

Modelo de Machine learning - Detección de Fases de la Luna

FACULTAD DE INGENIERÍA

UNIVERSIDAD DE ANTIOQUIA

Departamento de Electrónica y Telecomunicaciones

Semestre 2025-1

Segundo trabajo

Ferney Mejía Pérez

Harold José Urquijo Durán

ferney.mejiap@udea.edu.co

Harold.urquijo@udea.edu.co

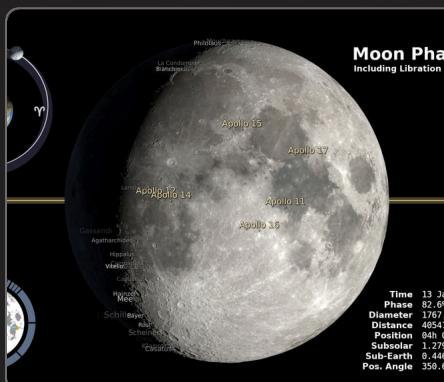
Implementación de Aprendizaje Automático para la Clasificación de Fases Lunares

En cuanto a la descripción del problema, este [proyecto](#) aplica aprendizaje automático para clasificar imágenes de las fases lunares: Llena, Menguante, Creciente y Nueva.

Se emplearán técnicas avanzadas de procesamiento de imágenes para mejorar la precisión del modelo.

Referencias:

1. Artificial intelligence-based lunar crater detection and classification.
2. In-Space Radiometric Calibration of Nanosatellite Camera.
3. Patterns of overlapping habitat use of juvenile white shark and human recreational water users along southern California beaches



