学校代码：10286

分类号：TP368.2

密 级：公开

UDC：621.38

学 号：200937

东南大学

SOUTHEAST UNIVERSITY

电子信息硕士学位论文

面向轻量化 YOLO 网络的 AI 加速器设计与实现

（学位论文形式：应用研究）

研究生姓名：**杨岸青**

导 师 姓 名：**徐建 副教授**

王彬 高 工

申请学位类别 **电子信息硕士** 学位授予单位 **东南大学**

领域名称 **电子信息** 论文答辩日期 **2023 年 5 月 16 日**

研究方向 **电路与系统** 学位授予日期 **2023 年 月 日**

答辩委员会主席 **汪小军** 评 阅 人

2023 年 月 日

東南大學

电子信息硕士学位论文

面向轻量化 YOLO 网络的 AI 加速器设计与实现

专 业 名 称：电子信息

研究生姓名：杨 岸 青

导 师 姓 名：徐建 副教授

王彬 高 工

Design and Implementation of AI Accelerator for Lightweight YOLO Networks

A Thesis Submitted to

Southeast University

For the Professional Degree of Master

of Electronic and Information Engineering

By

YANG Anqing

Supervised by

Asso. Prof. XU Jian

and

Senior Engineer WANG Bin

School of Information Science and Engineering

Southeast University

May 2023

摘　要

近年来，随着人们对深度学习研究的深入，人工智能已经从当初的美好畅想变成了现实，以卷积神经网络为代表的人工智能方法被应用在了包括目标检测在内的诸多领域。其中YOLO 系列网络由于检测精度高、速度快的特点成为了当今最热门的目标检测算法之一，得到广泛的应用。然而卷积神经网络结构复杂，参数量繁多，很难应用在低功耗的边缘端场合，因此对 YOLO 网络进行模型轻量化改进和硬件边缘端加速具有重要的研究意义与实用价值。本文的主要研究内容如下：

根据对卷积神经网络结构和 YOLO 系列网络架构的分析，本文提出了将 YOLOv3-tiny 的主干网络替换为 0.5 MobileNet，同时结合 LeakyReLU 激活函数和 GIOU 回归度量函数进行模型训练优化，从而得到了较高精度且轻量化的 YOLO 网络。实验结果表明，在 PASCAL VOC 数据集上，模型 mAP50 指标达到 65.9%，而参数量仅有 12.26M。

为进一步对模型轻量化，本文提出了一种新的基于相邻层权重的卷积神经网络裁剪方法对训练完成的模型进行剪枝，成功减少模型内参数中的冗余。随后进行了 BN 层融合，又将模型量化到 16bit，以减少参数并优化硬件实现需要的资源，在基本不降低精度的前提下完成了网络模型的进一步轻量化。

本文使用高层次综合技术对改进的轻量化 YOLO 网络模型设计硬件加速器，其中针对网络模型中存在的标准卷积、深度卷积和点卷积三种不同的运算分别设计了对应的硬件加速模块，可根据不同卷积实现专用的计算加速，还使用了循环分块、并行计算、双缓存模块和流水线等优化设计以加速神经网络的推理运行。

最后将设计的 AI 加速器部署在搭载 Zynq-7020 核心的 FPGA 平台上，证明其能正确加速设计的网络模型，并对本文的加速器分别进行了横向和纵向分析。本文加速器的能效比达到了 Intel i7-10750H CPU 的 28.44 倍，NVIDIA RTX2060 GPU 的 2.61 倍，加速性能优秀。

关键词： AI 加速器；目标检测；FPGA；卷积神经网络

Abstract

In recent years, with the in-depth research on deep learning, artificial intelligence has become a reality from the initial bright idea, and artificial intelligence methods represented by convolutional neural networks have been applied in many fields including target detection. Among them, YOLO series network has become one of the most popular target detection algorithms due to its high detection accuracy and fast speed, and is widely used today. However, convolutional neural networks have complex structures and a large number of parameters, which are difficult to be applied in low-power edge-end applications. Therefore, it is of great research significance and practical value to improve the model lightweighting and hardware edge-end acceleration of YOLO networks. The main research of this thesis is as follows:

Based on the analysis of the convolutional neural network structure and YOLO series network architecture, this thesis proposes to replace the backbone network of YOLOv3-tiny with 0.5 MobileNet, while combining the LeakyReLU activation function and GIOU regression metric function for model training optimization, so as to obtain a higher accuracy and lightweight YOLO network. The experimental results show that the model mAP50 metric reaches 65.9% on the PASCAL VOC dataset, while the number of parameters is only 12.26M.

To further lighten the model, a new convolutional neural network pruning method based on adjacent layer weights is proposed in this thesis to prune the completed training model and successfully reduce the redundancy in the parameters within the model. Subsequently, BN layer fusion is performed, and the model is quantized to 16 bits again to reduce the parameters and optimize the resources required for hardware implementation, which completes further lightweighting of the network model with basically no loss of accuracy.

In this thesis, high-level synthesis techniques is used to design hardware accelerator for the improved lightweight YOLO network model, in which corresponding hardware acceleration modules are designed for each of the three different operations in the network model: standard convolution, deep convolution, and point convolution, which can achieve dedicated computational acceleration according to the different convolutions, and also use optimized designs such as circular chunking, parallel computing, dual cache modules, and pipelines to accelerate The design is also optimized to accelerate the inference of neural networks.

Finally, the designed AI gas pedal is deployed on an FPGA platform with Zynq-7020 core to prove that it can correctly accelerate the designed network model, and the gas pedals in this paper are analyzed separately in cross-sectional and vertical directions. The energy efficiency ratio of the accelerator in this paper achieves 28.44 times that of the Intel i7-10750H CPU and 2.61 times that of the NVIDIA RTX2060 GPU, with excellent acceleration performance.

KEYWORDS: AI accelerator; object detection; FPGA; Convolutional neural network

目 录

图 录

表　录

第一章　绪论

本章首先介绍了本课题的研究背景及意义，然后针对卷积神经网络、卷积神经网络轻量化和 AI 加速器三个方面的研究现状进行了介绍。最后讲解了本论文所作的所做的工作、预期设计指标以及论文各章的内容安排。

1.1 课题研究背景及意义

近年来，随着科学研究和工业实践的不断演进，人工智能已经由当初的美好畅想变成了如今的一件件产品，进入我们生活的方方面面。其中以卷积神经网络为代表的深度学习算法融合了计算机，生物学，数学等多个学科的知识，成为了人工智能的重要的工具[1]。卷积神经网络相较传统的算法有着高准确率与高效率优势[2]，因此在计算机视觉[3]，图像处理[4]，自然语言处理[5]，语音识别[6]，医学诊断[7]，自动驾驶[8]等诸多场景中都得到了广泛应用。卷积神经网络(Convolutional Neural Network，CNN)是一个多层前馈神经网络，主要由卷积运算构成，并具有局部感知和权值共享的特点[9]。局部感知意味着每个神经元只连接上一层的部分神经元，因此具有局部感受野；而权值共享意味着一组神经元使用相同的连接权重，从而降低了所需权重参数的数量，降低了网络模型的复杂度。卷积神经网络对于简单几何变换处理后的图像有着非常好的适应性，被广泛用于计算机视觉领域，例如图像分类、目标检测和图像分割。

当前目标检测 AI 算法可以分为两类：单阶段目标检测和两阶段目标检测[10]。和两阶段目标检测相比，单阶段目标检测具有更高的效率。其中，YOLO 系列算法是最为知名、应用最广泛的单阶段目标检测算法。随着 YOLO 系列算法的发展，其结构变得越来越复杂，模型层数不断加深，准确率也随之提高[11]。然而，卷积神经网络规模的增大，会带来计算资源的增加，执行时间也会变长。尤其是在无人驾驶等实时应用场景中，卷积神经网络算法的实时性要求很高[12]。此外，大量参数造成的存储需求和大量计算产生的功耗也限制了卷积神经网络在智能手机、无人机等嵌入式移动设备上的广泛应用[13]。为解决这些问题，卷积神经网络轻量化技术和专用的硬件加速器成为研究的热点。

轻量化技术旨在保持可接受精度的基础上减少网络的参数量和计算量，常用的方法有神经网络压缩技术和紧密网络设计；神经网络压缩技术包括剪枝、量化和蒸馏等，目的在于去除网络模型中不重要的参数，减少计算量和内存开销。紧密网络设计旨在人为设计更为紧凑和高效的网络计算方式，在保持精度的同时尽量减少参数和计算量[14]。

现阶段卷积神经网络的硬件实现平台包括中央处理器(Central Processing Unit, CPU)、图形处理器(Graphics Processing Unit, GPU)、专用集成电路(Application Specific Integrated Circuit,

ASIC)和现场可编程门阵列(Field Programmable Gate Array, FPGA)。尽管 CPU 具有通用性，但其计算核心较少，难以提供卷积神经网络所需的大量并行计算能力，因此处理时间较长。相比之下，GPU 平台具有众多流处理单元，适合完成大量同构数据的并行计算，可以有效地实现卷积神经网络。然而，GPU 平台的功耗较高，不适用于嵌入式等对能效要求较高的设备[15]。因此，当前业界和研究趋势开始倾向于基于 FPGA 和 ASIC 硬件平台的卷积神经网络加速器。ASIC 平台具有性能高、功耗低的优点，但是其开发周期长、且灵活性不足，一旦算法改进就需要重新设计流片，成本巨大。FPGA 具有丰富的逻辑资源和存储资源，可以满足卷积神经网络的计算和存储需求，并且可以针对卷积神经网络的计算特点做出定制化的电路优化，从而缩短执行时间，加速网络计算。此外，FPGA 具有可配置特性，可以弥补 ASIC 灵活性不足的缺点，缩短硬件加速器的开发周期，并且方便未来的修改和优化[15]。

可以得出结论，庞大的推理运算量和超高的内存需求限制了 YOLO 网络在 FPGA 等嵌入式设备上的应用。为了降低网络的计算复杂度与内存需求、节约硬件功耗和成本，研究和设计面向轻量化 YOLO 网络的 AI 加速器具有重要的现实意义。

1.2 研究现状

1.2.1 卷积神经网络研究现状

卷积神经网络的研究源于仿生学。1980 年，福岛邦彦等人通过前人对猫脑视觉神经细胞的研究设计出了一个神经网络模型，以模拟视觉神经细胞的结构[16]。1998 年，LeCun 等人提出了第一个卷积神经网络 Lenet，LeNet 包含多个卷积层和池化层，可以有效地识别手写数字[18]。然后，Krizhevsky 等人在 2012 年提出了 AlexNet 网络，该网络具有更大的感受野和更深的网络层数，能够提取更复杂的特征，成功地将 ImageNet 数据集上的图像分类准确率提高了 10 个百分点以上[19]。自此之后，卷积神经网络开始向更多的权重参数和更深的网络结构发展。2014 年，牛津大学的研究员提出了 VGG-16 模型，VGG-16 的特点是所有的卷积层都采用 3x3 的卷积核，这种结构能够有效地减少参数数量，提高模型的训练速度和泛化能力。[20]。2015 年，微软研究院的 Kaiming He 等人提出了 ResNet，来解决深度卷积神经网络训练过程中的梯度消失和梯度爆炸问题，使得网络可以更加深层次地训练[21]。

当前，卷积神经网络检测算法主要分为两类：第一类是基于候选区域的两阶段目标检测模型，首先搜索边界框生成候选区域，然后使用卷积神经网络进行分类和定位。这类模型包括 R-CNN[22]、Fast R-CNN[23]、Faster R-CNN[24]和 R-FCN[25]等。第二类是基于回归的单阶段目标检测模型，它将目标物体的分类和定位过程合二为一。单阶段的目标检测模型不需要提前生成候选区域，而是直接预测物体的类别概率和所在位置的坐标。SSD 和 YOLO 系列网络是该类模型的主要代表[26]。

单阶段目标检测模型由于检测速度快、能够实现实时检测，在目标检测领域得到了广泛应用。2015 年，Joseph Redmon 等人提出了革命性的 YOLO(You Only Look Once, YOLO)目标检测算法模型[27]。该算法创新地将目标检测任务视为回归问题，把候选区域选择和检测合并为一步，大大提升了检测速度。此外，YOLO 算法采用多尺度检测机制，能够针对目标尺度多样性问题进行检测。然而，该算法在小目标检测和目标框回归精度等方面仍存在不足。为解决这些问题，Joseph Redmon 等人于 2016 年在 YOLO 网络的基础上提出了 YOLOv2 模型，引入了新的主干网络 DarkNet-19 和 Anchor Box、维度聚类、批量归一化、多尺度图像训练等方法，大幅提升了检测精度[28]。2018 年，YOLOv3 模型进一步改进，采用了残差网络结构和更好的主干网络 DarkNet-53，借鉴特征金字塔的思想，预测三种不同尺度的检测框并融合多层特征信息，解决了单阶段目标检测模型在小目标检测方面的不足[29]。2020 年，ALEXEY B 等人在 YOLOv3 基础上通过添加 CSP 结构、PAN 结构和 SPP 模块等方法，提出了全新的 Yolov4 算法，使用了新的数据增强方法 Mosaic 法和自对抗训练法，以及改进的 SAM 和 PAN、交叉小批量标准化等技术，大幅提高了检测精度[30]。同年，YOLOv5 也被提出，采用了 Mosaic 数据增强、自适应锚框计算、自适应图片缩放等策略对训练集进行预处理，主干网络中还引入了不同的 CSP Darknet 结构和 Focus 结构，在 Neck 中也添加了 CSP 结构。YOLOv5 具有和 YOLOv4 一样的快速检测速度，但检测精度相较 YOLOv4 更高[31]。

卷积神经网络算法的不断演进，然而其主流方向是朝着更高的精度发展，使得网络结构越来越复杂，模型参数量也越来越大，但是这也造成了难以部署和推理速度慢的问题。近年来网络模型在边缘端的部署也日渐开始被重视，因此根据边缘端硬件部署具体实际需求，对网络模型进行轻量化是十分必要的。

1.2.2 卷积神经网络模型轻量化研究现状

当前实现卷积神经网络模型轻量化的方法主要有：网络剪枝，量化，知识蒸馏和紧密网络设计四种方法[32]。网络剪枝主要是通过设定一个标准来判断参数的重要性，并以得到的重要性为依据对卷积神经网络中的不必要参数进行删除来达到减少参数的效果[33]；量化主要通过减少网络参数的位宽来减少模型参数量，如二值化，int8 和 int16 量化等[34]；知识蒸馏将知识从大型模型移到小型模型中，并通过训练小型模型，让其任务效果接近大型模型的效果[35]；紧密网络设计则是旨在设计新的计算方式的网络结构，达到减少模型参数且具备良好性能的目的。

网络剪枝是当前很流行的一种模型轻量化方法，它通过删除不必要的结构如神经元、通道、滤波器器或层来生成轻量级模型。网络结构要删除多少取决于删除该结构后产生的损失。与原始深度神经网络相比，轻量化模型需要更少的处理资源和存储空间。此外，在对模型进

行剪枝后，往往需要进行再次训练，从而微调模型参数以恢复精度。通过这种方法产生的轻量级网络可以以最小的精度损失达到更快的推理速度[36]。

模型量化就是通过减少数据位宽来对模型进行压缩。例如，经过计算机训练的原始模型的权重通常使用 32 位浮点数表示，我们可以将其量化到 16 位整数。类似的，还可以使用二进制或 8 位整数来代替模型中的 32 位浮点数。数据位的减少同时减少了存储和计算要求。然而，它也带来了从浮点型到整型的转换和精度降低的问题。

知识蒸馏的核心思想是将一个复杂的模型的知识转移给一个简单的模型，使得简单模型能够学习到与复杂模型相似的知识，从而达到更好的性能，复杂模型通常被称为教师模型，而简单模型通常被称为学生模型。具体来说，知识蒸馏通过在训练时使用教师模型的输出来指导学生模型的训练，从而使得学生模型能够更快、更好地学习到任务的特征和规律。

紧凑神经网络设计的思想在于通过设计更为高效的运算方式，从而降低卷积神经网络参数量和模型规模以达到轻量化的目的。近年来，代表性的轻量化神经网络包括 SqueezeNet[37]、ShuffleNet[38]和 MobileNet[39]。SqueezeNet 采用了 1×1 卷积替代 3×3 卷积、减少 3×3 卷积输入通道和下采样层后置三个策略，以完成模型的轻量化处理。ShuffleNet 采用组卷积(group convolution)有效减少了传统卷积的大量计算，但组间不能共享特征导致输出特征表达能力下降，因此提出了在组卷积后对输出特征图进行通道混洗来避免信息丢失。谷歌公司提出的 MobileNet 是一种轻量化卷积神经网络，适用于移动设备，具有体积小、计算量少的特点。MobileNet 的轻量化核心思想是用深度可分离卷积代替标准卷积。深度可分离卷积将标准卷积分解为一个深度卷积和一个点卷积。MobileNet 的参数量和计算量只有 VGG16 的 1/32 和 1/27，但分类准确性仅比 VGG16 低 0.9%，非常适合在边缘端设备上使用。

上述的卷积神经网络轻量化方法并不是互不相容的，很多时候都可以同时采用多种方法来实现更好的对网络模型轻量化，本文也将采用网络剪枝、模型量化以及紧密网络设计三种方法对模型进行优化。

1.2.3 AI 加速器研究现状

当前，各大科技公司与研究机构也都开始了对人工智能加速器的研究。比较热门的两种加速卷积神经网络方式是：基于专用的集成电路（Application Specific Integrated Circuit，ASIC）和使用现场可编程门阵列（Field Programmable Gate Array，FPGA）设计的硬件加速器[40]。

专用集成电路在特定的人工智能应用场景下一般具有很高的计算效率，ASIC的硬件结构和电路设计可以针对特定的AI应用进行优化，因此它可以在执行AI任务时获得更高的性能和吞吐量，拥有较低的能耗，并且几乎没有延迟。

2014年，中国科学院计算机技术研究所的陈云霁、陈天石团队开发了DianNao系列深度学习加速器芯片。该团队提出了一套机器学习的处理器性能建模方法，并使用该模型计算出DianNao的各项设计参数，实现了运算和访存之间的平衡，并显著提高了执行神经网络算法的效率。除了DianNao，该团队还提出了DaDianNao芯片，通过扩大计算单元和片上存储等芯片规模，提高了450倍的加速性能，并降低了150倍的平均能耗，超过了Nvidia K20M GPU[41][42]。

2016年，斯坦福大学的韩松提出了稀疏架构神经网络推理引擎EIE，该加速器通过协同设计适合深度学习的算法和硬件，使得模型推理更快更节能。首先，通过对全连接层进行剪枝和稀疏化操作来缩小模型尺寸，并使用行压缩（Compressed Sparse Row，CSR）对稀疏矩阵进行压缩，进一步减少计算量和内存占用。接着，针对压缩后的模型设计了一种定制化硬件，包括设计模型压缩的数据结构和控制流程。与DaDianNao相比，EIE获得了2.9倍的性能提升、19倍的能效提升和3倍的面积效率提高[43]。

2017年，谷歌公司推出了一种专为高效处理机器学习应用而设计的加速器，称为TPU(Tensor Processing Unit)。该加速器使用脉动阵列技术来实现高效运算，并配备了28MByte的片上缓存。TPU广泛应用于公司的数据中心中，计算速度比最新款GPU和CPU快15-30倍，能效比高30-80倍。两年后，谷歌又基于TPU推出了edge TPU，专门用于边缘端的AI加速[44][45]。

