Experiment no:8

1. **Aim:** To implement k-mean clustering for the image segmentation.
2. **Software Tool Used:** MATLAB
3. **Theory:**

K-means Clustering for Image Segmentation: Image segmentation is a crucial task in computer vision, where an image is divided into meaningful regions to simplify or change its representation, making it more meaningful and easier to analyse. K-means clustering is a popular unsupervised machine learning algorithm used for this purpose, as it can segment an image into distinct clusters based on pixel intensity, color, or other features.

K-means Clustering: K-means clustering is a partitioning algorithm that divides a dataset into K clusters, where each data point belongs to the cluster with the closest mean. The algorithm operates iteratively to minimize the variance within each cluster. In the context of image segmentation, K-means groups similar pixels into clusters based on their color, brightness, or spatial properties, which can then be used to segment the image.
The basic steps in K-means clustering are as follows:

1. **Initialization**: Select K initial centroids randomly or using a heuristic.
2. **Assignment Step**: Assign each data point (pixel) to the nearest centroid based on a distance metric (usually Euclidean distance).
3. **Update Step**: Calculate new centroids by averaging the data points assigned to each cluster.
4. **Repeat**: Repeat the assignment and update steps until the centroids no longer change significantly or a specified number of iterations is reached.

Objective Function: K-means minimizes the following objective function, which represents the sum of squared distances between each pixel X_i and the cluster centre μ_k to which it belongs:

$$\text{Distance}(p_i, c_j) = \sqrt{\left(\frac{x_i - x_j}{\text{max\_row}}\right)^2 + \left(\frac{y_i - y_j}{\text{max\_col}}\right)^2 + \left(\frac{I_i - I_j}{\text{max\_intensity}}\right)^2}$$

- pi= (xi, yi, Ii) represents the pixel feature with spatial coordinates (xi, yi) and intensity Ii.
- cj= (xj, yj, Ij) is the centroid feature with spatial coordinates (xj, yj) and intensity Ij.
- max_row and max_col are the maximum values for row and column indices, respectively (used for normalization of spatial differences).
- max_intensity is the maximum possible intensity value (often 255 for 8-bit grayscale images) for normalizing intensity differences. By minimizing this objective, K-means ensures that the pixels within each cluster are as similar as possible, thus achieving effective segmentation.

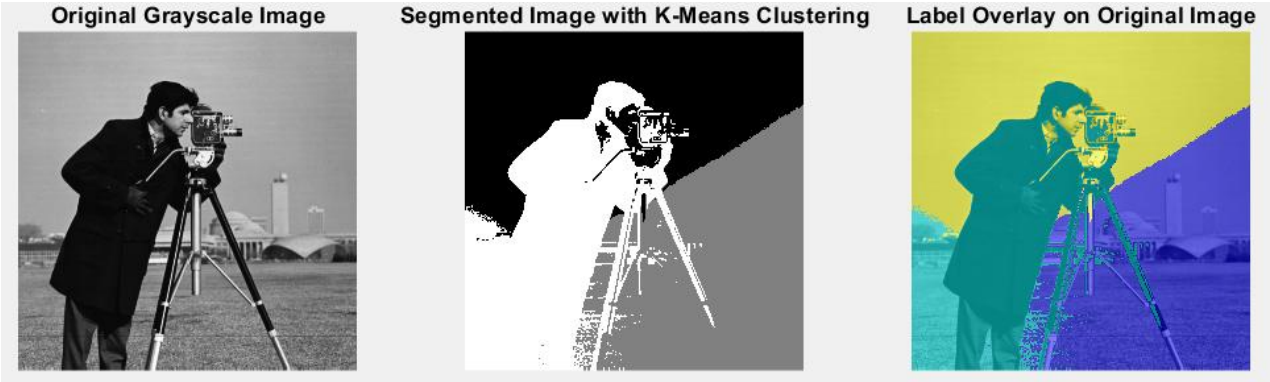
  For the RGB image the Euclidean Distance:

$$\text{distance}_{i,j} = \sqrt{\left(\frac{x_j - \text{centroid}_{i,x}}{\text{rows}}\right)^2 + \left(\frac{y_j - \text{centroid}_{i,y}}{\text{cols}}\right)^2 + \left(\frac{R_j - \text{centroid}_{i,R}}{255}\right)^2 + \left(\frac{G_j - \text{centroid}_{i,G}}{255}\right)^2 + \left(\frac{B_j - \text{centroid}_{i,B}}{255}\right)^2}$$

