

# TP2

May 6, 2025

## 1 Procesamiento de Imágenes

### 2 Trabajo Práctico N°2

#### 2.1 1. Histogramas

```
[1]: import cv2
from PIL import Image
import os
import numpy as np
import matplotlib.pyplot as plt
from skimage.exposure import match_histograms
from skimage import data
```

1. Calcular el histograma de una imagen en escala de grises y de cada canal de una imagen a color. Visualizar los histogramas usando Matplotlib

```
[3]: color_image_path = '../Images/Lenna.png'
bg_image_path = '../Images/Lenna_grey.png'

color_image = cv2.imread(color_image_path)
color_image_rgb = cv2.cvtColor(color_image, cv2.COLOR_BGR2RGB)

bg_image = cv2.imread(bg_image_path, cv2.IMREAD_GRAYSCALE)

hist_gray = cv2.calcHist([bg_image], [0], None, [256], [0, 256])
hist_r = cv2.calcHist([color_image], [2], None, [256], [0, 256])
hist_g = cv2.calcHist([color_image], [1], None, [256], [0, 256])
hist_b = cv2.calcHist([color_image], [0], None, [256], [0, 256])

plt.figure(figsize=(14, 10))

plt.subplot(3,2, 1)
plt.imshow(bg_image, cmap='gray')
plt.title('Imagen en escala de grises')
plt.axis('off')

plt.subplot(3,2, 2)
```

```

plt.imshow(color_image_rgb)
plt.title('Imagen en color')
plt.axis('off')

plt.subplot(3,2, 3)
plt.plot(hist_gray, color='black',)
plt.title('Histograma de imagen en escala de grises')
plt.xlabel('Intensidad de pixel')
plt.ylabel('Frecuencia')

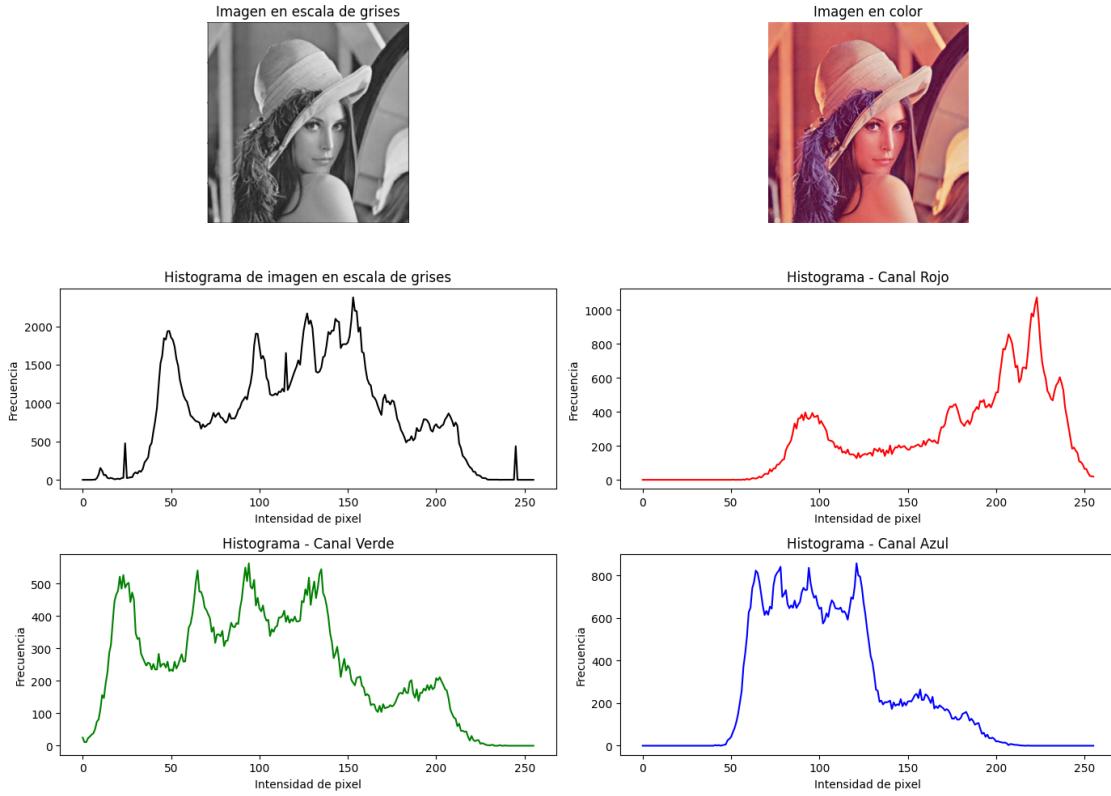
plt.subplot(3,2,4)
plt.plot(hist_r, color='red', label='Rojo')
plt.title('Histograma - Canal Rojo')
plt.xlabel('Intensidad de pixel')
plt.ylabel('Frecuencia')

plt.subplot(3,2,5)
plt.plot(hist_g, color='green', label='Verde')
plt.title('Histograma - Canal Verde')
plt.xlabel('Intensidad de pixel')
plt.ylabel('Frecuencia')

plt.subplot(3,2,6)
plt.plot(hist_b, color='blue', label='Azul')
plt.title('Histograma - Canal Azul')
plt.xlabel('Intensidad de pixel')
plt.ylabel('Frecuencia')

plt.tight_layout()
plt.show()

```



## 2. Convertir una imagen de color a escala de grises y HSV. Extraer cada canal de color por separado y mostrarlo con histograma.

```
[4]: image_path = '../Images/Lenna.png'
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Get histograms
hist_gray = cv2.calcHist([image_gray], [0], None, [256], [0, 256])
hist_hue = cv2.calcHist([image_hsv], [0], None, [256], [0, 256])
hist_saturation = cv2.calcHist([image_hsv], [1], None, [256], [0, 256])
hist_value = cv2.calcHist([image_hsv], [2], None, [256], [0, 256])

# Plot histograms
plt.figure(figsize=(16, 10))

plt.subplot(2, 3, 1)
plt.imshow(image_rgb)
```

```

plt.title('Imagen original')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(image_gray, cmap='gray')
plt.title('Imagen en escala de grises')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(image_hsv)
plt.title('Imagen en espacio de color HSV')
plt.axis('off')

plt.subplot(2, 4, 5)
plt.plot(hist_gray, color='black', label='Escala de grises')
plt.title('Histograma - Grises')
plt.xlabel('Intensidad')
plt.ylabel('Frecuencia')

plt.subplot(2, 4, 6)
plt.plot(hist_hue, color='red')
plt.title('Histograma - Hue')
plt.xlabel('Matiz')

plt.subplot(2, 4, 7)
plt.plot(hist_saturation, color='green')
plt.title('Histograma - Saturación')
plt.xlabel('Intensidad')

plt.subplot(2, 4, 8)
plt.plot(hist_value, color='blue')
plt.title('Histograma - Valor')
plt.xlabel('Intensidad')

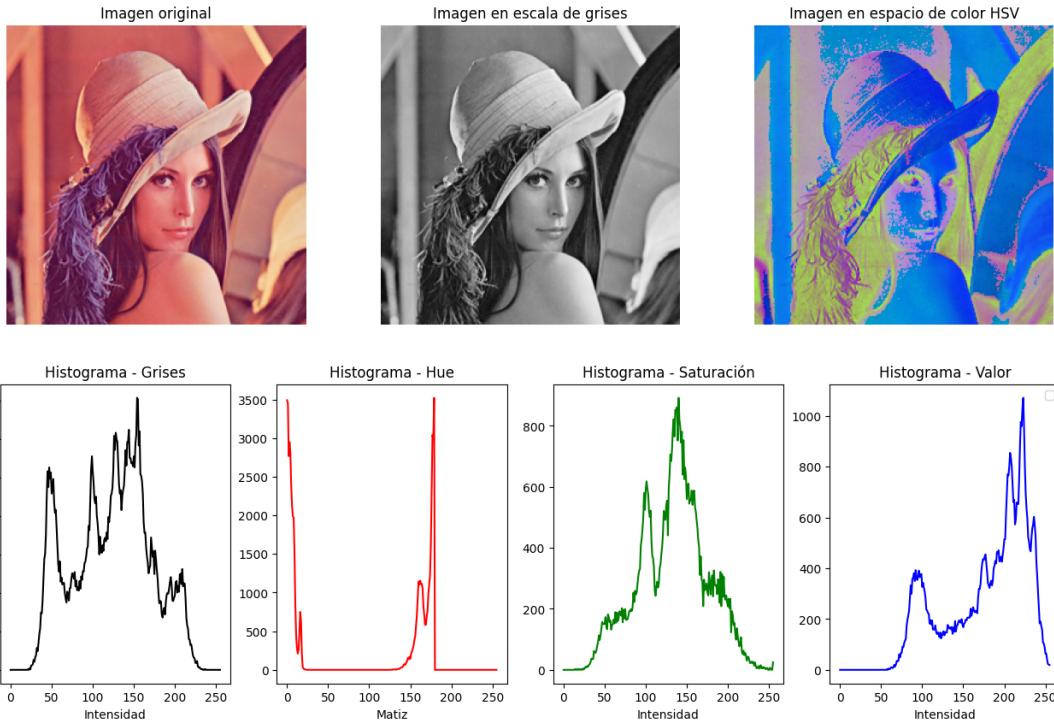
plt.legend()
plt.tight_layout()
plt.show()

```

```

/tmp/ipykernel_8443/2392681427.py:54: UserWarning: No artists with labels found
to put in legend. Note that artists whose label start with an underscore are
ignored when legend() is called with no argument.
    plt.legend()
/tmp/ipykernel_8443/2392681427.py:55: UserWarning: tight_layout not applied:
number of columns in subplot specifications must be multiples of one another.
    plt.tight_layout()

```



3. Cargar una imagen en escala de grises y a color. Posteriormente, modificar un conjunto de píxeles en una región específica (por ejemplo, convertir una zona a negro). Calcular el negativo de una imagen invirtiendo los valores de píxeles.

```
[5]: image_path = '../Images/Lenna.png'
image_bw_path = '../Images/Lenna_grey.png'

image = cv2.imread(image_path)
image_bw = cv2.imread(image_bw_path, cv2.IMREAD_GRAYSCALE)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

area = [100, 200, 100, 200]
target_intensity_bw = 148
target_intensity_color = 180, 100, 50

image_bw[area[0]:area[1], area[2]:area[3]] = target_intensity_bw
image_rgb[area[0]:area[1], area[2]:area[3]] = target_intensity_color

negative_bw = 255 - image_bw
negative_rgb = 255 - image_rgb

plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
```

```
plt.imshow(image_rgb)
plt.title('Imagen original')
plt.axis('off')
plt.subplot(2, 2, 2)
plt.imshow(image_bw, cmap='gray')
plt.title('Imagen en escala de grises')
plt.axis('off')
plt.subplot(2, 2, 3)
plt.imshow(negative_rgb)
plt.title('Imagen negativa')
plt.axis('off')
plt.subplot(2, 2, 4)
plt.imshow(negative_bw, cmap='gray')
plt.title('Imagen negativa en escala de grises')
plt.axis('off')
plt.tight_layout()
plt.show()
```

Imagen original

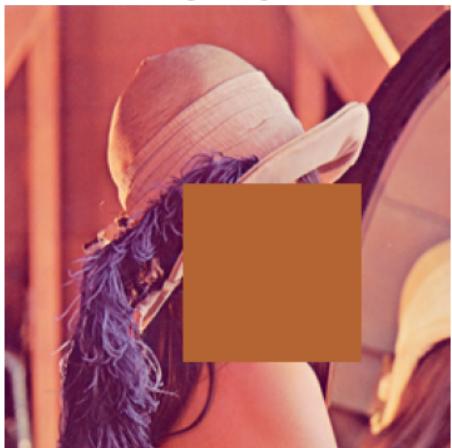


Imagen en escala de grises



Imagen negativa



Imagen negativa en escala de grises



4. Implementar una transformación lineal de la forma  $I' = I + \gamma$  donde  $\gamma$  es el factor de contraste y  $\delta$  es el ajuste de brillo.

```
[6]: # Usando opencv
def apply_linear_transformation(image, contrast, brightness):
    return cv2.convertScaleAbs(image, alpha=contrast, beta=brightness)

# Desde cero
def apply_linear_transformation_manual(image, contrast, brightness):
    height, width = image.shape[:2]
    channels = 1 if len(image.shape) == 2 else image.shape[2]

    result = image.copy()

    for y in range(height):
        for x in range(width):
            if channels == 1:
                val = contrast * image[y, x] + brightness
                result[y, x] = max(0, min(255, int(round(val))))
            else:
                for c in range(channels):
                    val = contrast * image[y, x, c] + brightness
                    result[y, x, c] = max(0, min(255, int(round(val))))
    return result

image_path = '../Images/Lenna.png'
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

alpha, beta = 1.5, 50

image_contrast_opencv = apply_linear_transformation(image, alpha, beta)
image_contrast_manual = apply_linear_transformation_manual(image, alpha, beta)

plt.figure(figsize=(12, 8))
plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title('Imagen original')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(image_contrast_manual)
plt.title('Transformación lineal')
plt.axis('off')
plt.tight_layout()
```

```
plt.show()
```