2019年，清华大学类脑计算研究中心发布了天机(Tianjic)芯片，该芯片在人工智能领域实现了新的突破。Tianjic是世界上第一款异构融合类脑芯片，采用多核架构、可重构功能核模块和混合编码方案的类数据流控制模式，不仅可以适应基于计算机科学的机器学习算法，还可以轻松实现受大脑原理启发的神经计算模型和多种编码方案[46]。

FPGA是一种可编程的硬件，可以根据具体应用的需求进行灵活的定制化设计。这使得FPGA在处理各种不同的AI任务时更加灵活，可以对算法进行优化，从而获得更好的性能。而且由于FPGA的硬件可编程性，它可以随时进行升级以适应新的技术和算法，从而保持灵活性和可扩展性[47][48]。

2015年，北京大学的Zhang等人发现前人基于FPGA的AI加速设计没有能够很好地利用硬件资源，导致计算吞吐量和内存带宽不匹配，因此无法达到硬件的最佳性能。为了解决这个问题，该团队分析了神经网络模型的计算吞吐量和内存带宽，并提出使用roofline模型来确定能获得最佳性能的设计方案。同时，对每层使用统一的循环展开因子，从而高效地处理片上数据。最终在Xilinx VC707板卡上实现了这一方案，并取得了优于以往实现的结果[49]。

2016年，麻省理工学院的Chen等人提出了一种名为Eyeriss的基于FPGA的高能效深度卷

积神经网络加速器。Eyeriss采用了分层数据流架构，共享数据以实现高效能量的计算。它还支持多层并行处理，包括输入特征映射和卷积核上的多层并行处理，以及输出特征映射上的多层并行处理。另外，Eyeriss采用了一种基于通道重排的技术，用于优化输入特征映射的内存布局。这种技术可以提高数据局部性，并减少数据移动和存储的开销。在ImageNet数据集上的实验表明，Eyeriss具有与GPU相当的精度，并且将能耗降低了2-4个数量级，达到了出色的能效。因此，Eyeriss成为一种重要的基于FPGA的深度卷积神经网络加速器，并为未来FPGA AI加速器的研究提供了重要的思路[50]。

2019 年，北京大学的 Liang 等人指出传统卷积算法在神经网络加速器上的计算性能存在局限性，并提出使用 Winograd 算法来生成输出特征图中的一列元素，利用元素间的结构相似性来减少乘法运算的数量，大幅降低了算法的复杂度。为此，他们设计了高效架构的 Winograd PE 引擎，通过并行化多个 PE 和行缓冲结构来高效复用不同瓦片的特征图数据，有效地提高了 FPGA 上的卷积神经网络加速器的性能[51]。

2019 年，为了在实时目标检测中实现高吞吐量和高功率效率，首尔大学的 Tuan 等人对 YOLO 网络的参数进行约束和量化，采用了二进制权重和灵活的低位激活。使用二进制权重可以将整个网络模型储存在 FPGA 的 block ram 中，从而大幅减少片外访问，从而提高性能。所有卷积层都被完全流水化，以提高硬件利用率，并且输入图像逐行传送到加速器。类似地，前一层的输出将逐行传输到下一层，中间数据跨层完全重用，从而消除了外部内存访问。每个卷积层都映射到一个专用的硬件块，与以往的研究相比，该设计具有最佳的能耗和效率[52]。

2021 年，中国农业大学的 Xiong 等人提出了一种软硬件协同设计方法，基于 CPU+FPGA 的异构平台，使用 HLS 作为 FPGA 开发工具，设计了卷积、最大池、上采样和 YOLO 单元，实现了流水线机制。融合卷积和 BN 层以提高模型速度，并减少 FPGA 资源消耗。为减少 FPGA 的存储资源使用，采用了跨通道数据访问模式，并对该模型进行了 16 位量化。最终部署了低资源、低功耗的轻量级神经网络 YOLOv3-tiny，为神经网络的边缘部署提供了有效的解决方案[53]。

由于 FPGA 的灵活性、低延迟、低功耗、高并行性和可编程性优势，同时考虑到 ASIC 芯片的高昂成本，本文将使用 ARM+FPGA 异构平台对神经网络加速。在设计轻量化 YOLO 网络的同时合理设计 AI 加速器硬件，达到边缘端 AI 加速的目的。

1.3 研究内容

本文为实现面向轻量化 YOLO 网络的 AI 加速器，主要完成了以下创新性研究工作：

（1）本文首先分析了 YOLO 系列目标检测算法的架构，提出了对 YOLOv3-tiny 网络的修改方法，运用紧密网络设计的思想将主干网络替换为 0.5 MobileNet，将原本的 ReLU 激活

函数和回归度量函数 IOU 分别替换为 LeakyReLU 函数和 GIOU 以提升精度，最后使用 VOC 数据集对网络进行训练。并将训练完成的网络模型与已有的一些网络模型进行对比，证明了本文提出的网络在轻量化和精度上的优势。

（2）为了进一步对卷积神经网络的轻量化，本文提出了一种新的基于相邻层权重的卷积神经网络裁剪方法对训练完成的模型进行剪枝，以减少模型内参数的冗余。随后又进行了 BN 层融合，减少参数并优化硬件实现需要的资源，最后对模型进行 16bit 量化，进一步减少了模型的参数，达成了本文提出的 YOLO 网络模型的轻量化。

（3）为了实现网络模型的硬件加速，本文使用高层次综合技术设计 AI 硬件加速器，提出了可以针对网络中存在标准卷积、点卷积和深度卷积三种不同的卷积，分别设计对应的硬件加速模块以专用计算加速，还使用了循环分块、并行运算、双缓存模块和流水线等优化设计加速运行。

（4）本文在完成了轻量化 YOLO 网络和对应的 AI 加速器设计后，在搭载 Zynq-7020 核心的 FPGA 平台上进行了测试和分析，分别与常规的 CPU、GPU 平台进行横向对比，与其它文献提出的 AI 加速器进行纵向对比，证明了本文设计的 AI 加速器对卷积神经网络加速的有效性和优越性。

1.4 设计指标

本文的具体设计要求如下：

（1）本文设计的目标检测算法模型参数量应小于 20M，mAP50 指标应不低于 60%，实现轻量化且高精度的目标检测。

（2）对本文设计的目标检测算法模型进行进一步的轻量化，在不影响精度的前提下尽可能压缩模型，最终参数量应小于 2M，且 mAP50 指标不低于 60%。

（3）使用 Zynq7020 FPGA 平台对本文设计的网络进行硬件加速，加速效果应快于常见的 Intel CPU，且功耗远低于该 CPU，以达成高能效比设计。

1.5 论文安排

本论文主要内容可分为六个章节：

第一章：绪论。本章首先介绍了论文的研究背景和研究意义，之后分别对卷积神经网络、网络轻量化和 AI 加速器的研究现状进行了介绍，最后对本文的研究内容、设计指标和组织架构进行了说明。

第二章：卷积神经网络介绍。本章首先介绍了卷积神经网络基础，对卷积神经网络基本结构进行了讲解，然后对 YOLO 系列目标检测网络从 v1 到 v3 的发展和具体网络架构进行了

详细阐述，最后介绍了本文硬件实现需要的 FPGA 芯片原理及其内部资源。

第三章：轻量化 YOLO 网络实现。本章首先对 YOLOv3-tiny 网络进行了介绍，然后以 YOLOv3-tiny 网络为基础，进行了替换主干网络，替换 IOU 函数和激活函数的改动，并使用 VOC 数据集进行了训练。训练完成后又进行了剪枝、BN 层融合和量化的策略以进一步轻量化。

第四章：硬件加速器设计。本章首先对本文采用的高层次综合技术进行了介绍，然后说明了本设计的总体架构，并对访存模块的采用的循环分块、双缓存模块和流水线设计进行了讲解，然后介绍了本设计中针对三种不同的卷积设计的对应的运算模块，最后针对上采样层和后处理模块的设计进行了介绍。

第五章：测试与结果分析。本章首先介绍了测试实验使用的硬件平台和软件平台，然后讲解了操作的具体流程与实验数据，最后将本设计的数据与 CPU、GPU 平台和其它文献设计的 AI 加速器分别进行了横向与纵向对比。

第六章：总结与展望。本章总结了全文的所有内容，并针对所做工作提出了一些不足之处与对未来研究的展望。

第二章　卷积神经网络与 FPGA 芯片原理

本章对包括卷积层，激活层，池化层，批量归一化层，全连接层的卷积神经网络的基本结构进行讲解，还介绍了 YOLO 系列目标检测网络的架构和发展，并对其工作原理进行分析，最后针对硬件实现需要的 FPGA 芯片进行了介绍。

2.1 卷积神经网络基础

神经网络是一种模拟动物神经网络行为特征的分布式并行信息处理模型的数学模型。而卷积神经网络是一种特殊的神经网络，它使用卷积和池化来提取特征，并在此基础上发展起来。与其他人工神经网络不同，卷积神经网络通过卷积滤波器提取局部特征，同时使用池化对特征进行下采样，以减少计算量并提高模型的泛化性能。卷积和池化可用于提取各种形式的畸变不变性目标特征，例如位移和缩放。在卷积过程中，网络使用滑动窗口的方式以固定步长提取局部特征，以局部感受野和权值共享的方式保存与像素点相关的信息，大大减少了网络参数量。池化下采样技术则从一个区域中选择一个值作为代表，以减少特征图和参数数量，进一步增加模型深度，提高泛化性能。

随着卷积神经网络的不断完善，从最初只包括输入层、卷积层、激活层、全连接层和输出层，发展到现在增加了批量归一化层和池化层以及其它一些功能层。一般卷积神经网络通过多层卷积层、批量归一化层和激活函数提取输入图像特征，同时使用池化层增加感受野，防止过拟合。最后将提取的特征图通过全连接层和输出层进行分类输出。下面将对卷积神经网络中每一层的结构与作用一一进行说明。

2.1.1　卷积层

卷积层是卷积神经网络中最重要的组成部分，同时也是计算量最大、最耗时的部分。卷积层使用卷积运算对输入的特征图进行特征提取，为后续分析打下坚实的基础。卷积运算源于信号处理中的卷积方法。对于连续函数，卷积的定义式如式 2.1 所示，其中 x(a)和 w(t-a)分别是实数域 R 上的可积函数。

$$s(t) = \int x(a)w(t-a)da \tag{2.1}$$

对于离散函数，卷积的定义式如式 2.2 所示。

$$s(n) = \sum_{m} x(m)w(n-m) \tag{2.2}$$

对于二维图像的卷积计算定义如式 2.3 所示。

$$s(i,j) = \sum_{m}\sum_{n} I(i-m, j-n)\, K(m,n) \tag{2.3}$$

式 2.3 中，I(m,n)表示输入特征图在点(m,n)上的像素值，K(m,n)表示 m×n 大小的卷积核或滤波器。

在进行图像处理的过程中，卷积层实际上是使用卷积核对图像进行滑动，在卷积核窗口内的图像特征值和卷积核权重值进行乘加运算，从而提取图像的局部特征。图 2.1 为 3x3 大小的卷积核对输入图像进行卷积运算的过程。

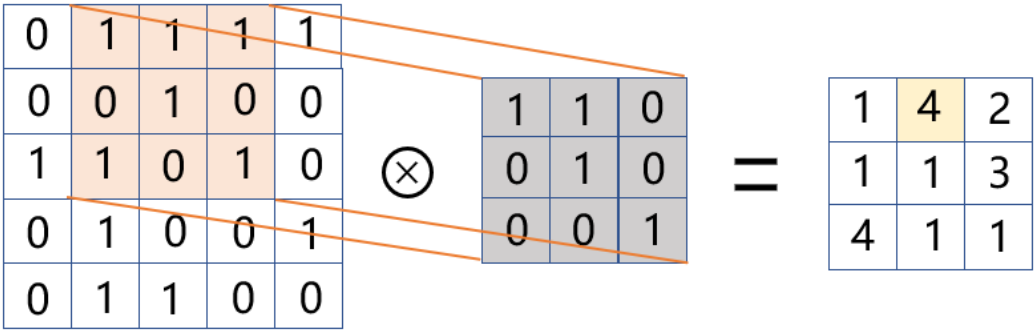


图 2.1 卷积运算示例

图 2.1 展示了一个 3×3 的卷积核对 5×5 的输入图像的卷积过程，按照规定的步长，卷积核在输入图像上自左至右、自上而下一步一步地卷积得到最右侧的 3×3 的特征图。在完整的卷积神经网络过程中，卷积核中的每个参数都是通过不断学习得到更新的，这些参数决定卷积层中正向传播的结果，利用梯度下降算法和反向传播算法实现参数的更新。

2.1.2 激活层

激活层是将上一层神经元的输出映射到下一层神经元输入的一个非线性函数，引入非线性的激活函数可以有效地加强深度卷积神经网络的表达能力。如果没有激活函数，由于卷积实际上就是线性的乘加运算，不管网络层深度有多深，最终都能用一个线性方程表示，它的表达能力不足，因此引入激活函数是很有必要的。常见的激活函数有 Sigmoid 函数、tanh 函数、ReLU 函数、LeakyReLU 函数等。

Sigmoid 函数是一种常见的激活函数，它将连续的实值输入映射到[0,1]之间的输出。特别地，当输入非常大的负值是输出为 0；当输入非常大的正值时输出为 1，其表达式如式 2.4 所示，函数曲线如图 2.2 所示。

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{2.4}$$

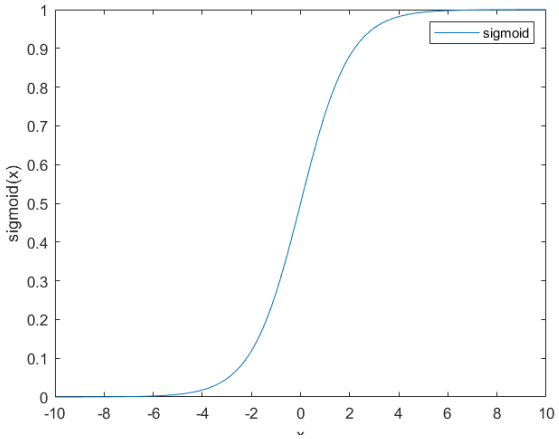


图 2.2 Sigmoid 函数曲线

Sigmoid 函数曾被广泛使用，但近年来使用越来越少。主要原因有：Sigmoid 函数只有在接近于零的较小区间里才有较大的梯度。而在此之外的区间梯度接近 0，很容易引起梯度消失问题；Sigmoid 函数的输出是非零均值，导致下一层神经元的输入是非零均值，所以在反向传播过程中会出现捆绑现象，使得要么全部更新到正向或全部向负向更新，导致网络收敛速度慢；由于 Sigmoid 函数中包含幂和除法运算，对于大型网络来说，计算量也会大大增加，从而导致运算速度较慢。

为解决非 0 均值的问题，与 Sigmoid 函数类似的 tanh 函数开始被应用到激活层中，其表达式如式 2.5 所示，函数曲线如图 2.3 所示。

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.5}$$

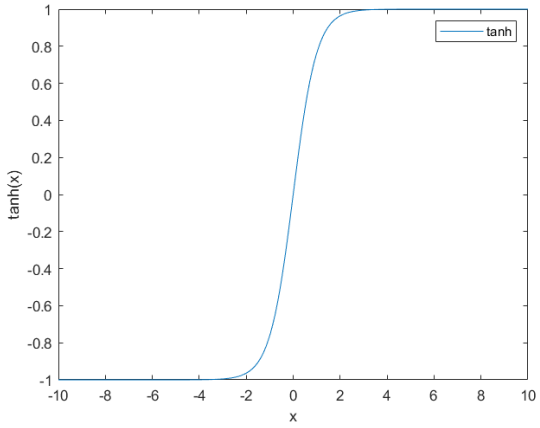


图 2.3 tanh 函数曲线

虽然 tanh 函数解决了非 0 均值的问题，但是仍然存在梯度消失和幂运算计算量太大的问题，于是 ReLU 函数便被提出。ReLU 函数是当前网络中最常用的激活函数，式 2.6 为其表达式，函数曲线如图 2.4 所示。

$$ReLU(x) = \max(0, x) \tag{2.6}$$

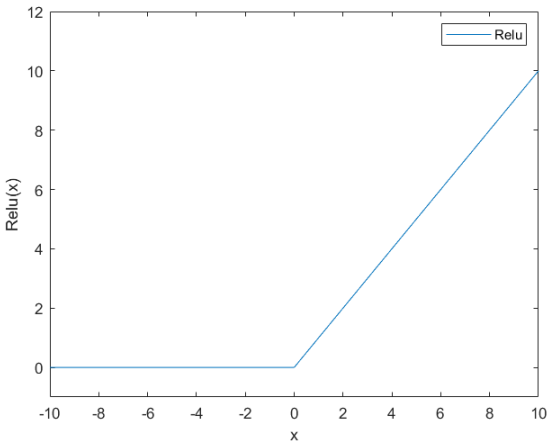


图 2.4 ReLU 函数曲线

可以从数学表达式和函数曲线中看出，ReLU 函数将小于 0 的值全部赋值为 0，而大于或等于 0 的值则保持不变。由于其导数值在输入正范围内都是 1，因此具有较快的收敛速度且避免了梯度消失的问题。另外，由于 ReLU 函数计算简单，且梯度不会饱和，所以使用 ReLU 函数的收敛速度要比 Sigmoid 和 tanh 快很多。

但由于 ReLU 函数在输入为负值的时候输出恒等于零，会出现神经元坏死现象，可能存在着某些神经元可能永远不会被激活，从而对应的参数可能永远不能被更新。为了很好的解决这一问题，提出了 LeakyReLU 函数，即当输入小于 0 的时候，输出值不再是 0，而是一个拥有较小斜率的函数，比如令斜率等于 0.1。

LeakyReLU 函数的表达式如式 2.7 所示，与 ReLU 函数的区别仅仅在于将负数乘以系数 α输出，这样就避免了输入负数值时输出恒定为 0 的问题。图 2.5 为系数α取 0.1 时的 LeakyReLU 函数曲线。

$$\mathrm{LeakyReLU(x)} = \begin{cases} x, & x > 0 \\ \alpha x, & \mathrm{x} \le 0 \end{cases} \tag{2.7}$$

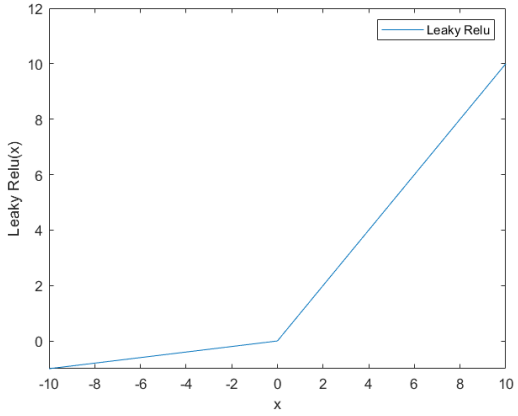


图 2.5 LeakyReLU 函数曲线

2.1.3 批量归一化层

批量归一化层（Batch Normalization layers，BN 层）的本质是对数据进行归一化处理，其优化方法被广泛使用。数据归一化可以将分散的数据进行统一，从而加快神经网络模型的收敛过程、提高训练速度，并且可以缓解梯度爆炸的问题。不同于常规的数据归一化方式，卷积神经网络中的 BN 层加入了 γ 和 β 两个参数，这两个参数是 BN 层的关键所在。通过学习 γ 和 β 两个参数，BN 层可以自适应地选择当前特征分布是否需要进行重构调整。BN 层可以有效地提高损失函数的收敛速度和减少模型训练时间。此外，BN 层还可以避免梯度消失和梯度爆炸的问题，并增强模型的泛化能力[54]。

