Machine Learning Homework 6
Kernel K-means and Spectral Clustering

309551053 資科工所 黃敏涓

○ a. code with detailed explanations (40%)

● Part1 (15%)

** whole project set gammac = 1/(255*255), gammas = 1/(100*100)

(a). Kernel K-means

```python
def KernelFunction(pixel, coord):
    s = np.exp(-gamma_s * cdist(coord, coord, 'sqeuclidean'))
    c = np.exp(-gamma_c * cdist(pixel, pixel, 'sqeuclidean'))

    return s * c

```

Compute kernel function using RBF kernel on both spatial information and color information and multiply them as complex kernel.

```python

def second_term(kernel, cluster, dataidx, idx):
    C = 0
    kernel_sum = 0
    for i in range(cluster.shape[0]):
        if cluster[i] == idx:
            C += 1
    if C == 0:
        C = 1
    for i in range(kernel.shape[0]):
        if cluster[i] == idx:
            kernel_sum += kernel[dataidx][i]

    return (-2) * kernel_sum / C
```

```python
def clustering(pixel, kernel, cluster):
    new_cluster = np.zeros(pixel.shape[0], dtype = np.int)
    C = np.zeros(K_cluster, dtype=np.int)
    third_term = np.zeros(K_cluster, dtype=np.float)
    for i in range(cluster.shape[0]):
        C[cluster[i]] += 1
    for index in range(K_cluster):
        for p in range(kernel.shape[0]):
            for q in range(kernel.shape[0]):
                if cluster[p] == index and cluster[q] == index:
                    third_term[index] += kernel[p][q]
    for index in range(K_cluster):
        if C[index] == 0:
            C[index] = 1
        third_term[index] /= (C[index] ** 2)
    for index in range(pixel.shape[0]):
        distance = np.zeros(K_cluster, dtype=np.float32) # 2* 1
        for i in range(K_cluster):
            distance[i] = second_term(kernel, cluster, index, i) + third_term[i]
        new_cluster[index] = np.argmin(distance)

    return new_cluster
```

Do the computation of formula in slide p.22

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

```python
def KernelKMeans(imgname, savepath, pixel, coord):
    init_method = ['random', 'modK']
    for method in init_method:
        gif = []
        FOLDER = f'{K_cluster}_cluster_{method}'
        save_method_path = f'{savepath}/{FOLDER}'
        if not os.path.exists(save_method_path):
            os.makedirs(save_method_path)

        cluster = initial(pixel, method)
        kernel = KernelFunction(pixel, coord)
        iteration = 0
        error = -100000
        prev_error = -100000
        while(iteration < EPOCH):
            iteration += 1
            print("iteration = {}".format(iteration))
            prev_cluster = cluster
            img = Visualization(imgname, save_method_path, iteration, cluster, method)
            gif.append(img)
            cluster = clustering(pixel, kernel, cluster)
            error = ComputeError(cluster, prev_cluster)
            print("error = {}".format(error))
            if error == prev_error:
                break
            prev_error = error
```

```python
        error = -100000
        prev_error = -100000
        while(iteration < EPOCH):
            iteration += 1
            print("iteration = {}".format(iteration))
            prev_cluster = cluster
            img = Visualization(imgname, save_method_path, iteration, cluster, method)
            gif.append(img)
            cluster = clustering(pixel, kernel, cluster)
            error = ComputeError(cluster, prev_cluster)
            print("error = {}".format(error))
            if error == prev_error:
                break
            prev_error = error
        gif[0].save(save_method_path+'/'+imgname.strip('.png')+'.gif',
                    save_all = True, append_images = gif)
```

Summarize:

step1: Get parameters using sys.argv

step2: Calculate kernel using spatial information and color information

step3: Use different initialized method to cluster each pixel

step4: Calculate distance between pixel and center

step5: Update cluster

(b). Spectral Clustering

```python
def KernelFunction(pixel, coord):
    # compute spatial information RBF kernel
    s = np.exp(-gamma_s * cdist(coord, coord, 'sqeuclidean'))
    # compute color information RBF kernel
    c = np.exp(-gamma_c * cdist(pixel, pixel, 'sqeuclidean'))

    return s * c
```

Compute kernel regarding spatial information and color information

```python
# update clustering results using new centroids
def clustering(E, centroids):
    cluster = np.zeros(scale * scale, dtype = int)
    for i in range(scale * scale):
        distance = np.zeros(K_cluster, dtype = np.float32)
        for j in range(K_cluster):
            distance[j] = np.sum(np.absolute(E[i] - centroids[j]))
        cluster[i] = np.argmin(distance)
    return cluster
```

Do clustering of each pixel

```python
# update centroids using
def UpdateCentroids(E, centroids, cluster):
    centroids = np.zeros(centroids.shape, dtype=np.float64)
    tmp = np.zeros(K_cluster, dtype=np.int32)
    for i in range(scale * scale):
        centroids[cluster[i]] += E[i]
        tmp[cluster[i]] += 1
    for i in range(K_cluster):
        if tmp[i] == 0:
            tmp[i] = 1
        centroids[i] /= tmp[i]
    return centroids
```

Update the new centroids according to the clustering results

```python
def KMeans(imgname, E, cut, init_method):
    centroids, cluster = initial(E, init_method)
    Visualization(imgname, cluster, iteration, cut)
    prev_error = -100000
    error = -100000
    gif = []
    iteration = 0
    while iteration < EPOCH:
        iteration += 1
        prev_cluster = cluster
        cluster = clustering(E, centroids)
        centroids = UpdateCentroids(E, centroids, cluster)
        img = Visualization(imgname, cluster, iteration, cut)
        gif.append(img)
        error = ComputeError(cluster, prev_cluster)
        print(f'Iter: {iteration}: {error}')
        if error == prev_error:
            break
        prev_error = error
    if K_cluster == 2:
        EigenSpace(E, cluster, cut)
```

I run 15 epoch to do the iteration, if error == prev_error than break the loop.

Summarize:

step1: Get parameters using sys.argv

step2: Calculate kernel using spatial information and color information

step3: Calculate Laplacian L

step4: Calculate the first k eigenvectors of L

step5: Use eigenvectors to do K-means

● Part2 (5%)

(a) Kernel K-means

Running command: python file.py imgname k_cluster, if want to have more cluster, then change the 'k_cluster' parameter.

EX: python kernelkmeans.py image1.png 3, python spectralclustering.py image1.png 4

(b) Spectural Clustering

Running command: python file.py imgname k_cluster initialization cut, if want to have more cluster, then change the 'k_cluster' parameter.

EX: python spectralclustering.py image1.png 3 kmeans++ ratio, python spectralclustering.py image2.png 3 random normalized

● Part3 (10%)

(a) Kernel K-means

```python
def initial(pixel, initial_method):
    if initial_method == 'random':
        classification = np.random.randint(K_cluster, size = pixel.shape[0])
    elif initial_method == 'modK':
        classification = []
        for i in range(data.shape[0]):
            classification.append(i % K_cluster)
        classification = np.asarray(classification)
    return classification
```

I use two initialization method: random and modK to do the cluster initialization.

(b) Spectral Clustering

```python
def initial(E, init_method):
    cluster = np.random.randint(0, K_cluster, scale * scale)
    # initialization using random
    if init_method == 'random':
        high = E.max(axis=0)
        low = E.min(axis=0)
        diff = high - low
        centroids = np.random.rand(K_cluster, K_cluster)
        for i in range(K_cluster):
            centroids[:, i] *= diff[i]
            centroids[:, i] += low[i]
    # initialization using kmeans++
    elif init_method == 'kmeans++':
        centroids = [E[np.random.choice(range(scale * scale)), :]]
        # find #K_cluster centroids
        for i in range(K_cluster - 1):
            dist = cdist(E, centroids, 'euclidean').min(axis=1)
            prob = dist / np.sum(dist)
            centroids.append(E[np.random.choice(range(scale * scale), p=prob)])
        centroids = np.array(centroids)

    return centroids, np.array(cluster)
```

In spectral clustering, I used two initial method: random and kmeans++ to do initialization.

● Part4 (10%)

```python
# use different cut
def CUT(W, D, cut):
    if cut == 'normalized':
        D_Square = np.diag(np.power(D, -0.5))
        L = np.identity(scale * scale) - D_Square @ W @ D_Square
        EigenValue, EigenVector = np.linalg.eig(L)
        idx = np.argsort(EigenValue)[1: K_cluster+1]
        U = EigenVector[:, idx].real.astype(np.float32)
        T = np.zeros(U.shape, dtype = np.float64)
        for i in range(scale * scale):
            T[i] = U[i] / np.sqrt(np.sum(U[i] ** 2))

        return T
    elif cut == 'ratio':
        L = D - W
        EigenValue, EigenVector = np.linalg.eig(L)
        idx = np.argsort(EigenValue)[1: K_cluster+1]
        U = EigenVector[:, idx].real.astype(np.float32)

        return U
```

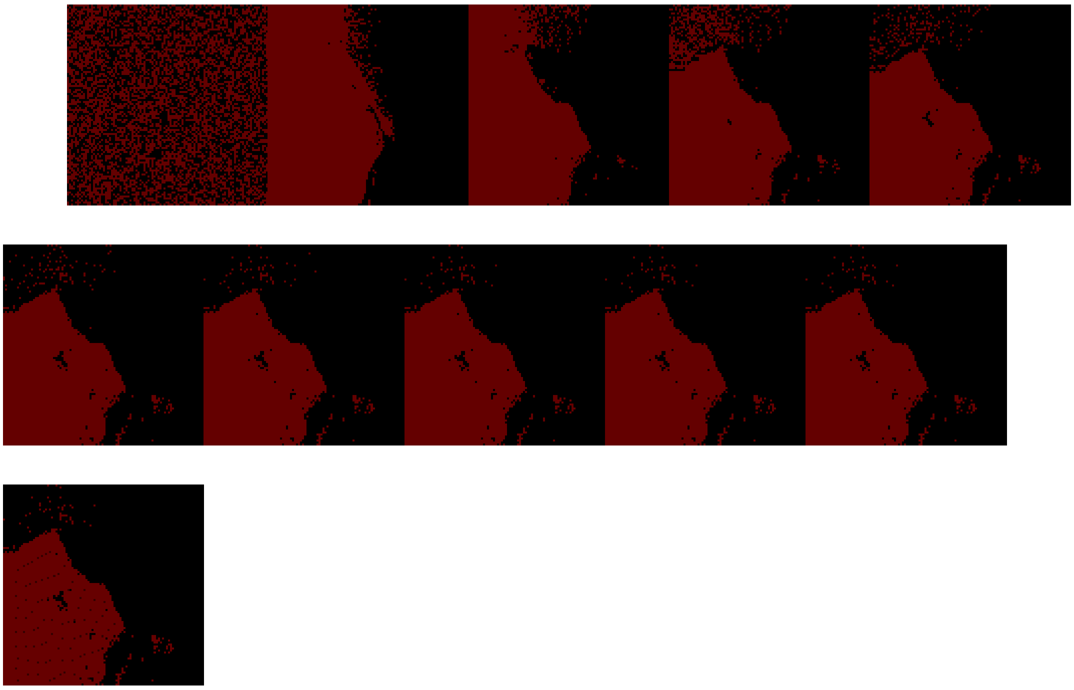
To do different cut method: normalized cut and ratio cut by calculating eigenvalue and eigenvector of laplacian matrix.

