

**INSTITUTO POLITÉCNICO NACIONAL
Escuela Superior de Cómputo
ESCOM**



Actividad 6:

- ❖ Operaciones relacionales y lógicas
- ❖ Etiquetado de componentes conexas

Realización: 08/10/2025

Entrega: 17/10/2025

Presentan:

- ❖ García Cerón Diego

Grupo: 7CV4

Materia:

VISIÓN POR COMPUTADORA | IMAGE ANALYSIS

Profesora:

MARIA ELENA CRUZ MEZA

Contenido

INTRODUCCIÓN.....	1
OBJETIVOS	1
COMPETENCIAS.....	1
MARCO TEÓRICO	1
Operaciones lógicas en imágenes digitales	1
AND (Intersección)	2
OR (Unión).....	2
XOR(Exclusivo)	2
NOT(Negación).....	3
Operaciones relacionales	3
Vecindad en imágenes digitales	4
Vecindad 4	4
Vecindad 8	5
DESARROLLO.....	6
PARTE A: OPERACIONES RELACIONALES Y LÓGICAS.....	6
PARTE A: APLICACIÓN DE OPERACIONES LÓGICAS DESDE EL SIMULADOR (PRUEBAS PREVIAS)	6
PARTE A: APLICACIÓN DE OPERACIONES LÓGICAS (CODIO PROPIO PYTHON)	9
PARTE B: OPERACIONES RELACIONALES	14
CUESTIONARIO PARTE A	17
PARTE B. ETIQUETADO DE COMPONENTES CONEXAS	18

INTRODUCCIÓN

En esta práctica se desarrolla una aplicación interactiva en Python utilizando OpenCV, Tkinter y SciPy, cuyo objetivo es analizar el concepto de conectividad en imágenes binarias, comparando los resultados obtenidos con vecindad 4 y vecindad 8.

El sistema permite al usuario cargar una imagen, binarizarla mediante umbral fijo o automático (Otsu), y posteriormente etiquetar los componentes conectados para visualizar cuántos objetos independientes se detectan en cada tipo de conectividad. Esta comparación evidencia cómo la definición de “vecino” (considerando o no las diagonales) influye directamente en el número y forma de los objetos detectados.

El proceso completo se apoya en una interfaz gráfica intuitiva que guía al usuario a través del flujo Cargar → Binarizar → Etiquetar → Mostrar, mostrando los resultados de forma visual y cuantitativa. De esta manera, la práctica permite comprender de manera práctica y experimental los fundamentos del análisis de conectividad en el procesamiento digital de imágenes.

OBJETIVOS

- ❖ Aplicar transformaciones lógicas (AND, OR, XOR, NOT) y operaciones relacionales en imágenes digitales para modificar sus características y extraer información relevante.

COMPETENCIAS

- ❖ Comprender el sistema de visión humana y modelos de color.
- ❖ Manipular imágenes digitales en Python.
- ❖ Colaborar en equipos para resolver problemas técnicos.

MARCO TEÓRICO

Operaciones lógicas en imágenes digitales

La aritmética de imágenes digitales consiste en aplicar operaciones matemáticas píxel a píxel entre dos o más imágenes, o entre una imagen y un valor constante. Cada píxel se trata como un número (intensidad o color), y las operaciones se aplican sobre estos valores. Estas operaciones se utilizan para realzar imágenes, combinar información, o detectar cambios entre fotogramas.

Las principales son:

- Suma: combina el brillo de dos imágenes o aumenta la luminosidad.
Ejemplo: $R(x, y) = A(x, y) + B(x, y)$
- Resta: permite detectar diferencias entre imágenes o resaltar bordes.
Ejemplo: $R(x, y) = A(x, y) - B(x, y)$
- Multiplicación: usada para ajustar el contraste o realizar enmascaramientos.
- División: puede corregir iluminación o normalizar intensidades.

Estas operaciones tratan a los píxeles como valores binarios (0 = negro, 1 = blanco), especialmente útiles en imágenes binarias (blanco y negro puro) y en procesamiento morfológico.

AND (Intersección)

Combina dos imágenes y mantiene el píxel en 1 solo si ambos son 1.

- ❖ Uso: para encontrar regiones comunes entre dos máscaras.
- ❖ Ejemplo: $R(x, y) = A(x, y) \text{ AND } B(x, y)$

A	B	AND ($A \wedge B$)
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 1. Tabla de verdad AND

OR (Unión)

El píxel resulta 1 si al menos una de las imágenes tiene un 1.

- ❖ Uso: para combinar o superponer áreas de interés.
- ❖ Ejemplo: $R(x, y) = A(x, y) \text{ OR } B(x, y)$

A	B	OR ($A \vee B$)
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 2. Tabla de verdad OR

XOR(Exclusivo)

El píxel resulta 1 si las imágenes difieren (uno es 1 y el otro 0).

- ❖ Uso: para detectar diferencias entre imágenes o bordes.
- ❖ Ejemplo: $R(x, y) = A(x, y) \text{ XOR } B(x, y)$

A	B	XOR ($A \oplus B$)
0	0	0

0	1	1
1	0	1
1	1	0

Tabla 3. Tabla de verdad XOR

NOT(Negación)

Invierte los valores de la imagen: los 1 se vuelven 0 y viceversa.

- ❖ Uso: para invertir una máscara o crear el complemento de una región.
- ❖ Ejemplo: $R(x, y) = \text{NOT } A(x, y)$

A	B	NOT A ($\neg A$)
0	0	1
0	1	1
1	0	0
1	1	0

Tabla 4. Tabla de verdad NOT

Operaciones relacionales

Estas operaciones comparan los valores de píxeles entre dos imágenes o con un valor constante. El resultado es una imagen binaria, donde:

- ❖ El píxel es 1 (blanco) si la condición es verdadera.
- ❖ El píxel es 0 (negro) si la condición es falsa.

Ejemplos:

- ❖ $R(x, y) = A(x, y) > B(x, y)$
- ❖ $R(x, y) = A(x, y) == 128$

Usos: detección de umbrales, segmentación y comparación entre regiones.

A	B	$A > B$	$A < B$	$A = B$	$A \geq B$	$A \leq B$	$A \neq B$
50	50	0	0	1	1	1	0
50	100	0	1	0	0	1	1
100	50	1	0	0	1	0	1

Tabla 5. Tabla de verdad operaciones relacionales

Vecindad en imágenes digitales

En el procesamiento digital de imágenes, el concepto de vecindad se utiliza para definir la relación espacial entre un píxel y los que lo rodean.

Cada píxel se localiza mediante sus coordenadas (x, y) , y la vecindad determina qué píxeles cercanos se consideran conectados o influenciados por él.

Este concepto es fundamental en operaciones como:

- Filtrado espacial (suavizado, detección de bordes).
- Segmentación y etiquetado de regiones.
- Morfología matemática.
- Detección de conectividad (componentes conectados).

Existen principalmente dos tipos de vecindad: la vecindad 4 y la vecindad 8.

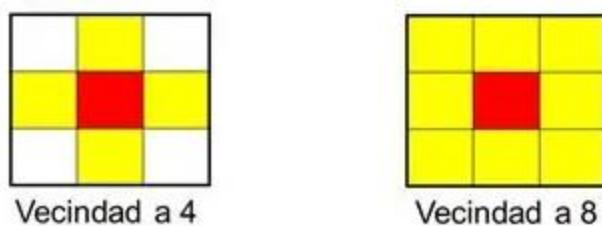


Figura 1. Vecindad 4 y 8

Vecindad 4

La vecindad 4 considera solo los píxeles adyacentes en sentido horizontal y vertical al píxel central. Es decir, los vecinos se encuentran arriba, abajo, a la izquierda y a la derecha.

$$\begin{array}{ccc} (x, y - 1) & & \\ (x - 1, y) & (x, y) & (x + 1, y) \\ & (x, y + 1) & \end{array}$$

- Número de vecinos: 4
- Uso: operaciones que consideran conectividad ortogonal.

Usos recomendados

- Análisis de conectividad ortogonal, donde se requiere que las regiones estén unidas solo por lados, no por esquinas.
- Identificación de bordes o contornos sin que se confundan píxeles diagonales.

- Operaciones morfológicas en las que se busca evitar uniones diagonales no deseadas entre regiones.
- Etiquetado de componentes conectados (4-conectividad) cuando se desea una interpretación más estricta de “contacto”.

