Image Compression and Linear Algebra

Sunny Verma, Jakkam Phanindra Krishna

November 15, 2013

Contents

1 Introduction

With rapid demand of storing large amount of data, the space to store this data (in the form of files) in the hard drives of the computer systems or onto the servers of big companies is getting less compared to the amount of data that is to be stored. As a result of it, various compression techniques are in demand which can help to reduce the size of data files. In this project, we will be discussing how Linear algebra can be used in the compression of images. Basically we will be discussing how SVD and Wavelet techniques are extensively used in image compression process resulting in saving computer's memory. The basic idea here is each image can be represented as a matrix and we apply linear algebra (SVD and Wavelet) on this matrix and get a redcuced matrix out of this original matrix and the image corresponding to this reduced matrix requires much lesser storage space as compared to the original image.

2 Image compression using SVD

An mxn pixels image can be represented by mxn matrix representation. Suppose we have an 9 megapixel gray-scale image, which is 3000 $\times$ 3000 pixels (a 3000 $\times$ 3000 matrix).

For each pixel, we have some level of black and white color, given by some integer between 0 and 255. 0 representing black color and 255 representing white color.Each of these integers(and hence each pixel) requires approximately 1 byte to store, resulting in anapproximately 8.6 Mb image.

A color image usually has three components, a red, a green, and a blue(RGB). Each of these is represented by a matrix, hence storing color images requires three times the space (25.8 Mb).

Mathematics Behind the SVD

2.1 Rayleigh quotient and 2-norm of a matrix

Definition: For a given complex Hermitian matrix M and nonzero vector x, the Rayleigh quotient $R(M, x)$, is defined as:

$$R(M, x) = x^* Mx/x^* x.$$

For real matrices and vectors, the condition of being Hermitian reduces to that of being symmetric, and the conjugate transpose x^* to the usual transpose x^T. Note that $R(M, cx) = R(M, x)$ for any real scalar $c \neq 0$. Recall that a Hermitian (or real symmetric) matrix has real eigenvalues. It can be shown that, for a given matrix, the Rayleigh quotient reaches its minimum value $\lambda_{\min}$ (the smallest eigenvalue of M) when x is $v_{\min}$ (the corresponding eigenvector). Similarly, $R(M, x) \leq \lambda_{\max}$ and $R(M, v_{\max}) = \lambda_{\max}$. The Rayleigh quotient is used in min-max theorem to get exact values of all eigenvalues. It is also used in eigenvalue algorithms to obtain an eigenvalue approximation from an eigenvector approximation. Specifically, this is the basis for Rayleigh quotient iteration.

The 2- norm of a vector is given by:

$$\|x\| = \sqrt{\Sigma_{i=1}^{n} {x_i}^2}$$

Subordinate to the vector 2-norm is the matrix 2-norm :

$$\|A\|_2 = \text{largest eigenvalue of } A^*A.$$

Due to this connection with eigenvalues, the matrix 2-norm is called the spectral norm.

For an arbitrary $m \times n$ matrix A, note that A^*A is $n \times n$ and Hermitian. The eigenvalues of A^*A are real-valued. Also, A^*A is at least positive semi-definite since $X^*A^*AX = (AX)^*AX \geq 0 \forall X$ Hence, the eigenvalues of A^*A are both real-valued and non-negative; denote them as

$$\sigma_1{}^2 \geq \sigma_2 2 \geq \sigma_3{}^2 \geq ... \geq \sigma_n{}^2 \geq 0.$$

Note that these eigenvalues are arranged according to size with $\sigma_1{}^2$ being the largest. These eigenvalues are known as the singular values of matrix A. Corresponding to these eigenvalues are n orthonormal (hence, they are independent) eigenvectors $U_1, U_2, ..., U_n$ with

$$(A^*A)U_k = (\sigma_k{}^2)U_k, 1 \leq k \leq n.$$

The n eigenvectors form the columns of a unitary $n \times n$ matrix U that diagonalizes matrix A^*A under similarity (matrix $U^*(A^*A)U$ is diagonal with eigenvalues on the diagonal).

Since the n eigenvectors $U_1, U_2, ..., U_n$ are independent, they can be used as a basis, and vector X can be expresssed as

$$X = \Sigma_{k=1}^n c_k U_k$$

where the c_k are X-dependent constants. Multiply A^*A by X to obtain :

$$A^*AX = A^*A(\Sigma_{k=1}^n c_k U_k) = \Sigma_{k=1}^n c_k \sigma_k{}^2 U_k$$

which leads us to

$$\|AX\|_2{}^2 = (AX)^*AX = X^*(A^*AX) = (\Sigma_{k=1}^n c_k{}^* U_k{}^*)(\Sigma_{j=1}^n c_j \sigma_j{}^2 U_j = \Sigma_{k=1}^n |c_k|^2 \sigma_k{}^2 \leq \sigma_1{}^2(\Sigma_{k=1}^n |c_k|^2) = \sigma_1{}^2 \|X\|_2{}^2$$

for arbitrary X. Hence, we have completed step1: we found a constant $K = \sigma_1{}^2$ such that $\|AX\|_2 \leq K\|X\|_2$ for all X. Step2 requires us to find at least one vector X_0 for which equality holds; that is, we must find an X_0 with the property that $\|AX_0\|_2 = K\|X_0\|_2$ the unit-length eigenvector associated with eigenvalue $\sigma_1{}^2$, will work. Hence, the matrix 2-norm is given by $\|A\|_2 = \sqrt{\sigma_1{}^2}$, the square root of the largest eigenvalue of A^*A.

2.2 Singular Value Decomposition (SVD)

Formal definition Given Let $A \in \mathbb{C}^{\mathrm{m}\times\,\mathrm{n}}$.($m$ and n be arbitrary and A not necessarily of full rank),a *singular value decomposition* (SVD) of A is a factorization

$$A{=}U\Sigma V^*$$

where

$$\begin{gathered} U{\in}\ \mathbb{C}^{\mathrm{m}\times\,\mathrm{m}} \text{ is unitary,} \\ V{\in}\ \mathbb{C}^{\mathrm{n}\times\,\mathrm{n}} \text{ is unitary,} \\ \Sigma{\in}\ \mathbb{C}^{\mathrm{m}\times\,\mathrm{n}} \text{ is diagonal,} \end{gathered}$$

In addition,it is assumed that the daigonal entries σ_j of Σ are nonnegative and in non increasing order; that is, $\sigma_1 \geq \sigma_2 \geq \cdots\cdots \geq \sigma_p \geq 0$,where p=min(m,n)Note that the diagonal matrix Σ has the same shape as A even when A is not square,but U and V are always square unitary matrices.

Existence and Uniqueness Theorem:*Every matrix $A \in \mathbb{C}^{m\times n}$ has a singular value decomposition.*

Proof. To prove existence of the SVD.we isolate the direction of the largest action of A ,and then proceed by induction on yhe dimension of A.

