Xi'an Jiaotong-Liverpool University

西交利物浦大學

EEE330

IMAGE PROCESSING

Lab 1 Report

Author:
Ruihao Wang

Student Number:
1405884

March 27, 2018

Contents

1 Introduction

This lab is a brief introduction of MATLAB Image Processing Toolbox which contains various built-in functions for basic operations such as read/display, crop, resize and quantization in image processing. With these functions in MATLAB, students have a chance to be familiar with image processing steps by finishing several tasks that correspond to operations previously mentioned. In this report, the results of each task will be firstly reported and then several crucial points will be discussed as well. The full code of MATLAB implementation are collected in appendix.

2 Task 1: Read and display images

The first task is read and display an image. Function `imread` is used to read an image file and convert it to numeric matrix that MATLAB can handle. This matrix has a shape of (512, 512) as the image file `'lenna512.bmp'` is a monochrome image which has only one channel. Each element inside matrix, ranging from 0 to 255, is a grayscale value which corresponds to one pixel of original image. Then, function `image()` is called to visualize the matrix or, in other word, display the image. Its result can be seen as in Figure 1 which is an unexpected one with wrong aspect ratio and default color mapping that is different from gray scale of original image.

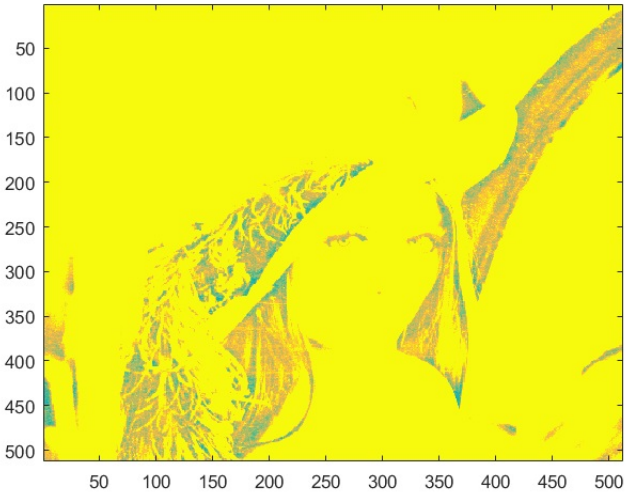
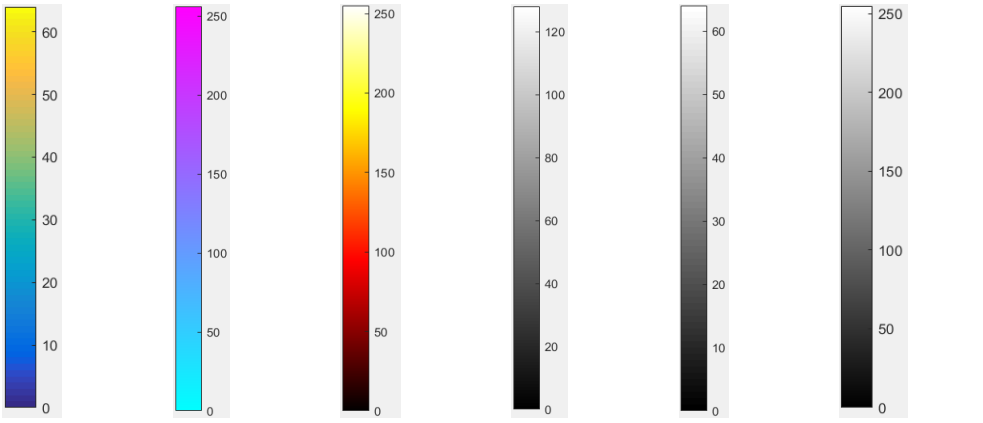


