2023 年秋季学期 图像处理 编程作业 03

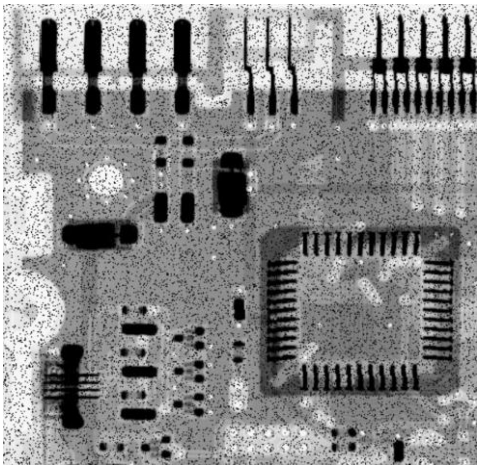
姓名：梁乐彬 学号：[redacted]

采用图像复原技术复原受噪声污染的图像

本次作业 100 分

1. Fig01.tif 表示原始图像，Fig02.tif，Fig03.tif, Fig04.tif 是受到不同类型噪声污染后的图像。请利用图像复原一章所学的相关内容，对受到噪声污染的图像进行复原，并分别计算原图像与复原后图像的均方误差 MSE 和信噪比 SNR。

【一】分析 Fig02.tif 的噪声类型为胡椒噪声，黑噪声。所以采用反谐波均值滤波处理。



构建反谐波均值滤波函数：

```python
def inverse_harmonic_mean_filter(image, mask_size, Q):
    # 创建结果图像的副本，以避免修改原始图像
    result_image = np.copy(image)

    # 计算填充尺寸
    pad_size = mask_size // 2

    # 遍历图像中的每个像素
    for i in range(pad_size, image.shape[0] - pad_size):
        for j in range(pad_size, image.shape[1] - pad_size):
            # 获取掩模窗口内的像素值
            values = []
            for m in range(-pad_size, pad_size + 1):
                for n in range(-pad_size, pad_size + 1):
                    values.append(image[i + m, j + n])

            # 仅考虑非零像素值
            non_zero_values = [val for val in values if val != 0]

            # 如果存在非零像素值，则进行反谐波均值滤波计算
            if len(non_zero_values) > 0:
                num = np.sum(np.power(non_zero_values, Q + 1))
                den = np.sum(np.power(non_zero_values, Q))

                # 避免除以零，如果 den 为零，则将结果设定为 0
                result_image[i, j] = num / (den + 1e-18) if den != 0 else 0

    return result_image
```

反谐波均值滤波器函数逻辑如下：

1.参数说明：image：输入的图像矩阵。

mask_size：滤波器的窗口大小，通常是奇数。

Q：滤波器的阶数，用于计算滤波器的参数。

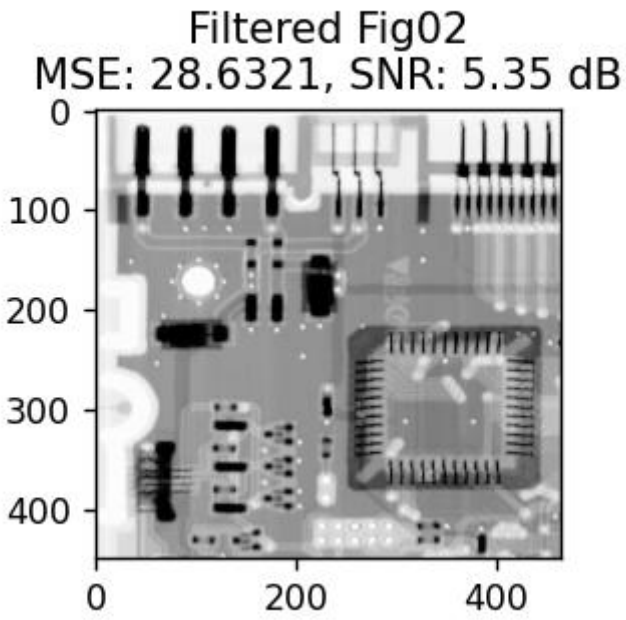
2.遍历图像像素：通过两个嵌套的循环遍历图像的每个像素，避免处理图像边缘，确保滤波器窗口始终在图像内。

