CV ASSIGNMENT 2

Q1: [COLAB LINK TO CODE FILE]

- **Methodology**
  - Import libraries

```
import numpy as np
from fcmeans import FCM
from matplotlib import pyplot as plt
import cv2
```

  - Import image

```
img = cv2.imread("/content/drive/MyDrive/CV_Sample_Images/Cap2.PNG").astype(np.float32)
```

  - Hyperparameters for the code.

```
n_clusters = 50 # max number of clusters in fuzzy c-means algorithm
threshold = 20  # threshold to identify very small regions and stray pixels
window = 3      # window size to assign majority cluster to removed regions
```

  - Defined a function for displaying the results as an image given the cluster labels and cluster centers.

Dushyant Panchal, 2018033

```python
def image_from_cluster_labels(labels, centers, shape):
    """
      Function to display results as images
        given cluster labels and centers.

    Input:
        - labels: 2d array of integral cluster labels
        - centers: 2d array of features for the cluster centers
        - shape: (m,n,k) shape of the rgb image under consideration
            m - height
            n - width
            k - n_channels

    Output:
        - out_img: output image after applying the color given by
            'centers' to the respective pixels as per the 'labels'.
            out_img is of the shape (m,n,k) = 'shape'
    """

    out_img = np.zeros(shape)

    m = shape[0]
    n = shape[1]

    for i in range(m):
        for j in range(n):
            label = labels[i,j]
            if label==-1:
                continue
            col = centers[label]
            out_img[i,j,0] = col[0]
            out_img[i,j,1] = col[1]
            out_img[i,j,2] = col[2]

    out_img*=255
    return out_img
```

- Performed fuzzy c-means clustering with max ‘n_clusters’ clusters.

```
# feature extraction for clustering
m,n,_ = img.shape
X = []

for i in range(m):
  for j in range(n):
    X.append([ img[i,j,0]/255, img[i,j,1]/255, img[i,j,2]/255, i/m, j/n ])
X = np.array(X)


# perform fuzzy c-means clustering
fcm = FCM(n_clusters)
fcm.fit(X)


# obtain the labels from the clustering
fcm_labels = np.array(fcm.predict(X)).reshape(m,n)
fcm_centers = np.array(fcm.centers)
```

- Obtained objects as the disconnected components in every cluster and set the label to -1 for components consisting of less than ‘threshold’ pixels.

```
# Removing stray/isolated pixels and very-small-regions

import skimage.measure

cluster_map = fcm_labels.copy()

sk_labels = skimage.measure.label(cluster_map).reshape(m,n)
n_objs = np.max(sk_labels) + 1

for j in range(n_objs):
  obj = (sk_labels==j)
  if np.sum(obj) < threshold:
    cluster_map[obj] = -1

cv2_imshow(image_from_cluster_labels(cluster_map, fcm_centers, img.shape))
```

- Iterated over the cluster labels, if a -1 was found replaced it with the most frequently occurring non-negative label in a (‘window’ x ‘window’) sized sliding window.

```
# Assigning the surrounding pixels to the removed regions

for i in range(m-window+1):
  for j in range(n-window+1):
    clusters_around = cluster_map[i:i+window,j:j+window]
    minus_1s = (cluster_map[i:i+window,j:j+window]<0)

    if np.sum(minus_1s)>0:
      clusters_around = clusters_around[clusters_around>=0]

      clusters, counts = np.unique(clusters_around, return_counts=True)
      majority_ind = np.argmax(counts)
      majority_cluster = clusters[majority_ind]

      cluster_map[i:i+window,j:j+window] += (minus_1s)*(1+majority_cluster)
```

- **Results:**
  - **n_clusters** = 50;
  - **threshold** = 20;
  - **window** = 3
  - The top-left image shows the original input used.
  - The top-right image shows the result after clustering and assigning the color of the cluster center to the pixels. 50 clusters were created, thus 50-color encoding of the entire image.
  - The bottom-left image shows black regions for the small objects that have been removed by the method discussed above through thresholding over disconnected component sizes. Components of a cluster having <20 pixels were removed.
  - The bottom-right image shows the final result, where the removed objects were assigned to the most frequently occurring cluster label within a 3x3 window of a removed pixel.
  - (Find the images below.)

