Assignment 2: Image Segmentation

Team Members:

1. Sohaila Hazem – 6388
2. Areej Salahuddin – 6389
3. Manar Abdelkader – 6485

Problem Statement:

In this assignment, we intend to perform image segmentation. We first Visualized the images and used Kmeans to produce segmentations. We later used Normalized-cut for the 5-NN graph, to compare our results with kmeans.

1. Download the Dataset and Understand the Format

used Berkeley Segmentation Benchmark

In [ ]:
```
drive.mount('/gdrive')
!ln -s "/gdrive/MyDrive/groundTruth/test" "/content/groundTruthTest"
!ln -s "/gdrive/MyDrive/images/test" "/content/imagesTest"
test_path = "/content/imagesTest"
gt_test_path = "/content/groundTruthTest"
```

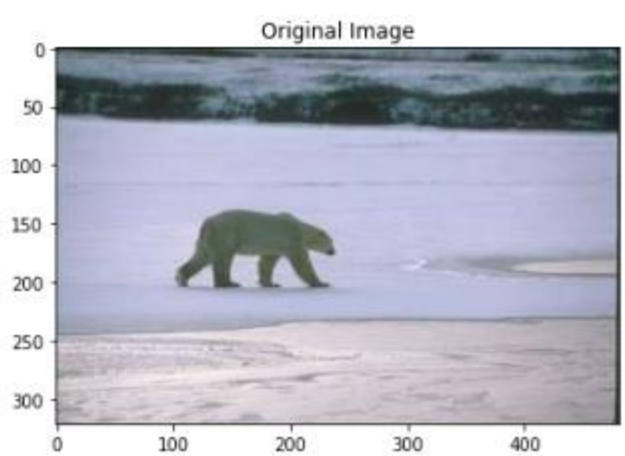
```
Mounted at /gdrive
```

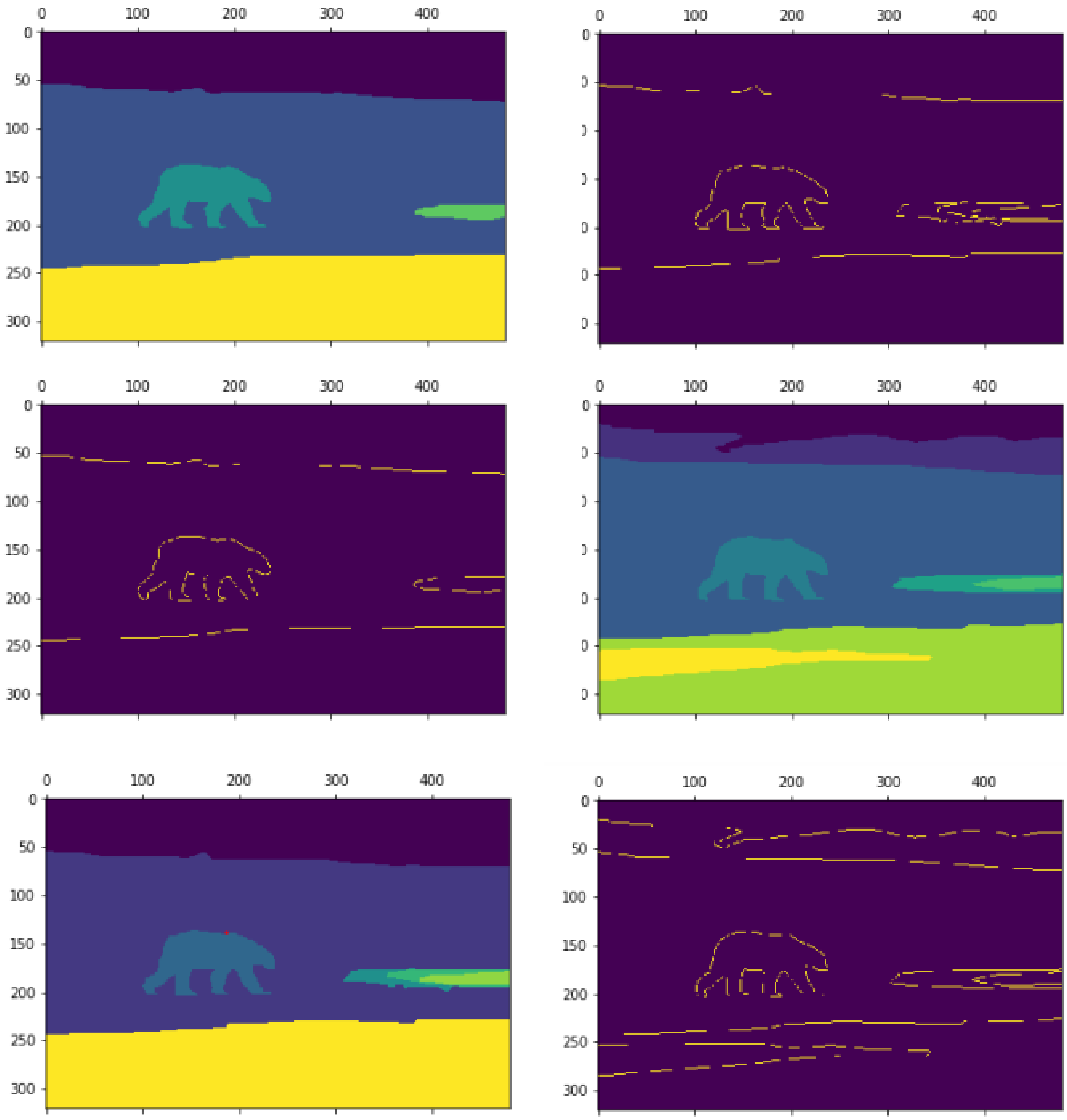
2. Read Data and Visualize the image and the ground truth segmentation

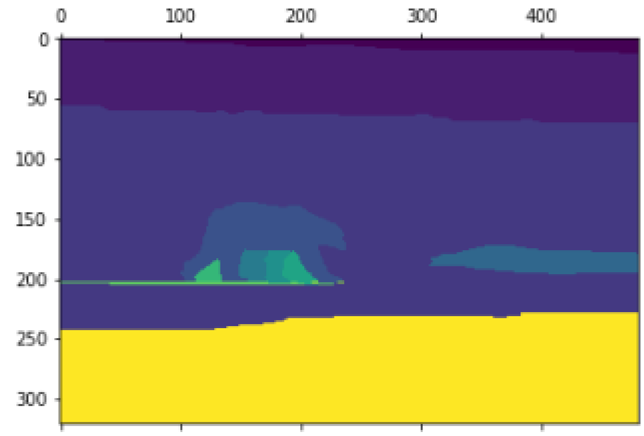
The test set is 200 images only. We will report our results on the first 50 images of the test set only

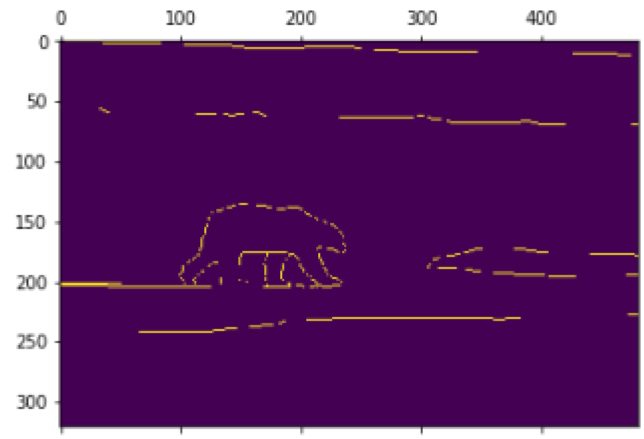
In [ ]:
```
images = []
grounds = []
files = []
files_mat =[]
for root, dirnames, filenames in os.walk(test_path):

  for filename in filenames:
    files.append(os.path.join(root, filename))
files= sorted(files)
for root, dirnames, filenames in os.walk(gt_test_path):

  for filename in filenames:
    files_mat.append(os.path.join(root, filename))
files_mat= sorted(files_mat)


for i in range(50):
  img = Image.open(files[i])
  img_array = np.asarray(img)
  #visualize
  plt.imshow(img)
  plt.title("Original Image")
  plt.show()
  mat = scipy.io.loadmat(files_mat[i])
  images.append(img_array)
  groundTruth = mat["groundTruth"]
  grounds.append(groundTruth)
  for k in groundTruth:
    for j in k:
      x = j["Segmentation"]
      y = j["Boundaries"]
      plt.matshow(x[0][0])
      plt.matshow(y[0][0])
```

