EQ2330 Image and Video Processing

EQ2330 Image and Video Processing, Project 1

LO, King Lam
kllo@kth.se

TSANG, Chin Yung Virginia
cyvtsang@kth.se

CHIU, Lok Ying
lychiu@kth.se

November 21, 2022

Summary

In this project we investigate various spatial and frequency domain image enhancement techniques. We demonstrate the usage of histogram equalization and its property with a low-contrast image, then the denoising effect of a mean and median filter in the spatial domain. In the frequency domain, we use the Wiener filter, a deblurring application of frequency domain filtering, to restore an out-of-focus image.

1 Introduction

There are many different restoration techniques depending on the type of image degradation. In this report we will demonstrate spatial and frequency domain enhancement techniques. We use an 8-bit representation for the gray level value i.e. 0 to 255. The spatial domain refers to the image plane and the image processing methods in this category are based on manipulations of pixels. The processes can be denoted by the expression $g(x, y) = T[f(x, y)]$ where T is an operator on the input image $f(x, y)$ defined over a neighborhood of point (x, y). While spatial processing is applicable in the case of additive noise, degradation such as image blur should be approached by frequency domain filters. The frequency domain refers to the analysis of image signals with respect to frequency.

2 System Description

2.1 Histogram equalization

A histogram of an image is a function that maps each gray level of an image to the number of times it occurs. It can also be interpreted as an estimate of the probability density function (pdf) of an underlying random process.

A low-contrast image is an image with fewer tonal contrasts. It can be simulated by reducing the dynamic range of the image with this equation

$$g(x, y) = min(max(\lfloor a \cdot f(x, y) + b\rceil, 0), 255), \tag{1}$$

Histogram equalization is a method of contrast adjustment that could improve low-contrast images. Applying this transformation to each pixel of the input image (x, y) will make the entire range of gray levels in output image $g(x, y)$ more uniformly distributed. For discrete values the probability of occurrence of gray level r_k in a digital image is approximated by

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

where MN is the total number of pixels in the image, n_k is the number of pixels with gray level r_k and L is the number of possible gray levels in the image. The transformation is then done by

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

We get the output image $g(x, y)$ by replacing the old gray level r_k with the new level s_k in each pixel.

2.2 Image denoising

Image denoising is to remove the noise on an image while keeping its details. It can be done by filtering the image by low pass filters such as the mean filter and the median filter.

The mean filter is a linear filter that denoises the image by replacing each pixel value with the average value of its neighbor and itself. For example, to denoise an image by a 3*3 mean filter, function 5 will be used.

$$R = 1/9 \sum_{i=1}^{9} z_i \tag{4}$$

where z is the pixel value of one of its neighbors.

The median filter is a nonlinear filter that replaces each pixel value with the median of its neighbor and itself. For example, to denoise an image by a 3*3 median filter, the fifth most value of each pixel's neighbors, including itself, will be chosen to replace its value.

Since the mean filter uses the average value to replace each pixel value, a single unrepresentative pixel will affect its output value, however, the output of the median filter will not be affected by it.

2.3 Frequency domain filtering

To generate a blurred image, the function 5 will be used.

$$g(x,y) = h(x,y) * f(x,y) + \eta(x,y) \tag{5}$$

In equation 5, $h(x,y)$ is the Gaussian blur kernel, $f(x,y)$ is the input image, $\eta(x,y)$ is the noise. As the noise here is quantization noise, it will be introduced as a function 6. $g(x,y)$ is the output image.

$$g(x,y) = min(max(h(x,y) * f(x,y), 255), 0) \tag{6}$$

For the deblurring part, the wiener filter shown as function 7 will be used.

$$\hat{F}(u,v) = [\frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + K}]G(u,v) \tag{7}$$

This equation is from Section 5.8 of the textbook[1]. $G(u,v)$ is the Fourier Transform of the input image $g(x,y)$, $H(u,v)$ is the Fourier Transform of the blur function $h(x,y)$, K is the noise-to-signal ratio, but since we do not know the Fourier Transform of the noise, and the Fourier Transform of the image we are trying to recover, but we have the noise variance, so K will be the noise variance here. $\hat{F}(u,v)$ is the recovered image.

