

# Procesamiento de Imágenes

## Trabajo Práctico 2

Francisco Devaux

Agustín Yornet de Rosas

2 de Mayo, 2025

## Índice

<b>Introducción</b>	<b>1</b>
<b>1. Histogramas</b>	<b>2</b>
<b>2 Combinación de Imágenes</b>	<b>6</b>
<b>3 Filtrado Espacial</b>	<b>9</b>

## Introducción

Este documento corresponde al segundo trabajo práctico de **Procesamiento de Imágenes**. En la siguiente página, se detallan los puntos a desarrollar.

## 1 Histogramas

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

**Rta.** Luego de transformar la distribución de intensidades de `paisaje1.jpg` para que se parezca a la de `paisaje4.jpg` por medio de `skimage.exposure.match_histograms()`, es posible ver que los histogramas, luego del ajuste, presentan mayor cantidad de similitudes en el rango de valores entre 0 y 110 (aproximadamente) con pequeñas oscilaciones. Para valores superiores a 110, la versión ajustada de `paisaje1.jpg` contiene frecuencias muy altas para valores específicos, cercanos a 110 y 190.

Lo discutido anteriormente puede verse en la **Figura 1**.

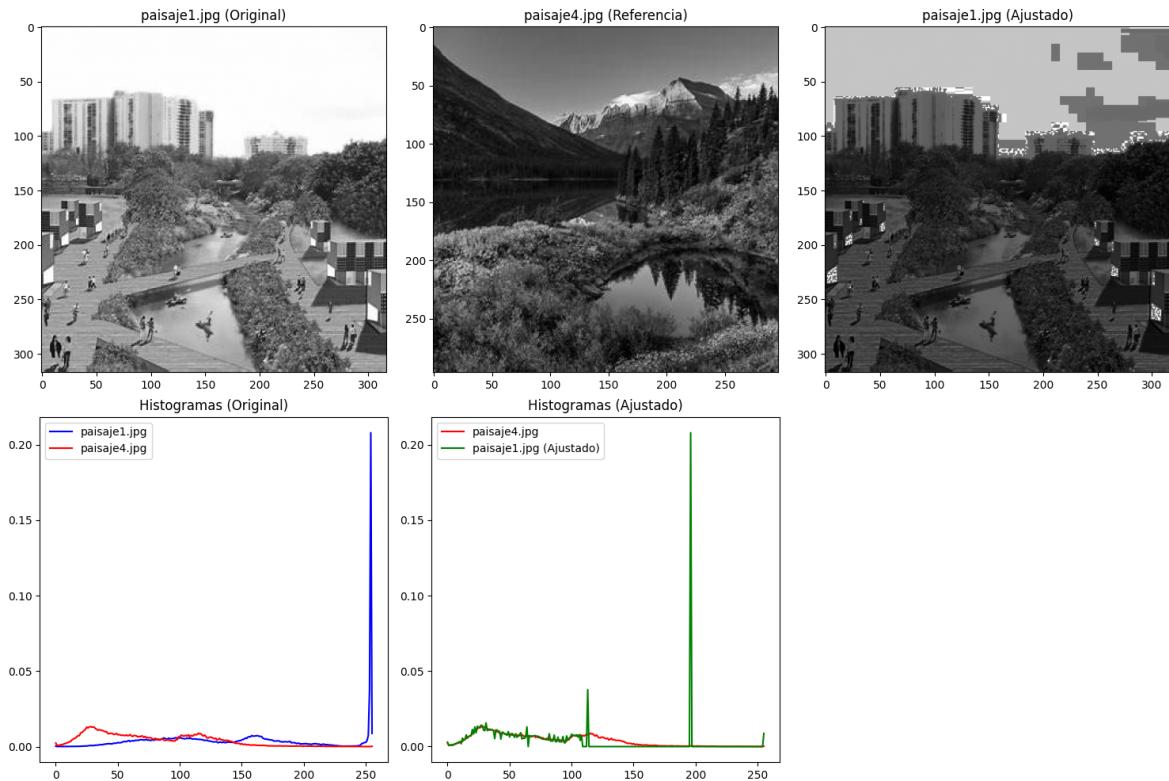


Figura 1: Comparación entre la versión original y la versión con intensidades ajustadas de la imagen `paisaje1.jpg`, y sus histogramas correspondientes.

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

**Rta.** Al comparar las imágenes y sus histogramas, es posible ver en la **Figura 2** que el contraste de la imagen original ha aumentado de forma notable en su versión ecualizada. Como es posible ver en el histograma de la versión ecualizada, las intensidades se distribuyen de forma más equitativa, aumentando las frecuencias de los valores más altos que eran menos frecuentes en la imagen original.

