1. **Read image, Display it using imshow and matplotlib.**

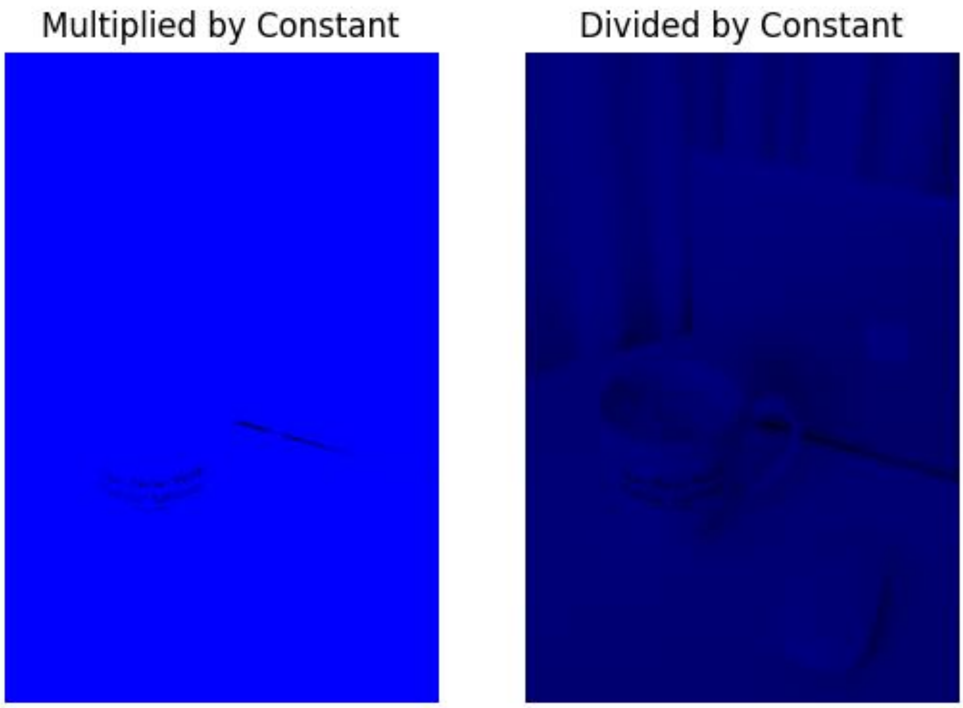
```
import cv2
import matplotlib.pyplot as plt

# Read the image
image_path = 'C:/Users/rcc/Downloads/pexels.jpg'
img = cv2.imread(image_path)
if img is None:
    print("Error: Image not found. Check the path.")
else:
    # Convert to RGB
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Display
    plt.imshow(img_rgb)
    plt.title("Original Image")
    plt.axis("off")
    plt.show()
```

Perform functions:

- **Implement multiplication and division by constant function.**

```
import numpy as np

# Multiplication and division
img_mul = cv2.multiply(img, np.array([2.0]))  # Brighten image
img_div = cv2.divide(img, np.array([2.0]))    # Dim image
```

```
# Display results
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img_mul, cv2.COLOR_BGR2RGB))
plt.title("Multiplied by Constant")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(img_div, cv2.COLOR_BGR2RGB))
plt.title("Divided by Constant")
plt.axis("off")

plt.show()
```



- **Resize image**

```
# Resize the image
img_resized = cv2.resize(img, (200, 200))

# Display the resized image
plt.imshow(cv2.cvtColor(img_resized, cv2.COLOR_BGR2RGB))
plt.title("Resized Image (200x200)")
plt.axis("off")
plt.show()
```

Resized Image (200x200)

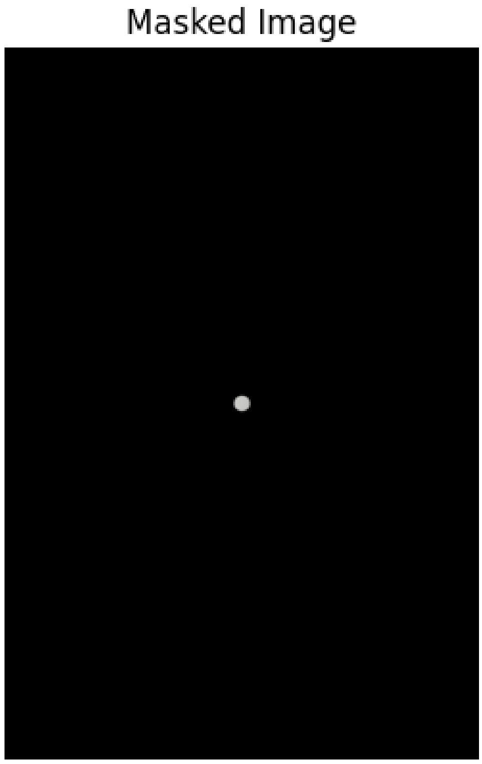
- **Slice part of image**

```
# Slice a part of the image (e.g., top-left corner)
img_sliced = img[:100, :100]

# Display the sliced part
plt.imshow(cv2.cvtColor(img_sliced, cv2.COLOR_BGR2RGB))
plt.title("Sliced Image (Top-left 100x100)")
plt.axis("off")
plt.show()
```

Sliced Image (Top-left 100x100)

- **Mask a part of image**

```
mask = np.zeros_like(img, dtype=np.uint8)
center = (img.shape[1] // 2, img.shape[0] // 2)
radius = 50
cv2.circle(mask, center, radius, (255, 255, 255), -1)
img_masked = cv2.bitwise_and(img, mask)

plt.imshow(cv2.cvtColor(img_masked, cv2.COLOR_BGR2RGB))
plt.title("Masked Image")
plt.axis("off")
plt.show()
```



- **Add two images**

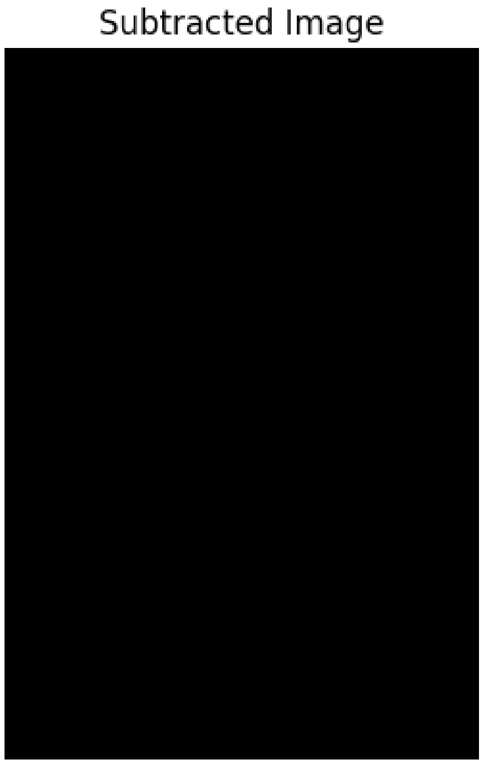
```
# Adding the image to itself
img_add = cv2.add(img, img)

plt.imshow(cv2.cvtColor(img_add, cv2.COLOR_BGR2RGB))
plt.title("Added Image")
plt.axis("off")
plt.show()
```

- **Weighted add two SWimages**
  # Weighted addition
  img_weighted = cv2.addWeighted(img, 0.7, img, 0.3, 0)

  plt.imshow(cv2.cvtColor(img_weighted, cv2.COLOR_BGR2RGB))
  plt.title("Weighted Added Image")
  plt.axis("off")
  plt.show()

- **Subtract two image**
  # Subtract the image from itself
  img_subtract = cv2.subtract(img, img)

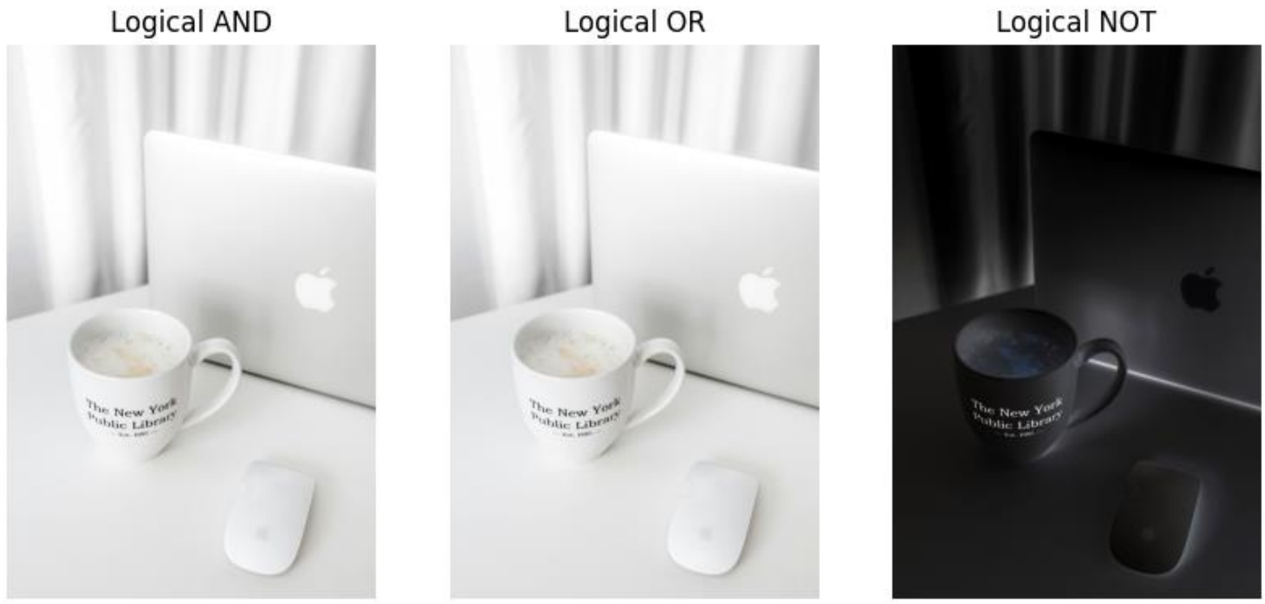
```
plt.imshow(cv2.cvtColor(img_subtract, cv2.COLOR_BGR2RGB))
plt.title("Subtracted Image")
plt.axis("off")
plt.show()
```



- **Perform logical AND, OR , NOT on image.**

```
# Logical AND, OR, NOT
img_and = cv2.bitwise_and(img, img)
img_or = cv2.bitwise_or(img, img)
img_not = cv2.bitwise_not(img)

# Display logical operations
plt.figure(figsize=(10, 5))

plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(img_and, cv2.COLOR_BGR2RGB))
plt.title("Logical AND")
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(img_or, cv2.COLOR_BGR2RGB))
plt.title("Logical OR")
plt.axis("off")

plt.subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(img_not, cv2.COLOR_BGR2RGB))
```

```
plt.title("Logical NOT")
plt.axis("off")

plt.show()
```

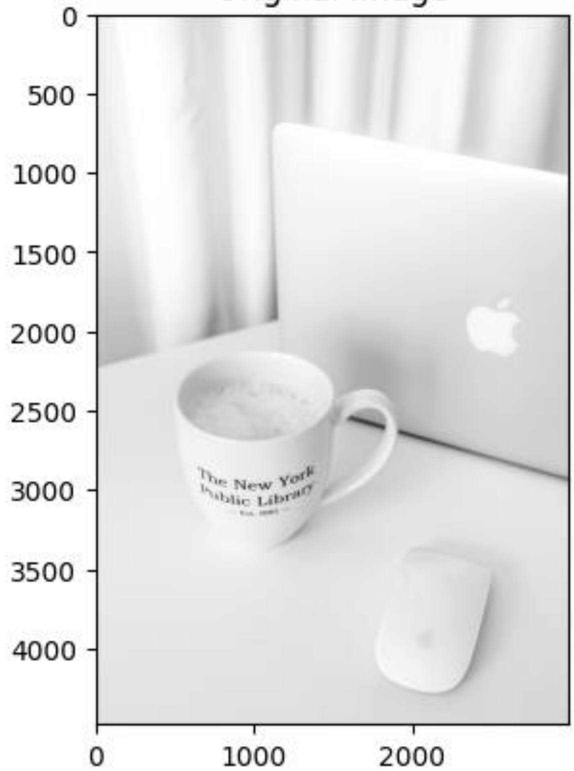


Mark four neighbors and eight neighbors of any pixel in the image, implement distance formula, implement image negation, log transformation, and power log Transformation.

