Report for EQ2330 Image and Video Processing

EQ2330 Image and Video Processing, Project 1

Hanqi Yang hanqi@kth.se

Qian Zhou qianzho@kth.se

November 22, 2021

Summary

The goal of this project is to utilize three different image enhancement techniques to process images with different features. They are histogram equalization, spatial filtering with mean and median filter and frequency domain filtering. The result indicates that histogram equalization can improve low-contrast image. Besides, both mean and median filtering are effective in recovering Gaussian noise. Median filtering is much better suited than averaging for the removal of salt&pepper noise. Wiener filter can be used to deblur degraded images.

1 Introduction

There are many ways to process images for different features. In this project, we discuss three categories of image processing methods. First, histogram equalization is used to improve the low-contrast image. Second, for images disturbed by Gaussian noise and salt&pepper noise, mean filter and median filter are used separately to denoise them. Finally, we apply Wiener filter to deblur the distorted image.

In this report, we assume a 8-bit representation for the gray level value, e.g., the gray level of a pixel can range from 0 to $2^8 - 1 = 255$. $f(x, y)$ can be used to represent a digital image where the output is the gray level at the point (x, y).

2 System Description

2.1 Histogram equalization

To generate a low-contrast image, we reduce the dynamic range of the image, e.g.,

$$g(x, y) = \min(\max(\lceil a \cdot f(x, y) + b \rfloor, 0), 255), \quad (1)$$

where $\lceil \cdot \rfloor$ denotes the round operator, $0 < \mathrm{a} < 1$, $0 < \mathrm{b} < 255(1 - \mathrm{a})$. Use $a = 0.2$ and $b = 50$.

One method of improving low-contrast images is histogram equalization, because the histogram of low-contrast image is quite narrow. For discrete value, the probability of occurrence of gray level r_k in a digital image is approximated as

$$p_r(r_k) = \frac{n_k}{MN}, k = 0, \ldots, L - 1, \quad (2)$$

where MN is the total number of pixels in the image, n_k is the number of pixels

with a gray level of r_k, and L is the number of possible gray levels in the image (e.g. 256 for an 8-bit image). the graph of $p_r(r_k)$ relative to r_k is often called histogram.
A particularly important transformation function in image processing is given in [1], and its discrete form is

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) = \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j, k = 0, \ldots, L-1. \quad (3)$$

Thus, the output image can be obtained by mapping each pixels in the input image with gray level r_k into a corresponding pixels with level s_k in the output image.

2.2 Image denoising

Two types of noise, Gaussian noise and salt&pepper noise, are investigated separately. For Gaussian noise with zero mean and variance 64, it is additive. For salt&pepper noise, both the probability of random pixels transformed to 0 or 255 are 5%.
We use spatial smoothing filter and order-statistics filter to reduce the effect of noise. The first filter is a 3×3 mean filter which is applied by convolution to the noisy image,

$$g(x,y) = h(x,y) * f(x,y). \quad (4)$$

where $g(x,y)$ is the denoised image, $f(x,y)$ is the noisy image and $h(x,y)$ is the function of the mean filter with mask

$$h(x,y) = \frac{1}{9}\begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}. \quad (5)$$

The second filter is a 3×3 median filter which replace the value of a pixel by the median of the gray levels in the neighborhood of that pixel.

2.3 Frequency domain filtering

The blurred image can be generated using the degradation model

$$g(x,y) = h(x,y) * f(x,y) + \eta(x,y). \quad (6)$$

where $g(x,y)$ is the blurred image, $h(x,y)$ is the function generating the Gaussian blur kernel with radius $r = 8$, $f(x,y)$ is the original image and $\eta(x,y)$ is a Gaussian noise with zero mean and variance 0.0833.
We use Wiener filter to deblur the degraded image. It is assume that the noise and the image are uncorrelated; that one or the other has zero mean; and that the gray levels in the estimate are a linear function of the levels in the degraded image. Based on these condition, the Wiener filter is given by

$$\widehat{F}(\omega_x, \omega_y) = \left[\frac{1}{H(\omega_x, \omega_y)} \cdot \frac{\left|H(\omega_x, \omega_y)\right|^2}{\left(\left|H(\omega_x, \omega_y)\right|^2 + \frac{\Phi_{\eta\eta}(\omega_x, \omega_y)}{\Phi_{ff}(\omega_x, \omega_y)}\right)}\right] G(\omega_x, \omega_y) \quad (7)$$

where $H(\omega_x, \omega_y)$ is the Fourier transform of degradation function, $\Phi_{\eta\eta}(\omega_x, \omega_y)/\Phi_{ff}(\omega_x, \omega_y)$ is the Noise-to-Undegraded-Signal ratio and $G(\omega_x, \omega_y)$ is the transform of degraded image. The restored image is given by the inverse Fourier transform of $\widehat{F}(\omega_x, \omega_y)$. Since $\Phi_{ff}(\omega_x, \omega_y)$ is unknown, a constant K can be used to approximate $\Phi_{\eta\eta}(\omega_x, \omega_y)/\Phi_{ff}(\omega_x, \omega_y)$.

