University of Cambridge

Digital Signal Processing L314

Assignment 4

Identification of JPEG compression history

Author:
Naunidh Dua

CRSID:
nsd30

17th January 2023

Contents

1 Introduction

1.1 Background

This project aims to analyse of a set of 3024 x 4032 images, each of which has been passed through a JPEG compression pipeline 0, 1 or 2 times. The set of images is comprised of 3 distinct scenes, each of which is present as both a grayscale and a colour image. The images are given in the lossless PNG format, with the task being to determine how many times each image went through the pipeline. These images can be seen in the Appendices, along with an example of a raw image that was used for testing.

JPEG images are extremely widespread in use, with some sources estimating that several billion JPEG images are produced daily[1]. JPEG compression is lossy, and has very good performance, achieving high compression rates with little change to visual quality. Despite other compression algorithms achieving far superior compression performance (in both visual quality and compression rate), JPEG still remains the dominant standard, even over its intended successor, JPEG 2000[2]. Gaining information about the history of an image's compression can be helpful in forensic analysis, potentially leaking information about the tools used to generate the image, as well as whether or not the image has been doctored since last compression.

1.2 Tools

All analysis was performed using the Julia programming language, using the following libraries: `ImageIO.jl, FileIO.jl, ImageShow.jl, Colors.jl, Plots.jl, FFTW.jl, StatsBase.jl, LinearAlgebra.jl, IJulia.jl`. Initially analysis was conducted using `Pluto.jl`, however, using the more standard Jupyter notebook resulted in better performance and ease of use, which is available at `https://github.com/NaunidhDua/jpeg_ch_id`. For testing, I passed raw DNG images through different processing pipelines, using the tools `pnmtopng, cjpeg, djpeg` and `dcraw`. I attempted to use `ImageMagick`, however I encountered dependencies on `ufraw`, which appears to be no longer maintained, and the alternative, `ufraw-batch` is no longer included in Debian Bullseye.

1.3 JPEG Theory

JPEG compression has a 5 stage pipeline, of which only the last three are used for grayscale images. The 5 stages are:

1. Colour Space Transformation
2. Chrominance down-sampling
3. Discrete Cosine Transform (DCT) conversion

4. Quantisation
5. Lossless compression via Run Length Encoding and Huffman encoding

The lossy stages of the compression are down-sampling and quantisation. In section 2, I will examine the effects quantisation has on compressed grayscale images. This work was pioneered by Fan *et al.* in 2003, and extended to colour images by Neelamani *et al.* in 2006 [3, 4].

If an image is not white noise, (ie there is a correlation between samples in the time domain), then pixels in the image contain non-zero mutual information. This means there is redundancy in the image, and there must exist a way to compress the data losslessly. Any transform that converts the image into the frequency domain, exposes this correlation. The ideal transform to use is the Karhunen–Loève transform, (KLT). As the compressor must work for arbitrary image sizes, a blockwise approach is logical.

JPEG compressors split the image into 8x8 blocks, and perform a 2D DCT on each of them (after shifting the pixel values to be centred at 0). There are a few reasons compressors use the DCT. The first is that the KLT is computationally expensive, and the DCT is a close approximation of the KLT (in fact it asymptotically equivalent)[5, 6]. In addition, the DCT maps from $\mathbb{R} \mapsto \mathbb{R}$, whereas the FFT maps from $\mathbb{R} \mapsto \mathbb{C}$. This is preferable as it reduces the amount of storage needed by half. Furthermore, the FFT has cyclic repetition continuation, and the DST has odd continuation across blocks, which add add (more significant) blocking artefacts than the even continuation of the DCT.

For 'real-world' images, the aforementioned transforms are 'energy-compacting'; that is, a large proportion of the signal energy is contained in a small number of values. The values of the resulting transformed image are often called coefficients, as they correspond to coefficients of the Cosines. 'Real-world' images tend to be 'pinkish', ie their energy is mostly contained in the lower frequencies. Therefore the lower frequency coefficients tend to be significantly larger that the higher frequency coefficients, by orders of magnitude. Figure 1 demonstrates this.

A naïve approach would be to throw away the highest frequencies and assume that they are zero in the decoder. However, JPEG compressors instead divide the transformed block by a quantisation matrix, and round the resulting values to integers. This quantisation matrix (generally) contains higher values in the bottom right, and lower values in the top left. This divides the higher frequency coefficients by larger values, thus more likely making them 0 (and also more likely to be the same low value). This effect is exploited by the run-length encoding and Huffman encoding steps, making each of them much more effective.

