Nikeet Pandit, Changin Oh
Harmonic Analysis and Image Processing

Image Compression: Final Report Part II

Table of Contents

Introduction

This is part two of the image compression report which discusses in-detail lossy compression. This report was written by Nikeet Pandit, except for the SVD compression section which was written by Changin Oh. All testing for this segment was done by Nikeet. Algorithms are appended at the end of the report, with proper accreditation for the code author.

Lossy Compression Theory

Discrete Cosine Transform and Discrete Fourier Transform

The DCT is closely related to the to the DFT. The DFT projects a signal onto an orthogonal cosine and sine basis, thus providing both amplitude and phase signal information, while the DCT projects onto a cosine basis only. As a result, the DCT offers amplitude information only provided by its correlation coefficients. Since the DCT represents the signal with only half the coefficients required by the DFT, the usefulness of DCT, when compared to the DFT, in terms of image compression is apparent. The effectiveness of both the DCT and DFT lies in the idea that these completely invertible transforms merely project an input signal from a spatial domain to a frequency domain. In fact, the frequency domain representation of the signal is a better representation to spectrally investigate image components which contribute minimally to the spatial reconstruction of the image, thus providing a meaningful basis to truncate information which is deemed less useful so the image may be effectively compressed.

The formula for the forward DCT defined for X(m) as a sequence of data points, m = 0, 1,…, (M-1) is given by the following equation [1]

$$G_x(0) = \frac{\sqrt{2}}{M}\sum_{m=0}^{M-1} X(m)$$

$$G_x(k) = \frac{2}{M}\sum_{m=0}^{M-1} X(m)\, cos\frac{(2m+1)k\pi}{2M}, k = 1, 2 \dots, (M-1)$$

where $G_x(k)$ is the k[th] DCT coefficient.

The formula for the forward DFT defined for X(m) is given by [2] in the following equation

$$X_x(k) = \sum_{m=0}^{M-1} X(m)\left\{cos\frac{2\pi km}{M} - i * sin\frac{2\pi km}{M}\right\}, k = 1, 2 \dots, (M-1)$$

Since both DFT and DCT are separable transforms, only the 1D transforms are provided and the 2D derivation is not explicitly required.

Singular Value Decomposition (SVD)

Let $A \in M_{mn}(\mathbb{R})$ with $m > n$. Then the Singular Value Decomposition states that A can be expressed as

$$A = U\Sigma V^T$$

where $U \in M_{mm}(\mathbb{R})$, $V \in M_{nn}(\mathbb{R})$ and $\Sigma \in M_{mn}(\mathbb{R}^+)$. Note that U and V are orthonormal matrices and Σ consists of nonnegative singular values as its diagonal elements and zero elsewhere.

From this definition, A can be written as the sum of outer products using the SVD as below:

$$A = \sum_{i=1}^{n} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$$

where σ_i are singular values of A with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$, and $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ are the orthonormal columns of U and V, respectively. Each term $\sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$ is called a rank 1 matrix and hence we can say that A can be expressed as the sum of rank 1 matrices.

Suppose that we add k rank 1 matrices and denote the result as $\hat{A}$. That is,

$$\hat{A} = \sum_{i=1}^{k} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T \quad (k < n)$$

Then $\hat{A}$ becomes the rank k approximation to A and according to the Eckart-Young theorem, the SVD guarantees that $\hat{A}$ is the best rank k approximation to A. Notice that $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ have unit length and thus σ_i scales them. In another way, the SVD can be considered of a matrix A into a decomposition into the multiplication of a rotation matrices and scaling matrix.

Relating the SVD algorithm to image compression, we decompose an image using SVD and then approximate it by adding the first few terms which contain the significant information of the image. The more terms we add, the better quality the approximate image will obtain. Small ranked SVD approximations are preferable since the amount of storage space increases as the approximate image quality increases. Generally, if the cumulative energy of the first few singular values is over 80%, the corresponding number of singular values is considered adequate approximation of the original signal.

Discrete Wavelet Transform

Wavelet analysis is a signal processing technique which allows for MRA (multi resolution analysis). MRA provides frequency information across various time scales, thus providing time and frequency localization simultaneously across varying bandwidths. More generally, wavelet analysis allows for the characterization of non-stationary signals in the time-frequency domain. This is because wavelets, through shifting and scaling of a mother wavelet function, allow for multi resolution decomposition with spatial localization. Therefore, wavelets represent a signal in a frequency-space domain. This is opposed to FFT and DCT, which represent in the frequency domain only and do not provide any spatial localization of spectral constituents. Based on this knowledge, compression of images using wavelets has definite advantages over the FFT and the DCT as demonstrated by the driving idea that it is possible to truncate high-frequency coefficients except with small, localized areas of high frequency constituents. Based on this theory, one can expect DWT compression to preserve edge detail even with many truncated coefficients.

A rigid mathematical treatment for wavelet theory and its requirements for MRA can be found in [3]. In this document, the algorithm for the FWT (Fast Wavelet Transform) will be discussed, which employs the

DWT through a recursive cascading bank of high pass and low pass filters [4]. In the 1D case, one level of discrete wavelet transform decomposition can be written compactly where the original input signal, x[n], is simultaneously convolved with high and low pass kernels as described below

$$x_{low} = x[n] * g[n], n = 0, 2, \ldots$$

$$x_{high} = x[n] * h[n], n = 0, 2, \ldots$$

where * refers to convolution, g refers to a low pass filter kernel, h refers to a high pass filter kernel, and convolution is evaluated every second instant as to downsample the filtered input signal by 2, an operation which is valid for perfect reconstruction when synthesizing the decomposition based on the Nyquist rule. It is important to note that that the above filter kernels are quadrature mirror filters, which also necessary to ensure perfect reconstruction after the separation into high and low frequency components after upsampling by a factor of 2. The low frequency components may be regarded as approximation coefficients while the high frequency components are then regarded as detailed coefficients. One level of decomposition using filter banks is shown below. Further levels of decomposition are performed by simultaneous low pass and high pass filtering of the approximation coefficients with half the cut of frequency from the previous level. In the next diagram taken from [5], a three-level DWT multistage filter bank is used with the associated frequency splitting characteristics.