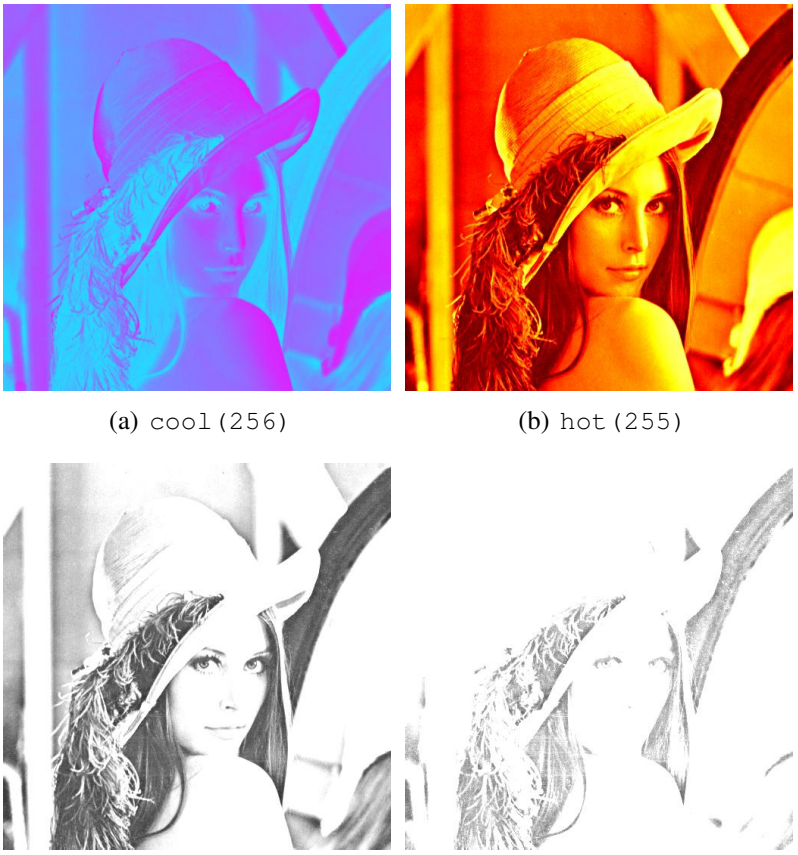
Figure 1: Output using `image()`

Function `image()` handles the read matrix as common data in MATLAB without the concerns about color mapping and aspect ratio. It has a default setting of color mapping as visualized in Figure 2 which will treat values more than 64 as color 'yellow' while the elements inside matrix are concentrate in the upper range of this bar or excess this range. Therefore, we have such an unexpected result whose majority appears yellow.



(a) `default` (b) `cool(256)` (c) `hot(255)` (d) `gray(128)` (e) `gray(64)` (f) `gray(256)`

Figure 2: Output images with different color map



(a) `cool(256)` (b) `hot(255)`

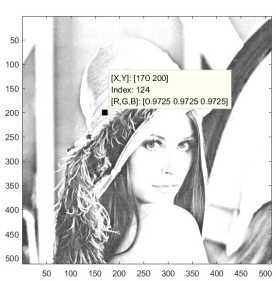
(c) `gray(128)` (d) `gray(64)`

Figure 3: Output images with different color map

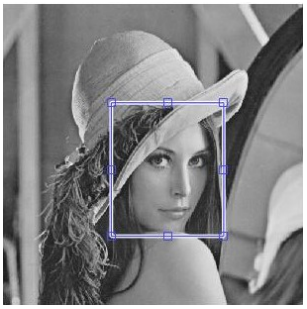
A desired display result needs extra commands: `colormap cool(256)` for specified color mapping and `truesize` for recover the actual aspect ratio of image. Figure 3 shows the displayed images with various kinds of color mapping. Inside the brackets of each color mapping name, there is an integer specifying the number of levels. For `cool(256)` and `hot(255)`, the number of level is exactly or almost equal to that in origin gray scale images. Thus, we can see the displayed images become chromatic but not as 'overexposed' as in Figure 3 (c) and (d). These two 'overexposed' images suffer the same problem that values in matrix map to brighter color.

