HCMUS - VNUHCM / FIT / Computer Vision & Cognitive Cybernetics Department

Digital image & video processing - LQN

Student ID: 21127690

Student name: Ngo Nguyen Thanh Thanh

Report: OPENCV – BASIC

I. Evaluation summary:

Task			**Requirement Met(%)**	**Notes**
Implementation	Algorithm to transform color	Linear mapping	100%	
		Non-linear mapping	100%	Logarithmic & Exponential mapping
		Probability Density Function-based mapping	100%	
	Algorithm to transform geometry	Pixel co-ordinate transformations	100%	Affine Transformation and Bilinear transform:
		Brightness interpolation	100%	Nearest-neighborhood interpolation and Linear interpolation
	Algorithm to smooth the image	Averaging filter	100%	
		Gaussian filter	100%	
		Median filter	100%	
	Algorithm to blur the image	Gaussian Blur	100%	
Total:			100%	

II. List of features:

List of Functions: a summary of the key functions in the program:

1. **read_image(file_path)**: Reads an image from the given file path in grayscale using OpenCV.
2. **linear_mapping(image, a=1.0, b=0)**: Applies a linear transformation (scaling and shifting) to the image.

3. **log_mapping(image, c=1.0)**: Applies a logarithmic transformation to compress pixel values.
4. **exp_mapping(image, c=1.0)**: Applies an exponential transformation to expand pixel values.
5. **histogram_equalization(image)**: Adjusts image contrast by redistributing pixel intensities.
6. **affine_transform(image, a, b)**: Applies affine transformation to modify pixel coordinates.
7. **nearest_neighbor_resize(image, scale_x, scale_y)**: Resizes the image using the nearest neighbor method.
8. **linear_interpolation_resize(image, scale_x, scale_y)**: Resizes the image using linear interpolation.
9. **bilinear_transform(image, a, b)**: Applies bilinear transformation with interpolation.
10. **averaging_filter(image, kernel_size)**: Applies a box filter to smooth the image.
11. **gaussian_kernel(size, sigma)**: Generates a Gaussian kernel for filtering.
12. **gaussian_filter(image, kernel)**: Applies a Gaussian filter to smooth the image.
13. **median_filter(image, kernel_size)**: Applies a median filter to remove noise.
14. **gaussian_blur(image, kernel)**: Applies Gaussian blur to the image.

- Menu Functions:

1. **menu()**: Displays the main menu with options for different tasks.
2. **color_transform_menu(image)**: Displays the color transformation menu.
3. **geometric_transform_menu(image)**: Displays the geometric transformation menu.
4. **smoothing_menu(image)**: Displays the image smoothing menu.

These functions handle the selection of image processing tasks and apply transformations or filters accordingly.

