Monochromatic Image Compression

With Discrete Cosine Transform – a Fourier Transform variant – and matrix manipulation.

By Kartavya Sharma

Table of Contents

Introduction

Efficiency and optimized utilization of resources is one of the most consistent goals we strive for. How to fit more into less. Undergoing a rapid increase in data flow and information generation, our world has limited means to curate vast amounts of data. One of the most fundamental applications of data is in computer systems, which take data as an input and deliver some information based on it. The most pressing issue, among others, is to store this data in such a information dependent world. We do not have unlimited space available to us, but the amount of data being generated can certainly reach, relatively, to unlimited amounts. To overcome such problems we use compression, a means through which the space data occupies in any container or storage medium is drastically or relatively reduced.

Data can be represented in bits – this representation is called *encode*. Data compression is the process of bit-rate reduction whereby data already available to the system is encoded using relatively fewer bits. In other words, it is the process of reducing the size of a data unit. That being said, data is useful if it can be shared – before each transmission data is encoded and prior to that is present in the form of its source coding. Transmission and processing of data required resources, by compressing it we are reducing the resources required for the 2 aforementioned processes.

My exploration will revolve around the Discrete cosine transform (DCT) and its application in Lossy Image Compression. DCT is adapted from the Fourier Transform, due to this I will begin by talking about the Fourier Transform and its fundamentals. Then I will state the DCT and connect how they are both interconnected. The next phase of my exploration would entail DCT's application in image compression and how this process can be segregated into different phases, each phase will be explored in sufficient depth. DCT's application phase involves matrix manipulation and dealing with equations which aid in this process. This exploration would culminate into the practical example showing the theoretical mathematics in action. I chose to pursue such a topic because it fuels my interest in computational systems while giving me insight into an ongoing global epidemic for efficiency and optimization. Through this exploration I hope to gain a new perspective on data compression and optimization, and what role mathematics has to play behind this obscure, yet important, process. I also hope that this exploration opens up other avenues of intriguing aspects in the efficiency and optimization domain.

The Fourier Transform

The Fourier Transform method states[1] for any sequence of N complex numbers $(x_0, x_1, \dots, x_{N-1})$ to a new sequence of N complex numbers. The equation responsible for this method is as follows:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i(2\pi k \frac{n}{N})}$$

The above equation holds for $0 \leq K \leq N - 1$.

The Fourier Transform is a time-based pattern which measures every possible cycle and returns the overall "cycle recipe" (amplitude, phase, and rotation speed for every cycle that was found)[2].

[1] Discrete Fourier Transform. *Brilliant.org*. Retrieved 16:59, March 17, 2019, from https://brilliant.org/wiki/discrete-fourier-transform/

[2] "An Interactive Guide To The Fourier Transform." *BetterExplained*, betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/.

A simple breakdown of the Fourier Transform

$$X_k$$

The Fourier Transform is a list of N complex numbers – a number composed of a *real* and *imaginary* part. Each complex number in the Transform list encodes the amplitude and phase of a single phasor – a line denoting a quantity as a vector.

$$x_n$$

A random time-domain signal with length of N samples. x_n will act as an *input* to the Fourier Transform method, this is the data set on which the function will base its *output* on.

$$e^{-i(2\pi k\frac{n}{N})} = e^{-i\varphi}$$

The notation $[e^{-i\varphi}]$ is a compact way of denoting a pair of $sine$ and $cosine$ waves. This way of representation was pursued due to its relation with the Euler's formula (more detail in the appendix, with relation to cryptography and file transfer). The notation $[2\pi k\frac{n}{N}]$ is a way to denoting the phase and frequency of a *sine* wave.

$$\sum_{n=0}^{N-1} x_n \cdot y_n \ \because\ y_n = \ e^{-i\varphi}$$

The *dot-product* is essential to the Fourier Transform. This dot product is what will enable us to connect the *sine* and the *cosine* sinusoidal waves.

The Fourier Transform's output contains complex numbers in the form $a + bi$ where they have a real part (a) and an imaginary part (b). This can be represented on the complex plane in various ways.

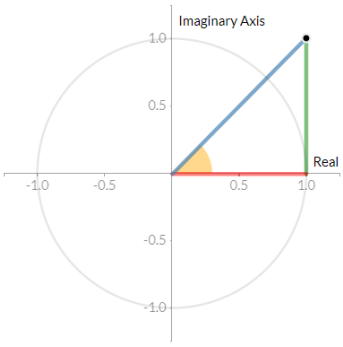


Figure 1: Complex Number on the Argand Diagram

Figure 1[3] represents a complex plane representing the complex number $1 + 1i$. Where the real part is expressed on the $x - axis$ and the imaginary part is expressed on the $y - axis$. Since we will soon move on to waves the circle will play a very integral role in its explanations.

