Universidad Autónoma de San Luis Potosí
Facultad de ingeniería
Inteligencia Artificial Aplicada
Práctica: 7
Nombre Práctica: Redes Neuronales Convolucionales
Nombre del Alumno: Manuel Ramírez Galván

Fecha: 19/03/2025

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 ejecútalo.
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.

Cargar los datos

```
#Cargar dataset del MNIST

from keras.datasets import mnist    Import "keras.datasets" could not be resolved

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

Imagen 1.- Código para Cargar Datos

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

```
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()
```

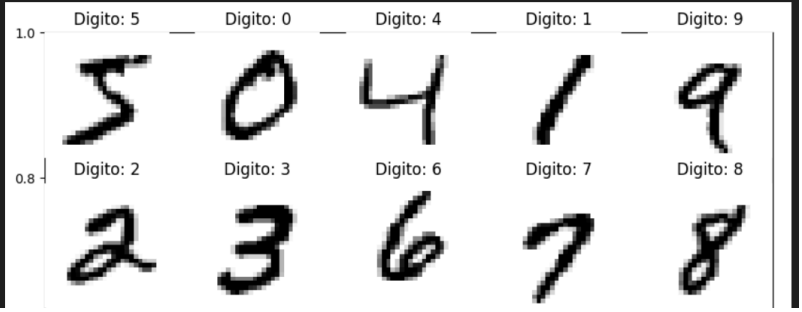

Imagen 2.- Datos del Dataset

Imagen 3.- Ver Tamaño del Arreglo

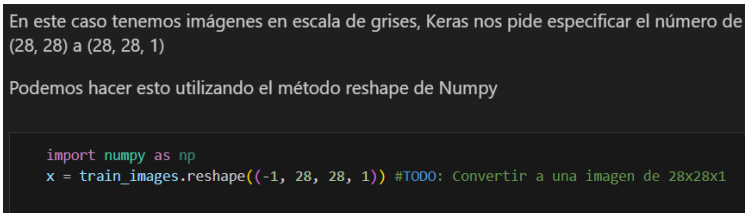

Imagen 4.- Convertir Imagen con Escala de Grises

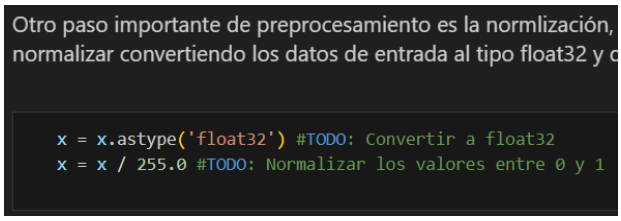

Imagen 5.- Convertir a Flotante y Normalizar

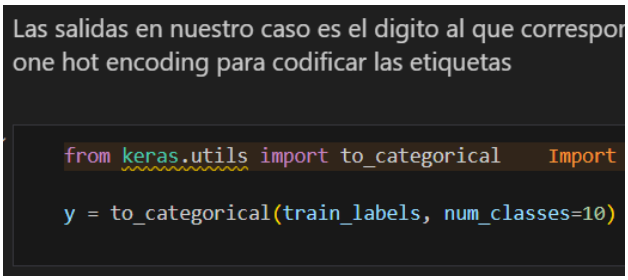

Imagen 6.- Agregar Variable Categórica

Imagen 7.- Definir Arquitectura de la Red

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

Imagen 8.- Compilar el Modelo

```python
#TODO: Entrenar el modelo
model.fit(x, y, epochs=10, batch_size=128)
```

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 10ms/step - accuracy: 0.7962 - loss: 0.7056
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9669 - loss: 0.1102
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9796 - loss: 0.0687
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9831 - loss: 0.0552
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9864 - loss: 0.0429
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9884 - loss: 0.0368
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9908 - loss: 0.0293
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.9911 - loss: 0.0277
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.9925 - loss: 0.0235
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - accuracy: 0.9929 - loss: 0.0217
```

