The University of Windsor

ELEC4490: Sensors and Vision Systems

Summer 2020

Assignment # 3

Image Enhancement

Friday, July 21, 2020

Emmanuel Mati

104418019

1a)

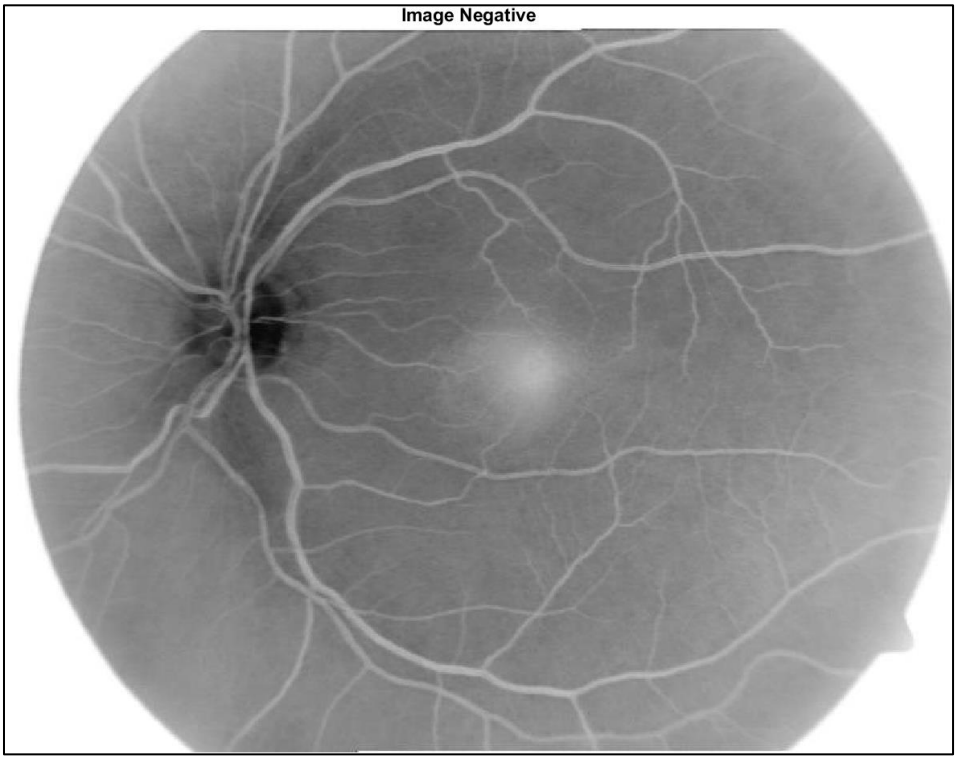
CODE:

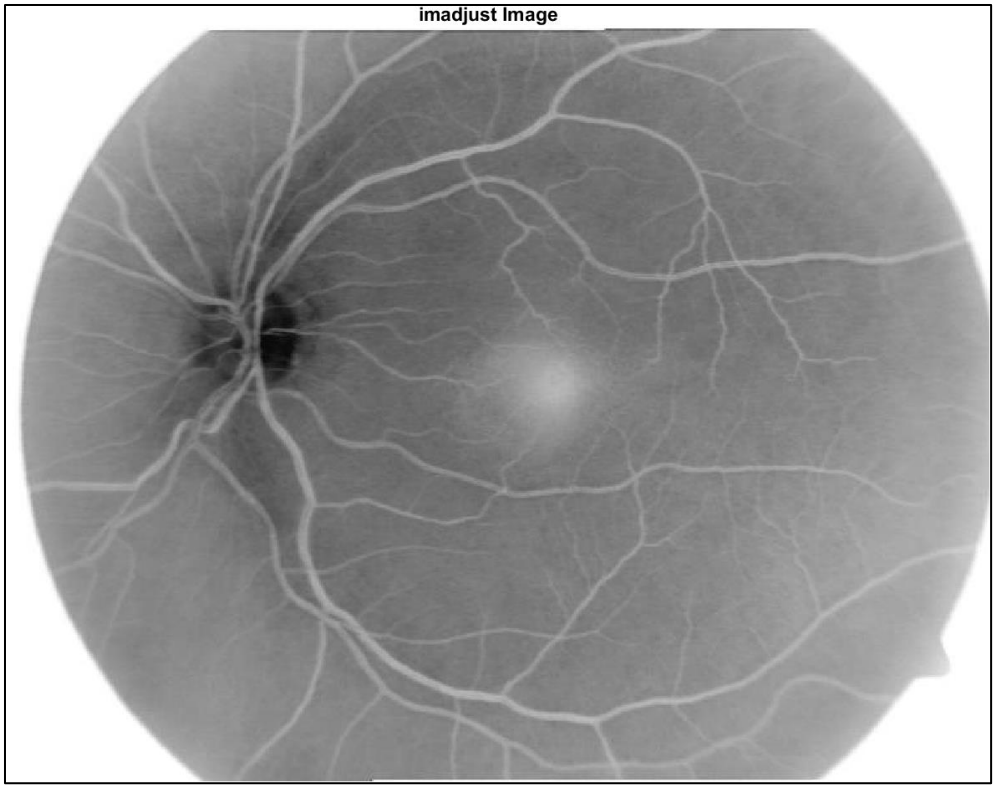
```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 3.1 a

%Clearing previous results
close all
clear all
clc

%getting our image and displaying it
x = imread('eye.jpg');
imshow(x);
title('Input Image');

%converting our image into a double for manipulation
x = double(x);

%Creating our negative array
NegativeValues(1:576, 1:768) = 255;

NegImage = imsubtract(NegativeValues, x); %Creating our Negative image

%Converting to 8 bit so we can show the image
NegImage=uint8(NegImage);

%Showing our negative Image
figure
imshow(NegImage);
title('Image Negative');

%%Comparing with imadjust
figure
imshow(imadjust(imread('eye.jpg'), [0 1], [1 0])) %we output the negative of
the image
title('imadjust Image');
```

Output Results

Input Image

Image Negative





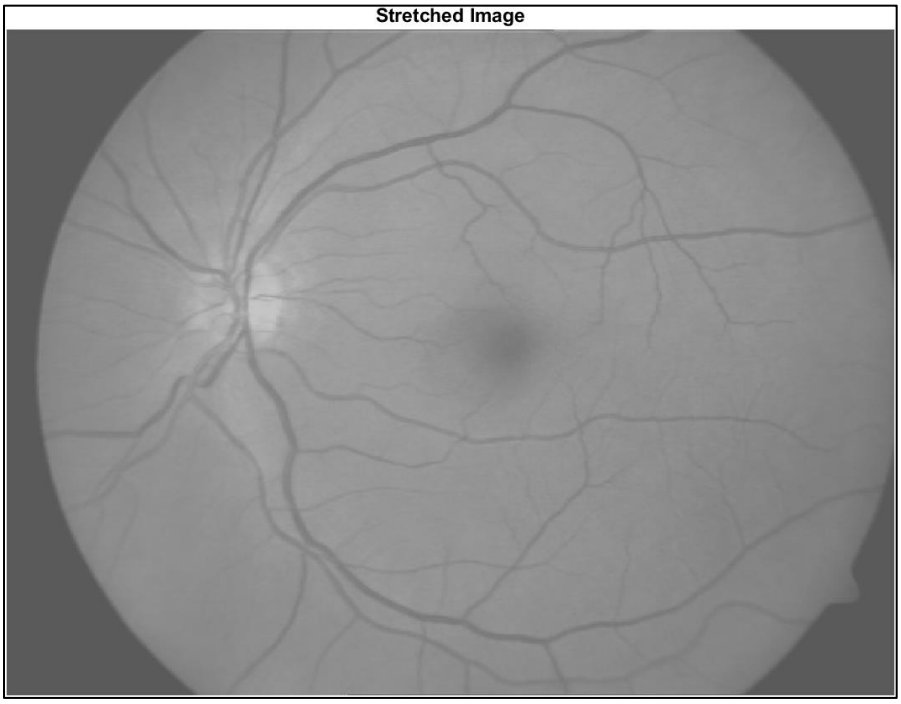
Results show exact same outcome when imadjust is used.

