CSE 107 Lab 3

Histogram Equalization

Darren Correa

4/9/2022

Abstract

The goal of this lab was to show processing images using histogram equalization can improve contrast. The approach used in this lab was breaking down the producer into steps and ensuring each step was correct. First, we compute a normalized histogram of the pixel intensities of an image and visualize it using a built-in plotting function. Using our normalized histogram, we compute a transformed histogram using the histogram equalization transformation function. Having that, we can create an equalized image with corrected contrast regardless of the conditions of the contrast in the original image. The process was successfully implemented, and I am happy with the results. Our mean and standard deviation values of the equalized images are an estimate to the success of the implementation.

Technical Discussion

The first step was the normalized histogram. This collected the percent usage of a pixel value in the original image. The normalized histogram is defined as

$$p(r_k) = \frac{n_k}{MN}$$

where n_k is the number of times a pixel value is used in the image and M and N are the number of rows and columns which will give us the total number of pixels in the image. The data was recorded into a length vector whose size matched the total possible pixel within a grayscale image, 256. As per the lab instructions, if we were to sum the length vector, the total should sum to one. I included the test in my compute histogram function and the result was successful. With the correct normalized histogram and using the built-in bar graph function, bar(), I was able to see the difference in contrast between the light and dark images in a numerical way. Figures 1 and 2 show the data.

To begin correcting the contrast issues of an image, we must compute the transformation histogram. The formula is defined as

$$T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

L being the possible pixel values of the image and $p_r(r_j)$ being the values computed in our normalized histogram vector. After computing the transformation histogram, as the `equalize_img` function shows, we can assign the pixel values to our output image matrix

using the transformation histogram. What is happening is we are mapping the pixel in the original image and transforming its value to a more uniform intensity value. Images 4 and 5 are the equalized images and show the results of the implementation.

Discussion of Results

I am happy with the results from this lab as they seem to match up with the results that were shown in the lab instructions. While reading the textbook, it mentioned how the equalized images are usually remarkably similar. As shown by images 4 and 5 this is the case. When testing with the lighter image, our equalized image is darker. When using the darker image, our equalized is lighter. Comparing the two equalized images side by side and there is hardly a difference. However, there is a difference in quality. Along the curves of the statue in the equalized images, there is more pixelation than in the originals. This is because the pixel intensity values of surrounding pixels are no longer as close as before. Since the histogram is now more uniform, that would change how some parts of image would look in terms of quality. If we look at Image 1, 4 and 5 we can see there are some quality issues where pixels go from light intensities to dark intensities of shadows or background details. The equalized mean and standard deviation values of both the light and dark images are also extremely close, so having the numerical conformation solidified my confidence in the function implementations.

Results

Image 1: Original Image

Image 2: Light Contrast Image

Image 3: Dark Contrast Image

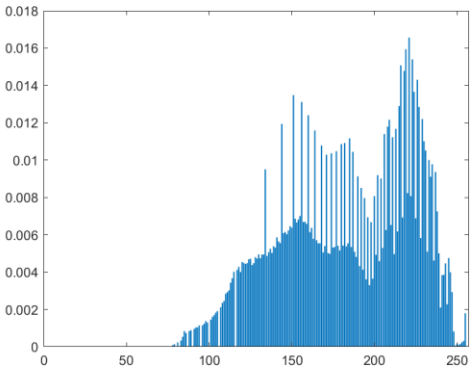

Figure 1: Light Image Histogram

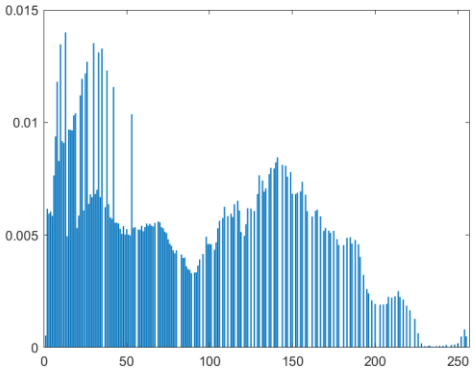

Figure 2: Dark Image Histogram

Image 4: Light Image Equalized

Image 5: Dark Image Equalized

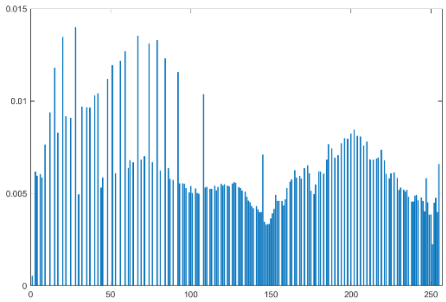

Figure 3: Light Equalized Image Histogram

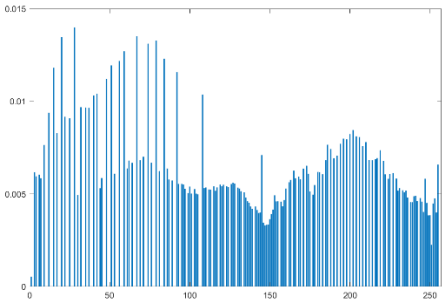

Figure 4: Dark Equalized Image Histogram

Light Image		Dark Image	
Original Mean: 182.0237	Original Standard Deviation: 39.1508	Original Mean: 82.8169	Original Standard Deviation: 60.3535
EQ Mean: 128.5759	EQ Standard Deviation: 73.8169	EQ Mean: 128.3990	EQ Standard Deviation: 73.994

