Numerical Complex Analysis

Image Compression Using Fourier Techniques

SID 430493250

Introduction

The importance of image compression has arisen as a natural consequence of the advancement in modern photography. The current generation of high resolution DSLR cameras capture images containing on the order of 25 million pixels, which in an uncompressed format translates to approximately 75 megabytes of data per image [**5**]. The problem of storage of these images has spawned a number of sophisticated algorithms to compress image files which significantly reduce the size of the stored data, whilst maintaining the quality of the image.

In this report, we will explore the use of Fourier techniques for the purposes of both image compression and image de-noising. Fourier techniques allow us to decompose an image in the frequency domain, remove or alter particular frequencies and then reconstruct the image. This is beneficial for image compression as by removing certain frequencies, the compressed image contains less information. Through judicious choice of which frequencies are removed, the compressed image can incur minimal distortion even though much of the underlying information is discarded. In reference to image de-noising, the ability to remove particular frequencies allows us to remove artefacts from the image, which depending on how the noise is introduced are likely to exhibit differing frequency to the true image.

We will first detail the Joint Photographic Expert Group (JPEG) algorithm, which has been the most widely used lossy image compression algorithm of the last two decades. The term lossy means that the original image is not recoverable after compression in contrast to lossless image compression in which the original image can be recovered. We will investigate the mathematical techniques underlying the JPEG algorithm such as the Discrete Cosine Transform (DCT) and quantization matrix. We will numerically implement the JPEG algorithm in Matlab. For comparison, we will also implement a compression algorithm based on the Fast Fourier Transform using the `fft` command in matlab and we will compare this to the DCT algorithm. Finally, we will use the DCT algorithm to remove artificially introduced noise from an image.

JPEG Compression

Computers store images as multi-dimensional matrices by representing each pixel of an image as an n-tuple. A common format used for representing pixels as n-tuples is YCbCr, which characterises each pixel by by 3 numbers; Y (corresponding to luminance or brightness), Cb (corresponding to blue chrominance) and Cr (corresponding to red chrominance). The intensity of each component is represented by a number in the range [0,255]. Figure 1 illustrates the decomposition of a color image into each of these three channels.

FIGURE 1. From left to right: Original image, luminance component, blue chrominance component, red chrominance component.

The JPEG compression algorithm, which was publicly released in 1992, compresses images of the YCbCr format. The JPEG algorithm compresses images by using the following broad steps on each of the three components of a YCbCr image:

Partition the image. Each of the three components of the image is partitioned into 8×8 blocks of pixels. The entries correspond to the intensities of each of the three components for a particular pixel and are in the range [0,255]. In order to apply the DCT, 128 is subtracted from each entry so the entries are in the range [-128,127]. We denote the resultant matrix M.

Apply the DCT. Each M is transformed using the two dimensional DCT into the coefficient space. The two dimensional DCT is given by

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

where $C(i) = \frac{1}{\sqrt{2}}$ if $i = 0$ and 1 otherwise. This can be expressed as a matrix multiplication using a matrix T with (i, j) entries given by

$$T_{i,j} = \frac{1}{\sqrt{8}} \qquad i = 0$$
$$T_{i,j} = \frac{1}{2}\cos\left(\frac{(2j+1)i\pi}{16}\right) \qquad i > 0.$$

The corresponding DCT of M, which we denote D then amounts to

$$D = TMT'.$$

The entries of D give the frequency of intensity changes of each component of the image. The coefficients in the top left of D correspond to the low frequency intensity changes and the coefficients in the bottom right correspond to the high frequency intensity changes.

Quantization. Quantization is the step where the actual compression takes place. This process involves element by element division of the D matrices by a matrix known as the quantization matrix Q and then rounding the entries. The JPEG standard quantization matrix is

