20145271 Geum Kang Hyeon

Apply K-means algorithm to both image value and its spatial domain

For a given input image (either gray or color), apply a K-means algorithm that is designed to take into consideration of both the image intensity and its spatial domain with varying parameters: the number of clusters and the trade-off between the intensity energy and the spatial energy.

[The objective function]

$$\sum_k \sum_{\{} x \in I(k)\}[\|f(x) - m_k\|^2 + a * \|x - c_k\|^2]$$

where I(k) denotes the index set of x that belongs to cluster k, m_k denotes the centroid of image intensity for cluster k, c_k denotes the centroid of spatial location for cluster k, and a determines the importance between the image intensity and the spatial relation.

[L2 Norm Energy]

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

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

Code Explaination

The process consists of 7 main stages.

1. Generate two lists with Pixel and Coordinate information, assign a random label to each pixels
2. Find a centroid pixel and coord from randomly labeled data.
3. The distance is expressed by the Objective Function of the sum of L2 Norm between the actual pixel, coordinate value and the centeroid pixel, coordinate.
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 0.005 (about 0.0005%).
7. Displays the image corresponding to each Alpha Value (0.1 ~ 1.6)

In [251]:

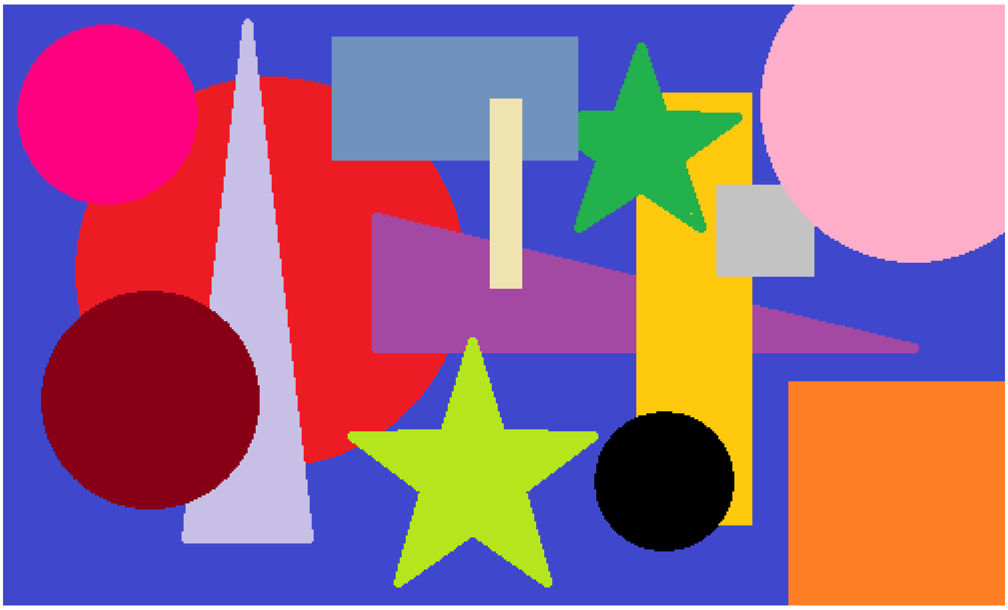
```
import matplotlib.pyplot as plt
import numpy as np
import math, random
import statistics
from PIL import Image
np.seterr(divide='ignore', invalid='ignore')

def norm(norm_type, source, target):
    if(norm_type == 0):
        return L1_distance(source, target)
    elif(norm_type == 1):
        return L2_distance(source, target)
    elif(norm_type == 2):
        return sum_of_square(source, target)

def L1_distance(source, target):
    sum = 0
    sum += abs(source[:] - target[:])
    return np.sum(sum)

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 Vectorization(list):
    x = []
    y = []

    for i in range(len(list)):
        x.append(list[i][0])
        y.append(list[i][1])
    return x,y


def plot(k, random_label, centroid_vector, img, energy_data, mean_pixel, stdev_pixel, mean_coord
, stdev_coord, alpha):

    for idx in range(k):
        for i in range(3):
            centroid_vector[idx][i] = centroid_vector[idx][i] * stdev_pixel[i] + mean_pixel[i]
        for i in range(2):
            centroid_vector[idx][i + 3] = centroid_vector[idx][i + 3] * stdev_coord[i] + mean_co
ord[i]

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

    for idx in range(k):
        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]))
```

```
    display(temp_img)

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

def plot_energy(energy_data, norm_type):
    if(norm_type == 0):
        plt.title("L1 Norm Energy")
    else:
        plt.title("L2 Norm Energy")