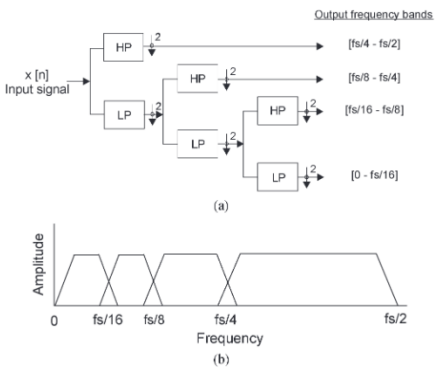

Figure 1

As we can see, high level decomposition constituent the lower frequency components while low level decomposition corresponds to the higher frequency components. In the above diagram, to achieve more frequency resolution, the level 1 high pass filter may also be separated into its detail and approximation components. The 1D transform can easily be extended to the 2D case and is treated in [5]. In fact, the 1D FWT as described above, is simply taken on the columns of the input image, followed by the 1D FWT on the resulting rows. Each level of decomposition in the 2D case will give approximation coefficients, and three detail coefficients corresponding to horizontal coefficients, vertical coefficients, and diagonal coefficients. These correspond to high frequency variations in the respective directions.

Block Compression

Block compression occurs when an input image sequentially blocked left to right, top to bottom. Stated in another way, an input image is windowed with non-overlapping windows, and some sort of compressing operation is performed for each block. After each block has had an operation performed onto it, the image may be reconstructed from the operated-on blocks. This is in essence the how the JPEG sequential compression is performed, where an input image is blocked by an 8x8 window size [6]. For each window, the DCT is performed, and a deconvolution step is performed to truncate spectrally insignificant information in the window. The spectrally insignificant information is usually the higher frequency components which contribute less variance than the low frequency components of the image.

Lossy Compression Algorithm Methodology

Data Types

As important as the image compression algorithm itself, is the data type that the image must be stored and compressed in. The test image used in this report is a gray scale image represented by uint8 (unsigned 8-

bit integers). If for example, after the compression, the image must be stored using double; to store the image will require at least 24 times the number of bytes relative to uint8 precision. This is due to the amount of precision both numeric data types can hold.

DCT Compression and Decompression Algorithm

For DCT compression, first the FDCT (forward discrete cosine transform) is taken for the input image to derive the amplitude DCT coefficients of the input image. As described in the theory of the DCT and DFT, most of the power of the image is contained in the low frequency components. The DCT coefficients are then divided by a weighting kernel as described in the formula below, where F^C refers to the compressed coefficients, F refers to the DCT coefficients, Q refers to quality, and K refers to the weighting function.

$$F^C(u,v) = round\left(\frac{F(u,v)}{K(u,v) \times Q}\right)$$

This formula corresponds to frequency domain and is equivalent to deconvolution in the time domain. Related to image compression, the division step acts to truncates the visually insignificant coefficients by weighting less or complete truncation, depending on the coefficient magnitude and its associate weight. The visually insignificant coefficients refer to higher frequency constituents, as they contribute less to the image variation. This step can be reversed with knowledge of the filter kernel, except due to the integer rounding procedure and when coefficients have been truncated (or floored). The compressed image is saved as an uint8 whose values corresponded to the compressed DCT coefficients of the input image. As a result of this weighting process, many of the coefficients will be floored to zero. Since many of the coefficients are zero, Huffman and RLE encoding can achieve significant lossless compression when the file is saved. To decompress the image and recover the lossy compressed image in the spatial domain, the kernel must be multiplied by the coefficients and the IDCT must be taken thereafter. The weighting kernel for all tests is taken from JPEG's standard grayscale kernel, which was determined experimentally in 1984. However, in the compression algorithm any kernel may be specified. More details, as well as the kernel structure, which was derived experimentally, and does not weight equally or smoothly in the vertical and horizontal direction can be found here [7]. Since this kernel is given defined for a size of 8x8, the kernel is simply interpolated via cubic interpolation to the size of the input image, so the weighting remains valid. The Q value refers to the quality and allows for a parameterizable input to trade image compression for image quality and vice versa. A higher Q represents more compression.

DCT Block Compression Algorithm

Block compression algorithms is only applied for the DCT. The methodology follows exactly the procedure as above, except the DCT compression and DCT decompression algorithm is applied to individual blocks of the input image sequentially. The JPEG weighting kernel is simply interpolated to the selected block size. In the case of another weighting kernel, it is simply constructed to the block size. Theoretically, this is a very similar operation to the Gabor transform and acts to preserve localized high frequency constituents in the input image even after the weighting procedure, which would otherwise not be preserved without the blocking. Therefore, we expect more edge detail for block compression using the DCT as opposed to DCT compression applied to the whole image. Attempts were made to do DFT block compression, however, since the DFT represents the signal in both magnitude and phase, the image could not be stored using uint8, as well as the number of coefficients required are double of the DFT. Therefore, experiments indicated this to be a very poor compression algorithm.

DFT Compression Algorithm

Since the DFT represents a signal with both magnitude and phase, the image cannot be saved in its spectral representation like the DCT and be a suitable compression algorithm. However, compression can be realized using the DFT through frequency domain filtering. In the frequency domain, low pass filters can be applied to smooth the image and the IDFT (inverse discrete Fourier transform) can be applied to reconstruct the smoothed image. The image is then stored as a uint8 type. Due to the smoothing effect of the image, when compared to the original image, it results in a smaller variation in the distribution of the pixels in the image matrix. Due to the smaller variation in the distribution, the Huffman and RLE lossless compression algorithms, after the image is saved and stored becomes, more effective because there is more repetition in the intensity levels describing the image.

In one version of the DFT compression algorithm, the user may select the percentage of coefficients to keep. The absolute value of the DFT coefficients is then taken and placed in descending order. The coefficients whose magnitude are greater than this threshold is preserved, while the rest are truncated. The coefficients are placed into the original order with some of the coefficients having been truncated as indicated in the prior step, and the IDFT of the lossy image. This will be referred to *ideal FFT compression algorithm.*

In another version of the DFT compression algorithm, simply the cut-off frequency is selected for frequency domain filtering and the kernel is selected. Gaussian kernels demonstrated the best performance, when compared to ideal frequency domain filtering, and has tests appended to this report, likely due its smooth kernel which prevents rippling. This method of compression is slightly more involved, as it requires some experimentation to determine the best cut off frequency and also requires inspection of the original image input spectrum. This will be referred to the *smooth FFT compression algorithm.*

SVD Compression Algorithm

For SVD compression algorithm, it consists of simply taking the SVD of the baseline image. Thereafter, the number of approximating terms is selected to trade off between quality and approximation. For SVD compression, there will be two methods of testing: (1) Approximating Testing; (2) Decomposition Testing. For approximating testing, the outer product of the decomposition matrices will be taken. After, the approximation image will be converted to uint8 precision and subsequently saved. For decomposition testing, the memory requirements (in MATLAB's memory), will be compared with the memory requirements for the sum of the three decomposition matrices against the total initial image, both of which are of type double.

