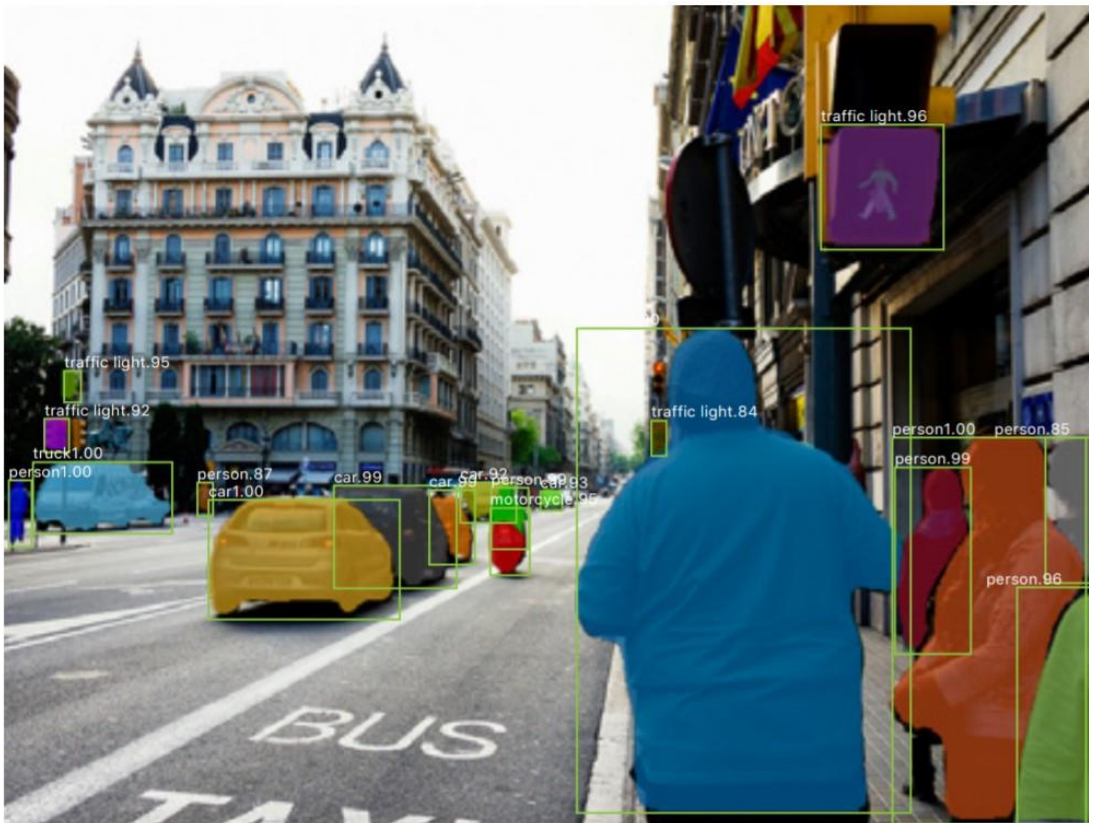
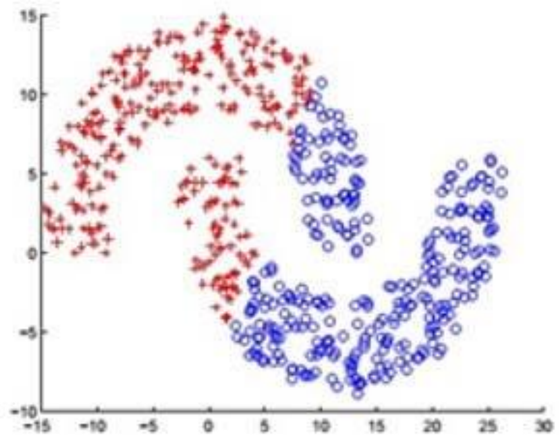


Image Segmentation Project

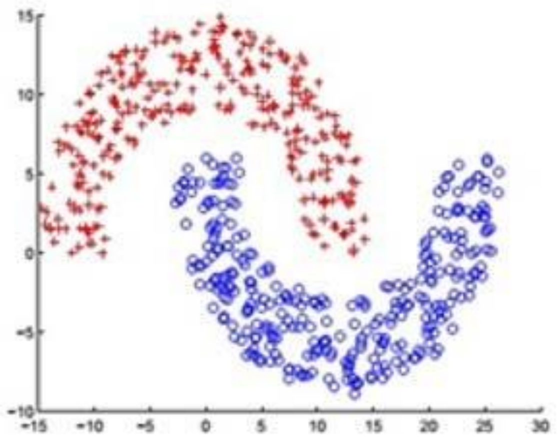
Name : Hussen Adel	5783
: Mohamed Adel	4982
: Ahmed Elsayed	5692
: Ahmed Aly	4808

Problem Statement:

Main idea is to segment images, Image segmentation means that we can group similar pixels together and give these grouped pixels the same label. The grouping problem is a clustering problem. We want to study the use of K-means and Spectral Clustering on the Berkeley Segmentation Benchmark.



(a) K-means



(b) Spectral Clustering

Solution Phases:

1-Dataset format.

2-Image and Ground truth Visualization.

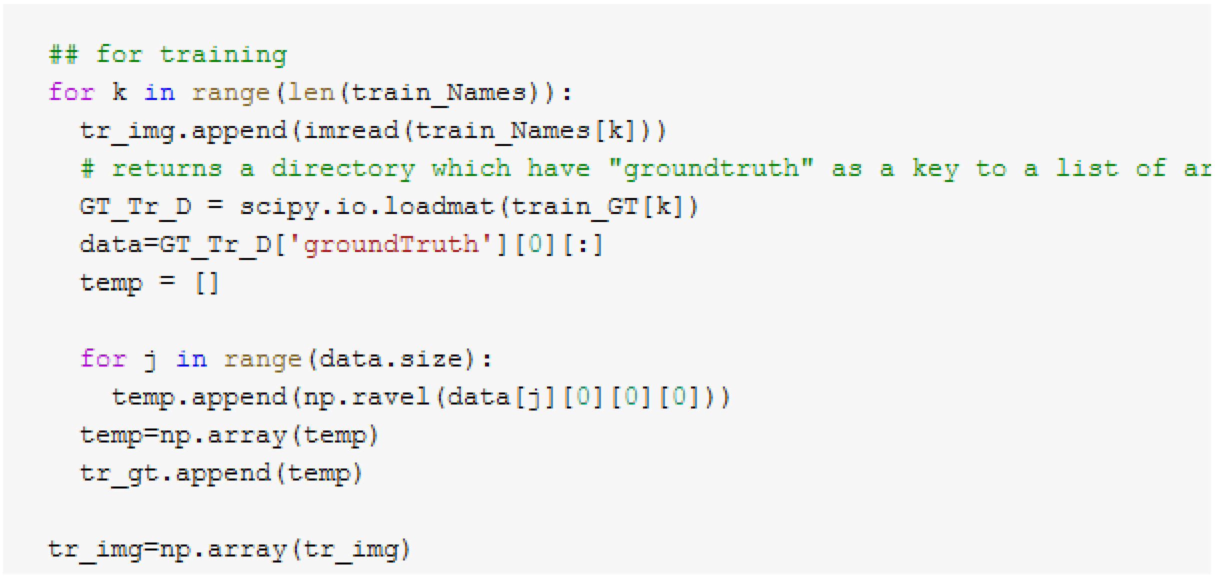
3-Segmentation Using K-means.

4-Big Picture.

5-Encoding the Spatial Layout of the pixels.

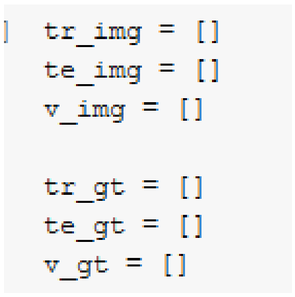
1-Dataset Format:

The dataset consists of 500 natural images, ground-truth human annotations and benchmarking code. The data is explicitly separated into disjoint train, validation, and test subsets. The test set is 200 images only. We will report our results on the first 50 images of the test set only. Each image is either 481 by 321 or 321 by 481. For each image exists a set of ground-truths that varies in their k values.



```
## for training
for k in range(len(train_Names)):
  tr_img.append(imread(train_Names[k]))
  # returns a directory which have "groundtruth" as a key to a list of ar
  GT_Tr_D = scipy.io.loadmat(train_GT[k])
  data=GT_Tr_D['groundTruth'][0][:]
  temp = []

  for j in range(data.size):
    temp.append(np.ravel(data[j][0][0][0]))
  temp=np.array(temp)
  tr_gt.append(temp)

tr_img=np.array(tr_img)
```

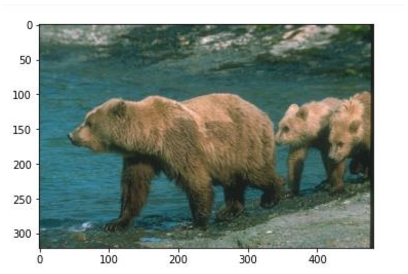
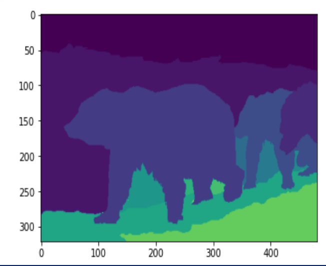
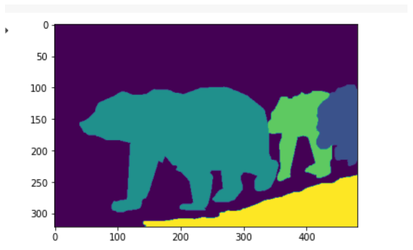
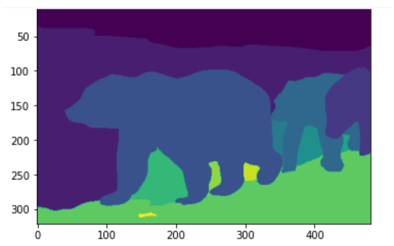
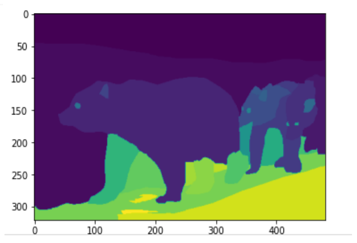
Each set of images and ground-truths are put in a NumPy array.

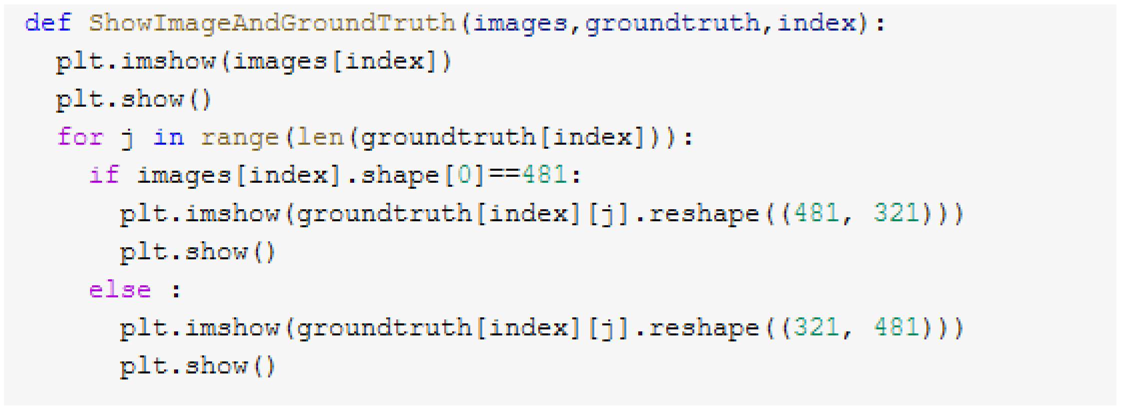


```
tr_img = []
te_img = []
v_img = []

tr_gt = []
te_gt = []
v_gt = []
```

2-Image and Ground truth Visualization:

Images and their corresponding ground truths.

