In this part, I want to introduce the basic idea for Kernel k mean and spectral clustering.

kernel k-mean

To the story of the kernel k-mean, first we need to explain the basic idea of k-mean, the target of the k-mean is that we have n point (vector) in some dimension space, and we want to divide into k parts, in another words is we want to find k clustering of those data point. Intuitive idea is that we can first choose k initial points as center, then we have two-part, E part and M part. E part let all the others point join the cluster that has minimize distance to the centers. M part for each cluster we want to select the new center of it. Repeat these two parts until convergence or epoch finish. But sometimes problem is more complicated, that k means only can detect cluster that are roughly linearly separable.

To solve this problem, first we need to find the problem of it, that is maybe in this dimension space we can't find this linearly relationship, for example in the two-dimension space we have two color data point we say maybe red or blue, and we can't find a line to separate it. How about high dimension space? Let the origin data space project or mapping to high dimension space, we have new view of those data, that can divide those data point easier, and this is the main idea of kernel k means, the kernel means the high dimension space that we want project to.

Unfortunately, we don't know the project function or mapping relation of it, but with the kernel trick that provide a way to define similarity metric of those data point. So that means although we don't know the mapping function, we can also have the relation (distance) in the high dimension space.

Spectral clustering

In the graph, we also have the clustering problem, in the graph G (V, E), each vertex just means a data point, and each edge in the graph will be the similarity metric of its two endpoints, and we define the action cut, the cut can partition graph into two sets A and B such that weight of edges connecting vertices in A to vertices in B is minimum. As the following formula.

$$cut(A,B) \coloneqq \sum_{i \epsilon A, j \epsilon B} W_{i,j}$$

We also hope that size of A and B are very similar, so we add some restrict on it. Here we have Normalized cut and Ratio cut on it.

Normalized Cut:

$$Ncut(A,B) \coloneqq cut(A,B)\left(\frac{1}{vol(A)} + \frac{1}{vol(B)}\right)\ vol(A) = \sum_{i \epsilon A} d_i$$

Ratio Cut:

$$Rcut(A,B) \coloneqq cut(A,B)\left(\frac{1}{|A|} + \frac{1}{|B|}\right)\ |A|: \#\ of\ nodes\ in\ A$$

We also have following relation:

$$vol(A) - association(A) = cut(A, B)$$

$$vol(A) = f^T Df, association(A) = f^T Wf \ cut(A, B) = f^T(D - W)f$$

And then we define a special matrix: L = D – W that represent the graph Laplacian.

Laplacian matrix satisfies the following properties:

1. For every vector $f \in R^n$ we have $f'Lf = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2$
2. L is symmetric and positive semi-definite
3. The smallest eigenvalue of L is 0, the corresponding eigenvector is the constant one vector 1
4. L has n non-negative, real-valued eigenvalues 0

We want to find the spectrum of L, project the weighted graph onto a line, basic idea is that we do the eigen decomposition on L, we will get its eigenvalue and eigenvector, then sort the eigenvalue, select the k eigenvector that is corresponding the first k small eigenvalue, the k eigenvector compose to the new matrix, this matrix has n*k dimension, n mean the number of data points, k mean the number of cluster, and each row also means the coordination of new space, in others word the we project origin weighted graph to new space. Help to find the cluster on it. We also can use this matrix as similarity metric, as coordination. So, we can compute the Euclidean distance as similarity, say the say story like k-mean I mentioned above.

But the homework problem is clustering on the image, recall the idea of gram matrix, we can let this gram matrix as weighted graph. In others words we can transform the problems from k image to graph.

a. Code with detailed explanation
   - Part 1

For kernel k-means we use the new kernel defined below to computer Gram matrix.

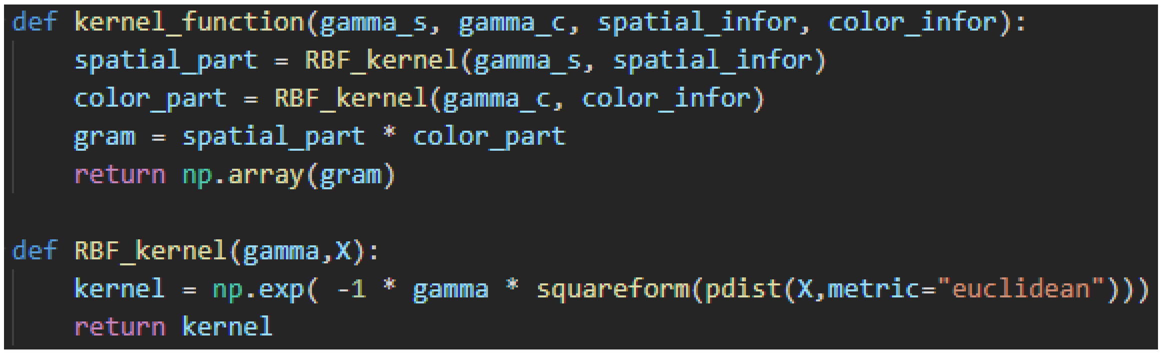
$$k(x, x') = e^{-\gamma_s ||S(x) - S(x')||^2} * e^{-\gamma_c ||C(x) - C(x')||^2}$$

In this formula we can see that it is compose by two RBF kernel funtion, two sample x and x' represented as feature vectors in some input space, In this homework the problem is clustering on the image, so we have two data space, one is spectral info space another is color info space.

In practice, we can easier use the library to compute the Euclidean distance in some space, use the pdist function, the input will be a n*k matrix, n mean the number of data, k means the dimension of this space, for example in the spectral info each row will be the position info of the origin image like this format (x,y) and another example in the color info each row will be the RGB info like (R,G,B)

Gram matrix will be a n*n dimension matrix, which n in the problem means the data point of the image that is the numbers of pixels. The element of this matrix means the distance, or we say similarity of two point, for example Gram[i][j] is the distance between ith pixel and jth pixel.

Following screenshot show how I implement the gram matrix with the RBF kernel.



```
def kernel_function(gamma_s, gamma_c, spatial_infor, color_infor):
    spatial_part = RBF_kernel(gamma_s, spatial_infor)
    color_part = RBF_kernel(gamma_c, color_infor)
    gram = spatial_part * color_part
    return np.array(gram)

def RBF_kernel(gamma,X):
    kernel = np.exp( -1 * gamma * squareform(pdist(X,metric="euclidean")))
    return kernel
```

Like I mentioned as above, although we don’t know the mapping function to the kernel space, with the kernel trick we can also have the similarity metric that is gram matrix to do the basic k-mean clustering. For the basic k-mean we have following formula and relation.