○ b. experiments settings and results (20%) & discussion (30%)

● Part1 (5%) & (5%)

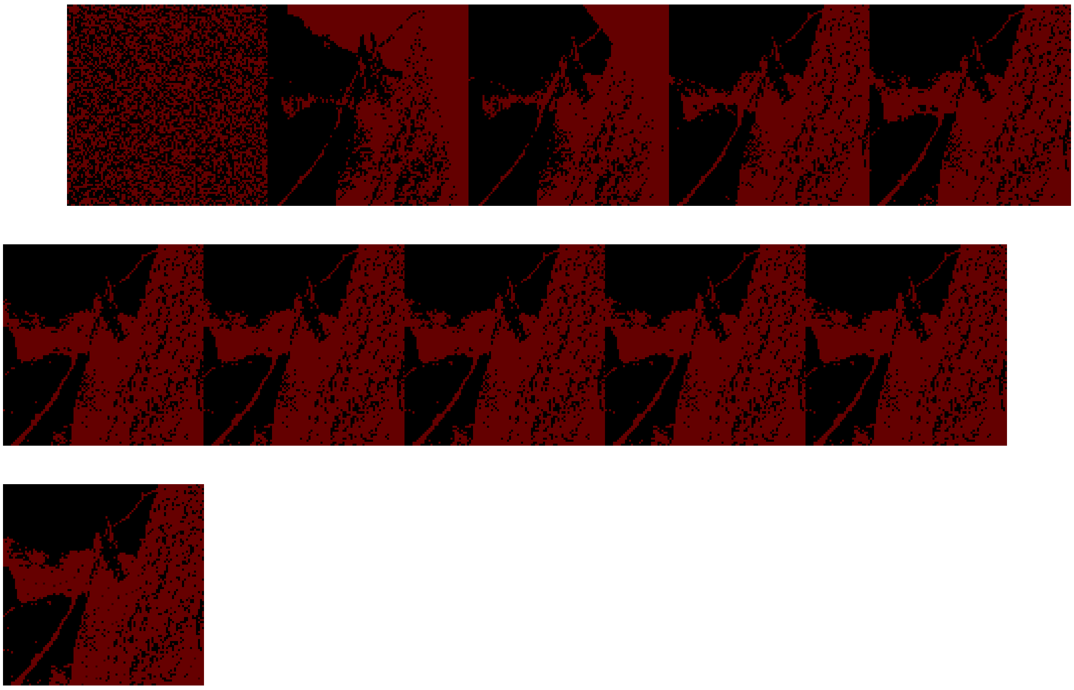
(a) Kernel K-means: cluster number = 2 (initialization: random)

1. Image1



Iteration 0~15

2. Image2

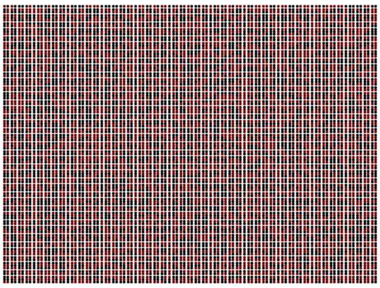


Iteration 0~15

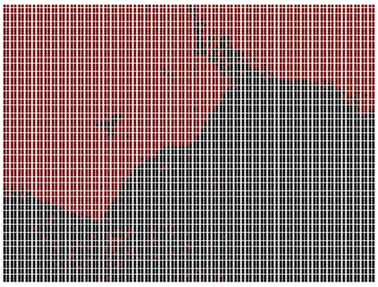
(b) Spectural Clustering : cluster number = 2

(initialization: random, cut: normalized)

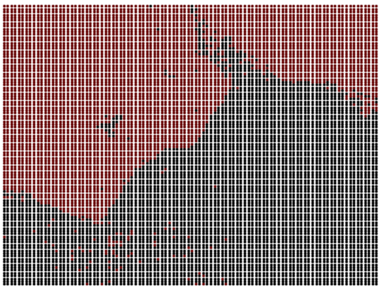
1. Image1



Iteration 0

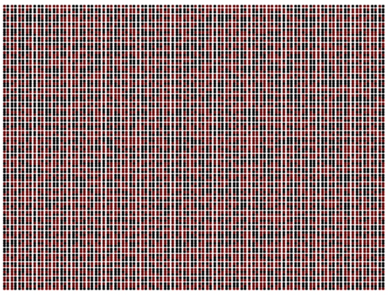


iteration3

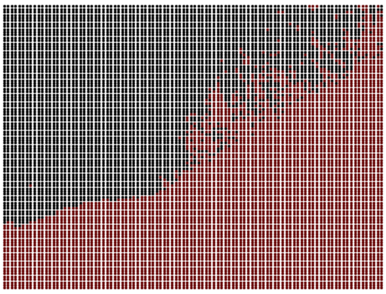


iteration6

2. Image2

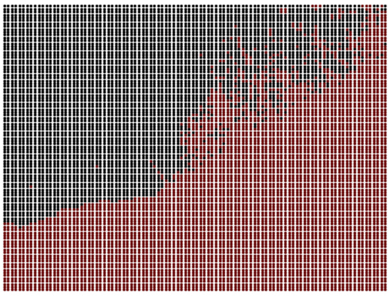


Iteration 0



iteration 3

iteration 6



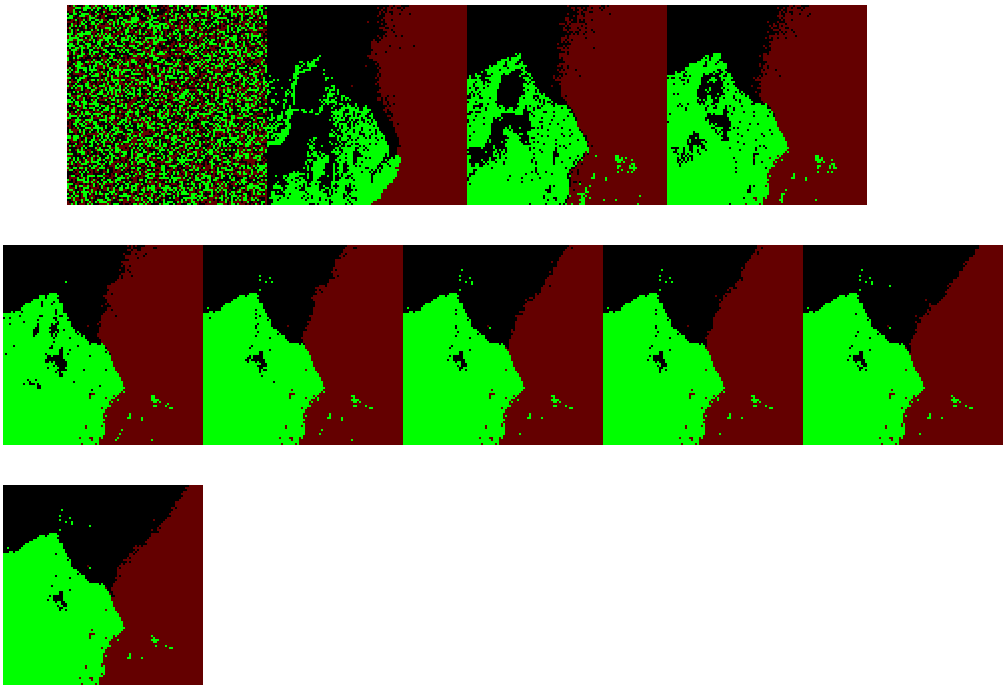
Iteration 9

● Part2 (5%) & (5%)

(a) Kernel K-means (initialization: random)

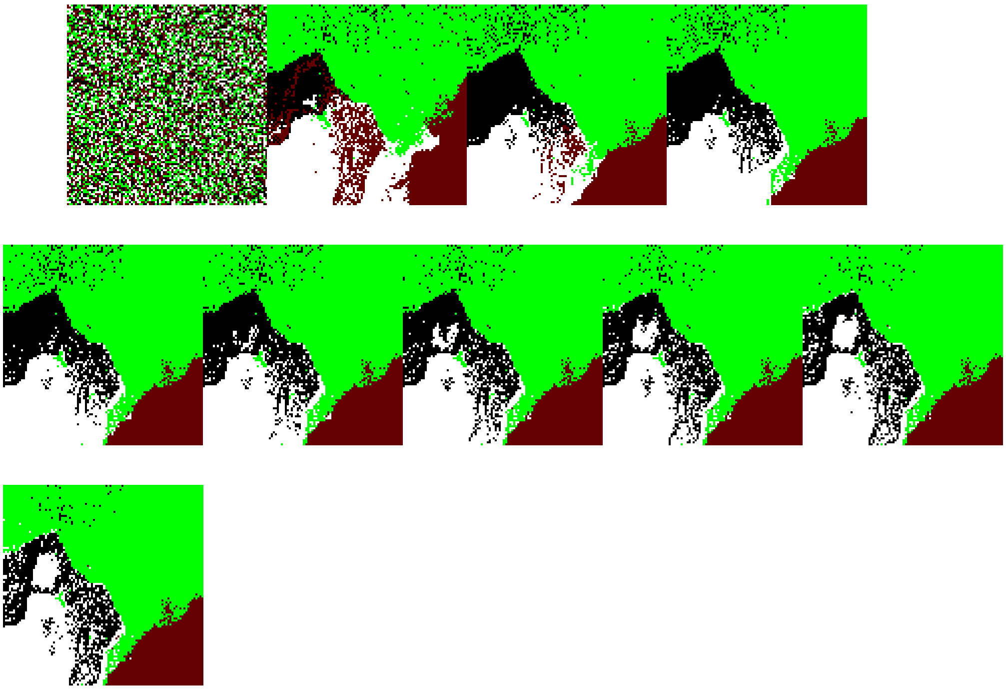
1. Image1

(i). Cluster number =3



Iteration (0~15)