3 Results

3.1 Histogram equalization

The histogram of ***lena512.bmp*** is shown in Figure 2. We create a low-contract image using equation with a = 0.2 and b = 0.5. Comparing the histograms from both the original image and the low-contrast image, it is clear that a large volume of pixels are concentrated in the gray level range 50 to 100, a relatively narrow range of tones. Applying the histogram equalization algorithm to the contrast-reduced image gives a more spread out histogram in Figure 4. Unlike in the continuous case there is a finite number of values across the gray levels in discrete digital images. This will not make a flat out histogram after equalization as the pdf of the image is also a discrete estimate.

3.2 Image denoising

Figure 7 shows the original image of ***lena512.bmp*** without noise and its histogram.

We generated two kinds of noise masks and applied them to the original image respectively. The first one is Gaussian noise with zero mean and variance 64, and the second one is salt & pepper noise with $p(0) = p(225) = .05$. The noisy images are shown in Figure 8 and their histograms are shown in Figure 9.

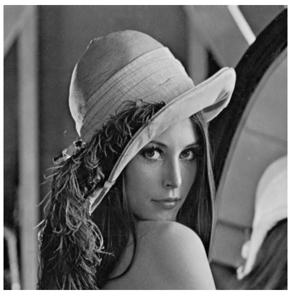

Figure 1: The ”Lena” image from ***lena512.bmp***

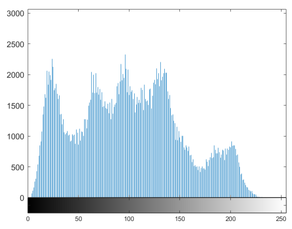

Figure 2: Histogram of ***lena512.bmp***

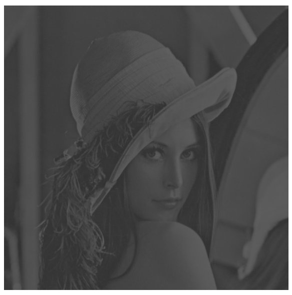

Figure 3: Low-contrast ***lena512.bmp***

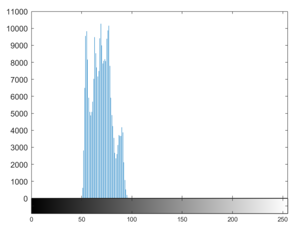

Figure 4: Histogram of low-contrast ***lena512.bmp***

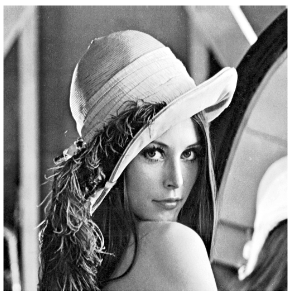

Figure 5: Histogram equalization of low-contrast ***lena512.bmp***

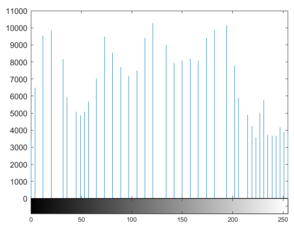

Figure 6: Histogram after histogram equalization ***lena512.bmp***

As shown in the figures, Gaussian noise makes the peaks on the images' histograms less sharp, meaning the images become blurred and some details are lost. After salt & pepper noise is applied, a lot of black and white dots appear on the image. In the histogram, the number of pixels having values of 0 and 255 increases significantly, and the number of pixels with values between [1,254] decreases.

We apply the 3*3 mean filter to the noisy images. The peaks of the histograms of the image with Gaussian noise reappeared, meaning that the details of the images are recovered. As for the image with salt & pepper noise, the number of pixels with values 0 and 255 decreased, meaning that the black and white noises are gone, however, the details failed to reappear, since the peaks disappeared.

Next, we applied the 3*3 median filter. The shapes of the histograms of both Gaussian noise image and salt & pepper noise after the median filter are similar to the original image's histogram.

