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

EEE330 - Image Processing
Assessment 1

Student Name: Wang Xiaoyang
Student ID: 1405888

March 28, 2018

Contents

1 Image Read and Display

In this section, an image of Lenna is read and displayed by different color-maps in Matlab. Figure 1 shows the original bitmap picture by matlab function *image*. Following figures are illustrated by different color-map and fixed by function *truesize* so the size is square as 512 × 512. The effect of changing color-maps is to map pixel values to different colors systems. As shown in the Figure 2 (a) to (d), picture that is mapped by *cool(256)* is filled by cool colors such as blue and purple, while picture with *hot(255)* looks warm because of colors like red and yellow. Then, there are two gray color-maps *gray(64)* and *gray(128)* which has different number of color levels. Therefore, it is intuitive to observe that Figure 2 (d), image with 128 gray levels, is more recognizable than that with 64 levels because more color levels means more color information and higher mapping accuracy.

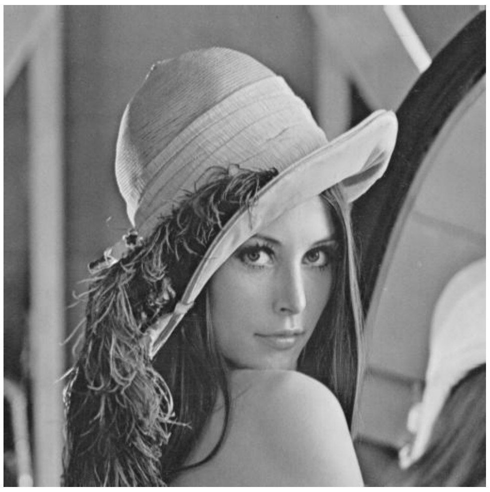


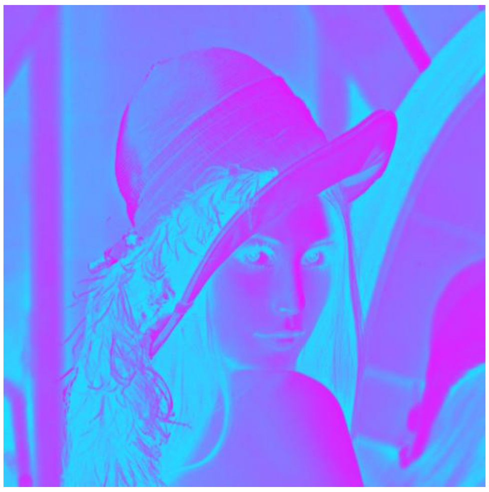
Figure 1: Original image of Lenna

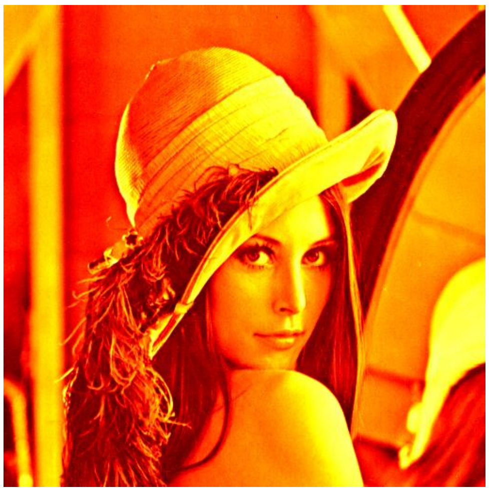
2 Image Crop

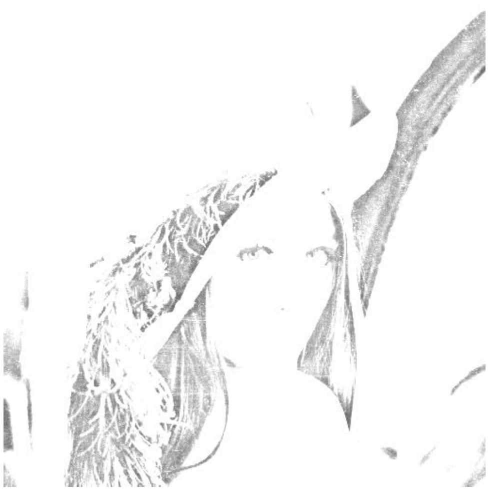
In this section, function *datacursormode* is called to obtain the coordinate points in original Lenna displayed bu *imshow*. These information are used to crop a square area containing face of Lenna as shown in Figure 3.

