UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSI

FACULTAD DE INGENIERÍA

ÁREA MECANICA ELÉCTRICA

INTELIGENCIA ARTIFICIAL APLICADA

PROYECTO PRIMER PARCIAL

Integrante: Ramírez Galván Manuel

Integrante: León Álvarez Cristian Stiven

Profesor: Sánchez Flores Alejandra

Semestre: 2024-2025/II

Fecha: 05/03/2025

Índice

Ilustraciones

1. Introducción

La inteligencia artificial busca desarrollar computadoras, robots o sistemas capaces de pensar de manera similar a los seres humanos. Se basa en el estudio de cómo funciona el cerebro al procesar información, aprender, tomar decisiones y resolver problemas. A partir de este conocimiento, se crean programas y algoritmos inteligentes. Su principal objetivo es mejorar las capacidades informáticas en áreas como el razonamiento, el aprendizaje y la solución de problemas, acercándolas cada vez más a la forma en que los humanos piensan y actúan.[1]

La IA se compone de:

- Razonamiento
- Aprendizaje
- Resolución de problemas
- Percepción
- Inteligencia lingüística

Por su amplio alcance, la inteligencia artificial se ha convertido en una tecnología clave en diversos ámbitos, desde la automatización de tareas en la industria hasta el desarrollo de asistentes virtuales y diagnósticos médicos avanzados. Su capacidad para analizar grandes volúmenes de datos y adaptarse a nuevas situaciones la hace cada vez más relevante en la vida cotidiana. A medida que la IA evoluciona, surgen tanto oportunidades como desafíos, incluyendo cuestiones éticas, de privacidad y el impacto en el empleo. Sin embargo, su desarrollo sigue avanzando con el objetivo de mejorar la eficiencia, la toma de decisiones y la interacción entre humanos y máquinas.[2]

TensorFlow:

TensorFlow es una biblioteca de código abierto utilizada para el desarrollo y entrenamiento de modelos de machine learning y deep learning. Proporciona herramientas y recursos para construir redes neuronales complejas, y es ampliamente utilizada tanto en investigación como en la industria.[3]

Keras:

Keras es una API de alto nivel para crear y entrenar modelos de deep learning, que funciona sobre TensorFlow. Es conocida por su simplicidad y facilidad de uso, lo que permite a los desarrolladores construir modelos rápidamente sin profundizar en detalles complejos.[4]

Numpy:

NumPy es una librería fundamental para el cálculo numérico en Python. Ofrece estructuras de datos eficientes, como arrays multidimensionales, y una amplia gama de funciones matemáticas que son utilizadas en la manipulación de grandes volúmenes de datos.[5]

Matplotlib:

Matplotlib es una biblioteca de visualización en Python que permite crear gráficos estáticos, interactivos y animados. Es especialmente útil para generar visualizaciones de datos, como diagramas de dispersión, histogramas y gráficos de líneas, facilitando el análisis y la interpretación visual de los resultados.[6]

2. Desarrollo

2.1.Código Utilizado

A continuación, se detallará el código dividiéndolo en diferentes etapas.

2.1.1. Importando Librerías.

Se inicia con la importación de librerías y en este caso se hizo uso de las siguientes:

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

2.1.2. Cargando el conjunto de datos.

En esta etapa se cargan los datos, mediante la Liberia numpy ya que son archivos con extensión ".npy" y se asignan a las variables correspondientes.

```
# Cargando el conjunto de datos
X_in = np.load('/content/drive/MyDrive/Colab Notebooks/Inteligencia/Teoria/X.npy')
y_in = np.load('/content/drive/MyDrive/Colab Notebooks/Inteligencia/Teoria/y.npy')
```

2.1.3. Limitando la cantidad de datos

Se limita el conjunto de datos únicamente a los 1000 primeros que corresponden con los "0" y "1".

```
X=X_in[0:1000]
y=y_in[0:1000]
```

2.1.4. Verificando la dimensión del arreglo.

Mediante la instrucción "shape" verificamos la dimensión del arreglo para tener certeza que realmente se hayan cargado únicamente los 1000 primeros datos, así como la información asociada los pixeles de la imagen

```
print(X.shape)
print(y.shape)
```

Luego de ejecutar la anterior instrucción se obtuvo lo siguiente

```
(1000, 400)
(1000, 1)
```

Recordemos que la resolución de la imagen es de 20 x 20, lo que da como resultado información de 400 pixeles, de aquí que tengamos los 1000 datos asociados a 1000 filas y 400 columnas asociadas a la información de cada píxel.

