Machine Learning Homework 6

K-means & spectral clustering

數據所碩一葉詠富 310554031

Github: https://github.com/frankye1000/NYCU-MachineLearning/tree/master/HW6

‣ Part1: You need to make videos or GIF images to show the clustering procedure of your kernel k-means and spectral clustering programs.

‣ Part2: In addition to cluster data into 2 clusters, try more clusters (e.g. 3 or 4) and show your results.

‣ Part3: For the initialization of k-means clustering used in kernel k-means, (e.g. k-means++) and spectral clustering (both normalized cut and ratio cut), try different ways and show corresponding results.

Columns introduction:

Image name: image name.

Initial mean: two type initial mean (1) random、(2)k-means++.

Type: three type method (1)Kernel kmeans、(2)Normalized、(3)Unnormalized.

K: K clusters.

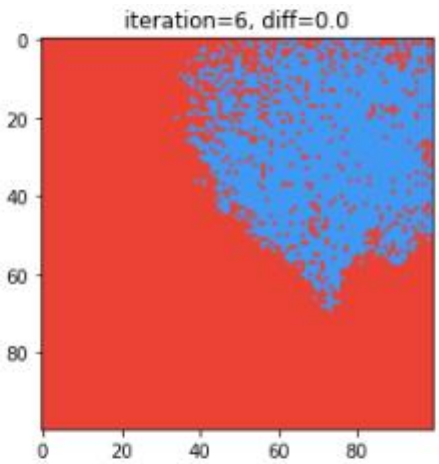
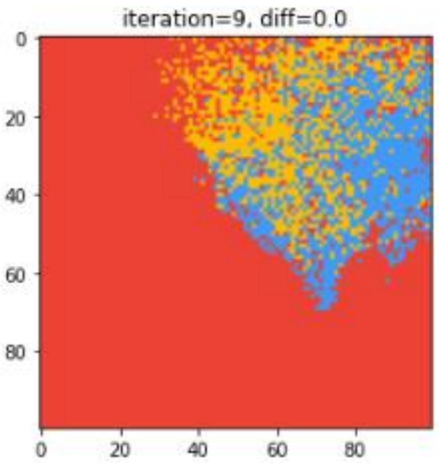
Image name	Initial mean	Type	K(Clusters)	result	GIF link
Image1	random	Kernel kmeans	2		HW6\GIF\image1_random_kernel_kmeans_2Clusters.gif
Image1	random	Kernel kmeans	3		HW6\GIF\image1_random_kernel_kmeans_3Clusters.gif

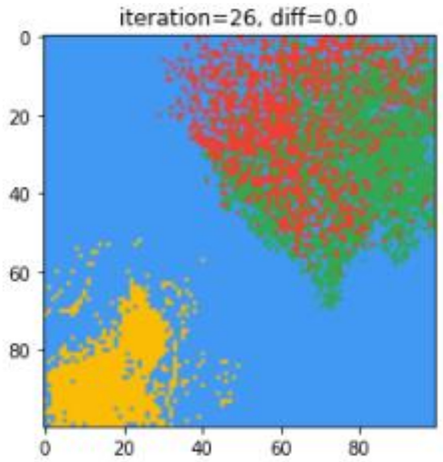
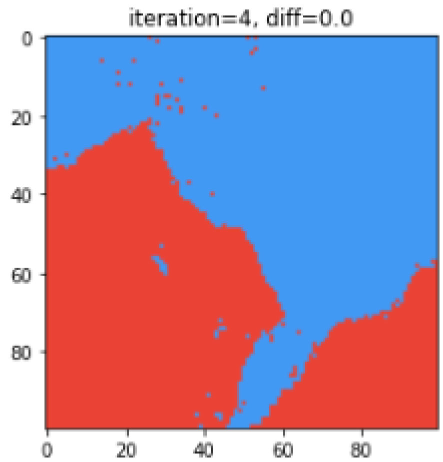
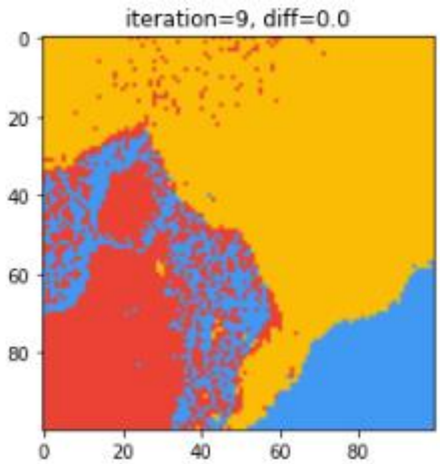
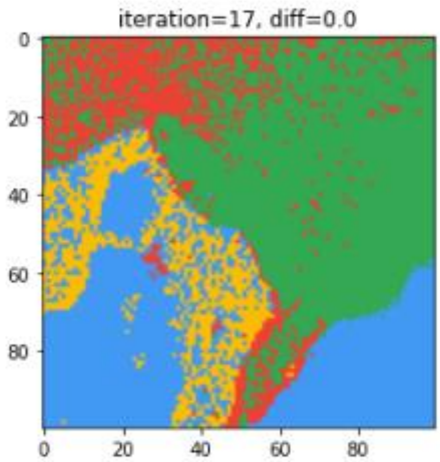
Image1	random	Kernel kmeans	4		HW6\GIF\ image1_random_kernel_kmeans_4Clusters.gif
Image1	random	Normalized	2		HW6\GIF\ image1_random_Normalized_2Clusters.gif
Image1	random	Normalized	3		HW6\GIF\ image1_random_Normalized_3Clusters.gif
Image1	random	Normalized	4		HW6\GIF\ image1_random_Normalized_4Clusters.gif

