

practical1

October 15, 2024

Nombre: Pablo Chantada Saborido

Correo: pablo.chantada@udc.es

[214]: # ======DEPENDENCIAS=====

```
import skimage.io as io
import skimage.data as data
import skimage.util as util
import skimage.exposure as exposure
from skimage.color import rgb2gray, rgb2lab, rgb2hsv
from skimage.filters import window
from skimage.filters import gaussian
from skimage.draw import disk

import random
import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

1 Ejercicio 1.1.- Leer y visualizar imágenes

- a) Lea la imagen 'lena.png'; llamemosla lena1. Escriba por pantalla su tamaño (shape), sus valores minimo y maximo, y su tipo. Visualice la imagen.

[215]:

```
def image_characteristics(image, image_name="Imagen"):
    print(f"Usando {image_name}:\n")
    print("Tamaño: ", image.shape)
    print("Valor Maximo: ", np.max(image))
    print("Valor Minimo: ", np.min(image))
    print("Tipo: ", image.dtype)
```

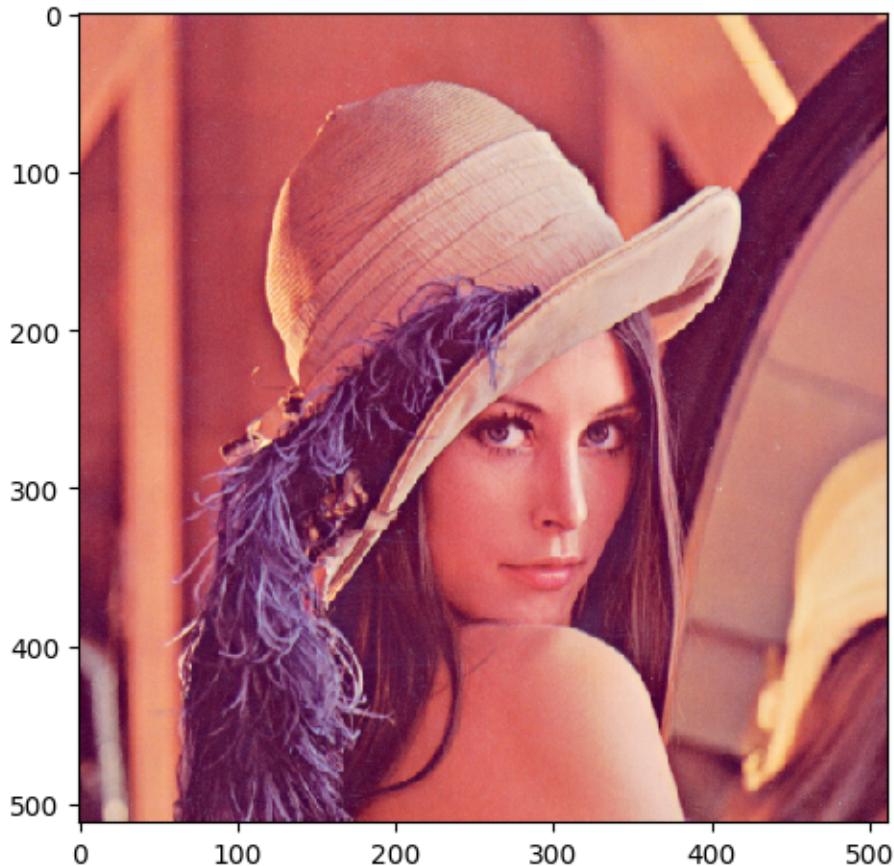
[216]:

```
lena1 = io.imread("lena.png")
# Características de la imagen
image_characteristics(lena1, "lena.png")
io.imshow(lena1)
```

Usando lena.png:

```
Tamaño: (512, 512, 3)
Valor Maximo: 255
Valor Minimo: 3
Tipo: uint8
```

[216]: <matplotlib.image.AxesImage at 0x7cd403f1ef00>



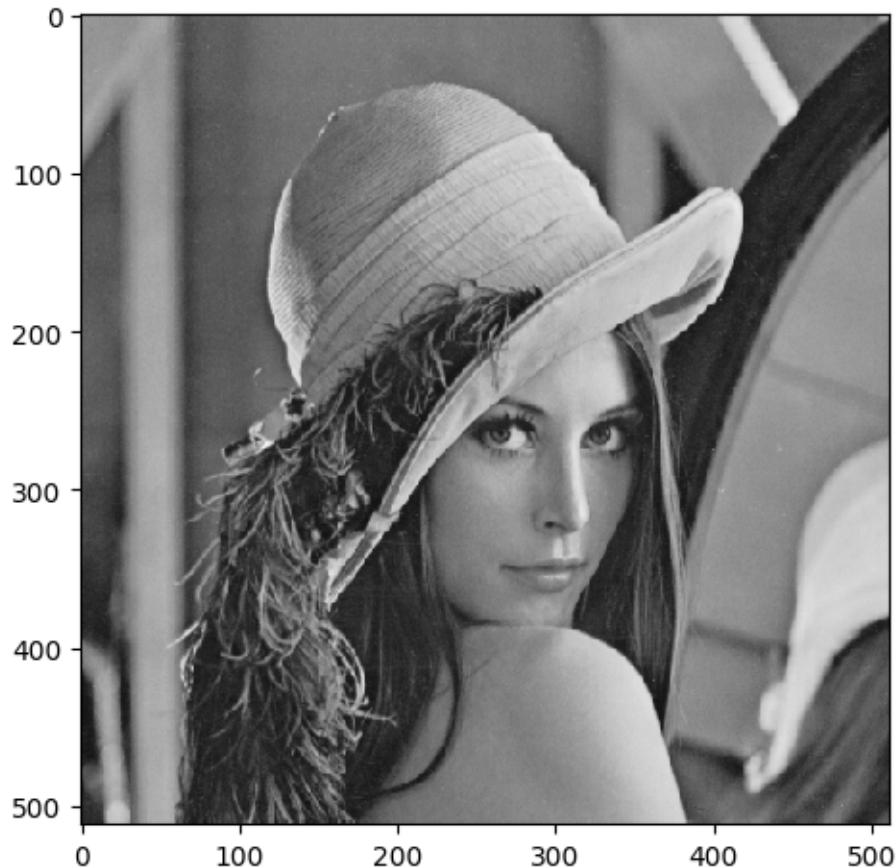
b) Lea la imagen 'lena.png', usando el parametro 'as gray=True'; llamemosle lena2. Escriba por pantalla su tamaño, sus valores mínimo y máximo, y su tipo. Visualice la imagen, consiguiendo que se vea en escala de grises.

```
[217]: lena2 = io.imread("lena.png", as_gray=True)
# Caracteristicas de la imagen
image_characteristics(lena2, "lena.png")
io.imshow(lena2)
```

Usando lena.png:

```
Tamaño: (512, 512)
Valor Maximo: 0.9654356862745097
Valor Minimo: 0.07254666666666666
Tipo: float64
```

```
[217]: <matplotlib.image.AxesImage at 0x7cd416537500>
```



c) Divida el valor de la imagen lena2 por 2 y sumele 0,25; llamemosle lena3 al resultado. Escriba por pantalla sus valores mínimo y máximo. Visualice en subfiguras anexas lena2 y lena3, usando escala de grises, y consiguiendo que las diferencias de brillo sean apreciables visualmente.

```
[218]: lena3 = np.divide(lena2, 2) + 0.25
# Caracteristicas de la imagen
image_characteristics(lena2, "lena2.png")

plt.figure(figsize=(10, 5))

# lena2 brillo por defecto
plt.subplot(1, 2, 1) # lena2
io.imshow(lena2)
```

```

plt.title("lena2 - Original")

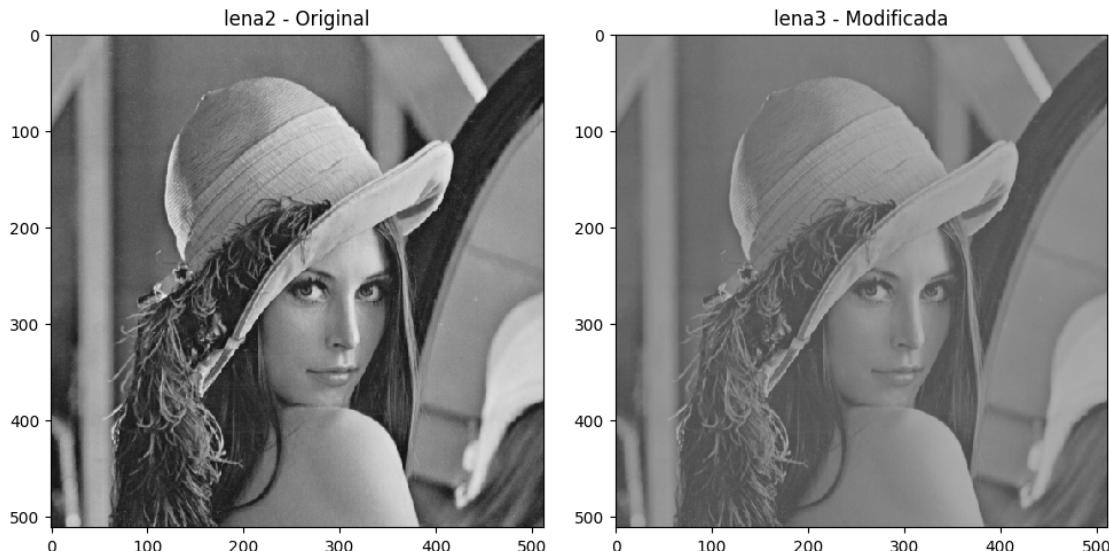
# lena3 menos brillo
plt.subplot(1, 2, 2) # lena3
io.imshow(lena3)
plt.title("lena3 - Modificada")

# Mostrar las imágenes
plt.show()

```

Usando lena2.png:

Tamaño: (512, 512)
 Valor Maximo: 0.9654356862745097
 Valor Minimo: 0.07254666666666666
 Tipo: float64



2 Ejercicio 1.2.- Escribir imágenes en disco

- a) Guarde las imágenes lena2 y lena3 del Ejercicio 1.1, en los ficheros lena2.png y lena3.png, respectivamente, de la carpeta resultados. Asegurese de que la diferencia de intensidades es apreciable en las imágenes guardadas (con un visor externo, o volviéndolas a leer)

```
[219]: path = "resultados/"
# Si la carpeta no existe la creamos
if not os.path.exists(path):
    os.mkdir(path)
```

```

# Convertimos el rango de la imagen de [0,1] a [0,255]
# Pasamos el formato a uint8 y escala de grises
lena2_converted = Image.fromarray((lena2 * 255).astype(np.uint8)).convert('L')
lena3_converted = Image.fromarray((lena3 * 255).astype(np.uint8)).convert('L')

# Guardamos las imágenes usando PIL o skimage
lena2_converted.save(os.path.join(path, "lena2.png"))
lena3_converted.save(os.path.join(path, "lena3.png"))

```

- b) Repita la operación anterior, pero multiplicando antes las imágenes por 10, y almacenando los resultados en lena2b.png y lena3b.png, respectivamente.

[220]:

```

lena2_mult = lena2 * 10
lena3_mult = lena3 * 10
lena2_converted = Image.fromarray((lena2_mult * 255).astype(np.uint8)) .
    convert('L')
lena3_converted = Image.fromarray((lena3_mult * 255).astype(np.uint8)) .
    convert('L')