2.1.5. Visualización de imágenes.

Se define una función que nos permite visualizar las imágenes, donde se reciben 3 parámetros que son el arreglo con la información de las imágenes, así como la cantidad de datos a mostrar, luego de esto con la cantidad de datos a mostrar se hace el cálculo de cuantas imágenes serás las visualizadas, seguido de esto se redimensionan a 20 x 20 y finalmente retorna la muestra de las imágenes.

```
#Función para visualizar imágenes
def visualizar_imagenes(arreglo, j,k):
    #asegurese de que el número de imágenes no exceda el tamaño del arreglo
    num_imagenes=min((k-j), arreglo.shape[0])
    #crear una figura para mostrar las imágenes
    fig,axes=plt.subplots(1,(k-j), figsize=(15,3))
    #recorrer y mostrar las imágenes
    for i in range(j,k):
        #obtener la imagen i-ésima y redimensionarla a 20x20
        imagen=arreglo[i].reshape(20,20)
        #mostrar la imagen
        axes[i-j].imshow(imagen, cmap='grey')#usar 'grey' para imágenes

    plt.show()
```

Luego de esto se llama la función para visualizar 10 elementos para verificar que efectivamente los datos correspondan a “0” y “1”.

```
visualizar_imagenes(X,990,1000)
```

El resultado de llamar la función fue el siguiente

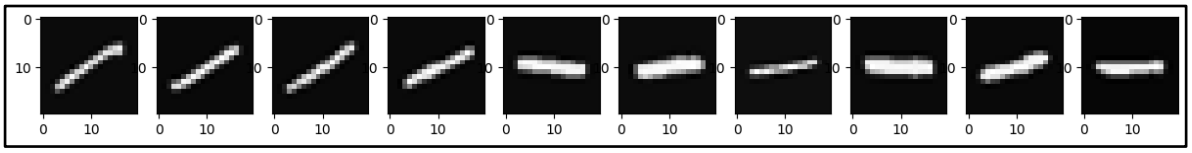

Ilustración 1. Visualización de las imágenes.

2.1.6. División y mezcla de datos de entrenamiento y prueba.

Llegados a este punto y luego de verificar que los datos efectivamente corresponden a datos de “0” y “1”, procedemos a dividir tomando el 80% de los mismos para el entrenamiento y el restante 20% para la validación o prueba.

```
# Usaremos un 80% para entrenamiento y un 20% para prueba
split_ratio = 0.8
split_index = int(X.shape[0] * split_ratio)
```

Seguido de esto se realiza la mezcla de datos para evitar algún posible cesgo

```
# Mezclar los datos antes de dividirlos
indices = np.arange(X.shape[0])
np.random.shuffle(indices)

X_shuffled = X[indices]
y_shuffled = y[indices]

# Dividir los datos
X_train, X_test = X_shuffled[:split_index], X_shuffled[split_index:]
y_train, y_test = y_shuffled[:split_index], y_shuffled[split_index:]
```

2.1.7. Normalizando los datos y creando el modelo.

A continuación, se normalizan los valores de grises de las imágenes limitándolos a valores entre “0” y “1”.

```
#Normalizar imágenes (valores de 0-255 a 0-1)
X_train, X_test = X_train / 255.0, X_test / 255.0
```

De igual forma se define el modelo con 3 capas con 25,15 y 1 neurona(s) respectivamente, cada una de estas con función de activación sigmoide.

```
#Construir el modelo de red neuronal
model = keras.Sequential([
    keras.Input(shape=(400,)),
    keras.layers.Dense(25, activation='sigmoid'),  # Capa oculta con 25 neuronas
    keras.layers.Dense(15, activation='sigmoid'),   # Otra capa oculta
    keras.layers.Dense(1, activation='sigmoid') # Capa de salida con 1 clase
])
```

2.1.8. Compilando el modelo y entrenamiento.

En esta etapa se define el optimizador, así como las perdidas e igualmente se le hace un seguimiento al modelo visualizando la precisión de este durante el entrenamiento, para este caso se usan 30 epocas.

```
#Compilar el modelo con entropía cruzada categórica
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
#Entrenar el modelo
model.fit(X_train, y_train, epochs=30, validation_data=(X_test, y_test))
```

