LAB02 - Image Segmentation

Dr. Tran Anh Tuan,

Faculty of Mathematics and Computer Science,

University of Science, HCMC

In [1]:
```
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 [2]:
```
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) # set the spacing between axes.
    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 [3]:
```
import os
import pandas as pd

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

In [4]:
```
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 [5]:
```
# Mount drive
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [6]:
```
import os
path_Data = "//content//gdrive//MyDrive//Teaching Class Drive//Bien Hinh va Xu Ly Anh (Image Segmentation)//O
bject 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
```

In [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
```

In [8]:
```
NameDB
```

Out[8]:
```
['Lung.png',
 'Iris.jpg',
 'Melanoma.jpg',
 'Retina.jpg',
 'Face.jpg',
 'Fire.jpg',
 'Mask.jpg',
 'Sign.jpg',
 'Cross.jpg',
 'Shelf.jpg',
 'Brain.jpg',
 'Tumor.png',
 'Hand.jpg',
 'Chest.jpg',
 'Bone.jpg',
 'Gesture.jpg',
 'Emotion.jpg',
 'Car.jpg',
 'Activities.jpeg',
 'Crack.jpg',
 'Code.jpg',
 'Dust.jpg',
 'Barcode.png',
 'QR.jpg',
 'Leaf.jpg',
 'Cloths.jpg',
 'Writing.png',
 'Defect.jpg']
```

KMeans Clustering

In [32]:
```
FileName = 'Emotion.jpg'
idx = NameDB.index(FileName)
print("Selected Image : ", "\nIndex ", idx, "\nName ", NameDB[idx])

image_orig = ImageDB[idx]
image_orig = ResizeImage(image_orig, 300, 0)
img = 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, img, image_gray, image_hsv, image_ycbcr], 1, 5)
```

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

In [33]:
```
vectorized = img.reshape((-1,3))
vectorized = np.float32(vectorized)
```

In [34]:
```
vectorized
```

Out[34]:
```
array([[ 27.,  54., 135.],
       [ 27.,  54., 135.],
       [ 24.,  51., 130.],
       ...,
       [ 19.,  19.,  27.],
       [ 19.,  23.,  35.],
       [ 13.,  20.,  28.]], dtype=float32)
```

In [35]:
```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
```

OpenCV provides cv2.kmeans(samples, nclusters(K), criteria, attempts, flags) function for color clustering.

1. samples: It should be of np.float32 data type, and each feature should be put in a single column.
2. nclusters(K): Number of clusters required at the end
3. criteria: It is the iteration termination criteria. When this criterion is satisfied, the algorithm iteration stops. Actually, it should be a tuple of 3 parameters. They are `( type, max_iter, epsilon )` :

Type of termination criteria. It has 3 flags as below:

cv.TERM_CRITERIA_EPS — stop the algorithm iteration if specified accuracy, epsilon, is reached. cv.TERM_CRITERIA_MAX_ITER — stop the algorithm after the specified number of iterations, max_iter. cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER — stop the iteration when any of the above condition is met.

1. attempts: Flag to specify the number of times the algorithm is executed using different initial labelings. The algorithm returns the labels that yield the best compactness. This compactness is returned as output.
2. flags: This flag is used to specify how initial centers are taken. Normally two flags are used for this: cv.KMEANS_PP_CENTERS and cv.KMEANS_RANDOM_CENTERS.

In [36]:
```
K = 3
attempts=10
ret,label,center=cv2.kmeans(vectorized,K,None,criteria,attempts,cv2.KMEANS_PP_CENTERS)
```

In [37]:
```
center = np.uint8(center)
res = center[label.flatten()]
result_label = label.reshape((img.shape[:2]))
result_image = res.reshape((img.shape))
```

In [38]:
```
center
```

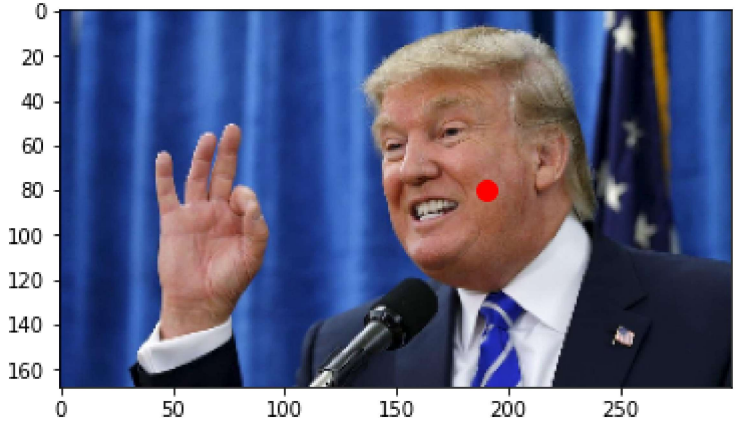
Out[38]:
```
array([[ 45,  79, 158],
       [ 35,  35,  55],
       [175, 142, 131]], dtype=uint8)
```

