ceti

CENTRO DE ENSEÑANZA
TÉCNICA INDUSTRIAL

INTELIGENCIA ARTIFICIAL

Saul Armando Cuenca Martínez
21110324

"Proyecto IA"

Parte Teórica

1.- ¿Qué tutorial escogieron?-

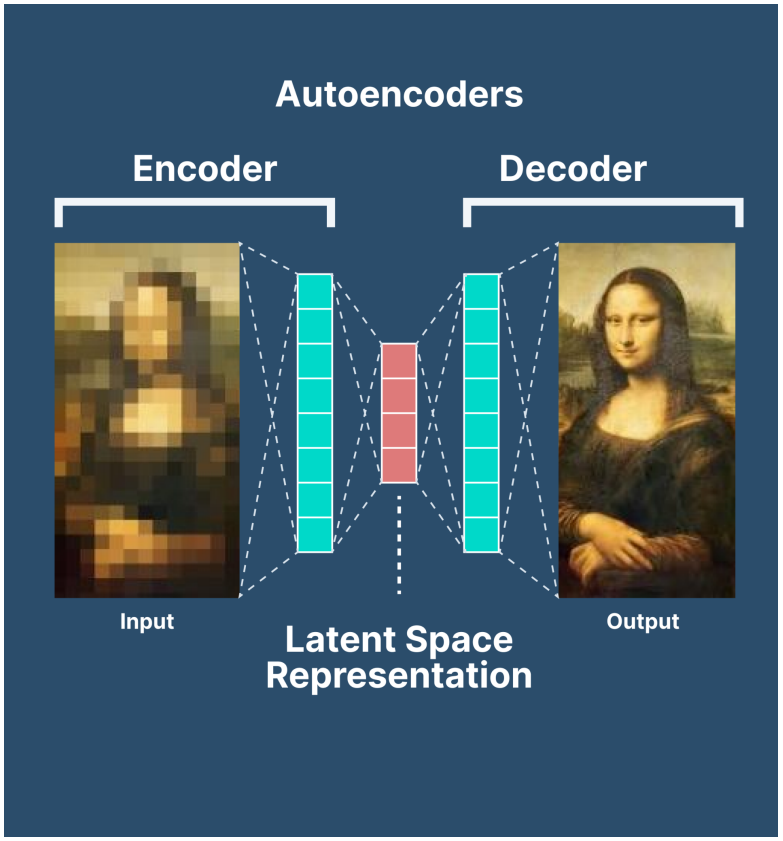
Learning about autoencoders with Python, Tensorflow and Keras

(Aprendiendo sobre codificadores automáticos con Python, Tensor Flow y Keras)

2.- ¿Por qué lo escogiste e información complementaria?

Los codificadores automáticos son un tipo de algoritmo de aprendizaje profundo que están diseñados principalmente para recibir una entrada y transformarla en una representación diferente. Además, tiene un papel fundamental en la construcción de la imagen.

