POINT AND LOCAL OPERATORS

OCTOBER 19, 2021

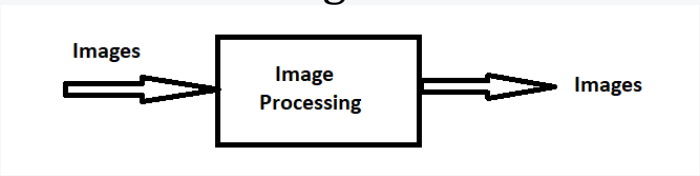
Submitted to: TA's Muhammed Zemzemoğlu and Mehmet Emin Mumcuoğlu

Name of Student: Moses Chuka Ebere
Student Number: 31491
Course: Computer Vision (EE 417)

TABLE OF CONTENTS

INTRODUCTION

In computer vision, it is often common for images to require some form of preprocessing before they can be analyzed. This image processing (or preprocessing) usually involves an image-to-image transformation as shown in the figure below:



After this, image analysis could be comfortably carried out.

The aforementioned image processing comes in handy when removing noise from images, filtering out high-frequency information, etc.

Point and local operators are some of the "tools" by which images are processed. Point operators, as the name implies, deals only with points in images, which are more accurately referred to as pixels (the smallest addressable element in images). By extension, this means that only the intensity of the pixel will be utilized (in different ways) in the processing algorithm. It is cardinally important to add that point operators only transform pixel intensities **linearly or non-linearly**. Some point operators include histogram transforms, linear scaling, conditional scaling, etc.

Local operators, on the other hand, involve the contribution of multiple pixels, specifically neighboring pixels in the processing algorithm. Common examples of local operators are local mean (or Box filter), local maximum, local minimum, etc.

This lab focuses on writing codes in MATLAB that make use of the above-mentioned point and local operators to transform different images.

EXPLANATION OF METHODS

Point Operators

Histogram Equalization

The aim of this point operator is to get an approximate flat distribution of the intensities in an image such that it spans the full gray scale spectrum (from 0 to 255). In other words, the gradation function uses 'G_{max}' (the maximum possible intensity value in an 8-bit gray image) to scale the relative cumulative frequency function of the image. The gradation function is as follows:

$$g(u) = c_I(u) \cdot G_{max}$$

where $c_I(u)$ is the relative cumulative frequency function $\sum_{v=0}^{u} h_I(u)$.
P.S. *Histogram equalization is based on a nonlinear function*.

In MATLAB, the inbuilt **histeq** function was used to perform the histogram equalization. The code is as follows:

```
%The command window along with existing variables and
%function are cleared from the memory.
clc; clear all;

%The image is read from the active folder
img = imread('city.png');


%The following function is used to display the histogram
%of the original image
imhist(img);

%applying histogram equalization on the city.png
img2 = histeq(img);

%The original image and the histogram equalized image
%along with their histograms are displayed
figure
subplot(2, 2, 1)
imshow(img)
title('Original Image')
```

```
subplot(2, 2, 2)
imshow(img2)
title('Histogram Equalized image')

subplot(2, 2, 3)
imhist(img)
title('Original Histogram')

subplot(2, 2, 4)
imhist(img2)
title('New Histogram')
```

Linear Scaling

This is a point operator that uses a linear gradation function to spread all intensity values in an image across the entire gray scale from 0 to G_{max}. The gradation function is:

$$g(u) = b\,(u + a)$$

where $a = -u_{min}$ *and* $b = \frac{G_{max}}{u_{max} - u_{min}}$

In MATLAB, the following function was written for the gradation function. The function accepts an image, linearly scales it, and returns the linearly scaled image.

```
function [output] = lab1linscale(img)

% The row, column, and channels of the image are obtained
along with the cardinality of the image.
[row, col, ch] = size(img);
Card = row*col;

% This is added in case the image introduced is an RGB
%image. It functions to convert it to a gray-scale image.
if (ch == 3)
    img = rgb2gray(img);
end

%convert the image to double before performing any
%mathematical operations
```

```
img = double(img);

%find the max and min intensities
umax = max(img(:));
umin = min(img(:));

% The parameters of the gradation function are initialized
Gmax = 255;
a = -(umin);
b = Gmax / (umax - umin);

% The new image is obtained using the gradation function.
img_new = b*(img +a);

% Convert the image back to unsigned 8-bit integers.
img_new = uint8(img_new);

% Return the new image
output = img_new;

end
```

The following code calls the linear scaling function and applies it on the input image, before displaying the results.

```
%The image is read from the active folder
a = imread('city.png');

%The following syntax calls the linear scaling function.
res = lab1linscale(a);

%The original image and the linearly scaled image are
%displayed along with their histograms.
figure
subplot(2, 2, 1)
imshow(a)
title('Original Image');

subplot(2, 2, 2)
imshow(res)
```

```
title('Linearly Scaled image');

subplot(2, 2, 3)
imhist(a);
title(['Histogram of the Original Image Umin = ',
num2str(min(a(:))),' and Umax = ', num2str(max(a(:)))]);

%subplot(2, 2, 4)
imhist(res);
title(['Histogram of the Linearly Scaled Image Umin = ',
num2str(min(res(:))),' and Umax = ',
num2str(max(res(:)))]);
```

