ICS 35.040
CCS L 71

团　体　标　准

T/AI 127.5—2024

信息技术 视觉特征编码
第 5 部分：语义分割图

Information technology – Visual feature coding

Part 5: Semantic segmentation map

2024 - 07 - 09 发布　　2024 - 07 - 09 实施

中关村视听产业技术创新联盟　发布

目　　次

前　　言

本文件按照 GB/T 1.1-2020《标准化工作导则 第1部分：标准化文件的结构和起草规则》的规定起草。

本文件是T/AI 127《信息技术 视觉特征编码》的第5部分。T/AI 127已经发布了如下部分：

——第 2 部分: 手工设计特征；

——第 3 部分: 深度学习特征；

——第 4 部分：深度特征图；

——第 5 部分：语义分割图；

——第 6 部分：结构点序列。

本文件由新一代人工智能产业技术创新战略联盟AI标准工作组提出。

本文件由中关村视听产业技术创新联盟归口。

本文件起草单位：中国科学技术大学，鹏城实验室，北京大学，上海交通大学，青岛海信网络科技股份有限公司。

本文件起草人：刘东，杨闰宇，闫宁，吴枫，段凌宇，白燕，林巍峣，刘士湛，王雯雯，赵海英，崔晓冉。

引　言

本文件规定了对视觉特征表示与编码技术的规范，旨在确立适用于手工设计特征、深度学习特征、深度特征图、语义分割图、结构点序列的视觉特征压缩规范，以及特征码流设计与系统构建规范，拟由六个部分组成：

——第1部分：系统。目的在于设计特征编码系统，提供整合特征码流的规范，实现特征高效交互与协同应用。

——第2部分：手工设计特征。目的在于确立适用于传统手工设计特征的表示与压缩标准。

——第3部分：深度学习特征。目的在于确立适用于从深度学习模型中提取的高维特征向量的表示与压缩标准。

——第4部分：深度特征图。目的在于确立适用于从深度学习模型中提取的通用深度特征图的表示与压缩标准。

——第5部分：语义分割图。目的在于确立适用于语义分割图的高效表征与无损压缩标准。

——第6部分：结构点序列。目的在于确立适用于结构点序列的时空域高效表征与压缩标准。

信息技术 视觉特征编码
第5部分：语义分割图

1 范围

本文件规定了语义分割图无损编码解码过程和编码格式。

本文件适用于图像和视频数据中目标检测、目标追踪、场景理解和辅助图像与视频编码等应用。

2 规范性引用文件

本文件没有规范性引用文件。

3 术语和定义

下列术语和定义适用于本文件。

3.1

语义分割图 semantic map

以不同像素值代表不同目标对象的图像。

3.2

四叉树划分 quadtree partition

将图像块递归划分为大小相同的四个图像块的划分方法，广泛应用于视频编码。

3.3

链式编码 chain code

用于描述图像中目标轮廓的算法。

3.4

像素 pixel

原始图像或转换图像的最小单元，每个像素包含空间坐标信息及其亮度值。

3.5

像素值 pixel value

像素的亮度值。

3.6

颜色值 color value

像素值在编码过程中的映射值。

3.7

帧 frame

视频信号空间信息的表示，由一个亮度样本矩阵（Y）构成或由一个亮度样本矩阵（Y）和两个色度样本矩阵（U和V）构成。

3.8

位流 bitstream

编码图像所形成的二进制数据流。

4 缩略语

下列缩略语适用于本文件。

CTU：编码树单元（Coding Tree Unit）

CU：编码单元（Coding Unit）

5 约定

5.1 概述

本文件中使用的数学运算符和优先级参照C语言。但对整型除法和算术移位操作进行了特定定义。除特别说明外，约定编号和计数从0开始。

5.2 算术运算符

算术运算符定义见表1。

表 1 算术运算符定义

算术运算符	定义
+	加法运算
-	减法运算（二元运算符）或取反（一元前缀运算符）
×	乘法运算
/	整除运算，沿向0的取值方向截断。例如，7/4和-7/-4截断至1，-7/4和7/-4截断至-1
a % *b*	模运算，*a*除以*b*的余数，其中*a*与*b*都是正整数