In summary, the mean filter performs well on the Gaussian noise image but not on the salt & pepper noise image, and the median filter performs well on both. This is because the mean filter is vulnerable to unrepresentative pixel values, while the median filter is able to use representative values by choosing the median of neighboring pixels.

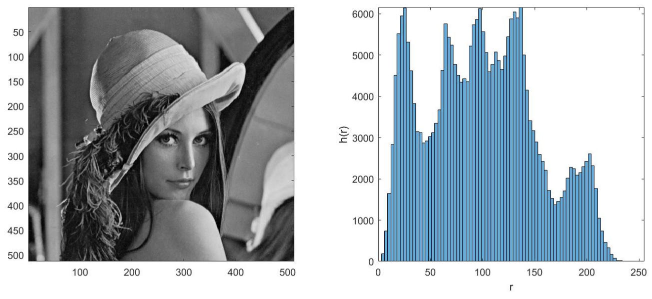

Figure 7: Original image and its histogram ***lena512.bmp***

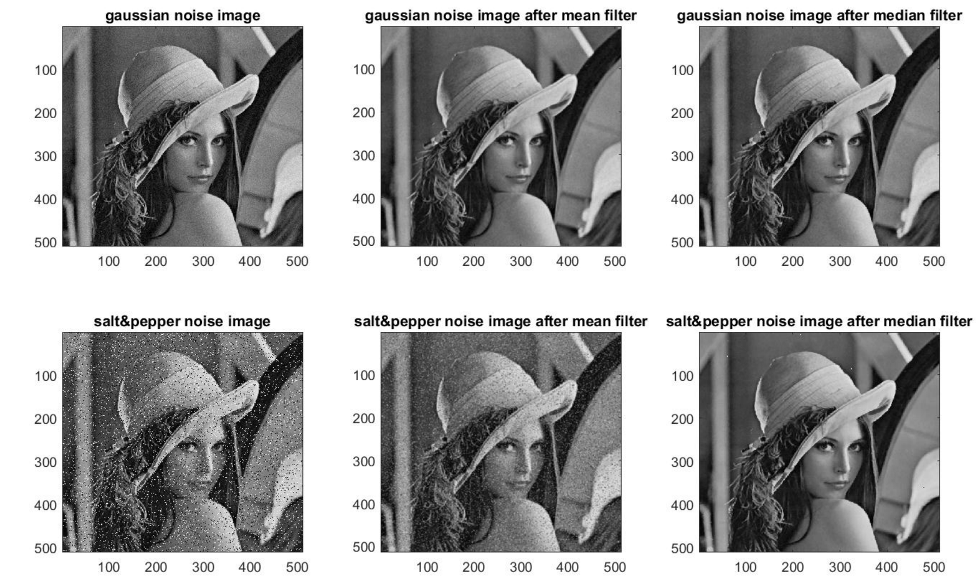
gaussian noise image
gaussian noise image after mean filter
gaussian noise image after median filter
salt&pepper noise image
salt&pepper noise image after mean filter
salt&pepper noise image after median filter

