Actividad_Autoencodes

November 22, 2023

0.1 Actividad: Autoencodes

Implementen un autoencoder para la base de datos de "Emojis". Para ello, sigan los siguientes pasos:

1. Dividan aleatoriamente su conjunto de datos de tal manera que el 80% de los datos sean para entrenamiento y un 20% para prueba. Procuren que en la división las clases mantengan la misma proporción tanto en los datos de entrenamiento como en los de prueba respecto a las proporciones.

[ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from collections import Counter


data = np.loadtxt("emojis.txt")
x = data[:,1:]
y = data[:,0]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.
 ↪2,stratify=y)
```

2. Sigan los pasos del tutorial "Building autoencoders with Keras" para ajustar un autoencoder de una capa oculta para los datos de Emojis utilizando los datos de entrenamiento.

[ ]:
```
import keras
from keras import layers
import matplotlib.pyplot as plt

encoding_dim = 32

input_img = keras.Input(shape=(x_train.shape[1]))
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
decoded = layers.Dense(x_train.shape[1], activation='sigmoid')(encoded)

autoencoder = keras.Model(input_img, decoded)
encoder = keras.Model(input_img, encoded)

encoded_input = keras.Input(shape=(encoding_dim,))
```

```
decoder_layer = autoencoder.layers[-1]
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))


# 3. Utilicen algunas imágenes del conjunto de prueba para ver la salida de la␣
 ↪capa intermedia
# y de la capa de decodificación.

encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

n = 5
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

#4. Evalúen el error cuadrático medio entre las imágenes de prueba
# y su correspondiente salida.

from sklearn.metrics import mean_squared_error

mse_test = mean_squared_error(x_test, decoded_imgs)

print(f'MSE | Datos de prueba: {mse_test}')
```

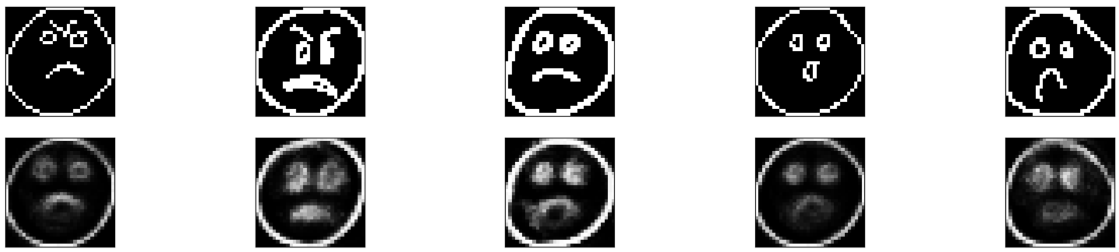
```
Epoch 1/50
8/8 [==============================] - 1s 45ms/step - loss: 0.6772 - val_loss:
0.6388
```

```
Epoch 2/50
8/8 [==============================] - 0s 26ms/step - loss: 0.5918 - val_loss:
0.5232
Epoch 3/50
8/8 [==============================] - 0s 16ms/step - loss: 0.4968 - val_loss:
0.4589
Epoch 4/50
8/8 [==============================] - 0s 15ms/step - loss: 0.4516 - val_loss:
0.4285
Epoch 5/50
8/8 [==============================] - 0s 16ms/step - loss: 0.4262 - val_loss:
0.4150
Epoch 6/50
8/8 [==============================] - 0s 15ms/step - loss: 0.4122 - val_loss:
0.4030
Epoch 7/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3999 - val_loss:
0.3913
Epoch 8/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3889 - val_loss:
0.3810
Epoch 9/50
8/8 [==============================] - 0s 16ms/step - loss: 0.3795 - val_loss:
0.3724
Epoch 10/50
8/8 [==============================] - 0s 16ms/step - loss: 0.3715 - val_loss:
0.3649
Epoch 11/50
8/8 [==============================] - 0s 17ms/step - loss: 0.3644 - val_loss:
0.3587
Epoch 12/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3579 - val_loss:
0.3526
Epoch 13/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3518 - val_loss:
0.3469
Epoch 14/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3461 - val_loss:
0.3419
Epoch 15/50
8/8 [==============================] - 0s 16ms/step - loss: 0.3409 - val_loss:
0.3373
Epoch 16/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3361 - val_loss:
0.3331
Epoch 17/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3318 - val_loss:
0.3292
```

```
Epoch 18/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3277 - val_loss:
0.3257
Epoch 19/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3240 - val_loss:
0.3224
Epoch 20/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3205 - val_loss:
0.3192
Epoch 21/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3172 - val_loss:
0.3162
Epoch 22/50
8/8 [==============================] - 0s 17ms/step - loss: 0.3140 - val_loss:
0.3133
Epoch 23/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3109 - val_loss:
0.3105
Epoch 24/50
8/8 [==============================] - 0s 16ms/step - loss: 0.3080 - val_loss:
0.3078
Epoch 25/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3051 - val_loss:
0.3053
Epoch 26/50
8/8 [==============================] - 0s 16ms/step - loss: 0.3024 - val_loss:
0.3029
Epoch 27/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2998 - val_loss:
0.3005
Epoch 28/50
8/8 [==============================] - 0s 18ms/step - loss: 0.2974 - val_loss:
0.2983
Epoch 29/50
8/8 [==============================] - 0s 14ms/step - loss: 0.2949 - val_loss:
0.2961
Epoch 30/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2926 - val_loss:
0.2941
Epoch 31/50
8/8 [==============================] - 0s 14ms/step - loss: 0.2904 - val_loss:
0.2921
Epoch 32/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2882 - val_loss:
0.2901
Epoch 33/50
8/8 [==============================] - 0s 14ms/step - loss: 0.2861 - val_loss:
0.2882
```

