Binarization of degraded document images using morphological operators and ternary entropy-based approach

Image Processing Course Final Project *Mandana Samiei*
17/12/2017 *Id: 40059116*

PART 1

1.

a) Introduction:

This paper provides 2 adaptive binarization methods using a ternary entropy-based approach.
Image degradation can occur due to:

- bleeding-through ink
- large black border
- fading ink
- uneven illumination
- contrast variation
- smear and various pattern backgrounds

How this approach works?

Suppose that we have given an input image:
1. First, the contrast of intensity is estimated by a grayscale morphological closing operator.
2. A double-threshold is generated by the Shannon entropy-based thresholding methods corresponding to 1-D histogram and 2-D histogram to classify pixels into 3 regions:
   - text category
   - near-text category
   - non-text category
3- The pixels in the near-text group are relabeled by the local mean and the standard deviation values.
4- This method classifies noise into 2 groups:
   - which are dealt with morphological operators- shrink and swell filters
   - those are dealt with graph searching strategy

Generally, binarization techniques can be divided into two categories, namely, global thresholding and local thresholding.
Global Thresholding:

- Fixed Thresholding Binarization: this technique binarizes with respect to a specified threshold. (when the background and foreground are distinct.)
- Histogram Shape-based Binarization: this technique uses the minimum value between two peaks of the histogram as a threshold value
- Optimal Thresholding Binarization: applies a criterion function to each pixel in order to separate an image into two regions (discriminant method, entropy thresholding, moment preservation, minimum error thresholding)

Local Thresholding (Dynamic thresholding): apply a separate threshold value ($T(x, y)$) to a single pixel or a particular region.

Basically, local thresholding techniques are more effective than global ones, however, they have some disadvantages, basically they depend on the region size, have a significant computational cost and individual image characteristics. So, mostly both techniques (hybrid approach) are used in images to achieve better results.

Motivation:

- As more and more documents are digitized, less time consuming and higher accuracy in document image binarization have become increasingly necessary.
- A great number of historical and badly degraded document images can be easily found in libraries, natural and public achieves.
- Due to the complex nature of different artifact, such poor quality documents are hard to read and process.

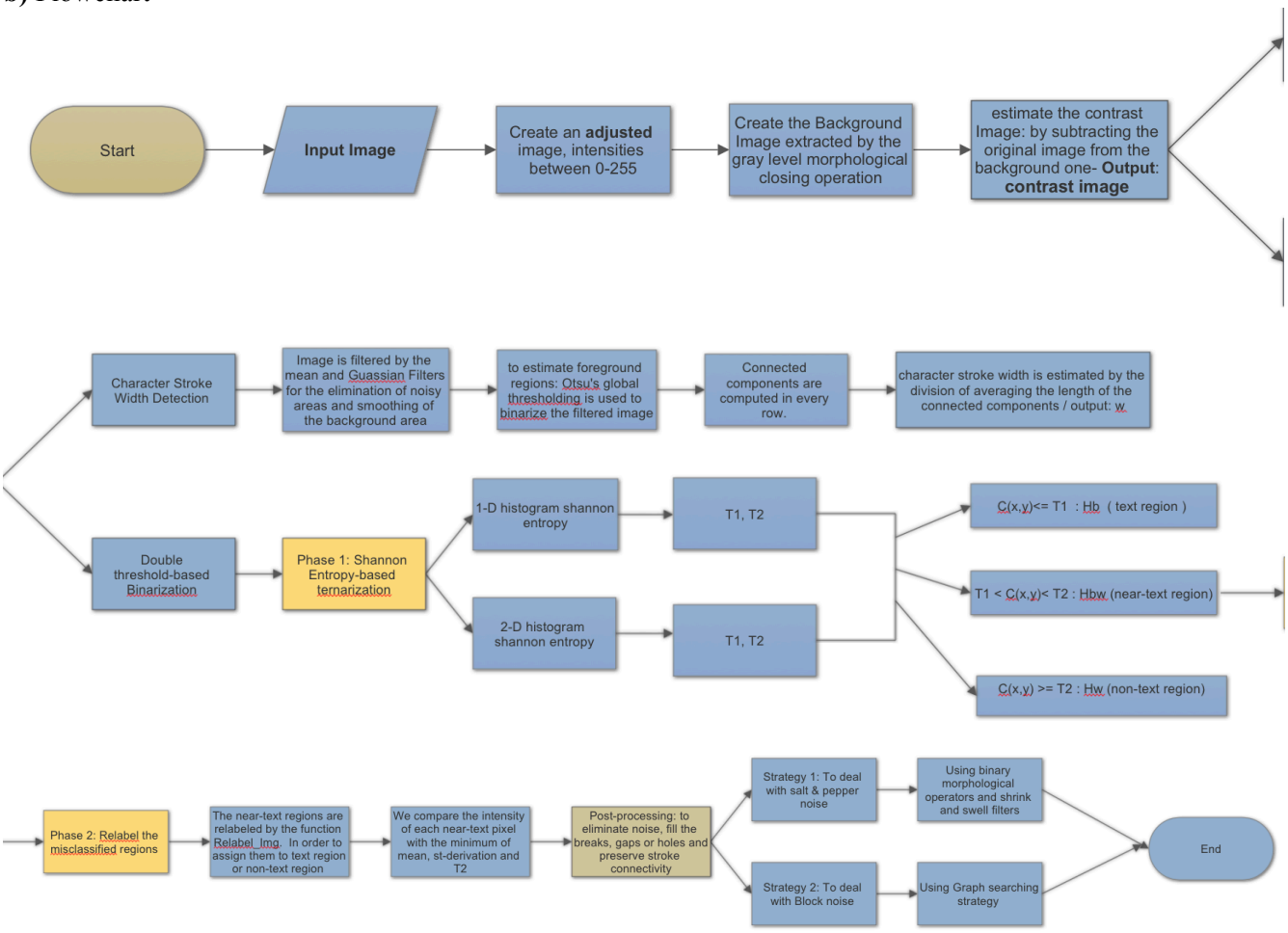
Contribution:

