INTRODUCTION

This experiment guides students through the functions of MATLAB regarding image recognition. It helps students to build an intuitive understanding of image histogram, predictive coding, and morphological operations, leading to further study on image processing technics.

Task1

**(1)**

**An image histogram is a histogram that illustrate the distribution of lightness (luminance) in an image by representing the pixel values for an image at each tonal level.**

**For example, for an sRGB image, we can think of it as a matrix of H\*W\*1, with each element of the matrix taking values in the range [0,255], and the image histogram h(g) defined as the number of elements whose pixel value is g. Similarly, in color images, the image histogram needs to be calculated for each color channel.**

**The histogram is displayed as: from left to right, in increasing order of lightness.** **The horizontal axis of the graph represents the tonal variations, while the vertical axis represents the total number of pixels in that tone.**

**Use the build-in imhist() function to obtain the histogram of im\_ldr, the histogram with greyscale level =256 is presented below:**

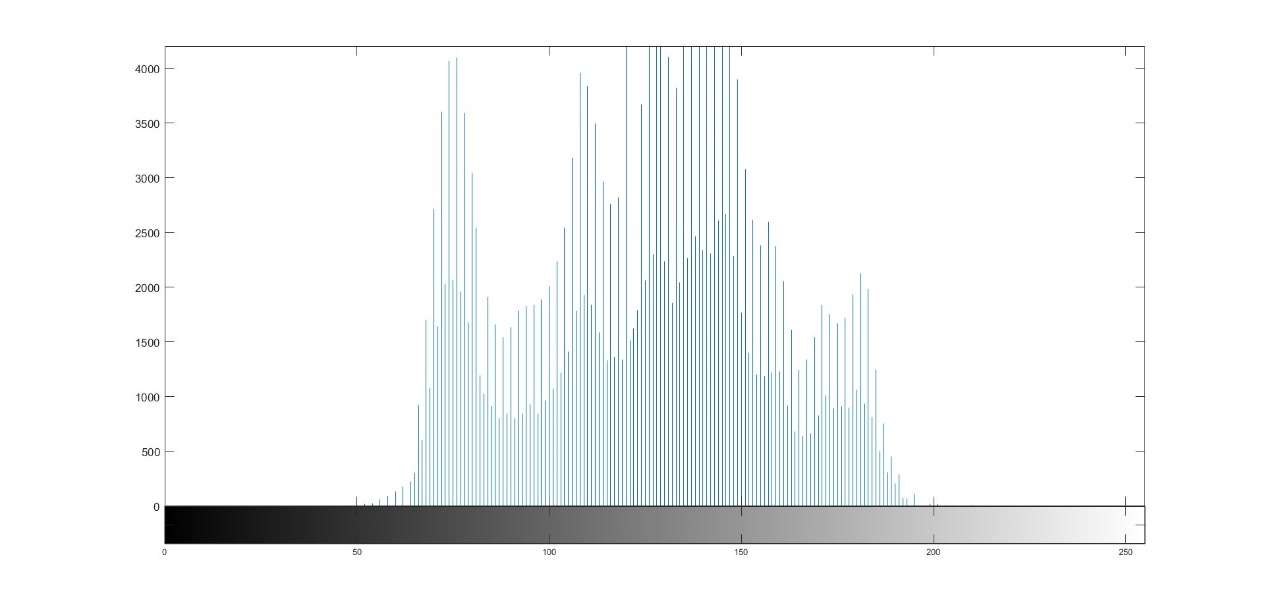


Figure 1-histogram for im\_ldr

**(2)**

**Histogram Equalization is a method for enhancing image contrast. The main idea is to turn the histogram distribution of a pair of images into an approximately uniform distribution, thus enhancing the contrast of the image.**

**The principle of its implementation can be described as follows:**

**Suppose an image with a histogram distribution**

**and we want to change image into image using a monotone nonlinear mapping**

**, AKA applying an transformation to each pixel point in image .**

**The histogram distribution of image is .**

**and the histogram distribution of image**

**change all the pixel points inside the image with a grey level of into , then we have:**

**The above equation explained that the total number of pixel points in the corresponding interval remains unchanged. In order to achieve histogram equalization, we have in particular:**

