Homework 10

Submission by : Gagan Ullas Nirgun , Viswadeep Mallarapu Bhaskar

In [12]:
```
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import scipy.sparse.linalg as sp_linalg
import torch
from PIL import Image
from sklearn.cluster import KMeans, SpectralClustering
```

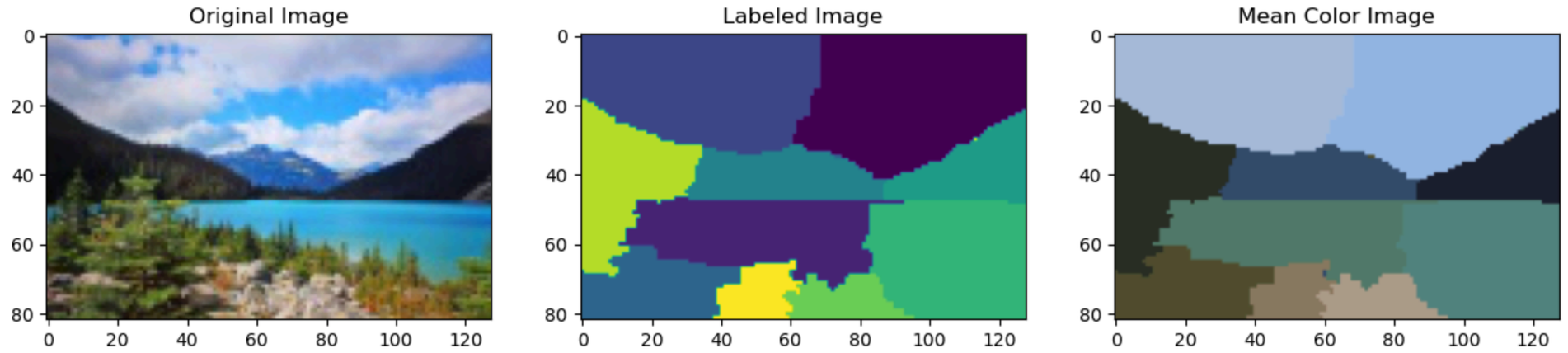
In [13]:
```
def preprocess_image(image_path):
    original_img = Image.open(image_path).convert('RGB')
    return np.array(original_img) / 255

def construct_adjacency_matrix(image_array):
    height, width, _ = image_array.shape
    nodes = height * width
    row_indices = []
    col_indices = []
    data = []

    for i in range(height):
        for j in range(width):
            for di, dj in ((-1, 0), (1, 0), (0, -1), (0, 1)):
                ni, nj = i + di, j + dj
                if 0 <= ni < height and 0 <= nj < width:
                    node_index = i * width + j
                    neighbor_index = ni * width + nj
                    row_indices.append(node_index)
                    col_indices.append(neighbor_index)
                    diff = image_array[i, j] - image_array[ni, nj]
                    distance = np.linalg.norm(diff) ** 2
                    data.append(np.exp(-distance / 0.1 ** 2))

    adjacency_matrix = sp.sparse.coo_matrix((data, (row_indices, col_indices)), shape=(nodes, nodes))
    return adjacency_matrix.tocsr()

def normalize_adjacency_matrix(adjacency_matrix):
    degrees_sqrt_inv = 1.0 / np.sqrt(adjacency_matrix.sum(axis=1))
    normalized_adj = adjacency_matrix.multiply(degrees_sqrt_inv)
    normalized_adj = normalized_adj.multiply(degrees_sqrt_inv.T)
    return normalized_adj

def sparse_to_torch_coo(sparse_matrix):
    torch_adj = torch.sparse_coo_tensor(
        torch.LongTensor([sparse_matrix.row, sparse_matrix.col]),
        torch.FloatTensor(sparse_matrix.data),
        torch.Size(sparse_matrix.shape)
    )
    return torch_adj.coalesce()

image_path = 'scene2.jpg'
image_array = preprocess_image(image_path)

adjacency_matrix = construct_adjacency_matrix(image_array)

normalized_adjacency_matrix = normalize_adjacency_matrix(adjacency_matrix)
```

```
torch_adjacency_matrix = sparse_to_torch_coo(normalized_adjacency_matrix)
```

