

Actividad_C1_P1_LuisGonzalez_MichaelLaudrup

July 18, 2025

0.1 Convocatoria 1 - Proyecto 1

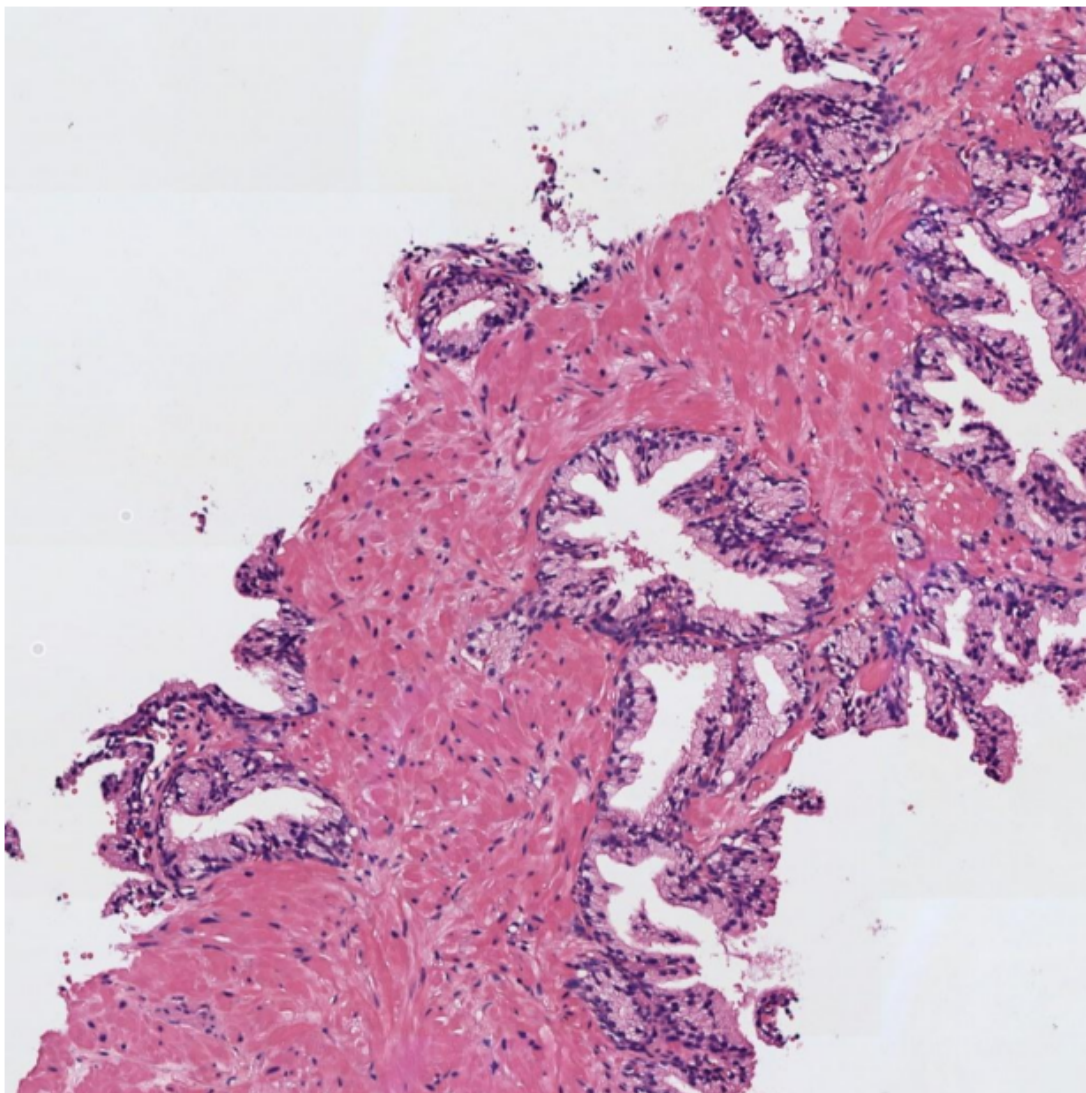
0) Cargar una de las imágenes histológicas

```
[278]: # Utilizar la librería skimage.io para leer la imagen 'histo_x.jpg' en formato
      ↪ RGB.
from skimage import io
import numpy as np
import matplotlib.pyplot as plt

# Leer las imágenes
histoRGB = io.imread('histo_1.jpg')

MAX_RGB_VAL = 255
MIN_RGB_VAL = 0
histoRGB_normalized = histoRGB.astype(float) / MAX_RGB_VAL

plt.figure(figsize=(10,8))
plt.imshow(histoRGB)
plt.axis('off')
plt.show()
```



