Homework2 – Contrast Adjustment

Arya Araban

Abstract

When studying different sets of digital images, we might come across images which have low or high contrarast in some points, or sometimes throughout the entire image. To address this problem we'll study something known as Contrast Adjustment, which with the help of itwe can modify any image's grayscale values in order to have a better visual represntaion of it.

Report Info

Date: 1399/08/23

Keywords:
Histogram
Equalization
Contrast Intensity

1- Introduction

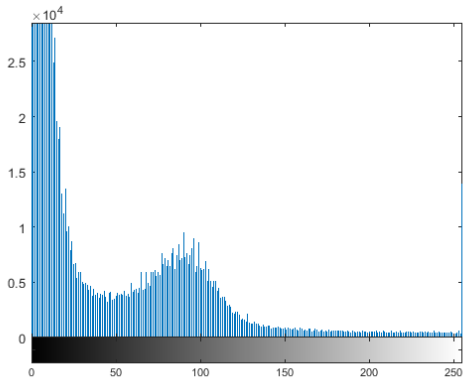
The Idea of contrast adjustment is to remap an image's original pixel intensities or colors into a new range of intensities or colors using a single mapping function that is applied to all its pixels. These mapping functions will usually be dependant on a single concept known as the Image's "Histogram", which is a gray-scale value distribution showing the frequency of occurrence of each gray-level value in an image. With the help of histograms, we can adjust the pixels of an image in such a way, that the overall image will be more pleasing to look at.

2- Technical Description

Problem 2.1.1:
Find a way to compute the histogram of a given Image using its grayscale values.

Solution:
In order to have a representation of a Histogram, We use a two-dimensional plot, In which the vertical axis specifies a given pixel(which can be a value in the range of [0-255]), and the horizontal axis gives us the occurrence of each of the pixels throughout the entire image.

An example of a histogram would be as follows:



In order to find the histogram of a given grayscale image, we loop over the entire image's pixels 256 times, in each iteration counting the frequency of a specific grayscale value, and then finaly storing the calculated number. To find a proper place to store the value, we can use an array with

image will have its histogram equalized.

for the sake of an example lets consider we have eight gray levels, doing the operations as stated above will look something like this:

Gray level	0	1	2	3	4	5	6	7
Number of pixels	0	0	0	6	14	5	0	0
Cumulative frequency	0	0	0	$\frac{6}{25}$	$\frac{20}{25}$	$\frac{25}{25}$	$\frac{25}{25}$	$\frac{25}{25}$
Result of multiplication	0	0	0	2	6	7	7	7

We then use this lookup table to map the gray level values of the original image, to the result.

Problem 2.1.3:
Discuss the difference between the histeq and imadjust functions in Matlab.

Solution:
The Imadjust function:
used to adjust image intensity values or colormap.

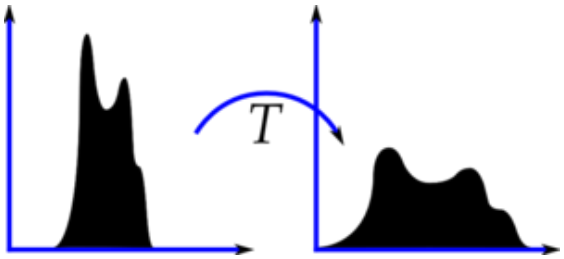
J = imadjust(I,[low_in high_in],[low_out high_out], gamma)
maps the values in intensity image I to new values in J such that values between low_in and high_in map to values between low_out and high_out. values below low_in map to low_out, and those above high_in map to high_out.
gamma specifies the shape of the curve describing the relationship between the values in I and J. If gamma is less than 1, the mapping is weighted toward higher (brighter) output values. If gamma is greater than 1, the mapping is weighted toward lower (darker) output values.
Finally, if gamma is given the value of 1, the function will result in Linear Mapping.

the size of 256, with each cell "i" of it having the frequency count of grayscale value "i-1".
In the end we can plot the this in 2D space with the help of a function.

Problem 2.1.2:
Perform Histogram Equalization for any given histogram.

Solution:
Histogram Equalization is a technique used to improve contrast in images by adjusting the image's current histogram. It accomplishes this by spreading out the grayscale intensity values.
This allows for areas of lower local contrast to gain a higher contrast, or vice-versa.
So simply put, Histogram Equalization stretchs a histogram to either ends, in order to improve the contrast of the image.