There are two forms of representation on the complex plane: *cartesian coordinates* and *polar coordinates*. The polar form of representation is more elegant and precise while as the cartesian system is easier to work with. This relation between polar and cartesian is established by Euler's formula:

$$e^{\varphi i} = \cos(\varphi) + \sin(\varphi)i$$

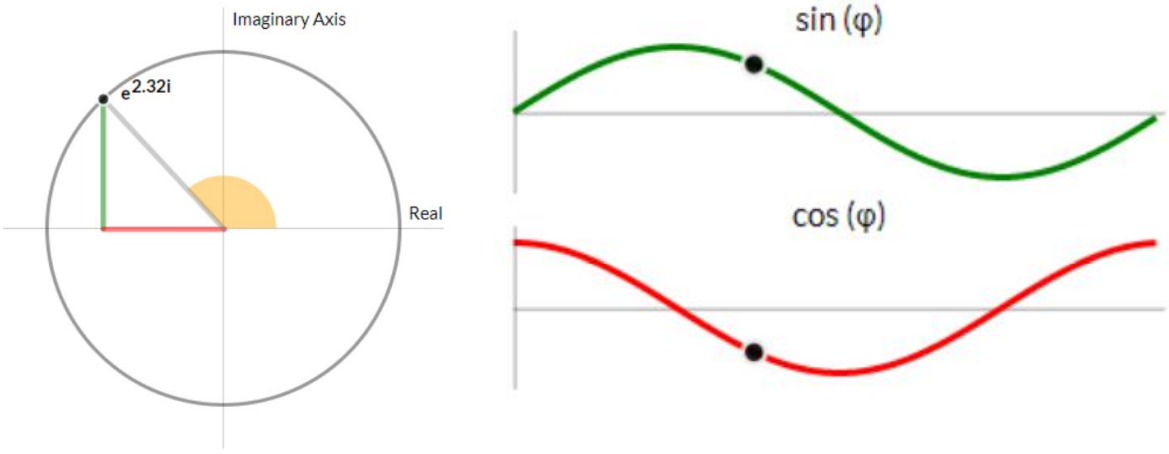
The formula is establishing a relationship between e and the Unit Circle.

[3] *Circles Sines and Signals - Complex Numbers*, jackschaedler.github.io/circles-sines-signals/complex.html.

A word on Euler's Formula

This relationship established by the Euler's Formula between e and the unit circle tells us that e raised to the power any imaginary number φ [$e^{\varphi i}$] will denote a point on the unit circle. With that said, we are also aware that any point on the unit circle can be represented in the cartesian form which makes use of *sine* and *cosine* waves.

$$e^{\varphi i} = \cos(\varphi) + \sin(\varphi)i$$



Final Geometric Interpretation of the Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\varphi i}$$

We can now use the Euler's formula to make a substitution in the equation to highlight the significance of the dot product and the result it will yield. Since $e^{-\varphi i} = \cos(\varphi) + \sin(\varphi)i$ (see 'Proofs'), therefore:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot (\cos(\varphi) + \sin(\varphi)i)$$

The dot product will aid in the construction of phasors – a complex quantity represented as a vector – which will ultimately represent our signal.

The dot product is at the core of the Fourier Transform. The dot product is calculated by taking two signals of equal length and multiplying their elements together. The standard Fourier Transform notation changes to a certain degree to facilitate this change:

$$X_k = \sum_{n=0}^{N-1} x[n] \cdot b[n]$$

Given that (see 'Appendix' for further explanation on the below vector arrays):

$$x[n] = [3, 5, 7, 9, 4]$$
$$b[n] = [8, 2, 5, 3, 1]$$

Hence, $x[n] \cdot b[n] = (3 \times 8) + (5 \times 2) + (7 \times 5) + (9 \times 3) + (4 \times 1) = 100$

On the surface such an operation might seem trivial, but in this context, it holds important value since it will enable us to measure the degree to which two signals or vectors are pointing or moving in the same direction. If they are *perpendicular*, with absolutely different paths, their dot products would be 0. Their dot product would be maximum when the two signals are running parallel in the *same direction*. The dot product may also be negative, indicating some similarity between the two vectors but *opposite* directions.

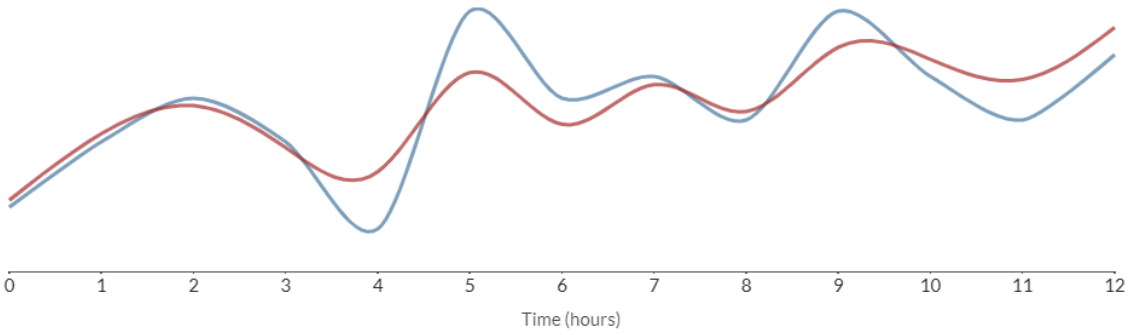


Figure 2: Example of Signals with Positive Correlation

The dot product of the two waves above, *Figure 2*[4], with a positive correlation, will help us measure this correlations magnitude. To better understand this for waves with many values, we can make use of *sine* waves. The dot product achieved from two *sine* waves ***A*** and ***B*** will show This correlation successfully.

