

**INTRODUCCIÓN A LA CIENCIA DE LA COMPUTACIÓN**

**LABORATORIO 1.02**

**GRUPO 3**

INFORME PROYECTO 2

**Profesor:**

* Brossard Nuñez, Ian Paul

**Integrantes del grupo:**

* López Rojas, Yaritza Milagros………………………………….(202410328) 20%
* Péres del Aguila Mayorga, Amira Emilyn…………………………..………..(202410312) 20%
* Mendoza Palacios, Gracia Luz……………………..………………...(202410390) 20%
* Rojas Gutierrez, Valentino Edgar…………………….………………(202410042) 20%
* Ildefonso Santos, Steve Andy……………………….…………….(202410402) 20%

Barranco, 2024

**Antecedentes:**

Este proyecto se centra en la implementación de un sistema de visión artificial que puede detectar un número escrito a mano y en papel, utilizando el dataset “digits” del paquete scikit-learn. Este dataset ofrece un conjunto de datos más pequeño y manejable para fines educativos y de demostración. Anteriormente se han desarrollado Teoría, Investigaciones y Modelos teóricos con respecto de la visión artificial. Tales como:

* + Teorías:
  + **Redes Neuronales Convolucionales (CNN):** Están diseñadas para automática y adaptativamente aprender características espaciales jerárquicas directamente de las imágenes en crudo. La teoría detrás de las CNN incluye la convolución, pooling (submuestreo) y la estructura jerárquica de capas que permiten detectar patrones complejos en las imágenes.
  + **Transformada de Fourier:** Se utiliza para transformar una imagen de su dominio espacial a su dominio de frecuencia. Esta teoría ayuda en la eliminación de ruido, compresión de imágenes y el análisis de texturas en las imágenes.
  + Investigaciones Recientes:
    - **Visión Transformers(ViT):** Los transformadores, originalmente desarrollados para procesamiento de lenguaje natural, se han adaptado recientemente para la visión por computadora. Los Vision Transformers han demostrado resultados competitivos con las CNN en varias tareas de clasificación de imágenes.

Dosovitskiy, A., et al. (2020). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." arXiv preprint arXiv:2010.11929.

* + - **GANs (Generative Adversarial Networks):** Las GANs se utilizan para generar imágenes realistas a partir de ruido, mejorando técnicas de superresolución y síntesis de imágenes.

Goodfellow, I., et al. (2014). "Generative Adversarial Nets." Advances in Neural Information Processing Systems (NeurIPS).

* + Modelos Teóricos:
    - **Modelo de Perceptrón Multicapa (MLP):** Un tipo de red neuronal artificial que consiste en múltiples capas de nodos, con cada capa completamente conectada a la siguiente. utilizado en las primeras etapas de reconocimiento de patrones y procesamiento de imágenes.

**Fundamento Teórico:**

El fundamento teórico de este proyecto se basa en la aplicación de técnicas de tratamiento de imágenes y análisis estadístico para el reconocimiento de patrones. Los pasos incluyen la manipulación de matrices de píxeles, el cálculo de promedios y el uso de la distancia euclídea para medir la similitud entre distintas imágenes de dígitos. La distancia euclídea es una medida de la "distancia recta" entre dos puntos en un espacio n-dimensional, en este caso, los valores de los píxeles de las imágenes de los dígitos. El proyecto también aprovecha técnicas de visualización para representar el promedio de dígitos y comparar un dígito recién introducido con las imágenes del conjunto de datos para determinar su identidad.

**Desarrollo:**

**Punto A:**

Para efectuar este requerimiento se usaron las librerías: sklearn, matplotlib y numpy. Haciendo énfasis en este último debido a su gran aporte en puntos posteriores que se explicarán más adelante.

Importar las librerías anteriormente mencionadas

from sklearn.datasets import load\_digits

import matplotlib.pyplot as plt

import numpy as np

----------------------------------------------------------

# Cargar el conjunto de datos de dígitos

digits = load\_digits()

----------------------------------------------------------

digit = int(input("Ingrese un numero del 0 al 9: "))

----------------------------------------------------------

#Filtrado de las matrices de igual ‘target’ al del numero pedido

Nos quedamos solo con los ‘target’ que son iguales al número pedido.

digit\_images = digits.images[digits.target == digit]

----------------------------------------------------------

Filtramos las matrices que coinciden con los ‘target’ filtrados.