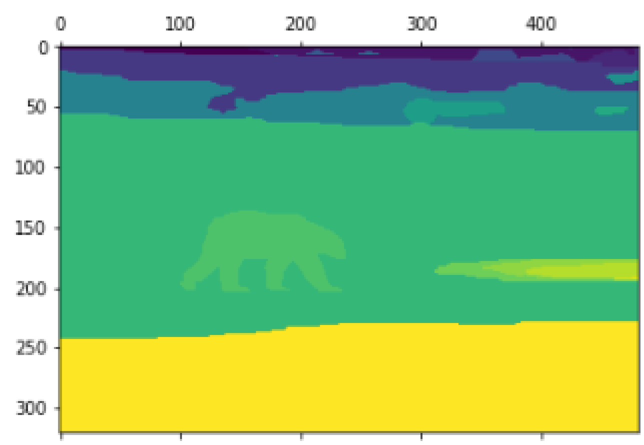
Example of 1 Image:




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


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


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


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

3. Segmentation using K-means

Kmeans implementation We are clustering for k = 3,5,7,9,11.

First, we select centroid points at random by selecting random indices and assigning centroids from our image that correspond to those random indices. Cluster Assignment: We created the distance matrix that describes the distance between all the points and the selected centroids and assign each point to the cluster of its closest centroid. Centroid Update: Calculate the mean of each cluster and update the centroid of each cluster with its mean value. We calculate the error by getting the norm of the difference between new & old centroids and if it's almost the same, then we get our final clusters.

```
In [ ]: def k_means(image, k,d):
    img_vec = image.reshape(-1,d)
    # random centroids
    centroids = []
    random_indices = np.random.randint(low = 0, high = img_vec.shape[0], size =
k)
    for index in range(k):
        centroids.append(img_vec[random_indices[index]])
    prev_centroids = np.zeros(np.array(centroids).shape)
    clusters = np.zeros(np.array(centroids).shape) #np.asarray
# calculate the error by getting the norm of the difference between new & old c
entroids
    err = np.linalg.norm(np.array(centroids) - np.array(prev_centroids))

    distance_matrix = np.zeros((img_vec.shape[0], np.array(centroids).shape[0
]))
    no_of_iterations = 0
    while err > 0.0001 and no_of_iterations < 100:
        no_of_iterations += 1
        for i in range(k):
            distance_matrix[:,i] = np.linalg.norm(img_vec - centroids[i], axis
= 1)
# cluster assignment step
        clusters = np.argmin(distance_matrix, axis = 1)
        prev_centroids = deepcopy(centroids)
# centroid update step
        for i in range(k):
            centroids[i] = np.mean(img_vec[clusters == i], axis = 0)
        err = np.linalg.norm(np.array(centroids) - np.array(prev_centroids))
    return centroids, clusters
```

F- Measure Implementation:

$$F_i = \frac{2n_{ij_i}}{n_i + m_{j_i}}$$

$$F = \frac{1}{r}\sum_{i=1}^{r} F_i$$

In [ ]:
```
def f_measure(y_true, y_pred):
  contingency_table = contingency_matrix(y_true, y_pred)
  max_position = contingency_table.argmax(axis=1)
  fi = 0
  F = 0
  for i in range(contingency_table.shape[0]):
        n_i = np.sum(contingency_table[i])
        n_ij = contingency_table.max(axis=1)[i]
        ji = contingency_table[:,max_position[i]]
        m_ji = np.sum(ji)
        F += fi + (2 * n_ij / (n_i + m_ji))

  F = F / contingency_table.shape[1]
  return F
```

Conditional Entropy Implementation:

In [ ]:
```
def conditional_entropy(y_true,y_pred):
  H = 0
  contingency_table = contingency_matrix(y_true,y_pred)
  sum_col = np.sum(contingency_table,axis =1)
  total = np.sum(sum_col)
  for n in range(contingency_table.shape[0]):
    for m in range(contingency_table.shape[1]):
      if(contingency_table[n][m] == 0):
        contingency_table[n][m]=1 #as log0 -->error
  HC = np.zeros((contingency_table.shape[0]))
  for i in range(contingency_table.shape[0]):
    n_i =  np.sum(contingency_table[i])
    HC[i] = (sum_col[i] / n_i) * np.sum(-contingency_table[i]*np.log10(continge
ncy_table[i]/n_i))

  H = np.sum(HC)/total
  return H
```

K-means clustering and getting the F-measure and conditional Entropy then comparing to get the bad and good results.

1. For each image, we segmented it using kmeans for k = 3,5,7,9,11.

2. We calculated the f-measure and conditional entropy for each image and its M ground truths for each k and saved the average for each image in a list.

3. We tabulated the results for the average of all M ground truth for each image and for each K, for both f-measure and conditional entropy.

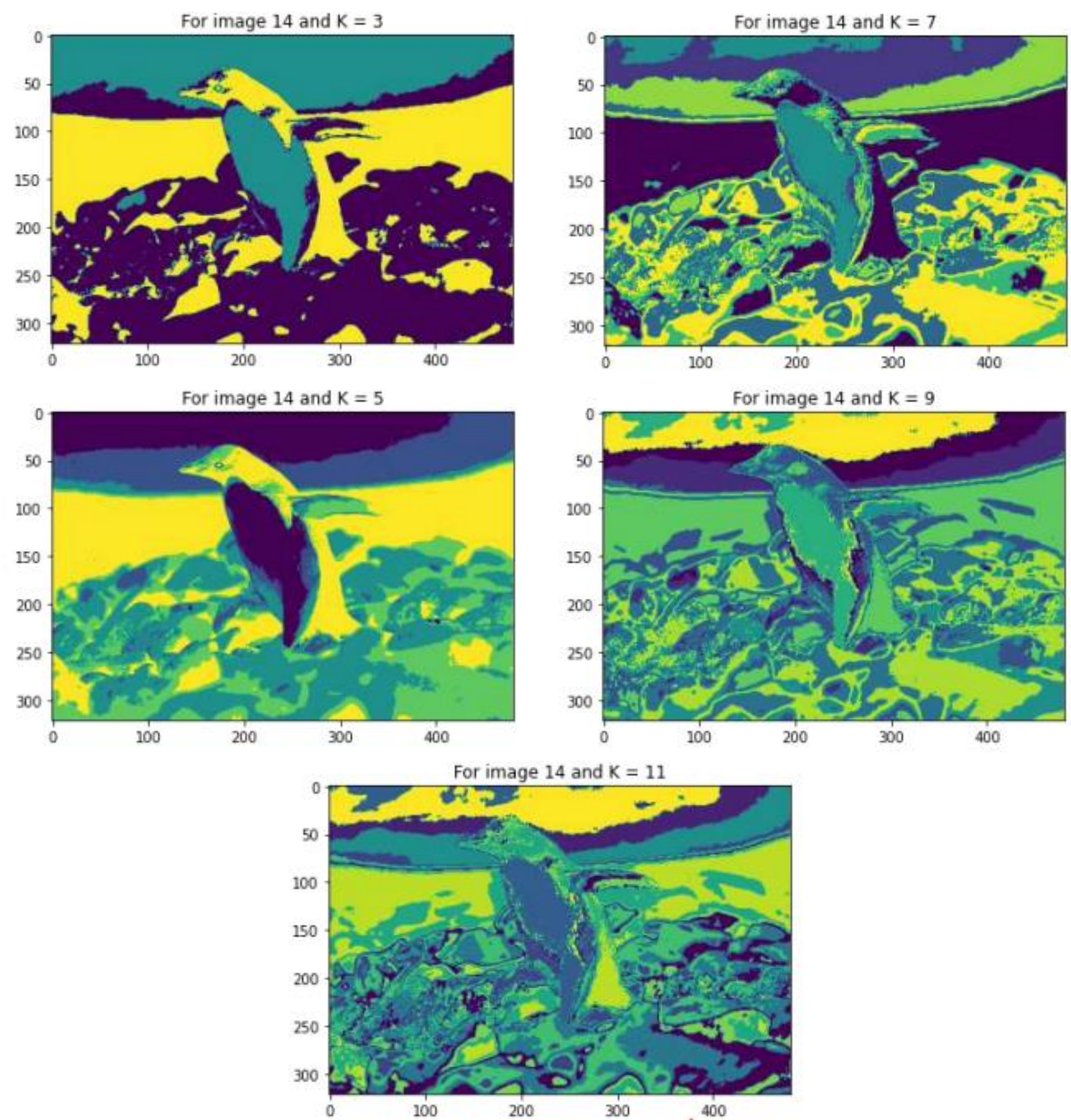
4. We reported the average per dataset according to f-measure and conditional entropy.