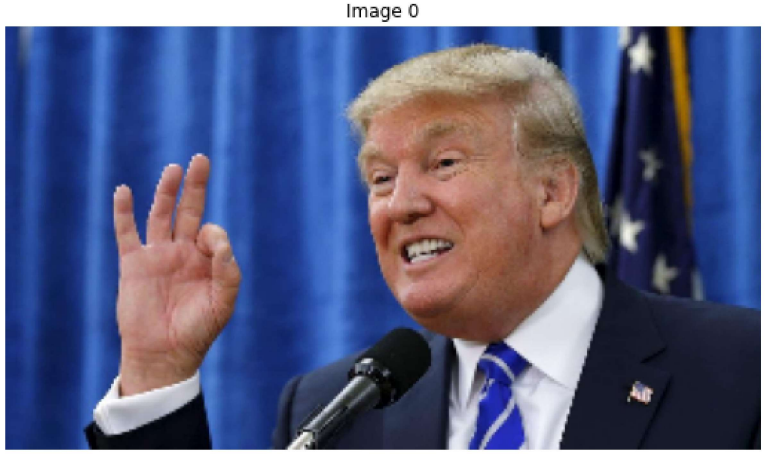
In [40]:
```
ShowImage([img, result_label, result_image], 1, 3)
print(img.shape)
```


```
(168, 300, 3)
```

In [52]:
```
rpoint = 80
cpoint = 190
idx = result_label[rpoint, cpoint]
print("Index at ({0},{1}) : {2}".format(rpoint, cpoint, idx))
plt.imshow(img)
plt.plot(cpoint, rpoint, "or", markersize=10)  # og:shorthand for green circle
plt.show()

SegMask = result_label == idx
ShowImage([img, SegMask], 1, 2)
```

```
Index at (80,190) : 2
```

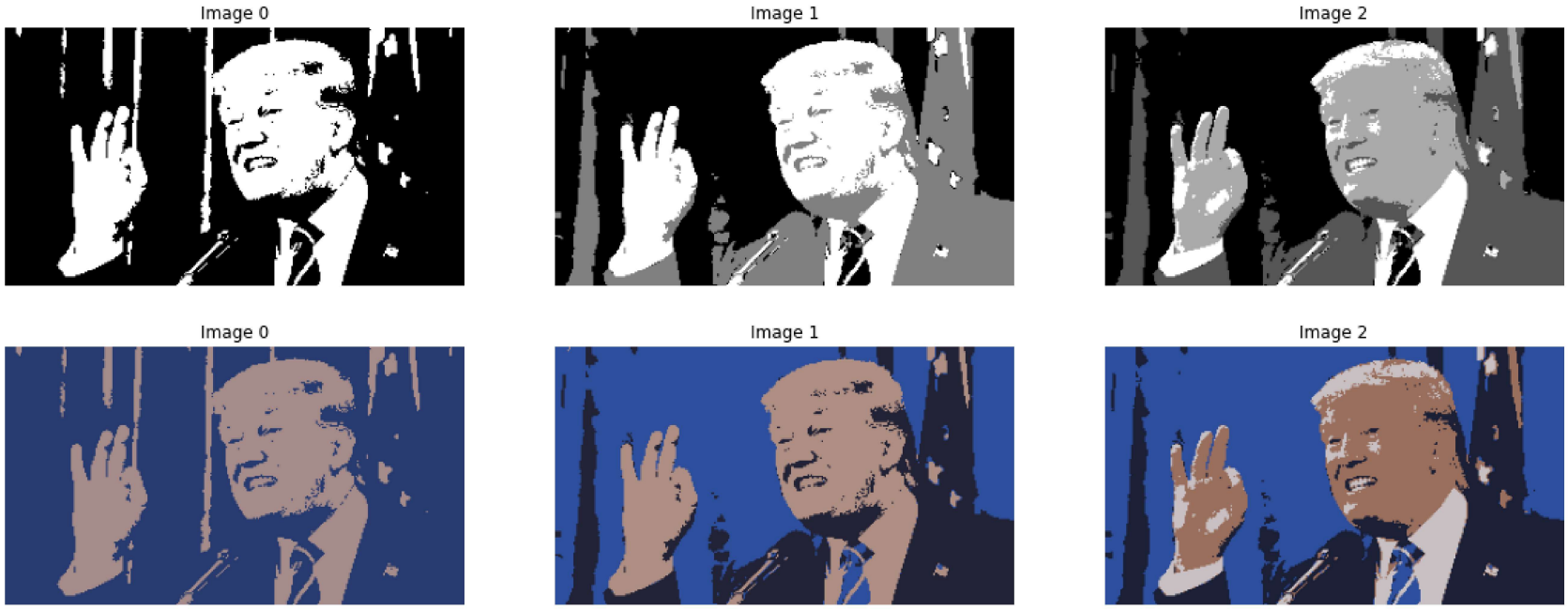




In [29]:
```
def ReArrangeIndex(image_index):
  AreaList = []
  for idx in range(image_index.max() + 1):
      mask = image_index == idx
      AreaList.append(mask.sum().sum())

  sort_index = np.argsort(AreaList)[::-1]
  index = 0
  image_index_rearrange = image_index * 0
  for idx in sort_index:
      image_index_rearrange[image_index == idx] = index
      index = index + 1
  return image_index_rearrange

def KmeansSegmentation(img, K = 3):
  vectorized = img.reshape((-1,3))
  vectorized = np.float32(vectorized)
  criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)

  attempts=10
  ret,label,center=cv2.kmeans(vectorized,K,None,criteria,attempts,cv2.KMEANS_PP_CENTERS)
  center = np.uint8(center)
  res = center[label.flatten()]
  result_label = label.reshape((img.shape[:2]))
  result_image = res.reshape((img.shape))

  return center, result_label, result_image
```