Set $\sigma_1 - \|A\|_2$. By a compactness argument,there must be vector $v_1 \in \mathbb{C}^n$ and $u_1 \in \mathbb{C}^m$ with $\|v_1\|_2 = \|u_1\|_2 = 1$ and $Av_1 = \sigma u_1$. Consider any extension of v_1 to an orthonormal basis $\{v_j\}$ of $\mathbb{C}^m$ and of u_1 to an orthonormal basis $\{u_j\}$ of $\mathbb{C}^m$,and let U_1 and V_1 denote the unitary matrices with columns u_j and v_j, respectively.Then we have

$$U_1^* A\, V_1 = S = \begin{bmatrix} \sigma_1 & w^* \\ 0 & B \end{bmatrix},$$

where 0 is a column vector of dimension m-1,w^* is a row vector of dimension n-1, and B has dimension (m-1) $\times$ (n-1). Furthermore,

$$\left\| \begin{bmatrix} \sigma_1 & w^* \\ 0 & B \end{bmatrix} \begin{bmatrix} \sigma_1 \\ w \end{bmatrix} \right\|_2 \geq \sigma_1^2 + w^* w = (\sigma_1^2 + w^* w)^{1/2} \left\| \begin{bmatrix} \sigma_1 \\ w \end{bmatrix} \right\|_2,$$

implying $\|S\|_2 \geq (\sigma_1^2 + w^* w)^{1/2}$. Since U_1 and V_1 are unitary, we know that $\|s_2\| = \|A\|_2 = \sigma_1$, so this implies $w = 0$.

If $n = 1$ or $m = 1$, we are done.Otherwise, submatrix B describes the action of A on the subspace orthogonal to v_1.By the induction hypothesis, B has an SVD $B = U_2 \Sigma_2 V_2^*$.Now it is easily verified that

$$A = U_1 \begin{bmatrix} 1 & 0 \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & V_2 \end{bmatrix}^* V_1^*$$

is an SVD of A,completing the proof of existence.

- In addition,it is assumed that the daigonal entries σ_j of Σ are nonnegative and in non increasing order;

- that is, $\sigma_1 \geq \sigma_2 \geq \cdots\cdots \geq \sigma_p \geq 0$,where p=min(m,n)Note that the diagonal matrix Σ has the same shape as A even when A is not square,but U and V are always square unitary matrices.

- The diagonal entries of Σ are known as singular values of A . The m columns of U and n columns of V are called lest-singular and right-singular vectors of A .

2.3 Example of SVD

We have stated that the purpose of (SVD) is to factor matrix A into $U\Sigma V^T$. The matrix U contains the left singular vectors, the matrix V contains the right singular vectors, and the diagonal matrix Σ contains the singular values. Where the singular values are arranged on the main diagonal in such an order

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0,$$

where r is the rank of matrix A, and where $(_p)$ is the smaller of the dimensions m or n.

Arbitrary Example We begin the process of Singular Value Decomposition by selecting the matrix A which has m rows and n columns. Now, we need to factor A into three matrices U, Σ, V^T .

First we will find V . If you multiply both sides of the equation $A = U\Sigma V^T$ by A^T we get

$$A^T A = (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma^T U^T U\Sigma V^T \ .$$

Since $U^T U = I$ this gives

$$A^T A = V\Sigma^2 V^T$$

Now we need to diagonalize $A^T A$. If you will notice, this is very similar to the diagonalization of matrix A into $A = Q\Lambda Q^T$. Except our symmetric matrix is not A, it is

A^TA. To find V and Σ we need to find the eigenvalues and eigenvectors of A^TA. The eigenvalues are the square of the elements of Σ (the singular values), and the eigenvectors are the columns of V (the right singular vectors).

Eliminating V from the equation is very similar to eliminating U . Instead of multiplying on the left by A^T we will multiply on the right by A^T . This gives:

$$AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma V^T V\Sigma^T U^T \ .$$

Since $V^T V = \mathrm{I}$, this gives

$$AA^T = U\Sigma^2 U^T$$

Again we will find the eigenvectors, but this time for AA^T . These are the columns of U (the left singular vectors).

Since A is $m \times n$, Σ is $m \times n$ and

$$A^TA$$

produces an $n \times n$ matrix, and:

$$AA^T$$

produces an $m \times m$matrix,

$$A = \begin{pmatrix} u_1 & \cdots & u_r & \cdots & u_m \end{pmatrix} \begin{pmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & \ddots & \\ & & & & 0 \end{pmatrix} \begin{pmatrix} v_1^T \\ \vdots \\ v_r^T \\ \vdots \\ v_n^T \end{pmatrix}$$

Where U is $m \times m$, S is $m \times n$, V is $n \times n$.

Example

Let:

$$A = \begin{pmatrix} 2 & -2 \\ 1 & 1 \end{pmatrix}$$

$$A^TA = \begin{pmatrix} 2 & 1 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} 2 & -2 \\ 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 5 & -3 \\ -3 & 5 \end{pmatrix}$$

Subtracting λI from A^TA

$$\left| \begin{pmatrix} 5 & -3 \\ -3 & 5 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right| = 0$$

Therefore,

$$\begin{vmatrix} 5-\lambda & -3 \\ -3 & 5-\lambda \end{vmatrix} = 0$$

Now find λ,

$$\begin{aligned} (5-\lambda)(5-\lambda) - 9 &= 0 \\ \Rightarrow 25 - 10\lambda + \lambda^2 - 9 &= 0 \\ \Rightarrow \lambda^2 - 10\lambda + 16 &= 0 \\ \Rightarrow (\lambda - 8)(\lambda - 2) &= 0 \end{aligned}$$

Therefore our eigenvalues are 8 and 2. We construct the matrix S^2 by placing the eigenvalues along the main diagonal in decreasing order.

$$S^2 = \begin{pmatrix} 8 & 0 \\ 0 & 2 \end{pmatrix}$$

Therefore, taking the square root of matrix S^2 gives,

$$S = \begin{pmatrix} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{pmatrix}$$

Now we need to find the eigenvectors of $A^T A$ which are the columns of V
First we will show where $\lambda = 8$,

$$\left[\begin{pmatrix} 5 & -3 \\ -3 & 5 \end{pmatrix} - \begin{pmatrix} 8 & 0 \\ 0 & 8 \end{pmatrix} \right] \hat{v_1} = 0$$

$$\Longrightarrow \begin{pmatrix} -3 & -3 \\ -3 & -3 \end{pmatrix} \hat{v_1} = 0$$

$$\Longrightarrow \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \hat{v_1} = 0$$

Therefore,

$$\hat{v_1} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Since V has an orthonormal basis, $\hat{v_1}$ needs to be of length one. We divide $\hat{v_1}$ by its magnitude to accomplish this. Thus,

$$\hat{v_1} = \begin{pmatrix} \frac{-\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix}$$

