

Custom Lossless Compression and High-Quality
Lossy Compression of White Blood Cell Microscopy
Images for Display and Machine Learning
Applications

Terese Nilsson Daniel Hamngren

June 13, 2013

Abstract

This master's thesis investigates both custom lossless compression and high-quality lossy compression of microscopy images of white blood cells produced by CellaVision's blood analysis systems. A number of different compression strategies have been developed and evaluated, all of which are taking advantage of the specific color filter array used in the sensor in the cameras in the analysis systems. Lossless compression has been the main focus of this thesis.

The lossless compression method, of those developed, that gave best result is based on a statistical autoregressive model. A model is constructed for each color channel with external information from the other color channels. The difference between the predictions from the statistical model and the original is further Huffman coded. The method achieves an average bit-rate of 3.0409 bits per pixel on the test set consisting of 604 images.

The proposed lossy method is based on taking the difference between the image compressed with an ordinary lossy compression method, JPEG 2000, and the original image. The JPEG 2000 image is saved, as well as the differences at the foreground (i.e. locations with cells), in order to keep the cells identical to the cells in the original image, but allow loss of information for the, not so important, background. This method achieves a bit-rate of 2.4451 bits per pixel, with a peak signal-to-noise-ratio (PSNR) of 48.05 dB.

Preface

This report is our master's thesis for the degrees in Master of Science in Engineering Mathematics (TN) and Engineering Physics (DH) respectively. The project was performed within the Centre for Mathematical Sciences at Lund Institute of Technology (LTH), and took place at CellaVision AB in Lund, during 20 weeks (30 ECTS) in the spring of 2013.

Our supervisors in this project were Kent Stråhlén at CellaVision and Niels Christian Overgaard at the Centre for Mathematical Sciences.

We would like to thank everyone that has helped us with this. Especially our supervisors Niels Christian Overgaard and Kent Stråhlén for their guidance and support. But also all the employees at CellaVision that has helped us with difficulties that have occurred.

Contents

1	Introduction	1
1.1	Background	1
2	Theory	3
2.1	Common Compression Standards	3
2.1.1	JPEG	3
2.1.2	JPEG 2000	4
2.1.3	PNG	5
2.2	The Color Filter Array	5
2.2.1	Separation of Color Channels	6
2.3	Entropy Coding	7
2.3.1	Huffman Coding	8
2.3.2	Arithmetic Coding	10
2.4	Run-Length Encoding	11
2.5	Discrete Wavelet Transform	12
2.5.1	Biorthogonal Wavelets	15
2.5.2	2D Wavelet Transform	15
2.5.3	Wavelet Transform for Compression	16
2.5.4	Embedded Zerotree Wavelet	17
2.6	Regression Analysis	18
2.6.1	Using Regression in Image Compression	18
2.6.2	Simple Linear Regression	18
2.6.3	Autoregressive Model	19
2.6.4	ARX-Model	20
2.7	PCA - Principal Component Analysis	21
2.8	Reversible Integer Transformation	22
3	Our Compression Strategies	27
3.1	Strategy 1 – Color Separation and Lossless JPEG 2000	28
3.2	Strategy 2 – Difference Image with Standard Lossy Compression	28
3.3	Strategy 3 – Wavelet Transform with ARX	29
3.4	Strategy 4 – Cross Channel ARX	29
3.4.1	Strategy 4.5 – Limited Cross Channel ARX	30

3.5	Color Transformations	31
3.5.1	Strategy 5 – YDgCoCg Transformation	31
3.5.2	Strategy 6 – PCA Integer Transformation	31
3.6	Strategy 7 – Mallat Packet Transform with Zerotree	32
3.7	Lossy Strategy – High-Quality Lossy Compression	34
4	Results	37
4.1	Lossless Compression	37
4.2	Lossy Compression	40
5	Discussion and Conclusion	43
5.1	Choice of method	43
5.2	Comparisons	44
5.3	Possible Improvements	46
5.4	Conclusion	46
A	Example of Cell Images	47
B	Solutions to Exercises	49

Chapter 1

Introduction

1.1 Background

Being able to store a lot of digital information using a limited amount of space is often a problem. The cost of storing data is lately drastically reduced and the availability for high density storage greatly improved, but the amount of data in need of storing is also greatly increased.

The development of imaging for medical purposes has come a long way since the discovery of X-rays in 1895 by Wilhelm Röntgen [1]. Today we have modalities in medical imaging, such as, radiography, magnetic resonance imaging (MRI), nuclear medicine, photo-acoustic imaging and tomography [2]. The different procedures produce a vast amount of data and in order to feasibly save this data, compression techniques are essential tools.

There are two main reasons for developing new compression techniques. One is to limit the need for storage space. The other is to save time when transmitting data. E.g. transmitting the data from the imaging device to a computer, or from one medical expert in one hospital to a colleague in another.

In medical imaging, is it important not to introduce artefacts or to remove information in the image. Compression techniques which are removing information are called lossy, techniques which do not are called lossless. The problem with lossy compression is twofold. Firstly, if the images are used in automatic image processing it is preferable to have as pure images as possible, in order not to introduce more errors to a perhaps delicate method. Secondly, if the image is saved lossy with a high enough quality for a certain application, the same image could not be used for another application with a need for an even higher quality.

In this thesis, the focus is set on images produced by the digital microscopy devices from the company CellaVision AB. CellaVision is focusing on hematology, the study of blood, and the main feature of the products is high-quality images depicting white blood cells. There is a variety of

different types of white blood cells that can be found in the blood stream and the distribution of white blood cells can be of great importance in the diagnosis of a patient. CellaVision has trained artificial neural networks on these images in order to automatically classify the white blood cells as an aid to the medical professionals. An example of one of the cell images we used to test our compression methods is presented in Figure 1.1 and more images are found in Appendix A.

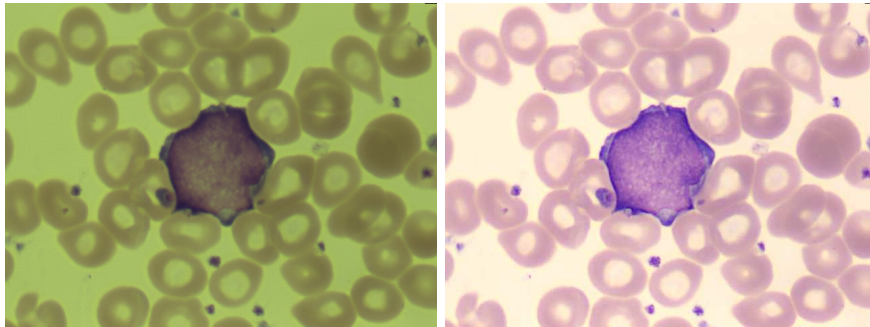


Figure 1.1: An image from the machine depicting a white blood cell among red blood cells. The image to the left is the raw image directly from the camera and the one to the right is the normalized image that is displayed to the user.

After the analysis of the cells, the images are stored and later displayed with a lossy compression. At high magnification, the compression artefacts may be visible. Another disadvantage is that the lossy compression makes it impossible to go back to the analysis stage and see why the automatic classification came to the conclusions it did.

Since the different blood cell images share common characteristics, e.g. color scheme and shapes, and it is known exactly how the image sensor is built, it is deemed plausible that a lossless compression algorithm tailored for the purpose of compressing images depicting blood cells, would achieve better compression than general lossless compression standards such as lossless JPEG 2000 or PNG.

The aim of this master's thesis is to:

- Create new custom lossless compression techniques in order to maximize the compression ratio on the microscopy images on blood cells supplied by CellaVision AB from their analysis systems.
- Create a high-quality custom lossy compression technique for the same type of images as mentioned above.

Chapter 2

Theory

There are two main fields in image compression [3]. The first is lossless compression. Lossless compression is when no information is removed from the image. When no information is removed it is possible to fully recreate the original image from the compressed image.

The second field is lossy compression. Here, information which is considered inessential under the circumstances is removed from the image in order to further reduce the space needed to store the image. The main goal is to remove as much information as possible while still keeping an acceptable image quality. A lot of information from an image can be removed before a human would notice any difference from the original image.

In this chapter we are going to present the theory that is needed to understand the different compression strategies examined in Chapter 3. We also present some of the common compression methods that already exists.

2.1 Common Compression Standards

There exist many different standards for saving images, some more well known than others. Examples of such standards are JPEG, TIFF, PNG, GIF, JPEG 2000 and BMP. Most of the image standards are also compressing the images, either lossy or losslessly. The different standards are specialised to handle certain types of images. For instance, PNG and GIF handle graphics well, whereas JPEG and JPEG 2000 is specialized in real life photographs. Below, three compression standards are presented in depth.

2.1.1 JPEG

The image compression standard JPEG was created in 1986 and stands for the Joint Photographic Experts Group. It is a lossy compression method. To compress the image, the RGB values are transformed into a new color space called YCbCr, which, instead of representing the image as red, green

and blue channels represents it as a luma (brightness) and two chroma (color difference) channels. The image is then divided into blocks of 8×8 pixels. A discrete cosine transform (DCT) is calculated for each of the blocks. After that the resulting coefficients are quantized, i.e. the amount of information in high frequency components is reduced. This can be done because the human eye is not very sensitive to fast changes, i.e. small details, in an image.

The last step is a kind of entropy coding (Section 2.3), which arranges the pixels in a zigzag pattern so the lowest frequencies appear first and uses run-length encoding (Section 2.4) to encode the long runs of zeros and then Huffman codes the result [4, 5].

When compressing images using JPEG it is possible to regulate the degree of compression. To do this, a parameter, which decides how much the quantization will affect the frequencies saved after the DCT, is changed. This parameter is called the quality parameter. Usually, it can take values between 1 and 100, where 100 corresponds to no quantization and thus maximal quality. There are also alternative scales for the quality parameter used in some applications.

2.1.2 JPEG 2000

JPEG 2000 was created in order to take care of some of the problems that JPEG suffers from. One of them is the blocking artefact introduced by the division into 8×8 blocks. The main difference between JPEG and JPEG 2000 is that JPEG 2000 uses a discrete wavelet transform (Section 2.5) instead of a discrete cosine transform. The wavelet transform is applied to the whole image, or to larger blocks, instead of the small blocks that JPEG uses. The sub-bands that are the result of the wavelet transform are split into smaller regions. Each region is then encoded with a method called EBCOT (Embedded Block Coding with Optimal Truncation) that starts with the most significant bits and continues towards the least significant bits. The least significant bit planes (Section 2.5.4) can be dropped to get higher compression.

As in the case with JPEG, a parameter is supplied in order to regulate the compression. The value of this parameter is the desired change in file size. Setting the parameter to ten gives a resulting image a tenth of the file size of the original.

JPEG 2000 can also save images losslessly. In that case, a reversible integer wavelet transform is used and no bit planes are dropped in EBCOT [5, 6].

2.1.3 PNG

PNG (Portal Network Graphics) is a lossless image compression technique. It was created in 1995 as a free replacement for the image format GIF, which uses a component that was patented.

A PNG image can be saved in both gray scale and RGB, with different number of bits per pixel, such as 1, 2, 4, 8 and 16 for gray scale and 24 and 48 for RGB. It is also possible to have an alpha channel that supports transparency.

The compression used in PNG is divided into two steps. First, a filtering with prediction is applied and then the real compression is made with a method called DEFLATE. The filtering step applies a filter for the whole image and another for each row in the image. The pixels in each row are predicted based on the pixels to the west, north and north-west of the current pixel. The predicted value is then subtracted from the true value and the difference is saved [7]. DEFLATE [8] is a combination of the LZ77 algorithm and Huffman coding. LZ77 [9] is a compression method that uses a dictionary while compressing the data and codes repeated streams of data with references to the earlier occurrence of that stream. Huffman coding is described in Section 2.3.1.

2.2 The Color Filter Array

The cameras used to capture the cell images, as well as the majority of all modern digital cameras, has a color filter array (CFA) between its lens and the sensor. This means that only intensities from a limited part of the color spectrum (e.g. red, green and blue) are recorded at each pixel. The output is therefore a gray-scale image with a mosaic pattern. An image with this kind of pattern is referred to as a mosaic image.

There are different kinds of CFAs. The most common one, and the one of interest in this master's thesis, is called the Bayer filter. The Bayer filter consists of a pattern with fields that are transparent to red, green and blue wavelengths respectively. There are twice as many fields for green as for red and blue [10]. The reason for this is that the human eye is more sensitive in the green spectral range. The pattern can be seen in Figure 2.1.

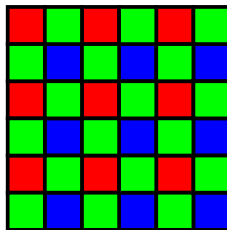


Figure 2.1: The pattern of the Bayer filter.

In order to form an ordinary three channel image from the mosaic image, the red, green and blue channels are interpolated to fill every pixel with color information from all of the channels. This means that a gray-scale image is made into a full color image with three color channels and takes therefore three times the space on disk.

The interpolation can be done in several ways. One example is to give a pixel the average value of the neighbouring pixels. Some of the interpolation techniques are reversible, such as the example above, meaning it is possible to fully reconstruct the mosaic image from the interpolated image. The interpolation used in the images in CellaVision's analysis systems is a reversible one.

2.2.1 Separation of Color Channels

It is convenient to handle the different color channels in the mosaic image separately. Here it is explained how to separate the color channels. The first step is to divide the mosaic image into its three color channels, red, green and blue, as in Figure 2.2.