5.3 逻辑运算符

逻辑运算符定义见表2。

表 2 逻辑运算符定义

逻辑运算符	定义
a&&*b*	*a*和*b*之间的与逻辑运算
a ‖ *b*	*a*和*b*之间的或逻辑运算
!	逻辑非运算

5.4 关系运算符

关系运算符定义见表3。

表 3 关系运算符定义

关系运算符	定义
>	大于
>=	大于或等于
<	小于
==	等于

5.5 位运算符

位运算符定义见表4。

表 4 位运算符定义

位运算符	定义
a>>*b*	将*a*以2的补码整数表示的形式向右移*b*位。仅当*b*取正数时定义此运算

5.6 赋值

赋值运算定义见0。

表 5 赋值运算定义

赋值运算	定义
=	赋值运算符
++	递增，*x*++相当于*x*= *x*+ 1。当用于数组下标时，在自加运算前先求变量值

5.7 位流语法、解析过程和解码过程的描述方法

5.7.1 位流语法的描述方法

位流语法描述方法类似C语言。位流的语法元素使用粗体字表示，每个语法元素通过名字（用下划线分割的英文字母组，所有字母都是小写）、语法和语义来描述。语法表和正文中语法元素的值用常规字体表示。

某些情况下，可在语法表中应用从语法元素导出的其他变量值，这样的变量在语法表或正文中用不带下划线的小写字母和大写字母混合命名。大写字母开头的变量用于解码当前以及相关的语法结构，也可用于解码后续的语法结构。小写字母开头的变量只在它们所在的小节内使用。

语法元素值的助记符和变量值的助记符与它们的值之间的关系在正文中说明。在某些情况下，二者等同使用。助记符由一个或多个使用下划线分隔的字母组表示，每个字母组以大写字母开始，也可包括多个大写字母。

位串的长度是4的整数倍时，可使用十六进制符号表示。十六进制的前缀是“0x”，例如“0x1a”表示位串“0001 1010”。

条件语句中0表示FALSE，非0表示TRUE。

语法表描述了所有符合本文件的位流语法的超集，附加的语法限制在相关条中说明。

描述语法的伪代码例子见表6。当语法元素出现时，表示从位流中读一个数据单元。

表 6 语法描述的伪代码

伪代码	描述符
/*语句是一个语法元素的描述符，或者说明语法元素的存在、类型和数值，下面给出两个例子。*/	
syntax_element	ue(v)
conditioning statement	
/*花括号括起来的语句组是复合语句，在功能上视作单个语句。*/	
{	
statement	
…	
}	
/*“while”语句测试condition是否为TRUE，如果为TRUE，则重复执行循环体，直到condition不为TRUE。*/	
while (condition)	
statement	
/*“do … while”语句先执行循环体一次，然后测试condition是否为TRUE，如果为TRUE，则重复执行循环体，直到condition不为TRUE。*/	
do	
statement	
while (condition)	
/*“if … else”语句首先测试condition，如果为TRUE，则执行primarY语句，否则执行alternative语句。如果alternative语句不需要执行，结构的“else”部分和相关的alternative语句可忽略。*/	
if (condition)	
primarY statement	
else	
alternative statement	
/*“for”语句首先执行initial语句，然后测试condition，如果conditon为TRUE，则重复执行primarY语句和subsequent语句直到condition不为TRUE。*/	
for (initial statement; condition; subsequent statement)	
primarY statement	

解析过程和解码过程用文字和类似 C 语言的伪代码描述。

5.7.2 函数

以下函数用于语法描述。假定解码器中存在一个位流指针，这个指针指向位流中要读取的下一个二进制位的位置。函数由函数名及左右圆括号内的参数构成。函数也可没有参数。

CU_stream()

语义分割图编码单元（CU）的定义，具体过程的定义见章节7.2。

one_color_stream()

CU仅有一种颜色值的定义，具体过程的定义见章节7.3。

chain_code_stream()

CU满足链式编码条件的定义，链式编码条件见章节7.2，具体过程的定义见章节7.3。

min_cu_stream()

CU达到最小设定值且不满足链式编码条件的定义，具体过程的定义见章节7.3。

predict_color1()

预测CU左上角像素的颜色值，具体过程见章节7.3。返回值为-1代表无法进行预测。在其余情况下均可进行预测，返回值为预测的数目，为1或2。同时返回预测的颜色值。

predict_initial_position()

预测链式编码模式的分界边初始点，具体过程见章节7.3。返回值为-1代表无法进行预测。在其余情况下均可进行预测，返回值为预测的数目，为1或2。同时返回预测的位置。

predict_color2()