Codificadores Automáticos

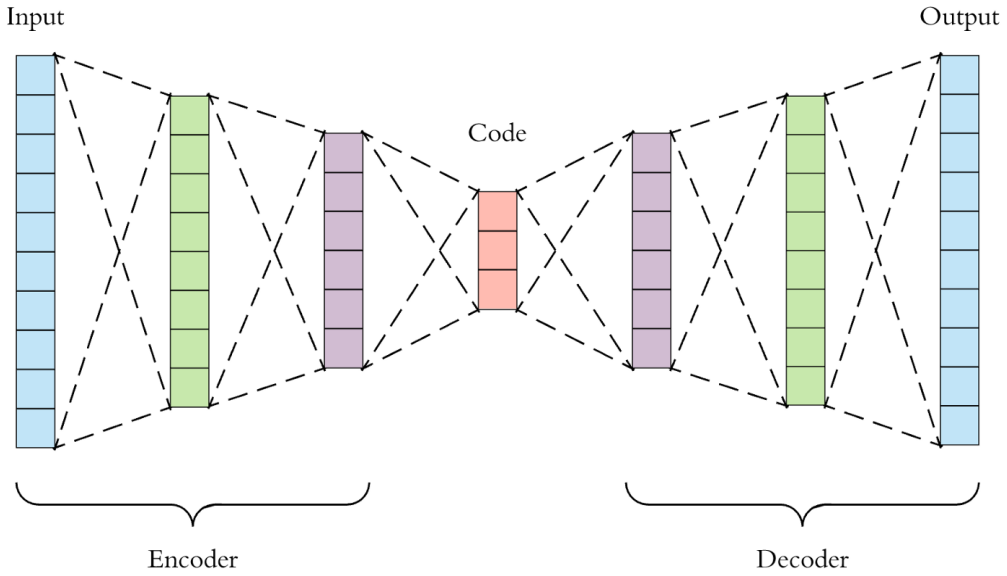


Los codificadores automáticos son de gran utilidad en el aprendizaje automático no supervisado, puesto que se pueden usar para comprimir los datos y reducir su dimensionalidad. La principal diferencia entre los codificadores automáticos y el análisis de componentes principales (PCA) es que mientras PCA encuentra las direcciones a lo largo de las cuales puede proyectar los datos con la variación máxima, los codificadores automáticos reconstruyen la entrada original con solo una versión comprimida de la misma.

Arquitectura

Un codificador automático es un tipo de red neuronal que puede aprender a reconstruir imágenes, texto y otros datos a partir de versiones comprimidas de sí mismos. Un codificador automático consta de tres capas, que son las siguientes:

- **Codificador:** Esta capa comprime la imagen de entrada en una representación de espacio latente, aquí se codifica la imagen de entrada como una representación comprimida en una dimensión reducida y la imagen comprimida es una versión distorsionada de la imagen original.
- **Código:** La capa de código representa la entrada comprimida alimentada a la capa del decodificador.
- **Decodificador:** La capa del decodificador como su nombre lo dice decodifica la imagen codificada de vuelta a la dimensión original. La imagen decodificada se reconstruye a partir de la representación del espacio latente, y se reconstruye a partir de la representación del espacio latente y es una reconstrucción con pérdidas de la imagen original.



Tipos de codificadores automáticos

Codificadores automáticos completos

Debajo de los codificadores automáticos completos hay una red neuronal no supervisada que se puede usar para generar una versión comprimida de los datos de entrada. Se realiza tomando una imagen e intentando predecir la misma imagen como salida, reconstruyendo así la imagen a partir de su región de cuello de botella comprimida.

Codificadores automáticos dispersos

Los codificadores automáticos dispersos se controlan cambiando el número de nodos en cada capa oculta. Significa que se aplica a la función de pérdida una penalización directamente proporcional al número de neuronas activadas.

Codificadores automáticos Contractual
La entrada pasa a través de un cuello de botella en un autocodificador contractivo y luego se reconstruye en el decodificador. La función de cuello de botella se usa para aprender una representación de la imagen mientras la pasa.

Codificadores automáticos de eliminación de ruido
Los codificadores automáticos de eliminación de ruido son similares a los codificadores automáticos regulares en que toman una entrada y producen una salida. Sin embargo, se diferencian porque no tienen la imagen de entrada como verdad básica.

Codificadores automáticos variacionales
Los codificadores automáticos variacionales (VAEs) son modelos que abordan un problema específico con los codificadores automáticos estándar. Cuando entrena un codificador automático, aprende a representar la entrada solo en una forma comprimida llamada espacio latente o cuello de botella. Sin embargo, este espacio latente formado después del entrenamiento no es necesariamente continuo y, en efecto, podría no ser fácil de interpolar.

Este tema me pareció muy interesante, ya había visto algo en clase de Visión artificial, pero quería más adentrarme al tema.

3.- ¿Es monetizable?

La monetización de los autoencoders (codificadores automáticos) puede depender del contexto y del uso específico que les des. Aquí hay algunas formas en las que los autoencoders podrían ser monetizados:

> **Desarrollo de Modelos y Soluciones Personalizadas:** Si eres capaz de desarrollar modelos de autoencoder personalizados para problemas específicos, podrías ofrecer servicios de consultoría o desarrollo a empresas que necesiten soluciones de aprendizaje automático para sus datos.

Desarrollo de Productos y Aplicaciones: Puedes crear productos o aplicaciones basados en autoencoders para resolver problemas específicos. Por ejemplo, podrías desarrollar una aplicación de mejoramiento de imágenes, compresión de datos, o generación de contenido.

