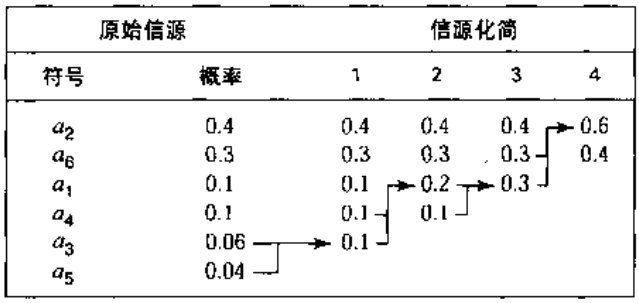


图 8.11 霍夫曼信源化简

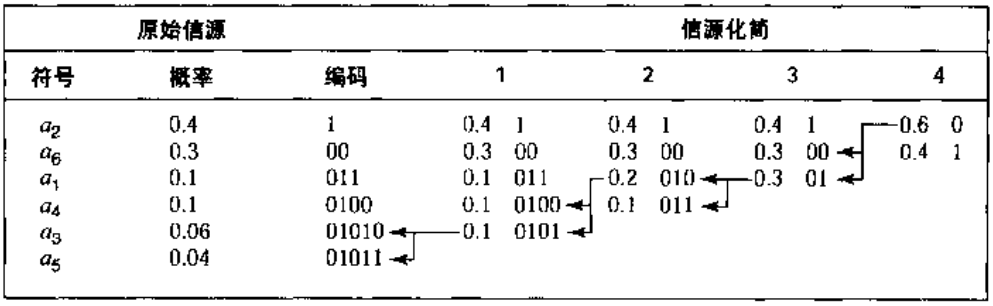


图 8.12 霍夫曼编码分配过程

然后,对每个化简后的信源重复这种操作直到达到原始信源。图 8.12 最左边显示了最终编码。这个编码的平均长度为:

$$L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5)$$
$$= 2.2 \text{ 比特 / 符号}$$

其信源的熵为 2.14 比特/符号。根据式(8.3.21)得到的霍夫曼编码的效率为 0.937。

霍夫曼编码过程对一组符号产生最佳编码,并且概率服从一次只能对一个符号进行编码的限制。在编码都建立了之后,编码和/或解码就简单地以查询表的方式完成了。编码本身是一种瞬时惟一的可解块编码。它之所以称为一种块编码,是因为每个信源符号都映射到一个编码符号的固定序列中。这种编码是瞬时的,因为符号串中的每个码字无须参考后继符号就可以进行解码。它又是惟一可解码的,因为任何符号串只能以一种方式进行解码。因此,任何霍夫曼编码的符号串,可以通过从左到右的方式对串中每个符号进行分析来解码。对于图 8.12中的二值编码,对编码串 010100111100 从左到右的扫描显示,第一个有效码字为 01010,这个编码的符号是 a_3。下一个有效的编码是 011,它所对应的符号为 a_1。以这种方式持续下去得到的完整解码信息是 $a_3 a_1 a_2 a_2 a_6$。

其他接近最佳的变长编码

当要对大量的符号进行编码的时候,构造最佳二值霍夫曼编码不是一件简单的工作。对有 J 个信源符号的一般情况,必须进行 $J-2$ 次的信源化简(见图 8.11)和 $J-2$ 次编码分配(见图 8.12)。因此对具有 256 个灰度级的图像构造最佳霍夫曼编码需要 254 次信源化简和 254 次编码分配。考虑到这项工作在计算上的复杂性,牺牲编码效率以换取编码构造的简单性有时是必要的。

表 8.5 说明了四种变长编码，它提供了这样一种折中方案。注意，霍夫曼编码的平均长度——表的最后一行——小于列出的其他编码。自然二进制码具有最大的平均长度。另外，霍夫曼技术达到的 4.05 比特/符号的编码速率接近由式(8.3.3)计算出来的 4.0 比特/符号的信源熵的底线，这个数字列在表的底部。尽管表 8.5 中没有其他编码的效率能与霍夫曼编码相比，但所有的编码都易于构造。像霍夫曼编码技术一样，它们都将最短码字分配给最容易出现的信源符号。

