[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):
```

✓ 1s completed at 3:14 PM

```
[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]  NameDB
```

```
['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',
```

✓ 1s completed at 3:14 PM

```
[8]   'Crack.jpg',
      'Code.jpg',
      'Dust.jpg',
      'Barcode.png',
      'QR.jpg',
      'Leaf.jpg',
      'Cloths.jpg',
      'Writing.png',
      'Defect.jpg']
```

```
[9]  def SegmentColorImageByMask(IM, Mask):
        Mask = Mask.astype(np.uint8)
        result = cv2.bitwise_and(IM, IM, mask = Mask)
        return result
     def SegmentationByOtsu(image, mask):
        image_process = image.copy()
        image_mask = mask.copy()
        image_process[image_mask == 0] = 0
        ListPixel = image_process.ravel()
        ListPixel = ListPixel[ListPixel > 0]
        from skimage.filters import threshold_otsu
        otsu_thresh = threshold_otsu(ListPixel)
        return otsu_thresh
```

```
[10] def FillHoles(Mask):
        Result = ndi.binary_fill_holes(Mask)
        return Result
     def morphology(Mask, Size):
        from skimage.morphology import erosion, dilation, opening, closing, white_tophat
        from skimage.morphology import disk
        selem = disk(abs(Size))
        if(Size > 0):
           result = dilation(Mask, selem)
        else:
           result = erosion(Mask, selem)
        return result
```

```
[11] 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):
        if(len(img.shape) == 3):
           vectorized = img.reshape((-1,3))
        else:
           vectorized = IM.reshape(-1)
        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)
```

```
[11]    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
```

```
[12] 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
```

```
[12]    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
```

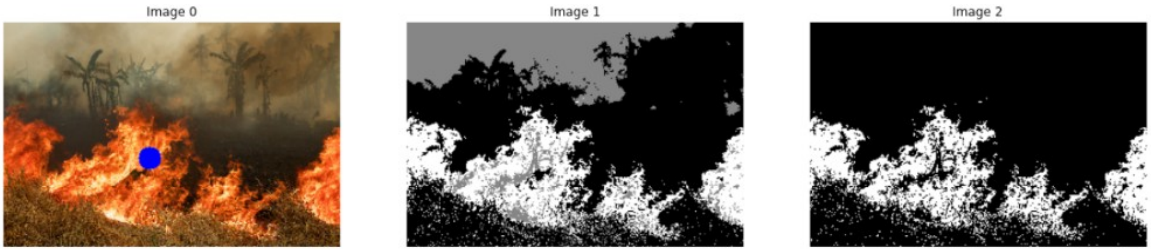
```
[13] import numpy as np
     from skimage.morphology import watershed
     from skimage.feature import peak_local_max
     import matplotlib.pyplot as plt
     from scipy import ndimage
```

```
[14] x, y = np.indices((80, 80))
     x1, y1, x2, y2 = 28, 28, 44, 52
     r1, r2 = 16, 20
     mask_circle1 = (x - x1) ** 2 + (y - y1) ** 2 < r1 ** 2
     mask_circle2 = (x - x2) ** 2 + (y - y2) ** 2 < r2 ** 2
     image = np.logical_or(mask_circle1, mask_circle2)
     distance = ndimage.distance_transform_edt(image)
     local_maxi = peak_local_max(
        distance, indices=False, footprint=np.ones((3, 3)), labels=image)
     markers = ndimage.label(local_maxi)[0]
     labels = watershed(-distance, markers, mask=image)

