Xi'an Jiaotong-Liverpool University
西交利物浦大学

EEE330 - Image Processing
Assessment 2

Student Name: Wang Xiaoyang
Student ID: 1405888

April 4, 2018

Contents

1 PSNR Function Implementation

This section is to implement a function to calculate peak signal noise ratio (PSNR) between two images. The steps for calculation are shown in Equation 1 and 2 . The fist step is to obtain the mean square error between images where N and M refers to row and column numbers. Then, do the $10log_{10}()$ with square of 255 to get the final results. Basic logic is shown in Algorithm 1.

$$MSE = \frac{1}{NM}\sum_{r_i=0}^{N}\sum_{c_i=0}^{M}(im_1(r_i, c_i) - im_2(r_i, c_i))^2 \tag{1}$$

$$PSRN = 10log_{10}(\frac{255^2}{MSE}) \tag{2}$$

Algorithm 1 PSNR Calculation

Input: Two images im_1 and im_2

Output: Calculation result of PSRN.

1: Obtain input image size and store them in $[dim1, dim2]$;
2: Do element-wise subtraction between two image and sum up the results;
3: Square the sum and divide the result by $dim1 \times dim2$ to get MSE;
4: Divide 255^2 by MSE and do $10log_{10}$ and finally obtain $PSRN$;
5: **return** $PSRN$;

2 Image Noise and Dynamic Range

In this section, the reference image shown in Figure 1 (a) is added by two types of noises which are *Salt-pepper Noise* and *Gaussian White Noise*. Function *imnoise* is called to create noise on image with required parameters input. Note that for Gaussian white noise, the input variance is normalized which means it corresponds to images with intensities ranging from 0 to 1. Then, the variance of 10 should be input as $10^2/255^2$.

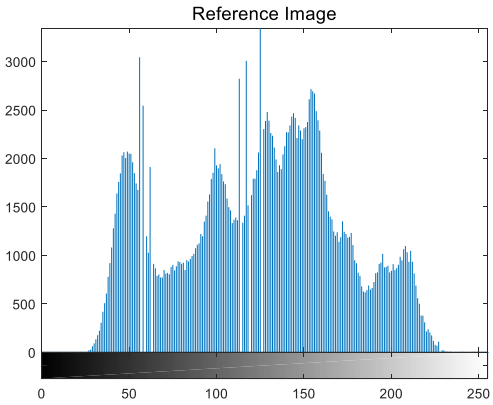
The comparisons of images are illustrated in Figure 2. It is intuitive to observe that picture with Gaussian white noise is very close to the reference but with slight noisy points uniformly distributed. In Figure 2 (b), salt-pepper noise add more obvious dark and bright pixels which seems like salt and pepper on the image. They are generated by sharp and sudden disturbance in image signals. Figure 2 (c) shows an image with low dynamic range where the contrast is quite low. The color difference is smaller between any two locations than those in reference, making it less recognizable in some edges such as feathers on Lenna's hat.

Next step is to check the histograms of the images discussed before. Histogram is a direct illustration of pixel numbers of each color level. By observing the distribution of pixels, we can get important information such as boundaries of color level and the range contains most of the pixels, which are useful for further processing. In Figure 3, histograms from (a) and (b) has the similar boundary while those in (c) are located more close to the middle level leading to a narrower range. As a consequence, the contrast is low as shown in Figure 2 (c). The histogram of image with salt-pepper has the similar shape to reference but (a) seems smoother.

Table 1: Comparison on PSNR Results

Image	im_WN	im_SP	im_low_dynamic_range
PSNR(dB)	32.00	37.03	29.46

(a) Reference Image



(b) Histogram of reference Image

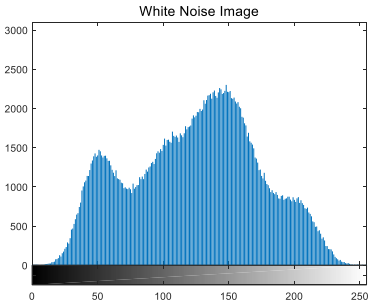
Figure 1: Reference Image and its histogram

