Universidad Católica del Uruguay

ÁLGEBRA APLICADA

Prof: Maglis Mujica

Autores: Juan Nocetti

Renzo Giacometti

Ionas Josponis

Año 2023

Índice

Introducción

En este proyecto, hemos desarrollado un programa en Python para poner en práctica todos los conceptos aprendidos en clase sobre matrices. Estos fueron aplicados utilizando imágenes y manipulándolas de diferentes formas.

Objetivos

Crear un programa en Python para aplicar conceptos y técnicas de manipulación de matrices, en el contexto del procesamiento de imágenes.

Marco Teórico

Matriz: Una matriz es un arreglo rectangular de mxn número reales (o complejos) ordenados m filas (líneas horizontales) y n columnas (líneas verticales)**.**

Tamaño de matriz: Es la cantidad de filas y columnas.

Matriz cuadrada: Son todas aquellas matrices con igual número de filas que de columnas.

Matriz identidad: Es toda matriz escalar cuyos elementos no nulos son todos iguales a 1.

Suma de matrices: Sea A=(aij) y B=(bij) dos matrices de mxn .La suma de A y B es la matriz A+B de mxn dada por:

$$A+B=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

Producto escalar por matriz: Dadas A, B ∈ Mmxn(R), A=((aij)) y B=((bij)) y k∈R, el producto del escalar k por la matriz A consiste en multiplicar cada elemento de la matriz A por k , o sea B=k.A⇔bij=k.aij,∀i=1,....,m,∀j=1,....,n.

Sea

$$k.\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} k.a_{11} & k.a_{12} & \cdots & k.a_{1n} \\ k.a_{21} & k.a_{22} & \cdots & k.a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ k.a_{m1} & k.a_{m2} & \cdots & k.a_{mn} \end{bmatrix}$$

Producto de dos matrices: Sea A=(aij) una matriz mxn, y sea B=(bij) una matriz nxp . Entonces el producto de A y B es una matriz mxp,

C=(cij), en donde cij= (renglón i de A) · (columna j de B)

Es decir, el elemento ij de AB es el producto punto del reglón i de A y la columna j de B. Si esto se extiende,se obtiene

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$$

Transpuesta de una matriz: Sea A una matriz de m filas y n columnas. At es un matriz de n filas y m columnas, que tiene por filas las columnas de A (y tiene por columna las filas de A) . At se denomina matriz transpuesta de A.

$$A \in M_{mxn}(\mathbb{R}), A = ((a_{ij})) \rightarrow A_t = ((b_|ji)) \in M_{nxm}(\mathbb{R})\ con\ b_{ji} = a_{ij}, i = 1, \ldots, m\ y\ j = 1, \ldots, n.$$

Matriz simétrica: Sea A una matriz cuadrada, es simétrica si y sólo si At =A

Si el número de columnas de A es igual al número de renglones de B, entonces se dice que A y B son compatibles bajo la multiplicación

Para poder multiplicar dos matrices,éstas deben ser conformables, es decir, el número de columnas de la primera matriz debe ser igual al número de filas de la segunda matriz

Herramientas

- IDE: Es un entorno de desarrollo para poder programar, en nuestra aplicación usamos PyCharm de JetBrains y Visual Studio Code.

- **Python:** Lenguaje de desarrollo.

-**GitHub:** Es una herramienta para desarrollar software colectivamente.

-**Numpy:** Librería de Python que nos facilita algunas operaciones entre matrices.

-**CV2**: Librería para trabajar con imágenes en Python.

-Skimage.io: Libreria de Python de procesamiento de imágenes que funciona con matrices numpy.

-**Trello**: Aplicación que se utiliza para administrar y organizar tareas de forma colaborativa.

Desarrollo

Para desarrollar este programa creamos dos clases principales, la clase utils y main.