5. Cargar dos imágenes diferentes (por ejemplo, fotos tomadas en diferentes condiciones de iluminacián). Calcular sus histogramas y compararlos usando diferentes métricas (correlación, chi-cuadrado, intersección).

```
[7]: def plot_histogram(ax, hist, color, title):
    ax.bar(np.arange(256), hist, color=color, width=1)
    ax.set_title(title)
    ax.set_xlim([0, 256])
    ax.set_yticks([])

image_paths = ['./part1/5/got-light.jpg', './part1/5/got-dark.jpg']
image_titles = ['Imagen clara', 'Imagen oscura']
channels = ['R', 'G', 'B']
colors = ['red', 'green', 'blue']

histograms = [[] for _ in range(len(image_paths))]

img_light = cv2.imread(image_paths[0])
img_dark = cv2.imread(image_paths[1])
img_light_rgb = cv2.cvtColor(img_light, cv2.COLOR_BGR2RGB)
img_dark_rgb = cv2.cvtColor(img_dark, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(16, 6))
for i, path in enumerate(image_paths):
    img_bgr = cv2.imread(path)
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
```

```

# Calcular histogramas individuales (en orden BGR)
hist_b = cv2.calcHist([img_bgr], [0], None, [256], [0, 256]).flatten()
hist_g = cv2.calcHist([img_bgr], [1], None, [256], [0, 256]).flatten()
hist_r = cv2.calcHist([img_bgr], [2], None, [256], [0, 256]).flatten()

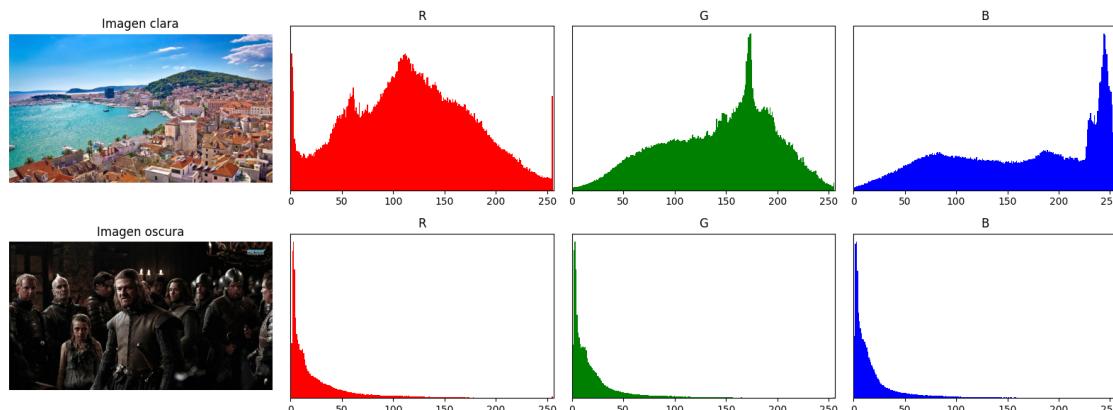
histograms[i] =[hist_r, hist_g, hist_b]

# Mostrar imagen
plt.subplot(2, 4, i * 4 + 1)
plt.imshow(img_rgb)
plt.axis('off')
plt.title(image_titles[i])

# Mostrar histogramas
for j in range(3):
    ax = plt.subplot(2, 4, i * 4 + j + 2)
    plot_histogram(ax, histograms[i][j], colors[j], f'[channels{j}]')

plt.tight_layout()
plt.show()

```



```

[8]: ## Comparacion de histogramas
def compare_histograms(hist1, hist2, method):
    return cv2.compareHist(hist1, hist2, method)

methods = {
    'Correlación': cv2.HISTCMP_CORREL,
    'Chi-cuadrado': cv2.HISTCMP_CHISQR,
    'Intersección': cv2.HISTCMP_INTERSECT,
    'Bhattacharyya': cv2.HISTCMP_BHATTACHARYYA,
}

```

```

# Para cada metodo, calcular la diferencia entre los histogramas de cada canal
results = {}
for method_name, method in methods.items():
    results[method_name] = {
        'R': compare_histograms(histograms[0][0], histograms[1][0], method),
        'G': compare_histograms(histograms[0][1], histograms[1][1], method),
        'B': compare_histograms(histograms[0][2], histograms[1][2], method)
    }

fig, axs = plt.subplots(2, 2, figsize=(12, 8))
axs = axs.flatten()

print(results)

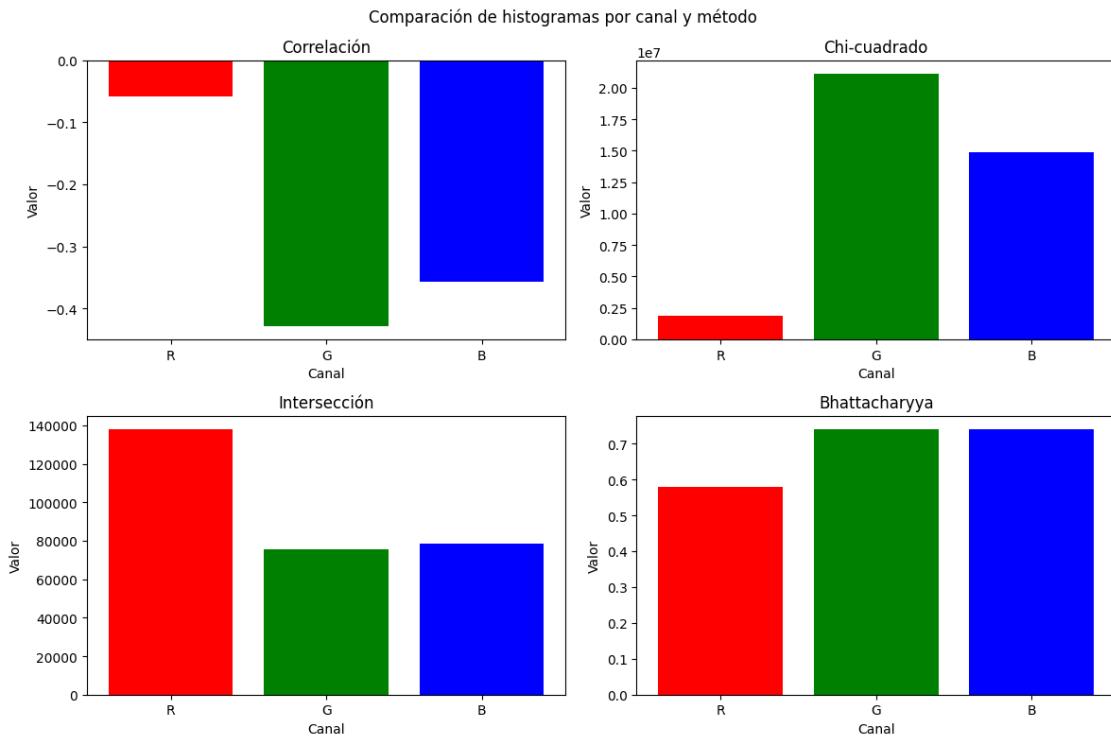
for i, (method_name, values) in enumerate(results.items()):
    canales = list(values.keys())
    valores = list(values.values())
    colores = ['red', 'green', 'blue']

    axs[i].bar(canales, valores, color=colores)
    axs[i].set_title(method_name)
    axs[i].set_xlabel('Canal')
    axs[i].set_ylabel('Valor')

plt.suptitle('Comparación de histogramas por canal y método')
plt.tight_layout()
plt.show()

```

```
{'Correlación': {'R': -0.05871336178219133, 'G': -0.4286901744570133, 'B': -0.357697398835404}, 'Chi-cuadrado': {'R': 1848619.652773731, 'G': 21126999.30071898, 'B': 14891690.656199958}, 'Intersección': {'R': 138186.0, 'G': 75780.0, 'B': 78350.0}, 'Bhattacharyya': {'R': 0.5797754501423779, 'G': 0.7394111561731072, 'B': 0.7410349670889882}}
```



**6. Explicar qué diferencias se pueden observar en los histogramas.** Del mismo modo que las imágenes se ven muy distintas a simple vista, sus histogramas son extremadamente diferentes. Principalmente, podemos apreciar que, mientras que la imagen clara tiene una distribución de intensidades que abarca todo el rango (de valores bajos a altos). La imagen oscura concentra prácticamente todos sus valores en las zonas bajas de intensidad en los tres canales de color.

Esto se refleja en las métricas de comparación del ejercicio 5:

- La correlación es baja (menor a cero en los tres canales).
- El chi cuadrado es sumamente alto (en el orden de  $10^7$ ), lo que indica que la diferencia entre ambas imágenes es muy grande.
- La intersección, aunque nominalmente es alta, es baja para este tipo de comparación, ya que porcentualmente no es significativa.
- El método de Bhattacharyya se aleja de cero, lo que indica que la diferencia entre ambas imágenes es muy grande.

**7. (\*) Transformar la distribución de intensidades de una imagen para que se parezca a la de otra. Implementar el ajuste de histograma usando OpenCV o skimage.exposure.match\_histograms(). Comparar los histogramas antes y después del ajuste.**

```
[104]: def plot_histogram(ax, hist, color, title):
    ax.bar(np.arange(256), hist, color=color, width=1)
    ax.set_title(title)
    ax.set_xlim([0, 256])
    ax.set_yticks([])
```

```

image_paths = ['./part1/5/got-light.jpg', './part1/5/got-dark.jpg']
image_titles = ['Imagen clara', 'Imagen oscura']
channels = ['R', 'G', 'B']
colors = ['red', 'green', 'blue']

histograms = [[] for _ in range(len(image_paths))]

img_light = cv2.imread(image_paths[0])
img_dark = cv2.imread(image_paths[1])
img_light_rgb = cv2.cvtColor(img_light, cv2.COLOR_BGR2RGB)
img_dark_rgb = cv2.cvtColor(img_dark, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(16, 6))
for i, path in enumerate(image_paths):
    img_bgr = cv2.imread(path)
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)

    # Calcular histogramas individuales (en orden BGR)
    hist_b = cv2.calcHist([img_bgr], [0], None, [256], [0, 256]).flatten()
    hist_g = cv2.calcHist([img_bgr], [1], None, [256], [0, 256]).flatten()
    hist_r = cv2.calcHist([img_bgr], [2], None, [256], [0, 256]).flatten()

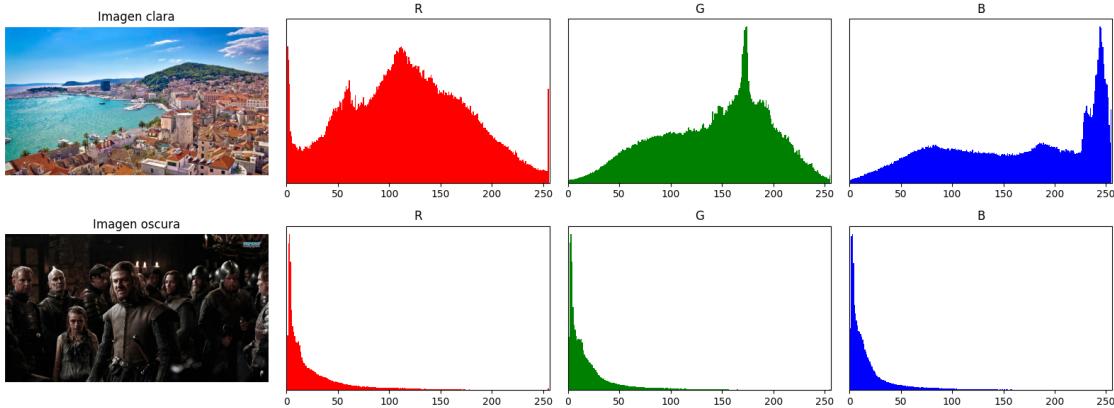
    histograms[i] =[hist_r, hist_g, hist_b]

    # Mostrar imagen
    plt.subplot(2, 4, i * 4 + 1)
    plt.imshow(img_rgb)
    plt.axis('off')
    plt.title(image_titles[i])

    # Mostrar histogramas
    for j in range(3):
        ax = plt.subplot(2, 4, i * 4 + j + 2)
        plot_histogram(ax, histograms[i][j], colors[j], f'{channels[j]}')

plt.tight_layout()
plt.show()

```



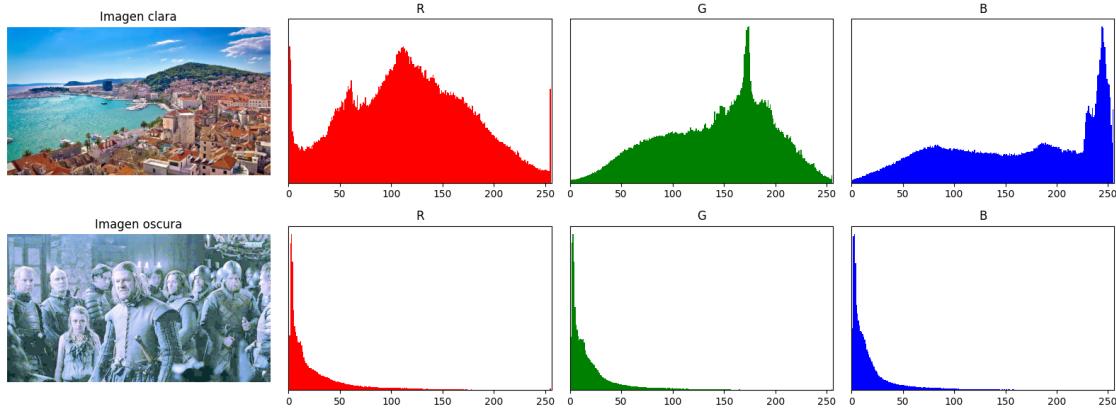
```
[105]: img_matched = match_histograms(img_dark_rgb, img_light_rgb, channel_axis=-1).
         astype(np.uint8)
images = [img_light_rgb, img_matched]

plt.figure(figsize=(16, 6))
for i, img_rgb in enumerate(images):
    # Calcular histogramas RGB
    hist_r = cv2.calcHist([cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR)], [2], None, [256], [0, 256]).flatten()
    hist_g = cv2.calcHist([cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR)], [1], None, [256], [0, 256]).flatten()
    hist_b = cv2.calcHist([cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR)], [0], None, [256], [0, 256]).flatten()

    histograms.append([hist_r, hist_g, hist_b])

    # Mostrar imagen
    plt.subplot(2, 4, i * 4 + 1)
    plt.imshow(img_rgb)
    plt.axis('off')
    plt.title(image_titles[i])

    # Mostrar histogramas
    for j in range(3):
        ax = plt.subplot(2, 4, i * 4 + j + 2)
        plot_histogram(ax, histograms[i][j], colors[j], f'{channels[j]}')
```