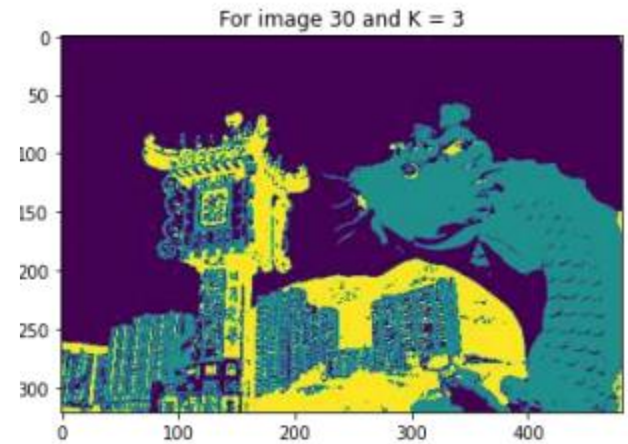
5. According to the f-measure and conditional entropy calculations, we got the good and bad results.

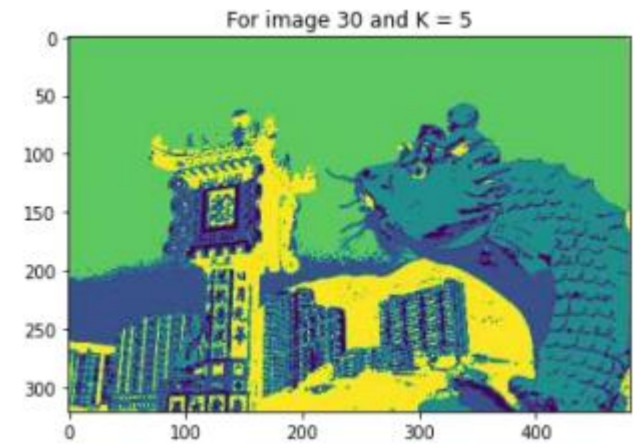
In [ ]:
```
totalavg_fmeasure = []
totalavg_entropy = []
min_f = 1
max_f = 0
min_c = 200
max_c = 0
good_image_fmeasure = []
good_gt_fmeasure = []
bad_image_fmeasure = []
bad_gt_fmeasure = []
good_image_entropy = []
good_gt_entropy = []
bad_image_entropy = []
bad_gt_entropy = []
avg_dataset_fmeasure = 0
avg_dataset_entropy = 0
clusters_5_list = []


for i in range(50):
  avg_fmeasure_list = []
  avg_conditional_entropy_list = []
  for k in [3,5,7,9,11]:
    f_measures_list = []
    conditional_entropy_list = []
    average_fmeasure = 0
    average_entropy = 0
    centroids, clusters = k_means(images[i],k,3)
    clusters = clusters.flatten()
    seg_img = clusters.reshape(images[i].shape[0], images[i].shape[1])
    if k == 5:
        if i in [5,10,15,20,30]:
            clusters_5_list.append(seg_img)
    for groundtruth in grounds[i]:
      for array in groundtruth:
        ground_truth = array[0][0][0]
        segmented_image = seg_img
        F = f_measure(ground_truth.flatten(),segmented_image.flatten())
        f_measures_list.append(F)
        C = conditional_entropy(ground_truth.flatten(),segmented_image.flatten
())
        conditional_entropy_list.append(C)

        if F < min_f:
          min_f = F
          bad_f = k
          bad_image_fmeasure = segmented_image
          bad_gt_fmeasure = ground_truth
          bad_image_fmeasure_idx = i

        if F > max_f:
          max_f = F
          good_f = k
          good_image_fmeasure = segmented_image
          good_gt_fmeasure = ground_truth
          good_image_fmeasure_idx = i

        if C < min_c:
          min_c = C
          good_c = k
          good_image_entropy = segmented_image
```

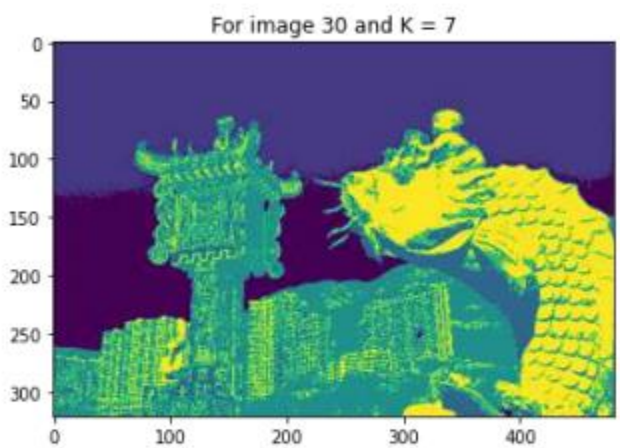
```
            good_gt_entropy = ground_truth
            good_image_entropy_idx = i

          if C>max_c:
            max_c = C
            bad_c = k
            bad_image_entropy = segmented_image
            bad_gt_entropy = ground_truth
            bad_image_entropy_idx = i

          avg_dataset_fmeasure += F
          avg_dataset_entropy += C

      average_fmeasure = sum(f_measures_list)/len(f_measures_list) #for one image
      average_entropy = sum(conditional_entropy_list)/len(conditional_entropy_lis
t)
      avg_fmeasure_list.append(average_fmeasure)
      avg_conditional_entropy_list.append(average_entropy)
      plt.imshow(seg_img)
      plt.title('For image {} and K = {}'.format(i + 1, k))
      plt.show()

    totalavg_fmeasure.append(avg_fmeasure_list)
    totalavg_entropy.append(avg_conditional_entropy_list)

avg_dataset_fmeasure = avg_dataset_fmeasure / (50*5)
avg_dataset_entropy = avg_dataset_entropy / (50*5)
```

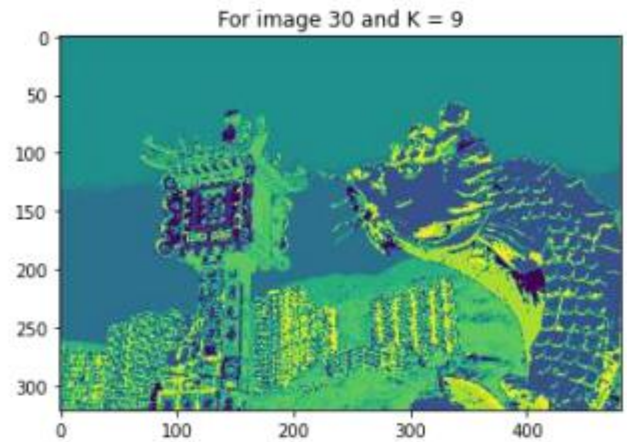
2 Examples for Kmeans Segmentation:

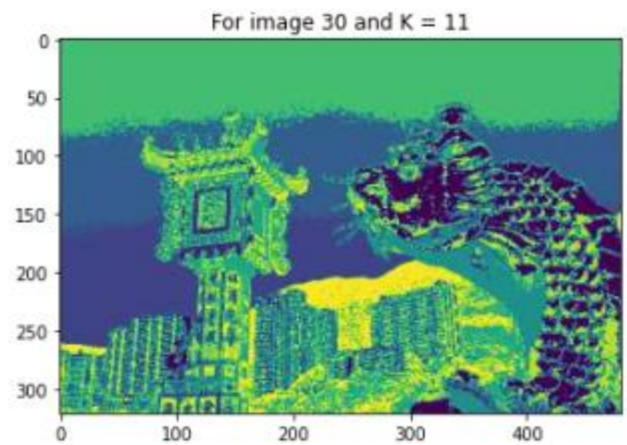



For image 30 and K = 3


For image 30 and K = 5


For image 30 and K = 7


For image 30 and K = 9


For image 30 and K = 11

Showing Results:

```
In [ ]: from tabulate import tabulate

        head = ["K = 3", "K = 5" , "K = 7" , "K = 9", "K = 11"]
        f = np.hstack([totalavg_fmeasure])
        c = np.hstack([totalavg_entropy])

        print(tabulate(f, headers=head))
        print(tabulate(c, headers=head))
```

F-measure:

K = 3	K = 5	K = 7	K = 9	K = 11
0.886132	0.571989	0.441275	0.344533	0.268577
0.71541	0.513314	0.354984	0.276713	0.206035
0.899756	0.504844	0.334154	0.248351	0.210763
0.798959	0.545116	0.360022	0.299972	0.249953
0.713382	0.523421	0.351429	0.267674	0.216023
0.821207	0.514674	0.396124	0.327421	0.28854
0.560164	0.343726	0.247416	0.197233	0.21165
0.589719	0.319761	0.204354	0.142094	0.110967
0.552837	0.373057	0.275188	0.216605	0.175305
0.982301	0.585231	0.379017	0.307376	0.236655
0.44663	0.266973	0.19886	0.15337	0.128722
0.828336	0.622943	0.451248	0.330982	0.275591
0.446031	0.271701	0.212784	0.165362	0.126332
0.782795	0.409019	0.309266	0.23817	0.183418
0.504632	0.228178	0.16195	0.114507	0.0845695
0.725866	0.427669	0.320062	0.264135	0.219642
0.677526	0.5001	0.331669	0.264749	0.205318
0.791535	0.408398	0.278134	0.202629	0.168155
0.449498	0.256836	0.171142	0.132058	0.107901
0.629312	0.350675	0.24462	0.186499	0.152284
0.371017	0.201437	0.123437	0.0817735	0.0620316
0.4156	0.25133	0.170902	0.122121	0.098415
0.778525	0.526091	0.337259	0.2473	0.198792
0.911437	0.562627	0.412569	0.352126	0.308737
1.01122	0.657767	0.522102	0.429021	0.348787
0.783753	0.688144	0.490581	0.395671	0.30567
0.898192	0.69365	0.486024	0.386345	0.315298
0.702448	0.431772	0.31575	0.243888	0.200728
1.20899	0.954143	0.577849	0.598254	0.486723
1.15446	0.772356	0.556365	0.460643	0.344728
0.969501	0.569839	0.401104	0.306109	0.266934
1.12757	0.853503	0.765385	0.563194	0.516978
0.692911	0.443638	0.257061	0.290688	0.234811
0.716471	0.412116	0.271147	0.213286	0.164423
0.846977	0.551106	0.420197	0.321178	0.263699
0.762346	0.47795	0.326038	0.247667	0.190416
0.975656	0.767343	0.530142	0.447082	0.343267
1.01482	0.617636	0.468939	0.395665	0.306536
0.635399	0.400413	0.28815	0.224355	0.231824
0.672843	0.445285	0.355314	0.303539	0.25754
0.497723	0.258741	0.177126	0.131283	0.131105
0.439016	0.235395	0.161925	0.116891	0.0899597
0.991214	0.731699	0.588048	0.555316	0.452043
0.840276	0.517662	0.4107	0.325256	0.269678
1.1186	0.854464	0.669804	0.554063	0.479411
0.867015	0.553402	0.417461	0.327099	0.296054
1.08085	0.784327	0.529975	0.427343	0.343852
1.05094	0.877614	0.760605	0.586458	0.486238
1.06877	0.645773	0.56306	0.427219	0.343822
0.790369	0.425049	0.336687	0.24695	0.216202

Conditional Entropy:

K = 3	K = 5	K = 7	K = 9	K = 11
0.172939	0.357984	0.464094	0.509736	0.586611
0.33345	0.522241	0.659583	0.745421	0.827347
0.199545	0.358616	0.459757	0.556284	0.618353
0.231539	0.336824	0.485562	0.572647	0.58238
0.300526	0.348972	0.486478	0.638146	0.720111
0.317653	0.469659	0.582111	0.668484	0.712187
0.379356	0.578469	0.714752	0.782852	0.846888
0.395516	0.597396	0.747865	0.851575	0.936582
0.309198	0.461586	0.59814	0.681873	0.752544
0.207363	0.416195	0.531175	0.62076	0.703513
0.381015	0.551178	0.682327	0.775942	0.874064
0.241826	0.387358	0.523628	0.609301	0.691936
0.398215	0.553915	0.690698	0.787687	0.879033
0.268327	0.458193	0.570631	0.669928	0.757116
0.358447	0.563011	0.696523	0.805727	0.878202
0.350284	0.507467	0.575814	0.655344	0.723992
0.376235	0.502046	0.639106	0.721199	0.807058
0.319997	0.529775	0.661408	0.76087	0.788661
0.369966	0.583229	0.676618	0.748793	0.804873
0.327698	0.536495	0.692048	0.795115	0.867995
0.368055	0.594061	0.747728	0.863127	0.943058
0.340322	0.548543	0.691074	0.813058	0.879282
0.267279	0.437677	0.549355	0.631069	0.694047
0.338752	0.522137	0.656491	0.713383	0.757734
0.18761	0.359406	0.474744	0.5607	0.640118
0.321357	0.384981	0.520507	0.592264	0.682124
0.259437	0.360544	0.487872	0.584231	0.648915
0.371714	0.565541	0.693426	0.786261	0.873864
0.184134	0.318599	0.423967	0.499471	0.573967
0.173143	0.314373	0.402984	0.452396	0.593557
0.304703	0.48788	0.609683	0.71438	0.777817
0.252685	0.380209	0.42243	0.542418	0.575351
0.321502	0.513525	0.634909	0.644697	0.715878
0.360572	0.538897	0.654946	0.747451	0.831573
0.35244	0.535188	0.619968	0.721925	0.762134
0.35052	0.540337	0.671441	0.757443	0.833747
0.254839	0.290806	0.433241	0.48189	0.624706
0.149461	0.276717	0.432253	0.411978	0.587886
0.383453	0.563206	0.699327	0.771684	0.8025
0.355671	0.554535	0.634658	0.773118	0.700071
0.394711	0.5965	0.732368	0.820118	0.762648
0.432753	0.645484	0.744375	0.816651	0.908558
0.314009	0.391791	0.530871	0.551578	0.654681
0.314171	0.48196	0.605184	0.671973	0.734627
0.223141	0.321568	0.395417	0.539068	0.56648
0.335413	0.530535	0.614614	0.673201	0.753582
0.266185	0.377121	0.503186	0.584285	0.674866
0.265875	0.37214	0.442803	0.551661	0.611145
0.218725	0.385734	0.448956	0.556583	0.62958
0.246589	0.456524	0.545237	0.656715	0.725775

