Advanced Image Analysis
Lab 2 Report: Image Compression/Quantization

Mohammad Rami Koujan
M.Sc. VIBOT
Heriot-Watt University

October 07, 2016

1 Introduction

The aim of this lab assignment is to probe both DCT (Discrete Cosine Transform) and DWT (Discrete Wavelet Transform), analyse their properties, and study their influences on compression. Moreover, using quantization while performing compression is another aspect that was looked into during this lab session.

2 DCT and Inverse DCT

A discrete cosine transform (DCT) expresses a sequence of finitely many data points in terms of a sum of cosine functions oscillating at different frequencies. DCTs are important to numerous applications in science and engineering, from lossy compression of images (where small high-frequency components can be discarded). Figure 1 shows an example of applying a DCT on an arbitrary image and then recovering it using the inverse DCT transform. Inspecting the DCT image reveals that the image is sparse in the DCT domain, which is one of the nice properties of this transformation that makes it interesting in some applications.

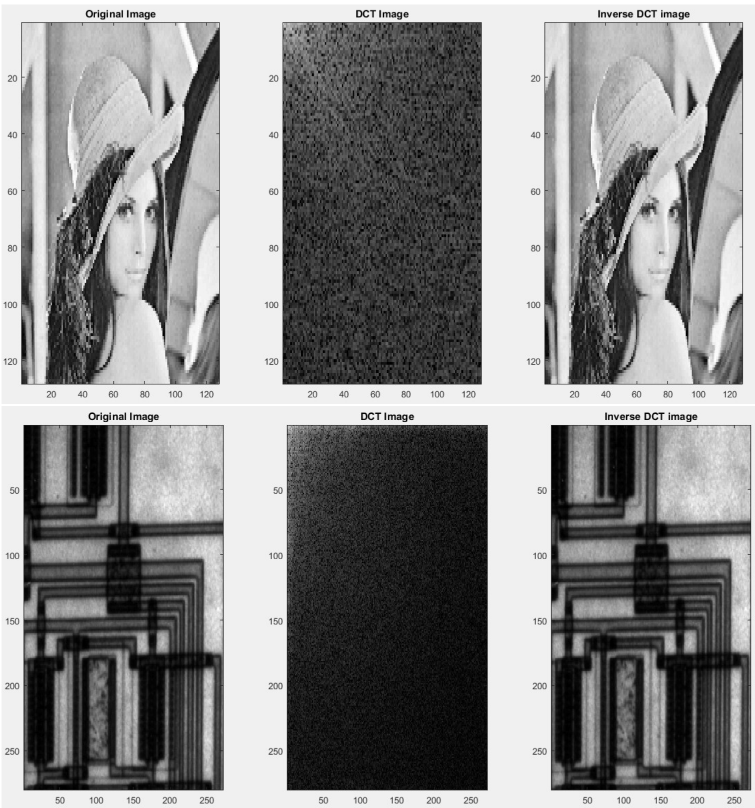

Figure 1: Applying DCT and inverse DCT on two distinct image

3 2D Haar Wavelet Transformation and Inverse Wavelet Transformation

In this section, a number of procedures were implemented in Matlab program during the lab session to perform 2D HWT and IHWT, and the built functions were tested on some images to visualize the effect of applying such a transform on 2D images. The Haar wavelet transform can be represented in a matricial form as:

$$W = \begin{bmatrix} H \\ G \end{bmatrix}$$

where H is a low pass filter with two Fourier coefficients and G is a high pass filter with two coefficients as well. The Haar low pass filter is given by:

$$H = (h0, h1) = \left[1/\sqrt{2}, 1/\sqrt{2}\right]$$

And the Haar high pass filter is given by:

$$H = (h0, h1) = \left[-1/\sqrt{2}, 1/\sqrt{2}\right]$$

By applying the Haar wavelet transform for a 2D image, the transformed image would be:

$$B = W_m * A * W_n^T$$

Where A is the input image of size (M, N), W_m is an M*M transformation matrix used to process the rows, W_n is N*N matrix used to process the columns and finally, B is the 2D Haar Wavelet Transform of the input image. Expanding the last equation gives:

$$B = W_m * A * W_n^T = \begin{bmatrix} H \\ G \end{bmatrix} A \left[H^T G^T\right] = \begin{bmatrix} HAH^T HAG^T \\ GAH^T GAG^T \end{bmatrix}$$

Where: HAH^T is the approximation sub-image, HAG^T represents the vertical details sub-image, GAH^T represents the horizontal details sub-image, and GAG^T represents the diagonal details sub-image.