Figure 8: Images after denoising ***lena512.bmp***

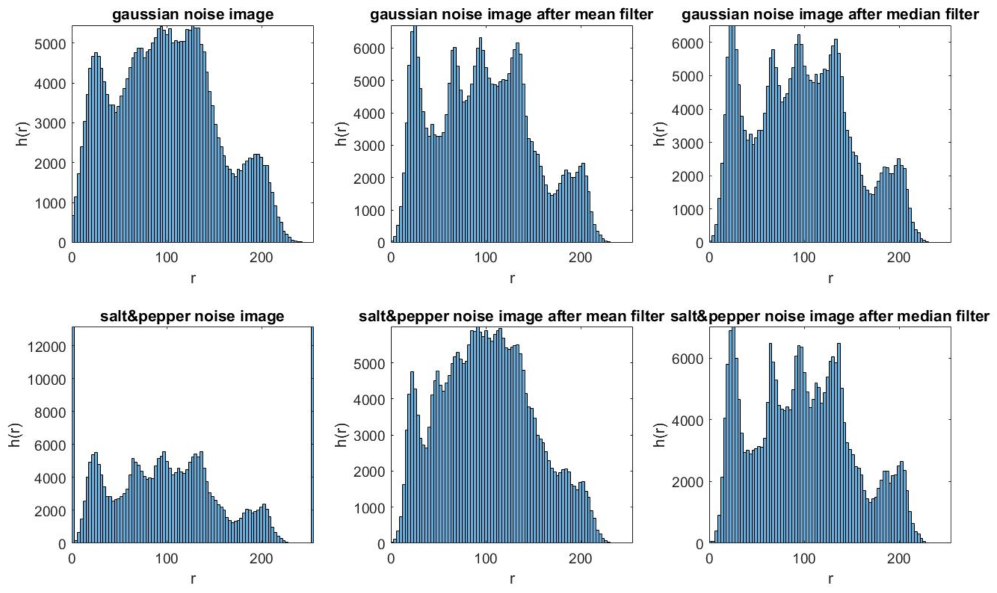

Figure 9: Histograms after denoising ***lena512.bmp***

3.3 Frequency domain filtering

We passed ***lena512.bmp*** to the function 6 to blur the image and add noise. Then we compare the Fourier spectra of the original image and the output image. We can see that most of the high-frequency signal is removed in Figure 11. This is because blurring is a type of low-pass filter, so the low to mid-range signal will be kept. Also, we can see some lines appear in the horizontal and vertical directions in Figure 11. The distribution of the lines is a Gaussian distribution, in which the lines are concentrated in the center, and if the lines are farther from the center, the lines will be less concentrated. This is because Gaussian blur is used to blur the image.

We used the function 7 to deblur the image. However, if we just apply the function 7 directly, one side of the image may wrap around to the other side, like in Figure 12. This is because multiplication in the frequency domain equals circular convolution in the spatial domain. Therefore, when the kernel is slid to the right edge of the image and does a convolution operation, it will wrap to the left side of the image. It means that in the output image, the right edge will be affected by the pixel on the left edge. The same principle will appear when the kernel is slid to the bottom of the image. As a result, we can see in Figure 12, some pixels in the left and top part of the image are shifted to the right and bottom part of the image. To solve this problem, we added padding, which reflects the edge pixel. The padding side equals the size of the blurring kernel, so when the kernel slides to the edge, it is only operating with the padding. After the inverse FFT, we cut the padding to extract the image we want. Figure 13 showed an example of an out-of-focus image, and figure 14 showed an example of deblurring an out-of-focus image with the wiener filter introduced as function 7. By comparing two images, we can see that after the deblurring process, the details of the image, which are the high-frequency parts, are restored.

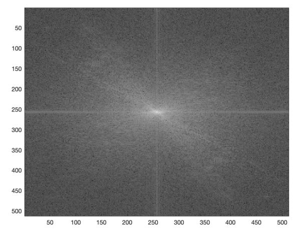
Figure 10: The Fourier spectra of ***lena512.bmp***

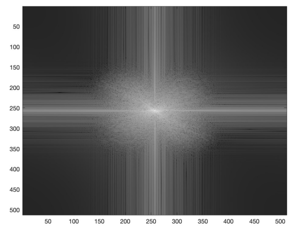
Figure 11: The Fourier spectra of ***lena512.bmp*** after applying function

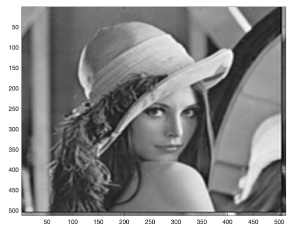
Figure 12: Example of deblurring without special treatment

