IB Mathematics Extended Essay:

The application of the Discrete Cosine Transform in Image Processing

Research Question:

To what extent is the Discrete Cosine Transform involved and effective in JPEG image compression?

Word Count: 3943 Words

kjj949

May 2025

Contents

1 Introduction **3**

2 Images as Matrices **4**

2.1 Representing Images Mathematically 4

2.2 Introduction to Greyscale Images 5

3 Methodology of the JPEG Algorithm **6**

4 Introduction to Mathematical Transforms **9**

4.1 The Purpose of Mathematical Transforms 9

4.2 The Fourier Transforms 9

5 The Discrete Cosine Transform **11**

5.1 Deriving the DCT 11

5.2 Application of the DCT in Image Compression 12

5.3 Sample DCT Calculation 14

5.4 Quantization and The Inverse DCT (IDCT) 18

5.5 Comparing the DCT and FTs 21

5.6 Analysis of findings 22

6 Completing the Compression Process **24**

7 Extensions and Limitations **24**

8 Conclusion **25**

1 Introduction

The digital world is filled with many types of media; photographs are one of the most commonly used visual assets. The repository of images is ever so increasing with the major presence of modern-day Internet usage. However, raw photos and videos consume a sizable portion of computer storage due to how much information they contain. Storing and sharing these files require large amounts of bandwidth, which can be costly and impractical. In 1992, the Joint Photographic Experts Group (JPEG) format was adopted to address this challenge by using image compression techniques (The Craft, 2022). It has become the standard for compression because of its ability to significantly reduce the file size of images while maintaining high quality. A key technique for this process is the Discrete Cosine Transform (DCT). The DCT separates images into parts of different frequencies that are then evaluated to create the compressed image. The focus of this essay is to look at how the image compression process is undergone and the importance of the DCT within it. In this essay, the JPEG algorithm will be investigated and, in preface to the concept of the DCT, the Fourier transforms will be explored. The DCT will also be derived from the Discrete Fourier Transform and an example of the DCT will be provided. Finally, the DCT will be compared with the Fourier Transform to evaluate whether it is the most efficient transform. The following research question was created to spearhead this investigation:

To what extent is the Discrete Cosine Transform involved and effective in JPEG image compression?

2 Images as Matrices

2.1 Representing Images Mathematically

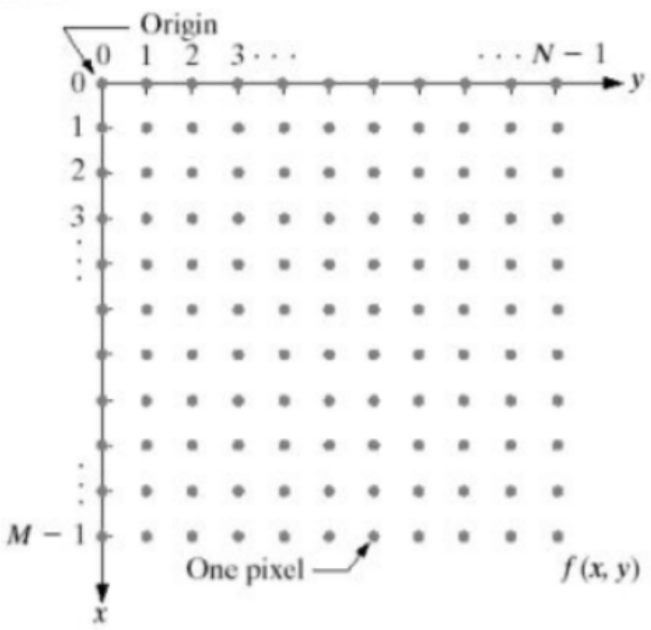


Figure 1: Convention for Digital Image representation (Woods & Gonzales, 2002)

The first step in understanding JPEG image processing is looking at how images are represented. Images are represented using matrices. Assume a digital image $f(x, y)$ is created such that the resultant image has M matrix rows and N columns. The value of the coordinates (x, y) become these discrete integers. At the origin, the values of the coordinates (x, y) become $(0, 0)$. In matrix form, an image can be represented as:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

2.2 Introduction to Greyscale Images

Greyscale images are usually referred to as black-and-white images, although its name suggests that these sorts of images will include a lot of grey. Greyscale images are represented by two-dimensional matrices. In the real world, the coordinates $f(x, y)$ represent what is called a pixel. Each pixel in the matrix is represented by p bits, which is the smallest unit of information in a computer. A common bit-to-pixel ratio for JPEG compression is $8 : 1$. The value of each pixel represents the intensity of the light, in other words, luminance. For a digital image with p bits, the range of intensity would be between $[0, 2^p - 1]$ (Express, 2019). Hence, for an 8-bit image, the range for each pixel would be:

$$[0, 2^8 - 1]$$

or

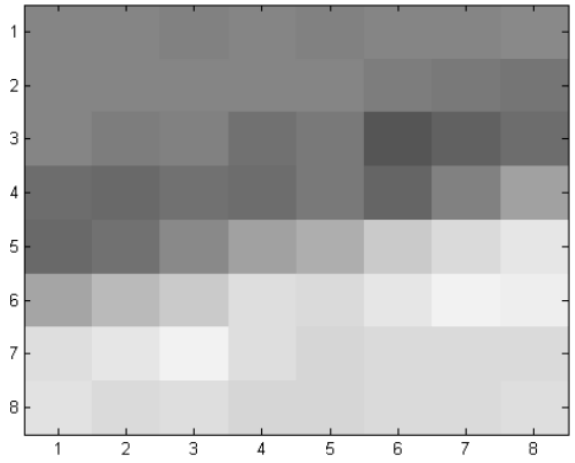
$$[0, 255]$$

From this it is declared that $f(x, y)$ would give the intensity of the image at positions (x, y) defined over a finite image in the xy-plane. Given that the image is defined over the rectangular intervals $x \in [a, b]$ and $y \in [c, d]$, the function $f(x, y)$ is defined as follows:

$$f : [a, b] \times [c, d] \to [0, 255]$$

What this reads is that, given a digital image, the output of a certain input $f(x, y)$ would be integers ranging from $0 - 255$, and these integers represent the intensity of light. The value 0 corresponds to the colour black and the value 255 corresponds to the colour white.