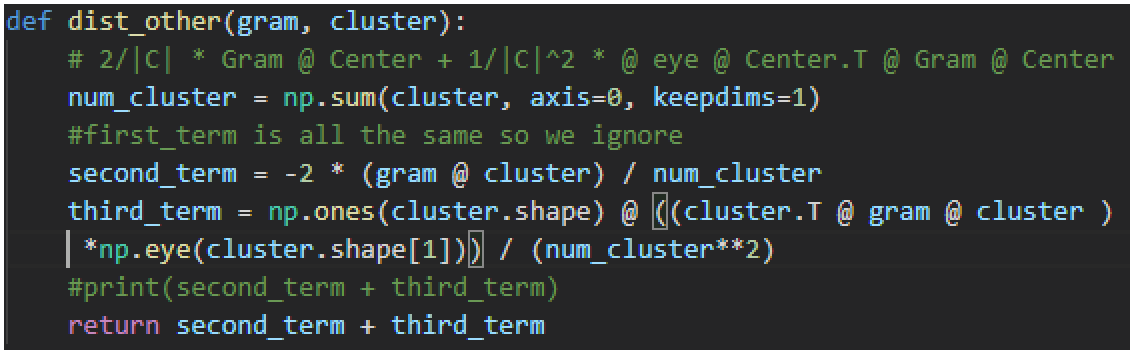
$$\underset{(C_1,\mu_1),\ldots,(C_k,\mu_k)}{argmin} \sum_{i=1}^{k} \sum_{x_j \in C_i} ||x_j - \mu_i||^2$$

$$\mu_k^{\phi} = \frac{1}{|C_k|} \sum_{n=1}^{N} \alpha_{kn} \phi(x_n)$$

$$\left\| \phi(x_J) - \mu_K^{\phi} \right\| = \left\| \phi(x_j) - \frac{1}{|C_k|} \sum_{n=1}^{N} \alpha_{kn} \phi(x_n) \right\|$$

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

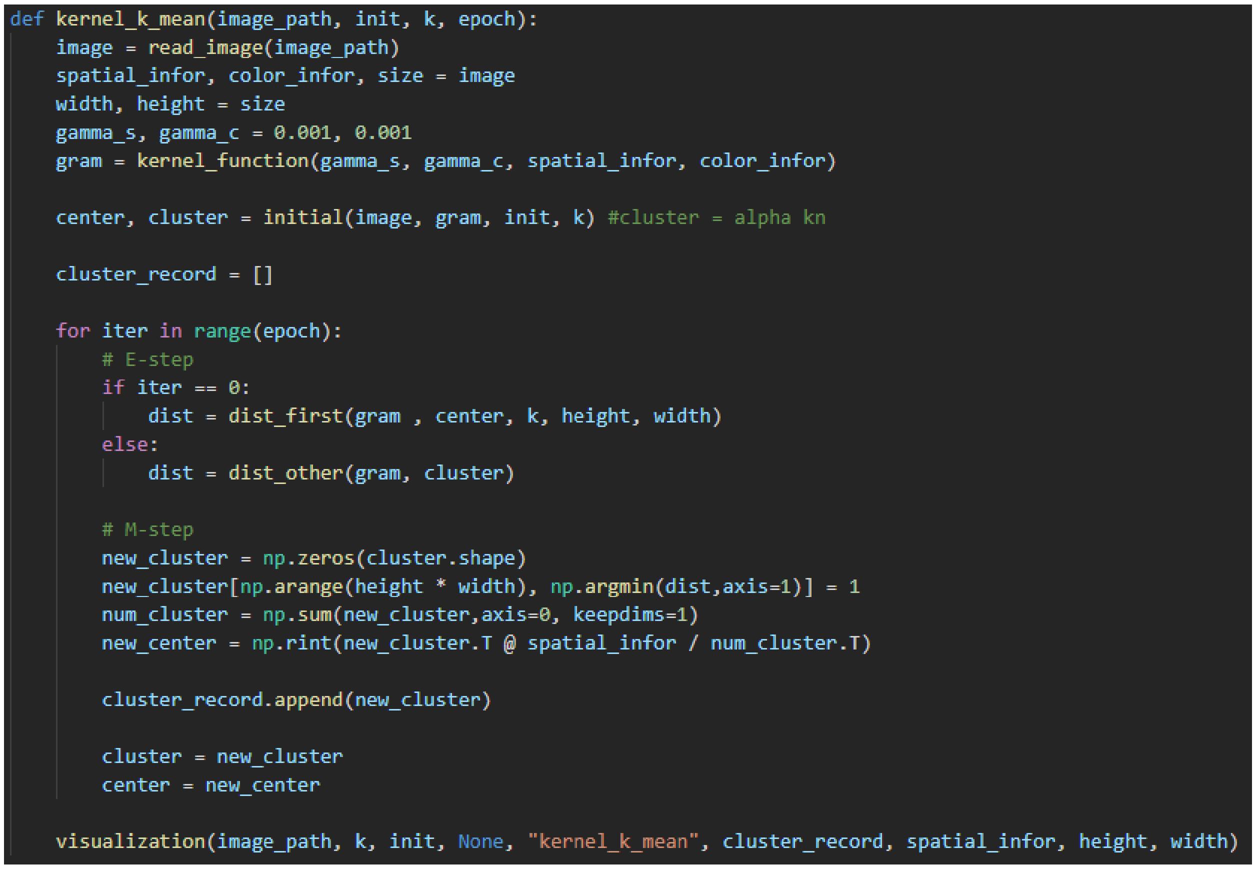
In the kernel k-means, we have above formula, and we can see this formula is compose by three term, first term we always be the constant, so we can ignore it, second and third term we can also compute as matrix form to speed up.



```python
def dist_other(gram, cluster):
    # 2/|C| * Gram @ Center + 1/|C|^2 * @ eye @ Center.T @ Gram @ Center
    num_cluster = np.sum(cluster, axis=0, keepdims=1)
    #first_term is all the same so we ignore
    second_term = -2 * (gram @ cluster) / num_cluster
    third_term = np.ones(cluster.shape) @ ((cluster.T @ gram @ cluster )
    *np.eye(cluster.shape[1])) / (num_cluster**2)
    #print(second_term + third_term)
    return second_term + third_term
```

In the remain part of kernel k-mean, we need to repeat the E step and M step as I mentioned above, this part just like the basic k-mean, let each point be a part of the cluster that have the minimize distance to a center. And in each cluster, we select the new center of this cluster, repeat and repeat until convergence or epoch finish.

Following screenshot is the implementation of the kernel k-mean.



```python
def kernel_k_mean(image_path, init, k, epoch):
    image = read_image(image_path)
    spatial_infor, color_infor, size = image
    width, height = size
    gamma_s, gamma_c = 0.001, 0.001
    gram = kernel_function(gamma_s, gamma_c, spatial_infor, color_infor)

    center, cluster = initial(image, gram, init, k) #cluster = alpha kn

    cluster_record = []

    for iter in range(epoch):
        # E-step
        if iter == 0:
            dist = dist_first(gram , center, k, height, width)
        else:
            dist = dist_other(gram, cluster)

        # M-step
        new_cluster = np.zeros(cluster.shape)
        new_cluster[np.arange(height * width), np.argmin(dist,axis=1)] = 1
        num_cluster = np.sum(new_cluster,axis=0, keepdims=1)
        new_center = np.rint(new_cluster.T @ spatial_infor / num_cluster.T)

        cluster_record.append(new_cluster)

        cluster = new_cluster
        center = new_center

    visualization(image_path, k, init, None, "kernel_k_mean", cluster_record, spatial_infor, height, width)
```

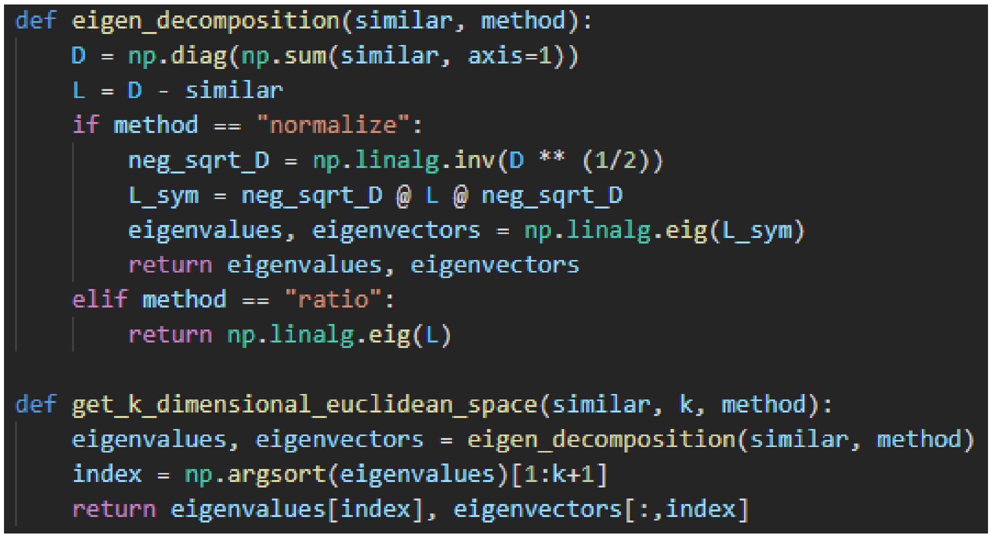
For spectral clustering

In practice, as I mentioned above, we can say the same story as k-mean, kernel k-mean, share the same code and method to find the center and its cluster. The biggest different is that we want to find the similarity metric on the Laplacian that is the spectrum of L, compose by the eigenvector. So, the all the thing we need to add is the function that do the eigen decomposition.

