COMPUTER VISION PROJECTS

Submitted to

Dr. Mahua Bhattacharya

ABV INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT GWALIOR-474010

Submitted by

Group no – ***26***

AMBUJ MISHRA – IPG_2014012

DIVANSHU KHANDELWAL – IPG_2014037

SUNIL KUMAR – IPG_2014117

PROJECT – 1

Objective

- Compress any image of your choice using the Haar Wavelet Transform in the frequency domain.

Introduction

Image Compression

- Image compression is the process of encoding or converting an image file in such a way that it consumes less space than the original file. It is a type of compression technique that reduces the size of an image file without affecting or degrading its quality to a greater extent.
- Image compression is typically performed through an image/data compression algorithm or codec. Typically such codecs/algorithms apply different techniques to reduce the image size, such as by:
  - Specifying all similarly colored pixels by the color name, code and the number of pixels. This way one pixel can correspond to hundreds or thousands of pixels.
  - The image is created and represented using mathematical wavelets.
  - Splitting the image into several parts, each identifiable using a fractal.
- Some of the common image compression techniques are:
  - Fractal
  - Wavelets
  - Chroma sub sampling
  - Transform coding
  - Run-length encoding

Lossy and lossless image compression

- Image compression may be lossy or lossless. Lossless compression is preferred for archival purposes and often for medical imaging, technical drawings, clip art, or comics. Lossy compression methods, especially when used at low bit rates, introduce compression artifacts. Lossy methods are especially suitable for natural images such as photographs in applications where minor (sometimes imperceptible) loss of fidelity is acceptable to achieve a substantial reduction in

bit rate. Lossy compression that produces negligible differences may be called visually lossless.

Haar wavelet transform

- Haar wavelet compression is an efficient way to perform image compression. It relies on averaging and differencing values in an image matrix to produce a matrix which is sparse or nearly sparse. A sparse matrix is a matrix in which a large portion of its entries are 0. A sparse matrix can be stored in an efficient manner, leading to smaller file sizes.
- In this project, we will concentrate on grayscale images; however, rgb images can be handled by compressing each of the color layers separately. The basic method is to start with an image A, which can be regarded as an m × n matrix with values 0 to 255. In MATLAB, this would be a matrix with unsigned 8-bit integer values. We then subdivide this image into 8 × 8 blocks, padding as necessary. It is these 8 × 8 blocks that we work with.

Methodology

- 'haar_wt' function take a grey image and a value 'delta' as inputs and outputs a compressed image. Haar wavelet transformation was used as a transformation matrix for compression process. 'haar_wt_rgb' does the same for an RGB image. 'delta' value governs the compression ratio. 'delta' is a value between 0 and 1, when delta=0, no compression is done. These functions can be considered as a starting point for analysis of Haar wavelet transformation based image compression.
- We have taken a 512x512 pixels sample image named ‘gimage.jpg’ as following which is currently in gray scale –

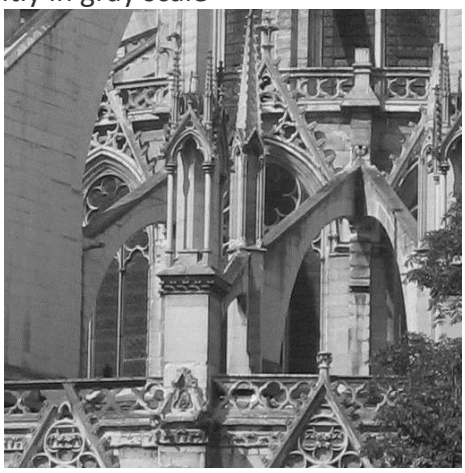


Figure 1 : Original image

Results

- 'haar_wt' function take a grey image and a value 'delta' as inputs. We have entered the ‘gimage.jpg’ image and a compression ratio ‘delta’. Delta value is changed for each run. Initially it was set to 0.01, then 0.05, 0.20 and 0.50. We have observed the final compressed image and the changes made in it.
- Final compressed images with the varying delta values are as following :

