Họ và Tên: Huỳnh Nguyễn Thế Dân

MSSV: 21110256

Lớp: 21TTH1

In [ ]:
```
import os

path = os.getcwd()
os.chdir(path)
print(os.listdir())
```

```
['21110256_HuynhNguyenTheDan_ IS Lab01.ipynb', 'Lab01_IS_Thresholding_based_Segmentation_NEW.html', 'Object Segmentation Data']
```

In [ ]:
```
os.chdir(os.path.join(path, 'Object Segmentation Data'))
print(os.listdir())
```

```
['Activities.jpeg', 'Barcode.png', 'Bone.jpg', 'Brain.jpg', 'Car.jpg', 'Chest.jpg', 'Cloths.jpg', 'Code.jpg', 'Crack.jpg', 'Cross.jpg', 'Defect.jpg', 'Dust.jpg', 'Emotion.jpg', 'Face.jpg', 'Fire.jpg', 'Gesture.jpg', 'Hand.jpg', 'Iris.jpg', 'Leaf.jpg', 'Lung.png', 'Mask.jpg', 'Melanoma.jpg', 'QR.jpg', 'Retina.jpg', 'Shelf.jpg', 'Sign.jpg', 'Tumor.png', 'Writing.png']
```

In [ ]:
```
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
```

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

    gs = gridspec.GridSpec(nRows, nCols)
    gs.update(wspace=WidthSpace, hspace=HeightSpace) # set the spacing between a
    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()
```

In [ ]:
```
import os
import pandas as pd

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

In [ ]:
```
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
```

In [ ]:
```
path_Data = os.path.join(path, 'Object Segmentation Data')
os.listdir(path_Data)
```

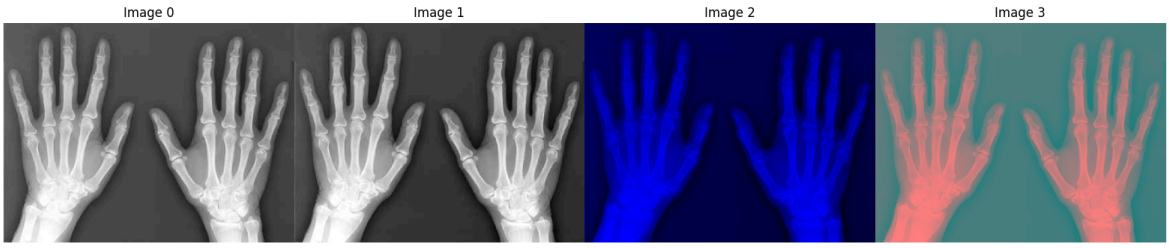
Out[ ]:
```
['Activities.jpeg',
 'Barcode.png',
 'Bone.jpg',
 'Brain.jpg',
 'Car.jpg',
 'Chest.jpg',
 'Cloths.jpg',
 'Code.jpg',
 'Crack.jpg',
 'Cross.jpg',
 'Defect.jpg',
 'Dust.jpg',
 'Emotion.jpg',
 'Face.jpg',
 'Fire.jpg',
 'Gesture.jpg',
 'Hand.jpg',
 'Iris.jpg',
 'Leaf.jpg',
 'Lung.png',
 'Mask.jpg',
 'Melanoma.jpg',
 'QR.jpg',
 'Retina.jpg',
 'Shelf.jpg',
 'Sign.jpg',
 'Tumor.png',
 'Writing.png']
```

In [ ]:
```
path_Data = os.path.join(path, 'Object Segmentation Data')
all_names = sorted(get_subfiles(path_Data))
print("Number of Images:", len(all_names))
IMG = []
for i in range(len(all_names)):
  tmp = cv2.imread(os.path.join(path_Data,all_names[i]))
  IMG.append(tmp)

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

```
Number of Images: 28
['Activities.jpeg', 'Barcode.png', 'Bone.jpg', 'Brain.jpg', 'Car.jpg', 'Chest.jp
g', 'Cloths.jpg', 'Code.jpg', 'Crack.jpg', 'Cross.jpg', 'Defect.jpg', 'Dust.jpg',
'Emotion.jpg', 'Face.jpg', 'Fire.jpg', 'Gesture.jpg', 'Hand.jpg', 'Iris.jpg', 'Le
af.jpg', 'Lung.png', 'Mask.jpg', 'Melanoma.jpg', 'QR.jpg', 'Retina.jpg', 'Shelf.j
pg', 'Sign.jpg', 'Tumor.png', 'Writing.png']
```

In [ ]:
```
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)
```

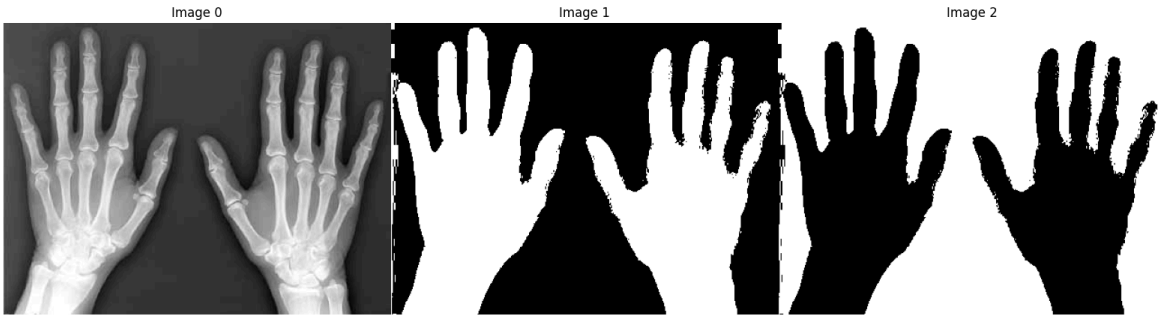
```
Selected Image :
Index  16
Name  Hand.jpg
```



In [ ]:
```python
def p_tile_threshold(image, pct):
  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))
```

In [ ]:
```python
T = p_tile_threshold(image_gray, pct = 0.55)
print(T)

mask_obj = image_gray > T
mask_bg = image_gray <= T
ShowImage([image_gray, mask_obj, mask_bg], 1, 3)
```

