```
[1]  from google.colab import drive
     drive.mount ('/content/drive')
```

```
Mounted at /content/drive
```

```
[2]  import numpy as np
     import cv2
     from matplotlib import pyplot as plt
     from skimage.color import rgb2gray
     from skimage.filters import threshold_otsu
     from skimage.measure import label, regionprops
     from skimage.segmentation import mark_boundaries
     from scipy import ndimage as ndi
     import pandas as pd
     import json
     import os
     import timeit
     import random
```

```
[3]  def ShowImage(ImageList, nRows = 1, nCols = 2, WidthSpace = 0.00, HeightSpace = 0.00):
         from matplotlib import pyplot as plt
         import matplotlib.gridspec as gridspec

         gs = gridspec.GridSpec(nRows, nCols)
         gs.update(wspace=WidthSpace, hspace=HeightSpace)
         plt.figure(figsize=(20,20))
         for i in range(len(ImageList)):
             ax1 = plt.subplot(gs[i])
             ax1.set_xticklabels([])
             ax1.set_yticklabels([])
             ax1.set_aspect('equal')

             plt.subplot(nRows, nCols,i+1)

             image = ImageList[i].copy()
             if (len(image.shape) < 3):
                 plt.imshow(image, plt.cm.gray)
             else:
                 plt.imshow(image)
             plt.title("Image " + str(i))
             plt.axis('off')

         plt.show()
```

```
[4]  import os
     import pandas as pd

     def get_subfiles(dir):
         "Get a list of immediate subfiles"
         return next(os.walk(dir))[2]
```

```
[5]  def ResizeImage(IM, DesiredWidth, DesiredHeight):
         from skimage.transform import rescale, resize

         OrigWidth = float(IM.shape[1])
         OrigHeight = float(IM.shape[0])
         Width = DesiredWidth
         Height = DesiredHeight

         if((Width == 0) & (Height == 0)):
             return IM

         if(Width == 0):
             Width = int((OrigWidth * Height)/OrigHeight)

         if(Height == 0):
             Height = int((OrigHeight * Width)/OrigWidth)

         dim = (Width, Height)
         resizedIM = cv2.resize(IM, dim, interpolation = cv2.INTER_NEAREST)
         return resizedIM
```

```
[6]  import os
     path_Data = "//content//drive//MyDrive//BIẾN HÌNH//Object Segmentation Data//"
     checkPath = os.path.isdir(path_Data)
     print("The path and file are valid or not :", checkPath)
```

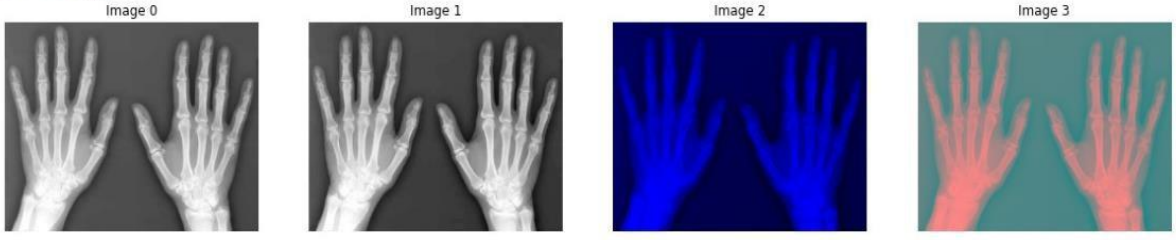
```
The path and file are valid or not : True
```

```
[7]  all_names = get_subfiles(path_Data)
     print("Number of Images:", len(all_names))
     IMG = []
     for i in range(len(all_names)):
         tmp = cv2.imread(path_Data + all_names[i])
         IMG.append(tmp)

     ImageDB = IMG.copy()
     NameDB = all_names
```

```
Number of Images: 28
```

```
[8]  FileName = 'Hand.jpg'
     idx = NameDB.index(FileName)
     print("Selected Image : ", "\nIndex ", idx, "\nName ", NameDB[idx])

     image_orig = ImageDB[idx]
     image_gray = cv2.cvtColor(image_orig,cv2.COLOR_BGR2GRAY)
     image_hsv = cv2.cvtColor(image_orig, cv2.COLOR_BGR2HSV)
     image_ycbcr = cv2.cvtColor(image_orig, cv2.COLOR_BGR2YCR_CB)
     ShowImage([image_orig, image_gray, image_hsv, image_ycbcr], 1, 4)
```

```
Selected Image :
Index  12
Name  Hand.jpg
```