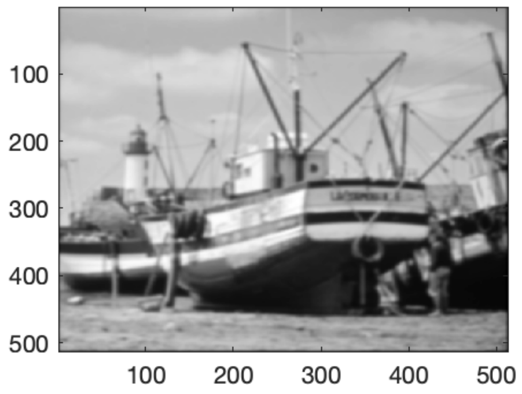

Figure 13: Example of out-of-focus image

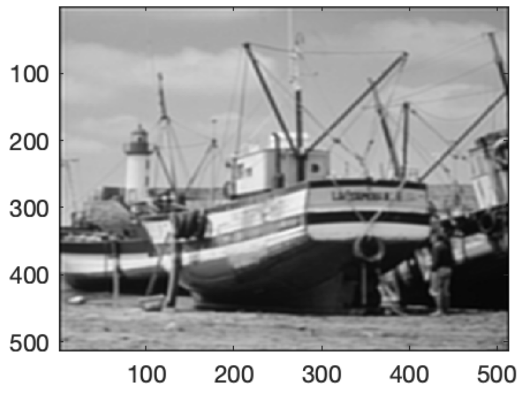

Figure 14: Example of deblurring an out of focus image with padding

4 Conclusions

To sum up, we used histogram equalization to enhance the contrast of the image. We used a mean filter and median filter to deal with Gaussian noise and salt & pepper noise. It turns out that the mean filter is good at handling Gaussian noise and the median filter is good at handling Gaussian noise and salt & pepper noise. We used Wiener filter to deblur an out-of-focus image. This can bring back some of the details of the image.

Appendix

Who Did What

Name	Part in-charge
TSANG, Chin Yung Virginia	Histogram equalization
CHIU, Lok Ying	Image denoising
LO, King Lam	Frequency domain filtering

t

MatLab code

4.0.1 q1.m

```
lena = imread('../images/lena512.bmp');

lenavec = lena(:); %image to vector
figure, histogram(lenavec); %plot histogram
imhist(lena); %display image
lenalowcont = lowcontrastfnct(0.2,50,lena); %function to generate a low contrast image
imagesc(lenalowcont, [0 255]);
figure,histogram(lenalowcont);
imshow(lenalowcont);
ylim([0,30000])

%histogram equalization without using histeq()

%Find the histogram of the image.
Val=reshape(lenalowcont,[],1);
Val=double(Val);
I = hist(Val,0:255);
%Divide the result by number of pixels
Output = I/numel(lenalowcont);
%Calculate the Cumlative sum
CSum=cumsum(Output);
%Perform the transformation S=T(R) where S and R in the range [0 1]
HIm=CSum(lenalowcont+1)*255;
imshow(HIm);
figure,histogram(HIm);
```

4.0.2 lowcontrastfnct.m

```
function [out] = lowcontrastfnct(a,b,image)
out = min(max( ...
    round(a*image+b) ...
    ,0),255);
end
```

