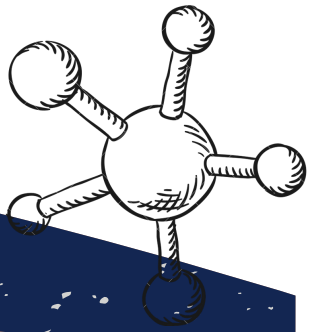
Image Segmentation

2021

Prepared by :
Hossam Elkady 5446
Naira Yasser 5299
Logyn Medhat 5386

Content Table



Unzipping the file :

```
! unzip BSR.zip
```

Imports :

```
[ ] import numpy as np
    import sklearn
    import matplotlib.pyplot as plt
    import matplotlib.image as mpimg
    import scipy.io
    import os
    import glob
    import cv2
    import matplotlib.image as mpimg
    import scipy.io as sio
    from sklearn.cluster import KMeans
    from sklearn.cluster import SpectralClustering
    from PIL import Image
    from sklearn.metrics.cluster import contingency_matrix
    import sklearn.metrics as metrics
    from scipy.spatial import distance_matrix
    from sklearn.neighbors import kneighbors_graph
```

Reading Images :

```
OrginalImagesRoot = 'BSR/BSDS500/data/images/1/'
GroundTruthRoot = 'BSR/BSDS500/data/groundTruth/1/'
Orginal_Test_Images=[]
Ground_Images=[]
Test_Images=[]
i=0
Totalfmeasures=np.zeros((50,5))
Totalentropy=np.zeros((50,5))

for img in glob.glob(f'{OrginalImagesRoot}/*.jpg'):

  Test_Img =cv2.imread(img)
  Test_Img=cv2.cvtColor(Test_Img, cv2.COLOR_BGR2RGB)
  Orginal_Test_Images.append(Test_Img)
  gname=os.path.splitext(os.path.basename(img))[0]
  i += 1
  Ground_Img = sio.loadmat(f'{GroundTruthRoot}{gname}')
  for i in range(0,5):
    Ground_Images.append(Ground_Img['groundTruth'][0][i][0][0][0])



Ground_Images=np.array(Ground_Images)
Orginal_Test_Images=np.array(Orginal_Test_Images)
for i in range(0,50):
  x=Orginal_Test_Images[i].reshape((-1,3))
  x=np.float32(x)
  Test_Images.append(x)
Test_Images=np.array(Test_Images)
```

Displaying Ground Truth:

```
def Display_Img_GroundTruth(TestIMG, GroundIMG, index):
    f, axes = plt.subplots(1,6,figsize=(20,5))
    r,c=2,3
    axes[0].imshow(TestIMG[index],aspect='auto')
    axes[0].axis('off')
    idx = index * 5

    axes[1].imshow(GroundIMG[idx] , aspect='auto')
    axes[1].axis('off')
    axes[2].imshow(GroundIMG[idx+1],aspect='auto')
    axes[2].axis('off')
    axes[3].imshow(GroundIMG[idx+2],aspect='auto')
    axes[3].axis('off')
    axes[4].imshow(GroundIMG[idx+3],aspect='auto')
    axes[4].axis('off')
    axes[5].imshow(GroundIMG[idx+4],aspect='auto')
    axes[5].axis('off')
    plt.subplots_adjust(wspace=0.02,hspace=0.02)
    plt.show()
```

```
def Display_Img_GroundTruth_Contour(TestIMG, GroundIMG, index):


    OrginalImagesRoot = 'BSR/BSDS500/data/images/1/'
    GroundTruthRoot = 'BSR/BSDS500/data/groundTruth/1/'
    img='BSR/BSDS500/data/images/1/2018.jpg'

    Test_Img =cv2.imread(img)
    Test_Img=cv2.cvtColor(Test_Img, cv2.COLOR_BGR2RGB)
    gname=os.path.splitext(os.path.basename(img))[0]

    Ground_Img = sio.loadmat(f'{GroundTruthRoot}{gname}')
    gimg=[]
    for i in range(0,5):
      gimg.append(Ground_Img['groundTruth'][0][i][0][0][1])

    gimg=np.array(gimg)
    idx = index * 5
    j=0

    f, axes = plt.subplots(1,6,figsize=(20,5))
    r,c=2,3
    axes[0].imshow(Test_Img,aspect='auto')
    axes[0].axis('off')

    for i in range(1,6):
      axes[i].imshow(gimg[j] , aspect='auto',cmap='binary')
      axes[i].axis('off')
      j=j+1
```

```
Display_Img_GroundTruth(Orginal_Test_Images,Ground_Images, 11)
```

```
Display_Img_GroundTruth_Contour(Orginal_Test_Images, Ground_Images, 30)
```

K-means Functions:

F-measure:

```
def f_measure(y_true, y_pred,k):
  contingency__matrix =contingency_matrix(y_true, y_pred)
  perc=[]
  recall=[]
  fmeasure=0
  for i in range(contingency__matrix.shape[0]):
    perc.append(max(contingency__matrix[i])/np.sum(contingency__matrix[i]))


    maxindex=np.argmax(contingency__matrix[i])
    recall.append(max(contingency__matrix[i])/np.sum(contingency__matrix[:,maxindex]))
    fmeasure=fmeasure+(2*perc[i]*recall[i])/(perc[i]+recall[i])
  return round(fmeasure/k,1)
```

