IMAGE SEGMENTATION

Done By: - Mahmoud Elbanna 6407
– Amr Ashraf 6301
– Mohamed Kamal 6331

ABSTRACT
Image Segmentation is the process of partitioning a digital image into multiple image segments, also known as image regions or image objects, Representing each object in different color.

Course
Pattern Recognition – ML

Image Segmentation

This Assignment focus on how to cluster the same object together.
This Approach will be done by K-means, by grouping together the same set of pixels together.

1)Data:

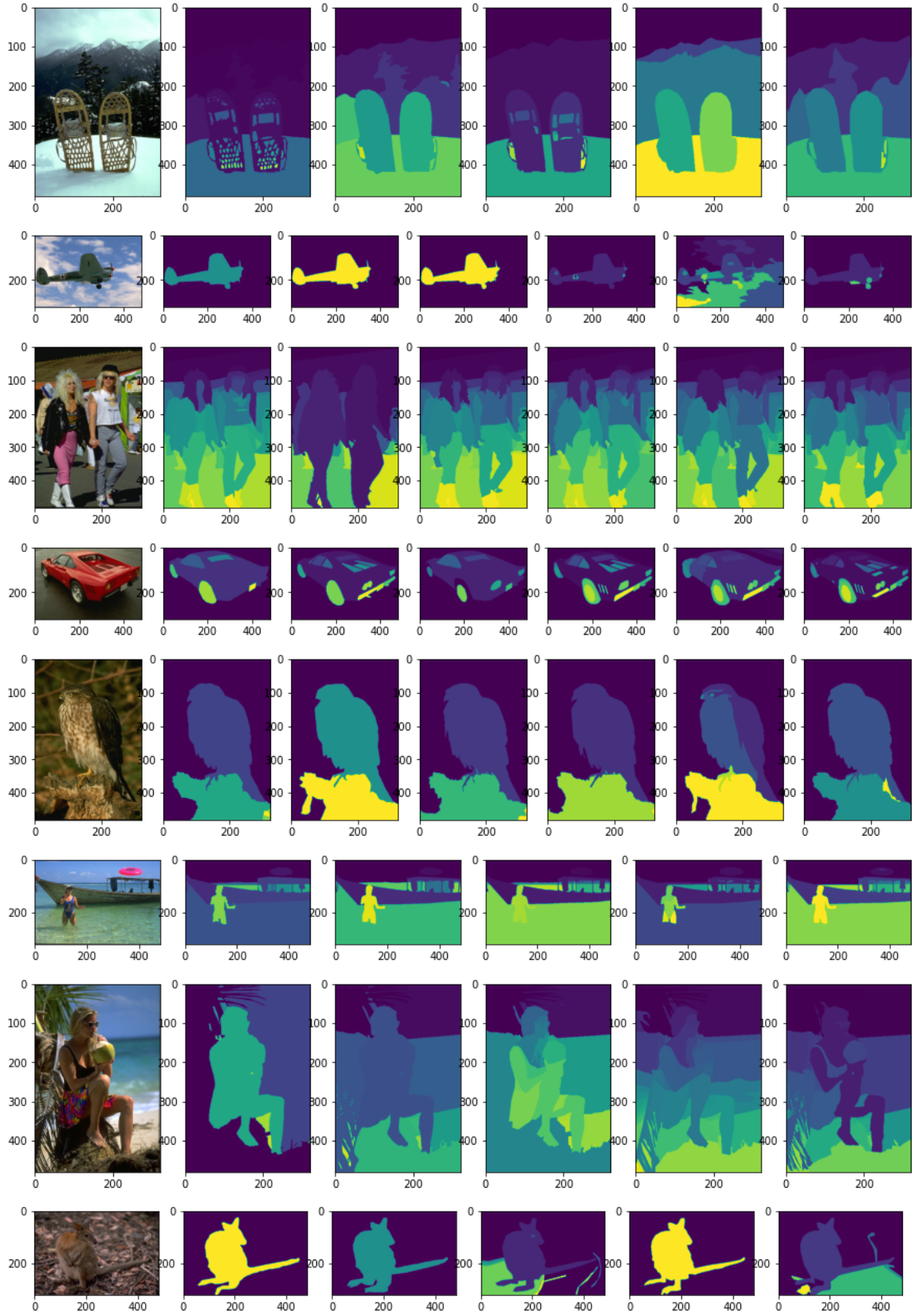
The Data Used is from Berkeley Segmentation Benchmark: http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/BSR/BSR_bsds500.tgz

2)Reading and Visualizing the images and their ground Truth:

```
def read_jpg():
  images = []
  for i in range(1,51):
    images.append(img.imread('/content/drive/MyDrive/imageSegmentation/test/img ('+str(i)+').jpg'))
  return images
```

```
def read_GT():
  GT_images = []
  for i in range(1,51):
    x=[]
    mat = scipy.io.loadmat('/content/drive/MyDrive/imageSegmentation/testGT/gt ('+str(i)+').mat')
    for k in np.squeeze(mat['groundTruth']).reshape(-1):
       x.append(k[0][0][0])
    GT_images.append(x)
  return GT_images
```

```
def drawxc(images,GT_images):
  for i in range(0,50):
    num = len(GT_images[i]) + 1  # +1 for the image
    fig, ax = plt.subplots(1, num, figsize=(15,15))
    ax[0].imshow(images[i])
    for j in range(1,num):
        ax[j].imshow(GT_images[i][j-1])
    plt.show()
```



3)a)Segmentation Using K-means:

We defined a kmeans function on our own by passing the parameters(image and number of clusters) so we made 2 functions one to send the list of needed clusters[3 5 7 9 11] and loop to the other by passing every time number of clusters, then generate random centroids but from the given data and to optimize it we made the distance between each 2 adjacent centroids as similar as possible , for example if we have array of numbers from 0 to 90 and we need to generate random 3 centroids there is a high possibility that the 1st centroid will be in range (0,30) the 2nd (31,60) the 3rd (61,90) to decrease number of iterations applied to update the centroids. After that we receive the output of each K in a list and return it

```
def kmeans(images,k):
  results=[]
  for image in images:
    x=image.reshape((-1,3))
    kmean_labels=K_means(x,k)
    results.append(kmean_labels.reshape(image.shape[:-1]))
  return results
```

```
[4]  def K_means(image,nk):
       A=image
       ll=[]
       for kk in range(nk,nk+1,2):
         C=np.zeros((kk,3))
         C[0]=A[0]
         C[kk-1]=A[154400]
         v=int(154401/kk)
         for i in range(1,kk-1): # set the intial values for centroids
           C[i]=A[v]
           v=v+int(154401/kk)

         C2=np.zeros((kk,3))
         C2[:,:]=C[:,:]
         N=0
         ED_array2=np.zeros((kk,154401))
         itr=0
         while (itr<11):
           itr+=1
           for i in range(0,kk):
             for j in range(0,154401):
               d=C2[i]-A[j]
               ED=0
               for s in range(0,3):
                 ED+=d[s]*d[s]
               ED_array2[i,j]=math.sqrt(ED)

           y2=np.zeros((154401,1))
           c=np.zeros((154401*kk,3))
           ic=np.zeros([kk, 1], dtype = int)
           for m in range(0,154401):
             idx = ED_array2[:,m].argsort()[::1]
             c[int(idx[0])*154401+ic[idx[0]]]=A[m]
             ic[idx[0]]+=1
             y2[m]=idx[0]

           C2_test=np.zeros((kk,3))

           for i in range(0,kk):
             if(int(ic[i])==1) : C2_test[i]=c[i*154401]
             else : C2_test[i]=np.mean(c[i*154401:i*154401+int(ic[i]),:],a

           flag=1
           for i in range(0,kk):
             if(C2_test[i,0]!=C2[i,0] or C2_test[i,1]!=C2[i,1]):
               flag=0
               break
           if(flag==1):break
         ll.append(y2)
       return y2
```

b)F-measure and Conditional Entropy:

here we calculate the f-measure and entropy as discussed in the lecture, but first we masked the ground truth as we found that the number of clusters in the ground truth differs from that we applied, so we masked the ground truth by taking the best clusters as its (the dominant clusters) . for example if we applied kmeans with k = 3 and the ground truth was k = 6 we will take the first 2 dominant cluster in the groundtruth and will map it next with the first 2 dominant clusters in our result and take the remaining clusters in the GT as only one cluster and map it next with the 3rd cluster of our result.

Then we calculate the contingency matrix using a built-in function which map each cluster in our result with the specific one in the GT by passing to it the masked GT and our result from the contingency matrix we calculate the precision (purity), recall, TP, TN , FP and FN to get the F-measure and conditional entropy

```
def Fmeasure(contingM):
  score = 0
  for i in range(0,len(contingM[0,:])):
    mx = np.argmax(contingM[:,i])
    prec = contingM[mx,i]/np.sum(contingM[:,i])
    recall = contingM[mx,i]/np.sum(contingM[mx,:])
    score+= (2 * prec * recall) / (prec + recall)
  return score/len(contingM[0,:])

def Entropy(contingM):
  Entro = 0
  s=0
  for i in range(0,len(contingM[0,:])):
    s=0
    for j in range(0,len(contingM)):
      if(contingM[j][i] != 0.0):
          v=contingM[j][i]/sum(contingM[:,i])
          s-=(v)*math.log2(v)
    Entro+=(sum(contingM[:,i])/(154401))*s
  return Entro
```

```
def mask(Gt,ksd):
  cop=np.copy(Gt)
  flat_array = cop.flatten()
  count_arr = np.bincount(flat_array)
  idx = count_arr.argsort()[::-1]
  for l in range(0,len(cop[:])):
   for m in range(0,len(cop[0,:])):
     if(l<481 and m<321):
      if(cop[l][m] not in idx[0:ksd-1]) : cop[l][m]=0
  return cop
```

```
#y=results[0][9]
k = [3,5,7,9,11]
avgFMDataset = []
avgENTDataset = []
TotavgFMk = []
Totfmeasures= []
TotavgENTk = []
Totentropy= []

for IM in range(0,len(GT_images)):
  avgFMk = []
  fmeasures= []
  avgENTk = []
  entropy= []
  for i in range(0,len(k)):
    M = len(GT_images[IM])
    for j in range(0,M):
      if(len(np.unique(GT_images[IM][j]))>k[i]) : GTR=mask(GT_images[IM][j],k[i])
      else : GTR=GT_images[IM][j]
      contingM = contingency_matrix(GTR,results[i][IM])
      fmeasures.append(Fmeasure(contingM))
      entropy.append(Entropy(contingM))
    Totfmeasures.append(fmeasures)
    Totentropy.append(entropy)
    avgFMk.append(sum(fmeasures[M*i:M*(i+1)])/M)
    avgENTk.append(sum(entropy[M*i:M*(i+1)])/M)
  TotavgFMk.append(avgFMk)
  TotavgENTk.append(avgENTk)
  avgFMDataset.append(sum(TotavgFMk[IM])/len(k))
  avgENTDataset.append(sum(TotavgENTk[IM])/len(k))
```

c) Displaying Validation Results

we displayed the results in a table using pandas_lib Dataframe we display entropy of each image by taking average of per cluster as for every image we applied kmeans with different number of clusters (k numbers) and for every GT there is (M numbers) for every i in k we calculate the entropy and fmeasure with every j in M so we get the average per k, so for every image there will be 5 average results for entropy and 5 for Fmeasure. And after displaying this result we calculate the average results per each image and displayed it in another table.

