2022 年全国大学生
嵌入式芯片与系统设计竞赛——
FPGA 创新设计竞赛
技术文档

作品名称	基于高云 FPGA 的智能售货机系统设计
队长	龚朴郅
队员 1	艾隆
队员 2	姜林澄

第一章 设计概述

1.1 设计目的

现如今，自动售货机被广泛应用，便利了人们的生活。随着收入水平的提高，人们追求更方便、更快捷的自助式服务。但自动售货机的货物较为单调（以饮料为主），且消费者购货、商家补货都很不方便。根据上述情况，本项目设计了基于高云FPGA的智能售货架系统，可以实现市面上大部分商品的智能识别。消费者输入身份信息后可以任取商品，系统将自动计算其购买商品的总金额并在其账户中扣除。

1.2 应用领域

智能售货架系统的应用范围广泛，可应用于小卖部、公园、景区、医院、工厂、学校、政府、机场、高速公路休息站等各类场所的智能化改造。此外，可适用于智能售货架系统的商品种类繁多，各类快消品（饮料、零食等）、智能产品及配件（蓝牙耳机、充电器、3D眼镜等）等都可以是智能售货架系统的待售产品。

1.3 技术特点

本项目设计了卷积神经网络结构以实现物体分类。在PC端，使用Matlab软件搭建卷积神经网络的训练、推理系统，并将卷积神经网络模型转化成硬件加速结构，部署到高云Tang 20k板卡中。在FPGA端，视频流通过连通域分割、卷积神经网络等多个环节，输出多种类多数量商品的实时定位与分类结果，并将识别结果直观地显示在HDMI屏上。在PicoRV软核的控制下，系统确认消费者的身份信息，自动计算消费者购买商品的总金额并在其账户中扣除。本系统采用流水线的图像处理方式，保证了视频流的传输速度和显示帧率，并极大程度提高了商品的识别速率。

1.4 关键性能指标

(1) 显示帧率能达到每秒60帧（HDMI屏的最大显示帧率为每秒60帧）。

(2) 可进行8个物体的定位和分类。

(3) 可进行8种物体的识别（包括无类别）。

(4) 物体的平均识别率达到95%。

(5) 识别速率稳定在每秒7~8帧。

(6) FPGA工程使用了14080个查找表(LUT)，约占总资源的68%；使用了7166个寄存器(REG)，约占总资源的44%；使用了41个BRAM(block RAM)，约占总资源的89%。

1.5 主要创新点

(1) 设计了基于FPGA的联通域分割算法。智能售货架系统具有特殊的环境条件，可利用图像的灰度化处理和二值化处理实现背景与商品的精确分离。对二值化图像进行的联通域分割，可以使整个图像被提取出若干个商品部分，将这些图像部分输入卷积神经网络便可判断其类别。本项目部署的卷积神经网络模型是物体分类的模型，但利用图像预处理、联通域分割算法等方法，可以达到物体定位和识别的效果。

(2) 设计了简单可行且易于移植的卷积神经网络实现物体分类。卷积神经网络由2个卷积层、2个池化层、1个全连接层组成。其输入大小为29*29，能输出8种类商品所对应的概率。在Matlab软件上进行卷积神经网络的训练、推理，并将模型部署到FPGA中，实现卷积神经网络模型的硬件加速。在部署模型时运用了量化的处理方法，将参与运算的数据由浮点数转换为8位整型数，减少了硬件资源的使用量。

(3) 采取了卷积神经网络和传统图像处理相融合的物体分类方法。单独使用传统图像处理方法进行分类时，无法对复杂颜色的商品进行分类。单独使用卷积神经网络进行分类时，分类效果不稳定。本项目融合了卷积神经网络的分类输出和传统图像处理的分类输出，在卷积神经网络的输出概率值之后，加上图像色域判断得到的概率值，使整体的分类效果更稳定。

(4) 构建了友善的人机交互系统。以PicoRV软核为控制核心，HDMI和串口屏商家作为显示终端。利用串口屏控件，商家和消费者能方便地补货取货。HDMI屏实时显示图像数据流，并显示中文标签、参数、图案等，便于商家查看设备运行情况。

第二章 系统组成及功能说明

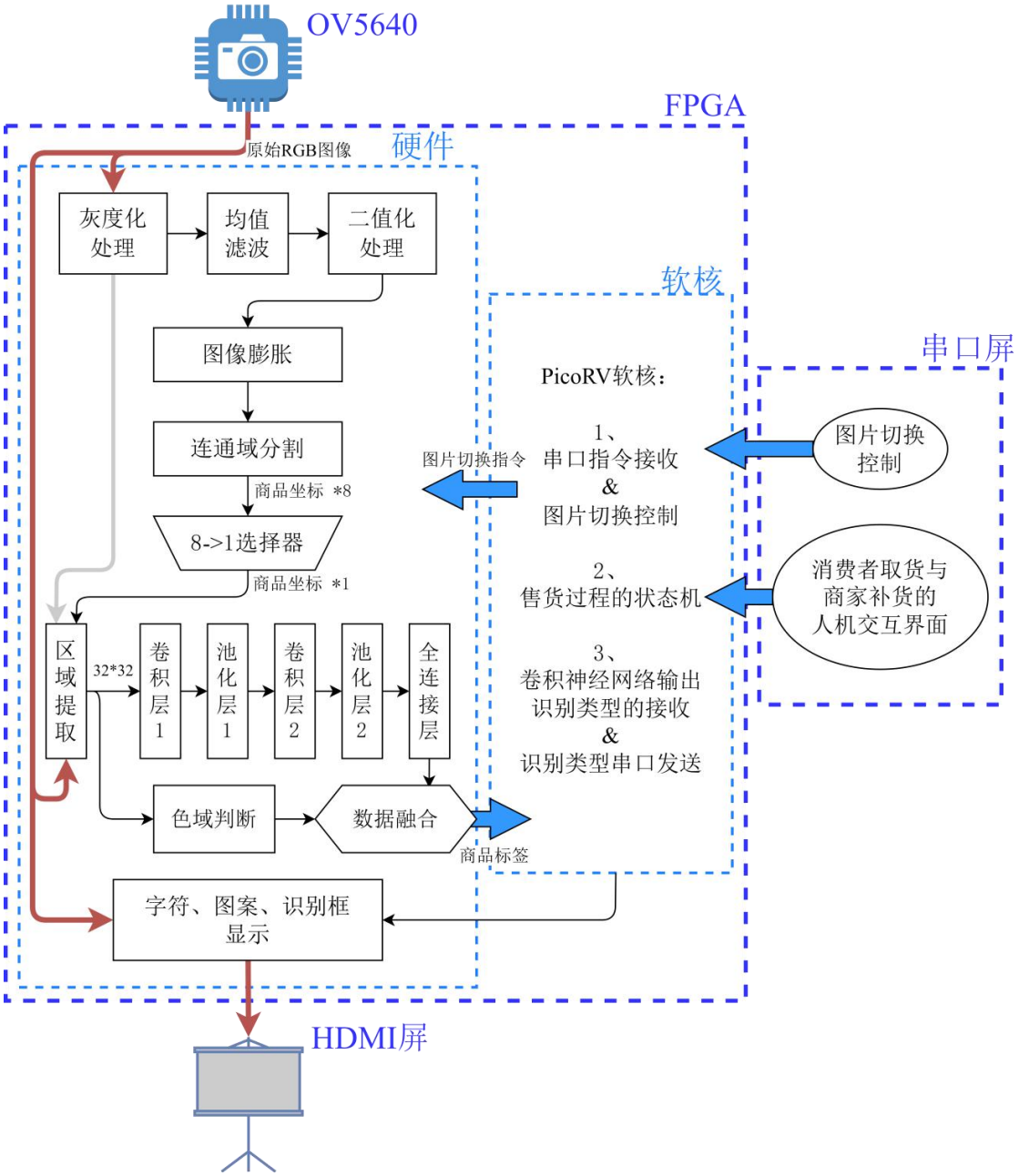

图 2.1 系统整体架构框图

如图2.1系统整体架构框图，系统被划分为FPGA部分和串口屏人机交互部分。其中，串口屏人机交互部分能通过按键、开关等控件发送串口指令，当用户（消费者或商家）输入的个人信息被系统确认时，串口屏发送串口指令。此外，还能使用串口屏控制图片的切换，直观地观察FPGA中各模块对图像的处理情况，以供测试过程中进一步的模型优化或代码修改。FPGA部分由软处理器端和硬件端

组成，PicoRV软处理器端负责调控，包括串口指令接收与图片切换控制、售货流程的状态机、卷积神经网络识别类型的接收和串口发送。硬件端分为图像预处理、图像膨胀、连通域分割、卷积神经网络图像识别、图像显示5个部分。图像预处理负责将图像数据进行灰度化处理、二值化处理和均值滤波。图像膨胀、连通域分割负责分割出商品的具体位置。卷积神经网络图像识别对图像进行区域提取，通过卷积层、池化层、全连接层后，输出图像分类的商品标签到PicoRV软核。图像显示将运行过程中的数据和运行结果以字符、图案、识别框的形式直观地显示在HDMI屏上。

2.1 硬件部分

2.1.1 图像预处理

图像预处理是进行数字图像处理的首要步骤。其不仅可以有效削弱图像的干扰因素，改善后续识别效果，还可以在简化图像数据的同时，增强有关信息的可检测性，确保后续识别的准确性与可靠性。本项目使用灰度化处理、二值化处理和均值滤波的图像预处理方法，减少了待处理的数据量，为后续的图像处理和物体识别做准备。下图2.2为图像预处理的流程框图。

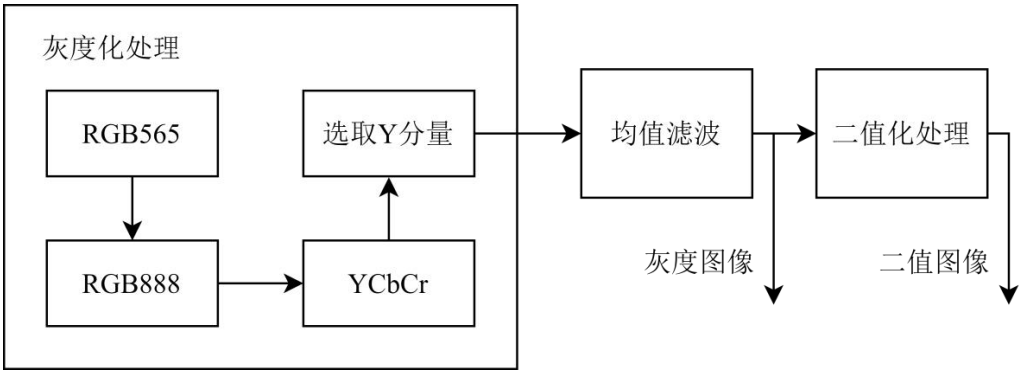

图 2.2 图像预处理流程框图

（1）灰度化处理

大多数视频图像均以三通道的格式进行存储，而对于一些特定场景而言，并

不需要彩色图像所包含的丰富信息。因此，可以将彩色图像转为灰度图像以降低数据处理量，进而减少硬件资源的使用。将彩色图像转为灰度图像的过程被称为图像的灰度化处理。

图像的灰度化处理主要是将彩色图像转化为YCbCr格式，并将Y分量提取出来，作为灰度图像的灰度值。YCbCr是通过有序的三元组来表示的，该三元是由Y(Luminance)、Cb(Chrominance-Blue)和Cr(Chrominance-Red)组成，其中Y表示颜色的明亮度和浓度，而Cb和Cr则分别表示颜色的蓝色浓度偏移量和红色浓度偏移量。人的肉眼对由 YCbCr 色彩空间编码的视频中的Y分量更敏感，而Cb和Cr的微小变化基本不会引起视觉上的不同。根据该原理，对图象进行灰度化处理，在有效降低图像数据量的同时也保证了视觉上损失的效果最小。

