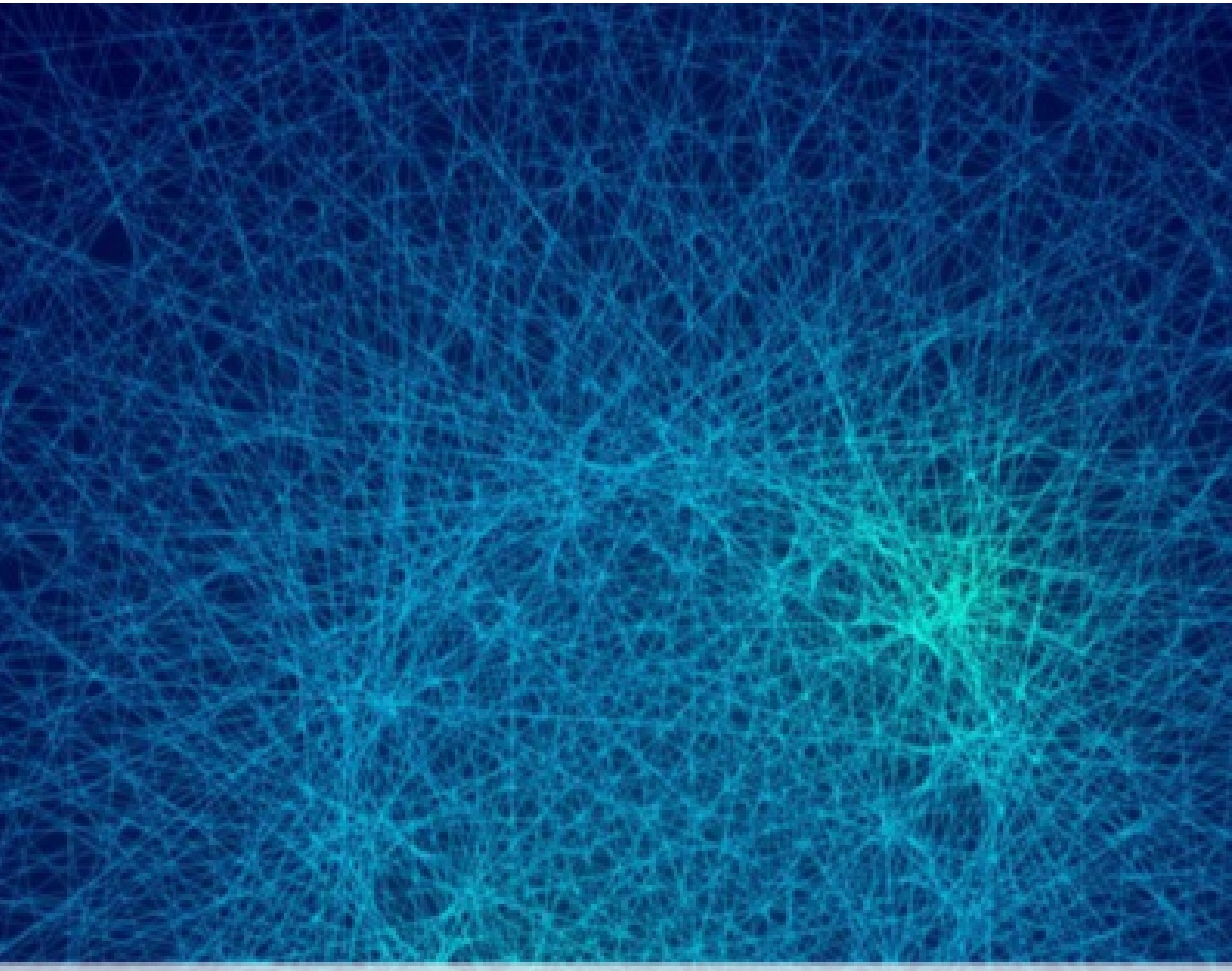
TIC



Redes neuronales completamente conectadas aplicadas en MNIST

TALENTO TECH

UTP Universidad Tecnológica de Pereira

faceIT

TIC

Redes neuronales completamente conectadas aplicadas en MNIST:

Reconocimiento de dígitos manuscritos

Para esta actividad vamos a trabajar con el conjunto de datos MNIST. La documentación oficial se encuentra en: http://yann.lecun.com/exdb/mnist/