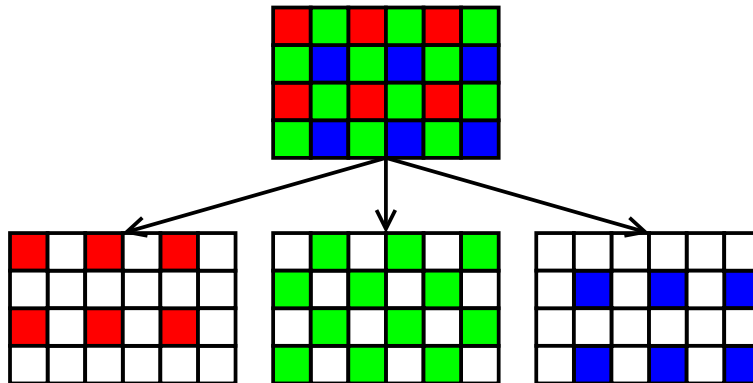


Figure 2.2: The mosaic image divided into its three color channels.

The color channels now contain empty pixels between the pixels containing information from that channel. These empty pixels have to be removed. The red and blue channels can easily be made into smaller images, without empty pixels, just by merging the pixels as in Figure 2.3.

The green channel is structured in a certain pattern, a quincunx pattern, which can be made into a smaller image in several ways. We have tried two different methods to remove the empty pixels in the green channel, called simple merging and structure separation [11]. In simple merging, every pair of odd and even columns are merged into a full column, see Figure 2.4b. The other method, structure separation, moves the odd rows into one image and the even rows into another. The empty pixels in the rows are omitted. This can be seen in Figure 2.4c. Structure separation, gives two smaller images

of the same size as for the red and blue channels, while simple merging gives only one image of twice the size.

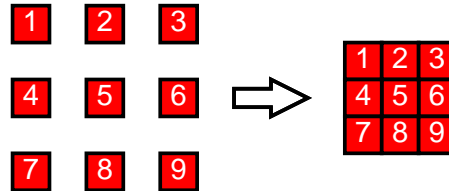
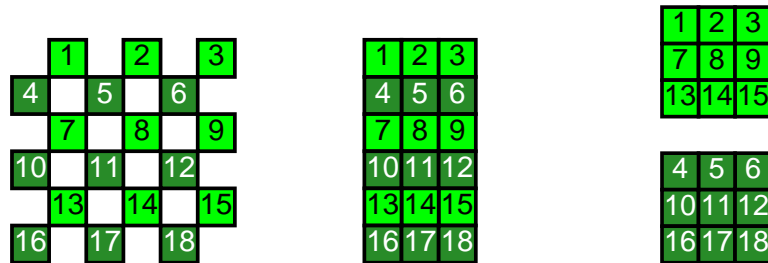


Figure 2.3: Making the red channel into a smaller image without empty pixels. The blue channel is treated the same way.



(a) Quincunx structure (b) Simple merging (c) Structure separation

Figure 2.4: Making the green channel into smaller image(s) without empty pixels using two different methods, simple merging (b) and structure separation (c).

We refer to the two images from structure separation as the first green channel, or G1, and the second green channel, or G2. The first green channel are from the odd rows, and the second green channel are from the even rows in the mosaic image.

2.3 Entropy Coding

Entropy coding as described by Cover et al. [12, ch. 2], uses the probability distribution of the symbols to compress data. These symbols can represent anything, but we will mainly focus on letters and numbers. Suppose the data is composed with a set of symbols $X = \{x_1, \dots, x_n\}$, where the symbol x_i occurs with probability p_i . The entropy, $H(X)$, of such set is defined as

$$H(X) = - \sum_{i=1}^n p_i \log_2 p_i,$$

and has many different interpretations. One of the interpretations is as the average number of bits needed to represent a symbol in the sequence. The

sequence of bits used to represent a symbol is called the codeword. Note that the entropy is only dependent on the relative frequency, or distribution, of the symbols, not the symbols themselves. Compression methods that are based on entropy coding have $H(X)$ as a lower bound of how much it is possible to compress the symbols, on average. This limit may be violated if you use a compression method where, for example, the order of, or patterns within, the symbols are taken into account and not only their distribution.

The entropy is used in a variety of compression algorithms. Among others, Huffman coding and arithmetic coding, both described below.

2.3.1 Huffman Coding

While studying at MIT, David A. Huffman developed, as an assignment in a course on information theory, an algorithm to construct an optimal code. This algorithm is now called Huffman coding and was first published in *A Method for the construction of Minimum Redundancy Codes* [13] in 1952. A Huffman code is an optimal code [12, ch. 5], a code that has the minimum expected length. This means that the expected length of the codewords, $L = \sum p_i l_i$, is equal to the entropy of the set. Here l_i denotes the length of the codeword for symbol x_i . Because the codewords need to have integer length, the expected length of the codewords might not always reach the entropy, but it is always true that

$$H(X) \leq L < H(X) + 1, \quad (2.1)$$

where $H(X)$ is the entropy of the set X .

Huffman codes are also so called prefix-free codes. That means that none of the codewords are prefixes of any of the other codewords [12, p. 106], and the code is thus uniquely decodable.

To construct a Huffman code you need the probabilities of the symbols in the sequence. This probability distribution can be estimated from the relative frequencies of the symbols in the data to be encoded. From the probabilities, a tree is constructed in the following way: Start by letting all the symbols be nodes in the tree. Connect the two least probable symbols and create a new node that has the sum of their probabilities as its probability. Repeat this procedure until only one node remains. If there are more than one possible pair of two least probable symbols, choose one of the pairs. It will give an optimal code in any case. This means that there can be many different Huffman trees for each set. In data compression, the choice of which Huffman tree to use is irrelevant. To get the codewords for each of the different symbols, the tree is traversed from the root to the leaves, giving a 1 to the upper branch and 0 to the lower [14, ch. 5.4].

To decode a coded sequence, start at the root of the tree and follow the branches corresponding to the bits in the code until you reach a leaf. Then one symbol is decoded. Continue from the first non-used bit in the code and

start from the root again. The process is repeated until there are no bits left in the code.

Example. Suppose a sequence is constructed using the symbols in $X = \{A, B, C, D\}$. Assume that the corresponding probability distribution is $\{\frac{1}{8}, \frac{1}{8}, \frac{2}{8}, \frac{4}{8}\}$. A Huffman code for X is constructed as follows. Start by placing the symbols as nodes in a binary tree, as in Figure 2.5a. Connect A and B, because they have the smallest probabilities. Create a new node with the sum of their probabilities, $\frac{1}{8} + \frac{1}{8} = \frac{2}{8}$, see Figure 2.5b. Now, C and the new node have lowest probabilities. Connect them, and create a new node with probability $\frac{2}{8} + \frac{2}{8} = \frac{4}{8}$ (Figure 2.5c). There are now only two nodes left, connect them and the tree is complete (Figure 2.5d). Throughout the whole tree construction, name the upper branch 0 and the lower branch 1 when creating the new nodes. Traversing the tree from the root to each leaf gives the codewords for the different symbols. For this example, the codewords together with their corresponding symbols are shown in Table 2.1, this is what is called a Huffman table.

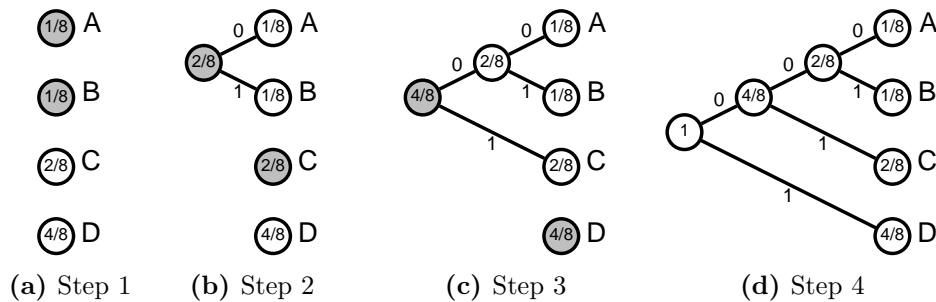


Figure 2.5: The different steps in the construction of a Huffman tree. Marked with gray are the nodes with smallest probability, which in that step will be connected into a new node.

Table 2.1: The Huffman table for the set X .

Symbol	Codeword
A	000
B	001
C	01
D	1

The following sequence of symbols,

D B C D D C A D,

would, using the Huffman table above, be coded as

1 001 01 1 1 01 000 1.

The spaces written in the code above are only included for convenience. There are no need for saving them, because Huffman codes are prefix-free codes and can thus be uniquely decoded without separators between the codewords.

The length of the code above is 14 bits. Had we used the more natural, but less effective, coding where each symbol is coded with two bits, A: 00, B: 01, C: 10, D: 11, then the code would be 16 bits. The advantage in coding is more noticeable when having a larger set of symbols. For the set X , the entropy is

$$H(X) = - \sum_{i=1}^4 p_i \log_2 p_i = -\frac{1}{8} \log_2 \frac{1}{8} - \frac{1}{8} \log_2 \frac{1}{8} - \frac{2}{8} \log_2 \frac{2}{8} - \frac{4}{8} \log_2 \frac{4}{8} = \frac{7}{4} \text{ bits,}$$

and the expected codeword length, L , is

$$L = \sum_{i=1}^4 p_i l_i = \frac{1}{8} 3 + \frac{1}{8} 3 + \frac{2}{8} 2 + \frac{4}{8} 1 = \frac{7}{4} \text{ bits,}$$

which tells us that this code reaches the entropy limit in (2.1). The total length of the code can be calculated as the length of the uncoded sequence times L . In this example the sequence has the length 8. The total length of the code is then $8 \cdot L = 8 \cdot \frac{7}{4} = 14$ bits. \square

Exercise 1. *Huffman code the sequence A B R A C A D A B R A.*

Exercise 2. *Decode the binary code*

0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 1

using the Huffman table on page 9.

2.3.2 Arithmetic Coding

The main idea of arithmetic coding is to represent the entire sequence of symbols by a subinterval of the unit interval, i.e. $[0, 1]$. This is done by dividing the unit interval into sections representing the probability of each symbol in the sequence. Then the section corresponding to the first symbol in the sequence is also divided proportionally to the probability of each

symbol. An interval is chosen according to the next symbol. This process is then repeated for every symbol in the sequence. Any number in the resulting very small interval can represent the entire sequence. The, in binary, shortest fraction that is inside that interval is the code for the sequence.

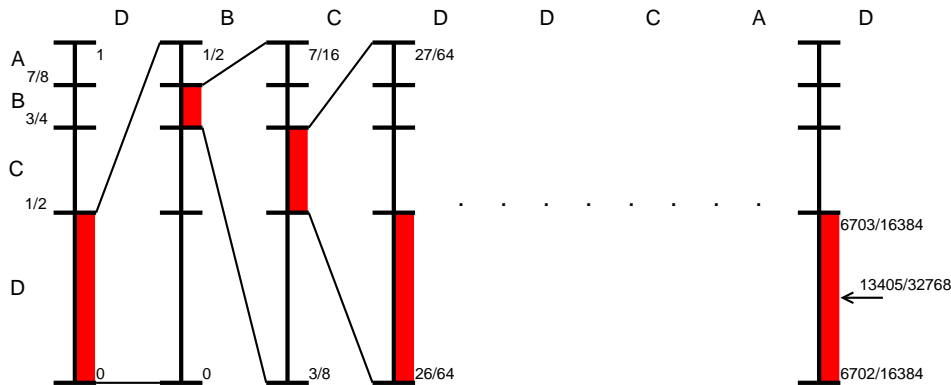


Figure 2.6: An example of how Arithmetic coding can be implemented. Every section is divided proportional to the probabilities of the symbols. The limits of the following interval is chosen according to the next symbol in the sequence starting with $[0, 1]$.

The sequence D B C D D C A D, that is coded in Figure 2.6 is the same as in the Huffman example in Section 2.3.1. Here the resulting binary code is

$$0.011010001011101_2,$$

which can be saved in 15 bits since we know that the first digit is always zero.

In this example the Huffman code gives a better result than the arithmetic code. This is not always the case. Arithmetic codes often get closer to the entropy limit, than Huffman codes. This is because they are not limited by the integer length of the codewords, but only the integer length of the code itself. Arithmetic coding is also better than Huffman coding when one of the symbols has a probability far from $\frac{1}{2^k}$, for some $k > 0$. The drawback for arithmetic coding is that it is a slower and more complex algorithm than Huffman coding [15].

2.4 Run-Length Encoding

In run-length encoding (RLE) [16, pp. 483–484], sub-sequences with only one symbol (a run) are coded as the symbol and a number, representing the length of the sub-sequence. If the data contains long runs of the different symbols, then RLE is efficient and gives good compression, otherwise it may even increase the data-size. There are many variations of RLE. For example,

you can choose between coding only one symbol, several symbols or all of the symbols with RLE. It depends on the data which of these variants is most suitable.

An example where only one symbol coded with RLE is the following. Here there are long runs of zeros and only the zeros are coded as 0 and the length of the run. The sequence

5 4 3 0 0 0 0 0 5 7 3 0 0 0 0 0 4 2 5 7 0 0 0 0 9 1

is coded as

5 4 3 0 5 5 7 3 0 6 4 2 5 7 0 4 9 1

and compressed from 27 to 18 elements. If a run is longer than the maximum value for representing a run, then the run has to be divided into smaller parts. The suitable maximum value for the runs are dependent on the application.