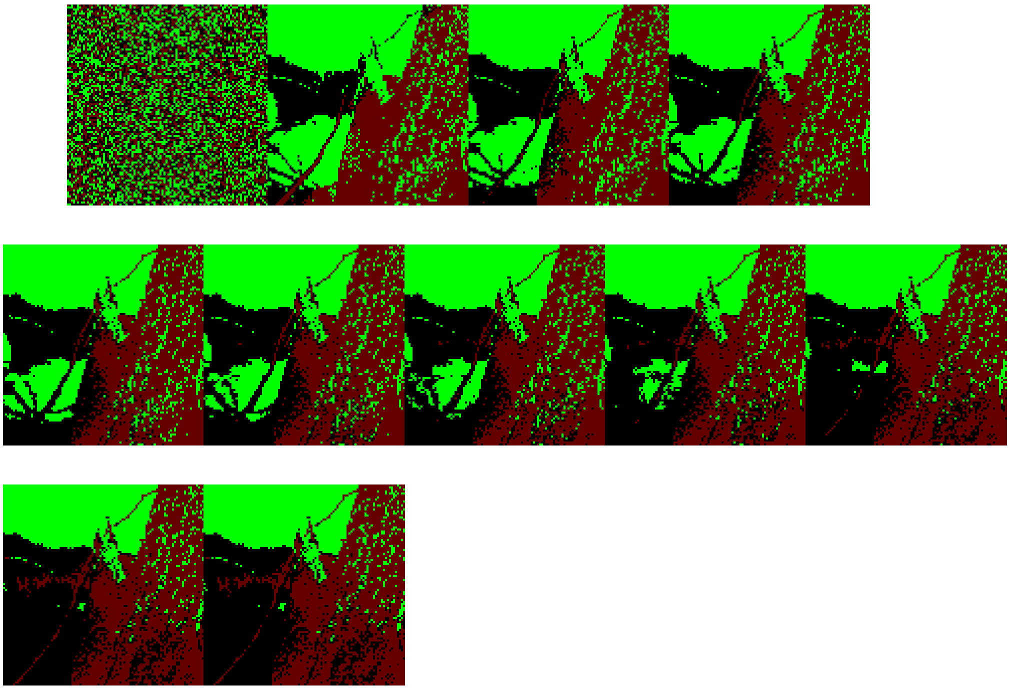
(ii). Cluster number = 4



Iteration (0~15)

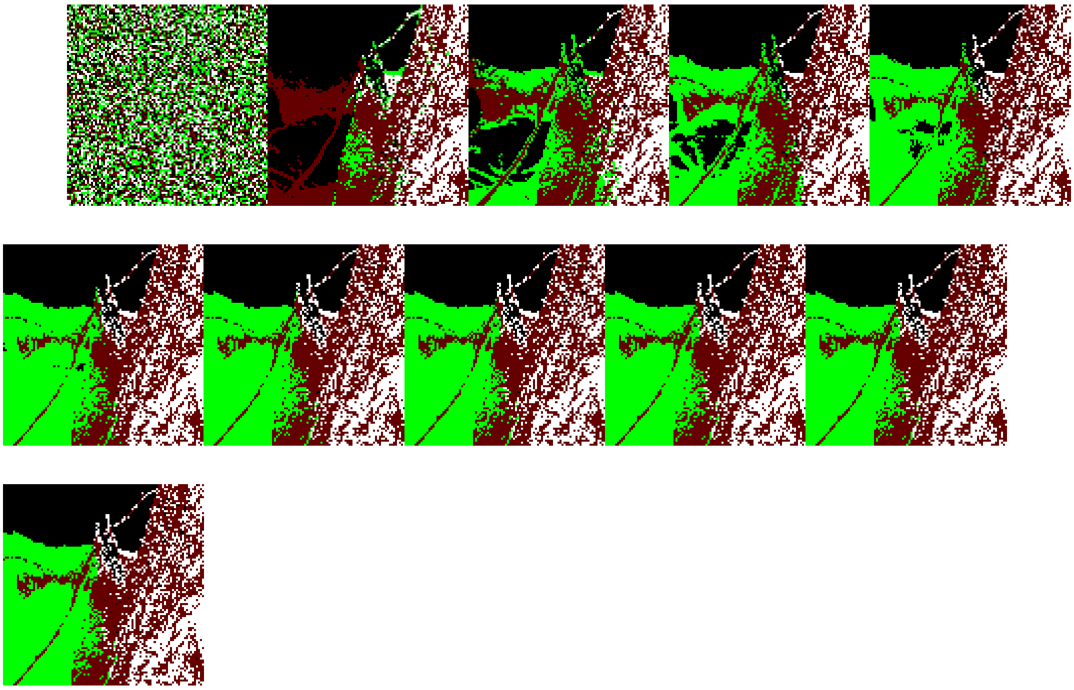
2. Image2

(i). Cluster number = 3



Iteration 0~15

(ii). Cluster number = 4

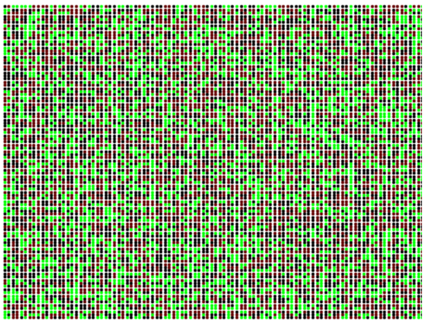


Iteration 0~15

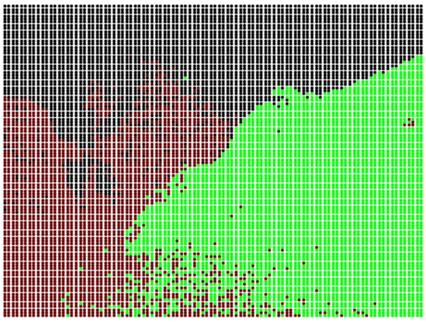
(b) Spectural Clustering (initialization: random, cut: normalized)