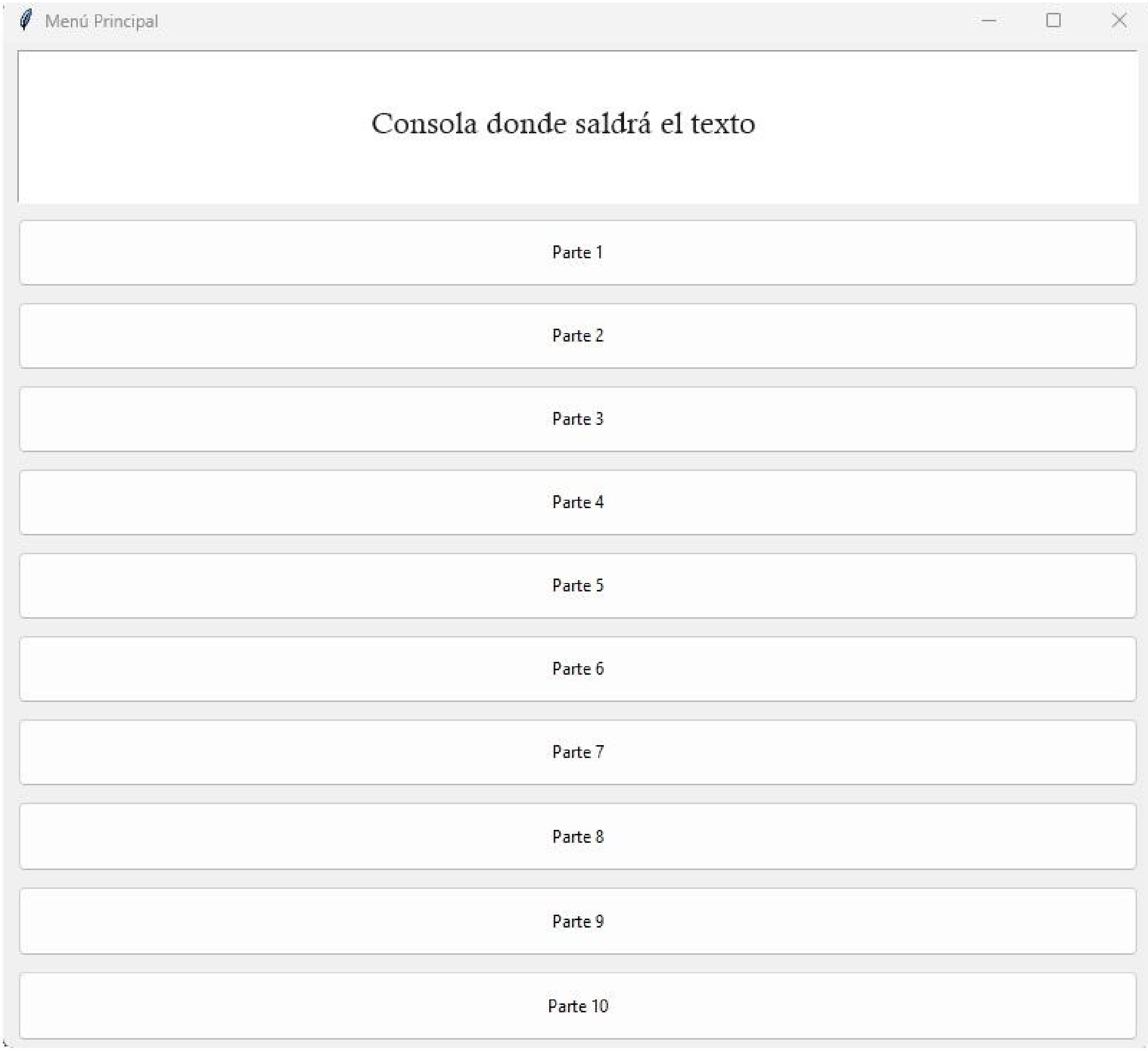
Dentro de la clase utils se encuentra toda la lógica para poder realizar las diferentes partes del proyecto. Mientras tanto, la clase main conecta la interfaz gráfica con la lógica del programa, en ella, podrás seleccionar la parte que desees ver.

Usabilidad de la aplicación, ¿Cómo usar el programa?

Sigue los siguientes pasos:

1. Ejecutar la aplicación.
2. Seleccionar la parte que deseas ver e ir cerrando las ventanas emergentes con los resultados para poder ver otra parte de la consigna.

Dependiendo de la parte seleccionada, se abrirán imágenes o se mostrará el texto resultante en la consola. En algunos casos se retornan tanto imágenes como texto.



imagen_a_matriz(): convierte la imagen a una matriz.

```python
def imagen_a_matriz(self):
    imagen = self.image
    # Asegurarse de que la imagen se haya cargado correctamente
    if imagen is not None:
        # Convertir de BGR a RGB
        imagen_rgb = cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)


        return imagen_rgb
    else:
        print("No se pudo cargar la imagen.")
```

matriz_a_imagen(): convierte una matriz a una imagen y asegura que todos los elementos de la matriz esten en el rango 0-255. Finalmente muestra la imagen.

```
def matriz_a_imagen(titulo, matriz):
    # Normaliza la matriz para que los valores estén en el rango 0-255
    matriz_normalizada = (matriz - np.min(matriz)) / (np.max(matriz) - np.min(matriz)) * 255

    # Utiliza np.clip() para asegurarte de que los valores estén en el rango 0-255
    matriz_normalizada = np.clip(matriz_normalizada, 0, 255)

    # Convierte los valores a enteros de 8 bits
    matriz_normalizada = matriz_normalizada.astype(np.uint8)

    # Muestra la imagen
    cv2.imshow(titulo, matriz_normalizada)
    cv2.moveWindow(titulo, 900, 100)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

imprimir_dos_matrices(): Muestra en pantalla dos imágenes dentro de una sola ventana, incluyendo un titulo.

