Machine Learning 108-1

HW6

0856733 數據一 黃明翰

Data(image) Preparation

Data (image)

```
im = imageio.imread('image1.png').astype('float32')
im2 = imageio.imread('image2.png').astype('float32')
```

In use ***imageio*** to read image date in both algorithms, which can read image as array (100x100).

Similarity matrix (W)

```
W1 = rbf_kernel(im,1)
W2 = rbf_kernel(im2,2)
```

We will use similarity matrix in both algorithms for two input images. The matrix is made up by given **RBF kernel**

```
def rbf_kernel(X,image_number):
    sigma_s = -0.02
    sigma_c = -0.02
    W = np.zeros((10000,10000))
    for i in range(10000):
        x1 = i//100
        y1 = i%100
        for j in range(i+1,10000):
            x2 = j//100
            y2 = j%100
            if image_number==1:
                w = np.exp(sigma_s*((x1-x2)**2+(y1-y2)**2))*np.exp(sigma_c*(np.linalg.norm(im[x1,y1]-im[x2,y2])**2))
            else:
                w = np.exp(sigma_s*((x1-x2)**2+(y1-y2)**2))*np.exp(sigma_c*(np.linalg.norm(im2[x1,y1]-im2[x2,y2])**2))
            W[i,j] = W[j,i] = w
    return W
```

which is equivalent to

$$k(x,x') = e^{-\gamma_s \|S(x)-S(x')\|^2} \times e^{-\gamma_c \|C(x)-C(x')\|^2}$$

Where $S(x)$ is the spatial information and $C(x)$ is the color information.

Kernel K-means

Initial Methods

I tried two different initial methods.

1. **Random assign**: randomly assign data point into class(0~k-1)

```
def init(method,k,image_number):
    if method == 0:
        print('Init method: random')
        return np.random.randint(k,size=10000)
```

2. **K-means++**: from Line 26 – Line 58
   In this method, I'll explain in pseudocode followed:
   A. Randomly choose a data point as the first cluster
   B. Calculate distances of each points to the closest clusters
   C. Turn distance into probability and choose next cluster by probability(The farther the higher probability to be choosed)
   D. Repeat B-C till find k cluster.
   E. (from Line 47)Assign each point into their closet cluster.

Clustering

After initial, I use the kernel data (similarity matrix) and the initial information to cluster data points into k(2~4) clusters.

```
def classify(kernel_data,res_prev,k):
    res = np.zeros(10000)
    unique, counts = np.unique(res_prev, return_counts=True)
    third = third_term(kernel_data,res_prev,k)
    for j in range(10000):
        temp = np.zeros(k)
        for c in range(k):
            temp[c]+=(0-2*second_term(kernel_data,res_prev,j,c)/counts[c]+third[c]/counts[c]**2)
        res[j] = np.argmin(temp)
    return res
```

First, calculate the distance from kernel data to the means:

$$\left\|\emptyset(x_j) - \mu_k^{\emptyset}\right\| = k(x_j, x_j) - \frac{2}{|C_k|}\sum_{n} \alpha_{kn} k(x_j, x_n) + \frac{1}{|C_k|^2}\sum_{p}\sum_{q} \alpha_{kp}\alpha_{kq} k(x_p, x_q)$$

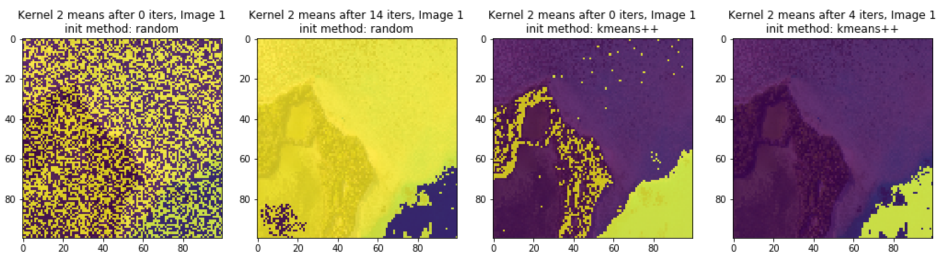
Then cluster each point into closest cluster use ***argmin.***

