96

整体分析对比到位

作业 3:采用图像复原技术复原受噪声污染的图像(本次作业 100 分)

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。

运行指引：

请运行 ipynb 文件，确保 ipynb 文件与 Fig01.tif，Fig02.tif，Fig03.tif, Fig04.tif 在同一目录下，依次运行各个代码块即可。

1.作业思路：

目视判断 Fig02.tif 受胡椒噪声影响、Fig03.tif 受盐粒噪声影响，Fig04.tif 受椒盐噪声影响，结合图像复原一章内容，适合于以上噪声的方法和其特点分别为：

1.1 谐波平均滤波器（**Harmonic Mean Filter**）:

- 适用于处理盐粒噪声和类高斯噪声。
- 不能处理胡椒噪声。

1.2 反谐波平均滤波器（**Contraharmonic Mean Filter**）:

- 通过调整滤波器的阶数（Q 值）来处理不同类型的噪声。
- Q 值为正时用于消除胡椒噪声。
- Q 值为负时用于消除盐粒噪声。
- 不能同时消除两种噪声。

1.3 中值滤波器（**Median Filter**）:

- 降低图像噪声，同时保持较小的模糊度。
- 适合处理单极或双极冲激噪声（如椒盐噪声）。

1.4 最大值/最小值滤波器（**Max and Min Filters**）:

- 最大值滤波器用于降低胡椒噪声。
- 最小值滤波器用于降低盐粒噪声。

1.5 修正阿尔法均值滤波器（**Alpha-Trimmed Mean Filter**）:

- 通过删除邻域内一定比例的最高和最低灰度值来处理噪声。
- 适合处理多种噪声，包括高斯噪声和椒盐噪声。

1.6 自适应中值滤波器（**Adaptive Median Filter**）:

- 能够处理更大概率的椒盐噪声。
- 在保留图像细节的同时平滑非冲激噪声。

2.图像复原方法实现及结果：

因需要计算原图像和复原图像的均方误差 MSE 和信噪比 SNR，在各种方法定义之前首先定义计算 MSE 和 SNR 的函数：

```
# 导入必要的库
import numpy as np
import cv2
import matplotlib.pyplot as plt

# 定义计算 MSE 和 SNR 的函数
def calculate_mse(original, restored):
    return np.mean((original - restored) ** 2)

def calculate_snr(original, restored):
    noise = original - restored
    signal_power = np.mean(original.astype(np.float64) ** 2)
    noise_power = np.mean(noise.astype(np.float64) ** 2)
    snr = 10 * np.log10(signal_power / noise_power) if noise_power !=
0 else float('inf')
    return snr
```

MSE计算会出现uint8计算溢出，SNR计算对

2.1 谐波平均滤波器实现和结果：

谐波平均滤波器的基本原理是取邻域像素值的倒数的平均值的倒数作为中心像素的值。这个方法对于盐粒噪声特别有效。

以下是谐波平均滤波器的 Python 实现：

```
# 定义函数
def harmonic_mean_filter(image, kernel_size):
    """
    实现谐波平均滤波器。
    :param image: 输入图像
    :param kernel_size: 滤波器的大小（必须是奇数）
    :return: 滤波后的图像
    """
    # 边界扩展
    pad_size = kernel_size // 2
    padded_image = cv2.copyMakeBorder(image, pad_size, pad_size,
pad_size, pad_size, cv2.BORDER_REFLECT)

    # 初始化输出图像
    filtered_image = np.zeros_like(image, dtype=np.float32)

    # 遍历每个像素
    for i in range(pad_size, padded_image.shape[0] - pad_size):
        for j in range(pad_size, padded_image.shape[1] - pad_size):
            # 提取邻域
```

```
            neighborhood = padded_image[i-pad_size:i+pad_size+1, j-pad_size:j+pad_size+1]

            # 计算谐波平均
            with np.errstate(divide='ignore', invalid='ignore'):
                harmonic_mean = np.mean(1 / neighborhood[neighborhood > 0])
                filtered_image[i - pad_size, j - pad_size] = 1 / harmonic_mean if harmonic_mean > 0 else 0

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

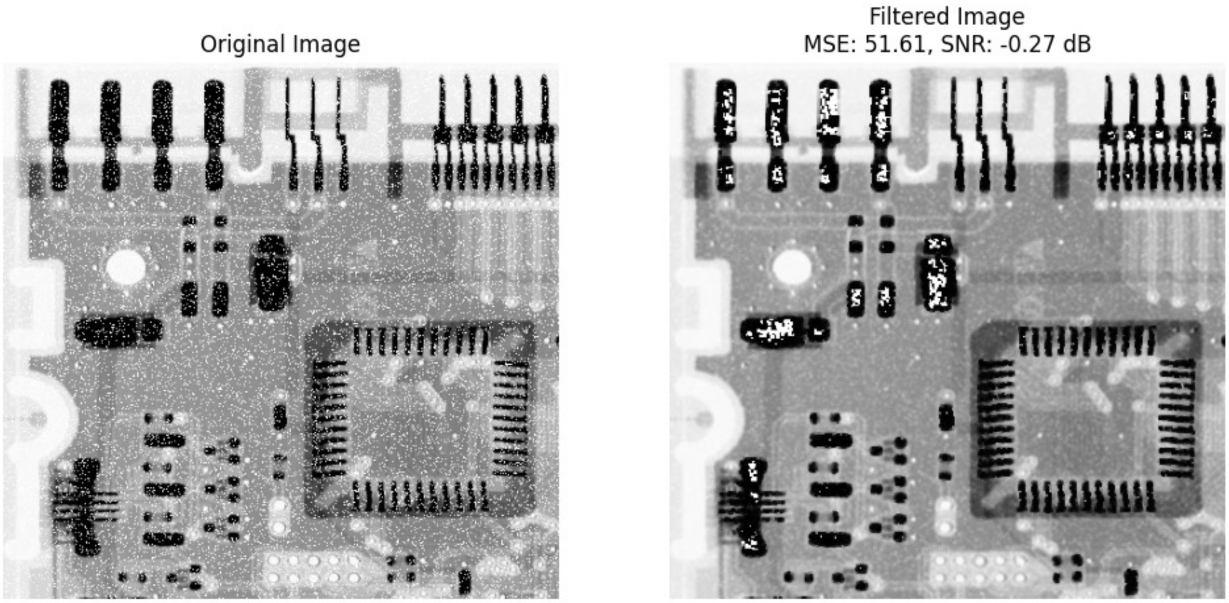
加载受噪声影响的图像并计算 MSE 和 SNR

