

第七届

# 全国大学生集成电路创新创业大赛

报告类型\*: 算法设计说明

参赛杯赛\*: 景嘉微杯

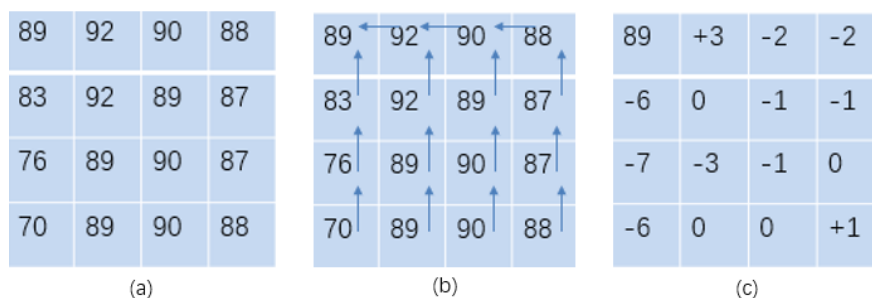
作品名称\*: 基于霍夫曼编码标志位的预测图像压缩算法

队伍编号\*: CICC3012

团队名称\*: 终将青春给了她

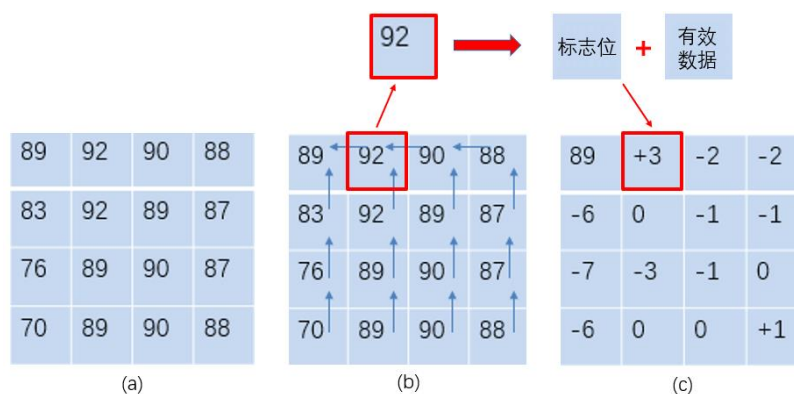
## 1. 灵感来源

我们的算法基于预测编码，以图表 1 的 4\*4 像素矩阵为例。原始矩阵(a)每一个矩阵元素都需要 1 字节存储空间，总共为 16Byte。然而，通常情况下，同一区域内的像素值基本相似，因此使用存储差值的方式（图中是比较经典的相邻元素作差）所需要的 bit 数会更少。例如，当差值为 0 时，只需要 0bit 来存储该差值；当差值为 3 时（暂不考虑符号，仅以绝对值作计算），只需要 2bit 来存储该差值；当差值为 6 时，需要 3bit 来存储差值。那么，相较于每个矩阵元素需要 8bit（即 1 字节）的原像素存储，存储每个差值所需的比特数明显减少很多。



图表 1

为了解决在解码过程中确定每个差值所使用的比特数这一关键问题，我们采用添加标志位的方法。例如下图中红框标记的矩阵元素“92”，我们用差值“+3”去替代原始值“92”。在解码过程中，利用前面已有的元素“89”再加上差值“+3”就可以还原到“92”。然而，我们不能只存储“+3”，而是需要知道“+3”这一差值在存储中占用了多少 bit。在有效数据“+3”之前存储一个标志位（请注意这里的一个标志位并不是指的 1 比特标志位），解码过程中，先解出标志位，然后根据标志位确定后续有效数据所占用的比特数。这样就能保证解码的正确性。以上是我们算法的思路，具体的算法细节将在后文中详细介绍。



## 2. 具体算法

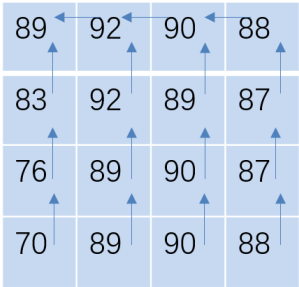
需要指出的是，为了清晰地展示我们的算法原理，我们在本章节中详细地介绍了我们的算法从一个最初版本不断优化的过程。但在实现函数说明和算法测试两份报告中我们只会展示最终版本的相关内容。