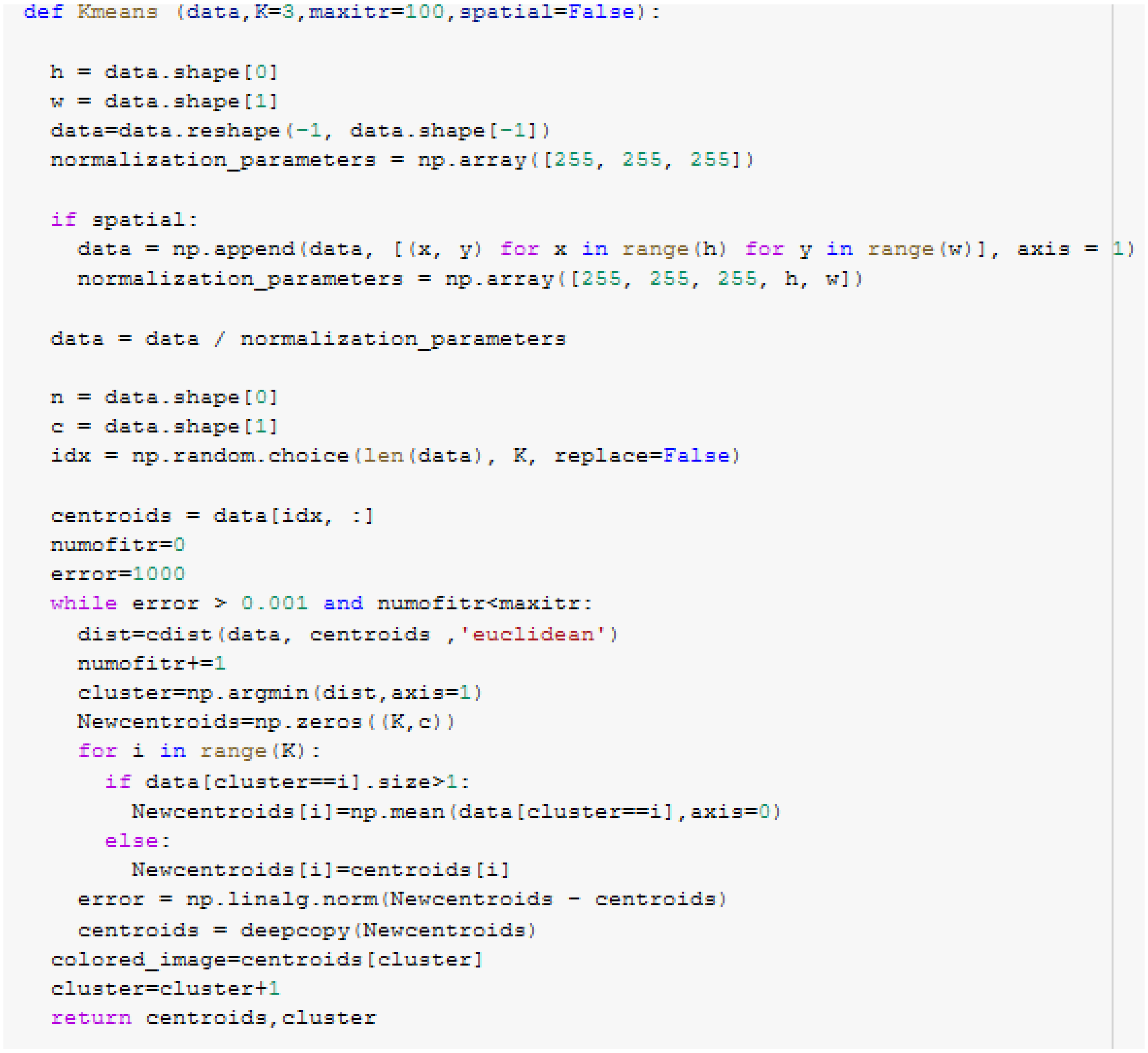


```
def ShowImageAndGroundTruth(images,groundtruth,index):
  plt.imshow(images[index])
  plt.show()
  for j in range(len(groundtruth[index])):
    if images[index].shape[0]==481:
      plt.imshow(groundtruth[index][j].reshape((481, 321)))
      plt.show()
    else :
      plt.imshow(groundtruth[index][j].reshape((321, 481)))
      plt.show()
```

3-Segmentation using k-means:

a. K-means implementation.
b. Segmentation.
c. Evaluation using F-score and Conditional Entropy.
d. Good and Bad results.

a) K-means implementation.

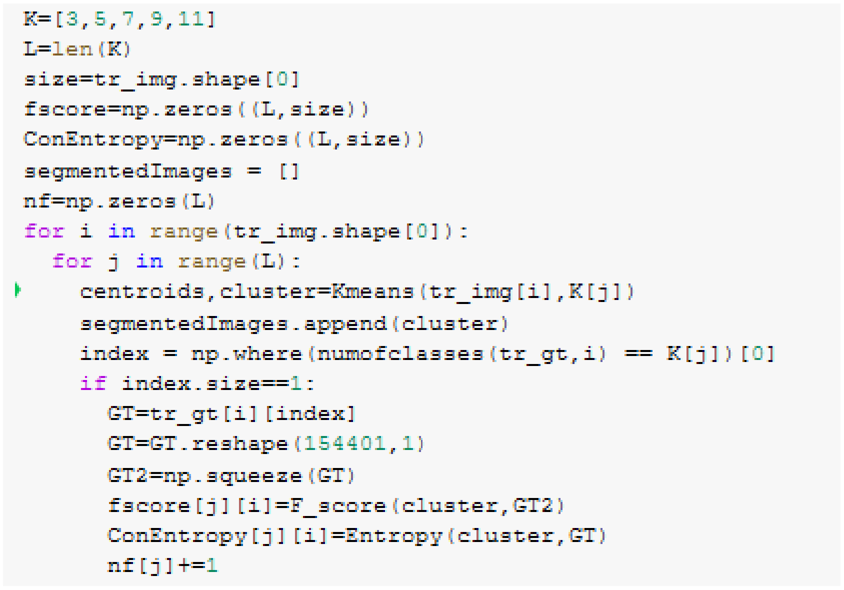


```
def Kmeans (data,K=3,maxitr=100,spatial=False):

  h = data.shape[0]
  w = data.shape[1]
  data=data.reshape(-1, data.shape[-1])
  normalization_parameters = np.array([255, 255, 255])

  if spatial:
    data = np.append(data, [(x, y) for x in range(h) for y in range(w)], axis = 1)
    normalization_parameters = np.array([255, 255, 255, h, w])

  data = data / normalization_parameters

  n = data.shape[0]
  c = data.shape[1]
  idx = np.random.choice(len(data), K, replace=False)

  centroids = data[idx, :]
  numofitr=0
  error=1000
  while error > 0.001 and numofitr<maxitr:
    dist=cdist(data, centroids ,'euclidean')
    numofitr+=1
    cluster=np.argmin(dist,axis=1)
    Newcentroids=np.zeros((K,c))
    for i in range(K):
      if data[cluster==i].size>1:
        Newcentroids[i]=np.mean(data[cluster==i],axis=0)
      else:
        Newcentroids[i]=centroids[i]
    error = np.linalg.norm(Newcentroids - centroids)
    centroids = deepcopy(Newcentroids)
  colored_image=centroids[cluster]
  cluster=cluster+1
  return centroids,cluster
```

The implementation is engulfed in the Kmeans() function.

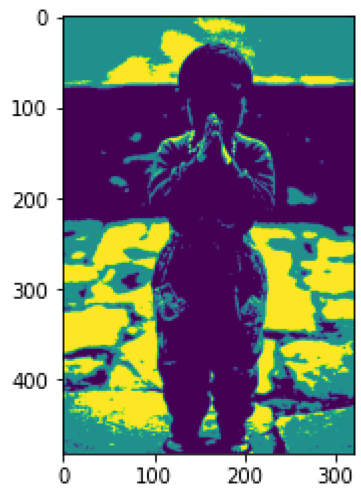
1) At first the image is reshaped from (481, 321, 3) to (154401, 3)
2) At first Centroids are randomly chosen from the image array depending on the number of k (k,3)
3) The New Centroids and the clusters are calculated in the while loop until either the error becomes smaller than 0.001 or the max number of iterations are reached.

b) Segmentation.

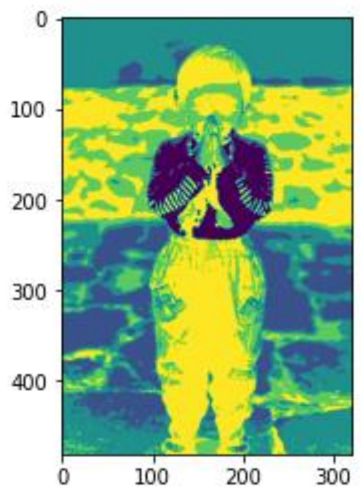


```
K=[3,5,7,9,11]
L=len(K)
size=tr_img.shape[0]
fscore=np.zeros((L,size))
ConEntropy=np.zeros((L,size))
segmentedImages = []
nf=np.zeros(L)
for i in range(tr_img.shape[0]):
  for j in range(L):
    centroids,cluster=Kmeans(tr_img[i],K[j])
    segmentedImages.append(cluster)
    index = np.where(numofclasses(tr_gt,i) == K[j])[0]
    if index.size==1:
      GT=tr_gt[i][index]
      GT=GT.reshape(154401,1)
      GT2=np.squeeze(GT)
      fscore[j][i]=F_score(cluster,GT2)
      ConEntropy[j][i]=Entropy(cluster,GT)
      nf[j]+=1
```

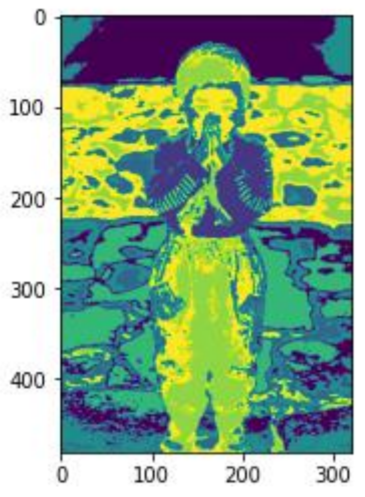
1) Images in the training set is segmented this is done for each value ok K.
2) Both F-score and entropy are calculated.