本系统使用的摄像头为OV5640，对摄像头的IIC配置使其将采集到的图像数据以RGB565的格式输出。所以在进行灰度化处理之前，先需要将RGB565格式转化为RGB888格式，转化方法如下表2.1：

表2.1 RGB565格式转RGB888

RGB565 (16bit)		RGB888 (24bit)	
[15:11]	{R7, R6, R5, R4, R3}	[23:16]	{R7, R6, R5, R4, R3, R2, R1, R0}
[10:5]	{G7, G6, G5, G4, G3, G2}	[15:8]	{G7, G6, G5, G4, G3, G2, G1, G0}
[4:0]	{B7, B6, B5, B4, B3}	[7:0]	{B7, B6, B5, B4, B3, B2, B1, B0}

再将RGB888格式转化为YCbCr格式，转化方法如下：

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ Cb = -0.172R - 0.339G + 0.511B + 128 \\ Cr = 0.511R - 0.428G - 0.083B + 128 \end{cases} \tag{2.1}$$

本项目使用Matlab完成卷积神经网络模型的训练，使用FPGA完成卷积神经网络验证模型的硬件结构搭建。在Matlab上使用rgb2gray函数实现图像的灰度化，所以上式中所有参数需要和rgb2gray函数中的参数值保持一致。另外，由于在FPGA上使用浮点运算时延高，空间资源占用量大，所以采用乘法运算、加法运算和移位运算代替浮点运算。将上式右端数值上扩大256倍，再右移8位，得到如

下公式：

$$\begin{cases} Y = (77R + 150G + 29B) >> 8 \\ Cb = (-43R + 85G + 128B) >> 8 + 128 \\ Cr = (128R - 107G - 21B) >> 8 + 128 \end{cases} \tag{2.2}$$

提取Y分量作为输出的灰度值，经计算可得，Y的位宽为6.

(2) 均值滤波

在保留图像细节特征的条件下对目标图像的噪声进行抑制，这种方法被称为图像滤波。图像滤波使图像更加平滑，其处理效果将直接影响到后续图像处理和分析的有效性和可靠性。

本项目在考虑滤波效果、资源使用量、主要噪声类型等多方面因素后，选择均值滤波作为图像滤波的方法。均值滤波也称为线性滤波，其采用的主要方法为邻域平均法。线性滤波的基本原理是以目标像素为中心（不包括目标像素）的周围八个像素构成一个滤波模板，用模板中的全体像素的平均值来代替原来像素值。现以中心点(x, y)为均值滤波将要处理的位置，*f*(x, y)表示(x, y)点均值滤波处理前的灰度值，*g*(x, y)表示(x, y)点均值滤波处理后的灰度值，可得：

$$g(x,y) = \frac{\sum_{n=-1}^{1}\sum_{m=-1}^{1} f(x+n, y+m) - f(x,y)}{8} \tag{2.3}$$

(3) 二值化处理

图像的二值化处理，可以使图像呈现明显的黑白效果，凸显有效区域的轮廓结构。本项目在图像灰度化的基础上采用二值化处理，为后续分隔独立元素的图像膨胀、连通域分割做准备。

本项目采用固定阈值法对图像的各像素点进行二值化分割。由于背景在补光条件下显示纯白色，而大多数商品不以纯白色为主色，且纯白色的灰度值大于其他颜色的灰度值，故以一个固定的灰度值作为阈值进行二值化分割，判断一个像素点是否为商品的一部分。通过串口-> PicoRV软核-> 硬件模块的信息传递方式，可以迅速准确地确定合适的灰度阈值。

2.1.2 图像膨胀

图像膨胀是图像形态学处理的一种基本运算。膨胀运算有利于消除噪声，并在图像中连接相邻的元素。图像膨胀是指利用N*N的结构元素，扫描二值图像的每一个像素，使结构元素所覆盖的像素点集做“与”操作，即如果点集的像素值都为1，该像素输出为1。否则该像素输出为0。

考虑到后续连通域分割算法的可行性以及硬件资源的限制，本项目采用的图像膨胀法，不对图像的全部像素点进行处理，因此该模块输出的图像相比原图像小。具体实现过程如下：将整个图像分割成大小为16*16的若干个小部分，对这些小部分依次执行膨胀操作，即如果小部分中有像素值为0的点，那么该部分输出的像素值为0。输出的结果是将整个图像的长宽各缩小到原来的1/16，下图2.3为图像膨胀的示意图。

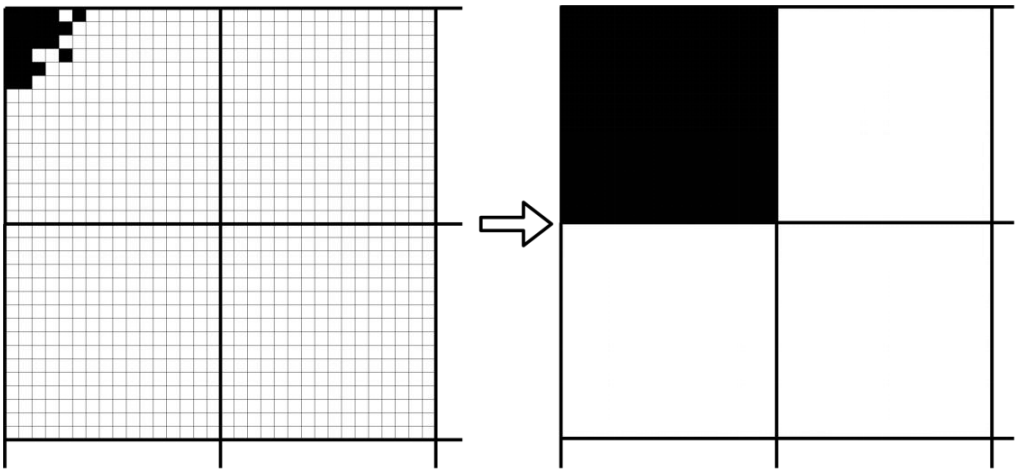

图 2.3 图像膨胀流程框图

2.1.3 连通域分割

连通区域是由具有相同像素值的相邻像素组成像素集合。图像的连通域分割，是将一个二值化图像中的每个白色像素进行标记，属于同一个连通域的白色像素标记相同，不同连通域的白色像素有不同的标记。如此，对于每个连通区域，都具有唯一的标识(Label)。本项目采用4邻接的邻接关系实现图像的连通域分割，连通域分割后最多可以输出8个坐标区间，即是图像中各个商品的空间位置。

嵌入式系统中常用的连通区域分割算法包括基于Seed-Filling的标记算法和

基于Two-Pass的标记算法。基于Two-Pass的分割算法需要对待标记图像遍历两次，并不适用于FPGA图像处理的流水线设计，而基于Seed-Filling的分割算法虽然需要对待标记图像遍历一次，但是使用了递归算法，而且不按照顺序读取像素点。故这两种算法均不可行。

本项目根据FPGA流水线处理图像的特点，设计了一个连通区域分割算法。该算法能顺序遍历整个图像，而且仅遍历一次，保证了连通域分割的速度。该算法在硬件结构方面主要由行寄存器数组、标注序号栈和商品位置寄存器数组三个部分组成。

其中，行寄存器数组用以储存上一行各个像素点的标注序号，行寄存器数组的长度为输入该模块的图像长度。本质上是利用移位寄存器存储输入的图像，当遍历像素点时，可以通过行寄存器数组读出该像素点左侧与上方的标注序号。

其中，标注序号栈用以存放那些暂未标注的序号，可以对其执行数据的弹出(stack_pop)和数据的推入(stack_push)，具体实现逻辑和标准的栈一致。

遍历二值化图像的各像素点，先对若其像素值进行检测，若其像素值为0，则该像素点标注为0；若其像素值为1，将根据情况进行如下的4种处理操作：

(1) 该像素点的左侧像素点标识非0，上方像素点标识为0。则该像素点标识为与左侧像素点标识相同，商品位置寄存器数组相对应位置的X1, X2, Y1, Y2值修改，如下图2.4，连通域分割情况一示意图。

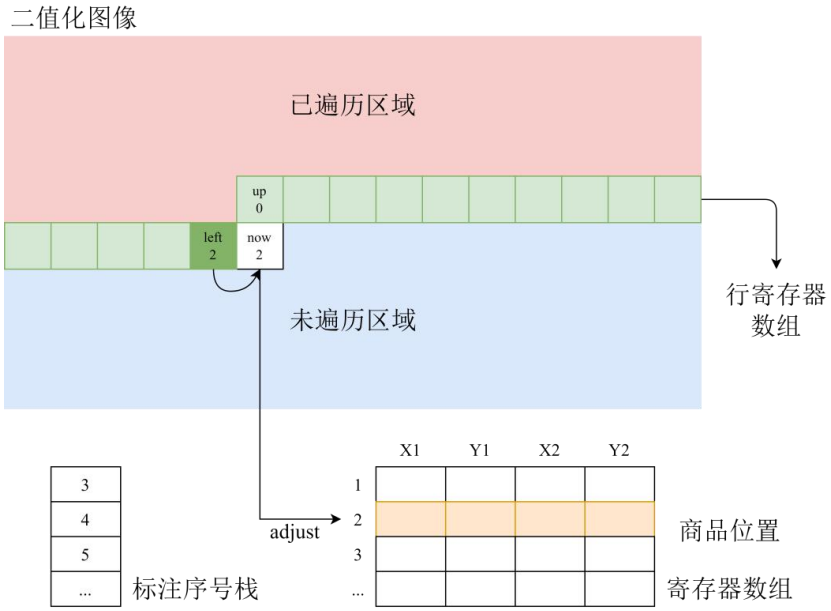

图 2.4 连通域分割情况一示意图

(2) 该像素点的左侧像素点标识为0，上方像素点标识非0，则该像素点标识为与上方像素点标识相同，商品位置寄存器数组相对应位置的X1, X2, Y1, Y2值修改，如下图2.5，连通域分割情况二示意图。

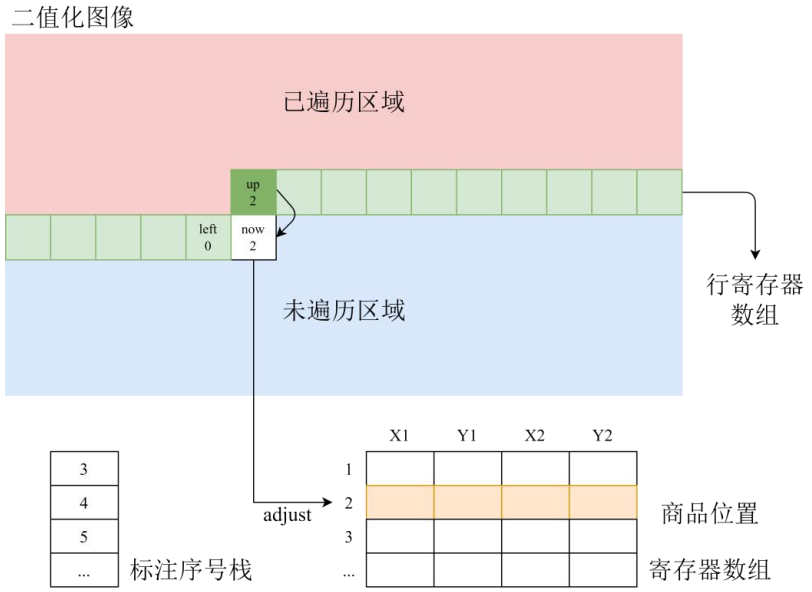

图 2.5 连通域分割情况二示意图

(3) 该像素点的左侧像素点、上方像素点标识均为0，则标注序号栈弹出一个元素，该像素点标识为与弹出的元素值相同，商品位置寄存器数组相对应位置的X1, X2, Y1, Y2值修改，如下图2.6，连通域分割情况三示意图。