3.1 HWT & IHWT Implementation

The implementation has been, indeed, split into three different functions to generate the $W_m \& W_n$ matrices, and apply the forward and inverse transforms.

```
function [Wm]=HWT_matrix(m)
    W1=zeros(m/2,m);
    W1(1,1:2)=[1,1];
    for i=2:m/2
        W1(i,:)=circshift(W1(i-1,:),2,2);
    end

    W2=zeros(m/2,m);
    W2(1,1:2)=[-1,1];
    for i=2:m/2
        W2(i,:)=circshift(W2(i-1,:),2,2);
    end

    Wm=double([W1;W2]);
    Wm=Wm/sqrt(2);
end
```

The function above is the one used to generate the HWT matrix by defining the first line of each filter and then doing double circular shifting to the right between each row and the other. Basically, this function was invoked by another two functions to perform the HWT & IHWT as follows:

```
function [hwt_im]=HWT(im,nIter)
    im=double(im);
    [m,n]=size(im);
    for c=1:nIter
        [Wm]=HWT_matrix(m);
        [Wn]=HWT_matrix(n);
        hwt_im(1:m,1:n)=Wm*im*Wn';
        m=m/2;
        n=n/2;
        im=hwt_im(1:m,1:n);
    end
    figure;imagesc(hwt_im);colormap(gray);title('HWT image');
end
```

```
function [inv_im]=IHWT(hwt_im,nIter)
    for c=1:nIter
        m=size(hwt_im,1)/2^(nIter-c);
        n=size(hwt_im,2)/2^(nIter-c);
        [Wm]=HWT_matrix(m);
        [Wn]=HWT_matrix(n);
        inv_im=Wm'*hwt_im(1:m,1:n)*Wn;
        hwt_im(1:m,1:n)=inv_im;
    end
    figure;imagesc(inv_im);colormap(gray);title('Inverse HWT image');
end
```

Figure 2 presents the results of calling the previous two functions on two dissimilar images, where nIter, number of iterations, was set to 1.

