分类号：TN492　　学校代码：10363
密　级：公开　　学　　号：2220320128

安徽工程大学
Anhui Polytechnic University

硕士学位论文

题目　基于ZYNQ的多目标检测系统设计与研究

论文作者　刘龙生
校内导师　张肖强
校外导师　朱标

专业学位类别　电子信息
研究方向（领域）　系统集成技术与应用

论文提交日期：2025 年 6 月 6 日

分类号：TN492　　单位代码：10363
密　级：公开　　学　　号：2220320128

题　　目 基于 ZYNQ 的多目标检测系统设计与研究

题　　目 DESIGN AND REAEARCH OF MULTI OBJECT DETECTION SYSTEM BASED ON ZYNQ

学 生 姓 名：刘龙生
指 导 教 师：张肖强（副教授）
校 外 教 师：朱标
专　　业：电子信息（085400）
研 究 方 向：系统集成技术与应用

论文答辩日期：2025 年 05 月 10 日

二、安徽工程大学学位论文原创性声明

本人郑重声明：我恪守学术道德，崇尚严谨学风。所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已明确注明和引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品及成果的内容。论文为本人亲自撰写，我对所写的内容负责，并完全意识到本声明的法律后果由本人承担。

学位论文作者签名：刘龙生

日期：2025 年 5 月 10 日

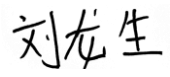
三、安徽工程大学学位论文版权使用授权书

学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅或借阅。本人授权安徽工程大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密 □，在 ____ 年解密后适用本版权书。

本学位论文属于

不保密 ☑。

学位论文作者签名：  指导教师签名： 

日期： 2025 年 5 月 10 日　　日期：2025 年 5 月 10 日

基于 ZYNQ 的多目标检测系统设计与研究

摘 要

随着人工智能时代的到来，基于深度学习的目标检测算法广泛应用到日常生活中，如自动驾驶、智能安防、故障检测、医疗成像等领域。卷积神经网络等深度模型通过对大量数据样本进行特征学习，进而加强了其对抽象概念的理解，能够对不同物体进行准确的实时检测。随着目标检测算法的检测精度的不断提升，其庞大的计算量与参数量对设备的计算能力要求较高。虽然 GPU 凭借着其强大的并行计算能力广泛应用到目标检测算法的实现中，但其高昂的成本与较高的功耗难以应用到对功耗敏感的场所。因此，本文将在 FPGA+ARM 架构的 ZYNQ 平台上搭建一个低功耗、高处理速度、高能效比的目标检测系统。本文的主要工作内容与创新点如下：

(1)针对 YOLOv3-tiny 算法在杂物较多存在遮挡的背景环境下进行目标检测推理时检测精度不高，存在漏检的问题，对算法进行了改进。首先，替换了损失函数，将更能体现预测框与真实框的重合度与距离情况的 GIOU 损失作为边界框坐标的回归损失，提高了预测的准确性。然后，修改了主干网络，将网络中的前三个池化层替换为卷积层，提高了对目标的表达能力；并在主干网络末尾添加了改进后的空间金字塔模块，更好的提取特征信息。实验结果表明，改进后 mAP 提高了 5.08%，精确度提高了 7%。

(2)在 PL 端设计了一款卷积神经网络加速器，在 ARM 的控制下能够加速推理运算。通过多通道并行设计实现在卷积运算的输入、输出通道两个维度的并行处理。设计了深度可配置的行缓存结构，提高了加速器的通用性。为了降低加速器的资源消耗，进行了乘加法优化设计。同时在特征图分块缓存方面进行了优化，优化后的设计省去了复杂寻址操作，更易在 FPGA 上实现。并通过仿真测试，验证其功能的正确性。

(3)在 PL 端实现了目标检测系统的其他模块，如图像采集模块、图像预处理模块、目标标记模块与图像显示模块等，这些模块与加速器在 FPGA 上搭建了目

标检测硬件系统，并与 PS 端协同工作，最终在 ZYNQ-XCZU3EG 平台上实现了改进后的 YOLOv3-tiny 目标检测算法。经过系统验证，能够实现对苹果、香蕉、胡萝卜的实时检测，检测速度达到了 21.7FPS，检测精度达到了 75%，吞吐率达到 81.23GOPS，能效比达到 18.21GOPS/W，计算密度达到 0.27GOPS/DSP。

关键词：目标检测，YOLOv3-tiny，深度学习，ZYNQ，FPGA，卷积神经网络加速器

DESIGN AND REAEARCH OF MULTI OBJECT DETECTION SYSTEM BASED ON ZYNQ

ABSTRACT

With the advent of the artificial intelligence era, deep learning-based object detection algorithms have been widely applied in daily life, including autonomous driving, smart security, fault detection, and medical imaging. Deep models such as convolutional neural networks enhance their understanding of abstract concepts by learning features from large datasets, enabling accurate real-time detection of various objects. As the detection accuracy of object detection algorithms continues to improve, the increasing computational complexity and large number of parameters impose high demands on computing devices. Although GPUs are widely used for object detection due to their powerful parallel computing capabilities, their high cost and energy consumption make them unsuitable for power-sensitive applications. Therefore, this paper proposes a low-power, high-speed, and high energy efficiency ratio object detection system implemented on the ZYNQ platform with an FPGA+ARM architecture. The main contributions and innovations of this paper are as follows:

(1) To address the issue of low detection accuracy and missed detections when using the YOLOv3-tiny algorithm for object detection in cluttered and occluded environments, the algorithm was improved. First, the loss function was replaced with GIoU loss, which better reflects the overlap and distance between the predicted and ground truth boxes, improving prediction accuracy. Then, modifications were made to the backbone network by replacing the first three pooling layers with convolutional layers to enhance feature representation and adding an improved spatial pyramid module at the end of the backbone to extract features more effectively. Experimental results show that after these improvements, the mean Average Precision (mAP)

increased by 5.08%, and accuracy improved by 7%.

(2) A CNN accelerator was designed on the PL side to speed up inference under ARM control. Multi-channel parallel design was implemented to achieve parallel processing in both input and output channel dimensions of convolution operations. A deeply configurable row buffer structure was designed to enhance the accelerator's versatility. To reduce resource consumption, an optimized multiply-accumulate design was applied. Additionally, optimizations were made in feature map block caching, eliminating complex addressing operations and making FPGA implementation more efficient. Functionality was verified through simulation testing.

(3) The PL side implemented other object detection modules, including image acquisition, preprocessing, object marking, and display. These, along with the accelerator, formed the FPGA-based detection system, working collaboratively with the PS side. Ultimately, the improved YOLOv3-tiny object detection algorithm was successfully implemented on the ZYNQ-XCZU3EG platform. System verification demonstrated real-time detection of apples, bananas, and carrots, achieving a detection speed of 21.7 FPS, an accuracy of 75%, a throughput of 81.23 GOPS, an energy efficiency of 18.21 GOPS/W, and a computational density of 0.27 GOPS/DSP.

KEY WORDS: object detection, YOLOv3-tiny, deep learning, ZYNQ, FPGA, convolutional neural network accelerator

目 录

第 1 章 绪论

1.1 研究背景及意义

目标检测[1]作为计算机视觉[2]中的重要研究领域，旨在识别图像或视频中的物体对其进行种类判定与位置定位[3]。现如今，基于深度学习[4]的目标检测技术在日常生活中已经有了广泛的应用。如在自动驾驶领域，利用目标检测技术对于道路上行人、车辆、交通标志等目标进行实时检测[5]，进而为自动驾驶系统[6]提供了感知能力，推动了无人驾驶技术的快速发展；在安防监控领域，通过将目标检测技术应用在人脸识别[7]、行为识别、异常行为检测[8]等方面，极大提高了监控系统的智能化水平。此外，目标检测技术在无人机、医疗影像[9]、智能交通、工业自动化等领域也取得了显著的应用成果，极大推动了社会的进步，在当今社会表现出不可替代的价值。

在早期的目标检测主要靠手工进行特征提取和经典的机器学习算法，如支持向量机[10](Support Vector Machine，SVM)、k-近邻算法[11](K-Nearest Neighbors，KNN)等。这些方法虽然能实现图像分类和物体定位，但是需要人工设计大量特征，且通常难以处理较为复杂的场景，存在准确性和鲁棒性的不足。随着深度学习[12]，尤其是卷积神经网络[13](Convolution Neutral Network，CNN)的出现，目标检测领域经历了巨大改革[14]。目前，基于深度学习的目标检测方法通过自主学习图像的不同层次的图像特征，使得目标检测不再依赖于手工特征，而是能够从所提供的大量的标注数据集中直接学习到高效的特征表达方式。随着近年来基于深度学习的目标检测算法的不断提出，极大推动了目标检测技术的发展，在检测精度与速度方面取得了很大进步。

当前实现基于 CNN 的目标检测算法的方式，主要通过嵌入式系统与主流推理平台 CPU[15](Central Processing Unit，中央处理器)和 GPU[16](Graphic Processing Unit，图像处理单元)来实现。其中，CPU 是一种通用处理器，适用于各种类型的计算任务，能够实现基于 CNN 的目标检测算法，但 CPU 的并行处理能力较差，而在执行目标检测任务时往往需要大量的并行计算，因此 CPU 处理速度较慢，

无法高效完成目标检测任务。相较于 CPU，GPU 拥有大量可并行运算的处理单元，在执行卷积神经网络推理时有着较高的执行效率，在目标检测中的表现较为突出，但高性能的 GPU 成本较高，且在工作时功耗较大，并不适用于对功耗敏感的边缘计算场景当中。ASIC[17](Application Specific Circuit Integrated，专用集成电路)是针对特定用途定制的集成电路，能够提供比 CPU 和 GPU 更高的计算效率与更低的功耗，但其成本高，设计难度大，且设计完成后无法更改，缺乏灵活性。FPGA[18](Field Programmable Gate Array，现场可编程门阵列)具有高度的可编程性，可根据不同的计算任务进行编程与配置，在推理运算时能同时执行大量的并行运算，且运行时功耗较低，因此适合在嵌入式目标检测系统中使用。ARM(Advanced RISC Machine，高级精简指令集计算机)具有能效高和灵活性强等特点。其中，ZYNQ 是一种融合了 PS(Processing System，处理器系统)和 PL(Programmable Logic，可编程逻辑)的芯片，其同时拥有了 ARM 处理器与 FPGA，既保留了 ARM 处理器高效运行操作系统的能力，又保留了 FPGA 灵活实现硬件加速的能力。本文在 ZYNQ 上采用 FPGA+ARM 的架构[19]兼具了 FPGA 的可编程性、高并行计算能力与 ARM 的灵活性、高效序列处理能力。

因此，采用 FPGA+ARM 的架构搭建目标检测系统具有重要的研究价值和工程意义。

1.2 国内外研究现状

1.2.1 目标检测算法研究现状

自从 20 世纪 90 年代以来，人们对于目标检测算法的研究，经历了从传统方法到深度学习方法的快速发展[20]。随着深度学习的不断发展，卷积神经网络在特征提取方面的优势，使得目标检测算法逐渐从传统的特征提取方法过渡到基于深度学习的模型[21]。目前，使用深度学习进行开发的目标检测网络架构主要分为两种：以 YOLO(You Only Look Once，你只看一次)系列为代表的一阶段目标检测算法[22]，其具有检测实时性较好的特点和以 R-CNN(Region-based Convolution Neural Networks，基于区域的卷积神经网络)系列为代表的二阶段目标检测算法[23]，其具有检测精度较好的特点。

2014 年，Girshick 等人[24]提出了 R-CNN 网络模型，将深度卷积神经网络应用于目标检测场景中，引入了区域提议生成方法，并将深度学习模型与传统的候选框生成算法结合，进而极大提高了检测精度。2015 年，He 等人[25]提出 SPP-Net(Spatial Pyramid Pooling Networks，空间金字塔池化网络)，解决了 CNN 对输入图像尺寸要求严格的限制，使得网络能够在不同尺寸的输入图像上进行处理，从而提升了网络的灵活性和效率。2015 年，Girshick 等人[26]提出了 Fast R-CNN 算法，在输入图像上只进行一次卷积操作，通过共享卷积特征减少了计算量，并提出一种单阶段的端到端训练方法，提升检测精度的同时优化了训练过程。同年，Ren 等人提出 Faster R-CNN 算法[27]，在 Fast R-CNN 的基础上进一步提升了检测速度和精度，解决了候选区域生成效率低的问题，并提出了 RPN(Region Proposal Network，区域提议网络)，从而实现了端到端的训练和推理。R-CNN 系列二阶段目标检测算法虽然有着较高的检测精度但实时性不如一阶段 YOLO 系列，不适用于对于实时性要求较高的场景。

2016 年，Redmon 等人提出了 YOLO 系列第一代 YOLOv1 算法[28]，将目标检测问题转化为回归问题，它通过一个单一的卷积神经网络实现同时预测物体的位置和类别，不再依赖于候选区域生成的步骤。为解决 YOLOv1 算法检测小目标的精度不高的问题，2017 年，Redmon 等人提出了 YOLOv2 算法[29]，使用了改进后的 Darknet-19 网络结构，引入多尺度训练与锚框技术，使用批量归一化提高了模型的稳定性，并显著提升了目标检测的精度和速度。2018 年，Redmon 团队再次提出了 YOLOv3 [30]与 YOLOv3-tiny 算法，其中 YOLOv3 使用了 Darknet-53 作为其网络架构的主干，引入了多尺度预测，优化了检测框架，提高了检测精度。YOLOv3-tiny 为 YOLOv3 的轻量化版本，通过减少网络深度和参数量，优化了计算效率，在牺牲一部分精度的情况下大幅提高了检测速度，更适用于资源受限的嵌入式系统与实时性要求苛刻的场景。2020 年，Bochkovskiy 等人提出了 YOLOv4 算法[31]，引入了 CSPDarknet53 网络与先进的数据增强技术，优化了损失函数，结合了多个现代计算机视觉中的优秀技术，进一步提高了检测精度与速度。

本设计在嵌入式平台上实现目标检测算法[32]，对于实时性要求较高，综合考

虑后，最终选择一阶段 YOLO 系列算法，同时为弥补 YOLO 系列算法相对于 R-CNN 系列算法检测精度略低的问题，对 YOLO 算法进行改进优化，以满足高精度与高处理速度的需求。

1.2.2 卷积神经网络加速器研究现状

近年来，基于 FPGA 实现卷积神经网络加速器是深度学习硬件加速领域的热点[33]，FPGA 能够根据特定的需求进行定制，其强大的并行计算能力能够同时完成多个卷积任务。然而加速器在执行推理运算时，往往对硬件资源的需求较高，同时为了改善在 FPGA 上设计加速器的性能瓶颈，众多学者从算法与硬件结构方面进行了优化设计[34]。

在算法优化方面，主要通过对网络模型进行优化，如定点量化、模型剪枝等操作。Gholami 等人讨论了深度神经网络数值量化问题及当前方法的优缺点[35]，过去十年，神经网络通过过度参数化获得了较大精度提升，但要实现高效实时网络并保持精度，需要重新考虑模型设计和训练。模型蒸馏通过训练大型模型，再以该模型进行训练更紧凑的模型，进而提高了效率。Jacob 等人针对智能移动设备和深度学习视觉识别模型的高计算成本[36]，提出了一种量化方案，将激活值和权重量化为 8 位整数，减少了约 4 倍的内存占用。该方法在典型的 ARM 上性能优于浮点运算，并在 MobileNets 等模型上实现了延迟与精度的良好平衡。Han 等人提出了“三阶段深度压缩”方法[37]，包括剪枝、训练量化和霍夫曼编码，能够在不影响精度的情况下，将神经网络存储需求减少 35 到 49 倍。通过压缩，AlexNet 和 VGG-16 网络的存储需求分别减少 35 倍和 49 倍，并显著提高了推理速度和能效，适用于移动设备。Ding 等人提出了 CirCNN[38]，一种基于块循环矩阵表示权重和处理神经网络的方法，旨在克服权重剪枝的缺点，利用快速傅里叶变换加速计算，减少了推理和训练的计算复杂度和存储复杂度，且能在 FPGA 平台上实现，能效比提升了 6-102 倍。

在硬件结构方面的优化，主要通过提高能效比、通用性与进行资源优化来实现。Shen 等人提出了一种灵活的数据缓冲方案的 CNN 加速器架构 Escher[39]，通过平衡输入和权重传输带宽，进而显著减少了带宽需求。与 Virtex-7 690T FPGA 上的现有设计相比，Escher 在 AlexNet 上将加速器峰值带宽需求降低了 2.4 倍，

在 GoogleNet 上，卷积层带宽需求降低了 10.5 倍。Guo 等人提出了一个可编程灵活的 CNN 加速器架构 Angel-Eye [40]，结合数据量化策略和编译工具，优化了 CNN 在嵌入式 FPGA 上的部署。在 VGG 网络、行人检测和人脸对齐应用的测试中，Angel-Eye 在 NVIDIA TK1 和 TX1 平台上表现出相似的性能，且能效提升最多 16 倍。Ma 等人提出了一种优化卷积操作的数据流和硬件加速方案[41]，通过量化分析 CNN 加速器的设计目标和多个设计变量，优化硬件设计以减少内存访问和数据移动，同时最大化资源利用率，实现了 VGG-16 模型，并达到了 645.25GOPS 的吞吐量和 47.97 毫秒的延迟。

综上，本设计对模型参数进行了 8 位定点量化以降低模型推理的计算量，并在硬件设计时进行了优化，以便于在 FPGA 上实现，并提高了通用性，优化了硬件资源使用情况。

1.3 研究内容

本文在实验室已有的目标检测平台[42]的基础上，对 YOLOv3-tiny 算法进行了改进[43]，并在设计卷积神经网络加速器时进行一些优化策略，且在 PS 端实现聚类与非极大值抑制，最终在 ZYNQ 平台上搭建一个目标检测系统，实现对苹果、香蕉、胡萝卜的实时检测。本文的主要研究内容如图 1-1 所示。

(1)针对 YOLOv3-tiny 算法在杂物较多存在遮挡的背景环境下进行目标检测时检测精度不高，存在漏检的问题，对算法进行改进优化[44]。将 GIOU(Generalized Intersection Over Union，广义交并比)损失替换原始的 IOU(Intersection Over Union，交并比)作为边界框坐标的回归损失，从而更好的反映预测框与真实框的重叠与距离情况，提高了检测精度[45]。此外，将主干网络的前三层池化层替换为卷积层，有效减少了特征丢失，并在主干网络的末尾添加了空间金字塔[46]，从而更好的融合特征信息[47]。最后将模型参数量化为 8 位定点数据[48]，有效降低了推理运算的计算量。

(2)针对卷积加速器在推理运算时对硬件资源需求较高，难以在资源受限的开发板上实现的问题，对乘加法运算操作进行了优化设计[49]。并针对大尺寸特征图进行分块缓存时存在特征图块的地址不连续，寻址复杂的问题，进行了分块优

化设计[50]。并设计了深度可配置的行缓存结构[51]，提高了加速器的通用性。

(3)在 PL 端实现硬件端各个模块的设计，搭建了整个目标检测硬件系统。在 PS 端实现了系统初始化，加速器的调度与控制以及聚类与非极大值抑制。在完成 PL 端与 PS 端的设计后，成功在 ZYNQ 平台上搭建了一个实时目标检测系统，并通过系统验证与性能分析，证明了本系统对于苹果、香蕉、胡萝卜检测的可行性。

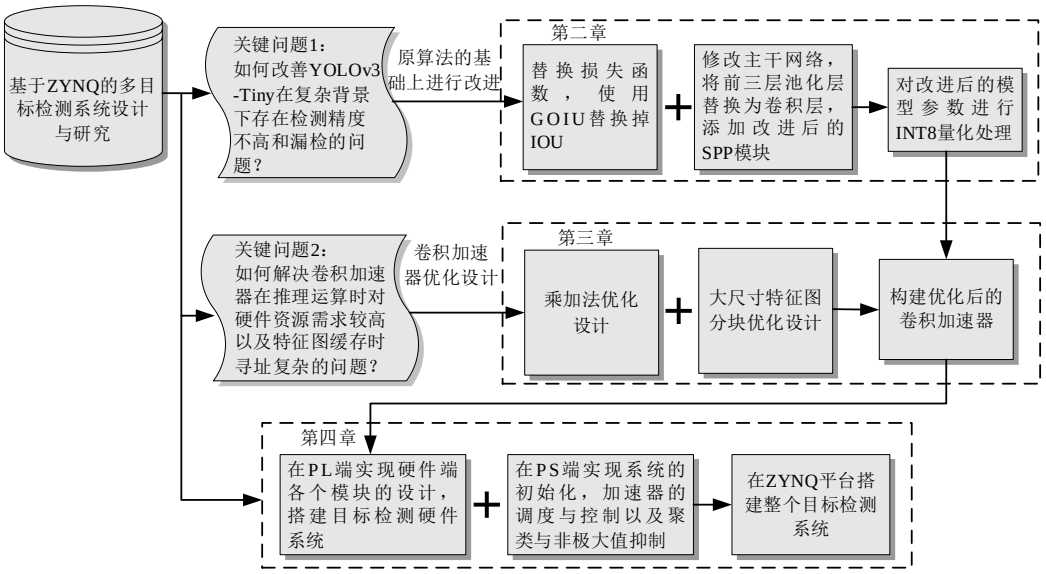


图 1-1 研究内容框架

1.4 论文结构安排

基于上述对本课题的背景导入与内容介绍，文章主要分为五个章节，各章节内容如下：

第一章是绪论。首先对目标检测技术的背景与意义进行介绍，说明了目标检测技术的重要性，其次介绍了基于 CNN 的目标检测算法的各种实现方法，并对各种实现方法进行对比。然后介绍了目标检测算法与卷积神经网络加速器的研究现状。最后阐述了本文的研究内容与结构安排。

第二章是目标检测算法介绍与改进。首先介绍了两种基于深度学习的目标检测算法，并说明选择 YOLOv3-tiny 为基础算法。其次对于 YOLOv3-tiny 算法的计算原理进行了介绍，并对 YOLOv3-tiny 算法进行改进，提高了检测效果。最后将 YOLOv3-tiny 模型进行了 8 位定点量化操作。

第三章是卷积神经网络加速器设计。首先对卷积加速器进行概念介绍，再对本设计的卷积加速器的总体架构进行阐述，接着对加速器中的计算模块、缓存模块、控制模块分别进行详细介绍，这些模块组合起来构成了本设计的卷积加速器。然后对卷积加速器进行乘加法与大尺寸特征图分块优化设计。最后对加速器进行仿真测试，验证其功能的正确性。

第四章是目标检测系统的搭建与实现。首先对目标检测系统的总体架构进行介绍，主要由 PL 与 PS 两部分组成，再对器件选型进行介绍，接着对 PL 端与 PS 端的设计进行详细阐述。最后对系统进行验证与性能分析。

第五章是总结与展望。总结本文的主要工作与创新之处，分析现阶段工作的不足，说明了能够进一步改进的地方，为后续的工作方向进行展望。

第 2 章 目标检测算法介绍与改进

本章介绍了基于深度学习的两种目标检测算法，并选择 YOLOv3-tiny 作为基础算法，然后详细介绍了 YOLOv3-tiny 算法的原理与网络结构，并针对基于 ZYNQ 平台的实现的 YOLOv3-tiny 算法在杂物较多存在遮挡的背景环境下进行目标检测推理时检测精度不高，存在漏检的问题，对于一阶段目标检测 YOLOv3-tiny 算法进行了损失函数以及网络结构的改进，进而提高了检测精度。

2.1 基于深度学习的目标检测算法

目标检测的主要任务就是对于想要检测物体进行分类与定位，随着深度学习的不断发展，目标检测算法逐渐转为了基于深度学习来进行实现了[52]。目前主流的基于深度学习的目标检测算法为一阶段目标检测算法与双阶段目标检测算法。本节主要针对这两种目标检测算法进行介绍。

2.1.1 一阶段目标检测算法

一阶段目标检测如图 2-1 所示，通常是采用了是端到端的形式。通过在网络中提取特征，直接从待检测图像中预测相关目标的所在位置与所属类别，而无需像二阶段目标检测算那样先生成可能包含待检测物体的预选框 RP(Region Proposal，候选区域)，将位置定位与图像分类合为一个阶段得到最终检测结果，极大简化了网络结构，加快了检测速度，非常适用于对于检测速度要求较高，资源受限的设备中。

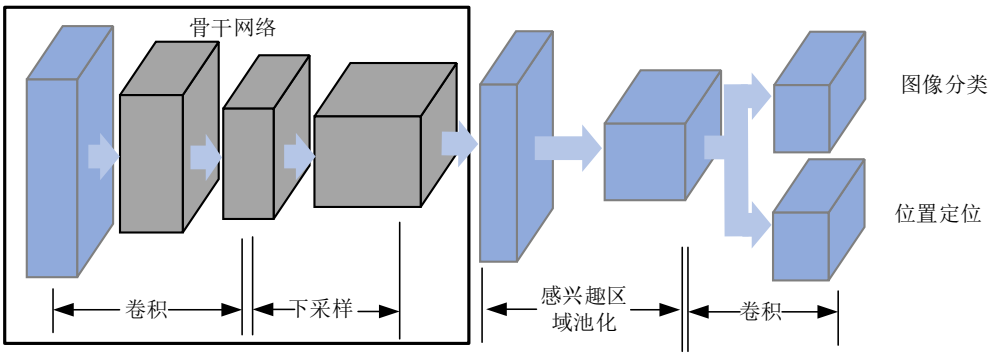


图 2-1 一阶段目标检测算法图

一阶段目标检测算法代表算法有：YOLO、SSD、RetinaNet 等，其中 YOLO 算法是非常经典的一阶段目标检测算法，通过将待检测图像划分为 N×N 的网格，每个网格负责预测一组边界框(bounding boxes)及其置信度、类别概率。每个网格根据其预测的边界框来决定是否有物体及其类别。YOLO 的核心思想是将目标检测的问题转化为一个回归问题，通过一个单一的神经网络直接输出物体类别和边界框，检测速度较快[53]，且网络复杂度较低，更加适合移植到资源受限的 FPGA 硬件系统当中。将 YOLO 系列部署到 ZYNQ 平台时，应充分考虑 ZYNQ 的资源与性能特点。虽然较新的 YOLO 模型(如 YOLOv7、YOLOv8)往往具有较好的检测效果，但新版本模型普遍采用了更加复杂和深层的网络结构，进而导致整体参数量和计算量成倍增长。此外，新版 YOLO 通常引入了更复杂的激活函数(如 SiLU)和更特殊的网络操作(如 Focus 模块)，这类运算在 FPGA 上实现难度较大且资源消耗极高。ZYNQ 芯片在逻辑单元数量、DSP 运算单元数量、片上 BRAM 容量及 DDR 带宽方面存在天然的物理限制，难以支撑这类大规模、高复杂度模型的高效运行。而 YOLOv3-tiny 为 YOLOv3 的轻量化版本，简化了网络结构，参数总量与计算复杂度显著降低，且激活函数采用 Leaky ReLU，避免了复杂激活函数带来的额外计算开销，因此更加适合移植到资源受限的 FPGA 硬件系统当中，因此本文选用了 YOLOv3-tiny 算法为本系统的基础算法。