```
Epoch 34/50
8/8 [==============================] - 0s 17ms/step - loss: 0.2840 - val_loss:
0.2864
Epoch 35/50
8/8 [==============================] - 0s 14ms/step - loss: 0.2819 - val_loss:
0.2845
Epoch 36/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2799 - val_loss:
0.2827
Epoch 37/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2780 - val_loss:
0.2811
Epoch 38/50
8/8 [==============================] - 0s 14ms/step - loss: 0.2760 - val_loss:
0.2793
Epoch 39/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2742 - val_loss:
0.2776
Epoch 40/50
8/8 [==============================] - 0s 14ms/step - loss: 0.2723 - val_loss:
0.2760
Epoch 41/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2705 - val_loss:
0.2744
Epoch 42/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2687 - val_loss:
0.2729
Epoch 43/50
8/8 [==============================] - 0s 14ms/step - loss: 0.2670 - val_loss:
0.2713
Epoch 44/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2653 - val_loss:
0.2698
Epoch 45/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2636 - val_loss:
0.2683
Epoch 46/50
8/8 [==============================] - 0s 14ms/step - loss: 0.2620 - val_loss:
0.2669
Epoch 47/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2603 - val_loss:
0.2656
Epoch 48/50
8/8 [==============================] - 0s 14ms/step - loss: 0.2587 - val_loss:
0.2642
Epoch 49/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2572 - val_loss:
0.2629
```

```
Epoch 50/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2557 - val_loss:
0.2617
16/16 [==============================] - 0s 1ms/step
16/16 [==============================] - 0s 1ms/step
MSE | Datos de prueba: 0.08176597702897656
```



5. Agreguen un factor de regularización a la capa intermedia y repitan los pasos 3 y 4 con este nuevo modelo.

[ ]:

```python
import keras
from keras import layers
import matplotlib.pyplot as plt
from keras import regularizers


encoding_dim = 32

input_img = keras.Input(shape=(x_train.shape[1]))

# Agregamos un factor de regularización a nuestra capa intermedia
encoded = layers.Dense(encoding_dim, activation='relu',
                activity_regularizer=regularizers.l1(10e-5))(input_img)
decoded = layers.Dense(x_train.shape[1], activation='sigmoid')(encoded)

autoencoder = keras.Model(input_img, decoded)
encoder = keras.Model(input_img, encoded)

encoded_input = keras.Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                epochs=100,
                batch_size=256,
```

```python
                 shuffle=True,
                 validation_data=(x_test, x_test))


# 3. Utilicen algunas imágenes del conjunto de prueba para ver la salida de la␣
 ↪capa intermedia
# y de la capa de decodificación.

encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

n = 5
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

#4. Evalúen el error cuadrático medio entre las imágenes de prueba
# y su correspondiente salida.

from sklearn.metrics import mean_squared_error

mse_test = mean_squared_error(x_test, decoded_imgs)

print(f'MSE | Datos de prueba: {mse_test}')
```

```
Epoch 1/100
8/8 [==============================] - 1s 29ms/step - loss: 0.6821 - val_loss:
0.6510
Epoch 2/100
8/8 [==============================] - 0s 16ms/step - loss: 0.6063 - val_loss:
0.5381
Epoch 3/100
8/8 [==============================] - 0s 16ms/step - loss: 0.5079 - val_loss:
0.4704
Epoch 4/100
8/8 [==============================] - 0s 16ms/step - loss: 0.4637 - val_loss:
```

```
0.4396
Epoch 5/100
8/8 [==============================] - 0s 16ms/step - loss: 0.4369 - val_loss:
0.4262
Epoch 6/100
8/8 [==============================] - 0s 16ms/step - loss: 0.4240 - val_loss:
0.4158
Epoch 7/100
8/8 [==============================] - 0s 17ms/step - loss: 0.4123 - val_loss:
0.4045
Epoch 8/100
8/8 [==============================] - 0s 16ms/step - loss: 0.4020 - val_loss:
0.3944
Epoch 9/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3929 - val_loss:
0.3863
Epoch 10/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3856 - val_loss:
0.3792
Epoch 11/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3792 - val_loss:
0.3731
Epoch 12/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3732 - val_loss:
0.3672
Epoch 13/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3673 - val_loss:
0.3617
Epoch 14/100
8/8 [==============================] - 0s 14ms/step - loss: 0.3618 - val_loss:
0.3565
Epoch 15/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3568 - val_loss:
0.3520
Epoch 16/100
8/8 [==============================] - 0s 17ms/step - loss: 0.3522 - val_loss:
0.3479
Epoch 17/100
8/8 [==============================] - 0s 14ms/step - loss: 0.3480 - val_loss:
0.3440
Epoch 18/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3439 - val_loss:
0.3402
Epoch 19/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3398 - val_loss:
0.3364
Epoch 20/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3358 - val_loss:
```