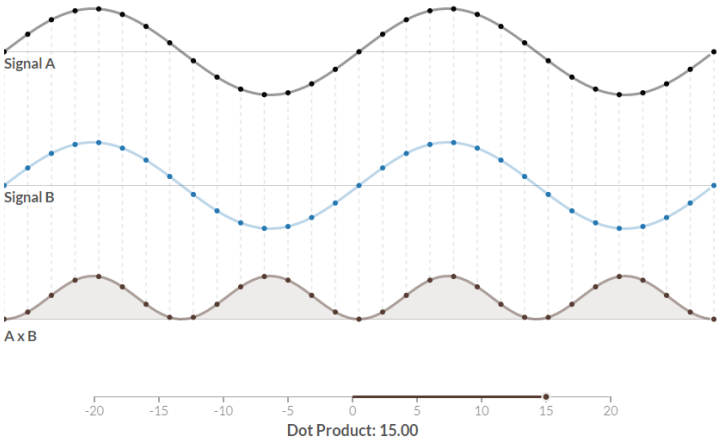


Figure 3: Dot product of two sine waves with 0-degree phase difference

Figure 3[5] shows that when the two *sine* waves are in complete phase their dot product is at its maximum, as stated previously this denotation is a concise way of comparing two signals and their movement.

As the waves will move out of phase their dot product will decrease owing to the reduction in the similarity between the two wave's directions.

Next, we will talk about how the Fourier Transform as a whole works.

[4] *Circles Sines and Signals - Complex Numbers*, jackschaedler.github.io/circles-sines-signals/dotproduct2.html.

[5] *Circles Sines and Signals - Complex Numbers*, jackschaedler.github.io/circles-sines-signals/dotproduct2.html.

The waves we've been dealing with are non-compounded waves which are only composed of one single waves. Fourier Transform's full capability is in analyzing compound waves, e.g. Square waves, and determining its components.

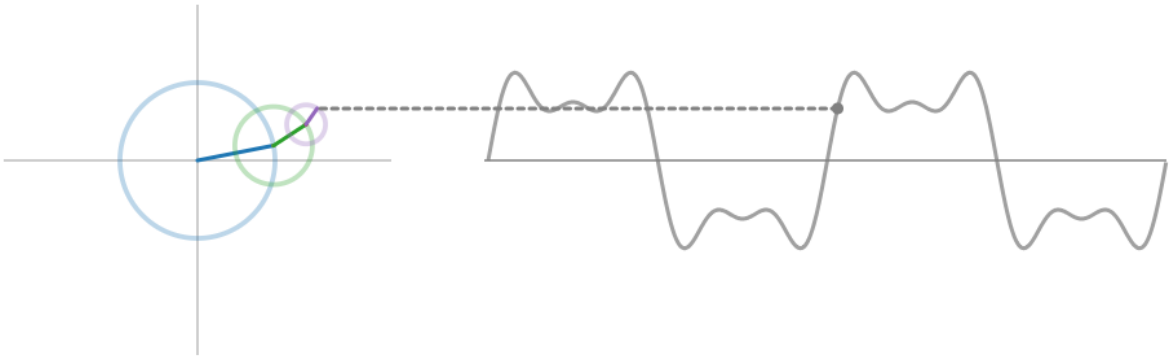


Figure 4: Compound Square wave using three phasors (circles)

The Fourier Transform from this point is quite straight forward:

1. Find the dot product of the *sine* and the square wave.
2. If the dot product is non-zero then the compound wave contains that sinusoidal wave.
3. Adjust the frequency of the *sine* wave and run it through the transform multiple times to determine the "recipe" for the compound wave.

Each time upon adjusting the frequency the dot product is non-zero that frequency is present in the compound wave form. Whenever we would get a zero as a dot product, we can immediately know that that wave form is not present in the compound wave.

Stating the Discrete Fourier Transform for reference for the next example.

For a sequence of $\boldsymbol{N}$ complex numbers $\{\boldsymbol{x_n}\} = \boldsymbol{x_0, x_1, \dots, x_{N-1}}$ into a different sequence of complex number $\{\boldsymbol{X_k}\} = \boldsymbol{X_0, X_1, \dots, X_{N-1}}$ which is defined by the equation:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\left(2\pi k \frac{n}{N}\right)}$$

$$= \sum_{n=0}^{N-1} x_n \cdot [\cos\left(\frac{2\pi kn}{N}\right) - i \cdot \sin\left(\frac{2\pi kn}{N}\right)]$$

Let $\boldsymbol{N}$ = 4.

$\boldsymbol{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ 12 - i \\ -i \\ -1 + 6i \end{pmatrix}$ ** The values of (x_0, x_1, x_2, x_3) in $\boldsymbol{x}$ are sample values for calculation.

Calculating the DFT (Discrete Fourier Transform) for $\boldsymbol{x}$**:**

$$X_0 = \left[6 \cdot e^{-i2\pi 0 \cdot \frac{0}{4}}\right] + \left[(12 - i) \cdot e^{-i2\pi 0 \cdot \frac{1}{4}}\right] + \left[(-i) \cdot e^{-i2\pi 0 \cdot \frac{2}{4}}\right] + \left[(-1 + 6i) \cdot e^{-i2\pi 0 \cdot \frac{3}{4}}\right] = 17 + 4i$$

$$X_1 = \left[6 \cdot e^{-i2\pi 1 \cdot \frac{0}{4}}\right] + \left[(12 - i) \cdot e^{-i2\pi 1 \cdot \frac{1}{4}}\right] + \left[(-i) \cdot e^{-i2\pi 1 \cdot \frac{2}{4}}\right] + \left[(-1 + 6i) \cdot e^{-i2\pi 1 \cdot \frac{3}{4}}\right] = -1 - 12i$$