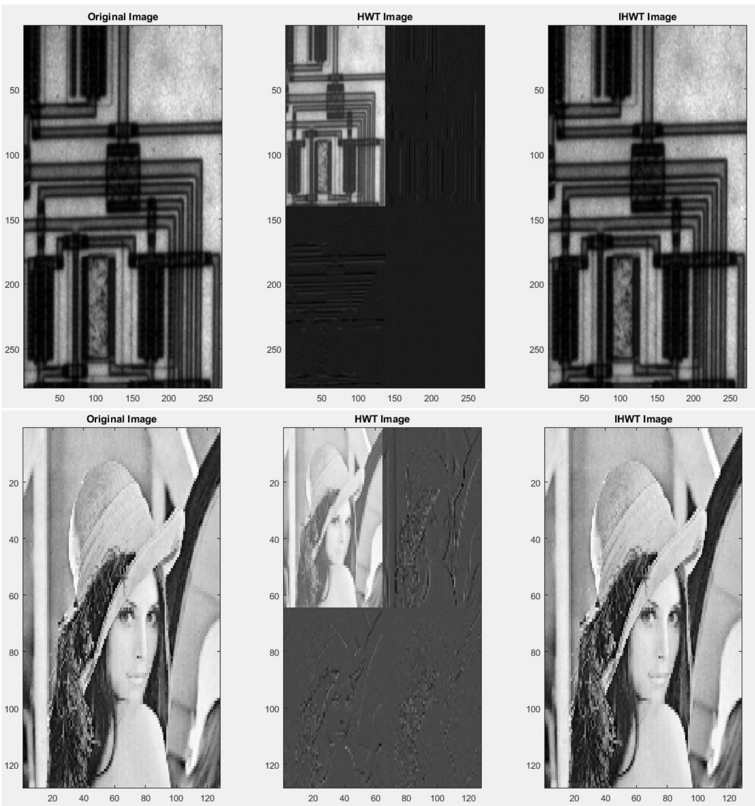

Figure 2: Applying HWT and inverse HWT on two distinct images

4 Daubechies and Biorthogonal wavelets

Although Haar filters are short , which means they lead to fast algorithm for computation, there is a disadvantage with such short filters. The problem is that those kinds of filters do not process enough data of the given signal at a time to catch jumps. Daubechies described a family of orthogonal low-pass filters. The first member of this family is the Haar filter h. Daubechies also showed how to construct other family members of an arbitrary even length and their accompanying high-pass filters. In this section, DB of length 4 and 6 were implemented and tested on some images. The following three functions demonstrate the implementation for DB4 and DB6 with only one iteration, though the functions can work for any number of iterations. The results of applying those filters on **Lena** image are shown in 3 .

```
function [Wm]=DWT_matrix(m,coff)
    % m is the number of rows/columns of the image of interest
    % There is a function in Matlab for Daubechies Coefficients: dbaux(N,sqrt(2))
    W1=zeros(m/2,m);
    if(coff==4)
        h=[(1-sqrt(3))/(4*sqrt(2)),(3-sqrt(3))/(4*sqrt(2)),(3+sqrt(3))/(4*sqrt(2)),(1+sqrt(3))/(4*sqrt(2))];
        W1(1,1:4)=h;
    elseif(coff==6)
        h=[0.332671,0.806892,0.459878,-0.135011,-0.085441,0.035226];
        W1(1,1:6)=h;
    end

    for i=2:m/2
        W1(i,:)=circshift(W1(i-1,:),2,2);
    end
    W2=zeros(m/2,m);
    if(coff==4)
        W2(1,1:4)=[h(4),-h(3),h(2),-h(1)];
    elseif(coff==6)
        W2(1,1:6)=[h(6),-h(5),h(4),-h(3),h(2),-h(1)];
    end
    for i=2:m/2
        W2(i,:)=circshift(W2(i-1,:),2,2);
    end

    Wm=double([W1;W2]);
end
```

```
function [dwt_im]=DWT(im,nIter,coff)
    im=double(im);
    [m,n]=size(im);
    for c=1:nIter
        [Wm]=DWT_matrix(m,coff);
        [Wn]=DWT_matrix(n,coff);
        dwt_im(1:m,1:n)=Wm*im*Wn';
        m=m/2;
        n=n/2;
        im=dwt_im(1:m,1:n);
    end
    figure;imagesc(dwt_im);colormap(gray);title('DWT image');
end
```

```
function [inv_im]=IDWT(dwt_im,nIter,coff)
    for c=1:nIter
        m=size(dwt_im,1)/2^(nIter-c);
        n=size(dwt_im,2)/2^(nIter-c);
        [Wm]=DWT_matrix(m,coff);
        [Wn]=DWT_matrix(n,coff);
        inv_im=Wm'*dwt_im(1:m,1:n)*Wn;
        dwt_im(1:m,1:n)=inv_im;
    end
    figure;imagesc(inv_im);colormap(gray);title('Inverse DWT image');
end
```