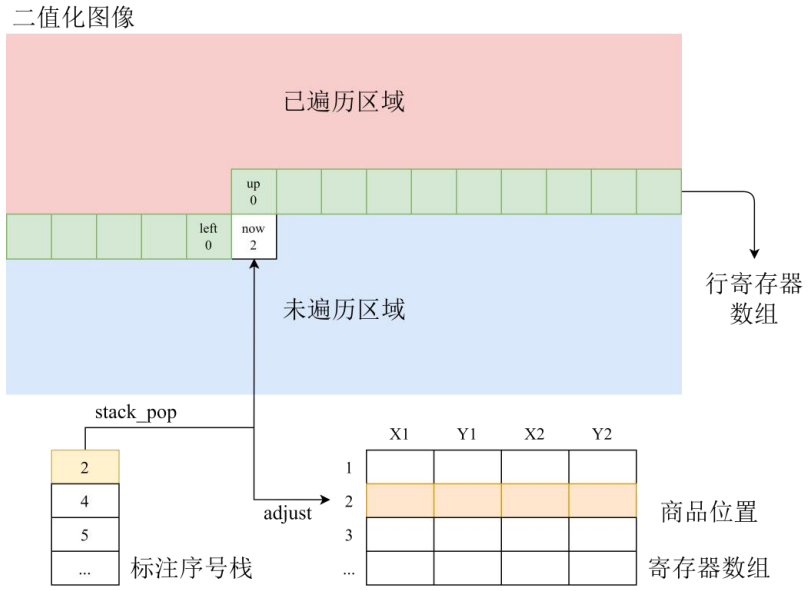

图 2.6 连通域分割情况三示意图

(4) 该像素点的左侧像素点、上方像素点标识均为非0，则标注序号栈推入左侧像素点的标识，商品位置寄存器数组相对应位置的X1, X2, Y1, Y2值初始化；该像素点标识为与上方像素点标识相同，商品位置寄存器数组相对应位置的X1, X2, Y1, Y2值修改，行寄存器数组相对应的标识点进行修改，如下图2.7，连通域分割情况四示意图。

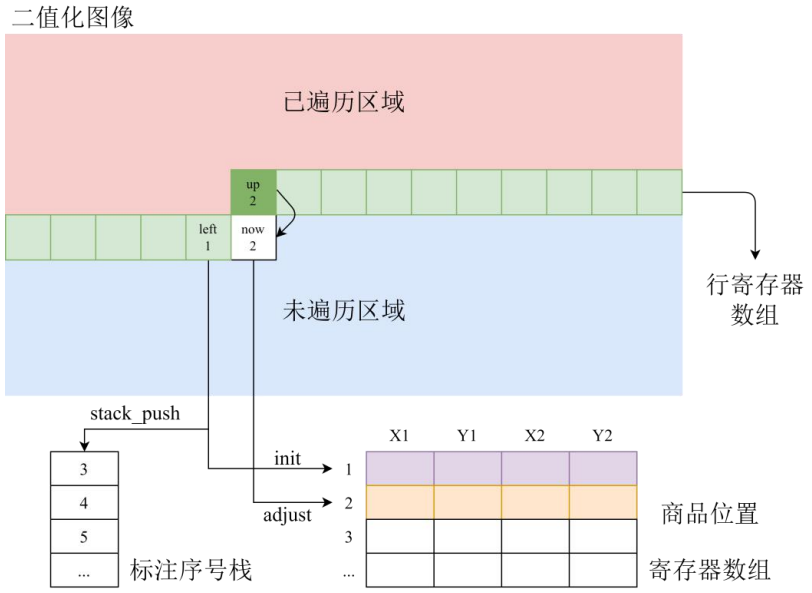

图2.7 连通域分割情况四示意图

2.1.4 卷积神经网络

卷积神经网络(Convolutional Neural Networks, CNN)，现被广泛应用于人脸识别、目标检测、视频图像分类等计算机视觉领域。随着卷积神经网络被不断应用于解决更为复杂的问题，运算的数据量和网络模型的深度随之增加，对计算能力和存储空间也提出了更高的要求。FPGA 具有的高度并行化特点，与卷积神经网络并行化运算特点相契合，利用 FPGA 硬件加速卷积神经网络可以加快物体识别的识别速度。

一个完整的深度学习框架中主要分为训练(Training)、推理(Inference)和部署(deployment)三个流程。训练是初始神经网络通过不断的优化自身参数提高准确率的过程；推理是确定现场数据的准确率的过程；部署则是把训练好的神经网络模型应用在某个硬件平台上使其顺利运行。本项目在 Matlab 上完成卷积神经网络的训练、推理，在 FPGA 上部署卷积神经网络的验证模型。

在综合考虑资源使用量、数据集大小、期望的分类效果等多方面因素，参考优秀的卷积神经网络设计[1][2][3][4][5]后，本项目设计了一种简单易于部署的卷积神经网络结构，如图 2.8。

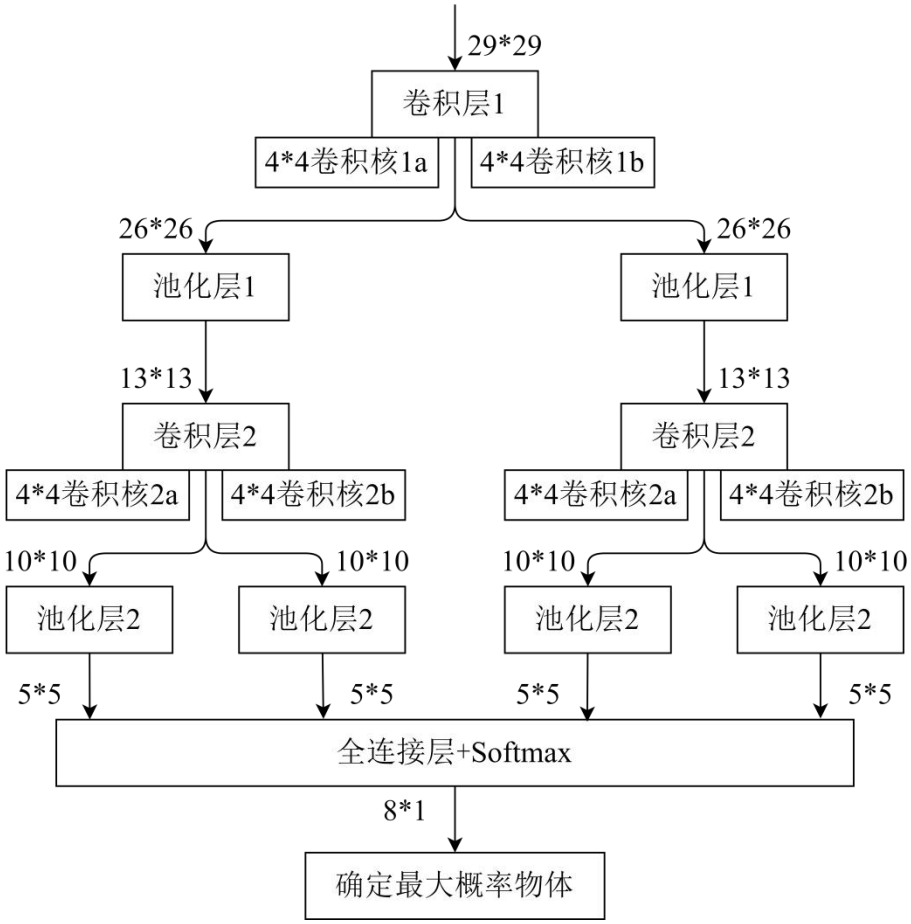

图 2.8 卷积神经网络的整体结构

其中，卷积神经网络的输入是大小为 29*29 的灰度图像，卷积层 1 使用卷积核 1a、卷积核 1b 进行卷积运算，两者大小均为 4*4，故输出的图像数是原来的 2 倍，再通过计算(29-5)/1+1=26，可得：卷积层 1 输出图像的大小是 26*26*2。

池化层 1 输入的是大小为 26*26*2 的图像，池化层 1 中的窗口大小为 2*2，通过计算 26/2=13，可得：最大池化层 1 输出图像的大小是 13*13*2。

卷积层 2 输入的是大小为 13*13*2 的图像，卷积层 2 使用了卷积核 2a、卷积核 2b，两者大小均为 4*4，同卷积层 1，通过计算(13-5)/1+1=10，可得：卷积层 2 输出图像的大小是 10*10*4。

池化层 2 输入的是大小为 10*10*4 的图像，同池化层 1，可计算得：池化层

2 输出图像的大小是 5*5*4。

最后通过全连接层，转换为 8 位宽的 100 个数据，通过矩阵运算，输出 8 个类别的概率，取最大概率的类别并输出该类别的标签。

(1)数据量化

卷积神经网络在 Matlab 上推理的过程中，参与运算的数据都是数值在(-1,1)区间的浮点数，现对所有参与运算的数据进行数据量化处理，使其精度降低。具体操作是：把所有参与运算的数据在数值上乘以 128，并强制转换为 8 位有符号整型数。当进行加法运算时，将 8 位有符号整型数直接相加；进行乘法运算时，将 8 位有符号整型数相乘，并将运算结果除以 128（右移 7 个单位）。在 Matlab 上的测试量化前后的数据输出，结果表明：与浮点数相比，采用 8 位有符号整型数处理完 2 个卷积层和 2 个全连接层后，所得的特征值与原来仍保持高达约 90%的相似度。根据以上的测试与分析，在 FPGA 卷积神经网络的数据处理中也采用 8 位有符号整型数进行计算，在基本保持模型效果不变的情况下，可以节省硬件资源，提高计算速度。

(2)数据存储

卷积神经网路部署到 FPGA 的过程中，需要置入神经网络的各层的各个参数，包括所有卷积核的权重（weight）、偏置（bias），全连接层的权重，在 Matlab 端将这些参数转为 8 位有符号整型数，打印在 Matlab 终端上。其中卷积核的权重、偏置在模块中赋值为 wire 型变量，作为 LUTRAM 的形式储存，保证了执行卷积运算时的速度，全连接层的权重置入 BRAM（block RAM），节省了片内的逻辑资源。

(3)激活函数

神经元的激活函数实现是一个重要的环节，其导数的变化范围影响网络的收敛速度和学习速度。如果使用线性激活函数，那么输入跟输出之间的关系为线性的，无论神经网络有多少层都是线性组合。使用非线性激活函数可以增加神经网