1) Realizar una transformación de color para convertir la imagen al espacio de color CMYK

- Pillow para convertir la imagen a CMYK necesita que los valores esten entre [0-255], dado que anteriormente se solicita de manera explicita realizar una normalización, se entiende que no se quiere usar pillow. Se descarta esta opción.
- Tal y como se aprecia en <https://scikit-image.org/docs/0.25.x/api/skimage.color.html#module-skimage.color>, no se encuentra ninguna función en skimage que permita convertir una imagen RGB a CMYK, por lo que se implementa una función propia. Siguiendo la referencia de wikipedia: https://es.wikipedia.org/wiki/Modelo_de_color_CMYK#Mapeado_de_RGB_a_CMYK

```
[279]: def rgb_to_cmyk_vectorized(rgb_img):
        """
        Convierte una imagen RGB (normalizada en [0, 1]) a CMYK usando operaciones
        de NumPy.
        Devuelve una única imagen CMYK de 4 canales.
        """
        r = rgb_img[:, :, 0]
        g = rgb_img[:, :, 1]
        b = rgb_img[:, :, 2]

        k = 1 - np.maximum.reduce([r, g, b])

        # Evitar la división por cero para los píxeles que son completamente negros.
        c = np.where(k == 1, 0, (1 - r - k) / (1 - k))
        m = np.where(k == 1, 0, (1 - g - k) / (1 - k))
        y = np.where(k == 1, 0, (1 - b - k) / (1 - k))

        cmyk_img = np.stack([c, m, y, k], axis=-1)
        return cmyk_img

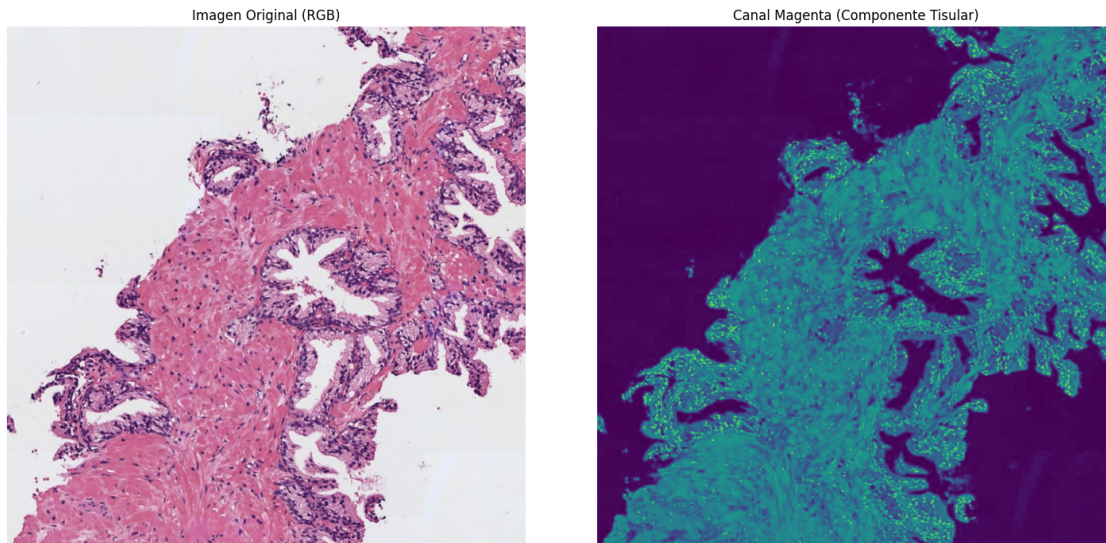
# Convertir la imagen normalizada a CMYK
cmyk_img = rgb_to_cmyk_vectorized(histoRGB_normalized)

# Extraer la componente magenta de la imagen
magenta_channel = cmyk_img[:, :, 1]

fig, axes = plt.subplots(1, 2, figsize=(15, 7))

axes[0].imshow(histoRGB_normalized)
axes[0].set_title('Imagen Original (RGB)')
axes[0].axis('off')

im = axes[1].imshow(magenta_channel)
axes[1].set_title('Canal Magenta (Componente Tisular)')
axes[1].axis('off')
plt.tight_layout()
plt.show()
```