The program with proof images:

```python
# Hàm đọc ảnh bằng OpenCV
def read_image(file_path):
    try:
        img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)  # Đọc ảnh dưới dạng grayscale
        return img
    except FileNotFoundError:
        print("Không tìm thấy file ảnh. Hãy kiểm tra lại đường dẫn!")
        exit()
```

```python
# ======================== Biến đổi màu ======================== #

# Linear Mapping
def linear_mapping(image, a=1.0, b=0):
    height, width = image.shape
    output = np.zeros_like(image, dtype=np.float32)
    for i in range(height):
        for j in range(width):
            output[i, j] = a * image[i, j] + b
    return np.clip(output, 0, 255).astype(np.uint8)

# Logarithmic Mapping
def log_mapping(image, c=1.0):
    output = c * np.log1p(image.astype(np.float32))
    return np.clip(output, 0, 255).astype(np.uint8)

# Exponential Mapping
def exp_mapping(image, c=1.0):
    output = c * (np.exp(image.astype(np.float32) / 255.0) - 1) * 255
    return np.clip(output, 0, 255).astype(np.uint8)
```

```python
# Histogram Equalization
def histogram_equalization(image):
    hist = np.zeros(256, dtype=int) ...
    cdf_scaled = (cdf * 255).astype(np.uint8)
    output = cdf_scaled[image]
    return output
```

```python
# ======================== Biến đổi hình học ======================== #

# Affine Transformation
def affine_transform(image, a, b):
    height, width = image.shape
    output = np.zeros_like(image)
    for i in range(height):
        for j in range(width):
            x_new = int(a[0] + a[1]*j + a[2]*i)
            y_new = int(b[0] + b[1]*j + b[2]*i)
            if 0 <= x_new < width and 0 <= y_new < height:
                output[y_new, x_new] = image[i, j]
    return output
```

```python
# Nearest Neighbor Resize
def nearest_neighbor_resize(image, scale_x, scale_y):
    height, width = image.shape
    new_height = int(height * scale_y)
    new_width = int(width * scale_x)
    output = np.zeros((new_height, new_width), dtype=image.dtype)
    for i in range(new_height):
        for j in range(new_width):
            x = int(j / scale_x)
            y = int(i / scale_y)
            output[i, j] = image[y, x]
    return output
```

```python
# Linear_interpolation_resize
def linear_interpolation_resize(image, scale_x, scale_y):
    height, width = image.shape
    new_height = int(height * scale_y)
    new_width = int(width * scale_x)
    output = np.zeros((new_height, new_width), dtype=image.dtype)

    for i in range(new_height):
        for j in range(new_width):
            # Tọa độ thực trong ảnh gốc
            x = j / scale_x
            y = i / scale_y

            # Các pixel lân cận
            x0 = int(np.floor(x))
            x1 = min(x0 + 1, width - 1)
            y0 = int(np.floor(y))
            y1 = min(y0 + 1, height - 1)

            # Hệ số nội suy
            alpha = x - x0
            beta = y - y0

            # Nội suy giá trị pixel
            output[i, j] = (
                (1 - alpha) * (1 - beta) * image[y0, x0] +
                alpha * (1 - beta) * image[y0, x1] +
                (1 - alpha) * beta * image[y1, x0] +
                alpha * beta * image[y1, x1]
            )

    return output
```

```python
# Bilinear_transform
def bilinear_transform(image, a, b):
    """
    Thực hiện phép biến đổi Bilinear trên ảnh.
    Args:
        image: Ảnh đầu vào (numpy array, grayscale).
        a: Danh sách 4 hệ số [a0, a1, a2, a3] cho công thức x'.
        b: Danh sách 4 hệ số [b0, b1, b2, b3] cho công thức y'.
    Returns:
        Ảnh sau khi biến đổi (numpy array).
    """
    height, width = image.shape
    output = np.zeros_like(image)

    for i in range(height):
        for j in range(width):
            # Tọa độ mới sau biến đổi Bilinear
            x_new = a[0] + a[1]*j + a[2]*i + a[3]*j*i
            y_new = b[0] + b[1]*j + b[2]*i + b[3]*j*i

            # Lấy giá trị pixel từ ảnh gốc (Nearest Neighbor)
            x_new_int = int(np.round(x_new))
            y_new_int = int(np.round(y_new))

            # Đảm bảo tọa độ mới nằm trong giới hạn của ảnh
            if 0 <= x_new_int < width and 0 <= y_new_int < height:
                output[i, j] = image[y_new_int, x_new_int]

    return output
```

```python
# ======================== Làm mịn ảnh (image smoothing) ======================== #

# Averaging Filter
def averaging_filter(image, kernel_size):
    pad = kernel_size // 2
    padded_image = np.pad(image, pad, mode='constant', constant_values=0)
    output = np.zeros_like(image)
    for i in range(pad, padded_image.shape[0] - pad):
        for j in range(pad, padded_image.shape[1] - pad):
            region = padded_image[i-pad:i+pad+1, j-pad:j+pad+1]
            output[i-pad, j-pad] = np.sum(region) / (kernel_size * kernel_size)
    return output

# Gaussian Filter
def gaussian_kernel(size, sigma):
    k = size // 2
    kernel = np.zeros((size, size), dtype=np.float32)
    for i in range(size):
        for j in range(size):
            x, y = i - k, j - k
            kernel[i, j] = np.exp(-(x**2 + y**2) / (2 * sigma**2))
    kernel /= np.sum(kernel)
    return kernel

def gaussian_filter(image, kernel):
    pad = kernel.shape[0] // 2
    padded_image = np.pad(image, pad, mode='constant', constant_values=0)
    output = np.zeros_like(image)
    for i in range(pad, padded_image.shape[0] - pad):
        for j in range(pad, padded_image.shape[1] - pad):
            region = padded_image[i-pad:i+pad+1, j-pad:j+pad+1]
            output[i-pad, j-pad] = np.sum(region * kernel)