Cursos y Consultoría en Aprendizaje Automático: Si tienes experiencia en la implementación y comprensión de autoencoders, podrías ofrecer cursos en línea, tutoriales o servicios de consultoría para enseñar a otros cómo utilizar y entender estos modelos.

Investigación y Desarrollo: Si estás involucrado en la investigación de vanguardia en el campo del aprendizaje profundo y autoencoders, podrías buscar oportunidades de financiamiento para proyectos de investigación o colaboraciones con instituciones académicas y empresas.

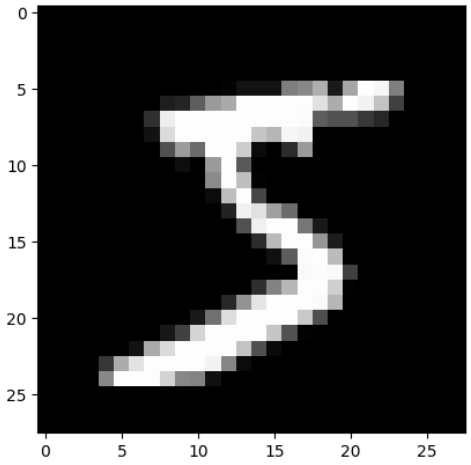
Desarrollo de Herramientas y Bibliotecas: Si creas herramientas, bibliotecas o frameworks que facilitan el uso de autoencoders, podrías monetizarlos ofreciendo licencias, soporte premium o servicios adicionales.

4.- Mostrar el ejemplo aplicado

A continuación muestro imagenes de como quedo mi proyecto realizado a base de información y tutoriales presentados por nuestro profesor, me gusto ver que cada vez que modifico me puede brindar el número o la medida que yo quiera de cualquier número como si lo quiero que se percate bien o sea desenfocado entre otros.

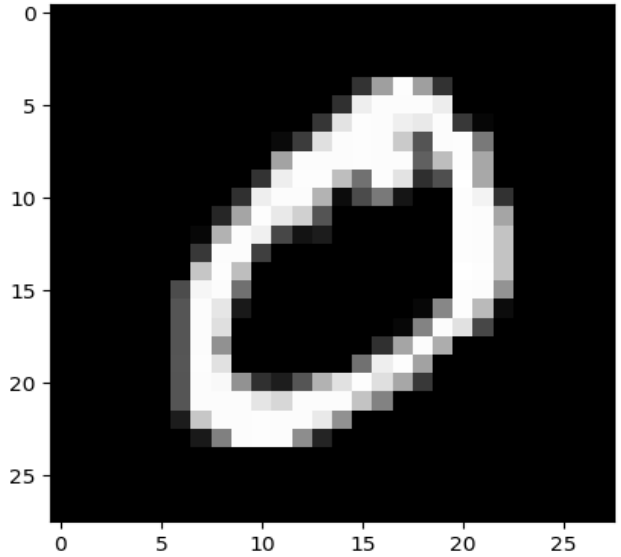
```
import tensorflow as tf
from tensorflow import keras
import cv2
import numpy as np
import matplotlib.pyplot as plt


(x_train, y_train),(x_test, y_test) = tf.keras.datasets.mnist.load_data()  # loads the popular "mnist" training dataset

plt.imshow(x_train[0], cmap="gray")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
<matplotlib.image.AxesImage at 0x7f37d65cfe80>
```




```
[3]  plt.imshow(x_train[1], cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f37d30a3b50>
```




```
[4]  x_train[0].shape
```

```
(28, 28)
```

```
[5]  28*28
```

```
784
```

```python
x_train[0]
```

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
        253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
        205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
         90,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 139, 253,
        190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
        253,  70,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  35,
        241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  16,  93, 252, 253, 187,   0,   0,   0,   0,   0,   0,
          0,   0],
```

```
  81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,  16,  93, 252, 253, 187,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0, 249, 253, 249,  64,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
 148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
 253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
 253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
 195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
  11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0]], dtype=uint8)
```

