Pattern Recognition
Assignment (2) Report
Image Segmentation

Team Members

Nouran Hisham 6532

Google colab notebook link:

https://colab.research.google.com/drive/1RPAxwGCQgmihf8PFZViLhsbZMFV8pSIR?usp=sharing

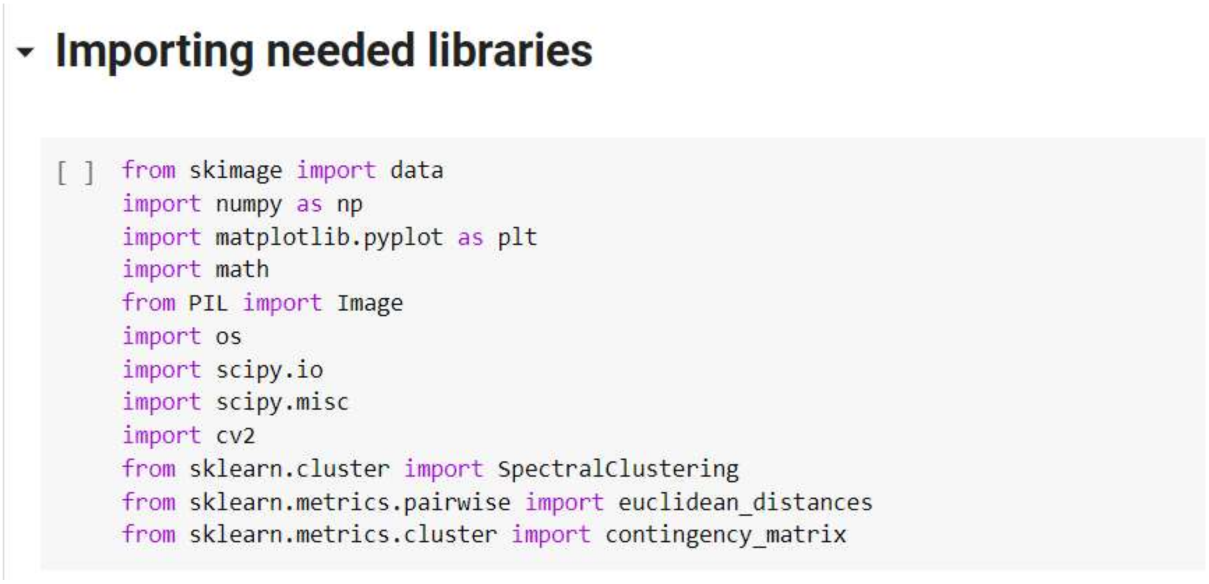
Note: Detailed output is present in the google colab notebook.

Problem Statement:

We intend to perform image segmentation. 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 on the Berkeley Segmentation Benchmark.

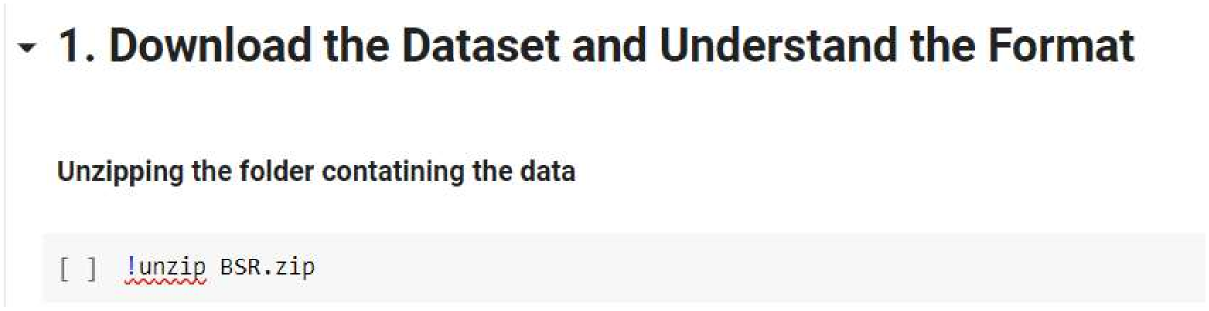
Dataset used is
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/BSR/BSR_bsds500.tgz

Imported libraries to be used are:


Importing needed libraries

```
[ ] from skimage import data
    import numpy as np
    import matplotlib.pyplot as plt
    import math
    from PIL import Image
    import os
    import scipy.io
    import scipy.misc
    import cv2
    from sklearn.cluster import SpectralClustering
    from sklearn.metrics.pairwise import euclidean_distances
    from sklearn.metrics.cluster import contingency_matrix
```

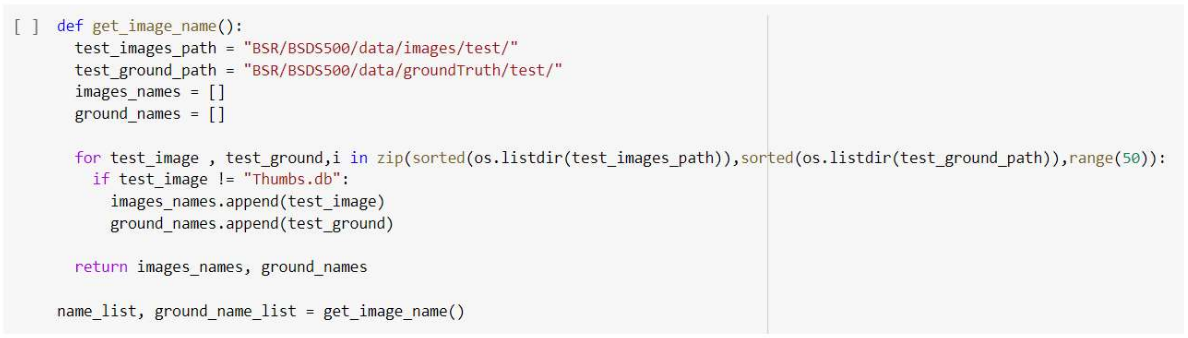
1.Download the Dataset and Understand the Format


1. Download the Dataset and Understand the Format

Unzipping the folder contating the data

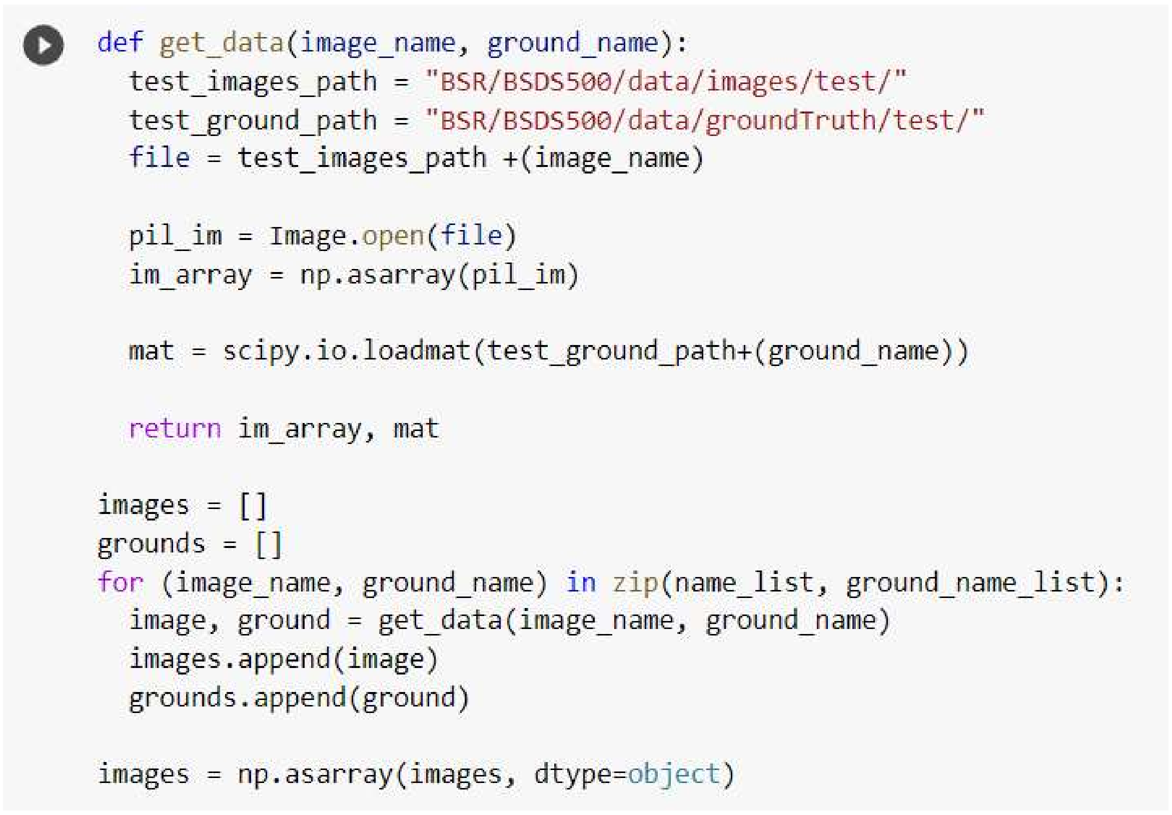
```
[ ] !unzip BSR.zip
```

Getting a list of first 50 test images' names



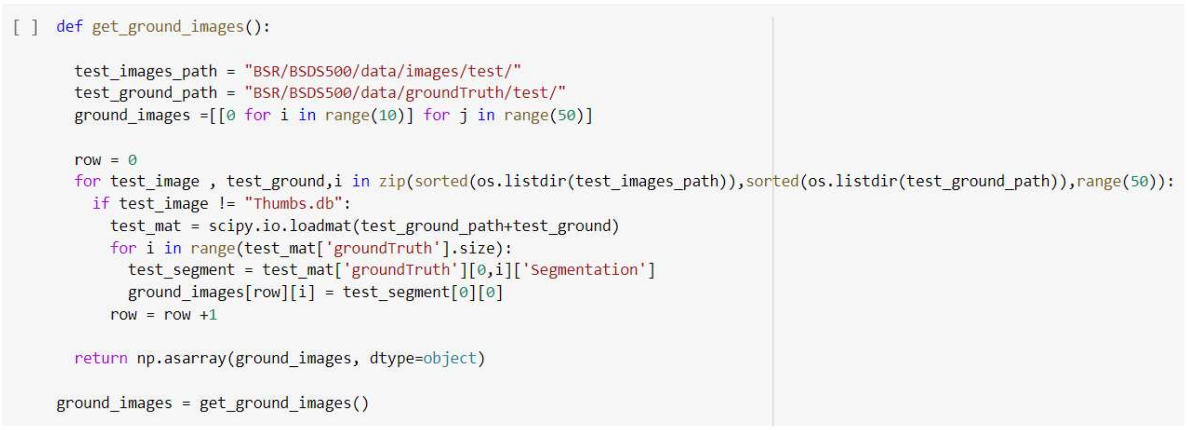
```
[ ]  def get_image_name():
       test_images_path = "BSR/BSDS500/data/images/test/"
       test_ground_path = "BSR/BSDS500/data/groundTruth/test/"
       images_names = []
       ground_names = []

       for test_image , test_ground,i in zip(sorted(os.listdir(test_images_path)),sorted(os.listdir(test_ground_path)),range(50)):
         if test_image != "Thumbs.db":
           images_names.append(test_image)
           ground_names.append(test_ground)

       return images_names, ground_names

     name_list, ground_name_list = get_image_name()
```