8. (\*) Aplicar ecualización de histograma a una imagen en escala de grises. Comparar la imagen original con la ecualizada

```
[106]: image_path = '../Images/Lenna_grey.png'

image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Calcular histograma
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(hist, color='black')
plt.title('Histograma de la imagen en escala de grises')
plt.xlabel('Intensidad de pixel')
plt.ylabel('Frecuencia')
plt.xlim([0, 256])

plt.subplot(1, 2, 2)
plt.imshow(image, cmap='gray')
plt.title('Imagen en escala de grises')
plt.axis('off')

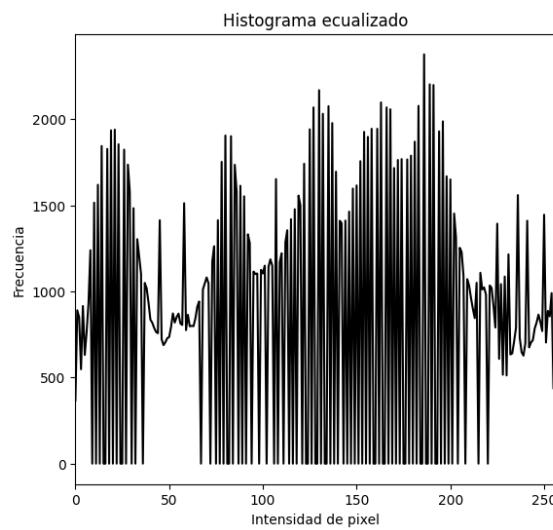
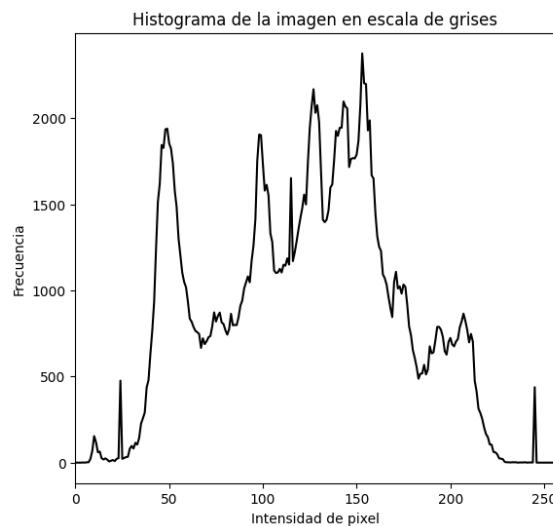
plt.show()

# Ecualizar histograma
image_eq = cv2.equalizeHist(image)
# Calcular histograma ecualizado
hist_eq = cv2.calcHist([image_eq], [0], None, [256], [0, 256])
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plt.plot(hist_eq, color='black')
plt.title('Histograma ecualizado')
```

```

plt.xlabel('Intensidad de pixel')
plt.ylabel('Frecuencia')
plt.xlim([0, 256])
plt.subplot(1, 2, 2)
plt.imshow(image_eq, cmap='gray')
plt.title('Imagen ecualizada')
plt.axis('off')
plt.show()

```



Visualmente, se puede apreciar que el contraste de la imagen es mayor que el de la imagen original. Esto se debe a que la ecualización de histograma redistribuye los valores de intensidad de la imagen original para que estén más uniformemente distribuidos en el rango de intensidades posibles.

Por otro lado, la imagen pierde visiblemente suavidad: se nota las diferencias abruptas de intensidad en los píxeles, lo que puede ser un efecto no deseado en algunas aplicaciones.

**9. (\*) Implementar una umbralización manual eligiendo un valor de umbral. Usar el método de Otsu para calcular un umbral óptimo automáticamente.**

```
[107]: image_path = './part1/8/dog.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Definir umbral
threshold = 125

# Aplicar umbral
_, manual_threshold = cv2.threshold(image, threshold, 255, cv2.THRESH_BINARY)

# Calcular histograma
manual_hist = cv2.calcHist([image], [0], None, [256], [0, 256])

# Usar el algoritmo de Otsu
_, otsu_threshold = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Calcular histograma
otsu_hist = cv2.calcHist([image], [0], None, [256], [0, 256])

plt.figure(figsize=(6, 6))
plt.imshow(image, cmap='gray')
plt.title('Imagen original')
plt.axis('off')
plt.show()

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(manual_threshold, cmap='gray')
plt.title('Umbral manual')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(otsu_threshold, cmap='gray')
plt.title('Umbral de Otsu')
plt.axis('off')
plt.tight_layout()
plt.show()
```

Imagen original



Umbral manual



Umbral de Otsu



[108]: ##### 10. Aplicar ecualización de histograma adaptativa (CLAHE) y analizar su efecto en imágenes con mucho contraste

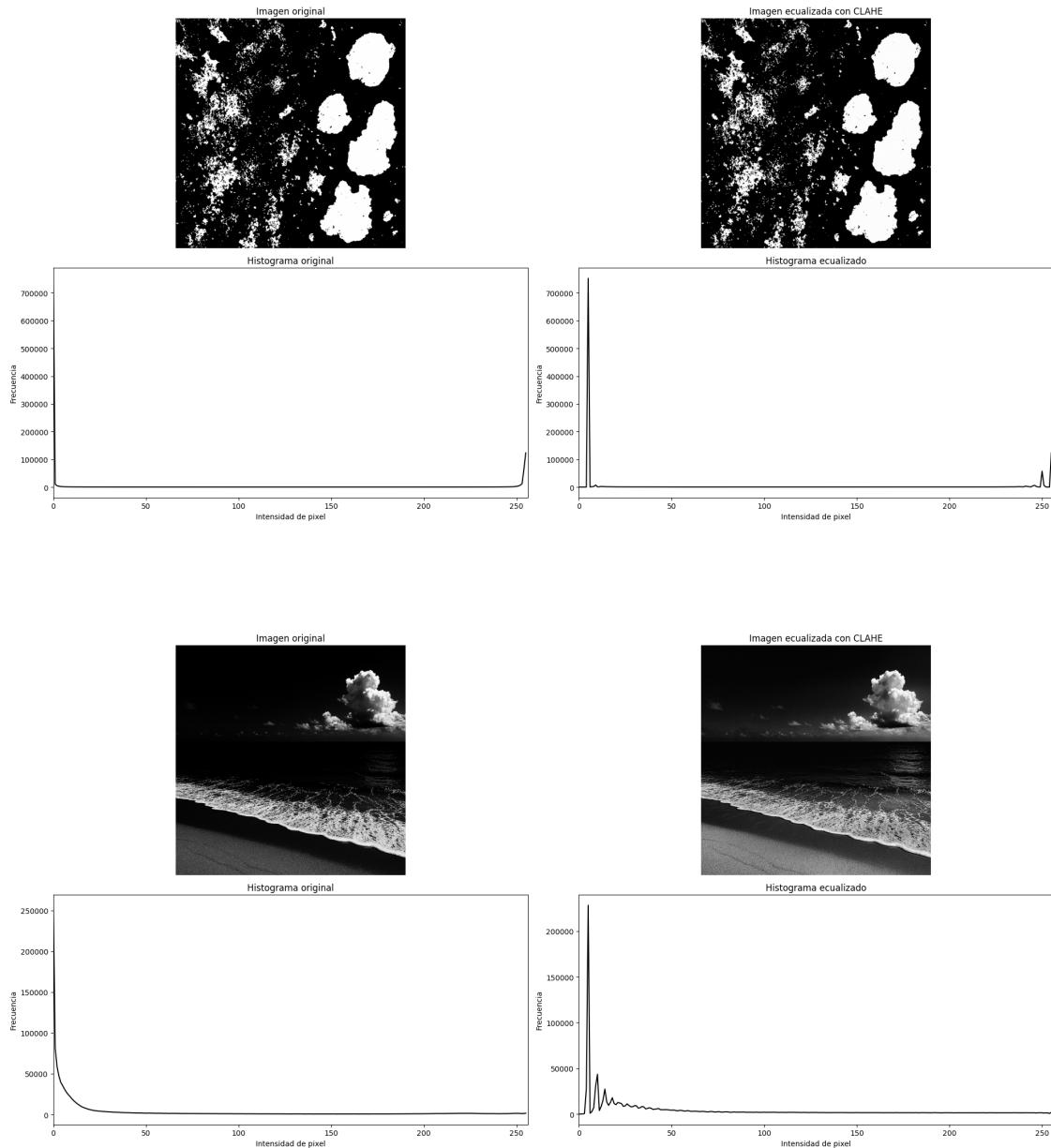
```
image_path = ['./part1/10/contrast_1.png', './part1/10/contrast_2.png', './part1/10/contrast_3.png']

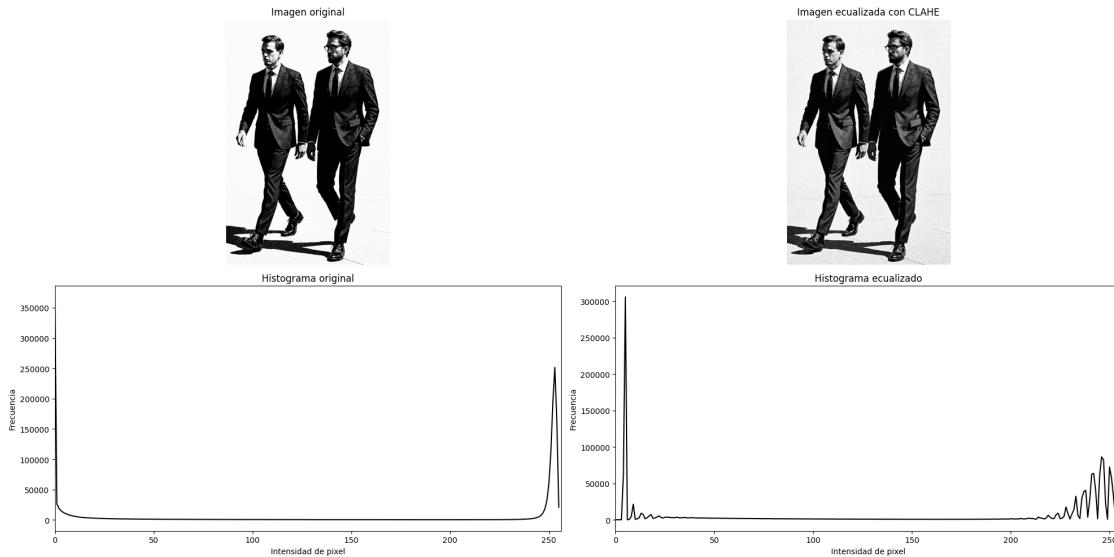
for path in image_path:
    image = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    # Crear objeto CLAHE
    clahe = cv2.createCLAHE(clipLimit=4.0, tileGridSize=(8, 8))
    # Aplicar CLAHE
    image_clahe = clahe.apply(image)
    # Calcular histogramas
    hist = cv2.calcHist([image], [0], None, [256], [0, 256])
    hist_clahe = cv2.calcHist([image_clahe], [0], None, [256], [0, 256])

    # Mostrar imagen original y ecualizada
    plt.figure(figsize=(20, 10))
    plt.subplot(2, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Imagen original')
    plt.axis('off')
    plt.subplot(2, 2, 2)
    plt.imshow(image_clahe, cmap='gray')
    plt.title('Imagen ecualizada con CLAHE')
    plt.axis('off')

    # Mostrar histogramas
    plt.subplot(2, 2, 3)
    plt.plot(hist, color='black')
    plt.title('Histograma original')
    plt.xlabel('Intensidad de pixel')
    plt.ylabel('Frecuencia')
    plt.xlim([0, 256])
    plt.subplot(2, 2, 4)
    plt.plot(hist_clahe, color='black')
    plt.title('Histograma ecualizado')
    plt.xlabel('Intensidad de pixel')
    plt.ylabel('Frecuencia')
    plt.xlim([0, 256])

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





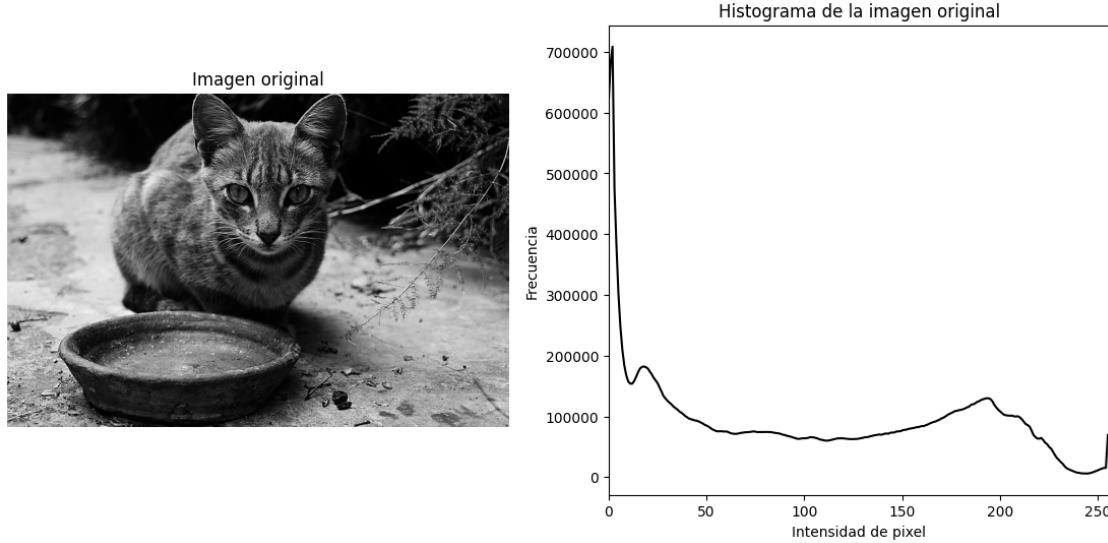
CLAHE parece limitar el reducir de la imagen cuando tenemos imágenes base de muy alto contraste. Al jugar un poco con el clipLimit y el tileSize, se va notando que la imagen tiende a resaltar intensidades de píxeles en el rango medio y suaviza un poco los extremos. Esto es particularmente notorio en la imagen de la playa, en la que se gana detalle en algunas zonas que estaban muy oscuras o muy claras.

**11. (\*) Implementar la transformación gamma  $I'=I^y$ , permitiendo ajustar el valor de  $y$  dinámicamente. Aplicar diferentes valores de  $y$  en distintas regiones de la imagen (por ejemplo, usando una máscara o adaptando  $y$  en función del brillo local). Visualizar el efecto de la corrección gamma en la imagen y en su histograma**