As will be seen, outer product multiplication of the three decomposition matrices does very minimally in terms of image compression. This is because it does not smooth the image well, especially when compared to frequency domain filtering. As a result, the Huffman and RLE lossless compression algorithms when the approximated image is saved can not exploit very much the repition of numbers in the compressed image matrix, therefore compression savings are minimal with respect to the original uncompressed saved image.

Where SVD compression excels and differentiates itself with respect to the other compression algorithms, lies in the fact that three approximating decomposition matrices have less elements in total than the total elements required to represent the image in full matrix form. The difference between the two depends on the amount of approximating terms of the SVD. This yields great potential for compression.

DWT Compression Algorithm

For the DWT compression algorithm, similar to the ideal FFT compression algorithm, the user may select the percentage of coefficients to keep. The absolute value of the DFT coefficients is then taken and placed

in descending order. The coefficients whose magnitude are greater than this threshold is preserved, while the rest are truncated. Wavelet decomposition is specified according to the level of decomposition desired, as well as the mother wavelet function. For example, if a Haar wavelet function is selected, this corresponds to ideal high and low pass frequency domain filters as described by the FWT above. After, the coefficients are reordered to their original assembly, with some having been truncated, and the inverse transform is taken to reconstruct the image in the spatial domain. The Haar wavelet function and 16 levels of decomposition were used for all testing appended. Both the Haar and 16 levels of decomposition proved to provide the best quality compression characteristics. The Haar wavelet potentially did very well as opposed to all others due to its ideal time domain characteristics. 16 levels of decomposition perhaps did the best due to the incredible segmentation of the signal into its frequency spatial characteristics where spectrally insignificant components with spatially localization could easily be truncated without excessive quality distortion to the image.

Results and Discussion

Testing Procedure

Figure 2

A 512x512 rose image will be tested. The image was originally a .TIFF file, a file type with no compression used. The image was then saved to a .JPEG, using 100% quality compression. This means that in effect the weighting matrix in the frequency domain division step, as outlined in the DCT compression algorithm section, will have virtually no effect (except due to rounding) to the DCT coefficients and therefore is approximately an invertible transformation. However, the algorithm will compress the image using Huffman and RLE lossless compression algorithms. This image will serve as the baseline image. All other images when compressed will also be saved to .JPEG with 100% quality as to provide one-to-one comparisons without inadvertently adding additional compressions. Further, the metadata of all the images is completely stripped to not add any extra information embedded to the image, which would also distort compression metric comparisons. The JPEG 100% quality image based on the original .TIFF image is provided here. All software is appended at the end of the report.

Quality Metrics and Visual Preference

Due to the sheer number of testing performed for the various lossy compression algorithms, first the tabular results of all the testing done will be appended. In the next section, discussion relating the tabular metrics and comparing the various compression algorithms supported with image examples will be provided. Based on testing, the metrics correspond quite well to what we see visually. Therefore, despite being a report on image compression, it is intuitive to focus on tabular metric results to summarize the results. In terms of quality metrics and when comparing to visual preference, a MSE of at least 20 should be achieved for reasonable reconstruction. An SSI of at least 0.9 and PSNR of about 35 yields visually reasonably good quality results. The SR (Sobolev) error, as indicated in [8], does in fact provide a metric for the resolvability of high frequency components. One can expect an SR of more than 25 to correspond to noticeable lacking edge detail, and a value less than this to be reasonably resolved. The TVE on the other hand due to its sheer magnitude for these compression algorithms across the board makes it slightly more complicating to deduce any specific notes, other than higher value corresponds to more error.

Metric Results for Lossy Compression

Ideal FFT Compression

Percentage of Coefficients to Keep	Elapsed time	Comp Ratio	Save %	Bit Rate	MSE	SSI	PSNR	SR	TVE
5.00	0.02	1.55	35.68	0.35	36.64	0.84	32.49	35.09	4230564.67
4.25	0.02	1.59	36.99	0.34	40.94	0.83	32.01	36.00	4316097.35
3.50	0.02	1.63	38.76	0.33	46.66	0.82	31.44	37.45	4429943.88
2.75	0.02	1.67	40.24	0.32	54.73	0.80	30.75	39.17	4569094.17
2.00	0.02	1.75	42.83	0.31	67.08	0.79	29.87	41.65	4691836.00
1.25	0.02	1.88	46.85	0.29	89.16	0.77	28.63	45.07	4824965.75
0.50	0.02	2.08	51.93	0.26	151.62	0.73	26.32	51.82	5048127.45

This method is described in the DFT compression algorithm to ideally truncate spectrally insignificant power of the input image.

Smooth FFT Compression

Cut Off frequency	Elapsed time	Comp Ratio	Save %	Bit Rate	MSE	SSI	PSNR	SR	TVE
70.00	0.08	3.33	69.95	0.16	68.24	0.86	29.79	43.04	4371572.99
90.00	0.09	2.86	65.04	0.19	49.61	0.88	31.18	39.34	4062037.43
110.00	0.08	2.50	59.98	0.22	38.03	0.89	32.33	36.58	3775085.66
130.00	0.09	2.21	54.84	0.24	30.19	0.90	33.33	34.36	3506508.47
150.00	0.08	2.01	50.23	0.27	24.55	0.91	34.23	32.51	3248306.83
170.00	0.08	1.86	46.27	0.29	20.30	0.92	35.06	30.83	3005613.00
190.00	0.10	1.73	42.33	0.31	16.97	0.93	35.84	29.35	2781569.58

This method is described in the DFT compression algorithm, where a Gaussian kernel is used to smooth high frequency components to reduce variability of pixel intensities.