Reading the images and the .mat files



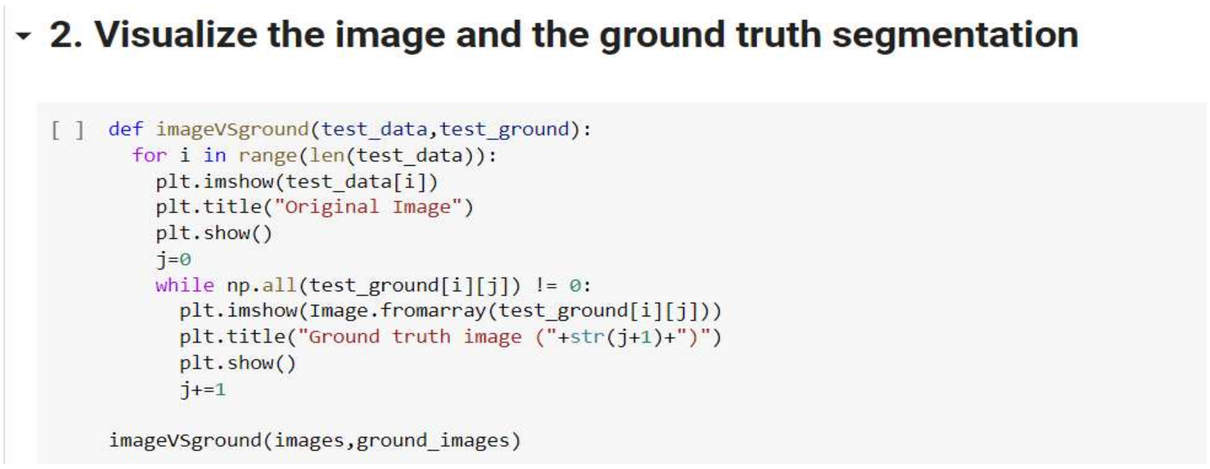
```
def get_data(image_name, ground_name):
  test_images_path = "BSR/BSDS500/data/images/test/"
  test_ground_path = "BSR/BSDS500/data/groundTruth/test/"
  file = test_images_path +(image_name)

  pil_im = Image.open(file)
  im_array = np.asarray(pil_im)

  mat = scipy.io.loadmat(test_ground_path+(ground_name))

  return im_array, mat

images = []
grounds = []
for (image_name, ground_name) in zip(name_list, ground_name_list):
  image, ground = get_data(image_name, ground_name)
  images.append(image)
  grounds.append(ground)

images = np.asarray(images, dtype=object)
```

Reading the ground images



```
[ ]  def get_ground_images():

       test_images_path = "BSR/BSDS500/data/images/test/"
       test_ground_path = "BSR/BSDS500/data/groundTruth/test/"
       ground_images =[[0 for i in range(10)] for j in range(50)]

       row = 0
       for test_image , test_ground,i in zip(sorted(os.listdir(test_images_path)),sorted(os.listdir(test_ground_path)),range(50)):
         if test_image != "Thumbs.db":
           test_mat = scipy.io.loadmat(test_ground_path+test_ground)
           for i in range(test_mat['groundTruth'].size):
             test_segment = test_mat['groundTruth'][0,i]['Segmentation']
             ground_images[row][i] = test_segment[0][0]
           row = row +1

       return np.asarray(ground_images, dtype=object)

     ground_images = get_ground_images()
```

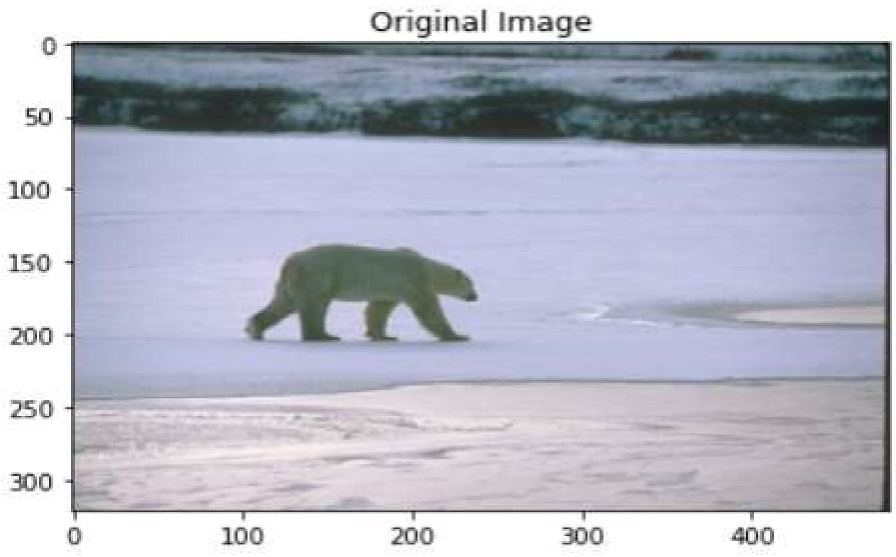
2.Visualize the image and the ground truth segmentation

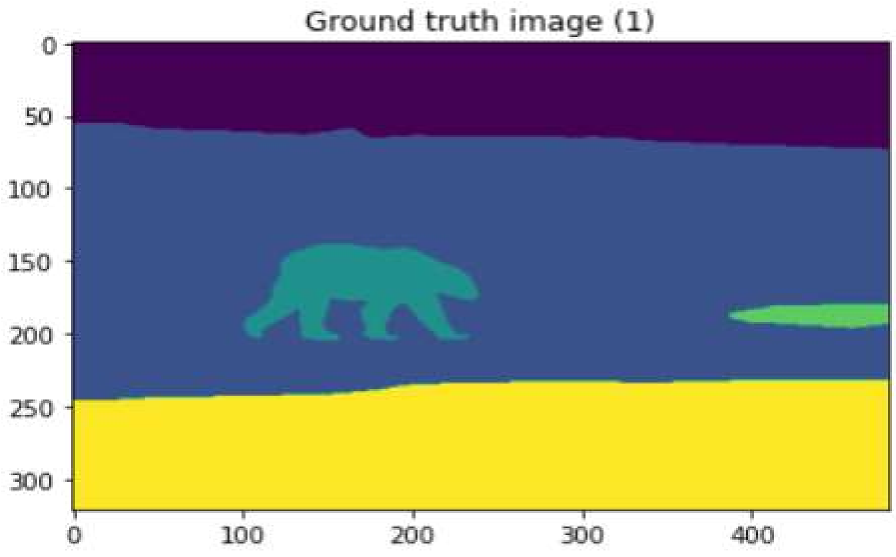


▾ 2. Visualize the image and the ground truth segmentation

```
[ ]  def imageVSground(test_data,test_ground):
       for i in range(len(test_data)):
         plt.imshow(test_data[i])
         plt.title("Original Image")
         plt.show()
         j=0
         while np.all(test_ground[i][j]) != 0:
           plt.imshow(Image.fromarray(test_ground[i][j]))
           plt.title("Ground truth image ("+str(j+1)+")")
           plt.show()
           j+=1

     imageVSground(images,ground_images)
```

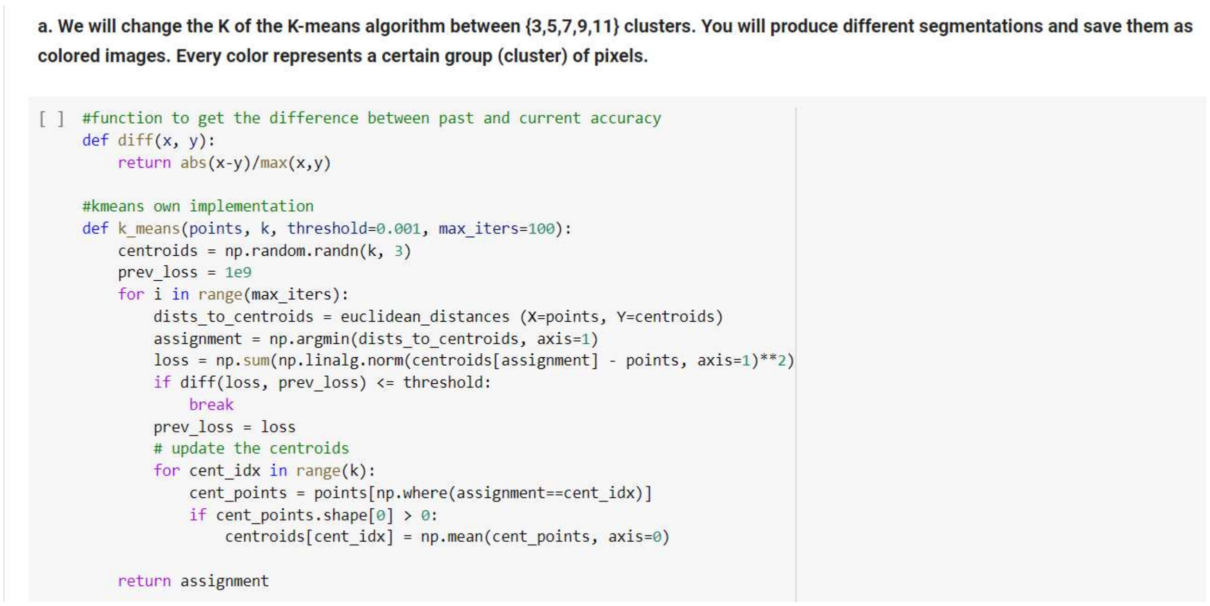
Image Visualization:





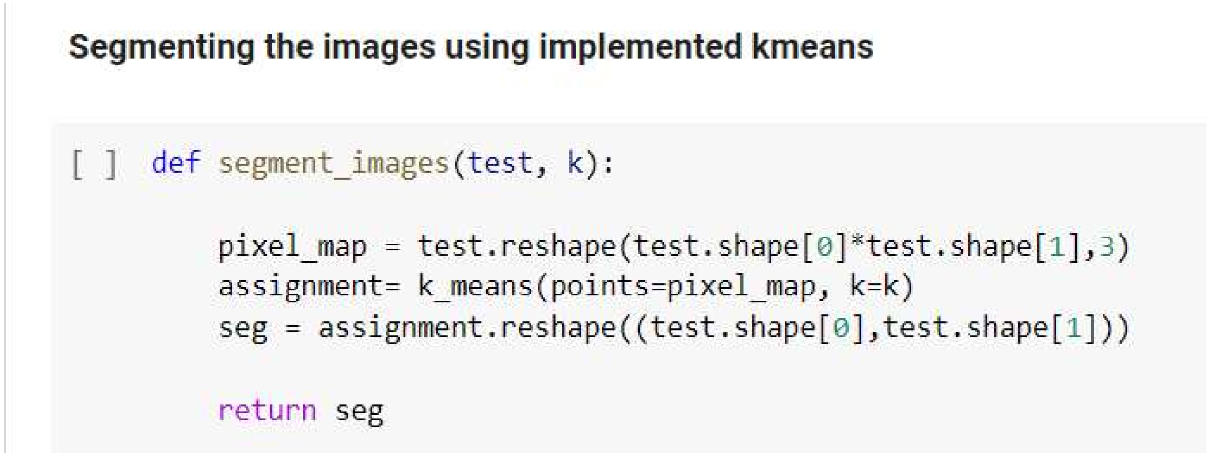
3.Segmentation using K-means

a. We will change the K of the K-means algorithm between {3,5,7,9,11} clusters. You will produce different segmentations and save them as colored images. Every color represents a certain group (cluster) of pixels



a. We will change the K of the K-means algorithm between {3,5,7,9,11} clusters. You will produce different segmentations and save them as colored images. Every color represents a certain group (cluster) of pixels.