2.1.2 两阶段目标检测算法

两阶段目标检测算法如图 2-2 所示，往往采用了“先生成候选框，再对候选框进行分类”的策略，与一阶段目标检测算法相比，能够实现更高的检测精度，但同时也带来了更复杂的网络结构与模型参数。二阶段目标检测算法代表算法有：Faster R-CNN、Mask R-CNN 等。两阶段目标检测算法的第一步是通过 RP 生成一些候选框，候选框中可能包括了待检测的目标，在得到候选框后，第二步就是通过卷积神经网络来对每个候选框进行特征的提取，特征提取过后，通过分类器与回归器实现对候选框内的物体进行种类的判断与精确定位。综上所述，由于二阶段目标检测算法的检测过程中需要生成候选框并进行二次处理，因此相较于一阶段而言检测速度更慢，且计算量也更高，难以部署再资源受限的硬件平台当中。

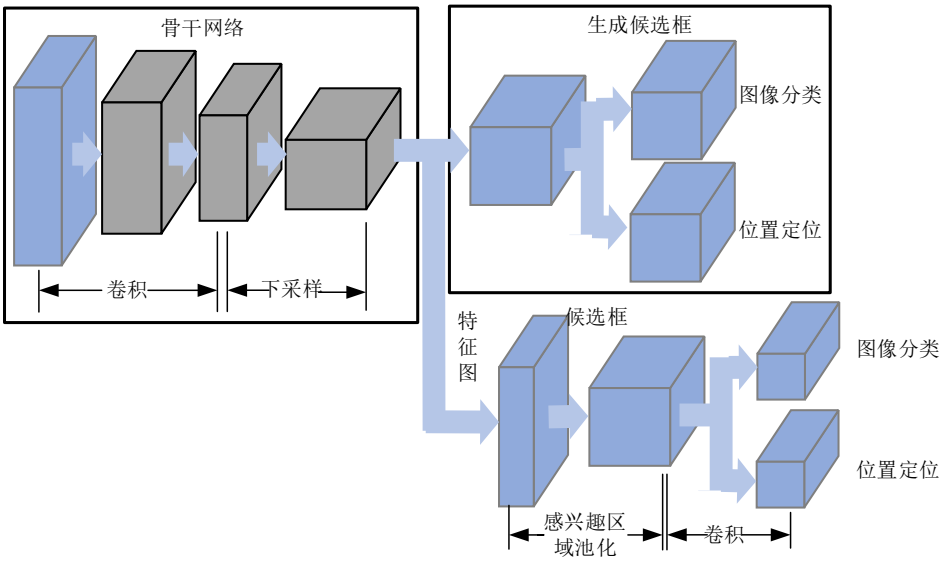


图 2-2 二阶段目标检测算法图

2.2 YOLOv3-tiny 算法介绍

本文选用了 YOLOv3-tiny 算法为基础算法，下面将从卷积神经网络角度对 YOLOv3-tiny 算法进行详细介绍。

2.2.1 卷积层

卷积层是 CNN 的主要组成部分，主要用于将图像的每个像素的特征信息进行提取，通过卷积运算，将输入给卷积层的特征图数据与卷积核进行卷积操作，输出一个卷积计算结果[54]。卷积运算的过程可以看作是卷积核不断的沿着水平方向与垂直方向进行滑动，然后与对应的特征图像素数据进行乘法计算并累加，进而得到了特征图对应位置的输出，卷积运算过程如图 2-3 所示。

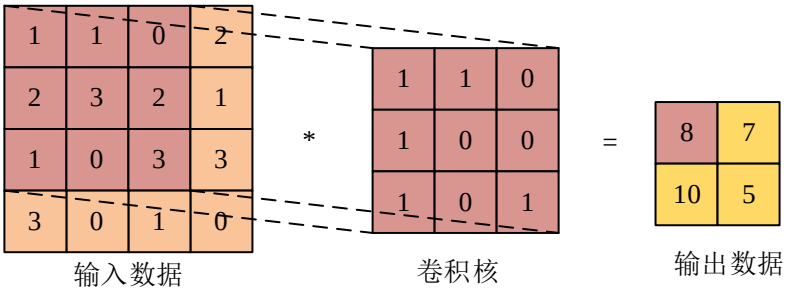


图 2-3 卷积运算过程

卷积运算的参数包括了卷积核尺寸(Kernal，K)，深度，滑动窗口步长(Stride，S)，边缘填充(Padding，P)等，其中卷积核尺寸大小为 N×N，一般为 3×3 和 1×1，

深度表示卷积层中的卷积核或者特征图的数量，步长表示滑动窗口每次移动的距离，边缘填充表示在数据的边缘进行添加一些值，往往添加 0 值。输出数据的长宽计算公式如下：

$$O=\frac{M-N+2P}{S}+1 \tag{2-1}$$

其中，O 为输出特征图的边长，M 为输入特征图的边长，N 为卷积核的尺寸大小，P 为边缘填充的大小，S 为滑动窗口步长。

2.2.2 激活函数

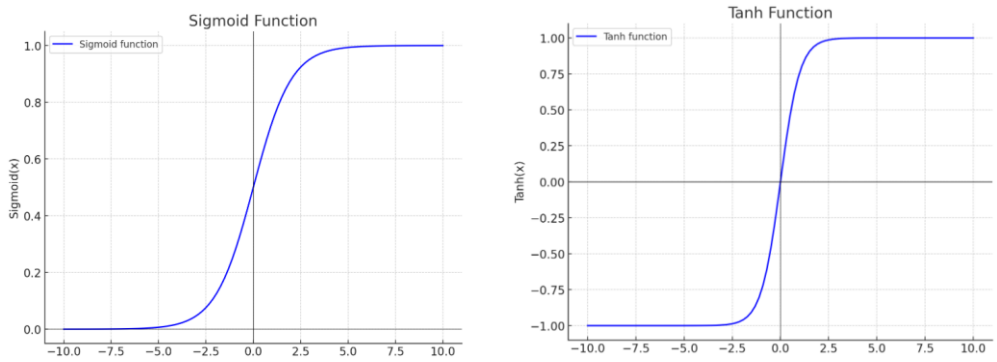
在卷积神经网络中，激活函数通常应用在神经网络的隐藏层与输出层，通过激活函数实现了非线性变换，使神经网络能够学习和表达更复杂的函数。在这方面，激活函数的选择的设计对于神经网络的性能起到了关键作用。因为激活函数的主要作用就是在输出层实现了非线性变换，非线性变换能够帮助网络更好地学习和解决现实世界中的非线性问题，通过在神经网络的每个输出层上加入激活函数，也就给网络添加了非线性操作，进而大大提高了网络模型的表达能力和学习能力。其中，常见的激活函数包括 Sigmoid、Tanh、ReLU、Leaky ReLU 等。

(1)Sigmoid 函数将输入值压缩到(0,1)区间内，因此，非常适用于二分类问题的输出层。然而，Sigmoid 函数存在梯度消失的问题，即在输入值的绝对值很大的时候，函数的梯度接近于 0，会导致网络在反向传播时难以更新权重，使网络模型训练变慢甚至停止。该函数图像如图 2-4(a)所示，函数的表达式如下：

$$Sigmoid(x)=\frac{1}{1+e^{-x}} \tag{2-2}$$

(2)Tanh 函数是 Sigmoid 函数的改进版本，其输出范围更大在(-1，1)之间，且函数图像关于坐标原点对称。由于 Tanh 函数在原点周围的梯度更大，且输出均值接近于 0，因此在实际应用中，该函数的表现往往优于 Sigmoid 函数。然而，Tanh 函数仍然存在梯度消失的问题。该函数图像如图 2-4(b)所示，函数的表达式如下：

$$\mathrm{Tanh}(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2-3}$$



(a)Sigmoid 函数图像　　(b)Tanh 函数图像

图 2-4 Sigmoid 和 Tanh 函数图像

(3)ReLU 函数因其简单高效的属性在近年来是非常受欢迎的激活函数。ReLU 函数的输出等于输入的正部分，即输出与正输入相同，当输入为负时，输出 0 值。其函数表达式如式(2-4)所示：

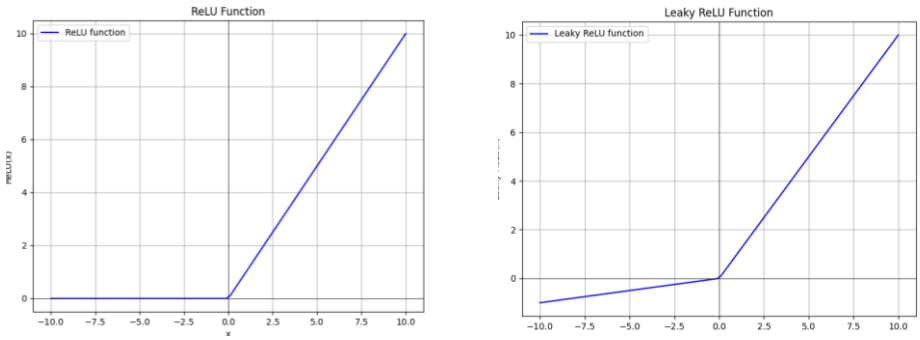
$$\mathrm{Re}\,LU(x)=\begin{cases}x,\ x\geq 0\\ 0,\ x<0\end{cases} \tag{2-4}$$

这一特性使 ReLU 函数在正区间内避免了梯度下降的问题，同时加快了网络模型的训练速度。然而，该函数对于负输入不进行任何的响应，可能会引发部分神经元在训练时不对任何数据激活。ReLU 函数图像如图 2-5(a)所示。

(4)Leaky ReLU 函数是 ReLU 函数的改进版本，通过给负值输入赋予一个较小的正梯度，而不是输出 0，加强了模型的处理能力。其函数表达式如式(2-5)所示：

$$Leaky(x)=\begin{cases}x,\ x\geq 0\\ \lambda x,\ x<0\end{cases}\quad(0<\lambda<1) \tag{2-5}$$

Leaky ReLU 函数解决了 ReLU 函数的可能存在的部分神经元在训练时不对任何数据激活的情况，进而提高了网络训练的稳定性和准确度。Leaky ReLU 函数图像如图 2-5(b)所示，图中的在输入为负时的 λ 值为 0.1。



(a)ReLU 函数图像　　(b)Leaky ReLU 函数图像

图 2-5 ReLU 和 Leaky ReLU 函数图像

2.2.3 池化层

池化层又称下采样，是 CNN 的常见组成部分，卷积神经网络在处理大规模图像数据时需要对于图像数据进行降低数据量处理，通过减小特征图尺寸，进而降低了网络模式的参数量，简化网络模型复杂度并防止网络的过拟合现象的发生，同时也保留了特征图的主要特征信息。

常见的池化有最大池化与平均池化。最大池化通过从滑动窗口中选择最大值作为输出，保留了窗口中最显著的特征，如图 2-6 所示。平均池化通过从滑动窗口中计算平均值作为输出，保留了窗口中最广泛的特征信息，如图 2-7 所示。这种选择性的特征保留方式有助于网络保留最关键的信息，进而提高了网络模型的性能与效率。

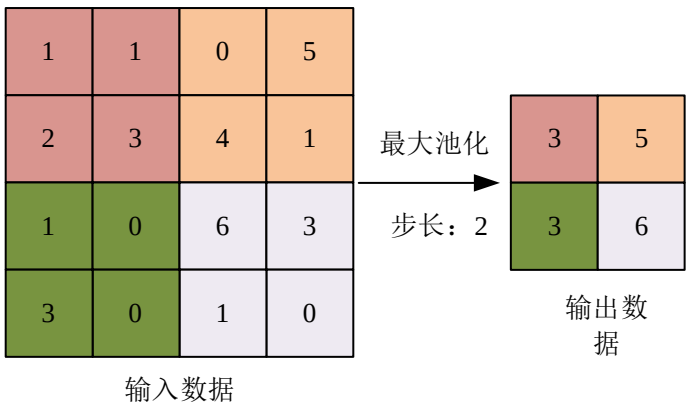


图 2-6 最大池化图

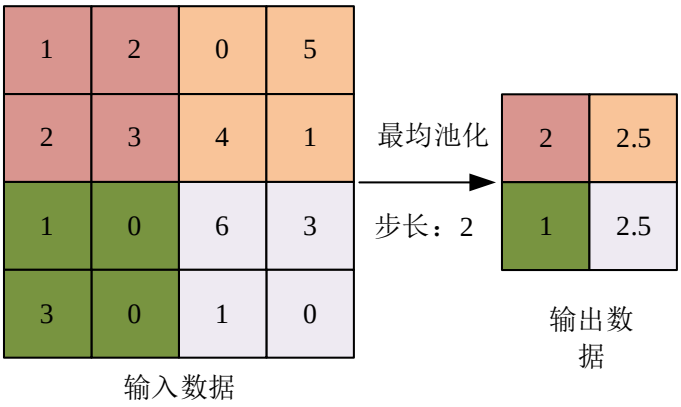


图 2-7 平均池化图

2.2.4 批量归一化

在 YOLOv3-tiny 网络中，批量归一化操作往往用在卷积层的输出数据上，且在激活函数应用之前进行，通过对于每个卷积层的输出进行批量归一化操作，显著提高了网络模型的收敛性，有助于规范化网络模型。假设单次训练中，网络模型包含 n 个样本，特征向量为 x_1、x_2、…、x_n。根据公式(2-6)，对这组输入数据执行批量归一化操作，输出归一化处理后的特征值 $\hat{x}_i$ 。

$$\hat{x}_i = \frac{x_i - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}} \tag{2-6}$$

其中，根据式(2-7)得到该批样本数据的均值 μ_j；根据式(2-8)得到该批样本数据的方差 σ_j^2； ε 为一个极小值，与方差相加为了避免分母为 0。

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2-7}$$

$$\sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_j)^2 \tag{2-8}$$

然后，将执行批量归一化处理后的数据再由式(2-9)执行一次仿射变换操作，最终得到批量归一化层的最终输出 y_i。其中，γ 为可训练的缩放参数，β 为可训练的平移参数。

$$y_i = \gamma \hat{x}_i + \beta \tag{2-9}$$

2.2.5 检测层

在 YOLOv3-tiny 中，检测层在网络的最后几层，作为执行目标检测任务的核心组件。该网络包含两个检测层，分别作用于不同大小的特征图，以识别不同大

小的目标。每个检测层用于预测一组预定义的锚框。每个锚框的参数包含了边界框的位置，类别概率与置信度。检测层输出一个特征图，特征图中的每个单元对应着输入特征图的一部分，输出经过后处理后，转换为了输入特征图上的真实边界框和目标类别的概率。但是，最终往往会得到很多的边界框，其中大多数边界框的多余的，因此，还会采用非极大值抑制(Non-Maxima Suppression，NMS)算法，来去除多余的边界框，保留最优的检测结果。

2.3 YOLOv3-tiny 算法改进

YOLOv3-tiny 算法在杂物较多存在遮挡的背景环境下进行目标检测推理时，可能由于在特征提取过程中存在特征信息丢失以及无法充分融合全局特征与局部特征的问题，进而导致网络检测精度不高，存在漏检的情况。针对这一情况，本节对于 YOLOv3-tiny 算法进行了改进[55]。首先，替换了损失函数，采用广义交并比替换交并比，更好的反映预测框与真实框之间的重合度，同时也可以作为一个距离度量指标。然后，对于 YOLOv3-tiny 算法的主干网络进行了修改，用三个卷积层替代池化层，减少了特征信息的丢失；并在主干网络的末尾添加了改进后的 SPP(Spatial Pyramid Pooling，空间金字塔池化)，有效提高了模型对目标的表达能力，并且充分提取了局部特征信息与全局特征信息，进而提高了检测精度。

2.3.1 替换损失函数

IOU 是 YOLOv3-tiny 目标检测中最常用的评估指标之一。它不仅用于衡量正负样本，还可以衡量预测框与真实框之间的相互重叠比例的情况与距离情况。计算公式如式(2-10)所示，其中 A 为真实框，B 为预测框。

$$I_{OU} = \frac{A \cap B}{A \cup B} \tag{2-10}$$

由式(2-11)可知，当预测框与真实框无重叠时，IOU 值为 0，此时就无法体现二者之间的距离了。此时，如果直接将 IOU 用作损失函数的话，梯度值就为零，导致无法优化边界框不相交的情况。

因此，为了解决非重叠区域的问题，本设计采用了 GIOU 替换原有的 IOU。GIOU 通过引入预测框和真实框的最小外接矩形来获取两个框在最小外接矩形

中各自的比重，进而解决了两个目标在不相交时梯度为零的问题，其计算公式如(2-11)所示：

$$G_{IOU} = I_{OU} - \frac{|C-(A\cup B)|}{C} \tag{2-11}$$

式中：C是指两个框的最小外接矩形的面积，指内部包含了两个框的最小矩形的面积；$A\cup B$是指两个框之间并集面积;I_{OU}是指原有的 IOU 值。原有 IOU 取值区间为[0,1]，而 GI0U 的取值区间为[-1,1]。而 GIoU Loss 的计算公式为：$L_{GIOU}=1-G_{IOU}$，其特征表现为非负性与不确定性。GIOU 作为 IOU 的下界，当两个框完全重合时，GIOU 与 IOU 的值均为 1;而在两个框完全不重叠的情况下，IOU 的值为 0，而 GIOU 的值为负值，且收敛于-1。

假设预测框为$K^M=(x_1^M,y_1^M,x_2^M,y_2^M)$，真实框为$K^N=(x_1^N,y_1^N,x_2^N,y_2^N)$，计算$K^M$和$K^N$的面积分别是$A^M$、$A^N$，当$x_1^M<x_2^M$、$y_1^M<y_2^M$、$x_1^N<x_2^N$和$y_1^N<y_2^N$满足时，公式(2-12)与公式(2-13)成立。

$$A^M=(x_2^M-x_1^M)\times(y_2^M-y_1^M) \tag{2-12}$$

$$A^N=(x_2^N-x_1^N)\times(y_2^N-y_1^N) \tag{2-13}$$

再计算K^M和K^N与交集面积I，当$x_1^I=max(x_1^M,x_1^N)$、$x_2^I=min(x_2^M,x_2^N)$、$y_1^I=max(y_1^M,y_1^N)$、$y_2^I=min(y_2^M,y_2^N)$，且$x_2^I>x_1^I$和$y_2^I>y_1^I$满足时，公式(2-14)成立。

$$I=(x_2^I-x_1^I)\times(y_2^I-y_1^I) \tag{2-14}$$

假设预测框K^M与真实框K^N的最小外接矩形$K^C=(x_1^C,y_1^C,x_2^C,y_2^C)$，当$x_1^C=min(x_1^M,x_1^N)$、$x_2^C=max(x_2^M,x_2^N)$、$y_1^C=min(y_1^M,y_1^N)$、$y_2^C=max(y_2^M,y_2^N)$满足时，计算最小外接矩形面积$C$如式(2-15)所示：

$$C=(x_2^C-x_1^C)\times(y_2^C-y_1^C) \tag{2-15}$$

综上所述，IOU 与 GIOU 的计算公式如下所示：

$$I_{OU}=\frac{I}{A^M+A^N-I} \tag{2-16}$$

$$G_{IOU}=I_{OU}-\frac{C-(A^M+A^N-I)}{C} \tag{2-17}$$

由式(2-16)与式(2-17)可知，IOU 只关注了两个框的重叠区域，而 GIOU 在关

注重叠区域的同时也关注了非重合区域。因此，GIOU 能更全面地衡量两个框之间的相似度，更准确地反映它们的重合度。同时，GIOU 更适合作为距离度量指标，因为它解决了在两个框不重叠时(即 IOU=0)导致无法优化的问题，从而保证训练能够持续进行。综上所述，将 GIOU 损失作为边界框坐标的回归损失，不仅反映了两个框之间的重叠与非重叠区域情况，还是一个良好的距离度量指标，提高了预测的准确性。

2.3.2 修改主干网络

如图 2-8 所示，原始 YOLOv3-tiny 算法的主干网络是由 7 层 DBL 层与 6 层池化层交替构成了特征提取网络。其中，DBL 层是由卷积层，批量归一化层和激活层构成，其中卷积核尺寸大小为 3×3，激活函数选用了 Leaky ReLU 激活函数。池化层选用了 2×2 的最大池化方式，通过最大池化，降低了特征图的尺寸。最后的两个检测层输出的尺寸大小为 13×13 与 26×26。网络模型在特征提取时，本身模型参数量有限，且池化过程中会进一步缩小特征图的尺寸，因此会造成在提取特征过程中，出现大量的特征丢失，尽管网络中使用了多尺寸融合的方法，但是锚框数量有限以及样本特征的尺寸跨度不够，仍会存在杂物较多存在遮挡的背景环境下，存在漏检的现象。

原始网络结构中，输入先通过卷积操作，使得特征图逐渐减小，然后在经过 2×2 最大池化时，通过池化操作选取了四个像素值中最大的像素值，进而浪费了上一层的另外三个像数值。

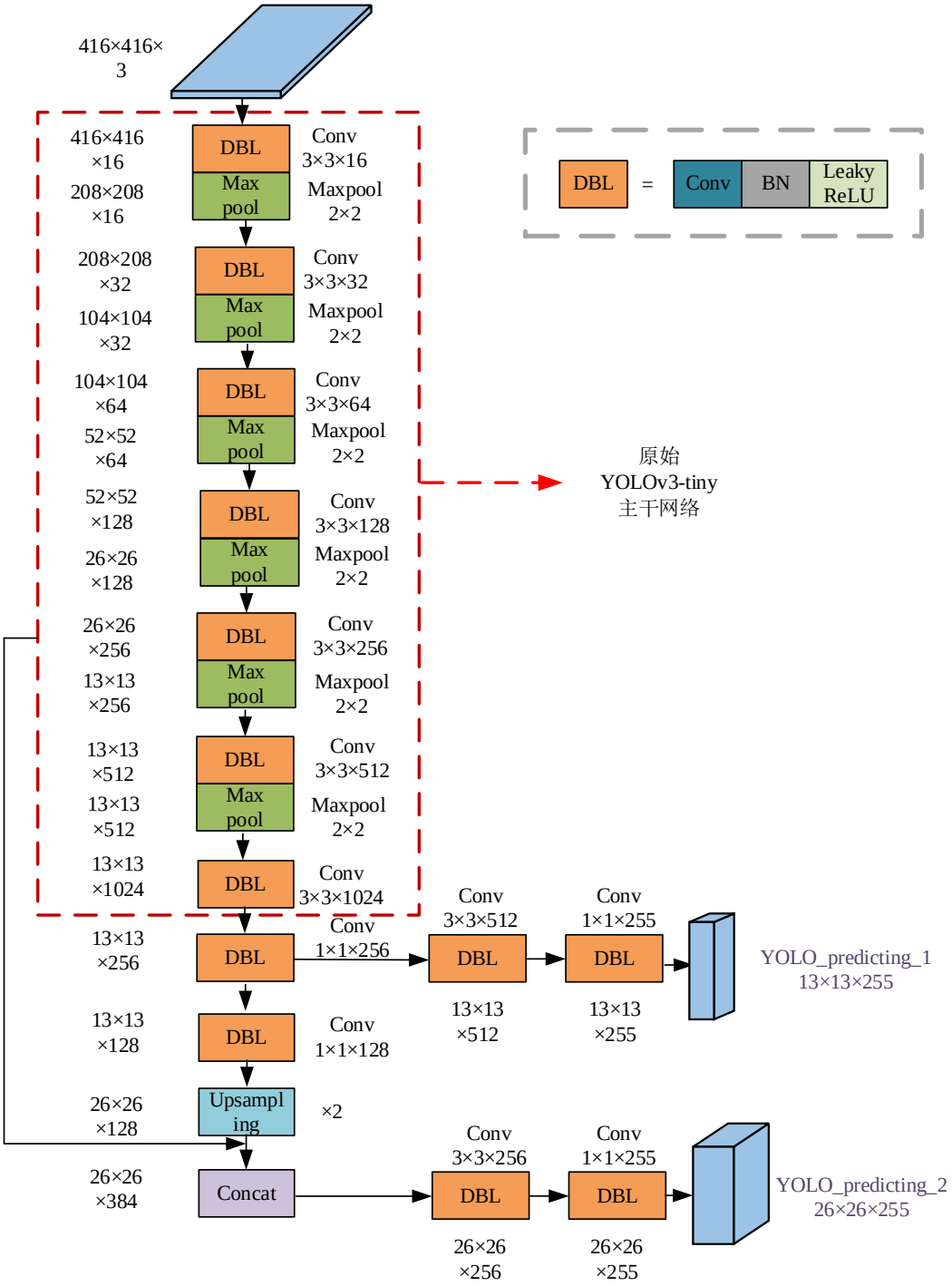


图 2-8 原始 YOLOv3-tiny 算法网络结构图

因此，本文设计了一种改进的 YOLOv3-tiny 主干网络如图 2-10 所示。新的主干网络采用 3 个步长为 2 的 3×3 卷积层替代了网络中的前三个池化层，卷积层相较于池化层完整保存了特征信息。步长为 2 的卷积操作在每次计算后将特征图尺寸缩小一半，不仅实现了池化层的降维效果，还能够保留上一层的全部特征信息，共同参与后续计算。这种方法有效减少了特征丢失，进而提高了模型对待

检测目标的表达能力。

同时，为了更好的融合全局特征以及局部特征信息，本文还在改进后的主干网络的末尾加入了改进后的空间金字塔。传统的空间金字塔池化旨在解决全连接层中的输入必须为固定的特征向量的问题，使网络能够适应不同大小的输入特征，而无需裁剪或缩放。本文采用的改进后的空间金字塔的池化模块，将多尺寸的局部特征信息与全局特征相结合，从而提取到了更加丰富的特征信息，其结构示意图如图 2-9 所示。优化后的 SPP 模块通过步长为 1 的池化操作行与边缘填充，实现特征图大小保持不变。具体而言，首先从主干网络中提取的 13×13×1024 特征图已具备丰富的语义信息。然后，经过三次最大池化操作，生成了 3 种特征图，并在通道维度上与原始特征图 13×13×1024 进行拼接，最终得到尺寸 13×13×4096 的输出特征图，其拥有更加丰富的特征信息，进而提高了对待测目标的检测效果。其中池化操作采用了 5×5、9×9、13×13 三种尺寸，且步长值为 1，池化操作后对于特征图进行了拼接。