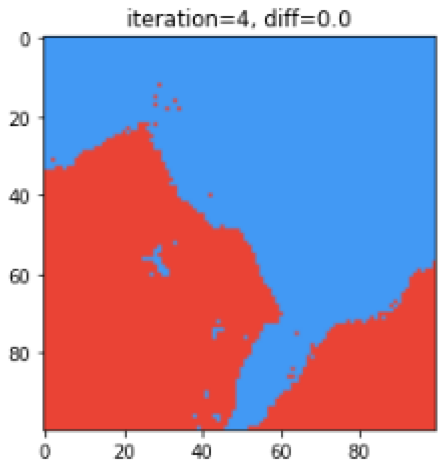
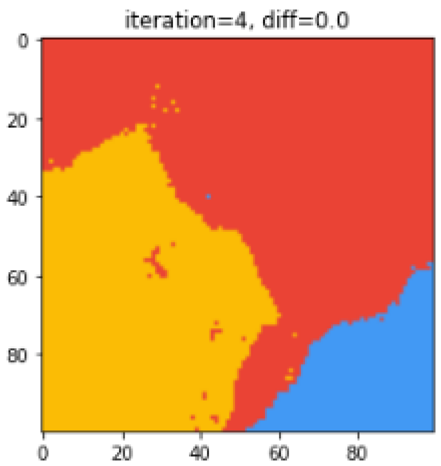
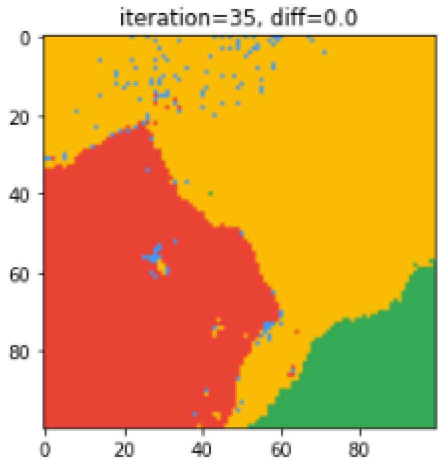
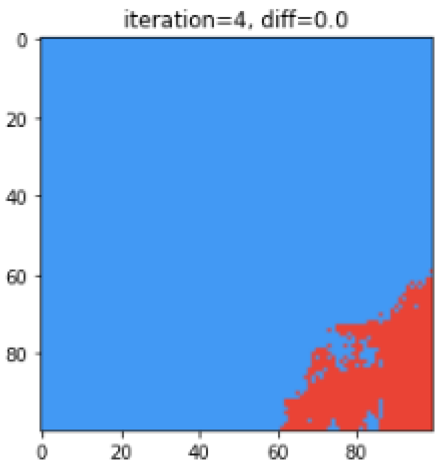
Image1	random	Unnormalized	2		HW6\GIF\image1_random_Unnormalized_2Clusters.gif
Image1	random	Unnormalized	3		HW6\GIF\image1_random_Unnormalized_3Clusters.gif
Image1	random	Unnormalized	4		HW6\GIF\image1_random_Unnormalized_4Clusters.gif
Image1	k-means++	Kernel kmeans	2		HW6\GIF\image1_kmeans_pp_Kernelkmeans_2Clusters.gif

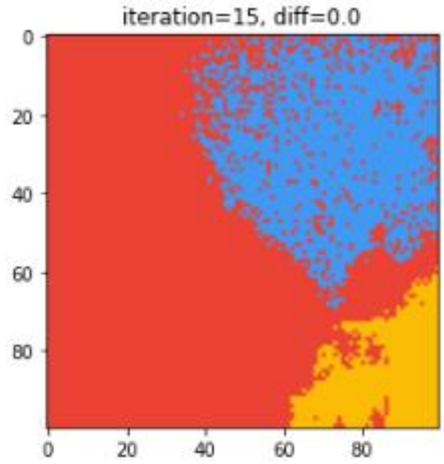
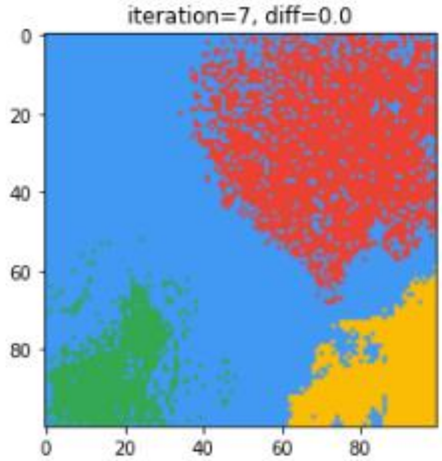
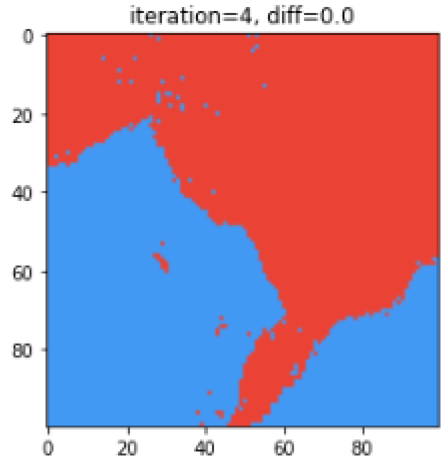
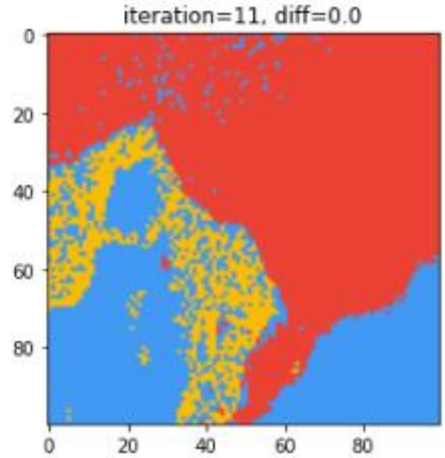
Image1	k-means++	Kernel kmeans	3		HW6\GIF\image1_kmeans_pp_Kernelkmeans_3Clusters.gif
Image1	k-means++	Kernel kmeans	4		HW6\GIF\image1_kmeans_pp_Kernelkmeans_4Clusters.gif
Image1	k-means++	Normalized	2		HW6\GIF\image1_kmeans_pp_Normalized_2Clusters.gif
Image1	k-means++	Normalized	3		HW6\GIF\image1_kmeans_pp_Normalized_3Clusters.gif