表 8.5　变长编码

信源符号	概　率	二值编码	霍夫曼编码	截取霍夫曼编码	B_2 编码	二值移位	霍夫曼移位
块 1							
a_1	0.2	00000	10	11	C00	000	10
a_2	0.1	00001	110	011	C01	001	11
a_3	0.1	00010	111	0000	C10	010	110
a_4	0.06	00011	0101	0101	C11	011	100
a_5	0.05	00100	00000	00010	C00C00	100	101
a_6	0.05	00101	00001	00011	C00C01	101	1110
a_7	0.05	00110	00010	00100	C00C10	110	1111
块 2							
a_8	0.04	00111	00011	00101	C00C11	111000	0010
a_9	0.04	01000	00110	00110	C01C00	111001	0011
a_{10}	0.04	01001	00111	00111	C01C01	111010	00110
a_{11}	0.04	01010	00100	01000	C01C10	111011	00100
a_{12}	0.03	01011	01001	01001	C01C11	111100	00101
a_{13}	0.03	01100	01110	100000	C10C00	111101	001110
a_{14}	0.03	01101	01111	100001	C10C01	111110	001111
块 3							
a_{15}	0.03	01110	01100	100010	C10C10	111111000	000010
a_{16}	0.02	01111	010000	100011	C10C11	111111001	000011
a_{17}	0.02	10000	010001	100100	C11C00	111111010	0000110
a_{18}	0.02	10001	001010	100101	C11C01	111111011	0000100
a_{19}	0.02	10010	001011	100110	C11C10	111111100	0000101
a_{20}	0.02	10011	011010	100111	C11C11	1111111010	0001110
a_{21}	0.01	10100	011011	101000	C00C00C00	111111110	00001111
熵	4.0						
平均码长		5.0	4.05	4.24	4.65	4.59	4.13

表 8.5 的第 5 列说明了一个对基本的霍夫曼编码策略的简单修正，它被称为截尾霍夫曼编码。一个截尾霍夫曼编码是通过对信源的具有最大概率的 ψ 个符号进行霍夫曼编码产生的，ψ 是小于 J 的正整数。后面跟有适当的固定长度编码的前缀编码用于表示其他所有的信

源符号。在表 8.5 中,将 ψ 任意地选择为 12,并且生成前缀编码作为第 13 个霍夫曼码字。即,在对 12 个最有可能出现的信源符号进行霍夫曼编码时,将概率为符号 a_{13} 到 a_{21} 的概率之和的一个"前缀符号"包括进来作为第 13 个符号。剩下的 9 个符号的编码用前缀编码进行编码,这些前缀编码变成了 10 和一个等于符号下标减去 13 的 4 位二进制值。

表 8.5 的第 6 列说明了被称为 B 编码的第二种接近最佳的变长编码。当信源符号概率遵从形式如下的幂律的时候,对某些正常数 β 和归一化常数 $c = 1/\sum_{j=0}^{J} j^{-\beta}$,这种编码接近最佳。

$$P(a_j) = cj^{-\beta} \tag{8.4.1}$$

例如,用二值表示法表示的典型打字机打印的文件的行程长度分布是指数的。如表 8.5 所显示的,每个码字都由表示为 $c = 1/\sum_{j=0}^{J} j^{-\beta}$ 的延伸比特和信息比特组成,这些位都是自然二进制数。延伸比特的惟一目的就是将独立的码字分割开,以便对串中的每个码字可以在 0 和 1 之间进行简单的交换。表 8.5 中的 B 编码称为 B_2 编码,因为每个延伸比特使用两个信息比特。对应于信源符号串 $a_{11}a_2a_7$ 的 B_2 编码序列是 001 010 101 000 010 或 101 110 001 100 110,这取决于第一位延伸比特被假设为 0 还是 1。

表 8.5 中剩下的两个变长码称为移位编码。移位编码的生成通过下列步骤:(1)对信源符号进行排列以便符号的概率呈单调递减的顺序;(2)将所有的符号分割为相等大小的符号块;(3)在所有的块中对单个元素进行相同的编码;(4)对每个块增加特定的上移和/或下移符号以便进行块的识别。解码器每次识别一个上移或下移符号,这个符号根据预先定义的基准块上移或下移一个块。