No se recomienda

- Cuando se desea detectar regiones que realmente están conectadas en diagonal, ya que podría fragmentar una misma zona en varias.
- En operaciones donde la continuidad visual (por ejemplo, una línea diagonal) es importante.

Vecindad 8

Incluye los píxeles adyacentes en todas las direcciones (horizontales, verticales y diagonales).

$$\begin{array}{ccc} (x - 1, y - 1) & (x, y - 1) & (x + 1, y - 1) \\ (x - 1, y) & (x, y) & (x + 1, y) \\ (x - 1, y + 1) & (x, y + 1) & (x + 1, y + 1) \end{array}$$

- Número de vecinos: 8
- Uso: operaciones que consideran conectividad diagonal y total.

Usos recomendados

- Análisis de conectividad completa, donde se considera que los píxeles están conectados incluso si lo están por una esquina.
- Segmentación de regiones continuas o diagonales, como bordes inclinados o formas redondeadas.
- Operaciones morfológicas donde se desea una conexión más “suave” entre regiones vecinas.
- Algoritmos de crecimiento de regiones (region growing) y relleno de áreas (flood fill).

No se recomienda

- En análisis donde la precisión de los bordes es crítica, ya que puede generar conexiones falsas entre regiones separadas por una esquina.
- Cuando se requiere evitar ambigüedad en la conectividad (por ejemplo, si se usa 8-conectividad en el objeto, se recomienda usar 4-conectividad en el fondo para evitar superposición).

DESARROLLO

PARTE A: OPERACIONES RELACIONALES Y LÓGICAS

PARTE A: APLICACIÓN DE OPERACIONES LÓGICAS DESDE EL SIMULADOR (PRUEBAS PREVIAS)

Obtuvimos 2 imágenes de internet, para hacer las pruebas y usamos el simulador para poder hacer una vista previa de las operaciones aritméticas y el redimensionamiento de imágenes

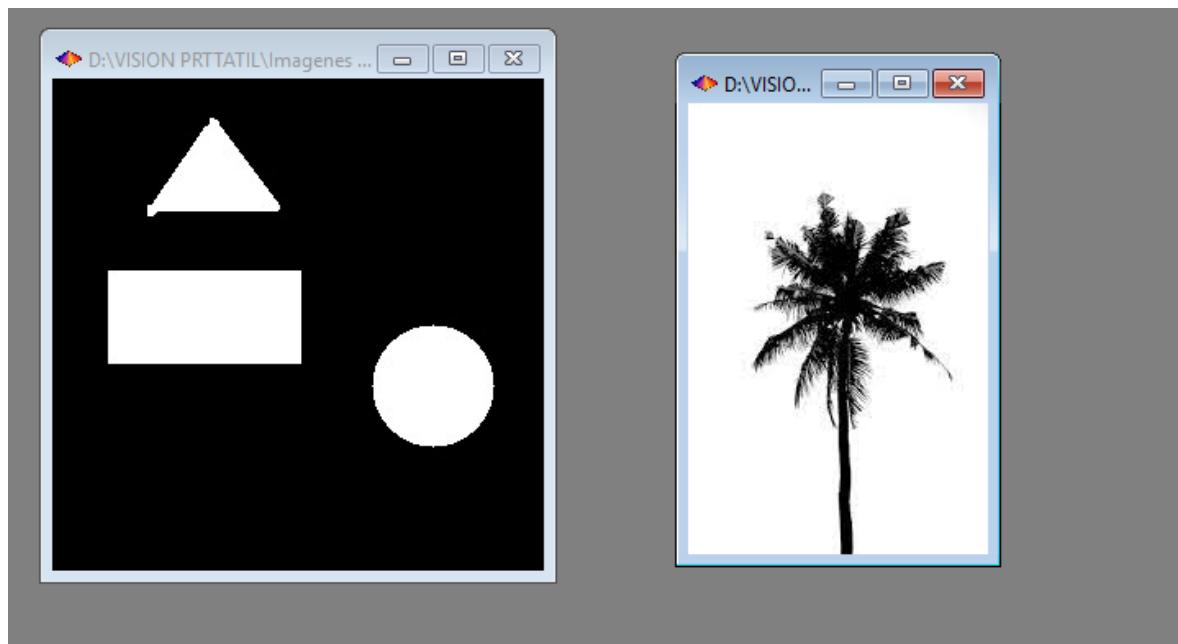


Figura 1. Imágenes de prueba principales

Para poder aplicar las lógicas debemos tener el mismo tamaño de imágenes así que aplicamos dimensionamiento de 300 x300:

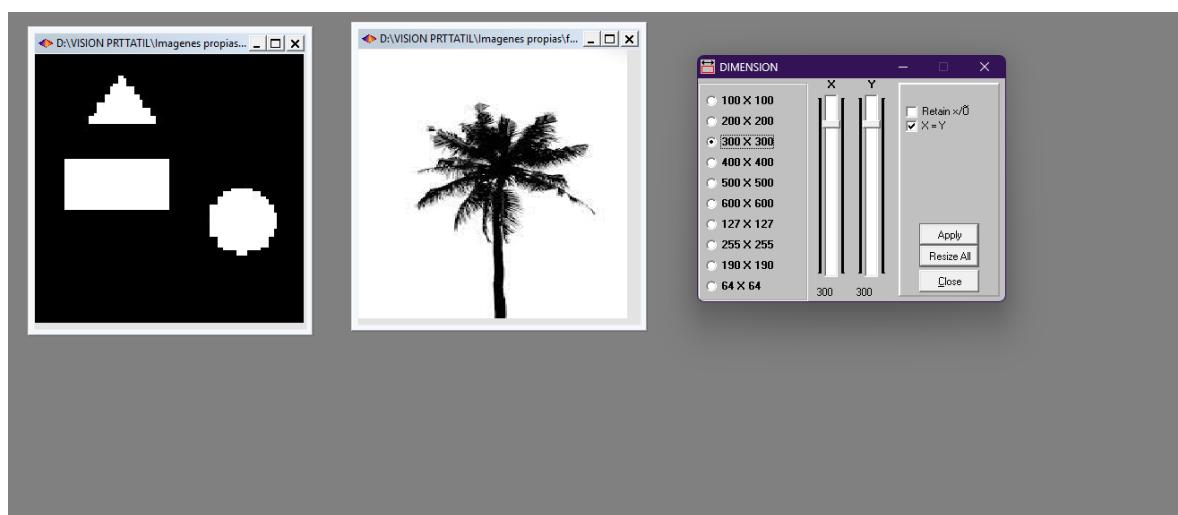


Figura 2. Redimensionamiento en el simulador

Las convertimos a binarias con gray y otsu:

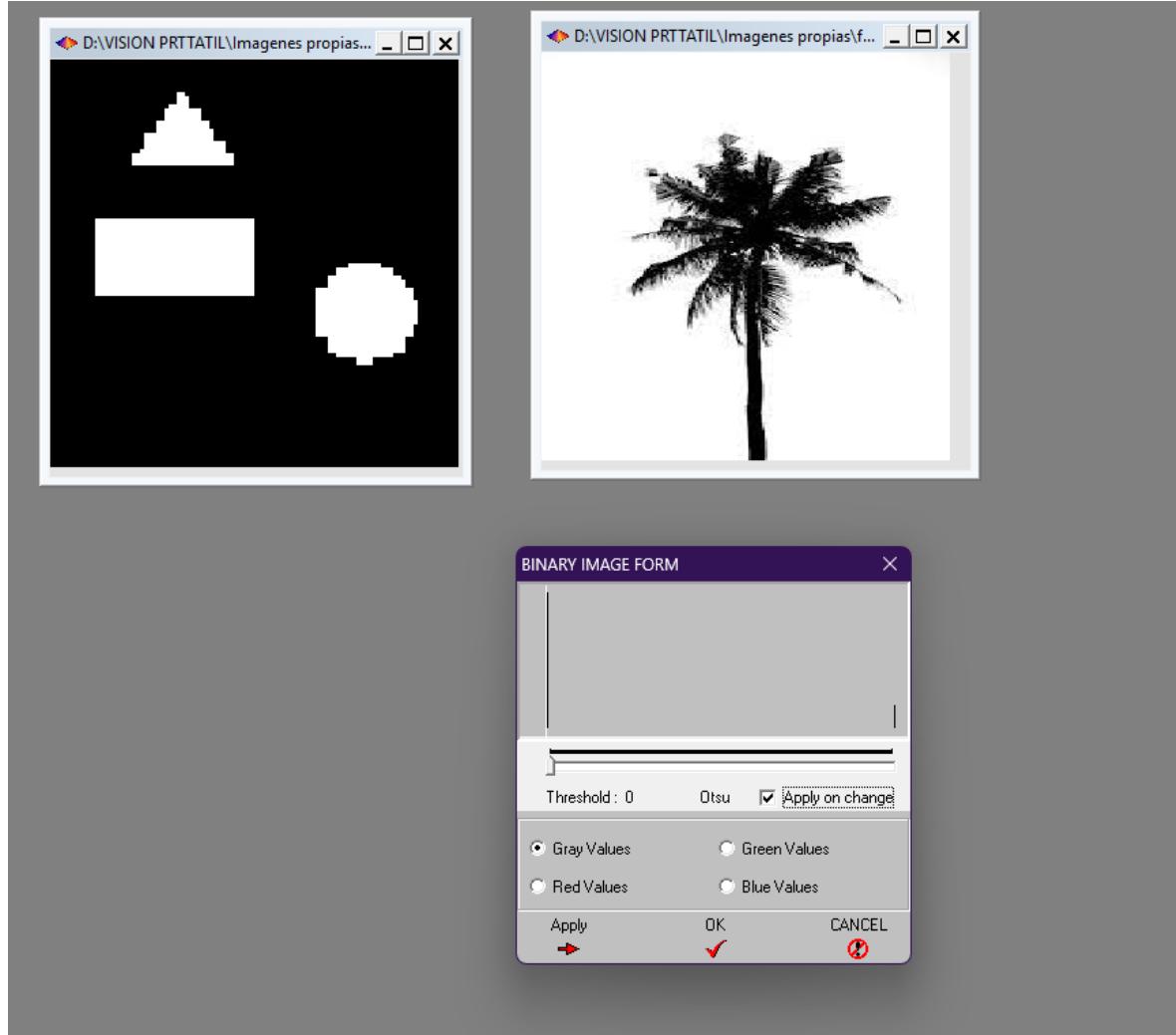


Figura 3. Binarizado desde el simulador

Aplicación de operaciones lógicas a las imágenes binarias:

Operación	Resultado	Interpretación visual
AND	El píxel es blanco solo si ambas imágenes tienen blanco.	Resalta la intersección entre las figuras.
OR	El píxel es blanco si alguna imagen tiene blanco.	Une las figuras de ambas imágenes.
XOR	El píxel es blanco si solo una imagen tiene blanco (pero no las dos).	Muestra las diferencias entre las imágenes.

NOT	Invierte la imagen (blanco ↔ negro).	Crea un negativo digital de una sola imagen.
------------	--------------------------------------	--

Tabla 6. Resultado esperado

Comparación entre operaciones desde el simulador antes de recurrir al Código:

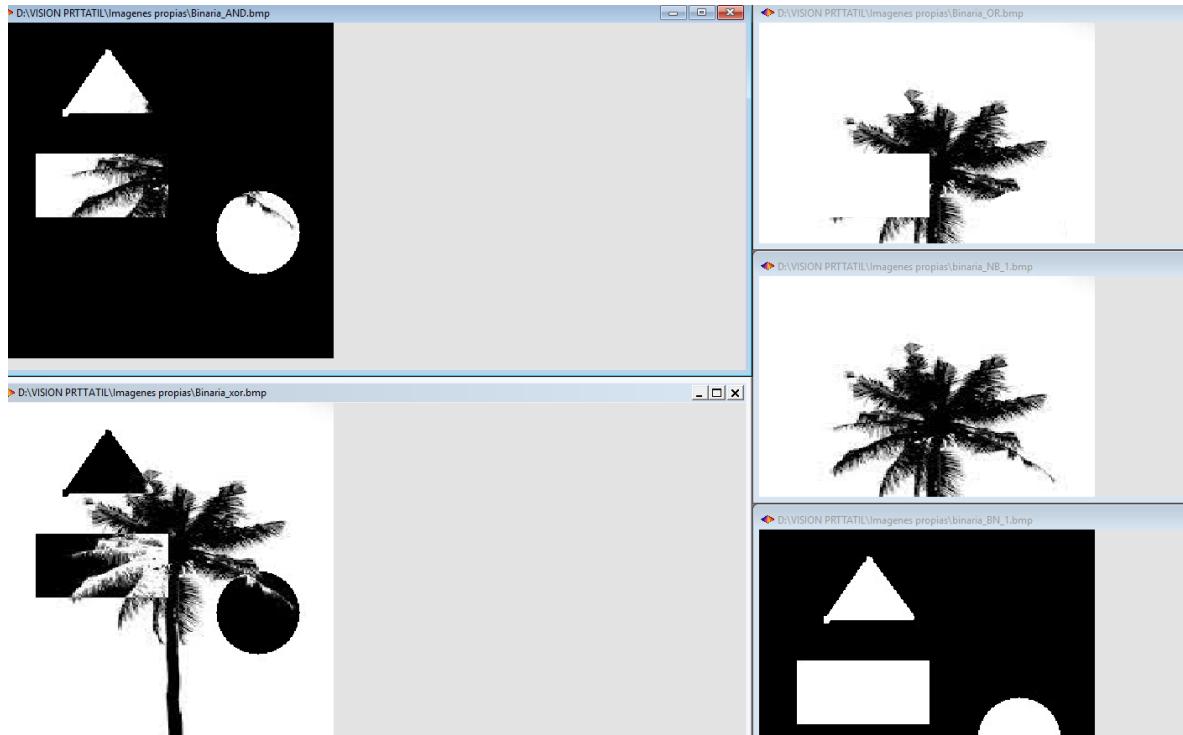


Figura 4. Resultados en el simulador

Operación	Descripción	Resultado observado	Interpretación / Conclusión
OR (O lógica)	Combina los píxeles blancos de ambas imágenes; si alguna tiene valor 1 (blanco), el resultado será blanco.	Se observa la unión de la palmera y las figuras geométricas; todo lo que era blanco en cualquiera de las imágenes permanece visible.	Representa la fusión o unión de información entre imágenes. Es útil para combinar regiones de interés o máscaras binarias.
AND (Y lógica)	Solo conserva los píxeles que son blancos en ambas imágenes.	En el resultado aparecen únicamente las zonas donde las figuras se superponen	Muestra la intersección entre imágenes, permitiendo identificar zonas comunes o solapadas.

		con partes claras de la palmera.	
XOR (O exclusiva)	Deja en blanco solo los píxeles diferentes entre ambas imágenes. Si los dos píxeles coinciden (ambos blancos o ambos negros), el resultado es negro.	Se visualizan las figuras y la palmera, pero sin las zonas de coincidencia; las áreas comunes desaparecen.	Permite observar las diferencias entre imágenes. Se usa para comparar o detectar cambios entre dos tomas.
NOT (Negación) (<i>no aplicada</i>)	Invierte los valores binarios: blanco \leftrightarrow negro.	—	Sirve para invertir la imagen binaria, resaltando el fondo u objeto según se requiera. Es útil para crear el negativo lógico.

Tabla 7. Descripción de los resultados obtenidos

PARTE A: APLICACIÓN DE OPERACIONES LÓGICAS (CODIO PROPIO PYTHON)

Para el diseño en Python se tomaron en cuenta los códigos previamente compartidos por la profesora y se construyeron como base. Las funcionalidades que se integraron fueron:

- ❖ PREPARACIÓN DE IMÁGENES:
 - Cargar imagen A
 - Cargar imagen B
 - Binarizar A
 - Binarizar B
- ❖ OPERACIONES lógicas
 - AND
 - OR
 - XOR
 - NOT A
 - NOT B
- ❖ OPERACIONES RELACIONALES
 - A > B
 - A < B
 - A == B
 - A != B

Preparación de imágenes

En esta etapa se lleva a cabo la carga y preprocesamiento de las imágenes que servirán como base para aplicar las operaciones lógicas y relacionales para evitar errores.

Cargar imagen A y B:

Se pueden cargar dos imágenes desde el sistema de archivos mediante un cuadro de diálogo. Cada una se lee en formato BGR utilizando la librería OpenCV (cv2.imread) y se almacena en memoria. En esta fase no se aplican modificaciones, únicamente se prepara la información visual para su procesamiento.