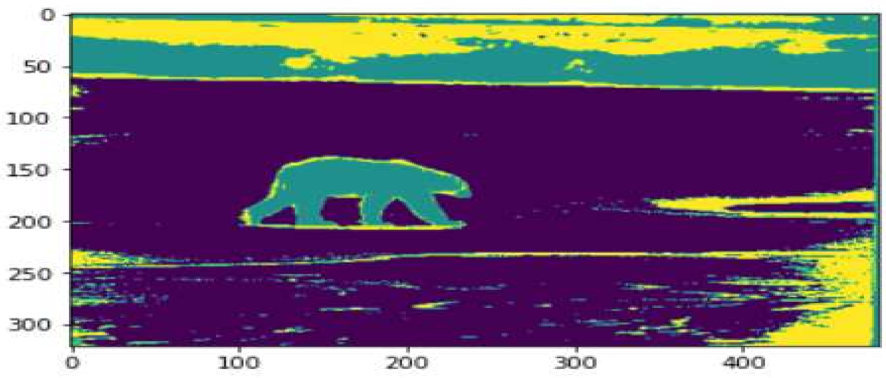
```
[ ] #function to get the difference between past and current accuracy
    def diff(x, y):
        return abs(x-y)/max(x,y)

    #kmeans own implementation
    def k_means(points, k, threshold=0.001, max_iters=100):
        centroids = np.random.randn(k, 3)
        prev_loss = 1e9
        for i in range(max_iters):
            dists_to_centroids = euclidean_distances (X=points, Y=centroids)
            assignment = np.argmin(dists_to_centroids, axis=1)
            loss = np.sum(np.linalg.norm(centroids[assignment] - points, axis=1)**2)
            if diff(loss, prev_loss) <= threshold:
                break
            prev_loss = loss
            # update the centroids
            for cent_idx in range(k):
                cent_points = points[np.where(assignment==cent_idx)]
                if cent_points.shape[0] > 0:
                    centroids[cent_idx] = np.mean(cent_points, axis=0)

        return assignment
```

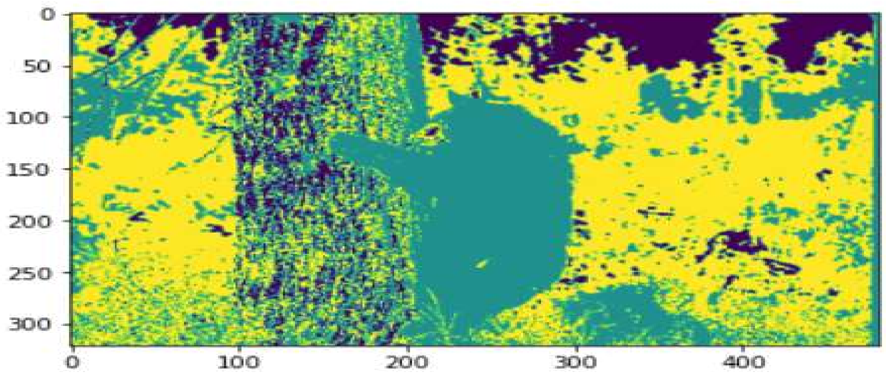


Segmenting the images using implemented kmeans

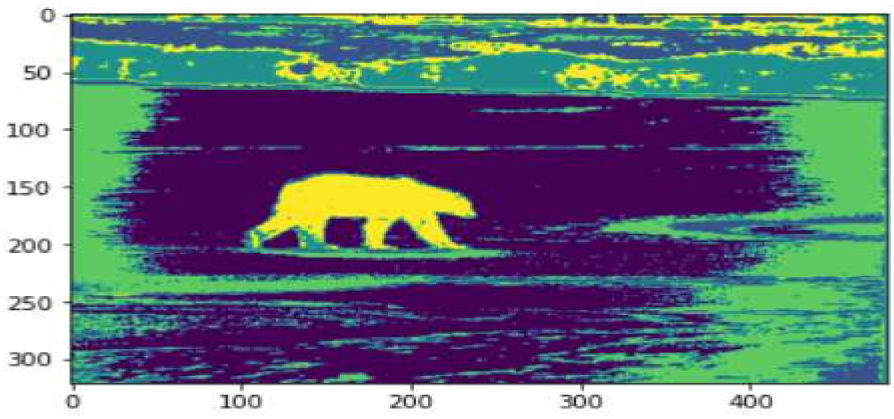
```
[ ] def segment_images(test, k):

        pixel_map = test.reshape(test.shape[0]*test.shape[1],3)
        assignment= k_means(points=pixel_map, k=k)
        seg = assignment.reshape((test.shape[0],test.shape[1]))

        return seg
```

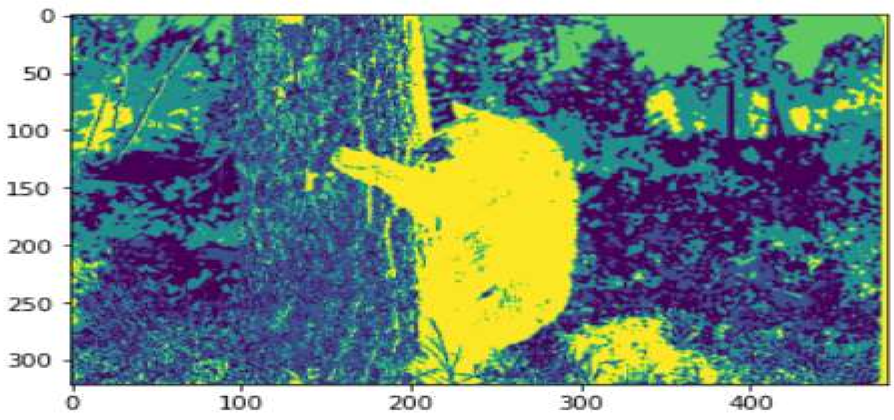
Segmentations results when k = 3:



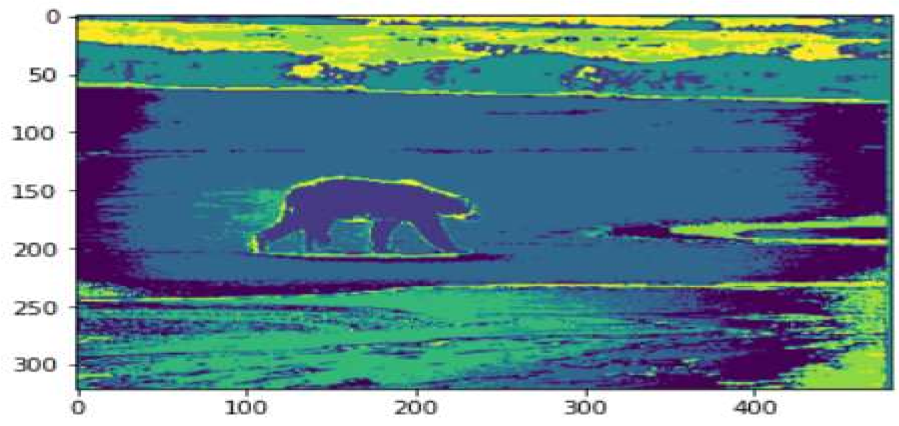


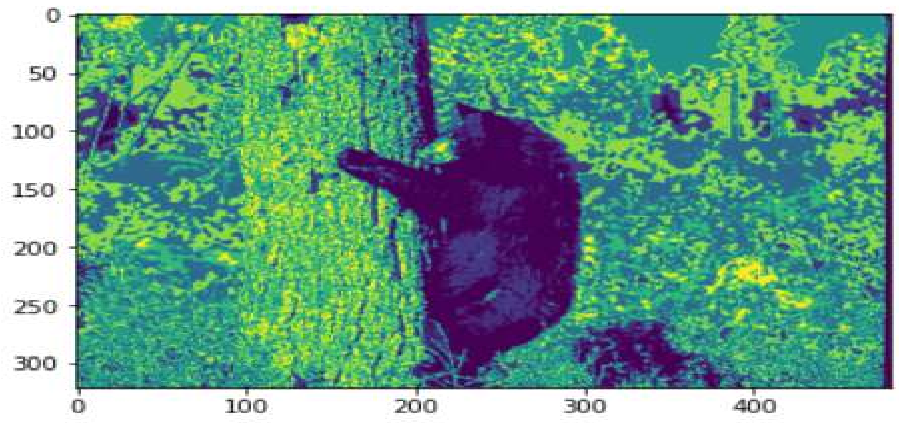
Segmentations results when k = 5:



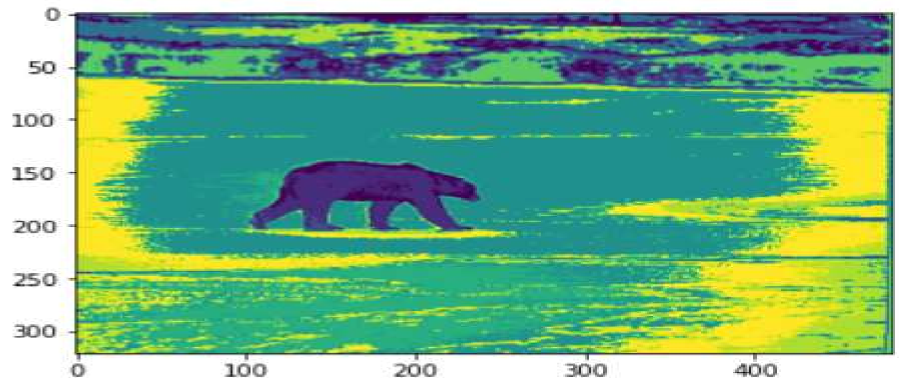


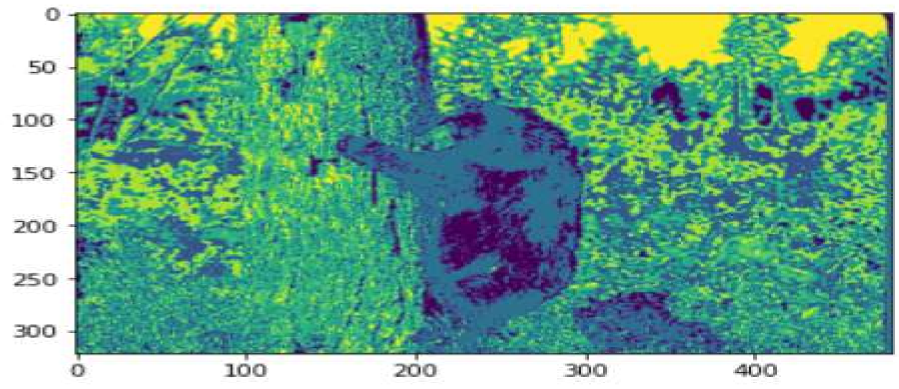
Segmentations results when k = 7:



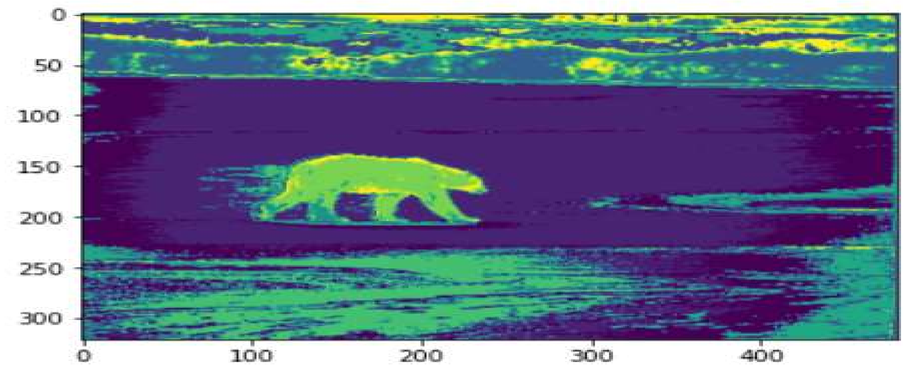


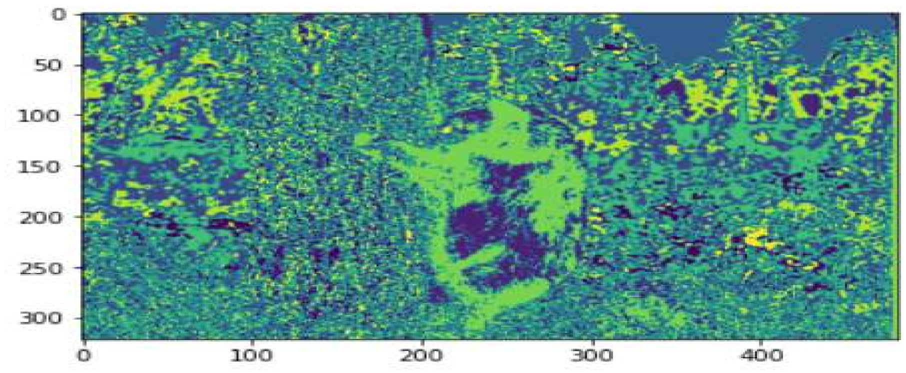
Segmentations results when k = 9:





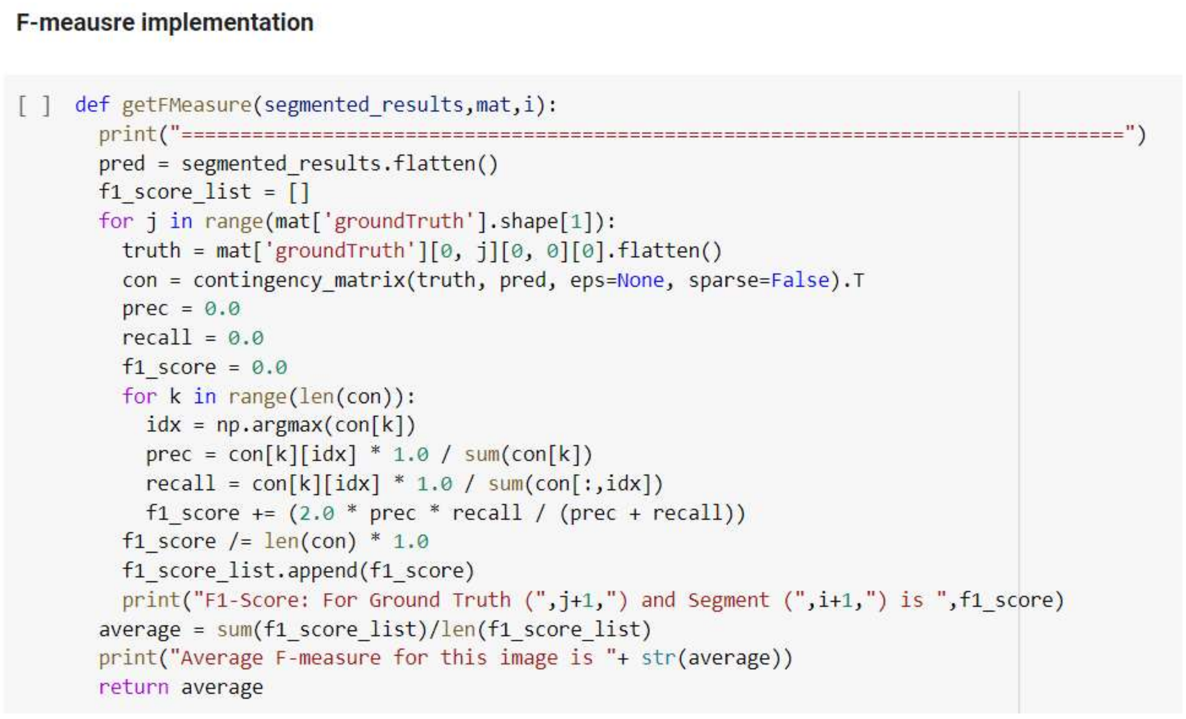
Segmentations results when k = 11:





b. We will evaluate the result segmentation using F-measure, Conditional Entropy for image I with M available ground-truth segmentations. For a clustering of K-clusters you will report your measures M times and the average of the M trials as well. Report average per dataset as well

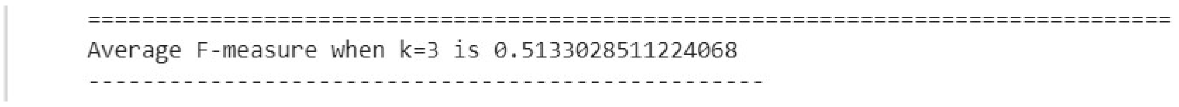
F-measure implementation:



F-meausre implementation

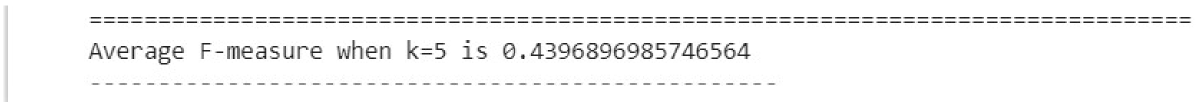
```
[ ] def getFMeasure(segmented_results,mat,i):
      print("================================================================================")
      pred = segmented_results.flatten()
      f1_score_list = []
      for j in range(mat['groundTruth'].shape[1]):
        truth = mat['groundTruth'][0, j][0, 0][0].flatten()
        con = contingency_matrix(truth, pred, eps=None, sparse=False).T
        prec = 0.0
        recall = 0.0
        f1_score = 0.0
        for k in range(len(con)):
          idx = np.argmax(con[k])
          prec = con[k][idx] * 1.0 / sum(con[k])
          recall = con[k][idx] * 1.0 / sum(con[:,idx])
          f1_score += (2.0 * prec * recall / (prec + recall))
        f1_score /= len(con) * 1.0
        f1_score_list.append(f1_score)
        print("F1-Score: For Ground Truth (",j+1,") and Segment (",i+1,") is ",f1_score)
      average = sum(f1_score_list)/len(f1_score_list)
      print("Average F-measure for this image is "+ str(average))
      return average
```

Average F-measure when k = 3:



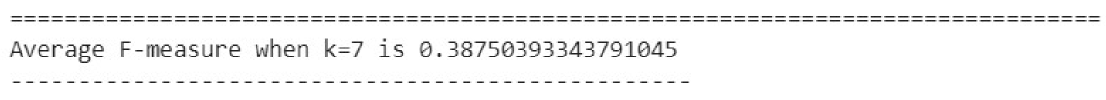
```
==========================================================================
Average F-measure when k=3 is 0.5133028511224068
----------------------------------------------
```

Average F-measure when k = 5:



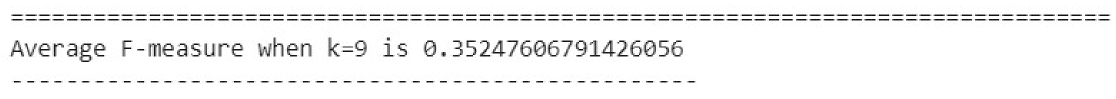
```
==========================================================================
Average F-measure when k=5 is 0.4396896985746564
----------------------------------------------
```

Average F-measure when k = 7:



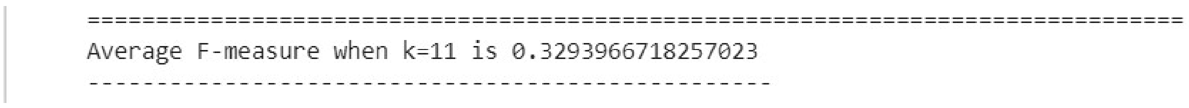
```
==========================================================================
Average F-measure when k=7 is 0.38750393343791045
----------------------------------------------
```

Average F-measure when k = 9:



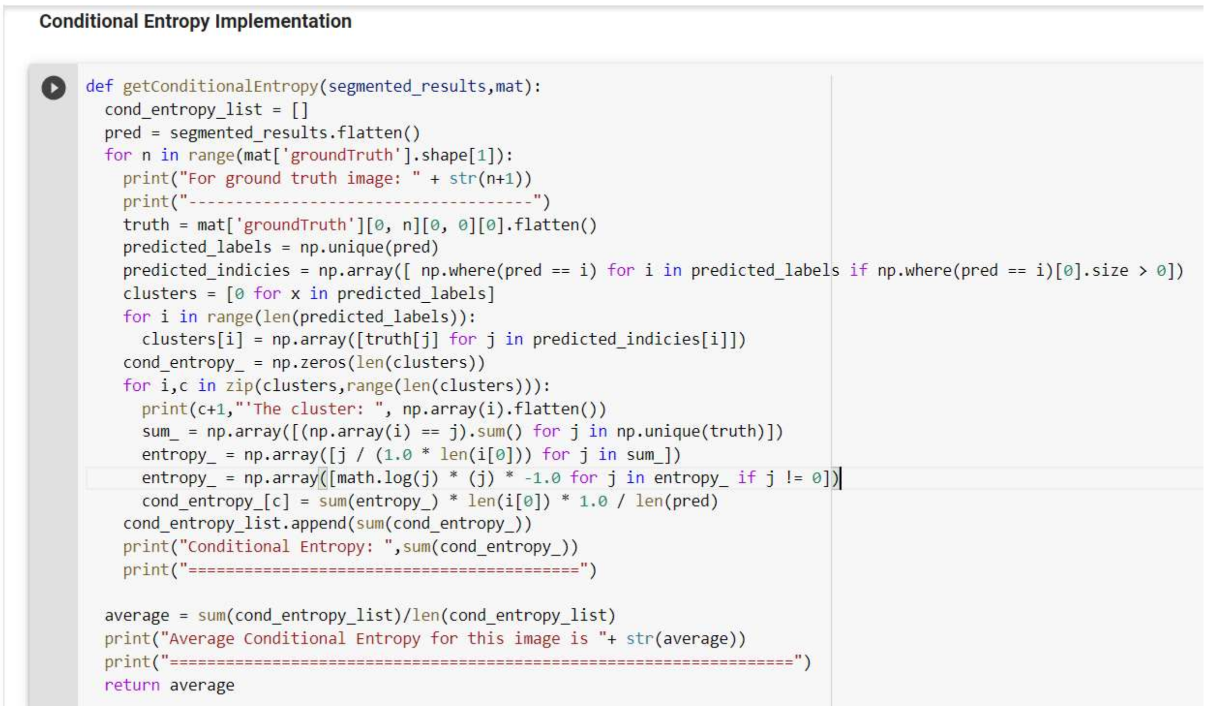
```
==========================================================================
Average F-measure when k=9 is 0.35247606791426056
----------------------------------------------
```

Average F-measure when k = 11:



```
==========================================================================
Average F-measure when k=11 is 0.3293966718257023
----------------------------------------------
```

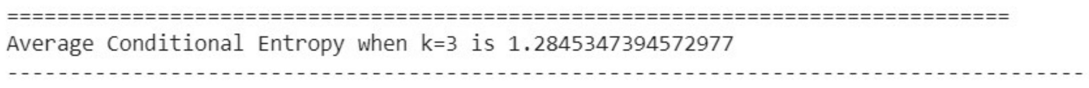
Conditional Entropy Implementation:



Conditional Entropy Implementation

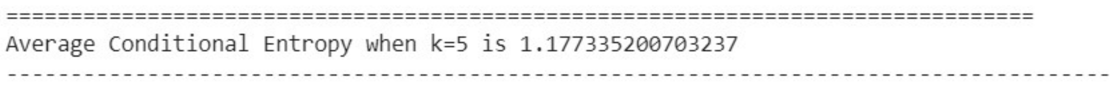
```
def getConditionalEntropy(segmented_results,mat):
  cond_entropy_list = []
  pred = segmented_results.flatten()
  for n in range(mat['groundTruth'].shape[1]):
    print("For ground truth image: " + str(n+1))
    print("------------------------------------")
    truth = mat['groundTruth'][0, n][0, 0][0].flatten()
    predicted_labels = np.unique(pred)
    predicted_indicies = np.array([ np.where(pred == i) for i in predicted_labels if np.where(pred == i)[0].size > 0])
    clusters = [0 for x in predicted_labels]
    for i in range(len(predicted_labels)):
      clusters[i] = np.array([truth[j] for j in predicted_indicies[i]])
    cond_entropy_ = np.zeros(len(clusters))
    for i,c in zip(clusters,range(len(clusters))):
      print(c+1,"'The cluster: ", np.array(i).flatten())
      sum_ = np.array([(np.array(i) == j).sum() for j in np.unique(truth)])
      entropy_ = np.array([j / (1.0 * len(i[0])) for j in sum_])
      entropy_ = np.array([math.log(j) * (j) * -1.0 for j in entropy_ if j != 0])
      cond_entropy_[c] = sum(entropy_) * len(i[0]) * 1.0 / len(pred)
    cond_entropy_list.append(sum(cond_entropy_))
    print("Conditional Entropy: ",sum(cond_entropy_))
    print("==========================================")

  average = sum(cond_entropy_list)/len(cond_entropy_list)
  print("Average Conditional Entropy for this image is "+ str(average))
  print("==================================================================")
  return average
```