Similarly, well show where $\lambda = 2$

$$\left[\begin{pmatrix} 5 & -3 \\ -3 & 5 \end{pmatrix} - \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}\right] \hat{v_2} = 0$$

$$\implies \begin{pmatrix} 3 & -3 \\ -3 & 3 \end{pmatrix} \hat{v_2} = 0$$

$$\implies \begin{pmatrix} 1 & -1 \\ 0 & 0 \end{pmatrix} \hat{v_2} = 0$$

Therefore,

$$\hat{v_2} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Similarly dividing by the magnitude to create the orthonormal basis gives,

$$\hat{v_2} = \begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix}$$

Now we need to construct the augmented orthogonal matrix V ,

$$V = \begin{pmatrix} v_1 & v_2 \end{pmatrix}$$

and,

$$V = \begin{pmatrix} \frac{-\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix}$$

Now we need to find the eigenvectors for AA^T . Since the eigenvalues for AA^T are the same as the eigenvalues for A^TA. We can go straight to finding the eigenvectors using the eigenvalues previously found.
First well show where $\lambda = 8$,

$$\left[\begin{pmatrix} 8 & 0 \\ 0 & 2 \end{pmatrix} - \begin{pmatrix} 8 & 0 \\ 0 & 8 \end{pmatrix}\right] \hat{u_1} = 0$$

$$\implies \begin{pmatrix} 0 & 0 \\ 0 & -6 \end{pmatrix} \hat{u_1} = 0$$

$$\implies \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \hat{u_1} = 0$$

Therefore,

$$\hat{u_1} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

Since $\hat{u_1}$ is already of length one, dividing by the magnitude we get the same vector back. Similarly, well show where $\lambda = 2$

$$\left[\begin{pmatrix} 8 & 0 \\ 0 & 2 \end{pmatrix} - \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}\right] \hat{u_2} = 0$$

$$\implies \begin{pmatrix} 6 & 0 \\ 0 & 0 \end{pmatrix} \hat{u_2} = 0$$

$$\implies \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \hat{u_2} = 0$$

Therefore,

$$\hat{u_2} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Setting up the augmented matrix U

$$U = \begin{pmatrix} u_1 & u_2 \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

Therefore

$$A = USV^T$$

$$A = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{pmatrix} \begin{pmatrix} \frac{-\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix}$$

2.4 Approximation of a matrix using SVD

A $m \times n$ matrix A can be expressed as a linear combination of the singular values of the matrix A and the corresponding vectors u_i and ${v_i}^T$ (where u_i is i^{th} column of U and ${v_i}^T$ s i^{th} column of V)

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

Now we can write A as:

$$A = u_1\sigma_1{v_1}^T + u_2\sigma_2{v_2}^T + ... + u_i\sigma_i{v_i}^T + ... + u_n\sigma_n{v_n}^T$$

$\Longrightarrow$

$$A = \sigma_1 u_1 v_1 + \sigma_2 u_2 v_2 + ... + \sigma_i u_i v_i + ... + \sigma_n u_n v_n$$

The terms $\{\sigma_1 u_1 v_1, \sigma_2 u_2 v_2, ..., \sigma_n u_n v_n\}$ are in order of dominance from greatest to least. (as $\sigma_1 \geq \sigma_2 \geq \cdots\cdots \geq \sigma_n$)

An approximation of the matrix A can be achieved by reducing the number of iterations involved in the linear combination:

$$A_i = \sigma_1 u_1 v_1 + \sigma_2 u_2 v_2 + ... + \sigma_i u_i v_i$$

Keeping only some of these terms may result in a lower image quality, but lower storage size. This process is sometimes called Principal Component Analysis (PCA) .

2.5 Saving memory using SVD

The matrix A requires n^2 elements. U and V^T require n^2 elements each and Σ requires n elements. Storing the full svd then requires $2n^2 + n$ elements. Keeping 1 term in the svd, $u_1 \sigma_1 v_1^T$, requires only $(2n + 1)$ elements.

If we keep first $i < \frac{n}{2}$ terms, then storing the reduced matrix requires $i(2n + 1)$ elements, which are less than the original number of elements $(2n^2 + n)$ in the matrix.

As a result of PCA, we get the reduced matrix, in other words the elements of this reduced matrix stores the corresponding pixels of the compressed image.

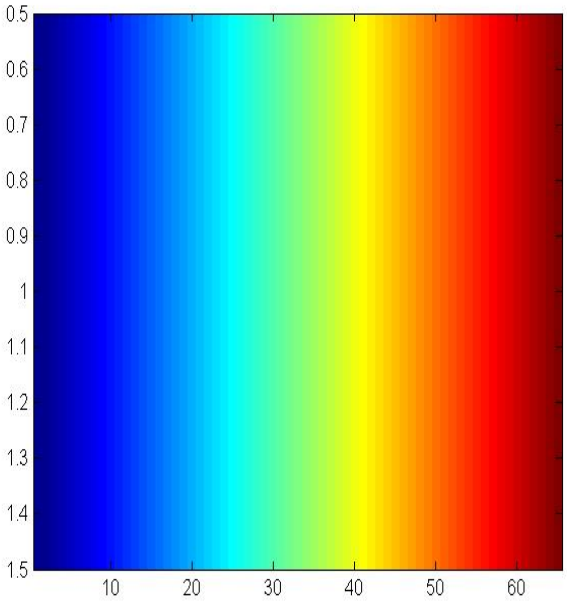
Thus the memory of the system is reduced in storing the image without compromising the quality of the image.

2.6 Image spectrum in Matlab

In order to understand how SVD is used in image compression, a brief discussion about how matlab constructs images is necessary .
Basically each entry in the matrix corresponds to small square of the image.
The numeric value of the entry corresponds to a color in matlab.
The color spectrum can be seen in matlab by typing the following commands :

$$>> C = 1 : 64;$$
$$image(C)$$

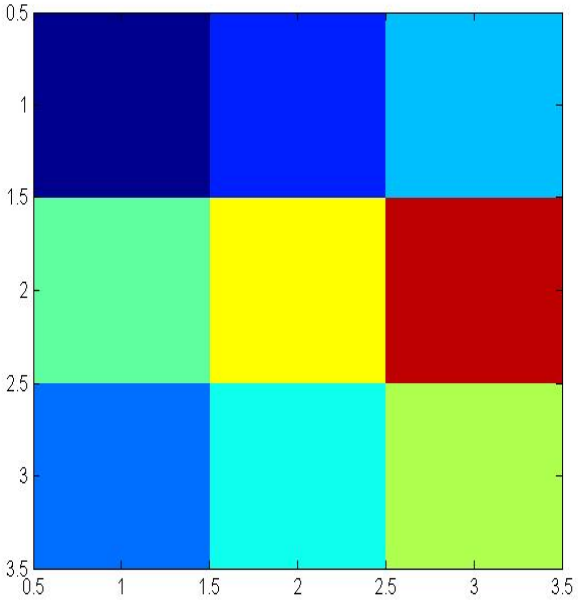
As a reult of this, we get the following image of the spectrum as the output :