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

This JPEG quantisation matrix has been optimised through numerous experiments on human vision [**2**]. The larger entries in the bottom right corner are a consequence of the fact that the human eye is poor at detecting large variations in colour over short distances (i.e. high frequency changes), so when we divide D by these entries of Q and round off, most term of the high frequency will become zero. An example of the effect of quantization is given below

$$D = \begin{pmatrix} -154 & 58 & 2 & -15 & -1 & 8 & -1 & -6 \\ 1 & 17 & 23 & -4 & -11 & 3 & 5 & 0 \\ 29 & 8 & -4 & -8 & 3 & -6 & 0 & 5 \\ -11 & -10 & 8 & -5 & 2 & -1 & 2 & -6 \\ 2 & -7 & 0 & -2 & 0 & -3 & 0 & -3 \\ -3 & -11 & 2 & 0 & 6 & -2 & 0 & 2 \\ 3 & 0 & 5 & 0 & 2 & -2 & 2 & -3 \\ 2 & -2 & -1 & 0 & 0 & -1 & 0 & 2 \end{pmatrix} \quad C = \text{round}(\frac{D_{ij}}{Q_{ij}}) = \begin{pmatrix} -10 & 5 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Although 55 out of the 64 terms are discarded in quantization, there is minimal loss in perceived quality of the compressed image C, as the frequencies to which the human eye are most sensitive are preserved. The image can be stored in this format and due to the large number of zero entries, the file takes up much less memory by employing a process known as entropy encoding. Entropy encoding essentially records the number of zeros in a succession rather than recording each zero entry [**6**].

A useful property of JPEG compression is that by multiplying the quantization matrix Q by a positive constant q, the degree of compression can be adjusted [**2**]. If Q is multiplied by a factor $q < 1$, when D is divided by Q, less terms will be zeroed leading to less compression and higher quality compressed image. Alternatively, for $q > 1$ more terms will be zeroed by the quantisation matrix leading to a greater degree of compression and a lower quality compressed image. This feature allows for different degrees of compression across different channels, which allows the JPEG algorithm to exploit the fact that the human eye is more sensitive to the luminance channel than the chrominance channels and hence can compress the latter channels more heavily. In our numerical results, we state the compression factors we use for each of the three channels $\mathbf{q} = (q_Y, q_{Cb}, q_{Cr})$.

Decompression. When the compressed image C needs to be decompressed for viewing, firstly we multiply by the quantisation matrix on an element by element basis to obtain reconstructed DCT coefficient matrix R and then perform the inverse DCT on R to obtain the reconstructed image RI. The inverse DCT can be calculated using the T matrix

$$RI = T'RT.$$

Below we compare the matrices associated with the original image I with the reconstructed image RI

$$I = \begin{pmatrix} 124 & 126 & 123 & 111 & 97 & 100 & 102 & 115 \\ 128 & 128 & 128 & 114 & 97 & 98 & 101 & 93 \\ 126 & 121 & 119 & 110 & 100 & 100 & 101 & 96 \\ 113 & 111 & 108 & 108 & 97 & 100 & 100 & 97 \\ 109 & 110 & 108 & 110 & 96 & 101 & 100 & 97 \\ 112 & 110 & 107 & 113 & 98 & 100 & 97 & 102 \\ 110 & 110 & 114 & 119 & 115 & 101 & 103 & 103 \\ 117 & 123 & 124 & 127 & 122 & 107 & 103 & 102 \end{pmatrix} \qquad RI = \begin{pmatrix} 126 & 124 & 117 & 108 & 100 & 98 & 103 & 107 \\ 127 & 125 & 119 & 109 & 101 & 97 & 100 & 104 \\ 124 & 122 & 118 & 109 & 101 & 96 & 98 & 100 \\ 113 & 114 & 112 & 107 & 100 & 96 & 97 & 99 \\ 102 & 105 & 107 & 106 & 101 & 98 & 98 & 99 \\ 100 & 105 & 109 & 109 & 105 & 101 & 99 & 100 \\ 109 & 114 & 119 & 118 & 112 & 105 & 100 & 99 \\ 119 & 124 & 127 & 125 & 117 & 107 & 100 & 98 \end{pmatrix}$$

We note that the average difference between the intensities of reconstructed image and the original image is only 3.02%, even though over 85% of the coefficients were discarded in the quantisation procedure. In the next section, we display the results of our numerical implementation of the above procedure in matlab.

Results

Using the code included in the appendix, we implemented the JPEG algorithm in Matlab. In Figure 2, we show the results of the algorithm for varying choices of $\mathbf{q} = (q_Y, q_{Cb}, q_{Cr})$. In Table 1, we detail the proportion of entries which are discarded in each of the compressions. We note that even after over 95% of the coefficients have been discarded, the compressed image in the top right of Figure 2 is almost indistinguishable from the original image.