Conditional Entropy:

```
def Conditional_entropy(y_true, y_pred,k):
  contingency__matrix = metrics.cluster.contingency_matrix(y_true, y_pred).T

  j=0
  x=0
  HSC=[]
  for i in range(contingency__matrix.shape[0]):
      for j in range(contingency__matrix.shape[1]):
        denim=0
        if contingency__matrix[i][j]/np.sum(contingency__matrix[i])!=0:
          denim=np.log10(contingency__matrix[i][j]/np.sum(contingency__matrix[i]))
        x =x+(-contingency__matrix[i][j]/np.sum(contingency__matrix[i]))*denim

      HSC.append(x)
      x=0



  HSC=np.array(HSC)
  HSC_result=0
  for i in range(contingency__matrix.shape[0]):
    x=np.sum(contingency__matrix[i])/np.sum(contingency__matrix)
    HSC_result=HSC_result+x*HSC[i]

  return HSC_result
```

K-means Functions:

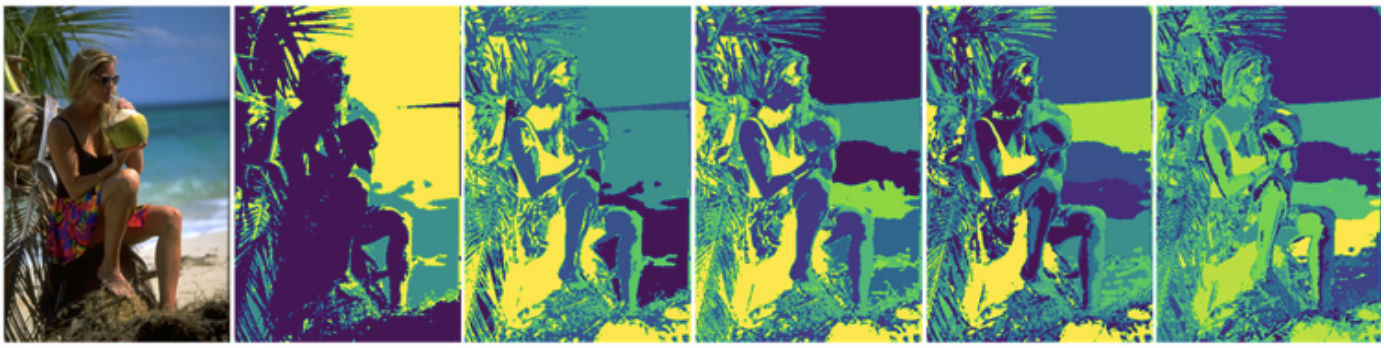
Segmentation :

```
def Segmentation(index):
  f, axes = plt.subplots(1,6,figsize=(20,5))
  j=0
  k=[3,5,7,9,11]
  seg=[]
  fmeasure=np.zeros((5,5))
  fmeasure2=np.zeros((5,5))
  entropy=np.zeros((5,5))
  for j in range(5):
    kmeans = KMeans(n_clusters=k[j])
    kmeans.fit(Test_Images[index])
    labels=kmeans.predict(Test_Images[index])
    for h in range(5):
     fmeasure[j][h] = f_measure(Ground_Images[index*5+h],labels,k[j])
     entropy[j][h] = Conditional_entropy(Ground_Images[index*5+h],labels,k[j])
    centers=kmeans.cluster_centers_
    centers = np.uint8(centers)
    segmented_data = centers[labels.flatten()]
    segmented_image= segmented_data.reshape((Orginal_Test_Images[index].shape))
    segmented_image = np.array([j for j in kmeans.labels_]).reshape(Orginal_Test_Images[index].sh
    seg.append(segmented_image)
  axes[0].imshow(Orginal_Test_Images[index],aspect='auto')
  axes[0].axis('off')
  idx=0
  for i in range(1,6):
    axes[i].imshow(seg[idx] , aspect='auto')
    axes[i].axis('off')
    idx=idx+1
  print('(Kmeans)Average fmeasure for each K')
  print(np.mean(fmeasure,axis=1))
  Totalfmeasures[index]=np.mean(fmeasure,axis=1)
  print('(Kmeans)Average entropy for each K')
  print(np.mean(entropy,axis=1))
  Totalentropy[index]=np.mean(entropy,axis=1)
```

```
Segmentation(11)
```

Add code cell
⌘/Ctrl+M B

```
(Kmeans)Average fmeasure for each K
[1.1  0.72 0.58 0.44 0.44]
(Kmeans)Average entropy for each K
Image i: 11 [0.21428477 0.37916445 0.46156823 0.53727139 0.61155126]
```