2.5 Discrete Wavelet Transform

The goal of wavelet transforms is to see patterns that would not be visible in the original function. It can be seen as an alternative to the Fourier transform but using short wavelets (localized waves) instead of infinite waves. Instead of oscillating forever, wavelets drop to zero [17, p. xix].

Example. In order to convey the very basics of the discrete wavelet transform, suppose a signal consists of four values $\{x_1, x_2, x_3, x_4\}$. A sum and differences transform would be created in the following way

$$\begin{aligned} a_1 &= \frac{1}{2}(x_1 + x_2) \\ d_1 &= \frac{1}{2}(x_1 - x_2) \\ a_2 &= \frac{1}{2}(x_3 + x_4) \\ d_2 &= \frac{1}{2}(x_3 - x_4), \end{aligned}$$

where a denotes average and d difference. As we can see, this transformation is invertible and thus all information in the existing signal remains. The inverse is as following

$$\begin{aligned} x_1 &= a_1 + d_1 \\ x_2 &= a_1 - d_1 \\ x_3 &= a_2 + d_2 \\ x_4 &= a_2 - d_2. \end{aligned}$$

The hope is that the signal is easier to store efficiently if it is saved as a number of averages and differences. This is generally true if the signal does not vary a lot. More iterations could be done on the resulting averages in order to further reduce the number of averages and thus saving as many differences as possible. \square

In order to do a discrete wavelet transform on a more general signal, $f(x)$, we first have to put some minor constraints on the signal. The function $f(x)$ has to be discretely divided into 2^N sections, where $f(x)$ is constant within each of these sections and $f(x)$ has to have values only within the interval $[0, 1]$. If the signal has values outside this interval, it can be adjusted to fit using a scaling of the signal.

Every discrete wavelet transform has two mother functions, from which more functions are defined. These two are the mother scaling function, $\phi(t)$, and the mother wavelet function, $\psi(t)$. They are both zero outside the unit interval, $[0, 1]$. From these two the functions

$$\psi_{n,j}(t) = 2^{n/2}\psi(2^n t - j) \quad j = 0, \dots, 2^n - 1$$

and

$$\phi_{n,j}(t) = 2^{n/2}\phi(2^n t - j) \quad j = 0, \dots, 2^n - 1$$

are defined, where n denotes the horizontal scale and j is the horizontal shift. There are a few characteristics of the scaling and the wavelet functions worth noting. When keeping n constant, the functions are all pairwise orthogonal, i.e. for every n , a set of basis functions is defined. This means that the signal can be described as coefficients multiplied with the different basis functions. All of the functions are also zero outside an interval of length 2^{-n} and the shift, j , is limited by the scale, n , ensuring that no functions can be shifted in a manner where it has non-zero values outside of the unit interval. One can see the wavelet transform as pairs of high-pass and low-pass filters, where the wavelet function can be regarded as the high-pass filter and the scaling function as the low-pass filter [18, p. 122].

A common and simple type of wavelet is the Haar wavelet, which has the mother wavelet function [19]

$$\psi(t) = \begin{cases} 1 & 0 \leq t < 1/2, \\ -1 & 1/2 \leq t < 1, \\ 0 & \text{otherwise} \end{cases}$$

and the mother scaling function

$$\phi(t) = \begin{cases} 1 & 0 \leq t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

Using the Haar scaling function above, we can describe the signal $f(t)$ as

$$f(t) = f(0)\phi_{N,0}(t) + \cdots + f(2^N - 1)\phi_{N,2^N-1}(t)$$

or in the more general case as

$$f(t) = a_{N,0}\phi_{N,0}(t) + \cdots + a_{N,2^N-1}\phi_{N,2^N-1}(t),$$

where the a -coefficients are formed by

$$a_{n,j} = \int_0^1 f(t)\phi_{n,j}(t)dt \quad \forall j.$$

For convenience, let us define the function entirely described by a -coefficients and scaling functions as

$$g_n(t) = a_{n,0}\phi_{n,0}(t) + \cdots + a_{n,2^n-1}\phi_{n,2^n-1}(t),$$

where n is the same vertical scale as of the scaling functions, $\phi_{n,j}$, describing $g_n(t)$. In the very first iteration $f(t) = g_N(t)$ is true.

We now have 2^N coefficients that are unchanged from the original representation of the signal and are ready to begin the wavelet transform. The following procedure will introduce another type of coefficients, here called the wavelet coefficients.

In order to calculate the first iteration of the wavelet transform, the scale of the scaling function and the wavelet function has to change. The new scale is $n = N - 1$. This means that the functions are twice as wide as before and fewer are needed in order to cover the unit interval. To find the wavelet coefficients, or for short, d -coefficients, the following integral is calculated

$$d_{n,j} = \int_0^1 g_{n+1}(t)\psi_{n,j}(t)dt \quad \forall j.$$

The a -coefficients are, just as above, calculated by

$$a_{n,j} = \int_0^1 g_{n+1}(t)\phi_{n,j}(t)dt \quad \forall j,$$

with the slight change of using the $g(t)$ -function from the previous iteration, hence the $n + 1$ as index. This leaves us with 2^{N-1} a -coefficients and an equal amount of d -coefficients. The signal $f(t)$ can now be described as

$$f(t) = a_{N-1,0}\phi_{N-1,0}(t) + \cdots + a_{N-1,2^{N-1}-1}\phi_{N-1,2^{N-1}-1}(t) + \\ d_{N-1,0}\psi_{N-1,0}(t) + \cdots + d_{N-1,2^{N-1}-1}\psi_{N-1,2^{N-1}-1}(t),$$

which makes it clear that from the scaling and wavelet function, together with the a - and d -coefficients, the original function can easily be obtained. The function $f(t)$ can also be described as

$$f(t) = g_{N-1}(t) + d_{N-1,0}\psi_{N-1,0}(t) + \cdots + d_{N-1,2^{N-1}-1}\psi_{N-1,2^{N-1}-1}(t).$$

With this, everything is prepared for the next iteration with $n = N - 2$. The d -coefficients are stored and the process continues with the a -coefficients. The iterations continues until the scale, n , has reached 1 or until a sufficient amount of iterations for the application in mind has been reached.

2.5.1 Biorthogonal Wavelets

Not all wavelets have to fulfil the orthogonality criteria stated above. There exist a type of wavelets called biorthogonal wavelets. A biorthogonal wavelet is when the wavelet transform is invertible but the wavelet is not necessary orthogonal [20].

One such biorthogonal wavelet is the 5-3 integer wavelet described in *Reversible Integer-to-Integer Wavelet Transforms for Image Coding* by Michael David Adams [21, ch. 5]. The transform is not only biorthogonal, but gives also integer results and is therefore suitable in lossless compression. The wavelet is based on the input signal $x(n)$, a low-pass sub-band signal $a(n)$ and a high-pass sub-band signal $d(n)$. There are also two quantities needed, $a_0(n) = x(2n)$ and $d_0(n) = x(2n + 1)$. In each iteration of the wavelet the low- and high-pass signals are given as

$$\begin{aligned} d(n) &= d_0(n) - \left\lfloor \frac{1}{2} (a_0(n+1) + a_0(n)) \right\rfloor, \\ a(n) &= a_0(n) + \left\lfloor \frac{1}{4} (d(n) + d(n-1)) + \frac{1}{2} \right\rfloor, \end{aligned}$$

where $\lfloor \cdot \rfloor$ is the rounding down operator. The low-pass sub-band signal, $a(n)$, is then the input signal to the next iteration.

2.5.2 2D Wavelet Transform

The difference between using the wavelet transform on a one dimensional signal and a two dimensional signal (an image), is simply the need to apply the wavelet transform in two steps. First the transform is applied to every row in the image, as they can separately be seen as one-dimensional signals. The next step is to apply the transform once more but along the columns of the results from the previous step [18, p. 113]. This is then iteratively done until the low-frequency part is small enough. In Figure 2.7 the schematic structure of the transformed image can be seen and in Figure 2.8 the result from an actual transform. The marked rectangles in the former figure are called sub-bands or sub-blocks.

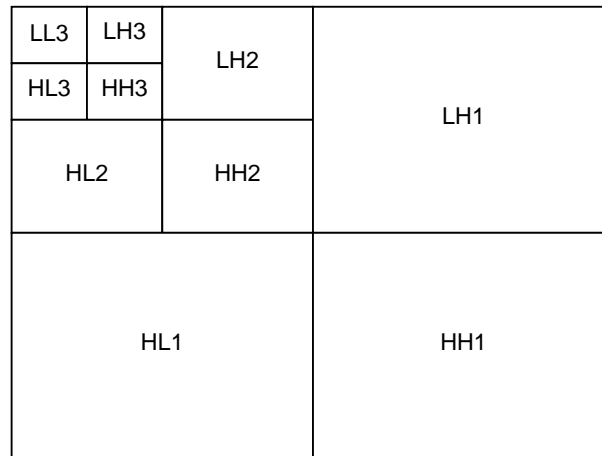


Figure 2.7: Structure of the transformed image. L denotes low-pass and H denotes high-pass. The number indicates in which iteration it was formed.

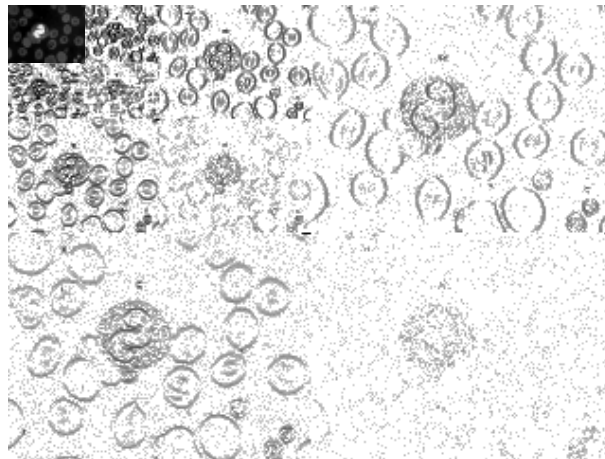


Figure 2.8: The 2D Haar discrete wavelet transform of the red color channel of an example image. The intensities are the logarithm of the absolute value of the original transform to enhance the structure of the result. Black represents in this figure high values and white low values.

2.5.3 Wavelet Transform for Compression

When using the wavelet transform for compression, the goal is to acquire large amounts of zeros among the resulting coefficients. When a lossy compression is acceptable, one can set a threshold for excluding coefficients below a certain amplitude and thus setting them to zero. The real benefit of a resulting transformation containing a lot of zeros is seen when applying the Zerotree algorithm for coding.

2.5.4 Embedded Zerotree Wavelet

The Embedded Zerotree Wavelet, or EZW for short, is an algorithm used in order to encode the wavelet coefficients.

In this representation it is said that each pixel has four children. If the parent is located at coordinate (i, j) , then the four children are at $(2i - 1, 2j - 1)$, $(2i - 1, 2j)$, $(2i, 2j - 1)$ and $(2i, 2j)$. This means also that the children to a parent in block LH3 will be in LH2, and the grandchildren in LH1. The same is valid for HL and HH [18, p. 136]. A visualisation of this is found in Figure 2.9.

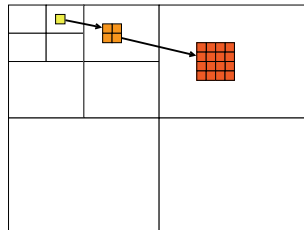


Figure 2.9: This figure shows the spatial relationship between a parent (yellow), its children (orange) and grandchildren (red).

Worth noting is that the LL block (top-left corner), is in the EZW algorithm left untouched. That means that that block has to be saved separately.

A special characteristic in the result of the wavelet transform is a certain grouping of zeros and this grouping is what the Zerotree representation is taking advantage of. More precisely, when a pixel is zero it is more likely for its children to be zero as well.

In order to understand the Zerotree algorithm, the concept bit plane has to be introduced. A bit plane is a matrix which only contains ones and zeros. Since every integer can be expressed as a binary number, every matrix containing integers can be expressed in a number of bit planes. Say if all the integers in the matrix can be expressed with a maximum of 8 bits. Then, eight bit planes has to be formed in order to fully represent the matrix. The first bit plane contains the most significant bit and the last bit plane the least significant bit.

The encoding of the Zerotree begins by dividing the result from the 2D discrete wavelet transform into a number of bit planes. The first ones of the bit planes usually have a very good structure for Zerotree coding since they are sparser.

The Zerotree encoder uses five different symbols in order to represent the bit planes, they are: [18, p. 136]

- POS** Indicates that the coded unit is significant and positive.
- NEG** Indicates that the coded unit is significant and negative.
- IZ** Stands for “isolated zero”. That means that the unit is zero but has at least one significant unit as child or grandchild in any generation.
- ZR** Is a Zerotree root and tells us that the entire subtree of children is zero.
- ZT** Means that this is the child of a Zerotree root and is thus zero.

The beauty of having this description is that only four of the symbols are needed, {**POS**, **NEG**, **IZ**, **ZR**}, in order to describe the bit planes completely, since when decoding the **ZR** symbol it will expand to a subtree filled with **ZT** symbols.

2.6 Regression Analysis

In many fields, the connection between, and dependence on, different stochastic variables is of great interest. The pursuit of this connection is regression analysis [22, p. 358]. When the dependence between variables is known, one can in a greater extent and with higher accuracy, estimate unknown variables given already known ones. There exist a lot of models for different types of regression. Among these, three are presented further down.