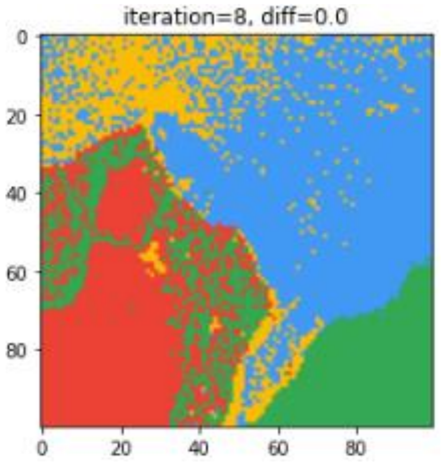
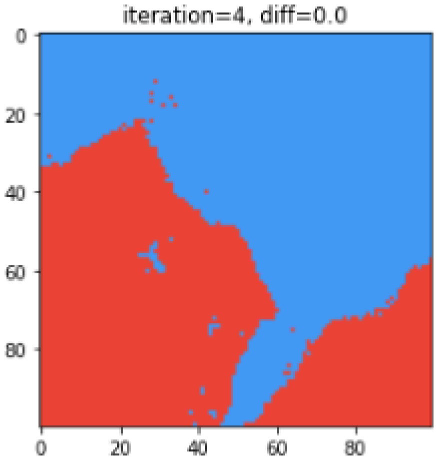
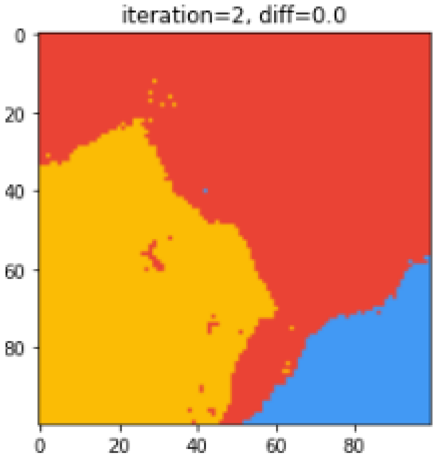
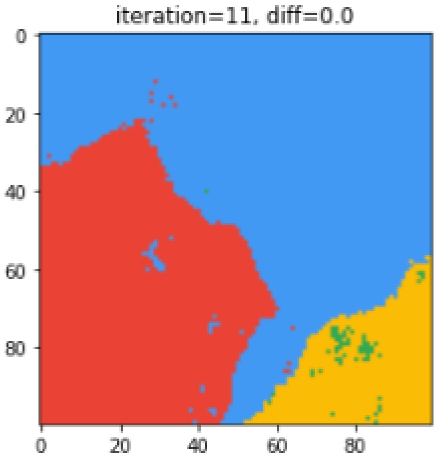
Image1	k-means++	Normalized	4		HW6\GIF\image1_kmeans_pp_Normalized_4Clusters.gif
Image1	k-means++	Unnormalized	2		HW6\GIF\image1_kmeans_pp_Unnormalized_2Clusters.gif
Image1	k-means++	Unnormalized	3		HW6\GIF\image1_kmeans_pp_Unnormalized_3Clusters.gif
Image1	k-means++	Unnormalized	4		HW6\GIF\image1_kmeans_pp_Unnormalized_4Clusters.gif

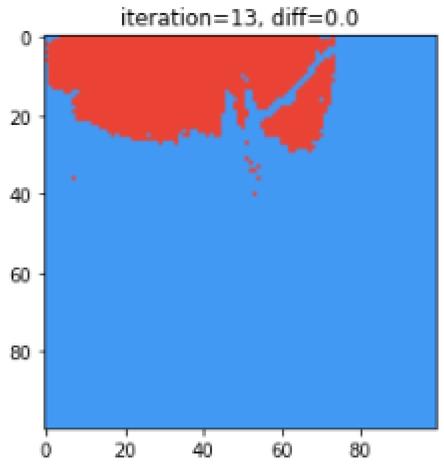
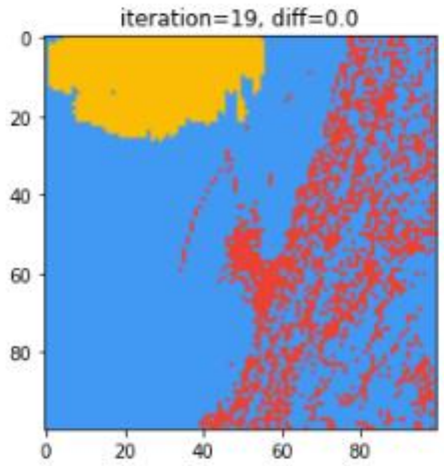
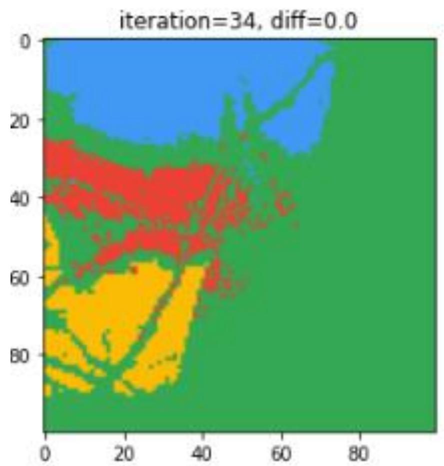
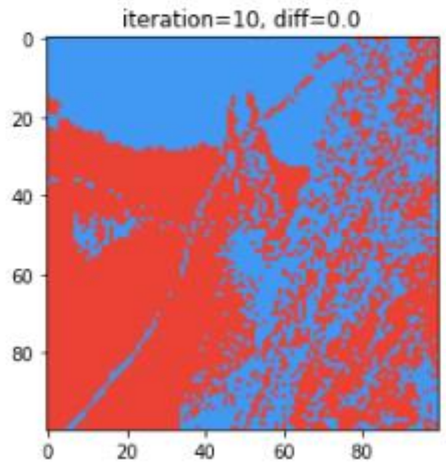
Image2	random	Kernel kmeans	2		HW6\GIF\image2_random_Kernelkmeans_2Clusters.gif
Image2	random	Kernel kmeans	3		HW6\GIF\image2_random_Kernelkmeans_3Clusters.gif
Image2	random	Kernel kmeans	4		HW6\GIF\image2_random_Kernelkmeans_4Clusters.gif
Image2	random	Normalized	2		HW6\GIF\image2_random_Normalized_2Clusters.gif