**Since the goal is a uniform distribution of the histogram, ideal we have**

**, is the number of pixel points and is the depth of the greyscale level, usually taken as 256. We then have**

**With the discrete form:**

**Use the build-in histeq() function to obtain the histogram of im\_ldr, processed image and the histogram with greyscale level = 256 is presented below:**



Figure 2-histogram equalized image\_idr

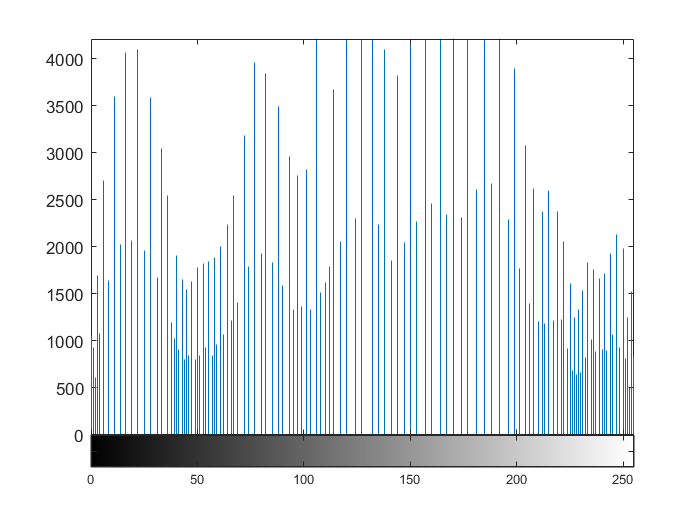


Figure 3-equalized histogram for im-idr

Task2

**(1)According to the instruction, first the input image should be segmented into 8\*8 non-overlapping blocks then a** two-dimensional **discrete cosine transform should be applied to these blocks, after the DCT, the left-top pixel of all blocks would be merged to get the output image.**

**The essential code is listed below:**

A = im2gray(imread('lenna512.bmp'));

B = zeros(64,64);

for i = 1:64

for j = 1:64

%Split the original image into non-overlapping 8\*8blocks

%making a total of 64\*64=4096

C = A((8\*i-7):8\*i, (8\*j-7):8\*j);

%Call the build-in function to perform a

%2-dimentional Fourier cosine on each block

D = dct2(double(C));

%Fill the top left element of the

%completed block to a new matrix

B(i,j) = D(1,1);

end

end

**ims is displayed below as figure4.**

**Note that imshow() function to display image could result in blank image because, because by default, when the image is double type imshow function will set the display range to [0, 1], so less than 0 will become black, greater than 1 will become white.**



