0.8856 - val_accuracy: 0.7546 - val_loss: 0.9556
Epoch 253/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 198s 109ms/step - accuracy: 0.7659 -
loss: 0.8842 - val_accuracy: 0.7551 - val_loss: 0.9540
Epoch 254/350
```

```
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 195s 108ms/step - accuracy: 0.7669 -
loss: 0.8787 - val_accuracy: 0.7552 - val_loss: 0.9524
Epoch 255/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 204s 113ms/step - accuracy: 0.7671 -
loss: 0.8793 - val_accuracy: 0.7559 - val_loss: 0.9514
Epoch 256/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 226s 124ms/step - accuracy: 0.7651 -
loss: 0.8831 - val_accuracy: 0.7560 - val_loss: 0.9496
Epoch 257/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 217s 119ms/step - accuracy: 0.7674 -
loss: 0.8730 - val_accuracy: 0.7562 - val_loss: 0.9477
Epoch 258/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 213s 117ms/step - accuracy: 0.7667 -
loss: 0.8770 - val_accuracy: 0.7563 - val_loss: 0.9463
Epoch 259/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 211s 116ms/step - accuracy: 0.7682 -
loss: 0.8741 - val_accuracy: 0.7568 - val_loss: 0.9452
Epoch 260/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 217s 119ms/step - accuracy: 0.7679 -
loss: 0.8755 - val_accuracy: 0.7568 - val_loss: 0.9441
Epoch 261/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 260s 143ms/step - accuracy: 0.7679 -
loss: 0.8704 - val_accuracy: 0.7576 - val_loss: 0.9428
Epoch 262/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 286s 157ms/step - accuracy: 0.7692 -
loss: 0.8729 - val_accuracy: 0.7581 - val_loss: 0.9412
Epoch 263/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 258s 142ms/step - accuracy: 0.7676 -
loss: 0.8749 - val_accuracy: 0.7581 - val_loss: 0.9401
Epoch 264/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 244s 134ms/step - accuracy: 0.7666 -
loss: 0.8735 - val_accuracy: 0.7585 - val_loss: 0.9384
Epoch 265/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 242s 133ms/step - accuracy: 0.7686 -
loss: 0.8712 - val_accuracy: 0.7586 - val_loss: 0.9368
Epoch 266/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 244s 134ms/step - accuracy: 0.7684 -
loss: 0.8699 - val_accuracy: 0.7588 - val_loss: 0.9363
Epoch 267/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 243s 134ms/step - accuracy: 0.7699 -
loss: 0.8645 - val_accuracy: 0.7597 - val_loss: 0.9344
Epoch 268/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 249s 137ms/step - accuracy: 0.7717 -
loss: 0.8582 - val_accuracy: 0.7599 - val_loss: 0.9329
Epoch 269/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 243s 134ms/step - accuracy: 0.7708 -
loss: 0.8656 - val_accuracy: 0.7599 - val_loss: 0.9319
Epoch 270/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 225s 124ms/step - accuracy: 0.7713 -
```

```
loss: 0.8598 - val_accuracy: 0.7604 - val_loss: 0.9306
Epoch 271/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 233s 128ms/step - accuracy: 0.7709 -
loss: 0.8578 - val_accuracy: 0.7610 - val_loss: 0.9293
Epoch 272/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 258s 142ms/step - accuracy: 0.7722 -
loss: 0.8555 - val_accuracy: 0.7613 - val_loss: 0.9278
Epoch 273/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 218s 120ms/step - accuracy: 0.7726 -
loss: 0.8522 - val_accuracy: 0.7620 - val_loss: 0.9266
Epoch 274/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 186s 102ms/step - accuracy: 0.7719 -
loss: 0.8599 - val_accuracy: 0.7619 - val_loss: 0.9255
Epoch 275/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 194s 107ms/step - accuracy: 0.7732 -
loss: 0.8512 - val_accuracy: 0.7624 - val_loss: 0.9234
Epoch 276/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 182s 100ms/step - accuracy: 0.7733 -
loss: 0.8507 - val_accuracy: 0.7625 - val_loss: 0.9232
Epoch 277/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 179s 99ms/step - accuracy: 0.7729 -
loss: 0.8546 - val_accuracy: 0.7630 - val_loss: 0.9216
Epoch 278/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 178s 98ms/step - accuracy: 0.7741 -
loss: 0.8515 - val_accuracy: 0.7634 - val_loss: 0.9205
Epoch 279/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 183s 101ms/step - accuracy: 0.7740 -
loss: 0.8496 - val_accuracy: 0.7638 - val_loss: 0.9194
Epoch 280/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 182s 100ms/step - accuracy: 0.7756 -
loss: 0.8418 - val_accuracy: 0.7638 - val_loss: 0.9180
Epoch 281/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 187s 103ms/step - accuracy: 0.7738 -
loss: 0.8549 - val_accuracy: 0.7640 - val_loss: 0.9169
Epoch 282/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 179s 99ms/step - accuracy: 0.7731 -
loss: 0.8477 - val_accuracy: 0.7644 - val_loss: 0.9150
Epoch 283/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 183s 101ms/step - accuracy: 0.7763 -
loss: 0.8407 - val_accuracy: 0.7644 - val_loss: 0.9139
Epoch 284/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 183s 101ms/step - accuracy: 0.7766 -
loss: 0.8422 - val_accuracy: 0.7650 - val_loss: 0.9126
Epoch 285/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 179s 99ms/step - accuracy: 0.7759 -
loss: 0.8430 - val_accuracy: 0.7653 - val_loss: 0.9116
Epoch 286/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 169s 93ms/step - accuracy: 0.7748 -
loss: 0.8440 - val_accuracy: 0.7657 - val_loss: 0.9103
```