# Guardamos las imágenes usando PIL o skimage
lena2_converted.save(os.path.join(path, "lena2b.png"))
lena3_converted.save(os.path.join(path, "lena3b.png"))

```

- **¿Qué ocurre y por qué?**

Las imágenes presentan fallos y distorsiones porque los valores de los píxeles exceden el rango permitido para imágenes de 8 bits, que es de 0 a 255. Esto provoca que se tomen valores incorrectos durante el procesado, provocando distorsiones visuales con respecto a la imagen original.

- **¿Cómo debemos hacer para que las diferencias en las imágenes se puedan almacenar?**

Una forma de solucionar esto es normalizar los valores de la imagen para que estén en un rango específico, como [0, 1]. De esta manera, evitamos que los valores se salgan de los límites permitidos. Para ello se puede utilizar el siguiente bloque de código:

```

# Normalizamos los valores para que se mantengan dentro del rango [0, 1]
image_normalized = (lena - np.min(lena)) / (np.max(lena) - np.min(lena))

# O podemos usar np.clip para asegurarnos de que los valores no exceden el rango al multiplicar
lena2_mult = np.clip(lena2 * 10, 0, 1)

```

Esto garantiza que, tras cualquier operación que aumente los valores de los píxeles (como la multiplicación por 10 en este caso), los valores se mantendrán dentro del rango, evitando distorsiones en el guardado o visualizado.

3 Ejercicio 1.3.- Enteros, Flotantes y Booleanos

- a) Cree tres imágenes de degradado, de tamaño 100×100 , y:
- A: tipo np.float64, y valores de 0.1 a 0.9.
 - B: tipo np.uint8, y valores de 25 a 230.
 - C: tipo np.int32, y valores de -230 a 230.

```
[221]: def intensity_gradient(size=(100, 100), dtype=np.uint8, values=(0, 255)):
    rows, cols = size
    mn, mx = values
    result = np.linspace(mn, mx, rows*cols).reshape(rows, cols)
    return result.astype(dtype)

A = intensity_gradient(size=(100, 100), dtype=np.uint8, values=(25, 230))
B = intensity_gradient(size=(100, 100), dtype=np.int32, values=(-230, 230))
C = intensity_gradient(size=(100, 100), dtype=np.float64, values=(0.1, 0.9))
```

b) Visualice las tres imágenes, en escala de grises y sin normalizar, y reporte su valor mínimo y máximo.

```
[222]: # Características de la imagen A
image_characteristics(A, "A")

# Características de la imagen B
print()
image_characteristics(B, "B")

# Características de la imagen C
print()
image_characteristics(C, "C")

# Visualizamos las imágenes
plt.figure(figsize=(15, 5))

# lena2 brillo por defecto
plt.subplot(1, 3, 1) # lena2
io.imshow(A, cmap="grey")
plt.title("A - np.float64, 0.1 a 0.9")

# lena3 menos brillo
plt.subplot(1, 3, 2) # lena3
io.imshow(B, cmap="grey")
plt.title("B - np.uint8, 25 a 230")

# lena3 menos brillo
plt.subplot(1, 3, 3) # lena3
io.imshow(C, cmap="grey")
plt.title("C - np.int32, -230 a 230")

# Mostrar las imágenes
plt.show()
```

Usando A:

Tamaño: (100, 100)

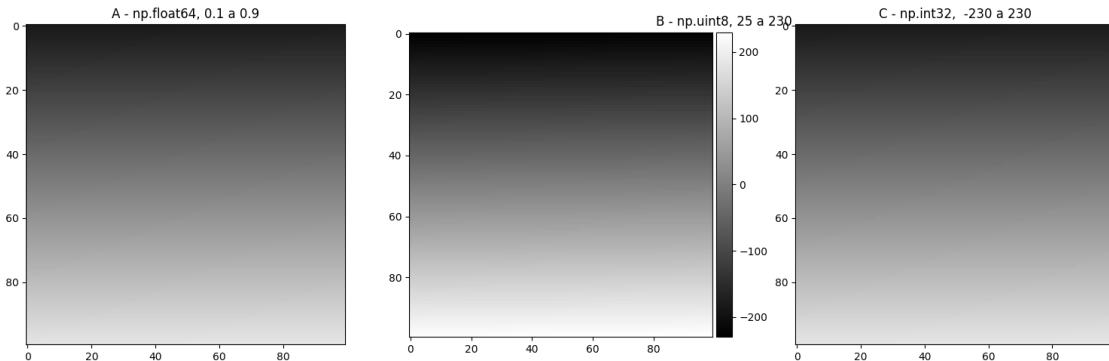
```
Valor Maximo: 230
Valor Minimo: 25
Tipo: uint8
```

Usando B:

```
Tamaño: (100, 100)
Valor Maximo: 230
Valor Minimo: -230
Tipo: int32
```

Usando C:

```
Tamaño: (100, 100)
Valor Maximo: 0.9
Valor Minimo: 0.1
Tipo: float64
```



c) Explore el resultado de las funciones img as float, img as uint, img as int e img as bool de skimage sobre las tres imágenes anteriores, así como la salida de la función skimage.dtype_limits aplicada sobre las imágenes de entrada y salida. Explique lo que ocurre.

```
[223]: # Mostrar solo la primera fila para cada conversión
```

```
img_floatA = util.img_as_float(A)
img_floatB = util.img_as_float(B)
img_floatC = util.img_as_float(C)
print("\nUsing img_as_float: ", img_floatA[0][0], img_floatB[0][0], img_floatC[0][0]) # Mostramos solo la primera fila

img_uintA = util.img_as_uint(A)
img_uintB = util.img_as_uint(B)
img_uintC = util.img_as_uint(C)
```

```

print("\nUsing img_as_uint: ", img_uintA[0][0], img_uintB[0][0], img_uintC[0][0]) # Mostramos solo la primera fila

img_intA = util.img_as_int(A)
img_intB = util.img_as_int(B)
img_intC = util.img_as_int(C)
print("\nUsing img_as_int: ", img_intA[0][0], img_intB[0][0], img_intC[0][0]) # Mostramos solo la primera fila

img_boolA = util.img_as_bool(A)
img_boolB = util.img_as_bool(B)
img_boolC = util.img_as_bool(C)
print("\nUsing img_as_bool: ", img_boolA[0][0], img_boolB[0][0], img_boolC[0][0]) # Mostramos solo la primera fila

```

Using img_as_float: 0.09803921568627451 -1.0710209613065333e-07 0.1

Using img_as_uint: 6425 0 6554

Using img_as_int: 3212 -230 3276

Using img_as_bool: False False False

- **Explique lo que ocurre**

Cada función convierte la matriz de la imagen a un tipo específico de dato. La característica más resaltante es el caso de **uint** e **int**. Al usar un formato sin signo como es el caso del **uint** los valores tienen el doble de rango positivo, pero no presentan valores negativos; dejando estos a 0. Lo contrario ocurre con los **int** ya que tenemos la mitad de rango positivo, pero lo compensamos con la presencia de valores negativos. Además, automáticamente se cambia el formato de **int32** a **int16**, ya que se puede almacenar la imagen en un formato que presenta menor consumo de memoria.

Para las otras dos funciones, en **img_as_float** puede tomar valores de [0, 1] o de [-1, 1] dependiendo de si el dato anterior tenía signo o no. La función **img_as_bool** pone como **True** los valores superiores a la mitad del valor máximo posible en el tipo de dato original, y **False** al resto y negativos.

4 Ejercicio 1.4.- Indexacion

a) Sobre una copia de brick, asigne un valor de intensidad "blanco puro", a la ventana definida por las filas 100 a 150, y las columnas 20 a 120. Visualice el resultado, su valor maximo, su valor minimo y el tipo

```
[224]: brick = data.brick()      # 512x512
astro = data.astronaut()

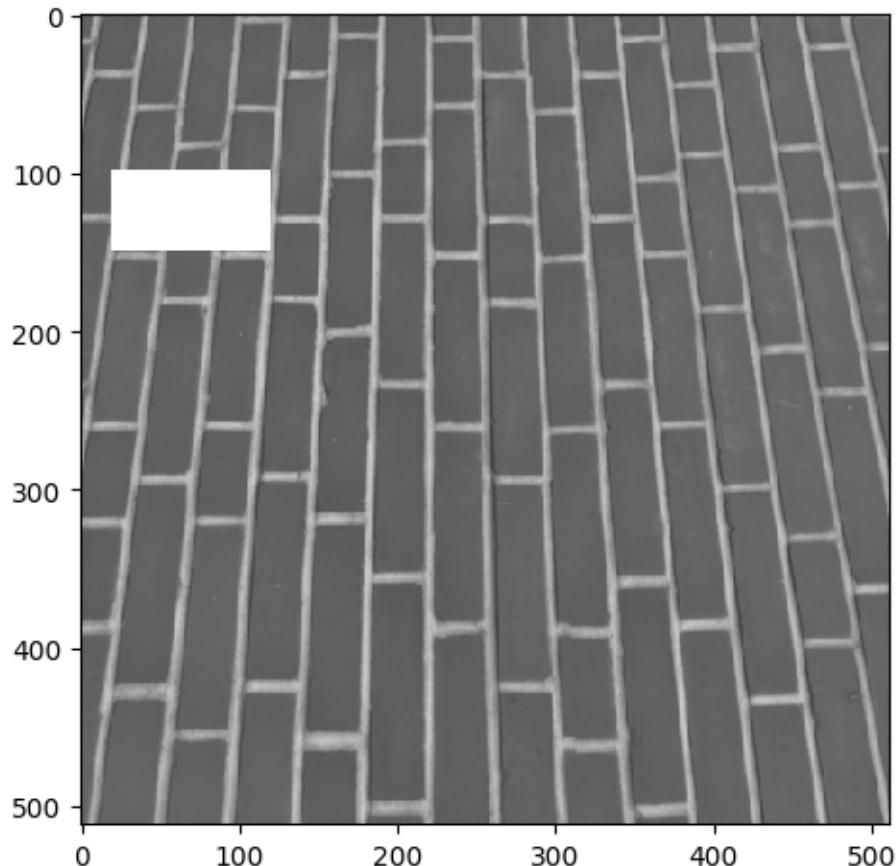
white_brick = brick.copy()
# Blanco puro
```