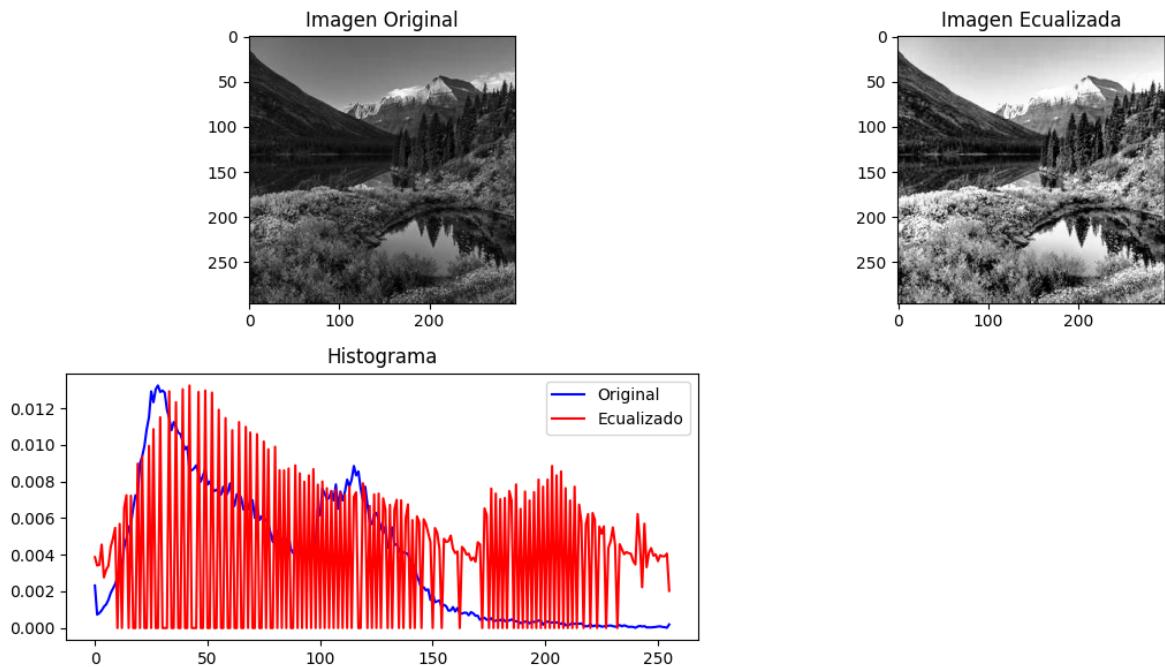


Figura 2: Versión Original vs. Ecualizada de paisaje4.jpg, y sus histogramas correspondientes.

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

**Rta.** El resultado de aplicar el método maual y el método de Otsu puede observarse en la **Figura 3**.

Fragmento extraido de [docs.opencv.org]:

“Con la umbralización global, usamos un valor elegido arbitrariamente como umbral. En cambio, el método de Otsu evita tener que elegir un valor y lo determina automáticamente.

Considera una imagen con solo dos valores de intensidad distintos (una imagen bimodal), donde el histograma consistiría únicamente en dos picos. Un buen umbral estaría en el medio de esos dos valores. De manera similar, el método de Otsu determina un valor de umbral global óptimo a partir del histograma de la imagen.

Para hacerlo, se utiliza la función `cv.threshold()`, donde se pasa `cv.THRESH_OTSU` como una bandera adicional. El valor del umbral puede elegirse arbitrariamente. Luego, el algoritmo encuentra el valor de umbral óptimo, el cual se devuelve como la primera salida.”

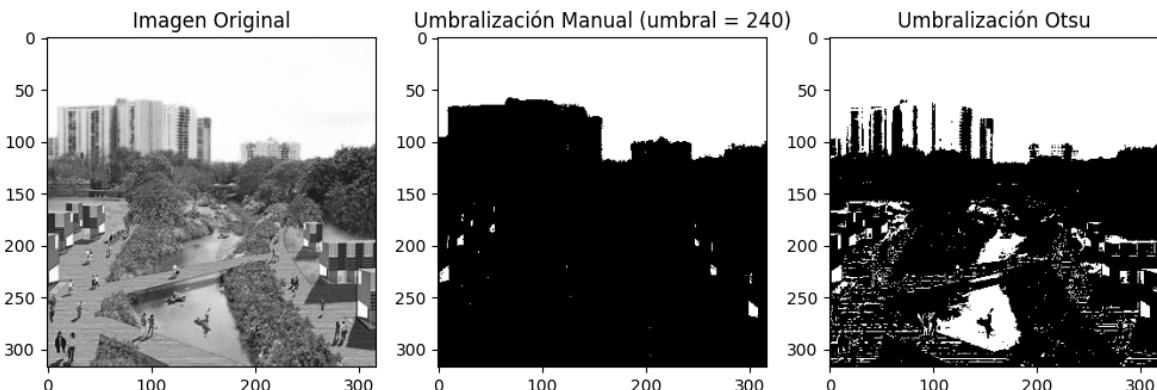


Figura 3: Comparación entre la versión original, la versión umbralizada manualmente (con valor de umbral igual a 240) y la versión umbralizada con método OTSU de la imagen `paisaje3.jpg`.

