CS 559 Fall 2020

Assignment 2 Grading Sheet

Please attach this sheet to your solutions

Last Name: **Giang**
First Name: Stephen
Red ID: 823184070

The following scores have been recorded for you. Please notify the lab assistant if you notice any discrepancies.

A1	A2	A3	A4	A5

Questions 1	
Question 2	
Question 4	
Question 3	
Question 4	
Question 5	
Extra Credit	
Total	

Question 1

Criteria	Possible scores	Earned	Comments
Explanation of condition	5		
relationship between L and n	8		

Question 2

Criteria	Possible scores	Earned	Comments
Explanation of solution	5		
Drawing precisely the normalized histogram	8		

Question 3

Criteria	Possible scores	Earned	Comments
Explanation of solution	5		
Drawing precisely the normalized histogram	9		

Question 4

Criteria	Possible	Earned	Comments
Explanation of the solution	8		
Program	12		
Results	10		

Question 5

Criteria	Possible	Earned	Comments
Explanation of the solution	8		
Program	12		
Results	10		

Extra Credit

Criteria	Possible	Earned	Comments
Ideas, Program, report, demonstration with images	15		

Homework 2
Computer Vision
CS 559
Stephen Giang RedID: 823184070

Problem 1: Explain conditions under which the use of a lookup table (LUT), instead of calculating the mapping pixel by pixel, reduces the computation. Express the conditions in terms of number of graylevels L and resolution n.

Notice the following:

Calculating the mapping pixel by pixel would force us to traverse through every pixel and apply the mapping calculating a total of n^2 times (that is for a square n $\times$ n image).

With the use of the lookup table, we know the following: $0 \leq f(x, y) < L$. This means we can make a table with L elements, each element being the mapping calculation of each grayscale levels to the new outputted grayscale level. Now that all the calculations are done, we use the table to assign each pixel value from the original image to the new image. This means we only need to do L calculations and n^2 assignments.

If $n^2 > L$, we reduce the computation by $n^2 - L$ using the lookup table.

If $n^2 = L$, then either method is just as good for computations.

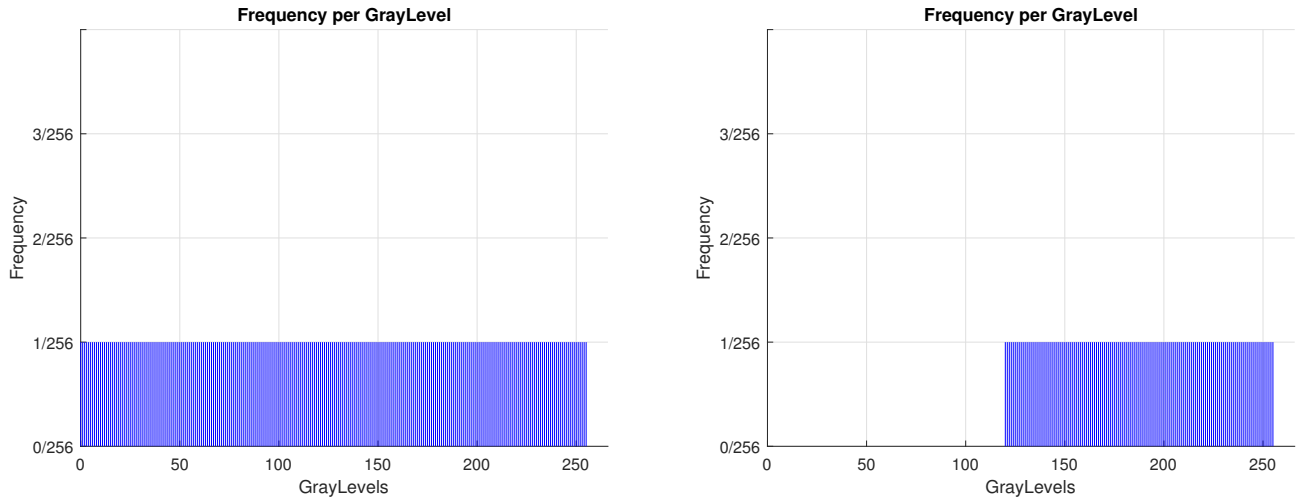
If $n^2 < L$, then calculating the mapping pixel by pixel would be reduce the computations by $L - n^2$ computations.

Problem 2: Many (infinite number) of passes of contrast stretching with the mapping

$$g = \begin{cases} 0.4f & 0 \leq f \leq 119 \\ f & 120 \leq f \leq 255 \end{cases}$$

is applied to a perfectly equalized 8-bit input image. Draw and carefully label the normalized histogram of the output image.