3 Image Resize

In this section, the original Lenna image is firstly down-sampled as half size and then up-sampled to recover the size but with lower quality. This procedure is repeated by three different resize algorithm, which are *Nearest neighbor interpolation, Bilinear interpolation and Bicubic interpolation*. The first method does not require calculation while the other two need a huge amount of calculations. As for bilinear interpolation, the algorithm takes 4 pixels in original image to determine 1 pixel in new image. In bicubic interpolation method, 16 pixels are necessary for the same purpose. The matlab built-in functions are firstly called. Figure 5 (a) to (f) illustrate all resized images. To compare the sampling quality of different methods, the peak signal noise ratio (PSNR) is calculated separately and the results are listed in Table 1. From the data, bicubic interpolation method has the best sample quality and the nearest neighbor


(a) Cool(256)


(b) Hot(255)


(c) Gray(64)

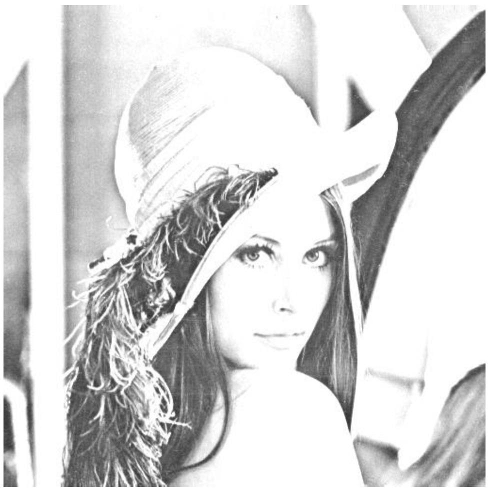

(d) Gray(128)

Figure 2: Image processed by different color-maps

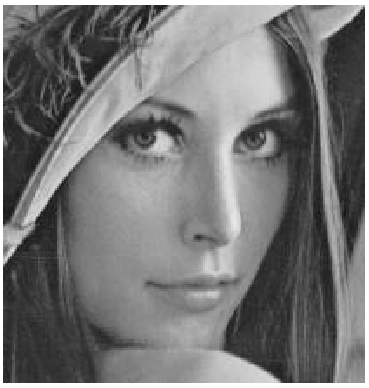


Figure 3: Cropped Lenna's face

interpolation performs worst. Then, the self-implementation of first two method are validated and the PSNR results are listed in Table 2. It can be observed that the results are the same for nearest neighbor interpolation method but that of bilinear algorithm is worse than built-in function.

Table 1: PSNR Results by built-in functions

Sample Method	PSNR(dB)
Nearest Neighbor Interpolation	28.2912
Bilinear Interpolation	31.3847
Bicubic Interpolation	34.0863

Table 2: PSNR Results by self-implementations

Sample Method	PSNR(dB)
Nearest Neighbor Interpolation	28.2912
Bilinear Interpolation	29.5219

