

Student Number: 29010497, 29010661

Module Code: CS3IA16

Date Completed: 05/12/2022

Assignment Report Title: Image Analysis (Image Compression)

Abstract:

The objective of this group assignment is to gain a better understanding of image compression by applying a variety of image compression techniques on the given image. By applying MatLab, our group will focus on image compression and decompression. In this technical report, our group will have demonstrated the compression cycle on how to load and display an image, how to compress an image and save it to memory and how to read compressed images from memory. We will be able to present the decompression pattern on how to decompress and display an image and how to compute and display compression ratio (CR) statistics. Thus, in the results the original image, compression ratio achieved by the compression and the compressed then decompressed image will be outlined, explained and referenced below in this assignment.

Introduction:

Image enhancement aspires to emphasise desirable image features and suppress other undesirable features. The desirable elements in an image can allow for a precursor of information to be extracted from it. Desirable features of an image can become observed by the human eye and interpreted or allow for perception of detail. For instance, automated image processing provides an “enhanced” input. This process is known to be an interactive yet iterative procedure. There are two broad categories of image enhancement to be aware of in this assignment. These are spatial domain methods and frequency domain methods. Spatial domain methods refer to the image plane itself and operate directly on the image pixels, while frequency domain methods operate on the Fourier transform of an image.

Image compression is a technique which is use for compressing an image to reduce the time it to transmit the image over the internet. It also helps to save space on the disk or memory when you store an image. What image compression does is that it uses data redundancy to remove the data which is not relevant for in the image for example if the image as similar pixels which are not useful than it will be removed to compress the image.

There two types in compression technique, lossy compression and lossless compression, when lossy compression applied, data lost in the process is not recoverable whereas with the lossless compression you can recovered all you data when you decompressed image. Lossy compression useful because it can reduce the file size way more than lossless compression but it will also sacrifice the image quality.

Methodology:

Theory

This experiment was conducted to create a program to compress an image using one of the famous algorithms which are used for image compression. We as a group researched which algorithm we could use for this experiment. We came up with the Huffman encoding and decoding algorithm, which is a lossless algorithm, what this means is that when the image is compressed and then decompressed back, the data is not lost, whereas other algorithms like DCT algorithms which are lossy, which mean you won't get the original image back, whatever data you lose in compression, will be unrecoverable.

Huffman Coding

Huffman coding is a lossless and efficient algorithm which is also a variable length coding. This is famous and widely used in data and image compression because of its efficiency. This technique is simpler and easy to implement because of its simple mathematics. In this algorithm each character is assigned a prefix code that is unique to each character, this is important because when it comes to decoding the data, it will make sure that there is no ambiguity in the bitstream.

The way Huffman works is that it creates a Huffman directory, and this is created using the Huffman tree. This Huffman directory creates unique symbols and probabilities for each character in the data, after this Huffman encoding is performed which converts each character into encoded 1s and 0s. To decode the data same tree is used, but in an opposite way which is called the traverse tree method.

Implementation

For our experiment we use MATLAB and inbuilt tools such as "huffmandict", "reshape", "huffmanenco", "huffmandeco", "immse", "imread", "imwrite". To implement this code, we have also used functions, to divide different functionality of the code, as we have created 3 functions, one for Huffman dictionary, second for Huffman encoding, and third for Huffman decoding, all these functions are in separate ".m" file, and the main code is in "HuffmanImgCompression.m" file.

Steps we took for Huffman implementation

- 1) First, we used the 'imread' function to read the image from the disk
- 2) Then we plotted the original image
- 3) Next, we created the Huffman directory
- 4) Now it was time to reshape the image because we needed vector values which are expected by the Huffman encoding function
 - To deal with the color image, we separated the RGB channels of the image and perform encoding and decoding of all the 3 channels separately, and at the end when we reshape the image, we combine the 3 channels together.
- 5) Then Huffman encoding is performed on the given directory and image in vector form.
- 6) After generating Huffman encoded data of the image it was time to decode the data and reshape the image to get the result, we also use 'imwrite' to write the file to the disk.

Results:


To test out the code we started with the small-size image and got statistics such as the average length of the code, entropy, efficiency, compression ratio, relative data redundancy, and mean-square error. Below is the table which contains the different images with their results.