2.6.1 Using Regression in Image Compression

If the value of a pixel could be fully predicted, then this pixel would not be needed to be saved in order to recreate the original image. If this predicted value could be used in order to fully predict the next pixel, and the next, and the next, only a few initial pixels would need saving in order to recreate the complete image. Alas, predictors like that does not exists. It is possible to rather accurately predict the value of the next pixel, but the prediction would more often than not need to be corrected. Here comes the beauty of regression analysis methods in image compression. The corrections for the predictions are usually easier to code than the original image, since the variance of the correction values are, in general, smaller than in the original image. With better methods for prediction, the variance of the correction values tend to be smaller and thus even easier to code.

2.6.2 Simple Linear Regression

Simple linear regression is fitting a straight line to a set of data points, or in one dimension higher, fitting a plane to the set of data points. A simple linear regression model is, as stated in *Sannolikhetssteori och statistikteori med tillämpningar* [22, p. 359],

$$y_i = \alpha + \beta x_i + e_i,$$

where (x_i, y_i) are the data points, i indicates the index for the data points, α and β are the model parameters and $e_i \in N(0, \sigma)$ the random errors. With a small standard deviation, σ , the vertical distance between the data points and the regression line tend in general to be smaller. The parameters are, in a least square sense, estimated as

$$\hat{\beta} = \frac{S_{xy}}{S_{xx}}$$

and

$$\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x},$$

where

$$S_{xy} = \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}),$$

$$S_{xx} = \sum_{i=1}^n (x_i - \bar{x})^2$$

and \bar{x} as well as \bar{y} represents the mean of x and y respectively. When $\hat{\alpha}$ and $\hat{\beta}$ are calculated, the estimated value of y , $\hat{\mu}_0$, can be obtained from the data point x_0 together with

$$\hat{\mu}_0 = \hat{\alpha} + \hat{\beta}x_0.$$

2.6.3 Autoregressive Model

The autoregressive (AR) model is a process used in many applications. This model, as opposed to simple linear regression, can take periodicities in the input signal into account when predicting unknown outputs. The model is also known in the field of signal processing as an Infinite Impulse Response (IIR) filter. The process is defined in the book *Stationary Stochastic Processes* [23, p. 132] as:

Let $A(z)$ be a stable polynomial of degree p . A stationary sequence $\{y_t\}$ is called an AR(p)-process with generating polynomial $A(z)$, if the sequence $\{e_t\}$, given by

$$y_t + a_1 y_{t-1} + \cdots + a_p y_{t-p} = e_t, \quad (2.2)$$

is a white noise sequence with $E[e_t] = 0$, constant variance $V[e_t] = \sigma^2$, and e_t uncorrelated with y_{t-1}, y_{t-2}, \dots . The variables e_t are the innovations to the AR-process. In a Gaussian stationary AR-process, also the innovations are Gaussian.

A generating polynomial $A(z)$ is defined as follows. Let

$$\{a_0 = 1, a_1, \dots, a_p\}$$

be real coefficients and define the polynomial

$$A(z) = a_0 + a_1 z^{-1} + \dots + a_p z^{-p},$$

in the complex variable z . Equation (2.2) can now be written as

$$A(z)y_t = e_t. \quad (2.3)$$

In order to predict future values using this model, the AR-coefficients in the generating polynomial have to be calculated. This is done, as described in *Time Series Analysis and Signal Modeling* [24, p. 230], by using the definition of an AR-process and setting up a system of equations in different positions in time. The equation at time t

$$y_t = -a_1 y_{t-1} - \dots - a_p y_{t-p} + e_t$$

can be expressed as

$$y_t = \mathbf{x}_t^T \boldsymbol{\theta} + e_t,$$

where

$$\mathbf{x}_t = [-y_{t-1}, \dots, -y_{t-p}]^T$$

and

$$\boldsymbol{\theta} = [a_1, \dots, a_p]^T.$$

At time t , there are t different measurements available. By using p of them and the following notation

$$\mathbf{y}_t = [y_{t-p}, \dots, y_t]^T, \quad (2.4)$$

$$\mathbf{X}_t = [\mathbf{x}_{t-p}, \dots, \mathbf{x}_t]^T, \quad (2.5)$$

the coefficients $\boldsymbol{\theta}$ for the AR-model can be estimated. They are estimated, in a least squares sense, by

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}_t^T \mathbf{X}_t)^{-1} \mathbf{X}_t^T \mathbf{y}_t. \quad (2.6)$$

2.6.4 ARX-Model

The ARX-model is an extension of the AR-model, where realisations of other processes also are taken into account when predicting the current process. If there is noticeable correlation between the investigated process and another process, is it a good idea to take advantage of this information and that is what this model does. In comparison to Equation (2.3), the ARX-model is defined as [24, p. 197]

$$A(z)y_t = B(z)u_t + e_t,$$

where u_t is the external signal and $B(z)$ is the polynomial governing the effect on the external signal on the process. More precise

$$B(z) = b_d z^{-d} + b_{d+1} z^{-(d+1)} + \dots + b_{d+s} z^{-(d+s)}$$

where d denotes the delay of the input and s the order of the external signal.

The procedure of estimating the coefficients of this model is very similar to the procedure in Section 2.6.3, but with

$$\mathbf{x}_t = \left[-y_{t-1}, \dots, -y_{t-p}, u_{t-d}, \dots, u_{t-(d+s)} \right]^T$$

and

$$\boldsymbol{\theta} = [a_1, \dots, a_p, b_d, \dots, b_{d+s}]^T$$

instead of the corresponding definitions in 2.6.3. The Equations (2.4), (2.5) and (2.6) are treated in the exact same manner as in the previous section.

2.7 PCA - Principal Component Analysis

Principal component analysis, or PCA for short, is a type of multivariate analysis that was invented by Karl Pearson in 1901. The primary objective of PCA is to reduce the amount of relevant dimensions in a multidimensional set of data [25, p. 1].

This is achieved by an orthogonal change of coordinates, where the data is decorrelated and most information (highest variance) is along the first basis vector, second most information is along the second basis vector and so forth. Along the last dimensions there is almost no information at all, i.e. noise.

A convenient method for finding the new basis for the principal components is to use Singular Value Decomposition, SVD [25, p. 45]. Let X be an $n \times p$ matrix, i.e. X consist of n data vectors, each in p dimensions. Then X can be factorized as

$$X = U \Sigma V^T, \quad (2.7)$$

where U is a $n \times r$ matrix and V is a $p \times r$ matrix. U and V have orthonormal columns. Σ is a diagonal $r \times r$ matrix, where the diagonal elements are the singular values of X and r is the rank of X .

The important result from this is that the columns of V are the base vectors, i.e. the principal components, and thus the new basis for the decorrelated set. In order to transform the data set, the following formula is used

$$\hat{X} = X V^T, \quad (2.8)$$

where \hat{X} is in the new basis where the data set is decorrelated.

Example. Suppose we have a two dimensional data set X . X is normally distributed with high correlation between the x and y coordinates, see Figure 2.10a. This data set can be decorrelated using the formula in (2.8), where V has been calculated with Singular Value Decomposition from (2.7). The decorrelated data is presented in Figure 2.10b in the basis given by the columns of V . \square

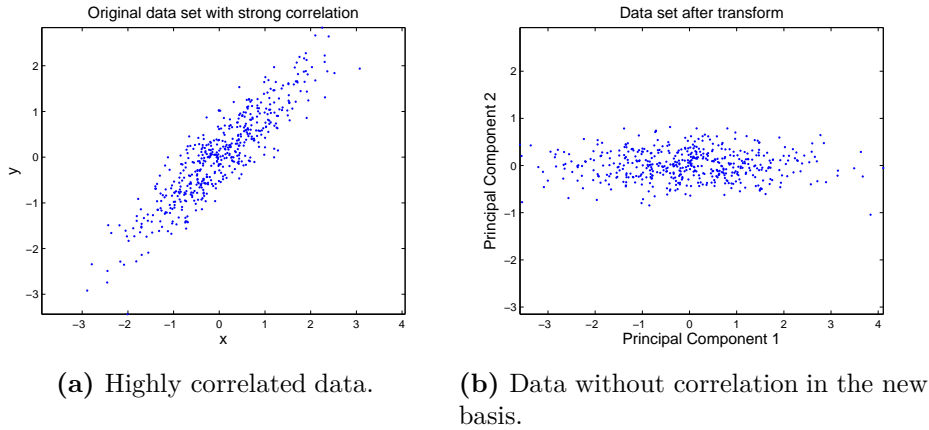


Figure 2.10: Example of a correlated data set before and after the transformation into a basis, where the data set is no longer correlated.

2.8 Reversible Integer Transformation

When transforming the color space in the context of lossless compression, it is necessary to be able to reverse the process exactly. Also, the result of the transform has to be integers, since integers are much easier to handle in the rest of the compression steps. If an interesting linear color space transform is found, it can be modified to fulfil the above criteria. How this modification is made is explained here.

One way to convert an arbitrary linear transform, with determinant ± 1 , to a nonlinear transform that is bijective for integers is described by P. Hao and Q. Shi in [26] and extended by S. Pei and J. Ding in [27]. A combination of them is used here. It is stated that any $N \times N$ matrix with determinant ± 1 can be factorized into a product of $N + 1$ single-row elementary reversible matrices (SERM), two permutation matrices and two diagonal sign matrices [27]. A SERM can be written as

$$\mathbf{S}_m = \mathbf{I} + \mathbf{e}_m \mathbf{s}_m^T, \text{ for } 1 \leq m \leq N,$$

where \mathbf{e}_m is the m th column of the identity matrix and $\mathbf{s}_m = \{s_{m,n}\}$, $n = 1, 2, \dots, N$ is a N -dimensional column matrix where the m th element is 0. \mathbf{S}_0 is defined as $\mathbf{S}_0 = \mathbf{I} + \mathbf{e}_N \mathbf{s}_0^T$, where \mathbf{s}_0 has its N th element as 0. For

example, \mathbf{S}_2 is of the form

$$\mathbf{S}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ s_{2,1} & 1 & s_{2,3} & s_{2,4} & \cdots & s_{2,N-1} & s_{2,N} \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}.$$

The factorization of an $N \times N$ matrix \mathbf{A} can be written as

$$\mathbf{D}_1 \mathbf{P}_1 \mathbf{A} \mathbf{P}_2 \mathbf{D}_2 = \mathbf{S}_N \mathbf{S}_{N-1} \cdots \mathbf{S}_1 \mathbf{S}_0,$$

or equivalently

$$\mathbf{A} = \mathbf{P}_1^T \mathbf{D}_1 \mathbf{S}_N \mathbf{S}_{N-1} \cdots \mathbf{S}_1 \mathbf{S}_0 \mathbf{D}_2 \mathbf{P}_2^T,$$

where \mathbf{P}_1 and \mathbf{P}_2 are permutation matrices and \mathbf{D}_1 and \mathbf{D}_2 are diagonal sign matrices, i.e. diagonal matrices with ± 1 on the diagonal. The permutation matrices and diagonal sign matrices are used for pivoting the matrix \mathbf{A} , so the matrix elements are at suitable places and have suitable signs for the factorization operation. To be able to handle the cases where the determinant of $\mathbf{D}_1 \mathbf{P}_1 \mathbf{A} \mathbf{P}_2 \mathbf{D}_2$ is -1 , the bottom right element of \mathbf{S}_0 , that is $\mathbf{S}_0(N, N)$, is changed from 1 to $k = \det(\mathbf{D}_1 \mathbf{P}_1 \mathbf{A} \mathbf{P}_2 \mathbf{D}_2)$.

From this factorization, the transformation $\mathbf{A}\mathbf{x}$ can be computed as a reversible integer transformation. Start by multiplying \mathbf{x} with $\mathbf{D}_2 \mathbf{P}_2^T$ from the left. Then multiply with the SERMs one at a time starting from the right-most, \mathbf{S}_0 . Because each SERM only has one row that is different from an identity matrix, only one of the variables that are going to be transformed is then changed at each matrix multiplication. The expression for the changed variable is

$$x_m + \sum_{n \neq m} s_{m,n} x_n, \quad (2.9)$$

where x_n , $1 \leq n \leq N$, are the variables that are going to be transformed. To keep the variables as integers, the sum in (2.9) is rounded with an arbitrary rounding operator, $[\cdot]$, as following,

$$x_m + \left[\sum_{n \neq m} s_{m,n} x_n \right].$$

This makes sure that the variables in each step always are integers. The first variable that is changed has to be saved as a temporary variable, \mathbf{Z} , and is used in the expressions for the new variables. After multiplying with all SERMs, this results in a transformation that is reversible and has integer values. It will lead to $N + 1$ expressions, one for the temporary variable and N for the transformed integer variables. To get the final transformed

variables, multiply the array with the N transformed integer variables with $\mathbf{P}_1^T \mathbf{D}_1$.

To do the reverse transformation, multiply the transformed variables with $\mathbf{D}_1 \mathbf{P}_1$ and start from the expression of the last one of the transformed variables. From that one, the temporary variable Z can be expressed using only transformed variables. Then, continue to the second last expression and extract the old variable that is not rounded from it. Continue upwards in the expressions until all old variables are extracted. To get correct order and sign of the original variables, multiply them with $\mathbf{D}_2 \mathbf{P}_2$.