1. Image1

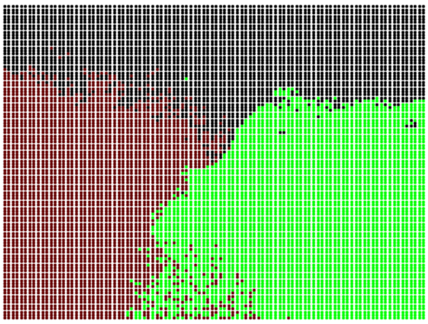
(i). Cluster number =3



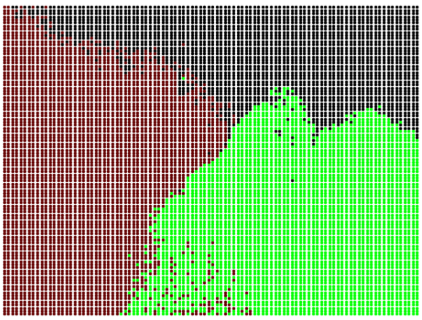
iteration 0



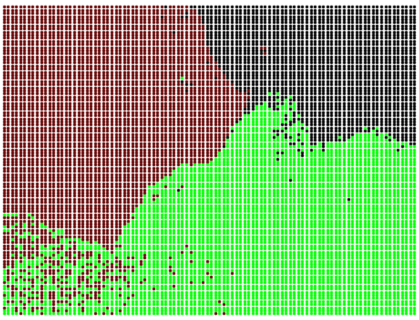
iteration 3



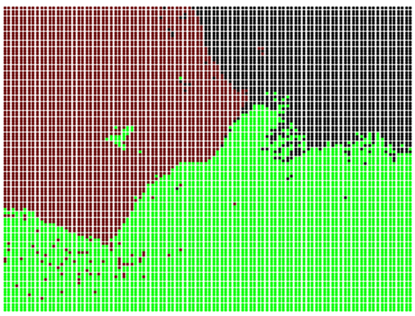
iteration 6



Iteration 9

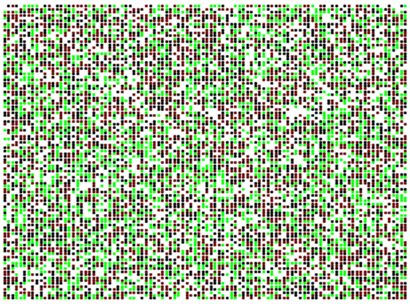


iteration 12

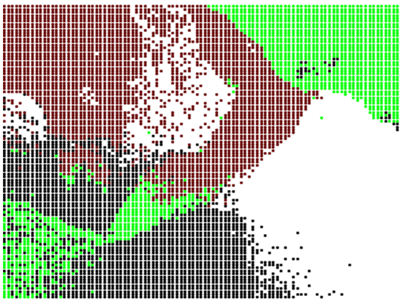


iteration 15

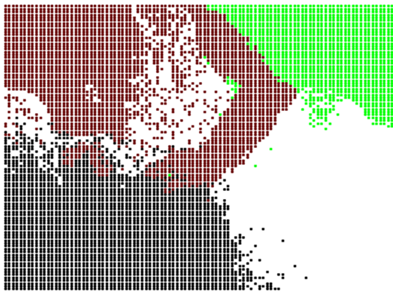
(ii). Cluster number = 4



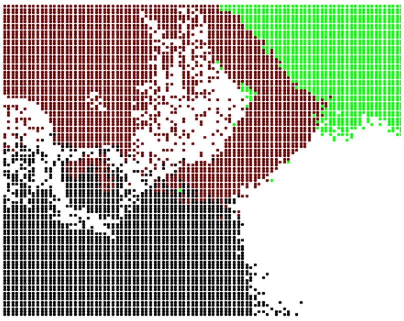
Iteration 0



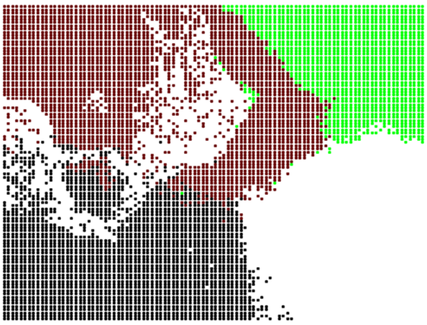
iteration 3



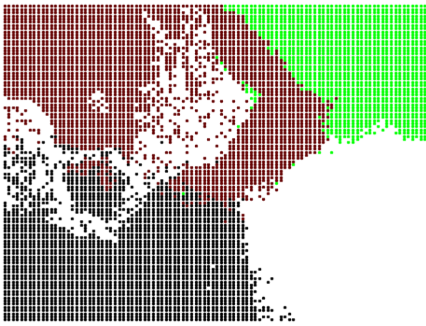
iteration6



Iteration 9



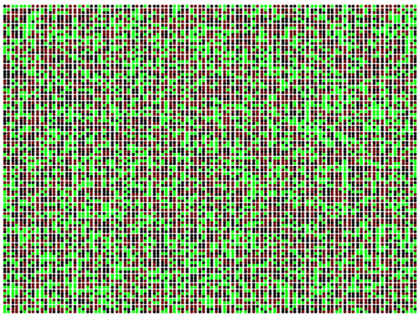
iteration 12



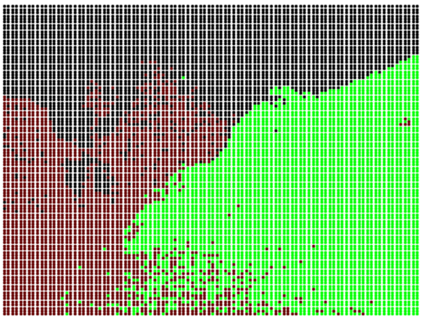
iteration 15

2. Image2

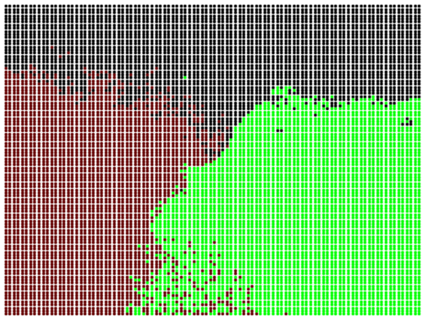
(i). Cluster number = 3



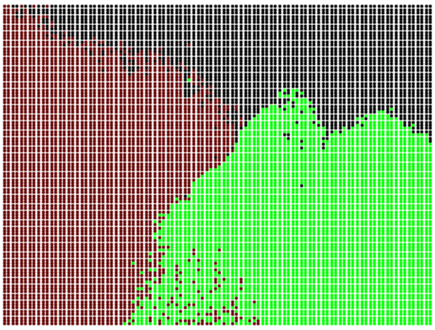
iteration 0



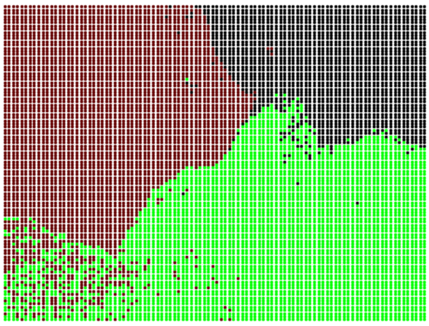
iteration 3



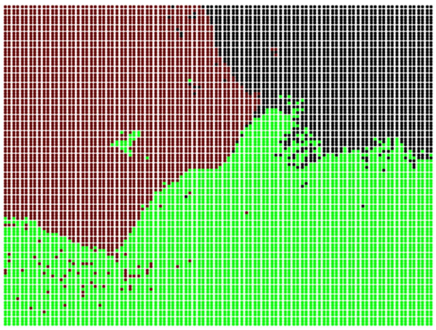
iteration 6



iteration 9

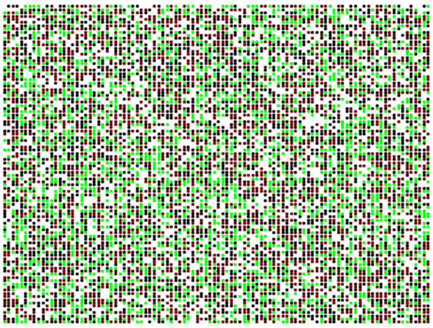


iteration12

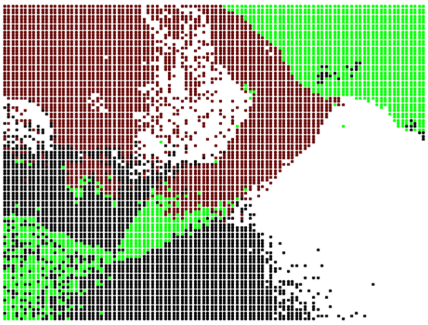


iteration 15

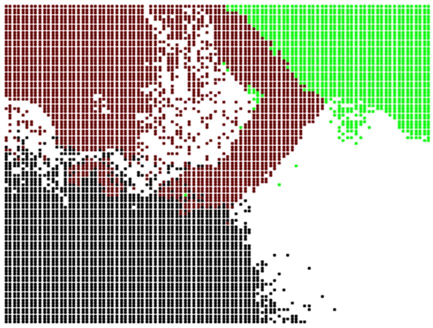
(ii). Cluster number = 4



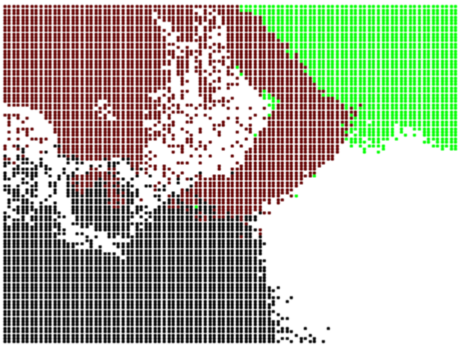
Iteration 0



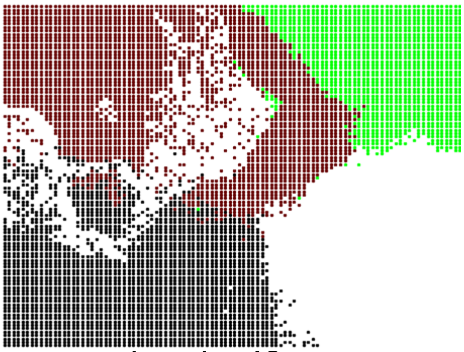
iteration 3



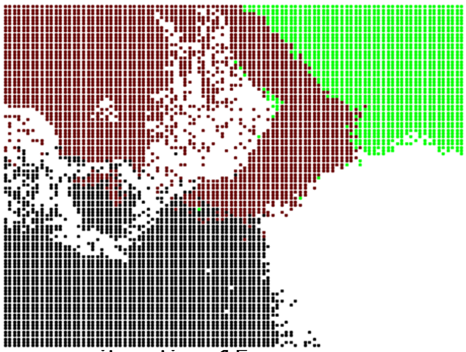
iteration6



Iteration 9



iteration 12



iteration 15

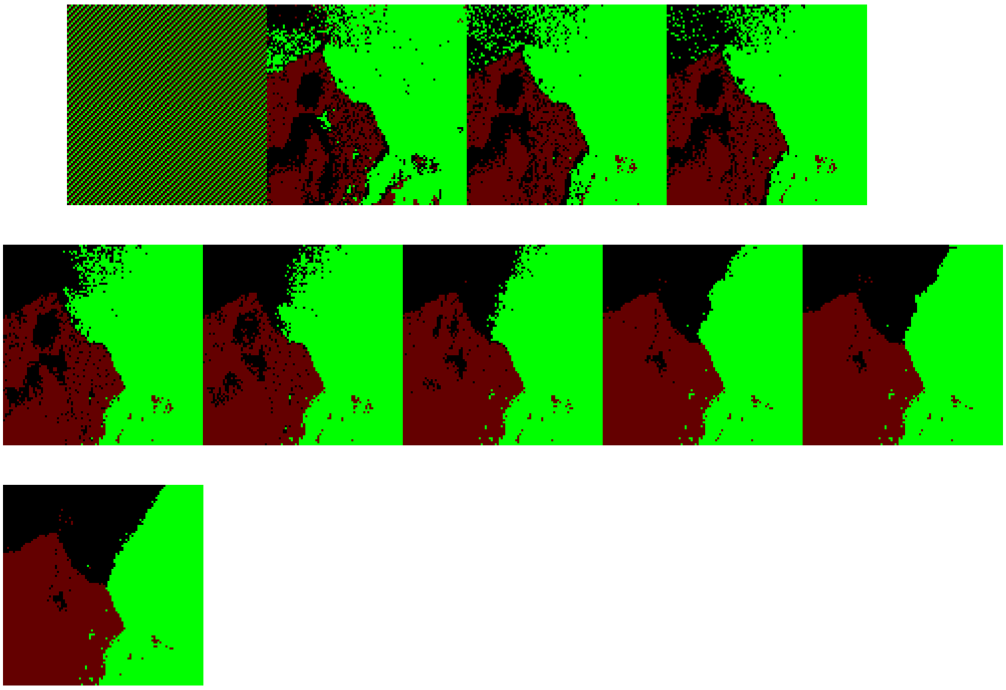
● Part3 (5%) & (10%)

(a) Kernel K-means

Initialization: **mod k** (results of random initialization method has shown above.)

