Proyecto 2

Maximiliano Benítez 4.364.951-6, Matías Cabrera 5.260.108-6,

Diego López 4.922.299-6, Ignacio Valle 4.928.322-7

Universidad Católica del Uruguay

Álgebra Aplicada

Sebastián Decuadro

29 de septiembre de 2023

Índice

I. Introducción

En el marco del curso de Álgebra Aplicada, se ha planteado a los estudiantes el desafío de aplicar los conceptos y técnicas aprendidos en el curso para explorar y analizar imágenes mediante el uso de matrices y operaciones de álgebra lineal.

Las imágenes, como representaciones visuales, se componen de píxeles que forman la estructura de cada una. Este proyecto, se enfocará en comprender cómo una imagen puede ser representada como una matriz, donde el color de cada píxel se representa con un valor en el rango de 0 a 255. Esta representación matricial de las imágenes brinda la oportunidad de manipular y analizar visualmente la información de las imágenes utilizando herramientas matriciales.

El programa se centrará en explorar las capacidades de estas matrices, realizando una serie de transformaciones y operaciones matriciales para lograr diversos resultados. Comenzando por cargar y examinar dos imágenes de propia elección, y a lo largo del proceso, se aplicarán acciones como el recorte, la transposición, la conversión a escala de grises, entre otras.

Este informe detallará minuciosamente cada fase del proyecto, desde la importación inicial de las imágenes hasta el análisis de las transformaciones realizadas.

II. Marco teórico

Este marco teórico pretende proporcionarle al lector una comprensión de los conceptos utilizados para entender el desarrollo de la tarea realizada. Estos conceptos abarcaran definiciones sobre matrices, su composición, así como también sus operaciones fundamentales, tales como suma de matrices, multiplicación por un escalar y la multiplicación entre matrices, entre otras.

Una matriz es un arreglo p-dimensional de números reales (o complejos), organizados en filas y columnas. Comúnmente se representa con letras mayúsculas, como A, B o M. Una matriz con *m* filas y *n* columnas se denomina matriz *m x n*, donde "*m*" es el número de filas y "*n*" es el número de columnas. A modo de ejemplo, una matriz A de "*n*" filas y "*m*" columnas se podría representar de la siguiente manera:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2j} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \cdots & \vdots & \cdots & \vdots \\ a_{i1} & a_{i2} & \cdots & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & \vdots & & & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix} \leftarrow \text{fila (renglón) } i$$

$$\text{columna } j \uparrow$$

Los elementos de la matriz se representan con la misma letra de la matriz en minúscula, con un doble subíndice donde el primero indica la fila y el segundo la columna a la que pertenece. La dimensión de la matriz está definida por la cantidad de filas y la cantidad de columnas, y se representa *m x n*. Esta característica es esencial para definir la forma y el tamaño de la matriz, y desempeña un papel crucial en las operaciones y cálculos matriciales.

Existen algunas matrices que, por sus propiedades y usos específicos son importantes definir para entender el desarrollo de la tarea.

Se dice que una matriz es cuadrada cuando la cantidad de filas y de columnas son iguales, es decir, tiene dimensiones *n x n*. Por ejemplo, una matriz 3x3 es cuadrada.

$$\begin{bmatrix} 1 & 4 & 3 \\ 6 & 2 & 1 \\ 2 & 7 & 5 \end{bmatrix}$$

Una matriz nula se define como aquella que tiene todos sus elementos iguales a 0. Por otro lado, existen matrices particulares de interés. Una de ellas es la matriz diagonal, y se define como toda matriz cuadrada, en la que todos los elementos fuera de la diagonal principal son 0. Los elementos en la diagonal principal pueden ser cualquier valor. Por ejemplo, una matriz diagonal 3 x 3 podría ser:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

Una matriz escalar es un caso particular de matriz diagonal, en la cual todos los elementos de la diagonal son iguales. A modo de ejemplo se podría definir la siguiente matriz escalar:

$$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix}$$

Finalmente, se define la matriz identidad como una matriz escalar, pero su particularidad es que todos los elementos de la diagonal son iguales a 1. Se denota como "I_n" donde "*n*" es la dimensión de la matriz.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Otro concepto importante es la igualdad de matrices, que se cumple cuando dos matrices tienen las mismas dimensiones y todos sus elementos en las mismas posiciones son iguales. Su definición formal es:

$$A_{m\,x\,n} = B_{p\,x\,q} \Leftrightarrow \begin{cases} m = p,\ n = q \\ a_{ij} = b_{ij}\ \forall\, i,\ j,\ 1 \leq\ i \leq m, 1 \leq\ j \leq n \end{cases}$$

Al momento de trabajar con matrices es necesario tener claro las operaciones que se pueden realizar. Una de las operaciones fundamentales, es la suma de matrices. La suma de matrices es posible realizarla cuando se tienen dos matrices con la misma dimensión. Para sumar dos matrices simplemente se suman los elementos que se encuentran en la misma posición en ambas. Dado dos matrices A y B, donde la suma de estas matrices da como resultado la matriz C,

cada elemento "c_{ij}" se calcula como la suma de los elementos "a_{ij}" y "b_{ij}". Una formalización sería:

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

Para clarificar el concepto, se podría ejemplificar de la siguiente manera:

$$\begin{bmatrix} 1 & 4 & 9 \\ 0 & 3 & 1 \\ 5 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 3 & 1 & 5 \\ 0 & 3 & 7 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+3 & 4+1 & 9+5 \\ 0+0 & 3+3 & 1+7 \\ 5+1 & 1+2 & 1+1 \end{bmatrix} = \begin{bmatrix} 4 & 5 & 14 \\ 0 & 6 & 8 \\ 6 & 3 & 2 \end{bmatrix}$$

Otra de las operaciones que se pueden realizar con matrices, es el producto escalar por matriz. Esta operación consiste en multiplicar el escalar por cada elemento de la matriz. Siendo A una matriz y *k* un escalar, la multiplicación de k*A se realiza multiplicando cada elemento de A por el escalar *k*. Formalmente se expresa:

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

Un ejemplo de producto escalar por matriz podría ser:

$$2\begin{bmatrix} 1 & 5 & 0 \\ 1 & 2 & -1 \\ 0 & -1 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 10 & 0 \\ 2 & 4 & -2 \\ 0 & -2 & 6 \end{bmatrix}$$

Así como existe el producto entre escalares, también se pueden multiplicar las matrices que cumplan ciertas condiciones. Esta es una operación se conoce como producto de matrices y es un poco más compleja a las anteriores. Para que el producto de dos matrices sea posible, el número de columnas de la primera matriz debe ser igual al número de filas de la segunda matriz. Si A es una matriz de *m x n*, la matriz B debe ser de *n x p* y el resultado de hacer A por B es una matriz C de dimensión *m x p*. Cada elemento de la matriz C se calcula como la suma de los productos de la fila de A y la columna de B. Llevado a una definición más formal, cada elemento de C se obtiene de la siguiente forma:

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

Un ejemplo concreto del producto de matrices:

$$\begin{bmatrix} 1 & 0 & 2 \\ -1 & 3 & 1 \end{bmatrix}\begin{bmatrix} 3 & 1 \\ 2 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1(3)+0(2)+2(1) & 1(1)+0(1)+2(0) \\ -1(3)+3(2)+1(1) & -1(1)+3(1)+1(0) \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ 4 & 2 \end{bmatrix}$$

Otra de las operaciones que se pueden aplicar a las matrices, es la transpuesta de una matriz. Esta operación consiste en reorganizar los elementos de la matriz de manera que las filas se conviertan en columnas y las columnas en filas. Sea A una matriz de dimensiones *m x n*, la

matriz transpuesta, denotada A^t, tendrá dimensiones *n x m*. La transposición se realiza de la siguiente manera:

Si A = [a_{ij}], donde "a_{ij}" representa el elemento de la fila i y la columna j de la matriz A, la matriz transpuesta A^t se construye de modo que "a_{ij}" se convierte en "a_{ji}". En otras palabras, los elementos que estaban en la fila i y la columna j se ponen en la fila j y la columna i de la matriz transpuesta. Formalmente definida como:

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

Para poder visualizar lo dicho anteriormente, podríamos ejemplificar la matriz A y su matriz transpuesta A^t como:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad A^t = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Por último, para comprender el desarrollo a continuación, queda abordar el concepto de inversa de una matriz. Se dice que una matriz A de dimensión *n x n* es invertible si existe una matriz B, también de dimensión *n x n*, tal que el producto de A por B genera como resultado la matriz identidad de la misma dimensión. Formalmente:

$$A.B = Id_{nxn} \;\; y \;\; B.A = Id_{nxn}$$

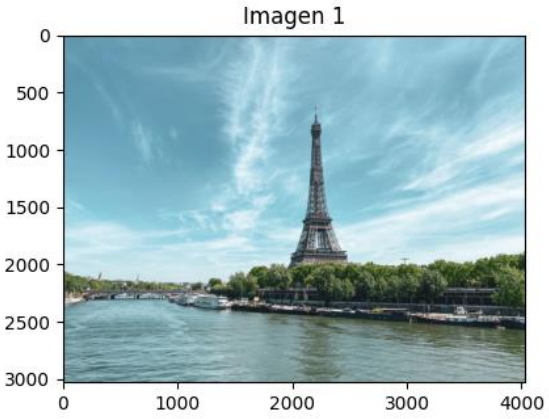
La matriz identidad cuando existe, es única para una matriz dada y comúnmente se denota como A^{-1}. Dicho esto, si A tiene una inversa, la notación A^{-1} se refiere a la matriz que, multiplicada por A, produce la matriz identidad. Un ejemplo de matriz inversa sería:

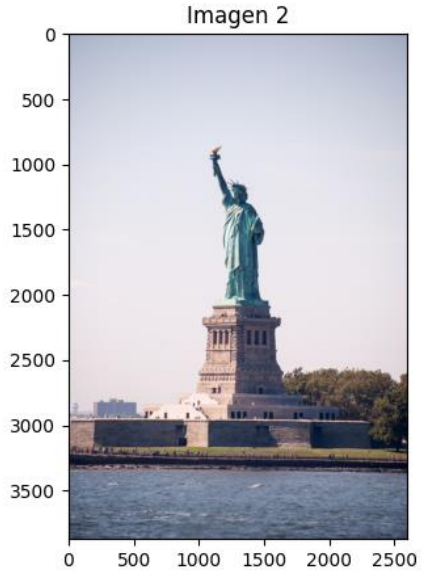
$$A = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} * B = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} \rightarrow Id = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

En el ejemplo anterior la matriz inversa de A es B, ya que el resultado de multiplicar estas 2 matrices genera la matriz identidad de la misma dimensión.