4.0.3 q2.m

```
%apply noise
lena_o = imread('lena512.bmp');
g_noise = mynoisegen('gaussian', 512, 512, 0, 64);
g_lena = g_noise+double(lena_o); % lena with gaussian noise
saltp_lena = lena_o; %lena with saltpepper noise
saltp_n = mynoisegen('saltpepper', 512, 512, .05, .05);
saltp_lena(saltp_n==0) = 0;
saltp_lena(saltp_n==1) = 255;

%mean filter
mean_filter = [1/9 1/9 1/9; 1/9 1/9 1/9; 1/9 1/9 1/9];
mean_g = conv2(mean_filter,g_lena);
mean_saltp = conv2(mean_filter, saltp_lena);

%median filter
median_g = medfilt2(g_lena);
median_saltp = medfilt2(saltp_lena);

% Display original image and its histogram
figure(1);
```

```
subplot(1,2,1);
imagesc(lena_o,[0 255]);
subplot(1,2,2);
histogram(double(lena_o));
xlabel('r');
ylabel('h(r)');
axis([0 255 0 inf]);
colormap gray(256);

% Display histogram
figure(2);
subplot(2,3,1);
histogram(g_lena);
xlabel('r');
ylabel('h(r)');
title('gaussian noise image');
axis([0 255 0 inf]);
subplot(2,3,2);
histogram(mean_g);
xlabel('r');
ylabel('h(r)');
title('gaussian noise image after mean filter');
axis([0 255 0 inf]);
subplot(2,3,3);
histogram(median_g);
xlabel('r');
ylabel('h(r)');
title('gaussian noise image after median filter');
axis([0 255 0 inf]);
subplot(2,3,4);
histogram(double(saltp_lena));
xlabel('r');
ylabel('h(r)');
title('salt&pepper noise image');
axis([0 255 0 inf]);
subplot(2,3,5);
histogram(mean_saltp);
xlabel('r');
ylabel('h(r)');
title('salt&pepper noise image after mean filter');
axis([0 255 0 inf]);
subplot(2,3,6);
histogram(median_saltp);
xlabel('r');
ylabel('h(r)');
title('salt&pepper noise image after median filter');
axis([0 255 0 inf]);

% Display the 'Lena' image
figure(3);
subplot(2,3,1);
imagesc(g_lena,[0 255]);
title('gaussian noise image');
subplot(2,3,2);
imagesc(mean_g,[0 255]);
title('gaussian noise image after mean filter');
subplot(2,3,3);
imagesc(median_g,[0 255]);
title('gaussian noise image after median filter');
```

```
subplot(2,3,4);
imagesc(saltp_lena,[0 255]);
title('salt&pepper noise image');
subplot(2,3,5);
imagesc(mean_saltp,[0 255]);
title('salt&pepper noise image after mean filter');
subplot(2,3,6);
imagesc(median_saltp,[0 255]);
title('salt&pepper noise image after median filter');
colormap gray(256);
```

4.0.4 q3.m

```
f = imread("../images/lena512.bmp"); % load an image

r = 8; % Gaussian kernel radius from the instruction
h = myblurgen('gaussian', r); % use gaussian blur to blur the image
% make sure the output image size is the same as the input image size
g = conv2(f, h, 'same');
[M, N] = size(g);

% apply the quantization noise
for i = 1:M

    for j = 1:N
        g(i, j) = min([max([g(i, j), 0]), 255]);
    end

end

f = im2double(f);
g = im2double(g);

F = fft2(f);
F = fftshift(F); % center the spectra
F_spectra = log(abs(F)); % take the log value for visualization purpose

G = fft2(g);
G = fftshift(G);
G_spectra = log(abs(G));

% read a blurred image
g_blur = imread("../images/boats512_outoffocus.bmp");
g_blur = im2double(g_blur);
var = 0.0833; % noise variance from the instruction
f_hat = wiener_filter(g_blur, h, var);

% show the result
figure(4);
subplot(2, 2, 1);
imagesc((F_spectra));
subplot(2, 2, 2);
imagesc(G_spectra);
subplot(2, 2, 3);
imagesc(g_blur);
subplot(2, 2, 4);
imagesc(f_hat);
colormap gray(256);
```

4.0.5 wiener_filter.m

```
function f_hat = wiener_filter(g, h, K)
    [g_M, g_N] = size(g);
    [h_M, h_N] = size(h);
    % apply padding which reflect the edges pixels
    g_padded = padarray(g, size(h), 'symmetric');

    [M, N] = size(g_padded);
    % make the size of H same as G for calculation purpose
    H = fft2(h, M, N);
    G = fft2(g_padded);
    % equation of the wiener filter
    filter = (1 ./ H) .* ((abs(H).^2) ./ (abs(H).^2 + K));
    % apply wiener filter to the blurred image
    F_hat = G .* filter;
    f_hat = abs(ifft2(F_hat));
    %remove the padding
    f_hat = f_hat(ceil(double(h_M) / 2):ceil(double(h_M) / 2) + g_M - 1,
            ceil(double(h_N) / 2):ceil(double(h_N) / 2) + g_N - 1);
end
```

References

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