Given a sample 8 pixel by 8 pixel greyscale image with differing intensities (Kim, n.d.):



The matrix that represents this image would be:

$$f(x,y) = \begin{bmatrix} 135 & 135 & 129 & 133 & 130 & 134 & 134 & 137 \\ 133 & 133 & 132 & 132 & 135 & 127 & 123 & 119 \\ 132 & 127 & 129 & 115 & 121 & 87 & 96 & 110 \\ 110 & 104 & 115 & 109 & 120 & 103 & 129 & 160 \\ 105 & 112 & 136 & 162 & 173 & 201 & 219 & 231 \\ 167 & 187 & 202 & 223 & 216 & 231 & 240 & 238 \\ 221 & 231 & 240 & 223 & 214 & 216 & 218 & 219 \\ 224 & 217 & 222 & 214 & 215 & 217 & 219 & 220 \end{bmatrix}$$

3 Methodology of the JPEG Algorithm

After understanding how images are represented mathematically, the JPEG compression algorithm can be explored. The two major forms of compression are lossy and lossless compression, involving the loss and preservation of data respectively. JPEG compression is a form of lossy compression. The JPEG lossy compression algorithm consists of many steps, each having their own unique purpose and serves importance in many ways.

1. Colour Transform

If the image is coloured, it will be converted to a model called the YCbCr model before undergoing compression. This will be further explained in the Extension chapter of the essay.

2. Down-sampling

After the colour transform, the image undergoes down-sampling. Down-sampling is used to reduce the length and width, in pixels, of the image. It is a crucial step for reducing the dimensionality of the data, leading to a faster compression. The given name for the image before it is compressed is the source image, and the same applies for pixels. Down-sampling is not to be confused with image compression, however. There are many algorithms used to undergo down-sampling, one of them being the box filter algorithm—a linear algorithm and one of the quickest algorithms used to down-sample an image. The box filter algorithm uses summed areas to achieve its goal—the new value of a compressed pixel is the average of the values of source pixels in a square centered around the targeted compressed pixel.

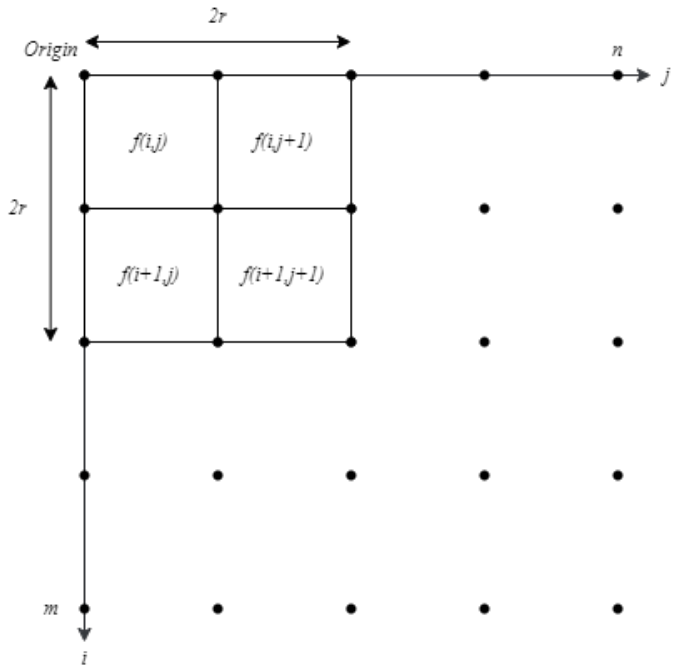


Figure 2: Window size of the box filter method (Author's Own, 2024)

Let $P(i, j)$ be the resultant intensity of a pixel after downsampling. For the source image, let $f(i, j)$ for $(i, j) \in (1, m)$ denote the intensities of the pixels. To find the average of the four pixel values depicted in Fig. 2, the formula could be derived as seen:

$$P(i, j) = \frac{1}{(2r)^2}[f(i, j) + f(i + 1, j) + f(i, j + 1) + f(i + 1, j + 1)]$$

Or, considering the case $r = 1$,

$$P(i, j) = \frac{1}{4}[f(i, j) + f(i + 1, j) + f(i, j + 1) + f(i + 1, j + 1)]$$

Thus, for an image with an even window size r, the general formula for $P(i, j)$ could be derived (Parthipan, 2017):

$$P(i, j) = \frac{1}{(2r)^2} \sum_{k=0}^{m} \sum_{l=0}^{m} f(i + k, j + l)$$

The box filter treats every source pixel with the same 'weight,' making the new pixel value simply the average of the pixels surrounding it. Down-sampling is not a mandatory part of the process and is often skipped to maintain the original size of the image.

3. Image Splitting

The image is then divided into data units called blocks. A block consists of an 8×8 square of pixels. For example, an image of size 512×512 pixels would be divided into

$$\frac{512 \times 512}{8 \times 8}$$

$$= 4096 \text{ blocks.}$$

For this extended essay, the assumption will be made that for any image with size MN pixels, the total number of pixels, MN, is divisible by 64. This would ensure that the image will be able to be split into an integer amount of blocks. If the total number of pixels is not divisible by 64, the image splitting process undergoes certain algorithms which are outside the scope of this investigation.

Once the image has been split into an appropriate amount of blocks, it is now ready to undergo the Discrete Cosine Transform (DCT). The rest of the steps in the JPEG algorithm will be further discussed in Chapters 5 and 6.

4 Introduction to Mathematical Transforms

4.1 The Purpose of Mathematical Transforms

The Discrete Cosine Transform at its core is a mathematical transform. There are many different types of mathematical transforms, two being integral and discrete transforms. These transforms relate a function in one domain to another function in a second domain. Before taking a look at the DCT, it would be helpful to investigate the Fourier Transform (FT).

