Nama : Almas Fauzia Wibawa
NIM : 17/409427/PA/17734

Tugas Pengenalan Pola

Implementasi K-Means

1. **Mengimpor data citra.**

```
import cv2
import numpy as np

img1 = cv2.imread('1.jpeg')
img1RGB = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img1vector = img1RGB.reshape((-1,img1RGB.shape[2]))

img2 = cv2.imread('2.jpg')
img2RGB = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
img2vector = img2RGB.reshape((-1,img2RGB.shape[2]))
```

Untuk mengolah citranya, digunakan library cv2. Setelah citra diimpor ke dalam program, perlu diubah dulu format warna pikselnya ke dalam RGB. Hal ini dikarenakan sifat cv2 yang secara default mengimpor citra dengan format warna BGR. Padahal, dibutuhkan fitur red, green, dan blue dalam proses segmentasi yang akan dilakukan.

Piksel citra masih tersimpan dalam bentuk matriks. Data tersebut perlu diubah bentuknya menjadi vektor agar lebih mudah untuk diolah dalam fungsi K-Means nantinya. Setelah itu, data citra siap diolah dalam fungsi K-Means.

2. **Mendefinisikan fungsi K-Means.**

Secara garis besar, yang dilakukan dalam fungsi K-Means untuk segmentasi citra adalah adalah:

a. Memilih secara random piksel untuk menjadi centroid awalnya.
b. Menghitung jarak antara semua piksel dengan semua centroid. Pilih yang paling kecil dan kelompokkan piksel tersebut dengan centroid yang bersangkutan.
c. Mengganti centroid dengan rata-rata semua piksel dalam satu cluster itu.
d. Ulangi Langkah b dan c sampai tidak terjadi perubahan centroid pada langkah c.

```
from sklearn.metrics import pairwise_distances_argmin
import random

def kMeans(X, k):
    centers = X[random.sample(range(0, X.shape[0]), k)]

    while True:
        labels = pairwise_distances_argmin(X, centers)

        new_centers = np.nan_to_num(np.array([X[labels == i].mean(0) for i in range(k)]))

        if np.all(centers == new_centers):
            break
        centers = new_centers

    return centers, labels
```

Langkah a dilakukan di line pertama pada fungsi kMeans(). Diambil 3 piksel secara random sebanyak jumlah cluster yang ingin dibentuk.

Pada line pertama di dalam perulangan, dilakukan menetapan label. Digunakan fungsi pairwise_distances_argmin(). Pada fungsi tersebut, dilakukan perhitungan jarak antara setiap piksel dengan centroid yang ada. Setelah itu, setiap piksel memilih centroid yang berjarak terkecil. Kemudian, ditetapkan label kepada setiap piksel berupa indeks dari centroid tersebut.

Langkah selanjutnya adalah memperbarui centroid. Langkah ini diimplementasikan pada perintah kedua di dalam perulangan. Centroid diganti dengan merata-rata setiap fitur pada semua piksel yang menjadi anggota setiap cluster. Untuk melakukan hal ini, digunakan fungsi mean() yang dapat digunakan untuk menghitung rata-rata.

Pada langkah ini, diperlakukan juga fungsi nan_to_num(). Hal ini diperlukan karena hasil fungsi mean() memiliki kemungkinan bernilai NaN. Nilai tersebut tidak relevan untuk konteks citra, dimana setiap piksel pasti memiliki nilai dalam range 0-255.

Setelah itu, dilakukan pengecekan apakah cluster yang baru saja terbentuk sudah konvergen. Pengecekan dilakukan dengan mencocokkan apakah centroid baru sama dengan centroid lama. Jika sama, berarti cluster sudah konvergen, dan iterasi K-Means sudah boleh dihentikan. Langkah ini diimplementasikan pada perintah ke-3 sampai ke-5 di dalam perulangan.

Fungsi ini akan mengembalikan 2 vektor, yaitu vektor berisi centroid akhir dari semua cluster dan vektor berisi label akhir yang ditetapkan pada setiap piksel pada citra.