```
0.3327
Epoch 21/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3320 - val_loss:
0.3293
Epoch 22/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3283 - val_loss:
0.3259
Epoch 23/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3248 - val_loss:
0.3229
Epoch 24/100
8/8 [==============================] - 0s 18ms/step - loss: 0.3214 - val_loss:
0.3197
Epoch 25/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3181 - val_loss:
0.3168
Epoch 26/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3150 - val_loss:
0.3140
Epoch 27/100
8/8 [==============================] - 0s 14ms/step - loss: 0.3120 - val_loss:
0.3114
Epoch 28/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3092 - val_loss:
0.3088
Epoch 29/100
8/8 [==============================] - 0s 15ms/step - loss: 0.3065 - val_loss:
0.3066
Epoch 30/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3041 - val_loss:
0.3044
Epoch 31/100
8/8 [==============================] - 0s 16ms/step - loss: 0.3018 - val_loss:
0.3023
Epoch 32/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2996 - val_loss:
0.3004
Epoch 33/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2975 - val_loss:
0.2986
Epoch 34/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2955 - val_loss:
0.2968
Epoch 35/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2937 - val_loss:
0.2951
Epoch 36/100
8/8 [==============================] - 0s 17ms/step - loss: 0.2919 - val_loss:
```

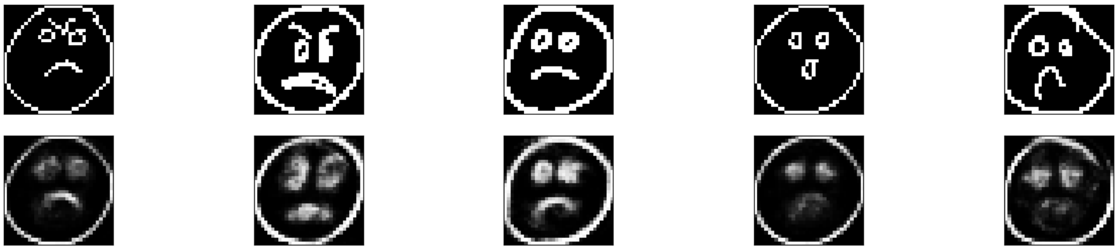
```
0.2936
Epoch 37/100
8/8 [==============================] - 0s 14ms/step - loss: 0.2902 - val_loss:
0.2920
Epoch 38/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2885 - val_loss:
0.2905
Epoch 39/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2869 - val_loss:
0.2892
Epoch 40/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2853 - val_loss:
0.2878
Epoch 41/100
8/8 [==============================] - 0s 17ms/step - loss: 0.2838 - val_loss:
0.2864
Epoch 42/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2823 - val_loss:
0.2851
Epoch 43/100
8/8 [==============================] - 0s 18ms/step - loss: 0.2808 - val_loss:
0.2838
Epoch 44/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2793 - val_loss:
0.2826
Epoch 45/100
8/8 [==============================] - 0s 17ms/step - loss: 0.2780 - val_loss:
0.2815
Epoch 46/100
8/8 [==============================] - 0s 17ms/step - loss: 0.2766 - val_loss:
0.2803
Epoch 47/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2752 - val_loss:
0.2791
Epoch 48/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2739 - val_loss:
0.2780
Epoch 49/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2726 - val_loss:
0.2769
Epoch 50/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2713 - val_loss:
0.2759
Epoch 51/100
8/8 [==============================] - 0s 14ms/step - loss: 0.2701 - val_loss:
0.2748
Epoch 52/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2688 - val_loss:
```

```
0.2738
Epoch 53/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2676 - val_loss:
0.2728
Epoch 54/100
8/8 [==============================] - 0s 14ms/step - loss: 0.2664 - val_loss:
0.2718
Epoch 55/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2653 - val_loss:
0.2708
Epoch 56/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2641 - val_loss:
0.2699
Epoch 57/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2630 - val_loss:
0.2690
Epoch 58/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2619 - val_loss:
0.2682
Epoch 59/100
8/8 [==============================] - 0s 14ms/step - loss: 0.2608 - val_loss:
0.2673
Epoch 60/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2598 - val_loss:
0.2665
Epoch 61/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2588 - val_loss:
0.2657
Epoch 62/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2578 - val_loss:
0.2649
Epoch 63/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2568 - val_loss:
0.2641
Epoch 64/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2558 - val_loss:
0.2634
Epoch 65/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2549 - val_loss:
0.2627
Epoch 66/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2539 - val_loss:
0.2619
Epoch 67/100
8/8 [==============================] - 0s 14ms/step - loss: 0.2531 - val_loss:
0.2612
Epoch 68/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2522 - val_loss:
```

