ICS 35.040
CCS L 71

团　体　标　准

T/SUCA 024.1-2024

信息技术 面向机器智能的数据编码
第1部分：图像

Information technology—Data coding for machines—

Part 1: Image

2024-05-20 发布　　2024-05-21 实施

深圳市 8K 超高清视频产业协作联盟　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/SUCA 024《信息技术 面向机器智能的数据编码》的第1部分，T/SUCA 024《信息技术 面向机器智能的数据编码》已经发布了以下部分：

——第1部分：图像。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别专利的责任。

本文件由深圳市8K超高清视频产业协作联盟提出并归口。

本文件起草单位：浙江大学、中国科学技术大学、中国电子技术标准化研究院、中国电信股份有限公司上海研究院、浙江大学宁波科创中心、宁波东方理工大学(暂名)、上海工程技术大学、深圳市8K超高清视频产业协作联盟、中国移动杭州研究院、中国科学院计算技术研究所、中国电信天翼视联科技有限公司、杭州海康机器人股份有限公司、华为技术有限公司、南京大学、天津大学、咪咕文化科技有限公司、中国移动通信有限公司研究院。

文件主要起草人：虞露、陈志波、范科峰、于化龙、李婧欣、张园、金鑫、董桂官、王慧芬、雷建军、纪雯、罗传飞、赵海武、王莉、赵寅、马展、邱溥业、程宝平、雷珺、李琳、刘伟东、何淇淇、刘澳、刘佳旺、冯若愚、高依欣、刘津铭、贾可、戚云鹏、李鑫、张畅、田港一、杨嘉欣。

引　言

T/SUCA 024旨在确立面向机器智能的媒体压缩的方法，拟由3个部分构成。

——第 1 部分：图像。目的在于确立面向机器智能的数据编码的图像压缩方法。

——第 2 部分：点云。目的在于确立面向机器智能的数据编码的点云压缩方法。

——第 3 部分：应用系统参考模型。目的在于确立面向机器智能的数据编码的应用系统参考模型。

面向机器智能的数据编码 第1部分：图像

1 范围

本文件规定了适应多种比特率、分辨率和质量要求的面向机器智能的图像压缩方法的解码过程。

本文件适用于安防监控、计算机视觉、无人驾驶、内容审核等面向机器智能相关应用的图像压缩。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 33475.2—2016 信息技术 高效多媒体编码 第2部分：视频

3 术语和定义

GB/T 33475.2—2016界定的以及下列术语和定义适用于本文件。

3.1

残差 residual

张量元素的重建值与其预测值（3.9）的差值。

3.2

反量化 dequantization

对被量化过的数据进行变换得到浮点数的过程。

3.3

档次 profile

T/SUCA 024中语法、语义及算法的子集。

[来源:GB/T 33475.2—2016，3.8，有修改]

3.4

图像 image

一帧组成的纹理信息的集合。

3.5

位串 bit string

由有限个二进制位组成的有序序列。

注：其最左边位是最高有效位（MSB），最右边位是最低有效位（LSB）。

[来源:GB/T 33475.2—2016，3.55，有修改]

3.6

位流 bitstream

编码图像所形成的二进制数据流。

[来源:GB/T 33475.2—2016，3.56]

3.7

预测 prediction

根据先前已解码的张量元素（3.14）估计待解码的张量元素（3.14）的具体实现。

3.8

预测补偿　prediction compensation

对语法元素解码所得张量元素（3.14）的残差（3.1）和其对应预测值（3.9）进行求和的过程。

3.9

预测值　prediction value

在张量元素（3.14）的解码过程中，用到的先前已解码的张量元素（3.14）的组合。

3.10

语法元素　syntax element

位流中的数据单元解析后的结果。

3.11

字节　byte

8位的位串（3.5）。

[来源:GB/T 33475.2—2016，3.79]

3.12

字节对齐　byte alignment

将字节（3.11）按照一定规则排列的方式。

注：从位流（3.6）的第一个二进制位开始，某二进制位的位置是 8 的整数倍。

[来源:GB/T 33475.2—2016，3.80，有修改]

3.13

特征　feature

包含通道、宽度和高度等信息的三维张量数据。

3.14

张量元素　tensor element

张量中给定通道、行和列等位置上的数据。

3.15

左上角　left-upper corner

张量中第0行第0列的空间位置。

3.16

右下角　right-bottom corner

张量中最后一行最后一列的空间位置。

3.17

超先验　hyper prior

对低分辨率图像张量数据的先验信息。

3.18

特征张量　feature tensor

图像（3.4）经过降采样得到的张量。

3.19

卷积核　convolution kernel

一个包括通道、宽和高等三维的权重张量数据。

注：用于和输入的张量进行卷积得到输出的张量中的一个通道的所有张量元素。

4 缩略语

下列缩略语适用于本文件。

CNN：卷积神经网络（Convolutional Neural Network）

CDF：累积分布函数（Cumulative Distribution Function）

FPN：特征金字塔网络（Feature Pyramid Network）

LSB：最低有效位（Least Significant Bit）

MSB：最高有效位（Most Significant Bit）

5 约定

5.1 通则

本章定义了本文件中使用的数学运算符及其优先级。在本文件中，当不使用括号改变运算符之间的优先级时，默认按照如下方式进行计算：

——优先级高的运算符先于优先级低的运算符进行计算，运算符的优先级的值越小表示该运算符的优先级越高；

——优先级相同的运算符按照从左到右的顺序依次计算。

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	优先级	定义
+	5	加法运算
-	5	减法运算（二元运算符）或取反（一元前缀运算符）
*	4	乘法运算
a^b	3	幂运算，表示a的b次幂。也可表示上标
/	4	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
÷	4	除法运算，不做截断或四舍五入
$\frac{a}{b}$	4	除法运算，不做截断或四舍五入
$\sum_{i=a}^{b} f(i)$	5	自变量i取由a到b（含b）的所有整数值时，函数$f(i)$的累加和
$a \% b$	4	模运算，a除以b的余数，其中a与b都是正整数
[ ]	4	向上取整

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	优先级	定义
!	2	逻辑非运算
express ? *a* : *b*	11	如果表达式*express*的结果为真或不为0，则使用*a*进行赋值；否则，使用*b*进行赋值

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	优先级	定义
>	7	大于
>=	7	大于或等于
<	7	小于
<=	7	小于或等于
==	8	等于
!=	8	不等于

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	优先级	定义
&	9	与运算
|	10	或运算
~	2	取反运算
a >> *b*	6	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
a << *b*	6	将*a*以2的补码整数表示的形式向左移*b*位。仅当*b*取正数时定义此运算

5.6　赋值

赋值运算定义见表5。

表 5　赋值运算定义

赋值运算	优先级	定义
=	12	赋值运算符
++	1	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
--	1	递减，x--相当于x = x – 1。当用于数组下标时，在自减运算前先求变量值
+=	12	自加指定值，例如x += 3相当于x = x + 3，x += (-3)相当于x = x + (-3)
-=	12	自减指定值，例如x -= 3相当于x = x - 3，x -= (-3)相当于x = x - (-3)

5.7 数学函数

数学函数定义见公式（1）和公式（2）。

$$\mathrm{Ceil}(x)=\lceil x\rceil \quad \cdots\cdots (1)$$

式中：

x ——自变量x。

$$\mathrm{Clip3}(i,j,x)=\begin{cases} i & ; \quad x<i \\ j & ; \quad x>j \\ x & ; \quad \text{其他} \end{cases} \quad \cdots\cdots (2)$$

式中：

x ——自变量x；

i ——下界；

j ——上界。

5.8 结构关系符

结构关系符定义见表6。

表 6 结构关系符

结构关系符	定义
->	例如：a->b表示a是一个结构，b是a的一个成员变量

5.9 位流语法、解析过程和解码过程的描述方法

5.9.1 描述方法

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。

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。

函数的参数使用下划线分割的英文字母组表示，所有字母都是小写。

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。

表7给出了描述语法的伪代码例子。当语法元素出现时，表示从位流中读一个数据单元。

表 7 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。*/	

表 7　语法描述的伪代码（续）

伪代码	描述符
syntax_element	ne(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	
/*“break”语句用于do-while、while和for循环体中，可使当前循环体立即终止循环。*/	
break	

解析过程和解码过程用文字和与语法描述相同的伪代码描述。

5.9.2 函数

5.9.2.1 概述

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。

5.9.2.2 byte_aligned( )

如果位流的当前位置是字节对齐的，返回TRUE，否则返回FALSE。

5.9.2.3 next_bits(n)

返回位流的随后n个二进制位，MSB在前，不改变位流指针。如果剩余的二进制位少于n，则返回0。

5.9.2.4 read_bits(n)

返回位流的随后n个二进制位，MSB在前，同时位流指针前移n个二进制位。如果n等于0，则返回0，位流指针不前移。

函数也用于解析过程和解码过程的描述。

5.9.3 描述符

描述符表示不同语法元素的解析过程（见表8）。

表 8 描述符

描述符	说明
u(n)	n位无符号整数。在语法表中，如果n是“v”，其位数由其他语法元素值确定。解析过程在7.2中定义
f(n)	取特定值的连续n个二进制位。解析过程在7.3中定义
ne(v)	有符号整数语法元素。解析过程使用基于神经网络的概率估计和数值熵解析，在7.4中定义
le(v)	无符号整数语法元素。解析过程使用码表解析，在7.5中定义

5.9.4 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。

注：禁止某些值的目的通常是为了避免在位流中出现伪起始码。

“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’。

5.10 数据形式

5.10.1 张量和张量中的元素

张量拥有至少三个维度，以下以拥有三个维度的张量举例说明其描述方法，拥有四个维度或更多维度的张量的描述方法依此类推。

T[:c][:h][:w]代表一个通道数为c、高度为h和宽度为w的张量，其中c、h、w均为正数。当不需要强调张量的各个维度的大小时，T[:c][:h][:w]可以简写为T。

对于张量T，T[i][j][k]代表一个由i、j、k索引确定的张量元素，表示张量T中第i个通道第j行第k列的张量元素。

对于张量T，T[:c][j1:j2][:w]代表第2个维度从j1索引到(j2-1)索引确定的所有元素，其中j1为小于h的非负整数，j2为不大于h且大于j1的非负整数。当j1不出现时，默认j1=0，当j2不出现时，默认j2为h。依次类推，其他维度上由给定索引确定的所有元素的表示方法为T[i1:i2][:h][:w]、T[:c][:h][k1:k2]、T[i1:i2][j1:j2][:w]、T[i1:i2][:h][k1:k2]、T[:c][j1:j2][k1:k2]。

5.10.2 矩阵和矩阵中的元素

M[:h][:w]代表一个高度为h和宽度为w的矩阵，其中h、w均为正数。当不需要强调张量的各个维度的大小时，M[:h][:w]可以简写为M。

对于矩阵M，M[j][k]代表一个由j、k索引确定的矩阵元素，表示矩阵M中第j行第k列的矩阵元素。

对于矩阵M，M[j][:w]代表第1个维度由j索引确定的所有元素。依次类推，其他维度上由给定索引确定的所有元素的表示方法为M[:h][k]。

对于矩阵M，M[j1:j2][:w]代表第1个维度从j1索引到(j2-1)索引确定的所有元素，其中j1为小于h的非负整数，j2为不大于h且大于j1的非负整数。当j1不出现时，默认j1的值为0，当j2不出现时，默认j2的值为h。依次类推，其他维度上由给定索引确定的所有元素的表示方法为M[:h][k1:k2]。

5.10.3 列表和列表中的元素

列表L[:a]代表一个长度为a的列表，其中a为正数。当不需要强调张量的各个维度的大小时，L[:a]可以简写为L。

对于列表L，L[i]代表其中第i个列表元素。

对于列表L，L[i1:i2]代表从i1索引到(i2-1)索引确定的所有元素，其中i1为小于a的非负整数，i2为不大于a且大于i1的非负整数。当i1不出现时，默认i1=0。

5.11 二维卷积

二维卷积表示为Conv(c_in, c_out, s, k_ver, k_hor)(input, weight, bias)，其中：

——c_in 表示输入张量的通道数；

——c_out 表示输出张量的通道数；

——s 表示卷积步长；

——k_ver 和 k_hor 表示卷积核的高和宽。

Conv的输入包括：

——输入张量 input[:c_in][:h_in][:w_in]；

——卷积核权重 weight[:c_out][:c_in][:k_ver][:k_hor]；

——偏置 bias[:c_out]，偏置的缺省值为 0。

Conv的结果为：张量result[:c_out][:h_out][:w_out]，其中：

——h_out = Ceil(h_in ÷ s)；

——w_out = Ceil(h_in ÷ s)。

解析和解码过程加载了神经网络的整套模型参数。Conv的weight和bias是该整套模型参数中的一部分参数，在解析和解码过程中不再变化。此时Conv简略表示为Conv(c_in, c_out, s, k_ver, k_hor)(input)。

当在一系列操作中调用本条中定义的Conv且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时Conv简略表示为Conv(c_in, c_out, s, k_ver, k_hor)。

Conv执行以下计算操作：

```
Conv( c_in, c_out, s, k_ver, k_hor )( input, weight, bias ) {
    for ( i = 0; i < c_out; i++ ) {
        for ( j = 0; j < h_out; j++ ) {
            for ( k = 0; k < w_out; k++ ) {
                output[ i ][ j ][ k ] = 0
                for ( x = 0; x < c_in; x++ ) {
                    for ( y = 0; y < k_ver; y++ ) {
                        for ( z = 0; z < k_hor; z++ ) {
                            j1 = ( j * s ) - ( k_ver - 1 ) / 2 + y
                            k1 = ( k * s ) -( k_hor - 1 ) / 2 + z
                            temp = ( j1 < 0 || j1 >= h_in || k1 < 0 || k1 >= w_in ) ? 0 : input[ x ][ j1 ][ k1 ]
                            output[ i ][ j ][ k ] += weight[ i ][ x ][ y ][ z ] * temp
                        }
                    }
                }
                output[ i ][ j ][ k ] += bias[ i ]
            }
        }
    }
}
```

5.12 超分重组

超分重组表示为Shuffle(s_vh)(input)，其中，参数包括：s_vh表示高度和宽度的放大倍数。

Shuffle的输入为：输入张量input[:c_in][:h_in][:w_in]，且c_in应是s_vh * s_vh的整数倍。

Shuffle的结果为：张量output[:c_out][:h_out][:w_out]，其中：

——h_out = s_vh * h_in；

——w_out = s_vh * w_in；

——c_out = c_in / (s_vh * s_vh)。

当在一系列操作中调用本条中定义的Shuffle且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时Shuffle简略表示为Shuffle(s_vh)。

Shuffle执行以下计算操作：

```
Shuffle( s_vh )( input ) {
    for ( i = 0; i < c_out; i++ )
        for ( j = 0; j < h_out; j++ )
            for ( k = 0; k < w_out; k++ )
                output[ i ][ j ][ k ] = input[ i * s_vh * s_vh + ( j % s_vh ) * s_vh + ( k % s_vh )][ j / s_vh][ k / s_vh]
}
```

