# Homework 1:
# Image Enhancement Using Spatial Filters

陳宥銓
313554013
Institute of Data Science and Engineering

## Contents

# 1 Implementation

## 1.1 Padding

Fig 1 shows how padding is performed. The following are the detailed steps.

1. Extract Image Dimensions
2. Calculate Padding Size $n$
3. Create an all-zero image of size $n$
4. Fill the center with the original image

```python
def padding(input_img, kernel_size):
    height, width, channels = input_img.shape
    # Calculate the size based on the kernel size
    pad_size = kernel_size // 2
    # Create a Padded image filled with zeros
    padded_image = np.zeros((height + 2 * pad_size, width + 2 * pad_size, 3))
    for c in range(channels):
        # Copy the original image into the center
        padded_image[pad_size:pad_size + height, pad_size:pad_size + width, c] = input_img[:, :, c]
    return padded_image
```

Figure 1: Zero Padding

## 1.2 Convolution

Fig 2 shows how convolution is performed with

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} G(i,j)I(x+i,y+j) \tag{1}$$

,where $k$ is kernel size

```python
def convolution(input_img, kernel):
    kernel_size = kernel.shape[0]
    height, width, channels = input_img.shape
    convolve_img = np.zeros_like(input_img, dtype=np.float32)
    padded_image = padding(input_img, kernel_size)

    # Convolution operation
    for i in range(height):
        for j in range(width):
            for c in range(channels):
                # Extract the region of the padded image corresponding to the kernel
                region = padded_image[i:i+kernel_size, j:j+kernel_size, c]
                # Compute the value for the current pixel
                convolve_img[i, j, c] = np.sum(region * kernel)
    convolve_img = np.clip(convolve_img, 0, 255).astype(np.uint8)
    return convolve_img
```

Figure 2: Convolution

## 1.3 Gaussian Filter

The Gaussian kernel is defined by the Gaussian distribution

$$G(x, y) = \frac{1}{2\pi\sigma^2} exp[-\frac{(x^2 + y^2)}{2\sigma^2}] \tag{2}$$

where x and y are the distance from the center of the kernel, $\sigma$ is the variance of the distribution. Then apply convolution Eq 1 with Gaussian kernel 2 to blur the image. We can found that $kernel_2$ sharpens images more effectively than $kernel_1$ because $kernel_2$ takes more neighboring pixels into account and detect edges in the image more comprehensively.

```python
def gaussian_filter(input_img, kernel_size, sigma):
    kernel = np.zeros((kernel_size, kernel_size))
    center = kernel_size // 2
    # Construct the Gaussian kernel
    for i in range(kernel_size):
        for j in range(kernel_size):
            # Distance from the center
            dx = i - center
            dy = j - center
            # Calculate the Gaussian at the current position
            kernel[i, j] = (1/(2*np.pi*sigma**2)) * np.exp(-(dx**2 + dy**2) / (2*sigma**2))
    # Normalize
    kernel = kernel / np.sum(kernel)
    # Apply convolution with the Gaussian kernel
    return convolution(input_img, kernel)
```

Figure 3: Gaussian Filter

## 1.4 Median Filter

For each position of the filter window, find the median from the pixel values from the neighborhood.

```python
def median_filter(input_img, kernel_size):
    height, width, channels = input_img.shape
    padded_image = padding(input_img, kernel_size)
    output_img = np.zeros_like(input_img)
    for i in range(height):
        for j in range(width):
            for c in range(channels):
                # Compute the median value of the neighborhood
                output_img[i, j, c] = np.median(padded_image[i:i+kernel_size, j:j+kernel_size, c])
    return output_img
```

Figure 4: Median Filter

## 1.5 Laplacian filter

Apply convolution Eq 1 with Laplacian kernel 1 and Laplacian kernel 2 to sharpen the image.

$$kernel_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, kernel_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{3}$$

```python
def laplacian_sharpening(input_img, idx):
    if idx == 1:
        kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    elif idx == 2:
        kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
    laplacian_img = convolution(input_img, kernel)
    # Apply convolution with the Laplacian kernel
    return laplacian_img
```

Figure 5: Laplacian Filter

# 2 Results and Analysis

## 2.1 Gaussian Filter

In this experiment, we used Gaussian filters 2 with

1. $\sigma = 1.0$ with kernel sizes $k = 3,\ 7,\ 11$
2. $\sigma = 1.0,\ 1.5,\ 2.0$ with kernel sizes $k = 7$

, then compare their results.