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.

**Rta.** El resultado de aplicar la transformación gamma a distintas regiones de un imagen puede observarse en la **Figura 4**.

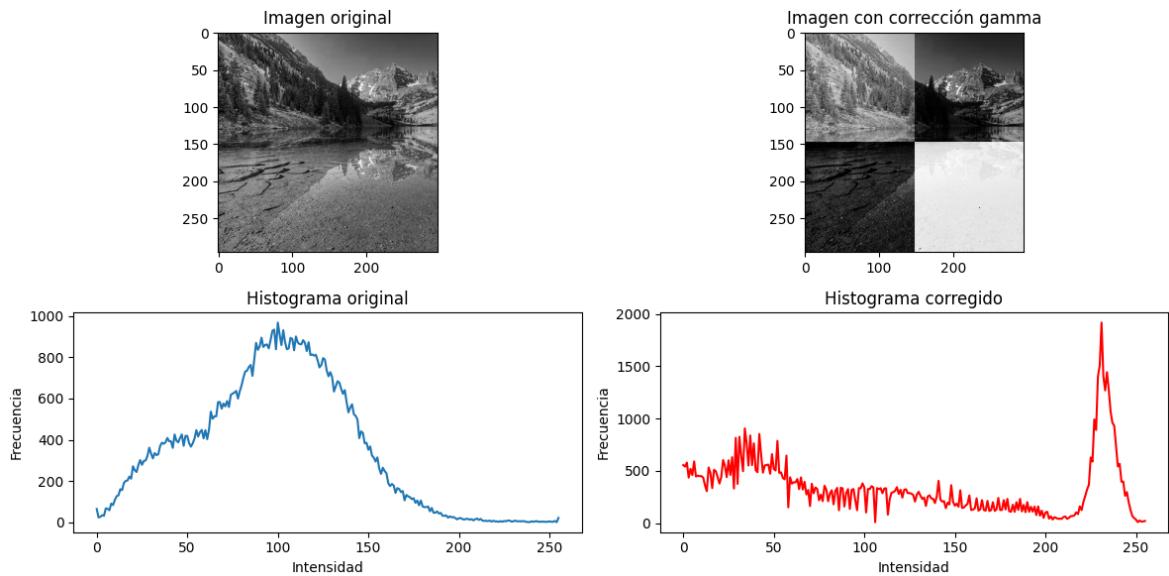


Figura 4: Comparación entre la versión original y la versión con transformaciones gamma aplicadas en distintas regiones de la imagen `paisaje1.jpg`, junto a sus respectivos histogramas.

## 2 Combinación de Imágenes

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

**Rta.** El resultado de multiplicar y dividir dos imágenes píxel a píxel puede observarse en la **Figura 5**.

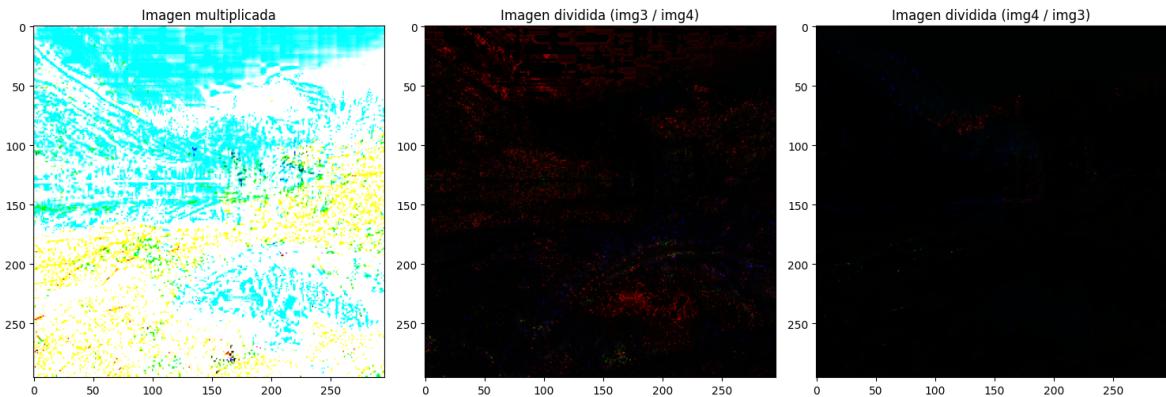


Figura 5: Comparación entre las imágenes resultantes de la multiplicación y las divisiones, en ambos sentidos, de las imágenes `paisaje3.jpg` y `paisaje4.jpg`.