```
96
```

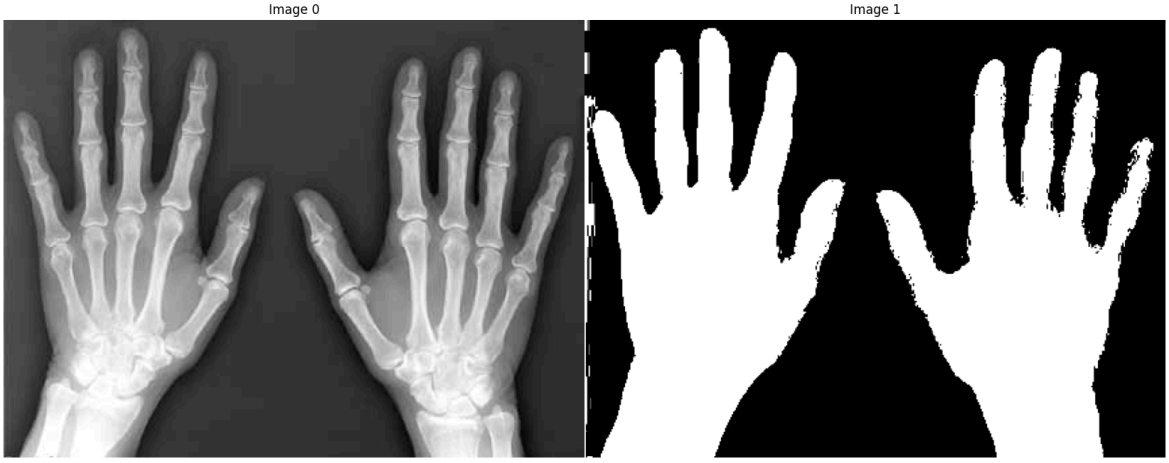


In [ ]:
```python
def otsu(gray):
  pixel_number = gray.shape[0] * gray.shape[1]
  mean_weigth = 1.0/pixel_number
  his, bins = np.histogram(gray, np.array(range(0, 256)))
  final_thresh = -1
  final_value = -1

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

  for t in bins[1:-1]: # This goes from 1 to 254 uint8 range (Pretty sure wont b
      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
      # print("Wb", Wb, "Wf", Wf)
      # print("t", t, "value", value)
      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]
```

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

97

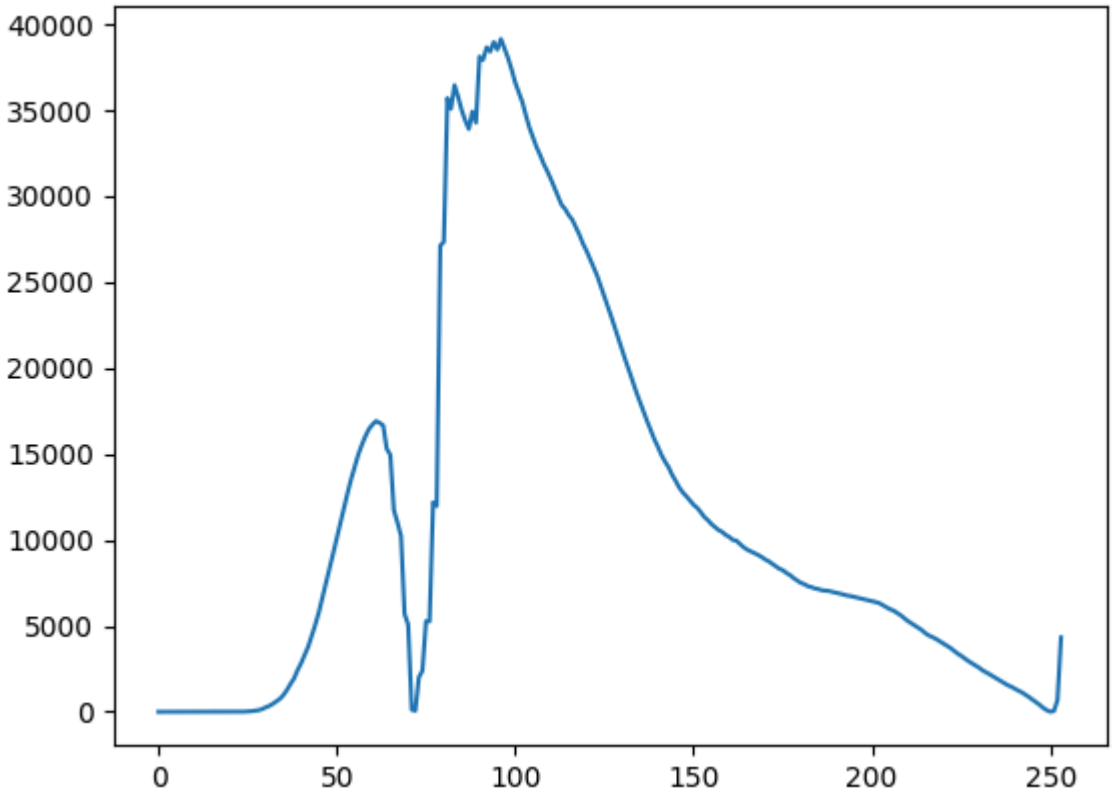


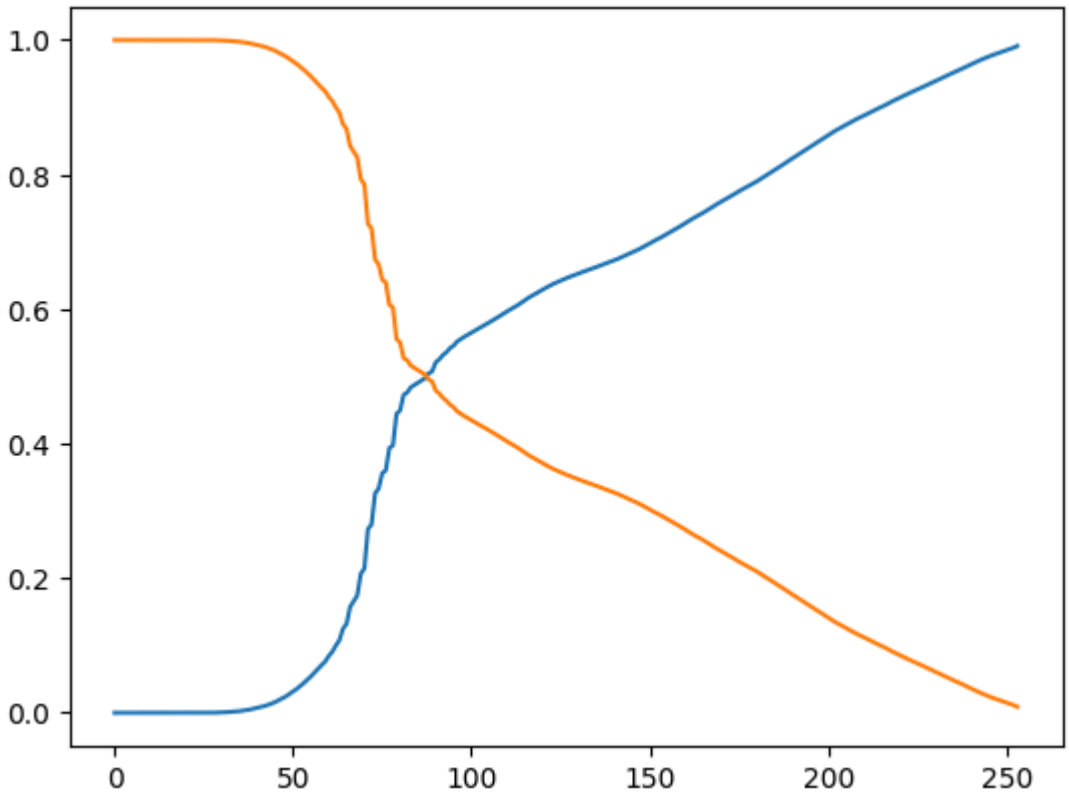
In [ ]:
```
WBackground, WForeground, Values = parms[0], parms[1], parms[2]
print(WBackground)
print(WForeground)
print(Values)