```
0.2606
Epoch 69/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2513 - val_loss:
0.2599
Epoch 70/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2504 - val_loss:
0.2592
Epoch 71/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2496 - val_loss:
0.2586
Epoch 72/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2488 - val_loss:
0.2580
Epoch 73/100
8/8 [==============================] - 0s 18ms/step - loss: 0.2480 - val_loss:
0.2574
Epoch 74/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2472 - val_loss:
0.2568
Epoch 75/100
8/8 [==============================] - 0s 14ms/step - loss: 0.2464 - val_loss:
0.2562
Epoch 76/100
8/8 [==============================] - 0s 23ms/step - loss: 0.2456 - val_loss:
0.2557
Epoch 77/100
8/8 [==============================] - 0s 17ms/step - loss: 0.2449 - val_loss:
0.2551
Epoch 78/100
8/8 [==============================] - 0s 17ms/step - loss: 0.2442 - val_loss:
0.2546
Epoch 79/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2435 - val_loss:
0.2541
Epoch 80/100
8/8 [==============================] - 0s 17ms/step - loss: 0.2428 - val_loss:
0.2536
Epoch 81/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2421 - val_loss:
0.2531
Epoch 82/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2414 - val_loss:
0.2526
Epoch 83/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2408 - val_loss:
0.2522
Epoch 84/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2401 - val_loss:
```

```
0.2518
Epoch 85/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2395 - val_loss:
0.2513
Epoch 86/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2389 - val_loss:
0.2510
Epoch 87/100
8/8 [==============================] - 0s 17ms/step - loss: 0.2383 - val_loss:
0.2505
Epoch 88/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2378 - val_loss:
0.2502
Epoch 89/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2372 - val_loss:
0.2498
Epoch 90/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2367 - val_loss:
0.2495
Epoch 91/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2361 - val_loss:
0.2491
Epoch 92/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2356 - val_loss:
0.2487
Epoch 93/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2351 - val_loss:
0.2484
Epoch 94/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2347 - val_loss:
0.2481
Epoch 95/100
8/8 [==============================] - 0s 17ms/step - loss: 0.2342 - val_loss:
0.2479
Epoch 96/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2337 - val_loss:
0.2476
Epoch 97/100
8/8 [==============================] - 0s 16ms/step - loss: 0.2333 - val_loss:
0.2473
Epoch 98/100
8/8 [==============================] - 0s 14ms/step - loss: 0.2328 - val_loss:
0.2471
Epoch 99/100
8/8 [==============================] - 0s 15ms/step - loss: 0.2324 - val_loss:
0.2468
Epoch 100/100
8/8 [==============================] - 0s 14ms/step - loss: 0.2320 - val_loss:
```

```
0.2466
16/16 [==============================] - 0s 1ms/step
16/16 [==============================] - 0s 784us/step
MSE | Datos de prueba: 0.07330598304472671
```



6. Implemente un autoencoder profundo agregando más capas internas, y evalúen este nuevo modelo tal como se hizo en los dos modelos anteriores.

[ ]:

```python
encoding_dim = 32

input_img = keras.Input(shape=(x_train.shape[1],))
encoded = layers.Dense(128, activation='relu')(input_img)
encoded = layers.Dense(64, activation='relu')(encoded)
encoded = layers.Dense(encoding_dim, activation='relu')(encoded)

decoded = layers.Dense(64, activation='relu')(encoded)
decoded = layers.Dense(128, activation='relu')(decoded)
decoded = layers.Dense(x_train.shape[1], activation='sigmoid')(decoded)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                epochs=100,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))

# 3. Utilicen algunas imágenes del conjunto de prueba para ver la salida de la capa intermedia
# y de la capa de decodificación.

decoded_imgs = autoencoder.predict(x_test)

n = 5
plt.figure(figsize=(20, 4))
for i in range(n):
```

```python
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

#4. Evalúen el error cuadrático medio entre las imágenes de prueba
# y su correspondiente salida.

from sklearn.metrics import mean_squared_error

mse_test = mean_squared_error(x_test, decoded_imgs)

print(f'MSE | Datos de prueba: {mse_test}')
```

```
Epoch 1/100
8/8 [==============================] - 1s 33ms/step - loss: 0.6765 - val_loss:
0.6087
Epoch 2/100
8/8 [==============================] - 0s 21ms/step - loss: 0.5372 - val_loss:
0.4682
Epoch 3/100
8/8 [==============================] - 0s 18ms/step - loss: 0.4483 - val_loss:
0.4297
Epoch 4/100
8/8 [==============================] - 0s 21ms/step - loss: 0.4244 - val_loss:
0.4141
Epoch 5/100
8/8 [==============================] - 0s 18ms/step - loss: 0.4115 - val_loss:
0.4041
Epoch 6/100
8/8 [==============================] - 0s 20ms/step - loss: 0.4012 - val_loss:
0.3923
Epoch 7/100
8/8 [==============================] - 0s 21ms/step - loss: 0.3904 - val_loss:
0.3809
Epoch 8/100
8/8 [==============================] - 0s 19ms/step - loss: 0.3803 - val_loss:
0.3742
Epoch 9/100
```