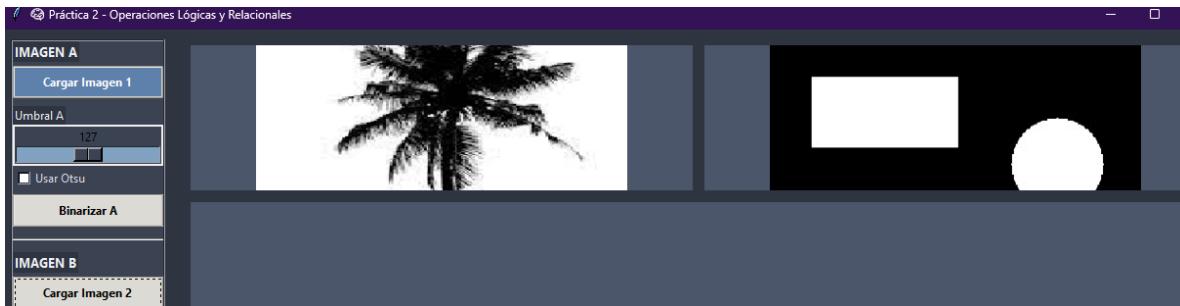


Figura 5. Carga de imágenes

Binarizar A y B:

- Una vez cargadas, las imágenes se convierten a escala de grises y luego a formato binario (blanco y negro) aplicando un umbral de intensidad.
- Este proceso asigna el valor 255 (blanco) a los píxeles cuya intensidad supera el umbral, y 0 (negro) a los que están por debajo.
- De esta forma, se genera una imagen binaria que facilita las operaciones lógicas (AND, OR, XOR, NOT), donde cada píxel representa un valor booleano (1 o 0).
- En el programa se permite ajustar manualmente el umbral o aplicar el método Otsu, que calcula automáticamente un valor óptimo de separación.

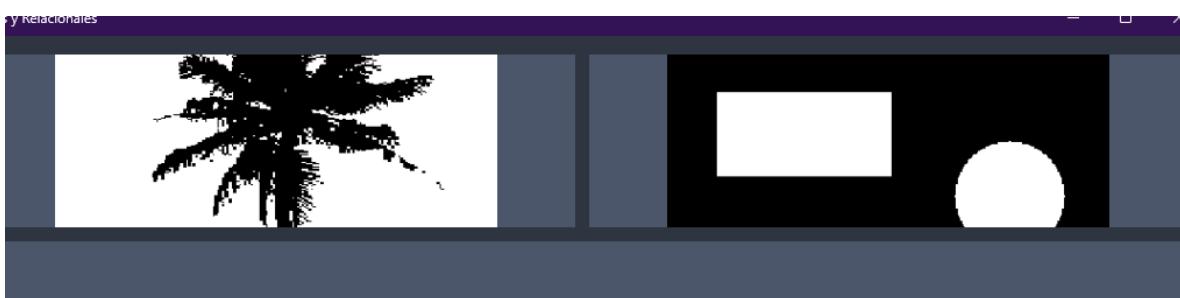


Figura 6. Binarización de imágenes

```

# ===== Utilidades =====
def cv_to_tk(img_bgr, max_size=(380, 380)):
    if img_bgr is None:
        return None
    if len(img_bgr.shape) == 2:
        img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_GRAY2RGB)
    else:
        img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    h, w = img_rgb.shape[:2]
    scale = min(max_size[0]/w, max_size[1]/h, 1.0)
    if scale != 1.0:
        img_rgb = cv2.resize(img_rgb, (int(w*scale), int(h*scale)), interpolation=cv2.INTER_LINEAR)
    pil_img = Image.fromarray(img_rgb)
    return ImageTk.PhotoImage(pil_img)

def ensure_same_size(a, b):
    if a is None or b is None:
        return a, b
    ha, wa = a.shape[:2]
    hb, wb = b.shape[:2]
    if (ha, wa) != (hb, wb):
        b = cv2.resize(b, (wa, ha), interpolation=cv2.INTER_NEAREST)
    return a, b

def to_binary(img, thresh=127, use_otsu=False, invert=False):
    if img is None:
        return None
    if len(img.shape) == 3:
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    else:
        gray = img.copy()
    ttype = cv2.THRESH_BINARY_INV if invert else cv2.THRESH_BINARY
    if use_otsu:
        _, binarized = cv2.threshold(gray, 0, 255, ttype + cv2.THRESH_OTSU)
    else:
        _, binarized = cv2.threshold(gray, int(thresh), 255, ttype)
    return binarized

```

Figura 7. Rutinas Binarización

Operaciones lógicas

Las operaciones lógicas permiten combinar o modificar imágenes binarias píxel a píxel, utilizando principios de la lógica booleana. Cada píxel se considera como un valor lógico, donde 255 representa 1 (verdadero) y 0 representa 0 (falso).

Estas operaciones son fundamentales en el procesamiento digital de imágenes, ya que permiten enmascarar regiones, detectar coincidencias o invertir áreas dentro de una imagen.

Operaciones lógicas

AND (Intersección lógica)

Esta operación devuelve 1 (blanco) solo cuando ambos píxeles correspondientes en las imágenes A y B son 1.

Sirve para encontrar áreas comunes entre dos imágenes o máscaras.

En términos de OpenCV, se implementa con `cv2.bitwise_and (A, B)`.

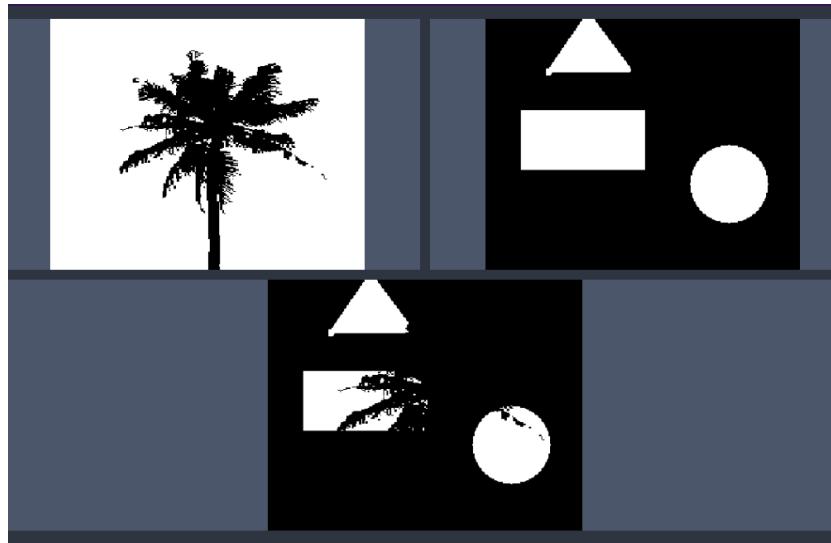


Figura 8. Resultado AND

- *OR (Unión lógica)*

Devuelve 1 si al menos uno de los píxeles en A o B es 1.

Se utiliza para combinar regiones o superponer resultados de diferentes procesos. En OpenCV, se realiza con `cv2.bitwise_or (A, B)`.

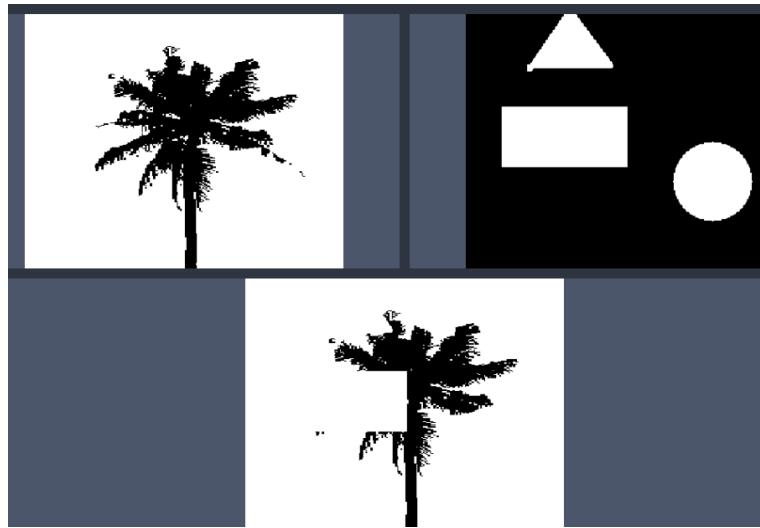


Figura 9. Resultado OR

- *XOR (Disyunción exclusiva)*

El resultado es 1 únicamente si los píxeles de A y B son diferentes (uno es 1 y el otro 0). Permite resaltar las diferencias entre imágenes o detectar bordes donde las regiones no coinciden. Se ejecuta con `cv2.bitwise_xor (A, B)`.