4.2 The Fourier Transforms

The Fourier transform is a type of transform that converts a signal in the time domain—how the signal exists as time progresses, to its corresponding frequency domain—the magnitude of the signal's presence within each frequency. What is initially represented as a function of amplitude and time, $f(t)$, the Fourier transform maps this function into a function $F(\omega)$, where ω is the angular frequency. In the context of image processing, each image can be represented as a sum of sinusoids, each with differing frequencies. The function $f(t)$ would represent the intensity (amplitude) of a pixel at a specific location. When converting an image from the spatial (time) domain into the frequency domain, the resulting amplitude describes how much of each frequency component is present. In order to grasp the Fourier

transform, it is necessary to consider the Fourier series, as the transform is an extension of this series. The Fourier series takes a signal and describes it in terms of sine and cosine waves (Weisstein, n.d.)—sinusoids. For a function $f(x)$ in the time domain that is periodic over $2L$, the Fourier series is given by (Rocks, 2022):

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}\left(a_n \cos\left(\frac{nx\pi}{L}\right) + b_n \sin\left(\frac{nx\pi}{L}\right)\right)$$

Where a_0, a_n, and b_n are constants to be found. This equation can also be written in complex form (blackpenredpen, 2019), given by:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{j\omega_k x}$$

where:

$$c_n = \frac{1}{2L}\int_{-L}^{L} f(x)e^{j\omega_k x}dx$$

and:

$$\omega_k = \frac{n\pi}{L}$$

Now, a second function $F(\omega)$ can be defined such that:

$$F(\omega) = \int_{-L}^{L} f(x)e^{j\omega_k x}dx$$

This function indicates the frequency domain, containing information about the amplitude and frequency of each continuous signal. This provides the basis of the Fourier transform. Extending these limits to infinity (Lectures, 2020), the function $F(\omega)$ could be rewritten as:

$$F(\omega) = \int_{-\infty}^{\infty} f(\omega)e^{j\omega_k x}dx$$

This provides the Fourier transform as an integral (Lectures, 2020). While the integral form of the Fourier transform applies to continuous signals, in practice, many real world signals

are discrete. This applies to digital images as well. Images are sampled discretely, since pixel values are integers. Thus, this leads to the need for having a Discrete Fourier Transform (DFT). This transform is an extension of the continuous Fourier transform, except with a summation to replace the integral. Given a discrete signal with N sampled points, the Discrete Fourier Transform is given by (DSPRelated, n.d.):

$$X(\omega_k) = \sum_{n=0}^{N-1} x(n) e^{-j\omega_k n}$$

$X(\omega_k)$ represents the kth sample of the signal at a frequency ω_k.

5 The Discrete Cosine Transform

5.1 Deriving the DCT

Now that the overlying concepts of the Fourier transforms have been established, the process of arriving at the Discrete Cosine Transform (DCT) becomes much more simple. The DCT effectively comes from the DFT of a function, however the imaginary frequency nodes are dismissed. In practice, this could mean that the DCT is more efficient than the FFT or DFT because the DCT only uses real numbers. A comparison of the DCT and the FFT will be conducted later on in this essay. Consider the DFT:

$$X(\omega_k) = \sum_{n=0}^{N-1} x(n) e^{-j\omega_k n}$$

Taking the real portion of the LHS and the RHS (DSPRelated, n.d.),

$$\begin{aligned} re\left[X(\omega_k)\right] &= re\left[\sum_{n=0}^{N-1} x(n) e^{-j\omega_k n}\right] \\ &= \sum_{n=0}^{N-1} x(n) re\left[e^{-j\omega_k n}\right] \\ &= \sum_{n=0}^{N-1} x(n)\cos\left(\omega_k n\right) \end{aligned}$$

This result is one of the many forms of the DCT, of which there are many standard variants. In essence, the DCT is a variant of the DFT but with real and even functions. The DCT that was derived is a variant that is shifted by half a sample. However, the standard DCT used in image compression is different from the one obtained. This DCT, known as the DCT-II—or the 2D DCT—is the most common DCT when it comes to image processing, along with its variant, the DCT-III, or Inverse DCT (IDCT)—which will be discussed later. The DCT-II (or simply, the DCT) is given by (Express, 2019):

$$f(i,j) = \frac{2}{N} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\left(\frac{(2x+1)i\pi}{2N}\right) \cos\left(\frac{(2y+1)j\pi}{2N}\right)$$

where,

N = size of pixel block (length or width)

$f(i,j)$ = the coordinates of the transform domain

$f(x,y)$ = the coordinates of the source image

C = the cosine transform matrix

5.2 Application of the DCT in Image Compression

Now that the basis of the DCT has been established, it is time to apply it to image compression. Right before an image undergoes the DCT, a level shift is applied to the intensity values. This is because a standard cosine wave ranges between $[-1, 1]$ and not $[0, 1]$ and so an intensity range of $[0, 255]$ would not work. For an image with p bits, the level shift works as follows:

$$[0, 2^p - 1] \rightarrow [-(2^{p-1} - 1), 2^{p-1} - 1]$$

In the case of an 8-bit image, the range would now become $[-127, 127]$. This is so the range of intensities would match the shape of a cosine wave. In terms of interpreting the DCT,

the DCT function will take an input as an image and output a sum of 'basis' images—with each output representing the abundance of a certain basis image. Adding all individual basis images will provide the original image. The standard set of DCT basis images look like:

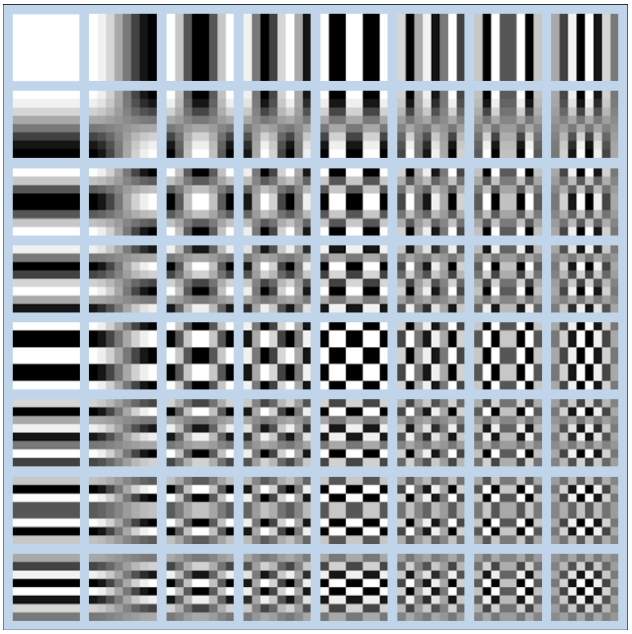


Figure 3: The basis images of the DCT (Roma, n.d.)