3 Task 2

3.1 Task 2-1: Cropping the face



(a) `datacusormode`



(b) `imcrop()`

(c) Result of (a)

(d) Result of (b)

Figure 4: Results of Task 2-1

This task, which has two methods to deal with, is to crop the face area from the original image. First, the coordinates of pixels can be read under `datacursormode` as shown in 4(a). Combining such coordinates with the estimated width and height, the desired area of face can be determined in image matrix and Figure 4(c) shows result. That is selection of specified rows and columns of original matrix. The other method is to use function `imcrop()` in MATLAB Image Processing Toolbox. It

provides a graphical interface, as shown in Figure 4(b), to make it convenient to crop specified region of original image. The blue bounding box restricts this region and its result is shown in Figure 4(d).

```
ul = [160, 190];     % upper-left point
side = [240, 190];   % estimated width and height
% A is image matrix
cropped = A(ul(1):ul(1)+side(1), ul(2):ul(2)+side(2));
```

Listing 1: Commands of first method: select columns and ros in matrix

3.2 Task 2-2: Downsampling / Upsampling

In this task, students are required to sequentially downsample and upsamle original images. After these operations, we have a recovered image with original size but the pixel values are varies from the original image. Such a difference is due to the estimation of pixels when image interpolation. The estimation of those interpolated pixels are based on various rules including `'nearst'`, `'bilinear'` and `'bicubic'`. The results of different interpolation can be seen in Figure 5. On the left side, there seems to be no distinct difference among downsampled images. On the right side, however, the upsampled images or recovered images have much more obvious differences. Image upsampled using `'nearest'` has jagged margins compared with the other two images. The one using `'bilinear'` is much more smooth but slightly blurred while `'bicubic'` scheme gives a better result with less blur and jagged margins.

To measure difference introduced by image interpolation, **peak signal to noise ratio (PSNR)** is introduced, which is defined as

$$PSNR = 10 \cdot log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot log_{10}(MAX_I) - 10 \cdot log_{10}(MSE) \tag{1}$$

where MAX_I is the maximum possible pixel value of the image. Generally, when pixels are represented using N bit, the value should be $2^N - 1$. on the other hand, **MSE represents mean squared error** which actually calculate the difference between two images

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \tag{2}$$

Instead of calculating SNR for each single pixel in image, PSNR can much more effectively measure the variation referred to original image. From the definition above, it is obvious that a larger PSNR means less noise or variation. In Table 1, the PSNR of different interpolation schemes are collected. Among the results, `'bicubic'` interpolation has the most PSNR which means it has the best performance of image recovery.

(a) Downsampling `'nearest'` (b) Upsampling `'nearest'`

(c) Downsampling `'bilinear'` (d) Upsampling `'bilinear'`

(e) Downsampling `'bicubic'` (f) Upsampling `'bicubic'`

Figure 5: Results of Task 2-2

Interpolation type	`nearest`	`bilinear`	`bicubic`
PSNR	28.2912	31.3847	34.0863

Table 1: PSNR of different types of interpolation

3.3 Task 2-3: Quantization

In Task 1, it is simply remapping the color represent of numbers inside matrix without changing these values. Rather than the remapping, quantization is a kind of lossy image compression that compressing a range of numbers as a single value. The original image has a color mapping of `gray(256)` which means the pixel values vary in range from 0 to 255. Thus, 8-bit depth image is required to achieve this resolution of levels. If the pixel values are integers varies from 0 to 15, only 4 bit per pixel is necessary to store the image. Figure 6 collects the results of Task 2-3, where 6(b) is my reimplementation of quantization while 6(c) using built-in function in Toolbox. Compared with the smooth shading in original image, the quantized two have more distinctive levels.