The JPEG standard defines a Quality Factor (QF) range from 1 to 100, where 100 has the least compression. The standard gives a quantisation matrix for QF 50, and a method to generate the matrices for other QFs, as shown in figure 2 and algorithm 1,

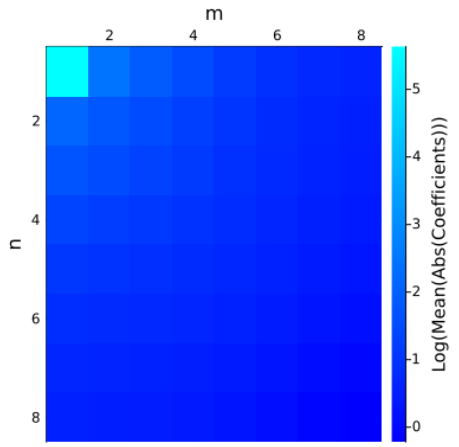


Figure 1: *A plot showing the energy compacting property of the DCT. A blockwise DCT has been applied to a raw image, and mean absolute value of each coefficient is plotted.*

however third party implementations of the JPEG compression algorithm (such as by digital camera manufacturers) often use their own quantisation matrices, and these are kept as trade secrets.

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

Figure 2: *The standard quantisation matrix for QF 50*

Algorithm 1 qMatrix(QF)

Require: MAT50, the standard quantisation matrix 2, QF, the quantisation level wanted.
Ensure: MATQF, the quantisation matrix for QF.
  if $QF < 50$ **then**
    $S = 5000/QF$
  else
    $200 - 2 * QF$
  end if
  MATQF $\leftarrow$ floor.$((S.\text{MAT50.} + 50)/100)$
  MATQF $\leftarrow$ [if i ! $= 0$ i else for i in MATQF] ▷ Prevents divide by 0.
  return MATQF

At the decoding stage, the lossless run-length and Huffman encoding steps are decoded. These values are then multiplied by the quantisation table. NB: Frequently, the quantisation matrix used is stored in the header of the image itself, to allow for decoding. Since the values were rounded to the nearest integer in the encoding stage, the restored coefficients are all multiples of their respective element in the quantisation table. This is the basis from which I will conduct my analysis.

2 Compression History Estimation

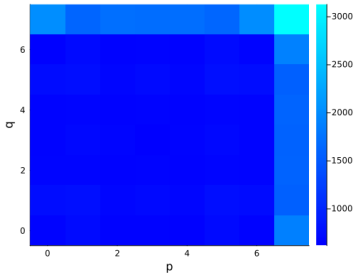
2.1 Identification of (potential) grid origin

If an image has been cropped, scaled by arbitrary factors or edited in general, it is possible that the alignment of the block grid does not match perfectly with the image itself. Fan *et al.* provided the following formula to assist with determining the grid origin:

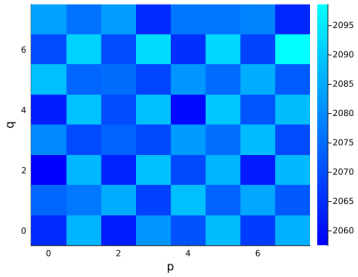
$$E(p,q) = \sum_i \sum_j |\text{IMG}[8i+p, 8j+q] - \text{IMG}[8i+p, 8j+q+1] - \text{IMG}[8i+p+1, 8j+q] + \text{IMG}[8i+p+1, 8j+q+1]| \quad (1)$$

where IMG is the image, and IMG[x,y] is the pixel at the x-th row and y-th column. The summation is over all possible block boundaries for a given alignment, (p,q).

If an image is compressed, E tends to be mostly 0 for many p and q, with one peak for the pair (p,q) that represents the bottom right most corner of the block, and decreasing values around this peak, as shown in figure 3a. As the peak is at the point (7,7), the grid is aligned with the image.



(a) *Compressed with QF 75.*

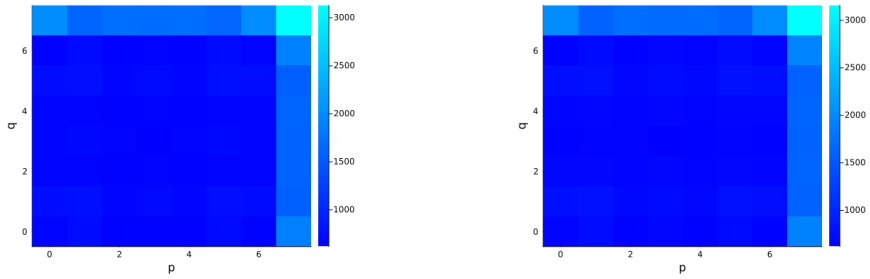


(b) *Uncompressed.*

Figure 3: *A heatmap of E for different images.*

For uncompressed images, the values of E tend to be very similar differing by about 1%-10%, as shown in figure 3b. Whilst equation 1 was not created with the intention

to identify whether or not an image has been compressed, it is clear that it can be used to do so. This idea is developed in the next subsection.



(a) *Compressed with QF 75 then 75 again.*

(b) *Compressed with QF 75 then 90.*

Figure 4: *A heatmap of E for different images.*

Figures 4a and 4b demonstrate that there is no effect of multiple compressions on the grid alignment as no editing has taken place between compressions. In addition there is no effect of QF choice on grid alignment.

