IMPLEMENTATION OF A LOSSY JPG-INSPIRED COMPRESSION ALGORITHM IN PYTHON

XF012484

Abstract

This report will explore the theory of digital image compression and implement both lossy and lossless compression in a jpg-inspired algorithm. The algorithm is written in Python 3.7 with the code provided in the appendix. The report attempts to inform the reader of the abstract principles of jpeg compression with some examples. The author's compression algorithm is then run on a number of sample images with comparing compression ratio, computation time and a selection of subjective and objective fidelity tests. Further research is explored in the summary that investigates compression using an architecture of convolutional neural networks.

Chapter 1

Introduction

Introduction to compression

In the age of information, storage space has become cheap and abundant due to exponential increases in memory storage and the cost of manufacturing electronic components. This has lead to a disregard for efficient memory storage in some areas of Computer Science. However, some aspects are still restricted by limited memory space, for instance, exchanging information over a network which both decreases cost and latency. Data compression can be used in order to reduce the number of bits it takes to represent the data. The two methods for implementing compression are Lossy and Lossless compression. Lossy implementations lose information during quantisation and attempts to construct a representation that is approximately similar in appearance. Lossless compression attempts to reduce the number of bits required to represent the data without any loss in information.[1]

Introduction to jpg compression

Jpg is a lossless image compression technique created by that is superset of a number of compression standards but by far the most popular for digital image compression is the JFIF standard. Jpg works by exploiting the psychovisual redundancy of the Human Visual System (HVS). Jpg is an extremely efficient (in terms of compression ratio) classical approach to image compression and is still one of the most widely used. While jpg has since been out performed by many other algorithms it is still one of the most influenzal compression techniques published. This is evident as the frequency transformation is used by a number of alternatives.

Chapter 2

Development

2.1 jpg compression

2.2 Encoding

Firstly, the input RGB bitmap is loaded into an array with the same dimensions as the image. A number of preprocessing operation are performed on the image array before any jpeg compression can occur.

2.3 Image preprocessing

RGB

An RGB bitmap is a data representation of an image with three colour components R (Red), G (Green) and B (Blue) where each pixel comprises of 24 bits, 8 bits for each colour value. A combination of the three values represent the final colour in the image. 8 bits representing each colour gives 256 shades of each, creating 16.77 million possible combinations, far more than the human eye can perceive[1]. RGB is just one of many colour models that have the ability to represent an image.

YCbCr

YCbCr is another colour model used to hold image information. YCbCr signals are typically created from an RGB gamut so are therefore the RGB footprint will always remain. Unlike RGB, YCbCr is separated in components Y, Cb and Cr. Y represents the luminance of the image and Cb and Cr represent the blue and red chrominance values respectively, they also characterise the hue and saturation. The purpose of this colour model is to separate the intensity (Y) from the colour (CbCr). The amount of image data is exactly the same between RGB and YCbCr. Images that use YCbCr standard are known as JFIF which is subset of JEG.[1]

Converting an image from RGB to the YCbCr colour model

Three component colour coordinates systems have been created to find the optimal Y, Cb

and Cr representations of an RGB bitmap. To transform an RGB bitmap to the YCbCr colour model the dot product is performed between the YCbCr colour coordinates and corresponding RGB colour values then adding 128 to both the chrominance components, like the following:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.334 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

Chrominance reduction

Because the human visual system evolved to see in low light levels (winter/night) it became very sensitive to luminance. Hue and saturation impact the perception of the image much less than the luminance. Because of this, the human eye is much better at detecting gray scale than it is colour. Jpg exploits this aspect of the human visual system by either sub-sampling or down-scaling the chrominance components within the image.[5]
There are a number of approaches to reducing the chrominance within an image to aid in image compression