```
# 加载图像
image_path  = "./Fig03.tif"
image = cv2.imread(image_path,0)

# 应用谐波平均滤波器
filtered_image = harmonic_mean_filter(image, kernel_size=3)

# 计算原图像与复原后图像的 MSE 和 SNR
mse = calculate_mse(image, filtered_image)
snr = calculate_snr(image, filtered_image)

filtered_image, mse, snr
```

```
(array([[248, 248, 248, ..., 247, 247, 247],
        [248, 248, 248, ..., 247, 247, 247],
        [248, 248, 248, ..., 247, 247, 247],
        ...,
        [232, 234, 234, ..., 153, 159, 168],
        [232, 234, 234, ..., 152, 158, 169],
        [232, 234, 234, ..., 151, 158, 169]], dtype=uint8),
 51.611073160406406,
 -0.27248029583606426)
```

```
# 创建图像展示
plt.figure(figsize=(12, 6))

# 原始图像
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

# 滤波后的图像
plt.subplot(1, 2, 2)
plt.imshow(filtered_image, cmap='gray')
plt.title(f'Filtered Image\nMSE: {mse:.2f}, SNR: {snr:.2f} dB')
plt.axis('off')
```

```python
# 展示图像
plt.savefig("谐波平均滤波器.png")
plt.show()
```



结果分析：

可以看到在滤波后的图像中出现了很多白色噪声，这可能是由于谐波平均滤波器在处理图像时对暗色（低灰度值）的噪声特别敏感，因为谐波平均滤波器通过取邻域内像素的倒数的平均值来计算新的像素值，如果邻域内有像素非常低（接近于零的），那么它们的倒数会非常高，这导致在暗区域计算的平均值会非常高，从而产生白色噪声。

尝试对谐波平均滤波器的实现进行修改，以减少暗区域的白色噪声，思路是，在计算谐波平均时对极低的像素值进行限制，通过设置一个阈值，在计算谐波平均时，忽略该阈值的像素值。但经过尝试，效果没有改善，甚至有所恶化，因此放弃了这个思路。

2.2 反谐波平均滤波器实现和结果

```python
def contraharmonic_mean_filter(image, kernel_size, Q):
    """
    实现反谐波平均滤波器。
    :param image: 输入图像
    :param kernel_size: 滤波器的大小(必须是奇数)
    :param Q: 滤波器的阶数
    :return: 滤波后的图像
    """
    pad_size = kernel_size // 2
    padded_image = cv2.copyMakeBorder(image, pad_size, pad_size, 
pad_size, pad_size, cv2.BORDER_REFLECT)
    filtered_image = np.zeros_like(image, dtype=np.float32)
```

```python
    for i in range(pad_size, padded_image.shape[0] - pad_size):
        for j in range(pad_size, padded_image.shape[1] - pad_size):
            neighborhood = padded_image[i-pad_size:i+pad_size+1, j-pad_size:j+pad_size+1]

            # 计算反谐波平均
            numerator = np.sum(neighborhood ** (Q + 1))
            denominator = np.sum(neighborhood ** Q)

            # 防止除以零
            filtered_image[i - pad_size, j - pad_size] = numerator / denominator if denominator != 0 else 0

    return filtered_image.astype(np.uint8)

# 加载胡椒噪声和盐粒噪声影响的图像
image_pepper_path = 'Fig02.tif'
image_salt_path = 'Fig03.tif'
image_pepper = cv2.imread(image_pepper_path, 0)  # 加载为灰度图像
image_salt = cv2.imread(image_salt_path, 0)

# 应用反谐波平均滤波器
# Q 值设为正数以消除胡椒噪声，设为负数以消除盐粒噪声
filtered_image_pepper = contraharmonic_mean_filter(image_pepper, kernel_size=3, Q=1.5)
filtered_image_salt = contraharmonic_mean_filter(image_salt, kernel_size=3, Q=-1.5)
```