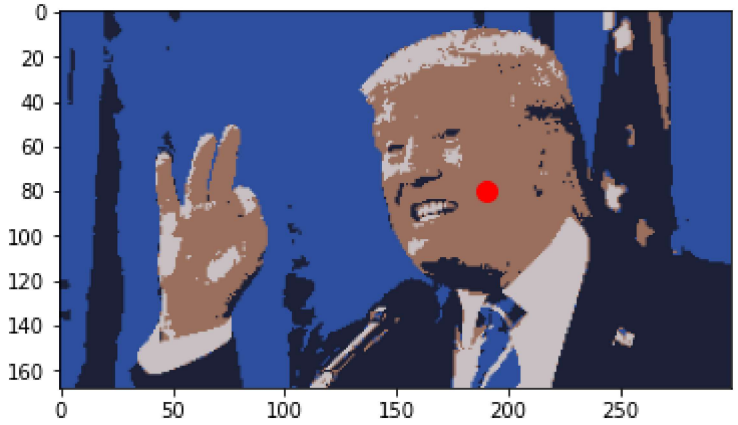
In [53]:
```
list_center = []
list_result_label = []
list_result_image = []
for K in [2,3,4]:
  center, result_label, result_image = KmeansSegmentation(img, K)
  result_label = ReArrangeIndex(result_label)
  list_center.append(center)
  list_result_label.append(result_label)
  list_result_image.append(result_image)
```

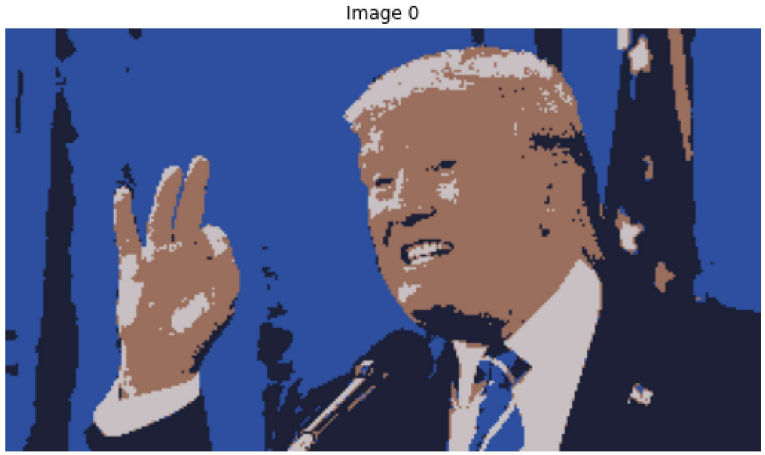
In [55]:
```
ShowImage(list_result_label, 1, 3)
ShowImage(list_result_image, 1, 3)
```

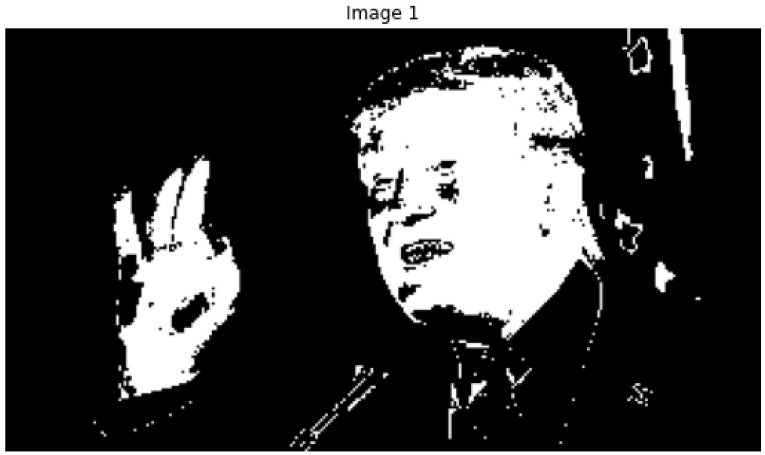


In [56]:
```
img_select = list_result_image[2]
result_label_select = list_result_label[2]

rpoint = 80
cpoint = 190
idx = result_label_select[rpoint, cpoint]
print("Index at ({0},{1}) : {2}".format(rpoint, cpoint, idx))
plt.imshow(img_select)
plt.plot(cpoint, rpoint, "or", markersize=10)  # og:shorthand for green circle
plt.show()

SegMask = result_label_select == idx
ShowImage([img_select, SegMask], 1, 2)
```

```
Index at (80,190) : 2
```







Fuzzy C Means Clustering

ref: [https://github.com/ariffyasri/fuzzy-c-means/blob/master/fuzzy-c-means-scikit-fuzzy-image.ipynb (https://github.com/ariffyasri/fuzzy-c-means/blob/master/fuzzy-c-means-scikit-fuzzy-image.ipynb)](https://github.com/ariffyasri/fuzzy-c-means/blob/master/fuzzy-c-means-scikit-fuzzy-image.ipynb)