5.13 交叉超分重组

交叉超分重组表示为CrossUpShuffle(input)。

CrossUpShuffle的输入为：输入张量input[:c_in][:h_in][:w_in]，其中c_in应是4的整数倍。

CrossUpShuffle的结果为：张量output[:c_out][:h_out][:w_out]，其中：

——h_out = 2 * h_in；

——w_out = 2 * w_in；

——c_out = c_in / 4。

当在一系列操作中调用本条中定义的CrossUpShuffle且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时CrossUpShuffle简略表示为CrossUpShuffle( )。

CrossUpShuffle执行以下计算操作：

```
CrossUpShuffle( input ) {
    for ( i = 0; i < c_out; i++ ) {
        for ( j = 0; j < h_in; j++ ) {
            for ( k = 0; k < w_in; k++ ) {
                output[ i ][ j * 2 ][ k * 2 ] = input[ i * 4][ j ][ k ]
                output[ i ][ j * 2 ][ k * 2 + 1 ] = input[ i * 4 + 2][ j ][ k ]
                output[ i ][ j * 2 + 1 ][ k * 2 ] = input[ i * 4 + 3][ j ][ k ]
                output[ i ][ j * 2 + 1 ][ k * 2 + 1 ] = input[ i * 4 + 1][ j ][ k ]
            }
        }
    }
}
```

5.14 交叉降分重组

降分重组表示为CrossDownShuffle(input)。

CrossDownShuffle的输入为：输入张量input[:c_in][:h_in][:w_in]，其中h_in和w_in均应是2的整数倍。

CrossDownShuffle的结果为：张量output[:c_out][:h_out][:w_out]，其中：

——h_out = h_in / 2；

——w_out = w_in / 2；

——c_out = c_in * 4。

当在一系列操作中调用本条中定义的CrossDownShuffle且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时CrossDownShuffle简略表示为CrossDownShuffle( )。

CrossDownShuffle执行以下计算操作：

```
CrossDownShuffle( input ) {
    for ( i = 0; i < c_in; i++ ) {
        for ( j = 0; j < h_out; j++ ) {
            for ( k = 0; k < w_out; k++ ) {
                output[ i * 4 ][ j ][ k ] = input[ i ][ j * 2 ][ k * 2 ]
```

```
            output[ i * 4 + 1 ][ j ][ k ] = input[ i ][ j * 2 + 1 ][ k * 2 + 1 ]
            output[ i * 4 + 2 ][ j ][ k ] = input[ i ][ j * 2 ][ k * 2 + 1 ]
            output[ i * 4 + 3 ][ j ][ k ] = input[ i ][ j * 2 + 1 ][ k * 2 ]
        }
    }
  }
}
```

5.15 张量拼接

张量拼接表示为Concat(input1, input2)。

该过程的输入是两个宽度和高度大小相同的张量input1(c1, h, w)和input2(c2, h, w)。

该过程的结果是张量output(c1 + c2, h, w)。

Concat执行以下计算操作：

```
for ( i = 0; i < c1+c2; i++ )
  for ( j = 0; j < h; j++ )
    for ( k = 0; k < w; k++ )
      output[ i ][ j ][ k ] = ( i < c1 ) ? input1[ i ][ j ][ k ] : input2[ i ][ j ][ k ]
```

5.16 渗漏激活函数

渗漏激活函数表示为LeakyReLU(input)。

LeakyReLU的输入为：输入张量input[:c][:h][:w]。

LeakyReLU的结果为：张量output[:c][:h][:w]。

当在一系列操作中调用本条中定义的LeakyReLU且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时LeakyReLU简略表示为LeakyReLU( )。

LeakyReLU执行以下计算操作：

```
LeakyReLU( input ) {
  negative_slop = 0.01
  for ( i = 0; i < c_in; i++ ) {
    for ( j = 0; j < h_out; j++ ) {
      for ( k = 0; k < w_out; k++ ) {
        if( input[ i ][ j ][ k ] >= 0 )
          output[ i ][ j ][ k ] = input[ i ][ j ][ k ]
        else
          output[ i ][ j ][ k ] = negative_slop * input[ i ][ j ][ k ]
      }
    }
  }
}
```

5.17 标准激活函数

标准激活函数表示为ReLU(input)。

ReLU的输入为：输入张量input[:c][:h][:w]。

ReLU的结果为：张量output[:c][:h][:w]。

当在一系列操作中调用本条中定义的ReLU且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时ReLU简略表示为ReLU( )。

ReLU执行以下计算操作：

```
ReLU( input ) {
    for ( i = 0; i < c_in; i++ ) {
        for ( j = 0; j < h_out; j++ ) {
            for ( k = 0; k < w_out; k++ ) {
                if( input[ i ][ j ][ k ] >= 0 )
                    output[ i ][ j ][ k ] = input[ i ][ j ][ k ]
                else
                    output[ i ][ j ][ k ] = 0
            }
        }
    }
}
```

5.18 二维深度卷积

二维深度卷积表示为 DepthConv(c, s, k_ver, k_hor)(input, weight, bias)，其中：

——c 表示输入张量和输出张量的通道数；

——s 表示卷积步长；

——k_ver 和 k_hor 表示卷积核的高和宽。

DepthConv的输入包括：

——输入张量 input[:c][:h_in][:w_in]；

——卷积核权重 weight[:c][:k_ver][:k_hor]；

——偏置 bias[:c]，偏置的缺省值为 0。

DepthConv的结果为：张量output[:c][:h_out][:w_out]，其中：

——h_out = Ceil(h_in ÷ s)；

——w_out = Ceil(h_in ÷ s)。

解析和解码过程加载了神经网络的整套模型参数。DepthConv的weight和bias是该整套模型参数中的一部分参数，在解析和解码过程中不再变化。此时DepthConv简略表示为DepthConv(c, s, k_ver, k_hor)(input)。

当在一系列操作中调用本条中定义的DepthConv且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时DepthConv简略表示为DepthConv(c, s, k_ver, k_hor)。

DepthConv执行以下计算操作：

```
DepthConv( c, s, k_ver, k_hor )( input, weight, bias ) {
    for ( x = 0; x < c; x++ ) {
```

```
        for ( j = 0; j < h_out; j++ ) {
            for ( k = 0; k < w_out; k++ ) {
                output[ x ][ j ][ k ] = 0
                for ( y = 0; y < k_ver; y++ ) {
                    for ( z = 0; z < k_hor; z++ ) {
                        j1 = ( j * s ) - ( k_ver - 1 ) / 2 + y
                        k1 = ( k * s ) - ( k_hor - 1 ) / 2 + z
                        temp = ( j1 < 0 || j1 >= h_in || k1 < 0 || k1 >= w_in ) ? 0 : input[ x ][ j1 ][ k1 ]
                        output[ x ][ j ][ k ] += weight[ x ][ y ][ z ] * temp
                    }
                }
                output[ x ][ j ][ k ] += bias[ x ]
            }
        }
    }
}
```

5.19 二维整数卷积

二维整数卷积表示为IntConv(c_in, c_out, s, k_ver, k_hor)(input, weight, bias, max, shift)，其中：

——c_in 表示输入整数张量的通道数；

——c_out 表示输出整数张量的通道数；

——s 表示卷积步长；

——k_ver 和 k_hor 表示整数卷积核的高和宽。

IntConv的输入包括：

——输入整数张量 input[:c_in][:h_in][:w_in]；

——整数卷积核权重 weight[:c_out][:c_in][:k_ver][:k_hor]；

——整数偏置 bias[:c_out]，偏置的缺省值为 0；

——输入整数张量的截断值 max；

——输出整数张量的右移位数 shift[:c_out]。

IntConv的结果为：整数张量output[:c_out][:h_out][:w_out]，其中：

——h_out = Ceil(h_in ÷ s)；

——w_out = Ceil(h_in ÷ s)。

解析和解码过程加载了神经网络的整套模型参数。IntConv的weight、bias、max和shift是该整套模型参数中的一部分参数，在解析和解码过程中不再变化。此时IntConv简略表示为IntConv(c_in, c_out, s, k_ver, k_hor)(input)。

当在一系列操作中调用本条中定义的IntConv且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时IntConv简略表示为IntConv(c_in, c_out, s, k_ver, k_hor)。

IntConv执行以下计算操作：

a) 对 input 进行截断操作：

```
for ( i = 0; i < c_in; i++ )
    for ( j = 0; j < h_in; j++ )
```

```
        for ( k = 0; k < w_in; k++ )
            clipped_input[ i ][ j ][k ] = Clip3( -max, max – 1, input[ i ][ j ][k ] )
```

b) 进行二维卷积操作：

```
temp[ :c_out ][ :h_out ][ :w_out ] =
Conv( c_in, c_out, s, k_ver, k_hor )( clipped_input[ :c_in ][ :h_in ][ w_in ], weight, bias )
```

c) 对 temp 进行右移位操作：

```
for ( i = 0; i < c_out; i++ )
    for ( j = 0; j < h_out; j++ )
        for ( k = 0; k < w_out; k++ )
            output[ i ][ j ][k ] = temp[ i ][ j ][k ] >> shift[ i ]
```

5.20 二维转置卷积

二维转置卷积表示为Tconv(c_in, c_out, s_vh, k_ver, k_hor)(input, weight, bias)，其中：

——c_in 表示输入张量的通道数；

——c_out 表示输出张量的通道数；

——s_vh 表示 c_out 的高和宽相较于 c_in 的高和宽的放大倍数；

——k_ver 和 k_hor 表示卷积核的高和宽。

Tconv的输入包括：

——输入张量 input[:c_in][:h_in][:w_in]；

——卷积核权重 weight[:c_out][:c_in][:k_ver][:k_hor]；

——偏置 bias[:c_out]，偏置的缺省值为 0。

Tconv的结果为：张量output[:c_out][:h_out][:w_out]张量，其中：

——h_out = s_vh * h_in；

——w_out = s_vh * w_in。

解析和解码过程加载了神经网络的整套模型参数。Tconv的weight和bias是该整套模型参数中的一部分参数，在解析和解码过程中不再变化。此时Tconv简略表示为Tconv(c_in, c_out, s, k_ver, k_hor)(input)。

当在一系列操作中调用本条中定义的Tconv且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时Tconv简略表示为Tconv(c_in, c_out, s_vh, k_ver, k_hor)。

Tconv执行以下计算操作：

```
Tconv( c_in, c_out, s_vh, k_ver, k_hor )( input, weight, bias ) {
    for ( i = 0; i < c_in; i++ )
        for ( j = 0; j < h_in; j++ )
            for ( k = 0; k < w_in; k++ )
                temp[ i ][ j * s_vh ][ k * s_vh ] = input[ i ][ j ][ k ]
    output[ :c_out ][ :h_out ][ :w_out ] =
      Conv( c_in, c_out, 1, k_ver, k_hor )( temp[ :c_in ][ :h_out ][ :w_out ], weight, bias )
}
```

5.21 张量填充

张量填充表示为Pad(up, bottom, left, right, front, back)(input)，其中：

——up 表示在张量的宽高平面上向上方填充张量样本的数量；

——bottom 表示在张量的宽高平面上向下方填充张量样本的数量；

——left 表示在张量的宽高平面上向左侧填充张量样本的数量；

——right 表示在张量的宽高平面上向右侧填充张量样本的数量；

——front 表示在张量的通道维度上向前填充张量样本的数量；

——back 表示在张量的通道维度上向后填充张量样本的数量。

Pad的输入包括：输入张量input[:c_in][:h_in][:w_in]；

Pad的结果为：张量output[:c_out][:h_out][:w_out]，其中：

——c_out = c_in + front + back；

——h_out = h_in + up + bottom；

——w_out = w_in + left + right。

默认的填充的值为0。

当在一系列操作中调用本条中定义的Pad且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时Pad简略表示为Pad(up, bottom, left, right, front, back)。

当不指定front和back时，两者的缺省值为0，此时Pad简略表示为Pad(up, bottom, left, right)。

Pad执行如下操作：

```
Pad( up, bottom, left, right, front, back )( input ) {
    for ( i = 0; i < c_out; i++ )
        for ( j = 0; j < h_out; j++ )
            for ( k = 0; k < w_out; k++ )
                if ( i >= front & i < ( c_out – back ) & j >= left & j < ( h_out – right ) & k >= up & k < ( w_out – bottom ) )
                        output[ i ][ j ][ k ] = input[ i – front ][ j – up ][ k – left ]
                else
                        output[ i ][ j ][ k ] = 0
}
```

5.22 张量裁剪

张量裁剪表示为Crop(h_out, w_out)(input)，其中：

——h_out 表示在裁剪后的张量的高度；

——w_out 表示在裁剪后的张量的宽度。

Crop的输入包括：输入张量input[:c][:h_in][:w_in]。

Crop的结果为：张量output[:c][:h_out][:w_out]。

h_out应小于或等于h_in，w_out应小于或等于w_in。

当在一系列操作中调用本条中定义的Crop且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时Crop简略表示为Crop(h_out, w_out)。

Crop执行如下操作：

```
Crop( h_out, w_out )( input ) {
    for ( i = 0; i < c; i++ )
        for ( j = 0; j < h_out; j++ )
            for ( k = 0; k < w_out; k++ )
```

```
            output[ i ][ j ][ k ] = input[ i ][ j ][ k ]
}
```

5.23 张量取绝对值

张量取绝对值表示为ABS(input)。

ABS的输入包括：输入张量input[:c][:h][:w]。

ABS的结果为：张量output[:c][:h][:w]。

当在一系列操作中调用本条中定义的ABS且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时ABS简略表示为ABS( )。

ABS执行如下操作：

```
ABS( input ) {
    for ( i = 0; i < c; i++ )
        for ( j = 0; j < h; j++ )
            for ( k = 0; k < w; k++ )
                output[ i ][ j ][ k ] = ( input[ i ][ j ][ k ] < 0 ) ? -input[ i ][ j ][ k ] : input[ i ][ j ][ k ]
}
```

5.24 张量截断

张量截断表示为Clip(low_thr, high_thr)(input)，其中：

——low_thr 表示张量中元素值被截断的下限值；

——high_thr 表示张量中元素值被截断的上限值。

Clip的输入包括：输入张量input[:c][:h][:w]。

Clip的结果为：张量output[:c][:h][:w]。

当在一系列操作中调用本条中定义的Clip且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时Clip简略表示为Clip(low_thr, high_thr)。

Clip执行如下操作：

```
Clip( low_thr, high_thr )( input ) {
    for ( i = 0; i < c; i++ )
        for ( j = 0; j < h; j++ )
            for ( k = 0; k < w; k++ )
                output[ i ][ j ][ k ] = clip3( low_thr, high_thr, input[ i ][ j ][ k ] )
}
```

5.25 张量加和

张量加和表示为Add(input1, input2)。

Add的输入包括：

——输入张量 input1[:c][:h][:w]；

——输入张量 input2[:c][:h][:w]。

Add的结果为：张量output[:c][:h][:w]。

Add执行如下操作：

```
Add( input1,input2 ) {
    for ( i = 0; i < c; i++ )
        for ( j = 0; j < h; j++ )
            for ( k = 0; k < w; k++ )
                output[ i ][ j ][ k ] = input1[ i ][ j ][ k ] + input2[ i ][ j ][ k ]
}
```