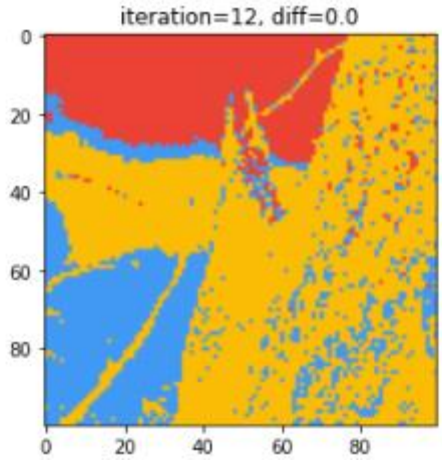
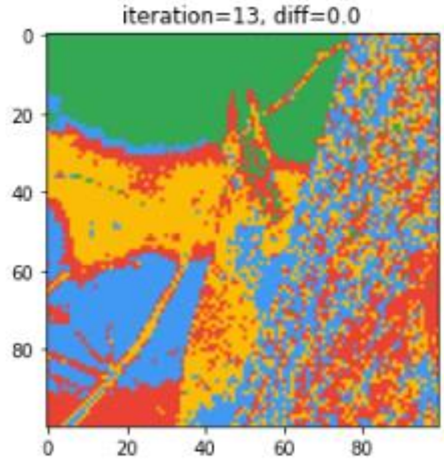
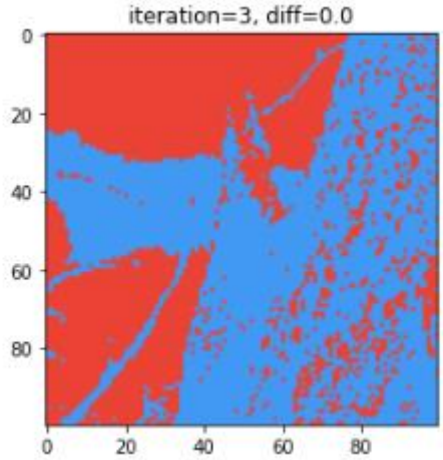
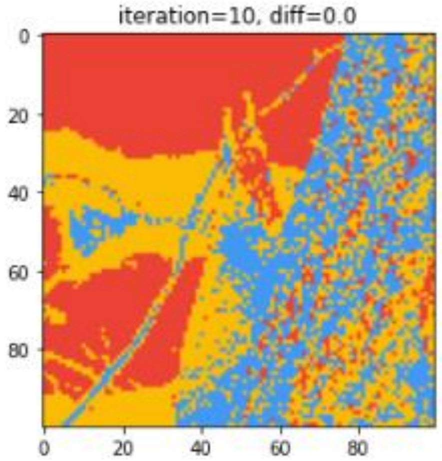
Image2	random	Normalized	3		HW6\GIF\image2_random_Normalized_3Clusters.gif
Image2	random	Normalized	4		HW6\GIF\image2_random_Normalized_4Clusters.gif
Image2	random	Unnormalized	2		HW6\GIF\image2_random_Unnormalized_2Clusters.gif
Image2	random	Unnormalized	3		HW6\GIF\image2_random_Unnormalized_3Clusters.gif

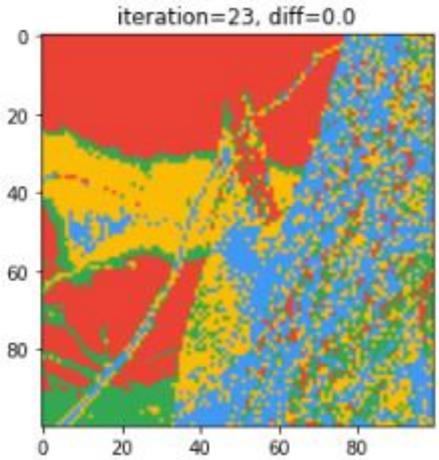
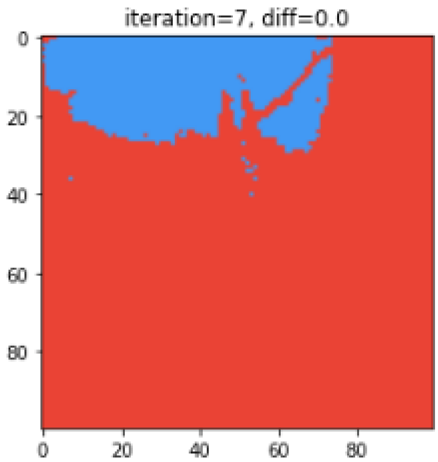
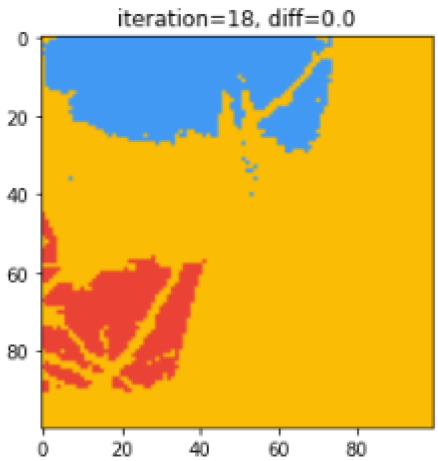
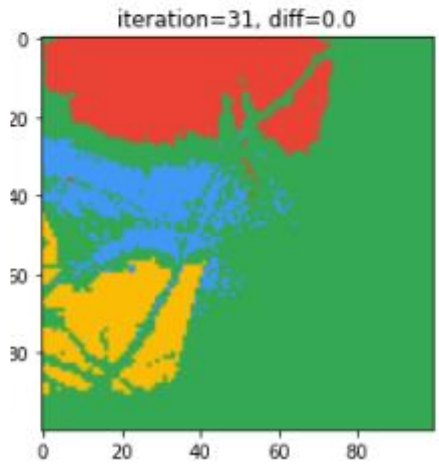
Image2	random	Unnormalized	4		HW6\GIF\image2_random_Unnormalized_4Clusters.gif
Image2	k-means++	Kernel kmeans	2		HW6\GIF\image2_kmeans_pp_Kernelkmeans_2Clusters.gif
Image2	k-means++	Kernel kmeans	3		HW6\GIF\image2_kmeans_pp_Kernelkmeans_3Clusters.gif
Image2	k-means++	Kernel kmeans	4		HW6\GIF\image2_kmeans_pp_Kernelkmeans_4Clusters.gif