1b)

Code:

```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 3.1 b

%Clearing previous results
close all
clear all
clc

%getting our image and displaying it
x = imread('eye.jpg');
imshow(x);
title('Input Image');

%converting our image into a double for manipulation
x = double(x);

%Creating our negative array
NegativeValues(1:576, 1:768) = 255;

NegImage = imsubtract(NegativeValues, x); %Creating our Negative image

%Converting to 8 bit so we can show the image
NegImage=uint8(NegImage);

%Showing our negative Image
figure
imshow(NegImage);
title('Image Negative');

%%Comparing with imadjust
figure
imshow(imadjust(imread('eye.jpg'), [0 1], [1 0])) %we output the negative of
the image
title('imadjust Image');
```

Output with lower bound = 90 & upper bound = 200



Results show exact same outcome when imadjust is used.

1C)

Code:

```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 3.1 c

%Clearing previous results
close all
clear all
clc

%getting our image and displaying it
x = imread('eye.jpg');
figure
imshow(x);
title('Input Image');


x = imadjust(imread('eye.jpg'), [], [], 0.5); %adjusting our gamma
figure
imshow(x);
title('Gamma Adjusted Image at 0.5');

x = imadjust(imread('eye.jpg'), [], [], 2); %adjusting our gamma
figure
imshow(x);
title('Gamma Adjusted Image at 2');
```

Outputs with gamma set to 0.5 and 2

Lower gamma values (less than 1)enhanced lighter features while higher gamma values (greater than 1) enhanced darker features.

2)

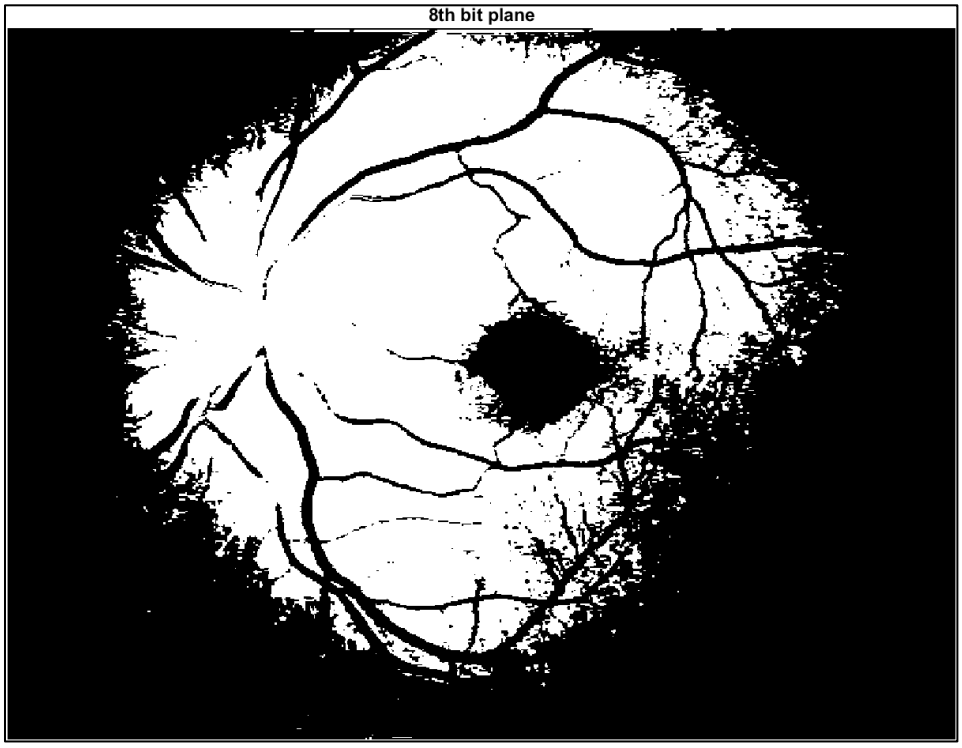
Code:

```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 3

%%Code for Part A

%Clearing previous results
close all
clear all
clc

%getting our image and displaying it
x = imread('eye.jpg');
figure
imshow(x);
title('Input Image');

x = double(x); % making our value a double for manipulation

%Calculating 8th bit plane
b8 = bitget(x, 8);
figure
imshow(b8)
title('8th bit plane')

%Calculating 7th bit plane
b7 = bitget(x, 7);
figure
imshow(b7)
title('7th bit plane')

%Calculating 6th bit plane
b6 = bitget(x, 6);
figure
imshow(b6)
title('6th bit plane')

%Calculating 5th bit plane
b5 = bitget(x, 5);
figure
imshow(b5)
title('5th bit plane')

%Calculating 4th bit plane
b4 = bitget(x, 4);
figure
imshow(b4)
title('4th bit plane')
```

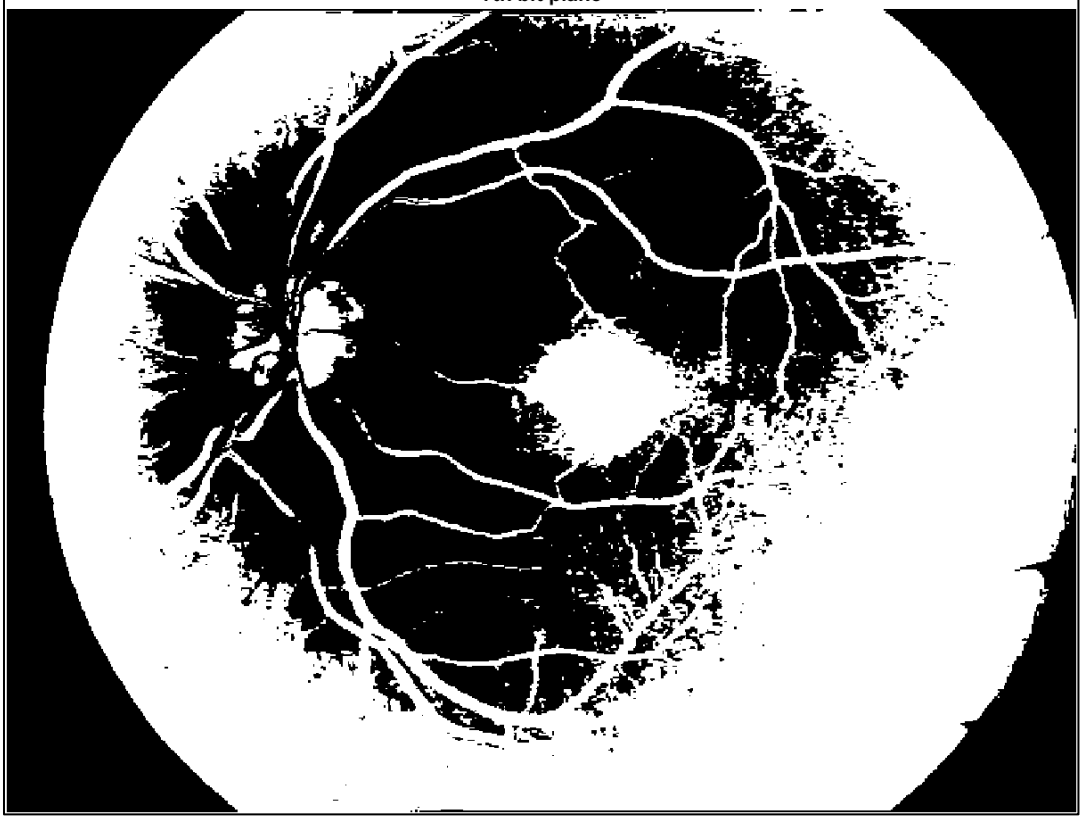
```
%Calculating 3rd bit plane
b3 = bitget(x, 3);
figure
imshow(b3)
title('3rd bit plane')

%Calculating 2nd bit plane
b2 = bitget(x, 2);
figure
imshow(b2)
title('2nd bit plane')

%Calculating 1st bit plane
b1 = bitget(x, 1);
figure
imshow(b1)
title('1st bit plane')

%%Code for Part B
%Connecting the bit slices together
NewImage = 2^7 * b8 + 2^6 * b7 + 2^5 * b6 + 2^4 * b5 + 2^3 * b4 + 2^2 * b3 +
2 * b2 + b1;

%displaying our new re-combined images
figure
imshow(uint8(NewImage));
title('Combined 8 to 1 bit-slices')

%%Code for Part C
%Bits 8-6 recombined
b8to6 = 2^7 * b8 + 2^6 * b7 + 2^5 * b6;
figure
imshow(uint8(b8to6));
title('Combined 8 to 6 bit-slices')

%Missing bits from combining 8 to 6 slices
figure
imshow(imread('eye.jpg') - uint8(b8to6))
title('Subtracted image from planes 8-6')

%Bits 8-3 recombined
b8to3 = 2^7 * b8 + 2^6 * b7 + 2^5 * b6 + 2^4 * b5 + 2^3 * b4 + 2^2 * b3;
figure
imshow(uint8(b8to3));
title('Combined 8 to 3 bit-slices')

%Missing bits from combining 8 to 3 slices
figure
imshow(imread('eye.jpg') - uint8(b8to3))
title('Subtracted image from planes 8-3')

%%Part D Watermarking our Image
WaterMark = zeros(576, 768);
WaterMark(144:432, 1:768) = 1;
```

```
%Placing the watermark on the th bit
WaterMarkedImage = 2^7 * b8 + 2^6 * b7 + 2^5 * b6 + 2^4 * WaterMark + 2^3 *
b4 + 2^2 * b3 + 2 * b2 + b1;

%displaying our new re-combined images
figure
imshow(uint8(WaterMarkedImage));
title('Water Marked Image at bit 5')

figure
imshow(imread('eye.jpg') - uint8(WaterMarkedImage))
title('Subtracted image from bit 5 and water mark')

%Placing the watermark on the 6th bit
WaterMarkedImage = 2^7 * b8 + 2^6 * b7 + 2^5 * WaterMark + 2^4 * b5 + 2^3 *
b4 + 2^2 * b3 + 2 * b2 + b1;

%displaying our new re-combined images
figure
imshow(uint8(WaterMarkedImage));
title('Water Marked Image at bit 6')

figure
imshow(imread('eye.jpg') - uint8(WaterMarkedImage))
title('Subtracted image from bit 6 and water mark')

%Placing the watermark on the 7th bit
WaterMarkedImage = 2^7 * b8 + 2^6 * WaterMark + 2^5 * b6 + 2^4 * b5 + 2^3 *
b4 + 2^2 * b3 + 2 * b2 + b1;

%displaying our new re-combined images
figure
imshow(uint8(WaterMarkedImage));
title('Water Marked Image at bit 7')

figure
imshow(imread('eye.jpg') - uint8(WaterMarkedImage))
title('Subtracted image from bit 7 and water mark')

%Placing the watermark on the 8th bit
WaterMarkedImage = 2^7 * WaterMark + 2^6 * b7 + 2^5 * b6 + 2^4 * b5 + 2^3 *
b4 + 2^2 * b3 + 2 * b2 + b1;

%displaying our new re-combined images
figure
imshow(uint8(WaterMarkedImage));
title('Water Marked Image at bit 8')

figure
imshow(imread('eye.jpg') - uint8(WaterMarkedImage))
title('Subtracted image from bit 8 and water mark')
```