```
Epoch 287/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 168s 93ms/step - accuracy: 0.7755 -
loss: 0.8419 - val_accuracy: 0.7657 - val_loss: 0.9091
Epoch 288/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 167s 92ms/step - accuracy: 0.7775 -
loss: 0.8403 - val_accuracy: 0.7664 - val_loss: 0.9077
Epoch 289/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 167s 92ms/step - accuracy: 0.7776 -
loss: 0.8360 - val_accuracy: 0.7662 - val_loss: 0.9070
Epoch 290/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 168s 92ms/step - accuracy: 0.7783 -
loss: 0.8351 - val_accuracy: 0.7666 - val_loss: 0.9057
Epoch 291/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 168s 93ms/step - accuracy: 0.7781 -
loss: 0.8339 - val_accuracy: 0.7672 - val_loss: 0.9040
Epoch 292/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 165s 91ms/step - accuracy: 0.7784 -
loss: 0.8337 - val_accuracy: 0.7673 - val_loss: 0.9038
Epoch 293/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 168s 92ms/step - accuracy: 0.7786 -
loss: 0.8322 - val_accuracy: 0.7674 - val_loss: 0.9023
Epoch 294/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 168s 92ms/step - accuracy: 0.7782 -
loss: 0.8311 - val_accuracy: 0.7678 - val_loss: 0.9012
Epoch 295/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 201s 111ms/step - accuracy: 0.7784 -
loss: 0.8312 - val_accuracy: 0.7684 - val_loss: 0.9000
Epoch 296/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 205s 113ms/step - accuracy: 0.7796 -
loss: 0.8256 - val_accuracy: 0.7680 - val_loss: 0.8987
Epoch 297/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 203s 111ms/step - accuracy: 0.7806 -
loss: 0.8215 - val_accuracy: 0.7690 - val_loss: 0.8975
Epoch 298/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 207s 114ms/step - accuracy: 0.7806 -
loss: 0.8227 - val_accuracy: 0.7695 - val_loss: 0.8964
Epoch 299/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 203s 112ms/step - accuracy: 0.7816 -
loss: 0.8219 - val_accuracy: 0.7692 - val_loss: 0.8950
Epoch 300/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 196s 108ms/step - accuracy: 0.7806 -
loss: 0.8239 - val_accuracy: 0.7693 - val_loss: 0.8936
Epoch 301/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 196s 108ms/step - accuracy: 0.7816 -
loss: 0.8200 - val_accuracy: 0.7700 - val_loss: 0.8923
Epoch 302/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 197s 109ms/step - accuracy: 0.7794 -
loss: 0.8265 - val_accuracy: 0.7699 - val_loss: 0.8912
Epoch 303/350
```

```
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 204s 112ms/step - accuracy: 0.7816 -
loss: 0.8233 - val_accuracy: 0.7703 - val_loss: 0.8908
Epoch 304/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 235s 129ms/step - accuracy: 0.7818 -
loss: 0.8162 - val_accuracy: 0.7705 - val_loss: 0.8896
Epoch 305/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 265s 146ms/step - accuracy: 0.7822 -
loss: 0.8129 - val_accuracy: 0.7709 - val_loss: 0.8880
Epoch 306/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 253s 139ms/step - accuracy: 0.7823 -
loss: 0.8172 - val_accuracy: 0.7713 - val_loss: 0.8867
Epoch 307/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 253s 134ms/step - accuracy: 0.7849 -
loss: 0.8108 - val_accuracy: 0.7716 - val_loss: 0.8861
Epoch 308/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 234s 129ms/step - accuracy: 0.7837 -
loss: 0.8140 - val_accuracy: 0.7721 - val_loss: 0.8847
Epoch 309/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 214s 118ms/step - accuracy: 0.7820 -
loss: 0.8146 - val_accuracy: 0.7718 - val_loss: 0.8838
Epoch 310/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 203s 112ms/step - accuracy: 0.7831 -
loss: 0.8150 - val_accuracy: 0.7724 - val_loss: 0.8828
Epoch 311/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 199s 109ms/step - accuracy: 0.7839 -
loss: 0.8112 - val_accuracy: 0.7724 - val_loss: 0.8822
Epoch 312/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 202s 111ms/step - accuracy: 0.7835 -
loss: 0.8111 - val_accuracy: 0.7724 - val_loss: 0.8808
Epoch 313/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 201s 110ms/step - accuracy: 0.7858 -
loss: 0.8014 - val_accuracy: 0.7729 - val_loss: 0.8795
Epoch 314/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 214s 118ms/step - accuracy: 0.7859 -
loss: 0.8053 - val_accuracy: 0.7729 - val_loss: 0.8787
Epoch 315/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 203s 112ms/step - accuracy: 0.7850 -
loss: 0.8052 - val_accuracy: 0.7728 - val_loss: 0.8780
Epoch 316/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 183s 101ms/step - accuracy: 0.7828 -
loss: 0.8090 - val_accuracy: 0.7732 - val_loss: 0.8769
Epoch 317/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 183s 101ms/step - accuracy: 0.7851 -
loss: 0.8076 - val_accuracy: 0.7740 - val_loss: 0.8757
Epoch 318/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 169s 93ms/step - accuracy: 0.7863 -
loss: 0.8024 - val_accuracy: 0.7742 - val_loss: 0.8747
Epoch 319/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 169s 93ms/step - accuracy: 0.7860 -
```

```
loss: 0.7991 - val_accuracy: 0.7745 - val_loss: 0.8735
Epoch 320/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 167s 92ms/step - accuracy: 0.7873 -
loss: 0.7971 - val_accuracy: 0.7748 - val_loss: 0.8726
Epoch 321/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 167s 92ms/step - accuracy: 0.7883 -
loss: 0.7929 - val_accuracy: 0.7747 - val_loss: 0.8713
Epoch 322/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 169s 93ms/step - accuracy: 0.7865 -
loss: 0.8020 - val_accuracy: 0.7749 - val_loss: 0.8706
Epoch 323/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 167s 92ms/step - accuracy: 0.7861 -
loss: 0.7971 - val_accuracy: 0.7754 - val_loss: 0.8696
Epoch 324/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 167s 92ms/step - accuracy: 0.7877 -
loss: 0.7963 - val_accuracy: 0.7753 - val_loss: 0.8688
Epoch 325/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 169s 93ms/step - accuracy: 0.7882 -
loss: 0.7904 - val_accuracy: 0.7757 - val_loss: 0.8677
Epoch 326/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 182s 100ms/step - accuracy: 0.7880 -
loss: 0.7960 - val_accuracy: 0.7758 - val_loss: 0.8669
Epoch 327/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 195s 107ms/step - accuracy: 0.7882 -
loss: 0.7930 - val_accuracy: 0.7761 - val_loss: 0.8657
Epoch 328/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 187s 103ms/step - accuracy: 0.7881 -
loss: 0.7934 - val_accuracy: 0.7762 - val_loss: 0.8648
Epoch 329/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 191s 105ms/step - accuracy: 0.7876 -
loss: 0.7939 - val_accuracy: 0.7768 - val_loss: 0.8640
Epoch 330/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 168s 93ms/step - accuracy: 0.7911 -
loss: 0.7846 - val_accuracy: 0.7769 - val_loss: 0.8626
Epoch 331/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 170s 94ms/step - accuracy: 0.7917 -
loss: 0.7859 - val_accuracy: 0.7770 - val_loss: 0.8617
Epoch 332/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 169s 93ms/step - accuracy: 0.7875 -
loss: 0.7902 - val_accuracy: 0.7774 - val_loss: 0.8604
Epoch 333/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 177s 97ms/step - accuracy: 0.7882 -
loss: 0.7948 - val_accuracy: 0.7776 - val_loss: 0.8599
Epoch 334/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 170s 94ms/step - accuracy: 0.7893 -
loss: 0.7843 - val_accuracy: 0.7782 - val_loss: 0.8587
Epoch 335/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 176s 97ms/step - accuracy: 0.7902 -
loss: 0.7849 - val_accuracy: 0.7782 - val_loss: 0.8580
```