The different Laplacian and relation to Ratio cut and Normalized cut like following.

- Unnormalized Laplacian $L = D - W$ serve in the approximation of the minimization of Ratio cut.
- Normalized Laplacian $D^{\frac{-1}{2}}LD^{\frac{-1}{2}}$ serve in the approximation of the minimization of Normalized cut.

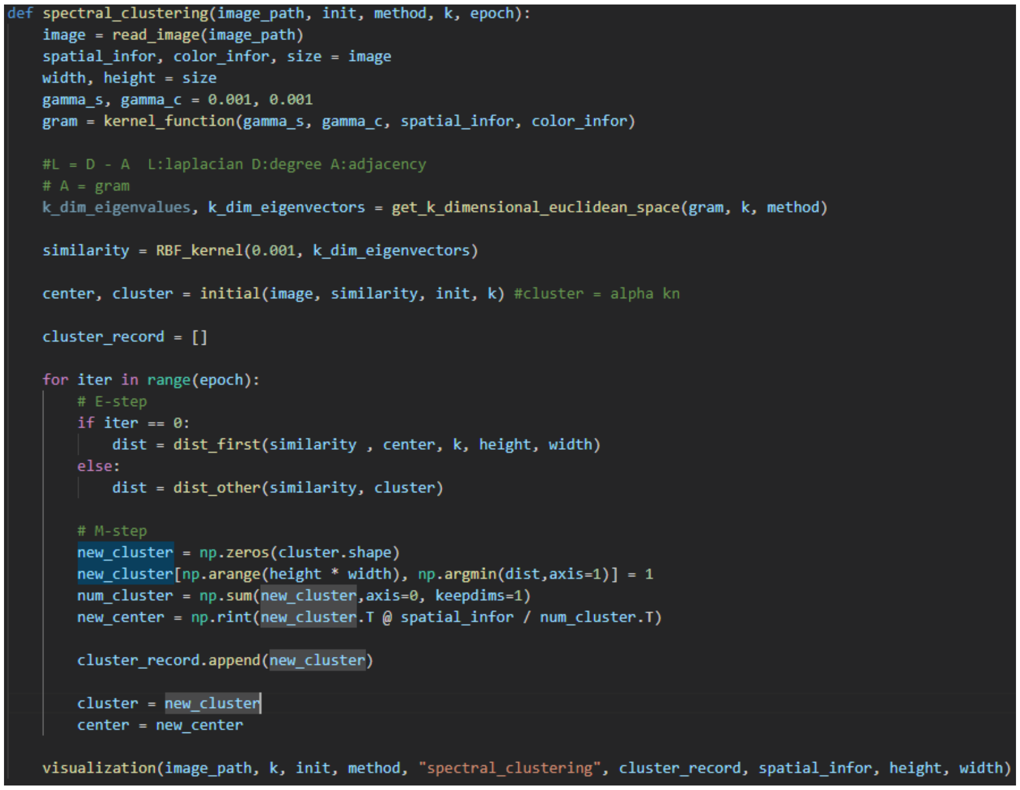
Following screenshot show the implement of eigen decomposition.



```python
def eigen_decomposition(similar, method):
    D = np.diag(np.sum(similar, axis=1))
    L = D - similar
    if method == "normalize":
        neg_sqrt_D = np.linalg.inv(D ** (1/2))
        L_sym = neg_sqrt_D @ L @ neg_sqrt_D
        eigenvalues, eigenvectors = np.linalg.eig(L_sym)
        return eigenvalues, eigenvectors
    elif method == "ratio":
        return np.linalg.eig(L)

def get_k_dimensional_euclidean_space(similar, k, method):
    eigenvalues, eigenvectors = eigen_decomposition(similar, method)
    index = np.argsort(eigenvalues)[1:k+1]
    return eigenvalues[index], eigenvectors[:,index]
```

Other prat is the same story of k-mean and same code.



```python
def spectral_clustering(image_path, init, method, k, epoch):
    image = read_image(image_path)
    spatial_infor, color_infor, size = image
    width, height = size
    gamma_s, gamma_c = 0.001, 0.001
    gram = kernel_function(gamma_s, gamma_c, spatial_infor, color_infor)

    #L = D - A  L:laplacian D:degree A:adjacency
    # A = gram
    k_dim_eigenvalues, k_dim_eigenvectors = get_k_dimensional_euclidean_space(gram, k, method)

    similarity = RBF_kernel(0.001, k_dim_eigenvectors)

    center, cluster = initial(image, similarity, init, k) #cluster = alpha kn

    cluster_record = []

    for iter in range(epoch):
        # E-step
        if iter == 0:
            dist = dist_first(similarity , center, k, height, width)
        else:
            dist = dist_other(similarity, cluster)

        # M-step
        new_cluster = np.zeros(cluster.shape)
        new_cluster[np.arange(height * width), np.argmin(dist,axis=1)] = 1
        num_cluster = np.sum(new_cluster,axis=0, keepdims=1)
        new_center = np.rint(new_cluster.T @ spatial_infor / num_cluster.T)

        cluster_record.append(new_cluster)

        cluster = new_cluster
        center = new_center

    visualization(image_path, k, init, method, "spectral_clustering", cluster_record, spatial_infor, height, width)
```