Conditional Scaling

This is another point operator that uses a linear gradation function to produce a new image having the same mean and standard deviation as another (reference) image. The gradation function is:

$$g(u) = b\,(u + a)$$

where $a = \mu_I \frac{\sigma_J}{\sigma_I} - \mu_J$ *and* $b = \frac{\sigma_I}{\sigma_J}$

In MATLAB, the following function was created for the gradation function. The function accepts two images (the image to be conditionally scaled and the reference image (for the mean and standard deviation)), and returns the conditionally scaled image.

```
function [output] = lab1condscale(j, i)

% The row, column, and channels of the images are obtained
along with the cardinality of the image.
[row, col, ch] = size(i);
Card = row*col;
[row_i, col_i, ch_i] = size(j);
Card_i = row_i*col_i;

% This is added in case the image introduced is an RGB
%image. It functions to convert it to a gray-scale image.
if (ch == 3)
```

```
    i = rgb2gray(i);
end

if (ch == 3)
    j = rgb2gray(j);
end

%convert the images to doubles before performing any
%mathematical operations
j = double(j);
i = double(i);

% Find the mean and standard deviation of each image, and
obtain the values of a and b
mi = mean(i(:));
mj = mean(j(:));
si = std(i(:));
sj = std(j(:));
a = (mi*(sj/si)) - mj;
b = si/sj;

% Apply the gradation function
j_new = b*(j + a);

% Convert the image back to unsigned 8-bit integers and
%return the new image.
output = uint8(j_new);
end
```

The following code calls the conditional scaling function and applies it on the input image, before displaying the results.

```
% Read the two image to be used.
b = imread('city.png');
c = imread('board.jpg');

%The following syntax calls the conditional scaling
function.
re = lab1condscale(b, c);
p = rgb2gray(c);
```

```
%The reference image, the current image, and the
conditionally scaled image are displayed.
figure
subplot(2, 2, [1,2])
imshow(p)
title('Reference Image')
xlabel({['\mu = ', num2str(mean(p(:)))], ['\sigma = ',
num2str(std(double(p(:))))]})


subplot(2, 2, 3)
imshow(b)
title('Current image')
xlabel({['\mu = ', num2str(mean(b(:)))], ['\sigma = ',
num2str(std(double(b(:))))]})


subplot(2, 2, 4)
imshow(re)
title('Resulting Image')
xlabel({['\mu = ', num2str(mean(re(:)))], ['\sigma = ',
num2str(std(double(re(:))))]})
```

Local Operators

Box Filter (Local Mean)

This is a local operator that uses a window of size (2k+1) x (2k+1) to scan an entire image in order to produce a new image whose pixel values are the mean values of the intensities of the neighboring pixels in the original image. The mean of the neighboring pixels in each window is calculated with the following:

$$\mu_{W_p(I)} = \frac{1}{(2k+1)^2} \cdot \sum_{i=-k}^{+k} \sum_{j=-k}^{+k} I(x+i, y+j)$$

In MATLAB, the following function was written. The function accepts an image and a number (for the window size), and returns a box filtered image.

```
function [output] = lab1locbox(img, k)

% The row, column, and channels of the image are obtained
%along with the cardinality of the image.
[r, c, ch] = size(img);
Card = r*c;

% This is added in case the image introduced is an RGB
%image. It functions to convert it to a gray-scale image.
if (ch == 3)
    img = rgb2gray(img);
end

%Convert the image to double before performing any
%mathematical operations
I = double(img);

% Use a for-loop to create a window for scanning the
image.
for i=(k+1):1:r-k
    for j=(k+1):1:c-k
        wp = I(i-k:i+k, j-k:j+k);
        Inew(i,j) = mean(wp(:));
    end
end

% Convert the box filtered image to unsigned 8-bit images
%and return the result.
output = uint8(Inew);

end
```

The following code calls the local mean filter function and applies it on the input image, before displaying the results.

```
% Read the image to be preprocessed, and initialized k
%(used to determine the window size).
d = imread('jump.png');
e = 5;

% This syntax calls the box filter function
```

```
r = lab1locbox(d, e);

%The original image and the box filtered image are
%displayed.
figure
subplot(2, 1, 1)
imshow(d)
title('Original Image')

subplot(2, 1, 2)
imshow(r)
title(['Box Filtered image, k = ', num2str(e)])
```

Local Max and Local Min

These are local operators that use a window of size (2k+1) x (2k+1) to scan an entire image in order to produce a new image whose pixel values are the max/min intensities of the neighboring pixels in the original image. The max and min of the neighboring pixels in each window are calculated with the following:

$$J(p)_{max} = \max\{I(x+i, y+j) : -\mathrm{k} \leq \mathrm{i} \leq \mathrm{k} \wedge -\mathrm{k} \leq \mathrm{j} \leq \mathrm{k}\}$$

$$J(p)_{min} = \min\{I(x+i, y+j) : -\mathrm{k} \leq \mathrm{i} \leq \mathrm{k} \wedge -\mathrm{k} \leq \mathrm{j} \leq \mathrm{k}\}$$