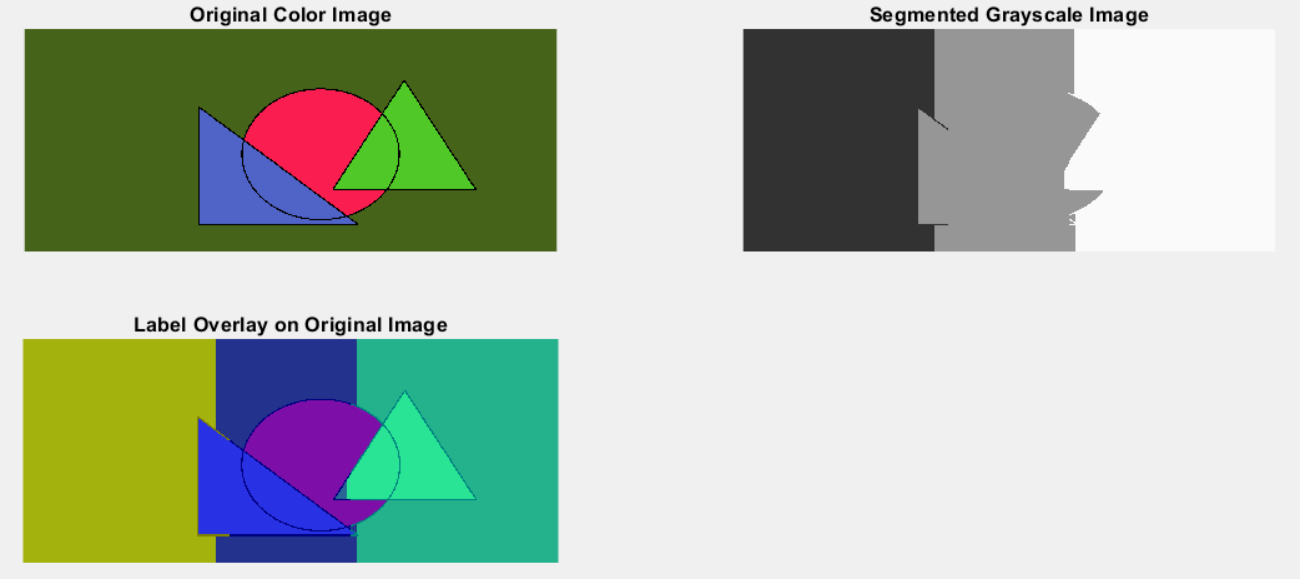
---

 rows and cols\text{cols}cols: Total number of rows and columns in the image, used to normalize spatial coordinates.
 255: Maximum intensity for RGB channels, used to normalize color intensities to a scale of 0 to 1.

4. Result:



Converged after 42 iterations



Converged after 20 iterations

5. Discussion:

The given code applies K-means clustering with K=3 to segment the *cameraman* grayscale image based on spatial coordinates (x, y) and pixel intensity. The segmentation result shows three distinct clusters, each representing different regions of the image based on similarity in location and intensity.

Grayscale Image Segmentation

1. **Original Image:**
   - The grayscale input image is displayed in the first subplot, showcasing its intensity distribution.
   - Pixel intensity values range between 0 and 255, representing different shades of Gray.
2. **Segmented Image:**
   - The segmentation divides the image into k=3 distinct regions based on intensity values.
   - Each region corresponds to a cluster and is assigned an intensity value chosen from a predefined set (e.g., 50, 150, 250).