络模型的非线性因素，使网络更加强大。常用的非线性激活函数有 sigmoid，Tanh，ReLU 等。

本项目使用的激活函数是非线性激活函数整流线性单元(Rectified linear unit，ReLU)，其是现代神经网络中最常用的激活函数，大多数前馈神经网络默认使用的激活函数。其收敛速度比 sigmoid、Tanh 快，不会出现梯度饱和、梯度消失的问题，最重要的是其计算复杂度低，在 FPGA 中，不需要采用查找表方式便可以输出函数结果。ReLU 函数定义如下：

$$f(x) = \max(0, x) \tag{2.4}$$

(4)卷积、池化

本系统中，卷积层进行的卷积操作使用的是 4*4 的结构元素，池化层进行的池化操作使用的是 2*2 的结构元素，即在图像上加 4*4 或 2*2 的矩形窗，利用矩形窗遍历整个图像，对矩形窗中的元素进行某种运算后输出结果。卷积神经网络流程中输入的图像数据都是串行的，需要将图像数据进行串并转换后，才能实现这样的“加窗”操作。串并转换根据数据的排序和数量不同，可利用移位寄存器、shift_RAM、FIFO 等实现。本系统寄存器资源尚且充足，所以采用较容易实现的移位寄存器。对输入的串行数据流进行移位处理，提取移位寄存器中的特定单元，便可以完成“加窗”操作。下图 2.9 为 4*4 矩形窗口的生成框图。

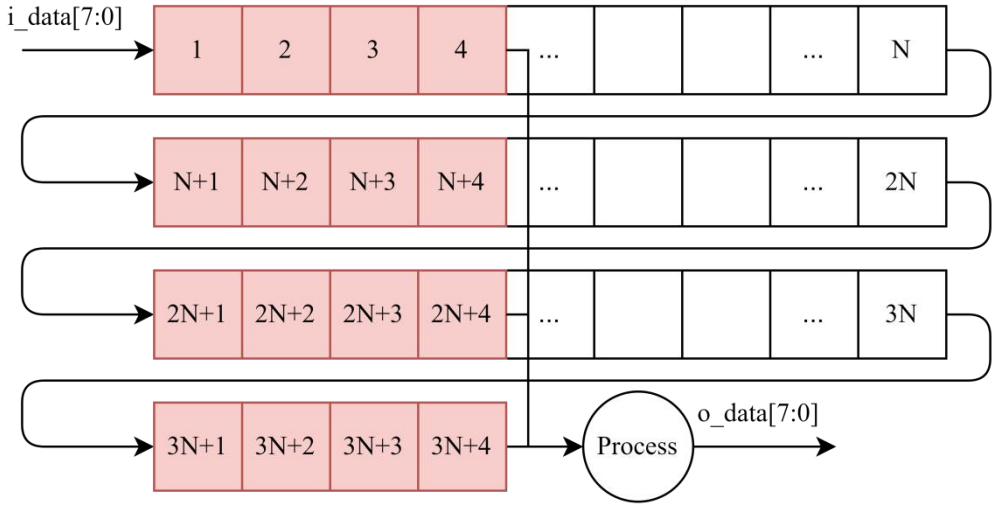

图 2.9 4*4 矩形窗口的生成框图

其中，Process 包括有卷积层的卷积处理和池化层的池化处理，其中卷积处理具体步骤为：将矩形窗口中的像素点值乘以相对应的权重值，相加后再加上偏置，计算结果数值右移，最后经过 ReLU 函数输出，如图 2.10，2.11。池化处理使用的是均值池化，即对矩形窗口中的像素点集取平均值后输出。

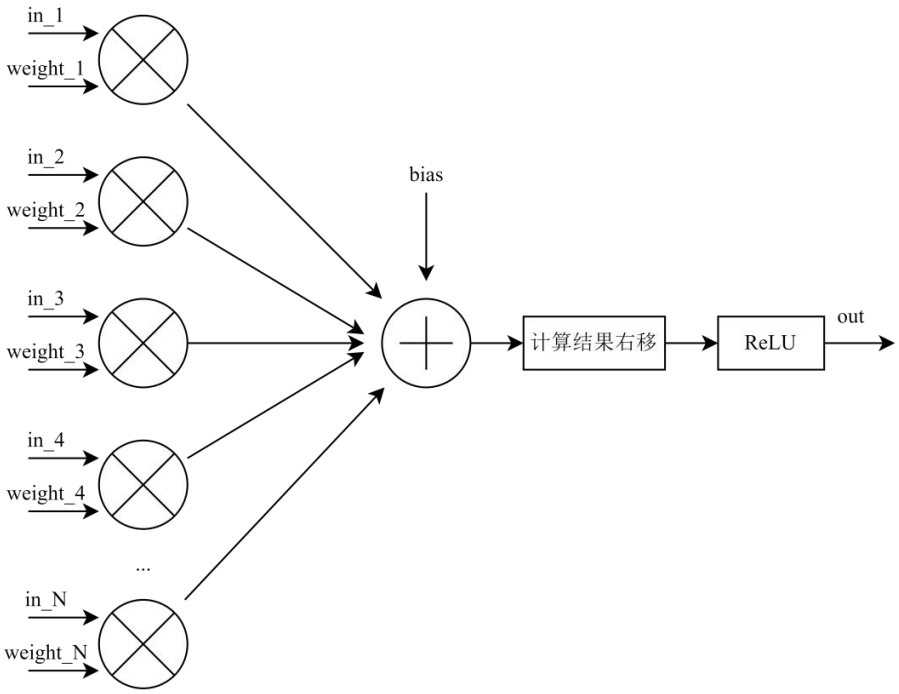

图 2.10 卷积过程示例

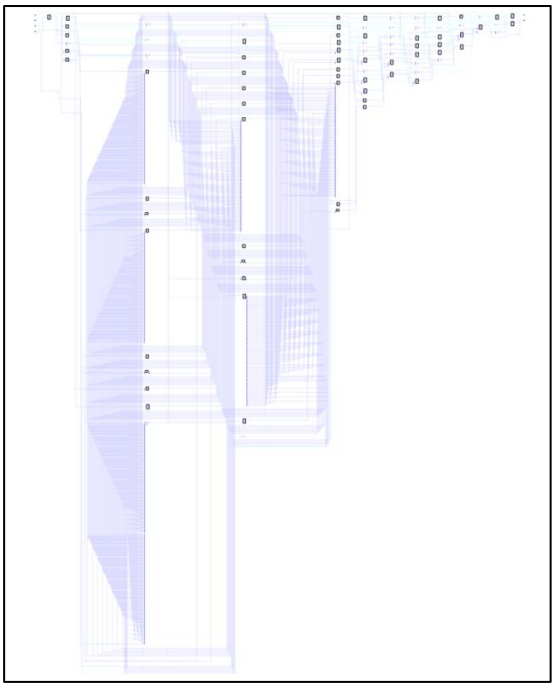

图 2.11 卷积过程 RTL 视图

(5)全连接

全连接层一般由两个部分组成：线性部分和非线性部分。对于线性部分，将输入全连接的数据连接成列向量 x，表示形式如下：

$$x = [x_0, x_1, \ldots x_n]^T, \tag{2.5}$$

若全连接层的输出为：

$$z = [z_0 . z_1, \ldots z_m]^T, \tag{2.6}$$

那么实际上它要转变为一个 m 维的向量，所以可以认为输入的列向量需要乘上一个 m*n 的矩阵，再加上一个偏置：

$$b = [b_0, b_1, \ldots b_n] \tag{2.7}$$

即有全连接层的矩阵运算公式：

$$W * x + b = z \tag{2.8}$$

对于非线性部分，一般是由输出分类器构成，本项目进行卷积神经网络的训练和推理时，使用 SoftMax 分类器，其运算函数如下所示：

$$f_i(x) = \frac{e^{ai}}{\sum_j e^{aj}} \tag{2.9}$$

在 Matlab 上卷积神经网络的训练和推理，其线性部分的输出结果需要经过 SoftMax 分类器，保证输出的所有类别概率和为 1。但 SoftMax 函数是一个单调递增的函数，即使不经过这个函数，各概率值之间的大小关系也不会变化。在 FPGA 上避免求指运算，直接比较线性部分输出的各个结果值，也可以得到最大概率下的商品种类。跳过非线性部分，可以简化运算步骤，但是无法输出正确的概率值。

2.1.5 卷积神经网络和传统图像处理相融合

本项目采取了卷积神经网络和传统图像处理相融合的物体分类方法。单独使用传统图像处理方法进行分类时，无法对复杂颜色的商品进行分类。单独使用卷积神经网络进行分类时，分类效果可能不稳定。本项目融合了卷积神经网络的分

类输出和传统图像处理的分类输出，具体融合的过程如下图 2.12.

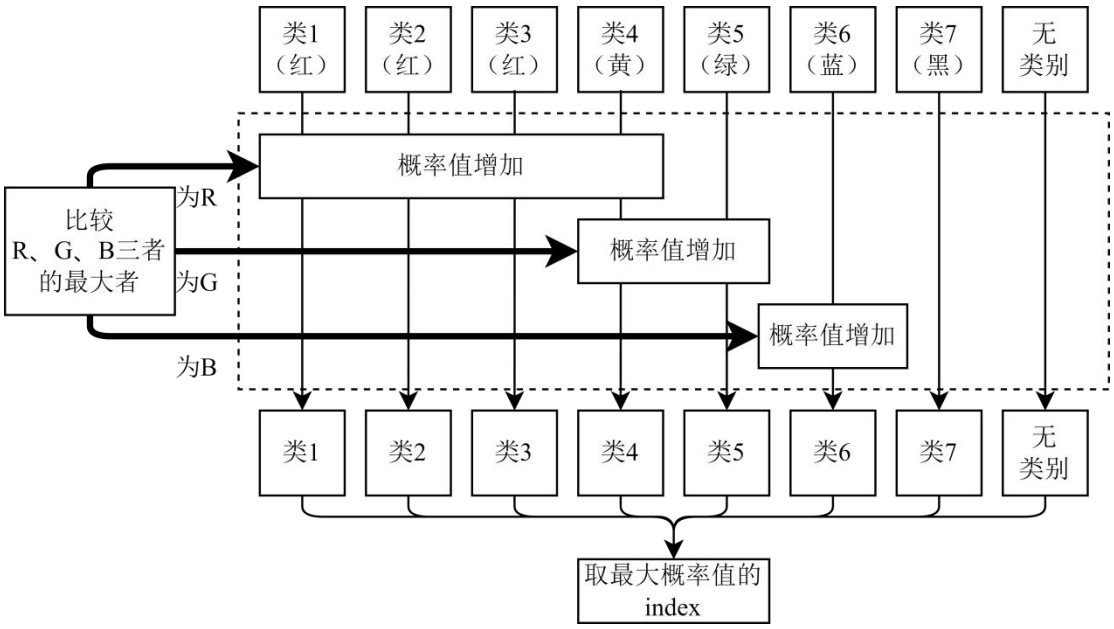

图 2.12 卷积神经网络和传统图像处理相融合的物体分类

利用卷积神经网络输出各种类的概率值后，不直接对其进行概率的比较。先比较该部分图像的 R、G、B 三者的最大者，如果 R 为最大者，则对以红色调或黄色调为主色调的商品种类所对应的概率进行增强；如果 G 为最大者，则对以绿色调或黄色调为主色调的商品种类所对应的概率进行增强；如果 B 为最大者，则对以蓝色调为主色调的商品种类所对应的概率进行增强。对增强后的各种类的概率值进行比较，得到最大概率值下的商品种类，并输出该种类对应的商品标签。