2) Umbralizar la imagen para separar los píxeles del fondo de la región tisular

```
[280]: import matplotlib.pyplot as plt
import cv2
import numpy as np

magenta_uint8 = (magenta_channel * 255).astype(np.uint8)

smoothed_image_cv2 = cv2.GaussianBlur(magenta_uint8, (5, 5), 0)

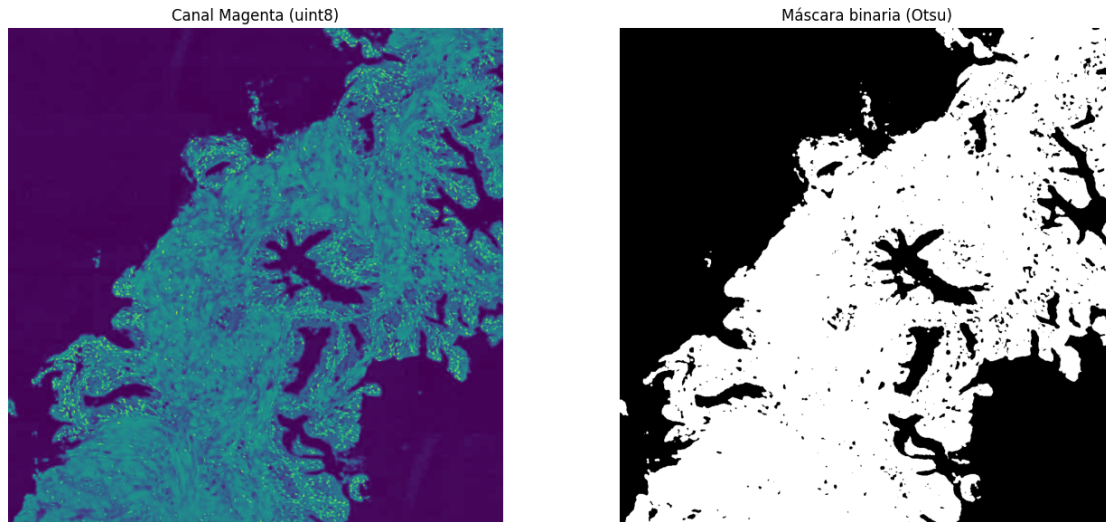
_, mask_otsu = cv2.threshold(smoothed_image_cv2, 0, 255, cv2.THRESH_BINARY +
↪cv2.THRESH_OTSU)

fig, ax = plt.subplots(1, 2, figsize=(14, 6))

ax[0].imshow(magenta_uint8)
ax[0].set_title('Canal Magenta (uint8)')
ax[0].axis('off')

ax[1].imshow(mask_otsu, cmap='gray')
ax[1].set_title('Máscara binaria (Otsu)')
ax[1].axis('off')

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



3) Limpiar la imagen eliminando los artefactos de lumen (objetos blancos pequeños que no son lúmenes)

