JPEG-like Image Compression using Neural-network-based Block Classification and Adaptive Reordering of Transform Coefficients

by

Hanns-Juergen Grosse

Thesis submitted to the University of Central Lancashire in

partial fulfilment of the requirements for the degree of

Doctor of Philosophy

October 1997

The work presented in this thesis was carried out in the Department of Electrical and Electronic Engineering, University of Central Lancashire, Preston, United Kingdom, in collaboration with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong.

Declaration

I declare that while registered with the University of Central Lancashire for the degree of Doctor of Philosophy I have not been a registered candidate or enrolled student for another award of the University of Central Lancashire, or any other academic or professional institution during the research programme. No portion of the work referred to in this thesis has been submitted in support of any application for another degree or qualification of any other university or institution of learning.

Hanns-Juergen Grosse

Abstract

JPEG-like Image Compression using
Neural-network-based Block Classification and
Adaptive Reordering of Transform Coefficients

by

Hanns-Juergen Grosse

The research described in this thesis addresses aspects of coding of discrete-cosine-transform (DCT) coefficients, that are present in a variety of transform-based digital-image-compression schemes such as JPEG. Coefficient reordering; that directly affects the symbol statistics for entropy coding, and therefore the effectiveness of entropy coding; is investigated. Adaptive zigzag reordering, a novel versatile technique that achieves efficient reordering by processing variable-size rectangular sub-blocks of coefficients, is developed. Classification of blocks of DCT coefficients using an artificial neural network (ANN) prior to adaptive zigzag reordering is also considered.

Some established digital-image-compression techniques are reviewed, and the JPEG standard for the DCT-based method is studied in more detail. An introduction to artificial neural networks is provided.

Lossless conversion of blocks of coefficients using adaptive zigzag reordering is investigated, and experimental results are presented. A versatile algorithm, that generates zigzag scan paths for sub-blocks of any dimensions using a binary decision tree, is developed. An implementation of the algorithm based on programmable logic devices (PLDs) is described demonstrating the feasibility of hardware implementations. Coding of the sub-block dimensions, that need to be retained in order to reconstruct a sub-block during decoding, based on the scan-path length is developed.

Lossy conversion of blocks of coefficients is also considered, and experimental results are presented. A two-layer feedforward artificial neural network trained using an error-backpropagation algorithm, that determines the sub-block dimensions, is described. Isolated nonzero coefficients of small significance are discarded in some blocks, and therefore smaller sub-blocks are generated.

Table of Contents

List of Tables

List of Figures

Acknowledgements

Firstly I would like to thank Martin R. Varley, my Director of Studies, for his consistent support and guidance through all stages of this research project, to which he has generously devoted much time and effort.

Next I would like to thank Trevor J. Terrell, my Second Supervisor, for the experienced guidance and direction he has given to the project.

I would also like to thank Phil Holifield, my Personal Tutor, for reading the first draft of the thesis and for many helpful discussions; and Isaac Y. K. Chan for the contributions to the publications.

I would like to thank the Department of Electrical and Electronic Engineering of the University of Central Lancashire for sponsoring my research studentship and providing the friendly learning environment.

Thanks are due to many members of staff for helping in various ways. I would like to single out vicariously late David Platt, Principal Technician, who is greatly missed.

In addition, I would like to thank Playboy magazine for the kind permission to reproduce the images of Lena Sjoobloom; and Lattice Semiconductor for the kind permission to reproduce the functional block diagram of the GAL16V8.

I wish to thank Bettina for her love and understanding that make me very grateful.

Finally I must thank my parents to whom I dedicate this thesis for their love, and their continuous encouragement and support. Danke!

Chapter 1

Introduction

1.1 Introduction

In this chapter the research project is outlined, and brought in context to related disciplines of telecommunications and computing. Section 1.2 briefly highlights some of the important advances of these technologies. Section 1.3 describes the aims and objectives of the research project, and section 1.4 provides an overview of the thesis. Finally section 1.5 concludes the chapter with a brief summary.

1.2 Background

In 1837 Samuel Morse invented telegraphy, and seven years later he built the first telegraph line; between Washington and Baltimore, USA; which used Morse code. It was in 1851 that the first commercial transmissions using Morse code were established between England and France. In 1875 Alexander Graham Bell invented the telephone (A. Isaacs (ed.) 1997).

In 1920 the introduction of the Bartlane cable picture transmission system reduced the delivery time for newspaper pictures between London, England and New York, USA from one week to three hours using digital signals on transatlantic submarine cables (M. D. McFarlane 1972).

In 1948 William Shockley and co-workers invented the first transistors at Bell Telephone Co.

In 1962 the first active telecommunication satellite, US Telstar 1, was launched and positioned into relatively low elliptical orbit (A. Isaacs (ed.) 1997).

In the late 1970s microcomputers became widely available. These systems, typically having up to 16 KB random-access memory (RAM) and a tape drive, were used to manipulate text and numerical data, but offered limited graphical support. In the USA the Advanced Research Projects Agency Network (ARPANET) was commissioned as an experimental network designed to support military research. ARPANET later became the Internet.

In the 1980s personal computers, constantly growing more powerful, became available for office and home use. These systems, typically having up to 640 KB RAM, and floppy and hard disk drives, supported a large variety of applications, and offered from the late 1980s graphical user interfaces. However, because of enormous amounts of data involved, digital image processing was still limited to dedicated systems; see for example (G. Hall and T. J. Terrell 1987).

In the 1990s tremendous improvement of processing power, and increases of RAM and hard disk storage are transforming personal computers into powerful general-purpose systems suitable for processing digital image data. Additional networking capabilities of office and home computers allow the exchange of data among distant computers. The Internet, connecting a variety of different computers around the world and growing at great pace, changes the way individuals work and communicate; it symbolizes the information technology revolution.

With demand for transmission and storage of information rapidly growing, data compression in general and image compression in particular remain key technologies (N. Jayant et al. 1993); and, therefore, constitute important areas of research.

1.3 Aims and Objectives of the Project

The aims of the research described in this thesis were to investigate and develop appropriate neural-network models for digital image compression, and to develop the use of neural networks in hybrid schemes for image compression exploiting perceptually important features.

The specific objectives were:

- To review important existing image-compression techniques,
- To review important existing neural-network models,
- To develop new image-compression techniques or to improve existing ones, and
- To identify prospective directions for further research.

1.4 Organization of the Thesis

Chapter 2, entitled 'Digital Image Compression', places digital image compression in context to the human visual system and digital image processing, and focuses on some of the available techniques for lossless and lossy compression.

Chapter 3, entitled 'JPEG Still Picture Compression Standard', discusses the Joint Photographic Experts Group (JPEG) still picture compression standard in some detail as this compression standard has been adapted to a new hybrid compression scheme.

Chapter 4, entitled 'Adaptive Zigzag Reordering of Transform Coefficients', describes a new lossless transcoding scheme that adaptively reorders transform coefficients for improved coding efficiency, and includes experimental results to demonstrate the effectiveness of the scheme.

Chapter 5, entitled 'Artificial Neural Networks', introduces neural networks, and describes the backpropagation training algorithm in detail.

Chapter 6, entitled 'Neural-network-based Block Classification', describes a lossy scheme that uses an artificial neural network to classify blocks prior to adaptive zigzag reordering, and includes experimental results to demonstrate the effectiveness of the scheme.

Chapter 7, entitled 'Conclusions and Recommendations for Further Work', summarizes the contributions made by this thesis and offers recommendations for further research directions.

1.5 Summary

Since their invention telecommunications and computing have developed at great pace. The demand for exchanging information continues to grow, therefore data and image compression remain key technologies.

The main objective of the research described in this thesis has been to investigate the application of neural networks to digital image compression, particularly in hybrid schemes.

Chapter 2

Digital Image Compression

2.1 Introduction

This chapter places digital image compression in context to the human visual system and digital image processing, and focuses on some of the available techniques for lossless and lossy compression.

Section 2.2 briefly summarizes the concept of digital image processing, introduces representations of digital images, and outlines a typical generic image-processing system.

Section 2.3 develops the necessity for digital image compression, distinguishes between lossless and lossy techniques, and summarizes the objectives of digital image compression. It introduces the three forms of data redundancy that can be exploited, and outlines a general image-compression model. The section also introduces entropy as a measure of the complexity of an information source.

Section 2.4 provides a very brief functional description of the human visual system, describes four properties as potentially being useful for digital-image-processing applications, and identifies two properties, spatial masking and local processing characteristic, as currently being most significant.

Section 2.5 describes a number of digital-image-compression techniques. It develops the concept of Huffman coding in detail, focuses also on run-length coding, quantization, and transform coding; and enumerates some other techniques.

Section 2.6 is concerned with image quality assessment based on subjective and objective measures. Finally section 2.7 concludes the chapter with a brief summary.

2.2 Digital Image Processing

2.2.1 Motivation for Digital Image Processing

Digital image processing aims to gather, restore, enhance, relate, evaluate, and manipulate information contained in a digital image for many different purposes by means of computer technology; image samples are quantized to a fixed but sufficient number of information carrying units. Processing, storage, and transmission of digital representations of images offer many advantages over these operations performed on analogue representations: processing flexibility, easy or random access in storage, higher signal-to-noise ratio (SNR), possibility of error-free transmission, readiness for encryption and coding, and compatibility with other types of information as well as digital networks and computers, to name but a few. Image storage applications include medical imaging, image-based document management, and multimedia applications. Image transmission applications include broadcast television, remote sensing via satellites, aircraft, radar, sonar, teleconferencing, computer communications, and facsimile transmissions (A. K. Jain 1981).

2.2.2 Representation of Digital Images

An image is a 2-D model representing a special and limited aspect of an observed scene. It contains only a very small part of the original information extracted from the electromagnetic energy spectrum; for example x-ray, ultraviolet, visible, and infrared bands; mechanical forces; for example pressure and torsion; or other physical measures using an appropriate sensor that produces an electrical signal proportional to the input signal.

Since the information is processed in digital computers, this signal must be digitized in location, i.e. image sampling, and amplitude, i.e. level quantization. Thus the continuous image is digitized on a grid of square or hexagonal sampling points by mapping the amplitudes to a linear or non-linear quantization function (M. Sonka et al. 1993, p. 27). The result is a raw image.

For common systems, spatial resolutions include 256×256, 512×512, 1024×1024, 360×576, and 720×576 picture elements (pixels); and 256-level quantization generates 8-bit integers ranging from 0, i.e. black, to 255, i.e. white.

Since data processing uses algorithms, and their implementations depend on the data representation, the data structure holding the digitized image data must be adequate. There is a variety of traditional and hierarchical image-data structures that can be categorized into different levels of abstraction.

A matrix $A(L,M)$ of L rows by M columns of integer elements, each representing the brightness or another property of the corresponding pixel, holds the grid of pixels; and is the most common data structure for the direct representation of images. It can be defined as follows:

$$A(L,M) = \begin{bmatrix} a(1,1) & a(1,2) & . & a(1,M) \\ a(2,1) & a(2,2) & . & a(2,M) \\ . & . & a(l,m) & . \\ a(L,1) & a(L,2) & . & a(L,M) \end{bmatrix} \tag{2.1}$$

The matrix representation refers to the spatial domain; image data is accessible through the row and column indices of the associated pixels. Scanning or processing the matrix in left-to-right top-to-bottom order is purely a historical convention (R. J. Clarke 1995, p. 22); scanning in zigzag order, often employed in the frequency domain, is one

alternative. Many processing techniques benefit from this natural type of image-data structure; for example digital image processing frequently uses arithmetic and logical operations, filter operations often process overlapping sub-images, and compression techniques often work on non-overlapping sub-images. Transformation of the image into a different domain, for example using the fast Fourier transform (FFT) or the discrete cosine transform (DCT) (N. Ahmed et al. 1974), and subsequent manipulation in the transform domain is also used for processing and compression. Note that intermediate representation with more quantization levels can minimize the propagation of quantization errors (J. J. Rodríguez and C. C. Yang 1994).

While a single matrix can be interpreted as a grey-scale image, a matrix in a set of matrices can contain information about one spectral band of a multispectral or colour image. Alternatively, it can represent one instant in a time sequence of images. Since most programming languages support matrices, i.e. 2-D arrays, the implementation of this type of image-data structure is straightforward.

Other traditional image-data structures are chains, graphs, lists of object properties, and relational databases. Hierarchical data structures comprising of pyramids and quadtrees are means for more complex methods of image representation in computer vision (M. Sonka et al. 1993, pp. 42-55).

2.2.3 Digital-image-processing System

A block diagram of a typical generic image-processing system is shown in figure 2.1. Sensor and digitizer, i.e. analogue-to-digital converter, accomplish image acquisition. Some sensors, for example charged-coupled device (CCD) cameras and scanners, combine sensor and digitizer. Image data is manipulated by the processor; and stored

temporarily in internal memory, i.e. RAM, and permanently in mass storage, for example hard disk or tape. A keyboard accepts user input. A visual display unit (VDU), i.e. cathode-ray-tube (CRT) monitor, and other output devices, i.e. printer, are used to visualize the processed image data. The interface provides a link to other computers.

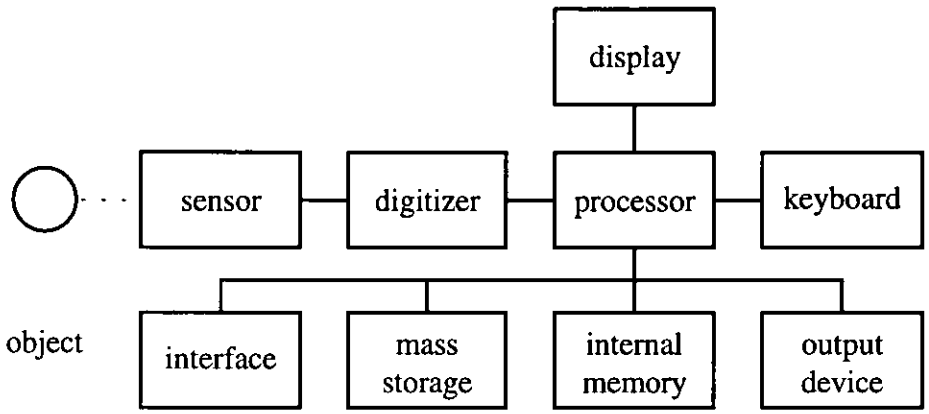


Figure 2.1 Generic Image-processing System

The display transforms the image data representing grey-level or colour values into luminance. Figure 2.2 depicts a typical transfer function (after S. A. Karunasekera and N. Kingsbury 1995).

However, as the function varies from display to display a faithful representation across computers is not achieved. The same problem applies to other input and output devices, and is addressed by device-independent colour management; see for example (Apple Computer 1995 and 1996).

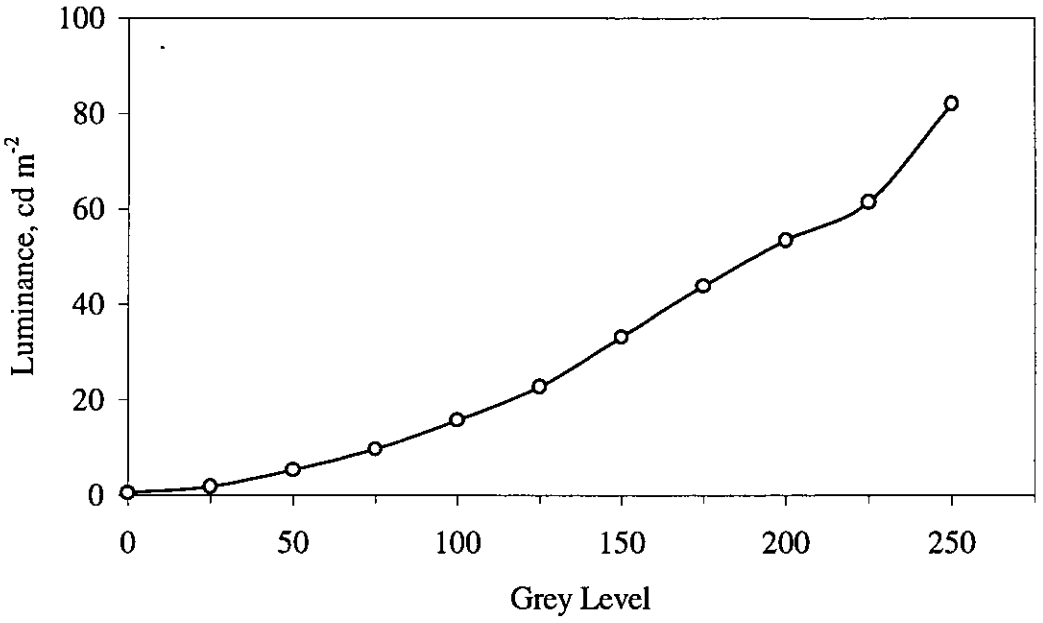


Figure 2.2 Typical Grey-level-to-luminance Transformation

2.3 Introduction to Digital Image Compression

2.3.1 Motivation for Digital Image Compression

Digital representations of images usually require enormous amounts of data; for example one image taken by Landsat's multispectral scanner (MSS) consists of about 31 MB, and one image taken by Landsat's thematic mapper (TM) consists of about 263 MB; see appendix A for details. In addition the amount of image data being collected, processed, stored, and transmitted increases rapidly because of higher utilization, new applications, and higher standards. A recent survey (B. Foster 1996) indicates for video microscopy a move toward higher spatial resolution, colour imaging, and sending images across networks. For these reasons storing and transmitting data is, and will remain, costly.

Processing of compressed images using efficient algorithms can also reduce the number of operations required to implement an algorithm (A. K. Jain 1981); R. S. Ledley

(1993) proposed that the processing of medical images be carried out in the compressed format.

A large variety of compression techniques has evolved over the years. Implementations exist in software, hardware, and as mixed solutions. In general, if the digital image reconstructed from the compressed representation is numerically identical to the original digital image, the employed compression technique is lossless. Lossless compression techniques relate to machine vision, and to applications where gathered information is too valuable or legal reasons prohibit any loss of information (R. C. Gonzalez and R. E. Woods 1992, p. 343). If the reconstructed image only approximates the original image, the employed compression technique is lossy. While data compression must generally be fully reversible or lossless, lossy image-compression techniques sacrifice some information in order to achieve higher compression. Lossy techniques relate to applications for human perception, and should, therefore, be designed to minimize a perceptually meaningful measure of distortion, rather than more traditional and more tractable criteria such as the mean square difference between original and reconstructed image (N. Jayant et al. 1993).

2.3.2 Objectives of Digital Image Compression

The main objective of digital image compression is to develop efficient digital representations of images that minimize the number of information carrying units, the bit rate, in order to reduce storage and transmission requirements, and ultimately to reduce costs. The bit rate can be measured in bits element^{-1}, bits pixel^{-1}, or bits s^{-1}.

Secondary objectives include:

- To minimize communication delay. The delay for encoding and decoding must match the requirements of an application. While, for example, real-time transmission demands short and same delay for encoding and decoding, the encoding delay for distribution via a storage medium is less important.
- To minimize complexity. The complexity is typically measured in terms of arithmetic capability, memory requirements, cost, and power consumption.
- To minimize the impact of errors on the reconstructed image.
- To support the exchange of compressed data among applications and across different computer systems as communication across networks grows in importance. This is addressed through standardization.

For lossy compression techniques an additional objective is to achieve the best image quality - however that might be defined - possible under given constraints.

The 'perfect' digital image-compression technique does not exist; the aim is, therefore, to minimize the bit rate in the digital representation of the image while maintaining required levels of image quality, complexity of implementation, and communication delay (N. Jayant et al. 1993). While, for example, a fixed bit rate in transmission results in varying quality, a fixed quality in storage causes a varying bit rate.

2.3.3 Data Redundancy

Three basic forms of data redundancy can be identified and exploited: coding redundancy, interpixel redundancy, and psychovisual redundancy. Digital image compression aims to remove redundancy and to reduce irrelevancy by exploiting one or more types of data redundancy.

Coding redundancy is due to the fact that integer pixel values are usually represented through natural binary codes: every codeword consists of the same number of bits regardless of its statistical probability of occurring. Coding redundancy can be exploited by assigning shorter codewords to more probable pixel values and longer codewords to less probable ones.

Interpixel redundancy arises due to the fact that shapes and objects in an image extend usually over a region of pixels; pixel values are therefore fairly similar to their neighbours. Interpixel redundancy can be exploited by relating pixels to the adjacent pixels; for example the difference between adjacent pixels can be calculated in various ways and used to represent an image.

Psychovisual redundancy is due to the fact the human visual system does not respond with equal sensitivity to all visual information. Certain information has less relative importance than other information and can, therefore, be eliminated without significantly impairing the perceived image quality.

As the limits of compression exploiting coding and interpixel redundancies have been reached (M. Kunt et al. 1985), the move towards perceptual coding is natural.

2.3.4 Digital-image-compression Model

An image-compression system, depicted in figure 2.3, consists of encoder, channel representing a transmission path or a storage medium, and decoder; the human eye is generally the ultimate receiver at the end of the system. On a high functional level the encoder block processes the original representation and feeds the encoded data into the channel. After transmission over the channel, the encoded representation is fed to the decoder block that generates the reconstructed representation.

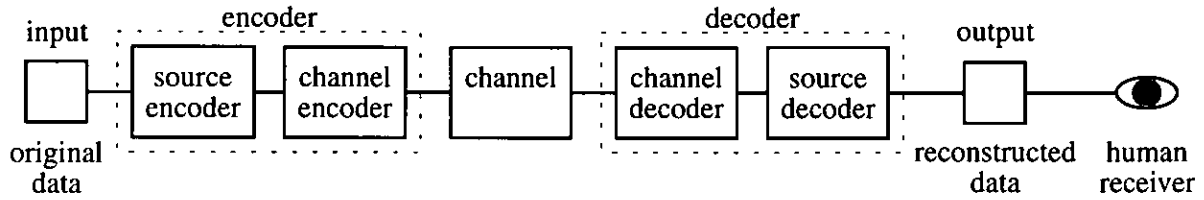


Figure 2.3 General Model of Image-compression System

Both the encoder and decoder consist of two sub-blocks. While, in an attempt to minimize the necessary bit rate for faithfully representing the input image, the source encoder removes data redundancies; the source decoder reverses the compression process. If an error-free system is required, it is the responsibility of the channel encoder-decoder pair to add redundancy to the encoded representation in order to recognize and correct any errors due to noise, distortion etc. introduced in the channel. However, the processes of source and channel coding can sometimes be integrated to increase efficiency of digital communication (N. Jayant et al. 1993). If the channel between encoder and decoder is noise free, the channel encoder and decoder can be omitted.

2.3.5 Entropy

The notion of coding is to find a new representation of an image that is smaller than the original representation of that image. Clearly, there is a lower bound that must depend on the image itself.

The histogram of an image represents the pixel distribution as a function of pixel value providing information on illumination conditions; contrast; range of values; and, maybe most importantly, probability distribution. If n_k pixels have the kth of L possible pixel values v_k in an image consisting of n pixels, then the probability of occurrence of value v_k can be defined as

$$P(v_k) = \frac{n_k}{n} \quad k = [0, 1, 2, \ldots, (L-1)] \tag{2.2}$$

The discrete function relates the count of a pixel value n_k to the total number of pixels n; probabilities range from zero, i.e. no occurrence, to one, i.e. exclusive occurrence. The sum of the probabilities is, of course, one:

$$\sum_{k=0}^{L-1} P(v_k) = 1 \tag{2.3}$$

Information theory models the generation of information as a probabilistic process; information content depends upon the probability of an event or symbol, i.e. pixel value in terms of image compression, occurring at each instance, i.e. pixel. Unlikely events, having low probability, carry more information than likely events, and vice versa. Ultimately, a certain event does not carry any information.

If the event E occurs with probability $P(E)$, then the self-information of that event is defined as

$$I(E) = \log_r \frac{1}{P(E)} = -\log_r P(E) \tag{2.4}$$

The amount of self-information $I(E)$ attributed to event E is inversely related to its probability $P(E)$; as $P(E)$ approaches one, $I(E)$ converges towards zero. The base r of the logarithm in the above equation specifies r-ary units of information. However, the base 2 conveniently generating binary units, i.e. bits, can be defined as

$$I(E) = \log_2 \frac{1}{P(E)} \text{ bits} = -\log_2 P(E) \text{ bits} \tag{2.5}$$

The entropy H, postulated by C. E. Shannon (1948a and b) as a measure of the complexity of an information source, defines the average amount of information conveyed per instance and can be defined as

$$H = -\sum_{j=1}^{J} P(E_j) \log_r P(E_j) = \sum_{j=1}^{J} P(E_j) I(E_j) \tag{2.6}$$

where J denotes the total number of events.

As less certainty, and thus more information, is conveyed; the entropy H increases. If all events are equally probable, the entropy is at a maximum. The base r of the logarithm in the above equation specifies r-ary units of information. Again, the base 2 conveniently generating binary units, i.e. bits, can be defined as

$$H = -\sum_{j=1}^{J} P(E_j) \log_2 P(E_j) \text{ bits} \tag{2.7}$$

Using the notation introduced in equation 2.2, the entropy of a digital image can be defined as

$$H = -\sum_{k=0}^{L-1} P(v_k) \log_2 P(v_k) \text{ bits} \tag{2.8}$$

Under the simplistic assumption that values of successive elements are statistically independent, i.e. no inter-element redundancy, the zero-order entropy H represents the lower bound: according to the noiseless coding theorem (C. E. Shannon 1948a and b), it is possible to encode information with entropy H bits element^{-1} using $H + \varepsilon$ bits element^{-1}, where ε is an arbitrarily small positive quantity.

Entropy coding is a well-established lossless method for reducing the bit rate of digital images by exploiting the statistical redundancy in those images. It exploits the nonuniform probability distribution of pixel values, generally exhibited by images, by encoding the pixel values using variable-length codewords rather than equal-length codewords.

2.4 Human Visual System

2.4.1 Function of Human Visual System

It is generally the human visual system that perceives and judges images after processing or coding, therefore attempts should be made to incorporate knowledge about the properties of the human visual system to digital image compression and quality assessment. This section summarizes some important properties of the human visual system. Further reading includes a description of the eye (R. C. Gonzalez and R. E. Woods 1992, pp. 22-28) and the human visual system (M. Kunt et al. 1985), a

brief functional description (D. J. Sakrison 1977), and a description of interactions among nerve cells in the retina (F. S. Werblin 1973).

The human visual system is a complex system in which the complexity of visual perception increases as the image information propagates through the system. Image information in the form of light intensity or luminance; that is a function of position, time, and wavelength or frequency; enters the human visual system. Refraction by the cornea, intraocular fluids, and lens focuses some of this information on the retina forming a retinal intensity image as a function of retinal position, time, and wavelength. Receptor cells at the back of the retina sense the intensities and, through a complex network of interconnecting cells, encode the image into neural signals to be carried by the optic nerve to the brain (D. J. Granrath 1981). Since optic-nerve fibres can only accurately transport signals over a range much smaller than the range of image information, the retina must compress the very large range of intensities presented by the outside world into a narrower range that can be handled by the optic-nerve fibres.

The human visual system is an anisotropic system: from a given sensitivity at 0°, i.e. horizontality, its sensitivity decreases to a minimum at 45° and then increases again reaching approximately the original level at 90° rotation. In addition, its sensitivity is frequency dependent. Compared to the sensitivity at 0°, the sensitivity at 45° to frequencies of 10 and 30 cycles deg^{-1} is reduced by 15 % and 30 % respectively (C. F. Hall and E. L. Hall 1977). Spatial frequencies within a range of about one octave, over a range of orientations of about ±20°, are indistinguishable from each other (W. E. Glenn 1993).

A common, but incomplete model of human vision incorporates a lowpass filter, a logarithmic nonlinearity, and a multichannel highpass filter; see (M. B. Sachs et al. 1971; C. F. Hall and E. L. Hall 1977; D. J. Sakrison 1977; and N. Jayant et al. 1993).

2.4.2 Relevant Properties of Human Visual System

It is the human eye that is generally the ultimate receiver of processed image data; see figure 2.3; therefore the properties of the human visual system should be considered, and suitable properties could be transferred to digital image compression. D. R. Fuhrmann et al. (1995) identified the following four properties as potentially being useful for digital-image-processing applications.

The human visual system responds to light in a nonlinear way. The smallest luminance difference that a human observer can detect when an object of a certain size is displayed at a certain background luminance level is defined as just-noticeable difference JND. For a wide range of light intensities L the just-noticeable difference JND, or ΔL, satisfies:

$$\frac{JND}{L} = \frac{\Delta L}{L} = constant \tag{2.9}$$

This is known as Weber's law, and suggests a logarithmic relationship between the physical and 'perceived' intensity of light, where the just-noticeable difference increases with increasing intensity. T. G. Stockham (1972), for example, proposed a visual model containing a logarithmic function and described its application to image enhancement. However, R. J. Clarke (1995, p. 8) reported that results of coding operations within a logarithmic/exponential domain had been inconclusive and argued that the conventional

display introduces a major nonlinearity in the processing chain that overrides the effects of the coding operations.

The human visual system performs spatial filtering. The optics of the eyeball have a lowpass characteristic. The lateral inhibition in the retina results in a highpass characteristic. The overall characteristic, that might be approximated by a bandpass characteristic, is centred somewhere between 4 and 8 cycles deg^{-1}; see (J. L. Mannos and D. J. Sakrison 1974; and R. J. Clarke 1985, p. 271, and 1995, pp. 7 and 75). Transform-based image-compression schemes offer a framework where the bit allocation of transform coefficients can be related to the spatial-frequency response, i.e. sensitivity, of the human visual system. Since only coefficients of the Fourier transform correspond directly to spatial frequency, the bit allocation must be modified for other transforms; H. Lohscheller (1984); N. B. Nill (1985); K. N. Ngan et al. (1989); and D. L. McLaren and D. T. Nguyen (1991) investigated the cosine transform. As the spatial frequency perceived by the eye depends on spatial resolution and viewing distance, the viewing conditions must be constrained. While a constant viewing distance of, for example, five times the image height (S. A. Karunasekera and N. Kingsbury 1995), and a fixed viewing position (D. R. Fuhrmann et al. 1995) can be obeyed for research purposes; these conditions cannot be assumed for practical applications in digital image compression. A. M. Lund (1993), for example, investigated viewing preferences, and found that the ratio of viewing distance to image height decreases as image size increases.

The human visual system performs spatial masking that is highly adaptive. This refers to the perceptibility of one signal in the presence of another in its time and frequency vicinity, and relates to the suppression of errors or distortion as a result of high image

activity or contrast. The aim of perceptual coding is to shape the error caused by lossy compression in a way so that the distortion is partially or fully masked by the signal, and therefore invisible to the human eye. In this context, it should be noted that high-frequency signals in visual information tend to have a short time or space support, while low-frequency signals tend to last longer (N. Jayant et al. 1993). Distortion masking, i.e. noise masking, has been incorporated in predictive and transform-coding techniques.

The human visual system has a small visual angle of 1 to 3°. Complex images are viewed with a series of brief fixations and rapid eye movements (D. R. Fuhrmann et al. 1995). This leads to local rather than global processing characteristics: the human observer tends to concentrate on those areas in which degradation is most visible and to assess the overall quality accordingly; see for example (J. O. Limb 1979; G. E. Legge and J. M. Foley 1980; and F. X. J. Lukas and Z. L. Budrikis 1982).

2.4.3 Significance of Human Visual System

As the properties of the human visual system govern the perception of visual information, digital image compression must take advantage of these properties in order to achieve lower bit rates by minimizing perceptually meaningful measures of distortion rather than more traditional criteria, such as the mean squared difference between the original and reconstructed image (N. Jayant et al. 1993). In digital image compression, coding bits can be allocated according to the importance of the information, in terms of the human visual system's sensitivity, that they convey. In quality assessment reliable numerical measures would allow efficient comparison of compression schemes, avoiding time consuming and expensive subjective tests under controlled conditions.

However, the human visual system and current digital-image-processing systems employ very different mechanisms.

While the human visual system responds to luminance, digital-image-processing systems manipulate grey-level or colour values that are transformed into luminance by the display. Since every display exhibits its own nonlinear transfer function, the perceived results vary from one digital-image-processing system to another.

While the human visual system responds to spatial frequency, digital-image-processing systems assume pixels of a certain size. The actual size of a pixel depends on the display, and the perceived spatial frequency is also a function of the viewing distance.

For practical applications spatial masking and local processing characteristic are currently the most significant properties.

2.5 Digital-image-compression Techniques

2.5.1 Properties of Digital-image-compression Techniques

Techniques for digital image compression can be classified in various ways. The criteria of accuracy distinguishes between information-lossless and information-lossy techniques, as described in subsection 2.3.1. Compression can be carried out in spatial, frequency, transform, 'visual', or other domains. It can exploit coding, inter-element, and psychovisual redundancies individually or in combination. Algorithms can be designed to adapt their parameters affecting, for example, bit allocation or quantization levels to changes in image statistics.

Algorithms process elements, i.e. pixels for approaches in the spatial domain, individually; in rectangular or square blocks; or segments of elements having similar properties, i.e. shapes. Encoding of blocks offers potential for significantly better performance than encoding of each element individually, since the requirement to transmit at least some information for every element is relaxed. The disadvantage of arbitrarily dividing an image into rectangular or square blocks is that, as the bit rate is decreased, the block structure, that is easily perceived and irritating to the observer, appears in the reconstructed image (R. J. Clarke 1995, p. 76). Encoding based on shapes derived from actual image content rather than on blocks circumvents the disadvantage and may supersede block-based encoding.

Research work has produced a large variety of compression techniques. The following subsections describe those techniques, that are relevant to this thesis.

2.5.2 Huffman Coding

Huffman coding, a well-known entropy-coding technique, reduces coding redundancy by constructing a variable-length code that assigns the shortest possible codewords to the most probable events, or symbols, using integer numbers of code symbols, for example bits for binary codes. Huffman coding is lossless and codes elements individually, i.e. one at a time. Huffman coding is optimal: it uses the variable-length code that achieves the minimum amount of redundancy possible when coding individual elements, i.e. for a particular set of symbols and their probabilities, no other integer code can be found that will give better coding performance than Huffman coding. It is a very popular technique used in many different schemes.

There are two basic restrictions imposed on the codewords:

- No two codewords consist of identical arrangements of code symbols.
- The code symbols are constructed in such a way that no additional indication is necessary to specify where a codeword begins and ends once the starting sequence of codewords is known.

For producing the minimum-redundancy variable-length code D. A. Huffman (1952) devised a method that builds up a tree by repetitively combining the least probable nodes, i.e. symbols and compound symbols, to a new node, i.e. compound symbol, with the summed probability until there is only one free node, i.e. the root node. Note that the probability of occurrence is proportional to the frequency of occurrence; see equation 2.2. Although r-ary trees can be built, binary trees are more popular. In non-adaptive schemes after the tree has been built and the code has been produced, encoding or decoding is simply accomplished by replacing original codewords with the Huffman codewords or vice versa. Storage and transmission of the code reduces efficiency.

A tree is a collection of nodes, that can contain information, and links, each connecting two nodes, that has certain properties. A path is a list of consecutive nodes that can be traversed via their links. The nodes directly succeeding a particular node are children of that node. In an ordered tree the order of the children is defined by some criteria. A node with at least one child is an internal node; a node without children is an external node. Internal nodes of a r-ary tree must have r children. The node directly preceding a particular node is the parent of that node, and the nodes also belonging to that parent are siblings. The one node without a parent is the root node. In a tree there exist exactly one path between the root node and every node, and exactly one path between any two nodes. The number of links from a node to the root node is the level, that can be used to

group nodes with the same distance from the root node. The internal path length is the sum of levels of all internal nodes. The external path length is the sum of levels of all external nodes. The path length is the sum of internal and external path length. In a binary tree every internal node has a left child and a right child, each of which can either be an internal or external node. Conventionally but arbitrarily, left children are identified by 0, and right children are identified by 1. Tracing the path from the root node to a particular external node generates a unique string of 0s and 1s; see (R. Sedgewick 1992, chapters 4 and 22).

After generating the symbol distribution, the tree for a binary Huffman code can be built with the following steps (M. Nelson 1992, pp. 34-35); note that probability or frequency of occurrence is represented through a weight:

- Locate the two nodes with the lowest weights in the list of free nodes. Note that nodes with identical weights are equally suitable in term of coding gain, but may change the height, i.e. maximum level, of the tree if internal and external nodes have identical weights.
- Create a parent node for these two nodes, and assign a weight equal to the summed weights of the two child nodes to it. To generate an ordered tree, that is necessary for adaptive Huffman coding, ensure that the weight of the left child is less than or equal to the weight of the right child.
- Add the parent node to and remove the two child nodes from the list of free nodes.
- Associate the left child node with 0, and the right child node with 1.
- Repeat above steps until only one free node is left. The free node is the root node of the tree.

Appendix B contains a worked example in which a Huffman tree is designed for an 8-level image of size 8×8.

The generation of the Huffman code can be equally described as a series of source reductions where the least probable source symbols are combined to form a new compound symbol with the summed probability that replaces the symbols from which it has been derived; see (R. C. Gonzalez and R. E. Woods 1992, pp. 343-345).

Huffman codes are instantaneous uniquely decodable block codes. They are called block codes, because each event is mapped to a codeword with a fixed sequence of code symbols, for example bits. They are instantaneous, because each codeword in a string of code symbols can be decoded without referencing succeeding events. They are uniquely decodable, because any string of code symbols can decoded in only one way without need for separation of the codewords (R. C. Gonzalez and R. E. Woods 1992, p. 345); see also (M. Nelson 1992, chapter 3; and R. J. Clarke 1995, appendix 1).

Non-adaptive Huffman schemes require two passes over the source symbols causing a delay: during the first pass the frequencies of occurrence of the events are collected, then the Huffman tree is constructed and stored or transmitted, and during the second pass the data is encoded. In adaptive Huffman schemes, the encoder and decoder start with identical initial trees, use the same algorithm to modify their trees and, therefore, stay synchronized. They require one pass, and are often more efficient than non-adaptive schemes; see (J. S. Vitter 1987).

Since codewords have to be an integer number of code symbols long, Huffman coding may have to assign either more or less code symbols to an event than theoretically necessary resulting in reduced efficiency; see equation 2.5. In general, Huffman coding

cannot reduce coding redundancy of data representing only two events, regardless of the probability distribution, since codewords require at least one code symbol.

2.5.3 Run-length Coding

Run-length coding exploits inter-element redundancy by representing a string of consecutive identical elements using a coding pair consisting of run length, that specifies the number of consecutive identical elements, and symbol, that specifies the value of the elements. Run-length coding is lossless. Although spatial-domain image data exhibits interpixel redundancy, strings of identical elements are rather short, especially in detailed natural images; however run-length coding can be utilized for 1-D and 2-D schemes in various ways. 2-D run-length coding processes a scan line in context with transitions in the previous scan line.

Run-length coding of binary images, that have only black and white pixels, is employed in facsimile (fax) coding. Strings of 0s and 1s in each scan line, i.e. row, are coded from left to right. The value, 0 or 1, of the first string of each row is either specified, or the value of the first string is conventionally assumed to be 0. As the string values alternate between 0 and 1, an initial run length of zero indicates in the latter scheme that the row actually starts with a black string. Additional entropy coding, for example Huffman coding, can be used to reduce the coding redundancy of the run lengths. The run lengths of black and white can be coded separately using two entropy coders that are specifically tailored to the individual statistics; see (R. C. Gonzalez and R. E. Woods 1992, p. 354).

Naturally, m-bit images can be decomposed into m 1-bit bit planes that can be coded using run-length coding for binary images. In order to reduce the effect of small grey-level variations, that can result in a very different bit pattern, an intermediate

representation of the image by an m-bit Gray code ensures that adjacent grey levels vary in only one bit plane; see (R. C. Gonzalez and R. E. Woods 1992, p. 350).

Assuming that in 8-bit images run lengths greater than 32, and pixel values greater than or equal to 224 would normally be rare; M. A. Sid-Ahmed (1995, p. 400) described an algorithm that uses, dependent on the context, an 8-bit symbol with its three most significant bits set to 1 not as pixel value but as repeat count in the range [0,31] preceding the pixel value. Generally, run lengths greater than 1, and less than or equal to 32 are coded through pairs consisting of repeat count and pixel value. Single pixels with values greater than or equal to 224 are coded through pairs consisting of a repeat count that is equal to zero, i.e. 11100000_2, and the pixel value. However, single pixels with values less than 224 can simply be coded through the abbreviated 'pair' consisting only of the pixel value. Run lengths greater than 32 are coded by generating more than one coding pair.

The concept of run-length coding can also be applied to sparse matrices, that are usually represented through a list of nonzero elements and their indices. For example, the nonzero elements in each row or column are coded from left to right, or from top to bottom respectively. The distance between the preceding and current nonzero element, i.e. the number of zero elements in between, and the value of current non-zero element are combined to form a pair. The value of the first nonzero element is coded with reference to the beginning of the scan line. While each index can only appear once in every scan line, the distances can generally produce a distribution that has a lower entropy.

2.5.4 Quantization

Quantization exploits psychovisual redundancy by mapping a range of input values, for example pixel values or coefficients, to a limited number of output values, i.e. symbols. The range of input values, that can be continuous or discrete, is divided into a number of regions, each of which is represented by one output value. A set of output values is also referred to as a pulse-code-modulated (PCM) signal. As information is being lost during the many-to-one mapping, quantization is lossy and not fully reversible. However, during the inverse process, dequantization, each symbol is replaced with a value that represents the associated range of input values. The range of input values can be divided into regions in various ways.

Uniform quantization simply divides the range of input values into N equally sized regions separated by equally spaced decision levels d_0 to d_N, neither taking the probability distribution of the values into account nor trying to minimize the introduced distortion. The quantizer represents an input value greater than d_i and less than or equal to d_{i+1}, $(d_i, d_{i+1}]$, by an output symbol of value i. The dequantizer generates a reconstructed value r_i from a symbol i using:

$$r_i = \frac{d_i + d_{i+1}}{2} \tag{2.10}$$

Nonuniform quantization refers to a range division using unequally spaced decision levels. It is also known as optimal quantization, since this approach usually involves optimization of a statistical measure or psychovisual measure; see (A. N. Netravali 1977). The Lloyd-Max quantizer, independently developed by S. P. Lloyd (1982) and J. Max (1960), minimizes the mean-square quantization error by determining the best decision and reconstruction levels taking the overall probability distribution of the input

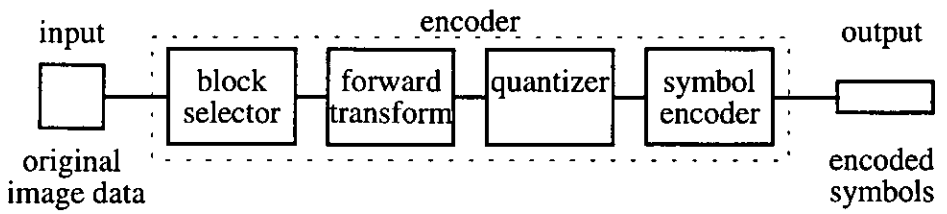
values into account; see also (R. C. Gonzalez and R. E. Woods 1992, pp. 370-371; and M. A. Sid-Ahmed 1995, pp. 433-450).

Adaptive quantization adjusts the quantization levels based on the local probability distribution; see (A. N. Netravali and B. Prasada 1977). In a block-based spatial-domain scheme each block of image data is quantized using the quantizer, from a number of available quantizers, that introduces least distortion. The quantizers may be scaled versions of a Lloyd-Max quantizer for unit-variance Laplacian probability distribution, and the overhead associated with the quantizer selection is appended to each block; see (R. C. Gonzalez and R. E. Woods 1992, pp. 371-374).

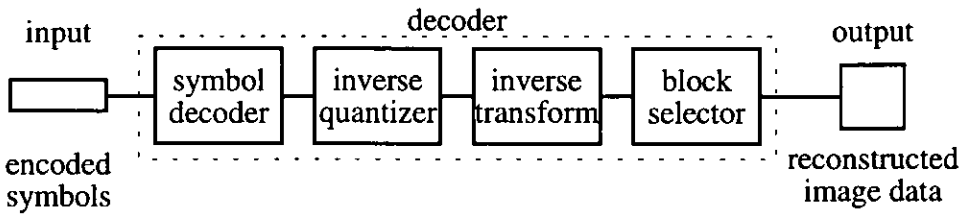
2.5.5 Transform Coding

Transform coding describes a concept of a group of lossy digital-image-compression techniques, rather than one particular scheme, that has been incorporated into standards such as the JPEG still picture compression standard for lossy compression; see for example (G. K. Wallace 1992). The core of any transform-based coding system, that consists of a number of different coding stages, is a reversible, linear, 1-D or 2-D transform; that maps image data, i.e. a set of pixels, into a set of transform coefficients that has the same size. The purpose of this transform stage is to remove interpixel redundancy by converting statistically dependent pixel values into a set of 'less correlated' or 'more independent' coefficients. For most natural images a significant number of these coefficients have small magnitudes and can be coarsely quantized, or discarded entirely, with little image distortion (R. C. Gonzalez and R. E. Woods 1992, p. 374).

Figure 2.4 depicts a typical transform-coding system. During encoding the block selector splits the original image into blocks of pixels that are then processed by the forward transform to produce blocks of transform-domain coefficients. The quantizer, making transform-based coding lossy, maps each block into a set of symbols, i.e. quantized and scaled transform coefficients, which is then entropy-coded by the symbol encoder. The result is a continuous stream of encoded symbols. During decoding the decoder performs the inverse sequence of steps. The symbol decoder decodes the data stream and produces sets of symbols, each of which is mapped by the inverse quantizer into a block of quantized transform-domain coefficients, which is then processed by the inverse transform to produce a block of pixels. The block selector merges the blocks of pixels into the reconstructed image. While nonadaptive transform coding does not take local image content into account, adaptive transform coding enables one or more coding stages to respond to local image content; see for example (A. Habibi 1977).



a) Encoder



b) Decoder

Figure 2.4 Transform-coding System

For a transform, playing the key role in this group of image-compression techniques, an inverse transform, that restores the data to its original form, must exist:

$$f(x,y) \underset{\text{inverse transform}}{\overset{\text{forward transform}}{\rightleftarrows}} T(u,v) \tag{2.11}$$

The spatial-domain representation $f(x,y)$, i.e. a set of pixels, can be transformed into its transform-domain representation, i.e. a set of transform coefficients, and vice versa. The forward transform maps an $L \times M$ block of image data into an $L \times M$ block of transform coefficients. Although 1-D transforms can be defined, 2-D transforms are a natural choice for digital image processing, that is concerned with 2-D image data. However, separable 2-D transforms are often implemented as two sets of 1-D transforms. 'Fast' implementations reduce the number of arithmetic operations.

A variety of transforms is available; for example FFT, DCT, Discrete sine transform (DST), Haar transform, Hadamard transform, Karhunen-Loève transform (KLT), Slant transform, and Walsh transform; see (R. C. Gonzalez and R. E. Woods 1992, chapter 3; and R. J. Clarke 1985). Selecting a transform for use in an image-compression scheme requires a compromise between transform efficiency and computational complexity to be made. The transform efficiency describes the transform's ability to decorrelate inter-element redundancy and to pack the energy that is spread across the image into as few transform coefficients as possible.

The 2-D FFT, a fast implementation of the 2-D discrete Fourier transform (DFT), carries out a 2-D spectral analysis of the image data. Only Fourier transform coefficients correspond directly to measured spatial frequency; however, the transform efficiency is lower than that of other transforms.

The KLT transforms image data into a set of uncorrelated coefficients; and furthermore, for a given arbitrary number of retained transform coefficients, it minimizes the mean

square error between original and reconstructed image. The KLT is optimal in terms of decorrelation and energy compaction; however, the computational complexity and the lack of fast algorithms limit its use.

The DCT (N. Ahmed et al. 1974) performs almost as well as the KLT; R. J. Clarke (1995, p. 62) reported that extensive experiments had demonstrated conclusively that the DCT has the best still image coding performance of all those transforms having data-independent basis vectors, and which approaches that of the optimum, data-dependent (KLT) transform. Although the DCT is slightly suboptimal in terms of decorrelation and energy compaction, it can be computed efficiently using an approach similar to that used for the Fourier transform. H. Lohscheller (1984); N. B. Nill (1985); and D. L. McLaren and D. T. Nguyen (1991) related the cosine transform to the human visual system. The DCT is an efficient and effective image-compression technique (N. B. Nill 1985); many transform-based coding schemes and standards benefit from its transform efficiency and computational efficiency.

In image compression, although transforms are defined for blocks of general dimensions $L \times M$, they are not applied to whole images at once, but to blocks, i.e. sub-images, of smaller dimensions. The reasons are twofold; see (M. A. Sid-Ahmed 1995, p. 404):

- The transform of small blocks is computationally less complex than that for the whole image.
- The correlation between pixels is less between distant pixels than between neighbouring pixels.

However, dependent on the chosen transform, the level of compression increases as the block dimensions increase. The most popular block dimensions are 8×8 and 16×16 (R. C. Gonzalez and R. E. Woods 1992, pp. 379-380).

Compression results from the deletion of any sufficiently small transform coefficients and the variable bit-rate quantization of the remainder in the quantizer. Note that, usually, coefficients of large magnitude are clustered around zero frequency, that is situated in the top left-hand corner of the coefficient block; and coefficients of smaller magnitude are distributed towards the highest spatial frequency in both horizontal and vertical directions, that is situated in the bottom right-hand corner of the coefficient block (R. J. Clarke 1995, p. 64). There are two methods for selection of coefficients for further processing: while in zonal coding each coefficient, dependent on its location within the block, is associated with a certain number of bits; in threshold coding coefficients exceeding some threshold are retained. Entropy coding, for example Huffman coding (D. A. Huffman 1952) or arithmetic coding (I. H. Witten et al. 1987), can be used subsequently to convert the remaining quantized and scaled transform coefficients into a continuous data stream.

The main advantage of transform coding is that it processes images in a similar manner to the human visual system (W. E. Glenn 1993). Compared to other lossy image-compression techniques, transform coding preserves subjective image quality better, and is less sensitive to changes in image statistics. Transform coding is less sensitive to channel noise: if a transform coefficient is corrupted during transmission, the resulting image distortion is spread through the sub-image (M. Sonka et al. 1993, p. 468). However, the main disadvantage is that, as the bit rate is decreased, the block structure becomes visible in the reconstructed image. Removing too many high-frequency

coefficients causes blurring of object-edge detail (R. J. Clarke 1995, pp. 86 and 162). In addition, the transform stages present in encoder and decoder generate an increased complexity compared to other techniques.

2.5.6 Other Techniques

This subsection enumerates some more coding techniques and provides appropriate references.

Arithmetic coding exploits coding redundancy by encoding the entire information as a single floating-point number equal to or greater than 0 and less than 1, [0,1), by modifying the number with every element added according to the rescaled probability distribution of the elements. Arithmetic coding is lossless; it can encode elements using fractional numbers of bits; and is, therefore, more efficient than Huffman coding, that must assign an integer number of bits per element; see (I. H. Witten et al. 1987; M. Nelson 1992, chapter 5; P. G. Howard and J. S. Vitter 1994; and R. J. Clarke 1995, appendix 1).

Predictive coding exploits inter-element redundancy by predicting the value of the present element from the values of a selection of elements that have been processed previously. The difference between the value of the present element and the prediction is encoded. Lossy predictive coding results from a combination of quantization and lossless predictive coding; see (A. N. Netravali and J. O. Limb 1980; A. K. Jain 1981; R. C. Gonzalez and R. E. Woods 1992, chapter 6; and R. J. Clarke 1995, chapter 2). Predictive coding is less complex than transform coding, for example; and hardware implementations are available.

Dictionary-based coding substitutes a number of consecutive elements with an index to a matching entry in a dictionary, i.e. codebook. The smaller size of the index, compared to the size of the elements replaced, results in compression. The size of the index can be variable to reduce coding redundancy. Static schemes using a predefined dictionary that remains unchanged during coding can take advantage of variable-length indices, however the dictionary has to be made available for encoding and decoding. Adaptive schemes start coding with an empty or default dictionary and add new entries to the dictionary during coding. J. Ziv and A. Lempel (1977 and 1978) described two adaptive dictionary-based techniques: while LZ77 uses a window sliding over previously processed elements as dictionary with fixed-length entries; LZ78 builds new variable-length dictionary entries up one element at a time by adding a new element to an existing entry when a match occurs, thus generating a potentially unlimited number of dictionary entries; see (M. Nelson 1992, chapters 7-9). Dictionary-based coding is lossless and more suitable for text data than spatial-domain image data, since matching a dictionary entry requires an identical string of elements.

Vector quantization is a lossy block-based spatial-domain coding technique that processes vectors of reordered elements. Each block is represented by an index to a codebook entry having the best similarity. The smaller size of the index, compared to the size of the block replaced, results in compression. Generally, blocks of image data consist of uniform areas, or areas of similar general shape or intensity profile rather than areas of chaotic or random structure, hence the codebook requires only a fraction of the number of entries theoretically possible. D. L. Ruderman (1994) reviewed and investigated the statistics of natural images, and reported invariance to scale and hierarchical invariance in natural images. While designing the codebook, and searching

the codebook during encoding is computationally intensive, decoding comprises of a simple look-up of the codebook entry specified by the stored or transmitted index; see (B. Marangelli 1991; P. C. Cosman et al. 1993; C. Constantinescu and J. A. Storer 1994; and R. J. Clarke 1995, chapter 4). Vector quantization can also be applied to transform-domain coefficients; see (C. Labit and J. P. Marescq 1986).

The review papers of A. N. Netravali and J. O. Limb (1980); and A. K. Jain (1981), for example, summarize the image coding techniques available at the beginning of the 1980s, that have evolved to the current techniques. Descriptions of bit-plane coding and other techniques can be found in (R. C. Gonzalez and R. E. Woods 1992, chapter 6). In addition to the techniques mentioned above, R. J. Clarke (1995) also described sub-band and wavelet coding as well as segmented, block-truncation, and fractal coding; and other techniques.

2.6 Image Quality Assessment

2.6.1 Motivation for Image Quality Assessment

The assessment of lossy image-compression techniques in terms of image quality is the means of comparing their effectiveness. The objective is to assess a reconstructed image accurately, quickly, and inexpensively.

2.6.2 Subjective Image Quality

Subjective assessment by human observers incorporating the human visual system takes psychovisual effects into account. It is important to establish controlled viewing conditions, and to average the evaluations of the observers. However, subjective

assessment is time-consuming and expensive, tends to be biased by environmental influences, and results tend to be difficult to compare.

A variety of procedures for psychovisual experiments has been developed; for example D. J. Sakrison (1977) described self-setting methods, rating experiments, and forced-choice experiments; and S. A. Karunasekera and N. Kingsbury (1995) employed timing methods.

Perceived image quality is often measured on a five-point scale of quality known as mean opinion score (mos) or, alternatively, on a five-point scale of impairment; see table 2.1 (N. Jayant et al. 1993). Other scales are also in use; for example J. L. Mannos and D. J. Sakrison (1974) employed a seven-point scale to order groups of images.

Quality
excellent
good
fair
poor
bad

a) Quality Scale

Impairment
imperceptible
perceptible but not annoying
slightly annoying
annoying
very annoying

b) Impairment Scale

Table 2.1 Scales for Subjective Image Quality Assessment

2.6.3 Objective Image Quality

Objective assessment aims to calculate a numerical value that indicates the quality of a reconstructed image compared to the original image.

The mean-square-error function calculates the average squared error per pixel

$$MSE = \frac{1}{L\,M} \sum_{l=1}^{L} \sum_{m=1}^{M} \left[i(l,m) - \hat{i}\,(l,m) \right]^2 \qquad (2.12)$$

where L and M represent the dimensions of the image; $i(l,m)$ is the pixel value of the original image; and $\hat{i}(l,m)$ is the pixel value of the reconstructed image. The mean-square error avoids averaging effects of positive and negative errors and amplifies larger errors.