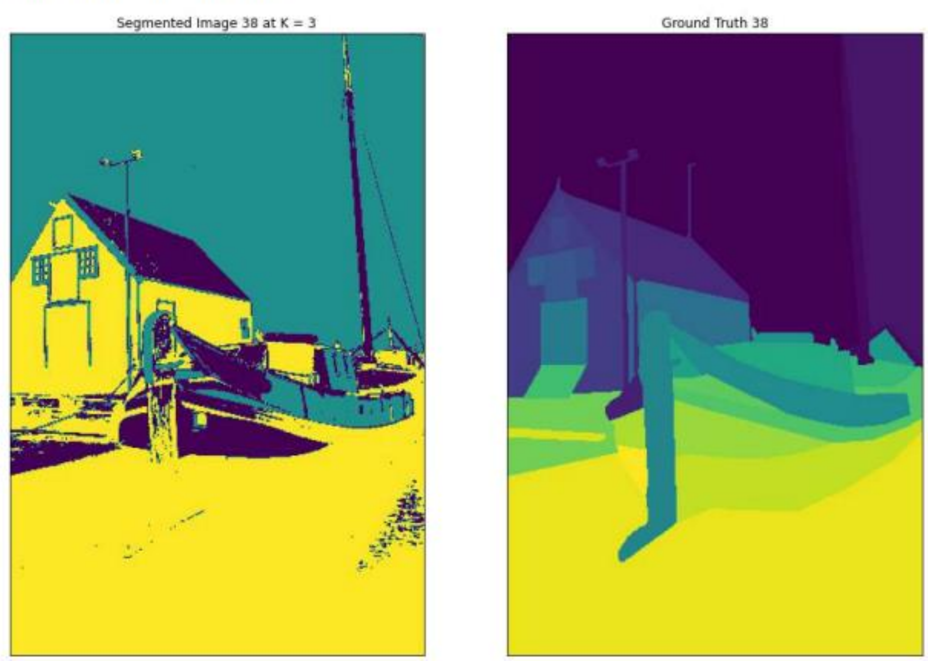
Good and Bad Results:

In [ ]:
```
def plot_img_seg(img, seg, k,image_no):
    figure_size = 15
    plt.figure(figsize=(figure_size,figure_size))
    plt.subplot(1,2,1),plt.imshow(img)
    plt.title('Segmented Image {} at K = {}'.format((image_no + 1),k)), plt.xti
cks([]), plt.yticks([])
    plt.subplot(1,2,2),plt.imshow(seg)
    plt.title('Ground Truth {}'.format(image_no + 1)), plt.xticks([]), plt.ytic
ks([])
    plt.show()
```

In [ ]:
```
print(f'Average F-Measure for the dataset= {avg_dataset_fmeasure},Average Condi
tional Entropy for the dataset= {avg_dataset_entropy} \n')
print("Good Conditional Entropy")
plot_img_seg(good_image_entropy,good_gt_entropy,good_c, good_image_entropy_idx)
print("Bad F-measure")
plot_img_seg(bad_image_fmeasure,bad_gt_fmeasure,bad_f, bad_image_fmeasure_idx)
print("Bad Conditional Entropy")
plot_img_seg(bad_image_entropy,bad_gt_entropy,bad_c, bad_image_entropy_idx)
print("Good F-measure")
plot_img_seg(good_image_fmeasure,good_gt_fmeasure,good_f,good_image_fmeasure_id
x)
```

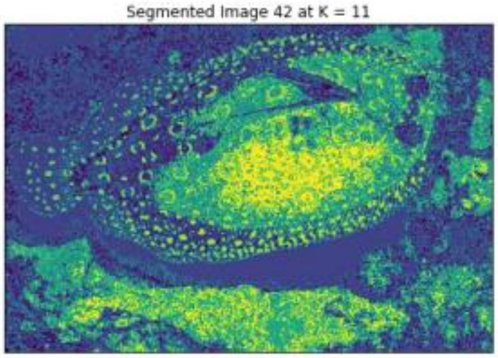
- Average F-Measure for the dataset= 2.3446270831748217
- Average Conditional Entropy for the dataset= 2.91042194047609

```
Good Conditional Entropy
```