```
[281]: # Utilizar la librería skimage.morphology.remove_small_objects para eliminar
        ↪ aquellos objetos cuya área sea menor a 300 píxeles
        # Más información en https://scikit-image.org/docs/dev/api/skimage.morphology.
        ↪ html#skimage.morphology.remove_small_objects
        # Visualizaer la máscara resultante
        from skimage.morphology import remove_small_objects
        import matplotlib.pyplot as plt

        # Invertimos la máscara para que los objetos sean True
        mask_bool = ~mask_otsu.astype(bool)

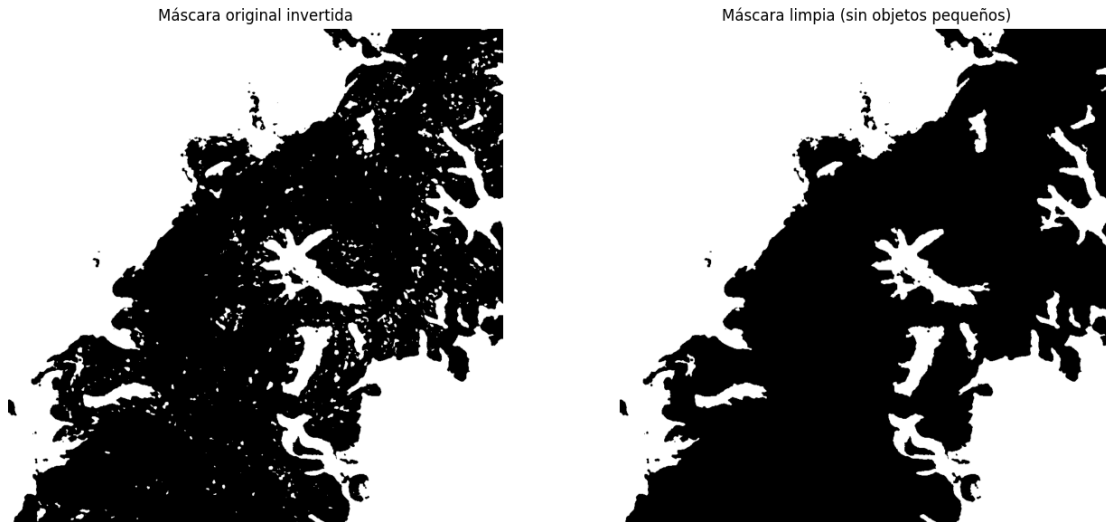
        mask_clean = remove_small_objects(mask_bool, min_size=300)

        fig, ax = plt.subplots(1, 2, figsize=(14, 6))

        ax[0].imshow(mask_bool, cmap='gray')
        ax[0].set_title('Máscara original invertida')
        ax[0].axis('off')

        ax[1].imshow(mask_clean, cmap='gray')
        ax[1].set_title('Máscara limpia (sin objetos pequeños)')
        ax[1].axis('off')

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



4) Rellenar con 0s el fondo de la imagen para quedarnos únicamente con los lúmenes

```
[282]: import cv2
import numpy as np
import matplotlib.pyplot as plt

lumens_mask = mask_clean.astype(np.uint8) * 255

h, w = lumens_mask.shape

semillas_fondo = [
    (0, 0),
    (h - 1, w - 1)
]

for seed in semillas_fondo:
    if lumens_mask[seed[0], seed[1]] == 255:
        cv2.floodFill(lumens_mask, None, seedPoint=(seed[1], seed[0]), newVal=0)
```

5) Rellenar los objetos de los lúmenes

```
[283]: # Rellenar los lúmenes con la función binary_fill_holes de la librería scipy.
        ↳ ndimage.morphology
# Visualizar la máscara resultante
import matplotlib.pyplot as plt
from scipy.ndimage import binary_fill_holes

filled_mask = binary_fill_holes(lumens_mask)

fig, ax = plt.subplots(1, 2, figsize=(14, 6))
```