对于一个批次的数据 I，样本数为 m，而 x_i 是 I 中的样本，首先求得所有样本的平均值μ_i和方差σ_i^2，如式 2.8 和 2.9 所示。

$$\mu_i = \frac{1}{m}\sum_{i}^{m} x_i \tag{2.8}$$

$$\sigma_i^2 = \frac{1}{m}\sum_{i}^{m}(x_i - \mu_i) \tag{2.9}$$

然后利用求得的均值和方差对数据归一化，如式 2.10 所示，其中ε是一个极小值。

$$x_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}} \tag{2.10}$$

最后引入了尺度变换因子γ和偏移项β，这两个可学习的参数可以恢复数据本身的表达能力。利用参数γ和β对标准化后的数据进行线性变换，如式 2.11 所示。

$$y_i = \gamma x_i + \beta \tag{2.11}$$

2.1.4 池化层

池化层一般接在 BN 层的后面，通过池化层对卷积得到的特征图进行下采样操作，使得特征图的尺寸缩小，从而有效减少模型的参数量和计算量，以便更好的提取特征，并且在一定程度上还可以缓解过拟合。池化运算一般有平均值池化和最大值池化两种方式。最大值池化的运算示例如图 2.6 所示。

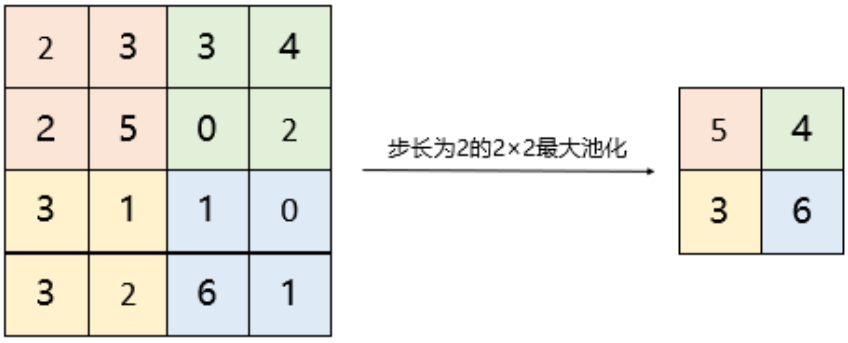


图 2.6 最大池化运算示例

图 2.6 中，利用一个 2×2 的池化核对 4×4 的特征图进行最大值池化操作，自左至右，自上而下以步长为 2 进行依次滑动，在每个 2×2 的方格内进行局部最大值处理，最终得到一个 2×2 的新特征图。新特征图的长宽变为原特征图的一半。池化层的最大特点就是所含神经元没有权重参数，而且能学习到图像的纹理和边缘结构。

2.1.5 全连接层

在卷积神经网络中，全连接层（fully connected layers，FC 层）通常作为最后的输出层，其作用类似于分类器。全连接层进行和卷积层类似的乘加运算，但不同的是全连接层内的所有输入都参与运算。全连接层的每个神经元节点都与前一层的所有神经元节点相连，实质上是将一个特征空间线性变换到另一个特征空间，其计算式如式 2.12 所示，$x_{i,j}$ 为输入特征图，$w_{i,j}$ 为全连接层权重，$y_{i,j}$ 为输出特征图。

$$y_i = \sum_{i,j}^{m,n} x_{i,j}\, w_{i,j} \tag{2.12}$$

全连接层的工作原理如图 2.7 所示。如果全连接层是整个卷积神经网络的最后一层，那么其输出值实际上代表着每个类别的分数。

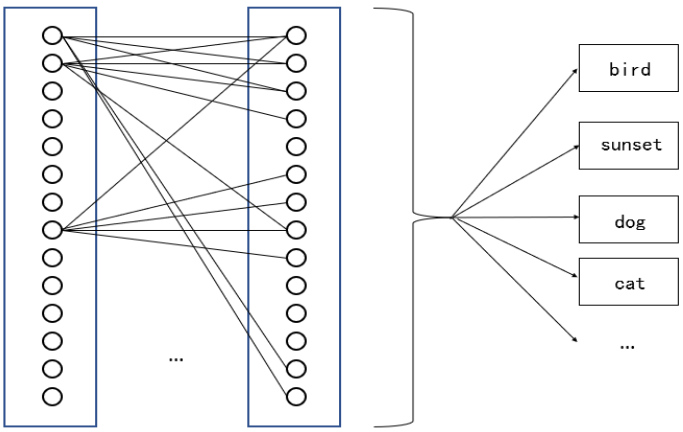


图 2.7 全连接层示例

2.2 YOLO 检测网络

2.2.1 YOLOv1

YOLO(You Only Look Once)系列目标检测算法由 J. Redmon 等人提出。YOLOv1 是一种端到端的单阶段目标检测算法，与传统的滑动窗口技术不同，它将输入图片分成 7×7 的网格，每个单元格检测其中心点是否存在目标。相比于以往的两阶段目标检测算法，YOLO 算法省去了候选框提取的步骤，从而节约了模型推理的时间。YOLO 算法具有快速检测和高准确度的优点，因此已成为当今目标检测领域的主要研究方向之一。YOLOv1 算法网络结构如图 2.8 所示。

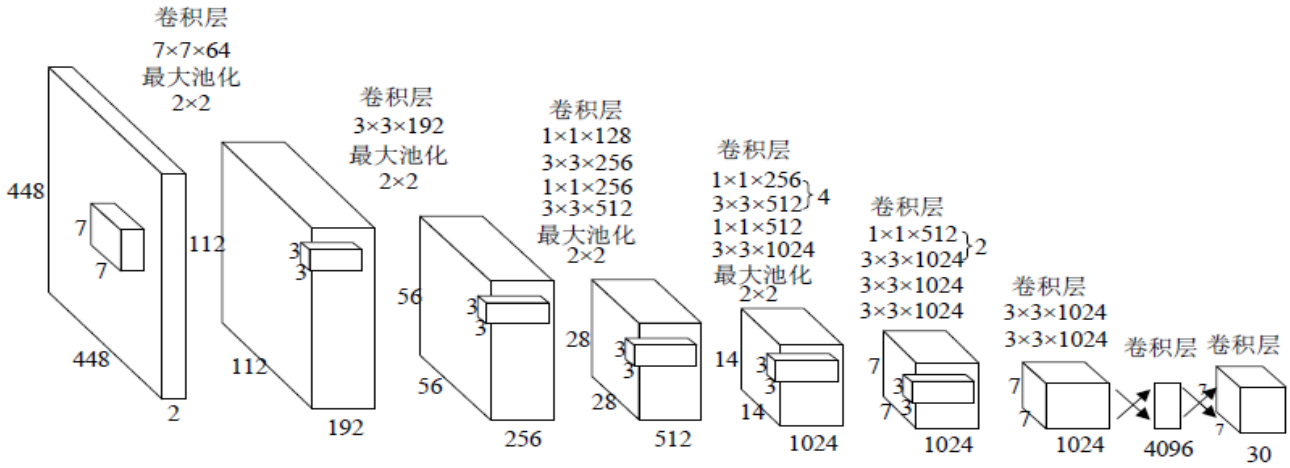


图 2.8 YOLOv1 网络结构

YOLOv1 算法实现原理步骤如下：

首先，对输入图片进行缩放，将其调整为 448×448 大小，并将其分成 7×7 的网格。如果待检测的目标位于网格中心，则该网格直接预测该目标。

然后，每个网格需要预测两个边界框，每个边界框都包含 5 个预测值：两个预测框中心坐标值和两个预测框宽高值，以及一个置信度预测值。此外，在 VOC 数据集中，每个网格还需要预测 20 个类别。因此，YOLOv1 网络的输出特征矩阵大小为 7×7×30。

最后，通过非极大值抑制算法，去除冗余边界框，并保留得分最高的预测边界框，其示例如图 2.9 所示。然而，YOLOv1 算法存在一些问题，例如主干网络提取特征的效果较差，网格总数也较少，每个网格最多只有两个预测框，这使得它定位精度较低，处理小目标不佳，而且也无法检测不同尺度的目标。

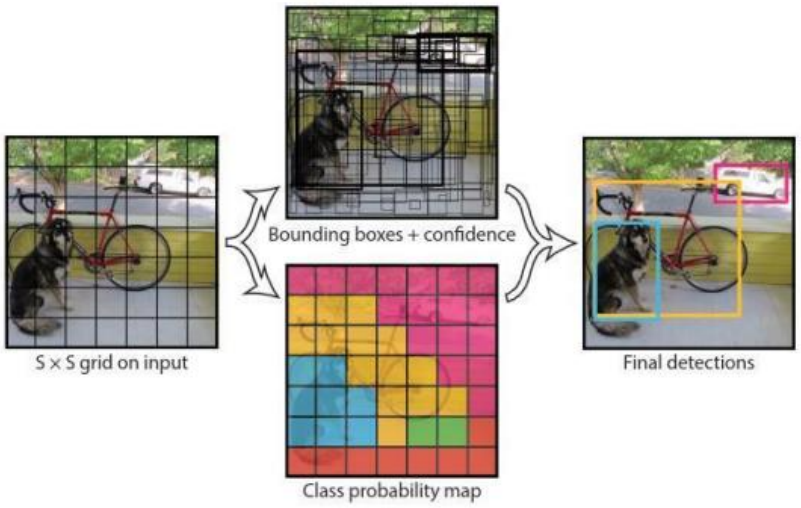


图 2.9 Yolo v1 预测原理

2.2.2 YOLOv2

针对YOLOv1算法存在的问题，J.Redmon等人提出了YOLOv2算法。算法的改进如下：在每个卷积层后加入批量归一化层，利用调整均值和方差来近似高斯分布，提高模型的非线性表达能力，避免训练不稳定的问题。

通过 K-means 聚类算法得到最佳的锚框大小和数量，与人工经验设计的先验框相比，聚类得到的先验框有更高的平均交并比，便于训练学习。

采用 Darknet-19 网络作为主干网络，它综合了 GoogLeNet 和 VGG-19 网络的优点，使用 1×1 和 3×3 的卷积核计算降低了模型参数和计算复杂度，同时提高了泛化能力，避免了过拟合。

除此之外，YOLOv2 还移除了全连接层、平均池化层和 Softmax 函数，添加了 3 个大小为 3×3×1024 的卷积层。同时，还加入了 Passthrough 层来融合高层和底层特征信息，提升了多尺度检测性能。

虽然经过以上的改进，YOLOv2 已经具有较高的检测水平，但仍存在缺陷：只采用一层特征图进行预测，无法细化检测小物体，因此还需要进一步优化。

2.2.3 YOLOv3

YOLOv3 算法的主干网络经过进一步的升级，采用了拥有 53 层的 Darknet-53 网络。相较于之前的 Darknet-19，这个新的网络在网络深度和特征提取能力方面都有显著的提高。Darknet-53 借鉴了 ResNet 的残差结构，解决了深度增加带来的梯度爆炸问题。

YOLOv3 算法中还采用了特征金字塔的方法，通过上采样获得更大的特征图。为了有效地融合来自不同尺度的特征图，YOLOv3 引入了 FPN 结构，在多个不同尺度的特征图之间建

立通道，使各层的语义特征得到有效融合。最终，YOLOv3 算法得到了 13x13、26x26 和 52x52 三种不同尺寸的新特征图。这些特征图分别对应于图像的深层、中层和浅层，有利于检测不同大小的物体。例如，深层特征图的尺寸较小，感受野范围较大，更适合检测大型物体。而浅层特征图则相反，更适合检测小型物体。这些优化措施进一步提高了 YOLOv3 算法的检测精度。YOLOv3 的架构如图 2.10 所示。

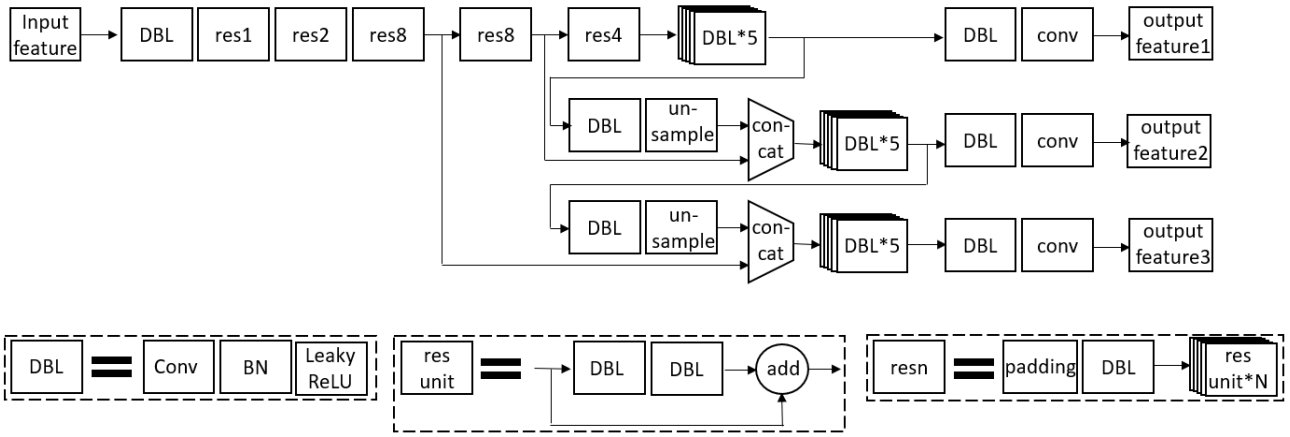


图 2.10 YOLOv3 网络结构

在 YOLOv3 算法被提出后，也不断有新的基于 YOLO 系列架构的检测网络被提出，但事实上，受限于计算资源和软件支持，YOLOv3 算法仍然当前是工业界被应用最广泛的目标检测算法之一。

2.3 FPGA 芯片原理

现场可编程门阵列（Field Programmable Gate Array, FPGA）是一种可编程的逻辑器件，FPGA 通过在芯片上配置逻辑门和触发器，可以按需创建和重组电路，适应不同的应用需求，具有灵活性和可重构性，能够实现数字电路的硬件加速和实时处理。FPGA 的片上资源包括 LUT（Look Up Table，查找表)、FF (Flip Flop，触发器)、DSP (Digital Signal Processor，数字信号处理器）以及 BRAM （Block RAM，块随机存储器)。

2.3.1 查找表

为了实现 FPGA 的可编程性，硬件使用了一种可重复配置的结构：查找表（Look Up Table，LUT）。查找表的功能是基于真值表的思想。它将所有可能的输入组合列出，并为每个组合分配一个对应的输出值。通过在 FPGA 配置期间将适当的真值表写入 LUT 的存储单元中，可以实现特定的逻辑函数。其本质是一个 2^N*1 位的随机存储器，根据地址存储想要输出的结果，可以实现 N 个布尔变量的任意逻辑功能。目前主流的 FPGA 多采用基于 SRAM 工艺的 LUT 结构，可以通过烧写比特流文件来重新配置 FPGA，从而实现反复擦写的能力。

在工作过程中，首先设计者给定一个组合逻辑，并使用开发软件编译生成所有输入输出值。然后，将这些值按照真值表的形式写入到存储器中。接下来，使用 N 个输入组成寻址 RAM 的地址，这样每次输入从 LUT 中查找对应地址的值并输出。一般而言，对于具有 N 个输入的 LUT，可供访问的内存位置数量为 2N，这意味着最多可以实现 2N×N 个函数。一个四输入的 LUT 的具体实现结构如图 2.11 所示，其中 a,b,c,d 是 LUT 的输入，x1 和 x0 是地址，y 是 LUT 的输出。

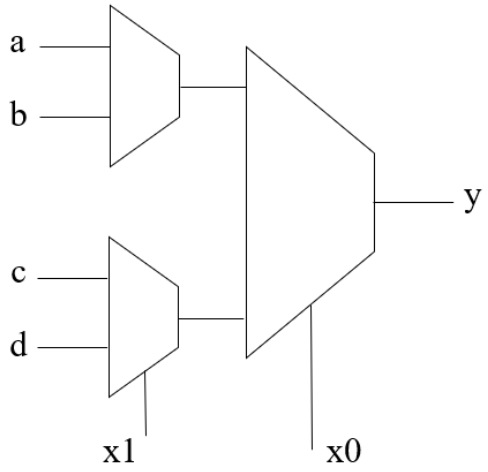


图 2.11 4 输入 LUT 电路结构

由于 LUT 结构的灵活性，这些内存块还可以作为 64 位存储器使用，通常称为分布式存储器。分布式存储器是 FPGA 片上最快的内存类型，因为它可以在片上的任何部分实例化，从而提高电路实现的性能。

总之，FPGA 中的 LUT 是用于实现逻辑函数和布尔运算的重要元素。它们基于真值表的概念，通过配置适当的存储单元内容来实现特定的逻辑功能。LUT 具有灵活性、可编程性和扩展性，并在 FPGA 设计中发挥着关键作用，用于构建各种逻辑电路和实现复杂的数字功能。

2.3.2 触发器

触发器通常用于存储和传输数字信号，在时钟信号的控制下进行状态转换。它们是构成数字逻辑电路的基本构建块之一，广泛应用于时序逻辑和存储器设计中。触发器的基本结构包括数据输入、时钟输入、时钟使能、复位和数据输出。在时钟的上升沿或下降沿时，输入的数据存储在触发器内部，并在时钟边沿更新输出。时钟使能管脚的作用是允许触发器保持一个以上时钟脉冲的特定值。只有当时钟和时钟启用都等于 1 时，新的数据输入才被锁定并传递到数据输出端口。

触发器在 FPGA 中起到了至关重要的作用。它们可用于实现寄存器、计数器、状态机以

及存储器等功能。通过合理的配置和连接触发器，可以实现各种复杂的数字电路设计。触发器的性能参数包括时钟速度、时序关系和存储能力等。时钟速度指的是触发器能够响应时钟信号的最大频率。时序关系表示触发器之间的相互依赖关系和数据传输顺序。存储能力指的是触发器可以存储的位数或状态数。总之，触发器是 FPGA 中的重要组成部分，用于存储和控制数字信号。它们在实现时序逻辑和存储器设计方面发挥关键作用，为构建复杂的数字电路提供了灵活和可编程的功能。触发器的结构如图 2.12 所示。

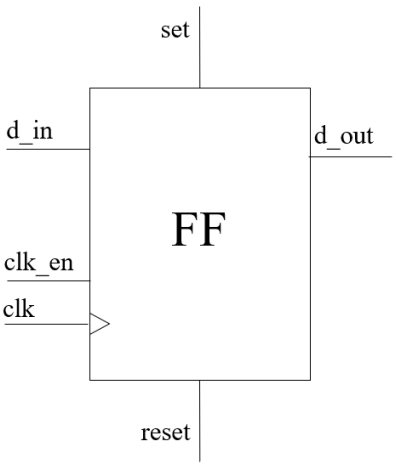


图 2.12 触发器

2.3.3 DSP

DSP 是一种专门设计用于高效处理数字信号的硬件模块，通常包含了一系列算术逻辑单元（ALU）、乘法器、累加器和寄存器等功能。FPGA 中的 DSP 模块用于执行各种数字信号处理任务，如滤波、快速傅里叶变换（FFT）、乘法累加（MAC）操作和数字调制解调等。DSP 模块的设计和功能可以通过 FPGA 的配置进行定制，以适应特定的应用需求。