    - The segmentation successfully identifies areas with similar intensity patterns, grouping them into clusters.
3. **Overlay Image:**
    - The label overlay visually combines the segmented regions with the original grayscale image.
    - The semi-transparent overlay helps to highlight the boundaries of clusters while retaining the details of the original image.
    - This enhances interpretability, especially in identifying regions with subtle intensity transitions.

RGB Image Segmentation

1. **Original RGB Image:**
    - The input color image consists of three channels (R, G, B) that define the color of each pixel.
    - The original image appears vibrant and complex due to varying intensities in the RGB channels.
2. **Segmented Image:**
    - The segmented output transforms the original color image into distinct regions based on spatial and color intensity similarity.
    - Each cluster corresponds to areas with similar color intensities, represented as shades of Gray in the output.
    - This demonstrates the algorithm's ability to group pixels with related color characteristics, such as red-dominant, green-dominant, or blue-dominant areas.
3. **Overlay Image:**
    - The label overlay superimposes the segmented regions onto the original RGB image, creating a clear visual representation of the clusters.
    - The segmentation boundaries are clearly visible, highlighting the effectiveness of the clustering process in distinguishing different regions.

Convergence and Performance

- **Iteration Count:**
    - Both grayscale and RGB segmentation converged after a reasonable number of iterations, indicating stability in the clustering process.
    - The centroids' adjustment in each iteration progressively minimized the intra-cluster variance.
- **Cluster Representation:**
    - Random initialization of centroids leads to variability in the results for each run, but the final output consistently reflects meaningful segmentation.

Key Observations from the Results:

- **Feature Space:**

Grayscale segmentation relied solely on intensity and spatial position, whereas RGB segmentation incorporated all three-color channels along with spatial information.
Normalization of features (e.g., dividing by max intensity or dimensions) ensured balanced clustering and avoided bias toward any single feature.

- **Cluster Quality:**

The segmented images reflect meaningful groupings based on the given k=3.

Fine-tuning the value of k could provide more granular or broader segmentation depending on the application's needs.

- **Segmentation Accuracy:**

The RGB segmentation output captures color transitions more effectively than the grayscale segmentation due to the inclusion of richer feature dimensions.

Effect of the Value of K on Segmentation

In this code, K=3 clusters are used, but changing K can significantly impact the segmentation result:

- **Lower Values of K**: If Kis set to a smaller value, such as K=2, the segmentation divides the image into fewer clusters, which may result in more generalized regions. For instance, the background and subject might be grouped together, leading to a loss of finer detail in segmentation.
- **Higher Values of K**: Increasing K, for example to K=4 or K=5, would allow the segmentation to capture more nuanced details by creating smaller, more specific clusters. This might reveal subtle variations in intensity within different regions of the image but can also lead to over-segmentation, where small differences in pixel values lead to unnecessary fragmentation.
- **Optimal Value of K**: Choosing the best value of K is often empirical and depends on the desired level of detail. For simple segmentation, lower values of K might suffice, while for more complex images with distinct regions, a higher K is useful.

6. Conclusion:

The K-Means clustering algorithm successfully segmented both grayscale and RGB images into distinct regions. The results demonstrate the algorithm's versatility in handling different types of image data. The overlay visualizations further enhance the understanding of cluster boundaries.

To improve the segmentation quality:

- Experimenting with adaptive initialization methods (e.g., K-Means for centroid selection) could yield faster convergence and better clusters.
- Increasing k for images with complex details could capture finer nuances, while reducing k could group broader regions.

