SIGNIFICANCE-LINKED CONNECTED COMPONENT ANALYSIS PLUS

A Dissertation

presented to

the Faculty of the Graduate School

at the University of Missouri - Columbia

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

Xiaobo Jiang

Dr. Xinhua Zhuang, Dissertation Supervisor

MAY 2018

The undersigned, appointed by the dean of the graduate school,

have examined the thesis entitled

SIGNIFICANCE-LINKED CONNECTED COMPONENT ANALYSIS PLUS

presented by Xiaobo Jiang

a candidate for the degree of

Doctor of Philosophy

and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Zhuang, Xinhua

Dr. Zhao, Yunxin

Dr. Uhlmann, Jeffrey

Dr. He, Zhihai

ACKNOWLEDGEMENTS

This dissertation could not have been finished without my advisor, Professor Xinhua Zhuang, who continuously guides and encourages me throughout my doctoral program. I am deeply influenced by his scientific methodology and dedicated working attitude. My appreciation also goes to the member of advisory committee, Dr. Yunxin Zhao, Dr. Jeffrey Uhlmann, and Dr. Zhihai He. They guided me through the doctoral process, never accepting less than my best efforts.

I am also grateful to my friends in the same lab, Bo Song and Cheng Zhang, for their brilliant thoughts and friendship during the time.

This dissertation is dedicated to my family. Special gratitude is extended to my wife, for being my solid backing and constant support in my life. Particular thanks to my parents, for their upbringing and support. Last but not the least, thank my daughter for the happiness she brings.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

An image coding algorithm, SLCCA Plus, is introduced in this dissertation. SLCCA Plus is a wavelet-based subband coding method. In wavelet-based subband coding, the input images will go through a wavelet transform and be decomposed into wavelet subband pyramids. Then the characteristics of the wavelet coefficients within and among subbands will be utilized to removing the redundancy. The rest information will be organized and go through entropy encoding. SLCCA Plus contains a series improvement method to the SLCCA. Before SLCCA, there are three top-ranked wavelet image coders. Namely, Embedded Zerotree Wavelet coder (EZW), Morphological Representation of Wavelet Date (MEWD), and Set Partitioning in Hierarchical Trees (SPIHT). They exploit either inter-subband relation among zero wavelet coefficients or within-subband clustering. SLCCA, on the other hand, outperforms these three coders by exploring both the inter-

subband coefficients relations and within-subband clustering of significant wavelet coefficients.

SLCCA Plus strengthens SLCCA in the following aspects: Intelligence quantization, enhanced cluster filter, potential-significant shared-zero, and improved context models. The purpose of the first three improvements is to remove redundancy information further while keeping the image error as low as possible. As a result, they achieve a better trade-off between bit cost and image quality. Moreover, the improved context lowers the entropy by refining the classification of symbols in cluster sequence and magnitude bit-planes. Lower entropy means the adaptive arithmetic coding can achieve a better coding gain.

For performance evaluation, SLCCA Plus is compared to SLCCA and JPEG2000. On average, SLCCA Plus achieves 7% bit saving over JPEG 2000 and 4% over SLCCA. The results comparison shows that SLCCA Plus shows more texture and edge details at a lower bitrate.

CHAPTER I: INTRODUCTION

Image compression is always an active research area. Recently, it attracts many researcher's attention, due to the widely used high-resolution images and videos. People have different preferences for visual effects, but one thing is not going to change: the clearer, the better. This preference makes high-resolution images highly demanded. With the rapid development of camera sensor and display technology, high-resolution video and images have been popularized, and this makes a lot challenge for storage and transmission technologies. More efficient image compression methods are needed.

Digital images had been developed quickly since the 1960s and 1970s, thanks to the invention of CCD image sensor. Unlike a film image, a digit image is constituted from many pixels. The pixel is the smallest addressable element of an image. Moreover, the pixels are arranged in rows and columns. Thus, the digital images are also referred to as raster images.

Each pixel is corresponded to an integer number to represent the level of brightness. The integer value is stored in the drive in binary form. Most commonly, we use one byte to store a pixel, which means the pixel value has a range from 0 to 255 (Fig. 1.1 [30]).

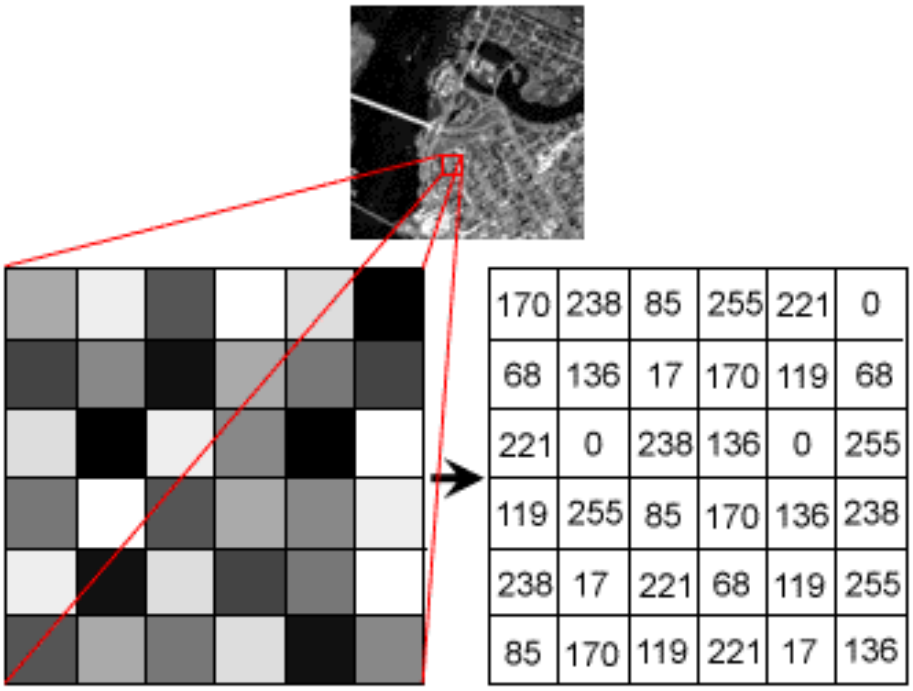

Figure 1.1 A digital image and the pixel value in it

Apparently, a digital image is better with a higher density of the pixels, or high-resolution. However, high-resolution will lead to substantial storage space and slow transfer speed. Fortunately, the digital image usually contains a significant amount of redundancy information, especially after we transfer the image to the frequency domain. Due to the limitation of human eye's sensitivity, a considerable amount of frequency information will not

affect image's quality (sometimes it is only the noise from the camera sensor). Thus, we can eliminate the redundancy. On the other hand, thanks to Shannon's entropy theory, we can encode the needed information in the image into a much smaller size.

The image compression is the art and science aim at taking advantage of this redundancy to efficient encoding the images with reduced size. The image compression process usually contains three main steps: 1) preprocess the image to expose as much redundancy as possible, 2) organize the information to eliminate the redundancy, and 3) encode the needed information.

Based on how we prepare the image to expose and explore its redundancy information, image compression has two essential categories: 1) Wavelet-based subband coding [7, 23-25], and 2) Hybrid coding of block prediction [7, 21, 22] and discrete cosine transform (DCT).

Wavelet-based image coder has delivered a much higher coding gain over the classical DCT-based coder such as JPEG. More recently, through using a delicate yet innovative block prediction scheme before DCT, HEVC Intra [26] coding has delivered an impressive coding gain.

In wavelet-based subband coding, the input images will go through a wavelet transform and be decomposed into wavelet subband pyramids. Then the

characteristics of the wavelet coefficients within and among subbands will be utilized for removing the redundancy and encoding. There are several renowned wavelet codecs: Embedded Zerotree Wavelet Coder (EZW) [1], Set Partitioning in Hierarchical Trees (SPIHT) [2], Morphological Representation of Wavelet Date (MEWD) [3], Significance-Linked Connected Component Analysis (SLCCA) [4] and JPEG 2000 [5]. Both EZW and SPIHT exploit the cross-subband insignificant coefficient correlation with conventional tree structures while SPIHT enhances EZW by partitioning the cross-subband tree structure. MRWD uses within-subband clustering of significant wavelet coefficients. SLCCA outperforms EZW and SPIHT and strengthens MRWD by using irregularly shaped structures (clusters) to map the significant coefficients and linking the clusters across subbands. The relatively widespread coder JPEG 2000 represents the wavelet coefficient values (include signs and magnitudes) in bit-planes and performs entropy coding directly to the bit-planes. By contrast, SLCCA embeds the coefficients' signs in clusters and only codes significant coefficients' magnitudes bit-plane wise. Comparatively, SLCCA shows better performance among these four codecs. This dissertation is the continuing and enhancing work of SLCCA.

The core of SLCCA is a unique data organization and representation (DOR) strategy for DWT images [6]. It is developed at University of Missouri by Bing-Bing Chai, Jozsef Vass and Xinhua Zhuang in 1996. Two characters of

the SLCCA's strategy are 1) within-subband clustering of significant wavelet coefficients and 2) Cross-subband significance linkage between a parent cluster and a child cluster. The advantage of SLCCA embodied in it can achieve high performance even at the meager bit rate. The detailed introduction of SLCCA is in Chapter III.

The work of SLCCA Plus in this dissertation is an upgrading based on SLCCA. The upgrades include intelligent quantization with an adaptive threshold, enhanced cluster filter, potential significant shared-zero, and improved context models. A detailed introduction is in Chapter IV. Chapter II introduces the background knowledge and main steps of Wavelet-based subband coding. Chapter V is the conclusion.

CHAPTER II: BACKGROUND KNOWLEDGE OF DIGITAL IMAGES AND IMAGE COMPRESSION

As aforementioned, image compression process usually contains three main steps: 1) preprocess the image to expose as much redundancy as possible, 2) organize the information to eliminate the redundancy, and 3) encode the needed information using entropy coding. In wavelet-based subband coding, these three steps can be instanced as the following process in Fig. 2.1:

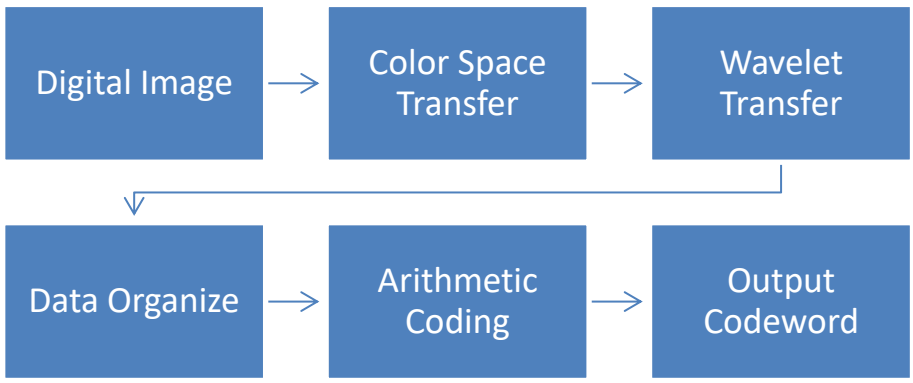

Figure 2.1. Wavelet-based subband coding main steps

Most wavelet codecs use color space transform and wavelet transform to preprocess the images and arithmetic coding for the entropy coding part, while their data organize methods are varied. This chapter introduces color space transform, wavelet transform, and arithmetic coding as the background of the wavelet-based subband coding.

2.1 Image Redundancy

In image and video coding, there are three types of image redundancy(Rabbani)[7]:

- *Spectral Redundancy*, which is due to the correlation between different color planes or spectral bands.
- *Spatial Redundancy*, which is due to correlation or dependence between neighboring pixels.
- *Temporal Redundancy*, which is due to the typically small changes between successive image frames.

The image and video compression algorithms always prepare an image or video taking advantage of these three kinds of redundancy to rearrange the original image/video information into a much compact form. Spectral and spatial redundancy is mostly used in wavelet-based subband coding and block prediction DCT coding, respectively. Temporal redundancy is used in videos.

2.2 Color Space — RGB and YCbCr

A raw color image captured from camera usually using RGB color space, which uses a combination of red, green and blue values to describe the color. That means each color pixel is constituted from three color pixels (red, green, and blue). In this form, we can treat every color digital image as three equal sized grey-scale images. For an 8-bit color depth, each color pixel uses a value ranging from 0 to 255 to represent the different intensity of red, green, or blue. That's 16,777,216 different colors. This precision is not only enough for human eyes but also significantly beyond the capabilities of existing displays.

Besides RGB, the most common color space used in image compression is YCbCr. It represents color space in three terms: one luminance component/luma (Y) and two chrominance components/chroma (Cb and Cr).

The YCbCr image is converted from the corresponding RGB image (Fig. 2.2 [31]). The conversion has several standards. Below is the ITU-R BT.601 standard which also used in the famous JPEG [8] standard:

- From 8-bit RGB to 8-bit YCbCr:

$$Y = 0.299R + 0.587G + 0.114B \tag{2.1}$$

$$Cb = 128 - 0.168736R - 0.331264G + 0.5B \tag{2.2}$$

$$Cr = 128 + 0.5R - 0.418688G - 0.081312B \tag{2.3}$$

- From 8-bit YCbCr to 8-bit RGB:

$$R = Y + 1.402\,(Cr - 128) \tag{2.4}$$

$$G = Y - 0.34414\,(Cb - 128) - 0.71414(Cr - 128) \tag{2.5}$$

$$B = Y + 1.772\,(Cb - 128) \tag{2.6}$$

This conversion changes the color space from three colors to luma and chroma values. However, only this conversion will not eliminate any redundancy information since the image size is not changed and the precision is still 8 bits. The redundancy savings come from a process called chroma subsampling.

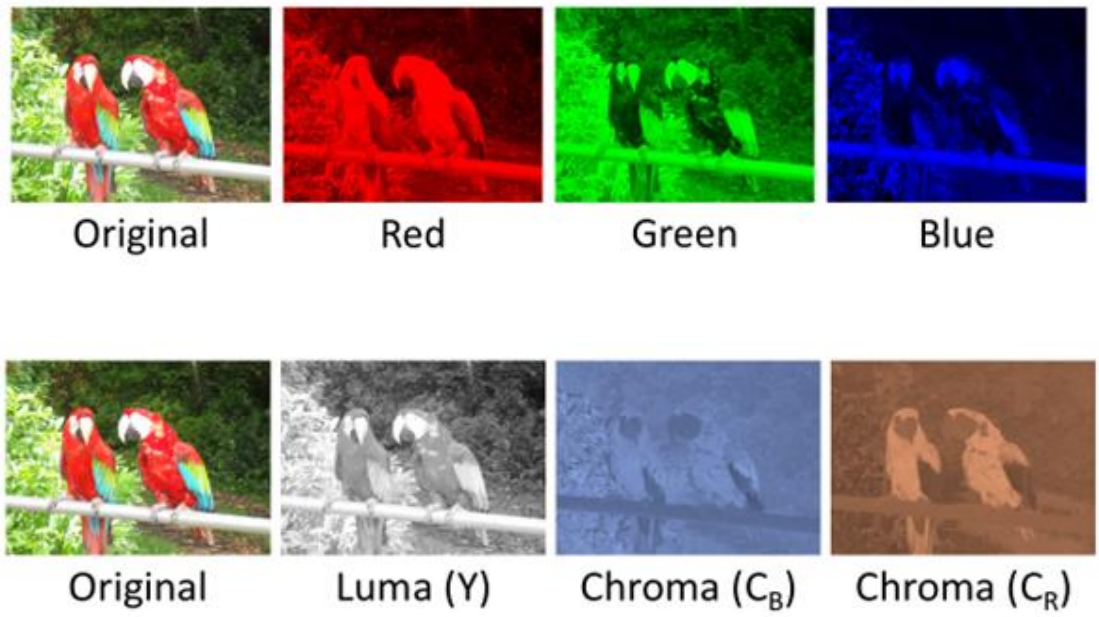

Figure 2.2. Examples of RGB and YCbCr color space

Chroma subsampling takes a YCbCr image and down-sampling its two chroma components, *Cb* and *Cr*. This process is based on a fact that human eyes are more sensitive to brightness than to chrominance, which means for human, most of the image detail lies in the luma component.

