EE 475/575
DIGITIAL IMAGE PROCESSING
PROJECT 5
Transform Coding

Instructor: Dennis Helder

Date: 11/25/2019

Student: Harsh Dubey

The purpose of this project is to introduce the concept of image transform coding. A comparison will be made of Fourier and Cosine transforms and their ability to preserve image quality as measured by root-mean-square-error (RMSE).

1. Write a Matlab program to compute the information loss associated with the following transform coding schemes:

	Case 1	**Case 2**
Transform:	Fourier	Cosine
Subimage Size:	8 x 8	8 x 8
Bit Allocation:	8-largest coding	8-largest coding

Calculate the amount of information loss by finding the RMSE between the original Lena image and the reconstructed Lena image obtained from your program. Compare the two RMSE values—which do you expect to be larger?

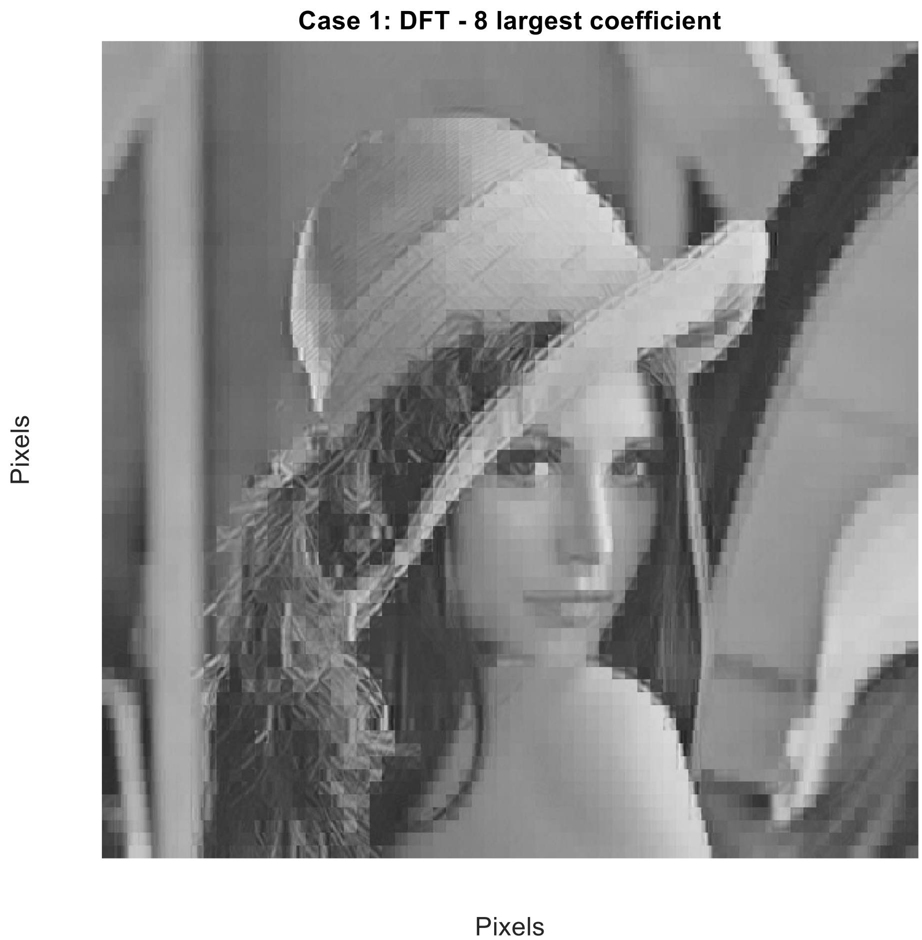

Figure 1: Case 1 : 8 x 8 Fourier transform

Figure 2: Case 2 : 8 x 8 Cosine transform

Table 1: RMSE values of both case 1 and case 2

	DFT	**DCT**
RMSE	10.10	8.86

According to the theory, we expected DFT to be larger. Which can be seen in the results too.

2. For both cases, decrease the number of retained coefficients to 7-largest, 6-largest, etc. until the reconstruction error for Case 2 becomes objectionable (a subjective criterion!). Plot the RMSE as a

function of the number of retained coefficients. How does the shape of your curves compare with theory?