Representación de ‘target’ e ‘images’:

‘target’ = array([ 0, 1, …])

[[0., 0., 0., ..., 5., 0., 0.],

[ 0., 0., 0., ..., 9., 0., 0.],

[ 0., 0., 3., ..., 6., 0., 0.],

...,

[ 0., 0., 1., ..., 6., 0., 0.],

[ 0., 0., 1., ..., 6., 0., 0.],

[ 0., 0., 0., ..., 10., 0., 0.]],

‘images’ = array([[[ 0., 0., 5., ..., 1., 0., 0.],

[ 0., 0., 13., ..., 15., 5., 0.],

[ 0., 3., 15., ..., 11., 8., 0.],

…...])

...,

[ 0., 4., 11., ..., 12., 7., 0.],

[ 0., 2., 14., ..., 12., 0., 0.],

[ 0., 0., 6., ..., 0., 0., 0.]],

--------------------------------------------------

Es un array de 3 dimensiones (# de ‘images’, altura, ancho), por ello el ‘axis=0’ para trabajar con el # de ‘images’.

average\_matrix = np.mean(digit\_images, axis=0)

Sirve para hallar el promedio de los ‘images’ filtrados (‘digit\_images’).

# Redondeo al entero más cercano

rounded\_average\_matrix = np.round(average\_matrix)

--------------------------------------------------

#este print no es necesario, solo esta para mejor comprensión del resultado

print(digit\_images)

print()

--------------------------------------------------

#este si imprime la matriz promediada

print(rounded\_average\_matrix)

--------------------------------------------------

#este print tampoco es necesario pero contribuye al comprendimiento

print(f"La cantidad de muestras para el numero {digit} son {len(digit\_images)}")

--------------------------------------------------

**Punto B:**

Usamos matplotlib para que, la matriz 8x8 generada anteriormente, se entienda de mejor manera en una escala de grises invertida.

La función plt.subplots() crea un “lienzo”, con un objeto de figura que representa a ventana (‘fig’) y un objeto de ejes que representa un gráfico (‘ax’)

--------------------------------------------------

fig, ax = plt.subplots()

ax.imshow muestra una imagen en un conjunto de ejes. En este caso rounded\_average\_matrix es la matriz que contiene los datos de la imagen que queremos mostrar y cmap='gray\_r'especifíca el mapa de colores a usar

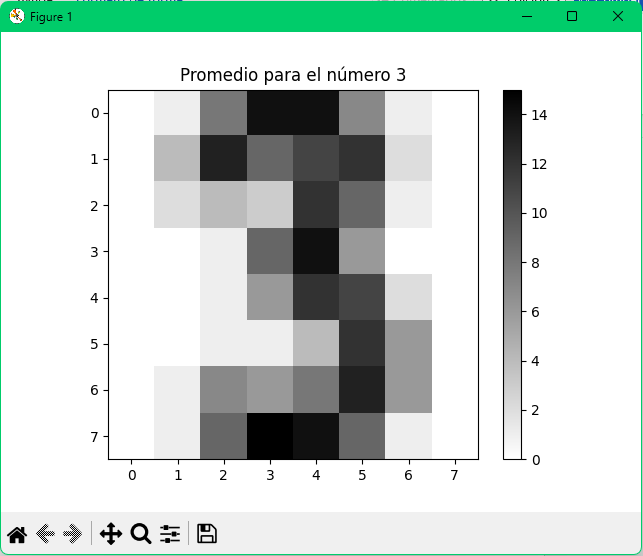
c = ax.imshow(rounded\_average\_matrix, cmap='gray\_r')

# Muestra la figura con la imagen en pantalla

plt.show()

--------------------------------------------------

**Ejemplo del Punto B con el número 3:**

Matriz dada por el Punto A: Retorno del Punto B:

>[[ 0. 1. 8. 14. 14. 7. 1. 0.]

[ 0. 4. 13. 9. 11. 12. 2. 0.]

[ 0. 2. 4. 3. 12. 9. 1. 0.]

[ 0. 0. 1. 9. 14. 6. 0. 0.]

[ 0. 0. 1. 6. 12. 11. 2. 0.]

[ 0. 0. 1. 1. 4. 12. 6. 0.]

[ 0. 1. 7. 6. 8. 13. 6. 0.]

[ 0. 1. 9. 15. 14. 9. 1. 0.]]

