```
from skimage.segmentation import mark_boundaries, slic
from skimage.exposure import rescale_intensity
from skimage import color
from skimage.measure import regionprops
from scipy.spatial.distance import cdist
from sklearn.metrics import silhouette_score
from skimage.util import img_as_float
from skimage import io
import scipy.stats
import numpy as np
import argparse
import cv2
import math
import skimage.filters
import matplotlib.pyplot as plt
from scipy.integrate import quad
import copy


image = 'iiitd.jpg'
segments = 2000
orig = cv2.imread(image)
# print(orig)
print(orig.shape)

numPix = orig.shape[0]*orig.shape[1]
# print(numPix)

# scikit-image
slic_img = io.imread(image)
image = io.imread(image)

segments = slic(orig,start_label = 1, n_segments=segments, compactness=20, sigma=3)
# vis = np.zeros(segments.shape, dtype="float")
regions = regionprops(segments,intensity_image=image)
print(segments.shape)
color_mean =[]
for r in regions:
    # paint_region_with_avg_intensity(r.coords, r.mean_intensity, i)
    rp = r.coords
    mi = r.mean_intensity
    color_mean.append(mi)
    for i in range(rp.shape[0]):
        slic_img[rp[i][0],rp[i][1],0:3] = mi
plt.imshow(slic_img)
plt.show()
```

```
(470, 870, 3)
(470, 870)
```


```
len(color_mean[0])
```

```
3
```

```
color_mean_x = color.label2rgb(segments, slic_img, kind='avg')
```

```
<ipython-input-4-e95762849a18>:1: FutureWarning: The new recommended value for bg_label
  color_mean_x = color.label2rgb(segments, slic_img, kind='avg')
```

```
# color_mean_x[1,1]


color_mean = {}

for i in range(segments.shape[0]):
    for j in range(segments.shape[1]):
#         print(segments[i,j])
        color_mean[segments[i,j]] = color_mean_x[i,j]
    if len(color_mean)==len(np.unique(segments)):
        break
# color_mean[i] = color_mean_x[segments[i,j]]

# color_mean = np.unique(color_mean_x)

# for i in range(segments[1]):
#     for j in range(segments[0]):
#         color_mean[segments[i,j]] =


color_mean
```

```
{1: array([231., 234., 234.]),
 2: array([232., 234., 236.]),
 3: array([233., 235., 236.]),
 4: array([233., 235., 236.]),
 5: array([232., 235., 236.]),
 6: array([232., 236., 236.]),
 7: array([233., 235., 236.]),
 8: array([233., 235., 236.]),
 9: array([232., 235., 235.]),
```

```
 10: array([231., 236., 235.]),
 11: array([232., 236., 235.]),
 12: array([232., 236., 236.]),
 13: array([232., 236., 236.]),
 14: array([233., 237., 236.]),
 15: array([233., 237., 236.]),
 16: array([234., 238., 237.]),
 17: array([234., 238., 237.]),
 18: array([234., 239., 238.]),
 19: array([234., 240., 238.]),
 20: array([234., 239., 238.]),
 21: array([235., 239., 238.]),
 22: array([235., 239., 238.]),
 23: array([235., 240., 238.]),
 24: array([236., 240., 239.]),
 25: array([236., 240., 239.]),
 26: array([236., 240., 239.]),
 27: array([235., 239., 238.]),
 28: array([234., 239., 239.]),
 29: array([235., 241., 241.]),
 30: array([235., 241., 240.]),
 31: array([233., 239., 238.]),
 32: array([232., 239., 238.]),
 33: array([235., 241., 241.]),
 34: array([235., 241., 242.]),
 35: array([236., 241., 241.]),
 36: array([236., 241., 241.]),
 37: array([237., 241., 240.]),
 38: array([237., 241., 240.]),
 39: array([237., 243., 241.]),
 40: array([235., 241., 240.]),
 41: array([233., 240., 240.]),
 42: array([233., 239., 240.]),
 43: array([232., 238., 240.]),
 44: array([232., 238., 240.]),
 45: array([232., 239., 240.]),
 46: array([233., 239., 239.]),
 47: array([234., 239., 241.]),
 48: array([233., 239., 240.]),