```
Epoch 336/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 190s 105ms/step - accuracy: 0.7918 -
loss: 0.7816 - val_accuracy: 0.7781 - val_loss: 0.8569
Epoch 337/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 191s 105ms/step - accuracy: 0.7892 -
loss: 0.7868 - val_accuracy: 0.7783 - val_loss: 0.8560
Epoch 338/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 192s 106ms/step - accuracy: 0.7903 -
loss: 0.7885 - val_accuracy: 0.7786 - val_loss: 0.8548
Epoch 339/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 191s 105ms/step - accuracy: 0.7904 -
loss: 0.7853 - val_accuracy: 0.7788 - val_loss: 0.8539
Epoch 340/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 187s 103ms/step - accuracy: 0.7903 -
loss: 0.7845 - val_accuracy: 0.7791 - val_loss: 0.8532
Epoch 341/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 187s 103ms/step - accuracy: 0.7907 -
loss: 0.7808 - val_accuracy: 0.7796 - val_loss: 0.8527
Epoch 342/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 199s 109ms/step - accuracy: 0.7906 -
loss: 0.7856 - val_accuracy: 0.7793 - val_loss: 0.8517
Epoch 343/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 205s 113ms/step - accuracy: 0.7924 -
loss: 0.7758 - val_accuracy: 0.7797 - val_loss: 0.8508
Epoch 344/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 199s 110ms/step - accuracy: 0.7918 -
loss: 0.7806 - val_accuracy: 0.7803 - val_loss: 0.8495
Epoch 345/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 191s 105ms/step - accuracy: 0.7917 -
loss: 0.7781 - val_accuracy: 0.7805 - val_loss: 0.8487
Epoch 346/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 198s 109ms/step - accuracy: 0.7923 -
loss: 0.7757 - val_accuracy: 0.7808 - val_loss: 0.8476
Epoch 347/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 213s 117ms/step - accuracy: 0.7923 -
loss: 0.7764 - val_accuracy: 0.7810 - val_loss: 0.8471
Epoch 348/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 204s 112ms/step - accuracy: 0.7918 -
loss: 0.7796 - val_accuracy: 0.7812 - val_loss: 0.8465
Epoch 349/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 189s 104ms/step - accuracy: 0.7908 -
loss: 0.7794 - val_accuracy: 0.7814 - val_loss: 0.8450
Epoch 350/350
1816/1816 ━━━━━━━━━━━━━━━━━━━━ 188s 104ms/step - accuracy: 0.7922 -
loss: 0.7756 - val_accuracy: 0.7815 - val_loss: 0.8443
```

El entrenamiento fue realizado por el término de 350 épocas obteniéndose una accuracy y precision del 78.15% para el conjunto de prueba.

(incluir foto del accuracy a lo largo de las épocas)

```python
# Metrics
train_score = model.evaluate(x_train, y_train, verbose=0)
test_score = model.evaluate(x_test, y_test, verbose=0)
print('Train loss:', train_score[0])
print('Train accuracy:', train_score[1])
print('Test loss:', test_score[0])
print('Test accuracy:', test_score[1])
```

```
Train loss: 0.4342772960662842
Train accuracy: 0.8883481025695801
Test loss: 0.8442897200584412
Test accuracy: 0.7814615964889526
```

```python
# Realizar predicciones en todo el conjunto de prueba
predictions = model.predict(x_test)

# Obtener las clases predichas
predicted_classes = np.argmax(predictions, axis=1)

# Obtener las clases verdaderas
true_classes = np.argmax(y_test, axis=1)

# Contar cuántas predicciones son correctas
correct_predictions = np.sum(predicted_classes == true_classes)

# Calcular la precisión
accuracy = correct_predictions / len(y_test)

print(f"Total de imágenes en el conjunto de prueba: {len(y_test)}")
print(f"Predicciones correctas: {correct_predictions}")
print(f"Precisión en el conjunto de prueba: {accuracy:.2%}")
```

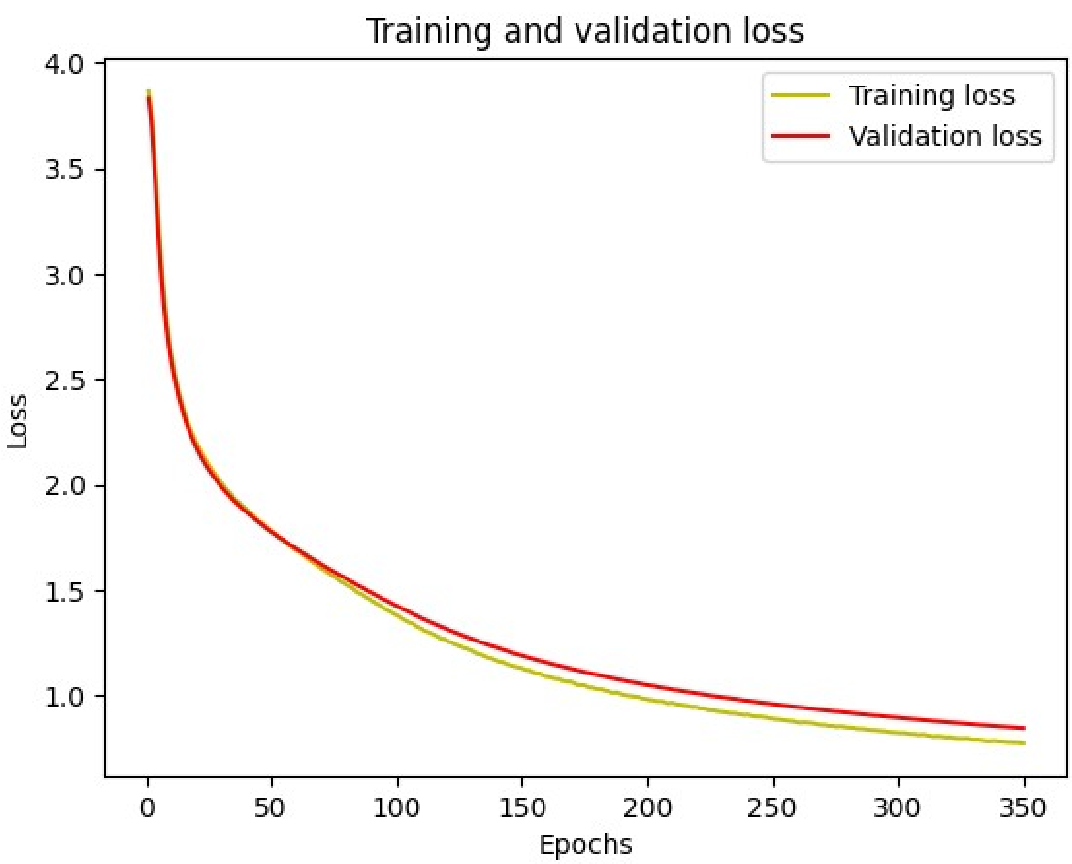
```
1205/1205 ━━━━━━━━━━━━━━━━━━━━ 11s 9ms/step
Total de imágenes en el conjunto de prueba: 38547
Predicciones correctas: 30123
Precisión en el conjunto de prueba: 78.15%
```

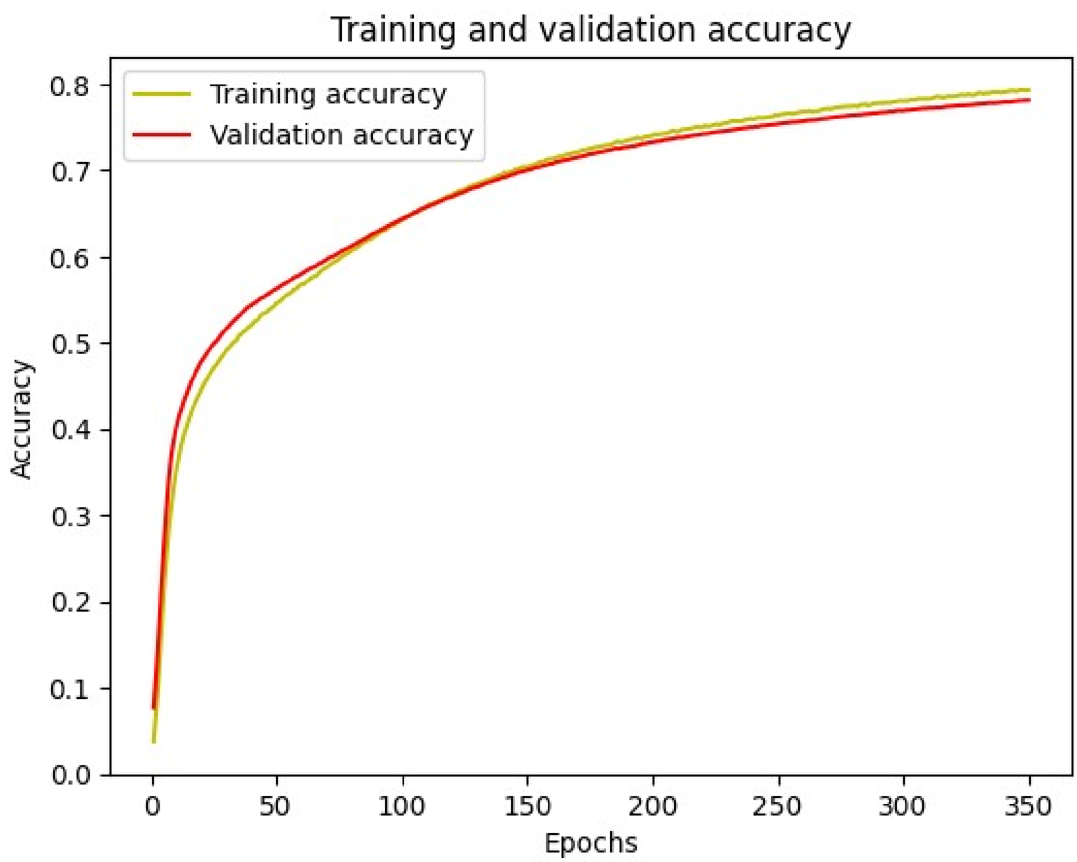
```python
model.save('kanji_model_350_epochs.hdf5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
```

