
*College of IT and Computer Eng.*

*Supervised by:*

*Dr.Mousa Farajallah*

*Done By:*

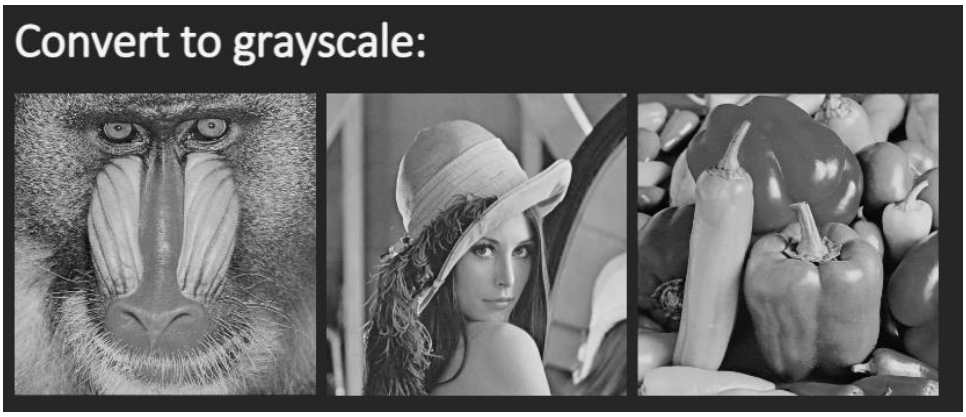
*Mays Dababsa*

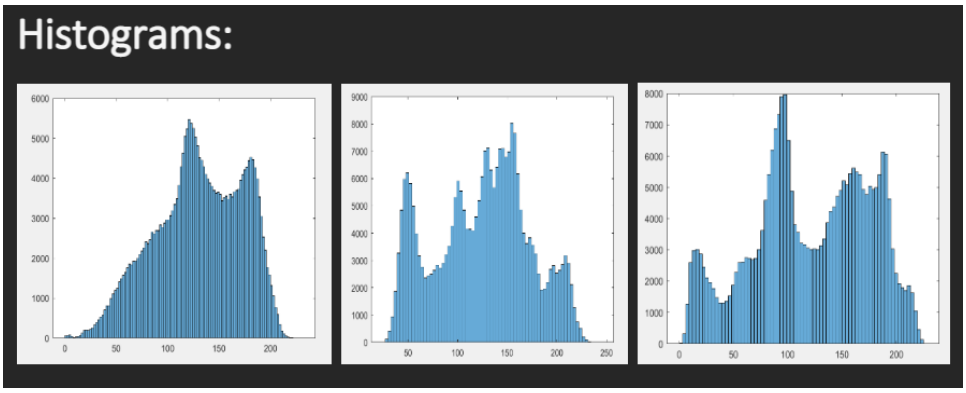
*Shatha Awawdh*

# *Compare Between DCT and DWT for Digital Watermarking*

Original images and their histograms:

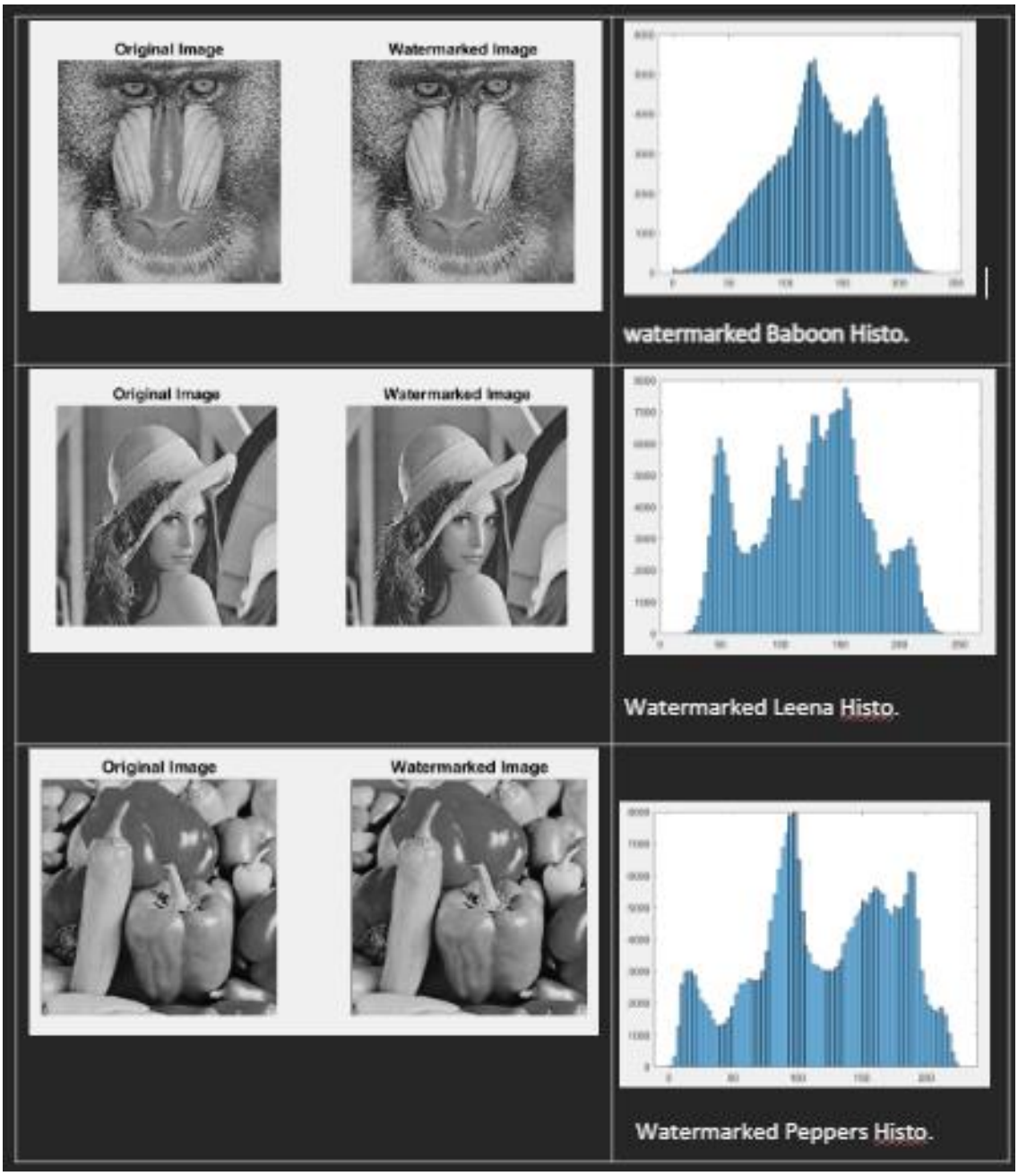
Baboon Lena Pappers



## STEP#1

Watermarked images compared to the original:




## Table1:

PSNR, MSE for the Watermarked Image After DWT

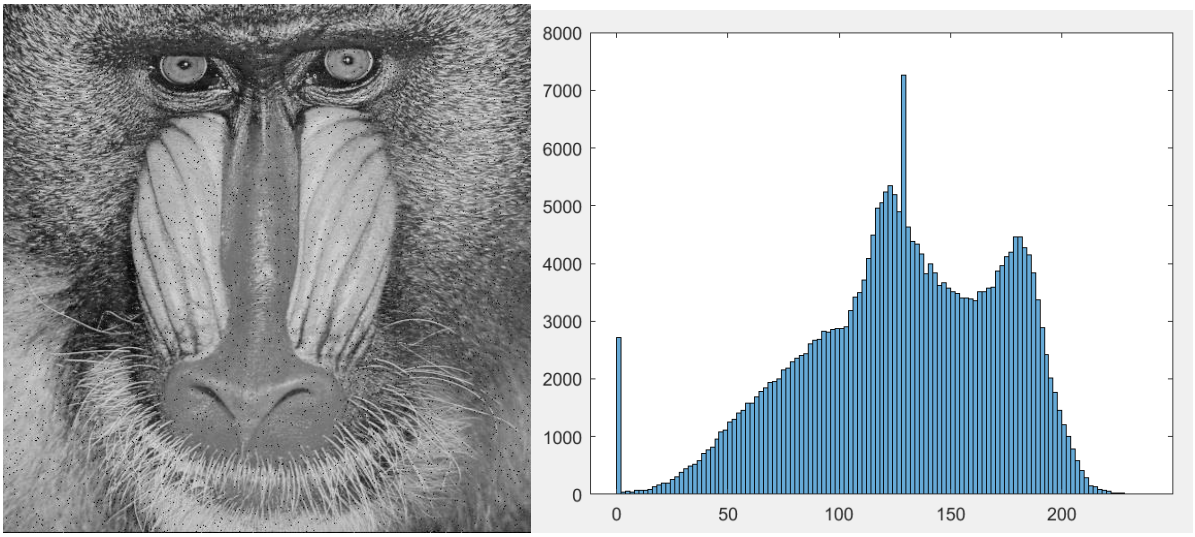
| DWT (db units) | Leena | Baboon | Papper |
|---|---|---|---|
| PSNR | 39.791503 | 39.323942 | 39.148246 |
| SME | 0.123333 | 0.126319 | 0.125972 |