为了生成表 8.5 中第 7 列的 3 位二值移位编码,21 个信源符号根据它们出现概率的不同进行排序,并将它们分成 3 个块,每个块有 7 个符号。上部块的单独符号(a_1 到 a_7)——考虑基准块——用二进制编码 000 到 110 进行编码。第 8 个二进制编码(111)没有被包含在基准块中;相反,这个二进制编码作为一个惟一的上移控制用于识别剩下的块(此时,没有用到下移符号)。剩下的两个块中的符号使用一个或两个上移符号在用于基准块编码的二进制码组合中进行编码。例如,符号 a19 的编码是 111 111 100。

表 8.5 中第 8 列的霍夫曼移位编码是用相似的方式生成的。主要的差别是,霍夫曼编码对基准块进行编码之前要分配移位符号的概率。正常情况下,这种分配是通过将基准块之外的所有信源符号的概率进行相加实现的;即,使用定义截尾霍夫曼编码前缀符号的相同概念。这里,从符号 a_8 到 a_{21} 的概率和是 0.39。移位符号是具有最大概率的符号,且被分配一个最短的霍夫曼码字(00)。

算术编码

与前述的变长编码不同,算术编码生成的是非块码。在算术编码领域中,可以追溯到 Elias(见 Abramson[1963])的研究工作,在信源符号和码字之间是不存在一一对应的关系的。相反,算术编码是给整个信源符号(或消息)序列分配一个单一的算术码字。这个码字本身定义了一个介于 0 和 1 之间的实数间隔。当消息中的符号数目增加时,用于描述消息的间隔变得更小,而表示间隔所需要的信息单元(如,位)的数目变得更多了。

消息的每个符号根据符号出现的概率减小间隔的大小。因为这种技术不像霍夫曼编码方法那样要求将每个信源符号转换成符号的整数(即,每次对一个符号进行编码),所以这种技术达到了(仅在理论上)由 8.3.3 节中的无噪声编码准则所设定的界限。

图 8.13 说明了算术编码的基本过程。这里,一个五符号序列(或称消息)为 $a_1a_2a_3a_3a_4$,是来自一个编码的四符号信源。在编码处理的开始,消息被假定为占据整个半开区间[0,1)。如表 8.6 所示,这个区间开始时根据每个信源符号的出现概率分成四个区域。例如,符号 a_1 对应子区间[0,0.2)。因为 a_1 是消息中进行编码的第一个符号,消息的间隔开始被限制在[0,0.2)的窄区间内。因此,在图 8.13 中,区间[0,0.2)被扩展到图形的全高度,并且在区间的末端用这个狭窄域的值进行了标注。这个窄域再根据初始信源符号的出现概率进行细分,而后接着处理下一个消息符号。用这种方式,符号 a_2 将子区间变窄为[0.04,0.08),a_3 进一步将子区间变窄为[0.056,0.072),如此进行下去。最后的消息符号必须被保留以作为特定的消息结束指示符,它将子区间变窄为[0.067 52,0.068 8)。当然,在这个子区间内的任何数字(例如,0.068)都可以被用来表示这个消息。

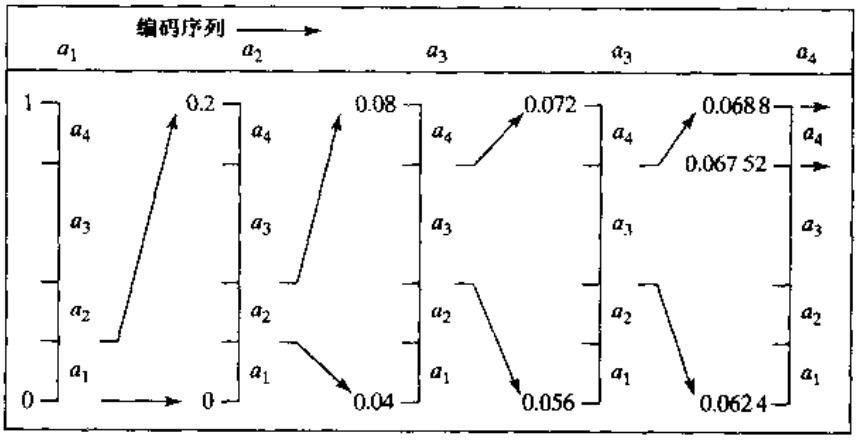


图 8.13　算术编码过程

表 8.6　算术编码示例

信 源 符 号	概　率	初始子区间
a_1	0.2	[0.0,0.2)
a_2	0.2	[0.2,0.4)
a_3	0.4	[0.4,0.8)
a_4	0.2	[0.8,1.0)

