Drishtti Narwal

AI & ML (A2)

PRN No. 22070126037

Aim:

The aim of this experiment is to implement K-Means clustering on an image to segment it into distinct color regions. The experiment explores clustering in both 3D (RGB) and 5D (LAB) color spaces to understand how different color representations affect the clustering results. Additionally, the experiment uses the Elbow Method to determine the optimal number of clusters for effective image segmentation.

Objectives:

1. To understand the concept of K-Means clustering and its application in image segmentation.
2. To preprocess an image and convert it into suitable color spaces (RGB and LAB) for clustering.
3. To implement K-Means clustering on the image data and visualize the resulting clusters.
4. To use the Elbow Method to determine the optimal number of clusters for the given image.
5. To evaluate the performance of clustering using metrics like Inertia and Silhouette Score.
6. To compare the results of clustering in RGB and LAB color spaces.

Theory:

1. K-Means Clustering: K-Means is an unsupervised machine learning algorithm used to partition data into K clusters based on similarity. It works by minimizing the inertia, which is the sum of squared distances between data points and their closest cluster centroids.In image processing, K-Means is used for color segmentation, where pixels are grouped into clusters based on their color values.
2. RGB Color Space: RGB (Red, Green, Blue) is a 3D color space where each pixel is represented by three values corresponding to the intensity of red, green, and blue.K-Means clustering in RGB space groups pixels based on their color similarity in the RGB space.
3. LAB Color Space: LAB is a 3D color space that separates color information into: L: Lightness (intensity) A: Green to Red color component B: Blue to Yellow color component LAB is perceptually uniform, meaning it aligns better with human vision. It can be extended to 5D by adding additional features (e.g., spatial coordinates).
4. Elbow Method: The Elbow Method is used to determine the optimal number of clusters (K) for K-Means.It involves plotting the inertia (sum of squared distances) against the number of clusters and identifying the "elbow" point where the inertia starts to decrease at a slower rate.
5. Performance Metrics: Inertia: Measures how tightly the clusters are formed. Lower inertia indicates better clustering.Silhouette Score: Measures how similar an object is to its own cluster compared to other clusters. A higher score indicates better-defined clusters.

```
# First install required libraries if not already available
!pip install matplotlib scikit-learn opencv-python Pillow
```

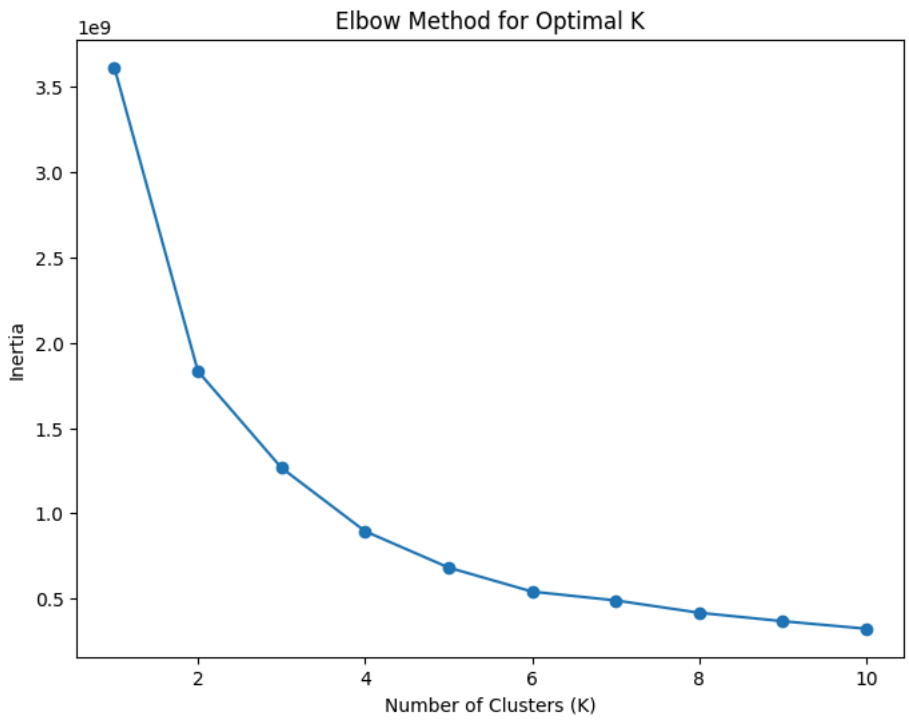
```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (11.1.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
```

```
# Import necessary libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from skimage import io
from skimage.color import rgb2lab


# Function to load an image and reshape it
def load_and_preprocess_image(image_path, color_space='rgb'):
    img = io.imread(image_path)

    # Resize image if necessary (optional)
    img = cv2.resize(img, (500, 500))

    if color_space == 'rgb':
        if img.shape[-1] == 3: # Check if the last dimension is 3
            return img.reshape((-1, 3)) # Reshaping to (pixels, 3 colors)
        else:
            # If not RGB, handle it appropriately (e.g., convert to RGB)
            # Here, we'll raise an exception for demonstration purposes
            raise ValueError("Image is not in RGB format. Please provide an RGB image.")
    elif color_space == 'lab':
        img_lab = rgb2lab(img)
        return img_lab.reshape((-1, 3)) # In LAB color space

    return img.reshape((-1, 3))

# Function to perform KMeans clustering and visualize results
def kmeans_clustering(data, k=5):
    # Perform KMeans clustering
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data)
    labels = kmeans.labels_
    centers = kmeans.cluster_centers_

    return kmeans, labels, centers

# Performance metrics
def plot_performance_metrics(kmeans, data):
    # Inertia
    inertia = kmeans.inertia_

    # Silhouette Score
    silhouette_avg = silhouette_score(data, kmeans.labels_)

    print(f"Inertia: {inertia}")
    print(f"Silhouette Score: {silhouette_avg}")


import matplotlib.pyplot as plt
import cv2

# Load the image
image = cv2.imread('/content/drive/MyDrive/SEM_6/CV-Lab/design.jpg')

# Convert the image from BGR to RGB (OpenCV loads images in BGR format)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the image using matplotlib
plt.imshow(image_rgb)
plt.axis('off')  # Hide the axes
plt.show()
```