Average Conditional Entropy when k = 3:



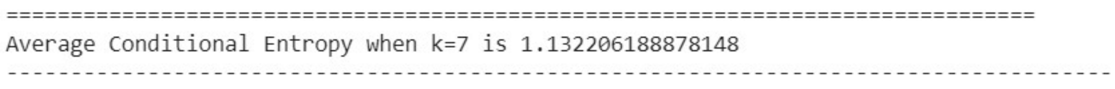
```
================================================================================
Average Conditional Entropy when k=3 is 1.2845347394572977
--------------------------------------------------------------------------------
```

Average Conditional Entropy when k = 5:



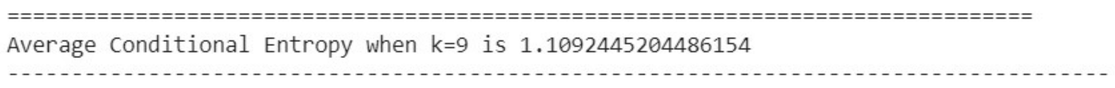
```
================================================================================
Average Conditional Entropy when k=5 is 1.177335200703237
--------------------------------------------------------------------------------
```

Average Conditional Entropy when k = 7:



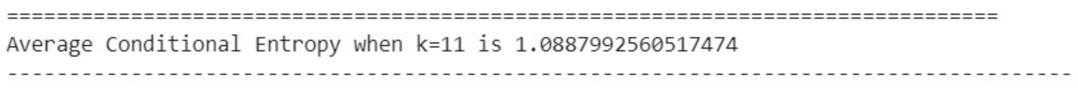
```
================================================================================
Average Conditional Entropy when k=7 is 1.132206188878148
--------------------------------------------------------------------------------
```

Average Conditional Entropy when k = 9:



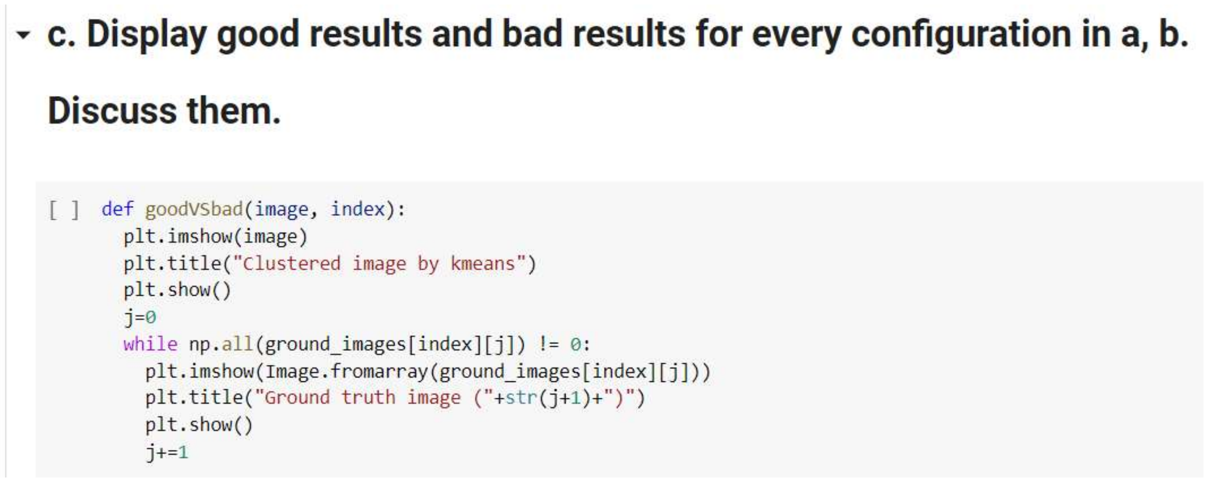
```
================================================================================
Average Conditional Entropy when k=9 is 1.1092445204486154
--------------------------------------------------------------------------------
```

Average Conditional Entropy when k = 11:



```
================================================================================
Average Conditional Entropy when k=11 is 1.0887992560517474
--------------------------------------------------------------------------------
```

c. Display good results and bad results for every configuration in a, b.



▾ c. Display good results and bad results for every configuration in a, b.

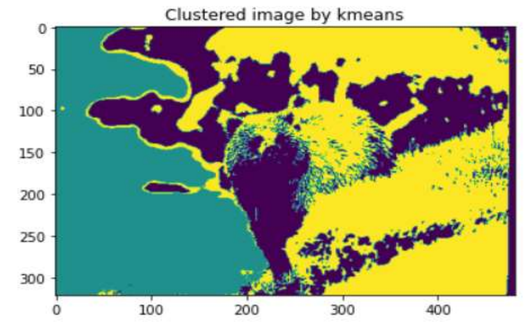
Discuss them.

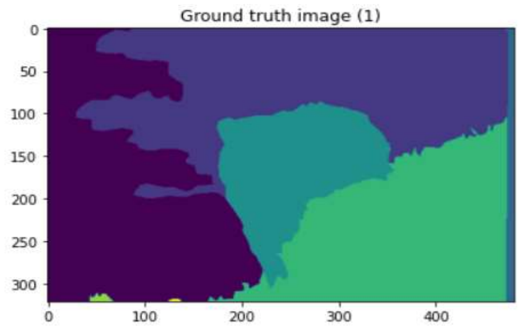
```
[ ]  def goodVSbad(image, index):
       plt.imshow(image)
       plt.title("Clustered image by kmeans")
       plt.show()
       j=0
       while np.all(ground_images[index][j]) != 0:
         plt.imshow(Image.fromarray(ground_images[index][j]))
         plt.title("Ground truth image ("+str(j+1)+")")
         plt.show()
         j+=1
```

F-measure:

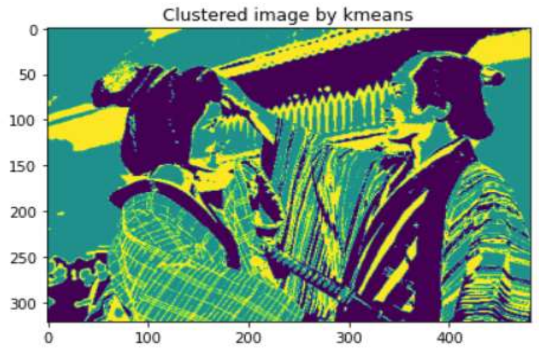
For k = 3:

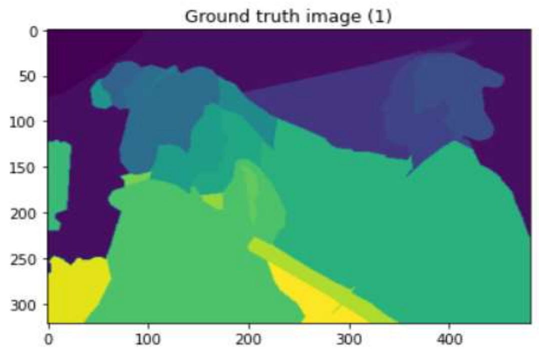
```
When k = 3:
=============
Good F-meausure (image #3) (high)
-----------------------------------------
```





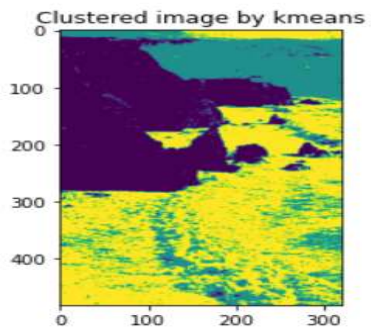
```
Bad F-measure (image #43) (low)
-------------------------------------------
```

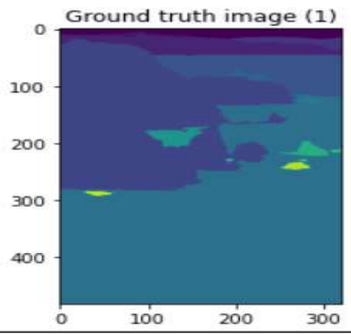




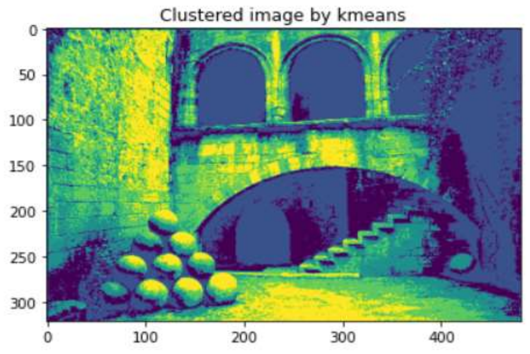
For k = 5:

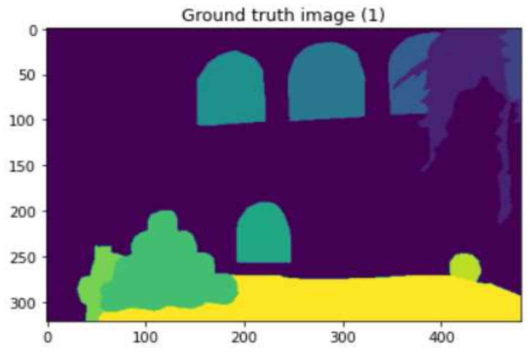
```
When k = 5:
=============
Good F-meausure (image #25) (high)
-----------------------------------------
```





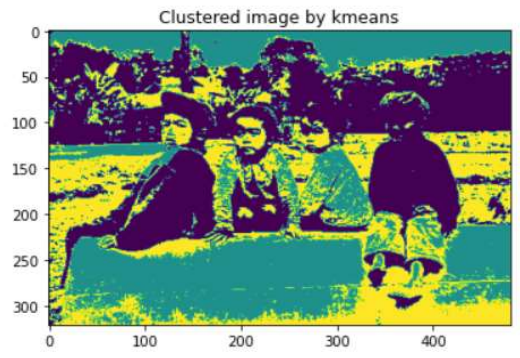
```
Bad F-measure (image #28) (low)
-------------------------------------------
```

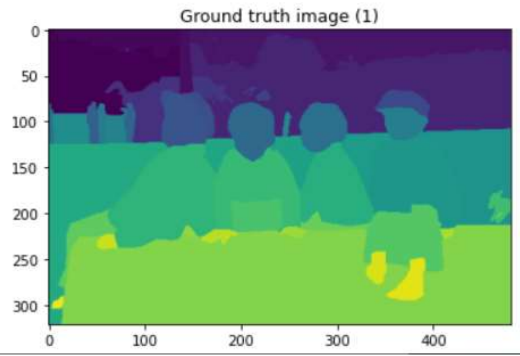




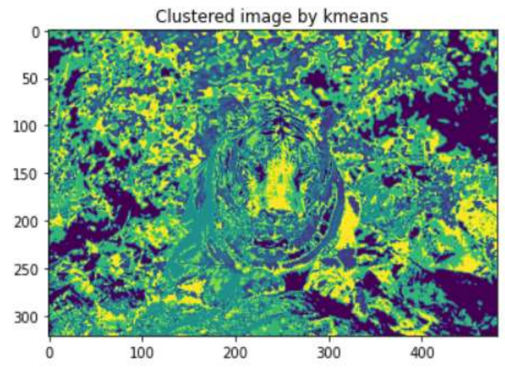
For k = 7:

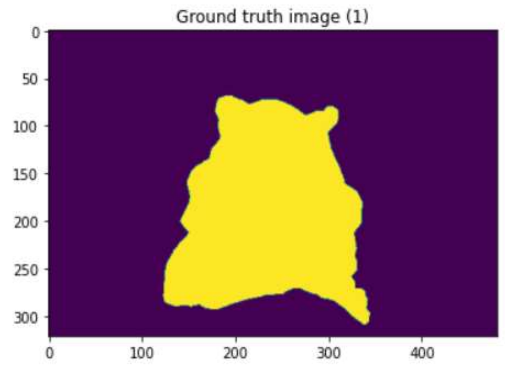
```
When k = 7:
=============
Good F-meausure (image #48) (high)
------------------------------------
```





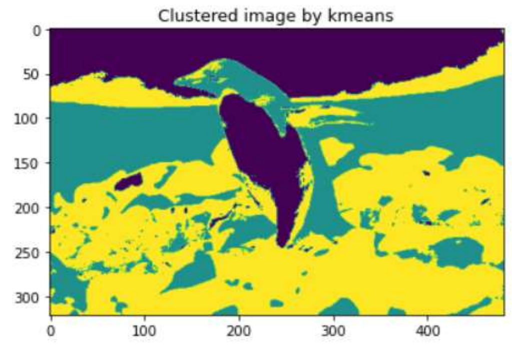
```
Bad F-measure (image #21) (low)
-----------------------------------
```

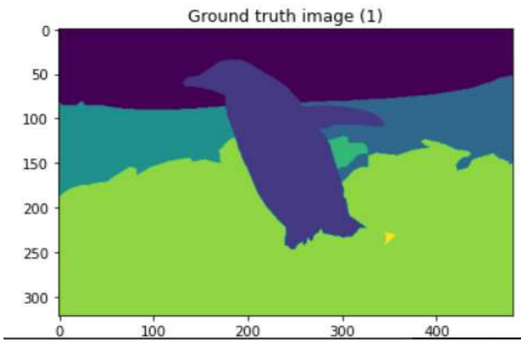




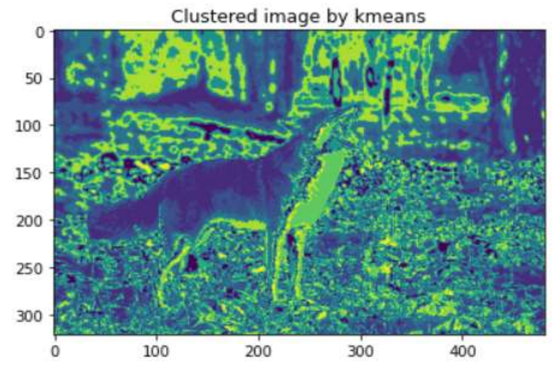
For k = 9:

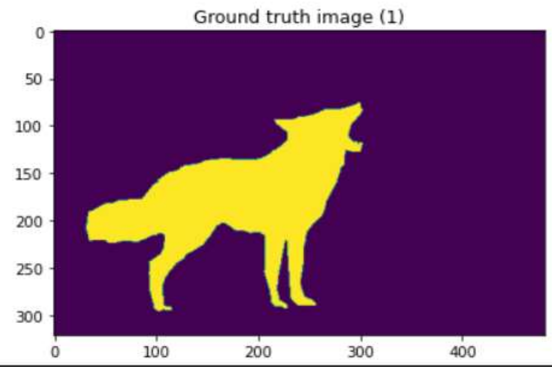
```
When k = 9:
=============
Good F-meausure (image #14) (high)
------------------------------------
```





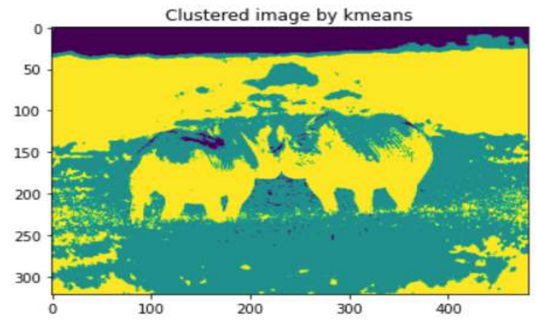
```
Bad F-measure (image #22) (low)
-------------------------------------
```

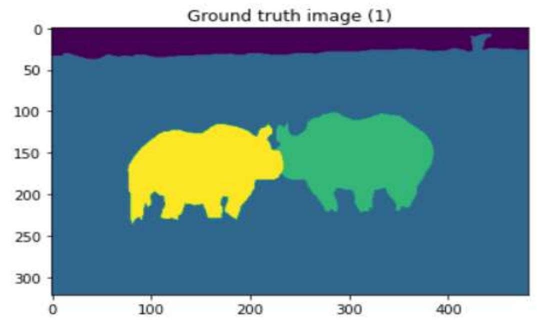




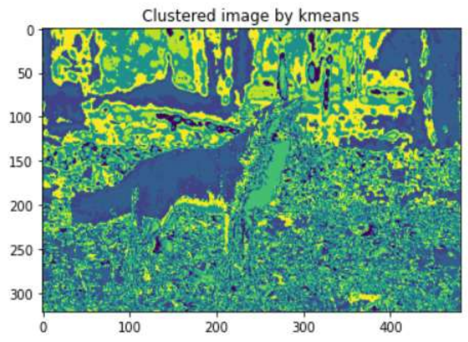
For k = 11:

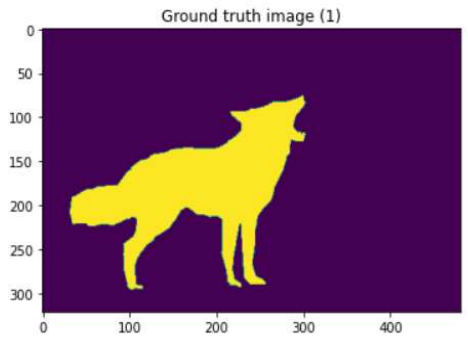
```
When k = 11:
=============
Good F-meausure (image #23) (high)
------------------------------------------
```





```
Bad F-measure (image #22) (low)
------------------------------------
```

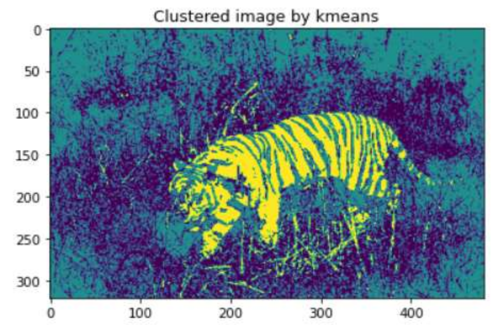


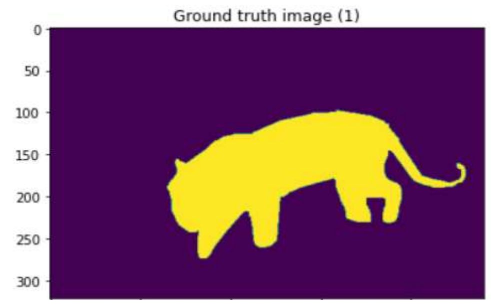


Conditional Entropy:

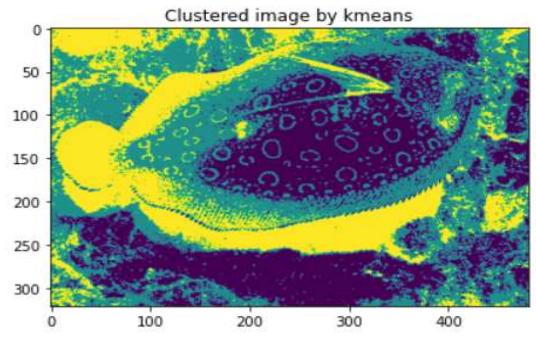
For k = 3:

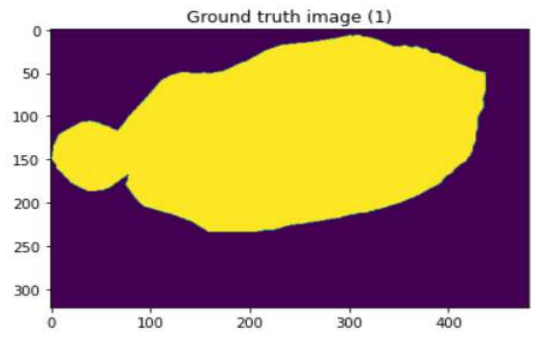
```
When k = 3:
=============
Good Conditional Entropy (image #19) (low)
-----------------------------------------------
```





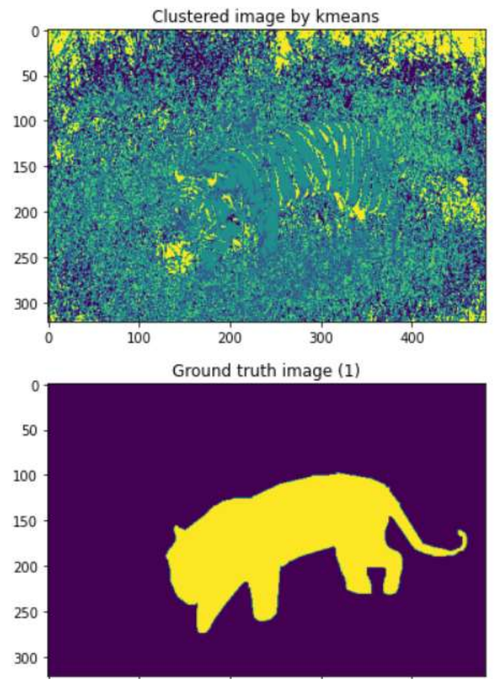
```
Bad Conditional Entropy (image #42) (high)
-----------------------------------------------------
```



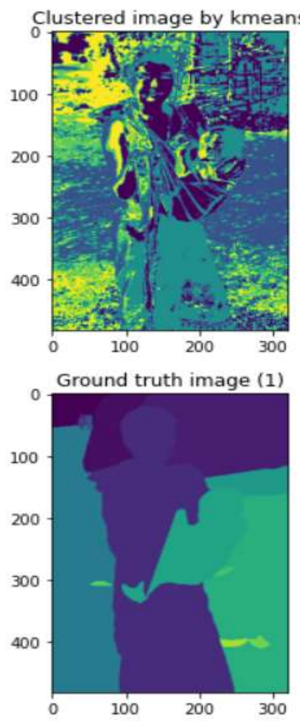


For k = 5:

```
When k = 5:
=============
Good Conditional Entropy (image #19) (low)
--------------------------------------------
```

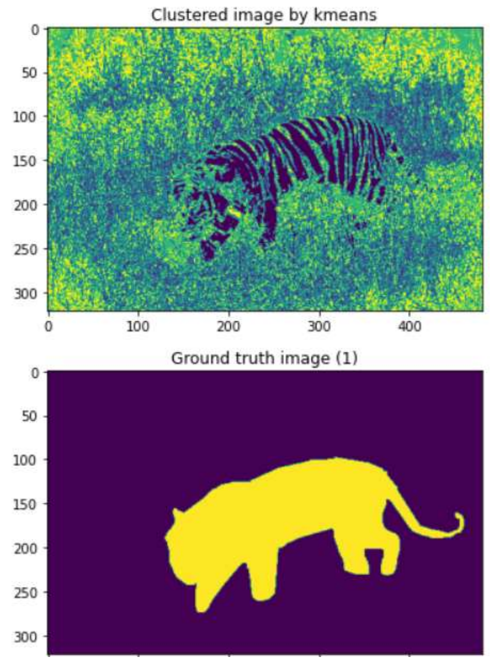


```
Bad Conditional Entropy (image #46) (high)
--------------------------------------------
```