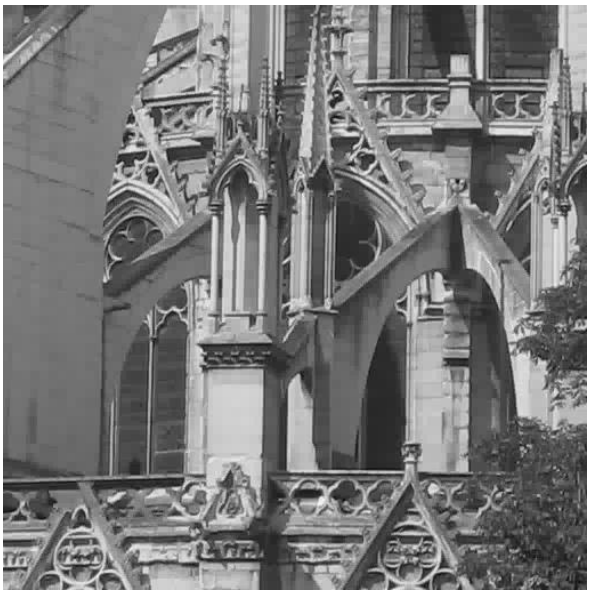


Figure 2 : 0.01 Compression ratio

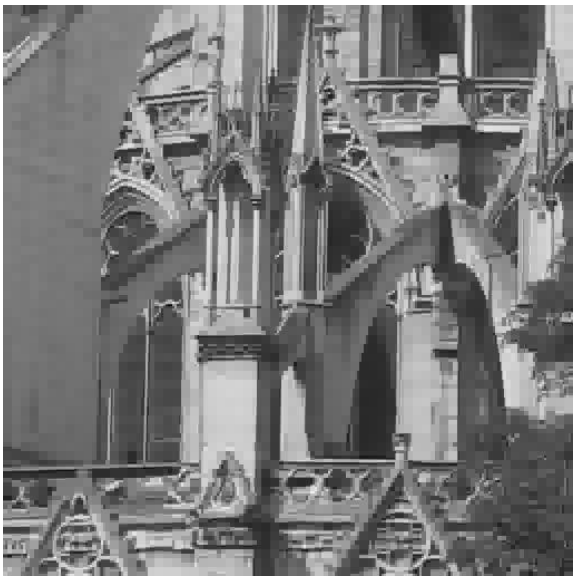


Figure 3 : 0.05 Compression ratio

Figure 4 : 0.20 Compression ratio

Figure 5 : 0.50 Compression ratio

- RGB images can also be handled by compressing each of the color layers separately. The basic method is to start with an image A, which can be regarded as an m × n matrix with values 0 to 255. We have taken a colored image 'abcd.jpg' as example and tried to compress it using haar wavelet transform as well.

Code we have used

```
function haar_wt(image_name,delta)
% 'image_name' is the name of the grey colored image with the file extension
% 'delta' must be a value between 0 and 1
% 'delta' value is a measure of compression ration
% when the 'delta' value is high, compression will be high

% Check number of inputs.
if nargin > 2
   error('harr_wt:TooManyInputs', ...
      'requires at most 1 optional inputs');
end

%Set the default value of 'delta' to 0.01
if (nargin==1)
   delta=0.01;
end

if (delta>1 || delta<0)
   error('harr_wt: Delta must be a value between 0 and 1');
end


%H1, H2, H3 are the transformation matrices for Haar wavelet Transform
H1=[0.5 0 0 0 0.5 0 0 0;0.5 0 0 0 -0.5 0 0 0;0 0.5 0 0 0 0.5 0 0 ;0 0.5 0 0 0 -0.5 0 0 ;0 0 0.5 0 0 0 0.5
0;0 0 0.5 0 0 0 -0.5 0;0 0 0 0.5 0 0 0 0.5;0 0 0 0.5 0 0 0 -0.5;];
H2=[0.5 0 0.5 0 0 0 0 0;0.5 0 -0.5 0 0 0 0 0;0 0.5 0 0.5 0 0 0 0;0 0.5 0 -0.5 0 0 0 0;0 0 0 0 1 0 0 0;0 0
0 0 0 1 0 0;0 0 0 0 0 0 1 0;0 0 0 0 0 0 0 1;];
H3=[0.5 0.5 0 0 0 0 0 0;0.5 -0.5 0 0 0 0 0 0;0 0 1 0 0 0 0 0;0 0 0 1 0 0 0 0;0 0 0 0 1 0 0 0;0 0 0 0 0 1 0
0;0 0 0 0 0 0 1 0;0 0 0 0 0 0 0 1;];

%Normalize each column of H1,H2,H3 to a length 1(This results in orthonormal columns of each
matrix)
H1=normc(H1);
H2=normc(H2);
H3=normc(H3);
```

```
H=H1*H2*H3; %Resultant transformation matrix
x=double(imread(image_name));
len=length(size(x));

if len~=2
   error('harr_wt: Input image must be a grey image, use "haar_wt_rgb" function to compress
RGB Images');
end

y=zeros(size(x));
[r,c]=size(x);
%Above 8x8 transformation matrix(H) is multiplied by each 8x8 block in the image

for i=0:8:r-8
   for j=0:8:c-8
      p=i+1;
      q=j+1;
      y(p:p+7,q:q+7)=(H')*x(p:p+7,q:q+7)*H;
   end
end


figure;
imshow(x/255);

%compression ratio depends on the delta value you select
%Larger the value 'delta', compression ratio will be larger
%delta=0.01;

n1=nnz(y);                    % Number of non-zero elements in 'y'

z=y;
m=max(max(y));
y=y/m;
y(abs(y)<delta)=0;              % Values within +delta and -delta in 'y' are replaced by zeros(This is the
command that result in compression)
y=y*m;
n2=nnz(y);                    % Number of non-zero elements in updated 'y'

%Inverse DWT of the image

for i=0:8:r-8
   for j=0:8:c-8
```

```
      p=i+1;
      q=j+1;
      z(p:p+7,q:q+7)=H*y(p:p+7,q:q+7)*H';
   end
end

figure;
imshow(z/255);                    % Show the compressed image
imwrite(x/255,'orginal.tif');        %Check the size difference of the two images to see the
compression
imwrite(z/255,'compressed.tif');

% Below value is a measure of compression ratio, not the exact ratio
%compression_ratio=n1/n2
```

