Xi'an Jiaotong-Liverpool University

西交利物浦大学

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE412 Image and Video Processing

Lab2 - Image Processing

Student Name Zhiyu.Xu

Student ID 1926905

September 2019

Contents

Task 1

1.1 Gaussian white noise with zero mean and variance 10

1.1.1 Result and Code

Figure 1 Result

```
%% ii Add Gaussion white nosie
im_wn = addGWN(im,10);% Add Gaussian white noise with zero mean and variance 10 to the image im;
figure;
subplot(1,2,1);imshow(im);title('Orignal Image');
subplot(1,2,2);imshow(uint8(im_wn));title('Image with Gaussian white noise');
truesize;
```

Code 1 main code

```
function distImg = addGWN(origImg,p)
%UNTITLED Adding white Gaussian(mean=0,variance=p) noise to the original
 %image;
 % function WGN (m, n, p) produces a matrix of Gaussian white noise in M rows and N columns;
 % P specifies the intensity of the output noise in W units(variance);
 % code need change unit, Formula is P(dB)=10lgP(W);
 distImg = double(origImg) + wgn(size(origImg,1),size(origImg,2),10*log10(p*p));
 end
```

Code 2 add Gaussian White Noise function

1.1.2 Explanation

The formula adds noise to the image by adding a Gaussian white noise matrix of the same size to the original image matrix. As can be seen from Figure 1, there are some randomly distributed white pixels in the image with Gaussian white noise on the right side compared to the original image on the left.

1.2 Add salt & pepper noise with noise density 10%

1.2.1 Result and Code

Figure 2 Result

```
%% iii Add salt & pepper noise with noise density 10%
im_SP = addSPN(im,0.1);
figure;
subplot(1,2,1);imshow(im);title('Orignal Image');
subplot(1,2,2);imshow(uint8(im_SP));title('Image with salt & pepper noise');
truesize;
```

Code 3 main code

```
function distImg = addSPN(origImg,p)
%UNTITLED Adding salt & pepper noise with noise density p(%) to the
%original image;
%salt & pepper noise: The amplitude of noise is basically the same, but the
%location is random;
[width,height] = size(origImg);
distImg = origImg;
k1=p;              %noise density;
k2=p;              %noise density;
%Rand(M,N) is a matrix that randomly generates M rows and N columns;Each
%matrix element is between 0 and 1;
%So the element less than k is 1 in the matrix, and vice versa is 0;
a1=rand(width,height)<k1;
a2=rand(width,height)<k2;
distImg(a1&a2)=0;          %set black points;
distImg(a1& ~a2)=255;      %set whitre points;
end
```

Code 4 Add salt & pepper noise function

1.2.2 Explanation

The formula adds salt & pepper noise by randomly adding white or black to 10% of the pixels in the image.

1.3 PSNR of the previous images

1.3.1 Result and code

Image	im_wn	im_sp	im_low_dynamic_range
PSNR(dB)	28.1239	15.2846	23.9912

```
%% iv Evaluate the PSNR
im_low_dynamic_range = imread('lenna512_low_dynamic_range.bmp');
PSNR_wn = PSNR1(im,im_wn);
PSNR_SP = PSNR1(im,im_SP);
PSNR_low = PSNR1(im,im_low_dynamic_range);
```

Code 5

1.3.2 Explanation

The PSNR1 function in Code 5 comes from lab1. PSNR is the abbreviation of "Peak Signal to Noise Ratio", which is an objective standard for evaluating images. A larger PSNR value indicates that the image is closer to the reference image, or the image quality is higher. PSNR1 function code is shown as below.

```
% Task2 PSNR function
function [PSNR] = PSNR1(im_1,im_2)
%im_1 = imread('iamgename1.***');
%im_2 = imread('iamgename2.***');
im1 = double(im_1);
im2 = double(im_2);
[r1,c1] = size(im1);
[r2,c2] = size(im2);
t = 0;
if (r1 == r2 && c1 == c2)
    for i=1:r1
        for j=1:c2
            t=t+(im1(i,j)-im2(i,j))^2;
        end
    end
end
mse = t/(r1 * c1);
PSNR = 10*log10(255^2/mse);
end
```