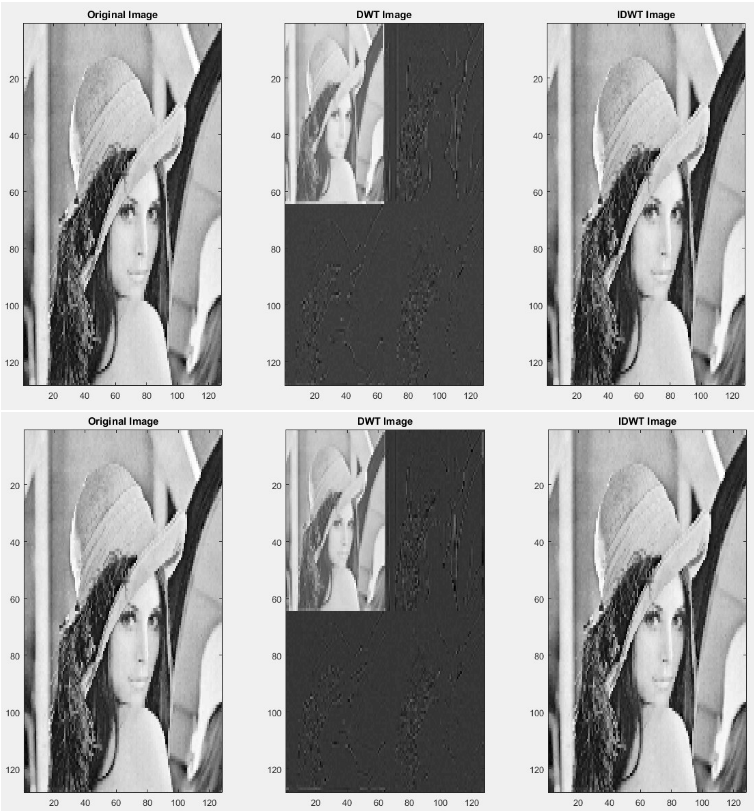

Figure 3: Top: Testing **Lena** image with DB4 and inverse DB4. Bottom: Testing **Lena** image with DB6 and inverse DB6

The previous two images exhibit somewhat similar results when applied to Daubechies with different length. Thereafter, an implementation for Biorthogonal filters were accomplished and tested as shown below:

```
function [Wm]=BWT_matrix(m)
    W1=zeros(m/2,m);
    W1(1,1:3)=[3/8,1/4,-1/8];
    W1(1,end-1:end)=[-1/8,1/4];
    for i=2:m/2
        W1(i,:)=circshift(W1(i-1,:),2,2);
    end
    W2=zeros(m/2,m);
    W2(1,1:3)=[1/4,-1/2,1/4];
    for i=2:m/2
        W2(i,:)=circshift(W2(i-1,:),2,2);
    end
    Wm=double(sqrt(2)*[W1;W2]);
end
```

```
function [Wm]=BWT_matrix_tilde(m)
    W1=zeros(m/2,m);
    W1(1,1:2)=[1/2,1/4];
    W1(1,end)=1/4;
    for i=2:m/2
        W1(i,:)=circshift(W1(i-1,:),2,2);
    end
    W2=zeros(m/2,m);
    W2(1,1:4)=[1/4,-3/4,1/4,1/8];
    W2(1,end)=1/8;
    for i=2:m/2
        W2(i,:)=circshift(W2(i-1,:),2,2);
    end
    Wm=double(sqrt(2)*[W1;W2]);
end
```

```matlab
function [bwt_im]=BWT(im,nIter)
    im=double(im);
    [m,n]=size(im);
    for c=1:nIter
        [Wm]=BWT_matrix_tilde(m);
        [Wn]=BWT_matrix(n);
        bwt_im(1:m,1:n)=Wm*im*Wn';
        m=m/2;
        n=n/2;
        im=bwt_im(1:m,1:n);
    end
    figure;imagesc(bwt_im);colormap(gray);title('BWT image');
end
```

```matlab
function [inv_im]=IBWT(bwt_im,nIter)
    for c=1:nIter
        m=size(bwt_im,1)/2^(nIter-c);
        n=size(bwt_im,2)/2^(nIter-c);
        [Wm]=BWT_matrix(m);
        [Wn]=BWT_matrix_tilde(n);
        inv_im=Wm'*bwt_im(1:m,1:n)*Wn;
        bwt_im(1:m,1:n)=inv_im;
    end
    figure;imagesc(inv_im);colormap(gray);title('Inverse BWT image');
end
```