$$X_2 = \left[6 \cdot e^{-i2\pi 2 \cdot \frac{0}{4}}\right] + \left[(12 - i) \cdot e^{-i2\pi 2 \cdot \frac{1}{4}}\right] + \left[(-i) \cdot e^{-i2\pi 2 \cdot \frac{2}{4}}\right] + \left[(-1 + 6i) \cdot e^{-i2\pi 2 \cdot \frac{3}{4}}\right] = -5 - 6i$$

$$X_3 = \left[6 \cdot e^{-i2\pi 3 \cdot \frac{0}{4}}\right] + \left[(12 - i) \cdot e^{-i2\pi 3 \cdot \frac{1}{4}}\right] + \left[(-i) \cdot e^{-i2\pi 3 \cdot \frac{2}{4}}\right] + \left[(-1 + 6i) \cdot e^{-i2\pi 3 \cdot \frac{3}{4}}\right] = 13 + 14i$$

Our transformed matrix would be $\boldsymbol{X}$**:**

$$\boldsymbol{X} = \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} 17 + 4i \\ -1 - 12i \\ -5 - 6i \\ 13 + 14i \end{pmatrix}$$

This was an example of the Discrete Fourier Transform converting a sequence of numbers from one form to the other. This would be a very important characteristic of the Discrete Cosine transform. Since converting an images' property to a quantitative handle will give us the ability to manipulate the image and compress it.

The Discrete Cosine Transform

Using our knowledge of the Fourier Transform we going to explore the mechanics for image compression. For the purpose of image compression, we are going to deal with two types of DCTs (Discrete Cosine Transform): A forward 2D DCT and an inverse 2D DCT. A 2D DCT means that the transform would act on a two-dimensional array, which means it can take in values from the x axis and the y axis – in simpler words a matrix of values with rows and columns. This 2D DCT in technical terms in called DCT-II and is represented by the equations:

Forward DCT-II:

Given an image $\boldsymbol{K}$, in the spatial domain, the pixel at coordinates (x, y), is denoted as $pixel(x, y)$. To transform $\boldsymbol{K}$ into an image in the frequency domain, $\boldsymbol{F}$ – for each coordinate, denoted as $F(u, v)$ – we can use the equation below[6]: (see 'Appendix' for further explanation)

$$F(u, v) = \frac{2}{N} C(u) C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x, y) \cos\left[\frac{\pi(2x + 1)u}{2N}\right] \cos\left[\frac{\pi(2y + 1)v}{2N}\right]$$

$$for\ u = 0, \dots, N - 1 \text{ and } v = 0, \dots, N - 1$$

$$Where\ N = 8\ and\ C(u) = \begin{pmatrix} \frac{1}{\sqrt{2}}\ for\ u = 0 \\ 1\ otherwise \end{pmatrix}\ and\ C(v) = \begin{pmatrix} \frac{1}{\sqrt{2}}\ for\ v = 0 \\ 1\ otherwise \end{pmatrix}$$

[6] "Inverse Discrete Cosine Transform." *2D Discrete Cosine Transform*, unix4lyfe.org/dct/.

Inverse DCT-II:

$$pixel(x,y) = \frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} C(u)C(v)F(u,v)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

$$Where\ N = 8\ for\ x = 0, \dots, N-1\ and\ y = 0, \dots, N-1$$

Why the Discrete Cosine Transform?

The DCT is a variant of the Fourier Transform, it is used in transformation for data compression. The DCT is an orthogonal transform. Meaning, it has a fixed set of fundamental – called basis – functions. The DCT uses these basis functions to map the image from its spatial domain into a frequency-based domain, can also be called the spectral domain. As stated earlier this makes the image's data numeric, therefore, available for manipulation for compression.

One of the biggest aspects in the DCT is the *cosine* while the DFT (Discrete Fourier Transform) is limited to computing with complex numbers, the DCT, since it only uses the *cosine*, can use real numbers – numbers which are usually easier to work with.

For the DCT we would only be using 8×8 blocks from the image – an image may have many blocks of pixels, they are dimensions. There are a few reasons why this number was chosen in particular.

- If patch sizes were larger, then it would be entirely possible that there are larger color gradients between different RGB (red, green, blue) pixels, since a larger patch would contain more of these. The final steps for the DCT involve averaging certain values from the matrix – more on this later – the finer differences would be missed by the process. This would end up making the final compression result worst.

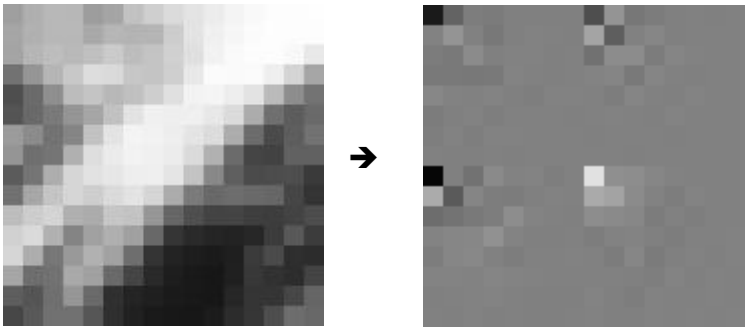
- There are other reasons too to choose 8×8 blocks instead of others. The DCT can be described in three easy steps – apply DCT on original matrix, apply mask, apply inverse DCT to get new compressed image matrix – this process is, however, not numerically efficient. It would be very arduous to perform if larger patches were taken, the time taken would be very large and the process would become very inefficient. Using smaller blocks would result in a loss of accuracy. It is therefore the standard in the industry to use 8×8 blocks from the image.

