Daniel Fischer

Numerical Methods and Computational Techniques

Transforms and Convolution: Image Analysis through MATLAB

In the modern world, images are ubiquitous. Represented most often on screens, images can either be photographs of the world, artistic creations, or representations of data. With great advances in screen technology, Light Emitting Diodes, and other display technologies, the paradigm of grainy, out of focus images is no more. Whether layperson, consumer, scientist, film-maker, or photographer, high resolution imagery in the public sphere has increased graphical processing necessities in numerous applications. This paper is discussing methods of transforming images using mathematical methods.

Improvements in technology have given birth to the ability to record and display large amounts of data in an image. At the same time, storage and transmission of that data is at a premium, and the larger a data set is the more expensive it becomes to store or transmit that information. Using numerical methods the data can be compressed and analyzed in order to transmit the same amount of information using limited resources. Fundamentally, all information in an image can be interpreted as a matrix of congealing light emissions of the colours red, green, and blue. This coalescing of photons allows the image to be resolved by the human eye, or as some marketing prefers to call it, ‘retina display’.

Not all images are posted online in the highest quality format for reasons of saving hard disk drive storage space, increasing load speeds of certain web pages for users to see more readily, or because of other circumstances. Fundamentally, images that have been compressed contain artifacts of this compression of information. The remnants of these algorithms creates an image with apparent representations of squares, rectangles, and oddly place 'zig-zag' shapes that are far from a natural representation. These shapes are created because the smallest collection of image information is identified as a pixel. Pixels are small, quantized samples of data that all together, and when there are enough of them in a file, represent an entire image. Compressed images take clusters of pixels' information and 'smears' their data between them, often using the compression method known as JPEG2000 or JPEG. The method of Joint Photographic Experts Group or jpeg for short, uses matrices to transform the image into smaller forms via wavelet methods and compresses the resultant image using the Huffman Compression algorithm.

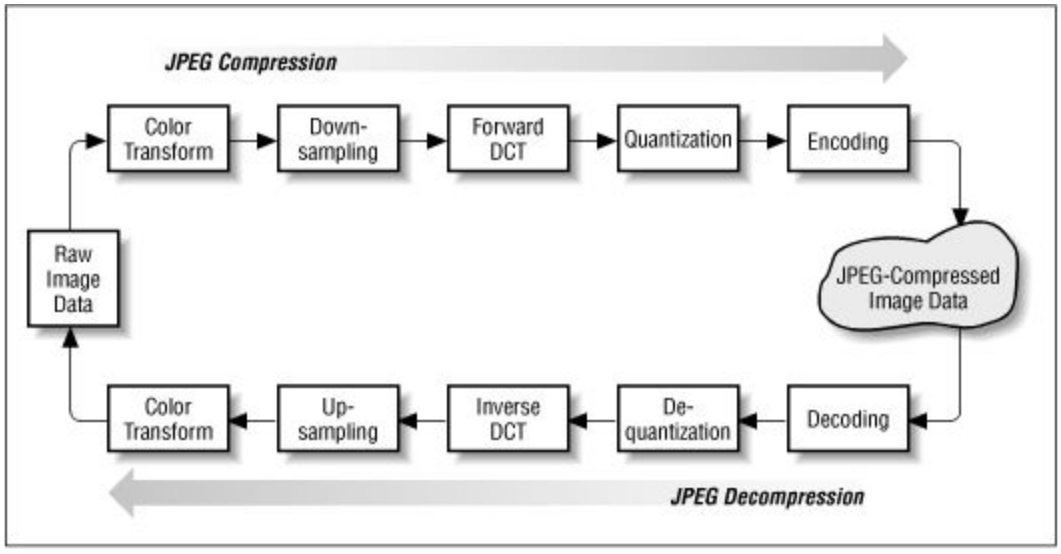


Figure 1: Regular jpeg compression using the discrete cosine transform. Credit for image, "http://www.fileformat.info/mirror/egff/ch09_06.htm". Note, the above figure implies that the

original image can be reconstructed from the compressed image; however, this is only this case if one has the original image data and can extrapolate the missing information from the compressed image. Typically, this isn't the case and the resulting image is lower quality, but takes up significantly less memory.

