

Homework 1: Image Enhancement Using Spatial Filters

313443036 多媒體所 1 葉軒宇

Part I. Implementation (5%):

In this homework, we have to implement three different types of spatial filters.

1. Padding function

```
2 usages
def padding(input_img, kernel_size):
    """對輸入的影像進行padding，以便在之後進行convolution時可以處理照片的邊緣"""
    if isinstance(kernel_size, int):
        pad_h = pad_w = kernel_size // 2
    else:
        pad_h = kernel_size[0] // 2
        pad_w = kernel_size[1] // 2

    if len(input_img.shape) == 3:
        padded_img = np.pad(input_img,
                             pad_width=((pad_h, pad_h), (pad_w, pad_w), (0, 0)),
                             mode='edge')
    else:
        padded_img = np.pad(input_img,
                             pad_width=((pad_h, pad_h), (pad_w, pad_w)),
                             mode='edge')

    return padded_img
```

This function adds padding to the input image to handle border effect during convolution section. I used edge padding, which replicates the border pixels. Besides, it also check if the image is grayscale or color and pads accordingly.

2. Convolution function

```

def convolution(input_img, kernel):
    """對輸入的照片進行2D-convolution"""
    #h為高度，w為寬度，c為channel數
    if len(input_img.shape) == 3:
        h, w, c = input_img.shape
    else:
        h, w = input_img.shape
        c = 1
        input_img = input_img.reshape(h, w, 1)

    kernel = np.flipud(np.fliplr(kernel))
    kernel_h, kernel_w = kernel.shape
    padded_img = padding(input_img, kernel_size=(kernel_h, kernel_w))
    output_img = np.zeros_like(input_img)

    for i in range(h):
        for j in range(w):
            for k in range(c):
                output_img[i, j, k] = np.sum(
                    padded_img[i:i + kernel_h, j:j + kernel_w, k] * kernel
                )

    return output_img

```

This function performs 2D-convolution between input image and kernel. It iterates over each pixel and channel to complete the convolution operation.

3. Gaussian Filter

```

def gaussian_filter(input_img, sigma=1.0, kernel_size=3):
    """對輸入的照片使用 gaussian filter來減少noise"""
    # 建立一個Gaussian kernel
    kernel = np.zeros((kernel_size, kernel_size))
    center = kernel_size // 2

    for i in range(kernel_size):
        for j in range(kernel_size):
            x = i - center
            y = j - center
            kernel[i, j] = ((1 / (2 * np.pi * sigma ** 2)) *
                            np.exp(-(x ** 2 + y ** 2) / (2 * sigma ** 2)))

    # Normalize kernel來避免改變整體照片的總量度
    kernel = kernel / np.sum(kernel)

    return convolution(input_img, kernel)

```

This function will generate Gaussian kernel based on the argument you set(sigma and kernel size) and applied to the image. There are one thing to mention is I've normalized the kernel to prevent overflow.

4. Median filter

```
def median_filter(input_img, kernel_size=3):
    """對輸入的照片使用 median filter來減少noise"""
    if len(input_img.shape) == 3:
        h, w, c = input_img.shape
    else:
        h, w = input_img.shape
        c = 1
        input_img = input_img.reshape(h, w, 1)

    padded_img = padding(input_img, kernel_size=(kernel_size, kernel_size))
    output_img = np.zeros_like(input_img)
    pad = kernel_size // 2

    for i in range(h):
        for j in range(w):
            for k in range(c):
                window = padded_img[i:i + kernel_size, j:j + kernel_size, k]
                output_img[i, j, k] = np.median(window)

    return output_img
```

This filter reduces noise by replacing each pixel's value with the median's value of its neighborhood.