```
/var/folders/38/y1t93myj3ydb0tzvq_fbzhzm0000gn/T/ipykernel_53669/3143525346.py:18: RuntimeWarning: divide by zero encountered in power
  numerator = np.sum(neighborhood ** (Q + 1))
/var/folders/38/y1t93myj3ydb0tzvq_fbzhzm0000gn/T/ipykernel_53669/3143525346.py:19: RuntimeWarning: divide by zero encountered in power
  denominator = np.sum(neighborhood ** Q)
/var/folders/38/y1t93myj3ydb0tzvq_fbzhzm0000gn/T/ipykernel_53669/3143525346.py:22: RuntimeWarning: invalid value encountered in scalar divide
  filtered_image[i - pad_size, j - pad_size] = numerator / denominator if denominator != 0 else 0
/var/folders/38/y1t93myj3ydb0tzvq_fbzhzm0000gn/T/ipykernel_53669/3143525346.py:24: RuntimeWarning: invalid value encountered in cast
  return filtered_image.astype(np.uint8)
```

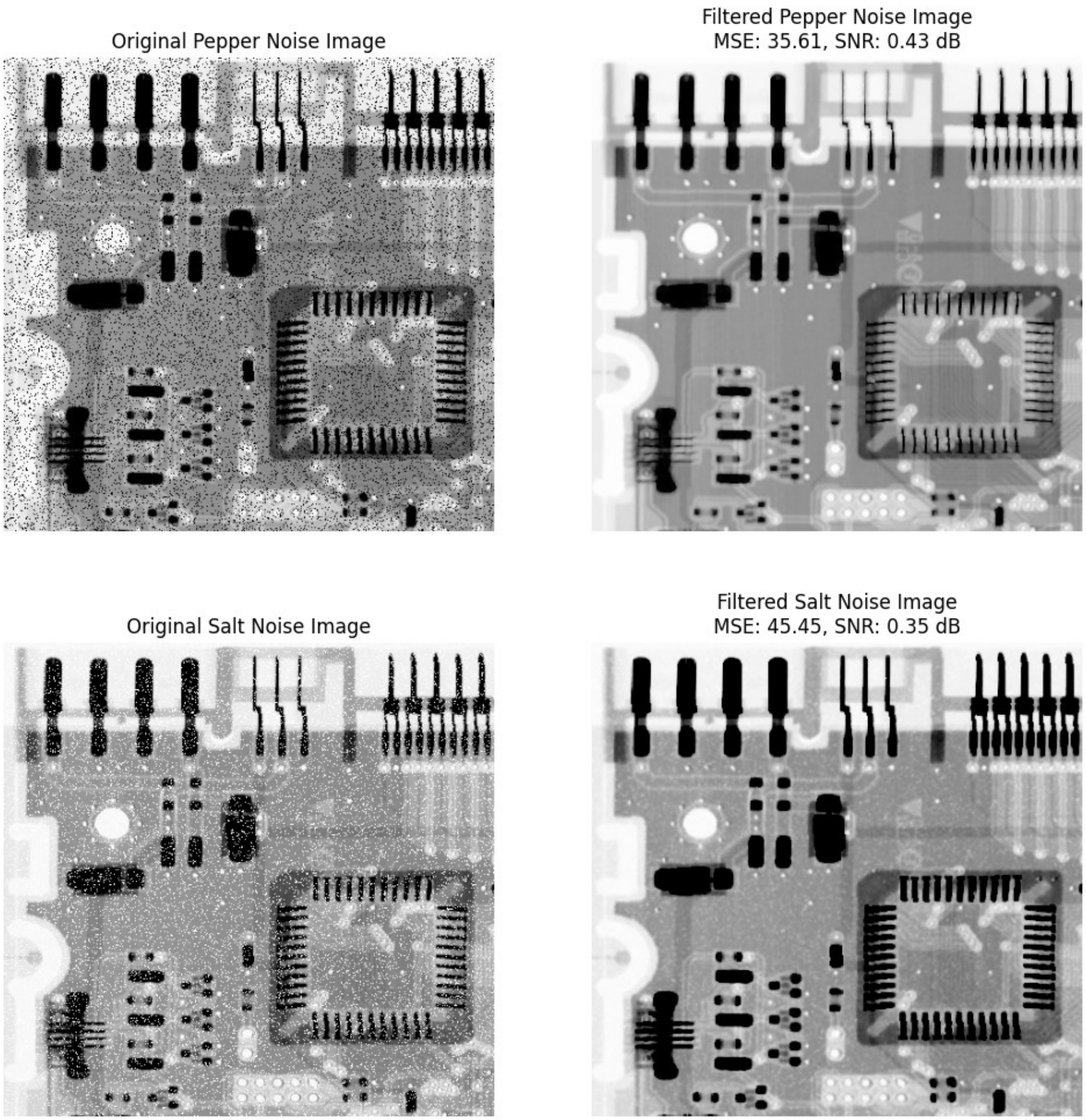
```python
# 计算 MSE 和 SNR
mse_pepper = calculate_mse(image_pepper, filtered_image_pepper)
snr_pepper = calculate_snr(image_pepper, filtered_image_pepper)
mse_salt = calculate_mse(image_salt, filtered_image_salt)
snr_salt = calculate_snr(image_salt, filtered_image_salt)
```

```
(filtered_image_pepper, mse_pepper, snr_pepper), (filtered_image_salt,
mse_salt, snr_salt)
```

```
((array([[248, 248, 248, ..., 247, 247, 247],
         [248, 248, 248, ..., 247, 247, 247],
         [248, 248, 248, ..., 247, 247, 247],
         ...,
         [232, 232, 232, ..., 155, 161, 168],
         [232, 232, 232, ..., 154, 161, 169],
         [232, 232, 232, ..., 153, 161, 169]], dtype=uint8),
  35.60838400554187,
  0.42520011955972814),
 (array([[248, 248, 248, ..., 247, 247, 247],
         [248, 248, 248, ..., 247, 247, 247],
         [248, 248, 248, ..., 247, 247, 247],
         ...,
         [232, 234, 234, ..., 152, 158, 168],
         [232, 234, 234, ..., 151, 158, 169],
         [232, 234, 234, ..., 151, 158, 169]], dtype=uint8),
  45.45493861607143,
  0.35016299756776526))
```

```
# 创建图像展示
plt.figure(figsize=(12, 12))

# 原始胡椒噪声图像
plt.subplot(2, 2, 1)
plt.imshow(image_pepper, cmap='gray')
plt.title('Original Pepper Noise Image')
plt.axis('off')

# 滤波后的胡椒噪声图像
plt.subplot(2, 2, 2)
plt.imshow(filtered_image_pepper, cmap='gray')
plt.title(f'Filtered Pepper Noise Image\nMSE: {mse_pepper:.2f}, SNR:
{snr_pepper:.2f} dB')
plt.axis('off')

# 原始盐粒噪声图像
plt.subplot(2, 2, 3)
plt.imshow(image_salt, cmap='gray')
plt.title('Original Salt Noise Image')
plt.axis('off')

# 滤波后的盐粒噪声图像
plt.subplot(2, 2, 4)
plt.imshow(filtered_image_salt, cmap='gray')
plt.title(f'Filtered Salt Noise Image\nMSE: {mse_salt:.2f}, SNR:
{snr_salt:.2f} dB')
```

```
plt.axis('off')