预测链式编码模式下另一颜色值，具体过程见章节7.3。返回值为-1代表无法进行预测。在其余情况下返回值为预测的颜色值。

edge_in_cu()

判断分界边是否编码完成，具体过程见章节7.3。返回值为1代表没有完成，返回值为0代表已经完成。

5.7.3 描述符

描述符表示不同语法元素的解析过程，见表7。

表 7 描述符

描述符	说明
u(n)	n位无符号整数，用高位在前的二进制表示。实际编码使用一个自适应算数编码二值模型编码n位。
ae(v)	自适应算数编码的语法元素。解析过程在7.2中定义

5.7.4 保留、禁止和标记位

本文件定义的位流语法中，某些语法元素的值被标注为“保留”（reserved）或“禁止”（forbidden）。

“保留”定义了一些特定语法元素值用于将来对本文件的扩展。这些值不应出现在符合本文件的位流中。

“禁止”定义了一些特定语法元素值，这些值不应出现在符合本文件的位流中。

“标记位”（marker_bit）指该位的值应为‘1’。

位流中的“保留位”（reserved_bits）表明保留了一些语法单元用于将来对本文件的扩展，解码处理应忽略这些位。“保留位”不应出现从任意字节对齐位置开始的21个以上连续的‘0’。

6 语法和语义

6.1 语义分割图语法

语义分割图语法见表8。

表 8 语义分割图语法

语义分割图语法	描述符
semantic_map_stream(){	
image_h	u(16)
image_w	u(16)
frame_num	u(16)
image_type	u(1)
if(frame_num>1) **mode**	u(1)
color_num	u(8)
for(i = 0; i < color_num; i++){	
color_map[i]	u(8)
}	
if(color_num > 1){	
m_numCtuInRow = (image_h + CTU_size - 1) / CTU_size	
m_numCtuInCol = (image_w + CTU_size - 1) / CTU_size	
total_num_ctu = m_numCtuInRow × m_numCtuInCol /*总共 CTU 数目*/	
for(frame = 0; frame < frame_num; frame++){	
for(ctuIdx = 0; ctuIdx < total_num_ctu; ctuIdx++){	
location_row = (ctuIdx / m_numCtuInCol) × CTU_size	
location_col = (ctuIdx % m_numCtuInCol) × CTU_size	
CU_stream(location_row, location_col, 0)	
}	
}	
}	
}	

其中，编码单元 CU_stream()，语法见表 9。

表 9 编码单元语法

编码单元语法	描述符
CU_stream(location_row, location_col, depth){ //depth 为划分深度，每次递归深度加一	
side_length = CTU_size >> (depth+1)	
if(location_row + 2 × side_length > image_h || location_col + 2 × side_length > image_w){	
if(location_row + side_length >= image_h && location_col + side_length >= image_w)	
CU_stream(location_row, location_col, depth + 1)	
else if(location_row + side_length >= image_h && location_col + side_length < image_w){	
CU_stream(location_row, location_col, depth + 1)	
CU_stream(location_row, location_col + side_length, depth + 1)	

表 9 编码单元语法（续）

编码单元语法	描述符
}	
else if(location_row + side_length < image_h && location_col + side_length >= image_w){	
CU_stream(location_row, location_col, depth + 1)	
CU_stream(location_row + side_length, location_col, depth + 1)	
}	
else{	
CU_stream(location_row, location_col, depth + 1)	
CU_stream(location_row, location_col + side_length, depth + 1)	
CU_stream(location_row + side_length, location_col, depth + 1)	
CU_stream(location_row + side_length, location_col + side_length, depth + 1)	
}	
}	
else if(CTU_size >> depth == min_split_size){	
one_color_flag	ae(v)
if(one_color_flag){	
one_color_stream()	
}	
else{	
chain_code_flag	ae(v)
if(chain_code_flag)	
chain_code_stream(min_split_size)	
else	
min_cu_stream()	
}	
}	
else{	
split_flag	ae(v)
if(!split_flag){	
one_color_flag	ae(v)
if(one_color_flag)	
one_color_stream()	
else	
chain_code_stream(CTU_size >> depth)	
}	
else{	
CU_stream(location_row, location_col, depth + 1)	
CU_stream(location_row, location_col + side_length, depth + 1)	
CU_stream(location_row + side_length, location_col, depth + 1)	
CU_stream(location_row + side_length, location_col + side_length, depth + 1)	
}	
}	
}	