FIGURE 2. Original image (top left), compressed image with $\mathbf{q}$=(1,1,1) (top right), $\mathbf{q}$=(1,10,10) (bottom left) and $\mathbf{q}$=(10,20,20) (bottom right).

	Luminance	**Blue Chrominance**	**Red Chrominance**	**Total**
Uncompressed	100%	100%	100%	100%
$\mathbf{q} = (1, 1, 1)$	9.44%	2.38%	2.41%	4.74%
$\mathbf{q} = (1, 10, 10)$	9.44%	1.02%	1.06%	3.84%
$\mathbf{q} = (10, 20, 20)$	1.85%	0.46%	0.52%	0.94%

TABLE 1. Percentage of non-zero elements by channel for varying degrees of compression corresponding to images in Figure 2.

As the length and location of the sequences of zero entries need to be recorded, the reduction in file size of the compressed image is not directly proportional to the reduction in the number of non-zero elements. However, file compression ratios on the order of 15:1 are possible with the JPEG algorithm without noticeable deterioration in image quality [**3**].

1. Fast Fourier Transform Compression

The DCT is the preferred transform used for image compression as it is typically able to incorporate more information from the image in fewer coefficients in comparison to other transforms. This is because the DCT implicitly assumes an even extension of the signal outside the domain from which it is sampled, leading to a continuous extension at the boundaries [**4**]. This continuity means that the DCT converges to the true signal with fewer terms in comparison to transforms like the Discrete Fourier Transform which can have discontinuities at the boundaries. To provide a comparison to the DCT, we implemented a compression algorithm based on the Fast Fourier Transform FFT which is included in the appendix. As illustrated in Figure 3 the quality of the image deteriorates more significantly using the FFT method for the same reduction in coefficients.

FIGURE 3. FFT compressed image (left) and DCT compressed image (right) both with 80% of coefficients discarded.

Experimenting with this algorithm showed the Fourier coefficients in the first row and first column of each 8×8 block were sufficient to characterise the image, but removing any of these coefficients leaded to banded distortions in the compressed image as shown in Figure 4. This limits the degree of compression of the FFT algorithm to a maximum of $\frac{15}{64} \approx 23\%$ before the image experiences severe distortion.

FIGURE 4. Original image (left), compressed image taking only the first row and column of Fourier coefficients (middle) and compressed image taking only the first four entries of the first row and column of Fourier coefficients (right).

2. Image De-noising

Image de-noising using Fourier Techniques is based on the tendency for image artefacts to exhibit different frequency to the rest of the image [**1**]. If for instance the information for one pixel becomes corrupted so the pixel displays the wrong color, this color will also likely be different to the color of the pixels around it, this manifests itself as a high frequency change in the frequency domain. By removing all the frequencies above a certain threshold we can remove these type of artefacts. Optimal image de-noising involves striking a balance between removing enough high frequency coefficients to remove the spurious artefacts, without losing too much of the true image. In Figure 5 we show a noisy image which has had artefacts added by adding random values to the intensities of certain pixels. We compare this image to its de-noised counterpart which has had the high frequency DCT coefficients discarded. We see that by discarding the high frequency components many of the artefacts have been removed resulting in a truer representation of the original image.

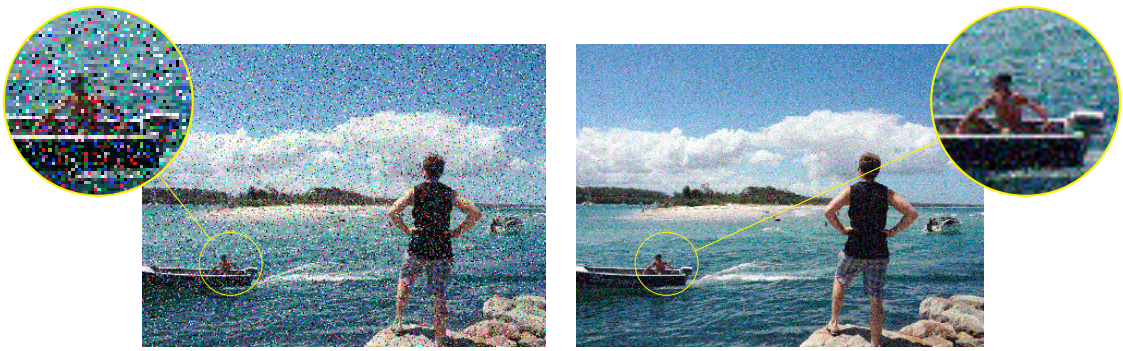

FIGURE 5. Image with artificially introduced noise (left) and de-noised image using DCT (right).