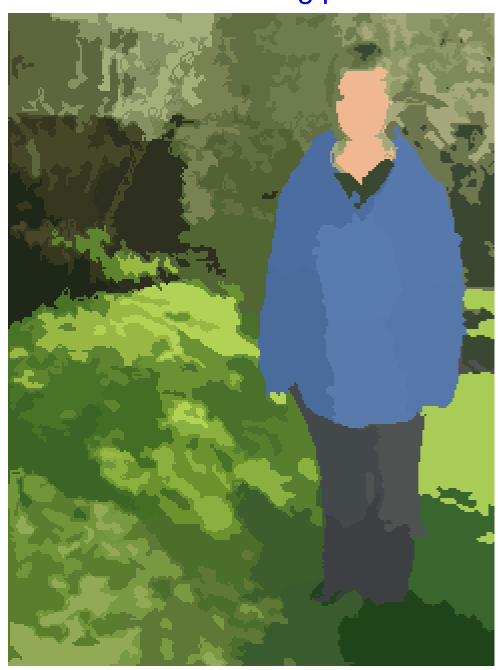
CV ASSIGNMENT 2

Input Image

Clustered Image

Removed very small regions.

Fixed with surrounding pixel values.



Dushyant Panchal, 2018033

CV ASSIGNMENT 2

Q2: [COLAB LINK TO CODE FILE]

- **Methodology**
  - Import libraries

```
import cv2
import numpy as np
from sklearn.cluster import KMeans
```

  - Import image

```
image = cv2.imread("/content/drive/MyDrive/CV_Sample_Images/bird.jpg")
```

  - Hyperparameters for the code.

```
slic_region_size = 16 # slic region size
K = 20                # n_clusters for k-means
```

  - Obtain SLIC super-pixels

```
# making a copy
img = image.copy()

# SLIC initialization
slic = cv2.ximgproc.createSuperpixelSLIC(img, algorithm = cv2.ximgproc.SLICO, region_size=slic_region_size)
slic.iterate()

# obtaining super-pixels generated by the slic algorithm
num_super_pixels = slic.getNumberOfSuperpixels()
slic_centers_rgb = np.zeros((num_super_pixels, 3))
slic_centers_xy = np.zeros((num_super_pixels, 2))
slic_sizes = np.zeros(num_super_pixels)
slic_labels = slic.getLabels()

# calculating cluster centers
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        sp_id = slic_labels[i,j]
        slic_centers_rgb[sp_id,:] += np.array([ img[i,j,0]/255, img[i,j,1]/255, img[i,j,2]/255 ])
        slic_centers_xy[sp_id,:] += np.array([ i/img.shape[0], j/img.shape[1] ])
        slic_sizes[sp_id] += 1

for i in range(num_super_pixels):
    slic_centers_rgb[i] /= slic_sizes[i]
    slic_centers_xy[i] /= slic_sizes[i]

print(num_super_pixels,"Super-Pixels Generated.")
```

```
248 Super-Pixels Generated.
```

  - K-means clustering on SLIC super-pixels

```
# clustering the super pixels on rgb data of the slic centers

kmeans = KMeans(n_clusters=K).fit(slic_centers_rgb)

cluster_centers = kmeans.cluster_centers_
cluster_labels = kmeans.labels_
_, nK = np.unique(kmeans.labels_, return_counts=True)
```

- Calculating contrast cues

```
# calculating contrast cues for the clusters so formed

contrast_cue = np.zeros(K)

for k in range(K):
  for i in range(K):
    dist = np.sum( (cluster_centers[i]-cluster_centers[k])**2 )**0.5
    contrast_cue[k] += (nK[i]/num_super_pixels)*(dist)

contrast_cue /= np.max(contrast_cue)
contrast_cue
```

```
array([0.25241475, 1.        , 0.20555029, 0.17806726, 0.29076443,
       0.34202066, 0.74970921, 0.2423622 , 0.28765837, 0.18578357,
       0.20337587, 0.8397911 , 0.19032879, 0.34802074, 0.17734195,
       0.21108398, 0.31572393, 0.2081466 , 0.91965513, 0.18590261])
```

