Procesamiento de imágenes con matrices

López Salomón María Guadalupe

Diciembre 2022

Dr. Ángel David Reyes Figueroa

Álgebra Matricial

CIMAT

1 Introducción

Desde el surgimiento de técnicas para el procesamiento de imágenes digitales (1960s) hasta el presente, este campo ha ido evolucionando significativamente. Además de las aplicaciones en medicina como son tomografías computarizadas, hasta programas espaciales las técnicas de procesamiento de imágenes se usan en un amplio rango de aplicaciones. Métodos computacionales son usados para mejorar el contraste o codificar los niveles de intensidad de color. El mejoramiento de imágenes y procedimientos de restauración son usados para procesar imágenes degradadas de objetos irreconocibles o resultados experimentales que son muy caros de duplicar. De la mano de estas técnicas computacionales las acompañan métodos matriciales para el procesamiento de imágenes. El campo del procesamiento de imágenes digitales está basado en fundamentos matemáticos y probabilísticos.

2 Objetivos y Motivación

Estamos interesados en exhibir algunas aplicaciones de la teoría matricial dado que representa una herramienta fundamental en la implementación de métodos importantes en el procesamiento de imágenes digitales, mismas que son de interés dada su creciente relevancia y significancia en la actualidad, así como en la creciente demanda de mejoras visuales.

3 Metodología

3.1 Representación digital de Imágenes

Podemos definir una imagen como una función bidimensional $f(x, y)$, con x, y que pertenecen a $\mathbb{R}^2$ donde x y y representan las coordenadas espaciales en el plano cartesiano y la amplitud de f de cualquier par de coordenadas (x, y) se llama la intensidad o nivel de grises de la imagen en ese punto.

Una imagen digital es áquella que se ha discretizado tanto en las coordenadas espaciales como en el brillo y se puede representar matemáticamente por medio de una matriz $F = (f_{ij})$ de dimensiones $m \times n$ donde i denota a la fila y j a la columna de dicha matriz respectivamente. De manera que el producto de m y n es el número de puntos requeridos para representar la imagen. Las filas i y las columnas j identifican un punto de la imagen y el valor del correspondiente elemento de la matriz indica el nivel de gris en ese punto.

Podemos usar valores enteros para describir las coordenadas espaciales: $x = 0, 1, 2, ..., M - 1$ y $y = 0, 1, 2, ..., N - 1$. La sección del plano real que abarcan las coordenadas de una imagen se llama *dominio espacial,* con x y $yvariables$ *espaciales* o *coordenadas espaciales.* Cada elemento de la matriz se llama *image element, picture element, pixel* o *pel.* En particular cada par de coordenadas

se llama *pixel*. La palabra 'pixel' se deriva de 'picture element' y determina la unidad lógica más pequeña de infromación visual usada para construir una imagen.

Por ejemplo, si consideramos una imagen de tamaño 5×12 tendremos algo de la forma:

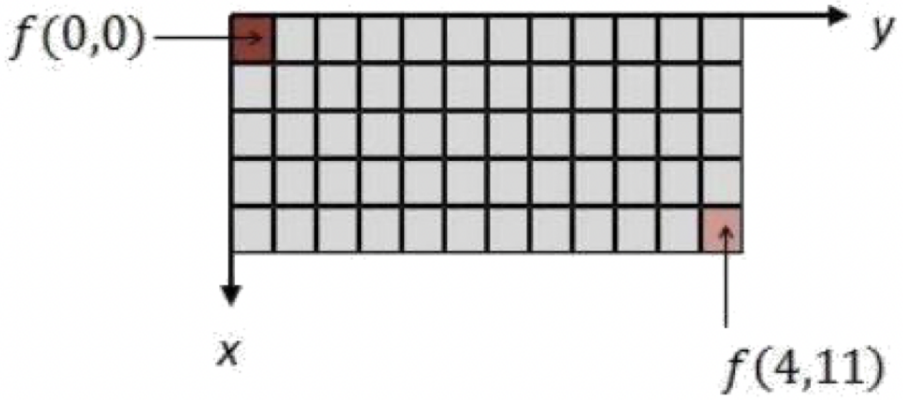

Figure 1: Imagen digital con 5×12 pixeles

De maner general:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

Figure 2: Arreglo númerico para una imagen de $M \times N$ pixeles

En la imagen binaria $f(x,y)$ cada pixel es 0 o 1, i.e., la matriz que define la imagen es una matriz con entradas 0 o 1. En este caso la imagen está configurada a blanco (1) o negro (0). Por otra parte, una imagen en escala de grises también llamada monocromática mide la intensidad de la luz. Las imágenes en escala

de grises varían en un rango entre $[0, NC-1]$ donde NC representa el número de niveles de gris. La imagen en escala de grises más común usan 8o 16 pixeles por pixel, resultando en $NC = 2^8 = 256$ o $NC = 2^{16} = 65536$ niveles de grises diferentes.

En una imagen a color el valor de $f(x, y)$ para cada pixel mide la intensidad del color en cada pixel y se representa por un vector con componentes de colores. Algunos de los espacios a color más comunes son: el *RGB* (Red, Green y Blue), HSV (hue, saturation and value) y CMYK (Cyan, Magenta, Yellow y Black). En particular para el espacio de color RGB, los tres parámetros representan las intensidades de los tres colores primarios del modelo, definimos un espacio tridimensional con tres direcciones ortogonales (R, G, y B). La implementación del modelo RGB asigna valores enteros a los pixeles entre 0 y 255 (8-bit). De lo anterior podemos inferir que los pixeles que componen una imagen pueden considerarse como un vector, de manera que una imagen prototipo se puede considerar como un vector base, que puede ser útil para describir otras imágenes usando el *producto interno.*

De manera similar, un conjunto de imágenes pueden formar un espacio para describir una variable en una dimensión y por lo tanto, definir un espacio. Una imagen en particular, se puede representar por un vector con los tres productos internos obtenidos mediante las tres imágenes prototipo.

3.2 Operaciones fundamentales con matrices

Como se estudió en el curso las operaciones más fundamentales que podemos realizar con matrices son: suma, resta y multiplicación.

A continuación exhibiremos las operaciones mencionadas utilizando como sujeto de prueba a la siguiente imagen:

Figure 3: Sujeto de prueba

Asi mismo utilizaremos Spyder para el tratamiento de la imagen, de manera que tenemos:

```
@author: mariasalomon
"""

#Cargamos las respectivas librerías que nos permitiran leer y procesar la imagen
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage import io
import cmath as cm
import scipy.fftpack as sfft


#Visualización imagen
imagen = io.imread('birthday.jpg')
io.imshow(imagen)
io.show()
io.imsave('birthday.jpg', imagen)

#Notamos que es una imagen rectangular de 757x1600 pixeles

#Para facilitar el tratamiento de la misma convertiremos nuestra imagen RGB a escala de grises:

#Lectura como matriz, conversión a escala de grises y cambio de tamaño a matriz cuadrada

imageGray = cv2.imread('birthday.jpg',0)
print(imageGray) #visualización como arreglo  y en escala de grises
imag=cv2.resize(imageGray, (757,757), interpolation=cv2.INTER_CUBIC) #Convertimos a una imagen cuadrada
#Note que el reescalamiento que se hace para cambiar el tamaño consiste en realizar un transformación de coordenadas
#Cuando se recorta la imagen se define una nueva imagen que se rellena con 0 y 1 respectivamente


io.imshow(imag)
io.show()

# 1. SUMA de imágenes como matrices

suma=cv2.add(imag,imag)
print('La suma de imágenes como matrices:')
print (cv2.add(imag,imag))

io.imshow(suma)
io.show()
#Podemos notar como al sumar las dos imágenes la imagen resultante se 'aclara' debido a que los pixeles resultantes son la suma de 0 y 1


#2. RESTA de imágenes como matrices
```