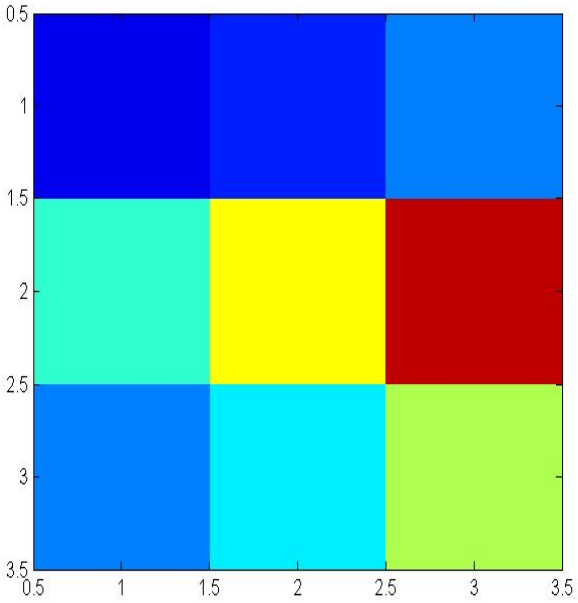
With this in mind, entering a 3×3 matrix of random integers should give a picture of nine square blocks comprising one large block. Furthermore, the color of each individual block will correspond to the color pictured at that numerical value as shown .

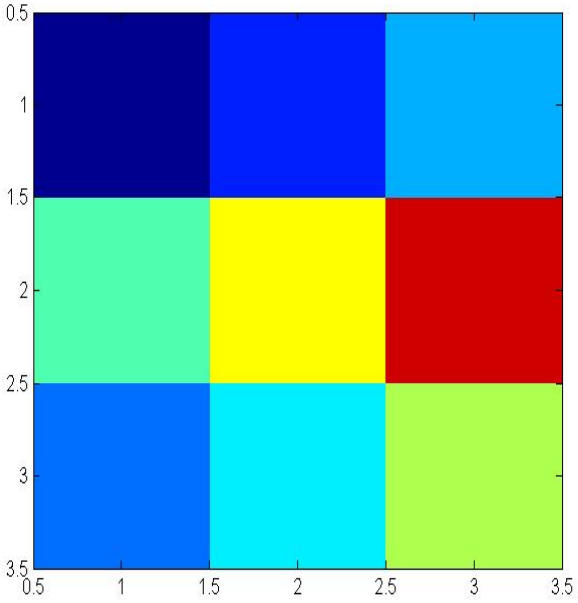
$$\text{Let S} = \begin{pmatrix} 1 & 10 & 20 \\ 30 & 40 & 60 \\ 15 & 25 & 35 \end{pmatrix}$$

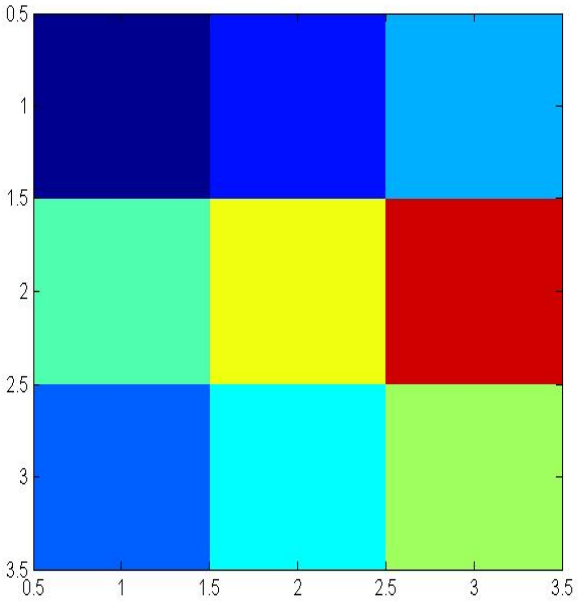
$$>> image(S)$$



It has been shown that any matrix S can be approximated using a lesser number of iterations when calculating the linear cimbinations defining S. This can be shown using matlab command "*svdimage*" . "*svdimage*" is an interactive program which shows the original image produced by S and the image produced by approximated s . The following are the three images with respect to first, second and third iterations for the 3×3 matrix S :







2.7 SVD image example

We take an image and discuss the affect that different iterations will have on it in the following pages.

Image output using 10 singular values

Image output using 20 singular values

Image output using 30 singular values

Image output using 40 singular values

Image output using 50 singular values

Image output using 60 singular values

Image output using 70 singular values

Image output using 80 singular values

Image output using 90 singular values

Image output using 100 singular values

Image output using 110 singular values

Image output using 120 singular values

Image output using 130 singular values

Image output using 140 singular values

Image output using 150 singular values

3 Haar wavelet

3.1 Preliminaries

Another technique for image compression is wavelet compression which we will discuss here.

Haar wavelet compression is an efficient way to perform both lossless and lossy image compression. It relies on averaging and differencing values in an image matrix to produce a matrix which is sparse or nearly sparse. A **sparse matrix** is a matrix in which a large portion of its entries are 0. A sparse matrix can be stored in an efficient manner, leading to smaller file sizes.

In information technology, "**lossy**" compression is a data encoding method that compresses data by discarding (losing) some of it. The procedure aims to minimize the amount of data that needs to be held, handled, and/or transmitted by a computer.

Lossless data compression is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data. By contrast, lossy data compression, permits reconstruction only of an approximation of the original data, though this usually allows for improved compression rates (and therefore smaller sized files).

In our project we concentrated on grayscale images; however, rgb(colour) images can be handled by compressing each of the color layers separately. The basic method is to start with an image A, which can be regarded as an m×n matrix with values 0(for black) to 255(for white). In Matlab, this would be a matrix with unsigned 8-bit integer values. We then subdivide this image into 8×8 blocks, padding as necessary. It is these 8×8 blocks that we work with

3.2 An Example

Below is a 512×512 pixel grayscale image of the flying buttresses of the Notre Dame Cathedral in Paris:

Figure 1: Flying buttresses of Notre Dame de Paris

And the following is the upper left 8×8 section of our image:

$$A = \begin{pmatrix} 88 & 88 & 89 & 90 & 92 & 94 & 96 & 97 \\ 90 & 90 & 91 & 92 & 93 & 95 & 97 & 97 \\ 92 & 92 & 93 & 94 & 95 & 96 & 97 & 97 \\ 93 & 93 & 94 & 95 & 96 & 96 & 96 & 96 \\ 92 & 93 & 95 & 96 & 96 & 96 & 96 & 95 \\ 92 & 94 & 96 & 98 & 99 & 99 & 98 & 97 \\ 94 & 96 & 99 & 101 & 103 & 103 & 102 & 101 \\ 95 & 97 & 101 & 104 & 106 & 106 & 105 & 105 \end{pmatrix}$$