In [61]:
```
pip install -U scikit-fuzzy
```

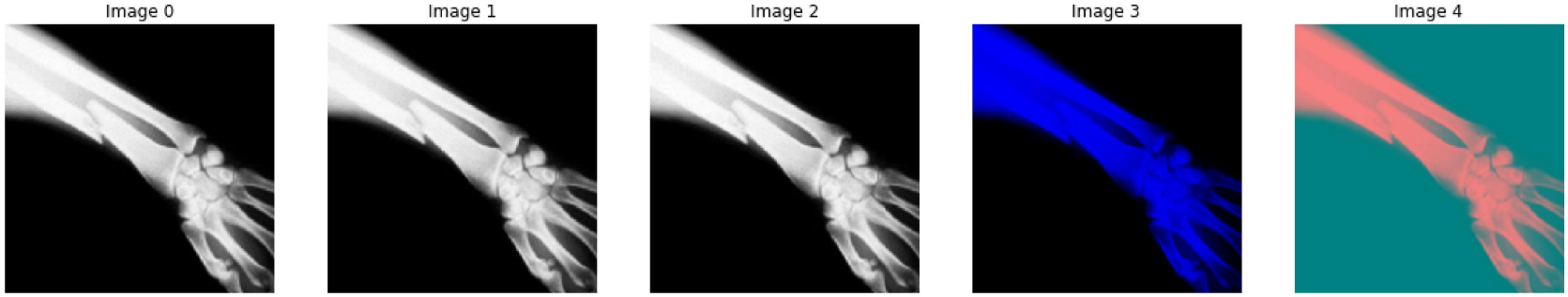
```
Collecting scikit-fuzzy
  Downloading https://files.pythonhosted.org/packages/6c/f0/5eb5dbe0fd8dfe7d4651a8f4e591a196623a22b9e5339101e
559695b4f6c/scikit-fuzzy-0.4.2.tar.gz (993kB)
     |████████████████████████████████| 1.0MB 3.1MB/s
Requirement already satisfied, skipping upgrade: numpy>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from
scikit-fuzzy) (1.19.5)
Requirement already satisfied, skipping upgrade: scipy>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from
scikit-fuzzy) (1.4.1)
Requirement already satisfied, skipping upgrade: networkx>=1.9.0 in /usr/local/lib/python3.7/dist-packages (f
rom scikit-fuzzy) (2.5.1)
Requirement already satisfied, skipping upgrade: decorator<5,>=4.3 in /usr/local/lib/python3.7/dist-packages
(from networkx>=1.9.0->scikit-fuzzy) (4.4.2)
Building wheels for collected packages: scikit-fuzzy
  Building wheel for scikit-fuzzy (setup.py) ... done
  Created wheel for scikit-fuzzy: filename=scikit_fuzzy-0.4.2-cp37-none-any.whl size=894069 sha256=5ad40210af
9f29dc9bd1ff4601122800e897af20515349b5d57eddd4eca6a161
  Stored in directory: /root/.cache/pip/wheels/b9/4e/77/da79b16f64ef1738d95486e2731eea09d73e90a72465096600
Successfully built scikit-fuzzy
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.4.2
```

In [62]:
```
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz
import os
import cv2
import numpy as np
from time import time
```

In [63]:
```
def change_color_fuzzycmeans(cluster_membership, clusters):
    img = []
    for pix in cluster_membership.T:
        img.append(clusters[np.argmax(pix)])
    return img
```

In [69]:
```
FileName = 'Bone.jpg'
idx = NameDB.index(FileName)
print("Selected Image : ", "\nIndex ", idx, "\nName ", NameDB[idx])

image_orig = ImageDB[idx]
image_orig = ResizeImage(image_orig, 200, 200)
img = 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, img, image_gray, image_hsv, image_ycbcr], 1, 5)
```

```
Selected Image :
Index  14
Name  Bone.jpg
```



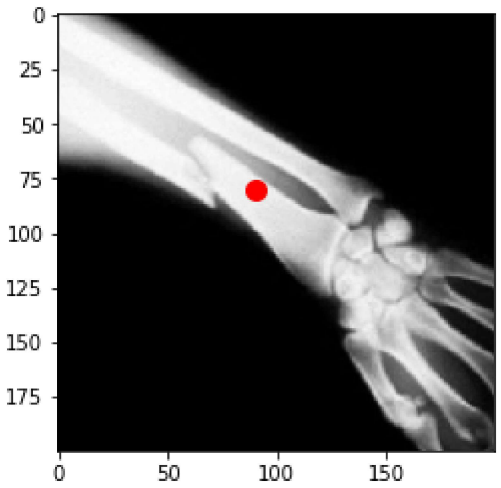
In [99]:
```
clusters = [2,3,6]
rgb_img = image_orig.reshape((image_orig.shape[0] * image_orig.shape[1], 3))
img = np.reshape(rgb_img, (200,200,3)).astype(np.uint8)
shape = np.shape(img)
print(shape)
```