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, 7.177203560149294e-06, 1.4355440712029859e-05, 2.1533161068044788e-05, 2.8710881424059718e-05, 5.024404249210451e-05, 9.331036462819408e-05, 0.00016508756818834337, 0.0002440424921045076, 0.00039477461958082114, 0.0006603502727533735, 0.0008756818834338214, 0.0011699684180304336, 0.0015647430376112546, 0.0019164513350559862, 0.0024834912431811657, 0.003244329600918748, 0.004163077806488659, 0.004995693367786391, 0.006302038472581108, 0.007378696525983348, 0.008678639104220 5, 0.010041630778064886, 0.011871949468848694, 0.013781223083548665, 0.016006316393913294, 0.018726672408842952, 0.021583405110536892, 0.024382716049382715, 0.02794286534596612, 0.03147430376112546, 0.0353861613551536, 0.039735859890 89865, 0.0441358024691358, 0.04908125179443009, 0.05404823428079242, 0.059788323284524835, 0.06510192362905541, 0.0709158771174275, 0.0760623026126902, 0.0843812805053115, 0.09063307493540051, 0.09997846683893195, 0.10795291415446454, 0.1249066896353718, 0.13213465403387883, 0.1573643410852713, 0.1656761412575366, 0.17433247200689062, 0.20666810221073786, 0.21463537180591444, 0.2731122595463681, 0.27989520528280215, 0.3261699684180304, 0.33354148722365773, 0.35608670686190064, 0.36021389606660925, 0.3936548952052828, 0.39729399942578236, 0.44427217915590006, 0.4494401378122308, 0.472616996841803, 0.4761125466551823, 0.4838070628768303, 0.4871805914441573, 0.4902813666379558, 0.4937194946884869, 0.4973585989089865, 0.5043640539764571, 0.5076370944587999, 0.5209517657192075, 0.5253301751363767, 0.5316968130921619, 0.5359532012632787, 0.5419609532012633, 0.5457579672695951, 0.551923629055412, 0.5554622451909274, 0.5589003732414585, 0.562051392477749, 0.5649727246626471, 0.5682457651449899, 0.5715690496698248, 0.5742894056847545, 0.5771102497846684, 0.5802469135802469, 0.5833835773758255, 0.5867571059431524, 0.589951191501579, 0.5933893195521103, 0.596662360034453, 0.5997990238300316, 0.6028495549813379, 0.6059933964972725, 0.6097832328452484, 0.6132357163364915, 0.6169250645994832, 0.6200832615561298, 0.6232055699109963, 0.6260335917312662, 0.6291630778064886, 0.6320772322710307, 0.6348908986505886, 0.6376830318690784, 0.6400157909847832, 0.6424346827447602, 0.6446597760551249, 0.6469207579672696, 0.6489592305483778, 0.6509043927648579, 0.6530146425495262, 0.654859316681022, 0.6568834338214183, 0.6586922193511341, 0.66057278208441, 0.6625394774619581, 0.6645133505598622, 0.6664298018949182, 0.6683390755096181, 0.6703488372093023, 0.6725954636807735, 0.6746339362618432, 0.6770384725811082, 0.6796583405110537, 0.6817183462532299, 0.684151593453919, 0.6864197530864198, 0.6891257536606373, 0.6920829744473155, 0.6948679299454493, 0.6977246626471433, 0.7008828596037898, 0.7036319265001435, 0.7062087281079529, 0.7092377260981912, 0.7120513924777491, 0.7150732127476314, 0.7181596325007178, 0.7215403387884007, 0.7245549813379271, 0.7278782658627619, 0.7308641975308642, 0.7345320126327878, 0.7373887453344817, 0.7403172552397358, 0.7434323858742463, 0.7467915590008612, 0.7502081538903244, 0.7534094171691 07, 0.7566393913293138, 0.7596612115991961, 0.7629127189204709, 0.7659273614699971, 0.7689563594602354, 0.7719422911283376, 0.7752799310938846, 0.7781294860752225, 0.781244616709733, 0.7840223944875108, 0.786993970714901, 0.789944013781223, 0.7932888314671 26, 0.7963465403387884, 0.7997990238300315, 0.8031151306345105, 0.8066393913293138, 0.8100057421762847, 0.8136807349985644, 0.817312661498708, 0.8207220786678151, 0.8242248062015504, 0.8276629342520815, 0.8310795291415446, 0.8344530577088717, 0.8380203847258111, 0.8414585127763422, 0.8449540625897215, 0.8483275911570485, 0.8518159632500718, 0.8551607809359747, 0.858670686190066, 0.8619293712316968, 0.8652526557565317, 0.8682026988228538, 0.8711814527706, 0.8739879414298019, 0.8771317829457365, 0.8798736721217341, 0.8826299167384438, 0.8850703416594 89, 0.8876040769451622, 0.8903100775193798, 0.8929801894918173, 0.8957651449899512, 0.8982988802756244, 0.9006029285099052, 0.9032371518805627, 0.9061800172265289, 0.9089936836060867, 0.9116063738156761, 0.9143267298306058, 0.9167240884295148, 0.9194731553258685, 0.9217341372380132, 0.92408125179443, 0.9266508756818834, 0.9289692793568762, 0.9313379270743611, 0.9338644846396784, 0.9364843525696238, 0.9388817111685328, 0.9411355153603216, 0.943755383290267, 0.9462604076945161, 0.9487582543784094, 0.9513494114269307, 0.9537037037037037, 0.956294860752225, 0.9585414872236577, 0.9612761986792995, 0.9637309790410565, 0.9662431811656618, 0.9688415159345392, 0.9711024978466839, 0.9733132357163364, 0.9753875968992248, 0.9775193798449612, 0.9795004306632213, 0.9811584840654608, 0.9830318690783807, 0.9848191214470284, 0.9869580821131209, 0.9886592018374963, 0.9910780935974735]