Repeat this step until the difference in clustering result between two iteration converges. (By experiment, I set the stopping threshold < 50)

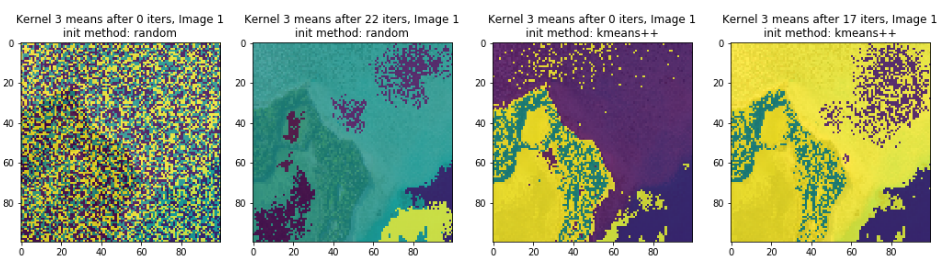
Results

- Image 1

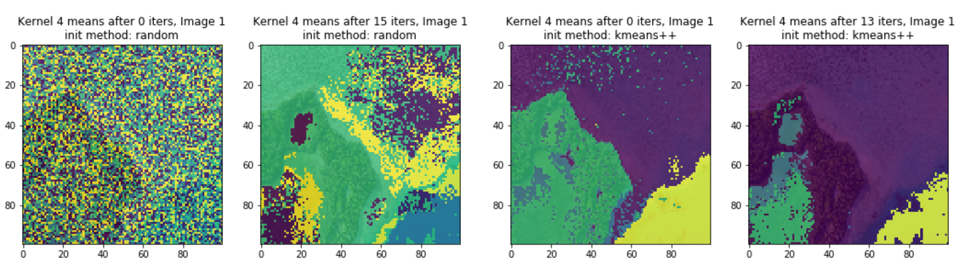
  - K=2



  - K=3

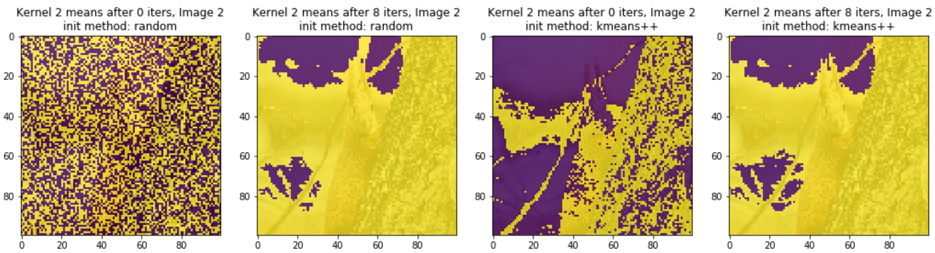


  - K=4

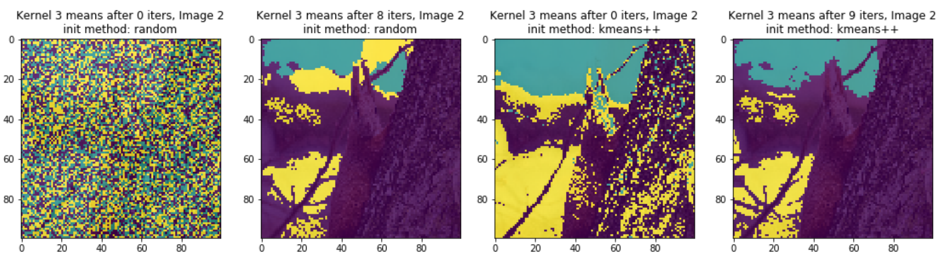


- Image 2

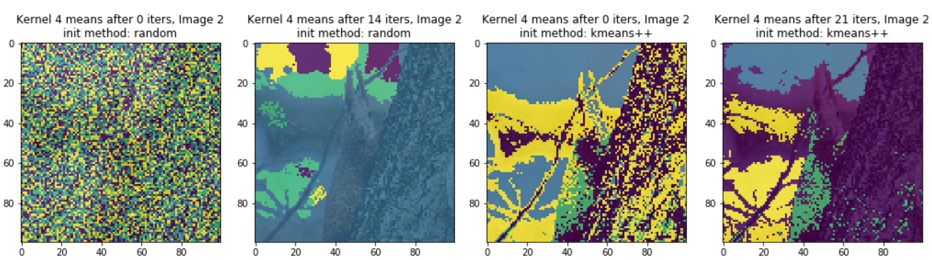
  - K=2



  - K=3



  - K=4



From the result we can see that in both input image, if we use kmeans++ as the initial method, since the starting center is far enough, it needs lesser time to converge and get more precise results.

Spectral Clustering

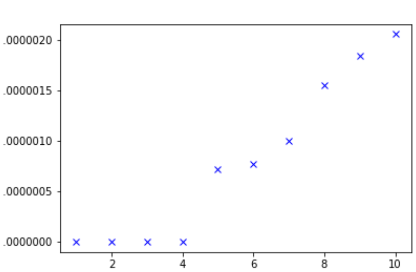
Ratio-Cut

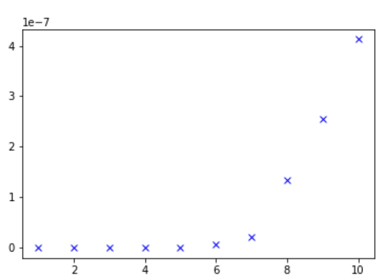
To approximate the minimum of ratio-cut method, we can use unnormalized Laplacian ***L=D-W*** to approach, where ***D*** is the degree matrix, ***W*** is the similarity matrix.

```
D1 = np.diag(np.sum(W1,axis=1))
D2 = np.diag(np.sum(W2,axis=1))
L1 = D1-W1
L2 = D2-W2
```

Then we find the eigenvalues and eigenvectors of ***L***, sort eigenvalues to find the top k+1 small eigenvalues and their eigenvectors. Since we use ***RBF kernel*** here, which cause the fully-connected graph, we start from the 2nd smallest eigenvalue and its eigenvector. Let ***U*** be the matrix containing these k eigenvectors.

```
# Ratio-Cut
eigen_values_1, eigen_vectors_1 = np.linalg.eig(L1)
