Image Compression Using Fourier Series

Yash Gupta
0869229
Software Engineering
ygupta1@lakeheadu.ca

Marko Javorac
1107295
Software Engineering
mjavorac@lakeheadu.ca

Douglas Jones
0695051
Electrical Engineering
drjones1@lakeheadu.ca

James Rocco
1094783
Electrical Engineering
jarocco@lakeheadu.ca

***Abstract*—In this project we are to conceptualize the design of a JPEG image compression software which will provide three modes of compression for any allotted image; low, medium and high. This will be carried out through a numerical computing program known as Matlab, where the algorithm designed will be implemented and tested. The expectations would include; the implementation of a fast fourier transform function,discrete cosine transform function, as well as the creation of a graphical user interface. This project will help the involved participants have a better understanding of the mathematical functions, fast fourier transform and discrete cosine transform, along with being a more proficient programmer in Matlab.**

I. Introduction

The most common type of image compression algorithm used today is called JPEG. JPEG stands for 'Joint Photographic Experts Group' and is a type of lossy image compression that allows you to compress raw images from cameras and store them as smaller files. Compression is important for allowing easier transport of images via the internet and creates smaller image sizes for help with storage. This type of compression is referred to as a lossy because the compression algorithm will give away unrecoverable image quality in order to compress the image. With this, the more compression selected for the image the higher rate of loss, resulting in an overall lower quality of the image.

In this project, three forms of image compression will be designed, implemented and tested using fourier techniques. The use of fourier transforms is a mathematical operation that takes the time domain of a function and transforms it into the frequency domain. These techniques are mostly used in electronics communication but have been a useful advantage and can be applied here for image compression. The advantage is that by being in the frequency domain, the image can be broken up into cosine and sine components allowing for easy removal of components, in turn compressing the image. Along with image compression, a graphical user interface (GUI) will be designed and implemented for easy use of the algorithms.

The three algorithms being discussed in this project are known as Discrete Fourier Transform, DFT, Fast Fourier Transform, FFT, and Discrete Cosine Transform, DCT. Discrete fouriers represent functions that are in vector form, where matrix calculations can be used. The discrete cosine transform is the algorithm used today with JPEG compression and along with FFT it is derived from the main algorithm of discrete fourier transform.

II. Fourier Series & Transforms

Fouriers analysis consists of two forms of operation, Fourier Series and Fourier Transform. Fouriers series consists of taking a periodic function and representing it as a trigonometric series in terms of sine and cosine. Whereas fouriers transform looks at non-periodic functions. It consists of taking these non-periodic functions in their time domain and transforming them into the frequency domain. This process can be inverted which is known as taking the inverse fourier transform, taking the frequency domain and representing it in the time. In this report, we will be analyzing the effects of fouriers transform, moreover, than the series and exploring the benefits of being able to manipulate the frequency domain for lossy image compression.

A. Discrete Cosine Transform

Discrete Cosine Transform (DCT) is used in standard jpeg compression. DCT is derived from Discrete Fourier Transform, but has a special property that allows it to be very efficient in comparison with other methods. DCT, when used with matrices, can be done in a singular matrix multiplication. This makes it both very easy to implement and efficient for a computer to run. One of the key advantages of DCT over other methods for compression is the fact that it can be implemented with an 8X8 matrix multiplication. This results in highly efficient compression in comparison to Discrete Fourier Transform, and is much easier to implement than the algorithm for Fast Fourier Transform.

The general expression for this matrix transform is: D = T*M*T', where T is the transformation matrix, and M is an 8X8 block of the image. The variable T' denotes the transformation matrix except it is transposed.

$$D_{i,j} = \sqrt{\frac{2}{8}} C(i)c(j) \sum_{x=0}^{7} \sum_{y=0}^{7} p(x,y) cos(\frac{(2x+1)i\pi}{16}) cos(\frac{(2y+1)i\pi}{16}) \quad (1)$$

$$C(i) = 1/\sqrt{2}, i = 0 \quad (2)$$

and C(i) = 1 for all other i values. Two dimensional transformation matrix T:

$$T_{i,j} = \sqrt{\frac{2}{N}}, i = 0, \quad \text{and} \quad T_{i,j} = \frac{1}{2} cos(\frac{(2y+1)i\pi}{16}) \quad \text{for other i values} \quad (3)$$

Matrix form of DCT:

$$D = T \cdot M \cdot T' \quad (4)$$

B. Discrete Fourier Transform

As mentioned before discrete functions are represented in terms of vectors, to compute vector to vector calculations, matrices must be used to determine the outcomes. In turn, the Discrete Fourier Transform, DFT, is an operation that consists of inputting N numbers and receiving an N amount of outputs, where 'N' represents the number of points. With image compression the number of points can be incredibly large which affects how long the computations of the calculation will take. Due to this disadvantage of the DFT, other algorithms were derived to increase computation time in the calculations of these transforms.

