Jashore University of Science and Technology

Department of Computer Science and Engineering

Course Title: Technology Transfer policy and Professional Ethics

Course Code: 4104

Bio-Medical Signal and Image Processing Lab

Lab Report On

RGB to Gray Scale,

Submitted To	Submitted By
Dr. A F M Shahab Uddin **Lecturer** **Dept. of Computer Science and Engineering** **Jashore University of Science and Technology**	**Shusmita Das Gupta** **ID: 180128** **4th Year 1st Semester** **Dept. of Computer Science and Engineering** **Jashore University of Science and Technology**

Date: 12/06/2023

Problem No: 01

Problem Name: RGB to Gray Scale using Average method and Weighted Average Method (Luminosity method)

Theory: We convert a color image into a grayscale image. There are two methods to convert it. The methods are: Average method and Weighted method.

Average Method:

Average method is the simplest one. You just must take the average of three colors. Since it's an RGB image, so it means that you have add r with g with b and then divide it by 3 to get your desired grayscale image. It's done in this way. Grayscale = (R+G+B)/3.

Weighted method:

In weighted average method, since red color has more wavelength of all the three colors, and green is the color that has not only less wavelength than red color but also green is the color that gives more smoothing effect to the eyes. It means that we have to decrease the contribution of red color, and increase the contribution of the green color, and put blue color contribution in between these two. So, the new equation that form is:

New grayscale image = ((.3*R) + (.59*G) + (.11*B)). According to this equation, red has contributed 30%, Green has contributed 59% which is greater in all three colors and blue has contributed 11%.

Average Method:

1. Load image.
2. Separate RGB channels.
3. Compute grayscale image using average method.
4. Display the original and grayscale images.

Weighted Method:

1. Load RGB image
2. Convert RGB image to gray scale using weighted average method.
3. Display Original and grayscale images.

Implementation: (Average Method)

```
img = imread(‘rgb.jpg');

redChannel = img(:, :, 1);

greenChannel = img(:, :, 2);

blueChannel = img(:, :, 3);

grayImage = uint8((double(redChannel) + double(greenChannel) +
double(blueChannel)) / 3);

subplot (1, 2, 1),imshow(img), title ('Original RGB Image');

subplot(1, 2, 2),imshow(grayImage), title('Grayscale Image');
```

Implementation: (Weighted method)

```
img=imread('rgb.jpg');

img=im2double(img);

 a=0.3*img(:,:,1)+.59*img(:,:,2)+0.11*img(:,:,3);

subplot(1,2,1), imshow(img), title('Original Image');

subplot(1,2,2), imshow(a), title('Gray Scale Image');
```

Output:

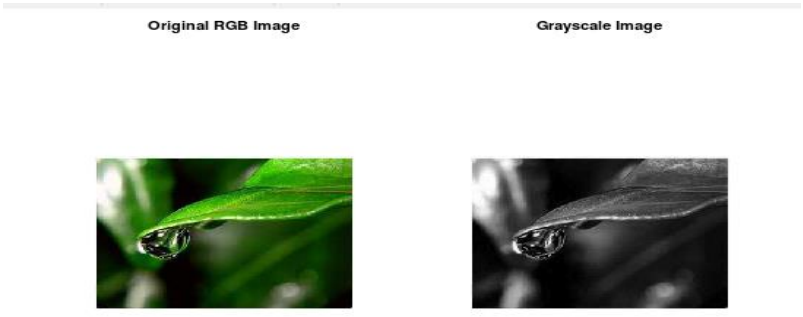

Fig1.1: Average Method

Fig1.2: Weighted Method

Conclusion:

From this figure we can see that the image has now been properly converted to grayscale using weighted method. As compare to the result of average method. This image is brighter.

Problem No: 02

Problem name: Noise Add and Remove on the image.

Theory: An overview of the significance of noise in images, discussing how noise can degrade image quality and affect various applications. It may explain different types of noise, such as Gaussian noise, salt-and-pepper noise, or speckle noise, and their characteristics. The introduction also outlines the objectives of the study, such as evaluating the impact of noise on image analysis algorithms or improving image quality by reducing noise.

