CHRIST
(DEEMED TO BE UNIVERSITY)
BENGALURU · INDIA

# CAT1:IMAGE PROCESSING AND OBJECT DETECTION

**SANJAY R**
**2348055**

**25th November 2024**

**MDS573C**
**Image And Video Analytics**
**Department of Statistics and Data Science**

# CHRIST (Deemed to be University)

## Department of Statistics and Data Science

**Course: MDS573C - Image And Video Analytics** **CAT 1**

**R. SANJAY** **2348055**

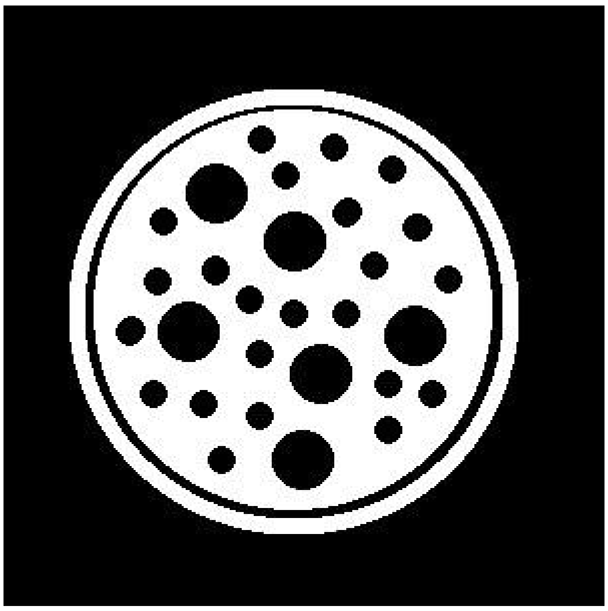
# *IMAGE PROCESSING AND OBJECT DETECTION*

***Download the image named 'Picture4A' uploaded in GCR and perform the following:***

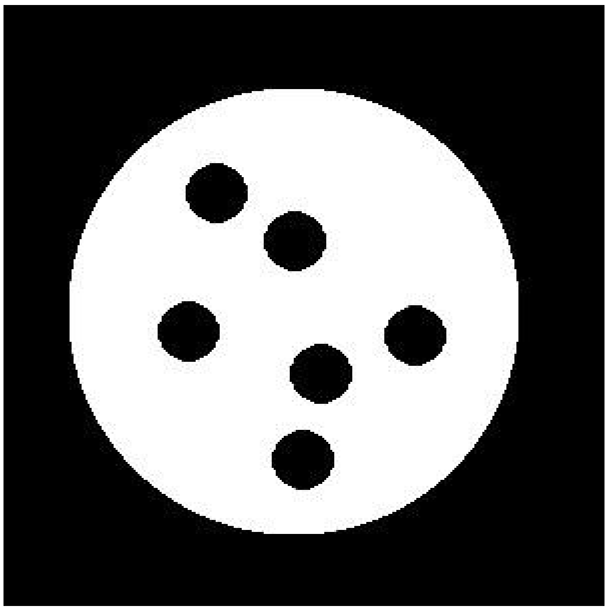
*i. Utilize appropriate image enhancement technique to improve the image quality.*
*ii. Isolate foreground objects from the background using segmentation.*
*iii. Perform region labeling and filter the connected components.*
*iv. Determine the total number of large dark circles and small dark circles present in the image.*
*v. The output image should be similar to 'Picture4B' uploaded in GCR.*

***Analysis and interpretation:*** *For each stage of the operations performed, specify the chosen technique, justify its selection, and analyze its impact on the output.*