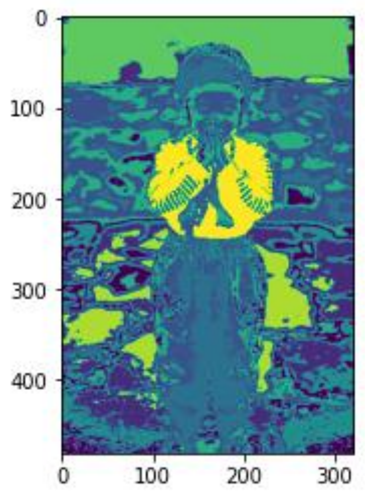
K=3



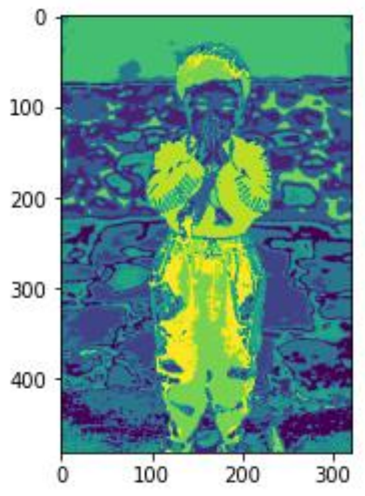
k=5


0
100
200
300
400
0
100
200
300

K=7

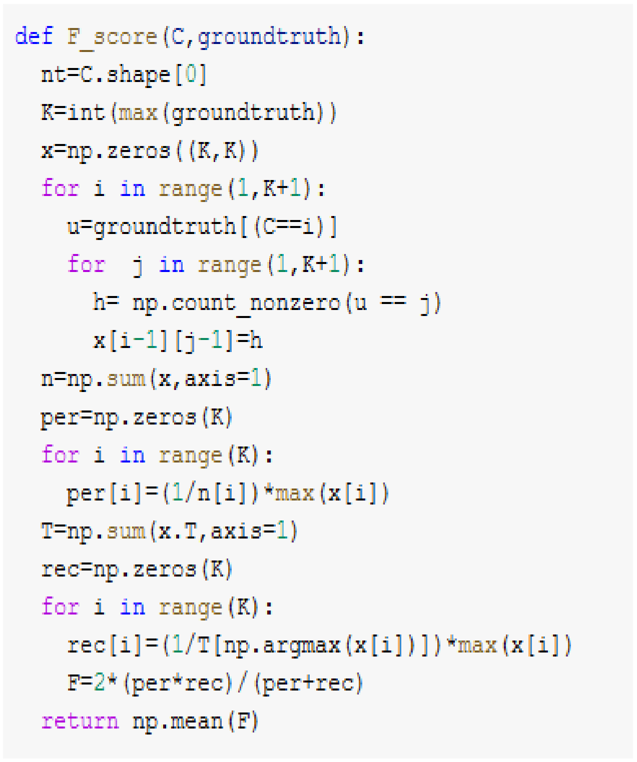

0
100
200
300
400
0
100
200
300

K=9

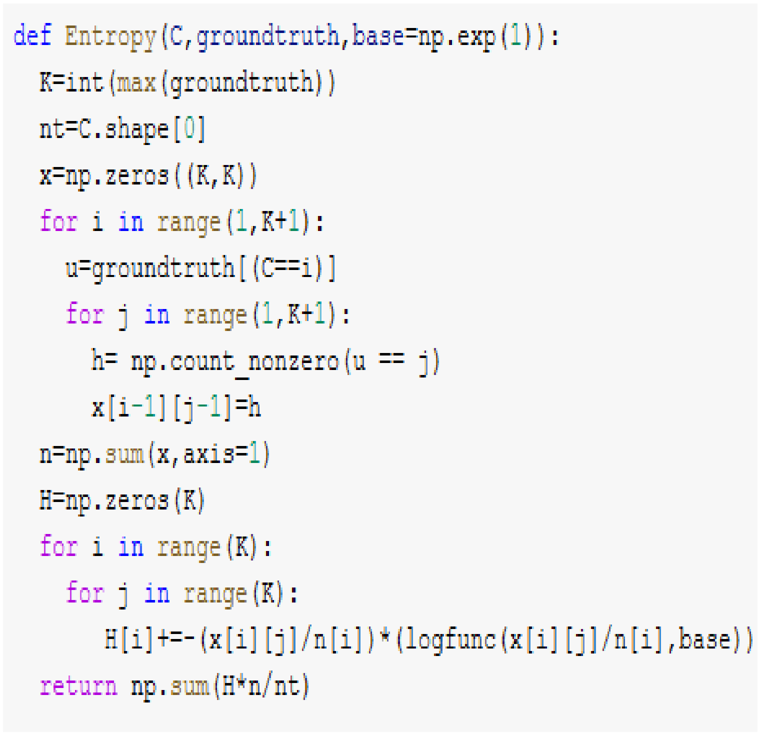

0
100
200
300
400
0
100
200
300

K=11

c) F-Measure & Entropy.



```python
def F_score(C,groundtruth):
  nt=C.shape[0]
  K=int(max(groundtruth))
  x=np.zeros((K,K))
  for i in range(1,K+1):
    u=groundtruth[(C==i)]
    for  j in range(1,K+1):
      h= np.count_nonzero(u == j)
      x[i-1][j-1]=h
  n=np.sum(x,axis=1)
  per=np.zeros(K)
  for i in range(K):
    per[i]=(1/n[i])*max(x[i])
  T=np.sum(x.T,axis=1)
  rec=np.zeros(K)
  for i in range(K):
    rec[i]=(1/T[np.argmax(x[i])])*max(x[i])
    F=2*(per*rec)/(per+rec)
  return np.mean(F)
```

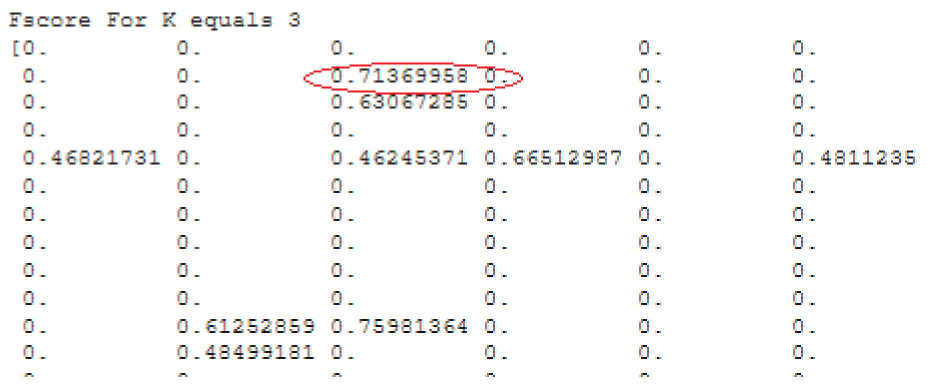


```python
def Entropy(C,groundtruth,base=np.exp(1)):
  K=int(max(groundtruth))
  nt=C.shape[0]
  x=np.zeros((K,K))
  for i in range(1,K+1):
    u=groundtruth[(C==i)]
    for j in range(1,K+1):
      h= np.count_nonzero(u == j)
      x[i-1][j-1]=h
  n=np.sum(x,axis=1)
  H=np.zeros(K)
  for i in range(K):
    for j in range(K):
       H[i]+=-(x[i][j]/n[i])*(logfunc(x[i][j]/n[i],base))
  return np.sum(H*n/nt)
```

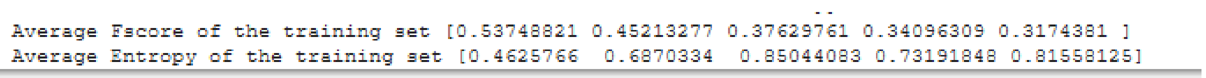
Training

F-score and Entropy are calculated for all values of K.

In the training set F-score was highest when k = 3



```
Fscore For K equals 3
[0.         0.         0.         0.         0.         0.
 0.         0.         0.71369958 0.         0.         0.
 0.         0.         0.63067285 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.46821731 0.         0.46245371 0.66512987 0.         0.4811235
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.61252859 0.75981364 0.         0.         0.
 0.         0.48499181 0.         0.         0.         0.
```



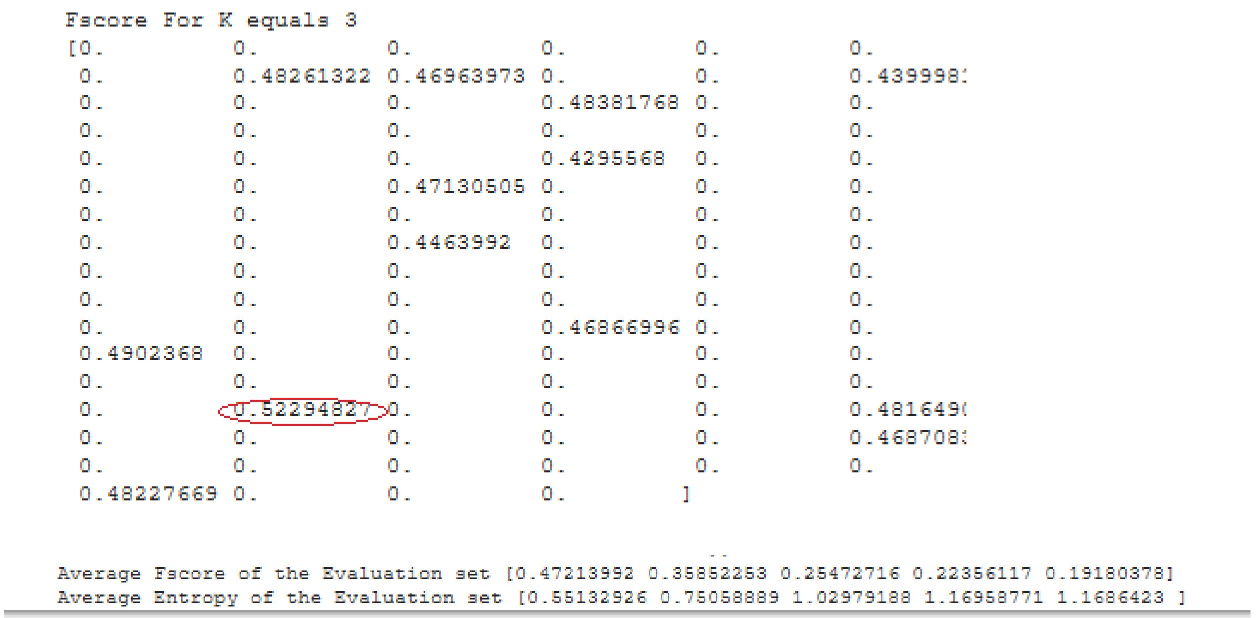
```
Average Fscore of the training set [0.53748821 0.45213277 0.37629761 0.34096309 0.3174381 ]
Average Entropy of the training set [0.4625766  0.6870334  0.85044083 0.73191848 0.81558125]
```

As we can see highest Average F-score was present when K=3

And the lowest Average Entropy was present when K=3 we deduce that K=3 is the best value for that segmentation.