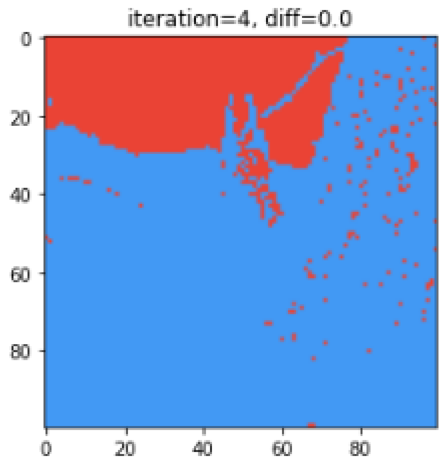
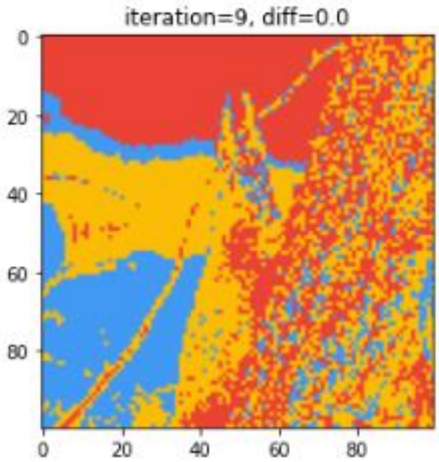
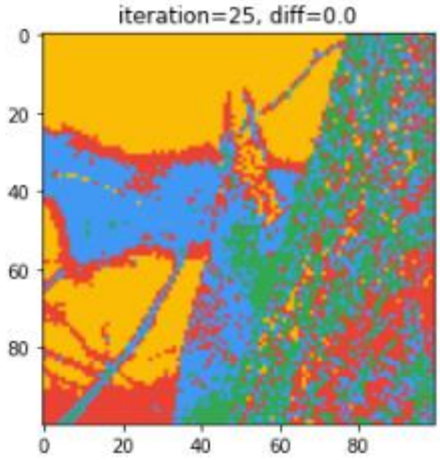
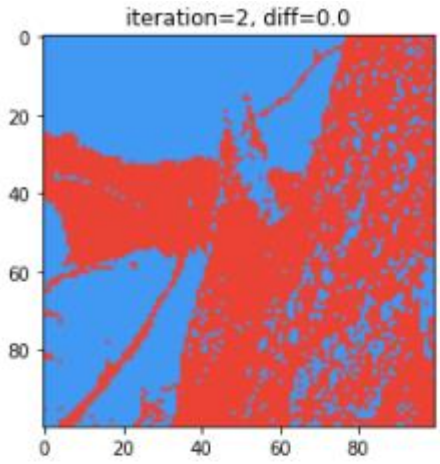
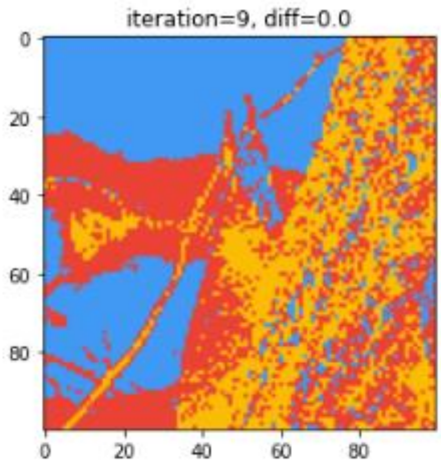
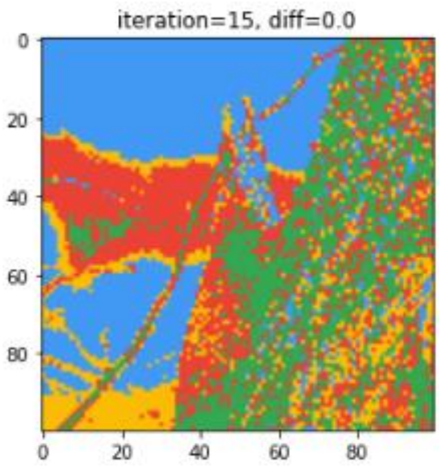
Image2	k-means++	Normalized	2		HW6\GIF\image2_kmeans_pp_Normalized_2Clusters.gif
Image2	k-means++	Normalized	3		HW6\GIF\image2_kmeans_pp_Normalized_3Clusters.gif
Image2	k-means++	Normalized	4		HW6\GIF\image2_kmeans_pp_Normalized_4Clusters.gif
Image2	k-means++	Unnormalized	2		HW6\GIF\image2_kmeans_pp_Unnormalized_2Clusters.gif

Image2	k-means++	Unnormalized	3		HW6\GIF\image2_kmeans_pp_Unnormalized_3Clusters.gif
Image2	k-means++	Unnormalized	4		HW6\GIF\image2_kmeans_pp_Unnormalized_4Clusters.gif

‣ Part4: For spectral clustering (both normalized cut and ratio cut), you can try to examine whether the data points within the same cluster do have the same coordinates in the eigenspace of graph Laplacian or not. You should plot the result and discuss it in the report.

Image: images1

Initial mean: k-means++

K(Clusters): 3

Discuss:

(1) Unnormalized clustering results are more rough, and it has a better result on land segmtation.