在图 8.13 中的算术编码消息中,使用三个十进制数字表示五符号消息。这样的转换得到每信源符号 3/5 或 0.6 个十进制数字,而根据式(8.3.3)与之相比的信源的熵为 0.58 个十进制数字或 10 元单元/符号。当编码序列的长度不断增加的时候,得到的算术编码也接近无噪声编码定理所设定的界限。实际上,有两个因素使编码的效能无法达到这个界限:(1)为了将一个消息同其他的消息分离开,增加了消息结束指示符,(2)使用的算法精度是有限的。算术编码的实际实现通过引入尺度策略和舍入策略解决第二个问题(Langdon 和 Risssanen[1981])。尺度策略在根据符号出现概率将子区间进行再次细分之前,将每个子区间重新归一化回[0,1)

区间的范围内。舍入策略保证根据算法的有限精度进行的截尾不会影响根据正确的表示对子区间进行编码。

8.4.2 LZW 编码

在分析了消除编码冗余的主要方法之后,现在考虑一种处理图像的像素间冗余的无误差压缩技术。这种技术叫做 Lempel-Ziv-Welch(LZW)编码,它对信源符号的可变长度序列分配固定长度码字,且不需要了解有关被编码符号的出现概率的知识。回顾 8.3.3 节中香农第一定理阐明,一个零记忆信源的第 n 次扩充可以用比非扩充信源本身更少的表示每个信源符号所需的平均比特数编码。尽管这种压缩技术必须得到美国第4588302号专利的许可,但 LZW 压缩技术已经被收入主流的图像文件格式中,其他的格式包括图形交换格式(GIF),标记图像文件格式(TIFF)和可移植文件格式(PDF)等。

LZW 编码在概念上非常简单(Welch[1984])。在编码处理的开始阶段,先构造一个对信源符号进行编码的编码本或"字典"。对 8 位的单色图像,字典中前 256 个字被分配给灰度值 0,1,2,…,255。当编码器顺序地分析图像像素的时候,字典中没有包括的灰度级序列由算法决定(例如,下边没有用到)其出现的位置。例如,如果图像前两个像素为白色,序列"255-255"可能被分配在位置为 256 的地方,这个位置下面的地址保留给灰度级 0 到 255。下一次再遇到两个连续的白色像素就用码字 256(包含序列 255-255 的位置的地址)表示它们。如果一个 9 位 512 字的字典用于编码处理,初始时用于表示两个像素的(8 + 8)位码字被一个单一的 9 位码字所代替。很明显,字典的大小是一个很重要的系统参量。如果字典太小,灰度级序列匹配会变得不太可能;如果太大,码字的尺寸反而会影响压缩性能。

例 8.12 一个 LZW 编码例子

考虑下列 4×4、8 位图像的垂直边缘:

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

表 8.7 中详细说明了对它的 16 个像素进行编码的步骤。假设一个 512 字带有下列初始内容的字典:

字典位置	条　目
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

位置 256 到 511 在开始时是用不到的。

表 8.7　LZW 编码示例

当前识别序列	被处理的像素	编 码 输 出	字典位置(码字)	字 典 条 目
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

图像通过从左到右,从上到下的顺序处理其像素进行编码。每个连续的灰度级值都与一个可变的——表 8.7 的第一列——"当前识别序列"相联系。如所见到的,这个可变序列初始时为空。在字典中搜索每个连续的序列,如果找到了,同表中第一行的情况一样,用新连接和识别(也就是,在字典中定位)的序列代替。表中第二行第一列是进行这样的操作的结果。没有生成输出代码,也没有改变字典。然而,如果没有找到连续序列,则将当前识别序列的地址作为下一个编码值输出,将连续的但未被识别的序列加到字典中,并且当前识别序列被初始化为当前像素的值。表中第二行就是这种情况。最后两列详细说明了在扫描整个 4×4 图像时加到字典中的灰度级序列。在表中定义了 9 个附加的码字。在编码的结尾部分,字典包含了 265 个码字且 LZW 算法已经成功识别了几个重复的灰度级序列——使这些序列从初始时的 128 比特图像减少到 90 比特(也就是,10 个 9 比特编码)。通过从顶部到底部的顺序读取第 3 列的编码得到编码输出。压缩率为 1.42:1。