We will concentrate on the first row:

$$r_1 = \begin{pmatrix} 88 & 88 & 89 & 90 & 92 & 94 & 96 & 97 \end{pmatrix}$$

Our transformation process will occur in three steps. The first step is to group all of the columns in pairs:

[88, 88], [89, 90], [92, 94], [96, 97]

we replace the first 4 columns of r_1 with the average of these pairs and replace the last 4 columns of r_1 with 1/2 of the difference of these pairs. We will denote this new row as r_1h_1 :

$$r_1h_1 = \begin{pmatrix} 88 & 89.5 & 93 & 96.5 & 0 & -0.5 & -1 & -0.5 \end{pmatrix}$$

The first 4 entries are called the **approximation coefficients** and the last 4 are called **detail coefficients**.
Next, we group the first 4 columns of this new row:

$$[88,\ 89.5],\ [93,\ 96.5]$$

and replace the first 2 columns of r_1h_1 with the average of the pairs and the next 2 columns of r_1h_1 with 1/2 of the difference of these pairs. We leave the last 4 rows of r_1h_1 unchanged. We will denote this second new row as $r_1h_1h_2$:

$$r_1h_1h_2 = \begin{pmatrix} 88.75 & 94.75 & -0.75 & -1.75 & 0 & -0.5 & -1 & -0.5 \end{pmatrix}$$

Finally, our last step is to group the first 2 entries of $r_1h_1h_2$ together:

$$[88.75,\ 94.75]$$

and replace the first column of $r_1h_1h_2$ with the average of the pairs and the second column of $r_1h_1h_2$ with 1/2 of the difference of these pairs. We leave the last 6 rows of $r_1h_1h_2$ unchanged. We will denote this last new row as $r_1h_1h_2h_3$:

$$r_1h_1h_2h_3 = \begin{pmatrix} 91.75 & -3 & -0.75 & 2 & -1.75 & 0 & -0.5 & -1 & -0.5 \end{pmatrix}$$

We then repeat this process for the remaining rows of A. After this, we repeat this same process to columns of A, grouping rows in the same manner as columns.The resulting matrix is:

$$\begin{pmatrix} 96. & -2.03125 & -1.53125 & -0.21875 & -0.4375 & -0.75 & -0.3125 & 0.125 \\ -2.4375 & -0.03125 & 0.78125 & -0.78125 & 0.4375 & 0.25 & -0.3125 & -0.25 \\ -1.125 & -0.625 & 0 & -0.625 & 0 & 0 & -0.375 & -0.125 \\ -2.6875 & 0.75 & 0.5625 & -0.0625 & 0.125 & 0.25 & 0 & 0.125 \\ -0.6875 & -0.3125 & 0 & -0.125 & 0 & 0 & 0 & -0.25 \\ -0.1875 & -0.3125 & 0 & -0.375 & 0 & 0 & -0.25 & 0 \\ -0.875 & 0.375 & 0.25 & -0.25 & 0.25 & 0.25 & 0 & 0 \\ -1.25 & 0.375 & 0.375 & 0.125 & 0 & 0.25 & 0 & 0.25 \end{pmatrix}$$

Notice that this resulting matrix has several 0 entries and most of the remaining entries are close to 0. This is a result of the differencing and the fact that adjacent pixels in an image generally do no differ by much. We will now discuss how to implement this process using matrix multiplication below.

3.3 Haar Wavelet Transformation Matrix

Haar Wavelet Transformation Matrix

If we let

$$H_1 = \begin{pmatrix} 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & -1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & -1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & -1/2 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & -1/2 \end{pmatrix}$$

then AH_1 is equivalent to the first step above applied to all of the rows of A.In particular,

$$AH_1 = \begin{pmatrix} 88 & 89.5 & 93 & 96.5 & 0 & -0.5 & -1 & -0.5 \\ 90 & 91.5 & 94 & 97 & 0 & -0.5 & -1 & 0 \\ 92 & 93.5 & 95.5 & 97 & 0 & -0.5 & -0.5 & 0 \\ 93 & 94.5 & 96 & 96 & 0 & -0.5 & 0 & 0 \\ 92.5 & 95.5 & 96 & 95.5 & -0.5 & -0.5 & 0 & 0.5 \\ 93 & 97 & 99 & 97.5 & -1 & -1 & 0 & 0.5 \\ 95 & 100 & 103 & 101.5 & -1 & -1 & 0 & 0.5 \\ 96 & 102.5 & 106 & 105 & -1 & -1.5 & 0 & 0 \end{pmatrix}$$

Similarly, by defining H_2 by:

$$H_2 = \begin{pmatrix} 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & -1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & -1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

then then AH_1H_2 is equivalent to the two steps above applied to all of the rows of A.In

particular,

$$AH_1H_2 = \begin{pmatrix} 88.75 & 94.75 & -0.75 & -1.75 & 0 & -0.5 & -1 & -0.5 \\ 90.75 & 95.5 & -0.75 & -1.5 & 0 & -0.5 & -1 & 0 \\ 92.75 & 96.25 & -0.75 & -0.75 & 0 & -0.5 & -0.5 & 0 \\ 93.75 & 96 & -0.75 & 0 & 0 & -0.5 & 0 & 0 \\ 94 & 95.75 & -1.5 & 0.25 & -0.5 & -0.5 & 0 & 0.5 \\ 95 & 98.25 & -2 & 0.75 & -1 & -1 & 0 & 0.5 \\ 97.5 & 102.25 & -2.5 & 0.75 & -1 & -1 & 0 & 0.5 \\ 99.25 & 105.5 & -3.25 & 0.5 & -1 & -1.5 & 0 & 0 \end{pmatrix}$$

Finally, if we define

$$H_3 = \begin{pmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & -1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

then then $AH_1H_2H_3$ is equivalent to the all three steps above applied to all of the rows of A. In particular,

$$AH_1H_2H_3 = \begin{pmatrix} 91.75 & -3 & -0.75 & -1.75 & 0 & -0.5 & -1 & -0.5 \\ 93.125 & -2.375 & -0.75 & -1.5 & 0 & -0.5 & -1 & 0 \\ 94.5 & -1.75 & -0.75 & -0.75 & 0 & -0.5 & -0.5 & 0 \\ 94.875 & -1.125 & -0.75 & 0 & 0 & -0.5 & 0 & 0 \\ 94.875 & -0.875 & -1.5 & 0.25 & -0.5 & -0.5 & 0 & 0.5 \\ 96.625 & -1.625 & -2 & 0.75 & -1 & -1 & 0 & 0.5 \\ 99.875 & -2.375 & -2.5 & 0.75 & -1 & -1 & 0 & 0.5 \\ 102.375 & -3.125 & -3.25 & 0.5 & -1 & -1.5 & 0 & 0 \end{pmatrix}$$

