20145271 Geum Kang Hyeon

K-means algorithm on color image

Let $f(x)$ be a color image and x be the index of image in the domain. The values of image $f(x)$ consist of [red, green, blue] intensity.

Apply K-means algorithm to image $f(x)$ based on its color value with given number of clusters K and visualize the progress of optimization and results of the algorithm for each selected number of clusters K.

1. Select any color image that consists of distinctive regions with different colors.
2. Apply K-means algorithm to the given image with at least 4 different choice of K.
3. For each K, plot the energy curve and the result image.

[Energy]

$$\frac{1}{n}\sum_{x\in\Omega}\|f(x)-m_c\|^2,$$

where Ω denotes the image domain and the number of pixels $|\Omega|$ is n, and m_c denotes the centroid for cluster c that is the cluster label of $f(x)$.

[Output Image]

$g(x)=m_c$ where $label(x)=c$

Each pixel of the output image $g(x)$ should be its centroid m_c where c is the cluster label of $g(x)$.

Code Explaination

The process consists of 7 main stages.

1. After reading the Image, assign a random label to each pixels
2. Find a centroid vector from randomly labeled pixels.
3. The distance is expressed by the L2 Norm between the actual pixel value and the centeroid vector.
4. Update labeling with the value of argmin in distance.
5. Calculate energy according to the expression provided above.
6. Repeat until the difference between the previous and current energies is less than 1 (about 0.0001%).
7. Display Images in each K and its color is centroid vector's RGB color.

In [234]:

```
import matplotlib.pyplot as plt
import numpy as np
import math, random
from PIL import Image
np.seterr(divide='ignore', invalid='ignore')

def L2_distance(source, target):
    sum = sum_of_square(source, target)
    return math.sqrt(sum)

def sum_of_square(source, target):
    sum = 0
    sum += (source[:] - target[:]) ** 2
    return np.sum(sum)


def plot(k, random_label, centroid_vector, img, energy_data):
    for idx in range(k):

        pixelMap = img.load()

        temp_img = Image.new(img.mode, img.size)
        pixelsNew = temp_img.load()

        for i in range(temp_img.size[0]):
            for j in range(temp_img.size[1]):
                if random_label[j*temp_img.size[0] + i] == idx:
                    pixelsNew[i,j] = (int(centroid_vector[idx][0]), int(centroid_vector[idx][1
]), int(centroid_vector[idx][2]))
                else:
                    pixelsNew[i,j] = (255,255,255)

        print("Image of Cluster ", idx+1)
        display(temp_img)

    plt.title("Energy")
    plt.plot(energy_data)
    plt.show()


def clustering(filename, k):
    img = Image.open(filename, 'r')

    width, height = img.size
    img = img.convert("RGB")
    pixel_values = list(img.getdata())

    num_pixel = width * height

    print("Origin Image")
    display(img)

    random_label = np.empty(num_pixel, dtype=int)
    backup_label = np.empty(num_pixel, dtype=int)

    for idx in range(num_pixel):
        random_label[idx] = np.random.randint(0,k,1)
```

```
    energy = 0
    energy_data = []

    for loop in range(100):

        new_energy = 0

        centroid_vector = np.zeros((k, 3), dtype=float)
        centroid_vector_count = np.zeros(k, dtype=int)
        countIdx = np.zeros(k, dtype=int)

        # centroid image
        for i in range(num_pixel):
            centroid_vector[int(random_label[i])] += pixel_values[i]
            centroid_vector_count[int(random_label[i])] += 1

        for i in range(k):
            centroid_vector[i,:] /= centroid_vector_count[i]

        zero_check = False
        for i in range(k):
            if(centroid_vector_count[i] == 0):
                random_label = np.empty(num_pixel, dtype=int)
                for idx in range(num_pixel):
                    random_label[idx] = np.random.randint(0,k,1)
                zero_check = True
                break

        if(zero_check == False):
             # compare image
            for i in range(num_pixel):
                temp = pixel_values[i]
                distance = np.zeros(k, dtype=float)

                for index in range(k):
                    distance[index] = L2_distance(temp, centroid_vector[index])

                label = np.argmin(distance)
                new_energy += sum_of_square(temp, centroid_vector[label])

                backup_label[i] = label


            random_label = backup_label

            # calculate energy
            new_energy /= num_pixel
            energy_data.append(new_energy)

            if energy == 0:
                energy = new_energy
            elif abs(new_energy - energy) < 1:
                plot(k, random_label, centroid_vector, img, energy_data)
                break
            else:
                energy = new_energy
```