There are four popular chroma subsampling mode: 4:4:4, 4:2:2, 4:4:0, and 4:2:0. Table 2.1 compares these four modes [31].

Table 2.1. Four chroma subsampling mode details

Subsampling mode	Horizontal downscale	Vertical downscale	Bits per pixel for the image
4:4:4	1x	1x	24
4:2:2	2x	1x	16
4:4:0	1x	2x	16
4:2:0	2x	2x	12

Figure 2.3. YCbCr 4:4:4 vs 4:2:0. The left image is 4:4:4. Right image is 4:2:0

We can see that YCbCr 4:2:0 has only half size compared to the original image. Thanks to the brightness information are fully preserved, we will not

feel the clarity downgrade. The only difference is the color seems less saturated (Fig. 2.3 [32]).

2.3 Discrete Wavelet Transform (DWT)

DWT is widely used in image compression. It uses finite length yet mutually compensated low/high-pass filters to decompose a signal into low/high-frequency subbands in the temporal domain [6, 9, 10]. With frequency contents cut in half, each subband in temporal domain can be characterized using a resolution that matches its scale. Wavelet theory provides a fundamental insight into the structure of subband decomposition and thus a more efficient approach to filters design [1, 6].

The advantage of applying 2D wavelet transform is that the image after transformed usually exhibits lower entropy and is thus more compressible.

2D DWT could be employed as a model for several levels. After the first level of 2D DWT, the input image is decomposed into four subbands shown in Fig. 2.4.

From Fig. 2.4 we can see that there are four subbands: LL1, HL1, LH1, HH1. The LL1 subband can be decomposed by DWT again. Moreover, after the second level 2D DWT, the image is like Fig. 2.5.

From Fig. 2.5 we can see that there are four new subbands: LL2, HL2, LH2, HH2 instead of the LL1 in Fig. 2.4.

Fig. 2.6 shows a three-level 2D DWT output of the "Lenna" image.

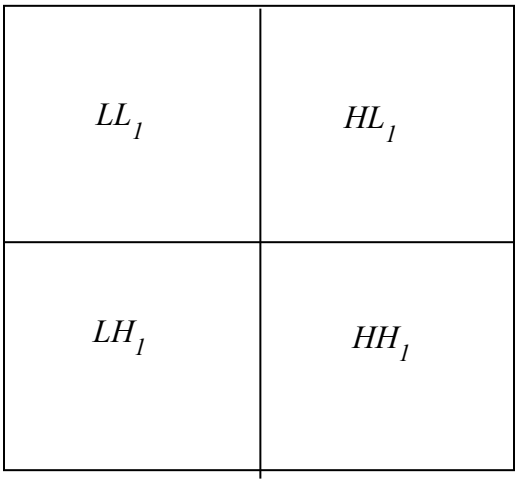

Figure 2.4. An image's structure after one level of 2D DWT

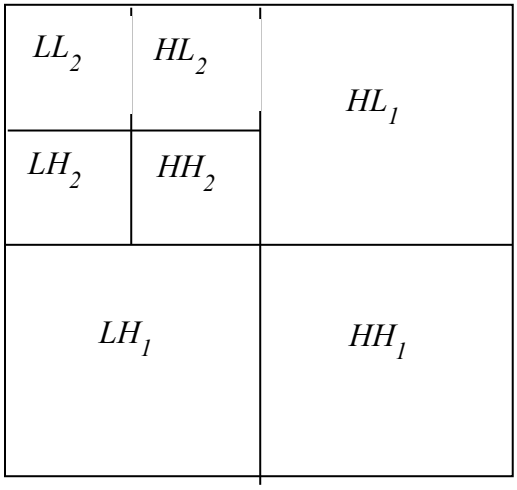

Figure 2.5. An image's structure after two scales of 2D DWT

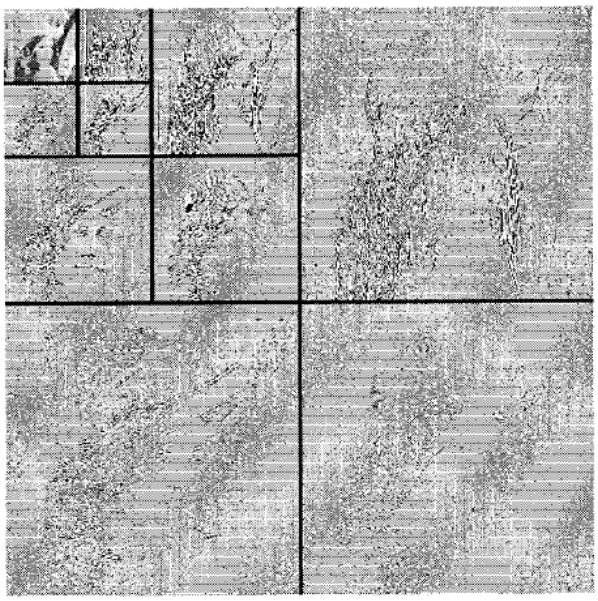

Figure 2.6. The subband pyramid of three-level 2D DWT

2.3.1 Implementation of 2D DWT in Image Coding

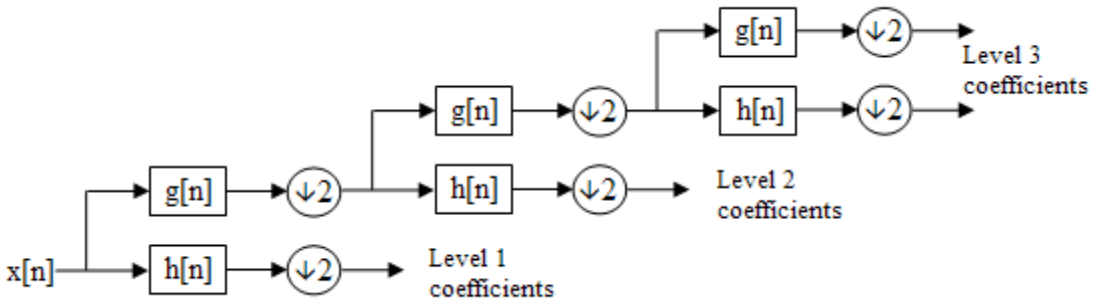

Figure 2.7. DWT inherit structure

An input sequence *x(n)* is passed through a low-pass filter *g(n)* and a high-pass filter *h(n)*. Then the output samples are down-sampling by factor 2. The low-pass output is the next level's input (shown in Fig. 2.7 [33]). These two filter steps can be defined as

$$y_{low}(n) = \sum_{k=-\infty}^{\infty} g(k)x(2n-k) \tag{2.7}$$

$$y_{high}(n) = \sum_{k=-\infty}^{\infty} h(k)x(2n-k) \tag{2.8}$$

Alternatively, written in convolutions:

$$y_{low} = (g * x) \downarrow 2 \tag{2.9}$$

$$y_{high} = (h * x) \downarrow 2 \tag{2.10}$$

Implementation:

a. Analysis Part

Input: image *x* (size *m* by *n*), 9/7 wavelet filters (low-pass analysis filter *g* with size 9, and high-pass analysis filter *h* with size 7) [20].

g =
{-3.782845550699535e-02,
-2.384946501937986e-02,
1.106244044184226e-01,
3.774028556126536e-01,
-8.526986790094022e-01,
3.774028556126537e-01,
1.106244044184226e-01,
-2.384946501937986e-02,
-3.782845550699535e-02}

h=
{-6.453888262893856e-02,
4.068941760955867e-02,
4.180922732222124e-01,
-7.884856164056651e-01,
4.180922732222124e-01,
4.068941760955867e-02,
-6.453888262893856e-02}

Output: 2D Wavelet decomposed image *y2* (size *m* by *n*)

Algorithm:

For i = 0 … m-1, do:

For j = 0 … n/2-1, do:

For k = 0 … 8, do:

*y1[i][j] = y1[i][j] + x[i][2*j+(k-4)]*g[k]*

(where x[i][a-j]=x[i][a+j] for j = 1, …, 4 when a = 0, n-1)

For k = 0 … 6, do:

*y1[i][j+n/2] = y1[i][j+n/2] + x[i][2*j+1+(k-3)]*h[k]*

For j = 0 … n-1, do:

For i = 0 … m/2-1, do

For k = 0 … 8, do

*y2[i][j] = y2[i][j] + y1[2*i+(k-4)][j]*g[k]*

For k = 0 … 6, do

*y2[i+m/2][j] = y2[i+m/2][j] + y1[2*i+1+(k-3)][j]*h[k]*

---

b. Synthesis Part:

Input: DWT image *y* (size *m* by *n*), low-pass synthesis filter *g* (size 7), high-pass synthesis filter *h* (size 9).

g =

{-6.453888262893856e-02,
-4.068941760955867e-02,
4.180922732222124e-01,
7.884856164056651e-01,
4.180922732222124e-01,
-4.068941760955867e-02,
-6.453888262893856e-02}

h =

{3.782845550699535e-02,
-2.384946501937986e-02,
-1.106244044184226e-01,
3.774028556126536e-01,
8.526986790094022e-01,

3.774028556126537e-01,
-1.106244044184226e-01,
-2.384946501937986e-02,
3.782845550699535e-02}

Output: reconstructed image $x2$ (size m by n)

Algorithm:

For j = 0 … n-1, do:

For i = ⌊−3/2⌋… (m-1+3)/2, do:

If 0 ≤ i ≤ m/2-1:

yl[i][j] = y[i][j]

Else if i < 0

yl[i][j] = y[-i][j]

Else: i >m/2-1

yl[i][j] = y[-i+2m-1][j]

For i = ⌊−3/2⌋… (m-1+3)/2, do:

For k = 0 … 6, do:

*x1[2*i+(k-3)][j] = x1[2*i+(k-3)][j] + yl[i][j]*g[k]*

For i = ⌊−5/2⌋… (m-1+3)/2, do:

If 0 ≤ i ≤ m/2-1:

yh[i][j] = y[i][j]

Else if i < 0

yh[i][j] = y[-i-1][j]

Else: i >m/2-1

yh[i][j] = y[-i+2m-2][j]

For i = ⌊−5/2⌋… (m-1+3)/2, do:

For k = 0 … 8, do:

*x1[2*i+1+(k-4)][j] = x1[2*i+1+(k-4)][j] + yh[i+m/2][j]*h[k]*

For i = 0 … m-1, do:

For j = ⌊−3/2⌋… (n-1+3)/2, do:

If 0 ≤ j≤ n/2-1:

yl[i][j] = x1[i][j]

Else if j < 0

yl[i][j] = x1[i][-j]

Else: j >n/2-1

yl[i][j] = x1[i][-j+2n-1]

For j = ⌊−3/2⌋… (n-1+3)/2, do:

For k = 0 … 6, do:

*x2[i][2*j+(k-3)] = x2[i][2*j+(k-3)] + yl[i][j]*g[k]*

For j = ⌊−5/2⌋… (n-1+3)/2, do:

If 0 ≤ i ≤ n/2-1:

yh[i][j] = x1[i][j]

Else if i < 0

yh[i][j] = x1[i][-j-1]

Else: i >n/2-1

yh[i][j] = x1[i][-j+2n-2]

For j = ⌊−5/2⌋… (n-1+3)/2, do:

For k = 0 … 8, do:

*x2[i][2*j+1+(k-4)] = x2[i][2*j+1+(k-4)] + yh[i][j+n/2]*h[k]*

2.3.2 The Characteristic of Wavelet-Transformed Images

A. Spatial-Frequency Localization

Subband coding decomposes an input image into several frequency bands. Each subband contains coefficients localized within a particular frequency range [17]. Moreover, each wavelet coefficient is calculated from a local piece of the input image. So they are also spatial localized.

B. Energy Compaction

Most natural images contain a relatively significant portion of smooth gradient content and textured content. Moreover, the images comprise a relatively small part of sharp edges and object boundaries. The flat areas have less variation than the textures and edges. They mostly consist of the low-frequency component. In contrast, the edges include primarily of the high-frequency component. The textured areas contain a mixture of the low and high-frequency component. Wavelet transform splits the input image into four equal-sized subbands by frequency. Moreover, the most energy in smooth and textured content is compacted into one of the four subbands, the LL subband [17]. By applying 2D DWT repeatedly to the LL subband at a coarser level, the energy will be compacted into few wavelet coefficients.

C. Within-Subband Clustering of Significant Coefficients

The coefficients in highpass subbands (HL, LH, HH) lack low-frequency energy in smooth and textured content while retaining high-frequency energy

found in edges and boundaries, which tend to be continuously and clustered [17].

D. Symmetry of HL and LH Subband

2-D Wavelet Transform is done by applying 1-D Wavelet Transform twice from two directions, horizontally and vertically, to the input image. HL subband is the first highpass filtered horizontally, and then lowpass filtered vertically. While LH subband is firstly lowpass filtered horizontally and then highpass filtered vertically. These two symmetric processes lead to asymmetric coefficients distribution in HL and LH subband.

E. Cross-Subband Similarity

As mentioned before, the highpass subbands retain the high-frequency energy of edges and textured content. Moreover, the coefficients are spatial localized. There is a dependency between the magnitudes of parent and children to a certain extent.

F. Decaying of Coefficients Magnitudes Across Subbands

The coefficient magnitudes in a coarser subband are smaller than the coefficient magnitudes in a finer subband [17].

2.4 The Adaptive Arithmetic Coding

An adaptive model represents the changing symbol frequencies (probabilities) seen so far in a source string [7, 11, 18]. Such a model matches the local statistics in the source string. In practice, it outperforms the fixed model version regarding compression efficiency.

The adaptive algorithm to encode the source string $s_{i_1}, s_{i_2}, s_{i_3}, \ldots, s_{i_m}$ of m

---

Step 1 Initialize the source model.

Given the source symbol set $S = \{s_1, \ldots, s_n\}$, initialize the frequency of each symbol to a constant. $f_i = f_0, i = 1, \ldots, n$.

Step 2 $k = 1, x = 0$.