Since we are dealing with 2D matrices with rows and columns, it can be logically derived that the horizontally oriented basis functions represent horizontal frequencies while as vertically oriented basis function represent vertical frequencies.

When we apply the DCT to an image matrix the resulting matrix is a set of coefficients which can be classified into two categories: DC and AC. The DC coefficient is considerably larger in magnitude than the AC coefficients. The AC coefficients are higher in frequency but lower in magnitude, suggesting that high frequency image components play a relatively small role in determining image quality when a large portion of the image comes from lower frequency components – the DC with high magnitude and low frequency.

In a two-dimensional DCT matrix the DC coefficient of the horizontal frequency is to the left, and the DC coefficient of the vertical frequency is to the top. Therefore, in a DCT matrix the DC coefficient is present in the top left corner of the matrix. **The farther away an AC coefficient is from the DC coefficient, the higher frequency it's corresponding waveform will have and the smaller its magnitude will be**. The AC coefficients with smaller magnitudes are not significant in determining the image quality.

Below is an example of an image being converted from the spatial image domain to the frequency or the spectral domain, there are certain characteristics in this conversion which indicate that this conversion has undergone DCT:



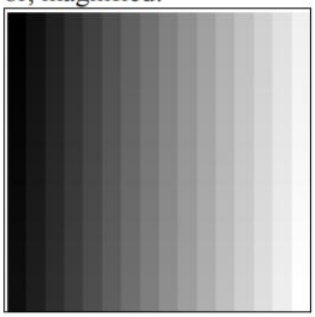
Original image (uncompressed) in spatial domain

Transformed image in frequency domain

The frequency domain shows a very interesting pattern of 8-bit (8×8) blocks. Also, in the top left corner of each block, the darkest block with the highest magnitude and the lowest number is the DC coefficient (i.e. lowest frequency of occurrence). Using this information, we can further explore how an image is corresponding to a matrix. At this point it is noteworthy to state how an image is represented as a matrix. For simplicity we are only going to deal with greyscale images – images composed of black and white. Every image in a computer is presented as a matrix. A matrix where each term is a pixel. In a greyscale image each matrix element is a number from the set $\{0, 1, 2, \ldots, 255\}$. The numeric values signify 0 as black pixels and 255 as white pixels.

We are not going to deal with colored – RGB based – images since the computer processes such images as three different images with each individual component: red; green; blue. Below is an example of an image represented as a matrix:

For example, the image  [1]
or, magnified:



is presented with the matrix

0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
1	17	33	49	65	81	97	113	129	145	161	177	193	209	225	241
2	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242
3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243
4	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244
5	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245
6	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246
7	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247
8	24	40	56	72	88	104	120	136	152	168	184	200	216	232	248
9	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249
10	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250
11	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251
12	28	44	60	76	92	108	124	140	156	172	188	204	220	236	252
13	29	45	61	77	93	109	125	141	157	173	189	205	221	237	253
14	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254
15	31	47	63	79	95	111	127	143	159	175	191	207	223	239	255

The image and the matrix above can be divided into 8-bit blocks and can be transformed using the DCT. When the matrix is processed through the DCT the range of values changes from $[0, 255] \rightarrow [-128, 127]$

Below is an input matrix for a greyscale image[7]:

$$\begin{pmatrix} 140 & 144 & 147 & 140 & 140 & 155 & 179 & 175 \\ 144 & 152 & 140 & 147 & 140 & 148 & 167 & 179 \\ 152 & 155 & 136 & 167 & 163 & 162 & 152 & 172 \\ 168 & 145 & 156 & 160 & 152 & 155 & 136 & 160 \\ 162 & 148 & 156 & 148 & 140 & 136 & 147 & 162 \\ 147 & 167 & 140 & 155 & 155 & 140 & 136 & 162 \\ 136 & 156 & 123 & 167 & 162 & 144 & 140 & 147 \\ 148 & 155 & 136 & 155 & 152 & 147 & 147 & 136 \end{pmatrix}$$

The output matrix from the DCT would be:

$$\begin{pmatrix} 186 & -18 & 15 & -9 & 23 & -9 & -14 & 19 \\ 21 & -34 & 26 & -9 & -11 & 11 & 14 & 7 \\ -10 & -24 & -2 & 6 & -18 & 3 & -20 & -1 \\ -8 & -5 & 14 & -15 & -8 & -3 & -3 & 8 \\ -3 & 10 & 8 & 1 & -11 & 18 & 18 & 15 \\ 4 & -2 & -18 & 8 & 8 & -4 & 1 & -7 \\ 9 & 1 & -3 & 4 & -1 & -7 & -1 & -2 \\ 0 & -8 & -2 & 2 & 1 & 4 & -6 & 0 \end{pmatrix}$$

It would be impractical to show calculations for each element from the matrix since each would require a long sum of 64 operations. For demonstration calculating a part of the DC coefficient is only logical. Due to the presence of two summations each calculation becomes very lengthy, though in the later part we will explore how this is implemented in code.