SVD Compression - Approximating Matrix

Rank	Elapsed time	Comp Ratio	Save %	Bit Rate	MSE	SSI	PSNR	SR	TVE
10.00	0.03	1.99	49.79	0.27	177.36	0.78	25.64	57.11	4828138.27
42.00	0.04	1.49	32.84	0.36	29.41	0.88	33.45	33.88	3463757.02
73.00	0.04	1.30	23.31	0.41	13.33	0.92	36.88	26.29	2715866.42
105.00	0.04	1.19	16.28	0.45	7.53	0.94	39.37	21.77	2142904.91
137.00	0.04	1.13	11.39	0.48	4.70	0.96	41.41	18.68	1734205.40
168.00	0.04	1.09	8.14	0.50	3.09	0.97	43.24	16.62	1444311.86
200.00	0.04	1.06	5.38	0.51	2.06	0.97	44.99	14.92	1203769.48

This method is described in the SVD compression algorithm and corresponds to the case where the outer product of the SVD decomposing matrices is taken and the resultant approximation image reconstruction is written out to storage with uint8 precision.

SVD Compression - Decomposing Matrices

Rank	Elapsed time	Comp Ratio	Save %	Bit Rate	MSE	SSI	PSNR	SR	TVE
10.00	0.05	25.35	96.06	0.32	181.09	0.14	-22.58	57.98	4809858.59
42.00	0.04	5.86	82.92	1.37	29.49	0.31	-14.70	34.27	3441973.65
73.00	0.04	3.27	69.45	2.44	13.09	0.42	-11.17	26.44	2676823.64
105.00	0.04	2.21	54.78	3.62	7.25	0.52	-8.60	21.80	2089767.01
137.00	0.04	1.65	39.32	4.85	4.43	0.61	-6.46	18.69	1666617.69
168.00	0.05	1.31	23.61	6.11	2.83	0.68	-4.51	16.47	1368286.38
200.00	0.04	1.07	6.62	7.47	1.81	0.75	-2.59	14.66	1109895.80

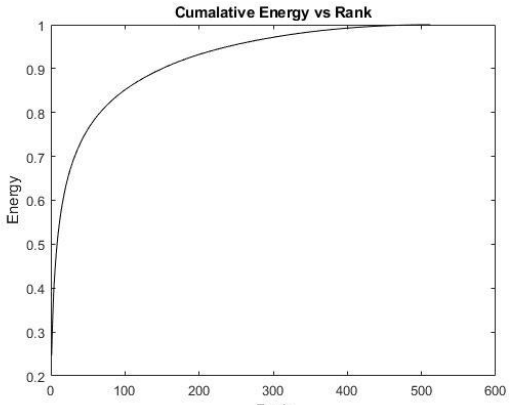

Figure 3

This method is described in the SVD compression algorithm and corresponds to the case where the SVD decomposing matrices is taken and metrics are computed by adding bytes required to store in MATLAB's memory the three decomposed matrices in double precision when compared to the original image, also in double precision. It must be noted the irregularities in the PSNR. No explanation can be given for this.

For SVD, most of the energy is stored in the first few approximating terms. This is visualized in the cumulative energy graph, corresponding to the rose image. The first ~100 ranks correspond to 85% of the image energy.

DWT Compression

Percentage of Coefficients to Keep	Elapsed time	Comp Ratio	Save %	Bit Rate	MSE	SSI	PSNR	SR	TVE
10.00	0.06	2.27	55.97	0.24	5.64	0.94	40.62	18.32	2429793.74
7.00	0.07	2.49	59.83	0.22	7.98	0.93	39.11	20.45	2719402.11
4.00	0.06	2.94	65.93	0.18	13.01	0.91	36.99	24.61	3127560.18
1.00	0.06	5.75	82.61	0.09	42.78	0.86	31.82	37.25	4273776.48
0.70	0.07	7.21	86.12	0.07	58.28	0.84	30.48	40.75	4576313.20
0.50	0.08	9.44	89.40	0.06	76.44	0.83	29.30	44.40	4876382.39
0.20	0.07	19.81	94.95	0.03	157.46	0.79	26.16	55.45	5644409.36

This method is described in the DWT compression algorithm, where wavelet decomposition is applied to the input image and coefficients with minimal variance, corresponding to the number of coefficients to keep, are completely truncated.

DCT Compression

Block Size	Elapsed time	Comp Ratio	Save %	Bit Rate	MSE	SSI	PSNR	SR	TVE
512.00	0.04	3.42	70.75	0.16	18.67	0.87	35.42	25.86	3571801.42

This method corresponds to the DCT compression algorithm. It can also be viewed as a special case of the block DCT compression, where the block size is simply the size of the input image (512x512).

DCT Block Compression

Block	Elapsed time	Comp Ratio	Save %	Bit Rate	MSE	SSI	PSNR	SR	TVE
8.00	0.14	1.12	11.04	0.48	13.76	0.91	36.74	24.28	2974588.37
16.00	0.09	2.02	50.41	0.27	13.87	0.91	36.71	24.26	3004574.66
32.00	0.03	3.07	67.40	0.18	14.36	0.91	36.56	24.55	3023411.03
64.00	0.02	3.54	71.73	0.15	14.96	0.90	36.38	24.97	3110616.66
128.00	0.02	3.73	73.22	0.14	16.13	0.90	36.05	25.30	3250510.23
256.00	0.03	3.66	72.68	0.15	18.62	0.87	35.43	25.83	3534891.67

This method corresponds to the block DCT compression algorithm. Different block sizes were used. The default JPEG grayscale 8x8 weighting kernel was interpolated using cubic interpolation to the required block size specified, and DCT block compression was implemented thereafter. This method closely resembles the JPEG sequential codec, without the zig-zagging operation before the lossless encoding.