5.26 二维加权卷积

二维加权卷积表示为MaskConv(c)(input, weight1, weight2, bias1, bias2)，其中：c表示张量的通道数。

MaskConv的输入包括：

——输入张量 input[:c][:h][:w]；

——卷积核权重 weight1[:c][:c][:3][:3]、weight2[:c][:c][:1][:1]；

——偏置 bias1[c]、bias2[c]，偏置的缺省值为 0。

MaskConv的结果为：张量output[:c][:h][:w]。

解析和解码过程加载了神经网络的整套模型参数。MaskConv的weight和bias是该整套模型参数中的一部分参数，在解析和解码过程中不再变化。此时MaskConv简略表示为MaskConv(c)(input)。

当在一系列操作中调用本条中定义的MaskConv且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时MaskConv简略表示为MaskConv(c)。

MaskConv依次执行以下步骤：

a) 使用渗漏激活函数：

```
tmp1[ :c ][ :h ][ :w ] = LeakyReLU( input[ :c ][ :h ][ :w ] )
```

b) 进行二维深度卷积：

```
tmp2[ :c ][ :h ][ :w ] = DepthConv( c, 1, 3, 3 )( tmp1[ :c ][ :h ][ :w ], weight1[ :c ][ :c ][ :3 ][ :3 ], bias1[ c ] )
```

c) 进行二维卷积：

```
tmp3[ :c ][ :h ][ :w ] = Conv( c, c, 1, 1, 1 )( tmp2[ :c ][ :h ][ :w ], weight2[ :c ][ :c ][ :1 ][ :1 ], bias2[ c ] )
```

d) 进行以下操作：

```
for ( i = 0; i < c; i++ )
    for ( j = 0; j < h; j++ )
        for ( k = 0; k < w; k++ )
            output[ i ][ j ][ k ] = input[ i ][ j ][ k ] * ( 1 + tmp3[ i ][ j ][ k ] )
```

5.27 二维残差卷积

二维残差卷积表示为ResConv(c_in, c_out, tp)(input, weight1, weight2, weight3, bias1, bias2, bias3)，其中：

——c_in 表示输入张量的通道数；

——c_out 表示输出张量的通道数；

——tp 表示 ResConv 的类型，其值为 0 或 1。

ResConv的输入包括：

——输入张量 input[:c_in][:h][:w]；

——卷积核权重 weight1[:c_in][:3][:3]、weight2[:c_out][:c_in][:1][:1]、
weight3[:c_out][:c_in][:1][:1]；

——偏置 bias1[c_in]、bias2[c_out]、bias3[c_out]，偏置的缺省值为 0。

ResConv的结果为：张量output[:c_out][:h][:w]。

解析和解码过程加载了神经网络的整套模型参数。ResConv的weight1、weight2和weight3以及bias1、bias2和bias3是该整套模型参数中的一部分参数，在解析和解码过程中不再变化。此时ResConv简略表示为ResConv(c_in, c_out, s, k_ver, k_hor)(input)。

当在一系列操作中调用本条中定义的ResConv且未明确设置输入张量的情况下，默认将输入张量设置为该系列操作中当前操作的前一个操作的输出张量，此时ResConv简略表示为ResConv(c_in, c_out, s, k_ver, k_hor)。

ResConv依次执行以下步骤：

a) 如果 tp 的值为 1，使用渗漏激活函数：

```
tmp0[ :c_in ][ :h ][ :w ] = LeakyReLU( input[ :c_in ][ :h ][ :w ] )
```

b) 如果 tp 的值为 0，令 tmp0=input；

c) 进行二维深度卷积：

```
tmp1[ :c_in ][ :h ][ :w ] =
DepthConv( c_in, 1, 3, 3 )( tmp0[ :c_in ][ :h ][ :w ], weight1[ :c_in ][ :3 ][ :3 ], bias1[ c_in ] )
```

d) 进行二维卷积：

```
tmp2[ :c_out ][ :h ][ :w ] =
Conv( c_in, c_out, 1, 1, 1 )( tmp1[ :c_in ][ :h ][ :w ], weight2[ :c_out ][ :c_in ][ :1 ][ :1 ], bias2[ c_out ] )
```

e) 如果 tp 的值为 1，令 tmp3=tmp2；

f) 如果 tp 的值为 0，使用渗漏激活函数：

```
tmp3[ :c_out ][ :h ][ :w ] = LeakyReLU( tmp2[ :c_out ][ :h ][ :w ] )
```

g) 进行以下操作：

```
if( c_in == c_out )
    tmp4[ :c_out ][ :h ][ :w ] = input[ :c_in ][ :h ][ :w ]
else
    tmp4[ :c_out ][ :h ][ :w ] =
    Conv( c_in, c_out, 1, 1, 1 )( input[ :c_in ][ :h ][ :w ], weight3[ :c_out ][ :c_in ][ :1 ][ :1 ], bias3[ c_out ] )
```

h) 进行张量加和：

```
output[ :c_out ][ :h ][ :w ] = Add( tmp4[ :c_out ][ :h ][ :w ], tmp3[ :c_out ][ :h ][ :w ] )
```

6 位流的语法和语义

6.1 语法描述

6.1.1 起始码

起始码是一组特定的位串。在符合本文件的位流中，除起始码外的任何情况下都不应出现这些位串。

起始码由起始码前缀和起始码值构成。起始码前缀是位串‘0000 0000 0000 0000 0000 0001’。所有的起始码都应字节对齐。

起始码值是一个8位整数，用来表示起始码的类型（见表9）。

表 9 起始码值

起始码类型	起始码值（十六进制）
图像头起始码（icm_header_start_code）	80
保留	其他

部分语法元素取特定值时可得到与起始码前缀相同的位串，称为伪起始码。符合本文件的编码器和解码器应按照附录A定义的方法处理伪起始码问题。

6.1.2 图像位流定义

图像位流定义见表10。

表 10 图像位流定义

图像位流定义	描述符
icm_bitstream( ) {	
image_header()	
if (image_structure_enabled_flag)	
image_structure_data()	
image_feature_data()	
if (image_rec_enabled_flag)	
image_rec_data()	
}	

6.1.3 图像头定义

图像头定义见表11。

表 11 图像头定义

图像头定义	描述符
image_header( ) {	
icm_header_start_code	f(32)
profile_id	u(4)
z_width_minus1	u(8)
z_height_minus1	u(8)
marker_bit	
feature_type_id	u(8)
image_structure_enabled_flag	u(1)
image_rec_enabled_flag	u(1)
marker_bit	

表 11 图像头定义（续）

if (image_structure_enabled_flag) {	

图像头定义	描述符
image_height_minus1	u(16)
marker_bit	
image_width_minus1	u(16)
}	
imh_extension_flag	u(1)
if(imh_extension_flag) {	
imh_extension_length	u(15)
for(i = 0; i < imh_extension_length; i++)	
imh_extension_data_byte[i]	u(8)
}	
while (! byte_aligned( ))	
stuffing_bit	’0’
}	

6.1.4 图像特征数据定义

图像特征数据定义见表12。

表 12 图像特征数据定义

图像编码数据定义	描述符
image_feature_data( ) {	
rate_control_q_id	u(5)
for (i = 0; i < C; i++)	
for (j = 0; j < zH; j++)	
for (k = 0; k < zW; k++)	
z[i][j][k]	ne(v)
yH = zH * zScaleFactor	
yW = zW * zScaleFactor	
for (i = 0; i < C; i++)	
for (j = 0; j < yH; j++)	
for (k = 0; k < yW; k++)	
y_residue[i][j][k]	ne(v)
ifd_extension_flag	u(1)
if (ifd_extension_flag) {	
ifd_extension_length	u(15)
for(i = 0; i < ifd_extension_length; i++)	
ifd_extension_data_byte[i]	u(8)
}	
while (! byte_aligned( ))	

表 12　图像特征数据定义（续）

图像编码数据定义	描述符
stuffing_bit	’0’
}	

6.1.5　图像结构数据定义

图像结构数据定义见表13。

表 13　图像结构数据定义

图像结构数据定义	描述符
image_structure_data( ) {	
group_mask_block_size_id	u(4)
for (i = 0; i < iH / bS; i++)	
for (j = 0; j < iW / bS; j++)	
group_mask_value[i][j]	le(v)
bounding_boxes_enabled_flag	u(1)
if (bounding_boxes_enabled_flag) {	
bounding_boxes_num_minus1	u(16)
for (i = 0; i <= BoundingBoxNum; i++) {	
bounding_box_x[i]	u(14)
bounding_box_y[i]	u(14)
bounding_box_h[i]	u(14)
bounding_box_w[i]	u(14)
bounding_box_category_id[i]	u(8)
}	
}	
isd_extension_flag	u(1)
if(isd_extension_flag) {	
isd_extension_length	u(32)
for(i = 0; i < isd_extension_length; i++)	
isd_extension_data_byte[i]	u(8)
}	
while (! byte_aligned( ))	
stuffing_bit	’0’
}	

6.1.6　图像重建数据定义

图像重建数据定义见表14。

表 14　图像重建数据定义

图像重建数据定义	描述符
image_rec_data( ) {	
crop_left_size	u(6)
crop_right_size	u(6)
crop_upper_size	u(6)
crop_bottom_size	u(6)
rec_image_format_id	u(4)
bit_depth_id	u(1)
}	

6.2　语义描述

6.2.1　图像头语义描述

图像头起始码　icm_header_start_code

位串‘0x00000180’。标识图像头的开始。

档次　profile_id

4位无符号整数。表示位流符合的档次。档次应符合附录B的规定。

超先验张量宽度　z_width_minus1

8位无符号整数。表示超先验张量的宽度zW，其值等于z_width_minus1的值加1。

超先验张量高度　z_height_minus1

8位无符号整数。表示超先验张量的高度zH，其值等于z_height_minus1的值加1。

图像特征类型标号　feature_type_id

8位无符号整数。表示能够使用解码的图像特征的任务网络类型，该类型与标号之间的关系见表15。

表 15　任务网络类型与 feature_type_id 之间的关系

feature_type_id的值	任务网络类型
0	Faster R-CNN X101-FPN（目标检测）
1	Mask R-CNN X101-FPN（实例分割）
2	Keypoint R-CNN X101-FPN（姿态估计）
3~256	保留

图像结构数据允许标志　image_structure_enabled_flag

二值变量。值为‘1’表示允许使用图像结构化数据，值为‘0’表示不允许使用图像结构化数据。

图像重建数据允许标志　image_rec_enabled_flag

二值变量。值为‘1’表示允许使用图像重建数据，值为‘0’表示不允许使用图像重建数据。

图像宽度　image_width_minus1

16位无符号整数。表示图像的宽度iW，其值等于image_width_minus1的值加1。

图像高度　image_height_minus1

16位无符号整数。表示图像的高度iH，其值等于image_height_minus1的值加1。

图像头扩展标志　imh_extension_flag

二值变量。值为‘0’表示图像头中不包含扩展数据，其值为‘1’表示图像头中包含扩展数据。

图像头扩展数据长度　imh_extension_length

15位无符号整数。表示图像头中扩展数据的长度，以字节为单位。如果位流中不存在imh_extension_length，则imh_extension_length的值等于0。虽然imh_extension_length在符合本文件的位流中不存在，但imh_extension_length的使用可以在本文件的某些未来版本中指定，并且符合本文件的解码器也应允许imh_extension_length存在。

图像头扩展数据字节　imh_extension_data_byte[i]

8位无符号整数。imh_extension_data_byte[i]可以是任何值，它的存在和值不影响本文件中指定的解码过程。符合本文件的解码器应忽略imh_extension_data_byte[i]的值。

6.2.2　图像特征数据语义描述

图像特征数据起始码　icm_feature_start_code

位串‘0x00000181’。标识图像特征数据的开始。

码率控制因子标号　rate_control_factor_id

5位无符号整数。表示解码应使用的码率控制因子qRC，码率控制因子标号和码率控制因子的对应关系见表16。

表16　码率控制因子

码率控制因子索引	码率控制因子qRC	码率控制因子索引	码率控制因子qRC
0	0.200	16	0.546
1	0.222	17	0.567
2	0.243	18	0.589
3	0.265	19	0.611
4	0.286	20	0.632
5	0.308	21	0.654
6	0.330	22	0.675
7	0.351	23	0.697
8	0.373	24	0.719
9	0.395	25	0.740
10	0.416	26	0.762
11	0.438	27	0.784
12	0.459	28	0.805
13	0.481	29	0.827
14	0.503	30	0.848
15	0.524	31	0.870

超先验张量元素　z[i][j][k]

表示超先验张量中第i个通道第j行第k列的元素，解析过程由ne(v)定义。

特征残差张量元素值　y_residue[i][j][k]

表示特征残差张量中第i个通道第j行第k列的元素，解析过程由ne(v)定义。

图像特征数据扩展标志　ifd_extension_flag

二值变量。值为‘0’表示图像特征数据中不包含扩展数据，其值为‘1’表示图像特征数据中包含扩展数据。

图像特征数据扩展数据长度　ifd_extension_length

16位无符号整数。表示图像特征数据中扩展数据的长度，以字节为单位。如果位流中不存在ifd_extension_length，则ifd_extension_length的值等于0。虽然ifd_extension_length在符合本文件的位流中不存在，但ifd_extension_length的使用可以在本文件的某些未来版本中指定，并且符合本文件的解码器也应允许ifd_extension_length存在。

图像特征数据扩展数据字节　ifd_extension_data_byte[i]

8位无符号整数。ifd_extension_data_byte[i]可以是任何值，它的存在和值不影响本文件中指定的解码过程。符合本文件的解码器应忽略ifd_extension_data_byte[i]的值。

6.2.3　图像结构数据语义描述

图像结构数据起始码　icm_structure_start_code

位串‘0x00000182’。标识图像结构数据的开始。

块掩膜覆盖区域尺寸大小标号　group_mask_block_size_id

4位无符号整数。规定块掩膜中一个掩膜点在图像中覆盖的方形区域的尺寸大小bS（见表17）。语义块掩膜中所有掩膜点在图像中覆盖的方形区域的尺寸大小应一致。

表 17　块掩膜中掩膜点对应图像中的区域尺寸大小

group_mask_block_size_id的值	bS的值
0	16
1	32
2~15	保留

块掩膜值　group_mask_value[i][j]

表示块掩膜中第i行第j列的值，解析过程由le(v)定义。

目标框允许标志　bounding_boxes_enabled_flag

二值变量。值为‘1’表示允许使用目标框，值为‘0’表示不允许使用目标框。

目标框数　bounding_boxes_num_minus1

16位无符号整数。表示目标框数目BoundingBoxNum，其值为bounding_boxes_num_minus1的值加1。

目标框起始点横坐标　bounding_box_x[i]

14位无符号整数。表示第i个目标框的起始点在图像中的横坐标值。

目标框起始点纵坐标　bounding_box_y[i]

14位无符号整数。表示第i个目标框的起始点在图像中的纵坐标值。

目标框宽度　bounding_box_w[i]

14位无符号整数。表示第i个目标框的宽度。

目标框高度　bounding_box_h[i]

14位无符号整数。表示第i个目标框的高度。

目标框类别标号　bounding_box_category_id[i]