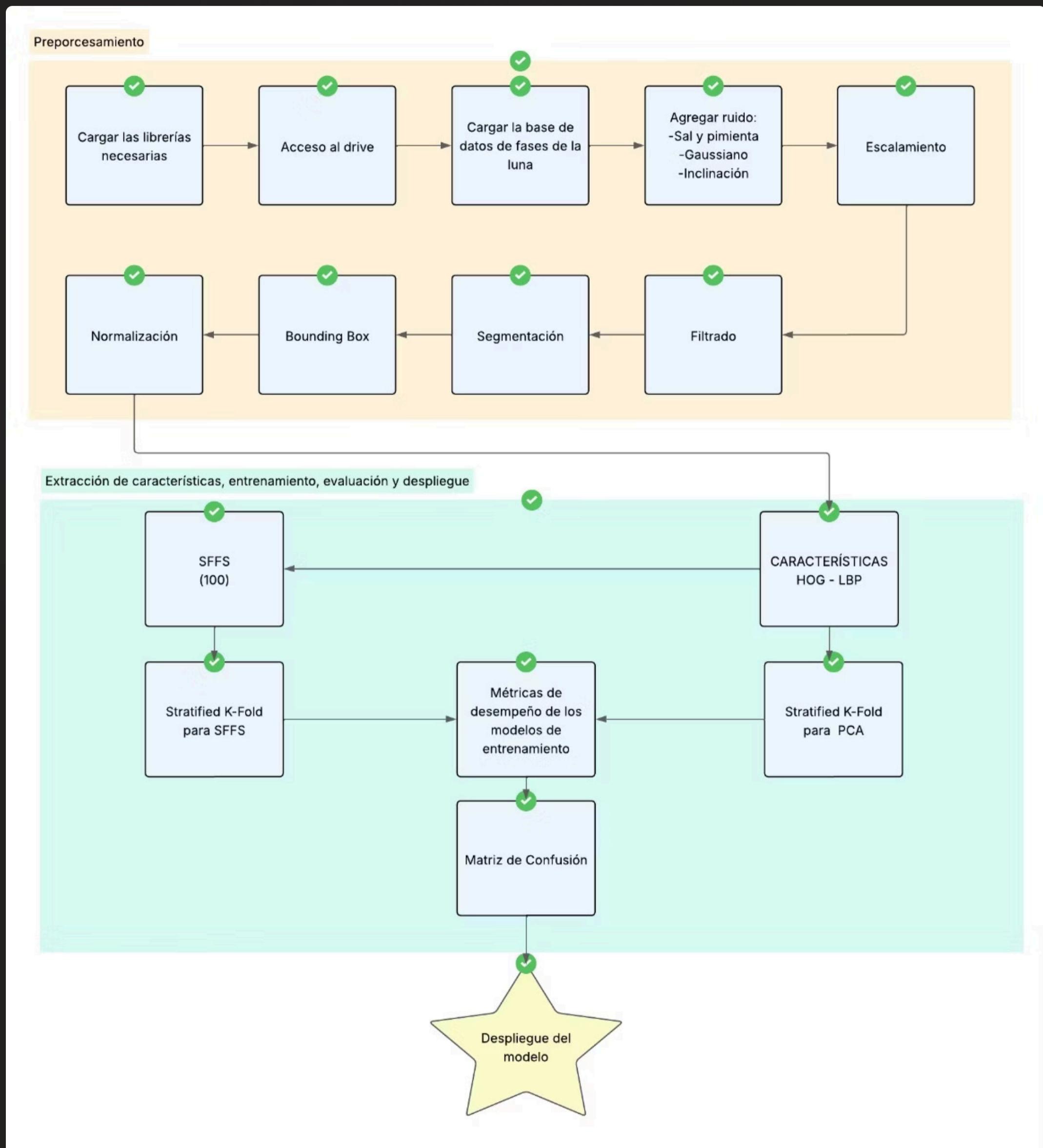
NASA Scientific Visualization Studio



[NASA Scientific Visualization Studio |...](#)

Dial-A-Moon || moon.0001.jpg (730x730) ||
comp.0001.tif (5760x3240) || ||

Procedimiento implementado



Conceptos del curso:



Agregación de ruido:

En esta sección de Agregación de ruido, se complementa cada categoría permitiendo que cada una quede con 100 imágenes con formato .jpg, donde el PASO 1, se encarga de definir una función que agrega ruido, dicha función recibe como parámetros 1 imagen y 1 entero, lo cual dependiendo de su valor puede agregarse:

- 1: Ruido Gaussiano
- 2: Ruido Sal y pimienta
- 3: Se inclina la imagen levemente
- 4: Se Desplaza la imagen levemente hacia la derecha



Escalado y Filtrado:

En la sección correspondiente al escalamiento, se busca llevar todas las imágenes de la base de datos a un tamaño uniforme. Esto es necesario debido a que las imágenes originales presentan variaciones de resolución.

En la sección de Filtrado se tiene como objetivo principal procesar las imágenes de la luna aplicándoles filtros para eliminar el ruido y mejorar su calidad, los filtros empleados son el Gaussiano y el de Mediana.



Segmentación:

En esta sección, se utiliza el método de umbralización de Otsu, que permite separar de forma automática la luna del fondo en escala de grises. Posteriormente, se aplican operaciones morfológicas como apertura y cierre, con el fin de eliminar pequeños ruidos y llenar huecos en la región de la luna.



Bounding Box:

Por último, se utiliza la técnica llamada "Bounding box", la cual consiste en delimitar un objeto de interés dentro de una imagen mediante un rectángulo (o polígono) que lo encierra. Esto se hace al aplicar filtros/kernels (pequeñas matrices) que recorren la imagen para extraer patrones.

