Image Compression

Ali Ghezawi

Ece 572 – Digital Image Processing

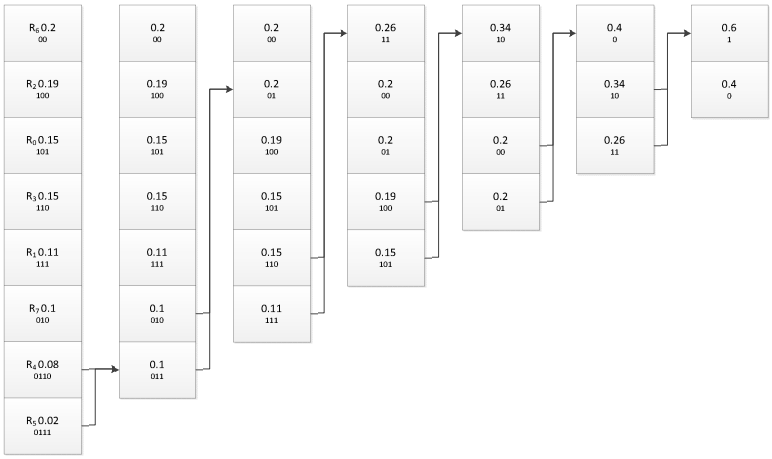
November 21, 2011

100/100

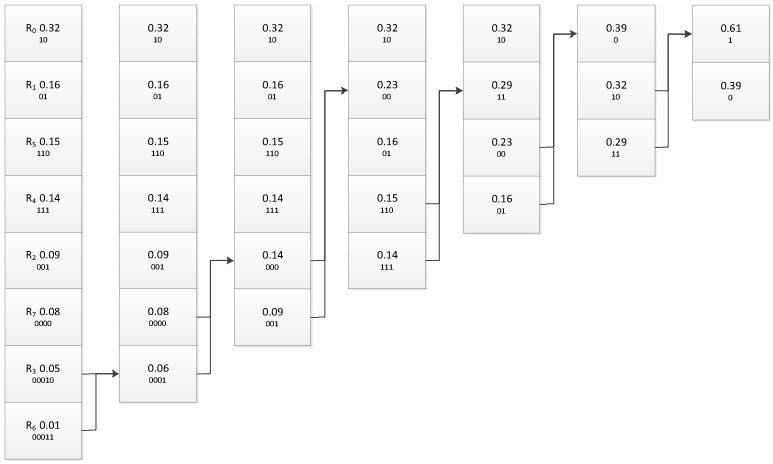
I. Huffman Coding

I.I Huffman Code

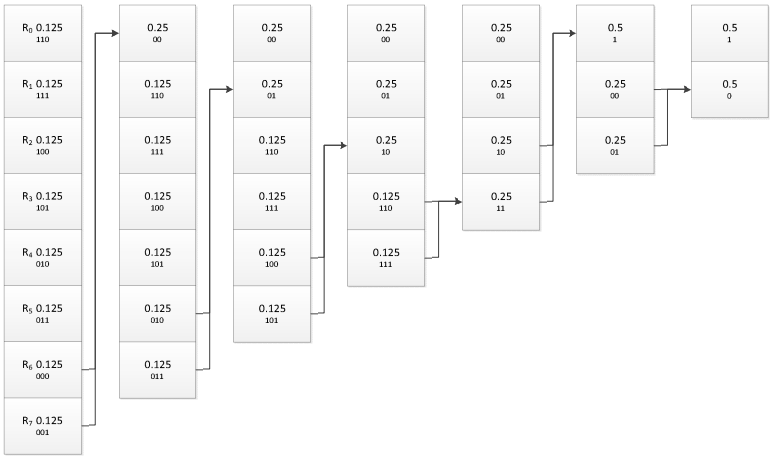
I.I.I Example One



I.I.II Example Two



I.I.III Example Three



I.II Entropy

$$H(A)=\sum_{j\in S}p_A(a_j)\log_2\left(\frac{1}{p_A(a_j)}\right),S=\{0,\ldots,L-1\}$$

I.II.I Example One

$$H(A)=\begin{pmatrix}0.15\log_2\left(\frac{1}{0.15}\right)+0.11\log_2\left(\frac{1}{0.11}\right)+0.19\log_2\left(\frac{1}{0.19}\right)+0.15\log_2\left(\frac{1}{0.15}\right)+\\0.08\log_2\left(\frac{1}{0.08}\right)+0.02\log_2\left(\frac{1}{0.02}\right)+0.2\log_2\left(\frac{1}{0.2}\right)+0.1\log_2\left(\frac{1}{0.1}\right)\end{pmatrix}=\boxed{2.8276\text{ bits}}$$