Conclusion: the best Fmeasure and Entropy was when k = 3 as its for the average it has the least entropy and the highest Fmeasure

```
x= np.zeros((50,10))
for i in range(0,50):
  for j in range(0,5):
    x[i][j*2] = TotavgENTk[i][j]
    x[i][j*2+1] = TotavgFMk[i][j]
y = np.zeros((50,2))
for i in range(0,50):
    y[i][0] = avgENTDataset[i]
    y[i][1] = avgFMDataset[i]
lst = list(range(1,51))
```

```
[186] from pandas._libs import index
    import pandas as pd
    import numpy as np
    import matplotlib as mpl
    l = [[1,2,3,4,5,6],[1,2,3,4,5,6],[1,2,3,4,5,6]]

    avgPerCluster = pd.DataFrame(x,
                    index=pd.Index(lst),
                    columns=pd.MultiIndex.from_product([['using 3 clusters (k=3)', 'using 5 clusters (k=5)', 'using 7 clusters (k=7)','using 9 clusters (k=9)','using 1

    avgPerCluster.style.set_table_styles(
        [{'selector': '',
          'props': [('background-color', '#652334')]},{'selector': 'tr:hover',
          'props': [('background-color', '#f52f55')]},{'selector' : '',
                              'props' : [('border',
                                          '6px solid #ff748f')]}]
         )
```

Download audio from this page

```
[187] avgPerDataset = pd.DataFrame(y,
                      index=pd.Index(lst),
                      columns=pd.MultiIndex.from_product([['Average evaluation per image'],['AVGEntropy', 'AVGFmeasure']], names=['', 'validation type']))
    avgPerDataset.style.set_table_styles(
        [{'selector': '',
          'props': [('background-color', '#652334')]},{'selector': 'tr:hover',
          'props': [('background-color', '#f52f55')]},{'selector' : '',
                              'props' : [('border',
                                          '6px solid #ff748f')]}]
         )
```

