Blake Johnson

9663980

Aditya 12pm

Homework 7

1. This question involves deconstructing and reconstructing the given images using discrete Fourier transformation and its inverse. This was easily achieved by using the built in matlab functions fft2() and ifft2().

%Load in the matlab images so that they are accessible to the workspace

in01 = load('IMG\_7401.mat');

in05 = load('IMG\_7405.mat');

orig01 = in01.I;

orig05 = in05.I;

%Show the original images

figure

imshow(orig01);

figure

imshow(orig05);

%Take the 2d DFT of the original images

dft01 = fft2(orig01);

dft05 = fft2(orig05);

%Take the real component of the inverse DFT of the DFT of the original

%images.

recon01 = uint8(real(ifft2(dft01)));

recon05 = uint8(real(ifft2(dft05)));

%Show the reconstructed images from the inverse DFT.

figure

imshow(recon01);

figure

imshow(recon05);

Original 7401



Original 7405



Reconstructed 7401



Reconstructed 7405



1. To create a reconstructed image using limited coefficients, I had to find the cutoff value for which coefficients to keep. After this, I took the coefficients that were greater than this value and used them to reconstruct the image using the inverse dft like in number one.

in01 = load('IMG\_7401.mat');

in05 = load('IMG\_7405.mat');

orig01 = in01.I;

orig05 = in05.I;

%Make a call to my ProcessPlusPrint function to deconstruct and reconstruct

%my images and save the outputs for the different coefficient percentages.

[a, b, c, d, e] = ProcessPlusPrint(orig01);

[f, g, h, i, j] = ProcessPlusPrint(orig05);

%Make a call to my local RMSE function and store the values.

RMSEa = RMSE(orig01, a);

RMSEb = RMSE(orig01, b);

RMSEc = RMSE(orig01, c);

RMSEd = RMSE(orig01, d);

RMSEe = RMSE(orig01, e);

RMSEf = RMSE(orig05, f);

RMSEg = RMSE(orig05, g);

RMSEh = RMSE(orig05, h);

RMSEi = RMSE(orig05, i);

RMSEj = RMSE(orig05, j);

%Print out the RMSE values

disp(RMSEa);

disp(RMSEb);

disp(RMSEc);

disp(RMSEd);

disp(RMSEe);

disp(RMSEf);

disp(RMSEg);

disp(RMSEh);

disp(RMSEi);

disp(RMSEj);

%Show each of the images, with all 5 coefficients of the first image first,

%then the 5 coefficients of the second image.

figure

imshow(a);

figure

imshow(b);

figure

imshow(c);

figure

imshow(d);

figure

imshow(e);

figure

imshow(f);

figure

imshow(g);

figure

imshow(h);

figure

imshow(i);

figure

imshow(j);

%This function calculates the RMSE values the same way as the given

%equation.

function X = RMSE(original, reconstruct)

[M, N] = size(original);

sum = 0;

for m=1:M

for n=1:N

sum = sum + (double(original(m,n)) - double(reconstruct(m,n)))^2;

end

end

X = sqrt(sum/(M\*N));

end

function [A, B, C, D, E] = ProcessPlusPrint(orig)

%Initialization section of the function that calculates dft and

%initializes matrices.

dft = fft2(orig);

dftABS = abs(dft);

[M,N] = size(dftABS);

percentValues = zeros(5);

%This block calls the local function that finds the coefficient cutoff

%values.

percentValues(1) = minmax(dftABS, .5);

percentValues(2) = minmax(dftABS, .2);

percentValues(3) = minmax(dftABS, .1);

percentValues(4) = minmax(dftABS, .05);

percentValues(5) = minmax(dftABS, .01);

%Outer for loop makes the process run for the different percentage

%values we need to find

for z=1:5

percentValue = percentValues(z);

newDFT = dftABS;

%The inner for loops check the dft to see if the current element is

%above the cutoff for the given percentage

for m=1:M

for n=1:N

if(dftABS(m,n)<percentValue)

newDFT(m,n) = 0;

else

newDFT(m,n) = dft(m,n);

end

end

end

%Reconstruct the image using the limited dft coefficients.

recon = uint8(real(ifft2(newDFT)));

if(z==1)

A = recon;

elseif(z==2)

B = recon;

elseif(z==3)

C = recon;

elseif(z==4)

D = recon;

else

E = recon;

end

end

end

%This function finds the cutoff element to determine which coefficients we

%should keep.

function J=minmax(orig, percent)

arr = reshape(orig, 1, []);

sortArr = sort(arr);