If we let H be the product of these three matrices, then

$$H = H_1H_2H_3 = \begin{pmatrix} 1/8 & 1/8 & 1/4 & 0 & 1/2 & 0 & 0 & 0 \\ 1/8 & 1/8 & 1/4 & 0 & -1/2 & 0 & 0 & 0 \\ 1/8 & 1/8 & -1/4 & 0 & 0 & 1/2 & 0 & 0 \\ 1/8 & 1/8 & -1/4 & 0 & 0 & -1/2 & 0 & 0 \\ 1/8 & -1/8 & 0 & 1/4 & 0 & 0 & 1/2 & 0 \\ 1/8 & -1/8 & 0 & 1/4 & 0 & 0 & -1/2 & 0 \\ 1/8 & -1/8 & 0 & -1/4 & 0 & 0 & 0 & 1/2 \\ 1/8 & -1/8 & 0 & -1/4 & 0 & 0 & 0 & -1/2 \end{pmatrix}$$

Note the following:

- The columns of the matrix H_1 form an orthogonal subset of $\mathbb{R}^8$ (the vector space of dimension 8 over $\mathbb{R}$); that is these columns are pair wise orthogonal (try their dot products). Therefore, they form a basis of $\mathbb{R}$. As a consequence, H_1 is invertible. The same is true for H_2 and H_3.

- As a product of invertible matrices, H is also invertible and its columns form an orthogonal basis of $\mathbb{R}$.

To apply the procedure to the columns, we just multiply A on the left by H^T . So the resulting matrix is

$$H^TAH = \begin{pmatrix} 96. & -2.03125 & -1.53125 & -0.21875 & -0.4375 & -0.75 & -0.3125 & 0.125 \\ -2.4375 & -0.03125 & 0.78125 & -0.78125 & 0.4375 & 0.25 & -0.3125 & -0.25 \\ -1.125 & -0.625 & 0 & -0.625 & 0 & 0 & -0.375 & -0.125 \\ -2.6875 & 0.75 & 0.5625 & -0.0625 & 0.125 & 0.25 & 0 & 0.125 \\ -0.6875 & -0.3125 & 0 & -0.125 & 0 & 0 & 0 & -0.25 \\ -0.1875 & -0.3125 & 0 & -0.375 & 0 & 0 & -0.25 & 0 \\ -0.875 & 0.375 & 0.25 & -0.25 & 0.25 & 0.25 & 0 & 0 \\ -1.25 & 0.375 & 0.375 & 0.125 & 0 & 0.25 & 0 & 0.25 \end{pmatrix}$$

The important part of this process is that it is reversible. In particular, H is an invertible matrix. If

$$B = H^TAH$$

then

$$A = (H^T)^{-1}BH^{-1}$$

B represents the compressed image of A (in the sense that it is sparse). Moreover, since H is invertible, this compression is lossless

3.4 Lossy Compression

The Haar wavelet transform can be used to perform lossy compression so that the compressed image retains its quality. First, the **compression ratio** of an image is the ratio of the non-zero elements in the original to the non-zero elements in the compressed image.

Let A be one of our 8×8 blocks and $H^T AH$ be the Haar wavelet compressed image of A. $H^T AH$ contains many detail coefficients which are close to 0. We will pick a number $\epsilon > 0$ and set all of the entries of $H^T AH$ with absolute value at mostϵ to 0. This matrix now has more 0 values and represents a more compressed image. When we decompress this image using the inverse Haar wavelet transform, we end up with an image which is close to the original.

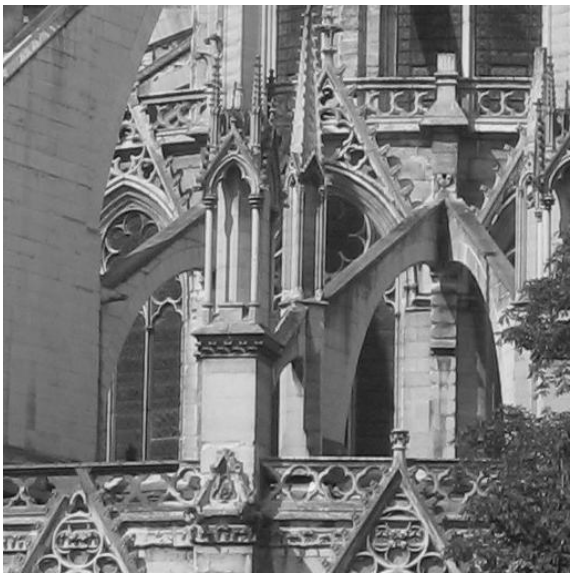
For example, with our matrix A from the example, if we choose $\epsilon = 0.25$ then we end up with the matrix:

$$\begin{pmatrix} 96. & -2.03125 & -1.53125 & 0 & -0.4375 & -0.75 & -0.3125 & 0 \\ -2.4375 & 0 & 0.78125 & -0.78125 & 0.4375 & 0 & -0.3125 & 0 \\ -1.125 & -0.625 & 0 & -0.625 & 0 & 0 & -0.375 & 0 \\ -2.6875 & 0.75 & 0.5625 & 0 & 0 & 0 & 0 & 0 \\ -0.6875 & -0.3125 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.3125 & 0 & -0.375 & 0 & 0 & 0 & 0 \\ -0.875 & 0.375 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1.25 & 0.375 & 0.375 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

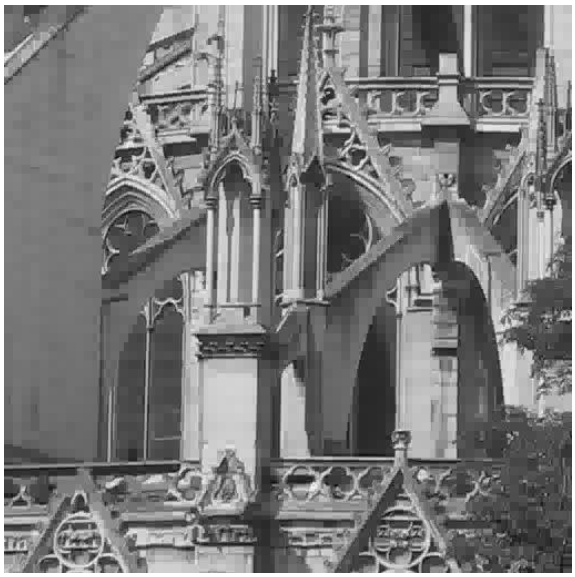
This matrix represents a compression ratio of $\frac{48}{27} = 1.7$.

Compression ratio If we choose our threshold value ϵ to be positive (i.e. greater than zero), then some entries of the transformed matrix will be reset to zero and therefore some detail will be lost when the image is decompressed. The key issue is then to choose ϵ wisely so that the compression is done effectively with a minimum damage to the picture. Note that the compression ratio is defined as the ratio of nonzero entries in the transformed matrix ($B = H^T AH$) to the number of nonzero entries in the compressed matrix obtained from B by applying the threshold ϵ.