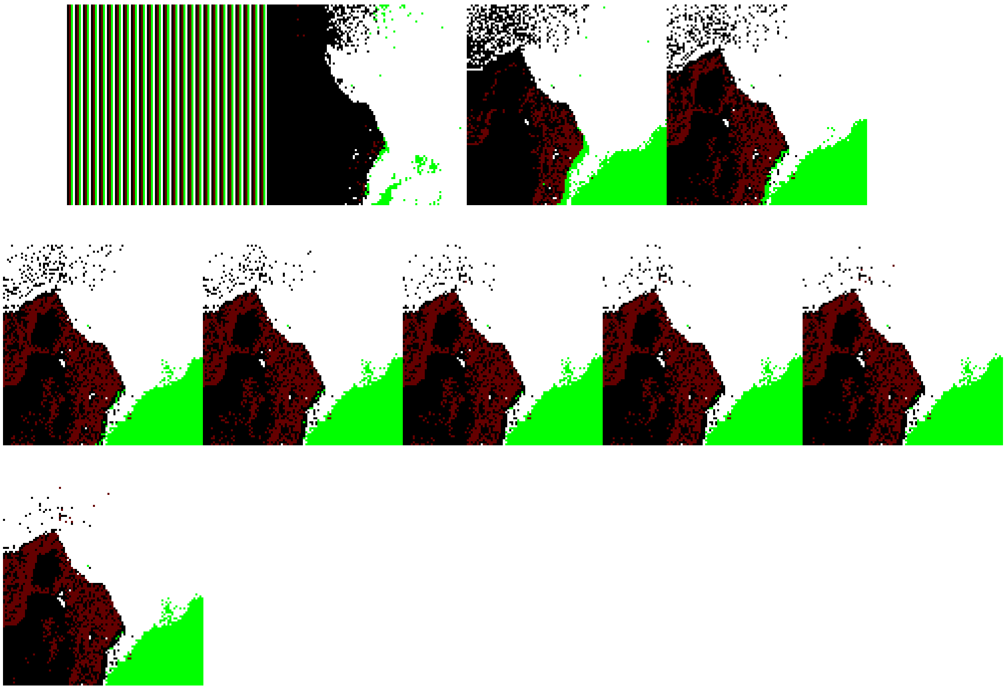
1. Image1

(i). Cluster number =3



Iteration 0~15

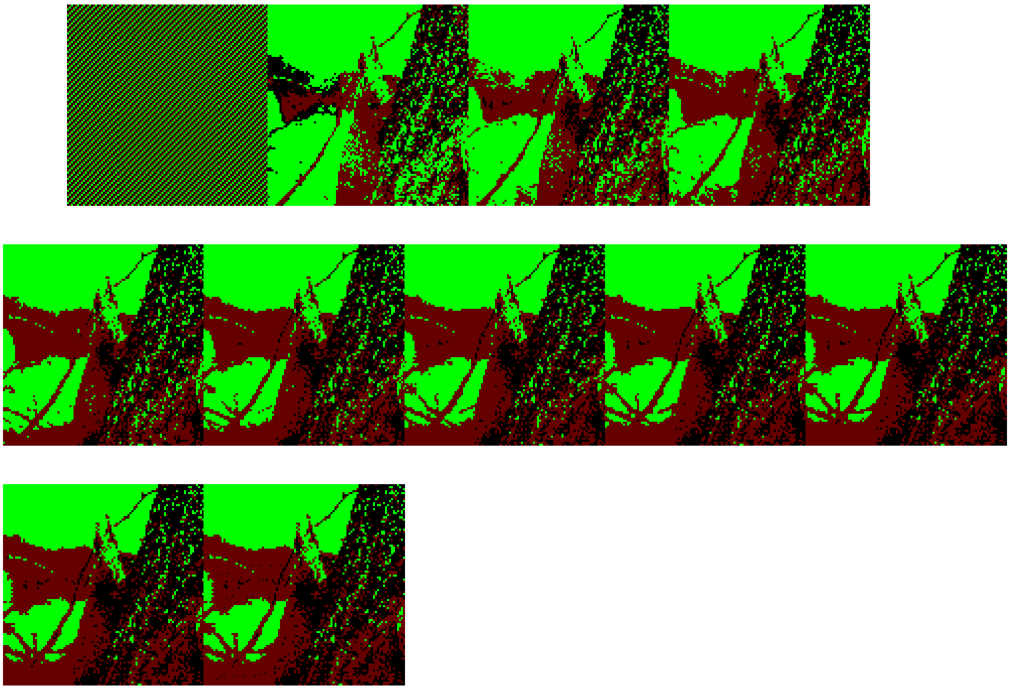
(ii). Cluster number = 4



Iteration 0~15

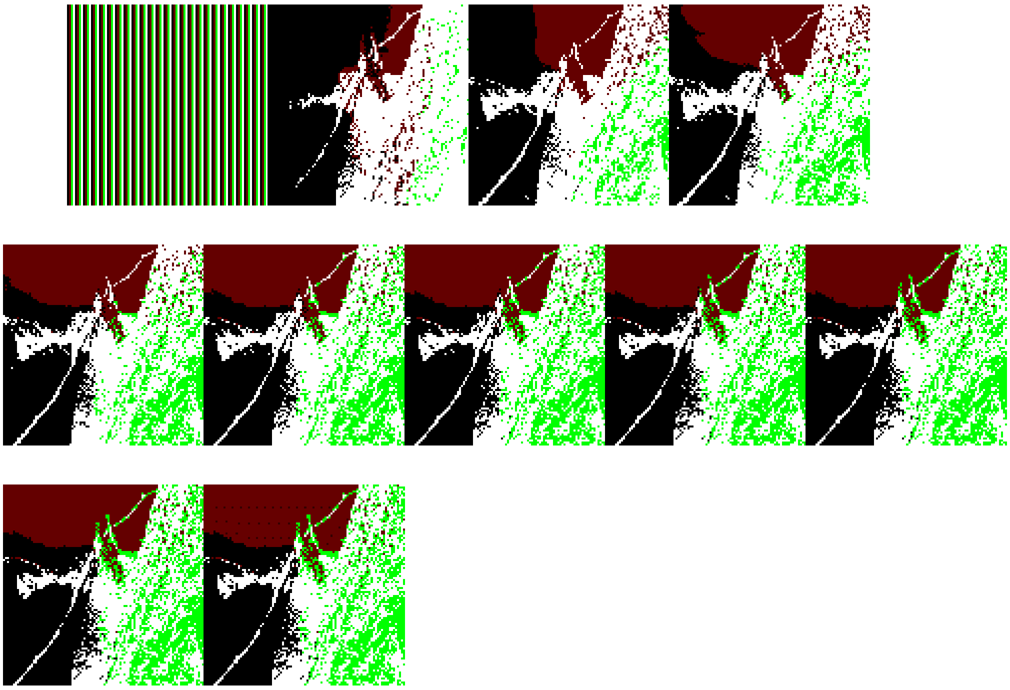
2. Image2

(i). Cluster number =3



Iteration 0~15

(ii). Cluster number = 4



Iteration 0~15

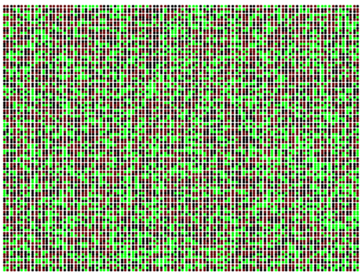
(b) Spectural Clustering

Initialization: **kmeans++** (results of random initialization method has shown above.)