I.II.II Example Two

$$H=\begin{pmatrix}0.32\log_2\left(\frac{1}{0.32}\right)+0.16\log_2\left(\frac{1}{0.16}\right)+0.09\log_2\left(\frac{1}{0.09}\right)+0.05\log_2\left(\frac{1}{0.05}\right)+\\0.14\log_2\left(\frac{1}{0.14}\right)+0.15\log_2\left(\frac{1}{0.15}\right)+0.01\log_2\left(\frac{1}{0.01}\right)+0.1\log_2\left(\frac{1}{0.08}\right)\end{pmatrix}=\boxed{2.6434\text{ bits}}$$

I.II.III Example Three

$$H=\begin{pmatrix}0.125\log_2\left(\frac{1}{0.125}\right)+0.125\log_2\left(\frac{1}{0.125}\right)+0.125\log_2\left(\frac{1}{0.125}\right)\\+0.125\log_2\left(\frac{1}{0.125}\right)+0.125\log_2\left(\frac{1}{0.125}\right)+0.125\log_2\left(\frac{1}{0.125}\right)\\+0.125\log_2\left(\frac{1}{0.125}\right)+0.125\log_2\left(\frac{1}{0.125}\right)\end{pmatrix}=\boxed{3\text{ bits}}$$

I.III Fixed-length Coding Length

$$\bar{L}=\sum_{k=0}^{L-1}l(r_k)p_r(r_k)=l(r_a)\because l(r_a)=l(r_b),a\in S\wedge b\in S,S=\{0,\ldots,L-1\}$$

$$\therefore\bar{L}=\boxed{3\text{ bits}}$$

I.IV Huffman Coding Length

$$\bar{L}=\sum_{k=0}^{L-1}l(r_k)p_r(r_k)$$

I.IV.I Example One

$$\bar{L}=2\cdot0.2+3(0.19+0.15+0.15+0.11+0.1)+4(0.08+0.02)=\boxed{2.9\text{ bits}}$$

I. IV.II Example Two

$$\bar{L} = 2(0.32 + 0.16) + 3(0.15 + 0.14 + 0.09) + 4 \cdot 0.08 + 5(0.05 + 0.01) = \boxed{2.72 \text{ bits}}$$

I. IV.III Example Three

$$\bar{L} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k) = l(r_a) \because l(r_a) = l(r_b), a \in S \wedge b \in S, S = \{0, \ldots, L-1\}$$

$$\therefore \bar{L} = \boxed{3 \text{ bits}}$$

I.V Conclusions

Uncertainty increases with a decrease in information. Information decreases with an increase in entropy. For a set of *n* probabilities, entropy increases to its maximum the closer all probabilities are to the reciprocal of *n*, and it decreases to its minimum when one of the probabilities equals one. In general, the probabilities have lower entropy when most of the probability is concentrated in fewer events. Showing this trend, example problem two had the least entropy largely due to the events with probability 0.32 and 0.16.

II. Integrated Approach

II.I Coding Redundancy

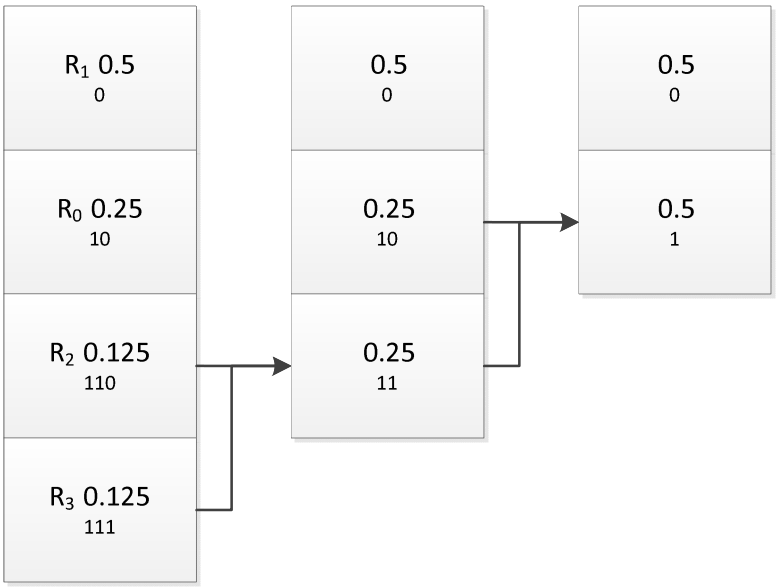
Shown in part II.II, the Huffman code is not the same as the current coding. Hence, there is coding redundancy.