2.2 软件部分

本项目的软件部分，主要被划分为PC机Matlab端以及FPGA的PicoRV软核端。其中，Matlab端主要实现了卷积神经网络的训练、推理，模型参数的导出。PicoRV软核端主要负责FPGA的系统控制。

2.2.1 Matlab 端神经网络的训练、推理

使用vott软件标记原始图像，导出.csv文件。Matlab读取.csv文件，用标记的坐标截取原始图像，生成各类型的图像片段，以此作为训练集并开始卷积神经网络的训练，训练所用的函数为Matlab内置的函数cnnsetup、 cnntrain，利用cnntest

函数得到loss曲线，若loss曲线效果较差，需要调整学习率和迭代次数重新学习。

2.2.2 PicoRV 软核端

PicoRV软核端负责系统的控制功能，包括串口指令接收与图片切换控制、售货流程的状态机、卷积神经网络识别类型的接收和串口发送。

其中，售货流程的状态机是这个部分的核心，其包括商品数量、商品价格的计算，售货机状态的转换，售货流程具体如下图2.13。

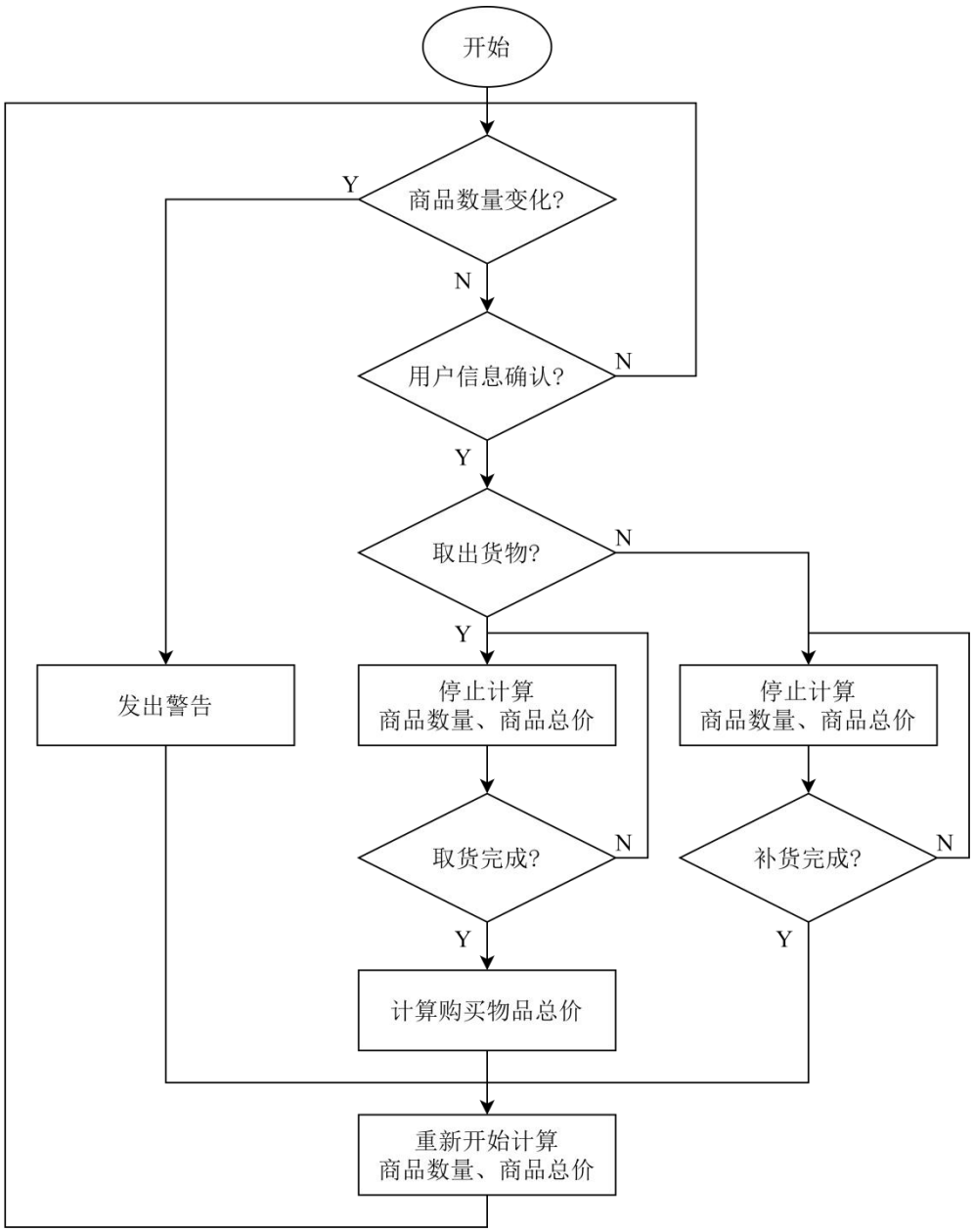

图 2.13 售货状态下的流程图

程序开始后，将对所有识别到的商品进行商品数量和商品总价的计算。在用户信息确认前，如果检测到商品数量的变化，将会发出警告并在HDMI屏上显示警告标识。用户信息确认后，如果该用户是商家，则商家可以对智能售货机系统进行补货，在此期间，将停止对商品数量和商品总价的计算；如果该用户是消费者，则消费者可以对智能售货机系统进行取货，在此期间，将停止对商品数量和商品总价的计算；在取货完成后，根据商品价格的变化情况计算该消费者购买商品的总价，并且将其显示。

第三章 完成情况及性能参数

3.1 软件部分功能测试

3.1.1 Matlab 端卷积神经网络的测试

该部分是对卷积神经网络模型的训练、推理、部署方面的测试。将训练集导入Matlab后，选择合适的学习率和迭代次数，开始卷积神经网络的训练。通过不断实际测试可得，此模型最佳的学习率为0.01，最佳迭代次数为1000。卷积神经网络训练完成后，得到loss曲线模型训练完成，如图3.1 MATLAB卷积神经网络的训练测试。

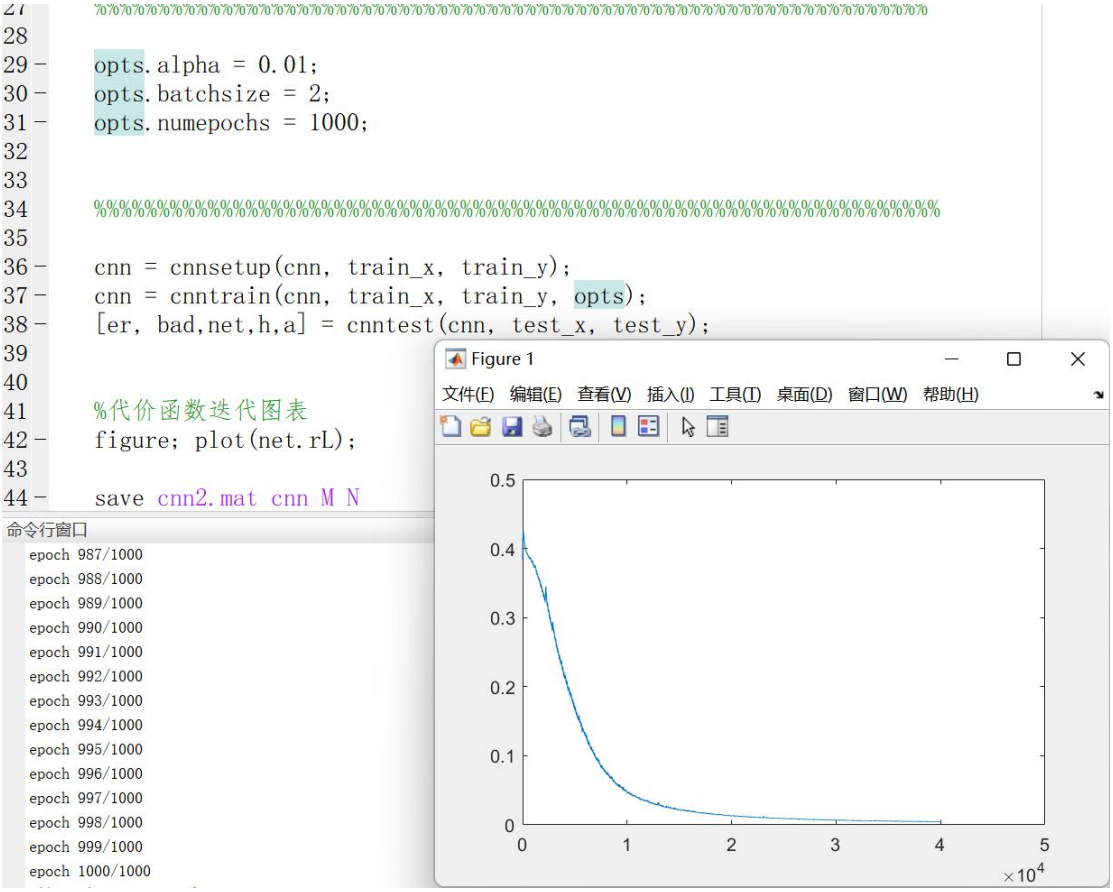

图 3.1 MATLAB 卷积神经网络的训练测试

对训练的卷积神经网络模型进行推理步骤，输入80张测试图片，测试结果为正确匹配率为100%，即80张测试图片都成功分类。输出的最大概率值稳定在

0.83~0.99间，如图3.2 MATLAB卷积神经网络的推理测试。

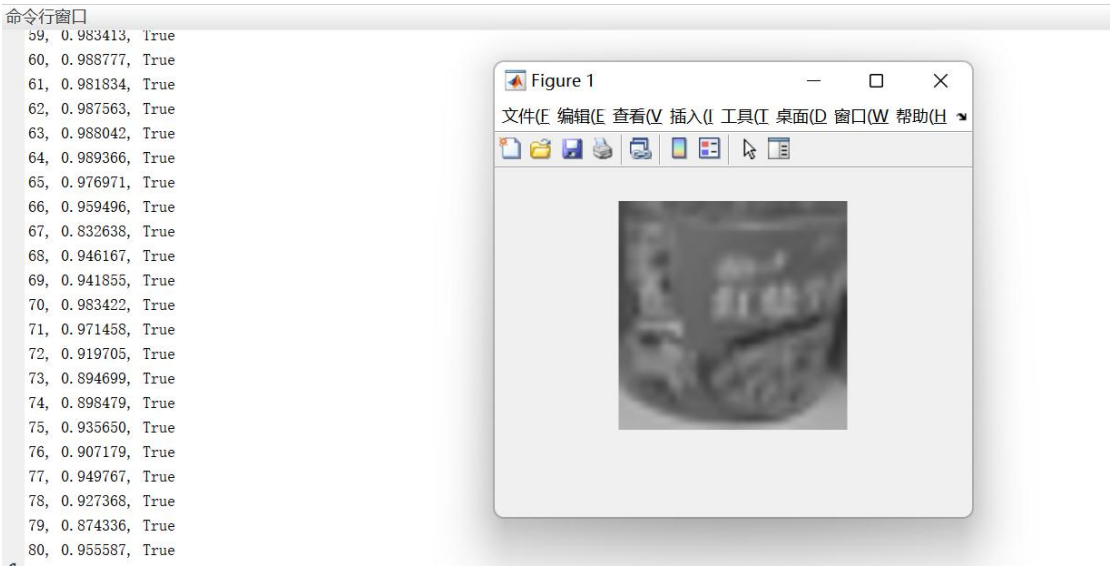

图 3.2 MATLAB 卷积神经网络的推理测试

将神经网络的各个参数，如卷积核的权重（weight）、偏置（bias）转化成 8 位有符号整型数并连接在一起，并打印出来。复制参数，赋值给 FPGA 中卷积模块的 weight、bias 常量(parameter)；将全连接层的权重转化成 8 位有符号整型数并连接在一起，按照文件规范生成.mif 文件，使全连接层的权重存储在 BRAM 中，完成卷积神经网络的部署，如图 3.3 卷积神经网络的部署测试。

