基于 HLS 的 Tiny-yolo 卷积神经网络加速研究

重庆大学硕士学位论文

（学术学位）

学生姓名：张丽丽

指导教师：黄智勇　副教授

专　　业：通信与信息系统

学科门类：工　学

重庆大学通信工程学院

二 O 一七年四月

Research on the Acceleration of Tiny-yolo Convolution Neural Network Based on HLS

A Thesis Submitted to Chongqing University

in Partial Fulfillment of the Requirement for the

Master's Degree of Engineering

By

Zhang Lili

Supervised by Assoc Porf Huang Zhiyong

Specialty：Communication and Information System

College of Communication Engineering of

Chongqing University, Chongqing, China

April 2017

摘　要

卷积神经网络（Convolution Neural Network, CNN）在计算机视觉领域得到了广泛的应用，特别是在图像的识别、分割以及目标检测等方面突显出了较好的应用前景，但是目前大部分卷积神经网络系统基本上都是在 GPU 环境下运行。尽管 GPU 能够实现实时处理，然而其功耗大，成本高，难以满足一些低功耗低成本应用领域的要求，因此能够研究出一套速度快、准确度高以及功耗低的目标检测系统具有重要的实际意义。

与其他目标检测神经网络相比，结构简单、检测速度快的 YOLO（You Only Look Once）卷积神经网络更适合应用于低功耗设备。目前已经有一些相关研究将 YOLO 运用于低功耗设备，由于在 ARM 架构的嵌入式设备上运行神经网络，其速度非常慢，因此大多数是基于 FPGA 开发的专用硬件加速器。虽然相对于 ARM，基于 FPGA 的专用硬件加速器大幅提升了目标检测速度，但是其实现难度大，开发周期长。通过分析卷积计算的并行性和 Tiny-yolo 网络结构的并行特征，基于 ARM+FPGA 双架构的 ZC702 开发板，使用 HLS 进行硬件加速。通过权衡网络的运行速度和硬件资源的消耗，以流水线并行处理算法为主，定点运算为辅，设计了 3 个 IP 核，在提高运行速度的同时，大大缩短了开发周期。

实验结果表明：经过硬件加速的 Tiny-yolo 网络比未经硬件加速的版本在速度上提高了 6 到 7 倍。由于加速的网络采用了定点计算，与原始的浮点数据类型的网络相比，目标检测结果有一定的误差，但基本能够保持较高的检测精度，适合应用于实际工程之中。

关键词：卷积神经网络，YOLO，硬件加速，HLS

ABSTRACT

Convolution Neural Networks (CNN) have been widely implemented in the field of computer vision, especially in the fields of image recognition, segmentation and target detection, which has get a good application prospect. However, at present, most of the convolution neural network systems are basically implemented under the environment of GPU. Although the GPU can achieve real-time processing, the power consumption and the cost are extremely high. So it is absolutely difficult to meet the requirements of some low-power and low-cost application areas. Therefore, it is of great practical significance to develop a picture detection system with high speed, high accuracy low power consumption.

Compared with other convolution neural networks, the YOLO convolution neural network is simple in structure and faster in detection, which is more suitable for low-power consumption devices. There are some related studies to apply YOLO convolution neural network to low-power consumption devices. Due to the low speed of embedded devices of ARM architecture running on the neural network, most of the relative devices are based on dedicated hardware accelerator of FPGA. Compared with the embedded device of ARM architecture, device based on dedicated hardware accelerator of FPGA has obviously improved the speed of target detection. However, it is also difficult to achieve and the development period can be a long time. Through the analysis of the parallelism of the convolution calculation and the parallel features of the Tiny-yolo network structure, the ZC702 development board based on the ARM + FPGA double architecture, using HLS for hardware to accelerate the calculating speed and weighing the running speed with hardware resource consumption. The algorithm is mainly based on the pipeline-parallel processing algorithm, supplemented with the fixed-point processing algorithm, and three IP cores were designed in order to improve the running speed and shorten the development period.

The experimental results show that the hardware-accelerated Tiny-yolo network is 6 to 7 times faster than the network without acceleration. As the accelerated network uses fixed-point calculation, compared to the type of original floating-point data network, the target test results have some certain errors. However, the basic detection accuracy can be retained, which is suitable for the field of engineering.

Keywords: Convolution Neural Networks, YOLO, Hardware Acceleration, HLS

目 录

1 绪论

1.1 课题研究的背景和意义

卷积神经网络(Convolution Neutral Network, CNN)是在人工神经网络的基础上发展起来的一种多层感知器，能够很好地适应图像的平移、比例缩放以及旋转等形式的变形，是提取图像特征的灵敏传感器。卷积神经网络是一种深度学习架构，其通过局部连接、权值共享以及子采样三个步骤来有效减少权值参数的数量。卷积神经网络的权值共享网络结构和生物神经网络结构特别相似，可以减小网络模型的复杂度和压减权值的数量。卷积神经网络是一种典型的前馈多层人工神经网络，能够自动学习有标签数据且从中提取出复杂的特征。卷积神经网络的优点是只需对输入图像进行很少的预处理，就能够从输入的像素图中得出视觉模式，即使输入的图像变化较大也能有较好的识别效果，卷积神经网络的识别能力对图像的畸变或简单几何变换并不敏感，以上优点是卷积神经网络在多层人工神经网络中被广为研究的重要原因[1-2]，这些优点在网络输入为多维图像时表现的更为突出。相对于传统识别算法，还有另外一个重要原因是图像可以直接用作网络输入，进而避免了复杂特征提取与数据重建的过程，因此卷积神经网络结构在机器视觉[3]、模式识别[4]、视频监控[5]、图像搜索等领域应用的越来越广泛。截止目前，卷积神经网络在手写字体、交通标志识别等多种任务中得到了比较多的应用。卷积神经网络凭借其出色的表现并随着深度学习的发展进步，作为一种特殊的深度学习架构受到了广泛的关注。综上分析，加大对卷积神经网络的研究，有利于深度学习的发展以及对解决各个应用领域的问题都有重要的实际意义。

随着互联网、社交网络及智能手机在现代生活中的普及和发展，大量的图片信息也铺天盖地[6]。据相关调查，Instagram用户每天上传的图片量就达到6000万张，WhatsApp用户每天的图片量更是高达5亿多张，在国内比较普及的社交软件微信，很多用户也是用图片在朋友圈中展示日常生活。图片已经以一种重要的传递信息的方式存在于网络中，同时也出现了一定的问题，在过去文字传递信息的年代，用户使用关键词就能很容易地查找到想要的信息，现在图片传递信息，就无法使用相同的方式进行搜索，这就影响了用户查找目标信息的效率[7-10]。图片虽然直观易于表达便于记录，但是也造成了信息搜索及存储的不便。

随着训练数据的增加和机器性能的提高，基于卷积神经网络的目标检测打破了传统目标检测的瓶颈，已成为当前目标检测的主要算法。但是目前大部分卷积神经网络系统基本上都是在GPU环境下实现的，尽管GPU能够实现实时处理，然

而其功耗大，成本高，难以满足一些低功耗低成本应用领域的要求，因此研究出一套速度快、准确度高、体积小、功耗低的图片检测系统具有重要的实际意义。

1.2 卷积神经网络的研究现状

神经网络最早是 Hubel 和 Wiesel 通过研究猫视网膜以及视觉皮层中枢神经细胞的信息处理机制而提出的，该信息处理机制解释了视觉神经的机理[11]。同时，他们提出了局部感受野的概念，该理论对卷积神经网络的发展和应用具有重大的意义。在该理论的启发下，Fukushima 经过研究提出了基于感受野的神经认知机，能够识别有变形或有位移的模式，并且这种神经认知机被认为是 CNN 的第一个实现网络[12]。在此后的工作中，Fukushim 重点研究了神经认知机在手写数字方面的应用，结果非常令人满意。自 20 世纪 90 年代以来，出现了大量的基于卷积神经网络的应用。最早开始的是用时延神经网络做语音识别和文档阅读，这个文档阅读系统通过一个被训练好的卷积神经网络和一个概率模型，概率模型可以实现语言方面的一些约束。此后，Lecun 等[13]在 1995 年研究出基于卷积神经网络的 LeNet-5 模型，这个模型成功地展现了卷积神经网络的应用。随后，国内外研究人员提出了多种形式的卷积神经网络，并把卷积神经网络大规模地应用于邮政编码识别、人脸识别等多个领域。但是，随着卷积神经网络计算繁琐度逐渐增加和运算量的增大，模型深度的增加以及模型的训练参数数量的增加，模型训练过程中常常陷入局部最小值。并且对于高维的数据输入，还容易产生过拟合现象，在当初的硬件条件限制下，卷积神经网络的发展出现了瓶颈。所以，早期的卷积神经网络只是应用在小尺度的图形上。2006 年，Hinton[14]教授团队对此问题提出了两个重要观点：一是相比于浅层学习对数据的特征表达，包括多个隐含层深度神经网络的学习能力要更加突出，学习得到的特征能够对数据进行更加本质的表示。二是通过逐层初始化的方法，解决深度神经网络的训练难度问题。基于以上两个重要观点，上述瓶颈问题暂时得到了解决。Sutskever 和 Krizhevsky 在 2012 年的 ImageNet 竞赛中，将深度学习的模型理论首次用在卷积神经网络上[15]。他们采用了一个 8 层深度卷积神经网络来对百万张网络图片进行图像分类及目标定位，图片数据集则包含了 1000 个不同的类。通过使用这种改进的方法，结果取得了明显地改善，执行图像分类任务的错误率低至 15.3%，和手动特征设计 26.3%的错误率相比，错误率降低了近一半[16]。此后，深度卷积神经网络便应用在 Google Plus 的图象标注和搜索上，2014 年 ImageNet 竞赛的冠军模型 GoogleNet 证明了使用更多卷积与更深的层次能够得到更好的结构[17]。2015 年 ImageNet 竞赛的冠军模型 Deep Residual Learning，仍然采用深度卷积模型，也就是将卷积进行到底，并在该模型中用到了 bottleneck 形式的块结构（即跨越几层的直连），最深的模型多达 152 层

[18]。谷歌在 2016 年研发的基于深度神经网络及搜索树的智能机器人“Alphago”在围棋大战中击败了九段围棋高手李世石[19]。

卷积神经网络早在 1994 年就被成功地应用于目标检测领域。由于训练数据的缺乏、硬件性能的限制和过拟合问题，基于卷积神经网络的目标检测在很长一段时间里没有取得进展。与当时的传统方法相比，不论是在检测精度上还是在检测速度上，基于卷积神经网络的目标检测都没有太大优势，因此该研究逐渐被忽视。直到 2012 年，卷积神经网络 AlexNet 在图像识别上取得了重大的突破，研究者才开始重新审视卷积神经网络，讨论如何将神经网络有效的应用在目标检测中。如今，基于卷积神经网络的目标检测已经超越传统目标检测方法，成为当前检测的主流方法。Vaillant 等人最先提出了将卷积神经网络应用于人脸检测方面，与全连接的多层感知机相比，采用该卷积神经网络获得的检测效果更好，体现了卷积神经网络在特征提取和特征选择上的独特优势。Garcia 等人在 Vaillant 等人的研究基础上增添了一个卷积层和一个池化层，提高了特征的表达能力，采用基于 Boosting 的训练策略，加快了卷积神经网络的收敛速度，增强了人脸检测器的鲁棒性。随后，Sermanet 等人又提出了在行人检测中应用卷积神经网络。该方法采用了卷积稀疏编码的方式无监督地预先训练每个卷积层，然后再利用训练样本有监督的微调整个卷积神经网络。虽然基于卷积神经网络的目标检测在各个方面得到了一定的应用与发展，但也有很多问题需要进一步解决。一方面，目前都是通过实验来证明卷积神经网络的有效性，训练参数的设置大多依靠经验和实践，缺乏理论指导和量化分析。另一方面，需要针对目标检测设计更加合理的网络结构，结合回复式神经网络提升检测效率，实现多尺度多类别目标检测。

卷积神经网络主要以通用处理器为基础，通过软件的方式实现。事实上，CNN 作为一种前馈网络结构，层与层之间具有高度的独立性，各层网络计算独立，层间无数据反馈。因此，CNN 是一种高度并行的网络结构，通用处理器为执行逻辑处理和事务处理而优化的特性并不适合用来挖掘 CNN 的并行性，基于软件方式的 CNN 在实时性和功耗方面都不能满足应用的需求[20]。FPGA 作为一种可编程器件，具有计算资源丰富、灵活可配等优点，越来越多的研究者采用 FPGA 开发卷积神经网络的应用[21-23]。然而，采用 FPGA 实现规模较大的卷积神经网络时，需要进行大量的卷积运算和非线性函数计算以及子抽样操作，在这些计算过程中，需要访问大量的数据以及解决中间数据的存储问题。

通过硬件加速网络的计算，需要在最大程度上利用网络模型的并行性特征。通常多层神经网络的连续层之间，都存在数据的相关性，不过这一级别的并行性并不是很高。由于 CNN 数据的传播具有前向特点，多层卷积神经网络连续层之间的这种数据相关性就严重影响了层间并行计算的进行，且各层之间的任务并行又

是有限的，开发起来还比较困难。而传统卷积神经网络的突出问题是太多的参数需要训练，在大多数情况下都超过了可用的训练样本数，不容易学习到参数最优值，且容易出现过拟合情况。并且逐层学习时，需要先确定上一层的参数才可以开始下一层参数的学习，整个过程比较繁琐且训练需要很长时间。卷积层的计算非常耗时，这就是深度神经网络系统需要重点解决的问题。有实验结论表明，一般的卷积神经网络只占了 5%左右的参数，但是计算量却占据了整个网络模型的90%-95%。所以，要想解决卷积神经网络系统的训练效率问题，就必须研究如何提高卷积计算的速度。

Lind 和 Ryoo 等人专注于有关通用并行计算的 GPU 问题，他们的研究基于英伟达（NVIDIA）的 GPU 上的统一计算设备架构（CUDA）编程的 API 来创建并行程序，而程序员用它来编写调用内核，从而在一组并行的线程中得到执行。尽管目前大型的卷积网络系统基本上都是在 GPU 环境下实现的，并且取得良好的效果，然而其功耗大，成本高，难以满足一些低功耗低成本应用领域的要求。

1.3 论文的主要工作与章节安排

从卷积神经网络的研究意义出发，详细介绍了卷积神经网络的国内外研究现状，分析了各个卷积神经网络的结构和特点,对比了各个网络的检测精度和速度。通过分析卷积神经网络结构的并行性和卷积计算的并行性，结合硬件的并行计算能力，采用高层次综合工具 HLS 进行硬件设计以达到加速 Tiny-yolo 卷积神经网络的目的。最后基于 ZC702 硬件平台,对比神经网络加速前后的检测速度和准确度，分析加速的原理和效果，并阐述了下一步需要继续研究的工作。文章具体分为 5 个章节：

第一章：绪论。主要介绍了卷积神经网络的研究背景意义，国内外研究现状，对本论文的研究内容进行了简要概述，并对论文的安排布局进行了简要介绍。

第二章：卷积神经网络的基本理论和加速研究概况。主要对卷积神经网络的概念和一些基本知识进行了阐述，其次介绍了几种检测速度较快的卷积神经网络以及它们的优缺点。最后从检测原理、训练和测试三个方面详细介绍了YOLO卷积神经网络。

第三章：Tiny-yolo 卷积神经网络加速算法研究。首先分析 Tiny-yolo 神经网络的数据量和计算量，并结合 ZC702 硬件资源对整个网络进行多模块设计和数据量化。然后优化和设计各模块中的卷积池化操作以达到并行加速的效果。

第四章：硬件加速实现。在ZC702芯片的FPGA+ARM的双架构上，使用HLS相关技术，对Tiny-yolo卷积神经网络进行分析和加速实现。

第五章：结论与展望。结合各章节及实验结论对全文进行总结，并对以后的研究工作进行了展望。

2 卷积神经网络的基本理论及加速研究现状

2.1 卷积神经网络的概述

卷积神经网络 CNN（Convolution Neural Networks），本质上是一种多层感知器，由生物学家通过研究猫的视觉皮层一步一步发展起来的[24]。视觉皮层细胞里面的结构非常复杂，并且视觉皮层细胞对视觉输入空间子区域的敏感度极高，用这样的方式平铺覆盖了整个视野区域，因此被称为感受野。对这些细胞进行有效区分，分为简单细胞和复杂细胞两种类型。简单细胞对来自感受野范围内边缘刺激的模式产生最大程度的响应，而复杂细胞的接受域较大，如果对复杂细胞进行刺激的位置是确定的，那么复杂细胞就具有局部不变性。可以把处于神经网络中的每个神经元节点都看作是线性的一维排列结构，层与层的每个神经元节点之间的连接均是全部连接。但是，卷积神经网络里面层与层间神经元节点的连接并不是全连接形式，而是在充分利用层间局部空间相关性的基础上，将上层的神经元节点和相邻每层的神经元节点连接起来，被称为局部连接。卷积层的卷积滤波器需要在整个感受野中执行重复操作，并对输入的图像进行实时卷积，卷积结果便构成了输入图像的特征图，并以一定的规则提取图像的局部特征，这就是卷积神经网络的权值共享。每个卷积滤波器采用相同偏置和权重矩阵，实现相同的参数共享。卷积神经网络是一种特殊的能够对图像进行识别的方式，是一种非常有效的带有前向反馈的网络。CNN 最初的主要目标是识别二维图形，因为它的网络结构对平移、比例缩放、倾斜或其他形式具有高度不变性。现在，卷积神经网络结构已被广泛应用在机器视觉、模式识别、视频监控和图像搜索等领域。

2.1.1 卷积神经网络的结构

图像通过输入层进入卷积神经网络，然后通过一系列隐藏层的转换,最后输出,每个隐藏层都由一组神经元组成，每个神经元和前一层神经元的连接都是完全连接，且单层神经元的功能是完全独立地，不存在任何连接共享。最后的全连接层作为输出层，实现分类功能并输出分类的分值。

对于完全连接结构的神经网络在 CIFAR-10 训练集中，图片的大小为 32*32*3（即 32 宽 32 高 3 颜色通道），第一隐藏层的神经元个数是 32*32*3=3072 个，这个数字看起来能够接受，但这种全连接的结构不能适应更大的图片，比如一个大小为 200*200*3 的图片，会产生 200*200*3=120000 个神经元。显而易见，这种完全连接结构会造成很大的浪费，且过多的参数容易导致过拟合。

卷积神经网络的巨大优势在于对巨量图片的输入，能以一种更加合理的方式限制其结构。卷积神经网络不同于一般神经网络结构，其各层安排在了长、宽、

高三个维度上。我们可以看到，一层的神经元只连接到它前面层的小区域内，这不同于上面所说的完全连接方式。作为多层神经网络之一的卷积神经网络，其结构如图 2.1 所示。

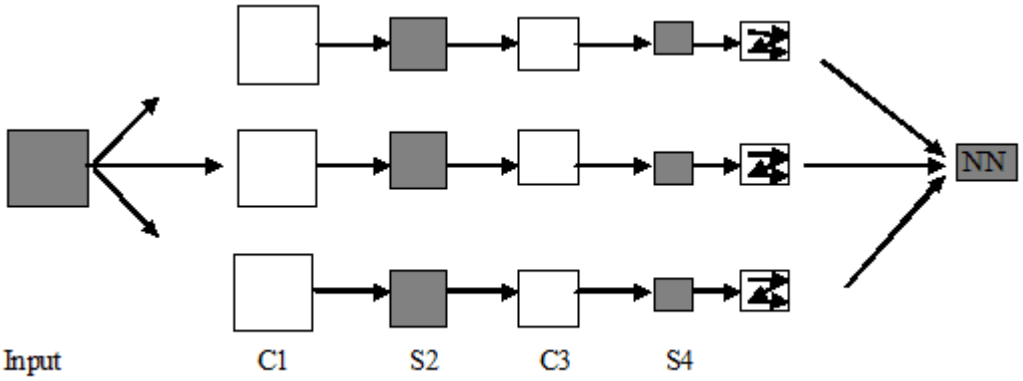


图 2.1 卷积网络基本构型

Fig 2.1 The basic configuration of convolution network

输入图像通过 3 个不同的卷积过滤器后生成 3 个不同的特征映射图(即 C1 层)，每个特征图的大小与输入大小一致，为了降低特征图的分辨率，在 C1 层之后进行了池化操作，接着以 sigmoid 函数作为激活函数生成分辨率较低的特征映射图，然后重复该过程得到 S4 层的特征图，最后将这些特征图的像素值连接成一个向量作为神经网络的输出。