3.构建滤波器窗口：在每个像素位置，构建一个大小为 mask_size x mask_size 的窗口，获取窗口内的像素值。

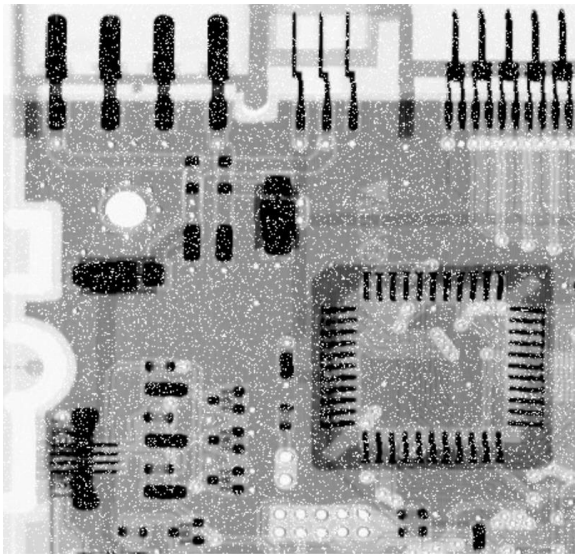
4.计算非零像素的幂和：将窗口内非零像素的值分别提升到 Q+1 次方，计算它们的和（num）和 Q 次方的和（den）。

5.更新像素值：如果窗口内有非零像素，根据反谐波均值滤波器的公式更新当前像素的值为零，则将结果设定为 0，以避免除以零的错误。

滤波效果：



【二】分析 Fig03.tif 的噪声类型为盐粒噪声，白噪声。所以采用谐波均值滤波处理。



构建谐波均值滤波函数：

```python
# 谐波均值滤波器函数
def harmonic_mean_filter(image, mask_size):
    result_image = np.copy(image)
    pad_size = mask_size // 2

    # 循环遍历图像像素
    for i in range(pad_size, image.shape[0] - pad_size):
        for j in range(pad_size, image.shape[1] - pad_size):
            values = []

            # 创建基于掩模大小的像素窗口
            for m in range(-pad_size, pad_size + 1):
                for n in range(-pad_size, pad_size + 1):
                    values.append(1 / (image[i + m, j + n] + 1e-18))

            # 使用谐波均值滤波器公式计算结果
            result_image[i, j] = (mask_size**2) / np.sum(values)

    return result_image
```

谐波均值滤波器函数逻辑如下：

1.参数说明：image：输入的图像矩阵。

mask_size：滤波器的窗口大小，通常是奇数。

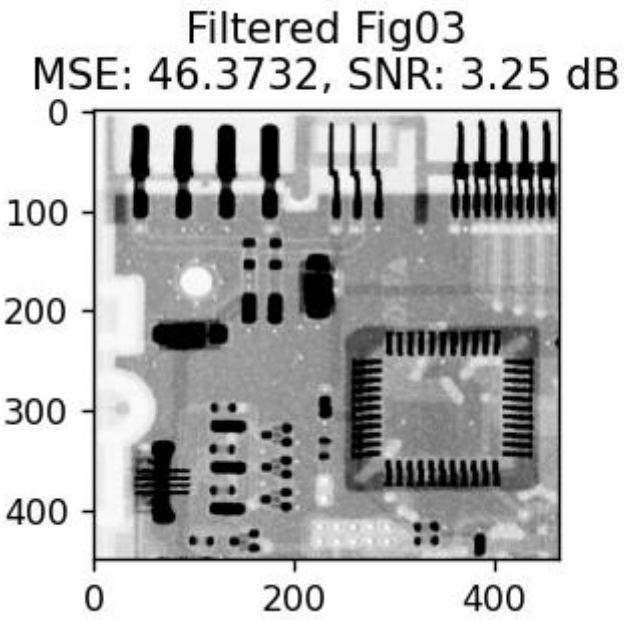
2.遍历图像像素：通过两个嵌套的循环遍历图像的每个像素，避免处理图像边缘，确保滤波器窗口始终在图像内。

3.构建滤波器窗口：在每个像素位置，构建一个大小为 mask_size x mask_size 的窗口，获取窗口内的像素值。

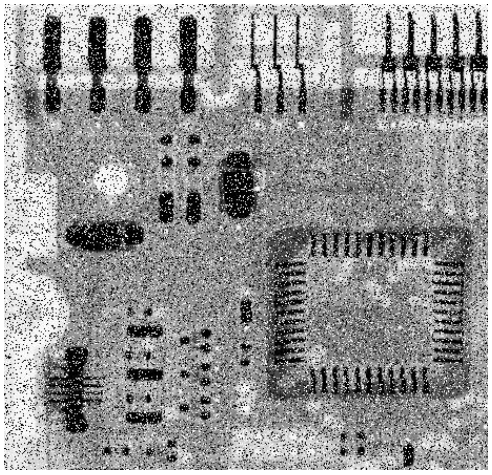
4.计算倒数并求和：对窗口内每个像素值取倒数，添加一个小的常数（防止除以零），然后计算它们的和。