3. Mengaplikasikan fungsi K-Means dengan K = 2.

Setelah fungsi K-Means didefinisikan, saatnya mengaplikasikan fungsi tersebut pada data citra yang telah diimpor di awal. Fungsi tersebut diimplementasikan kepada kedua citra dengan proses yang sama.

```
K1 = 2
centers1, labels1 = kMeans(img1vector, K1)
centers1 = np.uint8(centers1)
result1 = centers1[labels1.flatten()]
result1 = result1.reshape(img1RGB.shape)

centers2, labels2 = kMeans(img2vector, K1)
centers2 = np.uint8(centers2)
result2 = centers2[labels2.flatten()]
result2 = result2.reshape(img2RGB.shape)
```

Fungsi langsung dipanggil dengan 2 argumen, vektor yang menyimpan seluruh piksel citra, dan jumlah cluster yang akan dibentuk. Kali ini, jumlah cluster yang ingin dibentuk adalah 2. Langkah tersebut akan mengembalikan vektor centroid dan vektor label.

Setiap nilai pada vektor centroid kemudian dikonversikan ke dalam bentuk integer, layaknya data piksel direpresentasikan. Kemudian, didefinisikan vektor hasil yang berukuran sama seperti vektor citra. Vektor hasil akan menyimpan centroid dari cluster yang diterima setiap piksel. Vektor hasil kemudian diubah bentuk menjadi matriks dengan ukuran sama seperti matriks piksel citra awal agar dapat divisualisasikan.

4. Mengaplikasikan fungsi K-Means dengan K = 3.

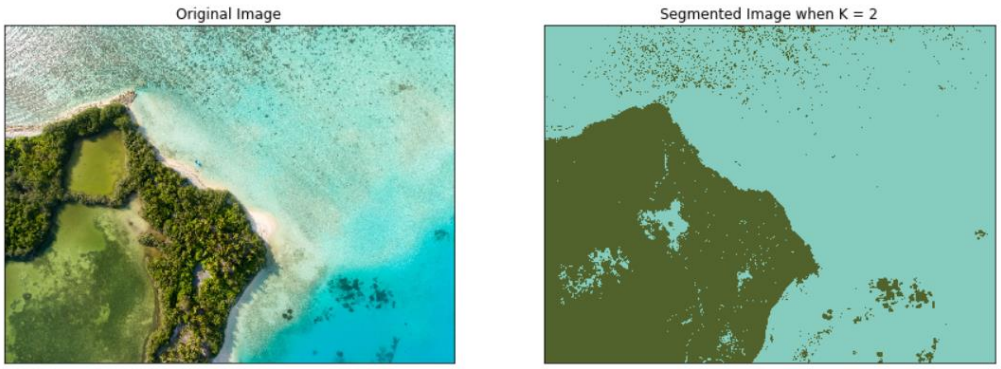
Proses ini sama seperti proses nomor 3. Bedanya hanya ada pada nilai K atau banyaknya cluster yang ingin dibentuk.

```
K2 = 3
centers3, labels3 = kMeans(img1vector, K2)
centers3 = np.uint8(centers3)
result3 = centers3[labels3.flatten()]
result3 = result3.reshape(img1RGB.shape)

centers4, labels4 = kMeans(img2vector, K2)
centers4 = np.uint8(centers4)
result4 = centers4[labels4.flatten()]
result4 = result4.reshape(img2RGB.shape)
```