The signal-to-noise ratio can be defined as

$$SNR = 10 \log_{10} \frac{\sum_{l=1}^{L} \sum_{m=1}^{M} i^2(l,m)}{\sum_{l=1}^{L} \sum_{m=1}^{M} \left[i(l,m) - \hat{i}(l,m) \right]^2} \text{ dB} \tag{2.13}$$

where L and M represent the dimensions of the image; $i(l,m)$ is the pixel value of the original image; $\hat{i}(l,m)$ is the pixel value of the reconstructed image.

The peak-signal-to-noise ratio can be defined as

$$PSNR = 10 \log_{10} \frac{L\, M\, {i_{MAX}}^2}{\sum_{l=1}^{L} \sum_{m=1}^{M} \left[i(l,m) - \hat{i}(l,m) \right]^2} \text{ dB} \tag{2.14}$$

where L and M represent the dimensions of the image; $i(l,m)$ is the pixel value of the original image; $\hat{i}(l,m)$ is the pixel value of the reconstructed image; and i_{MAX} is the maximum grey-level value, for example $i_{MAX} = 2^8 - 1 = 255$ for 8-bit pixel values.

2.6.4 Human-visual-system-based Objective Image Quality

Incorporating properties of the human visual system into objective assessment leads to objective assessment that can model the perceived image quality more accurately.

J. O. Limb (1979) investigated the root-mean-square error

$$RMSE_p = \sqrt[p]{\frac{1}{L\,M}\sum_{l=1}^{L}\sum_{m=1}^{M}\left|i(l,m)-\hat{i}\,(l,m)\right|^p} \tag{2.15}$$

where L and M represent the dimensions of the image; $i(l,m)$ is the pixel value of the original image; $\hat{i}\,(l,m)$ is the pixel value of the reconstructed image; and $p=[1, 2, 3, 4, 6]$ refers to the absolute, squared, cubed, fourth, and sixth error respectively; $RMSE_1$ is the average absolute value error, and $RMSE_2$ is the root-mean-square error. The higher the value of p, the greater is the relative emphasis given to large errors in the image. He also used a weighting function that implements the masking effect, and recognized the importance of local rather than global quality assessment. F. X. J. Lukas and Z. L. Budrikis (1982) reported a similar approach for monochrome time-variant pictures using nonlinear filters each consisting of excitation and inhibition paths followed by different combinations of filter and mask stages. They investigated raw, filtered, filtered temporally masked, filtered spatially masked, and filtered spatially and temporally masked errors with $p=[2, 4]$ for global averaging and two maximum-error procedures; and their work confirmed that filtered and masked error measures work better for local assessments than filtered error measures for global assessment.

D. R. Fuhrmann et al. (1995) favoured simple pointwise distance measures, and discouraged the use of metrics based on the spatial-frequency response, since these measures require precise knowledge of the viewing conditions. They found the Michelson contrast, or distortion contrast, most useful:

$$DCON = \frac{1}{L\,M} \sum_{l=1}^{L} \sum_{m=1}^{M} \frac{\left| j(l,m) - \hat{j}(l,m) \right|}{j(l,m) + \hat{j}(l,m)} \tag{2.16}$$

where L and M represent the dimensions of the image, $j(l,m)$ is the pixel luminance of the original image, and $\hat{j}(l,m)$ is the pixel luminance of the distorted image.

Although it is well-known, that the mean-square error is not a reliable objective measure, see for example (J. L. Mannos and D. J. Sakrison 1974; A. Tremeau et al. 1994; and D. R. Fuhrmann et al. 1995); and despite all efforts to establish objective measures based on the human visual system, see for example (J. O. Limb 1979; F. X. J. Lukas and Z. L. Budrikis 1982; and D. R. Fuhrmann et al. 1995); the mean-square error remains very popular. This is due to the fact that the mean-square error is easy to understand, is simple to calculate, and appears to be more 'objective' than a formula or procedure that involves some kind of filtering and masking. However, S. A. Karunasekera and N. Kingsbury (1995) presented several reconstructions of an image that have an identical mean-square error and look very different, thus proving the inappropriateness of this measure once more.

2.7 Summary

The amount of image data being processed increases due to higher utilization, new applications, and higher standards. The notion of digital image compression is to reduce storage and transmission requirements. Although some types of application require lossless compression of digital images, it is mainly the human eye that is the ultimate receiver of image data. A variety of compression techniques; for example Huffman coding, run-length coding, and predictive coding; has evolved over the years. As the

limits of these more conventional techniques have been reached; the move towards perceptual coding, exploiting properties of the human visual system, is natural. Many compression schemes combine different data compression techniques with good effect; for example entropy coding of run lengths, or entropy coding of dictionary indices. Transform coding, that is for example utilized in the JPEG still picture compression standard for lossy compression, decorrelates image data and processes images in a similar manner to the human visual system. Encoding of blocks, as utilized in transform coding and vector quantization, offers potential for significantly better performance than encoding of individual elements.

Chapter 3

JPEG Still Picture Compression Standard

3.1 Introduction

This chapter discusses the Joint Photographic Experts Group (JPEG) still picture compression standard in some detail. However, as the chapter focuses on the concept of the standard, many interesting details for implementation are necessarily omitted.

Section 3.2 briefly narrates the history of JPEG, references the international standard generated, describes the aims and requirements of the JPEG standard, and summarizes the selection process conducted in order to identify the most suitable compression method.

Section 3.3 defines the JPEG-compatible image; describes interleaved and noninterleaved processing; and outlines sequential, progressive, lossless, and hierarchical modes of operation.

Section 3.4 outlines the DCT-based coding method, and describes the processing steps in more detail using the baseline sequential process as an example.

Section 3.5 relates the DCT-based coding method to transform coding, that is described in chapter 2; and identifies potential difficulties. Finally section 3.6 concludes the chapter with a brief summary.

3.2 Background

Recognizing the need for an international standard (IS) for digital compression of continuous-tone still images, both grey-scale and colour, in order to boost the utilization of digital images in general-purpose computer systems; the International Organization for Standardization (ISO) and the International Telegraph and Telephone Consultative

Committee (CCITT) established in 1986 the Joint Photographic Experts Group. In November 1987 the International Electrotechnical Commission (IEC) joined with ISO to create a new Joint Technical Committee 1 (JTC 1) in the field of information technology, under which the JPEG committee continued to operate. In 1994 and 1995 the work on 'Digital compression and coding of continuous-tone still images' resulted in ISO/IEC 10918-1:1994 (Part 1 requirements and guidelines) and ISO/IEC 10918-2:1995 (Part 2 compliance testing) respectively, and the identical CCITT Recommendation T.81. ISO/IEC Draft IS (DIS) 10918-3 (Part 3 extensions) and ISO/IEC DIS 10918-4 (Part 4 registration procedures for JPEG profile, APPn marker, and SPIFF profile ID marker) currently await promotion to ISs. A. Léger et al. (1991) and G. K. Wallace (1990, 1991, 1992) reported on JPEG's progress. W. B. Pennebaker and J. L. Mitchell (1992) produced a very detailed description of the JPEG still image data compression standard and included ISO/IEC DIS 10918-1 and ISO/IEC DIS 10918-2.

JPEG aimed to develop a standard for digital compression of continuous-tone images across different applications and computer systems that meets the following requirements (G. K. Wallace 1992):

- To be at or near the state of art with regard to compression rate and accompanying image fidelity, over a wide range of quality ratings. In addition, the encoder should be parametric, so that the application or user can set the desired quality/compression trade-off.
- To be applicable to practically any kind of continuous-tone digital source image; i.e. not to be restricted to images of certain dimensions, colour spaces, pixel aspect

ratios, etc.; and not to be limited to classes of imagery with restrictions on scene content; for example complexity, range of colours, or statistical properties.

- To have traceable computational complexity to allow feasible software and hardware implementations.
- To have the following modes of operation: sequential encoding, i.e. each image component is encoded in a single left-to-right, top-to-bottom scan; progressive encoding, i.e. the image is encoded in multiple scans; lossless encoding, i.e. the image is encoded to guarantee exact reconstruction; and hierarchical encoding, i.e. the image is encoded at multiple resolutions so that lower-resolution versions may be accessed without first having to decompress the image at its full resolution.

In order to identify the most suitable method, JPEG conducted a selection process based on blind assessment of subjective picture quality. During a first contest in June 1987, three of the initial 12 candidate methods were short-listed: adaptive DCT (ADCT), differential PCM (DPCM) using binary arithmetic coding, and progressive block-truncation coding. In January 1988 in a second contest, JPEG chose the ADCT, because of its superior image quality, and the demonstrated feasibility in both software and hardware implementations. The ADCT was based on 8×8 blocks for two reasons: computational complexity and the availability of hardware implementations. The block size of 16×16 was explored and found not to give enough improvement in compression to justify the extra image buffering, precision of internal calculations, and complexity.

JPEG discovered later that a DCT-based lossless mode was difficult to define as a practical standard without placing severe constraints on both encoder and decoder implementations. As a consequence, JPEG chose a simple predictive method that is

independent from the DCT-based method to meet its requirement for a lossless mode of operation. Hence the DCT-based method applies only to lossy modes of operation. However, both methods employ either Huffman or arithmetic coding for entropy coding. Since Huffman and arithmetic coders encode and decode the same set of symbols, a transcoding process can be used to convert Huffman-coded data into arithmetic-coded data and vice versa. Note that, for the DCT-based method, one set of Huffman tables, i.e. codes, consists of one direct-current (DC) table and one alternating-current (AC) table.

3.3 Outline of the JPEG Standard

3.3.1 Image Components

In the JPEG standard, compressed image data consists of only one image, that contains $1 \leq Nf \leq 255$ image components C_1 to C_{Nf}. Note that a grey-scale image consists of only one component, and that a colour image consists of multiple components. Although colour images can be represented in different colour spaces, the JPEG standard is 'colour-blind', i.e. the JPEG compression algorithm is indifferent to the kind of information that is contained in a particular component. Each component C_i consists of a matrix of y_i rows by x_i columns of samples, i.e. pixels; and represents one colour-space coordinate within a particular colour space. Components can have different dimension in order to accommodate formats in which some components are sampled at different rates than others. The image has overall dimensions $1 \leq Y \leq 65535$ rows by $1 \leq X \leq 65535$ columns, where Y is the maximum of the y_i values and X is the maximum of the x_i values for all components C_1 to C_{Nf}. The relative vertical and

horizontal sampling factors of each component, V_i and H_i, relate the dimensions of the component, y_i and x_i, to the overall dimensions, Y and X; and must be integer values in the range $[1,4]$. The encoded parameters are Y and X, and V_i and H_i values for each component C_i. The decoder reconstructs the dimensions y_i and x_i of each component C_i using:

$$y_i = \left\lceil Y \times \frac{V_i}{V_{\max}} \right\rceil \tag{3.1}$$

$$x_i = \left\lceil X \times \frac{H_i}{H_{\max}} \right\rceil \tag{3.2}$$

where $V_{\max}$ and $H_{\max}$ are the maximum relative vertical and horizontal sampling factors of all components; and $\lceil\ \rceil$ denotes the ceiling function, i.e. round up.

3.3.2 Interleaving Image Components

The JPEG standard allows manipulation of the order in which the components are coded. If an image component is not interleaved with other components, data units are ordered in a simple left-to-right, top-to-bottom sequence. Note that the JPEG standard defines a data unit as an 8×8 block of samples in the DCT-based method and as a sample in the predictive method. If two or more components are interleaved, each component C_i is partitioned into rectangular regions of $V_i \times H_i$ data units. Regions are ordered within a component from left-to-right and top-to-bottom, and data units are ordered within a region from left-to-right and top-to-bottom. The JPEG standard defines a minimum coded unit (MCU) as the smallest group of interleaved data units; the maximum number of components in an MCU is four, and the maximum number of

data units in an MCU is ten. Therefore not every combination of four components that can be represented in noninterleaved order is allowed to be interleaved. However, the JPEG standard allows some components to be interleaved and some to be noninterleaved within an image. Note that for a noninterleaved scan the MCU is defined to be one data unit.

3.3.3 An Example of Interleaved Image Components

In the example below, an image consisting of three components C_A, C_B, and C_C that are processed in one interleaved scan is assumed. The image is processed using the DCT-based method, that operates on 8×8 blocks of samples.

Each component C_i has the dimensions y_i rows by x_i columns, and the relative vertical and horizontal sampling factors V_i and H_i respectively; see table 3.1. The image has the overall dimensions $Y = 32$ rows by $X = 32$ columns. The maximum relative vertical and horizontal sampling factors are $V_{max} = 2$ and $H_{max} = 2$ respectively.

Component C_i	y_i	x_i	V_i	H_i
C_A	32	32	2	2
C_B	32	16	2	1
C_C	16	32	1	2

Table 3.1 Component Parameters for Example of Three-component Image

Figure 3.1 visualizes the three components C_A, C_B, and C_C with their data units $A_1,\ldots,A_{16}$, $B_1,\ldots,B_8$, and $C_1,\ldots,C_8$ respectively indicated through dotted lines. Note that each region, indicated through solid lines, contains V_i by H_i data units. The MCUs are coded in a sequential manner as outlined in table 3.2.

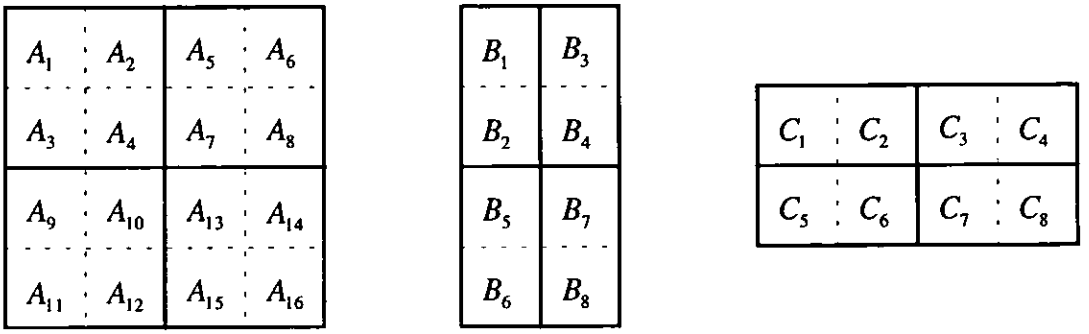


Figure 3.1 Data Units and Regions for Example of Three-component Image

MCU Number	Data Units in MCU							
1	A_1	A_2	A_3	A_4	B_1	B_2	C_1	C_2
2	A_5	A_6	A_7	A_8	B_3	B_4	C_3	C_4
3	A_9	A_{10}	A_{11}	A_{12}	B_5	B_6	C_5	C_6
4	A_{13}	A_{14}	A_{15}	A_{16}	B_7	B_8	C_7	C_8

Table 3.2 MCUs for Interleaved Scan of all Three Components

for Example of Three-component Image

3.3.4 Sample Precision

Each sample is an unsigned integer with precision P bits in the range $[0,2^P-1]$. All samples of each component within a frame have the same precision P. Note that a frame consists of one or more scans. P is 8 or 12 for the DCT-based method, dependent on the mode of operation; and is in the range [2,16] for the predictive method.

3.3.5 Modes of Operation

The JPEG standard defines four distinct modes of operation:

In the sequential DCT-based mode each group of one to four image components is completely coded in a single left-to-right, top-to-bottom scan. Although components are

interleaved for scans with two to four components, each component is coded separately. This mode minimizes coefficient storage requirements. A particular restricted form of this mode is known as the baseline sequential process. It represents a minimum capability that must be present in all DCT-based decoder systems. Sequential DCT-based processes that have capabilities beyond the baseline sequential requirements are known as extended sequential processes.

In the progressive DCT-based mode each scan, having one to four image components, is partially coded in multiple left-to-right, top-to-bottom sequences using spectral selection and successive approximation. In spectral selection quantized DCT coefficients are grouped into bands of related frequencies, usually lower frequency bands are coded first. In successive approximation quantized DCT coefficients are coded first with lower precision, they are refined in later scans. Either procedure is used separately, or they are mixed in flexible combinations. This mode has the highest coefficient storage requirements.

In the sequential lossless mode one to three neighbouring samples are used to predict the current sample. This prediction is then subtracted from the actual sample value, and the difference is losslessly entropy-coded. The prediction equation for each scan, having one to four components, is selected from a set of eight equations. Components are interleaved for scans with two to four components.

The hierarchical mode provides for progressive coding with increasing spatial resolution between progressive stages. It is similar to the progressive DCT-based mode, and useful in environments that have multiresolution requirements. The hierarchical mode also offers the capability of progressive transmission to a final lossless stage.

Table 3.3 summarizes the essential characteristics of the distinct coding processes.

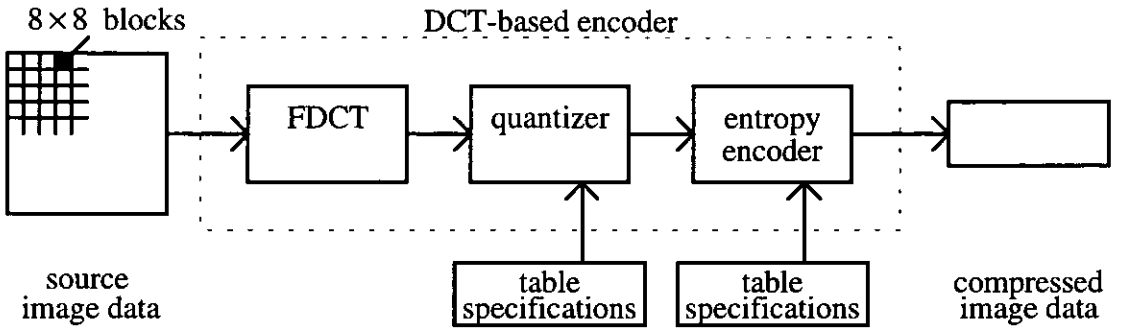
	Baseline Sequential Process	Extended DCT-based Processes	Lossless Processes	Hierarchical Processes
Method	DCT-based lossy process	DCT-based lossy process	predictive lossless process	extended DCT-based processes and lossless processes
Frame	single	single	single	multiple
Precision	8 bits per sample per component	8 or 12 bits per sample per component	$2 \leq N \leq 16$ bits per sample per component	(dependent on Method)
Mode	sequential	sequential or progressive	sequential	(dependent on Method)
Entropy Coding	Huffman coding with 2 sets of tables per scan	Huffman or arithmetic coding with 4 sets of tables per scan	Huffman or arithmetic coding with 4 DC tables per scan	(dependent on Method)
Coding	scans with 1, 2, 3, and 4 components			
Interleaving	interleaved and noninterleaved scans			

Table 3.3 Essential Characteristics of the Distinct Coding Processes

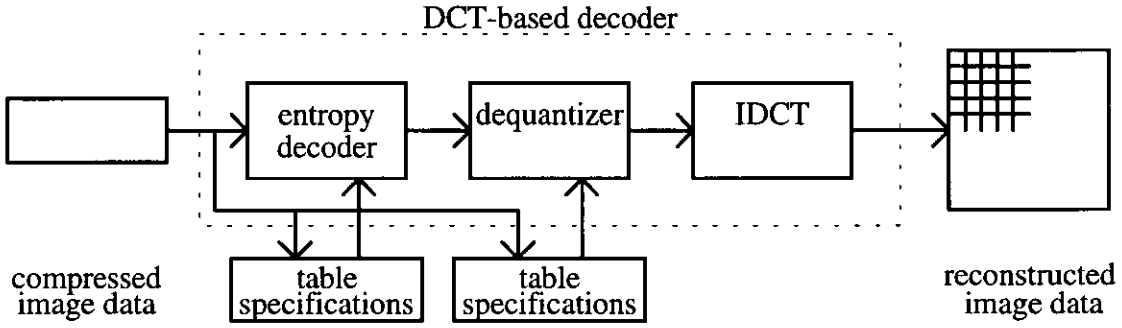
3.4 Baseline Sequential Process

3.4.1 DCT-based Coding

Figure 3.2 depicts the DCT-based encoder and decoder identifying the key processing steps. The compression of a single-component, i.e. grey-scale, image is assumed. Compression of a multicomponent, i.e. colour, image can be approximately regarded as the compression of multiple single-component images utilizing noninterleaved and interleaved processes, since all processes operate on each component independently.



a) Simplified DCT-based Encoder



b) Simplified DCT-based Decoder

Figure 3.2 DCT-based Coder Processing Steps

During encoding the samples of the component are grouped into 8×8 blocks; and, after level shifting, each block is transformed by the forward DCT (FDCT) into the corresponding 8×8 block of DCT coefficients. One coefficient represents the average over the level-shifted block of samples, therefore it is referred to as the DC coefficient. The remaining 63 coefficients are referred to as AC coefficients. Each of the 64 coefficients is then quantized, i.e. scaled and truncated, using one of 64 corresponding values from a quantization table. After quantization the DC coefficient and the AC coefficients are prepared for entropy coding. The quantized DC coefficient of the previous block is used to predict the quantized DC coefficient of the current block, and the difference is encoded. The quantized DCT coefficients are reordered into a 1-D array using a fixed zigzag sequence, i.e. scan path; and zero-valued AC coefficients are run-length coded. For further compression Huffman or arithmetic

coding is employed to entropy-encode the intermediate sequence of symbols, producing a continuous stream of data. Huffman tables, i.e. codes, are either predefined or computed specifically for a given image in an initial statistics-gathering pass prior to Huffman-encoding. Although arithmetic coding adapts to the statistics as it encodes the intermediate sequence of symbols, statistical conditioning tables can improve efficiency. The same tables used during quantization and entropy-encoding are needed during dequantization and entropy-decoding respectively.

Each processing step within the decoder performs essentially the inverse of its counterpart within the encoder. During decoding the entropy decoder decodes the continuous stream of data; and generates the intermediate sequence of symbols, that reassembles the 8×8 block of quantized DCT coefficients. The dequantizer produces dequantized DCT coefficients by rescaling the quantized DCT coefficients using the corresponding values from the quantization table. The inverse DCT (IDCT) generates an 8×8 block of reconstructed samples; that, after level-shifting, approximates the original block of samples.

In the baseline sequential process, used in this section for a more detailed description of the coder processing steps, 8-bit precision image samples transform to 11-bit precision DCT coefficients, and entropy coding employs Huffman coding. Appendix D contains a worked example.

3.4.2 Level Shift prior to Forward Discrete Cosine Transform

The source samples of a component are unsigned integers in the range [0,255]. However, in order to reduce the internal precision requirements in the DCT calculations (W. B. Pennebaker and J. L. Mitchell 1992, p. 38), the samples are shifted to the range

$[-128,127]$ by subtracting 128 from every sample. More generally, samples in the range $\left[0,\left(2^{P}-1\right)\right]$ are shifted to the range $\left[-2^{P-1},\left(2^{P-1}-1\right)\right]$ by subtracting 2^{P-1}, where P is the precision in bits. This processing step is omitted in figure 3.2.

3.4.3 8×8 Forward Discrete Cosine Transform

The purpose of the FDCT processing step is to remove inter-element redundancy by converting statistically dependent sample values into a set of 'less correlated' or 'more independent' coefficients. Note that the DCT is a one-to-one mapping; and it is, therefore, in principle fully reversible, i.e. lossless.

The samples of a component are grouped into 8×8 blocks as defined by the JPEG standard. Each block of samples is a 64-point discrete signal that is a function of the two spatial dimensions y and x. As shown in figure 3.3, the FDCT is used to transform, i.e. decompose, an 8×8 block of samples s into an 8×8 block of DCT coefficients S that is uniquely determined by the particular 64-point input signal. Each DCT coefficient $S(v,u)$ represents one of 64 unique 2-D spatial frequencies. Since coefficient $S(0,0)$ represents zero frequency in both directions, it is referred to as the DC coefficient. The horizontal DCT frequency increases from left to right and the vertical DCT frequency increases from top to bottom. The remaining 63 coefficients are referred to as AC coefficients. Because sample values usually vary slowly from sample to sample, the FDCT processing step concentrates most of the signal energy in the lower spatial frequencies.

$$\underset{\text{samples}}{\begin{bmatrix} s(0,0) & s(0,1) & . & s(0,7) \\ s(1,0) & s(1,1) & . & s(1,7) \\ . & . & s(y,x) & . \\ s(7,0) & s(7,1) & . & s(7,7) \end{bmatrix}} \xrightarrow{FDCT} \underset{DCT\ coefficients}{\begin{bmatrix} S(0,0) & S(0,1) & . & S(0,7) \\ S(1,0) & S(1,1) & . & S(1,7) \\ . & . & S(v,u) & . \\ S(7,0) & S(7,1) & . & S(7,7) \end{bmatrix}}$$

Figure 3.3 8×8 Forward DCT

The ideal functional definition of the FDCT is:

$$S(v,u) = \frac{1}{4} C(v) C(u) \sum_{y=0}^{7} \sum_{x=0}^{7} s(y,x) \cos \frac{(2y+1)v\pi}{16} \cos \frac{(2x+1)u\pi}{16} \tag{3.3}$$

where: $$C(u), C(v) = \begin{cases} 1/\sqrt{2} & \text{for } u, v = 0 \\ 1 & \text{otherwise} \end{cases}$$

Since equation 3.3 contains transcendental functions, it cannot be computed with perfect accuracy. However, the JPEG standard specifies accuracy requirements for this and other processing steps. The JPEG standard does not specify a unique DCT algorithm, thus it allows innovation and customization. No single algorithm is optimal for all implementations, and research in fast DCT algorithms is ongoing; see (W. B. Pennebaker and J. L. Mitchell 1992, chapter 4) for a summary.

3.4.4 Quantization

The purpose of the quantization processing step is to achieve further compression by representing DCT coefficients with no greater precision than is necessary to achieve the desired image quality. Note that quantization is a many-to-one mapping; and is, therefore, fundamentally lossy.

After the FDCT is computed for a block, each of the 64 DCT coefficients is quantized by a uniform quantizer. An 8×8-element quantization table Q, that is specified by the

application or user, provides the quantizer step size $1 \le Q(v,u) \le 255$ for each DCT coefficient $S(v,u)$; see figure 3.4. The quantization table should be appropriate for the colour coordinate that the component represents. For best subjective quality the quantization table should match the characteristics of the human visual system. As examples, tables C.1 and C.2 in appendix C provide luminance and chrominance quantization tables respectively; see (ISO/IEC 10918-1:1994, annex K).

$$\underset{\text{DCT coefficients}}{\begin{bmatrix} S(0,0) & S(0,1) & . & S(0,7) \\ S(1,0) & S(1,1) & . & S(1,7) \\ . & . & S(v,u) & . \\ S(7,0) & S(7,1) & . & S(7,7) \end{bmatrix}} \xrightarrow{\textit{quantization}} \underset{\text{quantized DCT coefficients}}{\begin{bmatrix} Sq(0,0) & Sq(0,1) & . & Sq(0,7) \\ Sq(1,0) & Sq(1,1) & . & Sq(1,7) \\ . & . & Sq(v,u) & . \\ Sq(7,0) & Sq(7,1) & . & Sq(7,7) \end{bmatrix}}$$

$$\uparrow$$

$$\underset{\text{quantization table}}{\begin{bmatrix} Q(0,0) & Q(0,1) & . & Q(0,7) \\ Q(1,0) & Q(1,1) & . & Q(1,7) \\ . & . & Q(v,u) & . \\ Q(7,0) & Q(7,1) & . & Q(7,7) \end{bmatrix}}$$

Figure 3.4 Quantization

The uniform quantization is defined as division of a DCT coefficient $S(v,u)$ by its corresponding quantizer step size $Q(v,u)$, followed by rounding to the nearest integer:

$$Sq(v,u) = round\left(\frac{S(v,u)}{Q(v,u)}\right) \tag{3.4}$$

Note that the quantized DCT coefficient $Sq(v,u)$ is normalized by the quantizer step size $Q(v,u)$.

3.4.5 DC Encoding and 2-D-to-1-D Zigzag Reordering

The purpose of these processing steps, that are omitted in figure 3.2, is to improve the effectiveness of entropy coding.

Since the DC coefficients of adjacent 8×8 blocks are usually strongly correlated, they are DPCM coded. The quantized DC coefficient of the previous block, DC_{i-1}, is used to predict the quantized DC coefficient of the current block, DC_i; see figure 3.5.

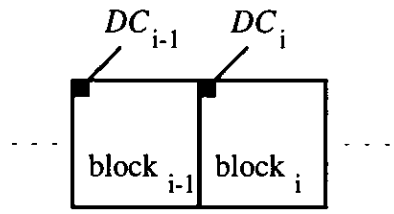


Figure 3.5 DC Coding

The difference, that will be entropy-encoded, is defined as:

$$DIFF = DC_i - PRED \tag{3.5}$$

where $PRED$ is either the quantized DC coefficient of the preceding block, DC_{i-1}; or zero, i.e. mid-range value, at the beginning of a scan.

Each 2-D block of quantized DCT coefficients is rearranged into an 1-D vector, $ZZ(0,\ldots,63)$, utilizing the 8×8 zigzag scan path shown in figure 3.6. $ZZ(0)$ denotes the DC difference value $DIFF$, that replaces the quantized DC coefficient $Sq(0,0)$.

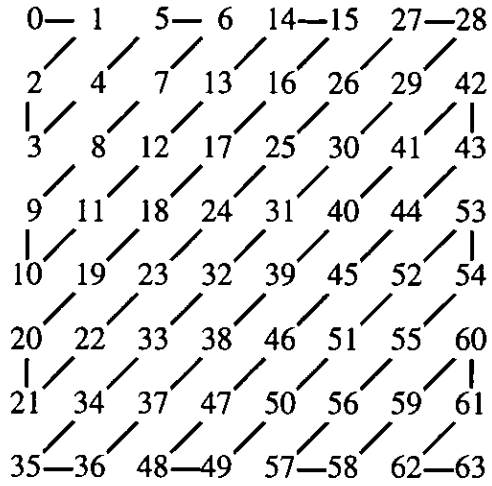


Figure 3.6 8×8 Zigzag Scan Path

Zigzag reordering helps to facilitate entropy coding by placing low-frequency coefficients, that are more likely to be nonzero, before high-frequency coefficients. The probability of coefficients being zero becomes an approximately monotonic increasing function of the index (W. B. Pennebaker and J. L. Mitchell 1992, p. 173). The DC encoding and 2-D-to-1-D zigzag reordering is shown in figure 3.7.

$$\begin{bmatrix} Sq(0,0) & Sq(0,1) & . & Sq(0,7) \\ Sq(1,0) & Sq(1,1) & . & Sq(1,7) \\ . & . & Sq(v,u) & . \\ Sq(7,0) & Sq(7,1) & . & Sq(7,7) \end{bmatrix}$$

quantized DCT coefficients

↓ *zigzag reordering*

$$[DIFF \quad Sq(0,1) \quad Sq(1,0) \quad Sq(2,0) \quad Sq(1,1) \quad \ldots \quad Sq(7,6) \quad Sq(7,7)]$$

vector

Figure 3.7 DC Encoding and 2-D-to-1-D Zigzag Reordering

3.4.6 Huffman Encoding

The purpose of the entropy-encoding processing step is to achieve additional compression by losslessly encoding the quantized and reordered DCT coefficients, i.e. by exploiting coding redundancy due to their statistical characteristics. After converting the vector into an intermediate sequence of symbols, this sequence is converted into a continuous stream of data. The baseline sequential process implements Huffman coding.

Each vector of quantized and reordered coefficients is converted into an intermediate sequence of symbols treating the DC difference value and the AC coefficients similarly but separately. The Huffman-encoding process segments the DC difference value and each nonzero AC coefficient into a set of approximately logarithmically increasing magnitude categories as shown in table 3.4. Note that only DC difference categories 0 to B and AC categories 1 to A are available in the baseline sequential process as indicated by the dotted line. Each category is a symbol and will be assigned a Huffman codeword. However, except for categories 0 and 10 the categories do not fully describe the values to be coded. Therefore, immediately following each codeword for a category $1 \leq K \leq F$, an additional K bits are appended to identify the sign and fully specify the magnitude of the value to be coded. For a positive value the K least-significant bits (LSBs) of the value are appended; for a negative value the K LSBs of the value minus one are appended. Table 3.5 outlines the additional bit sequences. Note that leading bits equal to one identify positive values, and leading bits equal to zero identify negative values.

Range	DC Difference Category (hexadecimal)	AC Category (hexadecimal)
0	0	n/a
-1,1	1	1
-3,-2,2,3	2	2
-7,...,-4,4,...,7	3	3
-15,...,-8,8,...,15	4	4
-31,...,-16,16,...,31	5	5
-63,...,-32,32,...,63	6	6
-127,...,-64,64,...,127	7	7
-255,...,-128,128,...,255	8	8
-511,...,-256,256,...,511	9	9
-1023,...,-512,512,...,1023	A	A
-2047,...,-1024,1024,...,2047	B	B
-4095,...,-2048,2048,...,4095	C	C
-8191,...,-4096,4096,...,8191	D	D
-16383,...,-8192,8192,...,16383	E	E
-32767,...,-16384,16384,...,32767	F	F
32768	10	n/a

n/a: not applicable

Table 3.4 Magnitude Categories for Huffman Coding

Range	Category (hexadecimal)	Additional Bits (binary)
0	0	n/a
-1,1	1	0,1
-3,-2,2,3	2	00,01,10,11
-7,...,-4,4,...,7	3	000,...,011,100,...,111
-15,...,-8,8,...,15	4	0000,...,0111,1000,...,1111
-31,...,-16,16,...,31	5	00000,...,01111,10000,...,11111
...	...	...
32768	10	n/a

n/a: not applicable

Table 3.5 Additional Bits for Sign and Magnitude

Using an appropriate DC table, the DC difference value of a vector is encoded through a codeword representing the DC difference category, and additional bits that may be required. As examples, table C.3 and C.4 in appendix C provide luminance and chrominance DC difference tables respectively; see (ISO/IEC 10918-1:1994, annex K).

Before the nonzero AC coefficients of a vector are encoded in a similar manner, consecutive zero AC coefficients are aggregated into runs of zeros. Each run of zeros in the range [0,15] is combined with the magnitude category of the nonzero AC coefficient that terminates the run of zeros to give a compound symbol as shown in table 3.6. Note that only AC categories 0 to A are available in the baseline sequential process as indicated by the dotted line. An extension symbol, referred to as zero run length (ZRL), codes a run of 16 zeros. Therefore, runs of zeros longer than 15 are represented through up to three extension symbols preceding a terminating compound symbol. A special symbol, referred to as end-of-block (EOB), is used to terminate a vector when all remaining AC coefficients are zero. However, for the condition that the last coefficient in a vector is nonzero, the EOB symbol is not generated.

Zero	AC Category (hexadecimal)															
Run	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	EOB	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
1	n/a	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
2	n/a	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
3	n/a	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
4	n/a	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
5	n/a	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
6	n/a	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
7	n/a	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
8	n/a	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
9	n/a	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
10	n/a	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
11	n/a	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
12	n/a	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
13	n/a	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
14	n/a	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
15	ZRL	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

n/a: not applicable

Table 3.6 Coding Symbols for Huffman Coding of AC Coefficients

Using an appropriate AC table, each nonzero AC coefficient of a vector is encoded through zero to three extension symbols; one compound symbol, representing the run of

remaining zero AC coefficients and the AC category; and additional bits. As an example, table C.5 in appendix C provides a luminance AC table; see (ISO/IEC 10918-1:1994, annex K).

The JPEG-compatible image data stream to be transmitted or stored consists of entropy-coded data segments, that contain the entropy-coded image data; and marker segments, that contain parameters, e. g. headers and tables.

3.4.7 Huffman Decoding

The entropy decoder decodes each vector of quantized and reordered DCT coefficients from the entropy-coded image data using the appropriate tables. Since each Huffman-coded category exactly defines the number of additional bits appended to each category, the stream of data is uniquely decodable.

3.4.8 1-D-to-2-D Zigzag Reordering and DC Decoding

Each 1-D vector is rearranged back into a 2-D block of quantized DCT coefficients utilizing the 8×8 zigzag scan path shown in figure 3.6. The 1-D-to-2 D zigzag reordering and DC decoding is shown in figure 3.8.

$$[DIFF \quad Sq(0,1) \quad Sq(1,0) \quad Sq(2,0) \quad Sq(1,1) \quad \ldots \quad Sq(7,6) \quad Sq(7,7)]$$

vector

$$\downarrow \textit{ zigzag reordering}$$

$$\begin{bmatrix} Sq(0,0) & Sq(0,1) & . & Sq(0,7) \\ Sq(1,0) & Sq(1,1) & . & Sq(1,7) \\ . & . & Sq(v,u) & . \\ Sq(7,0) & Sq(7,1) & . & Sq(7,7) \end{bmatrix}$$

quantized DCT coefficients

Figure 3.8 1-D-to-2-D Zigzag Reordering and DC Decoding

The quantized DC coefficient $Sq(0,0)$ replaces the DC difference value $DIFF$. The quantized DC coefficient of the current block, DC_i, is obtained by adding the difference value to the prediction value:

$$DC_i = PRED + DIFF \tag{3.6}$$

where $PRED$ is either the quantized DC coefficient of the preceding block, DC_{i-1}, or zero at the beginning of a scan.

3.4.9 Dequantization

As shown in figure 3.9, the dequantization processing step is used to denormalize, i.e. rescale, each 8×8 block of quantized DCT coefficients, S_q, into an 8×8 block of dequantized DCT coefficients, R.

$$\underset{\textit{quantized DCT coefficients}}{\begin{bmatrix} Sq(0,0) & Sq(0,1) & . & Sq(0,7) \\ Sq(1,0) & Sq(1,1) & . & Sq(1,7) \\ . & . & Sq(v,u) & . \\ Sq(7,0) & Sq(7,1) & . & Sq(7,7) \end{bmatrix}} \xrightarrow{\textit{dequantization}} \underset{\textit{dequantized DCT coefficients}}{\begin{bmatrix} R(0,0) & R(0,1) & . & R(0,7) \\ R(1,0) & R(1,1) & . & R(1,7) \\ . & . & R(v,u) & . \\ R(7,0) & R(7,1) & . & R(7,7) \end{bmatrix}}$$

$$\uparrow$$

$$\underset{\textit{quantization table}}{\begin{bmatrix} Q(0,0) & Q(0,1) & . & Q(0,7) \\ Q(1,0) & Q(1,1) & . & Q(1,7) \\ . & . & Q(v,u) & . \\ Q(7,0) & Q(7,1) & . & Q(7,7) \end{bmatrix}}$$

Figure 3.9 Dequantization

The dequantization, that removes the normalization, is defined as multiplication of a quantized DCT coefficient $Sq(v,u)$ by its corresponding quantizer step size $Q(v,u)$:

$$R(v,u) = Sq(v,u)\, Q(v,u) \tag{3.7}$$

3.4.10 8×8 Inverse Discrete Cosine Transform

As shown in figure 3.10, the IDCT processing step is used to transform, i.e. compose, each 8×8 block of dequantized DCT coefficients, R, into an 8×8 block of reconstructed samples, r.

$$\underset{\textit{dequantized DCT coefficients}}{\begin{bmatrix} R(0,0) & R(0,1) & . & R(0,7) \\ R(1,0) & R(1,1) & . & R(1,7) \\ . & . & R(v,u) & . \\ R(7,0) & R(7,1) & . & R(7,7) \end{bmatrix}} \xrightarrow{IDCT} \underset{\textit{reconstructed samples}}{\begin{bmatrix} r(0,0) & r(0,1) & . & r(0,7) \\ r(1,0) & r(1,1) & . & r(1,7) \\ . & . & r(y,x) & . \\ r(7,0) & r(7,1) & . & r(7,7) \end{bmatrix}}$$

Figure 3.10 8×8 Inverse DCT

The ideal functional definition of the IDCT is:

$$r(y,x)=\frac{1}{4}\sum_{v=0}^{7}\sum_{u=0}^{7}C(v)\,C(u)\,R(v,u)\cos\frac{(2y+1)v\pi}{16}\cos\frac{(2x+1)u\pi}{16} \tag{3.8}$$

where: $$C(u),C(v)=\begin{cases}1/\sqrt{2} & \text{for } u,v=0\\ 1 & \text{otherwise}\end{cases}$$

3.4.11 Level Shift after Inverse Discrete Cosine Transform

The samples are shifted from the range $[-128,127]$ back to the original range $[0,255]$ by adding 128 to every sample.

3.5 Remarks

The JPEG standard provides a complex framework and caters for a wide range of different applications. It does boost the utilization of digital images in general-purpose computer systems, and the exchange of compressed data among applications and across computer systems. However, the DCT-based lossy method is most popular. It, being a

transform-based coding technique, shares the advantages and disadvantages described in subsections 2.5.1 and 2.5.5, namely the introduction of blocking artefacts as the bit rate is decreased (R. J. Clarke 1995, pp. 86 and 162; and W. B. Pennebaker and J. L. Mitchell 1992, p. 38).

The JPEG standard provides examples of quantization tables (ISO/IEC 10918-1:1994, annex K), but does not specify default quantization tables. The application or user must provide quantization tables tailored to particular image characteristics, display devices, and viewing conditions (ISO/IEC 10918-1:1994, section 3.3). Although quantization values of individual DCT coefficients should be at the threshold of visibility, little is known about visibility thresholds when two or more DCT coefficients are nonzero, i.e. when masking occurs (W. B. Pennebaker and J. L. Mitchell 1992, pp. 36-38). Therefore, the difficult part of the problem is left to the application or user (M. A. Sid-Ahmed 1995, p. 478). Furthermore, only one of the up to four available quantization tables is globally used for all blocks of an image component within a frame discounting local changes in block content, i.e. complexity. A. B. Watson (1993a and b) developed a design procedure that generates an image-dependent perceptually optimum quantization table, however the quantization table cannot be changed within a component. To enhance the JPEG encoder, N. Jayant et al. (1993) outlined a perceptual preprocessor that uses prequantization to eliminate, i.e. set to zero, each DCT coefficient that is less than its corresponding visual threshold prior to the normal quantization processing step; thus maintaining JPEG-compatible image data streams and supporting any JPEG decoder. Although the JPEG standard employs quantization tables, each DCT coefficient is independently processed using a scalar quantization; this process is

inferior to vector quantization (R. J. Clarke 1995, p. 91). R. J. Clarke (1995 pp. 121-124) described the combination of transform coding and vector quantization.

3.6 Summary

JPEG, established in 1986, generated an international standard for digital compression of continuous-tone still images with the aim to boost the utilization of digital images in general-purpose computer systems. The JPEG standard defines sequential, progressive, lossless, and hierarchical modes of operation. While the lossy modes of operation utilize a DCT-based method; the lossless mode of operation is based on a predictive method. However, both methods employ either Huffman or arithmetic coding for entropy coding. A particular restricted form of the DCT-based sequential mode is known as the baseline sequential process. It represents a minimum capability that must be present in all DCT-based decoder systems.

For the baseline sequential encoding process each component of an input image is divided into 8×8 blocks, each of which is then transformed using the FDCT. The DCT coefficients are quantized using a user-specifiable quantization table. The quantized DCT coefficients are zigzag reordered and losslessly entropy-encoded using Huffman coding. Each processing step within the decoder performs essentially the inverse of its counterpart within the encoder.

The application or user must provide the quantization tables. The JPEG standard utilizes only one quantization table for an image component.

Chapter 4

Adaptive Zigzag Reordering of Transform Coefficients

4.1 Introduction

This chapter describes adaptive zigzag reordering for blocks of transform coefficients in JPEG-like image-compression schemes. Efficient reordering is achieved using variable-size rectangular sub-blocks. If the generated sub-blocks include all nonzero coefficients, the conversion is fully reversible, i.e. lossless. The zigzag scan paths are generated using a binary decision tree.

Section 4.2 discusses standard zigzag reordering of transform coefficients, used in the DCT-based method of the JPEG standard and introduced in chapter 3, in more detail.

Section 4.3 describes adaptive zigzag reordering, and draws a comparison with standard zigzag reordering using experimental results.

Section 4.4 develops a versatile zigzag-reordering algorithm that employs a binary decision tree.

Section 4.5 focuses on a hardware implementation of the zigzag-reordering algorithm that uses two GAL16V8 devices.

Section 4.6 addresses coding of the sub-block dimensions. Finally section 4.7 concludes the chapter with a brief summary.

4.2 Standard Zigzag Reordering

A generic 8×8 block of quantized transform coefficients, used in the DCT-based method of the JPEG standard, is shown in figure 4.1. Coefficient $Sq(0,0)$ represents zero frequency in horizontal and vertical directions. The horizontal DCT frequency

increases from left to right, and the vertical DCT frequency increases from top to bottom; see subsection 3.4.3.

$$\begin{bmatrix} Sq(0,0) & Sq(0,1) & . & Sq(0,7) \\ Sq(1,0) & Sq(1,1) & . & Sq(1,7) \\ . & . & Sq(v,u) & . \\ Sq(7,0) & Sq(7,1) & . & Sq(7,7) \end{bmatrix}$$

Figure 4.1 8×8 Block of Quantized DCT Coefficients

Reordering along a fixed 8×8 zigzag scan path, depicted in figure 4.2, approximately arranges the coefficients from low to high DCT frequencies (W. B. Pennebaker and J. L. Mitchell 1992, p. 34); see subsection 3.4.5.

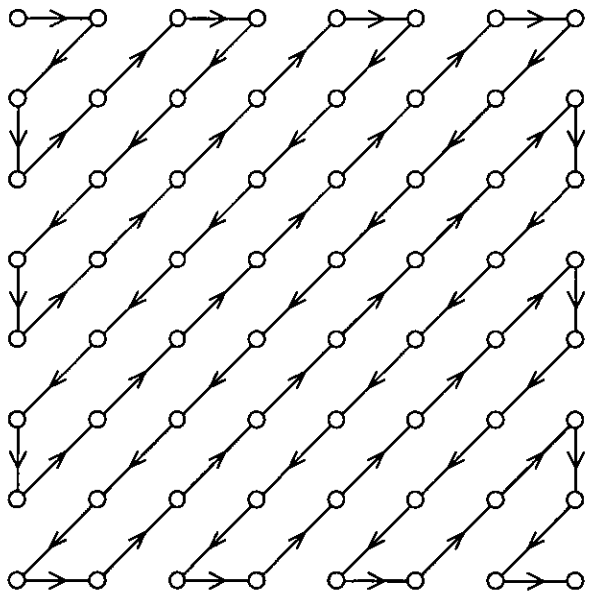


Figure 4.2 8×8 Zigzag Scan Path

Since low-frequency coefficients are more likely to be nonzero than high-frequency coefficients, the zigzag-reordered coefficients exhibit an approximately monotonic increasing probability of being zero; see (W. B. Pennebaker and J. L. Mitchell 1992, p. 173).

The entropy-coding processing step generates an intermediate sequence of symbols. While each nonzero coefficient is variable-length coded; each run of zero coefficients, i.e. zero run, is run-length coded; see subsection 3.4.6. Zigzag reordering is an important processing step; since it affects the zero runs, and therefore changes the statistics of the symbols used during entropy coding.

As an example, figure 4.3 depicts in a logarithmic scale the probability distribution of the zero runs preceding the last nonzero coefficient for standard 8×8 zigzag reordering, and image Lena with a spatial resolution of 512×512 pixels for quality setting $q = 50$. The probability of occurrence decreases as the length of zero run increases. Note that the use of extension symbols, coding zero runs longer than 15, is not taken into account. However, zero runs of lengths 16, 17, 19, 20, 21, and above 22 do not occur.

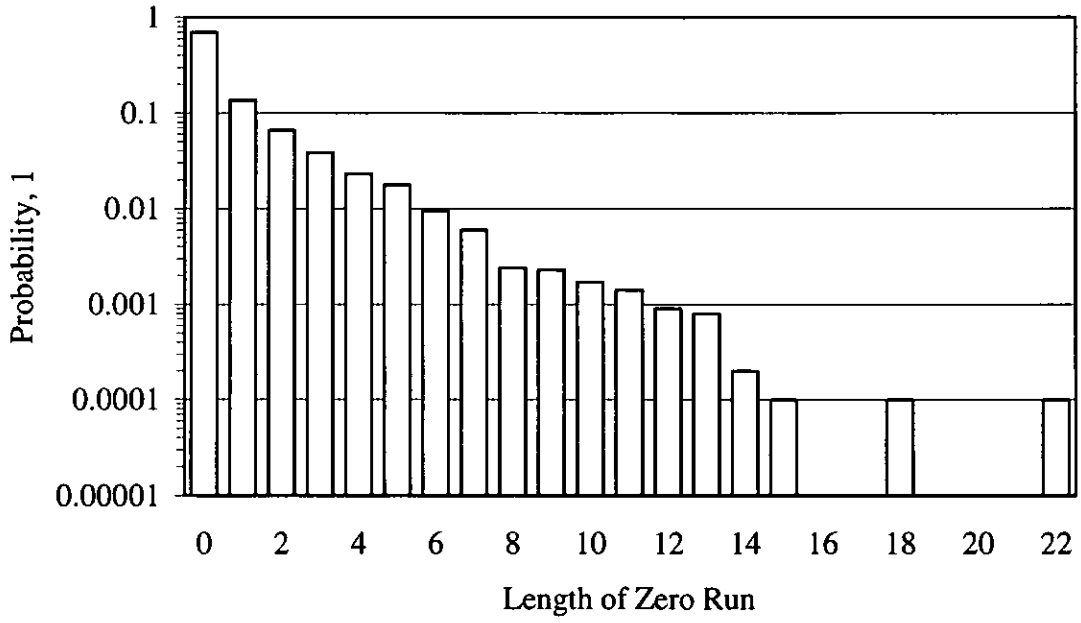


Figure 4.3 Probability Distribution of Runs of Zero Coefficients,

Standard Zigzag Reordering, Lena 512×512, $q = 50$

Note that a corresponding entropy for the zero runs of 1.63 bits has been calculated using equation 2.7. Figure 4.4 shows the image Lena with a spatial resolution of 512×512 pixels for quality setting $q = 50$. Subsection 4.3.3 provides details on the experimentation.

Figure 4.4 Decoded JPEG Image, Lena 512×512, $q = 50$

4.3 Adaptive Zigzag Reordering

4.3.1 Motivation for Adaptive Zigzag Reordering

The JPEG standard for the DCT-based method defines one fixed 8×8 zigzag scan path for coefficient reordering that is used for every block of DCT coefficients regardless of specific block content. Although this processing step approximately arranges the coefficients in order of increasing DCT frequency, and increasing probability of being zero; it does not directly address the symbol statistics for entropy coding.

Adaptive zigzag reordering processes an $L \times M$ sub-block that is yielded from the $L_{max} \times M_{max}$ block of coefficients, where $L_{max} = 8$ and $M_{max} = 8$ for the DCT-based method of the JPEG standard. This sub-block is not necessarily square, but is rectangular with the dimensions $1 \le L \le L_{max}$ rows and $1 \le M \le M_{max}$ columns. Note that the sub-block is defined to include the DC coefficient. By taking the specific block content into account, adaptive zigzag reordering reduces the entropy of the symbols, and thus improves efficiency of entropy coding.

4.3.2 Determination of Sub-blocks

For transcoding, i.e. lossless conversion of a block of coefficients, the sub-block must contain all nonzero coefficients. Hence the smallest possible rectangle to include all nonzero coefficients is identified. The coefficients within the sub-block are then zigzag-reordered using a zigzag scan path that is appropriate for the dimensions of the sub-block. Since sub-blocks generally have different dimensions, reordering is no longer a straightforward task; section 4.4 describes a zigzag-reordering algorithm based on a binary decision tree. The dimensions of the sub-block need to be retained in order to

traverse the zigzag scan path correctly during decoding; section 4.6 addresses coding of the sub-block dimensions.

As an example, figure 4.5 depicts an 8×8 block of transform coefficients with the corresponding 4×5 sub-block indicated by the dotted line.

$$\left[\begin{array}{rrrrr:rrr} -26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\ 1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Figure 4.5 Example of 8×8 Block of Transform Coefficients

Standard zigzag reordering of the block using the fixed 8×8 zigzag scan path is shown in figure 4.6. The nonzero coefficients are indicated by black dots. There are twelve zero runs of length zero, two zero runs of one, one zero run of two, and one zero run of five.

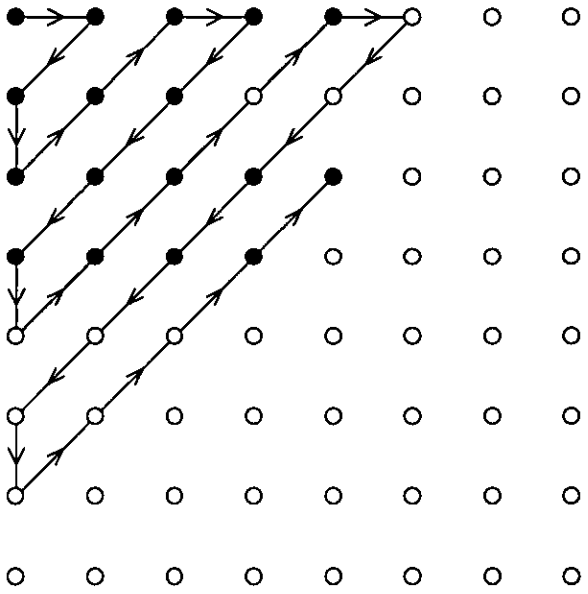


Figure 4.6 Example of Standard Zigzag Reordering

Adaptive zigzag reordering of the sub-block using the appropriate 4×5 zigzag scan path is shown in figure 4.7. The nonzero coefficients are indicated by black dots. There are 14 zero runs of length zero, and two zero runs of one.

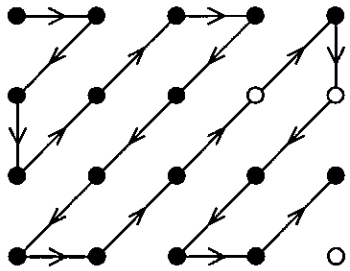


Figure 4.7 Example of Adaptive Zigzag Reordering

Adaptive zigzag reordering reduces the length of the zero runs as well as the number of different lengths of zero runs. However, it does not change the total number of zero runs. It modifies the probability distribution of runs of zero coefficients so that the entropy of the symbols is reduced and the potential effectiveness of entropy coding is improved.

Following the example used in figure 4.3, figure 4.8 depicts in a logarithmic scale the probability distribution of the zero runs preceding the last nonzero coefficient for adaptive zigzag reordering, and image Lena with a spatial resolution of 512×512 pixels for quality setting $q = 50$. The probability of occurrence decreases more rapidly as the length of zero run increases. Zero runs of lengths 10, 12, and above 13 do not occur. Note that a corresponding entropy for the zero runs of 1.15 bits has been calculated using equation 2.7.

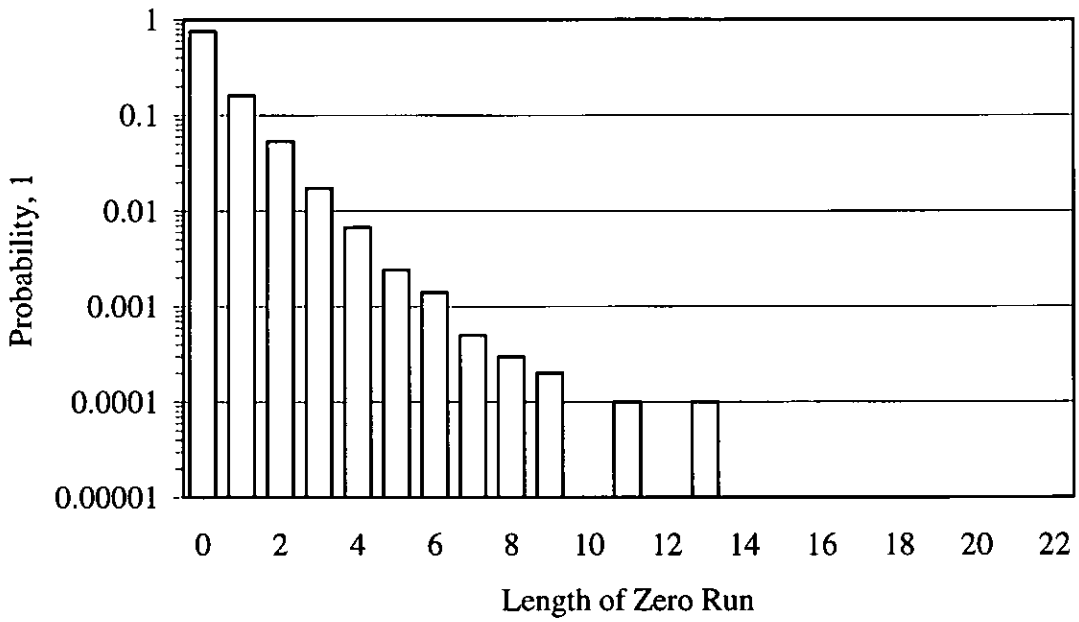


Figure 4.8 Probability Distribution of Runs of Zero Coefficients,

Adaptive Zigzag Reordering, Lena 512×512, $q = 50$

Figure 4.9 depicts the probability distribution of the sub-block dimensions for image Lena with a spatial resolution of 512×512 pixels for quality setting $q = 50$. The probability distribution is formed along the diagonal, hence sub-blocks tend to be approximately square. As a result of using quality setting $q = 50$, the probability of occurrence decreases as the row and column dimensions increase.