8位无符号整数。规定第i个目标框的类别标号（见表18）。

表 18　语义信息目标框类别

bounding_box_category_id[i]的值	类别
0	背景
1	人
2	车
3~255	保留

图像结构数据扩展标志　isd_extension_flag

二值变量。值为‘0’表示图像结构数据中不包含扩展数据，其值为‘1’表示图像结构数据中包含扩展数据。

图像结构数据扩展数据长度　isd_extension_length

16位无符号整数。表示图像结构数据中扩展数据的长度，以字节为单位。如果位流中不存在isd_extension_length，则isd_extension_length的值等于0。虽然isd_extension_length在符合本文件的位流中不存在，但isd_extension_length的使用可以在本文件的某些未来版本中指定，并且符合本文件的解码器也应允许isd_extension_length存在。

图像结构数据扩展数据字节　isd_extension_data_byte[i]

8位无符号整数。isd_extension_data_byte[i]可以是任何值，它的存在和值不影响本文件中指定的解码过程。符合本文件的解码器应忽略isd_extension_data_byte[i]的值。

6.2.4　图像重建数据语义描述

图像重建数据起始码　icm_rec_start_code

位串‘0x00000184’。标识图像重建数据的开始。

左边界裁剪大小　crop_left_size

6位无符号整数。表示重建的图像的左边界应裁剪掉的像素的列数。

右边界裁剪大小　crop_right_size

6位无符号整数。表示重建的图像的右边界应裁剪掉的像素的列数。

上边界裁剪大小　crop_upper_size

6位无符号整数。表示重建的图像的上边界应裁剪掉的像素的行数。

下边界裁剪大小　crop_bottom_size

6位无符号整数。表示重建的图像的下边界应裁剪掉的像素的行数。

重建图像数据格式标号　rec_image_format_id

4位无符号整数。其值和输出图像数据格式之间的对应关系见表19。

表 19　输出图像数据格式

rec_image_format_id的值	数据格式
0	YUV420
1	YUV422
2	YUV444
3	sRGB
4-15	保留

重建图像位宽标号　bit_depth_id

二值变量。值为‘0’表示输出图像的位宽等于8，其值为‘1’表示输出图像的位宽等于10。

7　解析过程

7.1　通则

该过程的输入是码流的当前指针位置。

该过程的输出是语法元素的值，这些语法元素使用的描述符应是u(n)或f(n)或le(v)或ne(v)中的一种。

该过程进行以下操作之一：

——如果语法元素的描述符是 u(n)，应使用 7.2 的操作；

——如果语法元素的描述符是 f(n)，应使用 7.3 的操作；

——如果语法元素的描述符是 ne(v)，应使用 7.4 的操作；

——如果语法元素的描述符是 le(v)，应使用 7.5 的操作。

7.2　无符号整数解析

无符号整数的解析值为函数read_bits(n)（见5.9.2.4）的返回值，该返回值用高位在前的二进制表示。

7.3　固定位宽符号解析

固定位宽符号的解析值为函数read_bits(n)（见5.9.2.4）的返回值，该函数用于取特定值的连续n个二进制位。

7.4　ne(v)符号解析

7.4.1　通则

该操作的过程为：

a)　按照附录 C 约定的方式在神经网络中加载模型参数；

b)　如果当前语法元素是 z，进行 7.4.2 的初始化操作，然后执行 7.4.3 的操作；

c)　如果当前语法元素是 y_residue, 进行 7.4.2 的初始化操作，然后执行 7.4.4 的操作。

7.4.2　初始化

该过程包括以下操作：

a)　按照如下计算设定 MaskCdf、MaskBypass 和 StateLowerBound 的值：

```
MaskCdf = 0x0000000000000001 << CdfPrecision – 1
MaskBypass = 0x00000001 << BypassPrecision – 1
StateLowerBound = 0x0000000000000001 << (StatePrecision – 1)
```

b)　按照如下计算设置 State 的初始值：

```
StateLSB = read_bits(StatePrecision)
StateMSB = read_bits(StatePrecision)
State = StateLSB | ( StateMSB << StatePrecision )
```

c)　如果当前语法元素是 z，

1)　将 CDFLength[:zN]按照 D.1 的要求设置；

2)　将 CDFS[:zN][:]按照 D.1 的要求设置；

3） 将 MaxValues[:zN]按照 D.1 的要求设置；
4） 将 Offsets[:zN]按照 D.1 的要求设置；
5） 将 Indexs[:C][:zH][:zW]按照 D.1 的要求设置。

d） 如果当前语法元素是 y_residue，
1） 将 CDFLength[:yN]按照 D.2 的要求设置；
2） 将 CDFS[:yN][:]按照 D.2 的要求设置；
3） 将 MaxValues[:yN]按照 D.2 的要求设置；
4） 将 Offsets[:yN]按照 D.2 的要求设置；
5） 将 ScaleTable[:yN]按照 D.2 的要求设置；
6） 进行 7.4.5 的概率估计操作，得到 Indexs[:C][:yH][:yW]。

7.4.3 z 的解析

该解析操作的输入为：码流的当前指针位置。
该解析操作的输出为：z[:C][:zH][:zW]。
该过程包括以下操作：

a） 令 i=0，j=0，k=0，循环执行以下操作：
1） 将 dN 设置为 CDFLength[Indexs[i][j][k]]；
2） 将 CDF[:dN]设置为 CDFS[Indexs[i][j][k]][:dN]；
3） 将 MaxValue 设置为 MaxValues[Indexs[i][j][k]]；
4） 将 Offset 设置为 Offsets[Indexs[i][j][k]]；
5） 执行 7.4.6 的操作，得到 z[i][j][k]的值；
6） 令 k = k + 1；
7） 如果 k 的值等于 zW，令 j = j + 1，k = 0；
8） 如果 j 的值等于 zH，令 i = i + 1，j = 0；
9） 如果 i 的值等于 C，结束循环。

7.4.4 y_residue 的解析

该解析操作的输入为：码流的当前指针位置。
该解析操作的输出为：y_residue[:C][:yH][:yW]。
该过程包括以下操作：

a） 令 i=0，j=0，k=0，循环执行以下操作：
1） 将 dN 设置为 CDFLength[Indexs[i][j][k]]；
2） 将 CDF[:dN]设置为 CDFS[Indexs[i][j][k]][:dN]；
3） 将 MaxValue 设置为 MaxValues[Indexs[i][j][k]]；
4） 将 Offset 设置为 Offsets[Indexs[i][j][k]]；
5） 执行 7.4.6 的操作，得到 y_residue[i][j][k]的值；
6） 令 k = k + 1；
7） 如果 k 的值等于 yW，令 j = j + 1，k = 0；
8） 如果 j 的值等于 yH，令 i = i + 1，j = 0；
9） 如果 i 的值等于 C，结束循环。

7.4.5 概率估计

该过程的输入为：z[:C][:zH][:zW]。

该过程的输出为：Indexs[:C][:yH][:yW]。

该过程中使用的概率估计网络见图 1，依次执行以下步骤：

a） 使用二维整数卷积 IntConv(C, C, 1, 1, 1)(z[:C][:zH][:zW])；

b） 使用标准激活函数 ReLU( )；

c） 使用二维整数卷积 IntConv(C, C, 1, 3, 3)；

d） 使用标准激活函数 ReLU( )；

e） 使用二维整数卷积 IntConv(C, C * 16, 1, 1, 1)；

f） 使用超分重组 Shuffle(4)；

g） 使用 Crop(yH, yW)；

h） 使用 ABS( )；

i） 使用 Clip(0, 2^yP – 1)，得到 Scale[:C][:yH][:yW]；

j） 进行如下计算，得到 Indexs[:C][:yH][:yW]：

```
for ( i = 0; i < C; i++ ) {
    for ( j = 0; j < yH; j++ ) {
        for ( k = 0; k < yW; k++ ) {
            Indexs[ i ][ j ][ k ] = yN - 1
            Scale[ i ][ j ][ k ] = ( Scale[ i ][ j ][ k ] < ScaleLowBound ) ? ScaleLowBound : Scale[ i ][ j ][ k ]
            for ( x = 0; x < yN; x++ ) {
                if ( Scale[ i ][ j ][ k ] < ScaleTable[ x ] )
                    Indexs[ i ][ j ][ k ]--
            }
        }
    }
}
```

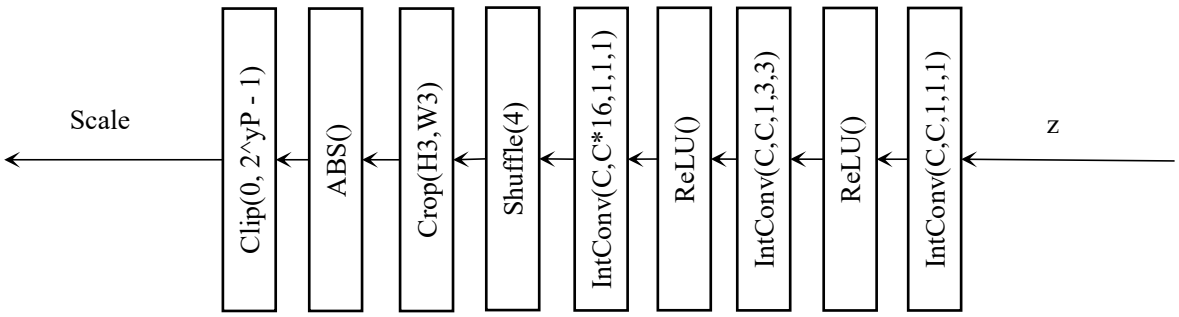

图 1　概率估计网络

7.4.6　数值熵解析

7.4.6.1　通则

该过程的输入包含：

——码流的当前指针位置；

——State。

该过程的输出为：Value。

该过程的操作包括：

a） 按照如下操作得到 ParseVal：

```
CDFCur = State & MaskCdf
ParseVal = 0
while ( CDF[ ParseVal ] <= CDFCur ) {
    ParseVal++
}
```

b） 按照如下操作更新 State：

```
CdfStart = CDF[ ParseVal ]
Freq = CDF[ ParseVal + 1 ] – CdfStart
State = Freq * ( State >> CdfPrecision) + ( State & MaskCdf ) – CdfStart
```

c） 进行 7.4.6.2 的解析状态值检验，将 State 设置为检验后的 NewState；

d） 如果 ParseVal 的值和 MaxValue 的值相等，则进行 7.4.6.3 的熵解析跳过，将解析值 ParseVal 的值设置为熵解析跳过得到的 RawVal；

e） 将 Value 的值设置为 ParseVal 与 Offset 的加和。

7.4.6.2 解析状态值检验

该过程的输入包含：

——码流的当前指针位置；

——State。

该过程的输出为：NewState。

该过程的操作包括：

a） 如果 State 的值小于 StateLowerBound，执行以下计算：

```
New = read_bits(StatePrecision)
NewState= (State << StatePrecision) | New
```

7.4.6.3 熵解析跳过

该过程的输入包含：

——码流的当前指针位置；

——State。

该过程的输出为：RawVal。

该过程的操作包括：

a） 将 BypassLength 的值初始化为 0，循环进行以下操作：

1） 进行 7.4.6.4 的数据读取，得到 BypassVal 和 State；

2） 将 BypassLength 的值设置为 BypassLength 和 BypassVal 的加和值；

3） 如果 BypassVal 的值不等于 MaskBypass，结束循环。

b） 将 RawVal 和 j 的值均初始化为 0，循环进行 BypassLength 次以下操作：

1） 进行 7.4.6.4 的数据读取，得到 BypassVal 和 State；

2） 执行如下计算：

```
RawVal = ( BypassVal << ( j * BypassPrecision )) | RawVal
```

c） 执行如下计算：

```
if ( RawVal & 1 )
    RawVal = –( RawVal >> 1 ) – 1
else
    RawVal = ( RawVal >> 1 ) + MaxValue
```

7.4.6.4 跳过熵解析的数据读取

该过程的输入包含：

——码流的当前指针位置；

——State。

——该过程的输出为：

——BypassVal；

——State。

该过程的操作包括：

a) 执行以下计算：

```
BypassVal = State & MaskBypass
state = state >> bypass_precision
```

b) 进行 7.4.6.2 的解析状态值检验，将 State 设置为检验后的 NewState。

7.5 符号查表解析

7.5.1 group_mask_value 的解析

该解析操作的输入为：码流的当前指针位置。

该解析操作的输出为：分组掩膜group_mask_value[:(iH / bS)][:(iW / bS)]。

该过程包括以下操作：

a) 令 i=0~(iH / bS)，每次执行以下操作：

1) 令 j=0~(iW / bS)，每次按照附录 D.3 中表 D.3 规定的块掩膜码表进行解析得到 group_mask_value[i][j]的值。

8 解码过程

8.1 通则

解码过程的输入是图像位流，输出包括已解码的图像结构数据、已解码的图像特征数据和已解码的图像重建数据中的一种或多种。

解码过程进行以下操作：

a) 在神经网络中加载附录 C 中规定的神经网络模型参数；

b) 解析图像位流（见 8.2）；

c) 如果 image_structure_enabled_flag 的值为 1，解码图像结构数据（见 8.4），输出分组掩膜和目标框；

d) 解码图像特征数据（见 8.3）；

e) 如果 image_rec_enabled_flag 的值为 1，解码图像重建数据（见 8.5），输出重建图像；否则，输出重建张量。

注：图像特征数据可以通过特征适配以用于本文件在表15中约定的任务网络以外的其他任务网络，见附录E。

8.2 解析图像位流

该过程进行以下操作：

a) 按照第 6 章规定的位流中语法元素的顺序，使用第 7 章规定的解析过程逐个解析图像位流中的语法元素并获得语法元素的解析值；

b) 按照第 6 章中的规定，根据这些语法元素的解析值推导得到的本文件中的全局变量的值。

8.3 解码图像特征数据

8.3.1 通则

解码过程依次执行以下操作：

a) 进行特征张量解码（见 8.3.2），得到特征张量 y[:C][:yH][:yW]；

b) 进行特征张量超分（见 8.3.3），得到重建张量 r[:C][:rH][:rW]。

8.3.2 特征张量解码

8.3.2.1 通则

该过程的输入为：

——y_residue[:C][:yH][:yW]；

——z[:C][:zH][:zW]。

该过程的输出为：y[:C][:yH][:yW]。

该过程依次执行以下操作：

a) 进行反量化（见 8.3.2.2）；

b) 进行超先验张量超分（见 8.3.2.3）；

c) 进行特征张量预测补偿（见 8.3.2.4）；

d) 进行特征张量调制（见 8.3.2.5）。

8.3.2.2 反量化

该过程的输入为：y_residue[:C][:yH][:yW]。

该过程的输出为：yResDQ[:C][:yH][:yW]。

该过程执行以下计算：

```
for ( i = 0; i < C; i++ )
   for (j = 0; j < yH; j++)
      for (k = 0; k < yW; k++)
         yResDQ[ i ][ j ][ k ] = float( y_residue[ i ][ j ][ k ] )
```

8.3.2.3 超先验张量超分

该过程的输入为：z[:C][:zH][:zW]。

该过程的输出为：yHyper[:C * 2][:yH][:yW]。

该过程中使用的超分网络见图 2，依次执行以下步骤：

a) 使用二维卷积 Conv(C, C, 1, 1, 1)；

b) 使用二维转置卷积 Tconv(C, C, 2, 4, 4)；

c) 使用张量裁剪 Crop(zH * 2, zW * 2)；

d） 使用渗漏激活函数 LeakyReLU( )；
e） 使用二维卷积 Conv(C, C, 1, 3, 3)；
f） 使用二维转置卷积 Tconv(C, C, 2, 4, 4)；
g） 使用张量裁剪 Crop(yH, yW)；
h） 使用渗漏激活函数 LeakyReLU( )；
i） 使用二维卷积 Conv(C, C * 2, 1, 3, 3)；
j） 使用渗漏激活函数 LeakyReLU( )，得到 yHyper[:C * 2][:yH][:yW]。