7. **CODE:**

```
img = imread("cameraman.tif");
if size(img, 3) == 3
   img = rgb2gray(img); % Ensure it's grayscale
end
img = double(img);
[rows, cols] = size(img);
[x, y] = meshgrid(1:cols, 1:rows);
features = [x(:), y(:), img(:)];
k = 3;
centroids = [100, 50, 128; 200, 150, 200; 50, 200, 64];
cluster_assignment = zeros(size(features, 1), 1);
```

```
iteration = 0;     max_row = rows;     max_col = cols;   max_intensity = 255;
while true
   iteration = iteration + 1;
   distances = zeros(size(features, 1), k);
   for i = 1:k
      diff_row = (features(:, 1) - centroids(i, 1)) / max_row;
      diff_col = (features(:, 2) - centroids(i, 2)) / max_col;
      diff_intensity = (features(:, 3) - centroids(i, 3)) / max_intensity;
      distances(:, i) = sqrt(diff_row.^2 + diff_col.^2 + diff_intensity.^2);
   end
   [~, cluster_assignment] = min(distances, [], 2);
   new_centroids = zeros(k, size(features, 2));
   for i = 1:k
      cluster_points = features(cluster_assignment == i, :);
      if ~isempty(cluster_points)
         new_centroids(i, :) = mean(cluster_points, 1);
      else
         new_centroids(i, :) = features(randi(size(features, 1)), :);
      end
   end
   if all(new_centroids == centroids)
      disp(['Converged after ', num2str(iteration), ' iterations']);
      break;
   end
   centroids = new_centroids;
end
labeled_img = reshape(cluster_assignment, [rows, cols]);
segmented_img = zeros(rows, cols);
intensities = linspace(0, 255, k);
for i = 1:k
   segmented_img(labeled_img == i) = intensities(i);
end
original_img_uint8 = uint8(img);
overlay_img = labeloverlay(original_img_uint8, labeled_img, 'Transparency', 0.5);

figure;
subplot(1, 3, 1); imshow(uint8(img)); title('Original Grayscale Image');
subplot(1, 3, 2); imshow(uint8(segmented_img)); title('Segmented Image with K-Means
Clustering');
subplot(1, 3, 3); imshow(overlay_img); title('Label Overlay on Original Image');
img = imread("COLUR.png");
img = double(img);
[rows, cols, channels] = size(img);
[x, y] = meshgrid(1:cols, 1:rows);
red_channel = img(:, :, 1); green_channel = img(:, :, 2); blue_channel = img(:, :, 3);
features = [x(:), y(:), red_channel(:), green_channel(:), blue_channel(:)];
k = 3;
random_indices = randperm(size(features, 1), k);
centroids = features(random_indices, :);
cluster_assignment = zeros(size(features, 1), 1);
```

```
iteration = 0;
while true
    iteration = iteration + 1;
    distances = zeros(size(features, 1), k);
    for i = 1:k
        diff_row = (features(:, 1) - centroids(i, 1)) / rows;
        diff_col = (features(:, 2) - centroids(i, 2)) / cols;
        diff_red = (features(:, 3) - centroids(i, 3)) / 255;
        diff_green = (features(:, 4) - centroids(i, 4)) / 255;
        diff_blue = (features(:, 5) - centroids(i, 5)) / 255;
        distances(:, i) = sqrt(diff_row.^2 + diff_col.^2 + diff_red.^2 + diff_green.^2 + diff_blue.^2);
    end
    [~, cluster_assignment] = min(distances, [], 2);
    new_centroids = zeros(k, size(features, 2));
    for i = 1:k
        cluster_points = features(cluster_assignment == i, :);
        if ~isempty(cluster_points)
            new_centroids(i, :) = mean(cluster_points, 1);
        else
            new_centroids(i, :) = features(randi(size(features, 1)), :);
        end
    end
    if all(new_centroids == centroids)
        disp(['Converged after ', num2str(iteration), ' iterations']);
        break;
    end
    centroids = new_centroids;
end
labeled_img = reshape(cluster_assignment, [rows, cols]);
segmented_img = zeros(rows, cols);
intensities = [50, 150, 250];
for i = 1:k
    mask = labeled_img == i;
    segmented_img(mask) = intensities(i);
end
original_img_uint8 = uint8(img);
segmented_img_uint8 = uint8(segmented_img);
overlay_img = labeloverlay(original_img_uint8, labeled_img, 'Transparency', 0.5);

figure;
subplot(2, 2, 1); imshow(original_img_uint8); title('Original Color Image');
subplot(2, 2, 2); imshow(segmented_img_uint8); title('Segmented Grayscale Image');
subplot(2, 2, 3); imshow(overlay_img); title('Label Overlay on Original Image');
```