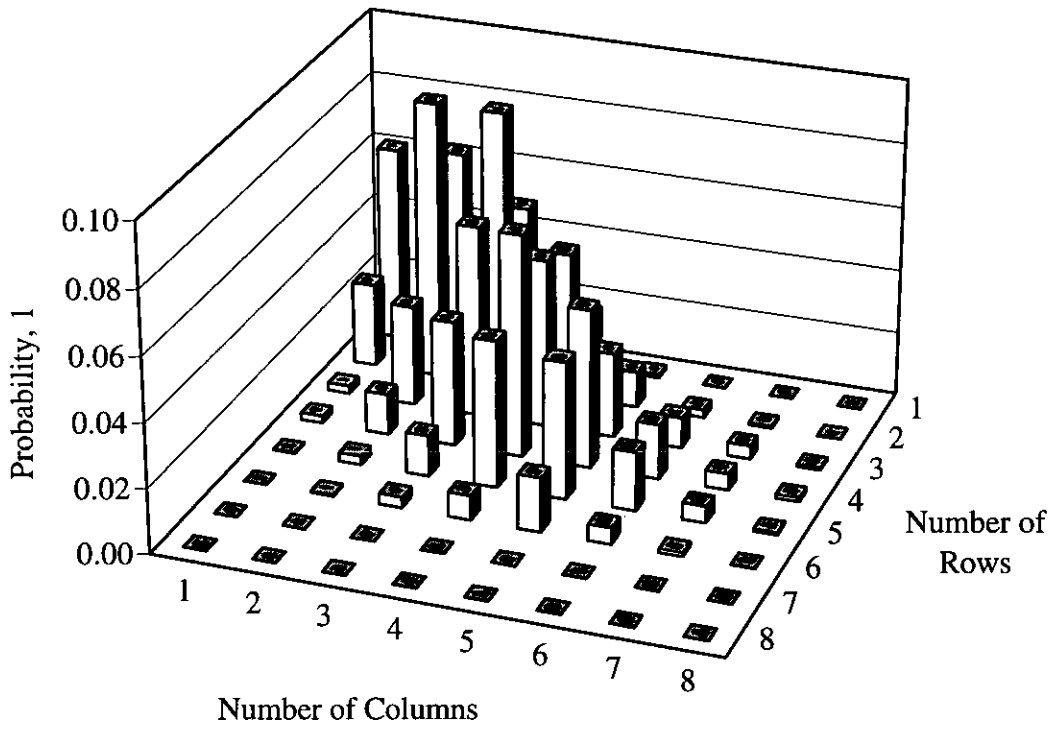


Figure 4.9 Probability Distribution of Sub-block Dimensions,

Lena 512×512, $q = 50$

4.3.3 Experimental Results

Experimental results have been obtained using MATLAB (MathWorks 1994). The transform-coefficient matrices have been generated using the Independent JPEG Group's software (Independent JPEG Group 1996). The quality setting q controls scaling of the quantization tables; see subsection 3.4.4. The experimental results have been produced for quality settings in the range from 10 ('poor' quality) to 90 ('good' quality). Appendix E contains the original images used for experimentation. Note that processing based on 8×8 blocks for images with spatial resolutions of 256×256 and 512×512 pixels involves 1024 and 4096 blocks respectively.

The entropies of the runs of zero coefficients for standard and adaptive zigzag reordering have been evaluated over the range of quality settings. Figures 4.10 and 4.11

compare the entropies for the image Lena with a spatial resolution of 512×512 and 256×256 pixels respectively. Figure 4.12 compares the entropies for the image Cameraman with a spatial resolution of 256×256 pixels. Figure 4.13 compares the entropies for the image F-16 with a spatial resolution of 512×512 pixels. For adaptive zigzag reordering, the entropy of runs of zero coefficients is always lower than that for standard zigzag reordering.

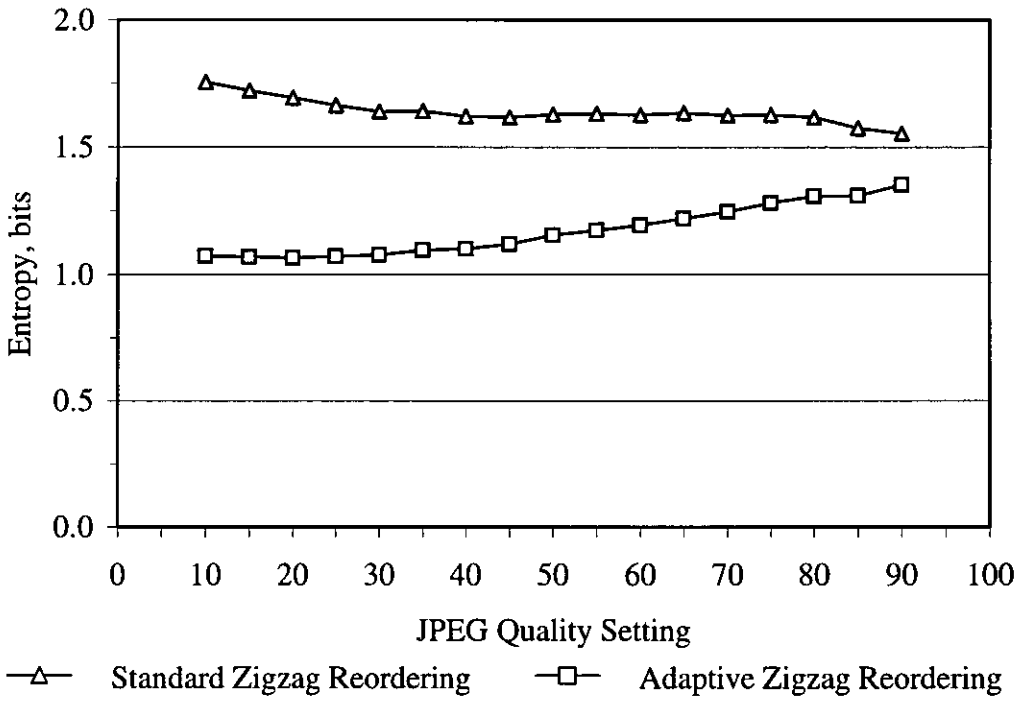


Figure 4.10 Entropy of Runs of Zero Coefficients versus Quality Setting, Lena 512×512

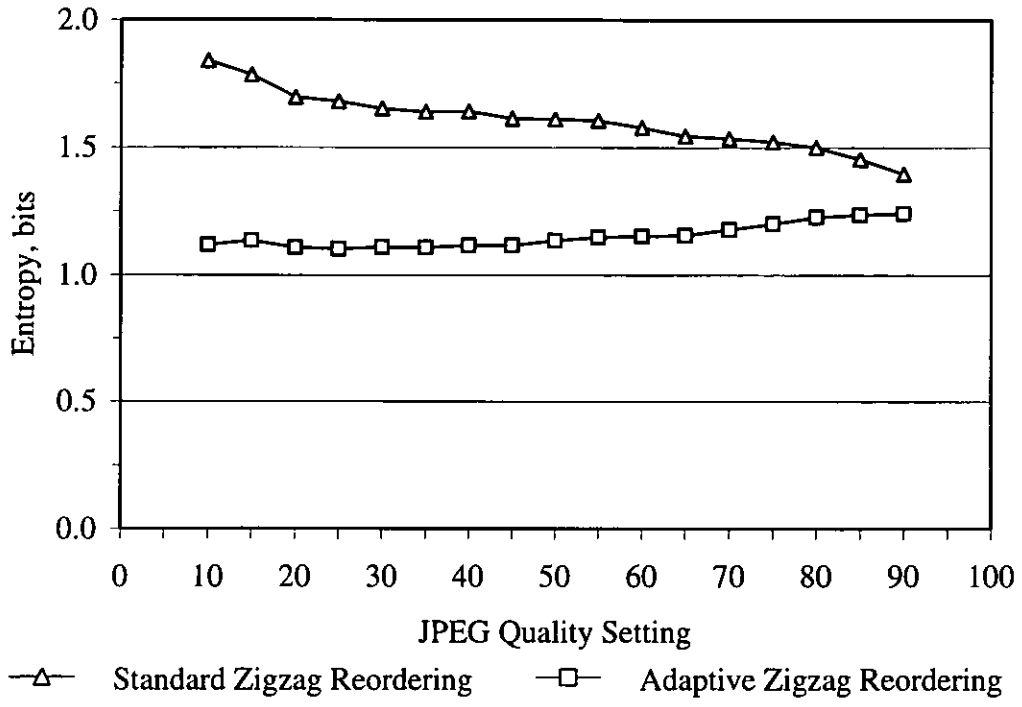


Figure 4.11 Entropy of Runs of Zero Coefficients versus Quality Setting,

Lena 256×256

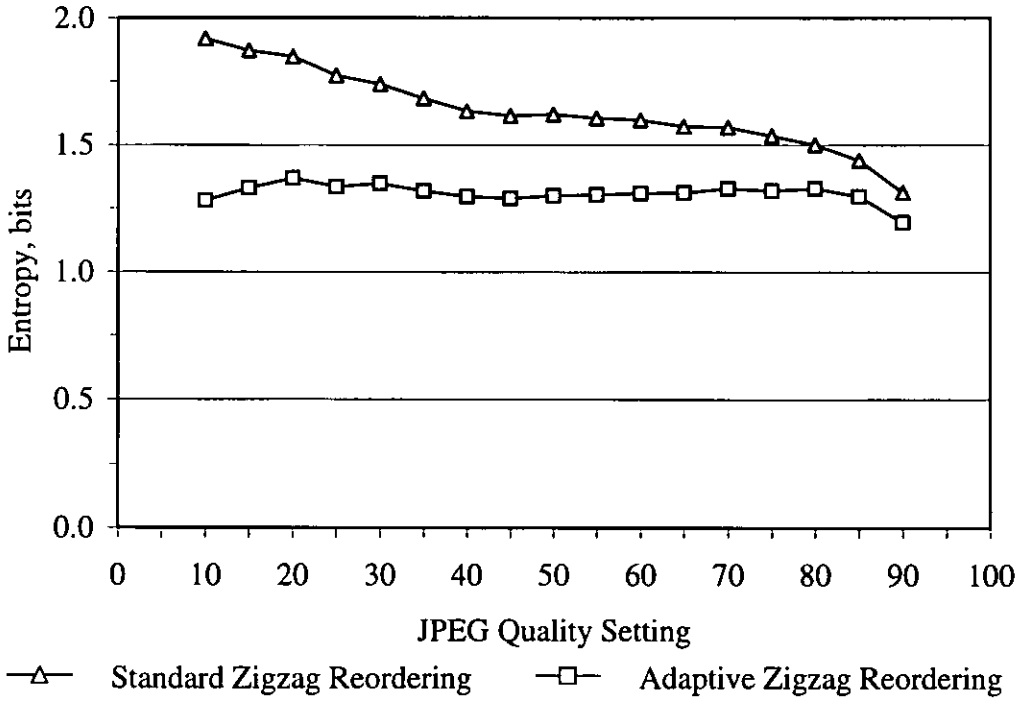


Figure 4.12 Entropy of Runs of Zero Coefficients versus Quality Setting,

Cameraman 256×256

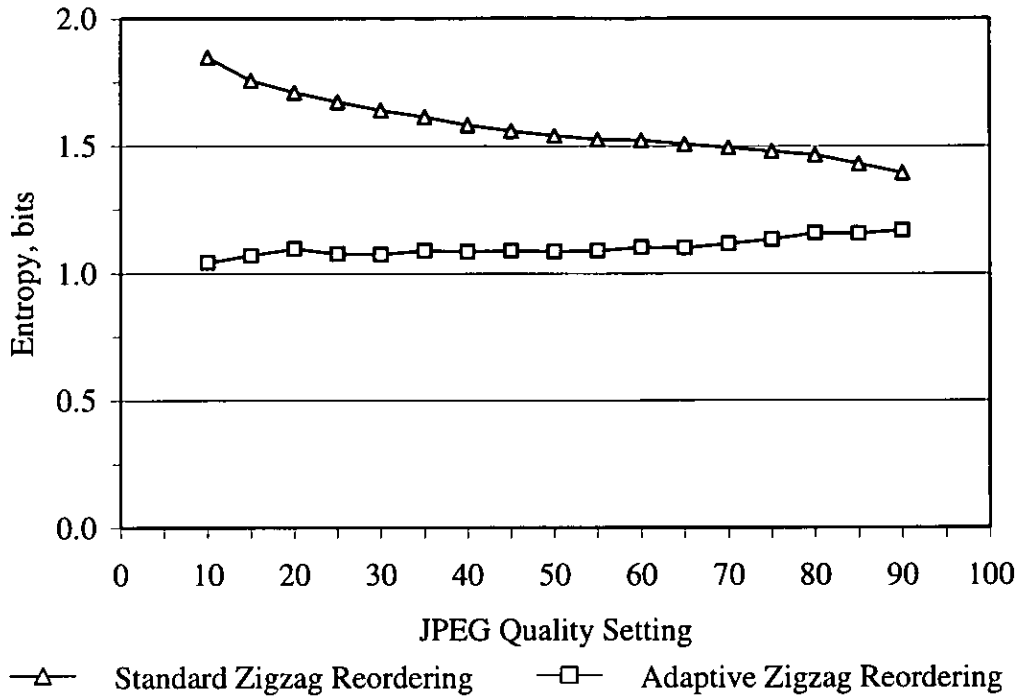


Figure 4.13 Entropy of Runs of Zero Coefficients versus Quality Setting,

F-16 512×512

Figure 4.14 summarizes the percentage entropy reduction over the range of quality settings for the four images. Adaptive zigzag reordering consistently produces a lower entropy indicating improved efficiency for entropy coding. For higher quality settings the number of nonzero coefficients increases, and therefore the sub-block dimensions approach the standard 8×8 block dimensions more frequently. However, for the images analysed, a significant entropy reduction of at least 15 % has been obtained for 'medium' quality settings ($q = 30,35,\ldots,70$).

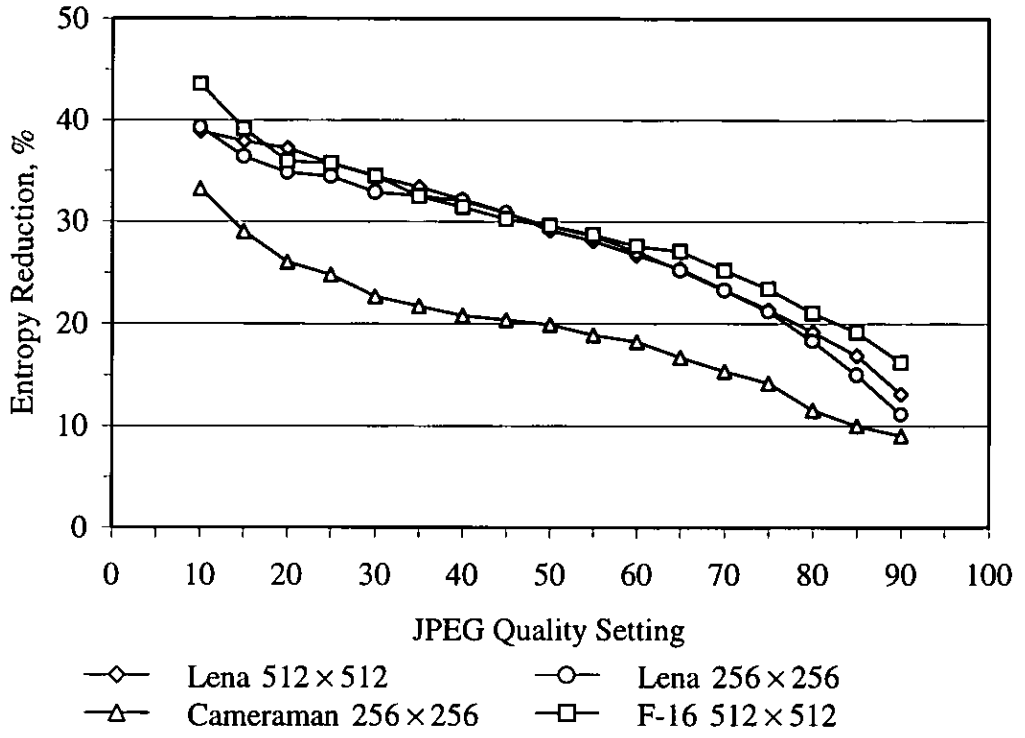


Figure 4.14 Entropy Reduction for Runs of Zero Coefficients versus Quality Setting

4.4 Versatile Zigzag-reordering Algorithm

4.4.1 Motivation for Versatile Zigzag-reordering Algorithm

Adaptive zigzag reordering reduces the entropy of the runs of zero coefficients by traversing a scan path that is tailored to the dimensions of a rectangular sub-block in a particular block of quantized transform coefficients. Although scan paths may be derived and provided in advance for all required sub-block dimensions, a versatile algorithm is more flexible and appropriate; especially when the sub-block dimensions, and therefore the number of possible scan paths and the lengths of the scan paths, increase. A versatile algorithm generates scan paths for sub-blocks of any dimensions. It may determine the scan paths "on the fly", i.e. as the scan path of a particular sub-block is being traversed. In addition, the algorithm may also be implemented in hardware.

The coordinates of the next element in the zigzag scan path; and therefore the whole zigzag scan path; can be determined through Boolean expressions that evaluate the coordinates of the current element, and the dimensions of the sub-block. The versatile zigzag-reordering algorithm described in this section is based on a binary decision tree using a sequence of three binary tests to determine the coordinates of the next element.

4.4.2 The Sub-block

A sub-block is defined as matrix $A(L,M)$ of L rows by M columns:

$$A(L,M)=\begin{bmatrix} a(1,1) & a(1,2) & . & a(1,M) \\ a(2,1) & a(2,2) & . & a(2,M) \\ . & . & a(l,m) & . \\ a(L,1) & a(L,2) & . & a(L,M) \end{bmatrix} \tag{4.1}$$

with $1 \le l \le L$ and $1 \le m \le M$.

Zigzag reordering that starts at the top left-hand position, as shown in figures 4.6 and 4.7 for two examples, utilizes four directions of movement, as shown in figure 4.15, and no movement at the last position of a sub-block, i.e. the bottom right-hand position.

A move in the upper-right direction requires a decrement of the current row index, indicated by $l--$; and an increment of the current column index, indicated by $m++$. A move in the right direction requires no change to the row index, indicated by l; and an increment of the column index. A move in the lower direction requires an increment of the row index, and no change to the column index. A move in the lower-left direction requires an increment of the row index, and a decrement of the column index.

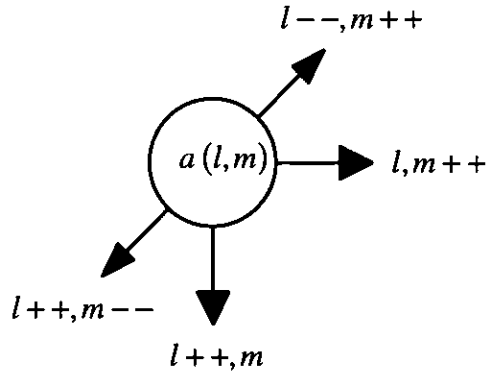


Figure 4.15 Directions of Movement

Certain changes in the row and column indices cannot occur at certain positions; for example the row index cannot be decremented for positions in the first row, and the column index cannot be incremented for positions in the last column. However, the coordinates of the next element, indicated by (l^+, m^+), in the zigzag scan path; and therefore the whole zigzag scan path; can be determined through Boolean expressions evaluating the coordinates of the current element, indicated by (l,m), and the dimensions of the sub-block, L and M.

4.4.3 Parameters

For the versatile zigzag-reordering algorithm five parameters that correspond to binary tests have been defined for convenience:

$Rl(l,m)$ indicates whether the current element $a(l,m)$ is positioned in the first row

$$Rl(l,m) = \begin{cases} 1 & for\ l = 1 \\ 0 & otherwise \end{cases} \tag{4.2}$$

$RL(l,m)$ indicates whether the current element is positioned in the last row

$$RL(l,m) = \begin{cases} 1 & for\ l = L \\ 0 & otherwise \end{cases} \tag{4.3}$$

$C1(l,m)$ indicates whether the current element is positioned in the first column

$$C1(l,m) = \begin{cases} 1 & for\ m = 1 \\ 0 & otherwise \end{cases} \tag{4.4}$$

$CM(l,m)$ indicates whether the current element is positioned in the last column

$$CM(l,m) = \begin{cases} 1 & for\ m = M \\ 0 & otherwise \end{cases} \tag{4.5}$$

$P(l,m)$ indicates whether the sum of row index l and column index m of the current element is odd

$$P(l,m) = \begin{cases} 1 & if\ (l+m)\ is\ odd \\ 0 & otherwise \end{cases} \tag{4.6}$$

The five parameters can be combined and evaluated through Boolean expressions. However, binary matrices of L rows by M columns may be used to represent the five parameters of all elements in an $L \times M$ sub-block compactly.

In matrix $R1(L,M)$ all elements in the first row are one, and the remaining elements are zero:

$$R1(L,M) = \begin{bmatrix} 1 & 1 & . & 1 \\ 0 & 0 & . & 0 \\ . & . & . & . \\ 0 & 0 & . & 0 \end{bmatrix} \tag{4.7}$$

In matrix $RL(L,M)$ all elements in the last row are one, and the remaining elements are zero:

$$RL(L,M)=\begin{bmatrix}0 & 0 & . & 0\\ . & . & . & .\\ 0 & 0 & . & 0\\ 1 & 1 & . & 1\end{bmatrix} \tag{4.8}$$

In matrix $C1(L,M)$ all elements in the first column are one, and the remaining elements are zero:

$$C1(L,M)=\begin{bmatrix}1 & 0 & . & 0\\ 1 & 0 & . & 0\\ . & . & . & .\\ 1 & 0 & . & 0\end{bmatrix} \tag{4.9}$$

In matrix $CM(L,M)$ all elements in the last column are one, and the remaining elements are zero:

$$CM(L,M)=\begin{bmatrix}0 & . & 0 & 1\\ 0 & . & 0 & 1\\ . & . & . & .\\ 0 & . & 0 & 1\end{bmatrix} \tag{4.10}$$

In matrix $P(L,M)$ all elements whose sum of row index l and column index m is odd are one, and the remaining elements are zero:

$$P(L,M)=\begin{bmatrix}0 & 1 & 0 & .\\ 1 & 0 & 1 & .\\ 0 & 1 & 0 & .\\ . & . & . & .\end{bmatrix} \tag{4.11}$$

4.4.4 The Truth Table

The truth table, shown in table 4.1, lists all 32 possible combinations of the five binary parameters $R1(l,m)$, $RL(l,m)$, $C1(l,m)$, $CM(l,m)$, and $P(l,m)$; and the corresponding changes in the row and column indices. l and l^+ are the row indices of the current

element and the next element respectively. $l++$ denotes an increment of the row index l, i. e. addition of 1; l denotes no change to the row index l; and $l--$ denotes a decrement of the row index l, i. e. subtraction of 1. Changes in the column index m are identified similarly.

Combination	$R1(l,m)$	$RL(l,m)$	$C1(l,m)$	$CM(l,m)$	$P(l,m)$	l^+	m^+
0	0	0	0	0	0	$l--$	$m++$
1	0	0	0	0	1	$l++$	$m--$
2	0	0	0	1	0	$l++$	m
3	0	0	0	1	1	$l++$	$m--$
4	0	0	1	0	0	$l--$	$m++$
5	0	0	1	0	1	$l++$	m
6	0	0	1	1	0	$l++$	m
7	0	0	1	1	1	$l++$	m
8	0	1	0	0	0	$l--$	$m++$
9	0	1	0	0	1	l	$m++$
10	0	1	0	1	0	l	m
11	0	1	0	1	1	l	m
12	0	1	1	0	0	$l--$	$m++$
13	0	1	1	0	1	l	$m++$
14	0	1	1	1	0	l	m
15	0	1	1	1	1	l	m
16	1	0	0	0	0	l	$m++$
17	1	0	0	0	1	$l++$	$m--$
18	1	0	0	1	0	$l++$	m
19	1	0	0	1	1	$l++$	$m--$
20	1	0	1	0	0	l	$m++$
21	1	0	1	0	1	$l++$	m
22	1	0	1	1	0	$l++$	m
23	1	0	1	1	1	$l++$	m
24	1	1	0	0	0	l	$m++$
25	1	1	0	0	1	l	$m++$
26	1	1	0	1	0	l	m
27	1	1	0	1	1	l	m
28	1	1	1	0	0	l	$m++$
29	1	1	1	0	1	l	$m++$
30	1	1	1	1	0	l	m
31	1	1	1	1	1	l	m

Table 4.1 Complete Truth Table for Changes in Row and Column Indices

If $RL(l,m) = 1$, i.e. the position of the current element is in the last row of the sub-block; and $CM(l,m) = 1$, i.e. the position is in the last column of a sub-block; the position of the current element is the last position in the sub-block, i.e. no changes in the row and column indices are required regardless of $Rl(l,m)$, $Cl(l,m)$, and $P(l,m)$; see combinations 10, 11, 14, 15, 26, 27, 30, and 31.

If $Rl(l,m) = 0$, i.e. the position of the current element is not in the first row of the sub-block; $CM(l,m) = 0$, i.e. the position is not in the last column of a sub-block; and $P(l,m) = 0$, i.e. sum of row and column indices is even; the position of the next element is situated in the upper-right direction, i.e. the row index l must be decremented and the column index m must be incremented; see combinations 0, 4, 8, and 12. However, if $Rl(l,m) = 1$, i.e. the position in is the first row of the sub-block and the row index l cannot be decremented; $CM(l,m) = 0$; and $P(l,m) = 0$; the position of the next element is situated in the right direction, i.e. the row index l must remain unchanged and the column index m must be incremented; see combinations 16, 20, 24, and 28.

If $RL(l,m) = 0$, i.e. the position of the current element is not in the last row of the sub-block; $Cl(l,m) = 0$, i.e. the position is not in the first column of a sub-block; and $P(l,m) = 1$, i.e. sum of row and column indices is odd; the position of the next element is situated in the lower-left direction, i.e. the row index l must be incremented and the column index m must be decremented; see combinations 1, 3, 17, and 19. However, if $RL(l,m) = 0$; $Cl(l,m) = 1$, i.e. the position is in the first column of a sub-block and the column index m cannot be decremented; and $P(l,m) = 1$; the position of the next

element is situated in the lower direction, i.e. the row index l must be incremented and the column index m must remain unchanged; see combinations 5, 7, 21, and 23.

If $RL(l,m)=0$, i.e. the position of the current element is not in the last row of the sub-block; $CM(l,m)=1$, i.e. the position is in the last column of a sub-block; and $P(l,m)=0$, i.e. sum of row and column indices is even; the position of the next element is situated in the lower direction, i.e. the row index l must be incremented and the column index m must remain unchanged; see combinations 2, 6, 18, and 22.

If $RL(l,m)=1$, i.e. the position of the current element is in the last row of the sub-block; $CM(l,m)=0$, i.e. the position is not in the last column of a sub-block; and $P(l,m)=1$, i.e. sum of row and column indices is odd; the position of the next element is situated in the right direction, i.e. the row index l must remain unchanged and the column index m must be incremented; see combinations 9, 13, 25, and 29.

A reduced truth table uses don't cares to represent compactly combinations that are unaffected by certain parameters; see table 4.2.

Entry	$RI(l,m)$	$RL(l,m)$	$CI(l,m)$	$CM(l,m)$	$P(l,m)$	l^+	m^+
0	X	1	X	1	X	l	m
1	0	X	X	0	0	$l--$	$m++$
2	1	X	X	0	0	l	$m++$
3	X	0	0	X	1	$l++$	$m--$
4	X	0	1	X	1	$l++$	m
5	X	0	X	1	0	$l++$	m
6	X	1	X	0	1	l	$m++$

X denotes don't care

Table 4.2 Reduced Truth Table for Changes in Row and Column Indices

4.4.5 Boolean Expressions

From the reduced truth table, given in table 4.2, Boolean expressions can be derived for combined changes in the row and column indices by logically ORing table entries that have the same effects on the row index l and the column index m respectively. Note that the coordinates of the current element, generally indicated by (l,m), are omitted for clarity. The following expressions, given in sum-of-products form, determine the changes in the row and column indices:

No move is defined by

$$\left.\begin{aligned} l^+ &= l \\ m^+ &= m \end{aligned}\right\} \; if\ (RL \cdot CM)\ is\ true \tag{4.12}$$

where l and l^+ are the row indices of the current element and the next element respectively, m and m^+ are the column indices of the current element and the next element respectively.

A move in the upper-right direction is defined by

$$\left.\begin{aligned} l^+ &= l-- \\ m^+ &= m++ \end{aligned}\right\} \; if\ \left(\overline{R1} \cdot \overline{CM} \cdot \overline{P}\right) is\ true \tag{4.13}$$

where $l--$ refers to a decrement of the current row index, and $m++$ refers to an increment of the current column index.

A move in the right direction is defined by

$$\left.\begin{aligned} l^+ &= l \\ m^+ &= m++ \end{aligned}\right\} \; if\ \left(\left(R1 \cdot \overline{CM} \cdot \overline{P}\right) + \left(RL \cdot \overline{CM} \cdot P\right)\right) is\ true \tag{4.14}$$

A move in the lower direction is defined by

$$\left.\begin{array}{l} l^{+} = l{+}{+} \\ m^{+} = m \end{array}\right\} \; if \left(\left(\overline{RL} \cdot C1 \cdot P\right) + \left(\overline{RL} \cdot CM \cdot \overline{P}\right)\right) is\ true \tag{4.15}$$

where $l{+}{+}$ refers to an increment of the current row index.

A move in the lower-left direction is defined by

$$\left.\begin{array}{l} l^{+} = l{+}{+} \\ m^{+} = m{-}{-} \end{array}\right\} \; if \left(\overline{RL} \cdot \overline{C1} \cdot P\right) is\ true \tag{4.16}$$

where $m{-}{-}$ refers to a decrement of the current column index.

However, Boolean expressions can also be derived from the reduced truth table for independent changes in the row and column indices by logically ORing table entries that have the same effect on the row index l or the column index m respectively. The following expressions, given in sum-of-products form, determine the changes in the row and column indices independently:

A decrement of the row index l is defined by

$$l^{+} = l{-}{-} \quad if \left(\overline{R1} \cdot \overline{CM} \cdot \overline{P}\right) is\ true \tag{4.17}$$

An increment of the row index l is defined by

$$l^{+} = l{+}{+} \quad if \left(\left(\overline{RL} \cdot \overline{C1} \cdot P\right) + \left(\overline{RL} \cdot C1 \cdot P\right) + \left(\overline{RL} \cdot CM \cdot \overline{P}\right)\right) is\ true \tag{4.18}$$

A decrement of the column index m is defined by

$$m^{+} = m{-}{-} \quad if \left(\overline{RL} \cdot \overline{C1} \cdot P\right) is\ true \tag{4.19}$$

An increment of the column index m is defined by

$$m^{+} = m{+}{+} \quad if \left(\left(\overline{R1} \cdot \overline{CM} \cdot \overline{P}\right) + \left(R1 \cdot \overline{CM} \cdot \overline{P}\right) + \left(RL \cdot \overline{CM} \cdot P\right)\right) is\ true \qquad (4.20)$$

The above expressions, given in sum-of-products form, may be reduced and rearranged as required.

4.4.6 The Binary Decision Tree

The construction of the binary decision tree is related to the reduced truth table shown in table 4.2. The parity column, representing the parity parameter $P(l,m)$, contains one don't care; therefore the parity parameter $P(l,m)$ provides more information than the other parameters, whose columns contain more than one don't care. The truth table depicted in table 4.3 removes this don't care by expanding entry 0 of the reduced truth table with respect to the parity parameter. Thus the parity column contains two separable groups: a group of four entries with $P(l,m) = 0$, and a group of four entries with $P(l,m) = 1$.

Entry	$R1(l,m)$	$RL(l,m)$	$C1(l,m)$	$CM(l,m)$	$P(l,m)$	l^{+}	m^{+}
0	X	1	X	1	0	l	m
1	X	1	X	1	1	l	m
2	0	X	X	0	0	$l--$	$m{+}{+}$
3	1	X	X	0	0	l	$m{+}{+}$
4	X	0	0	X	1	$l{+}{+}$	$m--$
5	X	0	1	X	1	$l{+}{+}$	m
6	X	0	X	1	0	$l{+}{+}$	m
7	X	1	X	0	1	l	$m{+}{+}$

X denotes don't care

Table 4.3 Truth Table for Construction of Binary Decision Tree

With reference to table 4.3, for the group with $P(l,m) = 0$; consisting of entries 0, 2, 3, and 6; the last-column column, representing the last-column parameter $CM(l,m)$, does

not contain don't cares. Thus it contains two separable groups: a group of two entries with $CM(l,m) = 0$, and a group of two entries with $CM(l,m) = 1$.

For the group with $P(l,m) = 0$ and $CM(l,m) = 0$, consisting of entries 2 and 3, the first-row column, representing the first-row parameter $R1(l,m)$, does not contain don't cares. Thus it separates the changes in the row and column indices. For $R1(l,m) = 0$ the row index l is decremented and the column index m is incremented; see entry 2. For $R1(l,m) = 1$ the row index l remains unchanged and the column index m is incremented; see entry 3.

For the group with $P(l,m) = 0$ and $CM(l,m) = 1$, consisting of entries 0 and 6, the last-row column, representing the last-row parameter $RL(l,m)$, does not contain don't cares. Thus it separates the changes in the row and column indices. For $RL(l,m) = 0$ the row index l is incremented and the column index m remains unchanged; see entry 6. For $RL(l,m) = 1$ both indices remain unchanged; see entry 0.

The group with $P(l,m) = 1$; consisting of entries 1, 4, 5, and 7; can be separated similarly. Hence the required changes in the row and column indices can be determined based on a sequence of three binary tests.

The binary decision tree is shown in figure 4.16. The root node represents the parity parameter $P(l,m)$, i.e. the test for the sum of row index l and column index m being odd. Note that, following convention, left children are identified by 0, and right children are identified by 1. The two children of the root node correspond for $P(l,m) = 0$ to the last-column parameter $CM(l,m)$ and for $P(l,m) = 1$ to the last-row

parameter $RL(l,m)$. The two children of the node corresponding to the last-column parameter $CM(l,m)$ represent the two row parameters $Rl(l,m)$ and $RL(l,m)$ respectively, and the two children of the node corresponding to the last-row parameter $RL(l,m)$ represent the two column parameters $Cl(l,m)$ and $CM(l,m)$ respectively. On the last level, eight external nodes refer to the changes in the row and column indices. Note that three of the changes in the row and column indices appear twice within the eight external nodes since there are only four directions of movement, as shown in figure 4.15, and no movement at the last position of a sub-block.

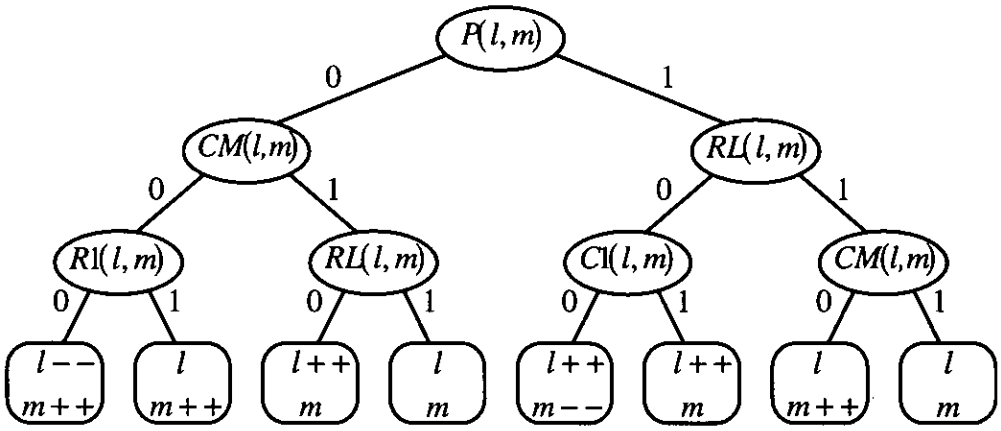


Figure 4.16 Decision Tree for Changes in Row and Column Indices

To obtain the changes in the row and column indices, and therefore the position of the next element, a sequence of three binary tests based on the position of the current element, indicated by (l,m), in the $L \times M$ sub-block is generated starting from the root node. The first test always evaluates the parity parameter $P(l,m)$; and depending on the result of this test either the last-column parameter $CM(l,m)$ for $P(l,m)=0$, or the last-row parameter $RL(l,m)$ for $P(l,m)=1$ is tested. The third test is conducted in a similar manner, and finally determines the changes in the row and column indices. The

binary decision tree generates a valid test sequence for the position of any element in any $L \times M$ sub-block. Appendix F contains a worked example.

4.5 Hardware Implementation of Zigzag-reordering Algorithm

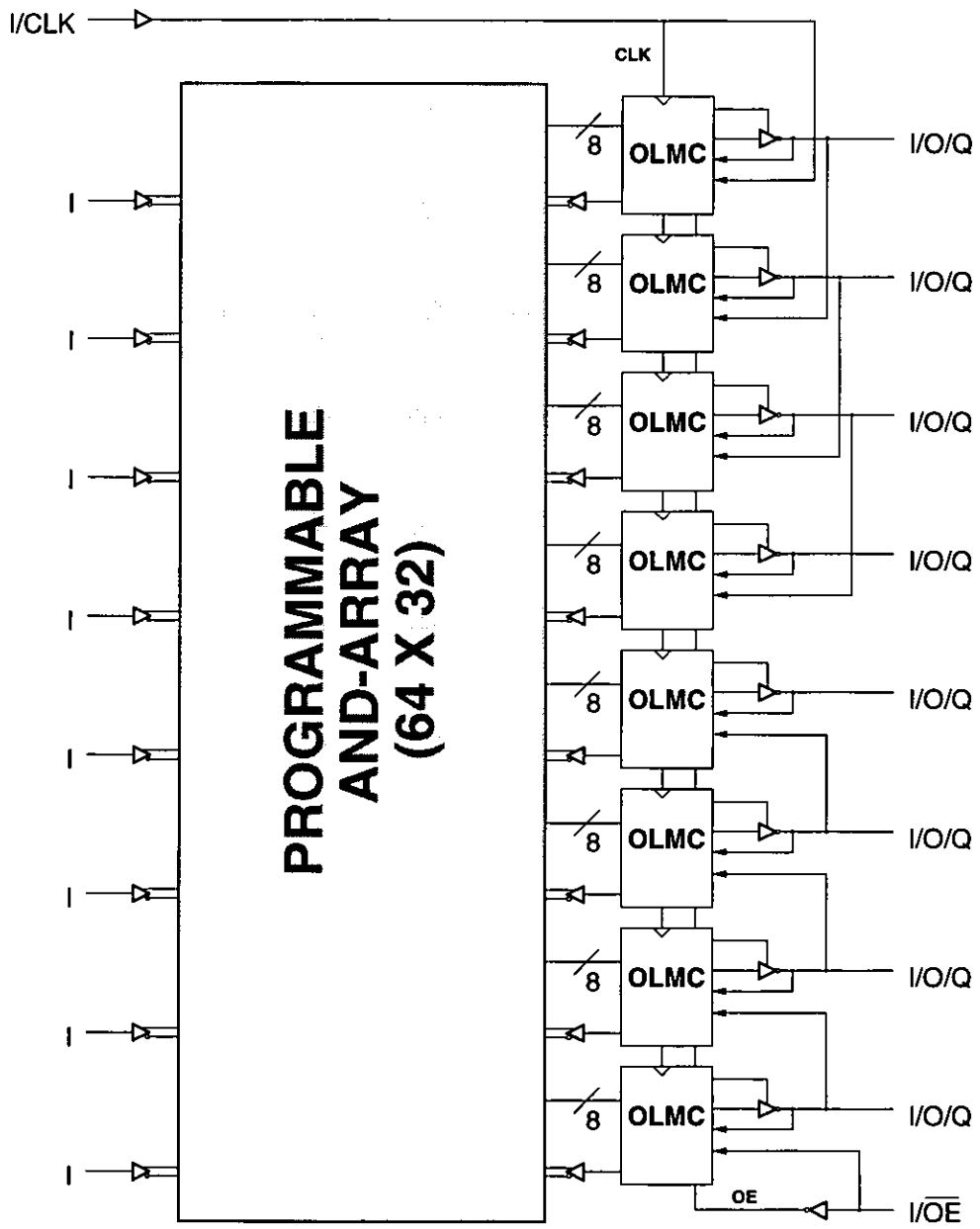
4.5.1 Motivation for Hardware Implementation of Zigzag-reordering Algorithm

JPEG aimed to achieve cost effective and computationally efficient implementations for software and hardware. Therefore, it was intended to keep the system simple enough to permit single-chip implementations (W. B. Pennebaker and J. L. Mitchell 1992, p. 305). However, the hardware implementation of the zigzag-reordering algorithm described in this section is based on two programmable logic devices (PLDs) and aims to demonstrate the feasibility of the approach. Note that a PLD is an array of basic logic element, i.e. gates, interconnected by programmable links; such as fuses for one-time programmable PLDs, or floating gates for erasable PLDs. The implementation constitutes a Moore state machine with binary inputs representing the dimensions of the sub-block to be reordered. It involves two stages, each of which is mapped into a separate GAL16V8 device; see (Lattice 1996 and 1997).

4.5.2 The GAL16V8 Device

The GAL16V8 device is an electrically erasable 20-pin generic array logic PLD with a user-programmable 64×32 AND array, a fixed 8×8 OR array, and an output stage employing output logic macro-cells (OLMCs) with eight product lines, i.e. AND gate outputs, connected to each OLMC. Figure 4.17 depicts the functional block diagram of the GAL16V8. The device has eight dedicated inputs and eight user-configurable pins;

each of which may be configured individually as input, combinational output, or registered output within the appropriate OLMC; see (Lattice 1996 and 1997). Registered outputs are also fed back into the AND array of the device enabling a state machine to be implemented on a single device.



Reproduced by Special Permission of Lattice Semiconductor.
© 1996 by Lattice Semiconductor.

Figure 4.17 Functional Block Diagram of GAL16V8 Device

4.5.3 The Tango-PLD Development Tool

Tango-PLD is a universal development tool for designing and simulating logic systems for PLDs. It consists of a language preprocessor, a design compiler, a logic minimizer, a functional simulator, and a fusemap generator. It provides a C-like hardware description language, Tango Design Language (TDL); and produces industry-standard Joint Electronic Device Engineering Council (JEDEC) fusemap files for programming PLDs. The functional simulator can verify a design before it is committed to hardware. Tango-PLD supports a variety of device architectures including the GAL16V8 device family; see (ACCEL 1989a and b).

4.5.4 The Moore State Machine for Versatile Zigzag-reordering Algorithm

The hardware implementation constitutes a Moore state machine consisting of two stages. Each stage is mapped into a separate GAL16V8 device, and the state machine is implemented by interconnecting the two devices as shown in figure 4.18.

The state machine has six binary inputs representing the dimensions, i.e. number of rows L and number of columns M, of the sub-block to be reordered. While the versatile zigzag-reordering algorithm operates on sub-blocks of any dimensions, this particular implementation operates on sub-blocks with up to eight rows and up to eight columns. Thus it allows all 64 sub-block dimensions from 1×1 to 8×8 to be generated. However, the implementation is compatible with the JPEG standard, that partitions image components into 8×8 blocks for the DCT-based method. Note that, due to the 3-bit representation of the number of rows, the binary pattern 000 indicates a sub-block containing one row, 001 indicates a sub-block containing two rows, etc. up to

111 indicating a sub-block containing eight rows. The number of columns in a sub-block is represented similarly.

A reset signal, labelled *RESET*, is used to initialize the row and column indices, l and m, to the binary pattern 000 corresponding to the position of the first element in the sequence regardless of the sub-block dimensions, L and M. The generation of the appropriate zigzag scan sequence is synchronized to a clock signal, labelled *CLK*. Since stage A is purely combinational, both signals are applied only to stage B.

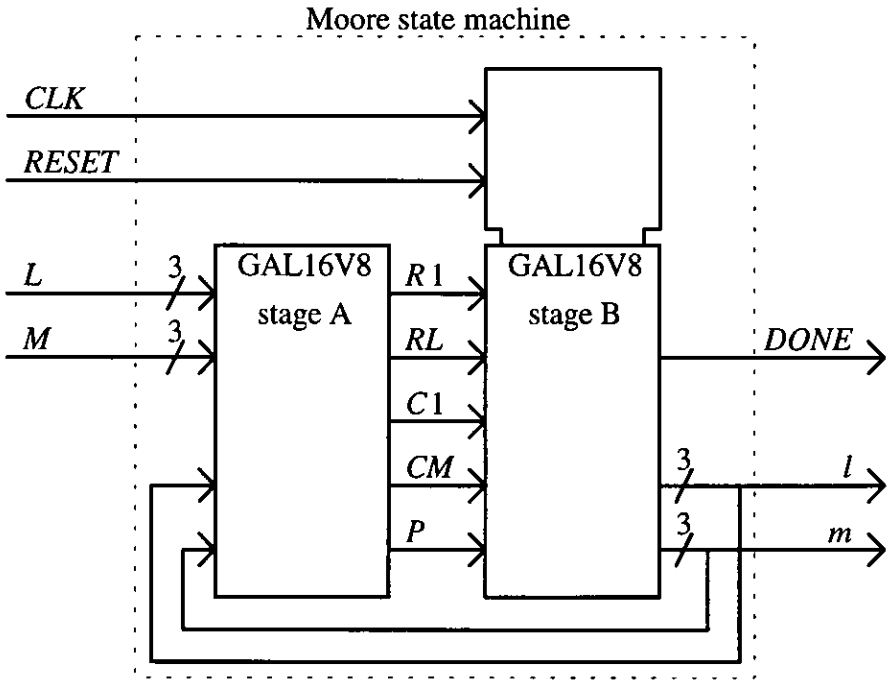


Figure 4.18 Block Diagram of Moore State Machine for Versatile Zigzag-reordering Algorithm

The state machine has six binary outputs that represent the row index l and the column index m of the position of the current element in the scan path as described for the six binary inputs.

A signal, labelled *DONE*, is asserted to indicate completion of the zigzag scan sequence of the current sub-block; the row and column indices, l and m, are initialized to 000 in readiness for the zigzag scan sequence of the next sub-block.

Stage A determines, according to equations 4.2 to 4.6, the five binary parameters $R1(l,m)$, $RL(l,m)$, $C1(l,m)$, $CM(l,m)$, and $P(l,m)$ from the current values of the row and column indices, l and m, and the current sub-block dimensions, L and M. This stage is purely combinational and has twelve binary inputs that are processed as four groups with three bits each, and five outputs that represent the five binary parameters. The parity P is evaluated by XORing the least-significant bits of the row and column indices. Note that the parity P is the same for both, row and column, indices starting from either zero or one. In the combinational output configuration, one of the eight product lines is used to control the tri-state input of the OLMC. Stage A utilizes 21 out of 64 product lines, i.e. 33 %; and six of the maximum seven product lines per output for two output signals.

Stage B determines the next row and column indices from the current indices and the five binary parameters using the clock signal to control the timing of the zigzag-scan-sequence generation, and the reset signal to initialize the row and column indices to 000 for the first scan. Note that the implementation of the increments and decrements is described in subsection 4.5.5. The stage has two 3-bit outputs that represent the row index l and the column index m. The outputs are implemented as registered outputs enabling them to be fed back internally to the AND array of the device. The stage also generates the *DONE* signal. In the registered output configuration, all of the eight

product lines are available. Stage B utilizes 42 out of 64 product lines, i.e. 66 %; and all of the maximum eight product lines per output for two output signals.

The TDL files, describing each stage individually and the entire state machine, are contained in appendix G. The files also include extracts from the full sets of test vectors. Each device has been individually simulated to verify its correct operation, and the entire state machine has also been simulated to ensure that all zigzag scan paths are correctly generated.

4.5.5 Implementation of Increments and Decrements

Boolean expressions denoting arithmetic increments and decrements do not fit within the GAL16V8 device. However; since, in practice, a row index is never decremented from 000 or incremented from 111, don't-care states can be used for these states in order to reduce the number of product lines per output.

Table 4.4 depicts the state table for the binary increments of the row index l. Boolean expressions can be derived for each bit; and don't cares can be assumed to be either zero or one.

l	000	001	010	011	100	101	110	111
l^+	001	010	011	100	101	110	111	*XXX*

X denotes don't care

Table 4.4 Binary Increments

The least-significant bit, denoted by $l0$ and $l0^+$ respectively, toggles between zero and one:

$$l0^+ = \overline{l0} \tag{4.21}$$

with the appropriate don't care assumed to be zero.

The next bit, denoted by $l1$ and $l1^+$ respectively, is defined by XORing $l0$ and $l1$:

$$l1^+ = \left(\left(\overline{l0} \cdot l1\right) + \left(l0 \cdot \overline{l1}\right)\right) \tag{4.22}$$

with the appropriate don't care assumed to be zero.

The most-significant bit, denoted by $l2$ and $l2^+$ respectively, is defined by:

$$l2^+ = \left(\left(l0 \cdot l1\right) + l2\right) \tag{4.23}$$

with the appropriate don't care assumed to be one.

Table 4.5 depicts the state table for the binary decrements of the row index l. Similarly, Boolean expressions can be derived for each bit; and don't cares can be assumed to be either zero or one.

l	000	001	010	011	100	101	110	111
l^+	*XXX*	000	001	010	011	100	101	110

X denotes don't care

Table 4.5 Binary Decrements

Again, the least-significant bit toggles between zero and one:

$$l0^+ = \overline{l0} \tag{4.24}$$

with the appropriate don't care assumed to be one.

The next bit is defined by XNORing $l0$ and $l1$:

$$l1^+ = \left(\left(l0 \cdot l1\right) + \left(\overline{l0} \cdot \overline{l1}\right)\right) \tag{4.25}$$

with the appropriate don't care assumed to be one.

The most-significant bit is defined by:

$$l2^{+} = \left((l0 \cdot l2) + (l1 \cdot l2)\right) \tag{4.26}$$

with the appropriate don't care assumed to be zero.

The binary increments and decrements are applied to the bits of the column index similarly. Using these tailored Boolean expressions for increments and decrements enables stage B to be implemented on a single GAL16V8 device.

4.6 Coding of Sub-block Dimensions

4.6.1 Motivation for Coding of Sub-block Dimensions

Adaptive zigzag reordering reduces the entropy of the runs of zero coefficients by traversing a scan path that is tailored to the dimensions of a rectangular sub-block in a particular block of quantized transform coefficients. Since the sub-blocks generally have different dimensions depending on the specific content of the corresponding block, the dimensions of the sub-block need to be retained in order to traverse the zigzag scan path correctly during decoding. Therefore the sub-block dimensions themselves need to be efficiently coded.

4.6.2 The Sub-block Dimensions

For an image-compression scheme operating on $L_{max} \times M_{max}$ blocks, $L_{max} M_{max}$ symbols are required to identify directly the $L_{max} M_{max}$ possible sub-block dimensions. Assuming the worst case, i.e. that all symbols are equally probable, the maximum entropy H_{max} can be obtained using equation 2.7:

$$H_{max} = -\sum_{j=1}^{L_{max}M_{max}} \frac{1}{L_{max}M_{max}} \log_2 \frac{1}{L_{max}M_{max}} \text{ bits} = \log_2 (L_{max}M_{max}) \text{ bits} \quad (4.27)$$

For the DCT-based method within the JPEG standard operating on 8×8 blocks, the maximum entropy of the sub-block dimensions is therefore:

$$H_{max} = \log_2 8^2 \text{ bits} = 6 \text{ bits} \quad (4.28)$$

It has been found that, although the sub-block dimensions are not evenly distributed in practice, entropy coding, such as Huffman or arithmetic coding, of the sub-block dimensions themselves is not sufficiently efficient to produce an overall reduction in bit rate. It has also been found that coding of the sub-block dimensions with reference to the dimensions of the preceding sub-block, that tends to have similar complexity, does not significantly improve efficiency.

However, the dimensions of a sub-block are correlated with the number of coefficients within the sub-block, thus allowing more efficient coding.

4.6.3 Sub-block Dimensions and Scan-path Length

In the JPEG standard the EOB symbol is used to terminate a vector, i.e. zigzag-reordered block, of quantized DCT coefficients after the last nonzero coefficient. Therefore the number of positions along a zigzag scan path of a sub-block, i.e. the scan-path length, is known and can be evaluated; it varies between 1 and $L_{max}M_{max}$, i.e. 64 for 8×8 blocks as defined by the JPEG standard for the DCT-based method. For any particular $L \times M$ sub-block, the minimum scan-path length depends on L and M; however, the maximum scan-path length is LM as longer scan paths require larger sub-blocks. Usually, the scan-path length does not uniquely identify the sub-block dimensions; however, it restricts the number of sub-block dimensions that are suitable

to contain a particular number of positions. Figure 4.19 depicts all four possible sub-block dimensions for the scan-path length of five. The last nonzero coefficient in each scan path is indicated by a black dot. In the 5×1 and 1×5 sub-blocks, shown in figure 4.19 a) and d) respectively, the last nonzero coefficient is at the fifth position; a different scan-path length leads to different sub-block dimensions. The 3×2 sub-block, depicted in figure 4.19 b) accommodates scan-path lengths of four, five, or six. However, the 2×3 sub-block, shown in figure 4.19 c) accommodates scan-path lengths of five or six; note that a 2×2 sub-block suffices for the scan-path length of four.

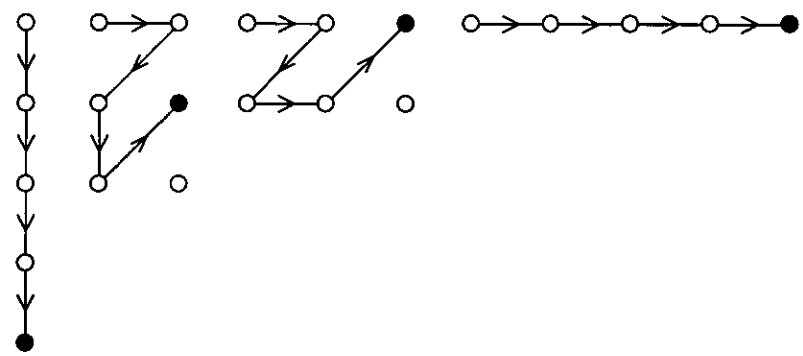


Figure 4.19 Scan-path Length of 5 for (a) 5×1, (b) 3×2, (c) 2×3, and (d) 1×5 Sub-blocks

Figure 4.20 depicts two of nine possible sub-block dimensions for a scan-path length of 14; the seven remaining sub-block dimensions are 7×2, 5×3, 4×4, 5×4, 3×6, 2×7, and 2×8.