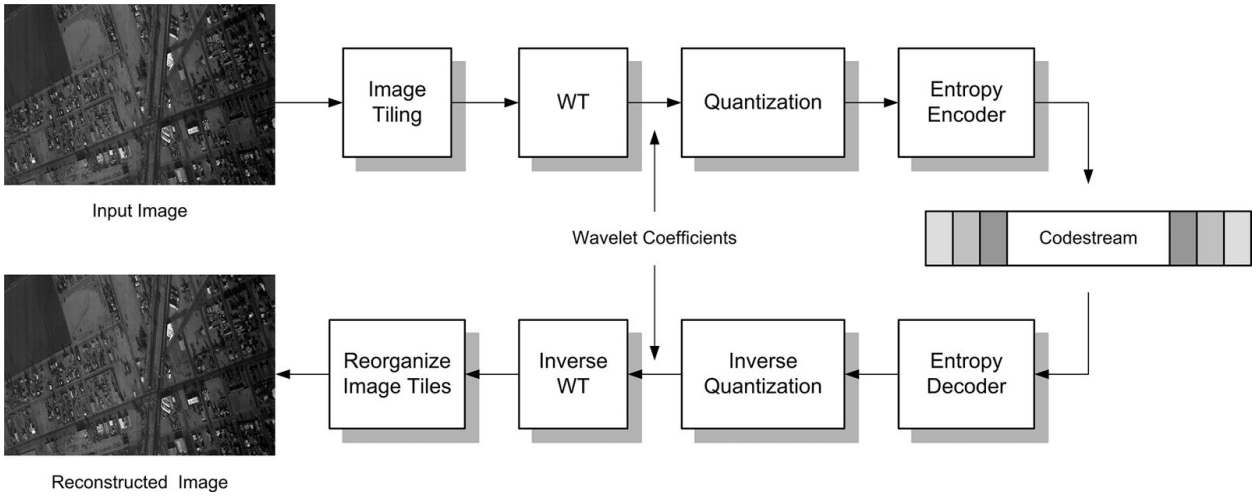


Figure 2: Jpeg 2000 compression process using a more general transform known as a wavelet transform. The Haar transform is the simplest of these transforms; however, it is ill-suited for jpeg 2000 compression, because it has insufficient resolution to detect closely spaced, large differences in the original data. Credit for image, "http://electronicimaging.spiedigitallibrary.org/article.aspx?articleid=1100216".

Figure 3: Two images shown using the different compression methods jpeg and jpeg 2000. The jpeg 2000 compressed images are much higher quality while still saving precious storage space. the difference between jpeg and jpeg 2000 is solely the type of wavelet transform used: mathematically, jpeg 2000 sacrifices the orthogonality of the transform matrix that is used in the jpeg method in favor of a biorthogonal transform matrix. Credit for image, "www.verypdf.com/pdfinfoeditor/jpeg-jpeg2k-1.png".

Once data has been removed or simplified in a compression, or if the original image is simply not of a high data density, that information is forever lost. This lends itself to the label of a ‘lossy’ file format. Lossless image files are often very large in size, making them cumbersome to transmit as quickly, taking up precious storage space, and require more GPU and CPU to edit.

The Haar Wavelet Transformation is a simple variation on how the JPEG compression works. Using a simple 2-length filter one can average near-by pixels in one section of the compressed image, while showing how the image changes in other directions, namely horizontally, vertically, and diagonally. The Haar Transformation can be represented by matrix multiplication of the original image by the following transformation matrix:

$$W_N = \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \sqrt{2}/2 & \sqrt{2}/2 & & 0 & 0 \\ \vdots & & & & \ddots & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \sqrt{2}/2 & \sqrt{2}/2 \\ -\sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & -\sqrt{2}/2 & \sqrt{2}/2 & & 0 & 0 \\ \vdots & & & & \ddots & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -\sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}$$

Figure 3: Haar Wavelet Transformation Matrix

Haar transformations use averages of only two points to provide an average value of a region. This method is not used in practise often due to the less than ideal transformation method places some numbers into irrational numbers, making the rule of thumb for wavelet analysis, to make integers go to integers, utterly ignored. More effective methods have since been implemented.

```
function H = Haar(h)
%%h is a MxN matrix
HN = zeros(length(h(1,:,1)));
HM = zeros(length(h(:,1,1)));
%%Haar MxM matrix
for m = 1:2:length(h(:,1,1))-1
    HM(floor(m/2)+1,m) = sqrt(2)/2;
    HM(floor(m/2)+1,m+1) = sqrt(2)/2;
    HM(floor(m/2)+1+floor(length(h(:,1,1))/2), m) = -sqrt(2)/2;
    HM(floor(m/2)+1+floor(length(h(:,1,1))/2), m+1) = sqrt(2)/2;
end
%%Haar NxN matrix
for n = 1:2:length(h(1,:,1))-1
    HN(floor(n/2)+1,n) = sqrt(2)/2;
    HN(floor(n/2)+1,n+1) = sqrt(2)/2;
    HN(floor(n/2)+1+floor(length(h(1,:,1))/2), n) = -sqrt(2)/2;
    HN(floor(n/2)+1+floor(length(h(1,:,1))/2), n+1) = sqrt(2)/2;
end
HR = h(:,:,1); %Deconstructing the image into its Color Channels
HG = h(:,:,2);
HB = h(:,:,3);
HR = HM*HR*HN'; %Performing the Haar Transform on each Channel
HB = HM*HB*HN';
HG = HM*HG*HN';
H(:,:,1) = HR; %Reconstructing the transformed matrix
H(:,:,2) = HG;
H(:,:,3) = HB;
end
```