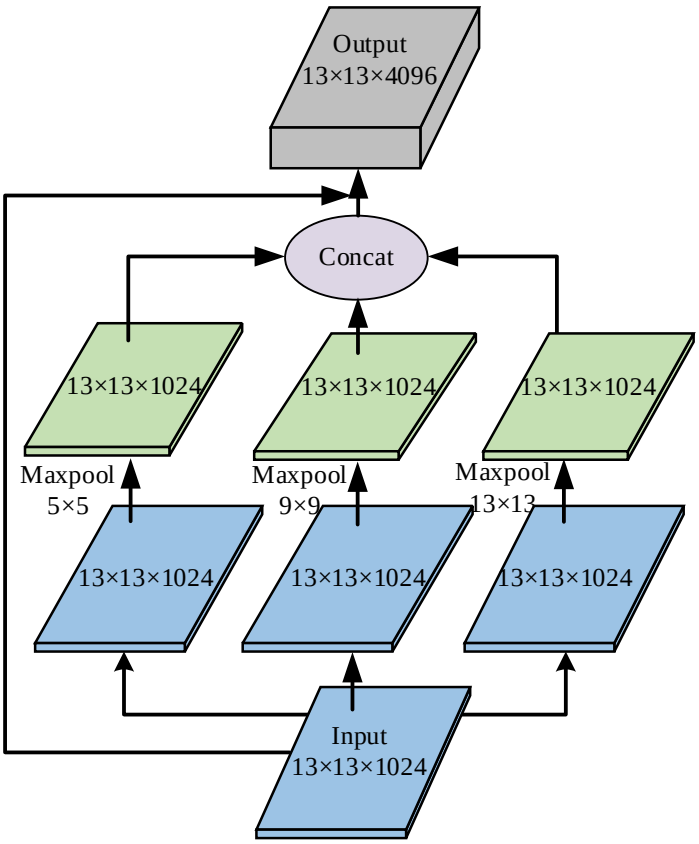


图 2-9 改进的空间金字塔池化模块

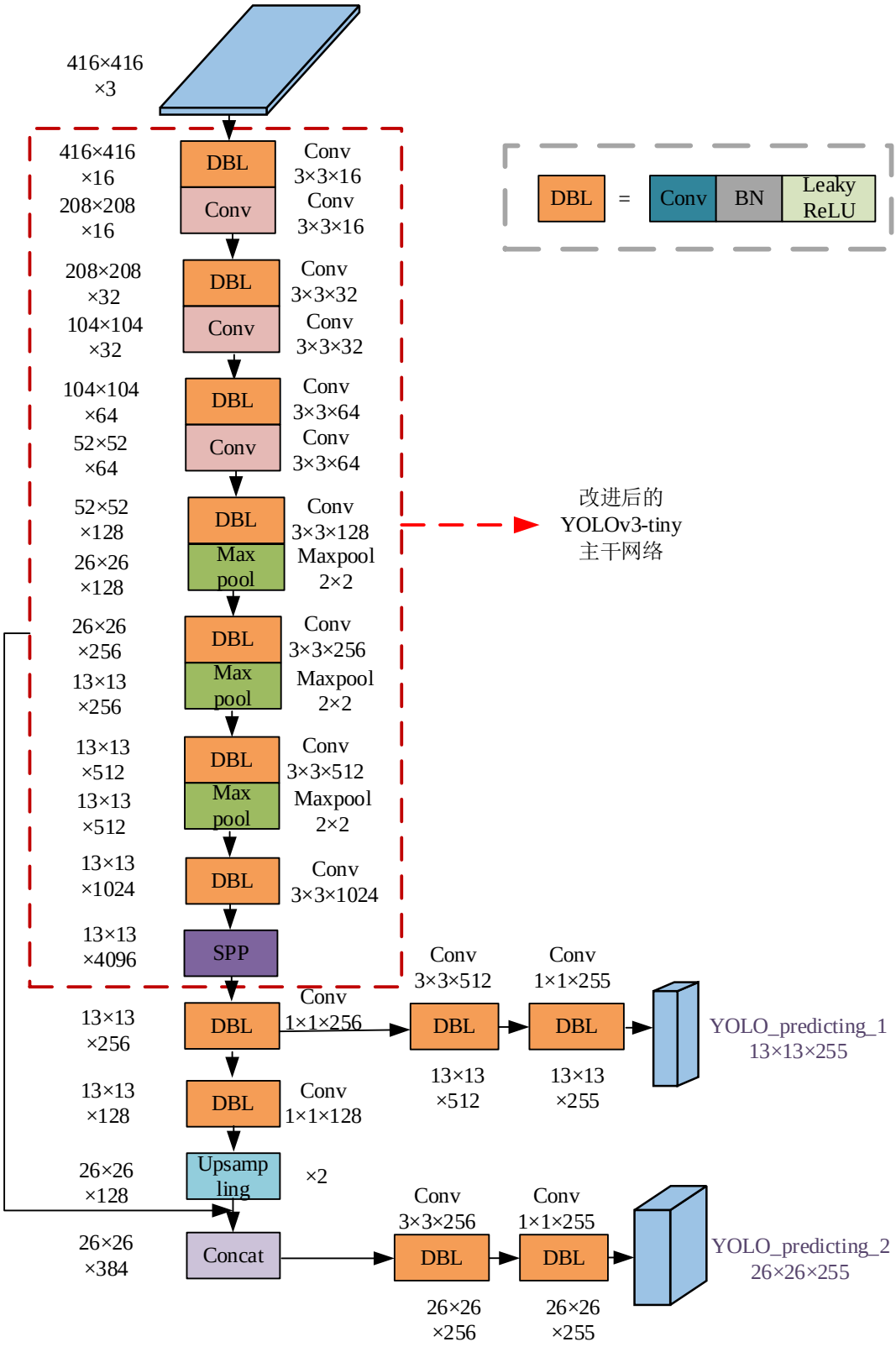


图 2-10 改进后的 YOLOv3-tiny 算法网络结构图

由图 2-10 可知，为了解决 YOLOv3-tiny 在杂物较多存在遮挡的背景环境下进行目标检测推理时检测精度不高，存在漏检的问题，本设计对于主干网络进行

了修改。首先，为了减少池化操作带来的特征丢失，将主干网络中的前三个最大池化层换成了三个步长为 2 的 3×3 卷积层，有效提高了模型对目标的表达能力；然后，为了提高检测效果，在主干网络的末尾添加了改进后的 SPP 模块，进而充分提取了局部特征信息与全局特征信息。

2.3.3 实验结果

目前主流的目标检测 COCO 数据集共包含了 80 个种类，本实验从 COCO 数据集中抽取出了三种类别：苹果、香蕉和胡萝卜，数据集共 4830 张图片，如图 2-11 所示，其中，数据集随机以 7:1:2 的比例分成训练集，验证集、测试集。本实验在 CPU 为 Intel(R) Core(TM) i5-8300H，GPU 为 NVIDIA GeForce 1050ti(4G)，运行内存为 32G 的电脑上进行。主要任务包括：首先比较损失函数替换前后对于检测效果的影响；然后比较主干网络修改前后对于检测效果的影响。

(1)损失函数替换对于模型检测效果的影响

为了探究损失函数由 IOU 损失替换为 GIOU 损失对于模型的检测效果的影响，本文分别使用了 YOLOv3-tiny 模型对于采用 IOU 损失与 GIOU 损失进行了训练验证，两次实验的其他条件均保持一致。实验结果表明，损失函数由 IOU 损失替换为 GIOU 损失对于 FPS(Frames Per Second,每秒帧数)仅有较小的影响，而对于检测的平均精确度(Mean Average Precision，mAP)有了较大的提升，如下表 2-1 所示。

表 2-1 不同损失函数实验结果

损失函数	mAP	FPS
IOU 损失	78.36%	169.1
GIOU 损失	80.28%	168.3

(2)主干网络修改对于模型检测效果的影响

为了探究模型的主干网络修改对于检测效果的影响，本文在采用 GIOU 损失的基础上，对于 YOLOv3-tiny 模型的主干网络修改前后进行了训练验证，两次实验的其他条件均保持一致。实验结果如表 2-2 所示，模型在主干网络修改后，仅以计算量 FLOPS(Floating Point Operations Per Second，每秒浮点运算次数)增加了 0.04G，参数量 Params(Parameters，参数)增加了 0.65M 为代价，使得 mAP 提高

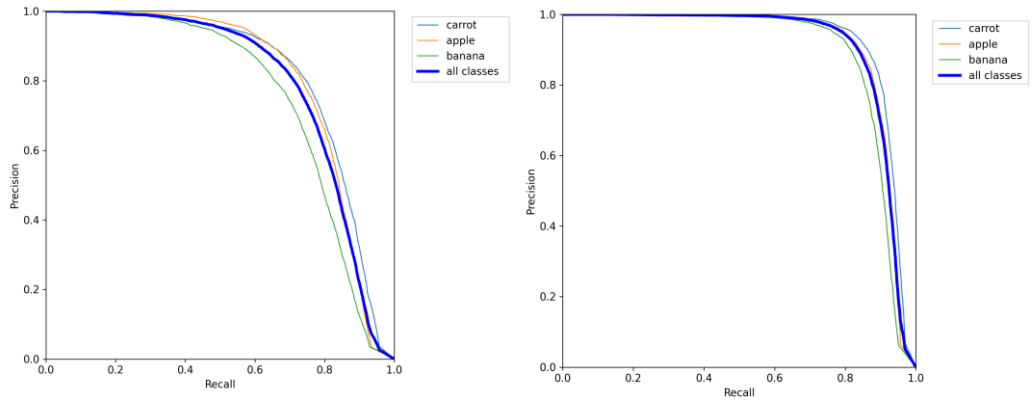
了 5.08%，精确度(Precision,P)提高了 7%，召回率(Recall,R)提高了 8%，F1 值提高了 7%，FPS 略微下降,但仍能达到 164FPS 的帧率，满足目标检测实时性的要求，实现了对于待测目标的检测效果的显著提升。

表 2-2 YOLOv3-tiny 模型主干网络修改前后性能对比

模型	mAP	P	R	F1	FPS	FLOPS/G	Params/M
原始 YOLOv3-tiny	80.28%	0.76	0.72	0.74	168.3	5.56	8.85
修改后 YOLOv3-tiny	85.36%	0.83	0.80	0.81	164.6	6.00	9.50

如图 2-12 所示，图中显示了改进前后 YOLOv3-tiny 的 PR(Precision-Recall，精确度—召回率)曲线图对比图，其中曲线与横纵坐标所围成的面积可以表示为平均精度 AP，面积越大代表模型的性能越好，图右侧表示三种目标对应的曲线颜色。从图中可以看出改进后的 YOLOv3-tiny 模型的所有种类所围成的面积都要大于原始的 YOLOv3-tiny 模型，这说明改进后的 YOLOv3-tiny 模型对于目标的检测效果要优于原始模型。

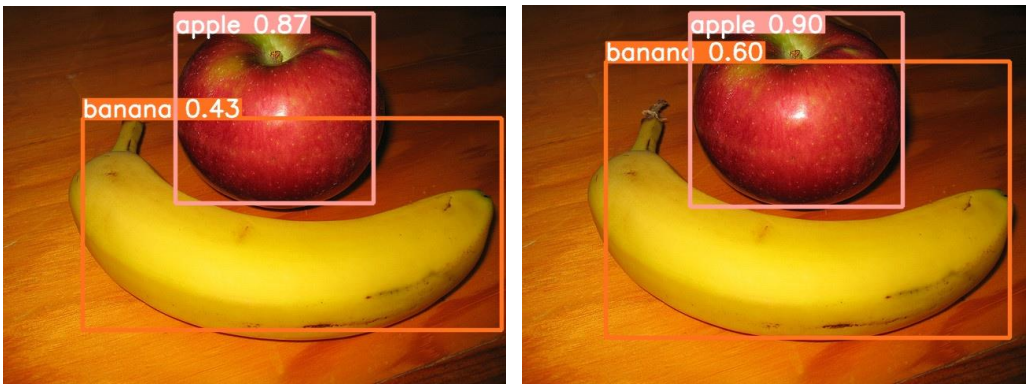
图 2-11 苹果香蕉胡萝卜数据集



(a)改进前的 PR 曲线　　(b)改进后的 PR 曲线

图 2-12 YOLOv3-tiny 模型改进前后的 PR 曲线图

图 2-13 展示了改进前后的模型的检测效果图，从图中可以看出，改进后的 YOLOv3-tiny 相较于改进前提高了对于水果的检测精度。



(a)YOLOv3-tiny　　(b)改进后的 YOLOv3-tiny

图 2-13 模型改进前后检测效果对比图

2.4 YOLOv3-tiny 模型 8 位定点量化

YOLOv3-tiny 模型训练得到的模型参数类型为 32 位的浮点数据类型，而浮点运算往往在 FPGA 上难以部署实现，需要消耗大规模的计算资源，且未经过量化的模型需要更高的存储带宽，对于 FPGA 的存储资源要求很高，因此通常需要将浮点数据量化为定点数据。本文将改进后的 YOLOv3-tiny 模型参数从 32 位浮点数据转化为 8 位定点数据，以减少对于 FPGA 的计算资源与存储资源的消耗。

本设计采用了线性量化，在进行线性量化前，需要在每层之间加入直方图观察点，并准备一些输入数据样本用于前向推理；在进行前向推理时，通过直方图观察点对于数据的分布情况进行记录，通过计算得到与量化相关的参数，如缩放

因子。如图 2-14 所示，实现了将 Float32 模型量化为 INT8 模型，其中，Q 表示量化操作后的整数值，F 表示量化前的浮点数，Z 表示量化后的零点所在位置，S 表示缩放因子。缩放因子与量化后的零点所在位置由公式(2-18)与公式(2-19)得到。

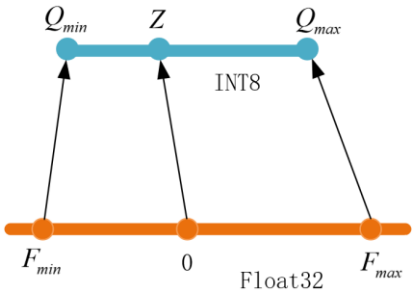


图 2-14 INT8 线性量化原理图

$$S = \frac{F_{\max} - F_{\min}}{Q_{\max} - Q_{\min}} \tag{2-18}$$

$$Z = round(Q_{\max} - \frac{F_{\max}}{S}) \tag{2-19}$$

模型经过量化后的推理过程可由下面的例子说明。卷积运算本质上就是一种乘加运算的组合，如式(2-20)所示：

$$F_3^{i,k} = \sum_{j=1}^{N} F_1^{i,j} F_2^{j,k} \tag{2-20}$$

通过将上述式子中的浮点数据转换成量化后的整数数据，然后进行一些调整得到下列式子：

$$S_3(Q_3^{i.k} - Z_3) = \sum_{j=1}^{N} S_1(Q_1^{i,j} - Z_1) S_2(Q_2^{j,k} - Z_2) \tag{2-21}$$

$$Q_3^{i.k} - Z_3 = \frac{S_1 S_2}{S_3} \sum_{j=1}^{N} (Q_1^{i,j} - Z_1)(Q_2^{j,k} - Z_2) \tag{2-22}$$

由上述式子可得，只有 $G = \frac{S_1 S_2}{S_3}$ 为浮点数据，其余运算均是 INT8 类型的定点运算。再通过浮点数据自身的存储特性，利用 frexp 函数可将因子 G 分解为尾数与指数两部分。

$$G = 2^{-n} G_0 \tag{2-23}$$

通过上述操作，就成功的将与G_0的乘法运算改成了定点运算，同时与2^{-n}的乘法运算可转变为通过右移来实现。经过上面这些公式的推导验证，就将整个模型的推理过程，由浮点运算转变为 INT8 类型的定点运算，很大程度上减少了推理运算过程中消耗的计算资源与存储资源。经过所有量化操作过后，改进后的 YOLOv3-tiny 模型训练得到的所有参数数据就通过二进制的.bin 文件进行存储，如图 2-15 所示，后续将.bin 文件通过读卡器保存到 SD 卡中，再将 SD 卡插入到 ZYNQ 的 PS 端，通过 PS 端实现对于模型参数数据的读取。

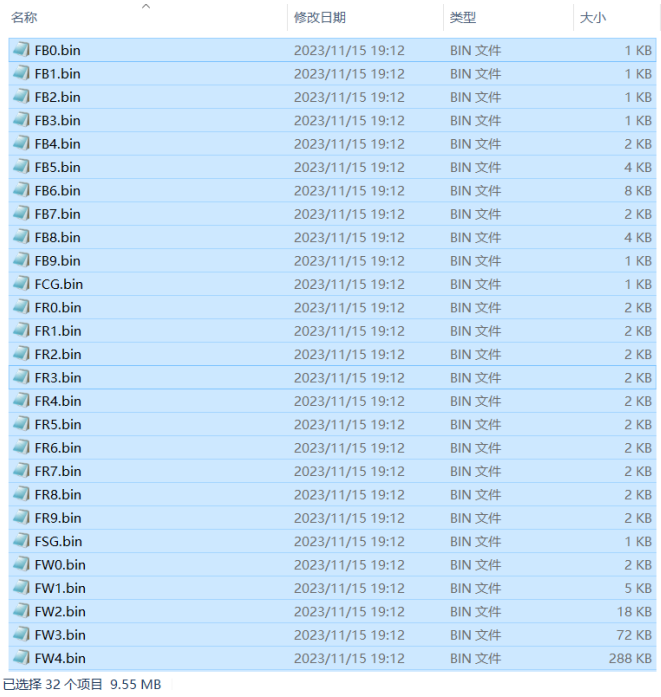


图 2-15 量化后得到的.bin 文件

2.5 本章小节

本章主要在算法方面进行了介绍与改进。首先介绍了基于深度学习的两种目标检测算法。其次详细介绍了一阶段目标检测算法 YOLOv3-tiny 的原理。然后针对基于 ZYNQ 平台的实现的 YOLOv3-tiny 算法在杂物较多存在遮挡的背景环境下进行目标检测推理过程中检测精度不高，存在漏检的问题，对于 YOLOv3-tiny 算法进行了替换损失函数与修改主干网络两方面的改进，提高了目标检测的检测精度。最后将 YOLOv3-tiny 模型进行了 8 位的定点量化处理，为后续在 ZYNQ 平台上搭建整个目标检测系统奠定了基础。

第 3 章 卷积加速器设计

本章首先阐述了卷积加速器相关的概念；然后说明了本系统所使用的卷积加速器的总体架构；再详细介绍加速器中各模块的设计，如计算模块、缓存模块、控制模块；并提出了一些卷积加速器方面的优化策略；最终对于设计的卷积加速器进行了仿真测试。

3.1 卷积加速器概念介绍

3.1.1 典型的卷积加速器架构

典型的卷积加速器架构如图 3-1 所示，主要用于在硬件端实现卷积神经网络的推理计算过程，一般包括了 DMA(Direct Memory Access，直接存储访问)、计算单元、缓存单元等，其中缓存单元分为了片上缓存以及外部存储器。DMA 通常用于使外设和内存之间的数据传输不需要 CPU 的干预，DMA 配置完成后，释放 CPU 来执行其他操作，减少了 CPU 的负担。计算单元通常用于进行目标检测过程中的推理计算，包括了卷积计算、池化计算等。片上缓存通常用于存储中间计算结果和局部数据，能够提供快速的数据访问；外部缓存器通常用于缓存特征图数据、模型参数等，能够提供更大容量的存储空间。卷积加速器在工作时，首先，将需要检测的特征图数据从外部存储器传输到输入特征图缓存区中，同时也将模型的权重参数也加载到权重缓存区当中。然后，将缓存区中的特征图数据与权重参数传输到计算单元上，计算单元进行推理计算，推理计算的结果传输到输出特征图缓存区中，并在需要时保存到外部寄存器中。卷积加速器在工作时存在延时，包括了存储访问延时、计算延时、传输延时等，延时的存在影响着卷积加速器的性能。因此，往往需要对于延时进行优化处理，来尽可能降低延时，包括了循环展开、流水线设计等操作。通过多重循环展开，将多个迭代过程合并成为一个大的迭代过程，实现了不同并行策略，提高了工作的并行度。流水线设计是将推理计算的过程分成了多个阶段，在推理计算时，在同一时刻能够同时完成多个阶段的计算工作，进而提高了吞吐量以及工作效率。

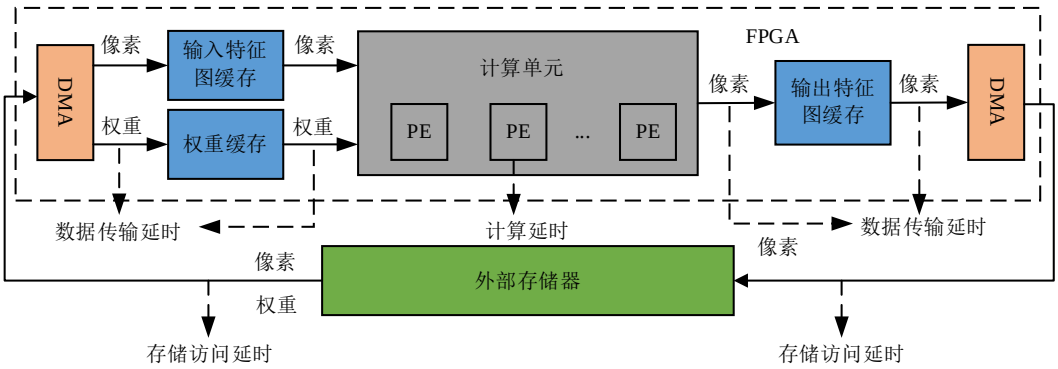


图 3-1 典型卷积加速器架构

3.1.2 并行度分析

卷积运算的加速，主要体现在对于运算过程实现了并行处理策略，减少了计算时间，降低了资源消耗。在卷积运算中，卷积窗口通过滑动执行了并行计算，因此可以通过同时处理多个卷积窗口的计算，进而高效利用硬件资源，加快模型的训练与推理任务。卷积运算过程存在 4 个循环方面，Loop-1 是指并行展开卷积核，Loop-2 是指并行展开输入通道，Loop-3 是指并行展开输入特征图内部，Loop-4 是指并行展开输出通道，如图 3-2 所示。

```
for(int o=0;o<Cho;o++)                                  ──→ Loop-4
    for(int y=0;y<Chi_y;y++)
        for(int x=0;x<Chi_x;x++)                        }──→ Loop-3
            for(int i=0;i<Chi;i++)                      ──→ Loop-2
                for(int ky=0;ky<Nky;ky++)
                    for(int kx=0;kx<Nkx;kx++)           }──→ Loop-1
                        pixeI_L(o;x,y)+=pixeI_L-1(i;S×x+kx,S×y+ky)×weight_L-1(i,o;kx,ky);
                pixeI_L(o;x,y)+=pixeI_L(o;x,y)+bias(o);
```

图 3-2 卷积运算循环展开

(1)卷积核展开

如图 3-3 所示，卷积核展开是卷积运算中一种常见的并行计算方法，通过配置参数 Nkx 和 Nky 来控制展开的并行度。在该方法中，每个周期可以同时计算 Nkx×Nky 个输入特征图像素与对应卷积核权重的乘积，并利用加法树对这些乘法结果进行累加。累加得到的输出与之前的数据进行相加，最终得到完整的计算结果。

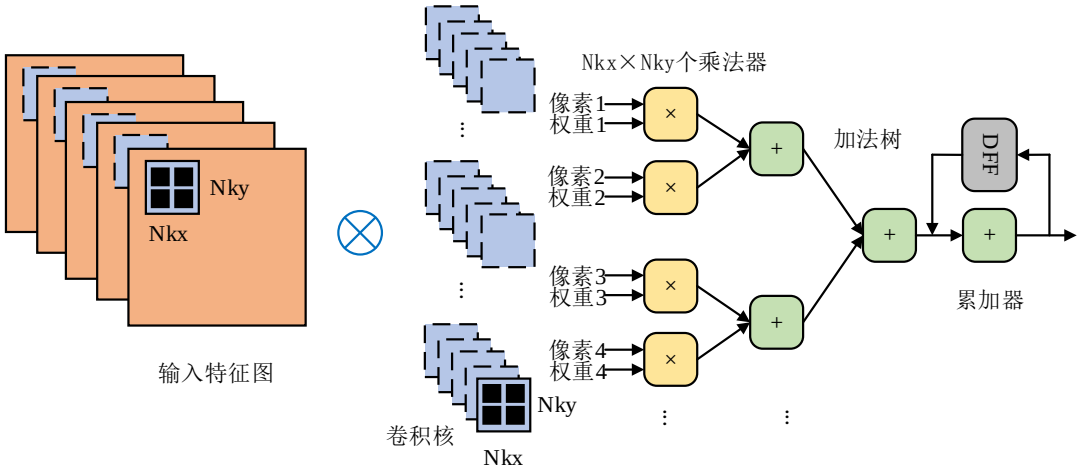


图 3-3 卷积核展开示意图

(2)输入通道展开

如图 3-4 所示，输入通道展开通过配置 Chi 参数来控制展开的并行度。每个计算周期并行计算 Chi 个输入通道的像素与对应卷积核权重的乘积。Loop-2 的计算结构与 Loop-1 相同，其并行度为 Chi，以适应按照输入通道展开。

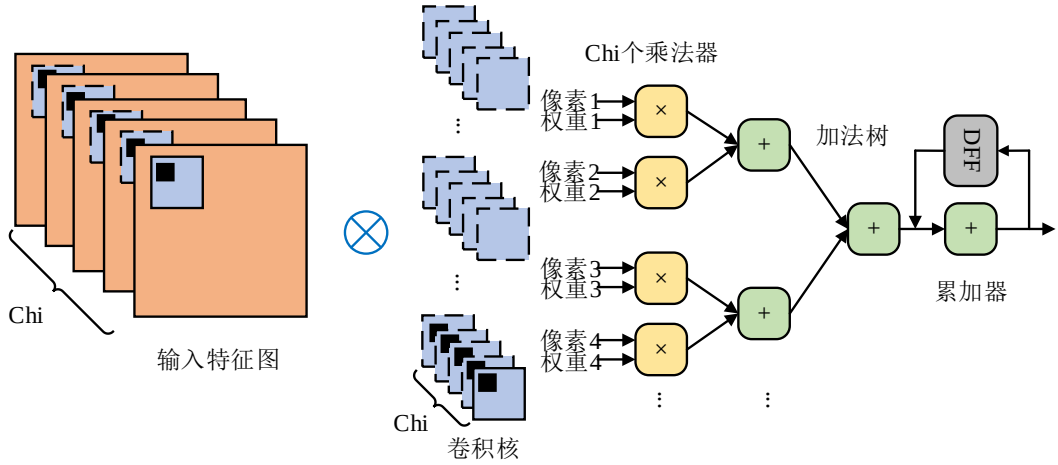


图 3-4 输入通道展开示意图

(3)输入特征图内展开

如图 3-5 所示，输入特征图内展开是通过配置 Chi_x 与 Chi_y 参数来控制展开的并行度。在卷积操作时，卷积核的权重数据通过不断滑动窗口与对应的输入特征图像素进行乘积计算，因此通过输入特征图内展开可以实现在每个计算周期选取 Chi_x×Chi_y 个像素与同一个权重数据相乘，也就实现了权重数据的 Chi_x×Chi_y 次复用。在经过 Chi_x×Chi_y 次并行乘法运算后，每个输出再进行