Step 3 Compute the probability of each source symbol:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{2.11}$$

Step 4 Apply an affine transform to x:

$$y = w_{i_k}(x) \tag{2.12}$$

Step 5 Update source model: $f_{i_k} = f_{i_k} + 1$.

If $\sum_{j=1}^{n} f_j == MAXFREQUENCY$

$$f_j = \frac{f_j}{2}. j = 1, \ldots, n \tag{2.13}$$

Step 6 $k = k + 1$

If $k \leq m$

Then

$$x = y \tag{2.14}$$

Go to Step 3.

The adaptive arithmetic [11] coding works well for a stationary Markov source compare to an arithmetic coding with the fixed model. The adaptive arithmetic coding updates the corresponding conditional probability estimation each time when the coder visits a particular context.

The probability distributions may vary from one portion to another in an image that is non-stationary. The adaptive arithmetic coding is more robust and follows the local probability distribution variation very well. So it can achieve higher compression.

2.5 Binary Implementation of Arithmetic Coding

The conventional arithmetic coding (CAC) has two problems:

- In CAC, every time encoding an incoming symbol, we scale down the coding interval [L, L+R) (L: lower bound; R: range) to a new (smaller) subinterval, corresponding to the binary subsequence so far the encoder receives including the new symbol. To find a unique number (i.e., code) within to represent the subinterval (i.e., L and R), it needs to infinitely increase decimal digits or have an arbitrary precision floating point machine in principle, that turns out to be impractical.
- Encoding cannot start before the whole symbol sequence is received. Decoding will not be able to start before all the code bits are received.

Binary Implementation of Arithmetic Coding (BIAC) [12] was introduced to fix these two problems. Along with the way in encoding the symbol sequence, BIAC will generate binary code word bit by bit and send them to the decoder. So the floating-point machine is no longer needed.

2.5.1 The Binary Prefix

Prefix: the start digits of a number. For example, 1297 has prefix "1", "12", "129" and "1297".

Common prefix: if multiple numbers have the same prefix, we call the same prefix common prefix. For example, 120, 129 and 1297 have common prefix "1" and "12".

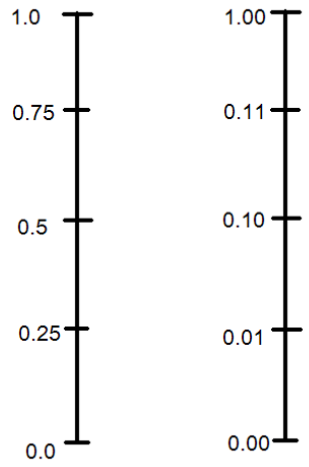

Figure 2.8: left is decimal present, and right is binary present

In BIAC, we only deal with the values less than 1. So "0." will always be the prefix. In the following part, "0." is omitted from the prefix. As Fig. 2.8:

- All the decimal values in interval [0, 0.5) has a common binary prefix 0
- All the decimal values in interval [0.5, 1) have a common binary prefix 1
- All the decimal values in interval [0, 0.25) have a common binary prefix 00
- All the decimal values in interval [0.25, 0.5) have a common binary prefix 01
- All the decimal values in interval [0.5, 0.75) have a common binary prefix 10
- All the decimal values in interval [0.75, 1) have a common binary prefix 11

If all the values in an interval have a common prefix *P,* then we say that *P* is this interval's prefix. If an interval has a prefix, its subinterval will also have that prefix.

In the following, $[x, y)_2$ refers to a binary representation of $[x,y)$.

2.5.2 Binary Implementation of Arithmetic Coding

BIAC is very similar to CAC. However, during the coding process, it directly uses binary numbers to represent *L* and *R*. The basic idea of BIAC is:

whenever the interval [*L, L+R*) has a prefix, output the prefix into a codeword.

Main_Process

Starts with initial interval $[0.0, 1.0)_2$

For (each incoming symbol) do:

1. If there are *n* symbols, like the CAC, the interval will be divided into *n* subintervals in proportion to the symbols' probabilities;
2. choose the one corresponding to the incoming symbol;
3. ***decide_the_output***.

---

decide_the_output

---

bits_to_follow = 0;

Step 1:

if (interval [*L, L+R*) lies in the lower half $[0.0, 0.1)_2$)

{ output "0";

output *bits_to_follow* number of "1"s;

bits_to_follow = 0;

$L = 2 \times L$, $R = 2 \times R$;

go to Step 1. }

else go to Step 2.

Step 2:

if (interval [*L, L+R*) lies in the upper half $[0.1, 1.0)_2$)

{ output "1";

output *bits_to_follow* number of "0"s;

bits_to_follow = 0;

$L = 2 \times (L - 0.5)$, $R = 2 \times R$;

go to Step 1. }

else go to Step 3.

Step 3:

if (interval [L, L+R) lies in $[0.01, 0.11)_2$)

{ *bits_to_follow* = *bits_to_follow* + 1;

$L = 2\times (L\text{-}0.25)$, $R = 2\times R$;

go to Step 1. }

else End.

***Note*:**

1. The interval [L, L+R) lies in the lower half $[0.0, 0.1)_2$. In this case, all the numbers in the interval will have a common prefix "0". The following encoding process will only choose the subinterval, so this prefix "0" will be in the final code word. So now we can output "0" into the code word even the encoding is not finished yet. However, there may exist more than one-bit prefix (i.e., $[0.00, 0.01)_2$ has prefix "00"). So we expand the lower half to be the full interval $[0.0, 1.0)_2$ and adjust $L = 2\times L$, $R = 2\times R$ (Fig. 2.9). This process will shift all the digits after the decimal point to the left by 1 bit (i.e., $[0.010, 0.011)_2$ becomes $[0.10, 0.11)_2$) and eliminate the coded prefix "0". At this point, we can continue to check if there exists "fresh" prefix.

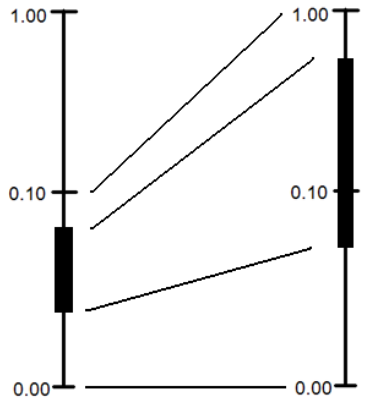

Figure 2.9 Interval [L, L+R) lies in the lower half

2. The interval [L, L+R) lies in the upper half $[0.1, 1.0)_2$. This case is very similar to the first one. We put "1" into the code word. Then expand this upper half to be the full interval $[0.0, 1.0)_2$ and adjust $L = 2\times(L-0.5)$, $R = 2\times R$ (Fig. 2.10).

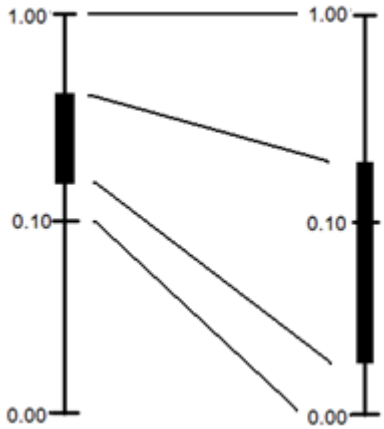

Figure 2.10 Interval [L, L+R) lies in the upper half

3. When the interval [L, L+R) straddles middle point and lies in $[0.01, 0.11)_2$ (i.e., in second and third quarters). There exists no common prefix as defined above. However, notice that in $[0.01, 0.11)_2$, the first bit is always opposite to the second bit in two bits prefix (either 01 or

10). By expanding the $[0.01, 0.11)_2$ to be the full interval $[0.0, 1.0)_2$ (accordingly adjusting $L = 2\times (L\text{-}0.25)$, $R = 2\times R$), the $[0.01, 0.10)_2$ and $[0.10, 0.11)_2$ become $[0.00, 0.10)_2$ or $[0.10, 1.00)_2$ respectively. Then, for example, if next incoming symbol's subinterval lies in lower half $[0.00, 0.10)_2$ (i.e., the next output is "0"), this subinterval should lie in $[0.01, 0.10)_2$ (prefix for this interval is "01") before expansion (Fig. 2.11). Therefore, the interval can safely be expanded as described above, if only we remember that whatever bit comes next, its opposite must be transmitted afterward as well.

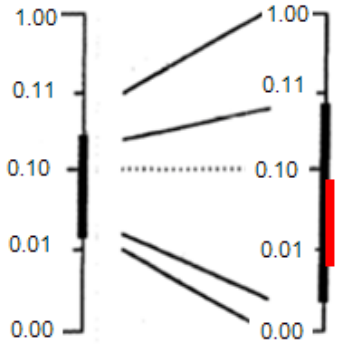

Figure 2.11. Red part is the subinterval for the next incoming symbol

Sometimes after this expand, the interval still straddles the middle point and lies in $[0.01, 0.11)_2$. Then we need to do another expand. Fig. 2.12 is an example where the current interval has been expanded three times. Suppose the next subinterval is in the lower half (i.e., output "0"). Then the next three output bits will be "1"s since the subinterval is not only in the $[0.01, 0.10)_2$ of the last interval, but in the $[0.011, 0.10)_2$, and moreover the $[0.0111, 0.10)_2$. We use a variable

bits_to_follow to remember the number of expansion. Moreover, the bit that is output next must be followed by that number of opposite bits. Now it is clear why in Step 1 and 2 there is "output *bits_to_follow* '0'/'1's".

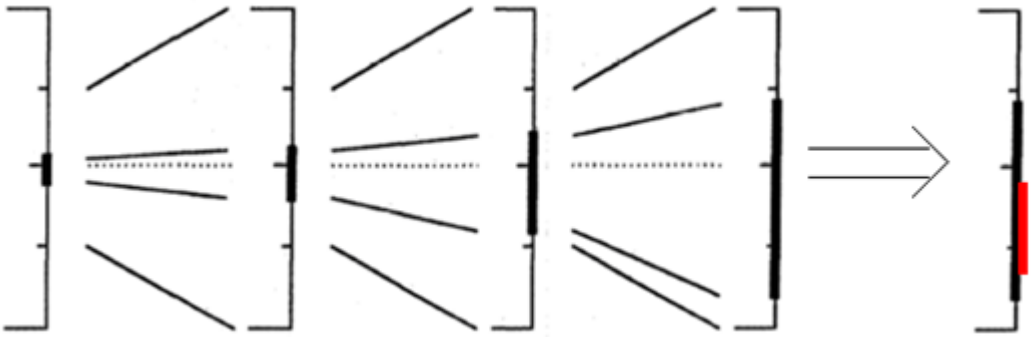

Figure 2.12 Three times of expansion (*bits_to_follow* = 3) and then the subinterval (red) lies in $[0.0, 0.1)_2$. The output is 0111

2.5.3 Termination

When all the symbols are encoded, we get a final interval. In BIAC, we choose a subinterval of the final interval and output that subinterval's prefix. To use the shortest prefix, we need the subinterval as big as possible. As described above, BIAC can guarantee that, after *decide_the_output*, either

$$L < 0.01 < 0.10 \leq (L + R) \tag{2.15}$$

or

$$L < 0.10 < 0.11 \leq (L + R) \tag{2.16}$$

Eq. (2.15) can make sure [0.01, 0.10) is the subinterval of [*L, L*+*R*). Eq. (2.16) can make sure [0.10, 0.11) is the subinterval of [*L, L*+*R*). So we can output "01" in the first case and "10" in the second.

A BIAC example is included in Appendix A.

2.6 Peak Signal-to-Noise Ratio

We evaluate the image compression performance mainly using peak signal-to-noise ratio (PSNR). Given an m x n input image I and an output image K, the Mean Squared Error (MSE) is defined as:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{2.17}$$

The PSNR is defined as:

$$PSNR = 10 \cdot \log_{10} \left(\frac{{MAX_I}^2}{MSE} \right) \tag{2.18}$$

Where $MAX_I = 255$. It is the maximum possible pixel value of the image.

2.7 Chapter Summary

This chapter introduced image redundancy and main steps of wavelet-based subband coding. Among them, the wavelet transform and arithmetic coding are the crucial steps. Wavelet transform puts an image into the spectral domain and exposes the spectral relation and redundancy in the image. Arithmetic coding compresses the information additionally after the data organization.

CHAPTER III: SIGNIFICANCE-LINKED CONNECTED COMPONENT ANALYSIS

Chapter II introduces the background of the wavelet-based subband coding. In this section, a unique data organization and representation strategy for DWT images will be presented. It is called the Significance-Linked Connected Component Analysis (SLCCA). SLCCA follows the primary process in Fig. 2.1 and has a remarkable data organize technic.

3.1 SLCCA Core Procedures

SLCCA processes an input image through the steps in Fig. 3.1. Since the color space transform method is usually considered as a separated technic to image compression, it is not shown in this process. The input image is an

image with pixel value ranging from 0 to 255. It can be one of the Red, Green, and Blue images. It can also be either luma or chroma image.

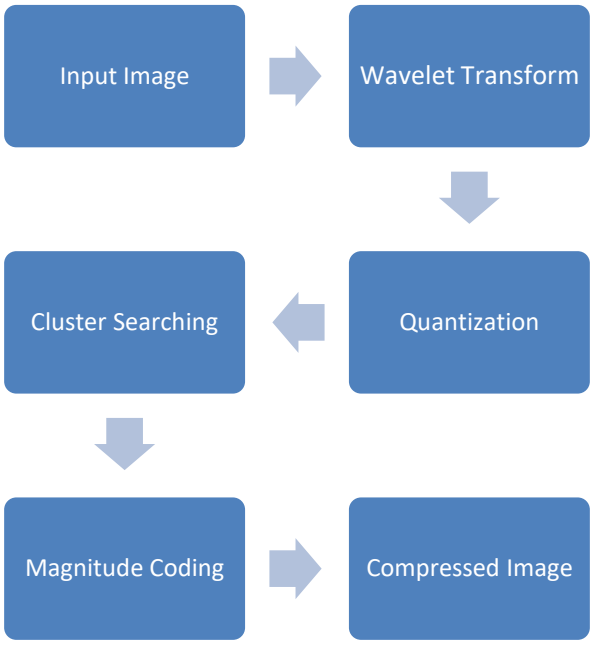

Figure 3.1. SLCCA Core Procedures

3.2 Wavelet Transform

The first step is a 2-D discrete wavelet transform (DWT). DWT uses finite length yet mutually compensated low/high-pass filters to decompose a signal into low/high-frequency subbands in the temporal domain [6, 9, 10]. The detailed process is introduced in Chapter II. 2-D DWT is employed in the input image for several inherited levels and decomposed the image into a multi-resolution wavelet pyramid.

3.3 Quantization

After the multi-level wavelet transform, the image has the same size as the input image. Each "pixel" in it is now the wavelet coefficient. All the wavelet coefficients are going through a uniform quantization [13].