III. Desarrollo del programa

En esta sección, se detalla el desarrollo del programa implementado en Python, más concretamente en el servicio en la nube de Google Colaboratory, para la edición y transformación de imágenes. El programa fue diseñado con la finalidad de poder editar de diversas maneras determinadas imágenes aprovechando los fundamentos de matrices y operaciones algebraicas. Antes de profundizar en la configuración y funciones del programa, es importante presentar las imágenes originales utilizadas como punto de partida para las transformaciones:





En primera instancia se comienza defiendo distintos aspectos referentes a la configuración inicial del programa, además de las distintas funciones utilizadas a lo largo del desarrollo de la solución planteada. Entre los aspectos de configuración, se encuentran la importación de las diversas librerías a utilizar junto con sus respectivas funciones, además de la

configuración inicial de la conexión a Google Drive para el levantamiento de las imágenes originales a tratar, así como el guardado de las eventuales ediciones de estas últimas.

Por otro lado, entre las funciones desarrolladas, primeramente, se encuentra la denominada *"crop_image"*, encargada de recortar, con medidas definidas, cierta imagen también previamente definida, para su posterior almacenamiento en una ubicación indicada con anterioridad. Esto es posible gracias al uso de funciones tales como *"imread"* y *"imwrite"* para la lectura y almacenamiento de la imagen respectivamente, al igual que a los operadores de *"image[x_inicial:x_final, y_inicial:y_final]"* que permiten ejecutar un "corte" medido sobre la coordenadas indicadas.

```
def crop_image(ruta_img: str, ruta_img_crop: str, x_inicial: int, x_final: int, y_inicial: int, y_final: int)-> None:
  try:
      image = cv2.imread(ruta_img)

      image_crop = image[x_inicial:x_final, y_inicial:y_final]

      cv2.imwrite(ruta_img_crop, image_crop)

      print("Imagen recortada con éxito. El tamaño de la imagen es de" + str(image_crop.shape))
  except Exception as e:
      print("Ha ocurrido un error:", str(e))
```

Respecto a la función "*show_image*", la misma posee la funcionalidad de mostrar de manera grafica el par de imágenes definidas, indicándoles a su vez a esta visualización, nombres previamente definidos para su fácil reconocimiento. Esto se logra gracias al uso de la librería "*matplotlib.pyplot*", donde en primera instancia se define una figura con su respectivo tamaño, y luego se definen las posiciones y formatos de ambas imágenes indicadas, para que en última instancia estas puedan ser mostradas. En este punto cabe mencionar que para que la librería utilizada pueda mostrar las imágenes de la manera esperada, se hace uso de la función

"cvtColor" de la librería *"cv2"* para hacer una conversión en el formato de los colores, más específicamente se pasa del formato BGR al formato RGB, el cual entiende *"matplotlib"*.

```
def show_image(img_1, img_2, title_1: str, title_2: str):
  plt.figure(figsize=(10, 5))

  plt.subplot(1, 2, 1)
  plt.imshow(cv2.cvtColor(img_1, cv2.COLOR_BGR2RGB))
  plt.title(title_1)

  plt.subplot(1, 2, 2)
  plt.imshow(cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB))
  plt.title(title_2)

  plt.show()
```

Además, la función *"gray_scale_converter",* la cual se encarga de pasar una imagen a una escala de grises, de una manera relativamente sencilla y concisa. Lo que hace es sumar los valores de los tres canales (R G B) de la matriz a lo largo del eje 2 y los divide entre 3 para obtener el promedio. Finalmente, se devuelve la nueva matriz (imagen), pero sin antes convertir el tipo de dato de sus entradas a un *"uint8"*, que vendría a ser un entero sin signo de 8 bits, es decir que se asegura que los datos de la matriz, representadora de una imagen en escala de grises, sean correctos, pudiendo tomar únicamente valores de 0 a 255.

```
def gray_scale_converter(img):
    image_gray = np.sum(img, axis=2) / 3
    return image_gray.astype(np.uint8)
```

Por último, entre las funciones desarrolladas se encuentra *"is_reverse",* cuya funcionalidad radica en determinar si cierta matriz (imagen), posee inversa o no. Esto lo logra delegando la mayor parte de la funcionalidad a la función "*inv*" del módulo *"linalg"* de la

librería "*numpy*". En caso de que la matriz indicada cumpla todas las condiciones, y tenga inversa, se devolverá la misma, por el contrario, si no es posible encontrar la inversa de la matriz, no devolverá nada.

```
def is_reverse(img):
  try:
      reverse = np.linalg.inv(img)
      return True, reverse
  except np.linalg.LinAlgError:
      return False, None
```