The importance of the discrete fouriers transform stems from the ability to represent the function in terms of frequency which allows the operation to reveal periodicities in the function. This allows for the repetitive nature to be discarded in image compression in order to shrink the image. This idea is implemented in the derived algorithms as well.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi \cdot kn/N} \tag{5}$$

C. Fast Fourier Transform

A fast Fourier transform is or a group of algorithms used to compute a DFT or IDFT of a sequence of numbers. This sequence is usually a signal. As we have seen that a Fourier Analysis converts a signal from its original domain i.e. from a domain of time or space to the frequency domain. The DFT is obtained by decomposing a sequence of values into components of different frequencies. But computing DFT using the traditional definition can take a long time. If N is the number of elements in the sequence, the number of operations performed through DFT will be N^2. As N increases, the time for computation increases respectively. This is where FFT comes in. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. FFT algorithms exploit the symmetries of the exponential term in DFT because it is periodic. FFT reduced the N^2 operations to N log(N) operations, which can make a huge difference in terms of computation speed as our N increases. There are many different FFT algorithms based on a wide range of published theories, from simple complex-number arithmetic to group theory and number theory.

The most simple algorithm used to implement FFT is the Cooley-Tukey algorithm. The algorithm re-expresses the DFT of N sequences as a product of N1 & N2, where N1 is the number of small DFTs of sequences N2. The Cooley-Tukey algorithm can be combined with other DFT algorithms, for example, Rader's, Bluestein's or prime-factor algorithm to deal with prime numbers.

The most common and simplest form of the Cooley-Tukey algorithm is radix-2 Decimation in Time. The DIT implementation requires N which can be expressed as a power of 2. In this algorithm, we split N into two halves in terms of even and odd positions in the sequence. The splitting process continues until we get a single term DFT for computation. For example for a sequence of N=8, the splitting process takes place 3 times. So we start with a single DFT of 8 terms and split it to 8 DFT of 1 term each. We can further optimize the function using the fact that the exponential term is periodic and hence reduces the computation of exponential terms by half. DIT first computes the even indexed elements. This algorithm is a good example of divide and conquer algorithms. When expressed on paper as a data flow diagram, the algorithm shows a butterfly effect.

III. JPEG Compression Algorithm

A. Overview

The general method for jpeg compression using DCT is as follows:

First, the DCT matrix (see formula 2 in the DCT equations section) must be made. In the case of JPEG compression, this matrix must be 8X8 in size because the JPEG quantization matrix, which is a standard that has been experimentally fine tuned, is also 8X8. This means that the transform can be done on any image as long as it is segmented into 8X8 pieces and then reattached later.

Then, after the transform has been done on the entire image, it is divided by the quantization matrix. This resultant is then rounded to the nearest integer values, and from this becomes a matrix with zeros in the matrix. This rounding to attain zeroes in the matrix is a key point in the compression process.

Finally, the image is multiplied by the quantization matrix, and has the inverse DCT implemented on it. The result is a compressed image. The reason that it becomes compressed is the fact that the zeroes in the matrix hold no data, in other words the image has less data even after the remaining elements have been reverted to their original values.

B. Algorithm Implementation

The algorithm that was fully implemented in this project is outlined here:

- Convert the image from RGB to YCbCr format
- Trim the image so that it is a multiple of 8, for the 8X8 matrices to fit evenly inside the image
- Initialize the DCT Matrix, so that the transform can be done on the blocks of the image
- Initialize the quantization Matrix, this is necessary for the compression of the image after DCT has been done on it.
- Because the image is a three dimensional matrix, here is where a for-loop should be started, with a variable ranging from one to three, for each plane in the matrix. All steps after this are within this loop until otherwise said.
- Convert the image to a double, so that when it is divided by the quantization matrix you get decimals.
- Subtract 128 from each element in the matrix, so that it ranges from -128-127. This is necessary because the
- DCT is centred about the origin.