# STEP#2

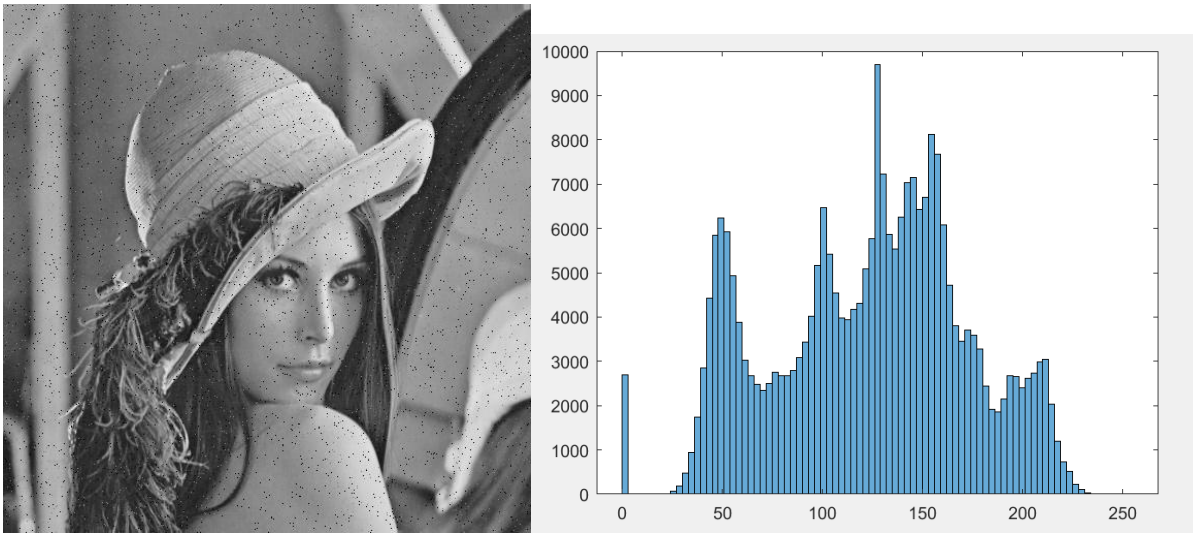
# Watermarked images after the salt & peppet noise:

- **DWT**

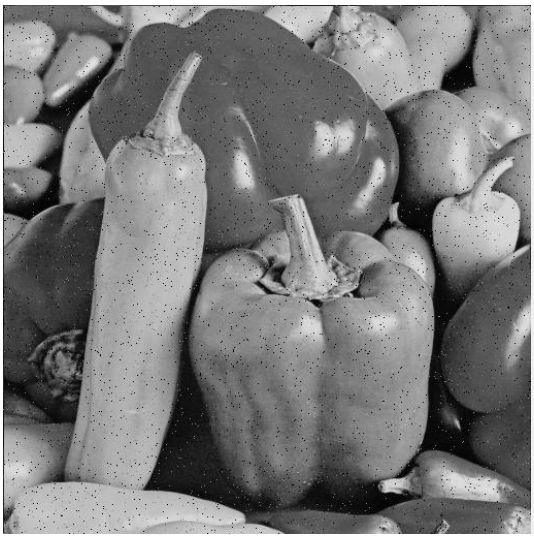
**Noisy Baboon**

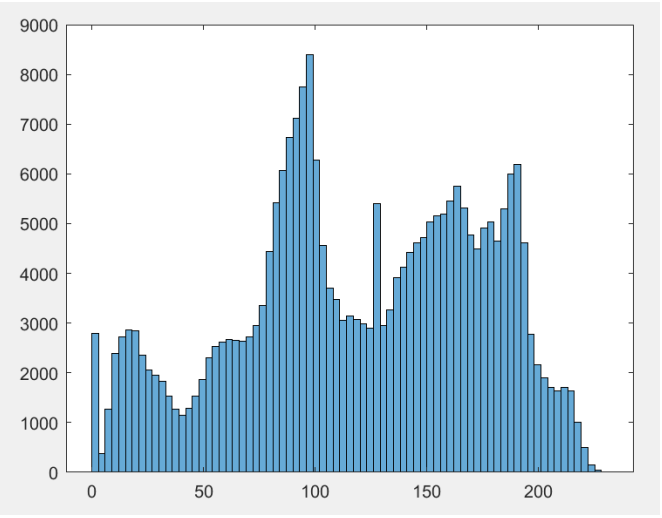



**Noisy Lena**



## Noisy Pepper






# Table2

| DWT/NC | Salt & Pepper |
|---|---|
| Leena | 0.316540 |
| Baboon | 0.316540 |
| Papper | 0.316540 |

# Table3

| DWT/SIM | SIM |
|---|---|
| Leena | 0.751131 |
| Baboon | 0.748344 |
| Papper | 0.756747 |

**Embeding and extraction**

```
%DWT Based Robust Spread Spectrum Watermarking

[filename1, pathname1]=uigetfile('*.*', 'Select the Cover Image');

coverimage=imread(num2str(filename1));
CI=coverimage; %uint8 image
coverimage=double(coverimage);
[filename2, pathname2]=uigetfile('*.*', 'Select the Watermark Image');
watermark=imread(num2str(filename2));
WMO=watermark; %binary water mrk


N=1; %Decomposition Levels
L=2^N; %Dimension reduction by L at level N
wavetype='bior6.8'; %Wavelet Type

K=0.05; %Embedding Strength


%Determine size of cover image
[Mc, Nc]=size(coverimage);

%Determine the size of watermark
[Mwmo,Nwmo]=size(watermark);
wmvector=reshape(watermark,Mwmo*Nwmo,1);

%----------Watermark Embedding----------
```

```
key=1000; %Initialize Key
rng(key); %Reset PN generator to state "key"
pnsequence=round(2*(rand(Mc/L, Nc/L)-0.5)); %Generate PRN Sequence

dwtmode ('per') %Setting Wavelet Decomposition Mode
[C1, S1]= wavedec2 (coverimage, N, wavetype); %DWT of Image
cA1=appcoef2(C1, S1, wavetype,N);
[cH1, cV1, cD1]=detcoef2('all',C1,S1,N);

%Adding PRN sequences to CD1 components when watermark bit = 0
for i=1:length(wmvector)
    if wmvector(i)== 0
      cD1=cD1+K*pnsequence;
    end
pnsequence=round(2*(rand(Mc/L,Nc/L)-0.5));
end

x=size(cA1,1); y=size(cA1,2);
cA1row=reshape(cA1,1,x*y); cH1row=reshape(cH1,1,x*y);
cV1row=reshape(cV1,1,x*y); cD1row=reshape(cD1,1,x*y);
cc=[cA1row,cH1row,cV1row,cD1row];
ccl=length(cc); C1(1:ccl)=cc;

watermarked_image=waverec2(C1,S1,wavetype); %IDWT
watermarked_image_uint8=uint8(watermarked_image);

%watermarked_image_uint8=imnise(watermarked_image_uint8,'salt & pepper',0.02);

imwrite(watermarked_image_uint8,'dwt_watermarked.jpg','quality',100);
```

```
%Calculate PSNR
mse = sum(sum((watermarked_image-coverimage).^2 ))/(Mc*Nc);

maxp = max(max(watermarked_image(:)), max(coverimage (:)));
PSNR = 10*log10((maxp^2)/mse);
fprintf('\nPSNR = %f\n',PSNR);

%Display Original and Watermarked image
figure(1)
subplot(121)
imshow(CI); title('Original Image')
subplot (122)
imshow(watermarked_image_uint8); title('Watermarked Image')


%------------------------Watermark Recovery----------------------

file_name='dwt_watermarked.jpg';
watermarked_image=double(imread(file_name)); %Reading watermarked image
[Mw,Nw]=size(watermarked_image); %Determine size of watermarked image
wmvectorr=ones(1,Mwmo*Nwmo); %Initialize watermark to all bits=1

[C2,S2]= wavedec2(watermarked_image,N,wavetype); %DWT
cD2=detcoef2('d',C2,S2,N);

key=1000;
rng(key); %Reset PN generator to state "key"
```

```
pnsequence=round(2*(rand(Mw/L,Nw/L)-0.5)); %Generate PRN Sequence

%Add PRN sequences to CDs coeffs. when watermark bit= 0
for i=1:length(wmvectorr)
    correlation(i)=corr2(cD2,pnsequence);
    pnsequence=round(2*(rand(Mw/L,Nw/L)-0.5));
end

T=mean(correlation); %T=T; %Finding Threshold for WM recovery
Tvec=T*ones(1,length(correlation));
figure(2); plot(correlation); hold on; plot (Tvec);
title('Correlation Pattern'); hold off;

for i=1:length (wmvectorr)
    if correlation(i)>T %Comparing correlation with Threshold
      wmvectorr(i)=0;
    end
end

WMR=reshape(wmvectorr,Mwmo,Nwmo); %Recovered Watermark

figure (3)
subplot (121)
imshow(WMO); title('Original Watermark')
subplot (122)
imshow(WMR); title('Recovered Watermark')
imwrite(uint8(WMR), 'recovered_watermark.jpg');
```