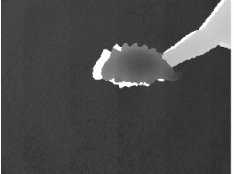
With our implementation of Huffman coding, we found that the smaller the image size better the compression works, and the grayscale image is easier to compress, we can say that because of the compression ratio, the higher the compression ratio lower the size, but the quality will be reduced. As you can see in the results table below, if we compare the 3 coloured images, you can see that the results are similar that's because they are of the same size whereas the 4th dinosaur image is smaller in size.

Another thing that came up in our test result is that in image number 3, the data redundancy is higher than the other images which says that image 3 had more irrelevant data compare to other images, in the image you can see that there are only 3 main parts sky, trees, and grass, this means that the image has same similar pixels which are representing same things and image compression removes those irrelevant data to compression. We can say that this algorithm is lossless because the mean-square error is '0', this means that the original image and decompressed image both are the same, and there is no data loss during the compression process.

comparing time to compress the image


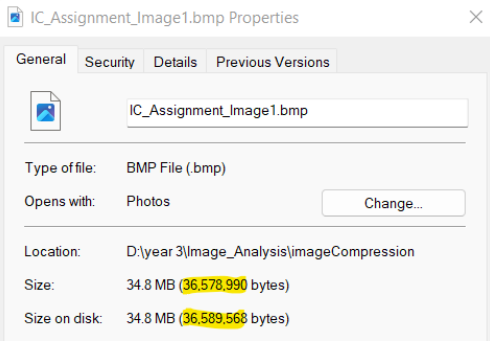
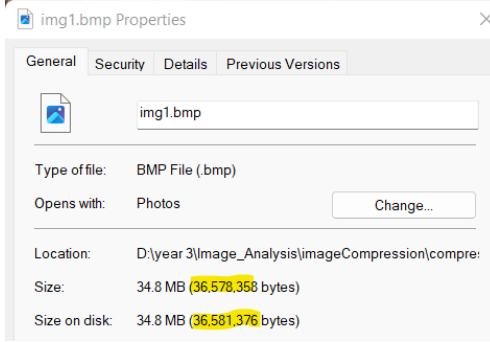
Image 4 took the least time to compress because it is lower in size so, there are fewer data to deal with compared to the other images which are 6 times bigger and have more data. Another thing that we notice was that when we run the code with the data displaying on the console took more time than removing the functionality of displacing the data on the console. Another thing that was time consuming was writing data onto the file.


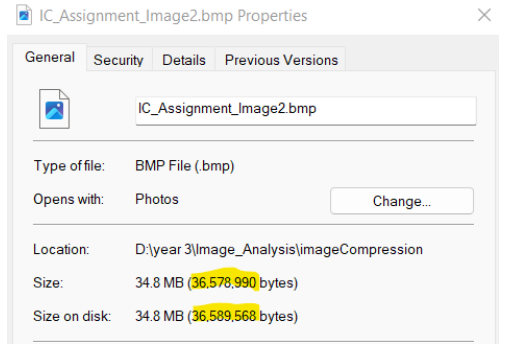
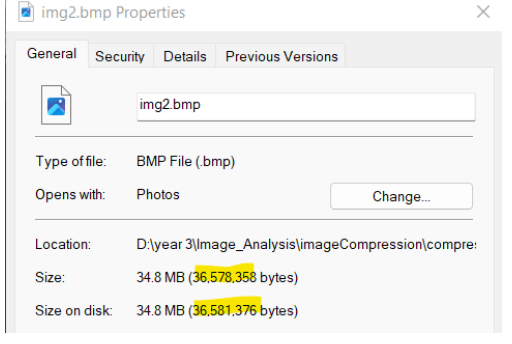