- Apply DCT to the 8X8 blocks of the plane that is currently being considered.
- Divide these transformed 8X8 blocks by the quantization matrix, element by element, with the compression levels set for that plane (Y, Cb, or Cr).
- Round the resultant matrix to the nearest integer values. Decimals below 0.5 will be rounded to zero, thus shrinking the amount of memory the image takes up.
- Now that the decimals have been rounded, multiply this matrix by the quantization matrix (element by element) to begin undoing the previous steps. The elements that have been rounded to zero will not be recovered.
- Apply inverse DCT on the 8X8 blocks of the plane, this takes the image out of the frequency domain and reverts back into the visual domain.
- Paste the 8X8 blocks of the current plane back together.
- Add 128 to the matrix to bring the values back to their original range.
- Initialize a three dimensional matrix to hold the compressed image, and assign each plane to its respective location in this matrix.
- End the outermost for-loop, all the work on the separate plains has been done.
- Convert the image from YCbCr back to RGB, so that it has its original colours.

IV. Compression using Fast Fourier Transform

A. Overview

The FFT compression implemented works on the basics of compression. The algorithm is converting an image matrix of 3 dimensions (RGB) into the frequency domain using FFT. To remove low signals, a threshold is used in the algorithm which is provided by the user and based on that, we are filtering Signals above the threshold, making it a high pass filter. This is done due to the fact the human vision is only able to perceive frequencies above a certain threshold, using this theory low frequencies can be removed from the image while keeping most of the details intact. After filtering out the low frequencies, the image is transformed back into its original domain using IFFT.

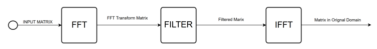


Fig. 1. Flow Diagram explaining the flow for FFT compression

B. Algorithm Implementation

The program uses a self implemented FFT function which is based on Radix-2 Decimation In Time or DIT derived from Cooley-Tukey algorithm. As we stated in the previous sections that the DIT only works best with signals or sequences which can be expressed in terms of 2^n. The program uses a matrix implementation of the algorithm according to which the exponential matrix in of DFT can be expressed as a product of three matrices in which two matrices are fix-up matrices. A general equation to calculate the DFT of a sequence N is:

$$X = F_N \cdot x \tag{6}$$

where X is the transform, F_N is the exponential matrix for N terms and x is the sequence or the signal. According to the algorithm the F_N can be expressed as :

$$F_N = \begin{bmatrix} I & D \\ I & -D \end{bmatrix} \cdot \begin{bmatrix} F_{N/2} & 0 \\ 0 & F_{N/2} \end{bmatrix} \cdot P \tag{7}$$

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & W & & 0 & 0 \\ 0 & 0 & W^2 & 0 & 0 \\ 0 & 0 & 0 & ... & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & W^(N/2-1)) \end{bmatrix} \tag{8}$$

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{9}$$

Permutation matrix of even and odd
As we can see from the equations that the N term can be split n times i.e. N=1. This recursion reduces the calculations from N^2 to N $\log_2$N. The algorithm takes a complete image matrix i.e. all three dimensions and computes the exponential matrix once for both column and rows. This is useful as it saves the time for computation. The resultant exponential matrices can be used for transforming the matrix to the frequency domain. Due to the fact that the algorithm can only work on sequences which can be represented in term of 2^n, we are initially padding the image with zeros so that the image is suitable for the DIT algorithm. After the image is compressed it is brought back to its original size. The overhead observed in this implementation is that it increases the matrix size which can further make the operations more expensive and is a trade-off with time.

V. Graphical User Interface

A. Overview

Our Graphical User Interface(GUI) was designed and developed in MATLAB using the GUIDE utility. The application allows the user to select an image from their local device using the “import image” button. The user then specifies the type of compression desired along with the amount of compression. The FFT method offers 3 unique levels of compression including High, Medium, and Low. The DCT method allows users to fine tune their compression using Luma(Y) and Chroma Red(Cr) and Chroma Blue(Cb) scales. The application requires the user to name their compressed file which is entered in the text box. The application also includes 3 panes that provide the user with a preview of the image in original, compressed, and difference.

B. Layout

The application is divided into 3 major sections, File I/O, Compression Settings, and Image Preview Panes.