5. Laplacian sharpening

```
def laplacian_sharpening(input_img, kernel_type=1):
    """對輸入照片使用 laplacian sharpening，根據需求不同可以選擇type 1 or 2的kernel"""
    if kernel_type == 1:
        kernel = np.array([
            [0, -1, 0],
            [-1, 5, -1],
            [0, -1, 0]
        ])
    else:
        kernel = np.array([
            [-1, -1, -1],
            [-1, 9, -1],
            [-1, -1, -1]
        ])

    sharpened = convolution(input_img, kernel)

    # 為了避免 overflow，使用clip
    sharpened = np.clip(sharpened, a_min=0, a_max=255)

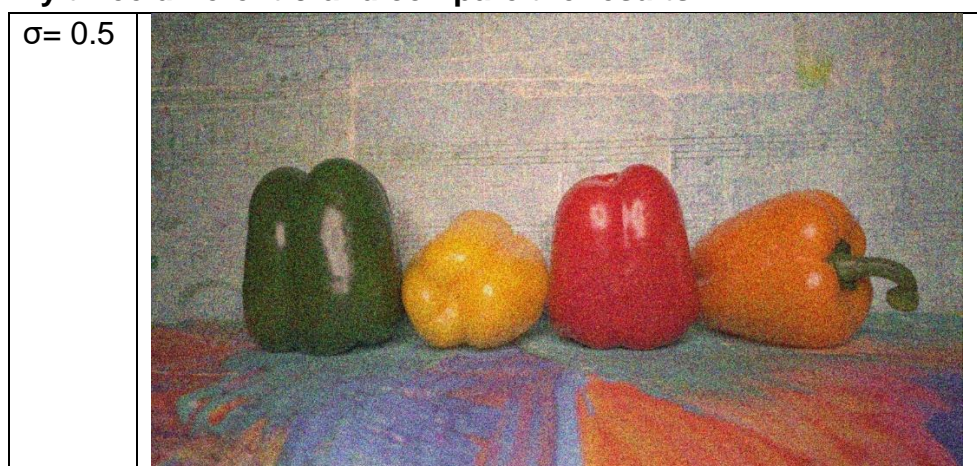
    return sharpened.astype(np.uint8)
```



This function enhances the edges and fine details in the image by using Laplacian kernels, there are two types of kernel can be selected. At the end of the function, I've make sure that the output pixel' value are within the valid range(0~255).

Part II. Results & Analysis (10%):

Please provide your **observations** and **analysis** for the following parts.


- Gaussian filter
 - Try three different σ and compare the results





| | |
|----------------|---|
| $\sigma = 1.0$ |  |
| $\sigma = 2.0$ |  |

As the value of σ increases, stronger blurring will be. And larger sigma may cause to lose some important detail.