This approach outperforms other state-of-the-art binarization methods for degraded document images when tested on four datasets:

- DIBCO 2009
- H-DIBCO 2010
- ICHFR 2010 and
- Carlos et al.'s database

The performance is evaluated by 9 distinct measure.

b) Flowchart



c) Short explanation for each part of the flowchart

1- Contrast Estimation:

Input: A grayscale image

Output: Contrast Image

Procedure:

- Create an adjusted image with intensities between 0-255

- Create the background image extracted by the gray level morphological closing operators
- Estimate the contrast image by subtracting the original image from the background one

2- Character Stroke Width Detection

Input: The contrast Image

Output: Window w

Procedure:

- Image is filtered using the mean and Gaussian filters for the elimination of noisy areas and smoothing of the background area
- Otsu's global thresholding is used to binarize the filtered image to estimate foreground regions
- Connected components are computed in every row
- Character stroke width is estimated by averaging the length of the connected components

3- Double Threshold Binarization:

Phase 1: Shannon Entropy-based ternarization:

Input: The contrast Image

Output: T1 and T2

- 1-D histogram Shannon entropy
- 2-D histogram Shannon entropy

Procedure:

$$\begin{cases} H_b(text\ regions) & if\ C(x,y) \le T1 \\ H_{bw}(near-text\ regions) & if\ T1 < C(x,y) < T2 \\ H_w(non-text\ regions) & if\ C(x,y) \ge T2 \end{cases}$$

Phase 2: Relabel the misclassified regions

Input: Near-text regions pixels

Output: Eliminated non-character pixels image

Procedure:

- In order to assign pixels to text regions or non-text regions, the near-text regions are relabeld by the function Relabel_Img.
- We compare the intensity of each near-text pixel with the minimum of mean and standard deviation and T2.

4- Post-processing:

Input: Document text pixels

Output: The binarized image

Procedure:

There are 2 strategies to deal with 2 types of errors:

- Strategy 1: to deal with salt and pepper noise, we are using morphological operators as well as shrink and swell filters to achieve better results.
- Strategy 2: to deal with block noise, a graph searching strategy is applied.

2.

Algorithm 3 - Pre-processing (Contrast Estimation)

Input: A grayscale Image $I(x,y)$

Output: The contrast image $C(x,y)$

Procedure:

for each r: number of rows

for each c: number of columns

$J(r,c) = I(r,c)$*255/(max - min) // Adjustment

end

end

//Background Image G(x,y) = $(f \oplus SE) \odot SE$

//Define a window SE for closing operation

for each x: number of rows
for each y: number of columns
for each i: number of rows
for each j: number of columns
$G1(x,y) = Max[f(x\text{-}i, y\text{-}j) + SE(i,j)]$ //Dilation
$G2(x,y) = Min[G1(x+i, y+j) + SE(i,j)]$ //Erosion after Dilation
end
end
end
end

//Subtraction
$C(x,y) = G2(x,y)\text{-}I(x,y)$

Algorithm 4- Post-processing (Noise Elimination)

Input: The binarized image $I(x,y)$
Output: The final image without "salt & pepper" and "block" noise
Procedure:
//First we need to find the regions corresponds to "salt noise", "pepper noise" and "block noise"
//Call CalcStrokeWidth(I)function to calculate SW
//First Eliminate Salt and Pepper noise
//Morphological operators to bridge unconnected components and eliminate spurious and isolated pixels
//To correct the areas that are larger than 1 pixel and smaller than SW by using shrink and swell filters
//Define values
//Suppose that we have 2 regions R1 and R2 to iterate over the image
Num1=0; Num2=0; R1= SW, R2=R1;
//counting the number of pixels in each region

```
for each i= 0 : number of rows/R1
    for each j= 0 : number of columns/R2
        for each k = 0:R1
            for each t = 0:R2
                if I(i*R1, j*R2) == 1
                    if (k < R1 && t < R2) // inside the regions
                        Num2++;
                    end
                 Num1++;
                end
            end/end/end/end
            if Num1 == Num2
                (R1-1)*(R2-1) = 0
            end
            if Num1 == [Num2 + 2*(R1+R2-2)]
                (R1-1)*(R2-1) = 1
            end
```