 49: array([232., 239., 240.]),
 50: array([233., 239., 239.]),
 51: array([233., 239., 239.]),
 52: array([233., 239., 239.]),
 53: array([232., 238., 240.]),
 54: array([230., 236., 239.]),
 55: array([229., 235., 239.]),
 56: array([227., 234., 237.]),
 57: array([226., 232., 236.]),
 58: array([223., 231., 235.]),
 59: array([223., 231., 235.]),
```

```
(color_mean.values())
```

```
dict_values([array([231., 234., 234.]), array([232., 234., 236.]), array([233., 235., 23
```

```
center_orig = []
# color_mean = []
for props in regions:
    cx, cy = props.centroid
    center_orig.append([cx,cy])

# print(center_orig)


segments.shape
```

```
(470, 870)
```

```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
data = np.float32(list(color_mean.values()))/255

_, labels, (centers) = cv2.kmeans(data, 18, None, criteria, 200, cv2.KMEANS_RANDOM_CENTERS)
# centers = np.uint8(centers)
# labels.append(1)
pred_mat = np.squeeze(labels)
a = np.array([int(pred_mat.mean())])
# print(pred_mat)
pred_mat = np.concatenate((pred_mat,a), axis=0)
# print(pred_mat)
# print(centers)
# avgVal = pred_mat.mean()
# pred_mat = list(pred_mat)
# pred_mat.append(int(avgVal))
# print(pred_mat.shape)
# print(segments.shape)
distanceClusters = cdist(centers,centers, 'euclidean')
clustered_img = pred_mat[segments]
# print(clustered_img)
dictX = {}
for i in np.unique(clustered_img):
    dictX[i] = np.count_nonzero(clustered_img==i)


# print(segments[segments.shape[0]//2,segments.shape[1]//2])
imgCenter = list(center_orig[segments[segments.shape[0]//2,segments.shape[1]//2]])
centerdist = cdist(center_orig,[imgCenter],'euclidean')
distVar = np.std(centerdist)
# print(slic_center_dist_var)


# print(dictX)
# print(cluster_dist)

contrastCues = []
for i in range(18):
    wc =0
    # xk,yk = centers[i]
```

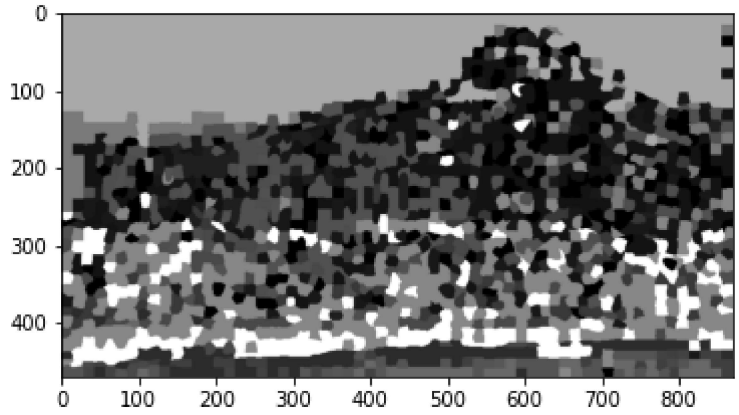
```
        # xi,yi = centers[j]
        for j in range(18):
            # xi,yi = centers[j]
            wc += distanceClusters[i,j]*(dictX[j]/(clustered_img.shape[0]*clustered_img.shape[1])
        contrastCues.append(wc)
    # print(contrastCues)
    # print(contrastCues)
    maxContrastCues = max(contrastCues)
    # print(contrastCues)
    print(maxContrastCues)
    contrastCues = np.divide(contrastCues, [maxContrastCues,])
    print(contrastCues)
    contrastCuesImage = np.zeros((clustered_img.shape[0],clustered_img.shape[1]))
    for i in range(clustered_img.shape[0]):
        for j in range(clustered_img.shape[1]):
            # print(clustered_img[i,j])
            contrastCuesImage[i,j] = contrastCues[int(clustered_img[i,j])]

    # print(contrastCuesImage)
```

```
0.705776259526125
[0.66788078 0.73628608 0.70543811 1.         0.63779162 0.64867717
 0.72086633 0.69906241 0.86384473 0.7313918  0.72609723 0.6298314
 0.68957292 0.76150235 0.59927684 0.71315255 0.78893789 0.81036215]
```

```
    print("Contrast Cue Image")
    plt.imshow(contrastCuesImage, cmap = 'gray')
    plt.show()
```

```
Contrast Cue Image
```

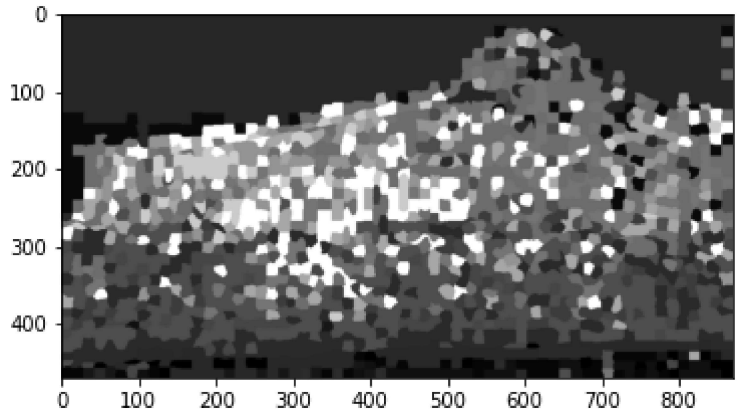



```
    spatial_cues = []
    for c in range(18):
        wc = 0
        for i in range(len(center_orig)):
            if(pred_mat[i] == c):
                wc+=scipy.stats.norm(0, distVar).pdf(centerdist[i,0])
        wc = wc/np.count_nonzero(pred_mat == c)
        spatial_cues.append(wc)
```

```
maxSpatialCues = max(spatial_cues)
spatial_cues = np.divide(spatial_cues, [maxSpatialCues,])
spatialCuesImage = np.zeros((clustered_img.shape[0],clustered_img.shape[1]))
for i in range(clustered_img.shape[0]):