5. Memvisualisasikan hasil segmentasi citra dengan K = 2 dan K = 3.

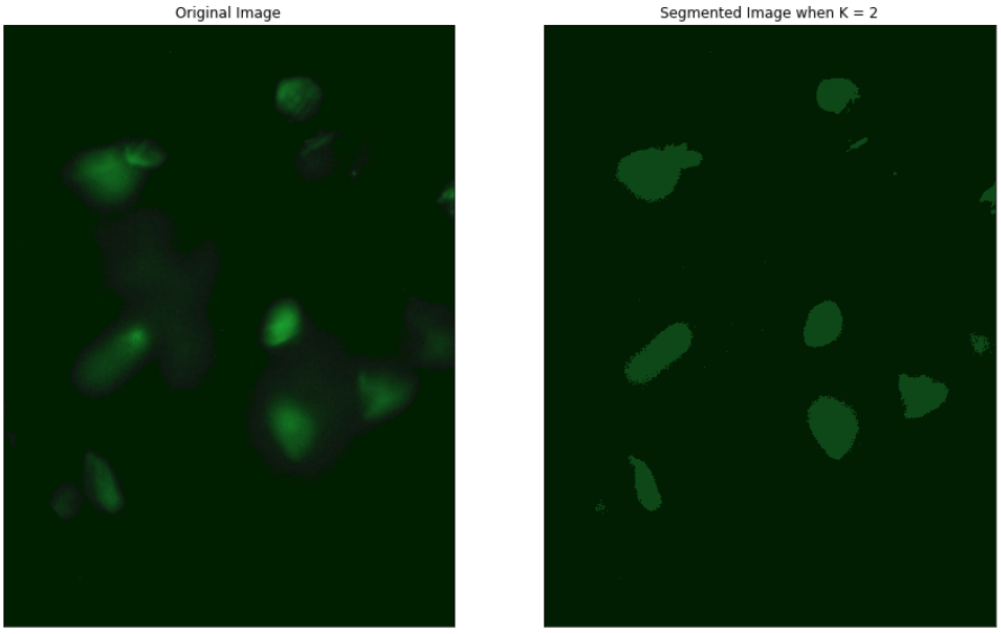
Digunakan library pyplot dari matplotlib untuk memvisualisasikan citra awal dan matriks hasil yang telah diperoleh. Berikut adalah visualisasi hasil segmentasi citra pertama dengan K = 2:

```
import matplotlib.pyplot as plt

figure_size = 15
plt.figure(figsize=(figure_size,figure_size))
plt.subplot(1,2,1),plt.imshow(img1RGB)
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2),plt.imshow(result1)
plt.title('Segmented Image when K = %i' % K1), plt.xticks([]), plt.yticks([])
plt.show()
```



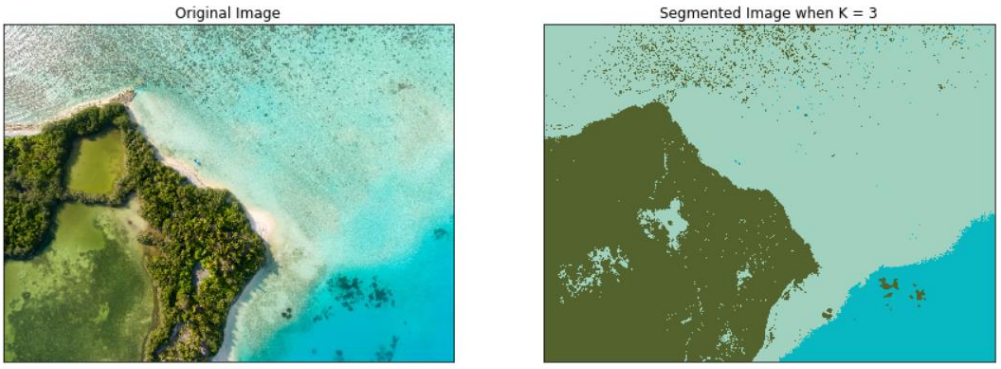
Berikut adalah visualisasi hasil segmentasi citra kedua dengan K = 2:

```
figure_size = 15
plt.figure(figsize=(figure_size,figure_size))
plt.subplot(1,2,1),plt.imshow(img2RGB)
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2),plt.imshow(result2)
plt.title('Segmented Image when K = %i' % K1), plt.xticks([]), plt.yticks([])
plt.show()
```