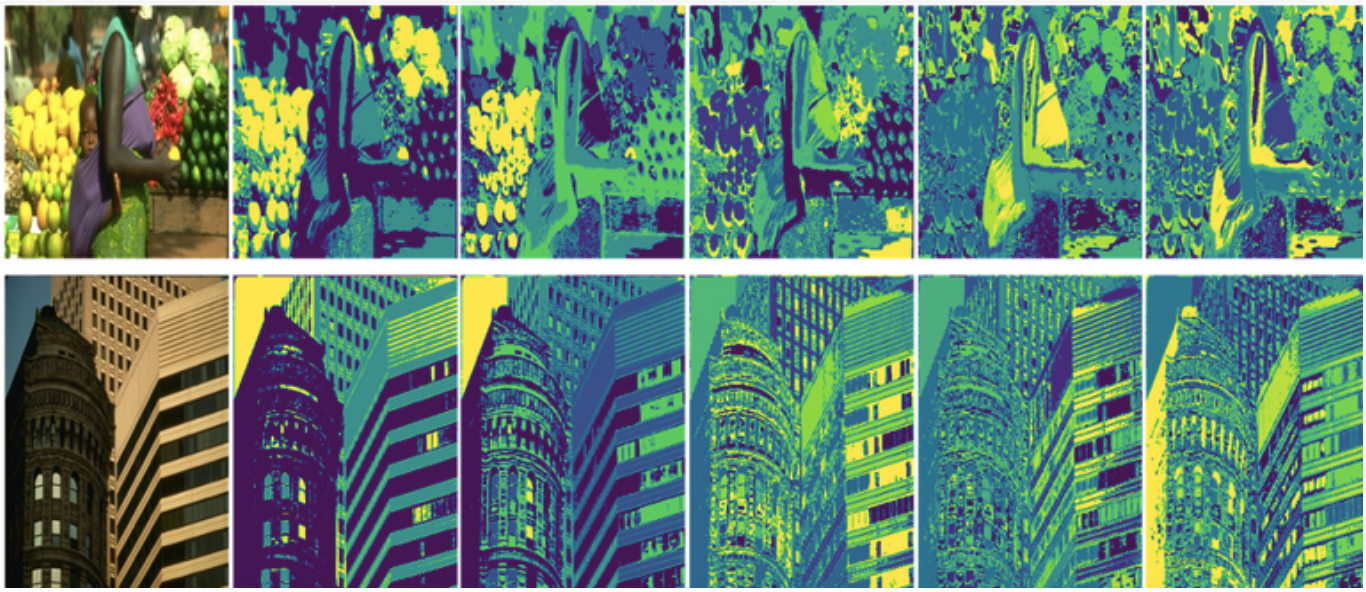


We will change the K of the K-means algorithm between {3,5,7,9,11} clusters, producing different segmentations.

```
i=0
for i in range(0,50):
  Segmentation(i)
```

```
(Kmeans)Average fmeasure for each K
[0.66 0.56 0.42 0.32 0.28]
(Kmeans)Average entropy for each K
Image i: 0 [0.42239157 0.51850114 0.62207625 0.70182101 0.7551748 ]
(Kmeans)Average fmeasure for each K
[0.36 0.22 0.12 0.1  0.04]
(Kmeans)Average entropy for each K
Image i: 1 [0.44263697 0.63851171 0.77792593 0.88931656 0.95372444]
(Kmeans)Average fmeasure for each K
[0.68 0.44 0.3  0.3  0.24]
(Kmeans)Average entropy for each K
Image i: 2 [0.35730505 0.55597505 0.68361688 0.77583823 0.85728294]
(Kmeans)Average fmeasure for each K
[1.02 0.74 0.64 0.54 0.48]
(Kmeans)Average entropy for each K
Image i: 3 [0.30637163 0.45333017 0.55732983 0.6284583  0.65664912]
(Kmeans)Average fmeasure for each K
[0.78 0.58 0.42 0.34 0.26]
(Kmeans)Average entropy for each K
Image i: 4 [0.30699473 0.46020682 0.60686761 0.72314385 0.77304942]
(Kmeans)Average fmeasure for each K
[0.44 0.3  0.18 0.16 0.14]
(Kmeans)Average entropy for each K
```

```
i=0
for i in range(0,50):
  Segmentation(i)
```



Total Fmeasure matrix

F-measure values ranges between 0-1

```
      k1        k2        k3        k4        k5
[[0.39781166 0.42633759 0.35330431 0.3367015  0.31636222]
 [0.43759623 0.29943833 0.23494091 0.18988329 0.1731321 ]
 [0.44734648 0.37739098 0.29719493 0.32521699 0.28842648]
 [0.37882184 0.41460754 0.37053704 0.35722448 0.37683654]
 [0.44831779 0.50203373 0.41243752 0.34914031 0.30093847]
 [0.44807111 0.3201621  0.3049202  0.24587236 0.20427926]
 [0.62274945 0.46885103 0.37389075 0.31723285 0.27236743]
 [0.56534618 0.45060722 0.38853618 0.34699723 0.31601115]
 [0.58095923 0.45328683 0.41444986 0.35009246 0.31611274]
 [0.56566322 0.42368017 0.40107182 0.3447441  0.31529395]
 [0.42200223 0.37515119 0.40690045 0.35674263 0.3172831 ]
 [0.51476627 0.43600357 0.44184636 0.39088938 0.35555233]
 [0.61316831 0.47352656 0.37814932 0.30957377 0.2728211 ]
 [0.32398549 0.2834918  0.26789692 0.23194519 0.21381751]
 [0.54184479 0.4293101  0.34688126 0.29790346 0.25287055]
 [0.5222616  0.43940822 0.41065198 0.37398902 0.34001192]
 [0.40703378 0.4466599  0.39257355 0.37594833 0.34368167]
 [0.53396028 0.39227752 0.34744145 0.29099054 0.2483272 ]
 [0.49651602 0.46556894 0.47219653 0.43810266 0.40846824]
 [0.60166967 0.51039081 0.42727718 0.38533067 0.34249672]
 [0.65364406 0.52427275 0.43379718 0.44207613 0.39885117]
 [0.50770453 0.4933908  0.44278403 0.39800667 0.35390977]
 [0.71930739 0.63960876 0.52400841 0.47934494 0.46135366]
 [0.72303892 0.63639168 0.5863255  0.5310872  0.47370597]
 [0.51867755 0.39962717 0.43568488 0.34860859 0.33109736]
 [0.50045738 0.40853964 0.31743804 0.25252588 0.2278968 ]
 [0.46312026 0.37855942 0.36941844 0.31058159 0.29700146]
 [0.65913026 0.51345671 0.57345338 0.51422315 0.4629692 ]
 [0.31852685 0.35978568 0.37028201 0.38644635 0.40203695]
 [0.51459446 0.3549523  0.3233129  0.2676318  0.23186938]
```

