EXPERIMENT NO. 6

Title:

To perform Image Segmentation using Thresholding Techniques.

Objective:

- To understand the concept and importance of image segmentation.
- To perform and compare various thresholding techniques for image segmentation.
- To implement Global Thresholding, Adaptive Thresholding, and Otsu's Binarization.
- To analyze the segmentation results for different images and methods.

.

Brief Theory:

Image segmentation is a process of partitioning a digital image into multiple segments to simplify and/or change the representation of an image into something more meaningful. One of the simplest and widely used segmentation methods is thresholding. Thresholding converts a grayscale image into a binary image based on a threshold value.

1. **Global Thresholding:**
    - Applies a single threshold value to the entire image.
    - If pixel intensity > threshold, set to max value; else set to 0.
2. **Adaptive Thresholding:**
    - Threshold value is calculated for smaller regions based on local image properties.
    - Two common methods: Adaptive Mean and Adaptive Gaussian.
3. **Otsu's Binarization:**
    - An automatic thresholding method that calculates the optimum threshold value by minimizing intra-class variance.
    - Effective for bimodal histograms.

Steps: -

Step 1: Import Necessary Libraries

Use OpenCV, NumPy for image processing.

Code: -

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image
import os
```

Step 2: Load the Image

- Read an image in grayscale using `cv2.imread()`.
- Ensure the image exists; otherwise, prompt the user.

Code: -

```
# Load the image (make sure 'image.jpg' is uploaded to the Colab session)
img_path = 'sample.jpg'

if not os.path.exists(img_path):
    print("Please upload 'image.jpg' to your Colab session.")
else:
    # Load in grayscale
    image = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
```

Step 3: Apply Thresholding:

- **Global Thresholding:** Use `cv2.threshold()` with a fixed value (e.g., 127).
- **Adaptive Mean Thresholding:** Use `cv2.adaptiveThreshold()` with `cv2.ADAPTIVE_THRESH_MEAN_C`.
- **Adaptive Gaussian Thresholding:** Use `cv2.adaptiveThreshold()` with `cv2.ADAPTIVE_THRESH_GAUSSIAN_C`.
- **Otsu's Binarization:** Use `cv2.threshold()` with `cv2.THRESH_OTSU`.

Code: -

```
 # 1. Global Thresholding
    ret1, thresh_global = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

    # 2. Adaptive Mean Thresholding
    thresh_adapt_mean = cv2.adaptiveThreshold(image, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2)

    # 3. Adaptive Gaussian Thresholding
    thresh_adapt_gauss = cv2.adaptiveThreshold(image, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