Figure 4: Visualización, resize y suma

Del código anterior obtuvimos los siguientes resultados:

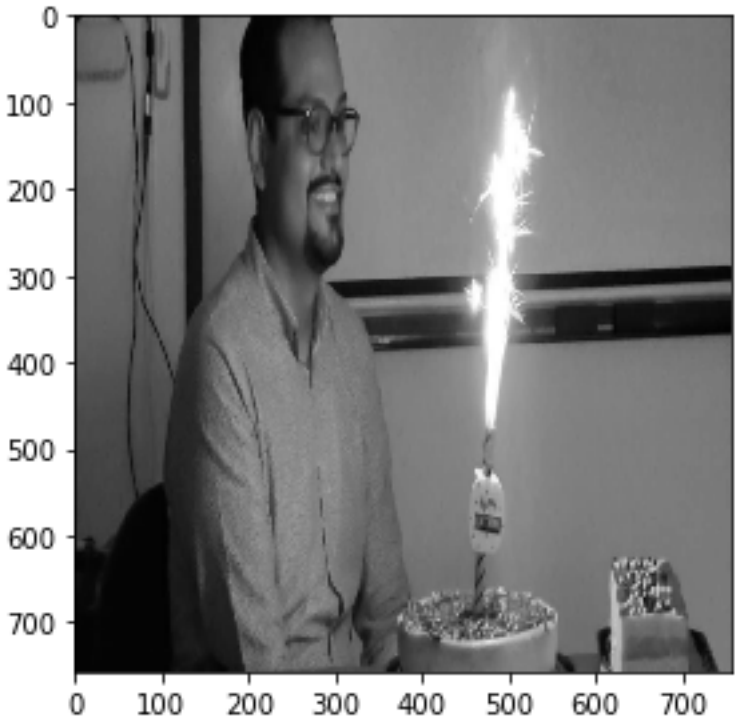

Figure 5: Resize

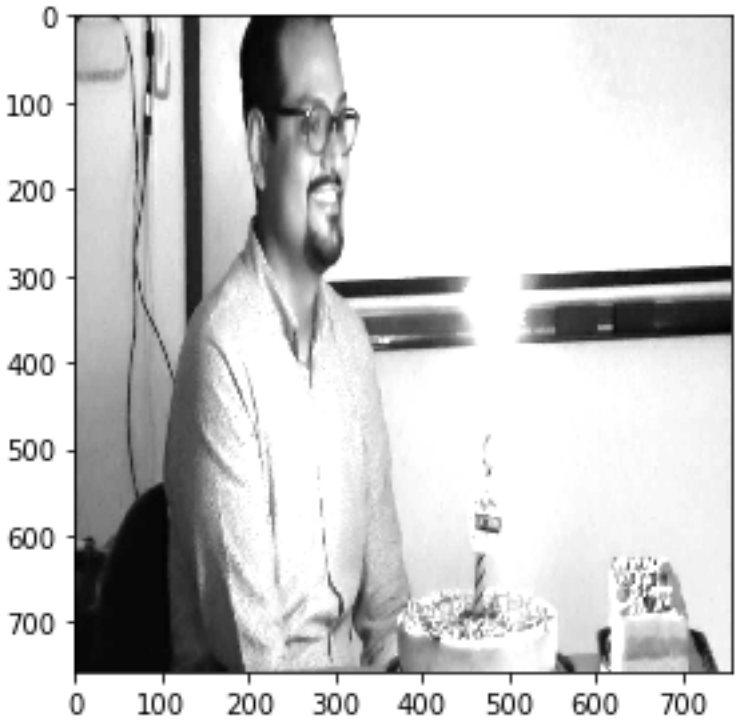

Figure 6: Suma de imágenes

```
#2. RESTA de imágenes como matrices

rest=cv2.subtract(imag,imag)
print('La resta de imágenes como matrices:')
print (cv2.subtract(imag,imag))

io.imshow(rest)
io.show()

#La imagen en negro tiene sentido dado que estamos restando dos imágenes iguales. Si por el contrario definieramos una nueva imagen de la forma:

imag1=2*imag
io.imshow(imag1)
io.show()

rest1=cv2.subtract(imag,imag1)
print('La resta de imágenes como matrices:')
print (cv2.subtract(imag,imag1))

io.imshow(rest1)
io.show()

#3. Multiplicación de matrices como imágenes
 #Multiplicación de la misma imagen
mult=cv2.multiply(imag,imag)
io.imshow(mult)
io.show()
 #Multiplicación de imágenes ligeramente diferentes

mult1=cv2.multiply(imag,imag1)
io.imshow(mult1)
io.show()

#Observamos los diferentes patrones que se forman en la imagen haciendo pequeñas variaciones

#Ahora veamos que sucede si se realiza la multiplicación de la matriz(imagen) por un escalar

mult2=cv2.multiply(imag,5) #escalar=5
io.imshow(mult2)
io.show()
```

Figure 7: Código resta y mult.