Total Entropy matrix

```
      k1        k2        k3        k4        k5
[[0.77553708 0.68475882 0.66728127 0.64580022 0.61880103]
 [0.31842193 0.31378551 0.31174516 0.31154676 0.30182677]
 [0.32528068 0.29888312 0.28927608 0.27376666 0.27136657]
 [1.00373098 0.95797512 0.91765999 0.87857479 0.85013629]
 [0.58032002 0.5289275  0.51411388 0.50395315 0.4953581 ]
 [0.28390471 0.2731391  0.26792469 0.26537992 0.26472217]
 [0.39771453 0.37822214 0.37203928 0.36631813 0.36318163]
 [0.61373949 0.52472855 0.4977638  0.46921282 0.44384654]
 [0.61561198 0.59831408 0.57810546 0.57248292 0.56463988]
 [0.44571418 0.38564301 0.34454089 0.34040815 0.32696753]
 [0.63969045 0.61365323 0.5972807  0.59129837 0.56753992]
 [0.77153013 0.68850646 0.64922906 0.62877022 0.61573531]
 [0.32940668 0.31878657 0.3029425  0.29784296 0.29442441]
 [0.80716865 0.78994352 0.76453363 0.76130418 0.75025164]
 [0.5214671  0.50388204 0.49565732 0.48922507 0.48601501]
 [0.55395749 0.54425417 0.52796351 0.49940506 0.48636644]
 [0.91888944 0.80187871 0.77774423 0.72682535 0.71811239]
 [0.3911246  0.36850374 0.36117649 0.35762681 0.35541686]
 [0.80422746 0.73662029 0.66410397 0.63147629 0.61826831]
 [0.53652824 0.482923   0.45371161 0.44203866 0.41286182]
 [0.40142372 0.34338314 0.33704733 0.282286   0.28114289]
 [0.69159513 0.53842096 0.47817279 0.47113454 0.4655069 ]
 [0.29125928 0.25501198 0.24985124 0.23224629 0.19733683]
 [0.45784838 0.34220743 0.30698146 0.2591445  0.24502116]
 [0.44622639 0.41563001 0.39513714 0.38696184 0.35081871]
 [0.31281878 0.30223658 0.29825987 0.29677438 0.29553192]
 [0.54577356 0.52774595 0.45790876 0.45783052 0.40546183]
 [0.86131146 0.79510057 0.72643079 0.7085423  0.68346401]
 [1.15161414 1.06857961 1.00060044 0.96247002 0.89854174]
 [0.14618072 0.12931899 0.1315051  0.12358828 0.10865955]
 [0.82829896 0.78911427 0.72641239 0.69602783 0.68334829]
 [0.39638209 0.30563318 0.29961331 0.2972485  0.28617125]
 [0.47607408 0.4498925  0.44224477 0.44018874 0.40836981]
 [0.7108448  0.68175216 0.66990633 0.66208519 0.65810934]
```

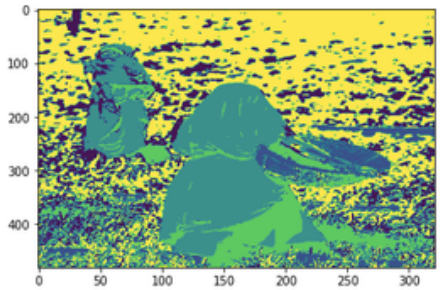
Getting the segmented images for 5 photos with k=5 :

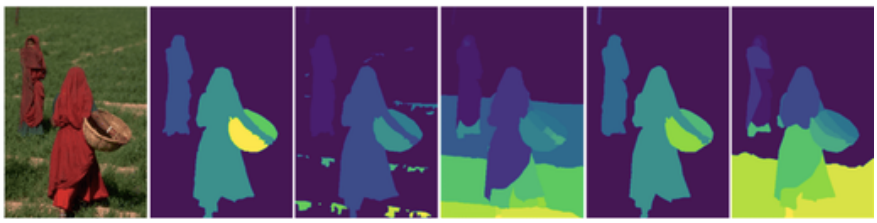
```
def SegmentationK5(index,i):
 j=0
 k=[3,5,7,9,11]
 seg=[]
 fmeasure=np.zeros((1,5))
 entropy=np.zeros((1,5))
 kmeans = KMeans(n_clusters=k[i])
 kmeans.fit(Test_Images[index+j])
 labels=kmeans.predict(Test_Images[index])
 for i in range(0,5):
  fmeasure[0][i] = f_measure(Ground_Images[index*5+i],labels,5)
  entropy[0][i] = Conditional_entropy(Ground_Images[index*5+i],labels,5)

 centers=kmeans.cluster_centers_
 centers = np.uint8(centers)
 segmented_data = centers[labels.flatten()]
 segmented_image= segmented_data.reshape((Orginal_Test_Images[index].shape))
 segmented_image = np.array([j for j in kmeans.labels_]).reshape(Orginal_Test_Images[index].shape
 seg.append(segmented_image)
 plt.imshow(segmented_image,aspect='auto')
 print('Average F-measure')
 print(np.mean(fmeasure))
 print('Average Entropy')
 print(np.mean(entropy))
```

```
[ ] def Segmentation_Aganist_GroundTruth(index,i):
      from sklearn.cluster import KMeans
      j=0
      k=[3,5,7,9,11]
      seg=[]
      for j in range(5):
       print("segmentation results using K-means at k=",k[i])
       SegmentationK5(index+j,1)
       Display_Img_GroundTruth(Orginal_Test_Images,Ground_Images, index+j)
```