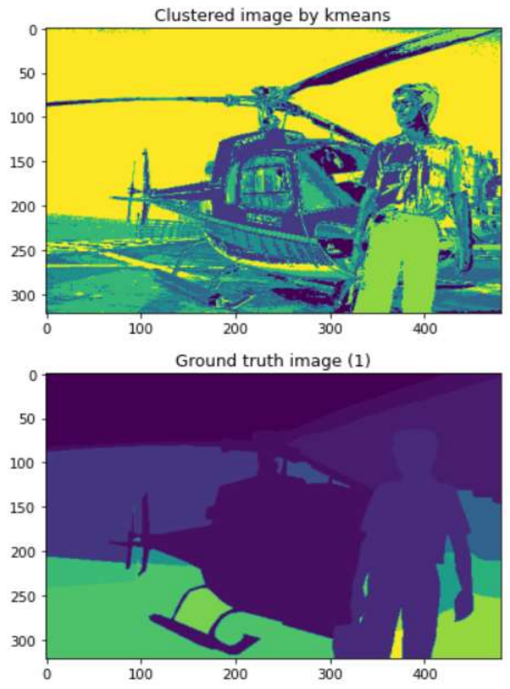


For k = 7:

```
When k = 7:
=============
Good Conditional Entropy (image #19) (low)
--------------------------------------------
```

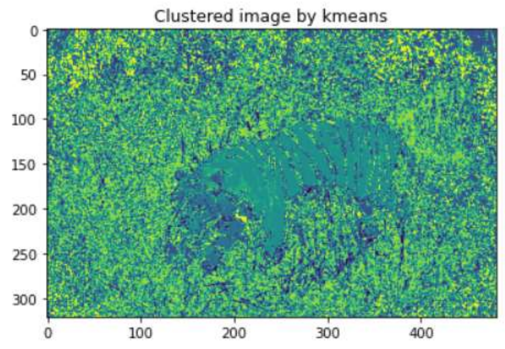


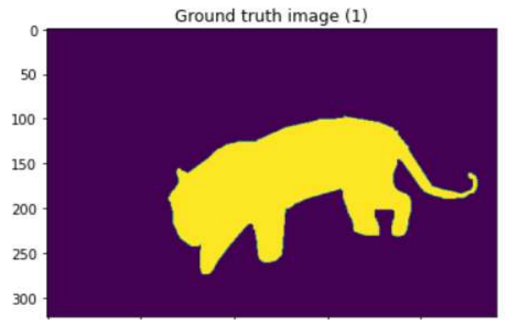
```
Bad Conditional Entropy (image #45) (high)
--------------------------------------------
```



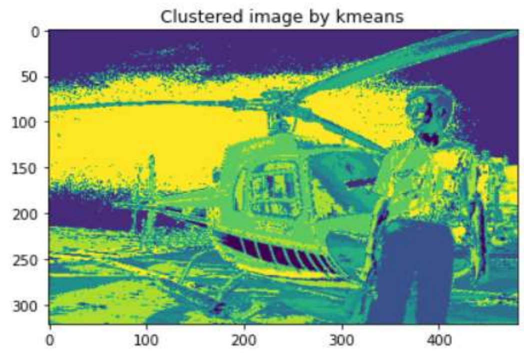
For k = 9:

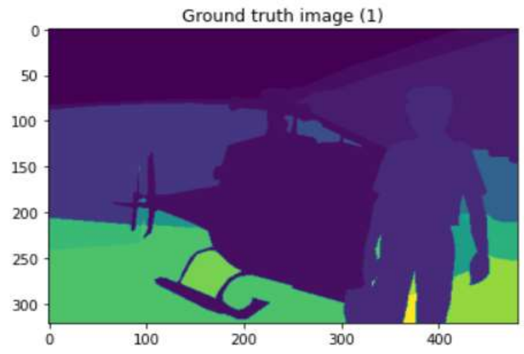
```
When k = 9:
=============
Good Conditional Entropy (image #19) (low)
---------------------------------------------
```





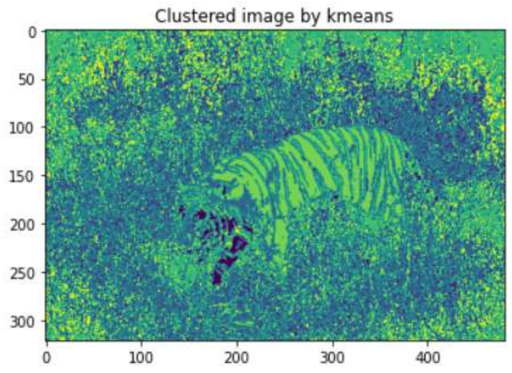
```
Bad Conditional Entropy (image #45) (high)
------------------------------------------------
```

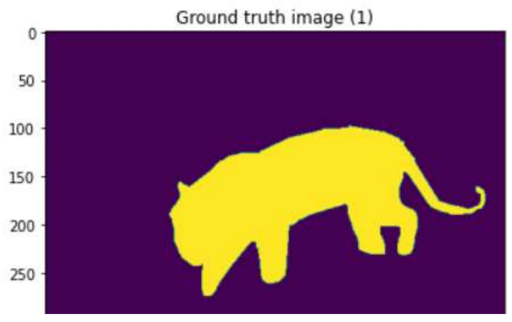




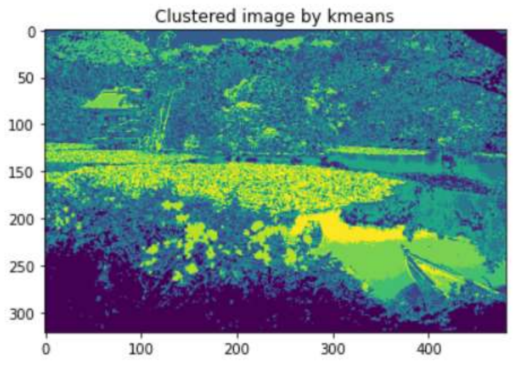
For k = 11:

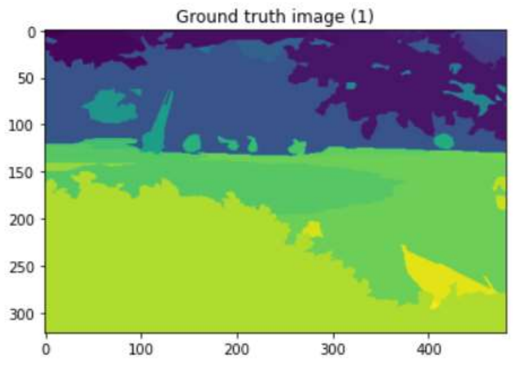
```
When k = 11:
=============
Good Conditional Entropy (image #19) (low)
---------------------------------------------
```





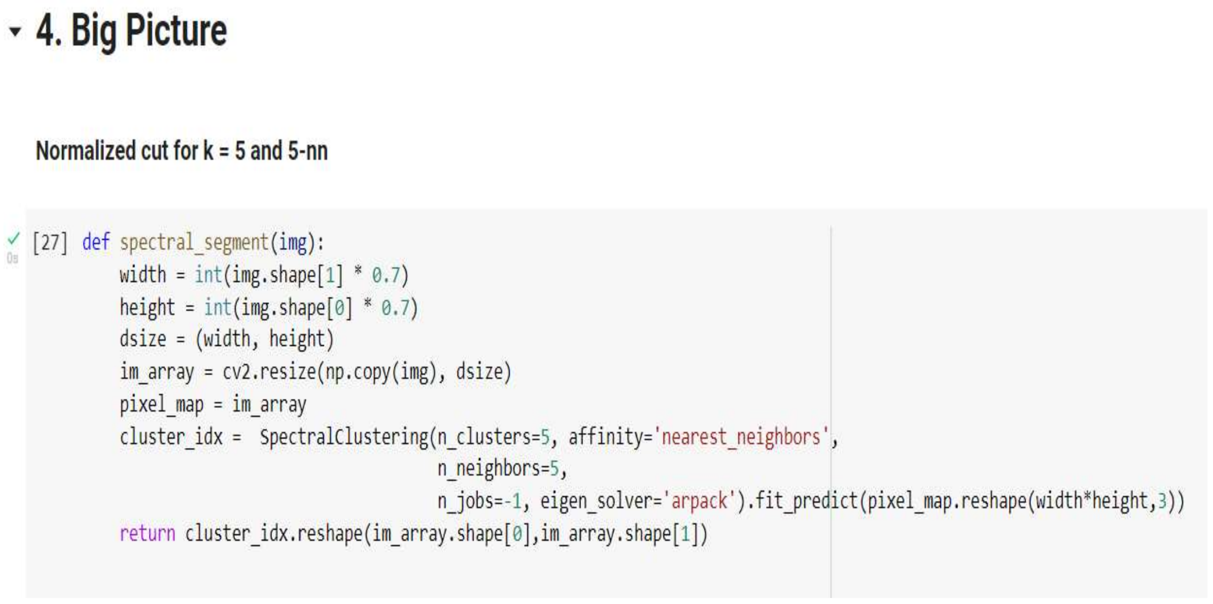
```
Bad Conditional Entropy (image #44) (high)
----------------------------------------------
```





4.Big Picture:

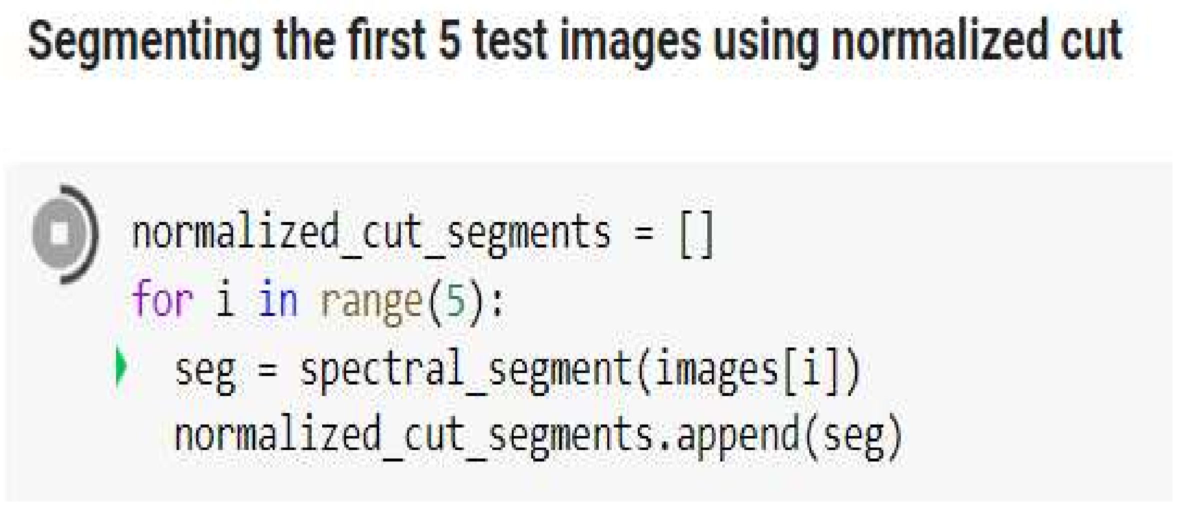
The built-in sklearn library spectral clustering was used and resizing of 0.7*original size of image is applied to make the segmentation a bit faster.