// Block Noise
//Graph Searching
//Define root: a complete black block
//Define node: a block which has more than 2*W black pixels
//Define edge: 2 adjacent block pixels belong to two different adjacent node
//The binarized image is divided to non-overlapping blocks with the window size bigger than SW
//Scan image from top to bottom and left to right then
//Find root R and set it as the root of a new tree
//For each node inside the tree, called A, consider its 4- neighboring blocks, denoted by A_i
//If A_i is a node and there is an edge between A and A_i then add A_i to the tree
//After the graph searching algorithm is applied to the image, all the trees that have been detected
//are removed from the image

```
            end
        end
    end
end
```

3. Implementation: The Double Threshold Binarization Method

```
function [T1,T2,Hb,Hbw,Hw] = myDoubleThresholdFinder(I)
if ndims(I) == 3
    I= rgb2gray(I);
end

MAX=0;
T1=0;
T2=0;
L=256;
[x,y] = size(I); s=x*y;

% finding probablity of each bin
p = imhist(I)./s;

for t1=1:L
    for t2=1:L

        %Compute Cumulative Density in each region
        P_t1 = sum(p(1:t1));
        P_t1t2 = sum(p(t1:t2));
        P_t2 = sum(p(t2:L));
        Hb =0; Hbw=0; Hw=0; H=0;

        %Compute H_b
        for i=1:t1
            temp = p(i)*log(p(i)/P_t1)/P_t1;
            if isnan(temp) %sometimes that p or P_t is 0 and result is NaN
                temp=0;
            end
            Hb = Hb + temp;
        end
        %Compute H_bw
        for i=t1+1:t2
            temp = p(i)*log(p(i)/(P_t1t2))/(P_t1t2);
            if isnan(temp)  %sometimes that p or P_t is 0 and result is NaN
                temp=0;
            end
            Hbw = Hbw + temp;
        end
        %Compute H_w
        for i=t2+1:L
            temp = p(i)*log(p(i)/(P_t2))/(P_t2);
            if isnan(temp)  %sometimes that p or P_t is 0 and result is NaN
                temp=0;
            end
            Hw = Hw+temp;
        end
        H = -Hb -Hbw -Hw;
        if H>MAX
```

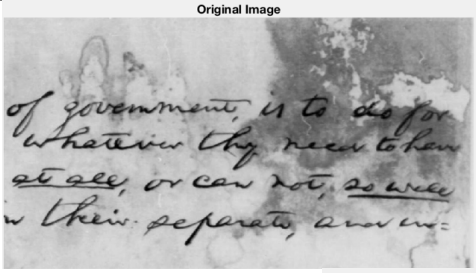
```
            MAX = H; T1=t1; T2=t2;
        end
    end
end

disp('Threshold values are ');
disp([T1, T2]);
end
```

Main Program

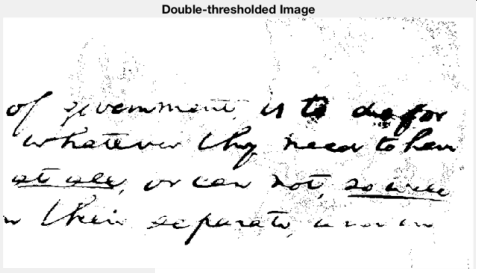
```
I = imread('3_10.jpg');
%only one dimention
if ndims(I) == 3
  I= rgb2gray(I);
end
r = size(I,1);
c = size(I,2);
J = zeros (r,c);

%using my func to find the threshold
[T1,T2,Hb,Hwb,Hw] = myThresholdFinder(I);


for i = 1: r
    for j=1: c
        if I(i,j) < T1
            J(i,j) = 0;
        elseif I(i,j) > T1 && I(i,j) < T2
            J(i,j) = 127;
        elseif I(i,j) > T2
            J(i,j) = 255;
        end
    end
end

figure
subplot(1,2,1);
imshow(I);
title('Original Image');
subplot(1,2,2);
imshow(J);
title('Double-thresholded Image');
```