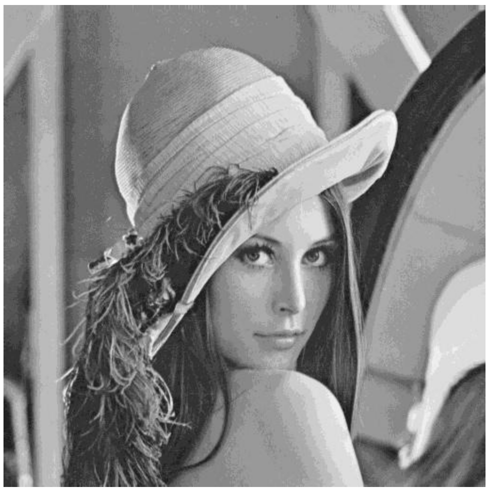
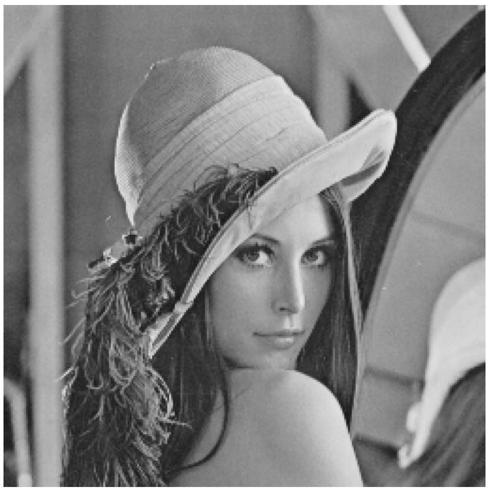


Figure 4: Image after being quantized

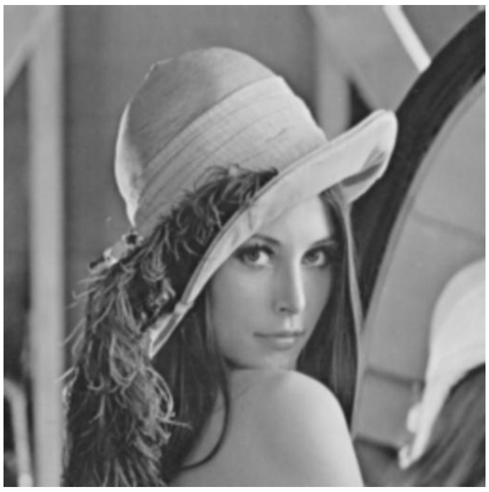
4 Image Quantization

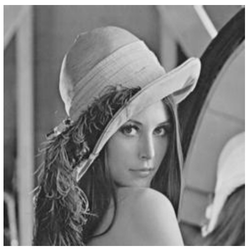
In this section, the original image is quantized from 256 to 16 gray levels. It means every 16 levels in original map now only indicate 1 level in new image. For example, levels 0 to 15 will be mapped to new level 0 and levels 16-31 will be new level 1, which is a floor operation on quotient of original value and 16. Accordingly, the quantized image can be obtained as shown in Figure 4. By observation, the new image is less smooth than original one. Several obvious lines can be observed in the area of Lenna's shoulder, which means there are larger color differences as a consequence of fewer gray levels.

(a) Down-sampled by nearest neighbor interpolation


(b) Up-sampled by nearest neighbor interpoltation

(c) Down-sampled by bilinear interpoltaion


(d) Up-sampled by bilinear interpoltaion


(e) Down-sampled by bicubic interpoltaion

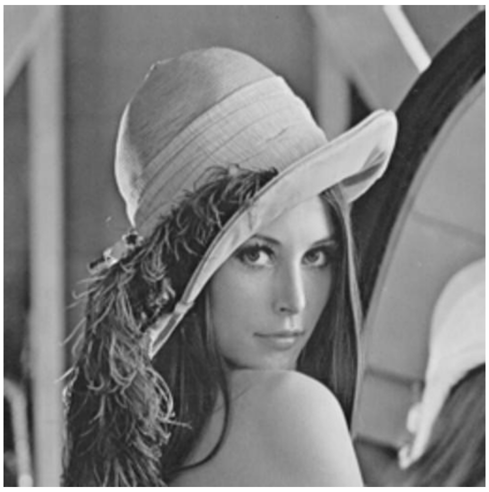

(f) Up-sampled by bicubic interpoltaion

Figure 5: Image down-sampled and up-sampled

Appendices

A Image Read and Display

```
imdata = imread('lenna512.bmp');
figure; image(imdata); truesize;
colormap cool(256); % colormap cool(256)
figure; image(imdata); truesize;
colormap hot(255); % colormap hot(255)
figure; image(imdata); truesize;
colormap gray(128); % colormap gray(128)
figure; image(imdata); truesize;
colormap gray(64); % colormap gray(64)
```