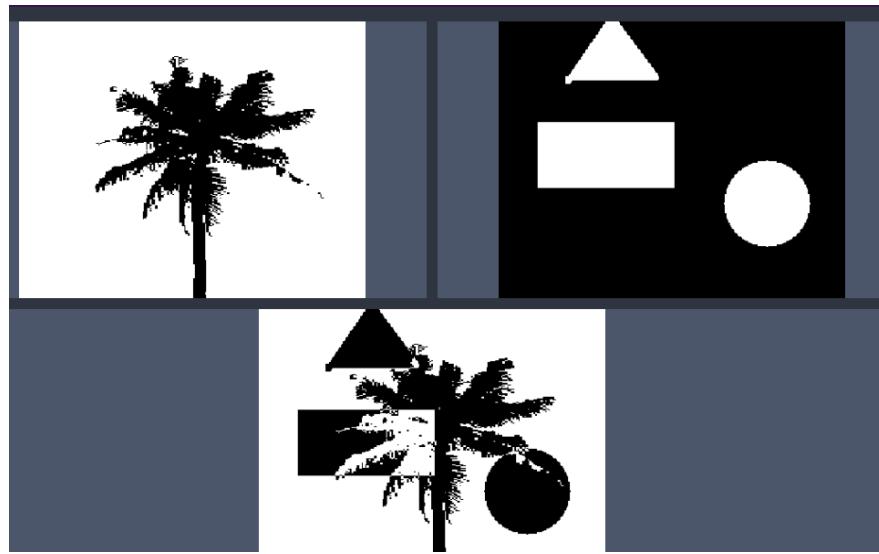


Figura 10. Resultado XOR

• NOT A / NOT B (Negación lógica)

Invierte los valores de una imagen: los píxeles blancos pasan a negro y viceversa. Sirve para obtener el complemento lógico de una máscara o imagen binaria. En OpenCV, se aplica con cv2.bitwise_not(A) o cv2.bitwise_not(B).

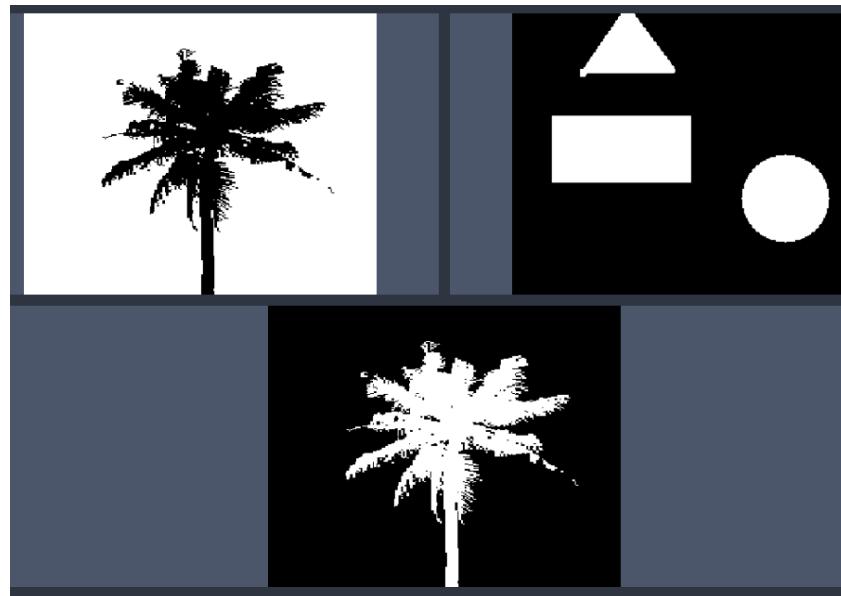


Figura 11. Resultado NOT A

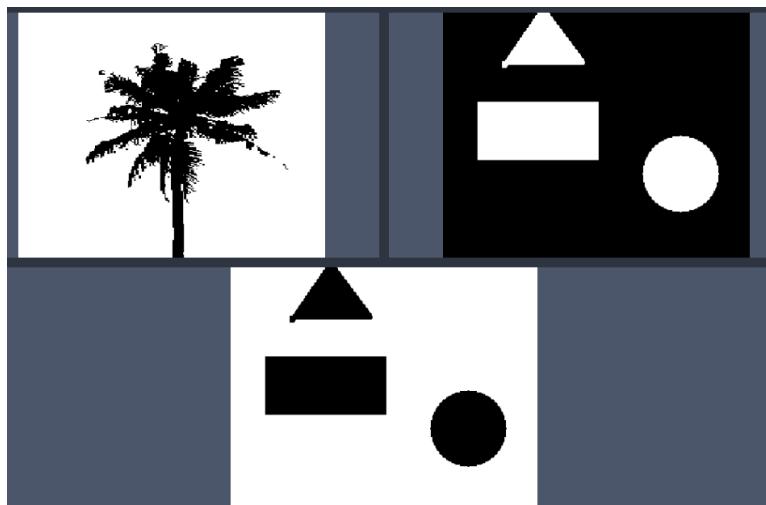


Figura 12. Resultado NOT B

```
# ===== Operaciones lógicas y relacionales =====
def get_pair_binary(self):
    if self.binA is None or self.binB is None:
        messagebox.showinfo("Atención", "Binariza ambas imágenes primero.")
        return None, None
    A, B = ensure_same_size(self.binA, self.binB)
    return A, B

def op_and(self):
    A, B = self.get_pair_binary()
    if A is None: return
    self.show_result(cv2.bitwise_and(A, B))

def op_or(self):
    A, B = self.get_pair_binary()
    if A is None: return
    self.show_result(cv2.bitwise_or(A, B))

def op_xor(self):
    A, B = self.get_pair_binary()
    if A is None: return
    self.show_result(cv2.bitwise_xor(A, B))

def op_not_a(self):
    if self.binA is None:
        return messagebox.showinfo("Atención", "Binariza primero la imagen A.")
    self.show_result(cv2.bitwise_not(self.binA))

def op_not_b(self):
    if self.binB is None:
        return messagebox.showinfo("Atención", "Binariza primero la imagen B.")
    self.show_result(cv2.bitwise_not(self.binB))
```

Figura 12. Rutinas operaciones lógicas

PARTE B: OPERACIONES RELACIONALES

Las operaciones relacionales permiten comparar dos imágenes píxel a píxel con base en sus valores de intensidad.

A diferencia de las operaciones lógicas (que trabajan con valores binarios 0 y 1), las relacionales evalúan el valor numérico completo del píxel (por ejemplo, entre 0 y 255 en escala de grises) y producen una imagen binaria como resultado, donde:

- 255 (blanco) representa una condición verdadera.
- 0 (negro) representa una condición falsa.

Estas operaciones son muy útiles para comparar regiones de brillo, detectar cambios entre imágenes o realizar segmentaciones por intensidad.

```
def op_gt(self):
    A, B = self.get_pair_binary()
    if A is None: return
    self.show_result(np.where(A > B, 255, 0).astype(np.uint8))

def op_lt(self):
    A, B = self.get_pair_binary()
    if A is None: return
    self.show_result(np.where(A < B, 255, 0).astype(np.uint8))

def op_eq(self):
    A, B = self.get_pair_binary()
    if A is None: return
    self.show_result(np.where(A == B, 255, 0).astype(np.uint8))

def op_neq(self):
    A, B = self.get_pair_binary()
    if A is None: return
    self.show_result(np.where(A != B, 255, 0).astype(np.uint8))
```

Figura 13. Rutinas operaciones relacionales

$A > B$ (Mayor que)

Evaluá si el valor de intensidad del píxel en la imagen A es **mayor** que el correspondiente en la imagen B.

Cuando la condición se cumple, el píxel resultante se establece en **255**; en caso contrario, en **0**. Se implementa con cv2.compare(A, B, cv2.CMP_GT).