```
[7]  x_train = x_train/255.0
     x_test = x_test/255.0
```

```
[43] x_train = x_train/255.0
```

```
[8]  x_train[0]
```

```
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.18039216,
        0.50980392, 0.71764706, 0.99215686, 0.99215686, 0.81176471,
        0.00784314, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.15294118, 0.58039216, 0.89803922,
        0.99215686, 0.99215686, 0.99215686, 0.98039216, 0.71372549,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.09411765, 0.44705882, 0.86666667, 0.99215686, 0.99215686,
        0.99215686, 0.99215686, 0.78823529, 0.30588235, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.09019608, 0.25882353,
        0.83529412, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
        0.77647059, 0.31764706, 0.00784314, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.07058824, 0.67058824, 0.85882353, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.76470588, 0.31372549,
        0.03529412, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.21568627,
        0.6745098 , 0.88627451, 0.99215686, 0.99215686, 0.99215686,
        0.99215686, 0.95686275, 0.52156863, 0.04313725, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.53333333,
        0.99215686, 0.99215686, 0.99215686, 0.83137255, 0.52941176,
        0.51764706, 0.0627451 , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
[33] encoder_input = keras.Input(shape=(28, 28, 1), name='img')
     x = keras.layers.Flatten()(encoder_input)
     encoder_output = keras.layers.Dense(64, activation="relu")(x)

     encoder = keras.Model(encoder_input, encoder_output, name="encoder")


     decoder_input = keras.layers.Dense(64, activation="relu")(encoder_output)
     x = keras.layers.Dense(784, activation="relu")(decoder_input)
     decoder_output = keras.layers.Reshape((28, 28, 1))(x)

     optimizer = tf.keras.optimizers.legacy.Adam(lr=0.001)

     autoencoder = keras.Model(encoder_input, decoder_output, name='autoencoder')
     autoencoder.summary()
```

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_3 (Flatten)         (None, 784)               0

 dense_8 (Dense)             (None, 64)                50240

 dense_9 (Dense)             (None, 64)                4160

 dense_10 (Dense)            (None, 784)               50960

 reshape_3 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 105360 (411.56 KB)
Trainable params: 105360 (411.56 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/legacy/adam.py:118: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

```
[34] from tensorflow.keras.optimizers import Adam
     from tensorflow.keras.losses import MeanSquaredError
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense

     # Supongamos que ya tienes tu modelo de autoencoder definido
     autoencoder = Sequential()
     # Agrega tus capas al modelo de autoencoder...

     # Define el optimizador
     opt = Adam(learning_rate=0.001)  # Puedes ajustar la tasa de aprendizaje según sea necesario

     # Compila el modelo de autoencoder
     autoencoder.compile(optimizer=opt, loss='mse')
```

```
[35] autoencoder.fit(x_train, x_train, epochs=3, batch_size=32, validation_split=0.1)
```

```
Epoch 1/3
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 2/3
1688/1688 [==============================] - 3s 2ms/step - loss: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 3/3
1688/1688 [==============================] - 3s 2ms/step - loss: 0.0000e+00 - val_loss: 0.0000e+00
<keras.src.callbacks.History at 0x7f37a7bb3130>
```

```
[36] example = encoder.predict([x_test[0].reshape(-1, 28, 28, 1)])[0]

     print(example)
```

```
1/1 [==============================] - 0s 49ms/step
[0.34018794 0.         0.         0.         0.         0.40422496
 0.47452575 0.         0.         0.         0.         0.
 0.53501034 0.         0.         0.         0.41138408 0.
 0.         0.5155208  0.         0.         0.5166345  0.1934535
 0.23864795 0.9870006  1.0115916  0.6335997  0.         0.0075147
 0.         0.         0.6354762  0.7625333  0.08451418 0.
 0.41985297 0.04934935 0.0019522  0.         0.10304391 0.6382737
 0.53851306 0.         0.         0.         0.         0.
 0.         0.         0.62875974 0.         0.6331482  0.3629265
 0.         0.         0.38029203 0.309655   0.         0.24464896
 0.         0.13954878 0.         0.04805906]
```