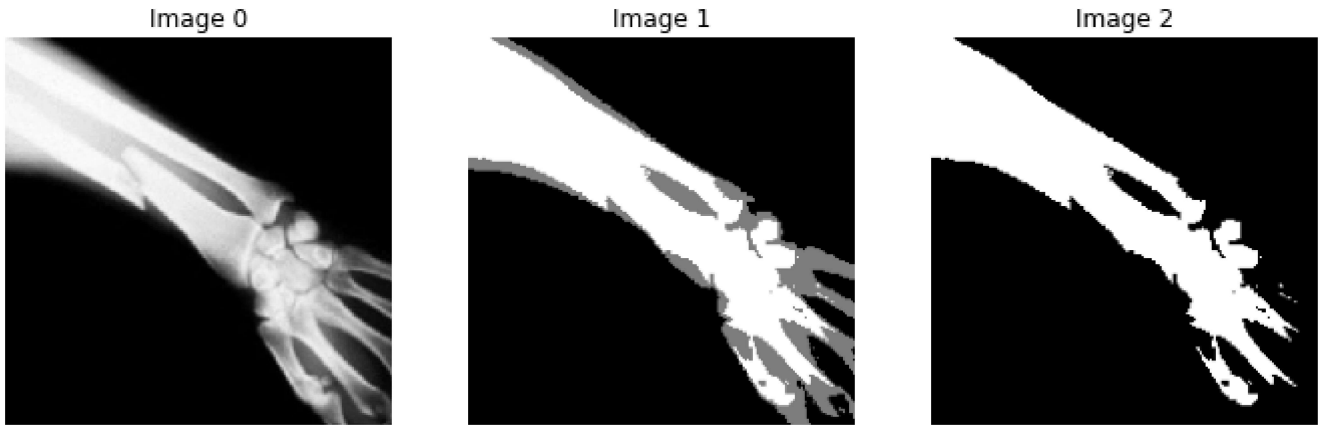
```
(200, 200, 3)
```

In [110]:
```
cluster = 3
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(rgb_img.T, cluster, 2, error=0.005, maxiter=1000, init=None,
seed=42)
new_img = change_color_fuzzycmeans(u,cntr)
fuzzy_img = np.reshape(new_img,shape).astype(np.uint8)
result_label = fuzzy_img[:,:,1]
```

In [114]:
```
rpoint = 80
cpoint = 90
idx = result_label[rpoint, cpoint]
print("Index at ({0},{1}) : {2}".format(rpoint, cpoint, idx))
plt.imshow(img)
plt.plot(cpoint, rpoint, "or", markersize=10)  # og:shorthand for green circle
plt.show()

SegMask = result_label == idx
ShowImage([img, result_label, SegMask], 1, 5)
```

```
Index at (80,90) : 225
```





In [115]:
```
list_result_label = []
# looping every cluster
for i,cluster in enumerate(clusters):
    # Fuzzy C Means
    rgb_img = image_orig.reshape((image_orig.shape[0] * image_orig.shape[1], 3))
    img = np.reshape(rgb_img, (200,200,3)).astype(np.uint8)

    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(rgb_img.T, cluster, 2, error=0.005, maxiter=1000, init=N
one,seed=42)
    new_img = change_color_fuzzycmeans(u,cntr)
    fuzzy_img = np.reshape(new_img,shape).astype(np.uint8)
    result_label = fuzzy_img[:,:,1]
    list_result_label.append(result_label)
```

In [116]:
```
ShowImage(list_result_label, 1, 3)
```

Meanshift

In [178]:
```
import numpy as np
from sklearn.cluster import MeanShift, estimate_bandwidth
from sklearn.datasets.samples_generator import make_blobs
from itertools import cycle
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.datasets.
samples_generator module is  deprecated in version 0.22 and will be removed in version 0.24. The correspondin
g classes / functions should instead be imported from sklearn.datasets. Anything that cannot be imported from
sklearn.datasets is now part of the private API.
  warnings.warn(message, FutureWarning)
```

In [197]:
```
FileName = 'Cloths.jpg'
idx = NameDB.index(FileName)
print("Selected Image : ", "\nIndex ", idx, "\nName ", NameDB[idx])

resize_w = 200
resize_h = 200

image_orig = ImageDB[idx]
image_orig = ResizeImage(image_orig, resize_w, resize_h)
img = 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, img, image_gray, image_hsv, image_ycbcr], 1, 5)
```

```
Selected Image :
Index  25
Name  Cloths.jpg
```

In [198]:
```
# Image is (687 x 1025, RGB channels)
image = np.array(img)
original_shape = image.shape

# Flatten image.
X = np.reshape(image, [-1, 3])
```

In [199]:
```
bandwidth = estimate_bandwidth(X, quantile=0.1, n_samples=100)
print(bandwidth)
```

```
27.443327850030514
```

In [200]:
```
ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)
ms.fit(X)
```

