Homework 1: Image Enhancement Using Spatial Filters

Part I. Implementation

- I use zero padding to ensure that the image retains the same size after applying convolution.

```python
def padding(input_img, kernel_size):
    ############### YOUR CODE STARTS HERE ###############
    """Applies zero padding to the input image."""
    '''
        選擇使用 zero padding 進行圖像處理
        np.pad mode='constant' 使用常數 edge: 使用邊緣pixel 來填充 reflect:進行鏡像的反射

    '''
    pad_size = kernel_size // 2
    output_img = np.pad(input_img, ((pad_size, pad_size), (pad_size, pad_size), (0, 0)), mode='constant', constant_values=0)

    ############### YOUR CODE ENDS HERE #################
    return output_img
```

- This code performs a convolution on an input image with a specified kernel by cropping the original image to match the kernel, then performing element-wise multiplication and summation.

```python
def convolution(input_img, kernel):
    ################ YOUR CODE STARTS HERE ################
    """Applies a convolution operation using the given kernel."""
    kernel_size = kernel.shape[0]
    img_h,img_w,img_c = input_img.shape
    padded_img = padding(input_img, kernel_size)
    output_img = np.zeros_like(input_img)

    for i in range(img_h):
        for j in range(img_w):
            for c in range(img_c):  # Iterate over the color channels
                region = padded_img[i:i+kernel_size, j:j+kernel_size, c]
                output_img[i, j, c] = np.sum(region * kernel) # element-wise multiplication and summation

    ################ YOUR CODE ENDS HERE ##################
    return output_img
```

- This function generates a Gaussian kernel of specified `size` and `sigma`. It first creates a 1D Gaussian distribution, then uses the outer product to form a 2D Gaussian kernel, and normalizes it so all values sum to 1.

```python
def gaussian_kernel(size, sigma):
    """Generates a Gaussian kernel."""
    #高斯kernel 給定size 和 sigma 生成的kernel是唯一的
    kernel_1d = np.linspace(-(size // 2), size // 2, size) #生成一維數組 asuume size=5 -> [-2, -1, 0, 1, 2]
    gaussian = np.exp(-0.5 * (kernel_1d ** 2) / sigma ** 2) # 套用高斯函式的function
    kernel_1d = gaussian / gaussian.sum() # normalization
    kernel_2d = np.outer(kernel_1d, kernel_1d) # 透過 外積來生成二維高斯函數
    return kernel_2d / kernel_2d.sum() # normalization
```

- Given the parameter of sigma and kernel.And then doing the convolution.

```python
def gaussian_filter(input_img):
    ############### YOUR CODE STARTS HERE ###############
    """Applies a Gaussian filter to the input image."""
    sigma = 0.5  # 0.5 2 5
    size = 3  # 3 5 7
    kernel = gaussian_kernel(size, sigma)
    ############### YOUR CODE ENDS HERE #################
    return convolution(input_img, kernel)
```

- Sort the pixels, select the median, and replace the middle pixel with the median value.

```python
def median_filter(input_img):
    ############### YOUR CODE STARTS HERE ###############
    """Applies a Median filter to the input image."""
    '''
        1. 將圖像進行 zero padding
        2. 9個pixel 排列 取中位數
        3. 中間改成中位數的value

        優點 : 去躁效果好 保留邊緣細節不錯
        缺點 : 計算量大 對高斯噪聲效果不好
    '''
    size = 3  # kernel_size
    padded_img = padding(input_img, size)
    output_img = np.zeros_like(input_img)

    for i in range(input_img.shape[0]):
        for j in range(input_img.shape[1]):
            for c in range(input_img.shape[2]):
                region = padded_img[i:i+size, j:j+size, c]  # Get the region of K*K size
                output_img[i, j, c] = np.median(region)
    ############### YOUR CODE ENDS HERE #################
    return output_img
```

- Use the specific kernel to apply the convolution.

```python
def laplacian_sharpening(input_img):
    ############### YOUR CODE STARTS HERE ###############
    """Applies Laplacian sharpening to the input image."""
    '''
        增強圖像的邊緣細節，使圖像看起來更銳利，用於增強圖像的邊緣
        缺點 : 不適合有大量噪聲的圖片 銳化過強可能放大噪聲
    '''

    # 8-connected Laplacian kernel
    kernel = np.array([[-1, -1, -1],
                       [-1,  9, -1],
                       [-1, -1, -1]])

    # 4-connected Laplacian kernel
    kernel = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, -1, 0]])

    ############### YOUR CODE ENDS HERE #################
    return convolution(input_img, kernel)
```