卷积神经网络的核心部分是特征提取和特征映射，在特征提取的时候每个神经元与前一层的局部感受区域相连，然后提取局部特征；进行特征映射时所有神经元共享权值，以此减少了整个神经网络的参数数量。而使用 sigmoid 函数对神经元进行激活时，很大程度上保证了特征的位移不变性。将特征提取和特征映射结合在一起，使得神经网络能够容忍输入样本的畸变进而提高网络的泛化能力。

利用映射面上共享权值的神经元，减少网络自由参数的个数，降低了网络参数选择的难度。卷积神经网络中每一个卷积层后都有一个池化层，这个池化层是用来求局部平均和二次提取，正是这种特有的两次特征抽取结构使网络对输入样本有较高的畸变容忍能力。

卷积层和池化层是卷积神经网络两个重要的部分。卷积，对输入数据采用若干个过滤器，每个过滤器对整个输入进行特征提取得到一个特征。例如，图像的第一卷积层采用 4 个 6*6 过滤器，可获取 4 种特征，对图像使用一个过滤器之后得到的结果，称为特征图谱。因此，特征图谱的数量与过滤器的数量一致。如果前面的输入层也是一个卷积层，那么过滤器的输入就相当于前一层的所有特征图

谱，经过卷积计算后，输出另外一个特征图谱。换言之，就是将滤波器的特征值作为输入，再进行一次过滤器过滤操作。如果引进权重的概念，将一个权重分布到整个图像上，那么这个特征就和位置没有关系了。同时，多个过滤器就可以探测出多个不同的特征。

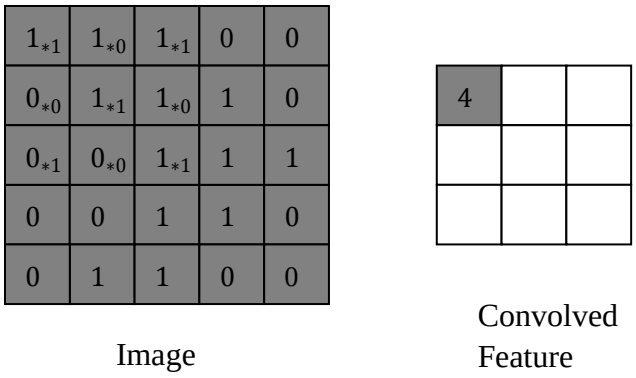


图 2.2 (A) 卷积的过程 1

Fig 2.2 (A) The process of convolution 1

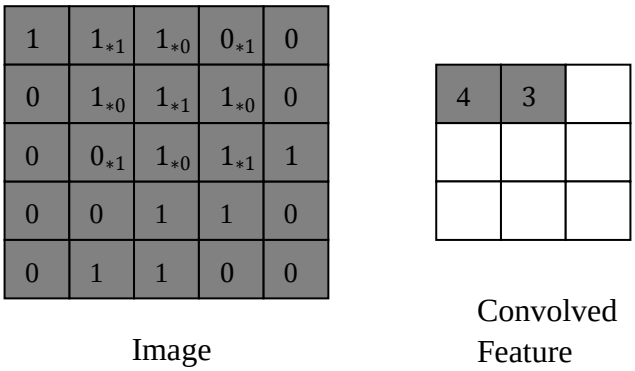


图 2.2 (B) 卷积的过程 2

Fig 2.2 (B) The process of convolution 2

如图 2.2 所示，展示了一个 3*3 的卷积核在 5*5 的图片上做卷积的过程。每次卷积过程就是一种特征提取方式，经过多次的卷积提取，就把图片中符合条件的特征提取出来了。

池化的目的是缩减网络的数据量和参数规模。譬如，当输入为一个 12*12 的图像时，使用一个 6*6 的子采样，就可以得到一个 2*2 的输出图像，其实就是将图像上的 36 个像素合并成输出图像中的一个像素。实现池化的过程有多种方法，最为常用的就是平均池化、最大池化和随机池化三种，如图 2.3 所示。

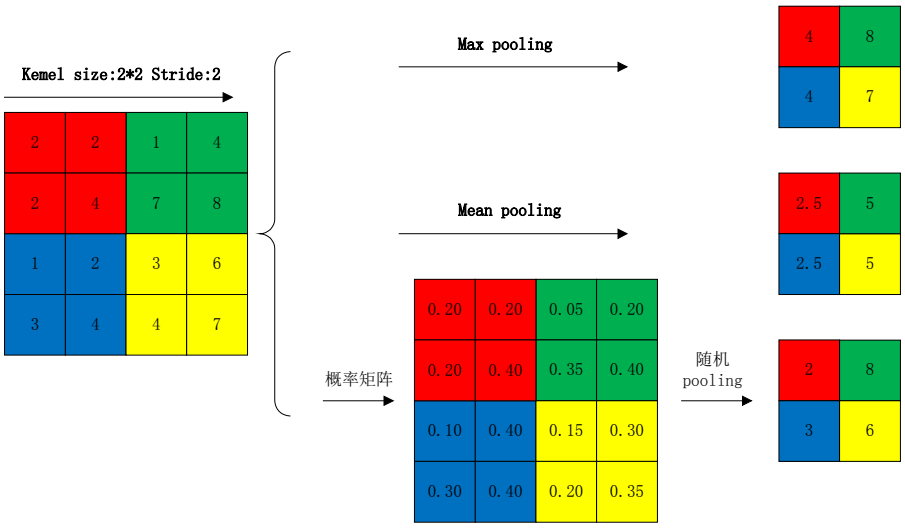


图 2.3 池化的方式

Fig 2.3 The way of pooling

① 平均池化（mean-pooling），即对相邻区域内特征点求平均值，有利于保留背景。

② 最大池化（max-pooling），即对相邻区域内特征点取最大值，有利于提取纹理。

③ 随机池化（stochastic-pooling），即对所有像素点按照数值大小赋予概率，然后按照概率进行采样。

特征提取过程中的误差大小主要取决于两个方面：一是由于邻域大小受到限制引起估计值的方差增大；二是由于卷积层参数偏差引起估计均值的偏移。根据引起误差的理论来讲，平均池化有利于减小第一种误差，可以更好地保留图像的背景信息；相反的是，最大池化有利于减小第二种误差，可以更好地保留纹理信息。

全连接层通常连接在一个池化层或卷积层之后，全连接层的输出就是最后的结果。由此可见，这一层中的每个神经元都将和前一层的每个神经元连接，这和传统神经网络相同。

通过改进的反向传播来进行训练时，池化层成为考虑的因素并且基于所有值来更新卷积核的权重，实际训练中我们设定一些前向反馈数据，方便进行优化调整。

2.1.2 参数优化

假定有一个 1000 像素*1000 像素的图像，那么就有 100 万个隐藏层神经元，如果它们全连接的话，就是说每个隐藏层神经元都和图像的每一个像素点相连接，则有 1000*1000*1000000=10^{12} 个连接，也就是 10^{12} 个权值参数。如果所有的连接

都计算，明显是不值得的。采用之前提到的池化层进行原始数据分组和特征提取，也就是先分块然后进行特征提取。经过这样的操作处理后，可以减少连接的数目，同时也就减少了神经网络权值参数的个数。如果局部感受区域是 10 像素*10 像素，即设定的过滤器是 10 像素*10 像素，那么隐藏层每个感受区域就只需要和这 10 像素*10 像素的局部图像进行连接，因此 100 万个隐藏层神经元形成了 1 亿个连接，也就是 10^8 个参数，和全连接相比减少了 4 个数量级，大幅度减少了计算量。

因为图像各局部的统计特性具有一致性，使得卷积操作就相当于是一种与图像上位置无关的特征提取过程。即某部分的学习特征可以应用到整个图像的任意部分，这样通过学习任意选定的小块样本获取特征并对整个图像进行卷积计算，从而得到图像上任意位置的不同特征的激活值。

简单来讲，我们在尺寸比较大的图像上任意选取一块作为样本，例如选择一个 8*8 的样本，通过对这个样本训练学习得到特征，然后把在这个小样本学习得到的特征作为卷积核，应用到整个图像的任意地方。也就是说，通过学习任意选定的小样本获取特征并和大尺寸图像做卷积，进而可以得到大尺寸图像上任意位置不同特征的激活值。

卷积神经网络的核心思想就是将完整输入的信息切分成一个一个的子采样层进行采样，然后将提取到的特征及权重值作为输入参数，并传到下一层。这充分利用了特征分区提取、权值共享、时间或者空间采样规则等方法。

卷积神经网络和一般神经网络相比，在图像处理方面有三个优点：（1）输入的图像与网络的拓扑结构吻合度相当高。（2）特征提取与模式分类可以在同一时间进行，并且是在训练过程中同时产生的。（3）权值共享能够大幅度地减少网络训练参数，简化神经网络结构，增强适应性。

目前，卷积神经网络已经在语音分析及图像识别等应用领域掀起了研究热潮。卷积神经网络的权值共享网络结构非常类似于生物神经网络，大大降低了网络模型的复杂度和减少了权值的数量。卷积神经网络的上述优点在网络输入是多维图像时表现的特别明显，且可以直接把图像作为网络输入不需任何处理，省去了传统识别算法中特征提取和数据重建的复杂过程。

2.2 深度卷积神经网络的研究进展

近些年来，深度卷积神经网络（DCNN）在图像分类和图像识别上获得了较快的发展。尤其是这两年，先后涌现出了Regions with CNN features(R-CNN)，Fast R-CNN，Faster R-CNN，Inside-Outside Net(ION)，HyperNet，Scale-dependent Pooling and Cascaded Rejection Classifiers (SDP-CRC)，You Only Look Once(YOLO)等速度越来越快且准确率越来越高的目标检测方法。

2.2.1 R-CNN

R-CNN 作为一种早期基于深度神经网络的目标检测算法[25]，它有以下检测流程：首先在图像上生成候选区域，接着在各个候选区域使用卷积神经网络进行特征提取，然后使用 SVM 模型算法进行一个分类，由于初次分类产生的特征是一个粗略的结果，为了得到一个精确的检测结果，在初次分类结果的基础上，再进行第二次 CNN 特征提取并结合一个回归模型得到更加精确的边界框。其检测系统如图 2.4 所示。

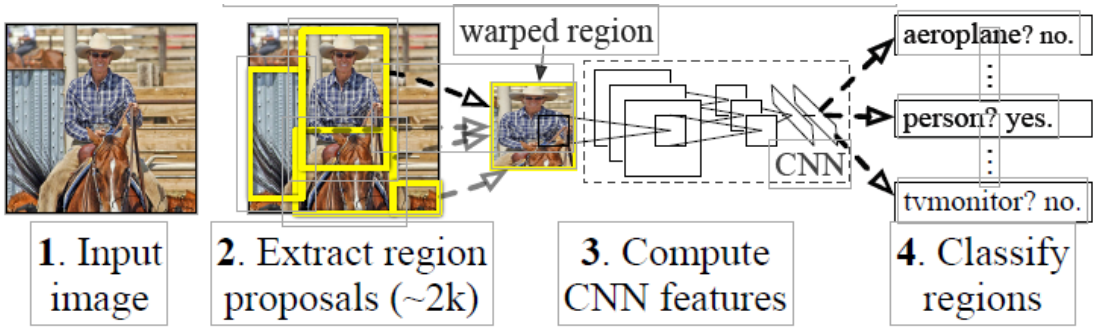


图 2.4 R-CNN 对象检测系统概述

Fig 2.4 R-CNN object detection system overview

其中，获取候选区域的方法是 selective search。由于 CNN 算法在处理图片时，对图片大小有要求（227*227），然而实际中一个图像的候选区域尺寸的差异是比较大的，为了使网络能够正常运行，我们就需要把这些不同大小的候选区域变化到同一尺寸来适应网络结构。

该方法在 VOC2011 test 数据集上的检测精度达到了 71.8%。该方法存在以下缺点：一是整个计算流程所需要的计算时间特别巨大，因为时间消耗主要在以下几点：首先使用 CNN 进行一个前期的特征提取，然后对前期特征进行分类得到一个初步的检测结果，在初步检测结果之后进行二次特征提取操作，以及最后对边界框的一个回归处理。这四个处理流程是以串行的方式进行的，训练过程需要耗费大量的时间。二是在处理的过程中提取的特征数量庞大，后面的处理依赖前面的特征，为了后面能够正确处理，我们必须要对前面的步骤的处理结果进行保存，存储量是特别巨大的，占用大量的内存空间。三是在同一张图片提取 2000 个不同大小，不同类别的候选区域，这些区域会存在一定的重合，而每个区域特征提取又要进行独立计算，因此完成效率很低。

2.2.2 Fast R-CNN

为了改进 R-CNN 计算量大、内存占用高而导致的训练和测试效率低下，Ross Girshick 提出了 Fast R-CNN 方法[26]。相较于 R-CNN 的改进主要使用 VGG19 网络

结构，基于这种改进之后，整个训练和测试时间分别比 R-CNN 提高了 9 倍和 213 倍。它的核心思想有两点：一是由 2000 多个候选区域分别进行卷积变成了直接对整个图像进行卷积得到特征图像；二是将候选区域分类和边框拟合两个操作进行合并。

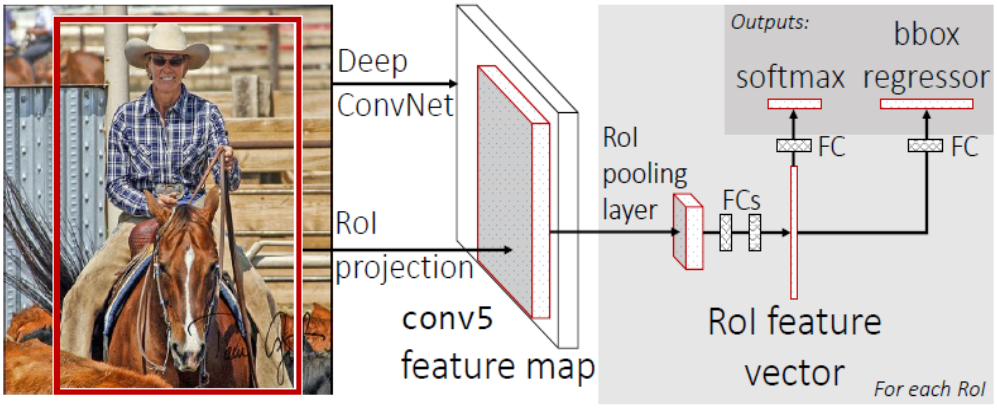


图 2.5 Fast R-CNN 检测原理

Fig 2.5 Detection principle of Fast R-CNN

为了实现候选区域的分类与边框拟合的合并，作者考虑使用了一个特殊的网络结构。这个网络输出层为两个全连接层，这两个全连接层分别同时执行类别预测与边框预测(如图 2.5 所示)，在对这两个步骤同时训练时，与 R-CNN 采用单独代价函数不同的是，采用了一个联合代价函数

$$L\left(p,u,t^{u},v\right)=L_{cls}\left(p,u\right)+\lambda\left[u\geq 1\right]L_{reg}\left(t^{u},v\right)$$

cls 和 reg 分别是 classification loss 和 regression loss。该方法相比于 R-CNN，速度快了很多。尤其是在测试一幅新图像时，这种方法在忽略生成候选区域时间的基础上可以达到实时检测。但是由于生成候选区域的 selective search 算法在处理一张图像需要 2 秒左右的时间，限制了该方法的应用。

2.2.3 Faster R-CNN

R-CNN 和 Fast R-CNN 在生成候选区域的时候都采用的是 selective search 算法，这种算法的缺陷是非常耗时。为了减少耗时，提高速度，Shaoqing Ren 提出了另外一种得到候选区域的方法，这种方法命名 Faster R-CNN[27]。具体做法是采用两个卷积神经网络来分别实现获取候选区域和候选区域的分类和边框回归两个任务。由于这两个神经网络都涉及到卷积计算，如果能够将卷积部分进行参数共享，就可以只在最后的几层分别实现各自的特定目标任务，而前几层执行相同的操作，

通过构造这两个神经网络，对一幅图像由于前几层卷积层是共享的，就只需要一次卷积计算，后几层基于前几层的基础上就可以实现候选区域生成和每个候选区域的类别和边框。

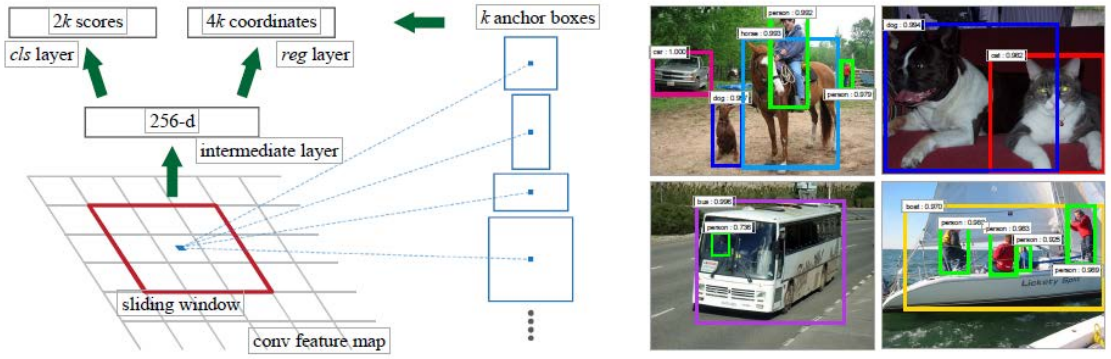


图 2.6 候选区域网络和利用其在 PASCAL VOC 2007 的检测结果

Fig 2.6 Region Proposal Network (RPN) and example detections using RPN proposals on PASCAL VOC 2007 test

如图 2.6 所示，候选区域生成网络（Region Proposal Network, RPN）方法的原理如下：首先在输入图像上进行卷积计算，提取到一个特征。得到特征图像之后，使用滑动窗口，将局部特征映射到一维特征。在得到一维特征图的基础之上，预测 k 个候选区域,每个区域预测两个概率（cls 层，2k 个输出）,每个区域对应 4 个坐标（reg 层，4k 个输出）。这里的 k 个矩形框被称之为锚（anchor），锚的特点是与滑动窗口中心相同，但是大小不同，长宽比不同的矩形框。候选区域生成网络方法具有位移不变性。

Faster R-CNN 与 Fast R-CNN 主要的不同是在候选区域的选取上，它使用的 RPN 算法使得可以共享部分卷积层。这种算法能够保持 Fast R-CNN 精度不变的同时，缩短了近十倍的训练时间和测试时间。但是这种方法的缺点是其候选区域生成算法的限制，只能让 RPN 和 Fast R-CNN 进行单独的训练。

2.2.4 Inside-Outside Net（ION）

ION 算法得到候选区域也是基于 Region Proposal 的[28]，主要的核心点有两个：第一，结合上下文特征信息；第二，使用多尺度特征来进行预测。具体做法是不再单一使用 ROI 局部特征，而是使用了一个特殊的空间递归神经网络结构，把 ROI 以外的上下文信息结合起来，然后把各个卷积层的特征结合起来再进行预测。图 2.7 是 ION 网络示意图。