```
8/8 [==============================] - 0s 21ms/step - loss: 0.3743 - val_loss:
0.3690
Epoch 10/100
8/8 [==============================] - 0s 20ms/step - loss: 0.3699 - val_loss:
0.3654
Epoch 11/100
8/8 [==============================] - 0s 21ms/step - loss: 0.3659 - val_loss:
0.3607
Epoch 12/100
8/8 [==============================] - 0s 20ms/step - loss: 0.3608 - val_loss:
0.3539
Epoch 13/100
8/8 [==============================] - 0s 21ms/step - loss: 0.3535 - val_loss:
0.3461
Epoch 14/100
8/8 [==============================] - 0s 20ms/step - loss: 0.3465 - val_loss:
0.3399
Epoch 15/100
8/8 [==============================] - 0s 19ms/step - loss: 0.3408 - val_loss:
0.3350
Epoch 16/100
8/8 [==============================] - 0s 21ms/step - loss: 0.3357 - val_loss:
0.3307
Epoch 17/100
8/8 [==============================] - 0s 20ms/step - loss: 0.3310 - val_loss:
0.3267
Epoch 18/100
8/8 [==============================] - 0s 22ms/step - loss: 0.3271 - val_loss:
0.3237
Epoch 19/100
8/8 [==============================] - 0s 21ms/step - loss: 0.3237 - val_loss:
0.3210
Epoch 20/100
8/8 [==============================] - 0s 20ms/step - loss: 0.3206 - val_loss:
0.3182
Epoch 21/100
8/8 [==============================] - 0s 22ms/step - loss: 0.3172 - val_loss:
0.3152
Epoch 22/100
8/8 [==============================] - 0s 20ms/step - loss: 0.3134 - val_loss:
0.3117
Epoch 23/100
8/8 [==============================] - 0s 20ms/step - loss: 0.3094 - val_loss:
0.3075
Epoch 24/100
8/8 [==============================] - 0s 20ms/step - loss: 0.3049 - val_loss:
0.3034
Epoch 25/100
```

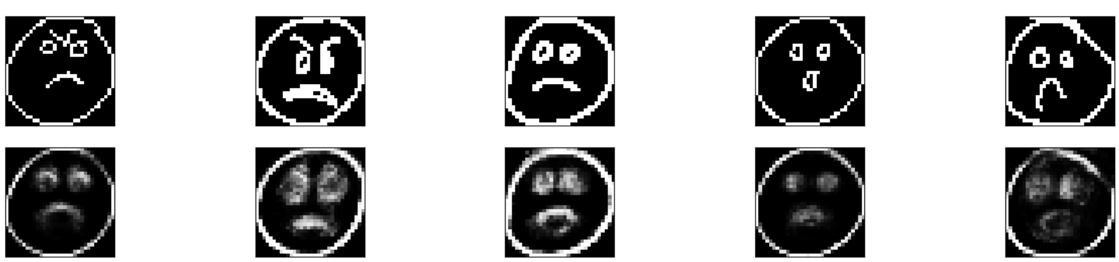
```
8/8 [==============================] - 0s 22ms/step - loss: 0.3007 - val_loss:
0.3001
Epoch 26/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2969 - val_loss:
0.2969
Epoch 27/100
8/8 [==============================] - 0s 22ms/step - loss: 0.2936 - val_loss:
0.2946
Epoch 28/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2909 - val_loss:
0.2926
Epoch 29/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2883 - val_loss:
0.2907
Epoch 30/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2862 - val_loss:
0.2887
Epoch 31/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2840 - val_loss:
0.2870
Epoch 32/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2818 - val_loss:
0.2857
Epoch 33/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2797 - val_loss:
0.2840
Epoch 34/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2776 - val_loss:
0.2822
Epoch 35/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2756 - val_loss:
0.2807
Epoch 36/100
8/8 [==============================] - 0s 24ms/step - loss: 0.2736 - val_loss:
0.2793
Epoch 37/100
8/8 [==============================] - 0s 23ms/step - loss: 0.2717 - val_loss:
0.2782
Epoch 38/100
8/8 [==============================] - 0s 18ms/step - loss: 0.2700 - val_loss:
0.2767
Epoch 39/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2682 - val_loss:
0.2753
Epoch 40/100
8/8 [==============================] - 0s 23ms/step - loss: 0.2664 - val_loss:
0.2740
Epoch 41/100
```

```
8/8 [==============================] - 0s 20ms/step - loss: 0.2649 - val_loss:
0.2734
Epoch 42/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2633 - val_loss:
0.2720
Epoch 43/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2619 - val_loss:
0.2709
Epoch 44/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2603 - val_loss:
0.2704
Epoch 45/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2591 - val_loss:
0.2693
Epoch 46/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2578 - val_loss:
0.2685
Epoch 47/100
8/8 [==============================] - 0s 22ms/step - loss: 0.2565 - val_loss:
0.2673
Epoch 48/100
8/8 [==============================] - 0s 18ms/step - loss: 0.2552 - val_loss:
0.2667
Epoch 49/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2539 - val_loss:
0.2661
Epoch 50/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2527 - val_loss:
0.2650
Epoch 51/100
8/8 [==============================] - 0s 22ms/step - loss: 0.2515 - val_loss:
0.2643
Epoch 52/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2504 - val_loss:
0.2636
Epoch 53/100
8/8 [==============================] - 0s 23ms/step - loss: 0.2492 - val_loss:
0.2632
Epoch 54/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2481 - val_loss:
0.2622
Epoch 55/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2469 - val_loss:
0.2616
Epoch 56/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2458 - val_loss:
0.2608
Epoch 57/100
```