     plt.figure(figsize=(9, 3.5))
     plt.subplot(131)
     plt.imshow(image, cmap='gray', interpolation='nearest')
     plt.axis('off')
     plt.subplot(132)
     plt.imshow(-distance, interpolation='nearest')
     plt.axis('off')
     plt.subplot(133)
     plt.imshow(labels, cmap='nipy_spectral', interpolation='nearest')
     plt.axis('off')
```

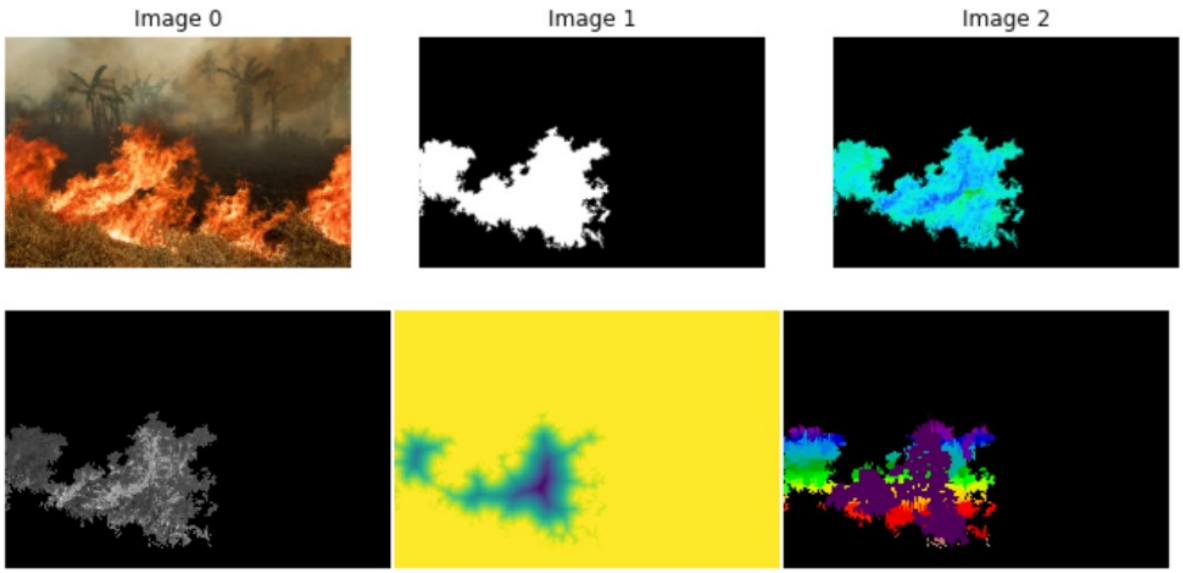
```
[14] /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: FutureWarning: indices argument is deprecated and will be removed in version 0.20.
       if __name__ == '__main__':
     /usr/local/lib/python3.7/dist-packages/skimage/morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated and will be r
       def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
     /usr/local/lib/python3.7/dist-packages/matplotlib/image.py:455: RuntimeWarning: overflow encountered in double_scalars
       newmin = vmid - dv * fact
     /usr/local/lib/python3.7/dist-packages/matplotlib/image.py:460: RuntimeWarning: overflow encountered in double_scalars
       newmax = vmid + dv * fact
     /usr/local/lib/python3.7/dist-packages/matplotlib/image.py:488: RuntimeWarning: invalid value encountered in multiply
       A_resampled *= ((a_max - a_min) / 0.8)
```

```
FileName = 'Fire.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)
```


```
[16] center, result_label, result_image = KmeansSegmentation(img, K = 3)
     result_label = ReArrangeIndex(result_label)

     rpoint = 120
     cpoint = 130
     img_select = img.copy()
     cv2.circle(img_select,(cpoint,rpoint), 10, (0,0,255), -1)

     idx = result_label[rpoint, cpoint]
     orig_mask = result_label == idx
     fill_mask = FillHoles(orig_mask)
     max_mask = SelectLargestRegion(fill_mask)
     image_max_mask = SegmentColorImageByMask(img, max_mask)

     ShowImage([img_select, result_label, orig_mask, fill_mask, max_mask, image_max_mask], 2, 3)
```