In MATLAB, the following function was written. The function accepts an image and a number (for the window size) and returns two new images – a local max filtered image and a local min filtered image.

```
function [output1,output2] = lab1locmaxmin(img, k)

% The row, column, and channels of the image are obtained
%along with the cardinality of the image.
[r, c, ch] = size(img);
Card = r*c;

% This is added in case the image introduced is an RGB
%image. It functions to convert it to a gray-scale image.
if (ch == 3)
```

```
    img = rgb2gray(img);
end

%Convert the image to double before performing any
%mathematical operations
I = double(img);

% Use a for-loop to create a window for scanning the
%image.The window must be created wrt the reference point
%which will be (k+1)pixels away from the top or left and
%it will go up to k pixels less than the column or row.
for i=(k+1):1:r-k
    for j=(k+1):1:c-k
        %the window will go from -k to +k
        wp = I(i-k:i+k, j-k:j+k);
        Imax(i,j) = max(wp(:));
        Imin(i,j) = min(wp(:));
    end
end

% Convert the resulting images to unsigned 8-bit images
%and return the results.
output1 = uint8(Imax);
output2 = uint8(Imin);

end
```

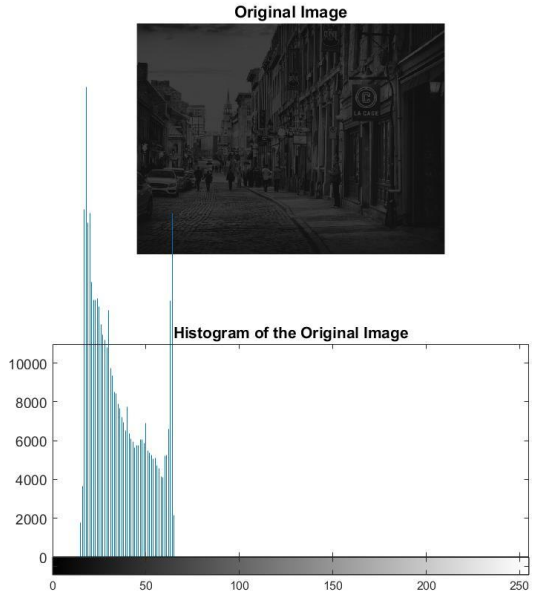
The following code calls the local max/min function and applies it on the input image, before displaying the results.

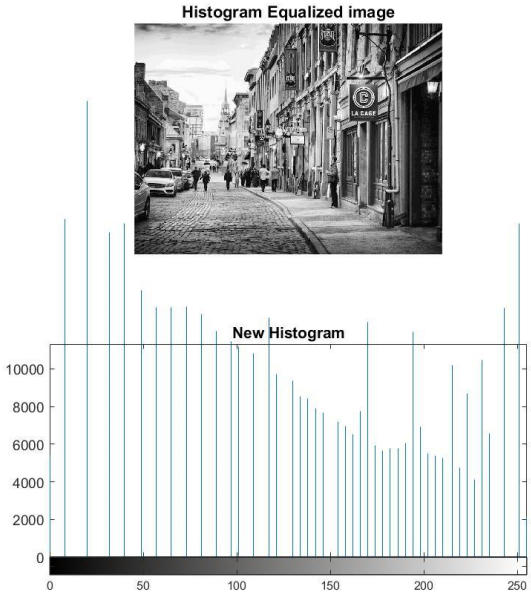
```
% Read the image to be preprocessed, and initialized k
%(used to determine the window size).
f = imread('currentimage.png');
g = 3;

This syntax calls the local max/min filter function
[resu, sur] = lab1locmaxmin(f, g);

%The original image and the local max & local min filtered
images are displayed.
figure
```

```
subplot(1, 3, 1)
imshow(f)
title('Original Image')

subplot(1, 3, 2)
imshow(resu)
title(['Local Max Filtered image, k = ', num2str(g)])

subplot(1, 3, 3)
imshow(sur)
title(['Local Min Filtered image, k = ', num2str(g)])
```