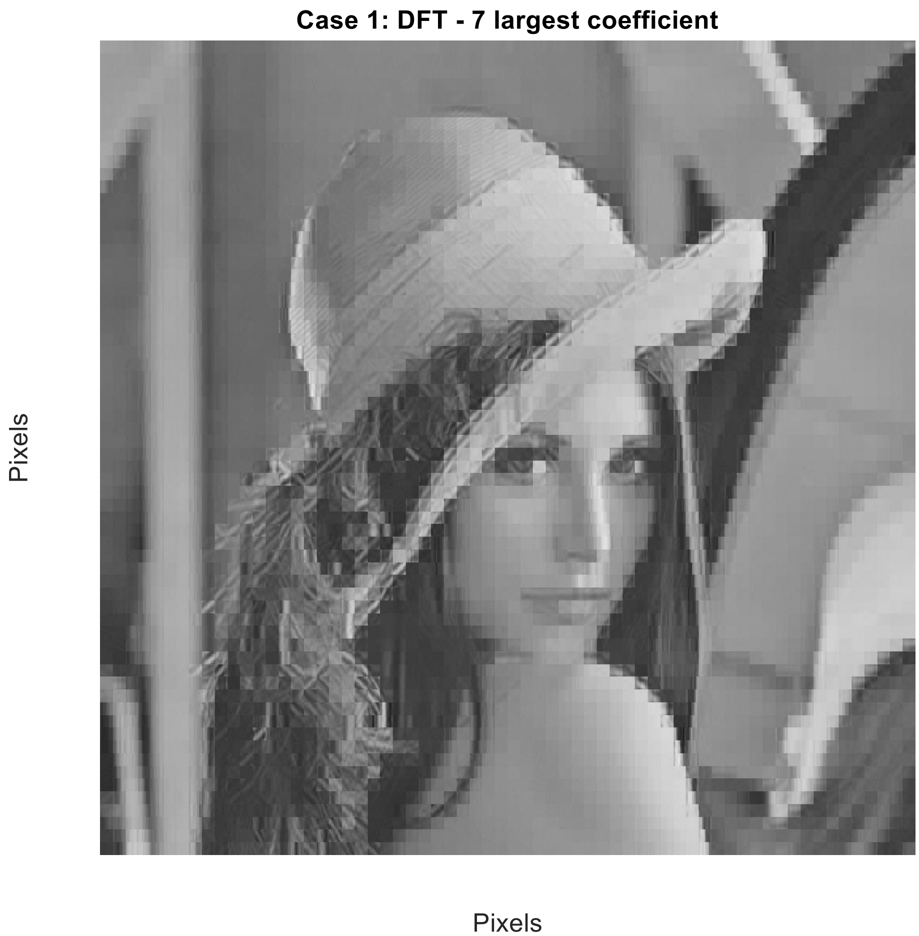

Figure 3: Case 1 : 8 x 8 Fourier transform, 7 Largest

Figure 4: Case 2 : 8 x 8 Cosine transform, 7 largest

Figure 5: Case 1 : 8 x 8 Fourier transform, 6 Largest

Figure 6: Case 2 : 8 x 8 Cosine transform, 6 largest

Figure 7: Case 1 : 8 x 8 Fourier transform, 5 Largest

Figure 8: Case 2 : 8 x 8 Cosine transform, 5 largest

