Lab 6 Image Restoration

Jincheng Yu

May 2, 2023

1 Introduction

In this experiment, our aim is to assess the effectiveness of various filters in restoring images that have been corrupted by noise. We have tested four scenarios using different filters: harmonic mean filter, contraharmonic mean filter, adaptive median filter, and Gaussian filter. Additionally, we have explored different methods for restoring degraded images, including full inverse filtering, radially limited inverse filtering, and Wiener filtering.

To be specific, contraharmonic mean filter is a type of image filter used to remove noise from images. It works by calculating the mean of pixel values within a defined neighborhood of each pixel, with the added step of raising each pixel value to a specified power. This filter is particularly useful for removing impulse noise, such as salt-and-pepper noise. Adaptive median filter is a variant of the median filter that can adapt to different levels of noise within an image. It adjusts the size of the neighborhood used for calculating the median based on the amount of noise present in the image. This filter is useful for removing salt-and-pepper noise and other types of noise with varying intensities.

In terms of the results, it was found that the contraharmonic mean filter was effective in filtering out salt and pepper noise after the Q value was adjusted. The adaptive median filter demonstrated good filtering performance on salt and pepper noise with higher noise levels and on images with additive uniform noise and additive salt and pepper noise. In cases where the degradation model was due to atmospheric turbulence, the full inverse filter did not produce satisfactory results. However, when a Butterworth low-pass filter was added, the restored image was acceptable. The Wiener filter produced better results compared to the full inverse filter, even without the use of a low-pass filter.

2 Contraharmonic mean filter

2.1 Mathematical principle

The Contraharmonic Mean filter is a commonly used image filter for noise reduction. It works by calculating the mean of pixel values within a defined neighborhood of each pixel, with the additional step of raising each pixel value to a specified power. The formula for the CHM filter is given as:

$$f(x,y) = \frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}}$$

where f(s,t) represents the image intensity at position (s,t), (x,y) is the location of the pixel being processed, and q is the order of the filter. The filter is particularly effective in removing impulse noise, such as salt-and-pepper noise. The value of q determines the type of noise that the filter can remove. A positive value of q removes pepper noise, while a negative value removes salt noise. The optimal value of q can be determined experimentally based on the type and level of noise present in the image.

2.2 Code

Pseudo code:

```
Data: Input image $I$; Kernel size $S$; Q value $Q$
Result: Output image $J$
1  for $Pointi$ in $((S-1)/2,(S-1)/2)$ to $(I'srows-(S-1)/2, I'scolumns-(S-1)/2)$ do
2      Select the area of I: A from (i-(S-1)/2,i-(S-1)/2) to (i+(S-1)/2,i+(S-1)/2);
3      $numerator = \sum_{(s,t)\in A} I(s,t)^{Q+1}$;
4      $dominator = \sum_{(s,t)\in A} I(s,t)^{Q}$;
5      if $dominator = 0$ then
6          $J(i) = 0$;
7      else
8          $J(i) = \frac{numerator}{dominator}$;
9      end
10 end
```

In python, the specific code has mainly two parts: contraharmonic and harmonic_sum_q. Contraharmonic: It contains the main process of contraharmonic mean filter.

```python
def contraharmonic(arr, size, q):
    dst = arr.copy()
    height, width = arr.shape
    half_size = int((size - 1) / 2)
    for i in range(half_size, height - half_size):
        for j in range(half_size, width - half_size):
            seg = arr[i - half_size:i + half_size, j - half_size:j +
                half_size]
            numerator = harmonic_sum_q(seg, q + 1)
            dominator = harmonic_sum_q(seg, q)
            if dominator == 0:
                dst[i, j] = 0
            else:
                dst[i, j] = numerator / dominator
    return dst
```

harmonic_sum_q: It contains the elements of harmonic expression calculation.

```python
def harmonic_sum_q(arr, q):
    total = 0
    for row in arr:
        for val in row:
            if val == 0 and q < 0:
                return 0
            else:
                total += val ** q
    return total
```

3 Adaptive median filter

3.1 Principle

The basic idea of adaptive median filter is to calculate the local median of the image in a sliding window and then replace the pixel value with the median value only if the difference between the pixel and the median is within a certain threshold.