我们还是对相邻元素进行作差，并且在列的方向上的相邻元素进行作差，如图表 2 左侧示意图。我们将差值数据根据差值的大小和正负分为了七类，这七类通过标志位进行区分，标志位之后存储有效数值，也就是差值的绝对值大小。比如差值为“+3”，那么我们存储的就是 2bit 标志位“00”以及 2bit 有效数值“11”，总体而言就是“0011”；差值为“-12”，那么我们存储的就是 4bit 标志位“1110”以及 4bit 有效数值“1100”，总体而言就是“11101100”。这就是我们算法的 2.0 版本（1.0 版本是实验版本，后文将介绍更多的改进版本）。

标志位采用了霍夫曼编码，原因如下：

1. 从图像分布的角度讲，相邻像素之间，像素值的变化主要以缓变为主，因此相邻像素的差值越小，一般概率越高，霍夫曼编码可以让概率高的标志位占用更少的比特数。
2. 霍夫曼编码虽然是不定长编码，但由于其前缀编码的特性，也就是任何一个编码不是其他编码的前缀，在解码过程中不会产生歧义，因此不需要知道每个编码所占用比特数。

差值范围	标志位	有效数值bit数	总bit数
0	10	0	2
[1,3]	00	2	4
[-3,-1]	01	2	4
[4,15]	110	4	7
[-15,-4]	1110	4	8
[16,255]	11110	8	13
[-255,-16]	11111	8	13

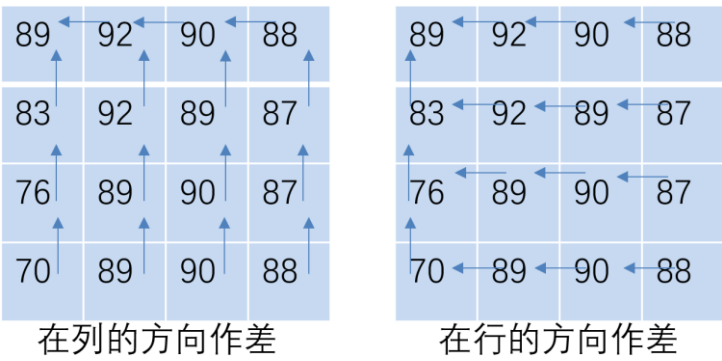


作差方向

差值分类表

图表 2

图表 2 左侧的作差方向我们称之为在列的方向上作差。实际上，块在行方向上也可以进行作差（如图表 3 右侧所示），并且压缩这些行方向上的差值有可能获得更好的压缩效果。因此，我们对每个块同时进行两个方向的作差，根据得到的差值矩阵，分别计算两个方向的压缩率，取压缩率更小的方向作为该块的作差方向。这是我们算法的 **3.0 版本**。



图表 3

在我们的实际应用中发现，有相当一部分的像素块，它们的标志位矩阵（每个差值元素所对应的标志位组成的矩阵），在一行或一列（取决于该块选择的作差方向）中的标志位是完全相同的。我们决定利用这一点来再次优化我们的压缩算法，也就是对一行或一列的标志位再进行一次独立的压缩。我们对一行或一列的标志位向量增加了一个判断位，该判断位使用三个元素的霍夫曼编码来表示，也就是区分了三种情况（如图表 4 所示）。图表 4 中的 X 表示八个全同的标志位的比特数，或者有一个标志位不同时 7 个相同的标志位的比特数。Y 表示有一个标志位不同时，不同的那个标志位所占用的比特数。也就是说，针对八个标志位全同的情况，先前的版本需要  $8X$  bit，而当前的版本只需要  $1+X$  bit；针对八个标志位中只有一个不同的情况，先前的版本需要  $7X+Y$  bit，而当前的版本只需要  $2+3+X+Y$  bit（需要额外的 3bit 来表示不同的标志位的位置）；针对八个标志位中超过一个不同的情况，需要比先前的版本多 2 bit。这是我们算法的 **4.0 版本**。

判断位	含义	处理后的bit数
0	一行/一列中所有的标志位全部相同	$1+X$
10	一行/一列中有一个标志位不同	$2+3+X+Y$
11	一行/一列中有多个标志位不同	$2+\text{原始比特数}$

图表 4

在我们的实际应用中还发现，RGB 三个通道的纹理分布是相近的。这意味着在我们的算法中，RGB 三个通道的标志位矩阵是相似的，也就是标志位矩阵只会有少数几个元素不同。这时，如果已经存储了 B 通道的标志位矩阵，那么 G 和 R 通道的标志位矩阵就不用完整地进行存储，只需要存储与 B 标志位矩阵不同元素的具体数值和位置。这是我们算法的 **5.0 版本**。