Figure 9: Case 1 : 8 x 8 Fourier transform, 4 Largest

Figure 10: Case 2 : 8 x 8 Cosine transform, 4 largest

Figure 11: Case 1 : 8 x 8 Fourier transform, 3 Largest

Figure 12: Case 2 : 8 x 8 Cosine transform, 3 largest

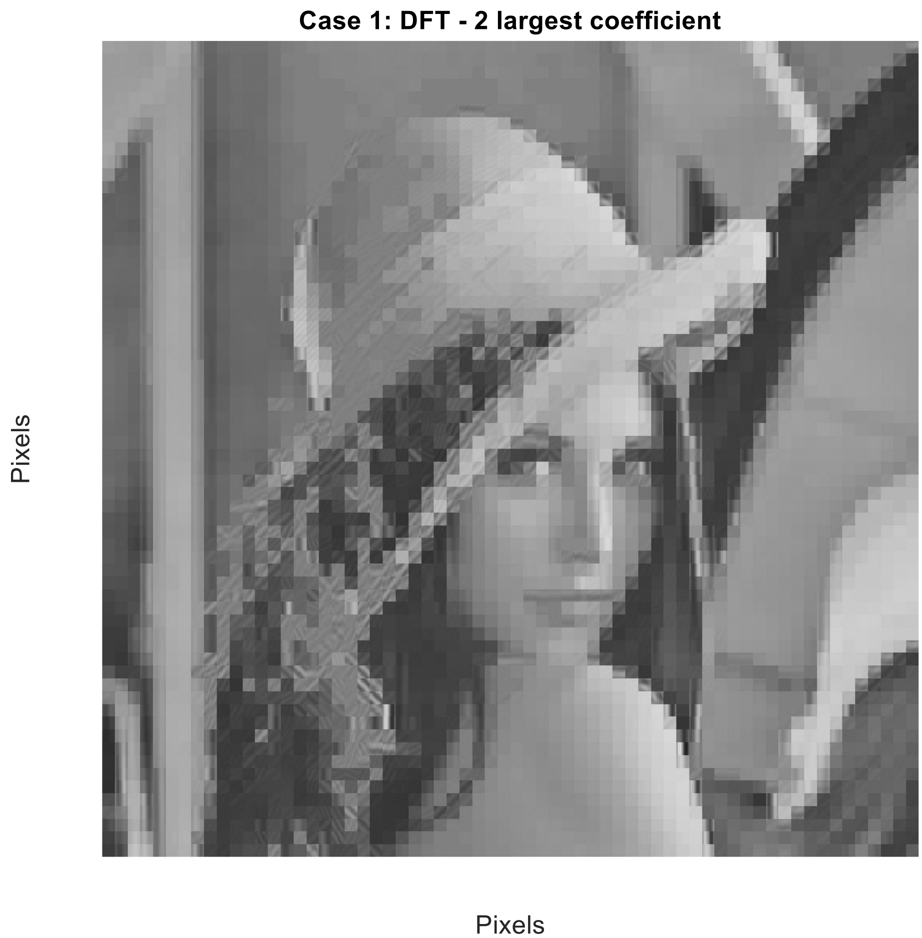

Figure 13: Case 1 : 8 x 8 Fourier transform, 2 Largest

Figure 14: Case 2 : 8 x 8 Cosine transform, 2 largest