As shown, the basis images are an 8×8 square of black and white patterns—these are the patterns that make up a greyscale image. For RGB images, the luminance component of the source image is extracted and that is what is used as the input. That is why converting RGB to YCbCr is necessary. The patterns shown in Fig. 3 can also be represented as a function of i and j, and the output—given in an 8×8 matrix—would show the presence of each pattern. The outputs would be numerical values and these values are called the DCT coefficients. The lower frequency patterns, such as the ones in the top left, would be most of what the source image is composed of, and so they would usually result in having the highest DCT coefficients. The first block, given by $(i, j) = (0, 0)$ is called the DC coefficient. In order to actually compress the image, some of these coefficients are removed. This truncates

some high frequency data which contribute to the image size. To elucidate this process, an example will be used along with a sample DCT calculation.

5.3 Sample DCT Calculation

Recall the standard DCT-II equation, which would give us the DCT of an image with a width of N pixels. For this example, an 8×8 pixel block would be used, and so $N = 8$. Now, the formula for the DCT would become:

$$f(i,j) = \frac{1}{4}C(i)C(j)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left(\frac{(2x+1)i\pi}{16}\right)\cos\left(\frac{(2y+1)j\pi}{16}\right)$$

In order to actually use this formula, the matrix form of the DCT must be obtained. This matrix is called the transform matrix T, and is obtained by (MathWorks, n.d.):

$$T_{i,j} = \left\{ \begin{array}{ll} \frac{1}{\sqrt{N}} & i = 0 \\ \frac{\sqrt{2}}{N}\cos\left[\frac{(2j+1)i\pi}{2N}\right] & i > 0 \end{array} \right\}$$

So, when $i = 0$, the first case becomes

$$T_{0,j} = \frac{1}{\sqrt{8}}$$
$$\approx 0.354$$

Now, for each of the other cases, the transform matrix T would become,

$$T = \begin{bmatrix} 0.354 & 0.354 & 0.354 & 0.354 & 0.354 & 0.354 & 0.354 & 0.354 \\ \frac{1}{2}\cos\left[\frac{(2(0)+1)\pi}{16}\right] & \frac{1}{2}\cos\left[\frac{(2(1)+1)\pi}{16}\right] & \cdots & \cdots & \cdots & \cdots & \cdots & \frac{1}{2}\cos\left[\frac{(2(7)+1)\pi}{16}\right] \\ \frac{1}{2}\cos\left[\frac{(2(0)+1)2\pi}{16}\right] & \cdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \cdots & \cdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \cdots & \ddots & \vdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \cdots & \cdots & \ddots & \vdots \\ \frac{1}{2}\cos\left[\frac{(2(0)+1)7\pi}{16}\right] & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \frac{1}{2}\cos\left[\frac{(2(7)+1)7\pi}{16}\right] \end{bmatrix}$$

After simplification, the transform matrix would become:

$$T = \begin{bmatrix} 0.354 & 0.354 & 0.354 & 0.354 & 0.354 & 0.354 & 0.354 & 0.354 \\ 0.490 & 0.416 & 0.278 & 0.098 & -0.098 & -0.278 & -0.416 & -0.490 \\ 0.462 & 0.191 & -0.191 & -0.462 & -0.462 & -0.191 & 0.191 & 0.462 \\ 0.416 & -0.098 & -0.490 & -0.278 & 0.278 & 0.490 & 0.098 & -0.416 \\ 0.354 & -0.354 & -0.354 & 0.354 & 0.354 & -0.354 & -0.354 & 0.354 \\ 0.278 & -0.490 & 0.098 & 0.416 & -0.416 & -0.098 & 0.490 & -0.278 \\ 0.191 & -0.462 & 0.462 & -0.191 & -0.191 & 0.462 & -0.462 & 0.191 \\ 0.098 & -0.278 & 0.416 & -0.490 & 0.490 & -0.416 & 0.278 & -0.098 \end{bmatrix}$$

Now, to go ahead and proceed with the calculation of the DCT, a sample 8×8 image would have to be created in which the pixel values are known. I decided to take a photo that I have previously taken while at a beach. I converted the image from RGB to greyscale in order to make the process easier to work with. The image is shown below:

Figure 4: Image taken at a beach (Author's Own, 2024)

For the sake of simplicity, I decided that I would just calculate the DCT for the first 8×8 pixels. This would provide the DCT of a single block. In practice, this process would have to be done an incredible amount of times to get the DCT of each 8×8 pixel block, however that process becomes almost immediate with computer technology in the modern day. The first 8×8 pixels of the image are shown below:

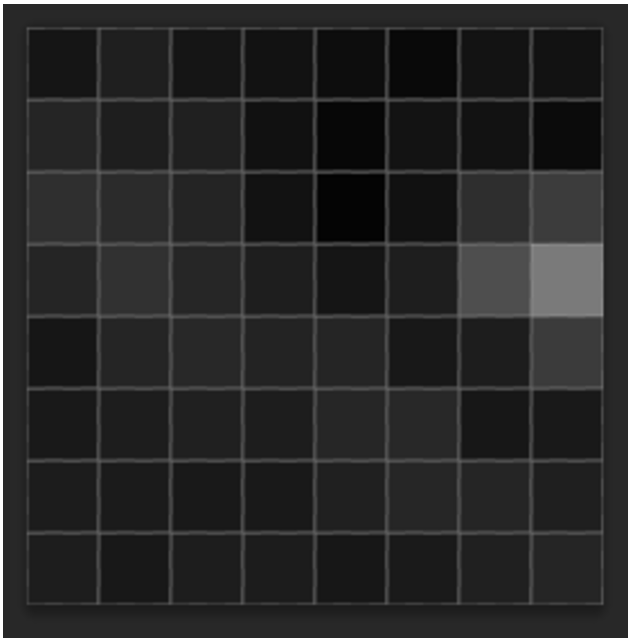


Figure 5: First pixel block of the beach image (Author's Own, 2024)