2.1.9. Evaluando el modelo y prediciendo.

Una vez finalizado el entrenamiento se hace una prueba para evaluar tanto la precisión del modelo, así como las pérdidas de este.

```
# Evaluar el modelo
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"\n Precisión en el conjunto de prueba: {test_acc:.4f}")
```

Finalmente se hace la visualización de la imagen y muestra los resultados de la predicción teniendo como referencial el valor real del dato y cuál fue la predicción hecha por la red.

```
#Hacer una predicción y mostrar una imagen con su predicción
predictions = model.predict(X_test)

index = np.random.randint(0, len(X_test))
image = X_test[index].reshape(20, 20)

# Obtener predicción (probabilidad entre 0 y 1)
probabilidad = predictions[index][0]  # predictions es un array de forma (n, 1)
prediccion = 1 if probabilidad > 0.5 else 0

# Etiqueta real (asumiendo que y_test son 0/1, no one-hot)
etiqueta_real = y_test[index]

# Mostrar
plt.imshow(image, cmap='gray')
plt.title(f"Predicción: {prediccion}, Real: {etiqueta_real}")
plt.axis('off')
plt.show()
```

2.2.Resultados

Una vez finalizado el entrenamiento se obtuvo un modelo cuya precisión es del 99,50 % y unas pérdidas de 0,6199, a continuación, se evidencian algunas pruebas realizadas una vez entrenado el modelo.

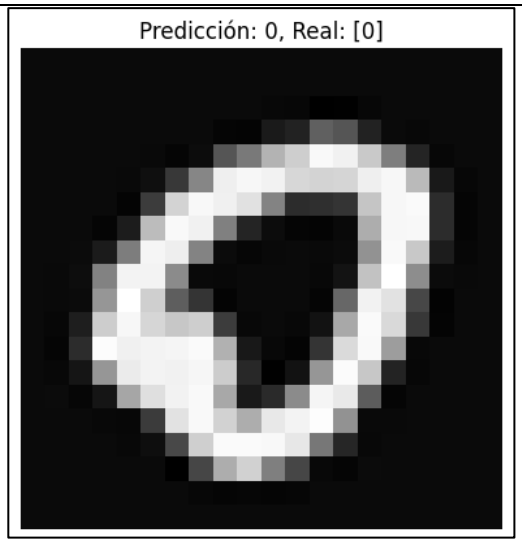

Ilustración 2. Primera prueba.

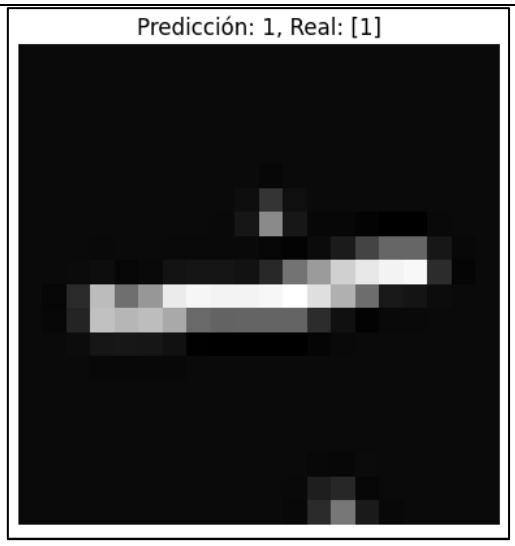

Ilustración 3. Segunda prueba.

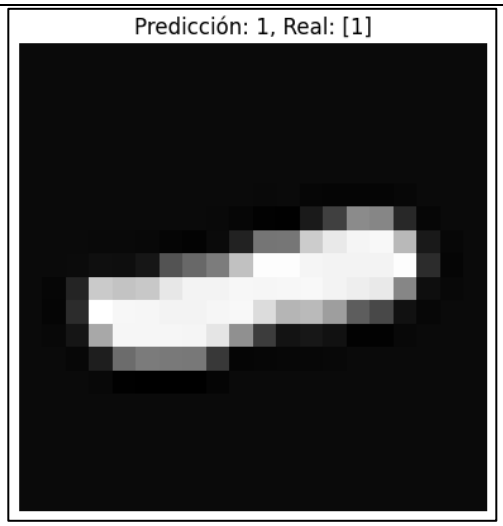

Ilustración 4. Tercera prueba.