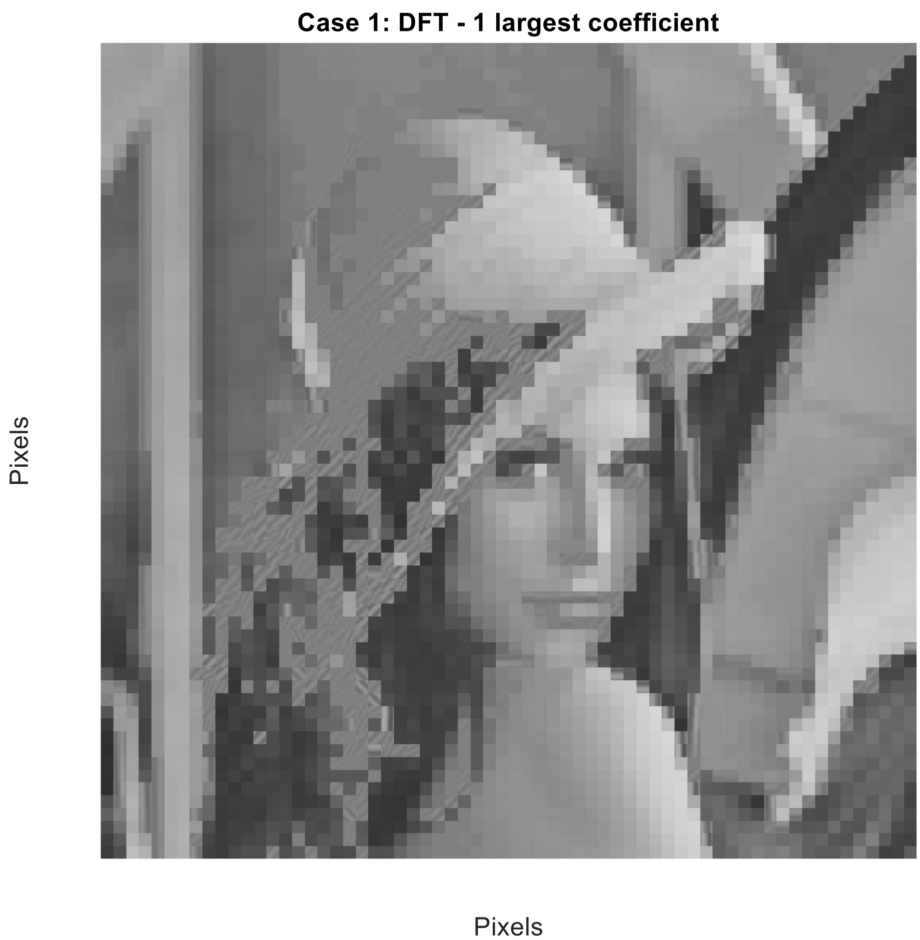

Figure 15: Case 1 : 8 x 8 Fourier transform, 1 Largest

Figure 16: Case 2 : 8 x 8 Cosine transform, 1 largest

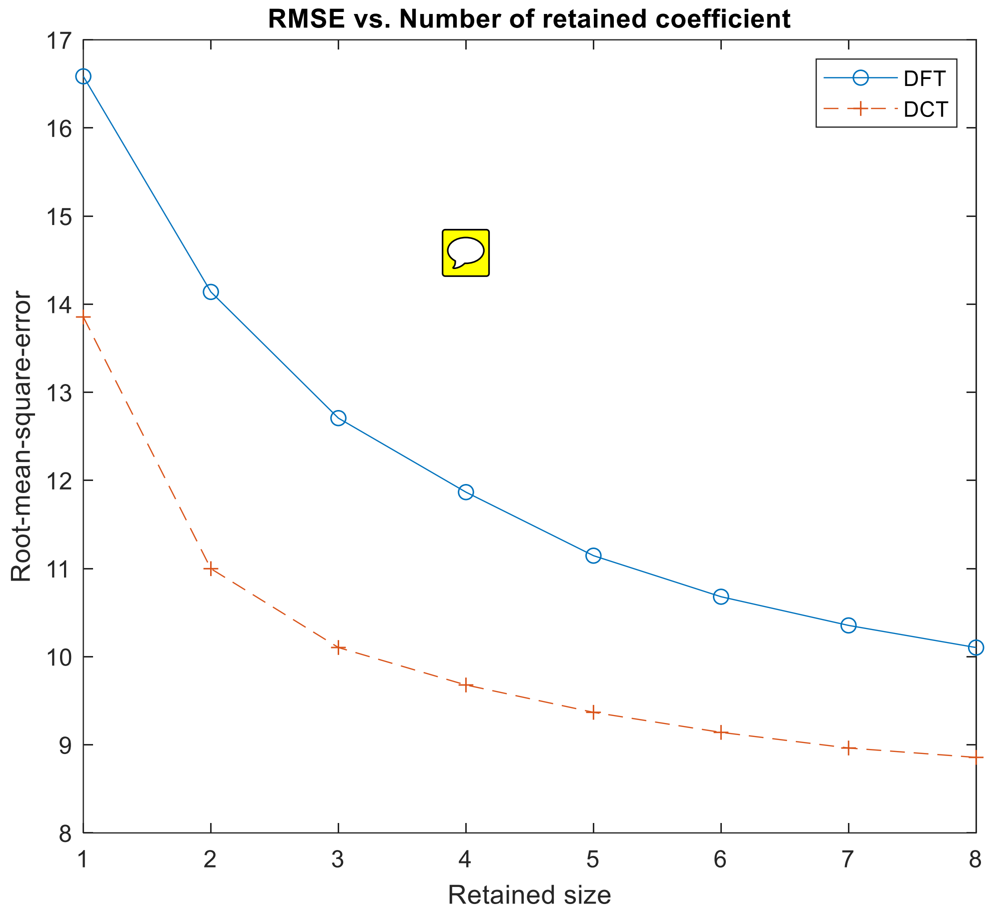

Figure 17: RMSE plot of DCT(red) vs DFT(blue)

The plot above is consistent with theory, since if we increase the number of co-efficient then we include more information and hence the RMSE should decrease.