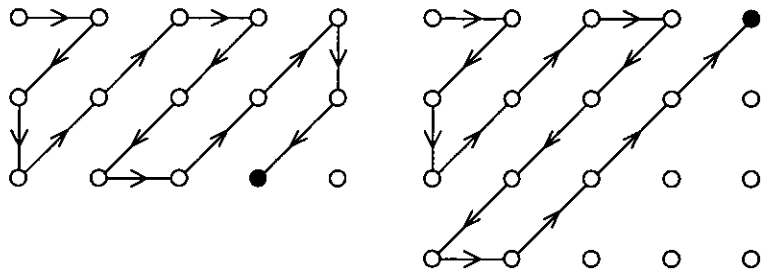


Figure 4.20 Scan-path Length of 14 for (a) 3×5, and (b) 4×5 Sub-blocks

Table 4.6 combines the scan-path lengths in the range [1,2,...,64] with the sub-block dimensions from 1×1 to 8×8, thus covering the 8×8 blocks defined by the JPEG standard for the DCT-based method. Note that Length refers to the scan-path length, L is the number of sub-block rows, M is the number of sub-block columns, and Number refers to the number of sub-block dimensions that can accommodate a particular scan-path length. Table 4.6 (1) contains 1×1 to 8×4 sub-blocks, that have scan-path lengths in the range [1,2,...,32]. Table 4.6 (2) contains scan-path lengths in the range [1,2,...,32] of 1×5 to 8×8 sub-blocks, and table 4.6 (3) contains scan-path lengths in the range [33,34,...,64] of 1×5 to 8×8 sub-blocks. The number of possible sub-block dimensions increases with the scan-path length, reaches its maximum value of 14 for scan-path lengths of 28 and 30, and decreases afterwards. Note that the maximum number of symbols to uniquely identify the dimensions of a sub-block with a given scan-path length is 14.

Length	*L* 1 1	2 1	3 1	4 1	5 1	6 1	7 1	8 1	1 2	2 2	3 2	4 2	5 2	6 2	7 2	8 2	1 3	2 3	3 3	4 3	5 3	6 3	7 3	8 3	1 4	2 4	3 4	4 4	5 4	6 4	7 4	8 4	
	M 1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	
1	X																																
2		X							X																								
3			X							X							X																
4				X						X	X														X								
5					X						X							X															
6						X					X							X	X							X							
7							X					X							X							X	X						
8								X				X	X						X							X	X						
9													X						X	X							X						
10													X							X	X						X	X					
11														X						X	X						X	X	X				
12														X	X					X	X						X	X	X				
13															X						X							X	X				
14															X						X							X	X				
15																X					X	X						X	X				
16																X						X	X					X	X				
17																						X	X						X				
18																						X	X						X	X			
19																							X						X	X	X		
20																							X						X	X	X		
21																							X	X						X	X		
22																								X						X	X		
23																								X						X	X		
24																								X						X	X		
25																															X		
26																															X	X	
27																															X	X	
28																															X	X	
29																																X	
30																																X	
31																																X	
32																																X	

Table 4.6 (1 of 3) Scan-path Lengths and Sub-block Dimensions

Length	*L* 1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	Number
	M 5	5	5	5	5	5	5	5	6	6	6	6	6	6	6	6	7	7	7	7	7	7	7	7	8	8	8	8	8	8	8	8	
1																																	1
2																																	2
3																																	3
4																																	4
5	X																																4
6									X																								6
7																	X																6
8																									X								7
9		X																															5
10		X								X																							7
11										X																							7
12			X							X																							9
13			X								X							X															7
14			X	X							X							X								X							9
15			X	X	X						X	X														X							11
16				X	X						X	X	X													X							11
17				X	X						X	X	X																				8
18				X	X						X	X	X						X														10
19				X	X							X	X						X								X						10
20				X	X	X						X	X						X								X						11
21					X	X	X					X	X	X					X								X						12
22					X	X	X					X	X	X	X			X									X						12
23					X	X	X					X	X	X	X			X								X	X						13
24					X	X	X					X	X	X	X			X								X	X						13
25					X	X	X						X	X	X			X	X								X						10
26						X	X						X	X	X			X	X								X	X					11
27						X	X						X	X	X			X	X	X							X	X					12
28						X	X						X	X	X			X	X	X	X						X	X	X				14
29						X	X						X	X	X				X	X	X						X	X	X	X			13
30						X	X	X					X	X	X				X	X	X						X	X	X	X			14
31							X	X						X	X				X	X	X						X	X	X	X			12
32							X	X						X	X				X	X	X						X	X	X	X			12

Table 4.6 (2 of 3) Scan-path Lengths and Sub-block Dimensions

Length	*L* / *M*	1 5	2 5	3 5	4 5	5 5	6 5	7 5	8 5	1 6	2 6	3 6	4 6	5 6	6 6	7 6	8 6	1 7	2 7	3 7	4 7	5 7	6 7	7 7	8 7	1 8	2 8	3 8	4 8	5 8	6 8	7 8	8 8	Number
33								X	X						X	X	X					X	X	X						X	X	X		11
34								X	X						X	X	X					X	X	X						X	X	X		11
35								X	X						X	X	X					X	X	X	X					X	X	X		12
36									X						X	X	X						X	X	X						X	X	X	X	11
37									X							X	X						X	X	X						X	X	X	X	10
38									X							X	X						X	X	X						X	X	X	X	10
39									X							X	X						X	X	X						X	X	X	X	10
40									X							X	X						X	X	X						X	X	X	X	10
41																X	X						X	X	X							X	X	X	8
42																X	X						X	X	X							X	X	X	8
43																	X							X	X							X	X	X	6
44																	X							X	X							X	X	X	6
45																	X							X	X							X	X	X	6
46																	X							X	X							X	X	X	6
47																	X							X	X							X	X	X	6
48																	X							X	X							X	X	X	6
49																								X	X								X	X	4
50																									X								X	X	3
51																									X								X	X	3
52																									X								X	X	3
53																									X								X	X	3
54																									X								X	X	3
55																									X								X	X	3
56																									X								X	X	3
57																																		X	1
58																																		X	1
59																																		X	1
60																																		X	1
61																																		X	1
62																																		X	1
63																																		X	1
64																																		X	1

Table 4.6 (3 of 3) Scan-path Lengths and Sub-block Dimensions

4.6.4 Entropy Coding of Sub-block Dimensions

Assuming the worst case, i.e. that all sub-blocks contain only scan-path lengths of 28 or 30, the maximum entropy of the symbols for the sub-block dimensions is:

$$H_{max} = \log_2 14 \text{ bits} = 3.8 \text{ bits} \tag{4.29}$$

Entropy coding, for example Huffman coding, can assign codewords according to the probability distribution of the sub-block dimensions for each scan-path length independently. However, the same codeword can be used with different scan-path lengths, so that the most-probable symbol, i.e. sub-block, within every scan-path length is coded with the same codeword. The scan-path length, that is known, and an additional symbol therefore identify the dimensions of a sub-block.

Since the number of possible sub-block dimensions is one for scan-path lengths of 1 and 57 to 64, these scan-path lengths uniquely identify sub-block dimensions 1×1 and 8×8 respectively; see table 4.6. Hence, for identification of the corresponding sub-block dimensions no additional symbol needs to be generated, stored, or transmitted. Symbols can be represented as a stream that is only accessed when necessary, i.e. when the scan-path length does not uniquely identify the sub-block dimensions.

It has been found that adaptive zigzag reordering as described in section 4.3 combined with coding of sub-block dimensions as described in this section produces a lower bit rate than standard JPEG.

The sub-block dimensions need to be retained in order to traverse the zigzag scan path correctly during decoding. The correlation between sub-block dimensions and scan-path length is investigated. Coding of the sub-block dimensions that takes the scan-path length into account is developed, and further improvements are suggested.

The chapter addresses issues that affect adaptive zigzag reordering of transform coefficients in various respects.

Chapter 5

Artificial Neural Networks

5.1 Introduction

This chapter introduces the notion of artificial neural networks (ANNs). The subject has attracted much attention, and research has generated a large body of knowledge, therefore this chapter concentrates on feedforward ANNs and the error-backpropagation algorithm, that are used in the image-compression scheme described in chapter 6.

Section 5.2 briefly describes biological neural networks, summarizes the historical foundation of ANNs, outlines properties and realizations of ANNs, and enumerates some areas of application.

Section 5.3 describes a single artificial neuron; develops propagation, activation, and output functions; and introduces a simple notation.

Section 5.4 focuses on feedforward ANNs, describes forward propagation and learning, introduces the error-backpropagation algorithm and other learning rules, and explains multilayer feedforward ANNs.

Section 5.5 briefly outlines the application of ANNs to digital image compression. Finally section 5.6 concludes the chapter with a brief summary.

5.2 Introduction to Artificial Neural Networks

5.2.1 Biological Neural Networks

The human brain is the most complicated and fascinating structure. It contains about 100×10^9 neurons interconnected via more than 100×10^{12} links (A. Zell 1994, chapter 2). Each neuron is a complex biochemical processing unit. Similar to any biological cell, the cell membrane and the contained cell body build the nerve cell that is

between 5 μm and 100 μm in size (M. Kunt et al. 1985). A main fibre called axon and a number of fibre branches called dendrites are attached to the nerve cell. Figure 5.1 depicts a simplified neuron with the dendrites, that work as inputs to the nerve cell, shown on the left; and the axon, that works as output from the nerve cell, shown on the right. The junction between the axon of one neuron and the dendrite of another neuron is called a synapse. An individual neuron can receive signals from thousands of presynaptic neurons, and can transmit to thousands of postsynaptic neurons; it can handle up to 200000 synapses. The information transfer from the presynaptic neuron to the postsynaptic neuron is made electrochemically.

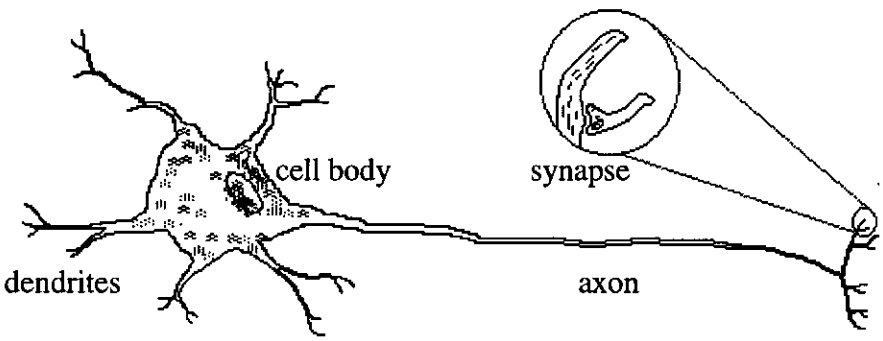


Figure 5.1 Simplified Nerve Cell

While stimulation via excitatory synapses increases the electrical potential of the cell membrane, stimulation via inhibitory synapses decreases the potential. Once a certain threshold is exceeded, the neuron fires: its stimulating signal, consisting of pulse trains, propagates via axon, synapses, and dendrites to the postsynaptic neurons. Each pulse has a magnitude of about 100 mV and a duration of about 1 ms. The repetition rate of these pulses is proportional to the intensity of a stimulus. Thus the nerve cells communicate through frequency modulation (FM). Synaptic connections change with time; they can increase, decrease, or even disappear. Axons can build new connections

and attach to neurons that were previously unconnected. This process is described as learning. Further reading includes a brief description of nerve cells (M. Kunt et al. 1985) and a comprehensive introduction to biological neurons (E. R. Kandel et al. 1991). A. Zell (1994) produced a comprehensive introduction to neural networks including biological foundations, network architectures, network simulation, and applications. M. T. Hagan et al. (1996) focused on the design of neural networks.

ANNs are computational models that mimic their biological counterparts. Note that the concept of artificial neural networks can be applied to software simulations and hardware implementations; see subsection 5.2.4. Similarly to the human brain ANNs consist of a number of simple processing units, i. e. neurons, and a number of interconnections, i. e. weights. Thus, two key features distinguish artificial neural networks from conventional computational systems:

- Artificial neural networks are naturally massively parallel; and
- Artificial neural networks are adaptive, i.e. trainable.

Exact modelling of biological neural networks is not yet possible; and is, for technical applications, often neither necessary nor desirable. For most artificial neurons the amplitude of the output signal is proportional to the intensity of a stimulus. Thus they communicate through amplitude modulation (AM). The learning ability of an ANN is based on changing the ANN itself by exploiting the following approaches individually or in combination:

- Building new connections,
- Removing existing connections,
- Changing weights of connections,
- Changing thresholds of neurons,

- Changing the functions of neurons,
- Inserting new neurons, and
- Removing existing neurons.

Changing the weights of connections is the most prominent approach, and accomplishes building and removing connections as well. However, changing the functions that define a neuron does not seem to correspond with biological nerve cells.

The learning strategy describes the degree of supervision during the learning period:
In supervised learning a 'teacher' provides the desired output pattern with each input pattern. The aim is to repeatedly change the trainable weights, so that the network can generate an approximation of the desired output for a known or new, but similar, input pattern. It is the fastest learning strategy, but does not correspond with learning in biological neural networks.

In reinforcement learning the network produces from each input pattern an output pattern that is then rated by a 'teacher'. The aim is to analyse these additional hints; for example correct and incorrect, or degree of correctness; and to repeatedly change the trainable weights, so that the network itself finds the correct output pattern for a given input pattern. This strategy is slower than supervised learning because of the limited information, but corresponds much better with learning in biological neural networks.

In unsupervised learning, also known as self-organised learning, the network receives only the input pattern and organizes similar input patterns into similar classes by activating the same or adjacent neurons. This strategy extracts statistical features from the input pattern, and meets learning in biological neural networks best, but is unsuitable for some tasks.

5.2.2 Foundations of Artificial Neural Networks

Research on artificial neural networks was stimulated in the 1940s when W. S. McCulloch and W. Pitts (1943) published their work on networks of McCulloch-Pitts neurons. D. O. Hebb (1949) introduced with the Hebb rule a simple rule for supervised learning that has been intensively used. K. Lashley (1950) recognized that biological neural networks store knowledge distributively.

The first successful neurocomputer, Mark I Perceptron, was built by F. Rosenblatt (1958) and co-workers. It contained a 20×20-pixel sensor and 512 servomechanical potentiometers realizing variable weights; and could recognize simple symbols. F. Rosenblatt (1959) described variations of the perceptron and introduced the perceptron convergence theorem. B. Widrow and M. E. Hoff (1960) developed the adaptive linear element (Adaline). B. Widrow founded later the first neurocomputing company, Memitor Corporation. N. J. Nilson (1965) summarized this period.

However, the popularity of artificial neural networks decreased rapidly with growing understanding of the limitations of the known techniques. M. Minsky and S. Papert (1969) analysed some perceptrons, showed that these perceptrons were not suitable for many problems, assumed the failure of bigger models, and announced this field of research to be a dead end. Limited research continued, generating important contributions; see for example (T. Kohonen 1972; C. von der Malsburg 1973; P. J. Werbos 1974; S. Grossberg 1976 and 1980; J. L. McClelland and D. E. Rumelhart 1981; and J. J. Hopfield 1982).

New interest in artificial neural networks grew in the 1980s, and research was reinforced. J. J. Hopfield had a strong influence due to an important publication

(J. J. Hopfield and D. W. Tank 1985) and his personal involvement. The error-backpropagation algorithm, originally described by P. J. Werbos (1974), was popularized by D. E. Rumelhart et al. (1986a and b), and demonstrated fast and efficient learning. Nettalk, a project of T. J. Sejnowski and C. R. Rosenberg (1986), was a feedforward ANN using a self-supervised backpropagation algorithm that learnt to read written words aloud. From 1986 many researchers started their work in various new areas of research and application.

J. A. Anderson and E. Rosenfeld (1988) compiled important contributions for a comprehensive summary of the foundations of neural networks. Further reading includes (J. A. Anderson et al. 1990; D. E. Rumelhart et al. 1986); and J. L. McClelland et al. 1986).

R. P. Lippmann (1987) produced a widely acclaimed comprehensive review, describing six important neural-network models for application in pattern classification, that was selectively updated by D. R. Hush and B. G. Horne (1993). B. Widrow and M. A. Lehr (1990) reviewed feedforward ANNs; and S. I. Amari (1990) compiled mathematical foundations of neurocomputing.

5.2.3 Properties of Artificial Neural Networks

The distinct properties of artificial neural networks include:

- Learning ability: an ANN learns by example; it extracts information from the training data without need for rules or formulae resulting in less need to determine relevant factors a priori. The ANN can adapt more easily to new conditions, i.e. input data, than conventional algorithms.

- Distributed knowledge: an ANN stores knowledge distributively in the weights of its neurons. This architecture suits parallel processing.
- Parallelism: an ANN consists of a large number of interconnected simple processing units, i.e. neurons, operating in parallel. This structure is very suitable for parallel processing, for example on transputer systems. However, the design must limit the amount of communication in order to lead to a practical system. Very large-scale integration (VLSI) circuits form an additional class of hardware: neurochips.
- Fault tolerance: storing information distributively within an ANN enables better fault tolerance for component and connection defects, if the system is appropriately designed.
- Associative storage: while conventional computers use address-based storage of information, an ANN uses content-based storage resulting in better and faster performance for pattern-association tasks.
- Robustness: a correctly trained ANN is less sensitive to distortion and noise in the input data than conventional algorithms.
- Implemented representation: in an ANN information is incorporated in the program rather than stored in an independent database. The active representation of knowledge is shaped by adjusting parameters.
- Need for training: before retrieving any information, most ANNs must iteratively adjust their parameters by repeatedly applying sufficient and relevant training data to their inputs, and changing their variables according to a specified learning rule. These variables are often initialized with small 'random' numbers in order to avoid saturation. Because of the distributed representation of knowledge, it is very

difficult to preset some fundamental knowledge. Some ANNs are designed rather than trained.

- Hidden knowledge: an ANN extracts information from the training data and stores knowledge by adjusting its parameters. This internal representation is difficult to interpret, analyse, and verify.
- Time consumption for learning: powerful algorithms and new concepts speed up the training process, but this initial period remains very time consuming, especially for large and complex networks.

5.2.4 Realization of Artificial Neural Networks

The concept of artificial neural networks is now widely accepted and generates a variety of products.

Packages for software simulation of artificial neural networks are available for academic and commercial use; for example ANSim and ANSpec, Aspirin/MIGRAINES, BrainMaker, Cortex-Pro, FAST, Galatea, GENESIS, ICSIM, LVQ-PAK and SOM-PAK, MATLAB with Neural Network Toolbox, MONNET, MUME, Nestor Development System, NeuFuz 4, Neural Shell, NeuralWorks Professional II/Plus, Neuralyst, NeuroForecaster, NeuroGraph, NEURO-Compiler, NeuroSolutions v2.0, NEUROtools, SENN++, PDP simulators (J. L. McClelland and D. E. Rumelhart 1988), PlaNet, Pygmalion, Rochester Connectionist Simulator, SESAME, SNNS, UCLA-SFINX, VieNet2, and Xerion.

Hardware solutions include multiple-instruction multiple-data (MIMD) and single-instruction multiple-data (SIMD) parallel-processing systems, co-processor boards for workstations and personal computers, neurocomputers built from standard or special

components, digital and analogue neurocomputing VLSI circuits, and optical neurocomputing systems.

5.2.5 Applications of Artificial Neural Networks

In industry and research artificial neural networks have been successfully applied in very different applications including (H. B. Demuth and M. Beale 1994, pp. 1/8 and 1/9):

- Aerospace: aircraft autopilot, flight path simulation, aircraft control systems, aircraft component simulation, and aircraft component fault detection.
- Automotive: automobile automatic guidance system, and warranty activity analysis.
- Banking: cheque and document reading, and credit application evaluation.
- Electronics: code sequence prediction, integrated-circuit chip layout, process control, chip failure analysis, and nonlinear modelling.
- Medical: breast cancer cell analysis, electroencephalogram (EEG) and electrocardiogram (ECG) analysis, prosthesis design, optimization of transplant times, and hospital quality improvement. A. S. Miller et al. (1992) reviewed the applications of ANNs to medical imaging and signal processing.
- Robotics: trajectory control, forklift robot, manipulator controllers, and vision systems.
- Speech: speech recognition, speech compression, vowel classification, and text-to-speech synthesis.

5.3 Artificial Neuron

5.3.1 Structure of Artificial Neuron

An artificial neuron is a basic processing unit and the building block for ANNs. Its purpose is to generate an output value dependent on the input values and its previous activations. Figure 5.2 shows the general structure of an artificial neuron with R inputs. The neuron consists of weight vector w, that modifies the R-element input vector p; scalar bias b, that can be used as an offset; propagation function f_{pro}, that generates the net input n from input vector p, weight vector w and bias b; activation function f_{act}, that calculates the activation c of the neuron from the net input n and previous activations; and finally output function f_{out}, that determines the scalar output a of the neuron. Note that the weights in vector w and the bias b are adjustable parameters. A weight of zero removes the connection between the output of some neuron and the input of a neuron; and the output of a neuron can be fed back to its input for direct feedback. The propagation, activation, and output functions determine the characteristics of the neuron. The following subsections outline some of the available functions. N. Hoffmann (1993, chapter 2) produced a more detailed summary.

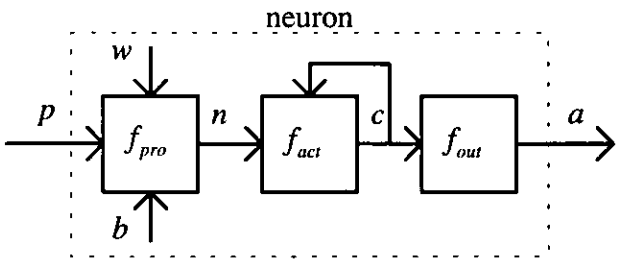


Figure 5.2 Structure of an Artificial Neuron

5.3.2 Propagation Function

The propagation function f_{pro} generates the net input n, a scalar, that represents the effective input to the neuron by evaluating the R-element input column vector p, the T-element weight row vector w, and scalar bias b

$$n = f_{pro}(p, w, b) \tag{5.1}$$

where $p = \begin{bmatrix} p(1) \\ p(2) \\ \vdots \\ p(R) \end{bmatrix}$ and $w = [w(1) \quad w(2) \quad \ldots \quad w(T)]$ with $R \leq T$.

Although many functions are suitable as a propagation function, most neurons use a sum of weighted inputs to generate the net input n as defined in equation 5.2.

$$n = \sum_{j=1}^{R} (w(j)\, p(j)) + b = w\, p + b \tag{5.2}$$

Each element of the input vector, $p(j)$, is multiplied by the corresponding element of the weight vector, $w(j)$; and the products are summed. This is the dot product of the row vector w and the column vector p. The scalar bias b is regarded as a weight element connected to a constant input of one. Higher-order neurons, for which $T > R$, have additional weights that scale the products of two or more input elements.

The propagation function of radial-basis neurons, for example, calculates the vector distance between weight vector w and input vector p that is multiplied by bias b

$$n = \sqrt{\sum_{j=1}^{R} (w(j) - p(j))^2}\; b \tag{5.3}$$

5.3.3 Activation Function

The activation function f_{act} calculates the current activation $c(t)$, a scalar, by evaluating the net input n, previous activations $c(t-1), c(t-2), \ldots$, and other parameters

$$c(t) = f_{act}(n, c(t-1), c(t-2), \ldots) \tag{5.4}$$

The linear activation function implements an activation that rises as the net input n increases discounting previous activations

$$c(t) = k\, n \tag{5.5}$$

where k is the slope. The parameter $k = 1$ gives the identity function. The bias b of the propagation function f_{pro} can be used to account for any offset.

Other functions; for example for brain state in the box (BSB), and distributed memory and amnesia (DMA) networks; model the activation in more detail. The net input n accumulates over time, and a decay term moves the activation back towards a steady state.

The Hopfield activation function evaluates the sign of the net input n; and for a net input n equal to zero, the activation remains unchanged

$$c(t) = \begin{cases} m & \text{for } n < 0 \\ c(t-1) & \text{for } n = 0 \\ 1 & \text{for } n > 0 \end{cases} \tag{5.6}$$

where, dependent on the model, $m = -1$ or $m = 0$.

5.3.4 Output Function

The output function f_{out} determines the scalar output a of the neuron that depends on the activation c. Output functions are usually monotonically increasing functions of the activation c; and may contain additional threshold, limit, or slope parameters

$$a = f_{out}(c,\ldots) \tag{5.7}$$

In some networks, i.e. competitive networks, the output of a neuron depends on its activation as well as on the activation of other neurons. Some ANNs require neurons with a differentiable output function.

The linear output function implements an output that rises as the activation increases

$$a = k\,(c - \vartheta) \tag{5.8}$$

where ϑ is a threshold that shifts the function out off the origin and k is the slope. $\vartheta = 0$ and $k = 1$ gives the identity function.

The hard limit output function outputs minimum value m for activations less than threshold ϑ and maximum value M for activations greater than or equal to threshold ϑ

$$a = \begin{cases} m & \text{for } c < \vartheta \\ M & \text{for } c \geq \vartheta \end{cases} \tag{5.9}$$

where $m < M$.

The saturating linear output function is a linear function within a range of input values, $[-l,l]$, and a hard limit function outside that range. The general function is:

$$a = \begin{cases} m & \text{for } (c-\vartheta) < -l \\ k(c-\vartheta) & \text{for } -l \le (c-\vartheta) \le l \\ M & \text{for } (c-\vartheta) > l \end{cases} \tag{5.10}$$

where $l = \dfrac{M-m}{2k}$.

The general log-sigmoid output function, that is differentiable, maps the input range $(-\infty,+\infty)$ into the output range (m,M):

$$a = m + \frac{M-m}{1+e^{-4k\frac{c-\vartheta}{M-m}}} \tag{5.11}$$

where k is the slope, ϑ is the threshold, m is the minimum value, and M is the maximum value.

The parameters $k = 1/4$, $\vartheta = 0$, $m = 0$, and $M = 1$ give a log-sigmoid output function that maps the input range $(-\infty,+\infty)$ into the output range $(0,1)$

$$a = \frac{1}{1+e^{-c}} \tag{5.12}$$

The parameters $k = 1$, $\vartheta = 0$, $m = -1$, and $M = 1$ give the hyperbolic tangent sigmoid output function.

If the neuron uses the linear activation function from equation 5.5, bias b of the propagation function f_{pro} accounts for threshold ϑ in the output functions.

The output function of radial-basis neurons is not monotonic

$$a = e^{-c^2} \tag{5.13}$$

The radial-basis neuron works as a detector that outputs one whenever the input vector p is identical to weight vector w.

5.3.5 Simplified Artificial Neuron

For many types of neuron either the activation or the output function is the identity function, hence both functions can be combined to a single transfer function f_{trans}. Figure 5.3 shows the structure of a simplified artificial neuron.

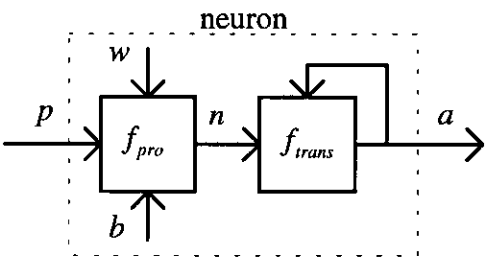


Figure 5.3 Structure of a Simplified Artificial Neuron

ANNs are usually arranged in layers each of which consists of identical neurons. H. B. Demuth and M. Beale (1994, chapter 2) devised a notation that can be easily extended from a single neuron, as shown in figure 5.4, to layers and networks. Dimensions are in row×column notation. Note that R is the number of inputs and weights, thus the number of weights is limited to the number of inputs.

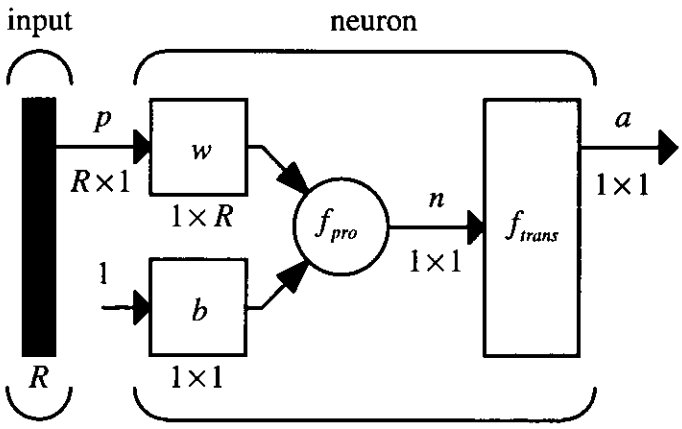
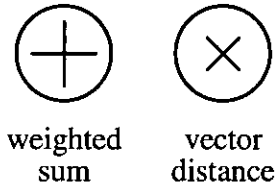
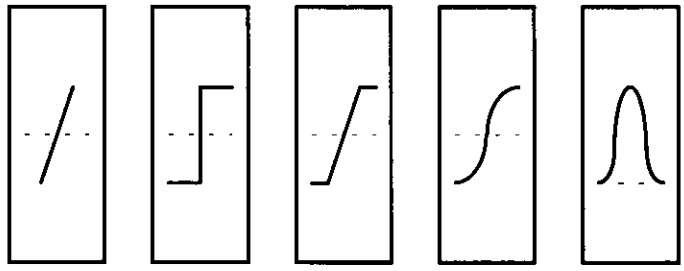


Figure 5.4 Notation of a Simplified Artificial Neuron

The propagation function f_{pro} and the transfer function f_{trans} can be visualized by the appropriate symbols, some of which are shown in figure 5.5.



a) Propagation Functions



b) Transfer Functions

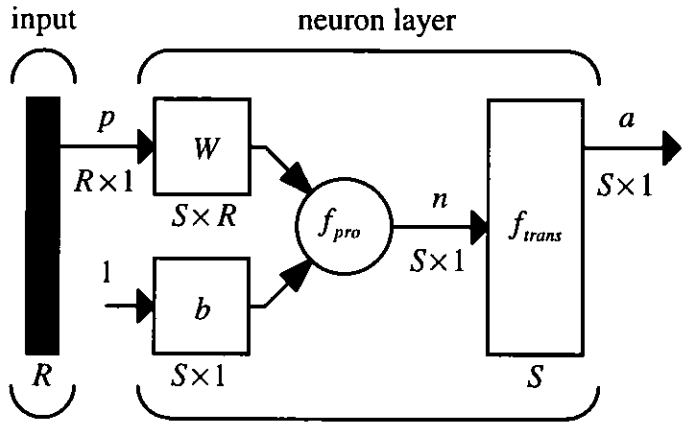
Figure 5.5 Symbols for Functions of Artificial Neuron

5.4 Feedforward Artificial Neural Networks

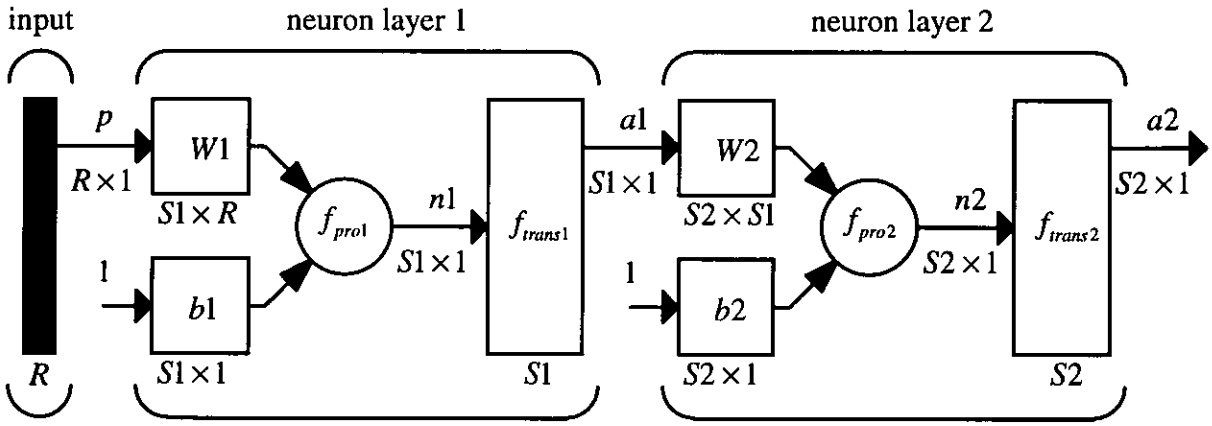
5.4.1 Structure of Feedforward Artificial Neural Networks

Hardware complexity and software performance limit the size of practical ANNs currently to up to about 10×10^3 artificial neurons and 100×10^3 connections. Artificial neurons can be interconnected to any kind of structure, however ANNs are usually arranged in layers each of which consisting of identical neurons.

Figure 5.6 depicts the layer diagrams of generic feedforward ANNs with one and two layers of trainable neurons. In feedforward ANNs each layer only receives inputs from preceding layers, i.e. there are no feedback connections. The one-layer ANN has R inputs and S neurons, hence weight matrix W consists of $S \times R$ elements. The two-layer ANN has R inputs, $S1$ and $S2$ neurons in layer 1 and 2 respectively, an $S1 \times R$-elements weight matrix $W1$, and an $S2 \times S1$-element weight matrix $W2$. The output of every neuron in layer 1 feeds into the input of every neuron in layer 2. The number of layers may be increased to extend the ANN. Note that the layer that generates the network output is referred to as the output layer, the remaining layers are referred to as hidden layers.



a) Feedforward ANN with One Layer



b) Feedforward ANN with Two Layers

Figure 5.6 Generic Feedforward Artificial Neural Networks

An ANN usually functions in either of two modes of operation. During learning the ANN adapts its structure and parameters to match a set of training data according to a specified learning strategy and learning rule. Note that most ANNs adapt their parameters, i.e. weights and biases, rather than their structure. During forward propagation, or recall, the ANN accepts input data and generates output data, however the weights and biases remain unchanged.

5.4.2 Forward Propagation

During forward propagation input data, i.e. an R-element input vector p, are presented to the inputs of the neurons in layer 1; see figure 5.6 b). Using the appropriate propagation function and transfer function, the $S1$-element output vector of layer 1, $a1$, is calculated and is presented to the inputs of the neurons in layer 2. The output vector of layer 2, $a2$, is determined similarly. For ANNs with more layers this process can be extended accordingly. Assuming that the input vector p remains unchanged, and that the transfer function does not utilize previous output values; recalculation of the network produces identical values.

5.4.3 Learning

During learning the network is modified so that the ANN adapts to its task. Although modifications to the structure of the network; for example number and type of neurons, and number of layers; are possible, most ANNs change their parameters in order to adapt. During a learning step the weights, that resemble synapses in biological neural networks, are adjusted

$$W(t) = W(t-1) + \Delta W \tag{5.14}$$

where the changes to the weights, ΔW, are defined by a learning rule. Note that the bias can be regarded as a weight element connected to a constant input of one. Learning usually requires many learning steps.

When the required output, i.e. target, to a given input is known; supervised learning can be utilized to minimize the difference between the output, actually generated from the input, and the target. The input vector p and the corresponding target vector t build a training pair. The training set is a collection of training pairs, and can be represented by

input matrix P and target matrix T. One application of the whole training set is referred to as an epoch.

Compared to weight adjustments per learning step based on individual training pairs; batch training, that produces one weight adjustment per epoch based on the complete training set, improves learning of an ANN; see (H. B. Demuth and M. Beale 1994, p. 5/7).

5.4.4 Hebb Rule

D. O. Hebb (1949) postulated that if two neurons were concurrently active, the weight of the corresponding connection would increase, hence the weight adjustment $\Delta W(i,j)$ can be defined as

$$\Delta W(i,j) = lr\ a(i)\ p(j) \tag{5.15}$$

where $a(i)$ is the output of neuron i; $p(j)$ is the jth input to neuron i, i.e. the output of neuron j; and lr is the learning rate.

The learning rate controls the size of the weight changes during learning. For supervised learning the target $t(i)$ replaces the output $a(i)$

$$\Delta W(i,j) = lr\ t(i)\ p(j) \tag{5.16}$$

However, as targets are only available for neurons in the last layer, equation 5.16 can only be applied to neurons in single-layer networks and neurons in output layers. Note that the difference between target and output is not taken into account. Weights can be initially set to zero. The order of applying the training pairs or increasing the number of epochs do not improve learning:

$$\Delta W(i,j) = k \sum_{q=1}^{Q} lr\, T(i,q)\, P(j,q) \tag{5.17}$$

where k is a scaling factor that can account for the number of epochs, Q is the number of training pairs in the training set, T is the target matrix, and P is the input matrix.

5.4.5 Delta Rule

The delta rule, also referred to as Widrow-Hoff rule, evaluates the difference between target and output to calculate the weight adjustment $\Delta W(i,j)$

$$\Delta W(i,j) = lr\,(t(i) - a(i))\, p(j) \tag{5.18}$$

where $t(i)$ is the target for neuron i; $a(i)$ is the output of neuron i; $p(j)$ is the jth input to neuron i, i.e. the output of neuron j; and lr is the learning rate. For $t(i) > a(i)$ the weight adjustment $\Delta W(i,j)$ is positive, for $t(i) < a(i)$ the weight adjustment is negative, and for $t(i) = a(i)$ the weight adjustment is zero. However the weight can only be changed when input $p(j)$ contributes to the output, i.e. $p(j) \neq 0$. The delta rule can be applied to neurons in single-layer networks.

Neurons in a perceptron network have a hard limit transfer function, and usually output either 0 or 1. Therefore the targets can only be 0 or 1. With a learning rate $l_r = 1$, equation 5.18 resembles the perceptron learning rule: for $(t(i) - a(i)) = 1$ the weight adjustment $\Delta W(i,j)$ is $p(j)$, for $(t(i) - a(i)) = -1$ the weight adjustment is $-p(j)$, and for $(t(i) - a(i)) = 0$ the weight adjustment is zero.

For batch training equation 5.18 can be extended to include the complete training set

$$\Delta W(i,j) = lr \sum_{q=1}^{Q} \left(T(i,q) - A(i,q)\right) P(j,q) \tag{5.19}$$

where Q is the number of training pairs in the training set, T is the target matrix, A is the output matrix, and P is the input matrix.

5.4.6 Error-backpropagation Algorithm

The error-backpropagation algorithm was described by P. J. Werbos (1974), and popularized by D. E. Rumelhart et al. (1986a and b). It can be applied to neurons with nonlinear, but monotonous differentiable transfer function in multilayer networks. Weights are initially set to 'random' values. The aim of the error-backpropagation algorithm is to find the weights of the ANN that minimize a cost function for a given training set. Since there are no targets for calculating weight adjustments in hidden layers, the algorithm first uses the input to generate the output of the ANN, updates the neurons in the output layer, and then works backwards.

The algorithm uses a gradient-descent technique to minimize the cost function E of the output layer *out* that is the squared difference between target and output. The q th pair of the training set contributes to the cost function

$$E(q) = \frac{1}{2} \sum_{i=1}^{S_{out}} \left(T(i,q) - A_{out}(i,q)\right)^2 \tag{5.20}$$

where S_{out} is the number of neurons in the output layer, T is the target matrix, and A_{out} is the output matrix of the ANN. Note that the scaling factor 1/2 does not compromise the minimization.

The cost function E of the error-backpropagation algorithm is the sum of the individual contributions

$$E = \sum_{q=1}^{Q} E(q) = \frac{1}{2} \sum_{q=1}^{Q} \sum_{i=1}^{S_{out}} \left(T(i,q) - A_{out}(i,q)\right)^2 \tag{5.21}$$

where Q is the number of training pairs in the training set.

The gradient-descent technique employed by the error-backpropagation algorithm uses the partial derivative of the error function E with respect to weight $W_l(i,j)$ in layer l to obtain a weight adjustment $\Delta W_l(i,j)$ that is opposite to the gradient

$$\Delta W_l(i,j) = -lr \frac{\partial E}{\partial W_l(i,j)} \tag{5.22}$$

where lr is the learning rate. Hence the error E decreases as learning progresses.

Using the sum of weighted inputs as the propagation function, the net input $N_l(i,q)$ of neuron i in layer l for training pair q is

$$N_l(i,q) = \sum_{j=1}^{R_l} \left(W_l(i,j)\, P_l(j,q)\right) \tag{5.23}$$

where R_l is the number of inputs to layer l, W_l is the weight matrix, and P_l is the input matrix of layer l containing neuron i. Note that the input matrix P_l is identical to the output matrix A_{l-1} of the preceding layer $l-1$. With reference to equation 5.2, the bias is regarded as a weight element connected to a constant input of one.

The partial derivative of the net input $N_l(i,q)$ with respect to weight $W_l(i,j)$ is

$$\frac{\partial N_l(i,q)}{\partial W_l(i,j)} = \frac{\partial}{\partial W_l(i,j)} \sum_{j=1}^{R_l} \left(W_l(i,j)\, P_l(j,q)\right) = P_l(j,q) \tag{5.24}$$

Expanding equation 5.22 using equation 5.21 gives

$$\Delta W_l(i,j) = -lr \frac{\partial}{\partial W_l(i,j)} \sum_{q=1}^{Q} E(q) = -lr \sum_{q=1}^{Q} \frac{\partial E(q)}{\partial N_l(i,q)} \frac{\partial N_l(i,q)}{\partial W_l(i,j)} \tag{5.25}$$

which can be simplified using the partial derivative from equation 5.24

$$\Delta W_l(i,j) = -lr \sum_{q=1}^{Q} \frac{\partial E(q)}{\partial N_l(i,q)} P_l(j,q) = lr \sum_{q=1}^{Q} \delta_l(i,q) P_l(j,q) \tag{5.26}$$

where $$\delta_l(i,q) = -\frac{\partial E(q)}{\partial N_l(i,q)} = -\frac{\partial E(q)}{\partial A_l(i,q)} \frac{\partial A_l(i,q)}{\partial N_l(i,q)} \tag{5.27}$$

Equation 5.27 defines the error signal that, after applying the chain rule, contains the first derivative of the transfer function

$$\frac{\partial A_l(i,q)}{\partial N_l(i,q)} = \frac{\partial}{\partial N_l(i,q)} f_{trans\,l}(N_l(i,q)) = f'_{trans\,l}(N_l(i,q)) \tag{5.28}$$

For a neuron in the output layer *out* the remaining partial derivative from equation 5.27 gives after differentiating equation 5.20

$$-\frac{\partial E(q)}{\partial A_{out}(i,q)} = -\frac{\partial}{\partial A_{out}(i,q)} \frac{1}{2} \sum_{i=1}^{S_{out}} (T(i,q) - A_{out}(i,q))^2$$

$$-\frac{\partial E(q)}{\partial A_{out}(i,q)} = (T(i,q) - A_{out}(i,q)) \tag{5.29}$$

Combining equations 5.28 and 5.29 with equation 5.27, and arranging gives the error signal for a neuron in the output layer *out*

$$\delta_{out}(i,q) = f'_{trans\,out}(N_{out}(i,q))(T(i,q) - A_{out}(i,q)) \tag{5.30}$$

For batch training the weight adjustment in the output layer *out* is obtained by inserting equation 5.30 into equation 5.26

$$\Delta W_{out}(i,j) = lr \sum_{q=1}^{Q} f'_{trans\,out}\left(N_{out}(i,q)\right)\left(T(i,q) - A_{out}(i,q)\right) P_{out}(j,q) \qquad (5.31)$$

For a neuron in a hidden layer l the derivative of the individual cost function $E(q)$ is not readily available and must be derived from the succeeding layer $l+1$ by applying the chain rule to the remaining partial derivative from equation 5.27

$$-\frac{\partial E(q)}{\partial A_l(i,q)} = \sum_{h=1}^{S_{l+1}} -\frac{\partial E(q)}{\partial N_{l+1}(h,q)} \frac{\partial N_{l+1}(h,q)}{\partial A_l(i,q)} \qquad (5.32)$$

where the summation accounts for the S_{l+1} terms $\partial N_{l+1}(h,q)$.

For layer $l+1$ the error signal is defined as for layer l in equation 5.27

$$\delta_{l+1}(h,q) = -\frac{\partial E(q)}{\partial N_{l+1}(h,q)} \qquad (5.33)$$

As in equation 5.23 using the sum of weighted inputs as propagation function, the net input $N_{l+1}(h,q)$ of neuron h in layer $l+1$ for training pair q is

$$N_{l+1}(h,q) = \sum_{i=1}^{R_{l+1}} \left(W_{l+1}(h,i)\, P_{l+1}(i,q)\right) \qquad (5.34)$$

where R_{l+1} is the number of inputs, W_{l+1} is the weight matrix, and P_{l+1} is the input matrix of layer $l+1$ containing neuron h.

The partial derivative of the net input $N_{l+1}(h,q)$ with respect to input $P_{l+1}(i,q)$ is

$$\frac{\partial N_{l+1}(h,q)}{\partial P_{l+1}(i,q)} = \frac{\partial}{\partial P_{l+1}(i,q)} \sum_{h=1}^{R_{l+1}} \left(W_{l+1}(h,i)\, P_{l+1}(i,q)\right) = W_{l+1}(h,i) \qquad (5.35)$$

Note that the output of layer l, $A_l(i,q)$, is identical to the input of layer $l+1$, $P_{l+1}(i,q)$. Combining equations 5.33 and 5.35 with equation 5.32, and arranging produces weighted error signal of layer $l+1$

$$-\frac{\partial E(q)}{\partial A_l(i,q)} = \sum_{h=1}^{S_{l+1}} \delta_{l+1}(h,q)\, W_{l+1}(h,i) \tag{5.36}$$

Combining equations 5.28 and 5.36 with equation 5.27, and arranging gives the error signal for a neuron in the layer l

$$\delta_l(i,q) = f'_{trans\,l}\left(N_l(i,q)\right) \sum_{h=1}^{S_{l+1}} W_{l+1}(h,i)\, \delta_{l+1}(h,q) \tag{5.37}$$

For batch training the weight adjustment in layer l is obtained by inserting equations 5.37 into equation 5.26

$$\Delta W_l(i,j) = lr \sum_{q=1}^{Q} f'_{trans\,l}\left(N_l(i,q)\right) \sum_{h=1}^{S_{l+1}} W_{l+1}(h,i)\, \delta_{l+1}(h,q)\, P_l(j,q) \tag{5.38}$$

For neurons with a log-sigmoid transfer function equation 5.12 can be differentiated as follows

$$f(x) = \frac{1}{1+e^{-x}} = \left(1+e^{-x}\right)^{-1} \tag{5.39}$$

$$f'(x) = -1\left(1+e^{-x}\right)^{-2}\left(-e^{-x}\right) = \frac{1}{1+e^{-x}}\,\frac{1}{1+e^{-x}}\,e^{-x} = \frac{1}{1+e^{-x}}\,\frac{1+e^{-x}-1}{1+e^{-x}}$$

$$f'(x) = \frac{1}{1+e^{-x}}\left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) = f(x)\left(1-f(x)\right) \tag{5.40}$$

Since the transfer function determines the output of a neuron from its net input

$$A_l(i,q) = f_{trans\,l}\left(N_l(i,q)\right) \tag{5.41}$$

the first derivative of the log-sigmoid transfer function can be expressed as

$$f'_{trans\,l}\left(N_l(i,q)\right) = A_l(i,q)\left(1 - A_l(i,q)\right) \tag{5.42}$$

For batch training the weight adjustment for neurons in the output layer *out* with a log-sigmoid transfer function is

$$\Delta W_{out}(i,j) = lr \sum_{q=1}^{Q} A_{out}(i,q)\left(1 - A_{out}(i,q)\right)\left(T(i,q) - A_{out}(i,q)\right) P_{out}(j,q) \tag{5.43}$$

For batch training the weight adjustment for neurons in layer l with a log-sigmoid transfer function is

$$\Delta W_l(i,j) = lr \sum_{q=1}^{Q} A_l(i,q)\left(1 - A_l(i,q)\right) \sum_{h=1}^{S_{l+1}} W_{l+1}(h,i)\,\delta_{l+1}(h,q)\,P_l(j,q) \tag{5.44}$$

For neurons with a linear transfer function equation 5.8 can be differentiated as follows

$$f(x) = k(x - \vartheta) \tag{5.45}$$

$$f'(x) = k \tag{5.46}$$

For batch training the weight adjustment for neurons in a single-layer network with a linear transfer function is

$$\Delta W(i,j) = lr \sum_{q=1}^{Q} k\left(T(i,q) - A(i,q)\right) P(j,q) \tag{5.47}$$

where k is a constant that can be summed and aggregated with the learning rate lr to resemble equation 5.19, the delta rule for batch training. Note that the error-backpropagation algorithm is referred to as the generalized delta rule.

Figure 5.7 depicts the structure of error-backpropagation algorithm for batch training. The weight matrices are initialized with 'random' numbers. The range can be derived

for each weight from the expected minimum and maximum values of the corresponding input. Learning progresses until the error decreases below a specified value or the maximum number of epochs is reached. During forward propagation the input matrix of the training set is presented to the input of the ANN. The outputs of all layers for all training pairs are calculated and stored. The error signals of the output layer for all training pairs are calculated from the output matrix of the ANN and the target matrix of the training set. Starting from the last hidden layer, the error signals of each hidden layer are calculated from the error signals of the succeeding layer. When all error signals are available, the weight matrices are updated.

```
initialize weights
while learning not finished
    present input matrix of training set to network
    obtain output matrices of all layers
    calculate error signals of output layer; equation 5.30
    select last hidden layer
    for all hidden layers
        calculate error signals of hidden layer; equation 5.37
        select preceding hidden layer
    adjust weights; equation 5.26
```

Figure 5.7 Structure of Error-backpropagation Algorithm

The backpropagation algorithm has been improved using momentum and adaptive learning rate, and the Levenberg-Marquardt optimization is an alternative technique to gradient descent; see (H. B. Demuth and M. Beale 1994, pp. 5/31-5/34).

5.4.7 Multilayer Feedforward Artificial Neural Networks

Single-layer ANNs have proved to be useful in a range of applications. They map similar input vectors to similar output vectors. The single-layer perceptron, first devised by F. Rosenblatt (1959), is suited for simple classification problems. Figure 5.8 shows a

single-layer perceptron having S neurons with a hard-limit transfer function that generates either 0 for net inputs less than zero or 1 otherwise. Note that bias b accounts for the threshold. The perceptron is trained on examples of correct behaviour using the perceptron learning rule.

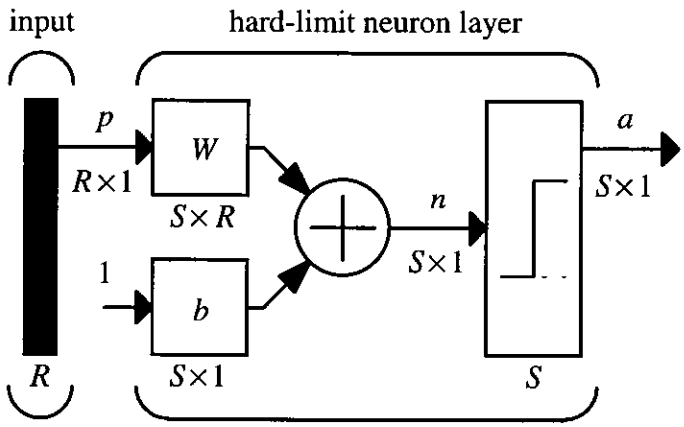


Figure 5.8 Single-layer Perceptron

F. Rosenblatt proved that, if the input vectors are linearly separable into a number of classes, the perceptron learning rule converges in finite time and positions decision hyperplanes between the classes. However, if the input vectors are not linearly separable, learning will never reach a stage where all vectors are properly classified.

The mapping of similar input vectors to similar output vectors restricts the usefulness of single-layer ANNs. For many practical problems very similar input vectors require very different output vectors. M. Minsky and S. Papert (1969) reported, with great negative effect on the popularity of neural networks, that these ANNs were not suitable for many problems including the exclusive-OR (XOR) problem. Note that the delta rule converges for linearly separable and linearly inseparable input vectors, but may or may not produce separating hyperplanes (R. C. Gonzalez and R. E. Woods 1992, pp. 602-603).

A two-layer ANN is the simplest form of a multilayer ANN. Assuming that each layer consists of identical neurons, a variety of networks can be created from a selection of neuron types, that are outlined in section 5.3. In principle, different types can be used for different layers or even different neurons in the same layer; however the common approach is to use the same type throughout the ANN (R. C. Gonzalez and R. E. Woods 1992, p. 605). While the number of neurons in the output layer is determined, for example, by the number of pattern classes; the number of neurons in the hidden layers determines the learning capacity of the ANN.

A two-layer ANN having $S1$ neurons with a hard-limit transfer function in the hidden layer and $S2$ neurons with a hard-limit transfer function in the output layer is shown in figure 5.9. Neurons in the hidden layer, i.e. layer 1, cannot be trained using the perceptron learning rule or delta rule, since targets are not available. A hidden layer with 'random' weights may be used to pre-process the input vectors so that they may become linearly separable (H. B. Demuth and M. Beale 1994, pp. 3/21-3/22).

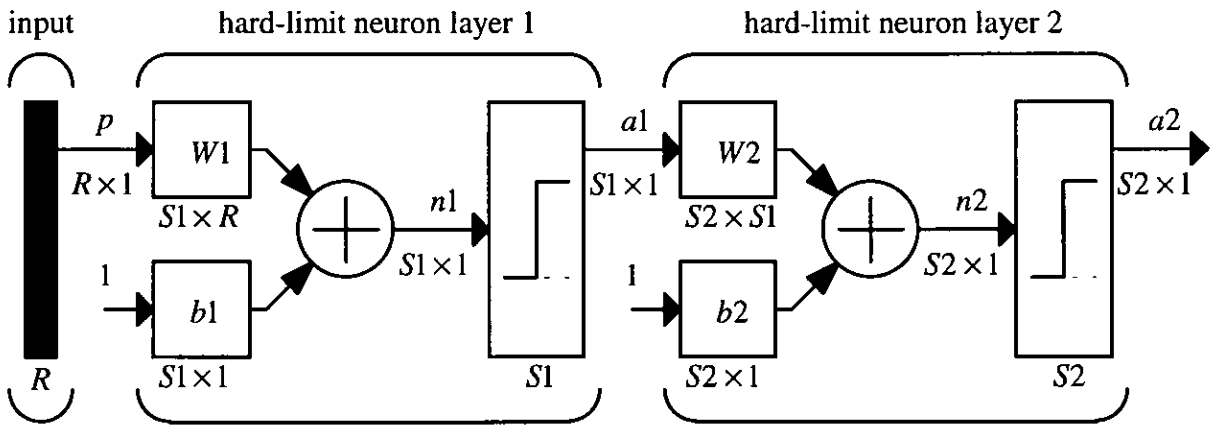


Figure 5.9 Two-layer Perceptron

Figure 5.10 shows a two-layer ANN having $S1$ neurons with a linear transfer function in the hidden layer and $S2$ neurons with a linear transfer function in the output layer.

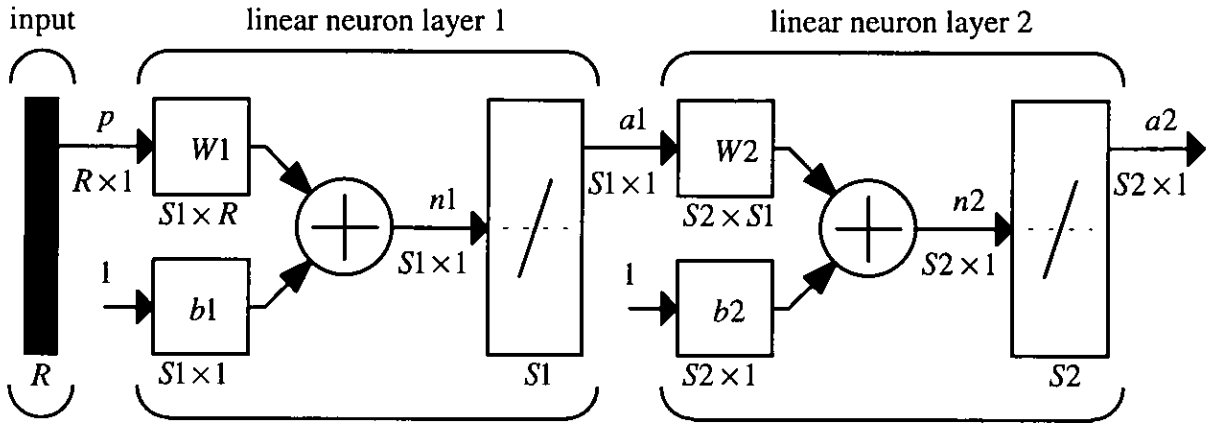


Figure 5.10 Two-layer Linear ANN

Using equations 5.2 and 5.5, the output vector a_l of linear layer l can be expressed as

$$a_l = k_l\left(W_l p_l + b_l\right) \tag{5.48}$$

where k_l is the slope of the transfer function, W_l is the weight matrix of layer l, p_l is the input vector to layer l, and b_l is the bias vector.