Notice that if f is a perfectly equalized 8-bit input image, then its histogram looks like the left histogram. If we take our mapped image, we get the right histogram:



Notice that with a perfectly equalized image, we get a flat histogram where there is an equal frequency with each graylevel, frequency being the probability of choosing a random pixel with a specific grayscale level. Because of the floor function when multiplying by .4, if we do this an infinite number of times, we get that all grayscale levels goes to 0 for $0 < f \leq 119$. This means that the frequency of 0 go to 120/L = 120/256. (There is a tiny bar over 0, that goes to 120/256 on the right histogram).

Problem 3: An 8-bit image has a normalized histogram defined by

$$h(l) = \begin{cases} \frac{1}{206} & 0 \leq l < 100 \\ \frac{1}{412} & 100 \leq l < 200 \\ \frac{1}{206} & 200 \leq l < 256 \end{cases}$$

Suppose that each pixel is ANDed with 0100 0000, and is set to white if the result of ANDing is non-zero, otherwise is set to black if the result is zero (this operation is called bit-plane slicing). Draw and carefully label the normalized histogram of the resulting output image.

The conversion from grayscale level (G) to binary code (B) is found through the linear combination:

$$G = a(128) + b(64) + c(32) + d(16) + f(8) + h(4) + j(2) + k(1)$$

where we have $a, b, c, d, f, h, j, k \in \{0, 1\}$. From this, we get $B = a\,b\,c\,d\,f\,h\,j\,k$

For $B = a\,b\,c\,d\,f\,h\,j\,k$, notice that when ANDing with 0100 0000, we only get nonzero results when $b = 1$. We get zero when $b = 0$.

(a) For $0 \leq G < 64$, we get $b = 0$.

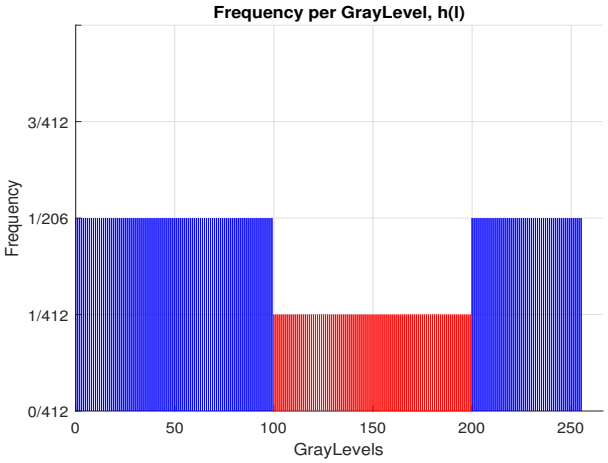
(b) For $64 \leq G < 128$, we get $b = 1$.

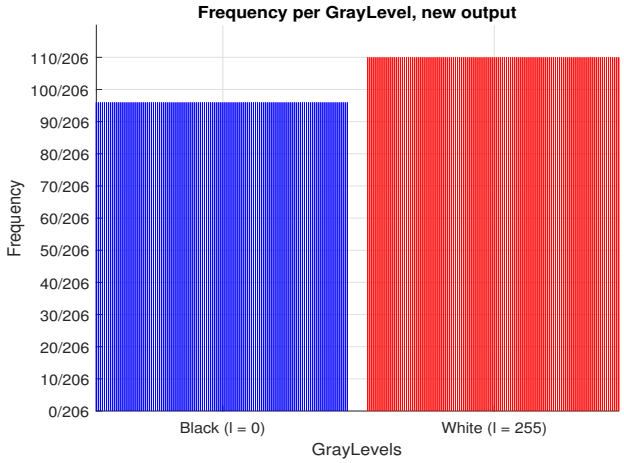
(c) For $128 \leq G < 192$, we get $b = 0$.

(d) For $192 \leq G < 256$, we get $b = 1$.

Using the above results, we get that for greyscale values $0 \leq l < 100$, 64 grayscale levels become 0 (Black). For greyscale values $100 \leq l < 200$, 64 grayscale levels become 0 (Black). For greyscale values $200 \leq l < 256$, 0 grayscale levels become 0 (Black). From these results, we get the following frequencies:

$$\text{Black Frequency} = \frac{1}{206}(64) + \frac{1}{412}(64) + \frac{1}{256}(0) = \frac{96}{206}$$
$$\text{White Frequency} = 1 - \text{ Black Frequency } = 1 - \frac{96}{256} = \frac{110}{206}$$





Problem 4: The purpose of this assignment is for you to come up with a measure (formula, procedure, or algorithm) to determine the similarity between two color images, with 0 being completely difference and 100 being exactly the same. Explain why your measure is appropriate. Program your procedure and apply it to determine how similar each image B, C or D is similar to the image A. One method would be to compare the histogram of the images, but you can propose an alternative method of your own. I do not expect you to proposed a sophisticated measure, rather I want you to appreciate that images that are so easy for our eyes to identify their similarity and differences, are very difficult to do the same using computer vision. Note: Download this assignment, then open it, click on each image and save it as .jpg.