Example. When $N = 4$, the decomposition can be written as

$$\mathbf{D}_1 \mathbf{P}_1 \mathbf{A} \mathbf{P}_2 \mathbf{D}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a_1 & a_2 & a_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ b_1 & b_2 & 1 & b_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ c_1 & 1 & c_2 & c_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & d_1 & d_2 & d_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ e_1 & e_2 & e_3 & k \end{bmatrix},$$

where \mathbf{P}_1 and \mathbf{P}_2 are permutation matrices, \mathbf{D}_1 and \mathbf{D}_2 are diagonal sign matrices and $k = \det(\mathbf{D}_1 \mathbf{P}_1 \mathbf{A} \mathbf{P}_2 \mathbf{D}_2)$. The permutation matrices and diagonal sign matrices are found by testing all the $147\,456 = (4!)^2 (2^4)^2$ possible combinations and choosing those that give the best result for the specific purpose.

The parameters, $a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3, d_1, d_2, d_3, e_1, e_2$ and e_3 can be found by solving the system of equations acquired when comparing the elements in $\mathbf{D}_1 \mathbf{P}_1 \mathbf{A} \mathbf{P}_2 \mathbf{D}_2$ with the elements in the five SERMs multiplied together. The reversible integer transformation from the old color space, R, G1, G2, B, to the new color space, C1, C2, C3, C4, is then done using the parameters in the following way. Here $[\cdot]$ denotes an arbitrary rounding operator to integers.

$$\begin{aligned} \begin{pmatrix} X1 & X2 & X3 & X4 \end{pmatrix}^T &= \mathbf{D}_2 \mathbf{P}_2^T \begin{pmatrix} R & G1 & G2 & B \end{pmatrix}^T \\ Z &= kX4 + [e_1X1 + e_2X2 + e_3X3] \\ Y1 &= X1 + [d_1X2 + d_2X3 + d_3Z] \\ Y2 &= X2 + [c_1Y1 + c_2X3 + c_3Z] \\ Y3 &= X3 + [b_1Y1 + b_2Y2 + b_3Z] \\ Y4 &= Z + [a_1Y1 + a_2Y2 + a_3Y3] \\ \begin{pmatrix} C1 & C2 & C3 & C4 \end{pmatrix}^T &= \mathbf{P}_1^T \mathbf{D}_1 \begin{pmatrix} Y1 & Y2 & Y3 & Y4 \end{pmatrix}^T. \end{aligned}$$

The inverse transformation, from $C1, C2, C3, C4$ to $R, G1, G2, B$ is found using the equations above from the last to the first equation. In each step, excluding the steps with permutation matrices, the variable that is not rounded, is extracted. The inverse transformation is

$$\begin{aligned} \begin{pmatrix} Y1 & Y2 & Y3 & Y4 \end{pmatrix}^T &= \mathbf{D}_1 \mathbf{P}_1 \begin{pmatrix} C1 & C2 & C3 & C4 \end{pmatrix}^T \\ Z &= Y4 - [a_1 Y1 + a_2 Y2 + a_3 Y3] \\ X3 &= Y3 - [b_1 Y1 + b_2 Y2 + b_3 Z] \\ X2 &= Y2 - [c_1 Y1 + c_2 X3 + c_3 Z] \\ X1 &= Y1 - [d_1 X2 + d_2 X3 + d_3 Z] \\ X4 &= kZ - k[e_1 X1 + e_2 X2 + e_3 X3] \\ \begin{pmatrix} R & G1 & G2 & B \end{pmatrix}^T &= \mathbf{P}_2 \mathbf{D}_2 \begin{pmatrix} X1 & X2 & X3 & X4 \end{pmatrix}^T. \end{aligned}$$

□

Exercise 3. *Derive the reversible integer transformation equations for the case $N = 2$. Assume $\mathbf{P}_1 = \mathbf{P}_2 = \mathbf{D}_1 = \mathbf{D}_2 = \mathbf{I}$, for convenience.*

Exercise 4. *Find the inverse to $\mathbf{S}_m = \mathbf{I} + \mathbf{e}_m \mathbf{s}_m^T$.*

Chapter 3

Our Compression Strategies

In this chapter we present some of the strategies we have investigated. The ones presented here are mostly the best ones, according to compression level, but also some interesting ones, that were not so successful strategies. All but one of the strategies are doing lossless compression.

The strategies are based on the fact that the output from the camera is a mosaic image, where there are only intensity information from one color channel in each pixel, see Section 2.2. Almost all of our strategies share the same framework as described in Figure 3.1. First the image is divided into three or four color channels, that each are coded with some lossy compression method. The difference in each pixel between the lossy image and the original image is then computed and saved in a difference image. The difference images are compressed with entropy coding. The compressed file contains the lossy compressed images as well as the entropy coded difference images. The strategies differ mostly in the lossy compression step, but also how the mosaic image is divided into color channels. We have also developed a lossy compression method, which only uses lossy compression on the background, but is lossless on the foreground.

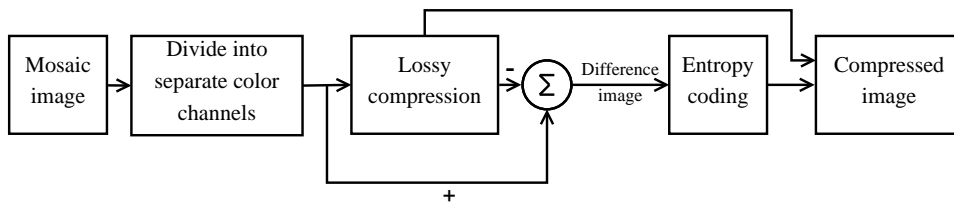


Figure 3.1: The framework for almost all strategies we have tried. The main differences between the strategies are in the lossy compression step and how the mosaic image is divided into color channels.

3.1 Strategy 1 – Color Separation and Lossless JPEG 2000

In this strategy, the mosaic image is divided into its three color channels, red, green and blue, as described in Section 2.2.1. Both simple merging and structure separation is tested. After this, the lossless version of JPEG 2000 is used on each of the color channels.

3.2 Strategy 2 – Difference Image with Standard Lossy Compression

This strategy, as Strategy 1, uses the color separation described in Section 2.2.1. The difference from Strategy 1 is that after the color separation step, the smaller images for each color channel are lossy compressed by either JPEG or JPEG 2000, instead of using the lossless version of JPEG 2000. After this step, the lossy images are compared to the original images, and one difference image for each color channel is created, see Figure 3.2. These difference images contain fewer values than the original images and the values are centered around zero, with zero as the most common value. To take care of this feature, the difference images are Huffman coded, so the most common values get the shortest codewords. The lossy images and the coded difference images are then saved together with the Huffman tables for the difference images.

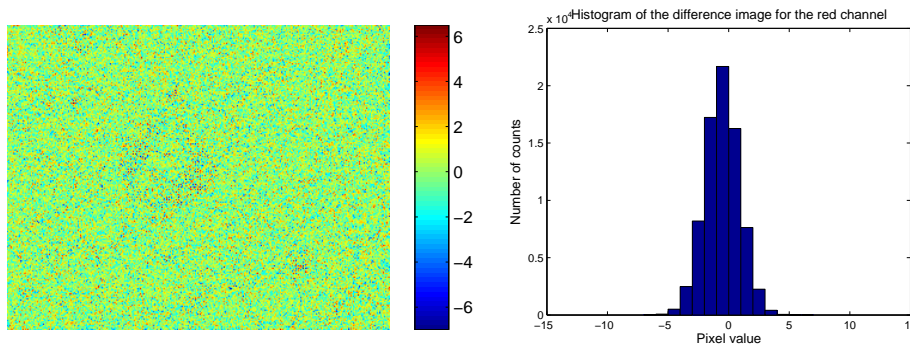


Figure 3.2: The difference image for the red color channel in one of the test images using JPEG 2000 as lossy compression together with its histogram.

As mentioned in the previous chapter about JPEG (Section 2.1.1) and JPEG 2000 (Section 2.1.2), the methods each have a parameter. For JPEG it is a quality parameter and for JPEG 2000 a compression ratio parameter. To choose the parameters, we optimized on the final compressed file size and chose the parameters that gave the smallest file sizes. The parameters chosen were, for JPEG, 65, and for JPEG 2000, 13.

3.3 Strategy 3 – Wavelet Transform with ARX

When applying a one-level 2D wavelet transform to an image, the result is four sub-bands. Compared to a mosaic image, the four sub-bands are related to the different color channels included in the mosaic image. Thus, a wavelet transform can be used to divide the mosaic image into four color channels. In this strategy, we use the 5-3 integer wavelet, described in Section 2.5.1. Each of the four sub-bands created by the wavelet transform, LL, LH, HL and HH, are then estimated using an ARX-model, see Section 2.6.4. For each row, an ARX-model is created using the two rows above, which implies that each row that is estimated has its own model parameters. To estimate a pixel with this ARX-model, the three pixels from the row above, centered at the current pixel, and the two before on the same row are used. The pixels used are shown in Figure 3.3.

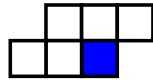


Figure 3.3: The current pixel marked with blue is estimated using the five pixels above and to the left.

For this to work we need to know the first two rows and the first two and the last column beforehand. Instead of saving them as they are, we only save the 2×2 top-left pixels of the images. From these pixels, the two first rows and the two first and last columns are estimated using linear regression, see Section 2.6.2. To estimate the rows, start with the first two pixel values, and estimate the third pixel value using them. Then the fourth pixel value is estimated using the second and third values. This continues until the end of the rows. The columns are then estimated in the same way, using the two pixel values above each pixel that is going to be estimated.

To be able to get the original image back, the rounded difference between the estimated pixel values, both those estimated with ARX and those with linear regression, and the real values are saved in difference images.

The four difference images are then Huffman coded and saved together with their Huffman tables and the four sets of 2×2 top-left pixels.

3.4 Strategy 4 – Cross Channel ARX

This method is a slight variation of Strategy 3. Instead of using a wavelet transform, in order to decorrelate the color channels, we divide the mosaic image into its color channels, as in Figure 2.2, and use structure separation for the green channel, see Figure 2.4c. One of the green channels is estimated using an ARX-model as in the former section. The other three channels use the information from that channel in their estimation. The

second green channel is coded with neighbouring pixels from the first green channel as external input in the ARX-model. The red and blue channels use information from both of the two green channels as external input, see Figure 3.4.

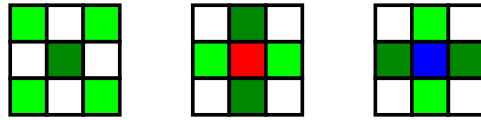


Figure 3.4: The squares are cut-outs from the mosaic image. The center pixels are the pixels that are to be estimated. The surrounding filled pixels are the pixels from other color channels used as external input in the ARX-model. These images illustrate the differences between how the neighbours are selected for the different color channels, in the following order: Second green, red and blue.

As in the previous section, difference images are created and later Huffman coded. An example of how the difference images from this method can look like is presented in Figure 3.5. Notice the increased spatial correlation in regions with cells in comparison to Figure 3.2.

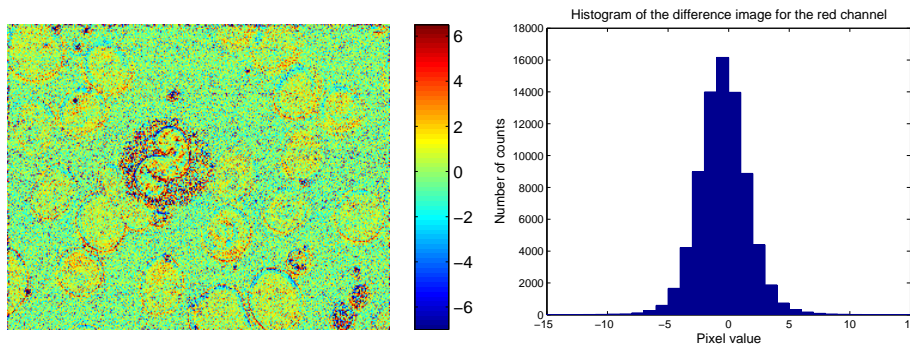


Figure 3.5: The difference image for the red color channel in one of the test images using a ARX-model with information from neighbouring pixels as lossy compression together with its histogram. Compare the structure to Figure 3.2.

3.4.1 Strategy 4.5 – Limited Cross Channel ARX

This strategy is a minor variation on the previous strategy *Cross Channel ARX*. The alterations made are to divide the different color channels into smaller images with the width 80 pixels. The *Cross Channel ARX* strategy are then applied to these images in order to acquire a number of difference images. The resulting difference images are then stitched together and Huffman coded.

This method also needs a set of 2×2 top-left pixels to be stored for each 80 pixel part of the color channels.

3.5 Color Transformations

To see if another color space than RGB is more suitable when compressing the images, we tried the two color transformations below. The transformations are applied to Strategy 2, before the lossy compression step. When using a color transformation in a lossless compression method, you have to use a transformation that maps integers to integers and is reversible.