    # 4. Otsu's Binarization
    ret2, thresh_otsu = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
```

Step 4: Display Results:

- Use `matplotlib` to display the original and segmented images.
- Use subplotting to compare all results side by side.

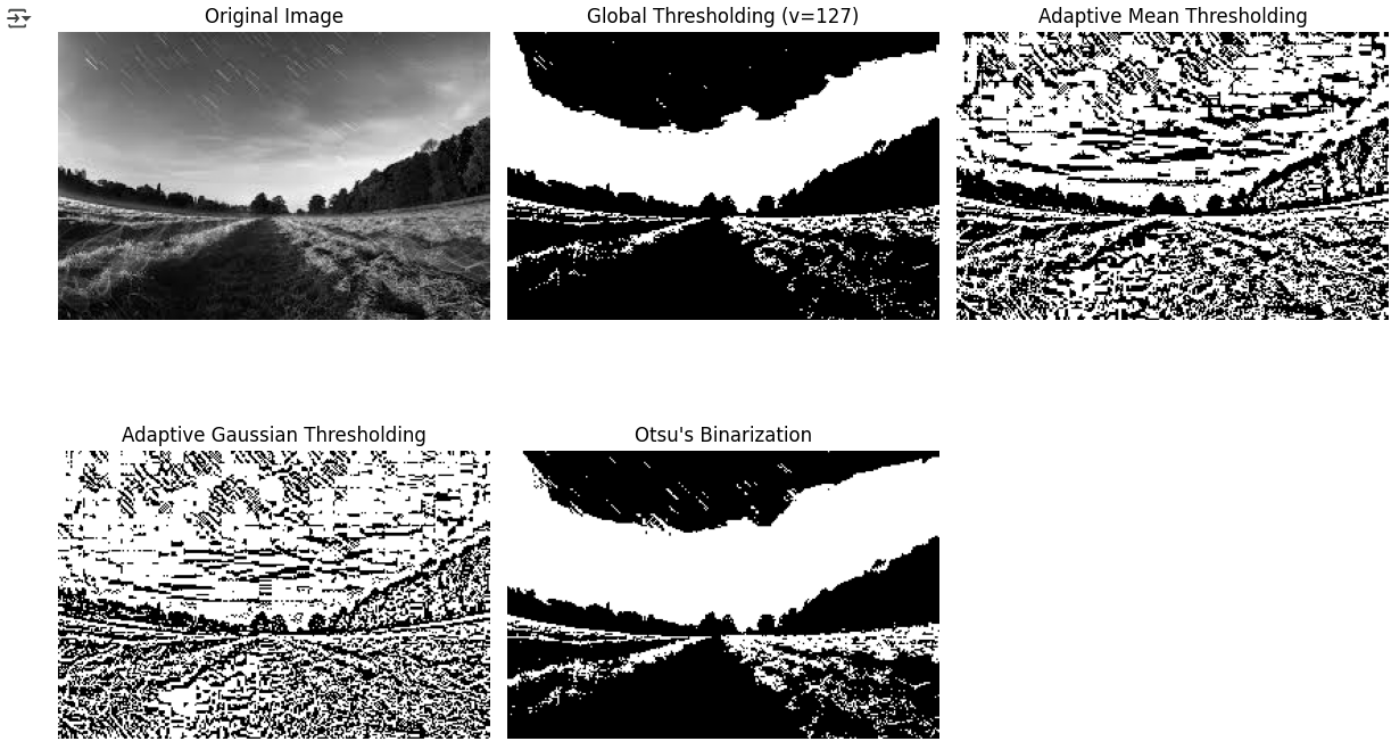
Code: -

```
# Display results
    titles = ['Original Image', 'Global Thresholding (v=127)',
              'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding',
              "Otsu's Binarization"]
    images = [image, thresh_global, thresh_adapt_mean,
              thresh_adapt_gauss, thresh_otsu]

    plt.figure(figsize=(12, 8))
    for i in range(5):
        plt.subplot(2, 3, i + 1)
        plt.imshow(images[i], cmap='gray')
        plt.title(titles[i])
        plt.axis('off')

    plt.tight_layout()
    plt.show()
```

Output:



Conclusion:

Thresholding techniques are effective for image segmentation, especially for binary classification of pixel values. Global thresholding is fast but less effective in varying lighting conditions. Adaptive methods handle uneven illumination better, and Otsu's method works well for images with bimodal histograms. The right choice of method depends on the image characteristics.

Practice Questions:

1. Implement global thresholding with different threshold values and observe the changes.
2. Use adaptive thresholding with different block sizes and constant values.
3. Visualize the histogram before applying Otsu's thresholding.
4. Combine Otsu's method with Gaussian filtering and observe improvements.
5. Perform thresholding on noisy images and compare results.
6. Apply all four techniques to different types of images (e.g., text, natural scene, medical image).

Expected Oral Questions

1. What is the goal of image segmentation?
2. How does global thresholding work?
3. What is the difference between adaptive mean and adaptive Gaussian thresholding?
4. Explain how Otsu's method determines the optimal threshold.
5. When should you prefer adaptive thresholding over global?
6. What are some limitations of thresholding techniques in segmentation?

FAQs in Interviews

1. **What is the difference between global and adaptive thresholding?**
   - Global uses a single threshold for the entire image, while adaptive calculates thresholds locally.
2. **Why is Otsu's method preferred in some scenarios?**
   - It automatically determines the optimal threshold value for bimodal histograms.
3. **How does image noise affect thresholding?**
   - Noise can cause incorrect thresholding; adaptive and Otsu’s methods help mitigate this.
4. **What preprocessing can improve thresholding results?**
   - Applying Gaussian blur or histogram equalization can improve results.
5. **Can thresholding be used for color images?**
   - Usually, it’s applied to grayscale; color images need channel-wise or converted processing.
6. **In what scenarios is thresholding not effective?**
   - In complex scenes with varying intensities or multi-class segmentation tasks, thresholding is limited.