3 Results

3.1 Histogram equalization

The histogram of original, low-contrast and equalized images are shown in Figure 1. The gray level of the original image is distributed between 0 and 255, while the gray level of the low-contrast image is compressed to a range of 50 to 101. After equalization, the gray level of the image is again distributed between 0 and 255 with higher values of occurrences. Different from the continuous case, the equalized histogram is not flat because the histogram of the image is a discrete approximation of the PDF. The original, low-contrast and equalized images are shown in Figure 2. A clear difference can be seen in the contrast of the three images.

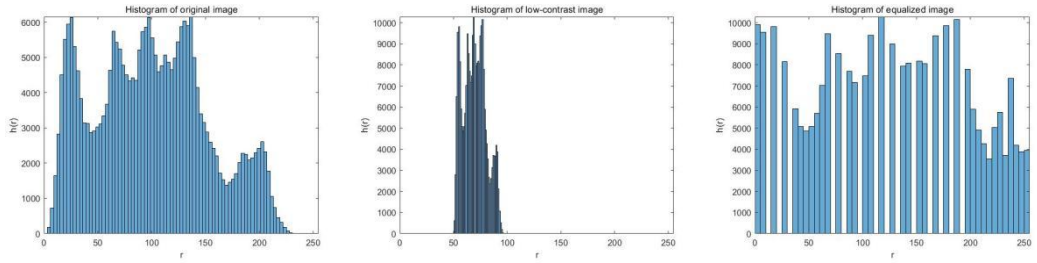

Figure 1: Histogram of original, low-contrast and equalized images. The gray level is denoted r and h(r) is the number of pixels with gray level r.

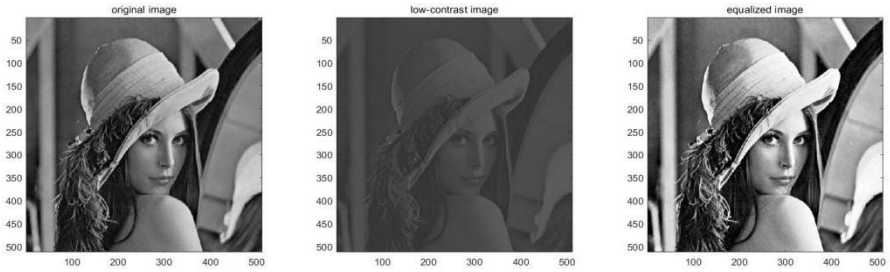

Figure 2: Images of “Lena” after different processing.

3.2 Image denoising

We investigate the denoising effect of spatial smoothing filters and order-statistics filters. Histogram of images with Gaussian noise and salt&pepper noise are shown Figure 3 and noisy images of “Lena” are shown in Figure 4. Compare to the original image in Figure 1, it can be seen that the sharp peaks of the histogram are smoothed out after the Gaussian noise has been added and the details of the original image are blurred. After adding the salt&pepper noise, the main part of the histogram does not change much, however there is a large increase in the gray level of 0 and 255. Looking at the image you can see that a large amount of black and white noise appears.

First we apply mean filter to the two noisy images, the peaks in the histogram of the Gaussian noise image reappear, meaning that image detail is restored. For the salt&pepper noise image, although the values of gray level of 0 and 255 are gone, the peaks in the histogram are also erased and a lot of noise is still visible in the image.

Median filter was then applied to the two noisy images and the recovery of the Gaussian noisy images did not differ much from the recovery of the mean filter. The recovered salt&pepper noise image is almost indistinguishable from the original, although a very small amount of black and white noise is still visible in the image.

In summary, both mean and median filtering are effective in recovering Gaussian noise. Median filtering is much better suited than averaging for the removal of salt&pepper noise. This is because that mean filtering uses a linear approach, averaging pixel values over the entire window. Mean filtering destroys the detailed parts of the image while denoising the image, thus blurring it. Median filtering uses a non-linear approach and it is very effective in smoothing impulse noise. It also protects the sharp edges of the image and selects appropriate points to replace the values of contaminated points, so it handles salt&pepper noise well.