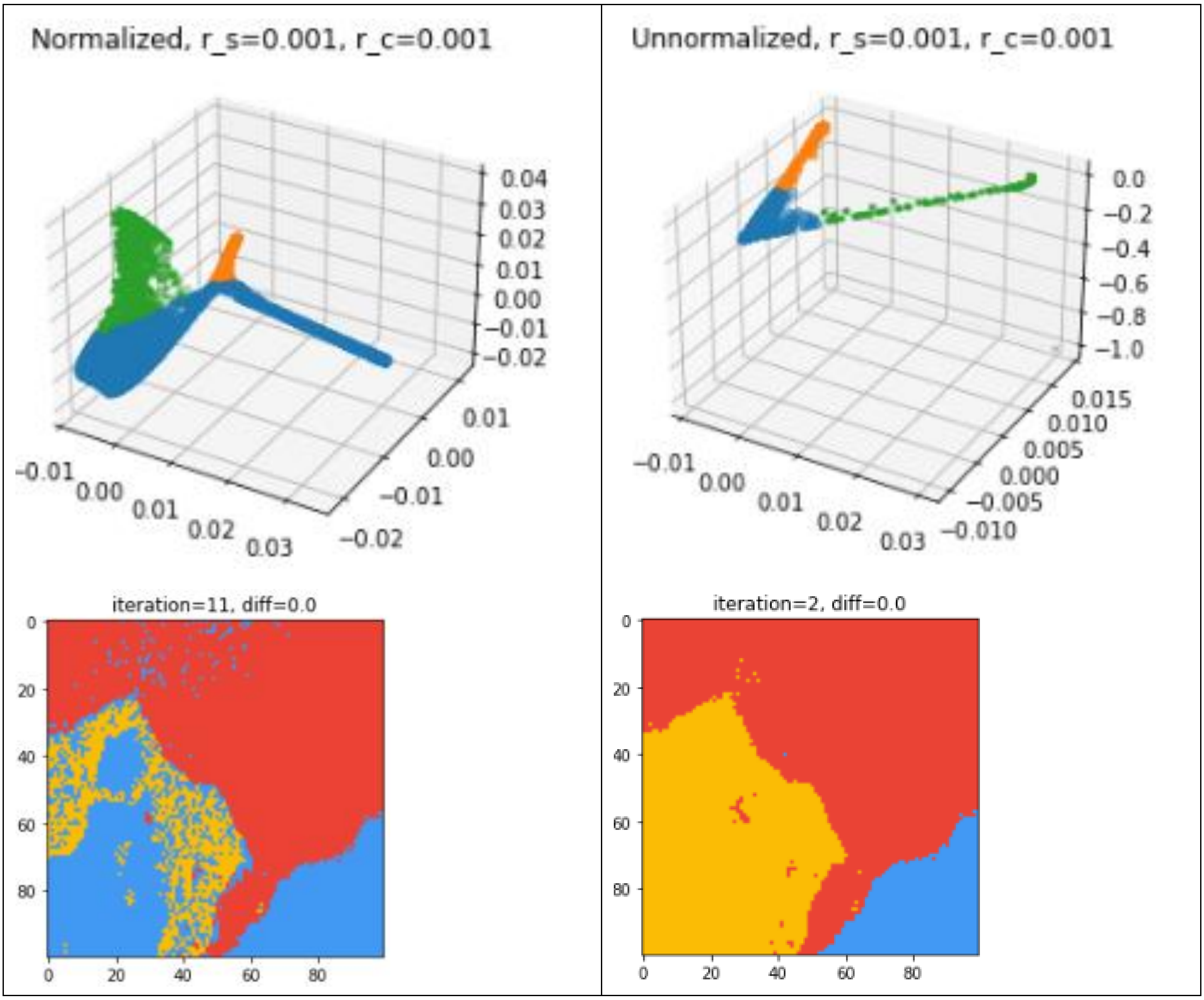


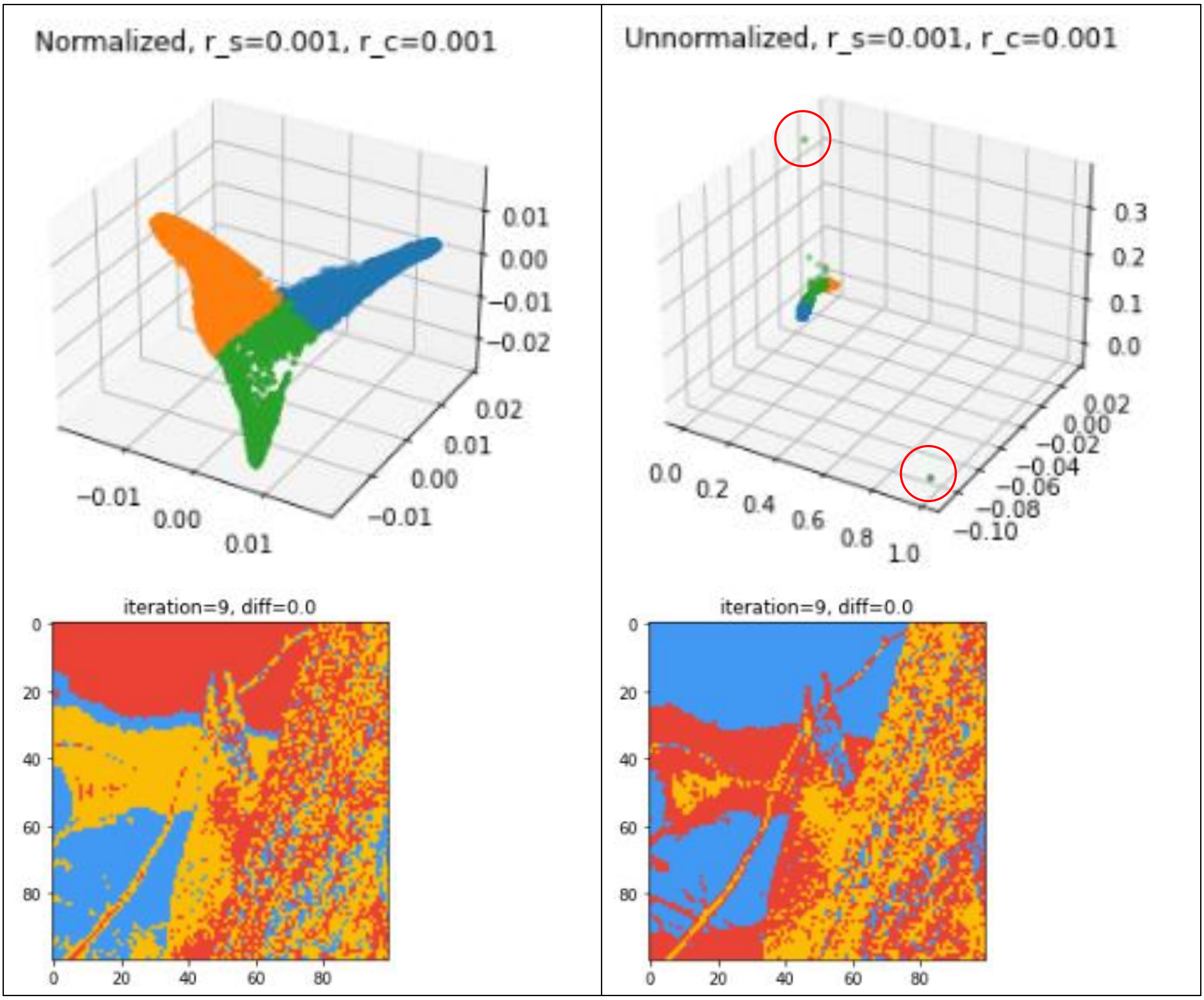
Image: images2

Initial mean: k-means++

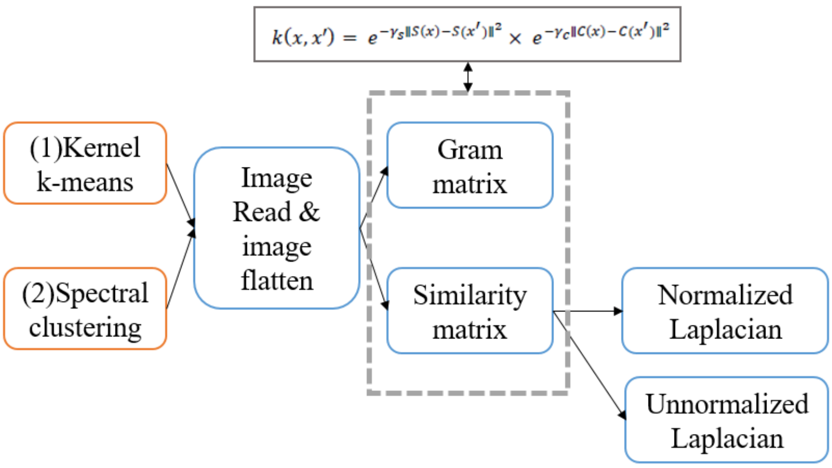
K(Clusters): 3

Discuss:

(1) Unnormalized clustering have same outliers. Outliers will affect the clustering results.