Our Outputs in Descending Bit-Plane order



7th bit plane

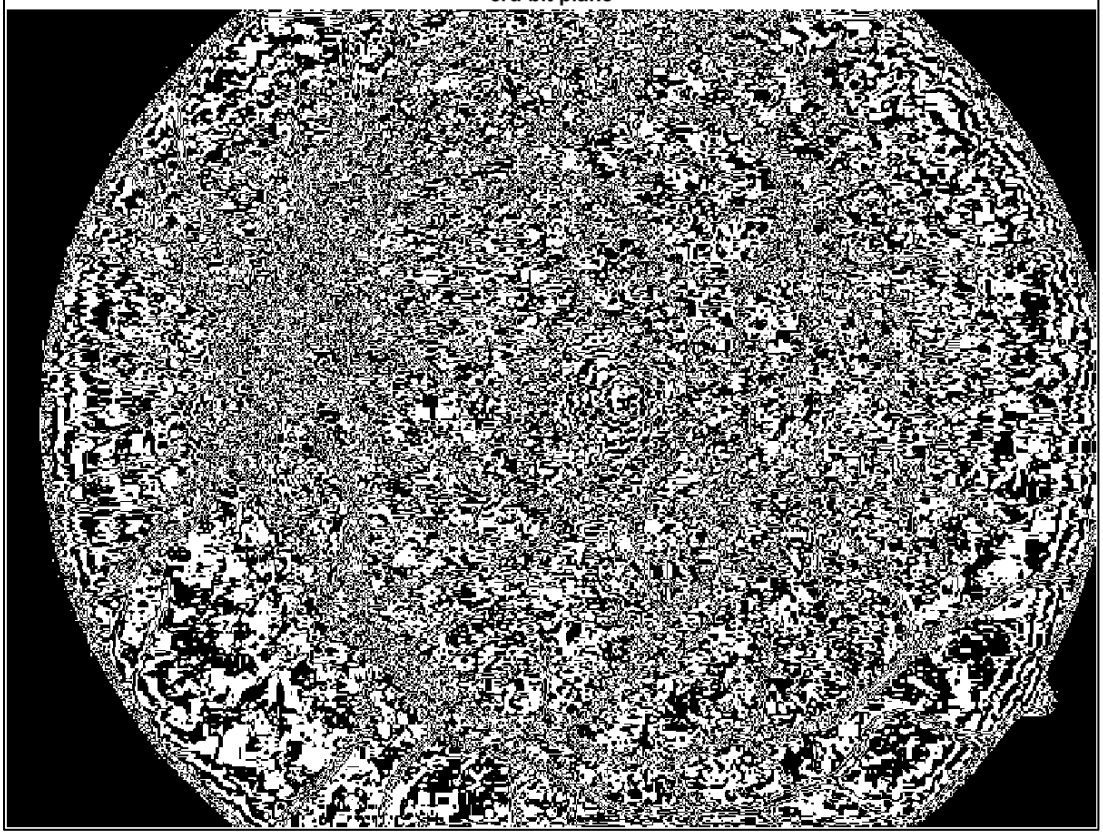


6th bit plane

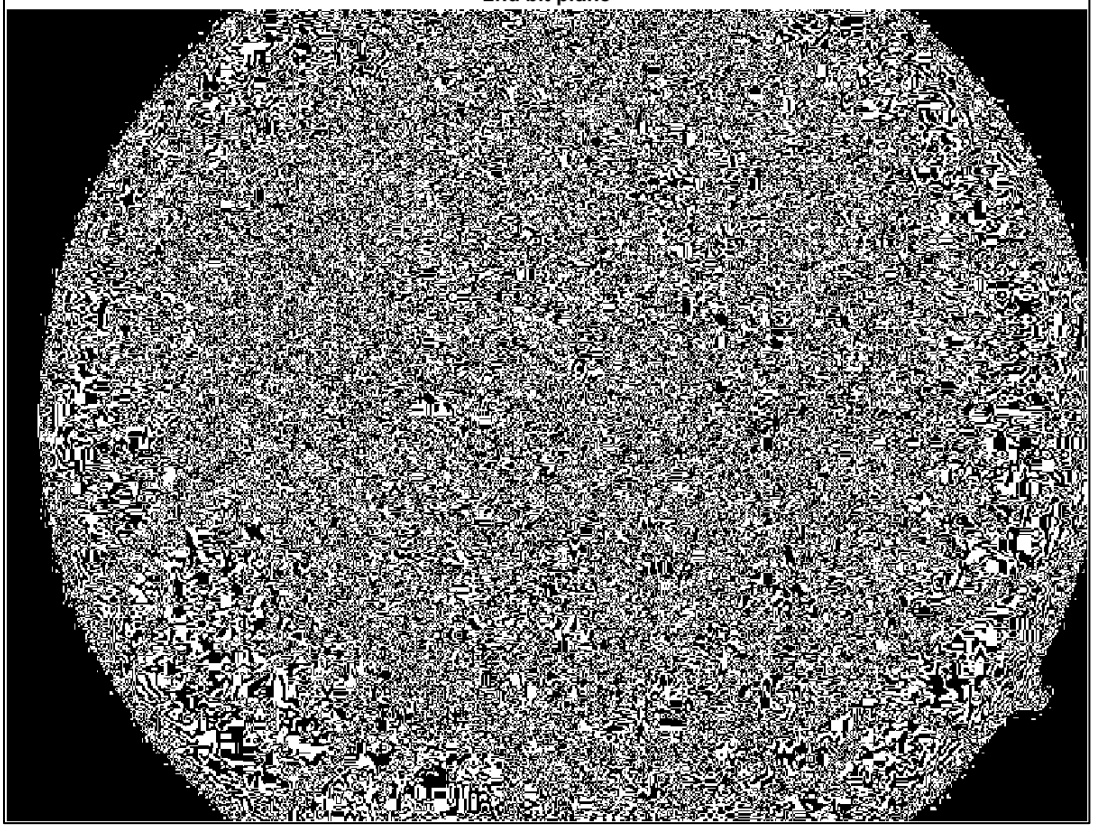
5th bit plane

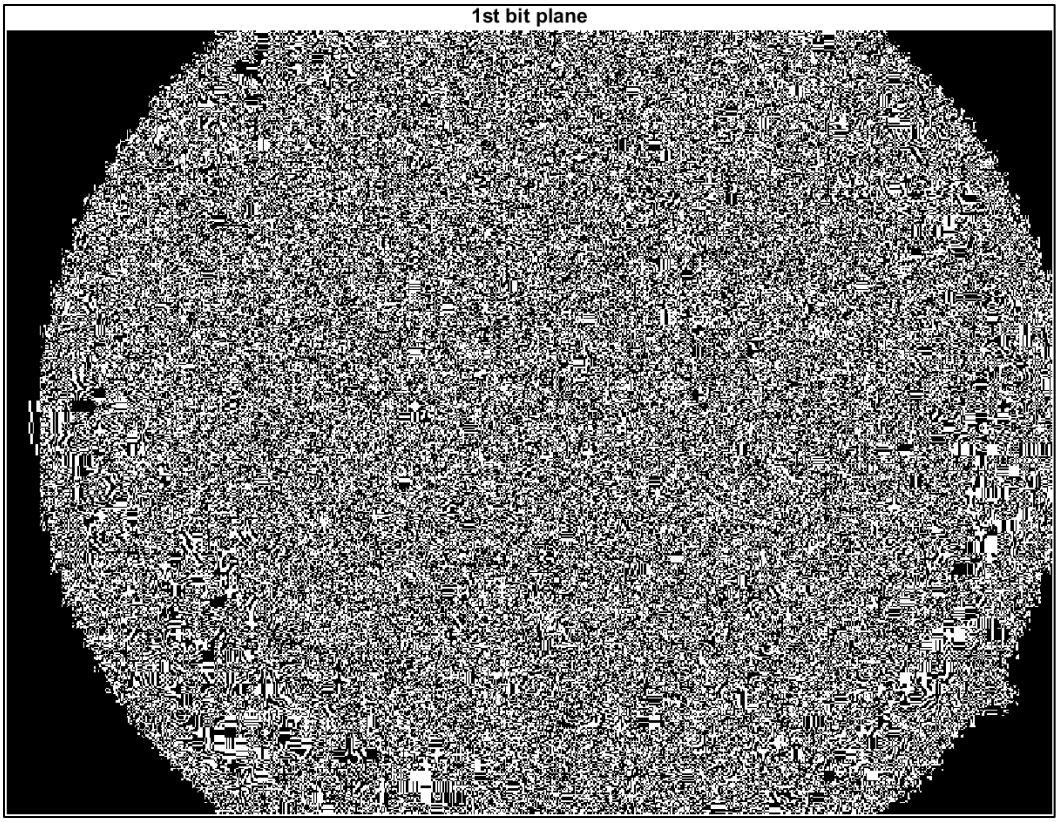
4th bit plane

3rd bit plane



2nd bit plane

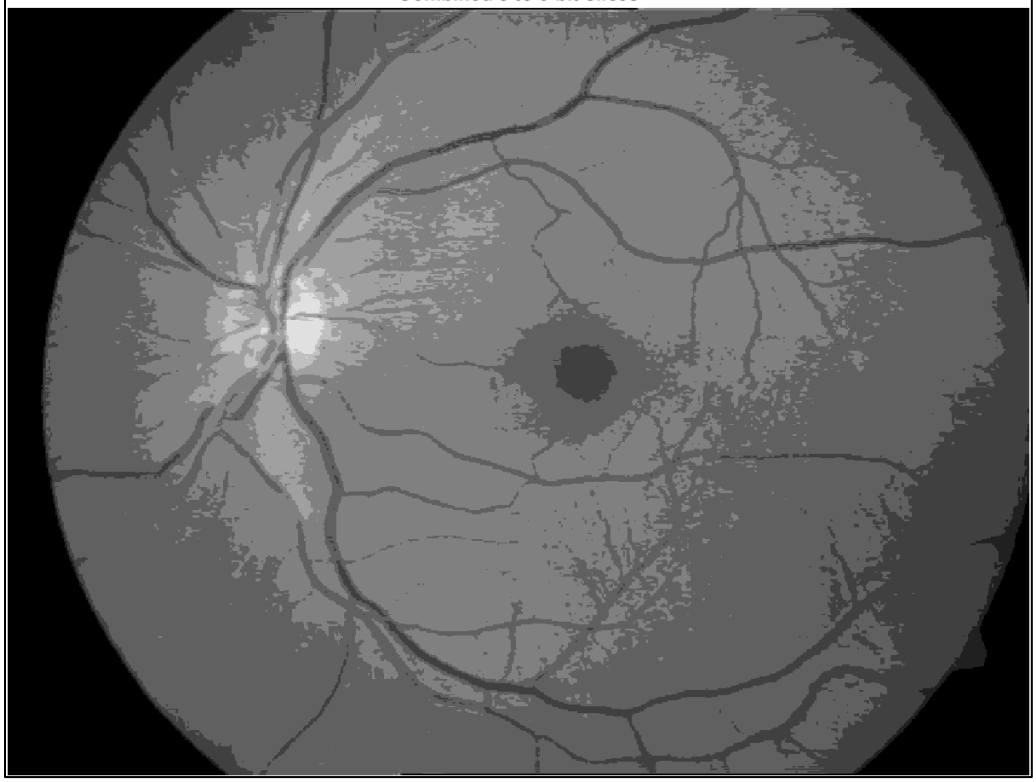




Part B Recombined Output Image

Part C Recombined Compressed Output Images

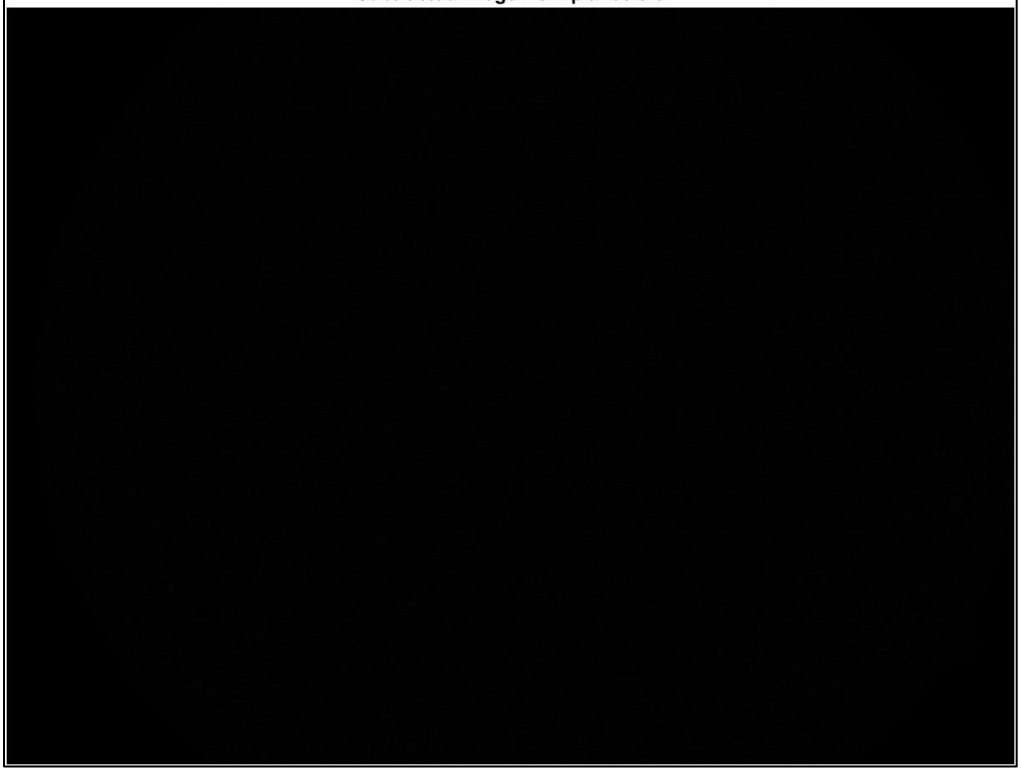
Combined 8 to 6 bit-slices



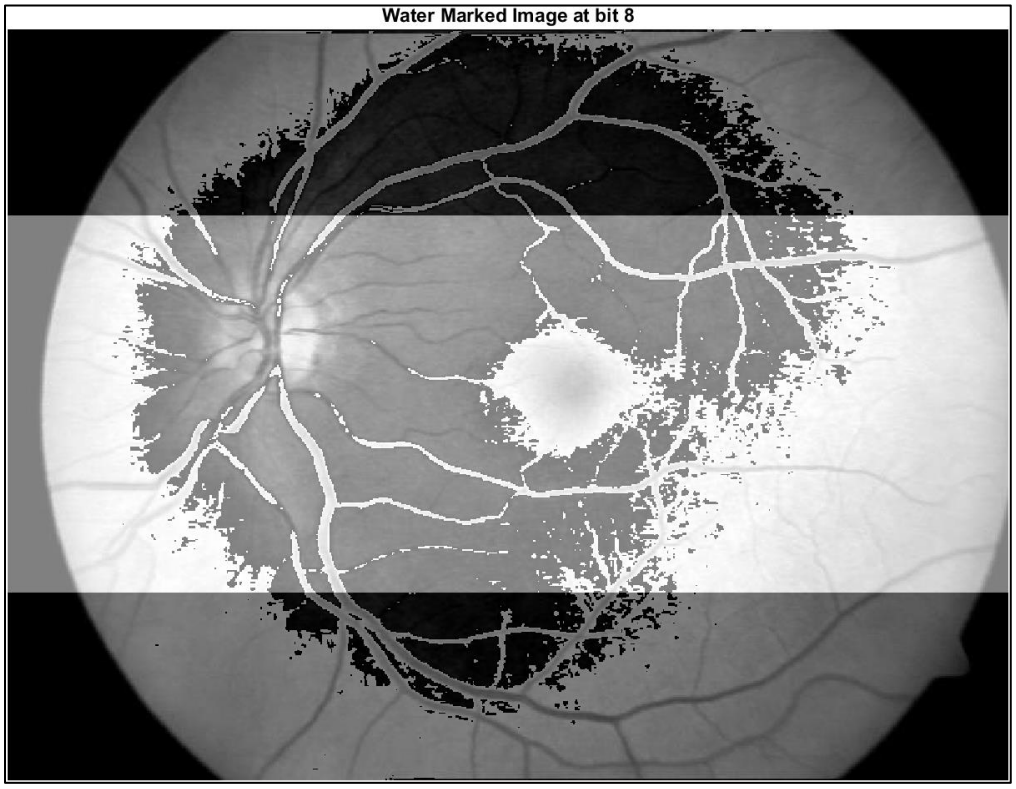
Subtracted image from planes 8-6