刚才示范的 LZW 编码的独特特征是编码的字典或码书在对数据进行编码的时候创建的。很明显,一个 LZW 解码器对编码数据流进行解码的同时生成了一个统一的解压缩字典。留下这个例子给读者(见习题 8.16)作为对前面例子的输出进行解码和重构码书的练习。尽管在这个例子中不需要,但在大多数实际应用中都需要一种处理字典溢出的策略。一种简单的解决办法就是,当字典已满并用一个新的初始化的字典进行编码的时候对字典进行刷新或重新初始化。作为一种更为复杂的选择是,监测压缩的性能并在字典变得效能低下或不能接受的时候对字典进行刷新。另一方面,可以对字典中用得最少的条目进行跟踪以便在必要的时候

对其进行替换。

8.4.3 位平面编码

另一种有效的减少像素间冗余的技术就是单独处理图像的位平面。这种技术称为位平面编码。它是以将一幅多级(单色的或彩色的)图像分解为一系列二值图像并通过几种熟知的二值图像压缩方法对每幅二值图像进行压缩的原理为基础的。在这一小节中,将描述使用最为普遍的分解方法,并回顾几种更为普通的用于压缩的方法。

位平面分解

一幅 m 比特的灰度图像具有的灰度级可以用以 2 为底的多项式进行表示:

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \cdots + a_1 2^1 + a_0 2^0 \tag{8.4.2}$$

基于这种性质,将这类图像分解成一个二值图像集的一种简单方法就是,将多项式的 m 个系数分离到 m 个 1 比特的位平面之中。如在第 3 章中所提到的那样,零级位平面是通过收集每个像素的 a_0 位生成的,面第$(m-1)$级位平面包含 a_{m-1} 比特或系数。一般来讲,每个位平面都根据 0 到 $m-1$ 进行编号,并通过令平面的像素等于每个像素在原图像中对应位的值或多项式的系数进行构造。这种方法固有的缺点是图像在灰度级上稍有变化就会对位平面的复杂性产生显著的影响。例如,如果一个亮度为 127(01111111)的像素与一个亮度为 128(10000000)的像素相邻,每个位平面将包含一个对应 0 到 1(或 1 到 0)的转换。比如,当 127 和 128 的两个二进制编码最高有效位不同的时候,位平面 7 将包含与一个 1 值像素相邻的一个 0 值像素,在这一点产生一次从 0 到 1(或 1 到 0)的转换。

一种可作为替代的分解方法(可以减少小的灰度级变化带来的影响)是首先用一个 m 比特的灰度编码表示图像。这个 m 比特的灰度编码 $g_{m-1}\cdots g_2 g_1 g_0$ 对应式(8.4.2)中的多项式,可以用下列方法计算得到:

$$g_i = a_i \oplus a_{i+1} \qquad 0 \leqslant i \leqslant m-2 \tag{8.4.3}$$

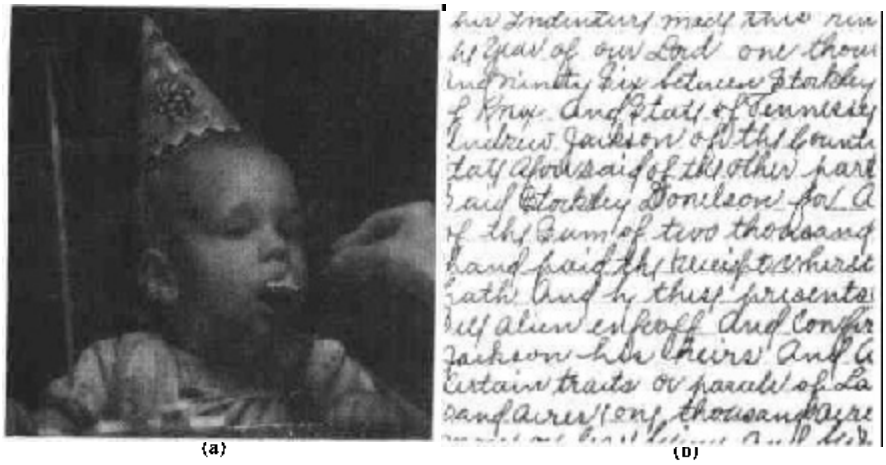
$$g_{m-1} = a_{m-1}$$

这里,$\oplus$表示异或运算。这种编码的惟一特性就是连续码字只在一比特位置上不相同。因此,灰度级小的变化不太可能影响到所有的 m 个位平面。例如,当灰度级 127 和 128 相邻的时候,只有第 7 个位平面含有一次 0 到 1 的转换,因为对应 127 和 128 的灰度编码分别为 11000000 和 01000000。

例 8.13 位平面编码

图 8.14(a)和(b)中所示的 1024×1024 大小的图像用于说明本节其余部分描述的压缩技术。一个小孩的 8 比特单色图像是使用高分辨率 CCD 照相机拍摄的。一份由总统安德鲁·杰克逊于 1796 年写的担保契约的二值图像是在一个文件扫描平台上生成的。图 8.15 和图 8.16 显示了小孩图像中的 8 个二值和灰度编码的位平面。

注意,高阶位平面比它们对应的低阶部分要简单得多。即,这些位平面中包含大块的均匀区域,其中的图像细节相当少、忙乱或随意性较小。另外,灰度编码位平面比对应的二值位平面的复杂性小。



(a)　　(b)

图 8.14　一幅 1024 × 1024 大小的(a)8 比特单色图像和(b)二值图像

常数值区域编码

压缩二值图像或位平面的一种简单但有效的方法是使用指定的码字识别大片连续的 1 和 0 区域。这样一种方法称为常数区域编码(CAC),图像被分成大小为 $p \times q$ 个像素的不同块,这些块被分成全白色、全黑色或混合亮度等不同的类。出现可能性最大或最频繁的类被分配给 1 比特码字 0,另两个类分给 2 比特码 10 和 11。因为通常用于表示每个常数区域的 pq 比特被 1 比特或 2 比特码字代替,所以可以实现压缩。当然,分配给混合亮度类的编码被作为一个前缀使用,这个前缀的后面跟着块的 pq 比特模式。

当对主要为白色的文本文件进行压缩的时候,稍微简单一些的方法是将纯白色区域的编码定为 0,面所有其他的块(包括纯黑色的块)用块的比特模式跟着的编码 1 进行编码。这种方法充分利用了对被压缩图像的预期结构化倾向而被称为白色色块跳跃(WBS)。由于不希望有太多纯黑色区域存在,因此这些区域都是与混合亮度区域聚合在一起的,允许对高概率的白色色块使用 1 比特码字。对这一过程(块大小为 $1 \times q$)的一种非常有效的改进是用 0 对纯白色线进行编码,面其他的线使用 1 后面跟通常的 WBS 编码序列进行编码。另一种方法是使用迭代的方法将二值图像或位平面不断分解为越来越小的子块。对二维块,一幅纯白色的图像用编码 0 表示,面其他的图像被分为子块并使用 1 为前缀进行相似的编码。即,如果一个子块是全白色的,色块的表示将使用前缀 1 表示此块为第一次迭代的子块,后面跟着编码 0,表示色块为纯白色。如果子块不是纯白色的,则分解过程将重复下去直到达到预先定义的子块尺寸,且通过 0(如果色块是全白色)或 1 后面跟着块比特模式进行编码。

一维行程编码

对常数区域进行编码的一种有效的替代方法是用一长度序列表示图像或位平面的每一行,这些长度描绘了对黑色和白色像素的连续行程。这种技术称为行程编码,它是在 20 世纪 50 年代发展起来的一种技术,连同这种技术在二维方面的扩展,现已成为传真(FAX)编码的标准压缩方法。基本概念是,对从左到右扫描一行时所遇到的 1 或 0 的连接组,使用这些连接组的长度进行编码,并且建立决定行程值的约定。决定行程长度值最通常的方法是:(1)指定每一行第一次行程的值,或(2)假设每一行从白色行程开始,这次行程的长度可能实际上为 0。

