Assignment No. 2

Name: Om Dalvi
PRN: 122B1B061
Div: A

Problem Statement

The task is to explore and apply various **mask (kernel) processing techniques** to a sample image using Python and OpenCV. The aim is to understand how different kernels affect an image's pixel values and to observe the resultant changes in terms of smoothing, noise reduction, and morphological transformations.

Objective

- To Learn how convolution kernels (masks) are applied to images to modify pixel values based on their neighborhood.
- To Gain hands-on experience in implementing several filter techniques including smoothing filters, noise reduction filters, and morphological operations.
- To Observe and compare the effects of different filters on a sample grayscale image using Python.

Outcome

After completing this assignment, you will be able to:

- Implement and apply average, weighted average, Gaussian, median, min (erosion), and max (dilation) filters using OpenCV.
- Understand the differences between these filters and their practical implications in image processing.
- Visualize the original and processed images side by side, with proper format adjustments (e.g., RGB conversion, increased display size).

Theory: Mask Processing Techniques

Mask (Kernel):

A mask, also known as a kernel in image processing, is a small matrix of numbers used to modify or extract features from an image. It defines the weights that are applied to a pixel and its neighboring pixels during convolution. Typically, the mask is a square matrix (e.g., 3×3, 5×5) where each element corresponds to a weight used to calculate a new pixel value. This process helps in emphasizing or de-emphasizing specific image features such as edges, textures, or noise.

Mask Processing:
Mask processing refers to the technique of applying a mask (kernel) to an image through a convolution operation. In this process, the mask slides over the image, and at each position, a weighted sum of the pixel values covered by the mask is computed. The resulting sum replaces the central pixel's value, thus altering the image based on the characteristics defined by the mask. This technique is fundamental in various image processing tasks such as smoothing (blurring), sharpening, edge detection, and noise reduction.

The following filter techniques are covered in this assignment:

1. **Average Filter (Box Filter/Mean Filter)**
2. **Weighted Average Filter**
3. **Gaussian Filter**
4. **Median Filter**
5. **Min Filter (Erosion)**
6. **Max Filter (Dilation)**

Each filter uses a specific kernel to process the image, and the choice of kernel determines the effect on the image. The assignment demonstrates how to apply these filters on a sample grayscale image.

---

1. Average Filter (Box Filter/Mean Filter)

Description

The average filter smooths an image by replacing each pixel value with the mean of its neighboring pixels. A 5x5 kernel is commonly used, which means that for each pixel, the filter computes the average value of the pixels in a 5x5 window around it.

Implementation

```
# Define a function for each mask processing technique
def average_filter(image):
    kernel = np.ones((5, 5), np.float32) / 25  # 5x5 kernel with mean
value
```

```
    return cv2.filter2D(image, -1, kernel)
```

Example

- **Input:** A grayscale image.
- **Output:** A blurred image where details are smoothed out by averaging neighboring pixels.

---

2. Weighted Average Filter

Description

Unlike the average filter, the weighted average filter assigns different weights to each pixel in the neighborhood. Typically, pixels closer to the center have higher weights, meaning the center pixel contributes more to the output. This often results in a better preservation of edges compared to the simple average filter.

Implementation

```
def weighted_average_filter(image):
    kernel = np.array([[1, 2, 3, 2, 1],
                       [2, 4, 6, 4, 2],
                       [3, 6, 9, 6, 3],
                       [2, 4, 6, 4, 2],
                       [1, 2, 3, 2, 1]], dtype=np.float32) / 81
    return cv2.filter2D(image, -1, kernel)
```

Example

- **Input:** A grayscale image.
- **Output:** A filtered image where the central part is given more importance, resulting in a smoother transition with less blurring of edges.

---

3. Gaussian Filter

Description

The Gaussian filter smooths an image by convolving it with a Gaussian function. The weights are determined by a normal distribution, which provides a natural, bell-curve weighting that minimizes edge artifacts. This filter is widely used for noise reduction and pre-processing for edge detection.

Implementation

```python
def gaussian_filter(image):
    return cv2.GaussianBlur(image, (5, 5), 0)  # 5x5 kernel with standard
deviation 0
```

Example

- **Input:** A grayscale image.
- **Output:** A softly blurred image that retains important structural details while reducing high-frequency noise.

---

4. Median Filter

Description

The median filter replaces each pixel with the median value of the pixel values in its neighborhood. This filter is particularly effective at reducing salt-and-pepper noise without blurring the edges too much.

Implementation

```python
def median_filter(image):
    return cv2.medianBlur(image, 5)  # 5x5 kernel
```

Example

- **Input:** A grayscale image with salt-and-pepper noise.
- **Output:** A noise-reduced image where the median calculation helps preserve edges while eliminating isolated noise

- spikes.

---

5. Min Filter (Erosion)

Description

The min filter, implemented via the erosion operation in image processing, replaces each pixel with the minimum value found in its neighborhood. This operation can be useful for removing small bright details and is often used in morphological operations.

Implementation

```
def min_filter(image):
    kernel = np.ones((5, 5), np.uint8)
    return cv2.erode(image, kernel)  # Erosion is equivalent to min filter
```

Example

- **Input:** A grayscale image.
- **Output:** An image where bright regions are shrunk, emphasizing the darker parts of the image.

---

6. Max Filter (Dilation)

Description

The max filter, implemented using the dilation operation, replaces each pixel with the maximum value in its neighborhood. It is often used to enlarge bright regions or fill in small dark holes in an image.

Implementation

```
def max_filter(image):
    kernel = np.ones((5, 5), np.uint8)
    return cv2.dilate(image, kernel)  # Dilation is equivalent to max 
filter
```

Example: **Input:** A grayscale image.

Output: An image where bright regions become more pronounced and expanded.

Conclusion

In this assignment, we applied six different mask processing techniques to a sample image. Each technique utilizes a specific kernel to modify pixel values based on its neighborhood, resulting in various effects—from smoothing and noise reduction to morphological transformations. These methods form the basis of many advanced image processing tasks and are essential tools in computer vision.

Feel free to modify the kernel sizes and weights or experiment with different image inputs to explore how each mask processing technique affects the outcome.