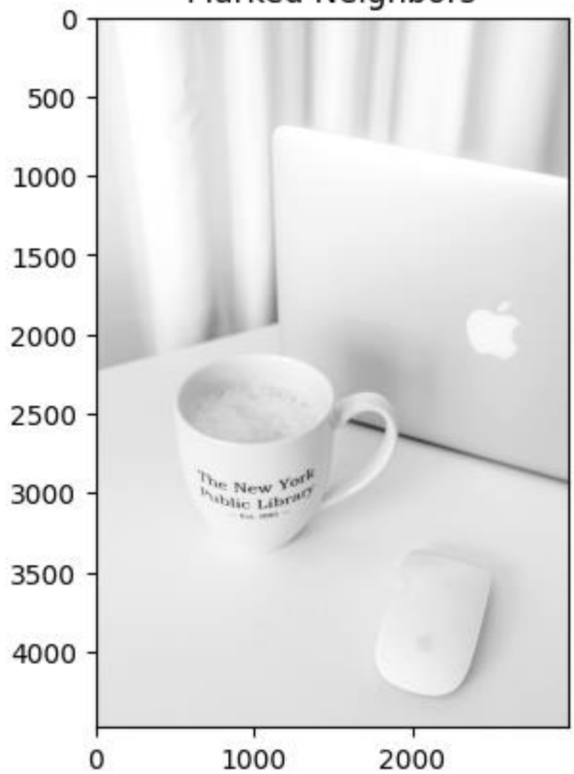
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('C:/Users/rc/Downloads/pexels.jpg', cv2.IMREAD_GRAYSCALE)
plt.imshow(image, cmap='gray')
plt.title("Original Image")
plt.show()
# 1. Mark four neighbors and eight neighbors of a pixel
def mark_neighbors(image, x, y):
    marked_image = image.copy()
    rows, cols = image.shape
    # Ensure x, y are within bounds
    if x < 0 or x >= rows or y < 0 or y >= cols:
        raise ValueError("Coordinates out of bounds")
    # Four neighbors
    neighbors_4 = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
    # Eight neighbors
    neighbors_8 = neighbors_4 + [(x-1, y-1), (x-1, y+1), (x+1, y-1), (x+1, y+1)]
    for nx, ny in neighbors_8:
        if 0 <= nx < rows and 0 <= ny < cols:
            marked_image[nx, ny] = 255  # Highlight neighbors in white
```

```
    return marked_image
x, y = 50, 50  # Example pixel
marked_neighbors = mark_neighbors(image, x, y)
plt.imshow(marked_neighbors, cmap='gray')
plt.title("Marked Neighbors")
plt.show()
# 2. Implementing distance formula
def distance_formula(p1, p2):
    return np.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)
point1 = (x, y)
point2 = (70, 70)
distance = distance_formula(point1, point2)
print(f"Distance between {point1} and {point2}: {distance}")
# 3. Image Negation
image_negation = 255 - image
plt.imshow(image_negation, cmap='gray')
plt.title("Image Negation")
plt.show()
# 4. Log Transformation
def log_transform(image, c=1):
    return (c * np.log1p(image)).astype(np.uint8)
log_transformed = log_transform(image, c=45)
plt.imshow(log_transformed, cmap='gray')
plt.title("Log Transformation")
plt.show()
# 5. Power-Log (Gamma) Transformation
def power_log_transform(image, gamma, c=1):
    normalized = image / 255.0  # Normalize to range [0,1]
    transformed = c * (normalized ** gamma)
    return (transformed * 255).astype(np.uint8)
gamma = 2.2
power_log_transformed = power_log_transform(image, gamma, c=1)
plt.imshow(power_log_transformed, cmap='gray')
plt.title("Power-Log Transformation")
plt.show()
```

Original Image

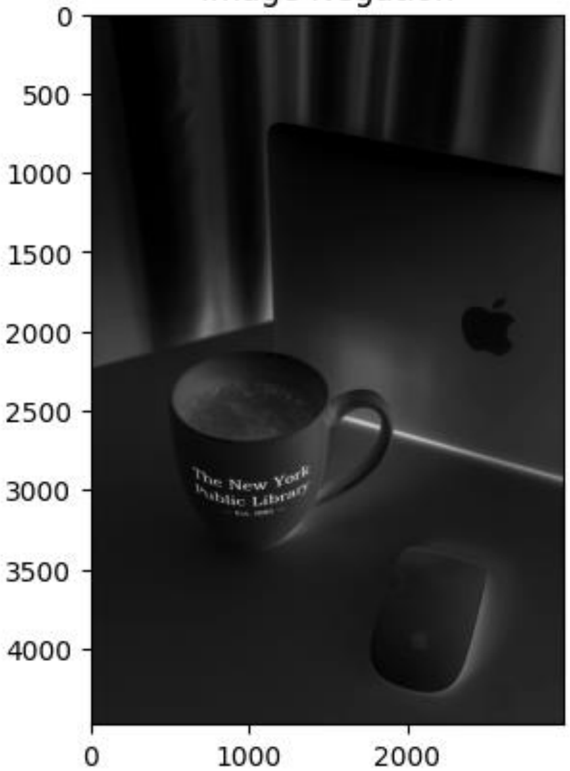


Marked Neighbors

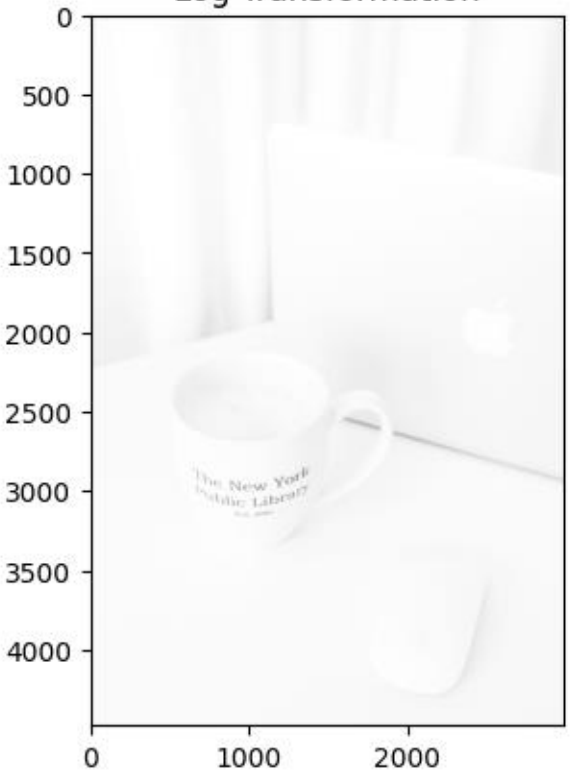


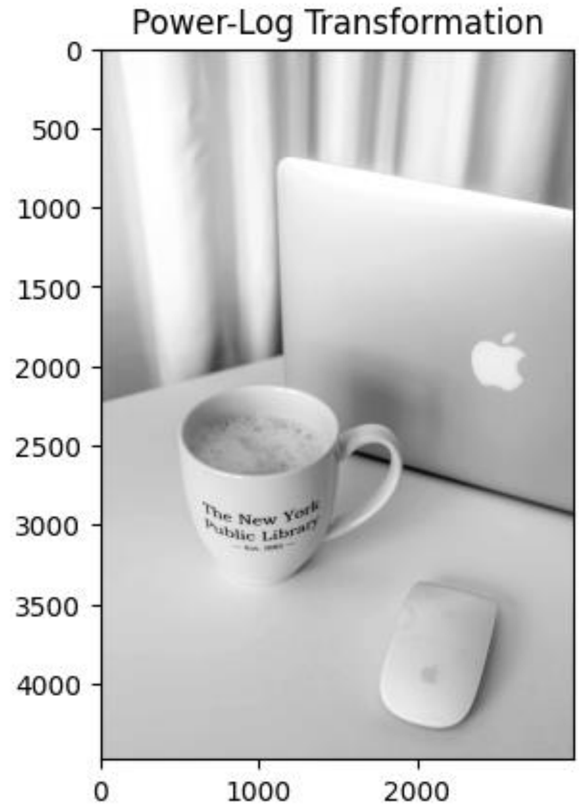
Distance between (50, 50) and (70, 70): 28.284271247461902

Image Negation



Log Transformation



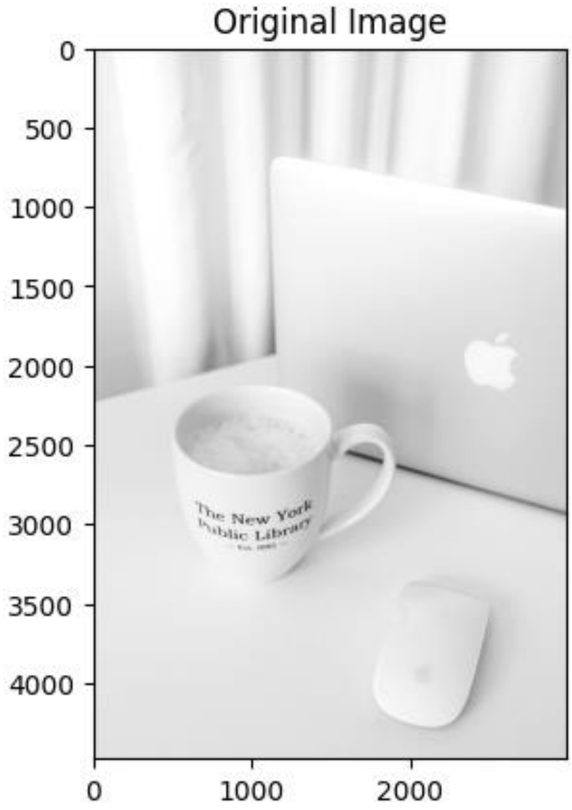


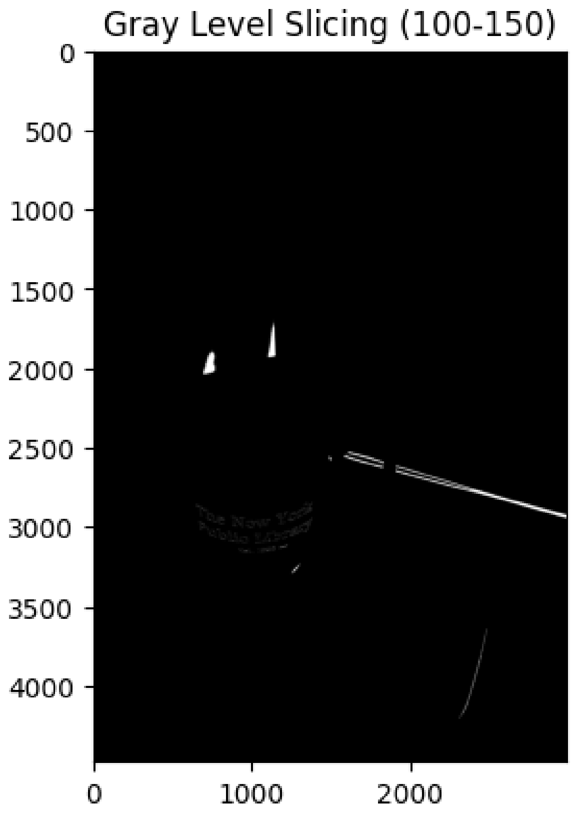
Gray level Slicing, Intensity slicing, Boxplot slicing, Bit plane slicing, Histogram Equalizer

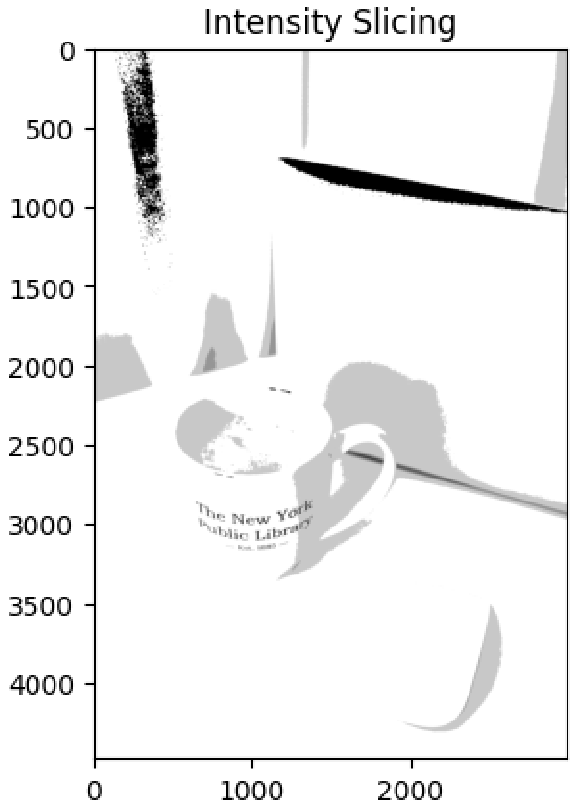
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the grayscale image
image = cv2.imread('C:/Users/rcc/Downloads/pexels.jpg', cv2.IMREAD_GRAYSCALE)

# Display original image
plt.imshow(image, cmap='gray')
plt.title("Original Image")
plt.show()

# 1. Gray Level Slicing
def gray_level_slicing(image, lower, upper, highlight_value=255, background_value=0):
    sliced_image = np.where((image >= lower) & (image <= upper), highlight_value, background_value)
    return sliced_image.astype(np.uint8)

gray_sliced = gray_level_slicing(image, lower=100, upper=150)
plt.imshow(gray_sliced, cmap='gray')
plt.title("Gray Level Slicing (100-150)")
plt.show()

# 2. Intensity Slicing
def intensity_slicing(image, levels):
    sliced_image = np.zeros_like(image)
    interval = 256 // len(levels)
```

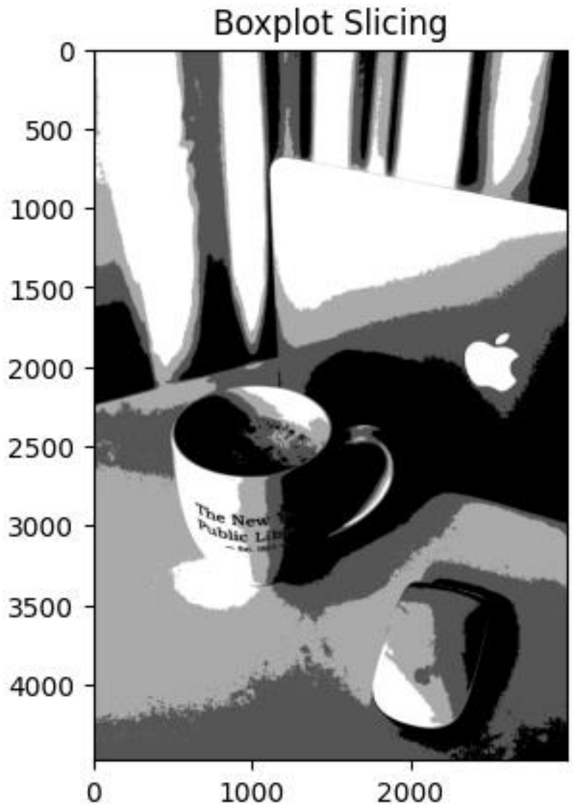
```
    for i, value in enumerate(levels):
        lower = i * interval
        upper = lower + interval
        sliced_image[(image >= lower) & (image < upper)] = value
    return sliced_image

levels = [50, 100, 150, 200, 255]  # Example intensity levels
intensity_sliced = intensity_slicing(image, levels)
plt.imshow(intensity_sliced, cmap='gray')
plt.title("Intensity Slicing")
plt.show()

# 3. Boxplot Slicing
def boxplot_slicing(image):
    boxplot_values = np.percentile(image, [25, 50, 75])  # Calculate Q1, Median, Q3
    sliced_image = np.where(image < boxplot_values[0], 0,
                np.where(image < boxplot_values[1], 85,
                np.where(image < boxplot_values[2], 170, 255)))
    return sliced_image.astype(np.uint8)

boxplot_sliced = boxplot_slicing(image)
plt.imshow(boxplot_sliced, cmap='gray')
plt.title("Boxplot Slicing")
plt.show()

# 4. Bit Plane Slicing
def bit_plane_slicing(image, bit_level):
    return ((image >> bit_level) & 1) * 255

# Displaying all 8 bit planes
bit_planes = [bit_plane_slicing(image, i) for i in range(8)]

fig, axes = plt.subplots(2, 4, figsize=(15, 6))
for i, ax in enumerate(axes.flatten()):
    ax.imshow(bit_planes[i], cmap='gray')
    ax.set_title(f'Bit Plane {i}')
    ax.axis('off')
plt.tight_layout()
plt.show()

# 5. Histogram Equalization
equalized_image = cv2.equalizeHist(image)
plt.imshow(equalized_image, cmap='gray')
plt.title("Histogram Equalization")
plt.show()

plt.figure(figsize=(10, 5))
plt.hist(image.ravel(), bins=256, range=(0, 256), alpha=0.5, label='Original')
plt.hist(equalized_image.ravel(), bins=256, range=(0, 256), alpha=0.5, label='Equalized')
```