eigen_values_2, eigen_vectors_2 = np.linalg.eig(L2)
eigen_vectors_1_sort = eigen_vectors_1[:,np.argsort(eigen_values_1)]
eigen_vectors_2_sort = eigen_vectors_2[:,np.argsort(eigen_values_2)]
for k in range(2,5):
    U1 = (eigen_vectors_1_sort[:,1:k+1])
    U2 = (eigen_vectors_2_sort[:,1:k+1])
    k_means(k,U1,1,'ratio cut')
    k_means(k,U2,2,'ratio cut')
```





(Sorted eigen-value of image1 and image2)

Normal-Cut

We can use normalized Laplacian $L_{sym} = D^{-1/2}LD^{-1/2}$ to approximate the minimum of normalized-cut. Then we also start from the 2nd smallest eigenvalue and its eigenvector of L_{sym}, let

U be the matrix of these k eigenvectors.

```
# Normal-Cut
D1_inv = np.linalg.inv(D1**0.5)
D2_inv = np.linalg.inv(D2**0.5)
L1_sym = np.dot(np.dot(D1_inv,L1),D1_inv)
L2_sym = np.dot(np.dot(D2_inv,L2),D2_inv)
eigen_values_normal_1, eigen_vectors_normal_1 = np.linalg.eig(L1_sym)
eigen_values_normal_2, eigen_vectors_normal_2 = np.linalg.eig(L2_sym)
eigen_vectors_normal_1_sort = eigen_vectors_normal_1[:,np.argsort(eigen_values_normal_1)]
eigen_vectors_normal_2_sort = eigen_vectors_normal_2[:,np.argsort(eigen_values_normal_2)]
for k in range(2,5):
    U1_normal = (eigen_vectors_normal_1_sort[:,1:k+1])
    U2_normal = (eigen_vectors_normal_2_sort[:,1:k+1])
    k_means(k,U1_normal,1,'normal cut')
    k_means(k,U2_normal,2,'normal cut')
```

Then we cluster the vector corresponding to the ***rows of U*** into k clusters with ***k-means*** algorithm.