Montaje en funcionamiento:

Agregación de ruido:

```
def agregar_ruido(imagen, opcion): """ imagen: numpy array o imagen PIL convertida a array opcion: 1 - ruido gaussiano 2 - ruido sal y pimienta 3 - inclinación (rotación leve) 4 - desplazamiento a la derecha """
# Convertir imagen a array si es PIL
if isinstance(imagen, Image.Image):
    imagen = np.array(imagen)

match opcion:
    case 1:
        # Ruido gaussiano
        img = random_noise(imagen, mode='gaussian', mean=0, var=0.01)
        img = img_as_ubyte(img)
        return img

    case 2:
        # Ruido sal y pimienta
        img = random_noise(imagen, mode='s&p', amount=0.05)
        img = img_as_ubyte(img)
        return img

    case 3:
        # Rotación leve (por ejemplo, 10 grados)
        img_pil = Image.fromarray(imagen)
        img_pil = img_pil.convert("RGB")
        img_rotada = img_pil.rotate(10, resample=Image.BICUBIC, expand=True, fillcolor=(0, 0, 0))
        return np.array(img_rotada)

    case 4:
        # Desplazamiento a la derecha
        desplazamiento_pixeles = 50
        if len(imagen.shape) == 2:
            filas, cols = imagen.shape
        else:
            filas, cols, _ = imagen.shape
        M = np.float32([[1, 0, desplazamiento_pixeles], [0, 1, 0]])
        img_desplazada = cv2.warpAffine(imagen, M, (cols, filas), borderMode=cv2.BORDER_CONSTANT,
                                         borderValue=0)
        return img_desplazada

    case _:
        raise ValueError("Opción no válida. Usa 1 (gaussiano), 2 (sal y pimienta), 3 (inclinación), o 4 (desplazamiento)")
```



Montaje en funcionamiento:

Escalamiento:

```
# Tamaño adecuado
nuevo_tamano = (256, 256)

# Rutas
ruta_base_original = "/content/drive/MyDrive/Fases_luna"
ruta_base_nueva = "/content/drive/MyDrive/Fases_luna_preprocesadas"

# Crear carpeta base si no existe
os.makedirs(ruta_base_nueva, exist_ok=True)

# Fases lunares
fases = ["Creciente", "Llena", "Menguante", "Nueva"]

for fase in fases:
    ruta_fase_origen = os.path.join(ruta_base_original, fase)
    ruta_fase_destino = os.path.join(ruta_base_nueva, fase)
    os.makedirs(ruta_fase_destino, exist_ok=True)

    # Filtrar solo archivos .jpg cuyo nombre es un número
    archivos = [
        f for f in os.listdir(ruta_fase_origen)
        if f.endswith(".jpg") and f.split('.')[0].isdigit()
    ]

    # Ordenar por número
    archivos = sorted(archivos, key=lambda x: int(x.split('.')[0]))

    for archivo in archivos:
        ruta_imagen = os.path.join(ruta_fase_origen, archivo)
        imagen = Image.open(ruta_imagen)

        # Redimensionar
        imagen_redimensionada = imagen.resize(nuevo_tamano)

        # Guardar imagen escalada
        ruta_destino = os.path.join(ruta_fase_destino, archivo)
        imagen_redimensionada.save(ruta_destino)

print(f" {fase} escalada y guardada en: {ruta_fase_destino}")
```

Imagen Luna Llena Escalada



Imagen Luna Nueva Escalada



Imagen Luna Menguante Escalada



Imagen Luna Creciente Escalada



Montaje en funcionamiento:

Filtrado:

```
# Rutas base
ruta_escaladas = "/content/drive/MyDrive/Fases_luna_preprocesadas"
ruta_filtradas = "/content/drive/MyDrive/Fases_luna_filtradas"

# Crear carpeta base si no existe
os.makedirs(ruta_filtradas, exist_ok=True)

# Filtro: parámetros
tamaño_kernel_gauss = (3, 3)
tamaño_kernel_mediana = 3

# Rango de imágenes a procesar
rango_inicio = 1
rango_fin = 100

# Fases
fases = ["Creciente", "Llena", "Menguante", "Nueva"]

for fase in fases:
    ruta_origen = os.path.join(ruta_escaladas, fase)
    ruta_destino = os.path.join(ruta_filtradas, fase)
    os.makedirs(ruta_destino, exist_ok=True)

    archivos = sorted([
        f for f in os.listdir(ruta_origen)
        if f.endswith(".jpg") and f.split(".")[0].isdigit()
    ], key=lambda x: int(x.split(".")[0]))

    for archivo in archivos:
        numero = int(archivo.split(".")[0])

        if numero < rango_inicio or numero > rango_fin:
            continue # saltar imágenes fuera del rango

        ruta_img = os.path.join(ruta_origen, archivo)
        imagen = cv2.imread(ruta_img)

        # Filtro gaussiano
        imagen_gauss = cv2.GaussianBlur(imagen, tamaño_kernel_gauss, 0)

        # Filtro de mediana
        imagen_filtrada = cv2.medianBlur(imagen_gauss, tamaño_kernel_mediana)

        # Guardar imagen filtrada
        ruta_guardado = os.path.join(ruta_destino, archivo)
        cv2.imwrite(ruta_guardado, imagen_filtrada)

print(f" {fase} filtrada y guardada en: {ruta_destino}")
```



Montaje en funcionamiento:

Segmentación:

```
# Rutas
ruta_filtradas = "/content/drive/MyDrive/Fases_luna_filtradas"
ruta_segmentadas = "/content/drive/MyDrive/Fases_luna_segmentadas"

# Fases
fases = ["Creciente", "Llena", "Menguante", "Nueva"]

for fase in fases:
    ruta_origen = os.path.join(ruta_filtradas, fase)
    ruta_destino = os.path.join(ruta_segmentadas, fase)
    os.makedirs(ruta_destino, exist_ok=True)

archivos = sorted([
    f for f in os.listdir(ruta_origen)
    if f.endswith(".jpg") and f.split(".")[0].isdigit()
], key=lambda x: int(x.split(".")[0]))

for archivo in archivos:
    ruta_img = os.path.join(ruta_origen, archivo)
    imagen = cv2.imread(ruta_img, cv2.IMREAD_GRAYSCALE)

    # Segmentación con Otsu
    _, binaria = cv2.threshold(imagen, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    # Apertura para eliminar pequeños ruidos
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    binaria = cv2.morphologyEx(binaria, cv2.MORPH_OPEN, kernel, iterations=1)

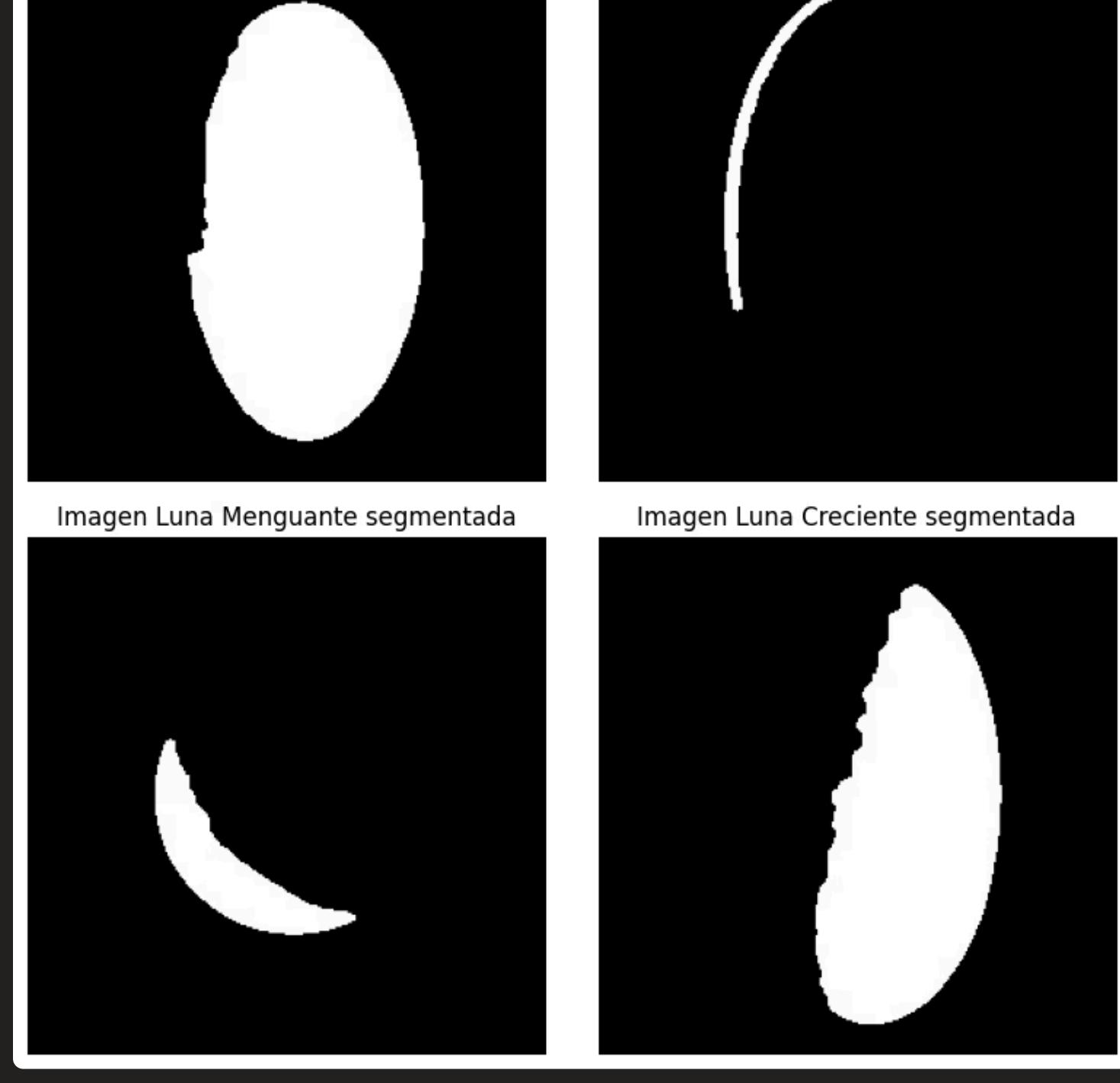
    # Cierre para rellenar huecos dentro de la luna
    binaria = cv2.morphologyEx(binaria, cv2.MORPH_CLOSE, kernel, iterations=1)

    # Detección de contornos para extraer el objeto más grande
    contornos, _ = cv2.findContours(binaria, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if contornos:
        # Elegir el contorno más grande (la luna)
        contorno_max = max(contornos, key=cv2.contourArea)
        mascara = np.zeros_like(binaria)
        cv2.drawContours(mascara, [contorno_max], -1, 255, thickness=cv2.FILLED)

        # Aplicar máscara a la imagen binaria
        imagen_segmentada = cv2.bitwise_and(binaria, mascara)
    else:
        imagen_segmentada = binaria # Si no hay contornos, guardar binaria tal cual

    # Guardar imagen segmentada
    ruta_guardado = os.path.join(ruta_destino, archivo)
    cv2.imwrite(ruta_guardado, imagen_segmentada)

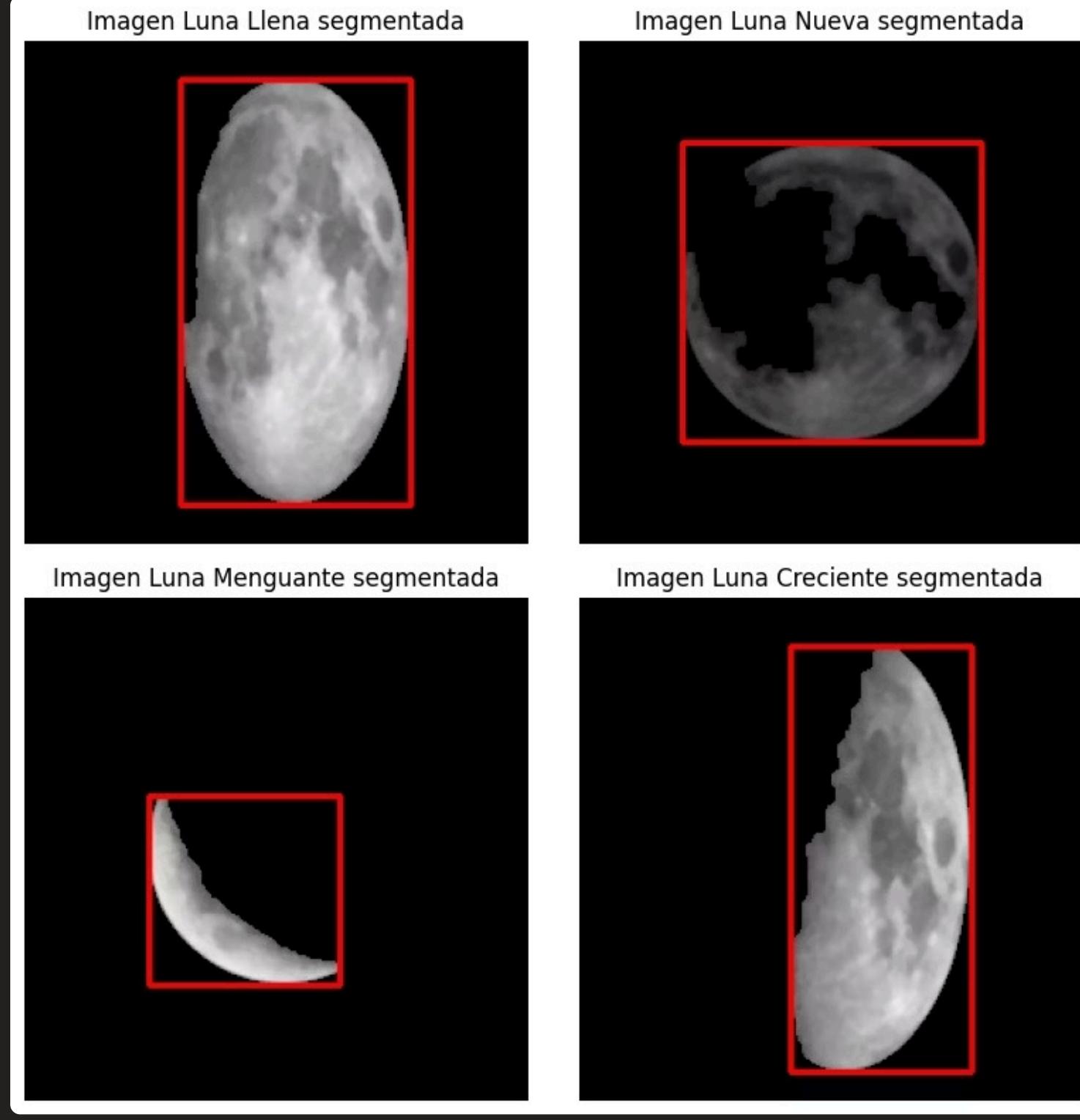
print(f"Segmentación de {fase} completada en: {ruta_destino}")
```



