



**Universidad Autónoma de San Luis  
Potosí**



**Facultad de Ingeniería  
Área Mecánica y Eléctrica**

**Tratamiento de imágenes**

**Hora de la Materia:**

18:00 – 19:00, 20:00

**Nombre del profesor:**

Jully Kado Mercado Elías

**Proyecto #2**

**Nombre del alumno:**

Martínez Carrillo Carlos Ignacio

315671

# Índice

Introducción.....	3
Desarrollo.....	3
Explicación del código.....	3
Justificación de parámetros.....	5
Posibles mejoras.....	6
Resultados.....	6
Conclusiones.....	7
Referencias y bibliografía.....	7

## Introducción

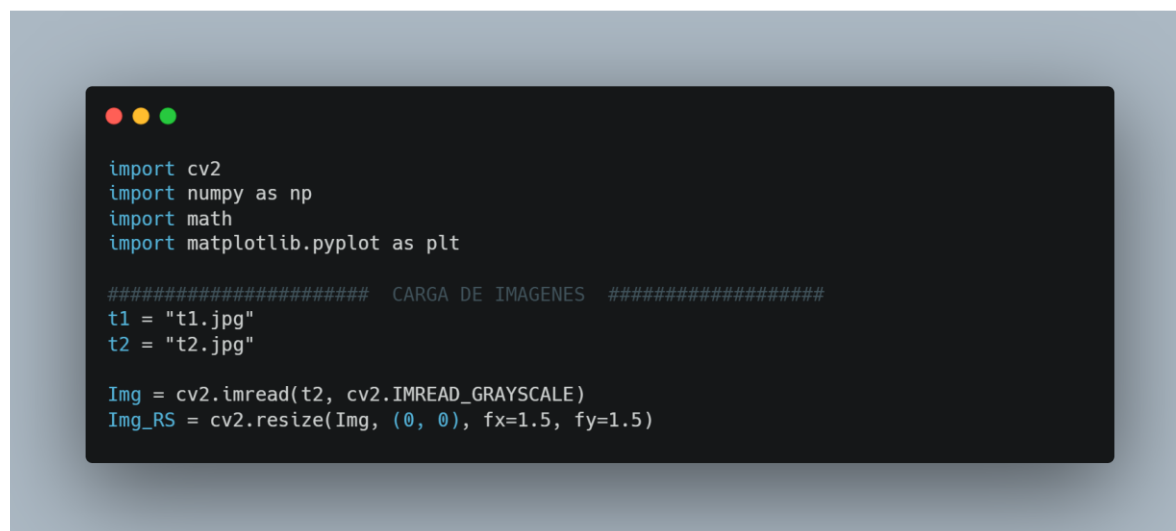
Mediante el presente proyecto, se realizará un programa para la detección de tumores en radiografías utilizando el conocimiento adquirido en la materia, siendo entre estos filtros, segmentación, bordes, contornos y regiones.

## Desarrollo

### Explicación del código

Para el programa en cuestión, primero se deben cargar las librerías a utilizar, siendo estas “cv2”, “matplotlib.pyplot” y “numpy”.

Seguido de esto se debe hacer la lectura de la imagen mediante su ruta.

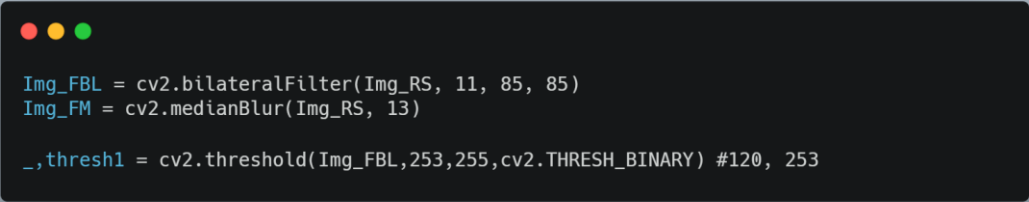


*Ilustración 1: carga de librerías e imágenes.*

Con el fin de reducir el ruido, se aplica un filtro bilateral con tamaño de kernel de 11 y desviaciones de 85 y 85.

Una vez hecho esto, se aplica un umbral con valor dependiendo de la imagen, ya que cada una tiene un valor de píxel diferente en el tumor. Se hace un umbral binario donde los valores que cumplan con la condición serán tomados como 255 o blanco.

```


    Img_FBL = cv2.bilateralFilter(Img_RS, 11, 85, 85)
    Img_FM = cv2.medianBlur(Img_RS, 13)

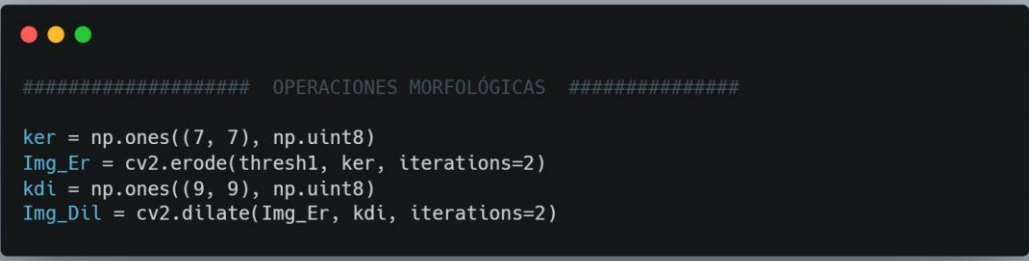
    _,thresh1 = cv2.threshold(Img_FBL,253,255,cv2.THRESH_BINARY) #120, 253