4. Big Picture

Normalized cut for k = 5 and 5-nn

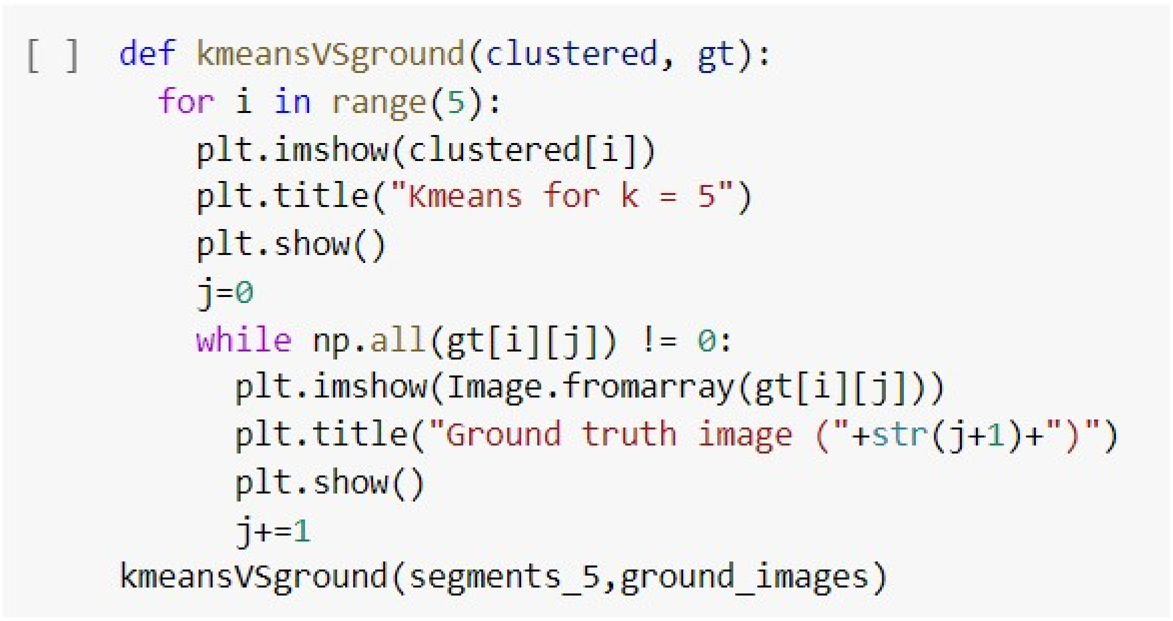
```
[27] def spectral_segment(img):
         width = int(img.shape[1] * 0.7)
         height = int(img.shape[0] * 0.7)
         dsize = (width, height)
         im_array = cv2.resize(np.copy(img), dsize)
         pixel_map = im_array
         cluster_idx =  SpectralClustering(n_clusters=5, affinity='nearest_neighbors',
                                           n_neighbors=5,
                                           n_jobs=-1, eigen_solver='arpack').fit_predict(pixel_map.reshape(width*height,3))
         return cluster_idx.reshape(im_array.shape[0],im_array.shape[1])
```



Segmenting the first 5 test images using normalized cut

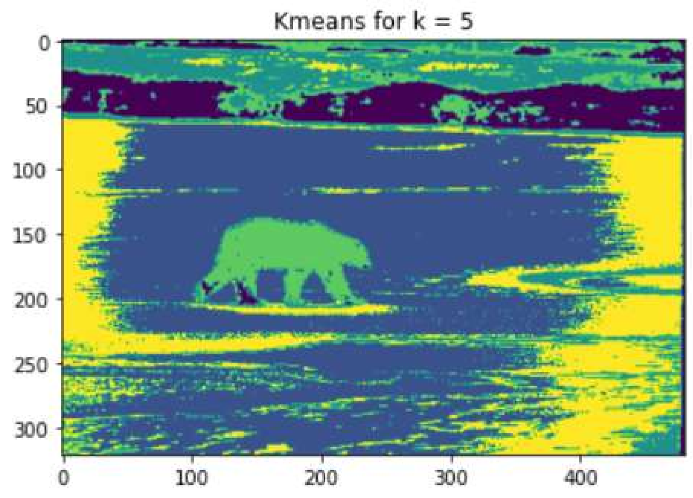
```
normalized_cut_segments = []
for i in range(5):
  seg = spectral_segment(images[i])
  normalized_cut_segments.append(seg)
```

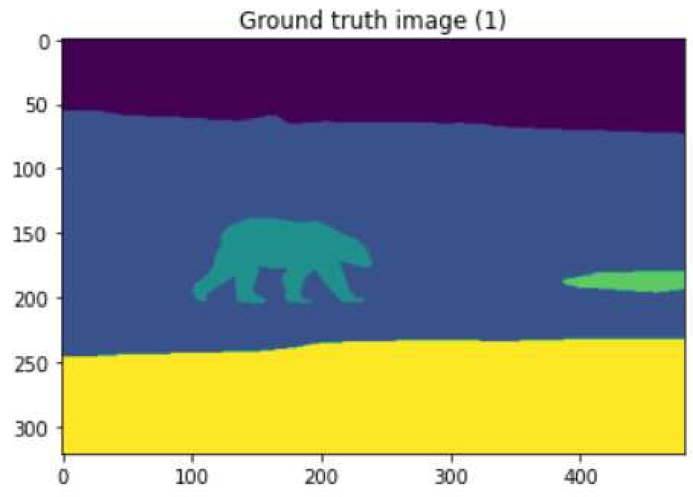
a. Select a set of five images and display their corresponding ground truth against your segmentation results using K-means at K=5.



```
[ ]  def kmeansVSground(clustered, gt):
       for i in range(5):
         plt.imshow(clustered[i])
         plt.title("Kmeans for k = 5")
         plt.show()
         j=0
         while np.all(gt[i][j]) != 0:
           plt.imshow(Image.fromarray(gt[i][j]))
           plt.title("Ground truth image ("+str(j+1)+")")
           plt.show()
           j+=1
     kmeansVSground(segments_5,ground_images)
```

Image Visualization:

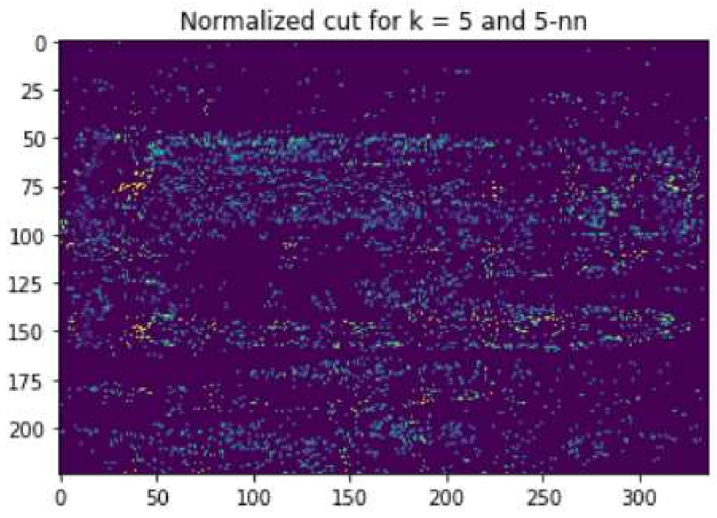


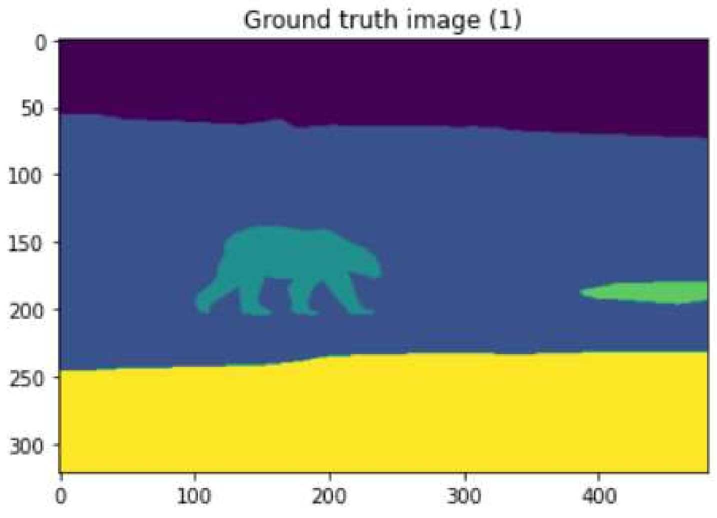


b. Select the same five images and display their corresponding ground truth against your segmentation results using Normalized-cut for the 5-NN graph, at K=5

```
def normalized_cutVSground(normalized, gt):
  for i in range(5):
    plt.imshow(normalized[i])
    plt.title("Normalized cut for k = 5 and 5-nn")
    plt.show()
    j=0
    while np.all(gt[i][j]) != 0:
      plt.imshow(Image.fromarray(gt[i][j]))
      plt.title("Ground truth image ("+str(j+1)+")")
      plt.show()
      j+=1

normalized_cutVSground(normalized_cut_segments,ground_images)
```

Image Visualization:

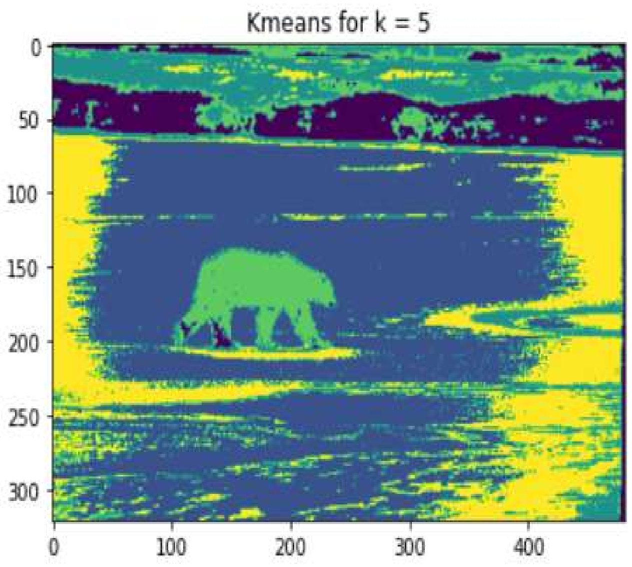


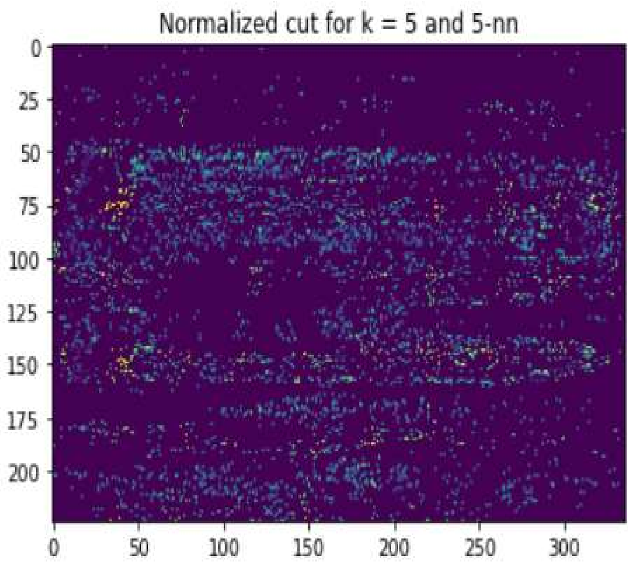


c. Select the same five images and contrast your segmentation results using Normalized-cut for the 5-NN graph, at K=5 versus using K-means at K=5.

```
[ ]  def kmeansVSnormalized_cut(clustered, normalized):
       for i in range(5):
          plt.imshow(clustered[i])
          plt.title("Kmeans for k = 5")
          plt.show()
          plt.imshow(normalized[i])
          plt.title("Normalized cut for k = 5 and 5-nn")
          plt.show()

     kmeansVSnormalized_cut(segments_5,normalized_cut_segments)
```

Image Visualization:





Conclusions:

- It was noted that the image becomes less clear when we increase the number of clusters.
- It was noted that the higher the f-measure, the better the quality of the clustering is.
- It was noted that the lower the conditional entropy, the better the quality of the clustering is.
- It was noted that at K = 5 clusters, the images are somehow near the ground truth images we compare our segmentation with.
- It was noted that the normalized cut at K = 5 clusters and 5-nn failed to segment the image and a lot of pixels were lost.
- It was noted that k-means clustering segmented the images relatively better than the normalized cut. However, they're both not good enough.