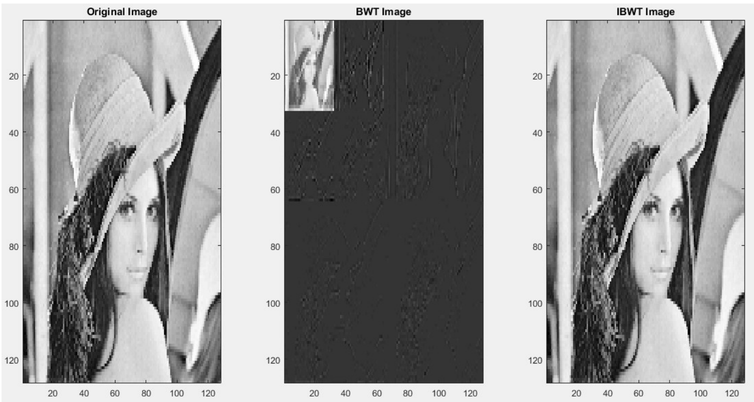

Figure 4: Testing **Lena** image with Biorthogonal and inverse Biorthogonal filters

In the next two figures, DWT and DCT are tested with two iterations instead of 1. As it can be seen, using more than one iteration ,in case of DWT, gives more sparse signal. However, for DCT, applying more than one iteration gives an image which looks similar to the original one since the forward kernel of the transformation was used twice. However, reapplying the IDCT twice reconstruct the original signal correctly.

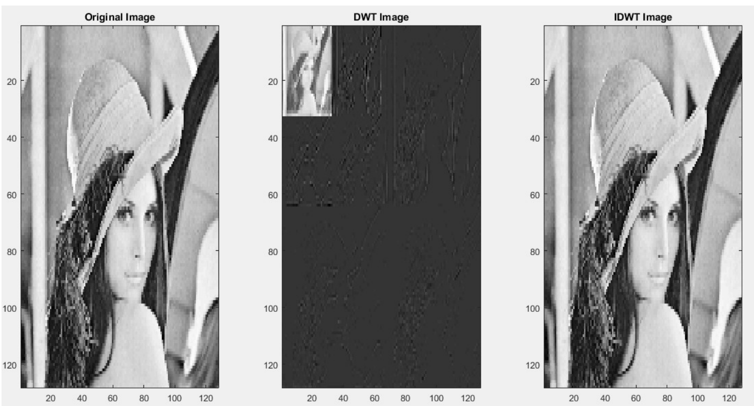

Figure 5: Applying 2 iterations of DWT on **Lena** image

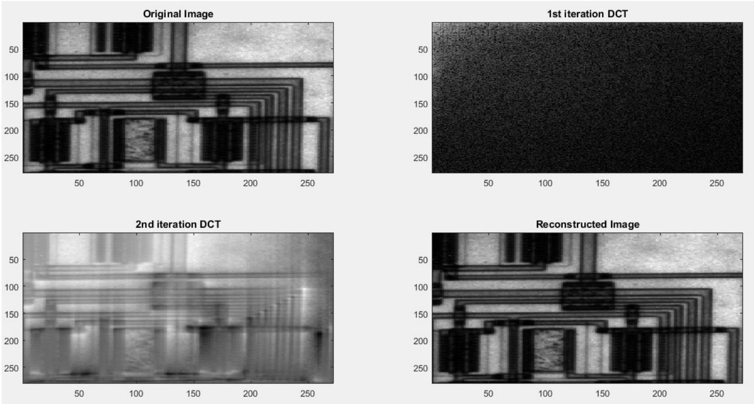

Figure 6: Applying 2 iterations of DCT on **Lena** image

5 Image Compression using DCT and DWT Wavelets including quantization

This section in mainly about using different DWT transformations and DCT in order to test and evaluate their compression performance by using MSE (Mean Squared Error) and PSNR (Peak Signal to Noise Ratio) metrics. Two kinds of quantization algorithms were used, naive and optimal (minimizes the MSE) , to quantize the images in the transformed domain before converting them back to the spatial domain. Table 1 presents different results, in terms of MSE and PSNR, of using distinct transformations, where n is the number of quantization levels used given that the results in this table are for naive quantization algorithm. The following steps were taken into account while performing the compression:

1. First of all, the image was transformed using one of the above mentioned transformations
2. Then, the transformed image was quantized which means mapping all the inputs within a specific range to a certain value. The quantization was done in Matlab function 'quantiz', for optimal case or by creating a new function for naive quantization case. This Matlab function requires two parameters in addition to the input image: 'codebook' for each 'partition'. These two parameters can be obtained using the built in Matlab function 'lloyds' that takes two arguments: the input image and number of levels.
3. After quantizing the image in the transformation domain, the inverse of that transformation is applied to get the compressed image in the image domain.
4. The quality of the compression was checked using the mean square error (MSE) and the peak signal to noise ratio (PSNR) where:

$$MSE = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I(i,j) - \tilde{I}(i,j)]$$
$$PSNR = 10 \log\left(\frac{MAX(I)^2}{MSE}\right)$$

n	DCT		HWT		Iterative HWT		Daubechies length 4		Daubechies length 6	
	MSE	PSNR	MSE	PSNR	MSE	PSNR	MSE	PSNR	MSE	PSNR
10	1108.4	15.174	58.909	27.919	102.75	25.503	56.253	28.119	57.935	27.991
20	692.49	17.217	20.912	32.417	47.993	28.809	22.78	32.045	22.355	32.127
30	515.73	18.496	10.872	35.258	28.174	31.122	11.494	35.016	12.126	34.783
40	390.56	19.704	6.3923	37.564	18.871	32.863	7.0908	37.114	7.8308	36.683
50	320.78	20.559	4.3752	39.211	13.979	34.166	5.1277	38.521	4.963	38.663
60	263.29	21.416	3.4959	40.185	10.439	35.434	4.193	39.395	4.081	39.513
70	224.66	22.106	2.6687	41.358	8.3398	36.409	2.5959	41.478	3.3054	40.428
80	195.56	22.708	1.9404	42.742	6.2953	37.631	2.5959	41.478	2.5467	41.561
90	174.06	23.214	1.9404	42.742	5.2869	38.389	1.8696	42.903	1.8435	42.964
100	155.81	23.695	1.3084	44.453	4.3419	39.244	1.8696	42.903	1.8435	42.964
Mean	404.13	20.429	11.281	38.385	24.647	33.957	11.587	37.897	11.883	37.768

Table 1: Compression using dissimilar transformations and quantization levels with **naive** quantization

n	DCT		HWT		Iterative HWT		Daubechies length 4		Daubechies length 6	
	MSE	PSNR	MSE	PSNR	MSE	PSNR	MSE	PSNR	MSE	PSNR
10	852.51	16.314	47.293	28.873	87.736	26.189	42.152	29.373	45.957	28.997
20	312.72	20.669	14.068	34.138	21.489	32.299	11.826	34.892	12.872	34.524
30	189.66	22.841	7.5124	36.863	11.425	35.042	5.7344	38.036	5.6112	38.13
40	188.79	22.861	4.3179	39.268	6.8658	37.254	3.2076	40.559	3.6344	40.016
50	135.99	24.286	2.8753	41.034	4.7634	38.841	2.1855	42.225	2.4041	41.811
60	124.62	24.665	2.1187	42.36	3.7155	39.921	1.5938	43.596	1.6922	43.336
70	96.404	25.78	1.621	43.523	2.9446	40.93	1.1573	44.986	1.379	44.225
80	75.403	26.847	1.3395	44.351	2.5758	41.511	0.9392	45.893	1.1088	45.172
90	75.057	26.867	1.1349	45.071	2.0025	42.605	0.78156	46.691	0.89458	46.104
100	74.219	26.916	0.99574	45.639	1.7919	43.087	0.62133	47.687	0.72393	47.024
Mean	212.54	23.804	8.3277	40.112	14.531	37.768	7.0199	41.394	7.6277	40.934

Table 2: Compression using dissimilar transformations and quantization levels with **optimal** quantization