2.2 Identification of compression

The first question to answer is whether or not the image has been compressed. As the JPEG compressor works blockwise, discontinuities across blocks are produced. Fan *et al.* introduced the following method:

Algorithm 2 get_Z'

Require: $0 \leq p, q \leq 7$, the grid origin location; img, the image to be analysed
Ensure: out, an array consisting of all the Z' values.

```
out ← []
for i = p:8:height(image)-8 do
    for j = p:8:width(image)-8 do
        t = |(img[i+3,j+3]-img[i+3,j+4]-img[i+4,j+3]+img[i+4,j+4])|
        append!(out,t)
    end for
end for
return out
```

Algorithms 2 and 3 are used to calculate the Z' and Z'' values of the image respectively, also called the "neighbour differences". These are defined as follows:

- Z' is a measure of the similarity of the pixel values in the middle of a block, the neighbour differences within a block.

Algorithm 3 get_Z''

Require: $0 \leq p, q \leq 7$, the grid origin location; img, the image to be analysed
Ensure: out, an array consisting of all the Z'' values.

```
out ← []
for i = p:8:height(image)-8 do
    for j = p:8:width(image)-8 do
        t = |(img[i+7,j+7]-img[i+7,j+8]-img[i+8,j+7]+img[i+8,j+8])|
        append!(out,t)
    end for
end for
return out
```

- Z'' is a measure of the similarity of the pixel values between blocks, the neighbour differences across a block boundary.

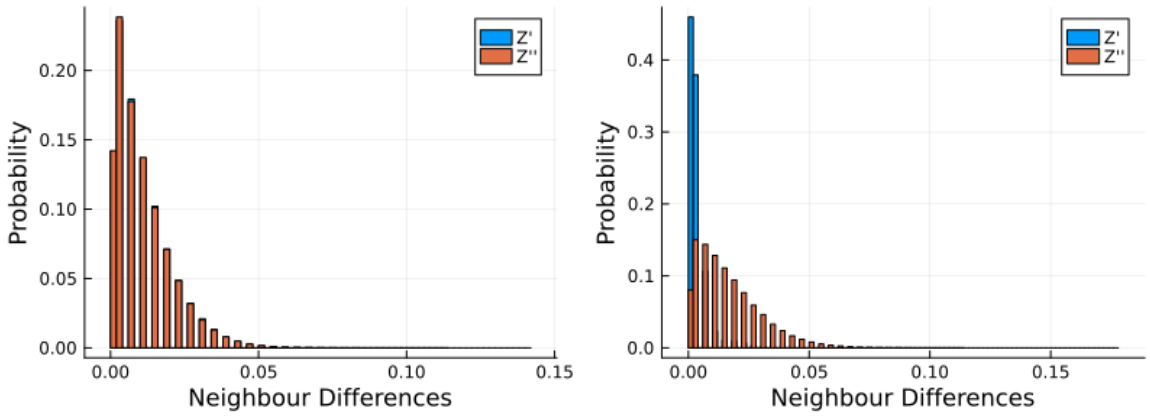
The values of Z' and Z'' are calculated for all blocks in the image.

$$K = |H(\text{get\_}Z'(\text{img}, p, q)) - H(\text{get\_}Z''(\text{img}, p, q))| \tag{2}$$

1. Determine the grid origin point, as described in the previous subsection.
2. Run 2 and 3 on the image with the grid origin determined.
3. Plot a histogram of the two arrays. An uncompressed image will have nearly identical plots, whilst a compressed image will differ.
4. Plot a graph of the absolute difference between the two histogram weights.
5. Calculate K, as in equation 2, the absolute difference between histogram weights. This provides a quantitative description of the similarity between the two histograms.

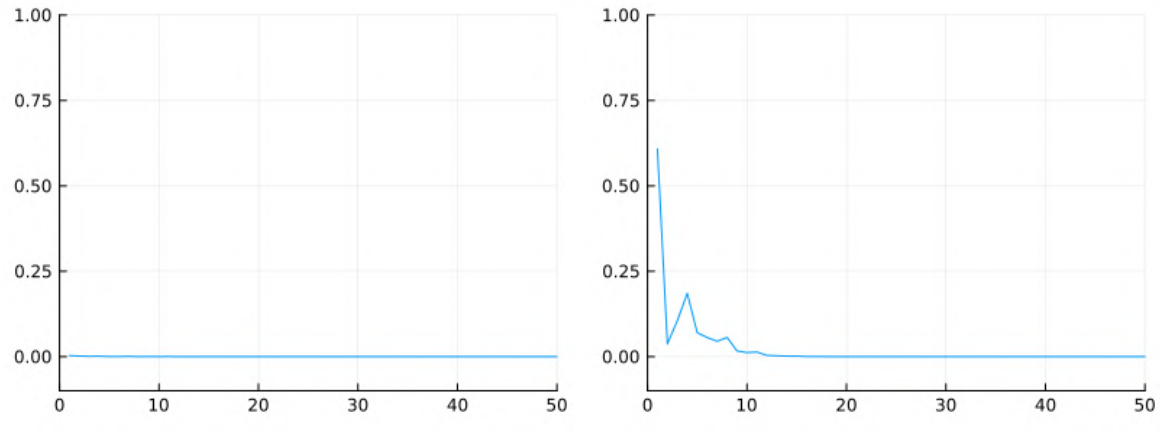
I was able to replicate the results of Fan *et al.* as shown in figure 5. The histograms of figure 5a are nearly identical, whilst the histograms of figure 5b are very different. The difference between the two histograms is shown in figure 6. The value of K is 0.00956 for the uncompressed image and 1.22 for the compressed image. Fan *et al.* suggested a threshold of 0.25 for K to determine whether or not an image has been compressed. This threshold works well for the images tested in this project as well.