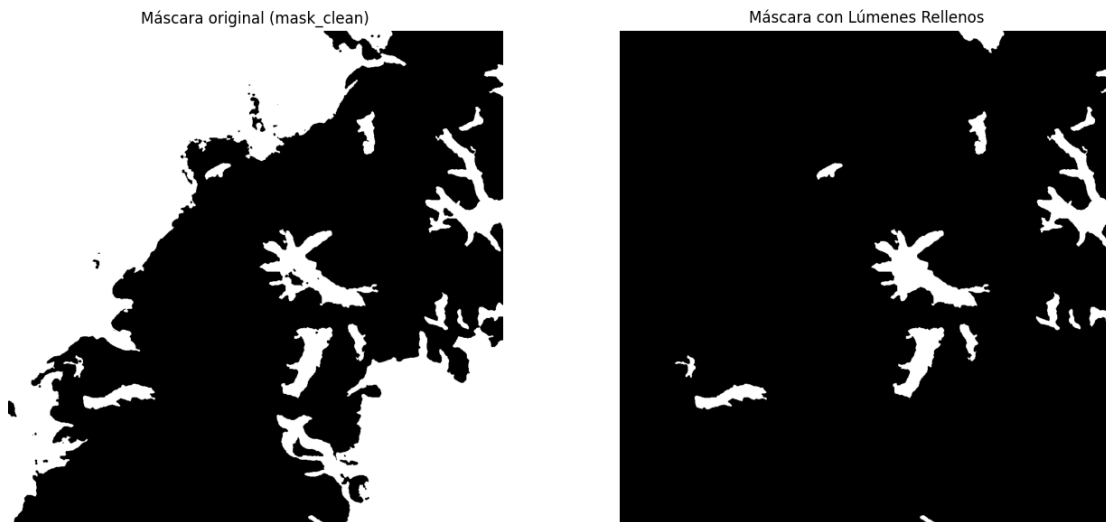
```

ax[0].imshow(mask_clean, cmap='gray')
ax[0].set_title('Máscara original (mask_clean)')
ax[0].axis('off')

ax[1].imshow(filled_mask, cmap='gray')
ax[1].set_title('Máscara con Lúmenes Rellenos')
ax[1].axis('off')

plt.tight_layout()
plt.show()

```



6) Detectar y dibujar los contornos de los lúmenes sobre la imagen original

```

[284]: import cv2
import numpy as np
import matplotlib.pyplot as plt

mask_uint8 = (filled_mask.astype(np.uint8)) * 255
contours, _ = cv2.findContours(mask_uint8, cv2.RETR_EXTERNAL, cv2.
    ↳CHAIN_APPROX_SIMPLE)

black_background = np.zeros_like(histoRGB)
cv2.drawContours(black_background, contours, -1, (0, 255, 0), thickness=2,
    ↳lineType=cv2.LINE_AA)

image_with_contours = histoRGB.copy()