```
8/8 [==============================] - 0s 21ms/step - loss: 0.2449 - val_loss:
0.2602
Epoch 58/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2438 - val_loss:
0.2600
Epoch 59/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2430 - val_loss:
0.2594
Epoch 60/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2423 - val_loss:
0.2590
Epoch 61/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2411 - val_loss:
0.2584
Epoch 62/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2404 - val_loss:
0.2578
Epoch 63/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2396 - val_loss:
0.2580
Epoch 64/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2389 - val_loss:
0.2573
Epoch 65/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2382 - val_loss:
0.2568
Epoch 66/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2373 - val_loss:
0.2570
Epoch 67/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2365 - val_loss:
0.2563
Epoch 68/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2357 - val_loss:
0.2562
Epoch 69/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2350 - val_loss:
0.2557
Epoch 70/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2343 - val_loss:
0.2560
Epoch 71/100
8/8 [==============================] - 0s 22ms/step - loss: 0.2336 - val_loss:
0.2555
Epoch 72/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2329 - val_loss:
0.2549
Epoch 73/100
```

```
8/8 [==============================] - 0s 19ms/step - loss: 0.2321 - val_loss:
0.2549
Epoch 74/100
8/8 [==============================] - 0s 18ms/step - loss: 0.2316 - val_loss:
0.2545
Epoch 75/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2308 - val_loss:
0.2544
Epoch 76/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2300 - val_loss:
0.2541
Epoch 77/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2294 - val_loss:
0.2537
Epoch 78/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2286 - val_loss:
0.2535
Epoch 79/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2278 - val_loss:
0.2533
Epoch 80/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2270 - val_loss:
0.2526
Epoch 81/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2262 - val_loss:
0.2525
Epoch 82/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2256 - val_loss:
0.2527
Epoch 83/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2250 - val_loss:
0.2521
Epoch 84/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2244 - val_loss:
0.2516
Epoch 85/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2234 - val_loss:
0.2516
Epoch 86/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2227 - val_loss:
0.2510
Epoch 87/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2219 - val_loss:
0.2509
Epoch 88/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2214 - val_loss:
0.2507
Epoch 89/100
```

```
8/8 [==============================] - 0s 20ms/step - loss: 0.2208 - val_loss:
0.2510
Epoch 90/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2201 - val_loss:
0.2500
Epoch 91/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2195 - val_loss:
0.2503
Epoch 92/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2191 - val_loss:
0.2503
Epoch 93/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2185 - val_loss:
0.2504
Epoch 94/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2180 - val_loss:
0.2500
Epoch 95/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2174 - val_loss:
0.2499
Epoch 96/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2165 - val_loss:
0.2496
Epoch 97/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2160 - val_loss:
0.2494
Epoch 98/100
8/8 [==============================] - 0s 21ms/step - loss: 0.2155 - val_loss:
0.2492
Epoch 99/100
8/8 [==============================] - 0s 20ms/step - loss: 0.2149 - val_loss:
0.2486
Epoch 100/100
8/8 [==============================] - 0s 19ms/step - loss: 0.2142 - val_loss:
0.2488
16/16 [==============================] - 0s 2ms/step
MSE | Datos de prueba: 0.07815937762367585
```



7. Agreguen ruido binario a algunas imágenes de prueba (es decir, prendan o apaguen algunos pixeles de manera aleatoria), y verifiquen la salida del autoencoder profundo. ¿El modelo es capaz de eliminar ruido en las imágenes de entrada?

[ ]:
```
x_train = np.reshape(x_train, (len(x_train), 32, 32, 1))
x_test = np.reshape(x_test, (len(x_test), 32, 32, 1))

noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

n = 5
plt.figure(figsize=(20, 2))
for i in range(1, n + 1):
    ax = plt.subplot(1, n, i)
    plt.imshow(x_test_noisy[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