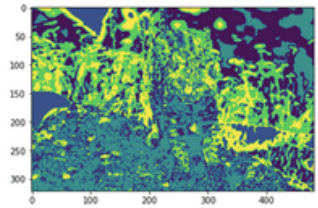
Ground Truth vs Segmentation

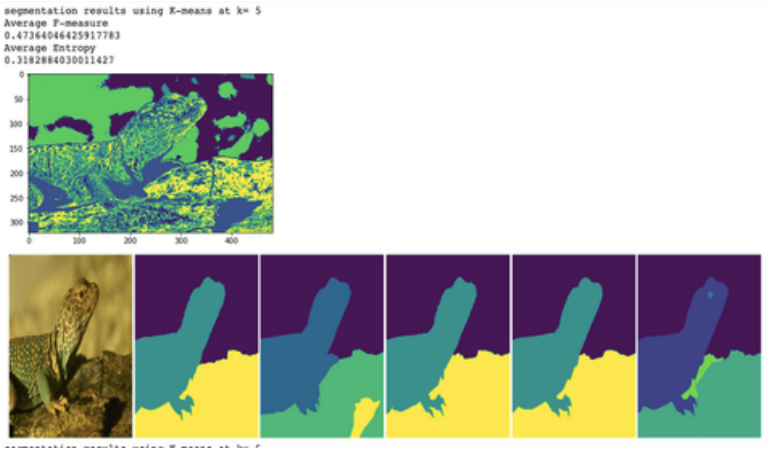
```
segmentation results using K-means at k= 5
Average F-measure
0.42156502941020674
Average Entropy
0.3847935093412197
```

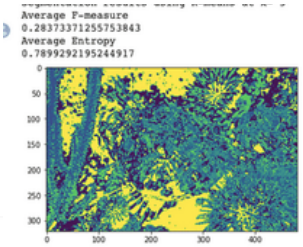


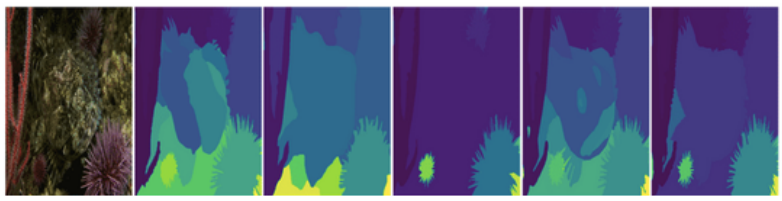


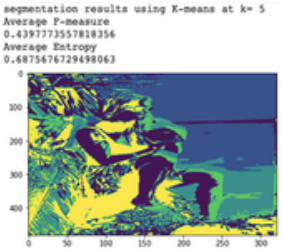
```
segmentation results using K-means at k= 5
Average F-measure
0.37918148173512434
Average Entropy
0.6131378183805177
```









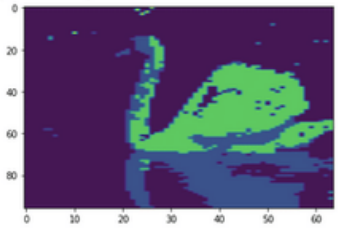


```
def spectral_clustering(index, scale_percent):
  # Reshaped Height and Width
  width = int(Orginal_Test_Images[index].shape[0] * scale_percent / 100)
  height = int(Orginal_Test_Images[index].shape[1] * scale_percent / 100)
  dim = (width, height)
  img=Orginal_Test_Images[index]
  resized = cv2.resize(img, dim)
  resizedTestImg = (resized).reshape(-1 ,3)


  cluster = SpectralClustering(n_clusters=5, affinity='nearest_neighbors', n_neighbors=5).fit_predict(resizedTestImg)
  fmeasure=np.zeros((1,5))
  entropy=np.zeros((1,5))
  segmented_results = np.array([j for j in cluster]).reshape(height,width)
  dim2=(segmented_results.shape[0],segmented_results.shape[1])
  for h in range(5):
     x = cv2.resize(Ground_Images[index*5+h], dim2)
     fmeasure[0][h] = f_measure(x,segmented_results,5)
     entropy[0][h] = Conditional_entropy(x,segmented_results,5)
  print('(Spectral Clustering) Average fmeasure for each K')
  print(np.mean(fmeasure,axis=1))
  print('(Spectral Clustering) Average entropy for each K')
  print(np.mean(entropy,axis=1))
  plt.imshow(segmented_results,aspect='auto')

  plt.show()
```

Ground Truth vs Spectral Clustering

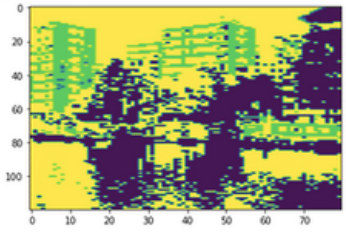
```
spectral_clustering(40,20)
Display_Img_GroundTruth(Orginal_Test_Images,Ground_Images, 40)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_spectral_embedding.py:236: UserWarning: Graph is not fully connected
  warnings.warn("Graph is not fully connected, spectral embedding"
(Spectral Clustering) Average fmeasure for each K
[0.28]
(Spectral Clustering) Average entropy for each K
[0.34745219]
```

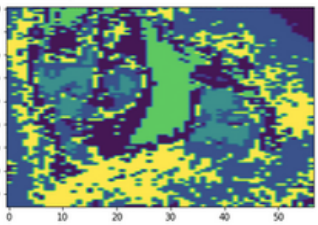


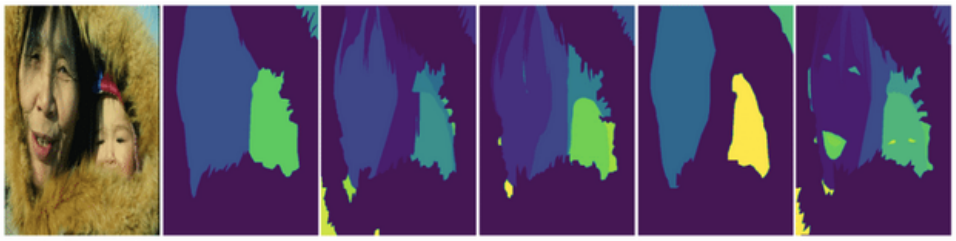
```
spectral_clustering(30,25)
Display_Img_GroundTruth(Orginal_Test_Images,Ground_Images,3
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_spe
  warnings.warn("Graph is not fully connected, spectral embe
(Spectral Clustering) Average fmeasure for each K
[0.38]
(Spectral Clustering) Average entropy for each K
[0.3777801]
```