(a) Original image (b) Naive Quantization (c) Quantization using toolbox

Figure 6: Results of Task 2-3

4 Discussion

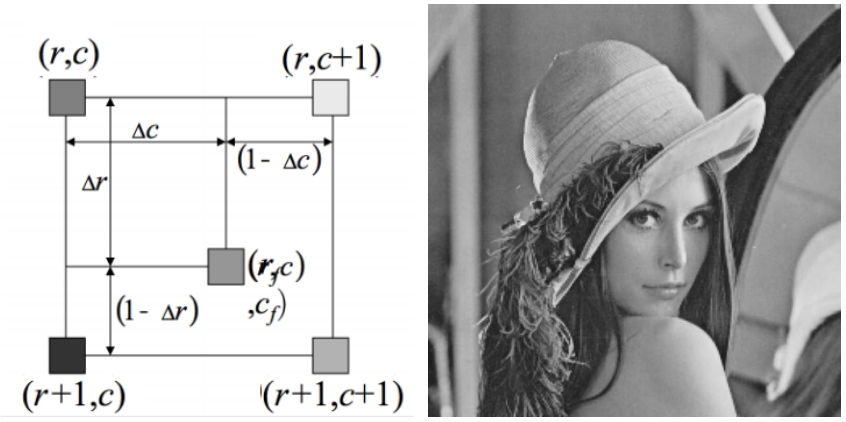
4.1 Image interpolation

With the help from various source of reference, the bilinear scheme is reimplemented without Toolbox function `imresize()` in order to further understand image interpolation(code refers to appendix Task 2.2c: Reimplementation Interpolation). The key point is to construct a mapping that projects pixel in output image to input one. The index of those projected points are not integer any more.

Based on their variation from adjacent integer index pixels, as shown in Figure 7(a), the gray scale of these points will be estimated using following equation.

$$\begin{aligned} I'(r', c') = & I(r, c) \cdot (1 - \Delta r) \cdot (1 - \Delta c) \\ & + I(r + 1, c) \cdot \Delta r \cdot (1 - \Delta c) \\ & + I(r, c + 1) \cdot (1 - \Delta r) \cdot \Delta c \\ & + I(r + 1, c + 1) \cdot \Delta r \cdot \Delta c \end{aligned}$$

The reimplementation result is shown in Figure 7(b). The image was sequentially downsampled and upsampled to original size. Compared with original image, there is inevitable blur and the PSNR of this result is 33.5235.



(a) Projected point (b) Result from reimplementation

Figure 7: Bilinear scheme reimplementation

4.2 Bit depth and image file format

When the storage of quantized images using `imwrite()` function from MATLAB Image Processing Toolbox, there is an discovery that the stored file is a completely dark image. Bit depth and color mapping are speculated to be the reason after referring the official document of MATLAB [1]. If some arguments of `imwrite()` are not specified, the image file will be automatically stored based on data type of image matrix. In the lab case, matrix has `uint8` data which means 8-bit unsigned integer. They can represent numbers from 0 to 255. Thus, when writing images to local disk, the file is automatically saved as 8-bit depth with `gray(256)` by default. However, quantization has rearranged the numbers from 0 to 15 which correspond to the dark part in color mapping. Thus, the argument in functions were specified as

```
imwrite(A_2,gray(16),'.\Task-2-3\matlab_quant.png','BitDepth',4)
```

Once the bit depth and color mapping had been specified, the image was correctly saved as shown in Figure 6(b) and 6(c). Here, as `.bmp` file supports only 1-bit, 8-bit and 24-bit depth, `.png` format supporting 4-bit depth was selected.

4.3 Implementation of image quantization

My own implementation of quantization (see code in appendix Task 2.3) is simply dividing original range into the number of requested levels. Each new level contains 16 levels of older one. Original values are divided by 16 and rounded down to an integer. This is a direct way to force numbers into the interval of (0,15). The result refers to Figure 6(b). It has a lower contrast than quantized image using Toolbox. The MATLAB implementation of quantization is much more complicated than my easy version. Function `multithresh()` generates multiple threshold values as the levels of new image with Otu's Method [2]. Differing from simple implementation generates thresholds with the same interval of 16, such a threshold values are not linear and related to the histogram of quantizing image:

```
[39 52 65 78 91 104 116 129 141 153 165 178 192 204 210 216]
```

This will result a more precise quantization level and a more concentrated histogram which means a higher contrast. Therefore, two quantized images appears quite different.

5 Conclusion

Required tasks in lab including crop, resize and quantization were accomplished. For a further learning, reimplementation of algorithms such as bilinear interpolation and quantization were compared with results from methods using Toolbox. Additionally, several discoveries associated with bit depth and image format were considered by coincidence. Through this lab, the knowledge of basic image processing and how MATLAB handles digital images have been enhanced.

