위치와 색상에 따른 분류

20151795 김동현
20151870 박정재
20150256 이용준

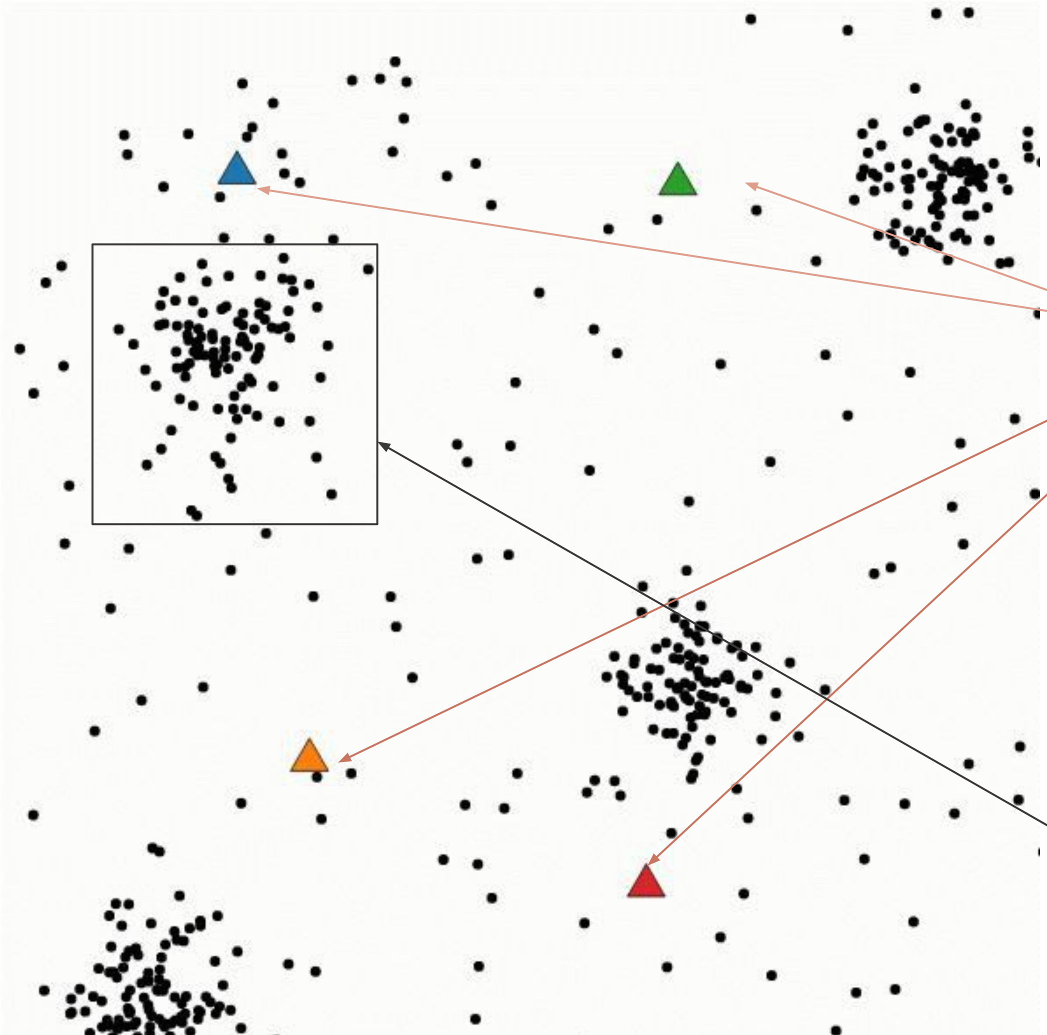
Table of Contents

목차

Part 1

K-means algorithm

K-means algorithm?

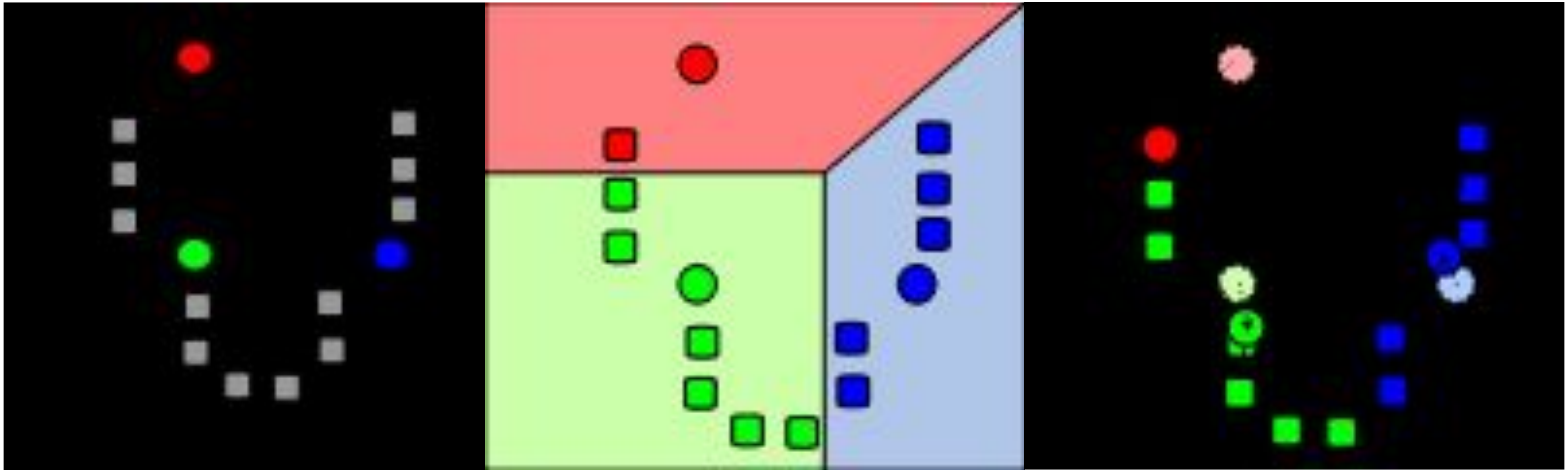


데이터를 무리 짓는 방법

- Centroid
- Distance
- Cluster

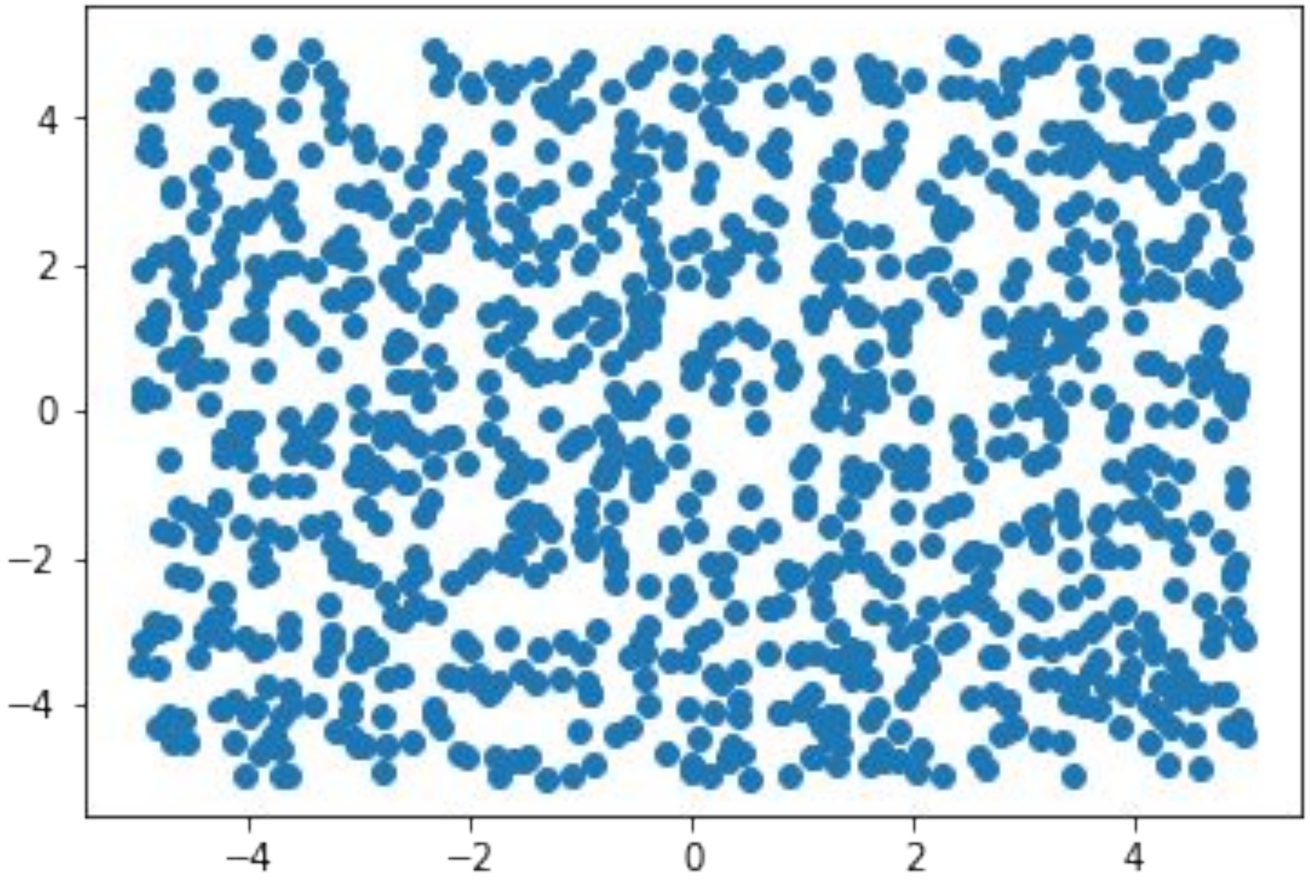
K-means algorithm?

