



Universidad Autónoma de San Luis Potosí
Facultad de ingeniería
Inteligencia Artificial Aplicada
Practica 7
Redes neuronales convolucionales
Ana Sofía Medina Martínez



Fecha 31/10/2024

Objetivo

Que el alumno a diseñar, entrenar y evaluar modelos de redes neuronales convolucionales (CNN) para clasificación de imágenes.

Procedimiento

- 7.1.- Inicie Jupyter Notebooks y abra el archivo "cnn".
- 7.2.- Siga los pasos del Notebook para construir una red neuronal utilizando la arquitectura LeNet-5.
- 7.3.- Al terminar, guarde el modelo entrenado.
- 7.4.- Modifique el script "prueba_cnn" para cargar su modelo y ejecútelo.
- 7.5.- Utilizando una cámara capture la imagen de un dígito escrito a mano y compruebe los resultados del modelo. Utilice la interfaz para ajustar el preprocesamiento de la imagen y conseguir que el dígito sea correctamente apreciable.

Resultados

Introducción a las redes neuronales convolucionales

En este ejemplo se diseñará una red neuronal convolucional que pueda clasificar dígitos escritos a mano.

Cargar los datos

```
[1] #Cargar dataset del MNIST

from keras.datasets import mnist

# Cargar dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
11490434/11490434 ————— 1s 0us/step

El dataset del MNIST contiene dígitos del 0 al 9 escritos a mano

```
[2] import matplotlib.pyplot as plt
shown_digits = set()

plt.subplots(figsize=(10, 10))
for idx, image in enumerate(train_images):
    if train_labels[idx] not in shown_digits:
        plt.subplot(5, 5, len(shown_digits) + 1)
        plt.imshow(image, cmap=plt.cm.gray_r)
        plt.axis('off')
        plt.title('Digito: %i' % train_labels[idx])
        shown_digits.add(train_labels[idx])
plt.show()
```



Preprocesamiento

Cada dígito es un arreglo de Numpy de 8 bits sin signo de 28 píxeles de alto y 28 píxeles de anchos

```
[3] print(f"Dimensiones de la imagen: {train_images[0].shape}, tipo de dato: {train_images[0].dtype}")
```

Dimensiones de la imagen: (28, 28), tipo de dato: uint8

En este caso tenemos imágenes en escala de grises, Keras nos pide especificar el número de canales en estas imágenes, por lo que tenemos que convertir cada imagen de (28, 28) a (28, 28, 1)

Podemos hacer esto utilizando el método reshape de Numpy

```
x = train_images.reshape((60000, 28, 28, 1))
#TODO: Convertir a una imagen de 28x28x1
```

+ Code + Text

Otro paso importante de preprocesamiento es la normalización, al tener imágenes de 8 bits sin signo, el valor máximo que un píxel puede alcanzar es 255, vamos a normalizar convirtiendo los datos de entrada al tipo float32 y dividiéndolos entre 255 para tener píxeles en el rango de [0, 1]

```
[5] x = x.astype('float32') / 255
#TODO: Convertir a float32
x = x / 255
#TODO: Normalizar los valores entre 0 y 1
```

Las salidas en nuestro caso es el dígito al que corresponde la imagen de entrada, vamos a resolver el problema utilizando clasificación multiclase, por lo que aplicaremos one hot encoding para codificar las etiquetas

```
[6] from keras.utils import to_categorical

y = to_categorical(train_labels, num_classes=10)
```

```
!pip install tensorflow
import tensorflow as tf
from tensorflow import keras
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!0.5.1,!0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.12.1)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!4.21.1,!4.21.2,!4.21.3,!4.21.4,!4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.0)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.44.0)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (13.9.3)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.13.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.2.3)
Requirement already satisfied: certifi>2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (0.7.1)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow) (3.0.6)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow) (2.18.0)
Requirement already satisfied: mdurl=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0->tensorflow) (0.1.2)
```

```
[17] from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

#TODO: Crear el modelo
model = keras.Sequential([ Conv2D(6, (5, 5), activation='relu', input_shape=(28, 28, 1)),
                           MaxPooling2D((2, 2)), Conv2D(16, (5, 5), activation='relu'),
                           MaxPooling2D((2, 2)), Flatten(), Dense(120, activation='relu'),
                           Dense(84, activation='relu'), Dense(10, activation='softmax') ])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an 'input_shape'/'input_dim' argument to a layer's __init__ (activity_regularizer=activity_regularizer, **kwargs)
```

```
[12] #TODO: Compilar el modelo
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
#TODO: Entrenar el modelo
history = model.fit(x, y, batch_size=32, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
1500/1500 — 28s 18ms/step - accuracy: 0.5804 - loss: 1.2144 - val_accuracy: 0.9051 - val_loss: 0.3211
Epoch 2/10
1500/1500 — 26s 17ms/step - accuracy: 0.9050 - loss: 0.3066 - val_accuracy: 0.9341 - val_loss: 0.2112
Epoch 3/10
1500/1500 — 26s 17ms/step - accuracy: 0.9396 - loss: 0.1960 - val_accuracy: 0.9580 - val_loss: 0.1422
Epoch 4/10
1500/1500 — 26s 17ms/step - accuracy: 0.9583 - loss: 0.1384 - val_accuracy: 0.9672 - val_loss: 0.1072
Epoch 5/10
1500/1500 — 26s 17ms/step - accuracy: 0.9683 - loss: 0.1061 - val_accuracy: 0.9692 - val_loss: 0.1020
Epoch 6/10
1500/1500 — 41s 17ms/step - accuracy: 0.9731 - loss: 0.0851 - val_accuracy: 0.9745 - val_loss: 0.0850
Epoch 7/10
1500/1500 — 42s 18ms/step - accuracy: 0.9762 - loss: 0.0746 - val_accuracy: 0.9743 - val_loss: 0.0836
Epoch 8/10
1500/1500 — 26s 17ms/step - accuracy: 0.9782 - loss: 0.0687 - val_accuracy: 0.9771 - val_loss: 0.0731
Epoch 9/10
1500/1500 — 41s 17ms/step - accuracy: 0.9818 - loss: 0.0570 - val_accuracy: 0.9790 - val_loss: 0.0707
Epoch 10/10
1500/1500 — 40s 17ms/step - accuracy: 0.9841 - loss: 0.0528 - val_accuracy: 0.9797 - val_loss: 0.0657
```