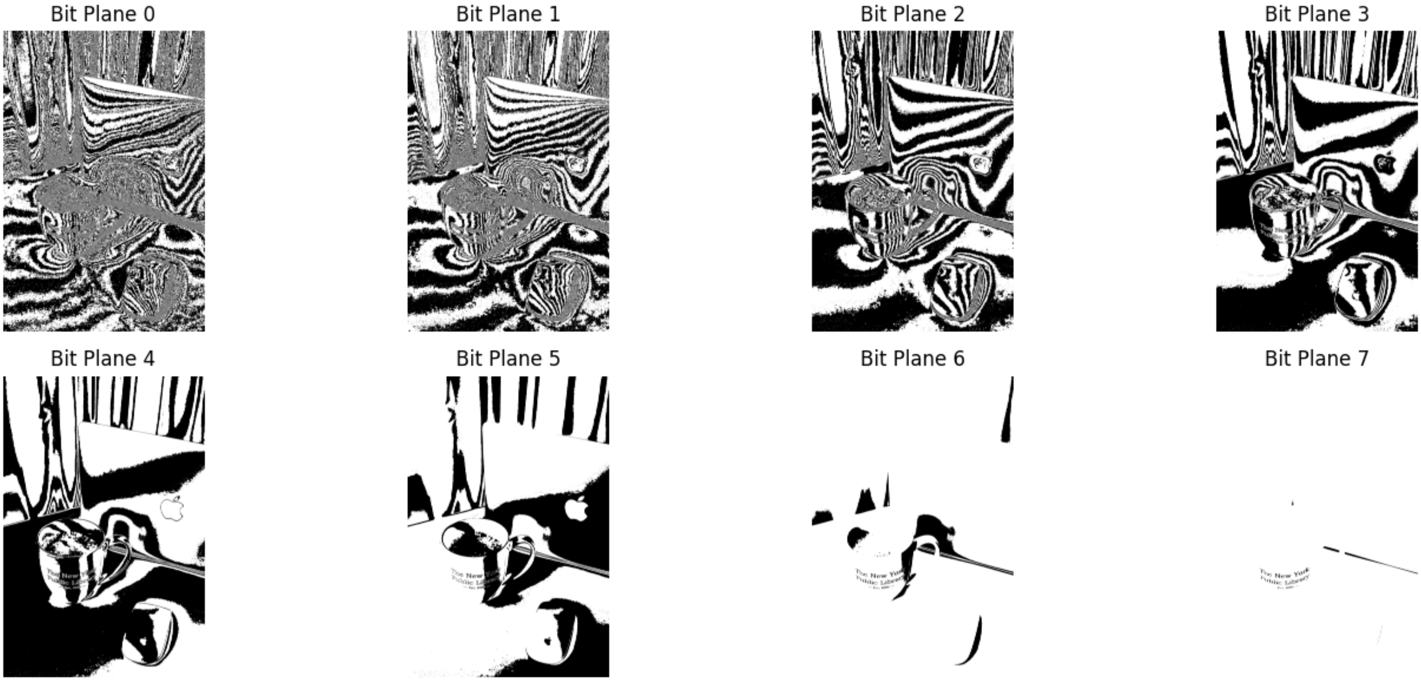
```
plt.title("Histogram Comparison")
plt.legend()
plt.show()
```

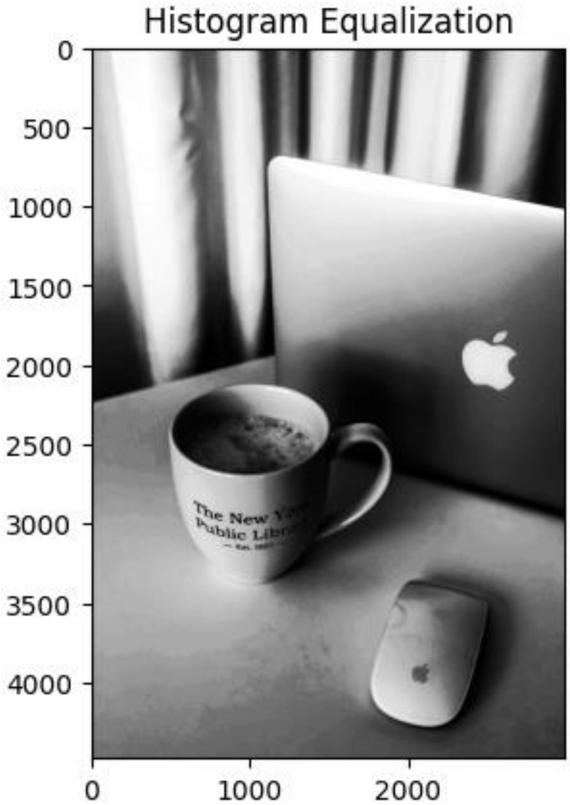


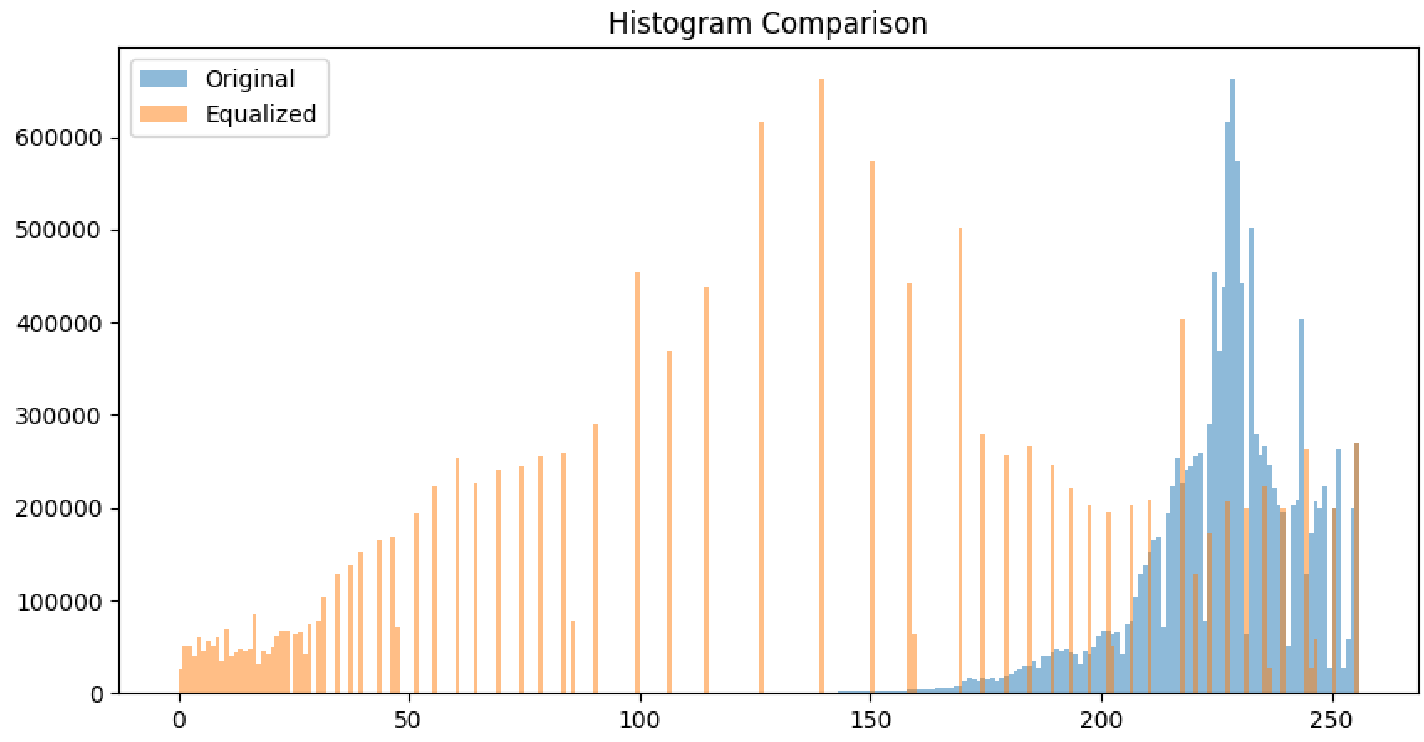



Intensity Slicing
0
500
1000
1500
2000
2500
3000
3500
4000
0
1000
2000
The New York
Public Library


Boxplot Slicing
0
500
1000
1500
2000
2500
3000
3500
4000
0
1000
2000


Bit Plane 0
Bit Plane 1
Bit Plane 2
Bit Plane 3
Bit Plane 4
Bit Plane 5
Bit Plane 6
Bit Plane 7


Histogram Equalization
0
500
1000
1500
2000
2500
3000
3500
4000
0
1000
2000
The New York
Public Library




```
import urllib.request

url = 'https://raw.githubusercontent.com/opencv/opencv/master/samples/data/lena.jpg'
urllib.request.urlretrieve(url, 'lena.jpg')
```

```
('lena.jpg', <http.client.HTTPMessage at 0x17b016063c0>)
```

Box filters, filter 2D, Gaussian Blur, and Median Blur, & write a program to hardcode convolution using a kernel of known size & value

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)

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

# 1. Box Filter
box_filter = cv2.blur(image, (5, 5))  # Kernel size: 5x5
plt.imshow(box_filter, cmap='gray')
```

```
plt.title("Box Filter (5x5)")
plt.axis('off')
plt.show()

# 2. Filter 2D
kernel = np.array([[1, 1, 1],
                   [1, 1, 1],
                   [1, 1, 1]], dtype=np.float32) / 9  # 3x3 averaging kernel
filter_2d = cv2.filter2D(image, -1, kernel)
plt.imshow(filter_2d, cmap='gray')
plt.title("2D Filter (3x3 Averaging)")
plt.axis('off')
plt.show()

# 3. Gaussian Blur
gaussian_blur = cv2.GaussianBlur(image, (5, 5), sigmaX=1)
plt.imshow(gaussian_blur, cmap='gray')
plt.title("Gaussian Blur (5x5, sigma=1)")
plt.axis('off')
plt.show()

# 4. Median Blur
median_blur = cv2.medianBlur(image, 5)  # Kernel size: 5
plt.imshow(median_blur, cmap='gray')
plt.title("Median Blur (5x5)")
plt.axis('off')
plt.show()

# 5. Hardcoded Convolution
def hardcoded_convolution(image, kernel):
    kernel_height, kernel_width = kernel.shape
    pad_h, pad_w = kernel_height // 2, kernel_width // 2

    # Pad the image with zeros
    padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)), mode='constant',
constant_values=0)

    # Prepare output image
    output = np.zeros_like(image)

    # Perform convolution
    for i in range(image.shape[0]):
```

```
        for j in range(image.shape[1]):
            region = padded_image[i:i+kernel_height, j:j+kernel_width]
            output[i, j] = np.sum(region * kernel)