Code 6 PSNR1 function

1.4 Histograms

1.4.1 Result and code

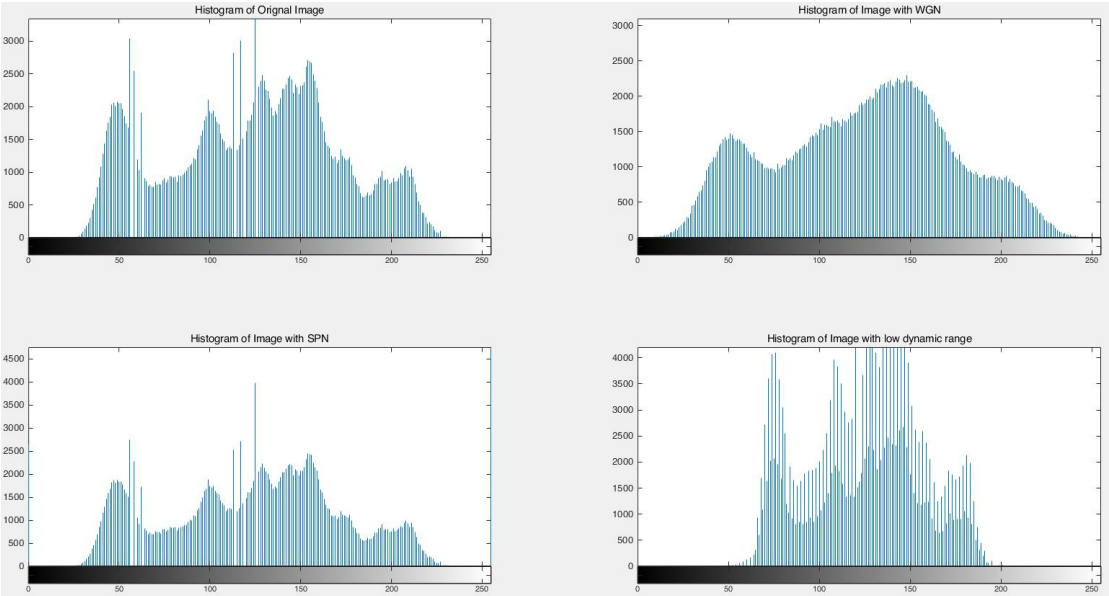

Figure 3 Histograms

```
%% v Histogram of all the previous images
figure
subplot(2,2,1);imhist(im);title('Histogram of Orignal Image');
subplot(2,2,2);imhist(uint8(im_wn));title('Histogram of Image with WGN');
subplot(2,2,3);imhist(uint8(im_SP));title('Histogram of Image with SPN');
subplot(2,2,4);imhist(im_low_dynamic_range);title('Histogram of Image with low dynamic range');
```

Code 7

1.4.2 Comment

In the histogram, the abscissa represents a gray value of 0 to 255, and the ordinate represents the number of pixel points. Histogram of image with GWN(Gaussian withe noise) is smoother compared to the histogram of original image. This is because that GWN is a random signal with a mean of 0 and a variance of 10. Adding it to the original image will make the histogram appear smoother. Histogram of image with SPN(salt & pepper noise) and histogram of original image have similar envelopes, but the latter has fewer overall pixels because 10% of the pixels are converted to gray values of 0 or 255. The last picture is a low dynamic range, so its histogram leaves only pixels with a gray value of 50 to 200 on the display.

1.5 Image Averaging

1.5.1 Result and code

Image	im_wn10	im_wn100	im_wn1000
PSNR(dB)	42.1693	32.8094	22.8981

```
%% vi remove noise by the technique of Image Averaging
im_wn10 = Average(im,10);% average 10 images
im_wn100 = Average(im,100);% average 100 images
im_wn1000 = Average(im,1000);% average 1000 images
PSNR_wn10 = PSNR1(im,im_wn10);
PSNR_wn100 = PSNR1(im,im_wn100);
PSNR_wn1000 = PSNR1(im,im_wn1000);
```

Code 8 main code

```
function [im_ave] = Average(im,n)
%Image with Gaussian white noise added, superimposed n times and averaged;
[r,c] = size(im);
im_ave = double(zeros(r,c));
for n=1:n
    im_noise=addGWN(im,n);%add Gaussion white nosie
    im_ave=im_ave+double(im_noise); %sum up
end
im_ave = im_ave/n;%averaged
im_ave = uint8(im_ave);
end
```

Code 9 Average function

1.5.2 Comment

Comparing the PSNR of the images after 10 times, 100 times, and 1000 times of average processing, we can find that the more the average times, the smaller the PSNR, indicating that the image quality is lower. Although image noise can be removed after multiple average processing, too much averaging can also distort the image.