Since SLCCA is a lossy compression, the wavelet coefficients are going through the quantization process by using a uniform scalar quantizer based on some bit rate or picture quality constraint. Quantization reduces the bit-depth of wavelet coefficients at the expenses of precision. Scalar quantization consists of a simple truncation of less significant bits, often obtained by right shifting wavelet coefficients' magnitude. Coefficients differing only in the digits being cut off will be indistinguishable after inverse-quantization.

The quantization step, *Stepsize*, is set based on the target compression ratio. During quantization, SLCCA assigns *Symbol* and *Magnitude* to each pixel based on corresponding coefficient value. The quantization procedure is as follows [13]:

Uniform quantization in SLCCA

For each pixel x
    If $|c(x)| < Stepsize$
        Magnitude(x) = 0
        Symbol(x) = Z
    Else

Magnitude(x) = $\lfloor(|c(x)| - Stepsize) / Stepsize \rfloor$

If $c(x) > 0$

    Symbol(x) = P

Else

    Symbol(x) = N

Where *c(x)* is pixel *x*'s wavelet coefficient value.

Pixels with symbol P/N are significant while those with symbol Z are insignificant. A significance map thus emerges from the pixel's symbols. Quantization produces a large number of insignificant pixels.

Given a quantization level, a quantized image produces a binary significance map by [14]:

$$A[x,y] = \begin{cases} 1, if\ the\ wavelet\ coefficient\ c\ at\ location\ [x,y] is\ significant, \\ 0, otherwise \end{cases} \quad (3.1)$$

The binary significance map is shown in Fig. 3.2. We could easily see from Figure 3.2 that the significant coefficients in each subband other than Low-Low (LL) subband usually indicate the occurrences of either edges or textures of high energy. It means that these significant coefficients represent the frequency "changes" in the image and there are connections between coarse and finer scales.

The wavelet image coding problem can thus be translated into the coding of significance map and sign/magnitude of significant coefficients.

Figure 3.2. The subband pyramid after binary significant map convert

3.4 Cluster Searching

After quantization, SLCCA scans each subband from highest level to the lowest level and uses irregularly shaped clusters to organize the adjacent significant pixels within a subband [4, 14]. A cluster starts from a seed coordinates, following by a sequence of symbols representing the pixels orderly in the cluster. The seed coordinates can either be coded directly or implied by cross-subband linkage from a cluster in parent subband [14].

3.4.1 Significance Clustering

Within-subband the SLCCA has a map coding method called significance clustering [14]. After quantization, a rather big portion of significant coefficients within each subband tend to form a few (3×3 or 2×2) connected components or clusters, shown in Fig. 3.3 [14]. It will be very inefficient if we encode too many insignificant coefficients. Hence, organizing and representing each subband as irregular-shape connected components or clusters is an efficient way for coding. This way can reduce the number of insignificant coefficients that barely contains the image's information. Clusters can be progressively constructed using conditional morphological dilation [16], resulting in an effective segmentation of the within-subband significance map.

Figure 3.3. Illustration of connected components or clusters

The conditional morphological dilation utilizes a structuring element (2 × 2 to 5 × 5, most common) to control the shape and size of the cluster as well as the boundary formation. Fig. 3.4 [14] is four examples of the typical structuring element.

However, the significant coefficients in the significant map are loosely clustered, especially in the high-frequency subband. If we use a small structuring element, i.e., 2 × 2 or 3 × 3, the cluster may contain a minimal amount of significant coefficients. Because some significant coefficients nearby will not be scanned. However, if we use a too large structuring element, the cluster will contain too many insignificant coefficients.

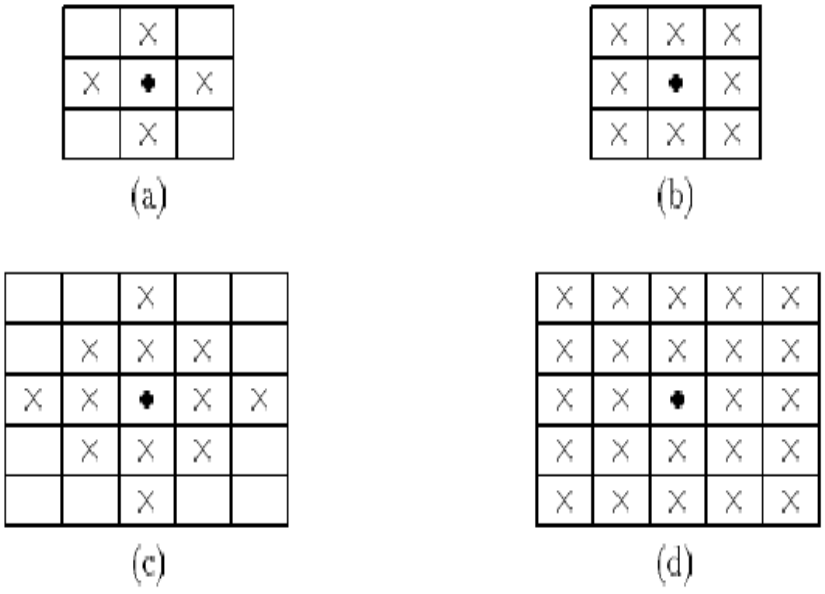

Figure 3.4. Four possible structuring elements. (a) 2 × 2 dilation (b) 3 × 3 dilation (c) 4 × 4 dilation (d) 5 × 5 dilation

The cluster detection operation [16] used by conventional SLCCA is illustrated in Fig. 3.5 [14]. In the figure, White pixels denote insignificant coefficients that are not to be coded; Black and gray pixels denote significant (*S*) and boundary insignificant (*I*) coefficients to be coded respectively.

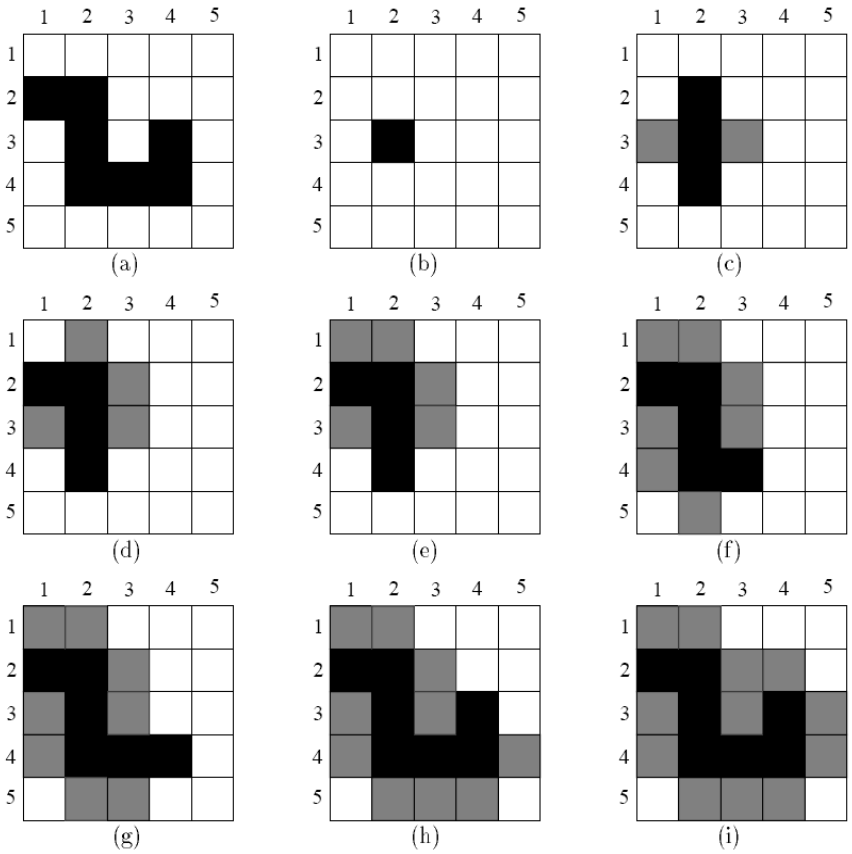

Figure 3.5. (a) A binary image produced after quantization. (b) Seed position (3,2). (c) Use 2x2 structuring element for conditional dilation. (d)-(i) Successive conditional dilation. (i) Cluster [11]

As illustrated in Table 3.1, beginning from the seed, a connected component or cluster is delineated by recursive conditional dilation. There are two ways

to perform conditional dilation, i.e., depth-first or breadth-first, which produce "*S, I*" strings of the same length with the same number of "*S*" and the same number of "*I*." And They both delineate the same geometric shape.

Table 3.1. Coded positions in each image in Fig. 3.5.

Figure	Current	Coded Positions
(c)	(3,2)	(2,2):S, (3,3):I, (4,2):S, (3,1):I
(d)	(2,2)	(1,2):I, (2,3):I, (2,1):S
(e)	(1,2)	(1,1):I
(f)	(4,2)	(4,3):S, (5,2):I, (4,1):I
(g)	(4,3)	(4,4):S, (5,3):I
(h)	(4,4)	(3,4):S, (4,5):I, (5,4):I
(i)	(3,4)	(2,4):I, (3,5):I

The final cluster is organized and represented by seed position and the following symbol string:

S I S I I I S I S I I S I S I I I I

In SLCCA, the depth-first approach is used. The cluster is organized and represented within a subband by the seed coordinate and a string of symbols *P*, *N*, and *Z*. Instead of the significant symbol "*S*" used above, the *P* and *N* represent the positive magnitude significant coefficient and negative magnitude significant coefficient, respectively. Moreover, instead of the insignificant symbol "*I*" used above, *Z* represents the insignificant boundary.

3.4.2 Significant Cluster Group

The conventional Significance Clustering method shows efficiency when the significant coefficients are “close enough” to each other. However, there is a situation which two clusters are neighbors. In fact, they may be adjacent so that they share some of the insignificant boundaries, i.e., Fig. 3.6.

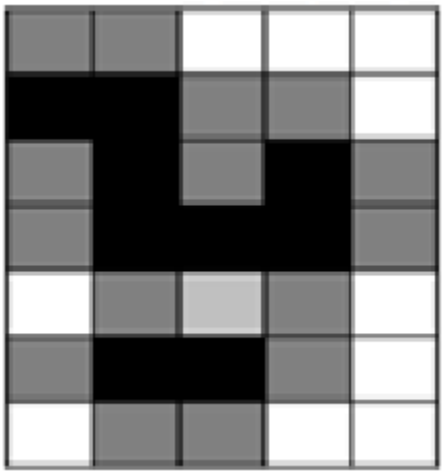

Figure 3.6. Two clusters close together

In this circumstance, if we could group these two clusters we can save one of the clusters’ seeds. The seed could be a significant linkage or a coordinate seed. So this cluster-group method could decrease the probability of the occurrence of the coordinate seed. As a result, the coding cost is reduced since the seed coordinates are bit-costly when coding.

To form a cluster group, we need do the conventional cluster scan first. After the cluster is found, scan all the boundary zeros’ neighbors using the same

structuring element. If there exists a significant coefficient in the neighbors, we call this boundary zero a shared zero and use it as a seed to find the new cluster, as shown in Fig. 3.7. We call these two clusters that connected through a shared zero a cluster group.

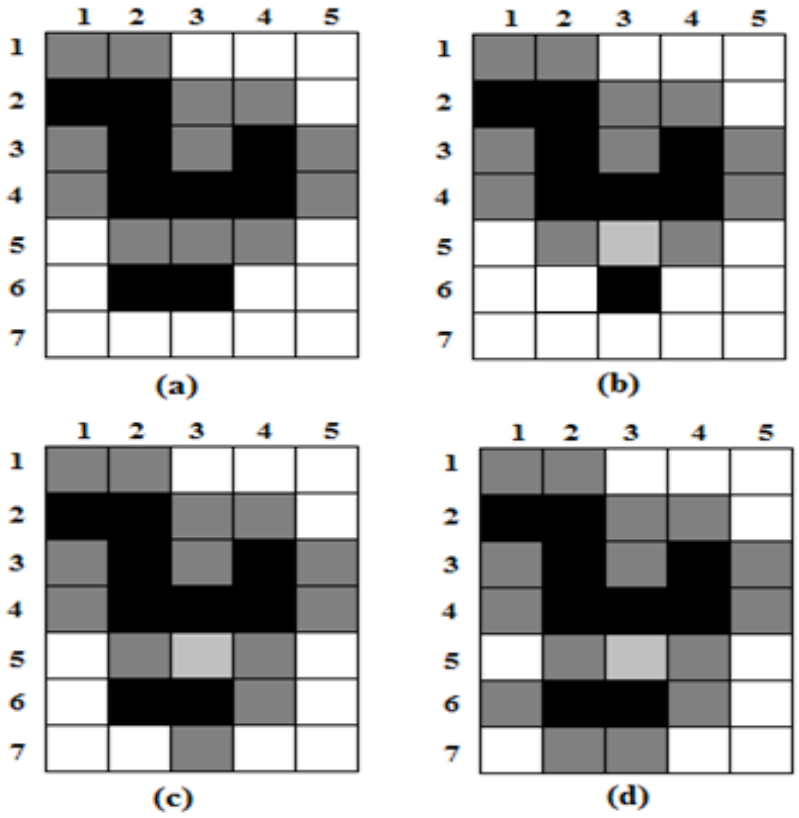

Figure 3.7. Two clusters are connected as a group through a shared-zero (5, 3)

Two clusters are shown in figure (a), the top one is the same one as shown in the figure. Although there are two shared zeros (5,3) and (5,2), only one of them (5,3) is used as a seed to produce the second cluster and eventually coded as SHARED-ZERO (*S-Z*).

The final cluster-group is organized as in Table 3.2 and represented by seed position (3,2) and the following symbol string:

[*S I S I I I S I S I I S (S-Z) S I I I I*] [*S I I S I I I*]

Table 3.2. Coded position of a new cluster that uses a shared-zero as the seed

Figure	Current	Coded positions and symbol
(b)	(5,3)	(6,3):S
(c)	(6,3)	(6,4):I, (7,3):I, (6,2):S,
(d)	(6,2)	(6,4):I, (7,3):I, (6,1):I,

In this cluster grouping method, a new symbol "SHARED-ZERO" is used. So there are four symbols (*P, N, Z*, and *SZ*) to indicate the significant and insignificant coefficients. One more symbol makes it less efficient when using arithmetic coding. However, the experiment shows that the gain from less coordinate seeds is more significant than the loss from one more symbol.

3.4.3 Significance Linkage

In zero-tree based image compression scheme such as EZW and SPIHT, the intention is to use the statistical properties of the trees to code the locations of the significant coefficients efficiently.[1]

In SLCCA, we also try to build a spatial connection between the coefficients crossing subband. However, unlike EZW and SPIHT, SLCCA exploiting the similarity among significant coefficients cross subband.