>La cantidad de muestras para el num 3 son 183

**Punto C:**

En este punto se usó OpenCV, específicamente la versión dos de este, que ayudara en la lectura y cambio de tamaño de matriz, además se importó matplotlib para un mejor entendimiento del proceso que realiza este punto.

import cv2

import matplotlib.pyplot as plt

mi\_imagen = cv2.imread("imagen\_buena.png", cv2.IMREAD\_GRAYSCALE)

imagen\_pequena = cv2.resize(mi\_imagen,(8,8))

*#invertimos los colores*

i = 0

while i <= 7:

    j = 0

    while j <= 7:

        imagen\_pequena[i][j] = 255 - imagen\_pequena[i][j]

        j += 1

    i += 1

*# aplicamos el martillo bonk ( 255 baja a 16 )*

i = 0

while i <= 7:

    j = 0

    while j <= 7:

        imagen\_pequena[i][j] = imagen\_pequena[i][j] / 255 \* 16

        j += 1

    i += 1

print(imagen\_pequena)

fig, ax = plt.subplots()

c = ax.imshow(imagen\_pequena, *cmap*='gray\_r')

plt.show()

**Punto D:**

Cálculo de la distancia euclidiana, y quedarse con los 3 dígitos más cercanos

def euclidean\_distance(matrix1, matrix2):

    return np.sqrt(np.sum((matrix1 - matrix2) \*\* 2))

def find\_closest\_index\_digits(new\_digit\_matrix, digits\_matrices, num\_closest=3):

    distances = []

    for i, digit\_matrix in enumerate(digits\_matrices):

        dist = euclidean\_distance(new\_digit\_matrix, digit\_matrix)

        distances.append((i, dist))

    # Función para obtener el segundo elemento de la tupla

    def get\_distance(tuple\_element):

        return tuple\_element[1]

    distances.sort(key=get\_distance)

    closest\_indices = [idx for idx, \_ in distances[:num\_closest]]

    return closest\_indices

**Punto E:**

Llamamos a los índices más cercanos con image\_new y digits.images

Se imprimen los 3 dígitos más cercanos.

Import pandas as pd

closest\_i = find\_closest\_index\_digits (image\_new, digits.images)

# Ya que encontramos los índices, con ayuda de estos

encontraremos los dígitos

closest\_d = [int(digits.target[x]) for x in closest\_i]

# Luego verificamos si hay coincidencias en los dígitos más cercanos

target\_c = {}

Inicializamos un contador de targets, si el dígito está presente se incrementa en 1. Caso contrario, se añade un 1

for digit in closest\_d:

if digit in target\_c:

target\_c[digit] += 1

else:

target\_c[digit] = 1

print(f"3 Dígitos más cercanos: {closest\_d}")

**Punto F :**

Intente clasificar a su nuevo dígito:

* Si 2 o 3 de estos targets corresponden a un mismo valor, o si los 3 corresponden a un mismo valor, imprime:“Soy la inteligencia artificial, y he detectado que el dígito ingresado corresponde al número X”

Igualamos max\_count al máximo valor del contador de targets

max\_count = max(target\_c.values())

---------------------------------------------------------------------------

Evalúa si hay al menos dos apariciones del target más cercano

if max\_count >= 2:

---------------------------------------------------------------------------

detec\_digit = [i for i, j in target\_c.items() if j == max\_count][0]

print(f"Soy la inteligencia artificial, y he detectado que el dígito ingresado corresponde al número {detec\_digit}")

* Si los 3 targets son diferentes, se realizará este proceso nuevamente pero ahora con 4 targets, y si aún no coincide seguirá añadiendo más targets sucesivamente hasta que se encuentre una coincidencia de al menos 2 dígitos. Para esto se convierte los datos a DataFrame y con .head encontraremos los índices de los dígitos de las imágenes. Luego repetiremos el procedimiento que se hizo antes del else.

else:  
 for num\_closest in range (4, len(digits.images)):

distances = []  
 for i, digit\_matrix in enumerate(digits. images):  
 dist = euclidean\_distance(image\_new, digit\_matrix)  
 distances.append((i, dist))

----------------------------------------------------------------------------

Se convierte la lista de tuplas en un DataFrame

distances\_df = pd.DataFrame(distances, columns=['ix', 'dist'])  
 sorted\_dist\_df = distances\_df.sort\_values(by='dist')  
 closest\_indices = sorted\_dist\_df.head(num\_closest)['ix']