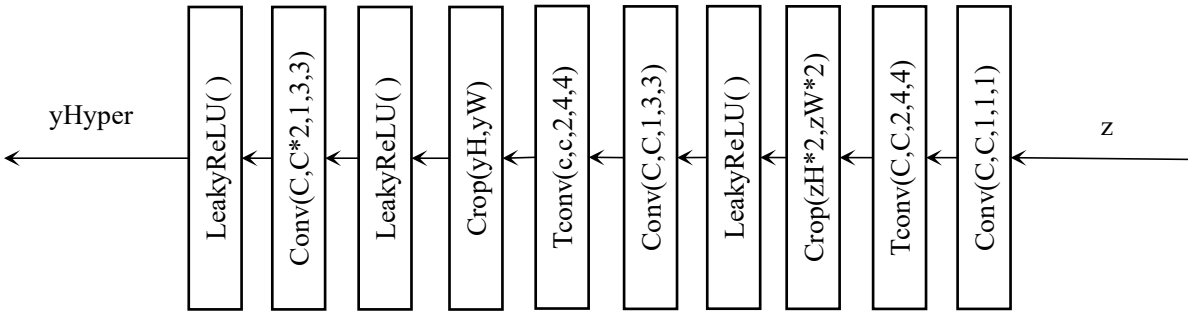

图 2　超先验张量超分网络

8.3.2.4　特征张量预测补偿

8.3.2.4.1　通则

该过程的输入为：
——yHyper[:C * 2][:yH][:yW]；
——yResDQ[:C][:yH][:yW]。
输出为：yRec[:C][:yH][:yW]。
该过程依次执行以下操作：
a） 拆分 yHyper[:C * 2][:yH][:yW]：
1） 进行张量填充：

```
yHP = yH + yH % 2
yWP = yW + yW % 2
yHyperPad[ :C * 2 ][ :yHP ][ :yWP ] = Pad( 0, yH % 2, 0, yW % 2)( yHyper[ :C * 2 ][ :yH ][ :yW ] )
```

2） 进行交叉降分重组：

```
yHyperPadDown[ :C * 8 ][ :yHP / 2 ][ :yWP / 2 ] = CrossDownShuffle(yHyperPad[ :C * 2 ][ :yHP ][ :yWP ])
```

3） 进行张量拆分：

```
for ( i = 0; i < 4; i++ )
    yHyperPadDownPart[ i ][ :C * 2 ][ :yHP / 2 ][ :yWP / 2 ] =
      yHyperPadDown[ C * 2 * i:C * 2 * ( i+1 ) ][ :yHP / 2 ][ :yWP / 2 ]
```

b） 拆分 yResDQ[:C][:yH][:yW]：
1） 进行张量填充：

```
yResDQPad[ :C ][ :yHP ][ :yWP ] = Pad( 0, yH % 2, 0, yW % 2)( yResDQ[ :C ][ :yH ][ :yW ] )
```

2） 进行交叉降分重组：

```
yResDQPadDown[ :C * 4 ][ :yHP / 2 ][ :yWP / 2 ] = CrossDownShuffle(yResDQPad[ :C ][ :yHP ][ :yWP ])
```

3） 进行张量拆分，

```
for ( i = 0; i < 8; i++ )
    yResDQPadDownPart[ i ][ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] =
      yResDQPadDown[ C / 2 * i:C / 2 * ( i+1 ) ][ :yHP / 2 ][ :yWP / 2 ]
```

c） 进行第 0 阶特征张量预测（见 8.3.2.4.2）；

d） 进行第 1 阶特征张量预测（见 8.3.2.4.3）；

e） 进行第 2 阶特征张量预测（见 8.3.2.4.4）；

f） 进行第 3 阶特征张量预测（见 8.3.2.4.5）；

g） 进行预测张量调整（见 8.3.2.4.10）；

h） 进行第 4 阶特征张量预测（见 8.3.2.4.6）；

i） 进行第 5 阶特征张量预测（见 8.3.2.4.7）；

j） 进行第 6 阶特征张量预测（见 8.3.2.4.8）；

k） 进行第 7 阶特征张量预测（见 8.3.2.4.9）；

l） 进行交叉超分重组：

```
yRec[ :C ][ :yH ][ :yW ] = CrossUpShuffle( yTmp[ :C * 4 ][ :yHP / 2 ][ :yWP / 2 ])
```

8.3.2.4.2 第 0 阶特征张量预测

该过程的输入为：

——yHyperPadDownPart[0][:C * 2][:yHP / 2][:yWP / 2]；

——yResDQPadDownPart[0][:C / 2][:yHP / 2][:yWP / 2]。

该过程的输出为：yTmp[:C / 2][:yHP / 2][:yWP / 2]。

该过程依次执行以下操作：

a） 进行张量填充：

```
yHyperPadDownPartCPad[ 0 ][ :C * 3 ][ :yHP / 2 ][ :yWP / 2 ] =
  Pad( 0, 0, 0, 0, 0, C )( yHyperPadDownPart[ 0 ][ :C * 2 ][ :yHP / 2 ][ :yWP / 2 ] )
```

b） 将 i 设置为 0，进行预测融合（见 8.3.2.4.11）；

c） 进行预测补偿：

```
yTmp[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ]=
  yResDQPadDownPart[ 0 ][ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] + yPredTmp[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ]
```

8.3.2.4.3 第 1 阶特征张量预测

该过程的输入为：

——yHyperPadDownPart[1][:C * 2][:yHP / 2][:yWP / 2]；

——yResDQPadDownPart[1][:C / 2][:yHP / 2][:yWP / 2]；

——yTmp[:C / 2][:yHP / 2][:yWP / 2]。

该过程的输出为：yTmp[C / 2:C][:yHP / 2][:yWP / 2]。

该过程依次执行以下操作：

a） 进行二维卷积：

```
yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] =
Conv( C / 2, C / 2, 1, 3, 3 )( yTmp[ :C / 2 ][ :yHP / 2 ][ :yWP / 2] )
```

b） 进行张量填充：

```
yHyperPadDownPartCPad[ 1 ][ :C * 5 / 2 ][ :yHP / 2 ][ :yWP / 2 ] =
```

```
Pad( 0, 0, 0, 0, 0, C / 2 )( yHyperPadDownPart[ 1 ][ :C * 2 ][ :yHP / 2 ][ :yWP / 2 ] )
```

c） 进行张量拼接：

```
yHyperPadDownPartCPad[ 1 ][ :C * 3 ][ :yHP / 2 ][ :yWP / 2 ] =
Concat( yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ],
yHyperPadDownPartCPad[ 1 ][ :C * 5 / 2 ][ :yHP / 2 ][ :yWP / 2 ] )
```

d） 将 i 设置为 1，进行预测融合（见 8.3.2.4.11）；

e） 进行预测补偿：

```
yTmp[ C / 2:C ][ :yHP / 2 ][ :yWP / 2 ] =
yResDQPadDownPart[ 1 ][ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] + yPredTmp[ C / 2:C ][ :yHP / 2 ][ :yWP / 2 ]
```

8.3.2.4.4 第 2 阶特征张量预测

该过程的输入为：

——yHyperPadDownPart[2][:C * 2][:yHP / 2][:yWP / 2]；

——yResDQPadDownPart[2][:C / 2][:yHP / 2][:yWP / 2]；

——yTmp[:C][:yHP / 2][:yWP / 2]。

该过程的输出为：yTmp[C:C * 3 / 2][:yHP / 2][:yWP / 2]。

该过程依次执行以下操作：

a） 进行二维卷积：

```
yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] = Conv( C, C / 2, 1, 3, 3 )( yTmp[ :C ][ :yHP / 2 ][ :yWP / 2] )
```

b） 进行张量填充：

```
yHyperPadDownPartCPad[ 2 ][ :C * 5 / 2 ][ :yHP / 2 ][ :yWP / 2 ] =
  Pad( 0, 0, 0, 0, 0, C/2 )( yHyperPadDownPart[ 2 ][ :C * 2 ][ :yHP / 2 ][ :yWP / 2 ] )
```

c） 进行张量拼接：

```
yHyperPadDownPartCPad[ 2 ][ :C * 3 ][ :yHP / 2 ][ :yWP / 2 ] =
Concat( yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ],
yHyperPadDownPartCPad[ 2 ][ :C * 5 / 2 ][ :yHP / 2 ][ :yWP / 2 ] )
```

d） 将 i 设置为 2，进行预测融合（见 8.3.2.4.11）；

e） 进行预测补偿：

```
yTmp[ C:C * 3 / 2 ][ :yHP / 2 ][ :yWP / 2 ] =
yResDQPadDownPart[ 2 ][ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] + yPredTmp[ C:C * 3 / 2 ][ :yHP / 2 ][ :yWP / 2 ]
```

8.3.2.4.5 第 3 阶特征张量预测

该过程的输入为：

——yHyperPadDownPart[3][:C * 2][:yHP / 2][:yWP / 2]；

——yResDQPadDownPart[3][:C / 2][:yHP / 2][:yWP / 2]；

——yTmp[:C * 3 / 2][:yHP / 2][:yWP / 2]。

该过程的输出为：yTmp[C * 3 / 2:C * 2][:yHP / 2][:yWP / 2]。

该过程依次执行以下操作：

a） 进行二维卷积：

```
yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] =
Conv( C * 3 / 2, C / 2, 1, 3, 3 )( yTmp[ :C * 3 / 2 ][ :yHP / 2 ][ :yWP / 2 ] )
```

b） 进行张量填充：

```
yHyperPadDownPartCPad[ 3 ][ :C * 5 / 2 ][ :yHP / 2 ][ :yWP / 2 ] =
  Pad( 0, 0, 0, 0, 0, C / 2 )( yHyperPadDownPart[ 3 ][ :C * 2 ][ :yHP / 2 ][ :yWP / 2 ] )
```

c） 进行张量拼接：

```
yHyperPadDownPartCPad[ 3 ][ :C * 3 ][ :yHP / 2 ][ :yWP / 2 ] =
Concat( yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ],
yHyperPadDownPartCPad[ 3 ][ :C * 5 / 2 ][ :yHP / 2 ][ :yWP / 2 ] )
```

d） 将 i 设置为 3，进行预测融合（见 8.3.2.4.11）；

e） 进行预测补偿：

```
yTmp[ C * 3 / 2:C * 2 ][ :yHP / 2 ][ :yWP / 2 ] =
yResDQPadDownPart[ 3 ][ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] +
yPredTmp[ C * 3 / 2:C * 2 ][ :yHP / 2 ][ :yWP / 2 ]
```

8.3.2.4.6 第 4 阶特征张量预测

该过程的输入为：

——yHyperPadDownPart[0][:C * 2][:yHP / 2][:yWP / 2]；

——yResDQPadDownPart[4][:C / 2][:yHP / 2][:yWP / 2]；

——yTmpModified[:C / 2][:yHP / 2][:yWP / 2]。

该过程的输出为：yTmp[C * 2:C * 5 / 2][:yHP / 2][:yWP / 2]。

该过程依次执行以下操作：

a） 进行张量填充：

```
yHyperPadDownPartCPad[ 0 ][ :C * 5 / 2 ][ :yHP / 2 ][ :yWP / 2 ] =
  Pad( 0, 0, 0, 0, 0, C/2 )( yHyperPadDownPart[ 0 ][ :C * 2 ][ :yHP / 2 ][ :yWP / 2 ] )
```

b） 进行张量拼接：

```
yHyperPadDownPartCPad[ 4 ][ :C * 3 ][ :yHP / 2 ][ :yWP / 2 ] =
Concat( yTmpModified[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ],
yHyperPadDownPartCPad[ 0 ][ :C * 5 / 2 ][ :yHP / 2 ][ :yWP / 2 ] )
```

c） 将 i 设置为 4，进行预测融合（见 8.3.2.4.11）；

d） 进行预测补偿：

```
yTmp[ C * 2:C * 5 / 2 ][ :yHP / 2 ][ :yWP / 2 ]=
  yResDQPadDownPart[ 4 ][ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] +
yPredTmp[ C * 2:C * 5 / 2 ][ :yHP / 2 ] [:yWP / 2 ]
```

8.3.2.4.7 第 5 阶特征张量预测

该过程的输入为：

——yHyperPadDownPart[1][:C * 2][:yHP / 2][:yWP / 2]；

——yResDQPadDownPart[5][:C / 2][:yHP / 2][:yWP / 2]；

——yTmpModified[C / 2:C][:yHP / 2][:yWP / 2]；

——yTmp[C * 2:C * 5 / 2][:yHP / 2][:yWP / 2]。

该过程的输出为：yTmp[C * 5 / 2:C * 3][:yHP / 2][:yWP / 2]。

该过程依次执行以下操作：

a） 进行二维卷积：

```
yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] =
Conv( C / 2, C / 2, 1, 3, 3 )( yTmp[ C * 2:C * 5 / 2][ :yHP / 2 ][ :yWP / 2 ] )
```

b） 进行张量拼接：

```
Tmp[ :C ][ :yHP / 2 ][ :yWP / 2 ] =
Concat(yTmpModified[ C / 2:C ][ :yHP / 2 ][ :yWP / 2 ], yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ])
```

c） 进行张量拼接：

```
yHyperPadDownPartCPad[ 5 ][ :C * 3 ][ :yHP / 2 ][ :yWP / 2 ] =
Concat(Tmp[ :C ][ :yHP / 2 ][ :yWP / 2 ], yHyperPadDownPart[ 1 ][ :C * 2 ][ :yHP / 2 ][ :yWP / 2 ])
```

d) 将 i 设置为 5，进行预测融合（见 8.3.2.4.11）；

e) 进行预测补偿：

```
yTmp[ C * 5 / 2:C * 3 ][ :yHP / 2 ][ :yWP / 2 ]=
  yResDQPadDownPart[ 5 ][ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] +
yPredTmp[ C * 5 / 2:C * 3 ][ :yHP / 2 ][ :yWP / 2 ]
```

8.3.2.4.8 第 6 阶特征张量预测

该过程的输入为：

——yHyperPadDownPart[2][:C * 2][:yHP / 2][:yWP / 2]；

——yResDQPadDownPart[6][:C / 2][:yHP / 2][:yWP / 2]；

——yTmpModified[C:C * 3 / 2][:yHP / 2][:yWP / 2]；

——yTmp[C * 2:C * 3][:yHP / 2][:yWP / 2]。

该过程的输出为：yTmp[C * 3:C * 7 / 2][:yHP / 2][:yWP / 2]。

该过程依次执行以下操作：

a) 进行二维卷积：

```
yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] =
Conv( C, C / 2, 1, 3, 3 )( yTmp[ C * 2:C * 3 ][ :yHP / 2 ][ :yWP / 2 ] )
```

b) 进行张量拼接：

```
Tmp[ :C ][ :yHP / 2 ][ :yWP / 2 ] =
Concat(yTmpModified[ C:C * 3 / 2 ][ :yHP / 2 ][ :yWP / 2 ], yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ])
```

c) 进行张量拼接：

```
yHyperPadDownPartCPad[ 6 ][ :C * 3 ][ :yHP / 2 ][ :yWP / 2 ] =
Concat(Tmp[ :C ][ :yHP / 2 ][ :yWP / 2 ], yHyperPadDownPart[ 2 ][ :C * 2 ][ :yHP / 2 ][ :yWP / 2 ])
```

d) 将 i 设置为 6，进行预测融合（见 8.3.2.4.11）；

e) 进行预测补偿：

```
yTmp[ C * 3:C * 7 / 2 ][ :yHP / 2 ][ :yWP / 2 ]=
  yResDQPadDownPart[ 6 ][ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] +
yPredTmp[ C * 3:C * 7 / 2 ][ :yHP / 2 ][ :yWP / 2 ]
```