Implementation:

```
c=imread('rgb.jpg');
```

```
g = rgb2gray(c);
n = imnoise(g,'salt & pepper',0.02);
subplot(2,2,1),imshow(g);
subplot(2,2,2),imshow(n);
avg=filter2(fspecial('average',3),n)/255;
subplot(2,2,3),imshow(avg);
title('Image filtered by Averaging filter');
med = medfilt2(n,[3 3]);
subplot(2,2,4),imshow(med);
title('Image filtered by Median filter');
```

Output:

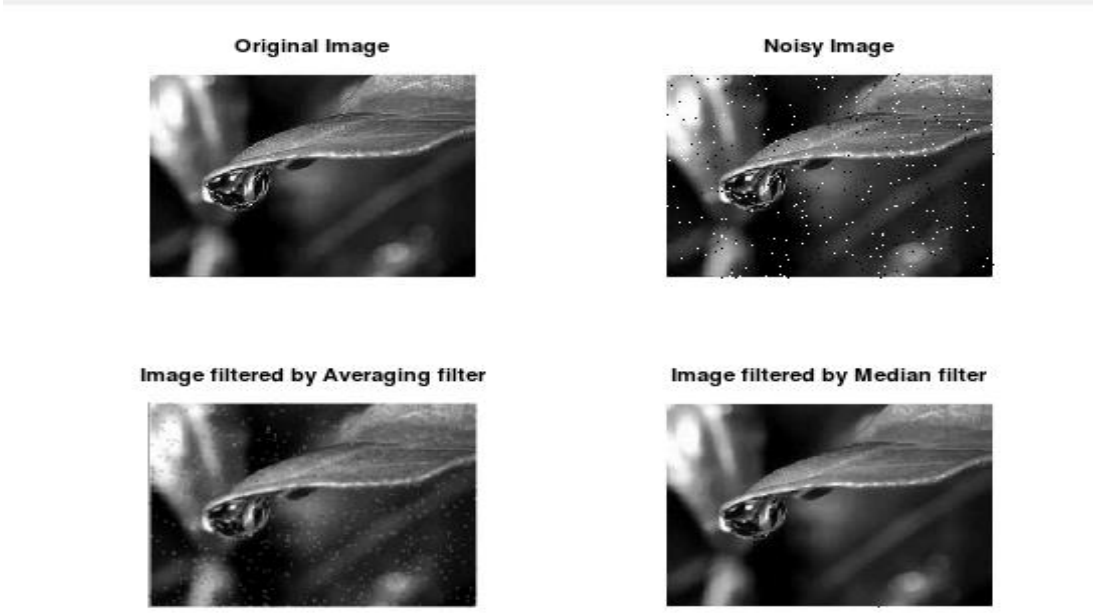

Fig 2.1: Adding noise and removing noise

Conclusion: The conclusion section summarizes the results and implications of the noise addition and removal process. It discusses the effectiveness of the noise addition techniques in simulating real-world noise scenarios and evaluates the

performance of noise removal methods in reducing noise while preserving image details.

Problem no: 03

Problem name: Contrast enhancement of an image.

Theory: The contrast of an image refers to the difference in brightness between the light and dark areas. It determines the range of tones present in an image, from the brightest highlights to the darkest shadows. An image with high contrast will have a significant difference between its lightest and darkest areas, while an image with low contrast will have minimal variation.

Experiment details:

1. Let, load the image which we want to contrast.
2. Then calculate the minimum and maximum pixel values.
3. Define the desired lower and upper limits for the contrast.
4. Perform manual contrast stretching
5. Clip pixel values outside the range [0, 1].
6. Display the adjusted image.

Implementation:

```
pkg load image

image = imread('rgb.jpg');

image = im2double(image);

enhancementValue = 0.2;

enhancedImage = image + enhancementValue;

subplot(1,2,1);

imshow(image);

title('Original Image');

subplot(1,2,2);

imshow(enhancedImage);
```

```
title('Enhanced Image');
```

Output:

Figure 3.1: Contrast of an image.