Validation

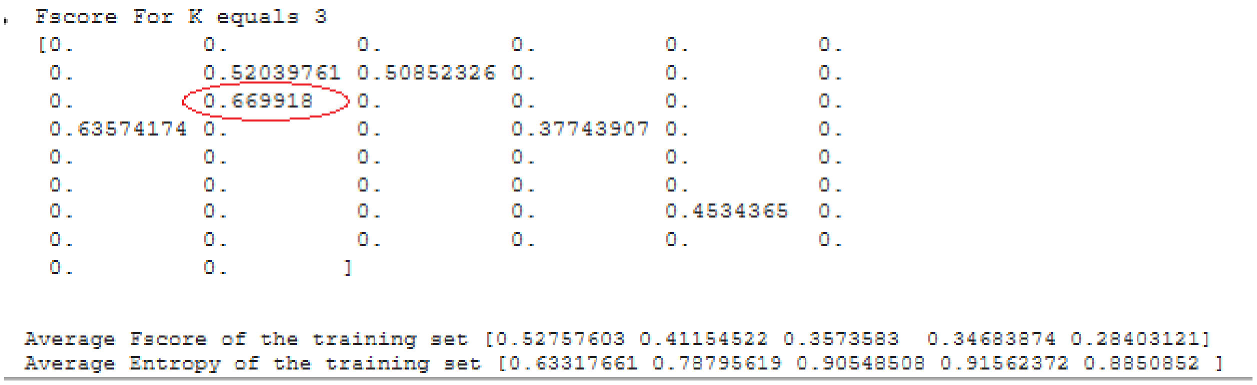
F-score and Entropy were calculated for the validation set and the results are at best when k = 3



```
Fscore For K equals 3
[0.         0.         0.         0.         0.         0.
 0.         0.48261322 0.46963973 0.         0.         0.439998
 0.         0.         0.         0.48381768 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.4295568  0.         0.
 0.         0.         0.47130505 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.4463992  0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.46866996 0.         0.
 0.4902368  0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.52294827 0.         0.         0.         0.481649
 0.         0.         0.         0.         0.         0.468708
 0.         0.         0.         0.         0.         0.
 0.48227669 0.         0.         0.         ]

Average Fscore of the Evaluation set [0.47213992 0.35852253 0.25472716 0.22356117 0.19180378]
Average Entropy of the Evaluation set [0.55132926 0.75058889 1.02979188 1.16958771 1.1686423 ]
```

Testing

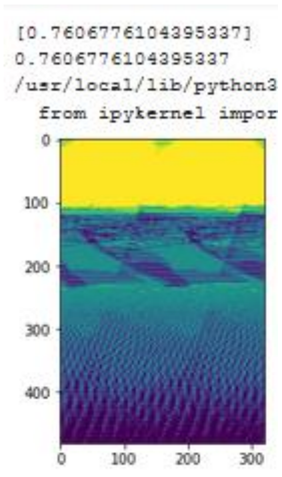
F-score and Entropy were calculated for the first 50 images in the testing set and the results are at best when k = 3



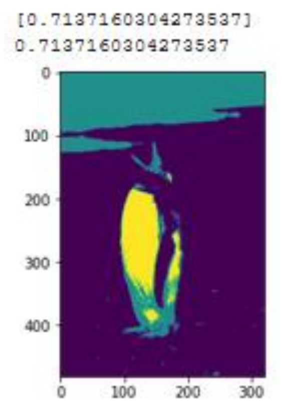
```
Fscore For K equals 3
[0.         0.         0.         0.         0.         0.
 0.         0.52039761 0.50852326 0.         0.         0.
 0.         0.669918   0.         0.         0.         0.
 0.63574174 0.         0.         0.37743907 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.4534365  0.
 0.         0.         0.         0.         0.         0.
 0.         0.         ]

Average Fscore of the training set [0.52757603 0.41154522 0.3573583  0.34683874 0.28403121]
Average Entropy of the training set [0.63317661 0.78795619 0.90548508 0.91562372 0.8850852 ]
```

d) Good results:

K=3

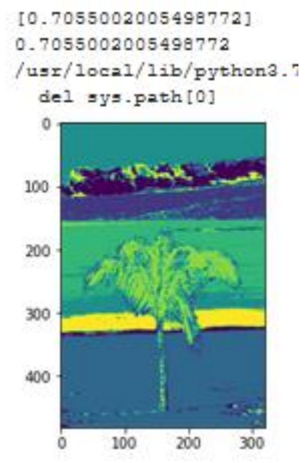


```
[0.7606776104395337]
0.7606776104395337
/usr/local/lib/python3
  from ipykernel impor
```



```
[0.7137160304273537]
0.7137160304273537
```

K=7

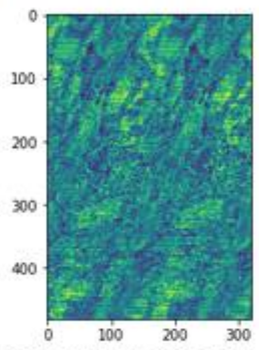


```
[0.7055002005498772]
0.7055002005498772
/usr/local/lib/python3.7
  del sys.path[0]
```

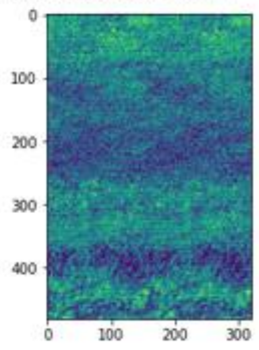
d) Bad results:

At k = 11

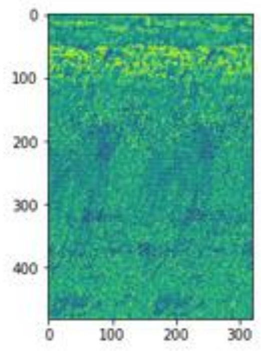
```
[0.20420925649823188]
0.20420925649823188
```



```
[0.18897548661676677]
0.18897548661676677
```



```
[0.21071613521366686]
0.21071613521366686
```

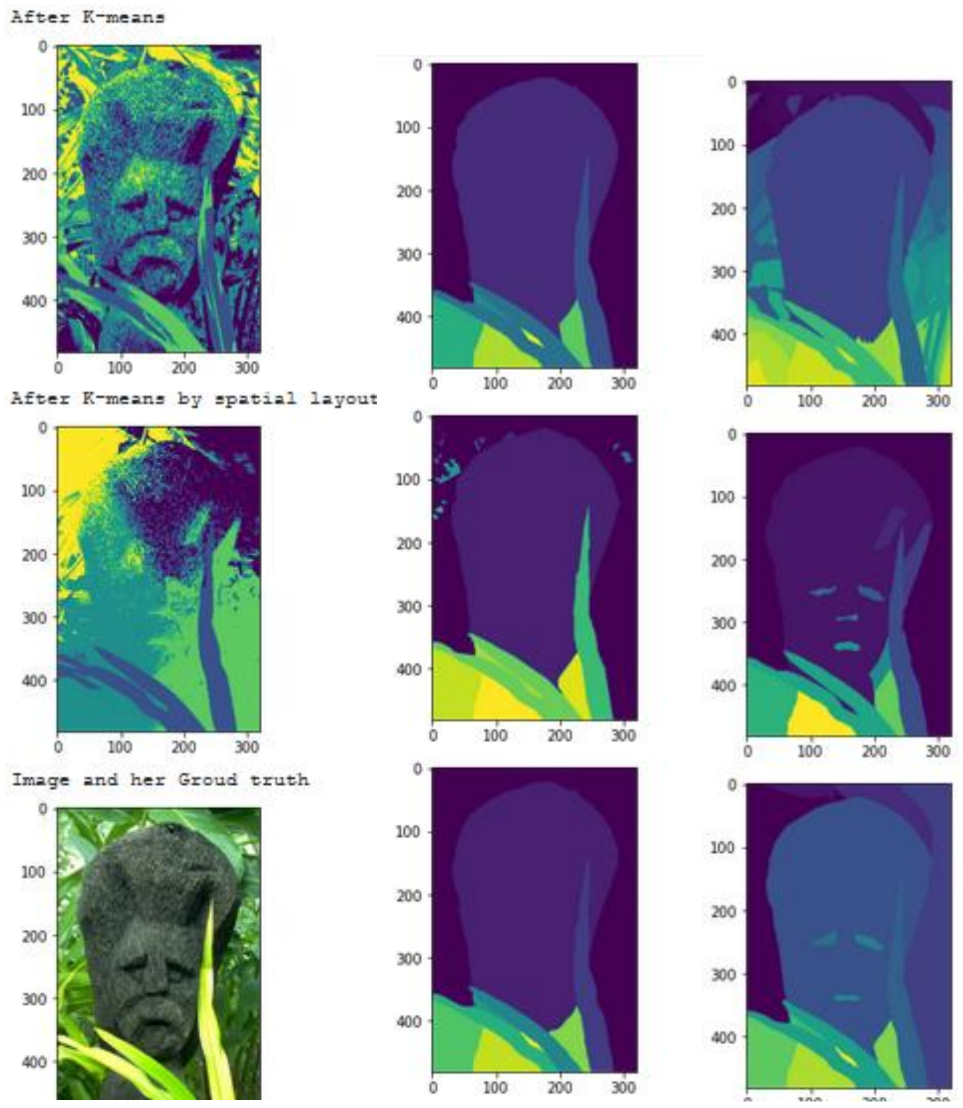


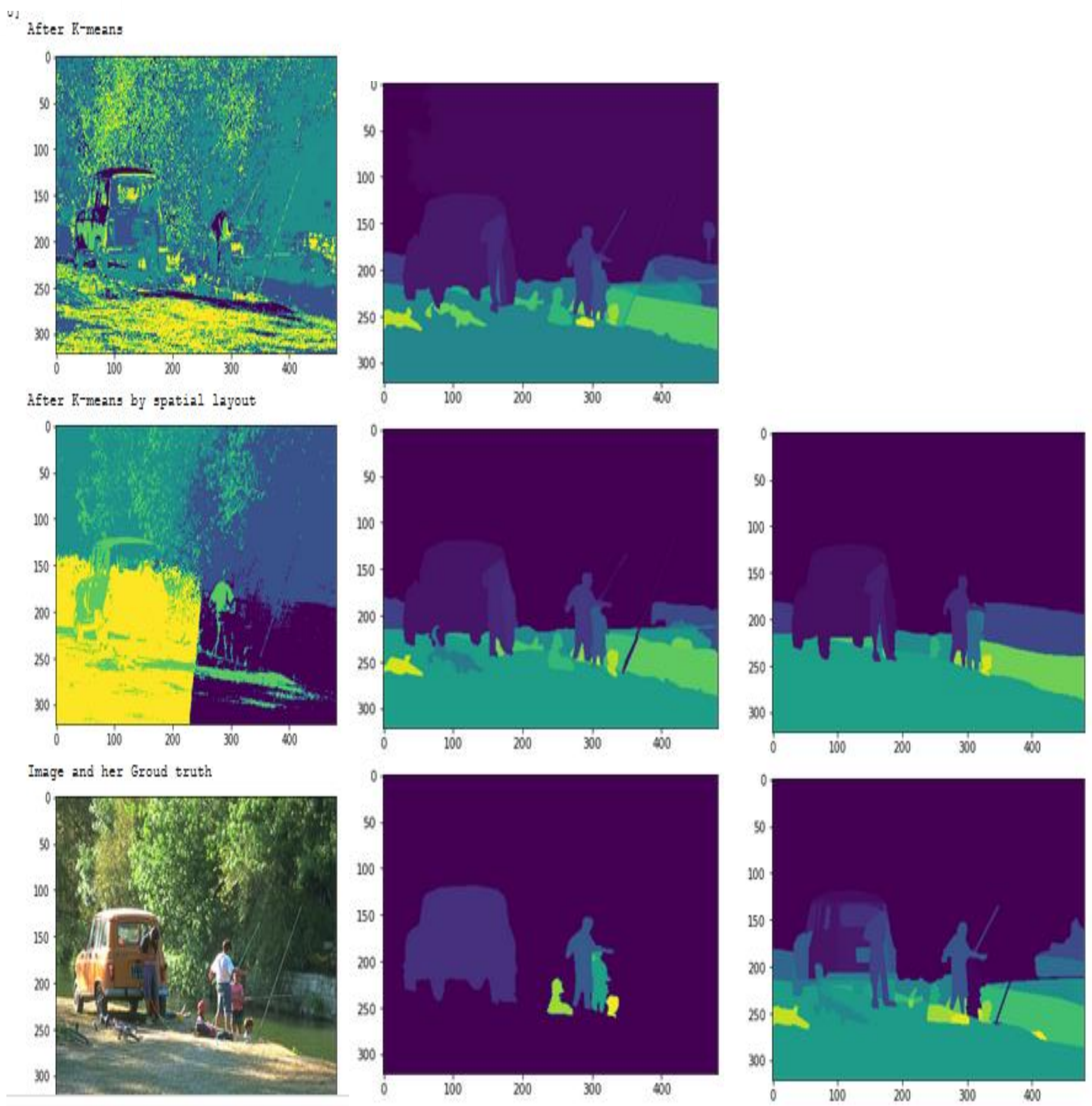
We notice that the lower the number of K is the better the segmentation of the image and the higher the F-score value. One of the reasons that led to that conclusion is that the lower the color degree variance exists in the image is harder to be segmented. But with high contrast difference the segmentation is better.