- **Try three different filter sizes and compare the results.**


| | |
|--------------------|--|
| Kernel size = 3 |  |
|--------------------|--|

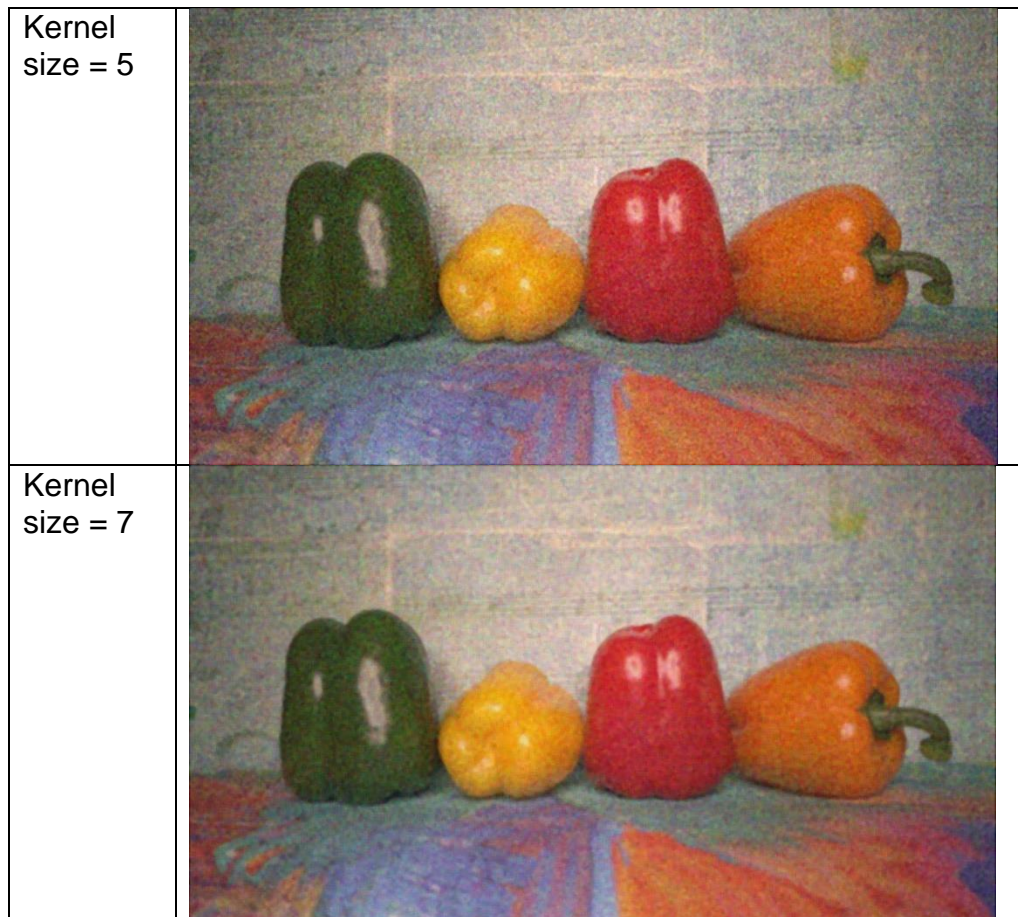
| | |
|-----------------|---|
| Kernel size = 5 |  |
| Kernel size = 7 |  |

Larger kernel size consider a boarder neighborhood, resulting in greater smoothing, but excessively large kernel may over-blur the image.

- **Median filter**

- **Try three different filter sizes and compare the results.**

| | |
|-----------------|--|
| Kernel size = 3 |  |
|-----------------|--|



Median filter is quite effective in reduce noise. Same problem as in gaussian filter, larger kernel may lead to lose some details.