    for j in range(clustered_img.shape[1]):
        # print(clustered_img[i,j])
        spatialCuesImage[i,j] = spatial_cues[int(clustered_img[i,j])]


print("Spatial Cue Image")
plt.imshow(spatialCuesImage, cmap = 'gray')
plt.show()
```

```
Spatial Cue Image
```




```
def generateMask(salMap):
    threshold = skimage.filters.threshold_otsu(salMap)
    print("threshold is: ",threshold)
    mask = copy.deepcopy(salMap)
#     print(mask.shape)
#     print(salMap.shape)
    for i in range(salMap.shape[0]):
        for j in range(salMap.shape[1]):
#             print([i,j])
            if salMap[i,j]>threshold:
                mask[i,j] = 1
            else:
                mask[i,j] = 0
    m,n = mask.shape
    r1,r2 = m//2 - int(0.15*m), m//2 + int(0.15*m)
    r3,r4 = n//2 - int(0.15*n), n//2 + int(0.15*n)
    c0 = 0
    c1 = 0
    for i in range(r1,r2):
        for j in range(r3,r4):
            if(mask[i,j] == 0):
                c0 += 1
            else:
                c1 += 1
```

```
    if(c0 > c1):
        return 1-mask, mask
    else:
        return mask, 1-mask

def getMu(mMap):
    return np.mean(mMap[mMap>0]), np.std(mMap[mMap>0])

from scipy.integrate import quad

def getD(z,sigma, mu):
    D = 1/(sigma*(2*math.pi)**(1/2)*np.exp(((z-mu)/sigma)))
    return D



def seperationScore(salMap):
    salMap = (salMap/np.max(salMap))*255
    foregroundMask, backgroundMask = generateMask(salMap)
#     print(backgroundMask)
#     print(foregroundMask)
    # print(np.max(foregroundMask))
    fMap = salMap*foregroundMask
    fMap = fMap/np.max(fMap)
#     print(fMap)
    bMap = salMap*backgroundMask
    bMap = bMap/np.max(bMap)
#     print(bMap)
    muF, sigmaF = getMu(fMap)
    muB, sigmaB = getMu(bMap)
    print("Mu foreground, sigma forground= ", muF,sigmaF)
    print("Mu background, sigma ackgound= ", muB,sigmaB)
    # fVal = fMap[fMap>0].flatten()
    # bVal = bMap[bMap>0].flatten()
    fDist = scipy.stats.norm(muF,sigmaF**2).pdf(fMap[fMap>0].flatten())
    bDist = scipy.stats.norm(muB, sigmaB**2).pdf(bMap[bMap>0].flatten())
    z = (muB*sigmaF**2 - muF*sigmaB**2)/(sigmaF**2 - sigmaB**2)
    z = z + (sigmaB*sigmaF)/(sigmaF**2 - sigmaB**2)
    z = z*((muF - muB)**2 - 2*(sigmaF**2 - sigmaB**2)*(math.log(sigmaB)- math.log(sigmaF))**(
    Ls =  quad(getD,0, z, args=(muF,sigmaF))[0] + quad(getD, z,1,args=(muB,sigmaB))[0]
    phi = 1/(1+ math.log(1+ (np.count_nonzero([fMap > 0]) + np.count_nonzero([bMap > 0]))*Ls,
    print("z is: ", z)
    print("Ls is: ", Ls)
    print("Phi is: ", phi)
    return phi



contrastScore = seperationScore(contrastCuesImage)
```

```
 threshold is:  194.12843001213838
 Mu foreground, sigma forground=  0.902199221515198 0.06289773124527578
```

```
Mu background, sigma ackgound=  0.8626389461196063 0.06637087307166828
z is:  -0.014269515287153551
Ls is:  0.29604680091738395
Phi is:  0.16439308102241007
```

```
spatialScore = seperationScore(spatialCuesImage)
```

```
threshold is:  120.53816874430629
Mu foreground, sigma forground=  0.7105905893886728 0.20677090247840735
Mu background, sigma ackgound=  0.5628837547327572 0.25138029714081633
z is:  -0.06076916383441587
Ls is:  0.5414614368847829
Phi is:  0.15759977798674102
```

```
finImage = np.zeros((clustered_img.shape[0],clustered_img.shape[1]))
for i in range(clustered_img.shape[0]):
    for j in range(clustered_img.shape[1]):
        finImage[i,j] = contrastScore*contrastCues[clustered_img[i,j]] + spatialScore*spatial
plt.imshow(final_saliency_image, cmap = 'gray')
plt.show()
```