Threshold values are
79 205

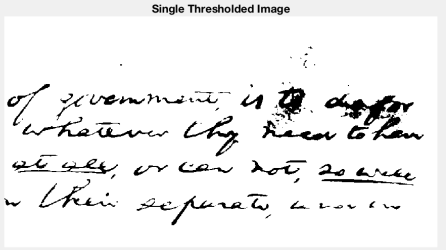
Original Image



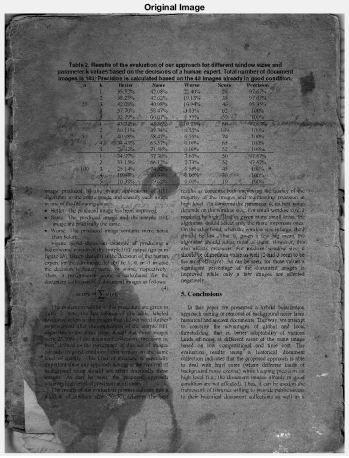
Double-thresholded Image



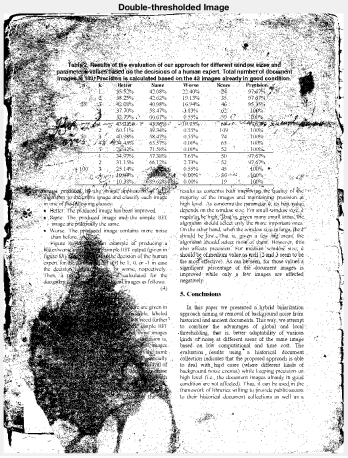
Single Thresholded Image



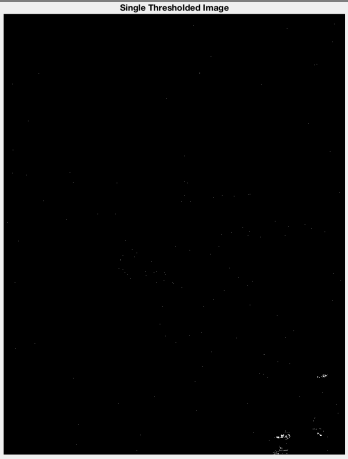
Original Image



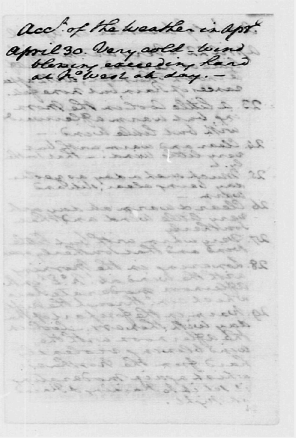
Double-thresholded Image



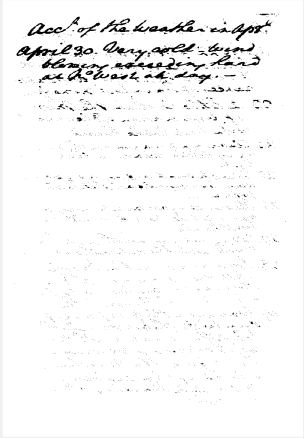
Single Thresholded Image



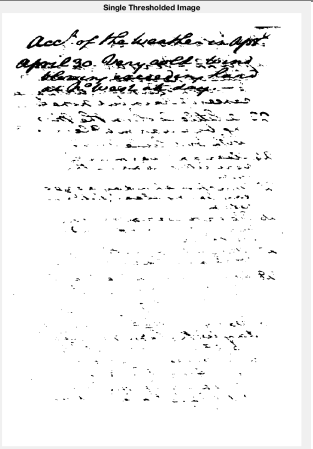
Original Image



Double-thresholded Image



Single Thresholded Image

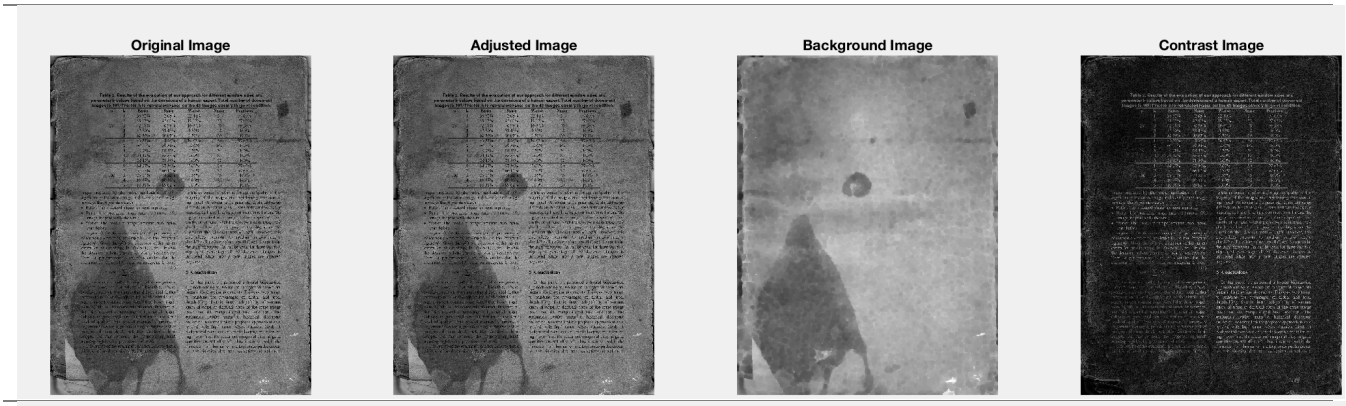


```
function I = Relabeling (I, C, T1, T2)
%Relabel the near-text pixels in the image I

M = size(I,1);
N = size(I,2);
w = CalcStrokeWidth(I);
     for i=1:M
         for j=1:N
             if C(i,j)>T2 || C(i,j)==T1
                 [Num, I_mean, I_std] = Relabel_Region(I,C,i,j,T1);
                 if I(i,j) < min(I_mean+I_std,T2) && Num>0
                     I(i,j) = 1;
                 else
                     I(i,j) = 0;
                 end
                 i = i+w;
                 j = j+w;
             end
         end
     end
end
```

```
function [ Num, I_mean, I_Std ] = Relabel_Region (I,C,i,j,T1)

Num=0; %Number of high contrast pixels (intensity > T1)
I_mean=0; %The mean of the detected high contrast pixels in the original image
I_std=0; %The standard deviation of the detected high contrast pixels in the
original image

W = CalcStrokeWidth(I);
M = size(I,1);
N = size(I,2);

    for i=W+1:M-W
        for j=W+1:N-W
            Cw = C(i-W:i+W,j-W:j+W);
            Iw = I(i-W:i+W,j-W:j+W);

            for x = 1:2*W+1
                for y = 1:2*W+1
                   if Cw(x,y) > T1 || Cw(x,y) == T1
                       Cw(x,y) = 1;
                       Num = Num +1;
                   else
                       Cw(x,y) = 0;
                   end
                end
            end

            if(Num > 0)
                I_mean = (Iw.*Cw)/Num;
                I_std = sqrt(double(Cw.*((Iw-I_mean).^2)/Num));
            end
        end
    end
end
```