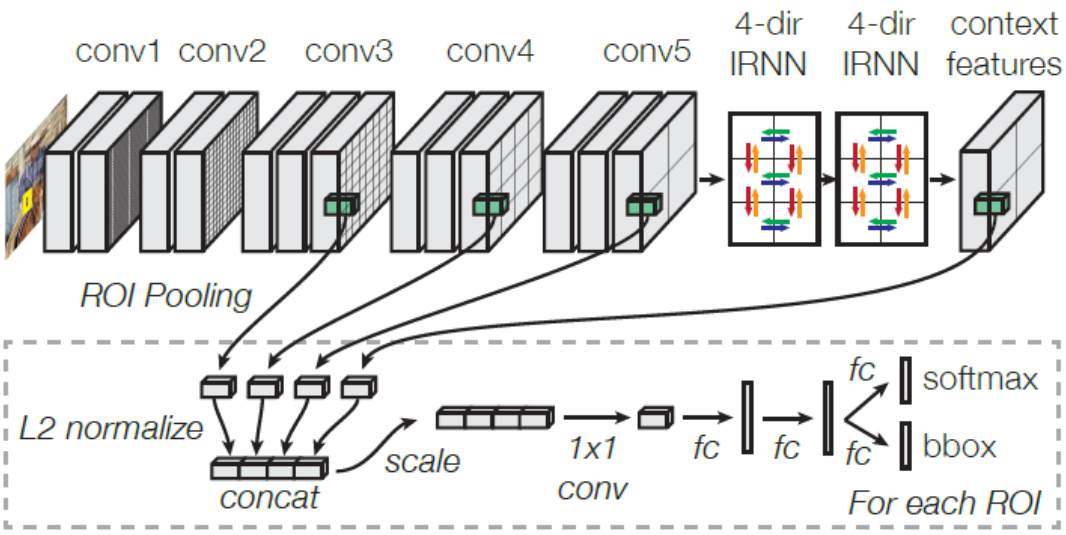


图 2.7 ION 网络

Fig 2.7 Inside-Outside Net (ION)

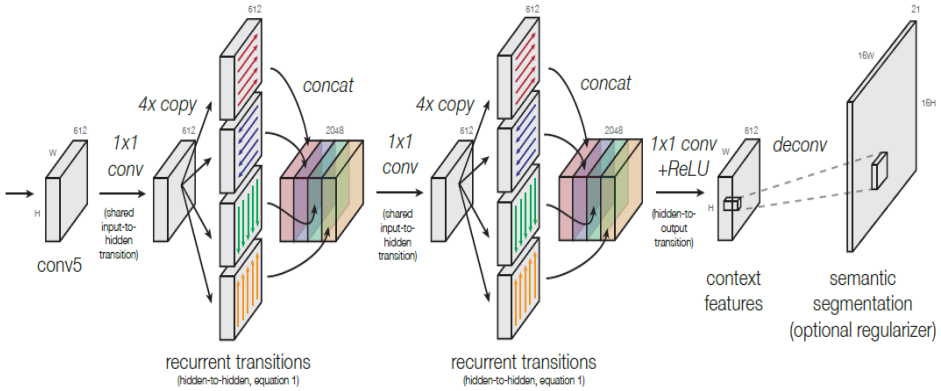


图 2.8 IRNN 四方向构架

Fig 2.8 Four-directional IRNN architecture

图 2.8 为 IRNN 四方向构架。ION 在图像的上、下、左、右四个方向分别使用 RNN，得到四个特征输出，将四个输出特征组合成一个特征，重复上面的操作，将得到的特征作为上下文特征，再融合前面几个卷积层的卷积输出，得到既包括上下文信息，又包括多尺度信息的特征。

2.2.5 HyperNet

HyperNet 通过改进 RPN 算法，在检测准确率上面有了进一步的提高[29]。其核心思想是与 Faster R-CNN 基本一致，但是它通过改进 RPN 算法，使得 region proposal 和检测率有了进一步的提高。

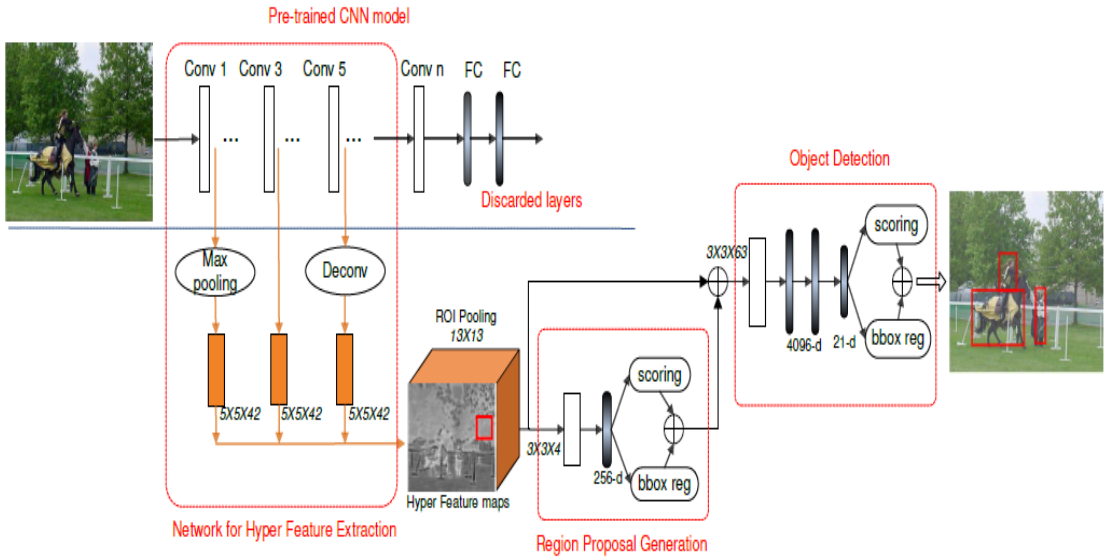


图 2.9 HyperNet 对象检测体系构架

Fig 2.9 HyperNet object detection architecture

根据相关研究表明，HyperNet 在使用 RPN 的时候，在卷积网络卷积层的前几层，由于特征图片分辨率高，尺寸大，这样高分辨率的特征图像对定位边框比较适合，但是由于抽象程度低，使得分类误差比较大。而后面几层卷积分辨率低，在定位边框的时候误差比较大，但是由于这些特征经过了多层卷积的一个特征提取，抽象度高，使得分类的时候误差小。因此，HyperNet 就是将较浅层和较深层的输出结合起来，得到 Hyper Feature 以提高目标检测的准确率和减小定位误差。

2.2.6 SDP-CRC

SDP-CRC 方法在 Fast R-CNN 的基础上能够提高候选区域的效率，同时对不同尺寸的目标大小有更好的适应性[30]。由于在神经网络中，一个输入图片目标大小不同，各种尺寸的目标在最后输出的时候获取的特征信息量不一样。鉴于此，作者提出了候选区域尺度池化策略，这种方法用神经网络前面某一层可以很好地表征小物体,用后面层的特征能更好地描述较大的物体。这样做到一个平衡，其本质就是根据目标的大小，来选择不同的卷积层特征。

以 VGG 网络为例，例如一个候选区域的尺寸小于 64 像素，那么就可以将第三层的特征通过池化过后作为输入特征输入分类器和边框回归器。如果候选区域的尺寸大于 128 像素，则在卷积神经网络最后一层的特征进行分类和回归。尺度池（SDP）模型在 VGG-16 中的应用如图 2.10 所示。

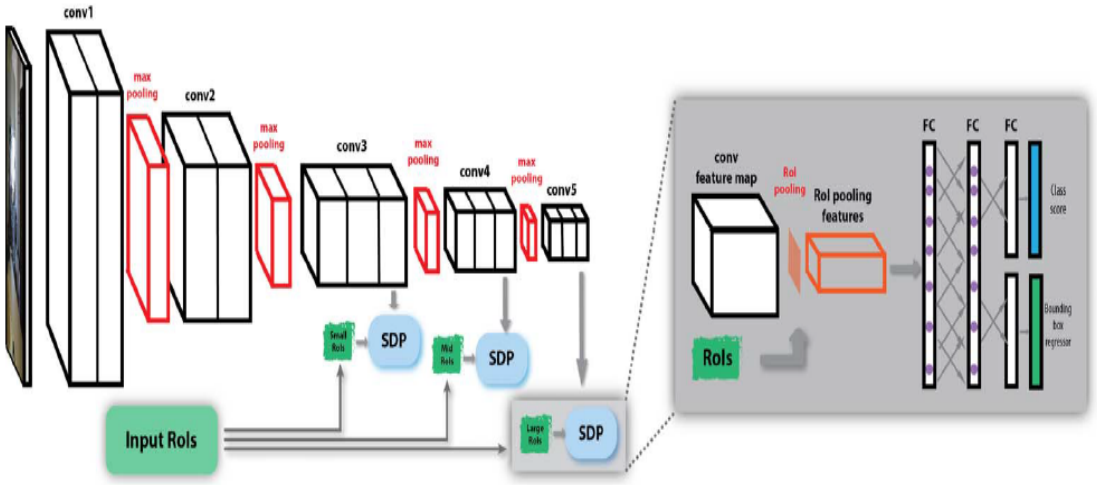


图 2.10 尺度池（SDP）模型在 VGG-16 中的应用

Fig 2.10 Details of scale-dependent pooling (SDP) model on VGG-16 net

Fast R-CNN 的精度比较高，但是有一个问题，需要产生成千上万个候选区域，每个候选区域都经过一次分类和回归计算，因此计算效率非常低。为了提高候选区域的计算效率，提出了使用舍弃负样本的级联分类器的策略。CRC 通过 AdaBoost 方法，将一些明显不包含某个物体的候选区域快速排出，将整个网络的计算集中在包含目标概率较大的候选区域，以此来提高候选区域的计算速度。

SDP-CRC 的准确率比 Fast RNN 提高了不少，检测时间缩短到了 471ms 每帧。

2.3 YOLO

You Only Look Once (YOLO)是 Joseph Redmon 等人提出的一种新的目标检测方法[31]。前面几种快速目标检测的卷积神经网络主要是采用分类器进行检测，相反 YOLO 网络采用的方法是同时进行目标定位和分类，统一作为一个回归问题来处理，是端到端的目标检测与目标识别卷积神经网络。能一次性预测多个 bounding box 的位置及类别，其最大的特点就是速度快。通常来说，目标检测问题实际上就是单一的回归问题，因此可以直接从图像像素边界框的坐标和类概率进行分析。YOLO 没有采用滑窗或者提取 proposal 的方式来训练网络，而是直接采用整图训练模型。这样处理的优点是可以更好的把目标和背景区域区分开。相比而言，Fast R-CNN 采用 proposal 训练方式常把背景区域错误的检为特定目标，YOLO 在加快检测速度的同时降低了一些精度。YOLO 检测系统如图 2.11 所示。

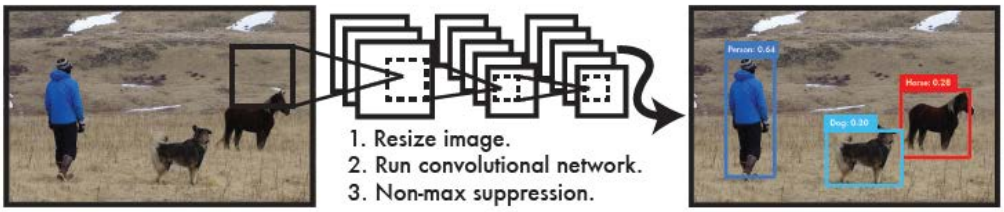


图 2.11 YOLO 检测系统

Fig 2.11 The YOLO detection system

下面重点介绍 YOLO 的原理，整个过程如图 2.12 所示。

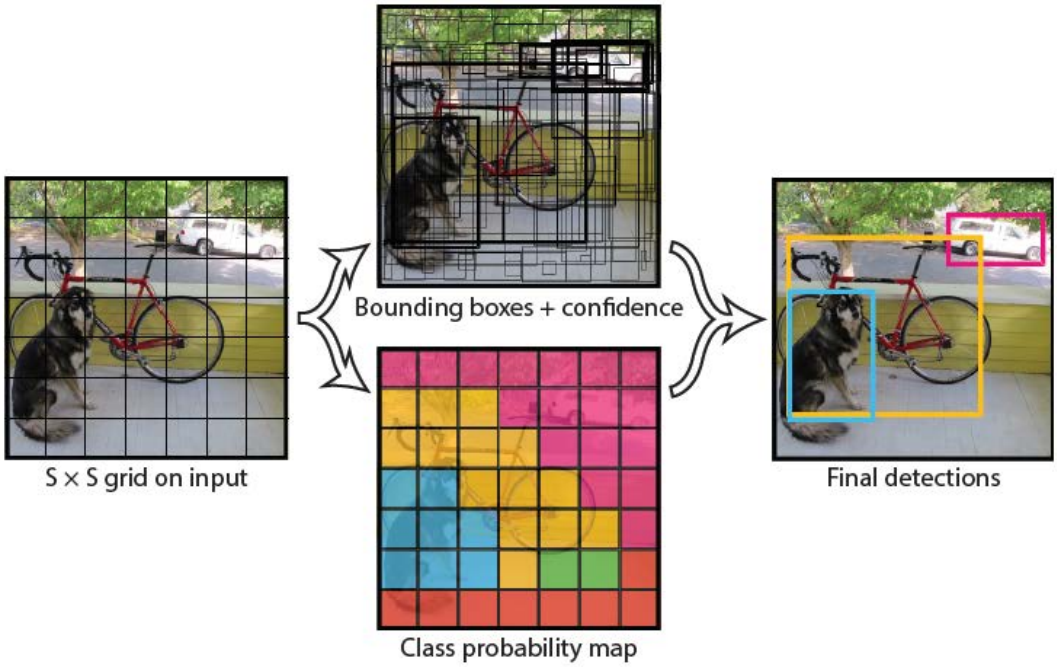


图 2.12 YOLO 网络检测原理

Fig 2.12 Detection principle of Fast R-CNN

YOLO 可以实现端到端的训练与实时检测，每一图像被划分成 S*S 个网格，如果图片中目标中心在某个网格(cell)之内，那么该网格就负责检测这个目标。由于每一个网格对应的目标的长宽比可能不一样，在训练设计 YOLO 神经网络的时候，预测 5 个参数，分别是 bounding box 的中心点坐标(x, y)，宽高（w, h）和置信度。这里的置信度评分(Pr(Object)*IOU(pred|truth))综合反映基于当前模型 bounding box 内存在目标的可能性 Pr(Object)和 bounding box 预测目标位置的准确性 IOU(pred|truth)。如果没有目标存在于 cell 中，则 Pr(Object)=0。在存在目标的前提下，YOLO 网络根据预测的 bounding box 和真实的 bounding box 计算 IOU，同时

会预测该物体属于某类的后验概率 Pr(Class_i|Object)。在预测物体类别概率的时候，每一个网格只预测一次。图像被划分为 7*7 网格（S=7），在每个网格中预测 2 个 bounding box (B=2)，总共有 20 个物体类别，所以最终用于预测的向量长度为 S*S*(B*5+C)=7*7*30，从而完成检测和识别任务。

2.3.1 网络设计

YOLO 网络从初始卷积层中提取图像特征,在全连接层预测输出概率和坐标。网络设计遵循了 GoogleNet 图像分类模型的思想，但又有些区别。YOLO 由 24 个卷积（conv）层和 2 个全连接（fc）层组成，其中 conv 层的卷积核主要有两种，一种是 3*3，另外一种是 1*1。最后一个 fc 层即 YOLO 网络的输出，长度为 S*S*(B*5+C)=7*7*30。此外，还设计了一个简化版的 Tiny-yolo 网络，包括 9 个级联的 conv 层和 2 个 fc 层，由于 conv 层的数量少了很多，因此 Tiny-yolo 速度比 YOLO 快很多。YOLO 网络的架构如图 2.13 所示。

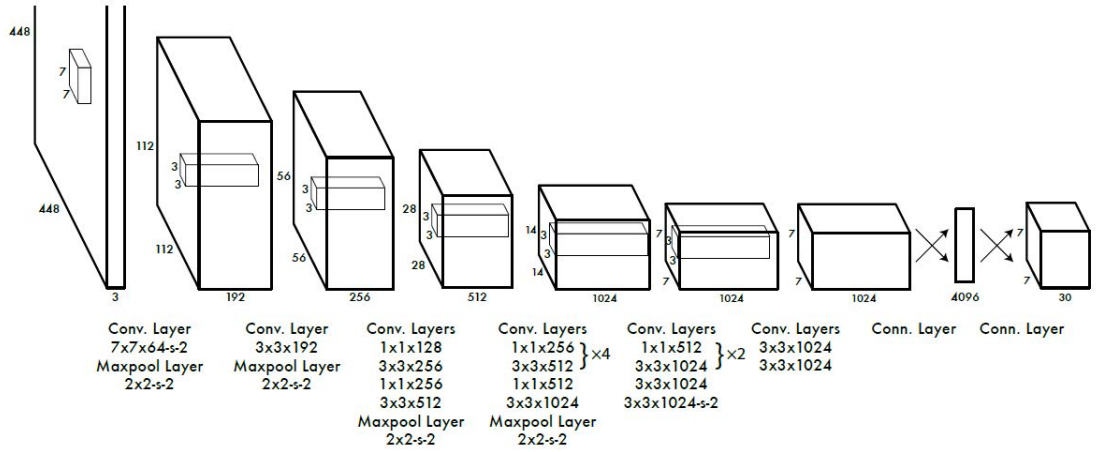


图 2.13 YOLO 网络构架

Fig 2.13 The architecture of YOLO

2.3.2 训练

训练 YOLO 网络是分步骤进行的：首先，从 YOLO 网络中取出前 20 个 conv 层，然后自己添加了一个 average pooling 层和一个 fc 层构成一个新的网络，让新网络在 ImageNet 进行预训练，得到的准确率是 88%。然后，将预训练模型的前 20 个 conv 层提取出来后，再添加了 4 个新的 conv 层和 2 个 fc 层，整体再进行检测训练。添加新层时都采用随机初始化的方式进行初始化各层权重值，初始化过后，使用 fine-tune 进行训练。在 fine-tune 新层时，改变图片的输入大小为 448*448，得到最终的训练结果，最后一个 fc 层负责预测物体的类别和位置。bounding box 主要使用中心点坐标（x,y）和宽高(w,h)来描述。Bounding box 的中心点坐标 x, y

是相对网格的相对坐标,而 Bounding box 的宽高是图像宽高归一化后得到的相对宽高。x, y, w, h 均被归一化到 0 和 1 之间。

在设计 Loss 函数时，采用的是平方和误差函数。这种方式便于优化，但是存在两个问题：一是这种误差函数没有考虑位置误差和分类误差的差异。二是一张图片分成 S*S 个网格，不是所有的网格都包含物体，那么对于没有包含物体的网格，分类概率为 0，这会在训练的时候导致模型不稳定。为了解决上述问题，采用了一系列方案：

① 首先给定位误差和分类误差赋予不同的权重，分别是增加 bounding box 坐标预测的 loss 权重，$\lambda_{coord}=5$；降低 bounding box 分类的 loss 权重，$\lambda_{noobj}=0.5$。

② 平方和误差对于大和小 bounding box 的权重是相同的，为了降低大小不一的 bounding box 宽高预测的方差，采用了平方根形式计算宽高预测 loss，即 sqrt(w) 和 sqrt(h)。

2.3.3 测试

YOLO 网络是运用 PASAL VOC 图像测试训练得到的，每幅图经过预测能得到 98 个（7*7*2）bouding box 和其相对应的概率。一个物体所对应的 bounding box 通常被一个 cell 直接预测出来。但是以下两种情况需要多个网格预测的结果经过非极大抑制处理后生成，一是尺寸较大的物体，二是靠近图像边界的。虽然 YOLO 相比 R-CNN 和 DPM，对非极大抑制的依赖性没那么强，但经过非极大抑制处理后可以将 MAP 提高大约 3 个点。

YOLO 神经网络这个结构执行检测的时候速度非常快，基础的 YOLO 网络能够达到每秒 45 帧，而 Tiny-yolo 能达到每秒 155 帧。YOLO 网络目前还存在的问题是对目标的定位准确性和精度比 Faster R-CNN 低。但是，检测速度非常快，非常适合应用在嵌入式等低功耗设备上。