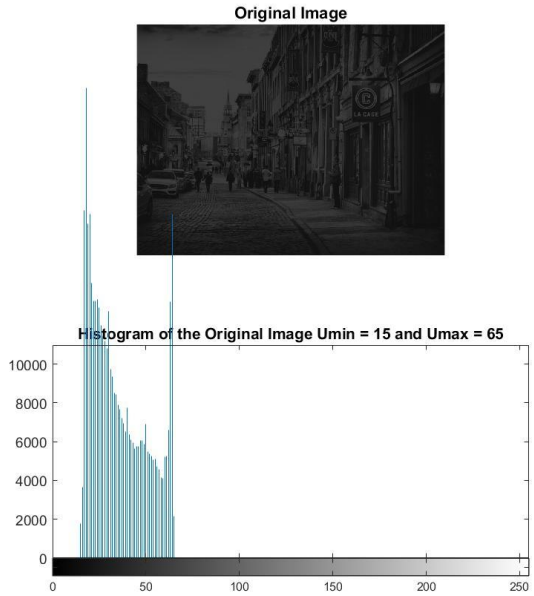
RESULTS

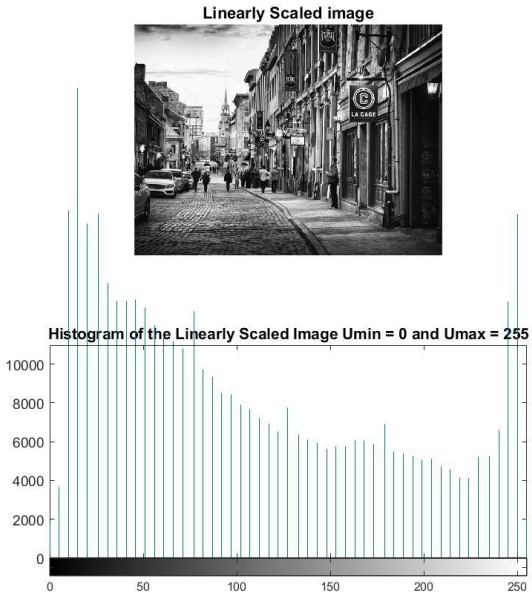
Histogram Equalization





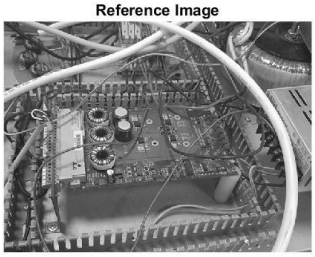
Linear Scaling





Conditional Scaling

Reference Image



μ = 116.5072
σ = 49.1187

Current image

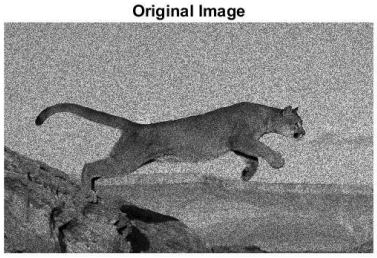
μ = 35.7225
σ = 15.0345

Resulting Image

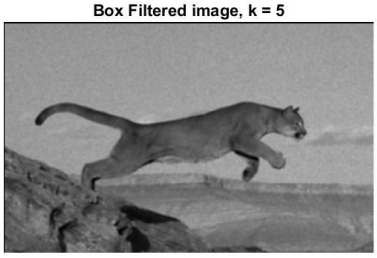
μ = 116.5056
σ = 49.1361

Box Filter

Original Image



Box Filtered image, k = 5



Local Max and Local Min

Original Image

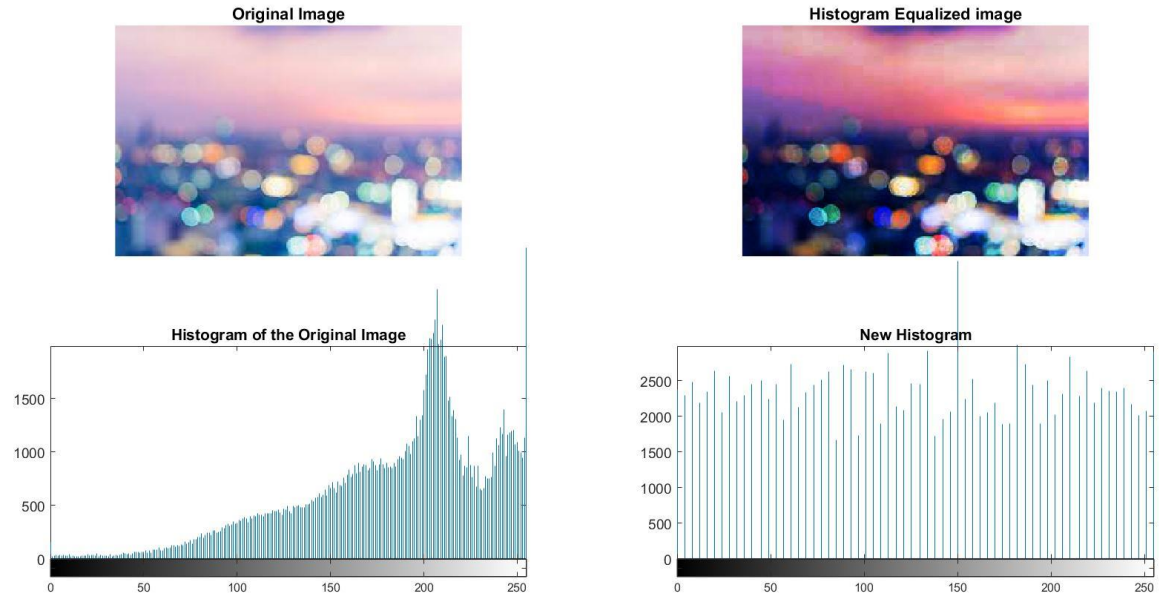
Local Max Filtered image, k = 3

Local Min Filtered image, k = 3

MULTIPLE RESULTS