[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, 0.999985644559288, 0.999978466838931
9, 0.9999712891185759, 0.9999497559575079, 0.9999066896353718, 0.999834912431811
7, 0.9997559575078955, 0.9996052253804192, 0.9993396497272466, 0.999124318116566
1, 0.9988300315819695, 0.9984352569623887, 0.998083548664944, 0.9975165087568189,
0.9967556703990812, 0.9958369221935113, 0.9950043066322136, 0.9936979615274188,
0.9926213034740167, 0.9913221360895779, 0.989958369221935, 0.9881280505311513, 0.
9862187769164513, 0.9839936836060866, 0.981273327591157, 0.9784165948894631, 0.97
56172839506172, 0.9720571346540339, 0.9685256962388745, 0.9646138386448464, 0.960
2641401091013, 0.9558641975308642, 0.9509187482055699, 0.9459517657192076, 0.9402
167671547517, 0.9348980763709446, 0.9290841228825725, 0.9239376973873098, 0.91561
87194946884, 0.9093669250645995, 0.900021533161068, 0.8920470858455354, 0.8750933
103646281, 0.8678653459661211, 0.8426356589147287, 0.8343238587424634, 0.82566752
79931094, 0.793331897789262, 0.7853646281940855, 0.7268877404536319, 0.7201047947
171978, 0.6738300315819695, 0.6664585127763422, 0.6439132931380993, 0.63978610393
33907, 0.6063451047947171, 0.6027060005742176, 0.5557278208440999, 0.550559862187
7692, 0.527383003158197, 0.5238874533448177, 0.5161929371231697, 0.51281940855584
26, 0.5097186333620441, 0.506280505311513, 0.5026414010910135, 0.495635946023542
9, 0.4923629055412001, 0.4790482342807924, 0.4746698248636233, 0.4683031869078380
6, 0.4640467987367212, 0.4580390467987367, 0.4542420327304048, 0.4480763709445879
7, 0.44453775480907265, 0.4410996267585415, 0.4379486075222509, 0.435027275337352
86, 0.43175423485501, 0.42843095033017514, 0.42571059431524544, 0.422889750215331
6, 0.41975308641975306, 0.41661642262417453, 0.41324289405684755, 0.4100488084984
209, 0.4066106804478897, 0.40333763996554695, 0.4002009761699684, 0.3971504450186
6203, 0.39400660350272754, 0.3902167671547516, 0.38676428366350846, 0.38307493540
05168, 0.3799167384438702, 0.3767944300890037, 0.37396640826873384, 0.37083692219
351133, 0.36792276772896926, 0.36510910134941144, 0.3623169681309216, 0.359984209
01521677, 0.3575653172552397, 0.3553402239448751, 0.3530792420327304, 0.351040769
45162216, 0.3490956072351421, 0.3469853574504737, 0.3451406833189779, 0.343116566
17858166, 0.3413077806488659, 0.33942721791559, 0.33746052253804193, 0.3354866494
401378, 0.3335701981050818, 0.33166092449038187, 0.32965116279069767, 0.327404536
31926497, 0.32536606373815674, 0.3229615274188917, 0.3203416594889463, 0.31828165
374677003, 0.31584840654608093, 0.3135802469135802, 0.3108742463393626, 0.3079170
2555268446, 0.3051320700545507, 0.3022753373528567, 0.2991171403962102, 0.2963680
7349985644, 0.29379127189204707, 0.29076227390180875, 0.28794860752225093, 0.2849
2678725236864, 0.2818403674992822, 0.2784596612115992, 0.27544501866207294, 0.272
121734137238, 0.2691358024691358, 0.26546798736721217, 0.2626112546655182, 0.2596
8274476026415, 0.25656761412575363, 0.25320844099913864, 0.24979184610967556, 0.2
465905828308929, 0.24336060867068618, 0.2403387884008039, 0.23708728107952914, 0.
23407263853000287, 0.23104364053976456, 0.22805770887166235, 0.2247200689061154,
0.2218705139247775, 0.218755383290267, 0.21597760551248923, 0.21300602928509904,
0.2100559862187769, 0.20671116853287394, 0.2036534596612116, 0.2002009761699684,
0.19688486936548952, 0.1933606086706862, 0.18999425782371518, 0.1863192650014355
3, 0.18268733850129198, 0.1792779213321849, 0.1757751937984496, 0.172337065747918
46, 0.16892047085845535, 0.16554694229112835, 0.16197961527418892, 0.158541487223
65777, 0.1550459374102785, 0.15167240884295147, 0.14818403674992822, 0.1448392190
6402525, 0.14132931380993397, 0.13807062876830317, 0.13474734424346826, 0.1317973
0117714614, 0.12881854722939995, 0.1260120585701981, 0.12286821705426357, 0.12012
632787826585, 0.11737008326155612, 0.11492965834051105, 0.11239592305483778, 0.10
968992248062015, 0.1070198105081826, 0.10423485501004881, 0.10170111972437554, 0.
09939707149009475, 0.09676284811943726, 0.09381998277347114, 0.0910063163939133,
0.08839362618432385, 0.0856732701693942, 0.08327591157048521, 0.0805268446741314
9, 0.07826586276198678, 0.0759187482055699, 0.07334912431811656, 0.07103072064312
374, 0.06866207292563882, 0.06613551536032156, 0.0635156474303761, 0.061118288831
467124, 0.05886448463967844, 0.056244616709732985, 0.05373959230548378, 0.0512417
4562159058, 0.04865058857306919, 0.046296296296294, 0.04370513924777491, 0.041
45851277634223, 0.038723801320700545, 0.03626902095894344, 0.03375681883433821,
0.031158484065460807, 0.028897502153316105, 0.026686764283663508, 0.0246124031007
7519, 0.02248062015503876, 0.02049956933677864, 0.01884151593453919, 0.0169681309
21619293, 0.015180878552971577, 0.013041917886879128, 0.011340798162503588, 0.008