Bad F-measure

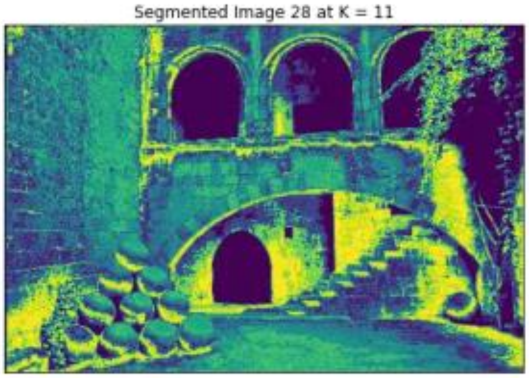
Segmented Image 42 at K = 11



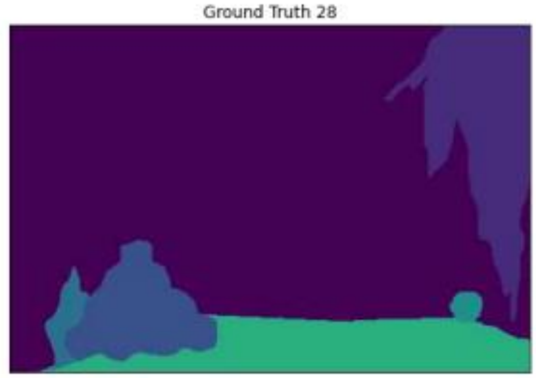
Ground Truth 42

Bad Conditional Entropy

Segmented Image 28 at K = 11

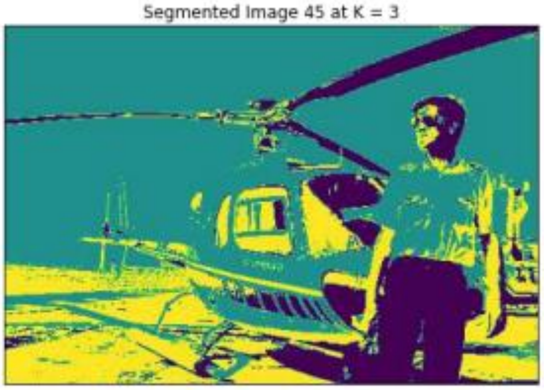


Ground Truth 28

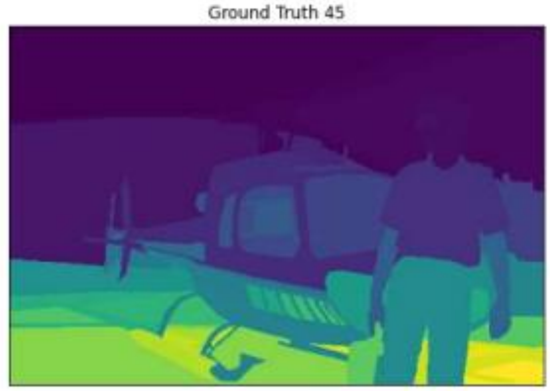


Good F-measure

Segmented Image 45 at K = 3



Ground Truth 45

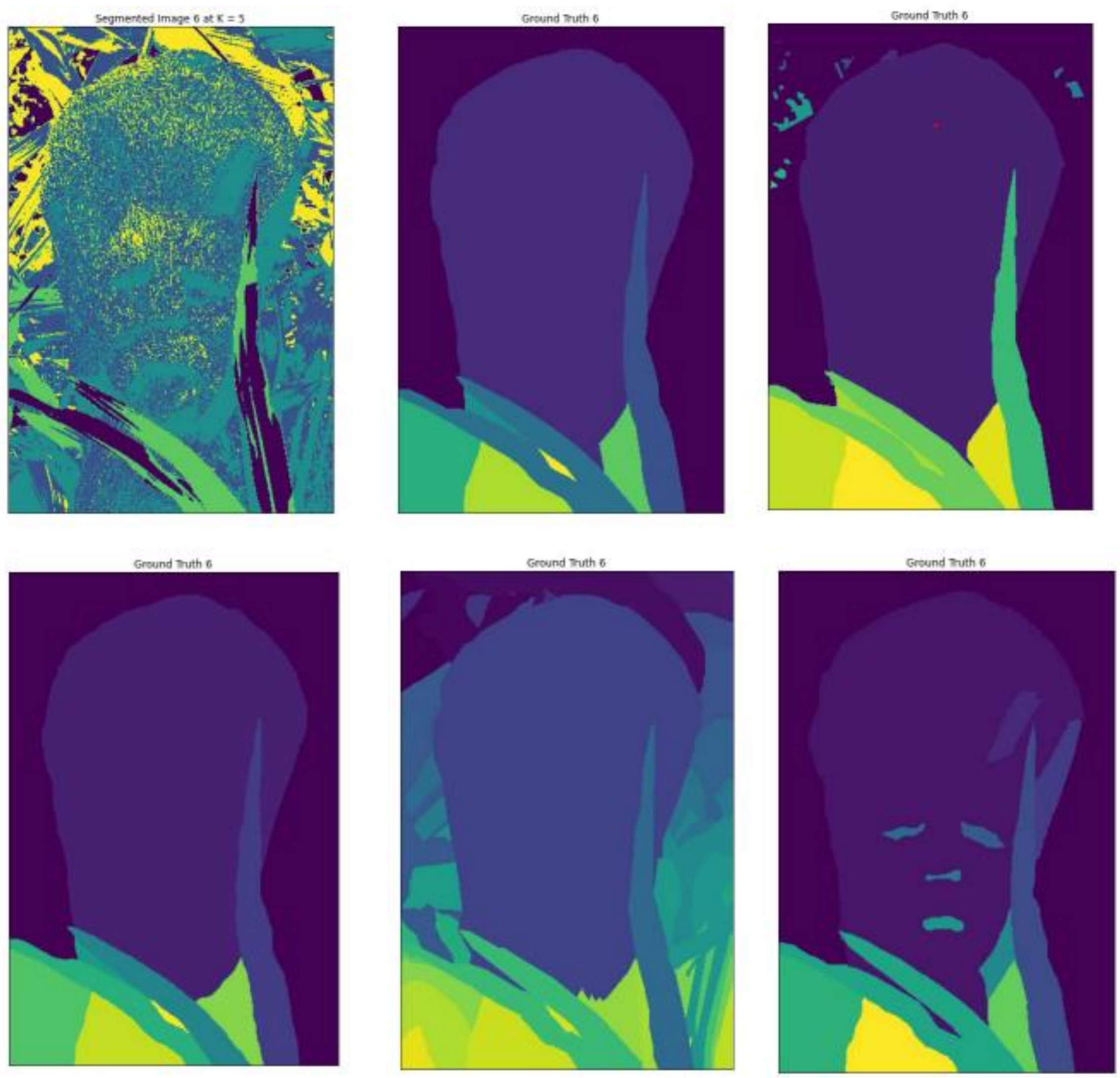


4. Big Picture:

a. Select a set of five images and display their corresponding ground truth against your segmentation results using K-means at K=5.

```
In [ ]: img_idx = [5,10,15,20,30]
        for i in range(5):
          groundTruths = grounds[img_idx[i]]
          clusters = clusters_5_list[i]
          for groundtruth in groundTruths:
            for array in groundtruth:
              plot_img_seg(clusters,array[0][0]['Segmentation'],5, img_idx[i])
```

2 Images of 5:



b. Select the same five images and display their corresponding ground truth against your segmentation results using Normalized-cut for the 5-NN graph, at K=5

Normalized Cut:

1. Compute the similarity matrix using kneighbors graph (5-NN) and the degree matrix.

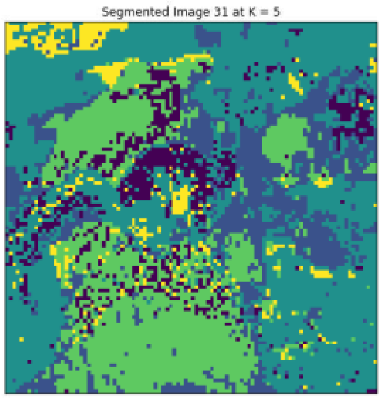
2. Eigenvalue decomposition on the asymmetric Laplacian matrix and sort the eigen vectors ascendingly according to their eigenvalues and extract the first k (number of clusters) eigen vectors.

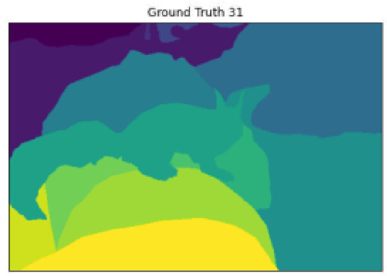
3. Normalize the first k eigenvectors and perform k-means on them to return the clusters.

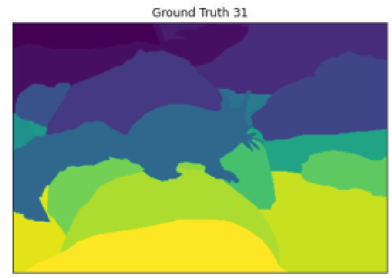
In [ ]:
```
# similarity matrix = A
def spectral_clustering(A, k, neighbors):
    # by adding up all the components of the corresponding row in the A matrix, 
we fill the cell along the diagonal of each row of the degree matrix.
    degree_matrix = np.diag(np.sum(A, axis=1))
    L = degree_matrix - A
    La = np.dot(inv(degree_matrix), L)
    eigenvalues, eigenvectors = eig(La)
    # sorting the eigen values ascendingly and taking the first k eigenvectors
    eigenvectors = eigenvectors[:,np.argsort(eigenvalues)]
    eigenvalues = eigenvalues[np.argsort(eigenvalues)]
    eigenvalues = eigenvalues.real
    eigenvectors = eigenvectors.real
    eigenvectors = eigenvectors.T
    U = eigenvectors[:k].T
    Y = np.zeros(U.shape)
    for i in range(0,U.shape[0]):
        norm = np.linalg.norm(U[i])
        Y[i] = U[i]/norm

    kmeans = KMeans(n_clusters=k, random_state=0).fit(Y)
    clusters = kmeans.labels_.flatten()

    return clusters
```

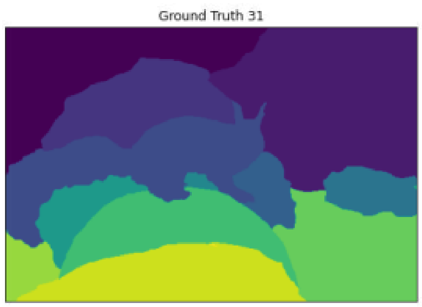
In [ ]:
```
normalised_cut_list = []
img_idx = [5,10,15,20,30]
for i in img_idx:
  groundTruths = grounds[i]
  image = images[i]
  image = resize(image, (100, 100))
  A = kneighbors_graph(image.reshape(-1,3), 5, mode='connectivity', include_sel
f=False)
  A = csr_matrix.toarray(A)
  clusters = spectral_clustering(A,5,5)
  seg_img = clusters.reshape(100,100)
  normalised_cut_list.append(seg_img)
for groundtruth in groundTruths:
    for array in groundtruth:
      plot_img_seg(seg_img,array[0][0]['Segmentation'],5, i)
```