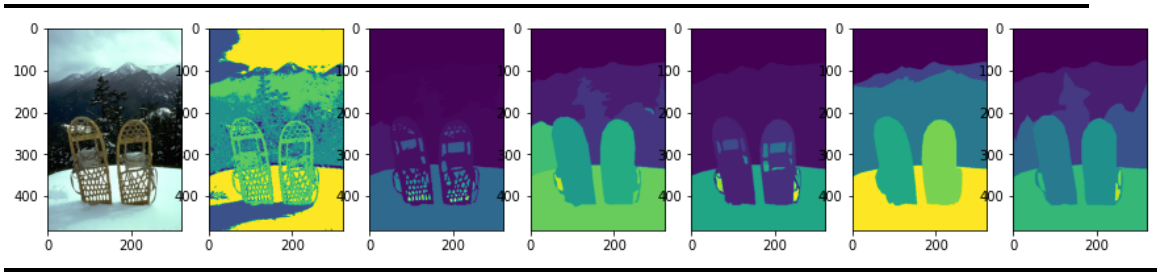
no. of clusters:	using 3 clusters (k=3)		using 5 clusters (k=5)		using 7 clusters (k=7)		using 9 clusters (k=9)		using 11 clusters (k=11)	
validation type:	AVGEntropy	AVGFmeasure	AVGEntropy	AVGFmeasure	AVGEntropy	AVGFmeasure	AVGEntropy	AVGFmeasure	AVGEntropy	AVGFmeasure
1	0.853975	0.579883	1.510222	0.427571	1.597302	0.418571	1.625414	0.360076	1.596690	0.308052
2	0.327775	0.728965	0.349016	0.492387	0.350432	0.372810	0.342070	0.335092	0.316586	0.296984
3	2.060089	0.411076	1.972595	0.446260	2.153521	0.315515	2.108580	0.355273	2.024873	0.359521
4	1.051471	0.580171	1.535271	0.443667	1.744803	0.432216	1.942805	0.355498	2.094203	0.310304
5	1.383635	0.374700	0.758007	0.499294	0.683572	0.428366	0.602006	0.341888	0.583118	0.308303
6	1.324454	0.525917	1.167475	0.537689	1.112706	0.425763	1.093738	0.370434	0.939453	0.348667
7	1.395181	0.527160	1.316937	0.428697	1.319495	0.315391	1.254508	0.281413	1.327771	0.268465
8	1.107538	0.508697	1.249310	0.304868	1.068211	0.274792	0.949852	0.282923	1.011232	0.187391
9	2.072765	0.379400	1.888243	0.554134	2.085531	0.408161	1.944266	0.492844	2.079765	0.397328
10	1.753837	0.590961	1.612005	0.522355	1.673811	0.452462	1.513814	0.410437	1.648101	0.356959
11	2.129131	0.394256	2.243047	0.305842	2.313865	0.287129	2.353585	0.294771	2.556456	0.272599
12	1.775722	0.389427	1.771760	0.347089	1.844972	0.270274	1.581927	0.298086	1.504825	0.266051
13	1.155964	0.539661	1.597532	0.441223	1.906510	0.364286	2.060998	0.302106	2.151502	0.271593
14	1.032446	0.474097	1.511486	0.389686	1.704903	0.339873	1.770144	0.311734	1.808208	0.289895
15	1.029342	0.410189	1.479211	0.357448	1.810341	0.226633	1.924500	0.317200	2.002866	0.307581
16	1.338867	0.676030	0.966436	0.631504	1.139744	0.507830	0.870718	0.471081	0.846365	0.427163
17	1.504706	0.425860	0.996335	0.509973	0.981972	0.389228	1.288252	0.251549	0.913718	0.280256
18	1.267480	0.433598	1.430845	0.424287	1.503436	0.353987	1.464743	0.334402	1.502343	0.258091
19	2.015945	0.252743	2.064009	0.308500	1.876152	0.326456	2.069873	0.229418	1.897678	0.224688
20	1.200628	0.450285	1.287281	0.362847	1.216585	0.320482	1.106819	0.311397	1.147068	0.247393
21	1.310738	0.664004	0.953003	0.559565	0.865208	0.466351	0.751779	0.488023	0.796475	0.401232
22	1.407655	0.431936	1.186585	0.422478	1.172616	0.364331	1.231811	0.288950	1.143343	0.255057
23	1.628101	0.436357	1.410581	0.457133	1.154688	0.490823	1.142670	0.430712	1.291474	0.276707
24	1.696901	0.473100	1.958170	0.349937	1.959625	0.300308	1.930079	0.272868	1.921272	0.265082
25	1.191625	0.557238	1.015651	0.429802	1.009598	0.369628	1.015123	0.306515	0.975895	0.272909
26	0.972533	0.405529	0.972086	0.381660	1.035915	0.252487	0.958265	0.238756	0.959590	0.198819
27	0.649074	0.419370	0.466677	0.408851	0.503114	0.273388	0.456947	0.276820	0.389888	0.239797
28	1.897842	0.320397	1.973393	0.318846	2.233706	0.288502	2.360341	0.249100	2.376471	0.212178
29	1.558883	0.450169	1.689582	0.333121	1.739338	0.319558	1.781852	0.258072	1.811237	0.235880
30	1.558837	0.466104	1.179332	0.552572	1.186291	0.436152	1.185858	0.358433	1.102883	0.379325
31	1.901215	0.451729	1.822523	0.299708	1.614906	0.293010	1.449163	0.318190	1.511773	0.250791
32	2.219878	0.372409	2.311903	0.374268	2.312852	0.394481	2.252943	0.328550	2.205953	0.349462
33	1.211177	0.449221	1.223589	0.472786	1.230747	0.434966	1.455315	0.333245	1.291369	0.366215
34	1.026475	0.531427	1.444808	0.431223	1.644809	0.303728	1.677960	0.349079	1.732567	0.296423
35	1.052886	0.454102	1.530836	0.411355	1.644082	0.422933	1.975993	0.306348	1.840712	0.375734
36	1.918265	0.453916	1.710761	0.448254	1.377037	0.428566	1.277393	0.374865	1.452624	0.291297
37	2.051747	0.368257	1.887150	0.382537	1.504463	0.388611	1.716990	0.313117	1.730888	0.287954
38	1.056825	0.394981	1.050246	0.292352	1.025253	0.240054	1.006857	0.188708	1.012759	0.167843
39	0.440146	0.425632	0.492444	0.318850	0.524471	0.253577	0.581737	0.199696	0.606690	0.172112
40	1.222827	0.517327	1.224979	0.416576	1.215884	0.307904	1.166396	0.286091	1.183175	0.227665
41	1.307136	0.441733	1.520335	0.412716	1.737112	0.357261	2.081624	0.253517	1.997615	0.252606
42	1.557421	0.508500	1.504438	0.491353	1.711185	0.387495	1.614643	0.381291	1.575270	0.374285
43	1.465238	0.508519	1.637360	0.332159	1.600769	0.292824	1.619533	0.275049	1.577925	0.203884
44	2.168172	0.403723	2.306031	0.334294	2.314829	0.324785	2.350128	0.247344	2.252714	0.275388
45	2.089165	0.410730	2.233312	0.344871	2.201221	0.345701	2.342526	0.232863	2.190808	0.280163
46	1.958072	0.311409	1.751625	0.370112	1.541863	0.402026	1.574470	0.340694	1.583136	0.274754
47	1.201716	0.413261	0.838485	0.504511	0.947559	0.415843	1.080398	0.292550	1.089174	0.256535
48	1.622023	0.484820	1.411178	0.442207	1.351592	0.427862	1.114225	0.371388	1.081983	0.317387
49	1.219759	0.419048	1.691134	0.332371	1.529454	0.433831	1.578991	0.403267	1.685758	0.354046
50	1.587388	0.447303	1.791649	0.374762	1.515663	0.372337	1.380737	0.347974	1.422146	0.325368

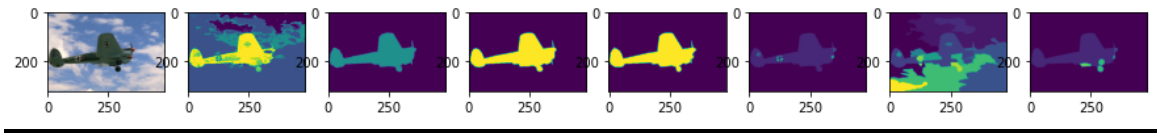
Download audio from this page

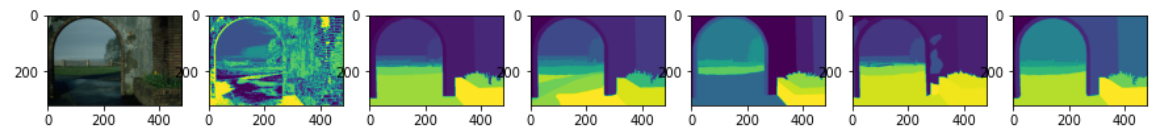
	Average evaluation per image	
validation type	AVGEntropy	AVGFmeasure
1	1.436721	0.418830
2	0.337176	0.445248
3	2.063932	0.377529
4	1.673711	0.424371
5	0.802068	0.390510
6	1.127565	0.441694
7	1.322779	0.364225
8	1.077229	0.311734
9	2.014114	0.446373
10	1.640314	0.466635
11	2.319217	0.310920
12	1.695841	0.314185
13	1.774501	0.383774
14	1.565437	0.361057
15	1.649252	0.323810
16	1.032426	0.542722
17	1.136996	0.371373
18	1.433770	0.360873
19	1.984731	0.268361
20	1.191676	0.338481
21	0.935441	0.516015
22	1.228402	0.352551
23	1.325503	0.418346
24	1.893210	0.332259
25	1.041579	0.387218
26	0.979678	0.295450
27	0.493140	0.323645
28	2.168350	0.277805
29	1.716178	0.319360
30	1.242640	0.438517
31	1.659916	0.322686
32	2.260706	0.363834
33	1.282439	0.411286
34	1.505324	0.382376
35	1.608902	0.394095
36	1.547216	0.399380
37	1.778248	0.348095
38	1.030388	0.256788
39	0.529098	0.273973
40	1.202652	0.351112
41	1.728764	0.343567
42	1.592592	0.428585
43	1.580165	0.322487
44	2.278375	0.317107
45	2.211406	0.322866
46	1.681833	0.339799
47	1.031466	0.376540
48	1.316200	0.408733
49	1.541019	0.388513
50	1.539517	0.373549