```
white_brick[100:150, 20:120] = 255  
image_characteristics(white_brick, "White Brick")  
io.imshow(white_brick)
```

Usando White Brick:

Tamaño: (512, 512)
Valor Maximo: 255
Valor Minimo: 63
Tipo: uint8

[224]: <matplotlib.image.AxesImage at 0x7cd4165817f0>



b) Cree una mascara binaria (llamemosle brickmsk) que valga 1 en las posiciones donde brick sea mayor que el 50 % del rango dinamico y 0 en el resto. Visualice el resultado, asi como su valor maximo, su valor minimo y el tipo

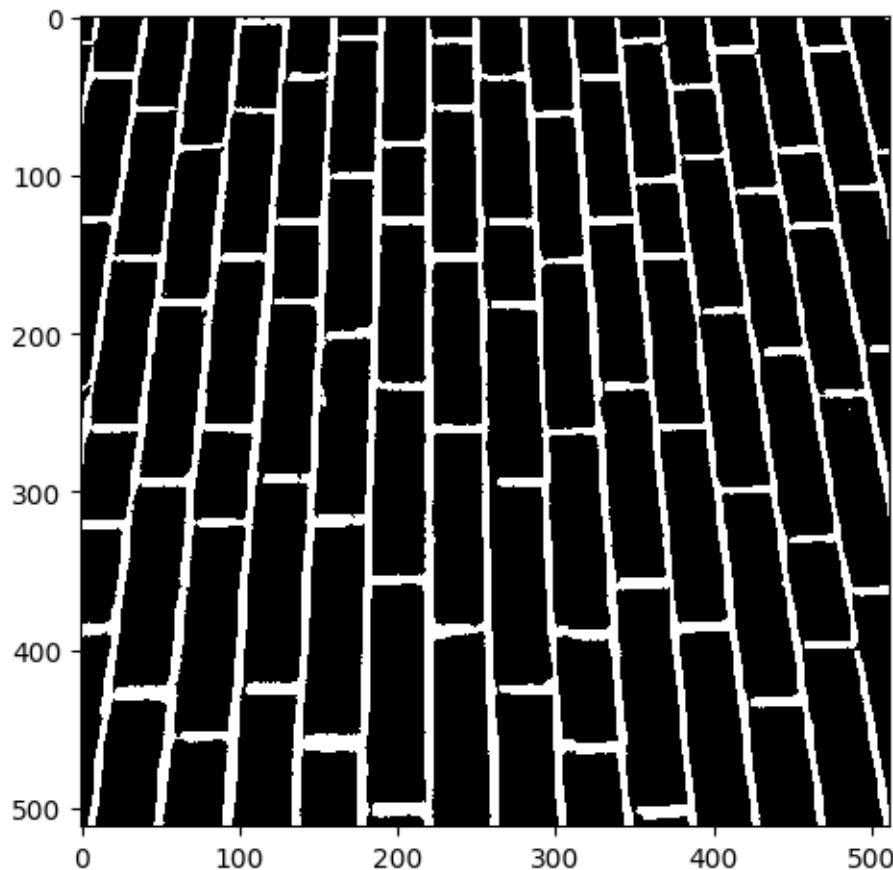
```
[225]: dynamic_range = np.max(brick) - np.min(brick)  
  
brickmsk = brick > (np.min(brick) + dynamic_range / 2)
```

```
image_characteristics(brickmsk, "White Brick")
io.imshow(brickmsk)
```

Usando White Brick:

Tamaño: (512, 512)
Valor Maximo: True
Valor Minimo: False
Tipo: bool

[225]: <matplotlib.image.AxesImage at 0x7cd40809e660>



c) Modifique los valores de una copia de astronaut para tomen un valor [r,g,b] aleatorio en las posiciones indicadas por la mascara brickmsk

```
[226]: astro_rgb = astro.copy()
# Generamos un color aleatorio
color = list(np.random.choice(range(256), size=3))
# En los pixeles de astro_rgb donde brickmsk sea True, se añade el color
astro_rgb[brickmsk] = color
```

```

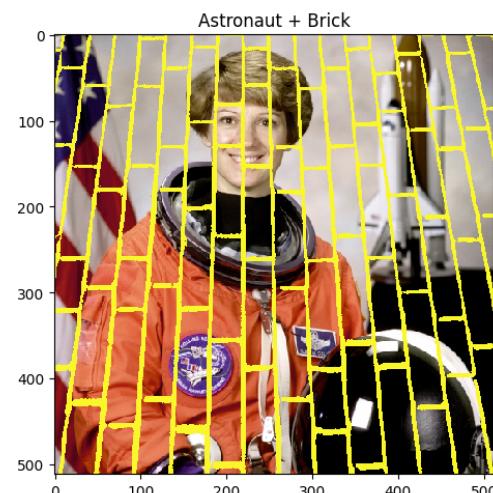
# Visualizamos las imágenes
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
io.imshow(astro)
plt.title("Astronaut Original")

plt.subplot(1, 2, 2)
io.imshow(astro_rgb)
plt.title("Astronaut + Brick")

# Mostrar las imágenes
plt.show()

```



- d) Usando una copia de astronaut, visualice, en cuatro subfiguras, cada uno de sus canales RGB, en escala de grises, junto con la imagen en color

```

[227]: astro_channels = astro.copy()

red = astro_channels[..., 0]      # Red
green = astro_channels[..., 1]     # Green
blue = astro_channels[..., 2]      # Blue
gray = rgb2gray(astro_channels)

# Visualizamos las imágenes
plt.figure(figsize=(15, 5))

plt.subplot(1, 5, 1)
io.imshow(red)

```

```

plt.title("Chanel Red")

plt.subplot(1, 5, 2)
io.imshow(green)
plt.title("Chanel Green")

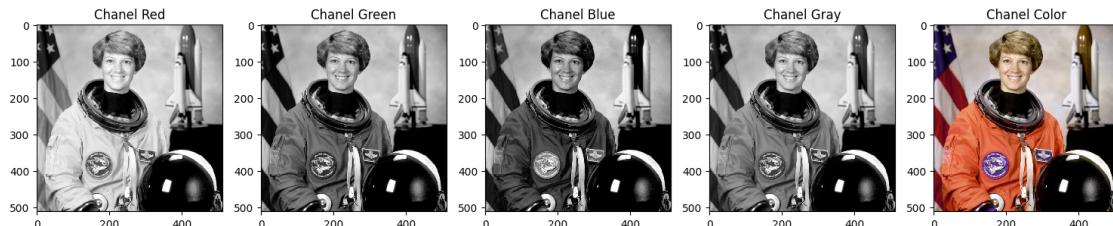
plt.subplot(1, 5, 3)
io.imshow(blue)
plt.title("Chanel Blue")

plt.subplot(1, 5, 4)
io.imshow(gray)
plt.title("Chanel Gray")

plt.subplot(1, 5, 5)
io.imshow(astro)
plt.title("Chanel Color")

# Mostrar las imágenes
plt.show()

```



- e) Usando la función `skimage.draw.disk`, obtenga las coordenadas de un círculo en una posición aleatoria del espacio 512×512 , y radio 20. Modifique la imagen resultante del apartado (c) para que los valores del canal verde en las posiciones del círculo sean 100 % del rango dinámico, dejando el resto de valores sin cambiar. Visualice el resultado

```

[228]: astro_circle = astro_rgb.copy()
# Cojemos el rango dinámico del canal verde
dynamic_range = np.max(astro_rgb[..., 1]) - np.min(astro_rgb[..., 1])

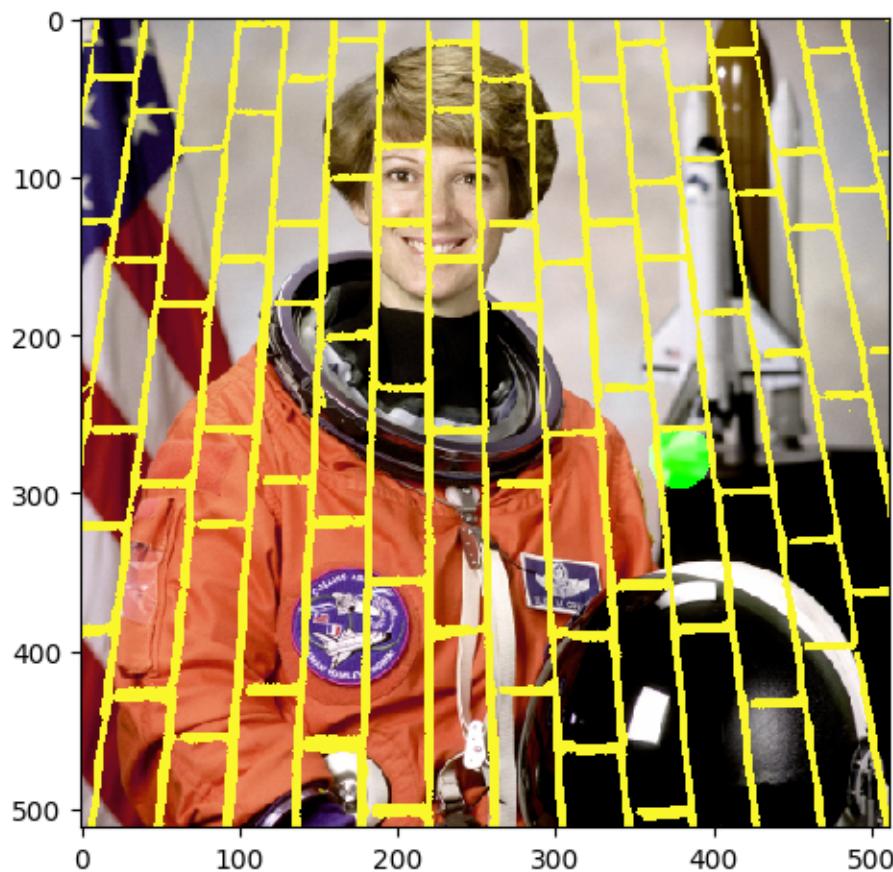
position = (random.randint(20,492), random.randint(20,492)) # Quitamos 20
  unidades para que el círculo no salga de la imagen
rr,cc = disk(position, 20)

astro_circle[rr,cc, 1] = dynamic_range

io.imshow(astro_circle)

```

```
astro_circle = None
```



5 Ejercicio 2.1.- Espacios de color

- a) Muestre en una cuadricula de 3×3 de subfiguras: los tres canales de RGB en escala de grises, los tres canales de CIE-Lab* en escala de grises, y los tres canales de HSV en escala de grises. Use los titulos de las figuras para indicar el nombre del canal y su rango de valores

```
[229]: def show_figures(size, row:int, col:int, images):
    plt.figure(figsize=size)
    for idx in range(1,col + 1):
        plt.subplot(row, col, idx)
        io.imshow(images[idx])
        plt.title(f"Image {images[idx]}")
    plt.show()
```

```
[230]: retina = data.retina()
butterfly = io.imread('butterfly.jpg')

# =====RETINA=====

retina_r = retina[..., 0]
retina_g = retina[..., 1]
retina_b = retina[..., 2]

retina_lab = rgb2lab(retina)
retina_l = retina_lab[..., 0]
retina_a = retina_lab[..., 1]
retina_B = retina_lab[..., 2]

retina_hsv = rgb2hsv(retina)
retina_h = retina_hsv[..., 0]
retina_s = retina_hsv[..., 1]
retina_v = retina_hsv[..., 2]

# =====BUTTERFLY=====

butterfly_r = butterfly[..., 0]
butterfly_g = butterfly[..., 1]
butterfly_b = butterfly[..., 2]

butterfly_lab = rgb2lab(butterfly)
butterfly_l = butterfly_lab[..., 0]
butterfly_a = butterfly_lab[..., 1]
butterfly_B = butterfly_lab[..., 2]

butterfly_hsv = rgb2hsv(butterfly)
butterfly_h = butterfly_hsv[..., 0]
butterfly_s = butterfly_hsv[..., 1]
butterfly_v = butterfly_hsv[..., 2]
```

```
[231]: plt.figure(figsize=(15, 15))