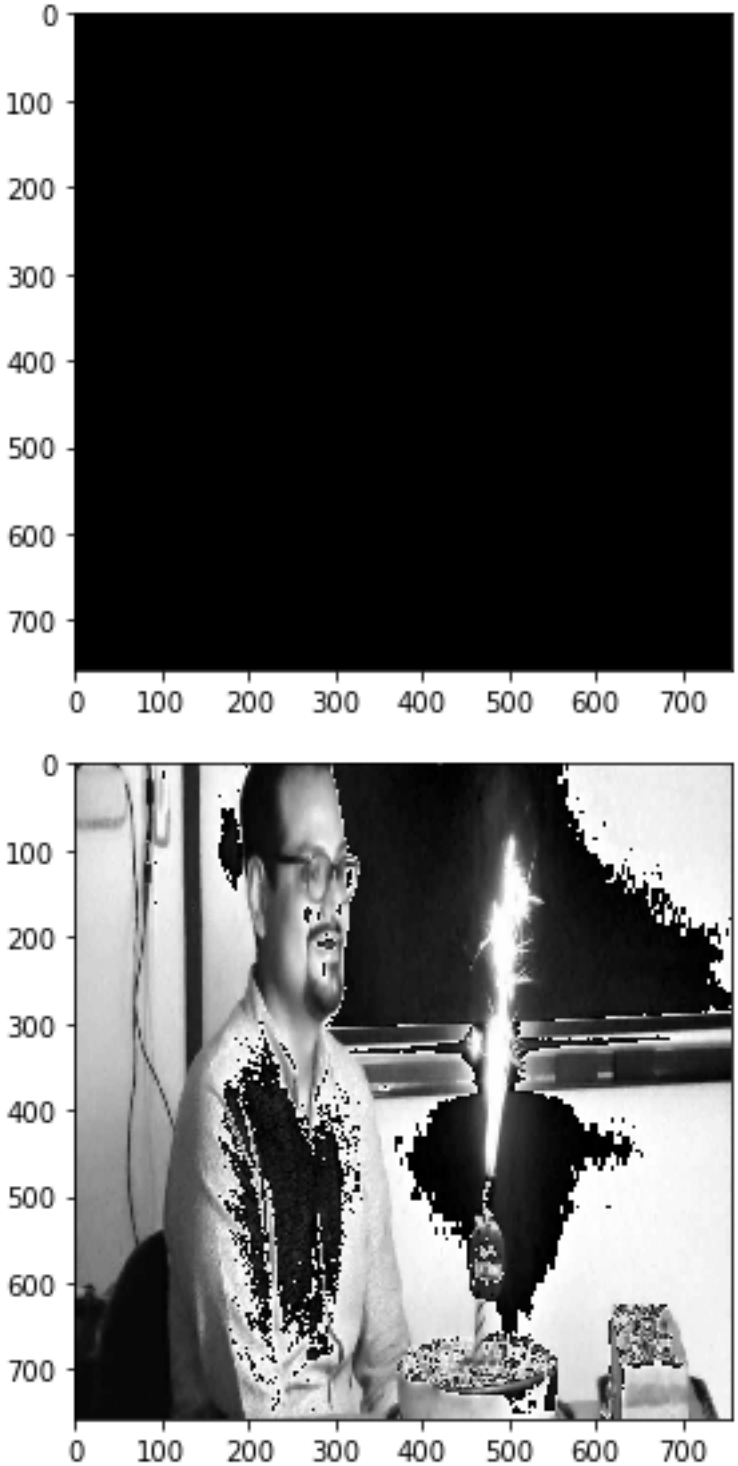

Figure 8: Resta de imágenes iguales y definición nueva imagen

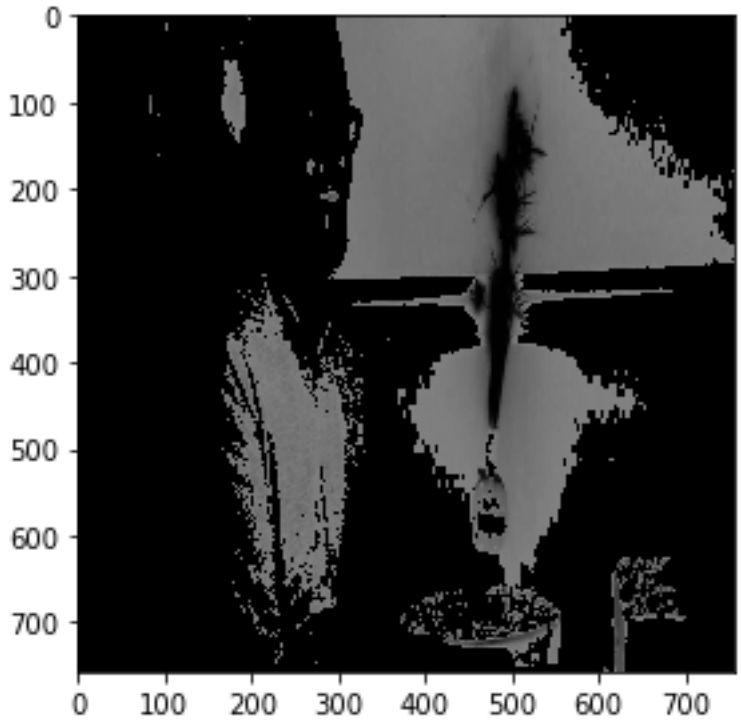

Figure 9: Resta de imágenes diferentes

En particular, la multiplicación de matrices es de vital importancia dado que se pueden definir matrices cuyos elementos sean todos 0 o 1 que denotamos como 'máscaras' o filtros del mismo tamaño que el de la matriz a la que se la aplicarán. Este 'mask' se opera sobre la imagen inicial haciendo la respectiva multiplicación elemento a elemento. Su importancia radica sobretodo en procesamiento de señales en donde a través de estas máscaras podemos filtrar señales o frecuencias que perturban nuestra información. En particular , cuando de imágenes espaciales hablamos, podemos filtrar ruido y aberraciones o también ajustar los niveles de definición de los pixeles que conforman la imagen permitiendo mayor definición.

3.3 Rotaciones.

La rotación de un objeto desde un ángulo $\theta > 0$ (en el sentido horario) alrededor del origen constituye una transformación geométrica que no deforma nuestro objeto de interés.

Matrices de transformación pueden ser aplicadas consecutivamente para generar movimientos compuestos y rotaciones a la imagen a la que se le aplica. Así mismo para escalar un objeto en un punto arbitrario $p = (x_1, y_1)$ es necesario realizar las siguientes transformaciones elementales:

1. Mover el pixel $p = (x_1, y_1)$ al origen del sistema coordenado, aplicando la traslación $T = (-x_1, -y_1)$ a todos los pixeles del objeto.
2. Cambiar las dimensiones del objeto al aplicar un escalamiento $S = (s_x, s_y)$.
3. Mover el pixel $P = (x_1, y_1)$ a su posición original