plt.savefig("反谐波平均滤波器.png")
# 展示图像
plt.show()
```

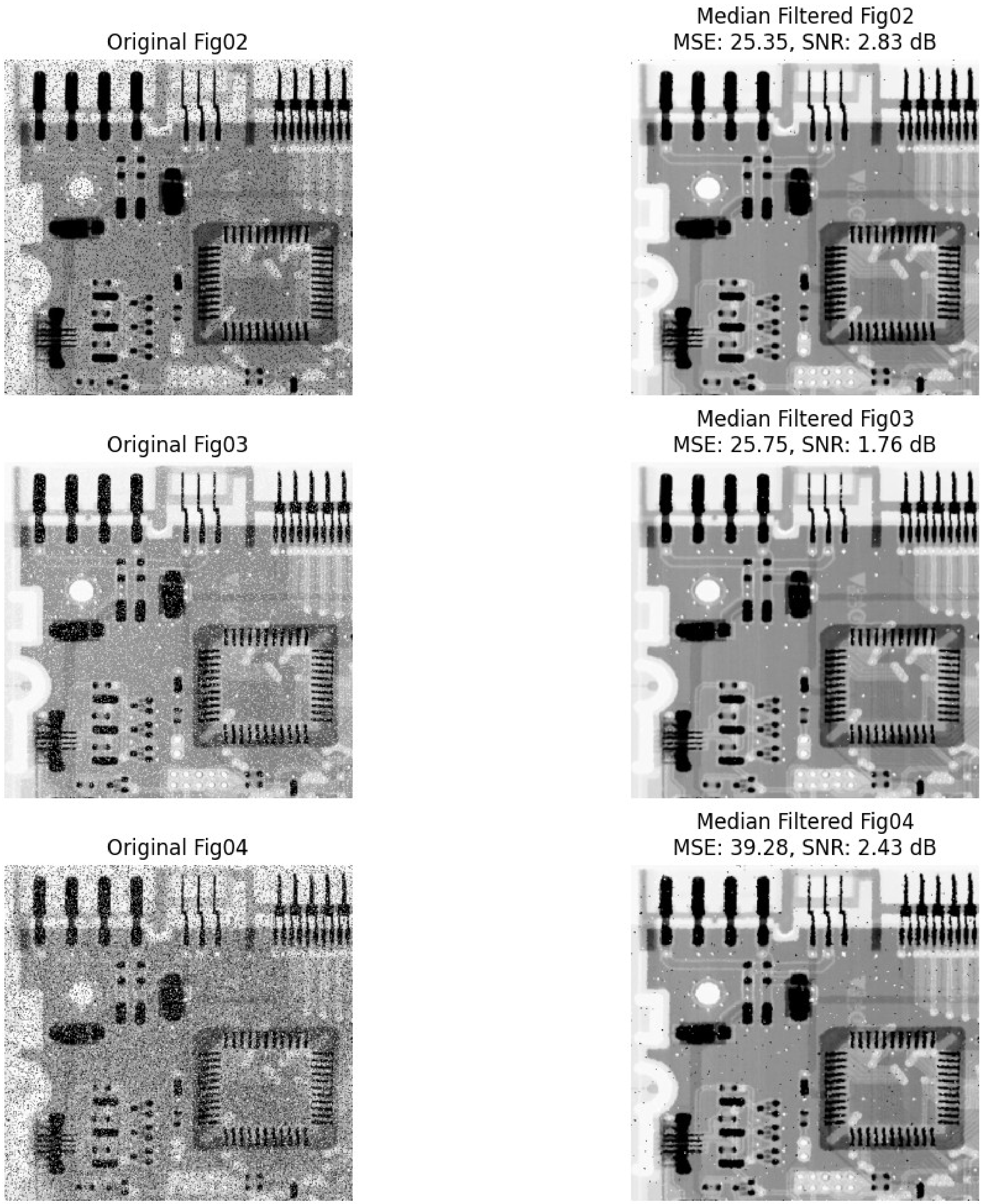


从图像中看到 Q 为 1.5 时可以较好地消除胡椒噪声，但细化并模糊了暗色的区域，Q 为负值时可以较好地消除盐粒噪声，但强化了暗色的区域。

2.3 中值滤波器结果

```
# 加载椒盐噪声影响的图像（Fig04）
image_path_fig04 = 'Fig04.tif'
image_fig04 = cv2.imread(image_path_fig04, 0)  # 加载为灰度图像

# 应用中值滤波器
median_filtered_fig02 = cv2.medianBlur(image_pepper, ksize=3)
median_filtered_fig03 = cv2.medianBlur(image_salt, ksize=3)
median_filtered_fig04 = cv2.medianBlur(image_fig04, ksize=3)

# 计算 MSE 和 SNR
mse_fig02_median = calculate_mse(image_pepper, median_filtered_fig02)
snr_fig02_median = calculate_snr(image_pepper, median_filtered_fig02)
mse_fig03_median = calculate_mse(image_salt, median_filtered_fig03)
snr_fig03_median = calculate_snr(image_salt, median_filtered_fig03)
mse_fig04_median = calculate_mse(image_fig04, median_filtered_fig04)
snr_fig04_median = calculate_snr(image_fig04, median_filtered_fig04)

# 展示结果
plt.figure(figsize=(12, 12))

# Fig02 原图与滤波后图像
plt.subplot(3, 2, 1)
plt.imshow(image_pepper, cmap='gray')
plt.title('Original Fig02')
plt.axis('off')

plt.subplot(3, 2, 2)
plt.imshow(median_filtered_fig02, cmap='gray')
plt.title(f'Median Filtered Fig02\nMSE: {mse_fig02_median:.2f}, SNR:
{snr_fig02_median:.2f} dB')
plt.axis('off')

# Fig03 原图与滤波后图像
plt.subplot(3, 2, 3)
plt.imshow(image_salt, cmap='gray')
plt.title('Original Fig03')
plt.axis('off')

plt.subplot(3, 2, 4)
plt.imshow(median_filtered_fig03, cmap='gray')
plt.title(f'Median Filtered Fig03\nMSE: {mse_fig03_median:.2f}, SNR:
{snr_fig03_median:.2f} dB')
plt.axis('off')

# Fig04 原图与滤波后图像
plt.subplot(3, 2, 5)
plt.imshow(image_fig04, cmap='gray')
plt.title('Original Fig04')
```

```
plt.axis('off')

plt.subplot(3, 2, 6)
plt.imshow(median_filtered_fig04, cmap='gray')
plt.title(f'Median Filtered Fig04\nMSE: {mse_fig04_median:.2f}, SNR:
{snr_fig04_median:.2f} dB')
plt.axis('off')

plt.savefig("中值滤波器对三种噪声影响的图像.png")
plt.show()
```

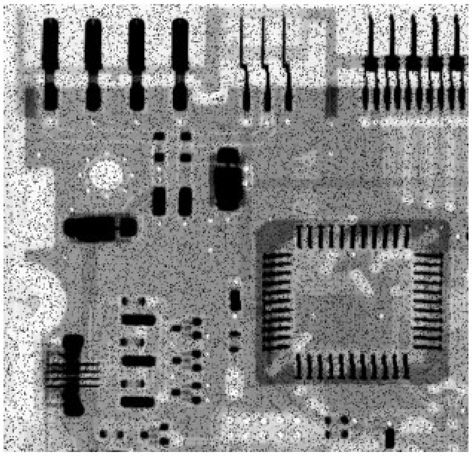


从上图中可以看到中值滤波器对盐粒噪声的消除效果较好，但是对受胡椒噪声影响的图像的效果较差，尝试对 Fig02 和 Fig04 进行多次滤波查看效果。

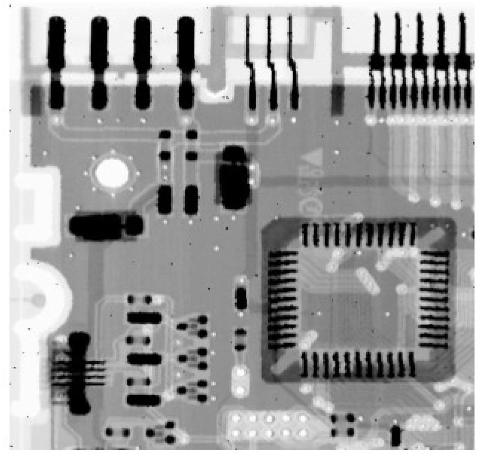
```python
# 对Fig02和Fig04进行三次连续的中值滤波，并计算每次的MSE和SNR

def multi_median_filter(image, num_times):
    """
    对图像应用多次中值滤波
    :param image: 输入图像
    :param num_times: 滤波次数
    :return: 滤波后的图像列表
    """
    filtered_images = []
    current_image = image.copy()
    for _ in range(num_times):
        current_image = cv2.medianBlur(current_image, ksize=3)
        filtered_images.append(current_image.copy())
    return filtered_images