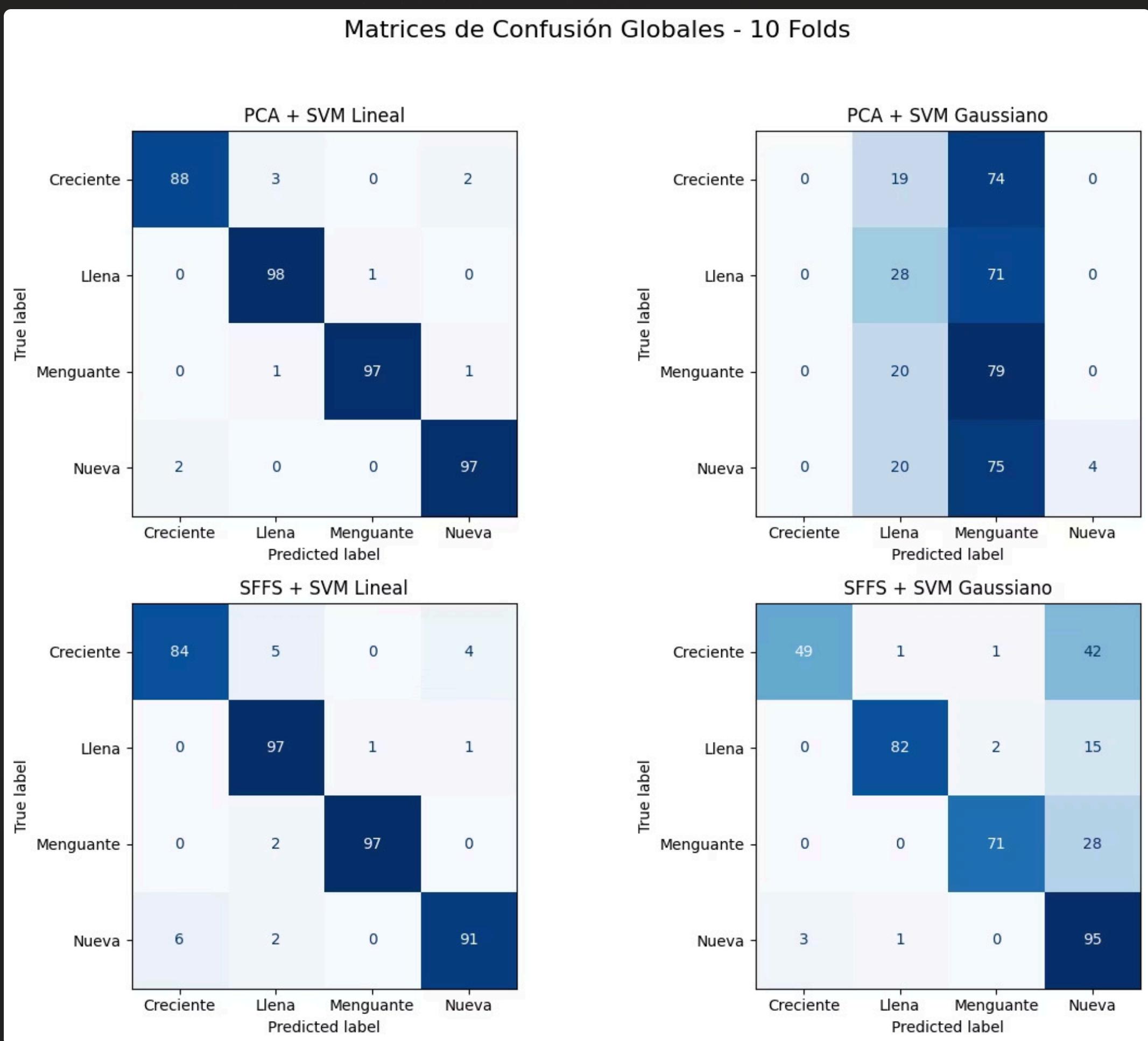
Montaje en funcionamiento:

Bounding Box:

```
#Aplicación de Bounding Box:  
clases = ["Creciente", "Llena", "Menguante", "Nueva"]  
base_path = "/content/drive/MyDrive/Fases_luna_segmentadas_final"  
img_size = (300, 300) # tamaño de visualización, no redimensionamos la imagen real  
  
#Almacenado en drive:  
# Nueva ruta para almacenar las imágenes con bounding box  
output_base_path = "/content/drive/MyDrive/Fases_luna_segmentadas_BoundingBox"  
os.makedirs(output_base_path, exist_ok=True)  
  
# Recorremos cada clase  
for clase in clases:  
    clase_input_path = os.path.join(base_path, clase)  
    clase_output_path = os.path.join(output_base_path, clase)  
    os.makedirs(clase_output_path, exist_ok=True)  
  
    for nombre_archivo in os.listdir(clase_input_path):  
        ruta_entrada = os.path.join(clase_input_path, nombre_archivo)  
        ruta_salida = os.path.join(clase_output_path, nombre_archivo)  
  
        # Leer y procesar la imagen  
        img = cv2.imread(ruta_entrada)  
        if img is None:  
            print(f"No se pudo leer la imagen: {ruta_entrada}")  
            continue  
  
        img_color = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  
        img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  
  
        _, thresh = cv2.threshold(img_gray, 30, 255, cv2.THRESH_BINARY)  
        contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)  
  
        if contours:  
            contorno_principal = max(contours, key=cv2.contourArea)  
            x, y, w, h = cv2.boundingRect(contorno_principal)  
            cv2.rectangle(img_color, (x, y), (x + w, y + h), (255, 0, 0), 2)  
  
            # Guardar imagen procesada  
            cv2.imwrite(ruta_salida, cv2.cvtColor(img_color, cv2.COLOR_RGB2BGR))  
  
print("Todas las imágenes han sido procesadas y guardadas con bounding boxes.")
```