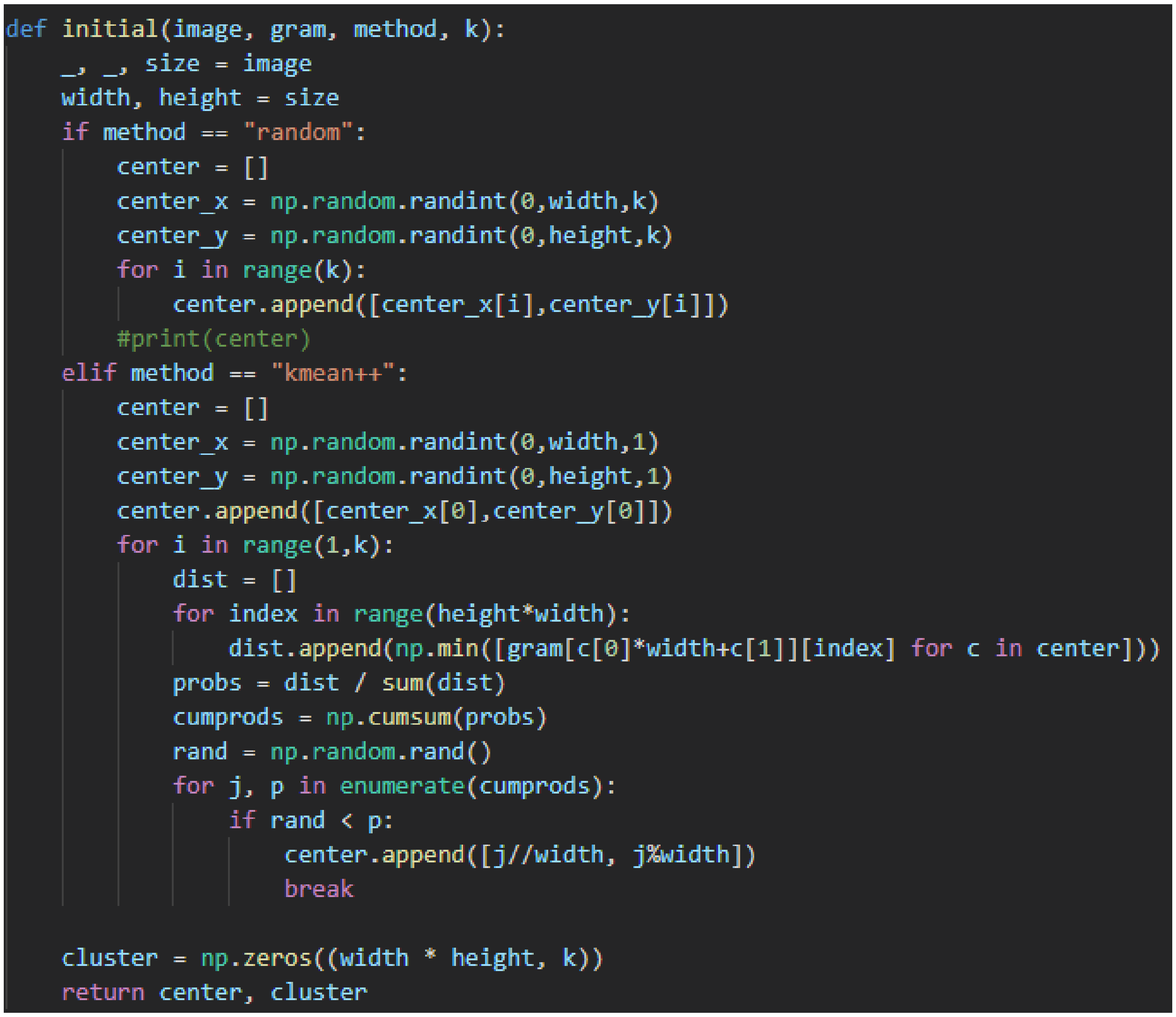
- Part 2

In the above screenshot we can see that k will be the parameter of the function, so we can try difference k of it. The main difference will at the cluster on above screenshot, cluster is the matrix that is n*k just like the indicator vector, it determines whether this data point belong to cluster, if belong element will be one else will be zero.

- Part 3

One of the potential problems of k-mean is that converges to local minimum which depends on the initialization. In other words, we can try to do some smart way to choose the center in the beginning. I want to introduce the kmean++ algorithm, not like the random choose center point, kmean++ hope that we can choose those centers as far as possible, or we say get the longer distance point to the center have higher probability can be chosen.

Following screenshot show how I implement the initial function with random and kmean++.



```
def initial(image, gram, method, k):
    _, _, size = image
    width, height = size
    if method == "random":
        center = []
        center_x = np.random.randint(0,width,k)
        center_y = np.random.randint(0,height,k)
        for i in range(k):
            center.append([center_x[i],center_y[i]])
        #print(center)
    elif method == "kmean++":
        center = []
        center_x = np.random.randint(0,width,1)
        center_y = np.random.randint(0,height,1)
        center.append([center_x[0],center_y[0]])
        for i in range(1,k):
            dist = []
            for index in range(height*width):
                dist.append(np.min([gram[c[0]*width+c[1]][index] for c in center]))
            probs = dist / sum(dist)
            cumprods = np.cumsum(probs)
            rand = np.random.rand()
            for j, p in enumerate(cumprods):
                if rand < p:
                    center.append([j//width, j%width])
                    break

    cluster = np.zeros((width * height, k))
    return center, cluster
```

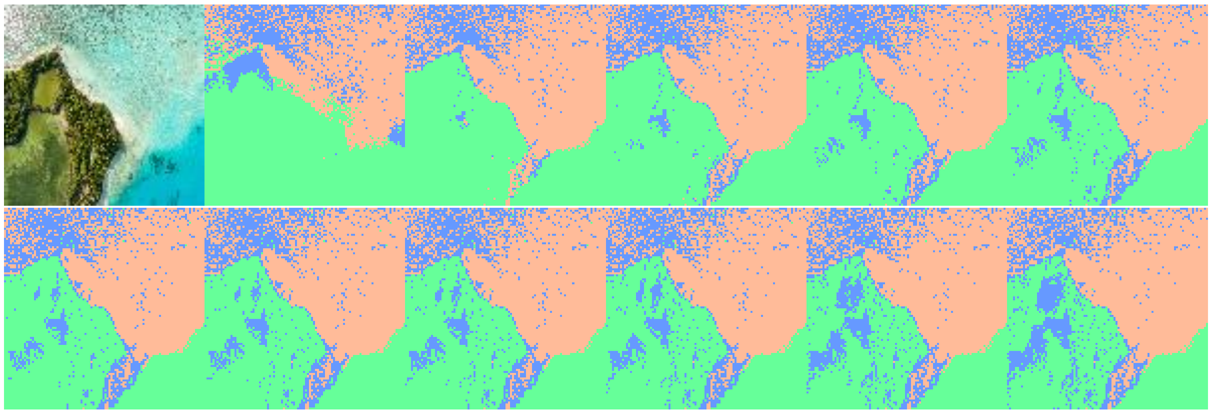
- Part 4

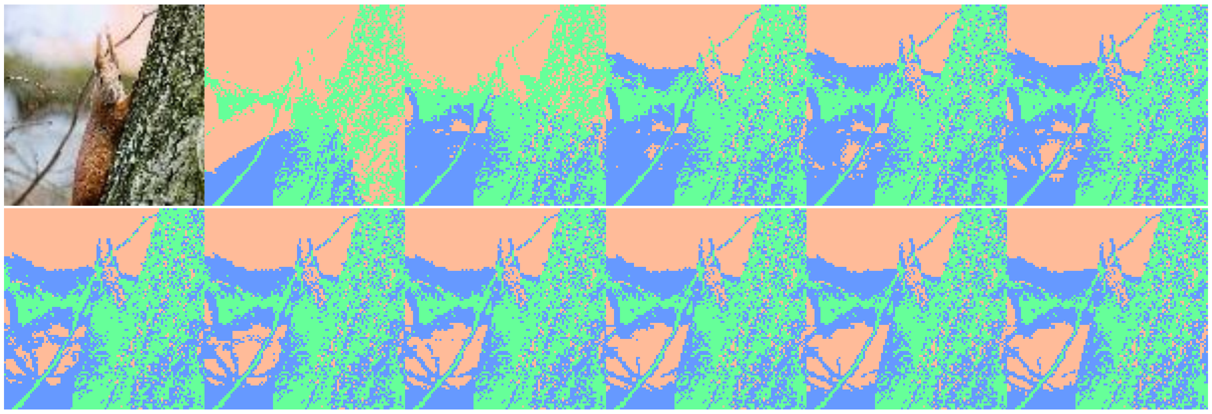
To examine whether the data point in the same cluster have the same coordination of the eigenspace of Laplacian matrix, we can print the eigenvector on it, by the recording of the clustering same as we plot the clustering image. We can see the visualization on it.