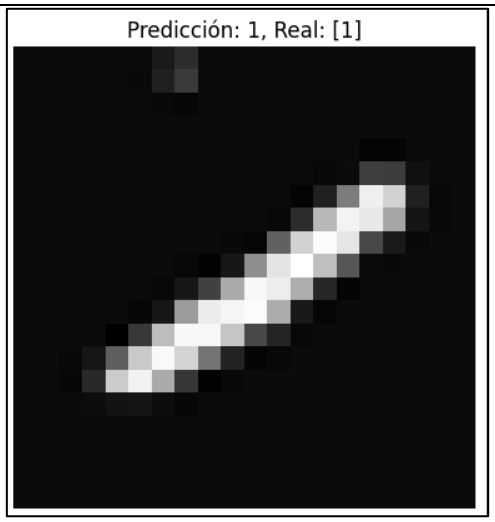

Ilustración 5. Cuarta prueba.

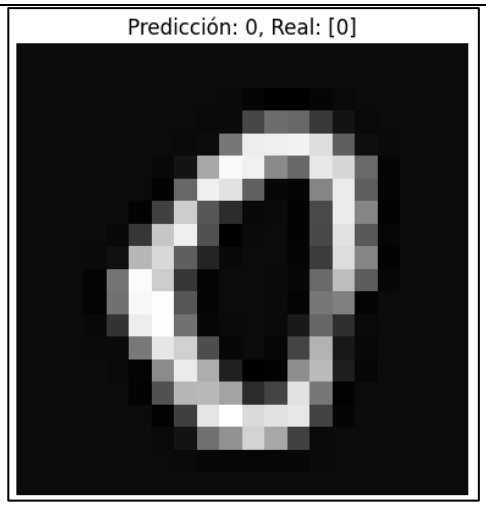

Ilustración 6. Quinta prueba.

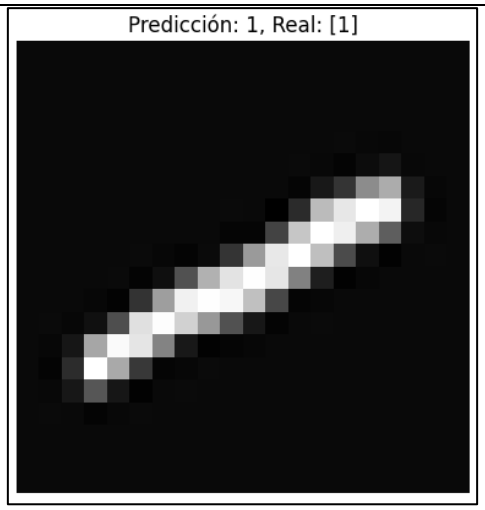

Ilustración 7. Sexta prueba.

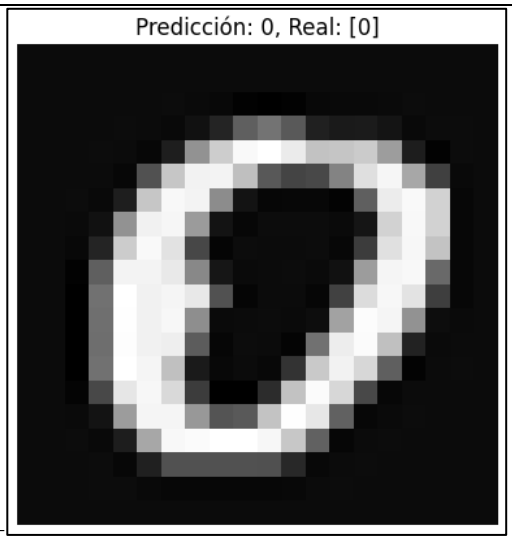

Ilustración 8. Séptima prueba.