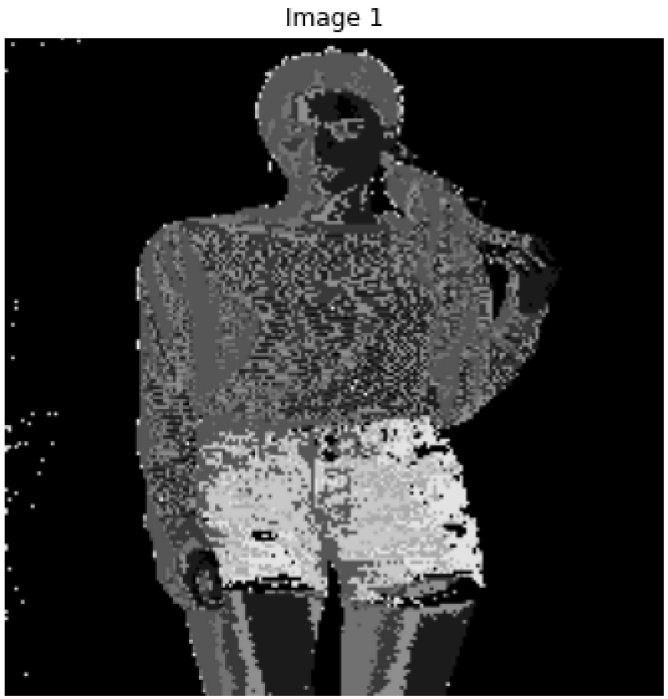
Out[200]:
```
MeanShift(bandwidth=27.443327850030514, bin_seeding=True, cluster_all=True,
          max_iter=300, min_bin_freq=1, n_jobs=None, seeds=None)
```

In [201]:
```
labels = ms.labels_
print(labels.shape)
cluster_centers = ms.cluster_centers_
print(cluster_centers.shape)

labels_unique = np.unique(labels)
n_clusters_ = len(labels_unique)

print("number of estimated clusters : %d" % n_clusters_)
```

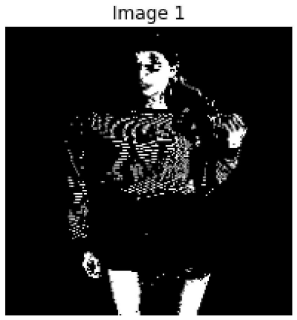
```
(40000,)
(10, 3)
number of estimated clusters : 10
```

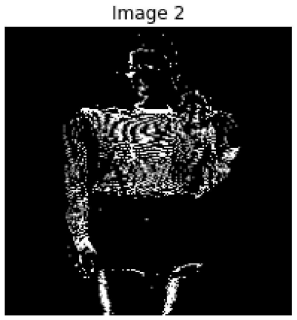
In [202]:
```
segmented_image = np.reshape(labels, original_shape[:2])  # Just take size, ignore RGB channels.
```

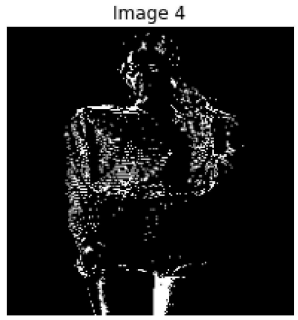
In [203]:
```
ShowImage([image_orig, segmented_image], 1, 3)
```



In [204]:
```
ShowImage([image_orig, segmented_image != 0], 1, 3)
ShowImage([segmented_image == 0, segmented_image == 1, segmented_image == 2, segmented_image == 3, segmented_image == 4], 1, 5)
```





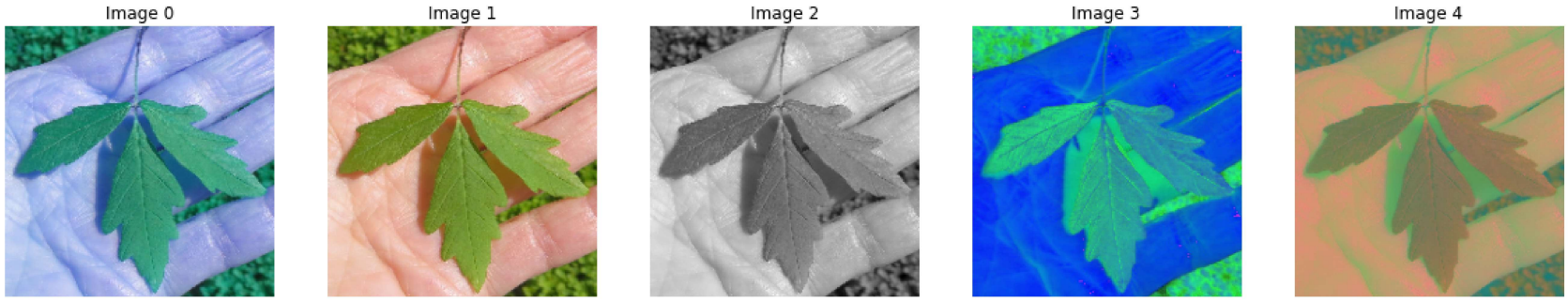


Hierachy Clustering

In [206]:
```
import time as time
import numpy as np
from scipy.ndimage.filters import gaussian_filter
import matplotlib.pyplot as plt
import skimage
from skimage.data import coins
from skimage.transform import rescale
from sklearn.feature_extraction.image import grid_to_graph
from sklearn.cluster import AgglomerativeClustering
```