5.更新像素值：使用谐波均值滤波器的公式更新当前像素的值：

滤波效果：



【二】分析 Fig04.tif 的噪声类型为椒盐噪声，黑白噪声均有。所以采用自适应中值滤波。



构建自适应中值滤波函数：

```python
# 自适应中值滤波器函数
def adaptive_median_filter(img, max_size=7):
    # 图像类型检查和转换为灰度图像
    if len(img.shape) == 3:
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    else:
        gray_img = img.copy()

    m, n = gray_img.shape
    Nmax = max_size // 2
    result_image = np.zeros_like(gray_img)

    # 循环遍历图像像素
    for i in range(Nmax, m - Nmax):
        for j in range(Nmax, n - Nmax):
            window_size = 3
            while window_size <= max_size:
                window = gray_img[i - window_size // 2:i + window_size // 2 + 1,
                                  j - window_size // 2:j + window_size // 2 + 1]

                # 使用排序数组来计算中值
                sorted_window = np.sort(window.flatten())
                median_index = len(sorted_window) // 2
                median = sorted_window[median_index]

                # 检查中值是否在窗口范围内
                if sorted_window[0] < median < sorted_window[-1]:
                    # 检查当前像素值是否为脉冲噪声
                    if sorted_window[0] < gray_img[i, j] < sorted_window[-1]:
                        result_image[i, j] = gray_img[i, j]
                    else:
                        result_image[i, j] = median
                    break
```

自适应中值滤波器逻辑如下:

1.参数说明：img: 输入的图像矩阵。

max_size: 滤波器的最大窗口大小，通常是奇数。

2.灰度处理：如果输入图像是彩色的（三通道），则将其转换为灰度图像。

3.遍历图像像素：通过两个嵌套的循环遍历图像的每个像素，避免处理图像边缘，确保滤波器窗口始终在图像内。

4.动态调整窗口大小：从初始窗口大小为 3 开始，逐步增加窗口大小，直到达到 max_size。对于每个窗口，提取窗口内的像素值。

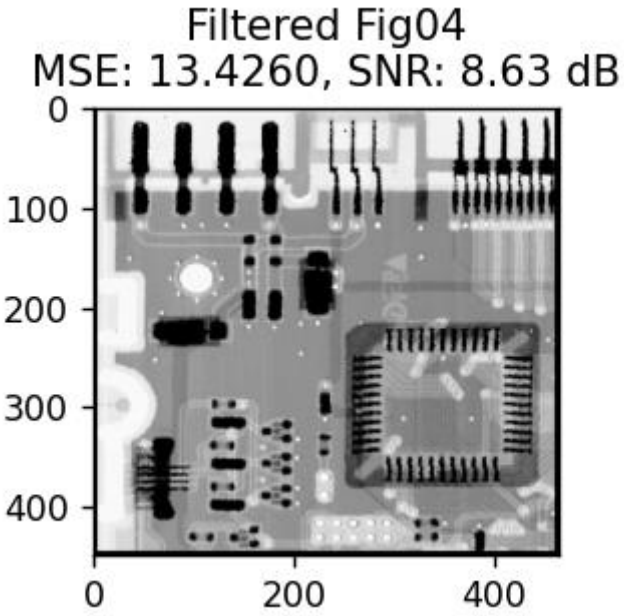
5.计算中值：对窗口内的像素值进行排序，计算中值。

6.检查中值范围：检查计算得到的中值是否在窗口内像素值的范围内。如果是，则继续；否则，增加窗口大小。

7.检查脉冲噪声：检查当前像素值是否被认为是脉冲噪声。如果是，将原始像素值保留；否则，将中值作为新的像素值。

8.循环结束：当找到适当的中值并更新像素值后，结束当前像素的处理

滤波效果：



MSE 和 SNR 的计算实现：

```
# 计算 MSE 和 SNR 的函数
def calculate_metrics(original_image, processed_image):
    mse = np.mean((original_image - processed_image) ** 2)
    original_energy = np.sum(original_image ** 2)
    noise_energy = np.sum((original_image - processed_image) ** 2)
    snr = 10 * np.log10(original_energy / noise_energy)

    return mse, snr
```

程序主逻辑：

```python
# 读取图像
fig01 = cv2.imread('Fig01.tif', cv2.IMREAD_GRAYSCALE)
fig02 = cv2.imread('Fig02.tif', cv2.IMREAD_GRAYSCALE)
fig03 = cv2.imread('Fig03.tif', cv2.IMREAD_GRAYSCALE)
fig04 = cv2.imread('Fig04.tif', cv2.IMREAD_GRAYSCALE)

# 对图像应用滤波器
# 对第一个图像使用反谐波均值滤波
filtered_fig02_pepper = contra_harmonic_mean_filter(fig02, mask_size=3, Q=1.5)
# 对第二个图像使用谐波均值滤波
filtered_fig03_salt = harmonic_mean_filter(fig03, mask_size=3)
# 对第三个图像使用自适应中值滤波
filtered_fig04 = adaptive_median_filter(fig04, 8)

# 计算评估指标
mse_fig02_pepper, snr_fig02_pepper = calculate_metrics(fig01, filtered_fig02_pepper)
mse_fig03_salt, snr_fig03_salt = calculate_metrics(fig01, filtered_fig03_salt)
mse_fig04, snr_fig04 = calculate_metrics(fig01, filtered_fig04)
```

最终完整结果：