Conclusion: The conclusion section summarizes the results and implications of the analysis or enhancement of image contrast. It discusses the effectiveness of the contrast measurement techniques or enhancement methods employed.

Problem no: 04

Problem Name: Arithmetic Operation in Image

Theory:

Arithmetic operations in image processing involve performing mathematical operations on the pixel values of an image. These operations can be used for various purposes, such as image enhancement, blending of images, or creating special effects. Common arithmetic operations include addition, subtraction, multiplication, and division.

Experiment Details:

1. Load two images of the same size.

2. Convert the images to the same color space if needed (e.g., grayscale or RGB).
3. Perform addition of the pixel values of the two images to create a blended image. This can be done by adding the corresponding pixel values from both images.
4. Perform subtraction of the pixel values of the two images to create a difference image. This can be done by subtracting the pixel values of one image from the corresponding pixel values of the other image.
5. Display the original images, blended image, and difference image to observe the results.

Implementation:

```
image1 = imread('rgb.jpg');
image2 = imread('original.jpg');
 [rows, cols] = size(image1);
[rows, cols] = size(image2);


if ~isequal(size(image1), size(image2))
  error('Images must have the same dimentions.');
endresult_image = zeros(rows,cols, 'uint8');
for i = 1:rows
  for j= 1:cols
    result_image(i,j) = image1(i,j) + image2(i,j);
     %result_image(i,j) = image1(i,j) - image2(i,j);
     %result_image(i,j) = image1(i,j) * image2(i,j);
     %result_image(i,j) = image1(i,j) / image2(i,j);
  endfor
end
```

```
subplot(1,3,1), imshow(image1), title('Image 1');
subplot(1,3,2), imshow(image2), title('Image 2');
subplot(1,3,3), imshow(result_image), title('Result Image');
```

Output:

Conclusion:

The conclusion section summarizes the results and implications of applying arithmetic operations on images. It discusses the effects achieved through arithmetic operations, such as combining images for image blending or creating new composite images with specific characteristics.

Problem no: 05

Problem Name: Negative image creation.

Theory:

A negative image is a total inversion, in which light areas appear dark and vice versa. A negative color image is additionally color-reversed, with red areas cyan, greens appearing magenta, and blues appearing yellow, and vice versa.

Experiment details:

1. Load image.
2. Get the maximum pixel value.

3. Subtract each value from the maximum value
4. Display the negative image.

Implementation:

```
image = imread('original.jpg');
negativeImage = 255 - image;
subplot(1, 2, 1);
imshow(image);
title('Original Image');
subplot(1, 2, 2);
imshow(negativeImage);
title('Negative Image');
```

Output:

Figure 5.1: Negative image.

Conclusion:
The conclusion section summarizes the results and implications of applying the image negative transformation. It discusses the visual effects achieved through the image negative process, such as the reversal of brightness or color tones.

Problem No: 06
Problem name: Prewitt operator for edge detection of an image.

Theory: Prewitt operator is used for edge detection in an image. It detects two types of edges: Horizontal edges and vertical edges. Edges are calculated by using different corresponding pixels intensities of an image. All the masks that are used for edge detection are also known as derivative masks.

Experiment details:

1. In octave, we first read the RGB image.
2. Then convert the image into grayscale if necessary.
3. Apply prewitt operator such as: vertical operator[-1 0 1; -1 0 1; -1 0 1]; horizontal operator[-1 -1 -1; 0 0 0; 1 1 1];
4. Calculate the gradient magnitude and adjust the threshold as needed.
5. Display the original image and the detected image.