Sub-sampling chrominance

Sub-sampling is used to reduce the amount of data that is associated in the image. Generally only the chrominance value are sub-sampled because they include the most redundant information for the human visual system. To perform sub-sampling on the chrominance components a sampling ratio must be selected. Jpg uses a sampling ratio of 4:2:0, the first component specifies the full resolution of the luminance. The second component defines that every other values is sampled in both the X and Y direction for the two chrominance layers. In order to retain as much information as possible the sampled blocks can be calculated as the average of those around it that do not get sampled. This effectively makes the chrominance component twice as small in dimensions and four times smaller is data. Once the image has been fully compressed and sent to the receiver the chrominance layers are re-sampled with the average value being distributed to the missing elements.[4]

Figure[1] - Block of data showing 4:2:0 sub-sampling:

175	176	175	175	175	175	174	174
176	176	175	175	175	175	175	174
176	176	175	175	175	175	175	175
176	176	176	176	176	175	175	175
177	177	177	177	177	175	175	175
177	177	177	177	176	175	175	175
177	177	177	177	177	175	175	175
178	178	177	177	177	175	175	175

Figure[2] - Averaged and sub-sampled chrominance block

Block represents a block from the Cb channel

176	175	175	174
176	176	175	175
177	177	176	175
178	177	176	175

Down-scaling chrominance

Another method for reducing the amount of data within the image during compression is to simply down scale the chrominance values, this is generally done by a factor of two in Jpg. Because the values are now closer to zero during the Jpg compression algorithm the Discrete Cosine Transform will reduce most of the values even closer to zero. The high frequency of low values works great for the Huffman encoding and can represent those values with a small number of bits.

[8x8 example]

Padding the image

Padding an image is required when the image dimentions are not completely divisible by the block size. A buffer of zeros is placed around the right and bottom edge of the image depending on how many additional pixels are needed. No more than the block size minus one additional rows or column will every need to be added. The padding of zeros can then be remove from the image after decompression.
Figure[3] - represents the bottom-right most block after padding:

103	105	106	105	105	105	0	0
104	106	105	105	105	106	0	0
102	104	104	103	104	104	0	0
99	101	101	102	102	103	0	0
96	96	98	99	100	102	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

Normalise image

An important step in preprocessing the image is to make sure the image values are centered centered around zero because the DCT works best between the ranges of -127 and 128. This is done by misusing 128 from each element's value.

2.4 The 2D Discrete Cosine Transform

The DCT

The DCT is a spectral transformation derived from the DFT takes image from the spatial domain to the frequency domain

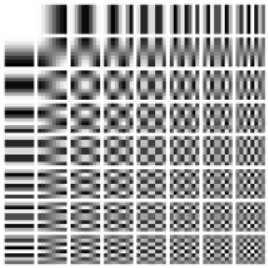
[OneD Basis function]

Block size

The first step in performing the 2D DCT is to separate the input image into smaller blocks. The jpg standard specifies the dimensions of a block as 8 x 8. These dimensions are selected due to the principle of spatial locality which states that, pixels that are close to one another are much more likely to be similar in their intensities, this is especially true for the chrominance channels. This makes a bigger block size have a larger colour gradient between blocks. If there are finer differences of detail in an image between block the decompressed image will not capture these. However, a block size to small will increase the amount of time to compute the DCT across the whole image.
The DCT is calculated on every row and column within a block and each block calculation is independent from one another.

Base DCT wave funtions

Figure[4] - Base DCT wave functions for an 8 x 8 block:



$https://en.wikipedia.org/wiki/Discrete_cosine_transform$ These are the 64 cosines waves that can represent any 8x8 image block for a specific channel. The first element represents a flat cosine wave with no frequency. The frequency of the cosine waves then increase in both the X and Y directions with the bottom-right element representing the summation of the highest frequency in X and Y.