3.5.1 Strategy 5 – YDgCoCg Transformation

YDgCoCg [28] is a color transformation that transforms the four channels extracted using structure separation into four new color channels Y, Dg, Do and Co. The transform is described by the following equations:

$$\left\{ \begin{array}{l} \text{Co} = \text{R} - \text{B} \\ \text{Dg} = \text{G}_2 - \text{G}_1 \\ u = \text{B} + \left\lfloor \frac{\text{Co}}{2} \right\rfloor \\ v = \text{G}_1 + \left\lfloor \frac{\text{Dg}}{2} \right\rfloor \\ \text{Cg} = v - u \\ \text{Y} = u + \left\lfloor \frac{\text{Cg}}{2} \right\rfloor \end{array} \right. \Leftrightarrow \left\{ \begin{array}{l} u = \text{Y} - \left\lfloor \frac{\text{Cg}}{2} \right\rfloor \\ v = u + \text{Cg} \\ \text{G}_1 = v - \left\lfloor \frac{\text{Dg}}{2} \right\rfloor \\ \text{B} = u - \left\lfloor \frac{\text{Co}}{2} \right\rfloor \\ \text{G}_2 = \text{Dg} + \text{G}_1 \\ \text{R} = \text{B} + \text{Co} \end{array} \right.$$

It is important to perform each operation in the exact same order as above to keep the reversibility. Because Dg, Co and Cg can be negative, those channels are shifted, so that they become positive, before compressing them lossy with JPEG or JPEG 2000. The numbers, that the different channels are shifted by, are saved in the file. Y is always positive, so there is no need for shifting. The difference images are then Huffman coded.

3.5.2 Strategy 6 – PCA Integer Transformation

A transformation that is more specific for the images we are compressing, is one created using a PCA of a subset of our test images. The PCA is based on the four color channels extracted using structure separation.

A PCA transform does not map integers to integers, but in a lossless compression method, a transformation needs to give integer values. If the transformed values are rounded to integers, the transform is no longer reversible, which it also has to be in a lossless compression method. Both

these problems can be solved using the estimation into a reversible integer transform, described in Section 2.8. Because the matrix for the PCA transform has the determinant ± 1 , it can be estimated, using five SERMs, two permutation matrices and two diagonal sign matrices, into a reversible integer transform. The PCA matrix used is

$$\begin{pmatrix} -0.54842442 & -0.52655953 & 0.64464027 & 0.080028042 \\ -0.55934215 & 0.42569473 & -0.039977193 & -0.71015620 \\ -0.54632694 & 0.42046905 & -0.20749407 & 0.69403100 \\ -0.29646844 & -0.60392463 & -0.73470187 & -0.087148324 \end{pmatrix},$$

and by trying all possible combinations of permutation and diagonal sign matrices, the best combination according to compression level was

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \quad D_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$P_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \quad D_2 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

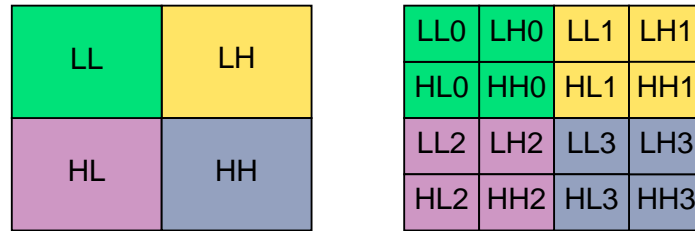
This estimation into a reversible integer transform gives four new color channels. If the new color channels are negative, they are shifted so they become positive, and the numbers that they are shifted with are saved. The shifted color channels are compressed lossy using JPEG or JPEG 2000. The lossy channels are compared with the uncompressed channels and the difference images are created and Huffman coded.

3.6 Strategy 7 – Mallat Packet Transform with Zerotree

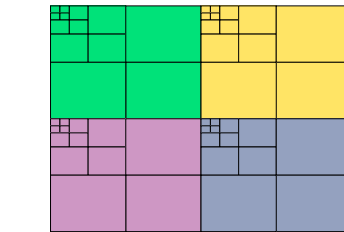
This method is based on Mallat packet transform described by Ning Zhang et al. in [29] and is a strategy where the lossy compression step in Figure 3.1 is omitted. The first step, to divide the mosaic image into color channels, is performed with a one-level 2D wavelet transform of the mosaic image, as in Strategy 3. This yields four sub-bands, LL, LH, HL and HH, each with low frequency content, see Figure 3.6a. To remove both spatial and spectral correlation of the mosaic image, another 2D wavelet transform is applied to each of the four sub-bands. This result in 16 sub-bands in total, see Figure 3.6b. Most of the energy, i.e. high amplitude values, in the image

is in the four sub-bands that have only low frequency content, LL0, LL1, LL2 and LL3. The other twelve sub-bands have high frequency, but low amplitude, and will not be further compressed before the Zerotree stage.

For each of the four low frequency sub-bands, a three-level 2D wavelet transform is performed, and the resulting image is of the form seen in Figure 3.6c.



(a) A one-level 2D wavelet transform gives the following structure. (b) Another one-level 2D wavelet transform of each sub-band in (a).



(c) For each of the four sub-bands, LL0, LL1, LL2 and LL3 from (b) a three-level 2D wavelet transform is performed.

Figure 3.6: The Mallat packet transform, stepwise from (a) to (c).

The wavelet transform used for all transformations above is the 5-3 integer wavelet, described in Section 2.5.1.

The four wavelet transformed sub-bands, LL, LH, HL and HH, are then compressed with one Zerotree each, see Section 2.5.4. The results from the Zerotree are then compressed further using arithmetic coding. In this strategy, the saved parts are the LL block in the last wavelet level in each sub-band, and the result from the arithmetic coding.

3.7 Lossy Strategy – High-Quality Lossy Compression

Unlike our other methods, this one gives lossy compression. The main idea behind this method is that the background is not as important as the foreground. With this in mind, the foreground, i.e. all the cells, is still compressed losslessly, while the background is not. This method is based on Strategy 2.

The first step is to find where the background of the image is. The algorithm for doing this is supplied by CellaVision and creates a mask using a threshold on the first green color channel. The threshold is calculated by first finding the rightmost peak of the histogram (Figure 3.7) and then finding the closest minimum to the left of this peak in a suitable distance. If the closest minimum is within a distance of 7 to 20 from the peak, this minimum is chosen as the threshold. Otherwise the threshold is set as 13 before the peak.

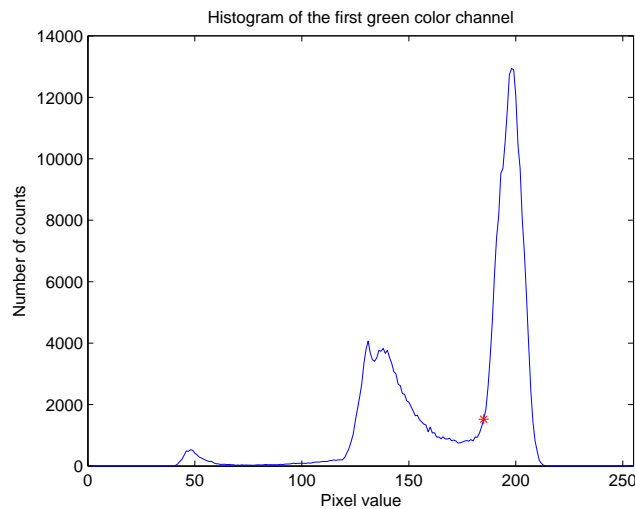


Figure 3.7: The histogram used to determine the threshold for constructing the mask. It is calculated from the first green color channel of the image in Figure 3.8. The red star marks where the threshold for the background is. In this example it is 13 before the rightmost peak.

After the threshold has been used in order to acquire a binary mask, a morphological dilation, i.e. making the masks larger, is applied in order to be sure that important information is not excluded. An example of the resulting binary mask is shown in Figure 3.8.

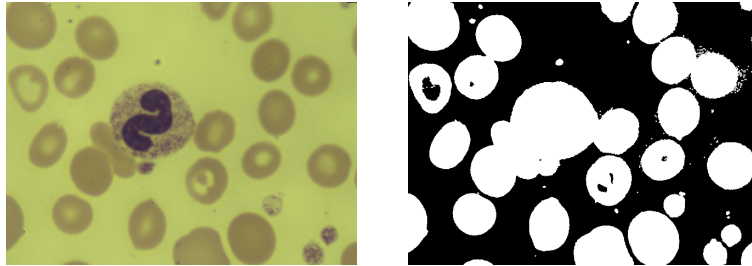


Figure 3.8: A cell image and the corresponding binary mask after the morphological dilation.

In the same manner as in Strategy 2, a standard lossy method, here JPEG 2000, is used to compress the color channels and then these images are compared to the original images and difference images are created. The next step is the one that introduces errors in the compression and makes the method lossy. We now set every background pixel in the difference images to zero, thus having a heavily increased amount of zeroes in the difference images in comparison to the lossless method, see Figure 3.9.

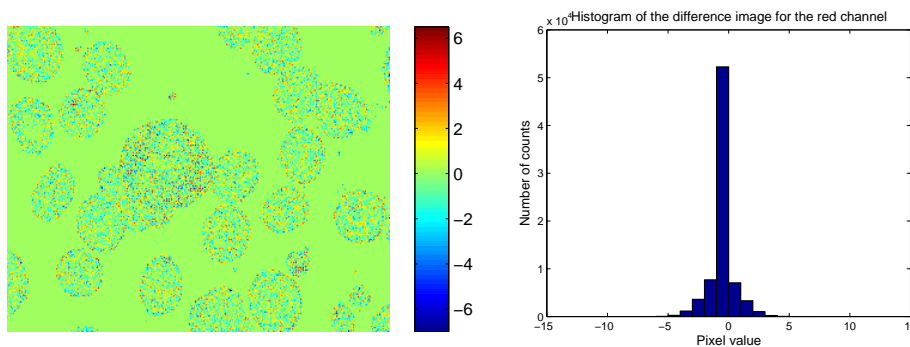


Figure 3.9: This is the difference image for the red color channel after removing the background together with its histogram. Compare with the difference image in Figure 3.2, from the lossless method described in Section 3.2.

Since we now have large fields of zeroes in the difference images, we apply run length encoding of the zeroes (Section 2.4) to compress the difference images. After this, ordinary Huffman coding is applied to the already compressed difference images and the results are saved together with the lossy JPEG 2000 images.

Chapter 4

Results

Each of the compression strategies presented in the previous chapter has been tested on a library of 604 test images, which are interpolated RGB images with the dimension 480×640 pixels. The results of the test are here presented as the mean bit-rate of the images. The bit-rate is a measure of how many bits per pixel (bpp) that are used when an image is saved.

An uncompressed RGB image has the bit-rate of 24 bpp, every pixel has three color channels, each saved with 8 bpp. We know that the images in the test set have been captured with a camera using a specific color filter array (the Bayer filter described in Section 2.2), and that the interpolation algorithm used for the images is reversible. Because of that, we can easily compress the images to a bit-rate of 8 bpp, by just recreating the mosaic image that is the output from the camera, and save the value of each pixel uncompressed using 8 bits.

4.1 Lossless Compression

The best lossless strategy among those we have tried is Strategy 4, *Cross Channel ARX*. In that strategy, the mosaic image is divided into four color channels using structure separation for the green channel. Then each color channel is coded using ARX with external information from the other color channels. The difference images from ARX are then compressed using Huffman coding. This compression method gives a bit-rate of 3.0409 bpp, i.e. the size of the compressed image is 12.67% of the original size of the uncompressed RGB image.

In Figure 4.1, we present the bit-rates of the best variations of the different lossless strategies. The rightmost strategy is the result when doing an ordinary compression using lossless JPEG 2000 on the mosaic image, it is included for comparison. The results from all the different variations of the strategies described are collected in Table 4.1.

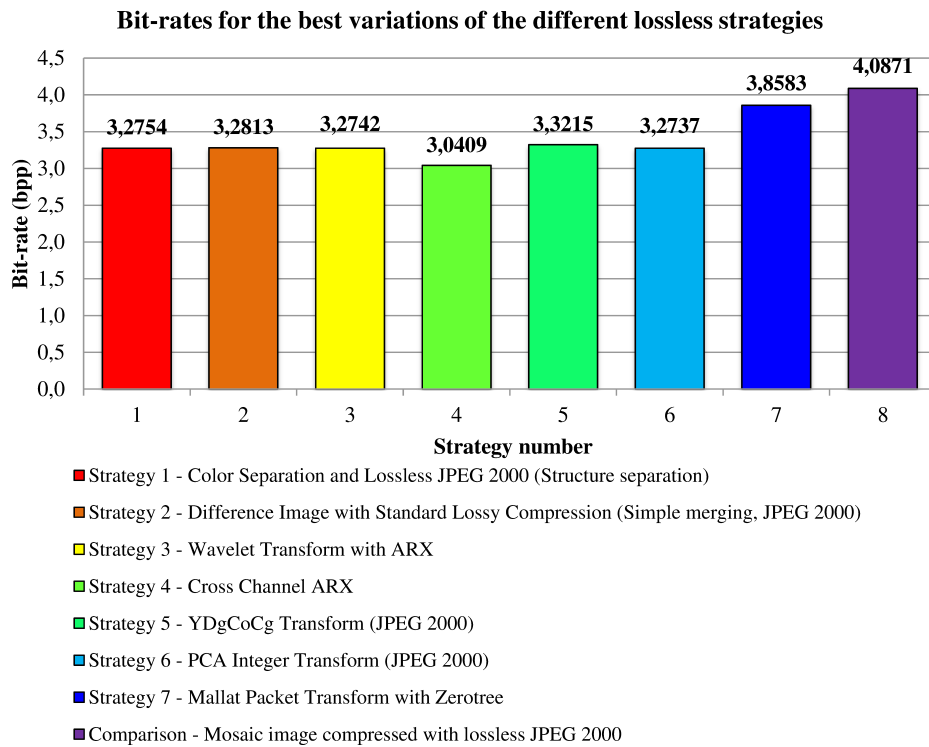


Figure 4.1: The bit-rates of the best variations of the different lossless strategies from Chapter 3. In parenthesis the variations of the strategy used are presented.