Imagen 9.- Entrenar el modelo

```python
#TODO: Evaluar el modelo con los datos de test
test_loss, test_acc = model.evaluate(x, y)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9948 - loss: 0.0163
Test accuracy: 0.994533360004425
Test loss: 0.017184991389513016
```

Imagen 10.- Evaluar Datos

```python
#TODO: Guardar el modelo
model.save('mnist_cnn.h5')
```

Imagen 11.- Guardar Modelo

```python
import cv2
import numpy as np
import os
import sys
from keras.models import load_model
from keras.preprocessing import image

def preprocess_image(img):
    #Obtenemos el umbral y el numero de iteraciones
    thresh_val = cv2.getTrackbarPos('Umbral', 'frame')
    iter_val = cv2.getTrackbarPos('Iteraciones', 'frame')

    #Preprocesamiento de la imagen
    frame = img.copy() #Copiamos la imagen para no modificar la original
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #Convertimos a escala de grises

    ret, thresh_img = cv2.threshold(gray, thresh_val, 255, cv2.THRESH_BINARY_INV) #Aplicamos
umbralizacion para obtener una imagen binaria

    thresh_img = cv2.morphologyEx(thresh_img, cv2.MORPH_OPEN, kernel, iterations=iter_val) #Aplicamos
operaciones morfologicas para eliminar ruido

    thresh_img = cv2.resize(thresh_img, (28, 28), interpolation=cv2.INTER_LINEAR) #Redimensionamos a
28x28

    thresh_img = np.reshape(thresh_img, (1, 28, 28, 1)) #Agregamos una dimension para que sea compatible
con la entrada de la red neuronal

    return thresh_img

def get_prediction(img, model):
    #Obtenemos la prediccion de la red neuronal
    prediction = model.predict(img)

    #Obtenemos el valor maximo de la prediccion
    #En este caso, el indice corresponde al digito que se esta mostrando
    return np.argmax(prediction)

def show_image(img, pred_val):
    #Redimensionamos la imagen para que se vea mejor
    final_img = img.copy()
    final_img = final_img.reshape((28, 28)) #Redimensionamos a 28x28
    final_img = cv2.resize(final_img, (224, 224), interpolation=cv2.INTER_LINEAR)
    final_img = cv2.cvtColor(final_img, cv2.COLOR_GRAY2BGR)

    if pred_val:
        print("Prediccion: {}".format(pred_val))
        cv2.putText(final_img, str(pred_val), (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 1,
cv2.LINE_AA)

     #Mostramos la imagen
    cv2.imshow('frame', final_img)

#Cargamos el modelo
model = load_model('./mnist_cnn.h5')

#Captura de video
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("No se puede abrir la camara")
    exit()


cv2.namedWindow("frame", cv2.WINDOW_NORMAL)
cv2.createTrackbar('Umbral', 'frame', 0, 255, lambda x: x)
cv2.createTrackbar('Iteraciones', 'frame', 1, 7, lambda x: x)

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))

while cv2.waitKey(1) != ord('q'):
    ret, frame = cap.read()

    if not ret:
        print("No se puede recibir el frame. Finalizando ...")
        break

    gray = preprocess_image(frame) #Preprocesamos la imagen

    #Clasificacion
    prediction = get_prediction(gray, model)

    show_image(gray, prediction) #Mostramos la imagen

    cv2.waitKey(1)

cap.release()
cv2.destroyAllWindows()
```

Imagen 12.- Código para Probar el Modelo

Resultados

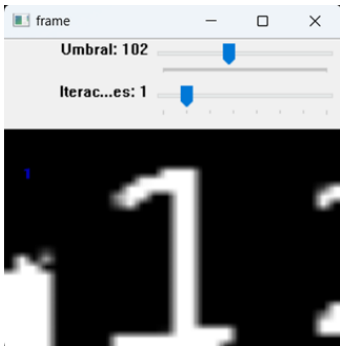

Imagen 13.- Identificar “1”