```
[110]: image_path = './part1/11/gamma.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
# convert to grayscale
# image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

hist = cv2.calcHist([image], [0], None, [256], [0, 256])
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Imagen original')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.plot(hist, color='black')
plt.title('Histograma de la imagen original')
plt.xlabel('Intensidad de pixel')
plt.ylabel('Frecuencia')
plt.xlim([0, 256])
```

```
plt.show()
```

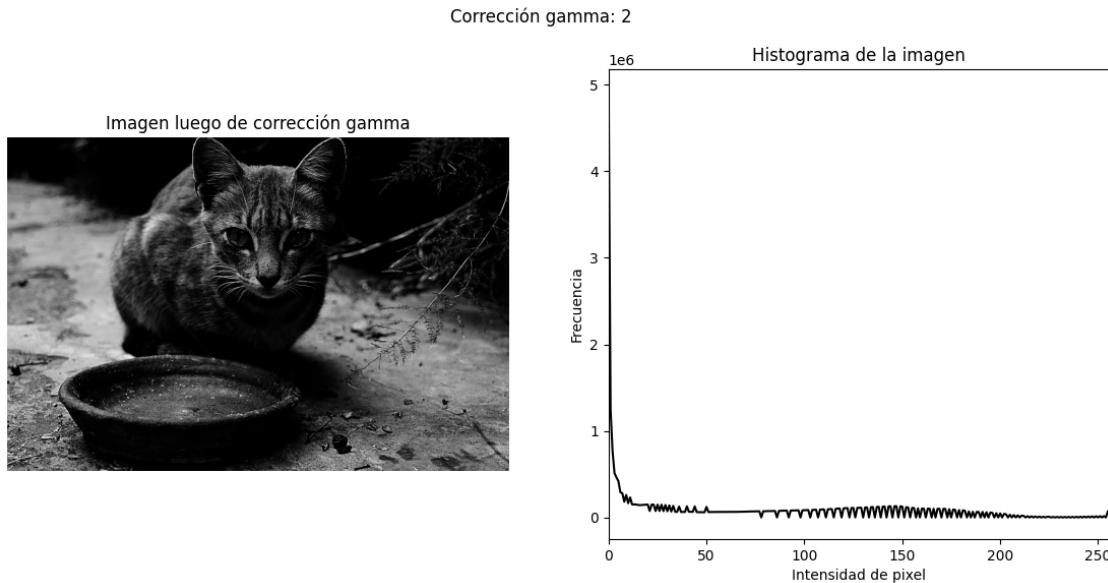


```
[71]: def gamma_correction(image, gamma):
    image_normalized = image / 255.0
    # Aplicar la corrección gamma
    image_corrected = np.power(image_normalized, gamma)

    # Normalizar la imagen corregida
    image_corrected = np.clip(image_corrected * 255, 0, 255).astype(np.uint8)
    return image_corrected

def plot_image_and_histogram(image, title):
    hist = cv2.calcHist([image], [0], None, [256], [0, 256])
    plt.figure(figsize=(14, 6))
    plt.suptitle(title)
    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Imagen luego de corrección gamma')
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.plot(hist, color='black')
    plt.title('Histograma de la imagen')
    plt.xlabel('Intensidad de pixel')
    plt.ylabel('Frecuencia')
    plt.xlim([0, 256])
    plt.show()
```

```
[72]: # 1. Aplicar una gamma a toda la imagen
gamma = 2
image_gamma = gamma_correction(image, gamma)
plot_image_and_histogram(image_gamma, f'Corrección gamma: {gamma}')
```

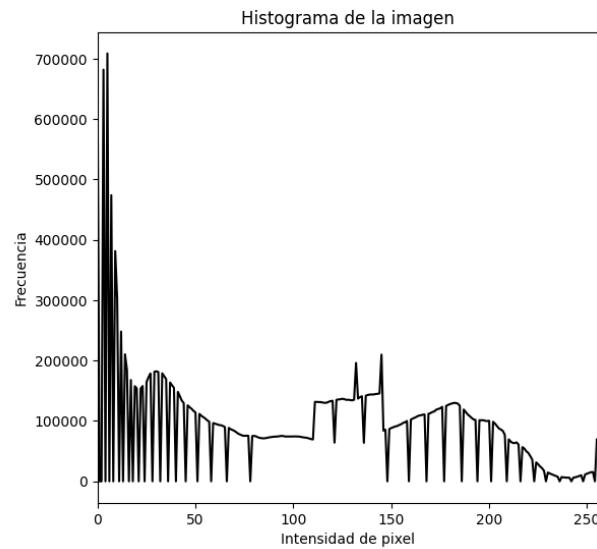


```
[73]: # 2. Aplicar gammas distintos a las regiones claras y oscuras
gamma_light = 1.2
gamma_dark = 0.8

corrected = image.copy()
corrected[image > 127] = gamma_correction(image[image > 127], gamma_light)
corrected[image <= 127] = gamma_correction(image[image <= 127], gamma_dark)

plot_image_and_histogram(corrected, 'Corrección gamma usando máscara')
```

### Corrección gamma usando máscara



```
[74]: # 3. Usar una máscara para aplicar la corrección gamma
image_copy = image.copy()
_, mask = cv2.threshold(image_copy, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Show the mask
plt.figure(figsize=(14, 6))
plt.title('Máscara')
plt.imshow(mask, cmap='gray')
plt.axis('off')
plt.show()

# Aplicar la corrección gamma a la imagen original usando la máscara
image_gamma_masked = image.copy()
image_gamma_masked[mask == 255] = gamma_correction(image[mask == 255], gamma_light)
image_gamma_masked[mask == 0] = gamma_correction(image[mask == 0], gamma_dark)
plot_image_and_histogram(image_gamma_masked, f'Corrección gamma: {gamma_light} (claro), {gamma_dark} (oscuro) con máscara')
```

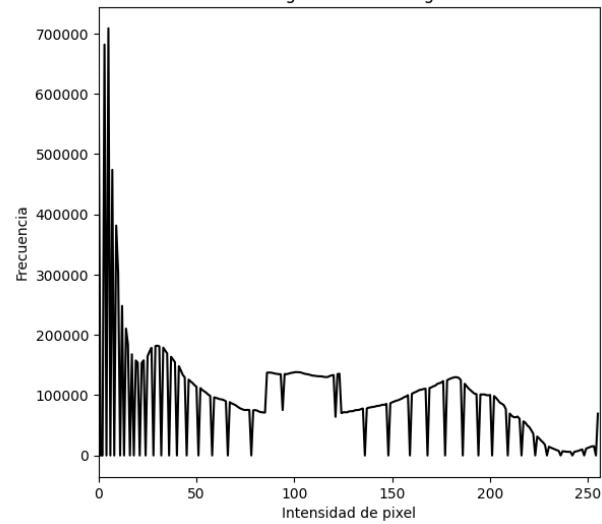
Máscara



Corrección gamma: 1.2 (claro), 0.8 (oscuro) con máscara



Histograma de la imagen



## 2.2 2. Combinación de imágenes

2.2.1 1. Suma de imágenes con ponderación: Cargar dos imágenes del mismo tamaño y combínalas con una ponderación específica usando la función cv2.addWeighted().

```
[105]: def load_image(path):
    return Image.open(path)

def get_image_parameters(path):
    image = Image.open(path)
    width, height = image.size
    bit_depth = image.bits
    size = os.path.getsize(path)
    return width, height, bit_depth, size

def convert_bgr_to_rgb(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

def convert_rgb_to_bgr(image):
    return cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

def print_image_data(path):
    width, height, bit_depth, size = get_image_parameters(path)
    print(f" | Resolución: {width}x{height} | Profundidad de color: {bit_depth} bits")
    print(f" | Tamaño: {size} kb")
    print("-" * 50)

def open_image_cv2(path):
    return cv2.imread(path)

def save_image(path, image):
    cv2.imwrite(path, image)

print_image_data("../Images/gray1.jpeg")
print_image_data("../Images/gray2.jpeg")
```

| Resolución: 980x980 | Profundidad de color: 8 | Tamaño: 58588 kb  
-----  
| Resolución: 980x980 | Profundidad de color: 8 | Tamaño: 102759 kb  
-----

```
[106]: bin1 = open_image_cv2("../Images/gray1.jpeg")
bin2 = open_image_cv2("../Images/gray2.jpeg")
```

```
[107]: plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(bin1)
```

```

plt.title("Binary1 Image")
plt.axis("off")

plt.subplot(1, 2, 1)
plt.imshow(bin1)
plt.title("Binary1 Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(bin2)
plt.title("Binary2 Image")
plt.axis("off")

plt.tight_layout()
plt.show()

alphas = np.arange(0.0, 1.01, 0.2)
fig, axes = plt.subplots(1, len(alphas), figsize=(20, 4))

for i, alpha in enumerate(alphas):
    beta = 1.0 - alpha
    added_image = cv2.addWeighted(bin1, alpha, bin2, beta, 0.0)

    axes[i].imshow(added_image)
    axes[i].set_title(f"={alpha:.1f}, ={beta:.1f}")
    axes[i].axis("off")

cv2.imwrite(f"part2/1/added_image_alpha_{alpha:.1f}.jpeg", added_image)

plt.tight_layout()
plt.show()

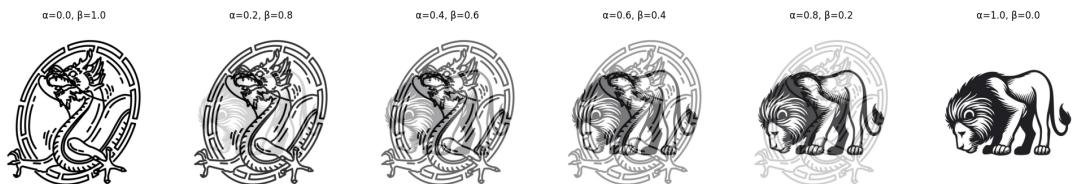
```

Binary1 Image



Binary2 Image





## 2.2.2 2. Resta de imágenes: Realizar la resta de dos imágenes para resaltar las diferencias entre ellas con cv2.subtract().

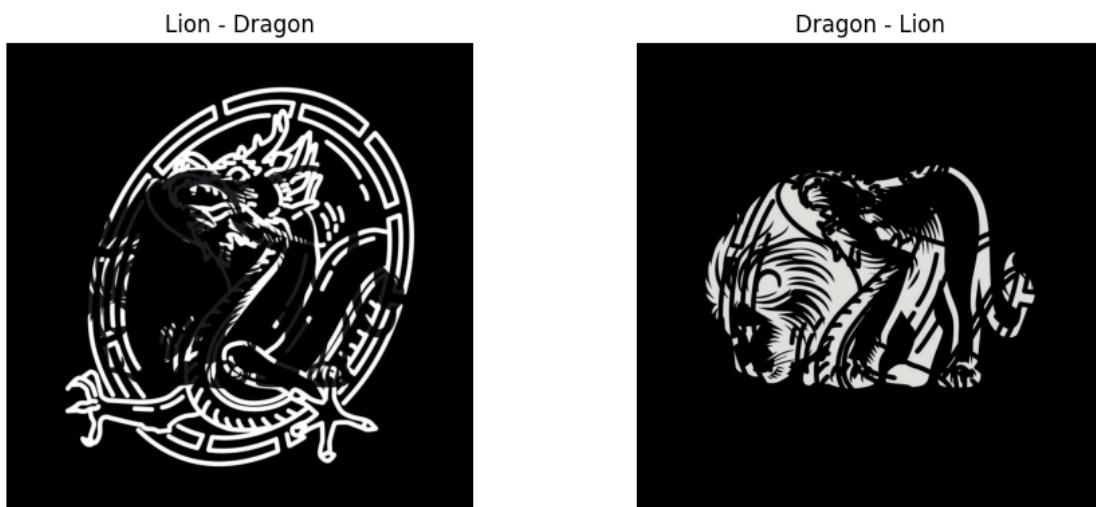
```
[108]: lion_subtract_dragon = cv2.subtract(bin1,bin2)
dragon_subtract_lion = cv2.subtract(bin2,bin1)