B Image Resize by Built-in functions

```
im = imread('lenna512.bmp');
%% Nearest neighbor interpolation
ds_NNI = imresize(im,0.5,'nearest');
up_NNI = imresize(ds_NNI,2.0,'nearest');

%% Bilinear interpolation
ds_BL = imresize(im,0.5,'bilinear');
up_BL = imresize(ds_BL,2.0,'bilinear');

%% Bicubic interpolation
ds_BC = imresize(im,0.5,'bicubic');
up_BC = imresize(ds_BC,2.0,'bicubic');

%% Display resampled images
figure; imshow(ds_NNI); figure; imshow(up_NNI);
figure; imshow(ds_BL);  figure; imshow(up_BL);
figure; imshow(ds_BC);  figure; imshow(up_BC);

%% PSNR
peaksnr1 = psnr(up_NNI,im);
peaksnr2 = psnr(up_BL,im);
peaksnr3 = psnr(up_BC,im);
```

C Image Resize by self-implementations

C.1 Test

```
org_im = imread('lenna512.bmp');
new_im1 = nearest_neighbor(org_im,0.5);
new_im2 = nearest_neighbor(new_im1,2);
new_im3 = bilinear(org_im,0.5);
new_im4 = bilinear(new_im3,2);
%% PSNR calculation
peaksnr1 = psnr(new_im2,org_im);
peaksnr2 = psnr(new_im4,org_im);
%% Display
figure; imshow(new_im1);
figure; imshow(new_im2);
figure; imshow(new_im3);
figure; imshow(new_im4);
```

C.2 Function Nearest Neighbor Interpolation

```
function new_im = nearest_neighbor(org_im,scale)
% Org_im: Input original image
% scale: scale fo resize operation
% new_im: Resized image;

[dim1,dim2] = size(org_im);
newdim1 = dim1*scale;
newdim2 = dim2*scale;
new_im = zeros([newdim1 newdim2]);

for i=1:newdim1
    for j =1:newdim2
        spdim1 = ceil(i/scale);
        spdim2 = ceil(j/scale);
        new_im(i,j) = org_im(spdim1,spdim2);
    end
end
new_im = uint8(new_im);
end
```

C.3 Function Bilinear Interpolation

```
function new_im = bilinear(org_im,scale)
% Org_im: Input original image
% scale: scale fo resize operation
% new_im: Resized image;

[dim1, dim2] = size(org_im);
newdim1 = dim1*scale;
newdim2 = dim2*scale;
new_im = zeros([newdim1,newdim2]);

for i=1:newdim1
    for j =1:newdim2
        scale_dim1 = i/scale;
        scale_dim2 = j/scale;
        refdim1 = floor(scale_dim1);
        refdim2 = floor(scale_dim2);
        u = scale_dim1 - refdim1;
        v = scale_dim2 - refdim2;
        if refdim1==0
            refdim1=1;
        else if refdim1>=dim1
            refdim1=dim1-1;
            end
        end
        if refdim2==0
            refdim2=1;
        else if refdim2>=dim2
            refdim2=dim2-1;
            end
        end
        new_im(i,j)=u*v*org_im(refdim1+1,refdim2+1)...
        +(1-u)*(1-v)*org_im(refdim1,refdim2)...
        +u*(1-v)*org_im(refdim1,refdim2+1)...
        +(1-u)*v*org_im(refdim1+1,refdim2);
    end
end
new_im = uint8(new_im);
end
```

D Image Quantization

D.1 Test

```
im = imread('lenna512.bmp');
new_im = quantization(im,4);
imshow(new_im);
```

D.2 Function Quantization

```
function qt_im = quantization(org_im,qt_factor)
% org_im: Input original image
% qt_factor: Input the power of desired quantization level
% qt_im: Output quantized image

new_lv=uint8(2^qt_factor);    % Obtain number of levels
qt_im=(org_im/new_lv)*new_lv; % Quantize values and map back
end
```