b. Experiments settings and results & discussion

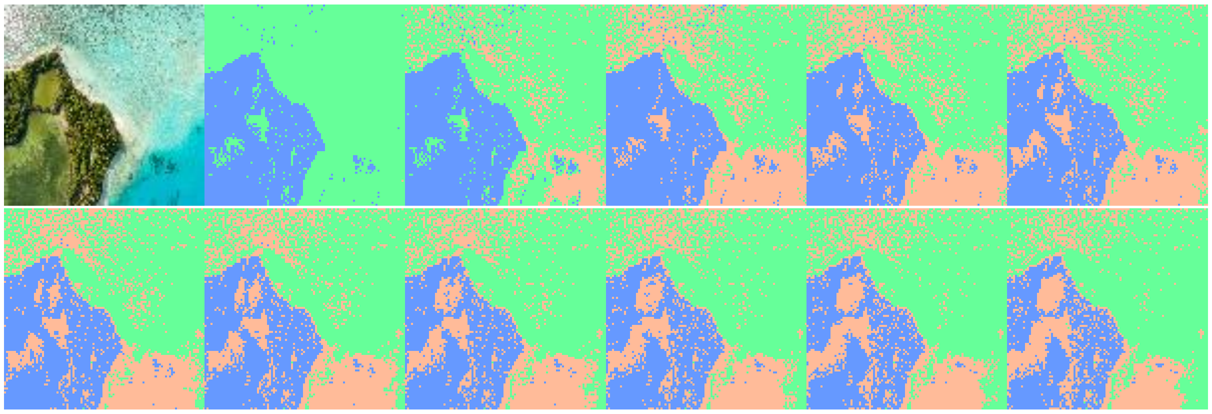
- Part 1

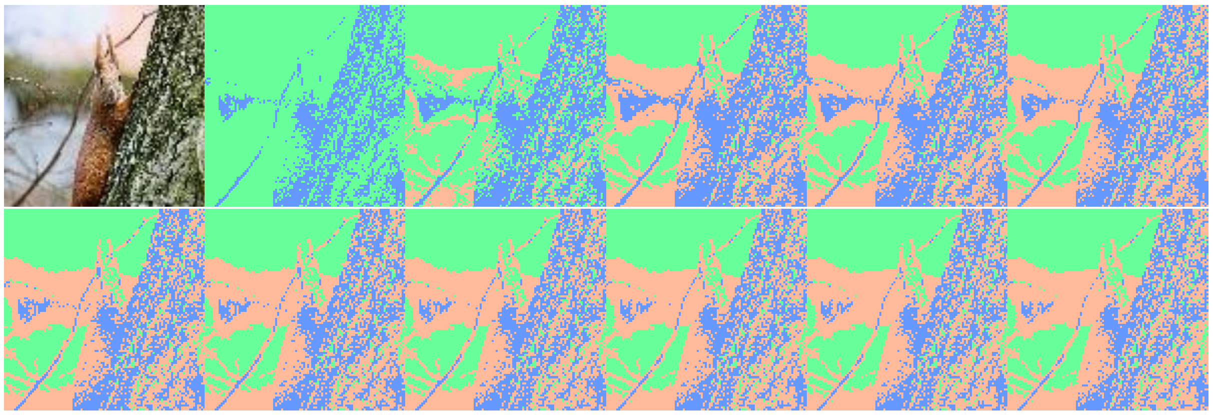
Kernel k mean, k=3, random method



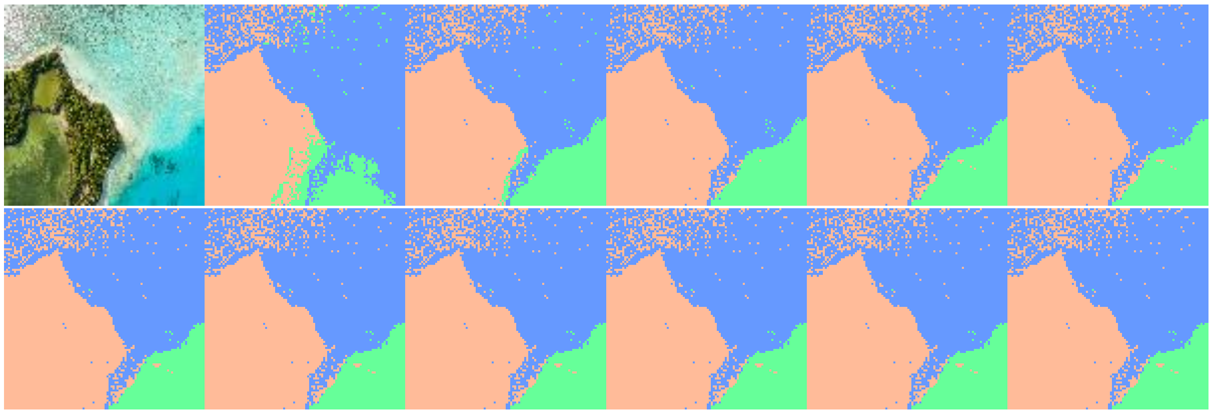


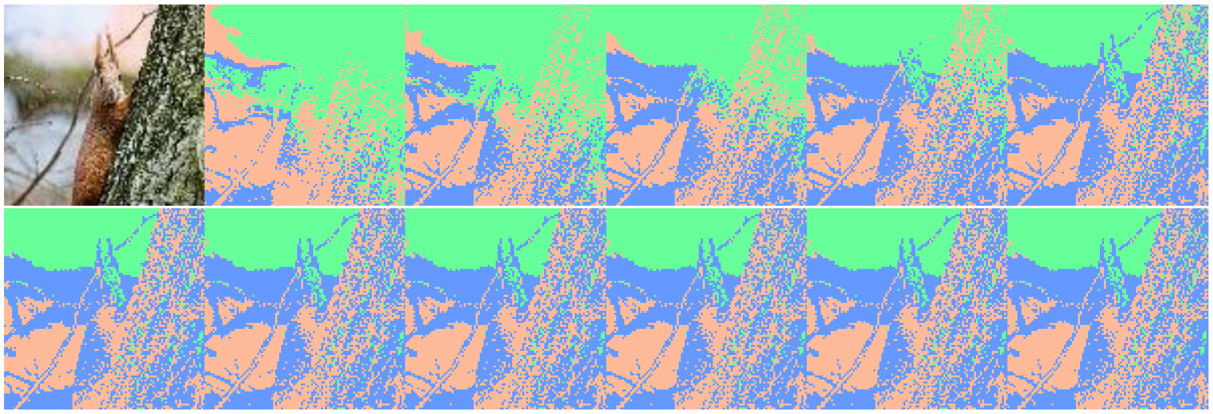
Spectral clustering, k=3, random method, normalize cut