# First row
plt.subplot(3, 3, 1)
io.imshow(retina_r)
plt.title("Canal Red - [0,255]")

plt.subplot(3, 3, 2)
io.imshow(retina_g)
plt.title("Canal Green - [0,255]")

plt.subplot(3, 3, 3)
```

```

io.imshow(retina_b)
plt.title("Canal Blue - [0,255]")

# Second row
plt.subplot(3, 3, 4)
io.imshow(retina_l, cmap='gray')
plt.title("Canal L* - [0, 100]")

plt.subplot(3, 3, 5)
io.imshow(retina_a, cmap='gray')
plt.title("Canal a* - [-128, 127]")

plt.subplot(3, 3, 6)
io.imshow(retina_B, cmap='gray')
plt.title("Canal B - [-128, 127]")

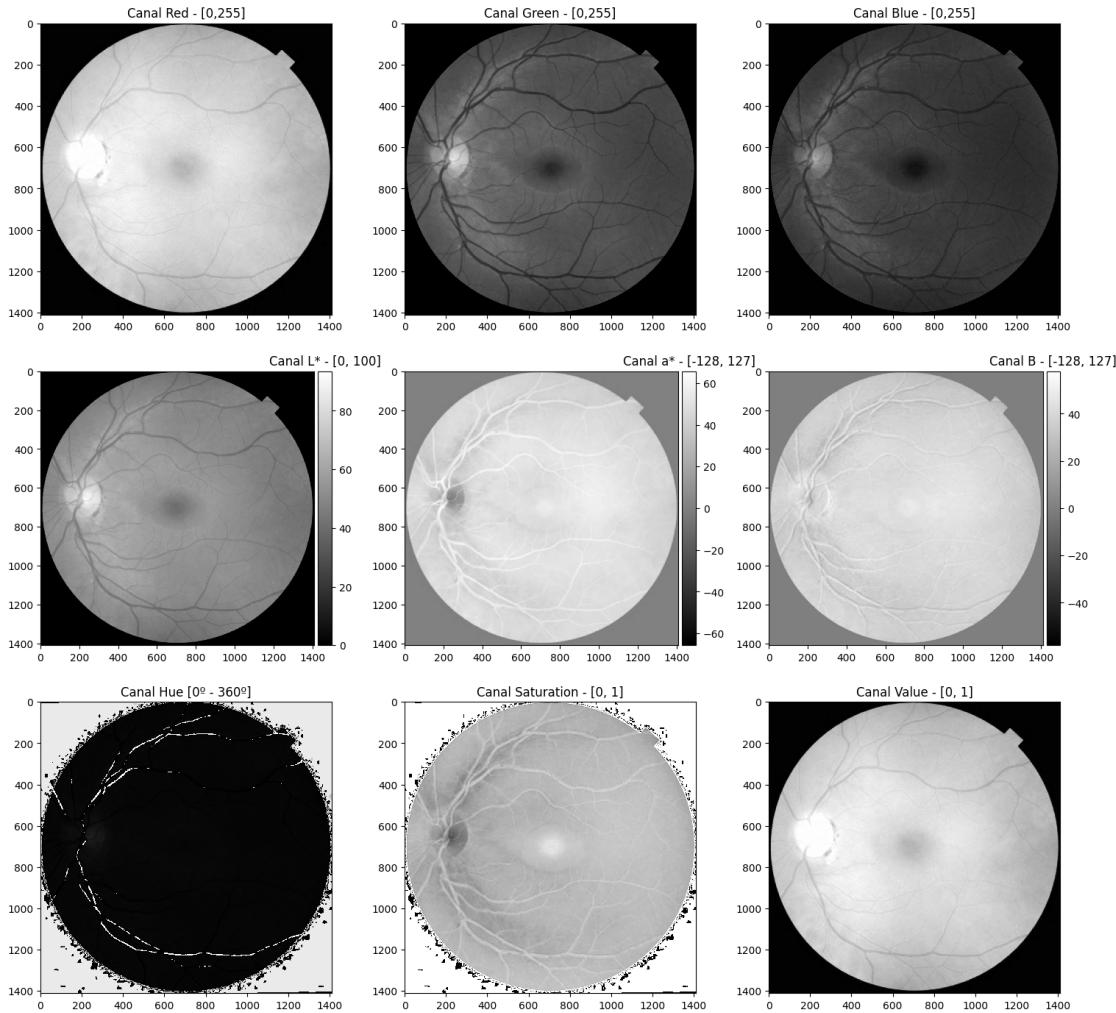
# Third row
plt.subplot(3, 3, 7)
io.imshow(retina_h, cmap='gray')
plt.title("Canal Hue [0° - 360°]")

plt.subplot(3, 3, 8)
io.imshow(retina_s, cmap='gray')
plt.title("Canal Saturation - [0, 1]")

plt.subplot(3, 3, 9)
io.imshow(retina_v, cmap='gray')
plt.title("Canal Value - [0, 1]")

plt.tight_layout()
plt.show()

```



```
[232]: plt.figure(figsize=(15, 15))

# First row
plt.subplot(3, 3, 1)
io.imshow(butterfly_r)
plt.title("Canal Red - [0,255]")

plt.subplot(3, 3, 2)
io.imshow(butterfly_g)
plt.title("Canal Green - [0,255]")

plt.subplot(3, 3, 3)
io.imshow(butterfly_b)
plt.title("Canal Blue - [0,255]")

# Second row
```

```

plt.subplot(3, 3, 4)
io.imshow(butterfly_l, cmap='gray')
plt.title("Canal L* - [0, 100]")

plt.subplot(3, 3, 5)
io.imshow(butterfly_a, cmap='gray')
plt.title("Canal a* - [-128, 127]")

plt.subplot(3, 3, 6)
io.imshow(butterfly_B, cmap='gray')
plt.title("Canal B - [-128, 127]")

# Third row
plt.subplot(3, 3, 7)
io.imshow(butterfly_h, cmap='gray')
plt.title("Canal Hue [0° - 360°]")

plt.subplot(3, 3, 8)
io.imshow(butterfly_s, cmap='gray')
plt.title("Canal Saturation - [0, 1]")

plt.subplot(3, 3, 9)
io.imshow(butterfly_v, cmap='gray')
plt.title("Canal Value - [0, 1]")

plt.tight_layout()
plt.show()

```



b) Observando los resultados: ¿Que canal es mas apropiado para la segmentacion objetivo? Razone la respuesta

Razone la respuesta El canal **G** (Green del RGB) es donde se aprecia una mejor segmentación en ambas imágenes, especialmente en la retina. Aunque en la imagen de “butterfly” existen otros candidatos con buenos resultados como **Value** o **L***, en la imagen de “retina” el canal verde destaca al ofrecer una mayor separación entre los vasos sanguíneos y el fondo, además de proporcionar un mayor nivel de detalle.

c) Compare diferentes canales de gris calculado a partir de las siguientes opciones: media de RGB, V de HSV, L* de CIE-Lab*.

```
[233]: mean_rgb = np.mean((retina_r + retina_b + retina_g) / 3)
mean_hsv = np.mean(retina_v)
mean_lab = np.mean(retina_l)

mean_rgb2 = np.mean((butterfly_r + butterfly_b + butterfly_g) / 3)
mean_hsv2 = np.mean(butterfly_v)
mean_lab2 = np.mean(butterfly_l)

print("RGB - Retina: ", mean_rgb)
```

```

print("HSV - Retina: ", mean_hsv)
print("Lab* - Retina: ", mean_lab)

print("\nRGB - Butterfly: ", mean_rgb2)
print("HSV - Butterfly: ", mean_hsv2)
print("Lab* - Butterfly: ", mean_lab2)

```

RGB - Retina: 24.47120436555074
HSV - Retina: 0.6252932087170494
Lab* - Retina: 39.64988000504808

RGB - Butterfly: 38.517530065106236
HSV - Butterfly: 0.49883625214335797
Lab* - Butterfly: 48.238734396917

d) Realizar la misma comparacion pero teniendo en cuenta la saturacion de HSV y la saturacion calculada a partir de Lab*

```
[234]: mean_hsv = np.mean(retina_v + retina_s)
mean_lab = np.mean(retina_l + retina_a + retina_B)

mean_hsv2 = np.mean(butterfly_v + butterfly_s)
mean_lab2 = np.mean(butterfly_l + butterfly_a + butterfly_B)

print("RGB - Retina: ", mean_rgb)
print("HSV - Retina: ", mean_hsv)
print("Lab* - Retina: ", mean_lab)

print("\nRGB - Butterfly: ", mean_rgb2)
print("HSV - Butterfly: ", mean_hsv2)
print("Lab* - Butterfly: ", mean_lab2)
```

RGB - Retina: 24.47120436555074
HSV - Retina: 1.3841554440454389
Lab* - Retina: 105.88901614144682

RGB - Butterfly: 38.517530065106236
HSV - Butterfly: 0.9064491577345284
Lab* - Butterfly: 56.935899137037175

e) Calcule los canales oponentes Red-Green y Blue-Yellow, calculados a partir de RGB o usando las componentes a* y b* de CIE-Lab*

```
[235]: red_green = retina_r - retina_g # Diferencia entre los canales
       ↪rojo y verde
blue_yellow = retina_b - (retina_r + retina_g) # Diferencia entre azul y la
       ↪combinación de rojo y verde
```

```

red_green2 = butterfly_r - butterfly_g          # Diferencia entreo
    ↵ los canales rojo y verde
blue_yellow2 = butterfly_b - (butterfly_r + butterfly_g)  # Diferencia entreo
    ↵ azul y la combinación de rojo y verde

plt.figure(figsize=(15, 15))

plt.subplot(2, 2, 1)
io.imshow(red_green)
plt.title("Red-Green (RGB)")