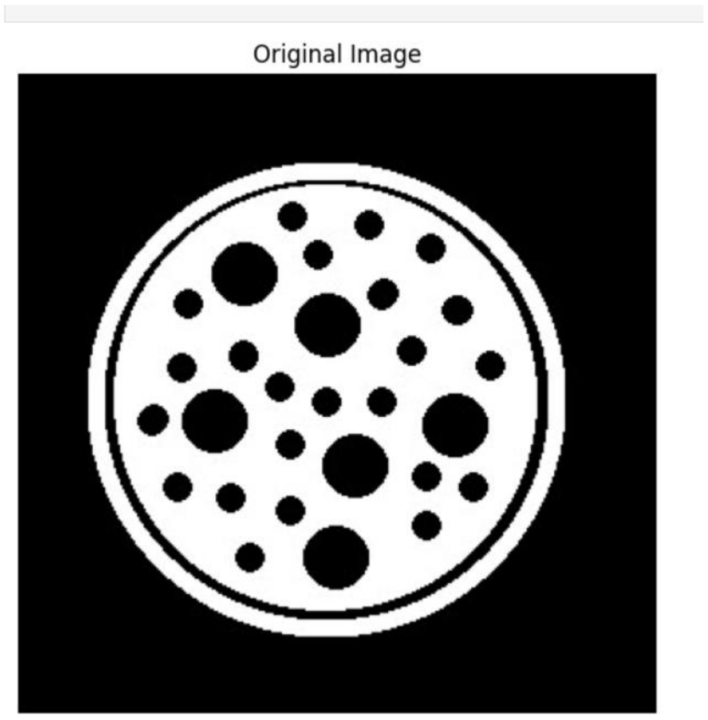
***Picture4A:***



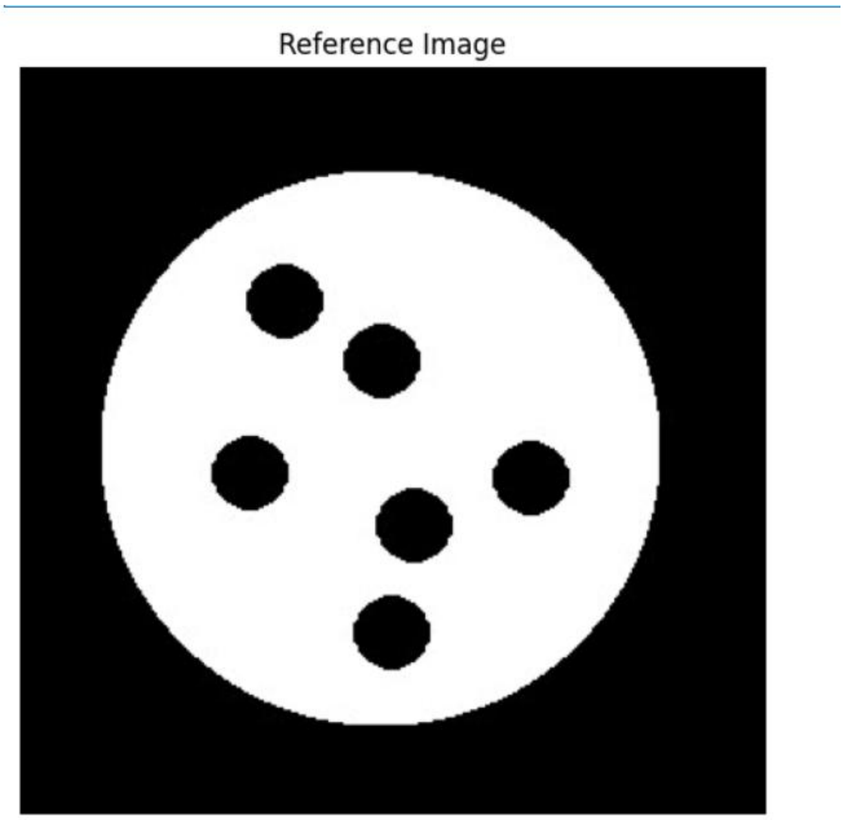
***Picture 4B:***



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load the original and reference images
image_path = 'Picture4A.jpg'
reference_image_path = 'Picture4B.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
# Display the original image
plt.figure(figsize=(8, 6))
plt.title("Original Image")
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.show()
```



```
reference_image = cv2.imread(reference_image_path,
cv2.IMREAD_GRAYSCALE)
# Display the reference image
plt.figure(figsize=(8, 6))
plt.title("Reference Image")
plt.imshow(reference_image, cmap='gray')
plt.axis('off')
plt.show()
```



## 1. Image Enhancement:

```
# Apply contrast stretching
min_val, max_val = np.min(image), np.max(image)
enhanced_image = ((image - min_val) / (max_val - min_val) *
255).astype('uint8')

# Apply Gaussian filtering to reduce noise
enhanced_image = cv2.GaussianBlur(enhanced_image, (5, 5), 0)

# Display original and enhanced images side-by-side
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title("Enhanced Image")
plt.imshow(enhanced_image, cmap='gray')
plt.axis('off')
plt.show()
```

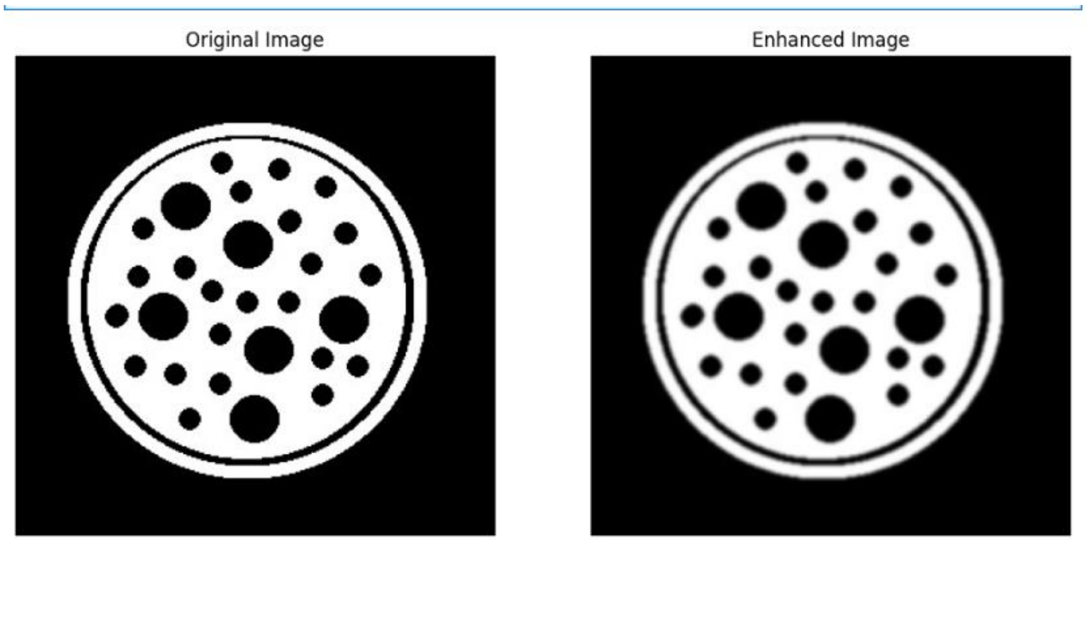
❖ ***Techniques chosen:***

✧ **Contrast Stretching:**

- ✓ **Purpose:** Enhances the dynamic range of pixel values, stretching them from the minimum and maximum values to cover the full range from 0 to 255.
- ✓ **Effect on Output:** Increases the contrast between the dark circles and the background, making the circles more distinct and visible. This helps improve feature differentiation for later processing.
- ✓ **Impact:** The circles are more clearly separated from the black background, ensuring better visibility for object detection and analysis.

✧ **Gaussian Filtering:**

- ✓ **Purpose:** Smooths the image to reduce noise, using a Gaussian kernel that preserves the edges of the circles while removing unwanted variations in pixel intensity.
- ✓ **Effect on Output:** The image appears smoother, with less noise, which prevents small, irrelevant details from interfering with the detection process.
- ✓ **Impact:** The edges of the dark circles are preserved, while noise is reduced, leading to clearer segmentation and better shape analysis.



➢ **Interpretations:**

✓ In the original image, the circles are visible but not clearly defined due to low contrast and some noise. In the enhanced image, after applying contrast stretching, the circles become more distinct with improved contrast, making them easier to identify against the black background.

✓ The original image displays circles with varying sizes but lacks sharpness in some areas, and some circles may be hard to distinguish. The enhanced image, by applying a Gaussian filter, reduces noise and smooths the image, maintaining the clarity of circle edges while eliminating unwanted details.

✓ The original image may contain background clutter and less pronounced circles, making object detection challenging. The enhanced image, due to better contrast and noise reduction, presents the circles with clear definition, improving the efficiency of tasks like object detection or circle counting.

## 2. Segmentation:

```
# Apply adaptive thresholding
segmented_image = cv2.adaptiveThreshold(
   enhanced_image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, 11, 2)

# Display original and segmented images side-by-side
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title("Segmented Image")
plt.imshow(segmented_image, cmap='gray')
plt.axis('off')
plt.show()
```

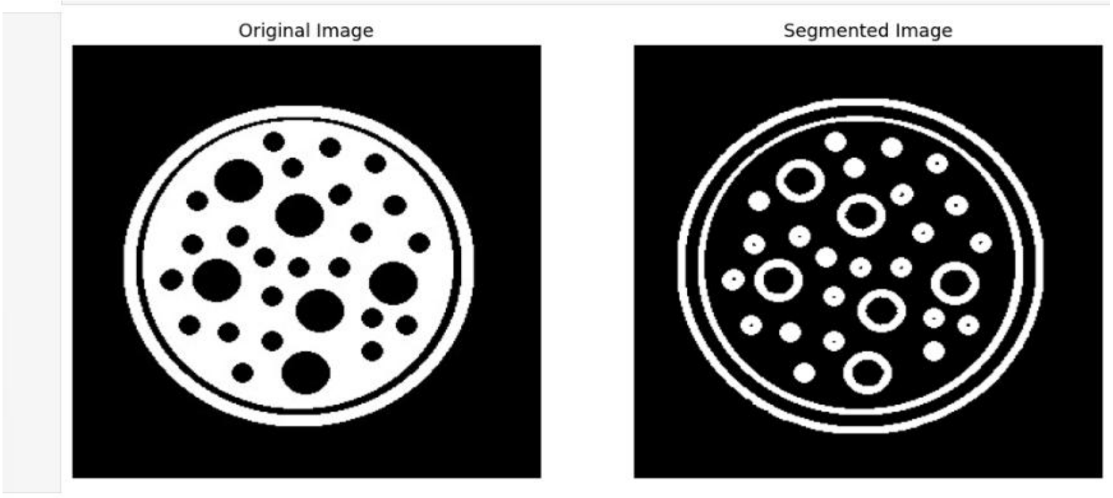
➢ **Techniques Chosen:**

✧ **Adaptive Thresholding:** Adaptive thresholding was applied to segment the image.

**Justification:** This technique is ideal for handling images with varying lighting conditions, as it adjusts the threshold dynamically based on local regions.
**Impact on Output:** The adaptive thresholding makes the dark circles stand out against the background, even if there are varying levels of illumination across the image.

✧ **Binary Inversion:**

**Chosen Technique:** The binary inversion option was used.
**Justification**: By inverting the thresholding, the dark circles become foreground objects, which simplifies further processing steps such as contour detection or object counting.
**Impact on Output:** The dark circles are now represented in white, which enhances their visibility and ensures they are correctly identified as objects to be analyzed.



➢ **Interpretations:**

✓ The segmented image (right) clearly isolates the dark circles (foreground) from the background, with the circles in white against a black background, making them more distinct than in the original image (left).

✓ The adaptive thresholding method has improved the visibility of the circles by handling varying lighting conditions, ensuring better local contrast and allowing for better separation between the foreground and the background.
✓ The use of binary inversion in the thresholding process has made the dark circles stand out as white, enhancing their contrast with the black background and simplifying the identification of objects in the image.

- ✓ In the original image, the circles and background are less clearly defined due to noise and variations in lighting. After segmentation, the circles are much more easily distinguishable from the background, facilitating further analysis.

- ✓ This segmentation step successfully prepares the image for the next phase of connected component analysis, where the isolated circles will be counted and analyzed, aiding in the accurate detection of the objects of interest.

## 3. Connected Components Analysis:

```
num_labels, labels, stats, _ =
cv2.connectedComponentsWithStats(segmented_image)

# Create an output image to visualize results
output_image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)

# Initialize counters for circles
large_circles = 0
small_circles = 0

# Define size thresholds for circle classification
small_threshold = 100
large_threshold_min = 180
large_threshold_max = 450  # Exclude regions larger than this threshold

# Loop through the connected components
for i in range(1, num_labels):  # Skip the background (label 0)
    area = stats[i, cv2.CC_STAT_AREA]
    x, y, w, h = stats[i, cv2.CC_STAT_LEFT], stats[i, cv2.CC_STAT_TOP],
stats[i, cv2.CC_STAT_WIDTH], stats[i, cv2.CC_STAT_HEIGHT]

    if large_threshold_min <= area <= large_threshold_max:
        large_circles += 1
        # Draw a green bounding box for large circles
        cv2.rectangle(output_image, (x, y), (x + w, y + h), (0, 255, 0), 2)
    elif small_threshold < area < large_threshold_min:
        small_circles += 1
        # Draw a blue bounding box for small circles
        cv2.rectangle(output_image, (x, y), (x + w, y + h), (255, 0, 0), 2)
```

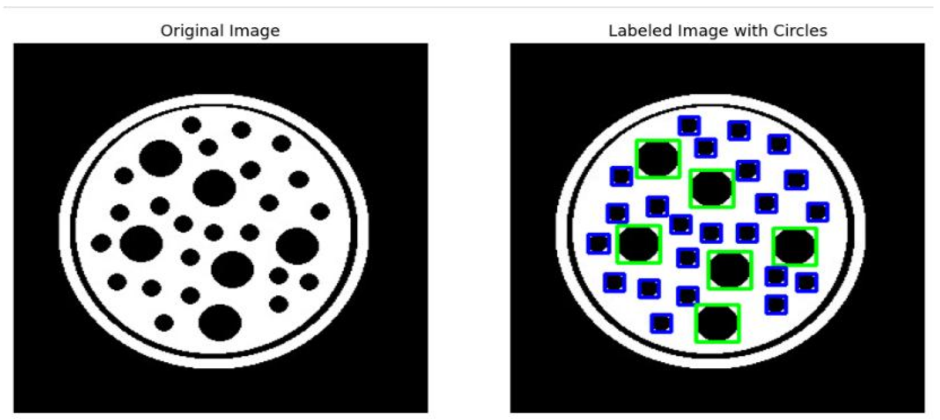
```
# Display the original and labeled images side-by-side
plt.figure(figsize=(12, 8))
# Original Images
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image, cmap='gray')
plt.axis('off')

# Labeled Image
plt.subplot(1, 2, 2)
plt.title("Labeled Image with Circles")
plt.imshow(cv2.cvtColor(output_image, cv2.COLOR_BGR2RGB))
plt.axis('off')
# Add a legend to explain the color coding
plt.figtext(0.5, 0.01, 'Green = Large Circles, Blue = Small Circles',
ha='center', va='center', fontsize=12, color='black')
plt.show()
```

**Chosen Technique: Connected Components Analysis:**

- ✓ Connected components analysis was applied to segment and label the distinct circles in the image based on their connected regions of white pixels.

- ✓ It is effective for detecting distinct objects in binary images, especially when the objects are separated by contrasting regions, as in this case. It helps in classifying and labeling regions based on their size, which is crucial when distinguishing between large and small circles. This makes the technique suitable for the task of differentiating the circles in the image based on area.

- ✓ The output image labels large circles with green boxes and small circles with blue boxes, making it easy to distinguish them by size, which was not possible in the original image.



Green = Large Circles, Blue = Small Circles

- **Interpretaions:**

✓ Compared to the original image, the labeled image on the right offers more detailed information, categorizing the circles into large and small based on size.

✓ The labeling is accurate, with large circles falling within the specified area range (180-450 pixels) marked in green, and small circles (between 100-180 pixels) marked in blue, ensuring clear differentiation.

## 4. Determine the total number of large dark circles and small dark circles present in the image.