    plt.plot(energy_data)
    plt.show()


def normalize(data):
    dataLen = len(data)
    elemLen = len(data[0])
    result = np.zeros((dataLen, elemLen), dtype=float)
    mean, stdev = [], []

    for j in range(elemLen):
        temp = []
        for i in range(dataLen):
            temp.append(data[i][j])
        mean.append(np.mean(temp))
        stdev.append(np.std(temp))

        for i in range(dataLen):
            result[i][j] = (temp[i] - mean[j]) / stdev[j]

    return result, mean, stdev

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

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

    num_pixel = width * height

    coord = np.empty((num_pixel, 2), dtype=int)
    random_label = np.empty(num_pixel, dtype=int)
    backup_label = np.empty(num_pixel, dtype=int)

    idx = 0
    for i in range(height):
        for j in range(width):
            coord[idx][0] = i
            coord[idx][1] = j
            random_label[idx] = np.random.randint(0,k,1)
            idx += 1


    # normalize pixel, coord
    mean_pixel, stdev_pixel, mean_coord, stdev_coord = [], [], [], []
    pixel_values, mean_pixel, stdev_pixel = normalize(pixel_values)
    coord, mean_coord, stdev_coord = normalize(coord)
    energy = 0
    energy_data = []

    for loop in range(100):
        new_energy = 0

        centroid_vector = np.zeros((k, 5), dtype=float)
        centroid_vector_count = np.zeros(k, dtype=int)
        countIdx = np.zeros(k, dtype=int)

        # centroid
        for i in range(num_pixel):
            centroid_vector[int(random_label[i])][0] += pixel_values[i][0]
            centroid_vector[int(random_label[i])][1] += pixel_values[i][1]
            centroid_vector[int(random_label[i])][2] += pixel_values[i][2]
            centroid_vector[int(random_label[i])][3] += coord[i][0]
            centroid_vector[int(random_label[i])][4] += coord[i][1]
            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):
                break;

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

                for index in range(k):
                    distance[index] = norm(norm_type, temp_pixel, centroid_vector[index][0:3]) +
(alpha * norm(norm_type, temp_coord, centroid_vector[index][3:]))

                label = np.argmin(distance)

                new_energy += (norm(norm_type*2, temp_pixel, centroid_vector[label][0:3]) + (alp
ha * norm(norm_type*2, temp_coord, centroid_vector[label][3:])))

                backup_label[i] = label

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

            #print(energy, new_energy)

            if energy == 0:
                energy = new_energy
            elif abs(new_energy - energy) < 0.005:
                plot(k, random_label, centroid_vector, img, energy_data, mean_pixel, stdev_pixel
, mean_coord, stdev_coord, alpha)
                break
            else:
                energy = new_energy
```

Original Image

In [249]:

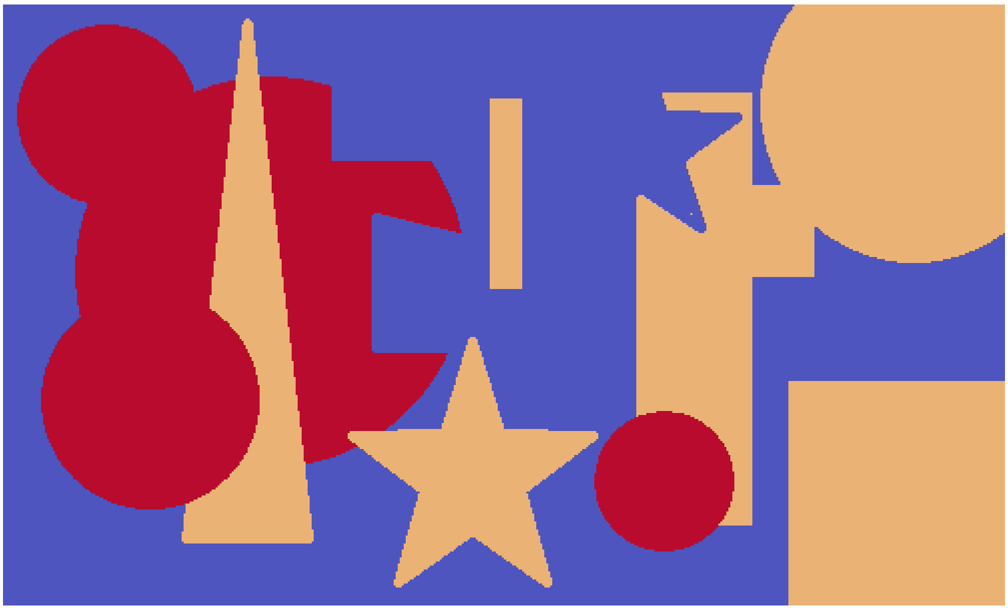
```
img = Image.open("sample_image.jpg", 'r')
display(img)
```