921906402526558]
[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, 61.98848174044677,
92.32166014204117, 150.28654909005283, 252.51513547165612, 336.74469421213246, 45
1.9583133482145, 606.3894639288484, 745.8181285116065, 967.6220163963853, 1262.36
78812017265, 1614.7286055557397, 1935.0613162778106, 2422.5297977997443, 2825.906
5595442926, 3302.8004099962395, 3797.069488915414, 4434.861938068663, 5084.204774
908244, 5811.812271910618, 6654.083839458281, 7498.05083968308, 8294.54056969950
8, 9224.471053041952, 10094.995855633975, 10981.90748407118, 11874.058107657947,
12700.262233625286, 13513.16697219943, 14242.439115849693, 14934.027179417686, 15
508.73874096093, 16010.51138969779, 16428.859501362585, 16693.21091552884, 16896.
574171769094, 16791.614596525666, 16639.555509984588, 15285.109282959205, 14929.3
8280761257, 11714.149909253289, 11004.293147860539, 10219.27768660759, 5679.23173
0243677, 5073.579737247771, 123.24141472617991, 58.648461752317374, 2001.04033179
87073, 2392.6577829860516, 5268.950353010444, 5254.354398392788, 12176.5551900666
92, 11966.383065741711, 27095.314780453125, 27308.658600584353, 35666.0430300879,
35044.565469401445, 36417.39718031364, 35744.19751121408, 34966.001248931796, 343
64.21346615379, 33869.73025210203, 34882.65140912109, 34233.45786129774, 38077.86
726340957, 37892.59322685242, 38608.978138924576, 38371.96552260224, 38920.518325
10232, 38486.38704964327, 39099.23946542174, 38562.512981668515, 37996.8968015979
84, 37325.176097930074, 36576.06860975904, 35990.27658861827, 35439.637661031986,
34664.11432575609, 33948.22821951572, 33371.95834681607, 32809.98269033177, 3235
0.1691246611, 31835.236527612855, 31421.315771047073, 30957.785194881373, 30456.7
83100444798, 29937.425032863102, 29462.358891135307, 29218.201456017257, 28865.72
9827187282, 28599.979538845462, 28164.93370452049, 27727.83281458891, 27205.89389
2188928, 26791.151947890543, 26318.000060818737, 25824.196706313553, 25334.684819
428345, 24717.959749172576, 24140.88689555606, 23520.394726797338, 22924.47435602
226, 22279.254845194115, 21623.155744369942, 21027.613682123414, 20374.1356785246
97, 19783.584683928726, 19150.37667067323, 18550.43749592747, 17986.08413662939,
17436.300511177742, 16884.86605552509, 16344.195823915403, 15839.515420049456, 15
401.198514715632, 14924.242221230443, 14540.532352804059, 14212.711742958652, 137
67.585734728604, 13413.065314658827, 13030.091103960614, 12747.55929417193, 1252
2.891160294801, 12266.38492509541, 12029.191798785081, 11857.460118545829, 11606.
150145322637, 11324.976516729532, 11137.900617467036, 10911.905464272108, 10729.9
6363568908, 10563.379058340091, 10455.443945459181, 10280.178002568477, 10165.480
154053963, 9989.952582858228, 9942.919092993956, 9748.17022235454, 9569.551095162
462, 9427.292390892113, 9330.808811245057, 9246.056111003629, 9124.673239037129,
9010.161460018326, 8861.194049524596, 8754.07970970982, 8607.99298769701, 8466.51
220174514, 8319.856802229502, 8234.220058568182, 8068.708005184971, 7949.40567878
9221, 7776.627633125015, 7637.815868632434, 7497.522087916895, 7422.327352025934
5, 7302.604028162345, 7246.913389650269, 7170.6809499378405, 7128.1368660517555,
7061.761450188312, 7044.24004411219, 7020.608883601425, 6963.315050297974, 6921.1
71306711046, 6869.894954323502, 6816.138822625666, 6756.686477806684, 6727.137385
773079, 6678.824717381586, 6639.723188232126, 6583.177662257657, 6544.22562804478
2, 6484.84322760639, 6450.129261835974, 6379.615012464027, 6319.17352520749, 620
6.365881134064, 6099.00247088182, 5968.83685748061, 5887.1711576198695, 5750.9946
97770723, 5618.501361354062, 5444.796937712134, 5285.947594519908, 5152.173961266
353, 5015.306044917024, 4895.263380026669, 4743.851567461469, 4564.580251645775,
4430.125414770768, 4336.7441278136575, 4228.022933260677, 4095.281411401857, 397
7.5980935110015, 3821.277826325603, 3710.1775665846976, 3540.7655199261526, 3383.
923602451769, 3255.81670080448, 3099.2056842458364, 2950.485891121189, 2822.19587
14001767, 2706.351557801227, 2565.983755607728, 2410.7164646357214, 2299.31283941
45565, 2176.102785504559, 2053.513476556655, 1942.8305452766735, 1806.96471456467
8, 1698.953194010894, 1554.2044780122785, 1464.5655963791087, 1345.4623542330887,
1233.9123964451135, 1132.854427912843, 996.9480826138233, 857.3360445879867, 705.
2859676476226, 561.64129905137, 406.35270945768445, 229.72962744429688, 93.886259
05862374, 4.781418654412587, 46.36922682568072, 676.1637732487703, 4325.178973678
15]