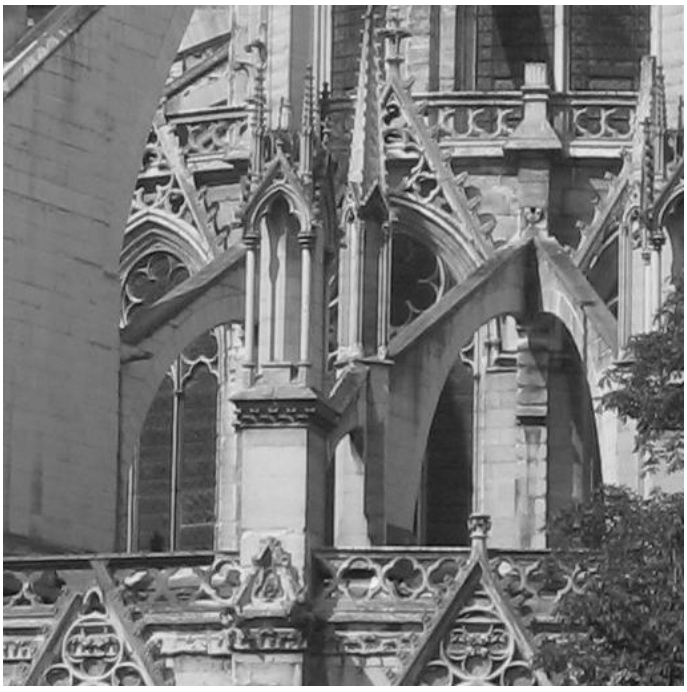
some images: Original Image , 10.1 Compression Ratio , 30:1 Compression Ratio , 50:1 Compression Ratio



(a) Original Image

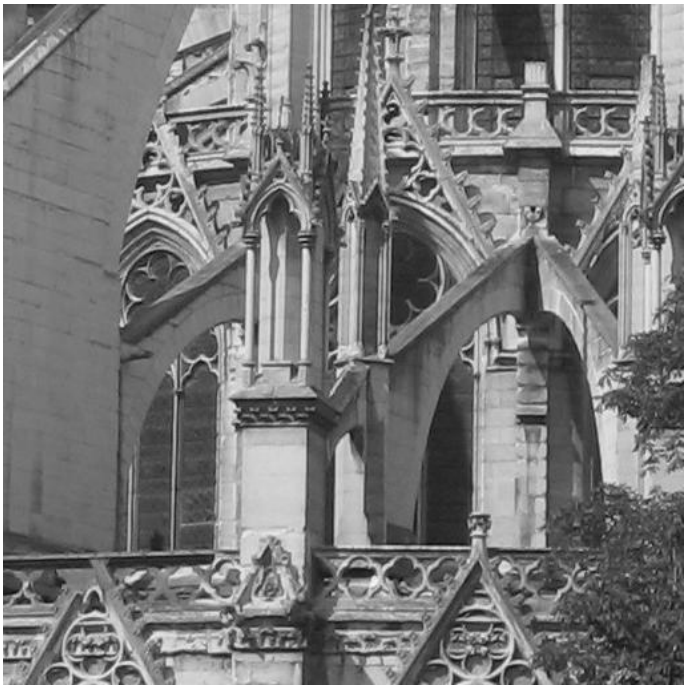


(b) 10:1 Compression Ratio

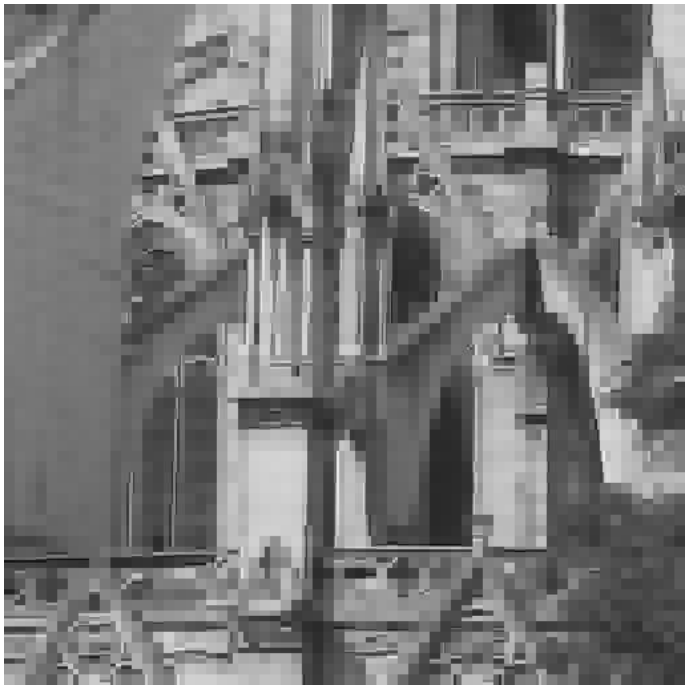


(c) Original Image

(d) 30:1 Compression Ratio



(e) Original Image



(f) 50:1 Compression Ratio

3.5 Normalization

we can make the compression process faster, more efficient

Let us first recall that an $n \times n$ square matrix A is called orthogonal if its columns form an orthonormal basis of $\mathbb{R}^n$, that is the columns of A are pairwise orthogonal and the length of each column vector is 1. Equivalently, A is orthogonal if its inverse is equal to its transpose. That latter property makes retrieving the transformed image via the equation

$$A = (H^T)^{-1}BH^{-1} = (H^{-1})^T BH^{-1} = HBH^T$$

much faster.

Another powerful property of orthogonal matrices is that they preserve magnitude. In other words, if v is a vector of $\mathbb{R}^n$ and A is an orthogonal matrix, then $\|Av\| = \|v\|$. Here is how it works:

$$\begin{aligned} \|Av\|^2 = (Av)^T(Av) & \\ &= v^T A^T Av \\ &= v^T Iv \\ &= v^T v \\ &= \|v\|^2 \end{aligned}$$

This in turns shows that $\|Av\| = \|v\|$. Also, the angle is preserved when the transformation is by orthogonal matrices: recall that the cosine of the angle between two vectors u

and v is given by:

$$\cos(\phi) = \frac{u.v}{\|u\| \, \|v\|}$$

so, if A is an orthogonal matrix, ψ is the angle between the two vectors Au and Av, then