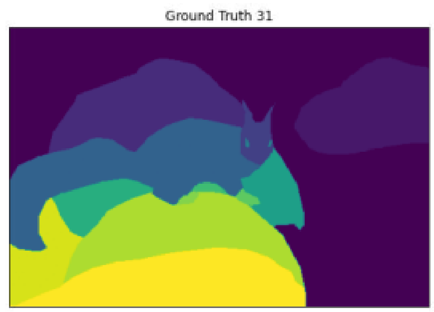
1 Image of 5:

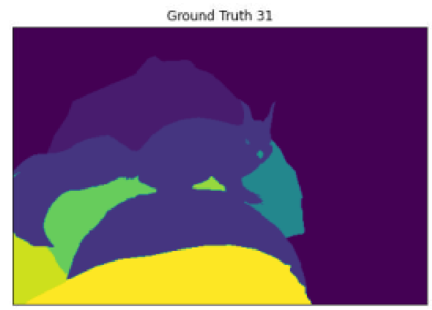










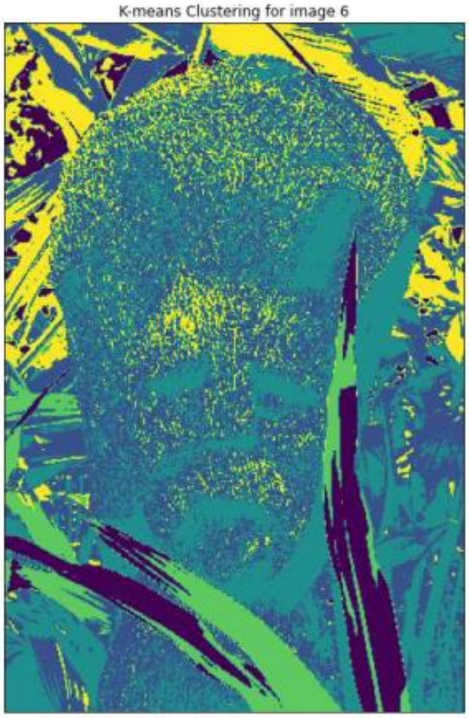


c. Select the same five images and contrast your segmentation results using Normalized-cut for the 5-NN graph, at K=5 versus using K-means at K=5.

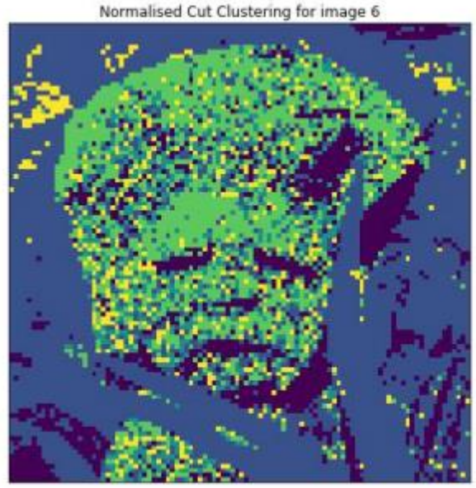
In [ ]:
```
def plot_compare(kmeans, normalised, image_no):
    figure_size = 15
    plt.figure(figsize=(figure_size,figure_size))
    plt.subplot(1,2,1),plt.imshow(kmeans)
    plt.title('K-means Clustering for image {}'.format((image_no + 1),k)), plt.
xticks([]), plt.yticks([])
    plt.subplot(1,2,2),plt.imshow(normalised)
    plt.title('Normalised Cut Clustering for image {}'.format(image_no + 1)), p
lt.xticks([]), plt.yticks([])
    plt.show()
```

In [ ]:
```
for i in range(5):
  plot_compare(clusters_5_list[i], normalised_cut_list[i],img_idx[i])
```