Explicadas las funciones utilizadas, el resto de las funcionalidades del programa son bastante triviales. En primera instancia, para mostrar las imágenes indicadas, se hace uso de la función *"show_image"* con sus respectivos argumentos. Tales como los títulos de las imágenes, como las propias imágenes, las cuales se obtuvieron a partir de un directorio en Google Drive.

```
path_1 = '/content/drive/My Drive/Colab Notebooks/Algebra Aplicada/eiffel_tower.jpg'
path_2 = '/content/drive/My Drive/Colab Notebooks/Algebra Aplicada/liberty_statue.jpg'

image_1 = cv2.imread(path_1)
image_2 = cv2.imread(path_2)

show_image(image_1, image_2, 'Imagen 1', 'Imagen 2')
```

Luego, de manera trivial se obtiene los tamaños de estas imágenes cargadas, a través del atributo *"shape"* ofrecido por la librería utilizada.

```
print('Tamaño Imagen 1:', image_1.shape)
print('Tamaño Imagen 2:', image_2.shape)
```

Para el recorte de las imágenes, se hace uso de la función ya definida, *"crop_image"*, a la cual se le especifica de manera selectiva las porciones de las imágenes (coordenadas) a recortar, junto con la ubicación de las imágenes resultante. Teniendo cuidado de que las imágenes recortadas obtenidas sean cuadradas para que se pueda realizar las operaciones solicitadas.

```
cropped_path_1 = '/content/drive/My Drive/Colab Notebooks/Algebra Aplicada/cropped_eiffel_tower.jpg'
cropped_path_2 = '/content/drive/My Drive/Colab Notebooks/Algebra Aplicada/cropped_liberty_statue.jpg'

crop_image(path_1, cropped_path_1, 600, 2100, 1450, 2950)
crop_image(path_2, cropped_path_2, 800, 2300, 500, 2000)

cropped_image_1 = cv2.imread(cropped_path_1)
cropped_image_2 = cv2.imread(cropped_path_2)

show_image(cropped_image_1, cropped_image_2, 'Imagen 1 Recortada', 'Imagen 2 Recortada')
```

De manera similar a lo ya mostrado, para poder visualizar la matriz representadora de una de las imágenes (en este caso la primera), junto con el tamaño de la misma, se hace uso de la función *"print"* y el atributo *"shape"* ofrecido.

```
print('Matriz Imagen 1:', cropped_image_1)
print('Tamaño Imagen 1:', cropped_image_1.shape)
```

Respecto a la atención de la traspuesta de las matrices, la función novedosa en este caso sería "*transpose*", la cual es provista por la librería utilizada, y permite trasponer las dimensiones de la matriz en cuestión (cambiar el orden de sus dimensiones). En este caso se transponen las dimensiones 0 y 1, por lo cual, al hacer esto, se obtiene que la altura de la imagen (matriz) se intercambió con el ancho de la misma, dejando las dimensiones de color tal cual están. Luego, para observar los resultados, tanto en formato de matriz, como en la visualización de las propias imágenes, se hace uso de los ya vistos *"print"*, y la función *"show_image"*.

```
cropped_image_1_transpose = cropped_image_1.transpose(1, 0, 2)
cropped_image_2_transpose = cropped_image_2.transpose(1, 0, 2)

print('Matriz traspuesta Imagen 1:', cropped_image_1_transpose)
print('Matriz traspuesta Imagen 2:', cropped_image_2_transpose)

show_image(cropped_image_1_transpose, cropped_image_2_transpose, 'Imagen 1 Traspuesta', 'Imagen 2 Traspuesta')
```

Por el lado de obtener las imágenes en escalas de grises, únicamente se hace uso de la ya explicada función *"gray_scale_converter"*, para obtener esta nueva imagen en la respectiva escala. Igualmente, las imágenes resultadas se muestran a través de *"show_image"*.

```
cropped_image_1_gray = gray_scale_converter(cropped_image_1)
cropped_image_2_gray = gray_scale_converter(cropped_image_2)

show_image(cropped_image_1_gray, cropped_image_2_gray, 'Imagen 1 Gris', 'Imagen 2 Gris')
```

Para verificar si las matrices (imágenes) manejadas poseen inversa, se hace uso de la función *"is_reverse"*, de la cual se obtiene, tanto el valor booleano que indica la existencia de la inversa de la matriz, como la propia inversa, si existe. Para luego, mediante bloques condiciones, indicar de manera natural la existencia o no de la inversa, y la propia inversa si corresponde, de las dos imágenes manejadas, en escalas de grises.

```
is_reverse_1, reverse_1 = is_reverse(cropped_image_1_gray)
is_reverse_2, reverse_2 = is_reverse(cropped_image_2_gray)

if is_reverse_1:
    print('La inversa de Imagen 1 Gris es:\n', reverse_1)
else:
    print('Imagen 1 Gris no tiene inversa.')

if is_reverse_2:
    print('\nLa inversa de Imagen 2 Gris es:\n', reverse_2)
else:
    print('Imagen 2 Gris no tiene inversa.')
```