Fan *et al.* did not suggest a reason for why the shape of the histograms are as such. I hypothesise the following: For an uncompressed image, the Z' and Z'' values will be similar, as the pixel values in the middle of a block will be similar to the pixel values across a block boundary. The envelope of two histograms appears to follow a Laplacian or exponential distribution (excluding the first bar). For a compressed image, the envelope of the Z' values still appears follow a (modified) Laplacian distribution, but for the Z'' values the envelope of the histogram appears to follow a Gaussian



(a) *Uncompressed.* **(b)** *Compressed with QF 75.*

Figure 5: *Histograms of Z' and Z'' for different images.*



(a) *Uncompressed.* **(b)** *Compressed with QF 75.*

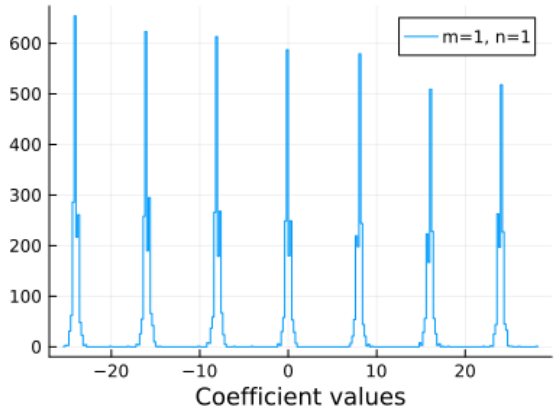
Figure 6: *Absolute difference between Z' and Z'' histograms for different images.*

distribution. This could be because blocking artefacts introduced by the compression themselves follow a Gaussian distribution.

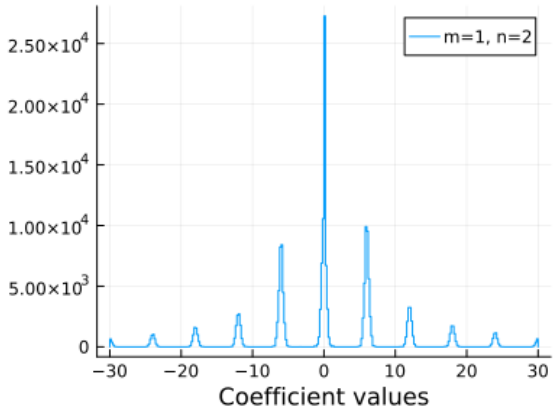
2.3 Multiple Compression Analysis

As mentioned in 1.3, after applying the DCT to the image, the coefficients are multiples of the elements in quantisation matrix. Whilst each block is not a multiple of the matrix, for a specific coefficient, the distribution of the coefficient will be spaced peaks, with the peaks being multiples of the element in that position of the quantisation matrix. For example, the quantisation matrix for QF 75 has a value of 6 at position [1,2], so the distribution of the coefficients at position [1,2] will have peaks at multiples of 6 (using

1-indexing). [1] This is shown in figure 7b.