2.4 卷积神经网络加速研究现状

硬件加速就是使用硬件计算模块取代现行 CPU 模块来运行算法，充分挖掘硬件的并行特性，大幅提升其性能。目前，市场上常见的硬件加速技术主要依托于集成电路 ASIC 和专用现场可编程逻辑门阵列 FPGA 等[32]。在 2009 年，Sankaradas 等人通过优化硬件第一次实现了卷积神经网络的加速，加速比高出 CPU 30 多倍。

当硬件加速受限于当前技术而难以进一步提高时，大量的研究人员转向研究加速算法。例如，Mathieu等人[33]通过傅立叶变换把卷积计算转换成点乘操作，加速比可以提高2倍多。Ranzato[34]等人证实了神经网络中一部分子集能够精确预测每层的参数，依据这个结论，Denton[35]等人为了实现降低计算量的目标，将卷积层的卷积核进行了近似处理，这种做法可以将加速比提高2到3倍。有学者针对机器

学习的通用计算部分使用ASIC，设计出一个高吞吐率的通用加速器DianNao[36]，该加速器强调内存读取优化，吞吐率能够达到452GOP/s，相比2GHz SIMD的处理器可以实现118倍的加速比。随后，以DianNao为基础，又开发出了依然针对机器学习通用计算部分的超级加速器DaDianNao[37]，加速器由64个芯片构成，而每个芯片内部都有16个近似于DianNao芯片的计算单元，不同于DianNao设计的是把权值数据缓存在每个计算单元周围的4个eDRAM中。DaDianNao的性能相对于GPU有约450倍提升，并且能够降低150倍的能耗。在神经网络软件加速方面，有研究者提出了Caffe的编程框架，该框架具有简明、易修改的特点，并优化了神经网络的计算过程[38]。此外，采用GPU的CUDA编程框架可以为深度可信网络的预训练和全局训练算法加速，相对于CPU来说，本地预训练可实现22倍的加速，而全局训练的性能提升了33倍左右[39]。在FPGA的加速方面，Daniel等人[40-42]设计实现了单FPGA的加速，他们通过使用一个嵌入式处理器和消息传递接口来控制运算单元之间的数据传输和计算流程来加速限制波兹曼机算法。Kim等人[43-44]也是针对波兹曼机提出了环状FPGA加速，使用易扩展的环状FPGA来并行处理部分神经元计算以达到加速整个算法的目标。文献[45]通过优化卷积神经网络算法，压缩结点与权值数据的位宽，设计并实现了针对预测过程的FPGA加速器。Jaderberg等人使用低秩矩阵分解的方式来加速线性卷积的计算，Liu等人则通过结合低秩矩阵分解与稀疏约束的方法实现加速线性卷积的计算，Vasilache等人通过在GPU环境中使用最简单的基-2 FFT算法实现了线性卷积的加速。目前，大型的卷积网络系统基本上都是在GPU环境下实现的，尽管GPU能够实现实时处理，然而其功耗大、成本高，难以满足一些低功耗低成本应用领域的要求。

2.5 本章小结

本章首先介绍了卷积神经网络的结构和原理，其次介绍了目前几种主流的目标检测神经网络模型并对比了它们各自的优缺点，然后从检测方案、网络设计、训练和检测速度四个方面详细介绍了 YOLO 卷积神经网络，最后介绍了卷积神经网络加速的研究现状。

3 Tiny-yolo 卷积神经网络加速算法研究

本章通过深入分析 Tiny-yolo 卷积神经网络的结构，挖掘其并行特征，并详细阐述了各层卷积计算和池化计算的优化方案。论文关于 Tiny-yolo 网络的优化加速方案主要体现在以下几个方面：

① 优化了 Tiny-yolo 网络的输入输出和网络权重的数据类型。

② 优化了 Tiny-yolo 网络的卷积计算方法。

③ 优化了 Tiny-yolo 网络的池化计算方法。

整个加速方案的设计流程如图 3.1 所示。

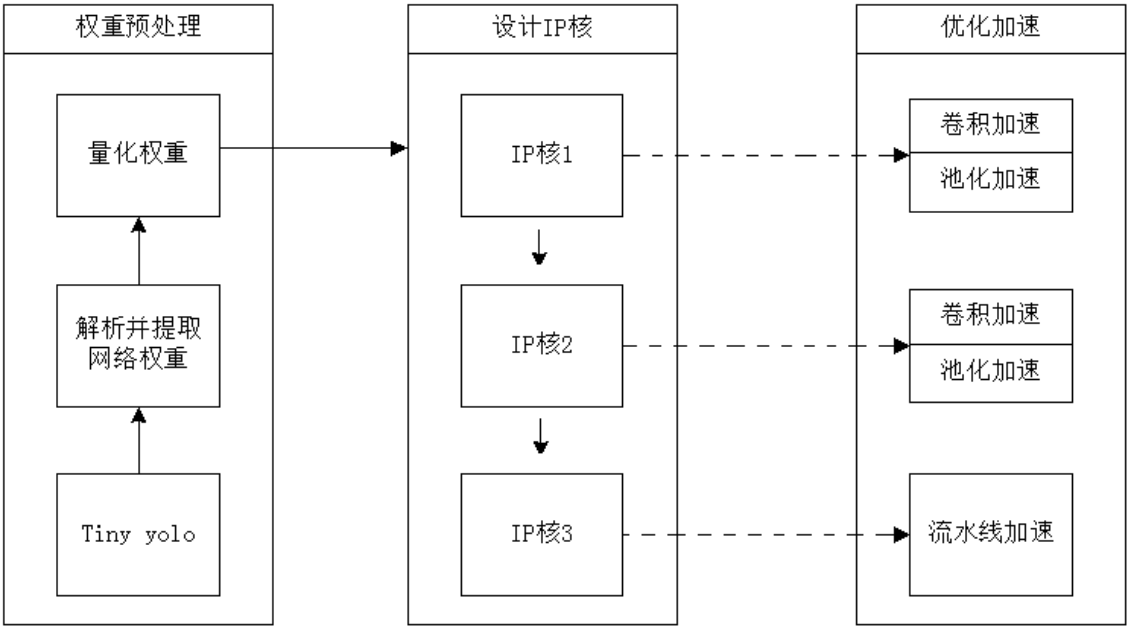


图 3.1 总设计流程图

Fig 3.1 The process of total design

3.1 基于 Tiny-yolo 网络的改进

Tiny-yolo 是一种基于端到端的目标检测框架，将目标检测问题转换成回归问题，利用整个网络同时完成目标分类和目标定位。从实验效果来看，Tiny-yolo 网络不仅检测速度远远快于其它目标检测网络，而且在检测精度方面也表现良好。但是为了让其能更好的工作于低功耗低成本的嵌入式设备之上，还需要结合硬件特点对算法进行进一步的优化。

① Tiny-yolo 卷积神经网络各层的输入和输出和网络权重均是浮点数据，在进行数据缓存和数据传输的时候，浮点数据会占用硬件平台大量的存储资源。

② 卷积层是 Tiny-yolo 卷积神经网络涉及大量计算的核心部分，卷积计算过程具有并行的特点，非常适合在硬件上进行并行加速。而卷积计算的本质是输入数据和网络权重乘加运算，若采用浮点数据就会耗费硬件平台大量的计算资源。

③ Tiny-yolo 卷积神经网络在检测过程中，池化操作需要在卷积操作完成之后进行，两者之间存在较大的数据传输开销。

针对以上问题，结合之前对 Tiny-yolo 网络和硬件资源的整体分析，本论文为实现网络在硬件上的加速作了以下改进：

1）对输入输出数据和网络权重进行了量化。神经网络各层的输入输出和网络权重都是浮点类型的数据，整个网络的计算集中在卷积过程，卷积计算是一个乘加过程，如果将数据的传输和计算映射到硬件上，则浮点类型数据会占用大量的存储资源（比如 BRAM）、消耗大量的计算资源（比如，DSP48）。基于 ZC702 硬件资源考虑，将 float 数据类型量化为 char 数据类型。同时，为了确保检测目标的准确性，在最后的输出层进行数据的反量化。数据量化既节省了资源，又提高了运算速度。

2）优化了 Tiny-yolo 网络的卷积计算方法。卷积层是卷积神经网络的核心，但由于卷积层的卷积运算很耗时，几乎占用了网络的全部运算时间。因此通过优化卷积神经网络在硬件中的卷积计算方式，来加快整个网络的卷积计算速度。

3）优化了 Tiny-yolo 网络的池化计算方法。卷积计算之后,需要对数据进行池化。池化处理的好处就是减少特征和参数，并且可以保持某种不变性（比如旋转、平移、伸缩等）。神经网络通常的计算方式是完成卷积操作之后,再进行池化操作，这种情况下会造成池化计算的延后。优化后的 Tiny-yolo 网络采用的是边卷积边池化的方式，大大节省了卷积操作和池化操作之间的数据传输时间。

3.2 数据的量化与反量化

神经网络的数据量主要集中在卷积层的输入输出以及卷积核的参数。通过分析，其中 conv0 层的输入数据就是待检测图片的归一化数据，在输入到神经网络之前会被调整到 416*416 的尺寸，而输出数据量与卷积核的个数成正比，conv0 之后紧接着是最大池化,最大池化会将卷积输出的数据量减小 4 倍，从而减小了整个网络的数据量和特征图的大小。各层输入输出尺寸和卷积核大小如表 3.1 所列，从表格中可以看出，前几层的输入数据量大，其中 conv0 的输入数据量最大，与输入数据量相比，卷积层的输出数据量更大，卷积核的参数数据量也是依次递增。巨大的数据量会占用大量的硬件资源，为了节约传输成本，并考虑到原始数据均是 4 个字节的 float 型数据，本论文采用了最大量化（即将绝对最大值量化到 127，

其它数据依次类推）的方式将 4 字节的 float 数据映射到 1 字节的 signed char 数据，然后传输到网络中进行计算。

表 3.1 各层输入输出尺寸和卷积核大小

Table 3.1 Each input and output size and convolution kernel size

layer	Input size	Filter size	Output size
Conv0	416 * 416 * 3	3 * 3 * 3 * 16	416 * 416 * 16
Maxpool0	416 * 416 * 16		208 * 208 * 16
Conv1	208 * 208 * 16	16 * 3 * 3 *32	208 * 208 * 32
Maxpool1	208 * 208 * 32		104 * 104 * 32
Conv2	104 * 104 * 32	32 * 3 * 3 * 64	104 * 104 * 64
Maxpool2	104 * 104 * 64		52 * 52 * 64
Conv3	52 * 52 * 64	64 * 3 * 3 * 128	52 * 52 * 128
Maxpool3	52 * 52 * 128		26 * 26 * 128
Conv4	26 * 26 * 128	128 * 3 * 3 * 256	26 * 26 * 256
Maxpool4	26 * 26 * 256		13 *13 *256
Conv5	13 * 13 * 256	256 * 3 * 3 * 512	13 * 13 * 512
Maxpool5	13 * 13 * 512		13 * 13 * 512
Conv6	13 * 13 * 512	512 * 3 * 3 * 1024	13 * 13 * 1024
Conv7	13 * 13 * 1024	1024 * 3 * 3 * 1024	13 * 13 * 1024
Conv8	13 * 13 * 1024	1024 * 1 * 1 * 125	13 * 13 * 125

3.2.1 量化 weights 与 biases

weights 即卷积核的参数，也是整个神经网络中数据量最大的参数。为了选出合适的量化参数，通过观察 weights 散点图，选择其量化系数。

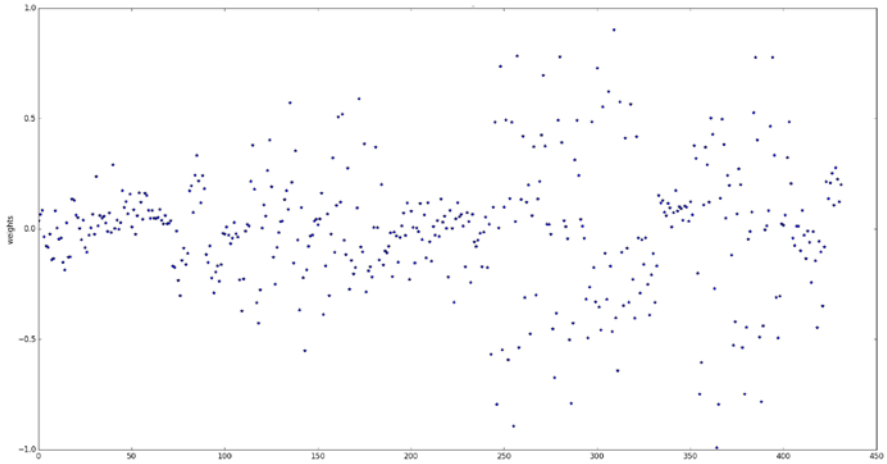


图 3.1conv0 的 weights 分布图

Fig 3.1 Weights distribution of conv0

通过观察图 3.1，Conv0 的 weights 分布是比较分散的，而且 0 值相对较少，如果把所有值都量化到-127 到 127 之间会造成较大的误差，但是由于 weights 的个数较少（总共 3 * 9 * 16=432 个数据），所以实际误差相对较小，图 3.2 是量化前后 conv0 层的 weights 频率分布图。

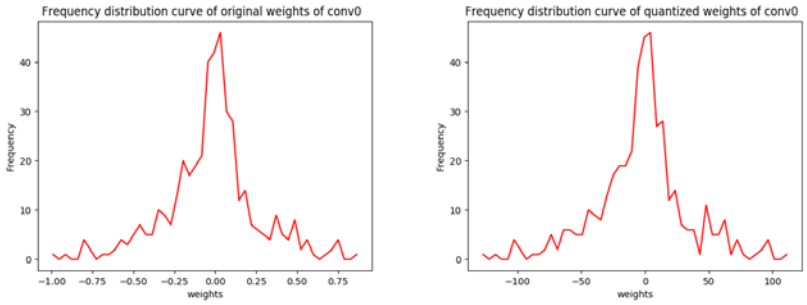


图 3.2 量化前后 conv0 层的 weights 频率分布图

Fig 3.2 Frequency distribution of original and quantized of conv0

从上述频率分布可以看出，量化后的频率分布和原始频率分布基本一致。

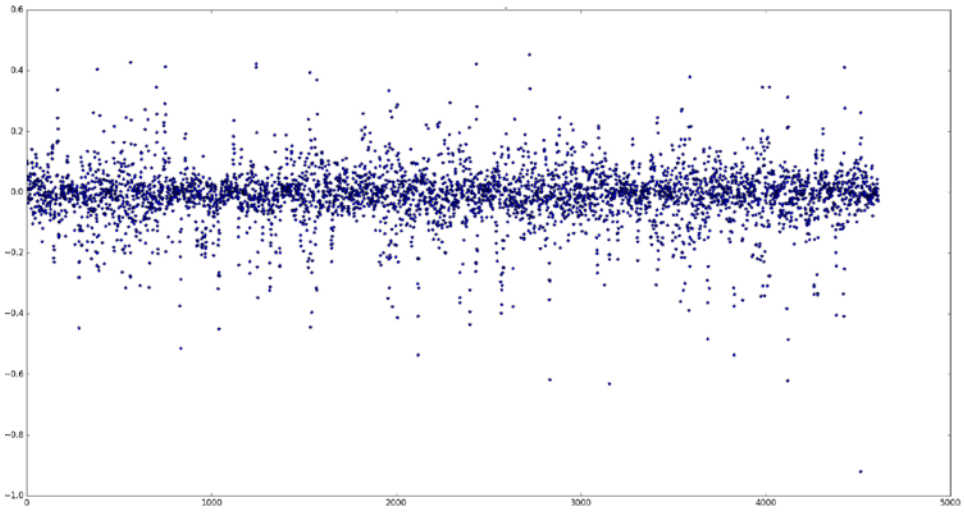


图 3.3 conv1 的 weights 分布图

Fig 3.3 Weights distribution of conv1

通过观察 conv1 的 weights 散点图 3.3，发现其绝对值最大的那个数据点（图中右下角）与其他数据点的距离较远，为了保证数据能够相对均匀的量化到[-127,127]，本论文选择次绝对最大值作为量化基准，计算该层 weights 的量化系数，量化前后频率分布如图 3.4。

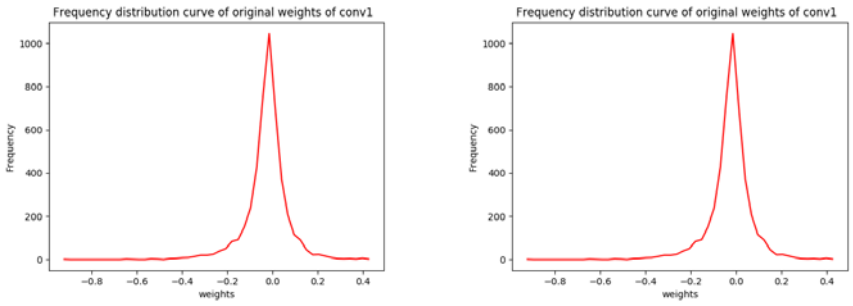


图 3.4 量化前后 conv1 层的 weights 频率分布图

Fig 3.4 Frequency distribution of original and quantized of conv1

conv2 与 conv1 的 weights 分布图类似，如图 3.5 所示。零点附近的值更多，更加有利于进行量化，本论文依然选择次绝对最大值作为量化基准，将右上角的数据进行截断量化，量化前后频率分布图如 3.6 所示。

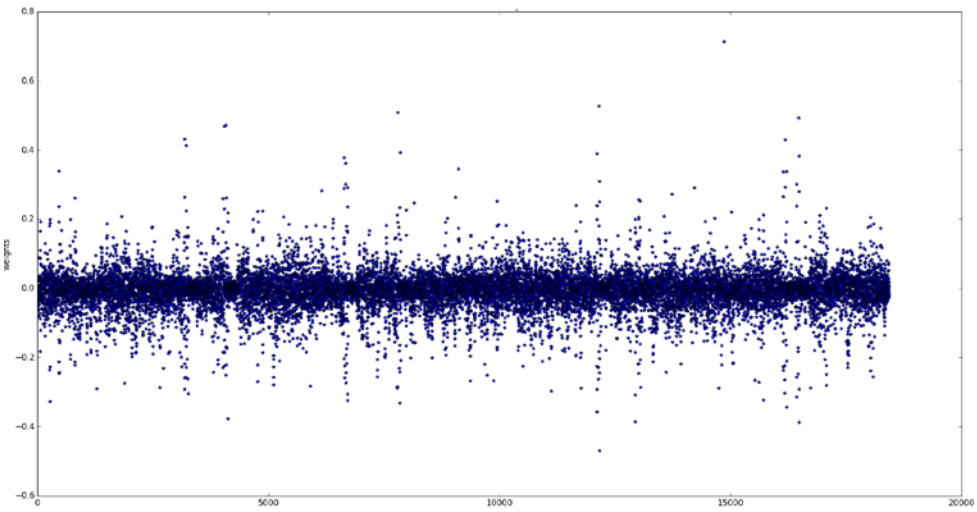


图 3.5conv2 的 weights 分布图

Fig 3.5 Weights distribution of conv2

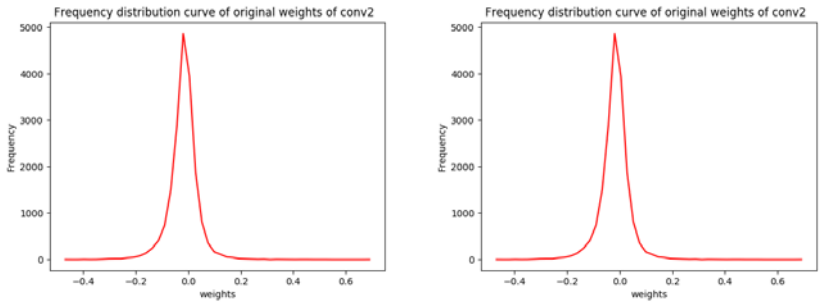
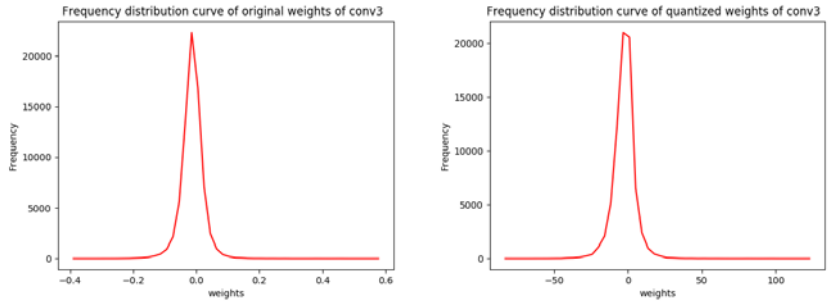