The strategies differ in what is saved in the compressed file. We divide the possible content into three categories: *Lossy*, *Huffman code* and *Huffman table*. In Figure 4.2, the file content for Strategy 2 and Strategy 4 is shown.

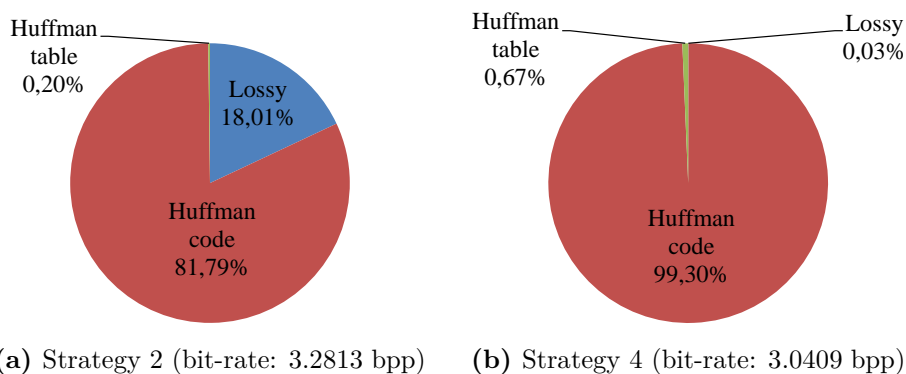


Figure 4.2: The content in the compressed file divided between *Lossy*, *Huffman code* and *Huffman table*.

Table 4.1: The results from all the different variations of the lossless compression methods that were tested. They are presented as the mean bit-rate (in bpp) and mean file size (in kB) of the test set. The size of the uncompressed test images are on disk $480 \cdot 640 \cdot 3 \cdot 8 = 7\,372\,800$ bits = 900 kB.

Method	Division into color channels	Lossy compression	Entropy coding	Size [kB]	Bit-rate \pm std [bpp]
Uncompressed					
Original image compressed with lossless JPEG 2000					
Original image compressed with PNG					
Mosaic image compressed with lossless JPEG 2000					
Strategy 1 (Section 3.1)	Simple merging	Lossless JPEG 2000	directly	122.83	3.2755 ± 0.140
	Structure separation	Lossless JPEG 2000	directly	122.83	3.2754 ± 0.140
Strategy 2 (Section 3.2)	Simple merging	JPEG	Huffman coding	133.49	3.5596 ± 0.199
	Simple merging	JPEG 2000	Huffman coding	123.06	3.2813 ± 0.180
	Structure separation	JPEG	Huffman coding	133.95	3.5721 ± 0.200
	Structure separation	JPEG 2000	Huffman coding	123.10	3.2827 ± 0.180
Strategy 3 (Section 3.3)	Wavelet transform	ARX	Huffman coding	122.78	3.2742 ± 0.106
Strategy 4 (Section 3.4)	Structure separation	Cross channel ARX	Huffman coding	114.03	3.0409 ± 0.135
Strategy 4.5 (Section 3.4.1)	Structure separation	Cross channel ARX	Huffman coding	115.84	3.0890 ± 0.109
	YDgCoCg	JPEG	Huffman coding	134.67	3.5911 ± 0.201
Strategy 5 (Section 3.5.1)	YDgCoCg	JPEG 2000	Huffman coding	124.56	3.3215 ± 0.171
	Integer PCA	JPEG	Huffman coding	129.33	3.4488 ± 0.186
Strategy 6 (Section 3.5.2)	Integer PCA	JPEG 2000	Huffman coding	122.76	3.2737 ± 0.144
	Mallat packet transform with Zerotree		Arithmetic coding	144.69	3.8583 ± 0.186

A test was also carried out using the best of the compression algorithms, Strategy 4, on an ordinary photo, i.e. not a cell image. The image still has a known color filter array (Bayer filter). This is done in order to see how specialised the algorithm is on cell images. On the picture seen in Figure 4.3a, Strategy 4 achieved a bit-rate of 4.9986 bpp, i.e. the size of the compressed image is 20.83% of the original size of the interpolated image.

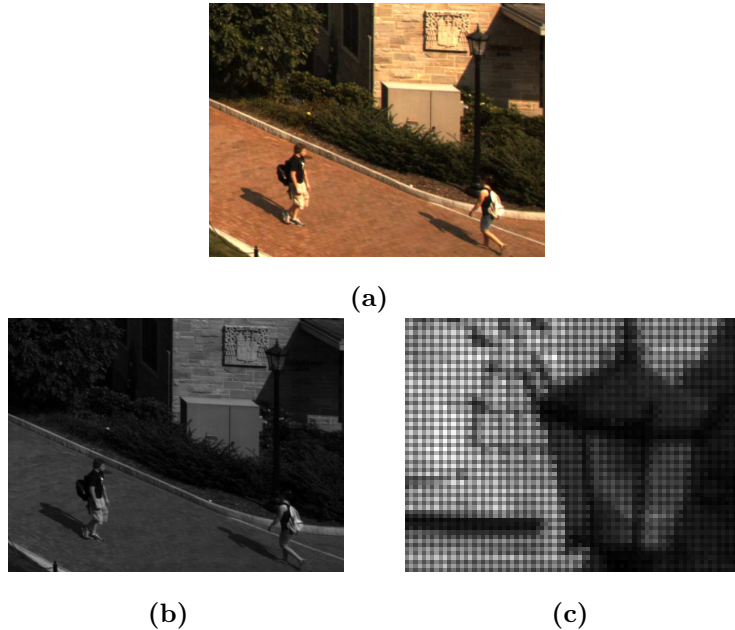


Figure 4.3: The image used to examine how our algorithm perform on a regular image, i.e. not a cell image. (a) is the interpolated image, (b) is the extracted mosiac image and (c) is a magnified part of the mosaic image in order to show the mosaic structure.

4.2 Lossy Compression

The lossy compression method achieved a compression rate of 2.4451 bpp with a peak signal-to-noise ratio, PSNR, 48.05 dB. The PSNR is a metric for how close the decompressed image is to the original image. It is defined as [30]

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right),$$

where MAX_I is the maximum *possible* value of the image, here 255. MSE , the mean squared error, is defined as

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2,$$

where I is the original uncompressed $m \times n$ image and K is the decompressed lossy image. The PSNR of a lossy image compression is generally in between 30dB and 50dB, higher values are better. In Table 4.2, a comparison for the PSNR is made between our lossy compression and lossy compression with JPEG and JPEG 2000. The parameters of JPEG and JPEG 2000 have been adjusted in order to get a compression rate as close as possible to the compression rate of our method.

Table 4.2: A comparison of peak signal-to-noise ratio between our lossy compression and two other compression standards at about the same compression rate. The results are the mean over the whole test set of 604 images.

Method	bpp	PSNR
JPEG	2.6000	44.55 dB
JPEG 2000	2.3855	47.44 dB
Proposed lossy compression	2.4451	48.05 dB

Chapter 5

Discussion and Conclusion

To put our results into perspective, we need to recall that a reversible interpolation of the mosaic image is used in CellaVisions's analysis systems. With this in mind, the process of compressing the uncompressed color RGB image with a bit-rate of 24 bpp to a bit-rate of 8 bpp is straight forward, given that the interpolation algorithm is known.

The interesting part is what comes after this step. It is still quite easy to further reduce the size of the images using knowledge of the distribution and by decorrelating the information. Almost every approach using the distribution of values and trying to decorrelate the information could achieve a bit-rate close to 3.6 bpp. The real challenge was to find the best way to tweak the methods in order to push the compression rate further down.

Since the total number of colors used in the set of cell images supplied by CellaVision was a lot less than the possible 2^{24} , it was hoped that this characteristic could explicitly be used. Having unused addresses in the color space seemed like a waste of space. This was implicitly dealt with by Huffman coding. By definition, there is no address to a value that is not used in a Huffman code. In Strategy 5 and Strategy 6, color transforms were used mainly to decorrelate the color space. As a side effect of these strategies, the number of colors in each color channel was reduced.

5.1 Choice of method

Most of the strategies presented used structure separation in order to handle the green channel from the mosaic image, but the simple merging achieved, by minimal margin, better results in Strategy 2. The reason for choosing structure separation was that it was more convenient with all the color channels having the same shape.

Regarding the lossy part of the compression in Strategy 2, it is stated how the quality and compression parameters for JPEG and JPEG 2000 respectively were chosen. These parameters were also used in the other

strategies that used JPEG and JPEG 2000 as lossy compression. Since the parameters chosen was a good choice for Strategy 2, we have assumed that the parameters would be suitable for the rest of the strategies as well.

In quite a few of our algorithms, we based the lossy compression on JPEG 2000 or JPEG and tried to compress the corresponding difference images in different manners. The focus was on the difference image because the size of the Huffman coded difference image overshadows the size of the lossy part. In Strategy 2, for instance, is the lossy part about 18% and the Huffman code is 82% of the file size (Figure 4.2a). When we started focusing on statistical models in Strategy 3 and Strategy 4, the aim was still to construct a better difference image for the Huffman algorithm to handle. However, the idea of using linear regression from only four pixels instead of saving two whole rows and three columns of the image lead to that the lossy part shrunk to 0.03% (Figure 4.2b), but the Huffman code grew as a consequence. In Strategy 2, although the encoded difference image is smaller than in Strategy 4, there are also four lossy JPEG images that has to be saved. In Strategy 3 and Strategy 4, the lossy part to save is only 16 pixels, the 2×2 top-left pixels for each color channel.

When compressing the difference images with Huffman coding, different Huffman tables for each color channel had to be supplied. The nature of Huffman coding is to adapt the code to the specific distribution of the symbols to be coded. If we had static Huffman tables we would not need to include the tables in every compressed image, but on the other hand the resulting code using these would not be optimal. The question at hand is “Would static Huffman tables have been worth the non optimal code it would produce?” The answer to this would be “Probably not.” As can be seen in Figure 4.2, the supplied Huffman tables are a very small part of the compressed file (0.20% and 0.67% in Strategy 2 and Strategy 4, respectively). When a Huffman table that is not tailored for the data to be coded is used, there are two possibilities. Either, the compression rate stays the same as with the tailored Huffman table, or the compression rate gets worse. The probability is higher for it to be worse. How much worse is hard to say, but because all the difference images are centred around zero and the distribution does not deviate a lot, it would probably not be very bad. The end note is that, on average, a maximum of 0.2% or 0.67%, depending on strategy, of the compressed file size could be saved if static Huffman tables were used. But, that gain could easily be overshadowed by a longer code for the difference images.

5.2 Comparisons

If we choose to compare Strategy 2 with Strategy 4, is it clear that the total compression is better with Strategy 4. But there are other benefits

with Strategy 2 which are not conveyed by the bit-rate. Since there is a lossy image saved together with the difference image, we could choose to just decode and present the lossy part of the image. This could save time and computation resources when lossless quality is not needed, e.g. when the image is displayed for a user.

Strategy 4 is closely related to Strategy 4.5 and the former is able to achieve a lower bit-rate. There are some drawbacks with Strategy 4 that are addressed in Strategy 4.5. Firstly, the parameters of the ARX-model are based on the entire previous row, meaning pixels on the far right has influence on how a pixel on the far left one row below is modelled. Secondly, it does not scale well with larger images, since the matrices used in finding the model parameters are growing linearly with the width of the image and thus makes the matrix multiplications more complex. The hope was that the more concentrated set of prior knowledge (only about 80 pixels instead of the full width of the image) would give a better prediction and thus a better bit-rate. This did not happen because for every 80 pixel, the border had to be linearly regressed and thus having a worse distribution than the ARX-model. This affected the final bit-rate negatively, so that Strategy 4 still achieves better compression. Because of the lower matrix multiplication complexity of Strategy 4.5, this would probably execute faster in a real world scenario.

In Section 4.1 we presented how Strategy 4, *Cross Channel ARX*, performed on an image that did not depict cells. As expected, the algorithm did not do very well on this picture. The reason behind this is probably a higher presence of sharp edges. Rapid changes in the picture are difficult for the ARX-model to predict. Our focus has been entirely on cell images, so we see this as a good thing, as our compression algorithms are not supposed to be used for general purpose compression.

Regarding our proposed lossy compression algorithm, the big advantage with this lossy compression, in comparison to JPEG or JPEG 2000, is that we have control over which parts of the image that are lossy compressed and which parts that are losslessly compressed. That control comes with a price, the size of the compressed image. There are still questions of how the artificial neural networks, used in CellaVision's products, would classify the images compressed with this method. Since the background extraction function of the cells was provided by CellaVision, the features sent to the artificial neural network for analysis would probably be identical to the features extracted from the original images. Given that the cells in the proposed method are saved losslessly there would not be any difference in the classification. But still, this remains untested. The images compressed with the lossless methods would naturally be classified in the same way as the original images.

5.3 Possible Improvements

An improvement that would be interesting to test for the lossy strategy would be to modify the run length algorithm. Now the runs are only made for each row, but with some clever thinking we could make a 2D run length algorithm. Instead of seeing the runs as one dimensional sequences, we could see the runs as squares in the image. This would probably give us fewer runs to code, but be more complex to implement than the current algorithm.