6 Appendix: Matlab Code

Task 1: `colormap, truesize`

```
%%
% This script is used for EEE330 Lab_1 Task_1
% Author: Ruihao Wang
% ID: 1405884
% Contents: Use different colormap to display and keep the size of
   image

%%
A = imread('lenna512.bmp');
figure(1);
image(A);

%%
figure(2)
image(A)
colormap cool(256);
truesize;
imwrite(A, cool(256),'./Task-1/cool-256.jpg');

%%
figure(3);
image(A);
colormap hot(255);
truesize;
imwrite(A, hot(255), './Task-1/hot-255.jpg');

%%
figure(4)
image(A)
colormap gray(128);
truesize;
imwrite(A, gray(128),'./Task-1/gray-128.jpg');

%%
figure(5)
image(A)
colormap gray(64);
truesize;
imwrite(A, gray(64), './Task-1/gray-64.jpg');
```

Task 2.1: `imcrop, datacursormode`

```
%%
% This script is used for EEE330 Lab_1 Task_2_1
% Author: Ruihao Wang
% ID: 1405884
% Contents: Crop the face of the image

%%
% manually crop using matrix operation
ul = [160, 190];
side = [240, 190];
A = imread('lenna512.bmp');
cropped = A(ul(1):ul(1)+side(1), ul(2):ul(2)+side(2));
figure(1);
image(cropped);
colormap gray(256);
truesize;
imwrite(cropped, gray(256), '.\Task-2-1\cropped.jpg');

%%
% use built-in function imcrop
[cropped_2, rect] = imcrop(A);
imwrite(cropped_2, gray(256), '.\Task-2-1\cropped_2.jpg');
```

Task 2.2 ab: Downsampling / Upsampling `imresize('nearest')`

```
%%
% This script is used for EEE330 Lab_1 Task_2_2
% Method: Nearset Neighbor Intepolation
% Author: Ruihao Wang
% ID: 1405884
% Contents: Down-sampling and Up-sampling

%%
% Down-sample original image using nearest neighbor interpolation
I_0 = imread('lenna512.bmp');
I_1 = imresize(I_0, 0.5, 'nearest');
figure(1);
imshow(I_0);
imwrite(I_0, gray(256), '.\Task-2-2\original.jpg');
figure(2);
imshow(I_1);
imwrite(I_1, gray(256), '.\Task-2-2\nearest_down.jpg');
```

```
18
19 %%
20 % Up-sample the previous down-sampled image using nearest neighbor
21 % interpolation
22 I_1_prime = imresize(I_1, 2.0, 'nearest');
23 figure(3);
24 imshow(I_1_prime);
25 imwrite(I_1_prime, gray(256), '.\Task-2-2\nearest_recover.jpg');
26
27 %%
28 % Calculate PSNR
29 peaksnr = psnr(I_1_prime, I_0);
30 fprintf('\n The Peak-SNR value is %0.4f', peaksnr);
```

Task 2.2 ab: Downsampling / Upsampling `imresize('bilinear')`

```
1  %%
2  % This script is used for EEE330 Lab_1 Task_2_2_c
3  % Method: Bilinear Interpolation
4  % Author: Ruihao Wang
5  % ID: 1405884
6  % Contents: Down-sampling and Up-sampling
7
8  %%
9  % Down-sample original image using bilinear interpolation
10 I_0 = imread('lenna512.bmp');
11 I_1 = imresize(I_0, 0.5, 'bilinear');
12 figure(1);
13 imshow(I_0);
14 figure(2);
15 imshow(I_1);
16 imwrite(I_1, gray(256), '.\Task-2-2\bilinear_down.png');
17
18 %%
19 % Up-sample the previous down-sampled image using bilinear neighbor
20 % interpolation
21 I_1_prime = imresize(I_1, 2.0, 'bilinear');
22 figure(3);
23 imshow(I_1_prime);
24 imwrite(I_1_prime, gray(256), '.\Task-2-2\bilinear_recover.jpg');
25
26 %%
27 % Calculate PSNR
```

```
peaksnr = psnr(I_1_prime, I_0);
fprintf('\n The Peak-SNR value is %0.4f', peaksnr);
```