    # Normalize output to range 0-255
    output = np.clip(output, 0, 255).astype(np.uint8)
    return output

# Example kernel for convolution
custom_kernel = np.array([[0, -1, 0],
                          [-1, 5, -1],
                          [0, -1, 0]])  # Example sharpening kernel
custom_convolution = hardcoded_convolution(image, custom_kernel)
plt.imshow(custom_convolution, cmap='gray')
plt.title("Custom Convolution (Sharpening)")
plt.axis('off')
plt.show()
```

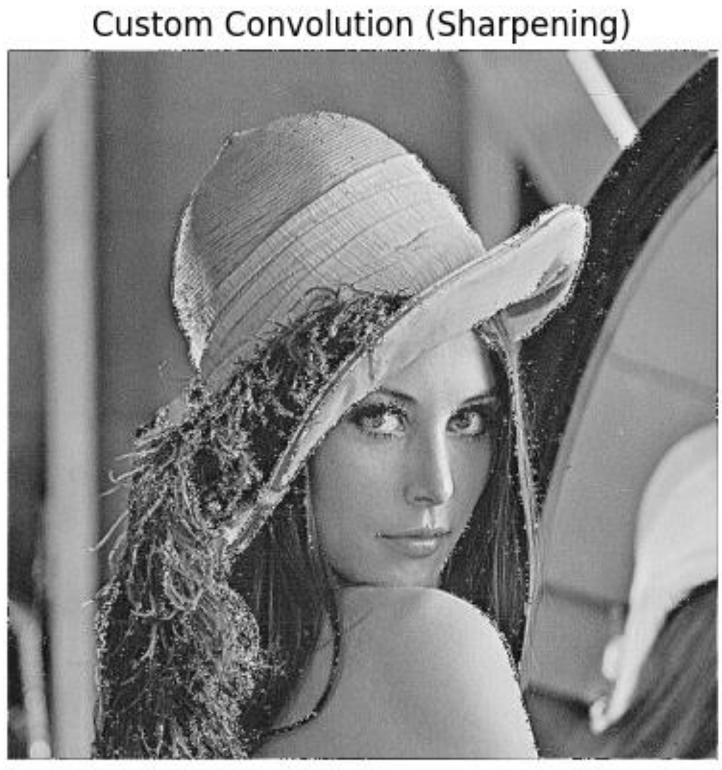
Original Image

Box Filter (5x5)

2D Filter (3x3 Averaging)

Gaussian Blur (5x5, sigma=1)

Median Blur (5x5)



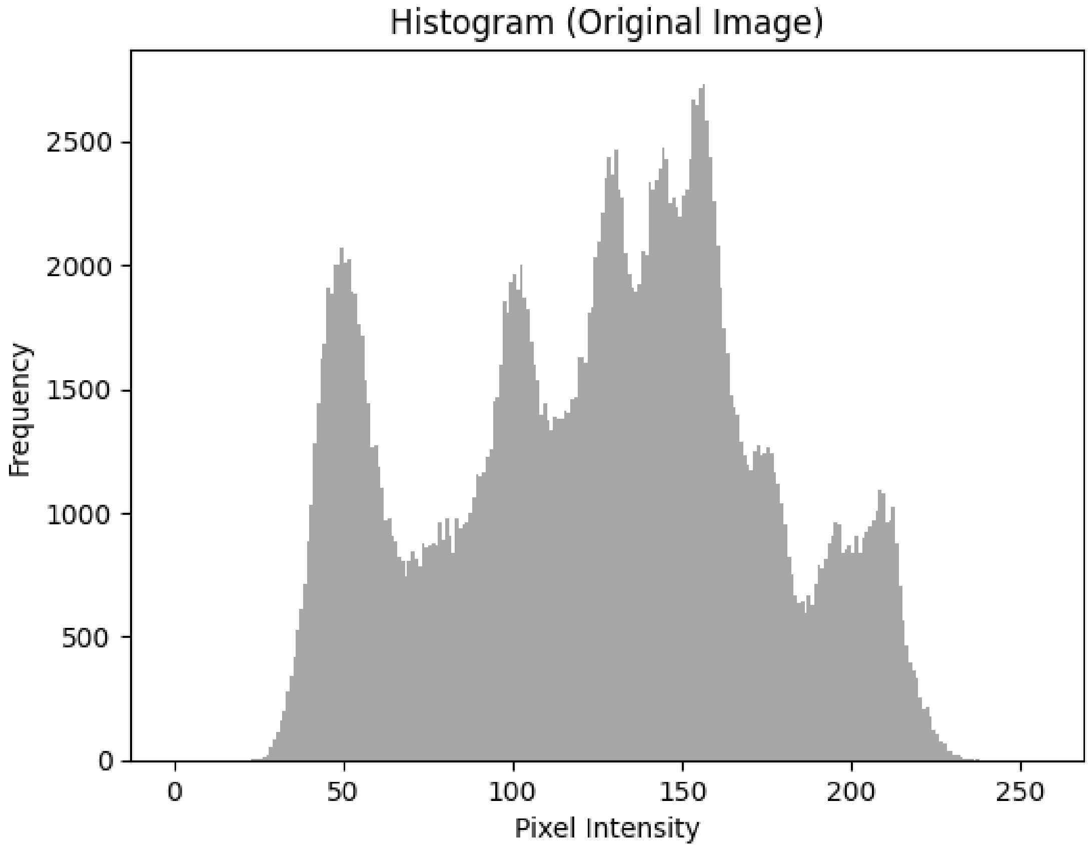
Plot the histogram after and before the effects(whatever effects are given above)

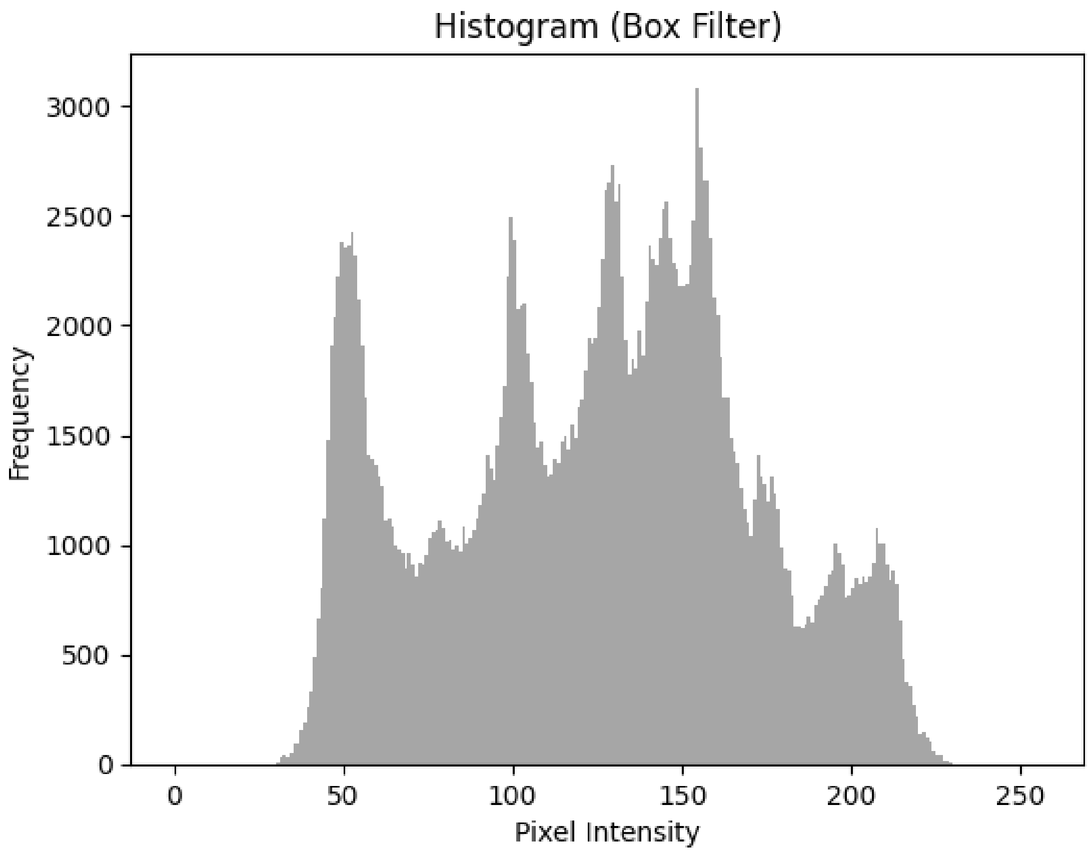
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the grayscale image
image = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)

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

# Function to plot histogram
def plot_histogram(image, title):
    plt.hist(image.ravel(), bins=256, range=(0, 256), color='gray', alpha=0.7)
    plt.title(title)
    plt.xlabel("Pixel Intensity")
    plt.ylabel("Frequency")
    plt.show()

# Plot original histogram
plot_histogram(image, "Histogram (Original Image)")

# 1. Box Filter
box_filter = cv2.blur(image, (5, 5))
plot_histogram(box_filter, "Histogram (Box Filter)")
```

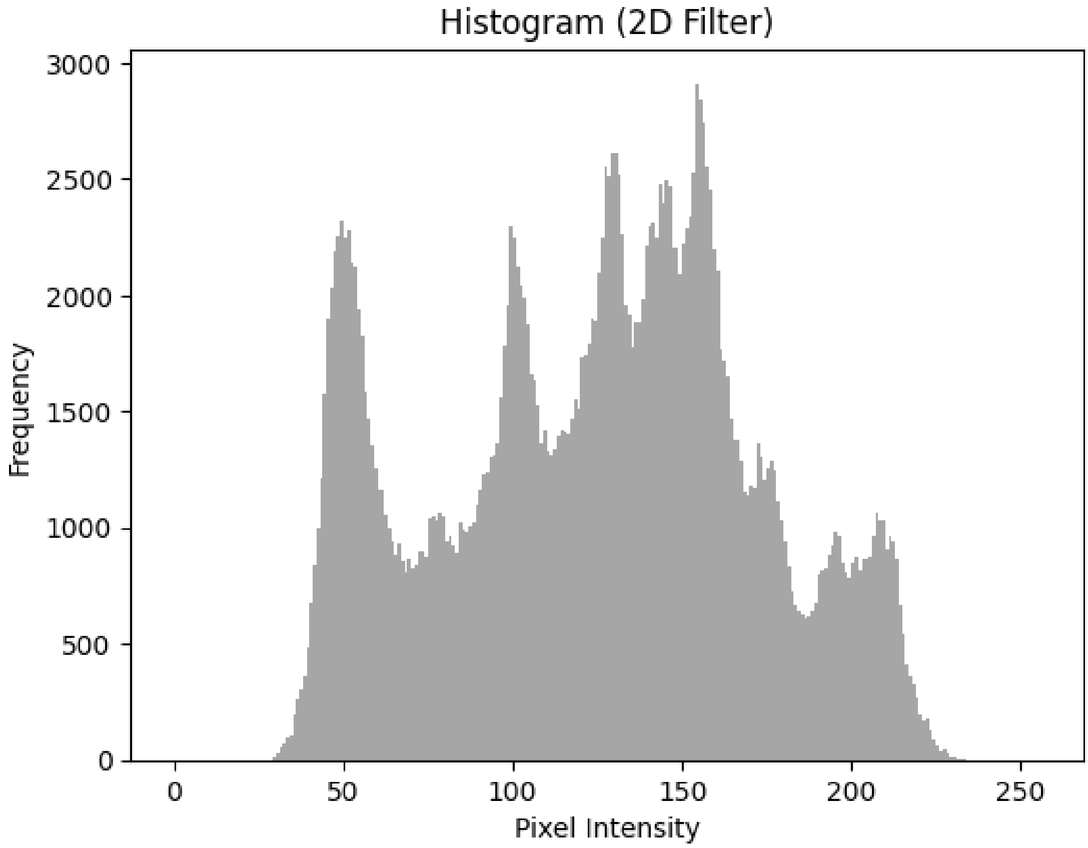
```
# 2. 2D Filter
kernel = np.array([[1, 1, 1],
                   [1, 1, 1],
                   [1, 1, 1]], dtype=np.float32) / 9
filter_2d = cv2.filter2D(image, -1, kernel)
plot_histogram(filter_2d, "Histogram (2D Filter)")

# 3. Gaussian Blur
gaussian_blur = cv2.GaussianBlur(image, (5, 5), sigmaX=1)
plot_histogram(gaussian_blur, "Histogram (Gaussian Blur)")

# 4. Median Blur
median_blur = cv2.medianBlur(image, 5)
plot_histogram(median_blur, "Histogram (Median Blur)")

# 5. Hardcoded Convolution
def hardcoded_convolution(image, kernel):
    kernel_height, kernel_width = kernel.shape
    pad_h, pad_w = kernel_height // 2, kernel_width // 2

    # Pad the image with zeros
    padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)), mode='constant', constant_values=0)

    # Prepare output image
    output = np.zeros_like(image)

    # Perform convolution
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            region = padded_image[i:i+kernel_height, j:j+kernel_width]
            output[i, j] = np.sum(region * kernel)

    # Normalize output to range 0-255
    output = np.clip(output, 0, 255).astype(np.uint8)
    return output

# Example kernel for convolution
custom_kernel = np.array([[0, -1, 0],
                          [-1, 5, -1],
                          [0, -1, 0]])  # Example sharpening kernel
custom_convolution = hardcoded_convolution(image, custom_kernel)
plot_histogram(custom_convolution, "Histogram (Custom Convolution)")

# Display all processed images
effects = {
    "Box Filter": box_filter,
    "2D Filter": filter_2d,
    "Gaussian Blur": gaussian_blur,
    "Median Blur": median_blur,
```