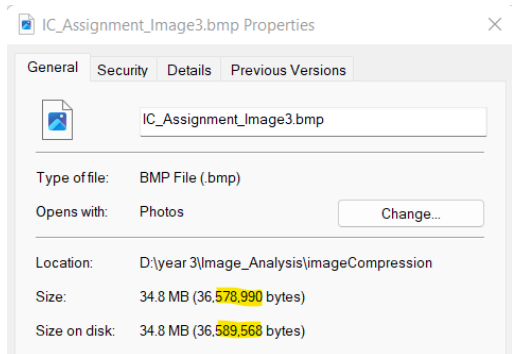
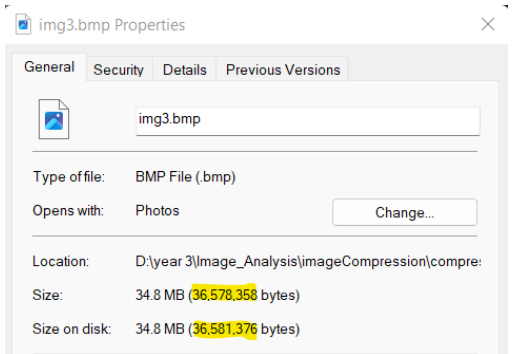
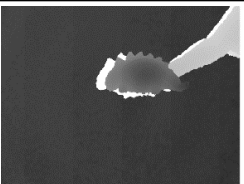
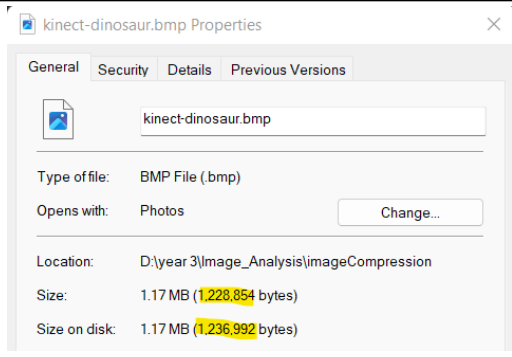
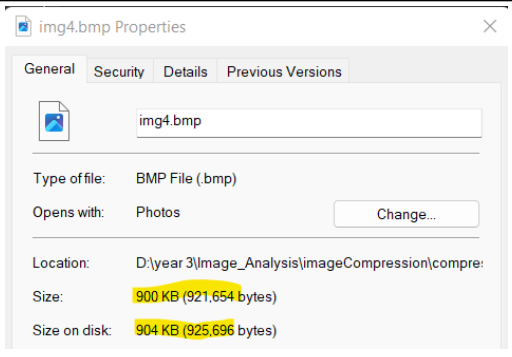
Test results								
Sr. no.	Images	Average code length	Entropy	Efficiency	Compression ratio	Data redundancy	MSE(mean-square error)	Time took to compress
1	 (size-34.8mb)	7.7361	7.7086	99.6557	1.0341	3.2993	0	17:28 min

2	 (size-34.8mb)	7.6025	7.5718	99.5962	1.0523	4.9693	0	17:59 min
3	 (size-34.8mb)	6.8625	6.8335	99.5775	1.1658	14.2189	0	17:58 min
4		3.759391	3.715848	98.84175	2.128004	5.300761	0	26:30 sec

Size comparison

Below is the size comparison evidence of the images, you see the highlighted numbers which are the size in bytes, as you can see the image size not reduce that much but the positive thing was that the algorithm is working and compressed the image. As you can see the first 3 images were reduced by 632 bytes from the original image and 4th image is reduced by 307,200 bytes were reduced from the original image. This shows that the 4th image was reduced quite a lot compared to other images from 1.17 Mb to 900kb.

Size Comparison		
Sr.no	Before compression	After compression
 Image 1 Compression Difference (632bytes)		