For $F(0,0)$:

$$= \frac{2}{8} \times \frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}} \times \left(\left(140 \times \cos\left[\frac{\pi\big((2\times 0)+1\big)\times 0}{16}\right] \times \left[\frac{\pi\big((2\times 0)+1\big)\times 0}{16}\right]\right) \right.$$

$$\left. \begin{aligned} &+ \left(140 \times \cos\left[\frac{\pi\big((2\times 0)+1\big)\times 0}{16}\right] \times \left[\frac{\pi\big((2\times 1)+1\big)\times 0}{16}\right]\right) + \cdots \\ &+ \left(144 \times \cos\left[\frac{\pi\big((2\times 0)+1\big)\times 0}{16}\right] \times \left[\frac{\pi\big((2\times 7)+1\big)\times 0}{16}\right]\right) + \cdots \end{aligned} \right\} y \text{ ranging from } 0 \text{ to } N-1$$

$$\left. \begin{aligned} &+ \left(pixel(1,y) \times \cos\left[\frac{\pi\big((2\times 1)+1\big)\times 0}{16}\right] \times \left[\frac{\pi(2y+1)\times 0}{16}\right]\right) \\ &+ \cdots + \left(pixel(7,y) \times \cos\left[\frac{\pi\big((2\times 7)+1\big)0}{16}\right] \times \left[\frac{\pi(2y+1)0}{16}\right]\right) \end{aligned} \right\} x \text{ ranging from } 0 \text{ to } N-1$$

The set of equations above would yield the DCT for one element of the matrix. It is very hard to include all the equations due to the nested summations. To grasp this concept using rows and columns, the matrix will take a column (x) and for every element in that column (vertical frequencies) (y) add them (summation). This process will then be done for every column and all the elements in it. The sum for each individual column will be added together to get the DCT for the matrix.

This is one of the most fundamental ways to implement the DCT with a two-dimensional matrix. This is a highly ineffective method due to the number of iterations the formula has to go through. Though there are other ways to implement this, they are out of the scope of this investigation.

[7] "Lossy Data Compression." *Lossy Data Compression: JPEG*, cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossy/jpeg/dct.htm.

As seen in the output matrix, the DC coefficient has the highest magnitude compared to the other coefficients. Applying DCT on an image matrix only sets the ground for the next step in which we will filter the AC coefficients which are of less importance and do not contribute significantly to the image quality. This is called the **quantization** phase.

Quantization

Quantization is the process of reducing the number of bits that are needed to store an integer in memory. This is done by reducing the coefficient's precision. Given a transformed DCT matrix with coefficients, we can reduce their precision as we move away from the DC coefficient. The farther away we are from the DC coefficient, the lesser that coefficient matters to the quality of the image and therefore we care less about maintaining its precision.

For quantizing values in a DCT matrix we would require a sample quantization matrix, these matrixes are mathematically computed and are readily available for use. A sample quantization matrix is shown below.

$$\begin{pmatrix} 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \\ 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 \\ 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 \\ 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 \\ 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 \\ 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \\ 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \\ 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \end{pmatrix}$$

For every element position in the DCT matrix there is a value in the quantization matrix. Each corresponding value shows a step size. As we move away from the DC coefficient the step size that is going to act on the DCT matrix coefficient increases. The elements that are most significant to the image are shifted by very little steps while as others – AC coefficients – are shifted by a large magnitude. The formula for achieving a **masked DCT matrix** for each value is shown below:

$$Quantized\ Value\ (x,y) = \frac{DCT(x,y)}{Quantum(x,y)}\ \boldsymbol{rounded\ to\ the\ nearest\ integer}$$

It can be deduced from the formula that high frequency AC coefficients will be divided by larger quantum values which will lead them to be rounded off to 0 (this is called masking). The more significant DC coefficient is divided by a small number therefore retaining its original precision. Since the AC values are zeroed out, in memory they will now occupy significantly less space. Below is an example[8]:

DCT matrix before quantization:

$$\begin{pmatrix} 92 & 3 & -9 & -7 & 3 & -1 & 0 & 2 \\ -39 & -58 & 12 & 17 & -2 & 2 & 4 & 2 \\ -84 & 62 & 1 & -18 & 3 & 4 & -5 & 5 \\ -52 & -36 & -10 & 14 & -10 & 4 & -2 & 0 \\ -86 & -40 & 49 & -7 & 17 & -6 & -2 & 5 \\ -62 & 65 & -12 & -2 & 3 & -8 & -2 & 0 \\ -17 & 14 & -36 & 17 & -11 & 3 & 3 & -1 \\ -54 & 32 & -9 & -9 & 22 & 0 & 1 & 3 \end{pmatrix}$$

DCT matrix after quantization:

$$\begin{pmatrix} 90 & 0 & -7 & 0 & 0 & 0 & 0 & 0 \\ -35 & -56 & 9 & 11 & 0 & 0 & 0 & 0 \\ -84 & 54 & 0 & -13 & 0 & 0 & 0 & 0 \\ -45 & -33 & 0 & 0 & 0 & 0 & 0 & 0 \\ -77 & -39 & 45 & 0 & 0 & 0 & 0 & 0 \\ -52 & 60 & 0 & 0 & 0 & 0 & 0 & 0 \\ -15 & 0 & -19 & 0 & 0 & 0 & 0 & 0 \\ -51 & 19 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

[8] "Lossy Data Compression : Coefficient Quantizaiton." *Lossy Data Compression: JPEG*, Sanford University, cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossy/jpeg/coeff.htm.

The process of quantization has starkly visible effects on the final matrix. The low-frequency high magnitude elements near the DC coefficient have been modified, but only by small amounts. The high-frequency low-magnitude elements have been, largely, reduced to zero. Therefore, eliminating their effect on the final image. In other words, the insignificant and unimportant data has been discarded form the image and the image has been compressed.