In [216]:
```
FileName = 'Leaf.jpg'
idx = NameDB.index(FileName)
print("Selected Image : ", "\nIndex ", idx, "\nName ", NameDB[idx])

resize_w = 200
resize_h = 200

image_orig = ImageDB[idx]
image_orig = ResizeImage(image_orig, resize_w, resize_h)
img = 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, img, image_gray, image_hsv, image_ycbcr], 1, 5)
```

```
Selected Image :
Index  24
Name  Leaf.jpg
```



In [210]:
```
# smoothened_img = gaussian_filter(image_gray, sigma=2)
# X = np.reshape(smoothened_img, (-1, 1))
```

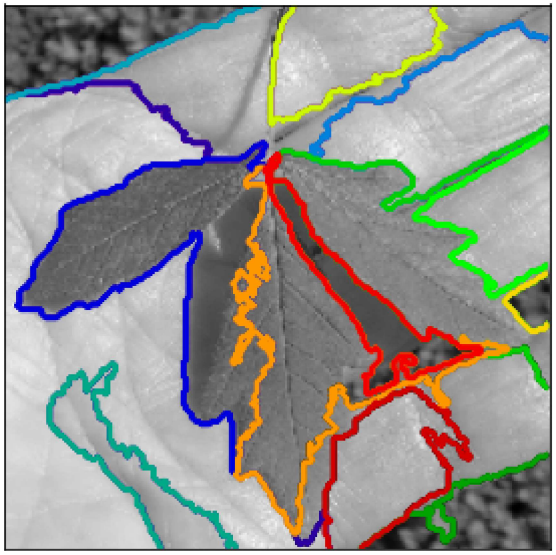
In [221]:
```
# smoothened_img = gaussian_filter(image_gray, sigma=2)
smoothened_img = image_gray.copy()
X = np.reshape(smoothened_img, (-1, 1))
```

In [222]:
```
# Define the structure A of the data. Pixels connected to their neighbors.
connectivity = grid_to_graph(*smoothened_img.shape)
```

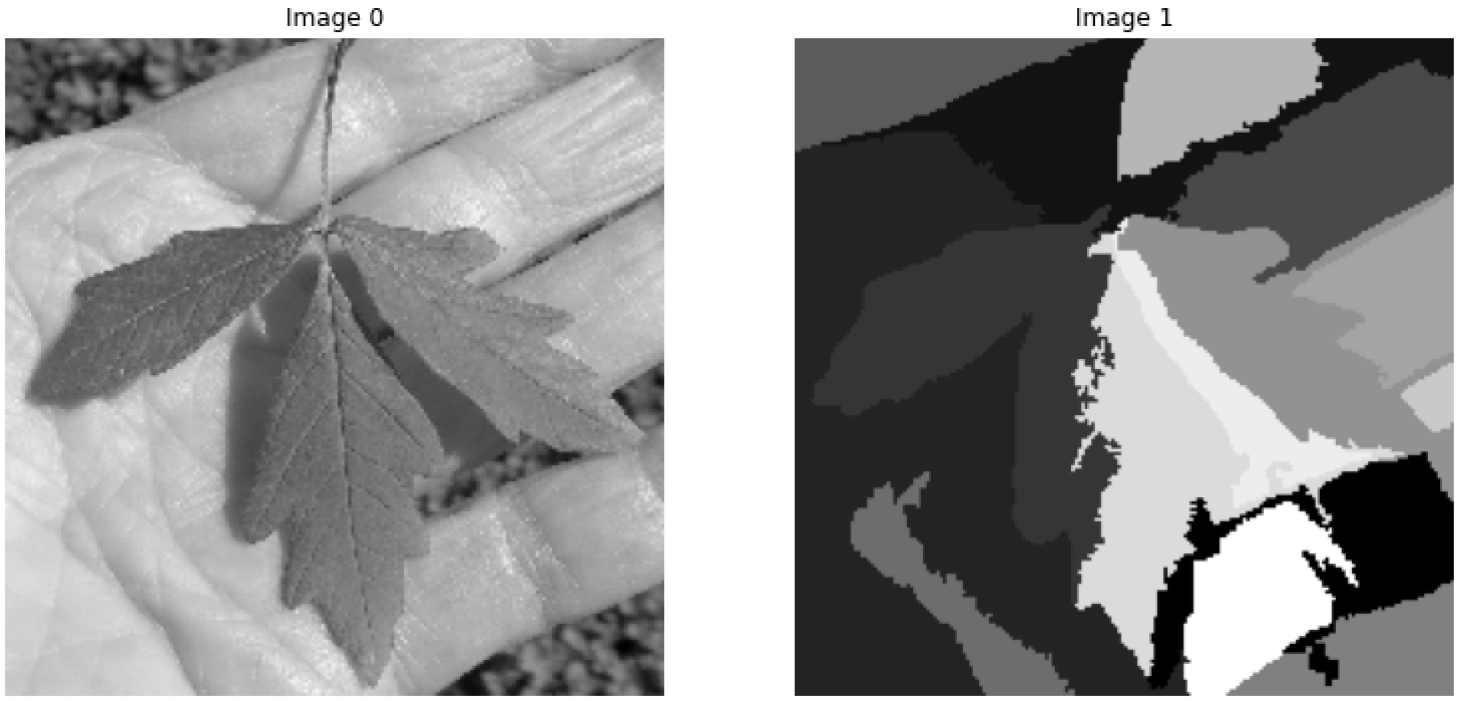
In [240]:
```
# Compute clustering
print("Compute structured hierarchical clustering...")
st = time.time()
n_clusters = 15  # number of regions
ward = AgglomerativeClustering(n_clusters=n_clusters, linkage='ward', connectivity=connectivity)
ward.fit(X)
result_label = np.reshape(ward.labels_, smoothened_img.shape)
print("Elapsed time: ", time.time() - st)
print("Number of pixels: ", label.size)
print("Number of clusters: ", np.unique(label).size)
```

```
Compute structured hierarchical clustering...
Elapsed time:  3.9659128189086914
Number of pixels:  40000
Number of clusters:  15
```