    return output
```

```python
# Median Filter
def median_filter(image, kernel_size):
    pad = kernel_size // 2
    padded_image = np.pad(image, pad, mode='constant', constant_values=0)
    output = np.zeros_like(image)
    for i in range(pad, padded_image.shape[0] - pad):
        for j in range(pad, padded_image.shape[1] - pad):
            region = padded_image[i-pad:i+pad+1, j-pad:j+pad+1]
            output[i-pad, j-pad] = np.median(region)
    return output

# ======================== Làm mịn ảnh (image blurring) ======================== #

def gaussian_blur(image, kernel):
    """
    Áp dụng Gaussian Blur cho ảnh.
    Args:
        image: Ảnh đầu vào (numpy array, grayscale).
        kernel: Kernel Gaussian (numpy array).
    Returns:
        Ảnh đã làm mờ (numpy array).
    """
    pad = kernel.shape[0] // 2
    padded_image = np.pad(image, pad, mode='constant', constant_values=0)
    output = np.zeros_like(image)

    for i in range(pad, padded_image.shape[0] - pad):
        for j in range(pad, padded_image.shape[1] - pad):
            region = padded_image[i-pad:i+pad+1, j-pad:j+pad+1]
            output[i-pad, j-pad] = np.sum(region * kernel)

    return output
```

```python
# ========================= Menu Chọn ========================= #
def safe_input(prompt):
    sys.stdout.write(prompt)
    sys.stdout.flush()
    return sys.stdin.readline().strip()

def menu():
    print("\nChọn chức năng:")
    print("1. Biến đổi màu")
    print("2. Biến đổi hình học")
    print("3. Làm mịn ảnh")
    print("4. Thoát")
    return int(input("Nhập lựa chọn của bạn: "))
```

```python
def color_transform_menu(image):
    print("\nChọn loại biến đổi màu:")
    print("1. Linear Mapping")
    print("2. Logarithmic Mapping")
    print("3. Exponential Mapping")
    print("4. Histogram Equalization")
    choice = int(input("Nhập lựa chọn: "))
    if choice == 1:
        a = float(input("Nhập hệ số a: "))
        b = int(input("Nhập hệ số b: "))
        result = linear_mapping(image, a, b)
    elif choice == 2:
        c = float(input("Nhập hệ số c: "))
        result = log_mapping(image, c)
    elif choice == 3:
        c = float(input("Nhập hệ số c: "))
        result = exp_mapping(image, c)
    elif choice == 4:
        result = histogram_equalization(image)
    else:
        print("Lựa chọn không hợp lệ.")
        return
    plt.imshow(result, cmap='gray')
    plt.title("Kết quả")
    plt.show()
```

```python
def geometric_transform_menu(image):
    print("\nChọn loại biến đổi hình học:")
    print("1. Affine Transformation")
    print("2. Resize (Nearest Neighbor)")
    print("3. Resize (Linear Interpolation)")
    print("4. Bilinear Transform")  # Tùy chọn mới
    choice = int(input("Nhập lựa chọn: "))
    if choice == 1:
        a = list(map(float, input("Nhập tham số a (3 số): ").split()))
        b = list(map(float, input("Nhập tham số b (3 số): ").split()))
        result = affine_transform(image, a, b)
    elif choice == 2:
        scale_x = float(input("Nhập hệ số scale_x: "))
        scale_y = float(input("Nhập hệ số scale_y: "))
        result = nearest_neighbor_resize(image, scale_x, scale_y)
    elif choice == 3:
        scale_x = float(input("Nhập hệ số scale_x: "))
        scale_y = float(input("Nhập hệ số scale_y: "))
        result = linear_interpolation_resize(image, scale_x, scale_y)
    elif choice == 4:  # Xử lý Bilinear Transform
        a = list(map(float, input("Nhập tham số a (4 số): ").split()))
        b = list(map(float, input("Nhập tham số b (4 số): ").split()))
        result = bilinear_transform(image, a, b)
    else:
        print("Lựa chọn không hợp lệ.")
        return
    plt.imshow(result, cmap='gray')
    plt.title("Kết quả")
    plt.show()
```

```python
def smoothing_menu(image):
    print("\nChọn loại làm mịn ảnh:")
    print("1. Averaging Filter")
    print("2. Gaussian Filter (Custom Kernel)")
    print("3. Median Filter")
    print("4. Gaussian Blur (Predefined Kernel)")  # Thêm tùy chọn Gaussian Blur
    choice = int(input("Nhập lựa chọn: "))
    if choice == 1:
        kernel_size = int(input("Nhập kích thước kernel: "))
        result = averaging_filter(image, kernel_size)
    elif choice == 2:
        kernel_size = int(input("Nhập kích thước kernel: "))
        sigma = float(input("Nhập sigma: "))
        kernel = gaussian_kernel(kernel_size, sigma)
        result = gaussian_filter(image, kernel)
    elif choice == 3:
        kernel_size = int(input("Nhập kích thước kernel: "))
        result = median_filter(image, kernel_size)
    elif choice == 4:  # Xử lý Gaussian Blur
        kernel_size = int(input("Nhập kích thước kernel (lẻ, ví dụ 3, 5, 7): "))
        sigma = float(input("Nhập sigma: "))
        kernel = gaussian_kernel(kernel_size, sigma)
        result = gaussian_blur(image, kernel)
    else:
        print("Lựa chọn không hợp lệ.")
        return
    plt.imshow(result, cmap='gray')
    plt.title("Kết quả")
    plt.show()
```

```python
# Chạy chương trình chính
def main():
    image_path = "Lenna.jpg"
    image = read_image(image_path)
    while True:
        choice = menu()
        if choice == 1:
            color_transform_menu(image)
        elif choice == 2:
            geometric_transform_menu(image)
        elif choice == 3:
            smoothing_menu(image)
        elif choice == 4:
            print("Thoát chương trình.")
            break
        else:
            print("Lựa chọn không hợp lệ.")

if __name__ == "__main__":
    main()
```