The parameter for JPEG 2000 was chosen to maximize the compression rate in the lossless Strategy 2. The same parameter is used in the high-quality lossy compression strategy. To improve the lossy strategy, the parameter should be adapted for this strategy. In the lossy strategy it is not only the size of the compressed file that is important. It is also important what amount of loss that is acceptable in the background of the images. Because of the many factors that has to be taken into account, this optimization problem is much harder to solve than the one in Strategy 2.

Another improvement of the lossy strategy is to investigate if it is possible to use different levels of lossy compression on the background and on the foreground. If it is possible the background could be compressed harder and the foreground could be compressed to an optimal level regarding the size of the difference image and the lossy image together.

5.4 Conclusion

General compression algorithms have been developed and improved for decades. But, it has in this master's thesis, been shown that there are still gains to be made developing custom compression techniques for specific purposes, in this case for blood cell images.

Appendix A

Example of Cell Images

In order to convey how the images that were treated in this master's thesis looked like, some examples are shown below in Figure A.1

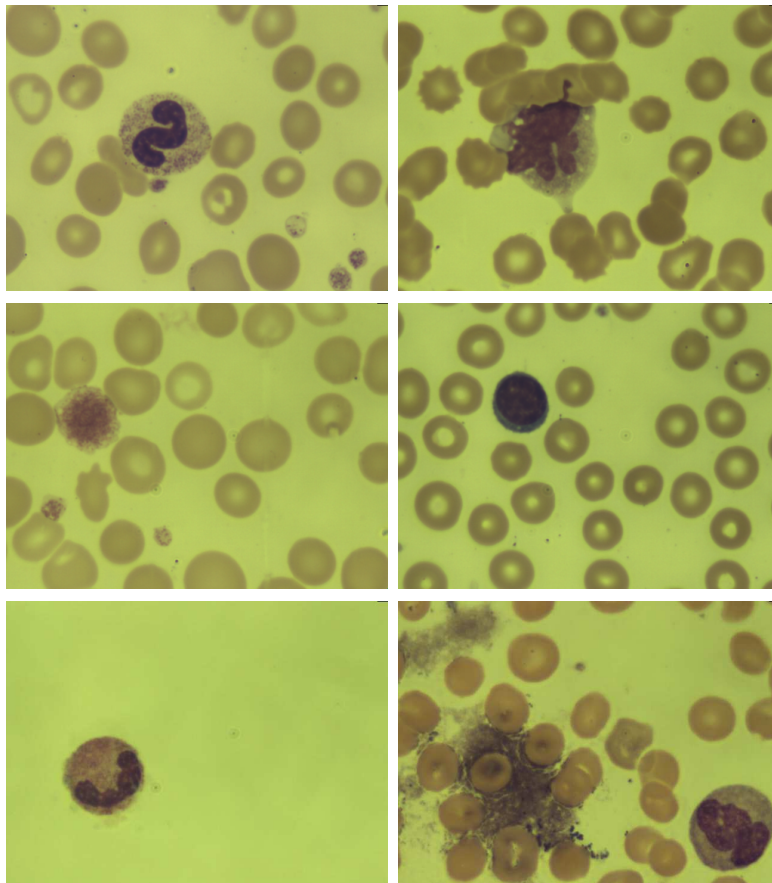


Figure A.1: Examples of different cell images our algorithms were developed for.

Appendix B

Solutions to Exercises

Exercise 1 (p. 10). The sequence A B R A C A D A B R A contains the symbols in the set $X = \{A,B,C,D,R\}$ with the probability distribution $\left\{\frac{5}{11}, \frac{2}{11}, \frac{1}{11}, \frac{1}{11}, \frac{2}{11}\right\}$. There are two possible Huffman trees for this set, which are shown in Figure B.1.

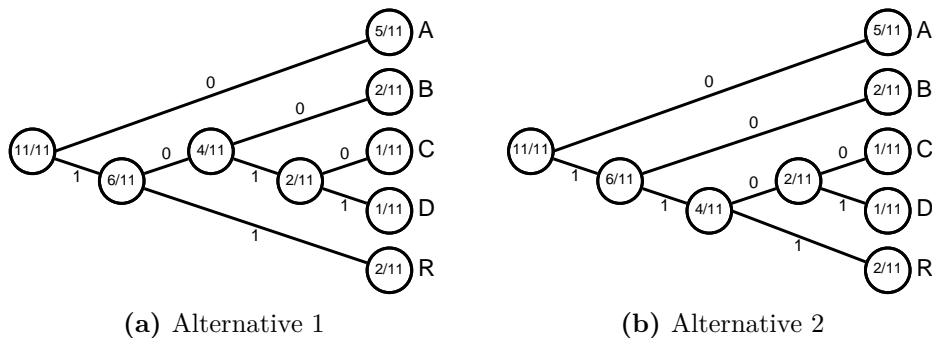


Figure B.1: The two possible alternatives of Huffman trees for the set $X = \{A,B,C,D,R\}$ with probability distribution $\left\{\frac{5}{11}, \frac{2}{11}, \frac{1}{11}, \frac{1}{11}, \frac{2}{11}\right\}$.

The trees above give the following Huffman tables:

Alternative 1		Alternative 2	
Symbol	Codeword	Symbol	Codeword
A	0	A	0
B	100	B	10
C	1010	C	1100
D	1011	D	1101
R	11	R	111

The sequence A B R A C A D A B R A is, using the Huffman tree in Alternative 1, coded as

0 1 0 0 1 1 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 1 0

and, using the tree in Alternative 2, as

0 1 0 1 1 1 0 1 1 0 0 0 1 1 0 1 0 1 0 1 1 1 0.

Both alternatives give codes of the length 23 bits and in this application there are no right or wrong choice of alternative. \square

Exercise 2 (p. 10). The decoded sequence is

B A D C A B A C.

\square

Exercise 3 (p. 25). When $N = 2$ and $\mathbf{P}_1 = \mathbf{P}_2 = \mathbf{D}_1 = \mathbf{D}_2 = \mathbf{I}$, the matrix \mathbf{A} can be factorized as

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ a & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & b \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ c & k \end{pmatrix},$$

where $k = \det \mathbf{A}$. Multiply $(\mathbf{X}_1 \ \mathbf{X}_2)^T$ with the SERMs above, one at a time, and round the part of each expression that are multiplied with a , b or c . Also, save the first changed variable, Z .

$$\begin{aligned} \begin{pmatrix} 1 & 0 \\ c & k \end{pmatrix} \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{pmatrix} &= \begin{pmatrix} \mathbf{X}_1 \\ k\mathbf{X}_2 + c\mathbf{X}_1 \end{pmatrix} \approx \begin{pmatrix} \mathbf{X}_1 \\ k\mathbf{X}_2 + [c\mathbf{X}_1] \end{pmatrix} = \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{Z} \end{pmatrix} \\ \begin{pmatrix} 1 & b \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{Z} \end{pmatrix} &= \begin{pmatrix} \mathbf{X}_1 + b\mathbf{Z} \\ \mathbf{Z} \end{pmatrix} \approx \begin{pmatrix} \mathbf{X}_1 + [b\mathbf{Z}] \\ \mathbf{Z} \end{pmatrix} = \begin{pmatrix} \mathbf{Y}_1 \\ \mathbf{Z} \end{pmatrix} \\ \begin{pmatrix} 1 & 0 \\ a & 1 \end{pmatrix} \begin{pmatrix} \mathbf{Y}_1 \\ \mathbf{Z} \end{pmatrix} &= \begin{pmatrix} \mathbf{Y}_1 \\ \mathbf{Z} + a\mathbf{Y}_1 \end{pmatrix} \approx \begin{pmatrix} \mathbf{Y}_1 \\ \mathbf{Z} + [a\mathbf{Y}_1] \end{pmatrix} = \begin{pmatrix} \mathbf{Y}_1 \\ \mathbf{Y}_2 \end{pmatrix} \end{aligned}$$

The transform from $(\mathbf{X}_1 \ \mathbf{X}_2)^T$ to $(\mathbf{Y}_1 \ \mathbf{Y}_2)^T$ is

$$\begin{cases} \mathbf{Z} = k\mathbf{X}_2 + [c\mathbf{X}_1] \\ \mathbf{Y}_1 = \mathbf{X}_1 + [b\mathbf{Z}] \\ \mathbf{Y}_2 = \mathbf{Z} + [a\mathbf{Y}_1] \end{cases}$$

and the inverse is

$$\begin{cases} \mathbf{Z} = \mathbf{Y}_2 - [a\mathbf{Y}_1] \\ \mathbf{X}_1 = \mathbf{Y}_1 - [b\mathbf{Z}] \\ \mathbf{X}_2 = k\mathbf{Z} - k[c\mathbf{X}_1]. \end{cases}$$

\square

Exercise 4 (p. 25). The inverse to $\mathbf{S}_m = \mathbf{I} + \mathbf{e}_m \mathbf{s}_m^T$ is $\mathbf{S}_m^{-1} = \mathbf{I} - \mathbf{e}_m \mathbf{s}_m^T$.

\square

Bibliography

- [1] Wikipedia. X-ray. http://en.wikipedia.org/wiki/X_rays, March 2013.
- [2] Wikipedia. Medical imaging. http://en.wikipedia.org/wiki/Medical_imaging, March 2013.
- [3] Oge Marques. Image Compression and Coding. In *Encyclopedia of Multimedia*, pages 318 – 323. Springer, 2008.
- [4] Wikipedia. JPEG. <http://en.wikipedia.org/wiki/JPEG>, March 2013.
- [5] David Austin (American Mathematical Society). Image Compression: Seeing What’s Not There. <http://www.ams.org/samplings/feature-column/fcarc-image-compression>, March 2013.
- [6] Wikipedia. JPEG 2000. http://en.wikipedia.org/wiki/JPEG_2000, March 2013.
- [7] Wikipedia. Portable Network Graphics. http://en.wikipedia.org/wiki/Portable_Network_Graphics, March 2013.
- [8] Wikipedia. DEFLATE. <http://en.wikipedia.org/wiki/DEFLATE>, March 2013.
- [9] Wikipedia. LZ77 and LZ78. http://en.wikipedia.org/wiki/LZ77_and_LZ78, March 2013.
- [10] King-Hong Chung and Yuk-Hee Chan. Lossless Compression of Bayer Color Filter Array Images. In *Computational Photography - Methods and Applications*, pages 83 – 101. CRC Press, 2010.
- [11] Chin Chye Koh, J. Mukherjee, and S.K. Mitra. New Efficient Methods of Image Compression in Digital Cameras with Color Filter Array. *IEEE Transactions on Consumer Electronics*, Vol. 49(No. 4):1448 – 1456, November 2003.

-
- [12] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. John Wiley & Sons, Inc., second edition, 2006.
- [13] David A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, Vol. 40(No. 9):1098 – 1101, 1952.
- [14] Stefan Höst. *Information Theory Engineering*. Department of Electrical and Information Technology, Lund University, 2013. Unfinished working manuscript, can be found at <http://www.eit.lth.se/index.php?ciuid=550&coursepage=3632>.
- [15] Wikipedia. Arithmetic coding. http://en.wikipedia.org/wiki/Arithmetic_coding, March 2013.
- [16] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.
- [17] Gilbert Strang and Truong Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.
- [18] Stephen Welstead. *Fractal and Wavelet Image Compression Techniques*. SPIE, 1999.
- [19] Wikipedia. Haar wavelet. http://en.wikipedia.org/wiki/Haar_wavelet, March 2013.
- [20] Wikipedia. Biorthogonal wavelet. http://en.wikipedia.org/wiki/Biorthogonal_wavelet, May 2013.
- [21] Michael David Adams. *Reversible Integer-to-integer Wavelet Transforms for Image Coding*. PhD thesis, The University of British Columbia, September 2002.
- [22] Gunnar Blom, Jan Enger, Gunnar Englund, Jan Grandell, and Lars Holst. *Sannolikhetsteori och statistikteori med tillämpningar*. Studentlitteratur, 2005.
- [23] Georg Lindgren, Holger Rootzén, and Maria Sandsten. *Stationary Stochastic Processes*. Lund University Centre for Mathematical Sciences Mathematical Statistics, 2009.
- [24] Andreas Jakobsson. *Time Series Analysis and Signal Modeling*. Lund University Centre for Mathematical Sciences Mathematical Statistics, 121015 edition, 2012.
- [25] I.T. Jolliffe. *Principal Component Analysis*. Springer, second edition, 2002.

-
- [26] Pengwei Hao and Qingyun Shi. Reversible integer KLT for progressive-to-lossless compression of multiple component images. In *Proceedings International Conference on Image Processing*, pages 633 – 636, 2003.
- [27] Soo-Chang Pei and Jian-Jiun Ding. Reversible Integer Color Transform. *IEEE Transactions on Image Processing*, Vol. 16(No. 6):1686 – 1691, June 2007.
- [28] Henrique S. Malvar and Gary J. Sullivan. Progressive-to-Lossless Compression of Color-Filter-Array Images using Macropixel Spectral-Spatial Transformation. In *Data Compression Conference (DCC), 2012*, pages 3 – 12, 2012.
- [29] Ning Zhang and Xiaolin Wu. Lossless Compression of Color Mosaic Images. *IEEE Transactions on Image Processing*, Vol. 15(No. 6):1379 – 1388, June 2006.
- [30] Wikipedia. Peak signal-to-noise ratio. http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio, May 2013.