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.

**Rta.** En los tres casos, cuando se provee la máscara circular, las operaciones AND, OR y XOR se realizan entre las dos imágenes sólo donde los valores de la máscara no son iguales a cero, es decir, en el interior del círculo.

Las operaciones bitwise AND, OR y XOR en imágenes BGR (Blue, Green, Red) funcionan canal por canal y pixel por pixel, comparando directamente los valores binarios de cada componente del color. Como los valores están en el rango [0, 255], cada componente tiene una representación binaria de 8 bits.

Viendo nuestras imágenes, podemos ver que aplicar el operador AND resulta en una imagen oscura porque el operador sólo mantiene un bit en 1 si los bits también son 1, y cero en caso contrario. Por ejemplo:

$$\begin{aligned} pixel_{img3}(x, y) &= 11001010 \\ pixel_{img4}(x, y) &= 10101100 \\ resultado &= 10001000 \end{aligned}$$

Aplicar el operador OR, por otro lado, resulta en una imagen brillante, ya que el

operador mantiene un bit en 1 si al menos uno de los bits es 1. Por ejemplo:

$$\begin{aligned} pixel_{img3}(x, y) &= 11001010 \\ pixel_{img4}(x, y) &= 10101100 \\ resultado &= 11101110 \end{aligned}$$

Aplicar el operador XOR, finalmente, resalta las diferencias entre las dos imágenes, ya que el operador mantiene un bit en 1 si los bits son diferentes. Por ejemplo:

$$\begin{aligned} pixel_{img3}(x, y) &= 11001010 \\ pixel_{img4}(x, y) &= 10101100 \\ resultado &= 01100110 \end{aligned}$$

Todo lo discutido anteriormente es observable en la **Figura 6** y en la **Figura 7**.

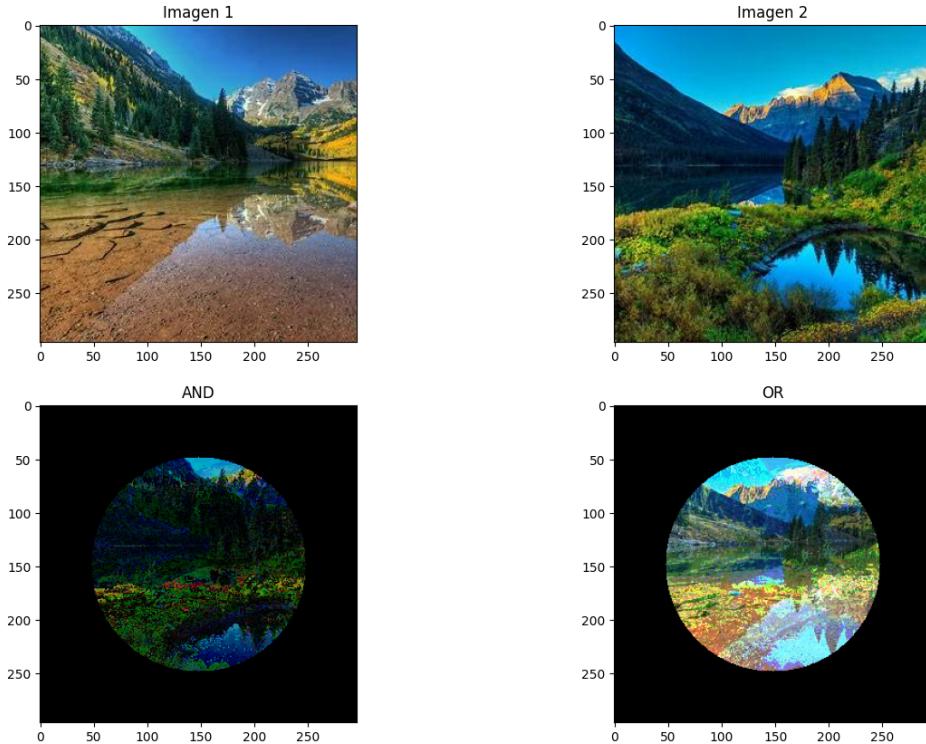


Figura 6: **Imagen Superior Izquierda:** paisaje3.jpg. // **Imagen Superior Derecha:** paisaje4.jpg. // **Imagen Inferior Izquierda:** Resultado de aplicar la operación `bitwise_and` sobre las imágenes paisaje3.jpg y paisaje4.jpg con una máscara binaria `cv2.circle`. // **Imagen Inferior Derecha:** Resultado de aplicar la operación `bitwise_or` sobre las imágenes paisaje3.jpg y paisaje4.jpg con una máscara binaria `cv2.circle`.