Also, DCT's performance is better than DFT's. Which was also expected. In figure 8.26 Digital image processing book(2008, prentice hall), it can be seen that with increase in sub-image size RMSE got lesser and DCT did better than DFT.

Table 2: RMSE table

	RMSE DFT	**RMSE DCT**
8 largest	10.10	8.86
7 largest	10.35	8.96
6 largest	10.68	9.14

5 largest	11.15	9.37
4 largest	11.87	9.68
3 largest	12.71	10.10
2 largest	14.14	11.0
1 largest	16.59	13.85

Appendix

```
%Author: Harsh Dubey
%Worked with: Lin Zeng and Gagan Singla
%Citation:
https://www.mathworks.com/matlabcentral/answers/152071-
jpeg-compression-algorithm-implementation-in-matlab
%Date: 11/24/19

clear all
close all

Orignallena=double(loadraster('lena512.img',512,512));
close;

%Finding DFT from 8 largest to 1 largest
FinalDFT = DFT(8);
figure;
imshow(uint8(FinalDFT));
title('Case 1: DFT - 8 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDFT(1)=sqrt(immse(FinalDFT,Orignallena));

FinalDFT = DFT(7);
figure;
imshow(uint8(FinalDFT));
title('Case 2: DFT - 7 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDFT(2)=sqrt(immse(FinalDFT,Orignallena));

FinalDFT = DFT(6);
figure;
imshow(uint8(FinalDFT));
title('Case 3: DFT - 6 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDFT(3)=sqrt(immse(FinalDFT,Orignallena));

FinalDFT = DFT(5);
```

```
figure;
imshow(uint8(FinalDFT));
title('Case 4: DFT - 5 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDFT(4)=sqrt(immse(FinalDFT,Orignallena));

FinalDFT = DFT(4);
figure;
imshow(uint8(FinalDFT));
title('Case 5: DFT - 4 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDFT(5)=sqrt(immse(FinalDFT,Orignallena));

FinalDFT = DFT(3);
figure;
imshow(uint8(FinalDFT));
title('Case 6: DFT - 3 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDFT(6)=sqrt(immse(FinalDFT,Orignallena));

FinalDFT = DFT(2);
figure;
imshow(uint8(FinalDFT));
title('Case 7: DFT - 2 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDFT(7)=sqrt(immse(FinalDFT,Orignallena));

FinalDFT = DFT(1);
rmseDFT(8)=sqrt(immse(FinalDFT,Orignallena));
figure;
imshow(uint8(FinalDFT));
title('Case 8: DFT - 1 largest coefficient');
xlabel('Pixels');ylabel('Pixels');

%Finding DCT for 8 largest and 1 largest
FinalDCT = DCT(8);
figure;
imshow(uint8(FinalDCT));
title('Case 1: DCT - 8 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDCT(1)=sqrt(immse(FinalDCT,Orignallena));


FinalDCT = DCT(7);
```

```
figure;
imshow(uint8(FinalDCT));
title('Case 2: DCT - 7 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDCT(2)=sqrt(immse(FinalDCT,Orignallena));

FinalDCT = DCT(6);
figure;
imshow(uint8(FinalDCT));
title('Case 3: DCT - 6 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDCT(3)=sqrt(immse(FinalDCT,Orignallena));

FinalDCT = DCT(5);
figure;
imshow(uint8(FinalDFT));
title('Case 4: DCT - 5 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDCT(4)=sqrt(immse(FinalDCT,Orignallena));


FinalDCT = DCT(4);
figure;
imshow(uint8(FinalDFT));
title('Case 5: DCT - 4 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDCT(5)=sqrt(immse(FinalDCT,Orignallena));


FinalDCT = DCT(3);
figure;
imshow(uint8(FinalDFT));
title('Case 6: DCT - 3 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDCT(6)=sqrt(immse(FinalDCT,Orignallena));


FinalDCT = DCT(2);
figure;
imshow(uint8(FinalDFT));