Task 2.2 ab: Downsampling / Upsampling `imresize('bicubic')`

```
%%
% This script is used for EEE330 Lab_1 Task_2_2_c_optional
% Method: Bicubic Interpolation
% Author: Ruihao Wang
% ID: 1405884
% Contents: Down-sampling and Up-sampling

%%
% Down-sample original image using bilinear interpolation
I_0 = imread('lenna512.bmp');
I_1 = imresize(I_0, 0.5, 'bicubic');
figure(1);
imshow(I_0);
figure(2);
imshow(I_1);
imwrite(I_1, gray(256), '.\Task-2-2\bicubic_down.png');

%%
% Up-sample the previous down-sampled image using bilinear neighbor
% interpolation
I_1_prime = imresize(I_1, 2.0, 'bicubic');
figure(3);
imshow(I_1_prime);
imwrite(I_1_prime, gray(256), '.\Task-2-2\bicubic_recover.jpg');

%%
% Calculate PSNR
peaksnr = psnr(I_1_prime, I_0);
fprintf('\n The Peak-SNR value is %0.4f', peaksnr);
```

Task 2.2 c: Reimplementation of Interpolation

```
function resized = bilinear_re(im, out_dims)

% Get i/o images' dimension
in_row = size(im,1);
in_col = size(im,2);
out_row = out_dims(1);
out_col = out_dims(2);
```

```
8
9  % Make a grid for output images
10 % Coordinate pairs
11 [rf, cf] = meshgrid(1:out_row, 1:out_col);
12
13 % Calculate the ratio of two sizes
14 s_r = in_row / out_row;
15 s_c = in_col / out_col;
16
17 % Project grid to input image
18 rf = rf * s_r;
19 cf = cf * s_c;
20
21 % Get integer part
22 r = floor(rf);
23 c = floor(cf);
24
25 % Any values out of range, tailored
26 r(r < 1) = 1;
27 c(c < 1) = 1;
28 r(r > in_row - 1) = in_row - 1;
29 c(c > in_col - 1) = in_col - 1;
30
31 % Delta distance
32 delta_r = rf - r;
33 delta_c = cf - c;
34
35 % Convert coordinates to equivalent linear index
36 in1_ind = sub2ind([in_row, in_col], r, c);
37 in2_ind = sub2ind([in_row, in_col], r+1,c);
38 in3_ind = sub2ind([in_row, in_col], r, c+1);
39 in4_ind = sub2ind([in_row, in_col], r+1, c+1);
40
41 % Interpolation for each channel
42 resized = zeros(out_row, out_col, size(im, 3));
43 resized = cast(resized, class(im)); % match the data type
44
45 for channel = 1:size(im,3)
46     chan = double(im(:,:,channel));
47     estimate = chan(in1_ind).*(1 - delta_r).*(1 - delta_c) + ...
48                chan(in2_ind).*(delta_r).*(1 - delta_c) + ...
49                chan(in3_ind).*(1 - delta_r).*(delta_c) + ...
50                chan(in4_ind).*(delta_r).*(delta_c);
```

```
    resized(:,:,channel) = cast(estimate, class(im));
end
```

Task 2.3: Quantization `multithresh, imquantize`

```
%%
% This script is used for EEE330 Lab_1 Task_2_3
% Author: Ruihao Wang
% ID: 1405884
% Contents: Image quantization

%%
% manually quantization
A = imread('lenna512.bmp');
quant_lev = 16;

figure(1)
imshow(A);
title('Original image');
imwrite(A, '.\Task-2-3\original.jpg');

A_1 = uint8(floor(A ./ 16));
figure(2)
imshow(A_1, gray(16));
title('Naive quatization');
imwrite(A_1, gray(16), '.\Task-2-3\naive_quant.jpg');
figure(3)
imhist(A_1, gray(16));
title('Histogram of Naive quatization');

%%
% quantization using MATLAB functions
threshGray = multithresh(A,quant_lev);      % generate multi-
   threshold
A_2 = imquantize(A, threshGray); % returns quantized matrix ranging
    (0, 16)
figure(4)
imshow(A_2, gray(16));
title('MATLAB functions used');
imwrite(A_2, gray(16), '.\Task-2-3\matlab_quant.jpg');
figure(5)
imhist(A_2, gray(16));
title('Histogram of MATLAB Implementation');
```

```
37
38
39 %%
40 %diff = range(A_1) - range(A_2);
```

References

[1] MathWorks, "MATLAB Official Documentation," [Online] https://cn.mathworks.com/help/matlab/ref/imwrite.html?lang=en.

[2] Wikipedia, "Otsu's method," [Online] https://en.wikipedia.org/wiki/Otsu%27s_method.