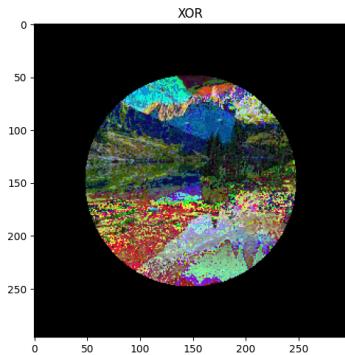


Figura 7: Resultado de aplicar la operación `bitwise_xor` sobre las imágenes `paisaje3.jpg` y `paisaje4.jpg` con una máscara binaria `cv2.circle`.

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).

**Rta.** El resultado de reemplazar `mariposa.png` en `paisaje3.jpg` se puede observar en la **Figura 8**.

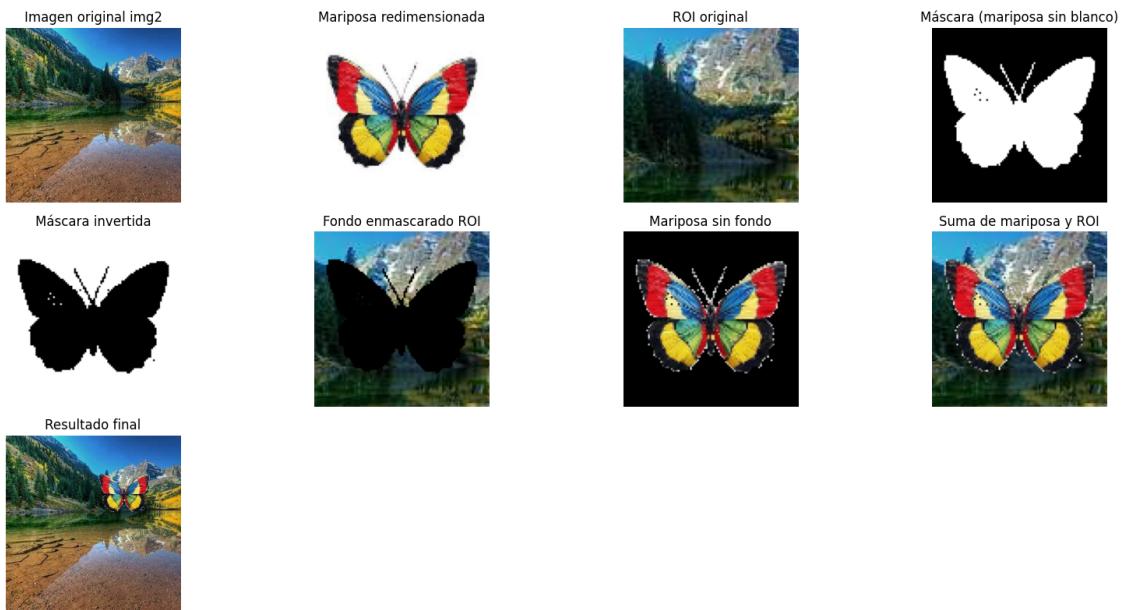


Figura 8: Proceso de reemplazar `mariposa.png` sobre la región de interés de `paisaje3.jpg`. Las imágenes detallan el proceso realizado, en orden de izquierda a derecha, y de arriba a abajo.

### 3 Filtrado Espacial

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

**Rta.** Luego de aplicar Suavizado, Sobel sin suavizar y Sobel con Suavizado, es posible observar que suavizar antes de aplicar Sobel mejora la calidad de la detección de bordes, reduciendo respuestas a detalles irrelevantes o ruido, como puede observarse en la **Figura 9.**

