6_segmentation_otsu_Guillermo

March 11, 2024

1 Maestría en Inteligencia Artificial Aplicada

2 TC 4033: Visión computacional para imágenes y video

3 Tecnológico de Monterrey

3.1 Dr. Gilberto Ochoa Ruiz

4 # 6. Otsu Thresholding

4.1 Equipo # 16

4.1.1 Edwin David Hernández Alejandre A01794692

4.1.2 Miguel Guillermo Galindo Orozco A01793695

4.1.3 Jorge Pedroza Rivera A01319553

4.1.4 Juan Carlos Alvarado Carricarte A01793486

4.1.5 Gerardo Aaron Castañeda Jaramillo A01137646

4.2 Table of Contents

Thresholding is used to create a binary image from a grayscale image

4.3 Importing Libraries

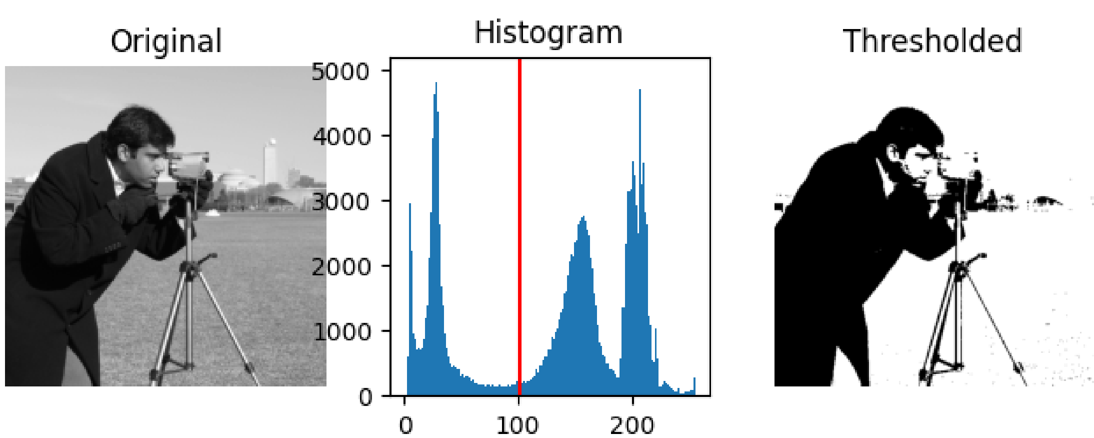
[1]:
```
import matplotlib.pyplot as plt
from skimage import data
from skimage.filters import threshold_otsu
from skimage.filters import threshold_multiotsu
import numpy as np
import cv2
```

4.4 Single Thresholding

We illustrate how to apply one of these thresholding algorithms. Otsu's method [2]_ calculates an "optimal" threshold (marked by a red line in the histogram below) by maximizing the variance between two classes of pixels, which are separated by the threshold. Equivalently, this threshold minimizes the intra-class variance.

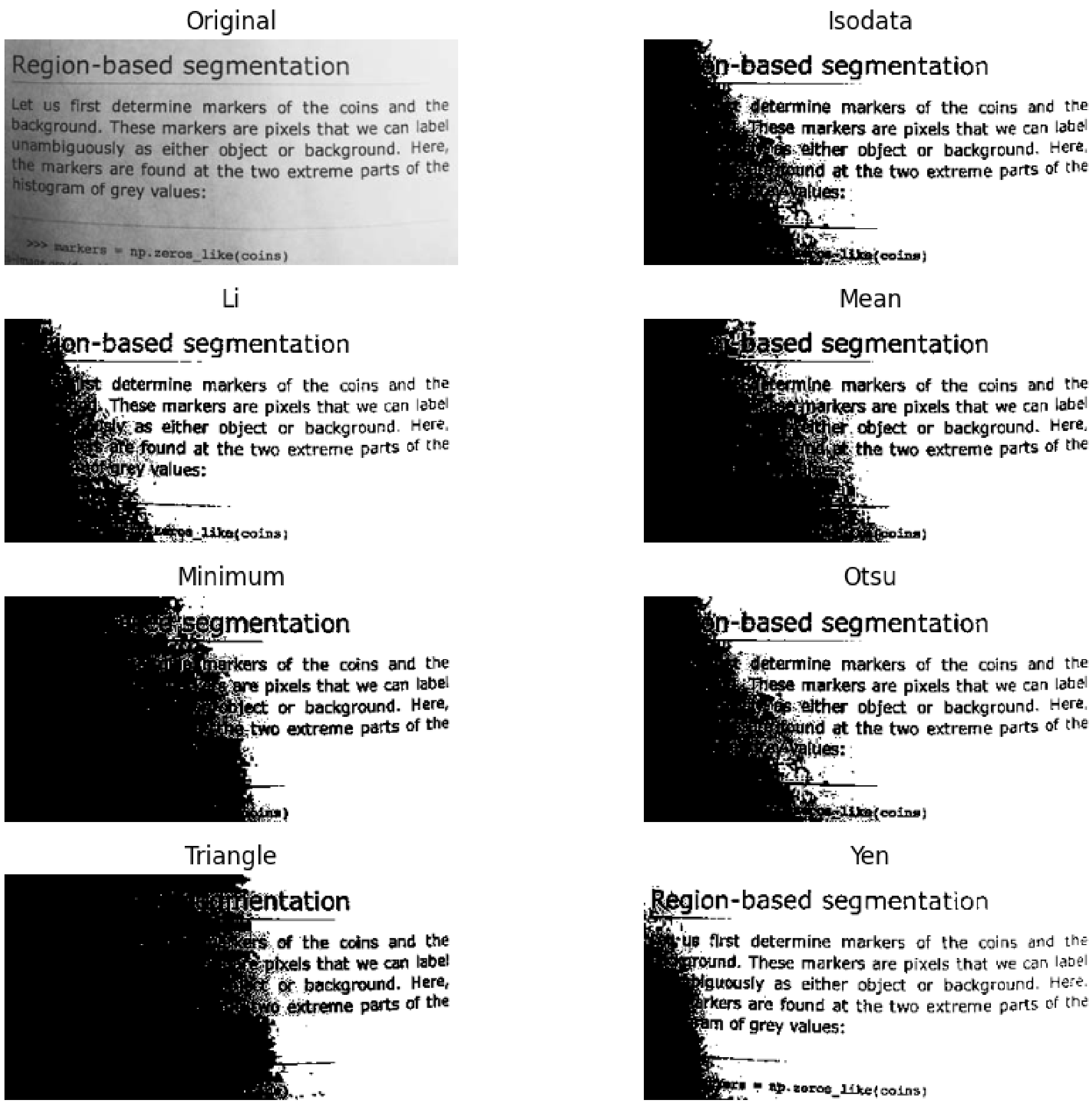
[4]:
```
image = data.camera()
thresh = threshold_otsu(image)
binary = image > thresh

fig, axes = plt.subplots(ncols=3, figsize=(8, 2.5))
ax = axes.ravel()
ax[0] = plt.subplot(1, 3, 1)
ax[1] = plt.subplot(1, 3, 2)
ax[2] = plt.subplot(1, 3, 3, sharex=ax[0], sharey=ax[0])

ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title('Original')
ax[0].axis('off')

ax[1].hist(image.ravel(), bins=256)
ax[1].set_title('Histogram')
ax[1].axvline(thresh, color='r')

ax[2].imshow(binary, cmap=plt.cm.gray)
ax[2].set_title('Thresholded')
ax[2].axis('off')

plt.show()
```

```
<ipython-input-4-ce81723eaa7b>:9: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases
later; explicitly call ax.remove() as needed.
  ax[2] = plt.subplot(1, 3, 3, sharex=ax[0], sharey=ax[0])
```