```
[17] example.shape
```

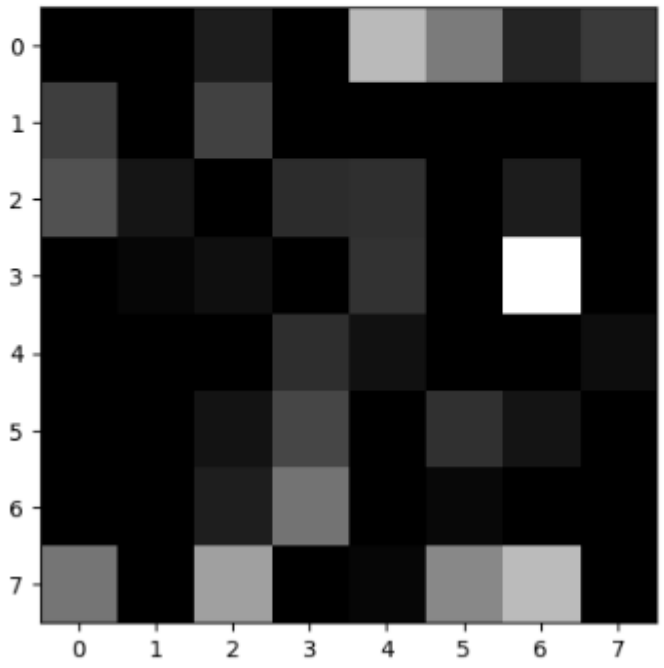
```
(64,)
```

```
[37] 64/784
```

```
0.08163265306122448
```

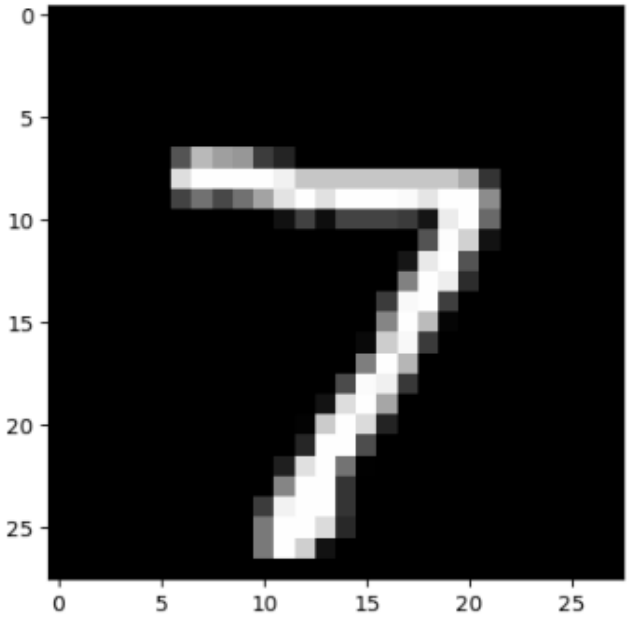
```
[23] plt.imshow(example.reshape((8,8)), cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f37a7fe1330>
```



```
[21] plt.imshow(x_test[0], cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f37a53a3160>
```




```
[22] ae_out = autoencoder.predict([x_test[0].reshape(-1, 28, 28, 1)])[0]
```

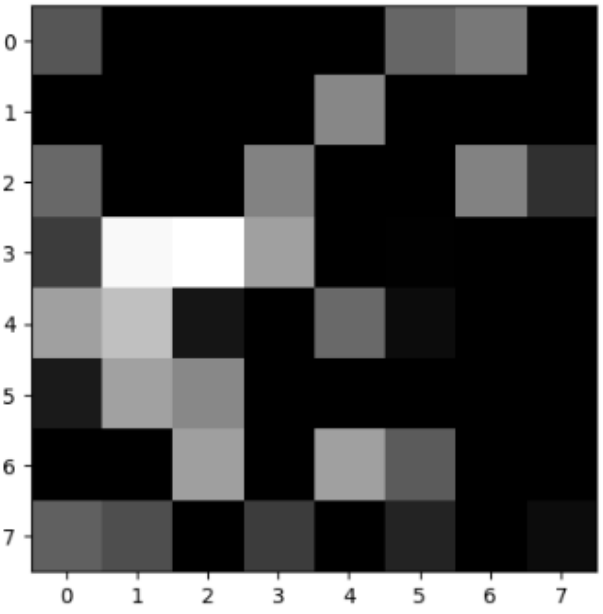
```
1/1 [==============================] - 0s 39ms/step
```