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.imshow(lion_subtract_dragon)
plt.title("Lion - Dragon")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(dragon_subtract_lion)
plt.title("Dragon - Lion")
plt.axis("off")

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



**2.2.3 3. (\*) Multiplicación y división de imágenes:** Multiplicar y dividir dos imágenes pixel a pixel utilizando cv2.multiply() y cv2.divide(), observando cómo afecta el brillo y contraste.

```
[109]: #We need resize the type of the image to avoid overflow
bin1_32 = bin1.astype(np.float32)
bin2_32 = bin2.astype(np.float32)

img_multiply = cv2.multiply(bin1_32,bin2_32)

#Need to normalize
img_multiply = cv2.normalize(img_multiply,None,0,255,cv2.NORM_MINMAX, dtype=cv2.
                             CV_8U)

#save_image("part2/3/multiply.jpeg",img_multiply)
```

```
[110]: bin2_32[bin2_32 == 0] = 1e-5

# Division
division = cv2.divide(bin1_32, bin2_32)
division *= 255.0 / division.max()

division = np.clip(division, 0, 255).astype(np.uint8)
```

```
[112]: plt.figure(figsize=(10, 5))

plt.subplot(121)
plt.imshow(img_multiply, cmap='gray')
plt.title("Multiplicación (Lion × Dragon)")
plt.axis("off")

plt.subplot(122)
plt.imshow(division, cmap='gray')
plt.title("División")
plt.axis("off")

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

Multiplicación (Lion x Dragon)



División



La multiplicación de imágenes tiende a aumentar el brillo en áreas con valores de píxel altos, pero puede oscurecer las áreas con valores bajos, reduciendo el contraste en esas zonas. Además, si los valores de los píxeles exceden el rango máximo (255 en imágenes de 8 bits), puede haber saturación, lo que provoca una pérdida de detalles en las áreas brillantes.

La división de imágenes suele reducir el brillo de las áreas más brillantes si el valor del denominador es alto, mientras que puede aumentar el brillo de las áreas oscuras si el denominador tiene valores bajos. La división puede también reducir el contraste, especialmente en áreas donde la diferencia de valores entre las imágenes es pequeña, y si un valor en el denominador es cercano a cero, puede provocar un aumento de brillo en ciertas zonas.

#### 2.2.4 4. Máscara binaria con operadores relacionales: Convierte una imagen a escala de grises y genera una máscara binaria donde los valores sean mayores a un umbral con operadores relacionales ( $>$ , $<$ ).

```
[119]: lenna = open_image_cv2("../Images/Lenna.png")

img_gray2 = cv2.cvtColor(lenna, cv2.COLOR_BGR2GRAY)

threshold = 100

mask_major = (img_gray2 >= threshold).astype(np.uint8) * 255

mask_minor = (img_gray2 < threshold).astype(np.uint8) * 255

plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)
```

```

plt.title("Grayscale")
plt.imshow(img_gray2, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title("Mask Major (> 100)")
plt.imshow(mask_major, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title("Mask Minor (< 100)")
plt.imshow(mask_minor, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()

```



**2.2.5 5. (\*) Combinación con operadores lógicos:** Usa operadores booleanos (`cv2.bitwise_and`, `cv2.bitwise_or`, `cv2.bitwise_xor`) para fusionar imágenes basándose en una máscara binaria. Describir que sucede en cada caso

```
[121]: img1 = np.zeros((400,600), dtype=np.uint8)
img1[100:300,200:400] = 255
img2 = np.zeros((400,600), dtype=np.uint8)
img2 = cv2.circle(img2,(300,200),125,(255),-1)

and_image = cv2.bitwise_and(img1,img2)
```

**and** Mantiene solo los píxeles donde ambas imágenes tienen valores distintos de cero en el área de la máscara. Es útil para extraer regiones comunes: si un píxel es negro (0) en una imagen o en la máscara, el resultado será negro.

```
[122]: or_image = cv2.bitwise_or(img1,img2)
```

**or** Combina todas las áreas donde al menos una imagen tiene píxeles no nulos en la máscara. Si un píxel es blanco (255) en una imagen o en la máscara, el resultado será blanco.

```
[123]: xor_image = cv2.bitwise_xor(img1,img2)
```

**xor** Destaca píxeles donde solo una de las imágenes tiene valores no nulos (exclusividad). Si un píxel difiere entre las dos imágenes (una es blanca y la otra negra), XOR lo mostrará como blanco; si son iguales, será negro.

```
[124]: plt.figure(figsize=(15, 6))
```

```
plt.subplot(2, 3, 1)
plt.title("Imagen 1 (Rectángulo)")
plt.imshow(img1, cmap='gray')
plt.axis('off')

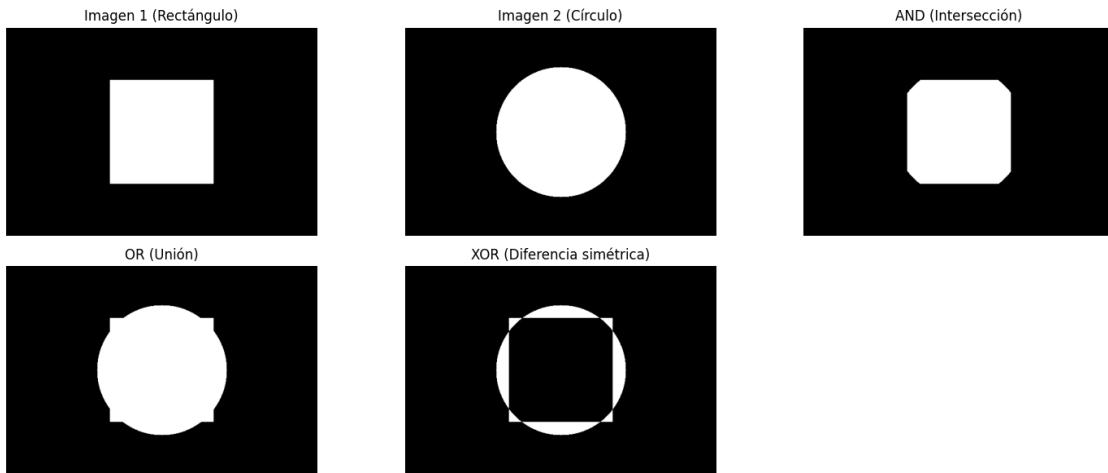
plt.subplot(2, 3, 2)
plt.title("Imagen 2 (Círculo)")
plt.imshow(img2, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.title("AND (Intersección)")
plt.imshow(and_image, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 4)
plt.title("OR (Unión)")
plt.imshow(or_image, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.title("XOR (Diferencia simétrica)")
plt.imshow(xor_image, cmap='gray')
plt.axis('off')

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



**2.2.6 6. Creación de una imagen compuesta:** Utilizar una imagen con fondo negro y otra con fondo blanco, aplicando una máscara binaria para superponer un objeto de una imagen sobre otra.

```
[126]: img2_white_background = np.full((400,600),255, dtype=np.uint8)
img2 = cv2.circle(img2_white_background,(300,200),125,(0),-1)

mask = img1.copy()

circle = cv2.bitwise_and(img1,img2_white_background, mask=cv2.bitwise_not(mask))

rectangle = cv2.bitwise_and(img1,img2_white_background, mask=mask)

compose_img = cv2.add(rectangle, circle)

plt.figure(figsize=(15, 6))

plt.subplot(2, 3, 1)
plt.title("Rectángulo (img1)")
plt.imshow(img1, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.title("Círculo en fondo blanco (img2)")
plt.imshow(img2_white_background, cmap='gray')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.title("Máscara")
plt.imshow(mask, cmap='gray')
```

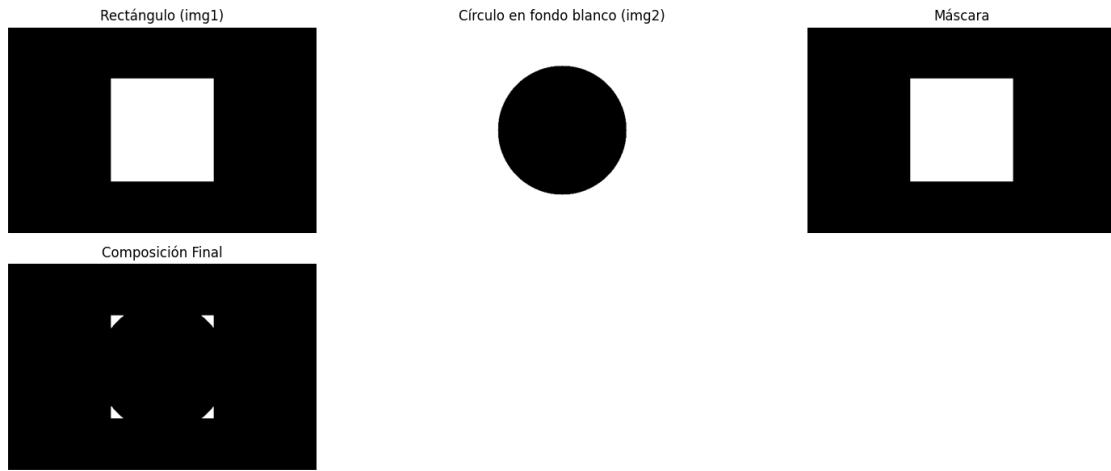
```

plt.axis('off')

plt.subplot(2, 3, 4)
plt.title("Composición Final")
plt.imshow(compose_img, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()

```



## 2.2.7 7. Operaciones avanzadas con imágenes en color: Cargar imágenes en color y realiza operaciones aritméticas como suma y resta, observando cómo afectan cada canal de color (R, G, B).

```
[127]: windows_image = open_image_cv2("../Images/Windows.jpeg")
linux_image = open_image_cv2("../Images/Linux.jpeg")

img_add = cv2.add(windows_image, linux_image)
```

```
[129]: img_sub1 = cv2.subtract(windows_image,linux_image)
img_sub2 = cv2.subtract(linux_image,windows_image)
```

```
[131]: plt.figure(figsize=(14, 8))

plt.subplot(2, 3, 1)
plt.title("Windows")
plt.imshow(cv2.cvtColor(windows_image, cv2.COLOR_BGR2RGB))
plt.axis('off')
```

```

plt.subplot(2, 3, 2)
plt.title("Linux")
plt.imshow(cv2.cvtColor(linux_image, cv2.COLOR_BGR2RGB))
plt.axis('off')

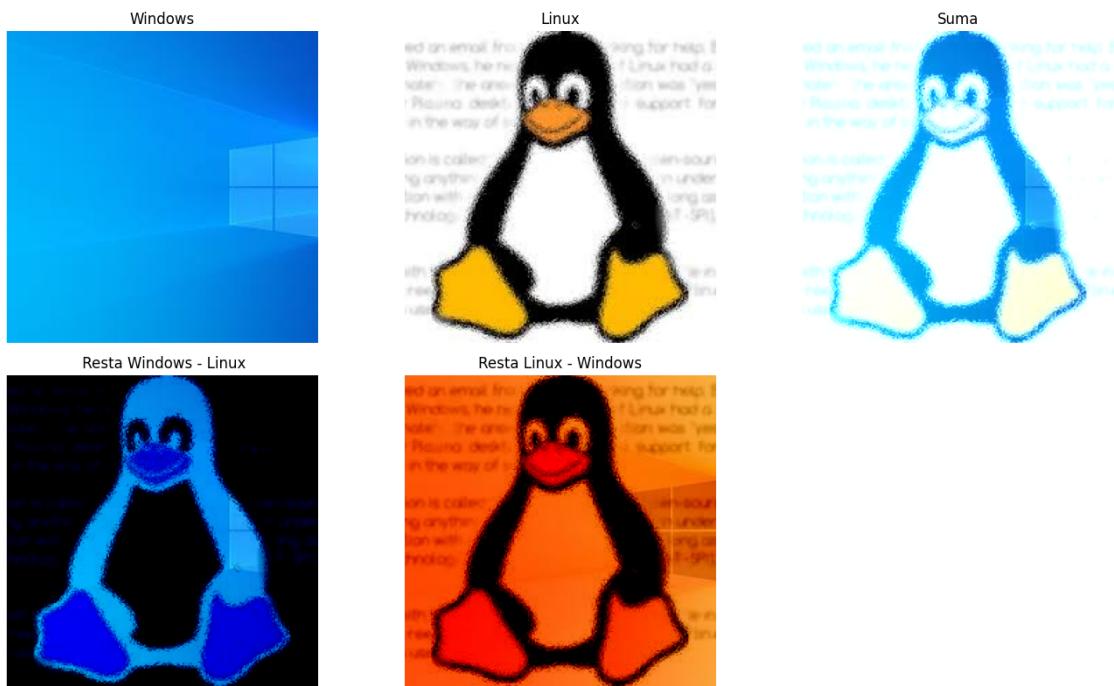
plt.subplot(2, 3, 3)
plt.title("Suma")
plt.imshow(cv2.cvtColor(img_add, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(2, 3, 4)
plt.title(" Resta Windows - Linux")
plt.imshow(cv2.cvtColor(img_sub1, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(2, 3, 5)
plt.title("Resta Linux - Windows")
plt.imshow(cv2.cvtColor(img_sub2, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.tight_layout()
plt.show()

```



Al sumar imágenes, los valores de cada canal aumentan, lo que puede intensificar el brillo de los colores, pero si se excede el valor máximo (255), se produce saturación, haciendo que los colores

se vean más claros o incluso blancos en áreas específicas. En la resta, los valores de cada canal disminuyen, oscureciendo los colores y pudiendo llevar a valores cercanos a cero, lo que genera tonos más oscuros o negros.