DSP 模块在 FPGA 中具有许多优势。首先，它们提供了高度并行的计算能力，可以同时处理多个数据流。其次，DSP 模块通常具有专用的算术逻辑和乘法器，可以实现高性能的数字运算。此外，FPGA 中的 DSP 模块还具有灵活性和可编程性，可以根据应用需求进行配置和重新编程。在 FPGA 设计中，DSP 模块通常以片上可配置资源的形式存在。设计人员可以根据应用的信号处理需求，灵活地分配和配置 DSP 模块的数量和功能。这样可以有效地利用硬件资源，并实现高性能的数字信号处理功能。

总之，FPGA 中的 DSP 模块是专门用于数字信号处理的硬件单元。它们具有高度并行的计算能力、专用的算术逻辑和乘法器，并且可以根据需求进行灵活的配置和重新编程。DSP 模块在 FPGA 设计中发挥着重要的作用，可用于实现各种数字信号处理任务，提供高性能和

定制化的解决方案。本文选取的 Zynq-7020 FPGA 中使用了 DSP48E1 模块，其电路简图如图 2.13 所示。

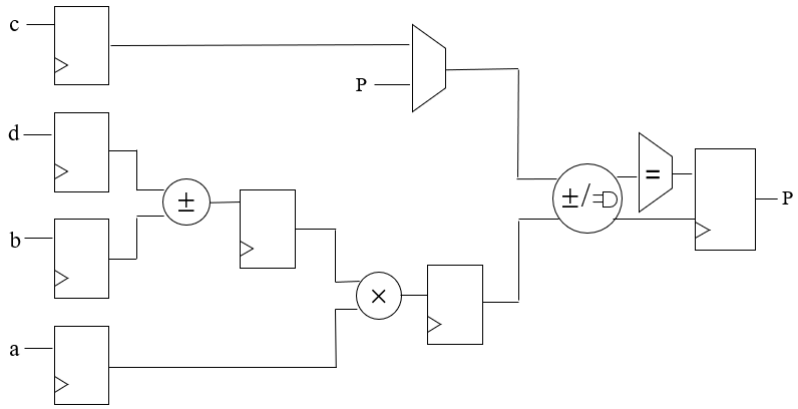


图 2.13 DSP48E1 电路简图

2.3.4 BRAM

BRAM 代表块随机存储器（Block RAM）。BRAM 是一种高速、可配置的存储器模块，用于在 FPGA 中实现数据存储和访问操作。BRAM 是以块的形式存在于 FPGA 芯片中，每个块都包含了多个存储单元组成的存储数组。每个存储单元可以存储一个固定大小的数据，并且具有快速的读写能力。BRAM 还包含了寻址和控制电路，用于有效地管理存储单元的访问。BRAM 的电路简图如图 2.14 所示。

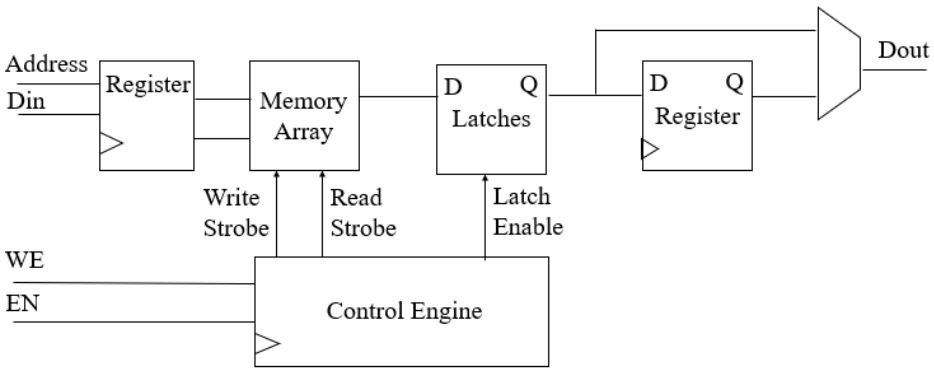


图 2.14 BRAM 结构简图

FPGA 中的 BRAM 模块可用于多种应用。首先，它们可以用作存储器，用于存储程序指令、数据和中间结果等。这使得 FPGA 能够实现复杂的算法和数据处理功能。其次，BRAM 还可用于实现缓冲区或 FIFO（First-In-First-Out）缓存，用于数据的流水线传输和临时存储。此外，BRAM 还可以用于存储查找表的内容，用于实现逻辑函数计算和状态存储。

FPGA 中的 BRAM 具有许多优点。首先，它们具有高速读写能力和低延迟，可以提供快速的数据访问。其次，BRAM 具有可配置性，可以根据应用需求调整存储单元的数量和大小。

此外，BRAM 还可以通过多个端口实现并行读写操作，以满足高带宽和低延迟的需求。

在 FPGA 设计中，设计人员可以使用设计工具来配置和连接 BRAM 模块，以满足特定应用的存储需求。他们可以灵活地分配和管理 BRAM 资源，以实现最佳的存储器组织和性能。

总之，FPGA 中的 BRAM 模块是高速、可配置的存储器单元，用于实现数据存储和访问操作。它们可用于存储程序指令、数据、中间结果等，并且具有高速读写能力、低延迟和可配置性。BRAM 在 FPGA 设计中发挥着重要作用，为实现复杂的算法和数据处理提供了高性能和灵活的存储解决方案。

2.4 本章小结

本章首先分析了卷积神经网络的整体结构和工作原理，并对每一层都做了详细的说明。然后介绍了 YOLO 系列目标检测网络从 v1 到 v3 版本的发展沿革与具体架构，最后对 FPGA 芯片的基本原理和内部的基本结构进行了介绍，为轻量化 YOLO 网络设计和 FPGA 上板实现打下基础。

第三章　轻量化 YOLO 网络实现

YOLO 检测网络经过发展和改进，其检测准确度已经得到了大幅的提升，但随之而来的问题是网络结构也越来越复杂，参数越来越多，对模型的部署带来了难题。因此本章将介绍 YOLO 网络的轻量化实现方法，以 YOLOv3-tiny 网络为基础，对模型进行了一系列的改进，实现了网络的轻量化。

3.1 YOLOv3-tiny

Darknet53 与 FPN 的引入使得 YOLOv3 的检测精度表现十分出色，但由于其模型复杂，参数量大，对于设备要求过高，检测速度的表现欠佳，因此 YOLOv3-tiny 被提出。YOLOv3-tiny 主要在两个方面对算法进行了简化：主干网络和检测分支。主干网络由原本的 Darknet53 简化为 4 个点卷积层和 9 个 3x3 卷积层。检测分支也从 3 个减少为 2 个，同时减少了每个检测分支上的卷积层，实现了模型压缩。不仅保证了算法在轻量级平台上的高性能和检测精度，而且提高了算法的检测速度，成为轻量级目标检测领域的首选算法之一，其网络结构如图 3.1 所示。

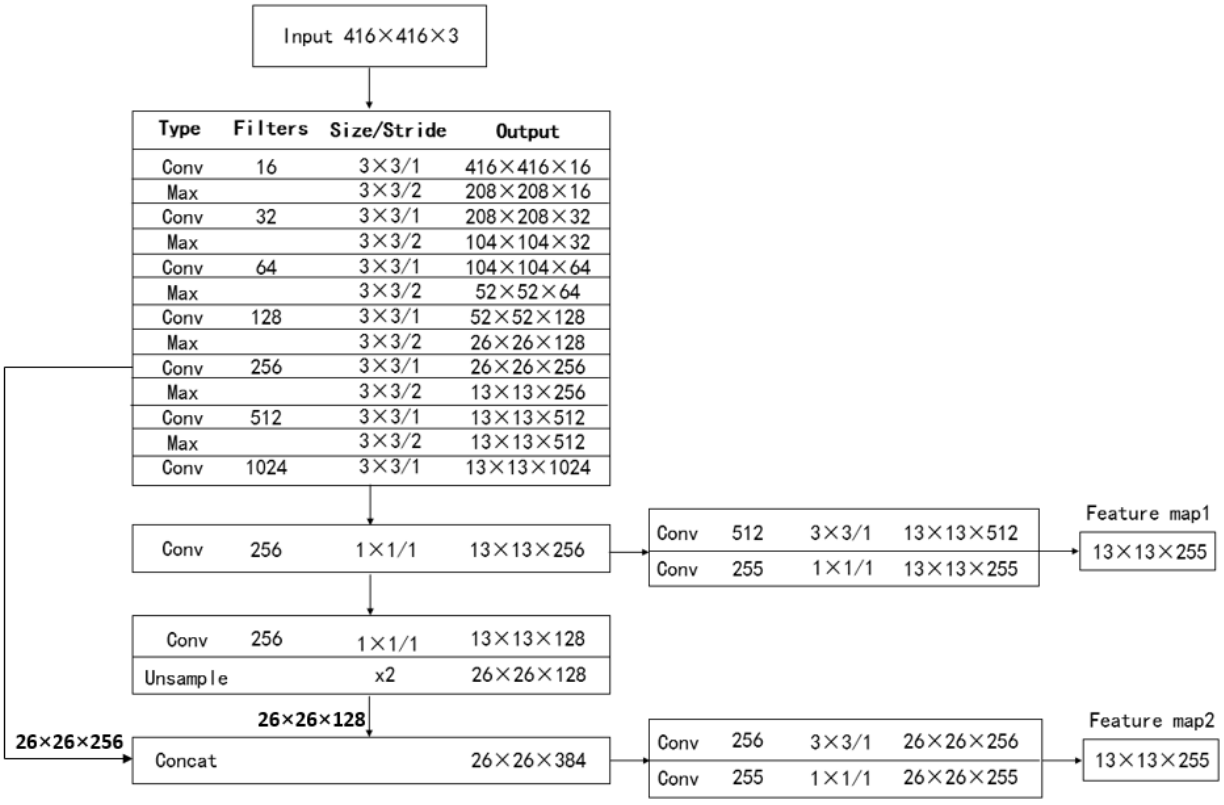


图 3.1 YOLOv3-tiny 网络结构

YOLOv3-tiny 的主干网络由 9 层卷积层和 6 层最大池化层构成，在卷积层中，原始输入图像进行待检测目标的特征提取。在最大池化层中，对卷积层输出特征矩阵进行降采样。与 YOLOv3 相同，YOLOv3-tiny 也采用了 FPN 结构。但为了提高检测速度，YOLOv3-tiny 仅对

13x13 和 26x26 两个不同分辨率的特征图进行融合，仅在这两种尺度的特征图上进行结果预测。通过七层卷积，提取图片特征，再经过两次卷积得到 13x13 尺度的输出。然后，将第九层的特征图进行 2 倍上采样，与第四层特征图融合。再经过两层卷积，得到 26×26 尺度的输出。

相比起 YOLOv3，虽然 YOLOv3-tiny 在一定程度上牺牲了一些精度，但这换取了非常快速的检测速度。因为它同时具备高检测精度和极快的检测速度，被广泛地应用于工业领域，是目标检测实际运用的首选算法之一。

3.2 轻量化 YOLO 网络设计

3.2.1 主干网络替换

YOLOv3-tiny 的探测精度低一个很重要的原因是主干网络比较浅，只有 7 层，难以提取更高层次的语义特征。因此可以考虑将其替换为 MobileNet 网络，可以在维持一个较小的计算量的前提下加深主干网络的层数以进一步提取特征。

MobileNet 网络的核心思想是通过深度可分离卷积（Depthwise Separable Convolution）代替标准卷积，来减少模型参数数量，从而达到降低计算复杂度和模型大小的目的。MobileNet 的结构如图 3.2 所示，它拥有 27 个卷积层，包括一层标准卷积（Conv）、13 层深度卷积（DW Conv）和 13 层点卷积（PW Conv），除此之外，MobileNet 的所有卷积层后都有一个批量归一化层和 ReLU 激活函数。

Type	Filters	Size/Stride
Conv	32	3×3/2
DW Conv	32	3×3/1
PW Conv	64	1×1/1
DW Conv	64	3×3/2
PW Conv	128	1×1/1
DW Conv	128	3×3/1
PW Conv	128	1×1/1
DW Conv	128	3×3/2
PW Conv	256	1×1/1
DW Conv	256	3×3/1
PW Conv	256	1×1/1
DW Conv	256	3×3/2
PW Conv	512	1×1/1
5× DW Conv		3×3/1
5× PW Conv	512	1×1/1
DW Conv	512	3×3/2
PW Conv	1024	1×1/1
DW Conv	1024	3×3/1
PW Conv	1024	1×1/1
Avg Pool	1024	7×7/7
FC	1000	
Softmax Classifier		

图 3.2 MobileNet 网络结构

标准卷积计算过程可以由图 3.3 表示，假设输入特征图的尺寸为 $H_i \times W_o$，通道数为 C_i，卷积核的大小为 $K \times K$，数量为 C_o，最终输出的特征图的尺寸为 $H_o \times W_o$，通道数为 C_o。其计算过程是首先使用 C_o 个 K×K 的卷积核，完成对输入特征图的特征提取。然后将计算出的 C_o 张特征图堆叠组合起来，最终得到一个 $H_o \times W_o$ 的特征图输出。

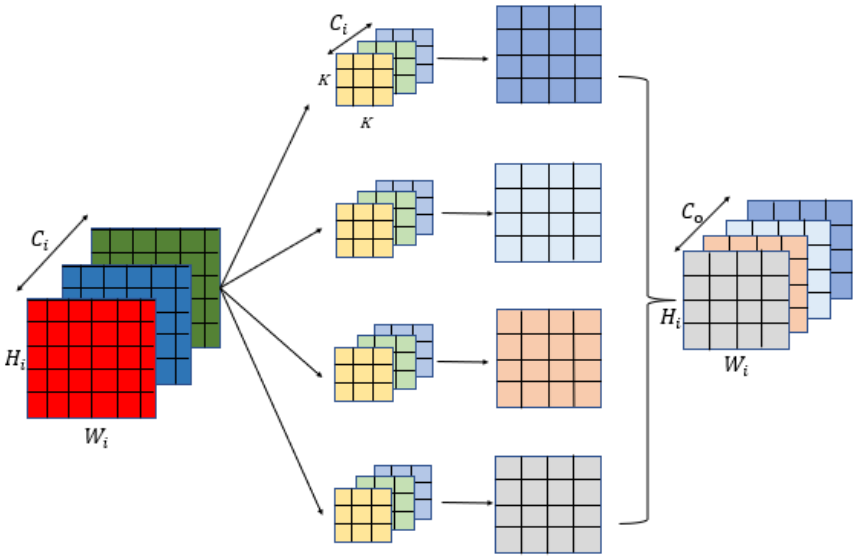


图 3.3 标准卷积计算过程

深度卷积与标准卷积不同，不需要做通道方向的累加。计算过程如图 3.4 所示，其卷积核的尺寸为 K×K×C，但是卷积核被拆分成单通道形式，即 C 个大小为 K×K 的滤波器，对输入特征图的对应通道做卷积操作。深度卷积的输出特征图通道数目与输入特征图、卷积核一致，均为 C。深度卷积的主要作用在于提取特征。其缺点在于通道数较少时，特征的维度也较少，难以获得足够的信息。

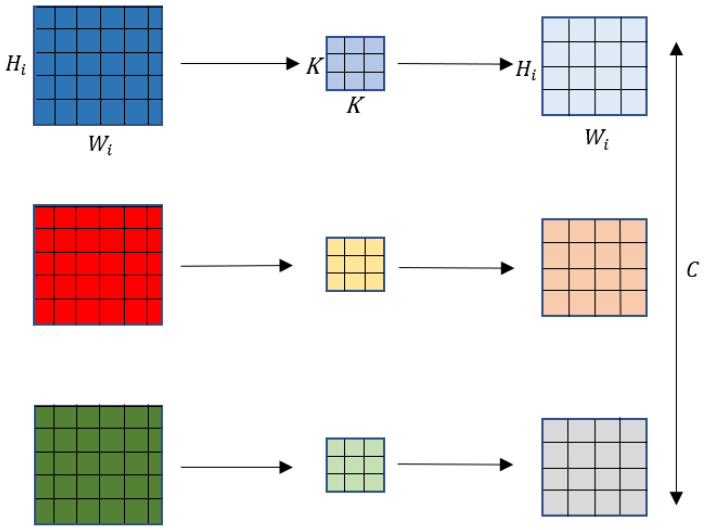


图 3.4 深度卷积计算过程

特征维度的升降主要由点卷积完成。点卷积与标准卷积的计算方式类似，只是卷积核的

大小都为 1×1，如图 3.5 所示。尺寸为 H×W×C 的输入特征图与 F_o 个 1×1×C 的卷积核卷积得到尺寸为 $H_o \times W_o \times F_o$ 的输出特征图。由于 1×1 的卷积核并不能改变输出特征图的高和宽，故在点卷积中 $H_o=H_i$，$W_o=W_i$，通过设置点卷积核的数量就能完成对特征图的升降维。

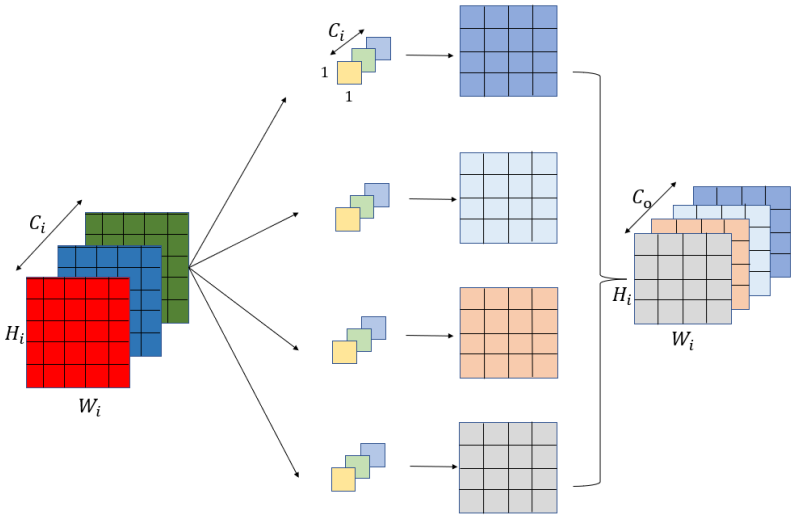


图 3.5 点卷积计算过程

MobileNet 利用深度可分离卷积代替传统网络结构中的标准卷积。其中深度卷积利用单个卷积核对输入特征图的每个通道进行特征提取，点卷积则组合深度卷积的输出。这样的模式能够在减少了网络模型的计算量和权重参数量的同时很好地完成原来标准卷积的功能。

下面假设在卷积步长为 1 的前提下比较标准卷积和深度可分离卷积在权重参数量和计算量上的差别。

标准卷积的卷积核尺寸是 $K \times K \times C_i$，共有 F_o 个，故标准卷积的参数量 $Para_{conv}$ 为：

$$Para_{conv} = K \times K \times C_i \times F_o \tag{3.1}$$

对于一个大小为 $H_o \times W_o \times F_o$ 的输出特征图，每个卷积核都要进行 $H_o \times W_o$ 次计算，故标准卷积的计算量 $Flops_{conv}$ 为：

$$FLOPS_{conv} = K \times K \times C_i \times H_o \times W_o \times F_o \tag{3.2}$$