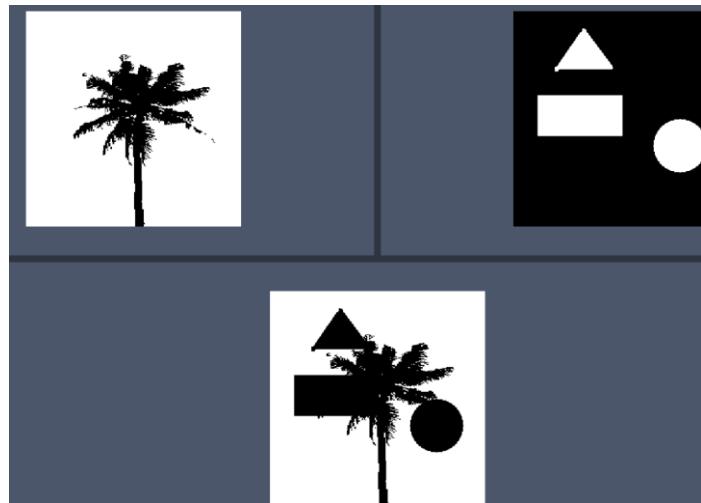


Figura 14. Resultado A > B

• A < B (Menor que)

Compara los valores de intensidad y devuelve 255 cuando el píxel de A es menor que el de B.

Es útil para identificar zonas más oscuras o con menor nivel de brillo.

Se aplica con cv2.compare(A, B, cv2.CMP_LT).



Figura 15. Resultado A < B

• A == B (Igualdad)

Devuelve 255 cuando los valores de intensidad de ambos píxeles son iguales.

Sirve para detectar coincidencias exactas entre dos imágenes o máscaras binarias.

Se realiza con cv2.compare(A, B, cv2.CMP_EQ).

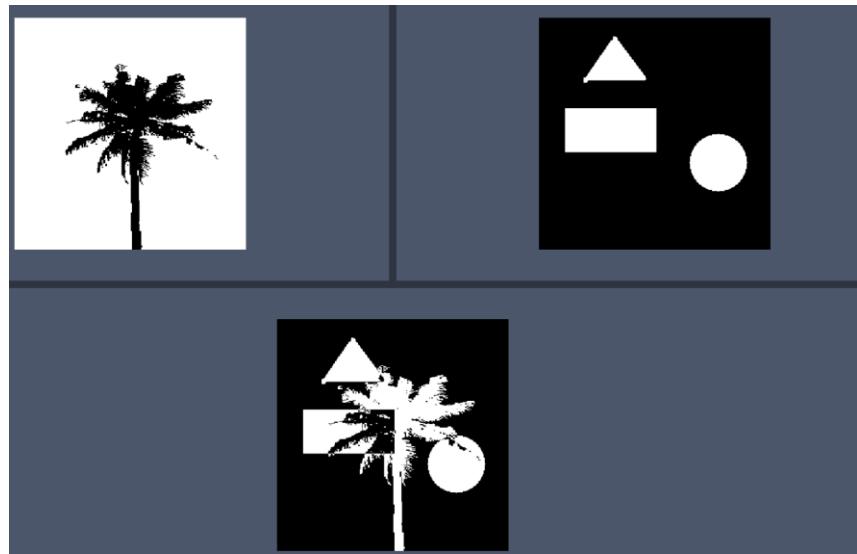


Figura 16. Resultado $A == B$

- $A != B$ (Diferente)

Devuelve 255 cuando los valores de los píxeles no son iguales.

Es útil para detectar diferencias o cambios entre imágenes, por ejemplo, en análisis de movimiento o comparación de versiones.

Se ejecuta con `cv2.compare(A, B, cv2.CMP_NE)`.

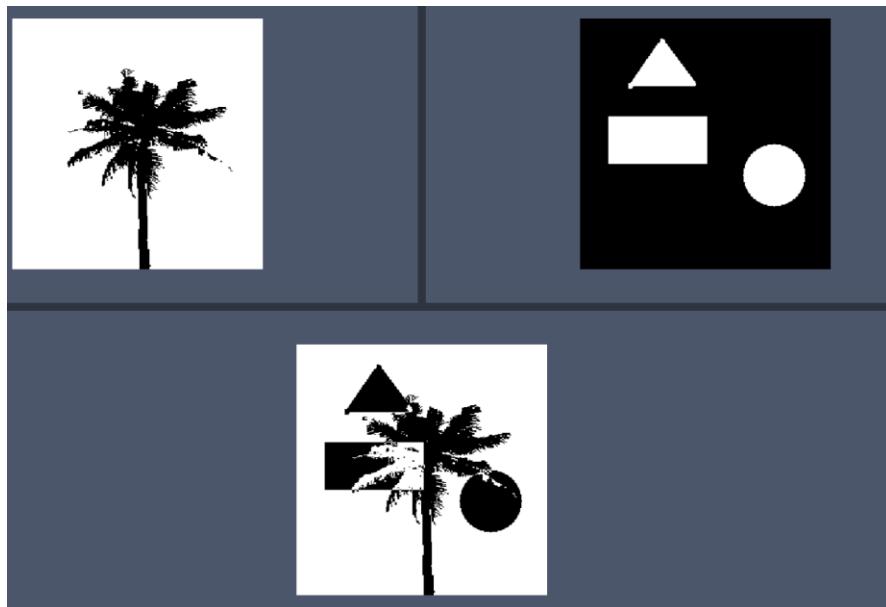


Figura 17. Resultado $A != B$

CUESTIONARIO PARTE A

1. ¿Qué operación lógica permitió extraer mejor la información relevante?

La operación AND fue la que mejor permitió extraer la información relevante, el AND sirve para resaltar las partes que coinciden entre dos imágenes (como zonas blancas o con forma parecida), eliminando el ruido o lo que no se repite. (Como entendí se pone dos figuras encimadas y solo se muestran las zonas que coinciden)

2. ¿Qué diferencias observaste entre las operaciones AND, OR, XOR y NOT?

Operación	Explicación con ejemplo cotidiano	Qué hace en la imagen
AND	Solo se ve lo que ambos tienen igual. (Lo que esta dentro del ambos contornos en común)	Muestra las partes comunes (intersección).
OR	Se ve todo lo que alguno tiene. (Por ejemplo, todo el contorno de las figuras encimadas)	Une las zonas blancas de ambas imágenes.
XOR	Solo se ve lo que no coincide. (Opuesto al Or)	Resalta las diferencias entre imágenes.
NOT	Lo blanco se vuelve negro y lo negro se vuelve blanco.	Invierte los colores, mostrando el negativo de la imagen.

Tabla 8. Resultados de cada operación lógica

3. ¿Qué utilidad tienen las operaciones relacionales en el procesamiento de imágenes?

Comparan valores de brillo o intensidad entre dos imágenes, por ejemplo “ $A > B$ ”, el programa marca en blanco los puntos donde la imagen A es más clara que B.

4. ¿Qué dificultades encontraste durante la práctica?

Que para poder aplicar las operaciones correctamente fue necesario redimensionar las imágenes para que pudieran hacerse de buena manera, la solución más fácil que se halla fue redimensionarlas con el simulador que se nos compartió y así ahorraron un paso.

5. ¿Cómo podrías aplicar estas técnicas en un contexto real?

La única forma que se me ocurre es la comparación entre objetos como degradación con el tiempo, descomposición, crecimiento, ect.. así como el crecimiento de maleza, hundimiento de sueldos etc.. algo así que necesite una comparación para identificar diferencias o puntos en común dentro de algún objeto.

PARTE B. ETIQUETADO DE COMPONENTES CONEXAS

La vecindad 8 generalmente es más precisa para definir regiones conectadas porque permite detectar conectividades más amplias, evitando que los objetos con conexiones en diagonal o curvas se etiqueten como elementos separados. La vecindad 4 es ideal cuando quieras limitar la conexión a píxeles ortogonales.