If you are not familiar with the details of the different algorithms and the underlying assumptions, it is often difficult to know which algorithm will give the best results. Therefore, Scikit-image includes a function to evaluate thresholding algorithms provided by the library. At a glance, you can select the best algorithm for your data without a deep understanding of their mechanisms.

[5]:
```python
from skimage.filters import try_all_threshold

img = data.page()

fig, ax = try_all_threshold(img, figsize=(10, 8), verbose=False)
plt.show()
```



[6]:

```
# TODO: Slide image for segmentation, alpha blerding
```

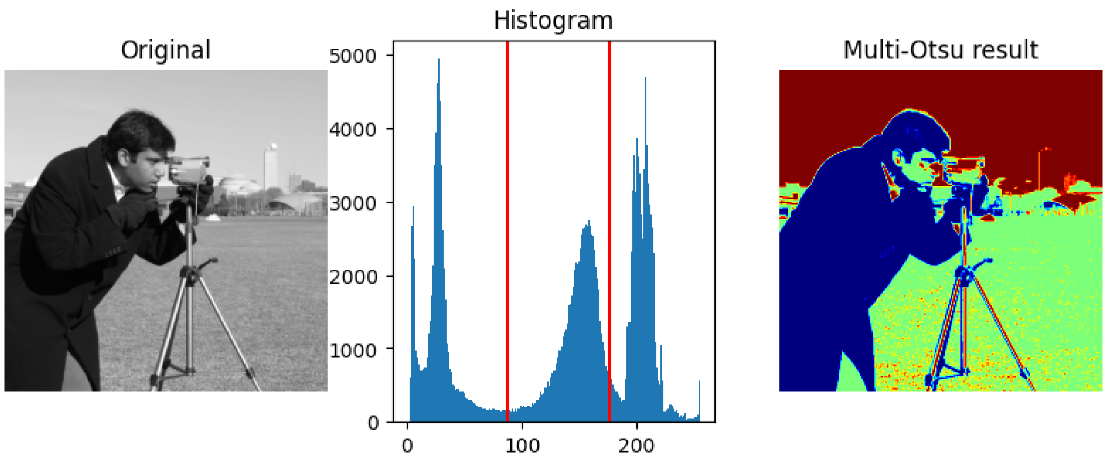
4.5 Multi Thresholding

The multi-Otsu threshold is a thresholding algorithm that is used to separate the pixels of an input image into several different classes, each one obtained according to the intensity of the gray levels within the image.

Multi-Otsu calculates several thresholds, determined by the number of desired classes. The default number of classes is 3: for obtaining three classes, the algorithm returns two threshold values. They are represented by a red line in the histogram below.

[7]:

```
# The input image.
image = data.camera()

# Applying multi-Otsu threshold for the default value, generating
# three classes.
thresholds = threshold_multiotsu(image)

# Using the threshold values, we generate the three regions.
regions = np.digitize(image, bins=thresholds)

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(10, 3.5))

# Plotting the original image.
ax[0].imshow(image, cmap='gray')
ax[0].set_title('Original')
ax[0].axis('off')

# Plotting the histogram and the two thresholds obtained from
# multi-Otsu.
ax[1].hist(image.ravel(), bins=255)
ax[1].set_title('Histogram')
for thresh in thresholds:
    ax[1].axvline(thresh, color='r')

# Plotting the Multi Otsu result.
ax[2].imshow(regions, cmap='jet')
ax[2].set_title('Multi-Otsu result')
ax[2].axis('off')

plt.subplots_adjust()

plt.show()
```



5 Ejercicio

6 1.

Experimenta con diferentes imagenes ademas de las provistas en en Colab, identifca imagenes con diferentes backgrounds y estilos, cuales son las limitaciones de single thresholding contra el algoritmo de Otsu

Comenzamos creando una función para aplicar el método de otsu para 1 solo threshold:

[8]:
```
def single_tres_otsu(img):
    thresh = threshold_otsu(image)
    binary = image > thresh

    fig, axes = plt.subplots(ncols=3, figsize=(8, 2.5))
    ax = axes.ravel()
    ax[0] = plt.subplot(1, 3, 1)
    ax[1] = plt.subplot(1, 3, 2)
    ax[2] = plt.subplot(1, 3, 3, sharex=ax[0], sharey=ax[0])

    ax[0].imshow(image, cmap=plt.cm.gray)
    ax[0].set_title('Original')
    ax[0].axis('off')

    ax[1].hist(image.ravel(), bins=256)
    ax[1].set_title('Histogram')
    ax[1].axvline(thresh, color='r')

    ax[2].imshow(binary, cmap=plt.cm.gray)
    ax[2].set_title('Thresholded')
    ax[2].axis('off')
```

```
    plt.subplots_adjust()

    plt.show()
```

Cargamos 6 imágenes utilizadas en actividades previas:

[9]:
```
image_1 = cv2.imread('/content/image_1.jpg', cv2.IMREAD_GRAYSCALE)
image_2 = cv2.imread('/content/image_2.jpg', cv2.IMREAD_GRAYSCALE)
image_3 = cv2.imread('/content/image_3.jpg', cv2.IMREAD_GRAYSCALE)
image_4 = cv2.imread('/content/image_4.jpg', cv2.IMREAD_GRAYSCALE)
image_5 = cv2.imread('/content/image_5.jpg', cv2.IMREAD_GRAYSCALE)
image_6 = cv2.imread('/content/image_6.jpg', cv2.IMREAD_GRAYSCALE)

images = []

images = [image_1, image_2, image_3, image_4, image_5, image_6]
```