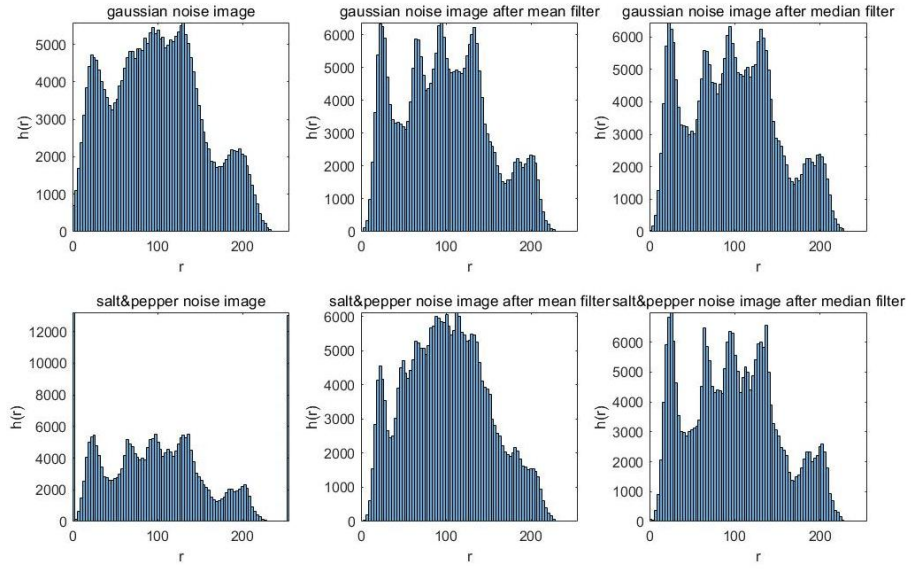

Figure 3: Histogram of Gaussian noise image and salt&pepper noise image before and after mean filter and median filter

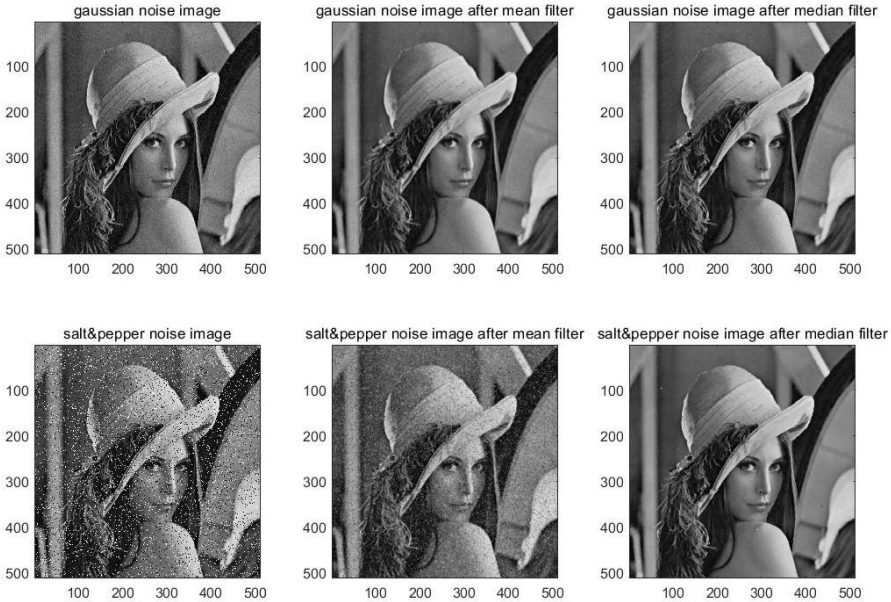

Figure 4: Noisy images of "Lena" before and after mean filter and median filter

3.3 Frequency domain filtering

We blur the “Lena” image and the Fourier spectra of the images before and after degradation are shown in Figure 5. As can be seen from the spectrum, the brighter points of the original image are spread throughout the spectrum, while the brighter points of the blurred image are concentrated around the axes and the origin, which means that the high frequency component is reduced and therefore the image becomes blurred. We then used the Wiener filter according to equation (7) to deblur the degraded image and the results are shown in Figure 6. When the input image has sharp edges on the boundaries, black and white lines will appear at the edge of the recovered image, which is caused by ringing artifact. The ringing artifact is caused by the residual waves on both sides of the sinc function in time domain, that is, sharp changes at the edge of LPF in frequency domain. To avoid ringing, we use the *padarray()* to pad the image by replicating the boundary pixel values before passing them to *fft2()*. After filtering, we cut the padded part of the deblurred image. In this way, the edges of the image taper off to a lower frequency.