Figure 4-ims

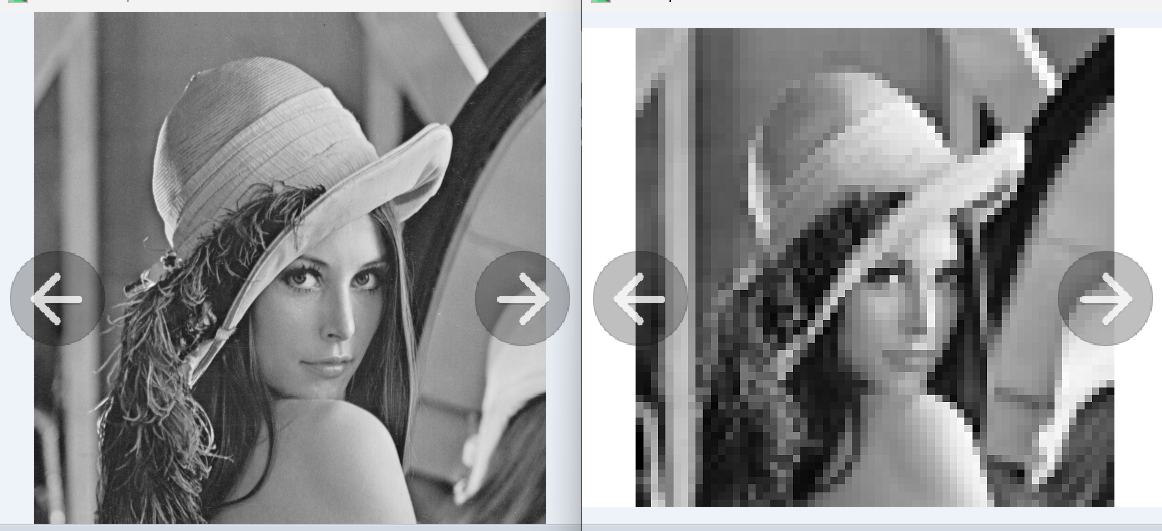


Figure 5-compare-im, ims(right)

As shown in figure5, the size of ims is only 64\*64. Comparing to the original image, it losses significant amount of details, making the pattern appears blurred.

**(2)**

**and are input parameters**

**The essential code is:**

for i = 1:64

for j = 1:64

%Split the original image into non-overlapping 8\*8blocks

%making a total of 64\*64=4096

C = A((8\*i-7):8\*i, (8\*j-7):8\*j);

%Call the build-in function to perform a

%2-dimentional Fourier cosine on each block

D = dct2(double(C));

%use the formula to quantilize each block

quantized\_blocks{end+1} = round(D./sq);

end

end

%end

**Before the stated piece of quantization process, an if conditional statement was used to determine the size of QP and decide on the values of S and sq ()**

**The results are stored in the cell array** quantized\_blocks, a cell a length of 4096, filled with 8\*8 matrixes.

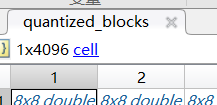


Figure 6-quantized

**(3)**

**According to the instruction, it is sufficient to invert the function procedure from the previous question, with the input parameters still being and as well as the** ims.bmp from the previous function.

**The essential code is:**

or k = 1:4096

%reverse quatization

D = cell2mat(quantized\_blocks(1,k))\*sq;

%reverse dct2

for i = 64

for j =64

B = idct2(D\*A(i,j));

imo((8\*i-7):8\*i, (8\*j-7):8\*j) = B;

end

end

end

**where an example**

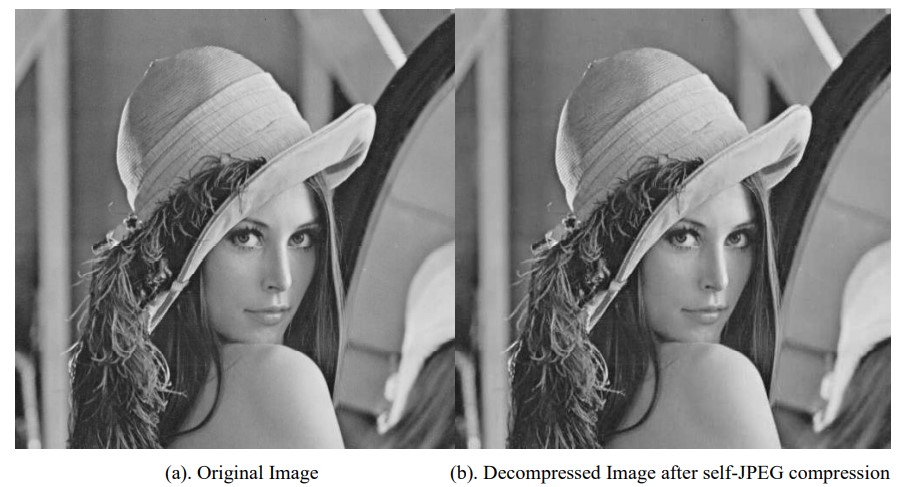


Figure 7-imo and decompressed image

**(4)**

**PSNR was calculated using the function obtained in lab1**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **QP** | **1** | **16** | **31** | **46** | **61** | **76** | **91** | **106** |
| **PSNR** | 5.2409 | 5.4512 | 7.2018 | 9.7552 | 16.4299 | 19.5752 | 8.7549 | 5.6591 |