# 对Fig02和Fig04应用三次中值滤波
filtered_images_fig02 = multi_median_filter(image_pepper, 3)
filtered_images_fig04 = multi_median_filter(image_fig04, 3)

# 计算MSE和SNR
mse_snr_fig02 = [(calculate_mse(image_pepper, img),
calculate_snr(image_pepper, img))
                for img in filtered_images_fig02]
mse_snr_fig04 = [(calculate_mse(image_fig04, img),
calculate_snr(image_fig04, img))
                for img in filtered_images_fig04]

# 展示结果
plt.figure(figsize=(12, 18))

# Fig02原图与滤波后图像
plt.subplot(4, 2, 1)
plt.imshow(image_pepper, cmap='gray')
plt.title('Original Fig02')
plt.axis('off')

for i, (img, (mse, snr)) in enumerate(zip(filtered_images_fig02,
mse_snr_fig02), start=2):
    plt.subplot(4, 2, i)
    plt.imshow(img, cmap='gray')
    plt.title(f'Median Filtered Fig02 - {i-1} times\nMSE: {mse:.2f},
SNR: {snr:.2f} dB')
    plt.axis('off')

# Fig04原图与滤波后图像
plt.subplot(4, 2, 5)
plt.imshow(image_fig04, cmap='gray')
plt.title('Original Fig04')
plt.axis('off')
```

```
for i, (img, (mse, snr)) in enumerate(zip(filtered_images_fig04, 
mse_snr_fig04), start=6):
    plt.subplot(4, 2, i)
    plt.imshow(img, cmap='gray')
    plt.title(f'Median Filtered Fig04 - {i-5} times\nMSE: {mse:.2f}, 
SNR: {snr:.2f} dB')
    plt.axis('off')

plt.show()
```

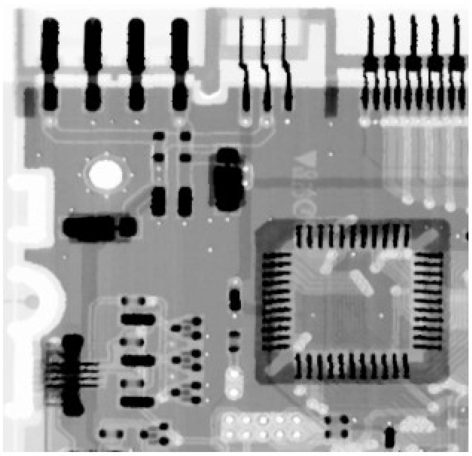
Original Fig02



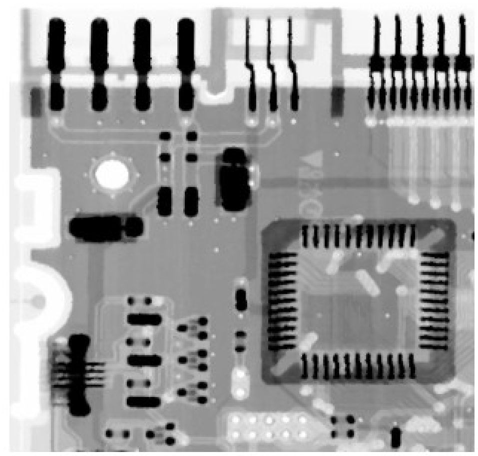
Median Filtered Fig02 - 1 times
MSE: 25.35, SNR: 2.83 dB



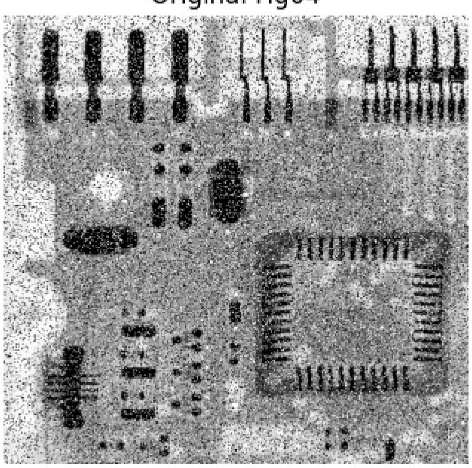
Median Filtered Fig02 - 2 times
MSE: 26.13, SNR: 2.64 dB



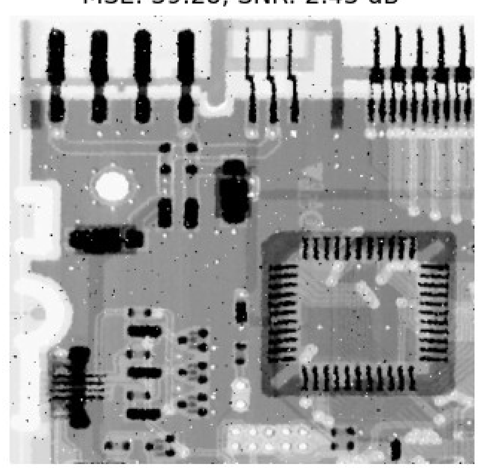
Median Filtered Fig02 - 3 times
MSE: 27.91, SNR: 2.31 dB



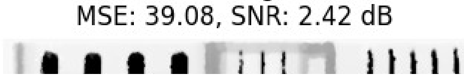
Original Fig04



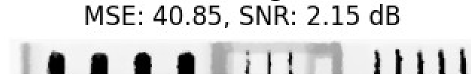
Median Filtered Fig04 - 1 times
MSE: 39.28, SNR: 2.43 dB



Median Filtered Fig04 - 2 times
MSE: 39.08, SNR: 2.42 dB



Median Filtered Fig04 - 3 times
MSE: 40.85, SNR: 2.15 dB



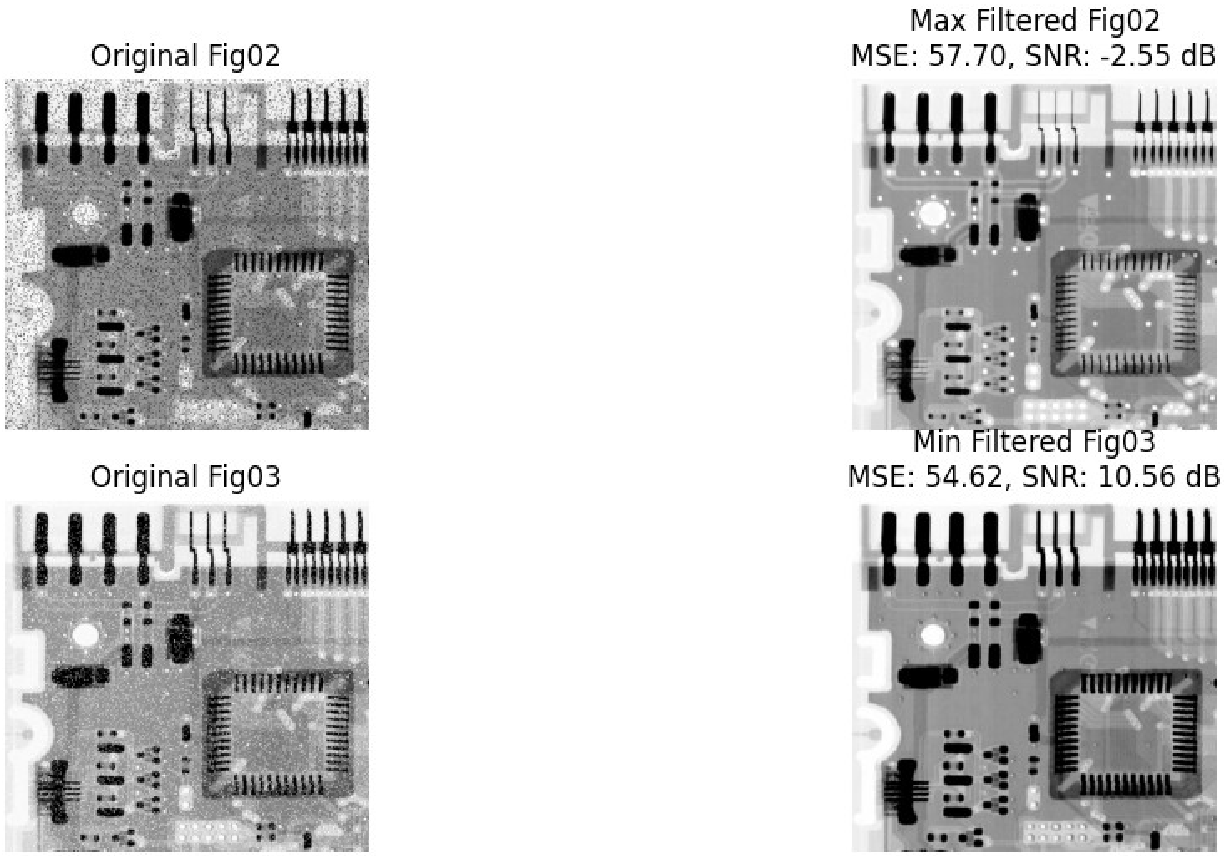
结果分析：

从图像中可以看出，mse 都是在逐渐升高，snr 都是在逐渐降低，图像变得越来越模糊，但是视觉效果变得更好，噪声更少。这可能是由于随着重复利用中值滤波，图像逐渐偏离其原始状态，导致 mse 升高。snr 降低意味着图像的噪声成分相对于信号成分变得更显著，可能是由于重复滤波引起的图像细节损失，使得图像看起来更平滑但也更模糊。虽然噪声减少了，但是图像中的高频信息（如纹理）也被模糊掉了，所以在使用中值滤波时需要找到降噪和保留细节之间的平衡。从上图看出，两次的滤波已经足以去除噪声，同时保留大部分的细节，过度滤波可能导致过度平滑和细节丢失。

2.4 最大最小值滤波器