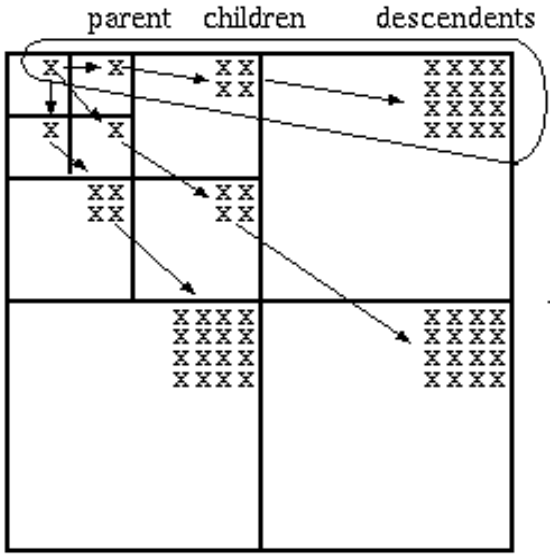

Figure 3.8. the relation between parent children and descendants in a 3-level subband pyramid

Relative to a given wavelet coefficient (ancestor), all coefficients at finer scales of similar orientation, which associate with the same spatial location, are called its descendants [14].

Notably, the coefficient at the higher level is called the parent, and all four coefficients (due to down-sampling by 2) associated with the same spatial location at the next lower level of similar orientation are called its children. The relation between them is shown in Fig. 3.8 [14].

The correlation between parent coefficient and child coefficient could lead to a very efficient method of map coding. We can see that magnitudes of wavelet coefficients statistically decay from a parent to its four children. Moreover, a significant coefficient at a lower subband likely has a significant parent at its corresponding higher subband [15]. Experiments showed that the correlation between the squared magnitude of a child coefficient and the squared magnitude of its parent tends to be between 0.2 and 0.6 with a high concentration around 0.35. [15] If there is a significant parent, which belongs to one subband, and at least one of its four children is significant and lies in another subband, these two clusters across subbands are called significance-linked. Fig. 3.9 [14] shows the significance-linkage between a parent and its four children.

The seed position (*x,y*) is derived from its parent position through significance linkage whenever possible. In this case, two more symbols *L* (LINK) and *NL* (NO-LINK) are used. *L* denotes the significant linkage, and *NL* indicates the significant linkage does not exist.

Since there are many significant coefficients in a connected component, the likelihood of finding significance linkage between two connected components is relatively high. If the significance linkage exists, we can use the linkage to mark the position of the children. Apparently, marking the significance

linkage costs much less than directly coding the seed coordinate position [14] since directly coded coordinates are very bit-consuming.

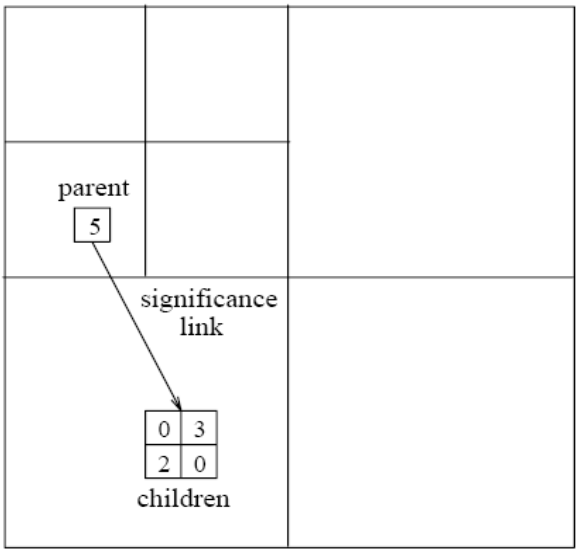

Figure 3.9. The values are the magnitudes of quantized coefficients. Nonzero values denote significant coefficients

3.4.4 Significance Map Scanning in SLCCA

From above descriptions, we see that to get a significance map of the image there are two things we need to know. One is the cluster groups within the subband, and the other one is the significance-linkage information cross subband.

Therefore, when we are scanning significance map, there are two symbol sequences. The first one contains two symbols, *L* (LINK) and *NL* (NO-LINK). The second one includes four symbols, *P*, *N*, *Z,* and *SZ*.

The order to scan the subband in SLCCA is shown in Fig. 3.10. Starting from the coarsest scale to the finest scale, SLCCA scans all the subband in each scale in the order of LH, HL, HH.

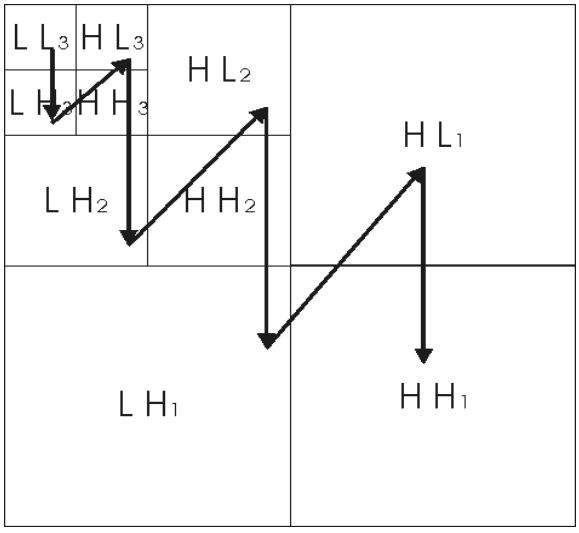

Figure 3.10. The scanning order of SLCCA

For each subband (LH, HL, and HH) in the coarsest scale, scan from left to right and top to bottom orderly to find a significant coefficient as the coordinate seed. Use a pre-specified conditional dilation to construct a cluster within the subband and leave the scanned symbol to the end of the symbol sequence (*LIST1*). After the cluster delineated, scan the symbols in the *LIST1* to find a possible boundary *Z*, which happens to have an un-coded

significant neighbor. Use this *Z* as a seed to find a new cluster. If the new cluster is not deleted by the elimination process, keep the symbol sequence to the end of the previous sequence (*LIST1*) and change this seed symbol *Z* to *SZ*. This cluster group scanning uses a breadth-first recursive process until a cluster-group delineated. Then rescan the symbol *P* and *N* in the sequence (*LIST1*) to check if there is a significant linkage exists in the child subband. If there is an un-visited yet significant child, code all its un-coded children, leave them to the end of child subband symbol sequence (*LIST2*) and use the first significant child as a seed to expend a new cluster and append the new cluster to the end of *LIST2*. In the meantime, put *L* symbol at the end of the link sequence (*LIST3*). If there is not an un-visited yet significant child, put *NL* symbol to the end of the link sequence (*LIST3*). After all, cluster-groups and linkages scanning finished, scan the un-visited pixel in the subband to find a seed, then do the cluster-group and linkage scan as the same as shown above.

For subband LH, HL, HH that not in the coarsest scale, first move the child subband symbol sequence (*LIST2*) of its parent subband to the *LIST1* of the current subband. Then rescan *LIST1* and do the cluster-group delineated and linkage search. After all cluster-groups and linkages found, scan the un-visited pixel in the subband to look for a seed, then do the cluster-group and linkage scan as the same as shown above.

3.5 Magnitude coding

The last step is magnitude coding. It represents the magnitudes of *P/N* pixels by their binary values. The binary bits compose a *0/1* symbol sequence following the pixel's order bit-plane wise. The side information, clusters, and magnitudes symbol sequences are finally coded using context-based adaptive arithmetic coding. The output bit stream is the final compressed file.

3.5.1 Bit Plane

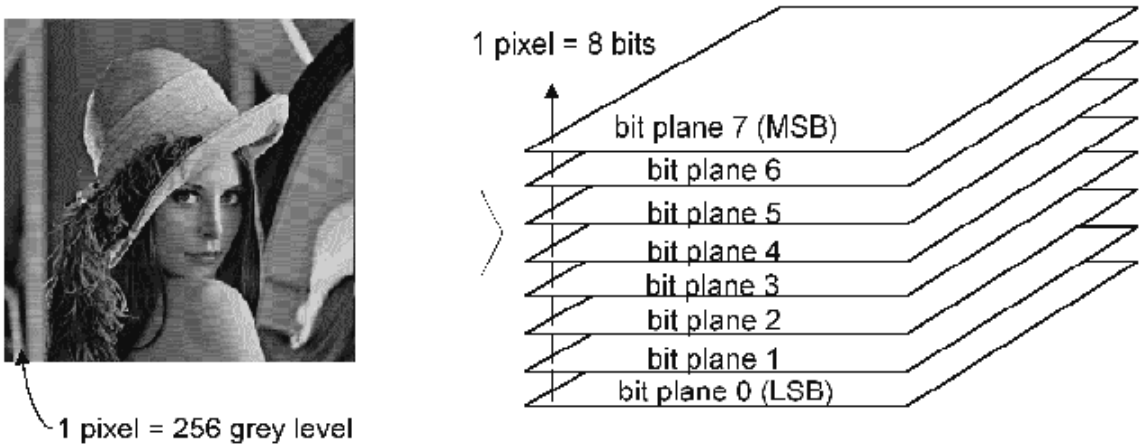

Figure 3.11. A pixel can be expanded into eight bit-planes

A bit plane of a discrete digital signal (such as image or sound) is a set of bits having the same position in the respective binary numbers (Fig. 3.11). The highest bit-plane is called the Most Significant Bit-plane (MSB). The lower bit-planes contribute less to the pixel.

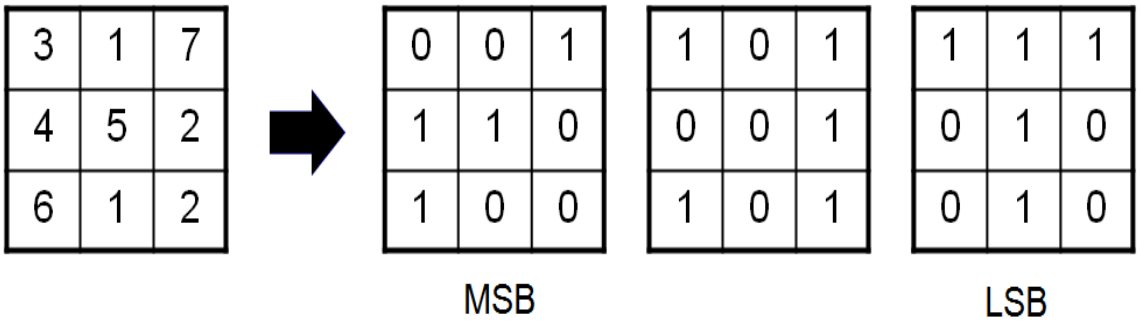

Figure 3.12. An example of bit-plane expand

Fig. 3.12 is an example of bit-plane expanding. Coefficients are in a sign-magnitude format with sign information being absorbed in significance map. Bit-plane entropy coding starts from the MSB. The number of bit-planes in a subband is determined by the maximum magnitude in the subband.

To produce more coding gain, SLCCA examines the context and defines the conditional probability of a significant coefficient in each bit-plane. In SLCCA, the context of a significant coefficient at a pixel in each bit-plane is defined by the significance status of its parent coefficient and eight neighboring pixels in the bit-plane. The details are in the next section.

3.6 Context Model of Arithmetic Coding

In arithmetic coding, the coding bit rate is related to the entropy of the symbol sequence. In an ideal situation, the bit rate is nearly equal to the entropy. Otherwise, the bit rate is larger than the entropy. So we need to

reduce the entropy of the arithmetic coding. The entropy in the arithmetic coding depends on the probability distribution of the symbols. The entropy will be lower if the probabilities of each symbol are diverse. To make the probabilities of each symbol as diverse as possible, we use a context model to classify each symbol in the input sequence to several classes. So that in each class a lower entropy could be obtained.

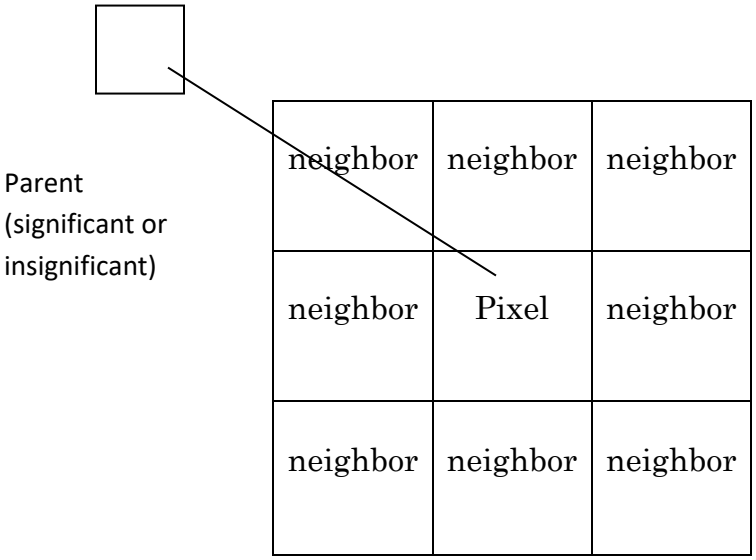

Figure 3.13. The neighbors and parent in SLCCA context model

In SLCCA there are total 18 different context models to divide the map symbols into 18 different classes.

We use two labels to classify the symbols, shown in Fig. 3.13. One is the total number of the significant coefficient within eight neighbors surround the symbol pixel. So there are nine possibilities. The other label is the

significance of the symbol pixel's parent pixel. This label has two possibilities. With these two labels, we can divide the symbols into 18 different classes.

3.6.1 Improved Context Model of SLCCA

In later improvement to SLCCA, the context model of the map symbol coding is upgraded. We discovered two more labels besides the two labels mentioned above. One is the shared-zero of the visited neighbors surrounding the symbol pixel. The other is the ratio of significant yet visited neighbors' number to total visited neighbors' number. Then we add the weight concept to the neighbors.

The label of shared-zero in the visited neighbors surrounding the symbol pixel has two possibilities. One is shared-zero exists, and the other is shared-zero does not exist. In practice, the criteria of shared-zero don't have noticeable improvement in the result. So we often pass over this label.

Table 3.3. Number of neighbors actually scanned in context calculation

# of scanned neighbors	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Occurrence times	418	930	2779	16536	13620	3340	878	291	99

The ratio label is proposed because in the experiment we found that for most symbol pixels, the calculation of the total number of the significant coefficient

within eight neighbors is only scanned 2 to 5 neighbors (Table 3.3). Therefore, if we define a ratio α:

$$\alpha = \frac{\#(significant\ coefficient\ in\ scanned\ neighbors)}{\#(scanned\ neighbors)} \tag{3.2}$$

In these cases, the ratio of significant yet visited neighbors' number to total visited neighbors' number shows more efficiency than only consider the significant coefficient within eight neighbors surrounds the symbol pixel. For example, a pixel with two significant neighbors out of three scanned neighbors has more possibility as a significant pixel than a pixel with two significant neighbors out of eight scanned neighbors. Thus, we employ this ratio as a label in the context model.

Moreover, we also noticed that for most pixels the number of significant coefficients in the scanned neighborhood is most likely 0, 1 or 2 (Table 3.4). For the number larger than 3, the probability is quite small. As a result, we can combine the number that bigger than two into one context model to reduce the context model number.

Table 3.4. Number of significant coefficients in scanned neighbor pixels

# of scanned significant neighbors	**0**	**1**	**2**	**3**	**>=4**
Occurrence times	15233	13998	6762	2682	216

Use this method, the total possibilities in the decide label and the overall context model number will be reduced.