Respecto al ajuste de contraste de las imágenes, se seleccionó únicamente la primera de ellas, y se multiplicó su matriz por los escalares 1.5 y 0.5, teniendo en cuenta que esta operación puedo haber repercutido en el rango de valores manejados en la matriz, se hace uso de la función *"clip"* de *"numpy"* para asegurase que los valores (píxeles) de la matriz resultado estén entre el dominio manejado 0 y 255 (los valores menores a 0 se establecen en 0 y los mayores a 255 se establecen en 255), además se establece el tipo de dato ya mencionado "uint8" a los valores de la matriz resultado. Una vez obtenida ambas versiones de la primera imagen, se procede a mostrar los resultados mediante la función *"show_image"*.

```
alpha_1 = 1.5
alpha_2 = 0.5

contrast_1_image_1 = alpha_1 * cropped_image_1_gray
contrast_2_image_1 = alpha_2 * cropped_image_1_gray

contrast_1_image_1 = np.clip(contrast_1_image_1, 0, 255).astype(np.uint8)
contrast_2_image_1 = np.clip(contrast_2_image_1, 0, 255).astype(np.uint8)

show_image(contrast_1_image_1, contrast_2_image_1, f'Imagen 1 α = {alpha_1}', f'Imagen 1 α = {alpha_2}')
```

Para lo que compete al volteo de la imagen en el eje de las *x*, primeramente, lo que se hace es generar una matriz de identidad de igual dimensiones que la imagen a tratar mediante la función *"identity"* de *"numpy"*. A esta matriz de identidad se la invierte horizontalmente con la función *"fliplr"* (de la misma librería) para que todos los valores 1 estén en la anti diagonal y así obtener los resultados esperados al multiplicar las matrices. Para la multiplicación de matrices, se vuelve a utilizar nuevamente la librería "*numpy"*, de la cual se obtiene la función *"dot"*, que facilita la realización de esta operación. Posteriormente, se muestran los resultados obtenidos.

```
identity_matrix = np.identity(cropped_image_1_gray.shape[0], dtype=np.uint8)
flipped_identity_matrix = np.fliplr(identity_matrix)

flipped_image = np.dot(cropped_image_1_gray, flipped_identity_matrix)
show_image(cropped_image_1_gray, flipped_image, 'Imagen 1', 'Imagen 1 Volteada')
```

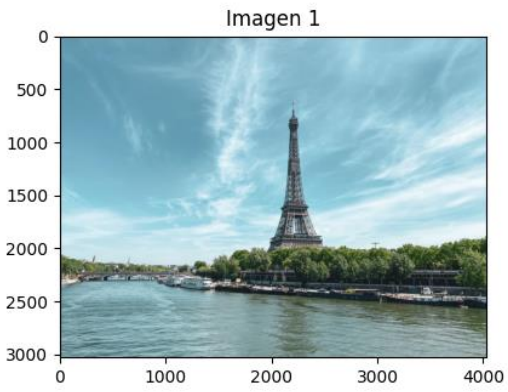
Por último, en lo que respecta al cálculo del negativo de una imagen, lo que se hace es generar una matriz auxiliar del mismo tamaño de la imagen seleccionada (en este caso la primera) con todos sus valores en 255, mediante el uso de la función "*full_like*" de la librería "*numpy*". Una vez obtenida, se resta a esta matriz, la matriz representadora de la imagen, para así obtener el negativo de la imagen. De manera final, se muestran, mediante *"show_image"*, tanto la imagen original en escala de grises, como la nueva imagen resultado de la resta antes mencionada, para que sea mucho más sencillo comparar ambas versiones.

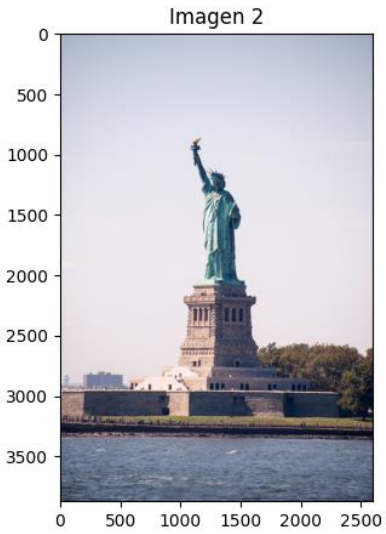
```
auxiliar_matrix = np.full_like(cropped_image_1_gray, 255, dtype=np.uint8)
negative_image = auxiliar_matrix - cropped_image_1_gray

show_image(cropped_image_1_gray, negative_image, 'Imagen 1', 'Imagen 1 Negativa')
```

IV. Resultados

En esta sección, se presentarán los resultados obtenidos a lo largo del proyecto, destacando las observaciones y transformaciones realizadas en las imágenes originales.