Hence the output vector of layer 1 is

$$a_1 = k_1\left(W_1 p_1 + b_1\right) = k_1\left(W_1 p + b_1\right) \tag{5.49}$$

Similarly, the output vector of layer 2 is

$$a_2 = k_2\left(W_2 p_2 + b_2\right) = k_2\left(W_2 a_1 + b_2\right) \tag{5.50}$$

Combining equations 5.49 and 5.50 gives

$$a_2 = k_2\left(W_2 k_1\left(W_1 p + b_1\right) + b_2\right) = k_1 k_2\left(W_1 W_2 p + W_2 b_1 + \frac{1}{k_1} b_2\right) \tag{5.51}$$

The output vector of a single-layer linear ANN is

$$a = k_{single}\left(W_{single} p + b_{single}\right) \tag{5.52}$$

For the parameters $k_{single} = k_1 k_2$, $W_{single} = W_1 W_2$, and $b_{single} = W_2 b_1 + \frac{b_2}{k_1}$ both ANNs produce identical output vectors for the same input vectors. Hence, a multilayer linear ANN is not more powerful than a single-layer linear ANN (H. B. Demuth and M. Beale 1994, p. 4/31).

A two-layer ANN having $S1$ neurons with a log-sigmoid transfer function in the hidden layer and $S2$ neurons with a log-sigmoid transfer function in the output layer is shown in figure 5.11. The ANN can be trained, using an appropriate training set, to generate reasonable output vectors for new, i.e. previously unseen, input vectors. Note that the output of this ANN is restricted to the range $(0,1)$, since the log-sigmoid transfer function uses equation 5.12.

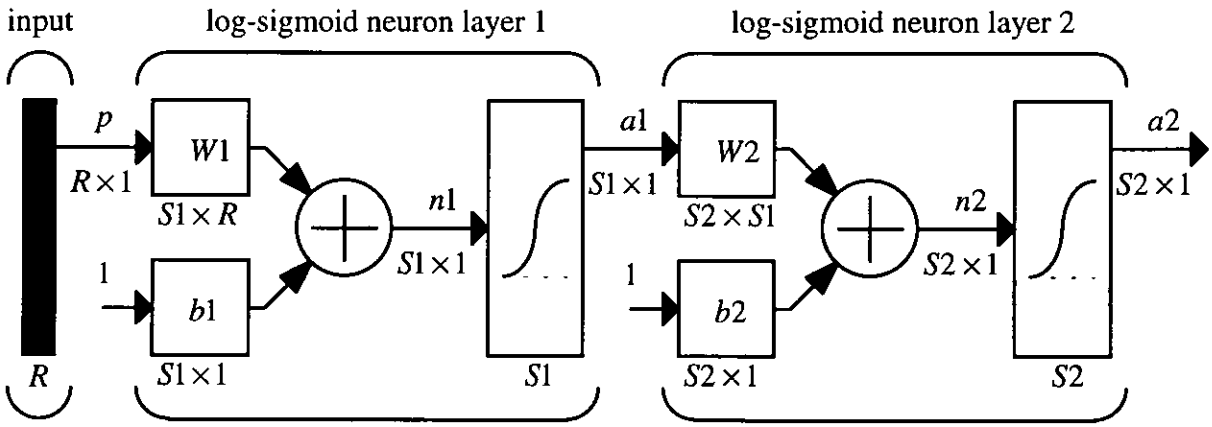


Figure 5.11 Two-layer Log-sigmoid ANN

Since the linear function is differentiable and monotonically increasing, neurons with this type of transfer function can be employed, for example in conjunction with neurons having log-sigmoid transfer function, in the output layer of multilayer feedforward ANNs that are trained using the error-backpropagation algorithm. This enables the ANN to output any value, rather than only values from a relatively small range generated

by a sigmoid function. Figure 5.12 depicts a two-layer ANN having $S1$ neurons with a log-sigmoid transfer function in the hidden layer and $S2$ neurons with a linear transfer function in the output layer.

Although this subsection refers to two-layer feedforward ANNs, the number of layers may be increased to extend an ANN. Multilayer nonlinear ANNs, that are trained using the error-backpropagation algorithm, can be applied to linearly separable and linearly inseparable problems. As nonlinear ANNs may have more than one local error minimum; the error-backpropagation algorithm, employing a gradient-descent technique, may not always, dependent on the initial weights, find the global error minimum. The number of hidden neurons has great effect on the performance of the ANN. If the number of hidden neurons is too small, the ANN may not be able to learn the information contained in the training set. If the number of hidden neurons is too large, the ANN may not be able to generate a reasonable output vector for a new input vector.

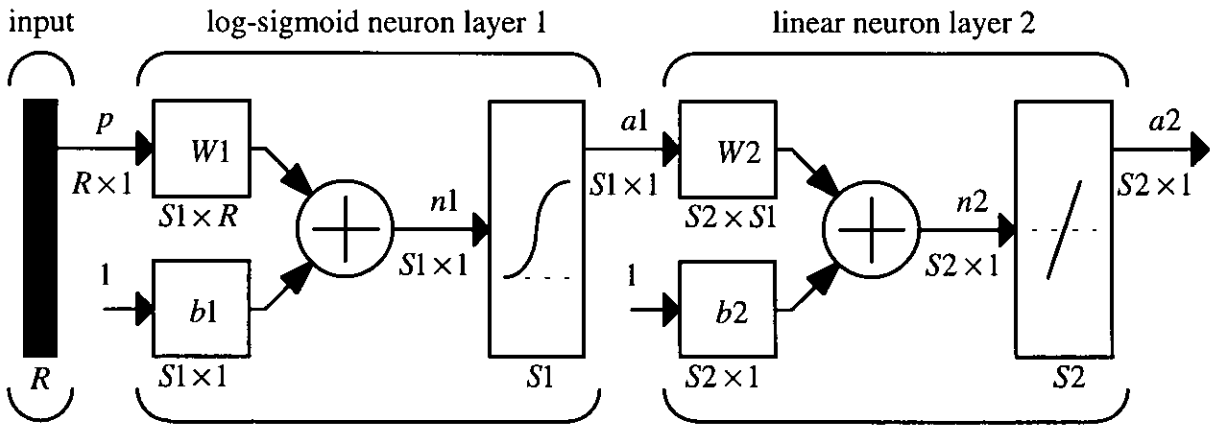


Figure 5.12 Two-layer Log-sigmoid Linear ANN

5.5 Artificial Neural Networks in Digital Image Compression

Over recent years numerous approaches have been proposed for employing ANNs in digital image processing in general and digital image compression in particular. This section outlines some of those techniques.

In predictive coding, multilayer feedforward ANNs can, unlike conventional predictors, take advantage of nonlinear inter-element redundancies. In addition neural-network-based predictors are less sensitive to noise than conventional predictors; see for example (Z. He and H. Li 1990).

In direct block-based application of ANNs to digital image compression each block of pixels extracted from the original image is interpreted as an input vector to a multilayer feedforward ANN. The number of neurons in the output layer is identical to the number of network inputs. The targets of the training set are identical to the corresponding inputs. To achieve compression, the number of neurons in the hidden layer is smaller than the number of network inputs; and the output precision of the neurons in the hidden layer, that represent the encoded block, may be smaller than that of the network inputs and neurons in the output layer. G. W. Cottrell et al. (1989) used a feedforward ANN using error backpropagation. G. L. Sicuranza et al. (1990) reported similar work; they introduced activity functions to classify each block, and to select one of four or six ANNs for adaptive encoding (S. Marsi et al. 1991). D. Cai et al. (1992) utilized two DCT-based activity functions to classify each block and to select one of four linear ANNs of identical structure for encoding. D. Cai and M. Zhou (1992) employed a statistical activity function to classify each block and to select one of two ANNs with different ratios of network inputs and neurons in the hidden layer. F. Arduini et al.

(1992) used the intensity and direction of spatial activity to split an image into variable-size blocks that are encoded by ANNs with appropriate number of network inputs and neurons in the output layers, and varying ratios of network inputs and neurons in the hidden layer. S. Carrato and S. Marsi (1992) proposed a parallel structure of ANNs with different ratios of network inputs and neurons in the hidden layer. Each block is concurrently processed by every ANN and the highest compression ratio to meet the predefined SNR is chosen, thus implementing feedback.

In vector quantization, ANNs cluster vectors from the training set into representative regions using competitive, i. e. unsupervised, learning. The weight vector of a neuron resembles the codeword. To overcome unequal utilization of the neurons, the Kohonen self-organizing feature map (KSOFM) defines a neighbourhood around the neuron that wins during a learning step and updates that neighbourhood. Thus adjacent neurons respond to similar input vectors. One or more ANNs are employed to efficiently design the codebook. S. P. Luttrell (1989) employed neural-network-based vector quantization for the compression of synthetic aperture (SAR) images. C. C. Lu and Y. H. Shin (1992) designed separate codebooks for edge and background blocks. M. R. Carbonara et al. (1992) designed equiprobable codebooks using frequency-sensitive competitive learning. H. Lui and D. Y. Y. Yun (1992) compared different approaches and proposed the near-optimal learning algorithm for achieving real-time vector quantization. S. Panchanathan et al. (1992) suggested a combination of the error-backpropagation algorithm and KSOFM for vector quantization.

Block truncation coding converts each block of pixels extracted from the original image into mean, variance, and a binary pattern indicating whether each pixel lies above or below the mean; see (R. J. Clarke 1995, pp. 175-177). G. Qiu et al. (1991) used a

Hopfield network to obtain the binary pattern, and included a classification based on block detail to implement adaptive compression (G. Qiu et al. 1993a); see also (H. B. Mitchell and M. Dorfan 1992).

L. O. Chua and T. Lin (1988) used a Hopfield network that receives spatial-domain image data and outputs binary codes to perform transform coding thus combining transform, quantization, and binary coding. H. Niemann and J. K. Wu (1993) used a two-layer feedforward linear ANN within their adaptive image-coding scheme to obtain the Karhunen-Loève transform.

Other neural-network-based digital-image-processing techniques may be exploited for digital image compression. R. A. Hutchinson and J. W. Welsh (1989), and C. Nightingale and R. A. Hutchinson (1990) considered ANNs for feature location. C. C. Klimasauskas (1990) used an ANN for edge detection. G. Qiu et al. (1993b) employed several multilayer feedforward ANNs for edge pattern learning for digital image compression. J. A. Parikh et al. (1990) reported on edge and line detection, and texture analysis using ANNs. H. Niemann and J. K. Wu (1993) devised an adaptive image coding scheme that uses neural-network-based texture classification to select a dedicated coding scheme. Image segmentation has attracted considerable attention; N. R. Pal and S. K. Pal (1993) included ANN-based approaches in their review of segmentation techniques. M. Mattavelli et al. (1995) built on earlier work (B. Macq et al. 1994) and applied ANNs to human-visual-system-based image restoration. The decoded image that is affected by coding noise is decomposed into perceptual channel components and processed pixel by pixel. Hence the number of network inputs is, in contrast to other approaches, governed only by the number of perceptual channel components.

N. P. Walker et al. (1994) described the compression of single and multiple, i.e. moving or 3-D, images using multilayer feedforward ANNs and KSOFMs. S. G. Romaniuk (1994) suggested automatic construction of ANNs for lossless image compression, instead of training ANNs of predetermined architecture. R. J. Clarke (1995, p. 224) pointed out that ANNs can be employed in any overall scheme that incorporates a stage of optimization, for example of prediction coefficients, codebooks, and transform coefficients.

5.6 Summary

Artificial neural networks; consisting of a large number of simple processing elements, i.e. neurons; are computational systems that are massively parallel and adaptive, i.e. trainable. During learning the structure and the parameters of the ANN can be modified so that it adapts to its task. ANNs are usually arranged in layers each of which consists of identical neurons. A simplified neuron consists of a propagation function; that generates the net input from inputs, weights, and bias; and transfer function; that determines the output of the neuron from the net input. A number of propagation and transfer functions have been defined. Different strategies, for example supervised and unsupervised learning, are available for learning. For supervised learning the training set contains, in addition to the input set, a target set that represents the desired outputs. ANNs can be simulated in software and implemented in hardware.

The error-backpropagation algorithm uses a gradient-descent technique to minimize the cost function E that may have more than one local error minimum. Dependent on the initial weights the error-backpropagation algorithm may not always find the global error minimum. However, it is capable of training multilayer feedforward networks

consisting of neurons with differentiable and monotonically increasing transfer functions.

Chapter 6

Neural-network-based Block Classification

6.1 Introduction

This chapter describes classification of blocks of transform coefficients in a JPEG-like image-compression scheme. The classification determines, using an artificial neural network (ANN), the dimensions of a sub-block to be encoded. The classification processing step precedes adaptive zigzag reordering, described in chapter 4, in the encoder. Since the generated sub-block does not necessarily include all nonzero coefficients, the conversion of a block of coefficients is, in some cases, lossy.

Section 6.2 focuses on quantization of transform coefficients, used in the DCT-based method of the JPEG standard as introduced in chapter 3.

Section 6.3 describes neural-network-based determination of sub-block dimensions.

Section 6.4 compares zigzag reordering with neural-network-based classification with standard as well as adaptive zigzag reordering using experimental results. Finally section 6.5 concludes the chapter with a brief summary.

6.2 Quantization of Transform Coefficients

The quantization processing step employed in the DCT-based method of the JPEG standard is shown in figure 6.1. Each coefficient $S(v,u)$ in the 8×8 block of transform coefficients represents a DCT frequency; see subsection 3.4.3.

The quantization step sizes $Q(v,u)$ are contained in a quantization table, and can be set individually for each DCT coefficient. Although only coefficients of the Fourier transform correspond directly to spatial frequency, visual thresholds can be determined for the DCT coefficients; see (H. Lohscheller 1984; and N. B. Nill 1985). For

quantization step sizes below corresponding visual thresholds, the human visual system should not be able to detect any difference between the reconstructed blocks of samples using unquantized and dequantized DCT coefficients (W. B. Pennebaker and J. L. Mitchell 1992, p. 35).

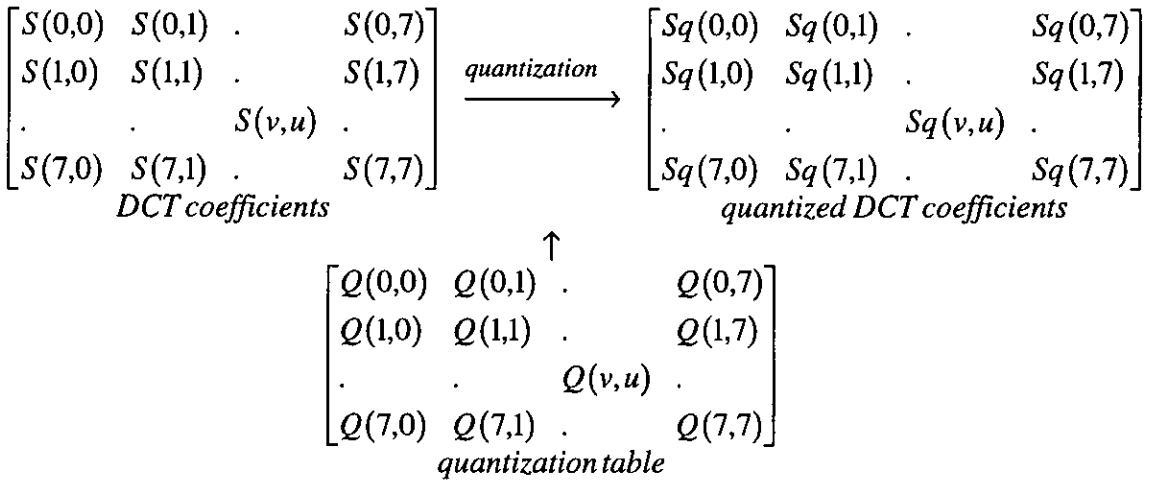


Figure 6.1 Quantization

While the transform processing steps cannot be computed with perfect accuracy, it is the quantization processing step in the DCT-based method of the JPEG standard that is specifically designed to achieve compression at the expense of accuracy. It corresponds to spatial filtering in the human visual system; see subsection 2.4.2.

6.3 Block Classification

6.3.1 Motivation for Block Classification

The JPEG standard for the DCT-based method accommodates up to four 8×8 quantization tables for processing images with up to 255 components. However, since a quantization table must be globally used for all blocks of an image component,

local changes in block content cannot be taken into account. Hence spatial masking cannot be exploited.

Block classification assesses a block of transform coefficients, and generates the dimensions of a sub-block to be retained. Since the classification processing step processes each block individually; it takes block content, i.e. the contribution of every coefficient, into account.

Adaptive zigzag reordering, described in chapter 4, performs lossless conversion; however isolated nonzero coefficients in a block of transform coefficients diminish the effectiveness of this processing step, since retaining isolated nonzero coefficients also requires that a large number of otherwise unnecessary zero coefficients are included in a sub-block. However, if the contribution of an isolated coefficient to reconstruction is found to be expendable, a significantly smaller sub-block may be retained. Note that the additional reconstruction error is limited to the corresponding block of samples. Hence the classification processing step assists, during encoding, the succeeding adaptive-zigzag-reordering processing step. Although isolated nonzero coefficients could be individually removed, the decision to sacrifice an isolated coefficient should take the contributions of all transform coefficients in a block into account.

The classification processing step is required in the encoder in order to generate the sub-block dimensions for adaptive zigzag reordering. The classification processing step employs a two-layer ANN that is trained using an error-backpropagation algorithm; see subsections 5.4.6 and 5.4.7. This additional processing step increases the workload of the encoder. However, the classification processing step is not required in the decoder.

6.3.2 Structure of the Artificial Neural Network

The classification processing step employs a feedforward ANN with 64 inputs and 64 outputs. The ANN consists of two trainable layers, i.e. hidden layer and output layer. Figure 6.2 depicts the ANN during learning; the neurons in both layers have log-sigmoid transfer functions; see equation 5.12. The hidden layer consists of 256 neurons. This number has been determined experimentally, and is a compromise between classification performance and network complexity.

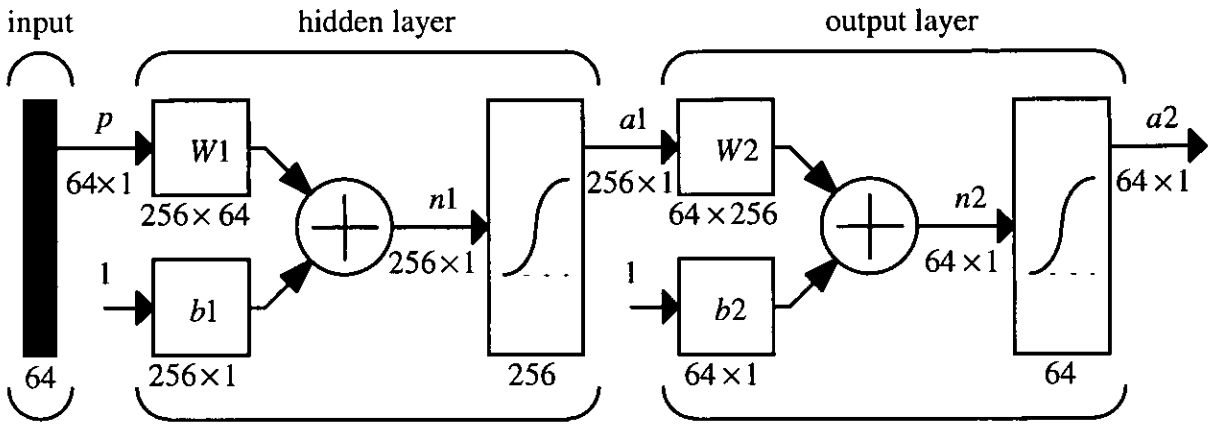


Figure 6.2 ANN for Block Classification during Learning

Figure 6.3 depicts the ANN during forward propagation, i.e. block classification; since the error-backpropagation algorithm is not being applied, the output layer produces valid and most appropriate 1-in-64 codes using the competitive transfer function that transforms the net-input vector of a layer of neurons so that the neuron receiving the greatest net input has an output of one and all other neurons have outputs of zero; see (H. B. Demuth and M. Beale 1994, pp. 13/17-13/18).

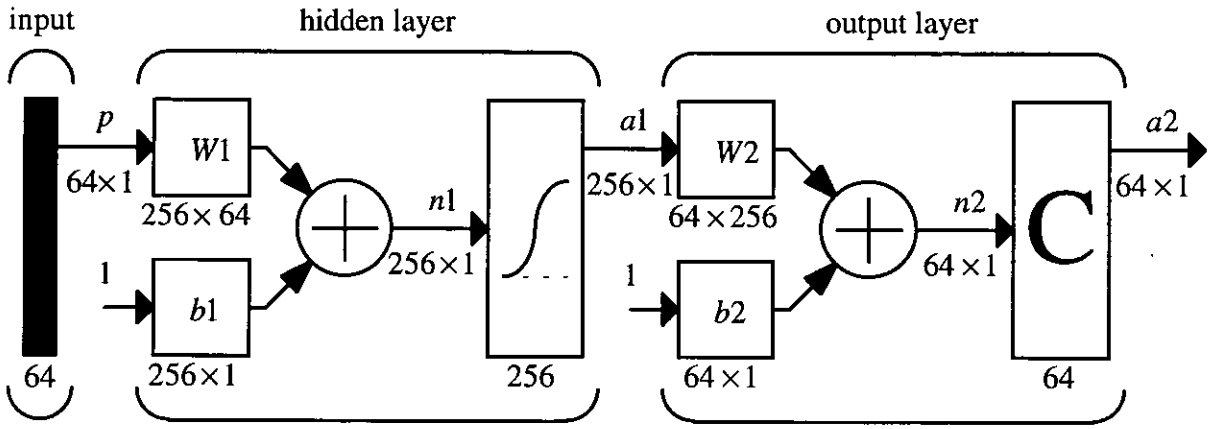


Figure 6.3 ANN for Block Classification during Forward Propagation

6.3.3 Network Inputs

Since every coefficient is to be taken into account, the number of inputs is determined by the block dimensions. A 64-element input vector is required for 8×8 blocks as defined by the JPEG standard for the DCT-based method.

The coefficients are not directly presented to the ANN. Note that 8-bit precision image samples transform to 11-bit precision DCT coefficients in the range $[-1023,1023]$. In order to homogenize network inputs, amplitudes of the DCT coefficients are classified according to their magnitude categories in JPEG; see table 3.4; and the classifications are normalized, i.e. divided by the maximum value within each block. The network inputs therefore receive input vectors representing blocks of normalized amplitude classifications, each of which is in the range $[0,1]$.

As an example, figure 6.4 depicts an 8×8 block of transform coefficients. Note that the block requires a 5×6 sub-block for lossless conversion; however, discarding the coefficient of value one at position $(5,6)$ would generate a smaller 4×5 sub-block that could be zigzag-reordered more efficiently.

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\ 1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 6.4 Example of 8×8 Block of Transform Coefficients

The corresponding 8×8 block of amplitude classifications is shown in figure 6.5. Note that the classifications are unsigned, and that larger magnitudes are de-emphasized due to the approximately logarithmically increasing magnitude categories.

$$\begin{bmatrix} 5 & 2 & 3 & 2 & 2 & 0 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 3 & 1 & 1 & 0 & 0 & 0 \\ 3 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 6.5 Example of 8×8 Block of Amplitude Classifications

Figure 6.6 depicts the resulting 8×8 block of normalized amplitude classifications that builds a 64-element input vector.

$$\begin{bmatrix} 1.0 & 0.4 & 0.6 & 0.4 & 0.4 & 0.0 & 0.0 & 0.0 \\ 0.2 & 0.4 & 0.6 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.4 & 0.2 & 0.6 & 0.2 & 0.2 & 0.0 & 0.0 & 0.0 \\ 0.6 & 0.2 & 0.4 & 0.2 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.2 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Figure 6.6 Example of 8×8 Block of Normalized Amplitude Classifications

6.3.4 Network Outputs

A 64-element output vector is required to identify directly all 64 possible sub-block dimensions using a simple 1-in-64 binary code; i.e. the vector has a one in the position of the sub-block dimensions that it represents, and zeros elsewhere. This code, although requiring 64 neurons, allows competitive selection of one output neuron, and has found to be more reliable than other codes; for example a 6-bit natural binary code that would require only six output neurons. However, note that the number of outputs could be reduced when the number of sub-block dimensions is limited; or when the sub-block dimensions, i.e. number of rows and number of columns, are coded separately. The log-sigmoid transfer function, employed during learning, is differentiable and monotonically increasing. Its output range is restricted to the range $(0,1)$; and is, therefore, appropriate for learning to output binary values (H. B. Demuth and M. Beale 1994, p. 11/42).

6.3.5 Learning

Before the ANN is employed in forward propagation for classification of blocks of transform coefficients, i.e. to determine the dimensions of sub-blocks, its weights are

adjusted during learning to suit the classification task. The ANN is trained using the error-backpropagation algorithm described in subsection 5.4.6. The weight matrices and bias vectors are initialized with 'random' numbers. Learning is carried out in two phases each of which uses batch training.

During the initial learning phase the ANN is trained on 64 idealized training pairs that correspond to the 64 possible sub-block dimensions. For each input vector in the training set, all elements that belong to a sub-block are set to one, and the elements that are outside the sub-block are set to zero. The corresponding target vectors contain the 1-in-64 codes that identify the appropriate sub-block dimensions. Note that the input vectors and the output vectors form input matrix and target matrix respectively. The initial learning phase adjusts the weights and biases towards the classification task using a smaller training set.

During the further learning phase the input matrix contains, in addition to the 64 idealized input vectors, 580 input vectors that have been derived from the images shown in appendix E; the target matrix consists of the appropriate code vectors. The 580 additional input vectors represent ten selected examples for each of 58 sub-block dimensions. However, for six of the 64 possible sub-block dimensions; namely 5×1, 6×1, 7×1, 8×1, 8×2, and 1×7; suitable examples have not been derived from the images. The generation of the authentic training pairs is described in subsection 6.4.2. The small number of idealized training pairs supports the ability of the ANN to classify ideal input vectors and input vectors that correspond to sub-block dimensions for which training pairs have not been available.

6.4 Experimental Results

6.4.1 Implementation

The neural network has been implemented, and experimental results have been obtained using MATLAB (MathWorks 1994) and its Neural Networks Toolbox (H. B. Demuth and M. Beale 1994). The transform-coefficient matrices have been generated using the Independent JPEG Group's software (Independent JPEG Group 1996). The quality setting q controls scaling of the quantization tables; see subsection 3.4.4. The experimental results have been produced for quality settings in the range from 10 ('poor' quality) to 90 ('good' quality). Appendix E contains the original images used for experimentation.

6.4.2 Authentic Training Pairs

The authentic training pairs have been generated by subjective classification of the 8×8 blocks of normalized amplitude classifications. The blocks have been studied, and the sub-block dimensions have been chosen so that most of the nonzero normalized amplitude classifications are contained in the sub-block, and only some of the smaller normalized amplitude classifications are excluded. The sub-block dimensions of the block of normalized amplitude classifications shown in figure 6.6, for example, would be 4×5. Blocks of normalized amplitude classifications that have been difficult to classify have been excluded from classification. The input matrix of the training set has been built from a selection of classified blocks, and the target matrix has been generated from the 1-in-64 codes of the corresponding sub-block dimensions.

The image Lena with a spatial resolution of 256×256 pixels has been used with quality settings $q = 75$ and $q = 90$, and the image Cameraman has been used with quality setting $q = 90$ to produce 3072 blocks of normalized amplitude classifications. From these blocks 229 blocks have not been classified, and 580 blocks have been selected to give 10 examples for each of 58 sub-block dimensions found in the images.

6.4.3 Learning

The initial learning phase has lasted for 5000 epochs, and the learning rate has been set to 0.01. Figure 6.7 depicts the mean-square error (MSE) per training pair over the course of the initial learning phase. Note that the MSE per pair allows direct comparisons of learning using training sets with different numbers of training pairs. The initial error caused through the intialization with 'random' numbers has been found to be about 16. During approximately the first 2800 epochs the MSE per pair decreases from 1 to 0.1. After 5000 epochs the MSE per pair reaches about 0.024.

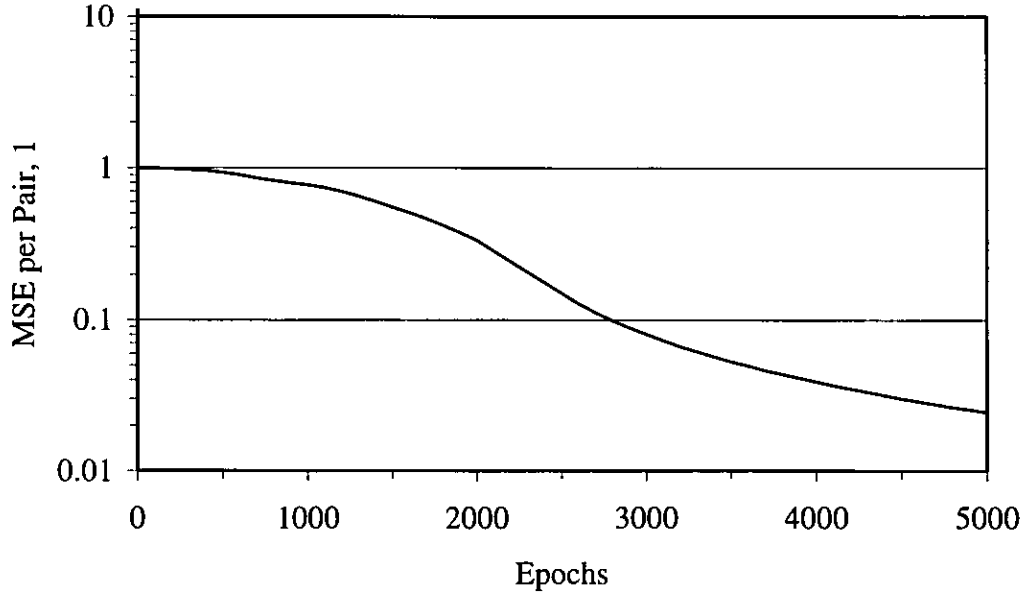


Figure 6.7 MSE per Training Pair versus Epochs during Initial Learning Phase

The further learning phase has lasted for 30000 epochs, and the learning rate has been set to 0.01. Figure 6.8 depicts the MSE per pair over the course of the further learning phase. The initial error of about 0.9 is caused through the additional training pairs. After 10000, 20000, and 30000 epochs the MSE per pair reaches about 0.085, 0.071, and 0.065 respectively.

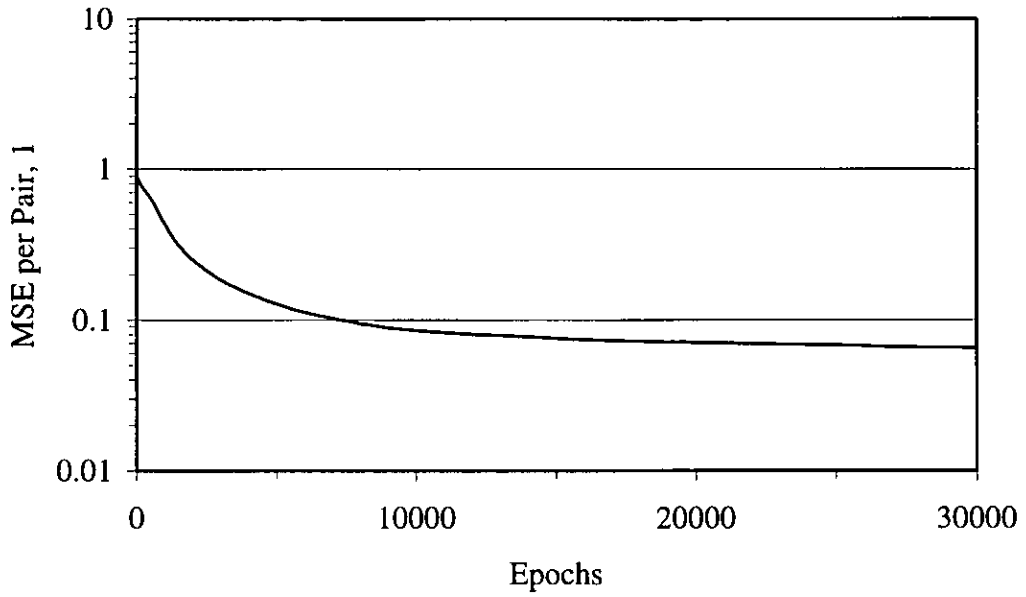


Figure 6.8 MSE per Training Pair versus Epochs during Further Learning Phase

6.4.4 Classification

The entropies of the runs of zero coefficients for zigzag reordering with neural-network-based classification, standard zigzag reordering, and adaptive zigzag reordering have been evaluated over the given range of quality settings and are presented versus the peak-signal-to-noise ratio (PSNR). The ANN used for block classification employs the weight matrices and bias vectors that have been obtained after 30000 epochs. Figures 6.9 and 6.10 depict the entropies for the image Lena with a spatial resolution of 512×512 pixels and 256×256 pixels respectively. Note that the PSNR generally

increases with increasing quality setting q. Figures 6.11 and 6.12 depict the entropies for the images Cameraman with a spatial resolution of 256×256 pixels, and F-16 with a spatial resolution of 512×512 pixels respectively. Note that, for a given quality setting and therefore the same PSNR, adaptive zigzag reordering featuring lossless conversion always produces a lower entropy of the runs of zero coefficients than standard zigzag reordering. However, zigzag reordering with neural-network-based classification featuring lossy conversion produces even lower entropies. These particular weight matrices and bias vectors lead to entropies below 1 bit.

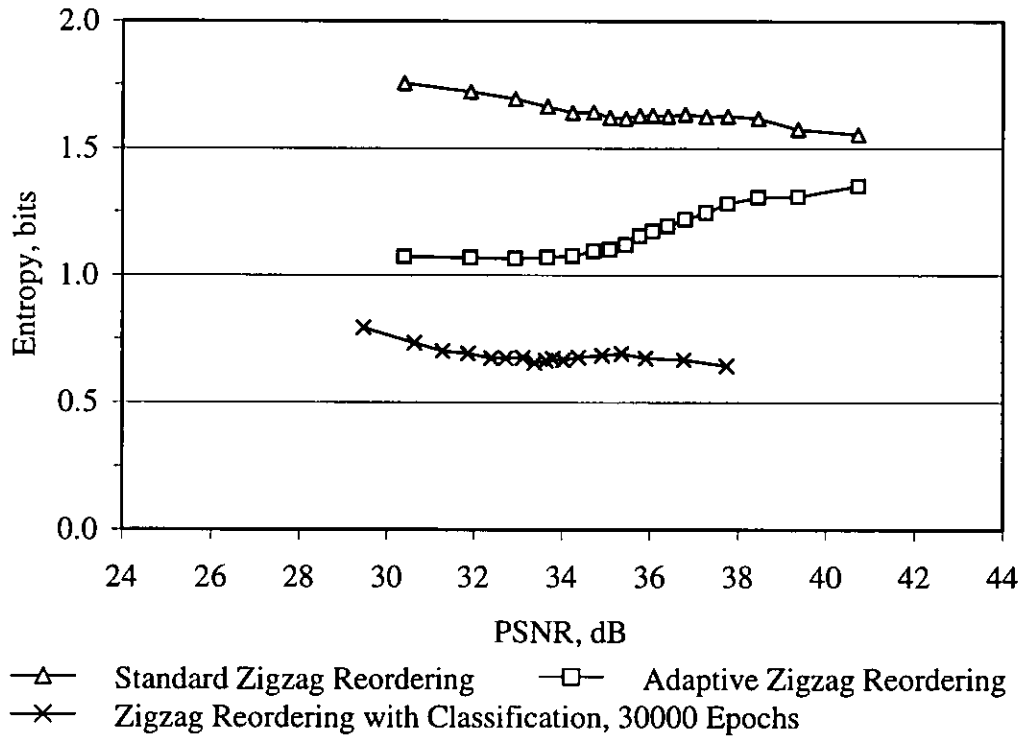


Figure 6.9 Entropy of Runs of Zero Coefficients versus Peak-signal-to-noise Ratio, Lena 512×512

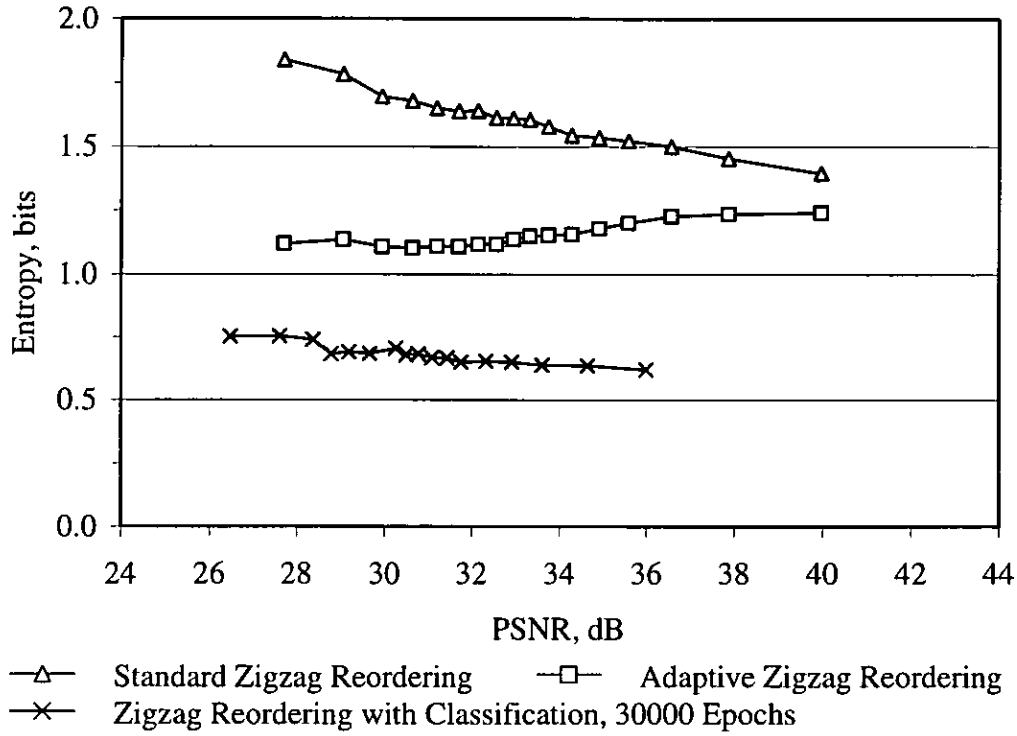


Figure 6.10 Entropy of Runs of Zero Coefficients versus Peak-signal-to-noise Ratio,

Lena 256×256

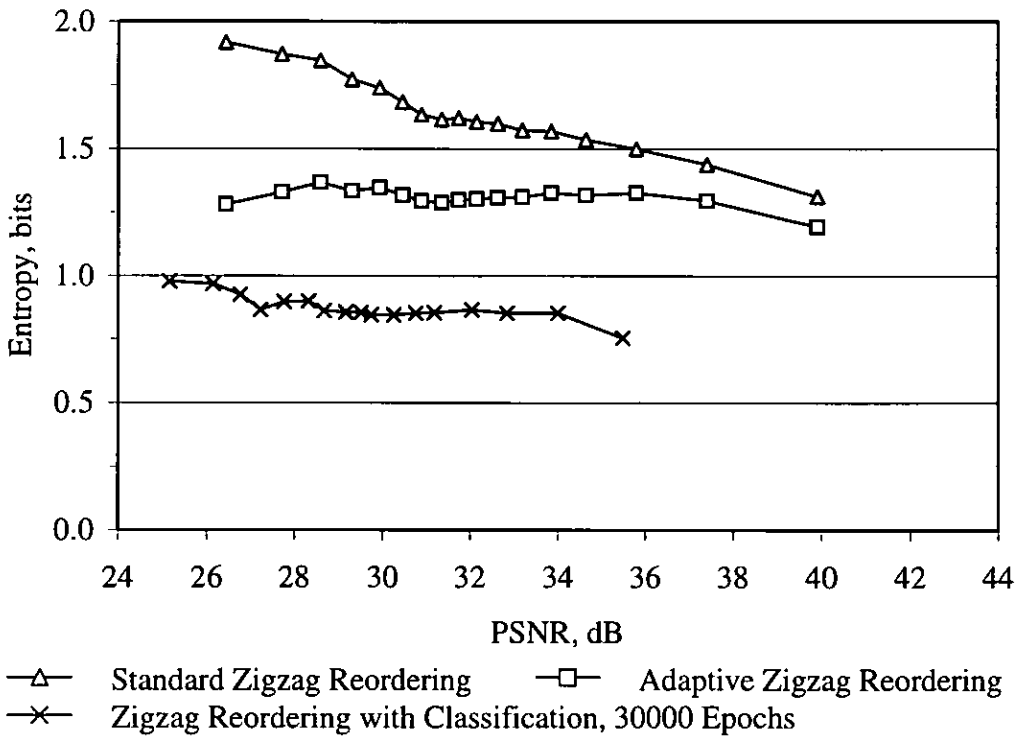


Figure 6.11 Entropy of Runs of Zero Coefficients versus Peak-signal-to-noise Ratio,

Cameraman 256×256

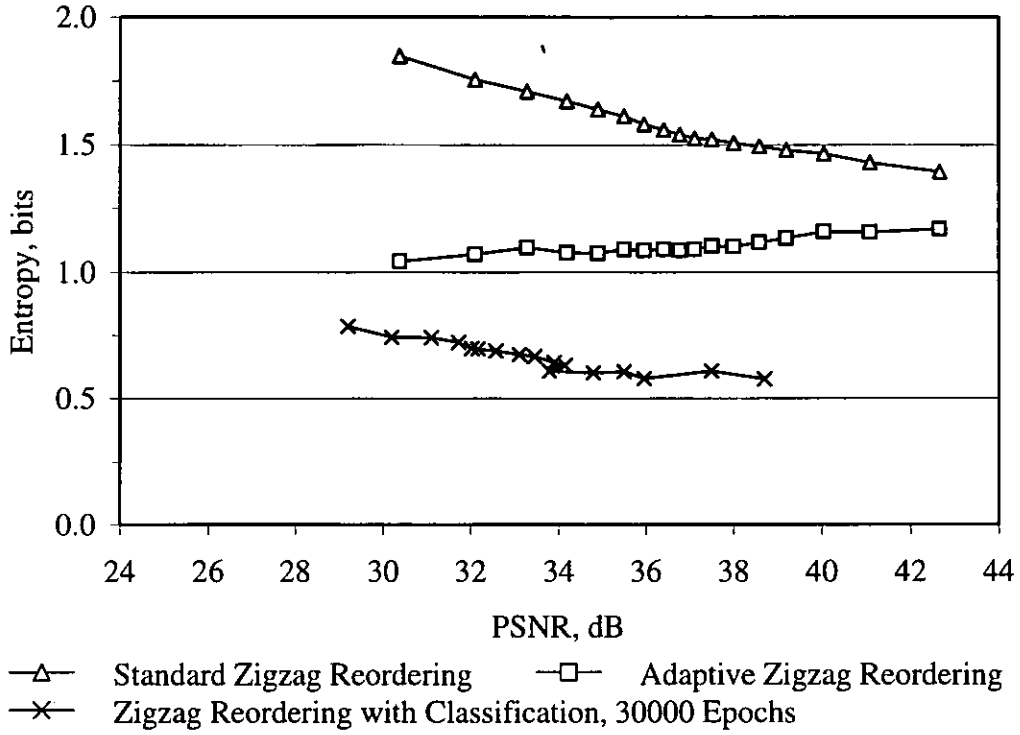


Figure 6.12 Entropy of Runs of Zero Coefficients versus Peak-signal-to-noise Ratio, F-16 512×512

Since some coefficients are discarded through the classification processing step, zigzag reordering with neural-network-based classification requires a higher quality setting in order to achieve the same PSNR as standard zigzag reordering. For a given PSNR the subjective image qualities of zigzag reordering with neural-network-based classification and standard zigzag reordering are similar. As an example, figure 6.13 and figure 6.14 depict the image Lena with a spatial resolution of 512×512 pixels for standard zigzag reordering and quality setting $q = 65$; and zigzag reordering with neural-network-based classification and setting $q = 85$ respectively. Note that the corresponding PSNRs are 36.81 dB and 36.77 dB respectively.

Reproduced by Special Permission of Playboy magazine.
© 1972 by Playboy.

Figure 6.13 Decoded JPEG Image, Lena 512×512, $q = 65$

Reproduced by Special Permission of Playboy magazine.
© 1972 by Playboy.

Figure 6.14 Decoded Block-classified Image, Lena 512×512, $q = 85$

It has been found that the block classification produces similar results with ANNs employing weight matrices and bias vectors that have been obtained after 10000, 20000, and 30000 epochs. As a typical example, figure 6.15 shows the entropies of the runs of zero coefficients for image Lena with a spatial resolution of 512×512 pixels using zigzag reordering with neural-network-based classification for weight matrices and bias vectors obtained after 10000, 20000, and 30000 learning epochs. Although the MSE per pair reduces during learning from 0.085 after 10000 epochs to 0.065 after 30000 epochs,

the entropies are only slightly reduced for quality settings in the range [10,45], and show little difference for quality settings in the range [50,90].

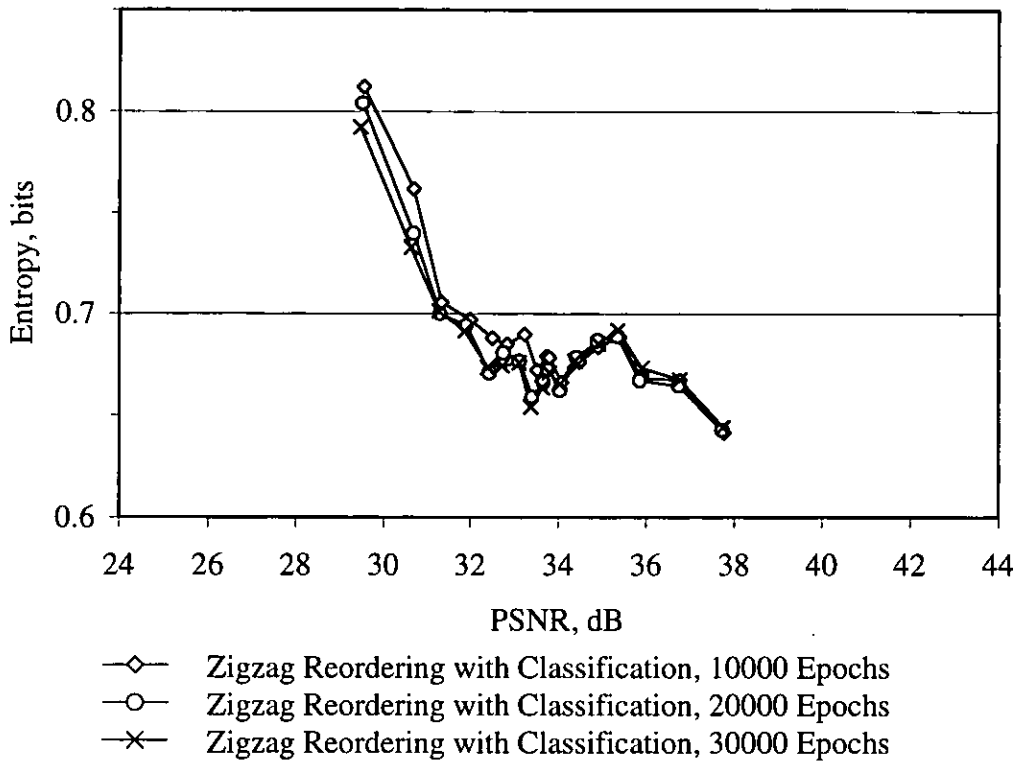


Figure 6.15 Entropy of Runs of Zero Coefficients versus Peak-signal-to-noise Ratio, Different Weight Matrices and Bias Vectors, Lena 512×512

6.5 Summary

Block classification assesses a block of transform coefficients, and generates the dimensions of a sub-block to be retained; it takes block content, i.e. the contribution of every coefficient, into account. Therefore, if the contribution of an isolated coefficient to reconstruction is found to be expendable, a significantly smaller sub-block may be retained.

The classification processing step is only required in the encoder in order to determine the sub-block dimensions for adaptive zigzag reordering. The additional processing step increases the workload of the encoder.

The classification processing step employs a feedforward ANN with 64 inputs and 64 outputs. A 64-element input vector is required for 8×8 blocks as defined by the JPEG standard for the DCT-based method. In order to homogenize network inputs, the coefficients are represented through their normalized amplitude classifications. A 64-element output vector is required to identify directly all 64 possible sub-block dimensions using a simple 1-in-64 binary code. The ANN consists of two trainable layers, i.e. hidden layer and output layer, and is trained using an error-backpropagation algorithm. During learning the neurons in both layers have log-sigmoid transfer functions. During forward propagation, the transfer function in the output layer is replaced with the competitive transfer function.

Learning is carried out in two phases each of which uses batch training. During the initial learning phase the ANN is trained for 5000 epochs on 64 idealized training pairs that correspond to the 64 possible sub-block dimensions. During the further learning phase the ANN is trained for 30000 epochs on the 64 idealized and 580 input authentic training pairs. For six of the 64 possible sub-block dimensions suitable examples have not been derived from the images.

The authentic training pairs have been generated by subjective classification of the 8×8 blocks of normalized amplitude classifications from three images. The input matrix of the training set has been built from a selection of classified blocks, and the

target matrix has been generated from the 1-in-64 codes of the corresponding sub-block dimensions.

Zigzag reordering with neural-network-based classification featuring lossy conversion produces lower entropies than standard zigzag reordering and adaptive zigzag reordering. Since some coefficients are discarded through the classification processing step, a higher quality setting is required in order to achieve the same PSNR as produced by standard zigzag reordering and adaptive zigzag reordering.

These particular weight matrices and bias vectors lead to entropies below 1 bit. Although the MSE per pair reduces during learning from 0.085 after 10000 epochs to 0.065 after 30000 epochs, the entropies are only slightly reduced for quality settings in the range [10,45], and show little difference for quality settings in the range [50,90].

Chapter 7

Conclusions and Recommendations for Further Work

7.1 Introduction

This chapter draws conclusions and provides recommendations for further work. Section 7.2 summarizes the contributions to knowledge described in this thesis. Section 7.3 offers recommendations for further research directions with respect to the contributions made.

7.2 Summary and Conclusions

Digital image compression, dating back to the late 1940s, exploits different forms of data redundancy; namely coding, interpixel, and psychovisual redundancy; in order to reduce storage and transmission requirements in digital image processing. If the reconstructed image is numerically identical to the original image, the employed compression technique is lossless. If the reconstructed image approximates the original image, the employed compression technique is lossy. A large variety of compression techniques; for example Huffman coding, run-length coding, predictive coding, transform coding, and vector quantization; has evolved over the years. The main advantage of transform coding is that it processes images in a similar manner to the human visual system.

The JPEG standard for the DCT-based method, that was aimed to boost the utilization of digital images in general-purpose computer systems, is now a well-established lossy technique that combines transform coding, quantization, run-length coding, and entropy coding. It is the combination of several processing steps that makes this technique superior to those techniques that address only a single redundancy.

Since the amount of image data being collected, processed, stored, and transmitted increases rapidly due to higher utilization, new applications, and higher standards, digital image compression remains a key technology. As the limits of techniques exploiting coding and interpixel redundancies have been reached, the move towards perceptual coding exploiting psychovisual redundancy, i.e. properties of the human visual system, is natural in attempting to reduce bit rates.

Work on artificial neural networks also dates back to the 1940s. Compared to conventional computational systems; artificial neural networks, consisting of a number of simple processing units, are massively parallel and adaptive. ANNs have the ability to learn by example. A variety of network architectures, for example feedforward networks and Kohonen self-organizing feature maps, has been developed; and feasible applications begin to emerge. The multilayer feedforward ANN trained using the error-backpropagation algorithm has attracted most interest.

The work presented in this thesis addresses aspects of coding of coefficients that are present, for example, in the JPEG standard for the DCT-based method. The statistics for entropy coding after coefficient reordering are analysed, and adaptive zigzag reordering, a novel versatile technique that achieves efficient reordering by processing variable-size rectangular sub-blocks of coefficients, is developed. Classification of blocks of DCT coefficients using a two-layer feedforward ANN prior to adaptive zigzag reordering is investigated.

The main original contributions to knowledge described within this thesis are:

- An analysis of the entropies of runs of zero coefficients for coefficient reordering along fixed and adaptive zigzag scan paths for images with different spatial

resolutions and JPEG quality settings that establishes the benefits of addressing the symbol statistics for entropy coding rather than assuming a model with increasingly probable zero coefficients. Such an analysis has not previously been published.

- The development of Boolean expressions and a binary decision tree to implement a versatile zigzag-reordering algorithm; that determines the scan paths 'on the fly', and removes the necessity to derive and provide scan paths for all required sub-block dimensions in advance. The versatile algorithm for adaptive zigzag reordering has been presented as a paper at an international symposium; see (H. J. Grosse et al. 1997c) in appendix H.
- The development of a hardware implementation of the versatile zigzag-reordering algorithm to investigate and to demonstrate the feasibility of such an implementation. The hardware implementation of the versatile zigzag-reordering algorithm has been presented as a paper at an international conference; see (H. J. Grosse et al. 1997b) in appendix H.
- The development of a coding scheme that takes the scan-path length into account to provide efficient coding of the sub-block dimensions, that need to be retained in order to traverse the zigzag scan path correctly during decoding. Such a scheme has not previously been published.
- The development of classification of blocks of transform coefficients, using a two-layer feedforward ANN, to discard expendable nonzero transform coefficients, and to determine the sub-block dimensions prior to adaptive zigzag reordering. The block classification using an ANN prior to adaptive zigzag reordering has been presented as a paper at a colloquium; see (H. J. Grosse et al. 1997a) in appendix H.

Note that the entropy-coding processing steps in the JPEG standard for the DCT-based method utilize an intermediate sequence of symbols. Since each run of zero coefficients is combined with the magnitude category of the succeeding nonzero coefficient to form a symbol, the reduction in entropy of runs of zero coefficients that has been achieved through adaptive zigzag reordering can be only partially exploited. In addition, the standard specifies a maximum length of codewords of 16 bits, that can limit the effectiveness of the Huffman coding. However, for lossless conversion using adaptive zigzag reordering and coding of sub-block dimensions, an overall reduction in bit rate of 3 % to 4 % has been achieved for the four grey-scale images with all quality settings used during informal tests. Note that two streams have been stored separately; and only one codebook has been derived from a range of images and quality settings, and used for coding of sub-block dimensions.

Zigzag reordering with neural-network-based classification further reduces the entropy of runs of the zero coefficients. However, since some coefficients are discarded through the classification processing step; naturally a higher quality setting, i.e. finer quantization of coefficients, is required in order to achieve the same objective image quality.

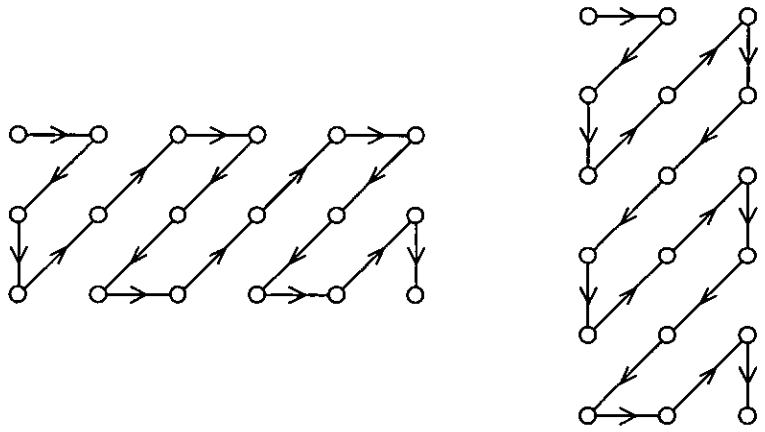
The JPEG standard for the DCT-based method provides a framework for digital compression of continuous-tone still images that provides flexibility, for example four modes of operation and user-specifiable quantization tables; but does not support adaptation to content changes within the image or its components. Although enhancements, for example image-dependent perceptually optimum quantization tables and perceptual prequantization, that maintain JPEG-compatible image data streams were suggested, the standard is inherently non-adaptive.