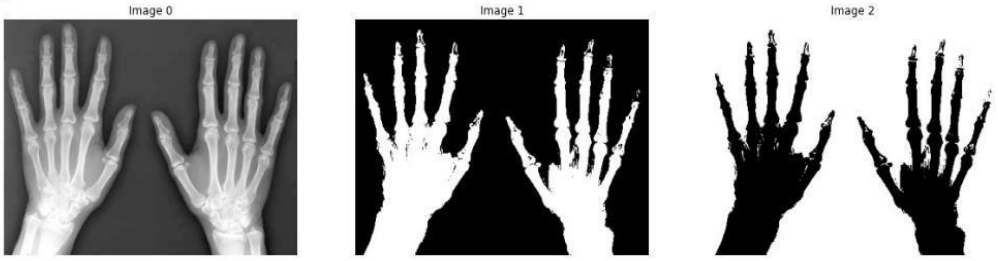


```python
[9]  def p_tile_threshold(image, pct):
       """Runs the p-tile threshold algorithm.
       Reference:
       Parker, J. R. (2010). Algorithms for image processing and
       computer vision. John Wiley & Sons.
       @param image: The input image
       @type image: ndarray
       @param pct: The percent of desired background pixels (black pixels).
           It must lie in the interval [0, 1]
       @type pct: float
       @return: The p-tile global threshold
       @rtype int
       """
       n_pixels = pct * image.shape[0] * image.shape[1]
       hist = np.histogram(image, bins=range(256))[0]
       hist = np.cumsum(hist)

       return np.argmin(np.abs(hist - n_pixels))
```

```python
[10] def otsu(gray):
       pixel_number = gray.shape[0] * gray.shape[1]
       mean_weigth = 1.0/pixel_number
       his, bins = np.histogram(gray, np.array(range(0, 256)))
       final_thresh = -1
       final_value = -1

       WBackground = []
       WForeground = []
       Values = []

       for t in bins[1:-1]:
           Wb = np.sum(his[:t]) * mean_weigth
           Wf = np.sum(his[t:]) * mean_weigth

           mub = np.mean(his[:t])
           muf = np.mean(his[t:])

           value = Wb * Wf * (mub - muf) ** 2
           WBackground.append(Wb)
           WForeground.append(Wf)
           Values.append(value)

           if value > final_value:
               final_thresh = t
               final_value = value

       final_img = gray.copy()
       print(final_thresh)
       final_img[gray > final_thresh] = 255
       final_img[gray < final_thresh] = 0
       return final_img, final_thresh, [WBackground, WForeground, Values]
```

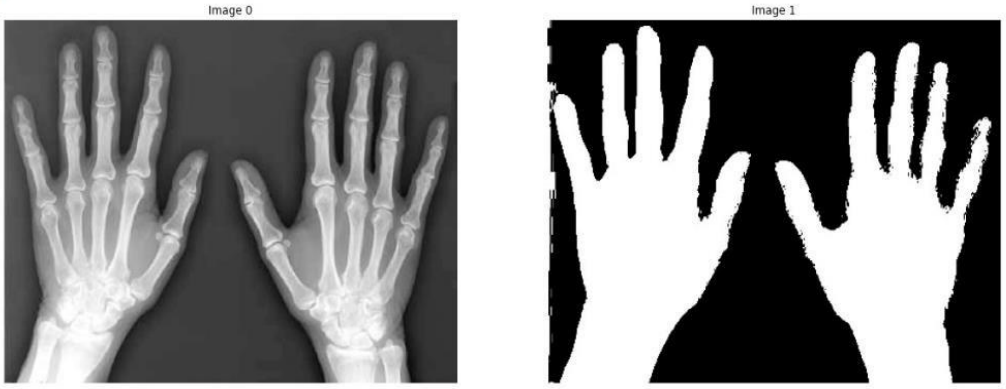
```
[11] T = p_tile_threshold(image_gray, pct = 0.7)
     print(T)

     class1 = image_gray > T
     class2 = image_gray <= T
     ShowImage([image_gray, class1, class2], 1, 3)
```

```
151
```