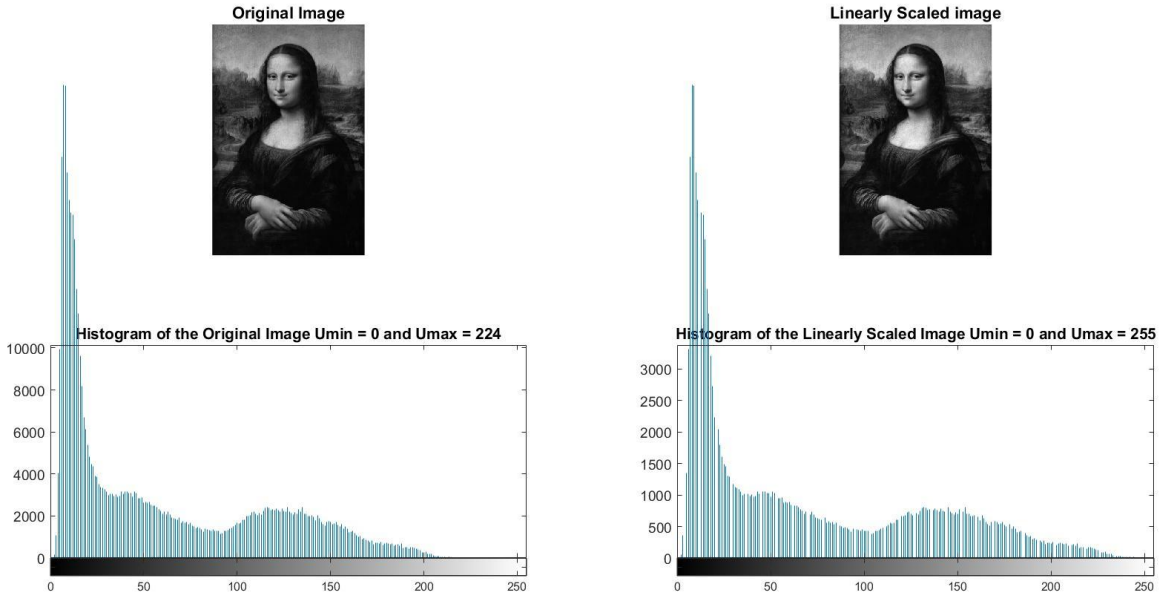
Histogram Equalization

Using a color image, let's see the outcome:



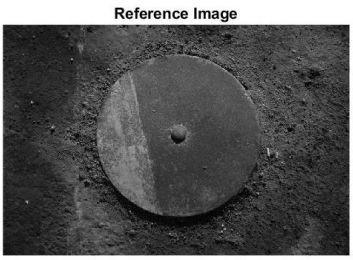
Linear Scaling

Performing linear scaling on a Grayscale image of Monalisa:



Conditional Scaling

Using another image as a reference, the above image of Monalisa was conditionally scaled below:



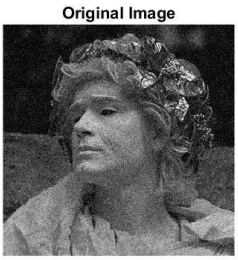
μ = 62.9331
σ = 28.2358

μ = 58.6723
σ = 54.5917

μ = 62.925
σ = 28.2412

Box Filter

Applying the box filter on a random RGB image from the internet and reducing the window size; and also working on a noisy image, the following were obtained:



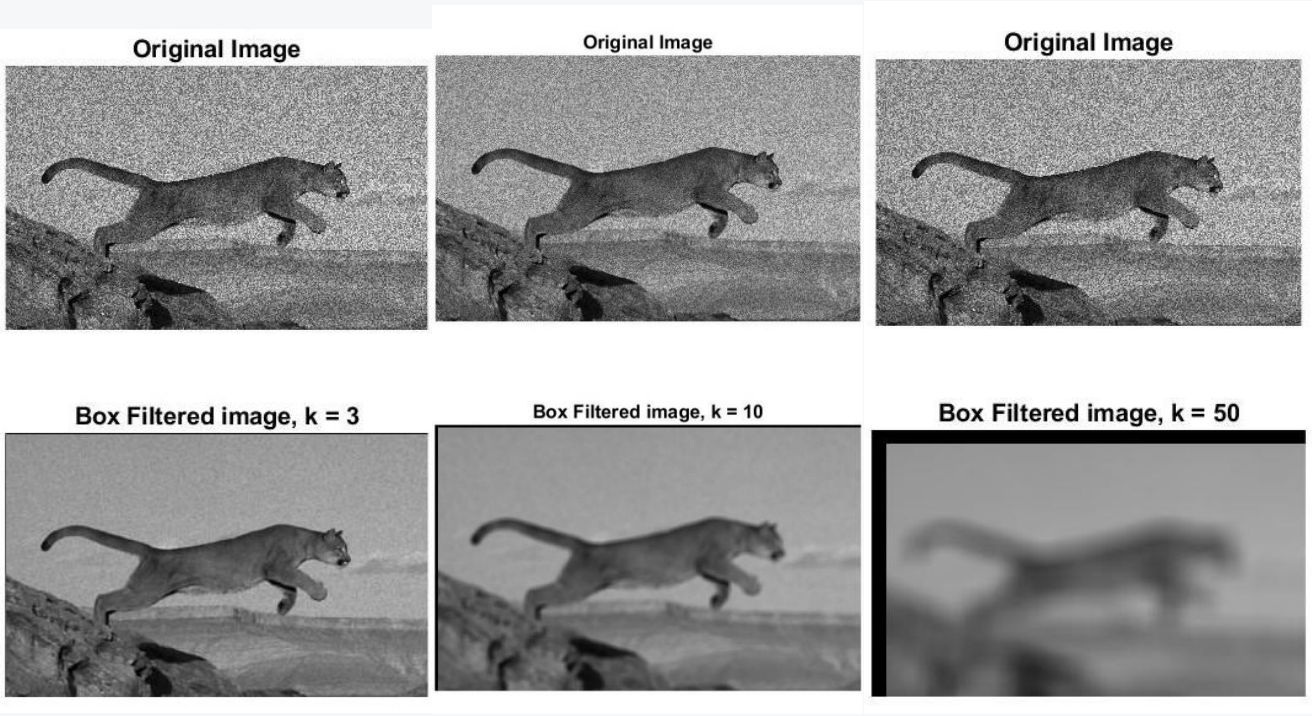
Local Max and Local Min

Using a random dark image from the web, the following results were obtained:

QUESTIONS

Box Filter

Let's the results when k = 3, 10, and 50.



We immediately notice that the larger the window size, the blurrier the resulting image. Therefore, we can make a temporary conclusion that the optimal window size is likely to be small.

Another observation is that, by increasing the window size, we increase the size of the black border that appears at the left and top of the image. This is due to the code syntax implemented.

- It's also a prime example of border-pixel strategy.

Local Max and Local Min

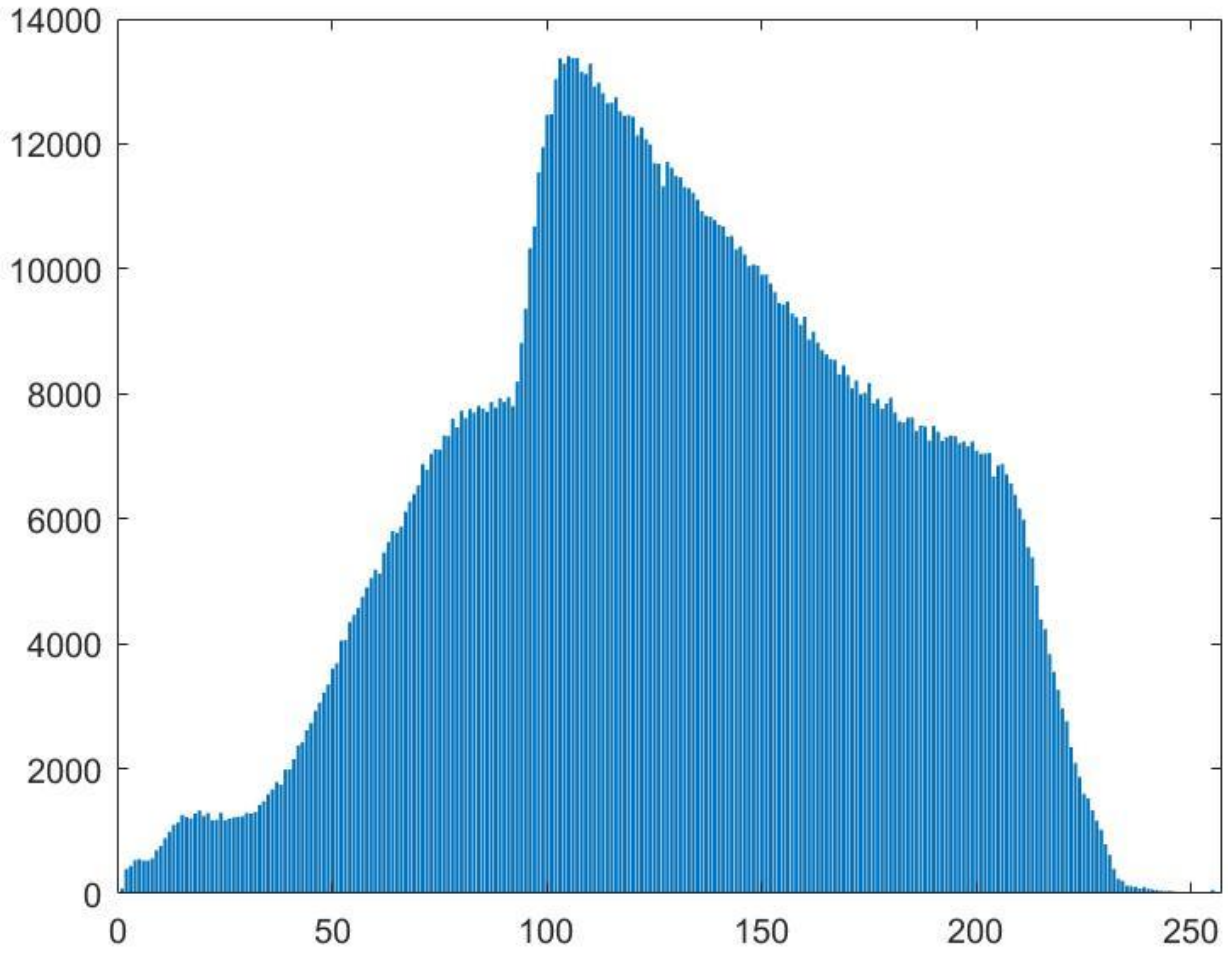
When k = 0, we obtain the same image. This is a confirmation of the fact the at k = 0, our local operator acts like a point operator since we are only working on one pixel at a time.