- 중심 설정
- 군집화
- 중심 조정




Part 2
위치에 따른 분류

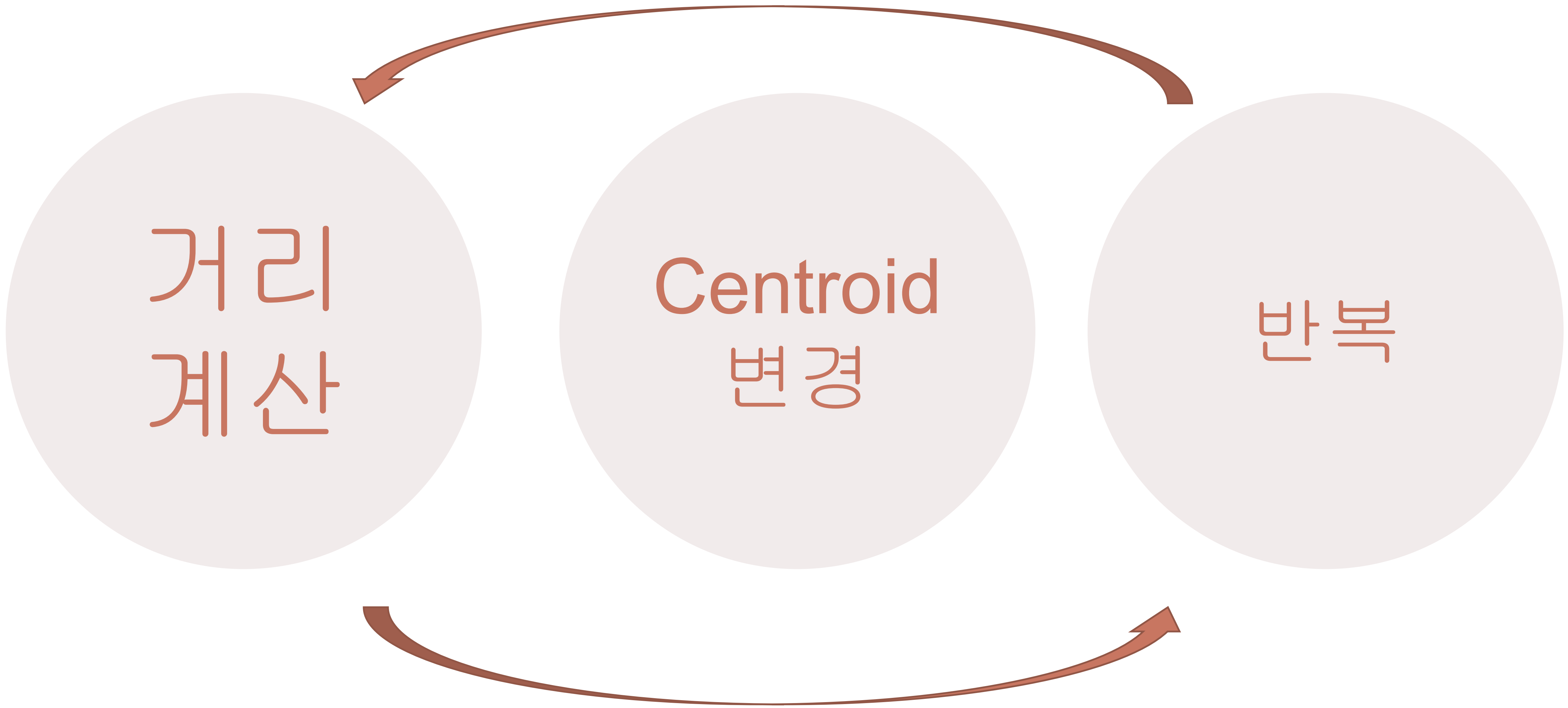
Scatter Plot 분류



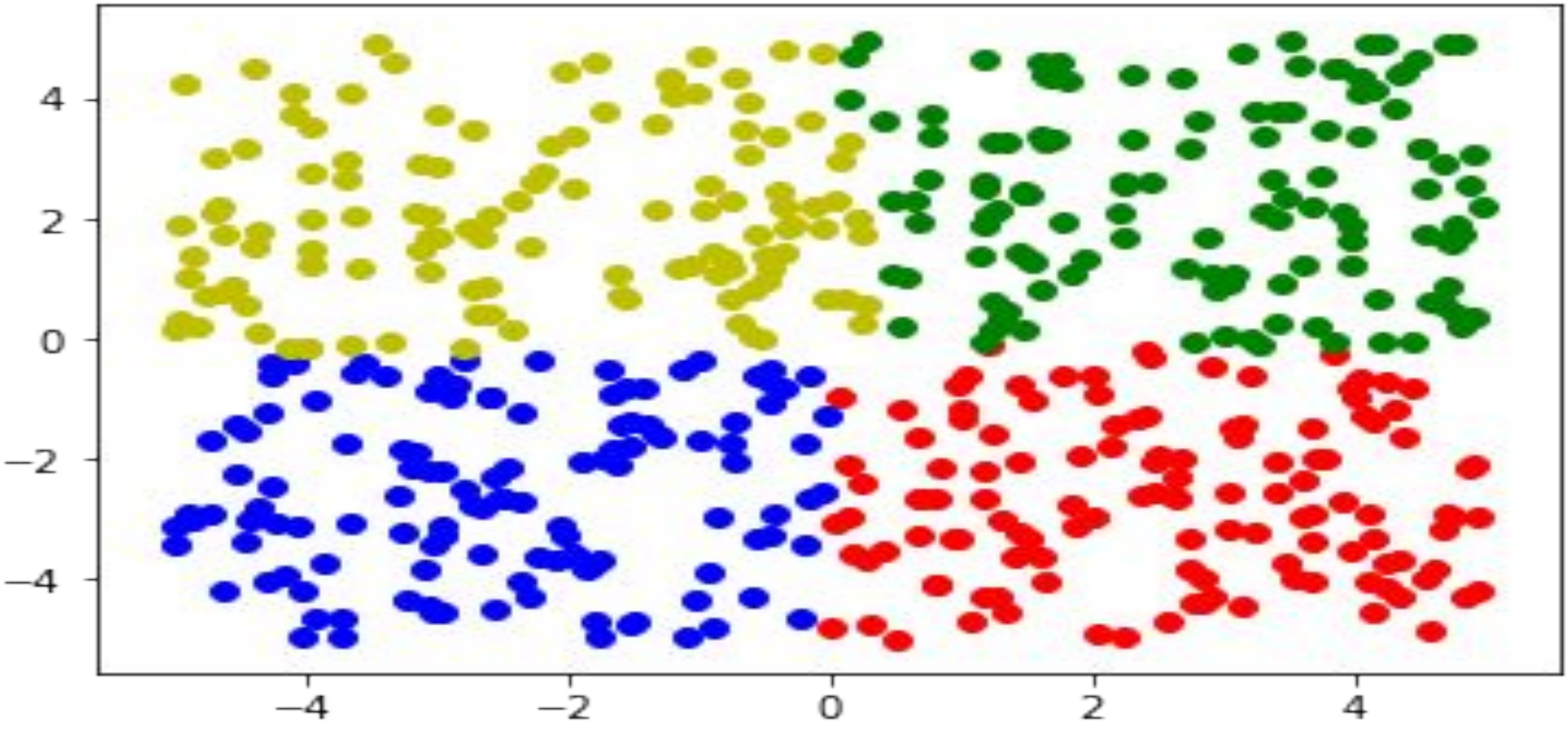
●완전히 무작위하게 찍은 1000개의 점

```
def GeneratePointCluster(n_points):
    x_points = []
    y_points = []

    for i in range(n_points):
        x_points.append(random.uniform(-5.0, 5.0))
        y_points.append(random.uniform(-5.0, 5.0))

#x,y 를 각 점에 리스트 자료형을 이용해서 대입한다.

    points = [x_points, y_points]

    return points

data = GeneratePointCluster(1000)
plt.scatter(data[0],data[1])
```