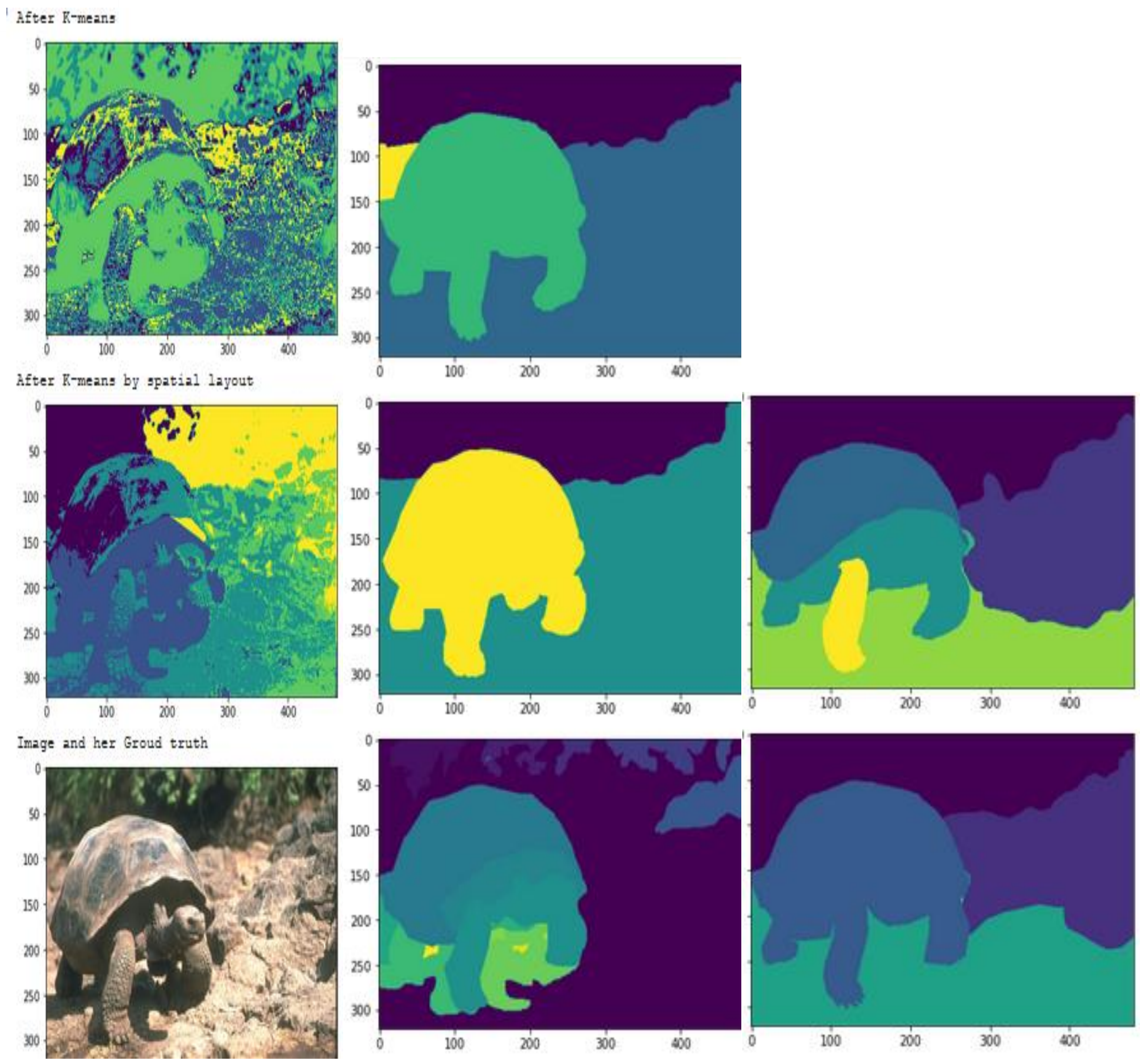
4-Big picture:

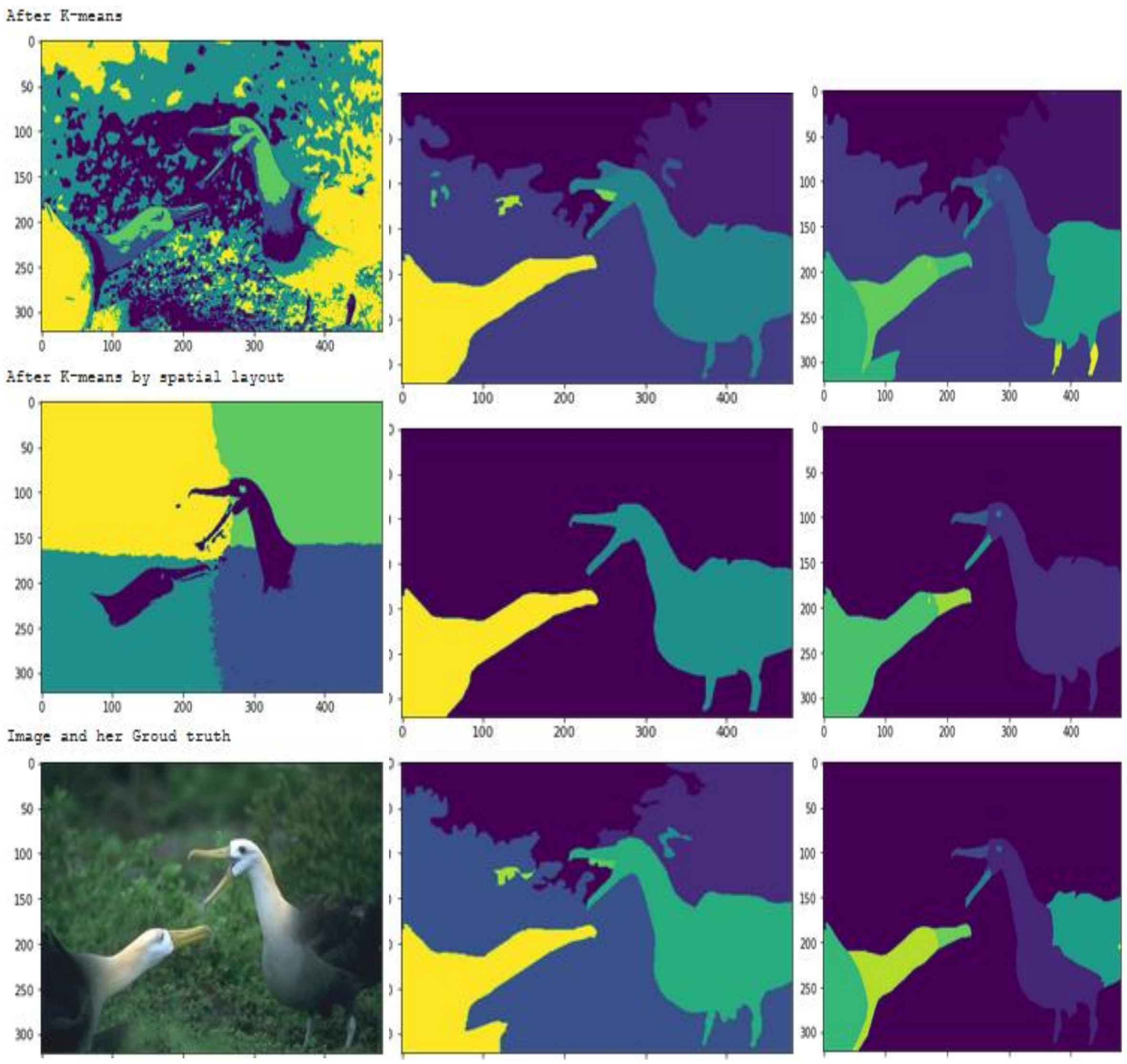
a) K-means.
b) Normalized cut.
c) K-means Vs Normalized cut.

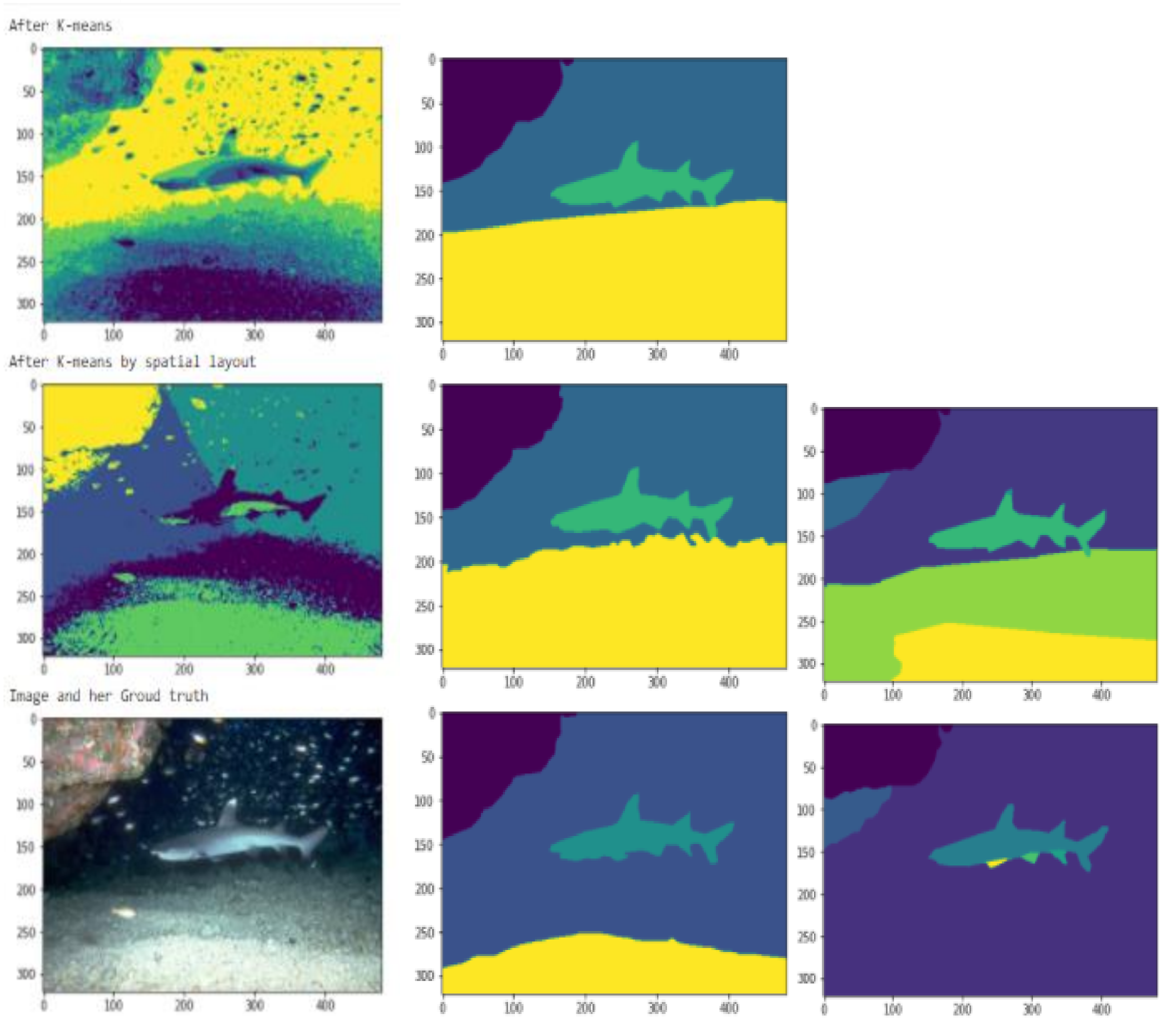
a) K-means:


After K-means
After K-means by spatial layout
Image and her Groud truth


After K-means
After K-means by spatial layout
Image and her Groud truth


After K-means
After K-means by spatial layout
Image and her Groud truth
0
50
100
150
200
250
300
400

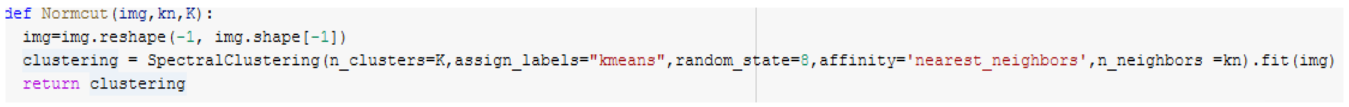

After K-means
After K-means by spatial layout
Image and her Groud truth


After K-means
After K-means by spatial layout
Image and her Groud truth

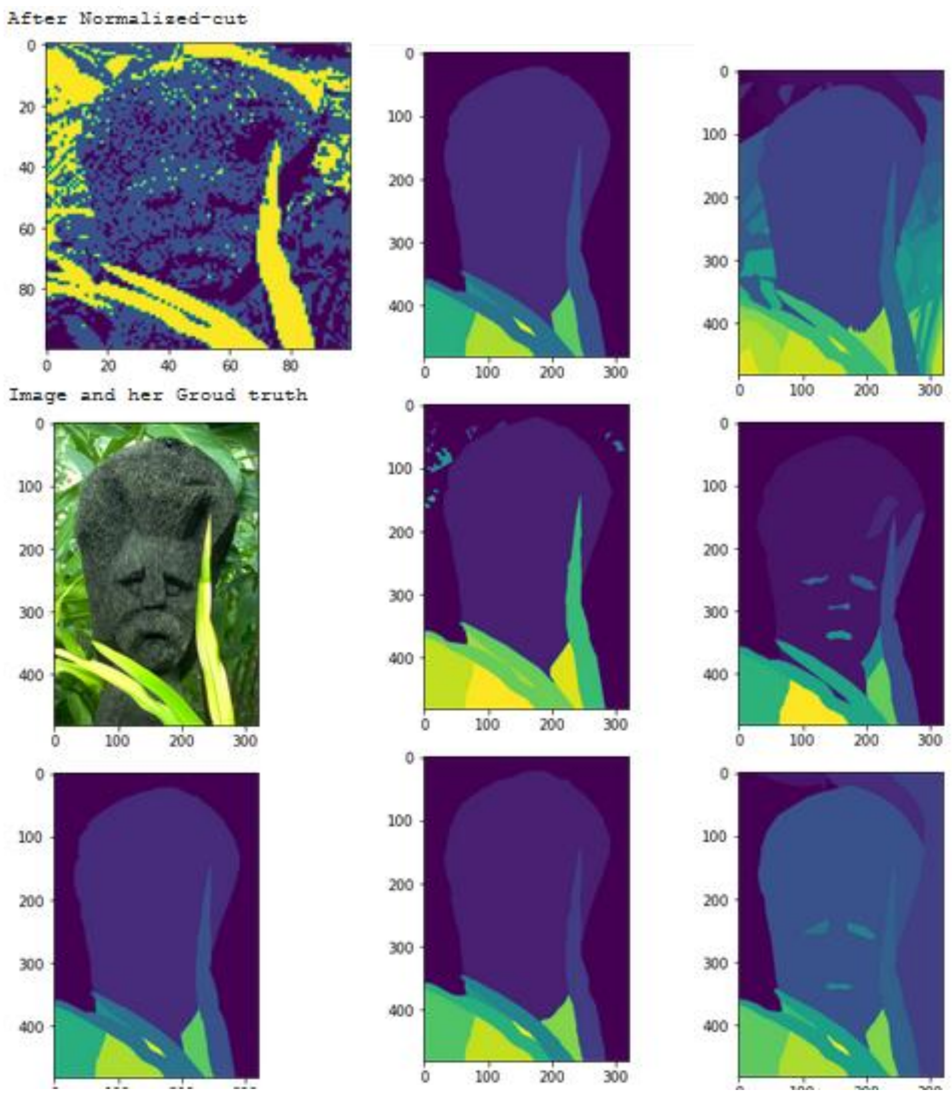
b) Normalized cut:

Due to high ram usage, we have changed the resolution of the training images from (321,481,3) to (100,100,3).

```
index=np.random.randint(200)
index=5
img=deepcopy(te_img[index])
img=cv2.resize(img, (100,100))
clustering=Normcut(img,5,5)
plt.imshow(clustering.labels_.reshape((100, 100)))
print("After Normalized-cut")
plt.show()
print("Image and her Groud truth")
ShowImageAndGroundTruth(te_img,te_gt,index)
```

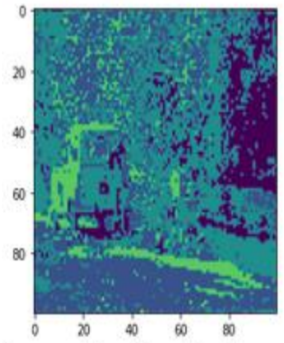


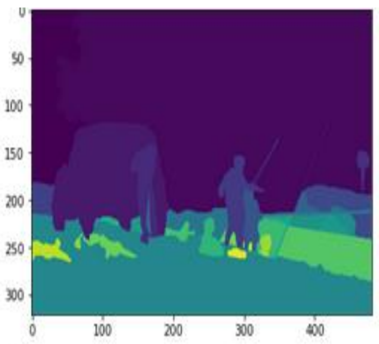
```
def Normcut(img,kn,K):
  img=img.reshape(-1, img.shape[-1])
  clustering = SpectralClustering(n_clusters=K,assign_labels="kmeans",random_state=8,affinity='nearest_neighbors',n_neighbors =kn).fit(img)
  return clustering
```



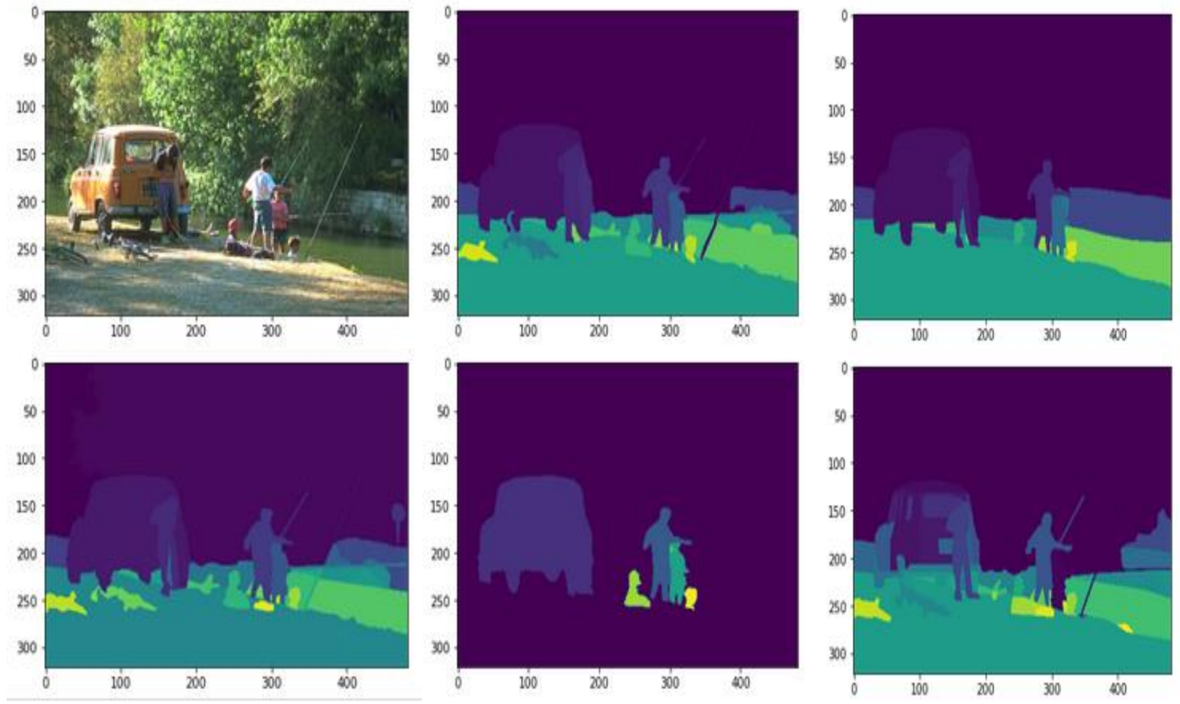
```
After Normalized-cut
Image and her Groud truth
```

```
After Normalized-cut
```