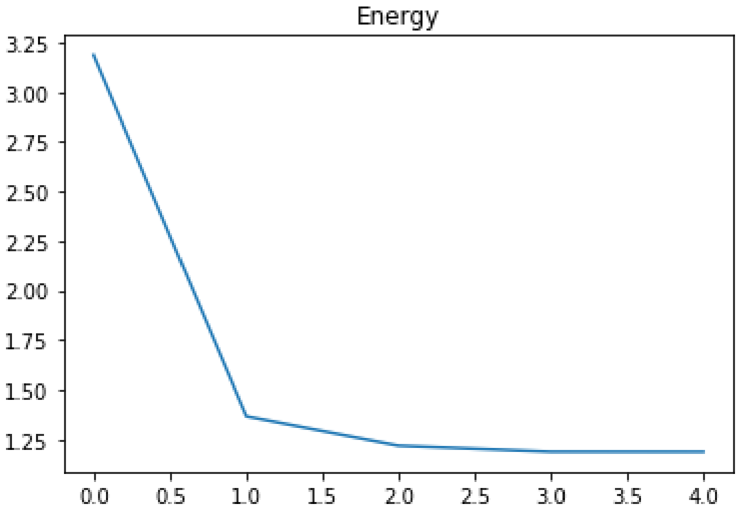


1. k = 3, Alpha = 0.1

In [207]:

```
clustering("sample_image.jpg", 3, 1, 0.1)
```

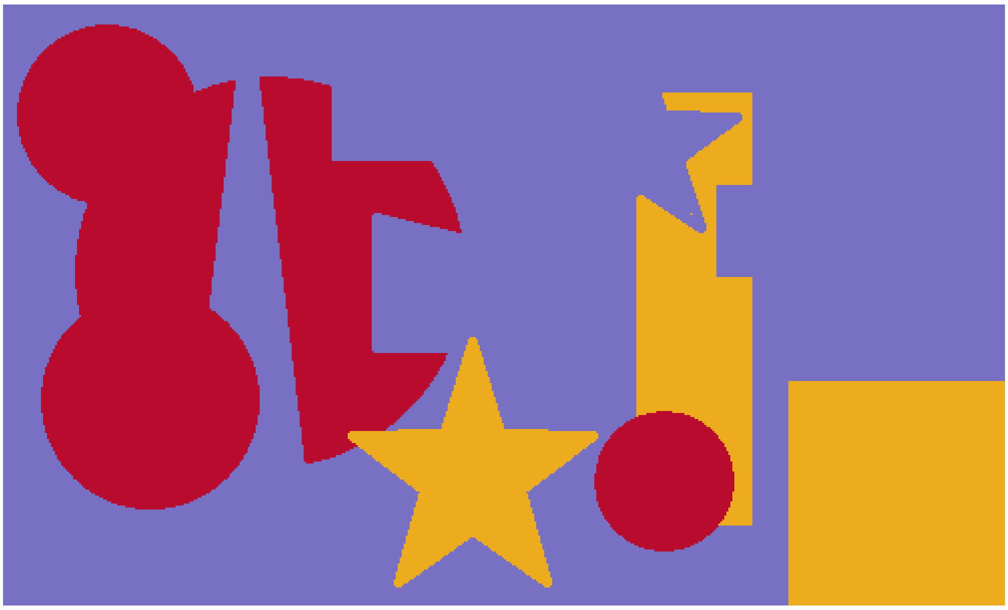


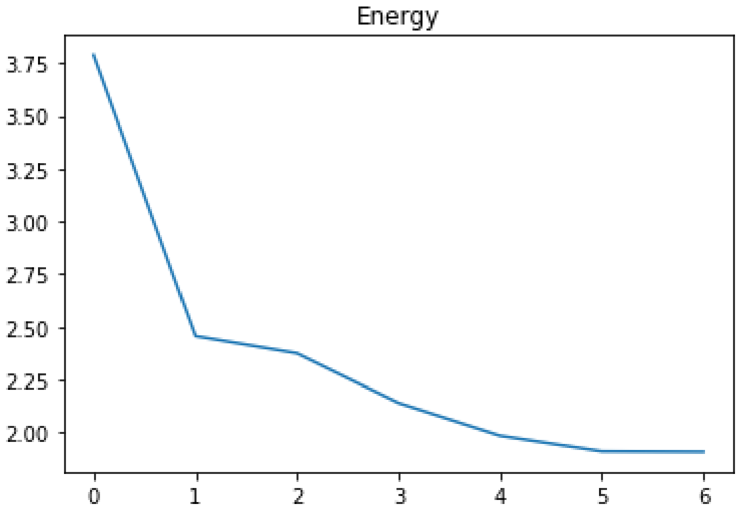


1. k = 3, Alpha = 0.4

In [208]:

```
clustering("sample_image.jpg", 3, 1, 0.4)
```

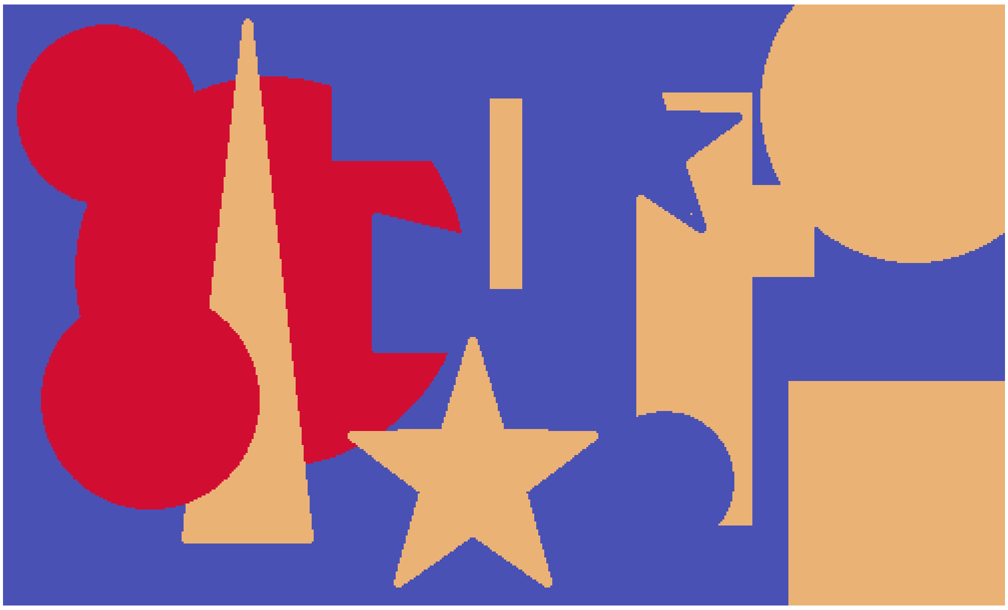


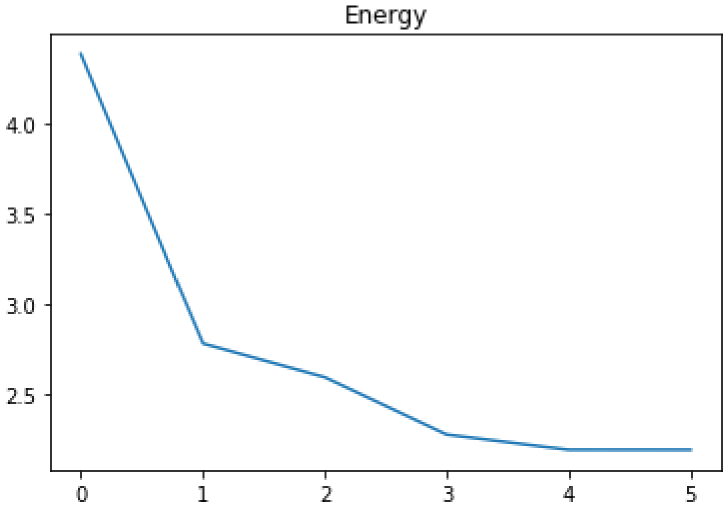


1. k = 3, Alpha = 0.7

In [209]:

```
clustering("sample_image.jpg", 3, 1, 0.7)
```

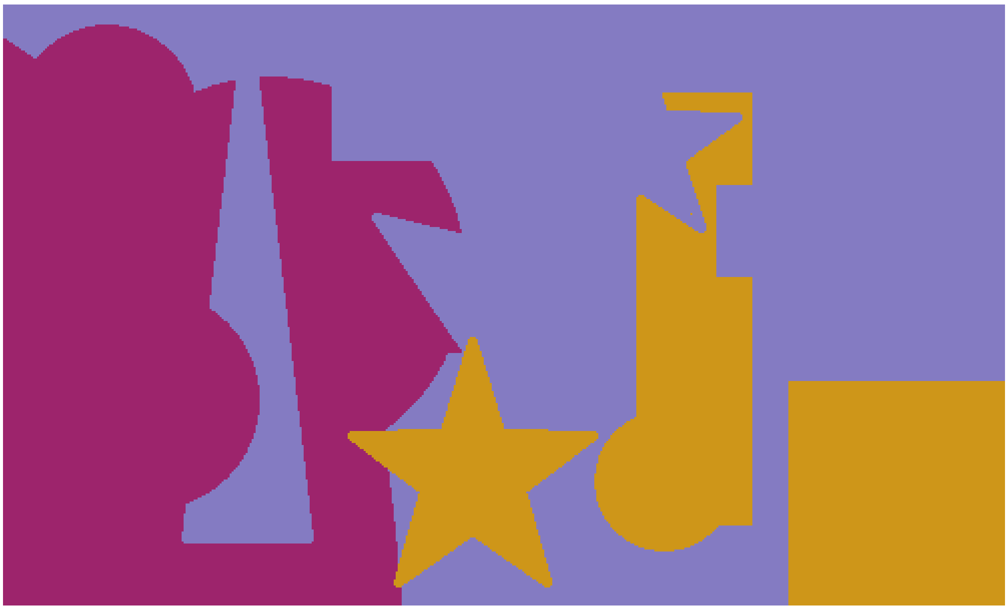