Task 2 Image enhancement

2.1 Piece-wise linear mapping transform

2.1.1 Result and code

Figure 4 Piece-wise linear mapping transform

```
%% (1)enhance the contrast of im_low_dynamic_range by piece-wise linear mapping transform
im_low_dynamic_range = imread('lenna512_low_dynamic_range.bmp');
im_enhanced1 = PWLMT(im_low_dynamic_range,20,220,50,200);
im_enhanced2 = PWLMT(im_low_dynamic_range,10,230,50,200);
im_enhanced3 = PWLMT(im_low_dynamic_range,5,250,50,200);
im_enhanced4 = PWLMT(im_low_dynamic_range,10,230,40,210);
im_enhanced5 = PWLMT(im_low_dynamic_range,40,190,60,190);
figure(1)
subplot(2,3,1);imshow(im_low_dynamic_range);title('Image with low dynamic range');
subplot(2,3,2);imshow(im_enhanced1);title('Enhanced Image by piece-wise linear mapping transform 1');
subplot(2,3,3);imshow(im_enhanced2);title('Enhanced Image by piece-wise linear mapping transform 2');
subplot(2,3,4);imshow(im_enhanced3);title('Enhanced Image by piece-wise linear mapping transform 3');
subplot(2,3,5);imshow(im_enhanced4);title('Enhanced Image by piece-wise linear mapping transform 4');
subplot(2,3,6);imshow(im_enhanced5);title('Enhanced Image by piece-wise linear mapping transform 5');
truesize;
% calculate PSNR to the reference image
PSNR_E1 = PSNR1(im,im_enhanced1);
PSNR_E2 = PSNR1(im,im_enhanced2);
PSNR_E3 = PSNR1(im,im_enhanced3);
PSNR_E4 = PSNR1(im,im_enhanced4);
PSNR_E5 = PSNR1(im,im_enhanced5);
```

Code 10 main code

```
function [im_out] = PWLMT(im,a,b,m,n)
%PWLMT is the abbreviation of piece-wise linear mapping transform;
%   The formula changes the gray value of an image by a piecewise linear
%   function;
[r,c] = size(im);
im_out = double(zeros(r,c));
im = double(im);
for i=1:r
    for j=1:c
        if  im(i,j) < m
            im_out(i,j)=round(a/m*im(i,j));
        else if  im(i,j) < n
                im_out(i,j)=round((b-a)/(n-m)*im(i,j)+a-((b-a)/(n-m))*m);
            else im(i,j) < 256;
                im_out(i,j)=round((255-b)/(255-n)*im(i,j)+b-((255-b)/(255-n))*n);
             end
        end
    end
end
im_out = uint8(im_out);
end
```