单独的累加操作，与前两个展开相比，输入特征图内展开未使用到加法树。

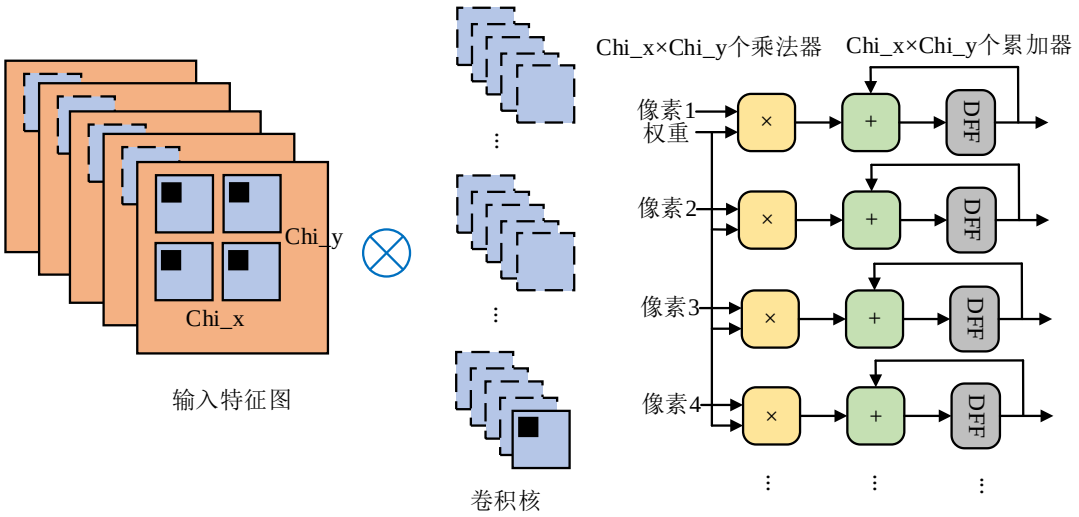


图 3-5 输入特征图内展开示意图

(4)输出通道展开

如图 3-6 所示，输出通道展开是通过配置 Cho 参数来控制展开的并行度。在每个计算周期，会选一个像素数据与 Cho 个在相同位置的权重数据进行相乘，也就实现了像素数据的 o 次复用。Loop-4 的计算结构与 Loop-3 相同，其并行度为 Cho。

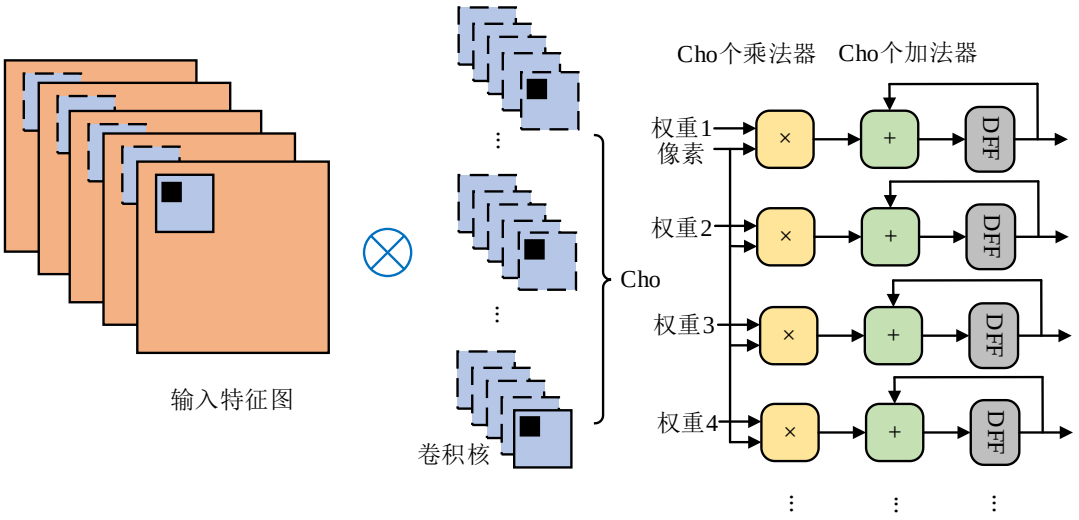


图 3-6 输出通道展开示意图

3.1.3 滑窗原理与行缓存结构

在卷积神经网络的推理计算过程中，卷积运算需要根据卷积核的大小来获取

特征图中的连续像素区域，通常称为“滑窗”操作。以一个 3×3 的卷积窗口为例，在窗口内的每个位置都需要读取对应位置的像素值，即在滑动窗口扫描数据时，一共需要读取 9 次像素数据。随着滑窗以步长为 1 进行依次滑动扫描整个特征图时，中间的像素会被重复读取 9 次，当特征图较大时，这种重复的数据访问会引入显著的延时，产生所谓的“空泡”，影响系统的实时处理能力，在实时系统中并不适用。因为滑窗在滑动扫描过程中，主要使用的是图像中相邻像素的水平和垂直方向上的数据，因此，对于水平方向的像素，可以利用寄存器对像素值进行缓存和延时处理，使得在每个时钟周期内，仅需读取水平方向上相邻的 3 个像素值，而不是每次都读取整个 3×3 窗口中的 9 个像素值。此外，还可以通过行缓存对特征图中之前行的像素数据进行缓存，这样当滑窗沿着水平方向上滑动时，可以直接从缓存中读取之前行的像素，而无需再次从外部存储器中读取，使得每个周期仅需从外部存储器中读取 1 个像素数据，进一步提高了数据访问效率，提升了卷积神经网络推理过程中的计算性能。该方案的实现原理与结构如图 3-7 所示。

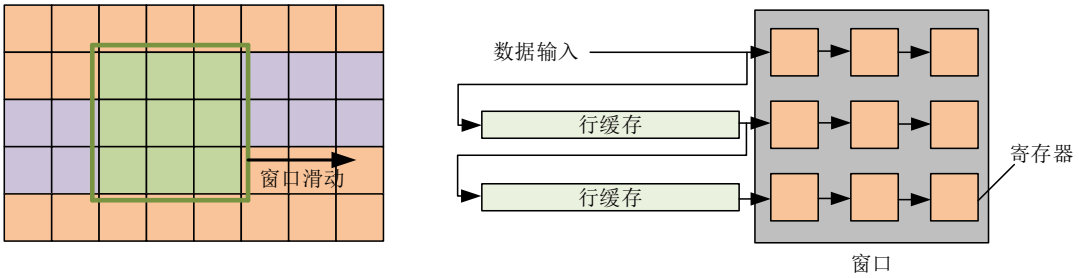


图 3-7 滑窗原理与传统行缓存结构

图 3-7 的传统行缓存结构简单，仅能实现对于固定尺寸大小的特征图的行缓存，而在改进的 YOLOv3-tiny 网络结构中，特征图大小往往多种多样。因此，如图 3-8 所示，设计了深度可进行配置的行缓存结构，能够实现对于[13，26，52，104，208，416]大小的特征图的行缓存。首先，选取网络中的最大特征图尺寸 416 作为行缓存的长度，然后，根据特征图的各种尺寸大小来设置相对应的抽头深度，连接到数据选择器 MUX 的输入端，最终，控制模块将当前层的特征图的大小传送到行缓存控制单元，行缓存控制单元再输出控制信号到 MUX 的控制端，选择所需的行缓存深度，实现了对于不同大小特征图的行缓存。

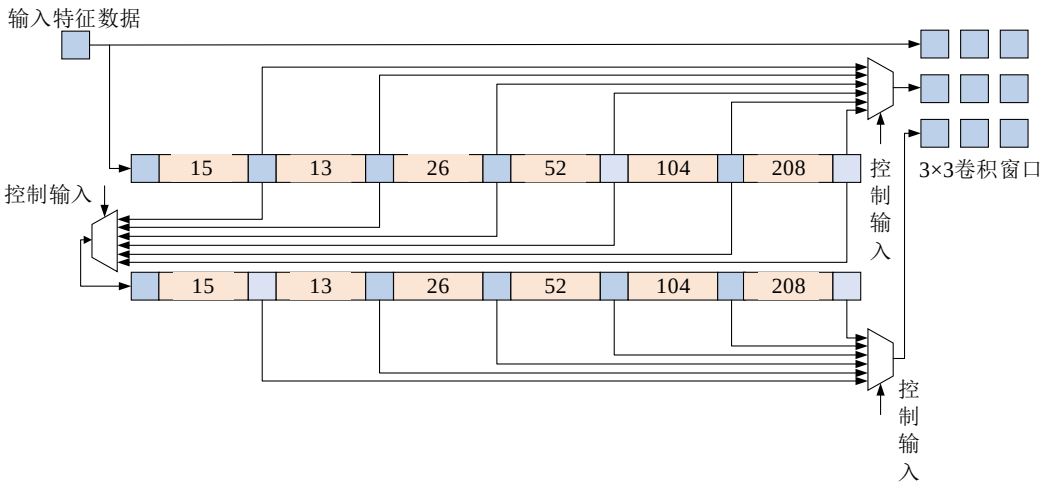


图 3-8 深度可配置的行缓存结构

3.2 卷积加速器总体架构

本文设计了一款 ARM+FPGA 软件硬件协同工作的卷积加速器用于实现卷积神经网络，如图 3-9 所示，分为 PS 与PL 两部分。首先，PL 端也就是硬件 FPGA 部分，主要包括了控制模块、片上缓存单元、计算单元(Processing Element，PE)以及 DMA。控制模块负责收到 PS 端传输来的指令，然后进行解码操作，解码后传递给除了 DMA 外的其余各模块，实现了控制除 DMA 之外的其余模块在 PL 上正常工作。片上缓存单元分为了输入缓存区与输出缓存区两部分，输入缓存区存储了待检测的图像数据以及模型的权重与偏置参数等，提供了 PE 计算所需的数据，输出缓存区则保存了 PE 计算后的结果数据。DMA 主要负责完成 PS 端的外部存储器和 PL 端的片上缓冲区之间数据交互，其中 DMA 与 DDR 之间的数据交互通过 AXI 总线来实现的，且 DMA 能够替代 CPU 实现在各系统设备与存储器之间进行高效的数据传输，所以通过 DMA 完成数据存储时的信息交互，能够显著降低资源存放占用的时间，提高系统的整体效率。AXI 总线协议广泛应用在系统中数据的连续批量传输，其主要优势在于能够充分发挥高带宽特性，即使在复杂的连接结构下，也能确保数据传输的高速性，从而在传输效率与总线系统连接复杂性之间实现有效平衡，基于以上 AXI 总线协议的这些优势，本设计采用了 AXI-Lite 协议用于 PS 端对于 DMA 和各模块之间的控制寄存器的配置，同时 AXI-Stream 协议实现了 DMA 对 PL 端的数据存储的操作。PE 主要是负责

卷积神经网络中的大部分计算任务，包括卷积、池化、量化、激活等。其次，PS端由 DDR 与 ARM 两部分中组成，外部存储器 DDR 保存了卷积神经网络的模型参数、图像数据与计算结果。ARM 端运行 C 语言程序，以及通过配置 DMA 控制了整个推理阶段。

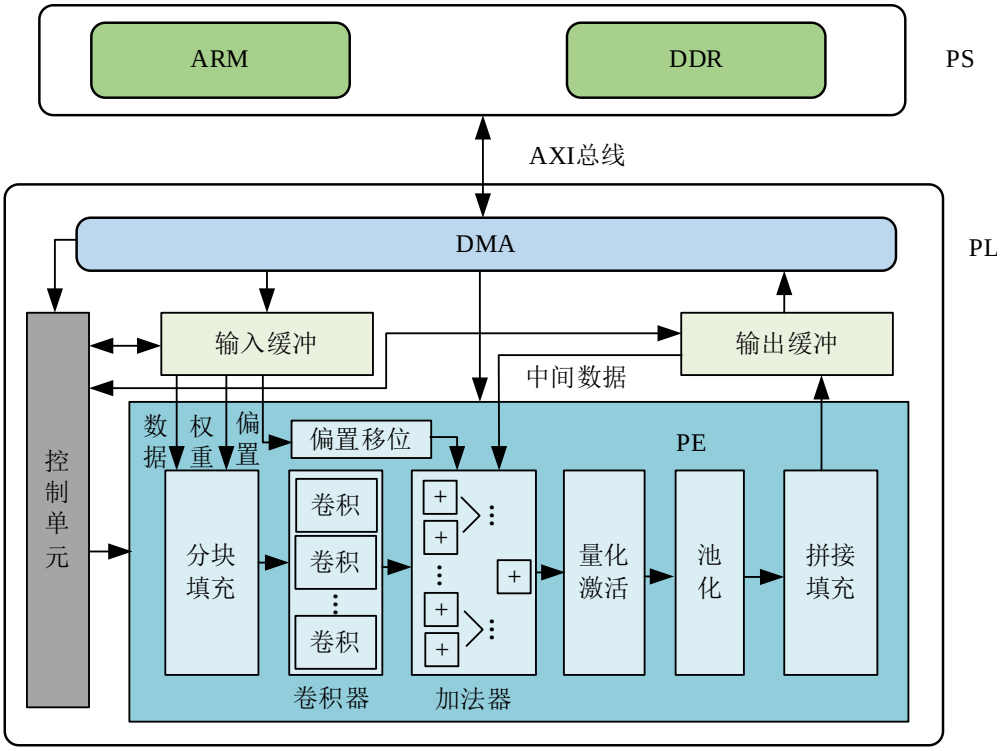


图 3-9 卷积加速器架构

采集到的图像数据在经过预处理操作过后与模型数据、控制数据都存储在 DDR 中，DMA 再通过 AXI-Stream 总线将图像数据与模型数据从 DDR 转移到输入缓存中。然后执行计算操作，首先图像数据经过分块填充操作得到模型的首层特征图，然后特征图与权重数据进行卷积计算，再通过加法器对卷积计算的结果进行求和，量化激活模块对于输入的数据流应用了量化处理与非线性激活函数，再通过池化模块将特征图的尺寸减小，所以还需进行拼接与填充操作使得计算后的输出特征图的大小与输入特征图匹配。经过上述处理后，将输出特征图保存到输出缓存当中，输出缓存中的数据可作为中间数据应用到其他 PE 的计算中，由此实现了一个 PE 的计算，不断重复这个操作，直到模型的所有层都计算完毕，最终将计算的结果利用 DMA 传输到 ARM 中。

3.3 计算模块设计

如图 3-10 展示了本设计的卷积加速器的计算模块硬件架构，PE 主要是由卷积模块、池化模块、量化激活模块、偏置移位模块共同构成。其中，卷积模块包含了卷积器与加法树，卷积器负责将像素数据与权重数据进行乘法操作，加法器负责将乘法结果进行累加操作。池化模块负责对于输入进来的特征图进行不同尺寸的最大池化操作，通过 2 个比较器得到相应数据的最大值作为输出。量化激活模块负责将数据都转化为 8 位整型以及选取相应的激活函数。偏置移位模块主要根据各层的量化结果，然后输入偏置通过偏置移位实现移位操作。下面将单独介绍主要模块的硬件架构。

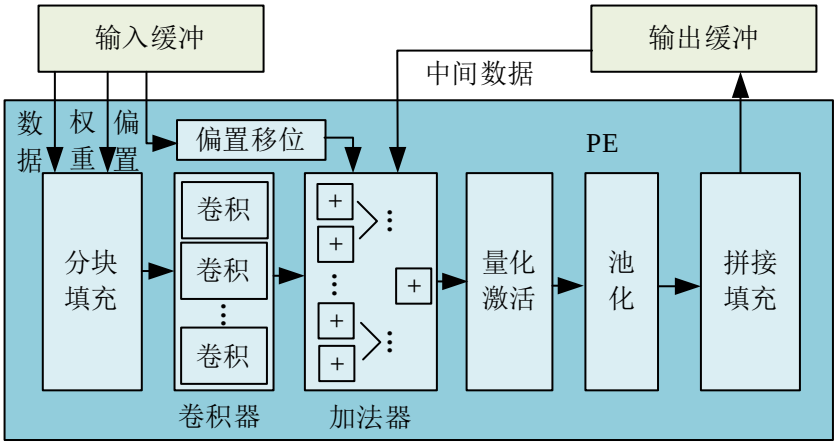


图 3-10 计算模块硬件架构

3.3.1 卷积运算模块设计

卷积运算是卷积神经网络的重点，本小节将介绍卷积运算模块的硬件结构以及进行多通道并行度分析。卷积运算模块采用了行缓存结构，如图 3-11 所示。假设输入特征图的长度大小为 L，在进行 3×3 的卷积操作时，特征图数据以串行输入的方式保存到(2L+3)个寄存器中，在每个时钟周期到来时，寄存器中的数据都会转移到下个寄存器当中，当第一个输入数据转移到第(2L+3)个寄存器时，会释放一个卷积运算窗口，且每 9 个周期会得到下个卷积运算的 9 个数据，然后输出运算窗口中的数据，并例化 9 个乘法器数据与权重缓存区窗口中的权重数据进行乘法操作，再通过加法树实现了对于乘法操作后的 9 个数据的累加以及与偏置数据进行加法运算，从而得到第一个卷积运算的结果，实现了在卷积核内的展开。

经过不断执行上述卷积运算窗口的乘加法操作，遍历整个特征图，最终将卷积运算的结果保存到输出缓存中的 RAM 中。

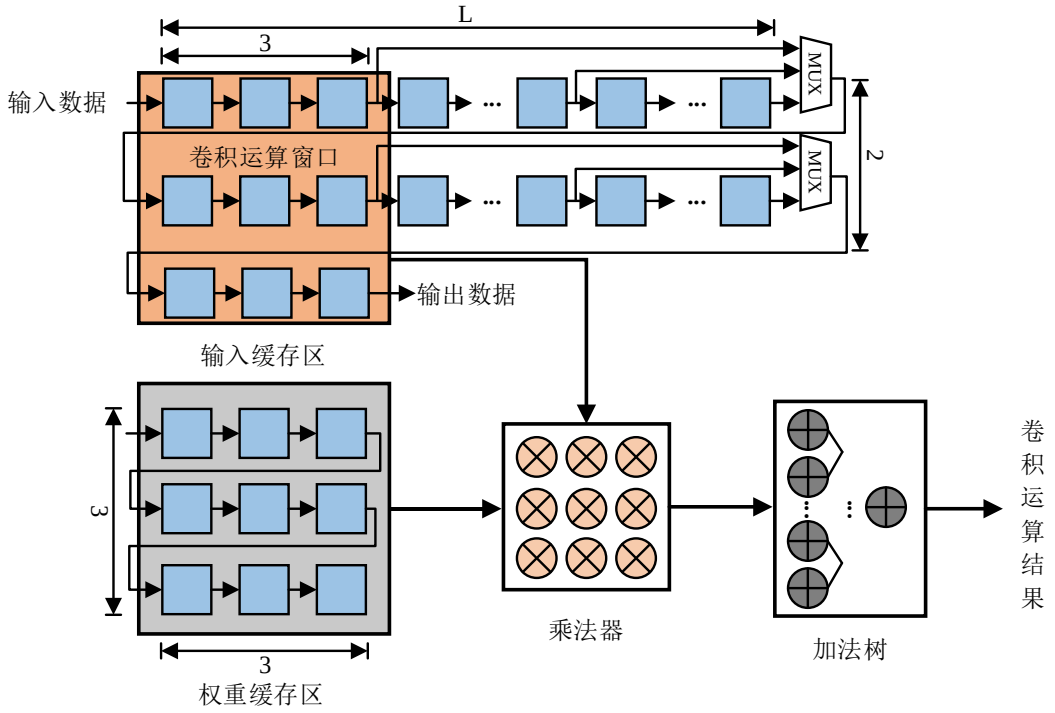


图 3-11 卷积运算硬件结构

考虑到卷积层的通道数过多，卷积运算时会消耗大量的运算时间与计算消耗，于是本文设计了多通道并行方案，综合考虑了片上资源的消耗，输入输出数据的位宽等，通过输入通道、输出通道两个维度的并行处理，实现了循环展开的任务[56]。多通道并行设计如图 3-12 所示，在进行卷积运算时，各个通道之间独立进行，互不干扰。在 3×3 卷积运算时，卷积模块在读取特征图数据和权重参数时，会通过 AXI 总线一次性传输 8 个通道共记 64bit 的数据，随后由 8 个卷积模块并行执行数据读取和卷积乘加运算。这样在一个时钟周期内，8 个输入通道的卷积计算能够同时完成。为了减少访问 DDR 的次数并节省计算时间，经过并行运算后的输出结果会暂存在输出缓存中的相应位置中。然后 8 通道的输入特征图数据会被复用，同时更新新的卷积核权重参数，再次使用新的 8 通道权重数据与 8 通道的输入特征图数据进行卷积运算，重复上述操作，令每组输入特征图数据与 8 组权重数据执行卷积运算，进而得到 8 组不同的输出特征图数据，实现“8 进 8 出”的并行策略，如图 3-13 所示。不断执行该操作，直到完成输入特征图所有通道的 3×3 尺寸的卷积运算。

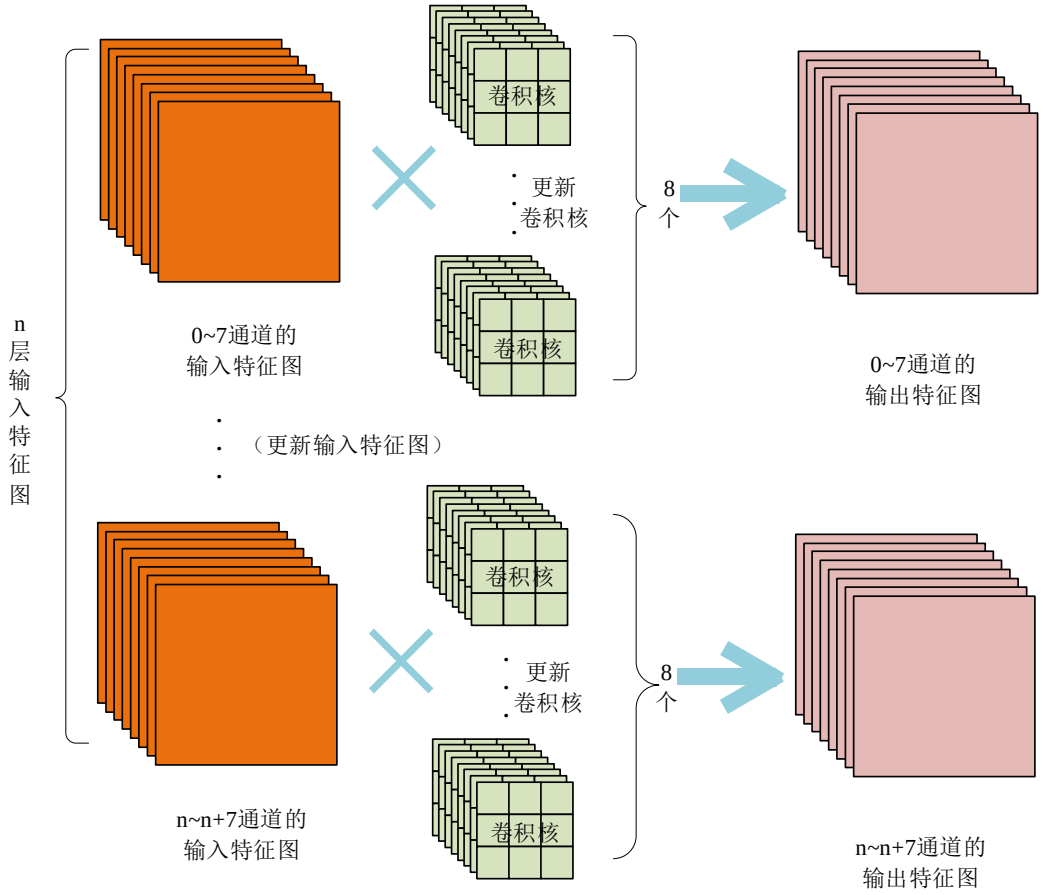


图 3-12 多通道并行设计方案

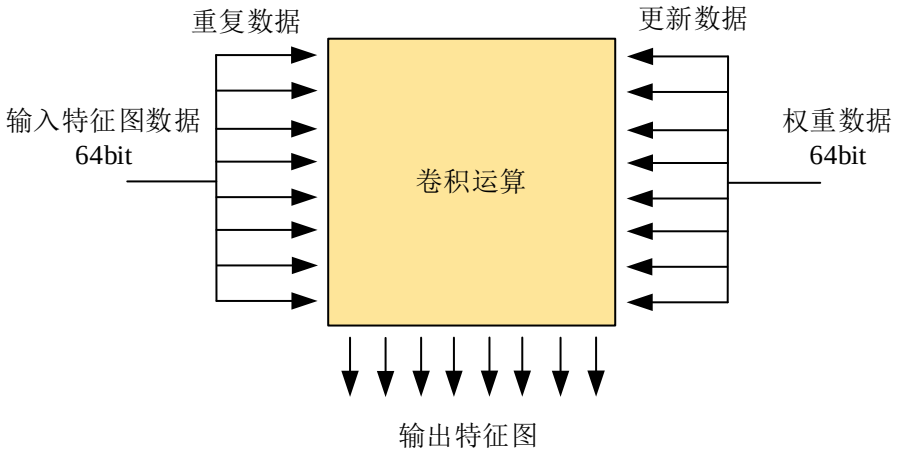


图 3-13 “8 进 8 出”并行策略

3.3.2 池化模块设计

池化操作一般分成平均池化与最大池化，主要功能是对于输入特征图进行下采样操作，减小特征图尺寸。本设计的池化模块实现了不同尺寸的最大池化操作，采用了行缓存结构，通过最大池化操作选取了池化窗口中的最大值数据，降低了

数据量，保留了图像最显著的特征信息。

以 2×2 最大池化为例，池化模型用到了 L+3 个寄存器，以及 2 个最大值比较器，L 为输入特征图的行长度，如图 3-14 所示。池化操作的流程为，首先，将第一个特征图数据保存到第一个寄存器中，然后在下个时钟周期时，将第一个寄存器中的数据转移到下个寄存器中，然后再次输入一个特征图数据保存到第一个寄存器，按照此流程不断进行，直到首个数据转移到第 L+1 个寄存器时，比较寄存器 1 就会比较第 L+1 个寄存器与第 1 个寄存器中的值，然后将最大值传到下一个寄存器当中，在下个时钟周期时，会将第一个比较结果保存到第 L+3 个寄存器，并且比较器 1 会输出新的比较结果到第 L+2 个寄存器，此时，比较器 2 就会比较两个比较器 1 输出的结果，进而得到 4 个数据中的最大值，完成了一次 2×2 最大池化操作。与直接级联结果相比节省了 1 个比较器，更加节省资源。