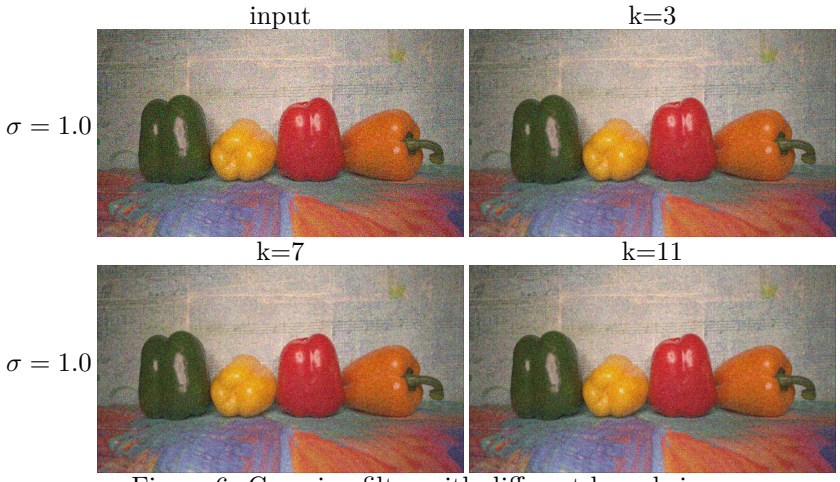



Figure 6: Gaussian filter with different kernel size

According to Fig. 6 and Fig. 7, when the size of the kernel is fixed, a larger $\sigma$ may cause the filtered image to be blurred too much, while a smaller $\sigma$ may cause the image to retain too much detail. Similarly, the larger the kernel size, the more obvious the filtering effect, and the details and noise in the image will be blurred more strongly. This may because larger kernels contain more pixels, taking a wider range of pixel values into account when calculating the average.

## 2.2 Median Filter

In this experiment, we compared three difference kernel size ($k = 3,\ 7,\ 11$) of Median filters. We found that smaller kernels (k=3) can processing small-scale

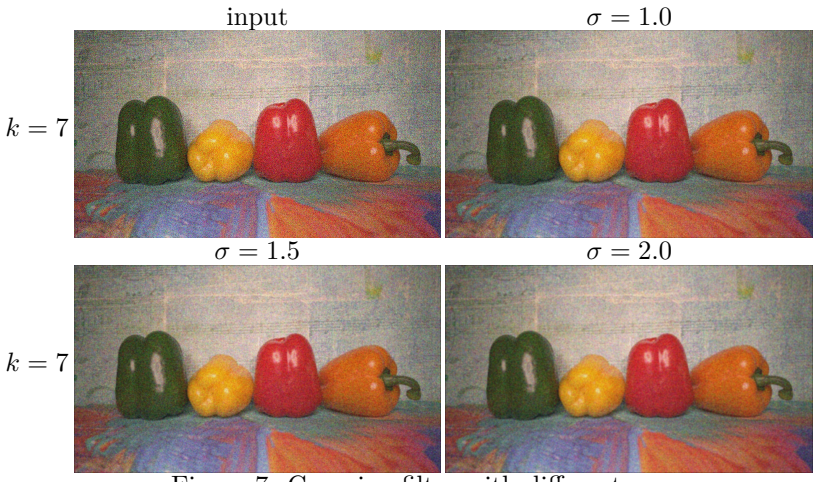


Figure 7: Gaussian filter with different $\sigma$

noise and retain more image details, while larger kernels (k = 5 or 7) can remove more noise, especially noise with a larger range, but may also blur image details.