Quality Trade Off Compression

Quality Factor	Elapsed time	Comp Ratio	Save %	Bit Rate	MSE	SSI	PSNR	SR	TVE
1.00	0.07	2.02	50.41	0.27	13.87	0.91	36.71	24.26	3004574.66
4.00	0.06	3.83	73.86	0.14	27.87	0.88	33.68	31.16	3814726.07
7.00	0.07	5.10	80.40	0.11	40.81	0.85	32.02	35.38	4184881.61
10.00	0.07	5.84	82.89	0.09	56.84	0.83	30.58	40.50	4477569.46
13.00	0.10	6.72	85.11	0.08	72.62	0.79	29.52	44.26	4726433.38
16.00	0.08	7.35	86.39	0.07	89.35	0.72	28.62	49.30	4872330.79
19.00	0.06	7.89	87.33	0.07	110.92	0.64	27.68	52.65	4998143.80

For a block size of 16, different quality factors were selected. The JPEG grayscale kernel is multiplied by the quality factor as indicated in quality 1. A higher quality factor results in a larger magnitude weighting kernel. Therefore, in the frequency domain division step, more DCT amplitude coefficients will be floored to zero. This is indicative in higher MSE and lower SSI with higher quality factors. Essentially, the larger the quality factor, the lesser the quality of image and the greater the compression.

Metric Discussion for Lossy Compression

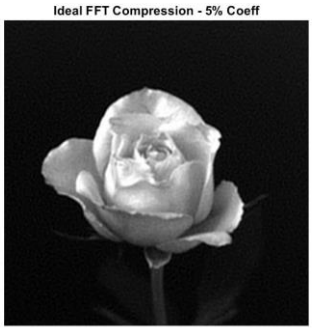

Figure 4

For the ideal FFT compression, we can make a few key observations. First, we see that lower value for the percentage of coefficients to keep, the greater the compression at the cost of degrading image quality. This is demonstrated by all quality and compression metrics. Further, this also demonstrates earlier points made in the DFT and DCT theory, that most of the power of an image is constituted in a few coefficients, usually the low frequency and DC components. This is shown by the fact that only 5% of the all the coefficients required to perfectly reconstruct the image has a reasonable approximation of the image. In the black foreground of the image, as seen in figure 4, we see an apparent ringing effect indicative by the variational intensity that is not there in the original image. This may be attributed to the ideal filter truncation of low power frequency constituents. When comparing the ideal FFT compression to the smooth FFT compression, we note several key points. First, the smooth FFT compression for any set of given compression statistics (compression ratio, save percentage, bit rate) will yield better quality metrics (MSE, SSI, PSNR, and TVE). As expected, as the cut off frequency of the smooth FFT compression kernel becomes larger, the SR error decreases. This is indicative of the fact that a larger cut off frequency selection results in less attenuation of the higher frequency components. For similar

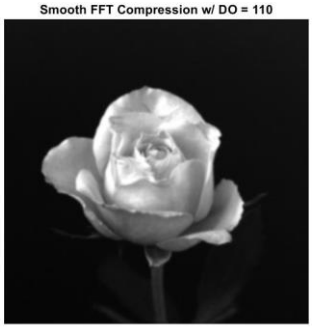

Figure 5

Figure 6

quality metrics, the smooth FFT compression yields about 15% greater saving percentage as shown visually in figure 4 and figure 5. The reason for better compression for the FFT compression lies in the smooth kernel function which does not introduce any rippling in the black background. As a result, there is lots of numerical intensity repition by the smoothing of high frequency components, when compared to the original image and the ideal FFT compression example. Therefore, in the smooth FFT compression image matrix, there is lots of potential for compression from the subsequent lossless compression. When comparing ideal FFT compression and Smooth FFT compression for the same compression statistics, smooth FFT compression will also preserve more edge detail and is a significant improvement over the ideal FFT compression. This is shown when comparing image 4 to image 6.

First we will look at SVD approximating compression case, where we approximate the matrix by taking the outer product of the three decomposing matrices and save this image and compare it to the baseline. We note from the table, that as more approximating terms are added, it results in an increase of image quality at the cost of image compression. However, we see the performance of both compression and preserving image quality is extremely poor when compared to the previous to DFT methods. For example, including 10 approximating terms has a compression ratio of 1.99, compared to figure 5 with a compression ratio of 2.5. Despite having less compression for the SVD case, it has significantly worse quality metrics across the board. This is visually apparent in figure 7.

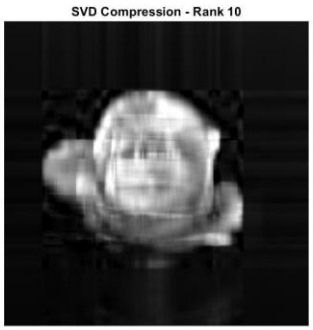

Figure 7

Where the SVD compression excels, however, is in being able to decompose the matrix into three simpler matrices that in total, before the outer product is taken to reconstruct the image, require significantly less elements in its decomposed state than then the total elements the original image requires. For this compression statistics, the amount of memory MATLAB requires to hold the image is compared for the baseline image and the sum of the three approximating images, before the outer product is taken (because otherwise it is the same). Double precision is used for both. Testing SVD compression this way in the decomposing case, figure 7 with 10 rank has a CR of 25.35, compared to the approximating case of a CR of only 1.99. We see, for example, including 73 approximating terms has a CR of 3.27 and MSE of 29.49. However, the SSI is unusually small (when compared to the other compression algorithms) for this given MSE, sitting at 0.42. No explanation can be given for this. Figure 8 shows the outer product multiplication of the decomposing terms with these compression and quality characteristics. We see the image is very faithfully reconstructed. When looking at image 3, which describes the cumulative energy distribution, it is apparent that the first 73 terms correspond to about 80% of the image energy.

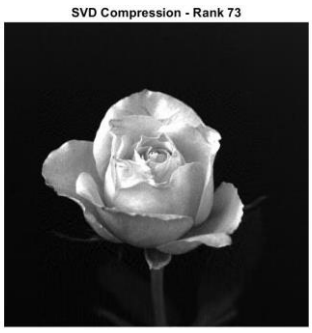

Figure 8

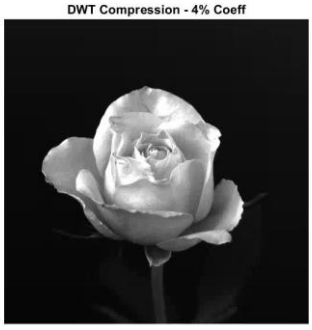

Figure 9

Wavelet compression, as will be demonstrated, provides the absolute best compression and quality characteristics in its reconstructed spatial state. However, it is comparable to the SVD compression quality and compression characteristics, when considering its considering decomposition SVD compression testing. Although before the outer product is taken, the image cannot actually be visualized. Since the wavelet compressed image can be represented by uint8, however, wavelet decomposition is far super in terms of the compression quality trade off and is the best algorithm.