8.3.2.4.9 第 7 阶特征张量预测

该过程的输入为：

——yHyperPadDownPart[3][:C * 2][:yHP / 2][:yWP / 2]；

——yResDQPadDownPart[7][:C / 2][:yHP / 2][:yWP / 2]；

——yTmpModified[C * 3 / 2:C * 2][:yHP / 2][:yWP / 2]；

——yTmp[C * 2:C * 7 / 2][:yHP / 2][:yWP / 2]。

输出为：yTmp[C * 7 / 2:C * 4][:yHP / 2][:yWP / 2]。

该过程依次执行以下操作：

a) 进行二维卷积：

```
yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] =
Conv( C * 3 / 2, C / 2, 1, 3, 3 )( yTmp[ C * 2:C * 7 / 2 ][ :yHP / 2 ][ :yWP / 2 ] )
```

b) 进行张量拼接：

```
Tmp[ :C ][ :yHP / 2 ][ :yWP / 2 ] =
```

```
Concat(yTmpModified[ C * 3 / 2:C * 2 ][ :yHP / 2 ][ :yWP / 2 ], yTmpConv[ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ])
```

c） 进行张量拼接：

```
yHyperPadDownPartCPad[ 7 ][ :C * 3 ][ :yHP / 2 ][ :yWP / 2 ] =
Concat(Tmp[ :C ][ :yHP / 2 ][ :yWP / 2 ], yHyperPadDownPart[ 3 ][ :C * 2 ][ :yHP / 2 ][ :yWP / 2 ])
```

d） 将 i 设置为 7，进行预测融合（见 8.3.2.4.11）；

e） 进行预测补偿：

```
yTmp[ C * 7 / 2:C * 4 ][ :yHP / 2 ][ :yWP / 2 ]=
  yResDQPadDownPart[ 7 ][ :C / 2 ][ :yHP / 2 ][ :yWP / 2 ] +
yPredTmp[ C * 7 / 2:C * 4 ][ :yHP / 2 ][ :yWP / 2 ]
```

8.3.2.4.10 预测张量调整

该过程的输入为：yTmp[:C * 2][:yHP / 2][:yWP / 2]。

输出为：yTmpModified[:C * 2][:yHP / 2][:yWP / 2]。

该过程中使用的预测网络见图 3，该过程依次执行以下操作：

a） 进行交叉超分重组 CrossUpShuffle(yTmp[:C * 2][:yHP / 2][:yWP / 2])；

b） 进行二维卷积 Conv(C / 2, C, 1, 3, 3)；

c） 使用标准激活函数 ReLU( )；

d） 进行二维卷积 Conv(C, C, 1, 3, 3)；

e） 使用标准激活函数 ReLU( )；

f） 进行二维卷积 Conv(C, C / 2, 1, 3, 3)；

g） 进行交叉降分重组：yTmpModified[:C * 2][:yHP / 2][:yWP / 2] = CrossDownShuffle( )。

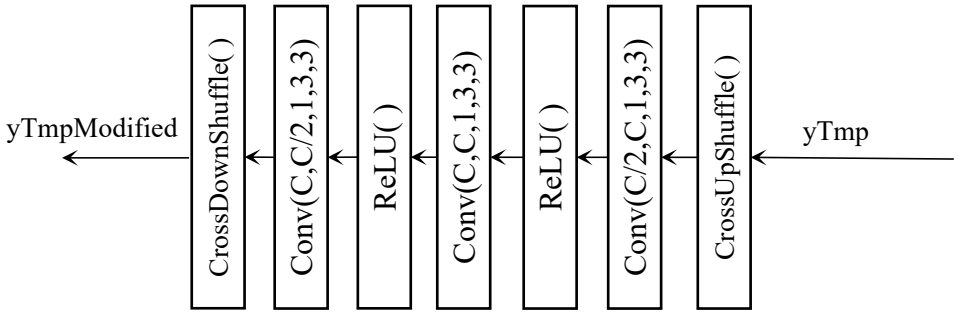

图 3 预测张量调整网络

8.3.2.4.11 预测融合

该过程的输入为：

——i，其取值范围是 0 至 7；

——yHyperPadDownPartCPad[i][:C * 3][:yHP / 2][:yWP / 2]。

输出为：yPredTmp[i * C / 2:(i + 1) * C / 2][:yHP / 2][:yWP / 2]。

该过程中使用的预测网络见图 4，当 i 取不同的值时，该网络有不同的实例，应使用不同的参数，该过程依次执行以下操作：

a） 进行二维卷积

Conv(C * 3, C * 9 / 4, 1, 1, 1)(yHyperPadDownPartCPad[i][:C * 3][:yHP / 2][:yWP / 2])；

b) 使用标准激活函数 ReLU( )；

c) 进行二维卷积 Conv(C * 9 / 4,C * 7 / 4, 1, 1, 1)；

d) 使用标准激活函数 ReLU( )；

e) 进行二维卷积：yPredTmp[i * C / 2:(i + 1) * C / 2][:yHP / 2][:yWP / 2] = Conv(C * 7 / 4,C / 2, 1, 3, 3)。

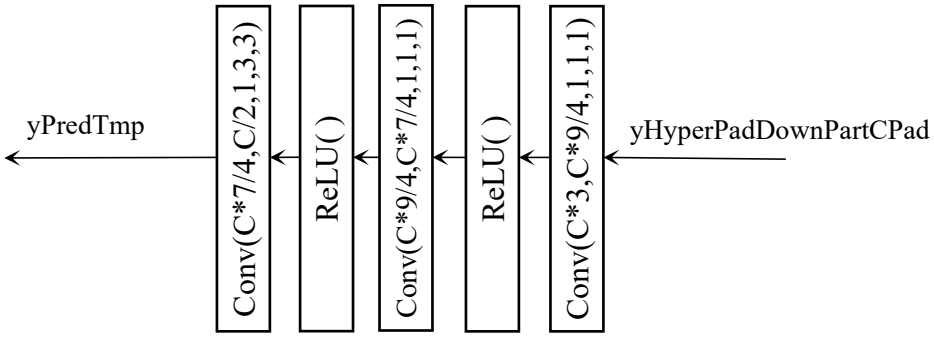

图 4　预测融合网络

8.3.2.5　特征张量调制

该过程的输入为：yRec[:C][:yH][:yW]。

该过程的输出为：y[:C][:yH][:yW]。

该过程使用的特征张量调制参数生成网络见图 5，该过程依次执行以下步骤：

a) 对码率控制因子进行扩展：

```
for ( i = 0; i < yH; i++ )
    for (j = 0; j < yW; j++)
        qRCMap[ i ][ j ] = qRC
```

b) 进行二维卷积：

```
qRCTensor[ :C ][ :yH ][ :yW ] = Conv( 1, C, 1, 3, 3)( qRCMap[ :1 ][ :yH ][ :yW ] )
```

c) 使用标准激活函数：

```
qRCTensor[ :C ][ :yH ][ :yW ] = ReLU( qRCTensor[ :C ][ :yH ][ :yW ] )
```

d) 进行二维深度卷积：

```
qRCOffsetTmp[ :C ][ :yH ][ :yW ] = DepthConv( C, 1, 1, 1 )( qRCTensor[ :C ][ :yH ][ :yW ] )
```

e) 进行二维卷积：

```
qRCOffset[ :C ][ :yH ][ :yW ] = Conv( C, C, 1, 1, 1) ( qRCOffsetTmp[ :C ][ :yH ][ :yW ] )
```

f) 进行二维深度卷积：

```
qRCScaleTmp[ :C ][ :yH ][ :yW ] = DepthConv( C, 1, 1, 1 )( qRCTensor[ :C ][ :yH ][ :yW ] )
```

g) 进行二维卷积：

```
qRCScale[ :C ][ :yH ][ :yW ] = Conv( C, C, 1, 1, 1) ( qRCScaleTmp[ :C ][ :yH ][ :yW ] )
```

h) 执行以下计算：

```
for ( i = 0; i < C; i++ )
    for (j = 0; j < yH; j++)
        for (k = 0; k < yW; k++)
            y[ i ][ j ][ k ] = ( yRec[ i ][ j ][ k ] - qRCOffset[ i ][ j ][ k ] ) * qRCScale[ i ][ j ][ k ]
```

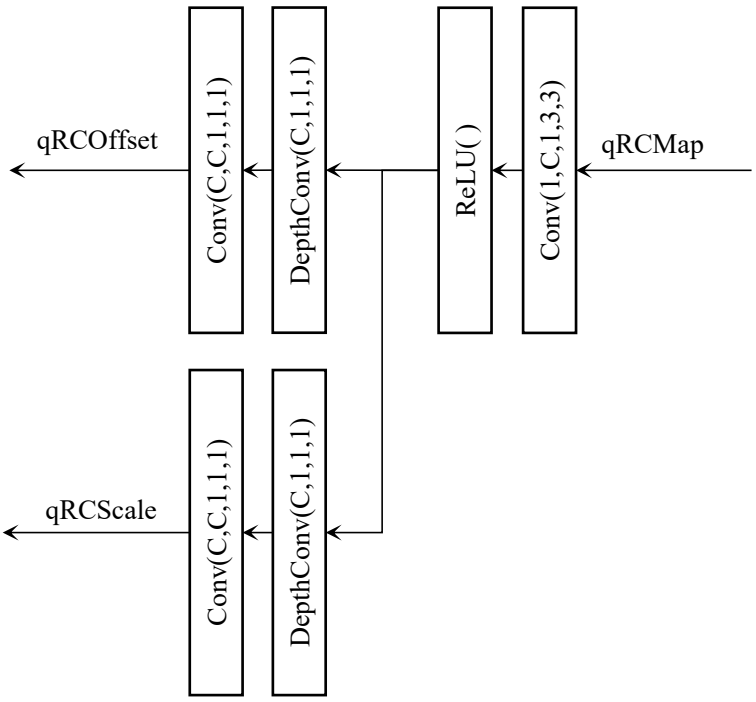

图 5　特征张量调制参数生成网络结构

8.3.3　特征张量超分

该过程的输入为：y[:C][:yH][:yW]。

该过程的输出为：r[:C][:rH][:rW]，其中，

——rW = yW * yScaleFactor；

——rH = yH * yScaleFactor。

该过程使用的反变换网络见图 6，该过程依次执行以下操作：

a)　进行二维残差卷积：

```
T1[ :C ][ :yH ][ :yW ] = ResConv( C, C, 0 )(y[ :C ][ :yH ][ :yW ])
```

b)　进行二维卷积：

```
T2[ :C * 4 ][ :yH ][ :yW ] = Conv( C, C * 4, 1, 3, 3 )(T1[ :C ][ :yH ][ :yW ])
```

c)　进行超分重组：

```
T3[ :C ][ :yH * 2 ][ :yW * 2 ] = Shuffle(2)(T2[ :C * 4 ][ :yH ][ :yW ])
```

d)　进行二维加权卷积：

```
T4[ :C ][ :yH * 2 ][ :yW * 2 ] = MaskConv( C )(T3[ :C ][ :yH * 2 ][ :yW * 2 ])
```

e)　进行二维残差卷积：

```
T5[ :C ][ :yH * 2 ][ :yW * 2 ] = ResConv( C, C, 0 )(T4[ :C ][ :yH * 2 ][ :yW * 2 ])
```

f)　进行二维卷积：

```
T6[ :C * 4 ][ :yH * 2 ][ :yW * 2 ] = Conv( C, C * 4, 1, 3, 3 )(T5[ :C ][ :yH * 2 ][ :yW * 2 ])
```

g)　进行超分重组：

```
T7[ :C ][ :rH ][ :rW ] = Shuffle(2)(T6[ :C * 4 ][ :yH * 2 ][ :yW * 2 ])
```

h)　进行二维加权卷积：

```
T8[ :C ][ :rH ][ :rW ] = MaskConv( C )(T7[ :C ][ :rH ][ :rW ])
```

i） 使用二维残差卷积：

```
r[ :C ][ :rH ][ :rW ] = ResConv( C, C, 0 )(T8[ :C ][ :rH ][ :rW ])
```

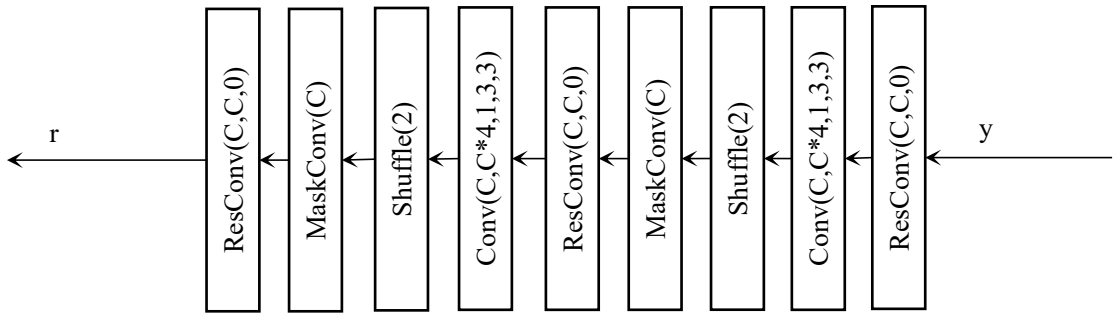

图 6　特征张量超分网络结构

8.4　解码图像结构数据

8.4.1　通则

解码过程如下：

a） 如果 bounding_boxes_enabled_flag 的值为 1，解码目标框（见 8.4.2），得到目标框。

8.4.2　解码目标框

该过程的输出为：目标框 BoundingBox[:BoundingBoxNum]。

该过程的操作如下：

a） 将 i 的值初始化为 0，重复 BoundingBoxNum 次以下操作：

1） 第 i 个目标框的起始点的横坐标 BoundingBox[i]->x 等于 bounding_box_x[i]的值；

2） 第 i 个目标框的起始点的纵坐标 BoundingBox[i]->y 等于 bounding_box_y[i]的值；

3） 第 i 个目标框的高度 BoundingBox[i]->h 等于 bounding_box_h[i]的值；

4） 第 i 个目标框的宽度 BoundingBox[i]->w 等于 bounding_box_w[i]的值；

5） 第 i 个目标框的类别标号 BoundingBox[i]->c 等于 bounding_box_category_id[i]的值；

6） 将 i 的值加 1。

8.5　解码图像重建数据

该过程的输入为：r[:C][:rH][:rW]。

该过程的输出为：重建图像，其图像宽度 riW 和高度 riH 分别为：

——riW = rW * 4 - crop_left_size - crop_right_size；

——riH = rH * 4 - crop_upper_size - crop_bottom_size。

该过程使用的反变换网络见图 7，该过程依次执行以下操作：

a） 进行二维残差卷积：

```
RT1[ :C ][ :rH ][ :rW ] = ResConv( C, C, 1 )( r[ :C ][ :rH ][ :rW ] )
```