深度可分离卷积的参数量包含深度卷积的参数量和逐点卷积的参数量。深度卷积的卷积核大小为 $K \times K \times C_i$；逐点卷积的卷积核大小为 $1 \times 1 \times C_i$，共有 F_o 个，故深度可分离卷积的参数量 $Para_{dsconv}$ 为：

$$Para_{ds\,conv} = K \times K \times C_i + F_o \times C_i \tag{3.3}$$

考虑计算量，深度卷积需要进行 $W_o \times F_o$ 次运算；逐点卷积大小为 $1 \times 1 \times C_i$，的卷积核需要做 $H_o \times W_o$ 次运算，而逐点卷积的卷积核共 F_o 个。故深度可分离卷积的计算量 $Flops_{dsconv}$ 为：

$$Flops_{ds\,conv} = K \times K \times C_i \times H_o \times W_o + C_i \times F_o \times H_o \times W_o \tag{3.4}$$

式 3.5 和式 3.6 分别比较了标准卷积和深度可分离卷积的参数量和计算量。

$$n_{para} = \frac{K \times \mathrm{K} \times C_i + C_i \times F_o}{K \times \mathrm{K} \times C_i \times F_o} = \frac{1}{F_o} + \frac{1}{K^2} \tag{3.5}$$

$$n_{compute} = \frac{K \times \mathrm{K} \times C_i \times H_o \times W_o + C_i \times F_o \times H_o \times W_o}{K \times \mathrm{K} \times C_i \times F_o \times H_o \times W_o} = \frac{1}{F_o} + \frac{1}{K^2} \tag{3.6}$$

观察式 3.5 和式 3.6 可知利用深度可分离卷积代替标准卷积后参数数量和计算量都下降为原来的 $1/F_o+1/K^2$。由于在 MobileNet 中通常使用的是3×3的卷积核，因此参数量和计算量均只有标准卷积的 1/8 到 1/9。

由上面的计算可以得知，使用深度可分离卷积代替标准卷积，可以有效减小计算量和参数量，而且 MobileNet 的深度也深于 YOLOv3-tiny 的 7 层卷积主干网络，有利于更好的特征提取。但是 MobileNet 的深度也带来了一个问题，相比起原本的 7 层卷积主干，参数量仍然很大，不符合轻量化要求。因此本文最终采用 0.5 Mobilenet 作为主干网络。0.5 Mobilenet 是将 MobileNet 的所有通道数缩减为原来的一半的网络模型。根据研究表明，MobileNet 进行通道的缩减能在保证准确率下降不太多的前提下有效减少模型的参数量，且对通道进行缩减优于直接裁掉卷积层[38]。除此之外，在本网络设计中，仍然使用了 LeakyReLU 代替了 MobileNet 中的 ReLU 函数，解决了当计算结果为负数时直接归零导致丢失信息的问题。修改后的网络结构如图 3.6 所示。

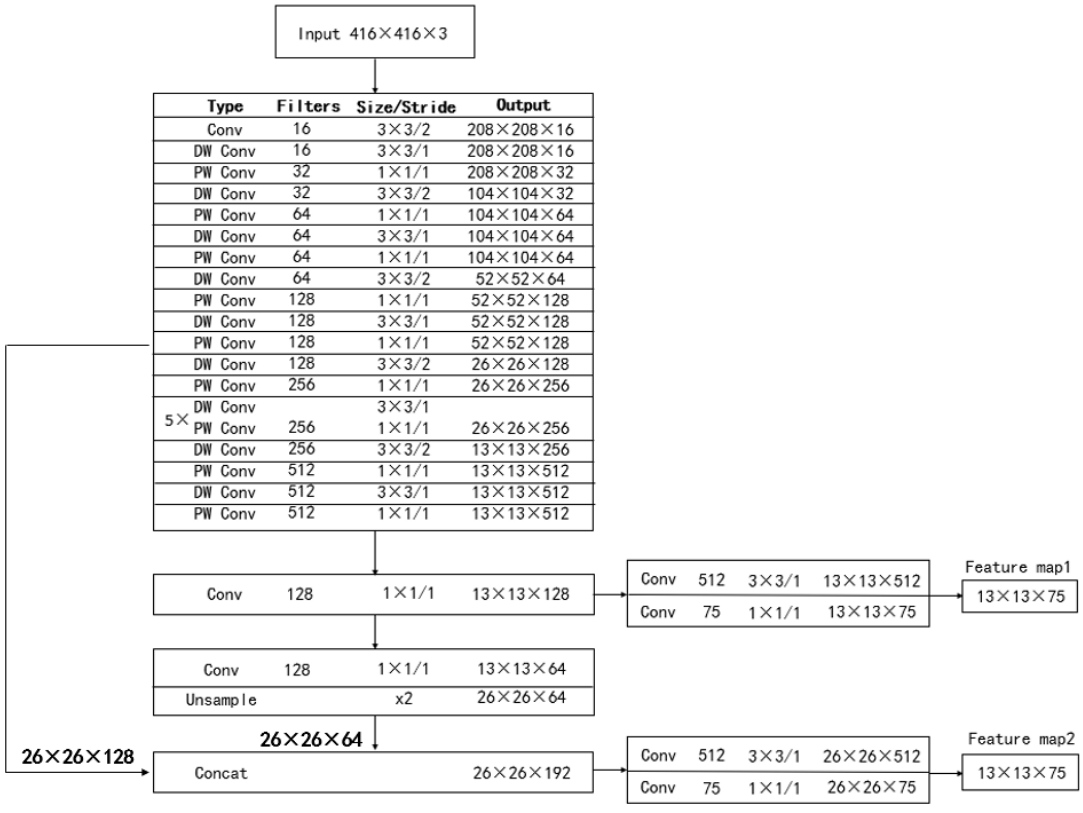


图 3.6 替换主干为 0.5 MobileNet 的 YOLOv3-tiny

3.2.2 GIOU

神经网络训练中，需要反复更新网络权重参数来提高位置预测的准确性。参数的更新是通过定义边界框回归损失函数、计算回归损失值 Loss，再通过反向梯度下降实现的。在目标检测中，IOU(Intersection over Union)是一项重要的回归度量函数。当图像输入网络模型后，模型应该给出一个预测结果，即在照片中可能存在的待检测物体的位置和范围。这样，模型的预测结果和目标之间就会出现误差，而评估这种误差的方法就是 IOU Loss。

IOU 的定义是指预测框与真实框之间的交集面积与它们的并集面积的比值。IOU 和 IOU Loss 的计算式如式 3.7 和式 3.8 所示。

$$IOU = \frac{A \cap B}{A \cup B} \tag{3.7}$$

$$IOU\ Loss\ = 1 - IOU \tag{3.8}$$

IOU 的取值范围在 0 到 1 之间，其中 1 表示完全重合，0 表示没有重合。相比起 L1 或 L2 范数等简单损失函数，IOU Loss 还具有尺度不变性，无论检测框的尺度大小如何，输出的 IOU Loss 始终在 0 到 1 之间。因此，IOU Loss 能够更好地反映预测框与真实框的检测效果。

然而，IOU 在作为损失函数时也有一些缺点。例如图 3.7 所示，左右两种情况下 A 框与 B 框都完全不重合，两个框之间的距离也不一样，但无论两个框距离多远其 IOU Loss 都为 1，因此 IOU 无法准确反映两者之间的距离大小。

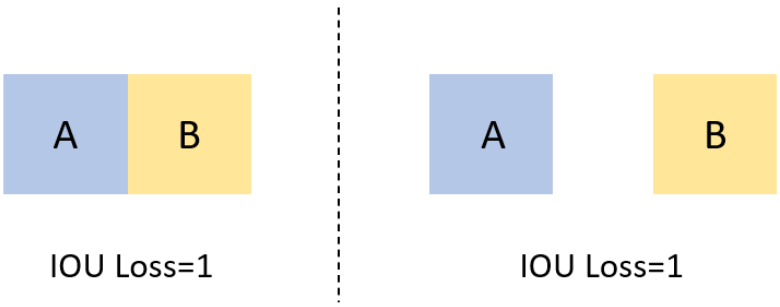


图 3.7 IOU 不能反应距离大小

此外，IOU 还存在一个缺陷，即不能精确地反映两个边界框之间的重合程度。如图 3.8 所示的三种情况，三个边界框的 IOU 数值相等，但是可以看出它们之间的重合程度却是完全不同的，左边的边界框的回归效果最好，中间的次之，右边的最差。

图 3.8 IOU 不能精确反应重合度大小

为克服 IOU 的缺点，GIOU[55]应运而生，如图 3.9 所示，黄色填充的 A 为预测框，蓝色填充的 B 为真实框，黑色实线框 C 是包含 A 与 B 的最小框。

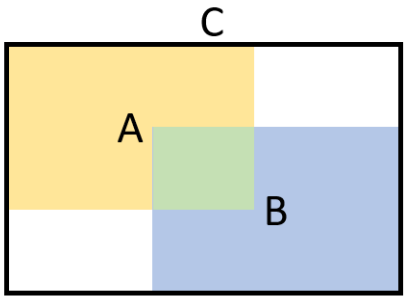


图 3.9 GIOU 示意图

那么 GIOU 和 GIOU Loss 的计算方式如式 3.9 和 3.10 所示。

$$GIOU = IOU - \frac{C - A \cup B}{C} = IOU - 1 + \frac{A \cup B}{C} \tag{3.9}$$

$$GIOU\ Loss = 1 - GIOU \tag{3.10}$$

当 A 与 B 非常远时，$(A \cup B)/C$无限接近于 0，IOU=0，GIOU 趋近于-1，当两框重合，$(A \cup B)/C$=1，IOU=1，GIOU=1。所以 GIOU 取值的区间是[-1, 1]。由于 GIOU 通过加入包含实际框和预测框的最小框，考虑到了 IOU 没有考虑到的非重叠区域，能够反应出 A 和 B 重叠的方式，能够更加准确地表示损失，因此本设计中采用 GIOU 来表示预测时的交并比。

最终本文对 YOLOv3-tiny 网络做出的修改如下：

（1） 使用 GIOU 替换了 IOU；

（2） 使用 0.5 MobileNet 替换了原本的主干网络；

（3） 使用 LeakyReLU 替换了 MobileNet 中的 ReLU 激活函数。

3.2.3 网络训练

本文基于 Pytorch 计算框架，使用 Intel i7-10750H CPU 和 Nvidia RTX2060 GPU 的硬件平台，使用 VOC2007 和 VOC2012 作训练集，共包含 16551 幅图像和 40025 个物体；VOC2007 作验证集共包含 4952 幅图像的 12032 个物体。PASCAL VOC 数据集是目标检测领域通用数据集，共包含 20 类已经标注好的对象。图 3.10 展示了 VOC 数据集中的九张例图。

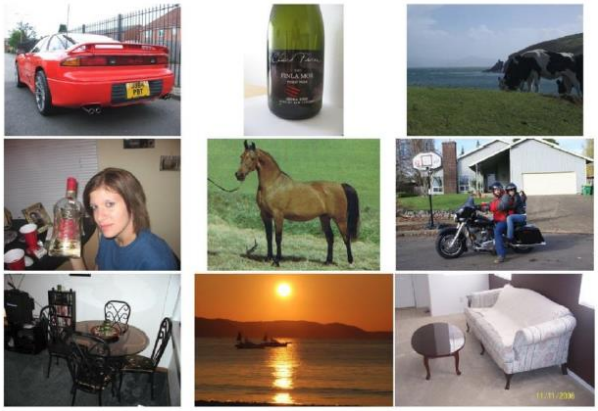


图 3.10 VOC 数据集样例图像

在神经网络训练完成后，需要一个指标来衡量该网络模型的精度，由于 VOC 数据集中一张图像里往往有着不止一类待检测物体，因此使用单一的指标很难衡量模型的精度，因此在这里引入了 TP(True Positive)、FP (False Positive)和 FN（False Negatives）三个参数，分别表示成功检测出正确的目标数，将错误的目标检测为正确的目标数，将正确的目标检测为错误的目标数。又定义了精确度 Precision 和召回率 recall 两个参数，分别定义如式 3.11 和式 3.12 所示。

$$Precision = \frac{TP}{TP + FP} \tag{3.11}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.12}$$

根据准确率以及召回率可以得到 P-R 曲线，如图 3.11 所示。每一类别都会有自己的 P-R 曲线，P-R 曲线与坐标轴围成的面积称为 AP(Average Precision)，代表着该类别的平均精度。而 mAP50 就是所有类别 IOU 阈值大于 0.5 时所有类别 AP 值的平均。本文使用 mAP50 这个参数来衡量检测模型的精度。

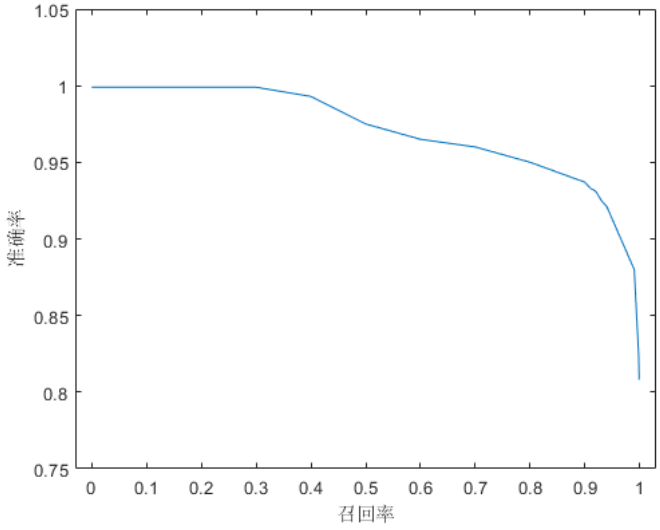


图 3.11 P-R 曲线

本网络在训练过程使用了 0.5 MobileNet 的预训练模型，采用冻结训练的策略，先冻结训练 50 个 epoch，一个 Batch 输入的训练集数目为 16，50 个 epoch 后解冻训练，一个 Batch 输入的训练集数目为 8，采用 SGD 优化器和余弦退火策略调整学习率。训练过程中的 mAP50 随迭代次数的变化如图 3.12 所示。

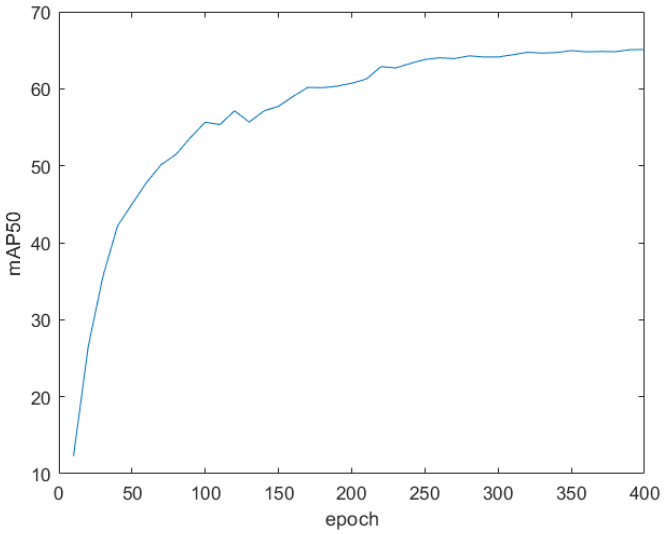


图 3.12 模型 mAP50 随训练 epoch 的变化

可以看出随着训练的进行，模型的 mAP50 呈不断上升的趋势，但受限于硬件与时间，本模型训练到 400 个 epoch 时停止，可以看出当 epoch 到 400 时，mAP50 的增长趋势已经相当缓慢，几乎达到了最高点。因此可以认为模型已经达到了理想的精度。为了证明本模型的有效性，将对本模型的精度和其它网络进行对比，其参数如表 3.1 所示。

表 3.1 本文提出的网络和其它网络的参数对比

网络模型	参数量	mAP50
YOLOv3	61.58M	79.2%
MobileNet-YOLOv3	24.65M	73.3%
YOLOv3-tiny	8.74M	61.5%
本文提出的 YOLO 网络	12.26M	65.9%

可以看出，本文提出的 YOLO 网络的参数量远小于原版的 YOLOv3 网络和使用 MobileNet 作为主干网络的 MobileNet-YOLOv3 网络，尽管 mAP50 略有下降，相对于参数量的缩减是可以接受的。而相比于原版的 YOLOv3-tiny，本文提出的网络 mAP50 也有一定提升，但是参数量也大于原版的 YOLOv3-tiny，因此为了进一步的轻量化，还需要对模型做一些修改。

3.3 模型轻量化方法

3.3.1 模型剪枝

卷积神经网络的复杂模型使得在实际应用中往往会受到计算资源和存储空间的限制，导

致难以直接部署到边缘设备或移动端。然而，卷积神经网络模型中存在许多冗余的神经元和权重，并且只有很少的参数对模型产生实际影响。因此，通过剪枝可以去除模型中不必要的参数以压缩模型。模型剪枝是最常用的压缩方法，它不仅可以降低模型复杂度、防止过拟合，还可以减小模型大小、提高模型速度。剪枝有三种方式：通道剪枝、权值剪枝和层剪枝。层剪枝虽然效果较差，但灵活度高，不需要特殊的软件或硬件支持；权值剪枝具有最高的灵活性，但需要软件和硬件支持，以加速特殊的矩阵运算；通道剪枝具有层剪枝和权值剪枝的优点，非常精细且灵活。图 3.13 展示了通道剪枝前后的网络结构示意图。

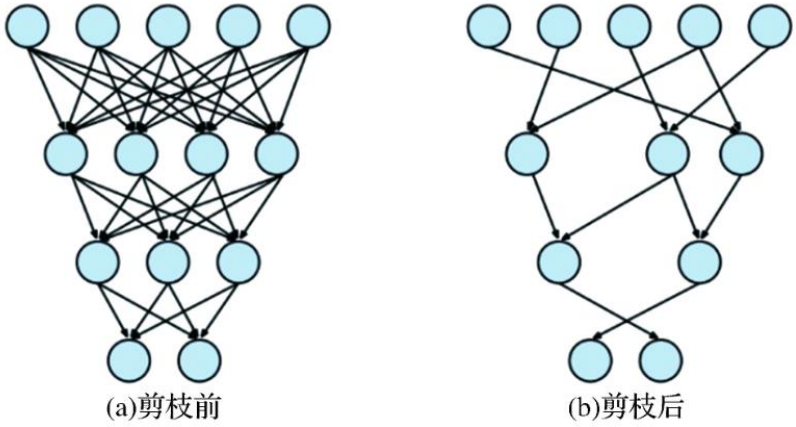


图 3.13 通道剪枝示意图

本文使用了基于相邻层权重的卷积神经网络裁剪方法，由于对于卷积神经网络中的一次卷积，过程如图 3.14 所示，如过有 n_i 个大小为 w_i*h_i 的特征图 X_i 和 n_{i+1} 个通道数为 n_i 的卷积核进行卷积，得到 n_{i+1} 个特征图 X_{i+1}，而这 n_{i+1} 个特征图又将和 n_{i+2} 个通道数为 n_{i+1} 的卷积核进行卷积，得到 n_{i+2} 个特征图 X_{i+2}。如果裁剪掉一个卷积核，那么输出的特征图数量将减少一个，对应的，下一层的卷积核也会减少一个通道[56]。