Noise

%????

```
originalImage = imread('dwt_watermarked.jpg');

% Adding salt and pepper noise with a degree of 0.02
noisyImage = imnoise(originalImage, 'salt & pepper', 0.02);

% Saving the noisy image
imwrite(noisyImage, 'PeppersRGB_noisy.bmp');

% Displaying the original and noisy images side by side
figure;
subplot(1, 2, 1);
imshow(originalImage);
title('Original Image');
subplot(1, 2, 2);
imshow(noisyImage);
title('Noisy Image');
```

**MSE**

```
original = imread('watermark.jpg');
recovered = imread('recovered_watermark.jpg');

% Convert the images to double precision for accurate calculations
original = im2double(original);
recovered = im2double(recovered);

% Calculate the squared difference between the original and recovered watermarks
squared_diff = (original - recovered) .^ 2;

% Calculate the MSE as the mean of the squared differences
mse = mean(squared_diff(:));

% Display the MSE
fprintf('Mean Squared Error (MSE): %f\n', mse);
```

**NC**

```
% Load original and recovered watermarks as grayscale images
original = imread('watermark.jpg');
recovered = imread('recovered_watermarkNoisePepper.jpg');

% Convert the images to double precision for accurate calculations
original = im2double(original);
recovered = im2double(recovered);

% Calculate the mean intensity of the original and recovered watermarks
mean_original = mean(original(:));
mean_recovered = mean(recovered(:));

% Calculate the normalized cross-correlation (NC) between the original and recovered watermarks
numerator = sum((original - mean_original) .* (recovered - mean_recovered), 'all');
denominator = sqrt(sum((original - mean_original) .^ 2, 'all') * sum((recovered - mean_recovered) .^ 2, 'all'));
nc = numerator / denominator;

% Display the NC
fprintf('Normalized Correlation (NC): %f\n', nc);
```

**SIM**

```
% Load original and extracted watermarks as grayscale images
original = imread('watermark.jpg');
extracted = imread('recovered_watermarkNoisePepper.jpg');

% Convert the images to double precision for accurate calculations
original = im2double(original);
extracted = im2double(extracted);

% Calculate the numerator of the SIM equation
numerator = sum(sum(original .* extracted));

% Calculate the denominator of the SIM equation
denominator = sqrt(sum(sum(original .^ 2)) * sum(sum(extracted .^ 2)));

% Calculate the SIM
sim = numerator / denominator;

% Display the SIM
fprintf('SIM (Similarity Index Measure): %f\n', sim);
```