b） 进行二维加权卷积：

```
RT2[ :C ][ :rH ][ :rW ] = MaskConv( C )( RT1[ :C ][ :rH ][ :rW ] )
```

c） 进行二维卷积：

```
RT3[ :C / 2 ][ :rH ][ :rW ] = Conv( C, C / 2, 1, 3, 3 )( RT2[ :C ][ :rH ][ :rW ] )
```

a) 进行二维卷积：

```
RT4[ :C * 2 ][ :rH ][ :rW ] = Conv( C / 2, C * 2, 1, 3, 3 )( RT3[ :C / 2 ][ :rH ][ :rW ] )
```

b) 进行超分重组：

```
RT5[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] = Shuffle( 2 )( RT4[ :C * 2 ][ :rH ][ :rW ] )
```

c) 进行二维残差卷积：

```
RT6[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] = ResConv( C / 2, C / 2, 1 )( RT5[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] )
```

a) 进行二维残差卷积：

```
RT7[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] = ResConv( C / 2, C / 2, 1 )( RT6[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] )
```

b) 进行二维加权卷积：

```
RT8[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] = MaskConv( C / 2 )( RT7[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] )
```

c) 进行二维残差卷积：

```
RT9[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] = ResConv( C / 2, C / 2, 1 )( RT8[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] )
```

a) 进行二维残差卷积：

```
RT10[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] = ResConv( C / 2, C / 2, 1 )( RT9[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] )
```

b) 进行张量加和：

```
RT11[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] =
Add( RT5[ :C / 2 ][ :rH * 2 ][ :rW * 2 ], RT10[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] )
```

c) 进行二维卷积：

```
RT12[ :C * 2 ][ :rH * 2 ][ :rW * 2 ] = Conv( C / 2, C * 2, 1, 3, 3 )( RT11[ :C / 2 ][ :rH * 2 ][ :rW * 2 ] )
```

d) 进行超分重组：

```
RT13[ :C / 2 ][ :rH * 4 ][ :rW * 4 ] = Shuffle( 2 )( RT12[ :C * 2 ][ :rH * 2 ][ :rW * 2 ] )
```

e) 使用二维残差卷积：

```
RT14[ :C / 2 ][ :rH * 4 ][ :rW * 4 ] = ResConv( C / 2, C / 2, 1 )( RT13[ :C / 2 ][ :rH * 4 ][ :rW * 4 ] )
```

f) 进行二维卷积：

```
RT15[ :3 ][ :rH * 4 ][ :rW * 4 ] = Conv( C / 2, 3, 1, 3, 3 )( RT14[ :C / 2 ][ :rH * 4 ][ :rW * 4 ] )
```

g) 进行裁剪：

```
for ( j = crop_upper_size; j < rH * 4 - crop_bottom_size; j++ ) {
    for ( k = crop_left_size; k < rW * 4 - crop_right_size; k++ ) }
        imR[ j - crop_upper_size ][ k - crop_left_size ] = RT15[ 0 ][ j ][ k ]
        imG[ j - crop_upper_size ][ k - crop_left_size ] = RT15[ 1 ][ j ][ k ]
        imB[ j - crop_upper_size ][ k - crop_left_size ] = RT15[ 2 ][ j ][ k ]
    }
}
```

h) 如果 rec_image_format_id 的值为 0 或 1 或 2，进行格式转换：

```
for ( i = 0; i < riH; i++ )
    for ( j = 0; j < riW; j++ )
        imY[ i ][ j ] = 0.257 * imR[ i ][ j ] + 0.504 * imG[ i ][ j ] + 0.098 * imB[ i ][ j ] + 16
        imCb[ i ][ j ] = 0.439 * imR[ i ][ j ] - 0.368 * imG[ i ][ j ] - 0.071 * imB[ i ][ j ] + 128
        imCr[ i ][ j ] = -0.148 * imR[ i ][ j ] - 0.291 * imG[ i ][ j ] + 0.439 * imB[ i ][ j ] + 128
```

i) 进行取整和截断，得到重建图像：

1) 如果 rec_image_format_id 的值为 0，重建图像的亮度分量 im0[:riH][:riW]、色度 Cr 分量

im1[:(riH+1)/2][:(riW+1)/2]和色度 Cb 分量 im2[:(riH+1)/2][:(riW+1)/2]按照以下计算得到：

```
for ( i = 0; i < riH; i++ )
    for ( j = 0; j < riW; j++ )
        im0[ i ][ j ] = bit_depth_id ? clip3( 0, 1023, ⌈ imY[ i ][ j ] *4 ⌉ ) : clip3( 0, 255, ⌈ imY[ i ][ j ] ⌉ )
for ( i = 0; i < ( riH + 1 ) / 2; i++ ) {
    for ( j = 0; j < ( riW + 1 ) / 2; j++ ) {
        im1[ i ][ j ] = bit_depth_id ? clip3( 0, 1023, ⌈ imCb[ i * 2 ][ j * 2 ] *4 ⌉ ) :
        clip3( 0, 255, ⌈ imCb[ i * 2 ][ j * 2 ] ⌉ )
        im2[ i ][ j ] = bit_depth_id ? clip3( 0, 1023, ⌈ imCr[ i * 2 ][ j * 2 ] *4 ⌉ ) :
        clip3( 0, 255, ⌈ imCr[ i * 2 ][ j * 2 ] ⌉ )
    }
}
```

2） 如果 rec_image_format_id 的值为 1，重建图像的亮度分量 im0[:riH][:riW]、色度 Cr 分量 im1[:riH][:(riW+1)/2]和色度 Cb 分量 im2[:riH][:(riW+1)/2]按照以下计算得到：

```
for ( i = 0; i < riH; i++ )
    for ( j = 0; j < riW; j++ )
        im0[ i ][ j ] = bit_depth_id ? clip3( 0, 1023, ⌈ imY[ i ][ j ] *4 ⌉ ) : clip3( 0, 255, ⌈ imY[ i ][ j ] ⌉ )
for ( i = 0; i < riH; i++ ) {
    for ( j = 0; j < ( riW + 1 ) / 2; j++ ) {
        im1[ i ][ j ] = bit_depth_id ? clip3( 0, 1023, ⌈ imCb[ i ][ j * 2 ] *4 ⌉ ) :
        clip3( 0, 255, ⌈ imCb[ i ][ j * 2 ] ⌉ )
        im2[ i ][ j ] = bit_depth_id ? clip3( 0, 1023, ⌈ imCr[ i ][ j * 2 ] *4 ⌉ ) :
        clip3( 0, 255, ⌈ imCr[ i ][ j * 2 ] ⌉ )
    }
}
```

3） 如果 rec_image_format_id 的值为 2，重建图像的亮度分量 im0[:riH][:riW]、色度 Cr 分量 im1[:riH][:riW]和色度 Cb 分量 im2[:riH][:riW]按照以下计算得到：

```
for ( i = 0; i < riH; i++ ) {
    for ( j = 0; j < riW; j++ ) {
        im0[ i ][ j ] = bit_depth_id ? clip3( 0, 1023, ⌈ imY[ i ][ j ] *4 ⌉ ) : clip3( 0, 255, ⌈ imY[ i ][ j ] ⌉ )
        im1[ i ][ j ] = bit_depth_id ? clip3( 0, 1023, ⌈ imCb[ i ][ j ] *4 ⌉ ) : clip3( 0, 255, ⌈ imCb[ i ][ j ] ⌉ )
        im2[ i ][ j ] = bit_depth_id ? clip3( 0, 1023, ⌈ imCr[ i ][ j ] *4 ⌉ ) : clip3( 0, 255, ⌈ imCr[ i ][ j ] ⌉ )
    }
}
```

4） 如果 rec_image_format_id 的值为 3，重建图像的 R 分量 im0[:riH][:riW]、G 分量 im1[:riH][:riW]和 B 分量 im2[:riH][:riW]按照以下计算得到：

```
for ( i = 0; i < riH; i++ ) {
    for ( j = 0; j < riW; j++ ) {
        im0[ i ][ j ] = clip3( 0, 255, ⌈ imR[ i ][ j ] ⌉ )
        im1[ i ][ j ] = clip3( 0, 255, ⌈ imG[ i ][ j ] ⌉ )
        im2[ i ][ j ] = clip3( 0, 255, ⌈ imB[ i ][ j ] ⌉ )
    }
}
```

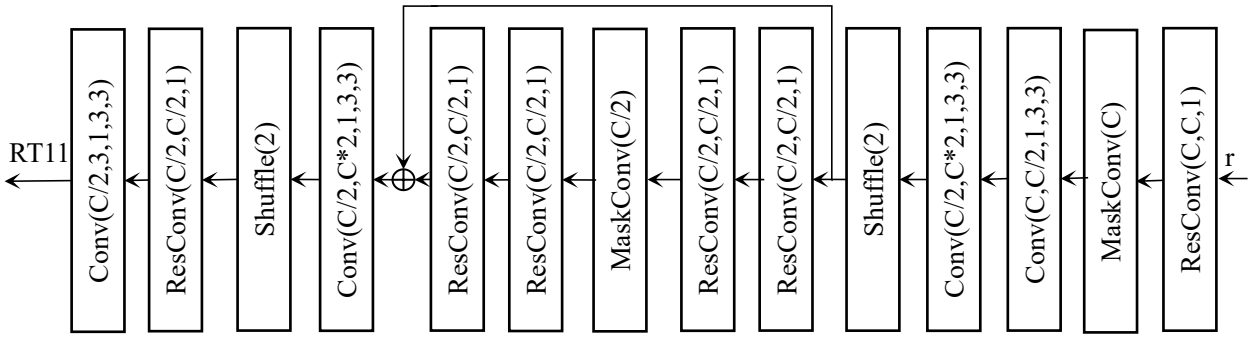
RT11
Conv(C/2,3,1,3,3)
ResConv(C/2,C/2,1)
Shuffle(2)
Conv(C/2,C*2,1,3,3)
ResConv(C/2,C/2,1)
ResConv(C/2,C/2,1)
MaskConv(C/2)
ResConv(C/2,C/2,1)
ResConv(C/2,C/2,1)
Shuffle(2)
Conv(C/2,C*2,1,3,3)
Conv(C,C/2,1,3,3)
MaskConv(C)
ResConv(C,C,1)
r

图 7　图像重建网络结构

附 录 A
（规范性）
伪起始码方法

本附录定义了防止在位流中出现伪起始码的方法。起始码的形式、含义，以及为了使起始码字节对齐而进行填充的方法见5.9.2和6.1.1。

为了防止出现伪起始码，编码时应按照以下方法处理：写入一位时，如果该位是一个字节的第二最低有效位，检查该位之前写入的22位，如果这22位都是‘0’，在该位之前插入‘10’，该位成为下一个字节的最高有效位。

解码时应按以下方法处理：每读入一个字节时，检查前面读入的两个字节和当前字节，如果这三个字节构成位串‘0000 0000 0000 0000 0000 0010’，丢弃当前字节的最低两个有效位。丢弃一个字节最低两个有效位可采用任意等效的方式，本文件不做规定。

在编码和解码时对于图像头中的数据不应采用上述方法。

附 录 B
（规范性）
档次

B.1 概述

档次提供了一种定义本文件的语法和语义的子集的手段。档次对位流进行了各种限制，同时也就规定了对某一特定位流解码所需要的解码器能力。档次是本文件规定的语法、语义及算法的子集。符合某个档次规定的解码器应完全支持该档次定义的子集。

本附录描述了不同档次所对应的各种限制。所有未被限定的语法元素和参数可以取任何本文件所允许的值。如果一个解码器能对某个档次所规定的语法元素的所有允许值正确解码，则称此解码器在这个档次上符合本文件。如果一个位流中不存在某个档次所不允许的语法元素，并且其所含有的语法元素的值不超过此档次所允许的范围，则认为此位流在这个档次上符合本文件。

profile_id定义了位流的档次。

B.2 档次

本文件定义的档次见A。

表 B.1 档次

profile_id的值	档次
0x0	禁止
0x1	基本特征档次（Main profile）
0x2	基本图像档次（High profile）
其他	保留

基本特征档次的尾流应满足以下条件：

——profile_id 的值应为 0x1；

——image_rec_enabled_flag 的值应为 0；

——解析和解码过程中使用的常量符号及其对应值应符合表 B.2 的规定。

基本图像档次的尾流应满足以下条件：

——profile_id 的值应为 0x2；

——image_rec_enabled_flag 的值应为 1；

——解析和解码过程中使用的常量符号及其对应值应符合表 B.2 的规定。

表 B.2 常量符号及其对应的含义和值

常量符号	含义	对应值
C	通道数	128
zN	z的累计概率列表数	128
yN	y的累计概率列表数	64
yP	y的概率精度	31

表B.2 常量符号及其对应的含义和值（续）

常量符号	含义	对应值
StatePrecision	状态值精度	32
CdfPrecision	频率值精度	16
BypassPrecision	熵解码跳过精度	4
ScaleLowBound	概率下限	0.11
zScaleFactor	超先验张量的放大倍数	4
yScaleFactor	特征张量的放大倍数	4

附 录 C
（规范性）
神经网络模型参数

神经网络模型参数包括神经网络中卷积操作使用的卷积核权重和偏置以及神经网络中其他操作的预设参数。

解析和解码过程中使用的神经网络模型参数，按照PyTorch的序列化状态字典格式存储在电子附件中，获取路径为：https://pan.zju.edu.cn/share/506555c6f2b1906fb4c549293f。

附 录 D
（规范性）
解析过程中使用的数据以及码表

D.1 解析 z[i][j][k]时使用的矩阵、列表和张量

解析语法元素z[i][j][k]时使用的矩阵、列表和张量及其获取方式如下：

——列表 CDFLength，其数据内容参见 URL0 的 csv 文件，其存放格式为：令 i=0~zN，逐行存放 CDFLength[i]；

——矩阵 CDFS，其数据内容参见 URL1 的 csv 文件，其存放格式为：令 i=0~zN，逐行存放 CDFS[i][:]；

——列表 MaxValues，其数据内容参见 URL2 的 csv 文件，其存放格式为：令 i=0~zN，逐行存放 MaxValues[i]；

——列表Offsets，其数据内容参见URL3的csv文件，其存放格式为：令i=0~zN，逐行存放Offsets[i]；

——张量 Indexs，其数据内容参见 URL4 的 csv 文件，其存放格式为：令 i=0~zN，逐行存放 Indexs[i]。

表 D.1 给出了本章中地址编号所对应的获取地址。

表 D.1 地址编号 URL0～4 对应的获取地址

地址编号	获取地址
URL0	https://pan.zju.edu.cn/share/9ffa3bd4936653b62edde793b4
URL1	https://pan.zju.edu.cn/share/7f28f3ee8841dd0ecd400a4981
URL2	https://pan.zju.edu.cn/share/12f16695be3d77134627de312b
URL3	https://pan.zju.edu.cn/share/a0d37a965e9a54dd860f259d94
URL4	https://pan.zju.edu.cn/share/967c95258b7ef510c50cd7b6b9