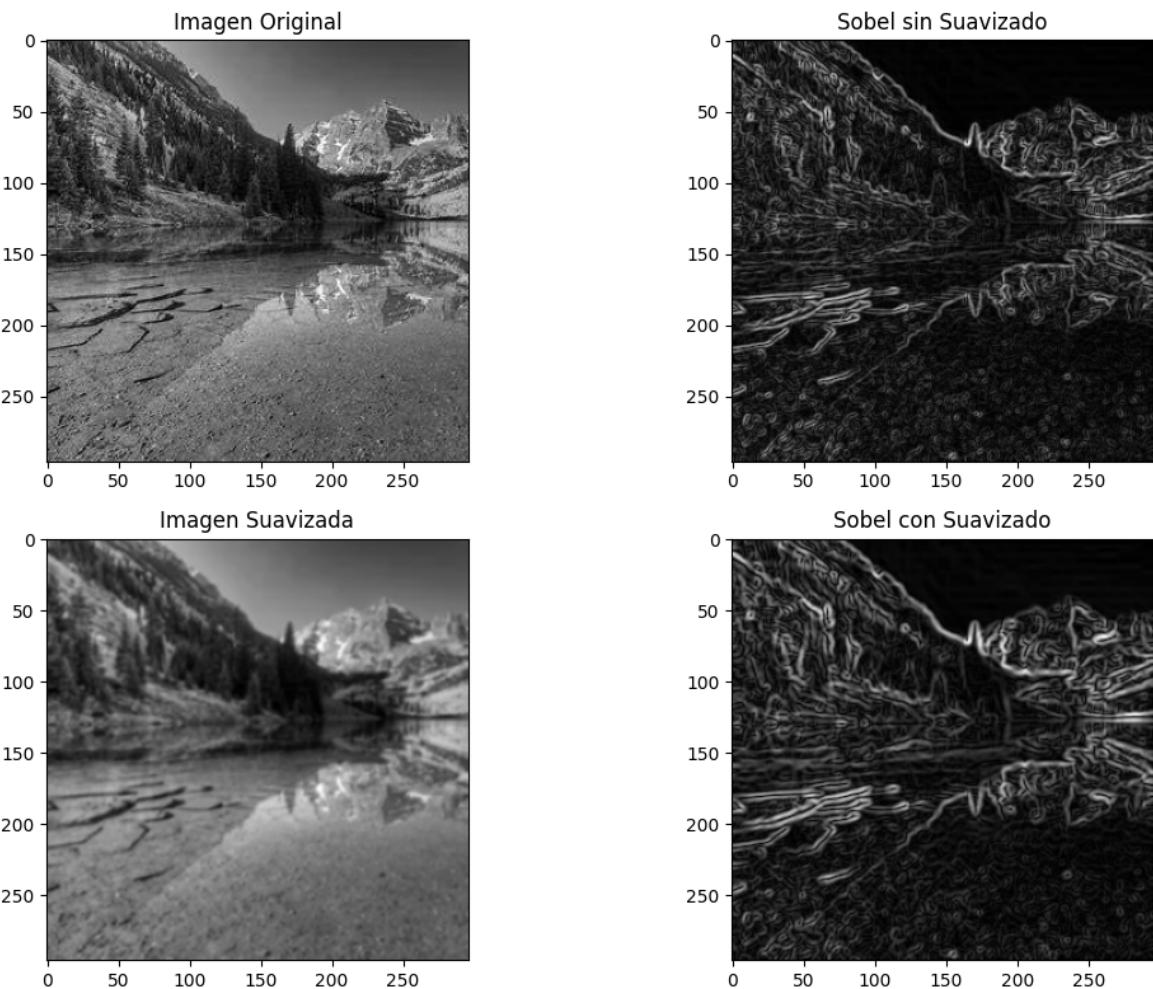


Figura 9: **Arriba a la izquierda:** imagen original en escala de grises. **Abajo a la izquierda:** imagen suavizada mediante filtrado. **Arriba a la derecha:** bordes detectados con Sobel sin suavizado previo, mostrando mayor ruido y detección de detalles menores. **Abajo a la derecha:** bordes detectados con Sobel tras suavizado, destacando estructuras principales con menor ruido.

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

**Rta.** Luego de aplicar los distintos métodos de detección de bordes, es posible observar en las **Figuras 10, 11 y 12** que el método de Canny es el método que mejor funciona para detectar bordes.