```
[22] ae_out = autoencoder.predict([x_test[0].reshape(-1, 28, 28, 1)])[0]
```

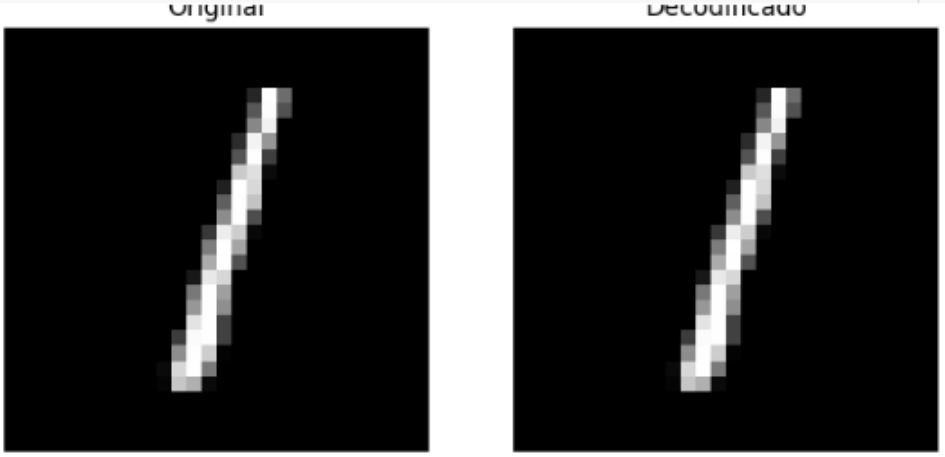
```
1/1 [==============================] - 0s 39ms/step
```


```
[46] plt.imshow(example.reshape((8,8)), cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f37a7c64f70>
```

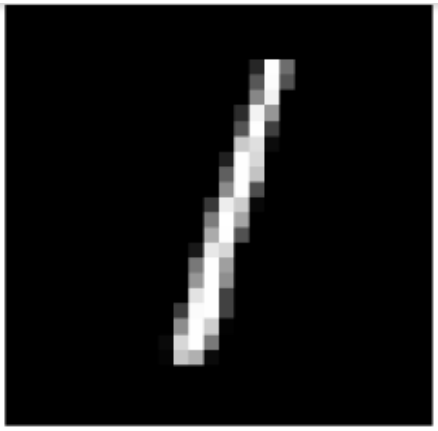


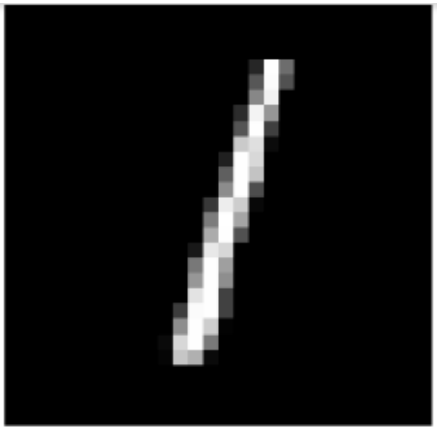
```
[50] import matplotlib.pyplot as plt

     # Supongamos que x_test es tu conjunto de datos de prueba

     for d in x_test[:5]:  # Muestra solo 5 ejemplos, siéntete libre de mostrar más o menos según de
         # Ajusta las dimensiones si es necesario y realiza la predicción
         ae_out = autoencoder.predict(d.reshape(-1, 28, 28, 1))
         img = ae_out[0]

         # Muestra las imágenes utilizando Matplotlib
         plt.figure(figsize=(8, 4))

         # Imagen original
         plt.subplot(1, 2, 1)
         plt.imshow(d.reshape(28, 28), cmap='gray')
         plt.title('Original')
         plt.axis('off')

         # Imagen decodificada por el autoencoder
         plt.subplot(1, 2, 2)
         plt.imshow(img.reshape(28, 28), cmap='gray')
         plt.title('Decodificado')
         plt.axis('off')