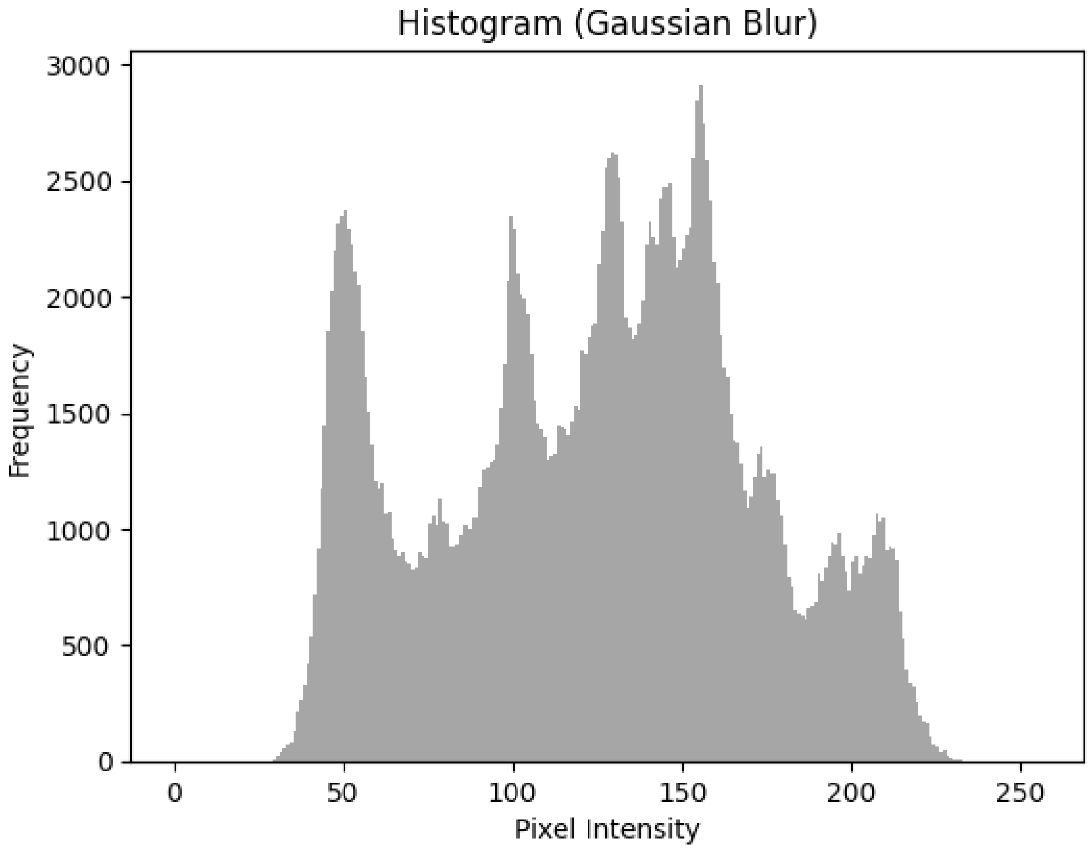
```
    "Custom Convolution": custom_convolution,
}

fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.ravel()

# Show processed images
for idx, (title, img) in enumerate(effects.items()):
    axes[idx].imshow(img, cmap='gray')
    axes[idx].set_title(title)
    axes[idx].axis('off')

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

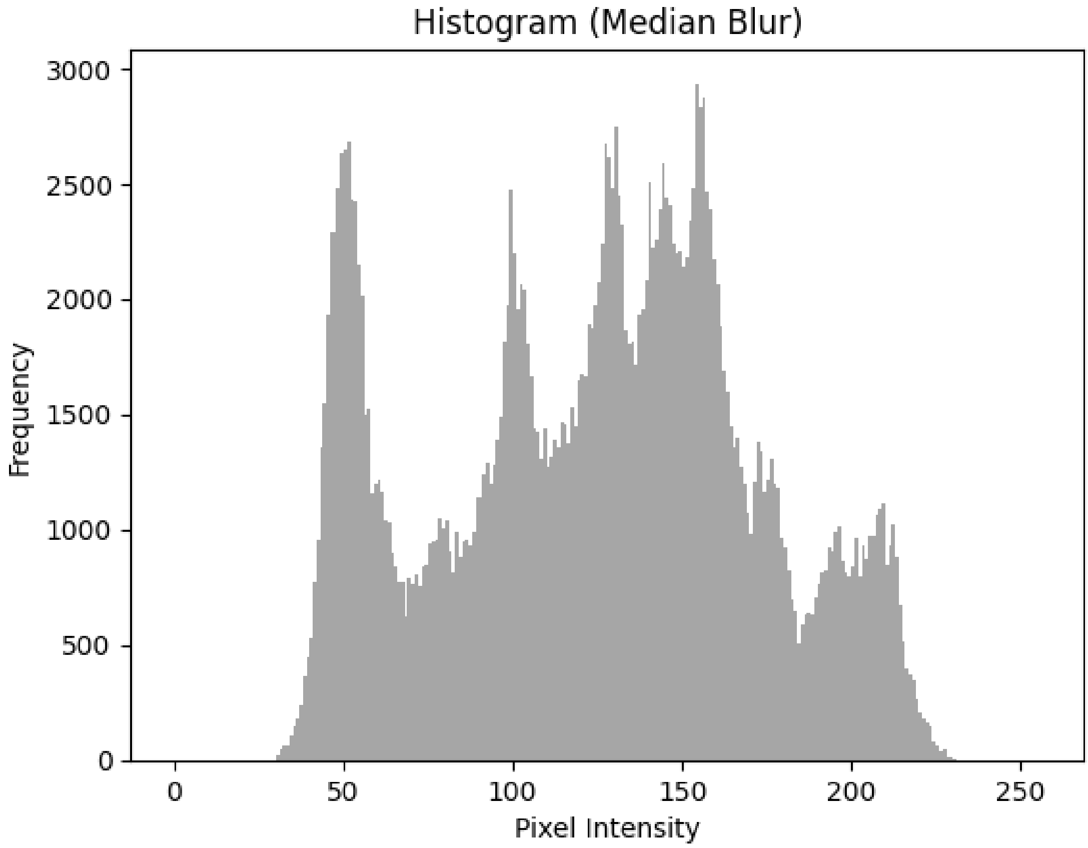
Original Image

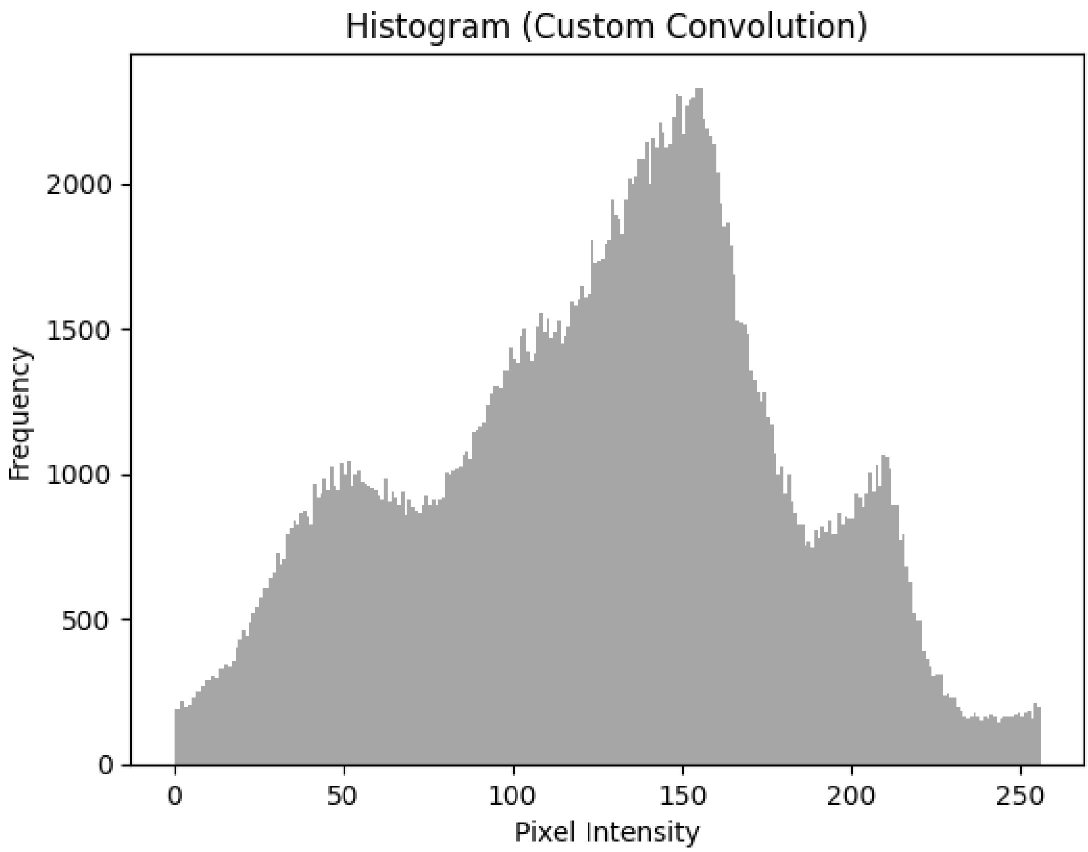

Histogram (Original Image)
Frequency
Pixel Intensity
0
500
1000
1500
2000
2500
0
50
100
150
200
250

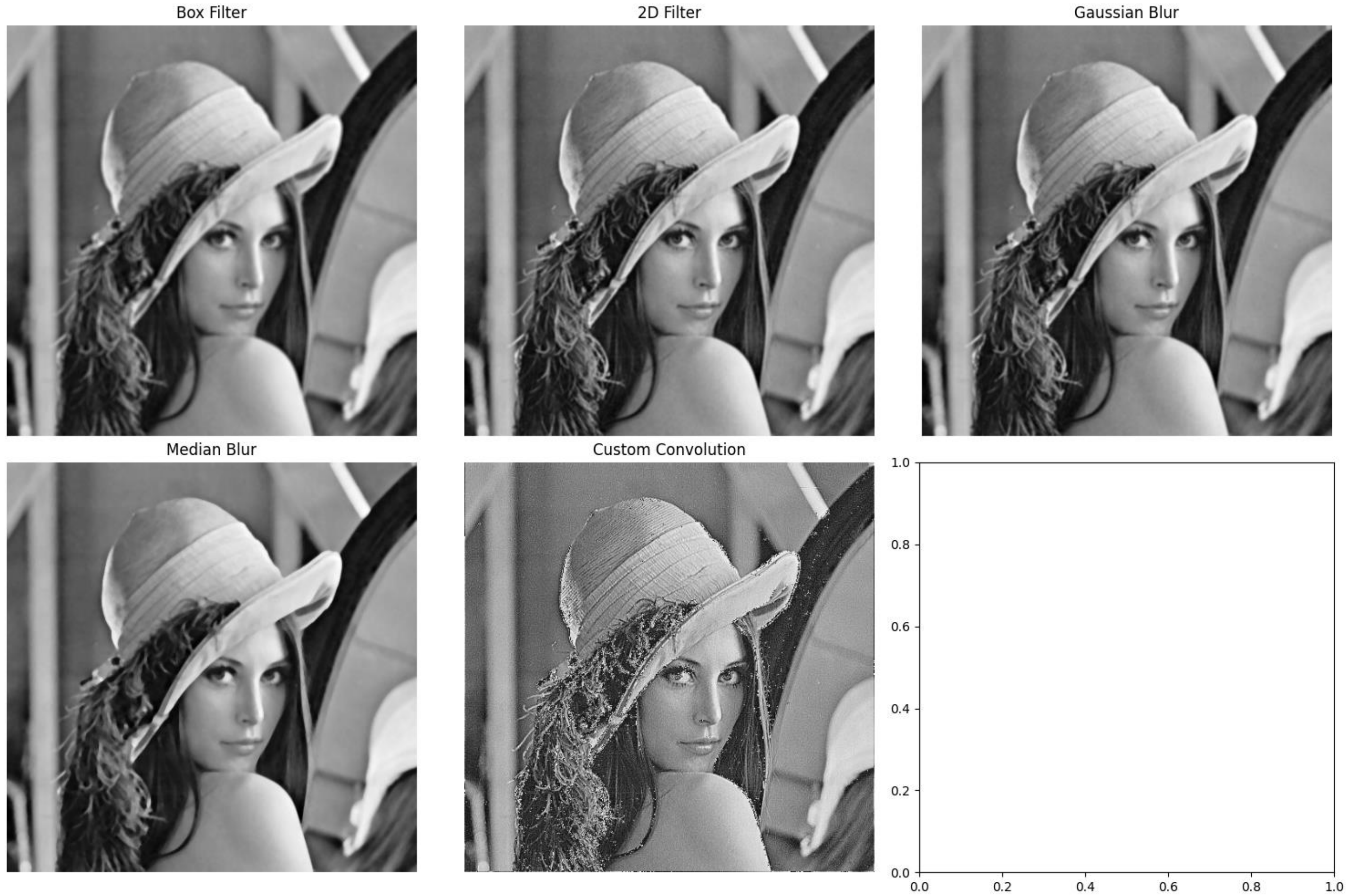

Histogram (Box Filter)
Frequency
Pixel Intensity
0
500
1000
1500
2000
2500
3000
0
50
100
150
200
250


Histogram (2D Filter)
Frequency
Pixel Intensity
0
500
1000
1500
2000
2500
3000
0
50
100
150
200
250


Histogram (Gaussian Blur)
Frequency
Pixel Intensity
0
500
1000
1500
2000
2500
3000
0
50
100
150
200
250


Histogram (Median Blur)
Frequency
Pixel Intensity
0
500
1000
1500
2000
2500
3000
0
50
100
150
200
250


Histogram (Custom Convolution)
Frequency
Pixel Intensity
0
500
1000
1500
2000
0
50
100
150
200
250

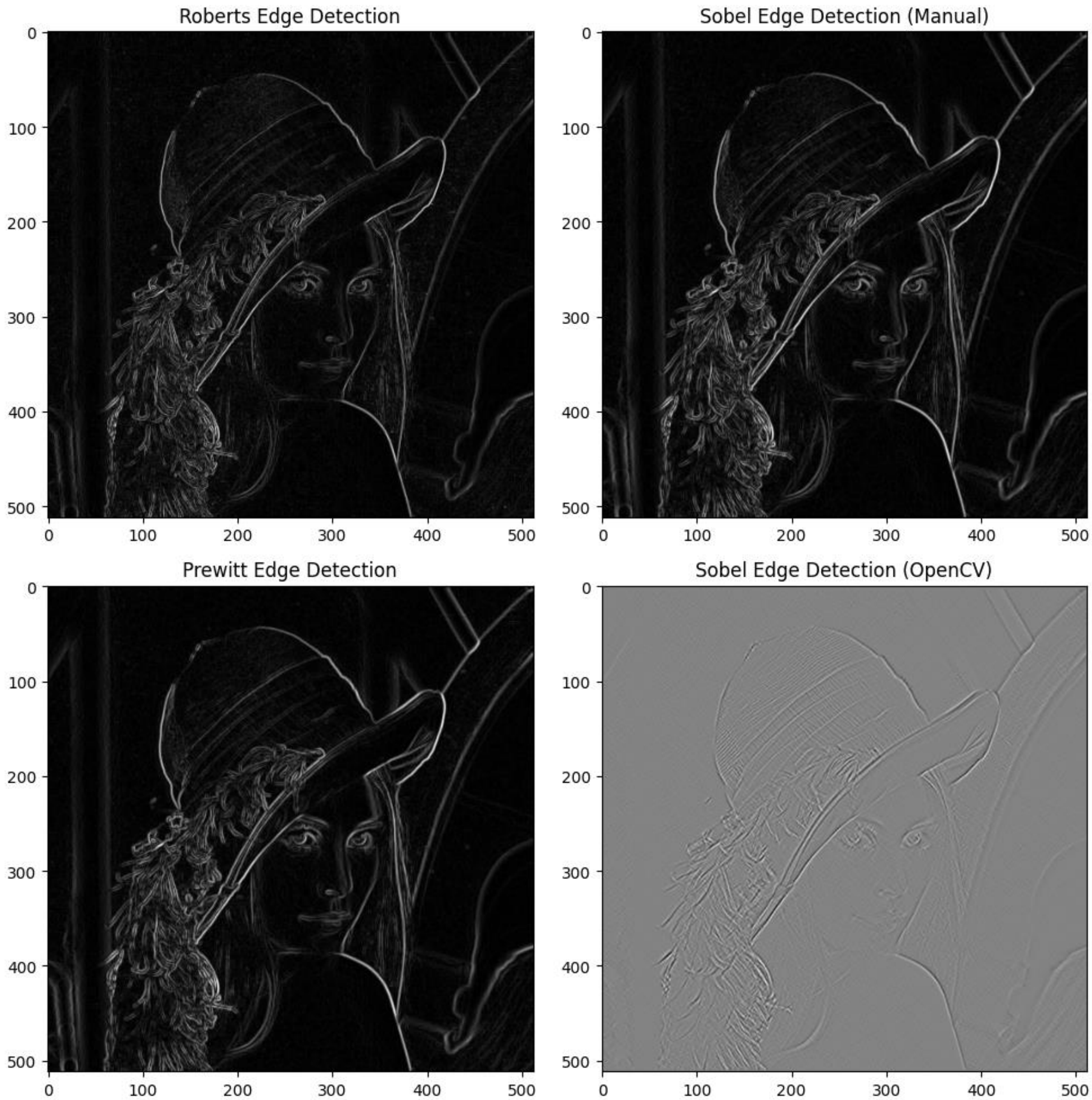


Edge detection using Roberts, Sobel, Prewitt, and Kenny using 2D filters as well as openCV functions.

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Define the edge detection functions

def roberts_edge_detection(image):
    image = np.float32(image)
    kernel_x = np.array([[1, 0], [0, -1]])
    kernel_y = np.array([[0, 1], [-1, 0]])

    grad_x = cv2.filter2D(image, -1, kernel_x)
    grad_y = cv2.filter2D(image, -1, kernel_y)
    grad = cv2.sqrt(grad_x**2 + grad_y**2)  # Combine gradients
    return grad

def sobel_edge_detection(image):
    image = np.float32(image)  # Convert to float32 for calculations
    kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    grad_x = cv2.filter2D(image, -1, kernel_x)
    grad_y = cv2.filter2D(image, -1, kernel_y)
```

```
    grad = cv2.sqrt(grad_x**2 + grad_y**2)  # Combine gradients
    return grad

def prewitt_edge_detection(image):
    image = np.float32(image)  # Convert to float32 for calculations
    kernel_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
    kernel_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])

    grad_x = cv2.filter2D(image, -1, kernel_x)
    grad_y = cv2.filter2D(image, -1, kernel_y)
    grad = cv2.sqrt(grad_x**2 + grad_y**2)  # Combine gradients
    return grad

# Load the image in grayscale
image = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)

edges_roberts = roberts_edge_detection(image)
edges_sobel = sobel_edge_detection(image)
edges_prewitt = prewitt_edge_detection(image)
edges_sobel_opencv = cv2.Sobel(image, cv2.CV_64F, 1, 1, ksize=3)

# Normalize
edges_roberts_normalized = cv2.normalize(edges_roberts, None, 0, 255, cv2.NORM_MINMAX)
edges_roberts_normalized = np.uint8(edges_roberts_normalized)

edges_sobel_normalized = cv2.normalize(edges_sobel, None, 0, 255, cv2.NORM_MINMAX)
edges_sobel_normalized = np.uint8(edges_sobel_normalized)

edges_prewitt_normalized = cv2.normalize(edges_prewitt, None, 0, 255, cv2.NORM_MINMAX)
edges_prewitt_normalized = np.uint8(edges_prewitt_normalized)

edges_sobel_opencv_normalized = cv2.normalize(edges_sobel_opencv, None, 0, 255,
cv2.NORM_MINMAX)
edges_sobel_opencv_normalized = np.uint8(edges_sobel_opencv_normalized)

plt.figure(figsize=(10, 10))

plt.subplot(2, 2, 1)
plt.imshow(edges_roberts_normalized, cmap='gray')
plt.title('Roberts Edge Detection')

plt.subplot(2, 2, 2)
plt.imshow(edges_sobel_normalized, cmap='gray')
plt.title('Sobel Edge Detection (Manual)')

plt.subplot(2, 2, 3)
plt.imshow(edges_prewitt_normalized, cmap='gray')
plt.title('Prewitt Edge Detection')
```

```
plt.subplot(2, 2, 4)
plt.imshow(edges_sobel_opencv_normalized, cmap='gray')
plt.title('Sobel Edge Detection (OpenCV)')

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

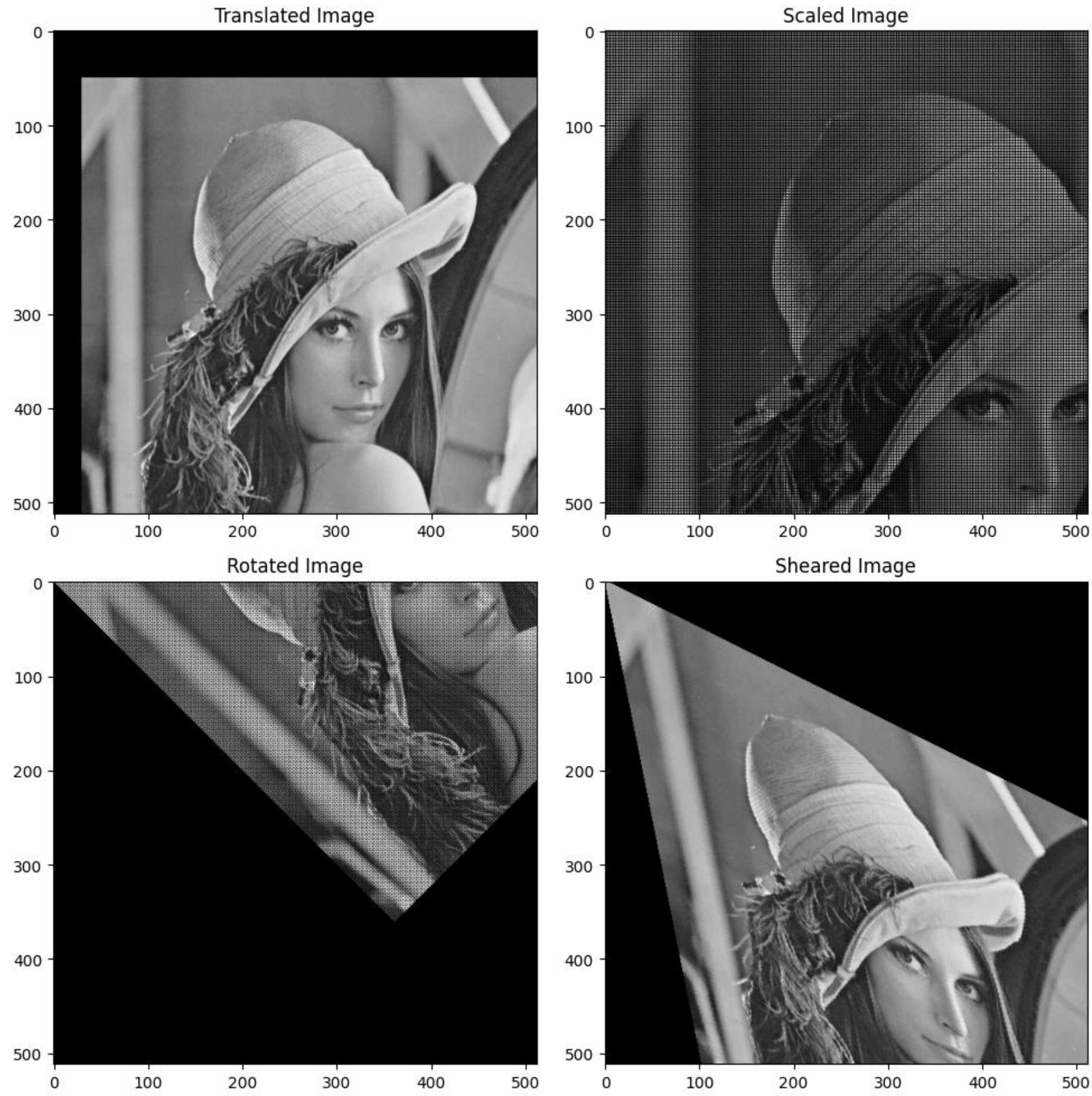


Perform transformations on image using hardcoding with matrix multiplication of image and matrices of transformation

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
def transform_image(image, transformation_matrix):
    rows, cols = image.shape
    transformed_image = np.zeros_like(image)
    for i in range(rows):
```