4. Implementation: Pre-processing Step (Contrast Estimation)

```
%%----------Part 1 - Question 4 --------------
%%--------- Mandana Samiei--------------------
%%--------- Student Id: 40059116 -------------
%% Contrast Estimation

%I = imread('/Users/mandana/Documents/MATLAB/Final_Pro-
ject/Part1/DIBCO2009/H04.bmp');
I = imread('/Users/mandana/Documents/MATLAB/Final_Pro-
ject/Part1/ICFHR2010/3_10.jpg');
if ndims(I) == 3
   I = rgb2gray(I);
end
r = size(I,1);
c = size(I,2);
minGrayLevel = double(min(min(I))); % Finding minimum intensity in the image
maxGrayLevel = double(max(max(I))); % Finding maximum intensity in the image
% Gray level mapping to a range from 0 to 255
Slope= 255/(maxGrayLevel - minGrayLevel);
currVal = 255 - Slope * maxGrayLevel;
ContrastStretchedImg = Slope*I + currVal;

% Extract Background Using Closing morphological Operator
se = strel('disk',10);
bgImg = imclose(ContrastStretchedImg,se);

% Subtract original image from background image
ContrastImg = bgImg - I;

figure;
subplot(1,4,1);
imshow(I);title('Original Image');
subplot(1,4,2);
imshow(ContrastStretchedImg);title('Adjusted Image');
subplot(1,4,3);
imshow(bgImg);title('Background Image');
subplot(1,4,4);
imshow(ContrastImg);title('Contrast Image');
```



5. Implementation: Post-Processing Step (Noise Elimination-Morphological operation and Graph searching)

```
%%----------Part 1 - Question 5 --------------
%%--------- Mandana Samiei--------------------
%%--------- Student Id: 40059116 -------------
%%SaltPepperNoiseElimination Function

%Read an input image to workspace
I = imread('3_10.jpg');
orgI = I; %original Image
I = bwmorph(II,'spur');%apply 'spur' morphological operator to remove the spuri-
ous pixels
I = bwmorph(I,'majority');% to set a pixel to 1 if five or more pixels in its 3-
by-3 neighborhood are 1s
I = bwmorph(I,'bridge');% to bridge the unconnected pixels
I = bwmorph(I,'clean');%to remove the isolated pixels

%To correct the areas that are larger than 1 pixel and smaller than SW by
%using shrink and swell filters
I = padarray(I, 1,'symmetric','both');
SW = CalcStrokeWidth(orgI)-1;
rows = size(I,1);
cols = size(I,2);
Num1 = 0; Num2 = 0;
R1= SW; R2= R1;
J = zeros(rows,cols); %output image
region_length = floor(rows/R1);
region_width = floor(cols/R2);

for i=0:region_length-1 % the times that we need to cross the image row for each
window move
    for j=0:region_width-1 %the times that we need to cross the image column for
each window move
        Num1 = 0; Num2 = 0;
        for k=1:R1 % inside the window
            for t=1:R2
                if I((i*R1)+k,(j*R2)+t) == 1
                    if k < R1 && t < R2 % inside R1*R2
                        Num2 = Num2+1;
                    end
                    Num1 = Num1+1; % outside R1*R2
                end

            end
        end
        if Num1 == Num2
            J(i*R1+1:(i*R1)+R1-1, j*R2+1:(j*R2)+R2-1) = 0;
        elseif Num1 == Num2+2*(R1+R2-2)
            J(i*R1+1:(i*R1)+R1-1, j*R2+1:(j*R2)+R2-1) = 1;
        end
    end
end

figure
sublot(1,2,1);
imshow(II);title('Original Image');
subplot(1,2,2);
imshow(J);title('Noise Eliminated Image');
```