Initial Methods

I tried two different initial methods.

1. **Random assign**: randomly assign k data points to be the center

```
def init(U,method,k,image_number):
    means = np.zeros((k,k))
    if method == 0:
        print('Init method: random')
        temp = np.random.randint(10000,size=k)
        for i in range(k):
            means[i] = U[temp[i]]
```

2. **K-means++**: from Line 30 – Line 42
   In this method, I'll explain in pseudocode followed:
   A. Randomly choose a data point as the first center
   B. Calculate distances of each points to the closest center
   C. Turn distance into probability and choose next center by probability(The farther the higher probability to be choosed)
   D. Repeat B-C till find k centers.

K-means

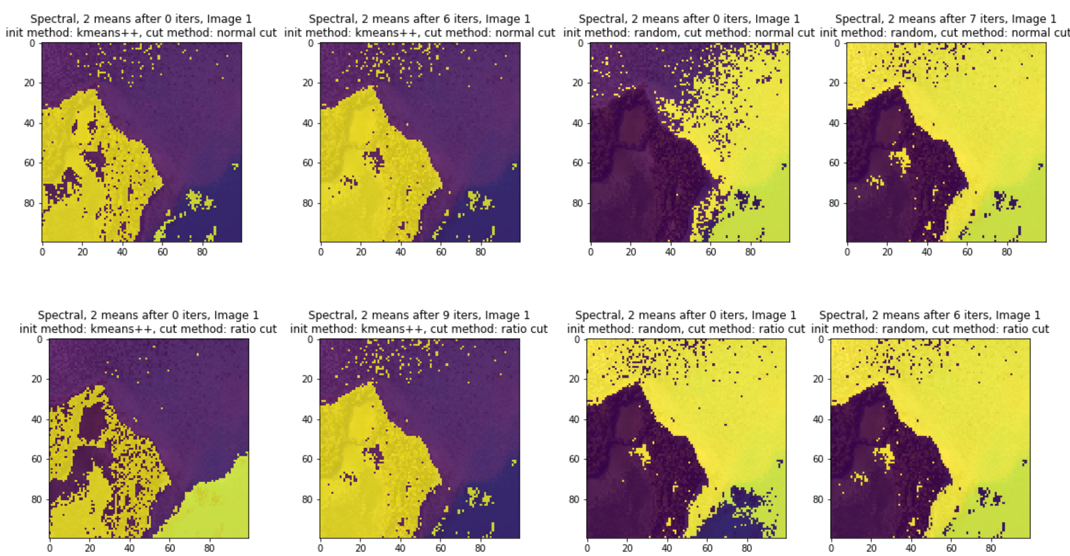
From Line 78 – Line 97, I implement the k-means algorithm:

1. Initial the centers using algorithm introduced aboved.
2. Cluster each point to their closest center (function classify, Line 45 – Line 52)
3. Update center according to the point in the corresponding cluster. (function update, Line 54 – Line 63)
4. Draw the result use matplotlib.