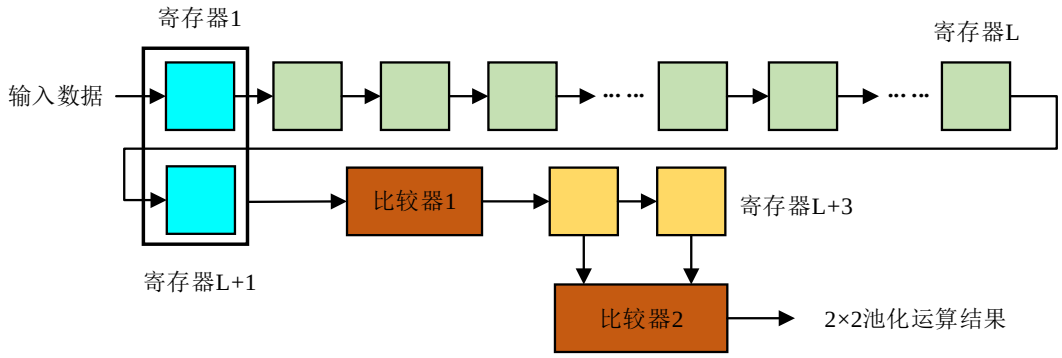


图 3-14 2×2 最大池化运算硬件结构

池化模块处理不仅能实现 2×2 最大池化操作，还能够实现空间金字塔池化模块中的 5×5、9×9 与 13×13 尺寸的最大池化操作。与 2×2 最大池化运算相比均采用了两个比较器，但是比较器每次比较的特征图数据为 5、9 与 13，且寄存器使用的数量为(4L+6)、(8L+10)与(12L+14)，池化操作流程与 2×2 最大池化相同，如图 3-15 所示，实现 5×5 最大池化运算操作。采用该硬件结构，能够实现完成一次卷积运算后，直接将卷积运算的结果输入到池化模块，进行相应的最大池化操作，无需等待一层卷积运算完全结束后再进池化操作，大大降低卷积加速器的计算延时。

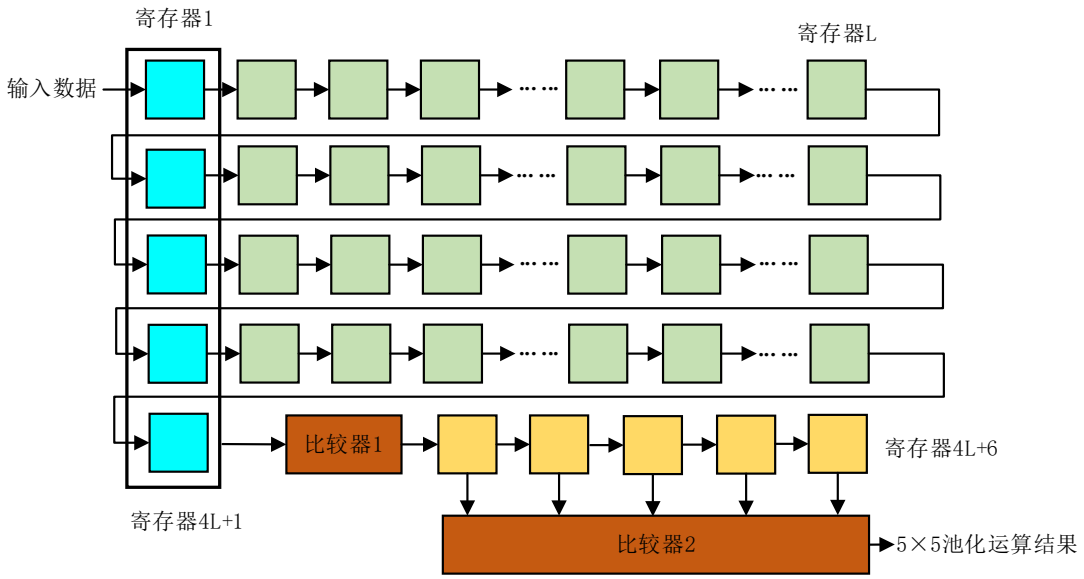


图 3-15 5×5 最大池化运算硬件结构

3.3.3 量化模块设计

在本文 2.4 节中，介绍了模型的 INT8 量化，本节将介绍量化模块的设计，即如何在 FPGA 上对于量化后的模型实现量化推理的过程。由于 INT8 类型的数据在进行卷积运算后输出的数据位宽会增大，因此需要通过量化模块将卷积模块输出的结果进行伸缩变换处理，将数据重新变为 INT8 类型的数据。首先，在进行卷积运算时，两个 INT8 类型的数据之间相乘会得到 INT16 类型的结果，然后，3×3 窗口的 9 个 INT16 类型的数据进行累加操作时，会得到 INT18 类型的结果。量化模块的作用就是将 INT18 类型的数据进行伸缩变换处理，转换为 INT8 类型的数据进行输出，如图 3-16 所示。

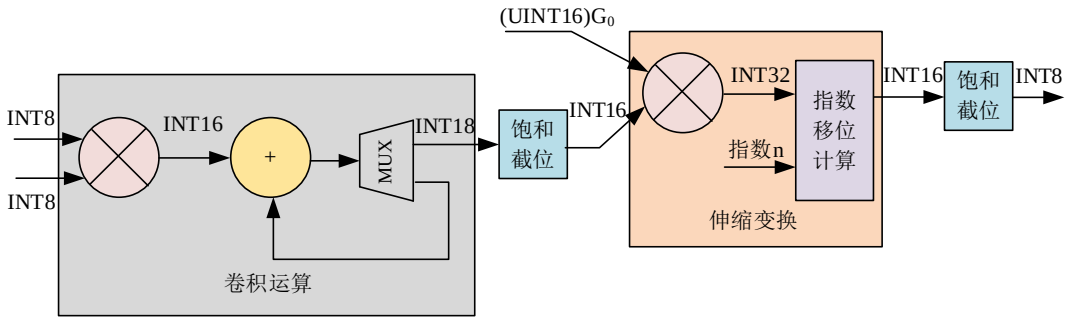


图 3-16 量化模块硬件结构

首先，经过将卷积运算后输入过来的 INT18 类型的数据进行饱和截位处理，转换为了 INT16 类型的数据，然后，在对 INT16 类型的数据进行伸缩变换，将

该数据与 UINT16 类型的 G_0 系数相乘，得到 INT32 类型的输出数据，再对于输出数据进行指数移位操作，将 INT32 数据右移 15+n 位，得到 INT16 类型的数据，最后，再通过一次饱和截位处理输出 INT8 类型的数据。上述操作中的 UINT16 类型的 G_0 系数与指数 n 是指公式(2-23)中的 G_0 与 n。

其中，INT32 的数据在进行指数移位操作时，并非是普通的右移 15+n 位处理，因为将数据进行直接右移操作会使数据向负无穷舍入，这样再进行饱和截位处理时会使计算误差不断累加，影响最终数据 INT8 量化的精度。因此，本设计采用了向最近的整数舍入右移的特殊右移处理方式，即在右移 15+n 位之前，先判断一下右移过程中要舍去的最高位，即 INT32 数据的第 15+n 位，若该位值为 1，则将右移后的结果加 1，若该位值为 0，则不改变右移结果。通过向最近的整数舍入右移的特殊右移处理方法，将移位过程从向负无穷舍入，改为了向零舍入，进而确保了数据 INT8 量化运算过程的准确性。

3.3.4 激活函数模块设计

激活函数的主要作用是引入非线性变换，使得神经网络能够学习并拟合更复杂的函数关系。在进行完卷积运算后，往往还需要进行激活函数处理，从而为后续网络层提供非线性特征。本文设计的激活函数模块实现了 Leaky ReLU 激活函数的功能，如图 3-17 所示，激活函数的表达式如式(2-5)所示，λ 取值为 0.1。在输入数据正值时，不要需要对输入数据进行处理，直接进行输出；在输入数据为负值时，需要将数据乘以 0.1 然后进行输出。在 FPGA 上，输入数据的最高位即为数据的符号位，最高位为 0 时，代表该值为正数，最高位为 1 时，代表该值为负数，因此以数据的最高位作为二选一的选择器的控制信号，当最高位为 0 时，直接选择输入作为输出，当最高位为 1 时，选择输入乘以 0.1 作为输出，其中，这里的 0.1 在硬件中以二进制形式表示。

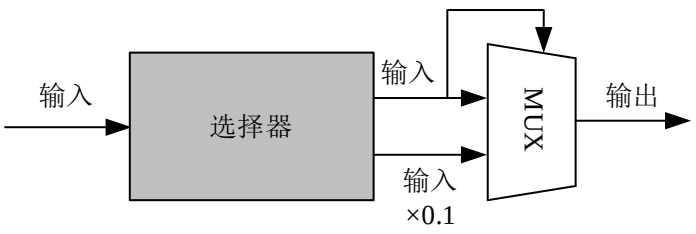


图 3-17 激活函数硬件结构

3.4 缓存模块设计

在卷积运算中，计算单元需从外部存储器中获取相关的数据。然而，由于传输带宽的限制，FPGA 的计算模块与外部存储器之间的数据传输时间通常高于计算模块与片上缓存单元之间的数据传输处理时间。为了解决这一数据传输问题，可以充分利用 FPGA 内部的存储资源，将计算需要的数据缓存在片上缓存单元中。同时，通过采用合理的数据复用方式，可以有效减少对外部带宽的依赖，从而提高系统的整体性能和数据处理效率。缓存模块设计主要包括了输入特征图缓存、卷积核权重缓存、输出缓存、外部缓存，下面将针对各个缓存模块进行介绍。

3.4.1 输入特征图缓存设计

因为 FPGA 的片上资源有限，难以将网络层的全部特征图数据缓存到片上资源上，因此采用分块加载的缓存管理策略。具体而言，先将特征图数据存入片外 DDR 上，再在片上按块加载处理。如图 3-18 所示，假设某一层的输入特征图尺寸为 W×H×C(宽、高、通道数)。根据硬件设计中支持的并行通道数与片上存储资源的容量限制，每次仅加载 h×w×n 尺寸的输入特征图，即一个较小的子块，并且输入特征图会按照通道和空间维度进行分割，将整个特征图划分为 $\frac{W\times H\times C}{w\times \mathrm{h}\times n}$ 个小块。这些小块会被逐一加载到 FPGA 的 BRAM 中进行存储，然后按照计算需求依次进行处理。

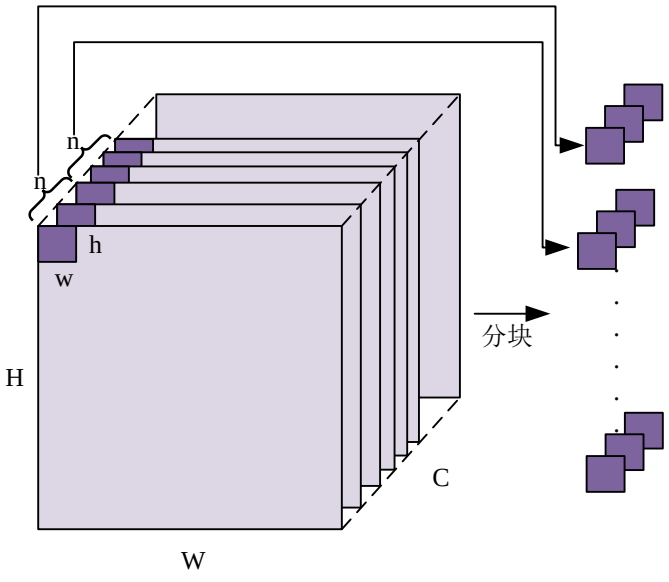


图 3-18 输入特征图分块缓存

如图 3-19 所示，展示了在片上 BRAM 上存储每一小块输入特征图的存储方式。以一个输入为 64 个通道且输入特征图大小为 4×4 为例，BRAM 的位宽为 512 比特，深度为 16。

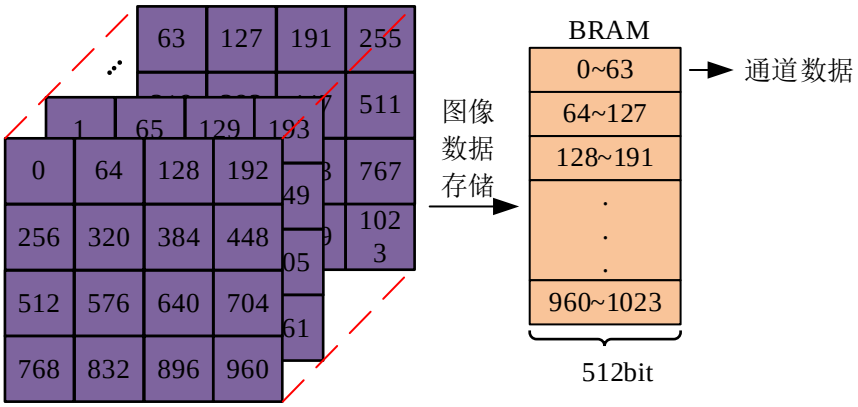


图 3-19 小块输入特征图 BRAM 存储方式

3.4.2 卷积核权重缓存设计

卷积核权重数据的存储方式要与输入特征图的存储布局及输出通道数量保持一致，以支持高效的计算操作。如图 3-20 所示，以一个 3×3 尺寸的卷积核、64 个输入通道和 64 个输出通道为例，一共使用了 64 个 BRAM 模块来存储卷积核的权重数据，其中，BRAM 的位宽为 512 比特。首先，在数据读取的过程中，每个 BRAM 模块会同时读取相同地址的数据。然后，根据 64 个输入通道和 64 个输出通道的卷积运算的需求，设计实现了一次性提取 8×64×64 个数据的方案，将这些数据直接送到计算模块进行计算处理。这种设计方法能够在一个时钟周期内提供足够的数据带宽，且权重数据的存储布局无需针对不同尺寸的卷积核来进行调整，也无需复杂的逻辑控制来适配不同尺寸的卷积运算，同时利用并行计算的特点，大幅降低了缓存过程中的复杂度，同时也保证了计算过程中的数据传输效率[57]。

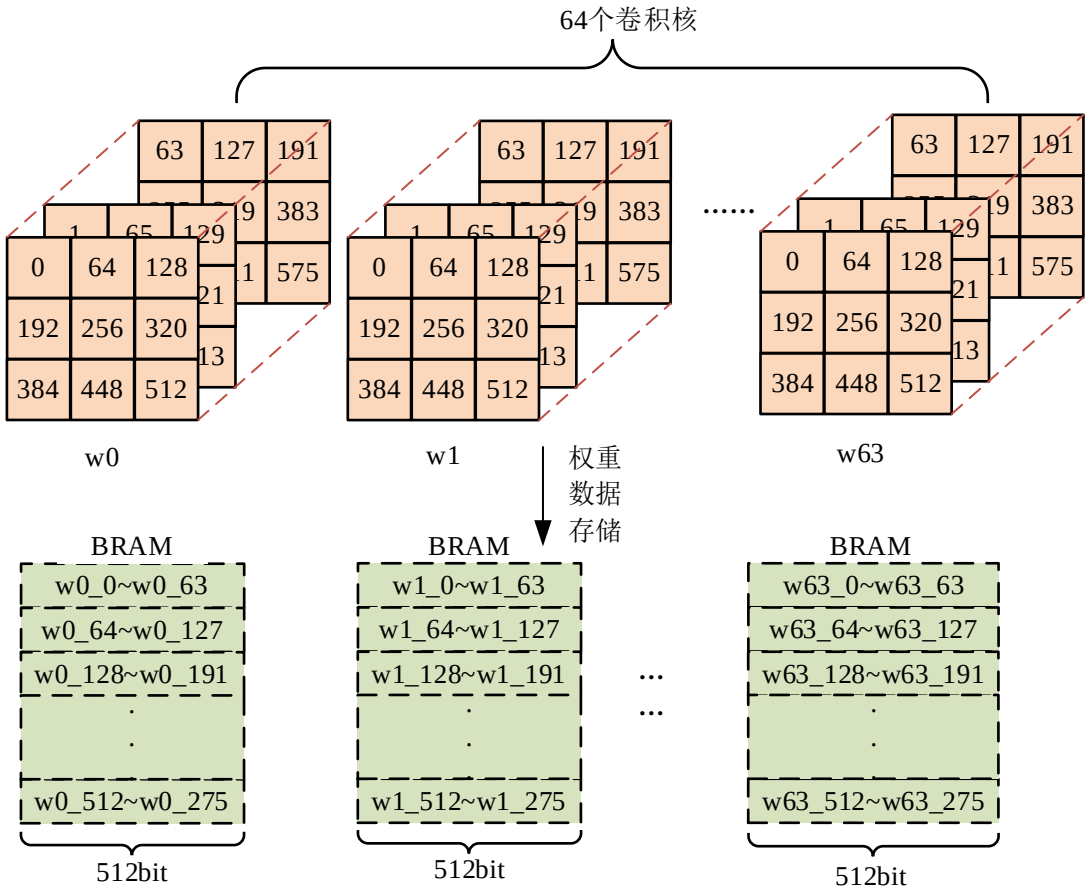


图 3-20 卷积核权重缓存设计

3.4.3 输出缓存设计

输出缓冲在卷积加速器设计中扮演着关键角色，主要用于存储中间计算结果，以支持高效的累加运算和数据传递。本文设计中采用 FIFO(First In First Out，先入先出)缓存结构来完成中间结果的存储和管理，其运行过程可以分为四种工作状态，如图 3-21 所示。状态一为等待数据有效：在计算开始时，FIFO 处于等待状态，直到接收到有效的输出数据，这一状态确保了不会缓存无效数据。状态二为存储初始输出：当第一个输出通道的计算结果生成后，将其写入 FIFO 缓存中，作为后续累加计算的初始值。状态三为累加运算：当下一个通道的计算结果生成时，FIFO 缓存会读取先前存储的数据，并将其与新生成的数据进行累加运算。累加后的结果再写回到 FIFO 中，以供后续通道计算使用。状态四为输出最终结果：随着最后一个通道的计算结果到来，FIFO 缓存会读取之前累加的中间结果，并与最后通道的数据进行最终的累加运算。累加后的结果作为当前计算块的最终输出，送至下一个处理模块。

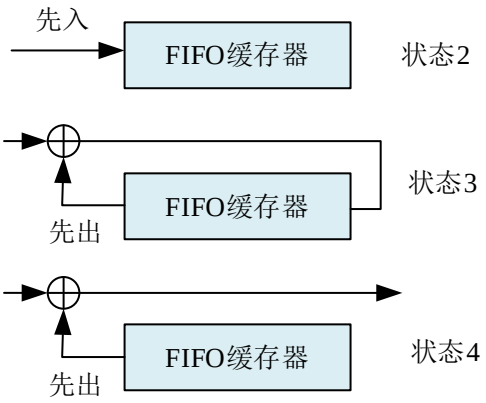


图 3-21 输出缓存设计

3.4.4 外部缓存设计

外部缓存模块的主要功能是存储整个神经网络的所有权重参数，以及摄像头采集到的特征图数据。为了满足卷积计算对数据吞吐量的高需求，需要采用一种特定的排布方式，以对权重参数进行高效存储和管理。未进行排序时，参数在内存中的排序方式并不连续，使得读取的突发长度不高，访存效率达不到设计需求。通过参数重新排序后，将每次需要访问的数据统一存放在一个存储区域中，大大提高了访问数据的效率。重排后的权重数据以连续存储的方式保存在 DDR 中，这样在实际推理过程中，可以通过流水化的方式从 DDR 中连续读取数据，避免了随机访问带来的高延迟。

参数存放的具体方案如下：如果将输出特征图分为 M 组，每组 N 份，每份进行编号且大小为 1×1×N，N 表示运算单元的通道并行度。内存排列过程时，首先是将第 1 组中的所有 w_1 权重数据保存到相应的地址空间中，接着是将第 1 组中的所有 w_2 权重数据保存到相应的地址空间中，以此类推，直到存储完第一组的数据。然后按照相同的方式存储第 2 组权重数据，直至每组数据都存放完毕，实现将每一层卷积运算所需的权重数据都依次映射到 DDR 的相同大小的存储空间。图 3-22 展示了尺寸大小为 3×3 的卷积核权重在内存中的排列方式。通过这种排列方式达到了对于权重数据高效复用的效果。

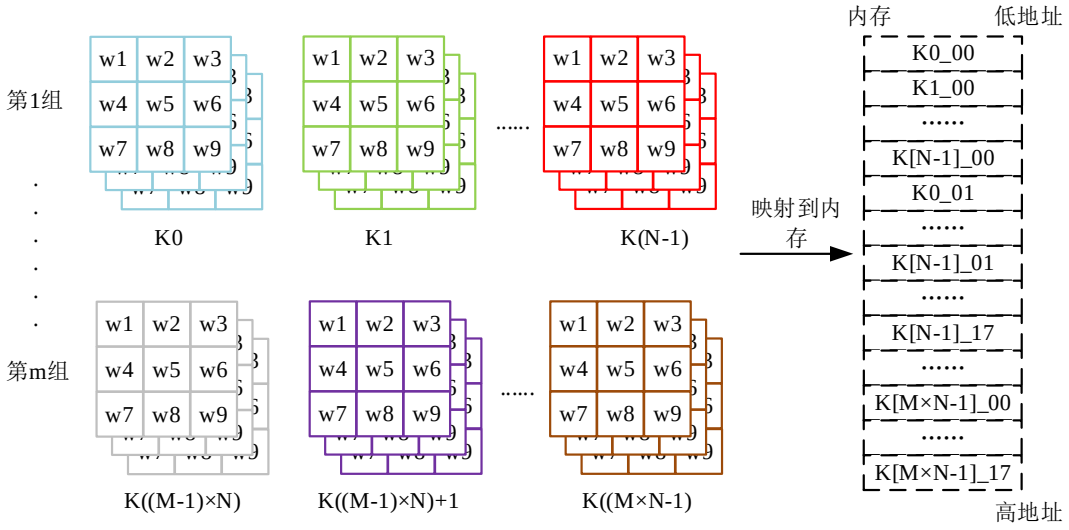


图 3-22 权重在外部存储器中的排列方式

3.5 控制模块设计

控制模块在收到 PS 端传输来的指令后进行解码操作，实现对于卷积加速器的控制与调度，进而在 FPGA 上完成了高效的卷积运算。为了减小设计难度，摈弃了通过状态机对卷积加速器的控制，通过构建 SoC 系统，利用 AXI-Lite 总线实现对于 7 个寄存器的读写操作，达到对卷积加速器的控制，下面将分别介绍这些寄存器。

第一个寄存器的每一位构成如图 3-23 所示，寄存器地址为 0x00，一共 10 位，前五位属于控制字，分别控制输入特征图、权重数据、偏置数据的接收、输出特征图的发送以及卷积运算的使能。后五位是状态字，分别表明当前卷积运算后的结果是否需要池化、是否为第一次或者最后一次卷积操作、乒乓缓存的指针指向地址以及该数据是否有效。通过配置前五位来实现五种任务的控制，并且任何一个任务完成后，都需要拉高 AP_DONE 信号触发中断，来告知 CPU 任务完成情况。

卷积加速器控制寄存器，地址：0x00

0	1	2	3	4	5	6	7	8	9	保留

0：IFM Recv Start
1：Weight Recv Start
2：Bias Recv Start
3:OFM Send Start
4:Conv Start
5:Pooling Enable
6:First Conv Flag
7:Last Conv Flag
8:PingPong Buffer Sel
9:Task Valid

图 3-23 卷积加速器寄存器 1 结构

第二个寄存器如图 3-24 所示，用于配置卷积运算的具体参数，地址为 0x04，前 16 位控制从内存中选取多少数据来执行卷积操作，中间 12 位控制特征图每行的数据量，最后三位控制当前特征图属于哪种行缓存类型。

卷积运算参数寄存器，地址：0x04

[0:15]	[16:27]	[28:30]

[0:15]:Convolution Data Length
[16:27]:Column Length
[28:30]:LineBuffer Type Select

图 3-24 卷积加速器寄存器 2 结构

第三个寄存器如图 3-25 所示，地址为 0x08，设置了量化参数中的量化伸缩比例因子：G_0 系数与指数 n。

量化参数寄存器，地址：0x08

[0:15]	[16:19]	保留

[0:15]:Scaling Factor:G_0
[16:19]:Scaling Factor:n

图 3-25 卷积加速器寄存器 3 结构

第四个寄存器如图 3-26 所示，地址为 0x0C，配置了量化参数中的零点。

量化参数寄存器2，地址：0x0C

[0:7]	[8:15]	保留

[0:7]:Input Zero Point
[8:15]:Output Zero Point

图 3-26 卷积加速器寄存器 4 结构

最后三个寄存器如图 3-27 所示，提供卷积加速器需要的地址信息。

地址寄存器，地址：0X10-1C

[0:31] Weight Read Address
[0:31] Bias Read Address
[0:31] OFM Send Address

图 3-27 卷积加速器寄存器 5、6、7 结构

板子通电后，控制模块会根据相关寄存器中的内容控制 DMA 实现对于数据的传输，选择卷积运算的计算方式、特征图的尺寸大小、量化伸缩比例因子等。当一轮计算完成后，控制模块会发送中断到 CPU 中，CPU 接收后会接着进行下一轮计算任务，然后使系统不断执行下去。

3.6 卷积加速器优化

3.6.1 乘加法优化策略

乘法与加法是卷积运算的基本运算，本节对于乘法器与加法器进行了资源优化[58]，将两个 INT8 数据的乘法运算映射到一个 DSP48E2 资源以及使用三级加法树进行加法运算。

(1)乘法运算优化

Zynq UltraScale 系列的 FPGA 拥有 DSP48E2 资源，DSP48E2 能够完成 27 位乘 18 位有符号数的乘法操作。因此 DSP48E2 在完成两个 INT8 数据的乘法后，仍有资源未被充分利用。结合 DSP48E2 的特点，本设计将两组 INT8 数据的乘法操作映射到一个 DSP48E2 资源上，实现该操作的前提是两个乘法有相同的乘数因子，例如一个特征图数据同时与两个权重数据相乘或者一个权重数据与两个特征图数据同时相乘。如图 3-28 所示，对于 A×B 与 C×B 操作，存在相同的乘数因子 B，首先，可通过将 B 进行拓展符号位，高 9 位补 0，得到的 18 位数据填充到 DSP48 的 B 端口，其次，将 A 与 C 进行拓展符号位，对 A 的低 18 位与 C 的高 18 位补 0，得到的两个 27 位数据分别填充到 DSP48 的 A 端口与 C 端口，然后，通过预加器将 A 端口与 C 端口的数据进行加法操作，再与 B 端口的数据通过 27×18 位的乘法器相乘，最后，相乘得到的 48 位结果的高位与低位分别是 A×B 与 C×B 的计算结果。实现两组 INT8 数据的乘法在一个 DSP48E2 上完成。

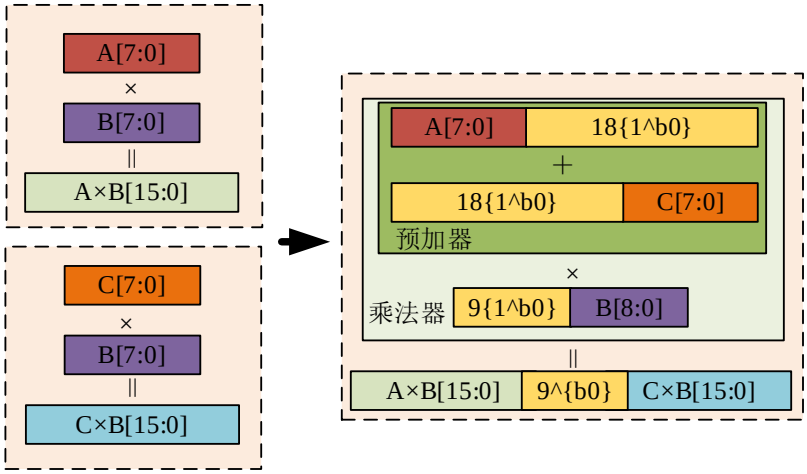
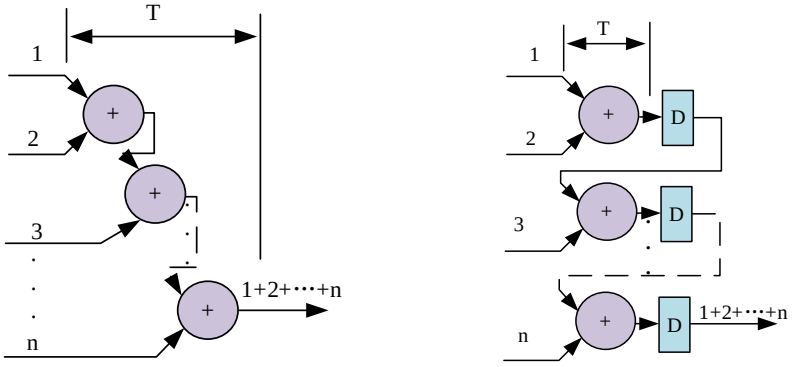