title('Case 7: DCT - 2 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDCT(7)=sqrt(immse(FinalDCT,Orignallena));
```

```
FinalDCT = DCT(1);
figure;
imshow(uint8(FinalDFT));
title('Case 8: DCT - 1 largest coefficient');
xlabel('Pixels');ylabel('Pixels');
rmseDCT(8)=sqrt(immse(FinalDCT,Orignallena));

figure;
plot(flip(rmseDFT),'-o');
title('RMSE vs. Number of retained coefficient');
hold on;
plot(flip(rmseDCT),'--+');
hold off;legend('DFT','DCT');
xlabel('Retained size');
ylabel('Root-mean-square-error');

%% %Processing by 8x8 blocks. FFT is computed for each
block
%DCT function
function output = DCT(mag)
Orignallena=double(loadraster('lena512.img',512,512));
close;
[s1 s2]=size(Orignallena);

Normalized_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];

Normalized_Matrix=flip(flip(Normalized_Matrix,2));
% user can change the size as desired
sizeofBlock=8;
shiftLevel=128;

%Processing block by block then find 1st 8 largest
values
temp=double(zeros(size(Orignallena)));
for y=1:sizeofBlock:s1-sizeofBlock+1
    for x=1:sizeofBlock:s2-sizeofBlock+1
```

```
        Imagecropped = Orignallena((y:y+sizeofBlock-
1),(x:x+sizeofBlock-1));
        t=((dct2(Imagecropped-
shiftLevel))./Normalized_Matrix);%dct and level shifted

        original_temp=t;
        CodingMatrix = sort(max(abs(t)),'descend');

     %Cosine coding
        for i=1:mag
            t(abs(t)==CodingMatrix(i))=9999 ;
        end

        CodingMatrix=(t==9999);
        DCT_coding=CodingMatrix.*original_temp;

        temp((y:y+sizeofBlock-1),(x:x+sizeofBlock-
1))=DCT_coding;
    end
end

%Find inverse DCT here
temp1=double(zeros(size(Orignallena)));
for y=1:sizeofBlock:s1-sizeofBlock+1
    for x=1:sizeofBlock:s2-sizeofBlock+1
        Imagecropped = (temp((y:y+sizeofBlock-
1),(x:x+sizeofBlock-1)).*Normalized_Matrix);
        t=(idct2(Imagecropped)+shiftLevel);
        temp1((y:y+sizeofBlock-1),(x:x+sizeofBlock-
1))=t;
    end
end

output=temp1;

end
%Same code as DCT for DFT
function output = DFT(codingsize)
Originallena=double(loadraster('lena512.img',512,512));
close;
[s1 s2]=size(Originallena);

Normalized_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];

Normalized_Matrix=flip(flip(Normalized_Matrix,2));

sizeofblock=8;
shiftLevel=128;

% DFT
temp=double(zeros(size(Originallena)));
for y=1:sizeofblock:s1-sizeofblock+1
    for x=1:sizeofblock:s2-sizeofblock+1
        Imagecropped = Originallena((y:y+sizeofblock-
1),(x:x+sizeofblock-1));
        t=((fft2(Imagecropped-
shiftLevel))./Normalized_Matrix);%dft and level shifted

        original_temp=t;
        codingMatrix = sort(max(abs(t)),'descend');

     %Transform coding and recontructing
        for i=1:codingsize
            t(abs(t)==codingMatrix(i))=9999 ;%can be
any value
        end

        codingMatrix=(t==9999);
        DFTcoding=codingMatrix.*original_temp;

        temp((y:y+sizeofblock-1),(x:x+sizeofblock-
1))=DFTcoding;
    end
end

%  Finding Inverse DFT
temp1=double(zeros(size(Originallena)));
for y=1:sizeofblock:s1-sizeofblock+1
    for x=1:sizeofblock:s2-sizeofblock+1
        Imagecropped = (temp((y:y+sizeofblock-
1),(x:x+sizeofblock-1)).*Normalized_Matrix);
```

```
        t=(abs(ifft2(Imagecropped)+shiftLevel));
        temp1((y:y+sizeofblock-1),(x:x+sizeofblock-
1))=t;
    end
end

output=temp1;

end
```