```python
from matplotlib import pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
```

```
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.plot(epochs, acc, 'y', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





La entrada del modelo es la matriz de 28x28 pixeles que representa la imágen, dicha matriz esta escalada a un intervalo [0, 1]. La salida de la red es un vector de 49 posiciones donde en cada posición se encuentra la probabilidad que la entrada pertenezca a esa clase.

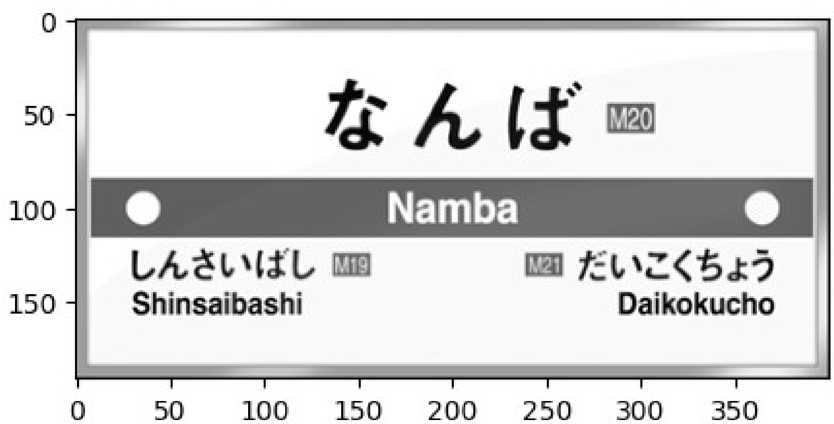
Luego, es posible cargar el modelo ya entrenado como se muestra a continuación:

```
num_classes = 49

def load(f):
    return np.load(f)['arr_0']

# Based on train set
x_img = load('model/k49-train-imgs.npz')
y_class = keras.utils.to_categorical(load('model/k49-train-
labels.npz'), num_classes)
y_class = np.argmax(y_class, axis=1)

# Checking the number of classes
y_class_unique = np.unique(y_class)
```

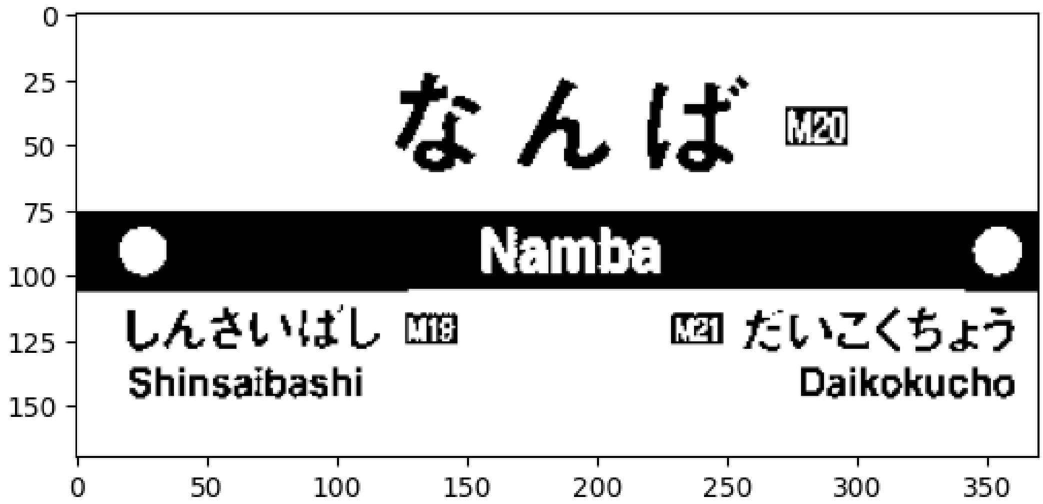
Segmentación

Por otro lado, para la separación de los hiragana se aplicó la transformada de Hough para identificar bordes. Primero, a la imágen se le realiza una transformación por threshold, asignando a todos los valores por debajo de 127 de luminancia el blanco mientras que a los mayores el negro.

```
IMG_NAME = "hiragana_sign_1.png"

# B&W Convertion
img = cv2.imread("./images/"+IMG_NAME)
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


fig = plt.figure()
fig.set_size_inches(5, 5)
plt.imshow(img, cmap='gray')
plt.show()
```