Scatter Plot 분류 과정



Scatter Plot 분류 완료



Part 3

이미지 분류

순서

이미지 불러오기

Shape of the Image

$$\begin{array}{cc} height & \begin{bmatrix} [r,g,b] & \cdots & [r,g,b] \\ \vdots & \ddots & \vdots \\ [r,g,b] & \cdots & [r,g,b] \end{bmatrix} \\ & width \end{array}$$

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

```python
# read image using opencv library.
img = cv2.imread('images.jpg', cv2.IMREAD_COLOR)
height = img.shape[0]
width = img.shape[1]
b, g, r = cv2.split(img)
img2 = cv2.merge([r, g, b])

img2.shape
```

```
(168, 300, 3)
```

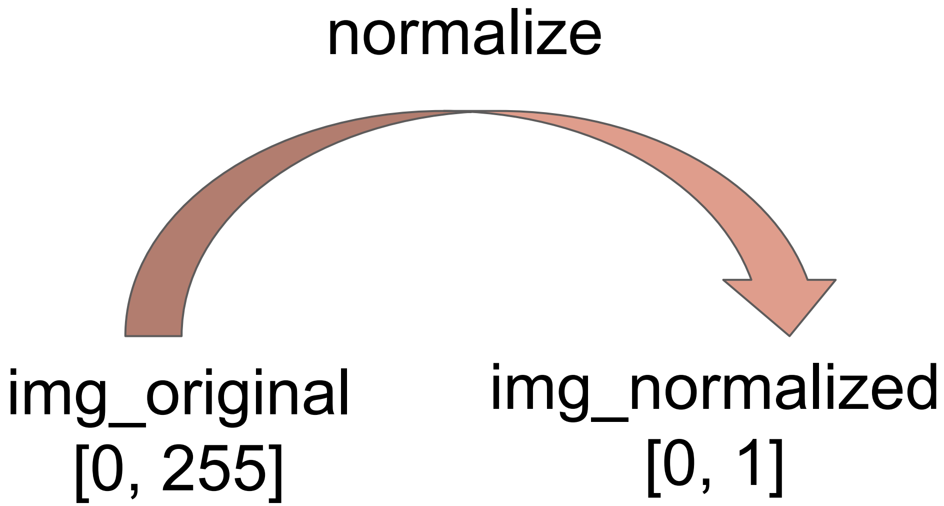
Input Image

Normalize Image Values

```
# normalize image values.
img_normalized = np.empty((height, width, 3), dtype=np.float64)
img_normalized = img_normalize(img2, height, width)
```

Normalize Function

```
def img_normalize(data, height, width):

    normalized_data = np.empty((height, width, 3), dtype=np.float64)

    for y in range(height):
        for x in range(width):
            normalized_data[y][x] = data[y][x] / np.array([255,255,255])

    return normalized_data
```



Allocate Initial Centroids Values Randomly

```
# Allocate centroids values.
c1_rgb = np.empty(3, dtype=np.float64)
c2_rgb = np.empty(3, dtype=np.float64)
c3_rgb = np.empty(3, dtype=np.float64)
c4_rgb = np.empty(3, dtype=np.float64)
c5_rgb = np.empty(3, dtype=np.float64)
c6_rgb = np.empty(3, dtype=np.float64)
c7_rgb = np.empty(3, dtype=np.float64)
c8_rgb = np.empty(3, dtype=np.float64)
c9_rgb = np.empty(3, dtype=np.float64)

c1_x = random.randint(0, width)
c1_y = random.randint(0, height)
c1_rgb = img_normalized[c1_y][c1_x]
print(c1_x, c1_y, img2[c1_y][c1_x])
```

c1_x = random_x
c1_y = random_y

c1_rgb = img_normalized[c1_y][c1_x]

```python
# show the location of initial centroids.
plt.imshow(img2)
plt.axis('off')
plt.title('The location of Initial Centroids')

plt.plot([c1_x], [c1_y], marker='*', color='red', label='centroid1', markersize=20)
plt.plot([c2_x], [c2_y], '*', color='orange', label='centroid2', markersize=20)
plt.plot([c3_x], [c3_y], '*', color='yellow', label='centroid3', markersize=20)
plt.plot([c4_x], [c4_y], '*', color='salmon', label='centroid4', markersize=20)
plt.plot([c5_x], [c5_y], '*', color='blue', label='centroid5', markersize=20)
plt.plot([c6_x], [c6_y], '*', color='purple', label='centroid6', markersize=20)
plt.plot([c7_x], [c7_y], '*', color='darkcyan', label='centroid7', markersize=20)
plt.plot([c8_x], [c8_y], '*', color='magenta', label='centroid8', markersize=20)
plt.plot([c9_x], [c9_y], '*', color='brown', label='centroid9', markersize=20)

plt.legend()
plt.show()
```

Label Matrix

```
# label each pixels.
labelMat = np.empty((height, width), dtype=int)
```

#each entries represent a pixel

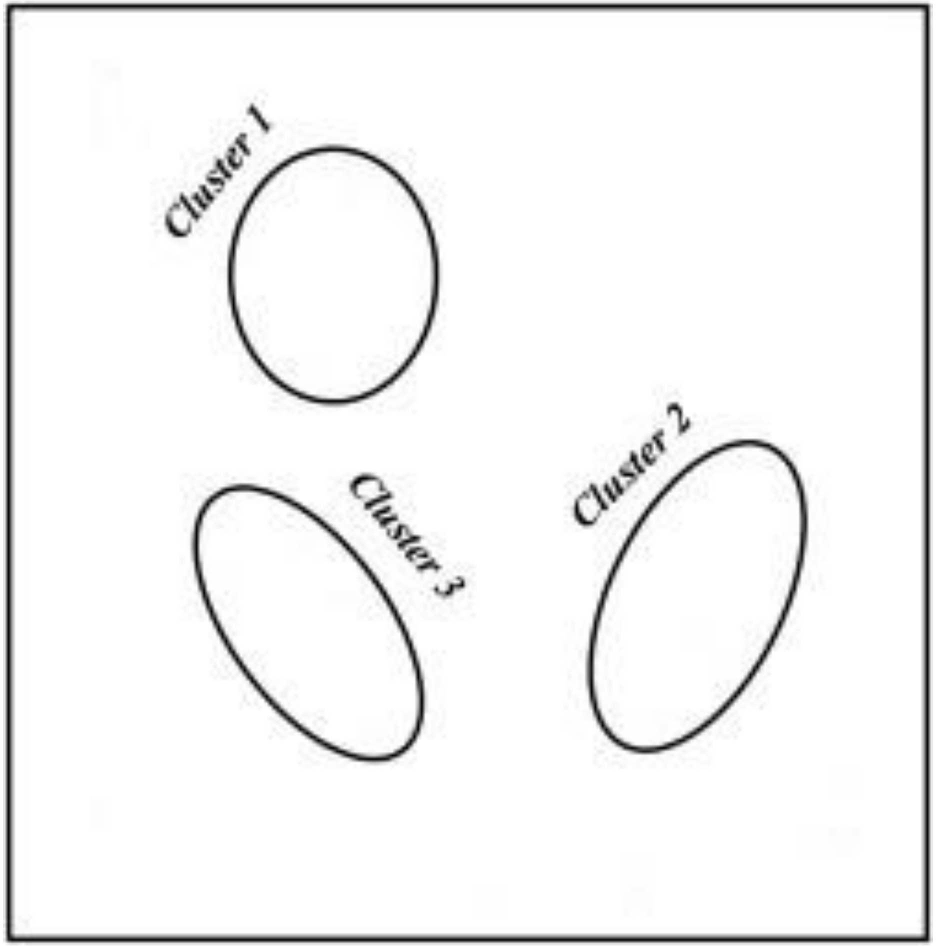
(height) x (width) dimension matrix

Label Matrix

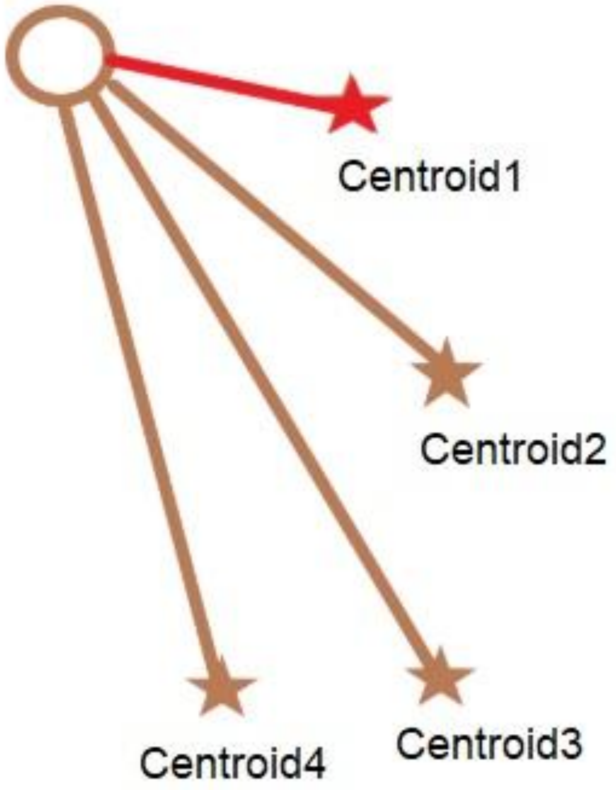
$$\begin{bmatrix} 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & & & 1 \\ \vdots & & & 3 & 3 \\ 8 & 8 & \cdots & 2 & 3 \end{bmatrix}$$