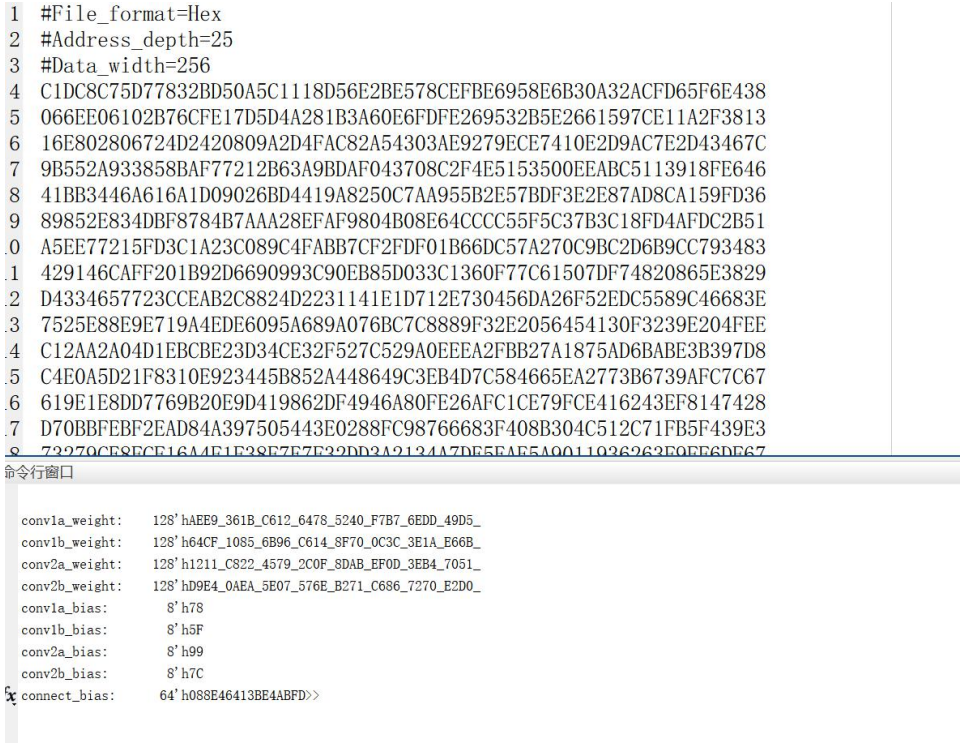

图 3.3 卷积神经网络的部署测试

3.1.2 PicoRV 软核端

测试 PicoRV 和单片机串口通信，测试运行过程的参数和运行结果的输出，运行参数包括相机帧率、显示帧率、识别速率、商品总数、商品总价等，具体测试结果如下图 3.4PicoRV 的测试，测得 PicoRV 运行正常，各参数输出正确。

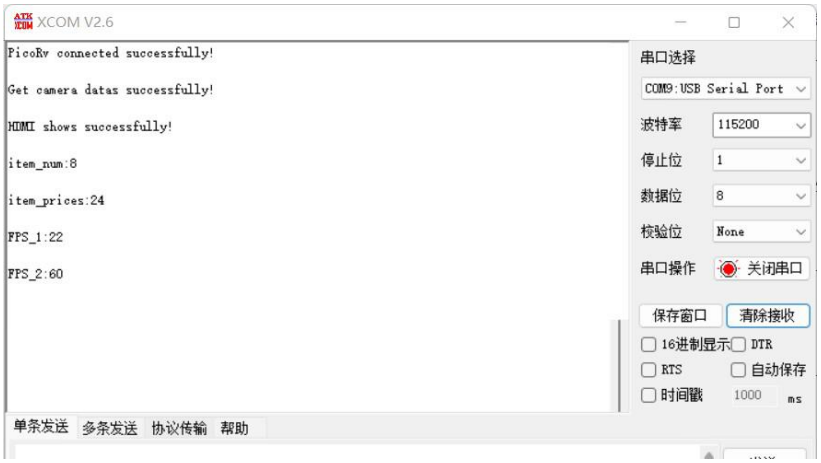

图 3.4PicoRV 的测试

3.2 总体功能测试

该部分是对系统的整体功能进行测试。运行该系统，HDMI 屏清晰地显示售货架的画面。相机帧率、显示帧率、识别速率成功显示，物体识别正常，商品总数，商品总价计算正确，如图 3.5 系统的整体功能测试。

图 3.5 系统的整体功能测试

下面对系统的取货功能进行测试。输入消费者1的身份信息，点击按下取货后，HDMI屏显示取货的图标，消费者可以开始选购商品。从售货机中取出若干件商品后，消费者点击取货完成并退出。HDMI屏上可以显示出用户信息与购买商品的总金额，并显示结算的图标，如图3.6系统的取货功能测试。

图 3.6 系统的取货功能测试

下面对系统的鲁棒性进行测试，如果没有检测到用户登录而商品数量发生变化，系统会显示警告，HDMI屏的右下角显示警告图标，如图3.7系统的鲁棒性测试。

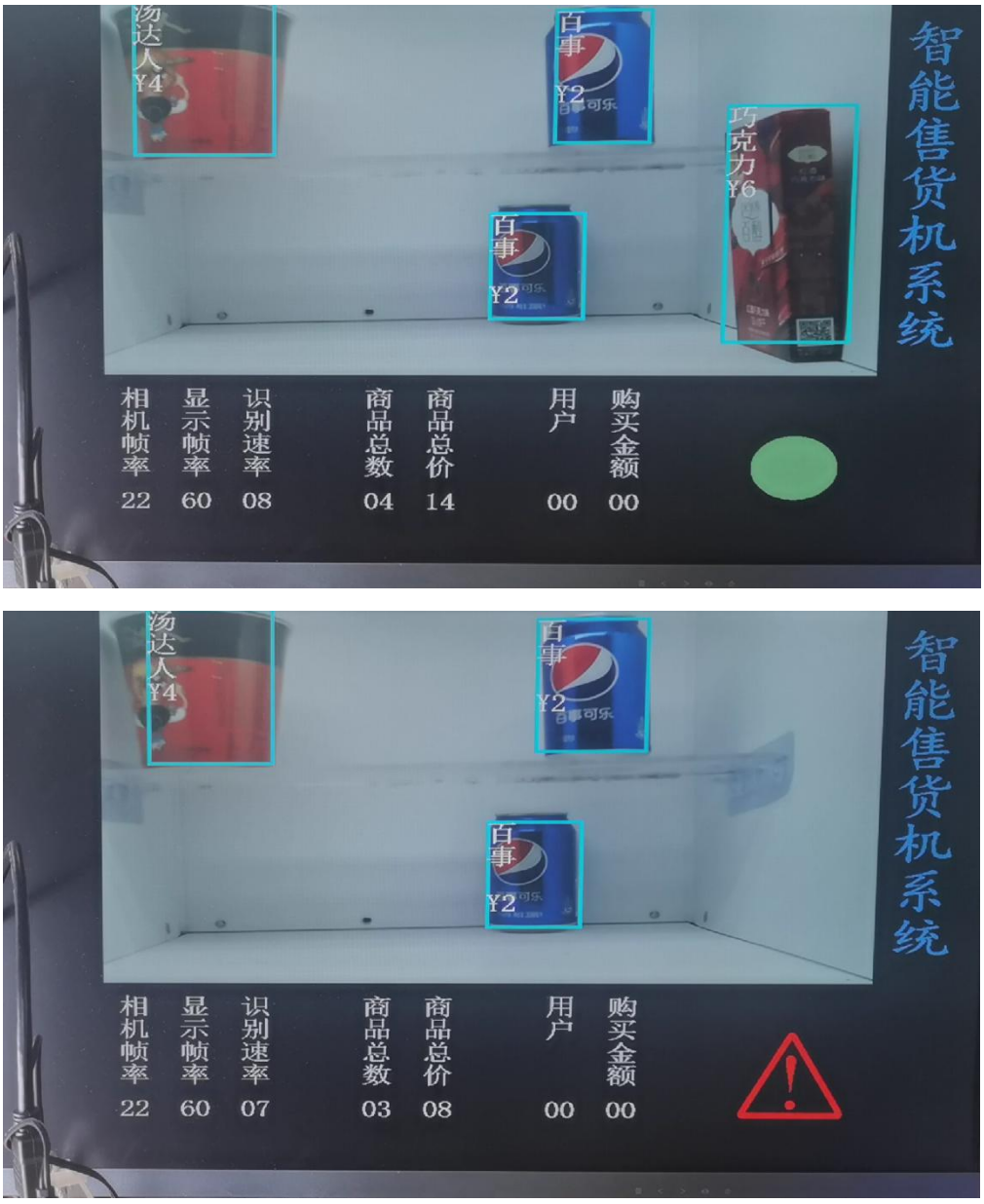

图 3.7 系统的鲁棒性测试

此外，系统还内置图片切换功能，可以对每一步的图像处理过程进行观察，方便调试。首先是图像的二值化处理，二值化处理的成功与否将直接影响商品和背景的分割，进而影响各个商品的精确定位，二值化处理的结果如图 3.8 图像的二值化处理结果显示。

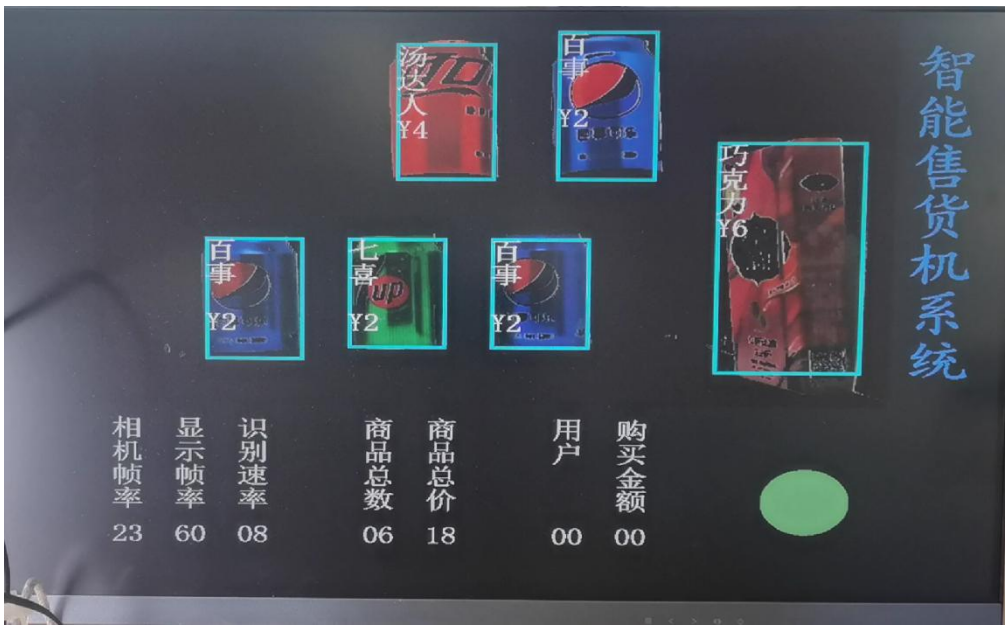

图 3.8 图像的二值化处理结果显示

其次是图像膨胀，通过图像的膨胀能够消除部分噪声，使商品的定位更加准确，如图 3.9 图像的膨胀处理结果显示。

图 3.9 图像的膨胀处理结果显示

最后是图像流进入卷积神经网络层的区域提取过程，具体操作是分别对各个商品所处的图像部分进行取点操作，如图 3.10 图像的卷积层前取点结果，对不同大小的商品，取点的密度也有不同，对于小型商品取点较为密，对于大型商品取点较为疏，每块取点的点集大小都是 29*29。