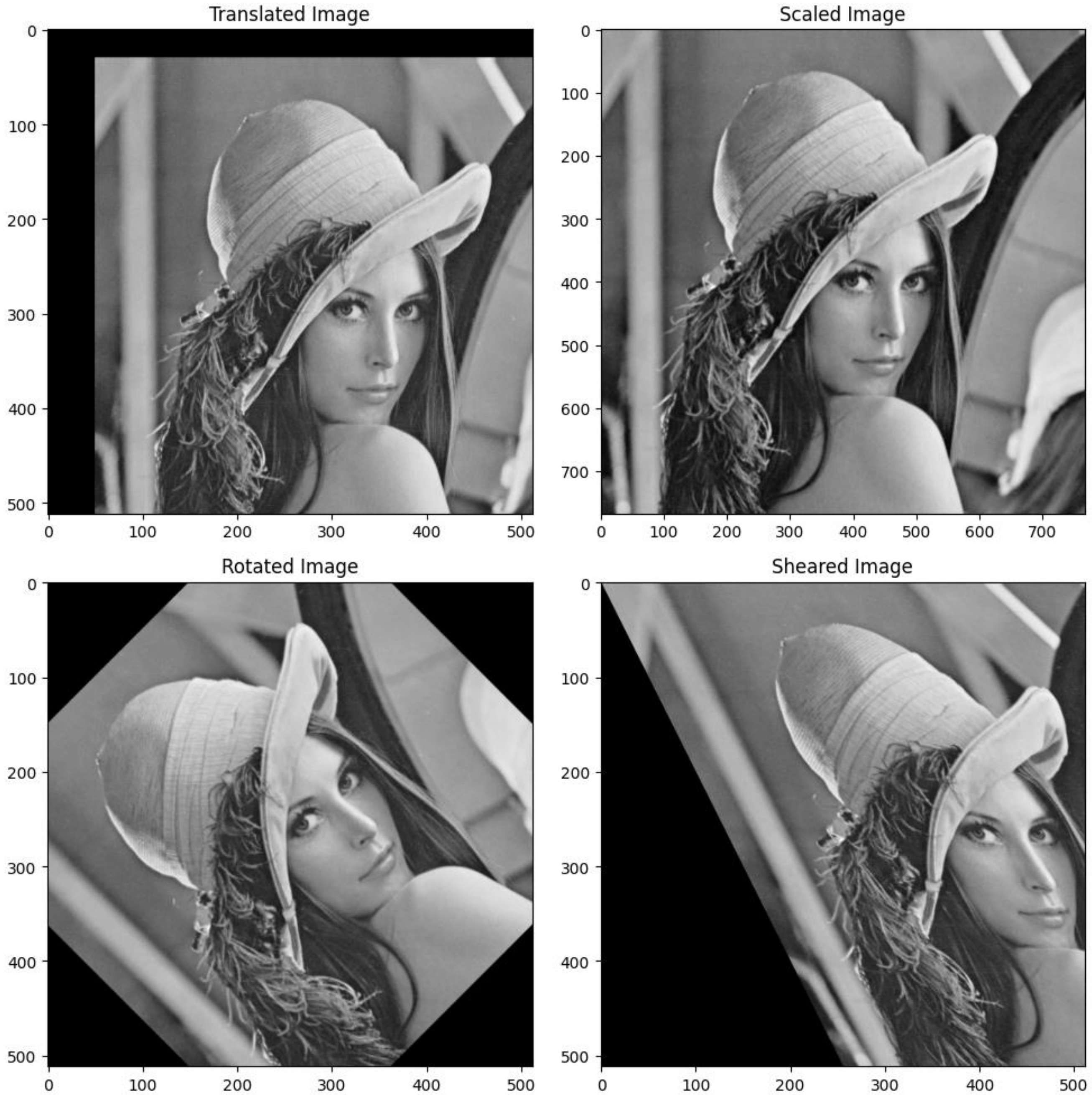
```
        for j in range(cols):
            original_coords = np.array([i, j, 1])
            transformed_coords = np.dot(transformation_matrix, original_coords)
            new_x, new_y = int(transformed_coords[1]), int(transformed_coords[0])
            if 0 <= new_x < cols and 0 <= new_y < rows:
                transformed_image[new_y, new_x] = image[i, j]

    return transformed_image
image = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)

# Translation matrix (shift image 50 pixels right, 30 pixels down)
T = np.array([[1, 0, 50],
              [0, 1, 30],
              [0, 0, 1]])

# Scaling matrix (scale by 1.5 in x and 1.5 in y)
S = np.array([[1.5, 0, 0],
              [0, 1.5, 0],
              [0, 0, 1]])

# Rotation matrix (rotate by 45 degrees)
theta = 45
theta = np.radians(theta)  # Convert to radians
R = np.array([[np.cos(theta), -np.sin(theta), 0],
              [np.sin(theta), np.cos(theta), 0],
              [0, 0, 1]])

# Shear matrix (shear by 0.5 in x direction and 0.2 in y direction)
H = np.array([[1, 0.5, 0],
              [0.2, 1, 0],
              [0, 0, 1]])
# Apply transformations
image_translated = transform_image(image, T)
image_scaled = transform_image(image, S)
image_rotated = transform_image(image, R)
image_sheared = transform_image(image, H)

plt.figure(figsize=(10, 10))

plt.subplot(2, 2, 1)
plt.imshow(image_translated, cmap='gray')
plt.title('Translated Image')
```

```
plt.subplot(2, 2, 2)
plt.imshow(image_scaled, cmap='gray')
plt.title('Scaled Image')

plt.subplot(2, 2, 3)
plt.imshow(image_rotated, cmap='gray')
plt.title('Rotated Image')

plt.subplot(2, 2, 4)
plt.imshow(image_sheared, cmap='gray')
plt.title('Sheared Image')

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



Perform transformation using the OpenCV library

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

def transform_image_opencv(image, transformation_matrix, output_size):
   transformed_image = cv2.warpAffine(image, transformation_matrix, output_size)
   return transformed_image

image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
translation_matrix = np.float32([[1, 0, 50], [0, 1, 30]])
height, width = image.shape
image_translated = transform_image_opencv(image, translation_matrix, (width, height))
image_scaled = cv2.resize(image, None, fx=1.5, fy=1.5, interpolation=cv2.INTER_LINEAR)
rotation_matrix = cv2.getRotationMatrix2D((width//2, height//2), 45, 1)
image_rotated = transform_image_opencv(image, rotation_matrix, (width, height))
shear_matrix = np.float32([[1, 0.5, 0], [0, 1, 0]])
image_sheared = transform_image_opencv(image, shear_matrix, (width, height))

plt.figure(figsize=(10, 10))

plt.subplot(2, 2, 1)
plt.imshow(image_translated, cmap='gray')
plt.title('Translated Image')

plt.subplot(2, 2, 2)
plt.imshow(image_scaled, cmap='gray')
plt.title('Scaled Image')

plt.subplot(2, 2, 3)
plt.imshow(image_rotated, cmap='gray')
plt.title('Rotated Image')

plt.subplot(2, 2, 4)
plt.imshow(image_sheared, cmap='gray')
plt.title('Sheared Image')

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



Perform Segmentation using

- **Simple thresholding**
- **Otsu Binarization**
- **Adaptive Thresholding**

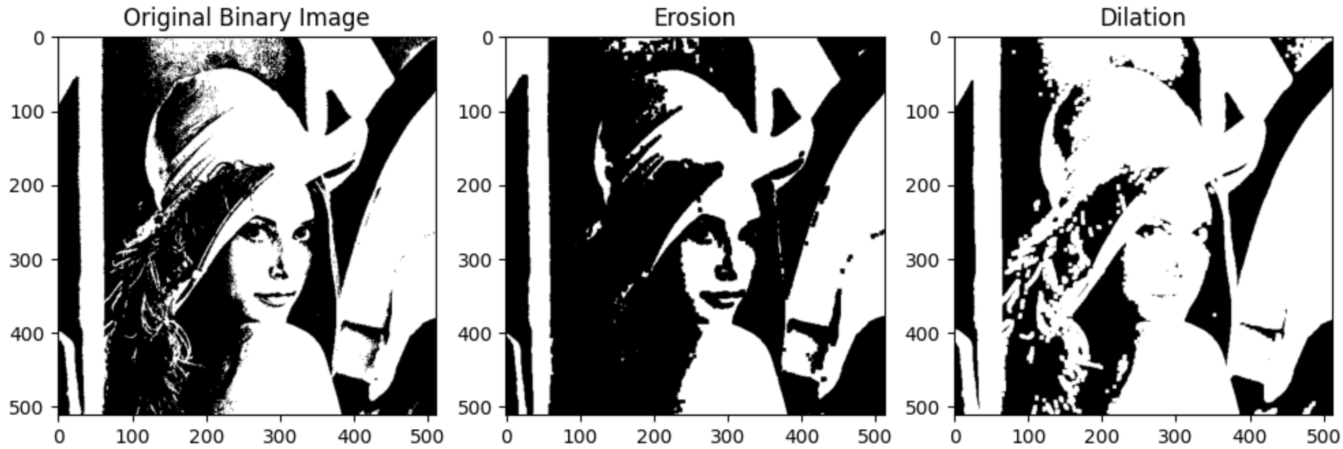
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
_, thresh_simple = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
_, thresh_otsu = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
thresh_adaptive = cv2.adaptiveThreshold(image, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
```