图 8.15 图 8.14(a)中图像的 4 幅最明显的二值(左列)和灰度编码(右列)位平面

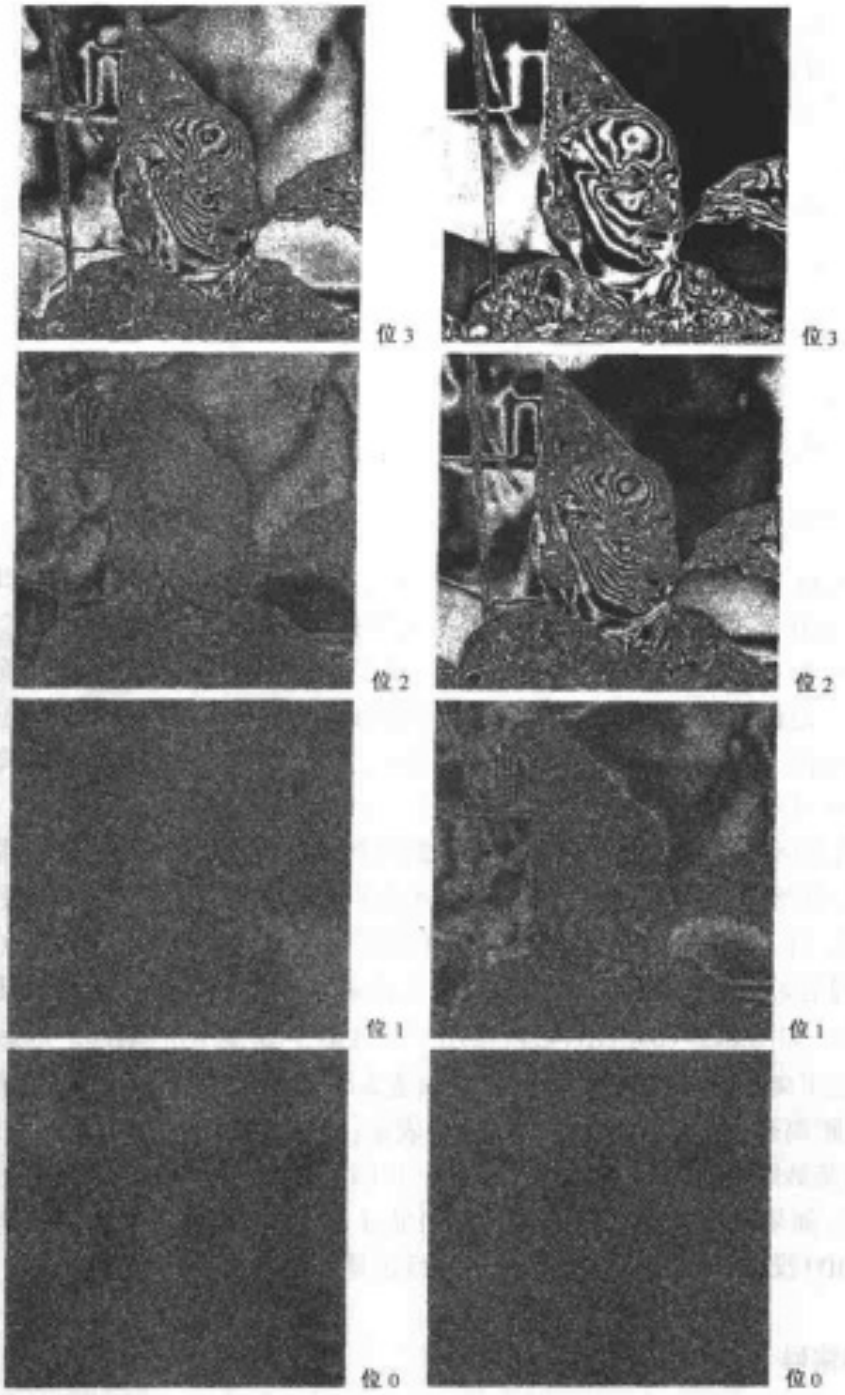


图 8.16 图 8.14(a)中图像的 4 幅最不明显的二值(左列)和灰度编码(右列)位平面