Con ayuda de .head() se obtiene los índices de las imágenes de dígitos más cercanos con las distancias euclidianas

----------------------------------------------------------------------------  
 closest\_d = [int(digits.target[k]) for k in closest\_indices]  
 target\_counts = {}  
 for d in closest\_d:  
 if d in target\_counts:  
 target\_counts[d] += 1  
 else:  
 target\_counts[d] = 1  
  
 max\_count = max(target\_counts.values())  
 if max\_count >= 2:  
 detec\_digit = [k for k, v in target\_counts.items() if v == max\_count][0]  
 print(f"Soy la inteligencia artificial, y he detectado que el dígito ingresado corresponde al número {detec\_digit}")  
 break

Repetimos el procedimiento que hicimos al comienzo

**Punto G:**

Si las imágenes corresponden a i, entonces se almacenan como True

average\_matrices = []

# Itera sobre cada número del 0 al 9  
for i in range(10):

digit\_images = digits.images[digits.target == i]

Matriz que contiene todas las imágenes de los dígitos

digit\_images es una submatriz que contiene las imágenes de dígito que sea i.

Calcula el promedio de todas las imágenes de digit\_images

average\_matrix = np.mean(digit\_images,axis=0)

# Redondea los valores de la matriz para trabajar con valores enteros

rounded\_average\_matrix = np.round(average\_matrix)

# Añade rounded\_average\_matrix a average\_matrices  
 average\_matrices.append(rounded\_average\_matrix)

# ------------------------------------------------------------------------

distances = [euclidean\_distance(image\_new, avg\_matrix) for avg\_matrix in average\_matrices]

Se calcula la distancia euclidiana entre image\_new y avg\_matrix, la lista distances contendrá las distancias calculadas.

min\_distance = min(distances)

detec\_digit = distances.index(min\_distance)

detec\_digit es el indice del dígito cuyo promedio de imágenes está más cerca de image\_new

print(f"\nSoy la inteligencia artificial version 2, y he detectado que el

digito ingresado corresponde al numero {detec\_digit}")

**Punto H:**

Mediante tests de prueba comprobamos que la versión 1 es la más efectiva. La primera versión, que decide según los k vecinos más cercanos, tuvo un 75% de aciertos en comparación con la segunda, que obtuvo un 50% de aciertos.

La versión 1 utiliza un enfoque basado en el consenso de los dígitos mas cercanos. Primero, encuentra los índices de los dígitos más cercanos a la imagen nueva y luego determina el dígito más frecuente entre ellos.

Por otro lado, la versión 2 se basa en calcular la distancia euclidiana entre la imagen nueva y las matrices promedio de cada dígito. Aunque este método es más simple y computacionalmente eficiente, la prueba demostró que es menos preciso ya que las matrices promedio pueden no capturar adecuadamente la variedad de las imágenes dentro de cada clase de dígito.

La versión 1 se beneficia de considerar múltiples vecinos y ajustarse según sea necesario, lo que la hace más efectiva en la tarea de reconocimiento de dígitos. Para más información verificar la siguiente carpeta drive: <https://drive.google.com/drive/folders/1nyxIdQRXMkFdx0hOcY0-nrgqwamkHhUB?usp=sharing>

**Conclusiones:**

1. Gracias a aportes teóricos en el campo de visión artificial, para la inteligencia artificial 1 , ubicada en la parte f del código, se implementó una lógica que tiene relación con los dígitos más cercanos que se encontraron previamente.  A partir de la condicional se determina si se podrá procesar fácilmente el resultado  mediante la lista por comprensión. En caso contrario, se añaden dígitos cercanos, el procedimiento se repite hasta que logre encontrar alguno en común con los que ya se encontraban en la lista.

1. Pudimos identificar que para el correcto funcionamiento de ambas inteligencias artificiales es necesario que el tamaño de la imagen sea de un aproximado 700px de ancho y 660 de alto. Ya que si las medidas no son las indicadas habrá errores al momento en el cuál se procesa la imagen en la escala de grises, y  por consiguiente habrá una confusión al momento de identificar los dígitos.

1. Por medio de métodos tales como K-NN, uso de numpy y matplotlib hemos podido desarrollar lo asignado. K-NN usado desde el punto ‘d’ hasta el punto ‘g’. Proporcionan una base sólida para la comprensión del preprocesamiento de imágenes y la comparación de similitudes.