Implementation:

```
img = imread('rgb.jpg');

if size(img,3)==3

img=rgb2gray(img);

end

vert_prewitt=[-1 0 1; -1 0 1; -1 0 1];

horz_prewitt=[-1 -1 -1; 0 0 0; 1 1 1];

vert_gradient = conv2(double(img), vert_prewitt,'same');

horz_gradient = conv2(double(img), horz_prewitt,'same');
```

```
gradient_mag = sqrt(vert_gradient.^2 + horz_gradient.^2);
threshold = 100;
bin_edges = gradient_mag>threshold;
subplot(1, 2, 1), imshow(img), title('Original Image');
subplot(1, 2, 2), imshow(bin_edges), title('Binary Edges');
```

Output:

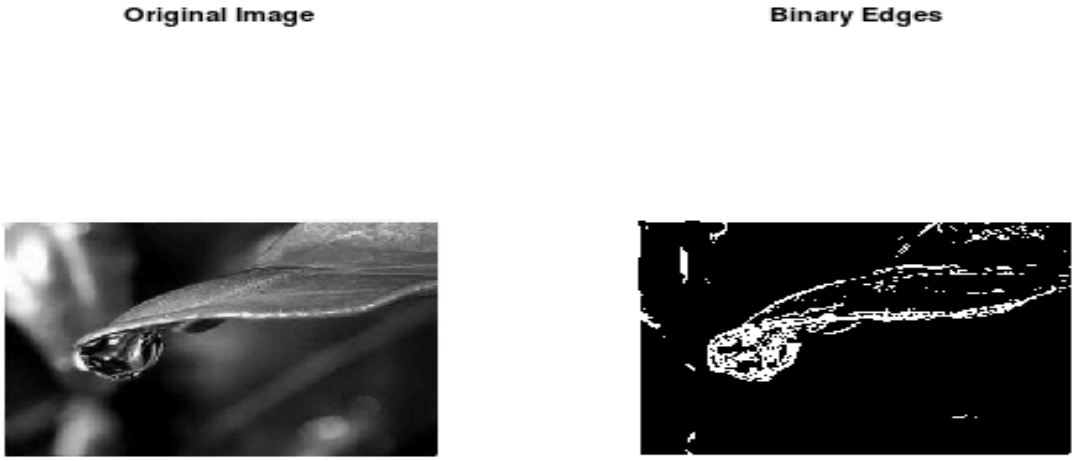

Figure 6.1: Prewitt operator used in an image.

Problem No: 07

Problem Name: Sobel operator for edge detection.

Theory:

The Sobel operator is very similar to Prewitt operator. It is also a derivative mask and is used for edge detection. Like Prewitt operator sobel operator is also used to detect two kinds of edges in an image: Vertical direction, Horizontal direction. The major difference is that in sobel operator the coefficients of masks are not fix and they do not violate any property of derivative masks.

Experiment Details:

1. Load image in octave.
2. Convert the image to grayscale if it is in color
3. Convert the image to double for computations
4. Apply horizontal Sobel filter = [-1 0 1; -2 0 2; -1 0 1];
5. Apply vertical Sobel filter = [-1 -2 -1; 0 0 0; 1 2 1];
6. Compute the gradient magnitude.
7. Apply a threshold to obtain binary edges and adjust the threshold value as needed.
8. Display the original image and binary edges

Implementation:

```
img = imread('rgb.jpg');
if size(img,3)==3
img = rgb2gray(img);
end
sobel_x = [-1 0 1; -2 0 2; -1 0 1];
sobel_y = [-1 -2 -1; 0 0 0; 1 2 1];
[rows,cols] = size(img);
gradient_mag = zeros(rows, cols);
gradient_ori = zeros(rows, cols);
for i = 2:rows-1
  for j = 2:cols-1
    grad_x = sum(sum(sobel_x .* double(img(i-1:i+1, j-1:j+1))));
    grad_y = sum(sum(sobel_y .* double(img(i-1:i+1, j-1:j+1))));
     gradient_mag(i,j) = sqrt(grad_x^2 +grad_y^2);
    gradient_ori(i,j) = atan2d(grad_y,grad_x);
  endfor
```

```
end
threshold =50%;
bin_edges = gradient_mag > threshold;
subplot(2, 2, 1), imshow(img), title('Original Image');
subplot(2, 2, 2), imshow(gradient_mag, []), title('Gradient Magnitude');
subplot(2, 2, 3), imshow(gradient_ori, []), title('Gradient Orientation');
subplot(2, 2, 4), imshow(bin_edges), title('Binary Edges');
```