(A) (B) (C) (D)

For this program, what I did was I found the difference between image A and image B in RGB values at each pixel. Then I divided each element by 255 to see a percent change. From there, I added all the entries, and divided by the total number of entries. This gave me a difference average from pixel to pixel. To find the similarity, its just (1 - average).

Notice my results:

- Image A is about 100.00 % the same as Image A
- Image B is about 87.80 % the same as Image A
- Image C is about 96.90 % the same as Image A
- Image D is about 68.78 % the same as Image A

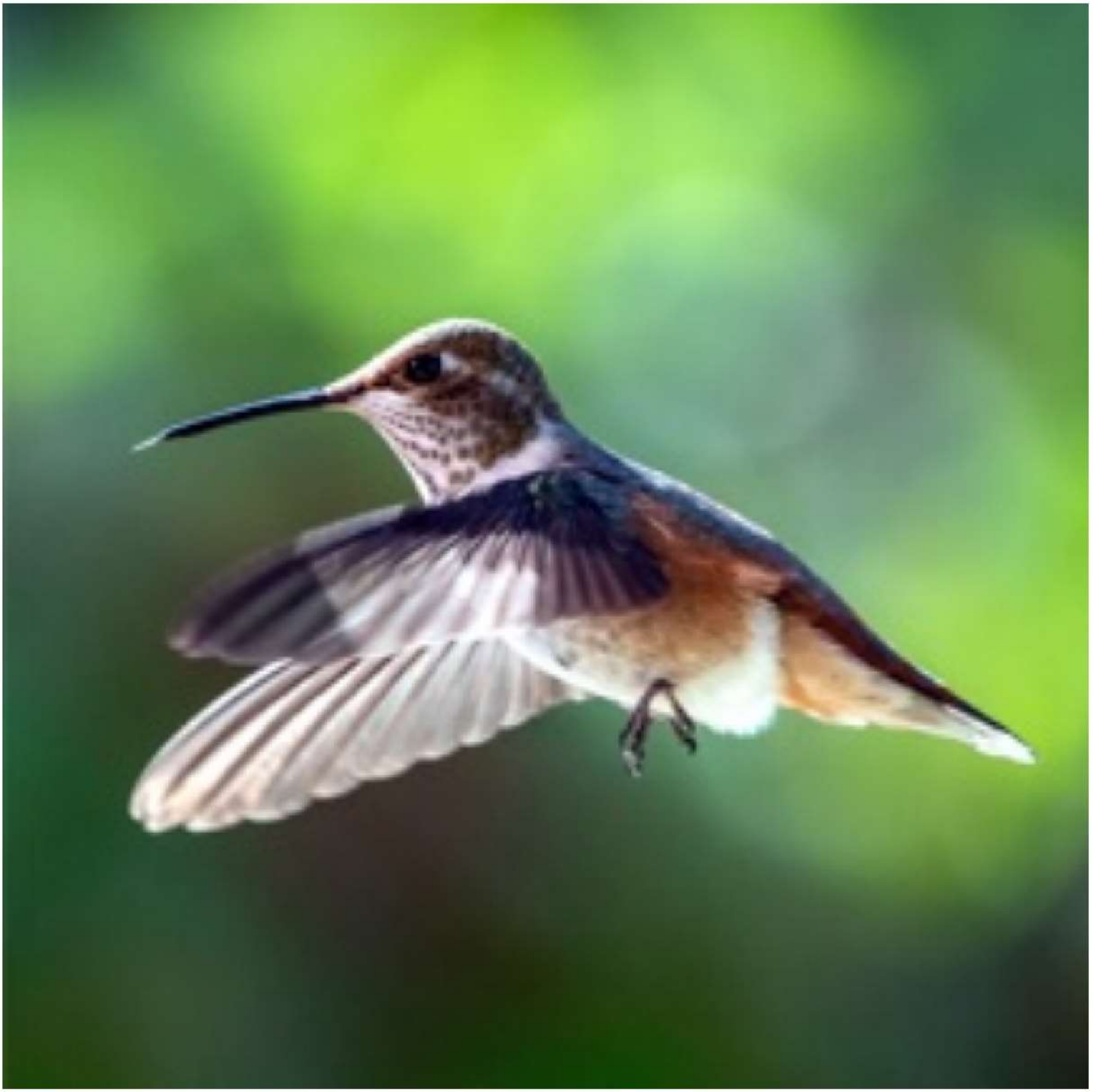
```
function [] = similar(imgA, imgB)

    imgB = imresize(imgB, [size(imgA,1), size(imgA,2)]);
    diffMat = abs(double(imgA) - double(imgB)) ./ 255;

    sum = 0;
    for row = 1 : size(diffMat,1)
        for col = 1 : size(diffMat,2)
            for colr = 1 : 3
                sum = sum + diffMat(row, col, colr);
            end
        end
    end
    average = sum / (size(diffMat,1) * size(diffMat,2)* size(diffMat,3));


    fprintf('The Second Image is about %.2f %% the same as the First Image\n', (1 -↙
average)*100);
end
```

```
clear
close all

A = imread('..\Photos\A.jpg');
B = imread('..\Photos\B.jpg');
C = imread('..\Photos\C.jpg');
D = imread('..\Photos\D.jpg');


similar(A,A)
% The Second Image is about 100.00 % the same as the First Image

similar(A,B)
% The Second Image is about 87.80 % the same as the First Image

similar(A,C)
% The Second Image is about 96.90 % the same as the First Image

similar(A,D)
% The Second Image is about 68.78 % the same as the First Image
```