Empty Cluster 선언

```python
while True:
    itCount += 1

    # Initiate/reset cluster values.
    cluster1_x = np.zeros(0, dtype=np.float64)
    cluster1_y = np.zeros(0, dtype=np.float64)
    cluster1_rgb = np.zeros((0, 3), dtype=np.float64)

    cluster2_x = np.zeros(0, dtype=np.float64)
    cluster2_y = np.zeros(0, dtype=np.float64)
    cluster2_rgb = np.zeros((0,3), dtype=np.float64)

    cluster3_x = np.zeros(0, dtype=np.float64)
    cluster3_y = np.zeros(0, dtype=np.float64)
    cluster3_rgb = np.zeros((0,3), dtype=np.float64)

    cluster4_x = np.zeros(0, dtype=np.float64)
    cluster4_y = np.zeros(0, dtype=np.float64)
    cluster4_rgb = np.zeros((0,3), dtype=np.float64)

    cluster5_x = np.zeros(0, dtype=np.float64)
    cluster5_y = np.zeros(0, dtype=np.float64)
    cluster5_rgb = np.zeros((0,3), dtype=np.float64)

    cluster6_x = np.zeros(0, dtype=np.float64)
    cluster6_y = np.zeros(0, dtype=np.float64)
    cluster6_rgb = np.zeros((0, 3), dtype=np.float64)

    cluster7_x = np.zeros(0, dtype=np.float64)
    cluster7_y = np.zeros(0, dtype=np.float64)
    cluster7_rgb = np.zeros((0,3), dtype=np.float64)
```



```python
for y in range(height):
    for x in range(width):
        # compute distances from each centroids.
        d1 = distance(c1_rgb, img_normalized[y][x], c1_x, x, c1_y, y, lbd)
        d2 = distance(c2_rgb, img_normalized[y][x], c2_x, x, c2_y, y, lbd)
        d3 = distance(c3_rgb, img_normalized[y][x], c3_x, x, c3_y, y, lbd)
        d4 = distance(c4_rgb, img_normalized[y][x], c4_x, x, c4_y, y, lbd)
        d5 = distance(c5_rgb, img_normalized[y][x], c5_x, x, c5_y, y, lbd)
        d6 = distance(c6_rgb, img_normalized[y][x], c6_x, x, c6_y, y, lbd)
        d7 = distance(c7_rgb, img_normalized[y][x], c7_x, x, c7_y, y, lbd)
        d8 = distance(c8_rgb, img_normalized[y][x], c8_x, x, c8_y, y, lbd)
        d9 = distance(c9_rgb, img_normalized[y][x], c9_x, x, c9_y, y, lbd)

        d_list = np.array([d1, d2, d3, d4, d5, d6, d7, d8, d9])

        # allocate pixels to the minimum cluster.
        argMin = d_list.argmin()

        if argMin == 0:
            cluster1_x = np.append(cluster1_x, x)
            cluster1_y = np.append(cluster1_y, y)
            cluster1_rgb = np.append(cluster1_rgb, [img_normalized[y][x]], axis=0)

            labelMat[y][x] = 0

        elif argMin == 1:
            cluster2_x = np.append(cluster2_x, x)
            cluster2_y = np.append(cluster2_y, y)
            cluster2_rgb= np.append(cluster2_rgb, [img_normalized[y][x]], axis=0)

            labelMat[y][x] = 1

        elif argMin == 2:
            cluster3_x = np.append(cluster3_x, x)
            cluster3_y = np.append(cluster3_y, y)
            cluster3_rgb= np.append(cluster3_rgb, [img_normalized[y][x]], axis=0)
```



Distance Function

RGB값의 차이

위치의 차이

$$d = \|f(z) - m\|^2 + \lambda * \|z - c\|^2$$

where z denotes the spatial index, $f(z)$ denotes the image value (r, g, b) at the spatial location z, m denotes the centroid of image intensity, c denotes the centroid of spatial location, and λ determines the importance between the image intensity and the spatial relation.

- z = (x 좌표, y 좌표): 해당 픽셀의 위치.
- f(z) = (r, g, b): 해당 위치의 이미지 값.

Lambda ↑ - 위치의 비중 ↑

Lambda ↓ - 색 (r,g,b) 의 비중 ↑

```python
# Re-define centroids & print.
print('Iteration Number:', itCount)

new_c1_x = np.mean(cluster1_x)
new_c1_y = np.mean(cluster1_y)
new_c1_rgb = np.mean(cluster1_rgb, axis=0)
print('c1', new_c1_x, new_c1_y, new_c1_rgb)

new_c2_x = np.mean(cluster2_x)
new_c2_y = np.mean(cluster2_y)
new_c2_rgb = np.mean(cluster2_rgb, axis=0)
print('c2', new_c2_x, new_c2_y, new_c2_rgb)

new_c3_x = np.mean(cluster3_x)
new_c3_y = np.mean(cluster3_y)
new_c3_rgb = np.mean(cluster3_rgb, axis=0)
print('c3', new_c3_x, new_c3_y, new_c3_rgb)

new_c4_x = np.mean(cluster4_x)
new_c4_y = np.mean(cluster4_y)
new_c4_rgb = np.mean(cluster4_rgb, axis=0)
print('c4', new_c4_x, new_c4_y, new_c4_rgb)

new_c5_x = np.mean(cluster5_x)
new_c5_y = np.mean(cluster5_y)
new_c5_rgb = np.mean(cluster5_rgb, axis=0)
print('c5', new_c5_x, new_c5_y, new_c5_rgb)

new_c6_x = np.mean(cluster6_x)
new_c6_y = np.mean(cluster6_y)
new_c6_rgb = np.mean(cluster6_rgb, axis=0)
print('c6', new_c6_x, new_c6_y, new_c6_rgb)
```

```
Iteration Number: 38
c1 260.66493916178456 27.064218116268588 [0.11572074 0.38162129 0.48457882]
c2 107.84148727984345 125.49706457925636 [0.32600757 0.34680493 0.33761623]
c3 181.10194805194806 125.75113636363636 [0.27217532 0.30544945 0.25993952]
c4 35.5 125.5 [0.15067694 0.40733803 0.36034405]
c5 255.7118501368133 83.74047568932856 [0.19593563 0.3050982  0.27842394]
c6 182.81551952349437 40.88749172733289 [0.2305628  0.42375391 0.36053386]
c7 110.15813498532768 41.516628627323115 [0.29307492 0.44594002 0.38474079]
c8 259.9481911567664 140.14537740062528 [0.21915071 0.41335855 0.10161749]
c9 36.494127111826224 41.49332260659694 [0.20010096 0.3724468  0.36103229]

Iteration Number: 39
c1 260.6587784539103 27.057245886860493 [0.11568406 0.38161361 0.48466505]
c2 107.84148727984345 125.49706457925636 [0.32600757 0.34680493 0.33761623]
c3 181.10194805194806 125.75113636363636 [0.27217532 0.30544945 0.25993952]
c4 35.5 125.5 [0.15067694 0.40733803 0.36034405]
c5 255.71837507893076 83.72910966112397 [0.19586135 0.30509573 0.27842394]
c6 182.81551952349437 40.88749172733289 [0.2305628  0.42375391 0.36053386]
c7 110.15813498532768 41.516628627323115 [0.29307492 0.44594002 0.38474079]
c8 259.94753293145794 140.1390935476669 [0.21924274 0.4133617  0.10161757]
c9 36.494127111826224 41.49332260659694 [0.20010096 0.3724468  0.36103229]

Iteration Number: 40
c1 260.6587784539103 27.057245886860493 [0.11568406 0.38161361 0.48466505]
c2 107.84148727984345 125.49706457925636 [0.32600757 0.34680493 0.33761623]
c3 181.10194805194806 125.75113636363636 [0.27217532 0.30544945 0.25993952]
c4 35.5 125.5 [0.15067694 0.40733803 0.36034405]
c5 255.71837507893076 83.72910966112397 [0.19586135 0.30509573 0.27842394]
c6 182.81551952349437 40.88749172733289 [0.2305628  0.42375391 0.36053386]
c7 110.15813498532768 41.516628627323115 [0.29307492 0.44594002 0.38474079]
c8 259.94753293145794 140.1390935476669 [0.21924274 0.4133617  0.10161757]
c9 36.494127111826224 41.49332260659694 [0.20010096 0.3724468  0.36103229]
```

```python
# end the loop if centroids converge, continue clustering if not.
if centCheck(new_c1_x, new_c1_y, new_c1_rgb, c1_x, c1_y, c1_rgb) and centCheck(new_c2_x, new_c2_y, new_c2_rgb, c2_x, c2_y, c2_rgb) and
    break
else:

    c1_x = new_c1_x
    c1_y = new_c1_y
    c1_rgb = copy(new_c1_rgb)

    c2_x = new_c2_x
    c2_y = new_c2_y
    c2_rgb = copy(new_c2_rgb)

    c3_x = new_c3_x
    c3_y = new_c3_y
    c3_rgb = copy(new_c3_rgb)

    c4_x = new_c4_x
    c4_y = new_c4_y
    c4_rgb = copy(new_c4_rgb)

    c5_x = new_c5_x
    c5_y = new_c5_y
    c5_rgb = copy(new_c5_rgb)

    c6_x = new_c6_x
    c6_y = new_c6_y
    c6_rgb = copy(new_c6_rgb)

    c7_x = new_c7_x
    c7_y = new_c7_y
    c7_rgb = copy(new_c7_rgb)
```