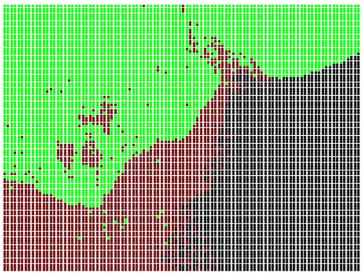
1. Image1

Normalization cut

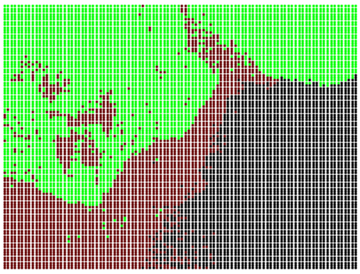
(i). Cluster number =3



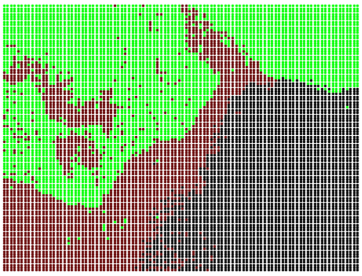
Iteration 0



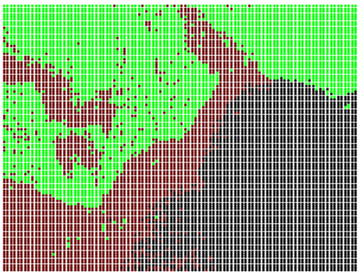
iteration 3



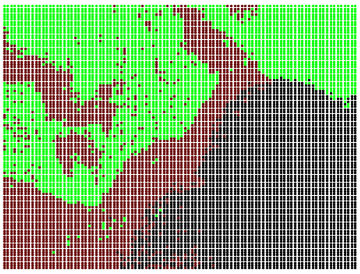
iteration6



Iteration 9

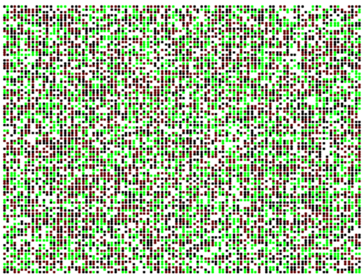


iteration 12

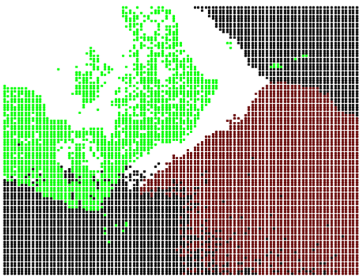


iteration 15

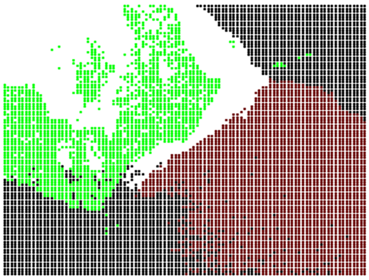
(ii). Cluster number = 4



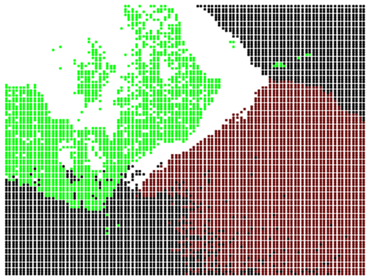
Iteration 0



iteration 3



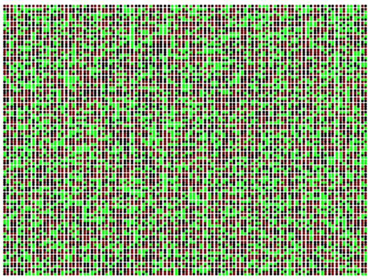
iteration 6



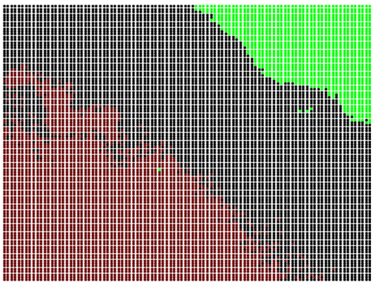
iteration 9

Ratio cut

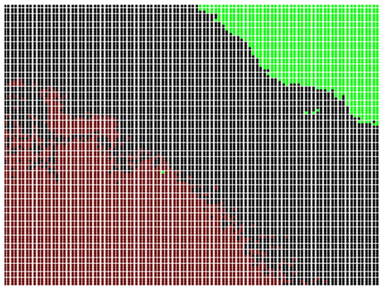
(i). Cluster number =3



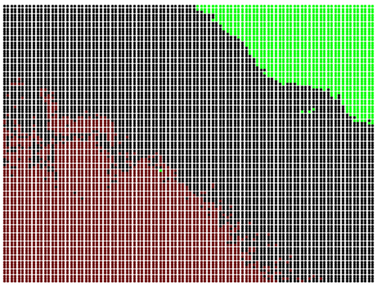
Iteration 0



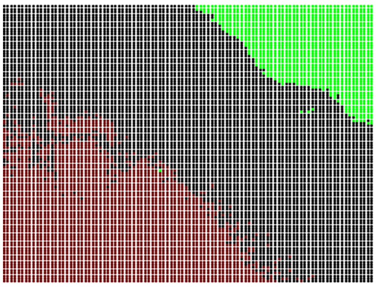
iteration 3



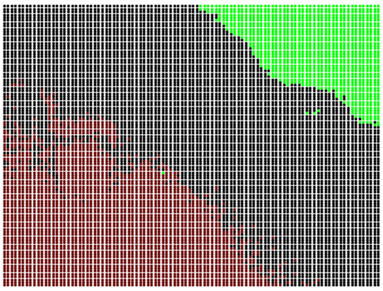
iteration 6



iteration 9

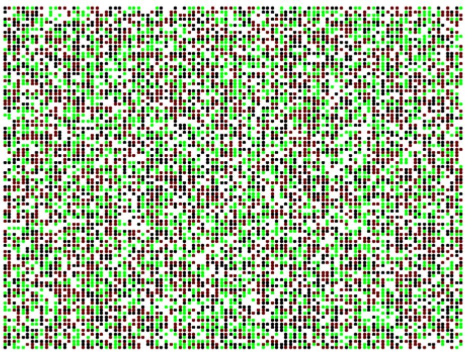


iteration 12

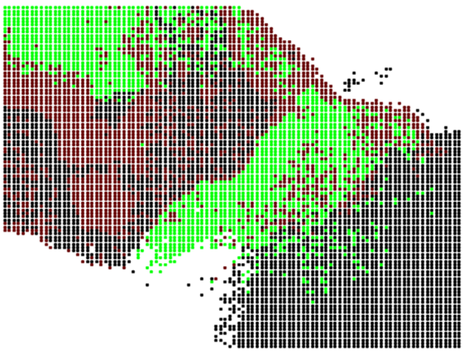


iteration 15

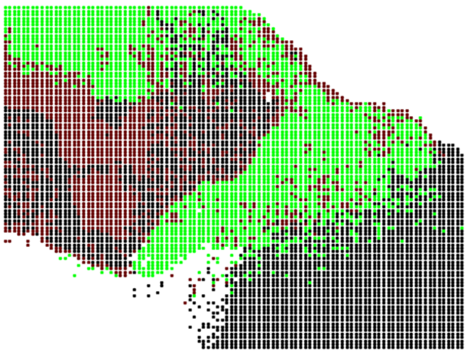
(ii). Cluster number = 4



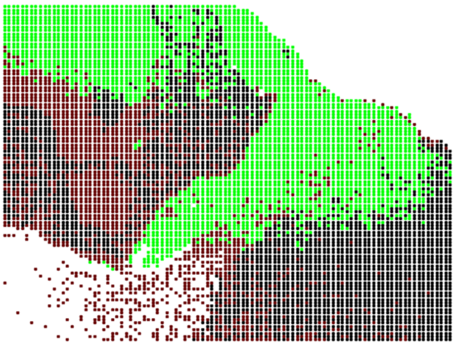
Iteration 0



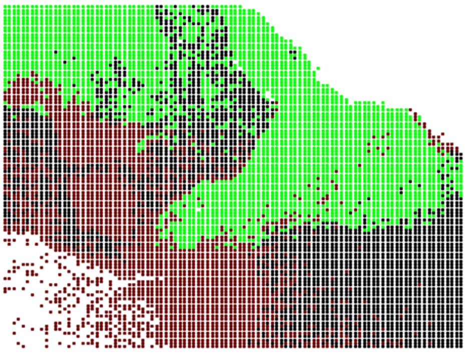
iteration 3



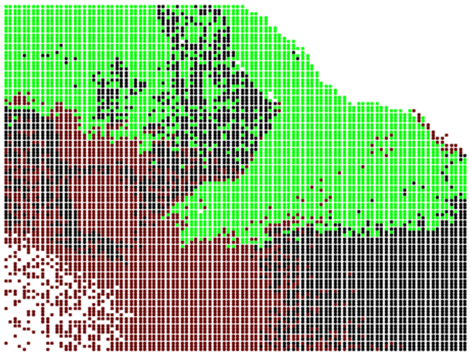
iteration 6



Iteration 9



iteration 12

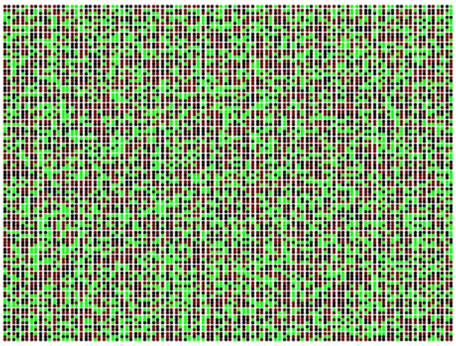


iteration 15

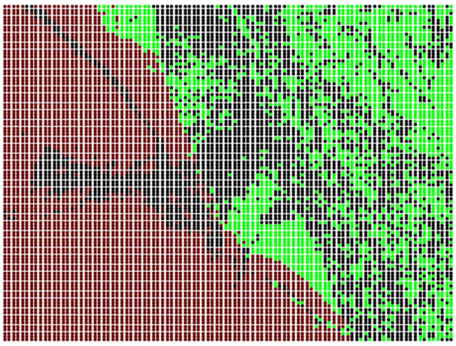
2. Image2

Normalization cut

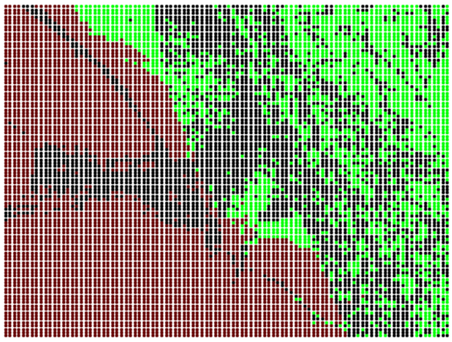
(i). Cluster number =3



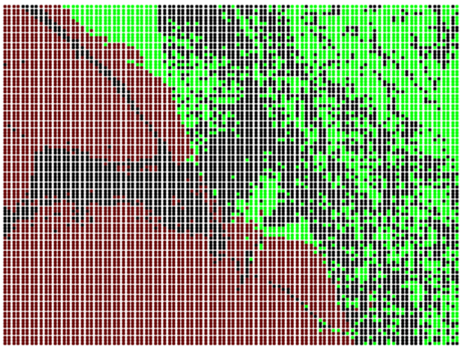
Iteration 0