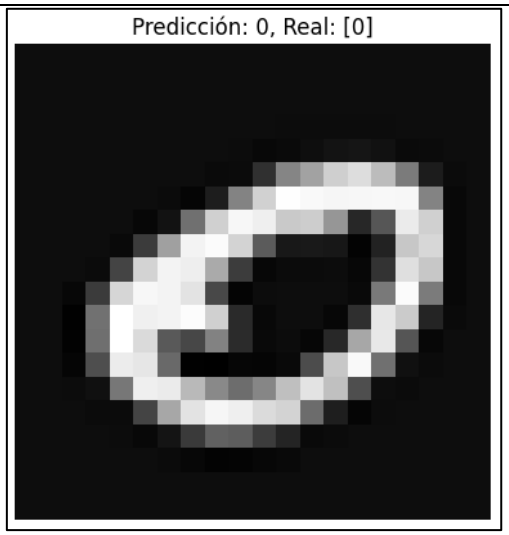

Ilustración 9. Octava prueba.

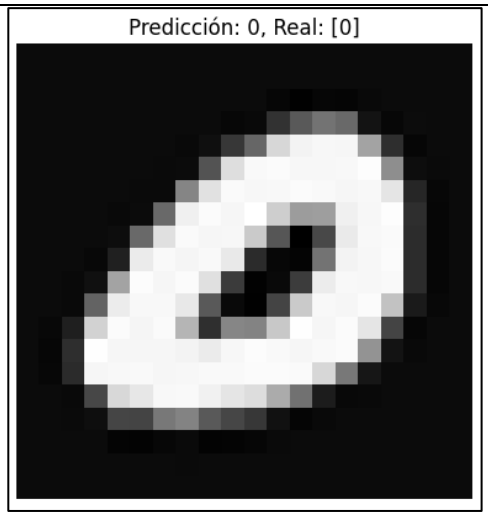

Ilustración 10. Novena prueba.

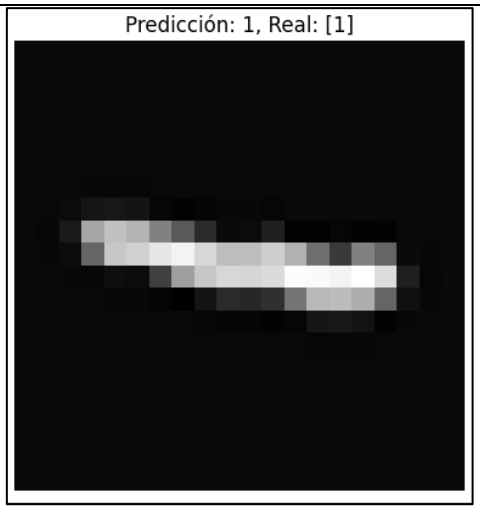

Ilustración 11. Decima prueba.

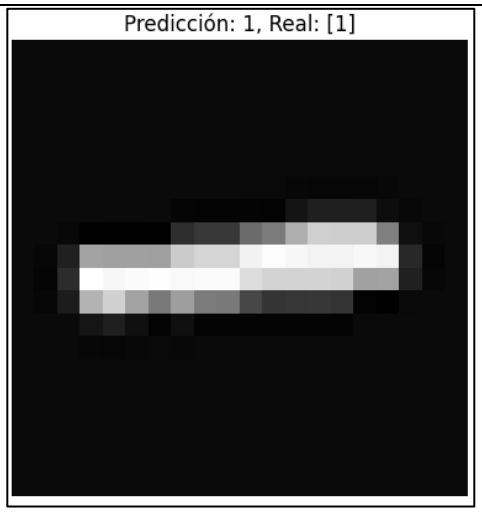

Ilustración 12. Décima primera prueba.

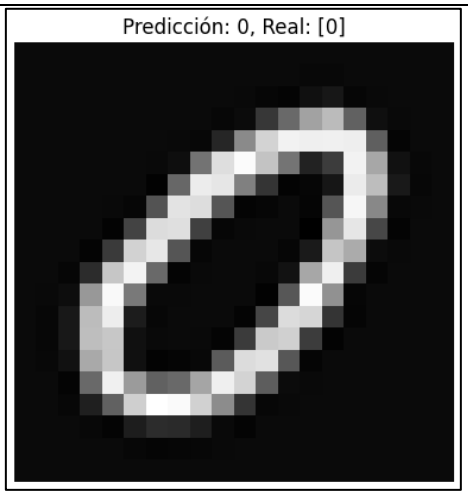

Ilustración 13. Décima segunda prueba.