Figure 10

Figure 11

Keeping 4% of significant coefficients, which corresponds to compression ratio of 2.94, which is a 6% greater saving percentage compared to figure 5 (smooth FFT compression), and a 30% increasing in saving percentage compared to figure 4 (ideal FFT compression), we see excellent characteristics both visually and in its metrics, provided by a low MSE, high SSI, high PSNR, and good SR. This wavelet compression example in shown in figure 9. Pushing wavelet compression to the extreme, for a compression ratio of 9.44 and including only 0.5% coefficients, we see the effect of the compression on all the quality metrics as well as visually. This is shown in 10. Comparing SVD for rank 25, which has about ~9.5 compression ratio, about the same as the previous wavelet example, the MSE is 15 points lower (60 compared 75, respectively). However, the SSI for wavelet compression using 0.5% compression is much higher at 0.83, compared to the SVD rank 25 SSI result of 0.23. Our preference is more in accordance with the wavelet decomposition, with the higher SSI despite the lower MSE. The rank 25 example is shown in figure 11. As expected based on the theory of wavelets, the edge detail is especially well preserved in figure 10 even with high compression.

Looking at the block DCT compression, where simple DCT compression can be viewed as a realization of block DCT compression with a block size equal to the input image, we notice there is an optimal block size where the quality compression tradeoff is maximized. Investigating the compression ratio, the MSE, and the SSI, as shown in figure 12, we note for the CR, that increasing block sizes initially yields great compression where it reaches it minimum value and starts to increase. At this point of increase, where the block size is 256x256, the SSI is minimum as well as the MSE is maximum. The MSE increases until this 256x256 block size and then levels out. The SSI remains relatively constant with slight reduction in value past 256x256 as well. With this in mind, it seems that the optimal block size is 32x32 which provides great compression

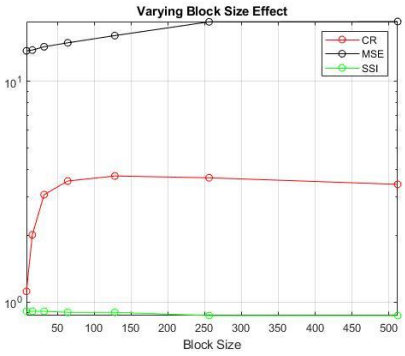

Figure 12

Figure 13

Figure 14

characteristics of 3.54 at a low MSE of 14.36 and high SSI of 0.91. Also, the SR is at a good level of 24.55. The optimal block size DCT compression is shown in figure 13, which is a significant improvement over both of the FFT compression methods. For this given compression ratio, it actually performs very similar to wavelet compression in terms of the compression quality trade-off, however, when reducing the quality of decompression of the DCT by parameterizing the quality factor (i.e., to achieve further image compression at cost of quality), as will be seen, the wavelet decompression fairs much better in terms of preserving the quality of the image for the same level of compression.

When using a block size of 512x512, the SR, which denotes edge detail error, is higher than that of when a block of 8x8 is used. This is confirmed visually as well. This corresponds well to theory described in the DCT compression and our comparison of DCT block compression to the Gabor transform. Based on all testing, to yield better image quality at the cost of image compression, it is then recommended to use a Q (quality factor) of less than 1 to allow for the optimal quality compression trade off based on the user's needs with a block size of 32. In fact, for a compression ratio of less than 3 wavelet compression and DCT block compression of size 32 yield comparable results, except DCT compression executes in half the time of wavelet compression.

In the next highlighted example, the quality factor is adjusted to 10, and a block size of 16 as opposed to 32 which was indicated to be the optimal size is used. This is shown in figure 14. This is because when using a quality factor greater than 1 (reducing in quality and more compression), and a block size of greater than 16 is used, there is a significant blocking artefact that arises in the compressed image that is very undesirable. Compared to wavelet compression, at a similar compression ratio, the quality metrics noted by MSE, SSI, PSNR and SR all show better characteristics. Visually, it also a better result. Therefore, these tests indicate the for a compression ratio past approximately 3, wavelet decomposition will yield better quality characteristics, when compared to DCT block-based compression with the parameterizable quality factor.

Conclusion

In summary, we provided a discussion for the DCT, DFT, SVD, DWT, and block-based compression. Thereafter, we discussed the algorithm methodology, which use these transforms, and described how the software is used to compress an input image. Since all of these transforms are invertible, these transformations do not provide any lossy compression in itself. Then, we outlined the testing procedure and provided the image baseline to be compared to in terms of quality and compression, when testing the different compression algorithms. A discussion which related the metric results to visual preference was also highlighted. Results for ideal FFT compression, smooth FFT compression, SVD compression for both the approximating and decomposing matrix cases, DWT compression, DCT compression, and DCT block-based decompression were provided, included with quality trade-off parametrization.

It was seen that the smooth FFT compression fared better than ideal FFT compression in all instants, however, these two compression algorithms provided the worst quality compression characteristics of all the algorithms tested. SVD compression, when using the approximated matrix testing scheme and saving this resultant image, did little in terms of compressing the image because the approximated image, even of lower ranks, was not very smooth, especially relative to the smooth FFT compression algorithm. Therefore, minimal compression could be realized by the lossless compression algorithm when the image is saved even high distortion to the image quality.

SVD compression, when considering the decomposition matrices before outer product multiplication, however, provided a unique method for compression in that the total elements of the decomposing matrices,

depending on the number of approximating terms, is significantly less than the total elements required to store the input image in its nominal matrix form. Therefore, when double precision is compared for both the baseline image and its decomposed form, SVD compression when considering its decomposed three matrices provides fantastic compression quality characteristics and is similar to wavelet decomposition metrics. An optimal block size for DCT compression was determined to be 32x32, which maximized both quality and compression. When the compression ratio is less than 3, it was determined that the block DCT with a 32x32 block size, using the parametrizable quality factor, provided the best quality compression characteristics at the fastest computational efficiency. At a compression factor of more than 3, DWT compression proved to be the best compression algorithm tested. Future work would be to investigate the JPEG 2000 standard to see how officially the DWT is used to decompress images. Also we note that when compared to lossless compression algorithms, lossy compression has much greater potential for compression at the cost of image quality.