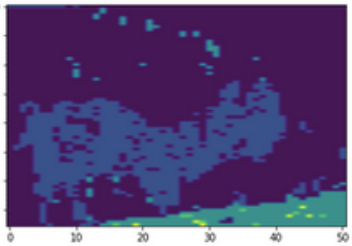


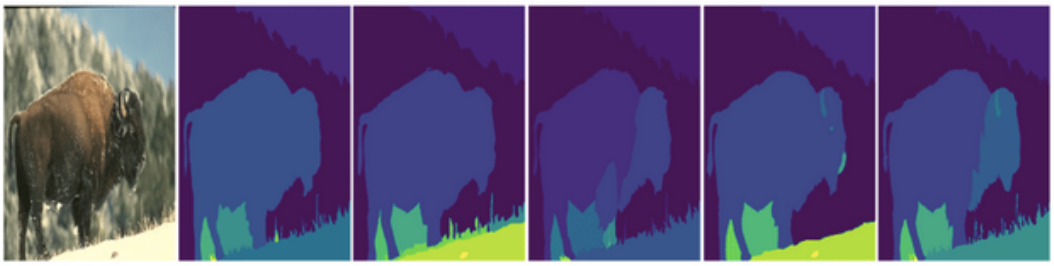
```
(Spectral Clustering) Average fmeasure for each K
[0.28547931]
(Spectral Clustering) Average entropy for each K
[0.52415097]
```

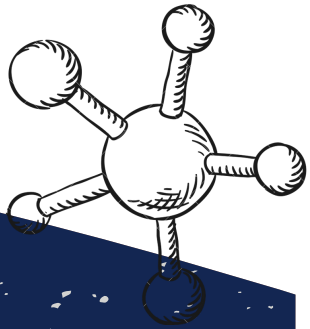




```
(Spectral Clustering) Average fmeasure for each K
[0.31107071]
(Spectral Clustering) Average entropy for each K
[0.57393222]
```



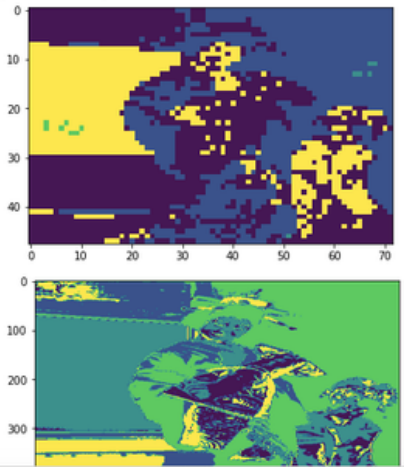




Segmentation vs Spectral Clustering

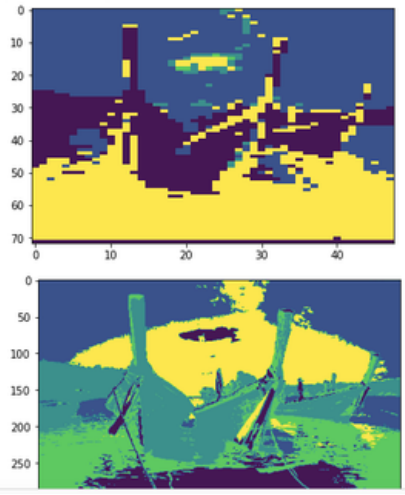

```
spectral_clustering(18,15)
SegmentationK5(18,1)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_
  warnings.warn("Graph is not fully connected, spectral e
(Spectral Clustering) Average fmeasure for each K
[0.42]
(Spectral Clustering) Average entropy for each K
[0.43227585]
```

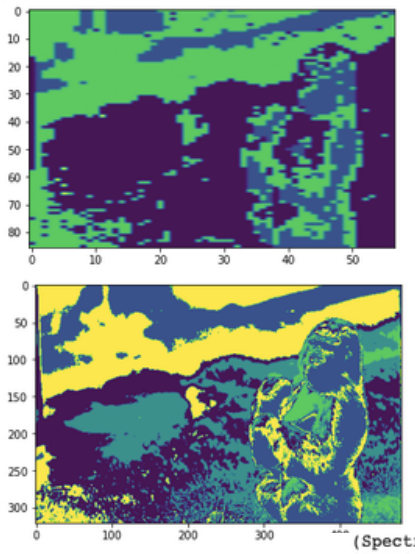


```
spectral_clustering(19,15)
SegmentationK5(19,1)
```

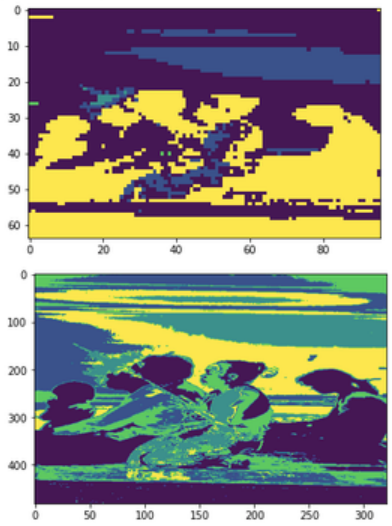
```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_s
  warnings.warn("Graph is not fully connected, spectral em
(Spectral Clustering) Average fmeasure for each K
[0.56]
(Spectral Clustering) Average entropy for each K
[0.31107478]
```