Resultados



Técnica	Kernel	Accuracy	F1 Score	Recall	Especificidad
0	PCA	Lineal	0.9744 ± 0.0199	0.9742 ± 0.0198	0.9739 ± 0.0201
1	PCA	Gaussiano	0.2846 ± 0.0506	0.1556 ± 0.0713	0.2825 ± 0.0448
2	SFFS	Lineal	0.9462 ± 0.0179	0.9456 ± 0.0176	0.9456 ± 0.0181
3	SFFS	Gaussiano	0.7615 ± 0.0726	0.7636 ± 0.0715	0.7589 ± 0.0707

LINEAS FUTURAS:

- Implementar modelos de deep learning más avanzados (como EfficientNet o ResNet) para mejorar la precisión y robustez en la clasificación de fases lunares.
- Desarrollar un sistema de evaluación continua con validación cruzada y monitoreo en tiempo real para ajustar automáticamente hiperparámetros y detectar sobreajuste o degradación del modelo.
- Realizar pruebas de robustez ante condiciones adversas (nubes, baja iluminación, ruido extremo) e incluir módulos de preprocessamiento adaptativo para mantener la precisión en entornos reales variables.

Referencias

- NASA Scientific Visualization Studio. Moon Phase and Libration, 2023. Recuperado de <https://svs.gsfc.nasa.gov/4955>
- O. Knuutila, E. Kallio, N. Partamies, et al. In-Space Radiometric Calibration of Nanosatellite Camera. Journal of Small Satellites, vol. 11, no. 3, pp. 1165–1186, 2022. Recuperado de <https://jossonline.com/storage/2022/10/> Final-Knuutila-In-Space-Radiometric-Calibration-of-Nanosatellite-Camera.pdf
- C. Stallo, H. Bookmap, D. Cretoni, et al. Integration between Communication, Navigation and Sensing for Space Applications: Case Study on Lunar Satellite Navigation System with Focus on ODTs Techniques. A Roadmap to Future Space Connectivity, pp. 243–270, 2023. Recuperado de https://link.springer.com/chapter/10.1007/978-3-031-30762-1_11
- F. Xing, Y. Yuan, et al. Artificial intelligence-based lunar crater detection and classification. Scientific Reports, vol. 13, 2023. Recuperado de <https://www.nature.com/articles/s41598-023-32807-x>
- C. Lowe, A. Pezner, S. Anderson, et al. Patterns of Overlapping Habitat Use of Juvenile White Shark and Human Recreational Water Users along Southern California Beaches. PLOS ONE, vol. 18, no. 6, e0286575, 2023. Recuperado de <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0286575>