Code

compute_histogram.m

```
function h = compute_histogram(img)
% comput_histogram  Computes a histogram of an input image by counting
%                   occurences of pixel values and computing the difference
%                   of the count and total number of pixels of the input
%                   image
%
% Syntax:
%   h = compute_histogram(img)
%
% Input:
%   img = uint8 matrix from an image
%
% Output:
%   h = computed normalized histogram vector. Contains only values 0 - 1
%   of which should sum to 1.
%
% History:
%   D. Correa   4/9/2022    Created


    %Initialize output h as vector
    h = zeros(1,256);

    %Count number of times each pixel value 0-255
    for i = 1:size(img,1)
        for j = 1:size(img,2)

            %Get pixel value from matrix and increment index of vector for
            % each time pixel value is found
            %Must account for vector beginning at 1 instead of 0.
            pixel_val = img(i,j);
            h(pixel_val+1) = h(pixel_val+1) + 1;
        end
    end

    %Get M by N size of image
    total_pixels = size(img,1) * size(img,2);

    %Used to check if values in vector will sum to 1
    sum = 0;

    %Divide number of times pixel values occured by total number of pixels
    for k = 1:256
        h(k) = h(k) / total_pixels;
        sum = sum + h(k);
    end



end
```

plot_histogram.m

```
function figure = plot_histogram(h)
% plot _histogram   Creates a histogram plot from normalized histogram
%
% Syntax:
%   figure = plot_histogram(h)
%
% Input:
%   h = computed normalized histogram vector. Contains only values 0 - 1
%   of which should sum to 1.
%
% Output:
%   figure = histogram plot
%
% History:
%   D. Correa   4/9/2022    Created


    %Using built in function to plot histogram
    figure = bar(h);


end
```

histogram_transform.m

```
function T = histogram_transform(h)
% histogram_transform   Compute transformed histogram vector filled with
%                       equalized pixel intensites for equalized image
%
% Syntax:
%   T = histogram_transform(h)
%
% Input:
%   h = computed normalized histogram vector. Contains only values 0 - 1
%   of which should sum to 1.
%
% Output:
%   T = transformed histogram vector.
%


    %Initialize transformation vector T
    T = zeros(1,256);

    %Using equation 3-15 from textbook
    %L = 255
    %h = p(r)
    sum = 0;
    %Take summation first of normalized histogram
    for k = 1:256
       T(k) = sum + h(k);
       sum = T(k);
    end
    %Multiply by L, the number of possible pixel intesities
    for l = 1:256
        T(l) = 255 * T(l);
    end
end
```

equalize_img.m

```
function O = equalize_img(img)
% equalize_img  Takes in a uint8 grayscale image matrix and output an image
transformed
%               by histogram equalization.
%
% Syntax:
%   O = equalize_img(img)
%
% Input:
%   img = uint8 image matrix with values 0-255.
%
% Output:
%   O = uint8 image matrix with values 0-255.
%
% History:
%   D. Correa   4/9/2022    Created

    %Initialize our output matrix
    O = zeros(size(img,1),size(img,2), 'uint8');

    %Compute normalized histogram of input image and plot
    h = compute_histogram(img);
    figure = plot_histogram(h);

    %Get mean and standard deviation from original image
    img_vec = reshape(img, 1, []);
    img_vec = double(img_vec);
    Mean_Orig = mean(img_vec);
    SD_Orig = std(img_vec);

   %Transform normalized histogram
   T = histogram_transform(h);

   %Visit each pixel in input image and assign transformed pixed value
   % to output image
   for i = 1:size(img,1)
       for j = 1:size(img,2)

           O(i,j) = round( T( img(i,j) + 1 ));


       end
   end

   %Show and save image
   imshow(O);
   %imwrite(O, 'Dark_Image_Equalized.png');
   imwrite(O, 'Light_Image_Equalized.png');

   %Compute histogram and show plot of equalized image
   h_eq = compute_histogram(O);
   eq_figure = plot_histogram(h_eq);
```

```
    %Get mean and standard deviation from equalized images
    O_vec = reshape(O, 1, []);
    O_vec = double(O_vec);
    Mean_EQ = mean(O_vec);
    SD_EQ = std(O_vec);

    %saveas(eq_figure, 'Dark_EQ_Figure.png');
    saveas(eq_figure, 'Light_EQ_Figure.png');


end
```

script.m

```
%Script to run all functions. Change image to be used when necessary.
%
% History:
%   D. Correa   4/9/2022    Created


%Read images from directory
img = imread('Lab_03_image1_dark.tif');
img2 = imread('Lab_03_image2_light.tif');

%Compute normalized histogram
h = compute_histogram(img);

%Plot normalized histogram
figure = plot_histogram(h);

%Save figures
%saveas(figure, 'Dark_Imgage_Figure.png');
%saveas(figure, 'Light_Imgage_Figure.png');

%Transform normalized histogram
T = histogram_transform(h);

%Equalize the images and compare with original data.
O = equalize_img(img);
```