A Program to Calculate and Plot the Histogram of Given Image

```
function [output] = Calc_histogram(q)

% The row, column, and channels of the image are obtained
along with the cardinality of the image.
[r, c, ch] = size(q);
Card = r*c;

% Create a 256x1 matrix since we are dealing with gray
scale image (8 bits)
% This will be used for the histogram H1(u)
H = ones(256,1);

for i=1:1:r
    for j=1:1:c
        % Initialize a new variable with the intensities at
each pixel of
        % the image
        u = q(i,j);
        % Since the intensities of a gray scale image go
from 0 to 255, it
        % will be help to increment each of the by 1 to
enable us use the
        % indices of H directly
        H(u + 1)=H(u + 1) + 1;
    end
end

% Return the histogram array
output = H;
end
```

Calling the function:

```
% Read in the image.
q = imread('jump.png');

% Plot the histogram of the image
bar(H);
```



A Program that Returns the Histogram Equalized Image

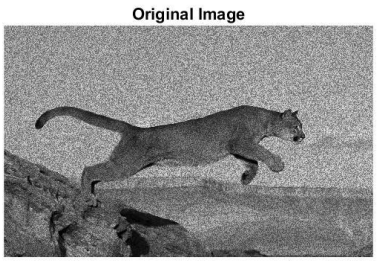
```
function [output] = Calc_histogram_eq(q)

% The row, column, and channels of the image are obtained
along with the cardinality of the image.
[r, c, ch] = size(q);
Card = r*c;

% Create a 256x1 matrix since we are dealing with gray
scale image (8 bits)
% This will be used for the histogram H1(u)
H = ones(256,1);

for i=1:1:r
    for j=1:1:c
```

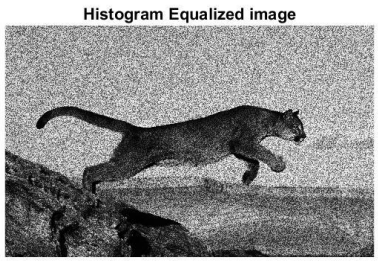
```
        % Initialize a new variable with the intensities at
each pixel of
        % the image
        u = q(i,j);
        % Since the intensities of a gray scale image go
from 0 to 255, it
        % will be help to increment each of the by 1 to
enable us use the
        % indices of H directly
        H(u + 1)=H(u + 1) + 1;
    end
end


% To write some code for the histogram equalization, we
need to obtain the
% parameters in our gradation function, g(u) = c1(u).Gmax;
where c1(u) is
% our relative cumulative histogram, i.e. summation of
h1(u). And h1(u) =
% H1(u)/cardinality

%Let's find h1(u)
h1 = H./Card;

%Create variables for c1 and the gradation function g(u)
c1 = zeros(256,1);
g_u = zeros(256,1);
sum = 0;

% Use a for-loop to find the relative cumulative histogram
for i=1:1:256
    sum = sum+H(i);
    c1(i) = sum/Card;
    g_u(i) = round(c1(i)*255);
end

% Now, for the histogram equalization, we can make use of
the gradation
% function, g_u, above.

histogram_eq = uint8(zeros(r,c));
```

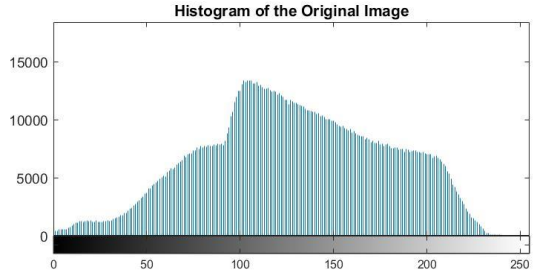
```
for i=1:1:r
    for j=1:1:c
        histogram_eq(i,j) = g_u(q(i,j)+1);
    end
end

% Return the histogram equalized image
output = histogram_eq;

end
```