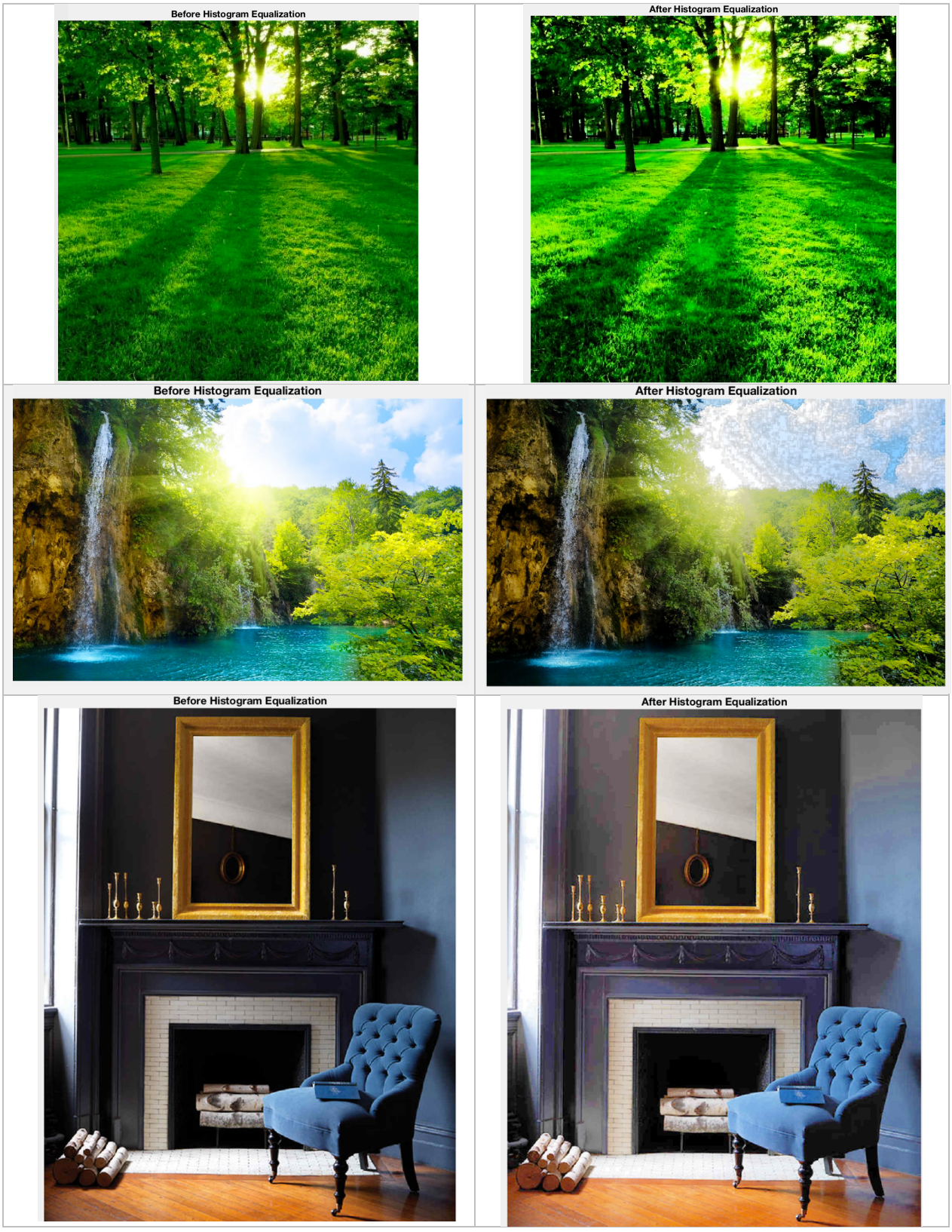
```
%%----------Part 1----------------------------
%%--------- Mandana Samiei--------------------
%%--------- Student Id: 40059116 -------------
%% Main Program

I = imread('/Users/mandana/Documents/MATLAB/Final_Pro-
ject/Part1/ICFHR2010/3_10.jpg');

% contrast estimation
C = ContrastEstimatation(I);

% apply double threshold binarization to obtain binary image
[T1,T2,Hb,Hbw,Hw] = myDoubleThresholdFinder(I);

% relabel the near-text pixels in the image
I2 = Relabeling(I,C,T1,T2);

% apply post prcessing steps to remove noise in image
% strategy1: removing salt and pepper noise
I3 = SaltPepperNoiseEliminitaion(I2);

figure;
subplot(1,4,1);
imshow(I);title('Original Image');
subplot(1,4,2);
imshow(C);title('Contrast Image');
subplot(1,4,3);
imshow(I2);title('Double tresholded Image');
subplot(1,4,4);
imshow(I3);title('Noise Eliminated Image');
```