- **Smoothing Spatial Filters**

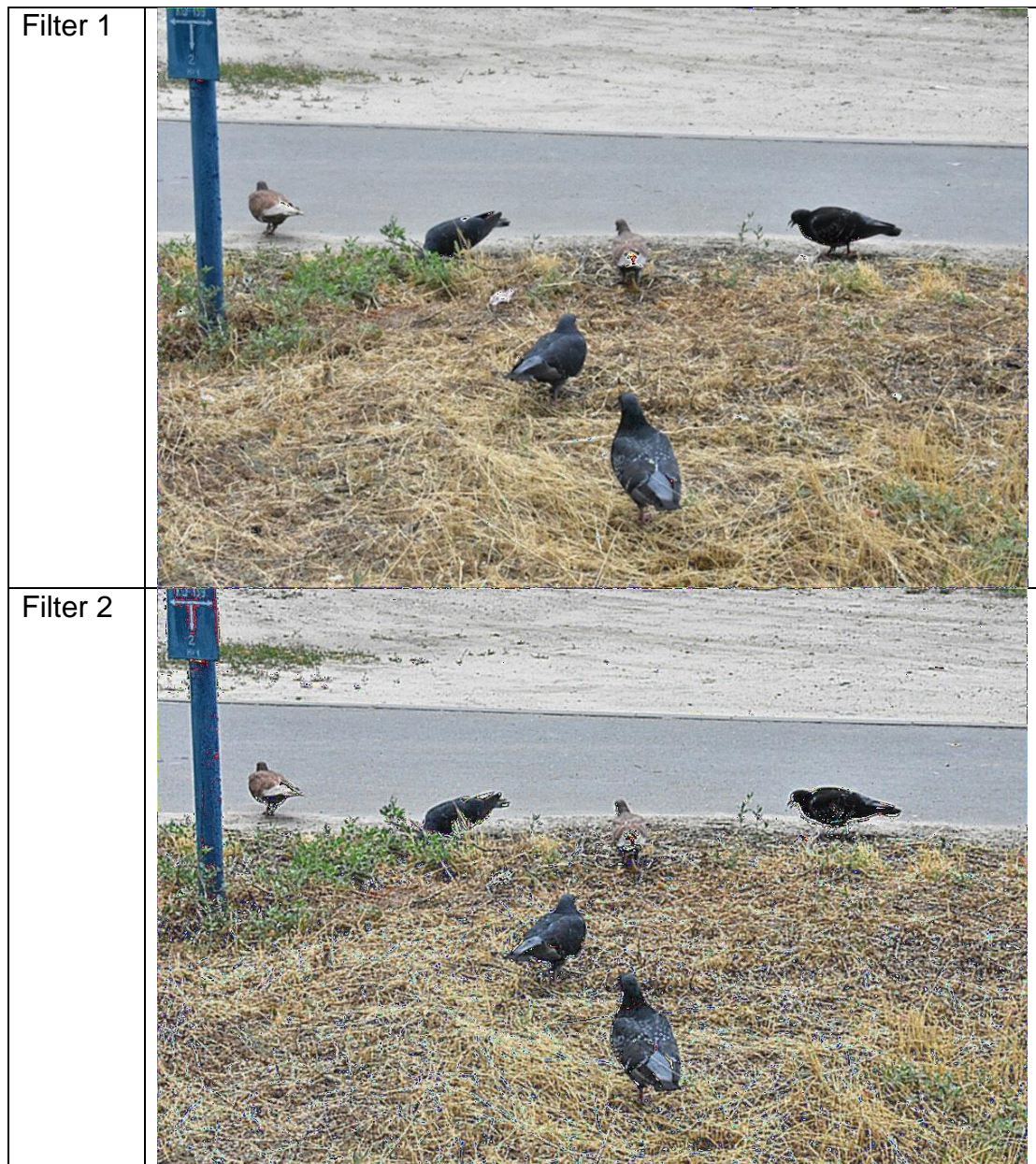
- **Compare the results of Gaussian filter and median filter.**

Median filter are best for reducing salt and pepper noise, Gaussian filter are best for reducing random noise. So choose of the filter are based on which kind of noise your image are. In this case, in my opinion, I think Gaussian filter with large sigma and middle kernel do better than median filter.

- **Laplacian filter**

- **Compare the results of two laplacian filters.**

| | | | | | |
|----------|----|----|----------|----|----|
| 0 | -1 | 0 | -1 | -1 | -1 |
| -1 | 5 | -1 | -1 | 9 | -1 |
| 0 | -1 | 0 | -1 | -1 | -1 |
| Filter 1 | | | Filter 2 | | |



Filter 1 enhances edges moderately; image appears sharper without significant artifacts. Filter 2 did strong edge enhancements.

Filter 2 do more sharpness than filter 1, however, it also amplify noise and cause halos and ringing around the edge. This makes it a little bit weird. On the contrary, although filter 1 doesn't sharpness that much, but the output picture seems more naturally.

Part III. Answer the questions (15%):

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

During the implementation of the convolution function, I faced several challenges regarding the processing of image boundaries and maintaining proper output dimensions. Initially, the processed images exhibited noticeable distortions along the edges and dimensional inconsistencies.

After analysis, I identified that the root cause was in the padding implementation, particularly in its handling of color image boundaries. So I change to use edge padding rather than zero padding.

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

I used edge padding. It may introduce bias in the filtered result near the edges since the replicated values do not represent new information. One of the solution is using zero padding, but compare with their result, I think in most cases using edge padding is still better.

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

In median filter, larger kernel size may remove noise effectively but also reduce image's sharpness and cause the edge getting blur.

Maybe can combine filter with some edge-preserving technique.

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

During Laplacian sharpening, the output picture over-sharpening, it cause halos and ringing around the edges. So I tried to applied gaussian filter before sharpening. Although using a Gaussian filter before Laplacian sharpening can effectively reduce the ringing around the edges, it also makes the output image blurry, which conflicts with the original purpose of sharpening..