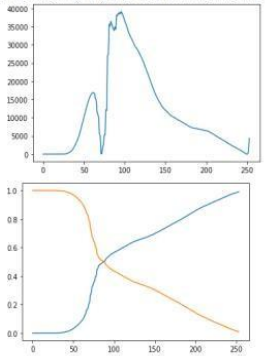

```
[12] final_img, final_thresh, parms = otsu(image_gray)
     ShowImage([image_gray, final_img], 1, 2)
```

```
97
```




```
[13] WBackground, WForeground, Values = parms[0], parms[1], parms[2]
     print(WBackground)
     print(WForeground)
     print(Values)

     plt.plot(Values)
     plt.show()
     plt.plot(WBackground)
     plt.plot(WForeground)
     plt.show()
```

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 7.177720356014929 4e-06, 1.4355440712029859e-05, 2.153316106804478 8e-05, 2.8710881424059718e-05, 5.0244042492104 51e-05, 0.331036
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9999928222796439, 0.9999856445592888, 0.9999784668389319, 0.9999712891185759, 0.9999497559575079, 0.9999066896353718, 0.9998348
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.5658203192640254, 5.175150726057292, 7.828934540128070 6, 10.528122063575447, 18.577642135962126, 34.77609422341671, 61.988481
```



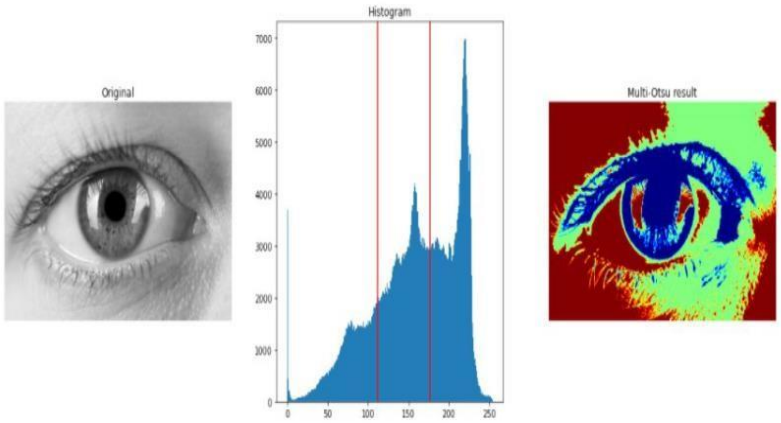
1 giây hoàn thành lúc 16:31

```
[14] def min_err_threshold(image):
        """Runs the minimum error thresholding algorithm.
        Reference:
        Kittler, J. and J. Illingworth. ‘‘On Threshold Selection Using Clustering
        Criteria,’’ IEEE Transactions on Systems, Man, and Cybernetics 15, no. 5
        (1985): 652-655.
        @param image: The input image
        @type image: ndarray
        @return: The threshold that minimize the error
        @rtype: int
        """
        hist = np.histogram(image, bins=range(256))[0].astype(np.float)

        w_backg = hist.cumsum()
        w_backg[w_backg == 0] = 1

        w_foreg = w_backg[-1] - w_backg
        w_foreg[w_foreg == 0] = 1

        cdf = np.cumsum(hist * np.arange(len(hist)))

        b_mean = cdf / w_backg
        f_mean = (cdf[-1] - cdf) / w_foreg

        b_std = ((np.arange(len(hist)) - b_mean)**2 * hist).cumsum() / w_backg
        f_std = ((np.arange(len(hist)) - f_mean) ** 2 * hist).cumsum()
        f_std = (f_std[-1] - f_std) / w_foreg

        b_std[b_std == 0] = 1
        f_std[f_std == 0] = 1

        error_a = w_backg * np.log(b_std) + w_foreg * np.log(f_std)
        error_b = w_backg * np.log(w_backg) + w_foreg * np.log(w_foreg)
        error = 1 + 2 * error_a - 2 * error_b

        final_img = image.copy()
        final_thresh = np.argmin(error)
        print(final_thresh)
        final_img[image > final_thresh] = 255
        final_img[image < final_thresh] = 0