+ Code + Text

```
image_path = "/content/drive/MyDrive/SEM_6/CV-Lab/design.jpg"

# Load and preprocess the image (3D color space: RGB)
img_rgb = load_and_preprocess_image(image_path, color_space='rgb')

# Define the data variable for clustering
data = img_rgb  # Use the preprocessed RGB image data

# Elbow Method to determine optimal K
distortions = []
for i in range(1, 11):
    kmeans_temp = KMeans(n_clusters=i, random_state=42)
    kmeans_temp.fit(data)  # Fit K-Means on the data
    distortions.append(kmeans_temp.inertia_)  # Append inertia to the distortions list

# Plot the Elbow Method graph
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), distortions, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.show()
```

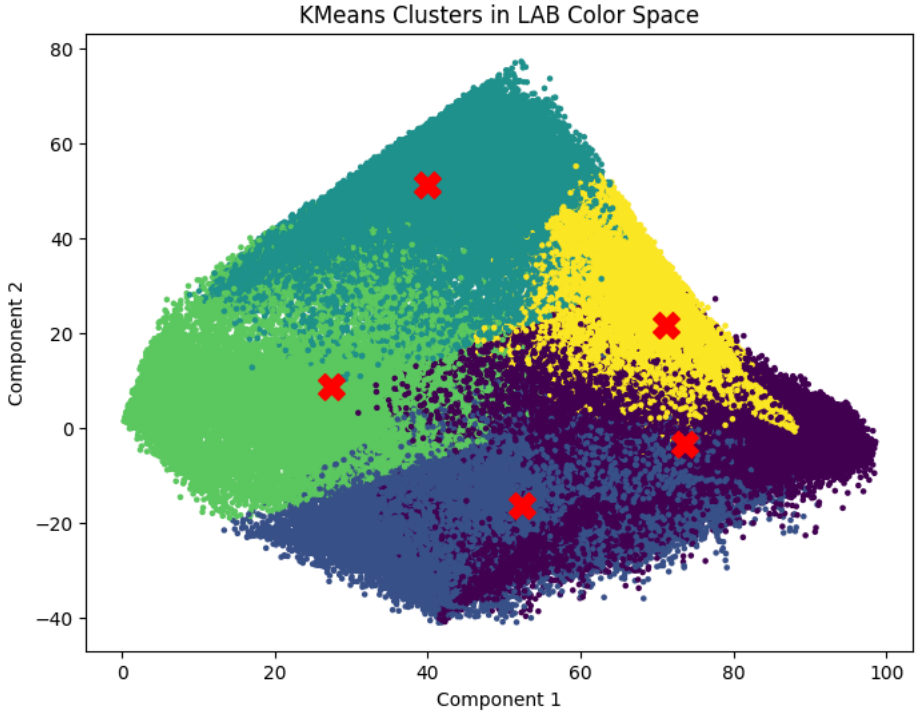



```
# Plot clusters
def plot_clusters(data, labels, centers, color_space='rgb'):
    plt.figure(figsize=(8, 6))
    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', s=5)
    plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, marker='X')

    plt.title(f'KMeans Clusters in {color_space.upper()} Color Space')
    plt.xlabel(f'Component 1')
    plt.ylabel(f'Component 2')
    plt.show()


# Load image (Change path to your own image)
image_path = "/content/drive/MyDrive/SEM_6/CV-Lab/design.jpg"

# Load and preprocess the image (3D color space: RGB)
img_rgb = load_and_preprocess_image(image_path, color_space='rgb')

# KMeans clustering in RGB color space
kmeans_rgb, labels_rgb, centers_rgb = kmeans_clustering(img_rgb, k=5)
plot_clusters(img_rgb, labels_rgb, centers_rgb, color_space='rgb')
```


```
# Performance metrics for RGB clustering
plot_performance_metrics(kmeans_rgb, img_rgb)
```

```
Inertia: 682394195.3562326
Silhouette Score: 0.4201575248001165
```

```
# Convert to LAB color space for 5D clustering (taking L, A, B, and extra channels)
img_lab = load_and_preprocess_image(image_path, color_space='lab')

# KMeans clustering in LAB color space
kmeans_lab, labels_lab, centers_lab = kmeans_clustering(img_lab, k=5)
plot_clusters(img_lab, labels_lab, centers_lab, color_space='lab')
```




```
# Performance metrics for LAB clustering
plot_performance_metrics(kmeans_lab, img_lab)
```

```
Inertia: 98161621.41229255
Silhouette Score: 0.40942917439853244
```

Conclusion

In this experiment, K-Means clustering was applied to segment an image into distinct color regions using both RGB and LAB color spaces. While RGB clustering effectively grouped pixels based on their red, green, and blue intensity values, it struggled with smooth gradients and highly textured areas. In contrast, LAB clustering, being perceptually uniform, offered better color separation, particularly in images with subtle color variations and complex structures. The Elbow Method was employed to determine the optimal number of clusters, ensuring that the results were neither too granular nor too coarse. Performance metrics like inertia and the Silhouette Score provided a quantitative evaluation of the clustering quality. Overall, the experiment highlighted the importance of choosing the appropriate color space, with LAB proving more effective for complex color structures, making K-Means clustering valuable for applications like object detection, background removal, and color-based image analysis.