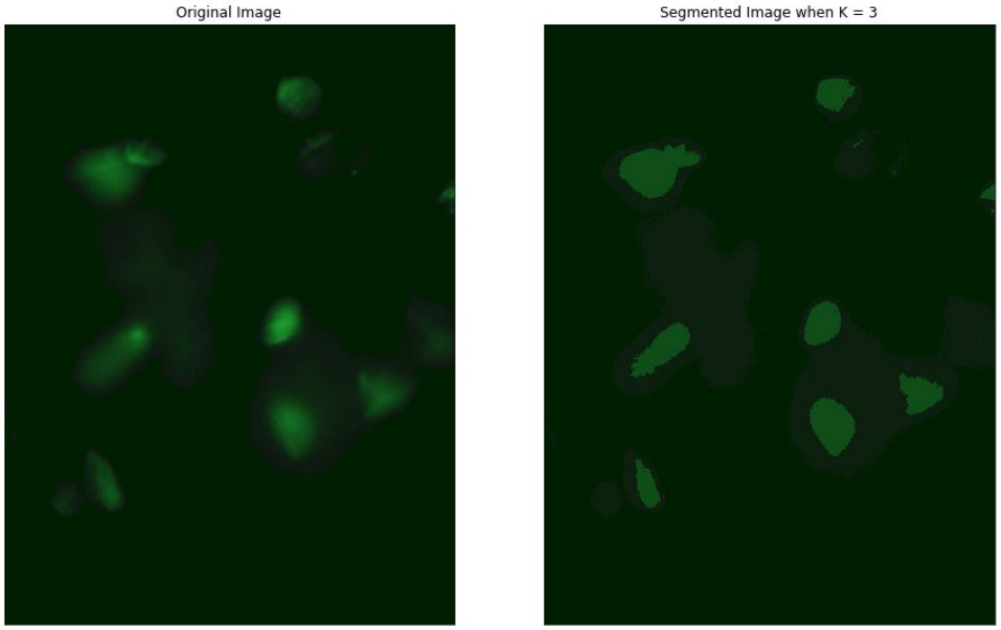
Berikut adalah visualisasi hasil segmentasi citra pertama dengan K = 3:

```
figure_size = 15
plt.figure(figsize=(figure_size,figure_size))
plt.subplot(1,2,1),plt.imshow(img1RGB)
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2),plt.imshow(result3)
plt.title('Segmented Image when K = %i' % K2), plt.xticks([]), plt.yticks([])
plt.show()
```



Berikut adalah visualisasi hasil segmentasi citra kedua dengan K = 3:

```
figure_size = 15
plt.figure(figsize=(figure_size,figure_size))
plt.subplot(1,2,1),plt.imshow(img2RGB)
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2),plt.imshow(result4)
plt.title('Segmented Image when K = %i' % K2), plt.xticks([]), plt.yticks([])
plt.show()
```



6. Analisa hasil.

Tujuan dari *image segmentation* adalah menyegmentasi citra berdasarkan objek yang ada di dalamnya. Oleh karena itu, dalam menilai seberapa baik segmentasi citra telah dilakukan, perlu dilihat apakah masing-masing objeknya sudah terpisahkan dengan baik.

Citra pertama merupakan citra panorama. Pada citra tersebut, terdapat gambar daratan dan laut atau pantai. Ketika dilakukan segmentasi dengan K = 2, kedua hal tersebut terbagi dengan baik. Terdapat 2 objek, dan citra pun terbagi menjadi 2 kelas. Walaupun begitu, terdapat titik-titik daratan yang berserakan di tengah laut.

Sedangkan, ketika K = 3, laut terbagi menjadi 2 kelas lagi. Dengan K = 3, tujuan segmentasi masih tercapai, yaitu memisah objek-objek yang berbeda. Dalam konteks ini dataran dan laut. Titik-titik daratan di tengah laut menjadi lebih sedikit disini. Sedangkan laut diklasifikasikan sebagai laut dengan baik. Perbedaan warna laut pada citra dapat terjadi karena perbedaan kadar garam atau kedalaman. Hal ini juga pantas digambarkan dalam segmentasi citra sehingga terbaginya laut menjadi 2 segmentasi juga bukan merupakan hal yang tidak baik.

Dari kedua ulasan tadi, dapat disimpulkan bahwa K = 3 dinilai lebih sesuai untuk diterapkan di citra pertama daripada K = 2.

Citra kedua merupakan citra biomedis. Pada citra tersebut, terdapat gambar sel. Dilakukannya segmentasi citra disini adalah untuk mengetahui keberadaan sel dalam citra. Dengan K = 2, citra sel dan citra background tersegmentasi dengan baik.

Sedangkan, dengan K = 3, terdapat 3 segmentasi, yaitu background, sel, dan bayang-bayang sel atau sel lain yang tidak cukup terlihat untuk tertangkap gambar. Tentu saja segmentasi ketiga yang tidak ada di hasil segmentasi dengan K = 2 ini sebenarnya juga merupakan informasi yang cukup penting. Oleh karena itu, dapat disimpulkan juga bahwa jumlah cluster yang lebih sesuai diterapkan untuk segmentasi citra pada citra kedua adalah 3 dibandingkan dengan 2.