其中，单颜色编码 one_color_stream()，语法见表 10。

表 10 单颜色编码语法

单颜色编码语法	描述符
one_color_stream(){	
predict= predict_color1()	
if(predict == -1)	
color	ae(v)
else{	
predict_color1_true_flag1	ae(v)
if(!predice_color1_true_flag1 && color_num > 2){	
if(predict == 2){	
predict_color1_true_flag2	ae(v)
if(!predict_color1_true_flag2)	
color	ae(v)
else	
color	ae(v)
}	
}	

其中，链式编码 chain_code_stream()，语法见表 11。

表 11 链式编码语法

链式编码语法	描述符
chain_code_stream(side_length){	
one_color_stream()	
if(side_length == 2){	
chain_2x2_mode	ae(v)
}	
else{	
predict_mode = predict_initial_position()	
if(predict_mode > 0){	
predict_initial_position_true_flag	ae(v)
if(predict_initial_position_true_flag){	
if(predict_mode == 2)	
predict_initial_position_side	ae(v)
predict_initial_position_mode	ae(v)
}	
}	
if(predict_mode == -1 || ! predict_initial_position_true_flag){	
position_side	ae(v)
initial_position	ae(v)
}	
}	
if(color_num > 2){	
color2= predict_color2()	
if(color2 == -1)	
color	ae(v)

表 11 链式编码语法（续）

链式编码语法	描述符
else{	
predict_color2_true_flag	ae(v)
if(!predict_color2_true_flag)	
color	ae(v)
}	
}	
if(side_length > 2){	
first_direction	ae(v)
while(edge_in_cu()){	
rest_direction	ae(v)
}	
}	
}	

其中，最小 CU 编码 min_cu_stream()，语法见表 12。

表 12 最小 CU 编码语法

最小 CU 编码语法	描述符
min_cu_stream(){	
one_color_stream()	
if(color_num > 2){	
for(i = 1; i < 4; i++)	
color	ae(v)
}	
}	

6.2 语义分割图语法

image_w

表示原始图像的宽度（单位：像素）。

image_h

表示原始图像的高度（单位：像素）。

frame_num

表示视频帧的数目。

image_type

表示图像类型，0 表示 YUV400，1 表示 YUV420。

mode

是否使用帧间预测，0 表示不使用，1 表示使用。目前为规定为 0。

color_num

语义分割图类别总数。

color_map[i]

顺序扫描图像，第 i+1 次新出现的像素值。为便于使用自适应算数编码，在编码开始时将图像像素值用 color_map 替换为此像素值出现的顺序序号，即颜色值；在解码结束前将颜色值用 color_map 替换为对应像素值。

CTU_size

CTU 的边长，默认为 128。

min_split_size

CU 的最小边长，默认为 2。

one_color_flag

判断 CU 是否只有一种颜色值，1 为只有一种颜色。

chain_code_flag

判断 CU 是否满足链式编码条件，1 为满足条件，链式编码条件见章节 7.2。

split_flag

判断是否继续进行四叉树划分，1 为继续划分，0 为停止划分，划分条件见章节 7.2。

color

颜色值在颜色值列表中的序号，颜色值列表及其调整方法见章节 7.3。

predict_color1_true_flag1

CU 左上角颜色值的预测标志位，1 表示预测正确，0 表示预测错误。预测方法见章节 7.3。

predict_color1_true_flag2

CU 左上角颜色值的第二预测标志位，1 表示预测正确，0 表示预测错误。预测方法见章节 7.3。

chain_2x2_mode

CU 大小为 2×2 且为链式编码模式时的情形，情形的序号见章节 7.3。

predict_initial_position_true_flag

链式编码模式下 CU 分界边初始点预测标志位，1 表示预测正确，0 表示预测错误。预测方法见章节 7.3。

predict_initial_position_side

链式编码模式下 CU 分界边初始点所在边，1 表示上相邻边，0 表示左相邻边。

predict_initial_position_mode

链式编码模式分界边初始点位置，如果预测初始点为左上角一点相邻点或最下（右）一点，则使用 0 表示此点，1 表示此点相邻一点。如果预测初始点不为如上条件点，则使用 0 表示此点，1 表示相邻下（右）一点，2 表示相邻上（左）一点。