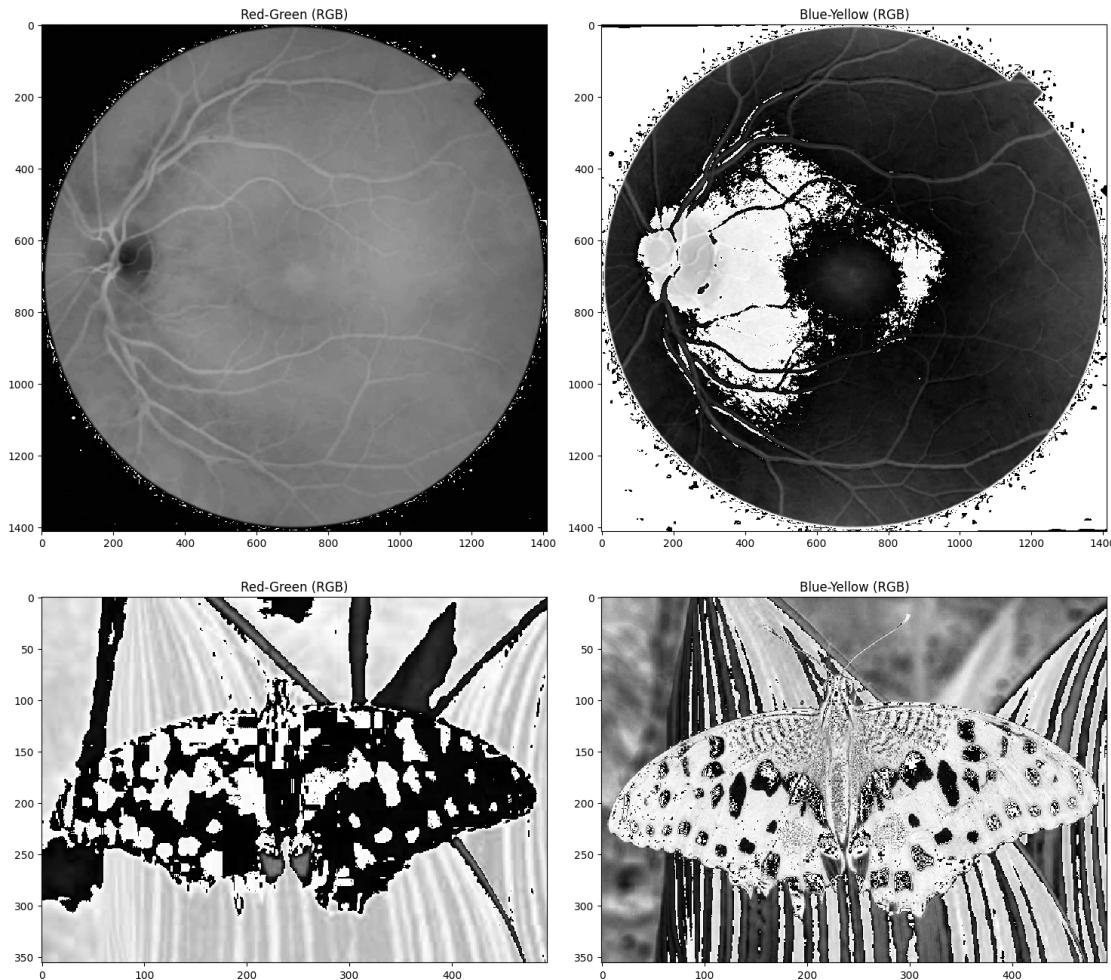
plt.subplot(2, 2, 2)
io.imshow(blue_yellow)
plt.title("Blue-Yellow (RGB)")

plt.subplot(2, 2, 3)
io.imshow(red_green2)
plt.title("Red-Green (RGB)")

plt.subplot(2, 2, 4)
io.imshow(blue_yellow2)
plt.title("Blue-Yellow (RGB)")

plt.tight_layout()
plt.show()

```



6 Ejercicio 3.1.- Representación de histogramas

- a) Calcule el histograma de brillo de las imágenes en escala de grises

```
[236]: img = data.camera()

histo, bins = np.histogram(img, bins=256, range=(0, 255))

plt.figure(figsize=(20, 5))
plt.subplot(1,2,1)
io.imshow(img)
plt.title("Camera GreyScale")

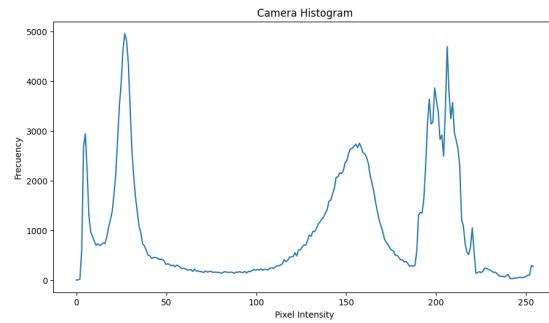
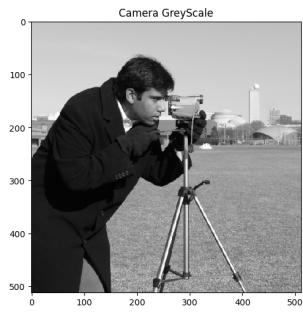
plt.subplot(1, 2, 2)
plt.plot(bins[:-1], histo)      # Cogemos hasta el ultimo punto para poder
                                ↵hacer el histograma
```

```

plt.title("Camera Histogram")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")

plt.subplots_adjust(wspace=0.4)
plt.show()

```



b) Represente el histograma con diferentes numeros de bins: 10 y 100

```

[237]: histo10, bins10 = np.histogram(img, bins=10, range=(0, 255))
histo100, bins100 = np.histogram(img, bins=100, range=(0, 255))

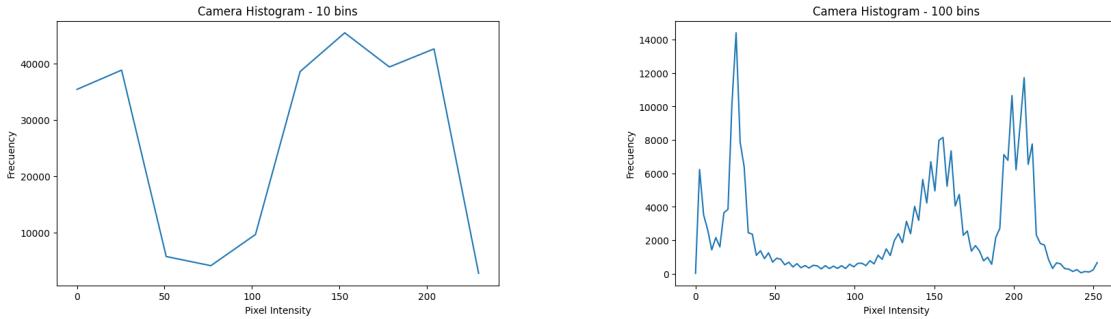
plt.figure(figsize=(20, 5))

plt.subplot(1, 2, 1)
plt.plot(bins10[:-1], histo10)
plt.title("Camera Histogram - 10 bins")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")

plt.subplot(1, 2, 2)
plt.plot(bins100[:-1], histo100)
plt.title("Camera Histogram - 100 bins")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")

plt.subplots_adjust(wspace=0.4)
plt.show()

```



c) Haga lo mismo para diferentes ventanas de la imagen. Represente visualmente la imagen general, cuadrados de subventanas y sus histogramas asociados. Seleccione regiones de alto contraste, de bajo contraste, y zonas irregulares de mas o menos brillo

```
[238]: img1 = img[100:150, 150:300]
img2 = img[200:300, 0:100]
img3 = img[100:200, 175:250]
img4 = img[150:300, 400:500]
img5 = img[200:300, 200:400]
img6 = img[450:, 200:400]

histo1, bins1 = np.histogram(img1, bins=256, range=(0, 255))
histo2, bins2 = np.histogram(img2, bins=256, range=(0, 255))
histo3, bins3 = np.histogram(img3, bins=256, range=(0, 255))
histo4, bins4 = np.histogram(img4, bins=256, range=(0, 255))
histo5, bins5 = np.histogram(img5, bins=256, range=(0, 255))
histo6, bins6 = np.histogram(img6, bins=256, range=(0, 255))

plt.figure(figsize=(30, 15))

plt.subplot(2, 3, 1)
io.imshow(img1)
plt.title("Image 1")

plt.subplot(2, 3, 2)
io.imshow(img2)
plt.title("Image 2")

plt.subplot(2, 3, 3)
io.imshow(img3)
plt.title("Image 3")

plt.subplot(2, 3, 4)
io.imshow(img4)
plt.title("Image 4")
```

```

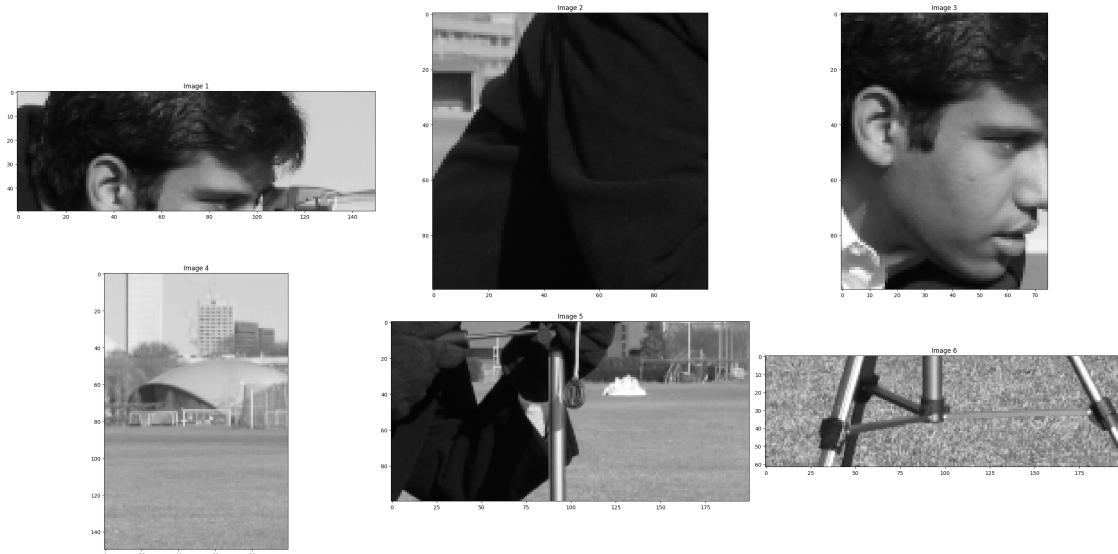
plt.subplot(2, 3, 5)
io.imshow(img5)
plt.title("Image 5")

plt.subplot(2, 3, 6)
io.imshow(img6)
plt.title("Image 6")

plt.show

```

[238]: <function matplotlib.pyplot.show(close=None, block=None)>



[239]: plt.figure(figsize=(25, 10))

```

plt.subplot(2, 3, 1)
plt.plot(bins1[:-1], histo1)
plt.title("Histograma de img1")
plt.xlabel("Intensidad de píxel")
plt.ylabel("Frecuencia")

plt.subplot(2, 3, 2)
plt.plot(bins2[:-1], histo2)
plt.title("Histograma de img2")
plt.xlabel("Intensidad de píxel")
plt.ylabel("Frecuencia")

plt.subplot(2, 3, 3)

```

```

plt.plot(bins3[:-1], histo3)
plt.title("Histograma de img3")
plt.xlabel("Intensidad de píxel")
plt.ylabel("Frecuencia")

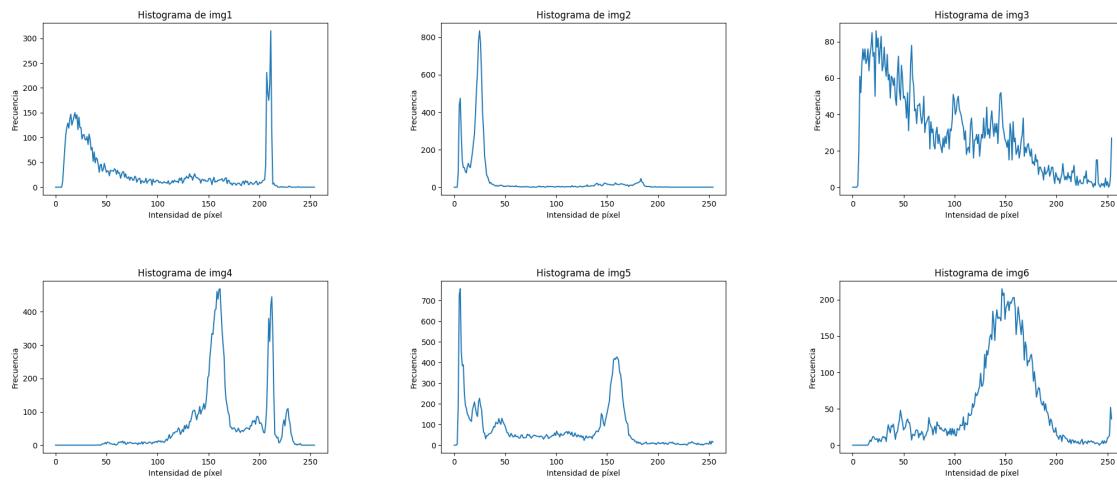
plt.subplot(2, 3, 4)
plt.plot(bins4[:-1], histo4)
plt.title("Histograma de img4")
plt.xlabel("Intensidad de píxel")
plt.ylabel("Frecuencia")

plt.subplot(2, 3, 5)
plt.plot(bins5[:-1], histo5)
plt.title("Histograma de img5")
plt.xlabel("Intensidad de píxel")
plt.ylabel("Frecuencia")

plt.subplot(2, 3, 6)
plt.plot(bins6[:-1], histo6)
plt.title("Histograma de img6")
plt.xlabel("Intensidad de píxel")
plt.ylabel("Frecuencia")

plt.subplots_adjust(hspace=0.5, wspace=0.4)
plt.show()

```



d) Repita el ejercicio anterior sumando una constante positiva y negativa: 0,25 y -0,25. Analice de nuevo los histogramas resultantes y sus cambios

[240]:

```

img1_pos = np.clip(img1 + 0.25, 0, 255)
img1_neg = np.clip(img1 - 0.25, 0, 255)

```

```

img2_pos = np.clip(img2 + 0.25, 0, 255)
img2_neg = np.clip(img2 - 0.25, 0, 1)

img3_pos = np.clip(img3 + 0.25, 0, 255)
img3_neg = np.clip(img3 - 0.25, 0, 255)

img4_pos = np.clip(img4 + 0.25, 0, 255)
img4_neg = np.clip(img4 - 0.25, 0, 255)

img5_pos = np.clip(img5 + 0.25, 0, 255)
img5_neg = np.clip(img5 - 0.25, 0, 255)

img6_pos = np.clip(img6 + 0.25, 0, 255)
img6_neg = np.clip(img6 - 0.25, 0, 255)

# Calcular los histogramas usando np.histogram (256 bins)
histo1_pos, bins1_pos = np.histogram(img1_pos, bins=256, range=(0, 255))
histo1_neg, bins1_neg = np.histogram(img1_neg, bins=256, range=(0, 255))

histo2_pos, bins2_pos = np.histogram(img2_pos, bins=256, range=(0, 255))
histo2_neg, bins2_neg = np.histogram(img2_neg, bins=256, range=(0, 255))

histo3_pos, bins3_pos = np.histogram(img3_pos, bins=256, range=(0, 255))
histo3_neg, bins3_neg = np.histogram(img3_neg, bins=256, range=(0, 255))

histo4_pos, bins4_pos = np.histogram(img4_pos, bins=256, range=(0, 255))
histo4_neg, bins4_neg = np.histogram(img4_neg, bins=256, range=(0, 255))

histo5_pos, bins5_pos = np.histogram(img5_pos, bins=256, range=(0, 255))
histo5_neg, bins5_neg = np.histogram(img5_neg, bins=256, range=(0, 255))

histo6_pos, bins6_pos = np.histogram(img6_pos, bins=256, range=(0, 255))
histo6_neg, bins6_neg = np.histogram(img6_neg, bins=256, range=(0, 255))

```

```

[241]: plt.figure(figsize=(30, 30))

# ====== IMG1 ======

plt.subplot(6, 3, 1) # Imagen original 1
plt.imshow(img1, cmap='gray')
plt.title("Image 1 - Original")
plt.axis('off')

plt.subplot(6, 3, 2) # Histograma positivo 1
plt.plot(bins1_pos[:-1], histo1_pos)
plt.title("Image 1 - Pos")

```

```

plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

plt.subplot(6, 3, 3) # Histograma negativo 1
plt.plot(bins1_neg[:-1], histo1_neg)
plt.title("Image 1 - Neg")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

# ====== IMG2 =====

plt.subplot(6, 3, 4) # Imagen original 2
plt.imshow(img2, cmap='gray')
plt.title("Image 2 - Original")
plt.axis('off')

plt.subplot(6, 3, 5) # Histograma positivo 2
plt.plot(bins2_pos[:-1], histo2_pos)
plt.title("Image 2 - Pos")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

plt.subplot(6, 3, 6) # Histograma negativo 2
plt.plot(bins2_neg[:-1], histo2_neg)
plt.title("Image 2 - Neg")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

# ====== IMG3 =====

plt.subplot(6, 3, 7) # Imagen original 3
plt.imshow(img3, cmap='gray')
plt.title("Image 3 - Original")
plt.axis('off')

plt.subplot(6, 3, 8) # Histograma positivo 3
plt.plot(bins3_pos[:-1], histo3_pos)
plt.title("Image 3 - Pos")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

plt.subplot(6, 3, 9) # Histograma negativo 3
plt.plot(bins3_neg[:-1], histo3_neg)
plt.title("Image 3 - Neg")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

```

```

# ====== IMG4 =====

plt.subplot(6, 3, 10) # Imagen original 4
plt.imshow(img4, cmap='gray')
plt.title("Image 4 - Original")
plt.axis('off')

plt.subplot(6, 3, 11) # Histograma positivo 4
plt.plot(bins4_pos[:-1], histo4_pos)
plt.title("Image 4 - Pos")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

plt.subplot(6, 3, 12) # Histograma negativo 4
plt.plot(bins4_neg[:-1], histo4_neg)
plt.title("Image 4 - Neg")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

# ====== IMG5 =====

plt.subplot(6, 3, 13) # Imagen original 5
plt.imshow(img5, cmap='gray')
plt.title("Image 5 - Original")
plt.axis('off')

plt.subplot(6, 3, 14) # Histograma positivo 5
plt.plot(bins5_pos[:-1], histo5_pos)
plt.title("Image 5 - Pos")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

plt.subplot(6, 3, 15) # Histograma negativo 5
plt.plot(bins5_neg[:-1], histo5_neg)
plt.title("Image 5 - Neg")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

# ====== IMG6 =====

plt.subplot(6, 3, 16) # Imagen original 6
plt.imshow(img6, cmap='gray')
plt.title("Image 6 - Original")
plt.axis('off')

plt.subplot(6, 3, 17) # Histograma positivo 6
plt.plot(bins6_pos[:-1], histo6_pos)

```

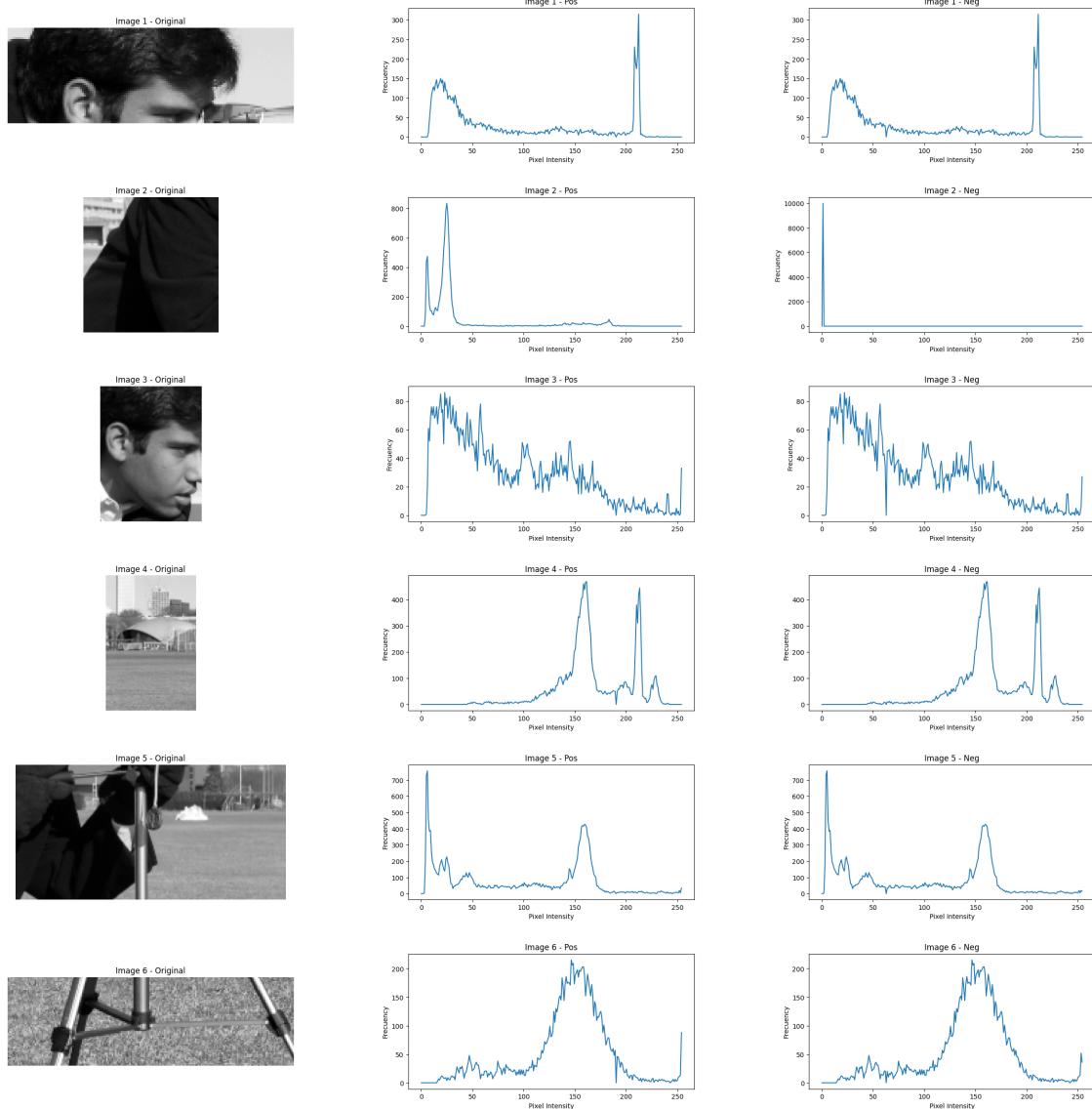
```

plt.title("Image 6 - Pos")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")

plt.subplot(6, 3, 18) # Histograma negativo 6
plt.plot(bins6_neg[:-1], histo6_neg)
plt.title("Image 6 - Neg")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")

plt.subplots_adjust(wspace=0.4, hspace=0.4)
plt.show()

```



ANALISIS Los histogramas no muestran grandes diferencias ya que las modificaciones realizadas son pequeñas respecto al rango (± 0.25). Sin embargo, es importante destacar que en algunas imágenes se observan secciones donde los píxeles “desaparecen” al modificar los rangos de intensidad, especialmente cuando los valores originales están cerca de los límites del rango.

El caso donde mas se aprecia es en “**Image 2 - Neg**”, donde casi todos los valores de intensidad se pierden. Esto ocurre porque muchos de los píxeles en la imagen original ya estaban cercanos al límite inferior (0), y al restar o sumar 0.25, los valores caen fuera del rango permitido (*valores negativos respecto a los 8 bits*), lo que provoca una acumulación de píxeles. Esta pérdida de información es más evidente cuando los valores originales están concentrados cerca de los límites.

e) Repita el ejercicio anterior multiplicando una constante mayor y menos que 1: 0,5 y 1.5. Analice de nuevo los histogramas resultantes y sus cambios