I used a python program in order to actually extract the pixel values from the image. Refer to **Appendix 1.1** for the code used to extract the intensities of each pixel. Represented below are the intensities for each respective pixel:

$$\text{Original} = \begin{bmatrix} 21 & 31 & 21 & 18 & 13 & 9 & 19 & 18 \\ 37 & 30 & 32 & 17 & 7 & 19 & 18 & 11 \\ 47 & 43 & 36 & 18 & 4 & 17 & 46 & 60 \\ 37 & 49 & 38 & 30 & 21 & 30 & 78 & 122 \\ 22 & 37 & 40 & 35 & 37 & 24 & 29 & 59 \\ 25 & 29 & 32 & 29 & 38 & 40 & 23 & 25 \\ 28 & 27 & 25 & 25 & 32 & 38 & 37 & 31 \\ 29 & 24 & 29 & 28 & 23 & 26 & 32 & 37 \end{bmatrix}$$

Recall that before the DCT can be applied to the matrix, a level shift needs to be applied. Therefore, 128 would need to be subtracted from each of the pixel values so that the center would be around 0. Refer to **Appendix 1.2** for the code used to apply the level shift. The matrix, M is as follows:

$$M = \begin{bmatrix} -107 & -97 & -107 & -110 & -115 & -119 & -109 & -110 \\ -91 & -98 & -96 & -111 & -121 & -109 & -110 & -117 \\ -81 & -85 & -92 & -110 & -124 & -111 & -82 & -68 \\ -91 & -79 & -90 & -98 & -107 & -98 & -50 & -6 \\ -106 & -91 & -88 & -93 & -91 & -104 & -99 & -69 \\ -103 & -99 & -96 & -99 & -90 & -88 & -105 & -103 \\ -100 & -101 & -103 & -103 & -96 & -90 & -91 & -97 \\ -99 & -104 & -99 & -100 & -105 & -102 & -96 & -91 \end{bmatrix}$$

The matrix is now ready to undergo the DCT. To calculate the DCT of this image, the following formula is used (MathWorks, n.d.):

$$DCT = T \times M \times T'$$

T' denotes the transpose of the transform matrix T.

The transpose of the transform matrix is shown below:

$$T' = \begin{bmatrix} 0.354 & 0.490 & 0.462 & 0.416 & 0.354 & 0.278 & 0.191 & 0.098 \\ 0.354 & 0.416 & 0.191 & -0.098 & -0.354 & -0.490 & -0.462 & -0.278 \\ 0.354 & 0.278 & -0.191 & -0.490 & -0.354 & 0.098 & 0.462 & 0.416 \\ 0.354 & 0.098 & -0.462 & -0.278 & 0.354 & 0.416 & -0.191 & -0.490 \\ 0.354 & -0.098 & -0.462 & 0.278 & 0.354 & -0.416 & -0.191 & 0.490 \\ 0.354 & -0.278 & -0.191 & 0.490 & -0.354 & -0.098 & 0.462 & -0.416 \\ 0.354 & -0.416 & 0.191 & 0.098 & -0.354 & 0.490 & -0.462 & 0.278 \\ 0.354 & -0.490 & 0.462 & -0.416 & 0.354 & -0.278 & 0.191 & -0.098 \end{bmatrix}$$

I wrote code in Python in order to carry out the matrix multiplication, and the resultant matrix provides the DCT for the 8×8 pixel block. Refer to **Appendix 1.3** for the code

used to carry out the matrix multiplication. The DCT has now been obtained:

$$DCT = \begin{bmatrix} -777.0 & -17.0 & 45.0 & -28.0 & -3.0 & -4.0 & 0.0 & 0.0 \\ -17.0 & 17.0 & 26.0 & -12.0 & -6.0 & 4.5 & -4.9 & -0.7 \\ -57.3 & 32.9 & -28.0 & 27.0 & -8.1 & 11.2 & 0.3 & -0.3 \\ -26.2 & 14.9 & -44.8 & 16.1 & 0.0 & -11.9 & 0.4 & 0.1 \\ 17.5 & -23.9 & 12.1 & -25.2 & 11.5 & -7.4 & -8.0 & 0.0 \\ 23.8 & -28.0 & 24.7 & -5.6 & -0.2 & -0.2 & -9.5 & -1.0 \\ -4.9 & 6.1 & 0.8 & -0.2 & 0.1 & 0.0 & 0.0 & -0.2 \\ -12.4 & 12.1 & -6.3 & -0.5 & 1.5 & -1.3 & -1.7 & 2.2 \end{bmatrix}$$

These values represent the weight of each of the basis DCT patterns present in the source image. Notice how the top left value, the DC coefficient is the highest out of all of them.