**As shown below, the larger the QP, the brighter the compressed image will be; the smaller the QP, the less bright the compressed image will be. As both high and low brightness will cause distortion, the PSNR varies in a "V" shape, and when the brightness is right (as shown in the table, around QP=79), the compressed image will be closest to the original and the PSNR will be the largest.**

Task3

**(1)**

**the predictive values p1, p2, p3, p4 are from applying 2\*2 block, 3\*3block with direct and weighted averages**

**for the** **2\*2 direct average predictive value p1, the essential code is listed below:**

% Start predict

for k = 2:m-1

for s = 2:n-1

p1(k,s) = (im(k,s-1)+im(k-1,s)+im(k-1,s-1)) / 3;

e1(k,s) = im(k,s)- p1(k,s);

end

end

% Edge filling

p1(1, :) = [];

p1(511, :) = [];

p1(:, 1) = [];

p1(:, 511) = [];

p1 = padarray(p1, [1 1], 'replicate');

**where** m n are the number of rolls and columns of the original image matrix.

**A same method is applied to get the** **3\*3 direct averaged predictive value p2.**

**For the 2\*2 weighted averaged predictive value p3 and the 3\*3 direct averaged predictive value p4.**

**The predict code is listed below, weighted factors were added.**

% Start predict

for k = 2:m-1

for s = 2:n-1

p3(k,s) = (3\*im(k,s-1)+3\*im(k-1,s)+2\*im(k-1,s-1)) / 8;

e3(k,s) = im(k,s)- p3(k,s);

end

end

**the resulted images along with the original image are presented below**

****

Figure 8-corresponding image of p1, p2, p3, p4 and the original image

**It is difficult to comment on the differences in these images.**

**(2)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Original image** | **2X2 block**  **with direct**  **average(p1)** | **2X2 block with**  **weighted**  **average(p3)** | **3X3 block**  **with direct**  **average(p2)** | **3X3 block**  **with weighted(p4)** |
| **entropy** | **7.3775** | **7.4271** | **7.4271** | **7.4116** | **7.4178** |

**The entropy is given by the formula:**

**Where is the probability of occurrence at each grey level**

**The essential code of the entropy is listed below:**

H\_x1 = -Ps(k)\*log2(Ps(k))+ H\_x1; % The formula for entropy

**Where** Ps(k) **is the probability of each pixel.**

**The higher the entropy, the clearer the image and the more homogeneous the image is internally. For the images in this lab, this indicates that the more similar the probability of each grey value within it.**

**The lower the entropy, the easier the image is to be compressed, as the information entropy must increase during the compression process.**

**Therefore, the original image is best suited for compression due to the lowest entropy.**

Task4

**Answers of (1) and (2) are combined for convenience**

**This task requires the use of image morphology operations to implement a function for license plate number recognition.**

**Image morphology methods are used to extract image components from an image that are meaningful for expressing and depicting the shape of a region, enabling subsequent recognition to capture the most essential (and most discriminative) shape features of the target object, such as boundaries and connected areas.**

**The basic idea is to use structural elements with a certain morphology to measure and extract the corresponding shapes in an image for the purpose of image analysis and recognition.**

**There are four basic operations in mathematical morphology: expansion, erosion, opening and closing. In this task, the corresponding shapes are extracted using the erosion operation.** **Erosion is one of the most basic morphological operations, it can remove boundary points from an image, shrink the image inwards along the boundary or remove parts smaller than the specified structural elements. Erosion is used to 'shrink' or 'refine' the foreground of a binary image, thereby enabling noise removal, element segmentation and other functions. In the process of etching, a structure element is usually used to scan the image to be etched pixel by pixel and the result is determined by the relationship between the structure element and the image to be etched.**

**Therefore, the specific morphological operations implemented by the function, along with essential code in this task are:**

**Note that the process described here is a binary morphological transformation, so the original figure must first be binarized.**