position_side

表示分界边初始点所在边的序号减一，序号见章节 7.3。

initial_position

表示分界边初始点所在边上的位置减一，位长与 CU 大小有关，与 position_side 共同确定分界边初始点。对 4×4 的 CU 块进行了特殊的设计。在图 4 序号 1，2，3 边有 3 种取值，序号 4 边有 2 种取值。

predict_color2_true_flag

链式编码模式下 CU 的另一颜色值的预测标志位，1 表示预测正确，0 表示预测错误。预测方法见章节 7.3。

first_direction

链式编码模式下表示分界边的第一个方向，解码方法见章节 7.3。

rest_direction

链式编码模式下表示连续的方向值，解码方法见章节 7.3。

7 语义分割图解码

7.1 语义分割图数据

语义分割是计算机视觉中的一个重要课题，它是场景理解的关键组成部分。图像经过语义分割可以得到语义分割图。语义分割图中相同的像素值代表相同的目标对象。语义分割图中包含了每种目标对象的位置、类别以及形状信息，可以应用于包括自动驾驶和目标跟踪等广泛的领域。本文件定义的解码过程输出语义分割图的数据形式为YUV400格式或UV分量都为0的YUV420格式，每个Y分量像素值代表一个语义类别。目前参考软件支持最多255种不同像素值的语义分割图无损解码，且图像长与宽皆为4的倍数，如果不为4的倍数，需要自行将输入语义分割图的长宽补为4的倍数。

7.2 解码框架

在编码开始时将图像像素值用color_map映射为此像素值出现的顺序序号便于自适应算术编码器编码，下文称其为颜色值；在解码结束前将颜色值用color_map替换为对应像素值。所以在解码过程中首先需要重建出所有的颜色值。此方案采用四叉树划分的方法进行划分与边界处理。首先将语义分割图划分为若干无重叠的，长宽均为128像素的正方形块（可出边界），命名为编码树单元(CTU)，然后将CTU依条件判断是否划分为四个大小相同的正方形块，命名为编码单元(CU)，对CU也递归地进行同样的操作。函数CU_stream()定义了CTU/CU的解码流程。如果CTU/CU存在区域在边界外则自动划分。最终可解码的CU满足的条件为如下四点之一。

a）CU仅有一种颜色值。

b）CU满足链式编码条件，即同时满足：

1) CU内有两种颜色值；

2) 两种颜色都为四连通；

3) 两种颜色都与CU的边界相连。

c）CU大小达到设定最小值（设定为2×2）。

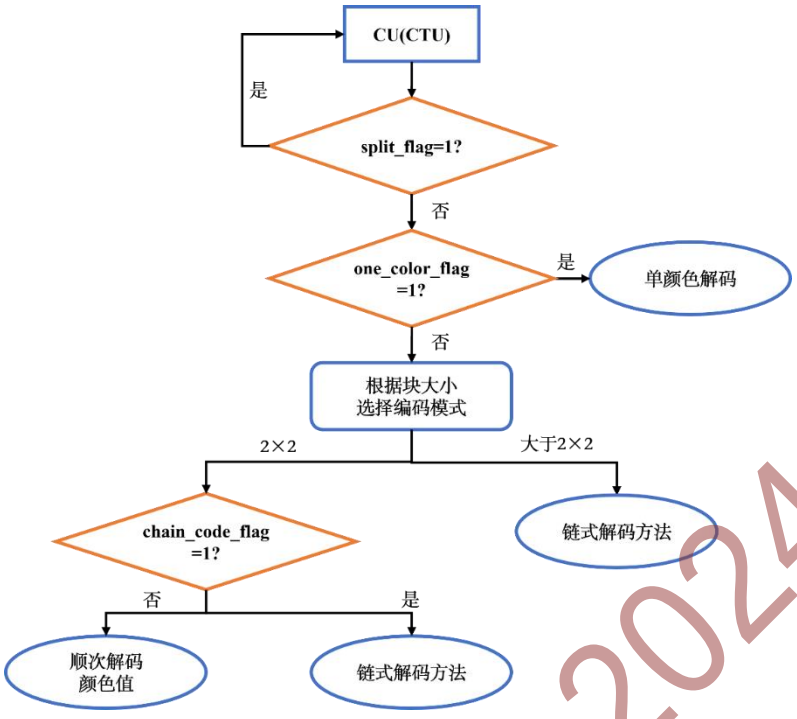

图1 解码框架图

解码框图见图1。根据模式选择位对CU使用不同方法进行解码，解码方案中所有的语法元素都使用了自适应算数编解码方法。自适应算数编解码器使用32位整形算数编解码器，见参考文献[1]。