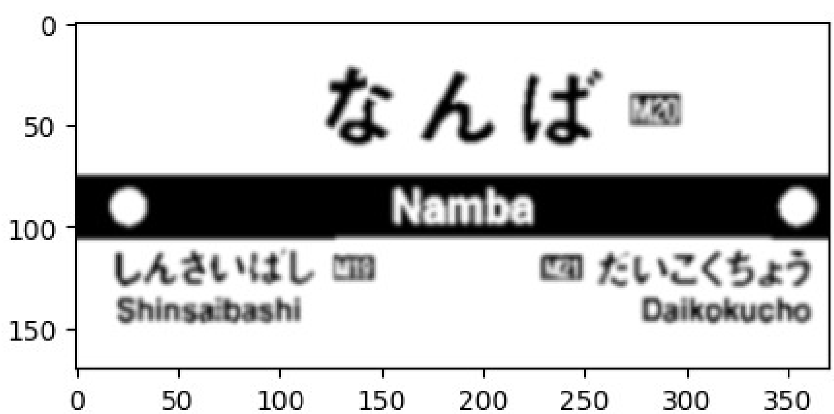



```
# Apply Threshold Transformation
_, img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
img = img[10:180, 10:380]
plt.imshow(img, cmap='gray')
plt.show()
```



Luego, se aplica un Gaussian Blur para eliminar ruido y suavizar pequeñas diferencias en los trazos.

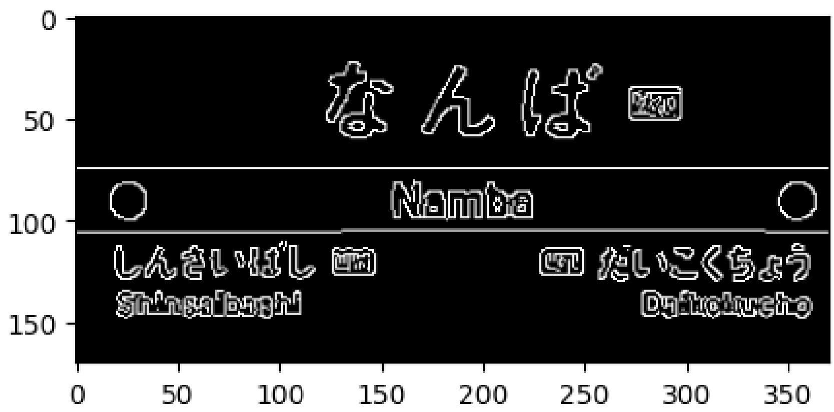
```
# Blurred
blurred_image = cv2.GaussianBlur(img, (5, 5), 0)

fig = plt.figure()
fig.set_size_inches(5, 5)
plt.imshow(blurred_image, cmap='gray')
plt.show()
```



Al resultado se le aplica la función Canny para obtener los bordes presentes.

```
# Canny
canny_image = cv2.Canny(blurred_image, 225, 255)

fig = plt.figure()
fig.set_size_inches(5, 5)
```

```
plt.imshow(canny_image, cmap='gray')
plt.show()
```



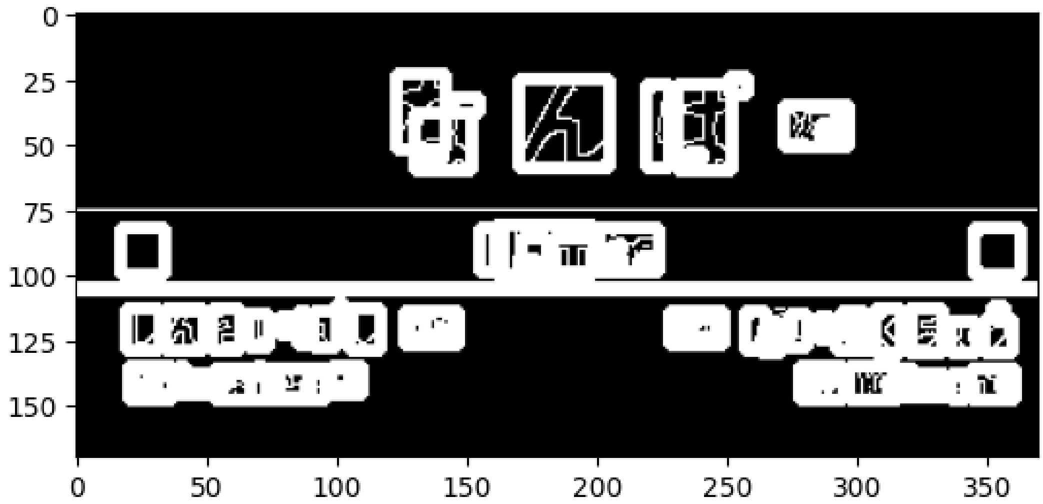
Mediante la transformada de Hough se consiguen las ubicaciones de dichos trazados.

Posteriormente se realiza el armado del recuadro donde se encuentra cada símbolo. Para ello, a cada línea detectada se le define un rectángulo con las coordenadas más a la izquierda, más a la derecha, más arriba y más abajo. Esto provoca múltiples rectángulos con intersección no nula, muchos de ellos pegados entre sí.

```
def _find_sections(canny_image: np.ndarray) ->
tuple[list[tuple[tuple[int, int], tuple[int, int]]]], np.ndarray]:
    contours, hierarchy = cv2.findContours(canny_image,
cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
    square_sections_limits = []
    for contour in contours:
        highest = contour.max(axis=0)[0]
        lowest = contour.min(axis=0)[0]

        if highest[0] <= lowest[0]:
            continue
        if highest[1] <= lowest[1]:
            continue

        square_sections_limits.append(((lowest[1], (highest[1]+1)),
(lowest[0], (highest[0]+1))))
    return square_sections_limits, hierarchy

def obtain_sections(
        canny_image: np.ndarray
        ) -> tuple[tuple[tuple[int, int], tuple[int, int]],
np.ndarray]:
    canny_image_copy = canny_image.copy()
    square_sections_limits, hierarchy =
_find_sections(canny_image_copy)
    return square_sections_limits, hierarchy
```

```python
sections_limits, _ = obtain_sections(canny_image)

def _sections_to_square_contours(sections_limits: np.ndarray) ->
tuple:
    square_contours = []
    for section_limit in sections_limits:
        square_contours.append(
                np.array(
                    [
                        np.array([[section_limit[1][0],
section_limit[0][0]]]),
                        np.array([[section_limit[1][0],
section_limit[0][1]-1]]),
                        np.array([[section_limit[1][1]-1,
section_limit[0][1]-1]]),
                        np.array([[section_limit[1][1]-1,
section_limit[0][0]]]),
                        np.array([[section_limit[1][0],
section_limit[0][0]]]),
                    ]
                )
            )
    square_contours = tuple(square_contours)
    return square_contours

def draw_sections(
        canny_image: np.ndarray,
        sections_limits: list[tuple[tuple[int, int], tuple[int, int]]]
        ) -> np.ndarray:
    canny_image_copy = canny_image.copy()
    square_contours = _sections_to_square_contours(sections_limits)
    return cv2.drawContours(canny_image_copy, square_contours, -1,
255, 3)


plt.imshow(draw_sections(canny_image, sections_limits), cmap="grey")
```

```
<matplotlib.image.AxesImage at 0x7412c99e6680>
```