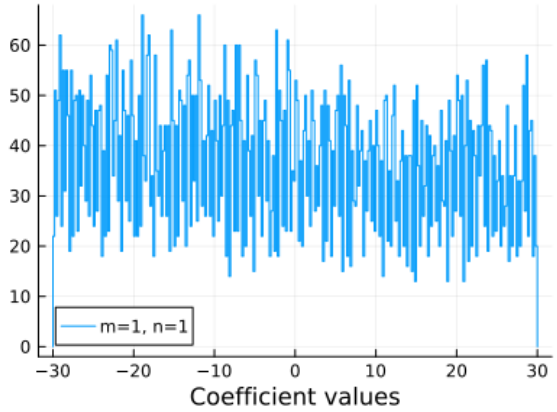


(a) *Coefficients at position [1,1].*

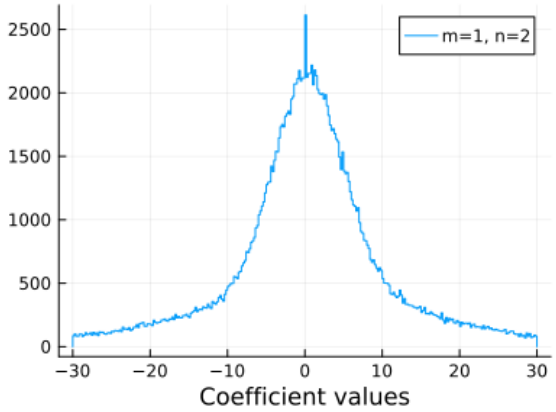


(b) *Coefficients at position [1,2].*

Figure 7: *Histograms of the coefficients at position [1,1] and [1,2] for an image compressed with QF 75.*



(a) *Coefficients at position [1,1].*



(b) *Coefficients at position [1,2].*

Figure 8: *Histograms of the coefficients at position [1,1] and [1,2] for an uncompressed image.*

Figure 8 shows the histograms of the coefficients at position [1,1] and [1,2] for an uncompressed image. The DC component (position [1,1]) tends to be uniformly distributed in real world images, and this is reflected in figures 8a and 7a. The AC components (including position [1,2]) tend to be Gaussian, and this is reflected in figures 8b and 7b. The effects of quantisation is evident in figures 7a and 7b, as the peaks are spaced by multiples of the quantisation matrix element; 8 and 6 respectively. The resulting graphs

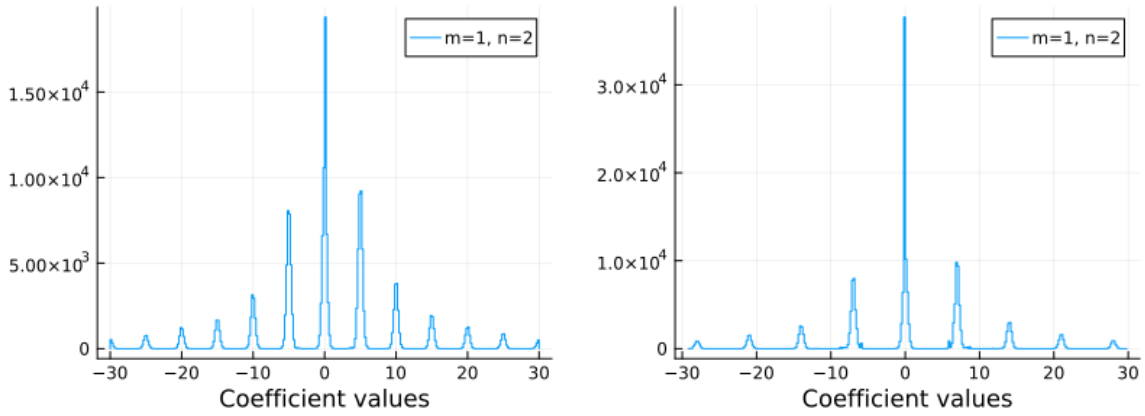
[1]The quantisation matrix is not necessarily symmetric, so the distribution of the coefficients at position [1,2] will not be the same as the distribution of the coefficients at position [2,1].

are essentially rounding the Uniform/Gaussian distribution to the nearest multiple of the quantisation matrix element. The peaks are not perfect peaks, there is some spread in each peak. Fan *et al.* suggest that this spread is due to two reasons:

1. The pixel values generated by the IDCT in the decoder rounds the real values to the nearest integer.
2. Pixel values greater than 255 or less than 0 are clipped to 255 or 0 respectively.

Fan *et al.* investigate the spread of the peaks and introduce distributions to model the spread, in order to generate an MLE for the quantisation matrix. The aim of this project however, is to determine the number of JPEG compressions that an image has been through.

If an image has been compressed multiple times with different QFs, the quantisation matrix will be different for each compression. Consider the first pass of compression, with a QF of 77. The quantisation matrix has a value of 5 at position [1,2], so coefficients at position [1,2] will be multiples of 5. The second pass of compression, with a QF of 70, will have a quantisation matrix with a value of 7 at position [1,2], so coefficients at position [1,2] will be multiples of 7. However, the distribution of coefficients being passed through this second quantisation filter is no longer a Gaussian, but a spaced peaked Gaussian, as in figure 9a.



(a) *QF 77.* **(b)** *QF 70.*

Figure 9: *Histograms of the coefficients at position [1,2] for different QFs.*

Consider the peaks at 5 and 10. Both of these peaks have the same nearest multiple of 7, which is 7. However 14 only has 1 nearest multiple of 7, which is 15. Thus the distribution of the coefficients at position [1,2] after the second pass of compression will not be a spaced peaked Gaussian, but one with inflated peaks at multiples of 7 which have more multiples of 5 near to them. This is shown in figure 10. The envelope of the peaks no longer follows a Gaussian distribution, and the peaks at 7 and 28 are inflated, as they have more multiples of 5 near to them.