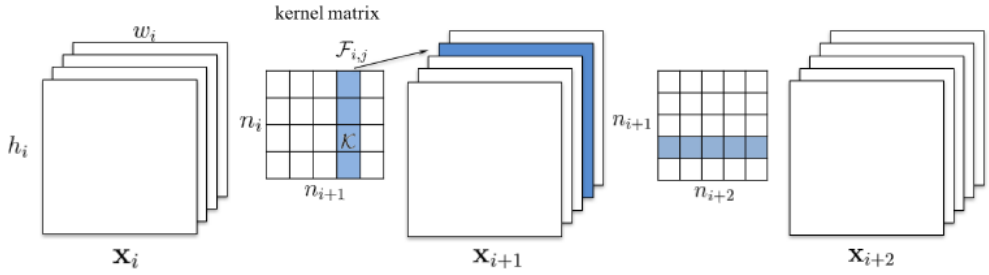


图 3.14 卷积过程

读取神经网络模型中所有卷积核的权重数据，计算层内每个卷积核 $F_{i,j}$ 的权重 L1 范数值 L_A 与下一层对应通道 $G_{i,j}$ 的权重 L1 范数值 L_B，表达式分别如式 3.13 和式 3.14 所示。

$$L_A = \sum |F_{i,j}| \tag{3.13}$$

$$L_B = \sum |G_{i,j}| \tag{3.14}$$

由于特征图与卷积核进行卷积时，一个卷积核会与所有的特征图进行卷积，而一张特征图会与不同卷积核的一个通道进行卷积。卷积核 $F_{i,j}$ 做的卷积运算为 n_i 次，其产生的一张特征图接下来所作的卷积运算为 n_{i+2} 次。因此在衡量卷积核的重要性时将卷积核的权重值之和与其所作卷积运算次数相乘，再加上下一层卷积核对应通道的权重之和与其所作卷积运算之积，表示为 I，表达式如式 3.15 所示。

$$I = L_A * n_i + L_B * n_{i+2} \tag{3.15}$$

再根据 I 的大小对每一层内的卷积核进行排序，令模型的剪枝率为 λ，将 I 处于后 λ 的卷积核都删除，同时剪掉下一个卷积层中与被删除的特征图进行卷积的通道。对网络的每一层都重复以上步骤，直至最后一层，得到需要的模型。

在直接剪枝后神经网络模型往往精度都会大幅较低，因此还需要使用原数据集对裁剪后的模型进行再训练，完成模型的微调以恢复精度。剪枝的流程图如下图 3.15 所示。

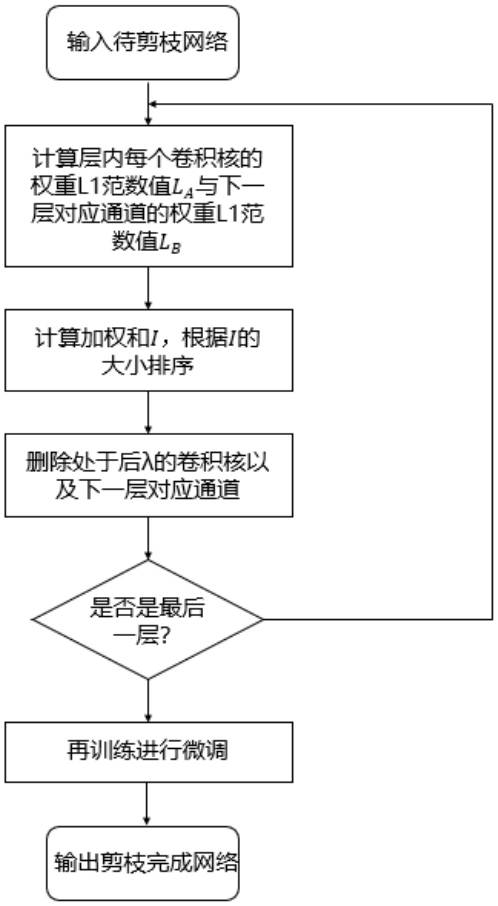


图 3.15 剪枝流程图

为了测试模型的最佳剪枝率，将上述方法应用到本文提出的轻量化 YOLO 网络模型，剪枝完成后，对不同剪枝率下的模型通过 50 个 Epochs 的微调，表 2 为不同剪枝系数的通道剪

枝结果对比。由表 3.2 可见，当剪枝率在 80%时效果较好，经过剪枝后模型的参数量由 12.26M 减少到 1.27M，减掉约 89%，mAP50 由 65.9 降低到 63.5，相对下降约 3%，如果继续裁剪，那么 mAP50 指标将会剧烈下降。因此本文选取以 80%的剪枝率对网络模型进行剪枝，本文的模型在剪枝完成后无论是参数量还是 mAP50 指标都优于 YOLOv3-tiny 网络。

表 3.2 不同剪枝率下模型的 mAP50 参数

剪枝率	参数量	mAP50
0	12.26M	65.9
10%	7.14M	65.7
20%	4.93M	65.4
30%	3.68M	65.5
40%	2.59M	65.1
50%	2.05M	64.9
60%	1.73M	64.3
70%	1.44M	64.1
80%	1.27M	63.5
90%	1.15M	59.8

3.3.2 BN 层融合

在训练深度网络模型时，BN 层能够加速网络收敛，并且能够有效解决梯度消失与梯度爆炸问题。虽然 BN 层在训练时起到了积极作用，然而，在网络前向推理 BN 层的存在也会引入了多余的运算，影响了模型的推理速度，而且消耗更多的存储空间。目前，很多先进的网络模型都使用了 BN 技术，本文提出的网络中也大量使用了 BN 层，因此，将 BN 层的参数合并到卷积层中的权重中是非常有必要的，BN 层融合后最终只会保留一个偏置，这样就可以在不会对模型的精度造成影响的前提下提升模型前向推断的速度，其计算式可以表示为：

$$x_{out} = \frac{\gamma(x_{conv} - \mu)}{\sqrt{\sigma_i^2 + \varepsilon}} + \beta \tag{3.16}$$

x_{conv} 表示由卷积层输入到 BN 层的特征图，μ 和 σ 分别表示特征图节点的均值和方差，ϵ 为很小的偏移量。γ 和 β 分别表示缩放因子和偏置值，是需要网络训练的参数。x_{conv} 的计算式为：

$$x_{conv} = x_{in} \times w + b \tag{3.17}$$

其中 x_{in} 表示卷积层的输入，w 表示卷积核的权值。

卷积层和 BN 层进行合并，用以加速网络的推理过程，合并计算式如下所示:

$$x_{out} = \frac{\gamma w}{\sqrt{\sigma_i^2 + \varepsilon}} + \beta + \frac{b - \gamma\mu}{\sqrt{\sigma_i^2 + \varepsilon}} \tag{3.18}$$

则将卷积层和 BN 层合并后，新的卷积计算方法为：

$$x_{out} = x_{in} \times w' + \beta' \tag{3.19}$$

其中

$$w' = \frac{\gamma w}{\sqrt{\sigma_i^2 + \varepsilon}} \tag{3.20}$$

$$\beta' = \beta + \frac{b - \gamma\mu}{\sqrt{\sigma_i^2 + \varepsilon}} \tag{3.21}$$

通过将 BN 层的权值与卷积层的计算融合，可以在前向推理过程中避免执行复杂的计算，如求平方根，只需使用卷积层的原始权值和偏置就可以获得同等计算效果，从而提高网络的推理速度，增强实时性，同时减少了硬件计算和资源占用，而不会损失模型精度。卷积与 BN 层融合前后的结构分别如图 3.16 和 3.17 所示，可以看出通过 BN 层融合，可以有效简化卷积神经网络的推理运算结构。

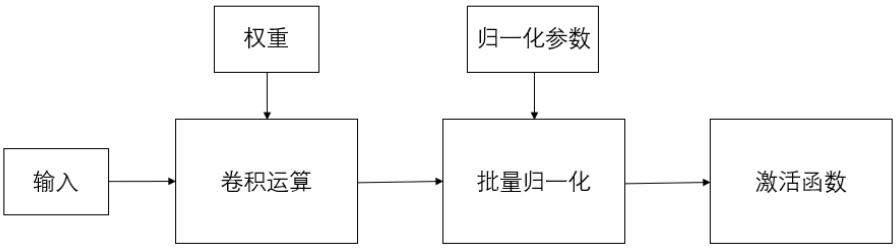


图 3.16 卷积与 BN 层未融合时的结构

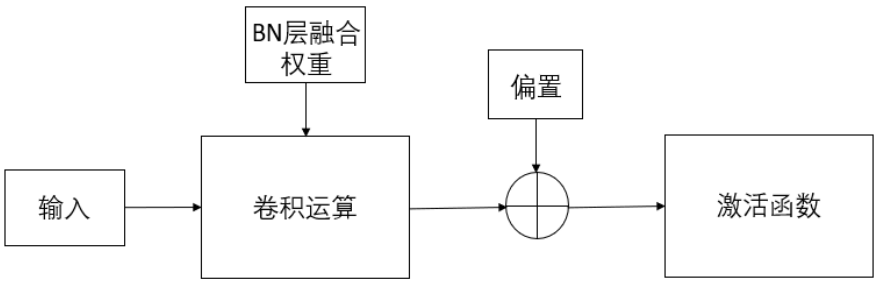


图 3.17 卷积与 BN 层融合后的结构

3.3.3 模型量化

当前大多数神经网络训练框架都需要使用浮点数进行训练，本文也使用了 32 位浮点数训练模型，但是在 FPGA 中实现浮点数的乘法较为复杂，因此往往需要将浮点数转换成定点数进行计算，而且使用定点数计算也能有效降低资源。为此，需要选择适当的方法对参数进行量化。对于参数的定点位数，由于卷积参数主要参与乘加运算，在 FPGA 中，乘法器和加法器主要耗费 DSP 和 LUT 资源。表 3.3 显示在分别使用定点数和浮点数的情况下，FPGA 实

现乘法器和加法器时资源使用的对比。由于卷积参数的定点化位数需要小于乘法器位宽，因此考虑将参数转换为 16 位或 8 位的定点数。

表 3.3 不同精度乘法器和加法器需要的 DSP 和 LUT 数量

器件	需要的 DSP	需要的 LUT
32 位浮点加法器	2	214
32 位浮点乘法器	3	135
16 位定点加法器	0	47
16 位定点乘法器	1	101
8 位定点加法器	0	25
8 位定点乘法器	1	87

由上表可以得知，当数据从32位浮点量化为16位定点或8位定点数时，DSP和LUT均有着大幅的降低，16位定点数的加法器不需要DSP，LUT数量为32位浮点数加法的约20%，16位定点数的乘法器需要的DSP仅为一个，LUT数量为32位浮点数加法的约75%；8位定点数的加法器同样不需要DSP，LUT数量为16位定点数加法的约53%，8位定点数的乘法器需要的DSP与16位定点数相等仅为一个，LUT数量为16位定点数加法的约86%。因此可以得出结论，无论是采用16位或8位定点数量化，卷积计算所需要的计算资源都会大幅降低。但是考虑到使用FPGA加速卷积神经网络时硬件资源的主要压力来自于DSP而非LUT，而且8位定点量化会带来更大的量化误差导致卷积神经网络模型精度下降，因此本设计中采用16位定点量化对网络进行量化。

本文主要采用了均匀量化的方法，这种量化算法被广泛应用于各种边缘设备。其原理如下：

假设要量化网络的其中一个卷积核，量化精度为n bit，首先计算出此卷积核中的权重的最大值和最小值R_{max}和R_{min}，那么量化的线性变换因子S就可以表示为：

$$S = \frac{R_{max} - R_{min}}{2^n - 1} \tag{3.22}$$

而量化的计算式可以表示为：

$$Q = round\left(\frac{R}{S} - Z\right) \tag{3.23}$$

其中，Q是量化后的定点数值，R是量化前的浮点数值，Z是原值域中零点量化后对应的值，即零点偏移量。需要注意的是，在进行了缩放和取整运算之后由于小数位可能会向上取整，故取整后的值可能会超出规定的量化上界，因此还需对超界值进行截断，使其等于量化上界值。

针对卷积神经网络中的每一层都使用上述的模型量化方法，最终即可得到量化后的模型。

经过上面提到的方法对模型进行量化后，模型的 mAP50 从 63.5%降低到了 63.4%，可以

得出结论，本文提出的轻量化的 YOLO 网络模型经过 16bit 量化后，mAP50 几乎不变，仍然有着较好的检测效果。

3.4 本章小结

本章针对 YOLO 系列网络模型参数太多难以部署的问题，以 YOLOv3-tiny 为基础，进行了替换主干网络，IOU 和激活函数的改动，提出了较高精度且轻量化的 YOLO 网络。然后又对模型进行了剪枝，BN 层融合和量化操作，进一步对模型轻量化，使其可以部署到硬件加速器中。

第四章　硬件加速器设计

在完成了轻量化网络模型设计后，还需要设计对应的硬件以完成边缘端加速。本章首先介绍了高层次综合技术，针对上一章设计的卷积神经网络设计了加速器总体架构。然后具体设计了访存、计算、上采样和Arm端后处理实现部分，并应用了一系列优化策略，最终完成了硬件加速器设计。

4.1 高层次综合技术

高层次综合技术（high-level synthesis，HLS）是一项综合技术，可将高级语言描述的行为自动转换为具有相同功能的硬件。因此这种技术也被称为行为级综合，可以让开发者在更高的抽象层次上定义电路，从而大大降低硬件设计的难度，缩短硬件开发周期。因此，高层次综合技术的目标是让开发者更多地关注算法设计和硬件架构，而非过多关注硬件的细节实现[57]。

Vivado HLS是由Xilinx公司提供的HLS工具，可通过HLS直接产生综合后的电路并简单部署在FPGA开发板上。在Vivado HLS软件中，可使用三种语言进行设计和开发，包括C、C++ 和System C。使用Vivado HLS进行算法开发时，可在抽象层工作，以缩短开发周期。相较于传统的硬件描述语言，高级语言验证算法的速度更快。通过优化指令控制的高级综合技术，可建立具体的高性能硬件实现。通过优化方案，可创建多个实现，并探索设计空间以找到最佳实现。同时，高级代码具有可读性和可移植性，可在不同的设备上实现。Vivado HLS还提供了一系列优化指令，如循环展开、循环流水化、内存划分等，可用于优化指令。通过这些优化指令，HLS可以大幅提高并行度和生成硬件的性能，从而缩短与手动设计的硬件的差距。

(1) PIPELINE指令

该指令通常在for循环中使用，用于指示编译器对循环体内的代码进行流水线处理。在最佳情况下，流水线间隔（Initiation Interval，II）为1，即每个cycle每个运算单元可以完成一个操作。实际上，II是否优化到1取决于代码中的数据依赖性和并行性。

(2) UNROLL指令

该指令将for循环中的循环体展开，硬件上相当于将循环体复制了循环次数N份，每个循环体同时执行。

(3) Array Partition指令

该指令通常用于对数组进行操作，将数组按照指定要求分割成多个小的数组，形成多个小的存储空间。在C语言中，int a[N]表示声明一个包含N个元素的数组a。而在HLS中，这表

示在片上BRAM中为a分配N个元素的空间。但是可以通过Array Partition指令来指定这N个元素在FPGA中的存储方式。默认情况下，a中的元素是连续存储在一个BRAM中的，这意味着每个周期只能提供一个数据，而在实际应用中往往都需要很大的片上带宽，因此需要使用该指令进行优化。

4.2 总体实现架构

通过对网络结构的分析，本文提出的轻量化YOLO网络中的计算主要包括卷积运算、激活运算、上采样运算和Concat运算四个主要部分，而卷积运算又包括标准卷积，深度卷积和点卷积三种不同的卷积。

卷积神经网络中卷积计算占总计算量很大部分的比例，因此需要重点加速，单独设计对应的硬件模块并进行优化。对于其它运算，Concat运算可以通过预设内存存取地址实现，无需设计硬件逻辑，同时也不占用额外存储资源，而激活和上采样运算的计算量相对于整个网络来说较小，因此可以通过简单的代码实现。

在本加速器中，特征图和网络权重存储在外部DDR中，数据交互主要由AXI总线实现。数据和权重首先通过AXI总线读取并缓存在数据buffer和权重buffer中，然后从buffer读取数据和权重输入卷积运算模块进行计算并激活，或直接进行上采样运算，运算完成后将输出结果缓存在输出buffer中。最后，通过AXI总线写回DDR。

该加速器的架构如图4.1所示。

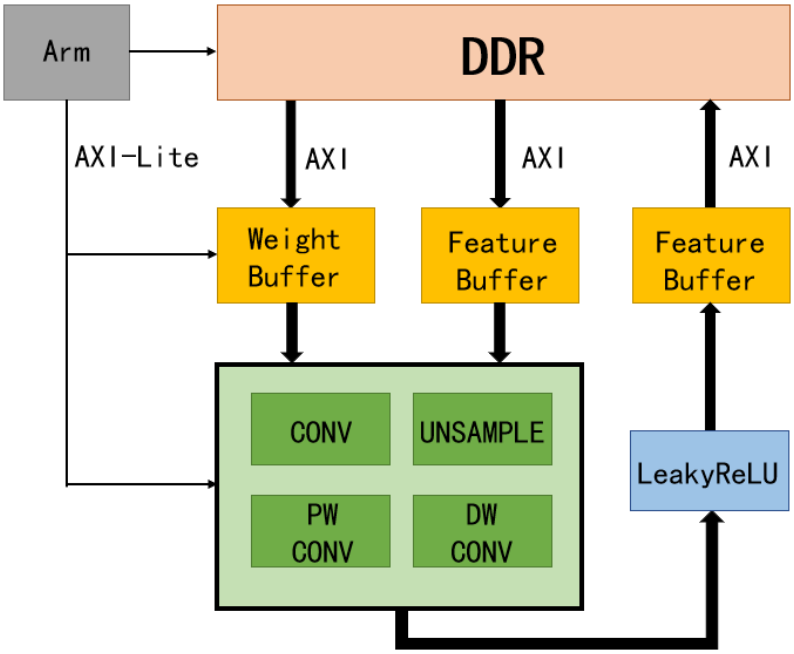


图 4.1 本文硬件总体架构

本文加速器计算时的数据流图如图 4.2 所示：

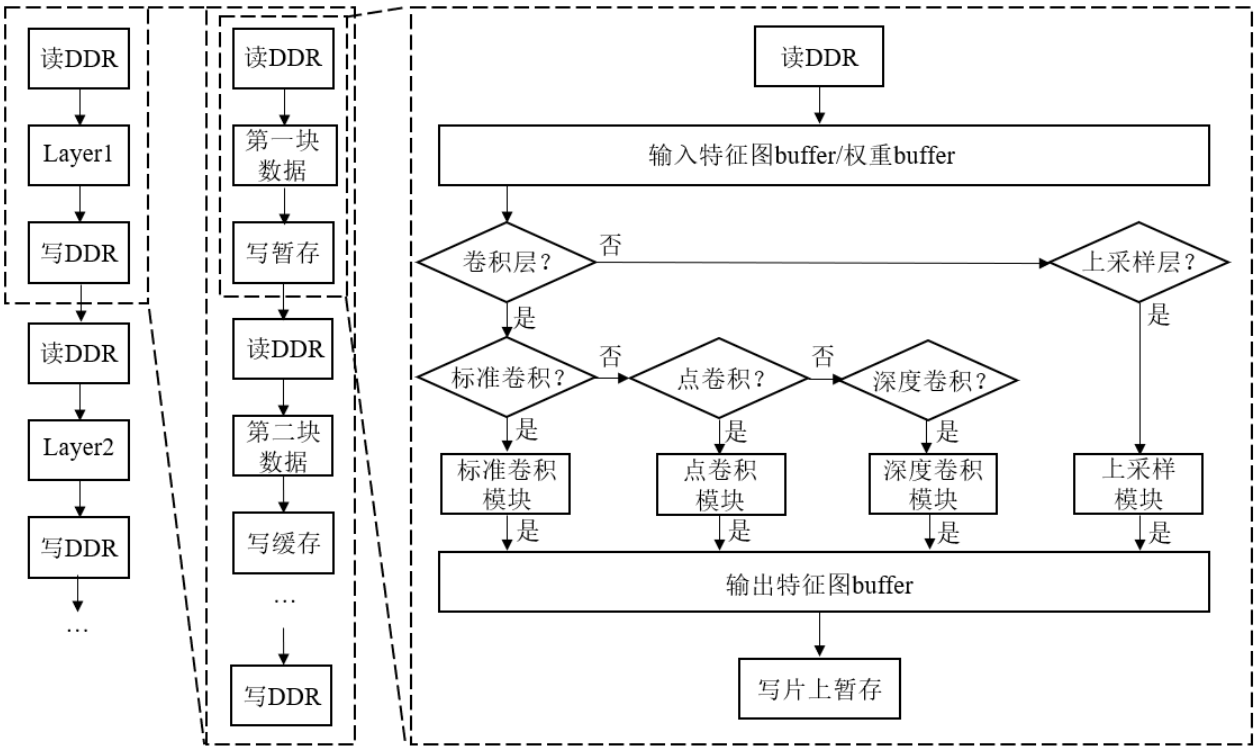


图 4.2 本文加速器计算数据流图

本文加速器在进行卷积神经网络推理运算时分层进行，当第一层网络计算完成后接着计算第二层网络，而在每一层内部也采用了分块卷积的策略逐块进行运算。加速器从外部 DDR 中读取输入特征图和卷积权重至对应的 buffer 后，首先判断进行何种运算，如果是卷积运算，此时根据卷积运算的种类将数据输入进对应的计算模块中，卷积计算完成后，接着进行激活运算，对数据进行非线性处理；如果是上采样运算，将数据输入进上采样模块中，完成后将输入写入输出缓存，再存入外部 DDR，等待下一次读取。