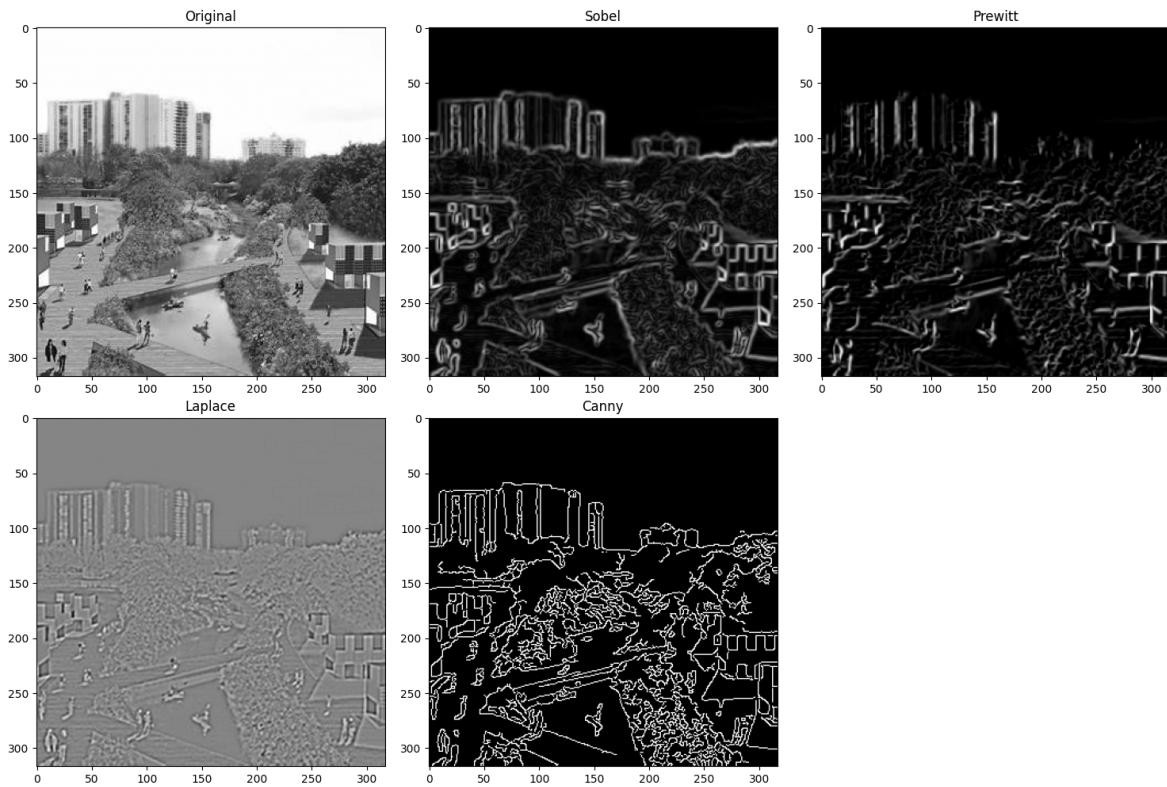


Figura 10: Comparación de Métodos de Detección de Bordes sobre `paisaje1.jpg`. De izquierda a derecha: Original - Sobel - Prewitt - Laplace - Canny.

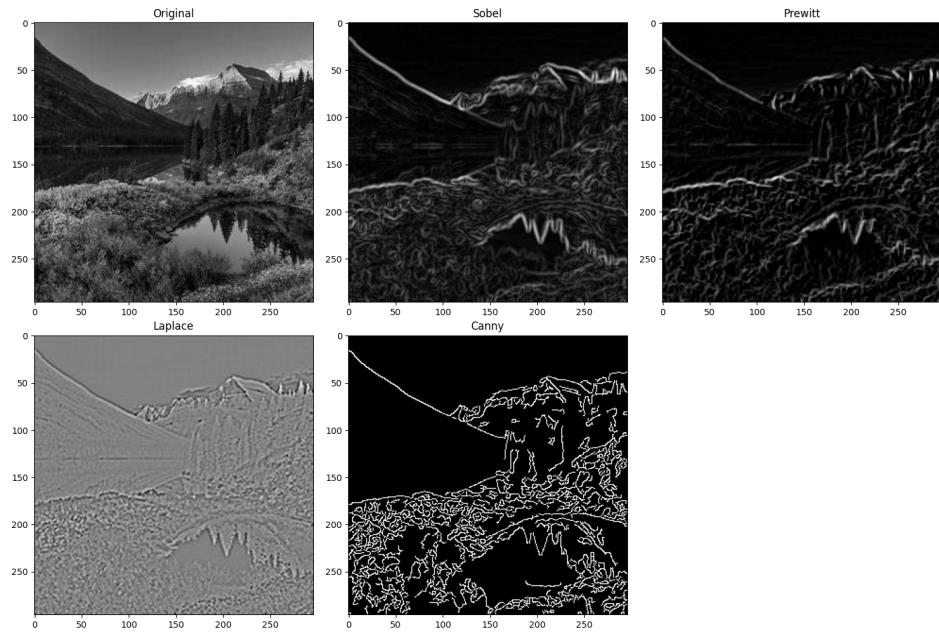


Figura 11: Comparación de Métodos de Detección de Bordes sobre `paisaje4.jpg`. De izquierda a derecha: Original - Sobel - Prewitt - Laplace - Canny.