## 2.2.8 8. (\*) Uso de operadores lógicos para reemplazar partes de una imagen: Reemplazar un área específica de una imagen con otra utilizando operadores lógicos y relacionales para definir la región de interés (ROI).

```
[139]: main_image = open_image_cv2("../Images/river.jpeg")
substitution_image = open_image_cv2("../Images/mountains.jpeg")

# Divide channels
blue, green, red = cv2.split(main_image)

# We can create different conditions playing with the channels
mask = (red > 100) & (blue > 200) | ~(green < 100)

mask = mask.astype(np.uint8) * 255

substitution_image = cv2.resize(substitution_image, (main_image.shape[1], ↴
    ↵main_image.shape[0]))

result = main_image.copy()
result[mask == 255] = substitution_image[mask == 255]
```

```
[141]: plt.figure(figsize=(16, 8))

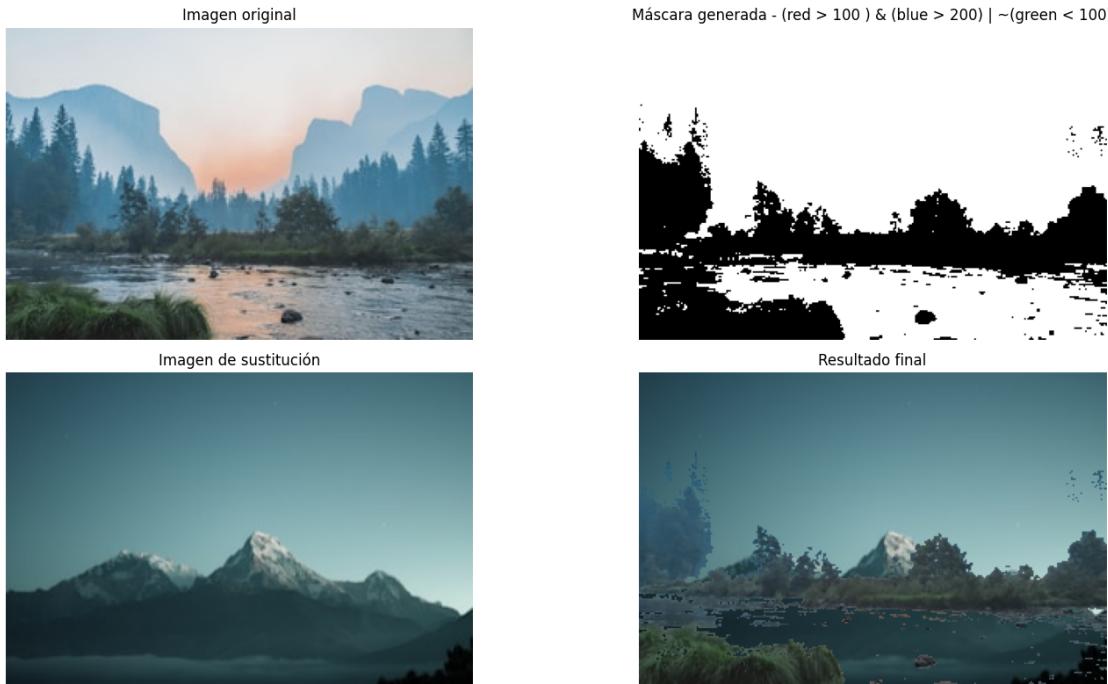
plt.subplot(2, 2, 1)
plt.title("Imagen original")
plt.imshow(cv2.cvtColor(main_image, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(2, 2, 2)
plt.title("Máscara generada - (red > 100) & (blue > 200) | ~(green < 100)")
plt.imshow(mask, cmap='gray')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.title("Imagen de sustitución")
plt.imshow(cv2.cvtColor(substitution_image, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(2, 2, 4)
plt.title("Resultado final")
plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
plt.axis('off')
```

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



## 2.3 3. Dominio Espacial

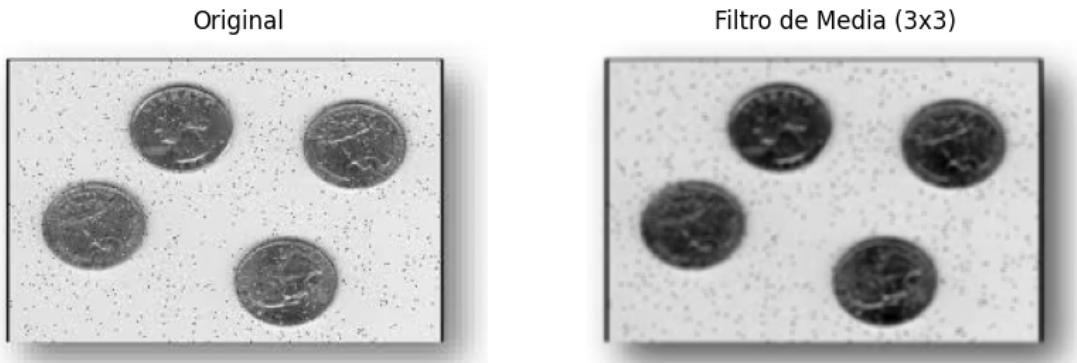
2.3.1 1. Filtro de Media: Implementar un filtro de media en una imagen usando convolución con un kernel de promediado.

```
[126]: noise_image = cv2.imread("../Images/noise.jpeg", cv2.IMREAD_GRAYSCALE)

kernel_size = (3, 3)
kernel_media = np.ones(kernel_size, np.float32) / (kernel_size[0] * kernel_size[1])

filtered = cv2.filter2D(noise_image, -1, kernel_media)

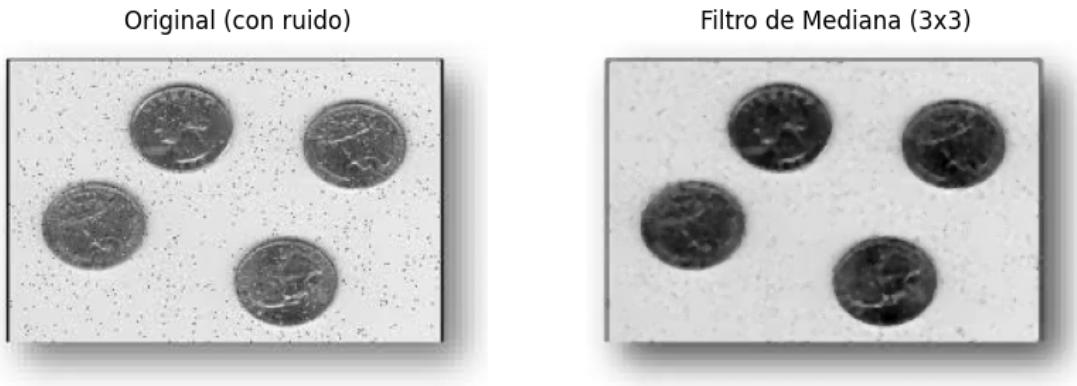
plt.figure(figsize=(10, 5))
plt.subplot(121), plt.imshow(noise_image, cmap='gray'), plt.title('Original'), plt.axis('off')
plt.subplot(122), plt.imshow(filtered, cmap='gray'), plt.title('Filtro de Media (3x3)'), plt.axis('off')
plt.show()
```



### 2.3.2 2. Filtro de Mediana: Aplicar un filtro de mediana para reducir el ruido en una imagen.

```
[127]: filtered = cv2.medianBlur(noise_image, ksize=3)

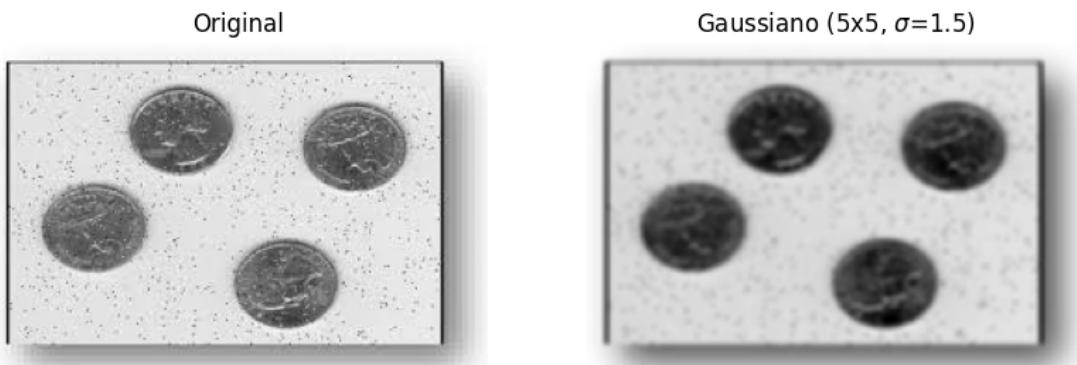
plt.figure(figsize=(10, 5))
plt.subplot(121), plt.imshow(noise_image, cmap='gray'), plt.title('Original (con ruido)'), plt.axis('off')
plt.subplot(122), plt.imshow(filtered, cmap='gray'), plt.title('Filtro de Mediana (3x3)'), plt.axis('off')
plt.show()
```



### 2.3.3 3. Filtro Gaussiano: Aplicar un filtro gaussiano para suavizar una imagen y analizar su efecto en los bordes.

```
[129]: ksize = (5, 5)
sigma = 1.5
filtered = cv2.GaussianBlur(noise_image, ksize, sigmaX=sigma)

plt.figure(figsize=(10, 5))
plt.subplot(121), plt.imshow(noise_image, cmap='gray'), plt.title('Original'), plt.axis("off")
plt.subplot(122), plt.imshow(filtered, cmap='gray'), plt.title(r'Gaussiano (5x5, $\sigma=1.5$)'), plt.axis("off")
plt.show()
```



El filtro Gaussiano suaviza la imagen al aplicar una convolución con una función de campana, lo que reduce el ruido de alta frecuencia. Como resultado, los bordes se ven menos definidos, ya que el filtro atenúa los cambios bruscos de intensidad entre regiones. En la imagen procesada (5x5,  $\sigma=1.5$ ), se nota que las monedas pierden nitidez en sus contornos, y el ruido salpicado de la imagen

original disminuye visiblemente.

### 2.3.4 4. Filtro Laplaciano: Aplicar el operador de Laplace para detectar bordes en una imagen en escala de grises.

```
[134]: lenna_image = cv2.imread("../Images/Lenna.png", cv2.IMREAD_GRAYSCALE)

laplacian = cv2.Laplacian(lenna_image, cv2.CV_64F)

laplacian_abs = np.uint8(np.absolute(laplacian))

plt.figure(figsize=(10, 5))
plt.subplot(121), plt.imshow(lenna_image, cmap='gray'), plt.title('Original'), plt.axis("off")
plt.subplot(122), plt.imshow(laplacian_abs, cmap='gray'), plt.title('Laplaciano (Bordes)'), plt.axis("off")
plt.show()
```



### 2.3.5 5. Filtro de Sobel: Calcular el gradiente de una imagen usando los filtros de Sobel en las direcciones X e Y.

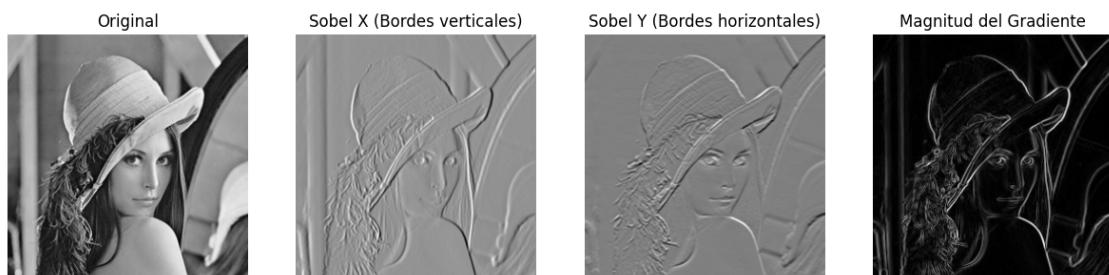
```
[ ]: sobel_x = cv2.Sobel(lenna_image, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(lenna_image, cv2.CV_64F, 0, 1, ksize=3)

magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
magnitude = np.uint8(magnitude / magnitude.max() * 255)
```

```

# Mostrar resultados
plt.figure(figsize=(15, 5))
plt.subplot(141), plt.imshow(lenna_image, cmap='gray'), plt.title('Original'), plt.axis("off")
plt.subplot(142), plt.imshow(sobel_x, cmap='gray'), plt.title('Sobel X (Bordes verticales)'), plt.axis("off")
plt.subplot(143), plt.imshow(sobel_y, cmap='gray'), plt.title('Sobel Y (Bordes horizontales)'), plt.axis("off")
plt.subplot(144), plt.imshow(magnitude, cmap='gray'), plt.title('Magnitud del Gradiente'), plt.axis("off")
plt.show()