Spectral clustering, k=3, random method, ratio cut

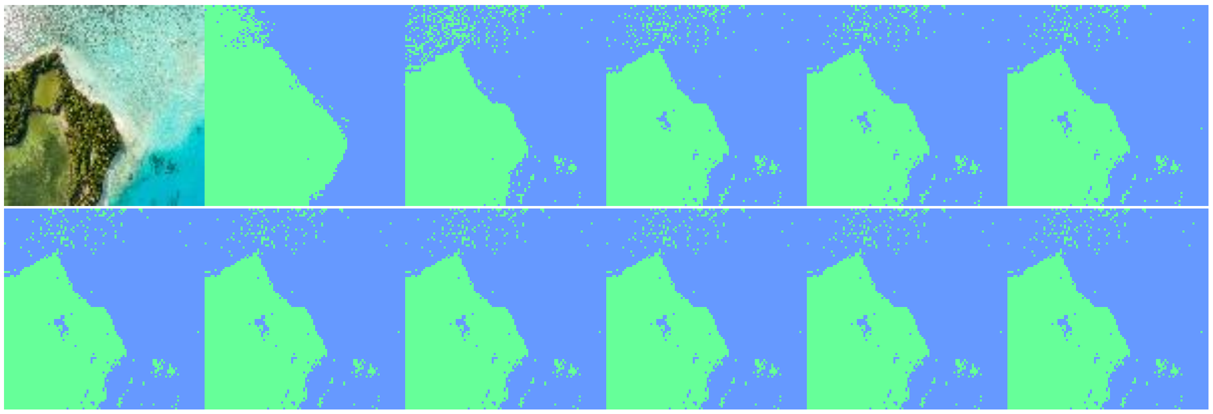




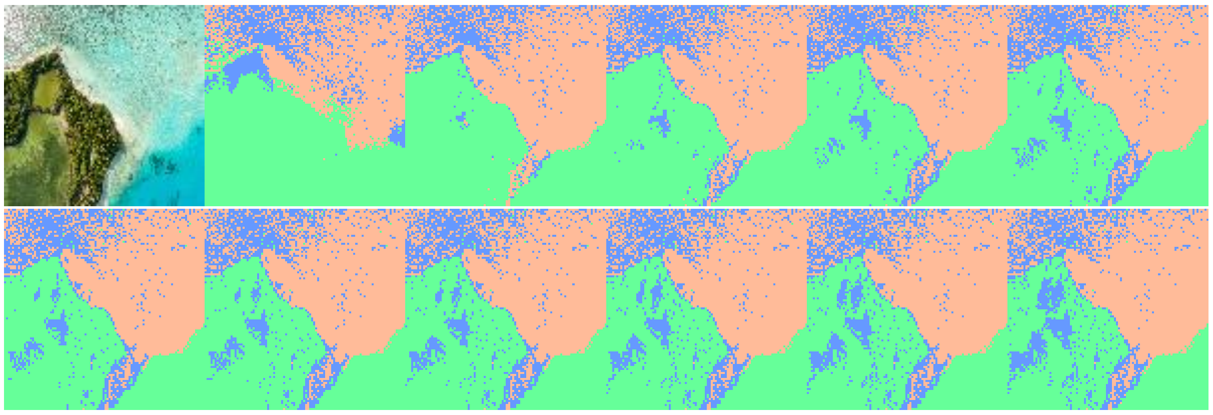
In the above experiment image, I think we all agree that the performance of clustering both on kernel k mean and spectral clustering that image 1 is better than Image 2. I think that image 1 is simpler and image 2 is more complicated. And I didn't see the more different about kernel k means and spectral clustering, also the normalized cut and ratio cut.