图 3.10 图像的卷积层前取点结果

第四章 总结

4.1 可扩展之处

本项目是智能售货机系统，可以进行8个物体的定位和分类，进行8种物体的识别（包括无类别）。在实际的应用中，售货机的售货量是非常大的，售货的类别繁多，所以本项目可以在物品类别和物品数量上做扩展，实现更多类型、更多数量的商品识别和检测。

在增加可识别种类数量的同时，也可以增加类别专门用以区分物体是实物还是纸张上的照片，增强系统的鲁棒性。

4.2 心得体会

这是我们小组第一次参加 FPGA 竞赛，对于陌生的开发环境，我们适应的较快，其中高云 IDE 充足的参考文档和例程给了我们很大的帮助。主要的难点在于合适的卷积神经网络模型的选取和算法的设计。在综合考虑资源使用量和预期的识别效果等方面，我们使用的是简单小巧的神经网络模型达到物体的分类效果。题目所要求的物体识别，我们考虑用以联通与分割算法为主的图像处理法来完成。过程中还是遇到了许多困难，比如资源使用量不足，黑屏闪屏花屏的 bug 无法定位，不过在不断跑逻辑分析仪和调整代码中，bug 一一解决，资源使用也基本到达极限。

第五章 参考文献

[1] 姜懿家. 卷积神经网络的压缩研究[J].电子制作, 2019, 2.

[2] Zhang C, Li P, Sun G, et al. Optimizing fpga-based accelerator design for deep convolutional neural networks[C]//Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays. 2015: 161-170.

[3] 牛鑫. 基于 FPGA 的播种机自主导航控制系统设计[J].农机化研究, 2022, 44(6):98-101, 106.

[4] 陈畅,黄均才,刘鉴栋,等. 卷积神经网络定点化设计及 FPGA 实现[J]. 单片机与嵌入式系统应用,2022,22(2):41-45.

[5] 张太然,朱建国,宋玉贵,等. 高速动态图像目标识别算法的设计与实现[J]. 计算机测量与控制,2022,30(2):229-236.胡铁乔,赵冬辉. 基于 ZYNQ 车载 GPS 接收机捕获模块的设计与实现[J]. 中国民航大学学报,2022,40(1):27-33,46.