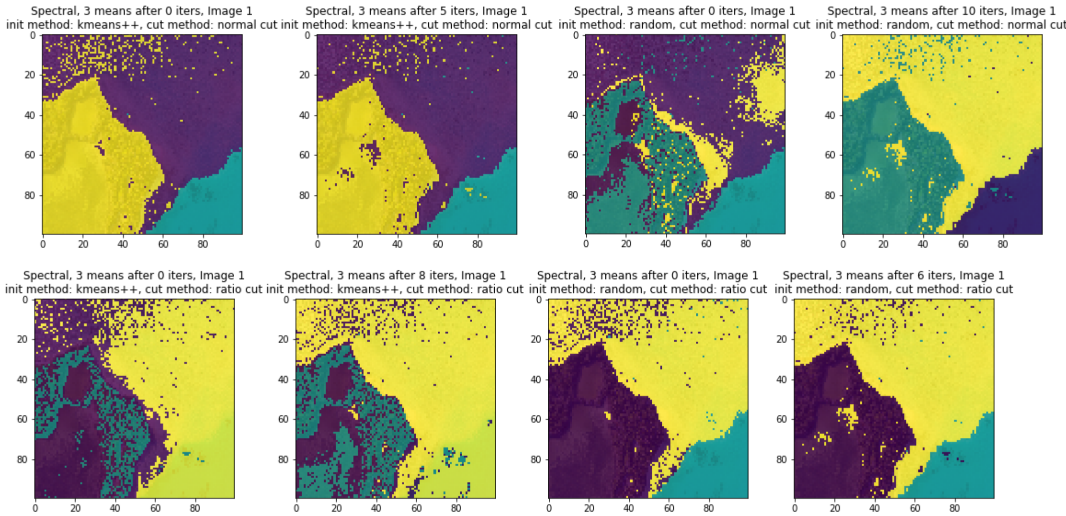
Results

First line: normal-cut
Second line: ratio-cut

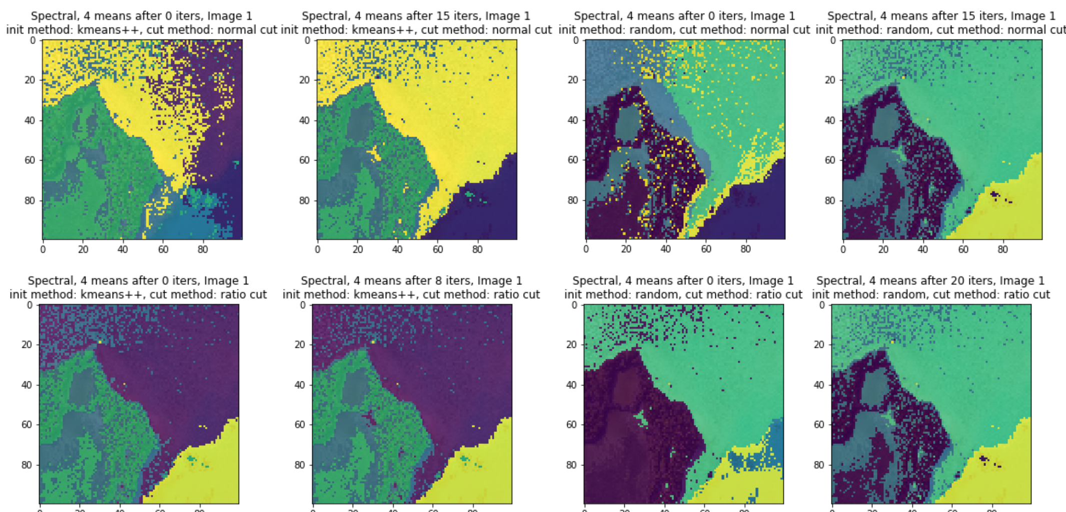
- Image 1
  - K=2



- K=3

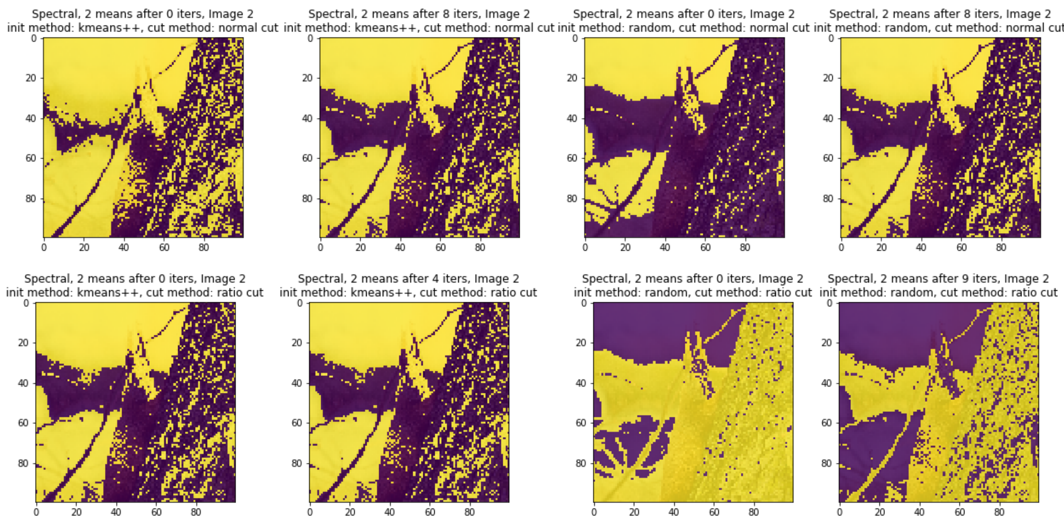

Spectral, 3 means after 0 iters, Image 1
init method: kmeans++, cut method: normal cut
Spectral, 3 means after 5 iters, Image 1
init method: kmeans++, cut method: normal cut
Spectral, 3 means after 0 iters, Image 1
init method: random, cut method: normal cut
Spectral, 3 means after 10 iters, Image 1
init method: random, cut method: normal cut
Spectral, 3 means after 0 iters, Image 1
init method: kmeans++, cut method: ratio cut
Spectral, 3 means after 8 iters, Image 1
init method: kmeans++, cut method: ratio cut
Spectral, 3 means after 0 iters, Image 1
init method: random, cut method: ratio cut