```

```

cv2.drawContours(image_with_contours, contours, -1, (0, 255, 0), thickness=2,
↳lineType=cv2.LINE_AA)

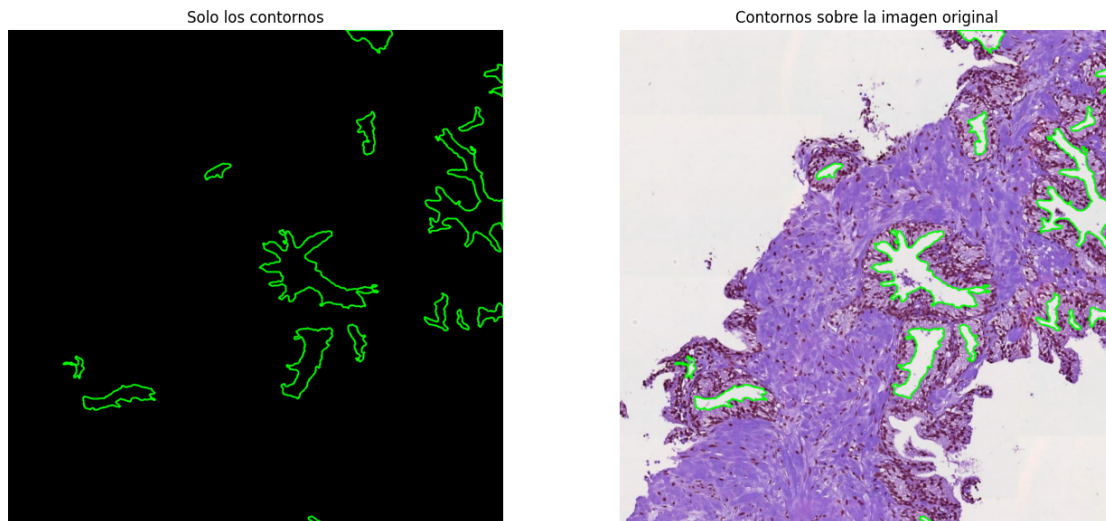
fig, axes = plt.subplots(1, 2, figsize=(14,6))

axes[0].imshow(cv2.cvtColor(black_background, cv2.COLOR_BGR2RGB))
axes[0].set_title('Solo los contornos')
axes[0].axis('off')

axes[1].imshow(cv2.cvtColor(image_with_contours, cv2.COLOR_BGR2RGB))
axes[1].set_title('Contornos sobre la imagen original')
axes[1].axis('off')

plt.tight_layout()
plt.show()

```



7) Identificar y cropear el lumen más grande

```

[285]: import cv2
import numpy as np
import matplotlib.pyplot as plt
mask_uint8 = (filled_mask.astype(np.uint8)) * 255

contours, _ = cv2.findContours(mask_uint8, cv2.RETR_EXTERNAL, cv2.
↳CHAIN_APPROX_SIMPLE)
max_contour = max(contours, key=cv2.contourArea)

x, y, w, h = cv2.boundingRect(max_contour)

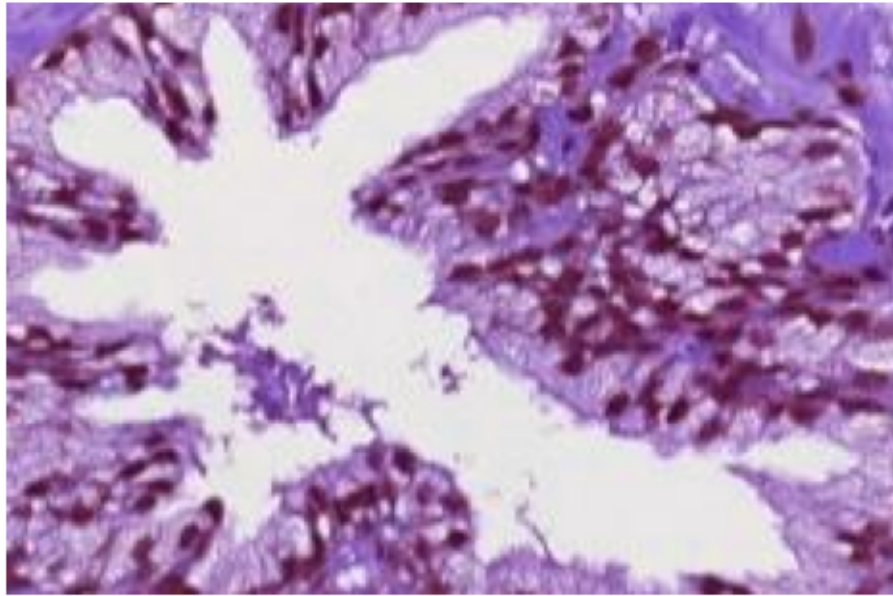
```



```
cropped_lumen = histoRGB[y:y+h, x:x+w].copy()

plt.figure(figsize=(6, 6))
plt.imshow(cv2.cvtColor(cropped_lumen, cv2.COLOR_BGR2RGB))
plt.title('Lumen de mayor área (cropeado)')
plt.axis('off')
plt.show()
```