```
# Print the counts of large and small circles
print(f"Number of Large Circles: {large_circles}")
print(f"Number of Small Circles: {small_circles}")
```

✧ **Output:**

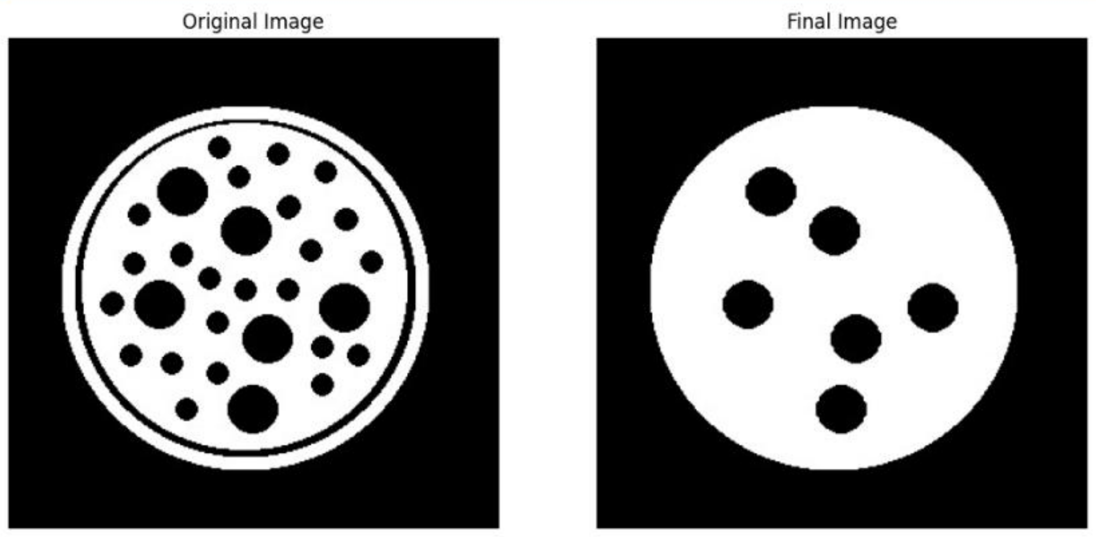
```
Number of Large Circles: 6
Number of Small Circles: 23
```

**5. The output image should be similar to 'Picture4B' uploaded in GCR.**

```
# Create a copy of the original image to modify
final_image = image.copy()

# Loop through the connected components again to identify small circles
for i in range(1, num_labels):  # Skip the background (label 0)
    area = stats[i, cv2.CC_STAT_AREA]
    x, y, w, h = stats[i, cv2.CC_STAT_LEFT], stats[i, cv2.CC_STAT_TOP],
stats[i, cv2.CC_STAT_WIDTH], stats[i, cv2.CC_STAT_HEIGHT]

    # Check if the area is below the small threshold
    if small_threshold < area < large_threshold_min:
        # Set the pixels within the small circle bounding box to white in the
final image
        final_image[y:y + h, x:x + w] = 255  # Replace with white (255 for
grayscale)


# Display the original and final images side-by-side
plt.figure(figsize=(12, 6))
# Original Image
plt.subplot(1, 2, 1)

plt.title("Original Image")
plt.imshow(image, cmap='gray')
plt.axis('off')