Spectral, 3 means after 6 iters, Image 1
init method: random, cut method: ratio cut

- K=4

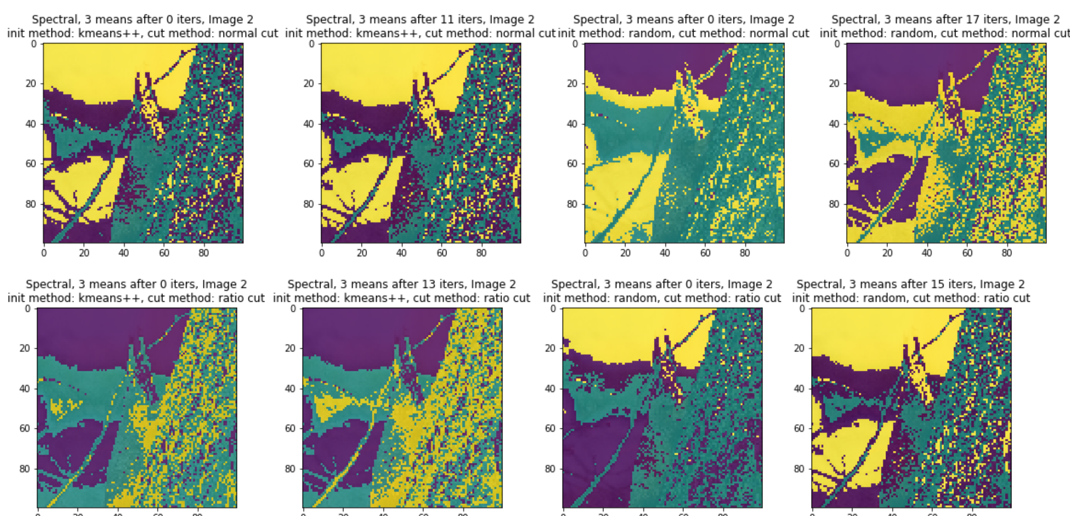

Spectral, 4 means after 0 iters, Image 1
init method: kmeans++, cut method: normal cut
Spectral, 4 means after 15 iters, Image 1
init method: kmeans++, cut method: normal cut
Spectral, 4 means after 0 iters, Image 1
init method: random, cut method: normal cut
Spectral, 4 means after 15 iters, Image 1
init method: random, cut method: normal cut
Spectral, 4 means after 0 iters, Image 1
init method: kmeans++, cut method: ratio cut
Spectral, 4 means after 8 iters, Image 1
init method: kmeans++, cut method: ratio cut
Spectral, 4 means after 0 iters, Image 1
init method: random, cut method: ratio cut
Spectral, 4 means after 20 iters, Image 1
init method: random, cut method: ratio cut