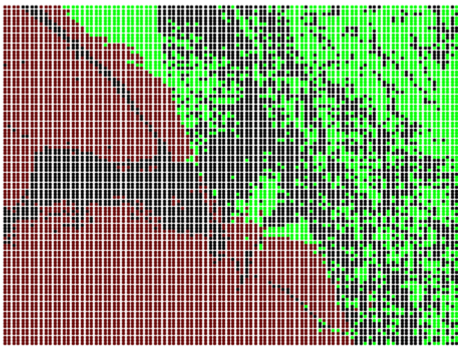
iteration 3



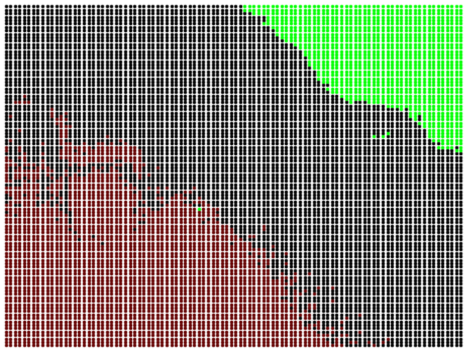
iteration 6



Iteration 9

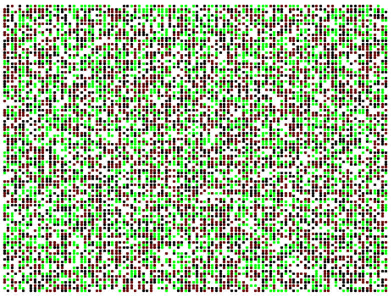


iteration 12

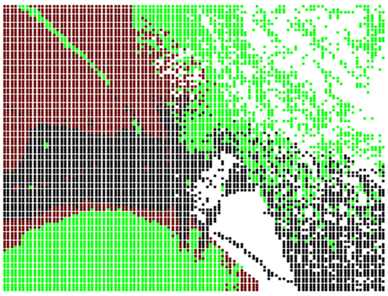


iteration 15

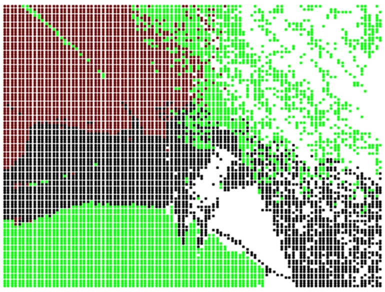
(ii). Cluster number = 4



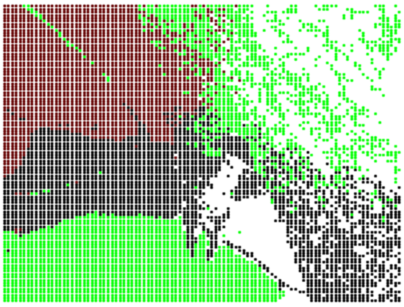
Iteration 0



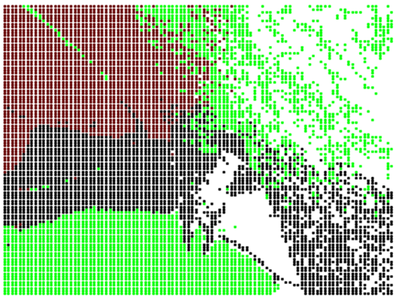
iteration 3



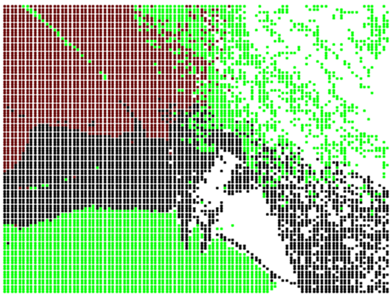
iteration 6



Iteration 9



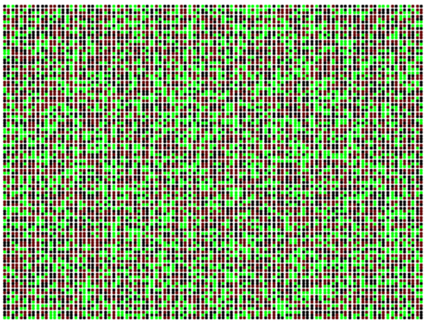
iteration 12



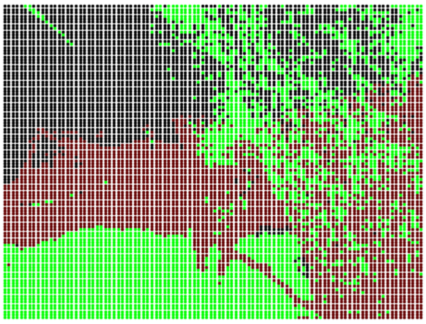
iteration 15

Ratio cut

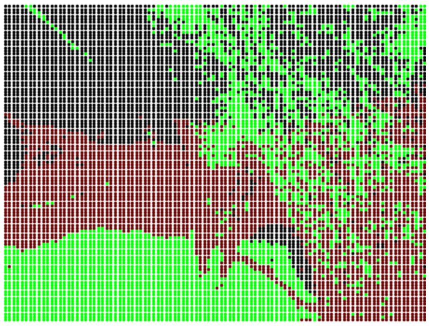
(i). Cluster number =3



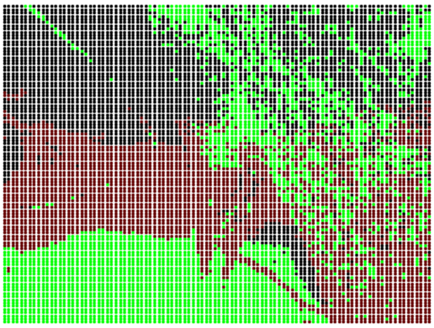
Iteration 0



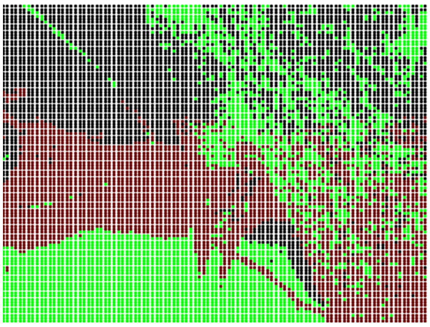
iteration 3



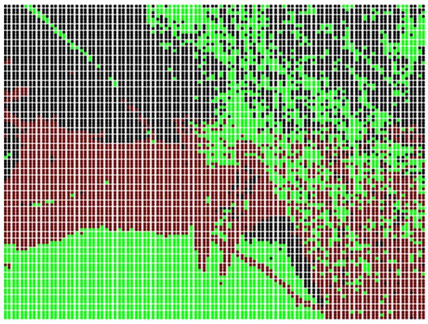
iteration 6



Iteration 9

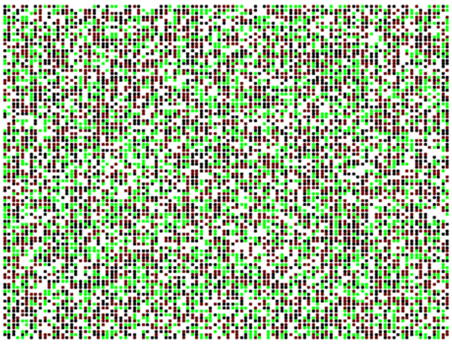


iteration 12

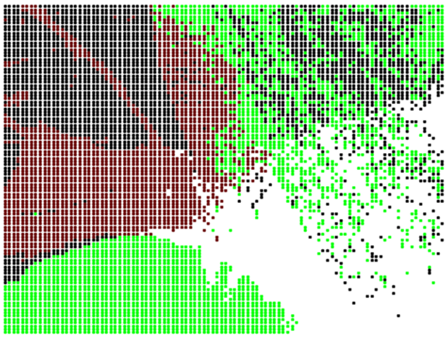


iteration15

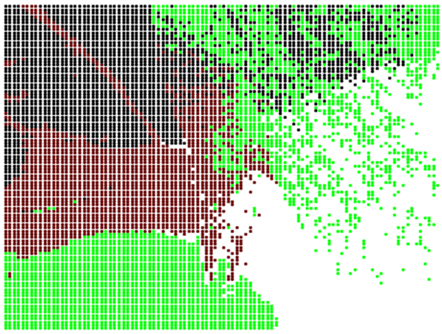
(ii). Cluster number = 4



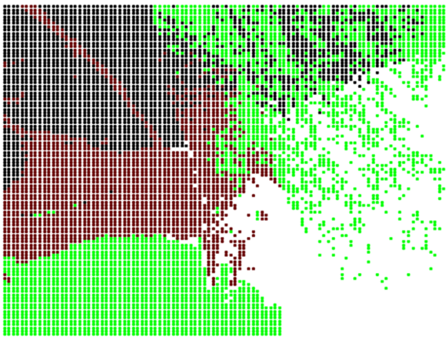
iteration 0



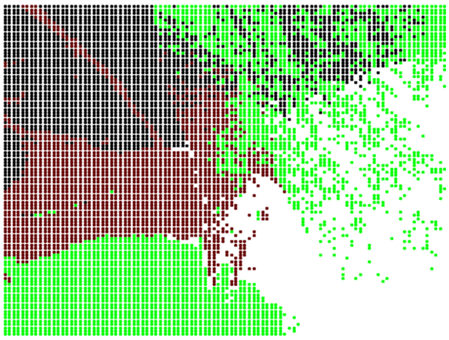
iteration 3



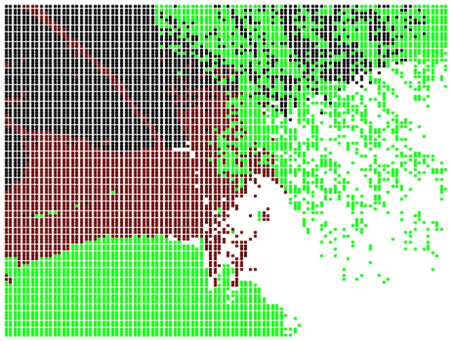
iteration 6



iteration 9



iteration 12

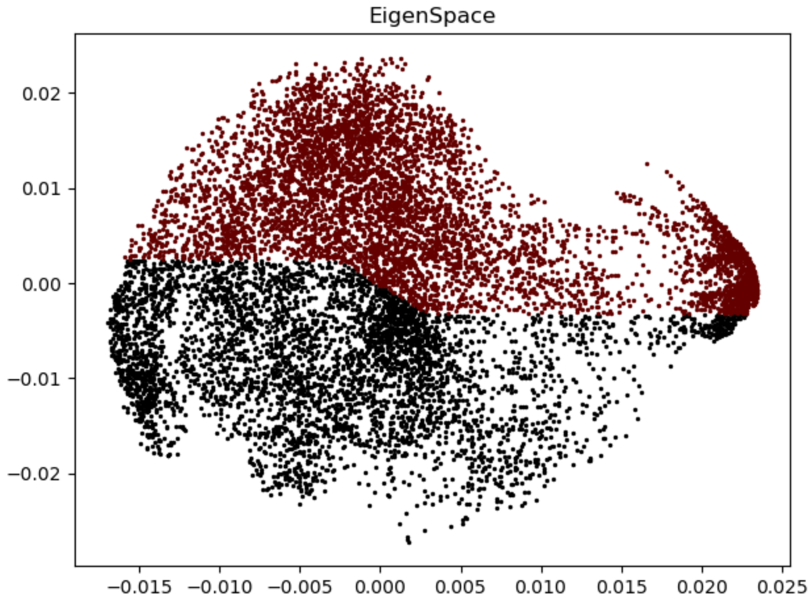


iteration 15

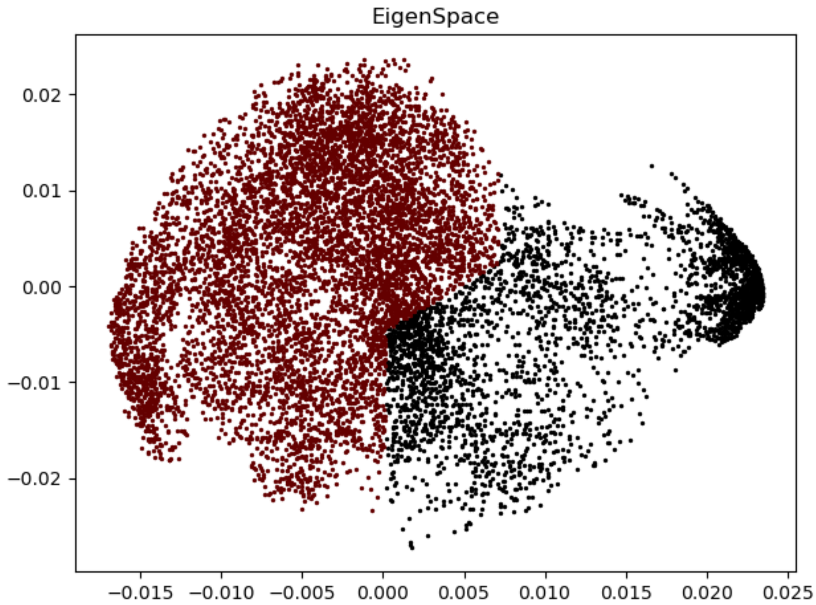
● Part4 (5%) & (10%)

Eigen space of spectral clustering

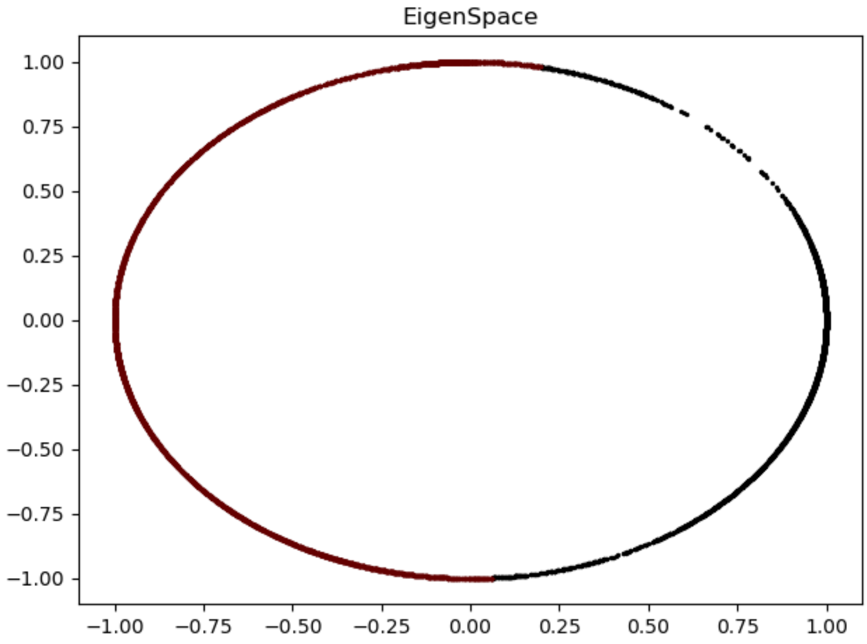
I. Image1 (k_cluster = 2, Ratio cut, Init: Kmeans++)



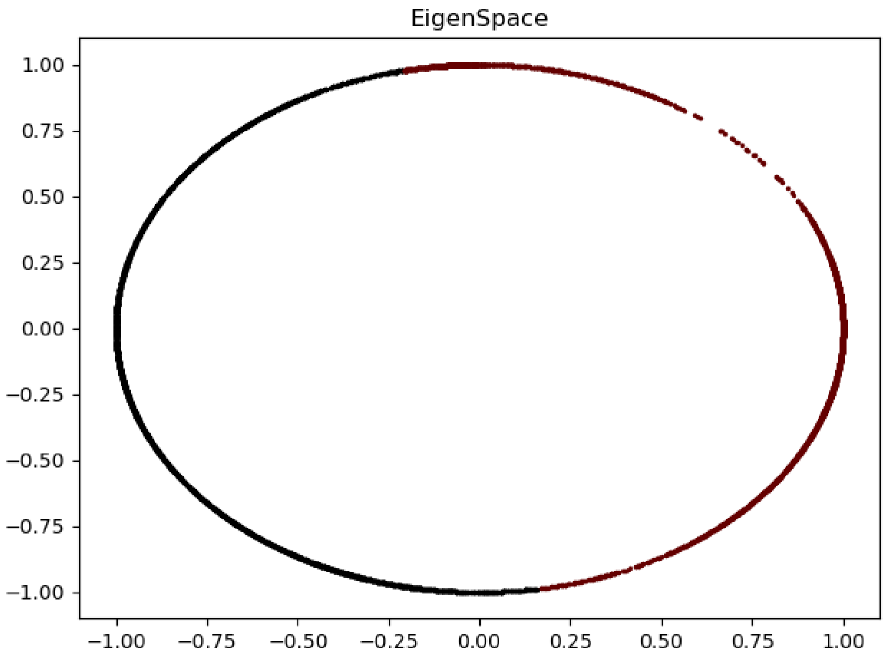
(k_cluster = 2, Ratio cut, Init: random)