```
def max_filter(image, kernel_size):
    """
    实现最大值滤波器。
    :param image: 输入图像
    :param kernel_size: 滤波器的大小（必须是奇数）
    :return: 滤波后的图像
    """
    return cv2.dilate(image, np.ones((kernel_size, kernel_size),
np.uint8))

def min_filter(image, kernel_size):
    """
    实现最小值滤波器。
    :param image: 输入图像
    :param kernel_size: 滤波器的大小（必须是奇数）
    :return: 滤波后的图像
    """
    return cv2.erode(image, np.ones((kernel_size, kernel_size),
np.uint8))

# 对 Fig02 应用最大值滤波器（去除胡椒噪声）
max_filtered_fig02 = max_filter(image_pepper, 3)

# 对 Fig03 应用最小值滤波器（去除盐粒噪声）
min_filtered_fig03 = min_filter(image_salt, 3)

# 计算 MSE 和 SNR
mse_max_fig02 = calculate_mse(image_pepper, max_filtered_fig02)
snr_max_fig02 = calculate_snr(image_pepper,max_filtered_fig02)
mse_min_fig03 = calculate_mse(image_salt, min_filtered_fig03)
snr_min_fig03 = calculate_snr(image_salt,min_filtered_fig03)

# 展示结果
plt.figure(figsize=(12, 6))

# Fig02 原图与最大值滤波后图像
plt.subplot(2, 2, 1)
plt.imshow(image_pepper, cmap='gray')
```

```
plt.title('Original Fig02')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(max_filtered_fig02, cmap='gray')
plt.title(f'Max Filtered Fig02\nMSE: {mse_max_fig02:.2f}, SNR:
{snr_max_fig02:.2f} dB')
plt.axis('off')