Calculating the 2D DCT of a block

Like the 2D DFT, the 2D DCT also has an equivalent and faster algorithm with a much more appealing time complexity of nLog(n). The author is using the 1D DCT II which is given as the following:

$$X_k \sum_{n=0}^{N-1} X_n cos[\frac{\pi}{N}(n+\frac{1}{2})k]$$

The 2D DCT is calculated by multiplying the horizontally oriented 1D basis functions with the vertically oriented set of the same functions.
The 2D DCT calculates the weighted coefficients of each specific frequency for a block. This then represents the contribution that frequency has to the image. A negative value represents a negative correlation, meaning its contribution is the opposite phase to the specific cosine frequency.[1]
Figure[5] - DCT calculated on Block:
The transformed block is composed of one Direct Current (DC) component (blue), featured in the top-left, and sixty-three Alternating Current (AC) Components. The DC component

-189	-46	-10	-7	-4	-3	-2	0
132	30	-4	-3	-3	-4	-1	0
-5	9	2	2	0	0	0	0
6	-4	0	-1	1	-1	1	1
-5	-4	0	0	1	2	2	1
0	3	0	-1	-2	0	-2	0
-2	1	0	0	0	1	0	0
-5	-1	0	2	0	1	0	0

represents the general image intensity for the block. Because there are generally less high frequency components within a block these values are generally much closer to 0.

2.5 Quantization

talk about quantisation (fundamentals (lecture slides))
talk about psychovisual redundancy[6]
The lower frequency components have a much larger impact on an image than the high frequency components. Because of this, JPG is able to exploit another aspect of the Human Visual System, its poor ability at detecting high frequency changes in image intensity. These higher frequencies can be removed through quantisation with minimal impact to the image. Like the DCT, quantisation is calculated block by block for the entire image.
The jpg standard quantisation matrix is the result of subjective experimentation involving the HVS. These experiments consisted of participants rating decompressed images and then altering the values in the quantisation matrix in accordance with these results, until they achieved a matrix that performed well on a range of input images.[3]
Figure[6] - Standard jpg (JFIF) quantisation matrix (colour used to represent the weightings)
The standard jpg quantisation matrix has a default quality of 50. One benefit of using

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

quantisation for lossy compression is the ability to scale the quality of the decompressed image. As a result, the quality of the image can be tailored to suit different needs. The quality value ranges between 1 and 100, where 1 give the worst quality with the highest compression ratio, while 100 gives the best quality. increasing the quality decreases the quantisation matrix values making it more conservative with its removal of high
if the quality is set to less than 50:

$$S = 5000/quality$$

if the quality is set to 50 or more:

$$S = 200 - (2 * quality)$$

$$e = (S * QM[i, j] + 50)/100$$

This calculation is performed on each element of the quantisation matrix. e becomes the new quantisation value for that element.

Calculating a quantised block

The next operation to be performed is to bit dividing the DC and AC components with the quantisation matrix, round to the nearest integer and place the value into a new block. As pixel values in YCbCr are represented with integers, any values rounded permanently lose frequency information demonstrating the lossy aspect of Jpg.

The quantisation matrix is weighted progressively more on the higher frequencies (the psychovisually redundant information) meaning these coefficient are scaled down more.[2]
Figure [7] - Quantisation matrix:

-11	-4	-1	0	0	0	0	0
12	3	0	0	0	0	0	0
0	1	1	1	0	0	0	1
1	0	1	0	1	0	1	1
0	0	1	1	1	1	1	1
0	1	1	0	0	0	0	1
0	1	0	0	1	1	0	1
0	0	0	1	0	1	1	1

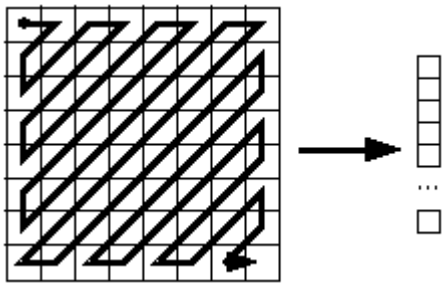
The purpose of the quantisation step is to reduce as many coefficients close to zero as possible. This has the intended purpose of making many of the coefficients small and the same. coefficients with the same value can then be more efficiently encoding during Huffman encoding. Because the higher frequency components generally have less representation in the image and that the quantisation table is heavily weight on the higher frequency components many of the high frequency coefficients are zero or almost zero,

Encoding DC components

Because the DC component is generally much larger than the AC values it can be encoded by misusing the previous block's DC component. This is done for all blocks other than the first which remains the same. On average this causes the DC components to approach 0 making them more likely to be equal to other values. Which also means it is less likely to need its own value in the Huffman Table. During decoding the first DC component is used to calculate all the original block DC components.