         # Muestra la figura
         plt.show()
```




```
1/1 [==============================] - 0s 69ms/step
```

[50]

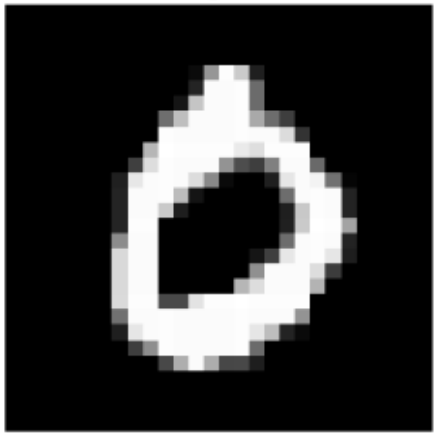


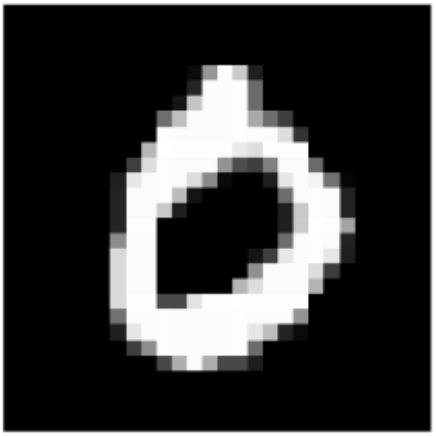