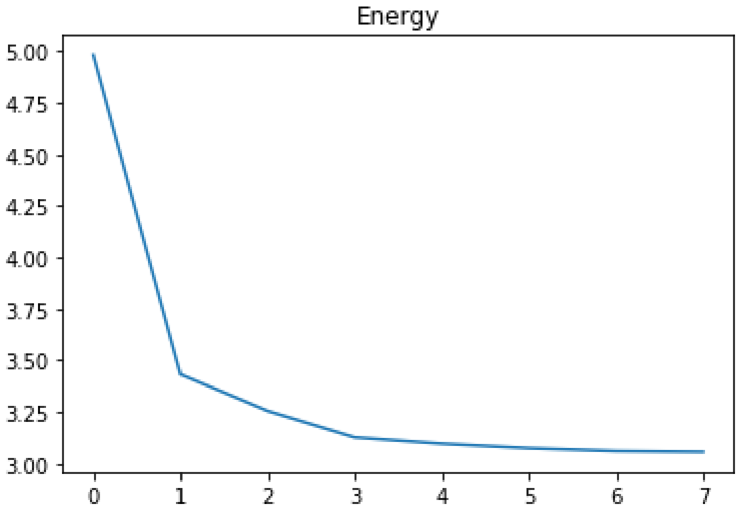


1. k = 3, Alpha = 1.0

In [210]:

```
clustering("sample_image.jpg", 3, 1, 1.0)
```

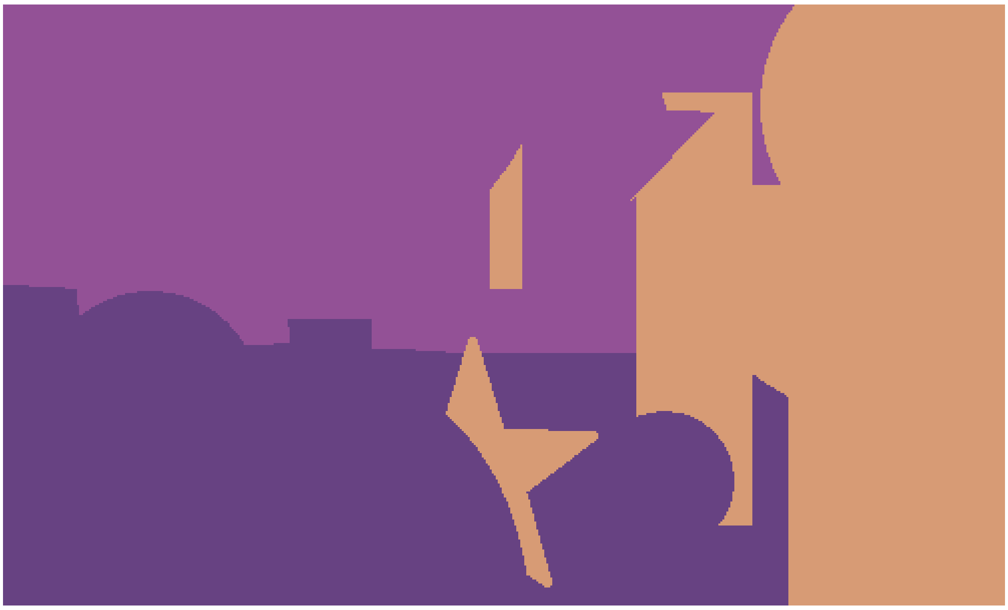


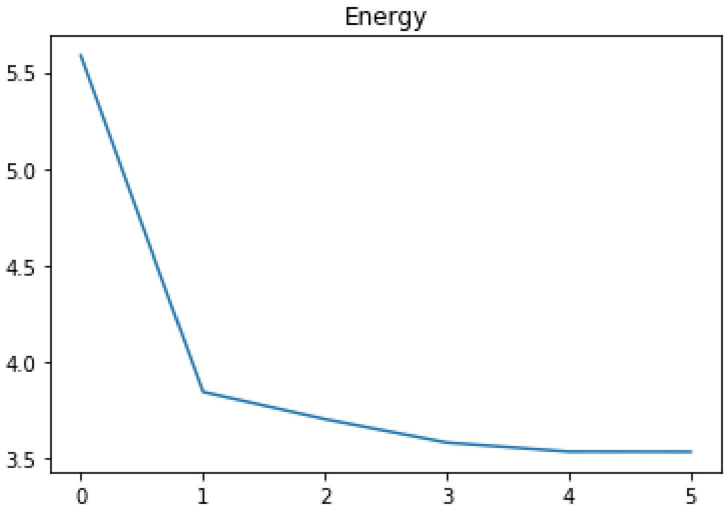


1. k = 3, Alpha = 1.3