```
def imprimir_dos_matrices(img1, img2, texto: str):
    # Mostrar las imágenes en escala de grises
    plt.subplot(121)
    plt.imshow(img1, cmap='gray')
    plt.title('Imagen 1 '+ texto)
    plt.axis('off')

    plt.subplot(122)
    plt.imshow(img2, cmap='gray')
    plt.title('Imagen 2 '+ texto)
    plt.axis('off')

    plt.tight_layout()
    plt.show()
```

calcular_transpuesta(): Muestra y calcula la transpuesta de la imagen dada.

```
def calcular_traspuesta(self):
    imagen = self.image
    imagen_rgb = cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)
    # Calcular la matriz traspuesta de la imagen
    imagen_traspuesta = np.transpose(imagen_rgb, (1, 0, 2))
    return imagen_traspuesta
```

convertir_a_escala_de_grises(): Convierte la imagen a escala de grises.

```
def convertir_a_escala_de_grises(self):
    imagen = self.image
    imagen_gris = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
    return imagen_gris
```

matriz_por_escalar(): Multiplica una matriz por un escalar alpha determinado.

```
def matriz_por_escalar(matriz, alpha):
    resultado = matriz * alpha
    resultado_limitado = np.clip(resultado, 0, 255)
    return resultado_limitado
```

crear_W() y voltear_imagen(): Estos dos métodos trabajan juntos para realizar la parte 9, el primer método crea la matriz W con la antidiagonal con números unos, mientras que el otro multiplica la matriz dada por W.

```
def crear_W(shape):
    if shape[0] != shape[1]: #Tiene que tener el mismo largo que ancho, sino tira excepcion personalizada
        raise ValueError("La matriz debe tener el mismo largo que ancho.")
    # Crear una matriz con unos en la antidiagonal
    I = np.eye(shape[0])
    w = np.fliplr(I)
    return w

2 usages  ionasjospo
def voltear_imagen(matriz1, matriz2):
    # Multiplicar la matriz original por la matriz con unos en la antidiagonal
    resultado = np.dot(matriz1, matriz2)
    resultado_limitado = np.clip(resultado, 0, 255)

    return resultado
```

son_iguales(): Verifica si dos matrices son iguales, devolviendo true si es así, false en caso contrario.

```
def son_iguales(matriz1, matriz2):
    return np.array_equal(matriz1, matriz2)
```

.

negativo_imagen(): Convierte la imagen a negativo.

```
def negativo_imagen(matriz, shape):
    if shape[0] != shape[1]: #Tiene que tener el mismo largo que ancho, sino tira excepcion personalizada
        raise ValueError("La matriz debe tener el mismo largo que ancho.")
    # Crear una matriz con 255
    matriz255 = np.full((shape[0], shape[1]), 255, dtype=np.uint8)
    return matriz255 - matriz
```

Respuestas a interrogantes planteadas:

¿Por qué ambas imágenes se recortan de forma cuadrada y con el mismo tamaño? (Parte 3)

Se pide en la letra del proyecto y además se realiza de esta manera para simplificar las futuras operaciones matriciales, asegurando que sean posibles todas las operaciones entre matrices y estas sean bien expresadas con las imágenes resultantes.

Comentarios sobre las matrices transpuestas de las imágenes (Parte 5):

Al ser realizada dicha operación, logramos visualizar un cambio significativo en la imágen, cambiando su orientación dando un giro de 90º de forma antihoraria. Incluso viendo las matrices resultantes corroboramos que la operación fue realizada correctamente.

Observaciones sobre multiplicaciones de matriz sobre distintos escalares (Parte 8):

El escalar mayor a 1 elegido fue el número 9, se eligió de forma al azar. Los cambios en la imágen que se obtuvo en la parte 6 (imágen en blanco y negro) fueron muy grandes, se puede observar un gran contraste que generá una cierta luminosidad.

Sin embargo, operando con el otro escalar elegido, el cual fue el 0,5, visualizamos una imágen con menos contraste , y colores más apagados.

¿La multiplicación entre matrices es conmutativa? (Parte 9)

Esta pregunta es resuelta a través de la prueba y realización de ambas operaciones (multiplicación cambiando el orden de los factores).

Este tipo de multiplicación no es conmutativa, lo pudimos observar viendo las matrices resultantes de ambas multiplicaciones ya que las mismas tienen valores distintos. Al igual que viendo las imágenes obtenidas, la primera (matriz x w) ésta volteada horizontalmente y la segunda (w x matriz) es volteada verticalmente. Por lo tanto, el orden de la multiplicación es importante y afecta la orientación de las imágenes.

Conclusiones

Se creó el programa para satisfacer todas las consignas proporcionadas. Se le agregó una interfaz gráfica para que quede mejor organizado y se pueda apreciar visualmente las diferentes partes.

Pasos a futuro

La única mejora a futuro sería hacer más bonita la interfaz gráfica.

Bibliografía

-Presentaciones de clase de Google Colabs.