%image binarization

%the license

im\_plate = rgb2gray(im\_plate);

im\_plate = 255\*double(im\_plate>200);

im\_plate = im\_plate(60:150,70:415);

%figure, imshow(im\_license)

% template

im\_template = rgb2gray(im\_template);

im\_template = 255\*double(im\_template>200);

%invert the image becasuse in morphological operations,

%the background of all the images should be set to black

im\_template = 255-im\_template;

%figure, imshow(im\_tem)

1. **use the structural element to corrupt the corresponding letter templates and split out the single character blocks, as shown in the image below. And corrode the license plate and extract the corresponding letter features.**

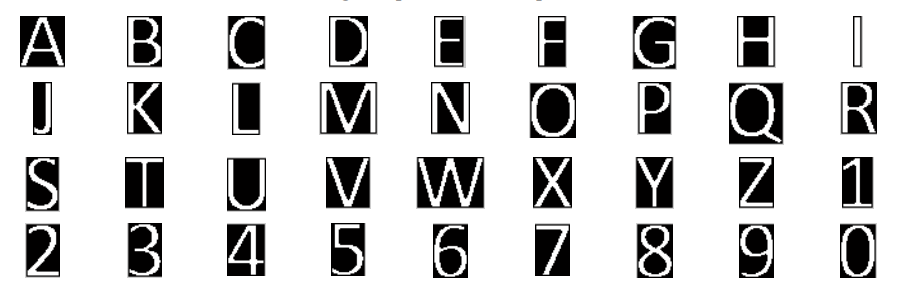


Figure 9-blocks

% erosion section

%the structure element

SE = strel('disk', 1, 0);

% erode the templates

im\_template\_erode = imerode(im\_template, SE);

%figure, imshow(im\_tem\_erode)

%split out each characters as masks

Note that the range of the block split is very important and the right parameters are needed to split the individual characters completely.

k = 0;

for i = 1:79:426

for j = 1:80:426

k = k+1;

masks(1:80,1:81,k) = im\_template\_erode(i:i+79,j:j+80)/255;

%figure, imshow(mask(:,:,k))

end

1. **Compare the characteristics of the character block with those of the letters in the license plate, by using the letter string mask. The letters that can be masked (i.e. the difference between the binary features of the two results is small) in the character blocks and license plate in the template are identified successfully.**

X = [];

A = 1:36;

X\_string = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','1','2','3','4','5','6','7','8','9','0'];

% store the index of A\_string

X\_index = [];

for i = 1:36

% find the row and column of the value equals 1

[a,b] = find(masks(:,:,A(i))==1);

% find the range of the characteristic, up, down, left, right

row\_up = min(a);

row\_down = max(a);

column\_left = min(b);

column\_right = max(b);

se = masks(row\_up-1:row\_down+1,column\_left-1:column\_right+1,A(i));

% store the se as a variable

[~,x] = find(imo\_1==1);

for j = 1:size(x,1)

if ~isempty(x(j))

% the corresponding characters of the position

X\_index = [X\_index,i];

end

X = [X,x(j)];

end

end

X\_final = sort(X);

% find the correspondence of value

str = [];

for i = 1:size(X\_final,2)

for j = 1:size(X,2)

if X\_final(i) == X(j)

Index = X\_index(j);

end

end

str = [str,X\_string(Index)];

**However, the result obtained by this process is not correct, as shown in the figure**





Figure 10-wrong result

It seems that during the corrosion operation, some problems can cause several letters to be incorrectly identified, the letters that will be incorrect are 'C', 'F', 'I', 'J', 'L' and 'O'.

The wrong recognition results from the similarity between one part of ‘C’ and ‘Q’, ‘J’ and ‘L’, ‘F’, ‘I’ and ‘O’.

Therefore, these letters need to be treated separately.

The improved function’s idea and essential codes for solving this problem are as follows.

1. Firstly, detect the alphanumeric characters (A, B, D, E, G, H, K, M, N, P, Q, R, S, T, U, V, W, X, Y, Z, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0) that will not lead to Recognition Error.