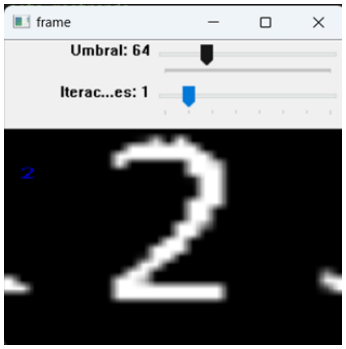

Imagen 14.- Identificar “2”

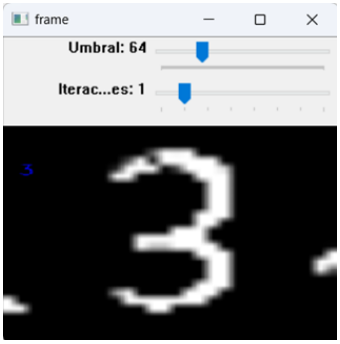

Imagen 15.- Identificar “3”

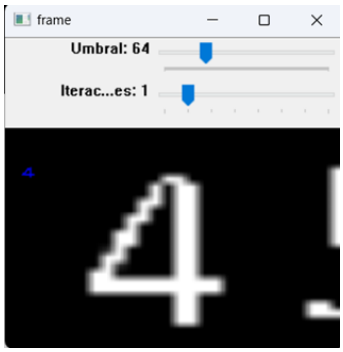

Imagen 16.- Identificar “4”

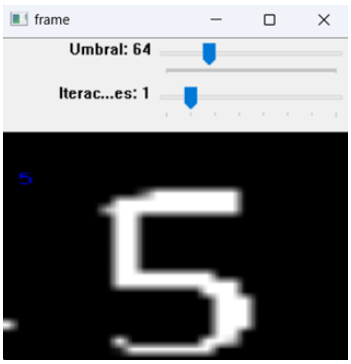

Imagen 17.- Identificar “5”

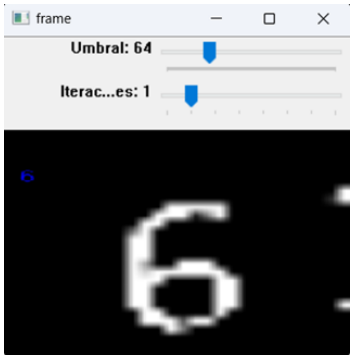

Imagen 18.- Identificar “6”

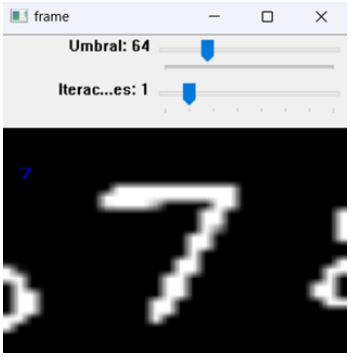

Imagen 19.- Identificar “7”

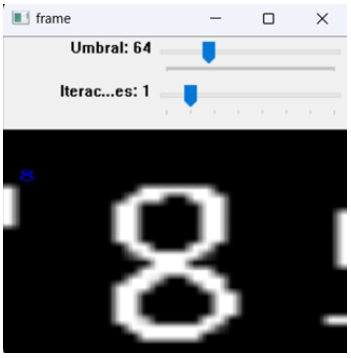

Imagen 20.- Identificar “8”

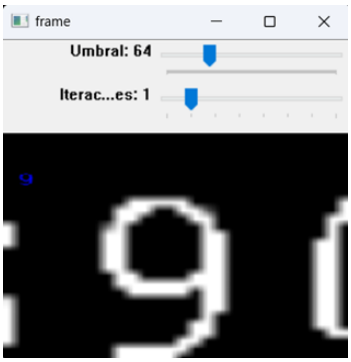

Imagen 21.- Identificar “9”

Imagen 22.- Identificar “0”

Comprensión

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

Su función principal es extraer características de las imágenes de entrada, como bordes, texturas, formas y patrones. Se inspira en el funcionamiento de la visión humana, donde distintas neuronas responden a diferentes patrones visuales.

Aplica una operación de convolución entre:

- Una imagen de entrada (una imagen en escala de grises o RGB)
- Un filtro o kernel (pequeña matriz, por ejemplo, de 3×3 o 5×5)

El filtro se desliza sobre la imagen, y en cada posición se realiza un producto punto entre los valores del filtro y la región de la imagen. Esto genera un mapa de activación o mapa de características.