[ ]:
```
from keras.callbacks import TensorBoard



input_img = keras.Input(shape=(32, 32, 1))

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
```

```
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train_noisy, x_train,
                epochs=100,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test_noisy, x_test),
                callbacks=[TensorBoard(log_dir='/tmp/tb', histogram_freq=0,␣
 ↪write_graph=False)])
```

```
Epoch 1/100
16/16 [==============================] - 5s 239ms/step - loss: 0.5537 -
val_loss: 0.5131
Epoch 2/100
16/16 [==============================] - 4s 227ms/step - loss: 0.5082 -
val_loss: 0.4967
Epoch 3/100
16/16 [==============================] - 4s 241ms/step - loss: 0.4976 -
val_loss: 0.4848
Epoch 4/100
16/16 [==============================] - 4s 263ms/step - loss: 0.4840 -
val_loss: 0.4649
Epoch 5/100
16/16 [==============================] - 4s 265ms/step - loss: 0.4557 -
val_loss: 0.4291
Epoch 6/100
16/16 [==============================] - 4s 276ms/step - loss: 0.4150 -
val_loss: 0.3865
Epoch 7/100
16/16 [==============================] - 4s 229ms/step - loss: 0.3742 -
val_loss: 0.3520
Epoch 8/100
16/16 [==============================] - 4s 249ms/step - loss: 0.3438 -
val_loss: 0.3273
Epoch 9/100
16/16 [==============================] - 4s 264ms/step - loss: 0.3221 -
val_loss: 0.3106
Epoch 10/100
16/16 [==============================] - 4s 245ms/step - loss: 0.3052 -
val_loss: 0.2966
Epoch 11/100
16/16 [==============================] - 4s 244ms/step - loss: 0.2923 -
val_loss: 0.2843
Epoch 12/100
16/16 [==============================] - 5s 285ms/step - loss: 0.2811 -
```

```
val_loss: 0.2775
Epoch 13/100
16/16 [==============================] - 4s 259ms/step - loss: 0.2719 -
val_loss: 0.2655
Epoch 14/100
16/16 [==============================] - 4s 246ms/step - loss: 0.2636 -
val_loss: 0.2582
Epoch 15/100
16/16 [==============================] - 4s 254ms/step - loss: 0.2563 -
val_loss: 0.2516
Epoch 16/100
16/16 [==============================] - 4s 230ms/step - loss: 0.2517 -
val_loss: 0.2466
Epoch 17/100
16/16 [==============================] - 4s 240ms/step - loss: 0.2477 -
val_loss: 0.2432
Epoch 18/100
16/16 [==============================] - 4s 250ms/step - loss: 0.2441 -
val_loss: 0.2395
Epoch 19/100
16/16 [==============================] - 4s 235ms/step - loss: 0.2383 -
val_loss: 0.2366
Epoch 20/100
16/16 [==============================] - 4s 234ms/step - loss: 0.2355 -
val_loss: 0.2327
Epoch 21/100
16/16 [==============================] - 4s 231ms/step - loss: 0.2322 -
val_loss: 0.2326
Epoch 22/100
16/16 [==============================] - 4s 228ms/step - loss: 0.2294 -
val_loss: 0.2275
Epoch 23/100
16/16 [==============================] - 4s 227ms/step - loss: 0.2259 -
val_loss: 0.2241
Epoch 24/100
16/16 [==============================] - 4s 241ms/step - loss: 0.2227 -
val_loss: 0.2216
Epoch 25/100
16/16 [==============================] - 4s 243ms/step - loss: 0.2205 -
val_loss: 0.2205
Epoch 26/100
16/16 [==============================] - 4s 231ms/step - loss: 0.2174 -
val_loss: 0.2171
Epoch 27/100
16/16 [==============================] - 4s 227ms/step - loss: 0.2157 -
val_loss: 0.2138
Epoch 28/100
16/16 [==============================] - 4s 228ms/step - loss: 0.2123 -
```

```
val_loss: 0.2120
Epoch 29/100
16/16 [==============================] - 4s 228ms/step - loss: 0.2102 -
val_loss: 0.2103
Epoch 30/100
16/16 [==============================] - 4s 229ms/step - loss: 0.2079 -
val_loss: 0.2076
Epoch 31/100
16/16 [==============================] - 4s 230ms/step - loss: 0.2060 -
val_loss: 0.2058
Epoch 32/100
16/16 [==============================] - 4s 252ms/step - loss: 0.2049 -
val_loss: 0.2055
Epoch 33/100
16/16 [==============================] - 4s 237ms/step - loss: 0.2034 -
val_loss: 0.2030
Epoch 34/100
16/16 [==============================] - 4s 232ms/step - loss: 0.2024 -
val_loss: 0.2054
Epoch 35/100
16/16 [==============================] - 4s 230ms/step - loss: 0.2009 -
val_loss: 0.2001
Epoch 36/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1987 -
val_loss: 0.2001
Epoch 37/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1976 -
val_loss: 0.1987
Epoch 38/100
16/16 [==============================] - 4s 229ms/step - loss: 0.1967 -
val_loss: 0.1976
Epoch 39/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1953 -
val_loss: 0.1965
Epoch 40/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1940 -
val_loss: 0.1948
Epoch 41/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1930 -
val_loss: 0.1939
Epoch 42/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1919 -
val_loss: 0.1944
Epoch 43/100
16/16 [==============================] - 4s 231ms/step - loss: 0.1925 -
val_loss: 0.1956
Epoch 44/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1904 -
```

```
val_loss: 0.1929
Epoch 45/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1905 -
val_loss: 0.1916
Epoch 46/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1896 -
val_loss: 0.1918
Epoch 47/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1879 -
val_loss: 0.1890
Epoch 48/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1868 -
val_loss: 0.1883
Epoch 49/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1866 -
val_loss: 0.1876
Epoch 50/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1860 -
val_loss: 0.1869
Epoch 51/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1846 -
val_loss: 0.1860
Epoch 52/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1842 -
val_loss: 0.1856
Epoch 53/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1833 -
val_loss: 0.1849
Epoch 54/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1827 -
val_loss: 0.1850
Epoch 55/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1822 -
val_loss: 0.1844
Epoch 56/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1822 -
val_loss: 0.1838
Epoch 57/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1809 -
val_loss: 0.1830
Epoch 58/100
16/16 [==============================] - 4s 223ms/step - loss: 0.1801 -
val_loss: 0.1823
Epoch 59/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1791 -
val_loss: 0.1819
Epoch 60/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1785 -
```

```
val_loss: 0.1825
Epoch 61/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1790 -
val_loss: 0.1832
Epoch 62/100
16/16 [==============================] - 4s 223ms/step - loss: 0.1782 -
val_loss: 0.1817
Epoch 63/100
16/16 [==============================] - 4s 223ms/step - loss: 0.1772 -
val_loss: 0.1799
Epoch 64/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1771 -
val_loss: 0.1832
Epoch 65/100
16/16 [==============================] - 4s 228ms/step - loss: 0.1781 -
val_loss: 0.1811
Epoch 66/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1766 -
val_loss: 0.1785
Epoch 67/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1750 -
val_loss: 0.1779
Epoch 68/100
16/16 [==============================] - 4s 223ms/step - loss: 0.1748 -
val_loss: 0.1779
Epoch 69/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1744 -
val_loss: 0.1777
Epoch 70/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1743 -
val_loss: 0.1826
Epoch 71/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1743 -
val_loss: 0.1770
Epoch 72/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1732 -
val_loss: 0.1775
Epoch 73/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1729 -
val_loss: 0.1761
Epoch 74/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1729 -
val_loss: 0.1766
Epoch 75/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1726 -
val_loss: 0.1753
Epoch 76/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1715 -
```

```
val_loss: 0.1751
Epoch 77/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1714 -
val_loss: 0.1766
Epoch 78/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1714 -
val_loss: 0.1742
Epoch 79/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1705 -
val_loss: 0.1741
Epoch 80/100
16/16 [==============================] - 4s 223ms/step - loss: 0.1698 -
val_loss: 0.1734
Epoch 81/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1697 -
val_loss: 0.1735
Epoch 82/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1699 -
val_loss: 0.1730
Epoch 83/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1695 -
val_loss: 0.1730
Epoch 84/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1689 -
val_loss: 0.1740
Epoch 85/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1689 -
val_loss: 0.1723
Epoch 86/100
16/16 [==============================] - 4s 223ms/step - loss: 0.1680 -
val_loss: 0.1742
Epoch 87/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1687 -
val_loss: 0.1725
Epoch 88/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1680 -
val_loss: 0.1729
Epoch 89/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1670 -
val_loss: 0.1711
Epoch 90/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1666 -
val_loss: 0.1714
Epoch 91/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1666 -
val_loss: 0.1709
Epoch 92/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1666 -
```