# Fig03 原图与最小值滤波后图像
plt.subplot(2, 2, 3)
plt.imshow(image_salt, cmap='gray')
plt.title('Original Fig03')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(min_filtered_fig03, cmap='gray')
plt.title(f'Min Filtered Fig03\nMSE: {mse_min_fig03:.2f}, SNR:
{snr_min_fig03:.2f} dB')
plt.axis('off')

plt.savefig("最大最小值滤波器.png")
plt.show()
```



结果分析：

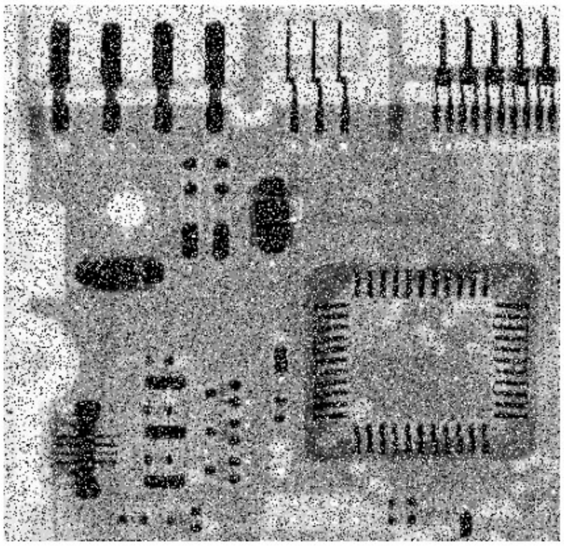
从图中可以看出，最大值滤波器对胡椒造神的效果较好，但细化了暗色区域，最小值滤波器对盐粒噪声的消除效果较好，但强化了暗色区域。这是由于最大值滤波器通过扩大亮区域来去除暗色噪声点，因为将暗色的噪声替换为邻域内较亮的像素值。这导致了原本较暗区域的细节变得不那么明显，甚至造成细节丢失。

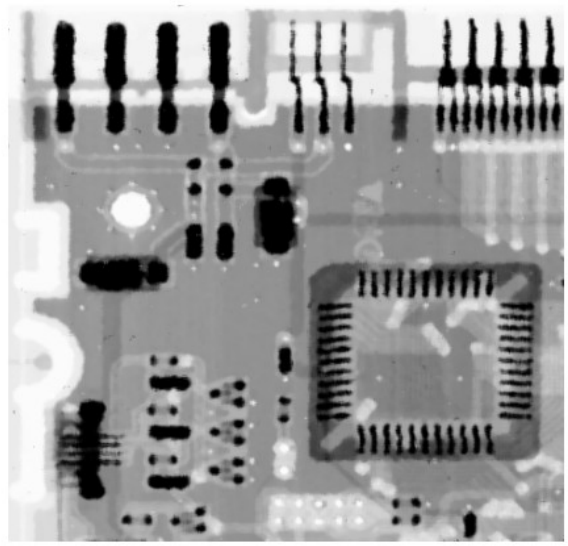
最小值滤波器与最大值滤波器相反。

2.5 修正阿尔法均值滤波器

```
def alpha_trimmed_mean_filter(image, kernel_size, d):
    """
    实现修正阿尔法均值滤波器。
    :param image: 输入图像
    :param kernel_size: 滤波器的大小（必须是奇数）
    :param d: 要修剪的像素值数量（每侧）
    :return: 滤波后的图像
    """
    pad_size = kernel_size // 2
    padded_image = cv2.copyMakeBorder(image, pad_size, pad_size,
pad_size, pad_size, cv2.BORDER_REFLECT)
    filtered_image = np.zeros_like(image, dtype=np.float32)
```

```
    # 定义剪裁的像素数量，确保不会剪裁过多
    d = min(d, kernel_size**2 // 2)

    for i in range(pad_size, padded_image.shape[0] - pad_size):
        for j in range(pad_size, padded_image.shape[1] - pad_size):
            # 提取邻域
            neighborhood = padded_image[i-pad_size:i+pad_size+1, j-
pad_size:j+pad_size+1]
            # 展平邻域并排序
            flattened = np.sort(neighborhood.flatten())
            # 剪裁 d 个最大和最小值
            trimmed = flattened[d:-d] if d > 0 else flattened
            # 计算均值
            filtered_image[i - pad_size, j - pad_size] =
np.mean(trimmed)

    return filtered_image.astype(np.uint8)

# 由于 Fig04 的噪声类型未指定，我们采用中性的参数 d 进行滤波
alpha_trimmed_filtered_fig04 = alpha_trimmed_mean_filter(image_fig04,
kernel_size=5, d=8)

# 计算 MSE 和 SNR
mse_alpha_trimmed_fig04 = calculate_mse(image_fig04,
alpha_trimmed_filtered_fig04)
snr_alpha_trimmed_fig04 = calculate_snr(image_fig04,
alpha_trimmed_filtered_fig04)

# 展示结果
plt.figure(figsize=(12, 6))

# Fig04 原图
plt.subplot(1, 2, 1)
plt.imshow(image_fig04, cmap='gray')
plt.title('Original Fig04')
plt.axis('off')

# 修正阿尔法均值滤波后的图像
plt.subplot(1, 2, 2)
plt.imshow(alpha_trimmed_filtered_fig04, cmap='gray')
plt.title(f'Alpha-Trimmed Mean Filtered Fig04\nMSE:
{mse_alpha_trimmed_fig04:.2f}, SNR: {snr_alpha_trimmed_fig04:.2f} dB')
plt.axis('off')

plt.savefig("修正阿尔法均值滤波器.png")
plt.show()
```





结果分析：

在调试代码的过程中发现，随着 d 的增加，噪声去除的视觉效果变得越来越来平滑，mse 也在逐渐降低，snr 的变化不大。

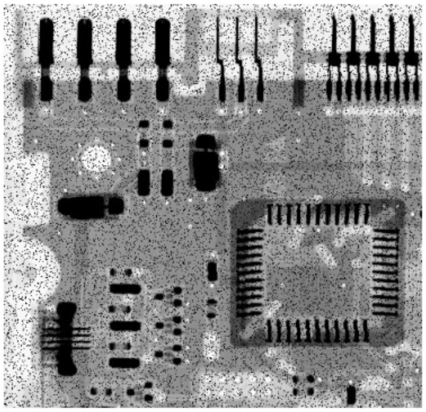
2.6 自适应中值滤波器

```
# 定义自适应中值滤波函数
def adaptive_median_filter(image, max_kernel_size):
    """
    Apply adaptive median filter to an image.
    :param image: numpy.ndarray, the image to apply the filter on.
    :param max_kernel_size: int, the maximum size of the median filter
kernel.
    :return: numpy.ndarray, the filtered image.
    """
    temp_image = image.copy()
    height, width = temp_image.shape
    filtered_image = np.zeros((height, width), dtype=np.uint8)

    for i in range(height):
        for j in range(width):
            k = 1
            z_min, z_med, z_max = 0, 0, 0
            s_max = max_kernel_size // 2
            while k <= s_max:
                kernel = temp_image[max(i - k, 0):min(i + k + 1,
height), max(j - k, 0):min(j + k + 1, width)]
                z_min = np.min(kernel)
                z_med = np.median(kernel)
                z_max = np.max(kernel)
```

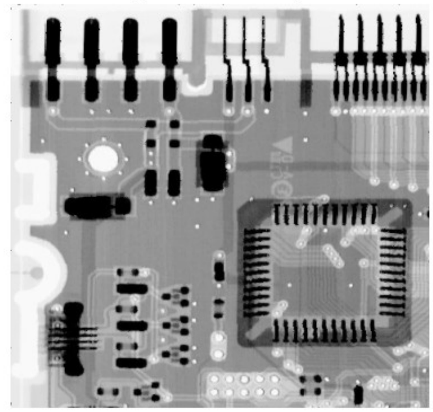
```
                if z_min < z_med < z_max:
                    break
                k += 1
            if z_min < temp_image[i, j] < z_max:
                filtered_image[i, j] = temp_image[i, j]
            else:
                filtered_image[i, j] = z_med
    return filtered_image