The work presented in this thesis takes a new approach, and supports an adaptive framework. For lossless conversion, zigzag-reordered sub-blocks must contain all nonzero coefficients; however, if coefficients are found to be expendable, smaller sub-blocks may be retained. It has been shown that adaptive zigzag reordering represents the retained coefficients more compactly. The block classification processing step allows different strategies to be implemented for the determination of the sub-block dimensions.

7.3 Recommendations for Further Work

Naturally, digital image processing moves towards higher spatial resolution and colour imaging. Although the total amount of image data increases rapidly, this development leads to lower bit rates; since increasing the number of pixels for a given image size increases the interpixel redundancy, and chrominance can be coded more efficiently than luminance; compare for example tables C.1 and C.2 in appendix C. In addition, the relative overhead per pixel caused by an overhead of fixed size, for example the quantization tables in the JPEG standard for the DCT-based method, decreases as the number of pixels increases. It is therefore suggested that further work in general encompasses colour images of increased spatial resolution.

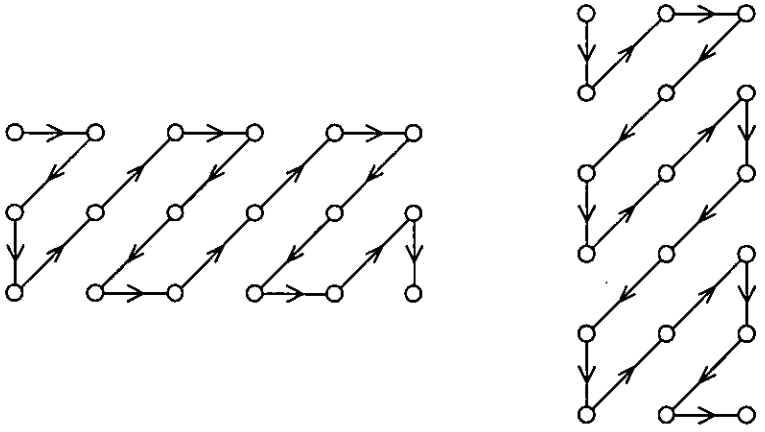
Adaptive zigzag reordering employs the versatile zigzag-reordering algorithm to generate a zigzag scan path that is tailored to the dimensions of a sub-block. However, the ratio between the row dimension and column dimension is not currently taken into account; see for example figure 7.1. Note that the direction of movement at the first position is always to the right as long as the number of columns is greater than one.



(a) (b)

Figure 7.1 Zigzag Scan Path for (a) 3×6, and (b) 6×3 Sub-blocks

The effect of different reordering algorithms on the entropy of the runs of zero coefficients could be investigated. As an example, figure 7.2 depicts two zigzag scan paths where the direction of movement at the first position, and hence the complete scan path, is influenced by the larger dimension.



(a) (b)

Figure 7.2 New Zigzag Scan Path for (a) 3×6, and (b) 6×3 Sub-blocks

Coding of the sub-block dimensions is based on the scan-path length. One set of 14 codewords is used for all scan-path lengths. Codewords are separately allocated for every scan-path length. The codebook design could be investigated with regard to the

distribution of the sub-block dimensions; see figure 4.9. Since, the distribution of the sub-block dimensions depends on the JPEG quality setting, this additional parameter could also be taken into account.

The JPEG standard requires synchronous operation, i.e. encoding and shortly delayed decoding at comparable speeds, and thus similar encoder and decoder complexity; but permits nonsynchronous mode of encoding if significant performance advantages are feasible. Note that, due to the variety of computer systems, encoders and decoders of similar complexity may operate at very different speeds. Although synchronous operation is an important feature of a general-purpose digital-image-compression scheme, an increasing number of applications relates to non-real-time one-to-many distribution of digital images via, for example, CD (compact disc) and the Internet where significant performance advantages may justify an increased encoder complexity. In recommending further work, a more detailed study into the design of encoders could be undertaken using adaptive zigzag reordering in the underlying framework.

In particular, the classification processing step, that determines the dimensions of a sub-block to be encoded, could be investigated in more detail. Additional training sets, taking subjective image quality into account, could be produced for different quality settings. Modifications to the ANN; including preprocessing, structure, and learning; could be investigated in more detail.

Bibliography

Hbk and Pbk denote hardback and paperback respectively.

ACCEL. 1989a. *Tango-PLD: reference manual.* San Diego, California, USA. ACCEL Technologies, Inc. 1989.

ACCEL. 1989b. *Tango: evaluation guide featuring Tango-PLD.* San Diego, California, USA. ACCEL Technologies, Inc. 1989.

AHMED, N., NATARAJAN, T., and RAO, K. R. 1974. Discrete cosine transform. *IEEE transactions on computers.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Jan. 1974. vol. C-23, no. 1. ISSN 0018-9340. pp. 90-93.

AMARI, Shun-Ichi. 1990. Mathematical foundations of neurocomputing. *Proceedings of the IEEE. vol.* 78, no. 9, Sep. 1990. pp. 1443-1463.

ANDERSON, James A., and ROSENFELD, Edward (eds). 1988. *Neurocomputing: foundations of research.* London, UK. Cambridge, Massachusetts, USA. The MIT Press. Jan. 1988. ISBN 0-262-01097-6.

ANDERSON, James A., PELIONISZ, A., and ROSENFELD, Edward (eds). 1990. *Neurocomputing 2: directions of research.* London, UK. Cambridge, Massachusetts, USA. The MIT Press. Apr. 1990. ISBN 0-262-51048-0.

Apple Computer. 1995. *All about ColorSync 2.0.* Cupertino, California, USA. Apple Computer, Inc. May 1995.

Apple Computer. 1996. *How to create color profiles for ColorSync 2.0.* Cupertino, California, USA. Apple Computer, Inc. 1996.

ARDUINI, Fabio, FIORAVANTI, Stefano, and GIUSTO, Daniele D. 1992. Adaptive image coding using multilayer neural networks. *In:* IEEE. 1992. *ICASSP-92: 1992 IEEE international conference on acoustics, speech, and signal processing.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Mar. 1992. vol. 2 of 5. Pbk ISBN 0-7803-0532-9. pp. 381-384. Hbk ISBN 0-7803-0533-7. Microfiche ISBN 0-7803-0534-5. 1992 IEEE international conference on acoustics, speech, and signal processing in San Francisco, California, USA, 23-26 Mar. 1992.

CAI, Defu, and ZHOU, Ming. 1992. Adaptive image compression based on backpropagation neural networks. *In:* CHEN, Su-Shing (ed.). 1992. pp. *678-683.*

CAI, Dejun, WANG, Wei, and WAN, Faguan. 1992. An unsupervised-neural-network algorithm for image compression. *In:* CHEN, Su-Shing (ed.). 1992. pp. *720-725.*

CARBONARA, Matthew R., FOWLER, James E., and AHALT, Stanley C. 1992. Compression of digital video data using artificial neural network differential vector quantization. *In:* ROGERS, Steven K. (ed.). 1992. pt 1 of 2. pp. 422-433.

CARRATO, S., and MARSI, Stefano. 1992. Parallel structure based on neural networks for image compression. *Electronics letters.* ASH, Eric A., and CLARRICOATS, Peter J. B. (eds). Stevenage, UK. The Institution of Electrical Engineers. 04 Jun. 1992. vol. 28, no. 12. ISSN 0013-5194. pp. 1152-1153.

CHEN, Su-Shing (ed.). 1992. *Neural and stochastic methods in image and signal processing.* Bellingham, Washington, USA. The International Society of Photo-Optical Instrumentation Engineers. Jul. 1992. vol. 1766. ISBN 0-8194-0939-1. Conference on neural and stochastic methods in image and signal processing in San Diego, California, USA, 20-23 Jul. 1992.

CHUA, L. O., and LIN, T. 1988. A neural-network approach to transform image coding. *International journal of circuit theory and applications.* SCANLAN, J. O. (ed.). Chichester, UK. John Wiley & Sons Ltd. Jul. 1988. vol. 16, no. 3. ISSN 0098-9886. pp. 317-324.

CLARKE, Roger J. 1985. *Transform coding of images.* London, UK. San Diego, California, USA. Academic Press and Harcourt Brace Jovanovich, Publishers. Nov. 1985. Pbk ISBN 0-12-175731-5. Hbk ISBN 0-12-175730-7.

CLARKE, Roger J. 1995. *Digital compression of still images and video.* London, UK. San Diego, California, USA. Academic Press and Harcourt Brace & Company, Publishers. 1995. Hbk ISBN 0-12-175720-X.

CONSTANTINESCU, Cornel, and STORER, James A. 1994. Improved techniques for single-pass adaptive vector quantization. *Proceedings of the IEEE. vol.* 82, no. 6, Jun. 1994. pp. 933-939.

COSMAN, Pamela C., OEHLER, Karen L., RISKIN, Eve A., and GRAY, Robert M. 1993. Using vector quantization for image processing. *Proceedings of the IEEE.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Sep. 1993. vol. 81, no. 9. ISSN 0018-9219. pp. 1326-1341.

COTTRELL, Garrison W., MUNRO, Paul, and ZIPSER, David. 1989. Image compression by backpropagation: an example of extensional programming. *In:* SHARKEY, N. E. (ed.). 1989. *Models of cognition: a review of cognitive science.* Norwood, New Jersey, USA. Ablex Publishing Corporation. Dec. 1989. ISBN 0-89391-528-9. pp. 208-240.

DEMUTH, Howard B., and BEALE, Mark. 1994. *Neural network toolbox user's guide.* Natick, Massachusetts, USA. The MathWorks, Inc. Jan. 1994.

FOSTER, Barbara. 1996. Video microscopy: where Advanced Imaging readers say we stand now. *Advanced imaging.* MAZOR, Barry (ed.). Melville, New York, USA. Advanced Imaging, a division of PTN Publishing Co. Sep. 1996. vol. 11, no. 9. ISSN 1042-0711. pp. 58, 60, and 62.

FUHRMANN, Daniel R., BARO, John A., and COX, Jerome R. 1995. Experimental evaluation of psychophysical distortion metrics for JPEG-encoded images. *Journal of electronic imaging.* DOUGHERTY, Edward R. (ed.). Bellingham, Washington, USA. Springfield, Virginia, USA. The International Society for Optical Engineering. The Society for Imaging Science and Technology. Oct. 1995. vol. 4, no. 4. ISSN 1017-9909. pp. 397-406.

GLENN, William E. 1993. Digital image compression based on visual perception and scene properties. *SMPTE journal. vol.* 102, no. 5, May 1993. pp. 392-397. 133rd SMPTE technical conference in Los Angeles, California, USA, 27 Oct. 1991.

GONZALEZ, Rafael C., and WOODS, Richard E. 1992. *Digital image processing.* 3rd ed. Wokingham, UK. Reading, Massachusetts, USA. Addison-Wesley

Publishing Co. Jun. 1992. Hbk ISBN 0-201-50803-6. 2nd ed. ISBN 0-201-11026-1.

GRANRATH, Douglas J. 1981. The role of human visual models in image processing. *Proceedings of the IEEE.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. May 1981. vol. 69, no. 5. ISSN 0018-9219. pp. 552-561.

GROSSBERG, Stephen. 1976. Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors. *Biological cybernetics.* Berlin, Germany. Springer-Verlag. 1976. vol. 23. ISSN 0340-1200. pp. 121-134. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 245-258.

GROSSBERG, Stephen. 1980. How does a brain build a cognitive code?. *Psychological review.* ESTES, William K. (ed.). Washington, District of Columbia, USA. American Psychological Association, Inc. Jan. 1980. vol. 87, no. 1. ISSN 0033-295X. pp. 1-51. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 349-400.

HABIBI, Ali. 1977. Survey of adaptive image-coding techniques. *IEEE transactions on communications. vol.* COM-25, no. 11, Nov. 1977. pp. 1275-1284.

HAGAN, Martin T., DEMUTH, Howard B., and BEALE, Mark. 1996. *Neural network design.* BARTER, Bill (ed.). Boston, Massachussetts, USA. London, UK. PWS Publishing Company, a division of International Thomson Publishing, Inc. 1996. Hbk ISBN 0-534-94332-2.

HALL, Charles F., and HALL, Ernest L. 1977. A nonlinear model for the spatial characteristics of the human visual system. *IEEE transactions on systems, man, and cybernetics.* SAGE, Andrew P. (ed.). New York, New York, USA.

The Institute of Electrical and Electronic Engineers, Inc. Mar. 1977. vol. SMC-7, no. 3. ISSN 0018-9472. pp. 161-170.

HALL, Graham, and TERRELL, Trevor James. 1987. Low-cost microprocessor-based image-processing system. Microprocessors and microsystems. Guildford, UK. Butterworth Science Ltd for Butterworth & Co. (Publishers) Ltd. Dec. 1987. vol. 11, no. 10. ISSN 0141-9331. pp. 534-540.

HE, Zhenya, and LI, Haibo. 1990. Nonlinear predictive image coding with a neural network. *In:* IEEE. 1990. *ICASSP-90: 1992 IEEE international conference on acoustics, speech, and signal processing.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Apr. 1990. vol. 2 of 5. pp. 1009-1012. 1990 IEEE international conference on acoustics, speech, and signal processing in Albuquerque, New Mexico, USA, 03-06 Apr. 1990.

HEBB, Donald O. 1949. *The organization of behaviour.* New York, New York, USA. John Wiley & Sons, Inc. 1949. pp. xi-xix, and 60-78 also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 45-56.

HOFFMANN, Norbert. 1993. *Kleines Handbuch neuronale Netze: anwendungsorientiertes Wissen zum Lernen und Nachschlagen.* Wiesbaden, Germany. Vieweg Publishing. 1993. Hbk ISBN 3-528-05239-2.

HOPFIELD, John J. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America.* Washington, District of Columbia, USA. National Academy of Sciences of the United States of America. Apr. 1982. vol. 79, no. 8. ISSN 0027-8424. pp. 2554-2558. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 460-464.

HOPFIELD, John J., and TANK, David W. 1985. 'Neural' computation of decisions in optimization problems. *Biological cybernetics.* Berlin, Germany. Springer-Verlag. 1985. vol. 52, no. 3. ISSN 0340-1200. pp. 141-152.

HOWARD, Paul G., and VITTER, Jeffrey Scott. 1994. Arithmetic coding for data compression. *Proceedings of the IEEE. vol.* 82, no. 6, Jun. 1994. pp. 857-865.

HUFFMAN, David A. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE.* GOLDSMITH, Alfred N. (ed.). New York, New York, USA. The Institute of Radio Engineers, Inc. Sep. 1952. vol. 40, no. 9. pp. 1098-1101.

HUSH, Don R., and HORNE, Bill G. 1993. Progress in supervised neural networks. *IEEE signal processing magazine.* WAKEFIELD, Greg H. (ed.). New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Jan. 1993. vol. 10, no. 1. ISSN 1053-5888. pp. 8-39.

HUTCHINSON, Robert A., and WELSH, W. J. 1989. Comparison of neural networks and conventional techniques for feature location in facial images. *In:* IEE. 1989. *First IEE international conference on artificial neural networks.* London, UK. The Institution of Electrical Engineers. Oct. 1989. vol. 313. ISBN 0-85296-388-2. pp. 201-205. First IEE international conference on artificial neural networks in London, UK, 16-18 Oct. 1989.

IEEE. 1994. *ICASSP-94: 1994 IEEE international conference on acoustics, speech, and signal processing.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Apr. 1994. vols 1-6. Pbk ISBN 0-7803-1775-0. Hbk ISBN 0-7803-1776-9. Microfiche ISBN 0-7803-1777-7. 1994 IEEE international conference on acoustics,

speech, and signal processing in Adelaide, South Australia, Australia, 19-22 Apr. 1994.

IEEE transactions on communications. vol. COM-25, no. 11, Nov. 1977, monthly. New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. ISSN 0090-6778.

IEEE transactions on consumer electronics. LUPLOW, Wayne C. (ed.). vol. 38, no. 1, Feb. 1992, quarterly. New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. ISSN 0098-3063.

Independent JPEG Group. 1996. *The Independent JPEG Group's software: C source code, release 6a.* [Online] Available ftp://ftp.simtel.net/pub/simtelnet/msdos/graphics/jpegsr6a.zip. 07 Feb. 1996.

ISAACS, Alan (ed.). 1997. *The Macmillan encyclopedia.* 1997 edition. London, UK. Macmillan Reference Books: a division of Macmillan Publishers Ltd. 1997. Hbk ISBN 0-333-66296-2.

ISO/IEC 10918-1:1994. *Digital compression and coding of continuous-tone still images, part 1: requirements and guidelines.* Geneva, Switzerland. International Organization for Standardization. 15 Feb. 1994.

ISO/IEC 10918-2:1995. *Digital compression and coding of continuous-tone still images, part 2: compliance testing.* Geneva, Switzerland. International Organization for Standardization. 1995.

ISO/IEC DIS 10918-3. *Digital compression and coding of continuous-tone still images, part 3: extensions.* Geneva, Switzerland. International Organization for Standardization.

ISO/IEC DIS 10918-4. *Digital compression and coding of continuous-tone still images, part 4: registration procedures for JPEG profile, APPn marker, and SPIFF*

profile ID marker. Geneva, Switzerland. International Organization for Standardization.

JAIN, Anil K. 1981. Image data compression: a review. *Proceedings of the IEEE.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Mar. 1981. vol. 69, no. 3. ISSN 0018-9219. pp. 349-391.

JAYANT, Nikil, JOHNSTON, James D., and SAFRANEK, Robert J. 1993. Signal compression based on models of human perception. *Proceedings of the IEEE.* FAIR, Richard B. (ed.). Oct. 1993. vol. 81, no. 10. ISSN 0018-9219. pp. 1385-1422.

KANDEL, Eric R., SCHWARTZ, J. H., and JESSELL, Thomas M. 1991. *Principles of neural science.* 3rd ed. Appleton & Lange. 1991, Mar. 1993. ISBN 0-8385-8068-8.

KARUNASEKERA, Shanika A., and KINGSBURY, Nick G. 1995. A distortion measure for blocking artifacts in images based on human visual sensitivity. GIROD, Bernd (ed.). *IEEE transactions on image processing.* MUNSON, D. C. (ed.). New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Jun. 1995. vol. 4, no. 6. ISSN 1057-7149. pp. 713-724.

KLIMASAUSKAS, Casimir C. 1990. Neural networks and image processing: finding edges only a human eye can see. *Dr. Dobb's journal.* Redwood City, California, USA. M&T Publishing, Inc. Apr. 1990. vol. 15, no. 4. ISSN 1044-789X. pp. 77-82, and 114-116.

KOHONEN, Teuvo. 1972. Correlation matrix memories. *IEEE transactions on computers.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Apr. 1972. vol. C-21, no. 4. ISSN 0018-9340.

pp. 353-359. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 174-180.

KUNT, Murat, IKONOMOPOULOS, Athanassios, and KOCHER, Michel. 1985. Second-generation image-coding techniques. *Proceedings of the IEEE.* MEDITCH, J. S. (ed.). New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Apr. 1985. vol. 73, no. 4. ISSN 0018-9219. pp. 549-574.

LABIT, C., and MARESCQ, J. P. 1986. Image coding by vector quantization in a transformed domain. *In:* KUNT, Murat, and HUANG, T. S. (eds). 1986. *Image coding [1985].* Bellingham, Washington, USA. The International Society of Photo-Optical Instrumentation Engineers. 1986. vol. 594. ISBN 0-89252-629-7. pp. 106-110. Conference on image coding in Cannes, France, 04-06 Dec. 1985.

LASHLEY, Karl S. 1950. In search of the engram. *Society of Experimental Biology symposium, no. 4: psychological mechanisms in animal behaviour.* Cambridge, UK. Cambridge University Press. 1950. pp. 454-455, 468-473, and 477-480. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 59-64.

Lattice. 1996. *Lattice data book. Lattice.* 1996. *Lattice ISP Encyclopedia CD-ROM.* Hillboro, Oregon, USA. Lattice Semiconductor Corporation. 1996.

Lattice. 1997. *GAL16V8.* [Online] Available http://www.latticesemi.com/cgi-bin/lattice_list_files. Jan. 1997.

LEDLEY, Robert S. 1993. The processing of medical images in compressed format. *In:* ACHARYA, Raj S., and GOLDGOF, Dmitry B. (eds). 1993. *Biomedical image processing and biomedical visualization.* Bellingham, Washington,

USA. The International Society of Photo-Optical Instrumentation Engineers. Feb. 1993. vol. 1905, pt 1 of 2. ISBN 0-8194-1138-8. pp. 677-687. Conference on biomedical image processing and biomedical visualization in San Jose, California, USA, 01-04 Feb. 1993.

LÉGER, Alain, OMACHI, Takao, and WALLACE, Gregory K. 1991. JPEG still picture compression algorithm. *Optical engineering.* Bellingham, Washington, USA. The International Society of Photo-Optical Instrumentation Engineers. Jul. 1991. vol. 30, no. 7. ISSN 0091-3286. pp. 947-954.

LEGGE, Gordon E., and FOLEY, John M. 1980. Contrast masking in human vision. *Journal of the Optical Society of America.* GOODMAN, Joseph W. (ed.). New York, New York, USA. American Institute of Physics, Inc. for Optical Society of America. Dec. 1980. vol. 70, no. 12. ISSN 0030-3941. pp. 1458-1471.

LIMB, John O. 1979. Distortion criteria of the human viewer. *IEEE transactions on systems, man, and cybernetics.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Dec. 1979. vol. SMC-9, no. 12. ISSN 0018-9472. pp. 778-793.

LIPPMANN, Richard P. 1987. An introduction to computing with neural nets. *IEEE ASSP magazine.* ETTER, Delores M. (ed.). New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Apr. 1987. vol. 4, no. 2. ISSN 0740-7467. pp. 4-22.

LIU, Hui, and YUN, David Y. Y. 1992. Competitive learning algorithms for image coding. *In:* ROGERS, Steven K. (ed.). 1992. pt 1 of 2. pp. 408-417.

LLOYD, Stuart P. 1982. Least squares quantization in PCM. *IEEE transactions on information theory.* GRAY, Robert M. (ed.). New York, New York, USA.

The Institute of Electrical and Electronic Engineers, Inc. Mar. 1982. vol. IT-28, no. 2. ISSN 0018-9448. pp. 129-137.

LOHSCHELLER, Herbert. 1984. A subjectively adapted image communication system. *IEEE transactions on communications.* LIMB, John O. (ed.). New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Dec. 1984. vol. COM-32, no. 12. ISSN 0090-6778. pp. 1316-1322.

LU, Cheng Chang, and SHIN, Yong Ho. 1992. A neural-network-based image compression system. *IEEE transactions on consumer electronics. vol.* 38, no. 1, Feb. 1992. pp. 25-29.

LUKAS, Frank X. J., and BUDRIKIS, Zigmantas L. 1982. Picture-quality prediction based on a visual model. *IEEE transactions on communications.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Jul. 1982. vol. COM-30, no. 7. ISSN 0090-6778. pp. 1679-1692.

LUND, Arnold M. 1993. The influence of video image size and resolution on viewing-distance preferences. *SMPTE journal. vol.* 102, no. 5, May 1993. pp. 406-415.

LUTTRELL, S. P. 1989. Image compression using a multilayer neural network. *Patter recognition letters.* BACKER, E., and GELSEMA, E. S. (eds). Amsterdam, The Netherlands. Elsevier Science Publishers B.V. for International Association for Pattern Recognition. Jul. 1989. vol. 10, no. 1. ISSN 0167-8655. pp. 1-7.

MACQ, Benoit, MATTAVELLI, M., VAN CALSTER, O., VAN DER PLANCKE, E., COMES, S., and LI, W. 1994. Image visual quality restoration by cancellation of the unmasked noise. *In:* IEEE. 1994. vol. 5 of 6. pp. 53-56.

MALSBURG, Christoph von der. 1973. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik.* 1973. vol. 14. ISSN 0023-5946. pp. 85-100. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 212-228.

MANNOS, James L., and SAKRISON, David J. 1974. The effects of a visual fidelity criterion on the encoding of images. *IEEE transactions on information theory.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Jul. 1974. vol. IT-20, no. 4. ISSN 0018-9448. pp. 525-536.

MARANGELLI, B. 1991. A vector quantizer with minimum visible distortion. *IEEE transactions on signal processing.* WHEELER, Pierce (ed.). New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Dec. 1991. vol. 39, no. 12. ISSN 1053-587X. pp. 2718-2721.

MARSI, Stefano, RAMPONI, Giovanni, and SICURANZA, Giovanni, L. 1991. Improved neural structures for image compression. *In:* IEEE. 1991. *ICASSP-91: 1991 international conference on acoustics, speech, and signal processing.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. May 1991. vol. 4 of 5. Pbk ISBN 0-7803-0003-3. pp. 2821-2824. Hbk ISBN 0-7803-0004-1. Microfiche ISBN 0-7803-0005-X. 1991 IEEE international conference on acoustics, speech, and signal processing in Toronto, Ontario, Canada, 14-17 May 1991.

MATHER, Paul M. 1987. *Computer processing of remotely sensed images: an introduction.* Chichester, UK. John Wiley & Sons Ltd for Paul M. Mather.

1987, Dec. 1989, 1994. Pbk ISBN 0-471-92653-1. Hbk ISBN 0-471-90648-4.

MathWorks. 1994. *MATLAB version 4.2.* Natick, Massachusetts, USA. The MathWorks, Inc. Oct. 1994.

MATTAVELLI, M., BRUYNDONCKX, O., COMES, S., and MACQ, Benoit. 1995. Post-processing of coded images by neural-network cancellation of unmasked noise. *Neural processing letters.* BLAYO, François, and VERLEYSEN, Michel (eds). Brussels, Belgium. D facto publications s.a. for Neurosciences et Sciences de l'Ingénieur Association. Mar. 1995. vol. 2, no. 2. ISSN 1370-4621. pp. 18-22.

MAX, Joel. 1960. Quantizing for minimum distortion. *IRE transactions on information theory.* ZADEH, Lotfi A. (ed.). New York, New York, USA. The Institute of Radio Engineers, Inc. Mar. 1960. vol. IT-6, no. 1. pp. 7-12.

MCCLELLAND, James L., and RUMELHART, David E. 1981. An interactive activation model of context effects in letter perception, pt 1: an account of basic findings. *Psychological review.* Washington, District of Columbia, USA. American Psychological Association, Inc. Sep. 1981. vol. 88, no. 9. ISSN 0033-295X. pp. 375-407. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 404-436.

MCCLELLAND, James L., and RUMELHART, David E. 1988. *Explorations in parallel distributed processing: a handbook of models, programs, and exercises.* London, UK. Cambridge, Massachusetts, USA. The MIT Press. Apr. 1988. Pbk ISBN 0-262-63113-X.

MCCLELLAND, James L., RUMELHART, David E., and the PDP Research Group. 1986. *Parallel distributed processing: explorations in the microstructure of*

cognition, vol. 2: psychological and biological models. London, UK. Cambridge, Massachusetts, USA. The MIT Press for Massachusetts Institute of Technology. 1986. vol. 2 of 2. Pbk ISBN 0-262-63110-5. Hbk ISBN 0-262-13218-4. vols1-2 Pbk ISBN 0-262-63112-1. vols 1-2 Hbk ISBN 0-262-18123-1.

MCCULLOCH, Warren S., and PITTS, Walter. 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biophysics.* 1943. vol. 5. pp. 115-133. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 18-28.

MCFARLANE, Maynard D. 1972. Digital pictures fifty years ago. *Proceedings of the IEEE. vol.* 60, no. 7, Jul. 1972. pp. 768-770.

MCLAREN, David L., and NGUYEN, D. Thong. 1991. Removal of subjective redundancy from DCT-coded images. *IEE proceedings I: communications, speech and vision.* AMIR-ALIKHANI, H., and LODGE, N. K. (eds). Stevenage, UK. The Institution of Electrical Engineers. Oct. 1991. vol. 138, pt I, no. 5. ISSN 0956-3776. pp. 345-350.

MILLER, Ade S., BLOTT, B. H., and HAMES, T. K. 1992. Review of neural-network applications in medical imaging and signal processing. *Medical & biological engineering & computing.* Stevenage, UK. Peter Peregrinus Ltd for Federation for Medical and Biological Engineering. Sep. 1992. vol. 30, no. 5. ISSN 0140-0118. pp. 449-464.

MINSKY, Marvin, and PAPERT, Seymour. 1969. *Perceptrons: an introduction to computational geometry.* London, UK. Cambridge, Massachusetts, USA. The MIT Press. 1969. 1988. 2nd ed. ISBN 0-262-63111-3. pp. 1-20, and 73 also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 161-170.

MITCHELL, H. B., and DORFAN, M. 1992. Block-truncation coding using Hopfield neural network. *Electronics letters.* ASH, Eric A., and CLARRICOATS, Peter J. B. (eds). Stevenage, UK. The Institution of Electrical Engineers. 05 Nov. 1992. vol. 28, no. 23. ISSN 0013-5194. pp. 2144-2145.

NELSON, Mark. 1992. *The data compression book.* New York, New York, USA. M&T Publishing, Inc. 1992. Pbk ISBN 1-55851-216-0.

NETRAVALI, Arun N. 1977. On quantizers for DPCM coding of picture signals. *IEEE transactions on information theory.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. May 1977. vol. IT-23, no. 3. ISSN 0018-9448. pp. 360-370.

NETRAVALI, Arun N., and LIMB, John O. 1980. Picture coding: a review. *Proceedings of the IEEE.* FREITAG, Harlow (ed.). New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Mar. 1980. vol. 68, no. 3. ISSN 0018-9219. pp. 366-407.

NETRAVALI, Arun N., and PRASADA, Birendra. 1977. Adaptive quantization of picture signals using spatial masking. *Proceedings of the IEEE.* WADE, Glen (ed.). New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Apr. 1977. vol. 65, no. 4. ISSN 0018-9219. pp. 536-548.

Network: computation in neural systems. AMIT, D. J. (ed.). vol. 5, no. 4, Nov. 1994, quarterly. Bristol, UK. Institute of Physics Publishing. ISSN 0954-898X.

NGAN, King N., LEONG, Kin S., and SINGH, H. 1989. Adaptive cosine transform coding of images in perceptual domain. *IEEE transactions on acoustics, speech, and signal processing.* WHEELER, Pierce (ed.). New York, New

York, USA. The Institute of Electrical and Electronic Engineers, Inc. Nov. 1989. vol. 37, no. 11. ISSN 0096-3518. pp. 1743-1750.

NIEMANN, Heinrich, and WU, Jian-Kang. 1993. Neural-network adaptive image coding. *IEEE transactions on neural networks.* MARKS, Robert J. (ed.). New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Jul. 1993. vol. 4, no. 4. ISSN 1045-9227. pp. 615-627.

NIGHTINGALE, Charles, and HUTCHINSON, Robert A. 1990. Artificial neural nets and their application to image processing. *British Telecom technology journal.* London, UK. Chapman & Hall for British Telecommunications plc. Jul. 1990. vol. 8, no. 3. ISSN 0265-0193. pp. 81-93.

NILL, Norman B. 1985. A visual-model-weighted cosine transform for image compression and quality assessment. *IEEE transactions on communications.* LESH, J. R. (ed.). New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Jun. 1985. vol. COM-33, no. 6. ISSN 0090-6778. pp. 551-557.

NILSON, Nils J. 1965. *Learning machines: foundations of trainable pattern classification systems.* New York, New York, USA. McGraw-Hill. 1965.

PAL, Nikhil R., and PAL, Sankar K. 1993. A review on image-segmentation techniques. *Pattern recognition.* Tarrytown, New York, USA. Pergamon Press, Inc. for Pattern Recognition Society. Sep. 1993. vol. 26, no. 9. ISSN 0031-3203. pp. 1277-1294.

PANCHANATHAN, S., YEAP T. H., and PILACHE, B. 1992. A neural network for image compression. *In:* ROGERS, Steven K. (ed.). 1992. pt 1 of 2. pp. 376-385.

PARIKH, Jo Ann, DAPONTE, John S., DAMODARAN, Meledath, and SHERMAN, Porter. 1990. Application of neural networks to pattern-recognition problems in remote-sensing and medical imagery. *In:* ROGERS, Steven K. (ed.). 1990. *Applications of artificial neural networks.* Bellingham, Washington, USA. The International Society of Photo-Optical Instrumentation Engineers. Apr. 1990. vol. 1294. ISBN 0-8194-0345-8. pp. 146-160. First conference on applications of artificial neural networks in Orlando, Florida, USA, 18-20 Apr. 1990.

PENNEBAKER, William B., and MITCHELL, Joan L. 1992. *JPEG still image data compression standard.* London, UK. New York, New York, USA. Van Nostrand Reinhold. Dec. 1992. Hbk ISBN 0-442-01272-1.

Proceedings of the IEEE. ROWE, Joseph E. (ed.). vol. 60, no. 7, Jul. 1972, monthly. New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. ISSN 0018-9219.

Proceedings of the IEEE. SCHELL, A. C. (ed.). vol. 78, no. 9, Sep. 1990, monthly. New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. ISSN 0018-9219.

Proceedings of the IEEE. WATSON, George F. (ed.). vol. 82, no. 6, Jun. 1994, monthly. New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. ISSN 0018-9219.

QIU, Guoping, VARLEY, Martin Roy, and TERRELL, Trevor James. 1991. Improved block-truncation coding using Hopfield neural network. *Electronics letters.* ASH, Eric A., and CLARRICOATS, Peter J. B. (eds). Stevenage, UK. The Institution of Electrical Engineers. 10 Oct. 1991. vol. 27, no. 21. ISSN 0013-5194. pp. 1924-1926.

QIU, Guoping, VARLEY, Martin Roy, and TERRELL, Trevor James. 1993a. Variable bit-rate block-truncation coding for image compression using Hopfield neural networks. *In:* IEE. 1993. *Third international conference on artificial neural networks.* London, UK. The Institution of Electrical Engineers. May 1993. vol. 372. ISBN 0-85296-573-7. pp. 233-239. Third international conference on artificial neural networks in Brighton, UK, 25-27 May 1993.

QIU, Guoping, VARLEY, Martin Roy, and TERRELL, Trevor James. 1993b. Image compression by edge-pattern learning using multilayer preceptrons. *Electronics letters.* ASH, Eric A., and CLARRICOATS, Peter J. B. (eds). Stevenage, UK. The Institution of Electrical Engineers. 01 Apr. 1993. vol. 29, no. 7. ISSN 0013-5194. pp. 601-603.

RODRÍGUEZ, Jeffrey J., and YANG, Christopher C. 1994. Effects of luminance quantization error on color image processing. SEZAN, M. Ibrahim (ed.). *IEEE transactions on image processing.* MUNSON, D. C. (ed.). New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. 11 Nov. 1994. vol. 3, no. 6. ISSN 1057-7149. pp. 850-854.

ROGERS, Steven K. (ed.). 1992. *Applications of artificial neural networks III.* Bellingham, Washington, USA. The International Society of Photo-Optical Instrumentation Engineers. Apr. 1992. vol. 1709, pts 1-2. ISBN 0-8194-0874-3. Third annual international conference on applications of artificial neural networks in Orlando, Florida, USA, 21-24 Apr. 1992.

ROMANIUK, Steve G. 1994. Theoretical results for applying neural networks to lossless image compression. *Network: computation in neural systems. vol.* 5, no. 4, Nov. 1994. pp. 583-597.

ROSENBLATT, Frank. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review.* Washington, District of Columbia, USA. American Psychological Association, Inc. 1958. vol. 65. ISSN 0033-295X. pp. 386-408. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 92-114.

ROSENBLATT, Frank. 1959. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms.* New York, New York, USA. Spartan Books. 1959.

RUDERMAN, Daniel L. 1994. The statistics of natural images. *Network: computation in neural systems. vol.* 5, no. 4, Nov. 1994. pp. 517-548.

RUMELHART, David E., HINTON, Geoffrey E., and WILLIAMS, Ronald J. 1986a. Learning internal representations by error propagation. *In:* RUMELHART, David E., MCCLELLAND, James L., and the PDP Research Group. 1986. pp. *318-362.* Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 696-700.

RUMELHART, David E., HINTON, Geoffrey E., and WILLIAMS, Ronald J. 1986b. Learning representations by back-propagating errors. *Nature.* MADDOX, John (ed.). London, UK. Macmillan Magazines Ltd. 09 Oct. 1986. vol. 323, no. 6088. ISSN 0028-0836. pp. 533-536. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 675-695.

RUMELHART, David E., MCCLELLAND, James L., and the PDP Research Group. 1986. *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations.* London, UK. Cambridge, Massachusetts, USA. The MIT Press for Massachusetts Institute of Technology. Sep. 1986. vol. 1 of 2. Pbk ISBN 0-262-68053-X. Hbk ISBN 0-262-18120-7. vols1-2 Pbk ISBN 0-262-63112-1. vols 1-2 Hbk ISBN 0-262-18123-1.

SACHS, Murray B., NACHMIAS, Jacob, and ROBSON, John G. 1971. Spatial-frequency channels in human vision. *Journal of the Optical Society of America.* MACADAM, David L. (ed.). New York, New York, USA. American Institute of Physics, Inc. for Optical Society of America. Sep. 1971. vol. 61, no. 9. ISSN 0030-3941. pp. 1176-1186.

SAKRISON, David J. 1977. On the role of the observer and a distortion measure in image transmission. *IEEE transactions on communications. vol.* COM-25, no. 11, Nov. 1977. pp. 1251-1267.

SEDGEWICK, Robert. 1992. *Algorithmen in C.* Translated from English [Algorithms in C, 1990, ISBN 0-201-51425-7]. Bonn, Germany. Munich, Germany. Paris, France. Addison-Wesley. 1992. Pbk ISBN 3-89319-669-2.

SEJNOWSKI, Terrence J., and ROSENBERG, Charles R. 1986. *Nettalk: a parallel network that learns to read aloud, technical report JHU/EECS-86/01.* Baltimore, Maryland, USA. Department of Electrical Engineering and Computer Science, John Hopkins University. 1986. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 663-672.

SHANNON, Claude E. 1948a. A mathematical theory of communication [1/2]. *The Bell system technical journal.* KING, R. W., and PERRINE, J. O. (eds). New York, New York, USA. American Telephone and Telegraph Company. Jul. 1948. vol. XXVII, no. 3. pp. 379-423.

SHANNON, Claude E. 1948b. A mathematical theory of communication [2/2]. *The Bell system technical journal.* KING, R. W., and PERRINE, J. O. (eds). New York, New York, USA. American Telephone and Telegraph Company. Oct. 1948. vol. XXVII, no. 4. pp. 623-656.

SICURANZA, Giovanni L., RAMPONI, Giovanni, and MARSI, Stefano. 1990. Artificial neural networks for image compression. *Electronics letters.* ASH, Eric A., and CLARRICOATS, Peter J. B. (eds). Stevenage, UK. The Institution of Electrical Engineers. 29 Mar. 1990. vol. 26, no. 7. ISSN 0013-5194. pp. 477-479.

SID-AHMED, Maher A. 1995. *Image processing: theory, algorithms, and architectures.* International ed. New York, New York, USA. McGraw-Hill, Inc. 1995. Hbk ISBN 0-07-057240-2.

SMPTE journal. FRIEDMAN, Jeffrey (ed.). vol. 102, no. 5, May 1993, monthly. White Plains, New York, USA. Society of Motion Picture and Television Engineers, Inc. ISSN 0036-1682.

SONKA, Milan, HLAVAC, Vaclav, and BOYLE, Roger. 1993. *Image processing, analysis and machine vision.* London, UK. Chapman & Hall for Milan Sonka, Vaclav Hlavac, and Roger Boyle. 1993. Pbk ISBN 0-412-45570-6.

STOCKHAM, Thomas G. 1972. Image processing in the context of a visual model. *Proceedings of the IEEE. vol.* 60, no. 7, Jul. 1972. pp. 828-842.

TREMEAU, A., CALONNIER, M., and LAGET, B. 1994. Color quantization error in terms of perceived image quality. *In:* IEEE. 1994. vol. 5 of 6. pp. 93-96.

VITTER, Jeffrey Scott. 1987. Design and analysis of dynamic Huffman codes. *Journal of the Association for Computing Machinery.* ROSENKRANTZ, Daniel J. (ed.). New York, New York, USA. Association for Computing Machinery, Inc. Oct. 1987. vol. 34, no. 4. ISSN 0004-5411. pp. 825-845.

WALKER, N. P., EGLEN, S. J., and LAWRENCE, B. A. 1994. Image compression using neural networks. *GEC journal of research.* WALKDEN, A. J. (ed.).

Chelmsford, UK. The General Electric Company plc. 1994. vol. 11, no. 2. ISSN 0264-9187. pp. 66-75.

WALLACE, Gregory K. 1990. Overview of the JPEG (ISO/CCITT) still image compression standard. *In:* PENNINGTON, K. S., and MOORHEAD, R. J. (eds). 1990. *Image processing algorithms and techniques.* Bellingham, Washington, USA. The International Society of Photo-Optical Instrumentation Engineers. Feb. 1990. vol. 1244. ISBN 0-8194-0291-5. pp. 220-233. Conference on image processing algorithms and techniques in Santa Clara, California, USA, 12-14 Feb. 1990.

WALLACE, Gregory K. 1991. The JPEG still picture compression standard. *Communications of the ACM.* MAURER, James (ed.). New York, New York, USA. Association for Computing Machinery, Inc. Apr. 1991. vol. 34, no. 4. ISSN 0001-0782. pp. 30-44.

WALLACE, Gregory K. 1992. The JPEG still picture compression standard. *IEEE transactions on consumer electronics. vol.* 38, no. 1, Feb. 1992. pp. xviii-xxxiv.

WATSON, Andrew B. 1993a. DCT quantization matrices visually optimized for individual images. *In:* ALLEBACH, Jan P., and ROGOWITZ, Bernice E. (eds). 1993. *Human vision, visual processing, and digital display IV.* Bellingham, Washington, USA. The International Society of Photo-Optical Instrumentation Engineers. Feb. 1993. vol. 1913. ISBN 0-8194-1146-9. pp. 202-216. Fourth conference on human vision, visual processing, and digital display in San Jose, California, USA, 01-04 Feb. 1993.

WATSON, Andrew B. 1993b. Visually optimal DCT quantization matrices for individual images. *In:* IEEE. 1993. *Data compression conference 1993.*

STORER, James A., and COHN, Martin (eds). Los Alamitos, USA. IEEE Computer Society Press for the Institute of Electrical and Electronic Engineers, Inc. Mar. 1993. Hbk ISBN 0-8186-3392-1. pp. 178-187. Microfiche ISBN 0-8186-3391-3. Third data compression conference in Snowbird, Utah, USA, 30 Mar. - 02 Apr. 1993.

WERBLIN, Frank S. 1973. The control of sensitivity in the retina. *Scientific american.* FLANAGAN, Dennis (ed.). New York, New York, USA. Scientific American. Jan. 1973. vol. 228, no. 1. ISSN 0036-8733. pp. 70-79.

WERBOS, Paul J. 1974. *Beyond regression: new tools for prediction and analysis in the behavioral science, PhD thesis in applied mathematics.* Cambridge, Massachusetts, USA. Harvard University. 1974.

WIDROW, Bernard, and HOFF, Marcian E. 1960. Adaptive switching circuits. *IRE WESCON convention record.* New York, New York, USA. Institute of Radio Engineers. Aug. 1960. pt 4. pp. 96-104. Also in: ANDERSON, J. A., and ROSENFELD, E. (eds). 1988. pp. 126-134.

WIDROW, Bernard, and LEHR, Michael A. 1990. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE. vol.* 78, no. 9, Sep. 1990. pp. 1415-1442.

WITTEN, Ian H., NEAL, Radford M., and CLEARY, John G. 1987. Arithmetic coding for data compression. *Communications of the ACM.* DENNING, Peter J. (ed.). New York, New York, USA. Association for Computing Machinery, Inc. Jun. 1987. vol. 30, no. 6. ISSN 0001-0782. pp. 520-540.

ZELL, Andreas. 1994. *Simulation Neuronaler Netze.* Bonn, Germnay. Munich, Germany. Paris, France. Addison-Wesley. 1994. Hbk ISBN 3-89319-554-8.

ZIV, Jacob, and LEMPEL, Abraham. 1977. A universal algorithm for sequential data compression. *IEEE transactions on information theory.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. May 1977. vol. IT-23, no. 3. ISSN 0018-9448. pp. 337-343.

ZIV, Jacob, and LEMPEL, Abraham. 1978. Compression of individual sequences via variable-rate coding. *IEEE transactions on information theory.* New York, New York, USA. The Institute of Electrical and Electronic Engineers, Inc. Sep. 1978. vol. IT-24, no. 5. ISSN 0018-9448. pp. 530-536.

Appendices

A Landsat Image Size Worked Example

Landsat-4 and Landsat-5 carried two sensor types producing images of similar structure. The general equation for calculating the size of an image, S, is

$$S = L\,M \sum_{i=1}^{I} R_i \tag{A.1}$$

where L is the number of horizontal pixels, i.e. number of pixels per scan line; M is the number of vertical pixels, i.e. number of scan lines; R_i is the resolution of spectral band i; and I is the number of spectral bands.

For identical resolution R of all spectral bands, equation A.1 reduces to:

$$S = L\,M\,R\,I \tag{A.2}$$

Table A.1 summarizes the specification for Landsat-4 and -5 MSS and TM images; see (P. M. Mather 1987, p. 84).

	MSS	TM
Pixels per Scan Line	3600	6900
Scan Lines	2286	5700
Band Resolution	6 bits	8 bits
Number of Bands	4	7

Table A.1 Specification for Landsat-4 and -5 MSS and TM Images

The size of an MSS image is therefore

$$S_{MSS\,\min} = 3600 \times 2286 \times 6 \times 4 \text{ bits} \approx 23.5 \text{ MB} \tag{A.3}$$

However, since storage locations of computer memory are usually organized in multiples of bytes, the realistic size of an MSS image file is

$$S_{MSS} = 3600 \times 2286 \times 8 \times 4 \text{ bits} \approx 31.4 \text{ MB} \tag{A.4}$$

The size of a TM image is

$$S_{TM} = 6900 \times 5700 \times 8 \times 7 \text{ bits} \approx 262.6 \text{ MB} \tag{A.5}$$

B Huffman Tree Design Worked Example

B.1 Introduction

This appendix provides a worked example of the design of a Huffman tree for an 8-level image of size 8×8.

B.2 Design Procedure

Figure B.1 depicts an 8-level image of size 8×8. Table B.1 presents the frequency of occurrence for every symbol.

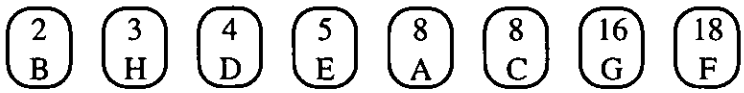
H H G G G B G G
H G C C C B G G
G C C C C C G E
G F F F F F E E
G F F F F F E E
G F F D F F G D
G F F D F F G D
A A A A A A A A

Figure B.1 8-level Image

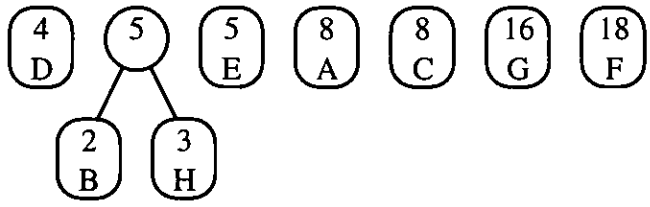
Symbol	A	B	C	D	E	F	G	H
Frequency of Occurrence	8	2	8	4	5	18	16	3

Table B.1 Symbol Distribution of 8-level Image

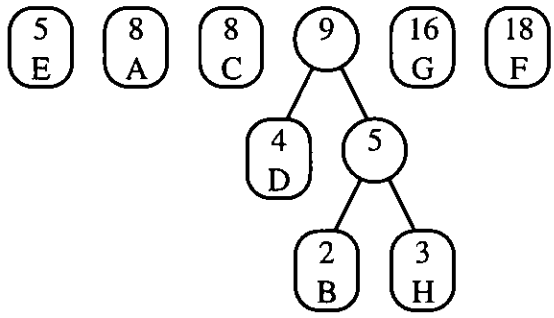
Figure B.2 depicts the generation of an appropriate Huffman tree. Using the compound node with a weight of 5 rather than symbol E with a weight of 5 for generating the second parent node introduces an additional level; see figure B.2 c).



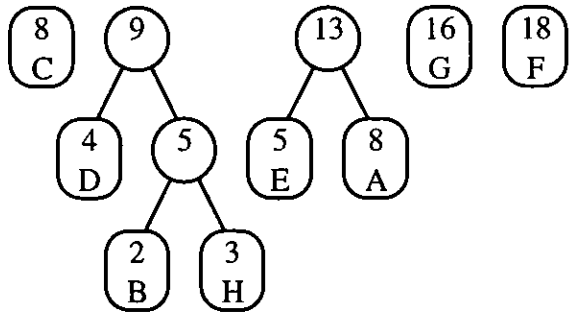
a) Symbols Arranged in Increasing Frequency of Occurrence



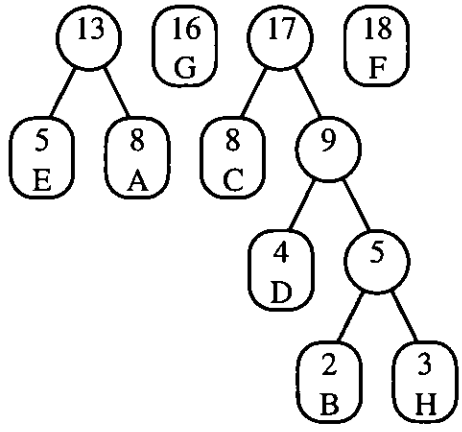
b) First Parent Node Generated



c) Second Parent Node Generated

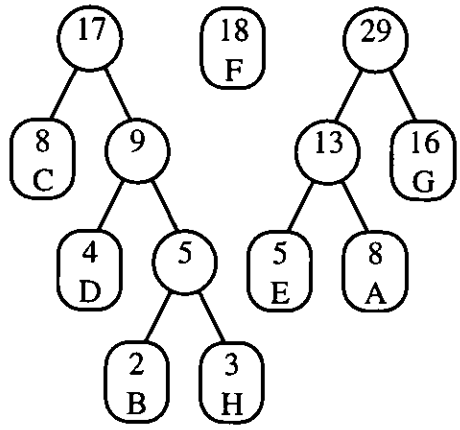


d) Third Parent Node Generated

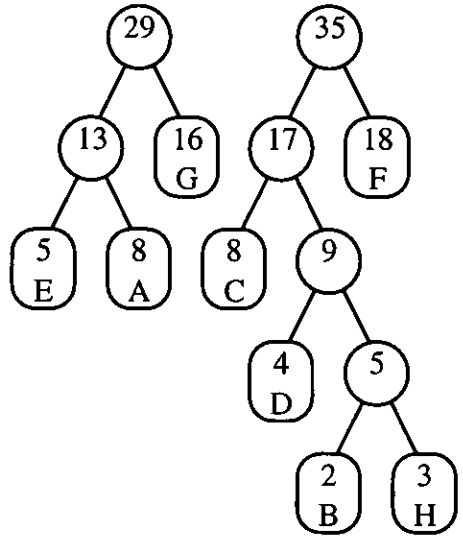


e) Fourth Parent Node Generated

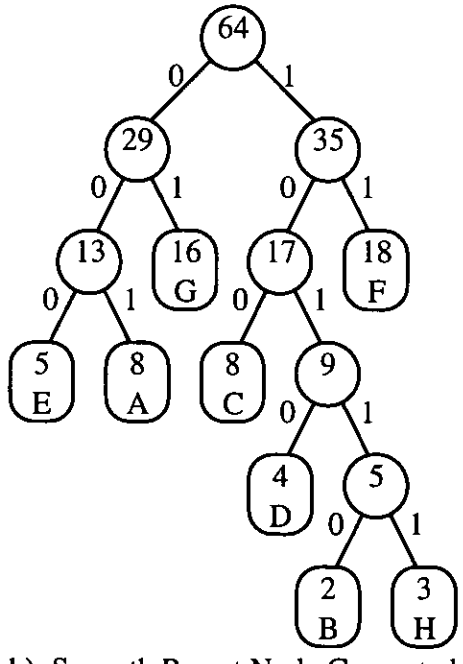
Figure B.2 a) - e) Generation of a Huffman Tree for 8-level Image



f) Fifth Parent Node Generated



g) Sixth Parent Node Generated



h) Seventh Parent Node Generated

Figure B.2 f) - h) Generation of a Huffman Tree for 8-level Image

Using symbol E instead would move symbols B, E, and H on the same level with symbol D as swapping the two nodes with a weight of 5 suggests; see figure B.2 h).

Tracing the path from the root node to a particular symbol generates a unique string of 0s and 1s associated with that symbol. Table B.2 provides the Huffman codewords that can be used to encode and decode the 8-level example image. In addition, the frequency of occurrence of every symbol has been multiplied with the length of the codeword of the symbol, and the image size has been calculated. The same information is provided for a 3-bit natural binary code. For comparison the self-information has been calculated using equation 2.5, and a zero-order entropy of 2.67 bits element^{-1} has been estimated using equation 2.7. In this example, the Huffman tree generates codewords with the number of bits equal to self-information for symbols A, B, C, D, and G; note that their probabilities of occurrence are integer powers of $(1/2)$. While symbol E is undercoded, symbols F and H are overcoded. However, the size of the Huffman-coded image approaches with 172 bits the lower bound of 170.6 bits.

Symbol	A	B	C	D	E	F	G	H
Frequency of Occurrence	8	2	8	4	5	18	16	3
Natural Code (binary)	000	001	010	011	100	101	110	111
Number of Bits	24	6	24	12	15	54	48	9
Image Size in Bits	192							
Huffman Code (binary)	001	10110	100	1010	000	11	01	10111
Number of Bits	24	10	24	16	15	36	32	15
Image Size in Bits	172							
Self-information	3.00	5.00	3.00	4.00	3.68	1.83	2.00	4.42
Number of Bits	24.00	10.00	24.00	16.00	18.39	32.94	32.00	13.25
Lower Bound in Bits	170.6							

Table B.2 Sizes of 8-level Image

C JPEG Example Tables

C.1 Introduction

This appendix provides examples of quantization and Huffman tables; see (ISO/IEC 10918-1:1994, annex K).