III. Summarization of the usage

This program allows users to perform various image processing tasks on the Lenna image, including color transformations, geometric transformations, and image smoothing.

1. **Color Transformations**:
   - Users can apply different transformations to modify the image's pixel values:
     - **Linear Mapping**: Scales and shifts pixel values.
     - **Logarithmic Mapping**: Compresses pixel values to enhance dark areas.
     - **Exponential Mapping**: Expands pixel values to brighten the image.
     - **Histogram Equalization**: Improves contrast by redistributing pixel intensities.
2. **Geometric Transformations**:

- Users can modify the image's geometry:
  - **Affine Transformation**: Applies linear transformations (rotation, translation, scaling).
  - **Resize (Nearest Neighbor)**: Resizes the image using the nearest neighbor method.
  - **Resize (Linear Interpolation)**: Resizes using linear interpolation for smoother scaling.
  - **Bilinear Transform**: Uses bilinear interpolation to resize the image with more accuracy.

3. **Image Smoothing**:
   - Users can smooth the image using different filters:
     - **Averaging Filter**: Smooths the image using a simple average filter.
     - **Gaussian Filter**: Applies a Gaussian filter to reduce noise and blur the image.
     - **Median Filter**: Uses the median of neighboring pixels to smooth the image.
     - **Gaussian Blur**: Applies a predefined Gaussian blur to soften the image.

The program's user interface allows selecting from a variety of image processing techniques through a menu-driven system. The processed image is displayed after each operation, providing visual feedback for users to explore different transformations and filters.

Function usage sumarization:

1. **read_image(file_path)**: Reads an image from the given file path in grayscale using OpenCV.
2. **linear_mapping(image, a=1.0, b=0)**: Applies a linear transformation to the image by scaling and shifting pixel values.
3. **log_mapping(image, c=1.0)**: Applies a logarithmic transformation to the image to compress the pixel values.
4. **exp_mapping(image, c=1.0)**: Applies an exponential transformation to the image to expand the pixel values.
5. **histogram_equalization(image)**: Performs histogram equalization to adjust the image contrast by redistributing pixel intensities.
6. **affine_transform(image, a, b)**: Applies an affine transformation to the image by modifying the coordinates of the pixels based on a set of parameters.
7. **nearest_neighbor_resize(image, scale_x, scale_y)**: Resizes the image using the nearest neighbor interpolation method, which selects the closest pixel for resizing.
8. **linear_interpolation_resize(image, scale_x, scale_y)**: Resizes the image using linear interpolation, where pixel values are computed by averaging the neighboring pixel values.
9. **bilinear_transform(image, a, b)**: Applies a bilinear transformation to the image by interpolating pixel values using a set of parameters.
10. **averaging_filter(image, kernel_size)**: Applies an averaging filter (box filter) to the image to smooth it by averaging the pixels within a given kernel size.
11. **gaussian_kernel(size, sigma)**: Generates a Gaussian kernel, which is used to apply Gaussian filtering to smooth the image.
12. **gaussian_filter(image, kernel)**: Applies a Gaussian filter to the image using the specified Gaussian kernel for smoothing.