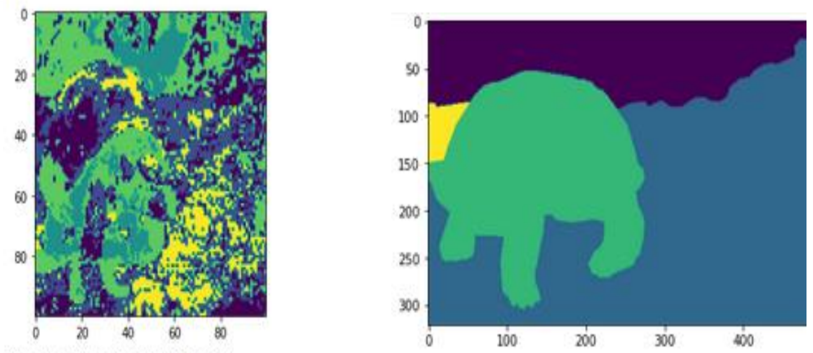




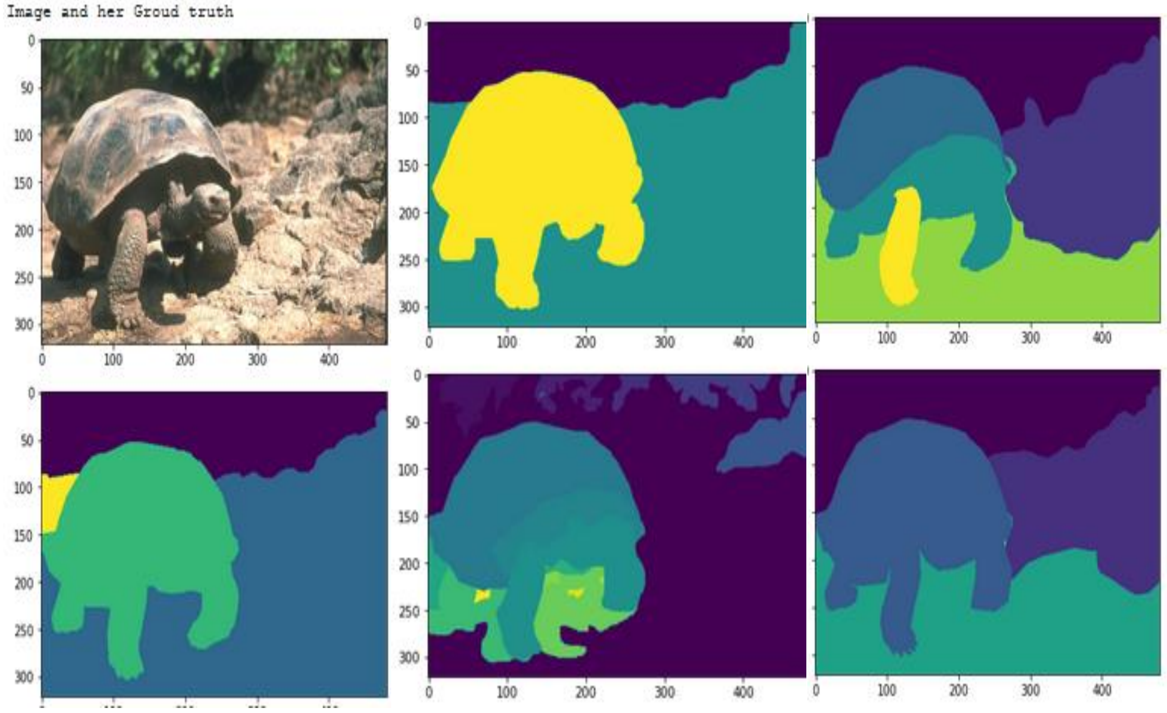
```
Image and her Groud truth
```

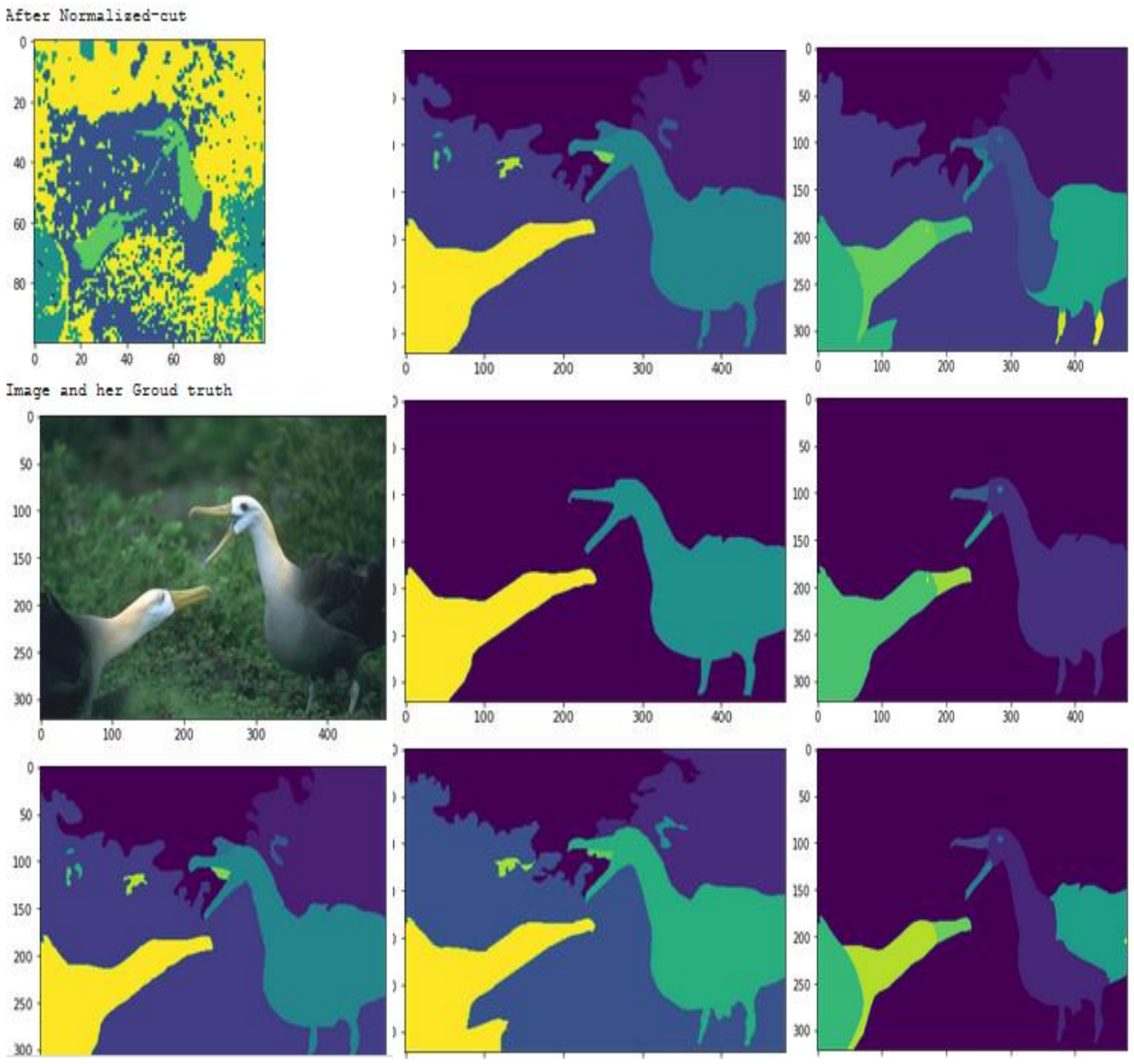


```
After Normalized-cut
```

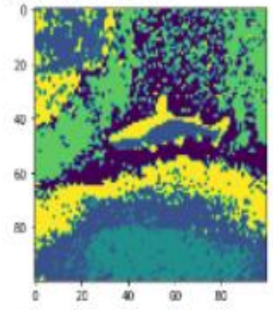


```
Image and her Groud truth
```




After Normalized-cut
Image and her Groud truth

After Normalized-cut



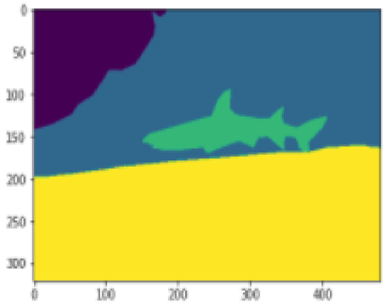
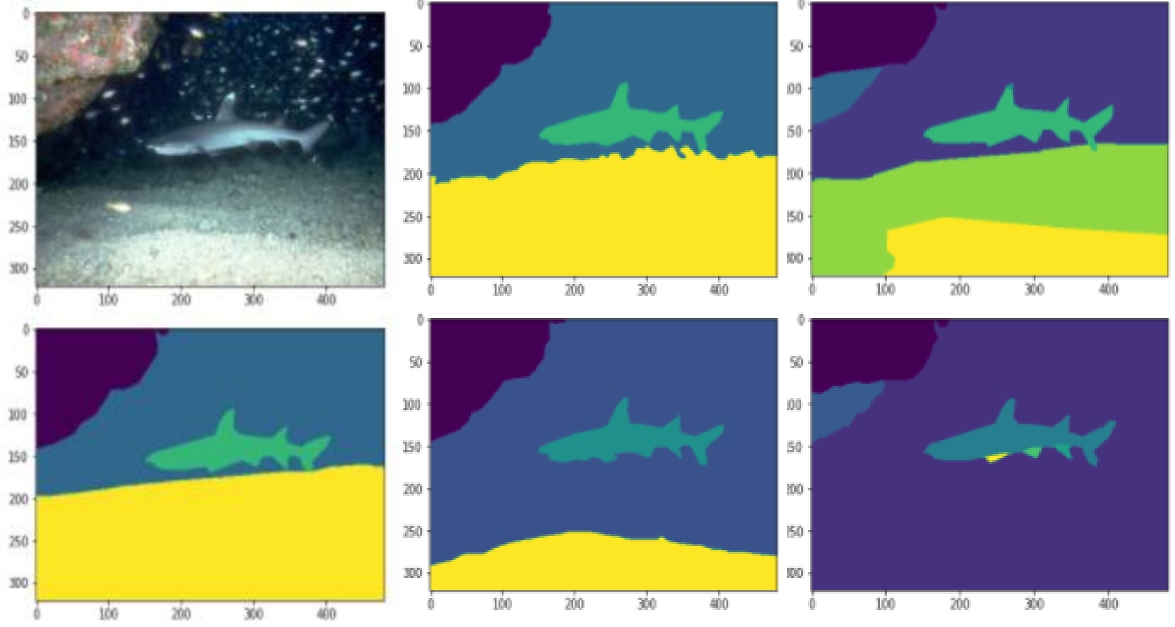
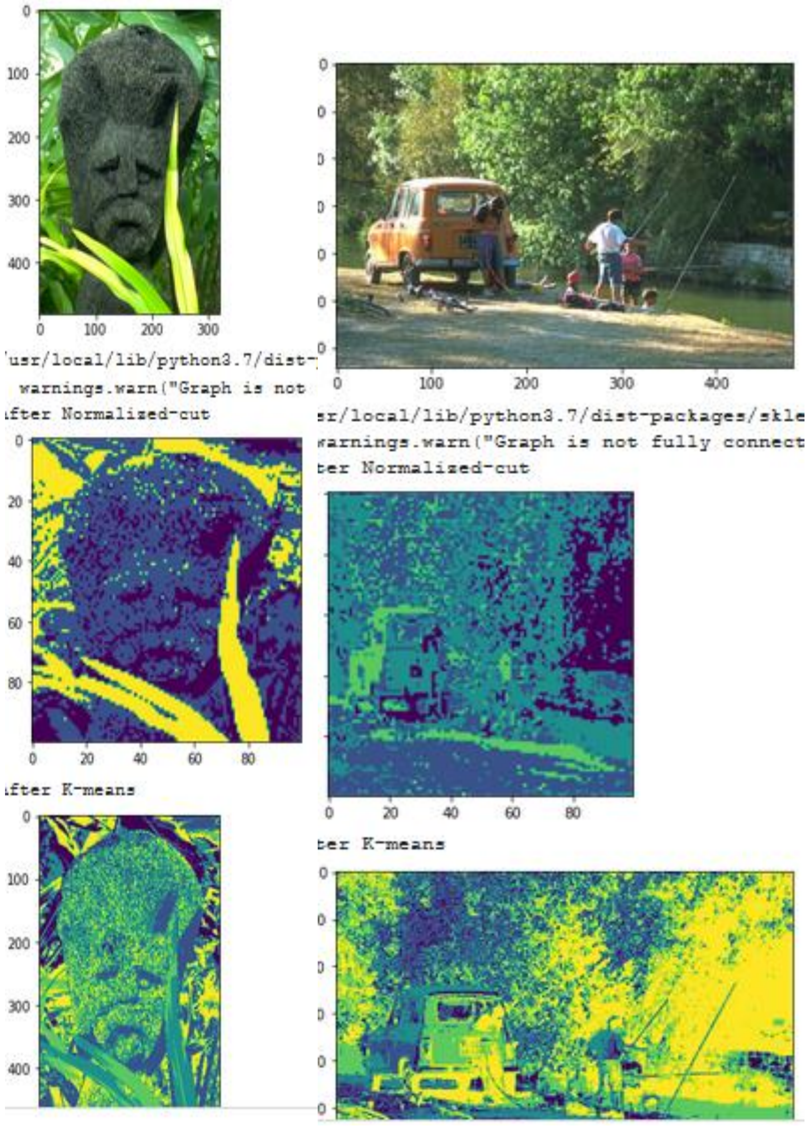


