Image segmentation using mean shift clustering algorithm

1. Importing necessary libraries

First, we need to import the libraries required for our analysis: `numpy` for numerical operations, `matplotlib` for plotting images, `cv2` (OpenCV) for image processing, and `sklearn` for implementing the mean shift clustering algorithm for image segmentation.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from sklearn.cluster import MeanShift
```

2. Uploading the image

Next, we upload an image file from our local machine to the Google Colab environment.

```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
```

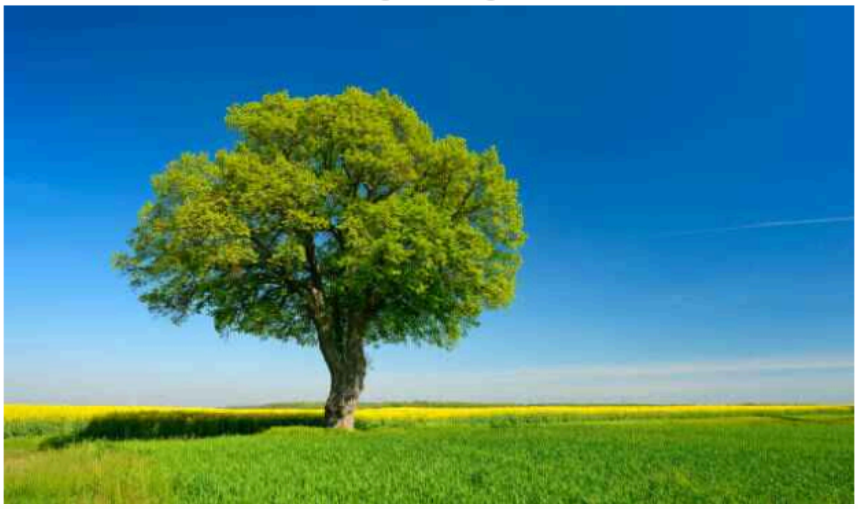
3. Preprocessing the image

The uploaded image is then read using OpenCV and converted from BGR to RGB format for proper visualisation.

```
for filename in uploaded.keys():
  original_image = cv2.imread(filename)
  original_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)

  # Displaying the original image
  plt.figure(figsize=(8,8))
  plt.imshow(original_image)
  plt.axis('off')
  plt.title('Original Image')
  plt.show()
```

Original Image



4. Reshaping the image

Now the image is reshaped into a 2D array where each row corresponds to a pixel and each column corresponds to the RGB colour values. The pixel values are then converted to a floating-point format, which is required for the Mean Shift algorithm.

Note: The original image is downscaled before applying Mean Shift. This will reduce the number of pixels, thus speeding up the clustering process.

```
original_image = cv2.resize(original_image, (original_image.shape[1] // 2, original_image.shape[0] // 2))
pixel_values = original_image.reshape((-1, 3))
pixel_values = np.float32(pixel_values)
```

5. Implementing the mean shift clustering algorithm

Instead of using all pixels for clustering, we randomly sample a subset of pixels (say, 10%). This will significantly reduce computation time.

```
sample_size = int(0.1 * pixel_values.shape[0])
sample_indices = np.random.choice(pixel_values.shape[0], sample_size, replace=False)
sampled_pixels = pixel_values[sample_indices]
```

The Mean Shift clustering algorithm is applied to the reshaped pixel values. The `bandwidth` parameter controls the size of the area used for clustering. The algorithm groups the pixel colours into clusters based on their similarity.

```
mean_shift = MeanShift(bandwidth=50)
mean_shift.fit(sampled_pixels)
```

```
▾    MeanShift
MeanShift(bandwidth=50)
```

6. Creating the segmented image

Next, we replace each pixel's colour with the colour of its corresponding cluster centre. The segmented image is reshaped back to the original image's dimensions (based on the sampled pixels) and converted back to an unsigned integer format for proper display.

```
labels = np.full(pixel_values.shape[0], -1)
labels[sample_indices] = mean_shift.labels_
segmented_image = mean_shift.cluster_centers_[labels].reshape(original_image.shape).astype(np.uint8)
```

7. Visualising the result

Finally, we plot the results of the segmentation.

```
plt.figure(figsize=(12,6))

# Displaying the original image
plt.subplot(1, 2, 1)
plt.imshow(original_image)
plt.axis('off')
plt.title('Original Image')

# Displaying the segmented image
plt.subplot(1, 2, 2)
plt.imshow(segmented_image)
plt.axis('off')
plt.title('Segmented Image')

plt.show()
```