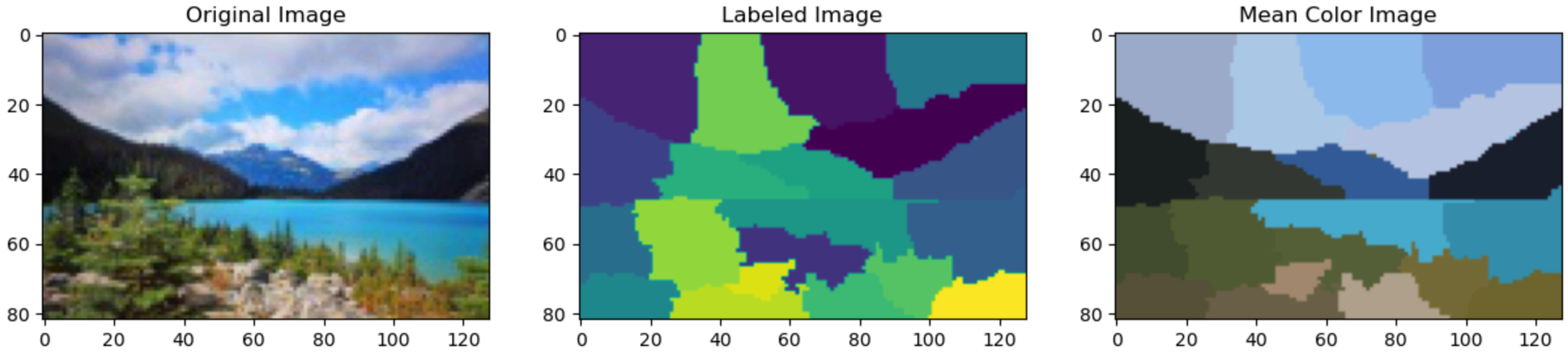
In [17]:

```
def clustered_mean_image_generator(image_array, torch_adjacency_matrix, n_clusters):

    _, normalized_eigenvectors = torch.lobpcg(torch_adjacency_matrix, k=n_clusters, largest=True)
    normalized_eigenvectors = normalized_eigenvectors / torch.norm(normalized_eigenvectors, dim=0, keepdim=True)
    normalized_eigenvectors = normalized_eigenvectors / torch.norm(normalized_eigenvectors, dim=1, keepdim=True)

    numpy_eigenvectors = normalized_eigenvectors.detach().numpy()

    km = KMeans(n_clusters=n_clusters, n_init='auto', init='k-means++', random_state=42)
    labels = km.fit_predict(numpy_eigenvectors)

    labeled_image = labels.reshape(image_array.shape[0], image_array.shape[1])

    mean_color_image = np.zeros_like(image_array)
    for cluster_label in range(n_clusters):
        indices = np.where(labeled_image == cluster_label)
        if indices[0].size > 0:
            mean_color = np.mean(image_array[indices], axis=0)
            mean_color_image[indices] = mean_color

    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)
    plt.title('Original Image')
    plt.imshow(image_array)

    plt.subplot(1, 3, 2)
    plt.title('Labeled Image')
    plt.imshow(labeled_image)

    plt.subplot(1, 3, 3)
    plt.title('Mean Color Image')
    plt.imshow(mean_color_image)

    plt.show()

print("-------------1 (a) & (b) --------------")
clustered_mean_image_generator(image_array, torch_adjacency_matrix, n_clusters=10)
print("-------------1 (c) --------------")
clustered_mean_image_generator(image_array, torch_adjacency_matrix, n_clusters=20)
```

```
-------------1 (a) & (b) --------------
```




```
-------------1 (c) --------------
```


Original Image
Labeled Image
Mean Color Image
0
20
40
60
80
0
20
40
60
80
100
120