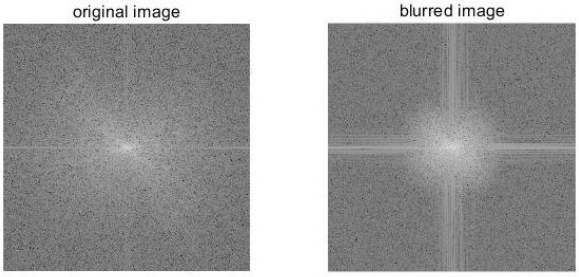

Figure 5: The Fourier spectra of the images before and after degradation

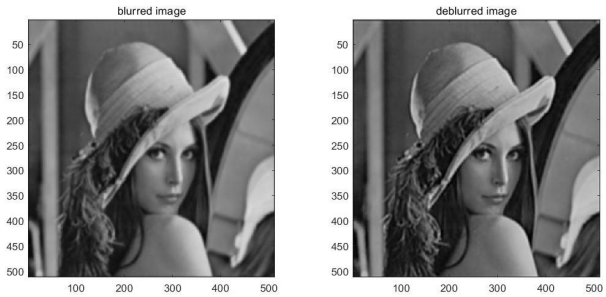

Figure 6: Comparison of the blurred and deblurred images

4 Conclusions

Histogram can be used to detect whether the image has low-contrast, which in that case will be quite narrow. After histogram equalization, the gray level of the image is again distributed between 0 and 255 with higher values of occurrences. From the image denoising module, we found that mean filter is effective only in removing Gaussian noise, while median filter works well in removing both Gaussian noise and salt&pepper noise. In the last part, we compare the difference between the original and blurred images in frequency domain. We design the Wiener filter to deblur the degraded image and avoid the ringing artifacts as far as possible.

Reference

[1] Rafael C. Gonzalez and Richard E. Woods, Digital Image Processing, Prentice Hall, 2nd ed., 2002

Appendix

The project was done collaboratively by both authors.

Matlab Code

- **Histequ.m**

```
% 2.1 Histogram equalization
clc;
clear all;

% Plot the histogram (with 8 bits resolution) of the input image
L=2^8;% grey level;
f=imread('images/lena512.bmp');
figure(1);
histogram(f);
xlabel('r');
ylabel('h(r)');
title('Histogram of original image');
axis([0 255 0 inf])

% Simulate a low-contrast image by reducing the dynamic range of the
image
a=0.2;
b=50;
g=a*f+b;
[Row,Col]=size(g);
for i=1:Row
    for j=1:Col
        if g(i,j)>255
            g(i,j)=255;
        elseif g(i,j)<0
            g(i.j)=0;
        end
    end
end
figure(2);
histogram(g);
xlabel('r');
ylabel('h(r)');
```

```
title('Histogram of low-contrast image');
axis([0 255 0 inf]);

% Plot the histogram of the enhanced image.
n_k=imhist(g);
p_r=n_k./(Row*Col);
for i=1:length(p_r)
    sum=cumsum(p_r(1:i));
    s_k(i)=sum(end);
end
s_k=(L-1)*s_k;
for i=1:Row
    for j=1:Col
        g_eq(i,j)=s_k(g(i,j));
    end
end
figure(3);
histogram(g_eq);
xlabel('r');
ylabel('h(r)');
title('Histogram of equalized image');
axis([0 255 0 inf]);

% Display the images
figure(4);
subplot(1,3,1);
imagesc(f,[0 255]);
title('original image');
subplot(1,3,2);
imagesc(g,[0 255]);
title('low-contrast image');
subplot(1,3,3);
imagesc(g_eq,[0 255]);
title('equalized image');
colormap gray(256);
```