图 3.6 量化前后 conv2 层的 weights 频率分布图

Fig 3.6 Frequency distribution of original and quantized of conv2

对于 conv2~8 各层，采取跟前面几层相同的量化方式，如果绝对最大值点离其他数据点距离太远，就进行截断，并选择次绝对最大值点作为计算量化系数的依据。后续各层的 weights 频率分布图如 3.7 所示：



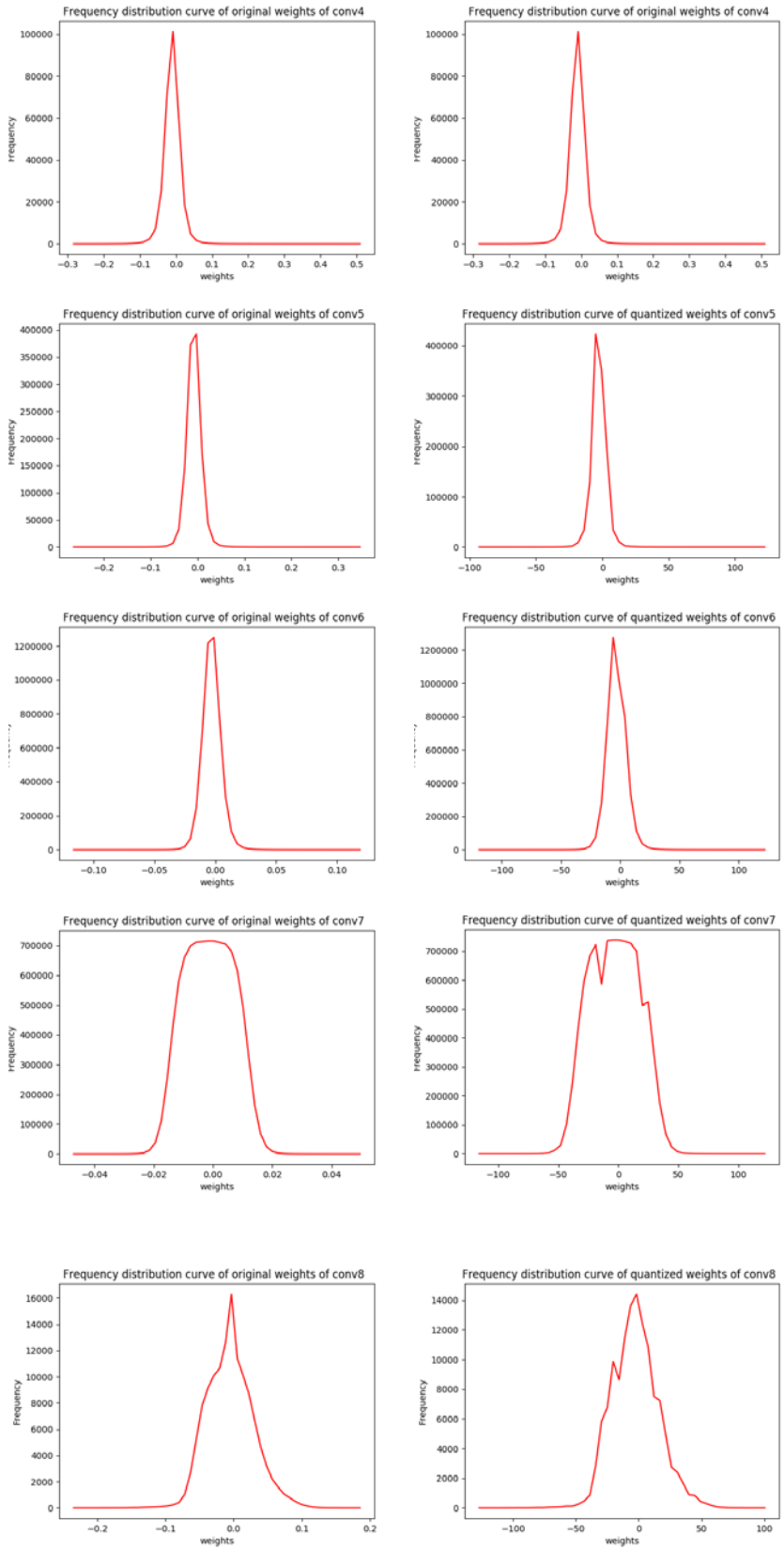


图 3.7 量化前后 conv3~8 层的 weights 频率分布图

Fig 3.7 Frequency distribution of original and quantized of conv3~8

从 weights 频率分布图可以看出，前六层的曲线基本上是吻合的，只在 conv7 层中量化曲线跟原始曲线有较大差距，主要原因是该层 weights 数量是最多的（1204 * 3 * 3 * 1024），用[-127, 127]这个范围共 255 个数据来还原它的原始分布就会造成精度不够，除此之外论文采用的量化方法能够基本反映原始 weights 的分布情况，结合之后的实验结果，证明该量化方法是可行的。

对 biases 的量化采用与量化 weights 同样的方法，通过实验发现，量化 biases 后对整体的计算速度没有明显提升，但是会造成神经网络的检测精度降低。由于 biases 数据量较小，对硬件资源的消耗较少，权衡之下，决定不量化 biases，而是采用原始数值进行计算。

3.2.2 量化输入输出数据

conv0 的输入即图片的输入，而图片在输入神经网络之前会归一化到[-1, 1]，那么 conv0 的输入数据的量化系数始终是 127.0。对于 conv1~conv8，由于各层的输入数据均是上一层的输出数据，会随着输入图片的不同而不同，与 weights 的固定分布相比是一个动态变化的数据分布。鉴于此，通过统计 100 张图片产生的各层输入输出数据，取各层输入输出的最大值，然后计算其量化系数，这种方法在最终实验的测试中证明是有效的。

3.2.3 反量化

论文中通过将原始的 float 类型数据量化到 signed char 类型的数据后，传入到 Tiny-yolo 卷积神经网络中，从硬件传输成本看，能降低 4 倍的数据传输量。数据以 signed char 类型传入神经网络中后需要进行相应的计算，若要表达神经网络原本的意义就需要将量化后的数据变换成原始数据，即需要进行反量化。本论文中尝试过两种反量化方法，一种是逐一反量化，另一种是统一反量化。逐一反量化的方法是将每一个传入的 signed char 数据进行反量化成 float 数据然后进行卷积计算；统一反量化的方法是让传入的 signed char 数据进行卷积计算，最后将卷积计算的结果进行反量化。相比统一反量化方法，逐一反量化的方法有两个缺点，一是将 8 位乘法运算转换成了 32 位乘法运行，消耗了大量的硬件计算资源；二是每一步的反量化过程中都存在误差损失，最终会造成误差损失累积，使得整个网络精度下降，导致最终实验结果失败。本论文最终选择的是统一反量化的方法，具体计算过程如图 3.8 所示。

由于加速过程中，唯一产生误差的环节就在于量化阶段，对于量化误差的大小，以最终测试的 mAP 值为准，若其值与原始版本的值相差不大，则认为误差在可接受的范围之内。

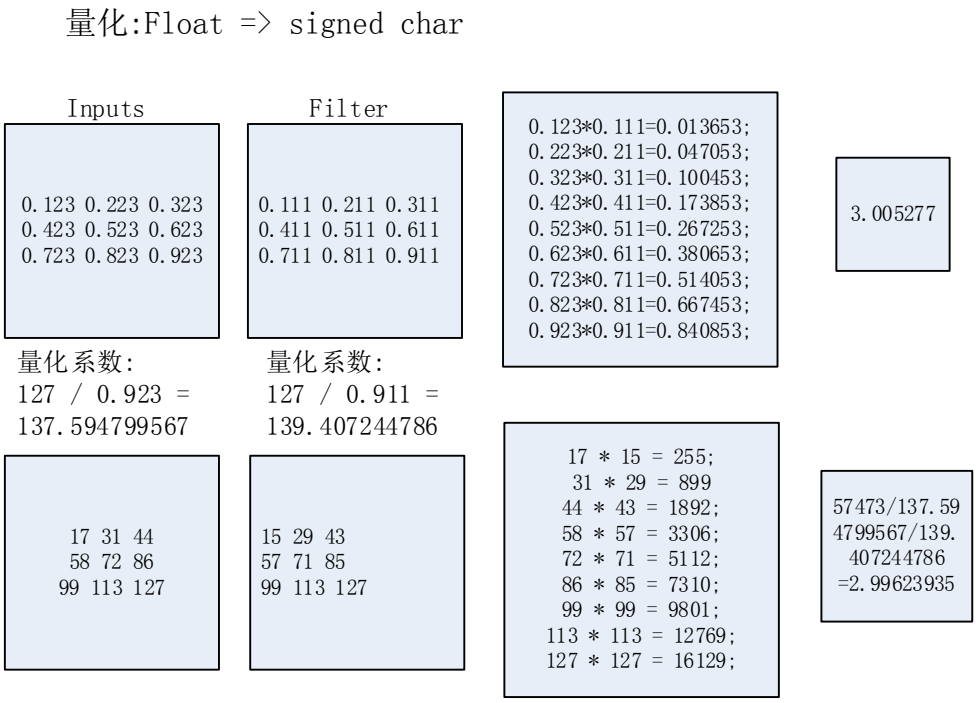


图 3.8 量化与反量化

Fig 3.8 Weights and anti-weights

3.3 卷积、池化计算方法的改进

通过分析 Tiny-yolo 神经网络结构，神经网络各层数据量大但各层都有相似之处，然后考虑到 conv0 层的输入通道数与其它层相比较为特殊，本论文为 conv0 单独设计一个 IP 核;conv8 层的卷积核的尺寸是 1*1 不同于前面所有卷积层的 3*3，所以为 conv8 单独设计了一个 IP 核；conv1~conv7 卷积层的结构相似，因此设计了一个 IP 核让各层共同调用。在 conv0~conv5 之后，都会进行相应的池化操作以减小输入特征的尺寸，为了提高硬件执行的速度，在设计 IP 核的时候将池化层和卷积层合并在一起，设计成统一的 IP 核。统一的卷积池化 IP 核比单独的卷积 IP 核和池化 IP 核有如下优势：将卷积层的数据直接用于池化过程而不需要经过额外的缓存和传输，也就是在卷积产生第一个输入的时候就开始进行池化操作，两者同时进行，无缝链接，不仅节省了传输接口资源，而且提高了处理速度。Tiny-yolo 神经网络卷积池化过程如图 3.9 所示。

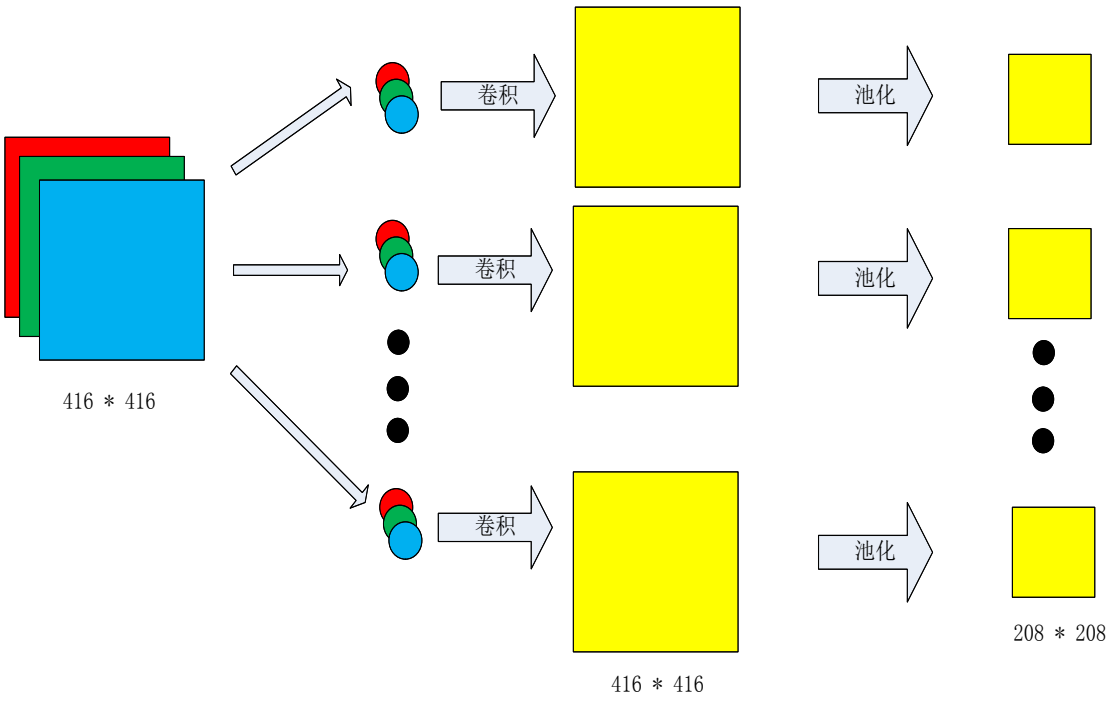


图 3.9 Tiny-yolo 卷积池化过程

Fig 3.9 The convolution and pooling process of Tiny-yolo

3.3.1 conv0_kernel

conv0 的输入数据是归一化并调整到 416*416 的图片数据，该图片数据有三个通道即 RGB 三通道，总共的数据量是 416*416*3，那么对应的每个卷积核的输入维度就是 3，而该层总共 16 个卷积核。输入数据在填充 0 边界后，通过 16 个卷积核产生 16 个特征图（feature map），由于卷积核的大小是 3*3，每次计算后卷积核窗口滑过一个像素单位，所以每个特征图的大小保持原始大小不变。得到卷积输出特征图后，紧接着进行最大池化操作，最大池化的池化核是 2*2，每次池化过后滑过两个像素单位，因此池化之后的特征图由原来的 416*416 缩小到 208*208，即缩小了 4 倍。由于 ZC702 的 BRAM 只有 280 个，每个 18k，能存储的字节数仅 645120Bytes，在 conv0 如此大的数据量的情况下，为了给后面 IP 核节省 BRAM 资源，论文在设计的时候通过增加并行度，去掉了特征图的缓存，具体设计如下：

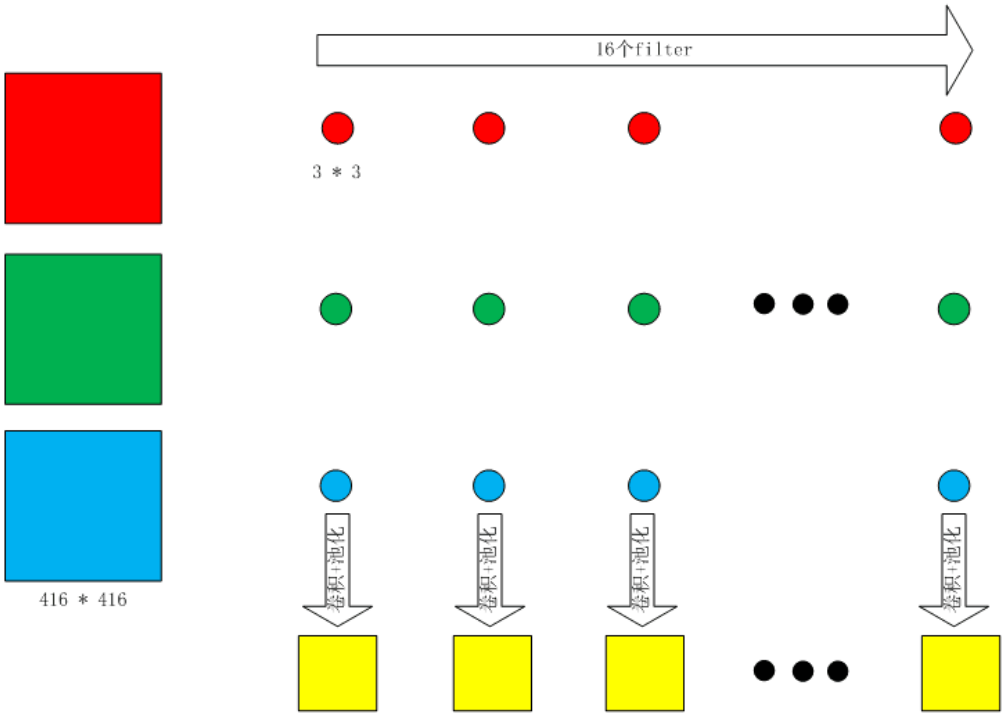


图 3.10 conv0 卷积池化过程

Fig 3.10 The convolution and pooling process of conv0

将三维的神经网络图展开成二维，数据流向如图 3.10 所示，一个卷积核需要对 3 通道的输入数据做卷积，然后池化产生最后的特征输出。对于 conv0 我们以一个卷积核为切分点，设计一个 conv0_kernel IP 核，调用 16 次即可完成 conv0 的整个计算过程。每个卷积核的每个维度的计算是相对独立的，每个通道通过卷积计算后都会产生一个跟输入同等大小的卷积输出，如果该输出来不及处理那么就必须进行缓存，但是 416*416 的图片所需要的字节数太大，同时又考虑到输入通道是 3 个通道，通道数量较少，所以在设计的时候，将 3 个通道的数据单独同时进行输入，然后在卷积过程中同时进行计算，最后将 3 个通道同时输出的卷积值进行累加得到最后的卷积值，并进行池化操作，这种方式不仅提高了 3 倍的计算速度，同时也节省 BRAM 的使用资源。

在每个通道进行卷积计算的过程中，由于需要保持卷积输出的尺寸不变，网络的实际输入尺寸为 418*418（填充 0 边界）。如果先对图片进行 0 填充，然后输入到 IP 核中，会额外增加传输的数据量，增加传输时间，因此将填充 0 的过程转移到 IP 核内部完成，边填充边进行卷积计算。此外将卷积计算的 9 次乘加运算进行并行处理，使得在一个周期内完成 9 次乘加操作，理论上可提高 9 倍运行速度，其具体实现过程如图 3.11 所示。