```
[242]: img1_less = np.clip(img1 * 0.5, 0, 255)
img1_more = np.clip(img1 * 1.5, 0, 255)

img2_less = np.clip(img2 * 0.5, 0, 255)
img2_more = np.clip(img2 * 1.5, 0, 255)

img3_less = np.clip(img3 * 0.5, 0, 255)
img3_more = np.clip(img3 * 1.5, 0, 255)

img4_less = np.clip(img4 * 0.5, 0, 255)
img4_more = np.clip(img4 * 1.5, 0, 255)

img5_less = np.clip(img5 * 0.5, 0, 255)
img5_more = np.clip(img5 * 1.5, 0, 255)

img6_less = np.clip(img6 * 0.5, 0, 255)
img6_more = np.clip(img6 * 1.5, 0, 255)

histo1_less, bins1_less = np.histogram(img1_less, bins=256, range=(0, 255))
histo1_more, bins1_more = np.histogram(img1_more, bins=256, range=(0, 255))

histo2_less, bins2_less = np.histogram(img2_less, bins=256, range=(0, 255))
histo2_more, bins2_more = np.histogram(img2_more, bins=256, range=(0, 255))

histo3_less, bins3_less = np.histogram(img3_less, bins=256, range=(0, 255))
histo3_more, bins3_more = np.histogram(img3_more, bins=256, range=(0, 255))

histo4_less, bins4_less = np.histogram(img4_less, bins=256, range=(0, 255))
histo4_more, bins4_more = np.histogram(img4_more, bins=256, range=(0, 255))

histo5_less, bins5_less = np.histogram(img5_less, bins=256, range=(0, 255))
histo5_more, bins5_more = np.histogram(img5_more, bins=256, range=(0, 255))
```

```
histo6_less, bins6_less = np.histogram(img6_less, bins=256, range=(0, 255))
histo6_more, bins6_more = np.histogram(img6_more, bins=256, range=(0, 255))
```

```
[243]: plt.figure(figsize=(30, 30))

# ====== IMG1 =====

plt.subplot(6, 3, 1) # Imagen original 1
plt.imshow(img1, cmap='gray')
plt.title("Image 1 - Original")
plt.axis('off')

plt.subplot(6, 3, 2) # Histograma img1 con menos contraste
plt.plot(bins1_less[:-1], histo1_less)
plt.title("Image 1 - Less Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

plt.subplot(6, 3, 3) # Histograma img1 con más contraste
plt.plot(bins1_more[:-1], histo1_more)
plt.title("Image 1 - More Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

# ====== IMG2 =====

plt.subplot(6, 3, 4) # Imagen original 2
plt.imshow(img2, cmap='gray')
plt.title("Image 2 - Original")
plt.axis('off')

plt.subplot(6, 3, 5) # Histograma img2 con menos contraste
plt.plot(bins2_less[:-1], histo2_less)
plt.title("Image 2 - Less Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

plt.subplot(6, 3, 6) # Histograma img2 con más contraste
plt.plot(bins2_more[:-1], histo2_more)
plt.title("Image 2 - More Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

# ====== IMG3 =====

plt.subplot(6, 3, 7) # Imagen original 3
plt.imshow(img3, cmap='gray')
```

```

plt.title("Image 3 - Original")
plt.axis('off')

plt.subplot(6, 3, 8) # Histograma img3 con menos contraste
plt.plot(bins3_less[:-1], histo3_less)
plt.title("Image 3 - Less Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

plt.subplot(6, 3, 9) # Histograma img3 con más contraste
plt.plot(bins3_more[:-1], histo3_more)
plt.title("Image 3 - More Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

# ====== IMG4 =====

plt.subplot(6, 3, 10) # Imagen original 4
plt.imshow(img4, cmap='gray')
plt.title("Image 4 - Original")
plt.axis('off')

plt.subplot(6, 3, 11) # Histograma img4 con menos contraste
plt.plot(bins4_less[:-1], histo4_less)
plt.title("Image 4 - Less Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

plt.subplot(6, 3, 12) # Histograma img4 con más contraste
plt.plot(bins4_more[:-1], histo4_more)
plt.title("Image 4 - More Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

# ====== IMG5 =====

plt.subplot(6, 3, 13) # Imagen original 5
plt.imshow(img5, cmap='gray')
plt.title("Image 5 - Original")
plt.axis('off')

plt.subplot(6, 3, 14) # Histograma img5 con menos contraste
plt.plot(bins5_less[:-1], histo5_less)
plt.title("Image 5 - Less Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

```

```

plt.subplot(6, 3, 15) # Histograma img5 con más contraste
plt.plot(bins5_more[:-1], histo5_more)
plt.title("Image 5 - More Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

# ====== IMG6 =====

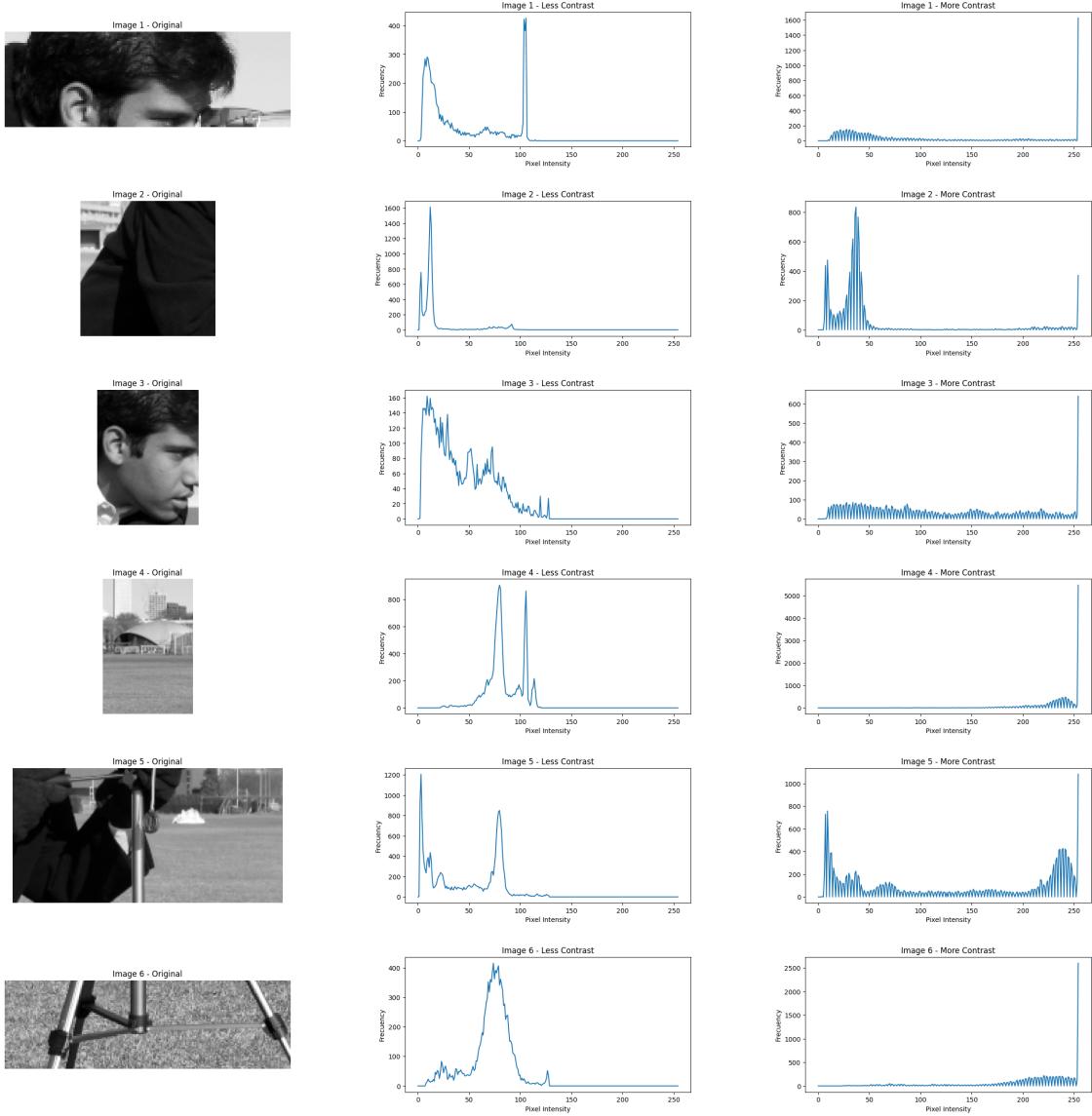
plt.subplot(6, 3, 16) # Imagen original 6
plt.imshow(img6, cmap='gray')
plt.title("Image 6 - Original")
plt.axis('off')

plt.subplot(6, 3, 17) # Histograma img6 con menos contraste
plt.plot(bins6_less[:-1], histo6_less)
plt.title("Image 6 - Less Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

plt.subplot(6, 3, 18) # Histograma img6 con más contraste
plt.plot(bins6_more[:-1], histo6_more)
plt.title("Image 6 - More Contrast")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frecuency")