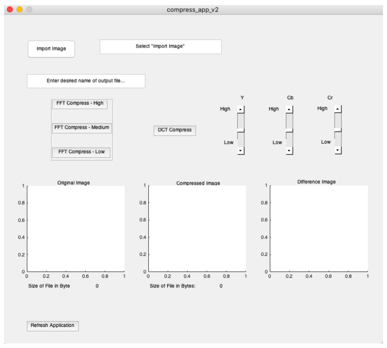


Fig. 2. Graphical Interface of the product

1) File I/O: The first component of the application handles specifying the desired image to compress. When a user selects

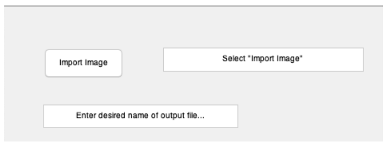
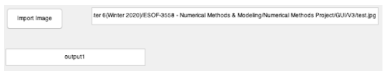


Fig. 3. File I/O layout

"import image", a file explorer will appear and will allow them to navigate to the image. Once a user selects an image, the text will be updated with the path to the folder so the user can verify their selection. The user will then enter the name of the output file in the text field.



ter 6(Winter 2020)/ESOF-3558 - Numerical Methods & Modeling/Numerical Methods Project/GUI/V3/test.jpg

output1

Fig. 4. File I/O in work

2) Compression Settings: The compression settings are composed of two distinct subsections. One dedicated to FFT and the other to DCT.

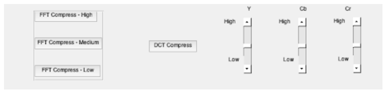


Fig. 5. Compression Settings

The DCT sections contain 3 sliders that correspond to the Y, Cb, and Cr values of the image. The user enters the desired configuration and then selects the "DCT Compress" button which executes the compression algorithm. The values in the slider range from 1 to 50, resulting in low to high compression factors. As a default, all the slider values are set at 25 which is a medium value in terms of compression.
The FFT section of the settings contains 3 options, High Compression, Medium Compression, and Low Compression. Within the FFT method, selecting one of these options transfers a specific quality factor that alters the resulting compression. From High to Low, these values translate to a compression factor of 0.001, 0.0005, and 0.000001 respectively.

C. Image Preview Panes

The image preview pans contain 3 distinct windows that are populated throughout the compression process. The first

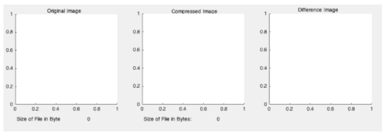
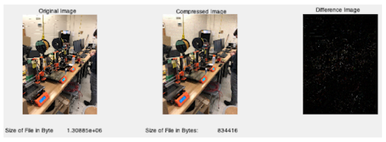


Fig. 6. Image View Panes for Orignal, Compressed & Difference Image

pane, starting from the left side, contains a preview of the image. This is populated as soon as a user selects the image but prior to altering the settings. The second pane contains the resulting image after either compression method is applied. This preview is only populated after the compression is completed. The final window contains an image that displays the difference from the original to the final compressed image. Below the original and compressed image, there is a data display that contains the size of the files in bytes for quick comparison.



Size of File in Byte 1.30885e+06
Size of File in Bytes: 834416

Fig. 7. Image View Pane displaying the original & compressed image along with their difference

VI. Results & Analysis

TABLE I
Results Obtained from Execution

Compression Method	***File Size (bytes)***	***Execution Time (sec)***
Orignal Image	67084	N/A
FFT LOW	48683	2.053812
FFT MEDIUM	42439	1.907386
FFT HIGH	36661	1.837890
JPEG LOW	47672	0.525469
JPEG MEDIUM	47655	0.323203
JPEG HIGH	8689	0.315261

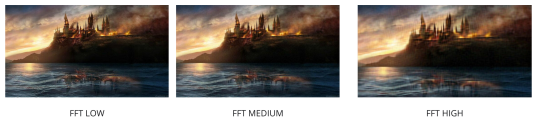


Fig. 8. Resulting Images from FFT Compression

As we can see from the results obtained, compression is a trade-off between quality v.s. size. We can see in JPEG compression that when we a high value was used, the quality went down but the image was compressed by a factor of 8. In FFT compression we can see that as we go up the image starts to become more blurry and starts to loose details but shows

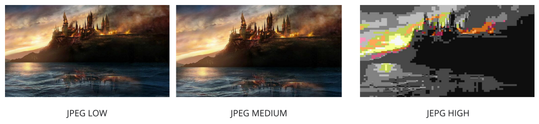


Fig. 9. Resulting Images from JPEG Compression

more compression. Both the algorithms used in the report are not comparable as they use two different ways to compress a file. Both the algorithms tells us that we can remove certain amount of data from the image and still will not be able to affect the quality we perceive as humans since human vision cannot en-capture.