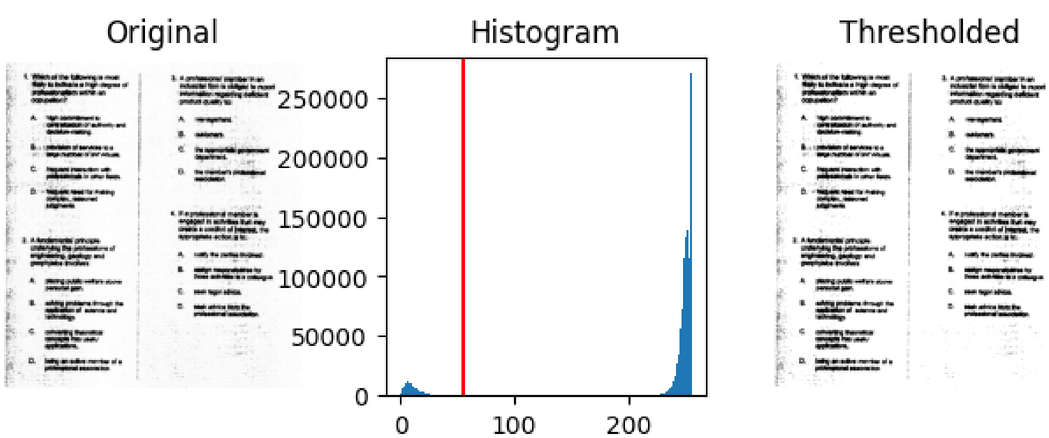
Aplicamos median filter a un par de ellas para mejor funcionamiento:

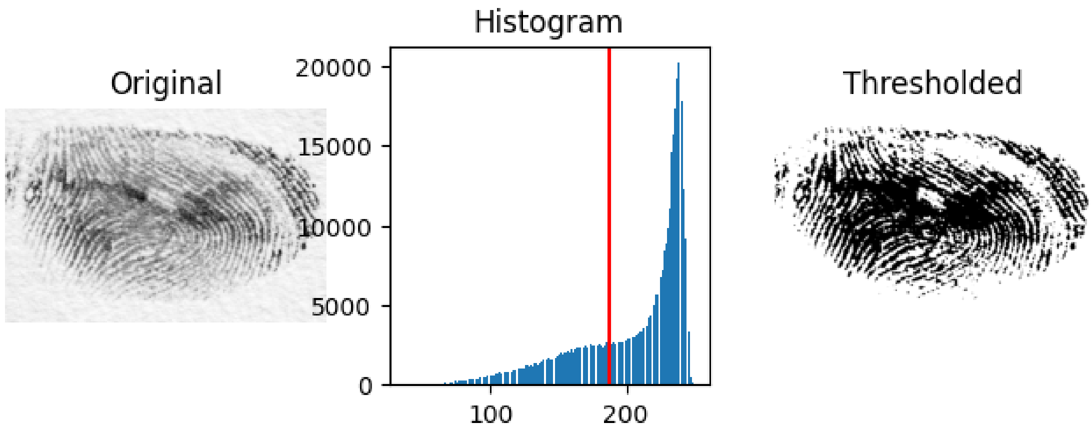
[10]:
```
images[0] = cv2.medianBlur(images[0], 5)
images[1] = cv2.medianBlur(images[1], 5)
#images[4] = cv2.medianBlur(images[4], 5)
#images[5] = cv2.medianBlur(images[5], 5)
```

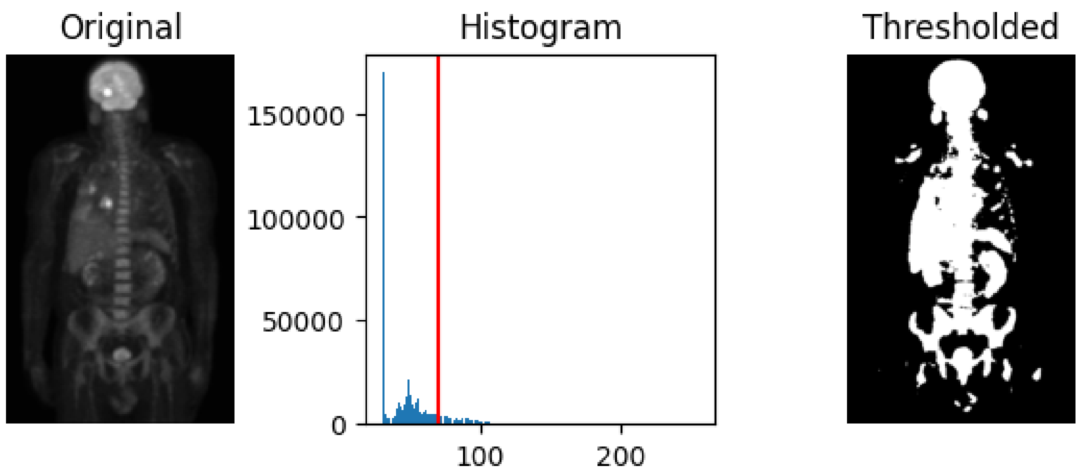
Aplicamos el método a cada una:

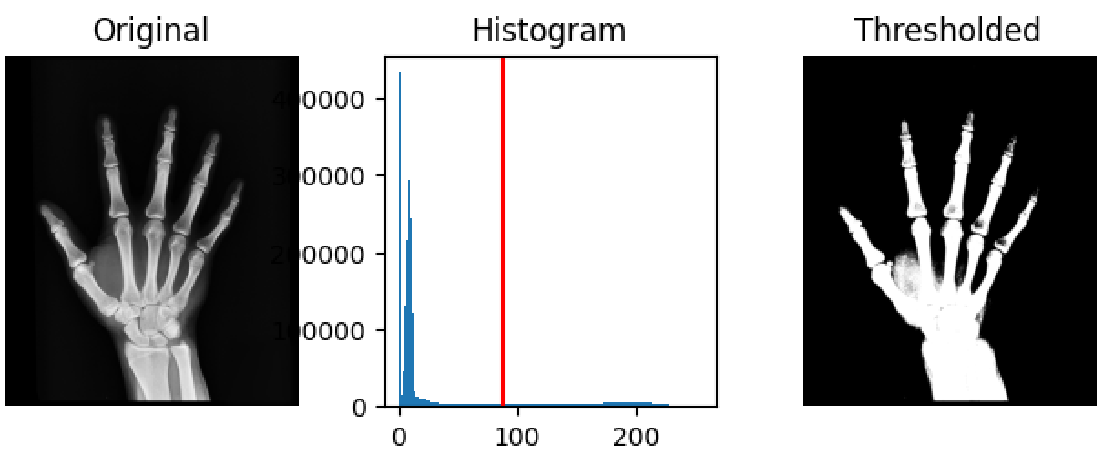
[11]:
```
for image in images:
  single_tres_otsu(image)
```