The algorithm has several stages. In the first stage, the filter examines the window size starting from the minimum size and increases the window size until a pixel within the window is found whose value is outside of the given range. In the second stage, the median value is calculated from the values of the pixels inside the window. In the third stage, the pixel value at the center of the window is compared with the median value. If the difference is below a certain threshold, the pixel value is replaced with the median value; otherwise, the window size is increased and the process is repeated until the difference between the pixel and the median is within the threshold.

3.2 Code

Pseudo code:

```
Data: Input image $I$; Kernel size $S$; Maximum size $M$
Result: Output image $J$
1  for $Pointi$ in $I$ do
2      Set the kernel size $k \leftarrow S$;
3      for $k <= M$ do
4          Select the kernel with size k: A;
5          if minimum of $A$ < median of $A$ < maximum of $A$ then
6              if minimum of $A$ < $I(i)$ < maximum of $A$ then
7                  $J(i) \leftarrow I(i)$;
8              else
9                  $J(i) \leftarrow medianofA$;
10             end
11         else
12             $k \leftarrow k + 2$;
13         end
14     end
15 end
```

In python, the specific code has one method: adaptive_median.

```python
def adaptive_median(arr, window_size=3, max_window_size=7):
    rows, cols = arr.shape
    dst = np.zeros((rows, cols), np.uint16)
    for i in range(rows):
        for j in range(cols):
            window_size_now = window_size
            while window_size_now <= max_window_size:
                window = arr[max(0, i - window_size_now // 2):min(rows
                    , i + window_size_now // 2 + 1),
                          max(0, j - window_size_now // 2):min(cols, j
                             + window_size_now // 2 + 1)]
                median = np.median(window)
                maximum = np.max(window)
                minimum = np.min(window)
                if median - minimum > 0 and median - maximum < 0:
                    if arr[i, j] - minimum > 0 and arr[i, j] - maximum
                         < 0:
                        dst[i, j] = arr[i, j]
                        break
                    else:
                        dst[i, j] = median
                        break
                else:
                    window_size_now += 2
    return dst
```

4 Full inverse filter and limit inverse filter

4.1 Principle

Full inverse filtering and limited inverse filtering are two commonly used techniques for image restoration. The full inverse filter is based on the assumption that the degradation function can be modeled as a linear time-invariant system, and the image can be recovered by applying the inverse filter to the degraded image. However, this approach often amplifies high-frequency noise and may result in unstable solutions.The result is obtained by directly dividing the image by the inverted degradation function:

$$H(u,v) = e^{-k*[(u-M/2)^2+(v-N/2)^2]^{\frac{5}{6}}}$$

$$F(u,v) = \frac{G(u,v)}{H(u,v)}$$

The limited inverse filter, is done by multiplying the inverse filter with a low-pass filter that attenuates high-frequency noise. The low-pass filter can be designed based on the cutoff frequency, which determines the maximum frequency that can be restored.

4.2 Code

Pseudo code:

```
   Data: Input image $I$;
   Result: Output image $J$
1  for $Pointi = (u,v)$ in $I$ do
2      Mask H: $H(u,v) = e^{-k*[(u-M/2)^2+(v-N/2)^2]^{\frac{5}{6}}}$;
3  end
4  Calculate fft of input image: I_f;
5  The fft of output image: J_f=I_f*H;
6  Calculate output image: J = ifft(J_f);
```

In python, the specific code has one method: inverse_filter.

```python
def inverse_filter(img):
    rows, cols = img.shape
    center = [int(rows / 2), int(cols / 2)]
    mask = np.zeros((rows, cols), np.float32)
    for i in range(rows):
        for j in range(cols):
            mask[i, j] = np.exp(-(-0.0025 * ((i - center[0]) ** 2 + (j
                - center[1]) ** 2) ** (5 / 6)))
    return mask
```

