# 中山大学计算机学院

# 数字图像处理

# 本科生实验报告

**（2024 学年秋季学期）**

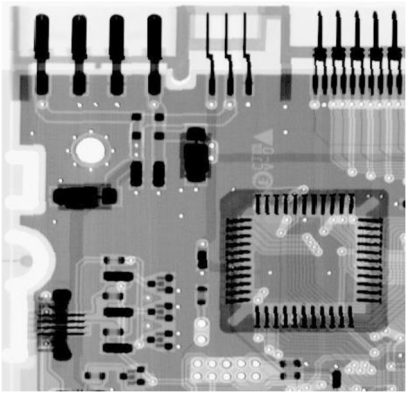
课程名称：Digital Image Process

| 教学班级 | 202410380 | 专业（方向） | 计算机科学与技术 |
|---|---|---|---|
| 学号 | 22320107 | 姓名 | 饶鉴晟 |

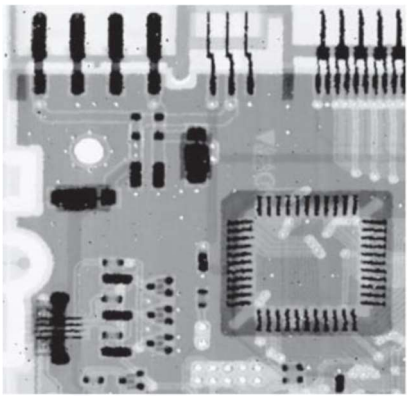
## 一、 实验题目

### 1. 中值滤波

下载图$(a)$，加入椒盐噪声，要求$Pa = Pb = 0.2$，将得到的结果应用中值滤波，然后比较并解释与5.10$(b)$的主要区别；



图$(a)$



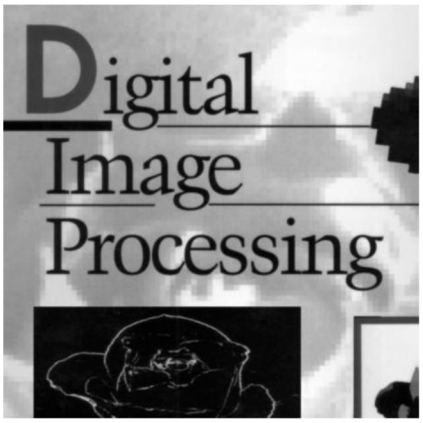
图$(b)$

### 2. 图像滤波与恢复

实现如式5.77所示的滤波器，使用$T = 1$在$+45^\circ$方向上模糊图$(b)$，如5.26$(b)$所示，然后向模糊图像添加均值为0、方差为10像素的高斯噪声，使用式5.85所示的参数的维纳滤波器恢复图像。

$$H(u,v) = \frac{T}{\pi(ua+vb)}\sin[\pi(ua+vb)]\,\mathrm{e}^{-\mathrm{j}\pi(\mathrm{ua}+\mathrm{vb})} \tag{5.77}$$

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)}\cdot\frac{|H(u,v)|^2}{|H(u,v)|^2+K}\right]\cdot G(u,v) \tag{5.85}$$




图$(b)$

$5.26(b)$

## 二、 实验目的

通过实现和应用不同的图像处理技术，深入理解数字图像处理的基本原理。实验（一）模拟了椒盐噪声的产生过程，并利用中值滤波器对椒盐噪声进行过滤。实验（二）通过设计和应用模糊滤波器以及维纳滤波器，学习如何实现滤波操作并分析其数学模型与实际效果。实验还模拟了高斯噪声的添加过程，并通过噪声去除算法（如维纳滤波器）进行图像恢复，评估噪声处理的效果。同时，通过实验分析噪声对图像质量的影响，进一步理解不同滤波器在去噪中的表现与作用。

## 三、 实验环境

- 操作系统：Windows 11 Workstation Pro 24H2
- IDE：PyCharm 2024.3 (Professional Edition) Build #PY-243.21565.199
- Package：Python=3.13，opencv-python=4.10.0.84，numpy=2.2.0