VII. Future Work

Through this project, we got the opportunity to explore the field of image compression and different mathematical formulas which holds a great amount of importance in the modern world of computing. During our research on these algorithms, the major issue that we faced was that of optimization. The implemented algorithms were optimized to the best of our capabilities but there is still a lot of room for further progress. In the future, more complex FFT algorithms can be used to deal with varied sizes of matrices, which are faster. Moreover, due to the fact that we are dealing with matrices, parallel computing can be used to fasten up the process of matrix multiplication. In our implementation, in order to save time, we are padding our matrix with zeros, in the future interpolation can be used to increase the size of the matrix without affecting the quality. Since this project introduced us with image compression, compressing videos or audio files can be the next challenge. One interesting project that can be looked into is to analyse properties of certain things from a video, for example calculating the velocity of a ball thrown by a robot in a video or the number of chemicals in a paper just by analysing the color. Compression has endless applications in today's world, so a lot can be achieved.

VIII. Appendix

A. Code snippet displaying JPEG compression

```
%compression one plane at a time
for plane  = 1:3
    CompMat = ModImg(:,:,plane);
    CompMat = double(CompMat) - 128;
    QQ = Q*Quality(plane);
    CompImgMat(:,:,plane) =
    CompressEachPlane(CompMat,T,QQ);
end
%decompression
for plane  = 1:3
    DCompMat = CompImgMat(:,:,plane);
    QQ = Q*Quality(plane);

    % Performing Inverse DCT
    DCompMat  = DCompressEachPlane(DCompMat,T,QQ);
    DCompMat = DCompMat + 128;
    DCompImgMat(:,:,plane) = DCompMat;
end

%Compress Each plane
for i = 1:repy
% variable to hold the y coordinate
    y =1;

    for j = 1:repx
     %Extracting 8x8 matrix for operations
        B = Mat8_operation(A,x,y);
        D = T*B*T';
        DQ = round(D./Q);

        %putting B in a big matrix
        Final = PushMatOperation(Final,DQ,x,y);
        y = y+8;
    end
    x = x+8;
end

%DCompress Each plane
for i = 1:repy
    y =1; % variable to hold the y coordinate
    for j = 1:repx
%Extracting 8x8 matrix for operations
        B = Mat8_operation(A,x,y);
        DQ = B.*Q;
        D = (T')*DQ*T;

        %putting D in a big matrix
        Final = PushMatOperation(Final,D,x,y);
        y = y+8;
    end
    x = x+8;
end
```

B. Code Snippet to Calculate the exponential matrix using FFT

```
if(n>0)

%generating a permutation matrix for splitting
the sequence based on even and odd positions
%this is done due to the fact that the
 exponential term

P = gen_perm_mat(n);

%exponential term
w = exp(-2*i*pi/2^n);

%This statement generates a fixup matrix which
is comprised of 2 identity matrices and 2
diagnal Exponential matrices of order (2^n)/2;
ID = gen_ID_mat(n,w);

%recursion for further splitting Matrix
F = Gen_FFT_Mat(n-1);

%this section of code is used to
generate the middle matrix
N = (2^n)/2;
BigN = 2^n;
F_final = zeros(2^n,2^n);
F_final(1:N,1:N) = F;
F_final(N+1:BigN,N+1:BigN) = F;
Final = ID*F_final*P;
else %WHEN n=0 i.e. only one term left
    Final = 1;
end
```

References

[1] Image Compression Using Fourier Techniques - Liam Flynn http://www.maths.usyd.edu.au/u/olver/teaching/Computation/ExampleProject.pdf Understanding JPEG Compression:

[2] https://www.image-engineering.de/library/technotes/745-how-does-the-jpeg-compression-work

[3] https://hackernoon.com/why-do-we-need-jpeg-compression-and-how-its-technically-working-52a3a9ced55d

[4] https://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm

[5] https://towardsdatascience.com/fast-fourier-transform-937926e591cb

[6] https://see.stanford.edu/materials/lsoftaee261/book-fall-07.pdf

[7] https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossy/jpeg/index.htm

[8] http://www.robots.ox.ac.uk/ sjrob/Teaching/SP/l7.pdf