5.4 Quantization and The Inverse DCT (IDCT)

After the DCT has been calculated, there are still a few more steps to actually compress the image. The first step is quantization. Quantization is the process in which the high frequency patterns are removed (Jiaqi, 2023). This is where the quality of the compressed image relative to its source is determined. This value ranges from 1 to 100, where 1 represents the most amount of compression and the lowest file quality, and 100 represents no compression and maintaining the original quality. The standard JPEG quantization value is 50, which saves disk space while also keeping a moderate quality for the image. Quantization is represented in an 8×8 matrix known as a quantization table. The quantization table for the standard quality of 50 is:

$$Q(i,j) = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure 6: JPEG Standard Quantization Values (Parker, 2017)

To undergo the process of quantization, the following formula is used:

$$C(i,j) = \text{round}\left(\frac{f(i,j)}{Q(i,j)}\right)$$

The round function is applied to round each quantized value to the nearest integer. Dividing the DCT matrix—$f(i,j)$ by the quantization matrix—$Q(i,j)$ would yield the following:

$$C(i,j) = \begin{bmatrix} -49 & -2 & 5 & -2 & 0 & 0 & 0 & 0 \\ -1 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ -4 & 3 & -2 & 1 & 0 & 0 & 0 & 0 \\ -2 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This matrix now displays the quantized values of the source image. Notice that many of the values in this matrix are now 0. This represents the amount of data that has been truncated. The next step is to reconstruct the original image. The first process in reconstructing this image is to de-quantize the values, represented in a matrix $DQ(i,j)$. To do this, the values given in the quantized matrix are simply re-multiplied by the values in the quantization table. This is formally represented by:

$$C(i,j)Q(i,j) = DQ(i,j)$$

Now, we must not get confused here because we are not performing matrix multiplication,

rather we are simply multiplying each value in the dequantized matrix by its corresponding value (the one with the same position) in the matrix $C(i, j)$. Using these de-quantized values, the next step to build the compressed image is to apply the Inverse DCT (IDCT) of this matrix. The Inverse DCT of this matrix is given by simply multiplying the de-quantized matrix by the transpose matrix first, then multiplying the resultant matrix by the transform matrix. This is seen in the following equation (remember, order matters):

$$IDCT = T' \times DQ \times T$$

This formula takes advantage of the fact that T is an orthonormal matrix, meaning its inverse is the same as its transpose. Applying the IDCT will result in the level-shifted pixel intensities of the compressed image. After that, all that is needed to do is revert the level shift to attain the pixel intensities of the final compressed image.

Refer to **Appendix 1.4** for the code used to carry out the matrix multiplication in order to find the IDCT. The IDCT of this matrix is:

$$IDCT = \begin{bmatrix} -106 & -107 & -108 & -108 & -108 & -109 & -111 & -114 \\ -90 & -91 & -98 & -114 & -130 & -131 & -117 & -103 \\ -90 & -85 & -89 & -108 & -125 & -115 & -79 & -45 \\ -98 & -90 & -89 & -99 & -107 & -91 & -54 & -20 \\ -97 & -95 & -94 & -99 & -103 & -98 & -84 & -72 \\ -102 & -102 & -102 & -100 & -98 & -99 & -102 & -106 \\ -106 & -106 & -104 & -100 & -95 & -93 & -94 & -97 \\ -99 & -100 & -101 & -104 & -106 & -104 & -100 & -96 \end{bmatrix}$$

Finally, the level shift will be reverted by adding 128 to each value in the matrix. The final matrix, $f(i,j)$, is now:

$$f(i,j) = \begin{bmatrix} 22 & 21 & 20 & 20 & 20 & 19 & 17 & 14 \\ 38 & 37 & 30 & 14 & -2 & -3 & 11 & 25 \\ 38 & 43 & 39 & 20 & 3 & 13 & 49 & 83 \\ 30 & 38 & 39 & 29 & 21 & 37 & 74 & 108 \\ 31 & 33 & 34 & 29 & 25 & 30 & 44 & 56 \\ 26 & 26 & 26 & 28 & 30 & 29 & 26 & 22 \\ 22 & 22 & 24 & 28 & 33 & 35 & 34 & 31 \\ 29 & 28 & 27 & 24 & 22 & 24 & 28 & 32 \end{bmatrix}$$

It can be seen that two of these values are below 0. This is quite an anomaly as the reverted level shift is meant to set the range of these pixel values to $[0, 255]$. This could have happened due to rounding inconsistencies throughout the entire process. To fix this, these values could be 'clipped' so they stay within the valid range. In this case, the values -2 and -3 would just become 0. The matrix provided will be pieced together along with the many other 8×8 matrices to rebuild the image. When comparing this matrix to the original matrix, it is shown that the values are mostly similar and in the full image, the differences are barely detectable by the human eye.

5.5 Comparing the DCT and FTs

The similarities between the original matrix and the compressed matrix using the DCT could be observed, however it is still questioned whether the DCT is the most effective method to compress images. Therefore, a small comparison will be conducted between the DCT and one of the Fourier Transforms over the same input.

Given a sample 1-dimensional input:

$$input = [8, 16, 24, 32, 40, 48, 56, 64]$$

The DCT will provide (Marshall, 2001):

$$DCT = [101, -51, 0, 5, 0, -1, 0, 0]$$

After truncating the last four values (making them 0), the inverse DCT will provide:

$$IDCT = [7, 15, 23, 31, 39, 48, 56, 63]$$

Now, the same input values will be put into the Fast Fourier Transform compression algorithm and then the Inverse FFT (IFFT) will also be calculated.

The FFT of the sample input will provide (Marshall, 2001):

$$FFT = [36, 10, 10, 6, 6, 4, 4, 4]$$

After truncating the same last four values, the inverse FFT will provide:

$$IFFT = [24, 12, 20, 32, 40, 51, 59, 48]$$

Upon visual comparison, it can be seen that the IDCT of the one-dimensional input values is very close to the original input values. It can also be seen that the IFFT is not too far off the original input values, however there is a bigger discrepancy between them.

5.6 Analysis of findings

A graph of these two results could be plotted on desmos to provide a visual representation of the difference between the DCT and the FFT. This is shown on Figure 7 below.