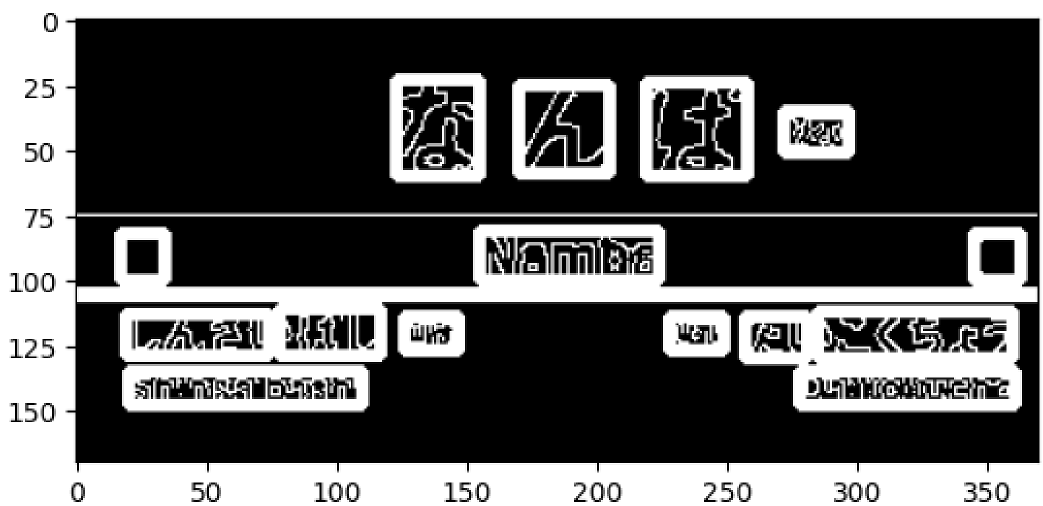


Ahora bien, podemos decir que dos rectángulos con intersección no nula o que comience uno inmediatamente después de otro es indicativo de que se trata del mismo símbolo. Por lo tanto, calculamos los nuevos límites como el rectángulo que englobe a todos los otros que cumpla esta última condición.

```
def _ranges_intersect(
        limit_a: tuple[int, int],
        limit_b: tuple[int, int],
        tol: float
        ) -> bool:
    min_a, max_a = limit_a
    min_b, max_b = limit_b
    if min_a < min_b:
        return max_a+tol >= min_b
    else:
        return max_b+tol >= min_a

def _augment_range(
        limit_a: tuple[int, int],
        limit_b: tuple[int, int]
        ) -> tuple[int, int]:
    min_a, max_a = limit_a
    min_b, max_b = limit_b
    lowest_min = min(min_a, min_b)
    highest_max = max(max_a, max_b)
    return (lowest_min, highest_max)

def _sections_intersect(
        section_a: tuple[tuple[int, int], tuple[int, int]],
        section_b: tuple[tuple[int, int], tuple[int, int]],
        tol: float
        ) -> bool:
    vertical_limits_a, horizontal_limits_a = section_a
```

```
    vertical_limits_b, horizontal_limits_b = section_b
    return _ranges_intersect(horizontal_limits_a, horizontal_limits_b,
tol) and _ranges_intersect(vertical_limits_a, vertical_limits_b, tol)

def _augment_section(
        section_a: tuple[tuple[int, int], tuple[int, int]],
        section_b: tuple[tuple[int, int], tuple[int, int]]
        ) -> tuple[tuple[int, int], tuple[int, int]]:
    vertical_limits_a, horizontal_limits_a = section_a
    vertical_limits_b, horizontal_limits_b = section_b
    new_vertical_limits = _augment_range(vertical_limits_a,
vertical_limits_b)
    new_horizontal_limits = _augment_range(horizontal_limits_a,
horizontal_limits_b)
    return (new_vertical_limits, new_horizontal_limits)

def merge_all_sections(
        sections_limits: list[tuple[tuple[int, int], tuple[int,
int]]],
        tol: float = 0
    ) -> list[tuple[tuple[int, int], tuple[int, int]]]:

    have_intersected = True

    while have_intersected:
        have_intersected = False

        group_labels_aux = list(range(len(sections_limits)))
        for i in range(len(sections_limits)):
            for j in range(len(sections_limits)):
                if i != j and _sections_intersect(sections_limits[i],
sections_limits[j], tol):
                    have_intersected = True
                    group_label = min(group_labels_aux[i],
group_labels_aux[j])
                    group_labels_aux[i] = group_label
                    group_labels_aux[j] = group_label

        group_labels = set()
        for group_label in group_labels_aux:
            group_labels.add(group_label)

        new_sections_limits = []
        for group_label in group_labels:
            new_section_limits = None
            for i in range(len(group_labels_aux)):
                if group_labels_aux[i] == group_label:
                    if new_section_limits is None:
                        new_section_limits = sections_limits[i]
                    else:
```

```
                        new_section_limits =
_augment_section(new_section_limits, sections_limits[i])
            new_sections_limits.append(new_section_limits)

        sections_limits = new_sections_limits

    return new_sections_limits

new_sections_limits = merge_all_sections(sections_limits, 2)
plt.imshow(draw_sections(canny_image, new_sections_limits),
cmap="grey")

<matplotlib.image.AxesImage at 0x7412c99c4a60>
```