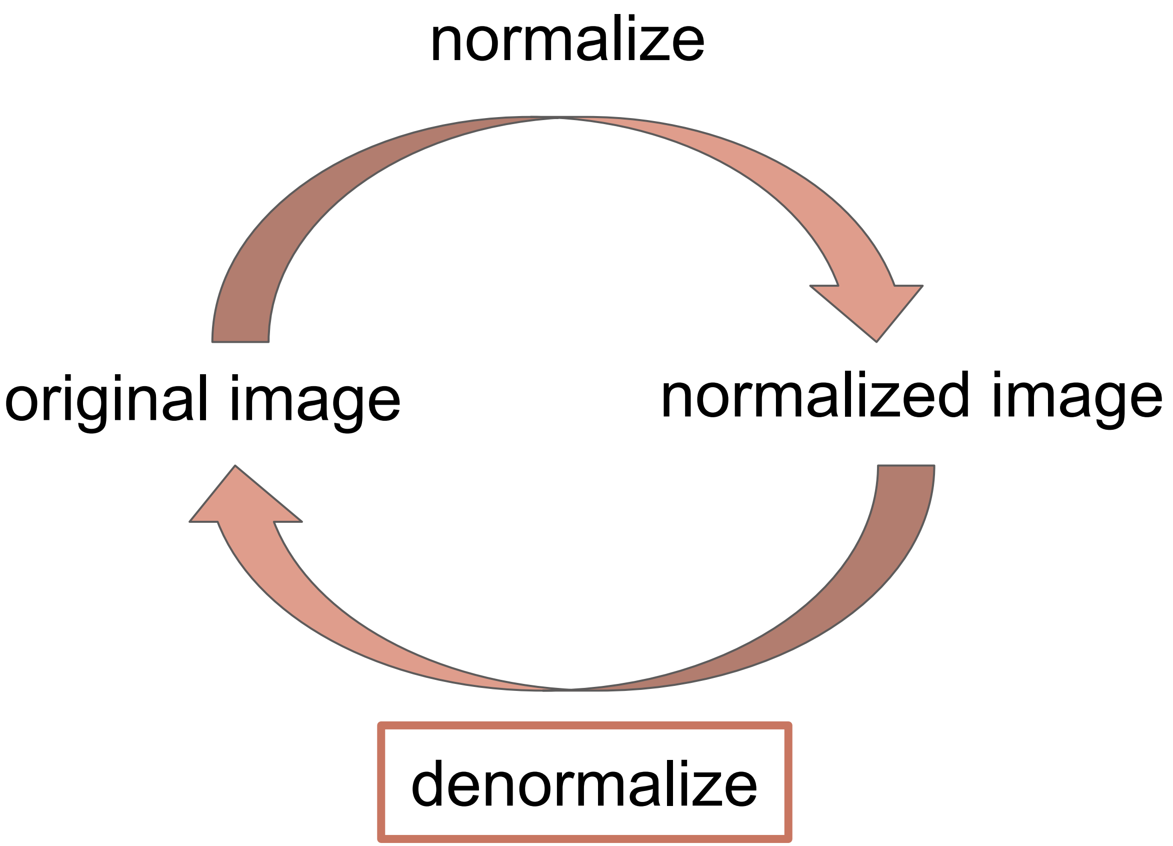
```python
# To check if values of centroids are same.
def centCheck(cx, cy, crgb, x, y, rgb):
    if cx == x and cy == y and (crgb ==rgb).all():
        return True
    return False
```

```
Iteration Number: 39
c1 260.6587784539103 27.057245886860493 [0.11568406 0.38161361 0.48466505]
c2 107.84148727984345 125.49706457925636 [0.32600757 0.34680493 0.33761623]
c3 181.10194805194806 125.75113636363636 [0.27217532 0.30544945 0.25993952]
c4 35.5 125.5 [0.15067694 0.40733803 0.36034405]
c5 255.71837507893076 83.72910966112397 [0.19586135 0.30509573 0.27842394]
c6 182.81551952349437 40.88749172733289 [0.2305628  0.42375391 0.36053386]
c7 110.15813498532768 41.516628627323115 [0.29307492 0.44594002 0.38474079]
c8 259.94753293145794 140.1390935476669 [0.21924274 0.4133617  0.10161757]
c9 36.494127111826224 41.49332260659694 [0.20010096 0.3724468  0.36103229]
```

```
Iteration Number: 40
c1 260.6587784539103 27.057245886860493 [0.11568406 0.38161361 0.48466505]
c2 107.84148727984345 125.49706457925636 [0.32600757 0.34680493 0.33761623]
c3 181.10194805194806 125.75113636363636 [0.27217532 0.30544945 0.25993952]
c4 35.5 125.5 [0.15067694 0.40733803 0.36034405]
c5 255.71837507893076 83.72910966112397 [0.19586135 0.30509573 0.27842394]
c6 182.81551952349437 40.88749172733289 [0.2305628  0.42375391 0.36053386]
c7 110.15813498532768 41.516628627323115 [0.29307492 0.44594002 0.38474079]
c8 259.94753293145794 140.1390935476669 [0.21924274 0.4133617  0.10161757]
c9 36.494127111826224 41.49332260659694 [0.20010096 0.3724468  0.36103229]
```

Image Denormalize

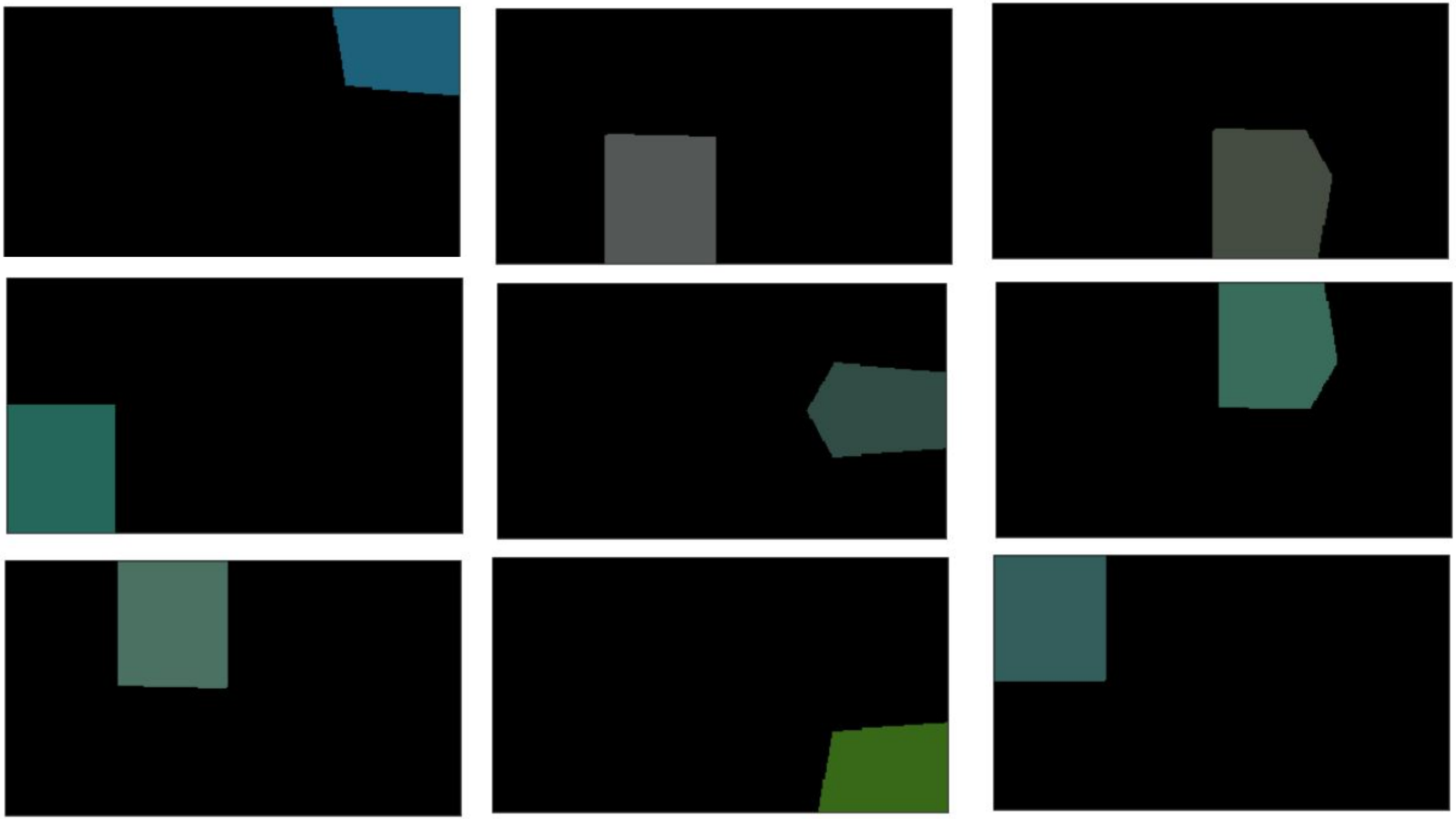
```
c1_rgb_denorm = np.empty((3), dtype=int)
c2_rgb_denorm = np.empty((3), dtype=int)
c3_rgb_denorm = np.empty((3), dtype=int)
c4_rgb_denorm = np.empty((3), dtype=int)
c5_rgb_denorm = np.empty((3), dtype=int)
c6_rgb_denorm = np.empty((3), dtype=int)
c7_rgb_denorm = np.empty((3), dtype=int)
c8_rgb_denorm = np.empty((3), dtype=int)
c9_rgb_denorm = np.empty((3), dtype=int)

# denormalize image values.
c1_rgb_denorm = img_denormalize(new_c1_rgb)
c2_rgb_denorm = img_denormalize(new_c2_rgb)
c3_rgb_denorm = img_denormalize(new_c3_rgb)
c4_rgb_denorm = img_denormalize(new_c4_rgb)
c5_rgb_denorm = img_denormalize(new_c5_rgb)
c6_rgb_denorm = img_denormalize(new_c6_rgb)
c7_rgb_denorm = img_denormalize(new_c7_rgb)
c8_rgb_denorm = img_denormalize(new_c8_rgb)
c9_rgb_denorm = img_denormalize(new_c9_rgb)
```