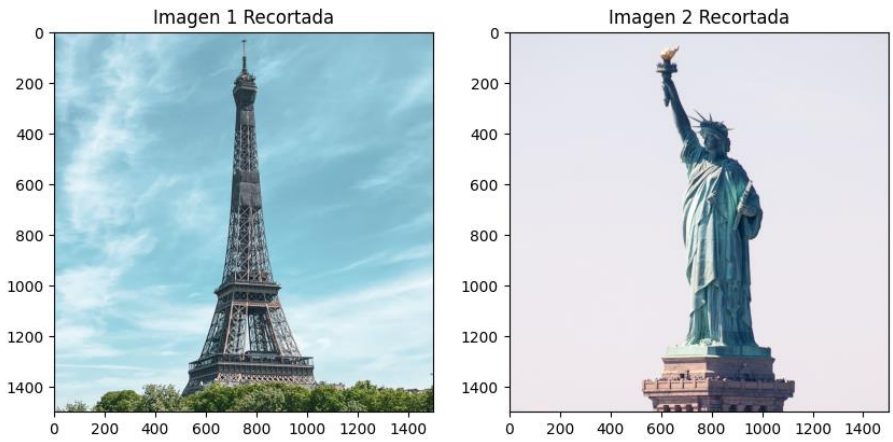
En la primera parte del proyecto, se cargaron las dos imágenes seleccionadas previamente y se visualizaron, imprimiendo adicionalmente el tamaño de cada una. A continuación, se presentan las imágenes originales y sus respectivos tamaños utilizadas:






```
Tamaño Imagen 1: (3024, 4032, 3)
Tamaño Imagen 2: (3872, 2592, 3)
```

En la segunda parte, se recortaron ambas imágenes para asegurarse de que tengan el mismo tamaño y sean cuadradas, como se requería. A continuación, se muestran las imágenes resultantes después del recorte:



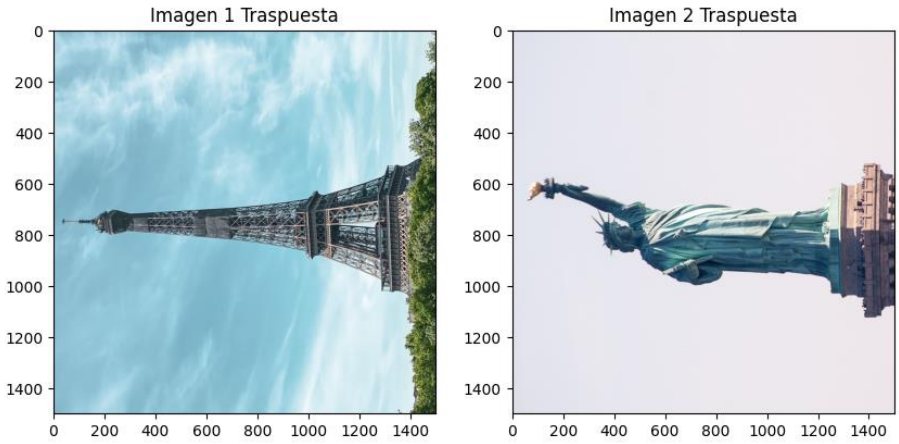
Este paso es fundamental debido a que la compatibilidad de dimensiones es esencial en las operaciones matriciales. Para sumar o restar dos matrices, deben tener las mismas dimensiones, es decir, el mismo número de filas y columnas. Lo mismo se aplica a la multiplicación de matrices, donde la matriz izquierda debe tener un número de columnas igual al número de filas de la matriz derecha. El recorte de las imágenes para que tengan el mismo tamaño y sean cuadradas garantiza esta compatibilidad, permitiendo operaciones entre ellas, como el ajuste de contraste, la transposición y otras operaciones matriciales. Además, la matriz identidad, fundamental en muchas operaciones matriciales, es cuadrada, subrayando la importancia de que las imágenes sean cuadradas para mantener la coherencia en las operaciones.

Para corroborar que las imágenes hayan quedado de dimensión cuadrada, se muestra la imagen como una matriz y además, se imprime su tamaño:

```
Matriz Imagen 1: [[[216 200 154]
  [215 199 153]
  [217 201 155]
  ...
  [205 186 135]
  [206 187 136]
  [206 187 136]]

 [[218 202 156]
  [217 201 155]
  [217 201 155]
  ...
Tamaño Imagen 1: (1500, 1500, 3)
```

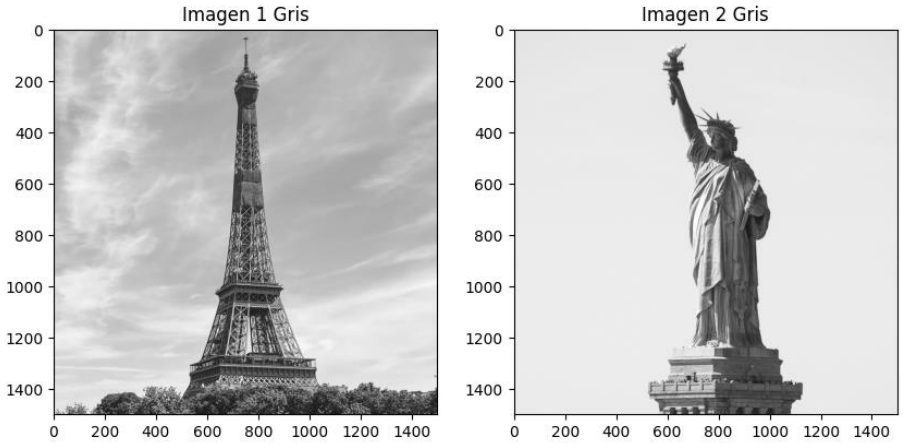
Luego, se procedió a calcular la matriz traspuesta de las imágenes obtenidas en el punto 3. A continuación, se presentan los resultados de este cálculo:

```
Matriz traspuesta Imagen 1: [[[216 200 154]
  [218 202 156]
  [216 200 154]
  ...
  [239 234 213]
  [238 233 212]
  [239 234 213]]

 [[215 199 153]
  [217 201 155]
  [216 200 154]
  ...
Matriz traspuesta Imagen 2: [[[236 224 222]
  [236 224 222]
  [236 224 222]
  ...
  [235 231 237]
  [235 231 237]
  [235 231 237]]
```