Figure 4: Haar Wavelet Transformation Code

The previous code transforms a given image using the formula:

$$B = W_M A W_N^T$$

When applied to an image the following is the result:

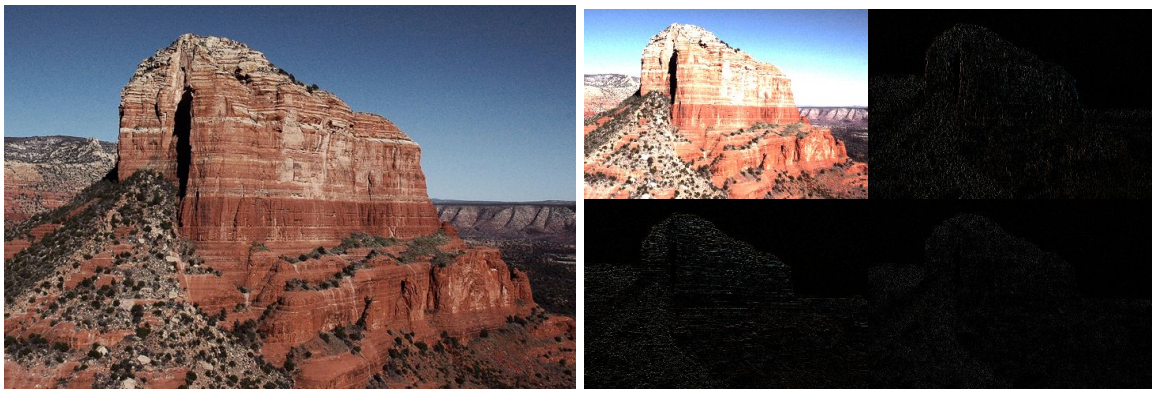


Figure 5: Before and After the Haar Wavelet Transform

Because of the transformation, the top-left corner holds most of the image's information, but in only 25% of the space. The other 75% of the image is much more compressible due to having fewer colors.

The need for compressed files is clear, but with modern innovations one can increase the fidelity of an image's appearance without adding significant bulk to its size. The applications of having resolution enhancements could be easily applied to consumer products, security systems, data clarity on scientific measurements, and software enhancement.

Bilinear Interpolation

MathWorks' MatLab software has numerous toolkits and built in functions to provide analysis to images. The first method implemented in order to provide a scaling of the sample image was done by interpolation. Matlab has many different methods of interpolation itself, but we wrote a bilinear interpolation algorithm ourselves.

Bilinear interpolation is the method used to alter the size of a file by changing the number of pixels used to convey the same information. This is done by taking any region of pixels, choosing the four closest pixels in the original image, and projecting them onto the new size image. The new pixels are calculated from a weighted average of the 4 original pixels based on how close the new pixel is to each original pixel. Using the "floor" and "ceil" functions we can both identify the lowest and highest closest pixels in both x and y directions of the image, as well as calculate the weight of the contribution of each pixel. If a pixel is in-line with the original pixels, either horizontally or vertically, the respective vertical or horizontal pixels of the original image, as the closest pixels horizontally or vertically are much closer than the off-axis pixels of the original.

```
function B = bilinear(img, scale)
[l,w,c]=size(img);
lb= floor(scale*(l-1));
wb = floor(scale*(w-1));
B = zeros(lb, wb, 3);
for x=1:lb
    for y=1:wb
        for z=1:c
            B(x,y,z) = img(floor(x/scale)+1,floor(y/scale)+1,z)*(1-(x/scale - floor(x/scale)))*(1-(y/scale-floor(y/scale))) ...
                + img(ceil(x/scale)+1,ceil(y/scale)+1,z)*(x/scale-floor(x/scale))*(y/scale-floor(y/scale))...
                + img(floor(x/scale)+1,ceil(y/scale)+1,z)*(1-(x/scale-floor(x/scale)))*(y/scale-floor(y/scale))...
                + img(ceil(x/scale)+1,floor(y/scale)+1,z)*(x/scale-floor(x/scale))*(1-(y/scale-floor(y/scale)));
        end
    end
end
```