And the limit inverse filter has just one more step after inverse filter, to generate a butterworth filter mask multiplied with the frequency output of inverse filter. In python, the specific code has one method: butterworth_filter.

```
def butterworth_filter(img, radius=30, n=2):
    rows, cols = img.shape
    center = [int(rows / 2), int(cols / 2)]
    mask = np.zeros((rows, cols), np.float32)
    for i in range(rows):
        for j in range(cols):
            distance_u_v = (i - center[0]) ** 2 + (j - center[1]) ** 2
            mask[i, j] = 1 / (1 + (distance_u_v ** 0.5 / radius) ** (2
                * n))
    return mask
```

5 Wiener filter

5.1 Principle

The Wiener filter is a widely used technique for image restoration, which aims to recover a degraded image by estimating the underlying signal and the noise components. The Wiener filter is based on the assumption that the degraded image can be modeled as a linear time-invariant system, and both the signal and noise components are characterized by their power spectral densities. Mathematically, the Wiener filter can be expressed as:

$$F(u,v) = [\frac{|H(u,v)|^2}{H(u,v)|H(u,v)|^2 + S(u,v)/S_f(u,v)}]G(u,v)$$

But in fact, it is hardly to know the noise data, and the power spectrum of the ungraded image. Therefore, a parameter K is usually used to represent the $S(u,v)/S_f(u,v)$. And by interactively selecting K the effect of restoration can be changed.

5.2 Code

Pseudo code:

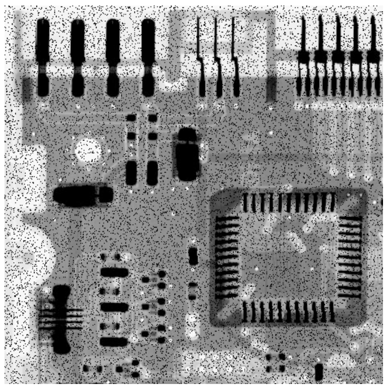
```
Data: Input image $I$; K value $K$
Result: Output image $J$
for $Pointi = (u, v)$ in $I$ do
    Mask H: $H(u,v) = \frac{|H(u,v)|^2}{H(u,v)|H(u,v)|^2+K}$;
end
Calculate fft of input image: I_f;
The fft of output image: J_f=I_f*H;
Calculate output image: J = ifft(J_f);
```

In python, the specific code has one method: wiener_filter.

```python
def wiener_filter(img, K=0.001):
    rows, cols = img.shape
    center = int(rows / 2), int(cols / 2)
    mask = np.zeros((rows, cols), np.float32)
    for u in range(rows):
        for v in range(cols):
            mask[u, v] = np.exp(-0.0025 * ((u - center[0]) ** 2 + (v -
                center[1]) ** 2) ** (5 / 6))
    mask = np.conj(mask) * mask / (mask * (np.conj(mask) * mask + K))
    return mask
```

6 Experiment

6.1 Q6_1_1


(a) Q6_1_1

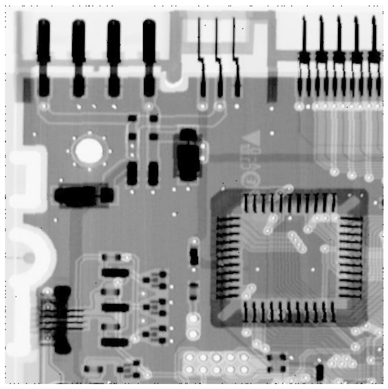
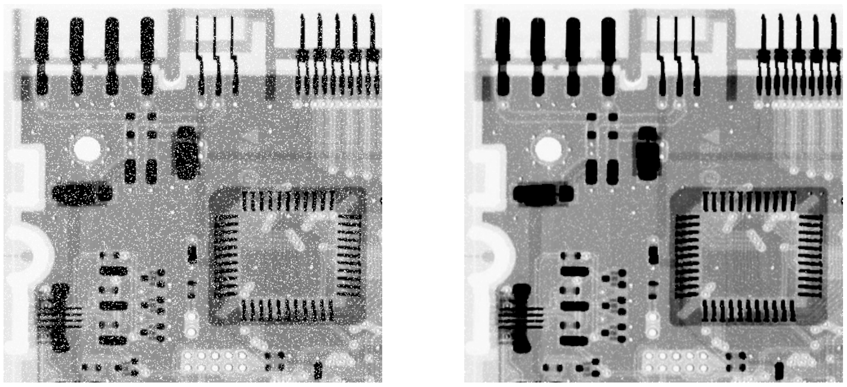

(b) Output

Figure 1: Contraharmonic Mean Filter(Q=1.5)

From the figure we can see, Q6_1_1 mainly contains pepper noise relating Q>0. And we can see the contraharmonic mean filter with Q=1.5 shows optimal performance on the original figure.