0.7	**1**	**0.7**
1		**1**
0.7	**1**	**0.7**

Figure 3.14. neighbors' weight in SLCCA

In SLCCA's context model, we consider the eight neighbors' significance of a pixel has the same weight to affect the significance of the central pixel. However, in the experiment, we found that there is some difference between the neighbors' influence on the central pixel's significance. The neighbor's influence on the central pixel's significance mostly depends on the distance between the neighbor and the central pixel. So, we give each of the neighbors a weight factor. The neighbor with a small distance to the central pixel has a larger weight than the neighbor with a large distance to the central pixel (Fig. 3.14). After the experiment, we choose 0.7 as the weight since it can lead to the best results.

For magnitude coding, we only add the weight to the neighbors. The ratio and shared zero are not used to classify the context models.

We can see an example of the new context models:

Four symbols' (*P, N, Z, SZ*) sequence context model criteria (total 32 contexts):

1. Parent's significance ---- 2 (significance, insignificance)
2. Weighted significant yet visited neighbors number ---- 4 (0, 1, 2, ≥ 3)
3. Ratio (Weighted significant yet visited neighbors #/ total visited neighbors #) ---- 2(< 0.8, ≥ 0.8)
4. Shared zero in the visited neighbors ---- 2(exist, not exist)

Total context models=2×4×2×2=32

Magnitude context model criteria (total 14 contexts):

1. Parent's significance---- 2 (significance, insignificance)
2. Weighted significant yet visited neighbors number ---- 7(0, 1, 2, 3, 4, 5, ≥ 6)

Total context models=2×7=14

3.7 Chapter Summary

This chapter introduced the SLCCA core procedure. Include wavelet transform, quantization, cluster searching, magnitude coding, and the context models for arithmetic coding. The unique cluster searching and magnitude coding methods achieve remarkable efficiency in data

organization for SLCCA. Moreover, the customized context model setup leads to a lower entropy thus the arithmetic coding can achieve more compress ratio.

CHAPTER IV: SLCCA PLUS

In the last chapter, a wavelet image coder SLCCA is introduced. This chapter will introduce four improvements to enhance the already high-performance SLCCA. Including intelligent quantization with an adaptive threshold, enhanced cluster filter, potential significant shared-zero, and improved context models. The improved algorithm is called SLCCA Plus.

At the end of this chapter, we will compare the performance among JPEG2000, SLCCA, and SLCCA Plus.

4.1 Intelligent Quantization with Adaptive Threshold

Quantization is a crucial step, where the image quality loss occurs. As introduced in the last chapter, SLCCA uses a uniform quantization [6] scheme.

We would emphasize that the minimum magnitude at *P*/*N*-pixels is zero rather than one after quantization. Interval (-*Stepsize*, *Stepsize*) is called the dead zone. All the wavelet coefficients that fall within the dead zone are thought of insignificant and labeled by symbol *Z*. Quantization produces a large number of *Z*-pixels typically.

The corresponding inverse quantization procedure is as follows:

Inverse Quantization Procedure in SLCCA

For each pixel x

if Symbol(x) == Z

$c'(x) = 0$

Else

$c'(x) = (\text{Magnitude}(x)+0.5) \times Stepsize + Stepsize$

If Symbol(x) == N

$c'(x) = -c'(x)$

Where $c'(x)$ is pixel x's reconstructed wavelet coefficient value. All the wavelet coefficients that fall within the dead zone have reconstructed coefficient value equal to 0. The wavelet pyramid has a salient characteristic: the coefficient energy decays from higher wavelet levels to lower ones. In a 5-level wavelet decomposition of Lenna (512 × 512, Grayscale), the coefficient values may as Table 4.1.

Coefficient absolute values at a higher level are much larger than those at lower levels. Meanwhile, it is worth noting that the number of the coefficient

in the lower level is far greater than higher levels. As a result, appropriately choosing a *Stepsize* may produce plenty of insignificant pixels in lower wavelet levels thus reduce bit cost. Since SLCCA reconstructs insignificant pixels as coefficient with value 0, their maximum inverse quantization error equals to *Stepsize*. In contrast, the significant pixels' maximum error equals to 0.5 × *Stepsize*. The vast amount insignificant pixels lower the image quality. An adaptive threshold is introduced to solve this issue. The significance threshold now equals to $r \times Stepsize$, where $0 < r \leq 1$. We can see an example in Fig. 4.1.

Table 4.1. The coefficients range in a 5-level wavelet decomposition of Lenna

	LL Subband	Level 5	Level 4	Level 3	Level 2	Level 1
Max Value	6686	1199	684	367	238	63
Min Value	1486	-1337	-799	-342	-299	-74

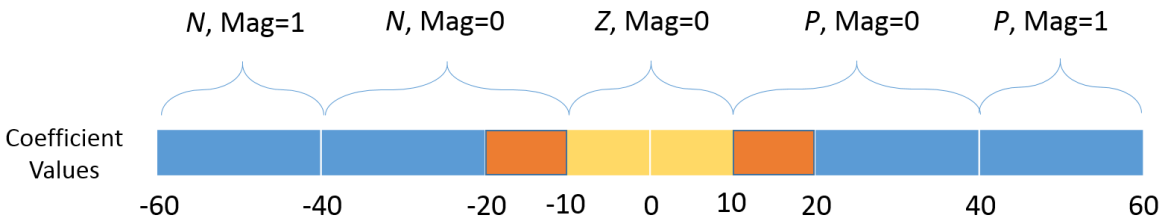

Figure 4.1. Intelligent Quantization Example. *Stepsize* = 20, r = 0.5. The pixels with coefficient values fall within the orange color interval used to be insignificant. Now they are marked as significant pixels.

By adequately shortening the dead zone, we can transform some Z-pixels, whose absolute coefficient values are below but near to *Stepsize*, into P/N-pixels. In doing so, we may increase the decoded image quality at the cost of enlargement of some clusters. A non-uniform quantization scheme and its inverse are proposed below for SLCCA Plus.

Non-uniform quantization in SLCCA Plus, with r = 0.95 by default

For each pixel x
  If $|c(x)| < r \times Stepsize$
    Magnitude(x) = 0
    Symbol(x) = Z
  Else
    Magnitude(x) = $\left\lfloor \left| |c(x)| - Stepsize \right| / Stepsize \right\rfloor$
    If $c(x) > 0$
      Symbol(x) = P
    Else
      Symbol(x) = N

Inverse non-uniform quantization in SLCCA Plus, with r = 0.95 by default

For each pixel x
  If Symbol(x) == Z
    $c'(x) = 0$
  Else
    If Magnitude(x) == 0
      $c'(x) = (1.5 - 0.5 \times (1 - r)) \times Stepsize$
    Else
      $c'(x) = (\text{Magnitude}(x) + 0.5) \times Stepsize + Stepsize$

If Symbol(x) == N

$c'(x) = -c'(x)$

We can see that this improvement is only for the zero magnitude pixels. The purpose of the adaptive threshold is to save the insignificant pixels which are close to the threshold. These insignificant pixels produce higher reconstruct error. Coding them as significant pixel achieves a better trade-off between bit cost and image quality. The threshold adjuster r can be calculated for each image or at each wavelet level for the best result. By default, it is set to 0.95 to reduce computational complexity.

4.2 Enhanced Cluster Filtering

A minuscule cluster usually does not contribute much quality to visual effects [14]. A small cluster with high *#(insignificant pixel)/#(significant pixel)* ratio contains a more bit proportion of side information. Therefore, in SLCCA they are eliminated to save coding cost. The elimination depends on cluster' significant area, i.e., the number of significant pixels in the cluster. Experiments demonstrate that the removal of one-significant-pixel clusters leads to better coding gain. Moreover, filter out the clusters with more than one significant pixel will reduce performance. We look into the filtered-out clusters and find a statistical pattern of their magnitudes in Table 4.2. We can see that only a few clusters contain significant pixel with magnitude

equals to 1 or 2. The P/N pixels in most one-significant-pixel clusters have zero magnitudes.

Table 4.2. Significant pixel's magnitude pattern in filtered clusters

Subband	# (filter-out cluster)	# (P/N magnitude =0)	# (P/N magnitude =1)	# (P/N magnitude =2)
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	2	2	0	0
5	2	2	0	0
6	5	4	1	0
7	9	8	1	0
8	16	15	1	0
9	15	14	1	0
10	50	49	1	0
11	54	51	3	0
12	49	46	2	1
13	149	149	0	0
14	17	17	0	0
15	94	94	0	0
Total	**462**	**451**	**10**	**1**

Based on this observation, we put forward an idea: the magnitude volume, besides significant area, represents the importance of a cluster. Thus, enhanced cluster filter adds magnitude volume into account and eliminates a cluster if it satisfies one of the following two conditions:

- It contains only one significant pixel.

- It contains two significant pixels, and both of their magnitudes are zero.

4.3 Potential Significant Shared-Zero

After quantization, a large portion of significant pixels tends to cluster. Organizing and representing significant pixels as irregular-shape connected components (clusters) is an efficient way to omit insignificant pixels' bit consuming. Clusters can be progressively constructed using morphological dilation [16]. Beginning with a known significant pixel (Seed), SLCCA uses breadth-first search (BFS) to explore the connected significant pixels. All the pixels (including significant and insignificant) covered by BFS process are recorded in a sequence orderly by their Symbols (P, N, or Z). With the coordinates of the Seed, this $P/N/Z$ sequence is sufficient for a reconstruction of the cluster.

An empirical fact is that the clusters sometimes tend to be fragmented, i.e., there are a large number of small-sized clusters adjacent to each other, as illustrated in Fig. 4.2. The bit cost of their seed coordinates is relatively high. We can see an example from Table 4.3 that in level 1 to 3, the seed size is even larger than the cluster's $P/N/Z$ sequence's size.

Figure 4.2. The BFS covers a significant pixel's eight neighbors and adds neighbor pixels into the cluster.

Table 4.3 SLCCA Subband information. The unit in the table is Byte.

Level	Subband	Link-cluster (P/N/Z) size	Link(L/NL) size	Seed-cluster (P/N/Z) size	Seed size	Magnitude size
5	LL	0	0	2	1	271
	HL	0	0	43	1	173
	LH	0	0	48	1	109
	HH	0	0	47	3	110
4	HL	183	1	0	0	361
	LH	163	3	0	0	188
	HH	151	0	0	0	182
3	HL	518	7	3	6	542
	LH	348	7	9	10	251
	HH	336	11	2	5	248
2	HL	900	34	8	30	520
	LH	531	26	8	8	200
	HH	469	33	7	15	135
1	HL	730	88	21	36	108
	LH	112	38	6	8	7
	HH	12	8	0	0	0

SLCCA Plus introduces a Shared-Zero exploring algorithm to solve this problem. Before preceding this algorithm, we need to introduce a potential-significant concept.

A potential-significant pixel is an insignificant pixel, but its absolute coefficient value is very close to the quantization threshold. We use the following procedure to determine an insignificant pixel's potential-significance.

Potential-Significance (pixel *x*)

If $c(x) \geq t \times Stepsize$

    Return P

If $c(x) \leq -t \times Stepsize$

    Return N

Return Z

Where $0< t \leq r$ (t is set to 0.85 by default), $c(x)$ is pixel x's wavelet coefficient value. If an insignificant pixel's potential-significance equals to P/N, it is called potential-significant P/N pixel.

We can see from Fig. 4.3 that with a suitable t, the values of potential-significant coefficients are very close to those of minimum significant coefficients.

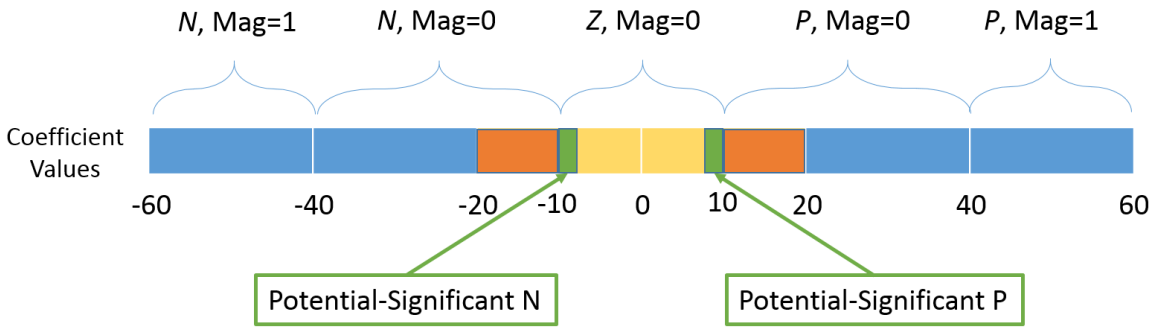

Figure. 4.3 An example of potential-significant. *Stepsize* = 20, r = 0.5, and t = 0.3. The pixels with coefficient values fall within the green color intervals are potential-significant.

The shared-zero exploring algorithm's central idea is finding the potential-significant pixels which are the boundaries of multiple clusters and changing them to significant pixels to merge the clusters. So that only one coordinate seed is needed. It works as follows. In a found cluster *C1*, SLCCA Plus examines each of the *Z* pixels. If a *Z* pixel is potential-significant and has a neighboring undiscovered cluster *C2*, it is called shared-zero pixel. We change it to *P/N* pixel corresponding to its potential-significance, and its magnitude remains 0. As a result, *C1* connects *C2*. We continue doing this shared-zero exploring procedure until no more unchecked *Z* pixels (including those in newly added clusters). At this stage, *C1* combines all the adjacent clusters. Then we redo the cluster dilation from *C1's* seed and get the cluster symbol sequence with the right order. Fig. 4.4 shows the potential-significant shared-zero exploring procedure.

Changing the potential-significant pixel to significant pixel produces some error inevitably. However, with the appropriately chosen t value, we can minify this error. Also, taking into account the bit saving from less coded coordinates, the shared-zero exploring algorithm achieves a better trade-off.