 <p>Image 2</p> <p>Compression difference (632bytes)</p>	 <p>IC_Assignment_Image2.bmp Properties</p> <p>General Security Details Previous Versions</p> <p>IC_Assignment_Image2.bmp</p> <p>Type of file: BMP File (.bmp)</p> <p>Opens with: Photos Change...</p> <p>Location: D:\year 3\Image_Analysis\imageCompression</p> <p>Size: 34.8 MB (36,578,990 bytes)</p> <p>Size on disk: 34.8 MB (36,589,568 bytes)</p>	 <p>img2.bmp Properties</p> <p>General Security Details Previous Versions</p> <p>img2.bmp</p> <p>Type of file: BMP File (.bmp)</p> <p>Opens with: Photos Change...</p> <p>Location: D:\year 3\Image_Analysis\imageCompression\compre:</p> <p>Size: 34.8 MB (36,578,358 bytes)</p> <p>Size on disk: 34.8 MB (36,581,376 bytes)</p>
 <p>Image 3</p> <p>Compression difference (632bytes)</p>	 <p>IC_Assignment_Image3.bmp Properties</p> <p>General Security Details Previous Versions</p> <p>IC_Assignment_Image3.bmp</p> <p>Type of file: BMP File (.bmp)</p> <p>Opens with: Photos Change...</p> <p>Location: D:\year 3\Image_Analysis\imageCompression</p> <p>Size: 34.8 MB (36,578,990 bytes)</p> <p>Size on disk: 34.8 MB (36,589,568 bytes)</p>	 <p>img3.bmp Properties</p> <p>General Security Details Previous Versions</p> <p>img3.bmp</p> <p>Type of file: BMP File (.bmp)</p> <p>Opens with: Photos Change...</p> <p>Location: D:\year 3\Image_Analysis\imageCompression\compre:</p> <p>Size: 34.8 MB (36,578,358 bytes)</p> <p>Size on disk: 34.8 MB (36,581,376 bytes)</p>
 <p>Image 4</p> <p>Compression difference (307,200bytes)</p>	 <p>kinect-dinosaur.bmp Properties</p> <p>General Security Details Previous Versions</p> <p>kinect-dinosaur.bmp</p> <p>Type of file: BMP File (.bmp)</p> <p>Opens with: Photos Change...</p> <p>Location: D:\year 3\Image_Analysis\imageCompression</p> <p>Size: 1.17 MB (1,228,854 bytes)</p> <p>Size on disk: 1.17 MB (1,236,992 bytes)</p>	 <p>img4.bmp Properties</p> <p>General Security Details Previous Versions</p> <p>img4.bmp</p> <p>Type of file: BMP File (.bmp)</p> <p>Opens with: Photos Change...</p> <p>Location: D:\year 3\Image_Analysis\imageCompression\compre:</p> <p>Size: 900 KB (921,654 bytes)</p> <p>Size on disk: 904 KB (925,696 bytes)</p>

Conclusion:

In this final section of this technical report we have demonstrated an exceptional understanding of image compression in MatLab and have been able to successfully show a variety of techniques, such as Huffman coding to implement this. A solid knowledge and a comprehensive comparison of results has been identified in this technical report. However, for the future there are a few improvements that can be made to our image compression project. An example of this is the time taken to compress an image. Currently for our project it takes over 20 minutes for an image to become compressed, which is quite a while to wait. If given more time we would update the code to make the compression time quicker and much more efficient to run. Another future work we started working on but did not complete by the demonstration date is by creating an app for the code. An application would be a much smoother way of running the code as we currently have different separate in-built code to run.

Appendix:

Code

File name : HuffmanImgCompression.m

```
%%
close all;
clear all;
clc;
%input the picture name to compress the image
I = imread('kinect-dinosaur.bmp');
%resize the image
I = imresize(I,0.4,'nearest');
%imwrite(I,'resizeimg.bmp');
% display the pixel value of Image
disp('the pixel values are:');
disp(I);
%plotting an image
subplot(1,2,1);
imshow(uint8(I));
title('Original image');
%creating a Huffman Dictionary
dict = huffDict(I);
% main loop for looping the Huffman encoding and decoding image RGB
for i = 1:3
    %image size
    [m,n] = size(I(:,:,i));
    %encoding the image
    hcode = Huffman_encoder(I(:,:,i));
    %displaying the image
    disp('The Huffman encoded code is:');
    disp(hcode);
    %writing Huffman encoded data to file
    hdata = fopen('huffdata.txt','w');
    fprintf(hdata,"%d",hcode);
    fclose(hdata);
    %decoding the image
    HD1 = Huffman_decoder(I(:,:,i),hcode);
    %displaying the image
    disp('The Huffman decoded code is:');
    disp(HD1);
    HD=uint8(HD1);
    %restoring the image
    Restore=reshape(HD,[],n);
    I2 = Restore;
    %putting RGB image data together to make the image
    if i == 1
        R = I2;
    elseif i == 2
        G = I2;
    elseif i == 3
        B = I2;
```

```

% combining 3 chanelns together into new image
    N(:,:,1) = R;
    N(:,:,2) = G;
    N(:,:,3) = B;
%ploatting an image
subplot(1,2,2);
    imshow(uint8(N));
    title('Decoded Image');
% writing decoded image to the disk
    imwrite(N,'decoded.bmp');
%caculating mean-sqaure error
    meanSqaureErr = immse(I,N);
end
end

```

File name: Huffman_encoder.m
Function for huffman encoding

```

function hcode=huffman_encoder(I)
%creating a huffman directory
dict = huffDict(I);
%reshaping the image into vector form
newvec=reshape(I,1,[]);
%encoding the data using huffman encoding
hcode=huffmanenco(newvec,dict);
end