References

[1] N. Ahmed and K. Rao, 1974. Discrete Cosine Transform. IEEE Transactions on Computers, C-23(1), p.90.

[2] G. Jenkins and E. Parzen. 1968. Spectral Analysis and its Applications. Holden-Day.

[3] Mallat. S. 1998. A Wavelet Tour of Signal Processing. The Sparse Way.

[4] R. Gonzales, R. Woods. 2018. Digital Image Processing 4ed. Pearson.

[5] J. Barros. 2012. Applications of Wavelet Transform for Analysis of Harmonic Distortion in Power Systems: A review. IEEE Transactions on Instrumentation and Measurements. V(61).

[6] K. Wallace. 1992. The JPEG Still Compression Standard. Multimedia Engineering, Digital Equipment Corporation. IEEE Transactions on Consumer Electronics.

[7] H. Lohscheller. 1984. A subjectively Adaptive Image Communication System. IEEE Transactions on Consumer Electronics.

[8] MK. Ahmad. 2006. New Distortion Measures in Image Processing. Sampling Theory in Signal and Image Processing.

[9] Gowri.P , Senbaga Priya.K, Hari Prasath.R.K, Pavithra.S. 2019. Image Compression Using Singular Value Decomposition. International Journal of Mathematics Trends and Technology 65.7: 74-81.

Algorithms

Block Compression Function

```
% Compress by Block
% This function performs an operation as selected by the user to each block
% The results from all the blocks are synthesised into the full image
% Inputs:
% (0) inImage - in image
% (1) Block Size: (1) Block is a square ... must be even
% (2) CompressType: 'DCT', 'FFT',
% (3) varargin (1) = Kernel type - 'Gaussian', 'JPEG'
% (4) varargin (2)  = quality... only inputted for JPEG kernel type
% (5) varagin  (3) = DO - cut off frequency

% Written by Nikeet Pandit
```

```
function [I_recovered, I_compressed, kernel, elTime] = block_compress(inImage,
BlockSize, CompressType, varargin)
    %--- Converting image to double
    inImage = double(inImage); CompressType = lower(CompressType);

    %--- Setting variable inputs
    if length(varargin) == 1
        kernel_type = lower(varargin{1});
    elseif length(varargin) == 2
        kernel_type = lower(varargin{1});
        quality = varargin{2};
    elseif length(varargin) == 3
        kernel_type = lower(varargin{1});
        quality = varargin{2};
        DO = varargin{3};
    end
    tic
    if kernel_type == "jpeg"
        kernel = ...
        [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]; %default kernel selection
        kernel = kernel*quality;

        %--- Interpolating JPEG kernel to block size if not 8
        if BlockSize ~= 8
            [X, Y] = meshgrid(1:8, 1:8);
            [Xq, Yq] = meshgrid(linspace(1,8,BlockSize));
            kernel = interp2(X,Y,kernel,Xq,Yq,'cubic');
        end

    elseif kernel_type == "gaussian"
        addpath('Calc - Fun');
        [~, kernel, ~, ~, ~, ~] = lowpass_im(inImage(1:BlockSize/2,1:BlockSize/2), DO,
kernel_type, 'FFT');

    else
        disp('Invalid Kernel Selection');
        return
    end

    %--- Applying compressiong to block based on user selection
    switch CompressType
        case 'dct'
        %--- applying quantization to each block
        fun_dct_quant = @(block_struct) round(((dct2(block_struct.data)))./kernel);
        I_compressed = (blockproc(inImage,[BlockSize, BlockSize],fun_dct_quant));
        elTime = toc;

        %--- recovering image
        fun_dct_recover = @(block_struct) (idct2(block_struct.data.*kernel));
        I_recovered = blockproc(I_compressed,[BlockSize, BlockSize],fun_dct_recover);
        I_recovered = uint8(I_recovered);

        case 'fft'
        %--- applying quantization to each block
        fun_fft_quant = @(block_struct) round(((fft2(block_struct.data)))./kernel);
```

```
        I_compressed = (blockproc(inImage,[BlockSize, BlockSize],fun_fft_quant));

        %--- recovering image
        fun_dct_recover = @(block_struct) real((ifft2(block_struct.data.*kernel)));
        I_recovered = blockproc(I_compressed,[BlockSize, BlockSize],fun_dct_recover);

        otherwise
            disp("Invalid Compress Type Selection")
    end
```

SVD Compression Function

```
% This function compresses an image using SVD
% Inputs:
% (0) inImage - in image
% (1) Rank

% Output:
% (0) Approximated Image
% (1) Decomposition Matrix 1
% (2) Decomposition Matrix 2
% (3) Decomposition Matrix 3
% (4) Elapsed Time
% (5) Cumulative Energy vs Rank

% Written by Nikeet Pandit

function [compIm, U_approx, S_approx, V_approx, elTime,cum_energy] =
svd_compress(InImage, rank)

tic

% compute the SVD of I
[U,S,V] = svd(double(InImage),"econ");

% compute the k rank approximation of I
U_approx = double(U(:,1:rank)); % converting to single precision to write out (save
memory)
S_approx = double(S(1:rank,1:rank));
V_approx = double(V(:,1:rank));

% The compressed Image
compIm = U_approx*S_approx*(V_approx)';

elTime = toc;

% Returning cumulative energy
cum_energy = cumsum(diag(S))/sum(diag(S));

end
```