```
#TODO: Evaluar el modelo con los datos de test
test_x = test_images.reshape((10000, 28, 28, 1))
```

```
[16] #TODO: Evaluar el modelo con los datos de test
test_x = test_images.reshape((10000, 28, 28, 1))
test_x = test_x.astype('float32')
test_x = test_x / 255
test_y = to_categorical(test_labels, num_classes=10)
model.evaluate(test_x, test_y)

313/313 ————— 3s 8ms/step - accuracy: 0.9778 - loss: 10.7084
[9.273012161254883, 0.9811999797821045]
```

```
[15] #TODO: Guardar el modelo
model.save('model.h5')
```

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This

Resultados con imagen en Colab

Modifiqué el código para que funcionara en un entorno de colab y detectara las imágenes

```
[1] import cv2
import numpy as np
from keras.models import load_model
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
```

```
[2] model = load_model('./mnist_cnn.h5')
```

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate the model.

```
[3] def preprocess_image(img, thresh_val=127, iter_val=1):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, thresh_img = cv2.threshold(gray, thresh_val, 255, cv2.THRESH_BINARY_INV)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
    thresh_img = cv2.morphologyEx(thresh_img, cv2.MORPH_OPEN, kernel, iterations=iter_val)
    thresh_img = cv2.resize(thresh_img, (28, 28))
    thresh_img = np.reshape(thresh_img, (1, 28, 28, 1))
    return thresh_img

def get_prediction(img, model):
    prediction = model.predict(img)
    return np.argmax(prediction)

def show_image(img, pred_val):
    final_img = img.reshape((28, 28))
    final_img = cv2.resize(final_img, (224, 224))
    plt.imshow(final_img, cmap='gray')
    plt.title(f"Predicción: {pred_val}")
    plt.axis('off')
    plt.show()
```

```
[15] from google.colab import files
uploaded = files.upload()
```

Choose Files. 3.webp
• 3.webp(image/webp) - 451252 bytes, last modified: 11/2/2024 - 100% done
Saving 3.webp to 3.webp

```
▶ image_path = next(iter(uploaded))  
img = cv2.imread(image_path)
```

```
[18] gray = preprocess_image(img)
```

```
[19] prediction = get_prediction(gray, model)
```

```
↗ 1/1 ————— 0s 18ms/step
```

```
[20] show_image(gray, prediction)
```



Predicción: 3



Otra predicción:

```
[21] Choose Files 8.webp
• 8.webp(image/webp) - 213750 bytes, last modified: 11/2/2024 - 100% done
Saving 8.webp to 8.webp

[22] image_path = next(iter(uploaded))
img = cv2.imread(image_path)


[23] gray = preprocess_image(img)

[24] prediction = get_prediction(gray, model)

1/1 ————— 0s 29ms/step

show_image(gray, prediction)
```

Predicción: 8



Comprensión

1. ¿Qué es una capa convolucional y cuál es su función?

Una capa convolucional en una red neuronal se encarga de aplicar filtros a la imagen de entrada para identificar características esenciales, como bordes y texturas. Su objetivo principal es reducir la dimensionalidad de la imagen manteniendo la información más importante.

2. Explique el propósito de las capas de agrupación (pooling)

Las capas de agrupación (pooling) disminuyen las dimensiones de las características extraídas, lo cual reduce el número de parámetros y optimiza el rendimiento de la red.

3. ¿Cómo se preprocesan típicamente las imágenes antes de alimentarlas a una CNN?

Se normalizan los valores de los píxeles y se redimensionan las imágenes a un tamaño estándar o también se utilizan técnicas de aumento de datos.

4. Mencione al menos dos arquitecturas de CNN populares

AlexNet y VGGNet.

5. ¿Qué ventajas tiene utilizar CNN con imágenes en comparación de las redes totalmente conectadas?

Las CNN son más adecuadas para el procesamiento de imágenes, ya que aprovechan la relación espacial entre los píxeles, lo que reduce significativamente el número de parámetros y mejora la habilidad de la red para identificar patrones visuales.

Conclusiones

Las redes neuronales convolucionales (CNN) representan un punto clave en el procesamiento de imágenes, destacando su habilidad para identificar y extraer patrones significativos mediante capas dedicadas, como las convolucionales y de agrupación. Gracias a estas capacidades, las CNN han impulsado importantes avances en áreas de la visión por computadora, como el reconocimiento de objetos y el análisis de imágenes especializadas, como las médicas.

Para realizar la práctica y cumplir con el objetivo, se optó por adaptar el código original al entorno de Google Colab, algunas modificaciones en el código fueron realizadas, permitiendo así su ejecución sin dependencias de hardware específico, como el uso directo de una cámara.