The three edges of the image that are not the left edge have maximal spatial correlation since the pixels there all have the same values. This spatial correlation means there is a decent amount of interpixel redundancy.

II.II Probability Distribution

Pixel Level	Original Code	Probability
R_0	00	$\frac{1}{4}$
R_1	01	$\frac{1}{2}$
R_2	10	$\frac{1}{8}$
R_3	11	$\frac{1}{8}$

II.III Huffman Code



II.IV Average Coding Length

II.IV.I Fixed-length

$$\bar{L} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k) = l(r_a) \because l(r_a) = l(r_b), a \in S \wedge b \in S, S = \{0, \ldots, L-1\}$$

$$\therefore \bar{L} = \boxed{2 \text{ bits}}$$

II.IV.II Huffman Code

$$\bar{L} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$

$$0.5 + 2 \cdot 0.25 + 3(0.125 + 0.125) = \boxed{1.75 \text{ bits}}$$

II.V Impact of Huffman Coding

ratio:

$$\frac{rc2}{rc1.75} = \boxed{1.143}$$

$$1 - \frac{1.75}{2} = \boxed{0.125}$$

II.VI Combinational Compression

II.VI.I Original Image

1	1	1	1	1	1	1	1
2	0	3	0	0	3	0	1
2	0	1	0	1	0	0	1
2	0	1	2	3	1	3	1
2	0	1	3	2	1	0	1
2	0	1	1	1	1	3	1
0	3	0	3	0	2	0	1
1	1	1	1	1	1	1	1

II.VI.II Predictive Encoding

1	0	0	0	0	0	0	0
2	-2	3	-3	0	3	-3	1
2	-2	1	-1	1	-1	0	1
2	-2	1	1	1	-2	2	-2
2	-2	1	2	-1	-1	-1	1
2	-2	1	0	0	0	2	-2
0	3	-3	3	-3	2	-2	1
1	0	0	0	0	0	0	0

II.VI.III Binary Version of Predictive Encoding

001	000	000	000	000	000	000	000
010	110	011	101	000	011	101	001
010	110	001	111	001	111	000	001
010	110	001	001	001	110	010	110
010	110	001	010	111	111	111	001
010	110	001	000	000	000	010	110
000	011	101	011	101	010	110	001
001	000	000	000	000	000	000	000

II.VI.IV Exclusive OR

001	000	000	000	000	000	000	000
011	101	010	111	000	010	111	001
011	101	001	100	001	100	000	001
011	101	001	001	001	101	011	101
011	101	001	011	100	100	100	001
011	101	001	000	000	000	011	101
000	010	111	010	111	011	101	001
001	000	000	000	000	000	000	000

II.VI.V Binary Images

II.VI.V.I Image One

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

II.VI.V.II Image Two

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

II.VI.V.III Image Three

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

II.VI.VI Run Length Coding

II.VI.VI.I Image One

01702112203112108043103322114017

II.VI.VI.II Image Two

80112121017015110121401511528

II.VI.VI.III Image Three

81111211111111211311111231115121111118

II.VI.VII Probability Distributions

II.VI.VII.I Image One

Symbol	Probability
0	7/32
1	10/32
2	6/32
3	4/32
4	2/32
7	2/32
8	1/32

II.VI.VII.II Image Two

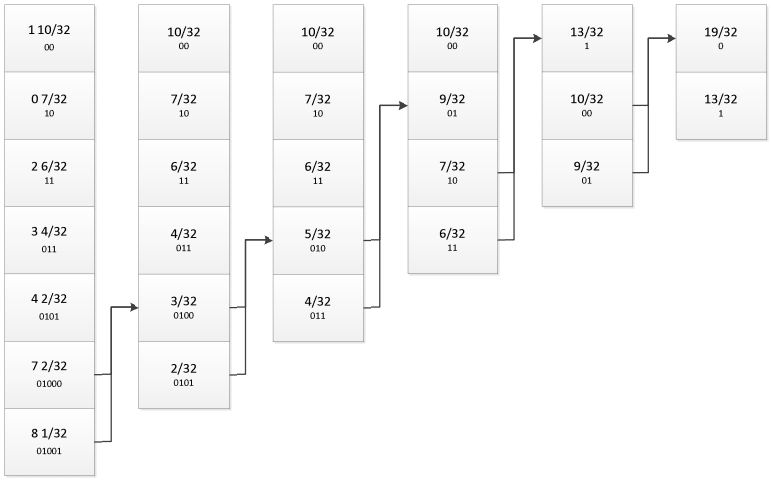
Symbol	Probability
0	5/30
1	14/30
2	4/30
4	1/30
5	3/30
7	1/30
8	2/30