7.3 解码工具的具体实现

7.3.1 四叉树划分

如果CU需要进行四叉树划分，将此CU划分为四个大小相同的CU。用一个标志位表示CU是否进行进一步划分，标志位为1代表划分，标志位为0代表不划分。如果已达到最小CU不再划分。如果CU跨边界则自动划分，不需要解码标志位。四叉树划分结构的一个实例见图2。为了进一步提升压缩性能，根据此CU左侧CU和上侧CU的划分深度，选择三个自适应算数编码模型之一进行解码。具体选择方法是如果同时满足存在左侧CU，左侧CU划分深度比当前CU深和存在上侧CU，上侧CU划分深度比当前CU深，则选择序号2的算数编码模型；如果这两个条件仅满足一个，则选择序号1的算数编码模型；否则选择序号0的算数编码模型。

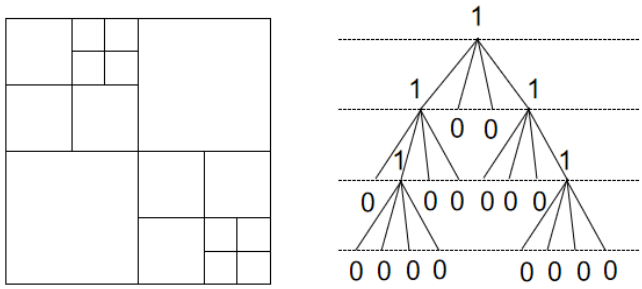

图 2 四叉树划分结构实例

7.3.2 单颜色解码

对于仅有一种颜色的 CU，需要解码此颜色值，函数 one_color_stream()定义了次方法的解码流程。为了降低码率，首先对颜色值进行预测。用 CU 左上角外三个颜色值预测 CU 左上角的颜色值，见图 3。

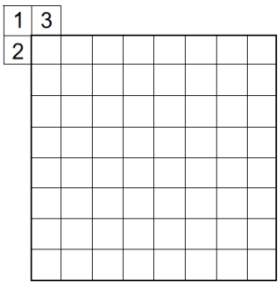

图 3 预测颜色值

预测颜色值的方法如下：

a）如果像素 2 的值等于像素 3 的值，预测为像素 2 的值。

b）如果像素 1 的值等于像素 2 的值不等于像素 3 的值，第一预测值为像素 3 的值，像素 2 的值为第二预测值。

c）如果像素 1 的值等于像素 3 的值不等于像素 2 的值，第一预测值为像素 2 的值，像素 3 的值为第二预测值。

d）如果三个颜色都不一样则不进行预测。

e）如果缺失其中的两个像素，则用第三个颜色值进行预测。

f）如果三个像素都缺失，即 CU 位于图像左上角，则不进行预测。

对于有两个预测值的情况，使用首先用一个标志位表示是否为第一预测值，如果为否，则用另一个标志位表示是否为第二预测值。函数 predict_color1()定义了预测流程。如果预测错误或没有进行预测，且语义分割图有多于两种颜色值，则解码颜色值在颜色值列表中的序号。如果语义分割图仅有两种颜色值则预测错误后无需解码。所有 CU 都用此方法预测左上角颜色值。在单颜色解码后，使用 move-to-front (MTF)变换处理颜色值列表，即将解码颜色值置于列表首位，之前列表中位于解码颜色值之前的颜色值在列表中后移一位。

7.3.3 链式解码方法

链式解码需要解码两种颜色值（第一种颜色值为左上角的颜色值，第二种颜色值为另一种颜色值）以及两种颜色的分界边（位于第二种颜色内，分隔开两种颜色的 8-连通链）。函数 chain_code_stream ()定义了次方法的解码流程。如果 CU 的大小大于 2×2，则解码方法包括如下四个步骤：

a）用如上解码单颜色的方法解码左上方颜色值（第一种颜色值）。

b）解码分界边初始点。

见图 4，按顺序扫描 CU 的四条边，颜色值改变点的位置即为初始点。初始点首先需要进行预测。预测方法是分别扫描 CU 左相邻边和上相邻边，找到颜色值与 CU 左上角不一样的值的位置，再根据找到位置的相邻两点优化预测，具体过程见图 5，相邻两点的选取如图中像素 1、2 所示。如果像素 1 与找到的点颜色值相同则预测初始点为找到的点右下方的位置；如果像素 2 与找到的点颜色值相同则预测初始点为找到的点的右（下）一点；如果像素 1 与像素 2 的颜色值都与找到的点颜色值不同则预测初始点为找到的点右上方（左下方）位置。