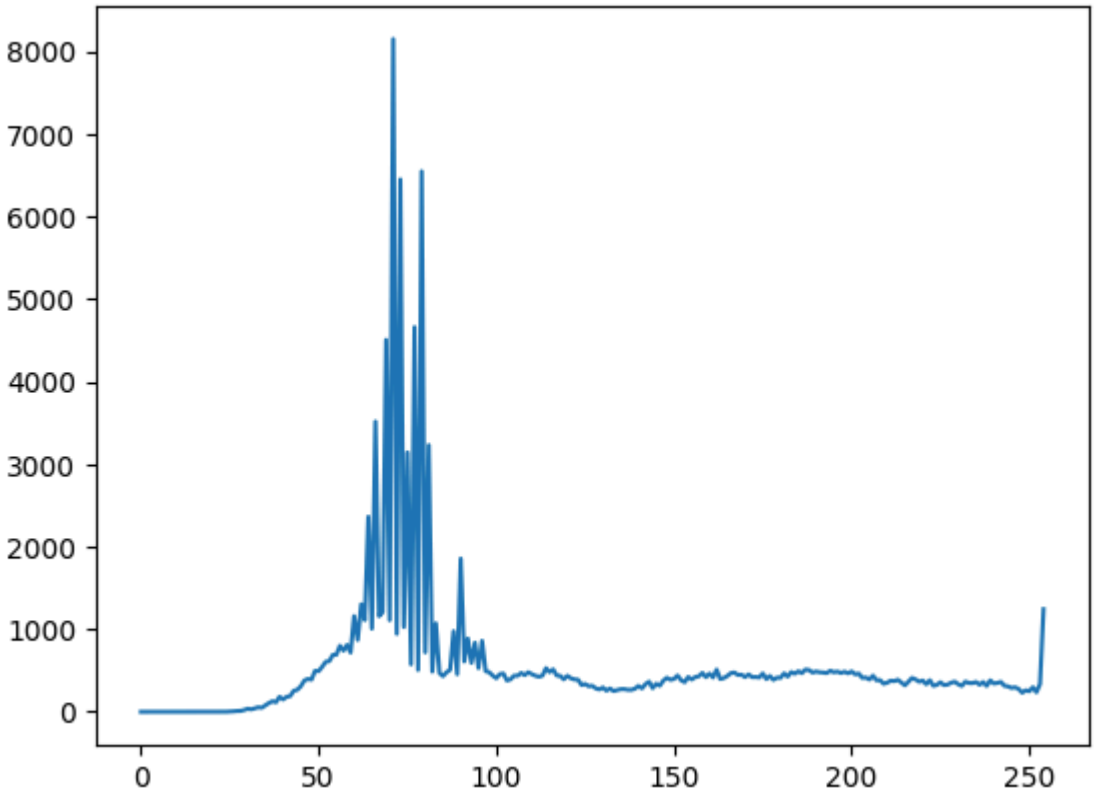


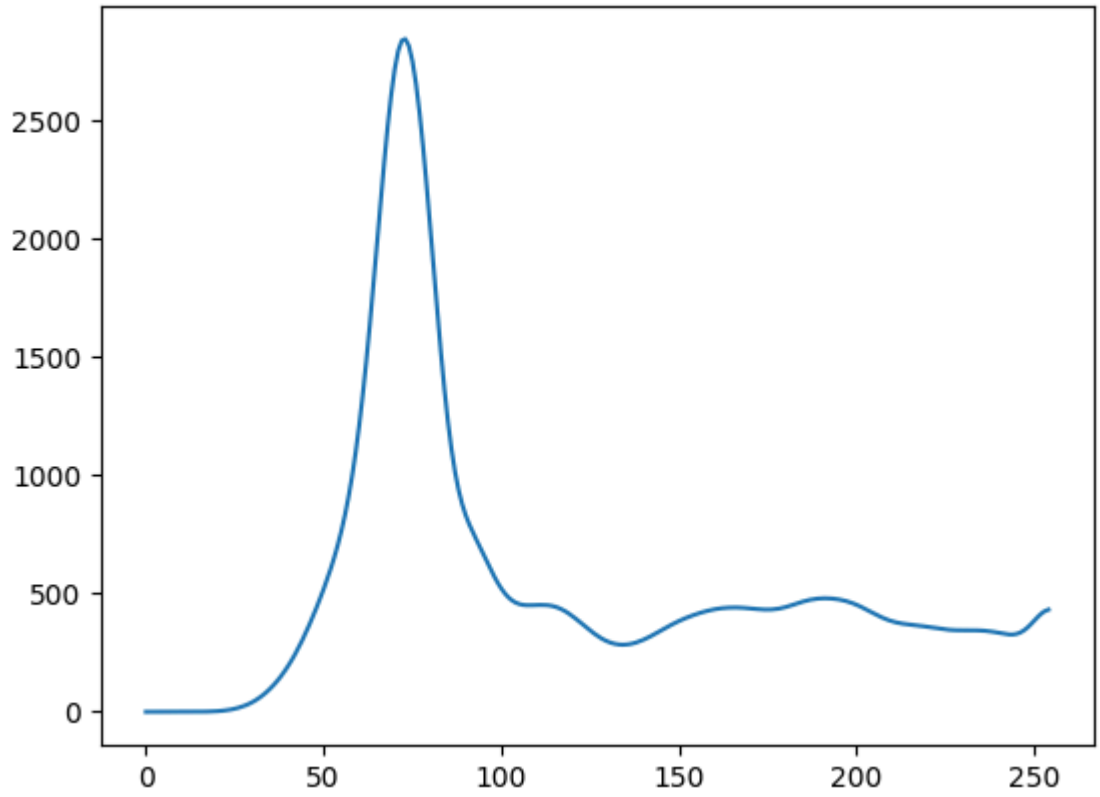


In [ ]:
```python
def two_peaks_threshold(image, smooth_hist=True, sigma=5):
    from scipy.ndimage import gaussian_filter

    hist = np.histogram(image, bins=range(256))[0].astype(np.float64)
    plt.plot(hist)
    plt.show()

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

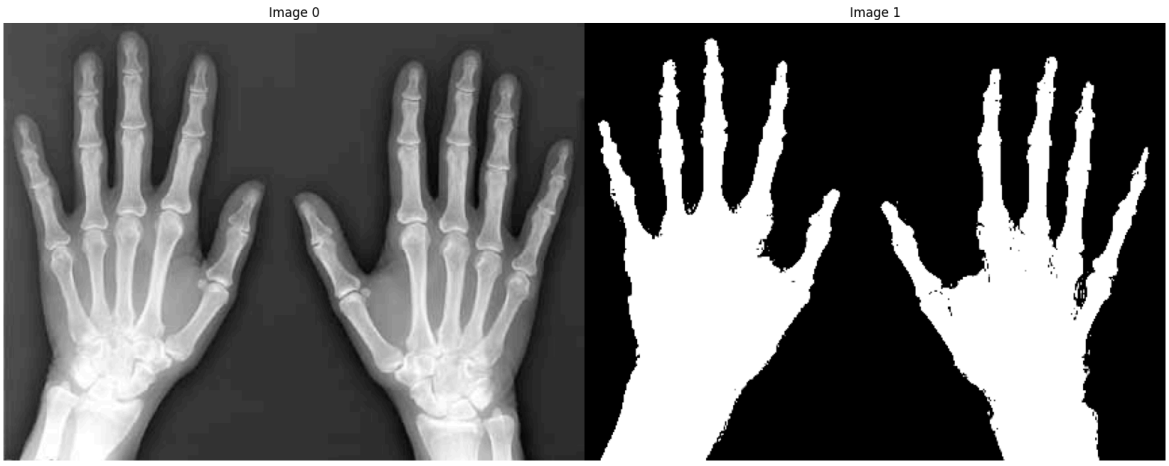
```
        plt.show()

    f_peak = np.argmax(hist)

