EE 440 Autumn 2018
Homework 1 Report

Professor: Ming-Ting Sun

October 6th, 2018

He Feng 1427841

Problem 1 Image processing and commercial tools

a. I used Adobe Photoshop for this question. Initially I clicked ‘image’ and then ‘image size’ to enlarge the horizontal and vertical dimensions by a factor of 2.5. My original picture is on the left, and my enlarged picture is on the right:

b. Similar to the last part, I used Photoshop to adjust the photo. The adjusted photo loos warmer, and it gives us the feeling that the exciting game is playing under the sunshine. Here is the adjusted photo:

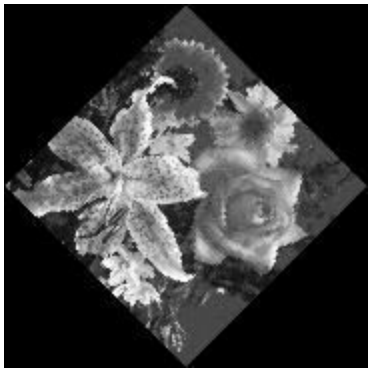
Problem 2 Matlab basics: image I/O and data types

Initially I use imread and imshow function to load and display the given picture, then I used class function to determine the type of the image. To show the maximum and minimum data value for the image, I used max and min function to get the ideal result. One thing that need to be noticed that I used image(:) as input of the function instead of image. The next part asked me to convert the data to the double type and show it on the screen. However, this step failed because we cannot show the double-typed image by using imshow function. Hence we transfer the image back to uint8-typed such that it can be shown on the screen. In the future if we want to display a double-typed image, we need to convert it to uint8 type before using imshow function. Here is the Matlab code for this question:

```
% He Feng
% EE 440 HW 1 Problem 2
close all;
clear all;

% Load and show the photo 1_4.bmp.
image = imread('1_4.bmp');
figure(1)
imshow(image);

% Get the type of the image data.
% Get the maximum and minimum data values for the image.
class(image)
max_number = max(image(:))
min_number = min(image(:))

% Convert the data to the double type
% We can not show the double-typed image by imshow
figure(2)
doubled_image = double(image);
imshow(doubled_image);

% To display a double image by using imshow, we need to transfer the image
% into the uint8 format.
figure(3)
imshow(uint8(doubled_image));

```

Problem 3 Matlab basics: Matlab commands

The first step of this section is to convert the image into 256-level grayscale image. Hence I need to read the image and its associated colormap. I used ind2gray function to get the image Y. Then I rotate the new grayscale image 45 degrees clockwise to generate a new image which called Z. At last I used imwrite function to store the image Y and Z in the file. Here are the image Y and image Z and the screenshot of the Matlab code:



```
% He Feng
% EE 440 HW 1 Problem 3
close all;
clear all;

% Read the photo and its associate colormap.
[X,map] = imread('1_2.tif','tif');

% Show the original photo X.
figure(1)
imshow(X,map);

% Transfer the photo X into gray scale.
Y = ind2gray(X,map);
figure(2)
imshow(Y);

% To make sure the type of image.
% Y is a 256 level gray scale image.
class(Y)

% Rotate image Y.
Z = imrotate(Y,-45);
figure(3)
imshow(Z);

% Store the image Y and Z into the folder.
imwrite(Y,'Y_image.jpg');
imwrite(Z,'Z_image.jpg');

```

Problem 4 Image Resolution

The first section of this problem wants us to reduce the resolution of an image by a factor of 4 in both vertical and horizontal dimensions by using two methods. The first method is to keep one pixel out of every 4x4 pixel area. I just used the command Y1 = X(1:4:384,1:4:256) to generate the image. (384, 256) is got from Workspace. Because this image data is in double-typed, hence we need to divided by 256 when I used imshow function. For the second method we need to replace every pixel area and replace with its average number. In this case I made two extra functions to get the result. The first one is called ‘scaleimage.m', which has the same influence as our first method above. The second function is called ‘averagethenscaleimage.m’, it contains the first function. I initially make the convolution, and then I used the new determined image as

input of the first function and get the final result. Here are the two pictures after resolution in order and the Matlab code:

```
% He Feng
% EE 440 HW 1 Problem 4
close all;
clear all;

% Draw the original picture.
figure(1)
X = load('1_3.asc');
imshow(X/256);

% Keep one pixel out of every 4x4 pixel area, and display image Y1.
figure(2)
Y1 = X(1:4:384,1:4:256);
imshow(Y1/256);
Y1_uint8 = uint8(Y1);
imwrite(Y1_uint8,'Y1.jpg');

% Replace every pixel area and replace with its average number by using two
% additional functions.
figure(3)
Y2 = averagethenscaleimage(X,4);
imshow(Y2);
imwrite(Y2,'Y2.jpg');