plt.subplots_adjust(wspace=0.4, hspace=0.4)
plt.show()

```



ANALISIS Al multiplicar los valores de intensidad por 0.50, esencialmente estamos reduciendo la intensidad de los píxeles a la mitad. Esto es evidente en todos los histogramas de “**Less Contrast**”, ya que estos no presentan valores después de 127, provocando que la imagen se vea mas oscura al no tener pixeles claros.

En el caso de de multiplicar por 1.5, observamos un mayor numero de picos y valores en 0 producidos por el estiramiento de los valores. Esto ocurre porque los píxeles que originalmente estaban cerca en el espectro de intensidades ahora se separan, provocando una acumulación en zonas más específicas. A medida que los valores se estiran hacia los extremos, se aprecia una acumulación en el valor máximo de 255.

7 Ejercicio 4.1.- Transformada de Fourier

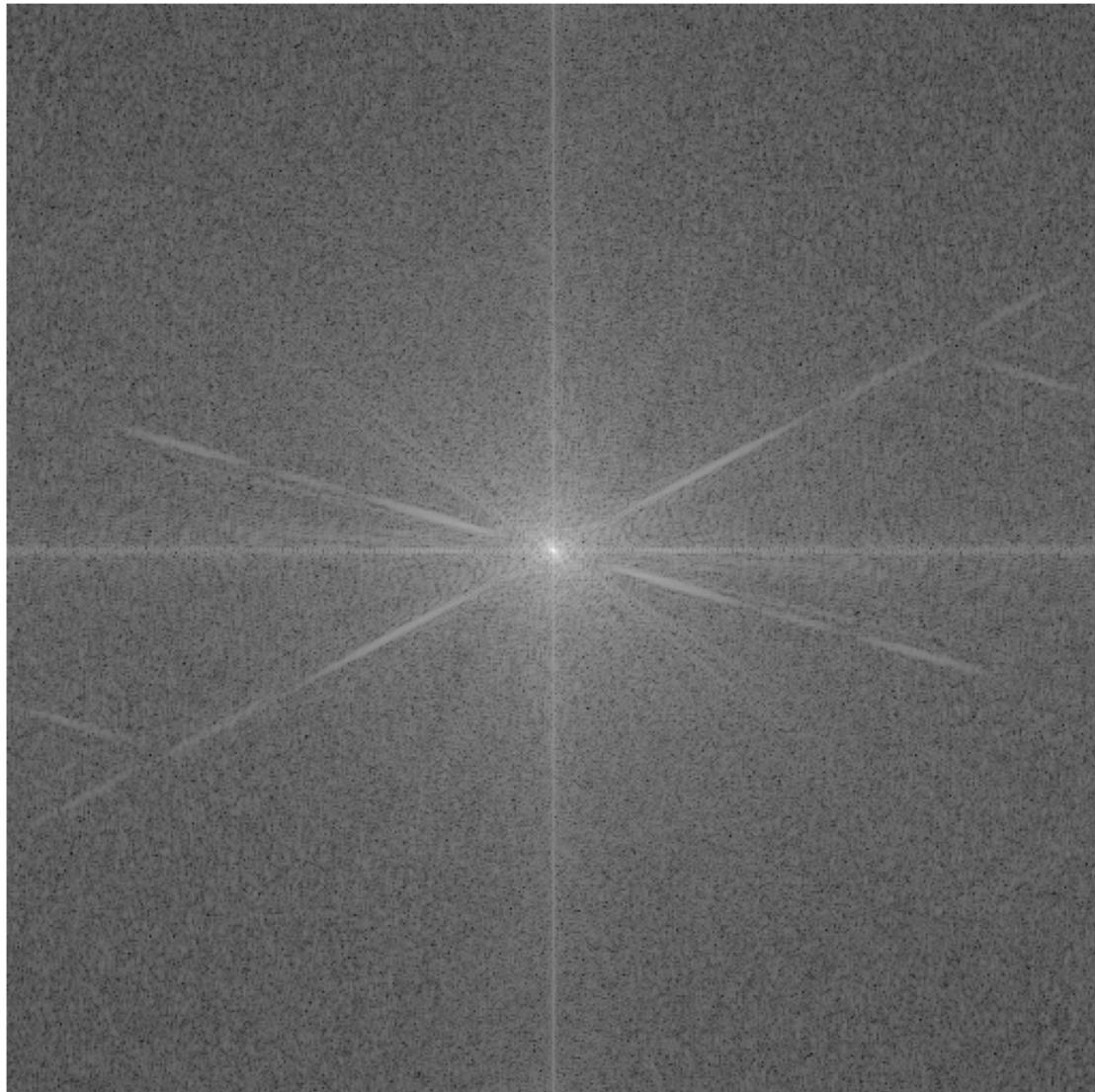
a) Utilice la funcion numpy.fft.fft2 para calcular la Transformada de Fourier de la imagen. Represente la magnitud de la transformada de Fourier de la imagen en escala logaritmica y aplique numpy.fft.fftshift para centrar la componente continua de frecuencia

```
[244]: img = data.camera()

fourier = np.fft.fft2(img)          # Transformacion
shifted = np.fft.fftshift(fourier)  # Centrar la componente
magnitude = np.abs(shifted)        # Maginitud
A = np.log(magnitude)

plt.figure(figsize=(15, 15))
plt.imshow(A, cmap='gray')
plt.title("Magnitud de la Transformada de Fourier (Escala Logarítmica)")
plt.axis('off')
plt.show()
```

Magnitud de la Transformada de Fourier (Escala Logarítmica)



- b) Repita el ejercicio anterior, tras multiplicar la imagen A por una ventana Gaussiana de una anchura adecuada, usando skimage.filters.window, para que se eliminen los componentes de frecuencia asociadas al borde de la imagen (imagen B)

```
[245]: wimage = img * window('gaussian', 200), img.shape)
wimage2 = img * window('gaussian', 20), img.shape)

fourier = np.fft.fft2(wimage)          # Transformacion
shifted = np.fft.fftshift(fourier)    # Centrar la componente
magnitude = np.abs(shifted)         # Magnitud
B = np.log(magnitude)
```

```

fourier = np.fft.fft2(wimage2)           # Transformacion
shifted = np.fft.fftshift(fourier)       # Centrar la componente
magnitude = np.abs(shifted)             # Magnitud
B2 = np.log(magnitude)

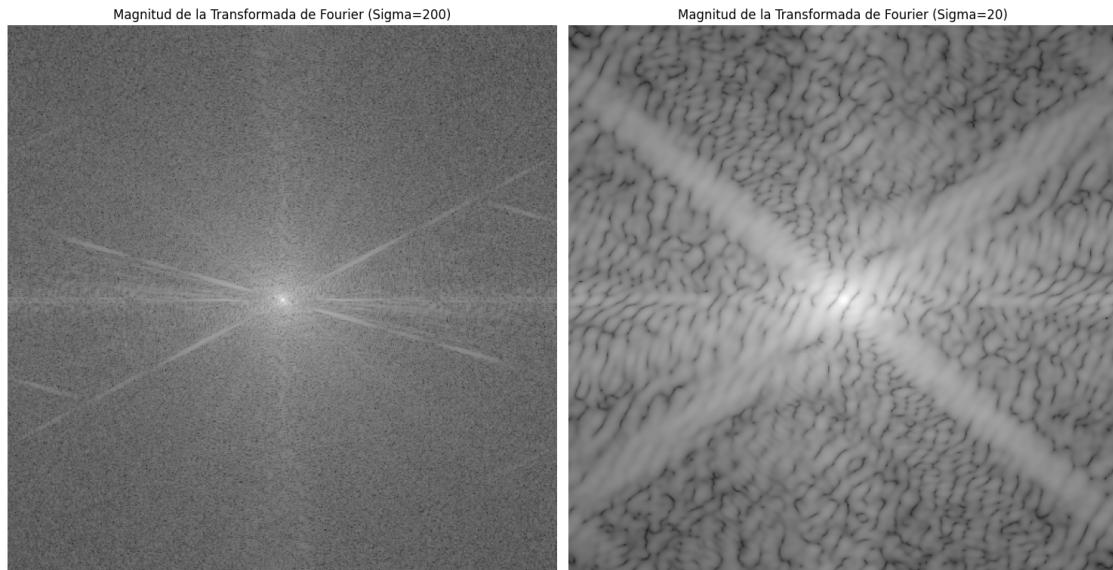
plt.figure(figsize=(15, 15))

# Primer subplot: Sigma=20
plt.subplot(1, 2, 1)
plt.imshow(B, cmap='gray')
plt.title("Magnitud de la Transformada de Fourier (Sigma=200)")
plt.axis('off')

# Segundo subplot: Sigma=150
plt.subplot(1, 2, 2)
plt.imshow(B2, cmap='gray')
plt.title("Magnitud de la Transformada de Fourier (Sigma=20)")
plt.axis('off')

plt.tight_layout()
plt.show()

```



EXPLICACIÓN DE LA SELECCIÓN DE SIGMA Usamos un valor de sigma alto debido al tamaño de la imagen (**512x512 píxeles**). Al utilizar valores de **sigma bajos**, como 10 o 20, la ventana actúa sobre un área demasiado pequeña en comparación con la imagen completa, provocando que la magnitud de la transformada de Fourier se vea dominada por las frecuencias altas, resultando en una visualización más “borrosa”.

Para mejorar la visualización de las frecuencias y obtener una mejor representación, tenemos que usar valores de sigma mas adecuados para el tamaño de la iamgen (*cerca del 50% del tamaño de la imagen en este caso*). Esto permite una suavización adecuada en el espacio de la imagen antes de la transformada.

c) Realice una convolucion de la imagen B con una Gaussiana de $\sigma = 10$ (imagen C) y represente el resultado, en dominio espacial y en frecuencia

```
[246]: conv = gaussian(B, sigma=10)
# Imagen convolucionada en el dominio espacial
plt.figure(figsize=(15, 7))
plt.subplot(1,2,1)
plt.imshow(conv, cmap='gray')
plt.title("Dominio Espacial Imagen C")
plt.axis('off')

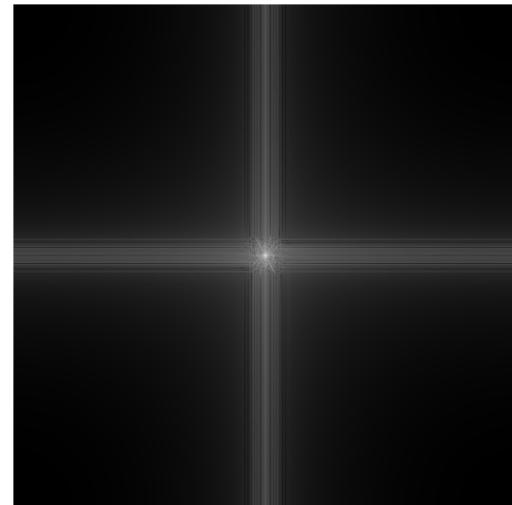
# Transformada de Fourier de la imagen convolucionada
fourier_C = np.fft.fft2(conv)
shifted_C = np.fft.fftshift(fourier_C)
magnitude_C = np.abs(shifted_C)
C = np.log(magnitude_C)

# Imagen convolucionada en el dominio de la frecuencia
plt.subplot(1,2,2)
plt.imshow(C, cmap='gray')
plt.title("Transformada de Fourier de la Imagen C (Convolución)")
plt.axis('off')
plt.show()
```

Dominio Espacial Imagen C



Transformada de Fourier de la Imagen C (Convolución)



d) Calcule la imagen resta $D = B - C$, y represente el resultado en espacial y frecuencia

```
[247]: # Obtener las dimensiones de C
height_C, width_C = C.shape

# Recortar la imagen B para que coincida con las dimensiones de C
B_cropped = B[:height_C, :width_C]

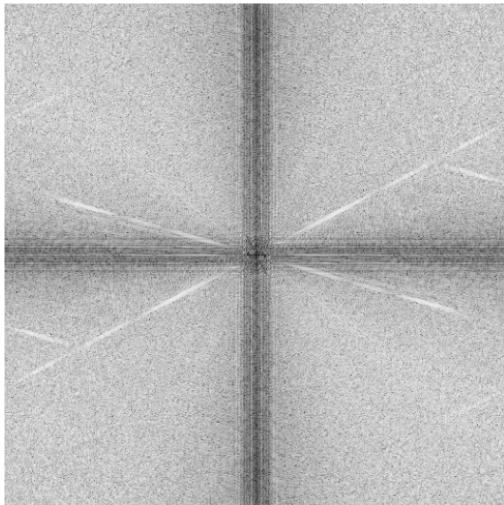
# Cálculo de la imagen D en el dominio espacial
D = B_cropped - C

# Imagen D en el dominio espacial
plt.figure(figsize=(15, 7))
plt.subplot(1,2,1)
plt.imshow(D, cmap='gray')
plt.title("Imagen D (B - C) en el dominio espacial")
plt.axis('off')

# Transformada de Fourier de la imagen D
fourier_D = np.fft.fft2(D)
shifted_D = np.fft.fftshift(fourier_D)
magnitude_D = np.abs(shifted_D)
D = np.log(magnitude_D)

plt.subplot(1,2,2)
plt.imshow(D, cmap='gray')
plt.title("Transformada de Fourier de la Imagen D (B - C)")
plt.axis('off')
plt.show()
```

Imagen D (B - C) en el dominio espacial



Transformada de Fourier de la Imagen D (B - C)