- Calculating spatial cues

```
# calculating spatial cues for the clusters so formed
spatial_cue = np.zeros(K)
img_center = np.mean(slic_centers_xy, axis=0)

for k in range(K):
  for i in range(num_super_pixels):
    if cluster_labels[i]==k:
      dist = np.sum( (slic_centers_xy[i] - img_center )**2 )**0.5
      spatial_cue[k] += ( np.exp(-dist) )/nK[k]

spatial_cue /= np.max(spatial_cue)
spatial_cue
```

```
array([0.6868269 , 0.94936634, 0.75904907, 0.83964138, 0.68575342,
       0.72163229, 0.8926482 , 0.7664883 , 0.75715903, 0.82111553,
       0.63060696, 0.99444975, 0.76010261, 0.66280089, 0.82854681,
       0.65981073, 0.77499851, 0.82064969, 1.        , 0.66766058])
```

- **Results:**
  - `slic_region_size = 16;`
  - `K = 20;`
  - Contrast cue and Spatial cue Saliency maps respectively

Q3: [COLAB LINK TO CODE FILE] (continued in the same code as Q2)

Methodology

- Define function to generate gaussian values given z, mean and std.

```
from math import log, exp, pi, log10
```

```
def gaussian(z, mu, sig):
  return exp( - ((z-mu)/sig)**2 )/(sig*(2*pi)**0.5)
```

- Define function to calculate separation measure as in the paper.

```
def separation_measure(sal, gamma=1000):
```

```
    # obtain otsu threshold
    img = (sal*255).astype('uint8')
    th, ret = cv2.threshold(img, 0,255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    print("Threshold:", th)
```

```
    # separate fg and bg saliency
    fg = sal[img>th]
    bg = sal[img<=th]

    # fit a gaussian on fg saliency values
    mu_f = np.mean(fg)
    sig_f = np.std(fg)
    var_f = sig_f**2

    # fit a gaussian on bg saliency values
    mu_b = np.mean(bg)
    sig_b = np.std(bg)
    var_b = sig_b**2
```

```
    # calculate point of overlap z*
    z_star_part1 = (mu_b*var_f - mu_f*var_b)/(var_f - var_b)
    z_star_part2 = (sig_f*sig_b/(var_f-var_b)) * ( (mu_f-mu_b)**2 - 2*(var_f-var_b)*(log(sig_b) - log(sig_f)) )**0.5
    z_star1, z_star2 = z_star_part1+z_star_part2, z_star_part1-z_star_part2
```

```python
# Calculate final phi values, i.e. the separation measure
phi_1, phi_2 = 0,0

if 0<=z_star1<=1:
  Ls = 0
  for b in range(gamma):
    z = b/gamma
    if z <= z_star1:
      Ls += gaussian(z, mu_f, sig_f)/gamma
    else:
      Ls += gaussian(z, mu_b, sig_b)/gamma
  phi_1 = 1/( 1+log10( 1 + gamma*Ls ) )

if 0<=z_star2<=1:
  Ls = 0
  for b in range(gamma):
    z = b/gamma
    if z <= z_star2:
      Ls += gaussian(z, mu_f, sig_f)/gamma
    else:
      Ls += gaussian(z, mu_b, sig_b)/gamma
  phi_2 = 1/( 1+log10( 1 + gamma*Ls ) )

return phi_1, phi_2
```

- Obtain separation measure for contrast cue based saliency

```python
sm_cont = separation_measure(sal = contrast_out_img)
sm_cont
```

```
Threshold: 88.0
z* values: 0.48516736857613596 -0.7762101171884834
(0.9999997009324579, 0)
```

- Obtain separation measure for spatial cue based saliency

```python
sm_spat = separation_measure(sal = spatial_out_img)
sm_spat
```

```
Threshold: 197.0
z* values: 0.7895977724262151 -0.5869977218926836
(0.3664519214818544, 0)
```

CV ASSIGNMENT 2

Results:

- `slic_region_size = 16;`
- `K = 20;`

Input Image

Contrast Cue Saliency

Separation Measure:
ϕ(S)=`0.9999997009324579`

Spatial Cue Saliency

Separation Measure:
ϕ(S)=`0.3664519214818544`

Weighted sum saliency: taking the separation measure quality scores as the weights for the individual saliency maps, summing the two up into a single map. Find below the final combined saliency map.

```
weighted_sum = contrast_out_img*0.9999997009324579
  +spatial_out_img*0.3664519214818544
weighted_sum /= np.max(weighted_sum)
cv2_imshow(weighted_sum*255)
```

References:

- https://towardsdatascience.com/fuzzy-c-means-clustering-is-it-better-than-k-means-clustering-448a0aba1ee7
- https://scikit-image.org/docs/stable/api/skimage.measure.html#skimage.measure.label
- http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html