하나의 Cluster 출력

```
# plot each clusters.
new_image = np.zeros((height, width, 3), dtype=int)

for y in range(height):
    for x in range(width):

        if labelMat[y][x] == 0:
            new_image[y][x] = c1_rgb_denorm
        else:
            new_image[y][x] = [0,0,0]

plt.imshow(new_image)
plt.xticks([])
plt.yticks([])
plt.show()
```

# set lambda value.
lbd = 0.1

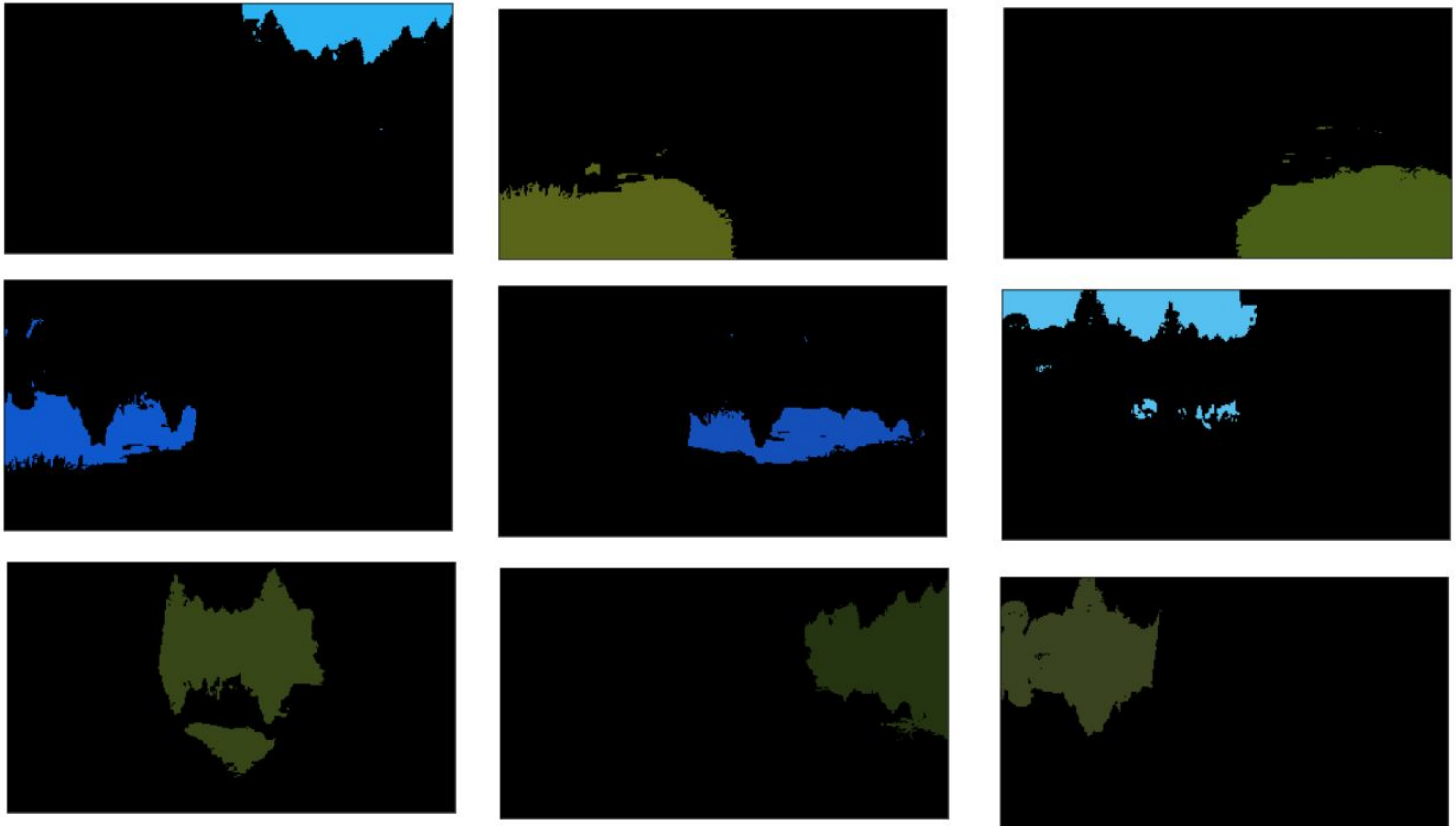


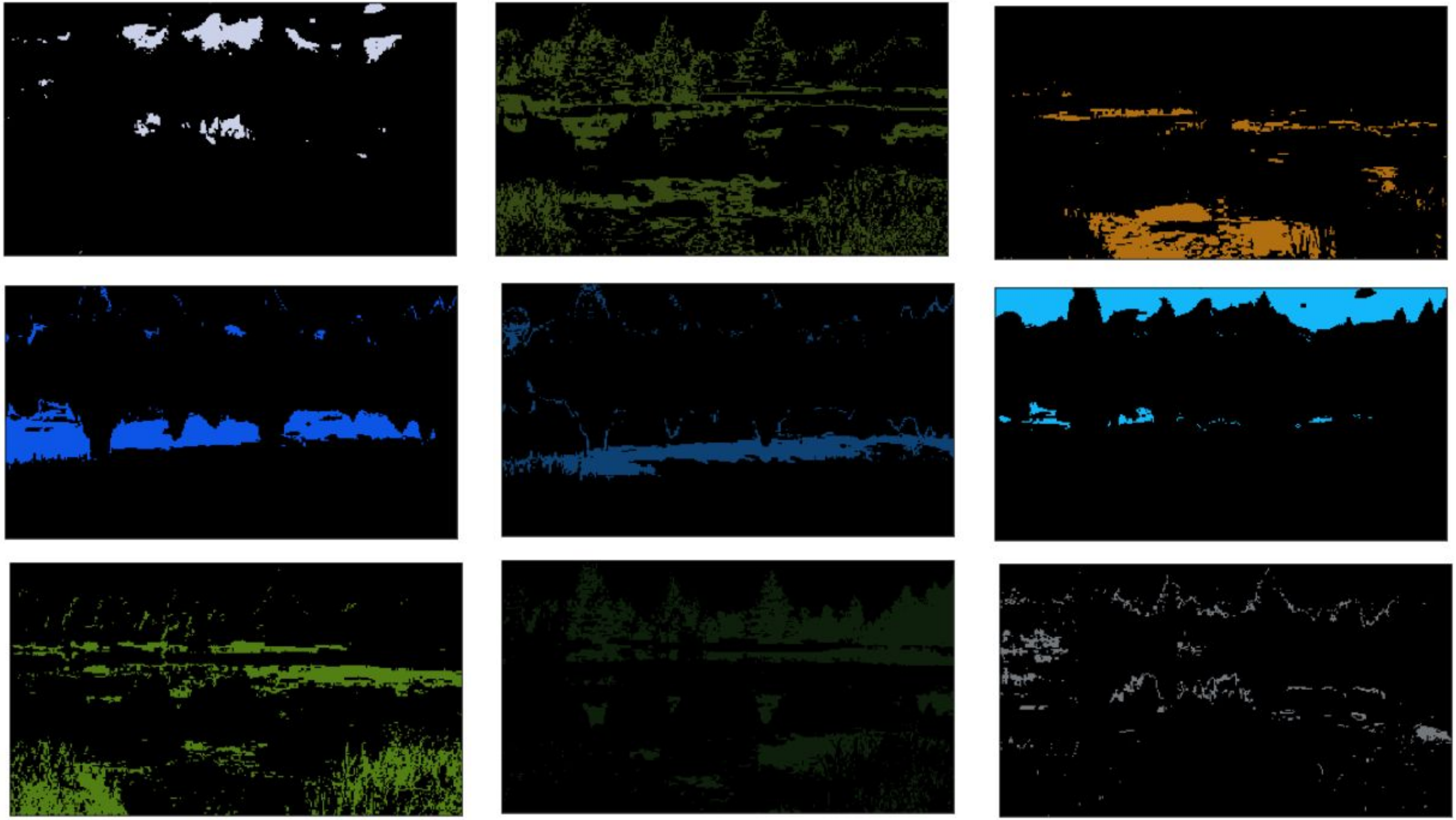
전체 이미지 출력

```
new_image = np.empty((height, width, 3), dtype=int)

for y in range(height):
    for x in range(width):

        if labelMat[y][x] == 0:
            new_image[y][x] = c1_rgb_denorm
        elif labelMat[y][x] == 1:
            new_image[y][x] = c2_rgb_denorm
        elif labelMat[y][x] == 2:
            new_image[y][x] = c3_rgb_denorm
        elif labelMat[y][x] == 3:
            new_image[y][x] = c4_rgb_denorm
        elif labelMat[y][x] == 4:
            new_image[y][x] = c5_rgb_denorm
        elif labelMat[y][x] == 5:
            new_image[y][x] = c6_rgb_denorm
        elif labelMat[y][x] == 6:
            new_image[y][x] = c7_rgb_denorm
        elif labelMat[y][x] == 7:
            new_image[y][x] = c8_rgb_denorm
        elif labelMat[y][x] == 8:
            new_image[y][x] = c9_rgb_denorm
```

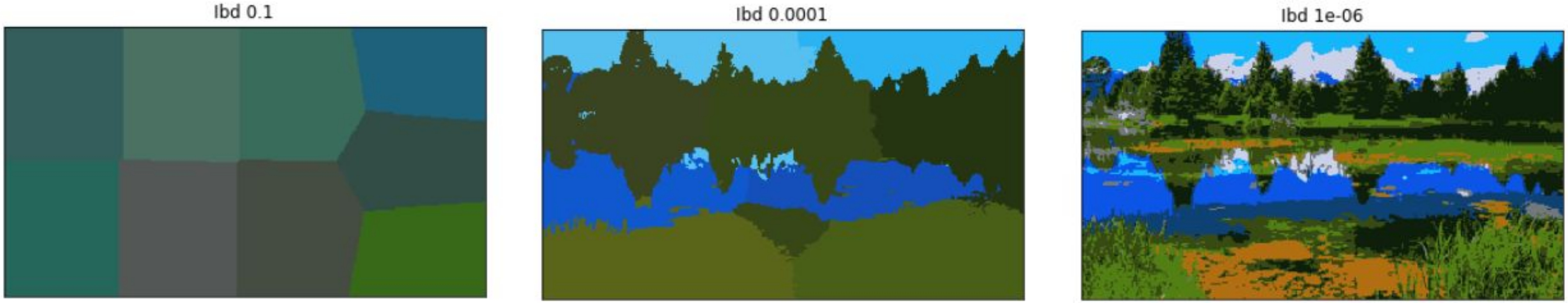
```
# Whole Image
plt.imshow(new_image)
plt.xticks([])
plt.yticks([])
plt.title('lbd {}'.format(lbd))
plt.show()
```

lbd 0.1



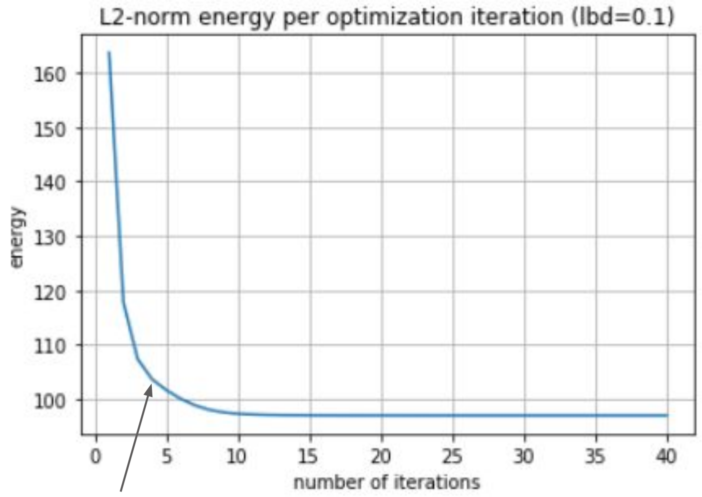


lambda 값에 따른 변화



Energy/Cost Function

```
# calculate energy
for y in range(height):
    for x in range(width):
        if labelMat[y][x] == 0:
            energy += distance(new_c1_rgb, img_normalized[y][x], new_c1_x, x, new_c1_y, y, lbd)