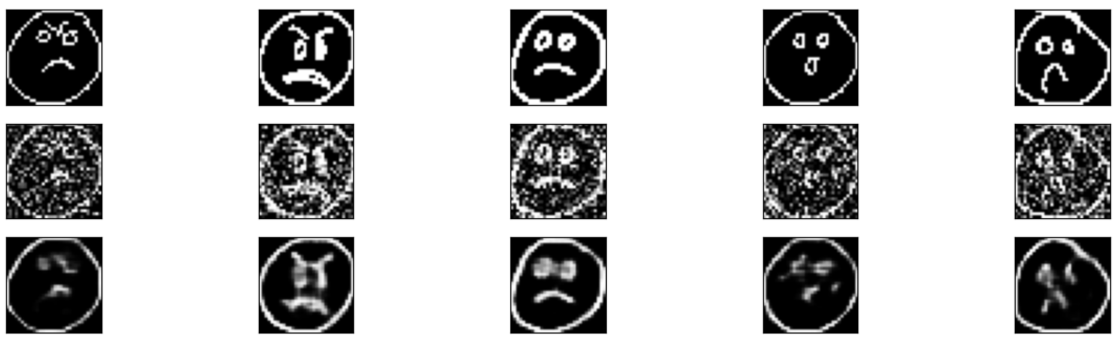
```
val_loss: 0.1726
Epoch 93/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1661 -
val_loss: 0.1703
Epoch 94/100
16/16 [==============================] - 4s 228ms/step - loss: 0.1656 -
val_loss: 0.1705
Epoch 95/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1655 -
val_loss: 0.1702
Epoch 96/100
16/16 [==============================] - 4s 224ms/step - loss: 0.1652 -
val_loss: 0.1700
Epoch 97/100
16/16 [==============================] - 4s 225ms/step - loss: 0.1657 -
val_loss: 0.1715
Epoch 98/100
16/16 [==============================] - 4s 227ms/step - loss: 0.1654 -
val_loss: 0.1693
Epoch 99/100
16/16 [==============================] - 4s 226ms/step - loss: 0.1643 -
val_loss: 0.1688
Epoch 100/100
16/16 [==============================] - 4s 240ms/step - loss: 0.1643 -
val_loss: 0.1720
```

[ ]:
```
<keras.src.callbacks.History at 0x26625c0aaf0>
```

[ ]:
```python
decoded_imgs = autoencoder.predict(x_test_noisy)

n = 5  # número de imágenes a mostrar
plt.figure(figsize=(15, 4))
for i in range(n):
    # Muestra la imagen original
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Muestra la imagen ruidosa
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(x_test_noisy[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```

```
    # Muestra la imagen reconstruida
    ax = plt.subplot(3, n, i + 1 + 2*n)
    plt.imshow(decoded_imgs[i].reshape(32, 32))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```

```
16/16 [==============================] - 0s 16ms/step
```



¿El modelo es capaz de eliminar ruido en las imágenes de entrada?

Si, el modelo es capaz de eliminar el ruido de las imágenes, esto se debe a que gracias a nuestro modelo es capaz de aprender a extraer las caracteristicas esenciales de las imágenes originales y a reconstruirlas a una imagen más limpia en base a ello.

Cabe mencionar que nuestro modelo usa capas convolucionales, que son capaces de extraer características visuales de alto nivel de las imágenes, y que son más eficientes y robustas que las capas densas. Las capas convolucionales también preservan la estructura espacial de las imágenes, lo que facilita la reconstrucción.