Image and her Groud truth



c) K-means vs Normalized cut:

Spectral Clustering is more computationally expensive than K-Means for large datasets because it needs to do the eigen decomposition. normalized cutting is better when identifying objects that are in an environment with a different coloring while it fails when the colors are

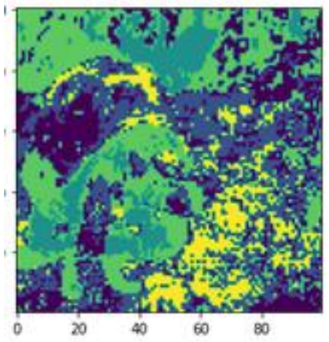


```
'usr/local/lib/python3.7/dist-
 warnings.warn("Graph is not
ıfter Normalized-cut
ıfter K-means
```

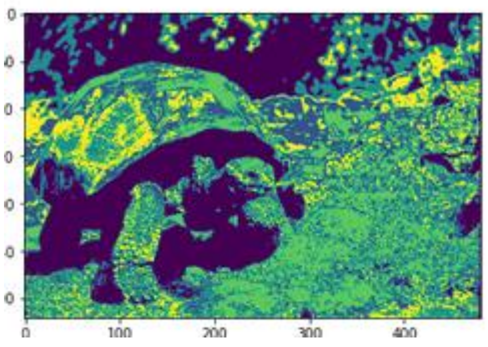
```
sr/local/lib/python3.7/dist-packages/skle
varnings.warn("Graph is not fully connect
ter Normalized-cut
ter K-means
```

close.

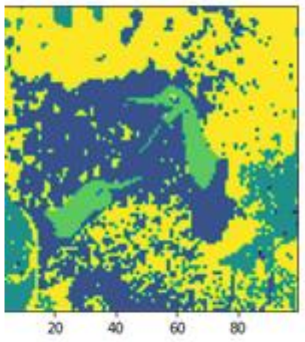
```
ter Normalized-cut
```



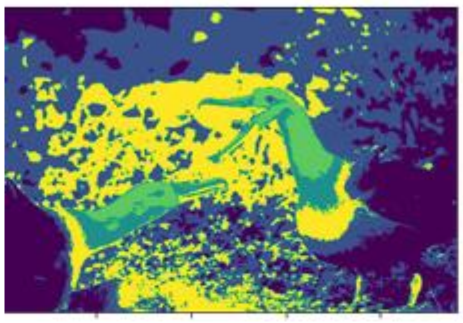
```
ter K-means
```



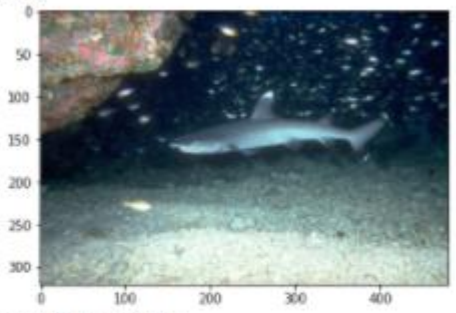
```
/local/lib/python3.7/dist-packages/skle
rnings.warn("Graph is not fully connect
r Normalized-cut
```



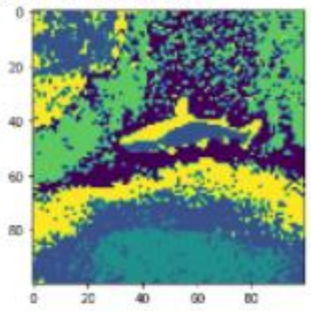
```
r K-means
```



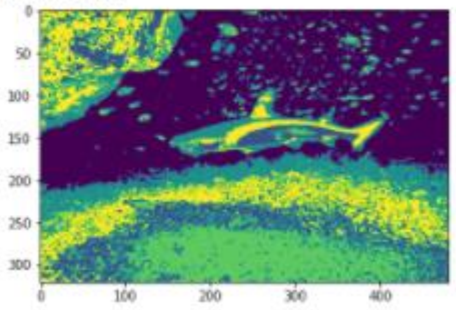
Image



After Normalized-cut



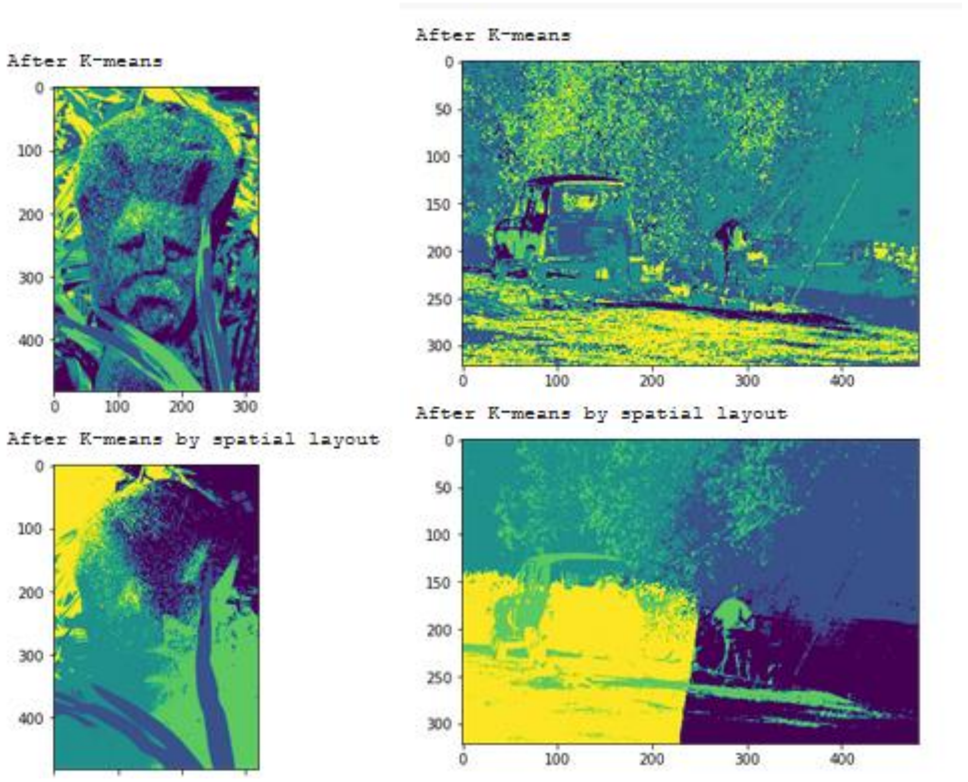
After K-means



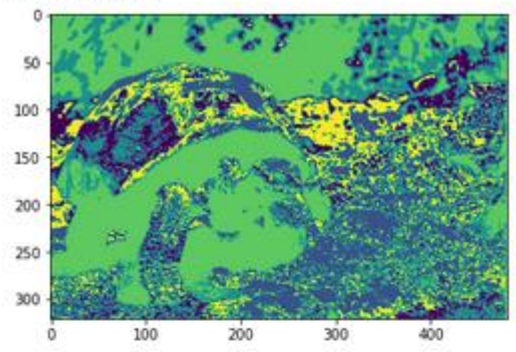
5-Encoding the Spatial Layout of the pixels.

```
if spatial:
  data = np.append(data, [(x, y) for x in range(h) for y in range(w)], axis = 1)
  normalization_parameters = np.array([255, 255, 255, h, w])
```

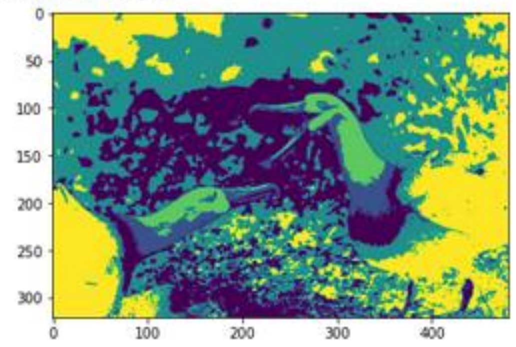
To take the Spatial layout of the pixels into consideration we have added another 2 dimensions in each image (x, y) where they corresponds to their location in the image frame.



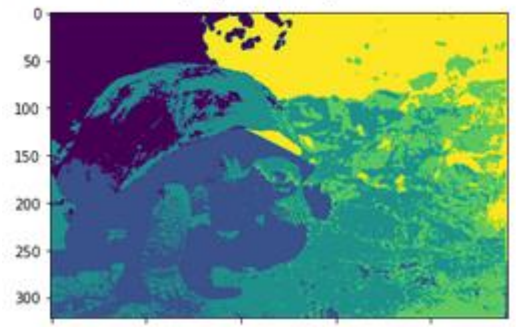
After K-means



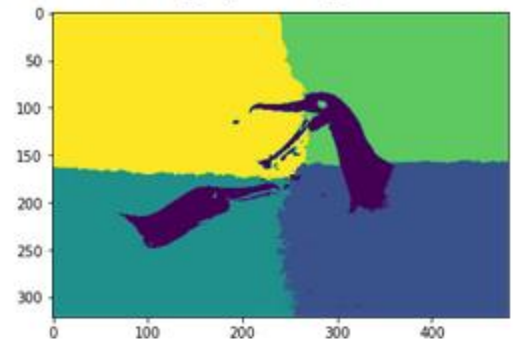
After K-means



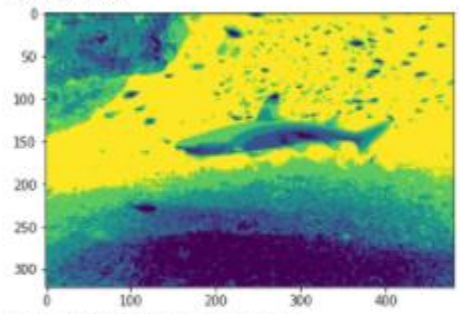
After K-means by spatial layout



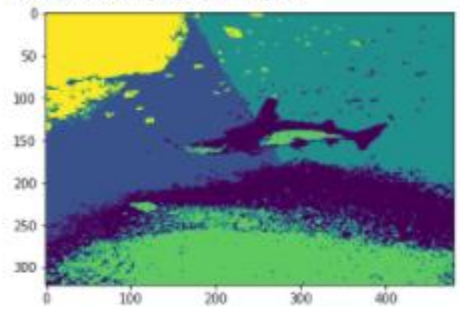
After K-means by spatial layout



After K-means



After K-means by spatial layout


0
50
100
150
200
250
300
0
100
200
300
400