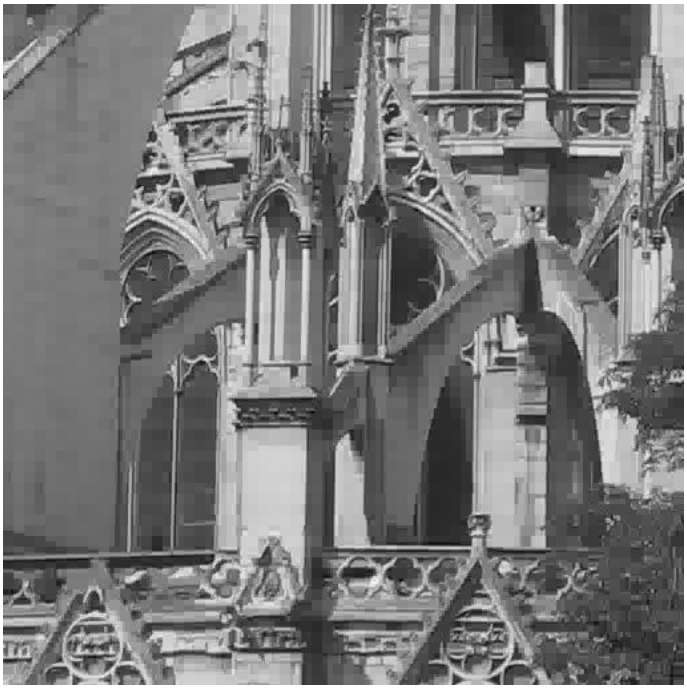
$$\cos(\psi) = \frac{(Au)(Av)}{\|Au\| \, \|Av\|} = \frac{(Au)^T(Av)}{\|u\| \, \|v\|} = \frac{u^t A^T Av}{\|u\| \, \|v\|} = \frac{u^T v}{\|u\| \, \|v\|} = \frac{u.v}{\|u\| \, \|v\|} = \cos(\psi)$$

Since both magnitude and angle are preserved, there is significantly less distortion produced in the rebuilt image when an orthogonal matrix is used. Since the transformation matrix H is the product of three other matrices, one can normalize H by normalizing each of the three matrices. The normalized version of H is

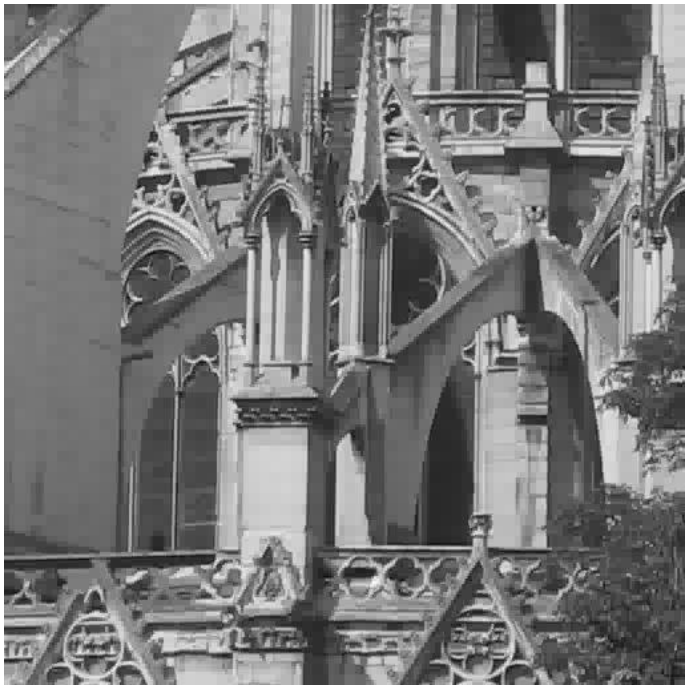
$$H = \begin{pmatrix} \sqrt{8}/64 & \sqrt{8}/64 & 1/2 & 0 & \sqrt{2}/4 & 0 & 0 & 0 \\ \sqrt{8}/64 & \sqrt{8}/64 & 1/2 & 0 & -\sqrt{2}/4 & 0 & 0 & 0 \\ \sqrt{8}/64 & \sqrt{8}/64 & -1/2 & 0 & 0 & \sqrt{2}/4 & 0 & 0 \\ \sqrt{8}/64 & \sqrt{8}/64 & -1/2 & 0 & 0 & -\sqrt{2}/4 & 0 & 0 \\ \sqrt{8}/64 & -\sqrt{8}/64 & 0 & 1/2 & 0 & 0 & \sqrt{2}/4 & 0 \\ \sqrt{8}/64 & -\sqrt{8}/64 & 0 & 1/2 & 0 & 0 & -\sqrt{2}/4 & 0 \\ \sqrt{8}/64 & -\sqrt{8}/64 & 0 & -1/2 & 0 & 0 & 0 & \sqrt{2}/4 \\ \sqrt{8}/64 & -\sqrt{8}/64 & 0 & -1/2 & 0 & 0 & 0 & -\sqrt{2}/4 \end{pmatrix}$$

Remark If you look closely at the process we described above, you will notice that the matrix W is nothing but a change of basis for $\mathbb{R}^8$. In other words, the columns of H form a new basis (a very nice one) of $\mathbb{R}^8$. So when you multiply a vector v (written in the standard basis) of $\mathbb{R}^8$ by H, what you get is the coordinates of v in this new basis. Some of these coordinates can be neglected using our threshold and this what allows the transformed matrix to be stored more easily and transmitted more quickly.

images The two images below represent a 10:1 compression using the standard and normalized Haar wavelet transform. Notice that the rounded and angled sections in the normalized image are of a higher quality.



(g) Standard, 10:1 Compression



(h) Normalized, 10:1 Compression

3.6 Progressive Transmission

The Haar wavelet transformation can also be used to provide progressive transmission of a compressed image.

Every time you click on an image to download it from the Internet, the source computer recalls the Haar transformed matrix from its memory. It first sends the overall approximation coefficients and larger detail coefficients and a bit later the smaller detail coefficients. As your computer receives the information, it begins reconstructing in progressively greater detail until the original image is fully reconstructed.

example: An easy way to do this is consider the images on page 7. First the 50:1 image in its Haar wavelet compressed state is sent. The receiving computer decompresses this to show the 50:1 image. Next, only the detail coefficients which were zeroed out in the 50:1 image but not in the 30:1 image are sent. This new data along with the 50:1 image allows us to reconstruct the 30:1 image. Similarly, the detail coefficients which were zeroed out to give the 30:1 image but not the 10:1 image are sent. From this we construct the 10:1 image. Finally the remaining detail coefficients are sent, allowing us to reconstruct the original image. We can stop this process at any time to view the compressed image rather than the original.

4 Conclusion

Using SVD and Wavelet techniques, we can save a lot of computer memory which can be used for other purposes. From our personal computers to the servers of all the big networking websites, all rely heavily on image compression techniques for saving memory, which in turn relies on linear algebra and we have discussed in detail in this project how the image compression is done using SVD and Wavelet.

References

[1] A.K. Jain," Fundamentals of Digital Image Processing"(1989).

[2] David S. Watkins. "Fundamentals of Matrix Computations" (1991).

[3] James W. Demmel " Applied Numerical Linear Algebra "(1997).

[4] Lloyd Trefethen and David Bau" Numerical Linear Algebra" (1997).