3. Conclusion

Fourier techniques have proven themselves to be invaluable in a wide range of applications as they allow us to decompose complicated objects such as functions, signals or images into simpler components. For the purposes of image compression this allows us to remove redundant information and retain only the most important components of the image. For image de-noising, the decomposition provided by Fourier techniques allow us to separate the true image from the noisy artefacts. More generally, employing similar methods, Fourier techniques can be used for the compression and de-noising of audio, video and other signals.

Appendix

```
%%%%%%%
% Image Compression Using The Discrete Cosine Transform
%%%%%%%
% LIAM FLYNN 2013
%%%%%%%

%% Compression Matricies

% DCT Matrix

T=zeros(8,8);
T(1,:)=ones(1,8)/sqrt(8);

for i=2:8
    for j=1:8                       % Create 8x8 matrix for 2D DCT on 8x8 pixel block
        T(i,j)=sqrt(2/8)*cos((2*(j-1)+1)*(i-1)*pi/16);
    end
end


% Baseline Quantization Matrix

Q=[16,11,10,16,24,40,51,61;                 % JPEG Standard
    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;
    ];



%% Compress Image

% Load Image
%OrigIMG=imread('beach.JPG'); % Load Image
IMG=rgb2ycbcr(OrigIMG);      % Convert from RGB to YCbCr
w=length(IMG(1,:,1));            % Image Width
h=length(IMG(:,1,1));            % Image Height
% Crop Image
h=h-mod(h,8);                   % Dimensions divisible by 8
w=w-mod(w,8);
hmax=h/8;
wmax=w/8;
IMG=IMG(1:h,1:w,:);

% Set YCbCr Qualities
Yq=10                          % Set luminance compression qY 0.0001 (low) to 50 (high)
Cbq=10                           % Set blue chrominance compression qCb 0.0001 (low) to 50 (high)
Crq=10                            % Set red chrominance compression 0.0001 qCr (low) to 50 (high)

Qual=[Yq,Cbq,Crq];

% Initialise Compressed Image
CompIMG=zeros(h,w,3);
nonzero=zeros(2,3);              % Compare Non-zero terms before and after compression

tic
```

```
% Loop for Cr,Y,Cb
for kk=1:3
I=IMG(:,:,kk);                  % Isolate specific Cr,Y,Cb

% Compression
MM=double(I)-128;               % Transform scale from 0-255 to -128-127 to apply DCT
CC=zeros(h,w);
QQ=Q*Qual(kk);                  % Quantization Matrix for specific Cr,Y,Cb

for i=1:hmax
    for j=1:wmax
M=MM(8*i-7:8*i,8*j-7:8*j);       % Split into 8x8 pixel blocks
D=T*M*T';                        % DCT on 8x8 block
%D=fft(M);
C=round(D./QQ);                  % Compress by dividing by Quantization matrix
CC(8*i-7:8*i,8*j-7:8*j)=C;       % Paste 8x8 blocks back together
    end
end

% De-Compression
RR=zeros(h,w);

for i=1:hmax
    for j=1:wmax
R=CC(8*i-7:8*i,8*j-7:8*j).*QQ;   % Multiply by Quantization matrix to decompress
RR(8*i-7:8*i,8*j-7:8*j)=R;       % Paste 8x8 blocks back together
    end
end

% Inverse DCT
for i=1:hmax
    for j=1:wmax
%CompIMG(8*i-7:8*i,8*j-7:8*j,kk)=ifft(RR(8*i-7:8*i,8*j-7:8*j));
CompIMG(8*i-7:8*i,8*j-7:8*j,kk)=T'*RR(8*i-7:8*i,8*j-7:8*j)*T;    % IDCT on 8x8 block
    end
end

CompIMG(:,:,kk)=CompIMG(:,:,kk)+128;          % Transform scale from -128-127 to 0-255

nonzero(1,kk)=nnz(MM);               % Non-zero entries before compresssion