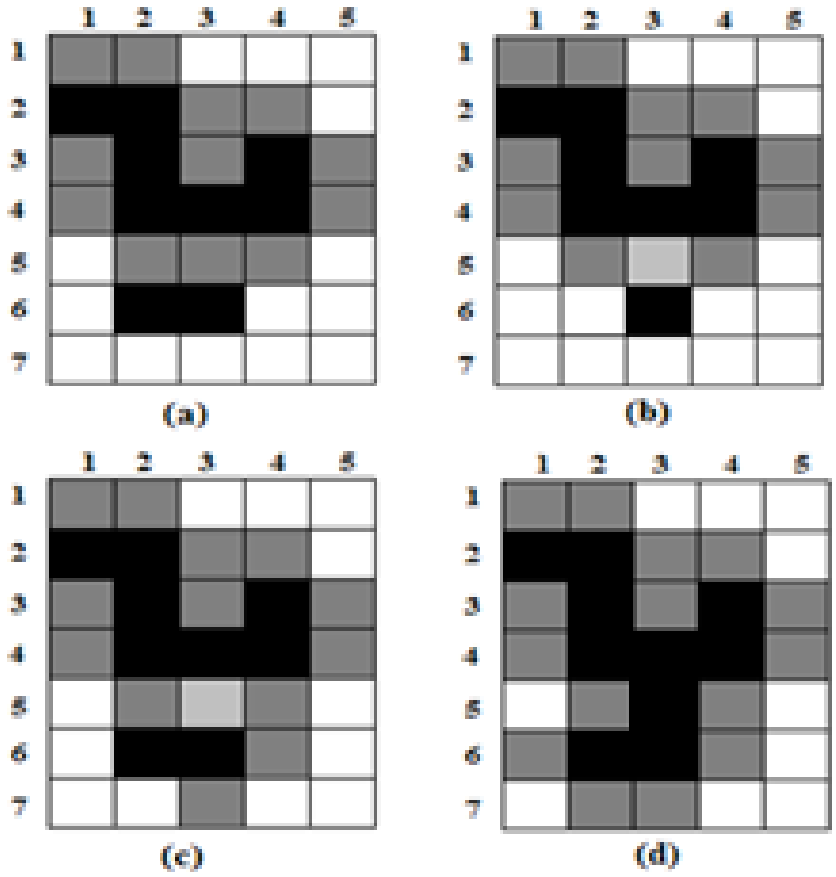

Figure. 4.4 Potential-significant exploring procedure. (a) A found cluster with two neighboring significant pixels. (b) A potential-significant Z is found. (c) The potential-significant leads to the new cluster. (d) Mark the potential-significant as significant.

4.4 Improved Context Models for Arithmetic Coding

SLCCA uses adaptive arithmetic coding [7, 11, 18] to encode *P/N/Z* sequence of clusters, magnitudes sitting on *P/N*-pixels, and other side information. Arithmetic coding has been effective in approximating the entropy of the source stream [11]. An adequate context model can provide a good prediction of the next source symbol to reduce the uncertainty or entropy [19]. In a wavelet transformed image, the significance status of a coefficient may have some correlation with the significance status of its (8, for instance) neighboring coefficients and the significance status of its parent coefficient as well [14].

The same context model is used in SLCCA to predict a *P/N/Z*-pixel or *0/1*-bit in magnitude bit-planes. The model is based on a combination of the significance odds of its eight neighboring pixels and the significance status of its parent. It has some limitations. First, when applied to *P/N/Z* sequence coding, the context model does not distinguish between *P*- and *N*-pixel. Second, the significance odds are not suitable for predicting 0/1-bit in magnitude bit-planes. Improvements are made in SLCCA Plus as follows.

4.4.1 Direction Sensitive Context Model for Cluster

In significance map, the presence of *P*- or *N*-pixels reveals directional continuity and directivity, as illustrated in Fig. 4.5. The direction varies with

different subbands. In HL (LH) subbands, the direction is horizontal (vertical).

Z	Z	Z	P	Z	Z	Z	Z	Z	Z
Z	Z	Z	P	Z	Z	Z	Z	Z	Z
Z	Z	N	P	Z	Z	Z	Z	P	Z
Z	N	N	Z	Z	Z	Z	Z	P	Z
Z	N	N	Z	Z	N	Z	Z	N	Z
P	N	Z	Z	P	N	N	N	N	Z
P	N	Z	N	P	N	N	P	N	Z
P	P	N	N	P	N	N	P	N	P
Z	P	P	N	N	P	N	P	N	P
Z	Z	P	N	P	P	N	P	P	P
Z	Z	P	Z	P	P	N	Z	P	P
Z	Z	Z	Z	Z	P	N	N	Z	Z

Figure 4.5. An example of P/N pixel's continuity and directivity in LH subband.

The new context model for a $P/N/Z$-pixel is direction sensitive. It utilizes three descriptive numbers $S_p(x)$, $S_n(x)$, $T(x)$, which are defined as the significance status for the parent pixel, the significance status for neighbors as a whole or neighbors' vote for pixel x to be P or N, respectively:

- The significance status of parent pixel

$$S_p(x) = \begin{cases} 0, & if\ parent\ pixel\ is\ insignificant \\ 1, & otherwise \end{cases} \tag{4.1}$$

- The neighbor' significance status

$$S_n(x) = \begin{cases} 0, & if\ \#(significant\ neighbors) < 3 \\ 1, & otherwise \end{cases} \tag{4.2}$$

- The neighbor' tendency toward P or N.

$$T(x) = \begin{cases} 0, if\ \#(P\ neighbors) = \#(N\ neighbors) \\ 1, if\ \#(P\ neighbors) < \#(N\ neighbors) \\ 2, if\ \#(P\ neighbors) > \#(N\ neighbors) \end{cases} \quad (4.3)$$

In calculating these numbers, we count only pixels, which are previously coded. In Eq. (4.3), we only check two horizontal (vertical) neighbors for HL (LH) subbands. For HH subbands, we check two horizontal and two vertical neighbors. Since coefficients in LL subband are non-negative, we simply code each magnitude bit-plane-wisely. That is, there is no $P/N/Z$ sequence sought in LL subband. At last, a total of 12 context models are defined for $P/N/Z$ sequence coding as follows:

$$K(x) = S_p(x) + 2S_n(x) + 4T(x) \quad (4.4)$$

4.4.2 Improved Context Model for Magnitude

Significant pixels' magnitudes are coded by their binary values bit-plane wise. In each subband, the maximum magnitude offers the fix-length of the binary values. Pixels with smaller magnitudes are more than those with larger magnitudes. As a result, for most of the pixels, there are a lot "0" bits in the higher bit-planes, e.g., Fig. 4.6.

In JPEG2000, which also uses bit-plane coding, the context model depends on the significance status of the pixel's eight neighbors and whether the pixel is a yet coded significant.

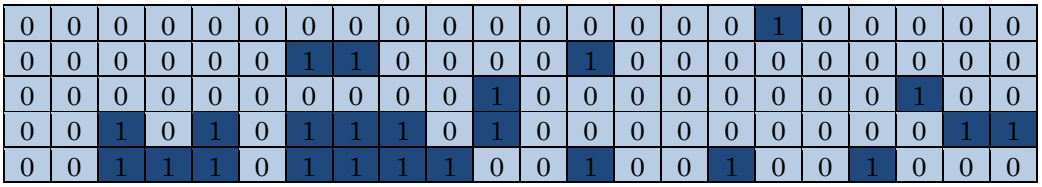

Figure 4.6. An example of bit-planes. Dark blue stands for "1" bits and light blue stands for "0" bits.

In SLCCA Plus, the context selection for a binary bit of a pixel is related to the following conditions. First, the "1" distribution in the bit's eight bit-plane neighbors. Second, the significance status of the pixel's eight neighbors. Third, the bit's position compares to the most significant bit (MSB) in the bit-planes. Fourth, the pixel's subband.

Explicitly, for a bit b of pixel x, first we calculate three quantities:

$$N_h(b) = count(\text{"1" in horizontal bit-plane neighbors of b}) \tag{4.5}$$

$$N_v(b) = count(\text{"1" in vertical bit-plane neighbors of b}) \tag{4.6}$$

$$N_d(b) = count(\text{"1" in diagonal bit-plane neighbors of b}) \tag{4.7}$$

The quantity $N^*(b)$, representing the influence of the bit's neighboring "1"s, integrates these three quantities:

$$N^*(b) = \lceil w_h N_h(b) + w_v N_v(b) + w_d N_d(b) \rceil \quad (4.8)$$

Where w_h, w_v and w_d are weights for different directions. Their values are set depend on the subband. For LH subbands, $w_h = 1.6$, $w_v = 0.28$, $w_d = 0.06$. For HL subbands, $w_h = 0.28$, $w_v = 1.6$, $w_d = 0.06$. For the other subbands, $w_h = 1.6$, $w_v = 1.26$, $w_d = 0.54$.

Then we check the significance status of the pixel's eight neighbors.

$$N(b) = count(significantneighborofx) \quad (4.9)$$

At last the context model *K(b)* is calculated by:

$$K(b) = \lfloor \alpha N(b) + \beta N^*(b) \rfloor \quad (4.10)$$

Where $\alpha = 0.7$, $\beta = 0.3$ if b is beneath the MSB of x. Otherwise, $\alpha = 035$, $\beta = 0.65$.

These improved context models provide more detailed classification to the magnitude bits. The context considers not only the pixel neighbors' significance but also the correlation among neighboring bits in the bit-planes.

4.5 Performance Evaluation

In this section, we evaluate the coding efficiency of SLCCA Plus. We compare SLCCA Plus with SLCCA and JPEG2000. We use 5-level wavelet

decomposition with 9/7 filter [20] (introduced in Chapter II) for both SLCCA Plus and SLCCA. In SLCCA Plus, r and t are set to default values. For JPEG 2000 coding, the Kakadu software tools provided by [27] is used. Fig. 4.7 shows the image set. All images are 512×512 grayscale natural images.

Figure 4.7 Experiment image set. Top left: Lenna. Top right: Barbara. Bottom left: Peppers. Bottom right: Mandrill

We evaluate their performance by PSNR, which is introduced in Chapter II. Table 4.4 presents the PSNR based bit-distortion performance. On average, SLCCA Plus achieves 7% bit saving over JPEG 2000 and 4% over SLCCA.

Table 4.4. PSNR based bit-distortion performance

Image	Bitrate (bit/pixel)	PSNR (dB)		
		JPEG2000	SLCCA	SLCCA Plus
Lenna	0.1	29.94	30.29	30.50
	0.3	34.90	35.02	35.25
	0.5	37.28	37.33	37.51
	0.7	38.71	38.73	38.94
	0.9	39.75	39.77	40.00
Barbara	0.1	25.00	24.84	25.09
	0.3	29.80	29.67	29.82
	0.5	32.87	32.48	32.92
	0.7	35.24	35.03	35.32
	0.9	37.35	36.97	37.35
Peppers	0.1	29.73	29.87	30.02
	0.3	34.15	34.16	34.36
	0.5	35.92	35.84	35.98
	0.7	36.97	36.86	37.07
	0.9	37.94	37.82	37.99
Mandrill	0.1	21.31	21.28	21.44
	0.3	23.66	23.75	23.94
	0.5	25.58	25.64	25.81
	0.7	27.08	27.19	27.37
	0.9	28.34	28.57	29.44

Fig. 4.8 is the detailed comparison for a zoomed in area in Lenna image. This area contains edges, textures, and flat part. We compare JPEG2000, SLCCA,

and SLCCA Plus at 0.3 bpp to the original image. JPEG2000 shows the worst result in the delicate texture of the hat. The texture is almost gone. While SLCCA shows some texture on the hat, but the texture is more like grids, not strips on the top of the hat. SLCCA Plus shows the best result among these three. It preserves more fine texture than SLCCA. The top of the hat shows some strip texture and the hat's band also has more clear wrinkles on it. In the flat, coarse texture area, such as the feather on the hat, the three codecs do not show much difference in this image at this bitrate.

Fig. 4.9 shows the detailed comparison of stem area in Peppers image. This area contains edges and flat part. We compare JPEG2000, SLCCA, and SLCCA Plus at 0.3 bpp to the original image. In JPEG2000, the stem's edge is blurry. Moreover, the texture of the stem does not preserve very well. SLCCA and SLCCA Plus have more explicit stem edge, while SLCCA Plus shows more evident texture on the stem.

Fig. 4.10 show the comparison of Barbara image under different bitrate by SLCCA Plus. The four images are at bitrate 0.1, 0.3, 0.5, and 0.7. Their corresponding PSNR are 25.09, 29.82, 32.92, and 35.32, respectively. SLCCA Plus can lead to an acceptable result at a very low PSNR (29.82 PSNR, 0.3 bpp), even with so many fine textures in the image.

Figure 4.8 Detail comparison for a zoomed in area in Lenna image at 0.3 bpp. Top left: original image. Top right: JPEG2000. Bottom left: SLCCA. Bottom right: SLCCA Plus

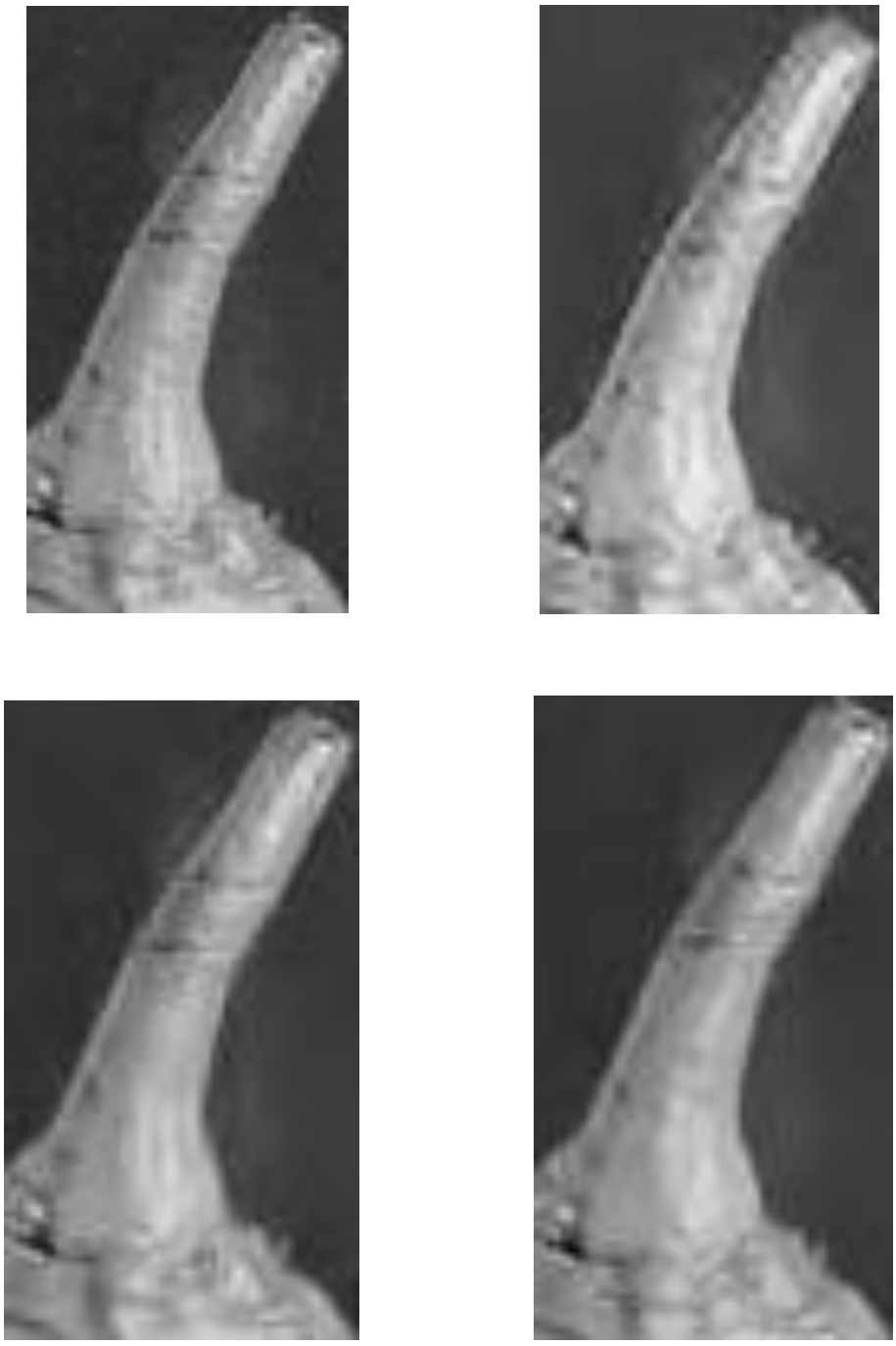

Figure 4.9 Detail comparison for a zoomed in area in Peppers image at 0.3 bpp. Top left: original image. Top right: JPEG2000. Bottom left: SLCCA. Bottom right: SLCCA Plus

Figure 4.10 Comparison of Barbara image under different bitrate by SLCCA Plus. Top left: 0.1 bpp, 25.09 PSNR. Top right: 0.3 bpp, 29.82 PSNR. Bottom left: 0.5 bpp, 32.92 PSNR. Bottom right: 0.7 bpp, 35.32 PSNR.