```

```
% He Feng

% This is the basic scaling function. We need to consider different cases
% as the scale factor can be an odd number or an even number.
function thumbnail = scaleimage(image, scalefactor)

% This is the remainder after divide scalefactor by 2.
remainder = mod(scalefactor, 2);

% Different cases when the remainder is 0 and 1.
if remainder == 0
    thumbnail=image((scalefactor/2: scalefactor: end-scalefactor), (scalefactor/2: scalefactor: end - scalefactor));
else
    thumbnail=image((((scalefactor+1)/2): scalefactor: end-scalefactor), (((scalefactor+1)/2): scalefactor: end - 1));
end


end
```

```
% He Feng

% We make the convolution first, and then used the new determined image as
% the input of the basic scaling function to get the final result.

function thumbnail = averagethenscaleimage(image, scalefactor)

% Make a matrix full of one.
one = ones(scalefactor, scalefactor);

% Make a kernal.
h = (1/(scalefactor*scalefactor)).*one;
double_image = double(image);
% Do the convolution
image_cov2 = conv2(double_image, h);
new_image = uint8(image_cov2);

% Scale the image after averaging.
thumbnail = scaleimage(new_image, scalefactor);

end
```

For the second part of this question, we want to enlarge the image from last step with a factor of 4 by using two methods. The first method is pixel repeating and the second one is bilinear interpolation. Both of them need an extra function to support. For the first method, I used for loop in the function such that it can go through all the elements of the initial image. As it goes through each element, I enlarge each of them such that the whole image can be enlarged. For the second method, I used for loop as well to go through the initial image, then I used if and elseif statement because there are different cases in this section. The two pictures after the process are shown below, and the Matlab code is also shown below:

For the two pictures above, they are not shown as their real size. Because the report has to be neat, hence I changed their size to make the report looks better. If you want to see the details please check the folder that I uploaded.

```
% Using pixel repeating to enlarge the image
figure(4)
Y1_new_1 = pixelrepeating(Y1,4);
imshow(Y1_new_1/256);
Y1_new_1_uint8 = uint8(Y1_new_1);
imwrite(Y1_new_1_uint8,'Y1_new_1.jpg');

% Using bilinear interpolation to enlarge the image.
figure(5)
Y1_new_2 = bilinearInter('Y1.jpg',4,4);
imshow(Y1_new_2);
Y1_new_2_uint8 = uint8(Y1_new_1);
imwrite(Y1_new_2_uint8,'Y1_new_2.jpg');
```

```
% He Feng
% EE 440 HW 1

% By using this function, we will enlarge the image by repeat the pixels in
% pixel area.
function output = pixelrepeating(image, scalefactor)

[a,b] = size(image);
% Initially set the output all zeros, which will be changed later.
output = zeros(scalefactor*a, scalefactor*b);

for x = 1:a
    for y = 1:b
        i = scalefactor*(x-1)+1;
        j = scalefactor*(y-1)+1;

        % Repeat each pixel to enlarge the image.
        for m = 1:scalefactor-1
            for n = 1:scalefactor-1
            output(i,j) = image(x,y);
            output(i+m,j) = image(x,y);
            output(i,j+n) = image(x,y);
            output(i+m,j+n) = image(x,y);
            end
        end
    end
end
end
```

```
% He Feng
% This function have the same role as interp2() function.
function IM = bilinearInter(im,varargin)

    image = imread(im);
    scale(1)=varargin{1};
    scale(2)=varargin{2};

    % recognize the size of the picture
    [row,column,layer]=size(image);% check the size

    % the size of the new picture
    Row = round(row*scale(1));
    Column = round(column*scale(2));

    % Create a new picture starts with full of ones.
    IM = uint8(ones(Row,Column,layer));

    % calculate the pixels of the new picture
    for i = 1:Row
        for j = 1:Column

            % The corresponding point in the original picture.
            x=i/scale(1);
            y=j/scale(2);

            % Check whether the corresponding point will overflow
            if x < 1
                x = x+1;
            elseif x > row
                x = x-1;
            end
```

```
33
34 -                if y < 1
35 -                    y = y+1;
36 -                elseif y > column
37 -                    y = y-1;
38 -                end
39
40                  % Round the number into the next smaller integer.
41 -                a = floor(x);
42 -                b = floor(y);
43
44                  % Round the corresponding point of the original picture toward
45                  % zero.
46                  % If the corresponding point is an integer
47 -                if x == fix(x) && y == fix(y)
48 -                    IM(i,j,:)=image(x,y,:);
49
50                  % Represents two points (x,b), (x,b+1)
51 -                elseif x == fix(x) && y ~= fix(y)
52 -                    IM(i,j,:) = image(x,b+1,:).*(y-b) + image(x,b,:).*(b+1-y);
53
54                  % Represents two points (a,y), (a+1,y)
55 -                elseif x~=fix(x) && y == fix(y)
56 -                    IM(i,j,:) = image(a+1,y,:).*(x-a)+image(a,y,:).*(a+1-x);
57
58                  % Otherwise, solve the vertical part fitst and then solve the
59                  % horizontal part.
60 -                else
61 -                    R1=image(a+1,b,:).*(x-a)+image(a,b,:).*(a+1-x);
62 -                    R2=image(a+1,b+1,:).*(x-a)+image(a,b+1,:).*(a+1-x);
63 -                    IM(i,j,:)=R2.*(y-b)+R1.*(b+1-y);
64 -                end
65
66 -            end
67 -        end
68 -    end
69
```

By observing the differences among two enlarged picture and the original 1_3.asc. We can see they look similar, but there are some differences among them. For the output of pixel repeating method, the output picture looks blurrier than the original image because we most of pixels are got from copy and the values have a little bit changed. For the output of bilinear interpolation, it looks better than the last picture, but there are still have some tiny differences based on observing the data of the output. Generally, they looks the same, and the output picture from bilinear interpolation looks more similar to the original picture.