Podemos obtener la matriz de la transformación compuesta a través de:

$$
\begin{aligned}
T_c &= T(x_1, y_1) \times S(s_x, s_y) \times T(-x_1, -y_1) \\
&= \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} s_x & 0 & x_1(1 - s_x) \\ 0 & s_y & y_1(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Figure 10: Matriz de transformación compuesta T_c

Figure 11: Rotación y traslación

```
#Transformaciones

ancho=imag.shape[1] #columnas
alto=imag.shape[0] #filas

#Traslación

M=np.float32([[1,0,100],[0,1,150]]) #Definimos las filas y columnas en donde aplicaremos la transformación
imageOut=cv2.warpAffine(imag,M,(ancho,alto))
io.imshow(imageOut)
io.show()

#Rotación

M1 = cv2.getRotationMatrix2D((ancho//2,alto//2),60,1) #Especificamos el centro de la imagen, ángulo de rotación
imageOut1 = cv2.warpAffine(imag,M1,(ancho,alto))
io.imshow(imageOut1)
io.show()
```

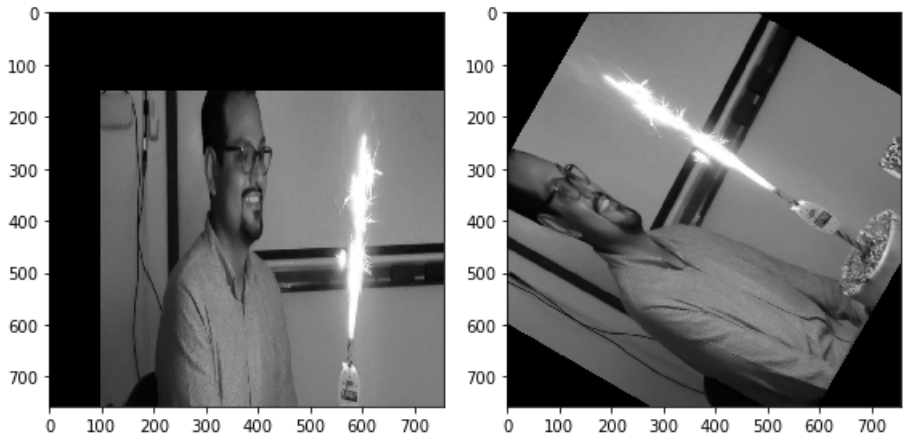

Figure 12: Traslación y rotación

3.4 Filtrado espacial

El filtrado o filtraje espacial permite acentuar una imagen e incrementar el detalle o suavizarla y reducir el ruido. Existen dos tipos principales de filtraje espacial: lineal y no lineal. El lineal se lleva a cabo aplicando una máscara de $p \times q$ a cada pixel de la image, i.e. elemento a elemento (multiplicación de

imágener-matrices). Estas máscaras son llamadas frecuentementes filtros y son aplicdos a través de correlación o convolución.

Un ejemplo de un filtro lineal es el filtro promedio el cual cambia la intensidad de valores $f(x, y)$para los valores promedio de pixeles en una vecindad (x, y).

Cabe recalcar que para filtrados más especializados conllevan herramientas más específicas, sin embargo las bases de las mismas radican en las operaciones hasta ahora mencionadas.

```
#Filtraje espacial

#Definimos un filtro circular que deja pasar únicamente las 'frecuencias' espaciales del centro, esto significa que
#los elementos que se encuentren en las orillas no se reflejarán en nuestra nueva imagen:


#Butterworth low pass filter
N,N = imag.shape
H= np.zeros((N,N), dtype=np.float32)
D0=40 #corte de frecuencia
n=1 #orden
for u in range(N):
    for v in range(N):
         D=np.sqrt((u-N/2)**2 + (v-N/2)**2)
         H[u,v]=1 / (1+ (D/D0)**(2*n))

plt.figure(5)
plt.figure(figsize=(10,8))
plt.title('Cómo se ve el filtro ')
plt.imshow(H, cmap='gray')
plt.show()

#Filtro de frecuencia en el dominio de la imagen
Gshift= imag * H
plt.figure(1)
plt.figure(figsize=(10,8))
plt.title("Imagen filtrada al centro ")
plt.imshow(np.abs(Gshift), cmap='inferno')
plt.show()
```