D.2 解析 y_residue[i][j][k]时使用的矩阵和列表

解析语法元素y_residue[i][j][k]时使用的矩阵和列表及其获取方式如下：

——列表 CDFLength，其数据内容参见 URL5 的 csv 文件，其存放格式为：令 i=0~yN，逐行存放 CDFLength[i]；

——矩阵 CDFS，其数据内容参见 URL6 的 csv 文件，其存放格式为：令 i=0~yN，逐行存放 CDFS[i][:]；

——列表 MaxValues，其数据内容参见 URL7 的 csv 文件，其存放格式为：令 i=0~yN，逐行存放 MaxValues[i]；

——列表Offsets，其数据内容参见URL8的csv文件，其存放格式为：令i=0~yN，逐行存放Offsets[i]；

——列表 ScaleTable，其数据内容参见 URL9 的 csv 文件，其存放格式为：令 i=0~yN，逐行存放 ScaleTable[i]。

表 D.2 给出了本章中地址编号所对应的获取地址。

表 D.2 地址编号 URL5～9 对应的获取地址

地址编号	获取地址
URL5	https://pan.zju.edu.cn/share/90b79d6ddbd909cf6e251515ed
URL6	https://pan.zju.edu.cn/share/ab35cbfe233478f1d8b9eb6193
URL7	https://pan.zju.edu.cn/share/7f03cf5b8163109e4ce7e158dc
URL8	https://pan.zju.edu.cn/share/9f60a1e0307a201bb58af270b9
URL9	https://pan.zju.edu.cn/share/b102cbe51b6e23740b4dd8c65d

D.3 块掩膜码表

解析group_mask_value时使用的码表见表D.3。

表 D.3 块掩膜 group_mask_value 的码表

符号值	符号串
0	0101
1	0100
2	0011
3	0010
4	0001
5	11111
6	11110
7	11101
8	11100
9	11010
10	11001
11	11000
12	10110
13	10101
14	10011
15	10010
16	10000
17	01111
18	01101
19	00001
20	00000
21	110110
22	101110
23	101001
24	100011
25	100010
26	011100
27	011000
28	1101110
29	1011110
30	1010001
31	0111011
32	0110011
33	0110010

表 D.3　块掩膜 **group_mask_value** 的码表（续）

符号值	符号串
34	11011110
35	10111110
36	10100000
37	01110100
38	110111111
39	101111110
40	101000010
41	011101011
42	011101010
43	1101111100
44	1010000111
45	1010000110
46	11011111011
47	10111111101
48	10111111100
49	101111111110
50	110111110101
51	101111111101
52	11011111010001
53	1101111101001
54	1011111111110
55	1011111111111
56	101111111100
57	1101111101000011111011110000000111111111111111111111111111111111111111111111110100
58	110111110100001111101111000000011111110101011111111
59	1101111101000011111011110000000111111101011111111111111110
60	11011111010000111110111100000001111111111111111111111111111111111111111111101010
61	110111110100001111101111000000011111111111111111111111111111111011111110101011111100
62	1101111101000011111011110000000111111111111111111111111111111011111110101011111111010
63	110111110100001111101111000000011111111111111111111111111111001
64	11011111010000110
65	1101111101000011111011110000000111111111111111111111111111111010
66	11011111010000101
67	1101111101000011111011110000000111111111111111111111111111110110
68	110111110100001000
69	1101111101000011111011110000000111111111111111111111111111101110
70	110111110100001111110110
71	1101111101000011111011110000000111111111111111111111111111011110
72	110111110100001111011111

表 D.3　块掩膜 **group_mask_value** 的码表（续）

符号值	符号串
73	1101111101000011111011110000000111111111111111111111111111111110111110
74	11011111010000111110111101
75	110111110100001111101111000000011111111111111111111111111111101111110
76	11011111010000100111000 1
77	11011111010000100111000 01
78	110111110100001001110000001
79	1101111101000011111011110000000111111111111111111111111111110111111110
80	11011111010000111110111100000000
81	1101111101000011111011110000000111111111111111111111111111101111111110
82	110111110100001001110000000110
83	1101111101000011111011110000000111111111111111111111111111011111111110
84	110111110100001111101111000000011110
85	1101111101000011111011110000000111111111111111111111111110111111111110
86	11011111010000111110111100000001111111110
87	1101111101000011111011110000000111111111111111111111111101111111111110
88	110111110100001111101111000000011111111110
89	1101111101000011111011110000000111111111111111111111111011111111111110
90	110111110100001111101111000000011111111111110
91	1101111101000011111011110000000111111111111111111111111011111111111110
92	110111110100001111101111000000011111111111111110
93	11011111010000111110111100000001111111111111111111110
94	110111110100001111101111000000011111111111111111111110
95	11011111010000111110111100000001111111010110
96	110111110100001111101111000000011111111111111111111111110010
97	11011111010000111110111100000001111111010 1110
98	1101111101000011111011110000000111111111111111111111111110001011
99	11011111010000111110111100000001111111010111110
100	1101111101000011111011110000000111111111111111111111111100111110
101	11011111010000111110111100000001111111010111110
102	1101111101000011111011110000000111111111111111111111111110011111110
103	11011111010000111110111100000001111111010111110
104	1101111101000011111010101010
105	11011111010000111110111100000001111111010111111110
106	1101111101000011111011110000011011
107	11011111010000111110111100000001111111010111111110
108	1101111101000011111010101011110
109	11011111010000111110111100000001111111010 0
110	1101111101000011111010101011111111
111	11011111010000111110111100000001111111010111111111110

表 D.3　块掩膜 **group_mask_value** 的码表（续）

符号值	符号串
112	1101111101000011111011110000000111111110101010
113	11011111010000111110111100000001111111010111111111110
114	110111110100001111101111000000011111111111111111111101011
115	11011111010000111110111100000001111111010111111111110
116	1101111101000011111011110000000111111101010111110
117	11011111010000111110111100000001111111010111111111110
118	1101111101000011111011110000000111111101010111111110
119	11011111010000111110111100000001111111111111111111111111111111101111111010101110
120	1101111101000011111011110000000111111111111111111111111111111101111111010101010
121	11011111010000111110111100000001111111111111111111111111111111111111111111110101110
122	11011111010000111110111100000001111111111111111111111111111111111111111111111111111111
123	1101111101000011111011110000000111111101011111111111111111
124	11011111010000111110111100000001111111111111111111111111111111111111111111111111010111110
125	1101111101000011111011110000000111111101011111111111111111011
126	11011111010000111110111100000001111111111111111111111111111101111111010101111111001
127	110111110100001111101111000000011111111111111111111111111111111111111111111110101111111
128	110111110100001110
129	11011111010000111110111100000001111111111111111111111111111111000
130	1101111101000011110
131	11011111010000111110111100000001111111111111111111111111111110
132	1101111101000001
133	1101111101000000
134	11011111010000111111
135	11011111010000111110111100000001111111111111111111111111111110
136	110111110100001111100
137	11011111010000111110111100000001111111111111111111111111111100110
138	1101111101000011111101
139	11011111010000111110111100000001111111111111111111111111111110
140	110111110100001111100
141	1101111101000010010
142	1101111101000100110
143	11011111010000111110111100000001111111111111111111111111111110
144	1101111101000001001111
145	11011111010000111110111100000001111111111111111111111111110011110
146	1101111101000011111011101
147	11011111010000111110111100000001111111111111111111111111111110
148	1101111101000011111011100
149	1101111101000010011101
150	11011111010000100111001

表 D.3　块掩膜 **group_mask_value** 的码表（续）

符号值	符号串
151	11011111010000111110111100000001111111111111111111111111111111111110
152	1101111101000011111011111001
153	110111110100001111101111000000011111111111111111111111111111001111110
154	11011111010000111110111110001
155	110111110100001111101111000000011111111111111111111111111111111111110
156	11011111010000100111000001
157	110111110100001111101111000001
158	1101111101000011111011110000001
159	110111110100001111101111000000011111111111111111111111111111111111110
160	11011111010000100111000000000
161	110111110100001111101010100
162	11011111010000100111000000010
163	110111110100001111101111000000011111111111111111111111111111111111110
164	11011111010000111110111000000010
165	11011111010000100111000000111
166	11011111010000111110111000000110
167	110111110100001111101111000000011111111111111111111111111111111111110
168	11011111010000111110111000000001110
169	1101111101000011111010101010110
170	11011111010000111110111000000000111110
171	110111110100001111101111000000011111111111111111111111111111111111110
172	11011111010000111110111000000001111110
173	110111110100001111101111000000011111111110
174	1101111101000011111011110000000111111111110
175	110111110100001111101111000000011111111111111111111111111111111111110
176	1101111101000011111011110000000111111111110
177	1101111101000011111010101011110
178	110111110100001111101111000000011111111111110
179	110111110100001111101111000000011111111111111111111111111111111111110
180	11011111010000111110111000000011111111111110
181	1101111101000011111011110000000111111111111110
182	11011111010000111110111000000011111111111111110
183	110111110100001111101111000000011111111111111111111111111111111111110
184	11011111010000111110111000000011111111111111110
185	1101111101000011111010101011111110
186	11011111010000111110111000000011111111111111111110
187	1101111101000011111011110000000111111111111111111110
188	11011111010000111110111000000011111111111111111111110
189	110111110100001111101111000000011111111111111111111110

表 D.3　块掩膜 **group_mask_value** 的码表（续）

符号值	符号串
190	1101111101000011111011110000000111111111111111111111111110
191	11011111010000111110111100000001111111111111111111111111111111111111111110
192	1101111101000011111011110000000111111111111111111111111110000
193	110111110100001111101111000000011111111010100
194	1101111101000011111011110000000111111111111111111111111100011
195	11011111010000111110111100000001111111111111111111111111111111111111111110
196	1101111101000011111011110000000111111111111111111111111111000100
197	1101111101000011111011110000000111111111111111111111111111001110
198	1101111101000011111011110000000111111111111111111111111110001010
199	11011111010000111110111100000001111111111111111111111111111111111111111110
200	110111110100001111110100
201	1101111101000011111011110000000111111101010110
202	110111110100001111101011
203	11011111010000111110111100000001111111111111111111111111111111111111111110
204	110111110100001111010100
205	1101111101000011111011110000000111111111111111111111111110011111111
206	110111110100001111010101011
207	11011111010000111110111100000001111111111111111111111111111111111111111110
208	1101111101000011111011110000010
209	1101111101000011111011110000000111111101010111110
210	110111110100001111011110000111
211	11011111010000111110111100000001111111111111111111111111111111111111111110
212	1101111101000011111011110001100
213	1101111101000011111010101011110
214	1101111101000011111011110000011010
215	11011111010000111110111100000001111111111111111111111111111111111111111110
216	1101111101000011111011110000000111111100
217	1101111101000011111011110000000111111101010111111110
218	110111110100001111011110000000111111011
219	11011111010000111110111100000001111111111111111111111111111111111111111110
220	1101111101000011111011110000000111111111111111111111111111011111111111111
221	11011111010000111110111100000001111111111111111111111111111011111111111110
222	1101111101000011111010101011111110
223	11011111010000111110111100000001111111111111111111111111111111111111111110
224	1101111101000011111011110000000111111111111111111100
225	11011111010000111110111100000001111111111111111111111111111011111111010100
226	1101111101000011111011110000000111111111111111111111011
227	11011111010000111110111100000001111111111111111111111111111111111111111110
228	1101111101000011111011110000000111111111111111111110100

表 D.3　块掩膜 **group_mask_value** 的码表（续）

符号值	符号串
229	1101111101000011111011110000000111111101010111 0
230	1101111101000011111011110000000111111111111111111111101010
231	1101111101000011111011110000000111111111111111111111111111111111111111111111111111111111111110
232	1101111101000011111011110000000111111111111111111111111111111101111111100
233	1101111101000011111011110000000111111111111111111111111111111011111101010110
234	1101111101000011111011110000000111111111111111111111111111111011111111011
235	1101111101000011111011110000000111111111111111111111111111111111111111111111111111111111110
236	1101111101000011111011110000000111111111111111111111111111111011111110100
237	110111110100001111101111000000011111110101111111110
238	1101111101000011111011110000000111111111111111111111111111111011111101011
239	110111110100001111101111000000011111110101111111111110
240	1101111101000011111011110000000111111111111111111111111111111111111111111100
241	1101111101000011111011110000000111111111111111111111111111111111111111111010110
242	1101111101000011111011110000000111111111111111111111111111111111111111111011
243	1101111101000011111011110000000111111111111111111111111111111011111101010111110
244	1101111101000011111011110000000111111111111111111111111111111011111101010101110
245	110111110100001111101111000000011111110101111111111111110
246	1101111101000011111011110000000111111111111111111111111111111111111111111111111111111111110
247	1101111101000011111011110000000111111111111111111111111111111111111111111101011110
248	1101111101000011111011110000000111111111111111111111111111011111101010111111111
249	1101111101000011111011110000000111111111111111111111111111011111101010111111101
250	110111110100001111101111000000011111110101111111111111100
251	1101111101000011111011110000000111111111111111111111111111111111111111111101010111110
252	110111110100001111101111000000011111110101111111111111111010
253	1101111101000011111011110000000111111111111111111111111111011111101010111111111011
254	1101111101000011111011110000000111111111111111111111111111011111101010111111111000
255	110111110100001111101111000000011111111111111111111111111101

附 录 E
（资料性）
特征适配

特征适配网络负责将特征向量尺寸调整为智能任务网络所需的特征向量，其中参考软件中特征适配器的网络结构由一层卷积层以及 ReLU 激活层组成（见图 E.1）。

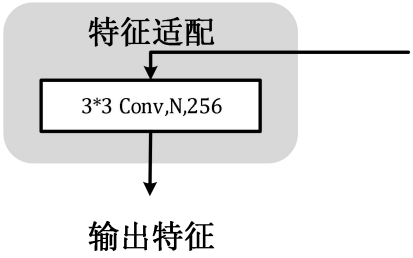

图 E.1 特征适配网络结构

当目标检测任务网络替换为 YOLO V3，考虑到 YOLO V3 的 Bacbone 的变化，一层卷积层适配器的参数量过少，在解码特征与 YOLO V3 接入点所需特征表征形式可能难以转化，所以训练适配器时除了采用一层卷积层外，还尝试了采用类似降采样的 Res-Block 组成的网络来进行适配，其网络结构见图 E.2。

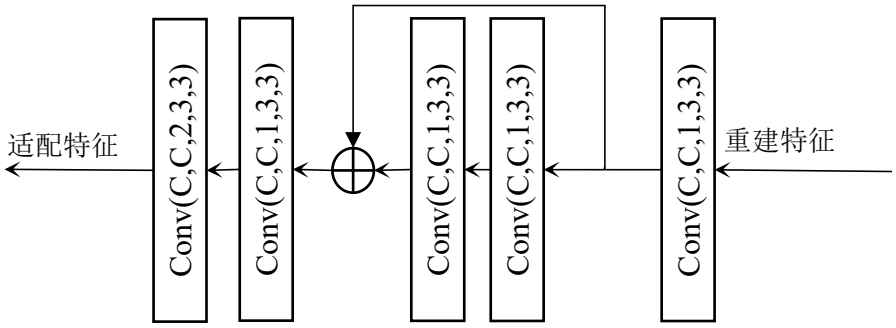

图 E.2 特征适配网络结构