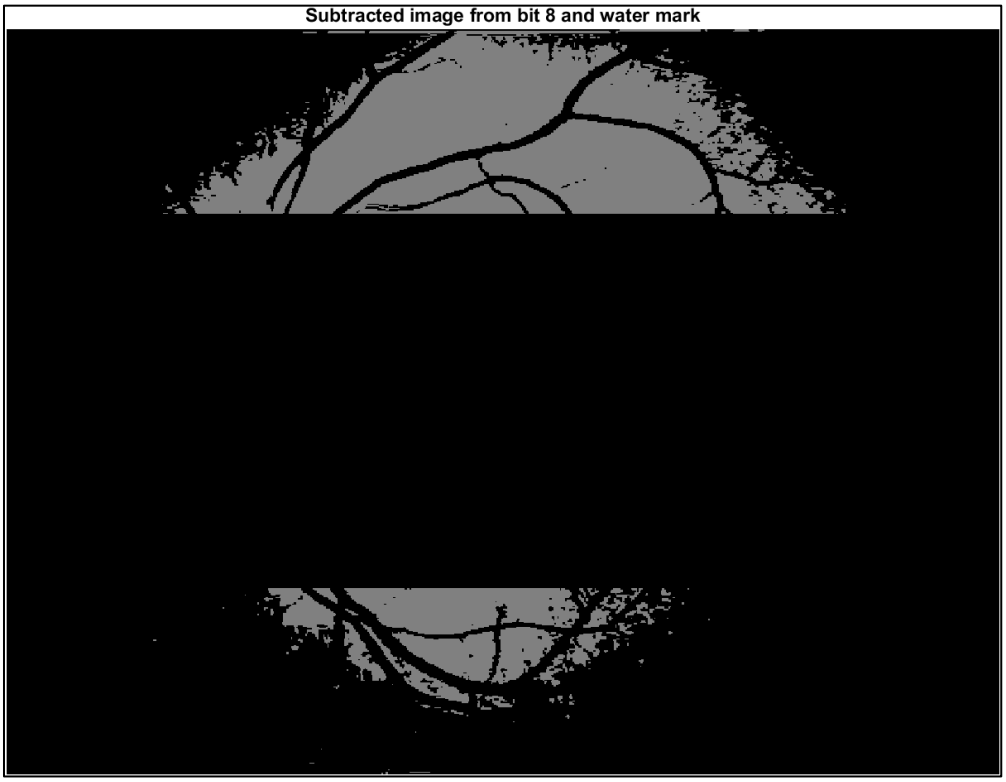
Combined 8 to 3 bit-slices

Subtracted image from planes 8-3

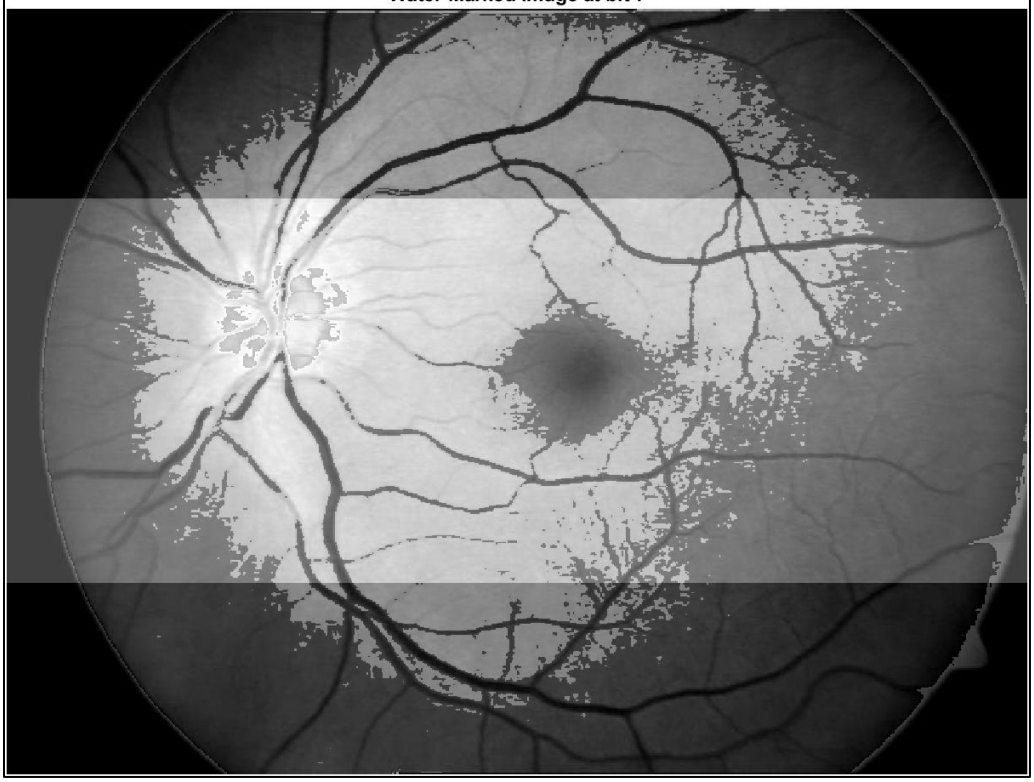


Part D Watermarked Image



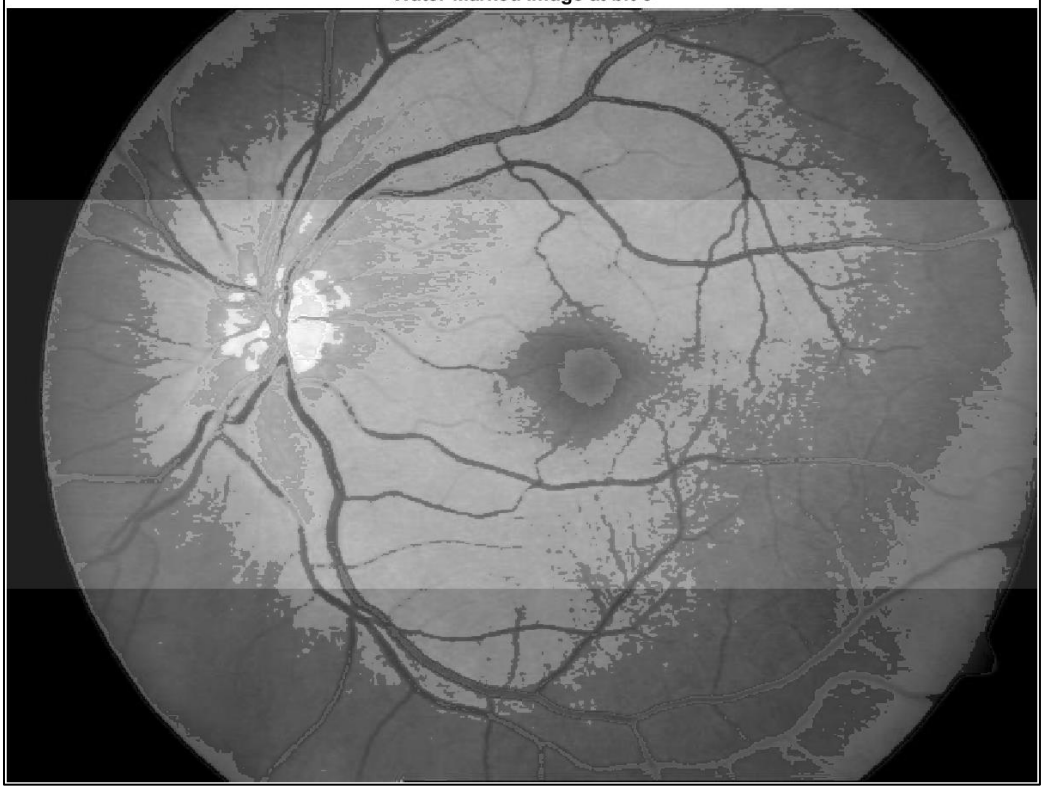


Water Marked Image at bit 7



Subtracted image from bit 7 and water mark

Water Marked Image at bit 6



Subtracted image from bit 6 and water mark

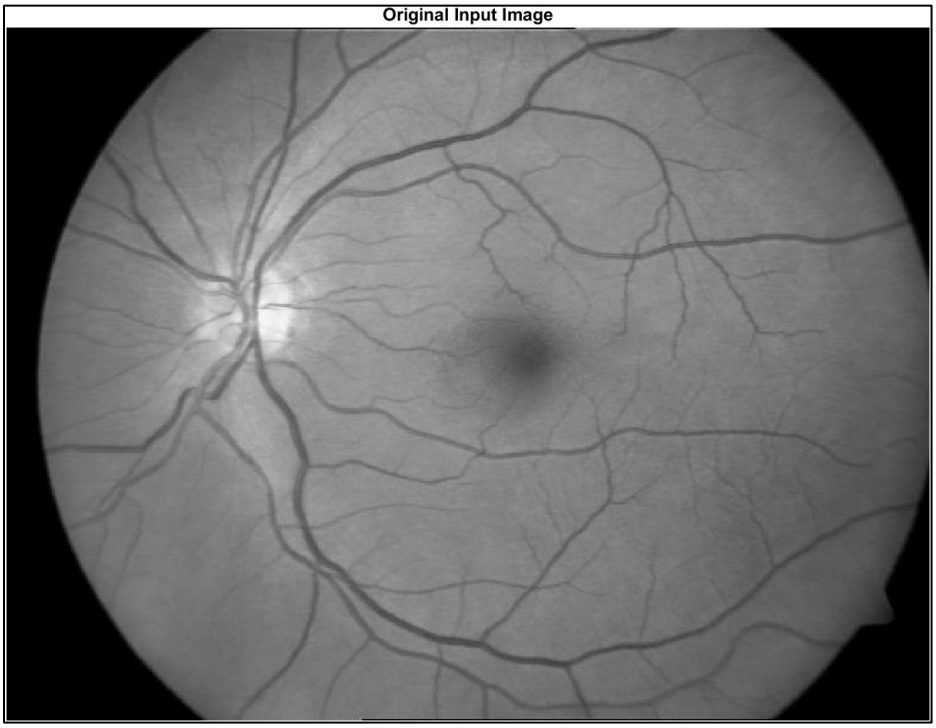
The water marke seemed to be best suited for the 6^{th} bit slice. In the 6^{th} bit, it did not remove the most detail and the water mark was still noticeable.