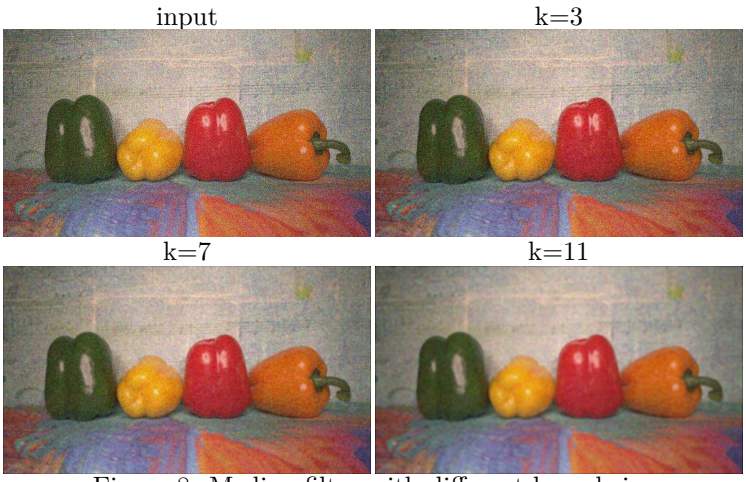


Figure 8: Median filter with different kernel size

## 2.3 Compare the Results of Gaussian Filter and Median Filter

The Gaussian filter may blur the entire image, including edges and details. In contrast, the median filter is effective for sharp bright or dark points without affect the edges or boundaries by averaging and can better preserve the details of the image.

### 2.4 Laplacian Filter

We implement following two types of Laplacian filter for image sharpening.

$$kernel_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, kernel_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{4}$$

Fig. 9 shows the results with different Laplacian kernel. Compared with the $kernel_1$, $kernel_2$ has an enhancement effect on edges in all directions, and the sharpening effect is more uniform, so it has a better effect in enhancing details.

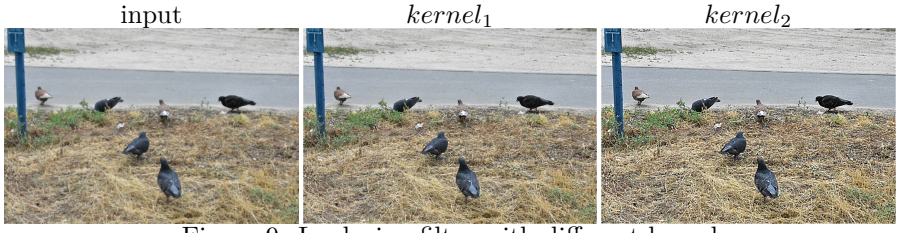



Figure 9: Laplacian filter with different kernel

## 3 Answer the questions

### 3.1 Please describe a problem you encountered and how you solved it.

When performing convolution, the result may exceed the range of pixels (0 to 255), leading to differences in the actual color of the displayed image. Therefore, the final result needs to be normalized to the range of 0 to 255.

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

We use zero padding method. The disadvantage may be that adding zeros may cause unnatural transitions, which may result in loss of information during subsequent processing. The possible solution might be to use other more complex padding such as reflect padding or replicate padding to retain more information

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

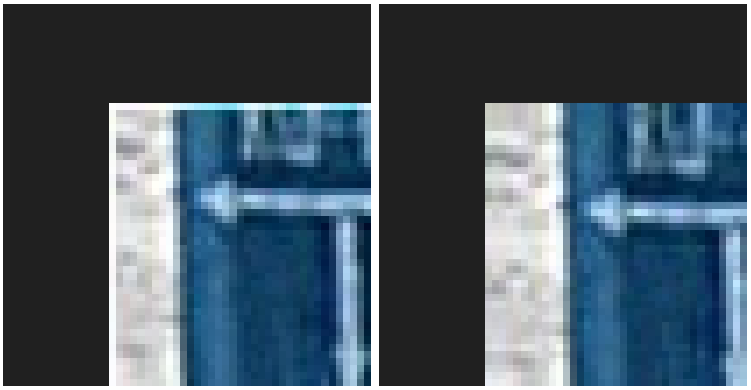
The difficulty in performing Gaussian filtering and median filtering is the setting of parameters. Both filters are highly dependent on parameter settings,

improper selection of parameters may lead to unsatisfactory results, so we need to conduct multiple experiments on the image to determine the best parameters.

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

At the boundary of the image, due to the use of zero padding, the Laplacian filter may cause the loss or distortion of boundary information, which can affect the overall quality of the image Fig. 10a. We can use reflect padding to reduce the impact on the edges. Fig. 10b shows the result of reflect padding performed by the OpenCV function.

Figure 10: Boundary of images with different padding method



(a) Zero padding (b) Reflect padding