C.2 Quantization Tables

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

Table C.1 Example of Luminance Quantization Table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Table C.2 Example of Chrominance Quantization Table

C.3 Huffman Tables for 8-bit Precision

DC Difference Category (hexadecimal)	Codeword (binary)
0	00
1	010
2	011
3	100
4	101
5	110
6	1110
7	11110
8	111110
9	1111110
A	11111110
B	111111110

Table C.3 Example of Luminance DC Difference Table

DC Difference Category (hexadecimal)	Codeword (binary)
0	00
1	01
2	10
3	110
4	1110
5	11110
6	111110
7	1111110
8	11111110
9	111111110
A	1111111110
B	11111111110

Table C.4 Example of Chrominance DC Difference Table

Zero Run	AC Category (hexadecimal)	Coding Symbol (hexadecimal)	Codeword (binary)
0	0	EOB	1010
0	1	01	00
0	2	02	01
0	3	03	100
0	4	04	1011
0	5	05	11010
0	6	06	1111000
0	7	07	11111000
0	8	08	1111110110
0	9	09	1111111110000010
0	A	0A	1111111110000011
1	1	11	1100
1	2	12	11011
1	3	13	1111001
1	4	14	111110110
1	5	15	11111110110
1	6	16	1111111110000100
1	7	17	1111111110000101
1	8	18	1111111110000110
1	9	19	1111111110000111
1	A	1A	1111111110001000
2	1	21	11100
2	2	22	11111001
2	3	23	1111110111
2	4	24	111111110100
2	5	25	1111111110001001
2	6	26	1111111110001010
2	7	27	1111111110001011
2	8	28	1111111110001100
2	9	29	1111111110001101
2	A	2A	1111111110001110
3	1	31	111010
3	2	32	111110111
3	3	33	111111110101
3	4	34	1111111110001111
3	5	35	1111111110010000
3	6	36	1111111110010001
3	7	37	1111111110010010
3	8	38	1111111110010011
3	9	39	1111111110010100
3	A	3A	1111111110010101

Table C.5 (1 of 4) Example of Luminance AC Table

Zero Run	AC Category (hexadecimal)	Coding Symbol (hexadecimal)	Codeword (binary)
4	1	41	111011
4	2	42	1111111000
4	3	43	1111111110010110
4	4	44	1111111110010111
4	5	45	1111111110011000
4	6	46	1111111110011001
4	7	47	1111111110011010
4	8	48	1111111110011011
4	9	49	1111111110011100
4	A	4A	1111111110011101
5	1	51	1111010
5	2	52	11111110111
5	3	53	1111111110011110
5	4	54	1111111110011111
5	5	55	1111111110100000
5	6	56	1111111110100001
5	7	57	1111111110100010
5	8	58	1111111110100011
5	9	59	1111111110100100
5	A	5A	1111111110100101
6	1	61	1111011
6	2	62	111111110110
6	3	63	1111111110100110
6	4	64	1111111110100111
6	5	65	1111111110101000
6	6	66	1111111110101001
6	7	67	1111111110101010
6	8	68	1111111110101011
6	9	69	1111111110101100
6	A	6A	1111111110101101
7	1	71	11111010
7	2	72	111111110111
7	3	73	1111111110101110
7	4	74	1111111110101111
7	5	75	1111111110110000
7	6	76	1111111110110001
7	7	77	1111111110110010
7	8	78	1111111110110011
7	9	79	1111111110110100
7	A	7A	1111111110110101

Table C.5 (2 of 4) Example of Luminance AC Table

Zero Run	AC Category (hexadecimal)	Coding Symbol (hexadecimal)	Codeword (binary)
8	1	81	111111000
8	2	82	111111111000000
8	3	83	1111111110110110
8	4	84	1111111110110111
8	5	85	1111111110111000
8	6	86	1111111110111001
8	7	87	1111111110111010
8	8	88	1111111110111011
8	9	89	1111111110111100
8	A	8A	1111111110111101
9	1	91	111111001
9	2	92	1111111110111110
9	3	93	1111111110111111
9	4	94	1111111111000000
9	5	95	1111111111000001
9	6	96	1111111111000010
9	7	97	1111111111000011
9	8	98	1111111111000100
9	9	99	1111111111000101
9	A	9A	1111111111000110
10	1	A1	111111010
10	2	A2	1111111111000111
10	3	A3	1111111111001000
10	4	A4	1111111111001001
10	5	A5	1111111111001010
10	6	A6	1111111111001011
10	7	A7	1111111111001100
10	8	A8	1111111111001101
10	9	A9	1111111111001110
10	A	AA	1111111111001111
11	1	B1	1111111001
11	2	B2	1111111111010000
11	3	B3	1111111111010001
11	4	B4	1111111111010010
11	5	B5	1111111111010011
11	6	B6	1111111111010100
11	7	B7	1111111111010101
11	8	B8	1111111111010110
11	9	B9	1111111111010111
11	A	BA	1111111111011000

Table C.5 (3 of 4) Example of Luminance AC Table

Zero Run	AC Category (hexadecimal)	Coding Symbol (hexadecimal)	Codeword (binary)
12	1	C1	1111111010
12	2	C2	1111111111011001
12	3	C3	1111111111011010
12	4	C4	1111111111011011
12	5	C5	1111111111011100
12	6	C6	1111111111011101
12	7	C7	1111111111011110
12	8	C8	1111111111011111
12	9	C9	1111111111100000
12	A	CA	1111111111100001
13	1	D1	11111111000
13	2	D2	1111111111100010
13	3	D3	1111111111100011
13	4	D4	1111111111100100
13	5	D5	1111111111100101
13	6	D6	1111111111100110
13	7	D7	1111111111100111
13	8	D8	1111111111101000
13	9	D9	1111111111101001
13	A	DA	1111111111101010
14	1	E1	1111111111101011
14	2	E2	1111111111101100
14	3	E3	1111111111101101
14	4	E4	1111111111101110
14	5	E5	1111111111101111
14	6	E6	1111111111110000
14	7	E7	1111111111110001
14	8	E8	1111111111110010
14	9	E9	1111111111110011
14	A	EA	1111111111110100
15	0	ZRL	11111111001
15	1	F1	1111111111110101
15	2	F2	1111111111110110
15	3	F3	1111111111110111
15	4	F4	1111111111111000
15	5	F5	1111111111111001
15	6	F6	1111111111111010
15	7	F7	1111111111111011
15	8	F8	1111111111111100
15	9	F9	1111111111111101
15	A	FA	1111111111111110

Table C.5 (4 of 4) Example of Luminance AC Table

D JPEG Baseline Sequential Process Worked Example

D.1 Introduction

This appendix provides a worked example of the coder processing steps in the baseline sequential process. An 8×8 block of samples is encoded and subsequently decoded following the processing steps described in section 3.4.

D.2 Encoding Processing Steps

Figure D.1 depicts an 8×8 block of source samples extracted from a real image; the small variations from sample to sample indicate the predominance of low spatial frequencies (G. K. Wallace 1992).

$$\begin{bmatrix} 139 & 144 & 149 & 153 & 155 & 155 & 155 & 155 \\ 144 & 151 & 153 & 156 & 159 & 156 & 156 & 156 \\ 150 & 155 & 160 & 163 & 158 & 156 & 156 & 156 \\ 159 & 161 & 162 & 160 & 160 & 159 & 159 & 159 \\ 159 & 160 & 161 & 162 & 162 & 155 & 155 & 155 \\ 161 & 161 & 161 & 161 & 160 & 157 & 157 & 157 \\ 162 & 162 & 161 & 163 & 162 & 157 & 157 & 157 \\ 162 & 162 & 161 & 161 & 163 & 158 & 158 & 158 \end{bmatrix}$$

Figure D.1 8×8 Block of Source Samples

Figure D.2 depicts the 8×8 block of samples level-shifted to the range $[-128,127]$.

$$\begin{bmatrix} 11 & 16 & 21 & 25 & 27 & 27 & 27 & 27 \\ 16 & 23 & 25 & 28 & 31 & 28 & 28 & 28 \\ 22 & 27 & 32 & 35 & 30 & 28 & 28 & 28 \\ 31 & 33 & 34 & 32 & 32 & 31 & 31 & 31 \\ 31 & 32 & 33 & 34 & 34 & 27 & 27 & 27 \\ 33 & 33 & 33 & 33 & 32 & 29 & 29 & 29 \\ 34 & 34 & 33 & 35 & 34 & 29 & 29 & 29 \\ 34 & 34 & 33 & 33 & 35 & 30 & 30 & 30 \end{bmatrix}$$

Figure D.2 8×8 Block of Samples to FDCT

Figure D.3 depicts the 8×8 block of DCT coefficients to one decimal place generated by the FDCT. Except for a few of the lower frequency coefficients the amplitudes are quite small.

$$\begin{bmatrix} 235.6 & -1.0 & -12.1 & -5.2 & 2.1 & -1.7 & -2.7 & 1.3 \\ -22.6 & -17.5 & -6.2 & -3.2 & -2.9 & -0.1 & 0.4 & -1.2 \\ -10.9 & -9.3 & -1.6 & 1.5 & 0.2 & -0.9 & -0.6 & -0.1 \\ -7.1 & -1.9 & 0.2 & 1.5 & 0.9 & -0.1 & 0.0 & 0.3 \\ -0.6 & -0.8 & 1.5 & 1.6 & -0.1 & -0.7 & 0.6 & 1.3 \\ 1.8 & -0.2 & 1.6 & -0.3 & -0.8 & 1.5 & 1.0 & -1.0 \\ -1.3 & -0.4 & -0.3 & -1.5 & -0.5 & 1.7 & 1.1 & -0.8 \\ -2.6 & 1.6 & -3.8 & -1.8 & 1.9 & 1.2 & -0.6 & -0.4 \end{bmatrix}$$

Figure D.3 8×8 Block of DCT Coefficients

Figure D.4 depicts the 8×8 block of quantized DCT coefficients processed using the luminance quantization table given in table C.1.

$$\begin{bmatrix} 15 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure D.4 8×8 Block of Quantized DCT Coefficients

Assuming that the quantized DC coefficient of the preceding block is 12, equation 3.5 generates the difference $DIFF = +3$. Figure D.5 depicts the 1-D vector reordered using the 8×8 zigzag scan path shown in figure 3.6.

$$[3 \;\; 0 \;\; -2 \;\; -1 \;\; -1 \;\; -1 \;\; 0 \;\; 0 \;\; -1 \;\; 0 \;\; 0 \;\; \ldots \;\; 0]$$

Figure D.5 1-D Vector of Reordered Values

Figure D.6 shows the intermediate sequence of symbols: one coding pair represents the DC difference category and the DC difference value itself followed by coding pairs each of which consists of zero run, AC category, and nonzero AC coefficient itself; and terminates with EOB.

$$[(2)(3) \;\; (1,2)(-2) \;\; (0,1)(-1) \;\; (0,1)(-1) \;\; (0,1)(-1) \;\; (2,1)(-1) \;\; \text{EOB}]$$

Figure D.6 Encoding of Intermediate Sequence of Symbols

Since it has been assumed that the source samples originate from a luminance component, the DC difference category and the AC categories are Huffman-encoded using tables C.3 and C.5 respectively. The additional bits for the DC difference value and the nonzero AC coefficients are generated using table 3.5. Figure D.7 shows the entropy-encoded stream of image data. Note that the spaces are solely for readability. Omitting any required parameters, such as quantization tables and Huffman tables; the 8×8 block of 8-bit source samples, totalling 512 bits, has been reduced to 31 bits.

[011 11 11011 01 00 0 00 0 00 0 11100 0 1010]

Figure D.7 Stream of Image Data

D.3 Decoding Processing Steps

Figure D.8 illustrates the entropy decoding starting with the uniformly spaced stream of image data in figure D.8 a). Since the first symbol to be decoded represents the DC difference category, table C.3 is used to decode the first symbol in the bit sequence: 011 codes DC difference category 2; see figure D.8 b). This category requires two additional bits: 11 codes 3 as table 3.5 reveals; see figure D.8 c).

Since the next symbol represents either zero run and AC category or ZRL or EOB, table C.5 is used to decode the next symbol in the bit sequence: 11011 codes a zero run of one and AC category 2; see figure D.8 d). This category requires two additional bits: 01 codes -2 as table 3.5 reveals; see figure D.8 e). These steps are repeated until the EOB is encountered; see figure D.8 f) - k).

[01111110110100000000001110001010]

a) Stream of Image Data

[(2)1111011010000000001110001010]

b) Decoding of DC Difference Category

[(2)(3) 1101101000000000001110001010]

c) Decoding of DC Difference Value

[(2)(3) (1,2)010000000001110001010]

d) Decoding of First AC Category

[(2)(3) (1,2)(−2) 0000000001110001010]

e) Decoding of First AC Amplitude

[(2)(3) (1,2)(−2) (0,1)(−1) 0000001110001010]

f) Decoding of Second Nonzero AC Coefficient

[(2)(3) (1,2)(−2) (0,1)(−1) (0,1)(−1) 0001110001010]

g) Decoding of Third Nonzero AC Coefficient

[(2)(3) (1,2)(−2) (0,1)(−1) (0,1)(−1) (0,1)(−1) 1110001010]

h) Decoding of Fourth Nonzero AC Coefficient

[(2)(3) (1,2)(−2) (0,1)(−1) (0,1)(−1) (0,1)(−1) (2,1)(−1) 1010]

j) Decoding of Fifth Nonzero AC Coefficient

[(2)(3) (1,2)(−2) (0,1)(−1) (0,1)(−1) (0,1)(−1) (2,1)(−1) EOB]

k) Decoding of EOB

Figure D.8 Decoding of Intermediate Sequence of Symbols

Evaluating the zero runs and appending an appropriate number of zeros reconstructs the 1-D vector as shown in figure D.9.

[3 0 −2 −1 −1 −1 0 0 −1 0 0 ... 0]

Figure D.9 Reconstructed 1-D Vector

Assuming that the quantized DC coefficient of the preceding block has been reconstructed as 12, the DC coefficient of the current block becomes 15. Figure D.10 depicts the 2-D block of quantized coefficients reordered back using the 8×8 zigzag scan path shown in figure 3.6.

$$\begin{bmatrix} 15 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure D.10 Reconstructed 8×8 Block of Quantized DCT Coefficients

Figure D.11 depicts the 8×8 block of dequantized DCT coefficients processed using the luminance quantization table given in table C.1.

$$\begin{bmatrix} 240 & 0 & -10 & 0 & 0 & 0 & 0 & 0 \\ -24 & -12 & 0 & 0 & 0 & 0 & 0 & 0 \\ -14 & -13 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure D.11 8×8 Block of Dequantized DCT Coefficients

Figure D.12 depicts the 8×8 block of samples generated by the IDCT.

$$\begin{bmatrix} 16 & 18 & 21 & 24 & 26 & 28 & 28 & 28 \\ 20 & 22 & 24 & 26 & 28 & 28 & 28 & 28 \\ 27 & 28 & 29 & 30 & 30 & 29 & 28 & 27 \\ 32 & 33 & 33 & 34 & 33 & 31 & 29 & 27 \\ 35 & 35 & 36 & 35 & 34 & 32 & 30 & 28 \\ 35 & 36 & 36 & 36 & 34 & 32 & 30 & 29 \\ 32 & 33 & 34 & 34 & 34 & 33 & 31 & 30 \\ 30 & 31 & 33 & 33 & 34 & 33 & 31 & 30 \end{bmatrix}$$

Figure D.12 8×8 Block of Samples from IDCT

Figure D.13 depicts the 8×8 block of reconstructed samples level-shifted back to the original range $[0,255]$.

$$\begin{bmatrix} 144 & 146 & 149 & 152 & 154 & 156 & 156 & 156 \\ 148 & 150 & 152 & 154 & 156 & 156 & 156 & 156 \\ 155 & 156 & 157 & 158 & 158 & 157 & 156 & 155 \\ 160 & 161 & 161 & 162 & 161 & 159 & 157 & 155 \\ 163 & 163 & 164 & 163 & 162 & 160 & 158 & 156 \\ 163 & 164 & 164 & 164 & 162 & 160 & 158 & 157 \\ 160 & 161 & 162 & 162 & 162 & 161 & 159 & 158 \\ 158 & 159 & 161 & 161 & 162 & 161 & 159 & 158 \end{bmatrix}$$

Figure D.13 8×8 Block of Reconstructed Samples

D.4 Reconstruction Error

Figure D.14 shows the 8×8 block of source samples minus reconstructed samples, $s - r$. A mean-square error of 5.2 has been calculated using equation 2.12.

$$\begin{bmatrix} -5 & -2 & 0 & 1 & 1 & -1 & -1 & -1 \\ -4 & 1 & 1 & 2 & 3 & 0 & 0 & 0 \\ -5 & -1 & 3 & 5 & 0 & -1 & 0 & 1 \\ -1 & 0 & 1 & -2 & -1 & 0 & 2 & 4 \\ -4 & -3 & -3 & -1 & 0 & -5 & -3 & -1 \\ -2 & -3 & -3 & -3 & -2 & -3 & -1 & 0 \\ 2 & 1 & -1 & 1 & 0 & -4 & -2 & -1 \\ 4 & 3 & 0 & 0 & 1 & -3 & -1 & 0 \end{bmatrix}$$

Figure D.14 8×8 Block of Error Values

E Images

Experimentation has been carried out on four 8-bit grey-scale images. The image Lena, depicted in figure E.1, shows head and shoulder of a woman in an indoor scene; it has a spatial resolution of 512×512 pixels. A version of the image with a spatial resolution of 256×256 pixels has also been used.

Figure E.1 Original Image, Lena 512×512

The image Cameraman, depicted in figure E.2, shows a man with camera and tripod in an outdoor scene; it has a spatial resolution of 256×256 pixels.

Figure E.2 Original Image, Cameraman 256×256

The image F-16, depicted in figure E.3, shows an aeroplane in a midair scene; it has a spatial resolution of 512×512 pixels.

Figure E.3 Original Image, F-16 512×512

F Versatile Zigzag Reordering Algorithm Worked Example

F.1 Introduction

This appendix provides a worked example for the versatile zigzag reordering algorithm. The scan path of a 3×2 sub-block is traversed using the binary decision tree described in subsection 4.4.6 and repeated in figure F.1 for convenience.

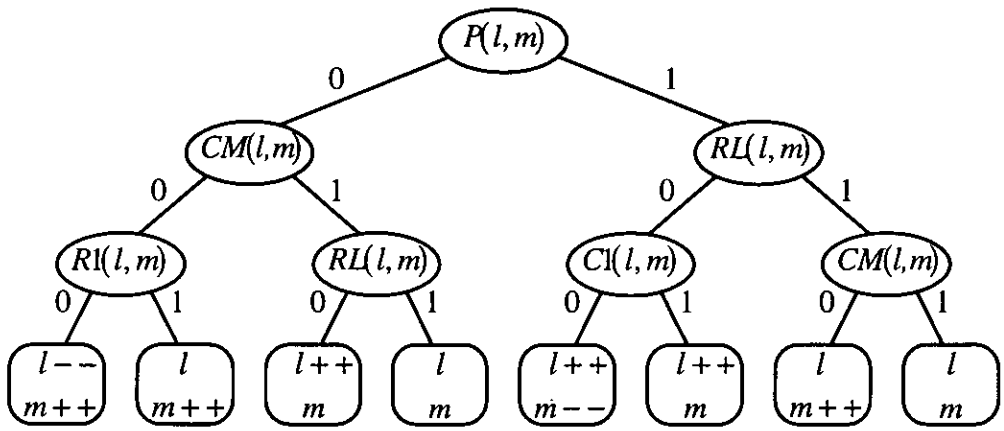


Figure F.1 Decision Tree for Changes in Row and Column Indices

F.2 Versatile Zigzag Reordering Algorithm

Figure F.2 depicts stages during the generation of the scan path for a sub-block with $L = 3$ rows and $M = 2$ columns. The current position in the scan path is indicated by a black dot.

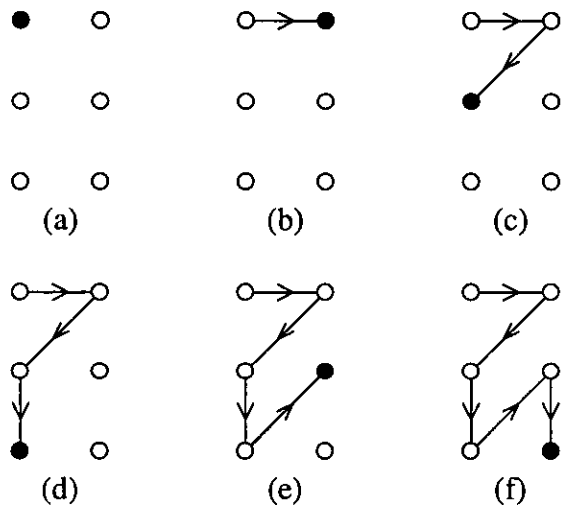


Figure F.2 Generation of Zigzag Scan Path for 3×2 Sub-block

The first position is given by $(l = 1, m = 1)$ since the scan path starts at the top left-hand position; see figure F.2 a). Determining the direction of movement at the first position begins with testing the parity parameter $P(l,m)$:

$$P(l = 1, m = 1) = 0 \tag{F.1}$$

since $(l + m) = (1 + 1) = 2$ which is even.

The second test evaluates therefore the last-column parameter $CM(l,m)$:

$$CM(l = 1, m = 1) = 0 \tag{F.2}$$

since $(m \neq M)$.

The third test evaluates therefore the first-row parameter $R1(l,m)$:

$$R1(l = 1, m = 1) = 1 \tag{F.3}$$

since $(l = L)$.

The position of the next element is situated in the right direction, i.e. the row index l must remain unchanged and the column index m must be incremented; see figure F.2 b).

The second position is therefore given by $(l = 1, m = 2)$. Determining the direction of movement at this position begins with testing the parity parameter $P(l,m)$:

$$P(l = 1, m = 2) = 1 \tag{F.4}$$

since $(l + m) = (1 + 2) = 3$ which is odd.

The second test evaluates therefore the last-row parameter $RL(l,m)$:

$$RL(l = 1, m = 2) = 0 \tag{F.5}$$

since $(l \neq L)$.

The third test evaluates therefore the first-column parameter $C1(l,m)$:

$$C1(l = 1, m = 1) = 0 \tag{F.6}$$

since $(m \neq 1)$.

The position of the next element is situated in the lower-left direction, i.e. the row index l must be incremented and the column index m must be decremented; see figure F.2 c).

The third position is therefore given by $(l = 2, m = 1)$. Determining the direction of movement at this position evaluates the test sequence:

$$P(l = 2, m = 1) = 1 \tag{F.7}$$

since $(l + m) = (2 + 1) = 3$ which is odd.

$$RL(l = 2, m = 1) = 0 \tag{F.8}$$

since $(l \neq L)$.

$$C1(l = 2, m = 1) = 1 \tag{F.9}$$

since $(m = 1)$.

The position of the next element is situated in the lower direction, i.e. the row index l must be incremented and the column index m must remain unchanged; see figure F.2 d).

The fourth position is therefore given by $(l = 3, m = 1)$. Determining the direction of movement at this position evaluates the test sequence:

$$P(l = 3, m = 1) = 0 \tag{F.10}$$

since $(l + m) = (3 + 1) = 4$ which is even.

$$CM(l = 3, m = 1) = 0 \tag{F.11}$$

since $(m \neq M)$.

$$R1(l=3,m=1)=0 \tag{F.12}$$

since $(l \neq 1)$.

The position of the next element is situated in the upper-right direction, i.e. the row index l must be decremented and the column index m must be incremented; see figure F.2 e).

The fifth position is therefore given by $(l=2,m=2)$. Determining the direction of movement at this position evaluates the test sequence:

$$P(l=2,m=2)=0 \tag{F.13}$$

since $(l+m)=(2+2)=4$ which is even.

$$CM(l=2,m=2)=1 \tag{F.14}$$

since $(m=M)$.

$$RL(l=2,m=2)=0 \tag{F.15}$$

since $(l \neq L)$.

The position of the sixth element is situated in the lower direction, i.e. the row index l must be incremented and the column index m must remain unchanged; see figure F.2 f).

The sixth position is therefore given by $(l = 3, m = 2)$. The full scan of the 3×2 sub-block is complete. Determining the direction of movement at this position evaluates the test sequence:

$$P(l = 3, m = 2) = 1 \tag{F.16}$$

since $(l + m) = (3 + 2) = 5$ which is odd.

$$RL(l = 3, m = 2) = 1 \tag{F.17}$$

since $(l = L)$.

$$CM(l = 3, m = 2) = 1 \tag{F.18}$$

since $(m = M)$.

The position remains unchanged, i.e. the row index l and the column index m must remain unchanged.

G Hardware Implementation Source Files

G.1 Introduction

This appendix contains the source files for the hardware implementation of the zigzag-reordering algorithm described in section 4.5. The files in sections G.2 and G.3 specify stage A and stage B respectively. The file in section G.4 combines the stages and simulates the state machine. Note that full sets of test vectors covering all 64 possible sub-block dimensions have been derived; however, the sections G.2 to G.4 include only abbreviated sets covering sub-block dimensions that are mentioned in chapter 4.

G.2 Source File Stage A

```
/*********DEPARTMENT*OF*ELECTRICAL*AND*ELECTRONIC*ENGINEERING*********
 *                                                                    *
 * File         : stage_a.tdl                                         *
 *                                                                    *
 * Description : file contains first, combinational stage of Moore    *
 *               state machine for versatile zigzag-reordering        *
 *               algorithm, and simulates the stage for given         *
 *               sub-block dimensions                                 *
 *                                                                    *
 * Author       : Hanns-Juergen Grosse                                *
 *                                                                    *
 * Copyright 1996-1997 by Hanns-Juergen Grosse. All rights reserved.  *
 *                                                                    *
 * Inputs       : (3 bits per input)                                  *
 *                external:                                           *
 *                ll2..0 (rows in sub-block)                          *
 *                mm2..0 (columns in sub-block)                       *
 *                from stage_b:                                       *
 *                l2..0  (current row index)                          *
 *                m2..0  (current column index)                       *
 *                                                                    *
 * Outputs      : (1 bit per output)                                  *
 *                p      (parity parameter)                           *
 *                r1     (first-row parameter)                        *
 *                rl     (last-row parameter)                         *
 *                c1     (first-column parameter)                     *
 *                cm     (last-column parameter)                      *
 *                                                                    *
 * Note         : row and column indices range from 000 to 111        *
 *                                                                    *
 ******************UNIVERSITY*OF*CENTRAL*LANCASHIRE******************/

stage_a(in  ll2..0,      /* rows in sub-block */
            mm2..0,      /* columns in sub-block */
            l2..0,       /* current row index */
            m2..0;       /* current column index */

        out p,           /* parity parameter */
            r1,          /* first-row parameter */
            rl,          /* last-row parameter */
            c1,          /* first-column parameter */
            cm)          /* last-column parameter */

{   group ll[ll2..0];
    group mm[mm2..0];
    group l[l2..0];
    group m[m2..0];

    /* output enable */
    p.oe = 1;
    r1.oe = 1;
    rl.oe = 1;
    c1.oe = 1;
    cm.oe = 1;
```

```
    p = l0 ^ m0;          /* parity parameter */

    if (0 == l)           /* first-row parameter */
        r1 = 1;

    if (ll == l)          /* last-row parameter */
        rl = 1;

    if (0 == m)           /* first-column parameter */
        c1 = 1;

    if (mm == m)          /* last-column parameter */
        cm = 1;

    /* write JEDEC file */
    putpart("g16v8", "stage_a",
            l12, l11, l10, mm2, mm1, mm0,  l2,  l1,  l0, GND,
              _,  r1,  rl,  m2,  m1,  m0,   p,  c1,  cm, VCC);

    /* simulate stage_a */
    test(ll, mm, l, m => p, r1, rl, c1, cm)
    {   tracef("%d %d %d %d    %d %d %d %d %d",
               ll, mm, l, m, p, r1, rl, c1, cm);

/*
 * test vectors for 5x1, 3x2, 2x3, 1x5, 3x5, 4x5,
 * and 8x8 zigzag scan paths
 *
 * Note:
 * row and column indices range from 000 to 111, therefore
 * the scan paths are defined as 4x0, 2x1, 1x2, 0x4, 2x4, 3x4, and 7x7
 *
 */

        /* row x column path */
        /* 5x1 path */
        (4, 0, 0, 0 => 0, 1, 0, 1, 1);
        (4, 0, 1, 0 => 1, 0, 0, 1, 1);
        (4, 0, 2, 0 => 0, 0, 0, 1, 1);
        (4, 0, 3, 0 => 1, 0, 0, 1, 1);
        (4, 0, 4, 0 => 0, 0, 1, 1, 1);

        /* 3x2 path */
        (2, 1, 0, 0 => 0, 1, 0, 1, 0);
        (2, 1, 0, 1 => 1, 1, 0, 0, 1);
        (2, 1, 1, 0 => 1, 0, 0, 1, 0);
        (2, 1, 2, 0 => 0, 0, 1, 1, 0);
        (2, 1, 1, 1 => 0, 0, 0, 0, 1);
        (2, 1, 2, 1 => 1, 0, 1, 0, 1);

        /* 2x3 path */
        (1, 2, 0, 0 => 0, 1, 0, 1, 0);
        (1, 2, 0, 1 => 1, 1, 0, 0, 0);
        (1, 2, 1, 0 => 1, 0, 1, 1, 0);
        (1, 2, 1, 1 => 0, 0, 1, 0, 0);
        (1, 2, 0, 2 => 0, 1, 0, 0, 1);
        (1, 2, 1, 2 => 1, 0, 1, 0, 1);
```

```
/* 1x5 path */
(0, 4, 0, 0 => 0, 1, 1, 1, 0);
(0, 4, 0, 1 => 1, 1, 1, 0, 0);
(0, 4, 0, 2 => 0, 1, 1, 0, 0);
(0, 4, 0, 3 => 1, 1, 1, 0, 0);
(0, 4, 0, 4 => 0, 1, 1, 0, 1);

/* 3x5 path */
(2, 4, 0, 0 => 0, 1, 0, 1, 0);
(2, 4, 0, 1 => 1, 1, 0, 0, 0);
(2, 4, 1, 0 => 1, 0, 0, 1, 0);
(2, 4, 2, 0 => 0, 0, 1, 1, 0);
(2, 4, 1, 1 => 0, 0, 0, 0, 0);
(2, 4, 0, 2 => 0, 1, 0, 0, 0);
(2, 4, 0, 3 => 1, 1, 0, 0, 0);
(2, 4, 1, 2 => 1, 0, 0, 0, 0);
(2, 4, 2, 1 => 1, 0, 1, 0, 0);
(2, 4, 2, 2 => 0, 0, 1, 0, 0);
(2, 4, 1, 3 => 0, 0, 0, 0, 0);
(2, 4, 0, 4 => 0, 1, 0, 0, 1);
(2, 4, 1, 4 => 1, 0, 0, 0, 1);
(2, 4, 2, 3 => 1, 0, 1, 0, 0);
(2, 4, 2, 4 => 0, 0, 1, 0, 1);

/* 4x5 path */
(3, 4, 0, 0 => 0, 1, 0, 1, 0);
(3, 4, 0, 1 => 1, 1, 0, 0, 0);
(3, 4, 1, 0 => 1, 0, 0, 1, 0);
(3, 4, 2, 0 => 0, 0, 0, 1, 0);
(3, 4, 1, 1 => 0, 0, 0, 0, 0);
(3, 4, 0, 2 => 0, 1, 0, 0, 0);
(3, 4, 0, 3 => 1, 1, 0, 0, 0);
(3, 4, 1, 2 => 1, 0, 0, 0, 0);
(3, 4, 2, 1 => 1, 0, 0, 0, 0);
(3, 4, 3, 0 => 1, 0, 1, 1, 0);
(3, 4, 3, 1 => 0, 0, 1, 0, 0);
(3, 4, 2, 2 => 0, 0, 0, 0, 0);
(3, 4, 1, 3 => 0, 0, 0, 0, 0);
(3, 4, 0, 4 => 0, 1, 0, 0, 1);
(3, 4, 1, 4 => 1, 0, 0, 0, 1);
(3, 4, 2, 3 => 1, 0, 0, 0, 0);
(3, 4, 3, 2 => 1, 0, 1, 0, 0);
(3, 4, 3, 3 => 0, 0, 1, 0, 0);
(3, 4, 2, 4 => 0, 0, 0, 0, 1);
(3, 4, 3, 4 => 1, 0, 1, 0, 1);

/* 8x8 path */
(7, 7, 0, 0 => 0, 1, 0, 1, 0);
(7, 7, 0, 1 => 1, 1, 0, 0, 0);
(7, 7, 1, 0 => 1, 0, 0, 1, 0);
(7, 7, 2, 0 => 0, 0, 0, 1, 0);
(7, 7, 1, 1 => 0, 0, 0, 0, 0);
(7, 7, 0, 2 => 0, 1, 0, 0, 0);
(7, 7, 0, 3 => 1, 1, 0, 0, 0);
(7, 7, 1, 2 => 1, 0, 0, 0, 0);
(7, 7, 2, 1 => 1, 0, 0, 0, 0);
(7, 7, 3, 0 => 1, 0, 0, 1, 0);
(7, 7, 4, 0 => 0, 0, 0, 1, 0);
(7, 7, 3, 1 => 0, 0, 0, 0, 0);
```

```
        (7, 7, 2, 2 => 0, 0, 0, 0, 0);
        (7, 7, 1, 3 => 0, 0, 0, 0, 0);
        (7, 7, 0, 4 => 0, 1, 0, 0, 0);
        (7, 7, 0, 5 => 1, 1, 0, 0, 0);
        (7, 7, 1, 4 => 1, 0, 0, 0, 0);
        (7, 7, 2, 3 => 1, 0, 0, 0, 0);
        (7, 7, 3, 2 => 1, 0, 0, 0, 0);
        (7, 7, 4, 1 => 1, 0, 0, 0, 0);
        (7, 7, 5, 0 => 1, 0, 0, 1, 0);
        (7, 7, 6, 0 => 0, 0, 0, 1, 0);
        (7, 7, 5, 1 => 0, 0, 0, 0, 0);
        (7, 7, 4, 2 => 0, 0, 0, 0, 0);
        (7, 7, 3, 3 => 0, 0, 0, 0, 0);
        (7, 7, 2, 4 => 0, 0, 0, 0, 0);
        (7, 7, 1, 5 => 0, 0, 0, 0, 0);
        (7, 7, 0, 6 => 0, 1, 0, 0, 0);
        (7, 7, 0, 7 => 1, 1, 0, 0, 1);
        (7, 7, 1, 6 => 1, 0, 0, 0, 0);
        (7, 7, 2, 5 => 1, 0, 0, 0, 0);
        (7, 7, 3, 4 => 1, 0, 0, 0, 0);
        (7, 7, 4, 3 => 1, 0, 0, 0, 0);
        (7, 7, 5, 2 => 1, 0, 0, 0, 0);
        (7, 7, 6, 1 => 1, 0, 0, 0, 0);
        (7, 7, 7, 0 => 1, 0, 1, 1, 0);
        (7, 7, 7, 1 => 0, 0, 1, 0, 0);
        (7, 7, 6, 2 => 0, 0, 0, 0, 0);
        (7, 7, 5, 3 => 0, 0, 0, 0, 0);
        (7, 7, 4, 4 => 0, 0, 0, 0, 0);
        (7, 7, 3, 5 => 0, 0, 0, 0, 0);
        (7, 7, 2, 6 => 0, 0, 0, 0, 0);
        (7, 7, 1, 7 => 0, 0, 0, 0, 1);
        (7, 7, 2, 7 => 1, 0, 0, 0, 1);
        (7, 7, 3, 6 => 1, 0, 0, 0, 0);
        (7, 7, 4, 5 => 1, 0, 0, 0, 0);
        (7, 7, 5, 4 => 1, 0, 0, 0, 0);
        (7, 7, 6, 3 => 1, 0, 0, 0, 0);
        (7, 7, 7, 2 => 1, 0, 1, 0, 0);
        (7, 7, 7, 3 => 0, 0, 1, 0, 0);
        (7, 7, 6, 4 => 0, 0, 0, 0, 0);
        (7, 7, 5, 5 => 0, 0, 0, 0, 0);
        (7, 7, 4, 6 => 0, 0, 0, 0, 0);
        (7, 7, 3, 7 => 0, 0, 0, 0, 1);
        (7, 7, 4, 7 => 1, 0, 0, 0, 1);
        (7, 7, 5, 6 => 1, 0, 0, 0, 0);
        (7, 7, 6, 5 => 1, 0, 0, 0, 0);
        (7, 7, 7, 4 => 1, 0, 1, 0, 0);
        (7, 7, 7, 5 => 0, 0, 1, 0, 0);
        (7, 7, 6, 6 => 0, 0, 0, 0, 0);
        (7, 7, 5, 7 => 0, 0, 0, 0, 1);
        (7, 7, 6, 7 => 1, 0, 0, 0, 1);
        (7, 7, 7, 6 => 1, 0, 1, 0, 0);
        (7, 7, 7, 7 => 0, 0, 1, 0, 1);
    } /* end of test */
} /* end of stage_a */
```

G.3 Source File Stage B

```
/*********DEPARTMENT*OF*ELECTRICAL*AND*ELECTRONIC*ENGINEERING*********
 *                                                                    *
 * File         : stage_b.tdl                                         *
 *                                                                    *
 * Description : file contains second, sequential stage of Moore      *
 *               state machine for versatile zigzag-reordering        *
 *               algorithm, and simulates the stage for given         *
 *               sub-block dimensions                                 *
 *                                                                    *
 * Author       : Hanns-Juergen Grosse                                *
 *                                                                    *
 * Copyright 1996-1997 by Hanns-Juergen Grosse. All rights reserved.  *
 *                                                                    *
 * Inputs       : (1 bit per input)                                   *
 *                from stage_a:                                       *
 *                p       (parity parameter)                          *
 *                r1      (first-row parameter)                       *
 *                rl      (last-row parameter)                        *
 *                c1      (first-column parameter)                    *
 *                cm      (last-column parameter)                     *
 *                external:                                           *
 *                clk     (clock signal)                              *
 *                oe      (output enable, to be connected to GND)     *
 *                reset   (reset signal)                              *
 *                                                                    *
 * Outputs      : (3 bits per output)                                 *
 *                l2..0   (current row index)                         *
 *                m2..0   (current column index)                      *
 *                (1 bit per input)                                   *
 *                done    (scan-complete signal)                      *
 *                                                                    *
 * Note         : row and column indices range from 000 to 111        *
 *                                                                    *
 ******************UNIVERSITY*OF*CENTRAL*LANCASHIRE*****************/

stage_b(in  p,          /* parity parameter */
            r1,         /* first-row parameter */
            rl,         /* last-row parameter */
            c1,         /* first-column parameter */
            cm,         /* last-column parameter */
            clk,        /* clock signal */
            oe,         /* output enable */
            reset;      /* reset signal */

        reg l2..0,      /* current row index */
            m2..0;      /* current column index */

        out done)       /* scan-complete signal */

{   group l[l2..0];
    group m[m2..0];

    /* clock signal */
    l.ck = clk;
    m.ck = clk;
```

```
/* output enable */
l.oe = !oe;
m.oe = !oe;
done.oe = 1;

/* synchronous clear */
l.clr = reset;
m.clr = reset;

/* overall decision tree */
if (rl & cm)
{   /* force back to start, ready for next sub-block */
    done = 1;
    l = 0;
    m = 0;
} else
{   done = 0;
    if (l == p)
    {   if (l == rl)
        {   /* row unchanged */
            l = l;

            /* increment column */
            m0 = !m0;
            m1 = m0 ^ m1;
            m2 = (m0 & m1) | m2;
        } else
        {   /* increment row */
            l0 = !l0;
            l1 = l0 ^l1;
            l2 = (l0 & l1) | l2;

            if (l == c1)
            {   /* column unchanged */
                m = m;
            } else
            {   /* decrement column */
                m0 = !m0;
                m1 = m0 !^ m1;
                m2 = (m0 & m2) | (m1 & m2);
            } /* end if (c1) */
        } /* end if (rl) */
    } else
    {   if (l == cm)
        {   /* increment row */
            l0 = !l0;
            l1 = l0 ^ l1;
            l2 = (l0 & l1) | l2;

            /* column unchanged */
            m = m;
        } else
        {   /* increment col */
            m0 = !m0;
            m1 = m0 ^ m1;
            m2 = (m0 & m1) | m2;
```

```
                if (1 == r1)
                {   /* row unchanged */
                    l = l;
                } else
                {   /* decrement row */
                    l0 = !l0;
                    l1 = l0 !^ l1;
                    l2 = (l0 & l2) | (l1 & l2);
                } /* end if (r1) */
            } /* end if (cm) */
        } /* end if (p) */
    } /* end if (r1 & cm) */

    /* write JEDEC file */
    putpart("g16v8", "stage_b",
            clk,  cm,  c1,   p, reset,  _,  _, r1,   r1, GND,
             oe,  l2,  l1,  l0,    m2, m1, m0,  _, done, VCC);

    /* simulate stage_b */
    test(clk, p, r1, r1, c1, cm => l, m, done)
    {   tracef("%w %d %d %d %d %d   %d %d %w",
               clk, p, r1, r1, c1, cm, l, m, done);

/*
 * test vectors for 5x1, 3x2, 2x3, 1x5, 3x5, 4x5,
 * and 8x8 zigzag scan paths
 *
 * Note:
 * row and column indices range from 000 to 111, therefore
 * the scan paths are defined as 4x0, 2x1, 1x2, 0x4, 2x4, 3x4, and 7x7
 *
 * indices automatically reset to first position
 *
 */

        oe = 0;
        reset = 1;
        /* reset to first position */
        (\C, 0, 0, 0, 0, 0 => 0, 0, 0);
        reset = 0;

        /* row x column path */
        /* 5x1 path */
        (\C, 0, 1, 0, 1, 1 => 1, 0, 0);
        (\C, 1, 0, 0, 1, 1 => 2, 0, 0);
        (\C, 0, 0, 0, 1, 1 => 3, 0, 0);
        (\C, 1, 0, 0, 1, 1 => 4, 0, 0);
        (\C, 0, 0, 1, 1, 1 => 0, 0, 1);

        /* 3x2 path */
        (\C, 0, 1, 0, 1, 0 => 0, 1, 0);
        (\C, 1, 1, 0, 0, 1 => 1, 0, 0);
        (\C, 1, 0, 0, 1, 0 => 2, 0, 0);
        (\C, 0, 0, 1, 1, 0 => 1, 1, 0);
        (\C, 0, 0, 0, 0, 1 => 2, 1, 0);
        (\C, 1, 0, 1, 0, 1 => 0, 0, 1);
```

```
/* 2x3 path */
(\C, 0, 1, 0, 1, 0 => 0, 1, 0);
(\C, 1, 1, 0, 0, 0 => 1, 0, 0);
(\C, 1, 0, 1, 1, 0 => 1, 1, 0);
(\C, 0, 0, 1, 0, 0 => 0, 2, 0);
(\C, 0, 1, 0, 0, 1 => 1, 2, 0);
(\C, 1, 0, 1, 0, 1 => 0, 0, 1);

/* 1x5 path */
(\C, 0, 1, 1, 1, 0 => 0, 1, 0);
(\C, 1, 1, 1, 0, 0 => 0, 2, 0);
(\C, 0, 1, 1, 0, 0 => 0, 3, 0);
(\C, 1, 1, 1, 0, 0 => 0, 4, 0);
(\C, 0, 1, 1, 0, 1 => 0, 0, 1);

/* 3x5 path */
(\C, 0, 1, 0, 1, 0 => 0, 1, 0);
(\C, 1, 1, 0, 0, 0 => 1, 0, 0);
(\C, 1, 0, 0, 1, 0 => 2, 0, 0);
(\C, 0, 0, 1, 1, 0 => 1, 1, 0);
(\C, 0, 0, 0, 0, 0 => 0, 2, 0);
(\C, 0, 1, 0, 0, 0 => 0, 3, 0);
(\C, 1, 1, 0, 0, 0 => 1, 2, 0);
(\C, 1, 0, 0, 0, 0 => 2, 1, 0);
(\C, 1, 0, 1, 0, 0 => 2, 2, 0);
(\C, 0, 0, 1, 0, 0 => 1, 3, 0);
(\C, 0, 0, 0, 0, 0 => 0, 4, 0);
(\C, 0, 1, 0, 0, 1 => 1, 4, 0);
(\C, 1, 0, 0, 0, 1 => 2, 3, 0);
(\C, 1, 0, 1, 0, 0 => 2, 4, 0);
(\C, 0, 0, 1, 0, 1 => 0, 0, 1);

/* 4x5 path */
(\C, 0, 1, 0, 1, 0 => 0, 1, 0);
(\C, 1, 1, 0, 0, 0 => 1, 0, 0);
(\C, 1, 0, 0, 1, 0 => 2, 0, 0);
(\C, 0, 0, 0, 1, 0 => 1, 1, 0);
(\C, 0, 0, 0, 0, 0 => 0, 2, 0);
(\C, 0, 1, 0, 0, 0 => 0, 3, 0);
(\C, 1, 1, 0, 0, 0 => 1, 2, 0);
(\C, 1, 0, 0, 0, 0 => 2, 1, 0);
(\C, 1, 0, 0, 0, 0 => 3, 0, 0);
(\C, 1, 0, 1, 1, 0 => 3, 1, 0);
(\C, 0, 0, 1, 0, 0 => 2, 2, 0);
(\C, 0, 0, 0, 0, 0 => 1, 3, 0);
(\C, 0, 0, 0, 0, 0 => 0, 4, 0);
(\C, 0, 1, 0, 0, 1 => 1, 4, 0);
(\C, 1, 0, 0, 0, 1 => 2, 3, 0);
(\C, 1, 0, 0, 0, 0 => 3, 2, 0);
(\C, 1, 0, 1, 0, 0 => 3, 3, 0);
(\C, 0, 0, 1, 0, 0 => 2, 4, 0);
(\C, 0, 0, 0, 0, 1 => 3, 4, 0);
(\C, 1, 0, 1, 0, 1 => 0, 0, 1);
```

```
/* 8x8 path */
(\C, 0, 1, 0, 1, 0 => 0, 1, 0);
(\C, 1, 1, 0, 0, 0 => 1, 0, 0);
(\C, 1, 0, 0, 1, 0 => 2, 0, 0);
(\C, 0, 0, 0, 1, 0 => 1, 1, 0);
(\C, 0, 0, 0, 0, 0 => 0, 2, 0);
(\C, 0, 1, 0, 0, 0 => 0, 3, 0);
(\C, 1, 1, 0, 0, 0 => 1, 2, 0);
(\C, 1, 0, 0, 0, 0 => 2, 1, 0);
(\C, 1, 0, 0, 0, 0 => 3, 0, 0);
(\C, 1, 0, 0, 1, 0 => 4, 0, 0);
(\C, 0, 0, 0, 1, 0 => 3, 1, 0);
(\C, 0, 0, 0, 0, 0 => 2, 2, 0);
(\C, 0, 0, 0, 0, 0 => 1, 3, 0);
(\C, 0, 0, 0, 0, 0 => 0, 4, 0);
(\C, 0, 1, 0, 0, 0 => 0, 5, 0);
(\C, 1, 1, 0, 0, 0 => 1, 4, 0);
(\C, 1, 0, 0, 0, 0 => 2, 3, 0);
(\C, 1, 0, 0, 0, 0 => 3, 2, 0);
(\C, 1, 0, 0, 0, 0 => 4, 1, 0);
(\C, 1, 0, 0, 0, 0 => 5, 0, 0);
(\C, 1, 0, 0, 1, 0 => 6, 0, 0);
(\C, 0, 0, 0, 1, 0 => 5, 1, 0);
(\C, 0, 0, 0, 0, 0 => 4, 2, 0);
(\C, 0, 0, 0, 0, 0 => 3, 3, 0);
(\C, 0, 0, 0, 0, 0 => 2, 4, 0);
(\C, 0, 0, 0, 0, 0 => 1, 5, 0);
(\C, 0, 0, 0, 0, 0 => 0, 6, 0);
(\C, 0, 1, 0, 0, 0 => 0, 7, 0);
(\C, 1, 1, 0, 0, 1 => 1, 6, 0);
(\C, 1, 0, 0, 0, 0 => 2, 5, 0);
(\C, 1, 0, 0, 0, 0 => 3, 4, 0);
(\C, 1, 0, 0, 0, 0 => 4, 3, 0);
(\C, 1, 0, 0, 0, 0 => 5, 2, 0);
(\C, 1, 0, 0, 0, 0 => 6, 1, 0);
(\C, 1, 0, 0, 0, 0 => 7, 0, 0);
(\C, 1, 0, 1, 1, 0 => 7, 1, 0);
(\C, 0, 0, 1, 0, 0 => 6, 2, 0);
(\C, 0, 0, 0, 0, 0 => 5, 3, 0);
(\C, 0, 0, 0, 0, 0 => 4, 4, 0);
(\C, 0, 0, 0, 0, 0 => 3, 5, 0);
(\C, 0, 0, 0, 0, 0 => 2, 6, 0);
(\C, 0, 0, 0, 0, 0 => 1, 7, 0);
(\C, 0, 0, 0, 0, 1 => 2, 7, 0);
(\C, 1, 0, 0, 0, 1 => 3, 6, 0);
(\C, 1, 0, 0, 0, 0 => 4, 5, 0);
(\C, 1, 0, 0, 0, 0 => 5, 4, 0);
(\C, 1, 0, 0, 0, 0 => 6, 3, 0);
(\C, 1, 0, 0, 0, 0 => 7, 2, 0);
(\C, 1, 0, 1, 0, 0 => 7, 3, 0);
(\C, 0, 0, 1, 0, 0 => 6, 4, 0);
(\C, 0, 0, 0, 0, 0 => 5, 5, 0);
(\C, 0, 0, 0, 0, 0 => 4, 6, 0);
(\C, 0, 0, 0, 0, 0 => 3, 7, 0);
(\C, 0, 0, 0, 0, 1 => 4, 7, 0);
(\C, 1, 0, 0, 0, 1 => 5, 6, 0);
(\C, 1, 0, 0, 0, 0 => 6, 5, 0);
(\C, 1, 0, 0, 0, 0 => 7, 4, 0);
(\C, 1, 0, 1, 0, 0 => 7, 5, 0);
```

```
        (\C, 0, 0, 1, 0, 0 => 6, 6, 0);
        (\C, 0, 0, 0, 0, 0 => 5, 7, 0);
        (\C, 0, 0, 0, 0, 1 => 6, 7, 0);
        (\C, 1, 0, 0, 0, 1 => 7, 6, 0);
        (\C, 1, 0, 1, 0, 0 => 7, 7, 0);
        (\C, 0, 0, 1, 0, 1 => 0, 0, 1);
    } /* end of test */
} /* end of stage_b */
```

G.4 Source File State Machine

```
/*********DEPARTMENT*OF*ELECTRICAL*AND*ELECTRONIC*ENGINEERING*********
 *                                                                    *
 * File         : moore.tdl                                           *
 *                                                                    *
 * Description : file reads the JEDEC fusemaps from stage_a.jed and   *
 *               stage_b.jed, and simulates the Moore state machine   *
 *               for the given sub-block dimensions                   *
 *                                                                    *
 * Author       : Hanns-Juergen Grosse                                *
 *                                                                    *
 * Copyright 1996-1997 by Hanns-Juergen Grosse. All rights reserved.  *
 *                                                                    *
 * Inputs       : (3 bits per input)                                  *
 *                ll2..0 (rows in sub-block)                          *
 *                mm2..0 (columns in sub-block)                       *
 *                (1 bit per input)                                   *
 *                clk    (clock signal)                               *
 *                oe     (output enable, to be connected to GND)      *
 *                reset  (reset signal)                               *
 *                                                                    *
 * Outputs      : (3 bits per output)                                 *
 *                l2..0  (current row index)                          *
 *                m2..0  (current column index)                       *
 *                (1 bit per input)                                   *
 *                done   (scan-complete signal)                       *
 *                                                                    *
 * Note         : row and column indices start range from 000 to 111  *
 *                                                                    *
 ******************UNIVERSITY*OF*CENTRAL*LANCASHIRE******************/

moore(net ll2..0,       /* rows in sub-block */
          mm2..0,       /* columns in sub-block */
          clk,          /* clock signal */
          oe,           /* output enable */
          reset,        /* reset signal */
          l2..0,        /* current row index */
          m2..0,        /* current column index */
          done,         /* scan-complete signal */
          p,            /* parity parameter */
          r1,           /* first-row parameter */
          r1,           /* last-row parameter */
          c1,           /* first-column parameter */
          cm)           /* last-column parameter */

{   group ll[ll2..0];
    group mm[mm2..0];
    group l[l2..0];
    group m[m2..0];

    /* read JEDEC files */
    getpart("g16v8", "stage_a",
            ll2, ll1, ll0, mm2, mm1, mm0,  l2,  l1,  l0, GND,
              _,  r1,  r1,  m2,  m1,  m0,   p,  c1,  cm, VCC);
```

```
    getpart("g16v8", "stage_b",
            clk,  cm,  c1,   p, reset,  _,  _, r1,   r1, GND,
             oe,  12,  11,  10,    m2, m1, m0,  _, done, VCC);

    /* simulate state machine */
    test(clk, 11, mm => 1, m, done)
    {   tracef("%w %d %d   %d %d %w",
               clk, 11, mm, 1, m, done);

/*
 * test vectors for 5x1, 3x2, 2x3, 1x5, 3x5, 4x5,
 * and 8x8 zigzag scan paths
 *
 * Note:
 * row and column indices range from 000 to 111, therefore
 * the scan paths are defined as 4x0, 2x1, 1x2, 0x4, 2x4, 3x4, and 7x7
 *
 * indices automatically reset to first position
 *
 */

        oe = 0;
        reset = 1;
        /* reset to first position */
        (\C, ?, ? => 0, 0, ?);
        reset = 0;

        /* row x column path */
        /* 5x1 path */
        (\C, 4, 0 => 1, 0, 0);
        (\C, 4, 0 => 2, 0, 0);
        (\C, 4, 0 => 3, 0, 0);
        (\C, 4, 0 => 4, 0, 1);
        (\C, 4, 0 => 0, 0, 0);

        /* 3x2 path */
        (\C, 2, 1 => 0, 1, 0);
        (\C, 2, 1 => 1, 0, 0);
        (\C, 2, 1 => 2, 0, 0);
        (\C, 2, 1 => 1, 1, 0);
        (\C, 2, 1 => 2, 1, 1);
        (\C, 2, 1 => 0, 0, 0);

        /* 2x3 path */
        (\C, 1, 2 => 0, 1, 0);
        (\C, 1, 2 => 1, 0, 0);
        (\C, 1, 2 => 1, 1, 0);
        (\C, 1, 2 => 0, 2, 0);
        (\C, 1, 2 => 1, 2, 1);
        (\C, 1, 2 => 0, 0, 0);

        /* 1x5 path */
        (\C, 0, 4 => 0, 1, 0);
        (\C, 0, 4 => 0, 2, 0);
        (\C, 0, 4 => 0, 3, 0);
        (\C, 0, 4 => 0, 4, 1);
        (\C, 0, 4 => 0, 0, 0);
```

```
/* 3x5 path */
(\C, 2, 4 => 0, 1, 0);
(\C, 2, 4 => 1, 0, 0);
(\C, 2, 4 => 2, 0, 0);
(\C, 2, 4 => 1, 1, 0);
(\C, 2, 4 => 0, 2, 0);
(\C, 2, 4 => 0, 3, 0);
(\C, 2, 4 => 1, 2, 0);
(\C, 2, 4 => 2, 1, 0);
(\C, 2, 4 => 2, 2, 0);
(\C, 2, 4 => 1, 3, 0);
(\C, 2, 4 => 0, 4, 0);
(\C, 2, 4 => 1, 4, 0);
(\C, 2, 4 => 2, 3, 0);
(\C, 2, 4 => 2, 4, 1);
(\C, 2, 4 => 0, 0, 0);

/* 4x5 path */
(\C, 3, 4 => 0, 1, 0);
(\C, 3, 4 => 1, 0, 0);
(\C, 3, 4 => 2, 0, 0);
(\C, 3, 4 => 1, 1, 0);
(\C, 3, 4 => 0, 2, 0);
(\C, 3, 4 => 0, 3, 0);
(\C, 3, 4 => 1, 2, 0);
(\C, 3, 4 => 2, 1, 0);
(\C, 3, 4 => 3, 0, 0);
(\C, 3, 4 => 3, 1, 0);
(\C, 3, 4 => 2, 2, 0);
(\C, 3, 4 => 1, 3, 0);
(\C, 3, 4 => 0, 4, 0);
(\C, 3, 4 => 1, 4, 0);
(\C, 3, 4 => 2, 3, 0);
(\C, 3, 4 => 3, 2, 0);
(\C, 3, 4 => 3, 3, 0);
(\C, 3, 4 => 2, 4, 0);
(\C, 3, 4 => 3, 4, 1);
(\C, 3, 4 => 0, 0, 0);

/* 8x8 path */
(\C, 7, 7 => 0, 1, 0);
(\C, 7, 7 => 1, 0, 0);
(\C, 7, 7 => 2, 0, 0);
(\C, 7, 7 => 1, 1, 0);
(\C, 7, 7 => 0, 2, 0);
(\C, 7, 7 => 0, 3, 0);
(\C, 7, 7 => 1, 2, 0);
(\C, 7, 7 => 2, 1, 0);
(\C, 7, 7 => 3, 0, 0);
(\C, 7, 7 => 4, 0, 0);
(\C, 7, 7 => 3, 1, 0);
(\C, 7, 7 => 2, 2, 0);
(\C, 7, 7 => 1, 3, 0);
(\C, 7, 7 => 0, 4, 0);
(\C, 7, 7 => 0, 5, 0);
(\C, 7, 7 => 1, 4, 0);
(\C, 7, 7 => 2, 3, 0);
(\C, 7, 7 => 3, 2, 0);
(\C, 7, 7 => 4, 1, 0);
```

```
        (\C, 7, 7 => 5, 0, 0);
        (\C, 7, 7 => 6, 0, 0);
        (\C, 7, 7 => 5, 1, 0);
        (\C, 7, 7 => 4, 2, 0);
        (\C, 7, 7 => 3, 3, 0);
        (\C, 7, 7 => 2, 4, 0);
        (\C, 7, 7 => 1, 5, 0);
        (\C, 7, 7 => 0, 6, 0);
        (\C, 7, 7 => 0, 7, 0);
        (\C, 7, 7 => 1, 6, 0);
        (\C, 7, 7 => 2, 5, 0);
        (\C, 7, 7 => 3, 4, 0);
        (\C, 7, 7 => 4, 3, 0);
        (\C, 7, 7 => 5, 2, 0);
        (\C, 7, 7 => 6, 1, 0);
        (\C, 7, 7 => 7, 0, 0);
        (\C, 7, 7 => 7, 1, 0);
        (\C, 7, 7 => 6, 2, 0);
        (\C, 7, 7 => 5, 3, 0);
        (\C, 7, 7 => 4, 4, 0);
        (\C, 7, 7 => 3, 5, 0);
        (\C, 7, 7 => 2, 6, 0);
        (\C, 7, 7 => 1, 7, 0);
        (\C, 7, 7 => 2, 7, 0);
        (\C, 7, 7 => 3, 6, 0);
        (\C, 7, 7 => 4, 5, 0);
        (\C, 7, 7 => 5, 4, 0);
        (\C, 7, 7 => 6, 3, 0);
        (\C, 7, 7 => 7, 2, 0);
        (\C, 7, 7 => 7, 3, 0);
        (\C, 7, 7 => 6, 4, 0);
        (\C, 7, 7 => 5, 5, 0);
        (\C, 7, 7 => 4, 6, 0);
        (\C, 7, 7 => 3, 7, 0);
        (\C, 7, 7 => 4, 7, 0);
        (\C, 7, 7 => 5, 6, 0);
        (\C, 7, 7 => 6, 5, 0);
        (\C, 7, 7 => 7, 4, 0);
        (\C, 7, 7 => 7, 5, 0);
        (\C, 7, 7 => 6, 6, 0);
        (\C, 7, 7 => 5, 7, 0);
        (\C, 7, 7 => 6, 7, 0);
        (\C, 7, 7 => 7, 6, 0);
        (\C, 7, 7 => 7, 7, 1);
        (\C, 7, 7 => 0, 0, 0);
    } /* end of test */
} /* end of moore */
```

H Publications

GROSSE, Hanns-Juergen, VARLEY, Martin Roy, TERRELL, Trevor James, and CHAN, Yiu Keung. 1997a. Sub-block classification using a neural network for adaptive zigzag reordering in JPEG-like image compression scheme. *In:* IEE. 1997/133. Neural and fuzzy systems: design, hardware and applications. London, UK. The Institution of Electrical Engineers. May 1997. ISSN 0963-3308. pp. 9/1-9/4. Colloquium on neural and fuzzy systems: design, hardware and applications in London, UK, 09 May 1997.