图 3-28 两组 INT8 数据的乘法映射到一个 DSP48

(2)加法运算优化

在卷积运算中，特征图数据与卷积核权重数据相乘后以及多通道特征图卷积运算后，还需要进行加法操作才能到输出结果。常见的累加硬件结构有两种：行波进位加法结构和加法器链结构。行波进位加法结构如图 3-29(a)所示，不需要寄存器，而是通过纯组合逻辑，将加法结果作为输入传递到下个加法操作，随着加法数的增加，组合逻辑路径延时 T 增加，影响处理速度。加法器链结构如图 3-29(b)所示，通过在每两级加法运算间利用寄存器实现同步，取消了组合逻辑的级联，从而使最大组合逻辑延迟 T 是单个加法运算的延时，但是会导致累加结果的输出延迟增加。资源消耗方面，行波进位加法结构主要依赖于更多的组合逻辑单元(查找表)，而加法器链结构则更多地使用触发器(寄存器)和较少的查找表资源。



(a)行波进位加法结构　　(b)加法器链结构

图 3-29 常用累加运算硬件结构

综合考虑以上两种结构后，本卷积加速器采用了二者结合的加法树结构，如图 3-30 所示。以在进行卷积运算中常见的 9 个数相加的操作为例，使用三级加

法树实现，每次处理三个输入，通过两级加法运算和两级寄存器同步。该结构的组合逻辑延时 T 是 9 级行波进位加法的 2/9，且所需的触发器和查找表资源为加法器链的 4/9，性能满足本系统的设计要求。

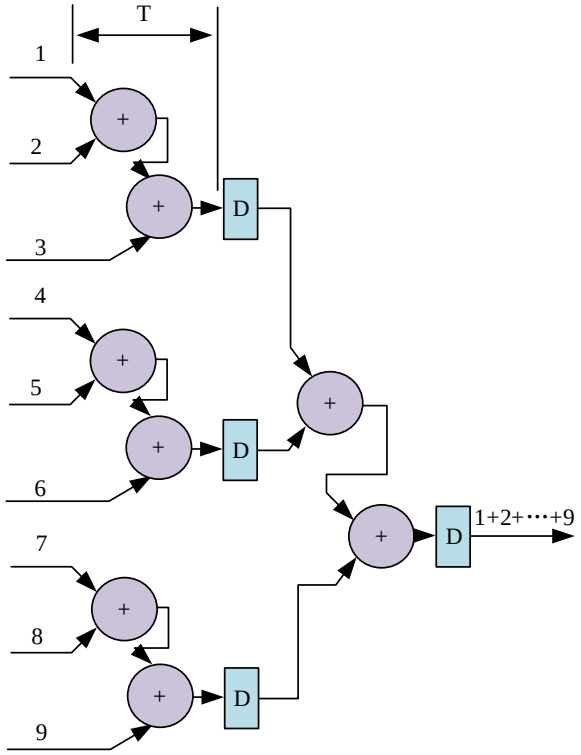


图 3-30 三级加法树结构

3.6.2 大尺寸特征图分块优化策略

特征图数据从 DDR 传输到计算模块进行卷积运算时，需要先将数据保存到片上存储中，消耗 BRAM36K 资源。以 YOLOv3-tiny 的大小 416×416×3 的输入特征图为例，当采用乒乓缓存结构时，需要消耗 240 个 BRAM 资源，往往超过了芯片内部的存储资源，因此，需要将整张特征图进行分块存储。

当前常用的分块方式如图 3-31 所示，将输入特征图按照指定的块长度 L 与块高度 H 进行分块，经过卷积运算后得到输出特征图上相应区域的结果。该方法适用于行缓存大小固定的情况，且分块前仍需要对于特征图块进行填充操作，相邻特征图块的地址并不连续。这种分块策略需要不连续寻址和复杂的填充操作，难以在 FPGA 上实现。

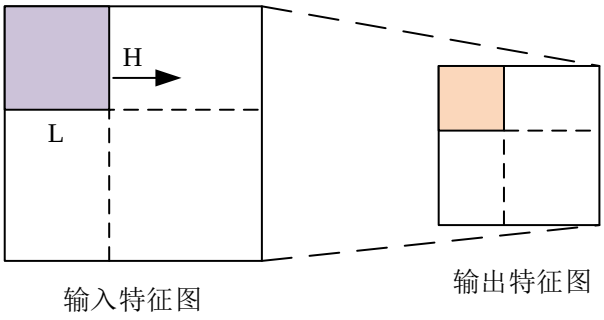


图 3-31 特征图行列分块策略

基于本加速器设计了深度可配置的行缓存结构，本文采用了如图 3-32 所示的分块策略。通过矩形分块的方式实现了特征图的分块，将大尺寸特征图按照指定的行高度 H 进行分块。这种分块策略使得分块后的特征图地址是连续的，且在进行填充时，填充的数据就在该特征图块的上个特征图块的最后一行以及下个特征图块的第一行，无需进行复杂寻址操作，相较于上一种方法，更容易在 FPGA 上实现。

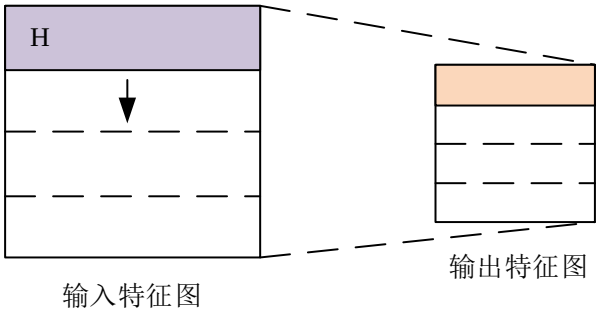


图 3-32 特征图行分块策略

3.7 卷积加速器仿真测试

如图 3-33 所示，通过 AXI 总线依次传输卷积运算所需的权重、偏置与图像数据。在传输第一帧图像时，image 寄存器等于 0，第二帧时，image 寄存器等于 1，以此类推，当开始传输第二帧数据时，卷积运算开始进行。其中，特征图数据从 222120121110020100 开始输出，并保存到 RAM 中，再从权重缓存区中得到权重数据。AXI 总线位宽为 64bit，每个周期能读取 8 个特征图或权重数据（8×8bit），执行这 8 个数据的乘累加操作，且 8 个偏置数据的值均为 1。如图 3-34 所示，第一个卷积窗口计算得到的数据为 32’h38a，8 个通道相加后结果为 32’h1c50，该结果超过了 INT8 的数据类型，再进行量化操作，量化因子 G_0 与 n

的值设置为 007f 和 0。quant_result 表示量化操作后的输出结果。其中，relu_out 表示经过 Leaky ReLU 激活函数后的输出结果。如图 3-35 所示，分别表示了 2×2 池化窗口的输入数据与池化后的输出结果，在 pool_enable 为 1 时，数据执行 2×2 池化运算，运算的结果保存到 RAM 中。如图 3-36 所示，先配置 send_enable 的值，该参数为数据发送使能标志位，当其值位 1 时，代表 AXI Stream Master 发送过来的数据有效，将发送来的数据保存到 slv_data 中，即卷积运算的最终结果。其中，slv_data[7:0]表示第一个输出通道的数据，slv_data[15:8]表示第二个输出通道的数据，以此类推。其中，第一个数据为 0，是因为在卷积运算时数据在 Leaky ReLU 激活函数存在一个周期的延时，而 valid 和 addr 不存在延迟，故使存储的数据有一个周期的延迟。综上，实际第一个输出数据为 1c，符合实验预期。

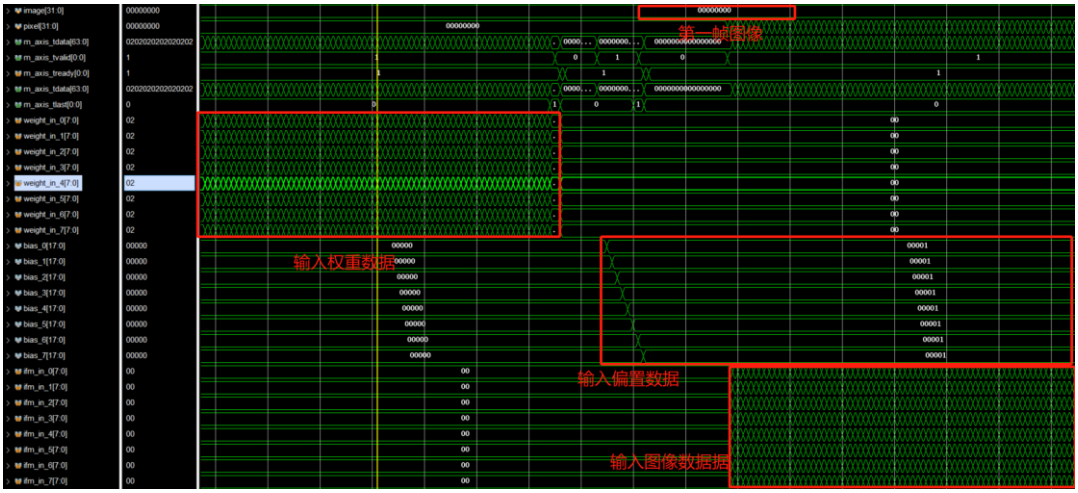


图 3-33 权重、偏置与输入图像数据的发送

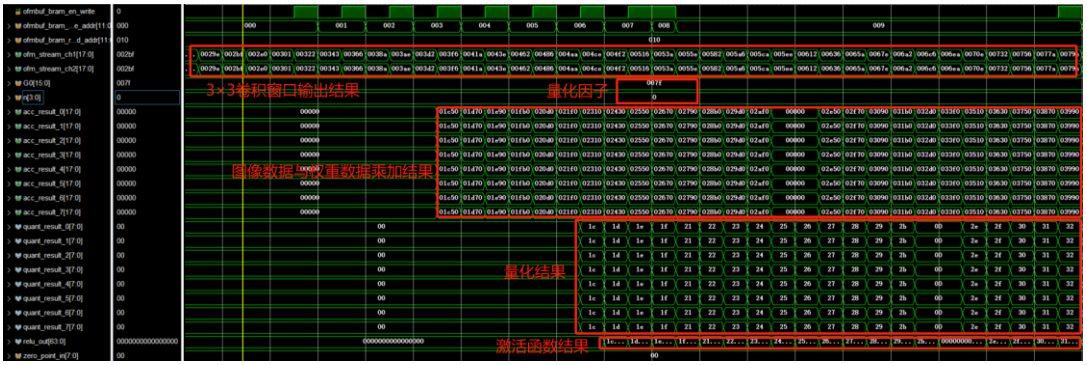


图 3-34 乘加运算、量化与激活的结果

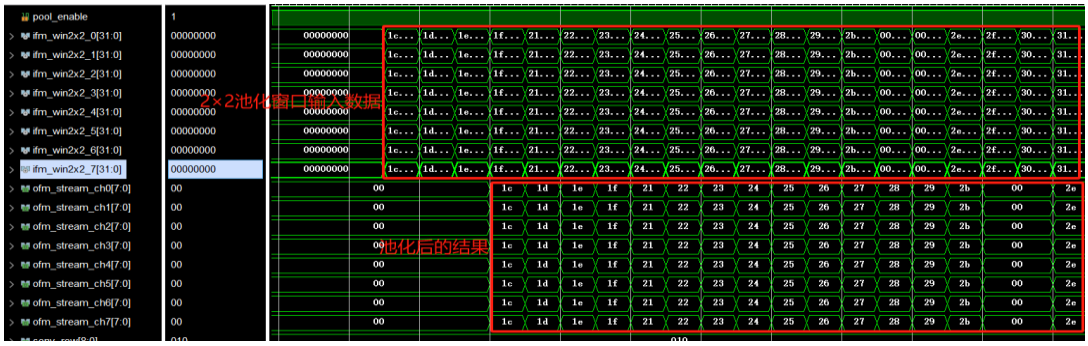


图 3-35 池化运算的结果

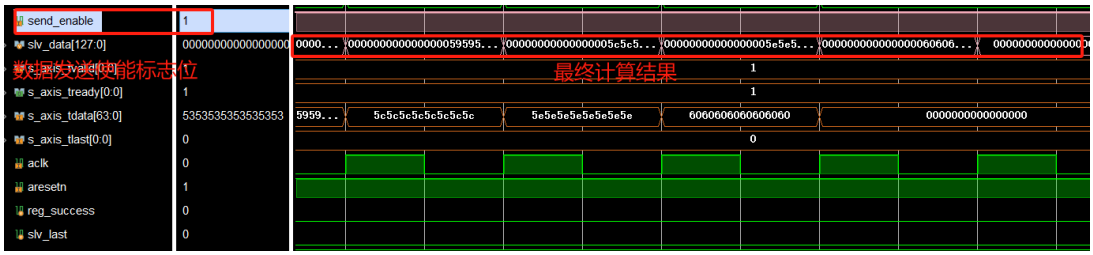


图 3-36 最终的计算结果

3.8 本章小结

本章先进行了卷积加速器方面的概念介绍，如典型的加速器架构，并行度分析以及滑窗原理与行缓存结构。然后介绍了本系统的卷积加速器的架构，并详细介绍了计算模块、缓存模块与控制模块。再对于卷积运算中的乘法和加法运算进行了优化，并提出了大尺寸特征图分块策略，实现在片上对于大尺寸特征图的分块存储。最后对于卷积加速器进行了仿真测试。

第 4 章 目标检测系统的搭建与实现

本章先介绍了目标检测系统的总体结构设计，由 PL 端与 PS 端两部分组成。然后介绍了系统器件的选型，并对 PL 端的各模块进行详细介绍，再对 C 语言实现的 PS 端进行详细介绍，通过 PL 端与 PS 端协同工作实现了本目标检测系统。最后，对本系统进行验证以及性能分析。

4.1 目标检测系统总体设计

本目标检测系统在 ZYNQ 开发板上搭建并实现，系统的总体结构设计如图 4-1 所示，由 PL 端与 PS 端两部分组成，还包括了 OV5640 摄像头、DDR4 存储器、SD 卡、串口、HDMI 显示器这些外设设备。其中 PL 端为系统的硬件部分，在 FPGA 上实现了图像采集模块、图像预处理模块、卷积加速器模块、目标标记模块与图像显示模块。PS 端为系统软件部分，以 ARM CortexA53 为核心，使用 C 语言编写程序，对于程序进行编译后在 CPU 上执行，实现了目标检测系统的初始化、SD 卡中的权重和偏置数据的加载、卷积加速器的调度与控制、聚类与非极大值抑制算法的实现等。通过系统中各部分协同工作，最终在 HDMI 显示设备上实现了实时目标检测任务，以及通过串口在上位机软件端上打印出待检测目标的位置与种类信息。其中，PL 端与 PS 端的数据交互主要是通过 AXI 总线实现的。

在外设中，OV5640 摄像头实现了图像数据的采集；DDR4 存储器保存了摄像头采集到的原始图像数据、目标检测过程所需的权重与偏置数据以及卷积运算过程的中间数据；SD 卡存储了量化后的网络模型中的权重与偏置参数，这些数据在 PS 端 ARM 处理器的控制下从 SD 卡转移到 DDR 中；通过串口实现了在上位机的串口软件界面打印系统的运行状态以及检测到的物体的位置与种类信息；目标检测的结果通过 Mini-Displayport 接口提供标准的视频显示输出，再通过 ALINX MiniDP 转 HDMI 转接器实现最终在 HDMI 显示器上进行结果图像显示。

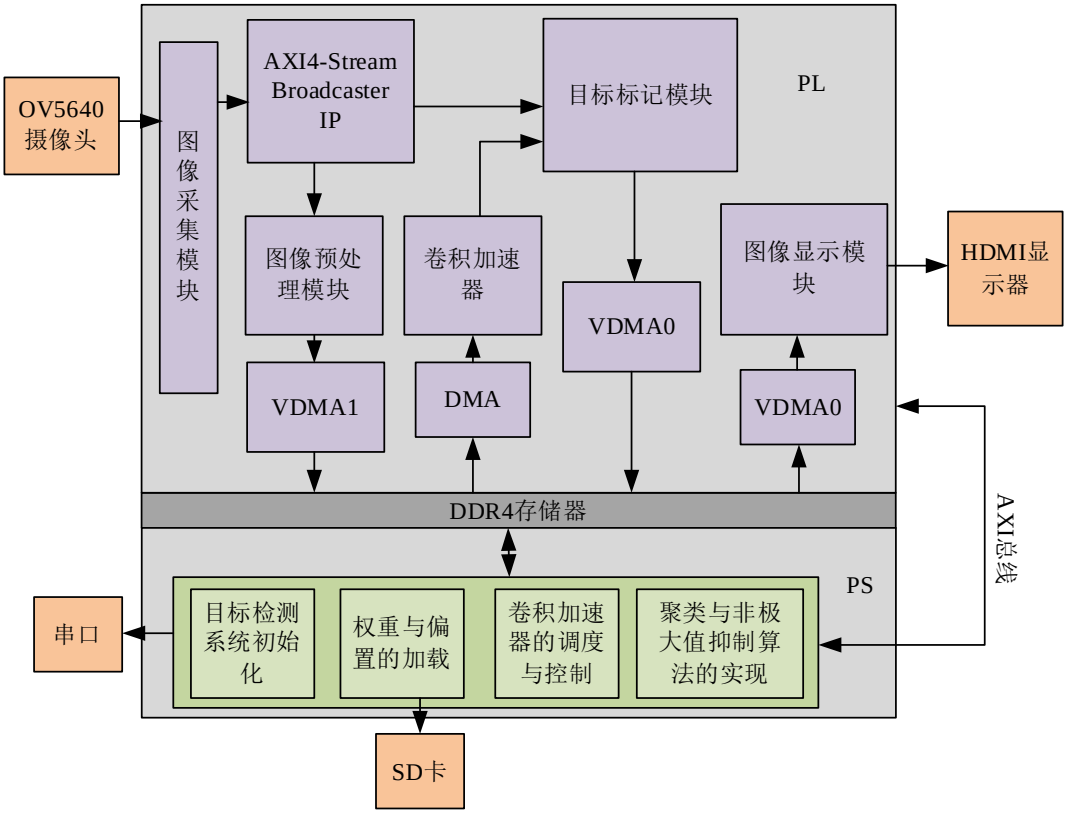


图 4-1 目标检测系统总体结构设计图

本目标检测系统首先通过 OV5640 摄像头得到了 1280×720 的原始图像数据，通过图像采集模块将摄像头采集到数据转换为 AXI4-Stream 的数据形式进行后续的图像处理；然后通过 AXI4-Stream Broadcaster IP 核将原始图像数据流复制后一分为二，分别传输到图像预处理模块与目标标记模块；一路视频流经过图像预处理模块将 1280×720 的原始图像数据进行缩放操作，得到 416×416 的图像数据，再经 VDMA1 IP 核将视频流数据缓存至 DDR 中，然后通过 AXI-DMA IP 核从 DDR 中取出预处理后的 416×416 图像数据传输到卷积加速器中，并与权重和偏置数据一起执行卷积运算操作；另一路视频流传输到目标标记模块，同时目标标记模块通过访问寄存器的值进而接收到 ARM 端输出的目标的位置与种类信息，通过目标标记模块将输入的 AXI-Stream 数据流的指定像素位置绘制指定颜色的图像框与目标种类字符；再将目标标记后的数据流结果通过 AXI-VDMA0 IP 核缓存至 DDR 中；最后通过 VDMA0 将 DDR 中的标记后的视频数据流传输到图像显示模块，图像显示模块再将数据流转化为适合传输的最小差分信号(Transition Minimized Differential Signaling，TMDS)，该信号会驱动 HDMI 接口，将目标检测的最终结果输出到 HDMI 显示器上。其中，图像预处理模块与目标标

记模块所使用的 IP 核均由 Vivado HLS 软件通过 C 语言进行设计再转化为 RTL 实现，封装成独立的 IP 核形式，然后就可以在 Xilinx FPGA 芯片的可编辑逻辑中综合并实现了。

4.2 系统器件选型

4.2.1 开发板选型

本目标检测系统选用了 Digilent 公司开发的一款独立的开发板 Genesys ZU-3EG，如图 4-2 所示，开发板搭载了 Xilinx Zynq UltraScale+MPSoC ZU3EG 主芯片，包含了一个四核 ARM Cortex-A53 处理器，适用于高性能的场景，集成了可编程逻辑和高性能处理器的异构架构。在外设方面，其板载了 4GB DDR4 内存，适合需要高带宽和可靠性的图像处理应用中；32M QSPI 闪存，可用于存储引导程序和配置数据；MicroSD 卡槽，支持了支持大容量存储扩展，可用于存储量化后的网络模型的各种参数；DisplayPort 接口，能够支持高分辨率视频输出，适用于嵌入式视觉；UART 接口，适用于串口通信与系统调试；多个 USB 接口等。Genesys ZU-3EG 开发板凭借着丰富的通信、存储、音视频、调试和用户交互功能，使其成为一个灵活、高效的开发平台，适用于从边缘计算到嵌入式视觉的各种高性能应用场景。

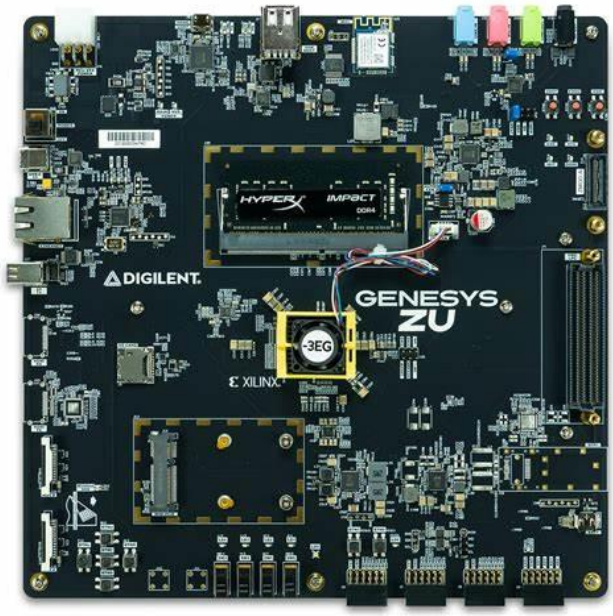


图 4-2 Genesys ZU-3EG 开发板示意图

4.2.2 摄像头选型

本系统选用了 OV5640 摄像头进行原始图像的采集工作，是一款高性能的 CMOS 图像传感器模块，内部集成了 ISP(Image Signal Processor，图像信号处理器)，支持多种图像处理功能，且能够输出高质量的图片和视频数据。摄像头采用了 DVP 接口进行图像数据的高速传输，具有较高的数据吞吐量；并通过 SCCB 接口进行配置和控制 OV5640 摄像头的寄存器，进而调节所采集到的图像数据的分辨率、视频数据格式等参数。

4.3 PL 端设计

在 PL 端的设计主要是在 Vivado 软件上进行硬件电路设计，各功能单元在 250MHz 的频率下运行，硬件部分的详细设计结构图如图 4-3 所示。其中，图像采集模块由图像采集 IP 与 Video in to AXI4 Stream IP 构成，图像显示模块由 AXI Stream to Video out IP、Video Timing Controller IP、AXI-Dynclk IP 与 DVI Transmitter IP 核构成。图像预处理模块、卷积加速器模块、目标标记模块中的 IP 核与 DVI Transmitter IP 核均为自设计的 IP 核；而 VDMA、DMA 与其余 IP 核以及均使用的 Xilinx 官方提供的标准 IP 核以降低整体开发时间。下面将分别对于各个模块进行介绍。

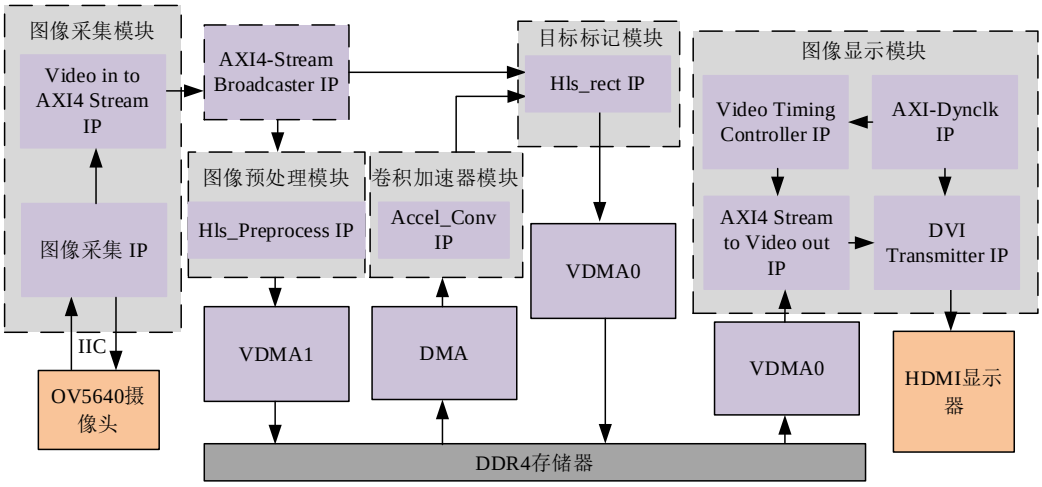
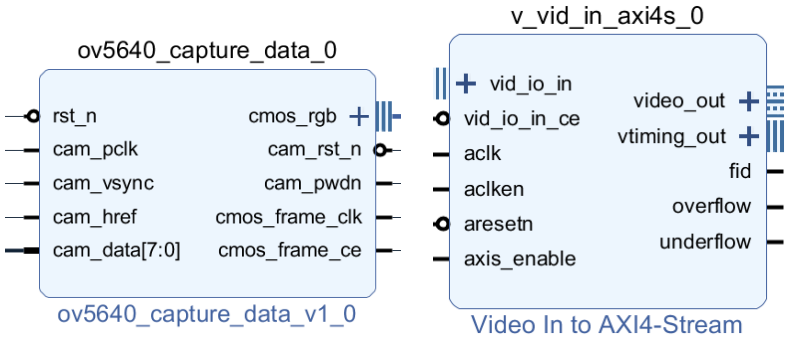