% fit the masks in the erosion operation

% not include the special case

X\_A = [];

A = [1,2,4,5,7,8,11,13,14,16:36];

A\_string = ['A','B','D','E','G','H','K','M','N','P','Q','R','S','T','U','V','X','Y','Z','W','1','2','3','4','5','6','7','8','9','0'];

A\_index = [];

However, a shrink operation is added before the erosion operation to eliminate pixels in the plate that may cause errors

imo = imerode(im\_plate,se);

imo\_1 = bwmorph(imo, 'shrink', inf);

2. Try to compare letters that may cause errors. To avoid errors, compare the result of the comparison knot (difference in pixel values) with the result of the previous comparison step (e.g., ‘c’ vs. ‘q’), and if the difference between them is very small, then the result is correct.

% this is for specail case

X\_B = [];

B = [3,6,9,10,12,15];

B\_string = ['C','F','I','J','L','O'];

A same shrink operation is added before the erosion operation as well

Now, compare the difference between the pixel value disparity, which is the correct result only if it is very small. Since the shrink operation has previously been used to minimize the error pixels, the result is now more accurate and allows errors to be excluded

for j = 1:size(x,1)

D = [];

for i = 1:size(temp,2) % compared with normal case

D = [D, abs(x(j)-temp(i))];

end

if min (D)<15 % if the distance is very small, it just one character

X\_B = X\_B;

else

if x <size(im\_plate,2)-5 & x > 5

temp = [temp,x(j)];

X\_B = [X\_B,x(j)];

B\_index = [B\_index,k]; % if it is the true character, then add the index

else

X\_B = X\_B;

end

end

end

end

The function can now give the correct result





Figure 11-correct result

****SUMMARY****

In this experiment students learned how to use MATLAB to perform morphological operations. Students mastered advance image processing technics including using histogram. Students gained insight into important concepts such as predictive coding. Students also develop skills on coding and mathematical to progress in the field of image processing.

# Appendix

**images**

Lenna512.bmp---------pre-given original image

lenna512\_low\_dynamic\_range.bmp-------------pre-given im\_ldr-

histogram\_lenna512\_low\_dynamic\_range.jpg-------------- generated histogram for im\_ldr in task1q1

lenna512\_ldr\_euqal.bmp----------------- generated histogram equalized image of im\_ldr in task1q2

equalized\_histogram\_lenna512\_low\_dynamic\_range.bmp-------------- generated equalized histogram of im\_ldr in task1q2

ims.bmp--------------generated compressed image in task2q1

compare2.bmp-----------image for comparing ims and the original image in task2q2

quantized\_blocks.png------a glance of quantified block for task2q2

p1.bmp--------------image from **2\*2 direct averaged predictive value p1 for task3q1**

p2.bmp-------------image **3\*3 direct averaged predictive value p2 for task3q1**

p3.bmp--------------image from **2\*2 weighted averaged predictive value p3 for task3q1**

p4.bmp------------image from **3\*3 weighted averaged predictive value p4 for task3q1**

task3.bmp--------------image with all predicted values and the original image **for task3q1**

task3.jpg------ image with all predicted values and the original image **for task3q1**

alphanumeric\_templates.bmp--------------pre-given image for task4

car\_license\_plate.bpm--------------pre-given image for task4

wrong.png-----------image for demonstrating results of task4\_draft.m for task4

right.jpg-----------image demonstrating results of task4\_improved.m for task4

**Scripts**

Task1.m--------script for task1

Task2.m----------------- function script for task2

Task2\_main.m-------------- main function script for task2

if\_task2\_main\_not\_working.m-------------- script for task2

task3.m-------------- script for task3q1

task3\_entropy.m------------ script for task3q2

task4\_draft.m--------------function that generates wrong result for task4

task4\_improved.bmp------------improved function that generates right result for task4

**other materials**

quantized\_blocks.mat--------------quantified generated 8\*8 block in task2q2, stored as a cell

quantized.mat--------------quantified generated 8\*8 block in task2q2, sored as a matrix