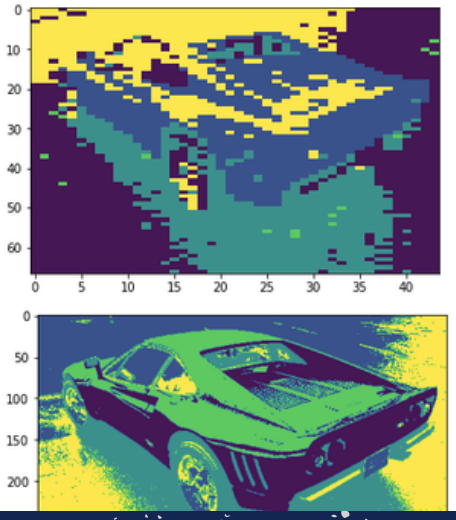
```
(Spectral Clustering) Average fmeasure for each K
[0.3]
(Spectral Clustering) Average entropy for each K
[0.41817106]
```



```
(Spectral Clustering) Average fmeasure for each K
[0.42]
(Spectral Clustering) Average entropy for each K
[0.32541807]
```



```
(Spectral Clustering) Average fmeasure for each K
[0.32]
(Spectral Clustering) Average entropy for each K
[0.55208321]
```



Spatial layout of the pixels.

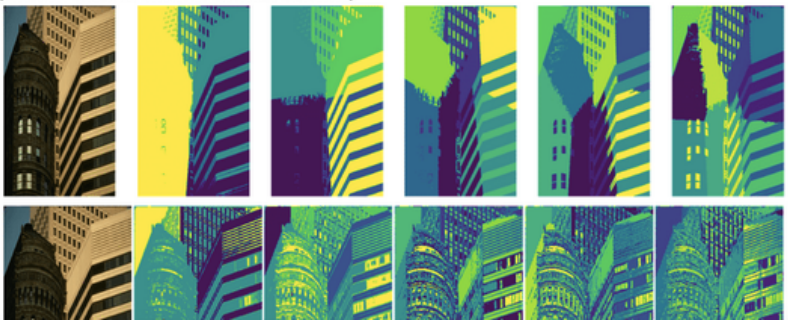
```
def spatialDistance(orgimage,index):
 f, axes = plt.subplots(1,6,figsize=(20,5))
 j=0
 k=[3,5,7,9,11]
 seg=[]
 fmeasure=np.zeros((5,5))
 entropy=np.zeros((5,5))
 for i in range(5):
  kmeans = KMeans(n_clusters=k[i])
  kmeans.fit(orgimage)
  labels=kmeans.predict(orgimage)
  for h in range(5):
     fmeasure[i][h] = f_measure(Ground_Images[index*5+h],labels,k[i])
     entropy[i][h] = Conditional_entropy(Ground_Images[index*5+h],labels,k[i])
  centers=kmeans.cluster_centers_
  centers = np.uint8(centers)
  segmented_data = centers[labels.flatten()]
  segmented_image= segmented_data.reshape((orgimage.shape))
  segmented_image = np.array([j for j in kmeans.labels_]).reshape(Orginal_Test_Images[index].shape[0],Orginal_Test_Images[index].shape[1]
  seg.append(segmented_image)
 seg=np.array(seg)
 print('(Spatial Layout)Average Fmeasure for each K(Spatial La
 print(np.mean(fmeasure,axis=1))
 print('(Spatial Layout)Average entropy for each K')
 print(np.mean(entropy,axis=1))
 axes[0].imshow(Orginal_Test_Images[index], aspect='auto')
 axes[0].axis('off')
 j=0
 for i in range(1,6):
    axes[i].imshow(seg[j], aspect='auto')
    axes[i].axis('off')
    j=j+1

plt.subplots_adjust(wspace=0.005,hspace=0.005)
```

Choosing the suitable photo to operate the spatial distance on it :

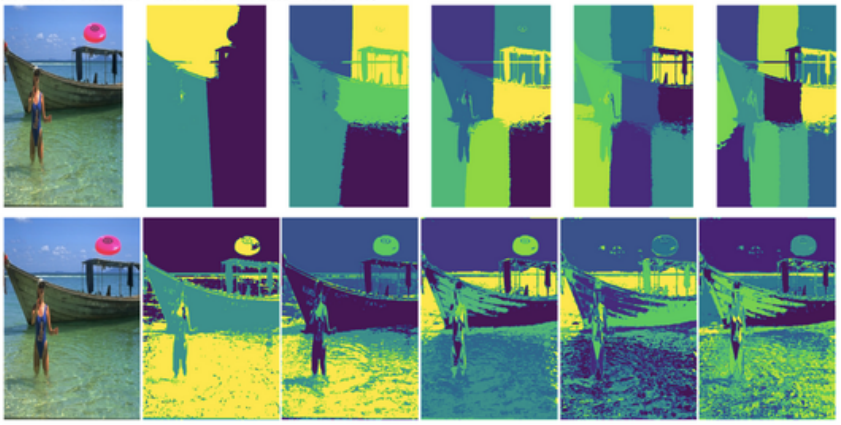
```
def choosing_sp(index):
  img3d = np.copy(Orginal_Test_Images[index])
  img2d = img3d[:,:,0]     #   convert to 2D array
  row, col = img2d.shape
  x=[]
  y=[]
  for i in range(row):
      for j in range(col):
            x.append(i) #       get x indices
            y.append(j) #       get y indices
  x= np.array(x).reshape(img3d.shape[0]*img3d.shape[1],1)
  y=np.array(y).reshape(img3d.shape[0]*img3d.shape[1],1)
  img3d2 = img3d.reshape(img3d.shape[0]*img3d.shape[1],3)
  finalimg = np.append(img3d2,x,axis = 1)
  finalimg = np.append(finalimg,y,axis = 1)
  finalimg = finalimg.reshape(img3d.shape[0], img3d.shape[1],5)
  x=finalimg.reshape((-1,5))
  x=np.float32(x)
  spatialDistance(x,index)
```