图 4-3 目标检测硬件部分设计

4.3.1 图像采集模块

图像采集模块包括了 OV5640 摄像头的采集 IP 核以及负责图像数据转换格式的 IP 核，如图 4-4 所示。



(a)图像采集 IP　　(b)Video in to AXI4 Stream IP

图 4-4 图像采集模块 IP 核

图像采集 IP 通过 IIC 总线对于 OV5640 摄像头进行初始化配置，使摄像头能够以 60FPS 的速度采集到分辨率为 1280×720 的视频数据。IIC 总线由数据线(SDA)与时钟线(SCL)组成。在通信过程中，主机设备通过操控时钟线的电平变化，以实现通信的同步。当主机有数据发送需求时，会先将要发送的数据编码成为位时间序列，随后经由数据线把该序列传送给从机。从机在接收到位时间序列之后，对其进行解码，从而获取到原始数据。

IIC 总线进行写传输时，主机先发送 START 信号，再发送 7 位的从机设备地址和 1 位操作位(写操作时为 0)的，这 8 位数据决定了主机与哪个从机进行读或写操作，然后主机会等待从机发送确认信号 ACK。接着，主机会发送要配置的摄像头寄存器的高 8 位地址，等待从机发送 ACK 应答后，再次发送摄像头寄存器的低 8 位地址，再次等待 ACK 应答。然后，主机会发送要写入摄像头寄存器的字节数据，等待接收 ACK 应答后，主机最后发送 STOP 信号，表示写传输的结束。IIC 总线写传输如图 4-5 所示。

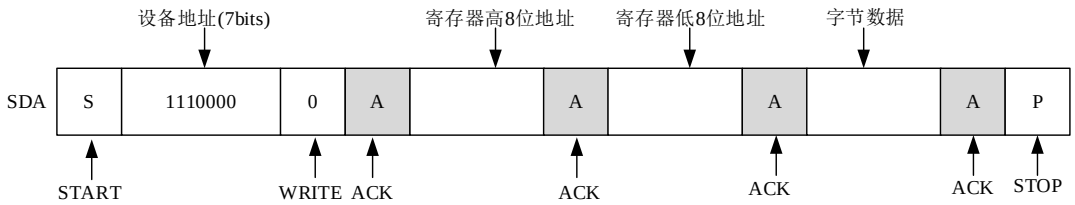


图 4-5 IIC 总线写传输图

读传输时与写传输存在略微差别，在读传输时，发送完寄存器地址后，相较于写传输，多了一个 STOP 状态与发送设置地址，且第二个设备地址的第 8 位为 1。IIC 总线读传输如图 4-6 所示。

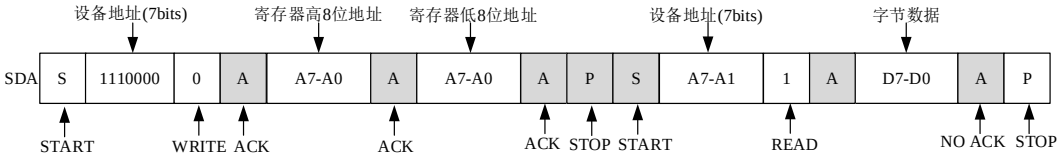


图 4-6 IIC 总线读传输图

OV5640 初始化配置如图 4-7 所示，OV5640_cfg 模块定义了一个 32 位的寄存器，8 位存储设备地址，16 位存储寄存器地址，8 位存储寄存器中的字节数据，通过 IIC 协议对于 OV5640 中的寄存器进行配置，完成对摄像头各项参数的选择。其中，power_ctrl 模块是上电控制模块，控制摄像头按照规定的时序逻辑上电；OV5640_data 模块将输出的 8 位数据拼接成 16 位的 RGB888 图像数据。

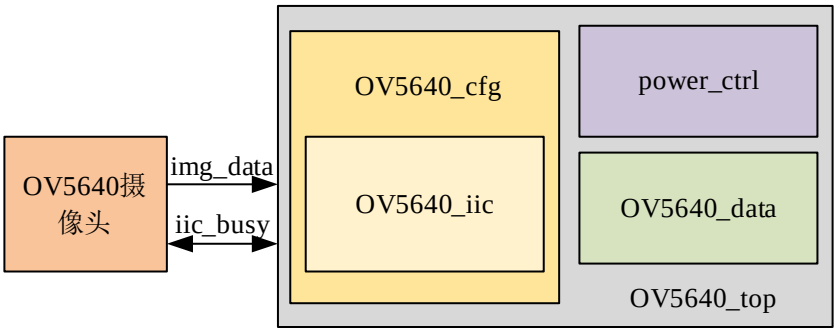


图 4-7 OV5640 初始化配置

通过图像采集 IP 核得到 RGB888 格式的图像数据后，还需要 Video in to AXI4 Stream IP 核将 RGB888 图像数据转换为 AXI-Stream 图像数据。图像采集模块在得到 AXI-Stream 图像数据后，还需经过 AXI4-Stream Broadcaster IP 核将图像数据复制后一分为二，分别进行后续处理操作。其中，通过 VDMA1 与 VDMA0 将不同的数据写入到 DDR4 存储器时，采用了多帧缓存的方式将不同的数据写入到不同的内存区域中，避免了同区域数据同时读写时出现错误。

4.3.2 图像预处理模块

图像预处理模块通过 Vivado HLS 软件自设计了 Hls_preprocess IP 核，如图 4-8 所示，实现了对输入图像的降采样与量化操作。通过降采样将输入的 1280×720 的图像进行截取缩放处理，输出 416×416 的图像，以满足 YOLOv3-tiny

算法的首层卷积层的输入特征图尺寸需求；通过量化将像素数据按照网络量化映射关系转换为可直接输入到卷积加速器执行卷积运算的数据格式。

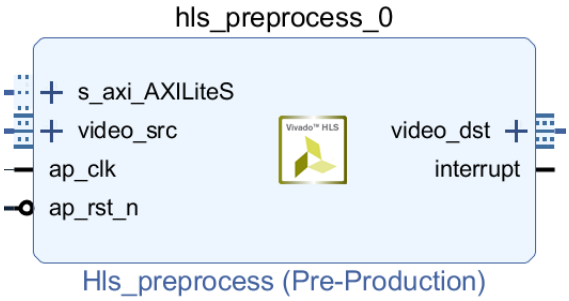


图 4-8 图像预处理 Hls_preprocess IP 核

Hls_preprocess IP 核结构如图 4-9 所示，时钟输入与数据有效输入保持同步，通过行列计数器判断当前像素的位置，像素控制器根据降采样位置映射关系生成行列控制信号，在像素位置和降采样映射位置匹配时，数据有效，并且启用量化通道 ROM。量化通过查找法实现，预先软件脚本生成的 RGB 三个通道映射表在 FPGA 上电后加载到 ROM 上。单像素的 RGB 值被拆分作为 ROM 的输入地址，根据映射表输出量化结果，三个通道的量化结果合并后与数据有效信号同步，作为新图像数据输出，输出的图像数据再经过 VDMA1 转移到 DDR4 当中。

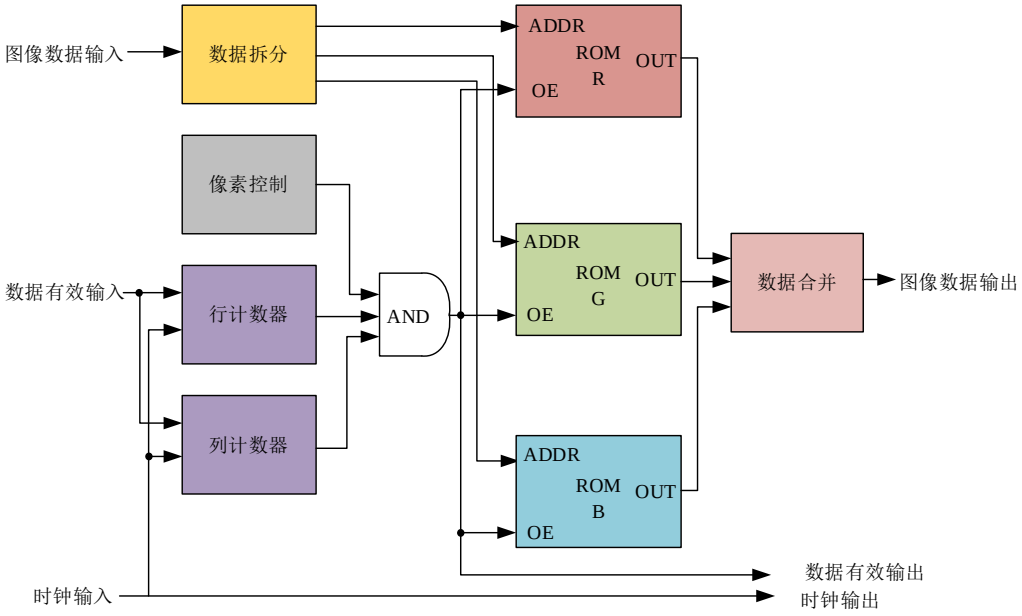


图 4-9 Hls_preprocess IP 核结构图

4.3.3 卷积加速器模块

卷积加速器模块是本目标检测系统设计的重点，因此单独在第三章进行了详

细的介绍说明，通过使用 Verilog 语言进行了卷积加速器模块的设计，并封装成 Accel_Conv IP 核，如图 4-10 所示。卷积加速器通过 DMA 从 DDR4 中获取到预处理后的图像数据后，执行相应的卷积运算操作。

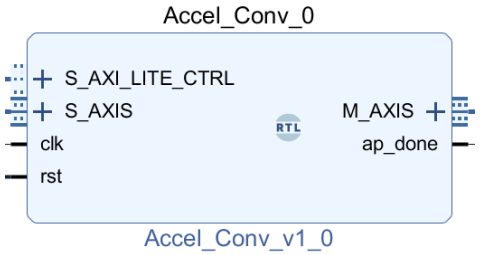


图 4-10 卷积加速器 Accel_Conv IP 核

4.3.4 目标标记模块

目标标记模块通过 Vivado HLS 软件自设计了 Hls_rect IP 核，如图 4-11 所示，实现了对摄像头采集到的图像中的目标进行实时标记的任务。卷积加速器与 ARM 端协同完成推理运算后，得到了待检测目标的位置与种类信息，然后通过目标标记模块在图像采集模块采集到的原始图像上绘制标记框与目标种类的字符，再由 VDMA0 将目标标记后的图像数据转移到 DDR4 中。

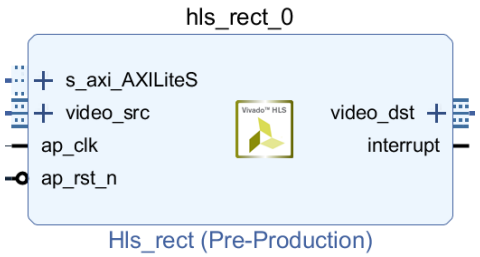


图 4-11 目标标记 Hls_rect IP 核

Hls_rect IP 核结构如图 4-12 所示，数据有效输入与时钟输入保持同步，通过行列计数器获取到当前像素的位置，CPU 再将推理得到的标记框数据传输到像素控制单元，像素控制单元的输出结果再与像素的行列数据进行与操作后，将会得到输出图像控制器的控制信号。若当前像素在目标标记框的所在位置时，则该像素位置就会变成所设置的标记框颜色；若当前像素在目标种类字符的所在位置时，则该像素位置就会以行列计数器控制的字符 ROM 的输出值进行输出；若当前像素在其他位置时，则以原始的图像数据进行输出。通过这种方式实现在输入的 AXI-Stream 数据流的指定位置绘制指定颜色的标记框与目标种类的字符的功能。

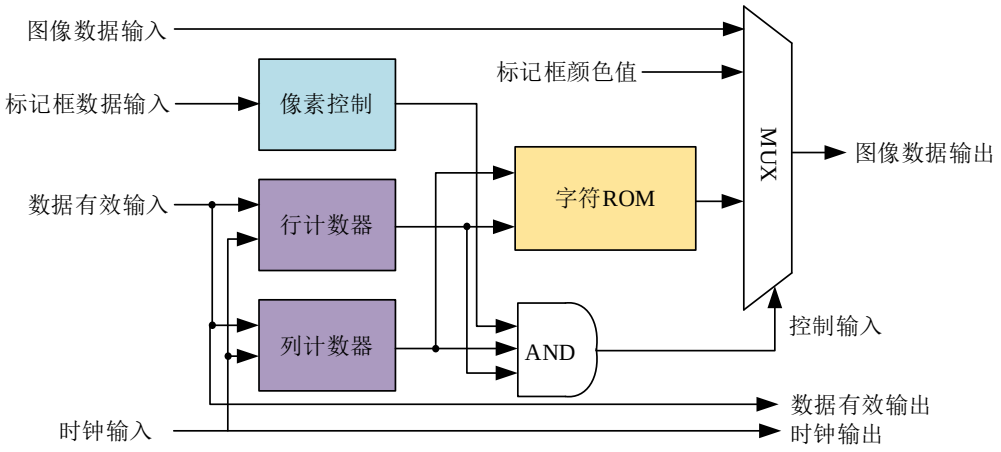
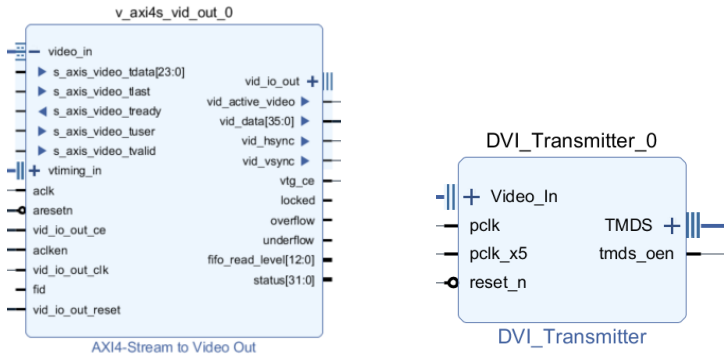


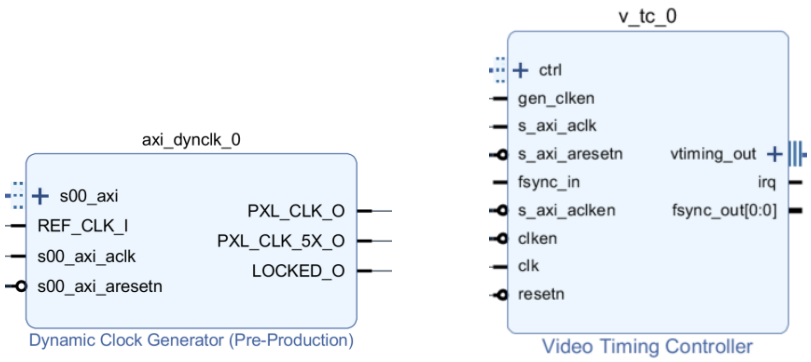
图 4-12 Hls_rect IP 核结构图

4.3.5 图像显示模块

图像显示模块包括了转换数据流格式的 AXI-Stream to Video Out IP 与 DVI Transmitter IP、调节时钟频率的 AXI-Dynclk-IP 以及控制视频输出时序的 Video Timing Controller IP，如图 4-13 所示。



(a)AXI-Stream to Video Out IP 核　(b)DVI Transmitter IP 核



(c)AXI-Dynclk IP 核　　(d)Video Timing Controller IP 核

图 4-13 图像显示模块 IP 核

其中，AXI-Dynclk IP 在目标检测系统中负责动态调整时钟频率，保证了视频输出与显示设备实现同步，进而提升了系统灵活性和显示质量。Video Timing Controller IP 的主要任务是生成和控制视频信号的时序，保证了视频数据能够按照正确的时序输出到显示设备。在进行视频显示时，通过 VDMA 将 DDR 中的视频数据流缓存出来，同时等待 Video Timing Controller IP 生成控制信号来控制 AXI-Stream to Video Out IP 将 AXI4-Stream 格式的视频数据流转化为 RGB888 格式，然后作为 HDMI 驱动的 DVI Transmitter IP 通过将 RGB888 格式的视频数据流转化为 TMDS 差分对信号输出，TMDS 驱动 HDMI 接口，进而实现目标检测后的结果在 HDMI 显示器上显示出来。TMDS 传输协议可分为编码和并串转换两个阶段。在编码时，编码器将视频数据与行同步和场同步信号分别编码为 10 位的字符流；在并串转换时，将编码后的字符流转换为串行的数据流，最终由三个差分输出通道进行输出。

在 Vivado2020.1 平台上将上述主要模块进行连接，得到了整个目标检测系统的 Block Design，连接情况如图 4-14 所示。并为各个 IP 核分配了不同的地址，以便后续 PS 端对于系统中各个 IP 核的调度与配置，硬件系统搭建完毕后，进行逻辑综合、布局布线、生成 bit 文件操作，并在后续 PS 端开发时将 bit 文件导入到 Vitis 软件中。

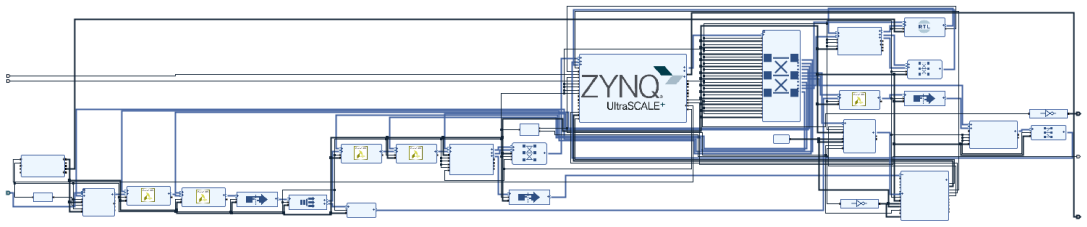


图 4-14 目标检测系统 Block Design 图

4.4 PS 端设计

硬件平台搭建完毕后，由 Vivado 软件导出 bit 文件到 Vitis 开发环境，进行 PS 端的设计。PS 端以 ARM Cortex-A53 CPU 为核心，对于 C 语言进行编译后，在 CPU 上执行程序的操作，实现了目标检测系统的初始化、SD 卡中的权重和偏置数据的加载、卷积加速器的调度与控制、聚类与非极大值抑制算法的实现。

4.4.1 系统初始化与 SD 卡数据加载

本目标检测系统在执行前需要对外设与各功能单元进行初始化操作，然后才能实现各自的功能，在初始化过程中，通过 CPU 按照指定的通信方式传递配置指令与初始化指令。由于用到了 SD 卡，因此需在板载支持包中添加 xilffs 库，实现将 SD 卡保存的模型的权重与偏置参数加载到 DDR 中，以供后续卷积加速器执行推理运算。其中，系统初始化与数据加载流程如图 4-15 所示。

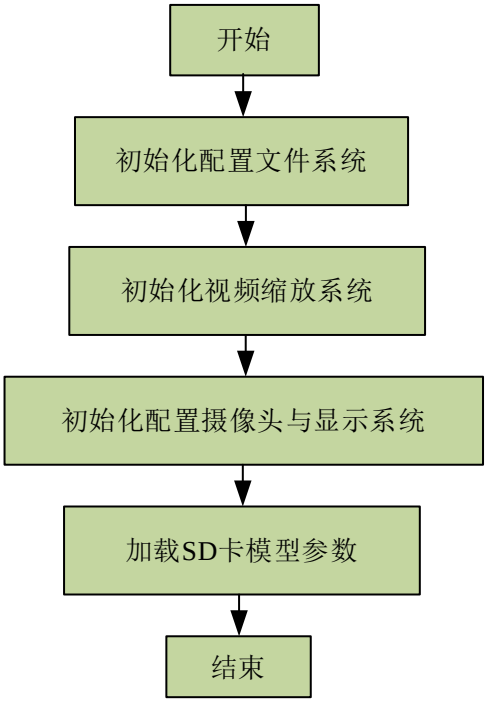


图 4-15 系统初始化与数据加载流程图

系统上电开始运行后，CPU 会先初始化读取 SD 卡文件数据的 fatfs 系统，并检查 SD 卡是否正常插入以及能否正常读取。在 SD 卡检查无误后，CPU 会进行初始化和配置视频缩放系统，使用 AXI-Lite 协议初始化和配置 Vivado HLS 软件设计的图像预处理与目标标记 IP 核，在配置寄存器中写入 IP 核的初始参数值，然后对 VDMA 进行配置以处理图像数据，设置循环模式、输入特征图缓存区的起始地址、水平偏移、水平尺寸与垂直尺寸。视频缩放系统配置完成后，CPU 开始初始化和配置摄像头与显示端口系统，配置采集的视频时序参数，视频分辨率为 1280*720、数据格式为 RGB565，经过处理后最终摄像头输出 RGB888 格式的图像数据。再对 DMA 的传输参数进行配置，将帧延迟设为 0，启用循环缓冲区模式，设置输入视频帧的水平和垂直尺寸，配置 DMA 引擎的读写通道，启动

伽马校正模块，启用 VDMA 将摄像头采集的数据写入存储器并从存储器读取数据供 DisplayPort 显示使用。

在文件系统、视频缩放系统、摄像头与显示端口系统初始化完成后，CPU 会将 SD 卡存储的模型的权重与偏置参数加载到 DDR 中，以便执行后续的卷积运算。其中，SD 卡中的数据按照脚本指定的内存排列方式进行排列，以二进制.bin 文件存储。

4.4.2 卷积加速器的调度与控制

通过卷积加速器进行推理运算是本系统的核心，其在 CPU 的调度与控制下进行。由 3.6.2 节所述，在 FPGA 上进行卷积运算时，难以将整个特征图直接存储，故采用了大尺寸特征图分块存储的策略，通过多次访问内存解决了存储资源有限的局限性。CPU 在调度与控制卷积加速器时，同样将推理运算的任务按照特征图块的划分情况进行划分，划分成多个子任务，通过对这些子任务的逐个控制最终实现了对于卷积加速器的控制。推理运算的调度与控制流程图如图 4-16 所示。

CPU 在对推理计算的逐个子任务进行控制时，系统初始化与 SD 卡数据加载阶段已经将权重与偏置数据加载到 DDR 中，原始图像的采集与缩放也已完成。在进行推理任务时，CPU 首先配置推理过程所用到的激活函数，再根据当前子任务的编号，计算得到加速器的 DMA 模块的配置参数，如输入输出特征图、权重、偏置的存储地址与长度，再将计算得到的配置参数写入到相应的寄存器当中。在完成 DMA 模块的地址与长度配置后，根据当前子任务下的特征图行列数以及分块情况，计算得到卷积模块的相应参数，并将参数写入到控制模块的寄存器。然后根据子任务的量化与池化要求，将量化与池化相应的参数也写入到控制模块的寄存器。

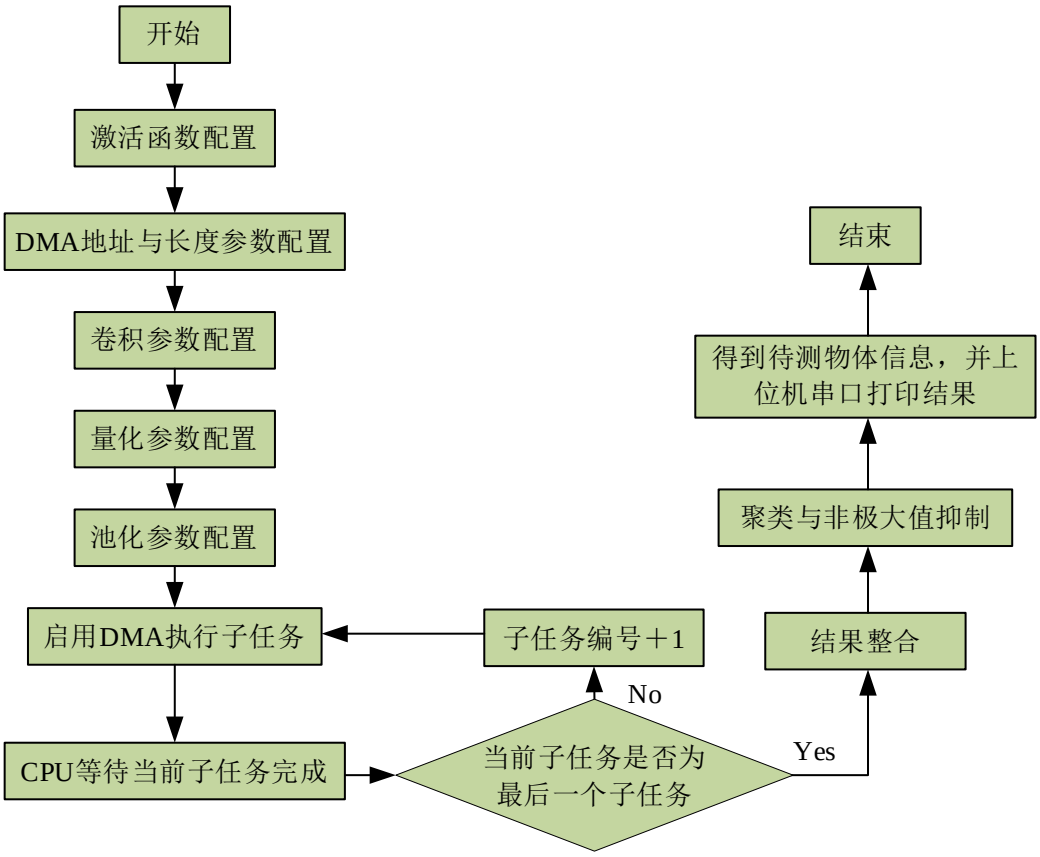


图 4-16 推理运算的调度与控制流程图

在获取到推理运算所需的配置参数后，卷积加速器便可以完成子任务的操作。启动 DMA 将卷积运算所需的数据传递到计算单元进行计算，CPU 在启动执行子任务操作后处于待命状态，待子任务完成后传递中断信号到 CPU。子任务完成后，CPU 会判断当前子任务是否为推理过程的最后一个子任务，若是则将结果进行整合，并在后续的聚类与非极大值抑制算法上进行进一步处理，最终得到待检测物体的位置信息、种类信息与概率，用于后续目标标记模块进行标记，并将结果在上位机串口软件上打印出来。若不是最后一个子任务，则将子任务编号加一，并继续完成后续子任务。

4.4.3 聚类与非极大值抑制算法

卷积加速器在执行完推理运算后每个物体往往会得到多个候选框，出现多个候选框交叉重叠预测同一物体的情况，当图像中存在多个待检测的物体时无法判断出实际存在的物体数量。因此，通过编写 C 语言程序实现了聚类与非极大值抑制的功能，程序如图 4-17 所示。

```
void clusterBoxes(box_t boxes[], int size) {
    int current_id = 0; // 初始化物体ID计数器
    for (int i = 0; i < size; i++) {
        // 如果该框框已经被分配过物体ID，则跳过
        if (boxes[i].object_id != -1) continue;

        // 为当前框框分配一个新的物体ID
        boxes[i].object_id = current_id;

        // 查找与当前框框有足够大IoU的其他框框，并将它们归为同一个物体
        for (int j = i + 1; j < size; j++) {
            // 仅考虑那些还未被分配物体ID的框框
            if (boxes[j].object_id == -1 && calcIoU(boxes[i], boxes[j]) > 0.1) {
                boxes[j].object_id = current_id;
            }
        }

        // 为下一个物体准备一个新的ID
        current_id++;
    }
}
```