Project - 2

Objective

- Convert an image into spatial frequencies using Fast Fourier Transform(FFT)
- Convert it back to the spatial domain using inverse Fast Fourier Transform (IFFT).

Introduction

Spatial frequency:

- In mathematics, physics, and engineering, spatial frequency is a characteristic of any structure that is periodic across position in space. The spatial frequency is a measure of how often sinusoidal components (as determined by the Fourier transform) of the structure repeat per unit of distance. The SI unit of spatial frequency is cycles per meter. In image-processing applications, spatial frequency is often expressed in units of cycles per millimeter or equivalently line pairs per millimeter.

Fast Fourier Transform

- A fast Fourier transform (FFT) is an algorithm that samples a signal over a period of time (or space) and divides it into its frequency components. These components are single sinusoidal oscillations at distinct frequencies each with their own amplitude and phase.
- An FFT algorithm computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IFFT). Fourier analysis converts a signal from its original domain to a representation in the frequency domain and vice versa. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. As a result, it manages to reduce the complexity of computing the DFT from $O(n)$^2 which arises if one simply applies the definition of DFT, to O(n logn), where n is data size.

Methodology

- As we are only concerned with digital images, we will restrict this discussion to the *Discrete Fourier Transform* (DFT).
- The DFT is the sampled Fourier Transform and therefore does not contain all frequencies forming an image, but only a set of samples which is large enough to fully describe the spatial domain image. The number of frequencies corresponds to the

number of pixels in the spatial domain image, *i.e.* the image in the spatial and Fourier domain are of the same size.

- For a square image of size N×N, the two-dimensional DFT is given by:

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j)\, e^{-\imath 2\pi (\frac{ki}{N} + \frac{lj}{N})}$$

- where *f(a,b)* is the image in the spatial domain and the exponential term is the basis function corresponding to each point *F(k,l)* in the Fourier space. The equation can be interpreted as: the value of each point *F(k,l)* is obtained by multiplying the spatial image with the corresponding base function and summing the result.
- The basic functions are sine and cosine waves with increasing frequencies, *i.e. F(0,0)* represents the DC-component of the image which corresponds to the average brightness and *F(N-1,N-1)* represents the highest frequency.
- In a similar way, the Fourier image can be re-transformed to the spatial domain. The inverse Fourier transform is given by:

$$f(a,b) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} F(k,l)\, e^{\imath 2\pi (\frac{ka}{N} + \frac{lb}{N})}$$

- Note the $\frac{1}{N^2}$ normalization term in the inverse transformation. This normalization is sometimes applied to the forward transform instead of the inverse transform, but it should not be used for both.$$
- To obtain the result for the above equations, a double sum has to be calculated for each image point. However, because the Fourier Transform is *separable*, it can be written as:

$$F(k,l) = \frac{1}{N} \sum_{b=0}^{N-1} P(k,b)\, e^{-\imath 2\pi \frac{lb}{N}}$$

- Where

$$P(k,b) = \frac{1}{N} \sum_{a=0}^{N-1} f(a,b)\, e^{-\imath 2\pi \frac{ka}{N}}$$

- Using these two formulas, the spatial domain image is first transformed into an intermediate image using N one-dimensional Fourier Transforms. This intermediate image is then transformed into the final image, again using N one-dimensional Fourier Transforms. Expressing the two-dimensional Fourier Transform in terms of a series of 2N one-dimensional transforms decreases the number of required computations.