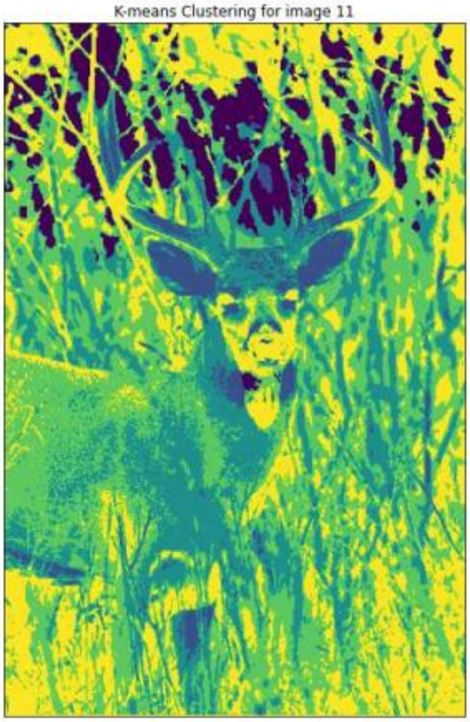
K-means Clustering for image 6



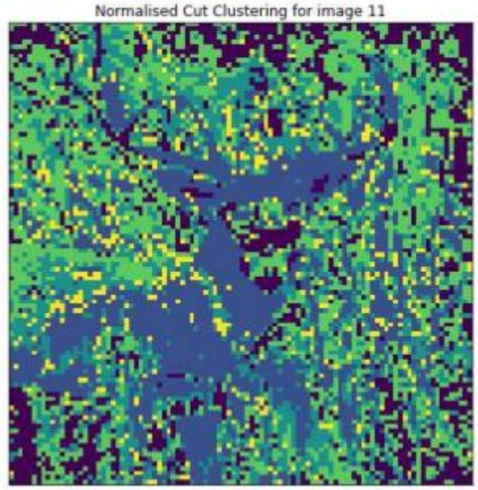
Normalised Cut Clustering for image 6



K-means Clustering for image 11

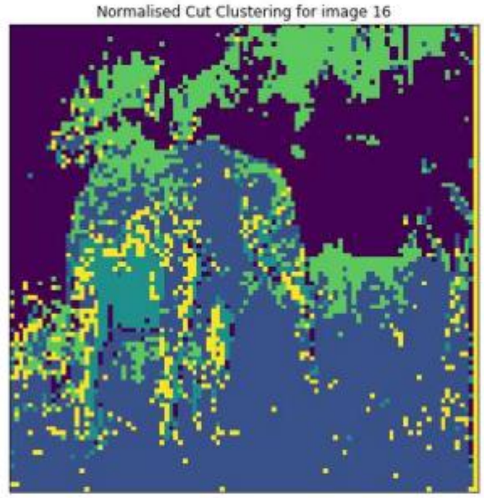


Normalised Cut Clustering for image 11

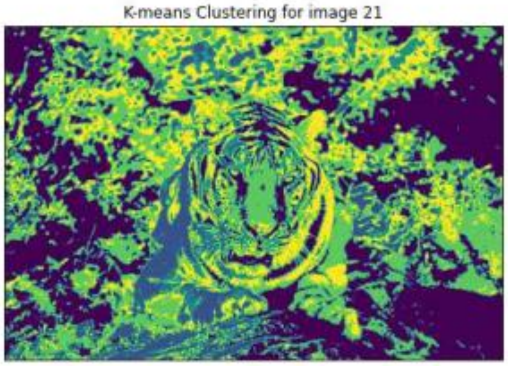


K-means Clustering for image 16

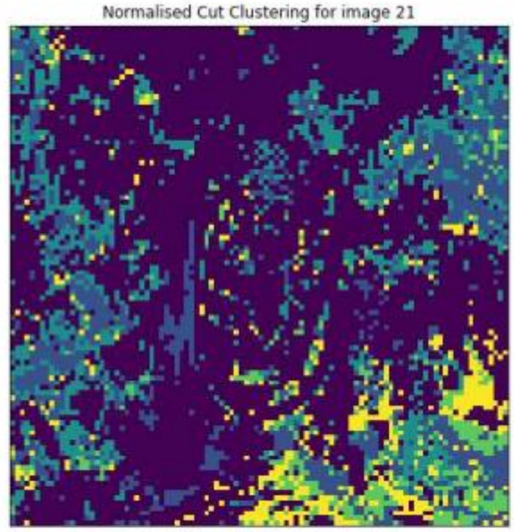
Normalised Cut Clustering for image 16



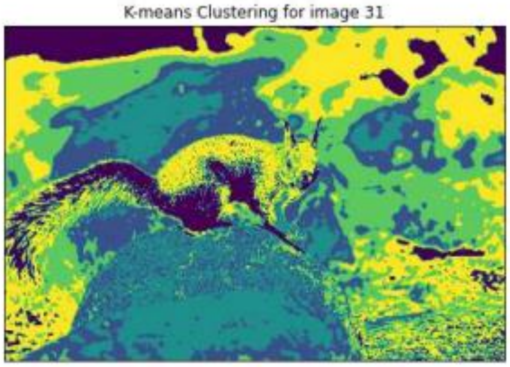
K-means Clustering for image 21



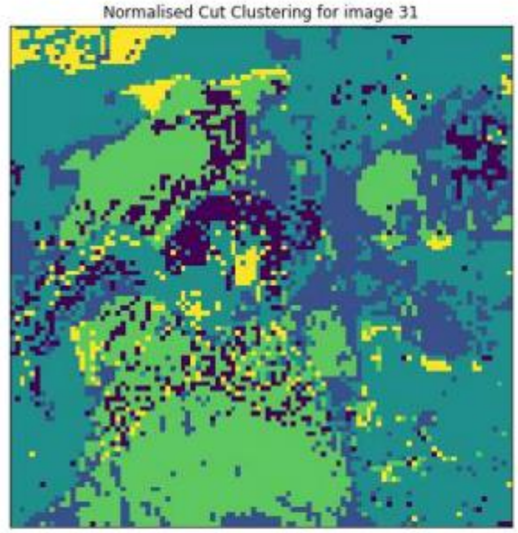
Normalised Cut Clustering for image 21



K-means Clustering for image 31



Normalised Cut Clustering for image 31



COMMENT

It was noticed that normalized cut was worse than k-means in clustering.

5.Bonus:

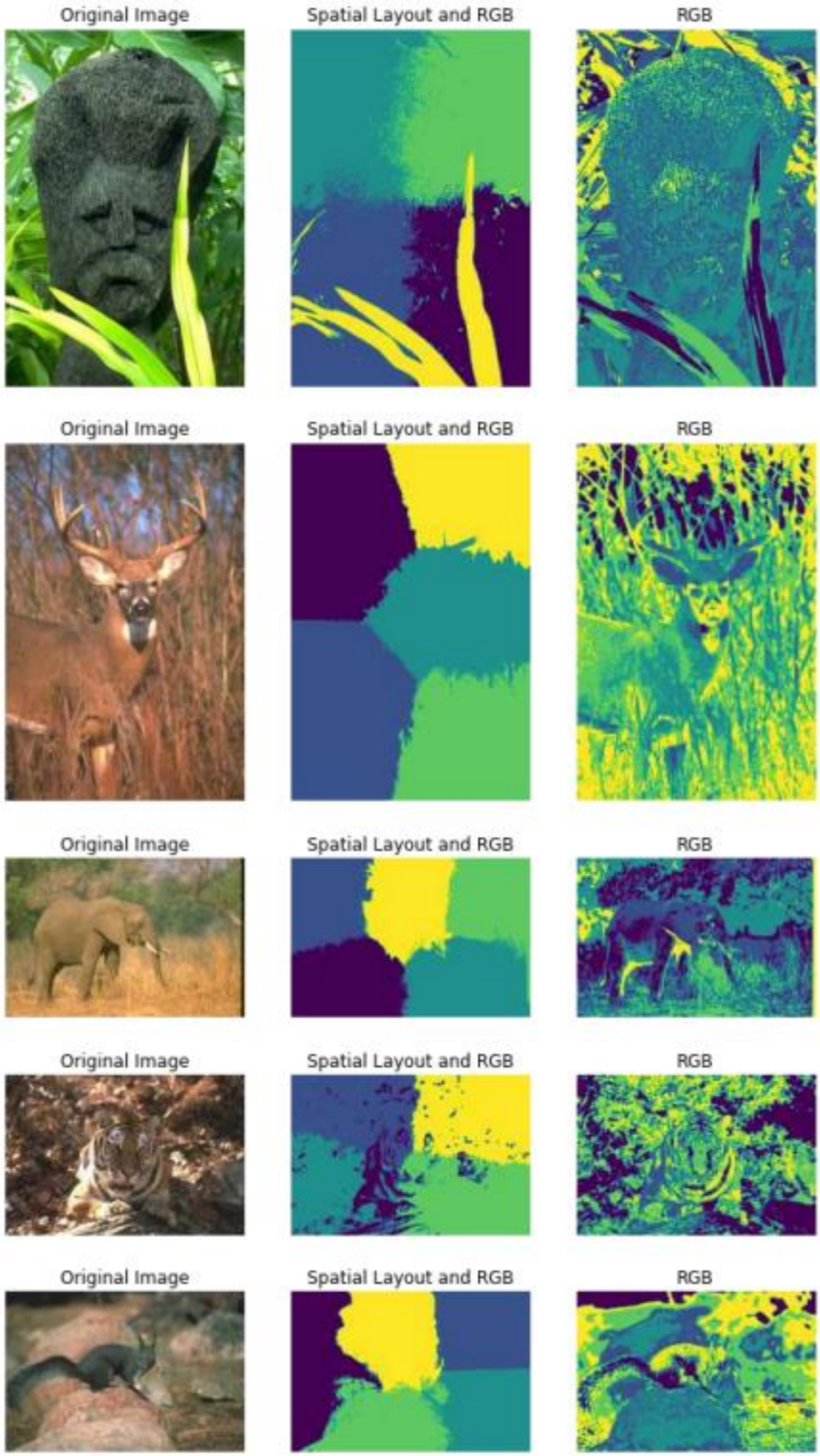
Modify Kmeans to encode the spatial layout of the pixels. (RGB + Spatial Layout)

In [ ]:
```
new_images = []
num = 5
k = 5
for i in [5,10,15,20,30]:
    width = images[i].shape[0]
    height = images[i].shape[1]
    spatial_layout = []
    for x in range(width):
        for j in range(height):
            spatial_layout.append([x,j])
    spatial_layout = np.array(spatial_layout)
    new_images.append(np.concatenate((images[i].reshape((-1, 3)), spatial_layou
t), axis=1).reshape((width, height, 5)))
```

In [ ]:
```
spatial_layout_rgb = []
for img in new_images:

    centroids,seg = k_means(img, 5, 5)
    seg = seg.flatten()
    seg = seg.reshape((img.shape[0], img.shape[1]))
    spatial_layout_rgb.append(seg)
```

Contrast the results:

In [ ]:
```
img_idx = [5,10,15,20,30]
for i in range(5):
    fig = plt.figure(figsize=(10, 5))
    original_img = images[img_idx[i]]
    spacial_rgb = spatial_layout_rgb[i]
    rgb = clusters_5_list[i]

    fig.add_subplot(1, 3, 1)
    plt.imshow(original_img)
    plt.axis('off')
    plt.title("Original Image")

    fig.add_subplot(1, 3, 2)
    plt.imshow(spacial_rgb)
    plt.axis('off')
    plt.title("Spatial Layout and RGB")

    fig.add_subplot(1, 3, 3)
    plt.imshow(rgb)
    plt.axis('off')
    plt.title("RGB")
```


Original Image
Spatial Layout and RGB
RGB
Original Image
Spatial Layout and RGB
RGB
Original Image
Spatial Layout and RGB
RGB
Original Image
Spatial Layout and RGB
RGB
Original Image
Spatial Layout and RGB
RGB