2.6 Zigzag scan

Figure[8] - Zigzag scan of block:



The scan works by retrieving values from the square 8 x 8 matrix in a zigzag pattern seen in figure [] and attaching it to a Quantised Value Stream (QVS). The zigzag scan attempts to take advantage of the fact that the high frequency coefficients, which are situated towards the bottom-right of the block, are very low. Therefore the image contains low entropy. The purpose of this scan is to collect the low frequency coefficients towards the end of the QVS. This can be utilised by both run-length encoding and Huffman encoding.

2.7 Zero Run-length encoding

Zero Run-length encoding (ZRLE) works by creating a tuple with the form (zeros, Qvalue) for every non-zero Qvalue encountered. The first parameter is the number of zeros that precedes the Qvalue. A list of ZRLE Qvalues takes the form of:

[(0,-11),(0,-4),(0,12),(1,13),(0,-1),(2,1),(0, 1),(2, 1),(4, 1)...]

Implementing ZRLE increases the computational overhead of the compression algorithm due to the construction of the Zero Run Length (ZRL) codes. Every time a Qvalue is preceded by a new length of zeros a key has to be generated for the code-table which naturally increases the bit length of the least common Qvalues. The author theorises that the ZRLE is advantages to compute, from a compression ratio perspective, if the average run length of zeros is greater than the additional bits required to represent the new keys. If ZRLE is suitable naturally the decode speed will increase.

The author decides not included ZRLE into the algorithm, this is explained in the discussion.

2.8 Entropy encoding: Huffman encoding

Huffman encoding is a lossless compression method. The premiss of Entropy encoding takes advantage of coding redundancy and is based on the assumption that not all intensity levels are equally likely to occur in any image. As a result of this statistical phenomenon the binary code length can be linked to the probability of that intensity value appearing within the image. Values which arise more frequently are encoded with a shorter number of bits and visa versa.

Create a frequency table of QDCT values

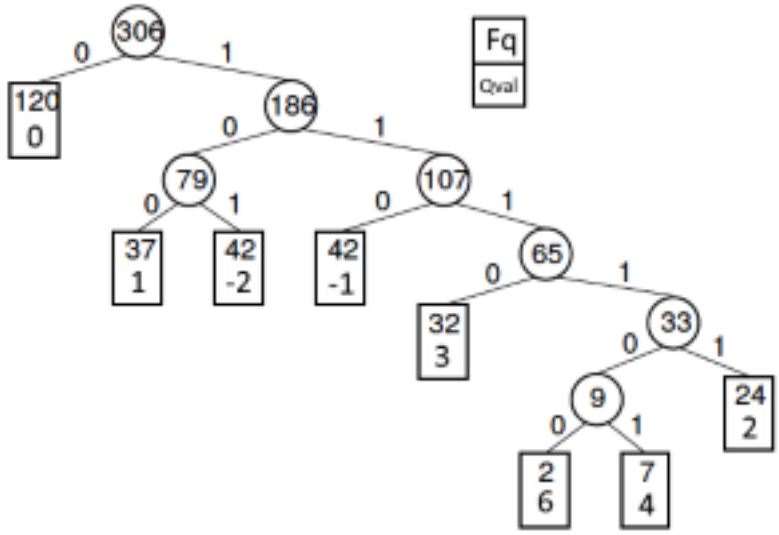
In order to construct a frequency table a single pass must be made on the value stream to collect frequency meta data. From now on the author will refer to the value from the value stream as the Qvalue. The frequency of a Qvalue id divided by the number of Qvalues in the stream, this is the probability of that Qvalue occurring in the image. The Qvalues and frequency are then sorted by frequency in descending order.
Figure[9] - Frequency table of Qvalues before sorting:

QValue	Freq
0	16579
1	16348
2	2929
-1	2766
-2	1228
3	1108

Constructing a Huffman Tree

A Huffman tree is essentially a binary tree which is composed of a many nodes. Nodes hold four pieces of data, the frequency, a value and a left and right child node.
First, all of the Qvalues and frequency pairs are pushed onto a heap. The first two nodes are popped from the heap and become the left and right child nodes of a third parent node whose frequency is the sum of the first two, its value is the python data type None. The author will refer ro these as pseudo nodes. The two least frequent Qvalues are selected because the Huffman tree is generated bottom up. This is to make sure the Qvalues with a highest probability are at the top of the tree which will give them the shortest corresponding binary code length. The third node is then pushed back onto the heap and the processes is repeated until only one element remains in the list, the root node. All other nodes will be situated in the tree based on their frequency.

Figure[10] - Theoretical representation of a Huffman Tree:



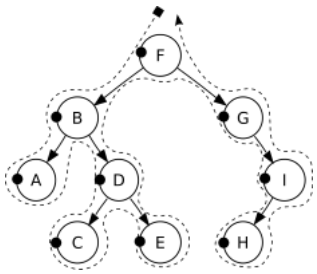
https : //en.wikipedia.org/wiki/Tree_traversal

Generating Huffman codes

Now that the Huffman Tree has been constructed it can be traversed with a recursive function and the Qvalues. The authors algorithm uses a pre-order traversal method. At each the node its value is compared to the Qvalue. If the nodes value is None (pseudo node) then

the function recurses further passing the left node as the new root node and pushes the old node onto the stack. If a leaf node is reached the function will pop nodes from the stack until an unexplored node is available. Each time the function recurses a single bit is added to a code string, recursing to a left child appends a '0' and recursing right appends a '1'. When the node's value matches the Qvalue its corresponding binary code has been created.

Figure[11] - Pre-order traversal



https : *//en.wikipedia.org/wiki/Tree_traversal* Next two hash tables (python dictionaries) are used to store the Qvalue and binary code, The first contains the Qvalue as the key and binary code as it's value, this will be referred to as the code-table. The second is the opposite of the code-table and will be known as the reverse-table.

Figure[12] = code-table, Qvalues with their appropriate binary code:

Qvalue	Code
0	0
1	11
2	1000
-1	10111
-3	1011011
4	1011010

Creating the binary stream

Now that the code-table is constructed the QVS is passed value by value into the code-table and the corresponding binary code is outputted to a binary stream. This is the compressed representation of the image. Jpg also requires that the reverse-table is encoded into the data as it will be needed for decoding.

2.9 Decode

Decoding the binary stream is effectively the reverse process of the encoder.

Decode Huff codes

Firstly, the decoder retrieves the reverse-table from the file and process the binary value stream one bit at a time. If the binary value resides within the reverse-table then the corresponding Qvalue is returned. If not, a new bit from the binary value stream is appended to the previous bit/s and the process is then repeated until the binary stream of values is empty.

Reconstructing matrix of quantised coefficients with zigzag scanning

The Qvalue stream is still ordered in a zigzag orientation. Because of this the stream must first be un-zizaged. The reordered Qvalue stream is then placed into the an 8 x 8 matrix in the original order of the quantised block which will be referred to as the quantised value block.

Decode DC coefficients

The first DC component encountered is the only DC component not encoded. All proceeding DC components are calculated by adding the previous encoded DC component.
[show example]

Dequantization

The original JPG standard quantisation matrix is used to dequantise the quantised value block. Dequantisation is the inverse of quantisation. unlike before, the quantisation matrix is bit multiplied with the corresponding index position in the quantised value block. Once the block is quantised, the new values are very similar to the original DCT block. Any discrepancies in values is due to the the loss of data through quantisation.