13. **median_filter(image, kernel_size)**: Applies a median filter to the image to remove noise by replacing each pixel with the median of its neighbors.
14. **gaussian_blur(image, kernel)**: Applies Gaussian blur to the image using a Gaussian kernel to create a blur effect.

- Menu Functions:

These are the functions used to display menus and handle user choices for different image processing tasks:

1. **menu()**: Displays the main menu with options to select different image processing tasks. It gives the user four options: Color Transformations, Geometric Transformations, Image Smoothing, and Exit.
2. **color_transform_menu(image)**: Displays a sub-menu for color transformations where the user can choose from linear mapping, logarithmic mapping, exponential mapping, and histogram equalization.
3. **geometric_transform_menu(image)**: Displays a sub-menu for geometric transformations where the user can choose from affine transformation, resizing using nearest neighbor or linear interpolation, and bilinear transformation.
4. **smoothing_menu(image)**: Displays a sub-menu for image smoothing where the user can choose from averaging filter, Gaussian filter (custom kernel), median filter, and Gaussian blur (predefined kernel).
These functions handle user input, call the respective transformation or filtering function, and display the results of the image manipulation.

IV. Implementation:

Description of Methods and Pseudo code

1. linear_mapping(image, a=1.0, b=0)

Purpose: Applies a linear transformation to the image by scaling (multiplying by a) and shifting (adding b) pixel values.

Details:

- For each pixel, calculate new_pixel = a * old_pixel + b.
- The result is clipped to ensure values remain within the valid pixel range [0, 255].

Pseudo code:

```
For each pixel (i, j) in the image:
    output[i, j] = a * image[i, j] + b
Clip output to the range [0, 255]
Return output
```

2. log_mapping(image, c=1.0)

Purpose: Applies logarithmic mapping to enhance the darker regions of the image.

Details:

- For each pixel, calculate new_pixel = c * log(1 + old_pixel).
- The result is clipped to ensure values remain within the valid pixel range [0, 255].

Pseudo code:

For each pixel (i, j) in the image:

  output[i, j] = c * log(1 + image[i, j])

Clip output to the range [0, 255]

Return output

3. exp_mapping(image, c=1.0)

Purpose: Applies exponential mapping to brighten the image, especially the darker regions.
Details:

- For each pixel, calculate new_pixel = c * (exp(old_pixel / 255.0) - 1) * 255.
- The result is clipped to ensure values remain within the valid pixel range [0, 255].

Pseudo code:

For each pixel (i, j) in the image:

  output[i, j] = c * (exp(image[i, j] / 255.0) - 1) * 255

Clip output to the range [0, 255]

Return output

4. histogram_equalization(image)

Purpose: Equalizes the histogram of the image to improve its contrast.
Details:

- Calculate the histogram of the image.
- Derive the cumulative distribution function (CDF) and scale it to the [0, 255] range.
- Map each pixel value to its corresponding value in the equalized CDF.

Pseudo code:

Calculate histogram of the image

Calculate cumulative distribution function (CDF) of the histogram

Scale CDF to range [0, 255]

For each pixel (i, j) in the image:

  output[i, j] = CDF[image[i, j]]

Return output

5. affine_transform(image, a, b)

Purpose: Applies an affine transformation (rotation, scaling, or translation) to the image.
Details:

- For each pixel, calculate the new coordinates using the formula x_new = a[0] + a[1] * x + a[2] * y and y_new = b[0] + b[1] * x + b[2] * y.
- Assign the pixel value to the new coordinates.

Pseudo code:

```
For each pixel (i, j) in the image:
  x_new = a[0] + a[1] * j + a[2] * i
  y_new = b[0] + b[1] * j + b[2] * i
  If x_new and y_new are within image boundaries:
    output[y_new, x_new] = image[i, j]
Return output
```

6. nearest_neighbor_resize(image, scale_x, scale_y)

Purpose: Resizes the image using the nearest neighbor interpolation method.
Details:

- Calculate the new image dimensions as new_width = scale_x * old_width and new_height = scale_y * old_height.
- For each pixel in the resized image, find the nearest pixel in the original image and assign its value.