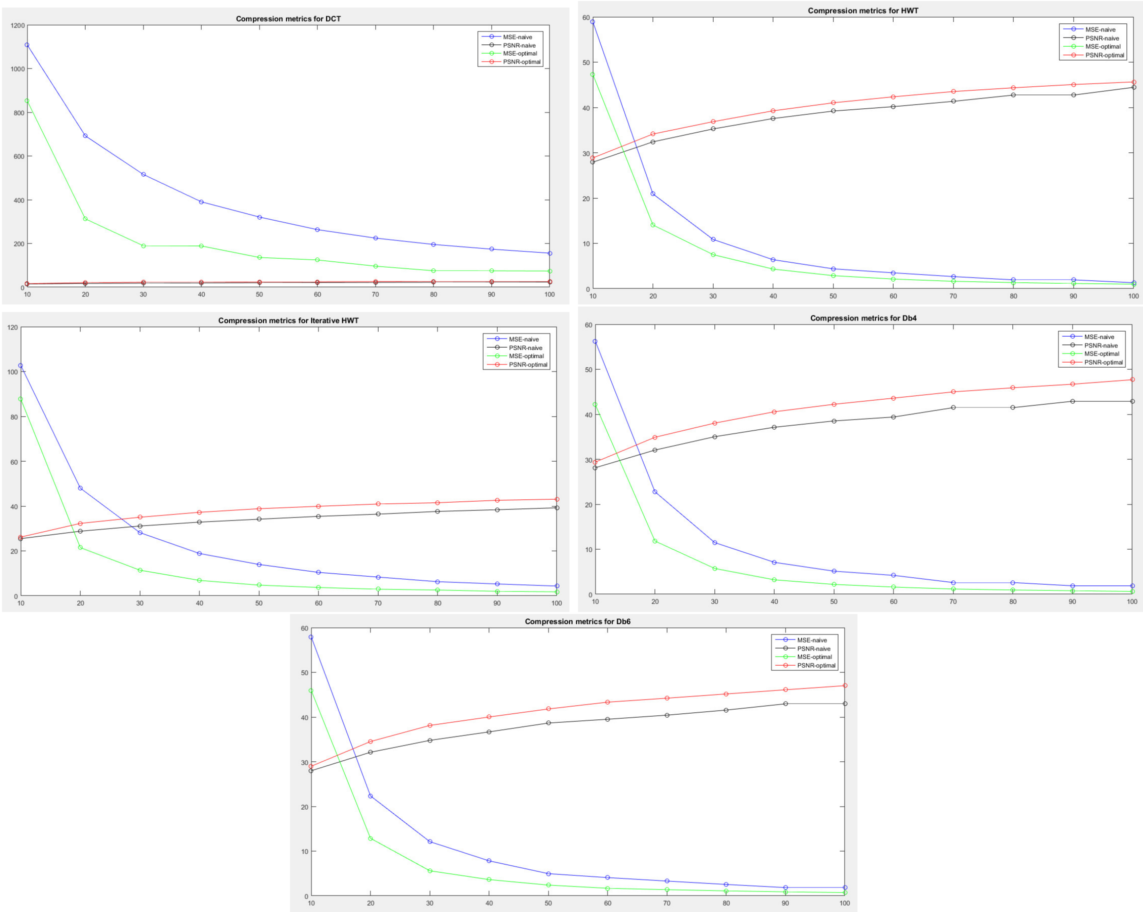

Figure 7: (results of compression in table 1 & 2). First-row: left: compression metric values for DCT, right: compression metric values for HWT. Second-row: left: compression metric values for Iterative HWT, right: compression metric values for Db4. Third-row: compression metric values for Db6

Looking into the results in table 1 & 2, and in figure 6, it can be clearly seen that increasing the number of quantization levels decreases the MSE error but raises the PSNR whether for naive or optimal quantization. Another visible trend in the shown values is that optimal quantization gives quite better results than the naive one since it chooses the quantization intervals based on minimizing the MSE. Another intriguing feature of the previous results is that WT always achieves much better results than DCT, where Daubechies of length 4 and 6 are the most remarkable ones. Figure 7 shows some of the compressed images using different transformations and quantization levels.