- Image 2

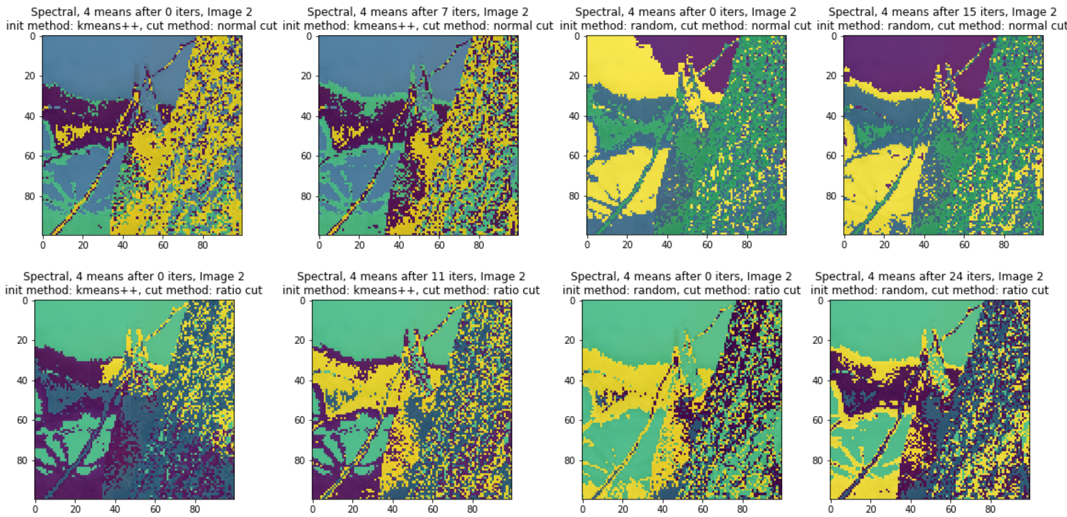
  - K=2


Spectral, 2 means after 0 iters, Image 2
init method: kmeans++, cut method: normal cut
Spectral, 2 means after 8 iters, Image 2
init method: kmeans++, cut method: normal cut
Spectral, 2 means after 0 iters, Image 2
init method: random, cut method: normal cut
Spectral, 2 means after 8 iters, Image 2
init method: random, cut method: normal cut
Spectral, 2 means after 0 iters, Image 2
init method: kmeans++, cut method: ratio cut
Spectral, 2 means after 4 iters, Image 2
init method: kmeans++, cut method: ratio cut
Spectral, 2 means after 0 iters, Image 2
init method: random, cut method: ratio cut
Spectral, 2 means after 9 iters, Image 2
init method: random, cut method: ratio cut

  - K=3


Spectral, 3 means after 0 iters, Image 2
init method: kmeans++, cut method: normal cut
Spectral, 3 means after 11 iters, Image 2
init method: kmeans++, cut method: normal cut
Spectral, 3 means after 0 iters, Image 2
init method: random, cut method: normal cut
Spectral, 3 means after 17 iters, Image 2
init method: random, cut method: normal cut
Spectral, 3 means after 0 iters, Image 2
init method: kmeans++, cut method: ratio cut
Spectral, 3 means after 13 iters, Image 2
init method: kmeans++, cut method: ratio cut
Spectral, 3 means after 0 iters, Image 2
init method: random, cut method: ratio cut
Spectral, 3 means after 15 iters, Image 2
init method: random, cut method: ratio cut

- K=4


Spectral, 4 means after 0 iters, Image 2
init method: kmeans++, cut method: normal cut
Spectral, 4 means after 7 iters, Image 2
init method: kmeans++, cut method: normal cut
Spectral, 4 means after 0 iters, Image 2
init method: random, cut method: normal cut
Spectral, 4 means after 15 iters, Image 2
init method: random, cut method: normal cut
Spectral, 4 means after 0 iters, Image 2
init method: kmeans++, cut method: ratio cut
Spectral, 4 means after 11 iters, Image 2
init method: kmeans++, cut method: ratio cut
Spectral, 4 means after 0 iters, Image 2
init method: random, cut method: ratio cut
Spectral, 4 means after 24 iters, Image 2
init method: random, cut method: ratio cut