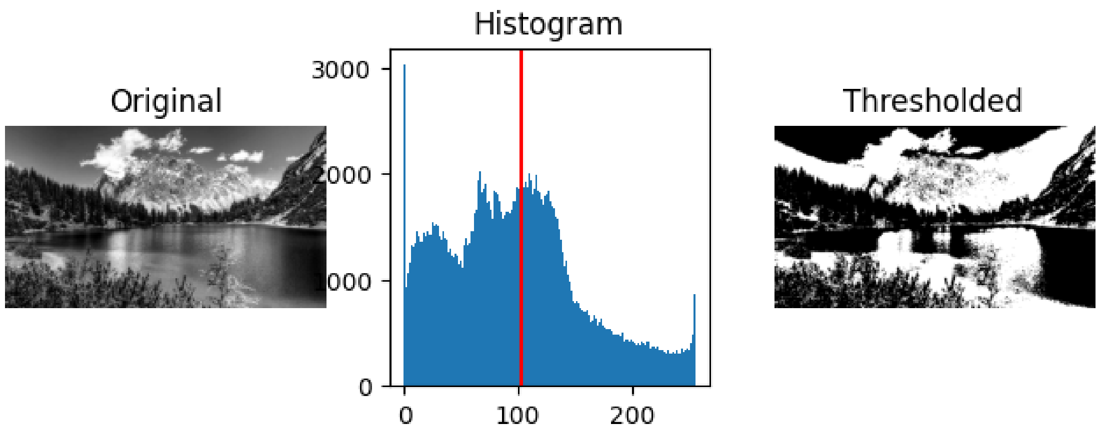
```
<ipython-input-8-1577835ef629>:9: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases
later; explicitly call ax.remove() as needed.
  ax[2] = plt.subplot(1, 3, 3, sharex=ax[0], sharey=ax[0])
```

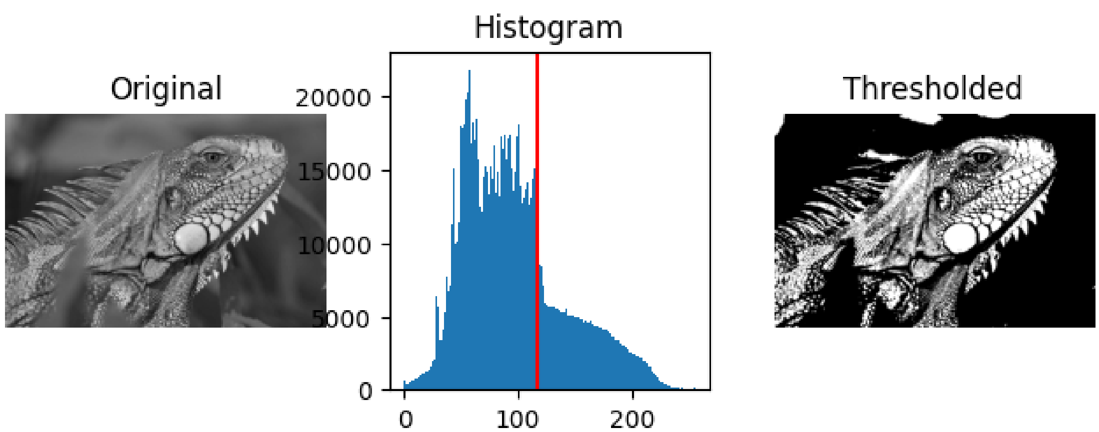



Original
Histogram
Thresholded
20000
15000
10000
5000
0
100
200


Original
Histogram
Thresholded
150000
100000
50000
0
100
200


Original
Histogram
Thresholded
400000
300000
200000
100000
0
0
100
200





Ahora generamos una función para aplicar le método de threshold múltiple:

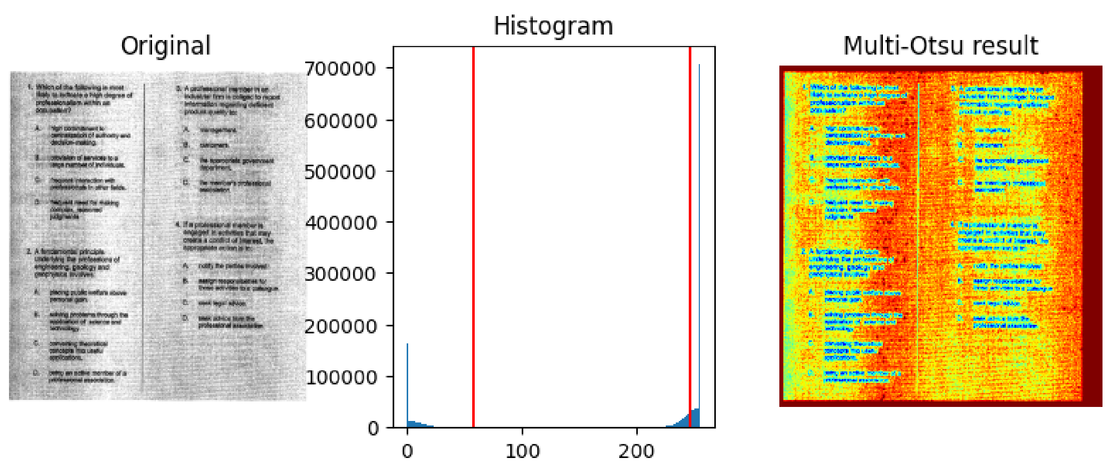
[12]:
```
def multi_tres_otsu(image):
    thresholds = threshold_multiotsu(image)
    regions = np.digitize(image, bins=thresholds)

    fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(10, 3.5))

    ax[0].imshow(image, cmap='gray')
    ax[0].set_title('Original')
    ax[0].axis('off')

    ax[1].hist(image.ravel(), bins=255)
    ax[1].set_title('Histogram')
```