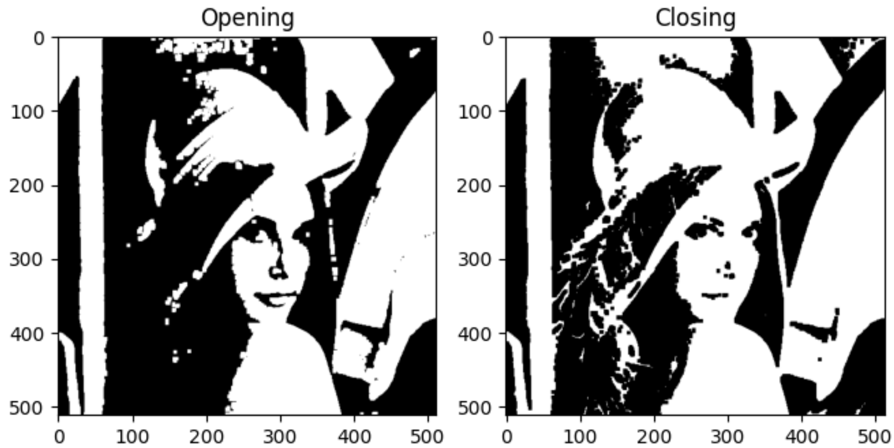
```
                           cv2.THRESH_BINARY, 11, 2)

plt.figure(figsize=(10, 10))
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

plt.subplot(2, 2, 2)
plt.imshow(thresh_simple, cmap='gray')
plt.title('Simple Thresholding')

plt.subplot(2, 2, 3)
plt.imshow(thresh_otsu, cmap='gray')
plt.title("Otsu's Binarization")

plt.subplot(2, 2, 4)
plt.imshow(thresh_adaptive, cmap='gray')
plt.title('Adaptive Thresholding')
plt.tight_layout()
plt.show()
```

Perform erosion ,dilation, opening ,closing

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
kernel = np.ones((5, 5), np.uint8)

erosion = cv2.erode(binary_image, kernel, iterations=1)
dilation = cv2.dilate(binary_image, kernel, iterations=1)
opening = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel)
closing = cv2.morphologyEx(binary_image, cv2.MORPH_CLOSE, kernel)
```

```
plt.figure(figsize=(10, 10))
plt.subplot(2, 3, 1)
plt.imshow(binary_image, cmap='gray')
plt.title('Original Binary Image')

plt.subplot(2, 3, 2)
plt.imshow(erosion, cmap='gray')
plt.title('Erosion')

plt.subplot(2, 3, 3)
plt.imshow(dilation, cmap='gray')
plt.title('Dilation')

plt.subplot(2, 3, 4)
plt.imshow(opening, cmap='gray')
plt.title('Opening')

plt.subplot(2, 3, 5)
plt.imshow(closing, cmap='gray')
plt.title('Closing')
plt.tight_layout()
plt.show()
```

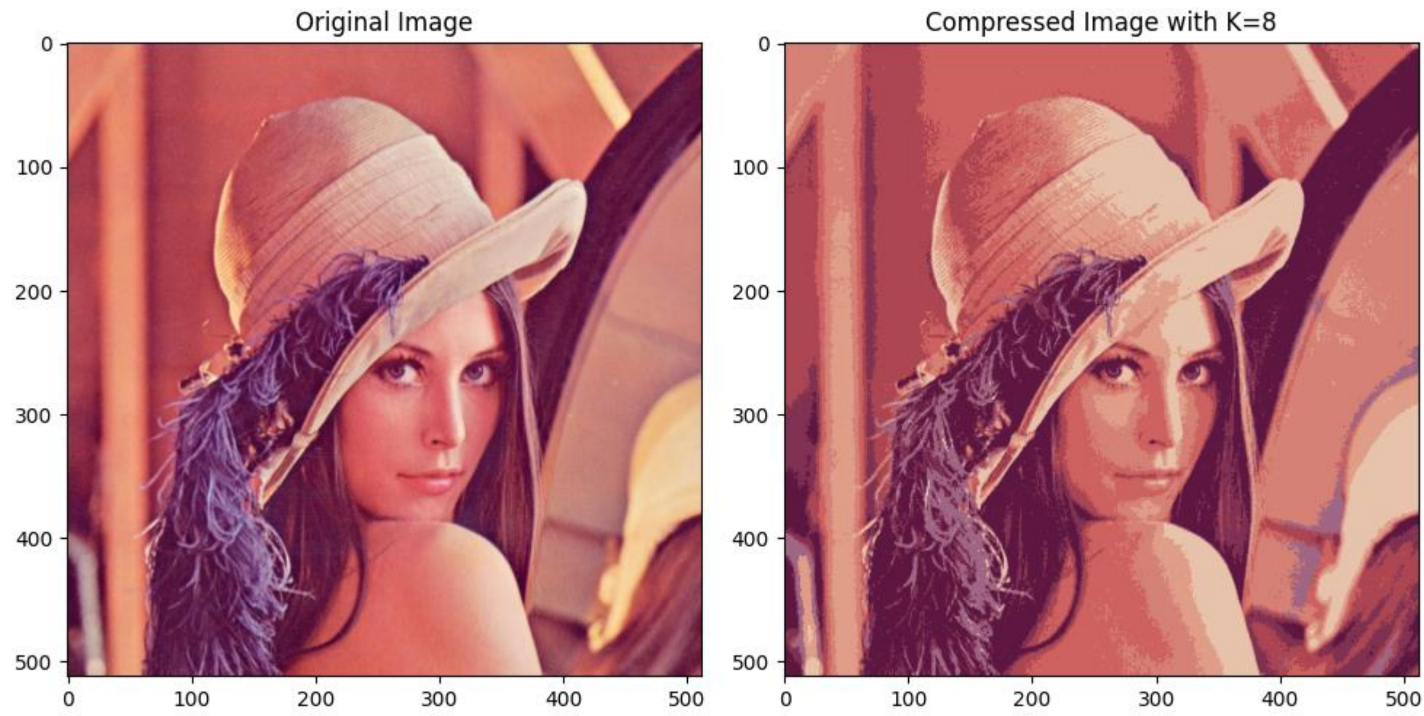




WAP to implement image compression using K-Means.

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

def compress_image_kmeans(image_path, k=8):
   image = cv2.imread(image_path)
   image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
   pixels = image_rgb.reshape(-1, 3)
   kmeans = KMeans(n_clusters=k, random_state=42)
   kmeans.fit(pixels)
   compressed_pixels = kmeans.cluster_centers_[kmeans.labels_]
   compressed_image = compressed_pixels.reshape(image_rgb.shape).astype(np.uint8)
   return compressed_image, kmeans
image_path = 'lena.jpg'
compressed_image, kmeans = compress_image_kmeans(image_path, k=8)
```

```
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(compressed_image)
plt.title(f'Compressed Image with K={8}')
plt.tight_layout()
plt.show()

print("Cluster centers (RGB values):")
print(kmeans.cluster_centers_)
```

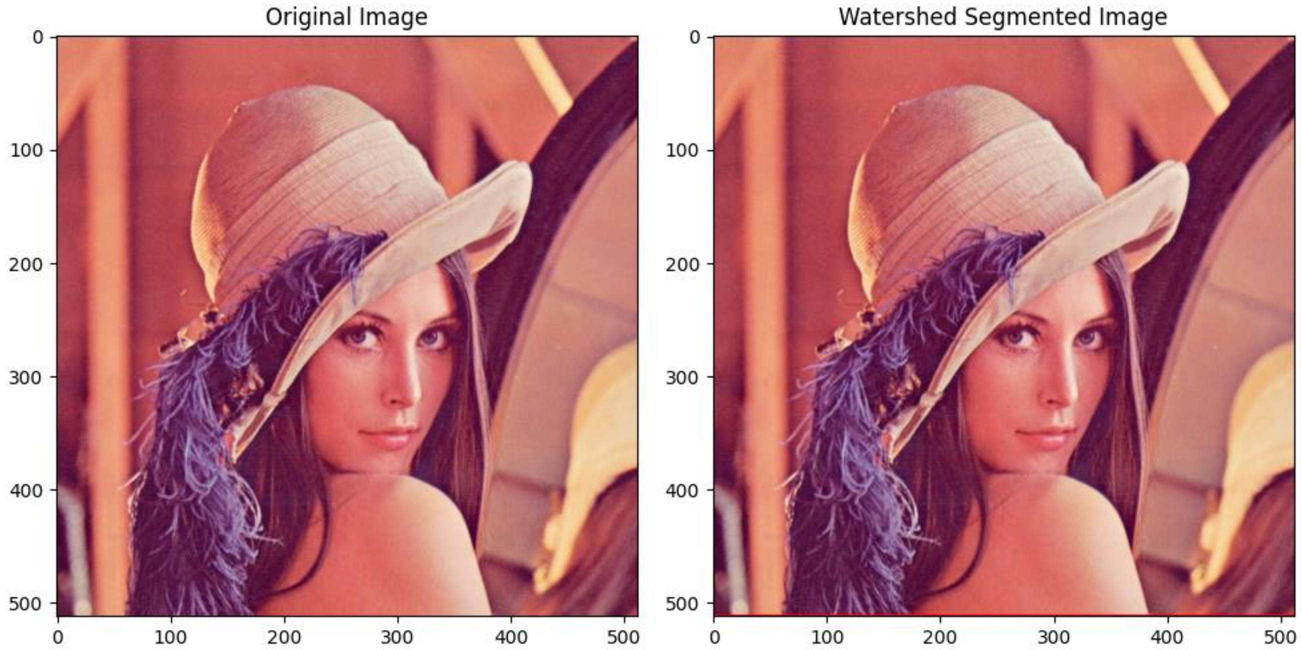



```
Cluster centers (RGB values):
[[129.70249267  50.43704616  82.54333517]
 [213.16440006 129.2301834  117.67438607]
 [172.93762304  70.1508859   82.04239368]
 [232.17514473 195.94697028 171.84913161]
 [224.3887498  158.81222743 137.56501373]
 [ 93.59140013  23.71865659  64.84226506]
 [205.9218581   98.04462676  96.88676524]
```

```
[159.12333147 103.21980657 133.78019343]]
```

WAP to implement Watershed algorithm.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def watershed_segmentation(image_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, binary = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY_INV)
    dist_transform = cv2.distanceTransform(binary, cv2.DIST_L2, 5)
    _, markers = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)
    markers = np.int32(markers)
    markers = markers + 1
    markers[binary == 255] = 0
    cv2.watershed(image, markers)
    image[markers == -1] = [0, 0, 255]
    return image, markers

image_path = 'image.jpg'
segmented_image, markers = watershed_segmentation(image_path)
image = cv2.imread(image_path)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(segmented_image, cv2.COLOR_BGR2RGB))
plt.title('Watershed Segmented Image')
plt.tight_layout()
plt.show()
```



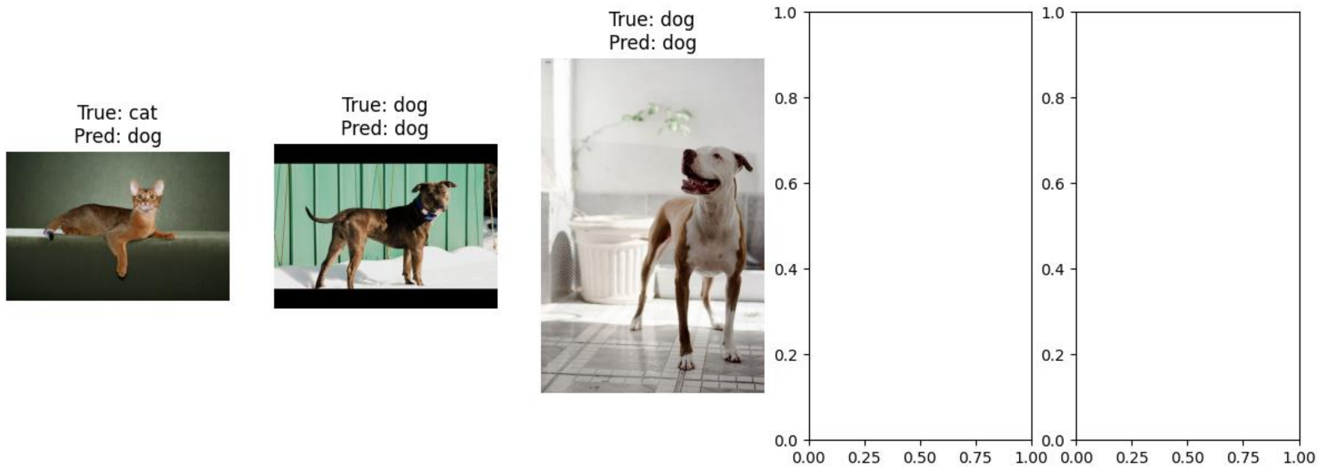
WAP for image classification using traditional ML algorithms.

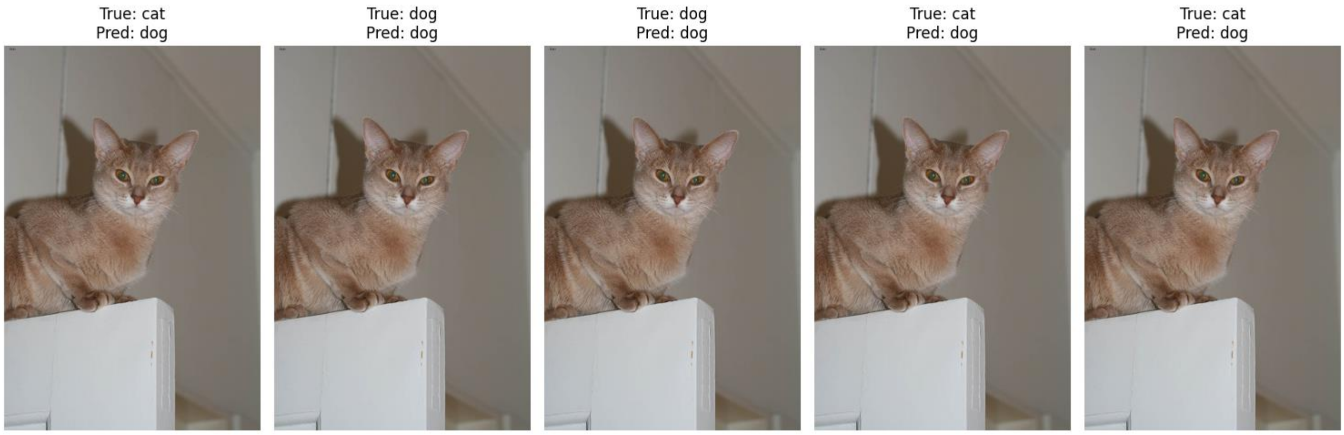
```
import cv2
import numpy as np
from skimage.feature import hog
from skimage import exposure
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import os
import matplotlib.pyplot as plt

def extract_hog_features(image_path):
    image = cv2.imread("C:/Users/priya/Downloads/dataset/cat/Abyssinian_71.jpg",
cv2.IMREAD_GRAYSCALE)
    image_resized = cv2.resize(image, (64, 64))
    features, _ = hog(image_resized, pixels_per_cell=(8, 8), cells_per_block=(2, 2),
visualize=True)
    return features

image_directory = 'dataset'
categories = ['cat', 'dog']
features_list = []
labels_list = []