Part II. Results & Analysis

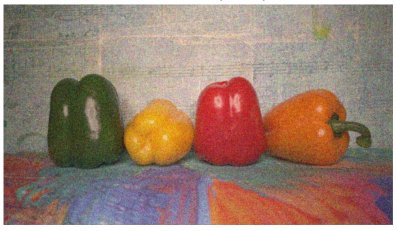
• Gaussian filter

- The **Gaussian filter** is a widely used smoothing filter in image processing, designed to reduce noise while preserving the overall structure of the image. It operates by applying a Gaussian function to compute the weights of the neighboring pixels for each pixel in the image. The closer a pixel is to the center, the higher the weight assigned to it, while more distant pixels receive lower weights.

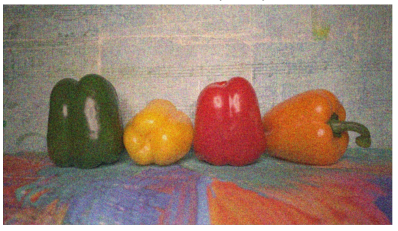
Different σ value : (0.5 vs. 2 vs. 5)

Gaussian (σ=0.5)

Gaussian (σ=2)



Gaussian (σ=5)



💡

- **σ = 0.5**: More details are preserved, and the smoothing effect is light. Noise is still quite noticeable, and the image sharpness is relatively high.
- **σ = 2**: The smoothing effect becomes more apparent, with reduced noise, but the edge details begin to blur.
- **σ = 5**: The strongest smoothing effect, with most details being blurred out. Noise is mostly eliminated, but the image becomes very smooth and soft.

Summary

- With a larger σ, the weights become more spread out, resulting in a smoother image but with a loss of detail.

Different filter size : (3 vs. 5 vs. 7)

- Image Part I :

Gaussian with kernel=3

Gaussian with kernel=5

Gaussian with kernel=7

💡

- **Gaussian Kernel (3×3)**:
  - Preserves more details, the image sharpness is higher, but the noise is also relatively noticeable.
- **Gaussian Kernel (5×5)**:
  - The smoothing effect is stronger, some details are blurred, and the noise is reduced.
- **Gaussian Kernel (7×7)**:
  - The strongest smoothing effect, noise is significantly reduced, but the image becomes very soft, and the edge details are noticeably blurred.

Summary

- We can see that a larger kernel smooths the image more effectively, but it also leads to increased blurring.

• Median filter

Different filter size : (3 vs. 5 vs. 7)

- Image Part I :

Median with kernel=3

Median with kernel=5

Median with kernel=7

💡

- **Median with kernel = 3**:
  - Minimal smoothing that removes salt-and-pepper noise while preserving image details and sharpness.
- **Median with kernel = 5**:
  - Moderate smoothing that effectively reduces noise while keeping some edge detail.
- **Median with kernel = 7**:
  - Strong smoothing, suitable for heavy noise, but it may blur edges and finer details.

Summary

A larger median kernel size increases smoothing and noise reduction, but at the cost of blurring finer details and edges. Smaller kernels preserve image sharpness and details while providing lighter noise reduction, making them suitable for minimal noise.

- **Smoothing Spaitial Filters**

Comparison of the Results:

Feature	Gaussian Filter	Median Filter
Principle	Smooths the image by applying weighted averages based on a Gaussian distribution to neighboring pixels	Sorts the pixels in a window and replaces the center pixel with the median value
Suitable Noise Type	Gaussian noise	Salt and pepper noise
Edge Preservation	Blurs edges to some extent, causing a loss of detail	Better at preserving edges without blurring them
Denoising Effect	Good at removing Gaussian noise	Excellent at removing salt and pepper noise
Image Blurring	Introduces global blurring, smoothing the entire image	Does not introduce global blur, only processes local extreme values
Computation	Relatively low, fast computation	Relatively high, requires sorting for each window
Image Smoothness	High overall smoothness, suitable for uniform noise	Local smoothing, only processes specific noisy points, preserves more detail
Handling of Image Borders	Effective at handling noise near the edges, no strong noise points	Well-suited for preserving edge details
Application Scenarios	Suitable for removing uniformly distributed noise like Gaussian noise	Suitable for removing discrete extreme noise like salt and pepper noise
Adjustability of Effect	Smoothing can be controlled by adjusting the standard deviation (σ)	Noise removal capacity can be controlled by adjusting window size

Summary:

- **Gaussian filter** is ideal for removing **Gaussian noise**, and works well for smoothing uniformly distributed noise, but it causes global blurring and may reduce edge details.
- **Median filter** is ideal for removing **salt and pepper noise**, effectively removing extreme noise points and preserving edge details, but requires more computation.