Ideal DWT Compression Function

```
% Ideal DWT Compression (method described in report)

% This function performs wavelet compression (DWT)
% Inputs:
% (0) inImage - in image
```

```
% (1) percent - percent of coefficients to keep
% (2) level - levels of wavelet decomposition
% (3) mWave - mother wavelet function

% Output
% (0) Coeffs Image - Wavelet Coefficients
% (1) Compressed Image
% (2) Elapsed time

% Written by Changin Oh

function [coeffsImage, compImage, elTime] = compressWithWavelet(inImage, percent,
level, mWave)

% Display error message if wrong percentage is given
if (percent > 100) || (percent < 0)
    error("Specify percentage correctly.")
end

% Measure time
tic

% Perform n-level wavelet decomposition of image using specific mother wavelet
[C, S] = wavedec2(inImage, level, mWave);

% Sort coefficients by magnitude
coeffs = sort(abs(C));

% Convert percentage into ratio
ratio = percent*0.01;

% Decide threshold level based on what percentage to keep coefficients
threshold = coeffs(floor(length(coeffs)*(1-ratio)));

% Find positions of coefficients whose magnitude is greater than threshold
positions = abs(C) > threshold;

% Keep coefficients whose magnitude is greater than threshold
deCompImage = C.*positions;

% Obtain compressed image
compImage = uint8(waverec2(deCompImage, S, mWave));

% Save time
elTime = toc;

% Decompose into approximation and detailed coefficients
Ac = appcoef2(C, S, mWave, level);

coeffsImage = rescale(Ac, 0, 255);
for i = level:-1:1
    [Hc, Vc, Dc] = detcoef2("all", C, S, i);

    coeffsImage = horzcat(imresize(coeffsImage, size(Hc)), uint8(Hc));
    coeffsImage = vertcat(coeffsImage, uint8([Vc Dc]));
end
end
```

Smooth FFT Compression

Smooth FFT compression uses the two functions appended below.

Low Pass Image Function

```
% Function low passes filters an image
% In function DFT may mean DFT or DCT!!!

% Inputs:
% (1) Im -image file read in by Imread(File)
% (2) D0 -cut off frequency
% (3) Type - 'Gaussian', or 'Butterworth'
% (4) Transform Type 'DCT', 'FFT'
% (5) If Butterworth is selected, must select order

% Outputs:
% (1) Filtered Image
% (2) Centered Kernel
% (3) Filtered Image (Freq domain)
% (4) Original Image in Freq domain

% Written by Nikeet Pandit

function [Im_Filter, H, Im_Filter_Freq, Im_DFT_return, Pt, Telapsed] = lowpass_im(Im,
D0, kernel_type, transform_type, varargin)

Tstart = tic;

transform_type = lower(transform_type);
kernel_type = lower(kernel_type);

if length(varargin) == 1 %Setting order for Butterworth if selected
    n = varargin{1};
else
    n = [];
end

% --- Determine image size and calculate padded image size
[M, N] = size(Im); P = M*2; Q = N*2;

% --- Construct kernel (frequency domain) to be used for filtering
H = kernel_construct(D0 , P, Q, kernel_type, n); %function is appended below

% --- Take DFT of the image with padding specified by P and Q

if transform_type == "fft"
    Im_DFT = fft2((Im),P,Q);
    Im_DFT_return = abs(fftshift((Im_DFT))); %centered spectrum
elseif transform_type == "dct"
    Im_DFT = dct2(Im,P,Q);
    Im_DFT_return = fftshift(Im_DFT);
else
    print("Improper Selection");
end


% --- Filter in frequency domain
Im_Filter = (H.*Im_DFT);

if transform_type == "fft"
    Im_Filter_Freq = fftshift(abs(Im_Filter)); %returning filtered spectrum
elseif transform_type == "dct"
    Im_Filter_Freq = fftshift((Im_Filter)); %returning filtered spectrum
end

% --- Check Power
```

```
Pt = (sum(abs(Im_Filter),'all')/(sum(abs(Im_DFT),'all')))*100;

%%Isolate real components only
if transform_type == "fft"
    Im_Filter = uint8(real(ifft2(Im_Filter)));
elseif transform_type == "dct"
    Im_Filter = uint8(idct2(Im_Filter));
end

% --- Crop Image to remove padding
Im_Filter = (Im_Filter(1:M, 1:N)); %returning cropped filtered image
H = fftshift(H);  %returning centered (and cropped) kernel function

Telapsed = toc(Tstart);


end
```

Kernel Construct Function

```
% Kernel Construct Fun

% Inputs:
% (1) DO is cut off frequency
% (2) P and Q are dimensions of padded kernel size
% (3) type is class string
% Options are: Gaussian, Butterworth

% Output:
% (1): Low pass kernel in frequency domain

% Written by Nikeet Pandit


function kernel = kernel_construct(D0, P, Q, type,varargin)

if length(varargin) ==1
    n = varargin{1};
end

type = lower(type);
u = 0:P-1; %frequency components
v = 0:Q-1;
[U, V] = meshgrid(u,v);
D = hypot(U-P/2, V-Q/2); %calculating distances

if type == "gaussian"
    kernel = fftshift(exp(-(D.^2)./(2*(D0^2))))'; %Shifts kernel to nominal fft2
position (uncentered)

elseif type == "butterworth"
    NumExp = D;
    DenExp = D0;
    %kernel = fftshift(1./(1+((NumExp./DenExp).^(2*n))))'; %Butterworth T.F.
(uncentered)
    Den = 1 + (D./D0)^(2*n);
    kernel = fftshift(1./Den)';

end
```

Ideal FFT Compression Function

```
% Ideal FFT Compression (method described in report)

% This function performs wavelet compression (DWT)
% Inputs:
% (0) inImage - in image
% (1) percent - percent of coefficients to keep


% Output
% (0) Input Image Spectrum
% (1) Compressed Image Spectrum
% (2) Compressed Image
% (3) Elapsed time

% Written by Changin Oh



function [spInImage, spCompImage, compImage, elTime] = compressWithFFT(inImage,
percent)

% Display error message if wrong percentage is given
if (percent > 100) || (percent < 0)
    error("Specify percentage correctly.")
end

% Measure time
tic

% Perform Fourier transform of image
trImage = fft2(inImage);

% Sort coefficients by magnitude
coeffs = sort(abs(trImage(:)));

% Convert percentage into ratio
ratio = percent*0.01;

% Decide threshold level based on what percentage to keep coefficients
threshold = coeffs(floor(length(coeffs)*(1-ratio)));

% Find positions of coefficients whose magnitude is greater than threshold
positions = abs(trImage) > threshold;

% Keep coefficients whose magnitude is greater than threshold
trCompImage = trImage.*positions;

% Obtain compressed image
compImage = uint8(ifft2(trCompImage));

% Save time
elTime = toc;

% Make spectrum of input image
spInImage = log(abs(fftshift(trImage))+1);

% Make spectrum of compressed image
spCompImage = log(abs(fftshift(trCompImage))+1);
end
```