```
image_to_process = SegmentColorImageByMask(image_hsv, max_mask)
mask_to_process = max_mask.copy()
ShowImage([img, mask_to_process, image_to_process], 1, 5)
image = image_to_process[:,:,0].copy()
distance = ndimage.distance_transform_edt(image)
local_maxi = peak_local_max(distance, indices=False, footprint=np.ones((3, 3)),
labels=image)
markers = ndimage.label(local_maxi)[0]
labels = watershed(-distance, markers, mask=image)
plt.figure(figsize=(9, 3.5))
plt.subplot(131)
plt.imshow(image, cmap='gray', interpolation='nearest')
plt.axis('off')
plt.subplot(132)
plt.imshow(-distance, interpolation='nearest')
plt.axis('off')
plt.subplot(133)
plt.imshow(labels, cmap='nipy_spectral', interpolation='nearest')
plt.axis('off')
plt.subplots_adjust(hspace=0.01, wspace=0.01, top=1, bottom=0, left=0,right=1)
plt.show()
```




```
[18] from scipy import ndimage as ndi
     import matplotlib.pyplot as plt
     from skimage.morphology import watershed, disk
     from skimage import data
     from skimage.filters import rank
     from skimage.util import img_as_ubyte

     image = img_as_ubyte(data.camera())
     denoised = rank.median(image, disk(2))
     markers = rank.gradient(denoised, disk(5)) < 10
     markers = ndi.label(markers)[0]
     gradient = rank.gradient(denoised, disk(2))
     labels = watershed(gradient, markers)
     fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8),
     sharex=True, sharey=True)
     ax = axes.ravel()
     ax[0].imshow(image, cmap=plt.cm.gray, interpolation='nearest')
     ax[0].set_title("Original")
     ax[1].imshow(gradient, cmap=plt.cm.nipy_spectral, interpolation='nearest')
     ax[1].set_title("Local Gradient")
     ax[2].imshow(markers, cmap=plt.cm.nipy_spectral, interpolation='nearest')
```

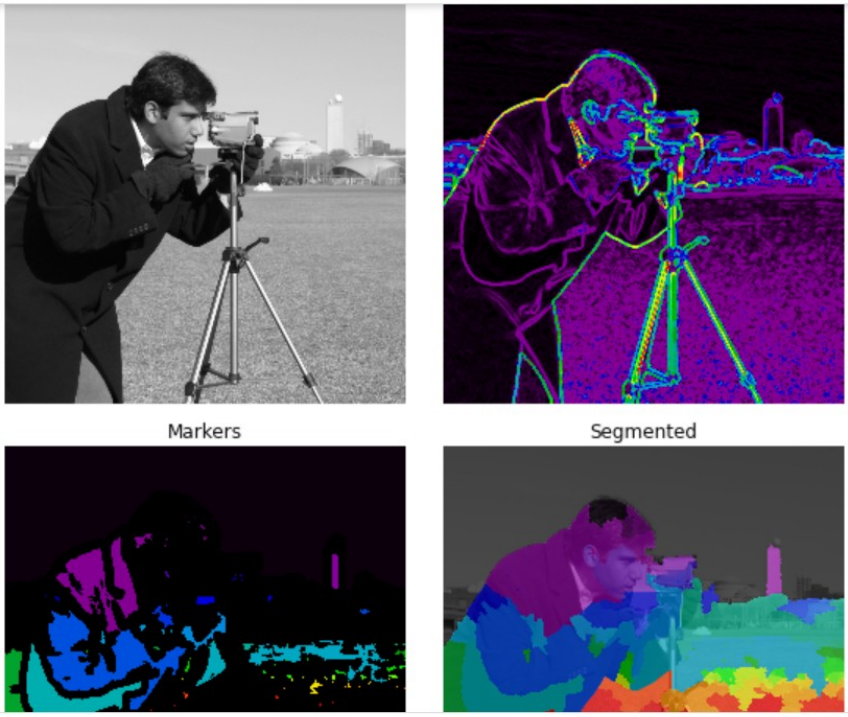
✓ 2s completed at 3:31 PM