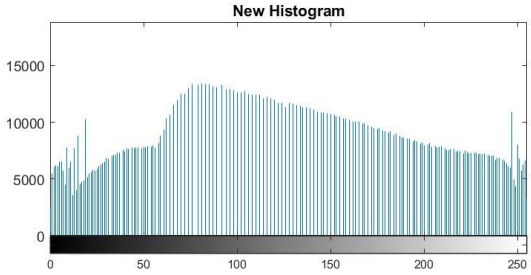
Calling the function:

```
% Read in the image.
q = imread('jump.png');

% Plot the histogram of the image
bar(H);
A = Calc_histogram_eq(q);

figure
subplot(2, 2, 1)
imshow(q)
title('Original Image')

subplot(2, 2, 2)
imshow(A)
title('Histogram Equalized image')

subplot(2, 2, 3)
imhist(q)
title('Histogram of the Original Image')

subplot(2, 2, 4)
imhist(A)
title('New Histogram')
```


Original Image


Histogram Equalized image


Histogram of the Original Image
15000
10000
5000
0
0
50
100
150
200
250

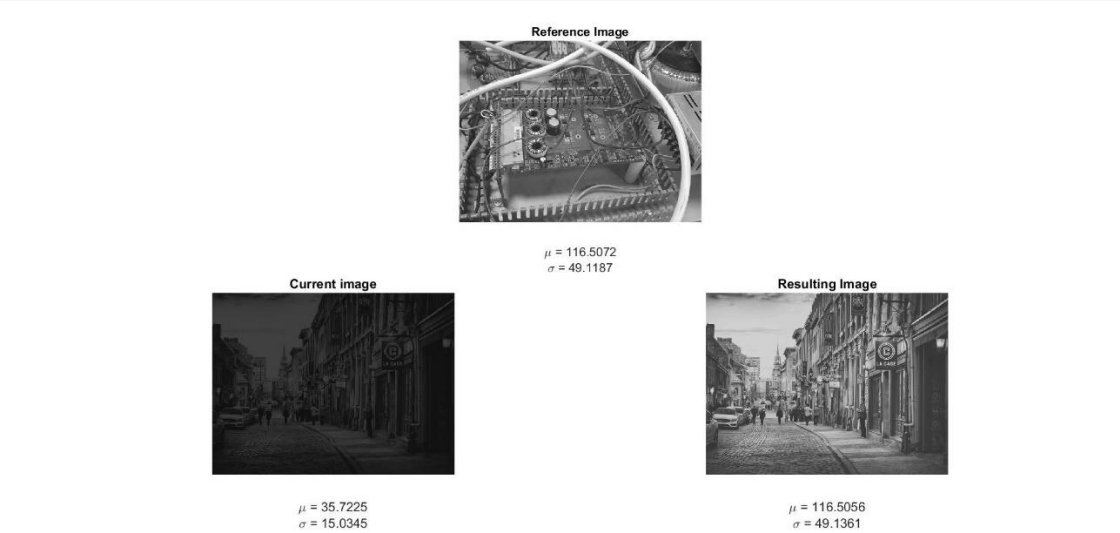

New Histogram
15000
10000
5000
0
0
50
100
150
200
250

DISCUSSION

- By visual inspection, the histogram equalization and the linear scaling seem to produce very similar results; however, on looking at their histograms, we notice differences. These differences are partly because histogram equalization employs a nonlinear gradation function, while linear scaling is based on a linear gradation function.

- In both linear scaling and histogram equalization, we end up spreading the histogram to occupy the full scale from 0 to G_{max}.

- An important observation with the conditional scaling is that the resulting image does not have **exactly** the same mean and standard deviation. There are minor differences that could be overlooked as shown below:



We can see that the parameters for our reference image were $\mu = 116.5072$; $\sigma = 49.1187$; while for the new image, they are: $\mu = 116.5056$; $\sigma = 49.1361$.

- Why is this the case? My guess is that this results from the conversion from uint8 to double and back.

- For the box filter and local max/min filters, we were forced to choose a border-pixel strategy. In this case, we opted to scan the image from (k+1) from the top and left. This resulted in a black L-

shaped area in our final images. Other border-pixel strategies could be applied for slightly different results.

- From our implementation of the local mean, we clearly see that it's a relatively good method for getting rid of noise in images.

REFERENCES

Reinhard Klette. 2014. Concise Computer Vision: An Introduction into Theory and Algorithms. Springer Publishing Company, Incorporated.

https://www.mathworks.com/help/matlab/ref/xlabel.html#btpmg0w-7

https://onlinepngtools.com/convert-png-to-grayscale

https://www.revolutiondatasystems.com/blog/grayscale-or-bitonal-which-is-a-better-method-for-scanning-my-records

https://www.pinterest.com/pin/484629609880276751/

https://unsplash.com/wallpapers/colors/dark

https://boofcv.org/index.php?title=Example_Wavelet_Noise_Removal

https://jooinn.com/bokeh-photography-effect.html