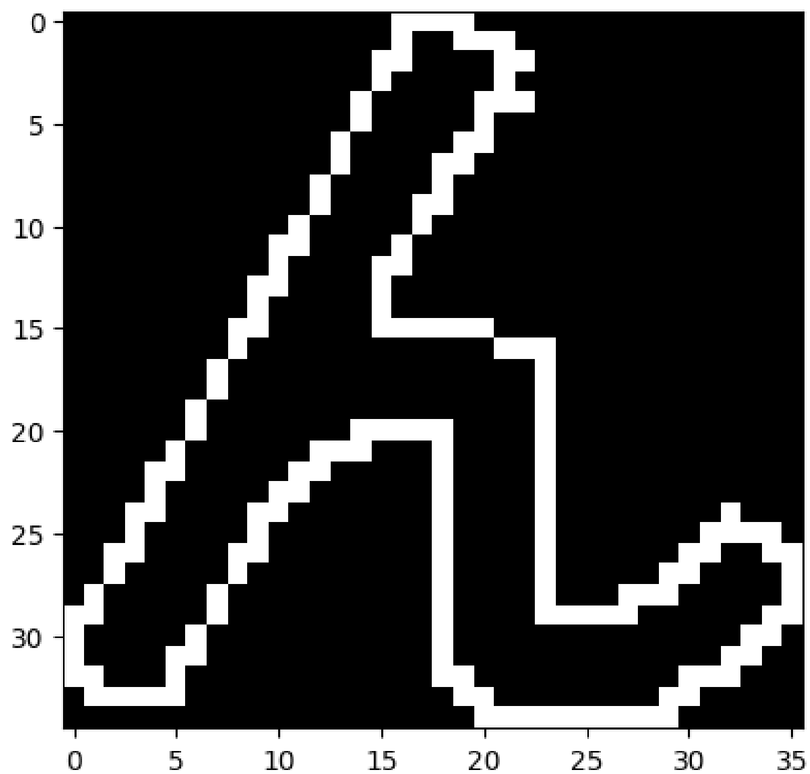


Para terminar con el seccionado, se recorta la imágen producto del filtro gaussiano a las porciones delimitadas por los rectángulos. Estos recortes constituyen los ejemplos a los que se debe identificar que hiragana le corresponde.

```
SECTION_VERTICAL = 0
SECTION_HORIZONTAL = 1
SECTION_LOWEST = 0
SECTION_HIGHEST = 1

def get_section(
        image: np.ndarray,
        section_limits: tuple[tuple[int, int], tuple[int, int]]
        ) -> np.ndarray:
    return image[
        section_limits[SECTION_VERTICAL]
[SECTION_LOWEST]:section_limits[SECTION_VERTICAL][SECTION_HIGHEST],
        section_limits[SECTION_HORIZONTAL]
[SECTION_LOWEST]:section_limits[SECTION_HORIZONTAL][SECTION_HIGHEST]
    ]
```

```
def sectionize_image(
        canny_image: np.ndarray,
        sections_limits: list[tuple[tuple[int, int], tuple[int, int]]]
        ) -> list[np.ndarray]:
    sectionized_image = []
    for section_limits in sections_limits:
        sectionized_image.append(get_section(canny_image,
section_limits))
    return sectionized_image

sections = sectionize_image(canny_image, new_sections_limits)
plt.imshow(sections[12], cmap="grey")
```

```
<matplotlib.image.AxesImage at 0x7412d69666e0>
```



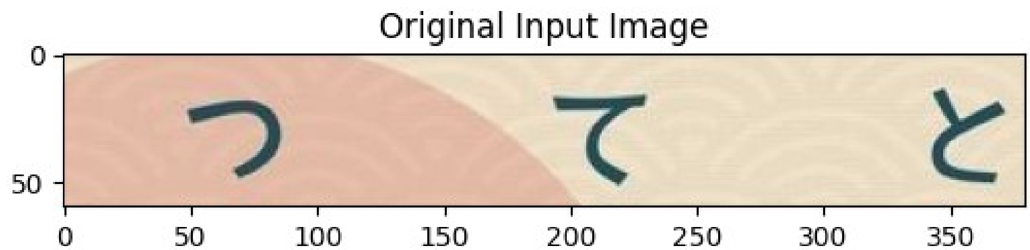
Pre-Procesamiento de la Entrada

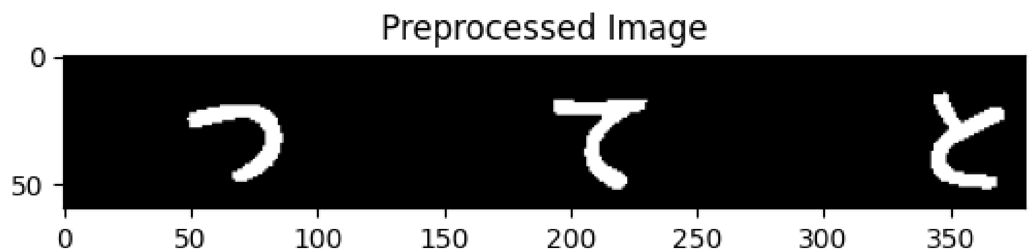
Las nuevas imágenes, resultado de la segmentación, contienen al hiragana a identificar. A cada hiragana se le aplica Canny y un cerrado morfológico de bordes para luego rellenar los trazos.

```
fill_borders = True

def input_preprocessing(img_name):
```

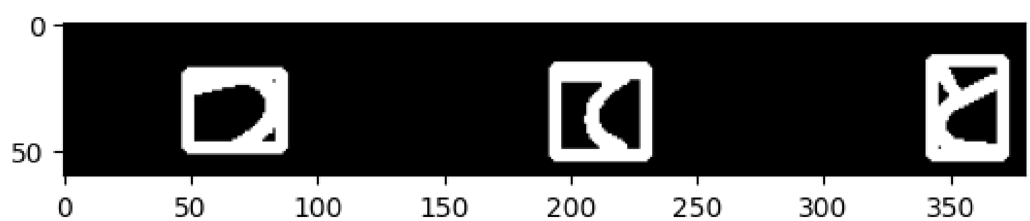
```
    # Load image
    img = cv2.imread("./images/"+img_name)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img[390:450, 300:680]

    plt.imshow(img, cmap="gray")
    plt.title("Original Input Image")
    plt.show()

    # B&W Convertion
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

    # Blurred
    blurred_image = cv2.GaussianBlur(img, (5, 5), 0)

    # Canny
    edges = cv2.Canny(blurred_image, 225, 255)

    if fill_borders:
        # Kernel for morphological operations
        kernel = np.ones((2, 1), np.uint8)

        # Morphological closing of gaps
        closed_edges = cv2.morphologyEx(edges, cv2.MORPH_CLOSE,
kernel)

        # Find contours
        contours, _ = cv2.findContours(closed_edges,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        # Fill
        mask = np.zeros_like(img)
        contours = cv2.drawContours(mask, contours, -1, (255),
thickness=cv2.FILLED)

        processed_image = contours

    else:
        processed_image = edges

    plt.imshow(processed_image, cmap="gray")
    plt.title("Preprocessed Image")
    plt.show()

    # Sections
    sections_limits, _ = obtain_sections(processed_image)
    new_sections_limits = merge_all_sections(sections_limits, 2)
    sections = sectionize_image(processed_image, new_sections_limits)

    return processed_image, new_sections_limits, sections
```

```
img_name = "hiragana.png"
borders_image, section_limits, sections =
input_preprocessing(img_name)
```






```
plt.imshow(draw_sections(borders_image, section_limits), cmap="grey")

<matplotlib.image.AxesImage at 0x7412c3f83160>
```



Luego, se realiza una reducción de las dimensiones de la imágenes para que estas pasen a ser de 28x28 y se escala el valor de cada pixel al intervalo [0, 1].