```



### 2.3.6 6. Filtro de Scharr: Comparar el resultado del filtro de Sobel con el filtro de Scharr.

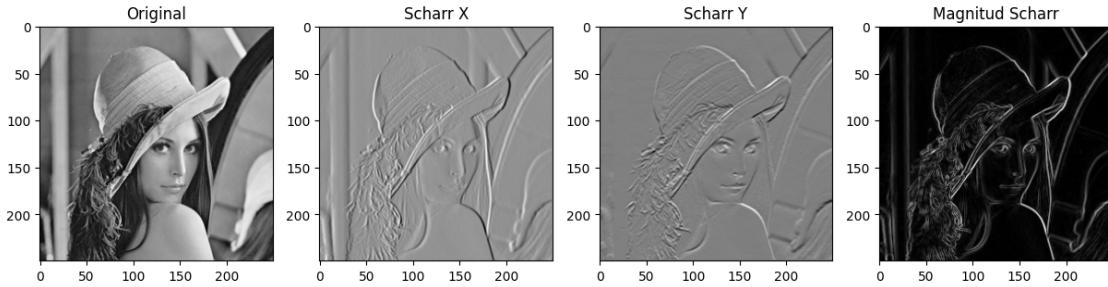
```

[136]: # Aplicar Scharr en X e Y
scharr_x = cv2.Scharr(lenna_image, cv2.CV_64F, 1, 0) # Horizontal
scharr_y = cv2.Scharr(lenna_image, cv2.CV_64F, 0, 1) # Vertical

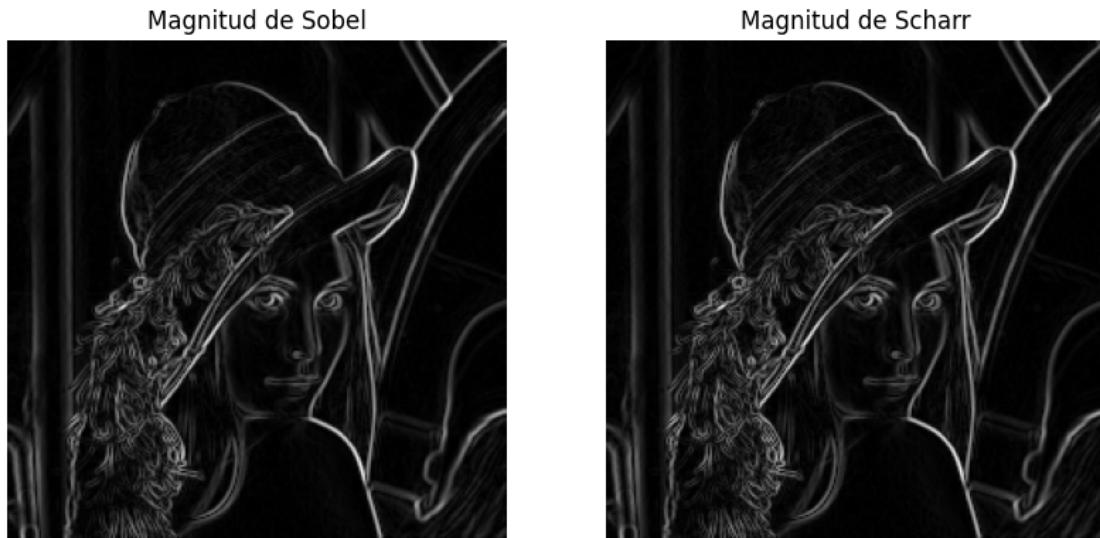
# Calcular magnitud
magnitude_scharr = np.sqrt(scharr_x**2 + scharr_y**2)
magnitude_scharr = np.uint8(magnitude_scharr / magnitude_scharr.max() * 255)

# Mostrar resultados
plt.figure(figsize=(15, 5))
plt.subplot(141), plt.imshow(lenna_image, cmap='gray'), plt.title('Original'), plt.axis("off")
plt.subplot(142), plt.imshow(scharr_x, cmap='gray'), plt.title('Scharr X'), plt.axis("off")
plt.subplot(143), plt.imshow(scharr_y, cmap='gray'), plt.title('Scharr Y'), plt.axis("off")
plt.subplot(144), plt.imshow(magnitude_scharr, cmap='gray'), plt.title('Magnitud Scharr'), plt.axis("off")
plt.show()

```



```
[141]: plt.figure(figsize=(10, 5))
plt.subplot(121), plt.imshow(magnitude, cmap="gray"), plt.title("Magnitud de Sobel"), plt.axis("off")
plt.subplot(122), plt.imshow(magnitude_scharr, cmap="gray"), plt.title("Magnitud de Scharr"), plt.axis("off")
plt.show()
```



### 2.3.7 7. Filtro de Prewitt: Aplicar el operador de Prewitt y comparar con Sobel.

```
[142]: kernel_prewitt_x = np.array([[1, 0, -1],
                                 [0, 1, 0],
                                 [-1, 0, 1]], dtype=np.float32)

kernel_prewitt_y = np.array([[1, 0, 0],
                            [0, -1, 0],
                            [0, 0, -1]], dtype=np.float32)
```

```

# Aplicar convolución
prewitt_x = cv2.filter2D(lenna_image, cv2.CV_64F, kernel_prewitt_x)
prewitt_y = cv2.filter2D(lenna_image, cv2.CV_64F, kernel_prewitt_y)

magnitude_prewitt = np.sqrt(prewitt_x**2 + prewitt_y**2)
magnitude_prewitt = np.uint8(magnitude_prewitt / magnitude_prewitt.max() * 255)

plt.subplot(221)
plt.imshow(lenna_image, cmap='gray')
plt.title('Imagen Original'), plt.axis('off')

plt.subplot(222)
plt.imshow(np.absolute(prewitt_x), cmap='gray')
plt.title('Prewitt X (Bordes verticales)'), plt.axis('off')

plt.subplot(223)
plt.imshow(np.absolute(prewitt_y), cmap='gray')
plt.title('Prewitt Y (Bordes horizontales)'), plt.axis('off')

plt.subplot(224)
plt.imshow(magnitude_prewitt, cmap='gray')
plt.title('Magnitud Prewitt (Resultado)'), plt.axis('off')

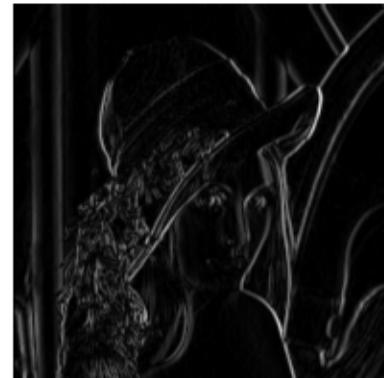
plt.tight_layout()
plt.show()

```

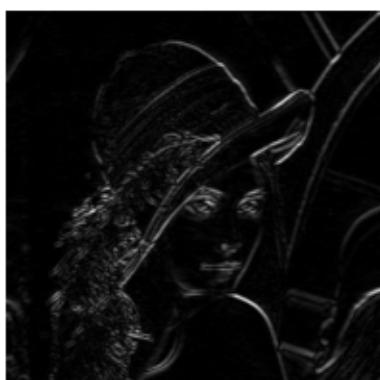
Imagen Original



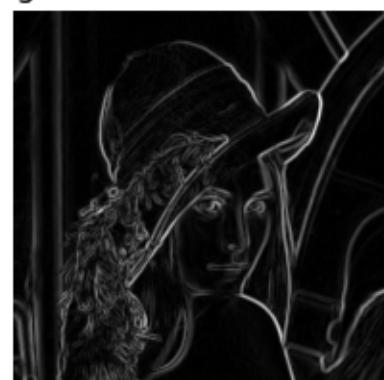
Prewitt X (Bordes verticales)



Prewitt Y (Bordes horizontales)



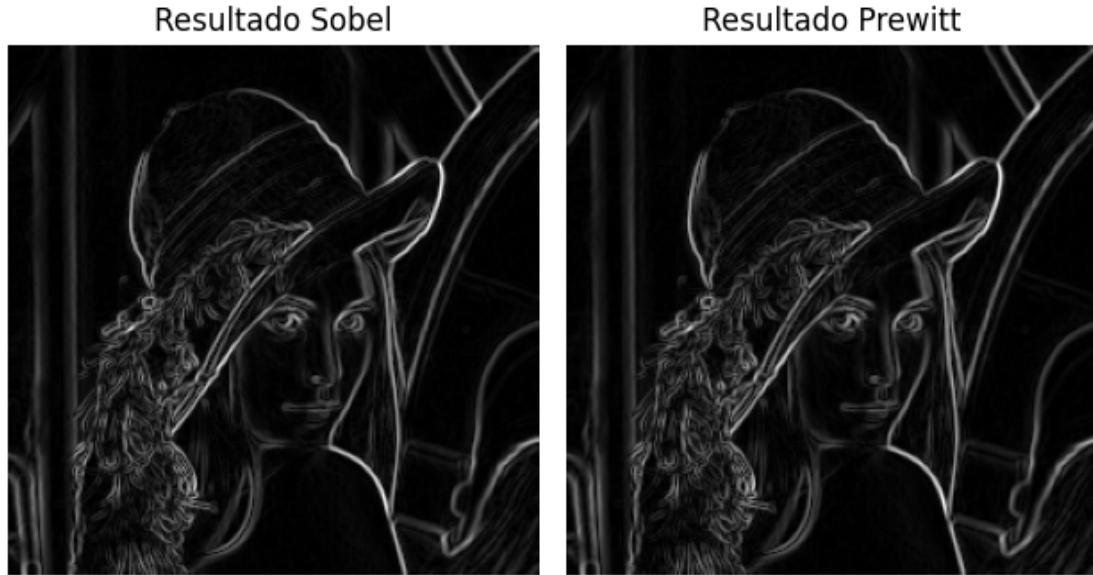
Magnitud Prewitt (Resultado)



```
[143]: plt.subplot(121)
plt.imshow(magnitude, cmap='gray')
plt.title('Resultado Sobel'), plt.axis('off')

plt.subplot(122)
plt.imshow(magnitude_prewitt, cmap='gray')
plt.title('Resultado Prewitt'), plt.axis('off')

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



Ambos producen resultados similares, el filtro de Sobel aplica una mayor ponderación a los píxeles centrales, lo que lo hace más sensible a los cambios de intensidad y le permite resaltar mejor los bordes en condiciones de ruido. Por otro lado, Prewitt utiliza un enfoque más simple, con coeficientes uniformes, lo cual puede generar bordes ligeramente menos definidos. En las imágenes procesadas en este ejemplo es muy complicado encontrar diferencias.

### 2.3.8 8. Suavizado y Sobel (\*): Aplicar un filtro gaussiano antes del operador de Sobel y analizar las diferencias en la detección de bordes.

```
[125]: # Gaussian Filter
blurred = cv2.GaussianBlur(lenna_image,(5,5),1) # Kernel 5x5, sigma 1

# Sobel operator
def sobel_edges(img):
    sobel_x = cv2.Sobel(img,cv2.CV_64F,1,0, ksize=3)
    sobel_y = cv2.Sobel(img,cv2.CV_64F,0,1, ksize=3)
    sobel_combined = cv2.magnitude(sobel_x, sobel_y)
    sobel_combined = cv2.convertScaleAbs(sobel_combined)
    return sobel_combined

sobel_original = sobel_edges(lenna_image)
sobel_blurred = sobel_edges(blurred)

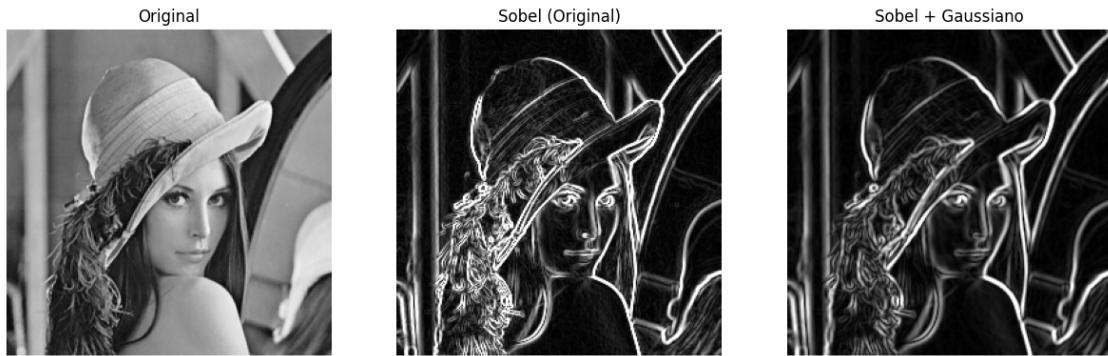
#TODO analizar las diferencias

plt.figure(figsize=(15, 5))
```

```

plt.subplot(131), plt.imshow(lenna_image, cmap='gray'), plt.title('Original'),_
    plt.axis('off')
plt.subplot(132), plt.imshow(sobel_original, cmap='gray'), plt.title('Sobel +_
    (Original)'), plt.axis('off')
plt.subplot(133), plt.imshow(sobel_blurred, cmap='gray'), plt.title('Sobel +_
    Gaussiano'), plt.axis('off')
plt.axis('off')
plt.show()

```



Aplicar un filtro gaussiano antes del operador de Sobel permite reducir el ruido presente en la imagen. En la comparación, se observa que el resultado de aplicar Sobel directamente sobre la imagen original presenta bordes más marcados, pero también puede acentuar el ruido. En cambio, al aplicar primero el suavizado gaussiano, los bordes se ven ligeramente más suaves pero más limpios y definidos, mostrando únicamente las estructuras más relevantes de la imagen.

### 2.3.9 9. Filtro Laplaciano del Gaussiano (LoG): Aplicar un filtro gaussiano seguido de un operador de Laplace para detectar bordes.