4.3 硬件接口设计

在本加速器的设计中，加速器的接口使用了AXI4协议。AXI4协议是ARM推出的第四代AMBA接口规范，AXI总线是一种高性能、高带宽、低延迟的片内总线，根据应用场合的不同，AXI4协议可分为三种子协议：

（1） AXI4接口协议：

AXI4协议是用于片上总线的一种高性能、低功耗的标准协议，由ARM公司提出。该协议支持多主、多从的系统结构，可以实现高效的数据传输和复杂的系统控制。AXI4协议主要包含四个通道：读通道、写通道、读写地址通道和内存映射通道，其中读通道和写通道分别用于从内存或外设读取数据和向内存或外设写入数据，读写地址通道用于发送读或写操作的地址信息，内存映射通道用于发送控制信息和其他系统信息。此外，AXI4协议还提供了

许多可选的特性，如QoS、缓存一致性和交错传输等。

（2） AXI4-Lite 接口协议：

AXI4- Lite协议是AXI4协议的一种简化版本，它不支持乱序传输和缓存一致性，并且没有读写地址通道，只有一个共享的地址信道。AXI4- Lite协议适用于实现简单的读写操作，如寄存器或控制寄存器的读写。

（3） AXI4-Stream接口协议：

AXI4-STREAM协议是一种轻量级的流式数据传输协议，用于片内的高速数据流传输。该协议不需要地址信息，只需传输数据本身和一些与数据相关的控制信息，例如时钟使能、数据有效和传输大小等。AXI4-STREAM协议的特点是高带宽、低延迟和可靠性好，适用于处理音频、视频和图像等高速数据流。

本文的加速器对外接口设计有12个AXI Master接口和1个AXI-Lite Slave接口。其中，卷积核权重参数的读取由4个AXI Master接口负责，4个AXI Master接口负责特征图的读取，4个AXI Master接口负责特征图的写入，设计多个接口并行读取以提高效率。AXI-Lite Slave接口负责读写加速器的寄存器，确认加速器状态。

4.4 访存部分设计

本文的加速器在对外接口和卷积计算单元间设计了访存部分，主要负责将片外的输入特征图数据读到片上缓存和将输出缓存的输出特征图数据写回片外存储。为此，本文设计了循环分块，双缓存以及流水化三种策略对数据的访存进行优化。

4.4.1 循环分块策略

由于FPGA片上存储资源(BRAM)非常有限，Zynq7020芯片仅有140个大小为36KB的BRAM，而卷积神经网络的权重和中间计算结果特征图都具有较大的存储需求，尤其是为了获得一定的计算并行度，必须采取措施来增大片上缓存区的带宽，这将导致更多的片上缓存资源消耗，远远超出芯片内部存储空间的大小。缓存整个特征图显然是不现实的，因此大量的数据必须存储在片外存储器中。然而，频繁访问片外存储器将带来大量的时延，极大地降低系统性能。

为了解决这个问题，本设计采用循环分块策略，将卷积层的输入数据分块，使每个块的大小都小于片上存储空间。通过卷积分块，增加了数据的局部性，同时减少了对片外存储器的访问次数，从而降低访问时延，提高了系统加速性能。具体措施如下。

假设输入特征图I为$N\times H_{in}\times W_{in}$，权重为$M\times N\times K\times K$，输出特征图O为$M\times H_{out}\times W_{out}$，有边缘填充。令输入通道、输出通道、输入特征图高、输入特征图高宽、输出特征图高、输出特

征图宽的分块因子分别为T_n，T_m，T_{ri}，T_{ci}，T_{ro}，T_{co}，又由于在本网络中同时存在步长为1和2的卷积，所以需要分别进行说明，令步长为S。则在分块卷积时，输入块的大小和输出块的大小之间的关系可由式4.1和式4.2表示。图4.3和图4.4分别展示了步长为1和步长为2的分块卷积。

$$T_{ri} = S \times T_{ro} + K - S \tag{4.1}$$

$$T_{ci} = S \times T_{co} + K - S \tag{4.2}$$

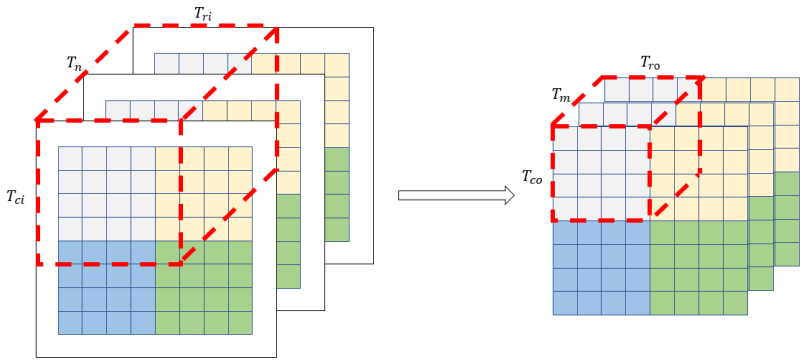


图 4.3 步长为 1 的分块卷积

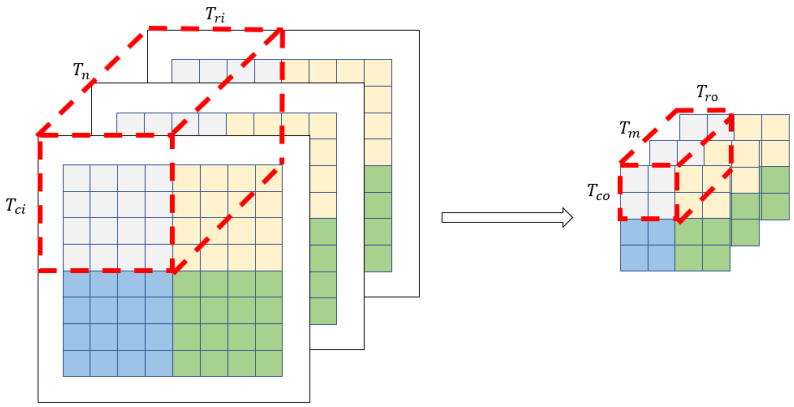


图 4.4 步长为 2 的分块卷积

则每次计算的时候，可以首先加载 $T_n \times (S \times T_{ro} + K - S) \times (S \times T_{ro} + K - S)$大小的输入特征图块，$T_m \times T_n \times K \times K$ 大小的权重，然后进行卷积计算，得到 $T_m \times T_{ro} \times T_{co}$ 大小的输出特征块，当这块输入特征和权重计算完成后，再读取下一块输入特征和权重，再进行计算。在计算过程中，把中间结果先暂存在片上存储中，当计算结束，再将最终结果存入片外存储，如图 4.5 所示。

图 4.5 分块卷积流程

使用循环分块策略后输入特征图像，输入卷积权重和输出特征图像理论消耗 BRAM 数量分别如式 4.3，4.4，4.5 所示：

$$N_{IF} = \frac{T_n \times T_{ri} \times T_{ci} \times datawidth}{C_{\mathrm{BRAM}}} \tag{4.3}$$

$$N_{Iw} = \frac{T_n \times T_m \times K \times K \times datawidth}{C_{\mathrm{BRAM}}} \tag{4.4}$$

$$N_{OF} = \frac{T_m \times T_{ro} \times T_{co} \times datawidth}{C_{\mathrm{BRAM}}} \tag{4.5}$$

其中 N_{IF} 表示输入特征图像，N_{IW} 表示输入特征图像，N_{OF} 表示输入特征图像。datawidth 表示数据位宽，由于本文网络模型使用了 8bit 量化，因此该参数值等于 8。C_{BRAM} 表示一块 BRAM 的存储能力，本文选取的 Zynq7020 芯片中该值等于 36Kb。在实际应用中，由于 BRAM 的读写接口数量有限，当加速器需要同时在不同缓冲区读取和写入数据时，需要将多个独立的子缓冲组合成一个较大的缓冲区，这会增加对 BRAM 资源的消耗。另外，使用下节介绍的双缓存模块技术也会对 BRAM 资源使用产生影响。

4.4.2 双缓存模块

除了循环分块计算外，本设计还使用了双缓存模块来加速硬件的访存效率，由于在网络运行中每一层的计算都需要频繁的对片外 DDR 读写数据，这将给加速器带来延迟导致吞吐率下降。为解决这一问题，使用双缓存模块进行乒乓操作以掩盖数据的传输时间。双缓存设计具体的结构如图 4.6 所示。

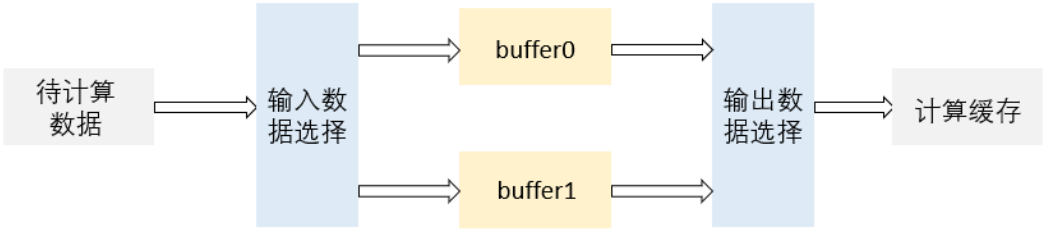


图 4.6 双缓存模块设计

在卷积神经网络加速器中，片上缓冲一般有三部分：（1）输入缓存区，用来储存输入特征图数据（2）权值缓存区，用来存储权重（3）输出缓存区，用来储存中间结果和最终输出特征图数据。双缓存单元在这三个缓冲区均被应用。

下面以数据从DDR输入计算模块为例，数据输入的时延主要可以分为三部分：计算地址偏移时延T_{addr}，从DDR读取数据到特征buffer时延T_1，以及从特征buffer读取数据到待计算的缓存单元时延T_2，如果三项操作采用顺序串联的执行方式，那么读取一个特征图块需要的总时延T_{input1}可以表示为：

$$T_{input1} = T_{add} + T_1 + T_2 \tag{4.6}$$

但是由于我们引入了双缓存模块，可以在从DDR读取数据到特征buffer0的同时将特征buffer1的数据传输到待计算缓存单元，实现DDR读取数据到特征buffer时间与读取数据到待计算的缓存单元时间的交叠。计算流程如图4.7所示：

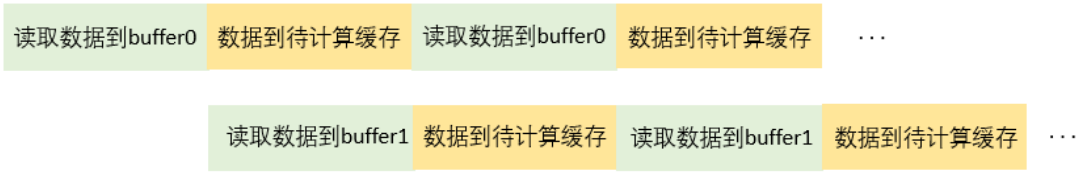


图 4.7 读取数据到 buffer 和读取数据到待计算缓存时间交叠

那么此时读取一个特征图块需要的总时延T_{input2}如式4.7所示：

$$T_{nput2} = \max(T_{addr}, T_1, T_2) \tag{4.7}$$

对于权值缓存区和输出缓存区，本设计也都采用了和输入缓存区一样的双缓存单元设计。通过占用额外的片上存储资源，双缓存设计可以有效降低数据传输时延，进而提高加速器的性能。

4.4.3 流水化处理

除了采用双缓存单元来加速硬件的访存效率外，本文还采用了流水化的思想来优化卷积运算过程。卷积运算时包括输入特征图像缓存，输入卷积权重缓存，卷积计算和输出特征缓存四个步骤，如果不采用流水化的思想，上述四个步骤顺序进行，那么完成一次卷积需要消耗的时间T_{CONV1}为：

$$T_{CONV1} = \max(T_{IF}, T_{IW}) + T_C + T_{OF} \tag{4.8}$$

其中T_{IF}为输入特征图像缓存的时间，T_{IW}为输入卷积权重缓存的时间，T_C为卷积计算的时间，T_{OF}为输出特征缓存的时间。由于输入特征图像和输入卷积权重各自有专用的缓存，因此这两项操作可同时进行。

而如果采用流水化处理的方法，完成一次卷积需要消耗的时间T_{CONV2}为：

$$T_{CONV2} = \max(T_{IF}, T_{IW}, T_C, T_{OF}) \tag{4.9}$$

图4.8展示了计算时间与数据传输时间的交叠，可以在同一个时钟周期内进行不同次数卷积的独立步骤。

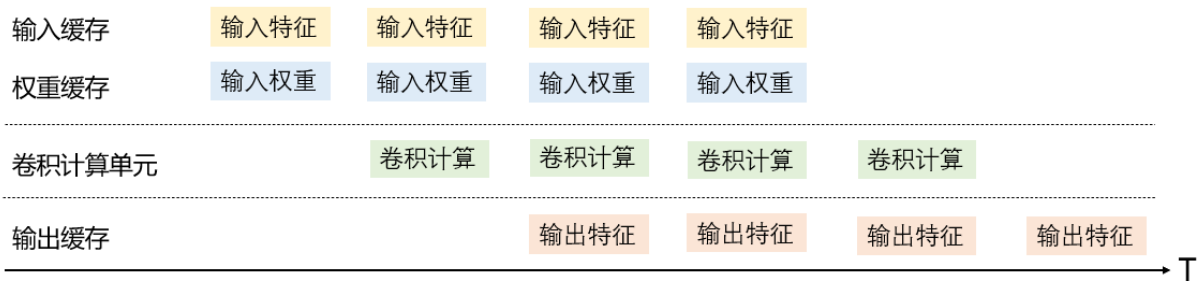


图 4.8 卷积运算流水过程

在计算一个卷积层时，首先传输第一组输入特征图和第一组权重进入特征缓存区和权值缓存区，传输完毕后可以开始卷积计算。在卷积计算的过程中可同时完成第二次输入特征图和权重，从而达到权重输入时间与卷积计算时间的交叠。卷积计算中产生的结果可经由输出缓存单元送出，实现特征图输出时间与卷积计算时间的交叠。依次重复之前的操作，实现卷积计算与数据输入输出的流水处理。

4.5 计算部分设计

卷积运算通常需要大量的计算量，并行计算可以大幅度提高卷积运算速度，因此在本节中首先对卷积的计算并行性进行分析。当前卷积神经网络并行计算主要有四个维度：输入通道并行、输出通道并行、卷积窗口内并行，不输出像素之间的并行。这四个维度的并行并不冲突，在实际应用中可以根据不同加速器架构的计算资源和存储资源，组合多种并行计算方案以提高系统整体的并行度[59]。

(1) 输入通道并行计算

输入通道并行是同一个卷积窗口与不同卷积核并行，同时计算多个输入通道的特征图和权重的卷积。在卷积运算过程中，多个输入的运算是独立的。因此，在同时对多个输入进行并行的卷积运算时，可以提高运算效率。

(2) 输出通道并行计算

输出通道并行可以同时计算多个输出特征图的结果或部分和，卷积层内不同特征图计算并行。这样，在卷积运算过程中，多张特征图中的卷积窗口运算是独立的，因此可以多个特征图的卷积窗口与同一个卷积核同时进行卷积运算。

(3) 卷积窗口内并行计算

即卷积窗口内K×K个神经元和权重之间乘法的并行。一个卷积窗口内神经元节点间计算并行。例如使用3×3的卷积窗口对特征图进行卷积运算时，卷积窗口内的神9个经元节点运算是独立的。如果在单个时钟周期内，9个神经元节点可以同时运算那么相比起每个神经元依次运算的效率就提升了9倍。

(4) 输出像素间并行计算

即不同卷积窗口间并行，同时计算输出特征图上多个神经元。在一张特征图内，卷积窗口不断滑动与卷积核进行卷积运算，滑动过程中的每一次运算都是独立的，因此可以将不同位置的窗口进行并行运算，也能提高卷积效率。

本网络中包含普通卷积，深度卷积和点卷积三种卷积运算方式，每种卷积都有自己独特的运算特性，因此需要针对不同的卷积设计不同的并行运算方案，以达到更好的并行度。在

本加速器的设计中，我们采用输入通道并行和输出通道并行这两种并行方式。

4.5.1 标准卷积和点卷积模块设计

在本网络中，存在着一层标准卷积与其后深度可分离卷积中的点卷积。标准卷积核的大小为3×3，特殊的，点卷积可以视作卷积核大小为1×1的标准卷积，因此点卷积核的设计思想与标准卷积一致。为区分点卷积和标准卷积，设置了参数k来表示卷积核的大小。标准卷积和点卷积核的参数列表如下表所示。

表 4.1 标准卷积和点卷积核参数列表

参数名	参数功能
k	卷积核的大小
C_{in}	输入特征图通道数
C_{ou}	输出特征图通道数
H_{in}	输入特征图高度
W_{in}	输出特征图宽度
S_r	卷积核在高度方向的步进
S_c	卷积核在宽度方向的步进
P_r	输入特征图左右补零列数
P_c	输入特征图左右补零列数