La matriz traspuesta es una representación donde las filas se han intercambiado por columnas y viceversa. Esta operación permite alterar la orientación de la imagen, intercambiando su altura y ancho, mientras que los canales de color se mantienen iguales.

En la siguiente parte, se aplicó la conversión a escala de grises a las imágenes recortadas. A continuación, se presentan las imágenes en escala de grises obtenidas:

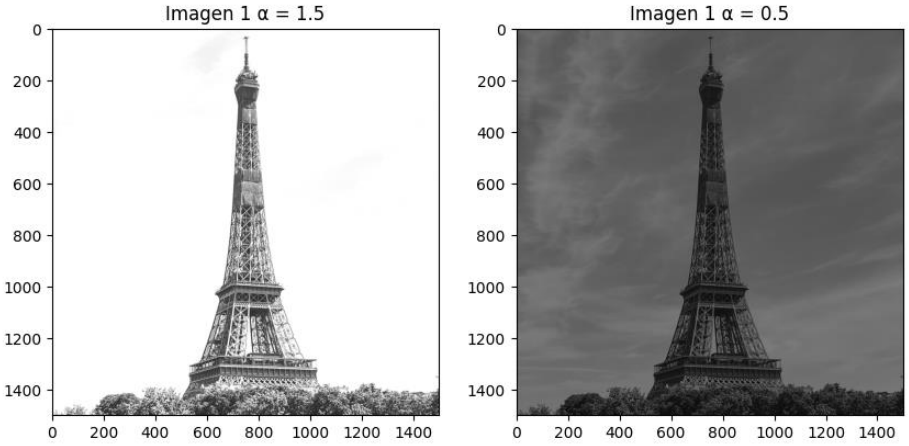


A partir de esta parte en adelante, todas las operaciones fueron realizadas con la “Imagen 1 Gris” obtenida en el paso anterior.

Se verificó si las imágenes en escala de grises tenían inversas. A continuación, se muestran los resultados de la verificación:

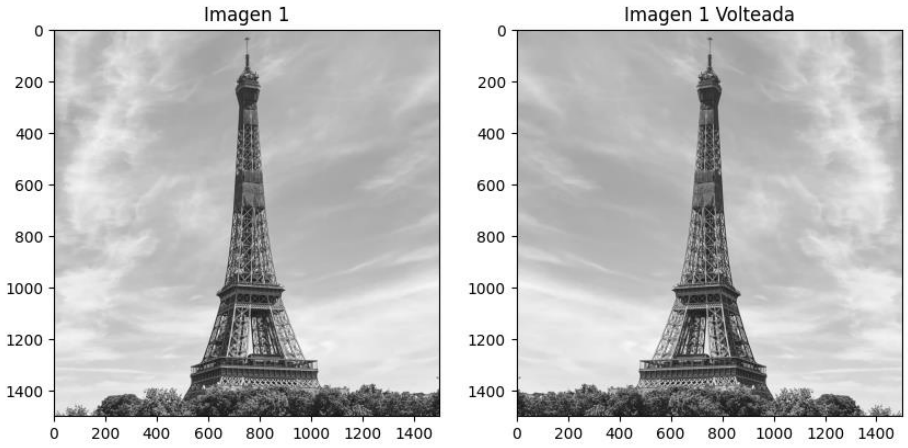
```
La inversa de Imagen 1 Gris es:
 [[-0.04367559  0.01572564 -0.00508553 ...  0.00074861  0.00083479
  -0.00135085]
 [ 0.06500079  0.0014913  -0.02846838 ... -0.00235274 -0.00104811
   0.00212346]
 [-0.05219743 -0.00444728  0.02776704 ...  0.00125521  0.00086443
  -0.00194968]
 ...
 [ 0.02458031  0.02867923 -0.04013598 ... -0.00219285  0.00040302
   0.00050898]
 [-0.02088826 -0.01023728  0.03159698 ...  0.00117855  0.00093124
  -0.00087402]
 [ 0.0229973   0.0218846  -0.04692291 ... -0.00277561  0.00021117
   0.00072845]]

La inversa de Imagen 2 Gris es:
 [[ 0.07813361 -0.06681882 -0.01803839 ... -0.16993306  0.24496575
  -0.19540616]
 [ 0.047287    0.00690233 -0.17002247 ... -0.13124355  0.17576794
  -0.10772425]
 [ 0.04879842 -0.10914226  0.01516794 ... -0.14868064  0.25837629
  -0.19685131]
 ...
 [-0.13623034  0.11255962 -0.02722568 ...  0.55843023 -0.61789705
   0.44910485]
 [ 0.01008744  0.03449038 -0.13280103 ... -0.05784269  0.11779491
  -0.07012532]
 [ 0.00087444 -0.10413963  0.31163759 ...  0.03377928 -0.16179955
   0.10114342]]
```