1) k is 3

In [235]:

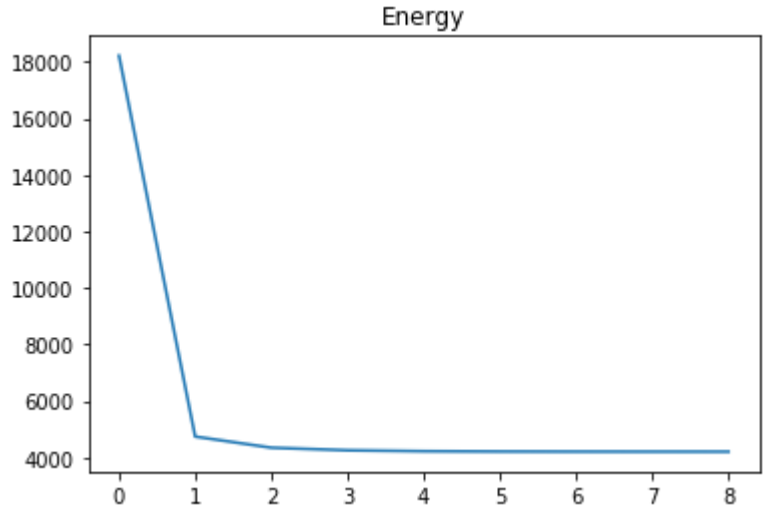
```
clustering('picasso.jpg', 3)
```

Origin Image

Image of Cluster 1

Image of Cluster 2

Image of Cluster 3


Energy
18000
16000
14000
12000
10000
8000
6000
4000
0
1
2
3
4
5
6
7
8

2) k is 5

In [236]:

```
clustering('picasso.jpg', 5)
```

Origin Image

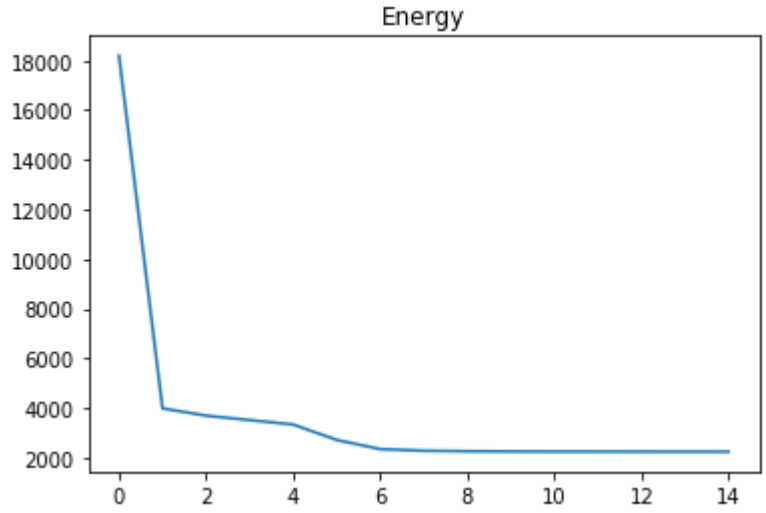
Image of Cluster 1

Image of Cluster 2

Image of Cluster 3

Image of Cluster 4

Image of Cluster 5


Energy
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
2
4
6
8
10
12
14

3) k is 7

In [237]:

```
clustering('picasso.jpg', 7)
```

Origin Image

Image of Cluster 1

Image of Cluster 2