- Part 2

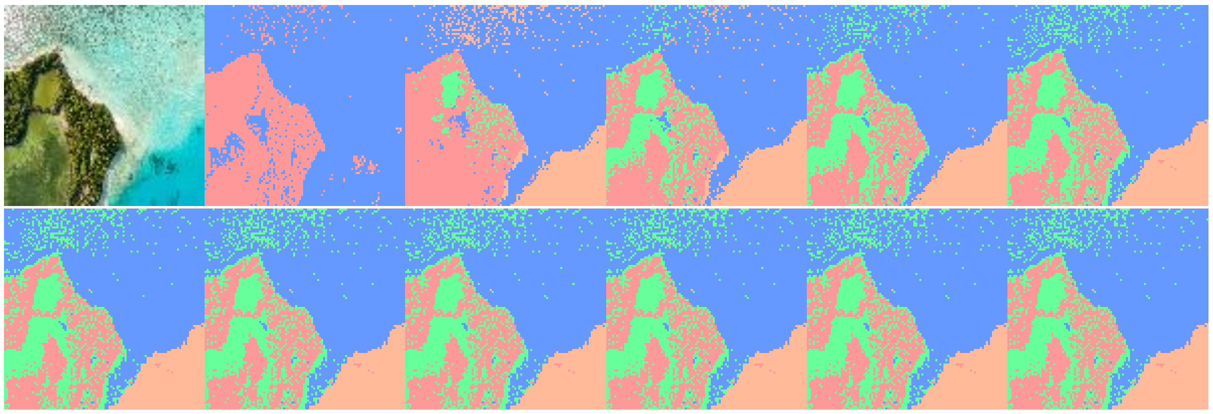
Kernel k mean, k=2, random method



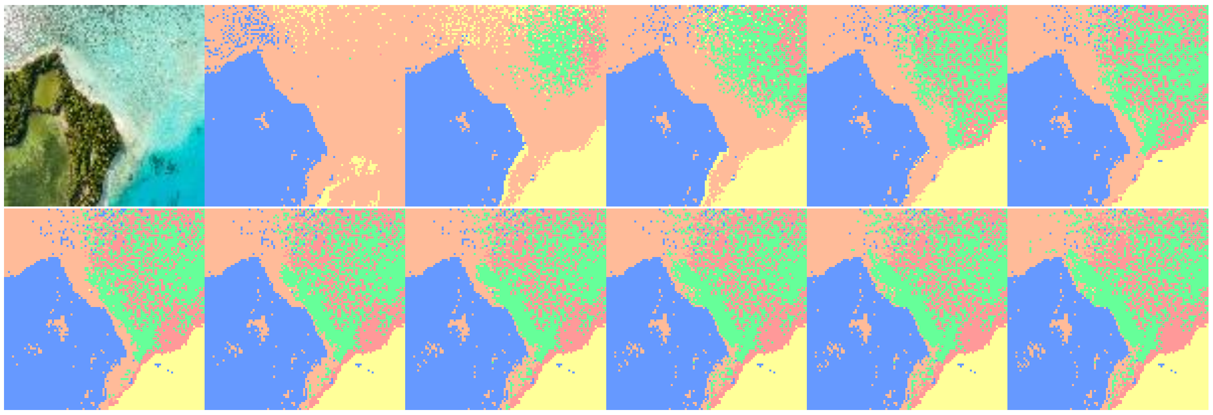
Kernel k mean, k=3, random method



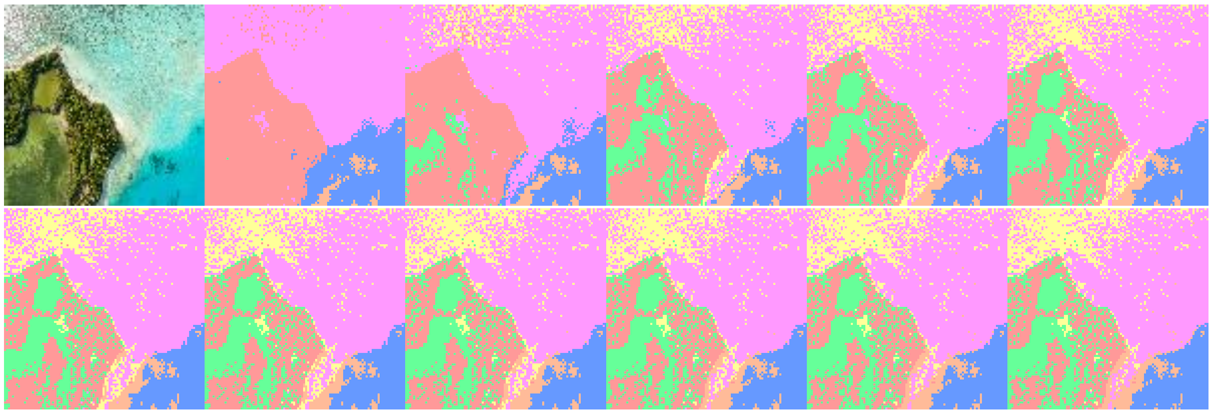
Kernel k mean, k=4, random method



Kernel k mean, k=5, random method



Kernel k mean, k=6, random method

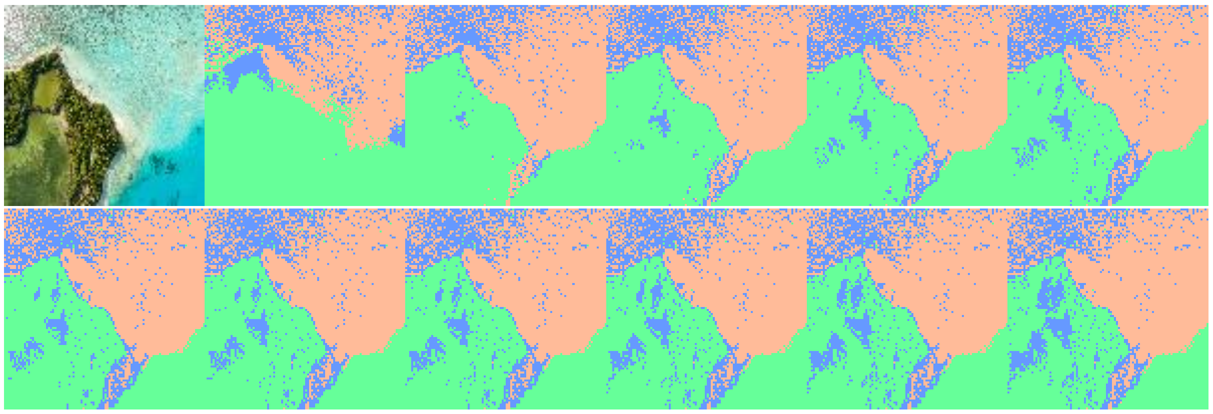


We can we that with the k increasing, more and more categories will be considering, for instance in k=2, we divide into two-part, land and ocean. For k=3 we have land, ocean, some stone on ocean. For k=4 we considering the grassland, tree, shallow sea, and deep sea. And so far.

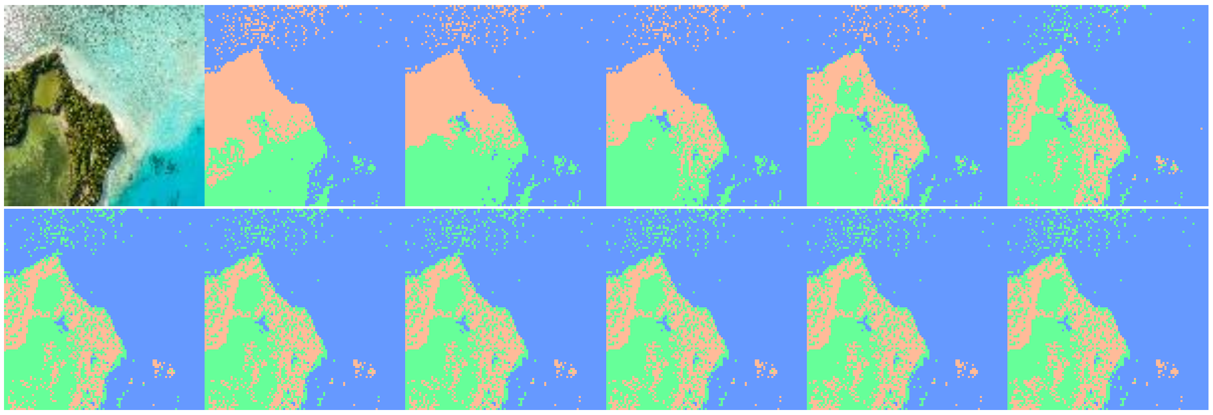
And we all agree that k is not always good as we are increasing, the same problem is still here determining the k will be the problem.