如果预测初始点位置位于 CU 边界外或 CU 左上角一点，则将预测调整为其相邻的位于同边界内的位置。预测分界边初始点的相邻两侧的两点也为预测点。函数predict_initial_position()定义了预测流程。具体解码方法是：

1）解码是否预测正确的标志位。如果预测正确继续之后步骤的解码，否则解码分界边初始点的位置。

2）如果左相邻边和上相邻边均可预测初始点，则解码相邻边的序号，0 为左相邻边，1 为上相邻边。

3）如果预测初始点为左上角一点相邻点或最下（右）一点，则使用 0 表示此点，1 表示此点相邻一点。如果预测初始点不为如上条件点，则使用 0 表示此点，1 表示相邻下（右）一点，2 表示相邻上（左）一点。

若没有找到颜色值改变点，则直接解码分界边初始点的位置。其解码过程为首先解码分界边初始点所在边的序号减一，再解码其在所在边的具体位置减一。见图 5，直接解码分界边初始点的结果左图为（0，3），右图为（1，3）。解码所在边的具体位置使用了定长编码，根据 CU 块大小设置位长。

为了进一步降低码率，对 4×4 的 CU 块进行了特殊的设计。见图 4，由于初始点位置在序号为 1，2，3 的边有 3 种取值，序号为 4 的边有 2 种取值，这与定长编码支持的 4 种取值差别较大。所以额外设置四个算术编码模型进行编解码。

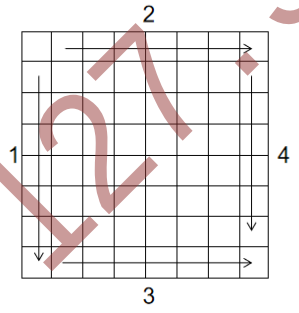

图 4 扫描顺序

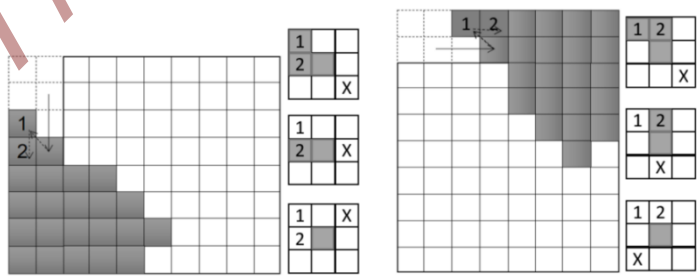

图 5 预测分界边初始点

c）解码第二种颜色值。

如果语义分割图仅有两种颜色值，则不需要解码另一种颜色。当语义分割图有多于两种颜色值时，见图4，仅在分界边初始点位于边界1或边界2时进行预测。在分界边初始点外相邻一点的两侧搜索一个颜色值作为预测，该颜色值为不同于第一种颜色值，且离此相邻点最近的颜色值。预测过程见图6（先搜索下或右点）。函数predict_color2()定义了预测流程。如

果没有搜索到与CU左上方颜色值不同的颜色值或预测错误则解码另一种颜色值在颜色值列表中的序号。在解码后，使用move-to-front (MTF)变换处理颜色值列表。

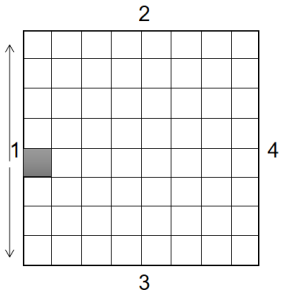

图6 预测第二种颜色值方法

d）解码分界边。

两种颜色间的分界边可由连续的方向表示，见图 7。当方向指向界外则解码结束。当方向指向的像素位于 CU 边界，根据方向指向的像素所处边界，前一方向的类别以及分界边初始点所在边界判断是否需要继续解码下一个方向值以明确分界边。CU 边界的类别见图 7。不需要解码下一个方向值的条件如下。其余的情况需要继续解码方向值以无歧义地表示分界边。