    # finding second peak
    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
```

In [ ]:
```
final_img, final_thresh, hist = two_peaks_threshold(image_gray)
ShowImage([image_gray, final_img], 1, 2)
```





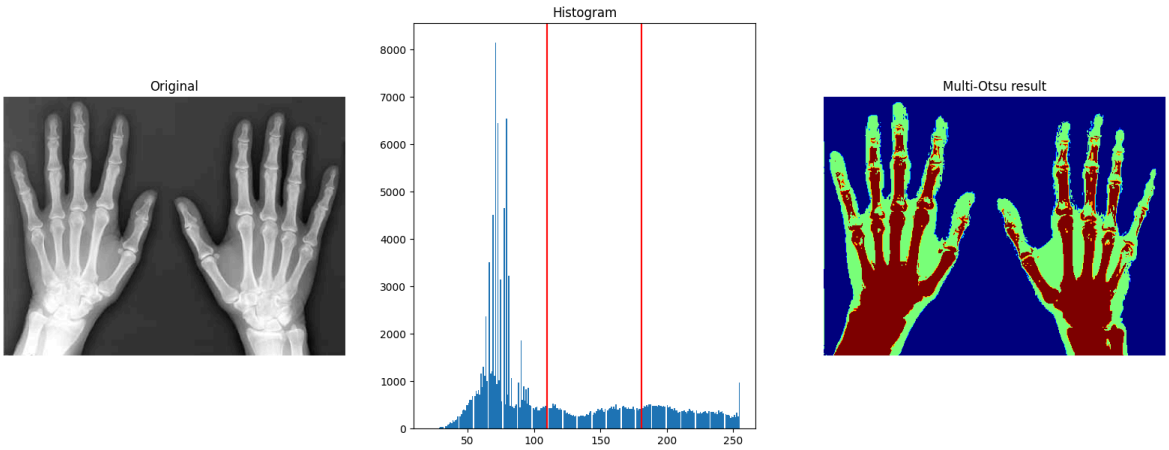
134



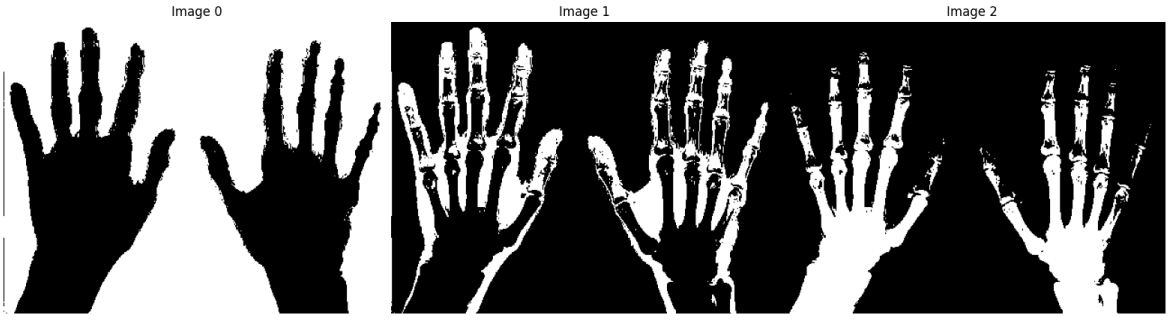
In [ ]:

```
from skimage.filters import threshold_multiotsu
# Applying multi-Otsu threshold for the default value, generating
# three classes.
thresholds = threshold_multiotsu(image_gray)
# Using the threshold values, we generate the three regions.
regions = np.digitize(image_gray, bins=thresholds)

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(20, 7))
# Plotting the original image.
ax[0].imshow(image_gray, 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_gray.ravel(), bins=255)
ax[1].set_title('Histogram')
for thresh in thresholds:
    ax[1].axvline(thresh, color='r')
# Plotting the Multi Otsu result.
```

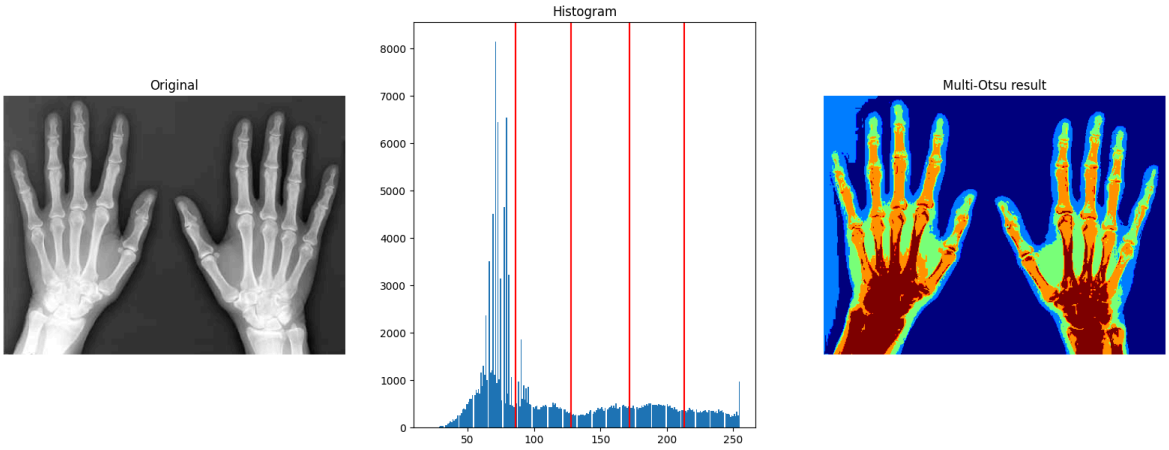
```
ax[2].imshow(regions, cmap='jet')
ax[2].set_title('Multi-Otsu result')
ax[2].axis('off')
plt.subplots_adjust()
plt.show()
```



In [ ]:
```
Segments = []
for idx in list(np.unique(regions)):
  mask = regions == idx
  Segments.append(mask)

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

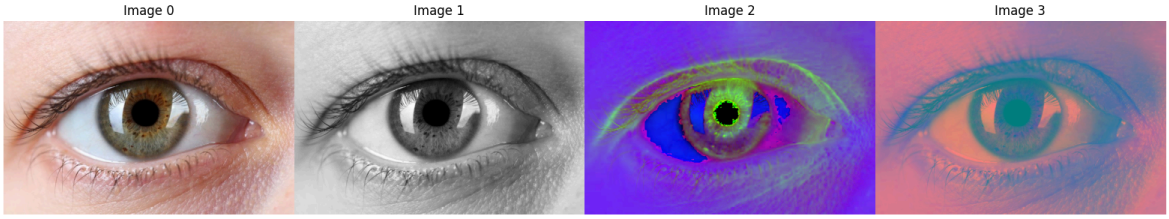


In [ ]:
```
thresholds = threshold_multiotsu(image_gray, classes=5)
regions = np.digitize(image_gray, bins=thresholds)

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(20, 7))
# Plotting the original image.
ax[0].imshow(image_gray, 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_gray.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()
```