        return final_img, final_thresh
        final_img, final_thresh = min_err_threshold(image_gray)
        ShowImage([image_gray, final_img], 1, 2)
```

+ Mã + Văn bản

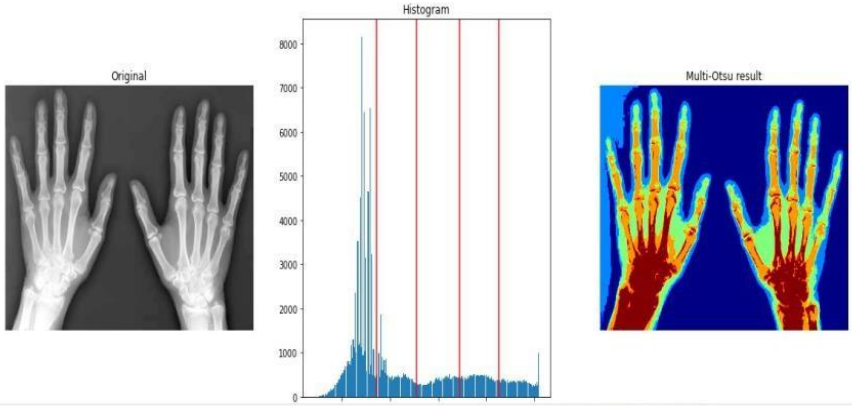
```
def two_peaks_threshold(image, smooth_hist=True, sigma=5):
    from scipy.ndimage import gaussian_filter
    """Runs the two peaks threshold algorithm. It selects two peaks
    from the histogram and return the index of the minimum value
    between them.
    The first peak is deemed to be the maximum value fo the histogram,
    while the algorithm will look for the second peak by multiplying the
    histogram values by the square of the distance from the first peak.
    This gives preference to peaks that are not close to the maximum.
    Reference:
    Parker, J. R. (2010). Algorithms for image processing and
    computer vision. John Wiley & Sons.
    @param image: The input image
    @type image: ndarray
    @param smooth_hist: Indicates whether to smooth the input image
        histogram before finding peaks.
    @type smooth_hist: bool
    @param sigma: The sigma value for the gaussian function used to
        smooth the histogram.
    @type sigma: int
    @return: The threshold between the two founded peaks with the
        minimum histogram value
    @rtype: int
    """
    hist = np.histogram(image, bins=range(256))[0].astype(np.float)
    plt.plot(hist)
    plt.show()

    if smooth_hist:
        hist = gaussian_filter(hist, sigma=sigma)
        plt.plot(hist)
        plt.show()

    f_peak = np.argmax(hist)

    s_peak = np.argmax((np.arange(len(hist)) - f_peak) ** 2 * hist)

    thr = np.argmin(hist[min(f_peak, s_peak): max(f_peak, s_peak)])
    thr += min(f_peak, s_peak)

    final_img = image.copy()
    print(thr)
    final_img[image > thr] = 255
    final_img[image < thr] = 0

    return final_img, thr, hist
    final_img, final_thresh, hist = two_peaks_threshold(image_gray)
    ShowImage([image_gray, final_img], 1, 2)
```

```
[22] from skimage.filters import threshold_multiotsu
     thresholds = threshold_multiotsu(image_gray)
     regions = np.digitize(image_gray, bins=thresholds)

     fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(20, 7))
     ax[0].imshow(image_gray, cmap='gray')
     ax[0].set_title('Original')
     ax[0].axis('off')
     ax[1].hist(image_gray.ravel(), bins=255)
     ax[1].set_title('Histogram')
     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()
```




```
[17] Segments = []

     for idx in list(np.unique(regions)):
       mask = regions == idx
       Segments.append(mask)

     ShowImage(Segments, 1, len(Segments))
```

```
thresholds = threshold_multiotsu(image_gray, classes=5)
regions = np.digitize(image_gray, bins=thresholds)

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(20, 7))
ax[0].imshow(image_gray, cmap='gray')
ax[0].set_title('Original')
ax[0].axis('off')

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



✓ 1 giây hoàn thành lúc 16:31

```
FileName = 'Iris.jpg'
idx = NameDB.index(FileName)
print("Selected Image : ", "\nIndex ", idx, "\nName ", NameDB[idx])

image_orig = ImageDB[idx]
image_gray = cv2.cvtColor(image_orig,cv2.COLOR_BGR2GRAY)
image_hsv = cv2.cvtColor(image_orig, cv2.COLOR_BGR2HSV)
image_ycbcr = cv2.cvtColor(image_orig, cv2.COLOR_BGR2YCR_CB)
ShowImage([image_orig, image_gray, image_hsv, image_ycbcr], 1, 4)
```

```
Selected Image :
Index  1
Name  Iris.jpg
```