6.2 Q6_1_2

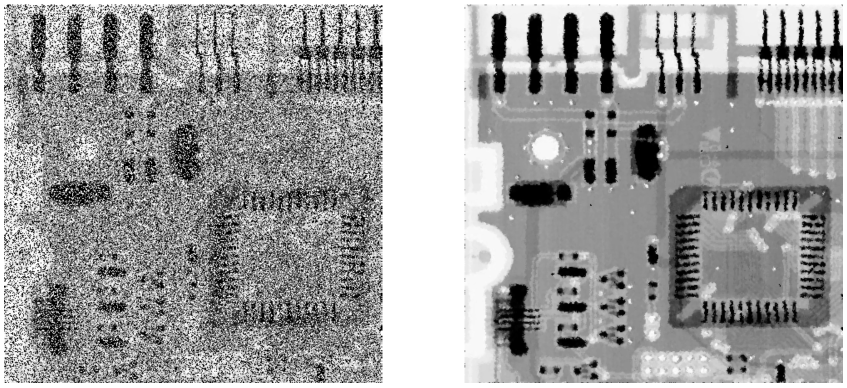


(a) Q6_1_2 (b) Output

Figure 2: Contraharmonic Mean Filter(Q=-1.5)

From the figure we can see, Q6_1_2 mainly contains salt noise relating Q<0. And we can see the contraharmonic mean filter with Q=-1.5 shows optimal performance on the original figure.

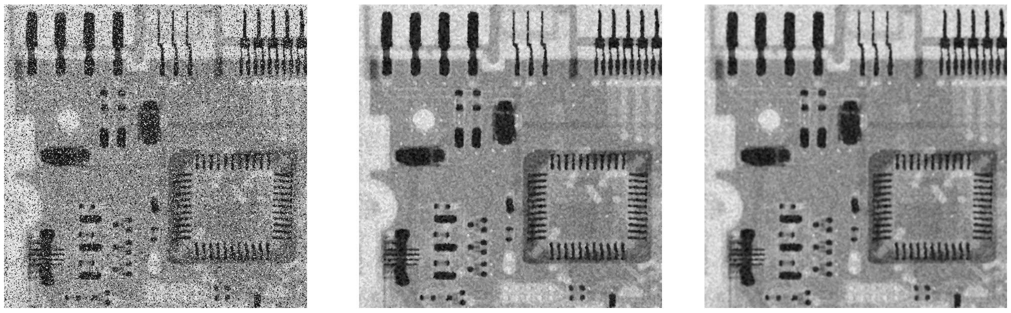
6.3 Q6_1_3



(a) Q6_1_3 (b) Output

Figure 3: Adaptive median filter

For this question, an adaptive median filter with a minimum window size of 3 and a maximum window size of 7 was utilized, and a median filter with a window size of 7 was used for comparison. The results indicate that the adaptive median filter yields satisfactory outcomes.