[~, length] = size(sortArr);

mm = length\*(1-percent);

J=sortArr(uint32(round(mm)));

end

RMSE Values

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DFT | 50% | 20% | 10% | 5% | 1% |
| IMG\_7401 | 1.9656 | 5.0204 | 7.3082 | 9.4335 | 13.5149 |
| IMG\_7405 | 2.3792 | 5.6921 | 8.2171 | 10.7855 | 16.4175 |

Reconstructed 7401 50%



Reconstructed 7401 20%



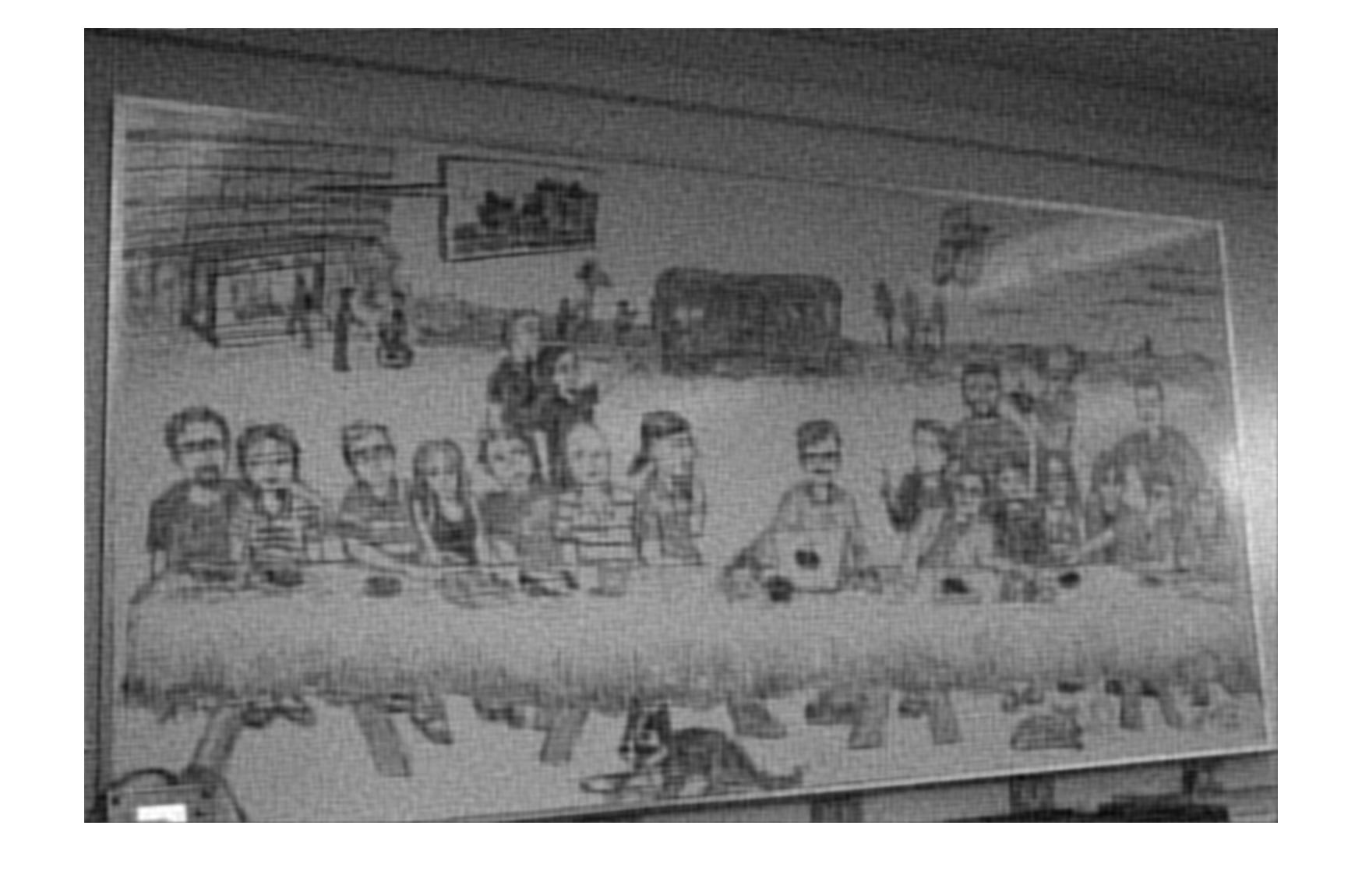
Reconstructed 7401 10%



Reconstructed 7401 5%



Reconstructed 7401 1%



Reconstructed 7405 50%



Reconstructed 7405 20%



Reconstructed 7405 10%



Reconstructed 7405 5%



Reconstructed 7405 1%



1. To create a reconstructed image using limited coefficients, I had to find the cutoff value for which coefficients to keep. After this, I took the coefficients that were greater than this value and used them to reconstruct the image using the inverse dct like in number one.