4.6 Chapter Summary

In this chapter, we introduced SLCCA Plus. The improvement includes intelligent quantization with an adaptive threshold, enhanced cluster filter, potential significant shared-zero, and improved context models. These four improvements lead to an impressive result that SLCCA Plus shows better performance than the popular wavelet-based codec JPEG2000 and SLCCA, especially in the textured and edge area.

CHAPTER V: CONCLUSIONS

An image coding algorithm, SLCCA Plus, is introduced in this dissertation. SLCCA Plus is a wavelet-based subband coding method. In wavelet-based subband coding, the input images will go through a wavelet transform and be decomposed into wavelet subband pyramids. Then the characteristics of the wavelet coefficients within and among subbands will be utilized to removing the redundancy. The rest information will be organized and go through entropy encoding. SLCCA Plus contains a series improvement method to the SLCCA. Before SLCCA, there are three top-ranked wavelet image coders. Namely, Embedded Zerotree Wavelet coder (EZW), Morphological Representation of Wavelet Date (MEWD), and Set Partitioning in Hierarchical Trees (SPIHT). They exploit either inter-subband relation among zero wavelet coefficients or within-subband clustering. SLCCA, on the other hand, outperforms these three coders by exploring both the inter-

subband coefficients relations and within-subband clustering of significant wavelet coefficients. The critical processes of SLCCA are multiresolution discrete wavelet decomposition, cluster searching within a subband, significance-link registration across subband, and bit-plane encoding of significant magnitudes by adaptive arithmetic coding.

SLCCA Plus strengthens SLCCA in the following aspects: Intelligence quantization, enhanced cluster filter, potential-significant shared-zero, and improved context models. The purpose of the first three improvements is to remove redundancy information further while keeping the image error as low as possible. As a result, they achieve a better trade-off between bit cost and image quality. Moreover, the improved context lowers the entropy by refining the classification of symbols in cluster sequence and magnitude bit-planes. Lower entropy means the adaptive arithmetic coding can achieve a better coding gain.

For performance evaluation, SLCCA Plus is compared to SLCCA and JPEG2000. On average, SLCCA Plus achieves 7% bit saving over JPEG 2000 and 4% over SLCCA. The results comparison shows that SLCCA Plus shows more texture and edge details in a lower bitrate.

More recently, in block prediction and discrete cosine transform (DCT) coding category, HEVC Intra coding has delivered an impressive coding gain

through using a delicate yet innovative block prediction scheme before DCT. HEVC outperforms SLCCA and SLCCA Plus. However, it is hard to integrate a block prediction scheme with wavelet transform. A block prediction error image could be statistically entirely different from an ordinary image.

Future research direction includes the combination of convolutional neural networks used in Super-Resolutions [28, 29] and wavelet subband coding to fully utilize the deep convolutional neural networks' significant advantages in extracting spectral and spatial features.

APPENDIX

Example of Binary Implementation of Arithmetic Coding

Symbol sequence to be code: 11010

At this time, we use non-adaptive model. The probability of 1 is 0.6, and the probability of 0 is 0.4.

We use five binary bits to represent the initial range [00000, 11111]:

Figure 6.1: Example of Binary Implementation of Arithmetic Coding

The lower half is [00000, 01111] and the upper half is [10000, 11111]. Furthermore, the first quarter is [00000, 00111]; the second quarter is

[01000, 01111]; the third quarter is [10000, 10111]; the fourth quarter is [11000, 11111].

Encoding

Code "1": choose the corresponding subinterval (in red):

Figure 6.2: Example of Binary Implementation of Arithmetic Coding

Now there is no expansion of the interval or output. Code the second "1": choose the corresponding subinterval:

Figure 6.3: Example of Binary Implementation of Arithmetic Coding

Now the interval is in the upper half. We expand the upper half, output "1" and get:

Figure 6.4: Example of Binary Implementation of Arithmetic Coding

Code the "0" and get the corresponding interval:

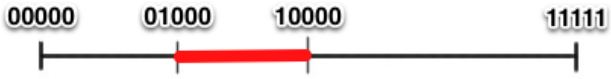

Figure 6.5: Example of Binary Implementation of Arithmetic Coding

Now the subinterval falls into the middle half and straddles the middle point. We expand the middle half and get:

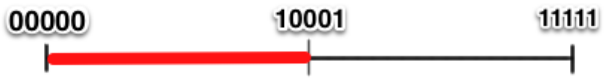

Figure 6.6: Example of Binary Implementation of Arithmetic Coding

Then code the next symbol "1" and get:

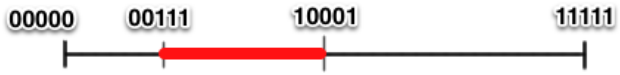

Figure 6.7: Example of Binary Implementation of Arithmetic Coding

There is still no need to expand and output. Next symbol is "0". We get:

Figure 6.8: Example of Binary Implementation of Arithmetic Coding

The subinterval is in the lower half, so we should expand the lower half and output "0". Then output a "1" immediately because we expanded the middle half once. Then we get:

Figure 6.9: Example of Binary Implementation of Arithmetic Coding

The subinterval is in the middle half and straddles the middle point. So we expand the middle half:

Figure 6.10: Example of Binary Implementation of Arithmetic Coding

Now the interval satisfies termination condition Eq. (2.16). So we output "10" and immediately a "0" for last middle half expansion.

At last, the output is "101100".

Decoding

Input: 101100.

First, set the initial interval:

Figure 6.11: Example of Binary Implementation of Arithmetic Coding

We use 5 bits precision to represent the interval. Correspondingly, we choose a 5-bit slide window to read the input. After each expansion, the window will slide to the right by one bit:

Figure 6.12: Example of Binary Implementation of Arithmetic Coding

To make sure the window can keep sliding to the right, we will add 0s or 1s to the end of input sequence:

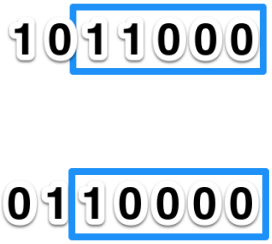

Figure 6.13: Example of Binary Implementation of Arithmetic Coding

The added bits will not affect the decoding precision.

At first, the input is “10110”. “10110” falls into symbol 1’s corresponding subinterval.

Figure 6.14: Example of Binary Implementation of Arithmetic Coding

So output symbol “1”. Then divide symbol 1’s interval by the probability of 0 and 1. We find that “10110” is still in symbol 1’s subinterval:

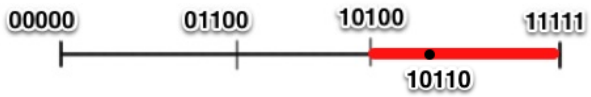

Figure 6.15: Example of Binary Implementation of Arithmetic Coding

The second symbol is “1”. Now the subinterval falls into the upper half of the whole interval. So we expand the upper half. Then slide the input window and get:

Figure 6.16: Example of Binary Implementation of Arithmetic Coding

The input “10110” becomes “01100” after expansion and falls into symbol 0’s subinterval:

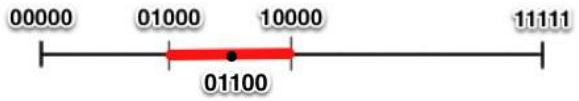

Figure 6.17: Example of Binary Implementation of Arithmetic Coding

The third symbol is 0. Now the subinterval falls into middle half and straddles middle points. So we need to expand the middle half and slide the input window:

Figure 6.18: Example of Binary Implementation of Arithmetic Coding

“01000” falls into symbol 1’s subinterval, so the next symbol is 1:

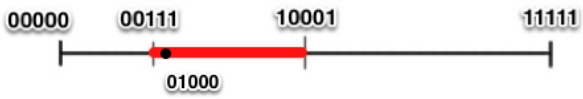

Figure 6.19: Example of Binary Implementation of Arithmetic Coding

Then ”01000” falls into symbol 0’s subinterval:

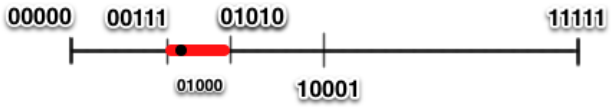

Figure 6.20: Example of Binary Implementation of Arithmetic Coding

We get the last symbol 0.

The symbol sequence we get is 11010. It is correct after the codec process.

REFERENCES

[1] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," IEEE Trans. Signal Processing, vol. 41, pp. 3445–3462, Dec. 1993.

[2] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," IEEE Trans. Circuits Syst. Video Technol., vol. 6, pp. 243–250, June 1996.

[3] S. Servetto, K. Ramchandran, and M. T. Orchard, "Wavelet based image coding via morphological prediction of significance," in Proc. IEEE Int. Conf. Image Processing, Oct. 1995, pp. 530–533.

[4] B.-B. Chai, J. Vass, and X. Zhuang. 1997. Significance-linked connected component analysis for low bit rate image coding. Multimedia Signal Processing, IEEE First Workshop on 23-25, pp. 145-150

[5] A. Skodras. 2001. The JPEG 2000 still image compression standard. Signal Processing Magazine, IEEE, vol. 18, issue. 5, pp. 36-58

[6] M. Vetterli and J. Kovacevic. 1995. Wavelets and Subband Coding, Prentice Hall

[7] M. Rabbani and P. W. Jones, Digital Image Compression Techniques, SPIE Optical Engineering Press, Bellingham, WA, 1991.

[8] W. Pennebaker, "JPEG technical specification, revision 8," Aug. 1990, Working Document No. JTC1/SC2/WG10/JPEG-8-R8.

[9] O. Rioul and M. Vetterli, "Wavelets and signal processing," IEEE Signal Processing Mag., vol. 8, pp. 14–38, Oct. 1991.

[10] S. G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," IEEE Trans. Pattern Anal. Machine Intell., vol. 11, pp. 674–693, July 1989.

[11] I. H. Witten, M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," Commun. ACM, vol. 30, pp. 520–540, June 1987.

[12] I. H. Witten, R. Neal, J. G. Cleary, "Arithmetic coding for data compression," Comm. ACM, vol. 30, pp. 520-540, June 1987.

[13] Junqiang Lan and Xinhua Zhuang. 2003. Embedded SLCCA for wavelet image coding. Multimedia and Expo, ICME '03. International Conference on vol. 2, pp. 357-369

[14] B. B. Chai, J. Vass and X. Zhuang, "Significance-Link Connected Component Analysis for Wavelet Image Coding," IEEE Trans. on Image Processing, Vol. 8, No. 6, pp.774-784, 1999.

[15] X. Zhuang, "Digital Image Compression" Tech. Rep., Univ. Missouri, Columbia, Dec. 2008.

[16] L. Vincent. 1993. Morphological grayscale reconstruction in image analysis: Applications and effective algorithms. IEEE Trans. Image Processing, vol. 2, pp. 176–201

[17] B. B. Chai, Significance-Linked Connected Component Analysis and Morpho-Subband Decomposition for Multiresolution Image Compression, Ph. D dissertation, Dec. 1997.

[18] M. Rabbani and P. W. Jones. 1992. Conditioning contexts for the arithmetic coding of bit planes. IEEE Trans. Signal Processing, vol. 40, no. 1, pp. 232-236

[19] T. Strutz. 2014. Entropy based merging of context models for efficient arithmetic coding. IEEE International Conference on Acoustic, Speech and Signal Processing,

[20] H. Y. Meng and Z. H. Wang. 2000. Fast spatial combinative lifting algorithm of wavelet transform using the 9/7 filter for image block compression. Electronics Letters, 36(21), pp. 1766-1767,

[21] H.G. Musmann, “Predictive image coding,” in Image Transmission Techniques, W. K. Pratt, Ed., pp. 73-122. Academic Press, Orlando, FL, 1979.

[22] M. Rabbani, L. A. Ray, and J. R. Sullivan, “Adaptive predictive coding with applications to radiographs,” Medical Instrumentation, vol. 20, pp. 182-191, 1986.

[23] W. H. Chen and W. K. Pratt, “Scene adaptive coder,” IEEE Transactions on Communications, vol. COM-32, no. 3, pp. 224-232, 1984.

[24] R. J. Clarke, Transform Coding of Images, Academic Press, London, 1985.

[25] J. W. Woods and S. D. O’Neil, “Subband coding of images,” IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 34, no. 5, pp. 1278-1288, Oct. 1986.

[26] V. Sze, M. Budagavi, and G. J. Sullivan Editors. High Efficiency Video Coding (HEVC) Algorithms and Architectures, Springer, ISSN: 1558-9412

[27] D. S. Taubman and M. W. Marcellin. 2002. JPEG2000: Image Compression Fundamentals, Standards, and Practice, Kluwer Academic Publishers, pp. 354-360

[28] C. Dong, C. C. Loy, K. He, and X. Tang. 2014. Learning a deep convolutional network for image super-resolution. in ECCV, pp. 184–199. 2, 7, 20, 22

[29] C. Dong, C. C. Loy, K. He, and X. Tang. 2016. Image super-resolution using deep convolutional networks. IEEE Transactions on Pattern Analysis and Machine Intelligence

[30] Topic 10 : Image Processing http://hosting.soonet.ca/eliris/remotesensing/bl130lec10.html

[31] JPEG YCbCr Support https://msdn.microsoft.com/en-us/library/windows/desktop/dn424131(v=vs.85).aspx

[32] http://www.avsforum.com/forum/139-display-calibration/1505166-do-most-tvs-now-support-4-4-4-ones-original-thread-not-updated-4.html

[33] https://www.researchgate.net/figure/Third-level-filter-bank-block-diagram-representation-of-a-DWT-and-b-UWT_fig6_264460565

VITA

Xiaobo Jiang was born in China and grew up in Shandong province. He received a Bachelor's degree in Electrical Engineering and Automation from Beijing University of Aeronautics and Astronautics in 2008, a Master's degree in Electrical and Computer Engineering from University of Missouri – Columbia in 2011, and a Doctorate in Computer Science from University of Missouri – Columbia in 2018. He was a graduate assistant in College of Engineering, University of Missouri – Columbia during his graduate studies period.