Para el etiquetado se copió el código que la profesora compartió en la práctica 3, para poder conocer previamente el etiquetado con vecindades.

```

import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt

# Matriz, simula una imagen binaria
# 0 --> Fondo
# 1 --> Sección de figura
imagen_binaria = np.array([
[0, 0, 0, 1, 1, 0, 0, 0],
[0, 1, 1, 1, 1, 1, 0, 0],
[0, 1, 1, 0, 0, 1, 1, 0],
[0, 0, 0, 1, 1, 0, 0, 0],
[0, 0, 1, 1, 0, 0, 1, 1],
[0, 1, 1, 1, 1, 1, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0]
], dtype=int)

# Definición de vecindad 4 -
vecindad_4 = np.array([[0, 1, 0],
[1, 1, 1],
[0, 1, 0]], dtype=int)

# Definición de vecindad 8
vecindad_8 = np.ones((3, 3), dtype=int) # Matriz de 8-conexión

# Etiquetados de vecindad
# recorre la imagen del array, y los procesa, regresa el array con mapeado de figuras
etiquetas_4, num_objetos_4 = ndimage.label(imagen_binaria, structure=vecindad_4)
etiquetas_8, num_objetos_8 = ndimage.label(imagen_binaria, structure=vecindad_8)

# Mostrar el número de objetos detectados
print("Número de objetos con vecindad 4:", num_objetos_4)
print("Número de objetos con vecindad 8:", num_objetos_8)
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

# Imagen binaria original
axes[0].imshow(imagen_binaria, cmap='gray')
axes[0].set_title("Imagen Binaria")
axes[0].axis('off')

# Etiquetado con vecindad 4
axes[1].imshow(etiquetas_4, cmap='nipy_spectral')
axes[1].set_title(f"Vecindad 4 - {num_objetos_4} Objetos")
axes[1].axis('off')

# Etiquetado con vecindad 8
axes[2].imshow(etiquetas_8, cmap='nipy_spectral')
axes[2].set_title(f"Vecindad 8 - {num_objetos_8} Objetos")
axes[2].axis('off')
plt.show()

```

Lo que hace básicamente el algoritmo es usar las librerías de script que nos permite utilizar la detección de vecindades, en este caso se usa para identificar estructuras previamente declaradas, se definen dos estructuras:

vecindad_4 → Estructura creada con un array y define la estructura de 4 n
 vecindad_8 → Estructura creada con un array y define la estructura de 8 n

Después simplemente se procesa la imagen, en donde se colocan dos variables que llevarán los contadores de los etiquetados y que procesan la imagen por medio de la comparación de los arrays:

etiquetas_4, num_objetos_4 = ndimage.label(imagen_binaria, structure=vecindad_4)

```

etiquetas_8, num_objetos_8 = ndimage.label(imagen_binaria, structure=vecindad_8)

imagen_binaria = np.array([
    [0, 0, 0, 1, 1, 0, 0, 0],
    [0, 1, 1, 1, 1, 1, 0, 0],
    [0, 1, 1, 0, 0, 1, 1, 0],
    [0, 0, 0, 1, 1, 0, 0, 0],
    [0, 0, 1, 1, 0, 0, 1, 1],
    [0, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 1, 0, 0, 0, 0]
], dtype=int)

# Definicion de vecindad 4 -
vecindad_4 = np.array([[0, 1, 0],
                      [1, 1, 1],
                      [0, 1, 0]], dtype=int)

#Definicion de vecindad 8
vecindad_8 = np.ones((3, 3), dtype=int) # Matriz de 8-conexión

# Etiquetados de vecindad
#recorre la imagen del array, y los procesa, regresa el array con mapeado de figuras
etiquetas_4, num_objetos_4 = ndimage.label(imagen_binaria, structure=vecindad_4)
etiquetas_8, num_objetos_8 = ndimage.label(imagen_binaria, structure=vecindad_8)

```

Figura 18. Construcción base de la Etiquetas conexas

Para verse de una forma visual se tiene el siguiente ejemplo:

Recibe este array que simula una imagen binaria:

```

0 1 1 0
0 0 1 0
1 0 0 1
1 1 0 1

```

Y una vez que lo procesa se podría ver el array de esta forma:

```

0 1 1 0
0 0 1 0
2 0 0 3
2 2 0 3

```

Hay 3 objetos (1, 2, 3).

- ❖ El objeto 1 está arriba.
- ❖ El objeto 2 abajo a la izquierda.
- ❖ El objeto 3 abajo a la derecha.

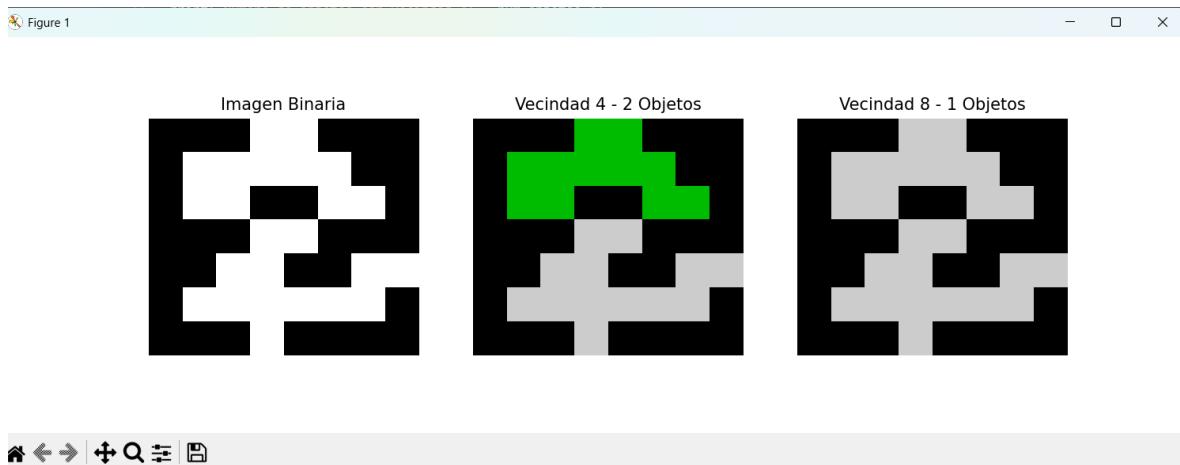


Figura 19. Resultado de la Etiquetas conexas

PARTE B. ETIQUETADO DE COMPONENTES CONEXAS (CODIGO PROPIO)

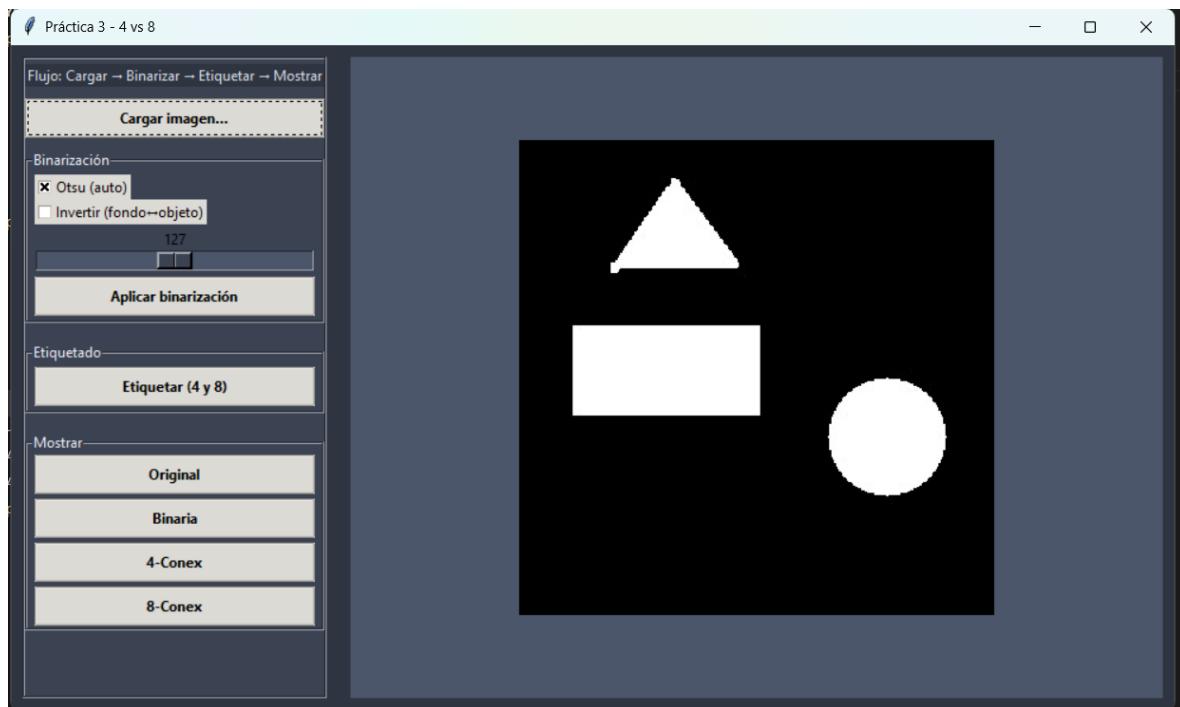


Figura 20. Interfaz Etiquetas conexas

Carga de imagen

Para la carga de imagen se colocó un botón con la ayuda de la librería de tkinter, que nos permite crear interfaces en Python, ahora al presionar el botón se agrega la negación a directorio el cual permite seleccionar la imagen de carga.