Output:

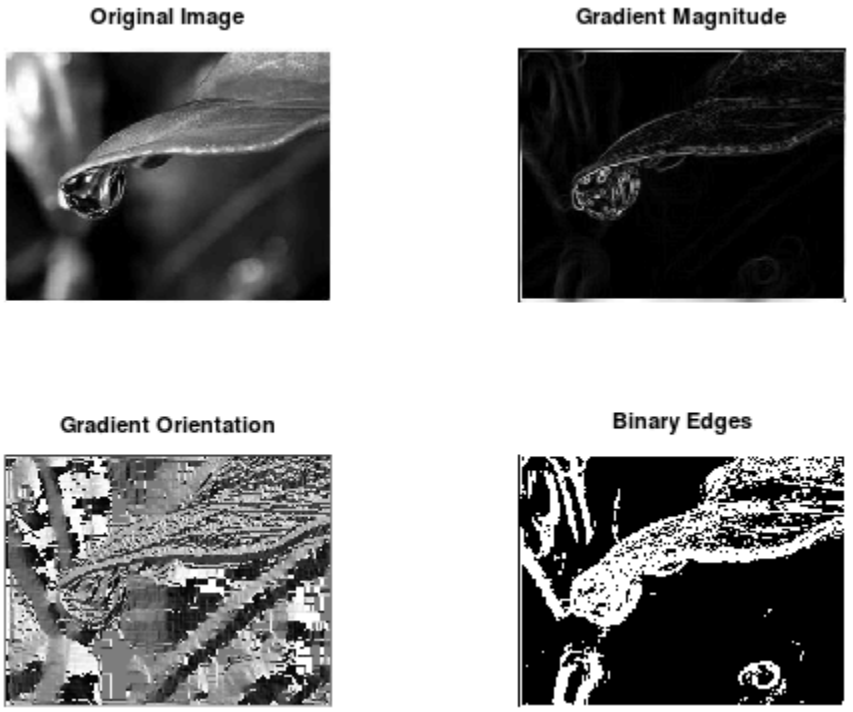

Figure 7.1: Sobel operator used in an image.

Problem No: 09

Problem Name: Histogram Equalization

Theory:

Histogram equalization is used to enhance contrast. It is not necessary that contrast will always be increase in this. There may be some cases were histogram

equalization can be worse. In that cases the contrast is decreased. We need to calculate PMF and CDF for the images.

Experiment Details:

1. Read the image
2. Convert the image to grayscale if it's in RGB format
3. Calculate the histogram of the input image
4. Calculate the cumulative distribution function (CDF)
5. Perform the histogram equalization
6. Display the original and equalized images.

Implementation:

```
img = imread('original.jpg');

if size(img,3) ==3

img = rgb2gray(img);

end

histogram = imhist(img);

cdf = cumsum(histogram)/numel(img);

out_img = uint8(255* cdf(double(img)+1));

subplot(1, 2, 1), imshow(img), title('Original Image');

subplot(1, 2, 2), imshow(out_img), title('Equalized Image');
```

Output:

Figure 9.1: Histogram Equalization on an image.

Conclusion: The conclusion section summarizes the results and implications of the histogram equalization process. It discusses the effectiveness of histogram equalization in enhancing image contrast and improving visual quality. The conclusion may evaluate the performance of different histogram equalization methods based on criteria such as contrast enhancement, preservation of image details, or perceptual quality.

Problem No: 10

Problem Name: JPEG image compression.

Theory:

Experiment Details:

1. Load the image
2. Convert the image to double precision
3. Set the quality factor (higher values for better quality; lower values for higher compression).
4. Split the image into RGB channels.
5. Define 8 X 8 block size
6. Compute the size of the image.
7. Compute the number of blocks in the image
8. Initialize the quantization matrix for each channel
9. Process each block of the image.
10. Initialize the dequantized coefficient matrices.