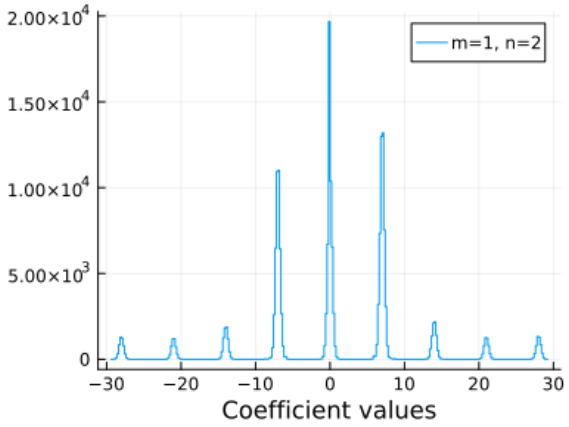


Figure 10: *Histogram of the coefficients at position [1,2] after two passes of compression, QF 77 then QF70.*

2.4 Results

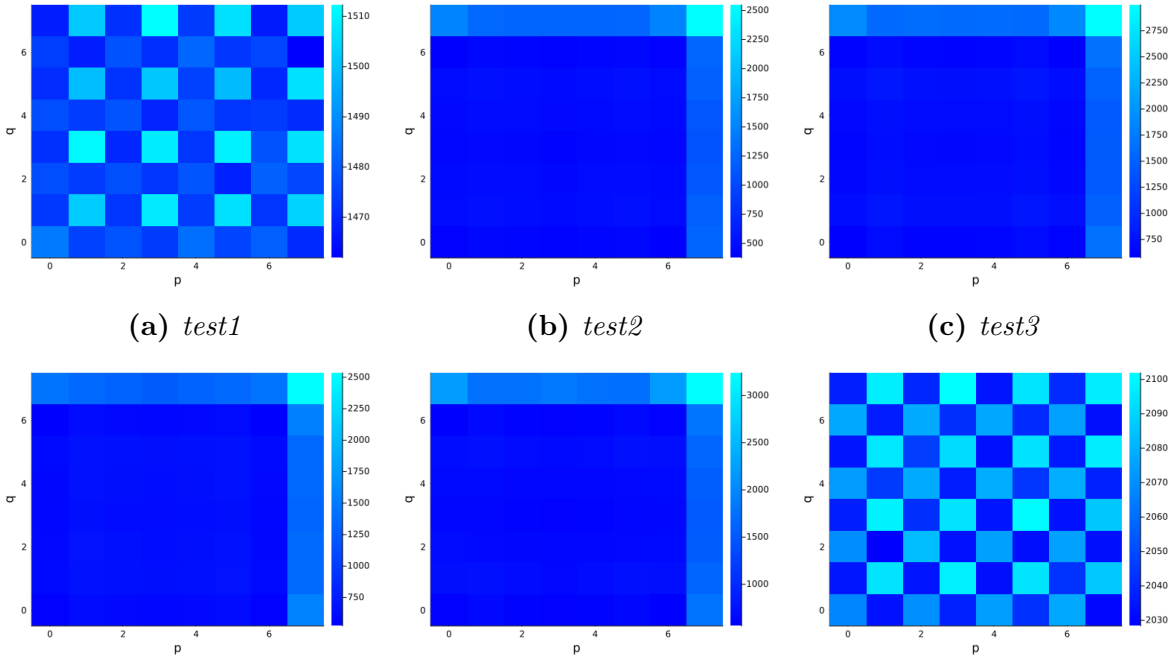
Passing the test images (see A) through the above steps results in the figures 11, 12 and 13. For colour images, one option would be to reverse the colour transformation done by the decoder, and analyse each of the Y,Cb,Cr channels independently. I converted the colour images to grayscale, as this performs the same task, but only selects the Y (Luminance) channel.

Figures 11 and 12a show clearly that images test1 and test3c are uncompressed, and the rest have been compressed with no change to the grid origin. Using the k=0.25 threshold, I conclude that test1 and test3c are uncompressed while the rest are compressed. From figure 13, I conclude that test3 has been compressed twice, while test1c, test2 and test2c appear to be compressed once.

2.5 Conclusion

Using multiple techniques introduced by Fan *et al.* , I was able to effectively determine the number of compressions a set of images went through. This was done through detecting evidence of blocking, analysis of neighbour differences in different regions of an image, and analysis of the distribution of DCT coefficients. I was able to conclude that images test1 and test3c are uncompressed, test3 has been compressed twice and test1c, test2 and test2c appear to be compressed once.

I particularly enjoyed the challenges of executing this project; the techniques and maths that I learnt both doing the project and throughout the whole L314 course were exciting and stimulating, and content that I have wanted to learn for many years.