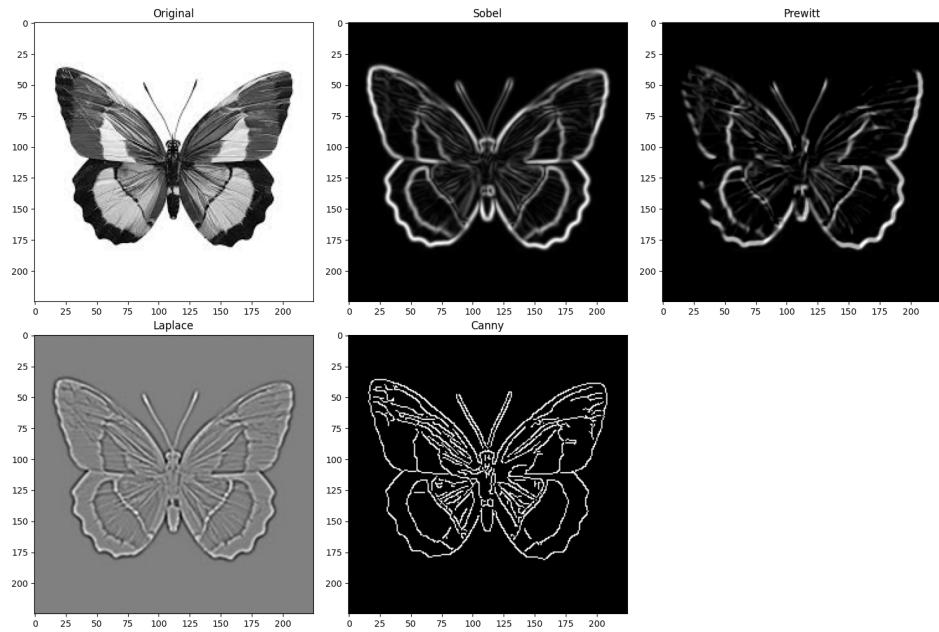


Figura 12: Comparación de Métodos de Detección de Bordes sobre `mariposa.png`. De izquierda a derecha: Original - Sobel - Prewitt - Laplace - Canny.

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

**Rta.** El resultado de aplicar un filtro de paso alto y sumarlo a `paisaje1.jpg` puede observarse en la **Figura 13**.

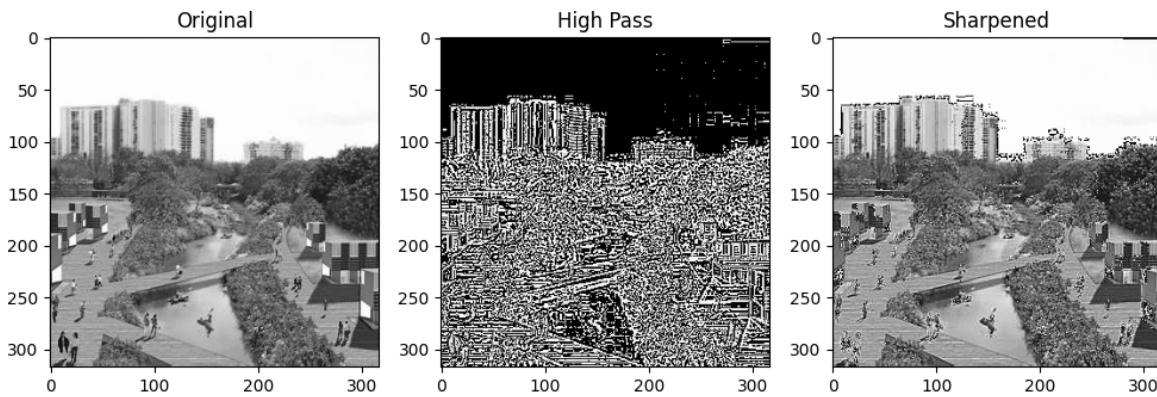


Figura 13: **Izq.** Imagen original. // **Centro.** Resta entre imagen original e imagen con Blur Gaussiano (filtro de paso alto). // **Der.** Suma entre imagen con filtro de paso alto e imagen original.

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

**Rta.** El resultado de aplicar el método de Diferencia de Gaussianos sobre `mariposa.jpg` puede verse en la **Figura 14**.

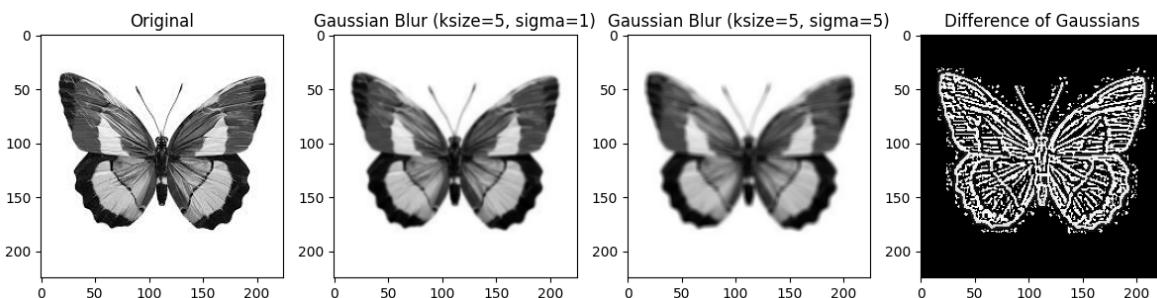


Figura 14: **De Izquierda a Derecha:** Imagen Original; Imagen con Gaussian Blur y  $\sigma = 1$ ; Imagen con Gaussian Blur y  $\sigma = 5$ ; Diferencia de Gaussianos.