第六部分　附录

6.1 核心算法代码（连通域分割，卷积层）

```
`include "../define.v"

module div_rect
#(
    parameter                           C0_W  = `CORROSION_WIDTH   ,
    parameter                           C_W = C0_W+C0_W+C0_W+C0_W  ,

    parameter                           R_W  = `RECT_NUMMAX_WIDTH  ,
    parameter                           RR_W = `RECT_NUMMAX        ,

    parameter                           V_L  = `CORROSION_DX
)
(
    input  wire                         sys_clk                    ,
    input  wire                         sys_rst_n                  ,
    input  wire                         item_rst_n                 ,

    input  wire       [   7:0]          i_smax                     ,

    input  wire                         i_valid                    ,
    input  wire                         i_wb                       ,

    output reg                          o_finish                   ,
    output reg        [`RECT_NUMMAX * 32 - 1 : 0] o_item
);
reg                   [`RECT_NUMMAX * 32 - 1 : 0] item_cor        ;//C_W <= 32

//----- stack
reg                   [R_W-1:0]         stack_index  [0:RR_W-1]    ;
reg                                     stack_pop                  ;
reg                                     stack_push                 ;
reg                   [R_W-1:0]         stack_data                 ;
integer i;
always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0) begin
        for(i = 0; i < RR_W; i = i + 1)
```

```
            stack_index[i] <= i + 1;
    end
    else if(stack_pop == 1'b1) begin
        for(i = 0; i < RR_W - 1; i = i + 1)
            stack_index[i] <= stack_index[i+1];
    end
    else if(stack_push == 1'b1) begin
        for(i = 0; i < RR_W - 1; i = i + 1)
            stack_index[i+1] <= stack_index[i];
        stack_index[0] <= stack_data;
    end


//----- cnt
reg                     [   3:0]        cnt_clk                     ;
always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0)
        cnt_clk <= 'b0;
    else if(i_valid == 1'b1)
        cnt_clk <= 'b1;
    else if(cnt_clk != 'b0)
        cnt_clk <= cnt_clk + 'b1;


reg                     [C0_W-1:0]      cnt_x                       ;
reg                     [C0_W-1:0]      cnt_y                       ;
reg                     [   7:0]        cnt_finish                  ;
always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0)begin
        cnt_x <= 'b0;
        cnt_y <= 'b0;
        cnt_finish <= 'b0;
        o_finish <= 'b0;
    end
    else if(cnt_clk == 'd5)
        if(cnt_x == `CORROSION_DX - 1) begin
            cnt_x <= 'b0;
            if(cnt_y == `CORROSION_DY - 1) begin
                cnt_y <= 'b0;
                cnt_finish <= 1'b1;
            end
            else
                cnt_y <= cnt_y + 1'b1;
        end
        else
```

```
            cnt_x <= cnt_x + 1'b1;

    else if(cnt_finish == 'b0) begin
        cnt_finish <= cnt_finish;
        o_finish <= 'b0;
    end
    else if(cnt_finish == `RECT_NUMMAX) begin
        cnt_finish <= 'b0;
        o_finish <= 'b1;
    end
    else begin
        cnt_finish <= cnt_finish + 'b1;
        o_finish <= 'b0;
    end


//----- process
reg                 [   5:0]        state                   ;
localparam                          BLACK = 6'b000_001      ;
localparam                          WHITE = 6'b000_010      ;
localparam                          NONE  = 6'b000_100      ;
localparam                          UP    = 6'b001_000      ;
localparam                          LEFT  = 6'b010_000      ;
localparam                          BOTH  = 6'b100_000      ;

reg                 [R_W-1:0]       last_horizen   [V_L-1:0]  ;
reg                 [R_W-1:0]       left                    ;
reg                 [R_W-1:0]       up                      ;

integer j;
always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0) begin
        item_cor   <= 'b0;
        stack_pop  <= 'b0;
        stack_push <= 'b0;
        stack_data <= 'b0;
        state      <= BLACK;
        for(j = 0; j < V_L; j = j + 1)
            last_horizen[j] <= 'b0;
        left <= 'b0;
        up   <= 'b0;
    end
    else if(i_valid == 1'b1) begin
```

```
    if(i_wb == 1'b1)
        state <= WHITE;
    else
        state <= BLACK;
    up <= last_horizen[V_L-1];
    if(cnt_x == 0)
        left <= 'b0;
    else
        left <= last_horizen[0];
end
else if(cnt_clk == 'd1 && state == WHITE) begin
    if     (up == 'b0 && left == 'b0)
        state      <= NONE;
    else if(up != 'b0 && left == 'b0)
        state      <= UP;
    else if(up == 'b0 && left != 'b0)
        state      <= LEFT;
    else if(up != 'b0 && left != 'b0 && up == left)
        state      <= UP;
    else if(up != 'b0 && left != 'b0 && up != left)
        state      <= BOTH;
end


else if(state == UP) begin
    if(cnt_clk == 'd2) begin
        if(cnt_x < item_cor[{up,   5'b0} + 8 + 8 + 8 +: C0_W])
                   item_cor[{up,   5'b0} + 8 + 8 + 8 +: C0_W] <= cnt_x;
        if(cnt_y < item_cor[{up,   5'b0}     + 8 + 8 +: C0_W])
                   item_cor[{up,   5'b0}     + 8 + 8 +: C0_W] <= cnt_y;
        if(cnt_x > item_cor[{up,   5'b0}         + 8 +: C0_W])
                   item_cor[{up,   5'b0}         + 8 +: C0_W] <= cnt_x;
        if(cnt_y > item_cor[{up,   5'b0}             +: C0_W])
                   item_cor[{up,   5'b0}             +: C0_W] <= cnt_y;
        for(j = 1; j < V_L; j = j + 1)
            last_horizen[j] <= last_horizen[j - 1];
        last_horizen[0] <= up;
    end
end


else if(state == LEFT) begin
    if(cnt_clk == 'd2) begin
        if(cnt_x < item_cor[{left, 5'b0} + 8 + 8 + 8 +: C0_W])
                   item_cor[{left, 5'b0} + 8 + 8 + 8 +: C0_W] <= cnt_x;
```

```
            if(cnt_y < item_cor[{left, 5'b0}     + 8 + 8 +: C0_W])
                      item_cor[{left, 5'b0}     + 8 + 8 +: C0_W] <= cnt_y;
            if(cnt_x > item_cor[{left, 5'b0}         + 8 +: C0_W])
                      item_cor[{left, 5'b0}         + 8 +: C0_W] <= cnt_x;
            if(cnt_y > item_cor[{left, 5'b0}             +: C0_W])
                      item_cor[{left, 5'b0}             +: C0_W] <= cnt_y;
            for(j = 1; j < V_L; j = j + 1)
                last_horizen[j] <= last_horizen[j - 1];
            last_horizen[0] <= left;
        end
    end


    else if(state == NONE) begin
        if(cnt_clk == 'd2) begin
            stack_pop <= 'b1;
            item_cor[{stack_index[0], 5'b0} + 8 + 8 + 8 +: C0_W] <= cnt_x;
            item_cor[{stack_index[0], 5'b0}     + 8 + 8 +: C0_W] <= cnt_y;
            item_cor[{stack_index[0], 5'b0}         + 8 +: C0_W] <= cnt_x;
            item_cor[{stack_index[0], 5'b0}             +: C0_W] <= cnt_y;
            for(j = 1; j < V_L; j = j + 1)
                last_horizen[j] <= last_horizen[j - 1];
            last_horizen[0] <= stack_index[0];
        end
        else if(cnt_clk == 'd3) begin
            stack_pop <= 'b0;
        end
    end


    else if(state == BOTH) begin
        if(cnt_clk == 'd2) begin
            stack_push <= 'b1;
            stack_data <= left;
            if(item_cor[{up,   5'b0} + 8 + 8 + 8 +: C0_W] >  item_cor[{left, 5'b0}
+ 8 + 8 + 8 +: C0_W])
               item_cor[{up,   5'b0} + 8 + 8 + 8 +: C0_W] <= item_cor[{left, 5'b0}
+ 8 + 8 + 8 +: C0_W];
            if(item_cor[{up,   5'b0}     + 8 + 8 +: C0_W] >  item_cor[{left,
5'b0}     + 8 + 8 +: C0_W])
               item_cor[{up,   5'b0}     + 8 + 8 +: C0_W] <= item_cor[{left,
5'b0}     + 8 + 8 +: C0_W];
            if(item_cor[{up,   5'b0}         + 8 +: C0_W] <  item_cor[{left,
5'b0}         + 8 +: C0_W])
```

```
                item_cor[{up,   5'b0}        + 8 +: C0_W] <= item_cor[{left,
5'b0}        + 8 +: C0_W];
            if(item_cor[{up,   5'b0}            +: C0_W] <  item_cor[{left,
5'b0}            +: C0_W])
                item_cor[{up,   5'b0}            +: C0_W] <= item_cor[{left,
5'b0}            +: C0_W];
            for(j = 1; j < V_L; j = j + 1)
                last_horizen[j] <= last_horizen[j - 1];
            last_horizen[0] <= up;
        end
        else if(cnt_clk == 'd3) begin
            stack_push <= 'b0;
            item_cor[{left,          5'b0} + 8 + 8 + 8 +: C0_W] <= 'b0;
            item_cor[{left,          5'b0}     + 8 + 8 +: C0_W] <= 'b0;
            item_cor[{left,          5'b0}         + 8 +: C0_W] <= 'b0;
            item_cor[{left,          5'b0}             +: C0_W] <= 'b0;
            for(j = 0; j < V_L; j = j + 1)
                if(last_horizen[j] == left)
                    last_horizen[j] <= up;
        end
    end


    else if(state == BLACK) begin
        if(cnt_clk == 'd2) begin
            for(j = 1; j < V_L; j = j + 1)
                last_horizen[j] <= last_horizen[j - 1];
            last_horizen[0] <= 'b0;
        end
    end



//-----
//CORROSION x1 坐标 转 LETTER_WRITE x1 坐标
//out = in * CORROSION_SIZE / LETTER_PIXEL_SIZE + PIC_X1 / LETTER_PIXEL_SIZE;
//CORROSION y1 坐标 转 LETTER_WRITE y1 坐标
//out = in * CORROSION_SIZE / LETTER_PIXEL_SIZE;
//CORROSION x2 坐标 转 LETTER_WRITE x2 坐标
//out = (in + 1) * CORROSION_SIZE / LETTER_PIXEL_SIZE + PIC_X1 /
LETTER_PIXEL_SIZE;
//CORROSION y2 坐标 转 LETTER_WRITE y2 坐标
//out = (in + 1) * CORROSION_SIZE / LETTER_PIXEL_SIZE;


// out = in * (4) + 28;
```

```
// out = in * (4) + 00;
// out = in * (4) + 32;
// out = in * (4) + 04;


always@(posedge sys_clk or negedge item_rst_n)
    if(item_rst_n == 1'b0) begin
        o_item <= 'b0;
    end
    else if(cnt_finish != 0 && cnt_finish != `RECT_NUMMAX) begin
        if( item_cor[{cnt_finish, 5'b0}         + 8 +:
8]  -  item_cor[{cnt_finish, 5'b0} + 8 + 8 + 8 +: 8] < i_smax ||
            item_cor[{cnt_finish, 5'b0}             +:
8]  -  item_cor[{cnt_finish, 5'b0}     + 8 + 8 +: 8] < i_smax)
            o_item  [{cnt_finish, 5'b0} +: 32] <= 'b0;
        else begin
            o_item  [{cnt_finish, 5'b0} + 8 + 8 + 8 +: 8] <= {item_cor[{cnt_finish,
5'b0} + 8 + 8 + 8 +: 6], 2'b0} + 'd28;
            o_item  [{cnt_finish, 5'b0}     + 8 + 8 +: 8] <=
{item_cor[{cnt_finish, 5'b0}     + 8 + 8 +: 6], 2'b0} + 'd00;
            o_item  [{cnt_finish, 5'b0}         + 8 +: 8] <=
{item_cor[{cnt_finish, 5'b0}         + 8 +: 6], 2'b0} + 'd32;
            o_item  [{cnt_finish, 5'b0}             +: 8] <=
{item_cor[{cnt_finish, 5'b0}             +: 6], 2'b0} + 'd04;
        end
    end



endmodule


module conv_1
#(
    parameter                       weight = 128'h0,
    parameter                       bias   = 8'h0
)
(
    input  wire                     sys_clk                 ,
    input  wire                     sys_rst_n               ,

    input  wire                     i_valid                 ,
    input  wire       [   7:0]      i_data                  ,
    output reg                      o_valid                 ,
    output wire       [   7:0]      o_data
);
```

```
localparam                              ROW_LEN  = 29             ;
localparam                              CORE_LEN = 4              ;
localparam                              TMP_SIZE = ROW_LEN + ROW_LEN + ROW_LEN
+ CORE_LEN;


//----- valid_cnt
reg                     [   2:0]        valid_cnt                 ;
always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0)
        valid_cnt <= 'd0;
    else if(i_valid == 1'b1)
        valid_cnt <= 'd1;
    else if(valid_cnt == 'd0 || valid_cnt == 'd6)
        valid_cnt <= 'd0;
    else
        valid_cnt <= valid_cnt + 'b1;


//----- cnt
reg                     [   7:0]        cnt_x_0                   ;
reg                     [   7:0]        cnt_y_0                   ;
wire                                    process                   ;
always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0) begin
        cnt_x_0 <= 'b0;
        cnt_y_0 <= 'b0;
    end
    else if(i_valid == 1'b1)
        if(cnt_x_0 == ROW_LEN - 1) begin
            cnt_x_0 <= 'b0;
            if(cnt_y_0 == ROW_LEN - 1)
                cnt_y_0 <= 'b0;
            else
                cnt_y_0 <= cnt_y_0 + 1'b1;
        end
        else
            cnt_x_0 <= cnt_x_0 + 1'b1;
assign process = (cnt_y_0 > CORE_LEN - 1 || (cnt_y_0 == CORE_LEN - 1 && cnt_x_0 >
CORE_LEN - 1)) ? 'b1 : 'b0;


//----- tmp
integer i;
reg signed              [   9:0]        tmp      [0:TMP_SIZE-1]    ;
always@(posedge sys_clk or negedge sys_rst_n)
```

```
    if(sys_rst_n == 1'b0)
        for(i = 0; i < TMP_SIZE; i = i + 1)
            tmp[i] <= 'd0;
    else if(i_valid == 1'b1) begin
        for(i = 0; i < TMP_SIZE - 1; i = i + 1)
            tmp[i + 1] <= tmp[i];
        tmp[0] <= {2'b0, i_data};
    end


reg             signed [  19:0]         tmp_1                   ;
reg             signed [  19:0]         tmp_2                   ;
reg             signed [  19:0]         tmp_3                   ;
reg             signed [  19:0]         tmp_4                   ;
reg                                    valid_1                 ;


reg             signed [   7:0]         tmp_1_in1               ;
reg             signed [   7:0]         tmp_1_in2               ;
wire            signed [  15:0]         tmp_1_out               ;
wire            signed [  19:0]         tmp_1_out_              ;
assign tmp_1_out_ = (tmp_1_out[15] == 1'b1) ? {4'b1, tmp_1_out} : {4'b0,
tmp_1_out};
signed_mul_8 u1_signed_mul_8
(
    .dout                           (tmp_1_out              ),
    .a                              (tmp_1_in1              ),
    .b                              (tmp_1_in2              )
);
reg             signed [   7:0]         tmp_2_in1               ;
reg             signed [   7:0]         tmp_2_in2               ;
wire            signed [  15:0]         tmp_2_out               ;
wire            signed [  19:0]         tmp_2_out_              ;
assign tmp_2_out_ = (tmp_2_out[15] == 1'b1) ? {4'b1, tmp_2_out} : {4'b0,
tmp_2_out};
signed_mul_8 u2_signed_mul_8
(
    .dout                           (tmp_2_out              ),
    .a                              (tmp_2_in1              ),
    .b                              (tmp_2_in2              )
);
reg             signed [   7:0]         tmp_3_in1               ;
reg             signed [   7:0]         tmp_3_in2               ;
wire            signed [  15:0]         tmp_3_out               ;
wire            signed [  19:0]         tmp_3_out_              ;
```

```
assign tmp_3_out_ = (tmp_3_out[15] == 1'b1) ? {4'b1, tmp_3_out} : {4'b0,
tmp_3_out};
signed_mul_8 u3_signed_mul_8
(
    .dout                           (tmp_3_out              ),
    .a                              (tmp_3_in1              ),
    .b                              (tmp_3_in2              )
);
reg             signed [   7:0]      tmp_4_in1               ;
reg             signed [   7:0]      tmp_4_in2               ;
wire            signed [  15:0]      tmp_4_out               ;
wire            signed [  19:0]      tmp_4_out_              ;
assign tmp_4_out_ = (tmp_4_out[15] == 1'b1) ? {4'b1, tmp_4_out} : {4'b0,
tmp_4_out};
signed_mul_8 u4_signed_mul_8
(
    .dout                           (tmp_4_out              ),
    .a                              (tmp_4_in1              ),
    .b                              (tmp_4_in2              )
);


always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0) begin
        tmp_1   <= 'd0;
        tmp_2   <= 'd0;
        tmp_3   <= 'd0;
        tmp_4   <= 'd0;
        valid_1 <= 'd0;
    end
    else if(process == 'd1 && valid_cnt >= 'd1 && valid_cnt <= 'd5) begin
        if(valid_cnt >= 'd1 && valid_cnt <= 'd4) begin
            tmp_1_in1 <= tmp[ROW_LEN + ROW_LEN + ROW_LEN + valid_cnt - 1];
            tmp_2_in1 <= tmp[          ROW_LEN + ROW_LEN + valid_cnt - 1];
            tmp_3_in1 <= tmp[                    ROW_LEN + valid_cnt - 1];
            tmp_4_in1 <= tmp[                              valid_cnt - 1];
            tmp_1_in2 <= weight['d12 + valid_cnt - 1];
            tmp_2_in2 <= weight['d08 + valid_cnt - 1];
            tmp_3_in2 <= weight['d04 + valid_cnt - 1];
            tmp_4_in2 <= weight['d00 + valid_cnt - 1];
        end
        if(valid_cnt >= 'd2 && valid_cnt <= 'd5) begin
            tmp_1 <= tmp_1_out_ + tmp_1;
            tmp_2 <= tmp_2_out_ + tmp_2;
            tmp_3 <= tmp_3_out_ + tmp_3;
```

```
            tmp_4 <= tmp_4_out_ + tmp_4;
        end
    end
    else if(valid_cnt == 'd6) begin
        valid_1 <= 'd1;
    end
    else begin
        tmp_1   <= 'd0;
        tmp_2   <= 'd0;
        tmp_3   <= 'd0;
        tmp_4   <= 'd0;
        valid_1 <= 'd0;
    end


reg                  [  19:0]        o_data_reg                ;
always@(posedge sys_clk or negedge sys_rst_n)
    if(sys_rst_n == 1'b0) begin
        o_valid    <= 'd0;
        o_data_reg <= 'd0;
    end
    else if(valid_1 == 'd1) begin
        o_valid    <= 'd1;
        o_data_reg <= tmp_1 + tmp_2 + tmp_3 + tmp_4 + {bias, 12'b0};
    end
    else
        o_valid <= 'd0;
assign o_data = (o_data_reg[19] == 1'b0) ? o_data_reg[19:12] : ~o_data_reg[19:12]
+ 1'b1;

endmodule
```