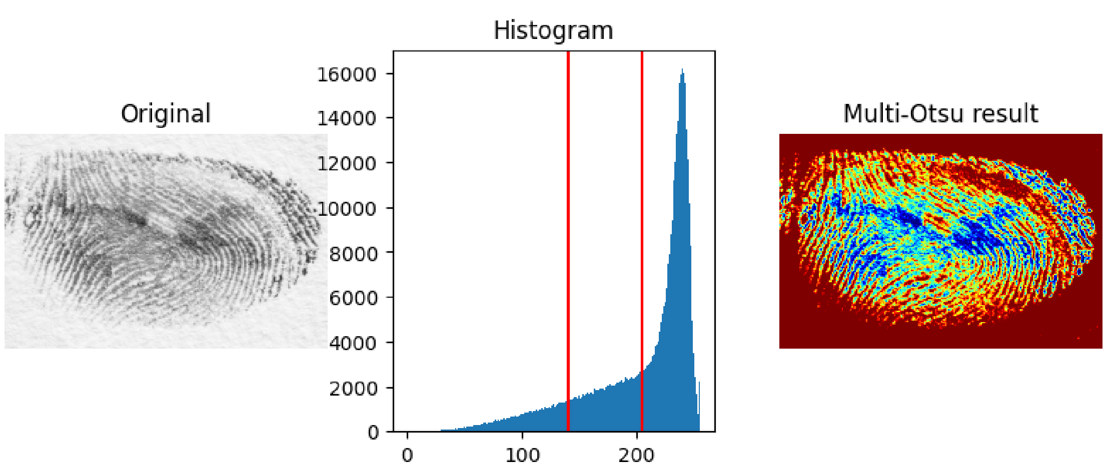
```python
    for thresh in thresholds:
        ax[1].axvline(thresh, color='r')

    ax[2].imshow(regions, cmap='jet')
    ax[2].set_title('Multi-Otsu result')
    ax[2].axis('off')

    plt.subplots_adjust()

    plt.show()
```

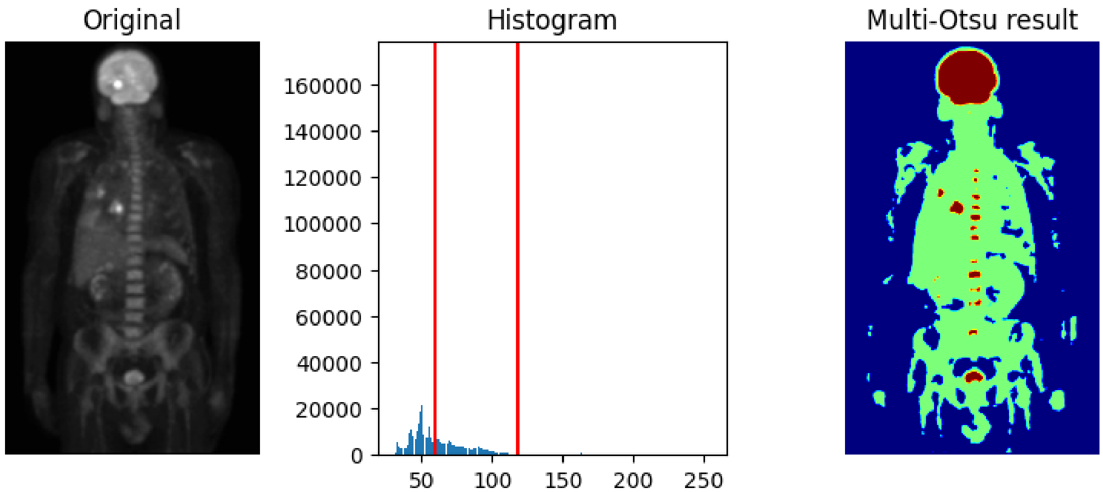
Lo aplicamos a las mismas imágenes:

[13]:
```python
images = [image_1, image_2, image_3, image_4, image_5, image_6]
```

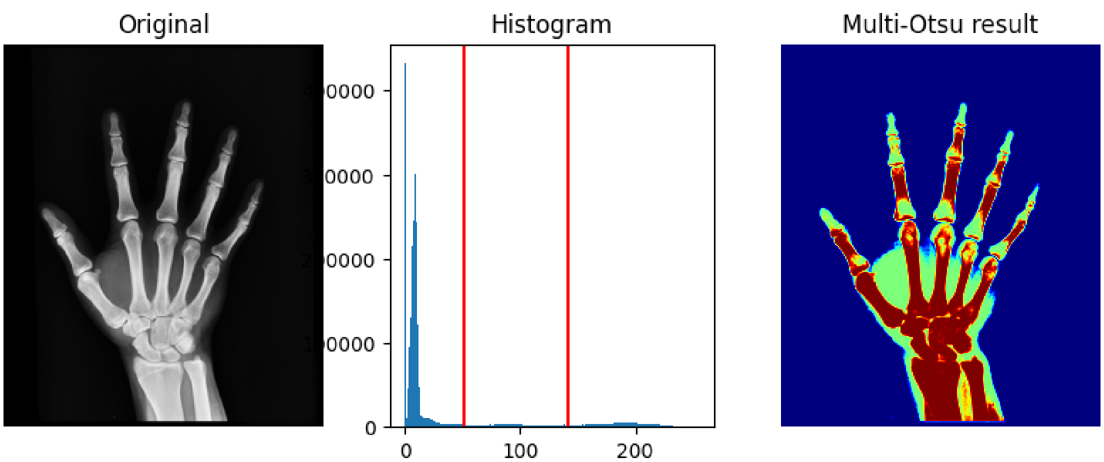
[14]:
```python
for image in images:
    multi_tres_otsu(image)
```