Pseudo code:

```
Calculate new height = height * scale_y
Calculate new width = width * scale_x
For each pixel (i, j) in the resized image:
  x = floor(j / scale_x)
  y = floor(i / scale_y)
  output[i, j] = image[y, x]
Return output
```

7. linear_interpolation_resize(image, scale_x, scale_y)

Purpose: Resizes the image using linear interpolation to produce smoother results.
Details:

- For each pixel in the resized image, calculate its position in the original image.
- Interpolate the surrounding pixels along both axes (x and y) using linear interpolation

Pseudo code:

```
Calculate new height = height * scale_y
```

```
Calculate new width = width * scale_x
For each pixel (i, j) in the resized image:
   x = j / scale_x
   y = i / scale_y
   x0 = floor(x)
   x1 = min(x0 + 1, width - 1)
   y0 = floor(y)
   y1 = min(y0 + 1, height - 1)
   alpha = x - x0
   beta = y - y0
   output[i, j] = (1 - alpha) * (1 - beta) * image[y0, x0]
             + alpha * (1 - beta) * image[y0, x1]
             + (1 - alpha) * beta * image[y1, x0]
             + alpha * beta * image[y1, x1]
Return output
```

8. bilinear_transform(image, a, b)

Purpose: Performs bilinear interpolation for resizing or transforming the image.
Details:

- Calculate the new pixel position using the formula x_new = a[0] + a[1] * j + a[2] * i + a[3] * j * i and y_new = b[0] + b[1] * j + b[2] * i + b[3] * j * i.
- Interpolate the pixel value based on the surrounding pixels.

Pseudo code:

```
For each pixel (i, j) in the image:
   x_new = a[0] + a[1] * j + a[2] * i + a[3] * j * i
   y_new = b[0] + b[1] * j + b[2] * i + b[3] * j * i
   If x_new and y_new are within image boundaries:
      x_new_int = round(x_new)
      y_new_int = round(y_new)
      output[i, j] = image[y_new_int, x_new_int]
Return output
```

9. averaging_filter(image, kernel_size)

Purpose: Applies an averaging filter (box blur) to smooth the image.
Details:

- For each pixel, calculate the average value of the surrounding pixels using a square kernel of size kernel_size.
- The result is applied to each pixel in the image.

Pseudo code:

For each pixel (i, j) in the image:

Define a region around (i, j) of size kernel_size x kernel_size

output[i, j] = sum(region) / (kernel_size * kernel_size)

Return output

10. gaussian_kernel(size, sigma)

Purpose: Generates a Gaussian kernel for use in Gaussian filtering.
Details:

- Calculate the Gaussian function for each pixel in the kernel based on the size (size) and standard deviation (sigma).
- Normalize the kernel so that the sum of all values equals 1.

Pseudo code:

For each (i, j) in the kernel:

x = i - size // 2

y = j - size // 2

kernel[i, j] = exp(-(x^2 + y^2) / (2 * sigma^2))

Normalize kernel so that sum(kernel) = 1

Return kernel

11. **gaussian_filter(image, kernel)**

Purpose: Applies a Gaussian filter to the image to blur it.
Details:

- For each pixel, calculate the weighted sum of the surrounding pixels using the Gaussian kernel.
- The result is applied to each pixel in the image.

Pseudo code:

For each pixel (i, j) in the image:

Define a region around (i, j) of size equal to the kernel

output[i, j] = sum(region * kernel)

Return output

12. median_filter(image, kernel_size)

Purpose: Applies a median filter to smooth the image and reduce noise.
Details:

- For each pixel, replace its value with the median of the surrounding pixel values within a kernel of size kernel_size.

Pseudo code:

For each pixel (i, j) in the image:

Define a region around (i, j) of size kernel_size x kernel_size

output[i, j] = median(region)

Return output

13. **gaussian_blur(image, kernel)**
Purpose: Applies Gaussian blur to the image using a predefined Gaussian kernel.
Details:

- For each pixel, calculate the weighted sum of neighboring pixels using the Gaussian kernel.
- The result is applied to each pixel in the image

Pseudo code:

For each pixel (i, j) in the image:

Define a region around (i, j) of size equal to the kernel

output[i, j] = sum(region * kernel)

Return output