```
ax = axes.ravel()
ax[0].imshow(image, cmap=plt.cm.gray, interpolation='nearest')
ax[0].set_title("Original")
ax[1].imshow(gradient, cmap=plt.cm.nipy_spectral, interpolation='nearest')
ax[1].set_title("Local Gradient")
ax[2].imshow(markers, cmap=plt.cm.nipy_spectral, interpolation='nearest')
ax[2].set_title("Markers")
ax[3].imshow(image, cmap=plt.cm.gray, interpolation='nearest')
ax[3].imshow(labels, cmap=plt.cm.nipy_spectral, interpolation='nearest', alpha
=.7)
ax[3].set_title("Segmented")
for a in ax:
   a.axis('off')
fig.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/skimage/morphology/_deprecated.py:5: skimage_deprecation: Function ``watershed`` is deprecated and will be remov
  def watershed(image, markers=None, connectivity=1, offset=None, mask=None,
```

Original Local Gradient

[18]

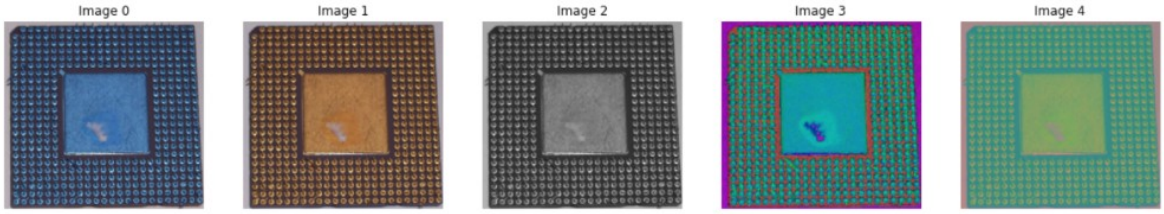



✓ 2s completed at 3:31 PM

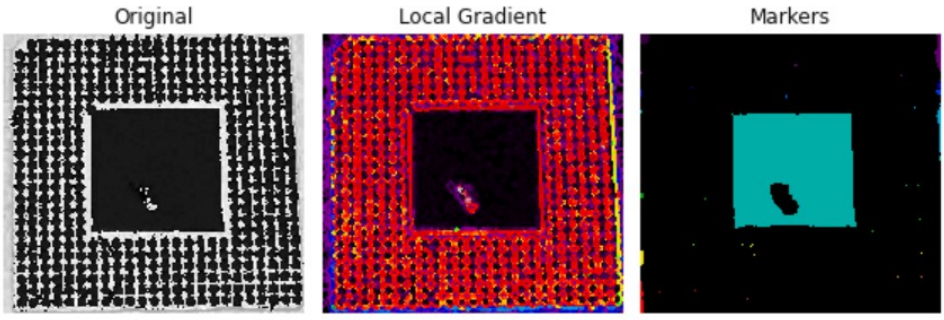
✓ 5s

```
[19] FileName = 'Dust.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)
```

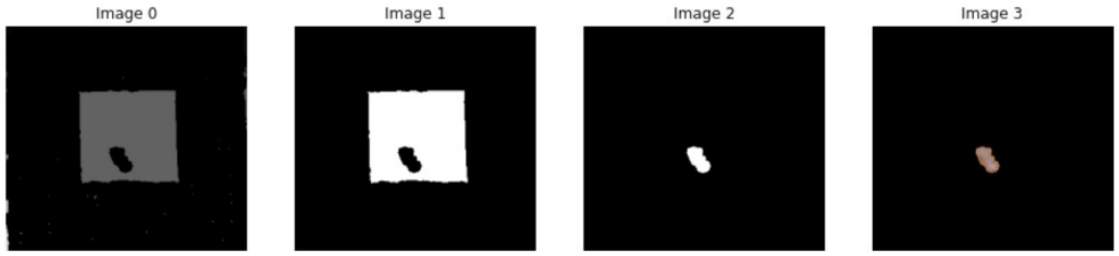
```
Selected Image :
Index  18
Name  Dust.jpg
```