# Final Image
plt.subplot(1, 2, 2)
plt.title("Final Image")
plt.imshow(reference_image, cmap='gray')
plt.axis('off')
plt.show()
plt.show()
```

- ✓ A copy of the original image was made to preserve the input data while allowing modifications on the final result.

- ✓ Connected components analysis segmented the image into distinct regions, assigning each connected group of pixels a unique label.
- ✓ The area of each component was calculated to differentiate between large circles, small circles, and noise.
- ✓ Regions classified as small circles (falling within a specific area threshold) and the boundary line were identified and removed by replacing their bounding boxes with white pixels.
- ✓ Larger circles that did not meet the removal criteria were retained, simplifying the image to highlight only prominent features.
- ✓ The resulting image effectively removes unnecessary noise and smaller shapes, isolating the main objects (larger circles).



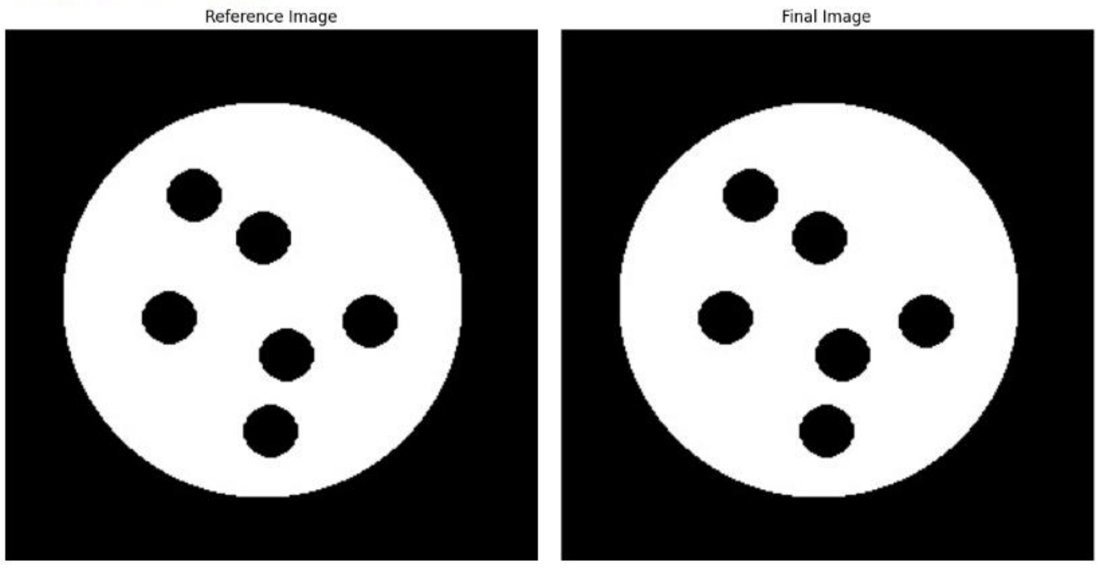
✧ **Interpretation:**

- ✓ The **original image** contains numerous circles of varying sizes, including noise and a boundary.
- ✓ The **final image** shows a clean representation, with only the significant large circles retained. Small circles and the circular boundary have been eliminated, simplifying the image for further tasks like object recognition or quantitative analysis.

- ✓ **Checking if the reference and final images have the same dimensions:**

```
if reference_image.shape == final_image.shape:
    difference = np.abs(reference_image - final_image)
    # Check if all pixel values are the same
    if np.all(difference == 0):
        print("The images are identical.")
    else:
        print("The images are different.")
        # Optional: Show the differences visually
        plt.figure(figsize=(8, 8))
        plt.title("Differences between Images")
        plt.imshow(difference, cmap='gray')
        plt.axis('off')
        plt.show()
else:
    print("The images have different dimensions and cannot be compared
directly.")


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

# Reference Image
plt.subplot(1, 2, 1)
plt.title("Reference Image")
plt.imshow(reference_image, cmap='gray')
plt.axis('off')

# Final Image
plt.subplot(1, 2, 2)
plt.title("Final Image")
plt.imshow(final_image, cmap='gray')
plt.axis('off')

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

```
The images are identical.
```



**Interpretaions:**

- ✓ The technique compares two images pixel by pixel using absolute differences to identify variations.
- ✓ The condition np.all(difference == 0) ensures that the comparison is strict, checking if all pixel values are identical across the images.
- ✓ If the images are identical, as indicated in the output, it confirms no discrepancies between the reference and final images in terms of pixel intensities.

**We observe that the 2 images are identical.**