# 读取图像
fig02_image = cv2.imread('Fig02.tif', cv2.IMREAD_GRAYSCALE)
fig03_image = cv2.imread('Fig03.tif', cv2.IMREAD_GRAYSCALE)
fig04_image = cv2.imread('Fig04.tif', cv2.IMREAD_GRAYSCALE)

# 最大滤波器核心大小
max_kernel_size = 7

# 对每张图像应用自适应中值滤波器
filtered_fig02 = adaptive_median_filter(fig02_image, max_kernel_size)
filtered_fig03 = adaptive_median_filter(fig03_image, max_kernel_size)
filtered_fig04 = adaptive_median_filter(fig04_image, max_kernel_size)

# 计算每张图像的 MSE 和 SNR
mse_fig02, snr_fig02 = calculate_mse(fig02_image, filtered_fig02),
calculate_snr(fig02_image, filtered_fig02)
mse_fig03, snr_fig03 = calculate_mse(fig03_image, filtered_fig03),
calculate_snr(fig03_image, filtered_fig03)
mse_fig04, snr_fig04 = calculate_mse(fig04_image, filtered_fig04),
calculate_snr(fig04_image, filtered_fig04)

# 展示结果
plt.figure(figsize=(18, 12))

# 原始 Fig02 图像与滤波后的图像
plt.subplot(3, 2, 1)
plt.imshow(fig02_image, cmap='gray')
plt.title('Original Fig02')
plt.axis('off')

plt.subplot(3, 2, 2)
plt.imshow(filtered_fig02, cmap='gray')
plt.title(f'Filtered Fig02 - MSE: {mse_fig02:.2f}, SNR:
{snr_fig02:.2f}')
plt.axis('off')

# 原始 Fig03 图像与滤波后的图像
plt.subplot(3, 2, 3)
plt.imshow(fig03_image, cmap='gray')
plt.title('Original Fig03')
plt.axis('off')
```

```python
plt.subplot(3, 2, 4)
plt.imshow(filtered_fig03, cmap='gray')
plt.title(f'Filtered Fig03 - MSE: {mse_fig03:.2f}, SNR:
{snr_fig03:.2f}')
plt.axis('off')

# 原始 Fig04 图像与滤波后的图像
plt.subplot(3, 2, 5)
plt.imshow(fig04_image, cmap='gray')
plt.title('Original Fig04')
plt.axis('off')

plt.subplot(3, 2, 6)
plt.imshow(filtered_fig04, cmap='gray')
plt.title(f'Filtered Fig04 - MSE: {mse_fig04:.2f}, SNR:
{snr_fig04:.2f}')
plt.axis('off')

plt.tight_layout()
plt.savefig("自适应中值滤波器.png")
plt.show()
```

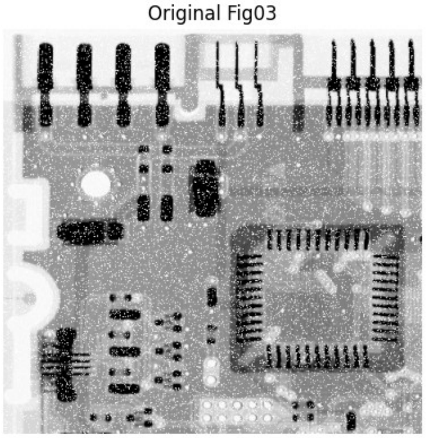
Original Fig02



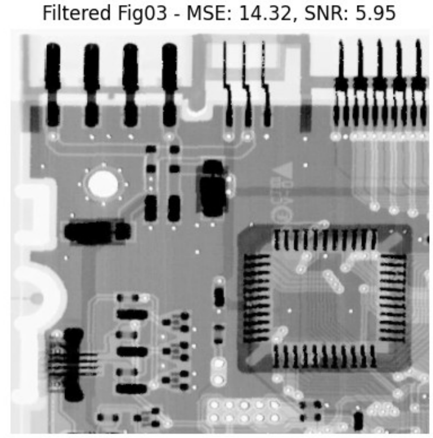
Filtered Fig02 - MSE: 13.79, SNR: 7.21



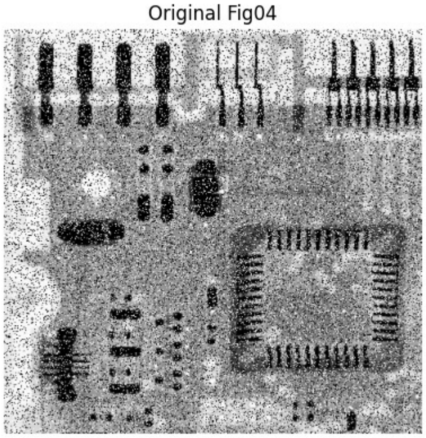
Original Fig03



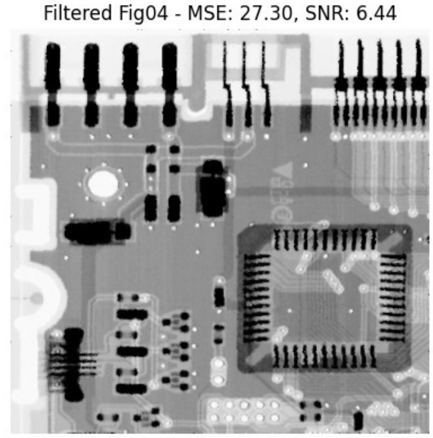
Filtered Fig03 - MSE: 14.32, SNR: 5.95



Original Fig04



Filtered Fig04 - MSE: 27.30, SNR: 6.44



结果分析：

从图中可以看出自适应中值滤波器能够较好的保留图像的细节，得到较低的 mse。