As stated previously, the quantization scheme or the template used can be achieved by numerous ways. The two most common ways are:

- Mathematically compute the error between the input image and its output image after decompressing it, that level of error can then be used to construct the steps (a brief overview of the process)

- Simply eye-ball it, the human eye may make certain errors but one can determine the steps to apply on each element in the two-dimensional DCT matrix – this process is purely subjective. Surprisingly, this method may be more credible in some cases than the mathematical error calculation method.

JPEG image compression allows one to use any quantization matrix, although the developers have conducted extensive research a standard set of quantization values have been established for different degrees of compression.

This final image then can be converted to a full all out image by shifting pixel values and identifying color based on element value. Though the details for the same are out of scope since a converter from compressed DCT matrix to image is a complete topic in itself.

Decompression

This section is extra since it is exploring the two-dimensional discrete cosine transform in reverse. The DCT is perfectly reversible and the image does not lose any definition until we require to manipulate the quantized coefficients. In the above matrixes the coefficients were simply zeroed out, a better approach would be to reduce their precision until an optimum value was achieved rather than zeroing them out. The equation for an Inverse Discrete Cosine Transform is included below:

Inverse DCT-II:

$$pixel(x,y) = \frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} C(u)C(v)F(u,v)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

$$Where\ N = 8\ for\ x = 0, \dots, N-1\ and\ y = 0, \dots, N-1$$

The specifications for the formula included above are the same, the only apparent difference is the position of $C(u)$ and $C(v)$. The definition of which still hold true for values of u and v:

$$Where\ N = 8\ and\ C(u) = \begin{pmatrix} \frac{1}{\sqrt{2}}\ for\ u = 0 \\ 1\ otherwise \end{pmatrix}\ and\ C(v) = \begin{pmatrix} \frac{1}{\sqrt{2}}\ for\ v = 0 \\ 1\ otherwise \end{pmatrix}$$

This method will decompress the image and bring it back to its original resolution.

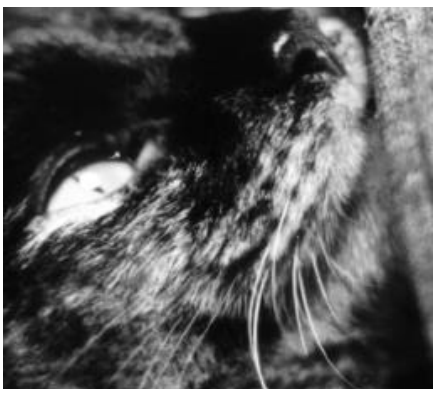
Implementation of 2D DCT-II in C++

Below is a code snippet which outlines the DCT process on an image matrix[9]:

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        temp = 0.0;
        for (x = 0; x < N; x++) {
            for (y = 0; y < N; y++) {
                temp += Cosines[x][i] *
                    Cosines[y][j] *
                    Pixel[x][y];
            }
        }
        temp *= 1/sqrt(2 * N) * Coefficient[i][j];
        DCT[i][j] = INT_ROUND(temp);
    }
}
```

There are 4 nested loops where each loop goes over a set of values. Every loop has a constraint of going through the values until the variable reaches N. This is the same process described above using the equation. There are two dimensional arrays being used to show a matrix and the output of the code would be a DCT matrix of an image matrix. This code helps me contextualize the process more aptly.

The code above reflects the real application of the DCT and displays it in context. Such algorithms are used by compression software to get compressed images. DCT is used in lossy JPEG compression which sacrifices a meniscal part of the image to achieve lower storage space.



Original image

Compressed image

An example of an original and a compressed image.

[9] "Lossy Data Compression." *Lossy Data Compression: JPEG*, cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossy/jpeg/dct.htm.

Conclusion

In conclusion, it can be said that there is a very distinct and noticeable relationship between the discrete Fourier Transform and its application in image processing. The transform takes an input of data which precisely draws parallels with real life image matrices. The two-dimensional discrete Fourier transform involves complex mathematical concepts which, I feel, could not be explained without prior conceptual knowledge. The Discrete Fourier Transform is an adept method at compressing images; however, I believe the true nature of the Discrete Fourier Transform is shrouded by a multitude of mathematical concepts. This makes comprehending the original process a very cumbersome task to say the least. For instance, throughout the exploration I had to comprehend the Fourier Transform to actually understand the Discrete Cosine Transform.

However adept the DFT process is, it lacks the mathematical elegancy and precision, this was evident through the 2D DFT-II matrix accepting any quantization matrix to create a compressed image. This poses avenues for ambiguity in the final output and gives way to subjectivity in mathematics. The DFT accepting multiplied by an arbitrary matrix makes it less mathematically precise. What I would like to further explore is how can a quantization matrix be developed mathematically and exploring its integration with the Discrete Fourier Transform formula. I would also be intrigued to explore the waveform and the visualizations for the images, comparing original image waveforms (on the xy plane) to compressed image waveforms and identifying peculiar characteristics between them.

This exploration improved my mathematical prowess and broadened my knowledge for complex mathematical models and their applications in the real world. It also provided me, yet another, avenue to connect computer science, a very specific part of it, to mathematics and explore their correlation in depth. After this exploration I now have a profound understanding of Sinusoidal waveforms, the Fourier Transform and the Discrete Cosine Transform and its applications in image processing.

Proofs

Proving Euler's Formula: $e^{-\varphi i} = \cos(\varphi) + \sin(\varphi)i$. The simplest way is to use the power series interpretation of $e^{-\varphi i}$ which uses Taylor Series to justify Euler's Formula. Proving the Taylor Series is well out of the scope of this exploration. Therefore, it stated below for reference:

The Taylor Series
Let $f(x)$ be a real-valued function that is infinitely differentiable at $x = x_0$. The series is an expansion for $f(x)$ centered around the point $x = x_0$ given by[10]:

$$\sum_{n=0}^{\infty} f^{(n)}(x_0)\frac{(x-x_0)^n}{n!}$$

** $f^{(n)}(x_0)$ represents the n^{th} derivative of $f(x)$ at $x = x_0$.

Using this we can derive Euler's Formula. We would first denote $e^{-\varphi i}$ in its power series notation and using that we would further derive the final equation suing the Maclaurin series which will help us in identifying the fundamental basis of the $\cos x + i \sin x$ formula.

[10] Taylor Series. *Brilliant.org*. Retrieved 07:26, April 5, 2019, from https://brilliant.org/wiki/taylor-series/

$$e^{ix} = 1 + ix + \frac{(ix)^2}{2!} + \frac{(ix)^3}{3!} + \frac{(ix)^4}{4!} + \frac{(ix)^5}{5!} + \frac{(ix)^6}{6!} + \frac{(ix)^7}{7!} + \frac{(ix)^8}{8!} + \cdots$$

$$= 1 + ix - \frac{x^2}{2!} - \frac{ix^3}{3!} + \frac{x^4}{4!} + \frac{ix^5}{5!} - \frac{x^6}{6!} - \frac{ix^7}{7!} + \frac{x^8}{8!} + \cdots$$

$$= \left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \frac{x^8}{8!} + \cdots\right) + i\left(x - \frac{ix^3}{3!} + \frac{ix^5}{5!} - \frac{ix^7}{7!} + \cdots\right)$$

$$= \cos x + i \sin x$$

This conveniently proves Euler's formula and its place in the Fourier Transform.

Appendix

Dot Product Notation and Meaning:

$$x[n] = [3, 5, 7, 9, 4]$$
$$b[n] = [8, 2, 5, 3, 1]$$

The two arrays above are example of sample signals. These signals are representing the $y - axis$ values of the functions of these signals. These are the 'already' computed values which once input in the Fourier Transform will be disintegrated into their constituent parts. That is what the Fourier Transform does, given a signal, it separates it from the time domain into the spectral domain, this will bring out the original signal's frequencies.

Discrete Cosine Transform Image Compression Formula:

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x,y) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

$for\ u = 0, \ldots, N-1$ and $v = 0, \ldots, N-1$

$$Where\ N = 8\ and\ C(u) = \begin{pmatrix} \frac{1}{\sqrt{2}}\ for\ u = 0 \\ 1\ otherwise \end{pmatrix}\ and\ C(v) = \begin{pmatrix} \frac{1}{\sqrt{2}}\ for\ v = 0 \\ 1\ otherwise \end{pmatrix}$$

Each component of the formula above is explained on page number 8 under the title the Discrete Cosine Transform. This is an explanation of the double sigma symbols and how they come together to form the formula seen above. Each sigma represents a variable in the $\cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right]$ equation. It's a nested function for summing all values of y for one value of x. For each value of x all values of y will be summed. These values of x and y will be input into the double-cos equation above and the sum would be generated based on that for each pixel.

Bibliography:

- Marshell, Dave. “The Discrete Cosine Transform.” *The Discrete Cosine Transform (DCT)*, 4 Oct. 2001, users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html.
- Vučković, Vesna. “IMAGE AND ITS MATRIX, MATRIX AND ITS IMAGE.” Faculty of Mathematics, Dec. 2008.
- Discrete Fourier Transform. Brilliant.org. Retrieved 20:55, March 26, 2019, from https://brilliant.org/wiki/discrete-fourier-transform/
- Raid, A.M., et al. “1405.6147.Pdf.” Mansoura University, Faculty of Computer Science and Information Systems, Apr. 2014.
- Marcus, Matt. “Marcus_proj.Pdf.” 1 June 2014.
- Khedr, Wael M., and Mohammed Abdelrazek. “Image Compression Using DCT upon Various Quantization.” International Journal of Computer Applications , 1 Mar. 2016.
- “An Interactive Guide To The Fourier Transform.” *BetterExplained*, betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/.
- Euler's Formula. *Brilliant.org*. Retrieved 07:25, April 5, 2019, from https://brilliant.org/wiki/eulers-formula/
- Taylor Series. *Brilliant.org*. Retrieved 08:46, April 5, 2019, from https://brilliant.org/wiki/taylor-series/
- Maclaurin Series. *Brilliant.org*. Retrieved 07:45, April 5, 2019, from https://brilliant.org/wiki/maclaurin-series/
- “In.” *unix4lyfe*, unix4lyfe, unix4lyfe.org/dct/in.png.
- “Out-2x2.” *unix4lyfe*, unix4lyfe, unix4lyfe.org/dct/out-2x2.png.
- “Chunk-In.” *unix4lyfe*, unix4lyfe, unix4lyfe.org/dct/chunk-in.png.
- “Chunk-Out.” *unix4lyfe*, unix4lyfe, unix4lyfe.org/dct/chunk-out.png.