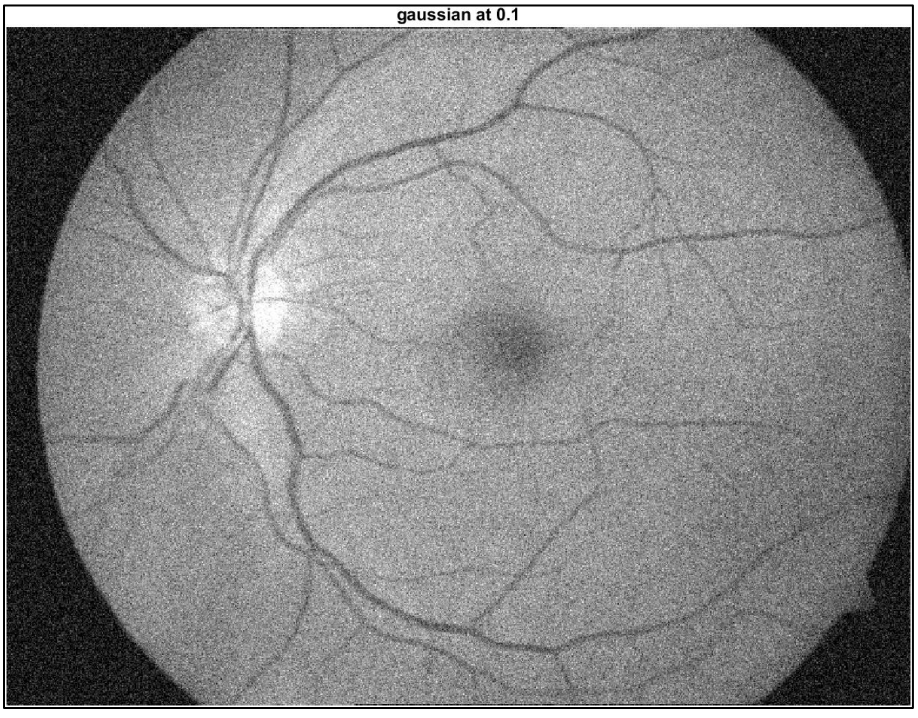
3a)

Code:

```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 3.3 A

%Clearing previous results
close all
clear all
clc

%getting our image and displaying it
x = imread('eye.jpg');
imshow(x);
title('Original Input Image');

x1 = imnoise(x,'gaussian', 0.1);
figure
imshow(x1)
title('gaussian at 0.1')

x1 = imdivide(x1, 10);
x2 = imnoise(x, 'gaussian', 0.1);
x2 = imdivide(x2, 10);
x3 = imnoise(x, 'gaussian', 0.1);
x3 = imdivide(x3, 10);
x4 = imnoise(x, 'gaussian', 0.1);
x4 = imdivide(x4, 10);
x5 = imnoise(x, 'gaussian', 0.1);
x5 = imdivide(x5, 10);
x6 = imnoise(x, 'gaussian', 0.1);
x6 = imdivide(x6, 10);
x7 = imnoise(x, 'gaussian', 0.1);
x7 = imdivide(x7, 10);
x8 = imnoise(x, 'gaussian', 0.1);
x8 = imdivide(x8, 10);
x9 = imnoise(x, 'gaussian', 0.1);
x9 = imdivide(x9, 10);
x10 = imnoise(x, 'gaussian', 0.1);
x10 = imdivide(x10, 10);

y = imadd(x1, x2);
y = imadd(y, x3);
y = imadd(y, x4);
y = imadd(y, x5);
y = imadd(y, x6);
y = imadd(y, x7);
y = imadd(y, x8);
y = imadd(y, x9);
y = imadd(y, x10);
figure
imshow(y)
```