Code 11 Function

3.1.2 Discussion

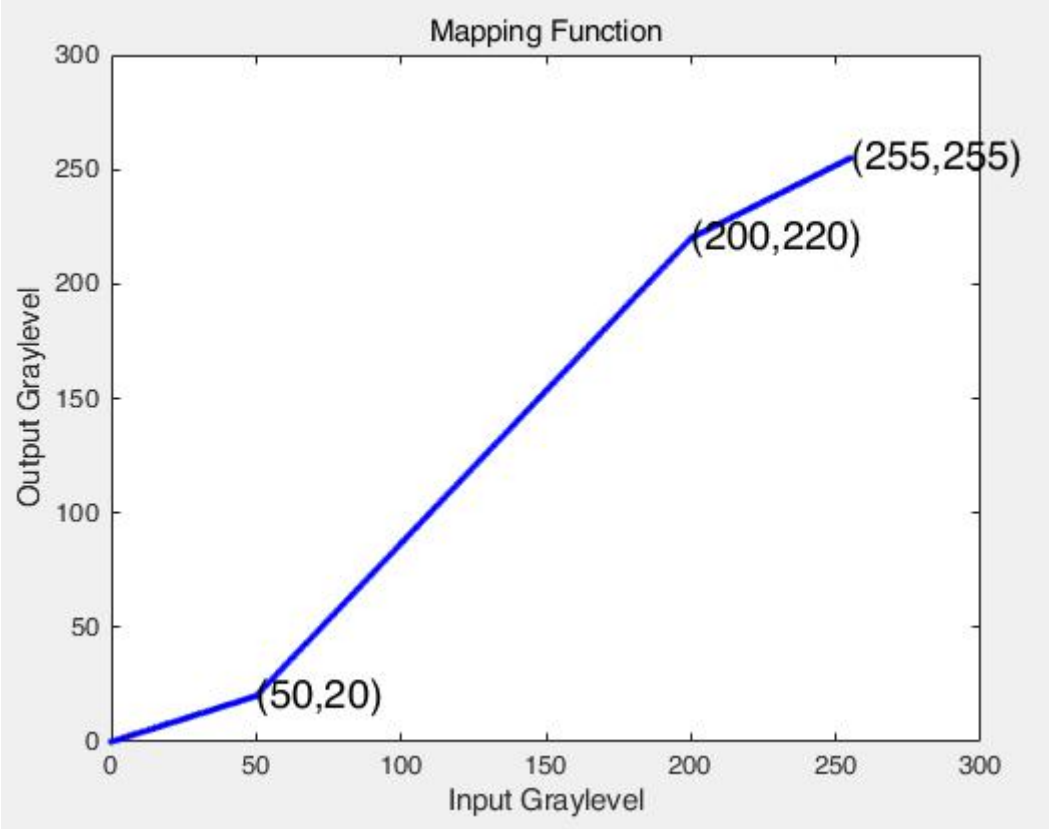

Figure 5 The general piece-wise linear function

As shown in Figure 5, this is a general piece-wise linear function Because in the histogram of Figure 3, we can find that the gray value of the low dynamic range map is roughly distributed between 50 and 200, so we also choose 50 and 200 for the

inflection point of this function. But for the rigor of the experiment, we choose four sets of inflection points near these two points, namely (50, 10) and (200, 230), (50, 5) and (200, 250), (40, 10) and (210, 230), (60, 40) and (190, 190). The final image results are shown in Figure 4. By visual inspection, Enhanced Image by piece-wise linear mapping transform2 is closest to the original image. Because its PSNR value is also the largest of the five enhanced images. PSNR values are shown in the table below.

Image	im_enhanced1	im_enhanced2	im_enhanced3	im_enhanced4	im_enhanced5	im_enhanced6
PSNR(dB)	31.3067	35.1455	33.7812	30.1155	24.9116	19.0767

2.2 Command histeq

2.2.1 Result and code

Figure 6 Enhanced by “histeq”command

```
%% (2)Use the command histeq
im_enhanced6 = histeq(im_low_dynamic_range);
figure(2)
subplot(1,2,1);imshow(im_low_dynamic_range);title('Image with low dynamic range');
subplot(1,2,2);imshow(im_enhanced6);title('Enhanced Image by "histeq" function');
truesize;
% calculate PSNR to the reference image
PSNR_E6 = PSNR1(im,im_enhanced6);
```

Code 12

2.2.2 Comment

Compare the current result with the best intensity mapping function in 2.1, we can clearly see that the contrast of the current results is significantly higher. This is because the function enhances the area of low gray value and high gray value, and the intermediate gray area becomes sparse, thereby improving the contrast of the image.

Task 3 Edge detection

3.1 Edge detection

Edge detection in digital images can identify locations where the brightness of the image changes dramatically. The Sobel operator (including horizontal and vertical operators) convolves the image for two-dimensional edge detection. In a convolution, the value of the center point will be the weighted sum of the neighbors and peak at the edge, enabling edge detection. The following are the horizontal detection operators and vertical detection operators used in the code.