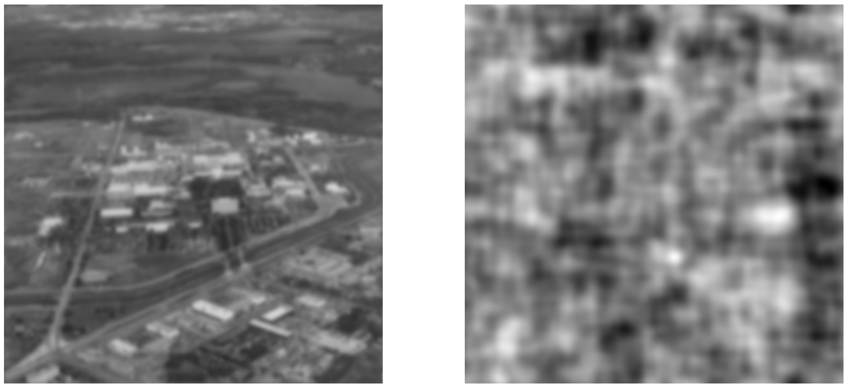
(a) Q6_1_4 (b) Intermediate (c) Output

Figure 4: Adaptive median filter combined with Gaussian filter

6.4 Q6_1_4

For this question, we can observe that the original figure contains both salt-pepper noise and uniform noise. A median filter can be effective to both noise. Therefore, the first step is adaptive median filter. And its result is shown in Fig.4(b). We can see most noise has been eliminated and the characteristic of figure has clear. But there are also many noise we cannot expel in first step. So, we choose to use Gaussian filter to blur the image to fade the noise and make the image more natural.

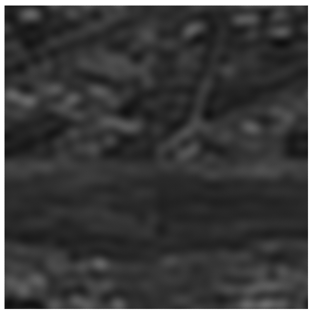
6.5 Q6_2



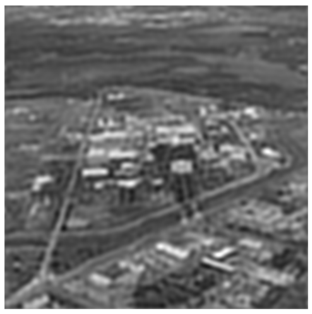
(a) Q6_2 (b) Output

Figure 5: Full inverse filter

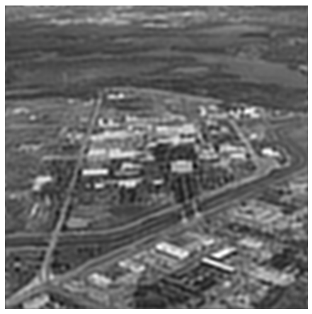
Initially, we applied the full inverse filter to the image, but it produced an almost unrecognizable output. This can be attributed to the fact that the degraded model is relatively insignificant when compared to the noise, resulting in a poor performance.