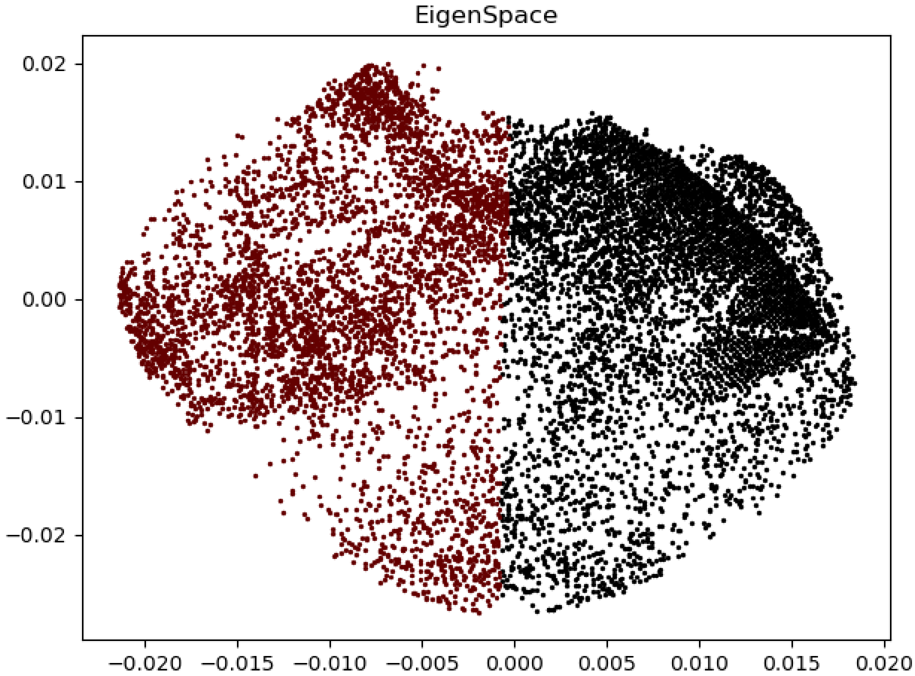
(k_cluster = 2, Normalized cut, Init: kmeans++)



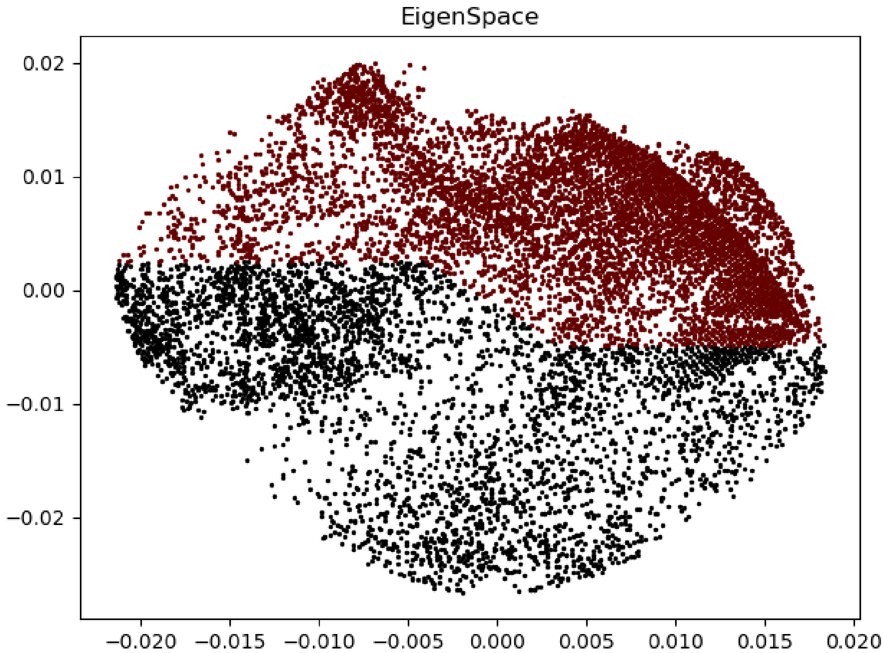
(k_cluster = 2, Normalized cut, Init: random)



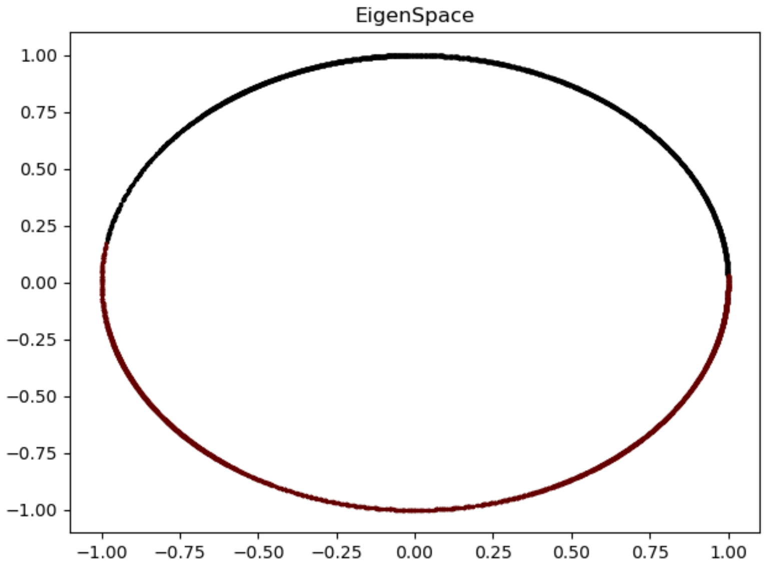
2. Image2 (k_cluster = 2, Ratio cut, Init: Kmeans++)



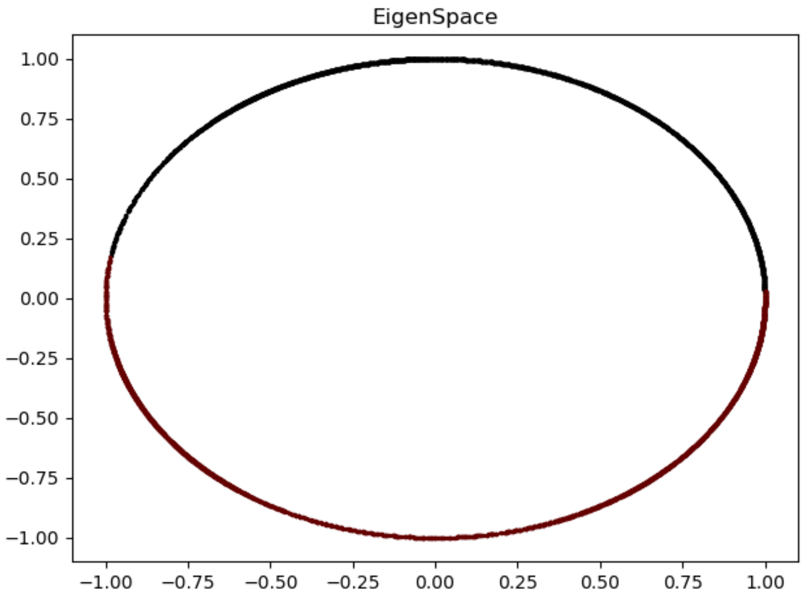
(k_cluster = 2, Ratio cut, Init: random)



(k_cluster = 2, Normalized cut, Init: kmeans++)

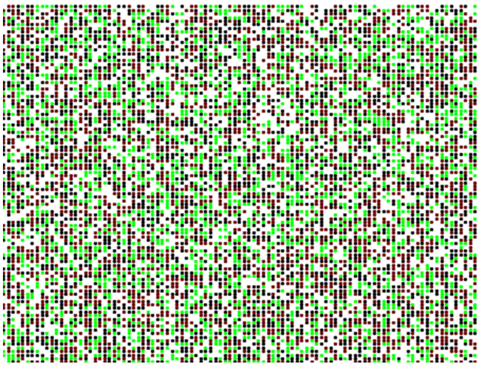


(k_cluster = 2, Normalized cut, Init: random)

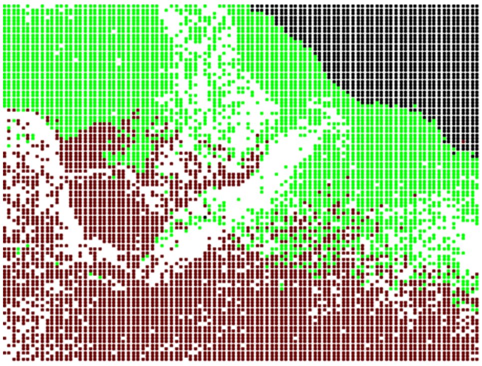


○ c. observations and discussion (10%)

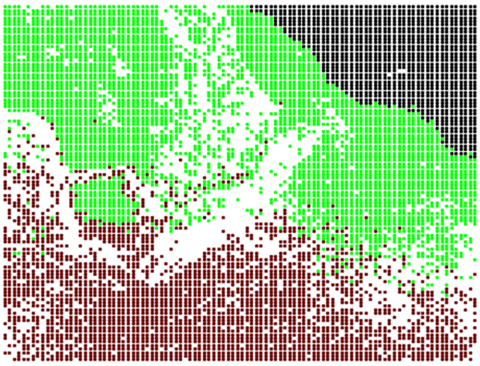
In this lab, I tried to fine tune parameters **gamma_c** and **gamma_s**, but it turned out that the results become strange. So I tried to set gamma_c = 1e-5 and gamma_s =1e-5 to do the experiment on (image1, ratio cut, random initialization), and the results are shown bellow:



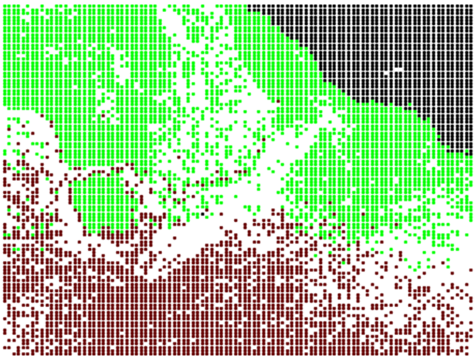
Iteration 0



iteration 3



iteration 6



Iteration 9