(a) Image with Gaussian white noise

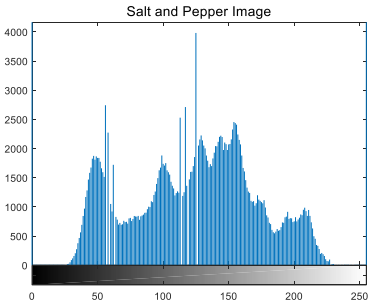
(b) Image with salt-pepper noise

(c) Image with low-dynamic range

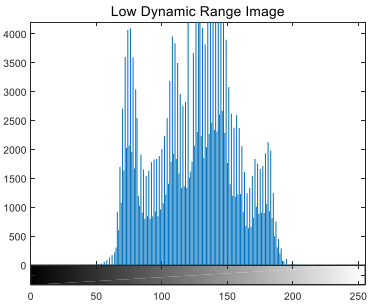
Figure 2: Comparison of noises and dynamic range on image



(a)



(b)



(c)

Figure 3: Comparison of noises and dynamic range on image histogram

3 Contrast Enhancement

3.1 Piece-wise Linear Function Mapping

In this section, a piece-wise linear function is implemented to enhance the image with low dynamic range. This method is straight forward by applying linear transforming to stretch and compress regions of image. The implementation logic is illustrated in Algorithm 2 where the detailed calculation process to obtain the linear functions is explained. First step it to obtain slopes in stretching and compressing regions according to input coordinates of boundary points. Then, we can find the exact linear functions using slope and boundary informations.

As shown in Figure 4, three different functions are tested with their boundary moving towards the middle. The detailed slope informations are recoded in Table 2. In Figure 5, histograms in (a) and (b) seems reasonable because it is obvious that the dynamic range is expanded to two sides while keeping the general shape of distribution. However, histogram that generated by function C is quite strange for there is a huge loss in the pixels of middle color levels. This can also be noticed in the related image in Figure 6 (c) where we can see great loss in image information. Results of function A and B seems quite good by intuitive observation but the one performs best is function B because its result has the highest PSNR as 37.46 in Table 2.

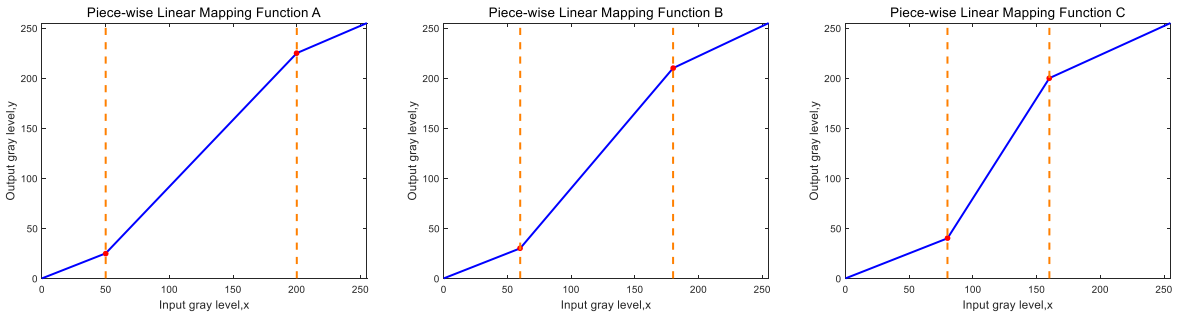
Algorithm 2 Algorithm for piece-wise linear mapping

Input: im_org: Original image; $codnt1$: Lower boundary point location (X_1, Y_1); $codnt2$: Upper boundary point location (X_2, Y_2);
Output: im_eh: The enhanced image;

$k_1 = (Y_1 - 0)/(X_1 - 0)$;
$k_2 = (Y_2 - Y_1)/(X_2 - X_1)$;
$k_3 = (255 - Y_2)/(255 - X_2)$;
$b_1 = 0$;
$b_2 = Y_1 - k_2 * X_1$;
$b_3 = Y_2 - k_3 * X_2$;
$fun_1(x) = k_1 * x + b_1$;
$fun_2(x) = k_2 * x + b_2$;
$fun_3(x) = k_3 * x + b_3$;
for each $pixel \in im\_org$ **do**
    if $value < X_1$ **then**
        $value \leftarrow fun_1(value)$
    else if $X_1 <= value <= X_2$ **then**
        $value \leftarrow fun_2(value)$
    else
        $value \leftarrow fun_3(value)$
    end if
end for