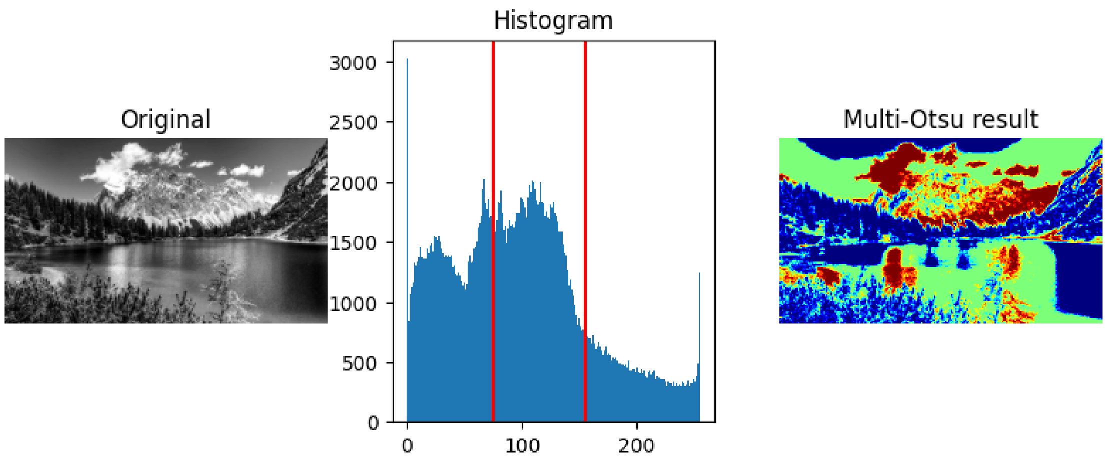


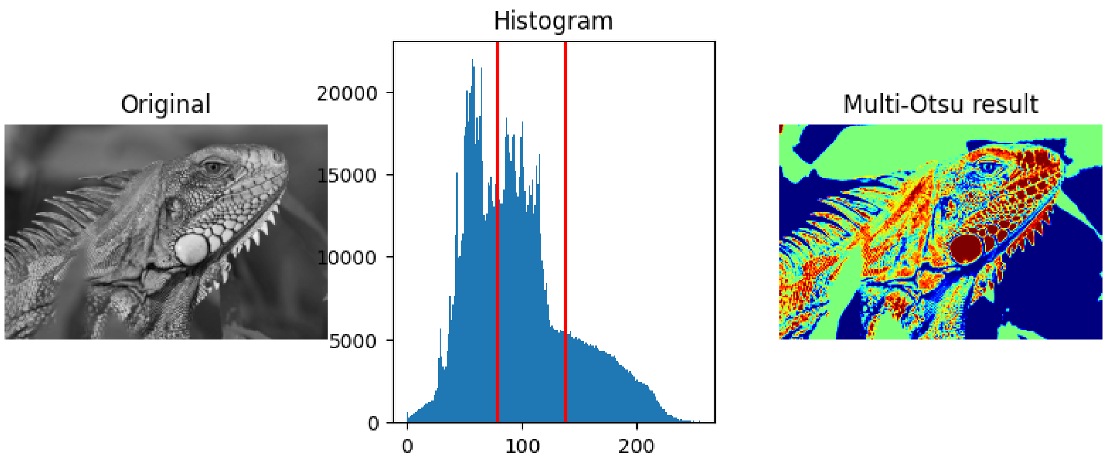


Original
Histogram
Multi-Otsu result
160000
140000
120000
100000
80000
60000
40000
20000
0
50
100
150
200
250



Original
Histogram
Multi-Otsu result
0
100
200





7 2. (Opcional)

Como en un proyecto previo, es posible aplicar Otsu para ventanas de diferentes tamaños, obteniendo mejores resultados. Realiza esta implementacion y ve como mejoran los resultados con el ejemplo de la hoja de papel.

Función para Otsu general

[15]:
```
def otsu_proc(image):
    thresh = threshold_otsu(image)
    binary = image > thresh
    return binary
```

Imagen sobre la cuál vamos a trabajar

[17]:
```python
imagen =  data.page()
```

Dividimos la imagen en 4 ventanas

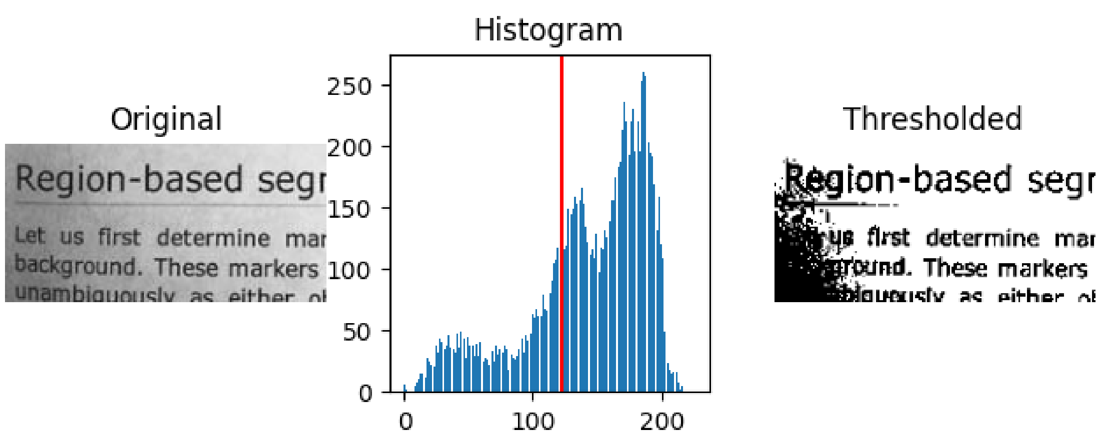
[18]:
```python
(h, w) = imagen.shape[:2]
# compute the center coordinate of the image
(cX, cY) = (w // 2, h // 2)
```

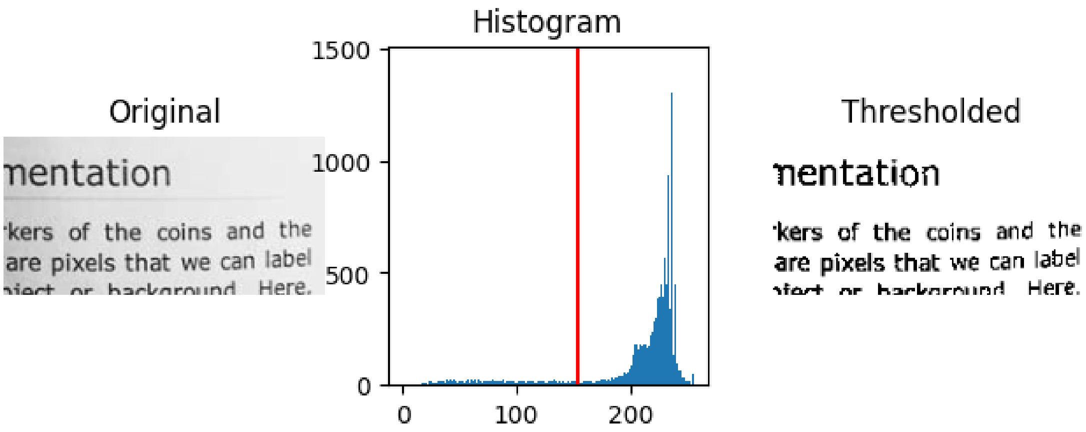
[19]:
```python
topLeft = imagen[0:cY, 0:cX]
topRight = imagen[0:cY, cX:w]
bottomLeft = imagen[cY:h, 0:cX]
bottomRight = imagen[cY:h, cX:w]
```

