FUNDACIÓN ESCUELA TECNOLÓGICA DE NEIVA
JESÚS OVIEDO PÉREZ

Taller Práctico: Clasificación de Imágenes en Google Colab

Objetivo del Taller

Crear un modelo de red neuronal que clasifique imágenes de dígitos escritos a mano utilizando el conjunto de datos MNIST.

Pasos Detallados

1. Configurar Google Colab

1. **Abrir Google Colab**:
   - Ve a Google Colab.
   - Si tienes una cuenta de Google, asegúrate de estar logueado.
   - Haz clic en "Nuevo cuaderno" (o "New Notebook") para crear un nuevo documento donde escribirás tu código.

2. Importar Bibliotecas Necesarias

En la primera celda del cuaderno, necesitas importar las bibliotecas que utilizarás. TensorFlow es una biblioteca popular para construir modelos de aprendizaje automático.

1. **Código**:

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
```

Explicación:

- `tensorflow`: biblioteca para el aprendizaje profundo.
- `layers` y `models`: módulos de Keras dentro de TensorFlow para construir redes neuronales.
- `matplotlib.pyplot`: biblioteca para la visualización de gráficos e imágenes.

3. Cargar el Conjunto de Datos

El conjunto de datos MNIST contiene imágenes de dígitos del 0 al 9. Cada imagen es de 28x28 píxeles.

1. **Código**:

```
# Cargar el conjunto de datos MNIST
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalizar los datos
x_train = x_train.reshape((60000, 28, 28, 1)).astype('float32') / 255
x_test = x_test.reshape((10000, 28, 28, 1)).astype('float32') / 255

```

Explicación:

- `mnist.load_data()`: carga el conjunto de datos y lo divide en un conjunto de entrenamiento y un conjunto de prueba.
- `reshape`: ajusta la forma de las imágenes para que sean compatibles con el modelo (28x28 píxeles y 1 canal de color, ya que son imágenes en escala de grises).
- `astype('float32')`: convierte los datos a un tipo de dato adecuado para el procesamiento.
- `dividir por 255`: normaliza los valores de píxeles a un rango entre 0 y 1.

4. Crear la Red Neuronal

Definimos la arquitectura de la red neuronal, que incluye capas convolucionales y de agrupamiento.

1. **Código**:

```
# Definir el modelo
model = models.Sequential()
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.Flatten())
model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))

    # Compilar el modelo
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
```

```
15.                     metrics=['accuracy'])
```

Explicación:

- `Sequential()`: crea un modelo secuencial donde las capas se apilan una sobre la otra.
- `Conv2D`: capa convolucional que extrae características de las imágenes. Aquí usamos 32 filtros de tamaño 3x3.
- `MaxPooling2D`: reduce la dimensionalidad y mantiene las características más relevantes.
- `Flatten()`: convierte la matriz de características en un vector.
- `Dense()`: capa totalmente conectada. La primera capa tiene 64 neuronas, y la segunda tiene 10 (una para cada dígito).
- `softmax`: función de activación que convierte las salidas en probabilidades.
- `compile()`: define el optimizador, la función de pérdida y las métricas a seguir durante el entrenamiento.

5. Entrenar el Modelo

Ahora entrenaremos el modelo con los datos de entrenamiento.

1. **Código**:

```
2. # Entrenar el modelo
3. model.fit(x_train, y_train, epochs=5)
```

Explicación:

- `fit()`: entrena el modelo usando los datos de entrada y las etiquetas. Aquí entrenamos durante 5 épocas, lo que significa que el modelo verá todo el conjunto de datos 5 veces.

6. Evaluar el Modelo

Después de entrenar, evaluaremos el rendimiento del modelo en datos que no ha visto.

1. **Código**:

```
2. # Evaluar el modelo
```

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Accuracy: {test_acc}')
```

Explicación:

- `evaluate()`: evalúa el modelo utilizando el conjunto de prueba y devuelve la pérdida y la precisión.
- Se imprime la precisión, que indica cuántas predicciones fueron correctas.

7. Hacer Predicciones

Finalmente, realizaremos algunas predicciones para ver cómo se comporta el modelo.

1. **Código**:

```
predictions = model.predict(x_test)

# Visualizar algunas predicciones
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title(f'Predicción: {predictions[i].argmax()}')
         plt.axis('off')
     plt.show()
```

Explicación:

- `predict()`: realiza predicciones sobre el conjunto de prueba.
- Utilizamos `matplotlib` para mostrar algunas imágenes y sus predicciones. `argmax()` devuelve el índice de la clase con la probabilidad más alta, que corresponde al dígito predicho.

Entregable

Cada grupo debe documentar su proceso en el cuaderno de Colab, incluyendo:

- El código utilizado.
- Capturas de pantalla de los resultados y de las visualizaciones.
- Un resumen de lo que aprendieron y posibles mejoras o experimentos futuros.

Al final, pueden descargar el cuaderno como PDF para presentar su trabajo.