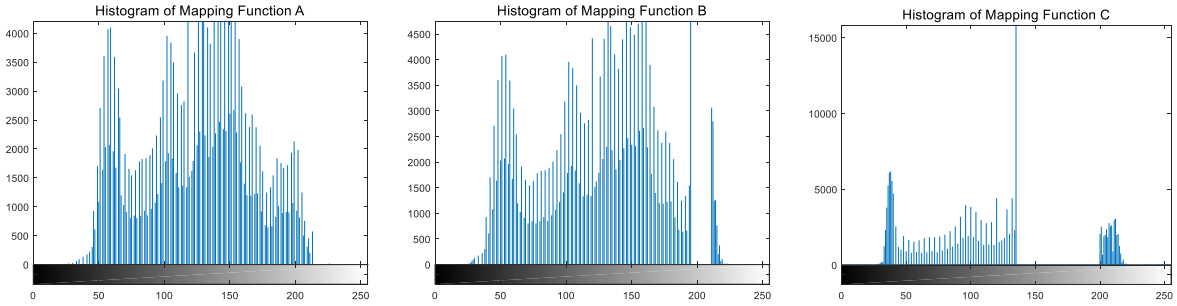
Table 2: Comparison on PSNR Results

Case	BP(X_1, Y_1)	BP(X_2, Y_2)	K_1	K_2	K_3	$PSNR(dB)$
A	$(50, 25)$	$(200, 225)$	0.5	1.33	0.55	35.57
B	$(60, 30)$	$(180, 210)$	0.5	1.5	0.6	37.46
C	$(80, 40)$	$(160, 200)$	0.5	2	0.58	35.38



(a) (b) (c)

Figure 4: Piece-wise Linear mapping functions



(a) (b) (c)

Figure 5: Histograms of piece-wise linear functions

(a) Image in case A (b) Image in case B (c) Image in case C

Figure 6: Enhanced Images

3.2 Histogram Equalization

In this section, histogram equalization method is applied to enhance image contrast. The mathematical expression of the algorithm is shown in Eq 3 where s_k is calculated level value, L is total number of color levels and MN is total pixel numbers of the image. The new level value is based on the sum of pixel numbers equal and less than that level. The basic logic is shown in Algorithm 3. In Figure 7, the range boundary are stretched much wider compared with those

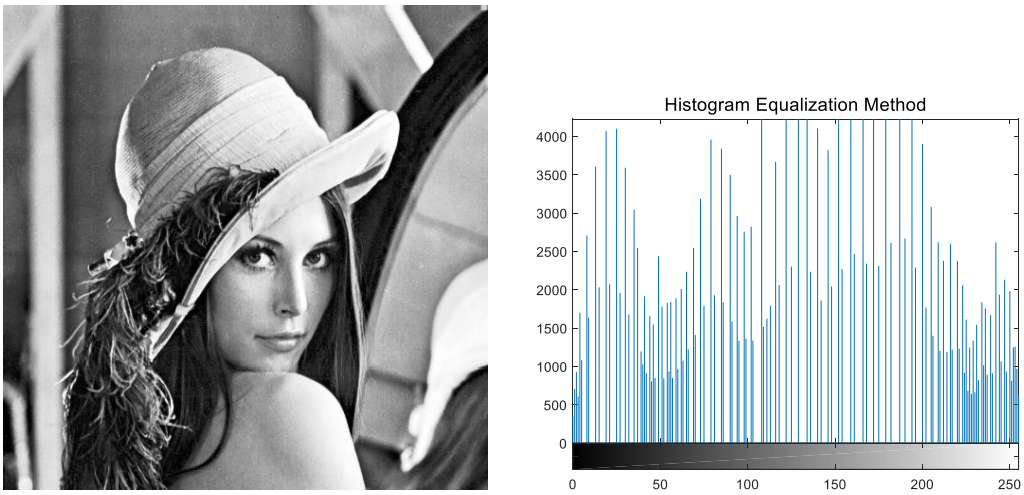
in Figure 5 and the enhanced image has a higher contrast.

$$s_k = T(r_k) = \frac{(L-1)}{(MN)} \sum_{j=0}^{k} n_j \quad (k = 0, 1, 2, ...L-1) \tag{3}$$

Algorithm 3 Algorithm for histogram equalization method

Input: im_org: Original image;
Output: im_eh: The enhanced image;

```
[counts, levels] = imhist(im_org);
for each level ∈ levels do
    if level = 0 then
        accum_pixel_num(level) ← counts(level);
    else
        accum_pixel_num(level) ← counts(level) + accum_pixel_num(level − 1);
    end if
    // accum_pixel_num(level) refers to accumulated pixel numbers under this color lever;
    Calculate new level value new_level using accum_pixel_num(level);
    Replace all pixels of level in original image with new_level;
end for
```



(a) Enhanced Image (b) Histogram of Enhanced Image

Figure 7: Enhanced Image and its histogram

4 Filters on Image

This section is to use two types of filter, median and average, to alleviate the effect of salt-pepper noise on image. The experiment results are illustrated in Figure 8 and Table 3. Then, we can check both from images and calculated data that 3×3 median filter performs the best to handle salt-pepper noise. This kind of noise is caused by impulse disturbance so the noise pixels are usually white and black, which are located at the boundary of color level graph. Therefore, the median filter can easily by pass the noisy point because it mainly focus on median value. By contrast, the average filter would take all pixels into account to do average operation, which cannot avoid effect of white and dark points. This is why median filter outperforms average one.