```
[20] image = image_hsv[:,:,0].copy()
     denoised = rank.median(image, disk(2))
     markers = rank.gradient(denoised, disk(5)) < 10
     markers = ndi.label(markers)[0]
     gradient = rank.gradient(denoised, disk(2))
     fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(8, 8),
     sharex=True, sharey=True)
     ax = axes.ravel()
     ax[0].imshow(image, cmap=plt.cm.gray, interpolation='nearest')
     ax[0].set_title("Original")
     ax[1].imshow(gradient, cmap=plt.cm.nipy_spectral, interpolation='nearest')
     ax[1].set_title("Local Gradient")
     ax[2].imshow(markers, cmap=plt.cm.nipy_spectral, interpolation='nearest')
     ax[2].set_title("Markers")
     for a in ax:
        a.axis('off')
     fig.tight_layout()
     plt.show()
```

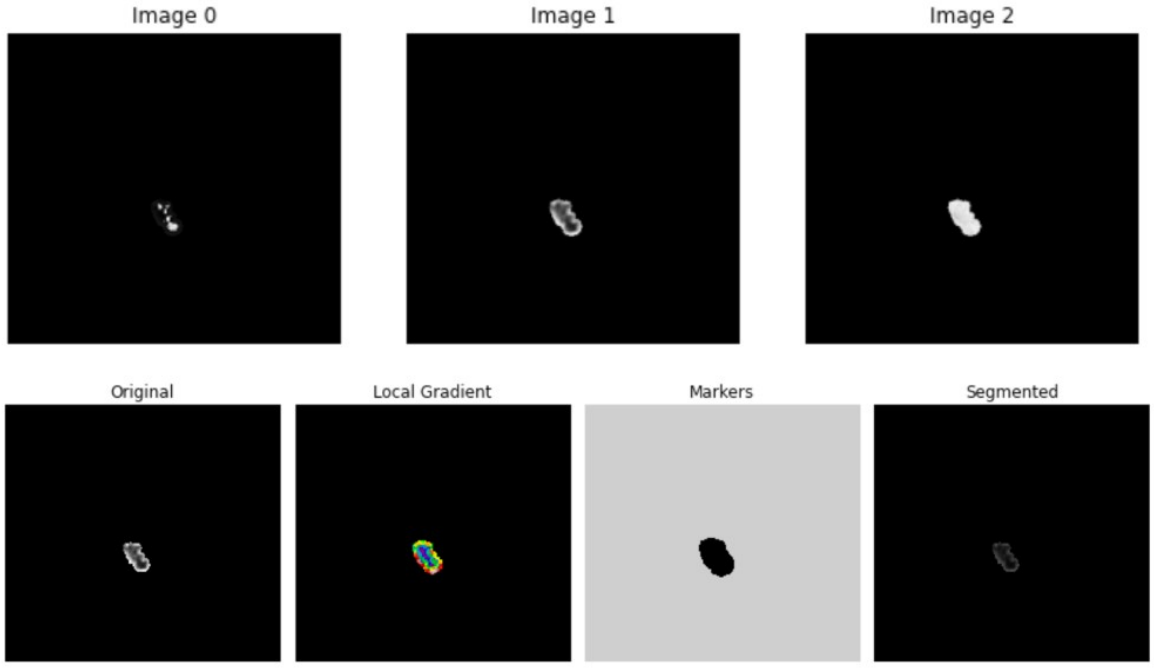



```
[21] max_area = 0
     ilabel = 0
     for i in range(1,markers.max()): # 0 is background not care in range
        area = (markers == i).sum()
        if(area > max_area):
         max_area = area
         ilabel = i
     max_mask = (markers == ilabel).astype(int)
     mask_to_process = FillHoles(max_mask) - max_mask
     image_to_process = SegmentColorImageByMask(img, mask_to_process)
     ShowImage([markers, max_mask, mask_to_process, image_to_process], 1, 5)
```




```
[22] image_to_process = SegmentColorImageByMask(image_hsv, mask_to_process)
     ShowImage([image_to_process[:,:,0], image_to_process[:,:,1], image_to_process
     [:,:,2]], 1, 5)
     image = image_to_process[:,:,1]

     denoised = rank.median(image, disk(2))
     markers = rank.gradient(denoised, disk(5)) < 10
     markers = ndi.label(markers)[0]
     gradient = rank.gradient(denoised, disk(2))
     labels = watershed(gradient, markers)
     fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(12, 12),
     sharex=True, sharey=True)
     ax = axes.ravel()
     ax[0].imshow(image, cmap=plt.cm.gray, interpolation='nearest')
     ax[0].set_title("Original")
     ax[1].imshow(gradient, cmap=plt.cm.nipy_spectral, interpolation='nearest')
     ax[1].set_title("Local Gradient")
     ax[2].imshow(markers, cmap=plt.cm.nipy_spectral, interpolation='nearest')
     ax[2].set_title("Markers")
     ax[3].imshow(image, cmap=plt.cm.gray, interpolation='nearest')
     ax[3].imshow(labels, cmap=plt.cm.nipy_spectral, interpolation='nearest', alpha
```