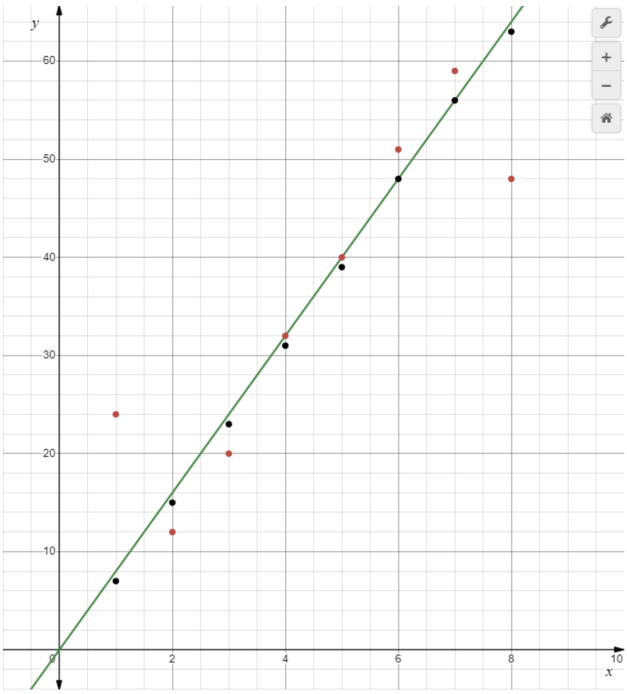


Figure 7: A visual comparison of the DCT and FFT using Desmos (Author's Own, 2024)

The line $y = 8x$ is shown in green. This resembles the original input values. The black points symbolize the DCT, and it can be seen that the values are very close to the line $y = 8x$. The red points represent the FFT, and there are noticeable deviations from the line $y = 8x$. A reason for this is not only because the DCT only evaluates real coefficients, however the periodicity assumed by the DCT is implicitly continuous. The periodicity assumed by the Fourier transforms are not continuous, they have jump discontinuities at the period boundaries (queensu, n.d.). This means that more terms need to be added to the series in order to achieve a desired result. It can thus be said that the DCT is the most efficient transform to undergo image compression.

6 Completing the Compression Process

Although the general procedure for image compression was outlined in Chapters 3 and 5, there are some more steps in the actual image compression process that were skipped for the sake of simplicity. A major one that was skipped is the encoding process. This process occurs after the quantization process and in order for this operation to be undergone, the matrix needs to be linearized to create a 1×64 matrix. A common encoding technique used is Huffman encoding (GeeksForGeeks, 2023). Skipping the encoding process may also have contributed to the negative pixel values after the inverted level shift.

7 Extensions and Limitations

The DCT does not solely apply to greyscale images. It also works on coloured images. In Chapter 3 the topic of coloured images was touched upon. Coloured images are an extension of the greyscale image. Along with luminance, coloured images introduce the concept of chrominance. Chrominance is the measure of the color of the emitted light. There are two ways to represent the colour of these images. The first way is a combination of the colours red, green and blue, commonly referred to as 'RGB' or RGB images. Along with the RGB representation of an image, the other form, and the one applied in image compression is the YCbCr model. This model incorporates Y', which is the luminosity of the image, as discussed previously. Cb and Cr represent the blue-difference chrominance and red-difference chrominance respectively.

To extend the scope of this investigation, a next step would be to fully investigate the basis behind the DCT and do an in-depth analysis of the Fourier series and transform's

derivations. Another possible step that could be taken is to explore the encoding process and quite possibly its role in mitigating the anomaly behind the negative pixel intensities after the inverted level shift. Instead of only sampling one 8×8 pixel block, the DCT calculations could be expanded in order to reconstruct an entire image. Finally, this investigation could be expanded to calculate the DCT of coloured images. One limitation to this investigation is that the pixel block after the DCT could not be reconstructed. It would have been helpful to have a visual comparison to notice how subtle the differences are.

8 Conclusion

In this essay, the aim was to answer the following research question: *To what extent is the Discrete Cosine Transform involved and effective in JPEG image compression?*

This essay briefed how the DCT is derived from the Fourier transforms, and the Fourier transform derivation was also briefly looked at. This essay also described how grayscale images can be represented in terms of matrices, and some basics behind the JPEG Algorithm, such as down-sampling and the colour shift. However, the main focus of this essay was to take a look at the DCT and its application within the JPEG Algorithm. By taking a look at the entire JPEG compression process, it is evident that the DCT is an integral part of image processing. To concretize this idea, a sample calculation of the DCT using an image taken by the author was completed. The efficiency of the DCT was also challenged as it was placed in comparison to the FFT. The results obtained showed that the DCT remains the most effective method to compress images, due to its continuous periodicity in the real plane. Finally, an outline of possible extensions was explored and discussed in order to add to the

depth and nuance within this essay. With these extensions, the DCT and its effectiveness can be viewed in true reverence to grasp the incredible success of the JPEG format.

Appendices

Appendix 1.1

Code used to extract the intensities of each pixel. Code was created on Visual Studio Code using the Python Programming Language. Code inspired by (Pai, 2014).

```
1  from PIL import Image
2  import numpy as np
3  im = Image.open('88.png').convert('L')
4  pix_val = list(im.getdata())
5  print(pix_val)
```

Appendix 1.2

Code used to apply the level-shift to the matrix. Addition was implemented by (Author, 2024).

```
1  from PIL import Image
2  import numpy as np
3  im = Image.open('88.png').convert('L')
4  pix_val = list(im.getdata())
5  pix_val = [i - 128 for i in pix_val]
6  print(pix_val)
```

Appendix 1.3

Code used to calculate the DCT of the level-shifted matrix (Author's Own, 2024).

```python
import numpy as np

#defining the DCT matrix as the original matrix as an array
original_matrix = np.array([
    [0.354, 0.354, 0.354, 0.354, 0.354, 0.354, 0.354, 0.354],
    [0.490, 0.416, 0.278, 0.098, -0.098, -0.278, -0.416, -0.490],
    [0.462, 0.191, -0.191, -0.462, -0.462, -0.191, 0.191, 0.462],
    [0.416, -0.098, -0.490, -0.278, 0.278, 0.490, 0.098, -0.416],
    [0.354, -0.354, -0.354, 0.354, 0.354, -0.354, -0.354, 0.354],
    [0.278, -0.490, 0.098, 0.416, -0.416, -0.098, 0.490, -0.278],
    [0.191, -0.462, 0.462, -0.191, -0.191, 0.462, -0.462, 0.191],
    [0.098, -0.278, 0.416, -0.490, 0.490, -0.416, 0.278, -0.098]
])

#transposing the DCT matrix
transpose_matrix = np.transpose(original_matrix)

#defining the level shifted matrix
levelshift = np.array([
    [-107, -97, -107, -110, -115, -119, -109, -110],
    [-91, -98, -96, -111, -121, -109, -110, -117],
    [-81, -85, -92, -110, -124, -111, -82, -68],
    [-91, -79, -90, -98, -107, -98, -50, -6],
    [-106, -91, -88, -93, -91, -104, -99, -69],
    [-103, -99, -96, -99, -90, -88, -105, -103],
    [-100, -101, -103, -103, -96, -90, -91, -97],
    [-99, -104, -99, -100, -105, -102, -96, -91]
])

# perform matrix multiplication
result_matrix = np.dot(np.dot(original_matrix, levelshift), transpose_matrix)
result_matrix_rounded = np.round(result_matrix,1)

print(result_matrix_rounded)
```