Figure 6: Bilinear Interpolation/Scaling code

When this bilinear transform is used to to scale the image up the following is the result:

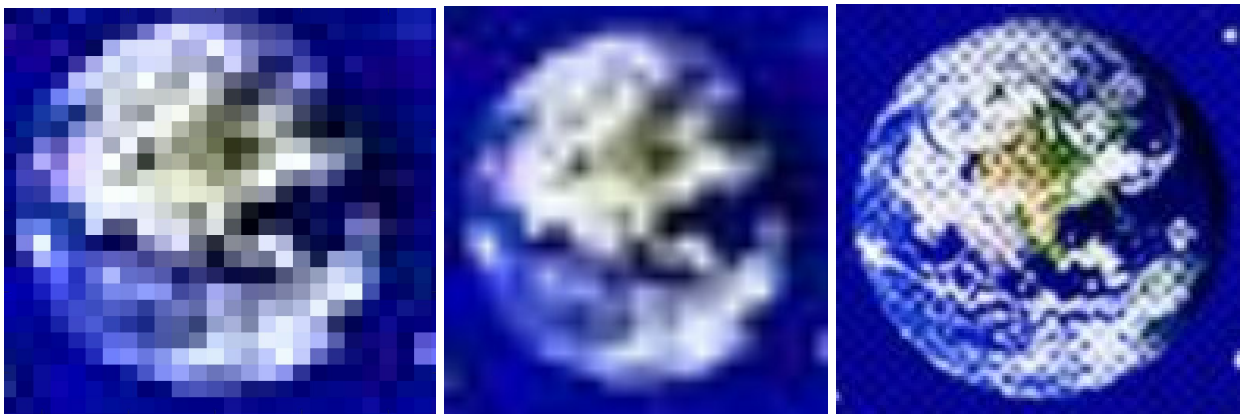


Figure 7: Lo-res(400x400), Bilinear Interpolated x 10 (4000x4000), and Hi-res(2092x2092) images from the same cover. The Hi-res and Lo-res images are original, whereas the Bilinear image is derived from the Lo-res image. Even though it has a higher pixel density than the Hi-res image, it contains less information than the original Hi-res, due to information being lost within the Lo-res image.

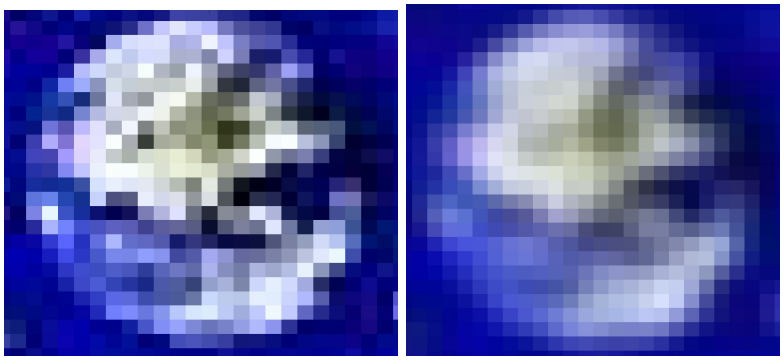
Figure 8: Left: Custom Bilinear function, Right: Matlab Bilinear function