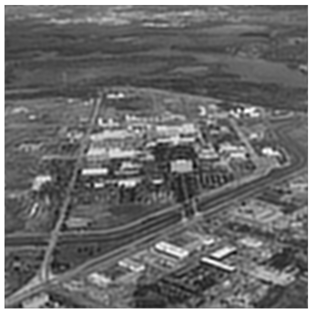
(a) radius= 20



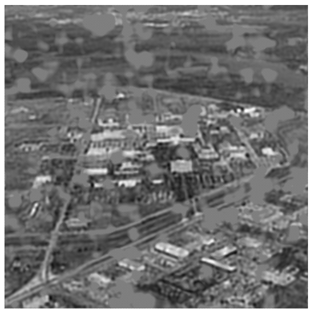
(b) radius= 40



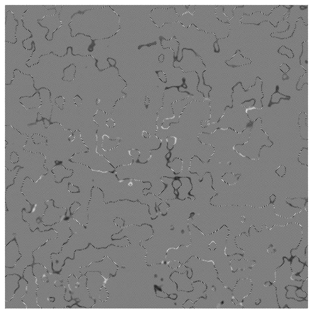
(c) radius= 50



(d) radius= 60



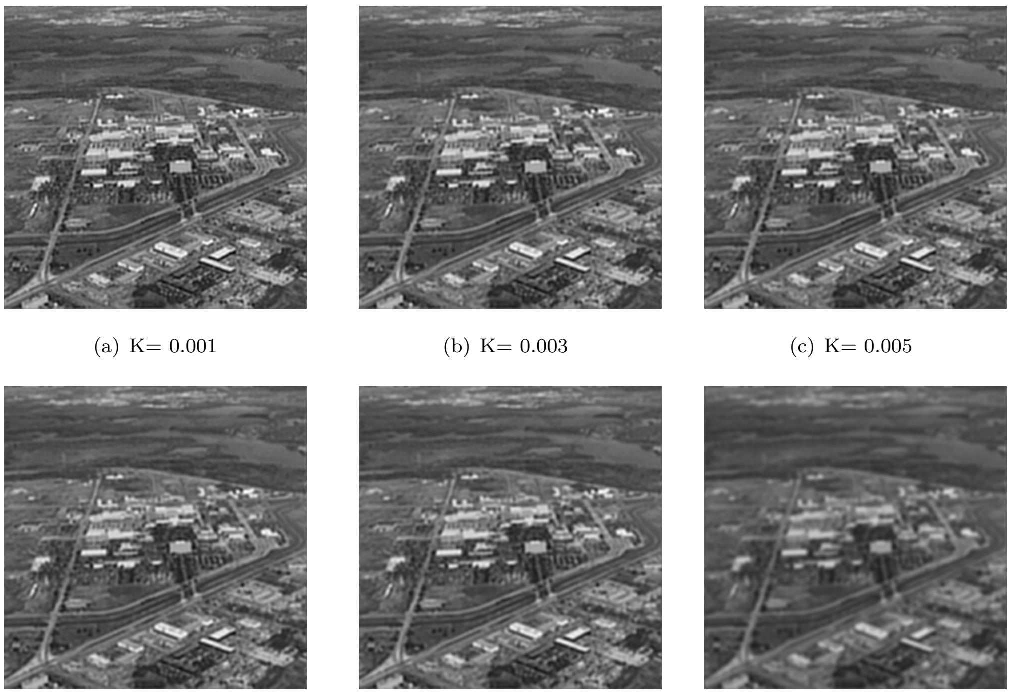
(e) radius= 70



(f) radius= 80

Figure 6: Full filter with different butterworth radius

By analyzing the figures, it is evident that the output image shows poor performance when the Butterworth radius is either too small or too large. In contrast, a radius value between 40 and 60 produces clearer outputs.


(a) K= 0.001 (b) K= 0.003 (c) K= 0.005

(d) K= 0.007 (e) K= 0.01 (f) K= 0.1

Figure 7: Full filter with different butterworth radius

By analyzing the figures, it is hard to distinguish the figure (a)-(e) but figure (f) has poor performance. Therefore, K between 0.001 and 0.01 should be appropriate value for restoration image.

7 Efficiency analysis

The two noise filter we used in this lab have different complexity. As for adaptive median filter, the time complexity of this algorithm is O(n^4), where n is the size of the window. In the worst case, each pixel needs to traverse all possible combinations of the maximum window size, thus requiring a time complexity of O(n^4).
The space complexity of this algorithm is O(n^2), where n is the size of the image. An output array of the same size as the original image needs to be created. The algorithm uses a sliding window, and the window size varies, but does not exceed max_window_size, so the window size is independent of the image size and does not affect the space complexity.

As for the Contraharmonic Mean Filter, the time complexity of this algorithm is O($N^2 * size^2$), where N is the size of the image and size is the size of the filter. For each pixel, the algorithm needs to iterate through each element of the filter and calculate the numerator and denominator, so the time complexity depends on the size of the image and the filter.
The space complexity of this algorithm is O(N^2), as it requires creating an output array of the same size as the original image. The algorithm uses a sliding window of size equal to the filter size, so the space complexity depends on the size of the image and is independent of the filter size.

And the complexity of full inverse filter, limit inverse filter and wiener filter are all identical and fixed. The time and space complexity are both $O(n^2)$ because it involves walking through all pixels in image.

8 Conclusion

In conclusion, Contraharmonic mean filter, median filter, and adaptive median filter are commonly used for image noise reduction. The contraharmonic mean filter is useful for removing noise that is caused by salt and pepper noise. Median filter is effective in removing impulse noise from an image, while adaptive median filter is better at preserving edges and fine details in the image.
Inverse filtering and Wiener filtering are commonly used to restore degraded images. Inverse filtering is useful for removing blur caused by a linear shift-invariant system, but it can amplify noise in the image. Wiener filtering is more robust to noise and is effective in restoring degraded images that have been corrupted by additive noise. Wiener filtering works by estimating the frequency response of the degradation system and then filtering the degraded image in the frequency domain.