II.VI.VII.III Image Three

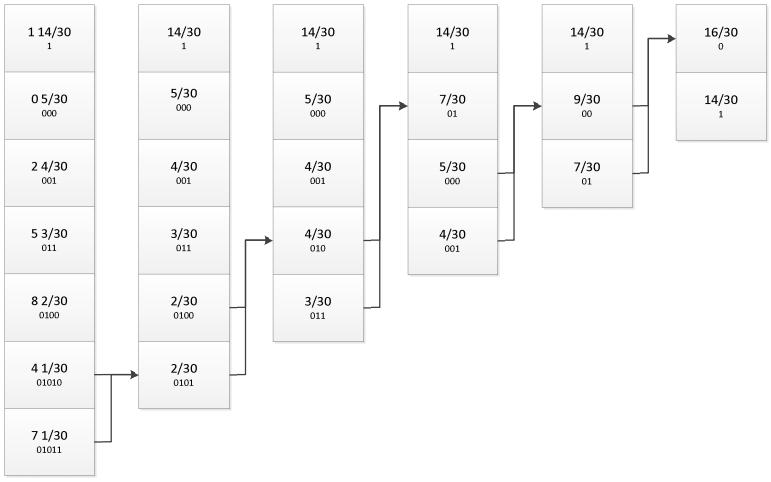
Symbol	Probability
1	29/38
2	4/38
3	2/38
5	1/38
8	2/38