A visualization of Histogram Equalization would be as follows:



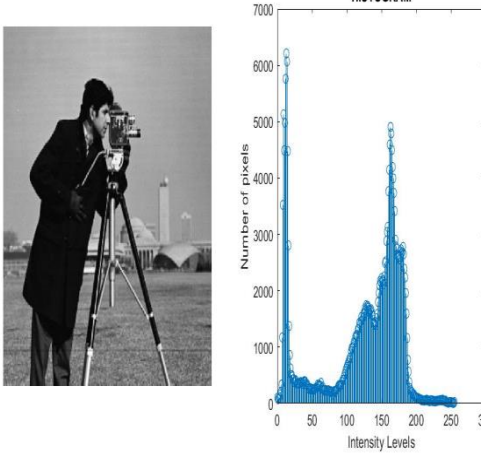
In order to perform Histogram Equalization on a given histogram:
First we calculate the cumulative frequency of all pixel values, followed by dividing the cumulative frequencies by the total number of pixels in the image, then we finally multiply them by maximum graycount (usually 255), and round the resulted number.
Performing these operations will give us a lookup table. once each of the gray levels of the original image is mapped to the corresponding resulted value, our

they don't exceed the original image's edges.

3- Describing the Results

Problem 2.1.1:

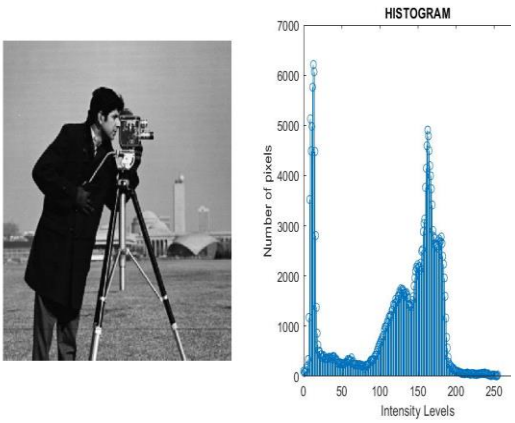
Below are my results of calculating the Histogram on the camera man image:



As seen by the visualization of the histogram, we notice that there are a high number of pixels with very low grayscale values, this is probably due to the camera man's jacket being black, we can also perceive that the intensity values between 150,175 are rather high. That is possibly due to the rather light sky that can be seen in the background of the image.

Problem 2.1.2:

As shown previously, the cameraman image with it's histogram looks like the following:



The histeq function:

Enhances contrast of an image using histogram equalization

J=histeq(I)

Performs histogram equalization on a given image I, with the output J being the image with its histogram equalized.

We can also give this function an optional second parameter "hgram", which instead of performing standard equalization, transforms the grayscale image I so that the histogram of the output grayscale image J matches the target histogram hgram.

Problem 2.2.1:

Discuss the Local Histogram Equalization algorithm, and compare it with global histogram equalization.

Solution:

Local Histogram Equalization is an algorithm which makes use of the histogram equalization algorithm explained previously.

In order to perform Local Histogram Equalization we break our original image Into n*n grid (we select the value of n). one time considering the grids can overlap, and another time considering they can't, and then we perform histogram equalization for each of those grids, as if it were an entire seperate image, finally we concatenate the resulted grids in the order of the original image.

in order to implement the algorithm correctly,we make n*n grids starting from the top left corner, moving horizontally, until we reach the image's edge, and then go to the next row. Note that on the final grid of each row as well as all the grids on the final row we modify the grid sizes to make sure

This is due to matlab using a more advanced method for its histeq function than the algorithm I discussed and implemented.

Problem 2.1.3:

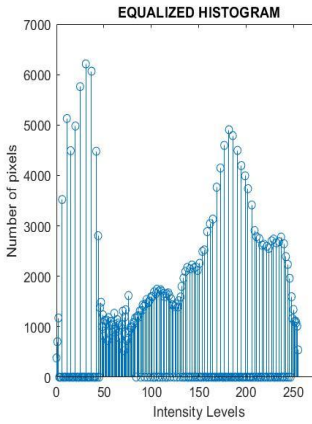
if we perform imadjust by only giving the input image as a parameter, it gives us an output image where it saturates the bottom 1% and the top 1% of all pixel values. This operation increases the contrast of the output image.

as stated before, if we only give the input image as a parameter to the histeq function, it returns the image with its histogram equalized.

this is the original camera man image:

And here are the results of performing histeq and imadjust:

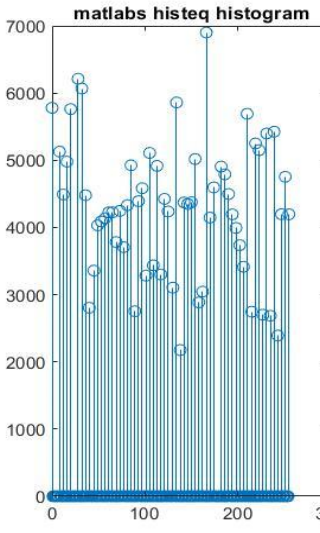
Now If we perform histogram equalization with the algorithm I explained above, the result will look like this:



As we can compare this histogram to the original histogram, we can see that the frequency of the grayscale intensity values have now been more spread out. If we compare both pictures, the light sky has now become darker, and the camera man's jacket has now become slightly more brighter.

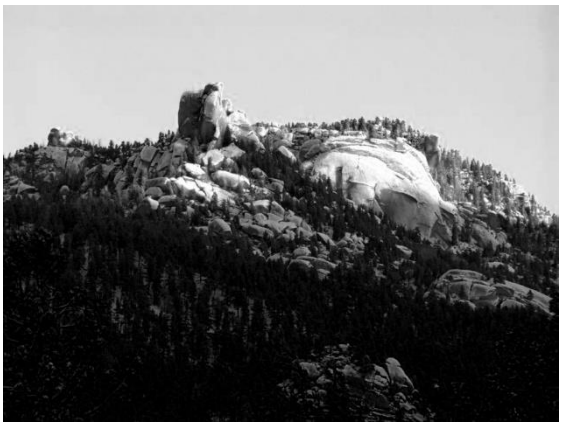
When I compared my equalized result with matlabs histeq function, the results of the images look pretty much the same,

However the histogram resulted by matlab's histeq function looks completely different from the one that I computed with my algorithm. Here is the equalized result using histeq:



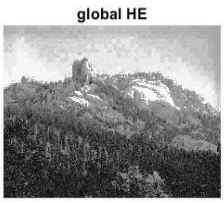
~~ Non-Overlapping ~~

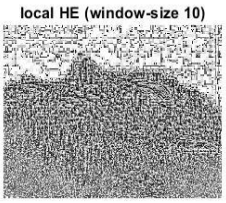
HE1 Original Image



imadjust function

histeq function





As seen in this example, if we pick a very small window size for the non-overlapping method, we'll get a noisy image, which we'll mostly only be able to realize the edges of the original image.

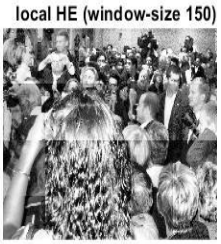
HE2 Original Image

It can be seen that the imadjust function gives a very clean output. This is because the original image has a low contrast, and imadjust addresses this by giving it an overall higher contrast (and in turn, higher quality) image. However, histeq doesn't seem to help us as much. This is due to equalization algorithm still giving us low/high contrast in some points.

Problem 2.2.1:

There are two ways in which we can implement the algorithm for local histogram equalizartion. The first method allows overlapping in the grids we make, while the second method doesn't allow overlapping of the grids.

First I'll explain the results for the Non-overlapping method:



We see as the window size increases, the consistency throughout the image increases, however here there are still visible lines which separate the grids, still due to the grayscale intensity values not exactly being on the same scale for each grid.

HE4 Original Image

in this example, we see that increasing the window size results in an image which isn't as noisy as before, however, isn't close to consistant. This is because we still have small grids, with various different grayscale intensities, which in turn histogram equalization gives a variety of different results based on the grid it's being applied to.

HE3 Original Image

In this example the overlapping method manages to return the edges of the original image with a higher contrast, while the rest of the image has a lower contrast

4- Appendix(code)

Problem 2.1.1:

Here is the function that I wrote which Returns the histogram of input image “img”

```
function gs_frequency_array = histOfImage(img)

    [X, Y] = size(img);
    % we Create a grayscale frequency array
    gs_frequency_array = 1 : 256;
    count = 0;

    % we Iterate over grayscale image matrix for all possible intensities,
    % and then count them, to finally put them in gs_frequency

    for intensity_val = 1 : 256
        for x = 1 : X
            for y = 1 : Y

                % if image pixel value at location (x, y) is equal to
                % intensity_val-1, then we increment its count
                if img(x, y) == intensity_val-1
                        count = count + 1;
                end
            end
        end
        % update ith position of frequency array with count, then reset count
        gs_frequency_array(intensity_val) = count;
        count = 0;

    end

end
```