Lumen de mayor área (cropeado)



8) Extraer 13 características geométricas que permitan caracterizar el lumen recortado

- Para medir la compacidad se ha usado como referencia:
https://d1rbsgppyrdqq4.cloudfront.net/s3fs-public/c160/Li_Goodchild_Church_CompactnessIndex.pdf?version=Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIASBVQ3ZQ477LUGVF2/20250718/us-west-2/s3/aws4_request&X-Amz-Date=20250718T095303Z&X-Amz-SignedHeaders=host&X-Amz-Expires=120&X-Amz-Signature=d0cfd74387118f3e4ca6ca5025f2a675c17c3a9e96432bc234c6081b6f6c03

[286]: *# Calcular las siguientes características del crop del lumen de mayor área, ↵
 ↪redondeando su valor hasta el cuarto decimal.*

```
# 1) Área
# 2) Área de la bounding box
# 3) Área convexa
# 4) Exentricidad
# 5) Diámetro equivalente
```

```

# 6) Extensión
# 7) Diámetro Feret
# 8) Longitud del eje mayor
# 9) Longitud del eje menor
# 10) Orientación
# 11) Perímetro
# 12) Solidez
# 13) Compacidad

from skimage.measure import regionprops
from math import pi, sqrt
import pandas as pd

x, y, w, h = cv2.boundingRect(max_contour)

mask_single = np.zeros_like(mask_uint8)
cv2.drawContours(mask_single, [max_contour], -1, color=255, thickness=-1)

region = regionprops(mask_single)[0]

area = region.area
bbox_area = w * h
hull = cv2.convexHull(max_contour)
convex_area = cv2.contourArea(hull)
eccentricity = region.eccentricity
equivalent_diameter = region.equivalent_diameter
extent = area / bbox_area
feret_diameter = sqrt(w**2 + h**2)

if len(max_contour) >= 5:
    ellipse = cv2.fitEllipse(max_contour)
    (_, (MA, ma), angle) = ellipse
    major_axis_length = max(MA, ma)
    minor_axis_length = min(MA, ma)
    orientation = angle
else:
    major_axis_length = minor_axis_length = orientation = np.nan

perimeter = cv2.arcLength(max_contour, True)
solidity = area / convex_area if convex_area > 0 else np.nan
compactness = (4 * pi * area) / (perimeter**2)

features = {
    "Área (píxeles2)": round(area, 4),
    "Área Bounding Box (píxeles2)": round(bbox_area, 4),

```

```

"Área Convexa (píxeles²)": round(convex_area, 4),
"Excentricidad ": round(eccentricity, 4),
"Diámetro Equivalente (píxeles)": round(equivalent_diameter, 4),
"Extensión": round(extent, 4),
"Diámetro Feret (píxeles)": round(feret_diameter, 4),
"Eje Mayor (píxeles)": round(major_axis_length, 4),
"Eje Menor (píxeles)": round(minor_axis_length, 4),
"Orientación (grados)": round(orientation, 4),
"Perímetro (píxeles)": round(perimeter, 4),
"Solidez ": round(solidity, 4),
"Compacidad": round(compactness, 4)
}

df = pd.DataFrame(features, index=["Lumen Mayor"])
display(df)

```

	Área (píxeles²)	Área Bounding Box (píxeles²)	\
Lumen Mayor	14110.0		40098
	Área Convexa (píxeles²)	Excentricidad	\
Lumen Mayor	28968.0	0.847	
	Diámetro Equivalente (píxeles)	Extensión	\
Lumen Mayor	134.0351	0.3519	
	Diámetro Feret (píxeles)	Eje Mayor (píxeles)	\
Lumen Mayor	295.1017	229.8147	
	Eje Menor (píxeles)	Orientación (grados)	Perímetro (píxeles) \
Lumen Mayor	142.8685	103.7801	1241.7687
	Solidez	Compacidad	
Lumen Mayor	0.4871	0.115	