Segmentation vs Spatial layout

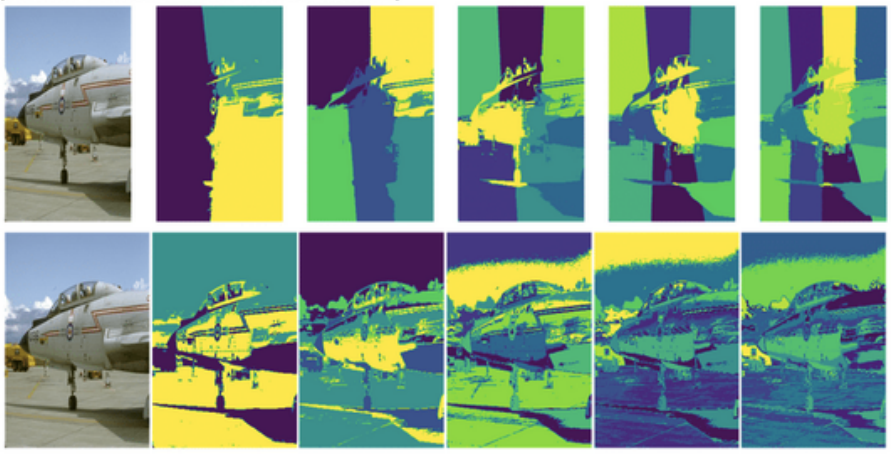
```
choosing_sp(4)
Segmentation(4)

(Spatial Layout)Average Fmeasure for each K(Spatial Layout)
[0.64051518 0.51277397 0.45086072 0.46156842 0.40287551]
(Spatial Layout)Average entropy for each K
[0.41343435 0.33967607 0.33430899 0.28199832 0.27361633]
(Kmeans)Average fmeasure for each K
[0.44845502 0.50194549 0.41241417 0.34935016 0.30300896]
(Kmeans)Average entropy for each K
[0.58017311 0.52890717 0.51410615 0.50351381 0.49425651]
```



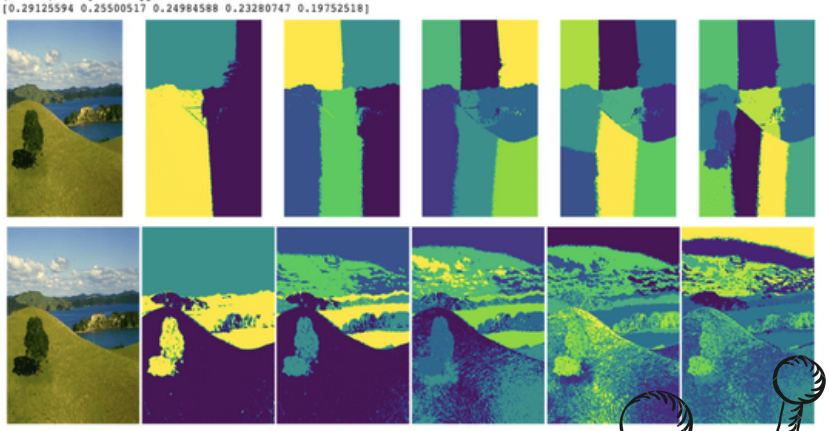
```
[0.4075446  0.3555404  0.30020401 0.2942693  0.28547603]
```

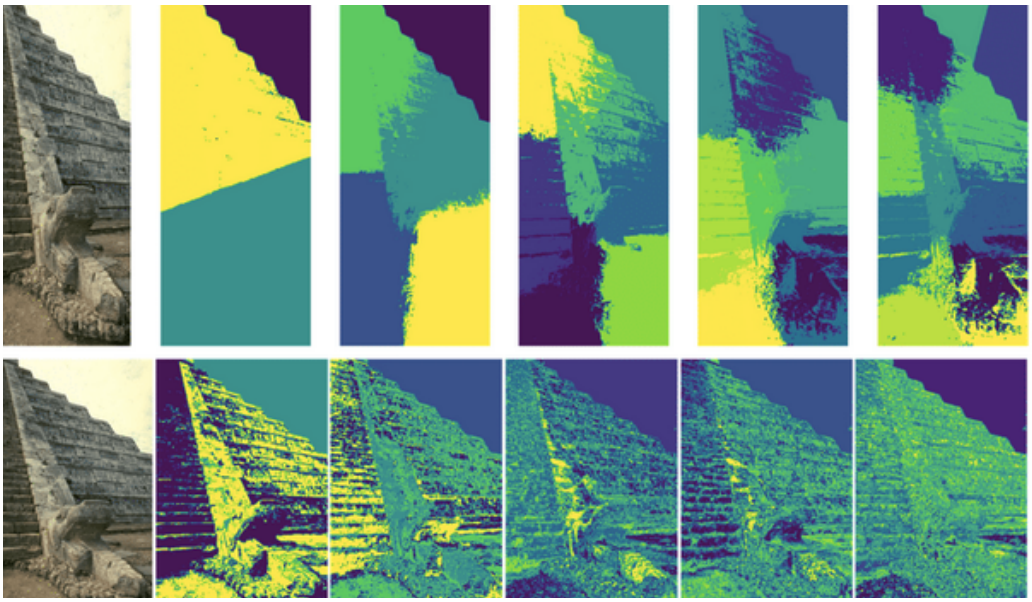


```
(Kmeans)Average entropy for each K
[0.40161385 0.34338245 0.33783986 0.32001832 0.28329277]
```



```
[0.71928915 0.64036596 0.52292484 0.47818625 0.46091507]
(Kmeans)Average entropy for each K
[0.29125594 0.25500517 0.24984588 0.23280747 0.19752518]
```





.57828051 0.54641926 0.54131689 0.49044167 0.45440924]