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

Su función principal es reducir la dimensión espacial de los mapas de características (feature maps), manteniendo la información más importante.

- **Reduce el tamaño** de los mapas de activación (ancho y alto).
- **Disminuye la cantidad de parámetros** y el costo computacional.
- **Hace la red más robusta** a pequeñas variaciones, desplazamientos o ruido en la imagen.

3. **¿Cómo se pre-procesan típicamente las imágenes antes de alimentarlas a una CNN?**

- **Redimensionamiento (resizing),** adaptar todas las imágenes al mismo tamaño que espera la red.
- **Conversión a escala de grises,** reducir complejidad si el color no es importante.
- **Normalización,** asegurar que los valores estén en un rango estándar (generalmente entre 0 y 1, o -1 y 1).
- **Conversión a tensor,** cambiar el formato de la imagen a un tensor que pueda entender el framework.
- **Organización por lotes (batching),** agrupar las imágenes en mini-lotes (batches) para un entrenamiento más eficiente.

4. **Mencione al menos dos arquitecturas de CNN populares**

1. VGGNet (VGG16 / VGG19)

- Usa bloques de convolución 3×3 repetidos con max pooling 2×2.
- Arquitecturas más conocidas: VGG16 (16 capas con pesos) y VGG19.
- Muy utilizada en clasificación de imágenes, transfer learning y tareas de visión por computadora.

2. ResNet (Residual Network)

- Se compone de bloques residuales, donde la salida de una capa se suma a la entrada (salto de conexión).
- Versiones populares: ResNet18, ResNet34, ResNet50, ResNet101, ResNet152.

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

Preservan la estructura espacial de la imagen

Las CNN trabajan con ancho × alto × canales, por lo que reconocen patrones en el espacio, como bordes, formas y texturas.

En cambio, una red densa requiere aplanar (flatten) la imagen, perdiendo la información de posición y vecindad entre los píxeles.

Menos parámetros, menor complejidad

Las CNN usan filtros compartidos que se aplican sobre toda la imagen, lo que reduce drásticamente el número de parámetros.

Esto hace que las CNN sean:

- Más rápidas de entrenar
- Menos propensas al sobreajuste
- Más eficientes en memoria

En redes densas, cada píxel se conecta a todas las neuronas, lo que genera millones de pesos en imágenes grandes.

Aprenden características automáticamente

Las capas convolucionales detectan patrones locales como líneas, curvas, bordes, etc.

Las CNN pueden aprender:

- Bordes en capas iniciales
- Texturas en capas intermedias
- Objetos completos en capas profundas

Las redes densas no tienen esta especialización y requieren más datos para aprender los mismos patrones.

Invariante a traslaciones y deformaciones

Las CNN son más robustas a desplazamientos o pequeñas rotaciones de los objetos en la imagen.

Gracias a:

- El uso de pooling (agrupamiento)
- El tamaño reducido de los filtros
- La repetición de estructuras en diferentes partes de la imagen

Las redes densas no manejan bien estas variaciones sin una enorme cantidad de datos.

Mejor rendimiento en visión por computadora

Las CNN son el estándar en tareas como:

- Clasificación de imágenes
- Detección de objetos
- Reconocimiento facial
- Reconocimiento de gestos o lenguaje de señas

Las redes totalmente conectadas no son prácticas para imágenes grandes o tareas visuales complejas.

Conclusiones

Las redes neuronales convolucionales son redes especializadas en el procesamiento de imágenes, ya que conservan la estructura espacial, reducen la cantidad de parámetros y aprenden automáticamente características visuales importantes. Permite detectar patrones como bordes, formas y texturas de manera eficiente y precisa.

En el reconocimiento de números del 0 al 9, se logran altos niveles de precisión, superando el 98%, aunque haya llegado al 99% este modelo en ninguno de los casos y por más iteraciones que se hicieran no hacia el reconocimiento del número cero.