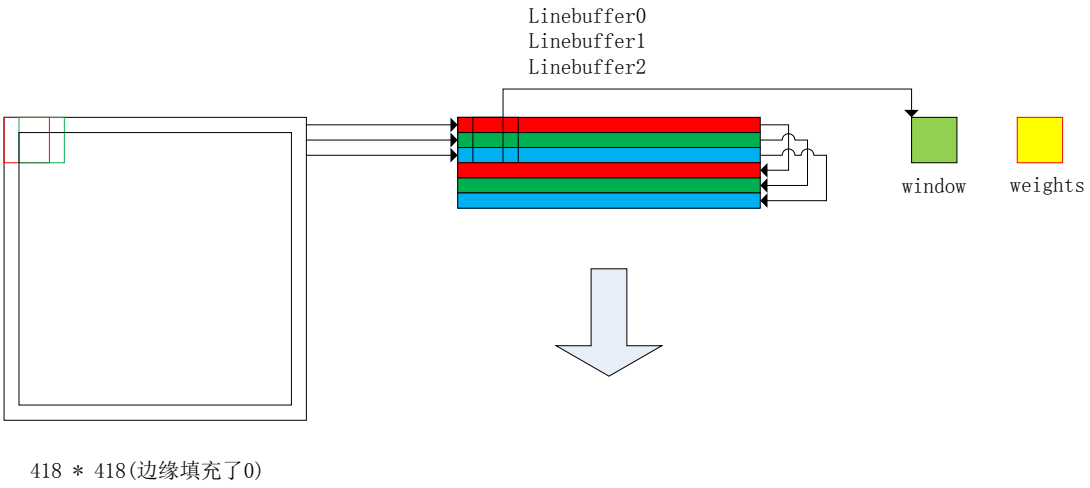


图 3.11 单通道卷积计算过程

Fig 3.11 The calculating convolution process of single channel

整个输入数据以数据流的方式流入 IP 核，输入数据流意味着每个时钟周期只能且必须读取一次数据，输出数据流意味着每个时钟周期只能且必须写入一次数据，每次读取或写入的数据是不同的数据。设计中数据都是按照数据流的方式输入或输出的，这样做的优势就是节省硬件资源，但同时提升了编程上的复杂度。如图 3.11 所示，输入数据以数据流的方式按行逐个进入 IP 核，那么在 IP 核内部就需要逐个提取然后再参与计算。为了提高处理速度，那么应该尽可能的在每一个时钟周期内进行更多的相关操作。本论文在设计时是以该原则作为优化目标，尽量在每一个时钟周期内做一次卷积计算，减少数据准备和等待过程。在上图中，输入数据的第一行、第一列、最后一行以及最后一列全是 0 元素，在卷积计算时，不需要从输入接口读取，而是直接读取常数 0。在读取数据过程中，开辟了 3 个行缓冲器（linebuffer），每个 linebuffer 的大小是 418，每个位置存储一个字节的数据，然后将这三个 linebuffer 分别声明为双端口 RAM：RAM_2P_BRAM，这样就可以在一个时钟周期内同时读写操作。在开始进行卷积计算之前，在前 418 个时钟周期内，linebuffer0 全部填充 0，linebuffer1 填充 416 个输入数据和 2 个 0，linebuffer2 填充一个 0 和一个输入数据；从第 419 个时钟周期开始，每一个时钟周期进行一次卷积计算。因为在第 419 个时钟周期的时候，卷积计算所需的 9 个输入数据已经准备完善，对应于 3 个 linebuffer 各自的前 3 个位置，然后把每个 linebuffer 的第 1 个和第 2 个元素(索引以 0 开始)一次性推送到 window 中，window 第一列在初始时均为 0，window 通过 HLS 指令声明为 9 个寄存器，然后与缓存 weights 的另外 9 个寄存器进行乘加操作，通过指定 HLS 的 pipeline 优化指令，实

现一个时钟周期完成 9 次卷积计算。在完成一次卷积计算的同时，将 window 每行进行左移一位，用于准备暂存下一个时钟周期的数据。在第 412 个时钟周期时，读取一个新的输入到 linebuffer2 的第 3 个位置，然后将 3 个 linebuffer 的第 3 个位置的数据推送到 window 的最后一列，然后进行卷积计算。依次类推，当计算完前 3 行后，将 linebuffer0 移动到 linebuffer2 后面，作为新输入数据的缓存器，而 linebuffer1 和 linebuffer2 中的数据继续保持并使用，最后重复上述过程，实现卷积计算的加速。如果不进行加速，理论上 conv0 的串行执行所需时钟周期是 418 * 418 * 9，而使用 pipeline 加速后所需时钟周期是 417*417，执行速度提升约 9 倍，由于输入的 3 个通道的数据是并行执行的，所在 conv0 理论上可提高 27 倍加速。

设计时将池化嵌入到了卷积过程中，每次产生一个卷积输出值，就开始考虑池化的过程。在前 416 个时钟周期内，每输出两个卷积值，就选择其中最大的一个值缓存在一个 linebuffer 中，在第 417、418 个时钟周期时，取出两者卷积值中的最大值，然后和缓存器 linebuffer 中暂存的前一行的对应位置的最大值进行比较，选择出最终的最大值，以此类推完成整个池化操作。图 3.12 为单通道池化计算过程。

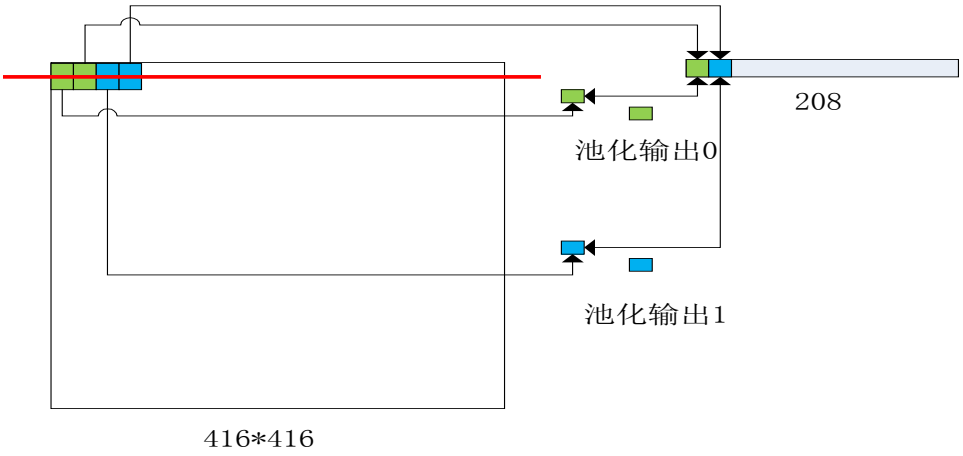


图 3.12 单通道池化计算过程

Fig 3.12 The calculating process of single channel pooling

3.3.2 conv1-7_kernel

如图 3.13 所示，以 conv1 为例，输入通道数增加了很多，如果同时处理这么多的输入数据，会造成计算资源不够，而导致整个 IP 核没法正确生成。conv1 每个通道的卷积输出的尺寸为 208 * 208，与 conv0 的 416 * 416 相比缩小了 4 倍，根据硬件资源，可以将单通道的卷积输出值缓存下来。那么输入数据的 16 个通道逐个输入 IP 核内，每次进行单通道卷积计算，然后将该通道的卷积输出值缓存在

208*208 的缓存器中，在进行下一个输入通道的卷积计算时，重复使用该缓存器，并进行相应位置元素的累加操作，然后重复该过程。当计算最后一个通道时，在得到每个位置的最终卷积值的同时，就进行相应的池化操作。由于 conv6 和 conv7 中不涉及池化操作，那么卷积输出值就是最终的输出值。而整个卷积计算过程和池化过程与 conv0_kernel 类似，由于输入通道未采用并行处理，所以理论加速主要集中在卷积过程即相对串行计算提高 9 倍的速度。

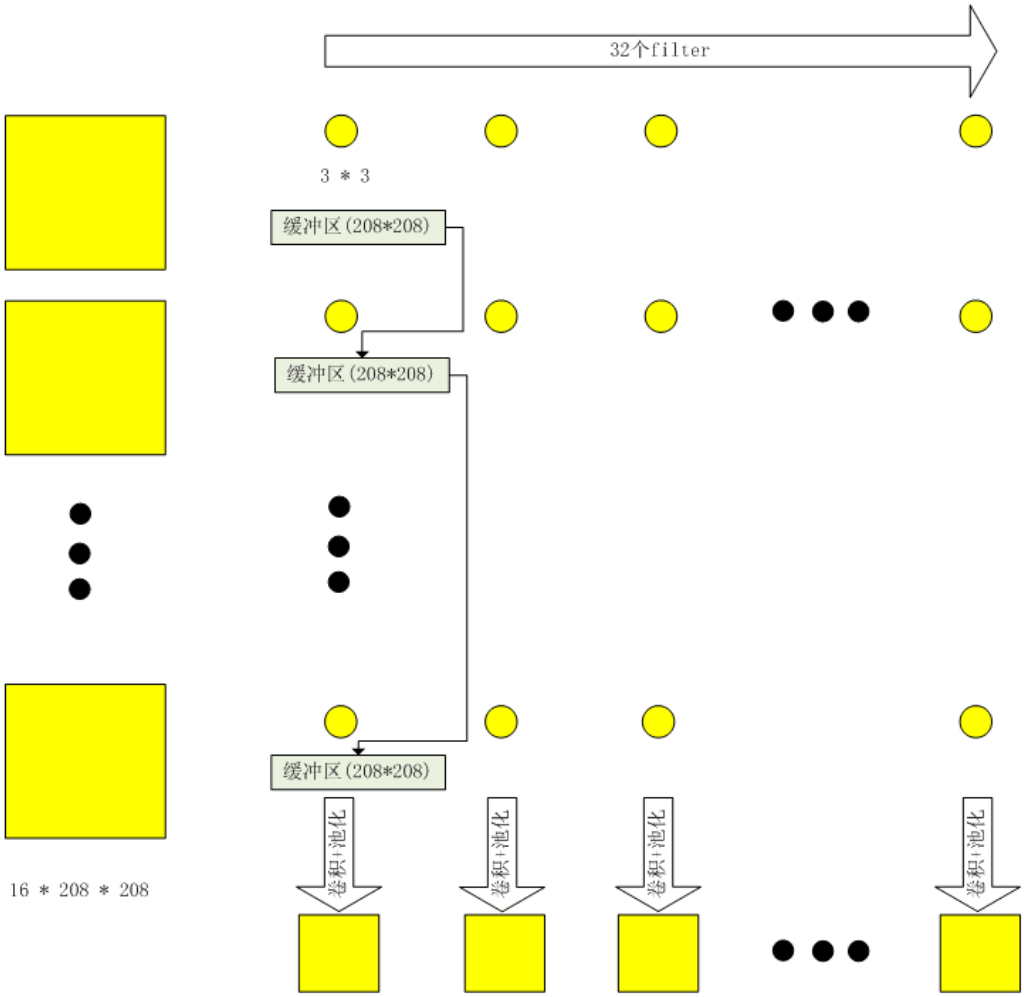


图 3.13 conv1~7 卷积池化过程

Fig 3.13 The convolution and pooling process of conv1~7

3.3.3 conv8_kernel

由于 conv8 层的卷积核的大小是 1 * 1，相当于数值的一一映射，而本论文采用的方法主要集中在卷积计算过程中，所以 conv8_kernel 的加速并不太明显，但是使用 pipeline 优化后依然比在 ARM 中执行的速度快。

3.4 本章小结

本章首先对 Tiny-yolo 网络的数据传输和计算瓶颈进行分析，然后结合硬件资源和卷积池化计算的特点，逐层针对性的进行优化加速。最后详细的阐述了算法加速的实现原理、具体的流程以及核心思想。

4 硬件加速实现

4.1 HLS 工具介绍

高层次综合工具（High-level Synthesis tool, HLS）是 Xilinx 在 2012 年发布的一套集成于 Vivado 开发环境的开发工具，主要目的是用于可编程逻辑器件（比如 FPGA）的设计和开发。使用 HLS 高层次设计工具，用户可以选择多种不同的高级语言（如 C，C++，System C）来进行 FPGA 的设计，通过仿真、优化及综合等步骤就可以以 RTL 代码的形式输出，既可以是网表形式，也可以导出为 Xilinx 的 IP 核。FPGA 设计是基于硬件描述语言且以数字逻辑设计为核心，和传统的计算机软件编程设计思路不同。正是因为这种程序设计思路的不同，对开发 FPGA 造成了很大的难度。采用 Verilog 或者 VHDL 进行算法或者复杂时序的 FPGA 设计时，往往需要耗费很多的时间和精力来进行严格的测试。产品的研发周期随着市场的激烈竞争需要越来越短，那么传统的 FPGA 设计方法及工具遭遇到了越来越大的挑战。引入 HLS 设计工具，在代码生成时可以快速优化 FPGA 硬件结构，提高执行效率，并且让设计者把重心放在功能的设计和实现上，降低开发难度，在开发过程中可以快速验证和修改算法，及时查看算法的实际效果，缩短开发周期。采用 HLS 设计的硬件电路具有良好的扩展性，并且开发过程简单，系统的设计使用 C 语言就能实现，如果开发人员不够精通硬件知识，HLS 就是很好的选择。

4.1.1 HLS 开发原理

HLS 的开发原理就是使用 HLS 工具把 C 语言设计的系统转换为硬件实现，具体实现过程：一是 HLS 工具基于 C 语言的源代码创建一个 RTL 级实现。HLS 工具可以从 C 源代码的顶层，获取控制命令与数据通道，依照默认指令和用户具体要求的指令实现设计过程。二是 HLS 经调度和绑定过程，可以将 C 代码映射到硬件逻辑。调度是用来确定操作发生在哪个时钟周期；而绑定是用来确定每个操作使用的库单元，比如元件的延迟。在这个处理过程中，充分考虑控制、数据流以及用户的命令。例如使用 C 语言编写的数组，通过内置编译器，HLS 开发工具能够直接将数组转换成寄存器。

HLS 的优势主要体现在：（1）在相同的源代码中通过不同的 HLS 描述能实现很多的设计，既能进行较小较快的系统设计，又可以对设计方法进行“探索”，找到最好的解决思路。（2）HLS 具有容易移植的功能。（3）HLS 命令的优先级主要是降低延迟和减少面积，满足性能的要求。

4.1.2 HLS 设计流程

HLS 进行设计时，主要步骤如图 4.1 所示：

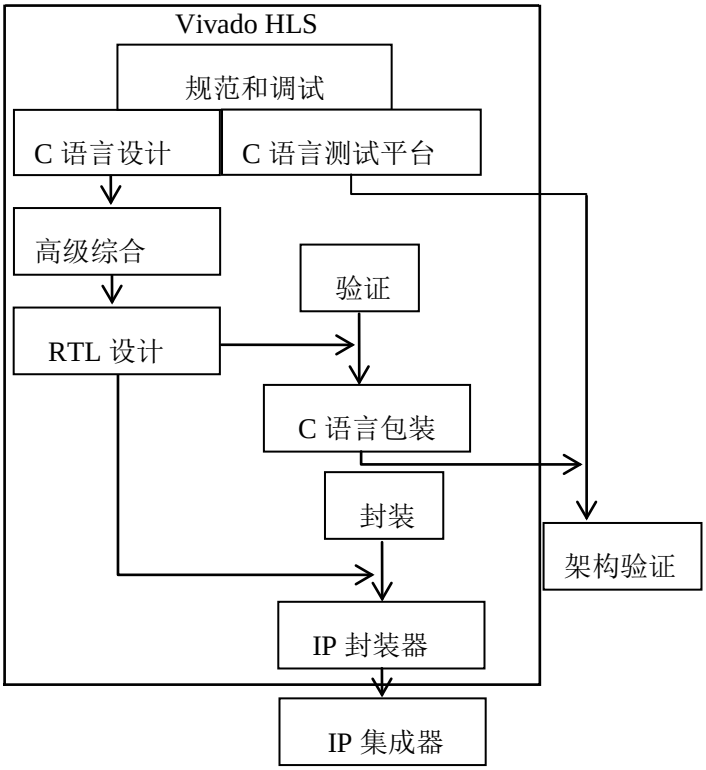


图 4.1 HLS 的设计流程

Fig 4.1 The design process of HLS

① 建立 HLS 工程。按照处理流程，确定功能模块，选择时钟周期和时序裕量。使用 C/C++等高级编程语言和 HLS 组件编写程序，设计功能模块与测试模块，创建合格的 HLS 工程。② 运行 C 仿真。在确保设计正确的前提下，创建 HLS 工程之后，就可以使用测试模块来对功能模块进行功能仿真。③ 调试代码。如果 C 仿真过程中出现问题，可以使用 HLS 中集成的调试器，对每个功能模块或测试模块进行调试。④ 综合设计。经仿真调试，保证测试模块与功能模块无误后，可以对设计进行综合，生成系列报告文件和 RTL 设计文件。⑤ 优化设计。按照设计要求，在统筹兼顾性能、面积及资源利用率等多方面因素的情况下，调整设计并重新综合，得到最满意的设计结果。⑥ 运行 C/RTL 联合仿真。C 仿真就是在高级语言层次上对测试模块及功能模块进行的仿真。采用 HLS 工具，RTL 级别的测试代码可通过 C 测试代码由 HLS 自动转化而来，在运行 C/RTL 联合仿真过程中，如果仿真测试失败，那么需要根据错误提示对设计进行修改，并重新执行综合及仿真过程；反之，就可以直接进行下一步操作。⑦ 观察仿真结果。通过 HLS 的帮助，能够对 C/RTL 联合仿真的过程进行跟踪记录，有利于掌握数字电路运行的时序细节。⑧ 导出 RTL 实现。当完成设计后，借助 HLS 的功能优势，可以通过 IP 核对“RTL 实

现”进行导出，导出的 IP 核包含了驱动、参考文档及使用示例等相关的文件。⑨ 系统集成。可以在相关的开发环境中，对生成的 IP 核进行系统集成。

4.1.3 HLS 优化设计

HLS 的优化首先需要验证 C 语言代码的功能是否正常，大致步骤可以分为：确定接口、流水线设计、优化数据结构、解决时延和面积问题，如图 4.2 所示。

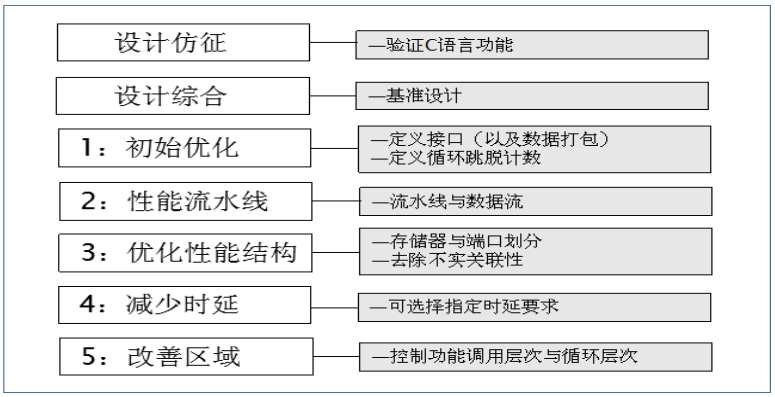


图 4.2 HLS 的优化方法

Fig 4.2 The optimization method of HLS

HLS 提供了很多优化命令可以对 C 语言设计的系统进行优化操作，改善其资源占用率及处理速度。HLS 优化策略的主要包括：对延迟和吞吐量、循环、存储数据的数组和面积进行优化。优化策略的实现主要有以下 5 个步骤：

步骤一：初始优化

表 4.1 显示的是应首先考虑加入设计内的指令，根据需要加入这些指令，有利于后续的操作和结果处理。

表 4.1 初始优化指令

Table 4.1 The instruction of initial optimization

指令和配置	描述
INTERFACE	指示如何根据功能描述创建 RTL 端口
DATE-PACK	把结构（struct)的数据字段打包到有更大字宽度的单一 scalar 中
LOOP-TRIPCOUNT	用于带有变量的循环，为循环迭代计数提供估算。这对综合没有影响，只对报告功能有影响
CONFIGINTERFACE	该配置控制着与顶层函数实参不存在关联的 I/O 端口，能够从最终 RTL 除去未使用的端口

步骤二：使用流水线提升性能

在优化流程的过程中，应尽量多的采用同步操作，在对函数及循环应用采用PIPELINE 指令，对包含函数及循环的层次使用 DATAFLOW 指令，让其并行工作。当延迟影响到设计难以在要求的时钟频率上实现时，可以使用 RESOURCE 指令对一些操作进行流水线优化，比如加法器和乘法器。使用流水线提升性能的常用指令如表 4.2 所列。

表 4.2 流水线优化指令

Table 4.2 The instruction of pipeline optimization

指令与配置	描述
PIPELINE	通过让循环中或函数中的操作同时执行，降低初始化时间间隔
DATAFLOW	实现任务层面的流水线化，让函数和循环同时执行；用于最小化时间间隔
REWOURCE	指定用于 RTL 中实现变量(阵列、算数运算或函数实参的资源（内核）
CONFIGCOMPILE	让循环根据自身的迭代计数自动流水线化

步骤三：优化结构以提高性能

在流水线化循环和函数时，可能会遇到其它问题，比如需要把大阵列划分为小阵列。当出现这些问题时，可以采用表 4.3 中的指令，有利于解决数据结构中的瓶颈问题。

表 4.3 优化结构指令

Table 4.3 The instruction of structure optimization

指令与配置	描述
ARRAY-PARTMON	把大型阵列划分为多个较小阵列或划分为单独的寄存器，以改善对数据的访问，消除块状 ARM 瓶颈
DEPENDENCE	用于提供能克服循环承载关联性，让循环流水线化（或使用较低时间间隔流水线化）的额外信息
INLINE	内联函数，消除所有函数层次结构，用于跨越函数边界实现逻辑优化，通过减少函数调用开销改善时延/时间间隔
UNROLL	展开循环，创建多个独立操作而非单个操作集
CONFIG ARRAY PARTITION	该配置用于判断包括全局阵列在内的阵列划分方式，以