```

*Ilustración 2: Aplicación de filtro y umbral.*

Se aplican operaciones morfológicas para eliminar las regiones pequeñas que no son de interés, y después retomar el tamaño de la región que sí lo hace.

```


    ##### OPERACIONES MORFOLÓGICAS #####

    ker = np.ones((7, 7), np.uint8)
    Img_Er = cv2.erode(thresh1, ker, iterations=2)
    kdi = np.ones((9, 9), np.uint8)
    Img_Dil = cv2.dilate(Img_Er, kdi, iterations=2)

```

*Ilustración 3: Operaciones morfológicas.*

Se aplica el método “cv2.Canny()” para la obtención de bordes y contornos del tumor, el método “cv2.findContours()” para obtener claramente los contornos generados, y el método “cv2.drawContours()” para dibujar los contornos sobre la imagen original.

```
##### BORDES Y CONTORNOS #####
Img_canny = cv2.Canny(Img_Dil, 100, 200)
copy = Img_RS.copy()
copy = cv2.cvtColor(copy, cv2.COLOR_GRAY2BGR)

contours, _ = cv2.findContours(Img_canny, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
copy = cv2.cvtColor(Img_RS, cv2.COLOR_GRAY2BGR)
cv2.drawContours(copy, contours, -1, (200, 0, 200), 2)
```

*Ilustración 4: Obtención de bordes y contornos.*

Una vez hecho todo este proceso, se hace el código para mostrar las imágenes en una sola ventana.

```
titles = ['Imagen Original', 'Máscara de tumor', 'Detección de Tumor']
images = [Img_RS, Img_Dil, copy]

plt.figure(figsize=(10, 5))
for i in range(3):
    plt.subplot(1, 3, i + 1)
    plt.imshow(images[i], cmap='gray' if i < 2 else None)
    plt.title(titles[i])
    plt.xticks([], plt.yticks([]))

plt.show()
```

*Ilustración 5: Muestra de imágenes.*

Una vez hecho todo este proceso, se hace el código para mostrar las imágenes en una sola ventana.

### **Justificación de parámetros**

Para el filtro se utiliza un kernel de 11 con desviación de 85 por ambas partes debido a que hacía mejor el suavizado del tumor, conservaba los bordes y reducía el ruido.

Para el umbral se tomaron los valores de 120 y 253 para reducir lo más posible las áreas de desinterés sin quitar área al tumor.

Se utilizan tamaños de kernel de 7 y 9 para las operaciones morfológicas ya que estos quitaban las regiones de desinterés sin reducir mucho las zonas de tumor y dilataban de manera correcta las mismas.

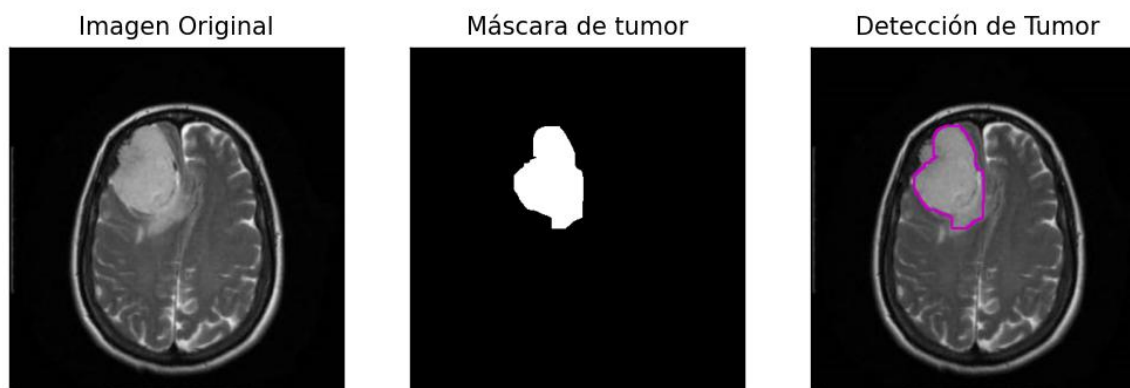
## Posibles mejoras

La detección del tumor no es enteramente fiel a la imagen. Posiblemente sea un error del umbralizado y las operaciones morfológicas, siendo que se pueda usar un valor menor de umbral y operaciones más precisas para una detección más fiel.

## Resultados



*Ilustración 6: Resultado de primera imagen.*



*Ilustración 7: Resultado de segunda imagen.*

## Conclusiones

Las aplicaciones de detección de regiones con tratamiento de imágenes son bastantes, y gracias al proyecto podemos ver que los procesos no siempre son iguales, incluso con estas dos imágenes relativamente parecidas y con la misma aplicación. Cada imagen tiene una composición distinta y el trabajo va enfocado a determinar qué procesos se deben hacer en cada una para obtener el mejor resultado posible.

## Referencias y bibliografía

- *OpenCV: Canny Edge Detection.* (s. f.).  
[https://docs.opencv.org/4.x/da/d22/tutorial\\_py\\_canny.html](https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html)
- *OpenCV: Smoothing images.* (s. f.).  
[https://docs.opencv.org/3.4/dc/dd3/tutorial\\_gaussian\\_median\\_blur\\_bilateral\\_filter.html](https://docs.opencv.org/3.4/dc/dd3/tutorial_gaussian_median_blur_bilateral_filter.html)