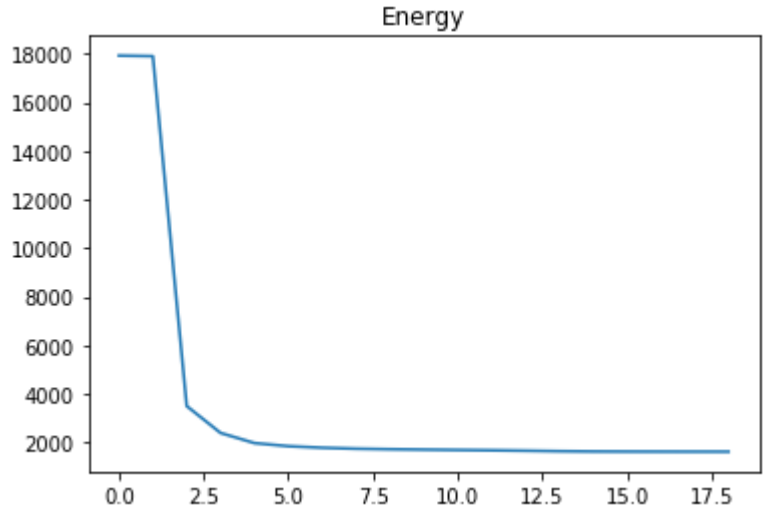
Image of Cluster 3

Image of Cluster 4

Image of Cluster 5

Image of Cluster 6

Image of Cluster 7


Energy
18000
16000
14000
12000
10000
8000
6000
4000
2000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

4) k is 9

In [238]:

```
clustering('picasso.jpg', 9)
```

Origin Image

Image of Cluster 1

Image of Cluster 2

Image of Cluster 3

Image of Cluster 4

Image of Cluster 5

Image of Cluster 6

Image of Cluster 7

Image of Cluster 8

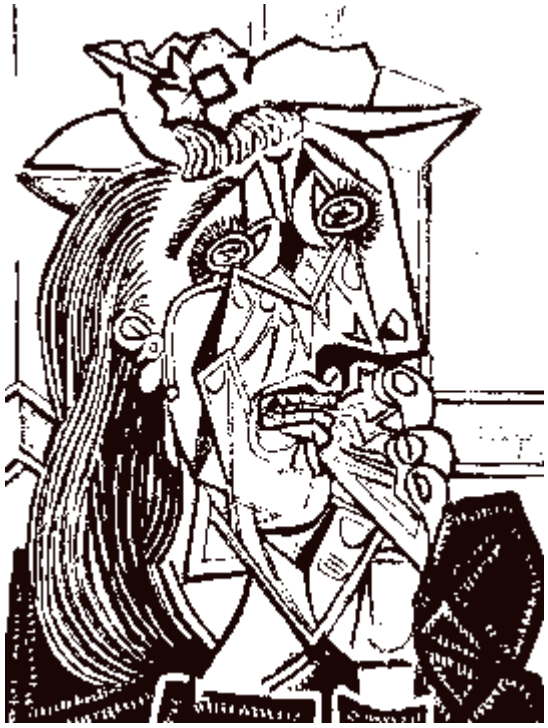
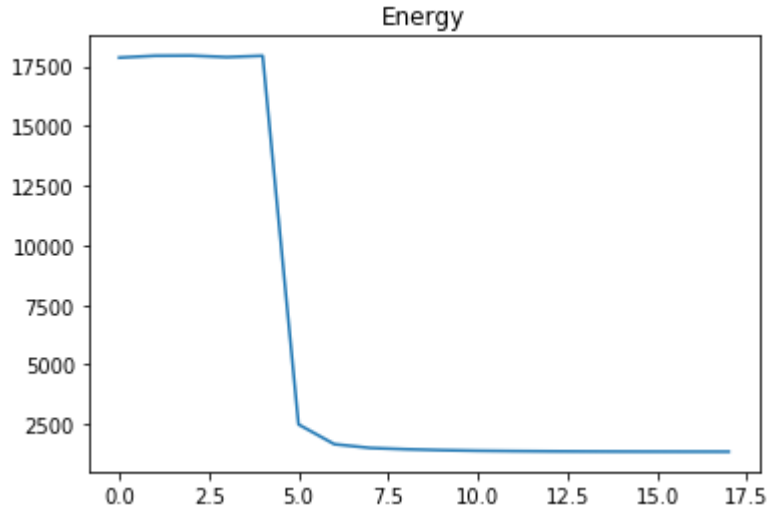


Image of Cluster 9


Energy
17500
15000
12500
10000
7500
5000
2500
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5