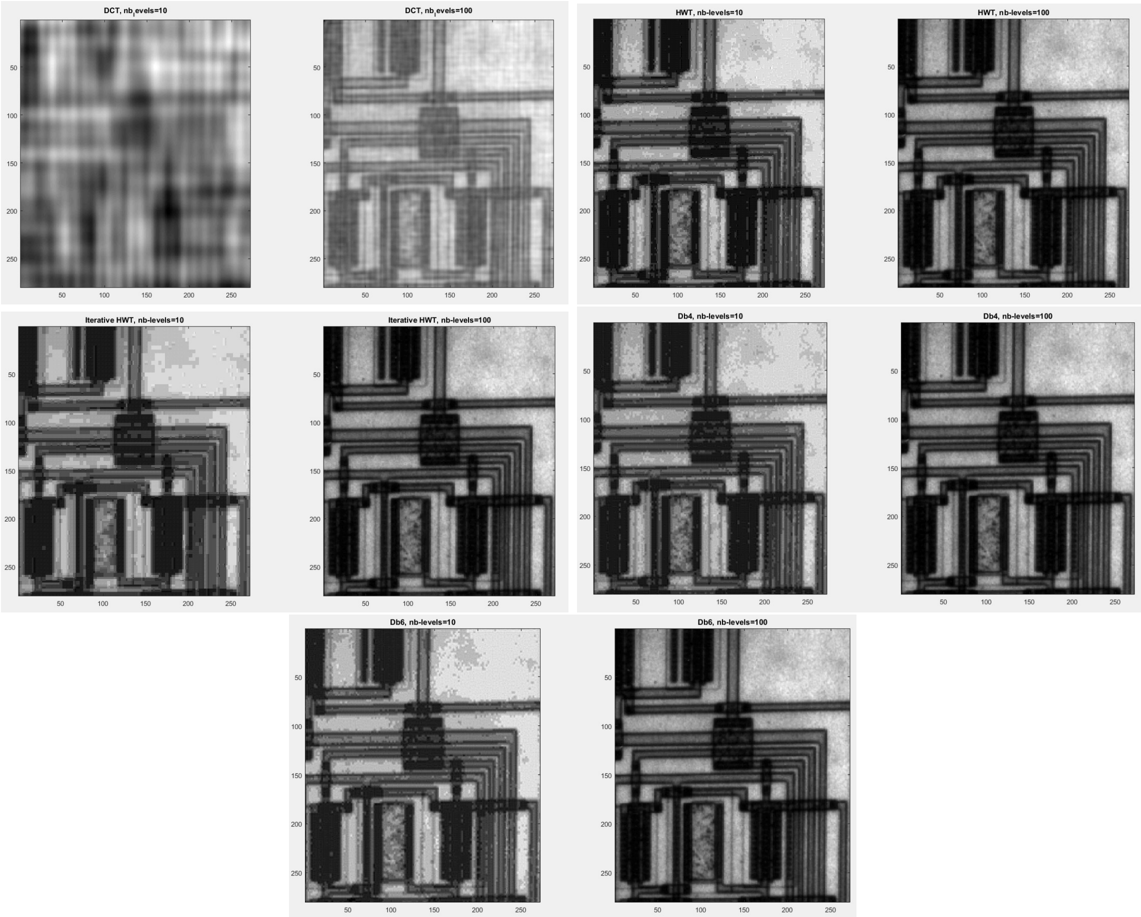

Figure 8: resultant compressed images from the experiment presented in table 1 &2 with nb_levels=10,100 and DCT, HWT, Iterative HWT, Db4, and Db6, respectively

The previous images prove what discussed earlier about the results. In other words, the DCT compressed images have the lowest quality for the minimum and maximum number of tested quantization levels. Haar and Iterative Haar (2 iterations) show better results compared to DCT but still worse than those obtained by Db4 and Db6.

6 Conclusion

During this lab assignment, 2D HWT, DWT of length 4 and 6, and BWT are implemented from scratch along with their inverse transformations and tested on different images. One important application of the mentioned transformations and the DCT is compression which was inspected in more details in this lab by compressing the 'circuit' image in the previously stated filters domains. A comparison has been drawn between those distinct transformations with regard to compression application based on two metrics, MSE &$PSNR$, and tables and figures were generated to elucidate and accentuate the differences.