II.VI.VIII Huffman Coding

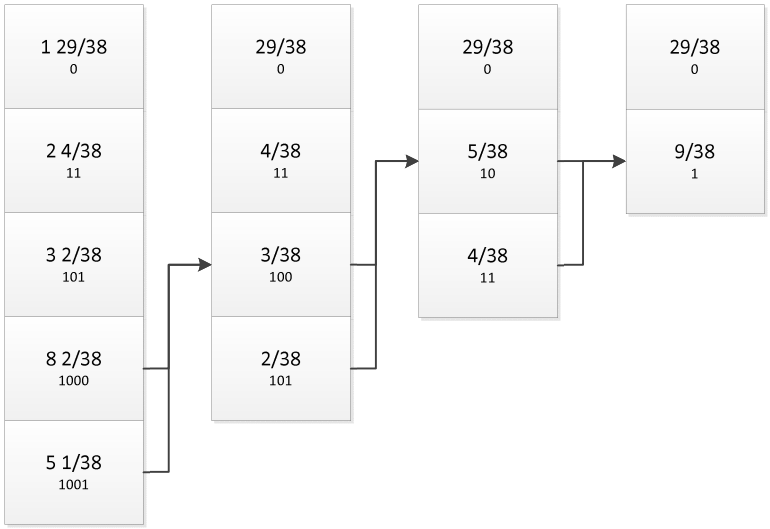
II.VI.VIII.I Image One



II.VI.VIII.I Image Two



II.VI.VIII.I Image Three



II.VI.IX Final Image Size

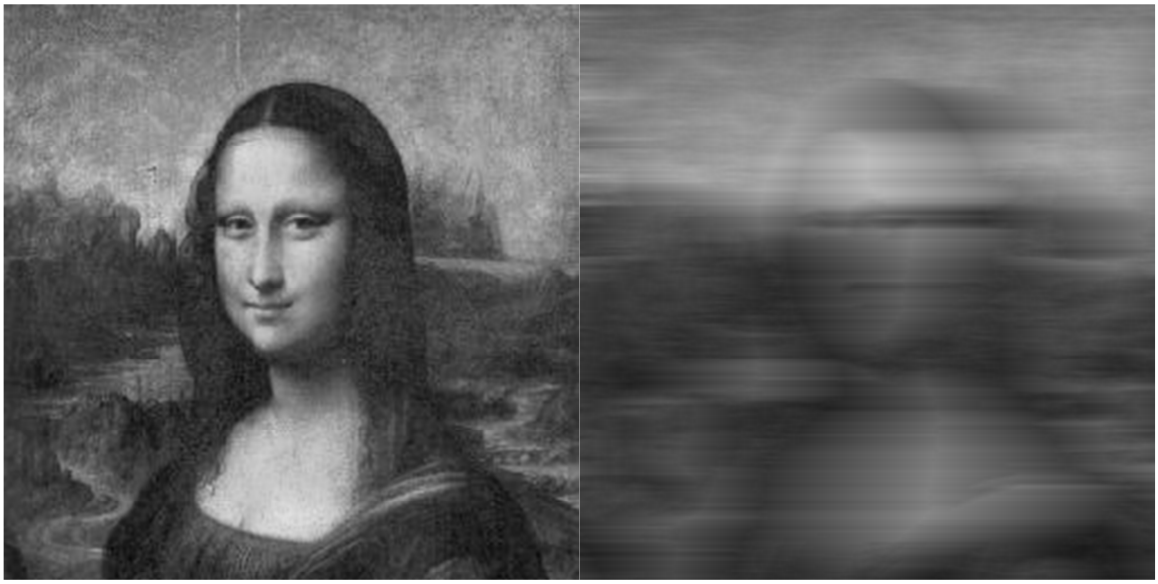
$$\left(7\cdot 2+10\cdot 2+6\cdot 2+4\cdot 3+2\cdot 4+2\cdot 5+1\cdot 5+5\cdot 3+14\cdot 1+ 4\cdot 3+1\cdot 5+3\cdot 3+1\cdot 5+2\cdot 4+29\cdot 1+4\cdot 2+2\cdot 3+1\cdot 4+2\cdot 4\right)=\boxed{204 \text{ bits}}$$

versus uncompressed

$$64\cdot 2=\boxed{128 \text{ bits}}$$

III. DM Lossy Predictive Coding

The results from decompressing a compressed image with various delta values in a delta-modulation predictive coder are found in the figure below. When delta was too small, almost any slope at all outpaced delta, causing a blurring effect across each row, yet when it was too great, it introduced a blotchy artifact throughout the image since it could not increment or decrement slowly enough.



(a) Original Image (b) Delta = 1

(c) Delta = 3 (d) Delta = 15

Fig. 1 Results from Delta Modulation Compression and Decompression.

```
/*************************************************************
 *  compression.cpp: compression
 *
 * Author: Ali Ghezawi (C) aghezawi@utk.edu
 *
 * Created: nov/12/11
 *
 *************************************************************/

#include "Dip.h"

/*
 * Returns an image encoded DM style
 * @param1 input image to encode
 * @param2 alpha
 * @param3 delta value
 */
Image dmEncode(const Image& IN_IMG, const float ALPHA, const float DELTA)
{
        Image encoded(IN_IMG.getRow(), IN_IMG.getCol(), IN_IMG.getChannel());

        for(int chan = 0; chan < IN_IMG.getChannel(); ++chan)
                for(int row = 0; row < IN_IMG.getRow(); ++row)
                {
                        int last;
                        for(int col = 0; col < IN_IMG.getCol(); ++col)
                                if(!col)
                                {
                                        last = IN_IMG(row, col, chan);
                                        encoded(row, col, chan) = IN_IMG(row, col,
chan);
                                }
                                else
                                {
                                        int err = IN_IMG(row, col, chan) - ALPHA*la
st > 0 ? 1 : 0;
                                        encoded(row, col, chan) = err;
                                        last = err ? DELTA + last : last - DELTA;
                                }
                }

                return encoded;
}

/*
 * Returns an image decoded DM style
 * @param1 input image to decode
 * @param2 alpha used
 * @param3 delta value
 */
Image dmDecode(const Image& IN_IMG, const float ALPHA, const float DELTA)
{
        Image decoded(IN_IMG.getRow(), IN_IMG.getCol(), IN_IMG.getChannel());

        for(int chan = 0; chan < IN_IMG.getChannel(); ++chan)
                for(int row = 0; row < IN_IMG.getRow(); ++row)
                        for(int col = 0; col < IN_IMG.getCol(); ++col)
                                if(!col)
                                        decoded(row, col, chan) = IN_IMG(row, col,
chan);
                                else
                                        decoded(row, col, chan) = IN_IMG(row, col,
chan) ? decoded(row, col - 1, chan) + DELTA : decoded(row, col - 1, chan) - DELTA;

                return decoded;
}
```

APPROVED