1	2	1
0	0	0
-1	-2	-1

Table 1 horizontal detection operators

-1	0	1
-2	0	2
-1	0	1

Table 2 vertical detection operators

3.1.1 Result and code

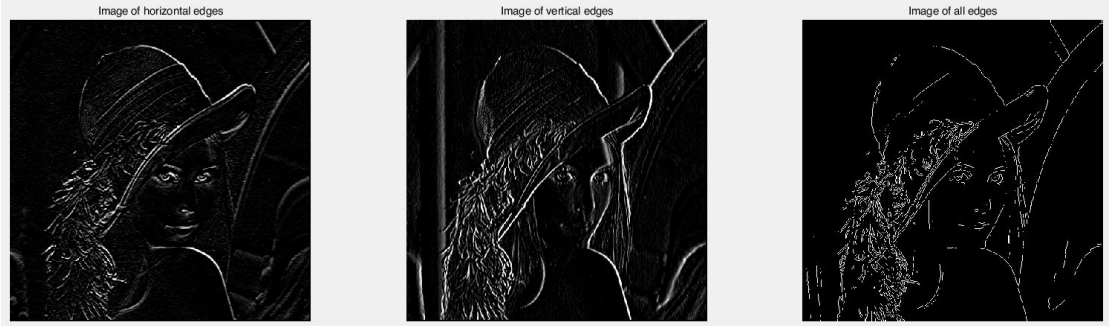

Figure 7

```
%Task3 show the horizontal edges, vertical edges and all edges of the image im
im = imread('lenna512.bmp');
im = double(im);
[r,c] = size(im);
im_horizontal_edges = zeros(r,c);
im_vertical_edges = zeros(r,c);
for i = 1:r-2
    for j = 1:c-2
        im_horizontal_edges(i,j)=(im(i+2,j)+2*im(i+2,j+1)+im(i+2,j+2))-(im(i,j)+2*im(i,j+1)+im(i,j+2));
        im_vertical_edges(i,j)=(im(i,j)+2*im(i+1,j)+im(i+2,j))-(im(i,j+2)+2*im(i+1,j+2)+im(i+2,j+2));
    end
end
im_all_edges = edge(im,'sobel');%Sobel edge detection operator
subplot(1,3,1);imshow(uint8(im_horizontal_edges));title('Image of horizontal edges');
subplot(1,3,2);imshow(uint8(im_vertical_edges));title('Image of vertical edges');
subplot(1,3,3);imshow(im_all_edges);title('Image of all edges');
truesize;
```

Code 13

Task 4 Filter

4.1 Median filter by 3X3 and 5X5

4.1.1 Result and code

Figure 8 Median filter by 3X3 and 5X5

```
%% Median filter by 3X3 and 5X5
im_MF_3X3 = medfilt2(im_SP,[3 3]);
im_MF_5X5 = medfilt2(im_SP,[5 5]);
PSNR_MF_3X3 = PSNR1(im,im_MF_3X3);
PSNR_MF_5X5 = PSNR1(im,im_MF_5X5);
figure(1)
subplot(2,2,1);imshow(im);title('Original Image');
subplot(2,2,2);imshow(uint8(im_SP));title('Image with salt & pepper noise');
subplot(2,2,3);imshow(im_MF_3X3);title('Image of 3X3 Median filter');
subplot(2,2,4);imshow(im_MF_5X5);title('Image of 5X5 Median filter');
truesize;
```

Code 14

Median Filter	3X3	5X5
PSNR(dB)	32.2886	30.6066

Table 3 PSNR

4.1.2 Comment

Median filtering is a kind of nonlinear filtering. Its principle is to sort the gray scales of all pixels in the small window centered on the current pixel from small to large, and take the middle value of the sorting result as the gray value of the pixel. Seen on figure 8, the salt & pepper noise is reduced, and it seems like the filter with 3x3 window performs better than 5x5 window. Seen on PSNR, the data also illustrates that the filter with 3x3 window performs better.

For the 3X3 filter, a 3X3 window is created centering on the pixel to be processed, the gray values of the nine pixels are sorted, and the intermediate value is selected. For the 3X3 filter, create a 5X5 window. Because the median filter is a nonlinear filter, it can directly reject the noise signal without affecting the sharp edges.

4.2 Average filter 3X3

4.2.1 Result and code

Figure 9

```
%% Average filter by 3X3
h = fspecial('average',[3,3]);%create an averaging filter h of 3X3 size;
im_AF_3X3 = imfilter(im_SP,h);
PSNR_AF_3X3 = PSNR1(im,im_AF_3X3);
figure(2)
subplot(1,3,1);imshow(im);title('Original Image');
subplot(1,3,2);imshow(uint8(im_SP));title('Image with salt & pepper noise');
subplot(1,3,3);imshow(im_AF_3X3);title('3X3 Average filter Image');
truesize;
```

Code 15

Average filter	3X3
PSNR(dB)	22.4253

Table 4 PSNR

4.2.2 Comment

By comparing the results of Figures 9 and 8, we can see that the median filter works better for removing salt & pepper noise. However, this does not mean that the

median filter must be better than the averaging filter. This is because the averaging filter is a linear filter. It is impossible to directly remove noise and only blur the noise and the original image. It can only be explained that the median filter is more suitable for removing salt & pepper noise.