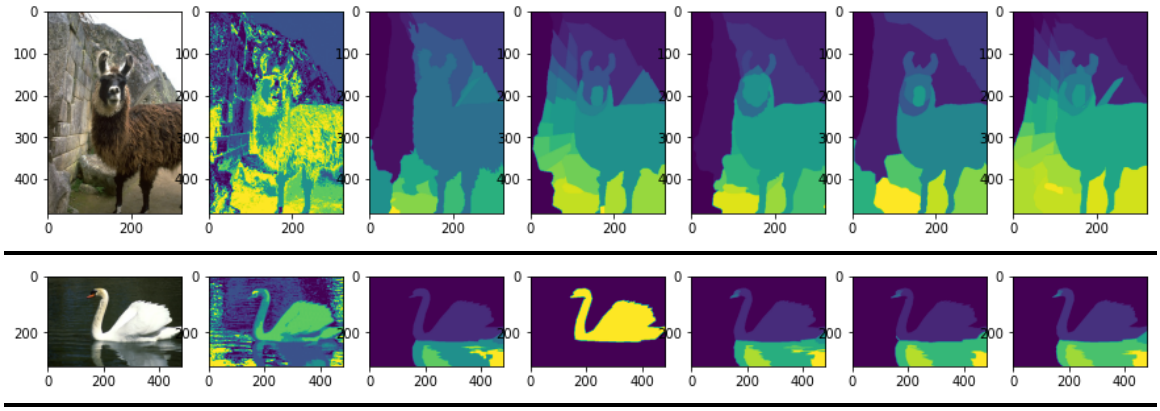
4)a)Displaying 5 images with K-means at k=5:

```
def drawK5(images,results,GT_images,mode):
  for i in range(0,5):
    if(mode==0 or mode ==1):
      num = len(GT_images[i]) + 2  # +1 for the image
      fig, ax = plt.subplots(1, num, figsize=(15,15))
      ax[0].imshow(images[i])
      if(mode==0) : ax[1].imshow(results[1][i])
      elif(mode==1) : ax[1].imshow(results[i])
      for j in range(2,num):
          ax[j].imshow(GT_images[i][j-2])
      plt.show()
    else :
      fig, ax = plt.subplots(1, 3, figsize=(15,15))
      ax[0].imshow(images[i])
      ax[1].imshow(results[1][i])
      ax[2].imshow(GT_images[i])
      ax[0].set_title('Original Image '+str(i+1))
      ax[1].set_title('Image Using K_means'+str(i+1))
      ax[2].set_title('Image Using NC'+str(i+1))
      plt.show()
```









b)Using Normalized-Cut at k=5:

before applying Ncut we resized the image to save memory in ram as the we need to calculate the dist,adj,degree,L and La matrices