Appendix 1.4

Code used to calculate the IDCT of the dequantized matrix (Author's Own, 2024).

```python
import numpy as np

#defining the DCT matrix as the original matrix as an array
original_matrix = np.array([
    [0.354, 0.354, 0.354, 0.354, 0.354, 0.354, 0.354, 0.354],
    [0.490, 0.416, 0.278, 0.098, -0.098, -0.278, -0.416, -0.490],
    [0.462, 0.191, -0.191, -0.462, -0.462, -0.191, 0.191, 0.462],
    [0.416, -0.098, -0.490, -0.278, 0.278, 0.490, 0.098, -0.416],
    [0.354, -0.354, -0.354, 0.354, 0.354, -0.354, -0.354, 0.354],
    [0.278, -0.490, 0.098, 0.416, -0.416, -0.098, 0.490, -0.278],
    [0.191, -0.462, 0.462, -0.191, -0.191, 0.462, -0.462, 0.191],
    [0.098, -0.278, 0.416, -0.490, 0.490, -0.416, 0.278, -0.098]
])

#transposing the DCT matrix
transpose_matrix = np.transpose(original_matrix)

#defining the dequantized matrix
dequantized = np.array([
[-784, -22, 50, -32, 0, 0, 0, 0],
  [-12, 12, 28, -19, 0, 0, 0, 0],
  [-56, 39, -32, 24, 0, 0, 0, 0],
  [-28, 17, -44, 29, 0, 0, 0, 0],
  [18, -22, 0, 0, 0, 0, 0, 0],
  [24, -35, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0]
])

# perform matrix multiplication to arrive at IDCT
result_matrix = np.dot(np.dot(transpose_matrix, dequantized), original_matrix)
result_matrix_rounded = np.round(result_matrix)

print(result_matrix_rounded)
```

References

A beginner's guide to JPEG files. The Craft. (2022, October 25). https://shorthand.com/the-craft/what-is-a-jpegfile/index.html

blackpenredpen. (2019, January 6). *Complex Fourier Series*. YouTube. https://www.youtube.com/watch?app=desktop&v=aC0j8CW58AM&t=7m44s

The discrete cosine transform (DCT): Mathematics of the DFT. DSP. (n.d.-a). Retrieved October 14, 2024. https://www.dsprelated.com/freebooks/mdft/Discrete_Cosine_Transform_DCT.html

Express, L. (2019, March 1). *Lecture 24: DCT Discrete Cosine Transform Image Compression*. YouTube. https://www.youtube.com/watch?v=Iu_ocMVcqwY&t=1251s

Fourier transforms for continuous/discrete time/frequency. DSP. (n.d.-b). https://www.dsprelated.com/freebooks/sasp/Fourier_Transforms_Continuous_Discrete_Time_Frequency.html

GeeksForGeeks. (2023, September 11). *Huffman coding: Greedy Algo-3*. GeeksforGeeks. https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

Gonzales, R. C., & Woods, R. E. (2002). *Introduction to image processing*. SlideServe. https://www.slideserve.com/fedora/introduction-to-image-processing-powerpoint-ppt-presentation#google_vignette

Jiaqi, Z. (2023, June 9). *Understanding DCT and quantization in JPEG compression*. DEV Community. https://dev.to/marycheung021213/understanding-dct-and-quantization-in-jpeg-compression-1col

JPEG encoding. CISC 457 - JPEG Encoding. (n.d.). Retrieved October 14, 2024. https://watkins.cs.queensu.ca/~jstewart/457/notes/30/30-jpeg-encoding.html

Kim, E. (n.d.). Retrieved October 14, 2024. *Project 5: Image Compression*. https://mason.gmu.edu/~ekim19/project5.html

Marshall, D. (2001, October 4). *The discrete cosine transform (DCT)*. https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html

MathWorks. (n.d.). Retrieved October 14, 2024. *Discrete Cosine Transform*. Discrete cosine transform - MATLAB& simulink. https://www.mathworks.com/help/images/discrete-cosine-transform.html

Pai, A. (2014). *Extracting pixel values of an image in Python*. HackerEarth. https://www.hackerearth.com/practice/notes/extracting-pixel-values-of-an-image-in-python/

Parker, M. (2017). *Digital Signal Processing 101*. Quantization Table - an overview. https://www.sciencedirect.com/topics/engineering/quantization-table

Parthipan, V. (2017). *Image down-scaler using the box filter algorithm*. https://repository.rit.edu/cgi/viewcontent.cgi?article=10864&context=theses

Physics and Math Lectures. (2020, July 2). *Deriving the Fourier Transform from Fourier Series*. YouTube. https://www.youtube.com/watch?v=wmUdNKLrWeo

Rocks, T. (2022, December 13). *Oxford Calculus: Fourier Series Derivation*. YouTube. https://www.youtube.com/watchv=LEonZYoW6sk

Roma, N. (n.d.). Retrieved October 14, 2024. *2-D DCT basis functions*. ResearchGate. https://www.researchgate.net/figure/2-D-DCT-basis-functions_fig2_220227009

Weisstein, E. W. (n.d.). Retrieved October 14, 2024. *Fourier series*. Wolfram MathWorld. https://mathworld.wolfram.com/FourierSeries.html