1）像素仅处于边界 1，方向指向左方或左上方，分界边初始点位于边界 1。
2）像素仅处于边界 2，方向指向上方或右上方，分界边初始点位于边界 1。
3）像素仅处于边界 2，方向指向上方或左上方，分界边初始点位于边界 2。
4）像素仅处于边界 3，方向指向下方或左下方，分界边初始点位于边界 1 或 3。
5）像素仅处于边界 3，方向指向下方或右下方，分界边初始点位于边界 2。
6）像素仅处于边界 4，方向指向右方或右下方，分界边初始点位于边界 1 或 3。
7）像素仅处于边界 4，方向指向右方或右上方，分界边初始点位于边界 2 或 4。
8）像素位于左下角或右上角或右下角。

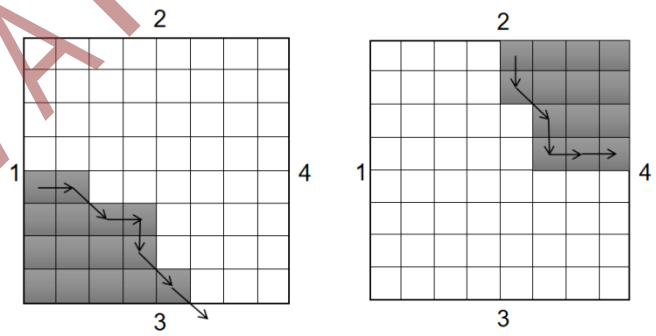

图7 解码分界边实例

函数edge_in_cu()判断分界边是否编码完成。

从分界边初始点起始，分界边的第一个方向有五种可能（初始点不在顶点时）或三种可能（初始点在三个顶点时），单独进行解码，解码方式见图8，用七个算术编码器模型分别建模。之后求下一步方向与当前方向的相对方向，将相对方向映射为0~5，具体的映射关系见图9。箭头表示之前的像素，X表示当前像素。这两个像素已经用箭头的方向解码。下一个

可能的像素位置用数字标明。上面一行表示分界边初始点位于边界1或3的方向值，下面一行表示分界边初始点位于边界2或4的方向值。用十六个算术编码器模型分别建模。

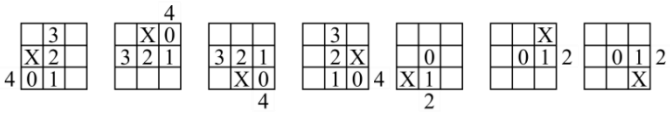

图8 第一个方向值，从左到右分别是分界边初始点X在边1、2、3、4时和在左下右上右下角时的取值

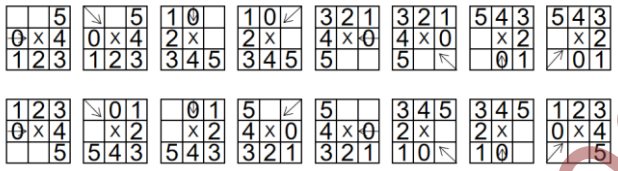

图9 第一个方向值后的方向值

如果 CU 的大小等于 2×2，见图 10，包含 6 种情况，需要解码相应序号。同样使用如上步骤 a）、c）的预测颜色值的方法。

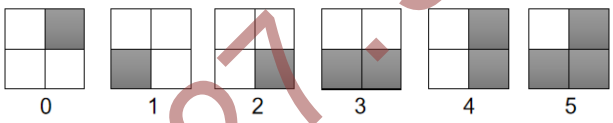

图10 CU的大小等于2×2，符合链式编码条件的6种情形

7.3.4 顺次解码颜色值

如果 CU 不满足单颜色值和链式编码条件且大小等于 2×2，直接按左上右上左下右下的顺序解码颜色值。函数 min_cu_stream()定义了次方法的解码流程。具体方法是首先用如上解码单颜色的方法解码左上方颜色值。如果颜色种类只有两种，则不需要继续解码，因为只有一种可能，即对角线为同一种颜色值。如果颜色种类大于两种，则按右上左下右下的顺序解码颜色值在颜色值列表中的序号。在每次解码颜色值后，使用 move-to-front (MTF)变换处理颜色值列表。

参 考 文 献

[1] Said, Amir, "Fast Arithmetic Coding (FastAC) Implementations," Tech. Report, 2004.
[2] B. Ya. Ryabko, "Data compression by Means of a 'Book Stack'," Jour Probl. Peredachi Inf., Vol.16, Issue 4, pp.16-21, 1980.