in01 = load('IMG\_7401.mat');

in05 = load('IMG\_7405.mat');

orig01 = in01.I;

orig05 = in05.I;

%Make a call to my ProcessPlusPrint function to deconstruct and reconstruct

%my images and save the outputs for the different coefficient percentages.

[a, b, c, d, e] = ProcessPlusPrint(orig01);

[f, g, h, i, j] = ProcessPlusPrint(orig05);

%Make a call to my local RMSE function and store the values.

RMSEa = RMSE(orig01, a);

RMSEb = RMSE(orig01, b);

RMSEc = RMSE(orig01, c);

RMSEd = RMSE(orig01, d);

RMSEe = RMSE(orig01, e);

RMSEf = RMSE(orig05, f);

RMSEg = RMSE(orig05, g);

RMSEh = RMSE(orig05, h);

RMSEi = RMSE(orig05, i);

RMSEj = RMSE(orig05, j);

%Print out the RMSE values

disp(RMSEa);

disp(RMSEb);

disp(RMSEc);

disp(RMSEd);

disp(RMSEe);

disp(RMSEf);

disp(RMSEg);

disp(RMSEh);

disp(RMSEi);

disp(RMSEj);

%Show each of the images, with all 5 coefficients of the first image first,

%then the 5 coefficients of the second image.

figure

imshow(a);

figure

imshow(b);

figure

imshow(c);

figure

imshow(d);

figure

imshow(e);

figure

imshow(f);

figure

imshow(g);

figure

imshow(h);

figure

imshow(i);

figure

imshow(j);

%This function calculates the RMSE values the same way as the given

%equation.

function X = RMSE(original, reconstruct)

[M, N] = size(original);

sum = 0;

for m=1:M

for n=1:N

sum = sum + (double(original(m,n)) - double(reconstruct(m,n)))^2;

end

end

X = sqrt(sum/(M\*N));

end

function [A, B, C, D, E] = ProcessPlusPrint(orig)

%Initialization section of the function that calculates dct and

%initializes matrices.

myDCT = dct(double(orig));

dctABS = abs(myDCT);

[M,N] = size(dctABS);

percentValues = zeros(5);

%This block calls the local function that finds the coefficient cutoff

%values.

percentValues(1) = minmax(dctABS, .5);

percentValues(2) = minmax(dctABS, .2);

percentValues(3) = minmax(dctABS, .1);

percentValues(4) = minmax(dctABS, .05);

percentValues(5) = minmax(dctABS, .01);

%Outer for loop makes the process run for the different percentage

%values we need to find

for z=1:5

percentValue = percentValues(z);

newDCT = dctABS;

%The inner for loops check the dft to see if the current element is

%above the cutoff for the given percentage

for m=1:M

for n=1:N

if(dctABS(m,n)<percentValue)

newDCT(m,n) = 0;

else

newDCT(m,n) = myDCT(m,n);

end

end

end

%Reconstruct the image using the limited dct coefficients.

recon = uint8(real(idct(newDCT)));

if(z==1)

A = recon;

elseif(z==2)

B = recon;

elseif(z==3)

C = recon;

elseif(z==4)

D = recon;

else

E = recon;

end

end

end

function J=minmax(orig, percent)

arr = reshape(orig, 1, []);

sortArr = sort(arr);

[~, length] = size(sortArr);

mm = length\*(1-percent);

J=sortArr(uint32(round(mm)));

end

RMSE Values

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DCT | 50% | 20% | 10% | 5% | 1% |
| IMG\_7401 | 1.9214 | 5.7880 | 9.0346 | 12.2163 | 19.0654 |
| IMG\_7405 | 2.1873 | 6.3045 | 9.8902 | 13.7483 | 23.9203 |

Reconstructed 7401 50%



Reconstructed 7401 20%



Reconstructed 7401 10%



Reconstructed 7401 5%



Reconstructed 7401 1%



Reconstructed 7405 50%



Reconstructed 7405 20%



Reconstructed 7405 10%



Reconstructed 7405 5%



Reconstructed 7405 1%