指令与配置	描述
	及划分是否会影响阵列端口
CONFIG SCHEDULE	确定综合调度阶段中的工作强度以及输出信息的信息显示

步骤四：降低时延

在 Vivado HLS 完成初始化时间间隔的最小化后，会自动的减小延迟，将延迟最小化。表 4.4 列出的优化指令，能够达到降低及指定的时延。通常，这些指令在循环与函数流水线化的时候不会用到，因为时延在大多数应用中并不是最重要的影响因素，而吞吐能力大小影响更大。假如循环与函数都没有进行流水化操作，那么延时就会影响吞吐能力，在一个任务未完成时不会开始下一输入。

表 4.4 降低时延指令

Table 4.4 The instruction of delay reduction

指令	描述
LATENCY	用于指定最小和最大时延约束
LOOP-FLATTEN	把嵌套循环分解为具备改进时延的单循环
LOOP-MERGE	融合连续循环，以降低总体时延，增加共享和改善逻辑优化

步骤五：缩小面积

在符合性能目标的前提下，需要在保持性能不变的情况下进一步缩小面积。比如最小化数据位宽，把较小的数组映射到较大的数组，使用当前的 RAM，可以有效控制设计的结构；内联命令可以影响 RTL 的层次与优化；使用表 4.5 中的指令，通过调用控制实现面积的优化。

表 4.5 缩小面积指令

Table 4.5 The instruction of area reduction

指令	描述
ALLOCATION	为使用的操作、内核和函数数量设定限额，这样会强制共享硬件资源并可能增大时延
ARRAY-MAP	把多个较小的阵列结合成单个大型阵列以帮助减少块状 RAM 资源数量
ARRAY-RESHAPE	把阵列从多元型阵列重塑为字宽度更大的阵列，用于在避免使用更多块状 RAM 的情况下改善块状 RAM 访问

指令	描述
LOOP-MERGE	融合连续循环，以降低总体时延，增加共享和改善逻辑优化
OCCURRENCE	在流水线化函数或循环时使用，用于让位于某位置的代码的执行速度低于其外层函数或循环中的代码执行速度
RESOURCE	指定将特定的库资源（内核）用于实现 RTL 中的变量（阵列、算数运算或函数实参）
STREAM	指定把特定的储存器通道在数据优化过程中实现为 FIFO 或 RAM
CONFIG BIND	判断综合绑定阶段的工作强度，可用于在全国层面最小化使用的操作数量
CONFIG DATAFLOW	该配置用于指定数据流优化中的默认储存器通道和 FIFO 深度

4.2 ZC702 测试平台介绍

ZC702 是 Xilinx 推出的第一款基于 Zynp@-7000 平台的评估板，如图所示：

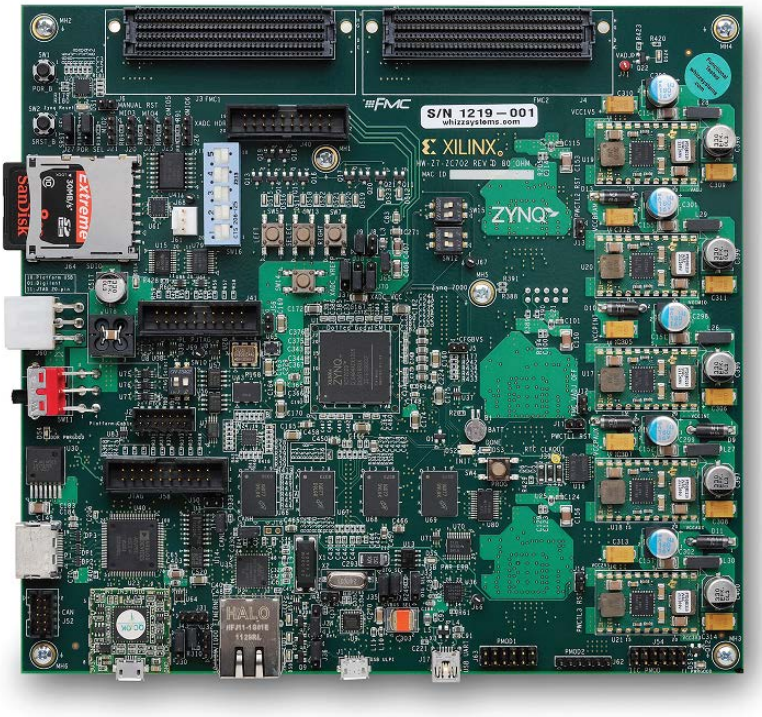


图 4.3 ZC702

Fig 4.3 ZC702

由图可见，Zynq®-7000 AP SoC ZC702 的评估套件主要包含硬件、设计工具、IP 核以及预验证参考设计的基本元件，也包括能够完成全嵌入处理平台的目标设计。

① 主要性能和优势

1）使用 Zynq-7000 SoC 能够对嵌入式应用进行快速原型设计并完成优化；2）硬件、设计工具、IP、以及预验证参考设计；3）可以演示嵌入式设计面向视频通道，高级存储接口 1GB DDR3 组件存储；4）实现 USB OTG、 UART、 IIC 和 CAN 总线的串行连接；5）可以完成含有 Dual ARM Cortex-A9 核处理器的嵌入式处理；6）使用 10-100-1000 Mbps Ethernet(GMII、RGMII 和 SGMII)的开发网络应用；7）使用 HDMI 输出实现视频显示应用；8）扩展 I/O，包含 FPGA Mezzanine Card (FMC)接口。

② 主要特性

符合 ROHS 规范的 ZC702 套件包括：XC7Z020-CLG484-1AP SoC 12V 墙式适配器具有 ATX 电压与电流测量功能的电源，主处理芯片，此芯片的 PL 端基于 Xilinx Artix-7 系列 FPGA 实现。

③ 配置

1）板上配置电路；2）16MB 四重 SPI 闪存；3）SDIO 卡接口（启动）；4）PC4 和 20 引脚 JTAG 端口。

④ 存储器

1）DDR3 组件存储器 1GB；2）支持 32 位数据宽度；3）16MB 四重 SPI 闪存；4）IIC-1KBEEPROM。

⑤ 通信与网络

1）Gigabit Ethernet GMII、RGMII 及 SGMII；2）USB OTG 1(PS)-主机 USB；3）IIC 总线头/HUB (PS)；4）支持 CAN 唤醒的 1 CAN (PS)；5）USB UART (PS)。

⑥ 视频/显示

1）HDMI 视频输出；2）8X LED。

⑦ 扩展连接器

1）FMC #1-LPC 连接器；2）FMC #2-LPC 连接器；3）IIC HUB/扩展器；4）双重 Pmod（与 LED 共享 8 个 I/O）；5）单个 Pmod（与 PJTAG 共享 4 个 I/O）。

⑧ 时钟技术

1）200MHz 固定 PL 振荡器（差分 LVDS）；2）156.25MHz（默认）I2C 可编程振荡器（差分 LVDS）；3）33.33MHz 固定 PS 系统振荡器（单端 CMOS）。

⑨ Control & I/O

1）3 个用户按钮；2）2 个用户开关；3）8 个用户 LED。

⑩ 功耗

1）12V 插墙式适配器或 ATX；2）具有电压和电流测量功能的电源。

由上面的介绍可知，ZC702 将 Zynqxc7Z020-1CLG484C 作为主处理芯片，此芯片的 PL 端基于 Xilinx Artix-7 系列 FPGA 实现，其逻辑资源如表 4.6 所示。

表 4.6 ZC702 逻辑资源表

Table 4.6 The table of ZC702 logic resource

资源类型	资源量	资源类型	资源量
FF	106400	BRAM	280
LUT	53200	DSP48	220
Memory LUT	17400	BUFG	32

之所以选择 ZC702，是因为 ZC701 资源太少，没有办法运行神经网络，ZC704 可以全完符合，但是价格太贵，ZC702 价格适中，经过网络优化，可以实现。

4.3 基于 HLS 的 Tiny-yolo 卷积神经网络加速算法实现

4.3.1 conv0_kernel 的 HLS 实现

conv0_kernel 函数接口以及端口配置如下：

```
void conv0_kernel(signed char inputs_r[416 * 416],
                  signed char inputs_g[416 * 416],
                  signed char inputs_b[416 * 416],
                  signed char weights[3 * 3 * 3],
                  float bias,
                  signed char outputs[208 * 208]);
#pragma HLS STREAM variable=inputs_r
#pragma HLS STREAM variable=inputs_g
#pragma HLS STREAM variable=inputs_b
    #pragma HLS STREAM variable=outputs
```

由于 conv0 的输入通道数较少（3 个通道），在设计 IP 核的时候将 3 个通道的输入数据同时输入，分别为 inputs_r[416*416]，inputs_g[416*416], inputs_b[416*416]，每个通道的大小就是图片的宽和高，整个网络的输入图片都会被调整到 416*416。weights[3*3*3]是一个 filter 的数据，bias 是当前 filter 对应的偏置,直接使用原始的 float 类型。outputs 的输出大小不是 416*416 而是 208*208，这

是因为设计时将池化层融合在了卷积层之中。416*416 的 RGB 图像对于 ZC702 来说十分耗资源，因此将 conv0_kernel 的硬件接口指定成 STREAM，数据的读写均按照 STREAM 的方式，这种方式的优点是节省资源且在数据量少的时候传输速度快，但是由于 STREAM 的读写限制导致在算法实现上相对更复杂。

算法伪代码：

```
    for (int height = 0; height < 416 + 1; height++) {
        for (int width = 0; width < 416 + 1; width++) {
            #pragma HLS PIPELINE II=1
            //前 418 个周期内缓冲 2 行和 1 个元素
            //从第 419 个周期开始，进行卷积计算
            //卷积计算
             for (int ch = 0; ch < 3; ch++) {
                #pragma HLS UNROLL
                for (int i = 0; i < 9; i++) {
                   tmp_conv += window[ch][i] * buffer_weights[ch][i];
                }
            }
            //池化计算
            //输出
        }
}
```

其中卷积计算部分指定#pragma HLS UNROLL 的目的是展开 27 次乘加运算，然后并行执行；该 IP 核的优化目标是#pragma HLS PIPELINE II=1，II 越小处理速度越快；由于#pragma HLS UNROLL 这个指令位于#pragma HLS PIPELINE II=1 之内，它会将循环体内的所有子循环全部展开，所以可以省略 UNROLL 指令。

主要资源分配：

```
signed char linebuffer0_r[418];
#pragma HLS RESOURCE variable=linebuffer0_r core=RAM_2P_BRAM
signed char linebuffer1_r[418];
#pragma HLS RESOURCE variable=linebuffer1_r core=RAM_2P_BRAM
signed char linebuffer2_r[418];
#pragma HLS RESOURCE variable=linebuffer2_r core=RAM_2P_BRAM
signed char linebuffer0_g[418];
#pragma HLS RESOURCE variable=linebuffer0_g core=RAM_2P_BRAM
signed char linebuffer1_g[418];
#pragma HLS RESOURCE variable=linebuffer1_g core=RAM_2P_BRAM
signed char linebuffer2_g[418];
#pragma HLS RESOURCE variable=linebuffer2_g core=RAM_2P_BRAM
signed char linebuffer0_b[418];
#pragma HLS RESOURCE variable=linebuffer0_b core=RAM_2P_BRAM
signed char linebuffer1_b[418];
#pragma HLS RESOURCE variable=linebuffer1_b core=RAM_2P_BRAM
signed char linebuffer2_b[418];
#pragma HLS RESOURCE variable=linebuffer2_b core=RAM_2P_BRAM
signed char window[3][9];
#pragma HLS ARRAY_PARTITION variable=window complete dim=0
signed char buffer_weights[3][9];
#pragma HLS ARRAY_PARTITION variable=buffer_weights complete dim=0
signed char buffer_maxpool[208];
#pragma HLS RESOURCE variable=buffer_maxpool core=RAM_2P_BRAM
```

在卷积计算过程中，根据设计方案需要使用 linebuffer 进行缓冲，每个通道需要 3 个 linebuffer，而 3 个输入通道同时输入 IP 核，那么需要 9 个 linebuffer，其资源类型是 RAM_2P_BRAM，一个时钟周期内可以读/写两次。window 是卷积最小单元的输入数据，其通过 ARRAY_PARTITION 指令声明成了寄存器，以实现一个时钟周期内同时读取 27 个数据；buffer_weights 跟 window 的数据做乘加运算，所以也被声明成寄存器。buffer_maxpool 用于池化操作过程的缓存，其长度为 208 而不是 416，是因为根据池化方案设计，在池化时只需存储一行数据的一半即可。

优化 PIPELINE：

```
INFO: [HLS 200-10] ---------------------------------------------------------------
INFO: [HLS 200-10] -- Scheduling module 'conv0_kernel'
INFO: [HLS 200-10] ---------------------------------------------------------------
INFO: [SCHED 204-11] Starting scheduling ...
INFO: [SCHED 204-61] Pipelining loop 'Loop 1'.
INFO: [SCHED 204-61] Pipelining result: Target II: 1, Final II: 1, Depth: 101.
```

根据 scheduling 的信息提示，本论文的目标 II 是 1，在综合成功后的最终 II 也是 1，说明#pragma HLS PIPELINE II=1 这个优化目标实现了。图 4.4 是 conv0 资源报表。

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	16	-	-
Expression	-	12	0	847
FIFO	-	-	-	-
Instance	-	24	8316	11796
Memory	10	-	0	0
Multiplexer	-	-	-	837
Register	-	-	2649	76
Total	10	52	10965	13556
Available	280	220	106400	53200
Utilization (%)	3	23	10	25

图 4.4 Conv0 资源报表

Fig 4.4 The resource report of Conv0

4.3.2 conv1_7_kernel 的 HLS 实现

其函数接口以及端口配置如下：

```
void conv1_7_kernel(signed char inputs[208 * 208 * 16],
                    int in_width,
                    int in_channels,
                    signed char weights[1024 * 3*3],
                    float bias,
                    signed char outputs[104 * 104]) {
#pragma HLS STREAM variable=inputs
```

```
#pragma HLS STREAM variable=weights
#pragma HLS STREAM variable=outputs
```

由于 conv1~7 共用该 IP 核，那么该 IP 核的数据输入和输出接口的大小就必须满足所有层的输入输出大小，输入数据的大小依次为：208*208*16，104*104*32，52*52*64，26*26*128，13*13*256，13*13*512，13*13*1024，最大为 208*208*16；filter 的大小依次为：16*3*3，32*3*3，64*3*3，128*3*3，256*3*3，512*3*3，1024*3*3，最大为 1024*3*3；而单通道的输出显然最大的是 104*104。和 conv0_kernel 一样，将数据输入输出接口都声明为 STREAM 以节省硬件资源。

算法伪代码：

```
for (int in_channel = 0; in_channel <in_channels; in_channel++) {
    for (int row = 0; row < in_width + 1; row++) {
        for (int col = 0; col < in_width + 1; col++) {
            #pragma HLS PIPELINE II=1
            //当 row<1 且 col<2 时，缓冲 2 行数据到 linebuffer 中
            //当 row==1 且 col==2 时，开始进行卷积计算
            for (int i = 0; i < 9; i++) {
                #pragma HLS UNROLL
                tmp_cov += window[ch][i] * buffer_weights[ch][i];
            }
            //池化计算(conv6 和 conv7 之后无池化层)
            //缓存单个输入通道对应的输出
            //当执行最后一个输入通道时，输出最终的输出数据
        }
    }
}
```

conv1_7_kernel 在数据处理时以单个输入通道作为最小的处理单元，不是像 conv0_kernel 那样同时处理所有输入通道的数据。在每处理完成一个输入通道后，就用缓存器将该输入通道对应的输出数据缓存下来，在处理下一个输入通道的数据时，重复利用该缓存器，先累加再缓存，以此不断重复直到最后一个输入通道处理完毕，最后输出最终的输出数据。需要注意的是 conv6 和 conv7 之后没有池化层，所以不需要进行池化操作。

主要资源分配：

```
signed char linebuffer_0[210];
#pragma HLS RESOURCE variable=linebuffer_0 core=RAM_2P_BRAM
signed char linebuffer_1[210];
#pragma HLS RESOURCE variable=linebuffer_1 core=RAM_2P_BRAM；
signed char linebuffer_2[210];
#pragma HLS RESOURCE variable=linebuffer_2 core=RAM_2P_BRAM
signed char window[9];
#pragma HLS ARRAY_PARTITION variable=window complete dim=1
signed char weights_buf[9];
#pragma HLS ARRAY_PARTITION variable=weights_buf complete dim=1
signed int out_buf[208 * 208];
#pragma HLS RESOURCE variable=out_buf core=RAM_2P_BRAM
signed char maxpool_buf[104];
#pragma HLS RESOURCE variable=maxpool_buf core=RAM_2P_BRAM
```

3 个 linebuffer 的大小要满足 conv1-7 所有层的单通道输入数据的宽度，资源声明为 RAM_2P_BRAM；window、weights_buf、maxpool_buf 与 conv0_kernel 一样，唯一不同是不需要多个输入通道同时处理。signed int out_buf[208 * 208]是用来缓存单个输入通道对应的输出数据的缓冲器，其类型是 int，是为了尽量提高由乘加计算产生的数据的精度，因为乘加运算时两个 char 类型的数据进行计算，两个 char 数据的乘积不能再用 char 类型来保存，否则会造成严重的精度损失。out_buf 的硬件资源声明为 RAM_2P_BRAM 的原因是在缓存每个输入通道的输出数据时，需要将前一个输入通道对应的输出数据进行累加（第一个输入通道除外），然后再存储，这个累加和存储过程需要在一个时钟内完成。

优化 PIPELINE：

```
INFO: [HLS 200-10] ---------------------------------------------------------
INFO: [HLS 200-10] -- Scheduling module 'conv1_7_kernel'
INFO: [HLS 200-10] ---------------------------------------------------------
INFO: [SCHED 204-11] Starting scheduling ...
INFO: [SCHED 204-61] Pipelining loop 'Loop 1'.
INFO: [SCHED 204-61] Pipelining result: Target II: 1, Final II: 1, Depth: 92.
```

根据 scheduling 的信息提示，本论文的目标 II 是 1，在综合成功后的最终 II 也是 1，说明#pragma HLS PIPELINE II=1 这个优化目标实现了。

conv1~7 资源消耗报表如图 4.5 所示：

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	5	-	-
Expression	-	4	0	1187
FIFO	-	-	-	-
Instance	-	24	8124	11876
Memory	131	-	0	0
Multiplexer	-	-	-	1322
Register	-	-	2453	119
Total	131	33	10577	14504
Available	280	220	106400	53200
Utilization (%)	46	15	9	27

图 4.5 Conv1~7 资源报表

Fig 4.5 The resource report of Conv1~7

4.3.3 conv8_kernel 的 HLS 实现

函数接口以及端口配置如下：

```
void conv8_kernel(signed char inputs[13 * 13 * 1024],
                  signed char weights[1024 * 1*1],
                  float bias,
                  float outputs[13 * 13]) {
#pragma HLS STREAM variable=inputs
#pragma HLS STREAM variable=weights
#pragma HLS STREAM variable=outputs
```

该 IP 核的输出接口类型声明为 float 类型，因为数据经过该层之后，输出的数据直接用于检测，而检测时需要的是 float 型的数据，而且 13*13 个 float 数据消耗的资源比较少，可以直接使用。conv8_kernel 与之前两个 IP 核的区别在于 filter 从 3*3 变成了 1*1，所以对该 IP 的加速主要集中在循环处理上面，相对比较简单。

算法伪代码

```
for (int in_channel = 0; in_channel < 1024; in_channel++)
{
   for (int row = 0; row < 13; row++)
   {
            for (int col = 0; col < 13; col++)
            {
               #pragma HLS PIPELINE II=1
               //直接计算
               //缓存单个输入通道对应的输出
               //当执行最后一个输入通道时，输出最终的输出数据
            }
    }
}
```