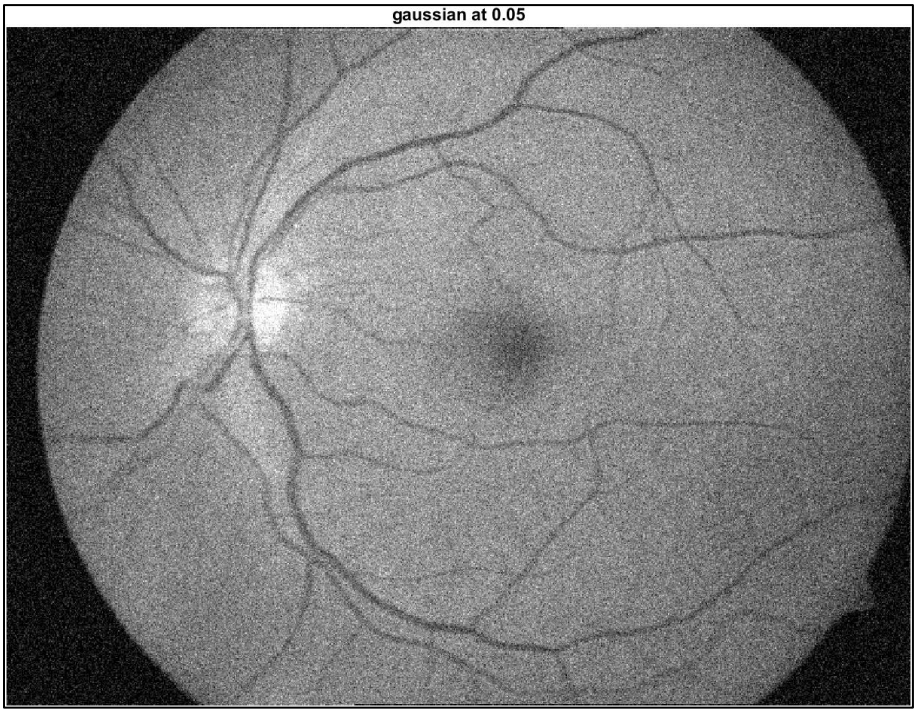
Output from using the same variance and averaging 10 times

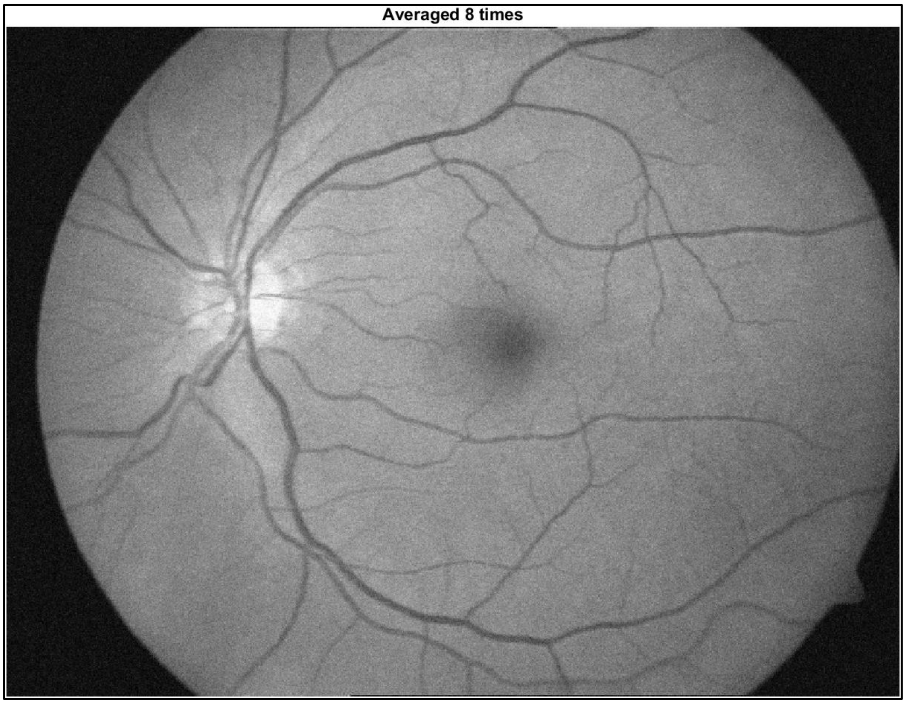




Averaged 10 times

gaussian at 0.05





When the gaussian variance was set to 0.1, we required 10 averages to produce a low noise image. Meanwhile, when the gaussian noise was set to 0.05, we only needed around 8 averages to produce a low-noise image.

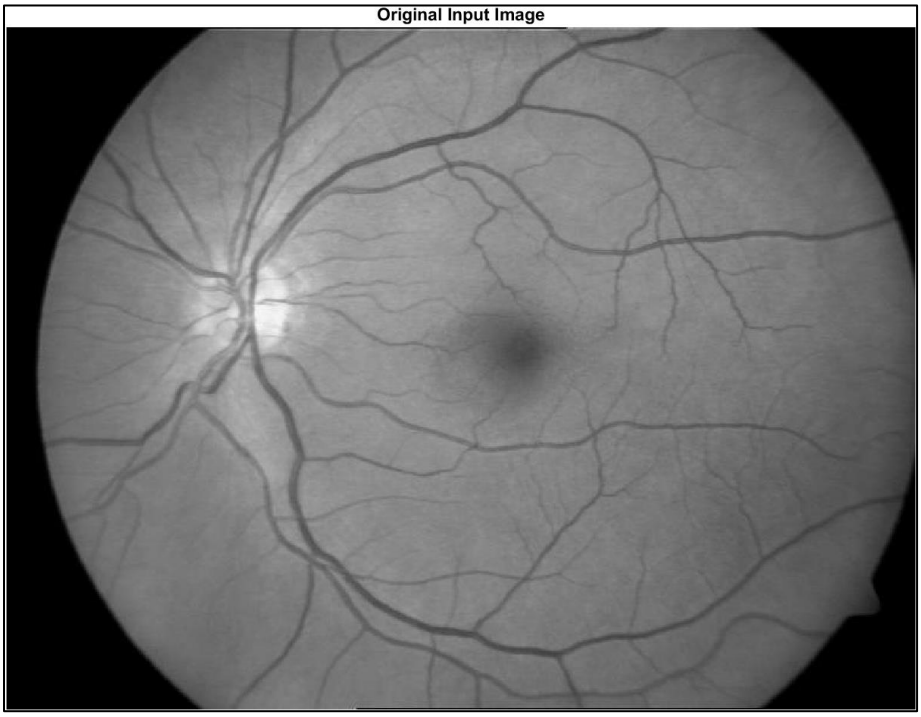
3b)