- Even with these computational savings, the ordinary one-dimensional DFT has N^2 complexity. This can be reduced to $N\ log_2 N$ if we employ the Fast Fourier Transform (FFT) to compute the one-dimensional DFTs. This is a significant improvement, in particular for large images. There are various forms of the FFT and most of them restrict the size of the input image that may be transformed, often to $N = 2^n$ where n is an integer. The mathematical details are well described in the literature.

- The Fourier Transform produces a complex number valued output image which can be displayed with two images, either with the real and imaginary part or with magnitude and phase. In image processing, often only the magnitude of the Fourier Transform is displayed, as it contains most of the information of the geometric structure of the spatial domain image. However, if we want to re-transform the Fourier image into the correct spatial domain after some processing in the frequency domain, we must make sure to preserve both magnitude and phase of the Fourier image.

- The Fourier domain image has a much greater range than the image in the spatial domain. Hence, to be sufficiently accurate, its values are usually calculated and stored in float values.

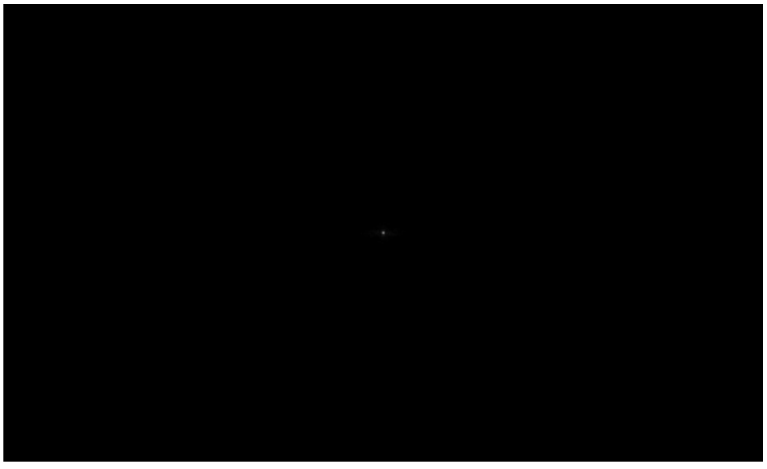
Implementations and Results

- Add an image on which we wants to implement Fast Fourier transform(FFT).

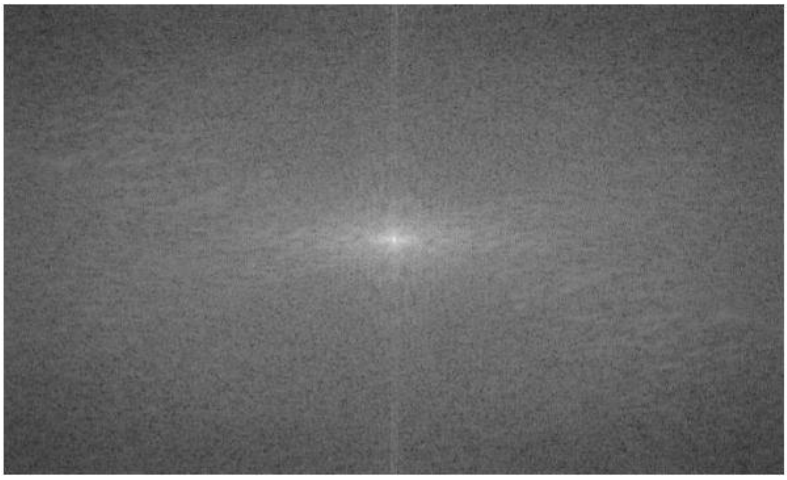
- We convert original image to gray image so that we can apply FFT.

- Then we apply FFT on the gray image using Matlab.

- Then we take Centered Fourier transform of Image.



- After that we take log transformed Image.



- At last we use IFFT to reverse the effects of the previous and we got gray image as a result.

Code we have used

```
clc
clear all; close all;
imdata = imread('abcd.jpg');
figure(1);
imshow(imdata);
title('Original Image');
%imdata2 = rgb2gray(imdata);
imdata = imread('abcd.jpg');
figure(2);
imshow(imdata);
title('Gray Image');
F = fft2(imdata);
S = abs(F);
figure(3);
imshow(S,[]);
title('Fourier transform of an image');
Fsh = fftshift(F);
figure(4);imshow(abs(Fsh),[]);
title('Centered fourier transform of Image')
S2 = log(1+abs(Fsh));
figure(5);
imshow(S2,[]);
```

- title('log transformed Image')
- F = ifftshift(Fsh);
- f = ifft2(F);
- figure(6);
- imshow(f,[]),title('reconstructed Image')