Binarización

Una vez que se selecciona la carga de imagen se necesita trabajar para poder aplicarle las vecindades, entonces es necesario presionar el botón de binarización, que convierte la imagen a grises y luego le aplica binarización, se tiene la opción de aplicar otsu que lo que hace buscar el umbral_automático que mejor se adapte al resultado de la imagen.

```
def to_binary(img, thresh=127, use_otsu=True, invert=False):
    if img is None:
        return None
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) if len(img.shape) == 3 else img.copy()
    flag = cv2.THRESH_BINARY_INV if invert else cv2.THRESH_BINARY
    if use_otsu:
        _, binimg = cv2.threshold(gray, 0, 255, flag + cv2.THRESH_OTSU)
    else:
        _, binimg = cv2.threshold(gray, int(thresh), 255, flag)
    return binimg
```

Figura 21. Rutina Binarizar

Para la binarización se reciben los parámetros la cual es la imagen y dos banderas, una bandera es para aplicar la binarización con otsu y la otra para la inversión, que la inversión se enfoca en

- cv2.threshold(gray, thresh, 255, flag)
- cv2.THRESH_BINARY_INV

Etiquetado de componentes

Se aplican dos etiquetados con `scipy.ndimage.label` sobre la imagen binaria (convertida a 0/1):

- Conectividad 4: usa una máscara estructural cruz (solo arriba/abajo/izquierda/derecha).
- Conectividad 8: usa una máscara 3×3 completa (incluye diagonales). Cada etiquetado produce:
 - Una matriz de etiquetas (cada objeto recibe un ID entero), y
 - El número de objetos detectados.
 - Los resultados se colorean aleatoriamente para facilitar la interpretación visual.

Visualización

La interfaz muestra, a elección del usuario:

- Original, Binaria, Etiquetas con vecindad 4, y Etiquetas con vecindad 8. Además, se imprime en consola el conteo de objetos detectados por cada conectividad para compararlos.

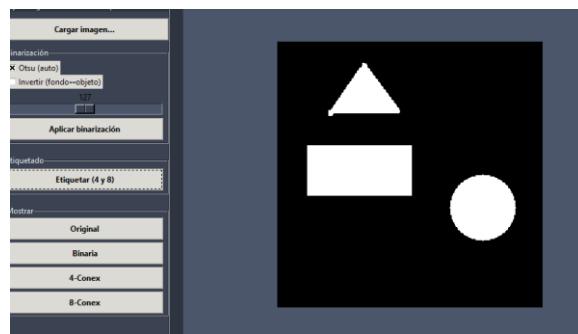


Figura 22. Imagen de prueba binarizada

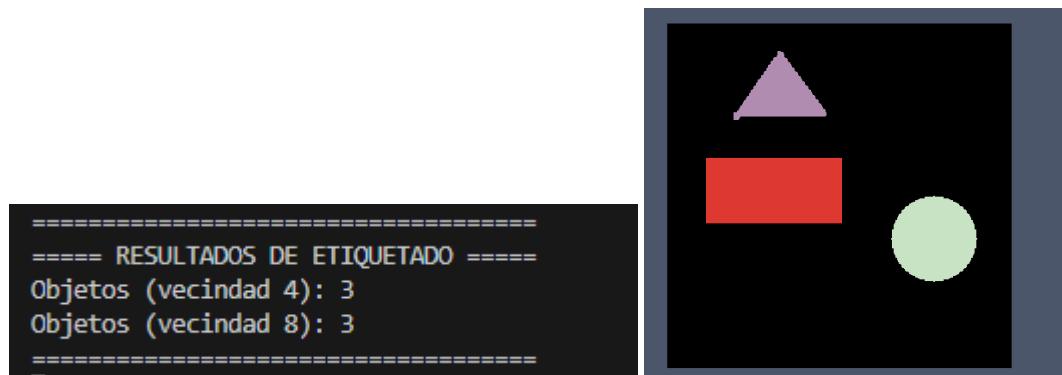


Figura 23. Etiqueta de la imagen binarizada

Hicimos la prueba con una imagen mayor:

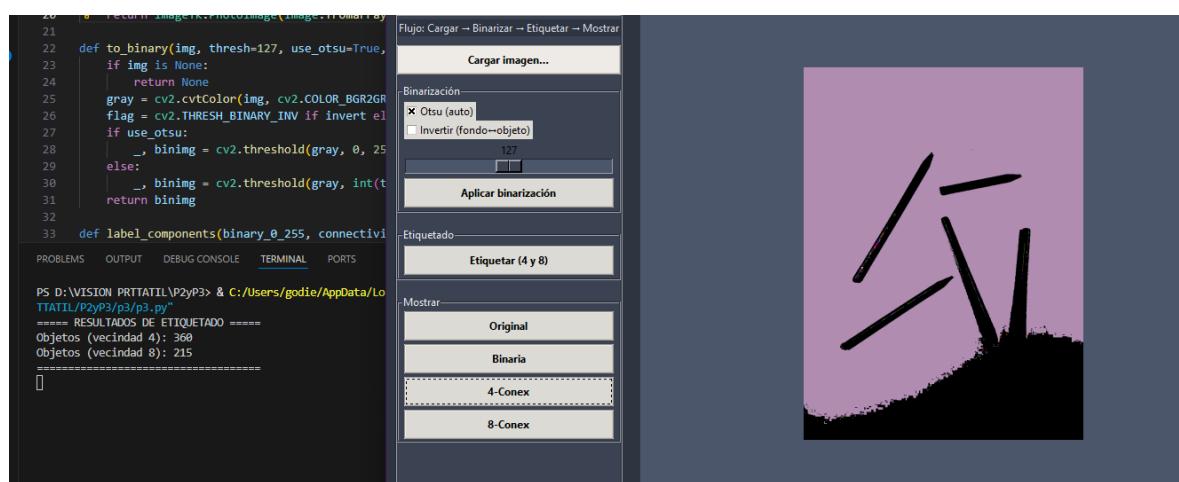


Figura 24. Etiqueta de la imagen binarizada

CONCLUSIÓN GARCIA CERON DIEGO

Durante esta práctica pude entender de forma más clara cómo funciona la conectividad en las imágenes binarias. Antes pensaba que los objetos detectados dependían solo del umbral, pero comprobé que también influye el tipo de vecindad que se usa. Con la vecindad 4 se separan más las

figuras, mientras que con la vecindad 8 se agrupan más fácilmente, ya que toma en cuenta las diagonales. En general, me pareció una práctica muy visual y útil para comprender cómo los programas “interpretan” las formas dentro de una imagen.

CONCLUSION LOPEZ MARTINEZ HECTOR ALEXIS

Esta práctica me ayudó a reforzar el concepto de conectividad y a ver cómo pequeños cambios en la definición de vecinos pueden alterar los resultados del análisis de una imagen. Usar la interfaz fue sencillo y permitió comparar los resultados de manera directa. Aprendí que la elección entre vecindad 4 o 8 depende del tipo de imagen y del propósito del análisis, ya que una puede unir regiones que la otra considera separadas. Fue una experiencia interesante para entender mejor cómo los algoritmos procesan las estructuras visuales.

REFERENCIAS

- [1] “Fundamentos bÁsicos del procesamiento de imÁgenes — documentaciÃ³n de Curso de imÃ¡genes mÃ©dicas - 1.0”. Bienvenido a FAMAF - Facultad de Matemática, Astronomía, Física y Computación. Accedido el 18 de octubre de 2025. [En línea]. Disponible: <https://www.famaf.unc.edu.ar/~pperez1/manuales/cim/cap2.html>
- [2] “VI. PROCESAMIENTO DE IMÁGENES”. Inicio. Accedido el 18 de octubre de 2025. [En línea]. Disponible: https://bibliotecadigital.ilce.edu.mx/sites/ciencia/volumen2/ciencia3/084/htm/sec_9.htm
- [3] “Cryptocurrency Prices, Market Cap, Trading Charts: Bitcoin, Ethereum and more at PiedPiper”. PiedPiper: Crypto Prices, Live Charts and News. Accedido el 18 de octubre de 2025. [En línea]. Disponible: <https://pp.one/>
- [4] Accedido el 18 de octubre de 2025. [En línea]. Disponible: <https://grupo.us.es/gtocoma/pid/tema1-1.pdf>
- [5] Science Code – Science, coding and more. Accedido el 18 de octubre de 2025. [En línea]. Disponible: https://sciencesoftware.wordpress.com/wp-content/uploads/2019/10/imagenes_digitales.pdf