```
    ax[2].imshow(markers, cmap=plt.cm.nipy_spectral, interpolation='nearest')
    ax[2].set_title("Markers")
    ax[3].imshow(image, cmap=plt.cm.gray, interpolation='nearest')
    ax[3].imshow(labels, cmap=plt.cm.nipy_spectral, interpolation='nearest', alpha
    =.7)
    ax[3].set_title("Segmented")
    for a in ax:
       a.axis('off')
    fig.tight_layout()
    plt.show()
```

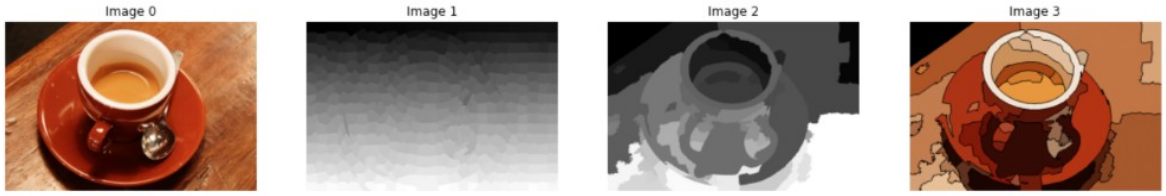


```
from skimage import data, io, segmentation, color
from skimage.future import graph
import numpy as np
def _weight_mean_color(graph, src, dst, n):
    """Callback to handle merging nodes by recomputing mean color.
    The method expects that the mean color of `dst` is already computed.
    Parameters
----------
    graph : RAG
    The graph under consideration.
    src, dst : int
    The vertices in `graph` to be merged.
    n : int
    A neighbor of `src` or `dst` or both.
    Returns
-------
    data : dict
    A dictionary with the `"weight"` attribute set as the absolute
    difference of the mean color between node `dst` and `n`.
```

✓ 2s completed at 3:31 PM

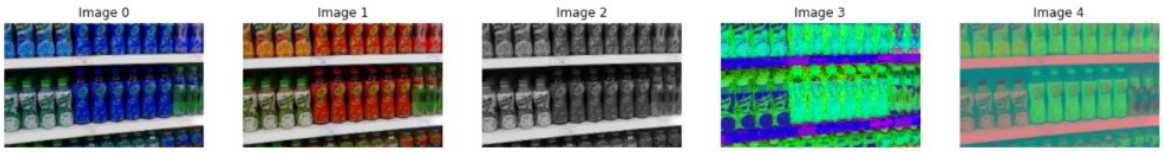
```
    data : dict
    A dictionary with the `"weight"` attribute set as the absolute
    difference of the mean color between node `dst` and `n`.
    """
    diff = graph.nodes[dst]['mean color'] - graph.nodes[n]['mean color']
    diff = np.linalg.norm(diff)
    return {'weight': diff}
def merge_mean_color(graph, src, dst):
    """Callback called before merging two nodes of a mean color distance graph.
    This method computes the mean color of `dst`.
    Parameters
----------
    graph : RAG
    The graph under consideration.
    src, dst : int
    The vertices in `graph` to be merged.
    """
    graph.nodes[dst]['total color'] += graph.nodes[src]['total color']
    graph.nodes[dst]['pixel count'] += graph.nodes[src]['pixel count']
    graph.nodes[dst]['mean color'] = (graph.nodes[dst]['total color'] /
    graph.nodes[dst]['pixel count'])
```