Figure 13: Filtraje espacial

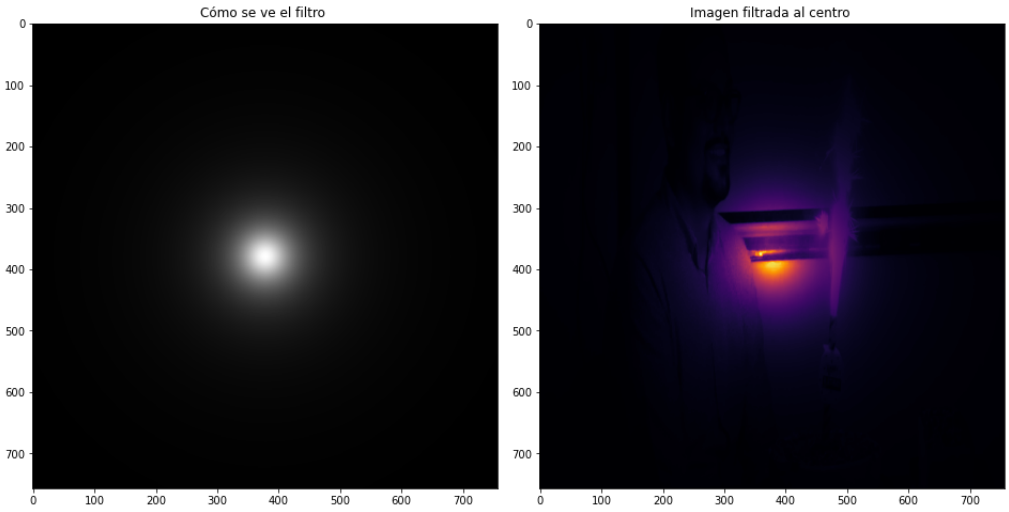

Figure 14: Filtro e imagen filtrada

Notamos como el filtro butterworth solo deja pasar las frecuencias espaciales del centro impidiendo el paso de las frecuencias que se encuentran en las orillas. Este tipo de filtros es muy importante en la reconstrucción de imágenes y en la limpieza de las mismas, para suavizar o reducir ruido.

Pueden implementarse diversos filtros espaciales, dependiendo del interés que se tenga en el tratamiento de la imagen a procesar, sin embargo, las bases operacionales encuentran su raíz en el álgebra matricial como se indicó previamente.

4 Conclusiones

Es fácil ver que el procesamiento de imágenes es fundamental en la vida cotidiana por lo que implementar técnicas rigurosas y efectivas de procesamiento de imágenes digitales, así como computacionalmente eficientes es un objetivo primordial.

Entender las diversas herramientas y conceptos que nos brinda el álgebra matricial es indispensable para desarrollar mejores procesamientos.

La cantidad de información visual digital que se genera hoy en día nos obliga a ahondar en la investigación e implementación de mejores técnicas de procesamiento y reconstrucción digital por lo que el diseño efectivo de filtros y transformaciones usando conocimientos algebraicos es imprescindible. Hasta ahora, los conocimientos obtenidos resultan limitados para el procesamiento digital sin embargo, son un comienzo suficiente para realizar tratamientos efectivos.

5 Bibliografía

- Gonzalez, R.C., and Woods, M.P. Digital image processing. Prentice Hall, 2nd edition, 2002
- Pratt, W.K. Digital image processing. John Willey & Sons, Inc., 3rd edition, 2001.
- Como trasladar, rotar, escalar y recortar una imagen | Python – OpenCV. (2020, 10 abril). OMES. Recuperado 6 de diciembre de 2022, de https://omes-va.com/trasladar-rotar-escalar-recortar-una-imagen-opencv/