(a)聚类算法

```
void applyNMSForEachObject(box_t boxes[], int size, int object_id) {
    for (int i = 0; i < size; i++) {
        // 只处理属于当前物体ID的框框，并且排除已经被抑制的框框（prob为0）
        if (boxes[i].object_id != object_id || boxes[i].prob == 0) continue;

        for (int j = i + 1; j < size; j++) {
            // 同样的条件应用于另一个框框
            if (boxes[j].object_id != object_id || boxes[j].prob == 0) continue;

            // 如果两个框框的IoU超过阈值，则进行抑制
            if (calcIoU(boxes[i], boxes[j]) > 0.1) {
                // 抑制概率较低的框框
                if (boxes[i].prob >= boxes[j].prob) boxes[j].prob = 0;
                else boxes[i].prob = 0;
            }
        }
    }
}
```

(b)非极大值抑制算法

图 4-17 聚类与非极大值抑制函数程序

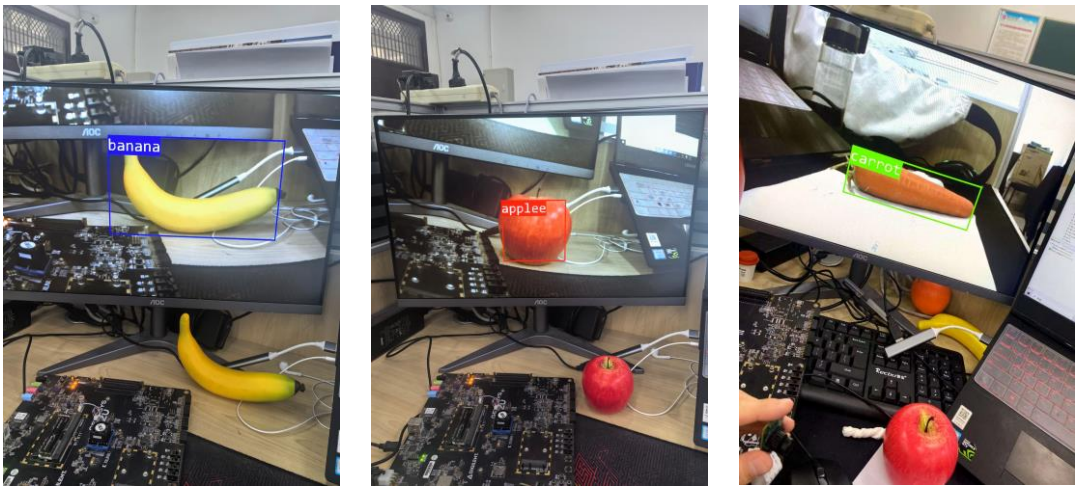
其中，实现聚类功能的函数是通过设置了一个阈值，当相邻框之间的 IOU 值大于所设阈值时，即视为这些框是属于同一物体的，将这些框归为一类，由此能够判断出图像中实际存在多少待检测的物体。将候选框进行聚类处理后，能判断出图像中有多少待测物体了，但是每个物体周围仍存在着许多交叉重叠的多余框。非极大值抑制[59]函数通过对每个物体周围的所有框进行处理，比较各个框之间的置信度大小，取其中置信度最高的候选框作为实际的待测物体框。由此实现了对于多个待测目标的检测，并将每个目标的位置与种类信息打印到串口工具上，如图 4-18 所示。

图 4-18 多个目标的检测结果打印输出

4.5 系统验证

本系统实现了在 ZYNQ 平台上搭建目标检测系统，实现实时目标检测的功能[60]。首先在 Pytorch 框架下使用改进后的 YOLOv3-tiny 模型对于 COCO 数据集中的苹果香蕉胡萝卜数据集进行训练，并经过 INT8 量化处理后得到了保存权重和偏置数据的二进制.bin 文件，将.bin 文件拷贝到 SD 卡中，后续会在 PS 端经过 CPU 处理后将参数数据保存到 DDR 中。PL 端在 Vivado 软件上搭建完硬件目标检测平台后，导出 bit 文件到 Vitis 软件，并在 Vitis 软件上进行 PS 端配置，通过 CPU 实现对于硬件端的调度与控制，并将结果通过串口工具打印出来。

目标检测系统的检测效果如图 4-19 所示，通过选用 60 张数据集中的图片进行验证本系统的检测精度与推理时间，精度达到了 75%，推理时间为 46ms，能够有效实现对于不同水果的检测，基本符合实时目标检测的要求。



(a)香蕉检测效果　　(b)苹果检测效果　　(c)胡萝卜检测效果

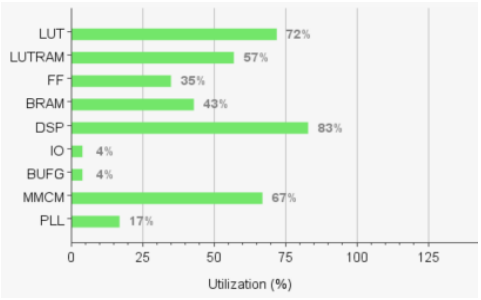
图 4-19 水果检查效果图

4.6 系统性能分析

4.6.1 资源消耗分析

本系统内部资源消耗使用情况如图 4-20 所示，LUT 与 FF 分别为逻辑功能块中的查找表与触发器；LUTRAM 是利用 LUT 实现的分布式随机存储器；BRAM 与 DSP 是 FPGA 中分别用于存储的硬件资源与执行数字信号处理的硬件单元。

其中，因为用到了较多的移位寄存器与加减法移位，所以 LUT 消耗较多，达到了 50694 个，占可用资源的 72%。合成普通寄存器与移位寄存器消耗了 49134 个 FF。乘法运算模块消耗了 300 个 DSP。缓存模块消耗了 92.5 个 BRAM，由于 DSP 数量的限制导致了 BRAM 资源利用率较低。



(a)资源利用率

Resource	Utilization	Available	Utilization %
LUT	50694	70560	71.85
LUTRAM	16425	28800	57.03
FF	49134	141120	34.82
BRAM	92.50	216	42.82
DSP	300	360	83.33
IO	11	252	4.37
BUFG	7	196	3.57
MMCM	2	3	66.67
PLL	1	6	16.67

(b)资源使用量

图 4-20 资源消耗情况

4.6.2 时序分析

本系统的时序分析报告如图 4-21 所示。建立时间的时序约束旨在保证数据信号在时钟沿到来之前保持稳定，其最坏路径时序裕量是 0.041ns；保持时间的时序约束是为了确保数据信号在时钟沿之后仍能维持稳定状态，其最坏路径时序裕量是 0.01ns；最坏脉冲宽度松弛是 0.0145ns。分析可知，系统时序情况良好，满足时序收敛条件，不存在时序违例情况，系统能够稳定运行。

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.041 ns	Worst Hold Slack (WHS):	0.010 ns	Worst Pulse Width Slack (WPWS):	0.145 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	262067	Total Number of Endpoints:	261890	Total Number of Endpoints:	65399

All user specified timing constraints are met.

图 4-21 系统时序分析报告

4.6.3 功耗分析

本系统的功耗情况如图 4-22 所示。系统的片上功耗达到了 4.457W，动态功耗为 4.123W，占总体功耗的 93%，静态功耗为 0.334W，占总体功耗的 7%。其中，PS 端由于 CPU 的工作消耗了 2.569W，占用较大的动态功耗。

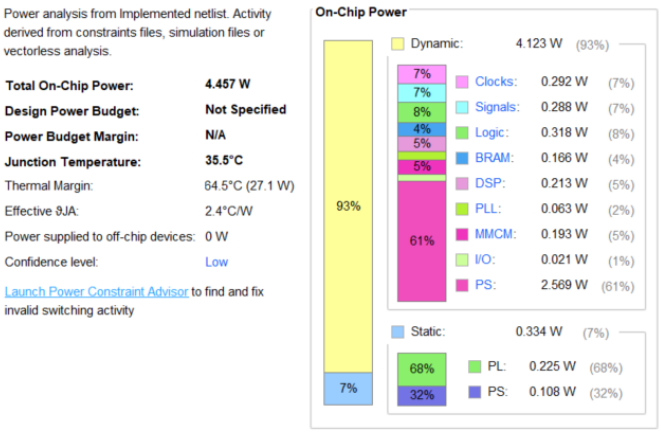


图 4-22 系统功耗情况

4.6.4 性能对比

表 4-1 给出了本文设计的卷积加速器与不同平台设计的卷积加速器在采用相同数据集的情况下的性能对比。

表 4-1 本系统与其他目标检测系统的性能对此

	文献[61]	文献[62]	文献[63]	文献[64]	本文
网络模型	YOLOv3-tiny	YOLOv3-tiny	YOLOv3-tiny	YOLOv3-tiny	改进的YOLOv3-tiny
开发平台	Ultra96 V2	ZYNQ-7100	ZYNQ-7035	Pynq-z2	ZU3EG
数据精度(bit)	8	8	16	16	8
DSP 使用量	242	548	485	32	300
功率(w)	4.26	3.153	3.71	2.8	4.46
吞吐率(GOPS)	31.50	18.24	28.99	12.25	81.23
能效比(GOPS/w)	7.40	5.78	7.81	4.38	18.21
计算密度(GOPS/DSP)	0.13	0.03	0.060	0.38	0.27
推理时间(s)	0.121	0.128	0.192	0.45	0.046
FPS	8.2	7.8	5.2	2.2	21.7

本设计的卷积加速器在多通道并行、乘加法优化、大尺寸特征图优化分块的设计下大大提高了系统的计算能力与访存效率。由表可知，本系统的吞吐率、能效比、处理速度均优于其他文献。其中，在能效比方面，本系统分别是的其他文献 2.46 倍，3.15 倍，2.33 倍 4.15 倍，在推理速度方面，本系统的速度分别是其

他文献的 2.65 倍，2.78 倍，4.17 倍与 9.8 倍。在计算密度方面本系统也有着一定优势，仅略次于文献[64]。综上所述，本系统具有高处理速度、高吞吐率与较高的计算密度等优点，能够较好的实现在资源有限的 ZYNQ 平台上进行目标检测的任务。

4.7 本章小结

本章先对本目标检测系统的总体框架设计进行了介绍，系统主要由 PS 端与 PL 端两部分组成，介绍了系统包含的外设与模块，再介绍了本设计选用的开发版与摄像头。针对 PL 端所涉及的各个模块进行了详细的介绍，分析各个模块的任务与实现过程，以及系统 Block Design 的搭建。完成 PL 端的设计后，导出 bit 文件到 PS 端，在 Vitis 软件上进行 PS 端的开发，并在 CPU 的控制下完成系统的初始化，卷积加速器的调度与控制和聚类与非极大值抑制的实现。最后，介绍了整个目标检测系统的实现过程与运行结果，并进行了资源消耗、时序与功耗的分析，最后将本系统与其他平台进行了性能对比，说明了本系统的高处理速度、高吞吐率的优势。

第 5 章 总结与展望

5.1 论文总结

本文主要对于基于深度学习的目标检测算法在 ZYNQ 平台的实现进行研究。本目标检测系统采用了 FPGA+ARM 的架构进行搭建，摄像头采集到图像数据后进行推理运算，并将检测结果在 HDMI 显示器上进行显示。在设计时对算法部分与卷积加速器部分进行了改进，提高了系统的检测效果，最后搭建了一个低延迟、高吞吐量、高计算密度的目标检测系统，能够实现对苹果、香蕉和胡萝卜的实时检测。本文的主要工作总结如下：

(1)对于国内外基于深度学习的目标检测算法与卷积神经网络加速器的研究现状进行研究与分析，选择 YOLOv3-tiny 算法作为本设计的基础算法，并分析其计算原理与网络结构。

(2)为了提高目标检测的检测效果，对于 YOLOv3-tiny 算法进行了改进。为了充分提取特征信息，提高检测精度，首先替换了损失函数，选择更能体现预测框与真实框的重合度与距离情况的 GIOU 作为损失函数；然后修改了主干网络，在主干网络末尾添加了改进后的空间金字塔，并将网络的前三个池化层替换为卷积层，进而提高了检测效果。

(3)分析了典型卷积加速器的架构与原理，在系统的 PL 端设计了一款卷积加速器，主要由计算模块、缓存模块、控制模块组成，实现了在 FPGA 上加速推理运算。并对卷积加速器进行乘加法、大尺寸特征图分块优化策略与多通道并行设计，有效降低了资源消耗，加快了处理速度且更易在 FPGA 上实现。在 Vivado 软件上实现对卷积加速器的仿真测试，验证了加速器设计的可行性。

(4)成功搭建了整个目标检测系统，在 Vivado 软件上实现了系统的 PL 端的硬件设计，主要包含了图像采集模块、图像预处理模块、卷积加速器模块、目标标记模块与图像显示模块。在 Vitis 软件上完成 PS 端的设计，进行系统的初始化，数据加载，加速器的调度控制，聚类与非极大值抑制的实现。通过 PS 端与 PL 端的协同工作，在 ZYNQ 平台上实现了整个目标检测任务，并进行了系统验

证与性能分析，分析可知，本系统具有高处理速度、高吞吐率与较高的计算密度的特点。

5.2 工作展望

本文在ZYNQ平台上搭建了目标检测系统，其能够实现对水果的实时检测，但仍有可以继续优化的地方，未来的研究方向有以下几点：

(1)本系统虽然有着较快的检测速度与较高的能效比，但是片上功耗达到了4.457W，相较于其他系统而言，本系统功率较高，后续可以探索不同的硬件设计，以降低功耗，适应那些对功耗敏感的场合。

(2)目前本系统通过在PS端实现聚类与非极大值抑制算法后能够检测到多个待测物体的位置与种类信息，但在硬件端进行目标标记时，目标标记 IP 核还不支持同时标记多个框，后续可以对目标标记 IP 核进行改进，实现在显示器上同时标记多个框的功能。

参考文献

[1] 马庆禄, 李章涵, 闫浩, 等. 基于自注意力和多点云特征融合的行车目标检测[J/OL]. 中国公路学报, 1-14 [2025-03-09].

[2] Zhao X, Wang L, Zhang Y, etc. A review of convolutional neural networks in computer vision[J]. Artificial Intelligence Review, 2024, 57(04): 99.

[3] Archana R, Jeevaraj P S E. Deep learning models for digital image processing: a review[J]. Artificial Intelligence Review, 2024, 57(01): 11.

[4] Herrmann L, Kollmannsberger S. Deep learning in computational mechanics: a review[J]. Computational Mechanics, 2024, 74(02): 281-331.

[5] 周青焱, 于苏誉, 陈浩, 等. YOLO 目标检测算法在自动驾驶中的应用[J]. 汽车知识, 2025, 25(01): 129-131.

[6] 吕雪, 王猛, 熊巍, 等. 一种智能汽车自动驾驶横向控制优化方法研究[J/OL]. 控制工程, 1-9[2025-03-09].

[7] 张亮敬, 姚国鹏, 吴作洲. 基于人脸识别的课堂考勤系统设计与实现[J]. 无线互联科技, 2024, 21(22): 23-27.

[8] 龚逸文, 阳威, 金康. 基于深度学习的行人异常行为检测[J]. 中国新技术新产品, 2024, (24): 36-39.

[9] 唐伟佳. 基于深度学习的心血管医疗影像分析与研究[D]. 北京: 北京邮电大学, 2019.

[10] Tanveer M, Rajani T, Rastogi R, etc. Comprehensive review on twin support vector machines[J]. Annals of Operations Research, 2024, 339(03): 1223-1268.

[11] Halder R K, Uddin M N, Uddin M A, etc. Enhancing K-nearest neighbor algorithm: a comprehensive review and performance analysis of modifications[J]. Journal of Big Data, 2024, 11(01): 113.

[12] LECUN Y, BENGIO Y, HINTON G. Deep learning[J]. nature, 2015, 521(7553):436-444.

[13] Zhao X, Wang L, Zhang Y, etc. A review of convolutional neural networks in computer vision[J]. Artificial Intelligence Review, 2024, 57(04): 99.

[14] Zou Z, Chen K, Shi Z, etc. Object detection in 20 years: A survey[J]. Proceedings of the IEEE, 2023, 111(03): 257-276.

[15] Denisov L, Galimberti A, Cattaneo D, etc. Design-time methodology for optimizing mixed-precision CPU architectures on FPGA[J]. Journal of Systems Architecture, 2024, 155: 103257.

[16] Ye Z, Gao W, Hu Q, etc. Deep learning workload scheduling in gpu datacenters: A survey[J]. ACM Computing Surveys, 2024, 56(06): 1-38.

[17] Konstantopoulou E, Athanasiou G, Sklavos N. Review and Analysis of FPGA and ASIC Implementations of NIST Lightweight Cryptography Finalists[J]. ACM Computing Surveys, 2025.

[18] Zhang Y, Wang H, Pan Z. An efficient CNN accelerator for pattern-compressed sparse neural networks on FPGA[J]. Neurocomputing, 2025, 611: 128700.

[19] Meloni P, Capotondi A, Deriu G, etc. NEURAghe: Exploiting CPU-FPGA synergies for efficient and flexible CNN inference acceleration on Zynq SoCs[J]. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2018, 11(03): 1-24.

[20] 山显英, 张琳, 李泽慧. 深度学习驱动下的目标检测研究进展综述[J]. 计算机工程与应用, 2025, 61(01): 24-41.

[21] 谷永立, 宗欣欣. 基于深度学习的目标检测研究综述[J]. 现代信息科技, 2022, 6(11): 76-81.

[22] 张新航, 张雅茹, 麻振华, 等. 基于深度学习方法的 YOLO 目标检测综述[J]. 长江信息通信, 2024, 37(08): 52-56.

[23] 陈憶悯, 李万益, 翁汉锐, 等. 基于深度学习的两阶段目标检测算法综述[J]. 信息与电脑(理论版), 2023, 35(14): 112-114.

[24] Girshick R, Donahue J, Darrell T, etc. Rich feature hierarchies for accurate object detection and semantic segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. Columbus, Ohio, USA: IEEE, 2014: 580-587.

[25] He K, Zhang X, Ren S, etc. Spatial pyramid pooling in deep convolutional networks for visual recognition[J]. IEEE transactions on pattern analysis and machine intelligence, 2015, 37(09): 1904-1916.

[26] Girshick R. Fast r-cnn[C]//Proceedings of the IEEE international conference on computer vision. Santiago, Chile: IEEE, 2015: 1440-1448.

[27] Ren S, He K, Girshick R, etc. Faster R-CNN: Towards real-time object detection with region proposal networks[J]. IEEE transactions on pattern analysis and

machine intelligence, 2016, 39(06): 1137-1149.

[28] Redmon J, Divvala S, Girshick R, etc. You only look once: Unified, real-time object detection[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. Las Vegas, Nevada, USA: IEEE, 2016: 779-788.

[29] Redmon J, Farhadi A. YOLO9000: better, faster, stronger[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. Honolulu, Hawaii, USA: IEEE ,2017: 7263-7271.

[30] Redmon J, Farhadi A. Yolov3: An incremental improvement[J]. arXiv preprint arXiv:1804.02767, 2018.

[31] Bochkovskiy A, Wang C Y, Liao H Y M. Yolov4: Optimal speed and accuracy of object detection[J]. arXiv preprint arXiv:2004.10934, 2020.

[32] 张立国, 孟子杰, 金梅. 针对嵌入式设备的 YOLO 目标检测算法改进方法[J]. 高技术通讯, 2024, 34(04): 356-365.

[33] Bai L, Zhao Y, Huang X. A CNN accelerator on FPGA using depthwise separable convolution[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2018, 65(10): 1415-1419.

[34] 宋亚朋. 智能目标检测系统在 FPGA 中的设计与实现[D]. 西安: 西安电子科技大学, 2022.

[35] Gholami A, Kim S, Dong Z, etc. A survey of quantization methods for efficient neural network inference[M]. Boca Raton, Florida: Chapman and Hall/CRC, 2022: 291-326.

[36] Jacob B, Kligys S, Chen B, etc. Quantization and training of neural networks for efficient integer-arithmetic-only inference[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. Salt Lake City, Utah, USA: IEEE, 2018: 2704-2713.

[37] Han S, Mao H, Dally W J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding[J]. arxiv preprint arxiv:1510.00149, 2015.

[38] Ding C, Liao S, Wang Y, etc. Circnn: accelerating and compressing deep neural networks using block-circulant weight matrices[C]//Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. Cambridge, Massachusetts, USA: ACM, 2017: 395-408.

[39] Shen Y, Ferdman M, Milder P. Escher: A CNN accelerator with flexible buffering

to minimize off-chip transfer[C]//2017 IEEE 25Th annual international symposium on field-programmable custom computing machines (FCCM). Washington, D.C., USA: IEEE, 2017: 93-100.

[40] Guo K, Sui L, Qiu J, etc. Angel-eye: A complete design flow for mapping CNN onto embedded FPGA[J]. IEEE transactions on computer-aided design of integrated circuits and systems, 2017, 37(01): 35-47.

[41] Ma Y, Cao Y, Vrudhula S, etc. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks[C]//Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey, California, USA: ACM, 2017: 45-54.

[42] 章咸斌. 基于深度学习的目标检测系统设计与实现[D]. 芜湖: 安徽工程大学, 2024.

[43] Fang W, Wang L, Ren P. Tinier-YOLO: A real-time object detection method for constrained environments[J]. Ieee Access, 2019, 8: 1935-1944.

[44] 罗凯. YOLOv3-Tiny 改进及其在口罩佩戴检测中的应用研究[D]. 西安: 西安石油大学, 2023.

[45] Tong C, Yang X, Huang Q, etc. NGIoU loss: Generalized intersection over union loss based on a new bounding box regression[J]. Applied Sciences, 2022, 12(24): 12785.

[46] He K, Zhang X, Ren S, etc. Spatial pyramid pooling in deep convolutional networks for visual recognition[J]. IEEE transactions on pattern analysis and machine intelligence, 2015, 37(09): 1904-1916.

[47] 查可豪. 基于 YOLOv5 的遥感图像目标检测方法研究[D]. 大连: 大连交通大学, 2024.

[48] 李润. 提升模型压缩推理准确率的方法研究[D]. 兰州: 兰州理工大学, 2024.

[49] Cho M, Kim Y. FPGA-based convolutional neural network accelerator with resource-optimized approximate multiply-accumulate unit[J]. Electronics, 2021, 10(22): 2859.

[50] 汤子鸣. 基于 FPGA 的边缘端深度卷积神经网络加速器设计与应用[D]. 广州：华南理工大学, 2023.

[51] 张鑫宇. 基于 ZYNQ 的目标检测硬件加速系统研究与实现[D]. 太原: 中北大学, 2022.

[52] 尹宝才，王文通，王立春. 深度学习研究综述[J]. 北京工业大学学报, 2015, 41(01): 48-59.

[53] 杨继强. 基于深度学习的瓷砖表面瑕疵检测算法研究与应用[D]. 青岛: 青岛科技大学, 2024.

[54] 徐莉娟. 面向监控视频车辆检测的 YOLOv3-tiny 算法研究与改进[D]. 西安: 西安石油大学, 2022.

[55] Fu L, Feng Y, Wu J, etc. Fast and accurate detection of kiwifruit in orchard using improved YOLOv3-tiny model[J]. Precision Agriculture, 2021, 22: 754-776.

[56] 王子明. 基于残差注意力网络的水下图像增强算法研究及硬件实现[D]. 西安: 西安理工大学, 2024.

[57] 梅其昌. 基于 FPGA 卷积神经网络加速器研究及应用[D]. 南京: 南京邮电大学, 2023.

[58] Gundrapally A, Shah Y A, Alnatsheh N, etc. A High-Performance and Ultra-Low-Power Accelerator Design for Advanced Deep Learning Algorithms on an FPGA[J]. Electronics, 2024, 13(13): 2676.

[59] Hosang J, Benenson R, Schiele B. Learning non-maximum suppression[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. Honolulu, Hawaii, USA: IEEE, 2017: 4507-4515.

[60] Babu P, Parthasarathy E. Hardware acceleration for object detection using YOLOv4 algorithm on Xilinx Zynq platform[J]. Journal of Real-Time Image Processing, 2022, 19(05): 931-940.

[61] Adiono T, Putra A, Sutisna N, etc. Low latency YOLOv3-tiny accelerator for low-cost FPGA using general matrix multiplication principle[J]. IEEE access, 2021, 9: 141890-141913.

[62] 李治林. 基于FPGA的目标检测算法加速技术研究[D]. 西安: 西安工业大学, 2024.

[63] Xiong Q, Liao C, Yang Z, etc. A method for accelerating YOLO by hybrid computing based on ARM and FPGA[C]//Proceedings of the 2021 4th International Conference on Algorithms, Computing and Artificial Intelligence. Singapore: Springer Singapore, 2021: 1-7.

[64] 高存远. 基于 FPGA 的 YOLOv3-Tiny 算法的设计[D]. 南京: 东南大学, 2020.

攻读硕士期间取得的学术成果

发表论文：

[1]刘龙生,张肖强,章咸斌,等.基于 FPGA 的高精度目标检测系统设计[J/OL].重庆工商大学学报(自然科学版),1-9[2025-03-04].

参与项目：

[1]横向项目，芜湖长信科技股份有限公司横向项目，“一种随机图形生成软件的开发”，2025.01-2026.12

致谢

时光荏苒，硕士研究生阶段的学习和研究即将告一段落。在这段求知与探索的旅程中，我不仅收获了专业知识和科研能力，更深刻体会到了学术道路上的挑战与坚持。在论文即将完成之际，我谨向所有在此过程中给予我帮助、支持与鼓励的师长、同学、家人和朋友们，表达我最诚挚的感谢。

首先，衷心感谢我的导师张肖强老师。从论文选题到研究方法，从实验设计到论文撰写，导师始终给予我悉心的指导和无私的帮助。导师渊博的学识、严谨的治学态度以及对科研的执着追求，不仅让我受益匪浅，也让我深刻理解了科学研究的真正意义。导师的耐心教诲和不断鼓励，使我在面对困难和挑战时能够坚定信心、勇往直前。

同时，我要感谢课题组的各位同学。在研究过程中，你们的讨论、建议以及提供的实验资源和技术支持，对我的研究工作起到了重要作用。正是由于你们的帮助和鼓励，使我在科研探索的道路上不断进步。在实验室的日子里，我们一起探讨问题、分享经验，彼此鼓励，共同成长，这段时光将成为我人生中宝贵的记忆。

此外，我要特别感谢我的家人。父母一直以来的理解、支持和无私的爱，是我坚持求学的最大动力。无论是在学术瓶颈期还是在生活困境中，他们的关怀与鼓励让我始终保持前行的勇气。感谢你们的默默付出，让我能够心无旁骛地完成学业。

最后，感谢所有曾经帮助和支持过我的朋友、师兄师姐以及学术界的同行们。正是你们的关心和鼓励，让我的求学之路充满温暖和力量。本论文的完成不仅是我个人努力的成果，更离不开众多人的支持与帮助。在此，我再次向所有给予我关怀和帮助的人表示最诚挚的感谢！