In [213]:

```
clustering("sample_image.jpg", 3, 1, 1.3)
```

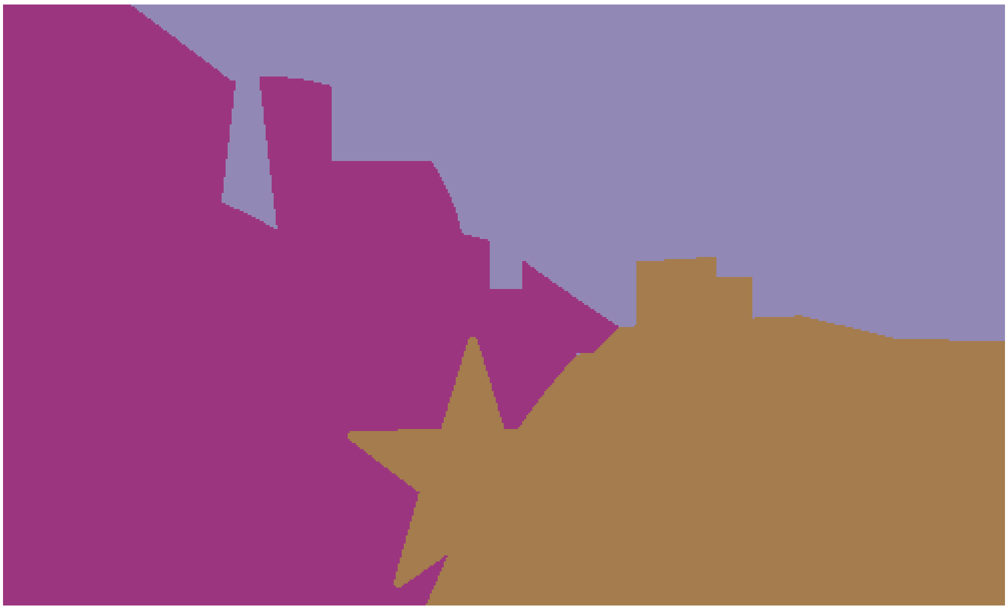


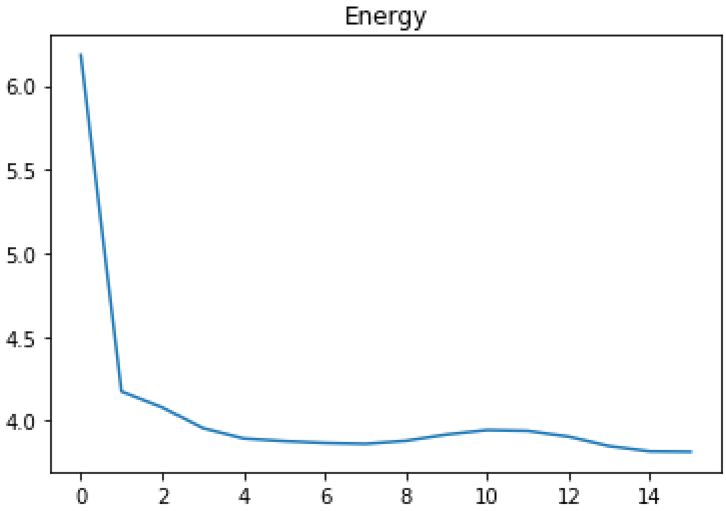


1. k = 3, Alpha = 1.6

In [252]:

```
clustering("sample_image.jpg", 3, 1, 1.6)
```




Energy