PART 2

1. RGB image matrix is converted into HSI (Hue, Saturation and Intensity) format and histogram equalization is applied only on the Intensity matrix. The Hue and Saturation matrix remains the same. The updated HSI image matrix is converted back to RGB image matrix.

```
function RGB  = ColorHisteq( I )
%COLOR HISTOGRAM EQUALIZATION

% Read the input image
HSV = rgb2hsv(I);
% Convert the RGB image to HSV image format
Heq = histeq(HSV(:,:,3));

% Perform histogram equalization on intensity component
HSV_mod = HSV;
HSV_mod(:,:,3) = Heq;

%Convert the resulting image back into the RGB color space
RGB = hsv2rgb(HSV_mod);
end
```

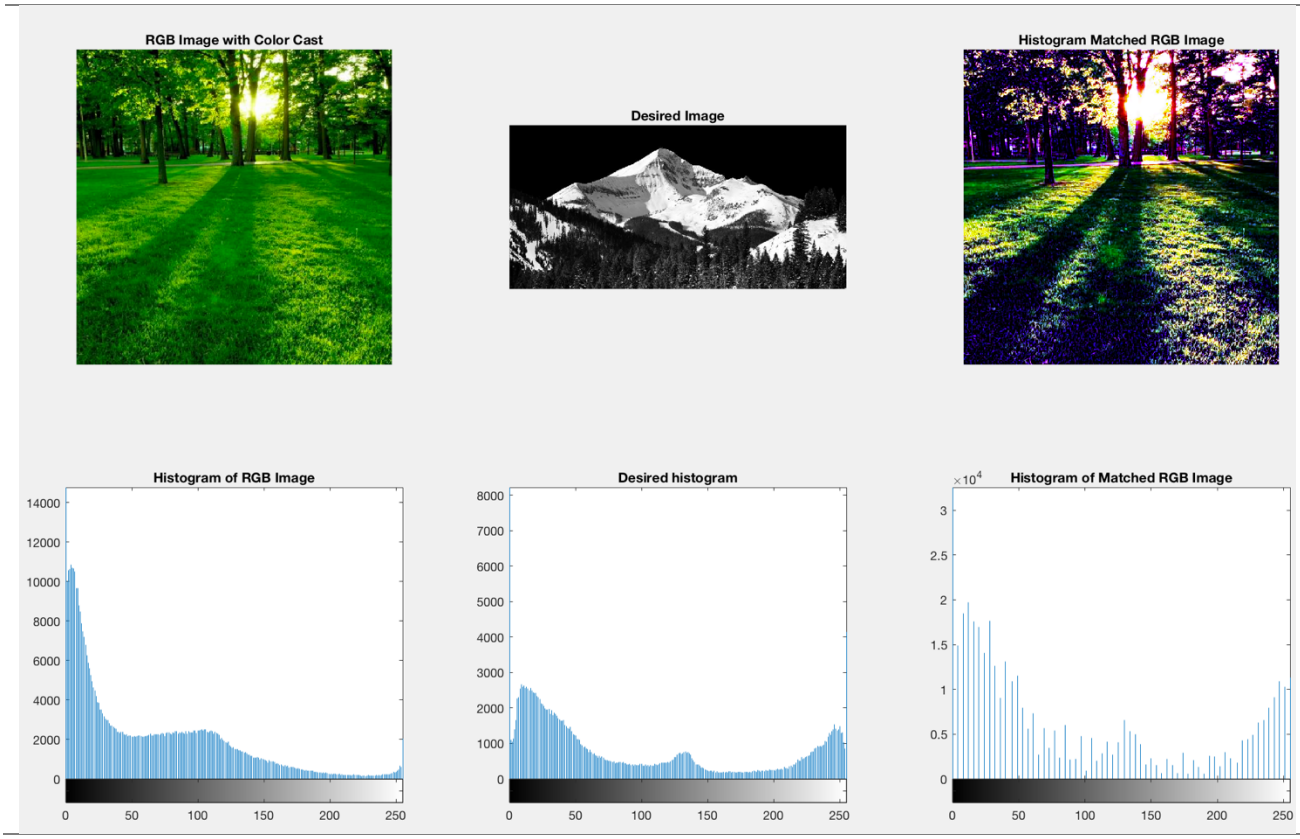
```
%%----------Part 2 - Question 1 --------------
%%--------- Mandana Samiei--------------------
%%--------- Student Id: 40059116 -------------
%%Read the color image into the workspace. test with 3 images

I1 = imread('Forest1.jpg');
I2 = imread('Nature1.jpg');
I3 = imread('Room.jpg');

%then convert the RGB image into the HSV color space.
RGB1 = ColorHisteq(I1);
RGB2 = ColorHisteq(I2);
RGB3 = ColorHisteq(I3);

figure
subplot(3,2,1),imshow(I1);title('Before Histogram Equalization');
subplot(3,2,2),imshow(RGB1);title('After Histogram Equalization');
subplot(3,2,3),imshow(I2);title('Before Histogram Equalization');
subplot(3,2,4),imshow(RGB2);title('After Histogram Equalization');
subplot(3,2,5),imshow(I3);title('Before Histogram Equalization');
subplot(3,2,6),imshow(RGB3);title('After Histogram Equalization');
```