GROSSE, Hanns-Juergen, VARLEY, Martin Roy, TERRELL, Trevor James, and CHAN, Yiu Keung. 1997b. Hardware implementation of versatile zigzag-reordering algorithm for adaptive JPEG-like image compression schemes. *In:* IEE. 1997. Sixth international conference on image processing and its applications. London, UK. The Institution of Electrical Engineers. Jul. 1997. vol. 443, pt 1 of 2. ISBN 0-85296-692-X. pp. 184-188. Sixth international conference on image processing and its applications in Dublin, Ireland, 14-17 Jul. 1997.

GROSSE, Hanns-Juergen, VARLEY, Martin Roy, TERRELL, Trevor James, and CHAN, Yiu Keung. 1997c. Adaptive zigzag-reordering algorithm for improved coding in JPEG-like image compression schemes. *In:* TBM Ltd. 1997. Second international symposium on digital signal processing. Colchester, UK. Trusty Business Machines Ltd. Jul. 1997. pp. 7-11. Second international symposium on digital signal processing in London, UK, 22 Jul. 1997.

SUB-BLOCK CLASSIFICATION USING A NEURAL NETWORK FOR ADAPTIVE ZIGZAG REORDERING IN JPEG-LIKE IMAGE COMPRESSION SCHEME

H.-J. Grosse, M. R. Varley, T. J. Terrell, and Y. K. Chan

ABSTRACT

In this paper a neural-network technique for classification of blocks of discrete cosine transform (DCT) coefficients using a backpropagation algorithm is described. The DCT is employed in a variety of transform-based image compression schemes. In the authors' recent JPEG-like image compression scheme, efficient reordering of coefficients is achieved by applying adaptive zigzag reordering to variable-size rectangular sub-blocks. The additional neural-network-based sub-block classification discards isolated nonzero coefficients of small significance in some sub-blocks and therefore further reduces their sizes. Initial experimental results are presented that demonstrate the potential of the additional neural-network-based sub-block classification in terms of improved coding gain.

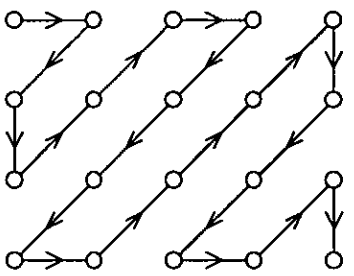
INTRODUCTION

Many image compression schemes, such as the standard JPEG scheme [1], operate by processing small non-overlapping image blocks (usually square $N \times N$ blocks of a fixed size, e.g. 8 pixels × 8 pixels) using a 2-D transform such as the discrete cosine transform (DCT) [2]. Whilst the transform itself is reversible and lossless, it is used to decorrelate the data so that the inter-element correlation in the transform domain is significantly less than that in the spatial domain. The resulting 2-D block of transform coefficients is then processed in the transform domain; in many cases this involves discarding some of the low-value transform coefficients to reduce the amount of data to be transmitted or stored, which causes a loss of information.

In the authors' recent JPEG-like scheme, each $N \times N$ block of quantized and thresholded transform coefficients is modified to yield the smallest possible sub-block to include all nonzero transform coefficients to be coded [3, 4]. As an example, Fig. 1 (a) depicts an 8×8 block of quantized transform coefficients,

$$\left[\begin{array}{cccccccc} -26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\ 1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

(a)



(b)

Figure 1 (a) 8×8 Block of Transform Coefficients, and (b) Zigzag Scan Path for 4×5 Sub-Block

H.-J. Grosse, M. R. Varley, and T. J. Terrell are with the Department of Electrical and Electronic Engineering, University of Central Lancashire, Preston, Lancashire, PR1 2HE, United Kingdom.

Y. K. Chan is with the Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon Tong, Kowloon, Hong Kong.

with the corresponding 4×5 sub-block indicated by the dotted line. The sub-block, being adapted to the particularities of the corresponding block, is not necessarily square, but is rectangular and extends from the top left-hand corner with a height and a width between 1 and N. Its dimensions, in the above example 4×5, need to be retained in order to traverse the scan path correctly using the adaptive zigzag-reordering algorithm. The algorithm, which in itself is transparent and lossless, generates the next position in the scan path, and therefore the whole scan path, through Boolean expressions operating on the current position and the dimensions of the sub-block; and produces a 1-D matrix of coefficients. Figure 1 (b) depicts the scan path for a 4×5 sub-block. The 1-D matrix is then converted into an intermediate symbol sequence, each symbol representing the number of zero coefficients preceding the current nonzero coefficient, the amplitude classification and the actual amplitude of the nonzero coefficient [1]. This facilitates entropy coding by placing low-frequency coefficients, that are more likely to be nonzero, before high-frequency coefficients. An end-of-block symbol (EOB) terminates the block after the last nonzero coefficient. Huffman coding [5], or arithmetic coding [6], can be used to convert the symbols to a continuous data stream.

SUB-BLOCK DETERMINATION USING NEURAL-NETWORK CLASSIFICATION

Isolated nonzero transform coefficients in a block diminish the effectiveness of adaptive zigzag reordering, since retaining isolated nonzero coefficients also requires that a large number of otherwise unnecessary zero coefficients are retained. However, if the contribution to reconstruction of an isolated transform coefficient is found to be expendable, a significantly smaller sub-block may be generated. The additional reconstruction error introduced by discarding the isolated nonzero coefficient is limited to the corresponding block of pixels.

The decision to sacrifice an isolated transform coefficient should take into account the contributions of all coefficients in the block in order to weight the contribution of the isolated coefficient correctly. As the adaptive zigzag-reordering algorithm requires only the sub-block dimensions, i. e. the row and column indices of the bottom right-hand corner, a neural-network-based classifier can be used to determine the appropriate dimensions.

FEEDFORWARD NETWORK AND TRAINING SET COMPOSITION

A feedforward network with 64 input neurons, 256 hidden neurons, and 64 output neurons has been trained using a backpropagation algorithm. The neurons in the two trainable layers, i. e. hidden layer and output layer, have log-sigmoid transfer functions because their output range, being between 0.0 and 1.0, is appropriate for learning to output binary values [7].

The input layer provides one neuron per element. In order to homogenize input values, amplitudes of the transform coefficients are classed according to their word lengths in bits for entropy coding in JPEG [1]; and the classifications are normalized, i. e. divided by the maximum value within the block. The input layer therefore receives the block of normalized amplitude classifications, that range from 0.0 to 1.0. As an example, Fig. 2 depicts an 8×8 block of quantized transform coefficients, the corresponding amplitude classifications according to JPEG and the normalized input values to the network.

The number of neurons in the hidden layer has been determined experimentally and is a compromise between performance and complexity.

The output layer uses a simple 1-in-64 binary code to identify the dimensions of the 64 possible sub-block classes. This code, although requiring 64 neurons, allows competitive selection of one output neuron and has been found to be more reliable than other codes, for example a 6-bit natural binary code that would require only 6 neurons.

Composition of the training set is of great importance as the performance of the network depends on the initial training, and the large amounts of image data available must be limited to a representative collection. The training set, that has been composed manually, consists of 64 idealized input sets and 10 examples of each of the 58 sub-block classes that have been selected from three images. However, for 6 of the 64 possible sub-block classes, suitable examples have not been found in the selected images. The small number of idealized sets, with all elements within the corresponding sub-blocks set to 1.0, supports the network's ability to classify ideal input sets and the sub-block classes for which input sets have not been available.

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\ 1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 5 & 2 & 3 & 2 & 2 & 0 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 3 & 1 & 1 & 0 & 0 & 0 \\ 3 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1.0 & 0.4 & 0.6 & 0.4 & 0.4 & 0.0 & 0.0 & 0.0 \\ 0.2 & 0.4 & 0.6 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.4 & 0.2 & 0.6 & 0.2 & 0.2 & 0.0 & 0.0 & 0.0 \\ 0.6 & 0.2 & 0.4 & 0.2 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

(a) (b) (c)

Figure 2 (a) Block of Transform Coefficients, (b) Block of Amplitude Classifications, and (c) Block of Normalized Amplitude Classifications

The neural network is used during compression of the image to determine the dimensions of the sub-blocks to be retained and encoded for transmission or storage prior to adaptive zigzag reordering. However, it is not required during reconstruction of the image.

EXPERIMENTAL RESULTS

The neural network has been implemented using MATLAB [8] and its Neural Network Toolbox [7]. For the sub-block determination it has been found that a standard backpropagation algorithm has trained the network better than a more sophisticated backpropagation algorithm using momentum and adaptive learning rate. The transform coefficient matrices have been generated using the Independent JPEG Group's software [9].

Several images have been processed using the standard JPEG algorithm, the adaptive zigzag-reordering algorithm, and the neural-network-based sub-block determination. The quality setting, q, controlling scaling of quantization tables, ranges from 10 ('poor' quality) to 90 ('good' quality) in steps of 5. Note that, in accordance with JPEG's EOB symbol, all results apply only to zero coefficients preceding the last nonzero coefficient.

As an example, Fig. 3 depicts the entropy of counts of consecutive zero coefficients for a well-known image, 'Lena', having a resolution of 512×512 pixels. For a given quality setting, and therefore the same peak signal-to-noise ratio (psnr), adaptive zigzag reordering always produces a lower entropy for counts of consecutive zero coefficients than standard JPEG. The sub-block determination using this particular neural-network classifier produces even lower entropy values, but requires a higher quality setting in order to achieve the same psnr, since some information is discarded by the neural network. Therefore, the aim is to find an appropriate compromise between reduction in entropy and increase in quality setting.

CONCLUDING REMARKS

It has been demonstrated that the neural-network-based sub-block classification improves the performance of adaptive zigzag reordering employed in the authors' recent JPEG-like scheme.

Since the sub-block dimensions need to be available for reconstruction, the authors are currently investigating efficient coding schemes for them.

In addition, since the blocks of transform coefficients represent visual information, the generation of training sets that relate better to properties of the human visual system (HVS) is ongoing. Different neural network architectures, for example learning vector quantization (LVQ), are also being investigated.

ACKNOWLEDGEMENT

H.-J. Grosse would like to thank the Department of Electrical and Electronic Engineering of the University of Central Lancashire for sponsoring his research.

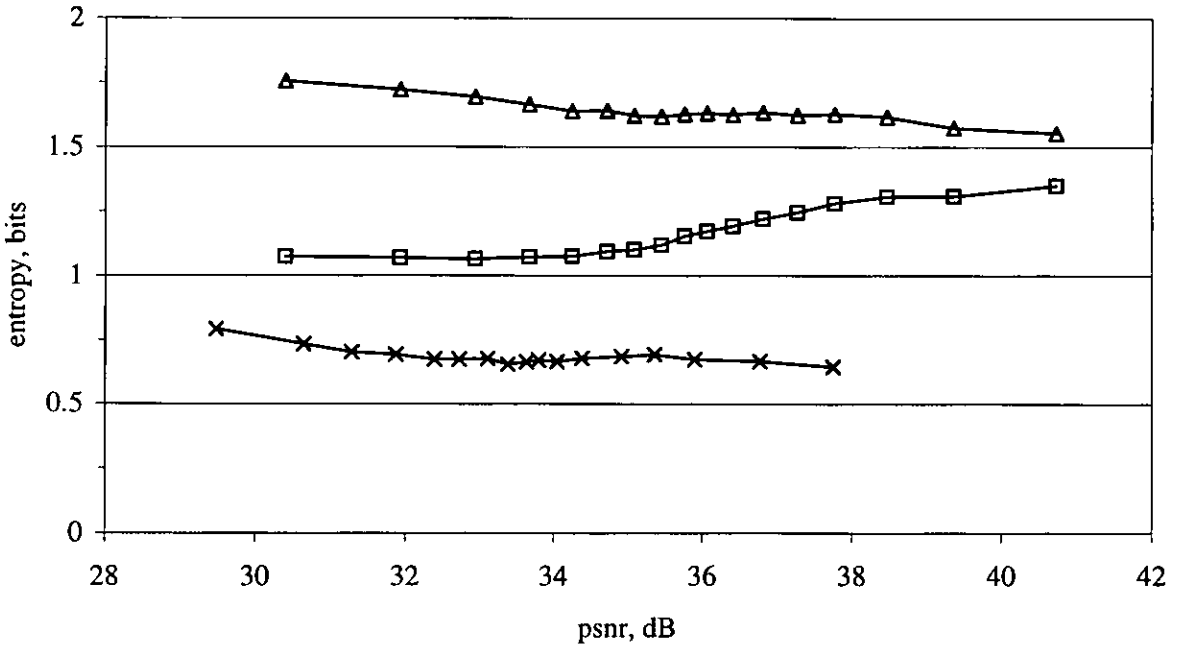


Figure 3 Entropy of Counts of Consecutive Zero Coefficients versus Peak Signal-to-Noise Ratio
—△— Standard JPEG —□— Adaptive Zigzag Reordering
—×— Neural-Network-Based Sub-Block Determination

REFERENCES

1 Wallace, G. K. 1992. The JPEG still picture compression standard. *IEEE transactions on consumer electronics.* Feb 1992. vol. 38, no. 1. pp. xviii-xxxiv.

2 Ahmed, N., Natarajan, T., and Rao, K. R. 1974. Discrete cosine transform. *IEEE transactions on computers.* Jan 1974. vol. C-23, no. 1. pp. 90-93.

3 Grosse, H.-J., Varley, M. R., Terrell, T. J., and Chan, Y. K. 1997. Adaptive zigzag-reordering algorithm for improved coding in JPEG-like image compression schemes. *In:* Second international symposium on digital signal processing. Colchester, UK. Trusty Business Machines Ltd. Venue: London, UK, 22-24 Jul 1997.

4 Grosse, H.-J., Varley, M. R., Terrell, T. J., and Chan, Y. K. 1997. Hardware implementation of versatile zigzag-reordering algorithm for adaptive JPEG-like image compression schemes. *In:* Sixth international conference on image processing and its applications. London, UK. The Institution of Electrical Engineers. Venue: Dublin, Ireland, 14-17 Jul 1997.

5 Huffman, D. A. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE.* Sep 1952. vol. 40, no. 9. pp. 1098-1101.

6 Witten, I. H., Neal, R. M., and Cleary, J. G. 1987. Arithmetic coding for data compression. *Communications of the ACM.* Jun 87. vol. 30, no. 6. pp. 520-540.

7 Demuth, Howard B., and Beale, Mark. 1994. Neural network toolbox user's guide. Natick, Massachusetts, USA. The MathWorks, Inc. Jan 1994.

8 MathWorks. 1994. MATLAB version 4.2. Natick, Massachusetts, USA. The MathWorks, Inc. Oct 1994.

9 Independent JPEG Group. The Independent JPEG Group's software: C source code, release 6a. [Online] Available ftp://ftp.simtel.net/pub/simtelnet/msdos/graphics/jpegsr6a.zip, 07 Feb 1996.

HARDWARE IMPLEMENTATION OF VERSATILE ZIGZAG-REORDERING ALGORITHM FOR ADAPTIVE JPEG-LIKE IMAGE COMPRESSION SCHEMES

H.-J. Grosse[1], M. R. Varley[1], T. J. Terrell[1], and Y. K. Chan[2]

[1] University of Central Lancashire, United Kingdom
[2] City University of Hong Kong, Hong Kong

ABSTRACT

In this paper a hardware implementation of an adaptive technique for reordering of discrete cosine transform (DCT) coefficients, that are used in a variety of transform-based image compression schemes such as JPEG, is described. Efficient reordering is achieved for variable-size rectangular sub-blocks using Boolean operations, that determine the position of the next coefficient to be coded. The algorithm has been developed for implementation in hardware using programmable logic devices (PLDs). The implementation constitutes a Moore state machine with binary inputs representing the number of rows and columns in the sub-block to be reordered. Experimental results are presented which demonstrate the potential advantages of this new technique, in terms of a significant reduction in entropy.

INTRODUCTION

Many image compression schemes, such as the standard JPEG scheme [1], operate by processing small non-overlapping image blocks (usually square $N \times N$ blocks of a fixed size, e.g. 8 pixels × 8 pixels) using a 2-D transform such as the discrete cosine transform (DCT) [2]. Whilst the transform itself is reversible and lossless, it is used to decorrelate the data so that the inter-element correlation in the transform domain is significantly less than that in the spatial domain. The resulting 2-D block of transform coefficients is then processed in the transform domain; in many cases this involves discarding some of the low-value transform coefficients to reduce the amount of data to be transmitted or stored, which causes a loss of information.

In the authors' new JPEG-like scheme the $N \times N$ block of quantized and thresholded transform coefficients is modified to yield a smaller sub-block of coefficients to be coded. This sub-block is not necessarily square, but is rectangular with a height and width between 1 and N. Adaptive zigzag reordering, which in itself is a fully reversible process, produces a 1-D array of coefficients which is then converted into an intermediate symbol sequence, each symbol representing the number of zero coefficients preceding the current nonzero coefficient, the amplitude classification and the actual amplitude of the nonzero coefficient. This facilitates entropy coding by placing low-frequency coefficients, that are more likely to be nonzero, before high-frequency coefficients.

In this paper, a version of the algorithm that has been developed for implementation in hardware using programmable logic devices (PLDs) is described. The implementation constitutes a Moore state machine with binary inputs representing the number of rows and columns in the sub-block to be reordered. The state machine steps through the appropriate number of states in sequence, and generates outputs corresponding to the row and column indices of each element in turn for a zigzag scan path. Since the implementation employs the parallel hardware of the PLDs, the appropriate operations are mapped directly into Boolean operations, implemented using logic gates, instead of nested decisions which would be used in a software implementation. This enables fast reordering to be achieved prior to coefficient coding.

Experimental results are presented for four images, which demonstrate the potential advantages of the new adaptive scheme over the standard JPEG scheme, in terms of a significant reduction in entropy.

DETERMINATION OF SUB-BLOCKS

The proposed algorithm is transparent and lossless, and identifies the smallest possible rectangle to include all nonzero transform coefficients after quantization, thus adapting to the particularities of every block. As an example, Fig. 1 depicts an 8×8 block of quantized

$$\left[\begin{array}{cccccccc}
-26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\
1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\
-4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
\hdashline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right]$$

Figure 1 8×8 Block of Transform Coefficients

transform coefficients, with the corresponding 4×5 sub-block indicated by the dotted line. Since the sub-blocks generally have different heights and widths depending on the specific content of the corresponding block, the dimensions of the sub-block, in the above example 4×5, need to be retained in order to traverse the scan path correctly using the new algorithm.

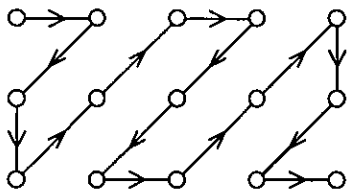
VERSATILE ZIGZAG-REORDERING ALGORITHM

A matrix, $A(L, M)$, of L rows by M columns can be defined as

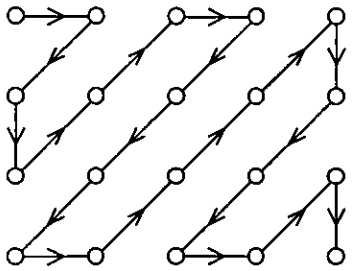
$$A(L,M) = \begin{bmatrix} a(1,1) & a(1,2) & . & a(1,M) \\ a(2,1) & a(2,2) & . & a(2,M) \\ . & . & a(l,m) & . \\ a(L,1) & a(L,2) & . & a(L,M) \end{bmatrix} \quad (1)$$

with $1 \le l \le L$ and $1 \le m \le M$.

One of many possible scan paths involves zigzag reordering as shown in Fig. 2 for two examples; the elements always succeed a neighbouring element.



(a)



(b)

Figure 2 Zigzag Scan Path for (a) 3×5, and (b) 4×5 Matrices

The scan path depends on the dimensions, L and M, of the matrix. As the dimensions of the matrix are often known in advance, 8×8 for blocks in JPEG for example, the matrix can easily be traversed referring to a single scan path. However, applications that allow matrices of different and variant dimensions need to produce scan paths tailored to the dimensions of the matrices being used in order to reduce complexity.

The next element's position (l^+, m^+), and therefore the whole scan path, is determined through Boolean expressions operating on the current element's position (l, m) and the dimensions of the sub-block, L and M. For zigzag reordering five parameters have been defined:

$R1$ indicates whether the current element is in the first row

$$R1 = \begin{cases} 1 & for\ l = 1 \\ 0 & otherwise \end{cases} \quad (2)$$

RL indicates whether the current element is in the last row

$$RL = \begin{cases} 1 & for\ l = L \\ 0 & otherwise \end{cases} \quad (3)$$

$C1$ indicates whether the current element is in the first column

$$C1 = \begin{cases} 1 & for\ m = 1 \\ 0 & otherwise \end{cases} \quad (4)$$

CM indicates whether the current element is in the last column

$$CM = \begin{cases} 1 & for\ m = M \\ 0 & otherwise \end{cases} \quad (5)$$

P indicates whether the sum of the row index l and the column index m is odd

$$P = \begin{cases} 1 & if\ (l + m)\ is\ odd \\ 0 & otherwise \end{cases} \quad (6)$$

For different scan paths other parameters will be required.

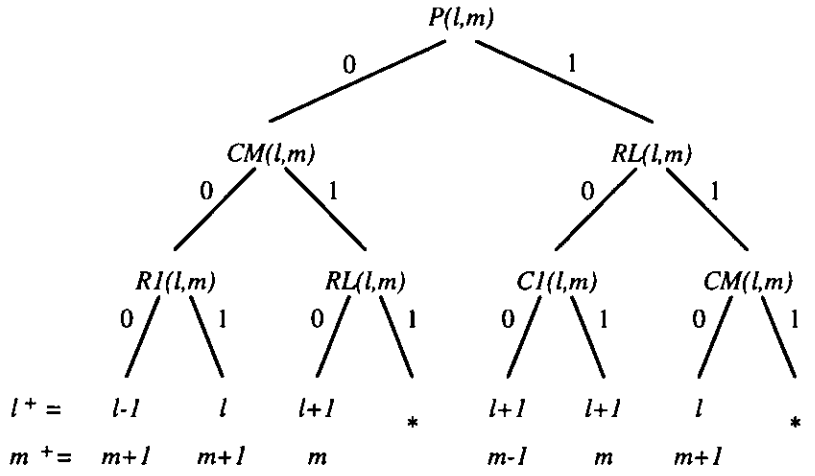
The following expressions determine the changes in row and column indices:

$$l^+ = l - 1 \text{ if } \left[\overline{R1} \cdot \overline{CM} \cdot \overline{P}\right] \text{ is true} \quad (7)$$

$$m^+ = m + 1 \text{ if } \left[\left(R1 \cdot \overline{CM} \cdot \overline{P}\right) + \left(RL \cdot \overline{CM} \cdot P\right) + \left(\overline{R1} \cdot \overline{CM} \cdot \overline{P}\right)\right] \text{ is true} \quad (8)$$

$$l^+ = l + 1 \text{ if } \left[\left(\overline{RL} \cdot CM \cdot \overline{P}\right) + \left(\overline{RL} \cdot C1 \cdot P\right) + \left(\overline{RL} \cdot \overline{C1} \cdot P\right)\right] \text{ is true} \quad (9)$$

$$m^+ = m - 1 \text{ if } \left[\overline{RL} \cdot \overline{C1} \cdot P\right] \text{ is true} \quad (10)$$



* indicates scan complete

Figure 3 Binary Decision Tree for Zigzag Reordering

The above expressions are given in sum-of-products form, but can be rearranged as required. A binary decision tree, as shown in Fig. 3, may be used to combine equations (7) to (10).

HARDWARE IMPLEMENTATION USING PROGRAMMABLE LOGIC DEVICES

As an illustration of how the versatile zigzag-reordering algorithm can be mapped into dedicated hardware, a Moore state machine has been implemented with six binary inputs representing the size of the sub-block to be reordered. Three of the binary inputs are used to specify the number of rows: 000 represents a sub-block containing one row, 001 represents a sub-block containing two rows etc. up to 111 for a sub-block with eight rows. Similarly, the number of columns is specified. Whilst the algorithm as described above can be applied to sub-blocks of any size, this particular hardware implementation allows all 64 sub-block sizes from 1×1 to 8×8. The state machine has six outputs that represent the row and column indices, l and m, of the current element in the scan path in 3 bits each. A reset signal (*RESET*) is used to initialize the row and column indices, l and m, to zero; corresponding to the first element in the sequence regardless of the sub-block size L and M. The appropriate zigzag scan sequence is then generated in synchronization to a clock signal (*CLK*). After the scan is complete, i. e. when $l = L$ and $m = M$, a signal (*DONE*) is asserted to indicate completion of the scan of the current sub-block, and the row and column indices, l and m, are automatically returned to zero in readiness for the zigzag scan of the next sub-block.

The hardware implementation involves two stages; each of which is mapped into a separate GAL16V8 device [3], which is a generic array logic PLD with a user-programmable AND array, a fixed OR array, and an output stage employing output logic macro-cells (OLMCs). The device has eight dedicated inputs and eight user-configurable pins, each of which may be configured individually as input, combinational output, or registered output within the appropriate OLMC. Registered outputs are also fed back into the device's AND array enabling a state machine to be implemented on a single device.

The two stages in this implementation are:

Stage A. This stage determines, according to equations (2) to (6), P, RI, RL, CI, and CM from the present values of the row and column indices, l and m, and the sub-block size as defined by L and M. This is a purely combinational stage with twelve inputs (L, M, l, and m consisting of 3 bits each) and five outputs (P, RI, RL, CI, and CM).

Stage B. This stage determines the next row and column indices, l^+ and m^+, from the present indices, l and m, and the five outputs of the preceding stage using a clock signal (*CLK*) to control the timing of the zigzag-scan-sequence generation, and a reset signal (*RESET*) to initialize the row and column indices to zero for the first scan. The outputs from stage B are the two 3-bit indices, l and m, which are implemented as registered outputs, enabling them to be fed back internally to the PLD's AND array. Additionally, a *DONE* signal is available from stage B to indicate that the zigzag scan of the current $L \times M$ sub-block is complete.

Each stage is mapped into a separate GAL16V8 PLD, and the state machine is implemented by interconnecting the two PLDs as shown in Fig. 4.

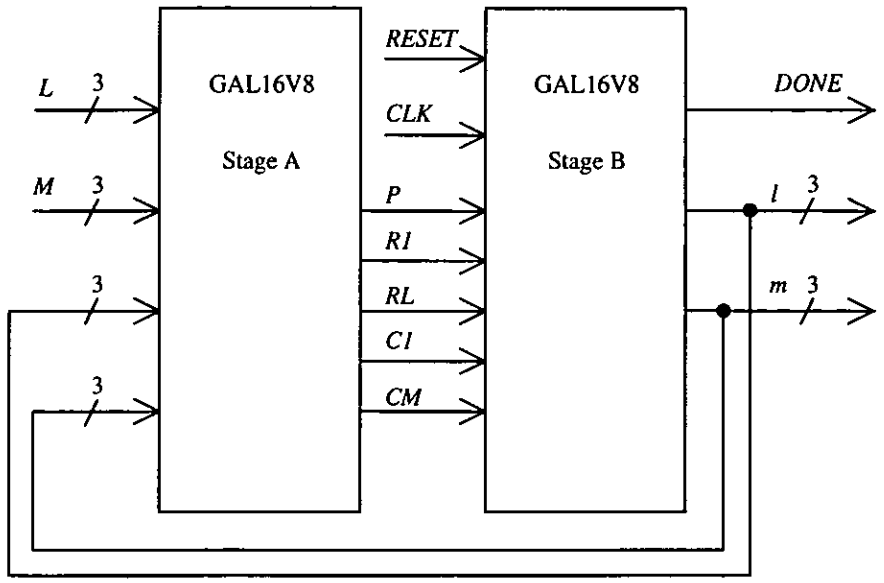


Figure 4 Implementation of Zigzag Scan Path using GAL16V8 PLDs

The fusemaps for the two devices have been created using the development tool Tango-PLD [4], that allows specification of the functionality of each device at a high level using the C-like Tango Design Language (TDL). A simple TDL file has been used to implement stage A of the state machine according to equations (2) to (6). Stage B has been implemented using a TDL file describing the binary decision tree shown in Fig. 3.

It has been found that the stage A implementation requires up to six product lines per output and readily fits within a GAL16V8 device. Since, in practice, a row or column index is never decremented from 000 or incremented from 111, "don't care" states can be used for these cases in order to reduce the number of product lines per output. Incorporating these considerations into the TDL description of the binary decision tree enables the state machine of stage B to be implemented on a single GAL16V8 device with the eight available product lines fully utilized for some of the registered outputs.

Each device has been individually tested to verify its correct operation, and the entire state machine, consisting of the two GAL16V8 devices interconnected as shown in Fig. 4, has also been tested to ensure that the zigzag scan paths are correctly generated.

ENTROPY REDUCTION VIA ADAPTIVE ZIGZAG REORDERING

In the standard JPEG scheme the 1-D matrix of zigzag-reordered coefficients is represented through an intermediate symbol sequence, each symbol representing the number of zero coefficients preceding the current nonzero coefficient, the amplitude classification and the actual amplitude of the nonzero coefficient [1]. An end-of-block symbol (EOB) terminates the block after the last nonzero coefficient. Huffman coding [5], or arithmetic coding [6], can be used to convert the symbols to a continuous data stream according to the JPEG specification.

EXPERIMENTAL RESULTS

The images 'Cameraman' and 'Lena256' (both with a resolution of 256×256), and 'F-16' and 'Lena512' (both with a resolution of 512×512) have been processed using the standard and the adaptive zigzag-reordering algorithms. The transform coefficient matrices have been generated using the Independent JPEG Group's software [7]. The quality setting, q, controlling scaling of quantization tables, ranges from 10 ('poor' quality) to 90 ('good' quality) in steps of 5. Note that, in accordance with JPEG's EOB symbol, all results apply only to zero coefficients preceding the last nonzero coefficient.

It has been observed that in all cases the entropy of counts of consecutive zero coefficients for adaptive zigzag-reordered scan paths is lower than that for the standard 8×8 zigzag scan path. Figure 5 summarizes the percentage entropy reduction for the four images. For higher quality settings the number of nonzero coefficients increases, therefore the sub-block dimensions approach the standard 8×8 block dimensions more frequently. However, for the images analysed using 'medium' quality settings $(q = 30 \text{ to } 70)$, a significant entropy reduction of at least 15 % has been obtained.

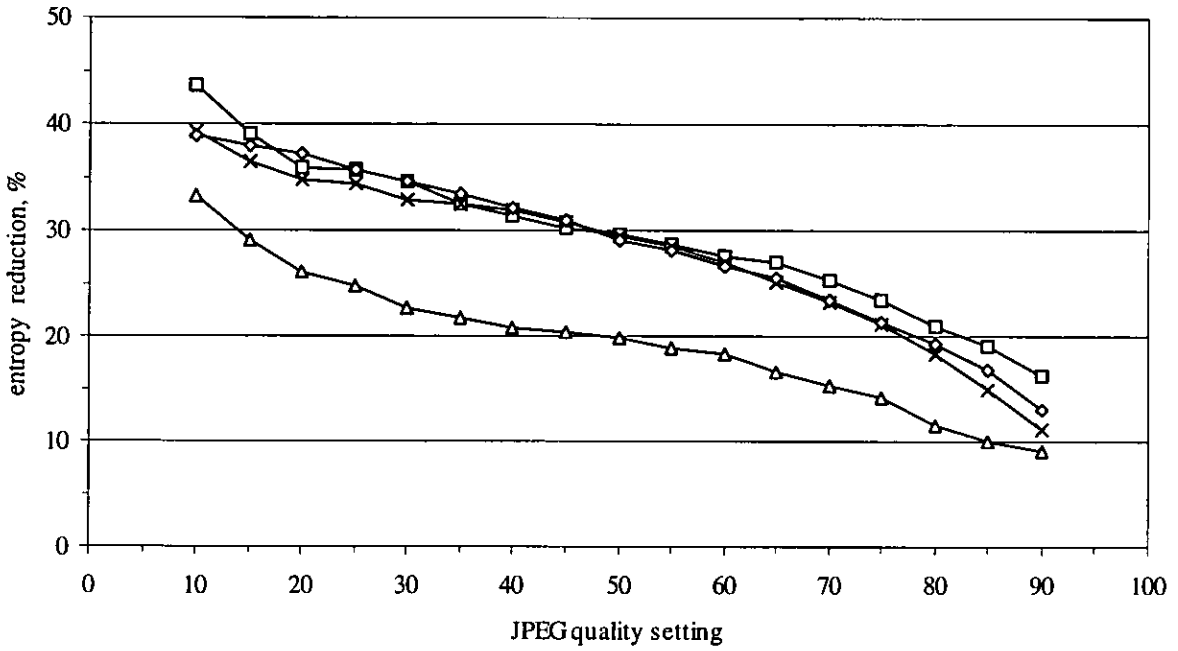


Figure 5 Entropy reduction of counts of consecutive zero coefficients versus JPEG quality setting for four images
—△— Cameraman —□— F-16 —×— Lena256 —◇— Lena512

CONCLUDING REMARKS

It has been demonstrated that the zigzag-reordering algorithm, consistently giving a significant reduction in the entropy of counts of consecutive zero coefficients over a wide range of quality settings, can be implemented in hardware. The implementation using two GAL16V8 PLDs has been developed with the Tango-PLD development system.

The versatility of the zigzag-reordering algorithm itself also supports the use of different block sizes for different regions of an image, for example 4×4 blocks for image regions containing significant detail and 16×16 blocks for background regions. The latter block size would, of course, require a different hardware implementation.

Since the sub-block dimensions need to be available for reconstruction, the authors are currently investigating efficient coding schemes for them. The algorithm is also being applied in research on discarding isolated nonzero coefficients using neural-network-based sub-block classification to further reduce the size of some sub-blocks.

ACKNOWLEDGEMENT

H.-J. Grosse would like to thank the Department of Electrical and Electronic Engineering of the University of Central Lancashire for sponsoring his research.

REFERENCES

1 Wallace, G. K. 1992. The JPEG still picture compression standard. IEEE transactions on consumer electronics. Feb 1992. vol. 38, no. 1. pp. xviii-xxxiv.

2 Ahmed, N., Natarajan, T., and Rao, K. R. 1974. Discrete cosine transform. IEEE transactions on computers. Jan 1974. vol. C-23, no. 1. pp. 90-93.

3 GAL data book. 1990. Lattice Semiconductor Corporation. Hillsboro, Oregon, USA.

4 Tango-PLD: reference manual. 1989. ACCEL Technologies, Inc. San Diego, California, USA.

5 Huffman, D. A. 1952. A method for the construction of minimum-redundancy codes. Proceedings of the IRE. Sep 1952. vol. 40, no. 9. pp. 1098-1101.

6 Witten, I. H., Neal, R. M., and Cleary, J. G. 1987. Arithmetic coding for data compression. Communications of the ACM. Jun 87. vol. 30, no. 6. pp. 520-540.

7 Independent JPEG Group. The Independent JPEG Group's software: C source code, release 6a. [Online] Available ftp://ftp.simtel.net/pub/simtelnet/msdos/graphics/jpegsr6a.zip, 07 Feb 1996

Adaptive Zigzag-Reordering Algorithm for Improved Coding in JPEG-like Image Compression Schemes

H.-J. Grosse, M. R. Varley, and T. J. Terrell
Department of Electrical and Electronic Engineering,
University of Central Lancashire,
Preston, Lancashire, PR1 2HE, United Kingdom

Y. K. Chan
Department of Computer Science,
City University of Hong Kong,
83 Tat Chee Avenue, Kowloon Tong, Kowloon, Hong Kong

Abstract

In this paper an adaptive technique for reordering of discrete cosine transform (DCT) coefficients, that are used in a variety of transform-based image compression schemes such as JPEG, is described. Efficient reordering is achieved for variable-size rectangular sub-blocks using an innovative binary decision tree which determines the position of the next coefficient to be coded. Experimental results are presented which demonstrate the potential advantages of this new technique, in terms of a significant reduction in entropy.

1 Introduction

Many image compression schemes, such as the standard JPEG scheme [1], operate by processing small non-overlapping image blocks (usually square $N \times N$ blocks of a fixed size, e.g. 8 pixels × 8 pixels) using a 2-D transform such as the discrete cosine transform (DCT) [2]. Whilst the transform itself is reversible and lossless, it is used to decorrelate the data so that the inter-element correlation in the transform domain is significantly less than that in the spatial domain. The resulting 2-D block of transform coefficients is then processed in the transform domain; in many cases this involves discarding some of the low-value transform coefficients to reduce the amount of data to be transmitted or stored, which causes a loss of information. The remaining coefficients are then coded, usually in a specific order corresponding, for example, to increasing spatial frequency. In JPEG schemes zigzag reordering is applied to the quantized and thresholded block of coefficients, producing a 1-D array of coefficients. This facilitates entropy coding by placing low-frequency coefficients, that are more likely to be nonzero, before high-frequency coefficients.

The new technique differs in that the $N \times N$ block of transform coefficients is modified to yield a smaller sub-block of coefficients to be coded. This sub-block is not necessarily square, but is rectangular with a height and width between 1 and N. Since the sub-blocks generally have different heights and widths depending on the specific content of the corresponding block, the reordering is no longer a straightforward task. The algorithm described in this paper produces a 1-D array containing the reordered coefficients from the variable-size sub-block, using an appropriate zigzag scan path. This algorithm determines the required scan path 'on the fly' using a binary decision tree, and can be applied to rectangular blocks of any height and width.

Experimental results are presented for four images, which demonstrate the potential advantages, in terms of a significant reduction in entropy, over the standard JPEG scheme.

2 Determination of sub-blocks

The proposed algorithm is transparent and lossless, and identifies the smallest possible rectangle to include all nonzero transform coefficients after quantization, thus adapting to the particularities of every block. As an example, Fig. 1 depicts an 8×8 block of quantized transform coefficients, with the corresponding 4×5 sub-block indicated by the dotted line. Since the sub-blocks generally have different heights and widths depending on

$$\left[\begin{array}{cccccccc} -26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\ 1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Figure 1 8×8 Block of Transform Coefficients

the specific content of the corresponding block, the dimensions of the sub-block, in the above example 4×5, need to be retained in order to traverse the scan path correctly using the new algorithm.

3 Versatile zigzag-reordering algorithm

A matrix, $A(L, M)$, of L rows by M columns can be defined as

$$A(L,M) = \begin{bmatrix} a(1,1) & a(1,2) & . & a(1,M) \\ a(2,1) & a(2,2) & . & a(2,M) \\ . & . & a(l,m) & . \\ a(L,1) & a(L,2) & . & a(L,M) \end{bmatrix} \quad (1)$$

with $1 \le l \le L$ and $1 \le m \le M$.

One of many possible scan paths involves zigzag reordering as shown in Fig. 2 for two examples; the elements always succeed a neighbouring element.

The next element's position in the scan path, (l^+, m^+), and therefore the whole scan path, is determined using binary decisions based on the current element's position, (l, m), and the dimensions of the sub-block, L and M. For zigzag reordering five parameters have been defined:

$R1$ indicates whether the current element is in the first row

$$R1 = \begin{cases} 1 & for\ l = 1 \\ 0 & otherwise \end{cases} \quad (2)$$

RL indicates whether the current element is in the last row

$$RL = \begin{cases} 1 & for\ l = L \\ 0 & otherwise \end{cases} \quad (3)$$

$C1$ indicates whether the current element is in the first column

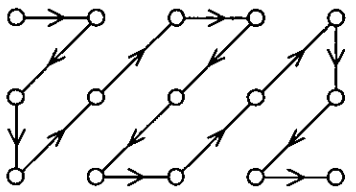
$$C1 = \begin{cases} 1 & for\ m = 1 \\ 0 & otherwise \end{cases} \quad (4)$$

CM indicates whether the current element is in the last column

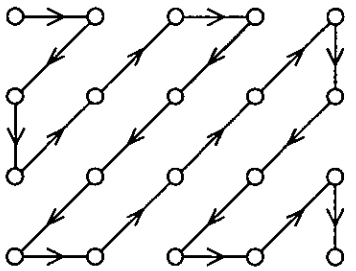
$$CM = \begin{cases} 1 & for\ m = M \\ 0 & otherwise \end{cases} \quad (5)$$

P indicates whether the sum of the row index l and the column index m is odd

$$P = \begin{cases} 1 & if\ (l + m)\ is\ odd \\ 0 & otherwise \end{cases} \quad (6)$$



(a)



(b)

Figure 2 Zigzag Scan Path for (a) 3×5, and (b) 4×5 Matrices

Figure 3 depicts a decision tree which may be used to find the next element's position, (l^+, m^+), with three tests operating on the parameters defined in equations (2) - (6). First the parity parameter $P(l, m)$ is tested, and depending on this result, either the last-column parameter $CM(l, m)$ for $P(l, m) = 0$, or the last-row parameter $RL(l, m)$ for $P(l, m) = 1$, is tested. Subsequent decisions are then taken as specified in the decision tree. Note that in the case for which both $l^+ = l$ and $m^+ = m$ the full scan of the $L \times M$ sub-block is complete.

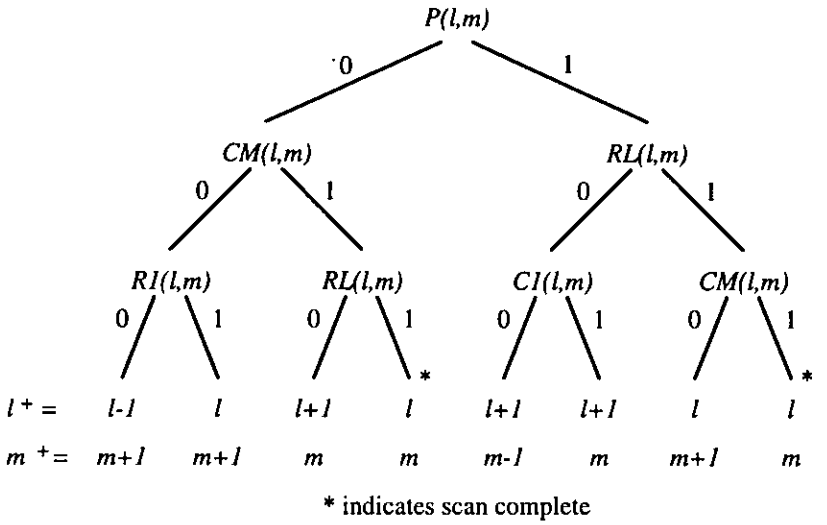


Figure 3 Binary Decision Tree for Zigzag Reordering

4 Entropy reduction via adaptive zigzag reordering

In the standard JPEG scheme the 1-D matrix of zigzag-reordered coefficients is represented through an intermediate symbol sequence, each symbol representing the number of zero coefficients preceding the current nonzero coefficient, the amplitude classification and the actual amplitude of the nonzero coefficient [1]. An end-of-block symbol (EOB) terminates the block after the last nonzero coefficient. Huffman coding [3], or arithmetic coding [4], can be used to convert the symbols to a continuous data stream according to the JPEG specification.

The new adaptive technique reduces the counts of consecutive zero coefficients; it modifies the distribution of the counts of consecutive zero coefficients and also reduces the number of different counts. This results in an overall reduction in entropy for the counts of consecutive zero coefficients, which may be exploited to give an increased compression ratio.

5 Experimental results

The images 'Cameraman' and 'Lena256' (both with a resolution of 256×256), and 'F-16' and 'Lena512' (both with a resolution of 512×512) were processed using the standard and the adaptive zigzag-reordering algorithms. The transform coefficient matrices were generated using the Independent JPEG Group's software [5]. The quality setting, q, controlling scaling of quantization tables, ranges from 10 ('poor' quality) to 90 ('good' quality) in steps of 5. Note that, in accordance with JPEG's EOB symbol, all results apply only to zero coefficients preceding the last nonzero coefficient.

It was observed that in all cases the entropy for adaptive zigzag-reordered scan paths is lower than that for the standard 8×8 zigzag scan path. As an example, Fig. 4 depicts the entropy of counts of consecutive zero coefficients for the image 'Lena512'. It is clear that the adaptive algorithm consistently produces a lower entropy, indicating the potential for improved coding gain. Figure 5 summarizes the percentage entropy reduction for all four images. For higher quality settings the number of nonzero coefficients increases, therefore the sub-block dimensions approach the standard 8×8 block dimensions more frequently. However, for the images analysed using 'medium' quality settings $(q = 30 \text{ to } 70)$, a significant entropy reduction of at least 15 % was obtained.

6 Concluding remarks

It has been demonstrated that the new zigzag-reordering algorithm consistently gives a significant reduction in the entropy of counts of consecutive zero coefficients over a wide range of quality settings. The versatility of the zigzag-reordering algorithm also supports the use of different block sizes for different regions of an image, for example 4×4 blocks for image regions containing significant detail and 16×16 blocks for background regions.

A hardware implementation of the decision tree has been developed using dedicated logic [6]. The algorithm

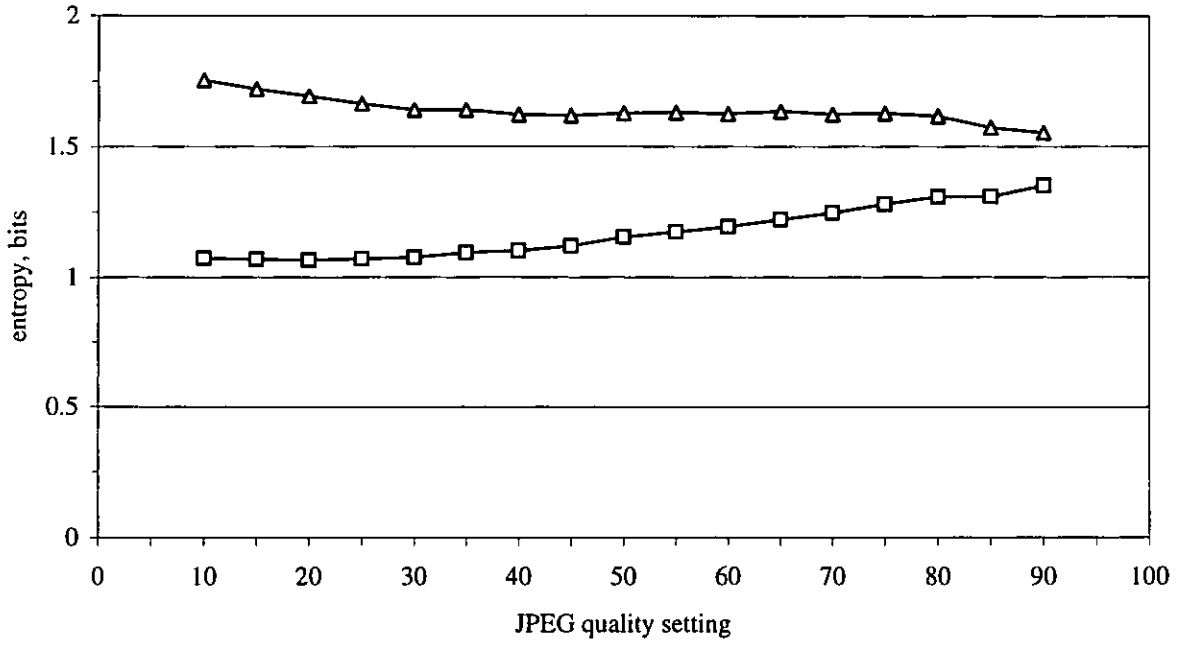


Figure 4 Entropy of counts of consecutive zero coefficients versus JPEG quality setting for Lena512
Standard Zigzag Reordering Adaptive Zigzag Reordering

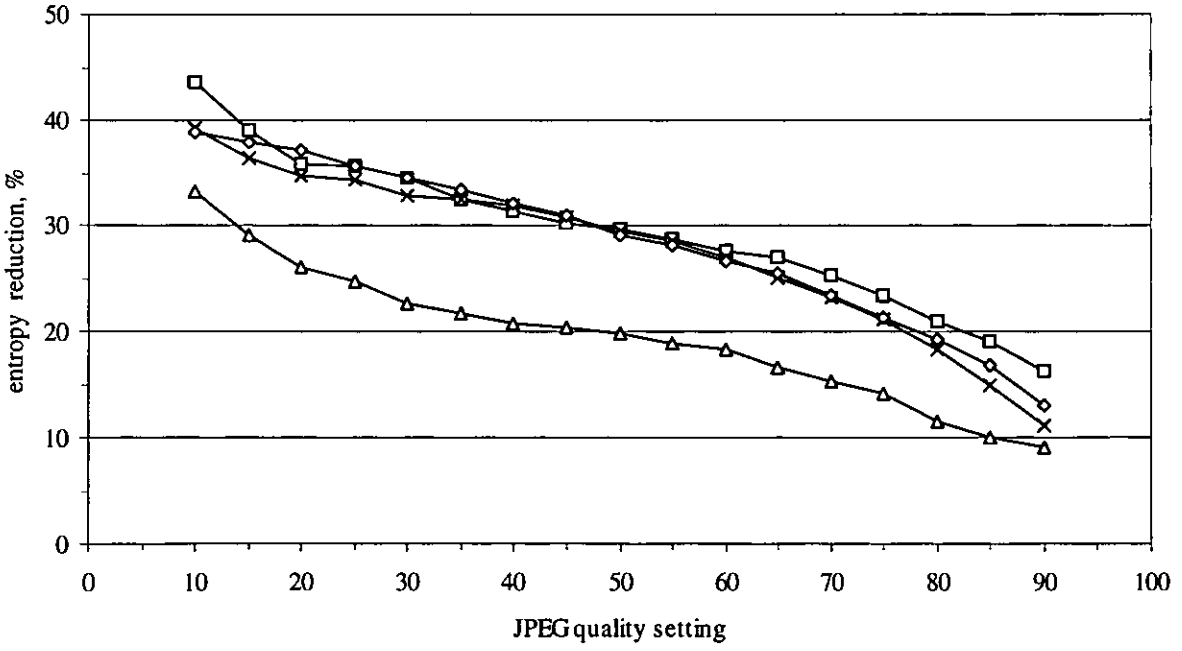


Figure 5 Entropy reduction of counts of consecutive zero coefficients versus JPEG quality setting for four images
Cameraman F-16 Lena256 Lena512

is also being applied in research on discarding isolated nonzero coefficients to generate even smaller sub-blocks [7].

A block of size $N \times N$ yields a sub-block of one of N^2 possible sizes, thus introducing an overhead of $2 \log_2 N$ bits per block for a simple fixed-length code. For $N = 8$, 64 symbols are necessary to uniquely identify every possible sub-block size, generating an overhead of 6 bits per block. It was found that even after employing additional entropy coding, such as Huffman coding [3] or arithmetic coding [4], this size of overhead is prohibitive. However, the sub-block size is correlated with the number of coefficients allowing more efficient encoding. The number of coefficients along the scan path, i. e. the scan path length, is known and can be evaluated: it varies between 1 and N^2, and provides some information suitable to narrow down the number of sub-block sizes possible for a particular number of coefficients. The authors are currently investigating improved coding schemes for the sub-block sizes.

7 Acknowledgement

H.-J. Grosse would like to thank the Department of Electrical and Electronic Engineering of the University of Central Lancashire for sponsoring his research.

8 References

1 Wallace, G. K. 1992. The JPEG still picture compression standard. *IEEE transactions on consumer electronics.* Feb 1992. vol. 38, no. 1. pp. xviii-xxxiv.

2 Ahmed, N., Natarajan, T., and Rao, K. R. 1974. Discrete cosine transform. *IEEE transactions on computers.* Jan 1974. vol. C-23, no. 1. pp. 90-93.

3 Huffman, D. A. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE.* Sep 1952. vol. 40, no. 9. pp. 1098-1101.

4 Witten, I. H., Neal, R. M., and Cleary, J. G. 1987. Arithmetic coding for data compression. *Communications of the ACM.* Jun 87. vol. 30, no. 6. pp. 520-540.

5 Independent JPEG Group. The Independent JPEG Group's software: C source code, release 6a. [Online] Available ftp://ftp.simtel.net/pub/simtelnet/msdos/graphics/jpegsr6a.zip, 07 Feb 1996.

6 Grosse, H.-J., Varley, M. R., Terrell, T. J., and Chan, Y. K. 1997. Hardware implementation of versatile zigzag-reordering algorithm for adaptive JPEG-like image compression schemes. *In:* Sixth international conference on image processing and its applications. London, UK. The Institution of Electrical Engineers. Venue: Dublin, Ireland, 14-17 Jul 1997.

7 Grosse, H.-J., Varley, M. R., Terrell, T. J., and Chan, Y. K. 1997. Sub-block classification using a neural network for adaptive zigzag reordering in JPEG-like image compression scheme. *In:* Neural and fuzzy systems: design, hardware and applications. London, UK. The Institution of Electrical Engineers. Venue: London, UK, 09 May 1997.