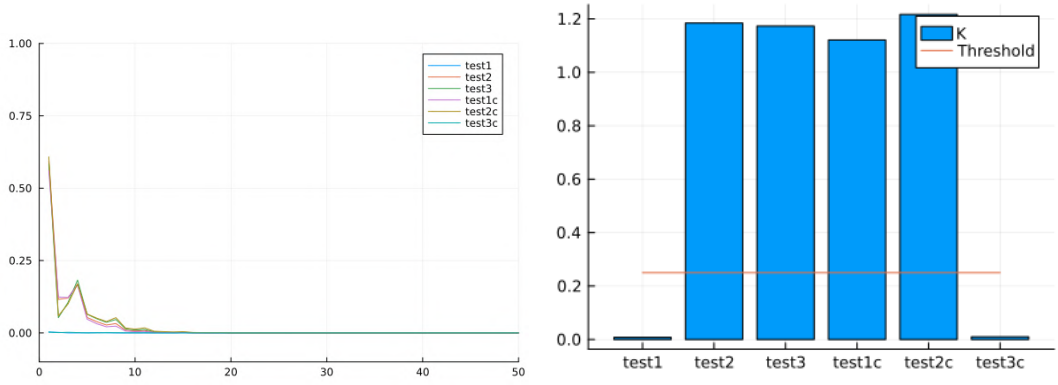
(a) *test1* **(b)** *test2* **(c)** *test3*

(d) *test1c* **(e)** *test2c* **(f)** *test3c*

Figure 11: *A heatmap of E for the test images.*

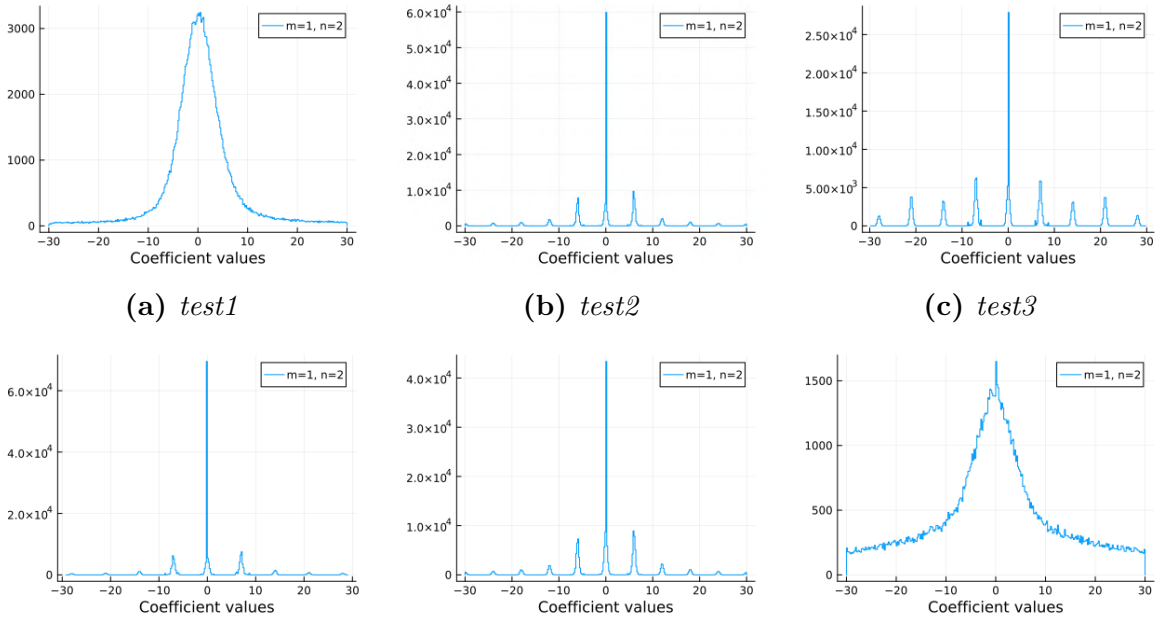
2.5.1 Further Work

Potential next steps include quantitatively determining the size of the peaks for the DCT coefficients. This would allow for two developments, easier determination of quantisation matrix elements (which I did by eye) and the ability to quantitatively estimate the similarity between the envelope of the peaks and a Gaussian. These developments would be useful for exact JPEG re-compression, which has also been studied by Lewis *et al.* [7]. Lewis *et al.* investigated exactly recompressing JPEG images than have been compressed once, this could potentially be extended to multiple compressions.



(a) *Absolute difference between Z' and Z'' histograms for the test images.* **(b)** *Chart showing the K for each test image.*

Figure 12



(a) *test1* **(b)** *test2* **(c)** *test3*

(d) *test1c* **(e)** *test2c* **(f)** *test3c*

Figure 13: *A histogram of the coefficients at position [1,2] for the test images.*

References

[1] Chris Baraniuk. *JPEG lockdown: Restriction Options sought by committee*. Oct. 2015. URL: https://www.bbc.co.uk/news/technology-34538705.