El conjunto de datos MNIST (Modified National Institute of Standards and Technology) es un conjunto de imágenes que contiene dígitos escritos a mano, ampliamente utilizado como punto de referencia en el campo de la visión por computadora y el aprendizaje automático. Fue creado a partir de dos conjuntos de datos más grandes recopilados por el Instituto Nacional de Estándares y Tecnología de EE. UU. El conjunto de datos MNIST consiste en un total de 70,000 imágenes en escala de grises de 28x28 píxeles. Estas imágenes están divididas en dos conjuntos:

Uno de entrenamiento, que contiene 60,000 imágenes

Otro de prueba, que contiene 10,000 imágenes

Cada imagen representa un único dígito del 0 al 9.

El objetivo principal de MNIST es proporcionar un conjunto de datos estándar para la evaluación de algoritmos de reconocimiento de patrones, especialmente en el contexto de reconocimiento óptico de caracteres (OCR). Los investigadores y los desarrolladores de algoritmos de aprendizaje automático utilizan MNIST como punto de partida para probar nuevas técnicas y algoritmos, debido a su simplicidad y facilidad de acceso. A pesar de que MNIST es un conjunto de datos relativamente pequeño y simple en comparación con algunos conjuntos de datos más modernos, sigue siendo ampliamente utilizado como una prueba inicial para nuevos algoritmos y como una referencia para comparar el rendimiento de diferentes modelos.

Pasos a ejecutar:

1

Carga del conjunto de datos MNIST

Utiliza la función **load_data()** del módulo **mnist** de Keras para cargar el conjunto de datos MNIST, que contiene imágenes de dígitos escritos a mano y sus etiquetas asociadas.

```
import keras
# Cargar el conjunto de datos MNIST
mnist = keras.datasets.mnist
# Cargue la división de entrenamiento y prueba del conjunto
de datos MNIST
(training_images, training_labels), (test_images,
test_labels) = mnist.load_data()
```

2

Visualización de una muestra

- Selecciona un índice específico (**index**) dentro del conjunto de entrenamiento para visualizar una muestra de imagen y su etiqueta correspondiente. Imprime la etiqueta y muestra la imagen utilizando **plt.imshow()** de Matplotlib.

```
import numpy as np
import matplotlib.pyplot as plt
# Puedes poner aquí entre 0 y 59999
index = 1
# Imprime la etiqueta y la imagen.
np.set_printoptions(linewidth=320)
print(f'Label: {training_labels[index]}')
print(f'Image:\n {training_images[index]}')
# Visualiza la imagen
plt.imshow(training_images[index])
```

3

Normalización de los datos

- Normaliza los valores de píxeles de las imágenes dividiendo por 255.0, lo que asegura que los valores estén en el rango [0, 1].

```
# Normalizar los valores de píxeles del tren y probar las
imágenes.
training_images  = training_images / 255.0
test_images = test_images / 255.0
```

4 Construcción del modelo

- Define un modelo secuencial utilizando **keras.models.Sequential()**. Este modelo consta de una capa de aplanamiento (**Flatten**) para convertir la imagen 2D en un vector 1D, seguida de dos capas completamente conectadas (**Dense**) con funciones de activación **relu** y **softmax**.

```
# Construir el modelo de clasificación.
model =
keras.models.Sequential([keras.layers.Flatten(input_shape=(28
,28)),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dense(10, activation='softmax')])
```

5 Compilación del modelo

- Compila el modelo utilizando **model.compile()**, especificando el optimizador (**adam**), la función de pérdida (**sparse_categorical_crossentropy**) y las métricas (**accuracy**).

```
# Compilar el modelo
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

6 Entrenamiento del modelo

- Entrena el modelo con los datos de entrenamiento utilizando **model.fit()**, especificando el número de épocas.

```
# Entrenar el modelo
history = model.fit(training_images, training_labels,
epochs=10)
```

7

Graficar el historial de entrenamiento

- Utiliza pandas para convertir el historial de entrenamiento del modelo en un DataFrame y traza las curvas de pérdida y métricas durante el entrenamiento y la validación.

```
# Graficar el historial de entrenamiento:
pd.DataFrame(history.history).plot(grid=True)
```

8

Evaluación del modelo

- Evalúa el modelo en el conjunto de entrenamiento y prueba utilizando **model.evaluate()**, lo que proporciona la pérdida y la precisión del modelo en los datos de prueba.