# Loop through the categories and load images
```

```
for label, category in enumerate(categories):
    category_folder = os.path.join(image_directory, category)

    for filename in os.listdir(category_folder):
        # Full image path
        image_path = os.path.join(category_folder, filename)

        # Extract HOG features from the image
        features = extract_hog_features(image_path)

        # Append features and corresponding label to the lists
        features_list.append(features)
        labels_list.append(label)

# Convert lists to numpy arrays
X = np.array(features_list)  # Feature matrix
y = np.array(labels_list)    # Label vector

# Split data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the SVM classifier (Support Vector Machine)
svm_classifier = SVC(kernel='linear', random_state=42)

# Train the classifier using the training data
svm_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = svm_classifier.predict(X_test)

# Evaluate the classifier
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=categories))

# Optionally, visualize some test images and predictions
fig, axes = plt.subplots(1, 5, figsize=(15, 5))

for i in range(5):
    ax = axes[i]
```

```
    img_path = os.path.join(image_directory, categories[y_test[i]],
os.listdir(os.path.join(image_directory, categories[y_test[i]]))[i])
    img = cv2.imread("C:/Users/priya/Downloads/dataset/cat/Abyssinian_129.jpg")
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    ax.imshow(img_rgb)
    ax.set_title(f"True: {categories[y_test[i]]}\nPred: {categories[y_pred[i]]}")
    ax.axis('off')

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

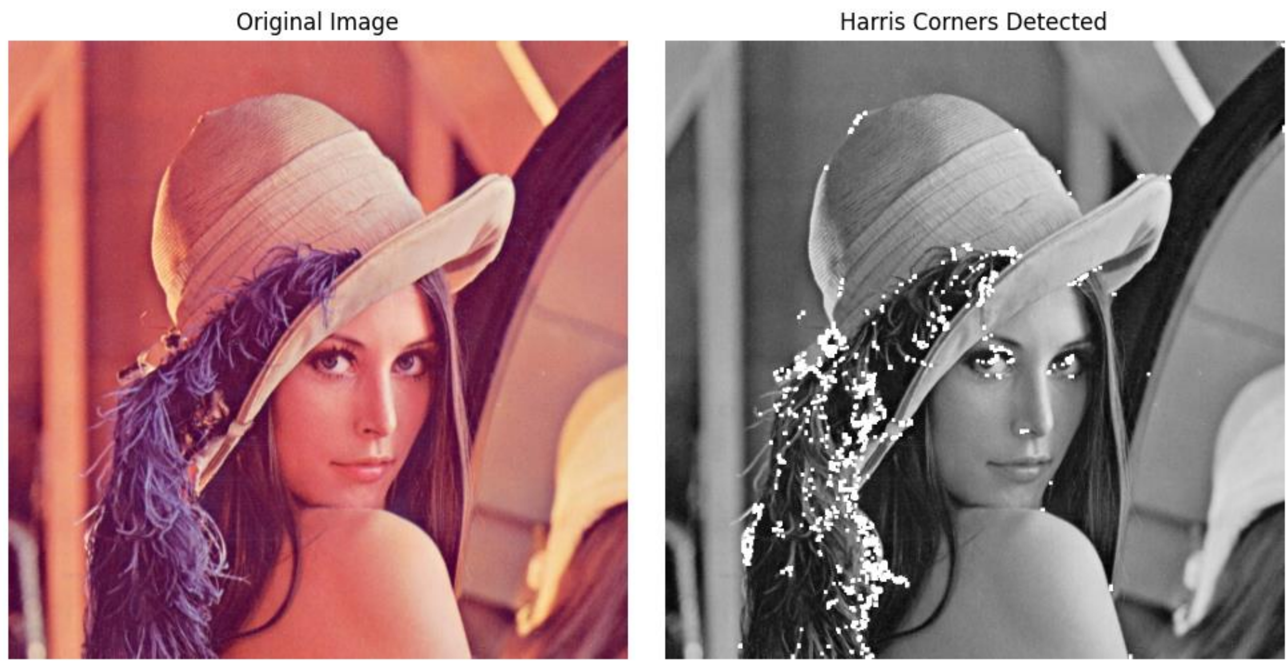
```
Confusion Matrix:
[[ 0 46]
 [ 0 35]]
Accuracy: 0.43209876543209874
Classification Report:
              precision    recall  f1-score   support

         cat       0.00      0.00      0.00        46
         dog       0.43      1.00      0.60        35

    accuracy                           0.43        81
   macro avg       0.22      0.50      0.30        81
weighted avg       0.19      0.43      0.26        81
```





WAP to extract Haris corner detection feature.

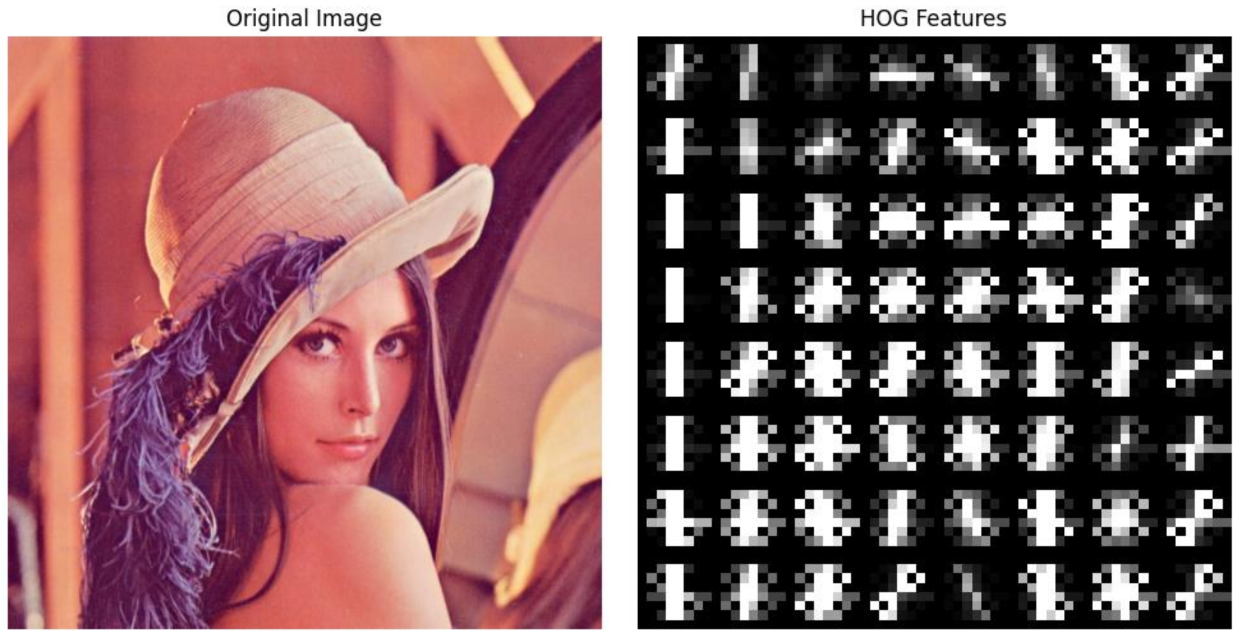
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def harris_corner_detection(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    image_float = np.float32(image)
    corners = cv2.cornerHarris(image_float, blockSize=2, ksize=3, k=0.04)
    corners_dilated = cv2.dilate(corners, None)
    image_with_corners = np.copy(image)
    image_with_corners[corners_dilated > 0.01 * corners_dilated.max()] = 255
    return image_with_corners, corners_dilated

image_path = 'lena.jpg'
image_with_corners, corners_dilated = harris_corner_detection(image_path)
image_color = cv2.imread(image_path)
image_color_rgb = cv2.cvtColor(image_color, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_color_rgb)
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(image_with_corners, cmap='gray')
plt.title('Harris Corners Detected')
plt.axis('off')
plt.tight_layout()
plt.show()
```



WAP to extract HOG features.

```
import cv2
import numpy as np
from skimage.feature import hog
from skimage import exposure
import matplotlib.pyplot as plt

def extract_hog_features(image_path):
    image = cv2.imread(image_path)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray_image_resized = cv2.resize(gray_image, (64, 64))
    features, hog_image = hog(gray_image_resized, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), visualize=True)
    hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))
    return features, hog_image_rescaled

image_path = 'lena.jpg'
features, hog_image_rescaled = extract_hog_features(image_path)
original_image = cv2.imread(image_path)
original_image_rgb = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(original_image_rgb)
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
```

```
plt.imshow(hog_image_rescaled, cmap='gray')
plt.title('HOG Features')
plt.axis('off')
plt.tight_layout()
plt.show()
print(f"Length of HOG feature vector: {len(features)}")
```

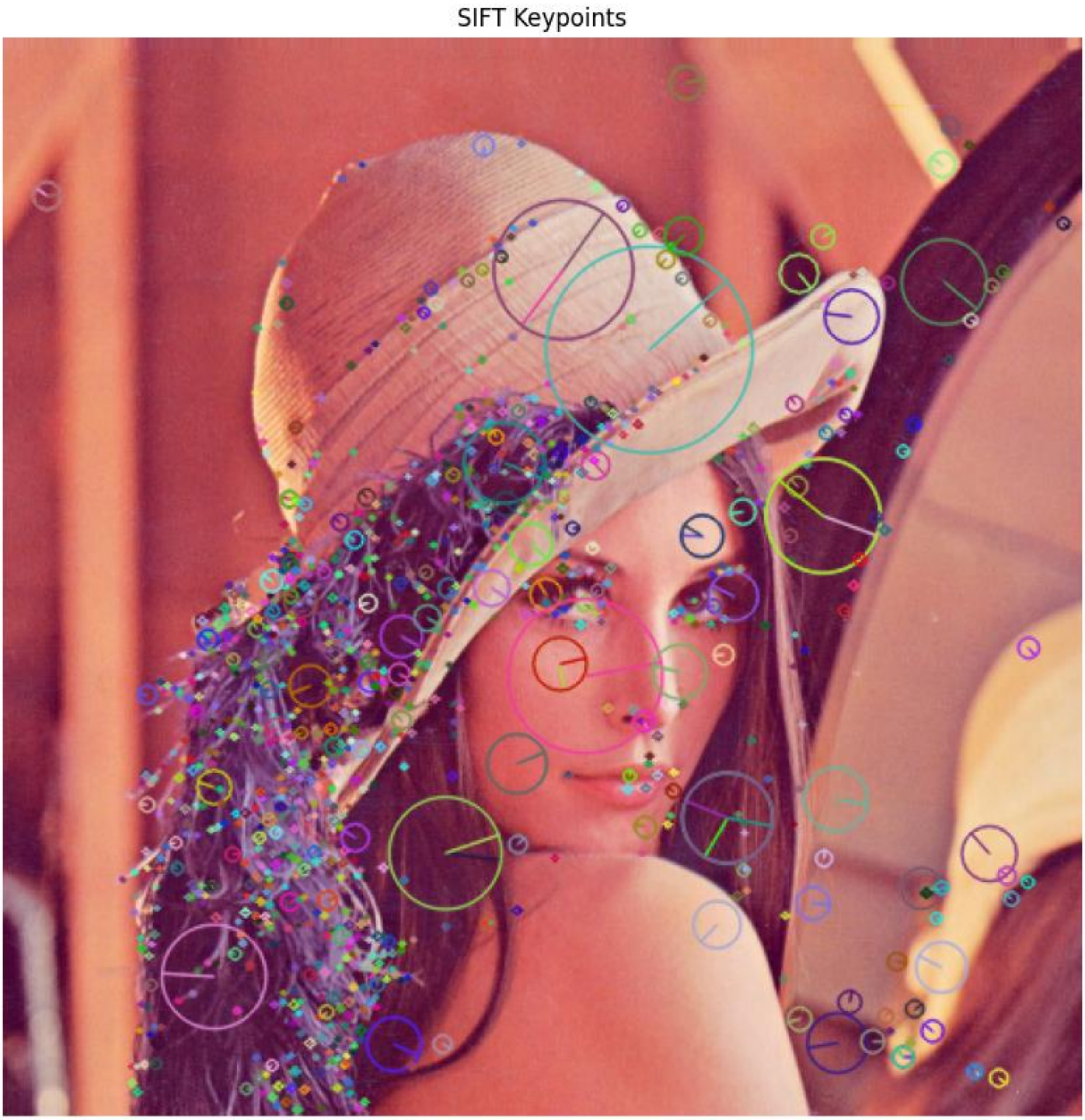


Length of HOG feature vector: 1764

WAP to extract SIFT features.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def extract_sift_features(image_path):
    image = cv2.imread(image_path)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    sift = cv2.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(gray_image, None)
    return image, keypoints, descriptors

image_path = 'lena.jpg'
image_with_keypoints, keypoints, descriptors = extract_sift_features(image_path)
image_rgb = cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB)
image_with_keypoints = cv2.drawKeypoints(image_rgb, keypoints, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.figure(figsize=(10, 10))
plt.imshow(image_with_keypoints)
```

```python
plt.title('SIFT Keypoints')
plt.axis('off')
plt.show()

print(f"Number of keypoints detected: {len(keypoints)}")
```



Number of keypoints detected: 1111