- **Imgdenoising.m**

```
% 2.2 Image denoising
clc;
clear all;

% Plot the histogram (with 8 bits resolution) of the input image
L=2^8;% grey level;
f=imread('lena512.bmp');
```

```
% Generate and apply noise
gaussian_n = mynoisegen('gaussian', 512, 512, 0, 64);
saltp_n = mynoisegen('saltpepper', 512, 512, 0.05, 0.05);

f_gaussian = double(f) + gaussian_n;
f_saltp = f;
f_saltp(saltp_n==0) = 0;
f_saltp(saltp_n==1) = 255;
f_saltp = uint8(f_saltp);

% Apply mean filter
meanfilter = (1/9)*ones(3,3);
f_gaussian_meanfilt = conv2(double(f_gaussian), double(meanfilter),
'same');
f_saltp_meanfilt = conv2(double(f_saltp), double(meanfilter),
'same');

% Apply median filter
f_gaussian_medfilt = medfilt2(f_gaussian);
f_saltp_medfilt = medfilt2(f_saltp);

% Display histogram
figure(1);
subplot(2,3,1);
histogram(f_gaussian);
xlabel('r');
ylabel('h(r)');
title('gaussian noise image');
axis([0 255 0 inf]);
subplot(2,3,2);
histogram(f_gaussian_meanfilt);
xlabel('r');
ylabel('h(r)');
title('gaussian noise image after mean filter');
axis([0 255 0 inf]);
subplot(2,3,3);
histogram(f_gaussian_medfilt);
xlabel('r');
ylabel('h(r)');
title('gaussian noise image after median filter');
axis([0 255 0 inf]);
subplot(2,3,4);
histogram(f_saltp);
```

```
xlabel('r');
ylabel('h(r)');
title('salt&pepper noise image');
axis([0 255 0 inf]);
subplot(2,3,5);
histogram(f_saltp_meanfilt);
xlabel('r');
ylabel('h(r)');
title('salt&pepper noise image after mean filter');
axis([0 255 0 inf]);
subplot(2,3,6);
histogram(f_saltp_medfilt);
xlabel('r');
ylabel('h(r)');
title('salt&pepper noise image after median filter');
axis([0 255 0 inf]);

% Display the 'Lena' image
figure(2);
subplot(2,3,1);
imagesc(f_gaussian,[0 255]);
title('gaussian noise image');
subplot(2,3,2);
imagesc(f_gaussian_meanfilt,[0 255]);
title('gaussian noise image after mean filter');
subplot(2,3,3);
imagesc(f_gaussian_medfilt,[0 255]);
title('gaussian noise image after median filter');
subplot(2,3,4);
imagesc(f_saltp,[0 255]);
title('salt&pepper noise image');
subplot(2,3,5);
imagesc(f_saltp_meanfilt,[0 255]);
title('salt&pepper noise image after mean filter');
subplot(2,3,6);
imagesc(f_saltp_medfilt,[0 255]);
title('salt&pepper noise image after median filter');
colormap gray(256);
```

- **Freq_domain_filtering.m**

```
% 3 Frequency domain filtering
f = imread('lena512.bmp');
r = 8; % radius
[Row,Col] = size(f);
```

```
n_mean = 0;
n_var = 0.0833;

% blur the image
h = myblurgen('gaussian',r); % generate h(x,y)
gaussian_n = mynoisegen('gaussian', Row, Col, n_mean, n_var); %
generate noise
f_blur = conv2(double(f),h,'same');
g = f_blur + gaussian_n;
for i=1:Row
    for j=1:Col
        if g(i,j)>255
            g(i,j)=255;
        elseif g(i,j)<0
            g(i,j)=0;
        end
    end
end

% Fourier transform
F = fft2(im2double(f));
F = fftshift(F);%  center the spectra
F = real(F);
F_log = log(F+1); % Log transform
G = fft2(im2double(g));
G = fftshift(G);%  center the spectra
G = real(G);
G_log = log(G+1); % Log transform

% plot the Fourier spectra
figure(1);
subplot(1,2,1);
imshow(F_log,[]);
title('original image');
subplot(1,2,2);
imshow(G_log,[]);
title('blurred image');

% image deblurring
g_deblur = wienerfilter(g,h,0.0833);

% plot the image
figure(2);
subplot(1,2,1);
```

```
imagesc(g,[0 255]);
title('blurred image');
subplot(1,2,2);
imagesc(g_deblur,[0 255]);
title('deblurred image');
colormap gray(256);
```

- **wienerfilter.m**

```
function F_hat = wienerfilter(g,h,K)
% Wiener filter
% Input: g(x,y) the blurred image
%       h(x,y) the blur function
%       K noise-to-undegraded-signal ratio

[M,N]=size(g);
[m,n]=size(h);
g1 = padarray(g,[m,n],'replicate','both');

M1 = M + 2*m;
N1 = N + 2*n;

% zero padding
if m<M1||n<N1
   h(M1,N1)=0;
end

H = fft2(h);
G = fft2(g1);

for i = 1:M1
   for j = 1:N1
      F(i,j)=1/H(i,j)*(abs(H(i,j)))^2/((abs(H(i,j)))^2+K)*G(i,j);
   end
end

% cut the padded edge
F_hat = uint8(ifft2(F));
cut1 = (m+1)/2;
cut2 = (n+1)/2;
F_hat =  F_hat(cut1:M+cut1-1,cut2:N+cut2-1);
end
```