Clasificación de imágenes.

Autor: German Preciat; Universidad de Guadalajara

Carrera: Ing. Biomédica

Materia: Analisis de datos clínicos

Contacto: german.preciat@academicos.udg.mx

La meta de este ejercicio es construir y entrenar un modelo de aprendizaje profundo (capas ocultas) para la clasificación de imágenes de prendas de vestir.

Pasos:

1. **Configuración del Entorno**: Configurar un entorno de desarrollo en Google Colab, aprovechando las bibliotecas fundamentales como TensorFlow.
2. **Preparación de Datos**: Explorar y preparar la base de datos de imágenes de ropa, comprendiendo la importancia de la estructura de los datos para el entrenamiento efectivo del modelo.
3. **Diseño del Modelo**: Construir un modelo de clasificación de imágenes utilizando TensorFlow, entendiendo los conceptos clave como capas, funciones de activación y optimizadores.
4. **Entrenamiento del Modelo**: Experimentar con el proceso de entrenamiento, ajustando parámetros y observando cómo el modelo aprende de los datos.
5. **Evaluación del Modelo**: Evaluar la precisión y el rendimiento del modelo en un conjunto de datos de prueba, comprendiendo la importancia de la validación para medir la eficacia del modelo.

Al completar estos pasos, se habrán desarrollado habilidades prácticas en análisis de imágenes, aprendizaje profundo y TensorFlow, sentando las bases para proyectos más avanzados y aplicaciones en el campo de la visión por computadora.

Paso 1: Importación de Bibliotecas y Conjunto de Datos**

- Importa TensorFlow y otras bibliotecas necesarias.
- Utiliza `tf.keras.datasets` para cargar el conjunto de datos de Fashion MNIST (un conjunto de datos de imágenes de ropa).

```
import tensorflow as tf
from tensorflow.keras import layers, models
```

```
import matplotlib.pyplot as plt

# Cargar datos
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.fashion
```

```
(train_images, train_labels), (test_images, test_labels) =
tf.keras.datasets.fashion_mnist.load_data()
```

utiliza la función load_data() de `TensorFlow` para cargar el conjunto de datos Fashion MNIST. Este conjunto de datos es comúnmente utilizado en tareas de aprendizaje automático, especialmente en la visión por computadora, como un reemplazo más simple del conjunto de datos MNIST.

La descomposición `(train_images, train_labels), (test_images, test_labels)` significa que el conjunto de entrenamiento se divide en dos partes: `train_images` contiene las imágenes que se utilizarán para entrenar el modelo, y `train_labels` contiene las etiquetas correspondientes para esas imágenes. Lo mismo se aplica al conjunto de prueba, donde test_images son las imágenes de prueba y `test_labels` son las etiquetas asociadas.

En resumen, se está cargando el conjunto de datos Fashion MNIST y dividiéndolo en imágenes de entrenamiento, etiquetas de entrenamiento, imágenes de prueba y etiquetas de prueba. Este conjunto de datos se utiliza comúnmente para tareas de clasificación de imágenes, donde el objetivo es entrenar un modelo para clasificar prendas de vestir en categorías específicas.

Paso 2: Exploración de Datos

- Visualiza algunas imágenes del conjunto de datos.
- Muestra estadísticas básicas sobre el conjunto de datos.

```
import numpy as np

# Visualizar imágenes aleatorias
plt.figure(figsize=(10, 10))
random_indices = np.random.randint(0, len(train_images), size=25)

for i, index in enumerate(random_indices):
    plt.subplot(5, 5, i+1)
    plt.imshow(train_images[index], cmap='gray')
    plt.axis('off')

plt.show()

# Mostrar estadísticas
print("Número de imágenes de entrenamiento:", len(train_images))
print("Número de imágenes de prueba:", len(test_images))
print("Forma de una imagen:", train_images[0].shape)
```

Paso 3: Preprocesamiento de Datos

- Normaliza las imágenes.

Las imágenes originales suelen tener píxeles con valores que van desde 0 hasta 255, donde 0 representa la intensidad mínima (negro) y 255 la intensidad máxima (blanco). Normalizar consiste en escalar estos valores para que estén en el rango de 0 a 1. Esto ayuda al modelo de la red neuronal a converger más rápido durante el entrenamiento y a mejorar la generalización del modelo. En este caso, se divide cada valor de píxel en las imágenes de entrenamiento y prueba por 255.0. Por lo tanto, después de esta operación, los valores de píxeles estarán en el rango [0, 1].

- Convierte las etiquetas a categorías.

Las etiquetas originales son simplemente números enteros que representan las clases (por ejemplo, 0 para camisetas, 1 para pantalones, etc.). Sin embargo, para entrenar un modelo de clasificación con TensorFlow, es común convertir estas etiquetas en formato categórico. Esto se hace mediante la técnica de "one-hot encoding", donde cada etiqueta se convierte en un vector binario que representa la clase. La función `to_categorical` de TensorFlow se utiliza para realizar esta conversión. En este caso, hay 10 clases en total , y por lo tanto, se especifica 10 como el número total de clases. Cada etiqueta se convierte en un vector de longitud 10, con un 1 en la posición correspondiente a la clase y 0 en todas las demás posiciones.

```
# Normalizar imágenes
train_images, test_images = train_images / 255.0, test_images / 255.0

# Convertir etiquetas a categorías
train_labels = tf.keras.utils.to_categorical(train_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)
```

Paso 4: Construcción del Modelo

- Crea un modelo secuencial simple.
- Añade capas convolucionales y de pooling.

```
# Crear modelo
model = models.Sequential()
```

Se crea un modelo secuencial, que es una pila lineal de capas. Los modelos secuenciales son apropiados para una pila simple de capas donde cada capa tiene exactamente un tensor de entrada y un tensor de salida.

```
# Añadir capas
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Paso 5: Compilación y Entrenamiento del Modelo

- Compila el modelo con una función de pérdida y un optimizador.
- Entrena el modelo con el conjunto de datos de entrenamiento.

```
# Compilar modelo
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

# Entrenar modelo
history = model.fit(train_images[..., tf.newaxis], train_labels, epochs=10, validati
```

Paso 6: Evaluación del Modelo y Visualización

- Evalúa el modelo con el conjunto de datos de prueba.
- Visualiza las métricas de entrenamiento y validación.

```
# Evaluar modelo
test_loss, test_acc = model.evaluate(test_images[..., tf.newaxis], test_labels, verb
print('\nExactitud en el conjunto de prueba:', test_acc)

# Visualizar métricas
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Época')
plt.ylabel('Exactitud')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

Double-click (or enter) to edit