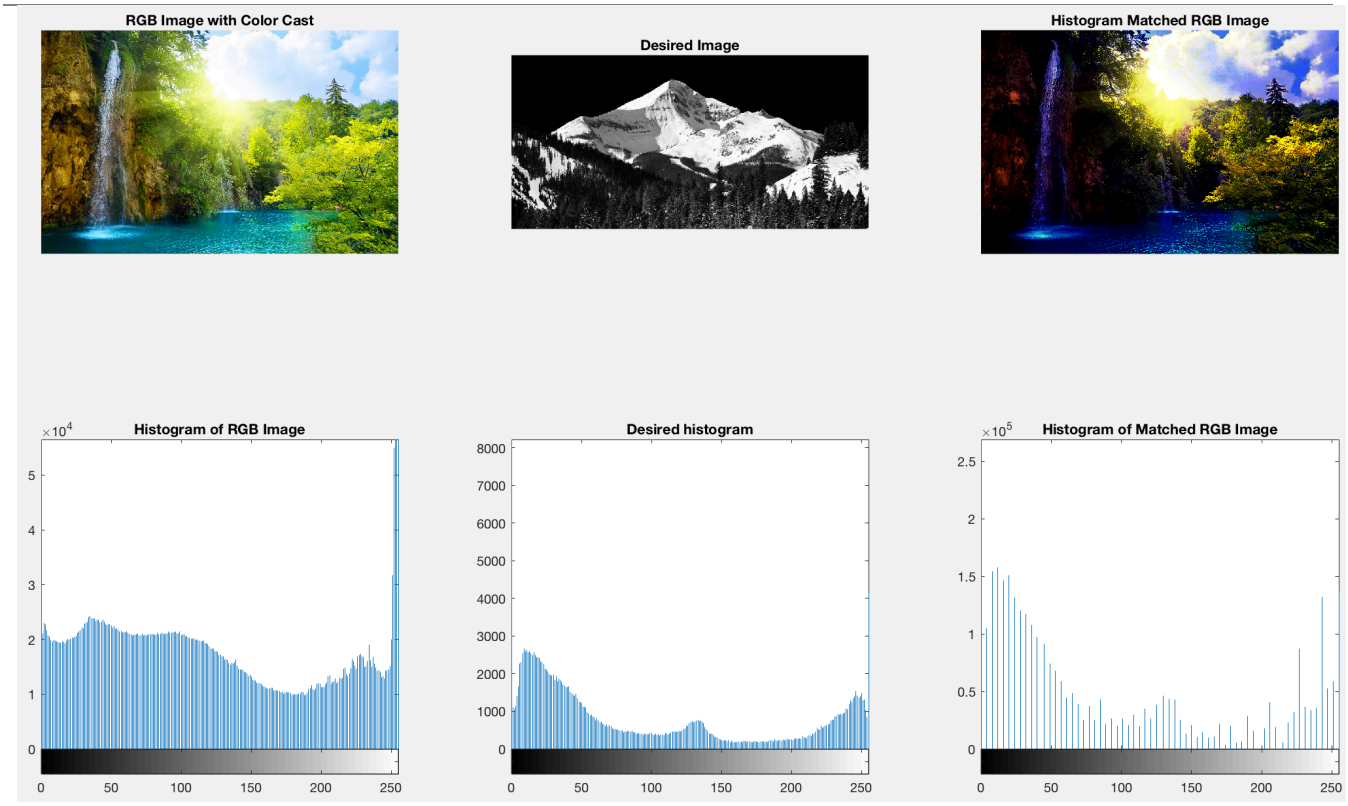


As you can see, in the second image, Histeq function didn't work well, because it caused "Blocky" or "Chess-Board" artifact in the brightest region of the image. We need to improve the bright intensities of the desired histogram.

2.

```
%%----------Part 2 - Question 2 --------------
%%--------- Mandana Samiei--------------------
%%--------- Student Id: 40059116 -------------
%Read the color image into the workspace.
I = imread('Forest1.jpg');
Ref = imread('RefImg.jpg');
if ndims(Ref) == 3
    J = rgb2gray(Ref);
end
B = imhistmatch(I,Ref);

figure
subplot(1,3,1);
imshow(I);
title('RGB Image with Color Cast');

subplot(1,3,2);
imhist(J,1024);
title('desired histogram');

subplot(1,3,3);
imshow(B);
title('Histogram Matched RGB Image');
```





Compare the results:
Generally human eye prefers high contrasted images due to have a better projection of details in the image. According to the above example, we can see that in the histogram matching we achieved a better result in both cases. Especially, in the second image that we have a very bright light of the sun. However, quality of an image is a subjective topic and it depends on a specific image and its application as well as the the person who want to recognize the best quality. Personally, I find the histogram matching result of a better quality than the other.
The desired image(histogram) is also an important factor in histogram matching, in this example, we chose a very high contrasted image with a wide range of intensities that changes the histogram of the input image in a way that it covers a broad range of intensities which is one of the characteristics of best quality images.