Here is the code I use to call this function

```
clear;
img = imread('C:\Users\Arya\Desktop\computer vision\HW\Images\2\Camera Man.bmp');
figure(1);
subplot(1,2,1)
imshow(img);


freq_arr = histOfImage(img);

n = 0 : 255;

% Display Histogram
subplot(1,2,2)
stem(n, freq_arr);
title('HISTOGRAM');
ylabel('Number of pixels');
xlabel('Intensity Levels');
```

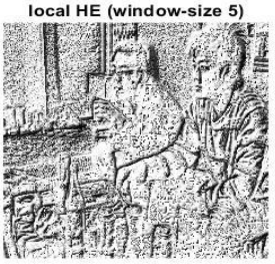
Problem 2.1.2:

I implemented the histogram equalization algorithm with the following code

Although we achieved that as we increase the window size, the consistency throughout the result image increases, however here the resulted imaged is still quiet a mess. This is due to the fact that the upper half of the original image has a very high contrast, while the other half has a low contrast.

~~ Overlapping ~~

In this method I managed to only get results for very small window sizes, because if we take a large window size, the algorithm will take a very long time to compute the resulted image



Problem 2.2.1:

the code which is used to perform local histogram equalization without overlap:

```
subplot(1,2,1)
imshow(histogramEqualization(img));
title("global HE");
[X, Y] = size(img);
WINDOW_SIZE =225 ;
matrix_rows = WINDOW_SIZE;
matrix_cols = WINDOW_SIZE;

for ii = 1 : matrix_rows : X
    for jj = 1 : matrix_cols : Y
        % Get the block
        row_begin = ii;
        row_end = min(X, ii + matrix_rows); %we put the min here so we don't go out of bounds
        col_begin = jj;
        col_end = min(Y, jj + matrix_cols); %we put the min here so we don't go out of bounds
        blk = img(row_begin : row_end, col_begin : col_end, :);
        % Perform histogram equalization with the block stored in blk
        O = histogramEqualization(blk);
        % the output of this is in O, concatenate this with the other grids
        out(row_begin : row_end, col_begin : col_end, :) = O;
    end
end
subplot(1,2,2)
imshow(out);
title("local HE (window-size " + WINDOW_SIZE + ")")
```

In order to make the grids have overlapping, all that's needed is a minor change in the for loop so that it doesn't take steps to skip overlapping regions:

```
for ii = 1 : X
    for jj = 1 : Y
        % Get the block
        row_begin = ii;
        row_end = min(X, ii + matrix_rows); %we put the min here so we don't go out of bounds
        col_begin = jj;
        col_end = min(Y, jj + matrix_cols); %we put the min here so we don't go out of bounds
        blk = img(row_begin : row_end, col_begin : col_end, :);
        % Perform histogram equalization with the block stored in blk
        O = histogramEqualization(blk);
        % the output of this is in O, concatenate this with the other grids
        out(row_begin : row_end, col_begin : col_end, :) = O;
    end
end
```

(note that it also makes use of the previously created histOfImage function)**:**

```
function [output_img] = histogramEqualization(img)

[X, Y] = size(img);


freq_arr = histOfImage(img);
total_pixels = sum(freq_arr); %the total number of pixels we have for all intensity values

pdf = freq_arr/total_pixels;
cdf=cumsum(pdf);
lut = round(cdf*255);

output_img = zeros(size(img));
for x=1:X                           %loop tracing the rows of image
    for y=1:Y                       %loop tracing thes columns of image
        t=(img(x,y)+1);             %pixel values in image
        output_img(x,y)=lut(t);     %Making the ouput image using cdf as the transformation fur
    end
end
output_img = uint8(output_img);
end
```

and the code where I make use of this function:

```
freq_arr = histOfImage(img);

n = 0 : 255;


new_img = histogramEqualization(img);
subplot(1,2,1)
imshow(new_img);
freq_arr2 = histOfImage(new_img);

subplot(1,2,2)
stem(n,freq_arr2);
title('EQUALIZED HISTOGRAM');
ylabel('Number of pixels');
xlabel('Intensity Levels');
```

Problem 2.1.3:

The code I used to fiddle with the histeq and imadjust functions:

```
close;
% Read the image
a=imread('C:\Users\Arya\Desktop\computer vision\HW\Images\2\Camera Man.bmp')
imdjst = imadjust(a);
hst = histeq(a);
figure(2);
subplot(2,1,1)
imshow(imdjst);
title('imadjust function');
subplot(2,1,2)
imshow(hst);
```