```
img = data.coffee()

orig_labels = segmentation.slic(img, compactness=30, n_segments=400)
g = graph.rag_mean_color(img, orig_labels)
merge_labels = graph.merge_hierarchical(orig_labels, g, thresh=35, rag_copy=False,
   in_place_merge=True,
   merge_func=merge_mean_color,
   weight_func=_weight_mean_color)
image_merge_labels = color.label2rgb(merge_labels, img, kind='avg', bg_label=0)
image_merge_labels = segmentation.mark_boundaries(image_merge_labels, merge_labels, (0, 0, 0))
ShowImage([img, orig_labels, merge_labels, image_merge_labels], 1, 4)
```

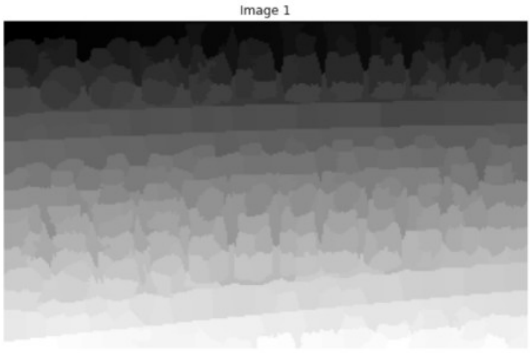


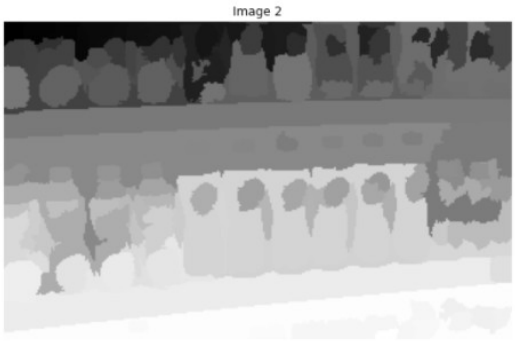

```
FileName = 'Shelf.jpg'
idx = NameDB.index(FileName)
print("Selected Image : ", "\nIndex ", idx, "\nName ", NameDB[idx])
image_orig = ImageDB[idx]
image_orig = ResizeImage(image_orig, 300, 0)
img = cv2.cvtColor(image_orig,cv2.COLOR_BGR2RGB)
image_gray = cv2.cvtColor(image_orig,cv2.COLOR_BGR2GRAY)
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)
```

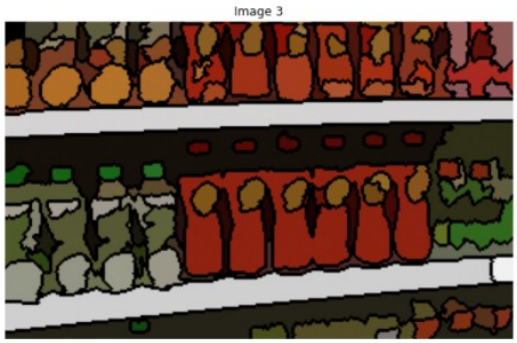
```
Selected Image :
Index  3
Name  Shelf.jpg
```



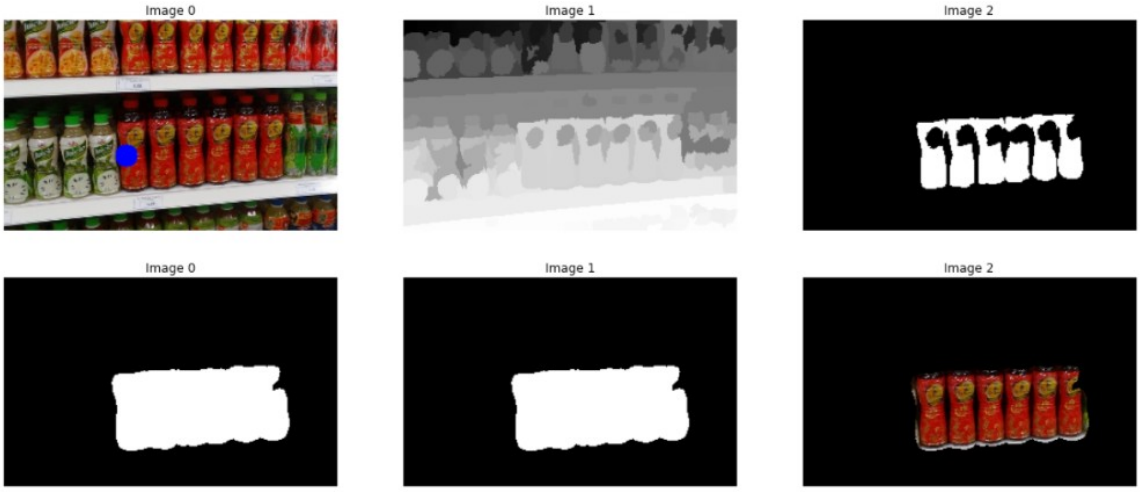