主要资源分配：

```
float out_buf[13 * 13];
#pragma HLS RESOURCE variable=out_buf core=RAM_2P_BRAM
```

优化 PIPELINE：

```
INFO: [HLS 200-10] ---------------------------------------------------------------
INFO: [HLS 200-10] -- Scheduling module 'q_c8_kernel_1p'
INFO: [HLS 200-10] ---------------------------------------------------------------
INFO: [SCHED 204-11] Starting scheduling ...
INFO: [SCHED 204-61] Pipelining loop 'Loop 1'.
INFO: [SCHED 204-61] Pipelining result: Target II: 1, Final II: 1, Depth: 53.
```

根据 scheduling 的信息提示，本论文的目标 II 是 1，在综合成功后的最终 II 也是 1，说明#pragma HLS PIPELINE II=1 这个优化目标实现了。

Conv8 资源消耗报表如图 4.6 所示：

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	13	0	511
FIFO	-	-	-	-
Instance	-	8	2477	3712
Memory	2	-	0	0
Multiplexer	-	-	-	1872
Register	-	-	3355	11
Total	2	21	5832	6106
Available	280	220	106400	53200
Utilization (%)	~0	9	5	11

图 4.6 Conv8 的资源报表

Fig 4.6 The resource report of Conv8

4.4 结果分析

整个神经网络的计算量主要集中在卷积计算部分，对卷积进行并行加速时对计算资源要求较大，主要消耗的硬件资源是 DSP48；而前几层网络的特征分辨率比较高，需要使用 BRAM 进行数据的缓冲。为了使 DSP48 和 BRAM 的资源满足硬件限制，本论文对卷积层进行了分解，对资源和速度进行了平衡。从上述资源报告可以看出，3 个 IP 核总共使用了 143 个 BRAM，106 个 DSP48，对 BRAM 和 DSP48 的利用率约 50%，不仅完全满足了硬件资源的要求，并且对于硬件资源的利用还有可挖掘的余地，整个 Tiny-yolo 网络还有很大的加速空间。在对比 Tiny-yolo 网络运行时间方面，在 ZC702 的 ARM 中运行 100 次的平均运行时间约 8s，而以这 3 个 IP 核为基础在 FPGA 中加速过后，100 次的平均运行时间约 1.2s，速度方面有明显的提升，提高了约 6 到 7 倍。之所以没有达到理论的 9 倍及以上的速度，主要是因为 IP 核比较小，多次重复调用增加了数据传输成本。此外，由于整个网络在加速过程中采用 signed char 进行了量化，所以存在一定的误差，但是通过合理的量化方案，依然能实现目标检测，在工程上是可用的。用作者论文中采用的 voc2007 测试集进行测试，本论文所得到的 mAP 为 55.0， 而原始版本所得 mAP 为 57.1，整体效果基本与原版一致。如下是检测结果对比：

原始检测结果如图 4.7 所示。

person: 69%

sheep: 82%

cow: 52%

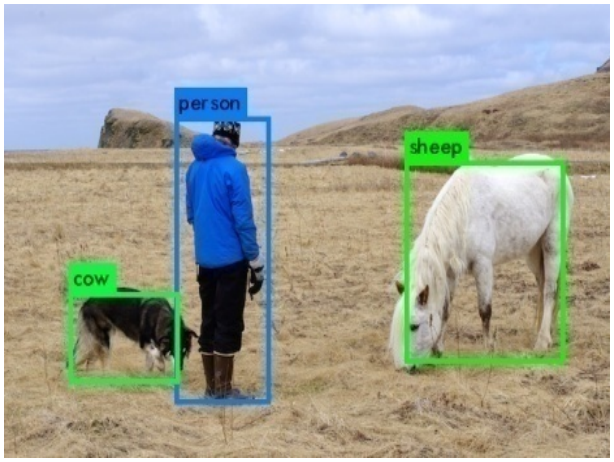


图 4.7 原始检测结果

Fig 4.7 The original test results

加速后检测结果如图 4.8 所示。

person: 55%

sheep: 69%

cow: 49%

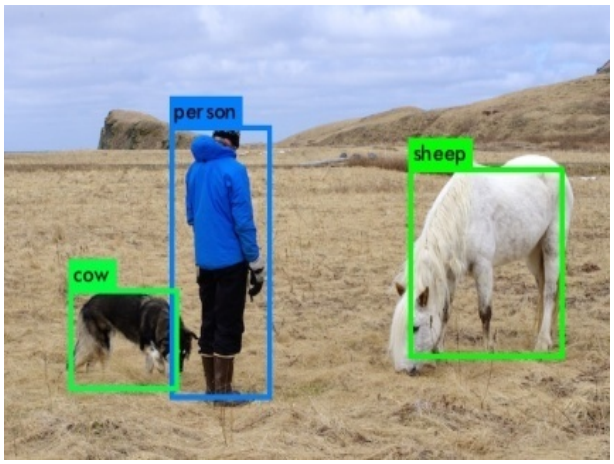


图 4.8 加速后检测结果

Fig 4.8 The accelerated test results

原始检测结果如图 4.9 所示。

car: 76%

bicycle: 25%

dog: 79%

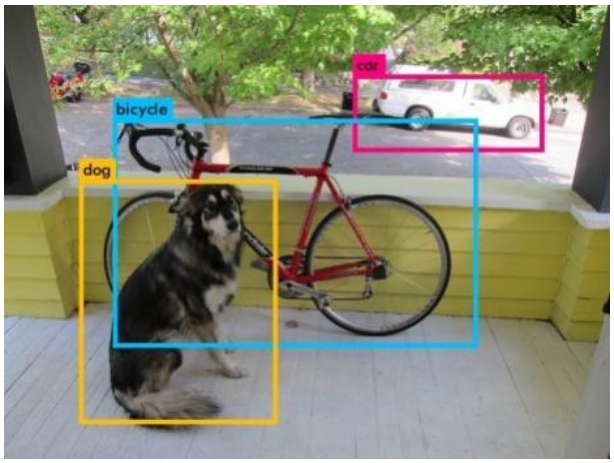


图 4.9 原始检测结果

Fig 4.9 The original test results

加速后检测结果如图 4.10 所示。

car: 57%

bicycle: 26%

dog: 48%

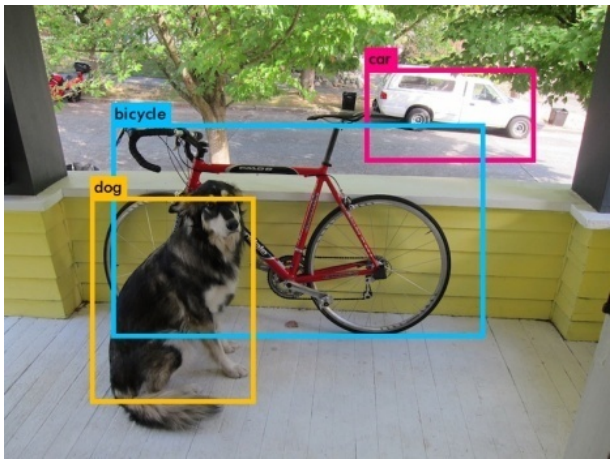


图 4.10 加速检测结果

Fig 4.10 The accelerated test results

从检测结果可以看出,目标框的位置有所偏移,各个检测目标的置信度有所降低,但是整体的检测结果基本和原始的一致。

4.5 本章小结

本章首先从开发原理、设计流程等方面介绍了 HLS 工具，然后介绍了 ZC702 测试平台以及其性能参数和资源概况。最后详细介绍了基于 HLS 的 Tiny-yolo 卷积神经网络加速算法的实现并对实验结果进行了分析。

5 总结与展望

5.1 总结

卷积神经网络（CNN）目前在计算机视觉领域应用广泛，在图像识别、目标检测以及图像分割等领域上都取得了巨大的成功。YOLO 卷积神经网络是近两年出现的一个新颖的端到端的目标检测模型，相比之前的 R-CNN 系列算法，YOLO 在保证检测精度的同时，大幅度提高了检测速度，为推动目标检测研究领域的发展做出了杰出的贡献。虽然 YOLO 网络相对于其他目标检测网络在检测速度上提升了很多，但是在对实时性、功耗以及成本要求较高的应用场景中依然受到一些限制。因此，基于低功耗和低成本的设备，研究 YOLO 卷积神经网络加速具有重要的意义，这也就是论文选题的出发点及研究的核心。在总结分析前人研究成果的基础上，提出了基于 HLS 的 Tiny-yolo 卷积神经网络的加速研究，主要工作概括为 3 点。

① 本论文对卷积神经网络的研究现状和关于卷积神经网络加速现状做了简单整理，详细介绍了 R-CNN、Fast R-CNN、Faster R-CNN、ION、HyperNet、SDP-CRC 以及 YOLO 的网络模型及各自的优缺点。通过对比分析，选择了端到端的 YOLO 卷积神经网络的简化版本 Tiny-yolo 作为研究，该网络能够一次性预测多个 bounding box 的位置和类别而且执行速度非常快。

② 针对目前很多应用对实时性、功耗以及成本的要求很高的情况，本论文对 Tiny-yolo 卷积神经网络进行了硬件加速。神经网络具有参数数据量大和卷积运算计算量大的特点，大量的数据和大量的运算在硬件平台上都直接影响着硬件资源的使用。为了减少浮点数据在存储时占用大量存储资源以及浮点运算占用大量计算资源，本论文对整个网络的权重和输入输出数据都进行了量化。为了加快硬件的设计，采用 HLS 工具，基于流水线的思想，设计了各个 IP 核。在设计 IP 核的时候将池化层和卷积层合并在一起，设计成统一的 IP 核。统一的卷积池化 IP 核比单独的卷积 IP 核和池化 IP 核有如下优势：将卷积层的数据直接用于池化过程而不需要经过额外的缓存和传输，也就是在卷积产生第一个输出的时候就开始进行池化操作，两者同时进行，无缝连接，不仅节省了硬件传输接口资源，而且提高了处理速度。对卷积过程的加速采用了卷积计算流水线的策略，使得 3*3 次乘法运算一次完成，实现 9 倍的卷积加速。

③ 通过最后的布线和整体设计，将设计好的 IP 核应用于 ZC702 芯片中，对比整个检测网络在 ARM 上和 ARM+FPGA 双架构上运行的时间，加速后的整个网络的目标检测时间提高了 6 到 7 倍。

5.2 展望

虽然加速的网络在目标检测速度上提升了 6 到 7 倍，实现了加速效果，但是跟 GPU 相比，仍然没有达到实时性的要求。不过本论文的方案仍有改进的空间，硬件平台的存储资源和计算资源的利用率只占约整体的 50%，只要能够充分的挖掘，在 ZC702 这种低功耗、低成本的设备上，依然能够实现实时检测。随着卷积神经网络（CNN）在计算机视觉领域的广泛应用，目标检测系统对实时性和便携性的需求与日俱增。结合本论文的研究工作，下一步还要在此基础上，就如何进一步加速神经网络的数据传输和运算做相关的研究。

致 谢

在攻读硕士学位期间，首先衷心感谢我的导师黄智勇副教授，黄老师三年的谆谆教导无时无刻不在影响着我的学习和生活。科研上，黄老师的勤奋和努力以及对科研的热情给了我很大的激励，让我在面对科研的失败时能够不懈怠不气馁。在生活上，黄老师深入体察学生的境况，对学生的生活也是细心了解，无私帮助。黄老师确实给我们学生做了一个很好的榜样，值得我们每一个学生去学习。

感谢我的师弟李杰，感谢朝气蓬勃、和睦相处、互帮互助的实验室大家庭！大家一起相互团结，互相学习，营造了实验室温馨和谐的学术氛围，使我在实验室的学术学习和生活充满乐趣。当然，还有很多值得我感恩的老师和同学，这里不一一列举，非常感谢研究生生涯与你们相遇，是你们使我的研究生生活如此丰富多彩！

感谢我的父母和爱人，不断地鼓励我，给我所需要的各种帮助和支持。你们的爱和鼓励是我不断前行的动力。

感谢充满活力、催人奋进、快速发展的通信工程学院！感谢求实创新、自强不息、厚德载物的重庆大学！

最后，衷心的感谢各位专家、教授、老师，感谢你们百忙之中进行论文评阅和答辩评审！

张丽丽

二O一七年四月 于重庆

参考文献

[1] Garcia C, Delakis M. Convolutional face finder: A nerual architecture for fast and robust face detection[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004, 26(11): 1408-1423

[2] Delakis M, Garcia C. Text detection with convolutional neural networks[C]. International Conference on Computer Vision Theory and Applications, 2008, 290-294

[3] Farabet C, Poulet C, Han J, etal. Cnp: An FPGA-based processor for convolutional networks[C]. Field Programmable Logic and Applications, 2009. FPL 2009 International Conference on IEEE, 2009, 32-37

[4] Lecun Y, Bottou L, Bengio Y, etal. Gradient-based learning applied to document recognition[J]. Proceding of the IEEE, 1998, 86(11): 2278-2324

[5] Ji S, Xu W, Yang M, etal. 3D convolutional neural networks for human action recognition[J]. IEEE Trans Pattern Anal Mach Intell, 2013, 35(1): 221-231

[6] 第 35 次中国互联网络发展状况统计报告[Oline]. Available: http://www.cnnic.cn/research/

[7] Vailaya A, Jain A, Zhang H. On image classification: City images vs Landscape[J]. Pattern Recognition. 1998, 31(12): 1921-1935

[8] Haralick R M, Shanmugam K, Dinstein I. Texural features for image classication[J]. IEEE Transactios on Man and Cybermetics, 1973, 3(6): 610-621

[9] Garcia C, Delakis M. Convotional face finder: A neural architecture for fast and robust face detecion[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004, 26(11): 1408-1423

[10] Chen Y, Han C, Wang C, etal. The application of a convolution neural network on face and license plate detection[C]. Proceeding of 18th International Conference on Pattern Recognition, 2006, 552-555

[11] Hubel D, Wiesel T. Receptive fields: Binocular interaction and functional architecture in the cat's visual cortex[J]. The Journal of Physiology, 1962, 160(1): 106-154

[12] Hopfield J J, Tank D W. Computing with neural circuits: A model[J]. Science, 1986, 233: 625-633

[13] Lecun Y, Jackel L, Bottou L, etal. Learning algorithms for classification: A comparism on handwritten digit recognistion[J]. Neural Networks, 1995, 21: 261-276

[14] Hinton G, Osindero S, Teh Y W. A fast learning algorithm for deep belief nets[J]. Neural Computation, 2006, 18(7): 1527-1554

[15] Deng J, Dong W, Socher R, etal. ImageNet: A large-scale hierarchical image database[C]. Computer Vision and Pattern Recognition, 2009, 248-255

[16] Hopfield J J. Neurons with graded response have collective computational properties like those of two-state neurons[J]. Proceeding of the National Academy of Science, 1984, 81: 3088-3092

[17] Szegedy C, Liu W, Jia Y, etal. Going deeper with convolutions[C]. IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2014, 1-9.

[18] He K, Zhang X, Ren S, etal. Deep residual learning for image recognition[J]. Computer Vision Foundation，2015，770-778

[19] Silver D, Huang A, Maddison C J, etal. Mastering the game of go with deep neural networks and tree search[J]. Nature, 2016, 529(7587): 484-489

[20] Sankaradas M, Jakkula V, Cadambi S, etal. A massively parallel coprocessor for convolutional neural networks[C]. 2013 IEEE 24th International Conference on Application-Specific System, Architectures and Processors IEEE, 2009, 53-60

[21] Frome A, Cheung G, Abdulkader A, etal. Large-scale privacy protection in google street view[C]. IEEE International Conference on Computer Vision, 2009, 2373-2380

[22] Mirowski P W, Lecun Y, Madhavan D, etal. Comparing SVM and convolutional networks for epileptic seizure prediction from intracranial EEG[C]. Machine Learning for Signal Processing, MLSP 2008 IEEE Workshop on IEEE, 2008, 244-249

[23] Nomura O, Morie T. Projection-field-type vlsi convolutional neural networks using merged/mixed analog-digital approch[C]. Neural Information Processing, 2008, 1081-1090

[24] Bengio Y, Delalleau O. Algorithmic Learning Theory[M]. Springer Berlin Heidelberg, 2011, 18-36

[25] Girshick R, Donahue J, etal. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR2014

[26] Girshick R. Fast R-CNN: Objection detection with caffe. ICCV2015

[27] Ren S, He K, Girshick R, etal. Faster R-CNN: Towards real-time object detection with region proposal networks[J]. Advances in Neural Information Processing Systems, 2015, 1-14

[28] Bell S, Zitnick C L, Bala K, etal. Inside-outside Net: Detecting objects in context with skip pooling and recurrent neural networks. arXiv preprint arXiv: 1512.04143(2015)

[29] Kong T, Yao A, Chen Y, etal. HyperNet: Towards accurate region proposal generation and joint object detection. arXiv preprint arXiv: 1604.00600(2016)

[30] Yang F, Wongun C, Lin Y. Exploit all the layers: Fast and accurate CNN object detector with scale dependent pooling and cascaded rejection classifiers. CVPR2016

[31] Redmon J, Divvala S, Girshic R, etal. You Only Look Once: Unified real-time object detection. arXiv preprint arXiv: 1506.02640(2015)

[32] Farabet C, Lecun Y, Kavukcuoglu K, etal. Large-scale FPGA-based convolutional networks[J]. Scaling up Machine Learning Parallel and Distributed Approaches, 2011, 399-419

[33] Mathieu M, Henaff M, Lecun Y. Fast training of convolutional networks through FFTs. arXiv preprint arXiv: 1312.5851(2013)

[34] Denil M, Shakibi B, Dinh L, etal. Predicting parameter in deep learning[C]. Proceesing of the twenty-eighth Annual Conference on Neural Information Processing System, 2014, 2148-2156

[35] Denton E, Zaremba W, Bruna J, etal. Exploiting linear structure within convolutional networks for efficient evaluation[C]. Proceesing of the 28th Annual Conference on Neural Information Processing System, 2014, 1269-1277

[36] Cheng T, Du Z, Sun N, etal. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning[C]. Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, 2014, 269-284

[37] Chen Y, Luo T, Liu S, etal. DaDianNao: A machine-learning supercomputer[C]. Microarchitecture of the 47th Annual IEEE/ACM International Symposium on IEEE, 2014,609-622

[38] Liu D, Chen T, Liu S, etal. PuDiannao: A polyvalent pachine learning accelerator[J]. Acm Sigplan Notices, 2015, 50(4): 369-381

[39] Li T, Dou Y, Jiang J, etal. Optimized deep belief networks on CUDA GPUs[C]. Neural Networks of 2015 International Joint Conference on IEEE, 2015, 1-8

[40] Ly D L, Chow P. A high-performance FPGA architecture for restricted boltzmann machines[C]. Proceeding of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2009, 73-82

[41] Ly D L, Chow P. A multi-FPGA architecture for stochastic restricted boltzman machine[C]. Field Programmable Logic and Applications International Conference on IEEE, 2009, 168-173

[42] Ly D L, Chow P. A high-performance, reconfigurable hardware architecture for restricted Boltzman machines[C]. IEEE Transactions on Neural Networks, 2010, 1780-1792

[43] Kim S K, Mcmahon P L, Olukotun K. A large-scale architecture for restricted Boltzmann machines[C]. Field-Programmable Custon Computing Machines, the 18th IEEE Annual International Symposium on IEEE, 2010,201-208

[44] Kim S K, Mcafee L C, Mcmahon P L, etal. A highly scalable restricted boltzmann machine FPGA implementation[C]. Field Programmable Logic and Applications, International Conference on IEEE, 2009,367-372

[45] Qiu J, Wang J, Yao S, etal. Going deeper with embeded FPGA platform for convolutional neural network[C]. ACM International Symposium on FPGA, 2016, 26-35

附 录

A 作者在攻读学位期间论文成果：

[1] 张丽丽，黄智勇等．Tiny-yolo 卷积神经网络加速研究．中南大学学报．（评审中）．