        elif labelMat[y][x] == 1:
            energy += distance(new_c2_rgb, img_normalized[y][x], new_c2_x, x, new_c2_y, y, lbd)
        elif labelMat[y][x] == 2:
            energy += distance(new_c3_rgb, img_normalized[y][x], new_c3_x, x, new_c3_y, y, lbd)
        elif labelMat[y][x] == 3:
            energy += distance(new_c4_rgb, img_normalized[y][x], new_c4_x, x, new_c4_y, y, lbd)
        elif labelMat[y][x] == 4:
            energy += distance(new_c5_rgb, img_normalized[y][x], new_c5_x, x, new_c5_y, y, lbd)
        elif labelMat[y][x] == 5:
            energy += distance(new_c6_rgb, img_normalized[y][x], new_c6_x, x, new_c6_y, y, lbd)
        elif labelMat[y][x] == 6:
            energy += distance(new_c7_rgb, img_normalized[y][x], new_c7_x, x, new_c7_y, y, lbd)
        elif labelMat[y][x] == 7:
            energy += distance(new_c8_rgb, img_normalized[y][x], new_c8_x, x, new_c8_y, y, lbd)
        elif labelMat[y][x] == 8:
            energy += distance(new_c9_rgb, img_normalized[y][x], new_c9_x, x, new_c9_y, y, lbd)

energy /= (height*width)

x_num.append(itCount)
y_energy.append(energy)
```



Formula

The energy/cost function is given by:

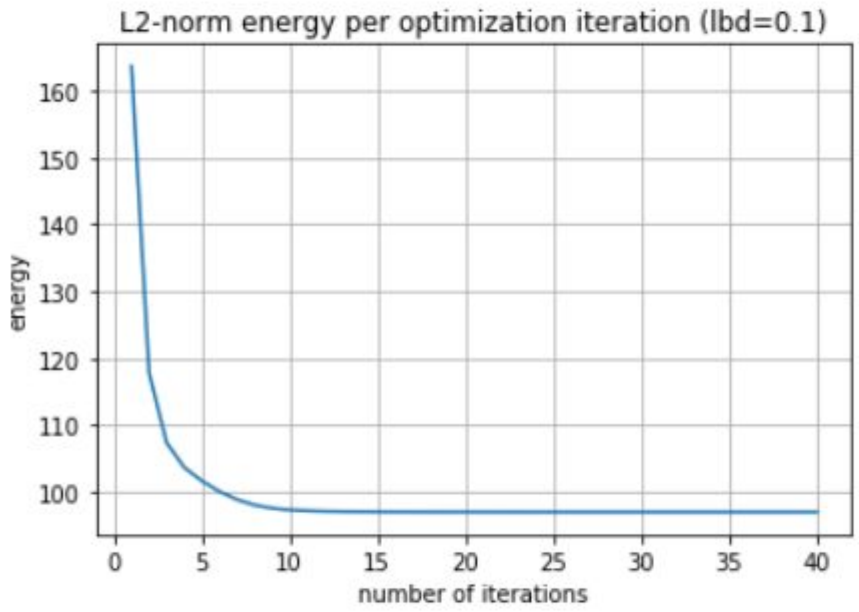
$$\frac{1}{n}\sum_k \sum_{z\in I(k)} \|f(z) - m_k\|^2 + \lambda * \|z - c_k\|^2$$

where $I(k)$ denotes the index set of z that belongs to cluster k, $f(z)$ denotes the image value (r, g, b) at the spatial location z, m_k denotes the centroid of image intensity for cluster k, c_k denotes the centroid of spatial location for cluster k, n denotes total number of pixels, and λ determines the importance between the image intensity and the spatial relation.