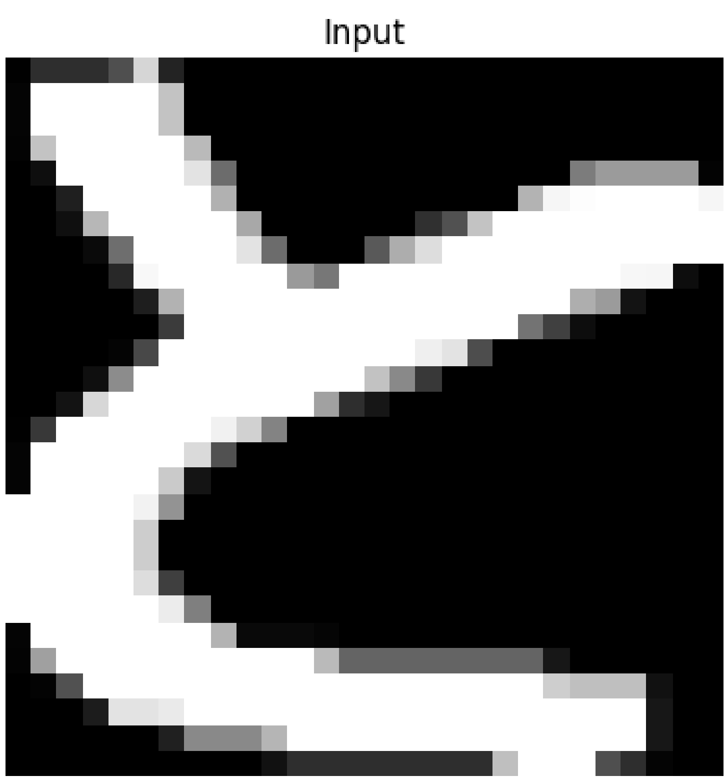
Pos-Procesamento de la Salida

Ahora se está en condiciones de predecir cada hiragana. Se obtiene un vector de 49 posiciones. Tomamos la posición con valor más grande de este vector que representa la clase ganadora por hiragana.

```
model =
keras.saving.load_model("model/kanji_model_with_dynamic_augmentation.h
df5")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have
yet to be built. `model.compile_metrics` will be empty until you train
or evaluate the model.
```

```python
def predict_hiragana(target, show_plot=False):
    section = cv2.resize(target, (28, 28))

    if show_plot:
        plt.imshow(section, cmap="grey")
        plt.title("Input")
        plt.axis("off")
        plt.show()

    section = section.astype('float32')
    section /= 255

    predictions = model.predict(np.array([section]))
    prediction = predictions.argmax()

    return prediction

if fill_borders:
    target = sections[2]
else:
    target = sections[1]
prediction = predict_hiragana(target, show_plot=True)
```

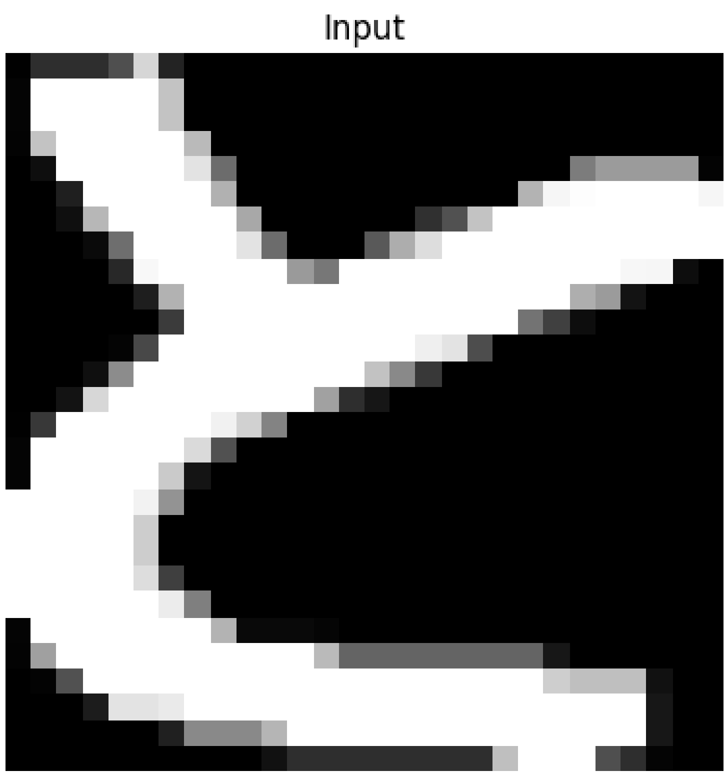


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
```

Esta clase se la mapea al carácter unicode del hiragana que representa.

```python
class2unicode = {
    0: "\u3042",
    1: "\u3044",
    2: "\u3046",
    3: "\u3048",
    4: "\u304A",
    5: "\u304B",
    6: "\u304D",
    7: "\u304F",
    8: "\u3051",
    9: "\u3053",
    10: "\u3055",
    11: "\u3057",
    12: "\u3059",
    13: "\u305B",
    14: "\u305D",
    15: "\u305F",
    16: "\u3061",
    17: "\u3064",
    18: "\u3066",
    19: "\u3068",
    20: "\u306A",
    21: "\u306B",
    22: "\u306C",
    23: "\u306D",
    24: "\u306E",
    25: "\u306F",
    26: "\u3072",
    27: "\u3075",
    28: "\u3078",
    29: "\u307B",
    30: "\u307E",
    31: "\u307F",
    32: "\u3080",
    33: "\u3081",
    34: "\u3082",
    35: "\u3084",
    36: "\u3086",
    37: "\u3088",
    38: "\u3089",
    39: "\u308A",
    40: "\u308B",
    41: "\u308C",
    42: "\u308D",
    43: "\u308F",
```

```
    44: "\u3090",
    45: "\u3091",
    46: "\u3092",
    47: "\u3093",
    48: "\u309D"
}

map_class2img = {}
for i in range(num_classes):
    idx = np.where(y_class == i)[0][:5]
    map_class2img[i] = x_img[idx]

if fill_borders:
    target = sections[2]
else:
    target = sections[1]
prediction = predict_hiragana(target, show_plot=True)
display('### Predicted Hiragana:
{}'.format(class2unicode[prediction]))
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step

'### Predicted Hiragana: と'
```

Finalmente, este proceso se ejecuta por cada hiragana ordenado por posicionamiento en la imágen. De esta manera, se obtiene el texto del cartel.

```
selected_row = []

for idx, limits in enumerate(section_limits):
        selected_row.append(idx)
        print(idx, limits)

selected_row = sorted(selected_row, key=lambda idx:
section_limits[idx][1][0])

predictions = [class2unicode[x] for x in map(lambda x:
predict_hiragana(sections[x]), selected_row)]

predicted_word = "".join(predictions)
display('### Predicted Hiragana word: {}'.format(predicted_word))
```

```
0 ((np.int32(20), np.int32(50)), (np.int32(49), np.int32(87)))
1 ((np.int32(18), np.int32(53)), (np.int32(194), np.int32(231)))
2 ((np.int32(15), np.int32(53)), (np.int32(343), np.int32(372)))
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step

'### Predicted Hiragana word: つてと'
```