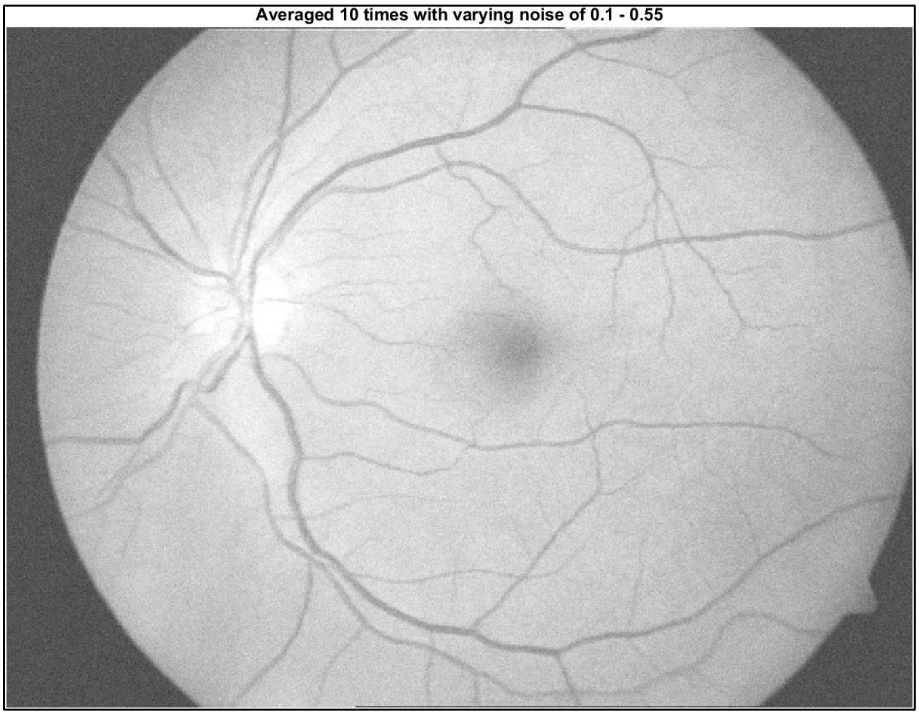
Code:

```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 3.3 B

%Clearing previous results
close all
clear all
clc

%getting our image and displaying it
x = imread('eye.jpg');
figure
imshow(x);
title('Original Input Image');

x1 = imnoise(x,'gaussian', 0.1);
figure
imshow(x1)
title('gaussian at 0.1 - 0.55')

x1 = imdivide(x1, 10);
x2 = imnoise(x, 'gaussian', 0.15);
x2 = imdivide(x2, 10);
x3 = imnoise(x, 'gaussian', 0.20);
x3 = imdivide(x3, 10);
x4 = imnoise(x, 'gaussian', 0.25);
x4 = imdivide(x4, 10);
x5 = imnoise(x, 'gaussian', 0.30);
x5 = imdivide(x5, 10);
x6 = imnoise(x, 'gaussian', 0.35);
x6 = imdivide(x6, 10);
x7 = imnoise(x, 'gaussian', 0.40);
x7 = imdivide(x7, 10);
x8 = imnoise(x, 'gaussian', 0.45);
x8 = imdivide(x8, 10);
x9 = imnoise(x, 'gaussian', 0.50);
x9 = imdivide(x9, 10);
x10 = imnoise(x, 'gaussian', 0.55);
x10 = imdivide(x10, 10);

y = imadd(x1, x2);
y = imadd(y, x3);
y = imadd(y, x4);
y = imadd(y, x5);
y = imadd(y, x6);
y = imadd(y, x7);
y = imadd(y, x8);
y = imadd(y, x9);
y = imadd(y, x10);
```

```
figure
imshow(y)
title('Averaged 10 times')

%%
%getting our image and displaying it
x = imread('eye.jpg');
figure
imshow(x);
title('Original Input Image');

x1 = imnoise(x,'gaussian', 0.05);
figure
imshow(x1)
title('gaussian at 0.05 - 0.45')

x1 = imdivide(x1, 8);
x2 = imnoise(x, 'gaussian', 0.10);
x2 = imdivide(x2, 8);
x3 = imnoise(x, 'gaussian', 0.15);
x3 = imdivide(x3, 8);
x4 = imnoise(x, 'gaussian', 0.20);
x4 = imdivide(x4, 8);
x5 = imnoise(x, 'gaussian', 0.25);
x5 = imdivide(x5, 8);
x6 = imnoise(x, 'gaussian', 0.30);
x6 = imdivide(x6, 8);
x7 = imnoise(x, 'gaussian', 0.35);
x7 = imdivide(x7, 8);
x8 = imnoise(x, 'gaussian', 0.40);
x8 = imdivide(x8, 8);
x9 = imnoise(x, 'gaussian', 0.45);

y = imadd(x1, x2);
y = imadd(y, x3);
y = imadd(y, x4);
y = imadd(y, x5);
y = imadd(y, x6);
y = imadd(y, x7);
y = imadd(y, x8);

figure
imshow(y)
title('Averaged 8 times')
```

Output after varying the noises from part A





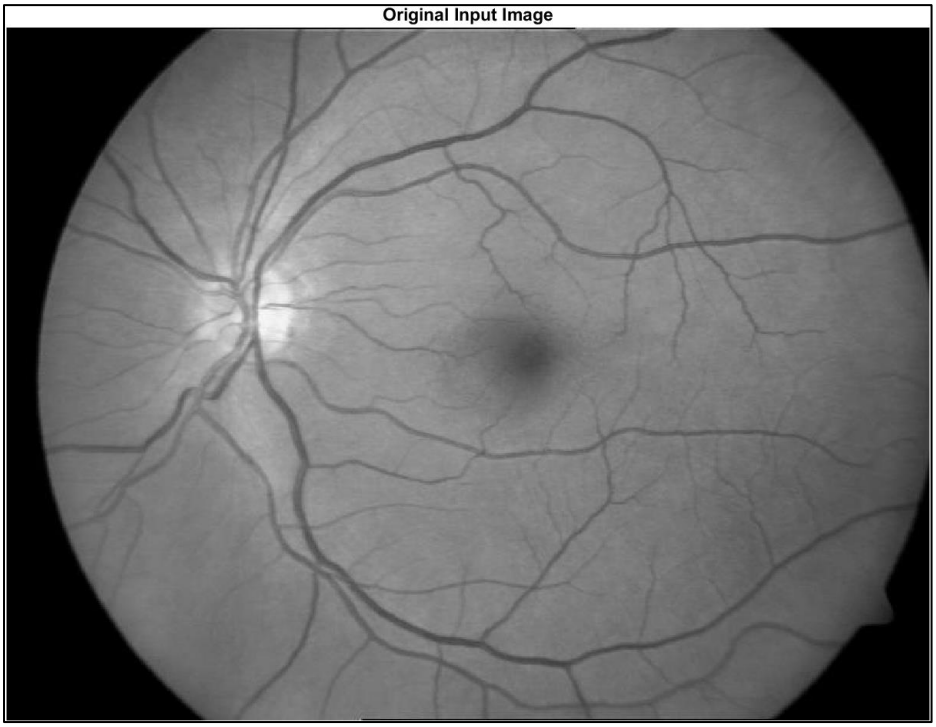
The results show that when they noise is varied and uses the same number of averages; the image becomes less clear. Noticeable is also the level of increased brightness in the images.

3C)

Code:

```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 3.3 C

%Clearing previous results
close all
clear all
clc

%getting our image and displaying it
x = imread('eye.jpg');
figure
imshow(x);
title('Original Input Image');

x1 = imnoise(x,'salt & pepper');
figure
imshow(x1)
title('Salt and Pepper Noise')

x1 = imdivide(x1, 10);
x2 = imnoise(x, 'salt & pepper');
x2 = imdivide(x2, 10);
x3 = imnoise(x, 'salt & pepper');
x3 = imdivide(x3, 10);
x4 = imnoise(x, 'salt & pepper');
x4 = imdivide(x4, 10);
x5 = imnoise(x, 'salt & pepper');
x5 = imdivide(x5, 10);
x6 = imnoise(x, 'salt & pepper');
x6 = imdivide(x6, 10);
x7 = imnoise(x, 'salt & pepper');
x7 = imdivide(x7, 10);
x8 = imnoise(x, 'salt & pepper');
x8 = imdivide(x8, 10);
x9 = imnoise(x, 'salt & pepper');
x9 = imdivide(x9, 10);
x10 = imnoise(x, 'salt & pepper');
x10 = imdivide(x10, 10);

y = imadd(x1, x2);
y = imadd(y, x3);
y = imadd(y, x4);
y = imadd(y, x5);
y = imadd(y, x6);
y = imadd(y, x7);
y = imadd(y, x8);
y = imadd(y, x9);
y = imadd(y, x10);
```

```
figure
imshow(y)
title('Averaged 10 times with salt and pepper')
```



Averaged 10 times with salt and pepper

We can say that image averaging is effective for noise reduction on salt and pepper type noise.

3D)

Code:

```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 3.3 D

%Clearing previous results
close all
clear all
clc

%getting our image and displaying it
x = imread('eye.jpg');
figure
imshow(x);
title('Original Input Image');

%Generating Salt and Pepper Noise
x1 = imnoise(x,'salt & pepper');
figure
imshow(x1)
title('Salt and Pepper Noise')

%Using medfilt2
x1 = medfilt2(x1);
figure
imshow(x1)
title('Salt and Pepper Noise Reduction Using medfilt2')
```

Output Results Using medfilt2

Original Input Image

Salt and Pepper Noise

We can clearly see that medfilt2 had the cleanest and clearest result between noise reduction methods. Averaging is still good but it requires a lot more work to perform.