Process:



Code:

(1) Kernel k-means

- Use EM algorithm to do k-means.

  - Determination of r_{nk}
    - since J is a linear function of r_{nk}, we can optimize for each n separately

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \left\| x_n - \mu_k \right\|^2$$

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_k \left\| x_n - \mu_k \right\| \\ 0 & \text{otherwise} \end{cases}$$

E-step

  - Determination of μ_k

$$\frac{\partial J}{\partial \mu_k} = 0 \Rightarrow 2 \sum_{n=1}^{N} r_{nk}(x_n - \mu_k) = 0 \Rightarrow \mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}}$$

M-step

- Define two type initial mean(random, k-means++).

  Random: random pick initial mean.

  K-means++: When selecting a new cluster center, the **farther** the point is from the existing cluster center, the greater the probability of being selected as the cluster center.

```
def initial_mean(X, K, initType):
    Cluster = np.zeros((K, X.shape[1]))

    if initType == 'kmeans_pp':
        # pick 1 cluster_mean
        Cluster[0] = X[np.random.randint(low=0, high=X.shape[0], size=1),:]
        # pick k-1 cluster_mean
        for c in range(1, K):
            Dist = np.zeros((len(X), c))
            for i in range(len(X)):
                for j in range(c):
                    Dist[i,j] = np.sqrt(np.sum((X[i] - Cluster[j])**2))
            Dist_min = np.min(Dist, axis=1)
            sum_ = np.sum(Dist_min) * np.random.rand()
            for i in range(len(X)):
                sum_ -= Dist_min[i]
                if sum_ <= 0:
                    Cluster[c] = X[i]
                    break

    else: # initType == 'random'
        random_pick = np.random.randint(low=0, high=X.shape[0], size=K)
        Cluster = X[random_pick,:]

    return Cluster
```

```python
def kmeans(X, K, H, W, initType='random', gifPath='default.gif'):
    Mean = initial_mean(X, K, initType)

    # Classes of each Xi
    C = np.zeros(len(X), dtype=np.uint8)
    segments = []

    EPS = 1e-9
    diff = 1e9
    count = 1
    while diff > EPS :
        # E-step
        for n in range(len(X)):
            dist = []
            for k in range(K):
                dist.append(np.sqrt(np.sum((X[n] - Mean[k]) ** 2)))
            C[n] = np.argmin(dist)

        # M-step
        New_Mean = np.zeros(Mean.shape)
        for k in range(K):
            class_N = np.argwhere(C == k).reshape(-1)
            for n in class_N:
                New_Mean[k] = New_Mean[k] + X[n]
            if len(class_N) > 0:
                New_Mean[k] = New_Mean[k] / len(class_N)    # standardization

        diff = np.sum((New_Mean - Mean) ** 2)
        Mean = New_Mean

        # group Coloring
        segment = clusterColoring(C, K, H, W)
        segments.append(segment)

        # plot
        plt.title('iteration={}, diff={}'.format(count, diff))
        plt.imshow(segment)
        plt.show()

        count += 1

    return segments
```

- Use gram matrix to calculate kernel k-means.

```python
def gramMatrix(X, r_s=1, r_c=1):
    n = len(X)
    # S(x) spatial information: coordinate
    S = np.array([[i, j] for i in range(100) for j in range(100)])
    # pdist: Repeatedly compared to find the distance (i -> i+1 distance, i -> i+2 distance, i -> i+3 distance...)
    # squareform: If a condensed distance matrix is passed, a redundant one is returned
    K = squareform(np.exp(-r_s * pdist(S, 'sqeuclidean'))) * squareform(np.exp(-r_c * pdist(X, 'sqeuclidean')))

    return K
```

- Main function to set kernel k-means parameter.

main

```
# set parameters
img_path         = 'image1.png'
k_means_initType = 'kmeans_pp'
type_            = 'Kernelkmeans'
K                = 4    # k clusters

# read image
image_flat, Height, Width = imread(img_path)
r_s = 0.001
r_c = 0.001

# kernel
gram_matrix = gramMatrix(image_flat, r_s, r_c)

# k-means
segments    = kmeans(gram_matrix, K, Height, Width, initType=k_means_initType)

# to gif
gif_path = os.path.join('GIF', '{}_{}_{}_{}Clusters.gif'.format(img_path.split('.')[0],
                                                                k_means_initType, type_, K))
arr2gif(segments, gif_path)
```

(2) Spectral clustering

- Use different laplacian to do different cut.

Different Laplacian and Relations to Different Cut

- Unnormalized Laplacian $\boxed{L = D - W}$ serve in the approximation of the minimization of RatioCut
- Normalized Laplacian $\boxed{D^{-1/2} L D^{-1/2}}$ serve in the approximation of the minimization of NormalizedCut.

- Calculate two type cut by similarity matrix.

```
def similarityMatrix(X, r_s=1, r_c=1):
    n = len(X)
    # S(x) spatial information: coordinate
    S = np.array([[i, j] for i in range(100) for j in range(100)])
    # pdist: Repeatedly compared to find the distance (i -> i+1 distance, i -> i+2 distance, i -> i+3 distance...)
    # squareform: If a condensed distance matrix is passed, a redundant one is returned
    K = squareform(np.exp(-r_s * pdist(S, 'sqeuclidean'))) * squareform(np.exp(-r_c * pdist(X, 'sqeuclidean')))

    return K
```