```

File name: huffDict
Huffman directory funtion

```

function dict=huffDict(I)
%open text file
info = fopen('stats.txt','w');
%getting the image size in m x n form
[m,n] = size(I);
%counting the total counts
Totalcount=m*n;
%creating unique symbols
symbols = unique(I);
%counts the number of symbols
counts=histc(I(:),symbols);
%caculating the probabilities
pro=counts./Totalcount;
%creating huffman directory and getting the average length code
[dict,avglen]=huffmandict(symbols,pro);
disp('the average length of the code is :');
disp(avglen);
%writes average length code to text file
fprintf(info,"the average length of the code is : %d\n",avglen);

```

```

%calculating entropy
ntropy=entropy(uint8(I));
disp('Entropy is:');
disp(ntropy);
%writing entropy data to text file
fprintf(info,"Entropy is: %d\n",ntropy);
%calculating efficiency
E=(ntropy/avglen)*100;
disp('Efficiency is:');
disp(E);
%writing efficiency to text file
fprintf(info,"Efficiency is: %d\n",E);
%calculating compression ratio
disp('Compression ratio is:');
CompressionRatio=(Totalcount*8)/(Totalcount*avglen)
%writing compressio ratio to text file
fprintf(info,"Compression ratio is: %d\n",CompressionRatio);
%calculating redundancy
disp('Relative data redundancy is:');
redundancy = (1 - (1/CompressionRatio))*100
%writing data reducnacy to text file
fprintf(info,"Relative data redundancy is: %d\n",redundancy);
disp("-----")
%closing an open text file
fclose(info);
end

```

File name: `huffman_decoder`
Function for decoding encoded data

```

function hdecode=huffman_decoder(I,hcode)
%creating huffman directory
dict = huffDict(I);
%decoding the encoded code
hdecode=huffmandeco(hcode,dict);
end

```

You can find code on this gitlab repository

https://csgitlab.reading.ac.uk/ak010661/image_analysis_2022/-/blob/main/image_compression/image_compression

References

Zhao, W., Cao, Z. and Liu, P. (1970) *A Combining Spatial Enhancement Method for Low Illumination Images*, undefined. Available at:

<https://www.semanticscholar.org/paper/A-Combining-Spatial-Enhancement-Method-for-Low-Zhao-Cao/972430b924d39c3303fbc87e478aa5b3ad5ffa80> (Accessed: December 5, 2022).

Voronin, V. et al. (1970) *Thermal Image Enhancement Algorithm using local and global logarithmic transform histogram matching with spatial equalisation*. Available at: <https://www.semanticscholar.org/paper/Thermal-Image-Enhancement-Algorithm-Using-Local-And-Voronin-Tokareva/d2bced4d3d030d3029934374fd3e5f1cc6a3881d> (Accessed: December 5, 2022).

Voronin, V. et al. (1970) *Underwater Image Enhancement Algorithm based on logarithmic transform histogram matching with spatial equalization*. Available at: <https://www.semanticscholar.org/paper/Underwater-Image-Enhancement-Algorithm-Based-on-Voronin-Semenishchev/34c1be9ac7eb9e018e7927caadf9aab8bc3ed0fe> (Accessed: December 5, 2022).

Narasimha, C. and Rao, A.N. (1970) *A Comparative Study: Spatial Domain Filter for Medical Image Enhancement*. Available at: <https://www.semanticscholar.org/paper/A-comparative-study%3A-Spatial-domain-filter-for-Narasimha-Rao/df197e6e709b6738cb8a0ec1c7529dc3f2a93df2> (Accessed: December 5, 2022).

Image compression - farm service agency (no date). Available at: https://www.fsa.usda.gov/Assets/USDA-FSA-Public/usdafiles/APFO/support-documents/pdfs/compression_infosheet_2017.pdf (Accessed: December 7, 2022).

Comparison of image compression techniques using Huffman ... - IEEE Xplore (no date). Available at: <https://ieeexplore.ieee.org/abstract/document/7218137/> (Accessed: December 7, 2022).

Sheldon, R. (2022) *What is image compression and how does it work?*, WhatIs.com. TechTarget. Available at: <https://www.techtarget.com/whatis/definition/image-compression> (Accessed: December 7, 2022).

Effort Allocation Sheet:

Name:	Contribution (%)	Contribution Explained:	Signature:
Aaliah Hussain	100%		Aliah Hussian
Roshan Magan	100%		Roshan magan