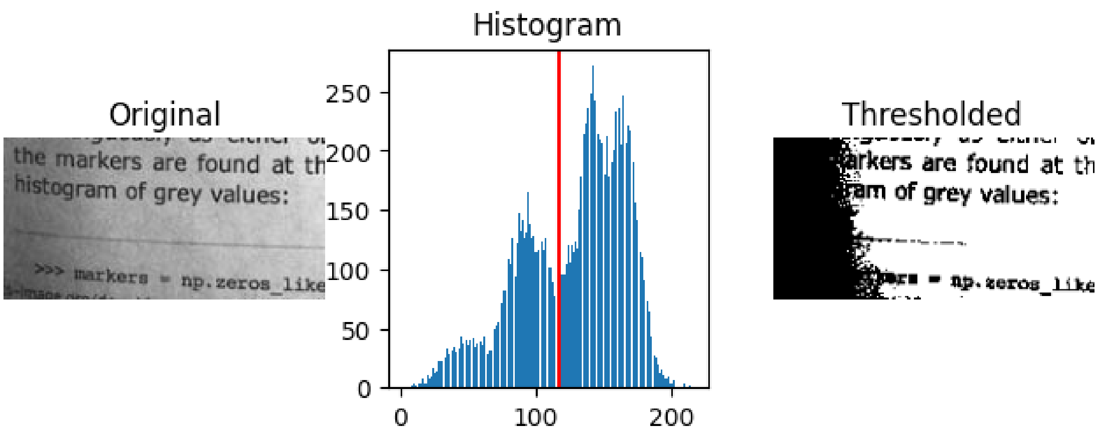
[20]:
```python
images_w = []

images_w = [topLeft, topRight, bottomLeft, bottomRight]
```

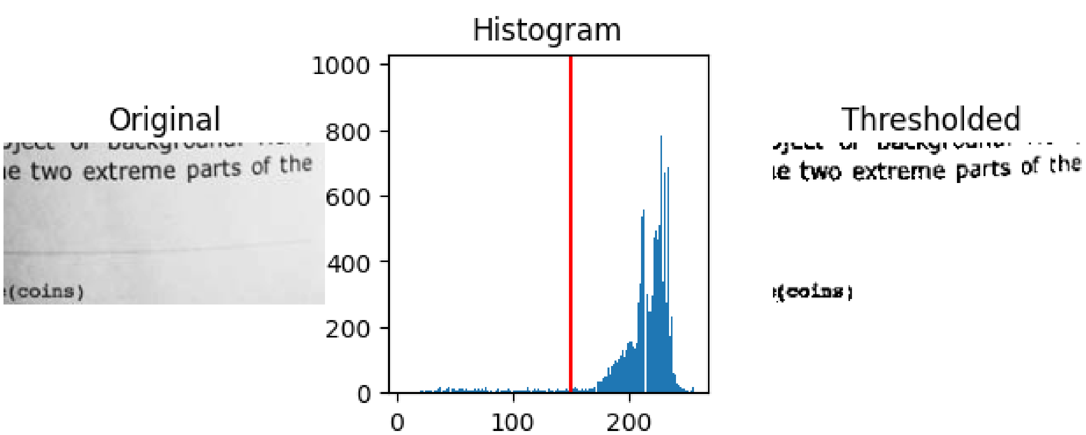
[21]:
```python
for image in images_w:
  single_tres_otsu(image)
```

```
<ipython-input-8-1577835ef629>:9: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases
later; explicitly call ax.remove() as needed.
  ax[2] = plt.subplot(1, 3, 3, sharex=ax[0], sharey=ax[0])
```




Original
Histogram
Thresholded
1500
1000
500
0
0
100
200


Original
Histogram
Thresholded
250
200
150
100
50
0
0
100
200


Original
Histogram
Thresholded
1000
800
600
400
200
0
0
100
200

[22]:
```python
otsu_img = []
for image in images_w:
  otsu_img.append(otsu_proc(image))
```

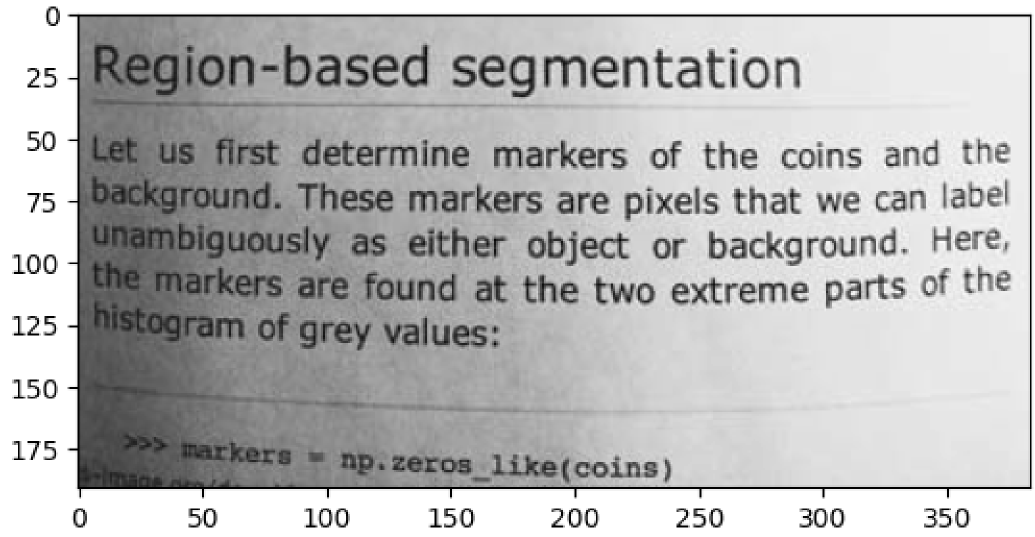
[24]:
```python
from PIL import Image
image_tl = Image.fromarray(otsu_img[0])
image_tr = Image.fromarray(otsu_img[1])
image_bl = Image.fromarray(otsu_img[2])
image_br = Image.fromarray(otsu_img[3])
```

[25]:
```python
#resize, first image
image1 = image_tl
image2 = image_tr
image3 = image_bl
image4 = image_br


image1_size = image_tl.size
image2_size = image_tr.size
image3_size = image_bl.size
image4_size = image_br.size

new_image = Image.new('RGB',(2*image1_size[0], 2*image1_size[1]))
new_image.paste(image1,(0,0))
new_image.paste(image2,(image1_size[0],0))
new_image.paste(image3,(0,image1_size[1]))
new_image.paste(image4,(image1_size[0],image2_size[1]))

new_image.save("/content/merged_image.jpg","JPEG")
```