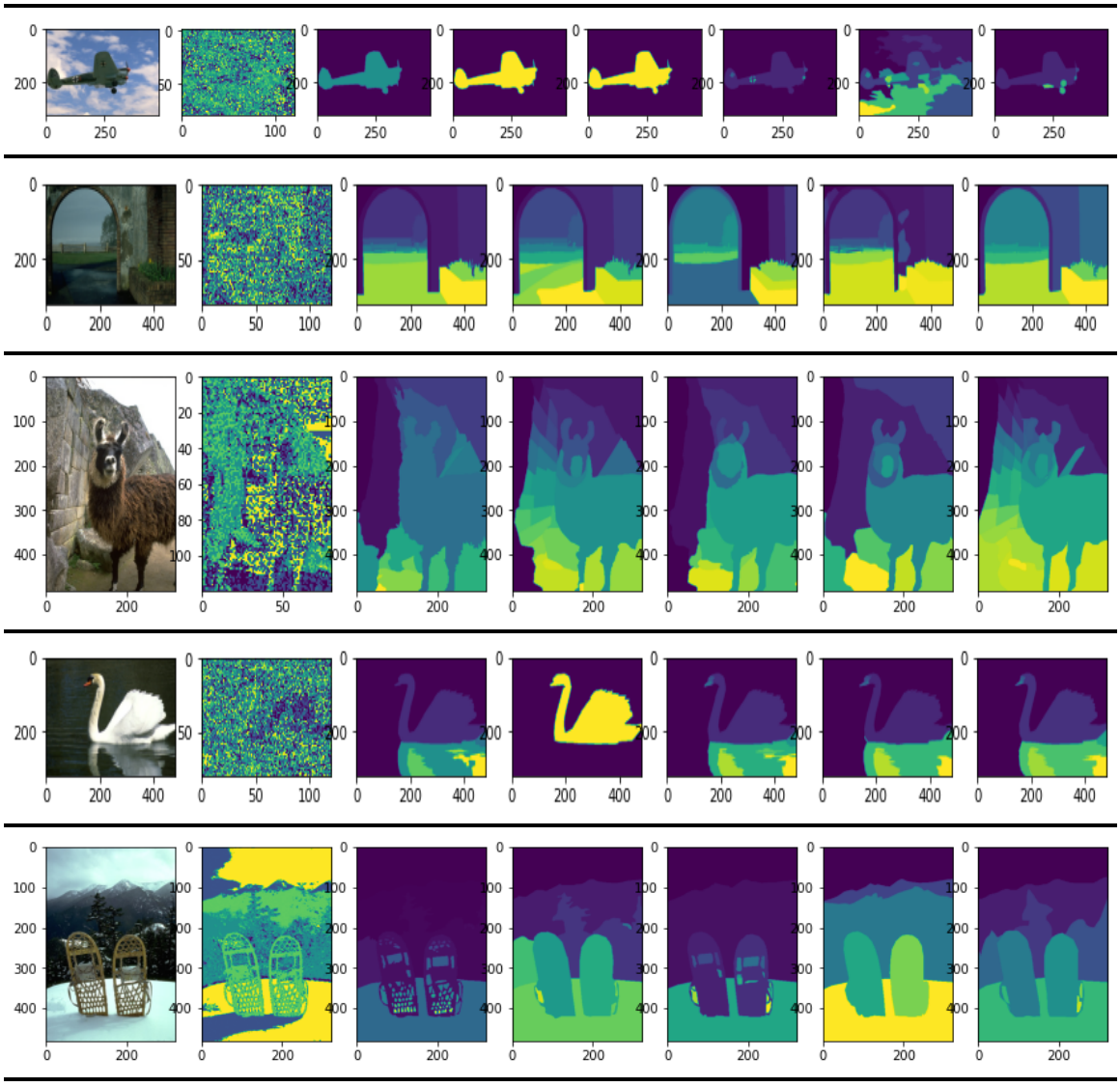
```
from sklearn.neighbors import kneighbors_graph

def ncut(Data,nn,sze):
    dist=np.zeros((sze,sze))
    for d in range(0,sze):
        for c in range(0,sze):
          dist[d,c]=np.linalg.norm(Data[d]-Data[c])
    Adj = kneighbors_graph( dist , nn , mode='connectivity', include_self=False).toarray()
    Degree = np.diag(np.sum(Adj, axis=1))
    L = Degree - Adj
    La = np.dot(np.linalg.inv(Degree),L)
    eigenvalues,eigenvectors=np.linalg.eigh(La)
    idx = eigenvalues.argsort()[::1]
    arreigenvalues = eigenvalues[idx]
    arreigenvectors = eigenvectors[:,idx]
    return arreigenvalues,arreigenvectors
```

```
import cv2

# eigval = []
# eigvec = []
resized_data = []
for im in range(0,5):
  scale = 0.25
  image = images[im]
  width = int(image.shape[1] * scale)
  height = int(image.shape[0] * scale)
  dsize = (width, height)
  resized_data.append(cv2.resize(np.copy(image), dsize))
  # x,y = ncut(resized_data[im].reshape((-1,3)),5,width*height)
  # eigval.append(x)
  # eigvec.append(y)

# np.save('/content/drive/MyDrive/imageSegmentation/eigval',eigval)
# np.save('/content/drive/MyDrive/imageSegmentation/eigvec',eigvec)
```

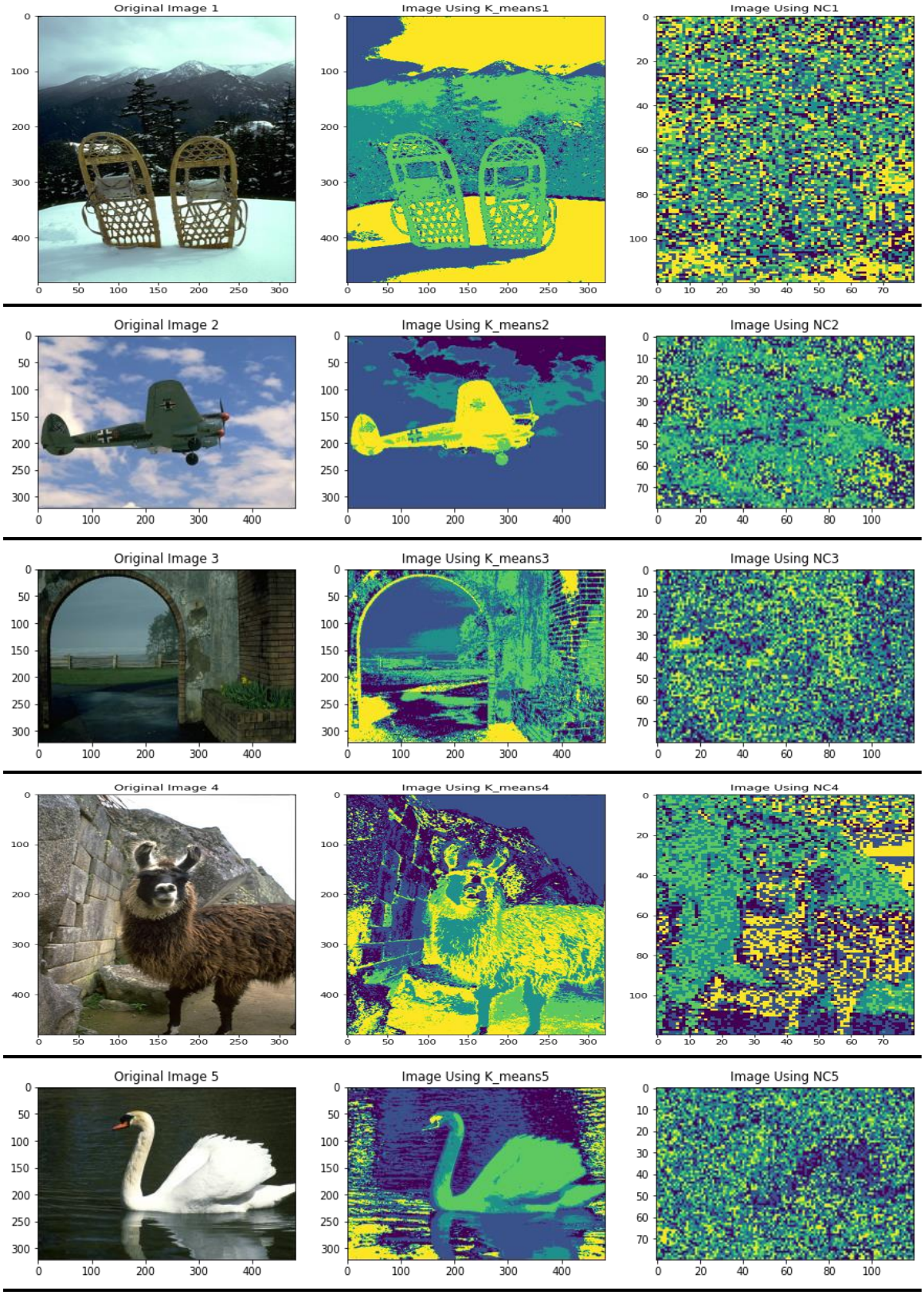
```
eigval=np.load('/content/drive/MyDrive/imageSegmentation/eigval.npy')
eigvec=np.load('/content/drive/MyDrive/imageSegmentation/eigvec.npy')
```

```
U=[]
Y=[]
for j in range(0,5):
  for i in range(0,len(eigval[0])): #0-> 513 ,
    if(eigval[j][i]>0) : break
  U.append(eigvec[j][:,i:i+5])
for j in range(0,5):
  cc=np.zeros((9600,5))
  for i in range(0,9600):
    if(np.linalg.norm(U[j][i]) != 0) : cc[i] = U[j][i]/np.linalg.norm(U[j][i])
  Y.append(cc)
```

```
NCR=[]
for i in range(0,5):
  kmean = KMeans(n_clusters=5,random_state=0).fit(Y[i])
  NCR.append(kmean.labels_.reshape(resized_data[i].shape[:-1]))
```