3. Conclusiones

El modelo de red neuronal diseñado y entrenado en este proyecto fue capaz de clasificar de imágenes de 20x20 píxeles de forma binaria determinando si es un “1” o “0”. Luego de dividir y mezclar los datos en conjuntos de entrenamiento y prueba, y normalizar las imágenes, el modelo fue entrenado durante 30 épocas, alcanzando una precisión de 99,5% en el conjunto de prueba y unas pérdidas de 0,6991.

4. Referencias Bibliográficas

[1] Introducción a la Inteligencia Artificial - Brita Inteligencia Artificial. (s.f.). Brita Inteligencia Artificial. https://brita.mx/introduccion-a-la-inteligencia-artificial/

[2] Russell, S., & Norvig, P. (2016). Artificial intelligence: A modern approach (3rd ed.). Pearson Education.

[3] Abadi, M., Barham, P., Chen, J., et al. (2016). TensorFlow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 265-283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[4] Chollet, F. (2015). Keras. GitHub repository. https://github.com/fchollet/keras

[5] Van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. Computing in Science & Engineering, 13(2), 22-30. https://doi.org/10.1109/MCSE.2011.37

[6] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95. https://doi.org/10.1109/MCSE.2007.55

5. Anexos

5.1.Código Utilizado

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

# Cargando el conjunto de datos
X_in = np.load('/content/drive/MyDrive/Colab Notebooks/Inteligencia/Teoria/X.npy')
y_in = np.load('/content/drive/MyDrive/Colab Notebooks/Inteligencia/Teoria/y.npy')

X=X_in[0:1000]
y=y_in[0:1000]

print(X.shape)
print(y.shape)

#Función para visualizar imágenes
def visualizar_imagenes(arreglo, j,k):
    #asegurese de que el número de imágenes no exceda el tamaño del arreglo
    num_imagenes=min((k-j), arreglo.shape[0])
    #crear una figura para mostrar las imágenes
    fig,axes=plt.subplots(1,(k-j), figsize=(15,3))
    #recorrer y mostrar las imágenes
    for i in range(j,k):
        #obtener la imagen i-ésima y redimensionarla a 20x20
        imagen=arreglo[i].reshape(20,20)
        #mostrar la imagen
        axes[i-j].imshow(imagen, cmap='grey')#usar 'grey' para imágenes
    plt.show()

visualizar_imagenes(X,990,1000)

# Usaremos un 80% para entrenamiento y un 20% para prueba
split_ratio = 0.8
split_index = int(X.shape[0] * split_ratio)

# Mezclar los datos antes de dividirlos
indices = np.arange(X.shape[0])
np.random.shuffle(indices)

X_shuffled = X[indices]
y_shuffled = y[indices]

# Dividir los datos
X_train, X_test = X_shuffled[:split_index], X_shuffled[split_index:]
```

```
y_train, y_test = y_shuffled[:split_index], y_shuffled[split_index:]

#Normalizar imágenes (valores de 0-255 a 0-1)
X_train, X_test = X_train / 255.0, X_test / 255.0

#Construir el modelo de red neuronal
model = keras.Sequential([
   keras.Input(shape=(400,)),
   keras.layers.Dense(25, activation='sigmoid'),  # Capa oculta con 25 neuronas
   keras.layers.Dense(15, activation='sigmoid'),   # Otra capa oculta
   keras.layers.Dense(1, activation='sigmoid') # Capa de salida con 1 clase
])

#Compilar el modelo con entropía cruzada categórica
model.compile(optimizer='adam',
            loss='binary_crossentropy',
            metrics=['accuracy'])

#Entrenar el modelo
model.fit(X_train, y_train, epochs=30, validation_data=(X_test, y_test))

# Evaluar el modelo
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"\n Precisión en el conjunto de prueba: {test_acc:.4f}")

#Hacer una predicción y mostrar una imagen con su predicción
predictions = model.predict(X_test)

index = np.random.randint(0, len(X_test))
image = X_test[index].reshape(20, 20)

# Obtener predicción (probabilidad entre 0 y 1)
probabilidad = predictions[index][0]  # predictions es un array de forma (n, 1)
prediccion = 1 if probabilidad > 0.5 else 0

# Etiqueta real (asumiendo que y_test son 0/1, no one-hot)
etiqueta_real = y_test[index]

# Mostrar
plt.imshow(image, cmap='gray')
plt.title(f"Predicción: {prediccion}, Real: {etiqueta_real}")
plt.axis('off')
plt.show()
```