Convolution

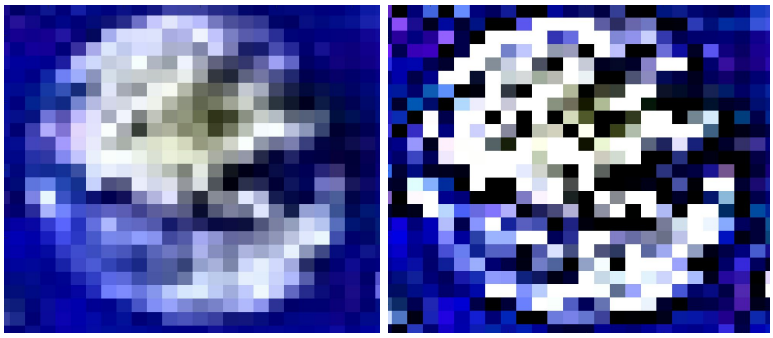
Convolution is defined as the transformation of an image using a kernel, or a matrix that describes how each pixel of interest is given an evaluation similar to a dot product of how the surrounding pixels are going to be used for transformation of a central focal pixel. This particular 'mask' is the convolutional matrix that is arbitrarily defined by the user to yield different results. It can be used to blur images, sharpen corners, provider greater contrast, or provide different functions, such as edge detection, among other things. The following code uses convolution on a given image 'A' and Kernel, or filter, 'f':

```
function F = userfilter(A, f)
[l,w] = size(A(:,:,3));
F(l,w,3) = A(l,w,3);
for x=1:l
    F(x,1,:) = A(x,1,:);
    F(x,end,:) = A(x,end,:);
end
for y=1:w
    F(1,y,:) = A(1,y,:);
    F(end,y,:) = A(1,end,:);
end
F(2:l,2:w,:) = 0;
nx = floor(length(f(1,:))/2);
ny = floor(length(f(:,1))/2);
for x=1+nx:l-nx
    for y=1+ny:w-ny
            for z=1:3
                for x1=1:length(f(1,:))
                    for y1 = 1:length(f(:,1))
                        F(x,y,z)=F(x,y,z)+f(x1,y1)*A(x-nx+x1-1,y-ny+y1-1,z);
                    end
                end
            end
    end
end
end
```

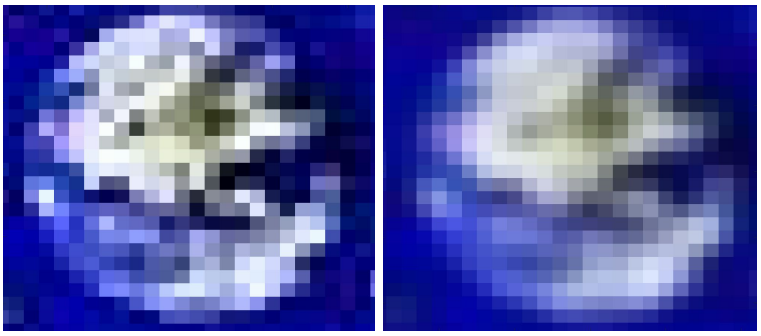
Figure 9: Convolution Code. This code ignores edges where the filter would fall outside of the image.



$$\text{Box Blur} := \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



$$\text{Sharpen} := \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



$$\text{Gaussian Blur} := \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 10: Original and Convoluted images with corresponding Kernal. Low-Res image chosen to exaggerate effect.

These transformations have allowed us to better see how numerical methods are instrumental in the transformation of imagery in the digital realm. Whether you are simply resizing a photograph, 'auto-enhancing' it, putting it into a slide-show and stretched it to fit the frame, your computer is programmed to perform small transformations (often bilinear interpolation) to help the image appear more clearly. Most images we ever see have been manipulated using numerical methods.

Using MatLab's powerful functions and brilliant techniques, an idle curiosity of the details within colourful pixels has been transformed into richer understanding (with less aliasing).

Works Cited

- Dong, Chao, et al. "Learning a deep convolutional network for image super-resolution." *Computer Vision–ECCV 2014*. Springer International Publishing, 2014. 184-199.
- Dong, Chao, et al. "Image Super-Resolution Using Deep Convolutional Networks." *arXiv preprint arXiv:1501.00092* (2014).
- Peleg, Tomer, and Michael Elad. "A statistical prediction model based on sparse representations for single image super-resolution." *Image Processing, IEEE Transactions on* 23.6 (2014): 2569-2582.
- Wang, Zhaowen, et al. "Deeply Improved Sparse Coding for Image Super-Resolution." *arXiv preprint arXiv:1507.08905* (2015).
- JPEG2000, the Next Millenium Compression Standard for Still Images - *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Maryline Charrier, Diego Santa Cruz, Mathias Larsson
- A New Wave in Applied Mathematics - *Science Magazine*, Barry Cipra

- The Fast Wavelet Transform Beyond Fourier Transforms - *Dr. Dobb's Journal of Software Tools*, Mac Cody
- Decomposition of Hardy Functions Into Square Integrable Wavelets of Constant Shape - *SIAM Journal of Mathematical Analysis*, Alexander Grossman and Jean Morlet
- A Method for the Construction of Minimum-Redundancy Codes - *Proceedings of the Institute of Radio Engineers*, David Huffman
- Multiresolution Approximations and Wavelet Orthonormal Bases of $l^2(R)$ - *Transactions of the American Mathematical Society*, Stephane Mallat
- Plotting and Scheming with Wavelets - *Mathematics Magazine*, Colm Mulcahy
- The Discrete Cosine Transform - *SIAM Review*, Gilbert Strang
- Wavelets - *American Scientist*, Gilbert Strang