- Part 3

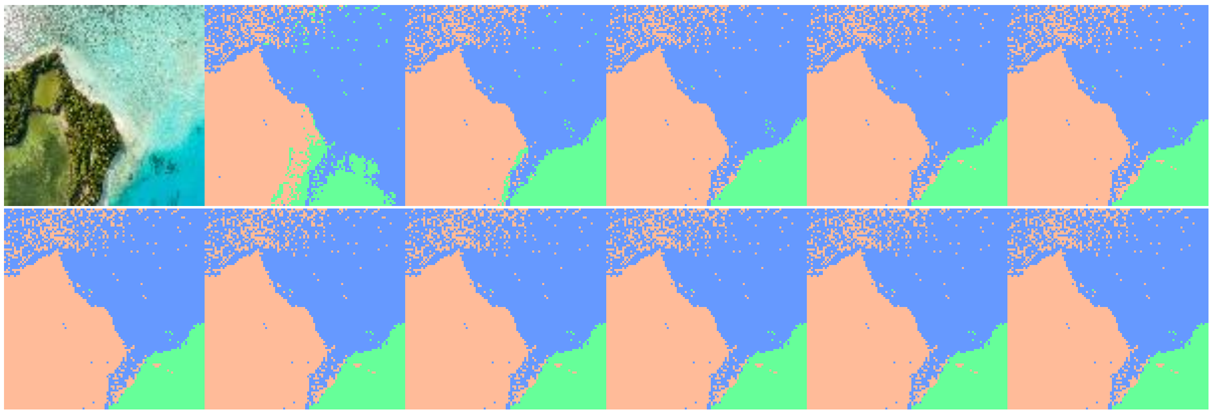
Kernel k mean, k=3, random method



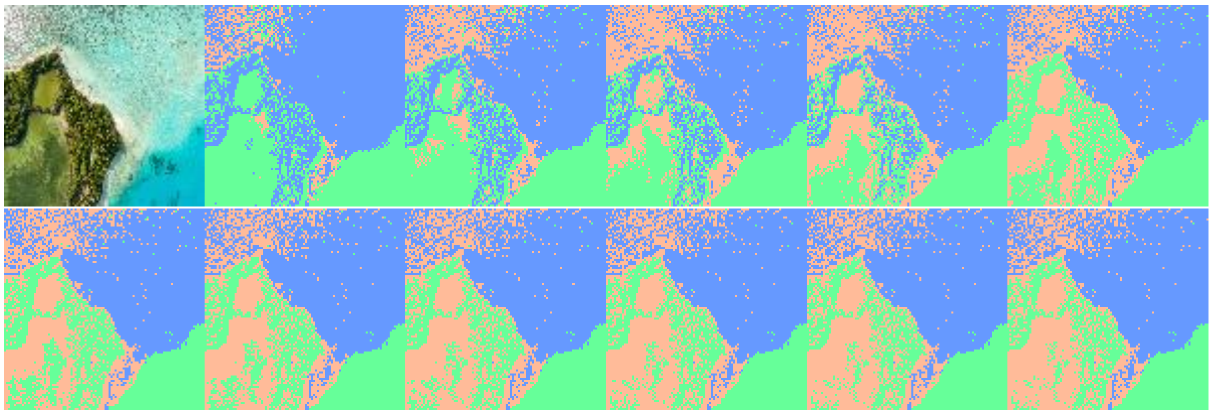
Kernel k mean, k=3, kmean++ method



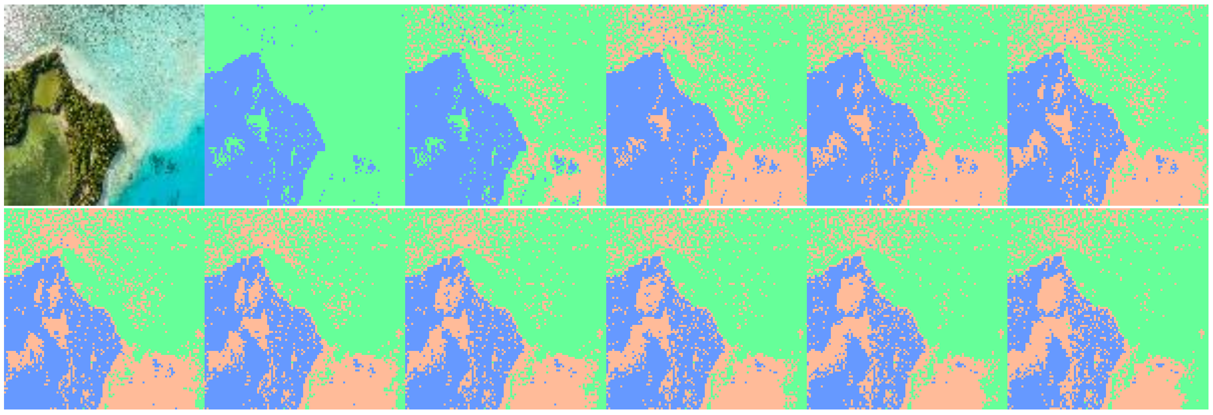
Spectral clustering, k=3, random method, normalize cut



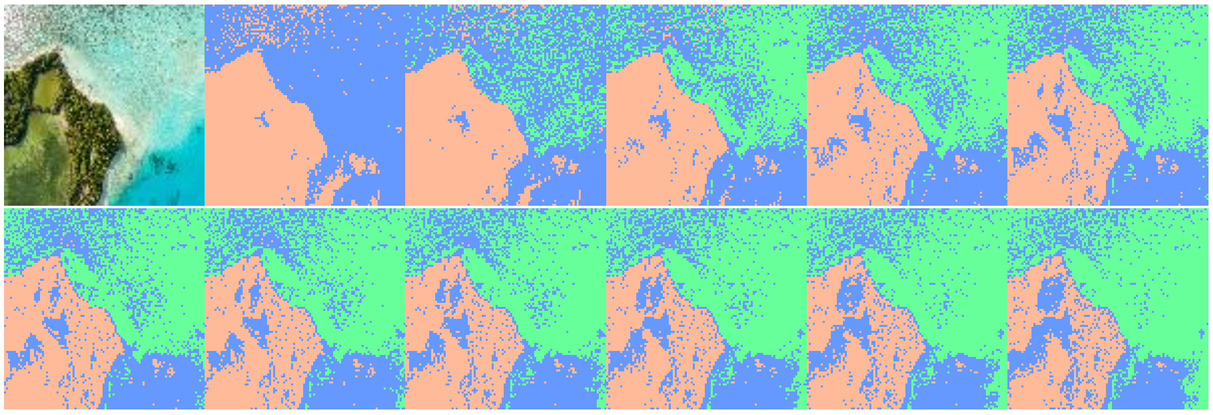
Spectral clustering, k=3, kmean++ method, normalize cut



Spectral clustering, k=3, random method, ratio cut



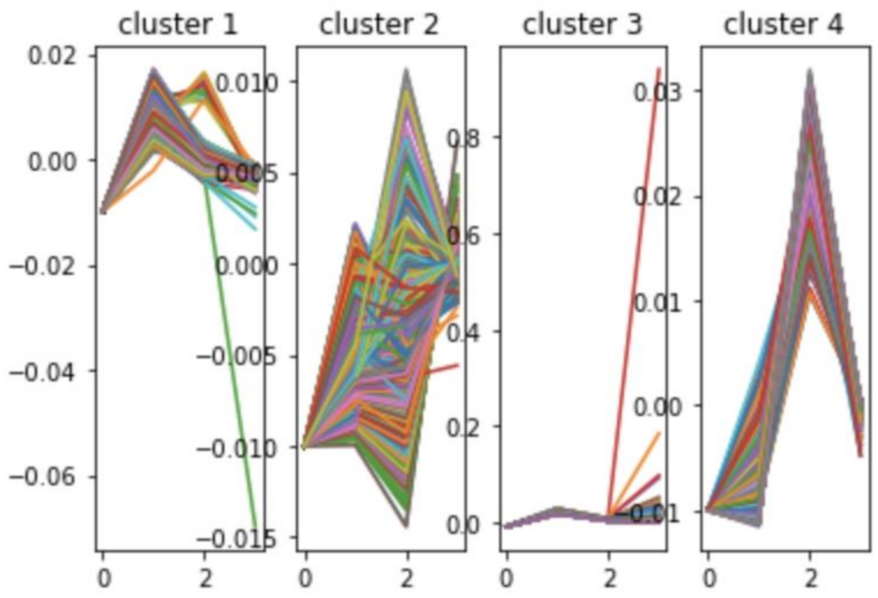
Spectral clustering, k=3, kmean++ method, ratio cut



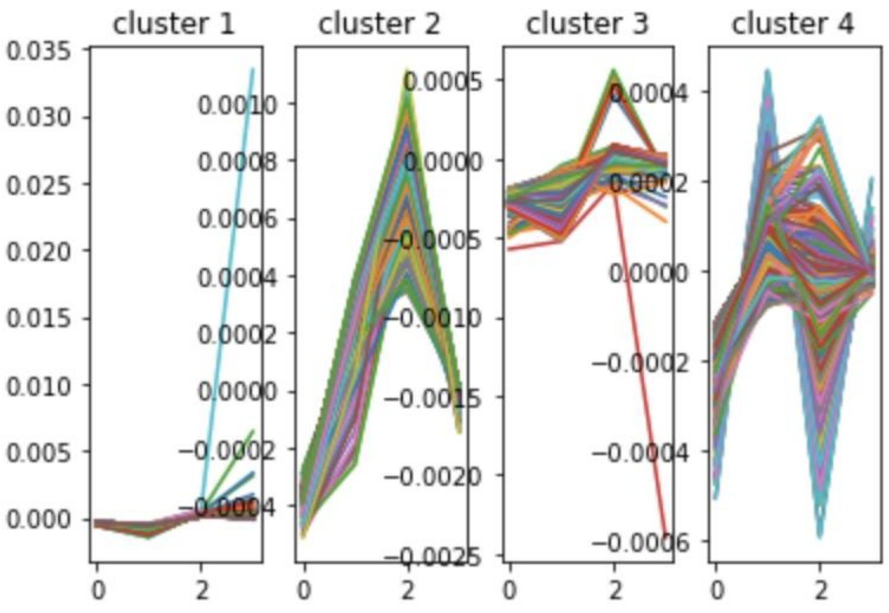
Different between initial method like random method and kmean++, I think kmean++ has better center to start the iterative of E step and M step, but in the experiment image I didn't see the more different between two method.

- Part 4

Spectral clustering, k=4, random method, normalize cut



Spectral clustering, k=3, random method, ratio cut



Like the above image, we can see that some cluster that eigenvector merge like a line, in others that the coordination of the data point we be very similar, but not the same coordination, and others cluster that will be divergent.

c. Observation and discussion

In this part I want to discuss the difference between kernel k mean and spectral clustering, first in spectral clustering eigen decomposition part spent most on the computation time, in my computer it spent about 8 minutes on it, but the performance of spectral clustering I didn’t think it worth to do, instead of doing the kernel k mean.