In [241]:
```
# Plot the results on an image
plt.figure(figsize=(5, 5))
plt.imshow(smoothened_img, cmap=plt.cm.gray)
for l in range(n_clusters):
    plt.contour(result_label == l,
                colors=[plt.cm.nipy_spectral(l / float(n_clusters)), ])
plt.xticks(())
plt.yticks(())
plt.show()
```

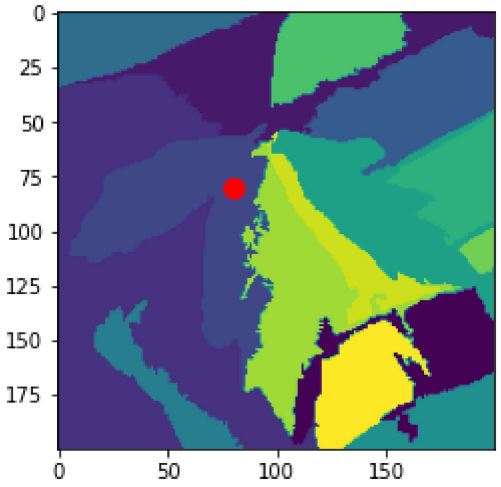


In [243]:
```
ShowImage([image_gray, result_label], 1, 3)
```

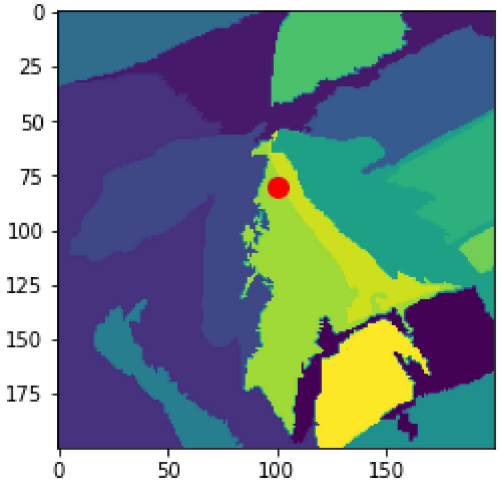


In [255]:
```
point = [80, 80]
idx = result_label[point[0], point[1]]
print("Index at ({0},{1}) : {2}".format(point[0], point[1], idx))
plt.imshow(result_label)
plt.plot(point[1], point[0], "or", markersize=10)  # og:shorthand for green circle
plt.show()
SegMask1 = result_label == idx

point = [80, 100]
idx = result_label[point[0], point[1]]
print("Index at ({0},{1}) : {2}".format(point[0], point[1], idx))
plt.imshow(result_label)
plt.plot(point[1], point[0], "or", markersize=10)  # og:shorthand for green circle
plt.show()
SegMask2 = result_label == idx

point = [80, 120]
idx = result_label[point[0], point[1]]
print("Index at ({0},{1}) : {2}".format(point[0], point[1], idx))
plt.imshow(result_label)
plt.plot(point[1], point[0], "or", markersize=10)  # og:shorthand for green circle
plt.show()
SegMask3 = result_label == idx
```

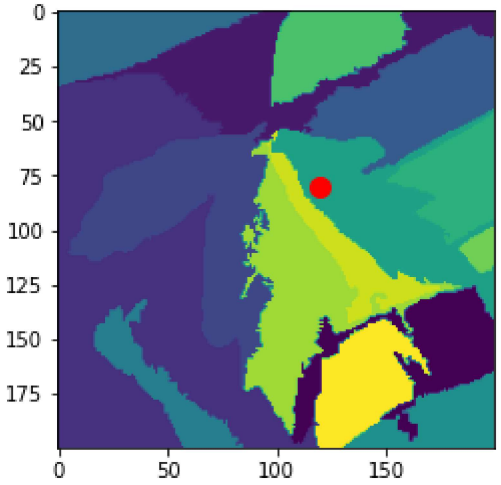
```
Index at (80,80) : 3
```




```
Index at (80,100) : 12
```




```
Index at (80,120) : 8
```


0
25
50
75
100
125
150
175
0
50
100
150

In [256]:
```
result_segment = SegMask1 + SegMask2 + SegMask3
ShowImage([img, result_segment], 1, 3)

kernel = np.ones((5,5),np.uint8)
result_segment = cv2.morphologyEx(np.float32(result_segment), cv2.MORPH_OPEN, kernel)
ShowImage([img, result_segment], 1, 3)
```

Image 0

Image 1

Image 0

Image 1