# 四、 实验步骤

## 1. 编程作业#1：中值滤波

### (1) 利用 OpenCV 库导入图片并将图片转换为灰度矩阵。

```
import cv2
import numpy as np
import random

N = 5 # 滤波核的维数
pa = pb = 0.2 # 椒盐噪声出现的概率

# 读取图像并转化为 numpy 数组
image_a = cv2.imread('a.jpeg', cv2.IMREAD_GRAYSCALE)
noised_matrix = image_a.copy()
```

首先导入所需的库。在这个实现当中我设置了三个超参数，分别是滤波核的维数$N$，椒噪声和盐噪声出现的概率$pa$和$pb$。

然后使用 OpenCV 库的 imread()函数将图像转化为 numpy 矩阵，并指定图像读取方式为灰度图像，使得生成的矩阵为灰度矩阵。接着创建一个噪声矩阵，利用原图像进行深拷贝初始化。

### (2) 对灰度矩阵处理实现椒盐噪声的添加

```
# 添加椒盐噪声
for i in range(noised_matrix.shape[0]):
    for j in range(noised_matrix.shape[1]):
        random_value = random.uniform(0, 1)
        if pa + pb > random_value > pa:
            noised_matrix[i][j] = 255
        elif random_value < pa:
            noised_matrix[i][j] = 0
# 填充操作
temp_matrix = np.pad(
      noised_matrix, pad_width=((N // 2, N // 2), (N // 2, N // 2)),
      mode='constant', constant_values=255)
mf_matrix = np.zeros((noised_matrix.shape[0], noised_matrix.shape[1]))
# print(f"{a_matrix.shape, image_mf.shape}")
```

我们利用两层迭代对灰度矩阵的每个值进行处理。

首先我们利用 Python 的 random()函数生成了一个0到1之间的随机值，当该随机值满足 pa < random_value < pa + pb 时设置该次迭代的灰度矩阵像素为盐噪声（灰度值变异为255），满足 random_value < pa 时设置该像素为椒噪声（灰度值变异为0）。

然后我们对加了噪声的图像进行填充操作，从而方便后续的滤波处理。这里我们采用 0 填充法，即填充一圈灰度值为255，宽度为$N//2$的像素。填充后的灰度矩阵存储到临时矩阵当中，方便后续的滤波处理。

**(3) 中值滤波**

```
# 中值滤波操作
for i in range(N // 2, temp_matrix.shape[0] - N // 2):
    for j in range(N // 2, temp_matrix.shape[1] - N // 2):
        filter_kernel = temp_matrix[i-(N//2):i+(N//2)+1, j-(N//2):j+(N//2)+1]
        # print(filter_kernel)
        new_value = np.median(filter_kernel)
        # print(new_value)
        mf_matrix[i - N // 2][j - N // 2] = new_value
mf_matrix = np.uint8(mf_matrix) # 将中值滤波后的图像转换为 uint8 类型
```

我们同样利用两层迭代遍历填充后的噪声矩阵，对其进行滤波操作。首先我们利用 Python 的切片功能把滤波核取出，并利用 np.median()函数计算滤波核内灰度值的中位数。需要注意的是，np.median()函数会返回一个 float64 类型的数值，而灰度矩阵要求值为 uint8 类型的数值，因此稍后要进行类型转换。

然后，我们将中位数灰度值存储到新的矩阵当中，命名为 mf_matrix[]。

最后，我们利用 np.uint8()函数将所有灰度值转化为 uint8 类型，从而 OpenCV 可以正确将其输出为灰度图像。

**(4) 计算相关参数，衡量图像处理的质量。**

```
# 计算处理后图像和原图像的均方差和 psnr 峰值信噪比，用于评估处理效果
mse = np.mean((mf_matrix - image_a) ** 2)
psnr = cv2.PSNR(image_a, mf_matrix)
```

对于滤波的量化结果而言，均方误差越小越好（说明与原图相差越少），$PSNR$峰值信噪比越大越好（说明噪声占图像比重小）

$N = 3$:

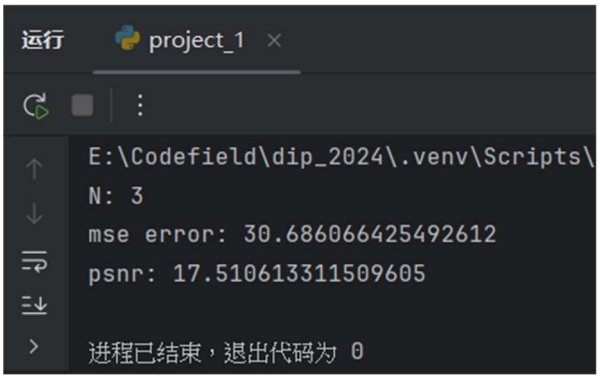



图1 – 1

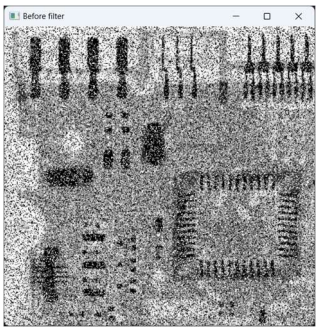



图1 – 2

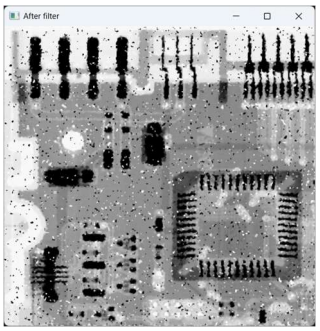



图1 – 3

$N = 5$:

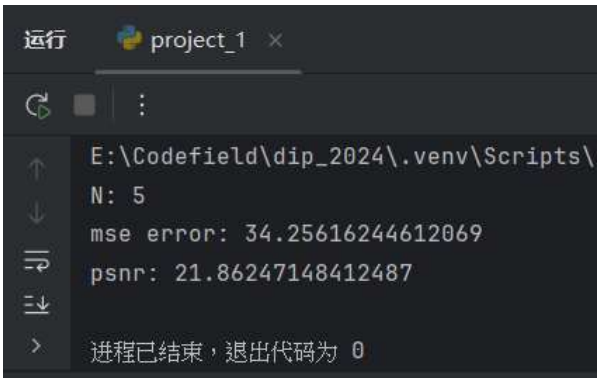



图 2 – 1

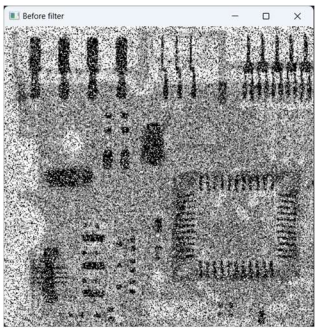



图2 – 2

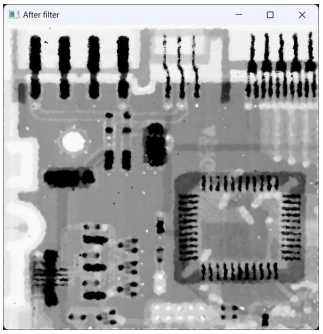



图2 – 3

$N = 7$:

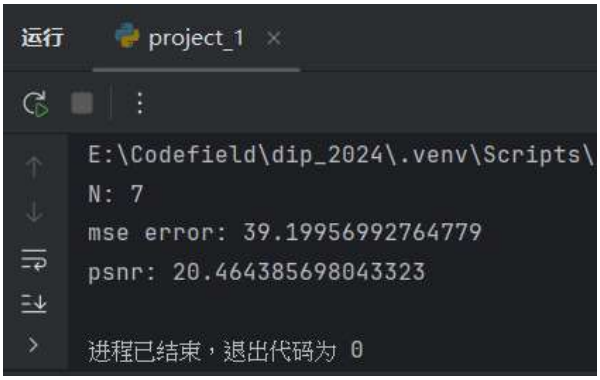



图3 – 1

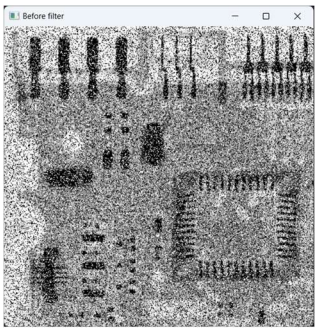



图3 – 2

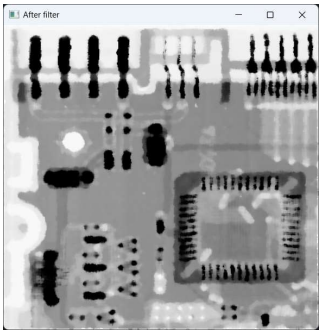
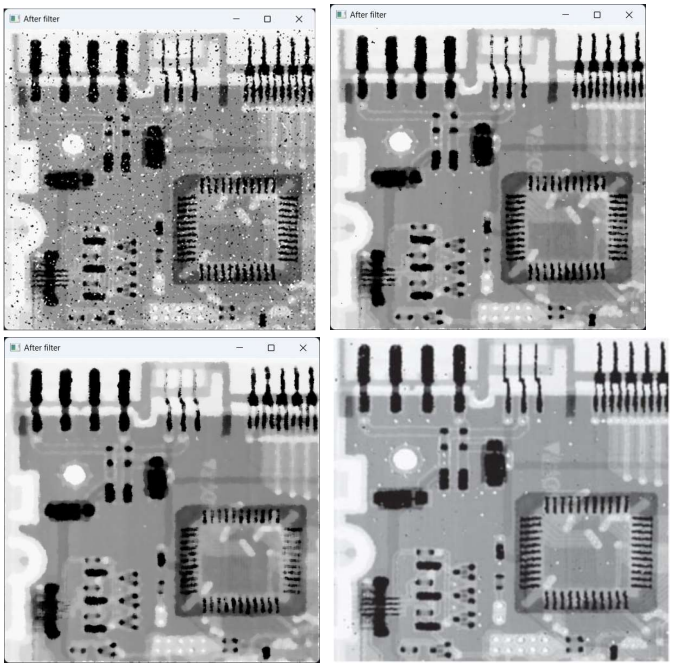



图3 – 3

$N = 3, 5, 7$时以及题目给出的图$(b)$的处理效果对比（从左到右，从上到下）：



如图所示，在$N=3$的时候（图$1-3$），由于滤波核太小，因此不能很好地过滤掉椒盐噪声，滤波后的图像仍然含有大量的椒盐噪声。从量化结果来看，虽然与原图像均方误差不大，但是$PSNR$值较大，说明图像噪声较多，滤波效果不好。

在$N=5$的时候（图$2-3$），滤波效果比较理想，和目标效果图(b)视觉效果上相差无几，而从量化结果上看，$N=7$时均方误差和$PSNR$值均比较适中，说明滤波效果好。

在$N=7$的时候（图$3-3$），虽然噪声含量最低，但是由于滤波核过大，可以看到图像出现了比较明显的泛化，部分图像的细节丢失，滤波效果不如$N=5$的时候。

综上所述，在设置滤波核大小为$5\times5$的时候，滤波效果最好。

## 2. 编程作业#2：图像滤波与恢复

### (1) 构建运动模糊频域滤波器

```
def motion_blur_filter(shape, T=1, a=0.1, b=0.1):
    M, N = shape
    # 频率坐标 u,v（使用 numpy 的 fftshift 风格从 -N/2~N/2 来定义）
    u = np.arange(M)
    v = np.arange(N)
    u = u - M//2
    v = v - N//2
    U, V = np.meshgrid(u, v, indexing='ij')

    # 避免分母为 0 的情况
    denom = (U*a + V*b)
    H = np.zeros((M, N), dtype=complex)

    # 对于非零点
    nonzero_mask = denom != 0
    H[nonzero_mask] = (T / (np.pi * denom[nonzero_mask])) * \
        np.sin(np.pi * denom[nonzero_mask]) * np.exp(-1j * np.pi *
denom[nonzero_mask])

    # 求极限
    H[~nonzero_mask] = T

    return H
```

我们根据式 5.77 所给的运动模糊滤波器的表达式构建滤波器。其中滤波器的表达式为：

$$H(u,v)=\frac{T}{\pi(ua+vb)}\sin[\pi(ua+vb)]\,\mathrm{e}^{-\mathrm{j}\pi(\mathrm{ua}+\mathrm{vb})} \tag{5.77}$$

需要注意的是，在利用 OpenCV 构建滤波器的时候，要考虑分母为0的情况，需要进行极限近似。考虑到有极限（根据洛必达法则）：

$$\lim_{x\to 0}\frac{\sin x}{x}=1$$

所以：

$$\lim_{u,v\to 0}H(u,v)=T$$

### (2) 构建维纳滤波器

```
def wiener_filter(G, H, K=0.01):
    H_conj = np.conjugate(H)
    H_abs2 = np.abs(H)**2
    F_hat = (H_conj / (H_abs2 + K)) * G
    return F_hat
```