11. Process each block of the quantized coefficients.
12. Extract the current block from each channel
13. Perform dequantization on each block
14. Store the dequantized coefficient in the corresponding matrices.
15. Combine the RGB channels.
16. Clip the values to the valid range [0, 1]
17. Display the original and compressed images.
18. Save the

Implementation:

```
img = imread('original2.jpg');

if size(img, 3) == 3

   img = rgb2gray(img);

end

img = double(img);

[rows, cols] = size(img);

num_blocks = (rows / 8) * (cols / 8);

quantization_matrix = [16 11 10 16 24 40 51 61;

                       12 12 14 19 26 58 60 55;

                       14 13 16 24 40 57 69 56;

                       14 17 22 29 51 87 80 62;

                       18 22 37 56 68 109 103 77;

                       24 35 55 64 81 104 113 92;

                       49 64 78 87 103 121 120 101;

                       72 92 95 98 112 100 103 99];


compressed_img = zeros(size(img));

for i = 1:8:rows
```

```
    for j = 1:8:cols
        block = img(i:i+7, j:j+7);
        quantized_block = round(block./ quantization_matrix);
        compressed_img(i:i+7, j:j+7) = quantized_block;
    end
end
subplot(1, 2, 1), imshow(uint8(img)), title('Original Image');
subplot(1, 2, 2), imshow(uint8(compressed_img)), title('Compressed Image');
```

Output:

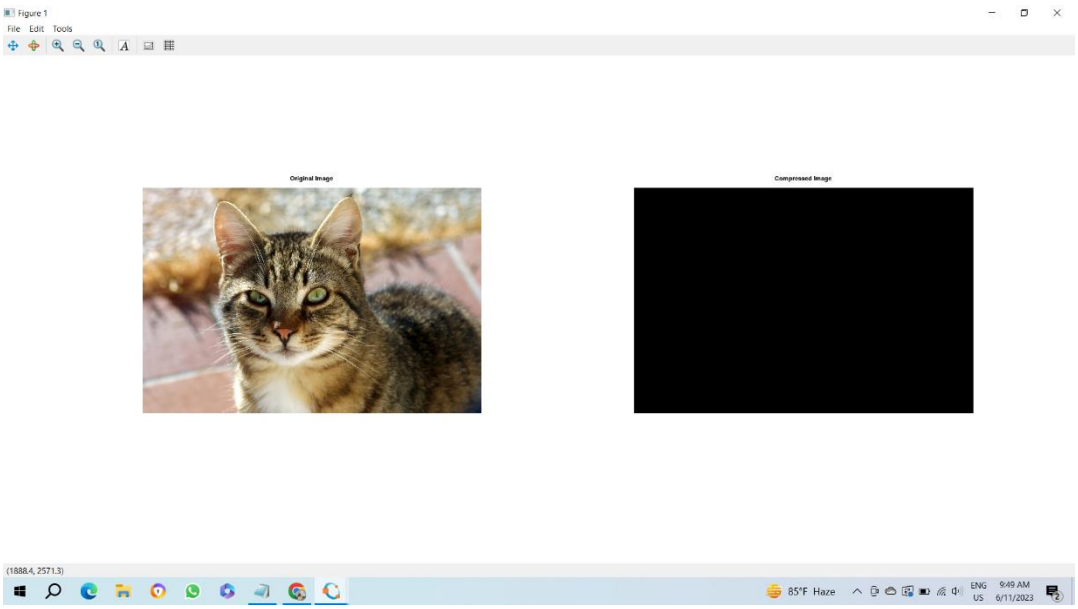

Figure 10.1: JPEG Image Compression.

Conclusion: The conclusion section summarizes the results and implications of the analysis or enhancement of image contrast. It discusses the effectiveness of the contrast measurement techniques or enhancement methods employed. The conclusion may evaluate the performance of different contrast measures or enhancement algorithms based on criteria such as visual quality, preservation of image details, or suitability for specific applications. It may also highlight the potential applications of contrast analysis or enhancement in areas like medical imaging, computer vision, or image-based machine learning. Finally, the conclusion may mention any limitations or challenges related to image contrast and suggest future research directions or improvements in the field.