```
orig_labels = segmentation.slic(img, compactness=30, n_segments=400)
g = graph.rag_mean_color(img, orig_labels)
merge_labels = graph.merge_hierarchical(orig_labels, g, thresh=35, rag_copy=False,
   in_place_merge=True,
   merge_func=merge_mean_color,
   weight_func=_weight_mean_color)
image_merge_labels = color.label2rgb(merge_labels, img, kind='avg', bg_label=0)
image_merge_labels = segmentation.mark_boundaries(image_merge_labels, merge_labels, (0, 0, 0))
ShowImage([img, orig_labels, merge_labels, image_merge_labels], 2, 2)
```

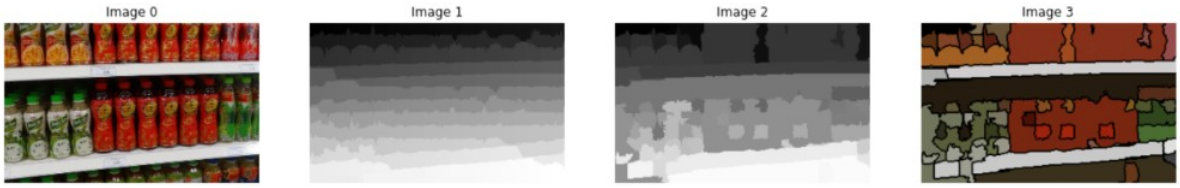




```
rpoint = 120
cpoint = 110
img_select = img.copy()
result_label = merge_labels.copy()
cv2.circle(img_select,(cpoint,rpoint), 10, (0,0,255), -1)
idx = result_label[rpoint, cpoint]
orig_mask = result_label == idx
dilation_mask = morphology(orig_mask, 5)
fill_mask = FillHoles(dilation_mask)
max_mask = SelectLargestRegion(fill_mask)
image_max_mask = SegmentColorImageByMask(img, max_mask)
ShowImage([img_select, result_label, orig_mask], 1, 3)
ShowImage([fill_mask, max_mask, image_max_mask], 1, 3)
```




```
from skimage.filters import sobel
gradient = sobel(rgb2gray(img))
orig_labels = segmentation.watershed(gradient, markers=250, compactness=0.01)
g = graph.rag_mean_color(img, orig_labels)
merge_labels = graph.merge_hierarchical(orig_labels, g, thresh=35, rag_copy=False,
   in_place_merge=True,
   merge_func=merge_mean_color,
   weight_func=_weight_mean_color)
image_merge_labels = color.label2rgb(merge_labels, img, kind='avg', bg_label=0)
image_merge_labels = segmentation.mark_boundaries(image_merge_labels, merge_labels, (0, 0, 0))
ShowImage([img, orig_labels, merge_labels, image_merge_labels], 1, 4)
```



```
rpoint = 120
cpoint = 110
img_select = img.copy()
result_label = merge_labels.copy()
cv2.circle(img_select,(cpoint,rpoint), 10, (0,0,255), -1)
idx = result_label[rpoint, cpoint]
orig_mask = result_label == idx
dilation_mask = morphology(orig_mask, 5)
fill_mask = FillHoles(dilation_mask)
max_mask = SelectLargestRegion(fill_mask)
image_max_mask = SegmentColorImageByMask(img, max_mask)
ShowImage([img_select, result_label, orig_mask], 1, 3)
ShowImage([fill_mask, max_mask, image_max_mask], 1, 3)
```