- Use the same k-means & initial mean.

```python
def initial_mean(X, K, initType):
    Cluster = np.zeros((K, X.shape[1]))

    if initType == 'kmeans_pp':
        # pick 1 cluster_mean
        Cluster[0] = X[np.random.randint(low=0, high=X.shape[0], size=1),:]
        # pick k-1 cluster_mean
        for c in range(1, K):
            Dist = np.zeros((len(X), c))
            for i in range(len(X)):
                for j in range(c):
                    Dist[i,j] = np.sqrt(np.sum((X[i] - Cluster[j])**2))
            Dist_min = np.min(Dist, axis=1)
            sum_ = np.sum(Dist_min) * np.random.rand()
            for i in range(len(X)):
                sum_ -= Dist_min[i]
                if sum_ <= 0:
                    Cluster[c] = X[i]
                    break

    else:  #initType == 'random'
        random_pick = np.random.randint(low=0, high=X.shape[0], size=K)
        Cluster = X[random_pick,:]


    return Cluster
```

```python
def kmeans(X, K, H, W, initType='random', gifPath='default.gif'):
    Mean = initial_mean(X, K, initType)

    # Classes of each Xi
    C = np.zeros(len(X), dtype=np.uint8)
    segments = []

    EPS = 1e-9
    diff = 1e9
    count = 1
    while diff > EPS :
        # E-step
        for n in range(len(X)):
            dist = []
            for k in range(K):
                dist.append(np.sqrt(np.sum((X[n] - Mean[k]) ** 2)))
            C[n] = np.argmin(dist)

        # M-step
        New_Mean = np.zeros(Mean.shape)
        for k in range(K):
            class_N = np.argwhere(C == k).reshape(-1)
            for n in class_N:
                New_Mean[k] = New_Mean[k] + X[n]
            if len(class_N) > 0:
                New_Mean[k] = New_Mean[k] / len(class_N)    # standardization

        diff = np.sum((New_Mean - Mean) ** 2)
        Mean = New_Mean

        # cluster Coloring
        segment = clusterColoring(C, K, H, W)
        segments.append(segment)

        # plot
        plt.title('iteration={}, diff={}'.format(count, diff))
        plt.imshow(segment)
        plt.show()

        count += 1

    return C, segments
```

- Main function to set spectral clustering parameter, and calculate Normalized Laplacian and Unnormalized Laplacian.
- Also use np.linalg.eig() to get eigenvalue, eigenvector.

main

```
EPS=1e-9

# set parameters
img_path         = 'image2.png'
k_means_initType = 'kmeans_pp'
type_            = "Normalized"   # Select type
K                = 3               # k clusters

image_flat, Height, Width = imread(img_path)
r_s = 0.001
r_c = 0.001

# similarity matrix
W = similarityMatrix(image_flat, r_s, r_c)

# degree matrix
D = np.diag(np.sum(W, axis=1))

# Unnormalized Laplacian(RatioCut)
L = D - W

# Normalized Laplacian(NormalizedCut)
D_             = D**(-0.5)
D_[D_ == inf] = 0
L_sym          = D_@L@D_
```

Select type

```
if type_ == "Normalized":
#     # save eigenvalue, eigenvector
#     eigenvalue, eigenvector = np.linalg.eig(L_sym)
#     np.save('npy/{}_{}_eigenvalue_{:.3f}_{:.3f}'.format(type_, img_path.split('.')[0], r_s, r_c), eigenvalue)
#     np.save('npy/{}_{}_eigenvector_{:.3f}_{:.3f}'.format(type_, img_path.split('.')[0], r_s, r_c), eigenvector)

    # load eigenvalue, eigenvector
    eigenvalue = np.load('npy/{}_{}_eigenvalue_{:.3f}_{:.3f}.npy'.format(type_, img_path.split('.')[0], r_s, r_c))
    eigenvector= np.load('npy/{}_{}_eigenvector_{:.3f}_{:.3f}.npy'.format(type_, img_path.split('.')[0], r_s, r_c))
    sort_index = np.argsort(eigenvalue)

    # U:(n,k)
    U = eigenvector[:,sort_index[1:1+K]]
    # T:(n,k) each row with norm 1
    sums = np.sqrt(np.sum(np.square(U), axis=1)).reshape(-1,1)
    T = U / sums

    # k-means
    C, segments = kmeans(T, K, Height, Width, initType=k_means_initType)

    # to gif
    gif_path = os.path.join('GIF', '{}_{}_{}_{}Clusters.gif'.format(img_path.split('.')[0], k_means_initType, type_, K))
    arr2gif(segments, gif_path)


if type_ == "Unnormalized":
#     # save eigenvalue, eigenvector
#     eigenvalue, eigenvector = np.linalg.eig(L)
#     np.save('npy/{}_{}_eigenvalue_{:.3f}_{:.3f}'.format(type_, img_path.split('.')[0], r_s, r_c), eigenvalue)
#     np.save('npy/{}_{}_eigenvector_{:.3f}_{:.3f}'.format(type_, img_path.split('.')[0], r_s, r_c), eigenvector)

    # load eigenvalue, eigenvector
    eigenvalue = np.load('npy/{}_{}_eigenvalue_{:.3f}_{:.3f}.npy'.format(type_, img_path.split('.')[0], r_s, r_c))
    eigenvector= np.load('npy/{}_{}_eigenvector_{:.3f}_{:.3f}.npy'.format(type_, img_path.split('.')[0], r_s, r_c))
    sort_index = np.argsort(eigenvalue)

    # U:(n,k)
    U = eigenvector[:,sort_index[1:1+K]]
    # T:(n,k) each row with norm 1
    sums = np.sqrt(np.sum(np.square(U), axis=1)).reshape(-1,1)
    T = U / sums

    # k-means
    C, segments = kmeans(T, K, Height, Width, initType=k_means_initType)

    # gif
    gif_path = os.path.join('GIF', '{}_{}_{}_{}Clusters.gif'.format(img_path.split('.')[0], k_means_initType,type_, K))
    arr2gif(segments, gif_path)
```

Tool function:

(1) read image & image flatten.

```
def imread(path):
    image = cv2.imread(path)
    H, W, C = image.shape
    image_flat = image.reshape(H * W, C)
    return image_flat, H, W
```

(2) define every cluster color.

```
def clusterColoring(X, K, H, W):
    # define group color
    colordict = {0: [66, 153, 244],
                 1: [234, 67, 53],
                 2: [251, 188, 5],
                 3: [52, 168, 83],
                 4: [127,255,212],
                 }
    temp = np.zeros((H,W,3))
    for h in range(H):
        for w in range(W):
            temp[h,w,:] = colordict[X[h*W+w]]

    return temp.astype(np.uint8)
```

(3) save array to gif.

```
def arr2gif(segments, gif_path):
    for i in range(len(segments)):
        segments[i] = segments[i].transpose(1, 0, 2)
    write_gif(segments, gif_path, fps=10)
```