7. Aplikasikan elbow method pada setiap citra.

Dalam elbow method, dilakukan beberapa kali K-Means clustering dengan jumlah cluster yang berbeda. Kali ini, akan dilakukan K-Means clustering untuk jumlah cluster 1 sampai 10. Kesepuluh hasil clustering akan dibandingkan untuk memperoleh berapa jumlah cluster yang paling sesuai untuk data yang bersangkutan. Perbandingan hasil akan dilakukan dengan menghitung nilai Within-Cluster Sum of Square (WWCS) atau Sum of Squared Error (SSE)-nya. Oleh karena itu, terlebih dahulu, dibuat vektor untuk menyimpan nilai SSE dari setiap clustering yang dilakukan.

```
from sklearn.metrics.pairwise import paired_distances

def calculate_SSE(X, n):
    SSE = [[None] for k in range(n)]
    for k in range(n):
        print(k, end=" ")
        centers, labels = kMeans(X, k+1)
        centers = np.uint8(centers)
        result = centers[labels.flatten()]

        SSE[k] = 0
        SSE[k] = np.sum(paired_distances(X, result))
    return SSE
```

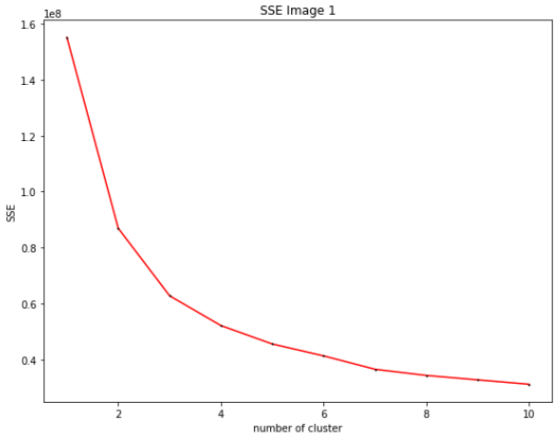
Kemudian, secara berulang, dilakukan K-Means dengan jumlah cluster sebagaimana yang diinginkan. Setiap clustering selesai, dilakukan perhitungan SSE. Hasilnya disimpan dalam vektor SSE yang telah dibuat sebelumnya.

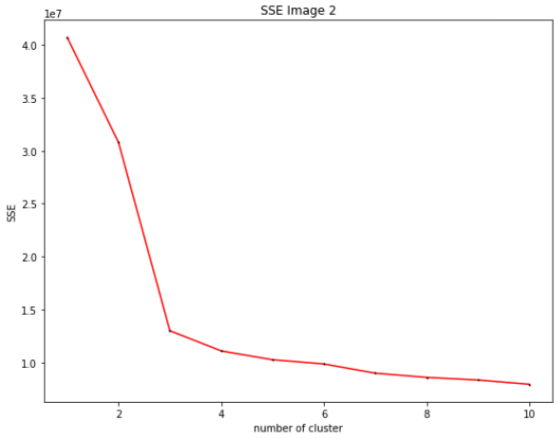
Fungsi tersebut kemudian diaplikasikan untuk menghitung SSE pada citra pertama dan kedua jika K = 1 sampai K = 10.

```
n = 10
SSE1 = calculate_SSE(img1vector, 10)
SSE2 = calculate_SSE(img2vector, 10)
```

Setelah itu, dilakukan visualisasi untuk SSE yang dihasilkan dari semua percobaan. Kembali digunakan library pyplot dari matplotlib.

```
import matplotlib.pyplot as plt

SSE1 = np.reshape(SSE1, (len(SSE1), 1))
SSE2 = np.reshape(SSE2, (len(SSE2), 1))
k = [i+1 for i in range(n)]

figure_size = 10
plt.figure(figsize=(20,7))
plt.subplot(1,2,1), plt.plot(k, SSE1, 'ro-', ms=1, mec='k')
plt.title('SSE Image 1'), plt.xlabel('number of cluster'), plt.ylabel('SSE')
plt.subplot(1,2,2),plt.plot(k, SSE2, 'ro-', ms=1, mec='k')
plt.title('SSE Image 2'), plt.xlabel('number of cluster'), plt.ylabel('SSE')
plt.show()
```





Dapat dilihat, pada citra pertama, grafik berlanjut menyerupai garis linier setelah K = 4. Sehingga, dapat disimpulkan bahwa jumlah cluster yang paling sesuai digunakan dalam proses clustering pada citra pertama adalah sebanyak 4.

Pada citra kedua, grafik berlanjut menyerupai garis linier setelah K = 3. Dapat disimpulkan bahwa 3 adalah jumlah cluster yang paling tepat digunakan untuk proses clustering pada citra kedua.