```

[124]: blurred = cv2.GaussianBlur(lenna_image,(5,5),1.2)

laplacian = cv2.Laplacian(blurred,cv2.CV_64F)

laplacian_norm = cv2.convertScaleAbs(laplacian)

# Mostrar resultados
plt.figure(figsize=(15, 5))
plt.subplot(141), plt.imshow(lenna_image, cmap='gray'), plt.title('Original'),_
    plt.axis('off')
plt.subplot(142), plt.imshow(blurred, cmap='gray'), plt.title('Gaussiano'), plt.-
    axis('off')
plt.subplot(143), plt.imshow(laplacian_norm, cmap='gray'), plt.title('LoG'),_
    plt.axis('off')

```

```
plt.show()
```



### 2.3.10 10. Filtro de Paso Alto Personalizado: Implementar un filtro de realce de bordes con una matriz de convolución personalizada.

```
[123]: kernel_high = np.array([
    [1,2,1],
    [0,0,0],
    [-1,-2,-1]
])

lenna_filtered = cv2.filter2D(lenna_image,-1, kernel_high)

plt.figure(figsize=(12, 6))
plt.subplot(131), plt.imshow(lenna_image, cmap='gray'), plt.title('Original'), plt.axis('off')
plt.subplot(132), plt.imshow(lenna_filtered, cmap='gray'), plt.title('Paso Alto'), plt.axis('off')
plt.show()
```



### 2.3.11 11. Filtro Canny: Aplicar el detector de bordes de Canny y ajustar los umbrales para obtener diferentes resultados.

```
[122]: edges_canny500_200 = cv2.Canny(lenna_image, threshold1=500, threshold2=200)
edges_canny50_50 = cv2.Canny(lenna_image, threshold1=50, threshold2=50)
edges_canny150_100 = cv2.Canny(lenna_image, threshold1=150, threshold2=100)

plt.figure(figsize=(20, 5))
plt.subplot(1, 4, 1), plt.imshow(lenna_image, cmap='gray'), plt.
    title('Original'), plt.axis('off')
plt.subplot(1, 4, 2), plt.imshow(edges_canny500_200, cmap='gray'), plt.
    title('Bordes Canny 500-200'), plt.axis('off')
plt.subplot(1, 4, 3), plt.imshow(edges_canny50_50, cmap='gray'), plt.
    title('Bordes Canny 50-50'), plt.axis('off')
plt.subplot(1, 4, 4), plt.imshow(edges_canny150_100, cmap='gray'), plt.
    title('Bordes Canny 150-100'), plt.axis('off')
plt.tight_layout()
plt.show()
```



### 2.3.12 12. (\*) Comparación de Métodos de Detección de Bordes : Comparar Sobel, Prewitt, Laplace y Canny trabajando diversas imágenes con características diferentes.

```
[143]: img_clean = data.camera()
img_noisy = data.page()
img_noisy = cv2.add(img_noisy, np.random.normal(0, 25, img_noisy.shape).
                     astype(np.uint8))
img_texture = data.text()

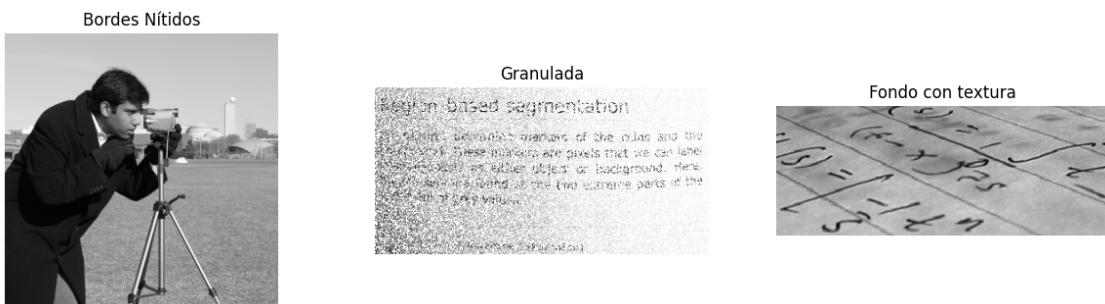
plt.figure(figsize=(15, 8))

plt.subplot(2, 3, 1)
plt.imshow(img_clean, cmap='gray')
plt.title("Bordes Nítidos")
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(img_noisy, cmap='gray')
plt.title("Granulada")
plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(img_texture, cmap='gray')
plt.title("Fondo con textura")
plt.axis('off')
```

[143]: (np.float64(-0.5), np.float64(447.5), np.float64(171.5), np.float64(-0.5))



```
[144]: def sobel_edges(img):
    sobel_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
```

```

    return cv2.magnitude(sobel_x, sobel_y)

def prewitt_edges(img):
    kernel_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
    kernel_y = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])
    prewitt_x = cv2.filter2D(img, cv2.CV_64F, kernel_x)
    prewitt_y = cv2.filter2D(img, cv2.CV_64F, kernel_y)
    return cv2.magnitude(prewitt_x, prewitt_y)

def laplace_edges(img):
    return cv2.Laplacian(img, cv2.CV_64F, ksize=3)

def canny_edges(img, low=150, high=100):
    return cv2.Canny(img, low, high)

```

```
[145]: images = [img_clean, img_noisy, img_texture]
methods = {
    "Sobel": sobel_edges,
    "Prewitt": prewitt_edges,
    "Laplace": laplace_edges,
    "Canny": canny_edges
}

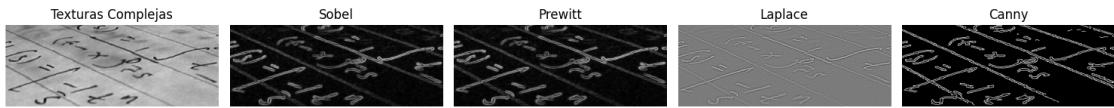
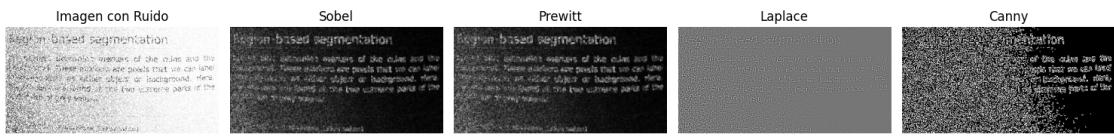
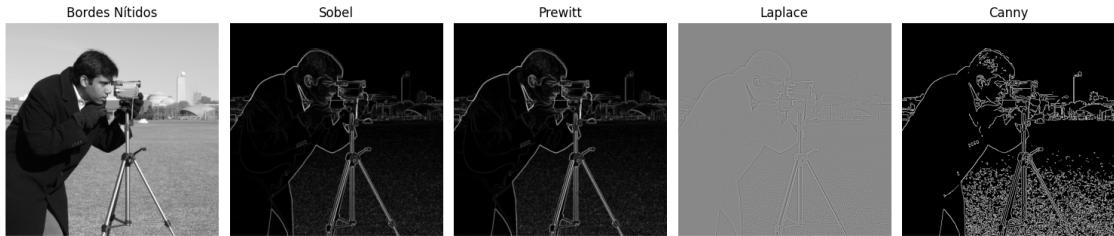
results = {}
for name, method in methods.items():
    results[name] = [method(img) for img in images]
```

```
[146]: titles = ['Bordes Nítidos', 'Imagen con Ruido', 'Texturas Complejas']
plt.figure(figsize=(18, 10))

for i, title in enumerate(titles):
    plt.subplot(3, 6, i*6 + 1)
    plt.imshow(images[i], cmap='gray')
    plt.title(title)
    plt.axis('off')

    for j, (name, edges) in enumerate(results.items()):
        plt.subplot(3, 6, i*6 + j + 2)
        plt.imshow(edges[i], cmap='gray')
        plt.title(name)
        plt.axis('off')

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



### 2.3.13 13. (\*) Realce de Detalles: Aplicar un filtro de paso alto y sumarlo a la imagen original para mejorar los detalles.

```
[119]: kernel_high = np.array([
    [1,2,1],
    [0,0,0],
    [-1,-2,-1]
])

lenna_filtered = cv2.filter2D(lenna_image,-1, kernel_high)

lenna_enhanced = cv2.addWeighted(lenna_image,0.8,lenna_filtered,0.2,0)

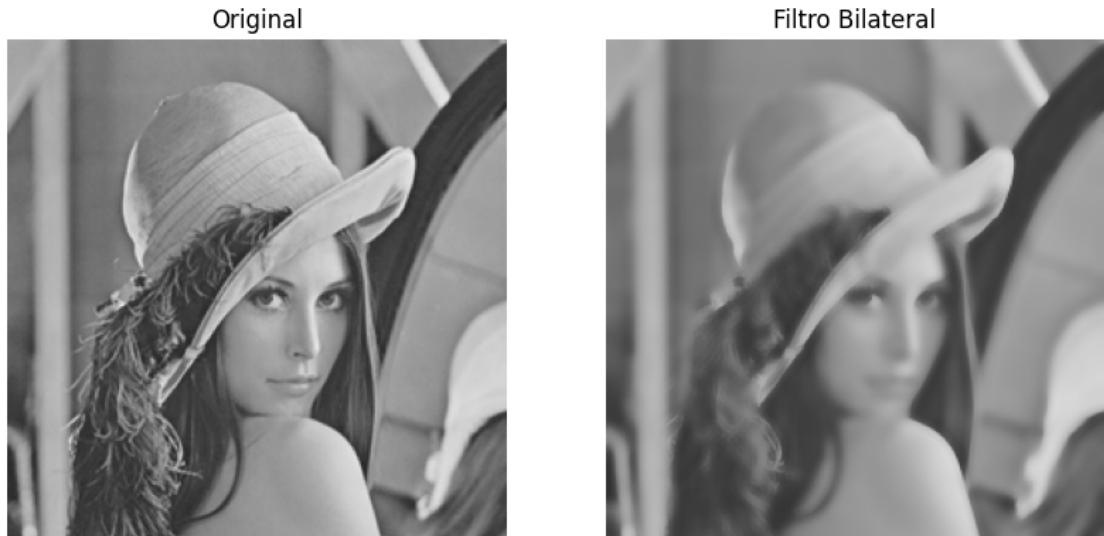
plt.figure(figsize=(12, 6))
plt.subplot(131), plt.imshow(lenna_image, cmap='gray'), plt.title('Original'), plt.axis('off')
plt.subplot(132), plt.imshow(lenna_filtered, cmap='gray'), plt.title('Paso Alto'), plt.axis('off')
plt.subplot(133), plt.imshow(lenna_enhanced, cmap='gray'), plt.title('Original + Paso Alto'), plt.axis('off')
plt.show()
```



### 2.3.14 14. Reducción de Ruido con Bilateral Filter: Aplicar un filtro bilateral para suavizar la imagen sin perder detalles importantes.

```
[120]: filtered = cv2.bilateralFilter(lenna_image, d=9, sigmaColor=75, sigmaSpace=75)

plt.figure(figsize=(10, 5))
plt.subplot(121), plt.imshow(cv2.cvtColor(lenna_image, cv2.COLOR_BGR2RGB)), plt.
    title('Original'), plt.axis('off')
plt.subplot(122), plt.imshow(cv2.cvtColor(filtered, cv2.COLOR_BGR2RGB)), plt.
    title('Filtro Bilateral'), plt.axis('off')
plt.show()
```



### 2.3.15 15. (\*) Filtro de Diferencia Gaussiana (DoG): Aplicar la técnica de Diferencia de Gaussiana para resaltar bordes.

```
[ ]: blur1 = cv2.GaussianBlur(lenna_image, (5, 5), sigmaX=1.0) # Small sigma, thin borders
blur2 = cv2.GaussianBlur(lenna_image, (5, 5), sigmaX=3.0) # Large sigma, thick borders

dog = blur1 - blur2

dog_normalized = cv2.normalize(dog, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)

plt.figure(figsize=(15, 5))
plt.subplot(141), plt.imshow(lenna_image, cmap='gray'), plt.title('Original'), plt.axis('off')
plt.subplot(142), plt.imshow(blur1, cmap='gray'), plt.title(r'Gaussiano $\sigma=1.0$'), plt.axis('off')
plt.subplot(143), plt.imshow(blur2, cmap='gray'), plt.title(r'Gaussiano $\sigma=3.0$'), plt.axis('off')
plt.subplot(144), plt.imshow(dog_normalized, cmap='gray'), plt.title('DoG'), plt.axis('off')
plt.show()
```