nonzero(2,kk)=nnz(CC);               % Non-zero entries after compresssion

end

CompIMG=real(uint8(CompIMG));
CompIMG=ycbcr2rgb(CompIMG);
toc

% Compare True Image with Compressed
figure(2)
subplot(1,2,1)
imshow(OrigIMG)
title('Original')
subplot(1,2,2)
imshow(CompIMG)
title('Compressed')

nonzero
100*sum(nonzero(2,:))/sum((nonzero(1,:)))

%%
imwrite(CompIMG,'beachC4.jpg','jpeg');
```

```
%%%%%%%
% Image Compression Using The Fast Fourier Transform
%%%%%%%
% LIAM FLYNN 2013
%%%%%%%

%% Compression Matricies

%Quantization Matrix

Q=zeros(8,8);
Q(1:8,1)=ones(1,8);
Q(1,1:8)=ones(1,8);

%% Compress Image

% Loop for Cr,Y,Cb
for kk=1:3
I=IMG(:,:,kk);                  % Isolate specific Cr,Y,Cb

% Compression
MM=double(I)-128;               % Transform scale from 0-255 to -128-127 to apply DCT
CC=zeros(h,w);
QQ=Q;

for i=1:hmax
    for j=1:wmax
M=MM(8*i-7:8*i,8*j-7:8*j);      % Split into 8x8 pixel blocks
D=fft(M);
C=round(D.*QQ);
CC(8*i-7:8*i,8*j-7:8*j)=C;      % Paste 8x8 blocks back together
    end
end

% Inverse FFT
for i=1:hmax
    for j=1:wmax
CompIMG(8*i-7:8*i,8*j-7:8*j,kk)=ifft(CC(8*i-7:8*i,8*j-7:8*j));
   end
end

CompIMG(:,:,kk)=CompIMG(:,:,kk)+128;          % Transform scale from -128-127 to 0-255



end



%%%%%%%
% Image De-noising
%%%%%%%
% LIAM FLYNN 2013
%%%%%%%

%Quantization Matrix

Q=zeros(8,8);
Q(1:6,1:6)=ones(6,6);       % Discard coefficients 6,7,8 in each direction


%% Denoise Image

% Initialise Denoised Image
```

```
CompIMG=zeros(h,w,3);
NoiseIMG=OrigIMG;                  % Initialise Noisy Image
tic

% Loop for Cr,Y,Cb
for kk=1:3
I=IMG(:,:,kk);                  % Isolate specific Cr,Y,Cb

MM=double(I)-128;               % Transform scale from 0-255 to -128-127 to apply DCT
MM=MM+randint(h,w,[-150,150]).*binornd(1,0.2,h,w);      % Add artifical Noise
CC=zeros(h,w);

for i=1:hmax
    for j=1:wmax
M=MM(8*i-7:8*i,8*j-7:8*j);       % Split into 8x8 pixel blocks
D=T*M*T';                        % DCT on 8x8 block
C=round(D.*Q);                  % Compress by dividing by Quantization matrix
CC(8*i-7:8*i,8*j-7:8*j)=C;       % Paste 8x8 blocks back together
    end
end

% Inverse DCT
for i=1:hmax
    for j=1:wmax
CompIMG(8*i-7:8*i,8*j-7:8*j,kk)=T'*CC(8*i-7:8*i,8*j-7:8*j)*T;     % IDCT on 8x8 block
    end
end

CompIMG(:,:,kk)=CompIMG(:,:,kk)+128;          % Transform scale from -128-127 to 0-255

NoiseIMG(:,:,kk)=MM+128;

end
```

Bibliography

[1] A. Buades, B. Coll and J.M. Morel, A review of image denoising algorithms, with a new one, *SIAM - Multiscale Modeling & Simulation*, **4** No.2 490-530, (2005)

[2] S. Cabeen and P. Gent, Image Compression and the Discrete Cosine Transform, *College of the Redwoods*, (2011)

[3] B. Furht, A Survey of Multimedia Compression Techniques and Standards. Part I: {JPEG} Standard, *Real-Time Imaging*, **1** No.1, 49 - 67 (1995)

[4] A.K. Jain, Fundamentals of Digital Image Processing, *Prentice Hall*, (1989)

[5] A. Murabayashi, 2008, *Uncompressed Image File Size*, PhotoShelter, http://blog.photoshelter.com/2008/06/uncompressed-image-file-size

[6] J.H. Wilkinson, Apparatus for compressing image data employing entropy encoding of data scanned from a plurality of spatial frequency bands, *Google Patents*, US Patent 5,537,493, (1996)