Problem 5: Write a program in Python or Matlab that takes an RGB image of size W by H and an integer k and produces an output image of size kW by kH where k is an integer greater than one (sometime this is called resampling). The enlarged image must be in appearance as close as possible to the original image. Use bilinear interpolation or another technique of your choice. Demonstrate your work by producing an input image and two output images one with k=2 and one with k=3.

For this program, what I did was I mapped each pixel (x,y) to (kx,ky). Then for the pixels between (kx,ky) and (k(x+1),ky), I filled it in with a weighted average. For example, for the set $\{1\ \underline{A}\ \underline{B}\ 4\}$, I found that we can equally space each entry by a distance of $\frac{4-1}{3} = 1$. Then I simply added each left entry with that distance to get equally spaces sets between (kx,ky) and (k(x+1),ky). From there, I did the same thing but column wise.

Notice the images below with each photo being 1x, 2x, 3x respectively:



```
function B = bilinearInterpl(A, k)

    % recursion for colored images
    if (size(A,3) == 3)
        B = uint8(zeros(size(A,1)*k, size(A,2)*k,3));
        B(:,:,1) = bilinearInterpl(A(:,:,1), k);
        B(:,:,2) = bilinearInterpl(A(:,:,2), k);
        B(:,:,3) = bilinearInterpl(A(:,:,3), k);
        return
    end

    B = uint8(zeros(size(A,1)*k, size(A,2)*k));
    H = size(A,1);
    W = size(A,2);
    kH = k*H;
    kW = k*W;

    % mapping function
    for i = 1 : H
        for j = 1 : W
            B(k*i,k*j) = A(i,j);
        end
    end

    % rows
    for i = k : kH
        leftVal = 0;
        rightVal = 0;
        % finds row with mapped pixels
        if (mod(i,k) == 0)
            for j = k : kW
                % finds mapped pixels
                if (mod(j,k) == 0)
                    leftVal = j;
                    rightVal = j + k;
                else
                   distance = ( double(B(i, rightVal)) - double(B(i, leftVal)) )/ k;
                   B(i, j) = uint8(floor(double(B(i ,(j-1))) + distance));
                end
            end
        end
    end

    % columns
    for j = k : kW
        topVal = 0;
        bottomVal = 0;
        for i = k : kH
            % finds completed rows from linear interpl (row loop)
            if (mod(i,k) == 0)
                topVal = i;
```

```
                bottomVal = i + k;
            else
                distance = ( double(B(bottomVal,j)) - double(B(topVal,j)) )/ k;
                B(i,j) = uint8(floor(double(B((i-1),j)) + distance));
            end
        end
    end

    % padding for top rows
    for i = 1 : (k - 1)
        B(i,:) = B(k,:);
    end

    % padding for left columns
    for j = 1 : (k - 1)
        B(:,j) = B(:,k);
    end

end
```

```
clear
close all

figure(1)
A = imread('..\Photos\Bird.jpg');
imshow(A)

figure(2)
B = bilinearInterpl(A, 2);
imshow(B)

figure(2)
C = bilinearInterpl(A, 3);
imshow(C)
```