Imagen Original

[26]:
```python
plt.imshow(imagen, cmap='gray')
```

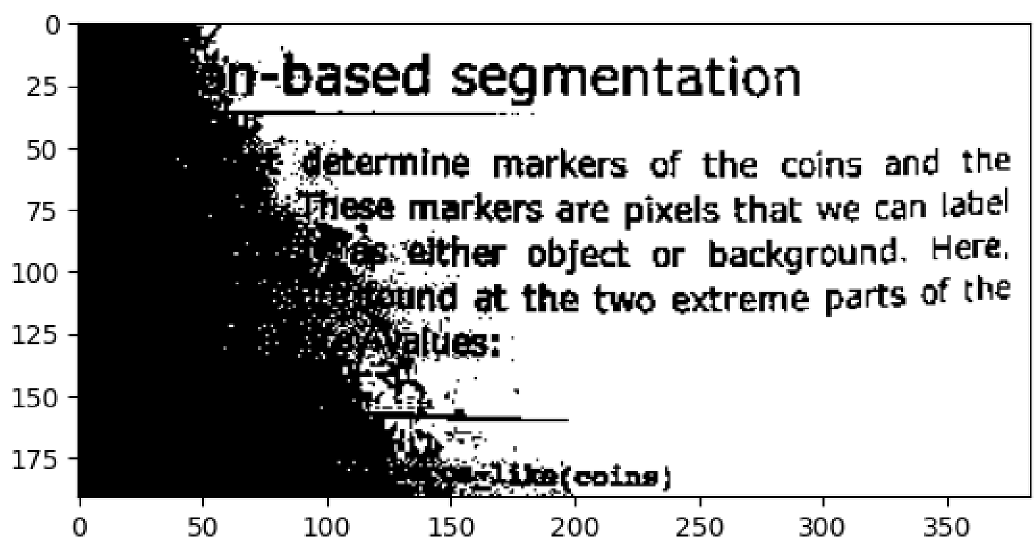
[26]:
```
<matplotlib.image.AxesImage at 0x7ee97cf7eb30>
```



Segmentación OTSU general

```
[27]: plt.imshow(otsu_proc(imagen), cmap='gray')
```

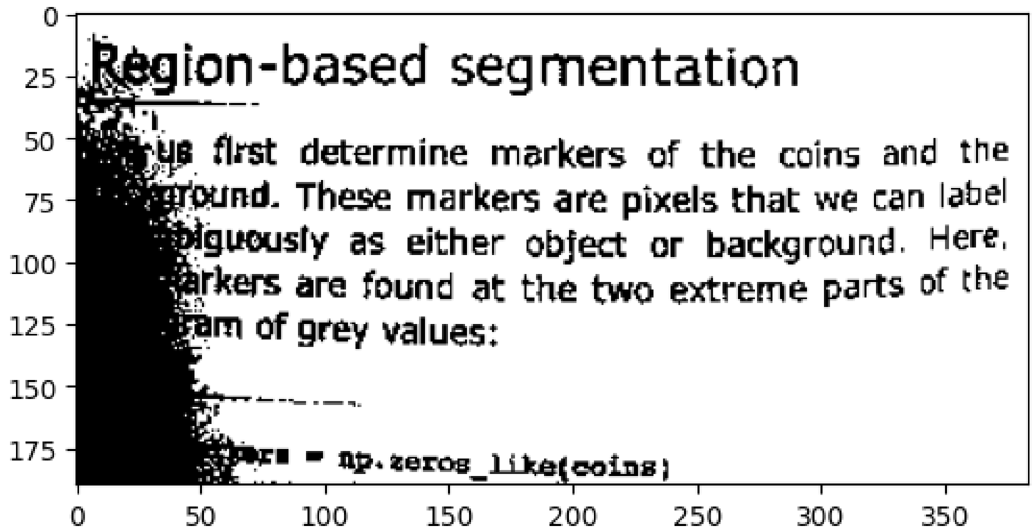
```
[27]: <matplotlib.image.AxesImage at 0x7ee97a446c20>
```



Segmentación OTSU por ventanas (4)

```
[28]: plt.imshow(new_image)
```

```
[28]: <matplotlib.image.AxesImage at 0x7ee97cf1cf10>
```



8 Conclusiones

El método de otsu es una técnica de thresholding automática que se utiliza para determinar uno o varios umbrales optimo al separar los pixeles de una imagen en clases distintas. Este método es bueno en su eficiencia al hacer el cálculo, y ayuda al no tener que establecerlo anualmente. Además de que tiene buena adaptabilidad para la distribución de intensidades en iluminación y contraste. Generalmente es utilizado en aplicaciones de segmentación de imágenes, donde se requiere separar objetos de interés de un fondo.

Una nota particular con la primera imagen y con 1 solo umbral, fue que el ruido que tenia la imagen no lo permitía funcionar bien. Por lo se soluciono aplicando un median filter. Seguido, fue posible convertir la imagen a una imagen binaria donde la identificación del texto fuera mucho mas sencilla.

Al observar la diferencia entre las imágenes con 1 solo umbral, y con 3 clases, fue posible como variaba la distribución de intensidades entre las imágenes. Sin embargo, es claro notar que, dependiendo de la imagen, estas pueden requerir cierto pre-procesamiento de imagen para adaptarlas al método. Ya que es sensible al ruido.

Adicionalmente, al realizar la segmentación de Otsu por ventana, podemos observar que se puede obtener una mejor definición del threshold óptimo en esa ventana, permitiendo obtener imágenes mejor segmentadas.

9 Referencias

- baeldung. (2023). Understanding Otsu's Method for Image Segmentation. Retrieved from Baeldung: https://www.baeldung.com/cs/otsu-segmentation

- Geeks For Geeks. (2021). Mahotas – Otsu's method. Retrieved from Geeks For Geeks: https://www.geeksforgeeks.org/mahotas-otsus-method/
- Gonzalez, R., & Woods, R. (2018). Digital Image Processing. Pearson.
- Murzova, A. (2020). Otsu's Thresholding with OpenCV. Retrieved from LearnOpenCV: https://learnopencv.com/otsu-thresholding-with-opencv/
- Muthukrishnan. (2020). Otsu's method for image thresholding explained and implemented. Retrieved from MUTHU: https://muthu.co/otsus-method-for-image-thresholding-explained-and-implemented/
- OpenCV. (n.d.). Image Thresholding. Retrieved from OpenCV: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

[ ]:

```

```