Inverse Discrete Cosine Transform

The author uses DCT III also known as the inverse DCT equation for recalculating the YCbCr intensity values. The inverse DCT is given by the following equation:

$$\text{large}X_k = \frac{1}{2}X_0 + \sum_{n=1}^{N-1} X_n cos\left[\frac{\pi}{N} n\left(k + \frac{1}{2}\right)\right]$$

2.10 Undo pre-processing steps

unnormalise

The image is unnormalised by adding 128 to every pixel intensity

Resampling chrominance layers

Resample the chrominance layers consists of contrasting

Colour space conversion (YUV 2 RGB)

Chapter 3

Testing and Results

3.1 Testing

Unfortunately the author was unable to compress the images provided for the coursework and will therefore be using an independent image. However, similar images are selected to see the contrast between natural and man-made objects under compression.

Subjective fidelity check

Lena Original:  Lena Decompressed: 

Building Original:  Building compressed:


note: didn't have enough time to format sorry

3.2 Results

Results

Image quality comparison of Lena:

Quality\Test	Speed	CR	MSE	MSNR	PSNR
10	44.3	30:1	105	10.2	27.3
50	43.9	18:1	100.3	10.7	28.1
99	46.6	8:1	49.8	15.8	31.1

Image quality comparison of Building:

The MSE is the cumulative squared error between the compressed and the original image. The PSNR is a measure of the peak error.47

Quality\Test	Speed	CR	MSE	MSNR	PSNR
10	45	40:1	105	10	27.8
50	45.5	23:1	100.7	10.3	28.1
99	47	14:1	72.1	11.8	29.3

Chapter 4

Conclusion

4.1 Summary

Compression ratio was better on unnatural images. This is due to the smaller number of edges within the image which represent higher frequencies. Most of the building data is low entropy and contains redundant information. However, the edges of the compressed building are far more suspectable to errors.

The author also noted that, compression quality does not effect the computational time nearly as much as the size of the image. This is due to the fact the DCT is a computationally expensive calculation to run with a time complexity of nLog(n) which when computed across the entire image can take a long time.

After it was to late the author realised that the algorithm did not resample the chrominance layers correctly and has caused the image to be twice as small and has very poor image quality.

4.2 Further research

The author investigated a number of other potential methods for image compression, such as 'An End-to-End Compression Framework Based on Convolutional Neural Networks' which uses a convolutional neural network architecture and utilises the frequency domain similar to jpeg. The CNN alters values in the YCbCr image and attempts to reduce the amount of data representing the image while maintaining an image that looks unchanged to the HVS.

4.3 Reflections

Given another opportunity, the author would reformat all the code. With the first attempt complete a far more planned algorithm could me made now that the author understand the jpg process much better. This includes removing poor coding practise when transforming and manipulating arrays. The author wanted to demonstrate that they understood the full process of jpeg so coded in a very C++ like way. In hindsight this was a poor choice and coding phonically would have been better as not using python libraries for some of the calculation meant the computational time was very slow.

The author definitely implement ZRLE before creating the Qvalue stream during Huffman encoding. This would decrease the computational time to process the larger images supplied. Additionally, the author has already implanted zigzag encoding so is already half way.

Author would liked to have developed some graphs for the data collected

Lastly, this was the authors first attempt at using LaTeX to construct a report. The equations and tables fit nicely however some more formatting is needed.

Chapter 5

Appendix

[1] Digital image compression - Refeal C Gonzalez [2] http://www.massey.ac.nz/ mjjohnso/notes/59731/presentations/jpeg.pdf
[3] JPEG tutorial - Andrew B. Lewis [4] Image Compression and the Discrete Cosine Transform
[5] Basic Image Compression Algorithm and Introduction to JPEG Standard [6] Jpeg Image
Compression Using Discrete Cosine Transform - A Survey A.M.Raid [7] An End-to-End Compression Framework Based on Convolutional Neural Networks, IEEE [8]ttps://dsp.stackexchange.com/questions/1748/chroma-subsampling-how-to-properly-calculate-the-data-rate