c)Comparing between k-means and NC:

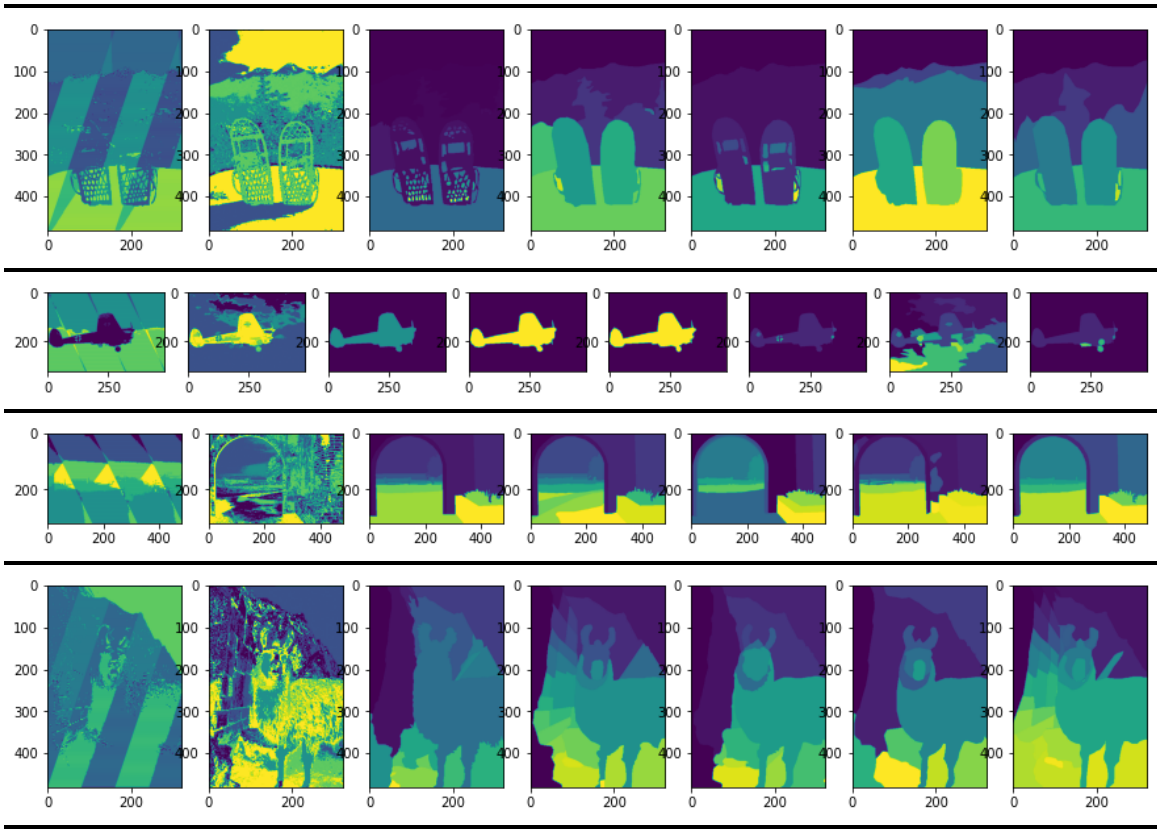


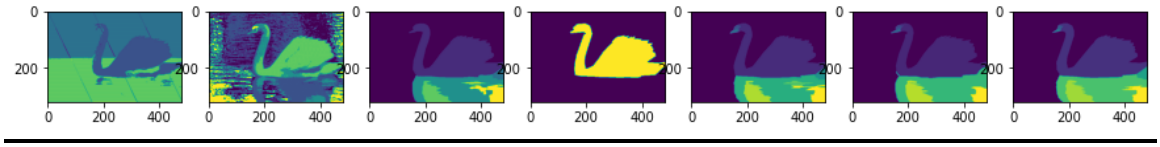
5)Modifying K-means by adding 2 features:

First Trail:

We added 2 new columns the value of x,y coordinate for every pixel so every pixel now have 5 attributes R,G,B and X,Y so the similarity between each pixels have the same color and are adjacent to each other will increase, but for some image it will give noise as it may increase similarity between 2 adjacent pixels having different colors and it may decrease similarity between 2 pixels having same color but far from each other in the xy coordinate.

```
def SpatialLayout(images,k):
  resultsImp = []
  for i in range(0,5):
    a = images[i].shape[0]
    b = images[i].shape[1]
    x = np.arange(a)
    y = np.arange(b)
    x, y = np.meshgrid(x,y)
    image5F = np.concatenate((images[i],x.reshape(a,b,1), y.reshape(a,b,1)), axis=2)
    x=image5F.reshape((-1,5))
    kmean = KMeans(n_clusters=k,random_state=0).fit(x)
    resultsImp.append(kmean.labels_.reshape(images[i].shape[:-1]))
  return resultsImp
```

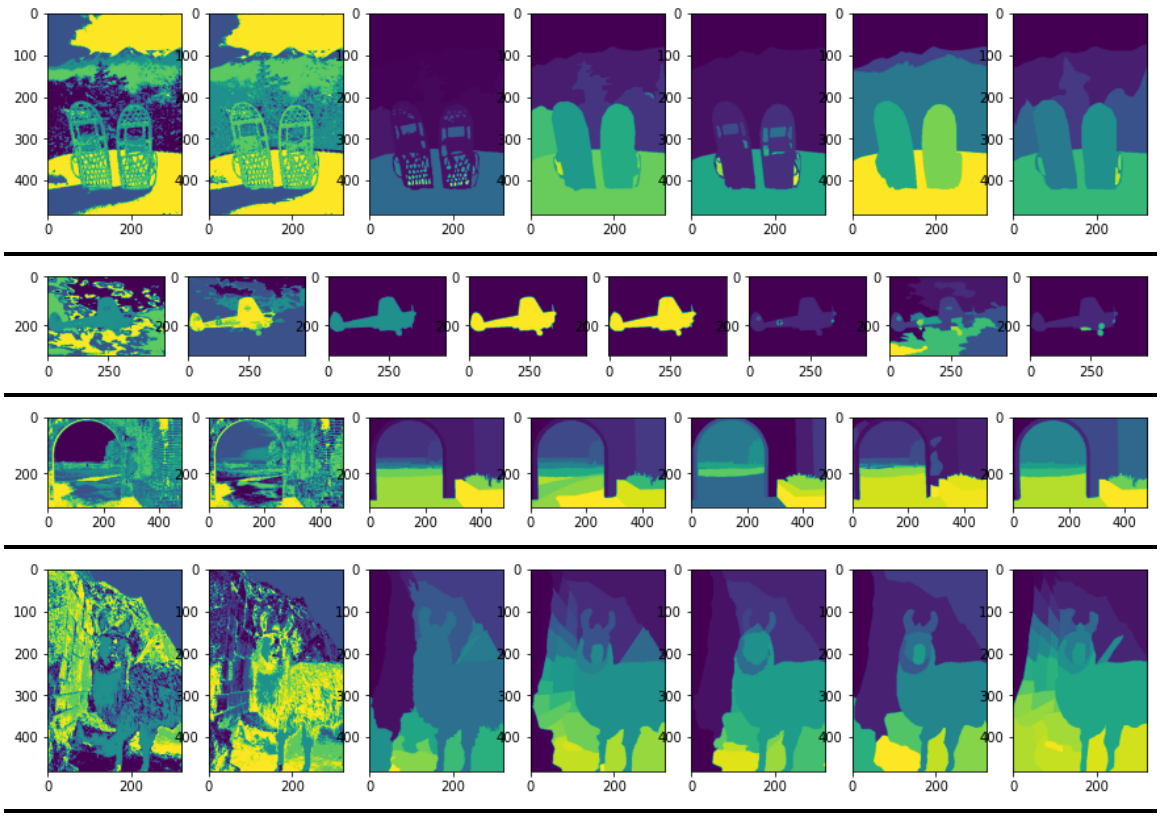


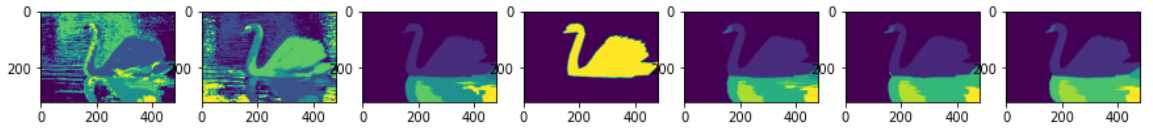


Second Trail:

here we applied another trail to modify kmeans by passing it the old value of kmeans so it will learn from its old result and also another feature we expected that it may decrease time spend in updating centroids

```
def SpatialLayoutTrail(images,k,results):
  resultsImp = []
  for i in range(0,5):
    a = images[i].shape[0]
    b = images[i].shape[1]
    y=images[i].reshape((-1,3))
    kmeany = KMeans(n_clusters=k,random_state=0).fit(y)
    image5F = np.concatenate((images[i],kmeany.labels_.reshape((a,b,1)), results[1][i].reshape((a,b,1))), axis=2)
    x=image5F.reshape((-1,5))
    kmean = KMeans(n_clusters=k,random_state=0).fit(x)
    resultsImp.append(kmean.labels_.reshape(images[i].shape[:-1]))
  return resultsImp
```





Source code
https://colab.research.google.com/drive/19vKAhl04LUNfmlo2VEwXoXA9YJtWVV_G
displaying results in table:
https://colab.research.google.com/drive/1pyFd-Rvk-5gVXWaBz1iY8lRCENzC8Hh8?usp=sharing