- N: 픽셀의 개수/이미지 크기 (height x width)
- M(k): cluster k의 이미지 centroid
- C(k): cluster k의 위치 centroid

Energy/Cost Graph Plot

```
fig, ax = plt.subplots()
ax.plot(x_num, y_energy)
ax.set(xlabel='number of iterations', ylabel='energy',
title='L2-norm energy per optimization iteration (lbd={})'.format(lbd))
ax.grid()
plt.show()
```



Part 4

Applications

SNS 데이터를 활용하여 고객의 마음 읽기

<가정>

SNS 상에서 두 개의 브랜드가 함께 언급되는 빈도가 많을 수록, 소비자들이 두 브랜드를 더 유사하게 인식할 것이다.

출처 : LG CNS

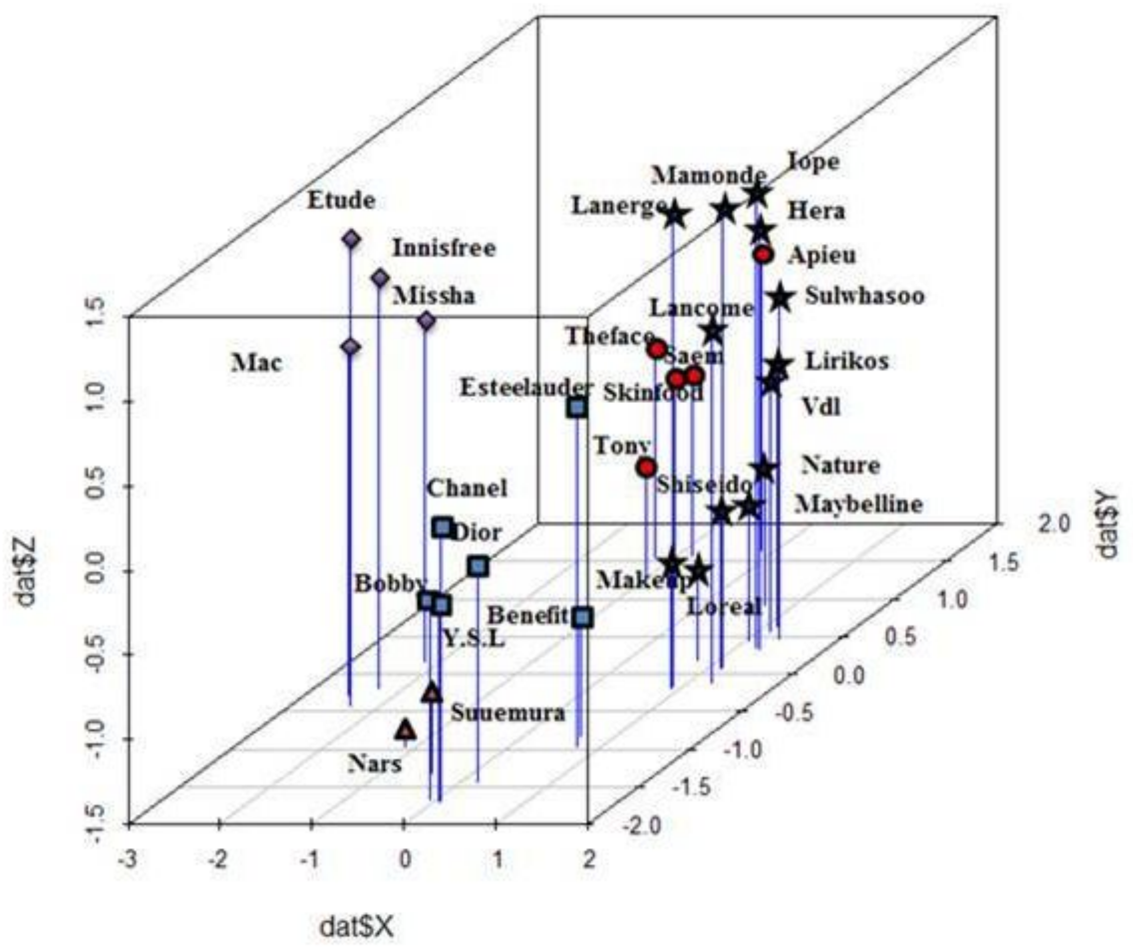
SNS와 화장품 브랜드

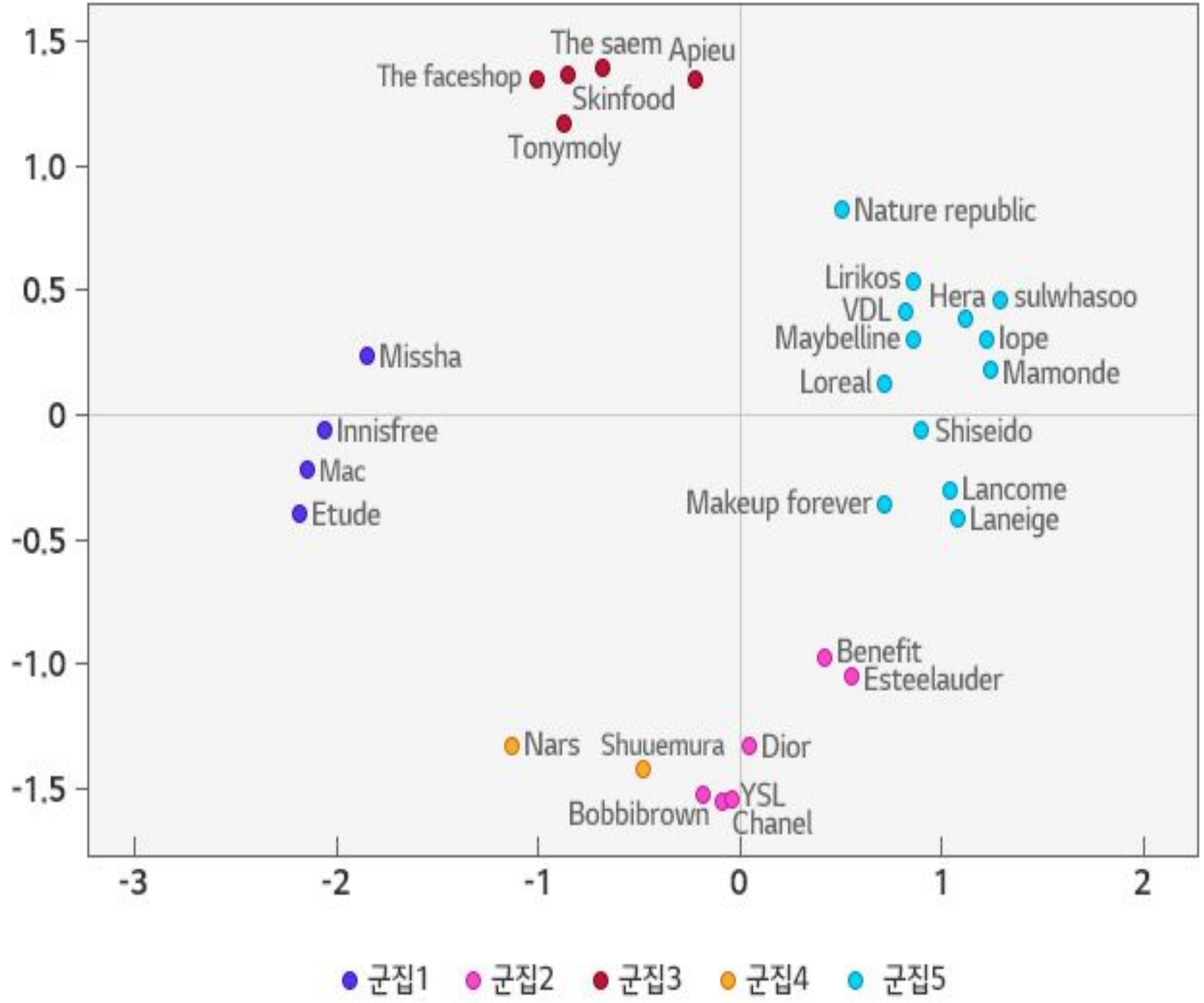
Brand Names	Innisfree	Missha	Etude	The faceshop	Mac	···	Nars
Innisfree	-	406	1137	391	195	···	-
Missha	105	-	836	247	185	···	-
Etude	1137	837	-	630	499	···	323
The faceshop	391	246	631	-	116	···	-
Mac	0	0	369	0	-	···	608
···	···	···	···	···	···	···	···
Nars	89	97	244	-	607	···	-

I [표1] Co-memtion Matrix

출처 : LG CNS

SNS와 화장품 브랜드





출처 : LG CNS

Thank You

감사합니다