En esta siguiente parte, se aplicó el ajuste de contraste a una de las imágenes en escala de grises multiplicándola por dos escalares distintos α1 = 1.5 y α2 = 0.5. A continuación, se presentan los resultados para dos casos:



La multiplicación en ambos casos alteró la intensidad del color en ambas imágenes. Para el caso del escalar α1 la intensidad de los valores de los pixeles aumentó (los valores en las entradas de la matriz aumentaron), incrementando así el brillo de la imagen. De forma contraria, para el caso del escalar α2 la intensidad de los valores de los pixeles disminuyó (los valores en las entradas de la matriz disminuyeron), por lo que la imagen se torna más oscura.

Luego, se exploró la multiplicación de matrices y se demostró la no conmutatividad al voltear una imagen alrededor del eje x. A continuación, se muestran los resultados:



Es importante observar aquí que el orden en el que multiplicamos las matrices importa significativamente. Si invertimos el orden y primero multiplicamos la matriz auxiliar por la imagen original, el resultado es una imagen con el mismo contenido, pero sin efecto de reflejo horizontal. Esto demuestra claramente la no conmutatividad de la multiplicación de matrices en la aplicación específica de voltear imágenes.

Por último, se calculó el negativo de una imagen en escala de grises. A continuación, se muestra el resultado:

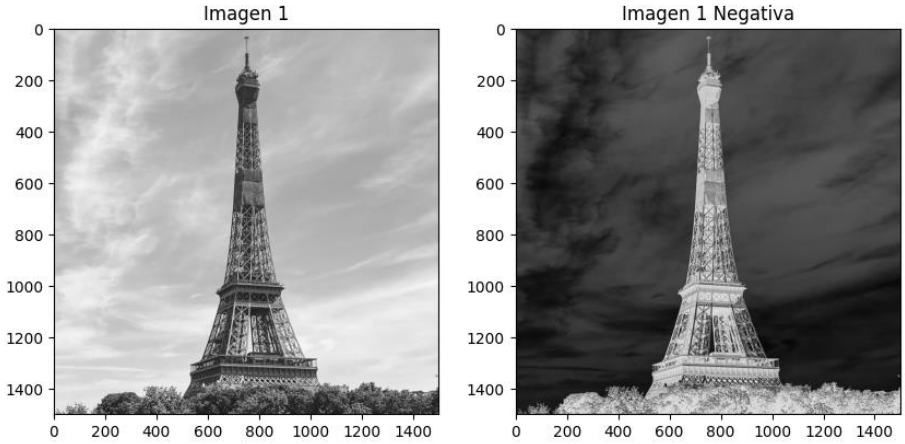


Imagen 1
Imagen 1 Negativa
0
200
400
600
800
1000
1200
1400

V. Conclusiones

En este proyecto, se ha explorado la aplicación de conceptos y técnicas matriciales en el procesamiento de imágenes. Comenzando por comprender cómo una imagen puede representarse como una matriz, donde cada píxel se corresponde con un valor numérico. A lo largo del desarrollo del programa implementado en Python, se utilizaron diversas funciones y operaciones matriciales para transformar y analizar imágenes de manera efectiva.

Una de las principales conclusiones es la versatilidad de las matrices en el procesamiento de imágenes. Desde el recorte de imágenes hasta la conversión a escala de grises, estas operaciones entre matrices permitieron manipular las imágenes de acuerdo con distintas necesidades. Además, se exploró la importancia de la transposición de matrices en la transformación de imágenes y se confirmó la existencia de inversas en ciertos casos.

La aplicación del ajuste de contraste demostró cómo la multiplicación por un escalar puede cambiar la apariencia de una imagen, lo que resalta la utilidad de estas operaciones en el procesamiento de imágenes. Además, al explorar el volteo de imágenes en el eje x, permitió ver que la no conmutatividad de la multiplicación de matrices y cómo el orden de las operaciones afecta el resultado.

Como conclusión final, este proyecto no solo proporcionó una comprensión más profunda de la aplicación de las matrices en el procesamiento de imágenes, sino que también demostró cómo las operaciones matriciales pueden ser herramientas poderosas para modificar y analizar imágenes de manera efectiva.

VI. Referencias Bibliográficas

1. Universidad Católica del Uruguay. (s.f.). Matrices [Presentación de Google Colaboratory]. Álgebra Aplicada, Facultad de Ingeniería. https://colab.research.google.com/drive/1_UlXjT_vlzRfuIw0c8FiKlECYwjtphi9?usp=sharing#scrollTo=XcwbZQ-lovXU
2. Benítez, M; Cabrera, M; López, D; Valle, I. (2023). Proyecto 2: Análisis de imágenes con matrices. [Software Computacional]. https://colab.research.google.com/drive/1AaF5so3jx3rBa7jaIYEXn3fQqMQC72Cg?usp=sharing