[2] Brad Kelechava. *Why JPEG 2000 never took off*. Jan. 2022. URL: https://blog.ansi.org/2018/07/why-jpeg-2000-never-used-standard-iso-iec/.

[3] Zhigang Fan and Ricardo L De Queiroz. "Identification of bitmap compression history: JPEG detection and quantizer estimation". In: *IEEE Transactions on Image Processing* 12.2 (2003), pp. 230–235.

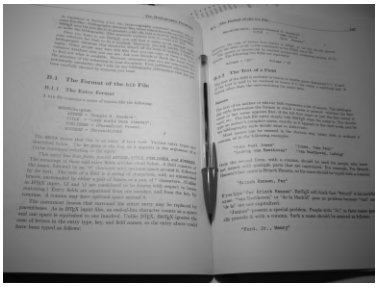
[4] Ramesh Neelamani et al. "JPEG compression history estimation for color images". In: *IEEE Transactions on Image Processing* 15.6 (2006), pp. 1365–1378.

[5] Nasir Ahmed, T Natarajan and Kamisetty R Rao. "Discrete cosine transform". In: *IEEE transactions on Computers* 100.1 (1974), pp. 90–93.

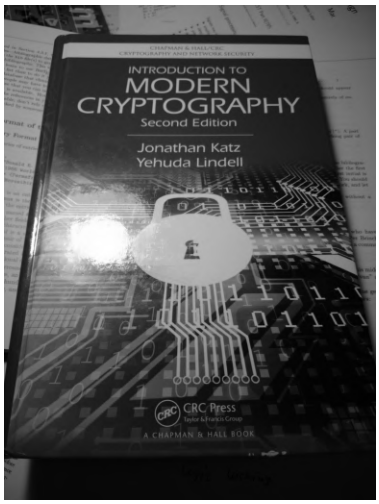
[6] K.S. Shanmugam. "Comments on "Discrete Cosine Transform"". In: *IEEE Transactions on Computers* C-24.7 (1975), pp. 759–759. DOI: 10.1109/T-C.1975.224301.

[7] Andrew B Lewis and Markus G Kuhn. "Exact JPEG recompression". In: *Visual Information Processing and Communication*. Vol. 7543. SPIE. 2010, pp. 256–264.

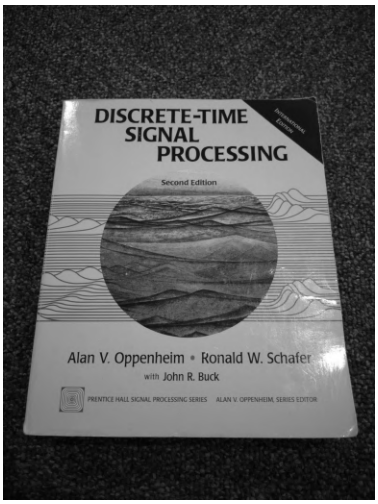
Appendix A Test Images



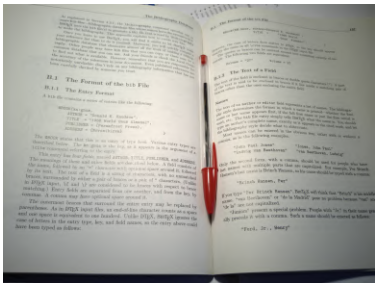
(a) *test1*



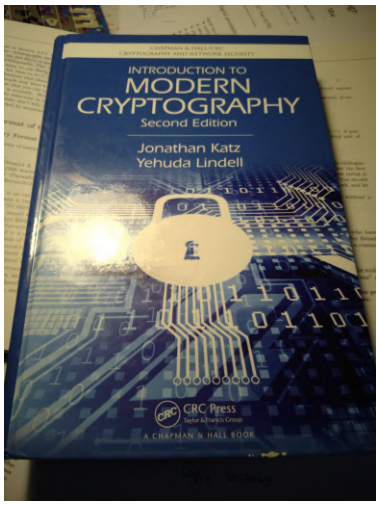
(b) *test2*



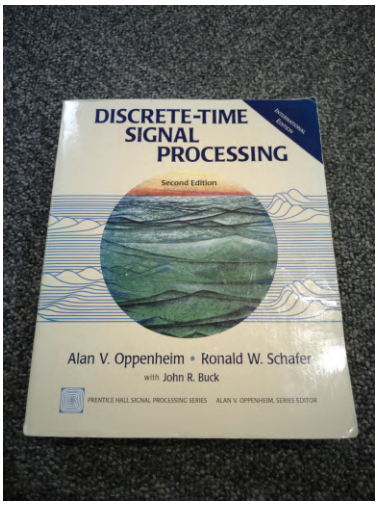
(c) *test3*



(d) *test1c*



(e) *test2c*



(f) *test3c*

Figure 14: *The test images analysed in this project.*

Appendix B Raw Image Example

Figure 15: *An example of the raw images used for testing.*