(a) 3×3 median filter (b) 5×5 median filter (c) 3×3 average filter

Figure 8: Filtered Image

Table 3: Comparison on PSNR Results

Type	size	$PSNR(dB)$
Median	3×3	34.71
Median	5×5	31.04
Average	3×3	27.05

Appendices

A PSNR Function

```
function psnr = psnr_cal(im1,im2)
% This function takes two images im1 and im2
% and calculate PSNR between them
[dim1,dim2] = size(im1);
MSE = sum(sum((im1 - im2).^2))/(dim1*dim2);
psnr = 10*log10(255^2/MSE);
end
```

B Script for Image Noise and Dynamic Range

```
%% Add noise
im = imread('lenna512.bmp');
im_wn = imnoise(im,'gaussian',0,10^2/255^2);
im_sp = imnoise(im,'salt & pepper',0.1);
im_low_dynamic_range = imread('lenna512_low_dynamic_range.bmp');
%% Calcualte PSNR
psnr_wn = psnr_cal(im_wn,im);
psnr_sp = psnr_cal(im_sp,im);
psnr_low_dynamic_range = psnr_cal(im_low_dynamic_range,im);
%% Show histogram
figure;imhist(im); title('Reference Image','fontsize',14)
```

```
figure;imhist(im_wn); title('White Noise Image','fontsize',14)
figure;imhist(im_sp); title('Salt and Pepper Image','fontsize'
   ,14)
figure;imhist(im_low_dynamic_range); title('Low Dynamic Range
   Image','fontsize',14)
```

C Piece-wise Linear Transform Function

```

function [eh_im,k1,k2,k3,b2,b3] = plf_eh(org_im,codnt1,codnt2)
start_point = [0,0];
end_point = [255,255];
x1 = start_point(1); y1 = start_point(2);
x2 = codnt1(1); y2 = codnt1(2);
x3 = codnt2(1); y3 = codnt2(2);
x4 = end_point(1);y4 = end_point(2);

k1 = (y2-y1)/(x2-x1);
k2 = (y3-y2)/(x3-x2);
k3 = (y4-y3)/(x4-x3);

b1 = 0;
b2 = y2-k2*x2;
b3 = y3-k3*x3;
[dim1,dim2] = size(org_im);
eh_im = zeros([dim1,dim2]);
for i = 1:dim1
    for j = 1: dim2
        if org_im(i,j)<x2
            eh_im(i,j) = k1*org_im(i,j);
        elseif (org_im(i,j)>=x2&&org_im(i,j)<=x3)
            eh_im(i,j) = k2*org_im(i,j)+b2;
        else
            eh_im(i,j) = k3*org_im(i,j)+b3;
        end
    end
end
eh_im = uint8(eh_im);
end
```

D Histogram Equalization Function

```
function eh_im = histeq_eh(org_im)
% This function takes an input image and do contrast by the
   method of histogram equalization

[counts,bins] = imhist(org_im);
[dim1,dim2] = size(org_im);
```

```
6
7  num_lv = length(bins); % Total number of color levels
8  accum_lv = zeros(length(counts)); % A matrix to store accumulated
       pixel numbers
9  eh_im = zeros([dim1,dim2]); % A matrix to store new image
10 pr_factor = (max(num_lv)-1)/(dim1*dim2); % Calcualte (L-1)/(MN)
11
12 for i=1:num_lv
13     if i == 1
14         accum_lv(i) = counts(i);
15     else
16         accum_lv(i) = counts(i) + accum_lv(i-1); % Accumulated
               pixle number
17     end
18     new_lv = pr_factor*accum_lv(i); % Calculate new level value
19     idx = org_im==bins(i); % Obtain coordinates of pixles of
           current level
20     eh_im(idx)=new_lv; % Replace original level by new value
21 end
22 eh_im = uint8(eh_im);
23 end
```