```
1/1 [==============================] - 0s 69ms/step
```

Original

Decodificado





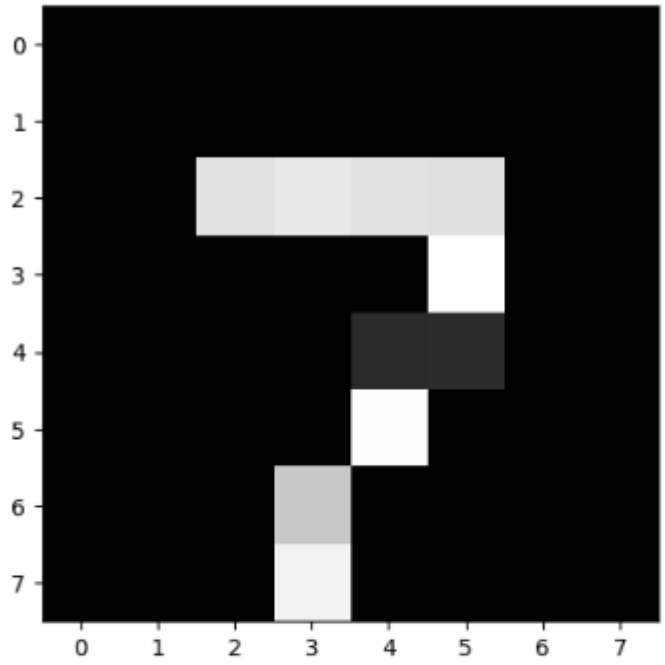
```
1/1 [==============================] - 0s 39ms/step
```

Original

Decodificado

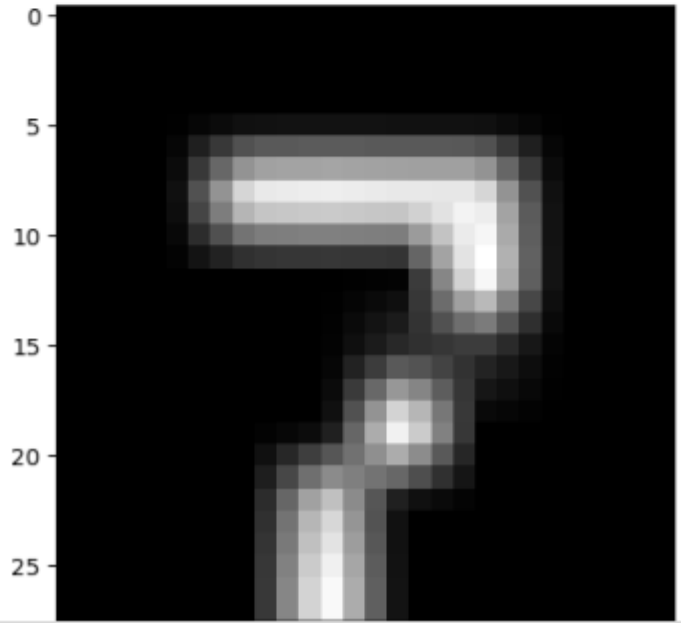
```
[51] smaller = cv2.resize(x_test[0], (8,8))
     back_to_original = cv2.resize(smaller, (28,28))
     plt.imshow(smaller, cmap="gray")
```

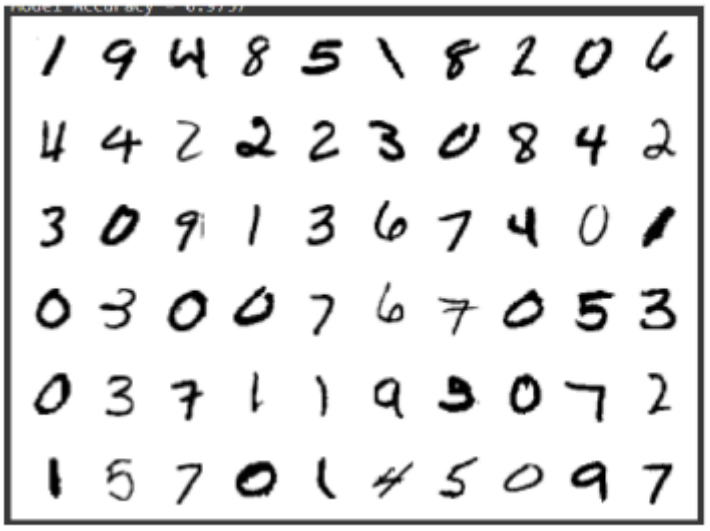
```
<matplotlib.image.AxesImage at 0x7f37a7a957e0>
```

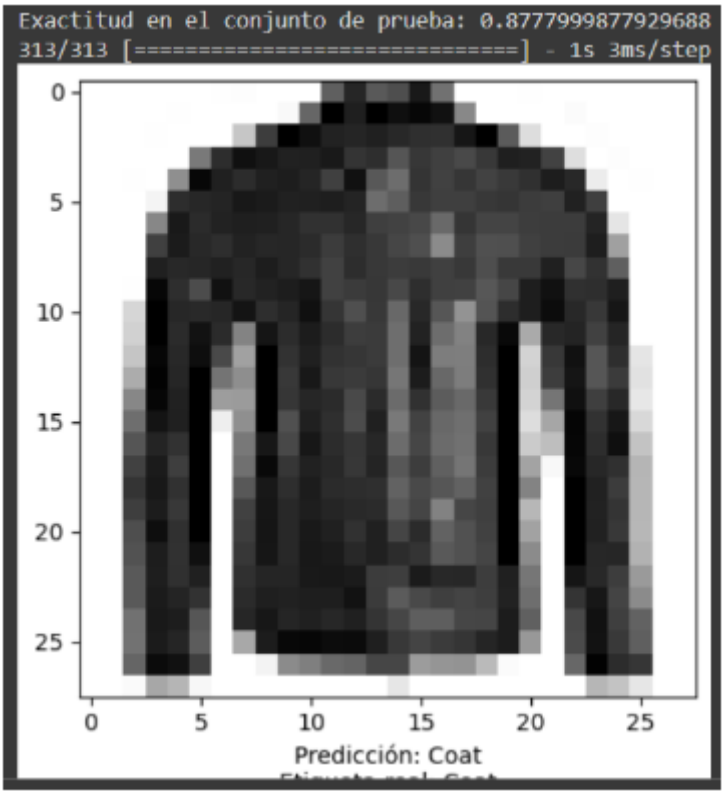



```
[52] plt.imshow(back_to_original, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f37a03c49a0>
```







5.- Repositorio de Github

En el repositorio se muestra el programa en .py y .ipynb recomiendo verlo en google colab para que muestre imagen por imagen y se percatan las diferencias

Enlace:
https://github.com/SaulCuenca/SaulCuenca-IA-P3

6.- Videotutorial explicando la aplicación realizada o el experimento.

Enlace del video:
https://drive.google.com/file/d/1pSLK1YPEIKA-pLypnUaqi0BKjx0S8ETA/view?usp=sharing

Enlace del Tik Tok:
https://vm.tiktok.com/ZM6NjLFDu/