• Laplacian filter

Different laplacian filters : (4-connected vs. 8-connected)

0	-1	0
-1	5	-1
0	-1	0

Filter 1

-1	-1	-1
-1	9	-1
-1	-1	-1

Filter 2

- Image Part II:

4-connected Laplacian

8-connected Laplacian

💡

- **4-Connected Laplacian**:
  - This sharpening filter focuses on enhancing edges in the four main directions (up, down, left, right).
  - The result shows increased clarity around edges with minimal noise amplification, making it effective for subtle edge enhancement without much distortion.
- **8-Connected Laplacian**:
  - The 8-connected filter emphasizes all eight directions surrounding each pixel, resulting in stronger edge detection and enhancement.
  - This approach brings out more detail but also introduces more artifacts and noise in the image, making the edges appear more pronounced. However, some finer details might appear grainy due to amplified noise.

Summary

The **4-connected Laplacian** is more subtle, preserving image quality with fewer artifacts, while the **8-connected Laplacian** provides stronger edge enhancement but may introduce noticeable noise. The choice between them depends on the desired level of edge emphasis and tolerance for noise in the image.

Part III. Answer the questions

1. Please describe a problem you encountered and how you solved it.

Problem: One challenge I encountered was handling the image boundaries during the convolution operation. Since the convolution process requires neighboring pixels, edge pixels don't have enough surrounding pixels, which causes the operation to fail at the edges.

Solution: I solved this by using **zero padding**. This method adds extra zero pixels around the image borders, allowing the convolution to be performed even on the edge pixels, thus preventing any computation errors at the boundaries.

2. What padding method do you use, and does it have any disadvantages? If so, please suggest possible solutions to address them.

Padding method used: I used **zero padding** in the code, which pads the image boundaries with zero values.

Disadvantages: Zero padding can introduce unnatural transitions at the image edges because replacing edge pixels with zeros may result in blurred or disconnected edges, especially in smoothing operations like Gaussian filtering.

Solutions: Alternative padding methods can help reduce this issue:

- **Reflect padding**: Reflect the pixels along the border, which creates a more natural transition between the edge and the interior of the image.
- **Replicate padding**: Use the edge pixel values to pad the boundaries, reducing the abrupt transitions.
- **Circular padding**: Wrap the interior pixels around the edges, creating a smoother and more continuous padding.

3. What problems do you encounter when using Gaussian filter and median filter to denoise images? Please suggest possible solutions to address them.

Gaussian filter:

- **Problem**: While the Gaussian filter effectively removes noise, it also tends to blur the edges of the image since it applies global smoothing to the image, affecting both noise and important details like edges.
- **Solution**: An **edge-preserving filter** such as the bilateral filter or non-local means filter could be used. These filters smooth the image while preserving important edges, thus maintaining more image details.

Median filter:

- **Problem**: The median filter works well for removing "salt-and-pepper" noise but is less effective for Gaussian noise. Additionally, larger kernel sizes can result in a loss of fine details in the image.
- **Solution**: Depending on the noise type, different filters should be selected. For non-salt-and-pepper noise, other filters like the Gaussian filter may be more appropriate. Also, adjusting the kernel size to balance noise removal and detail preservation is important.

4. What problems do you encounter when using Laplacian filters to sharpen images? Please suggest possible solutions to address them.

Problem: When using the Laplacian filter for image sharpening, a common issue is **noise amplification**. Since the Laplacian filter enhances edges and high-frequency components, noise, which is also a high-frequency component, can be amplified, resulting in a noisy, over-sharpened image.

Solution: To address this, you can apply a smoothing filter like the Gaussian filter before sharpening with the Laplacian filter. This approach helps to reduce noise while still sharpening the important edges, leading to a more balanced result without amplifying the noise.