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

image_orig = ImageDB[idx]
image_orig = cv2.cvtColor(image_orig,cv2.COLOR_BGR2RGB)
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  17
Name  Iris.jpg
```

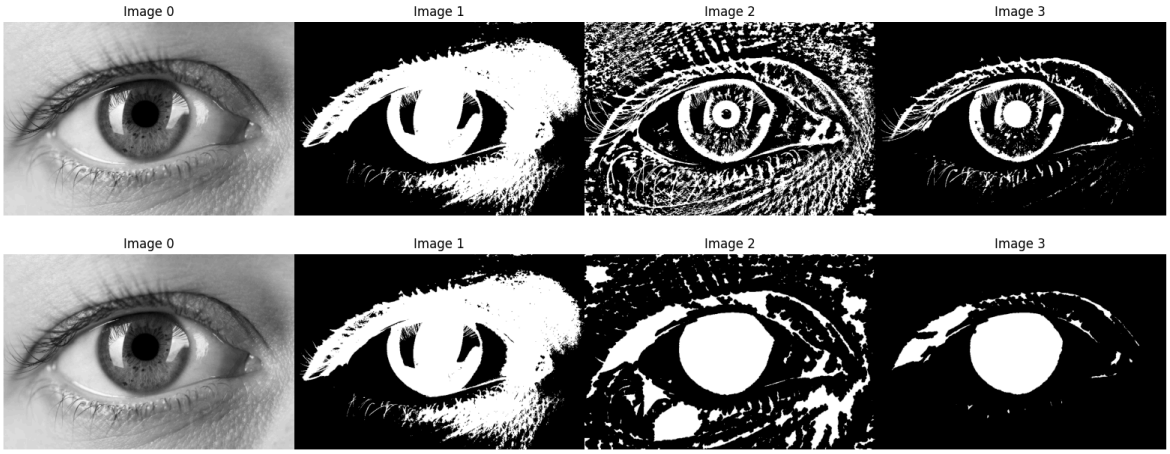


In [ ]:
```
from scipy import ndimage as ndi
def FillHoles(Mask):
  Result = ndi.binary_fill_holes(Mask)
  return Result
```

In [ ]:
```
def SelectLargestRegion(Mask):
  import pandas as pd
  from skimage.measure import label, regionprops

  mask = Mask.copy()
  mask_output = mask * 0
  label_img = label(mask)
  regions = regionprops(label_img)
  max_area = 0
  ilabel = 0
  for props in regions:
    area = props.area
    if(area > max_area):
      max_area = area
      ilabel = props.label

  mask_output = mask_output + (label_img == ilabel).astype(int)
  return mask_output
```

In [ ]:
```
def morphology(Mask, Size):
  from skimage.morphology import erosion, dilation, opening, closing, white_toph
  from skimage.morphology import disk
  selem = disk(abs(Size))
  if(Size > 0):
      result = dilation(Mask, selem)
  else:
      result = erosion(Mask, selem)
  return result
```

In [ ]:
```
from skimage.filters import (threshold_otsu, threshold_niblack, threshold_sauvol
image = image_gray
binary_global = image < threshold_otsu(image)

thresh_niblack = threshold_niblack(image, window_size=45)
thresh_sauvola = threshold_sauvola(image, window_size=45)

binary_niblack = image < thresh_niblack
binary_sauvola = image < thresh_sauvola

ShowImage([image, binary_global, binary_niblack, binary_sauvola], 1, 4)

binary_niblack = FillHoles(binary_niblack)
binary_sauvola = FillHoles(binary_sauvola)

binary_niblack = morphology(binary_niblack, -3)
binary_sauvola = morphology(binary_sauvola, -3)

ShowImage([image, binary_global, binary_niblack, binary_sauvola], 1, 4)
```



In [ ]:
```
def SegmentColorImageByMask(IM, Mask):
    Mask = Mask.astype(np.uint8)
    result = cv2.bitwise_and(IM, IM, mask = Mask)
    return result
```

In [ ]:
```
binary_niblack_segmentation = SelectLargestRegion(binary_niblack)
binary_sauvola_segmentation = SelectLargestRegion(binary_sauvola)

ShowImage([image, binary_niblack_segmentation, binary_niblack_segmentation], 1,

binary_niblack_color = SegmentColorImageByMask(image_orig, binary_niblack_segmen
binary_sauvola_color = SegmentColorImageByMask(image_orig, binary_sauvola_segmen

ShowImage([image_orig, binary_niblack_color, binary_sauvola_color], 1, 4)
```

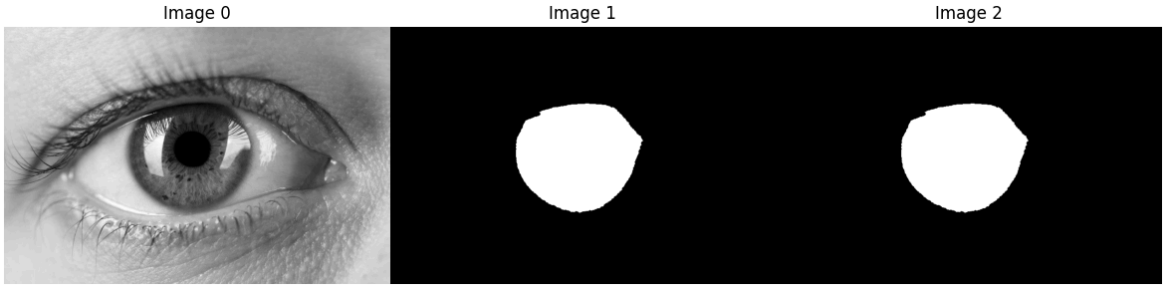


Image 0
Image 1
Image 2

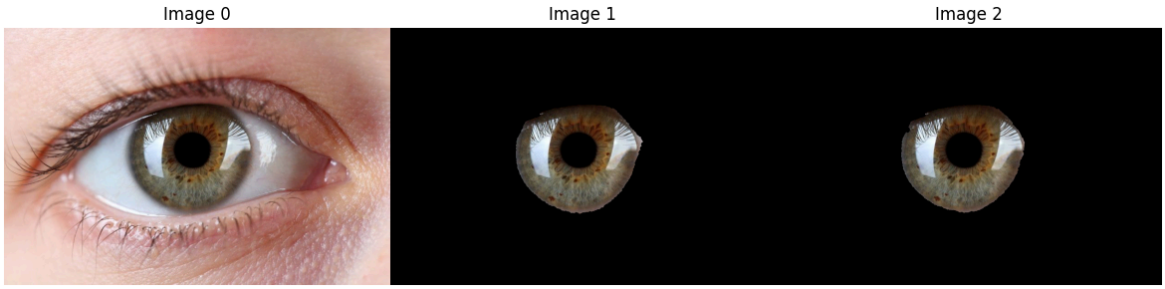


Image 0
Image 1
Image 2