我们根据式 5.85 所给的维纳滤波器的表达式构建滤波器。其中维纳滤波器的表达式为：

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)} \cdot \frac{|H(u,v)|^2}{|H(u,v)|^2 + K}\right] \cdot G(u,v) \tag{5.85}$$

所以我们可以根据上面所示的代码构建维纳滤波器。

### (3) 构建高斯噪声器

```
def add_gaussian_noise(image, mean=0, sigma=10):
    gauss = np.random.normal(mean, sigma, image.shape)
    noisy = image + gauss
    noisy = np.clip(noisy, 0, 255).astype(np.uint8)
    return noisy
```

根据题目的要求，我们需要构建一个均值为0，标准差为10的高斯噪声。上面的函数接收一个灰度矩阵作为参数，在该灰度矩阵上叠加均值为0，标准差为10的高斯噪声。

### (4) 定义傅里叶变换和反变换

```
def dft2(image):
    return np.fft.fftshift(np.fft.fft2(image))

def idft2(freq_image):
    return np.real(np.fft.ifft2(np.fft.ifftshift(freq_image)))
```

我们使用 Numpy 自带的 np.fft.fft2()函数和 np.fft.ifft2()函数进行傅里叶变换和反变换。两个函数都接收一个灰度矩阵作为参数，分别输出频域傅里叶变换结果矩阵和空域傅里叶反变换结果矩阵。

(5) 定义主函数

```
if __name__ == "__main__":
    # 读取灰度图像
    img = cv2.imread('b.jpeg', cv2.IMREAD_GRAYSCALE)
    img_float = img.astype(np.float64)

    # 构建运动模糊滤波器 H(u,v)
    H = motion_blur_filter(img_float.shape, T=1, a=0.1, b=0.1)

    # 对图像进行模糊处理（频域操作）
    F = dft2(img_float)
    G = F * H  # 模糊后的频域表示
    blurred = idft2(G)
    blurred = np.clip(blurred, 0, 255).astype(np.uint8)

    # 添加高斯噪声
    noisy_blurred = add_gaussian_noise(blurred, mean=0, sigma=10)

    # 转换回频域
    G_noisy = dft2(noisy_blurred.astype(np.float64))

    # 维纳滤波恢复图像
    K = 0.01
    F_hat = wiener_filter(G_noisy, H, K=K)
    restored = idft2(F_hat)
    restored = np.clip(restored, 0, 255).astype(np.uint8)

    # 保存结果
    cv2.imwrite('./img/b/blurred_image.jpeg', blurred)
    cv2.imwrite('./img/b/noisy_blurred_image.jpeg', noisy_blurred)
    cv2.imwrite('./img/b/restored_image.jpeg', restored)
```

首先读取需要处理的灰度图像，并转换为浮点类型方便处理，然后根据给定参数构建运动模糊滤波器 $H(u,v)$。

接下来，通过计算图像的频域表示 $F(u,v)$，将其与 $H(u,v)$ 相乘从而实现运动模糊，之后对模糊后的结果进行反傅里叶变换得到时域的模糊图像。之后在该模糊图像上叠加高斯噪声，并再次将有噪声的图像转换回频域以便应用维纳滤波。

通过维纳滤波器根据参数 $K$ 对加噪模糊的频域数据进行处理，以恢复接近原始清晰度的图像。最终程序将模糊图像、有噪声的模糊图像，以及恢复后的图像分别保存到指定路径下，从而完成一次完整的图像退化与恢复过程。

实验结果如下图所示。从上到下，从左到右依次为原图像，运动模糊处理后的图像，添加高斯噪声后的图像，以及应用维纳滤波器还原的图像。


需要注意的是，实验当中有一些参数的设置需要根据情况调整，比如添加运动模糊的时候控制模糊程度的 $a$ 和 $b$ 参数，以及维纳滤波器中的 $K$ 参数，用于评估图像的信噪比并进行处理。经过我的多次手动调整后，$a = b = 0.1$ 时的运动模糊与题目要求最相似，$K = 0.01$ 时的滤波效果最好。

# 五、 参考资料

[1] OpenCV. (2023). OpenCV Tutorials. Available at: https://docs.opencv.org/4.x/d9/df8/tutorial_root.html