```
# Evaluar el modelo en el conjunto de entrenamiento
loss, accuracy = model.evaluate(training_images,
training_labels)
print("Pérdida en el conjunto de entrenamiento:", loss)
print("Precisión en el conjunto de entrenamiento:", accuracy)

# Evaluar el modelo con datos no vistos
loss, accuracy = model.evaluate(test_images, test_labels)
print("Pérdida en el conjunto de prueba:", loss)
print("Precisión en el conjunto de prueba:", accuracy)
```

9

Predicción de una muestra

- Selecciona un índice específico dentro del conjunto de prueba para realizar una predicción utilizando **model.predict()**. Imprime la etiqueta real y la clasificación predicha.

```
# Evaluar el modelo con datos no vistos
loss, accuracy = model.evaluate(test_images, test_labels)
print("Pérdida en el conjunto de prueba:", loss)
print("Precisión en el conjunto de prueba:", accuracy)
```

TIC

Preguntas de comprensión

1.¿Qué conjunto de datos se utiliza en este código y qué problema de aprendizaje automático se aborda?

2.¿Por qué es importante normalizar los valores de píxeles de las imágenes antes de entrenar el modelo?

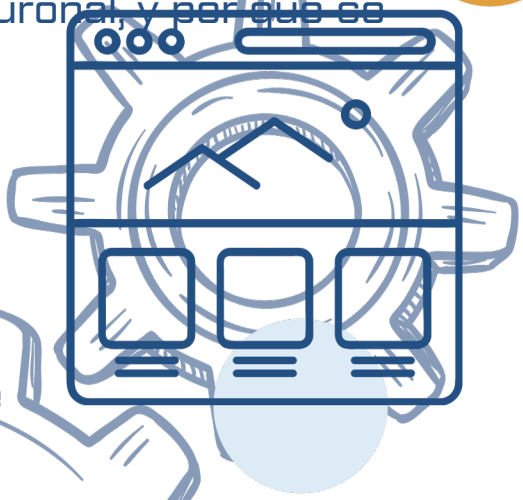
3.¿Qué arquitectura de red neuronal se utiliza en este código y cuántas capas tiene?

4.¿Cuál es la función de activación utilizada en la capa oculta y en la capa de salida de la red neuronal, y por qué se eligen esas funciones?

5.¿Qué función de pérdida se utiliza para compilar el modelo y qué métricas se utilizan para evaluar su rendimiento?

6.¿Cuántas épocas se utilizan para entrenar el modelo y por qué se elige ese número?

7.¿Qué significa la función model.summary() y qué información proporciona?



TIC

Ejercicios de exploración

1.Ejecuta la predicción de una muestra y observa que el resultado es una lista de números.
2.¿Por qué crees que es así y qué representan esos números?
3.Experimenta con diferentes valores para el número de neuronas en la
4.Cambia la función de activación de la capa oculta a "sigmoid" o "tanh" y observa cómo afecta el rendimiento del modelo.
5.¿Qué sucedería si eliminas la capa Flatten()?
6.Reflexiona sobre cómo afectaría esto al procesamiento de las imágenes y por qué.
7.Prueba diferentes optimizadores, como "sgd" o "rmsprop", y observa cómo afectan el rendimiento del modelo.
8.Aumenta el número de épocas de entrenamiento y observa cómo afecta el rendimiento del modelo.
9.Modifica la arquitectura de la red neuronal agregando capas adicionales o cambiando el número de neuronas en cada capa, y observa cómo afecta el rendimiento del modelo.
10.Elimina la normalización de los valores de píxeles y observa cómo afecta el rendimiento del modelo.
11.Considera las capas finales (de salida). ¿Por qué hay 10 de ellas? ¿Qué pasaría si tuvieras una cantidad diferente a 10?
12.Intenta entrenar la red con un número diferente de capas finales y reflexiona sobre cómo afectaría esto a la capacidad de clasificación del modelo.
13.Prueba el modelo con imágenes de prueba y observa cómo se comporta en datos no vistos.

Estas preguntas y ejercicios te ayudarán a profundizar tu comprensión del código y a explorar diferentes aspectos del entrenamiento de redes neuronales en el reconocimiento de dígitos escritos a mano usando el conjunto de datos MNIST.