从标准卷积和点卷积的配置参数表中可以看出，不仅需要输入特征图的通道深度、高度和宽度来确定输入特征图的大小，还需要通过卷积核的大小K来配置到底是标准卷积还是点卷积，同时也以C_{out}来指示进行第几个卷积核的卷积运算。输入特征图通道数等于卷积核的通道数而卷积核的个数等于输出特征图通道数。另外，输出特征图的宽度、高度则需要输入特征图的宽度和高度、卷积核步长、宽度和高度以及特征图是否填充等参数确定，其宽高的计算式分别如式4.10和4.11所示。

$$H_{out} = \frac{H_{in} - k + 2 \times P_r}{S_r} + 1 \tag{4.10}$$

$$W_{out} = \frac{W_{in} - k + 2 \times P_c}{S_c} + 1 \tag{4.11}$$

卷积模块的资源消耗主要由乘法器、加法器、MUX 和累加器组成。对FPGA而言，主要消耗的资源是DSP和LUT。卷积模块消耗的DSP 资源N_{DSP} 和消耗的LUT 资源N_{LUT} 的计算公式分别为：

$$N_{DSP} = C_{in} \times C_{out} \times (N_{MUL\,DSP} + N_{ADD\,DSP}) \tag{4.12}$$

$$N_{LUT} = C_{in} \times C_{out} \times (N_{MUL\,LUT} + N_{ADD\,LUT}) + C_{out} \times N_{SEL\,LUT} \tag{4.13}$$

上式中$N_{MUL\,DSP}$，$N_{MUL\,LUT}$，$N_{MUL\,LUT}$，$N_{ADD\,LUT}$分别对应表3-3中乘法器和加法器消耗的DSP和LUT资源，$N_{SEL\,LUT}$表示MUX和累加器的LUT资源消耗，约为32。

在确定了卷积核的配置参数后，要进行卷积核的优化设计。针对标准卷积和点卷积，可以采用输入通道并行和输出通道并行相结合的设计，伪代码表示如图4.9所示。

```
for(kr=0;kr<K;kr++)
for(kc=0;kc<K;kc++)
for(r=0;r<Hout;r++)
                for(c=0;c<Wout;c++)
                    for(co=0;co<Cout;co++)
#pragma HLS unroll factor= Tn
                        for(ci=0;ci<Cin;ci++)
#pragma HLS UNROLL factor= Tm
                            out[co][r][c]+=In[ci][r+kr][c+kc]*w[co][ci][kr][kc];
```

图 4.9 标准卷积和点卷积伪代码

上图的描述的硬件可以综合出一个累乘加树结构，包含一个并行乘法单元和一个加法树，如图 4.10 所示。

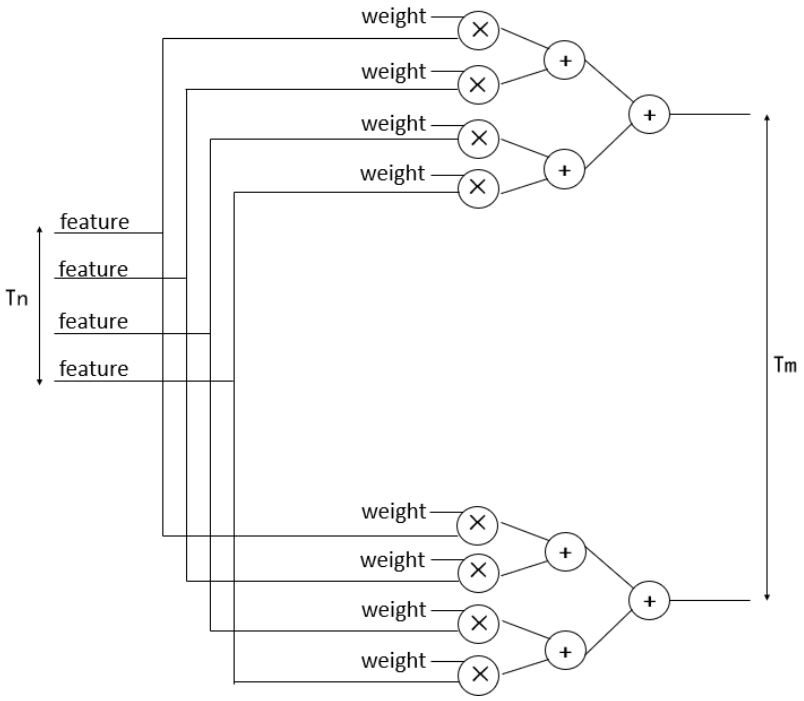


图 4.10 累乘加树结构

图4.10所示的累乘加树对卷积运算进行了二维展开，对输入特征图进行T_n路并行处理，对输出特征图进行T_m路并行处理，每个时钟周期内都将读取T_n张特征图，同时有$T_m \times T_n$个权重读入卷积核，每张特征图都将与T_m组权重进行并行的乘法运算，乘法计算完成后T_m组累加器对结果进行累加并输出卷积计算结果。卷积模块的实现核心是使用FPGA中的DSP和LUT资源进行计算，使用寄存器资源暂存数据。

因为标准卷积存在于主干网络的第一层输入卷积和网络中的检测分支部分，对于输入图

像，是标准的RGB三通道图像数据，每个图像数据通道都有4个不同的卷积窗口同时对其进行计算，因此每个时钟周期能够完成并行度为12的卷积计算。对于检测分支部分的标准卷积运算，设置4个卷积单元，每个通道有4个卷积核对其进行计算，每个时钟周期能够完成的并行度为16。

对点卷积采用和标准卷积同样的设计方法，同样设置4个点卷积单元。对于输入点卷积的四通道特征图，共有四个1×1×4的卷积核在对其进行计算，因此每个时钟周期能够完成的并行度为16。

4.5.2 深度卷积模块设计

深度卷积时，卷积核的通道数为 1，而且深度卷积的输出特征图深度等于输入特征图的深度，只是在做对应位置一对一的乘加运算，不需要进行通道累加就能得到输出特征图；而且由于算法中深度卷积核的大小均为 3*3，所以加速器的深度卷积核大小也都为 3*3，不需要额外进行配置；深度卷积的卷积核个数为N，C_{out}也为1。深度卷积核的参数如下表所示。

表 4.2 深度卷积核的参数列表

参数名	参数功能
k	卷积核的大小
C_{in}	输入特征图通道数
H_{in}	输入特征图高度
W_{in}	输出特征图宽度
S_r	卷积核在高度方向的步进
S_c	卷积核在宽度方向的步进
P_r	输入特征图左右补零列数
P_c	输入特征图左右补零列数

从配置表中可以看出，深度卷积和标准卷积与点卷积一样，需要配置输入特征图的通道深度、高度和宽度来确定输入特征图的大小，而输出特征图的宽度、高度则可通过输入特征图的宽度、高度、卷积核步长和是否填充等参数确定，但不同的是，深度卷积的输出特征图的通道数同输入特征图通道数一样，卷积核的通道数也只有1。其中卷积核的步长大小和特征图是否填充等参数还关系到数据排布方式中特征图和权值的寻址问题。

以上是对深度卷积参数的解释，接下来对深度卷积进行硬件设计。对于深度卷积，输入通道事实上就是输出通道，对深度卷积进行一维的通道展开即可，输入和输出特征图通道都进行T_n路并行，每个时钟周期内都将读取T_n张特征图，同时有T_n个权重读入卷积核，每张特征图都将与对应的一组权重进行并行的乘法运算，乘法计算完成后T_n组累加器对结果进行累加并输出卷积计算结果，伪代码表示如图4.11所示。深度卷积模块的实现核心是使用FPGA中的DSP和LUT资源进行计算，使用寄存器资源暂存数据。

```
for(kr=0;kr<K;kr++)
    for(kc=0;kc<K;kc++)
        for(r=0;r<Hout;r++)
            for(c=0;c<Wout;c++)
                for(chi=0;chi<CHin;chi++)
#pragma HLS UNROLL
                    out[cho][r][c]+=In[chi][r+kr][c+kc]*w[cho][chi][kr][kc]
```

图 4.11 深度卷积伪代码

为保持数据分块的一致性，深度卷积设定4个深度卷积单元，每次并行计算4张特征图，因此每个时钟周期能够完成并行度为4的卷积计算。

4.6 上采样层设计

网络中只有一层上采样层，功能是将大小为13×13的特征图放大，变为大小26×26的特征图以完成两张图像的Concat操作，以综合图像的多维度信息。上采样的原理包括简单补0、复制原值、反卷积、插值方法和扩张卷积等，本网络中使用的是复制原值的方法，即每个像素按比例扩大后，扩大出的新像素填充原值的方法。以一个大小为2×2的图像上采样为4×4的图像为例，其运算如图4.12所示。

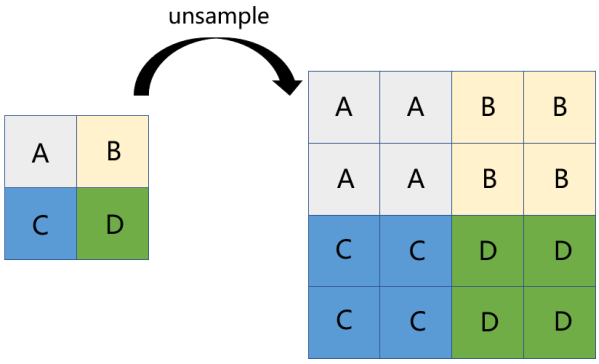


图 4.12 上采样运算

上采样模块的伪代码如图4.13所示。

```
for(n=0;n<N;n++)
        for(i=0;i<Kin;i++)
            for(j=0;j<Kin;j++){
#pragma HLS PIPELINE
                tmp=*(in+ Kout* Kout+i* Kout+j);
                *( out+n*Kout* Kout +i* Kout *scale+scale*j)=tmp;
                *( out+n*Kout* Kout +i* Kout *scale+scale*j +1)=tmp;
                *( out+n*Kout* Kout +i* Kout *scale+scale*j +Kout)=tmp;
                *( out+n*Kout* Kout +i* Kout *scale+scale*j +Kout+1)=tmp;}
```

图 4.13 上采样伪代码

其中 N 表示特征图的数量，Kin 表示输入特征图的大小，Kout 表示输出特征图的大小，scale 表示上采样系数，in 表示输入特征图像素的地址，out 表示输入特征图上采样后输出对应像素的地址。

4.7 Arm 端后处理实现

后处理模块主要任务是从网络推理得到的输出特征中进行检测框的筛选，找出最准确的预测框。当输入图像经过本文设计的轻量化YOLO检测网络推理后，会得到大小为13×13×75和26×26×75的两张特征图，其中13和26代表着将输入图像分割为了13×13和26×26两种不同大小网格，每个网格包含3个预测框，而每个预测框有25个参数，分别由4个位置坐标信息和1个类别置信度和20个类别预测值组成。可以利用这25个参数解码出一个确定的预测框，然后再在一定区域内的同种类预测框中采用非极大值抑制（Non-Maximum Suppression，NMS）算法筛选出预测最准确的一个框，非极大值算法的流程图如图4.14所示。

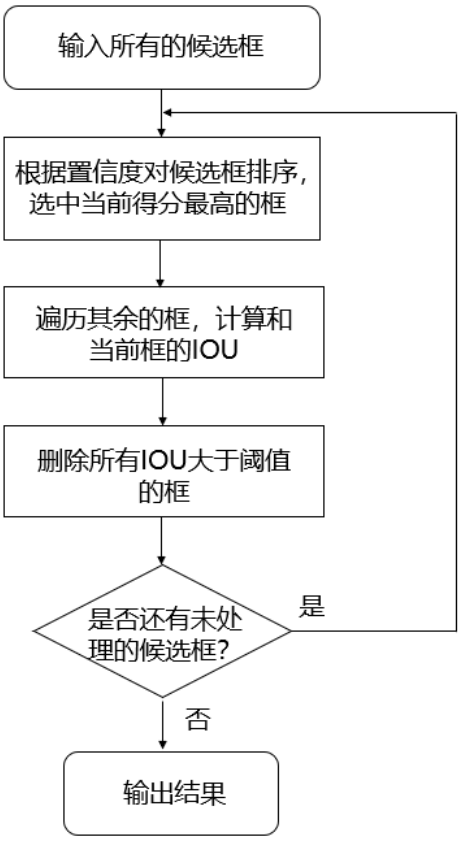


图 4.14 非极大值抑制算法流程图

4.8 本章小结

本章根据上一章设计的轻量化 YOLO 卷积神经网络算法，针对各层结构使用 Vivado HLS 工具设计了相应的硬件。首先介绍了整体设计架构和数据流图，然后在访存部分应用了图像循环分块策略以降低内存需求，双缓冲和流水化策略加速硬件运行；在计算部分对普通卷积、

深度卷积核点卷积三种不同卷积单独设计并进行了并行化处理，最后又介绍了上采样层和Arm 端后处理实现的过程。

第五章　测试与结果分析

为验证目标检测模型和硬件加速器设计的有效性和优越性，还需要对加速器进行性能测试，因此，本章将网络模型部署在 PC 上的 GPU、CPU 平台和本文设计的 AI 加速器平台进行推理，完成横向性能对比，并将本文设计的 AI 加速器与其它文献设计的 AI 加速器进行纵向性能对比。

5.1 软硬件平台介绍

5.1.1　硬件平台

本设计采用米联客公司出品的MLK-CZ01-7020-400开发平台，实物图如图5.1所示。

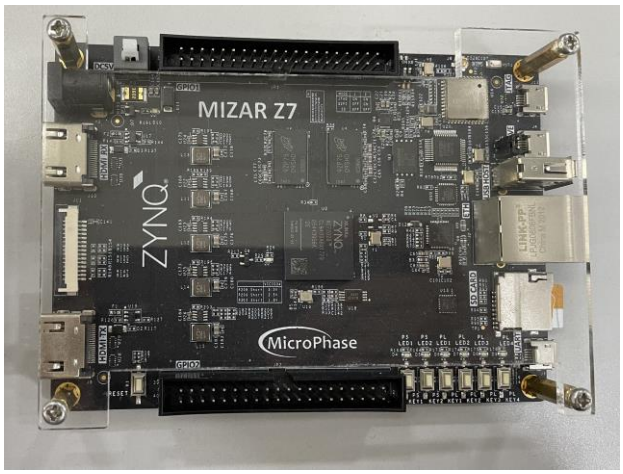


图 5.1 MLK-CZ01-7020-400 开发平台

主控芯片为Xilinx公司的Zynq7020-CLG400-2，系统架构为ARM+FPGA的异构架构，如图5.2所示。

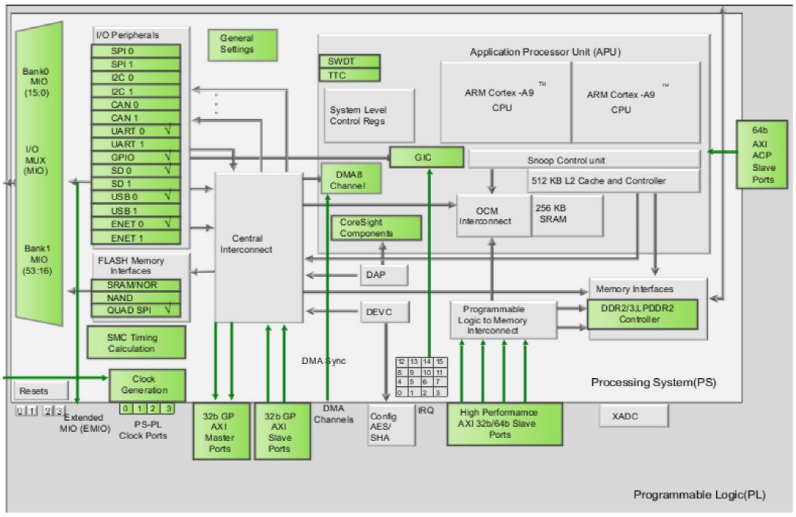


图 5.2 Zynq-7020 SOC 芯片架构

该芯片由处理器系统（Processing System, PS）和可编程逻辑部分（Programmable Logic, PL）两部分组成。处理器系统可以被视为一个基于 ARM 核的 SOC，它包含双核 ARM

Cortex-A9 处理器、AMBA 互连、内部存储器、外部储存接口和外围设备。这些外围设备包括 USB 总线接口、以太网接口、SD/SDIO 接口、I2C 总线接口、CAN 总线接口、UART 接口和 GPIO 等。可编程逻辑部分通常也称为 FPGA，它由查找表、触发器、计算模块和存储模块等组成，可以通过编程实现对应的硬件功能[58]。

集成了 ARM 处理器和可编程逻辑的 Zynq7020 SOC 芯片具有多个优点：首先，它们结合了强大的处理能力和灵活的可编程逻辑，可以满足高性能计算和定制化硬件设计的需求。其次，高度集成的设计降低了系统复杂性，节约了功耗和成本。此外，Zynq-7000 系列芯片拥有完善的开发生态系统，提供丰富的工具和资源，简化了开发过程并提高了开发效率。综上所述，Zynq-7000 系列芯片的优点包括强大的处理能力、灵活性、高度集成、低功耗设计和全面的开发支持，使其成为广泛应用于多个领域的理想选择。

5.1.2 软件平台

本设计主要采用 windows 平台上的 VS Code 软件使用 python 语言进行卷积神经网络代码的编写，主要使用 Anaconda 包中的 Pytorch 环境和其它相关 python 库进行卷积神经网络训练与轻量化改进。

VS Code 是一个强大的集成开发环境（IDE），具有丰富的功能和插件生态系统。它提供了代码编辑、调试、版本控制等功能，使得开发人员可以高效地编写和管理深度学习代码。Anaconda 是一个 Python 数据科学平台，包含了许多常用的科学计算和深度学习库。它提供了方便的环境管理和软件包安装，使得配置和管理深度学习环境变得简单而可靠。

本设计还使用 Vivado HLS 软件进行 PL 端的加速器设计，Vivado 进行 PL 和 PS 端的设计综合，Vitis 软件进行 PS 端开发和进行测试。

Vivado HLS 是 Xilinx 公司推出的一种高级综合工具，用于将 C、C++等高级语言的代码转换为硬件描述语言，以便在 FPGA 上进行高层次的硬件设计和优化。Vivado 是 Xilinx 公司推出的一款集成化设计环境，用于 FPGA 系统级设计。它包括了从设计、仿真、综合到布局布线和生成比特流文件等多个环节的工具和流程。Vivado 提供了一种全面的解决方案，使得开发人员能够高效地进行 FPGA 设计和开发，并支持多种 FPGA 芯片系列。Vitis 是 Xilinx 公司推出的一套开发工具套件，旨在简化基于 FPGA 的应用程序开发。它提供了一个统一的开发环境，支持使用高级编程语言进行应用程序开发，并能够将代码优化、编译和部署到 FPGA 上。Vitis 还提供了丰富的库和示例代码，以及用于性能分析和调试的工具，帮助开发人员实现高性能、低延迟的 FPGA 加速应用。综上所述，Vivado HLS 是一种用于高层次硬件设计的工具，Vivado 是一套用于 FPGA 系统级设计的集成化设计环境，而 Vitis 是一套用于基于 FPGA 的应用程序开发的开发工具套件。

5.2 实验流程

在Vivado HLS中使用高层次语言C++，根据第四章提出的设计方案，进行相关硬件功能模块的设计。设计完成后进行高层次综合，生成IP核。该IP核中对外有12个AXI4 Master接口和一个AXI4-lite Slave接口，其中m_axi_IN1-4 4个接口负责特征图像的输入，m_axi_W1-4 4个接口负责卷积权重的输入，m_axi_OUT1-4 4个接口负责特征图像的输出，s_axi_CTRL负责读写数据控制和状态寄存器。再在Vivado软件中添加该IP，将其与Zynq核以及其它IP在Block Design中连接到一起。Zynq核的GP和HP接口与加速器通过AXI SmartConnect 和AXI Interconnect连接，Processor System Reset IP控制系统的复位，整个加速器的运行频率为150 Mhz。最终整个系统的Block Design如图5.3所示。

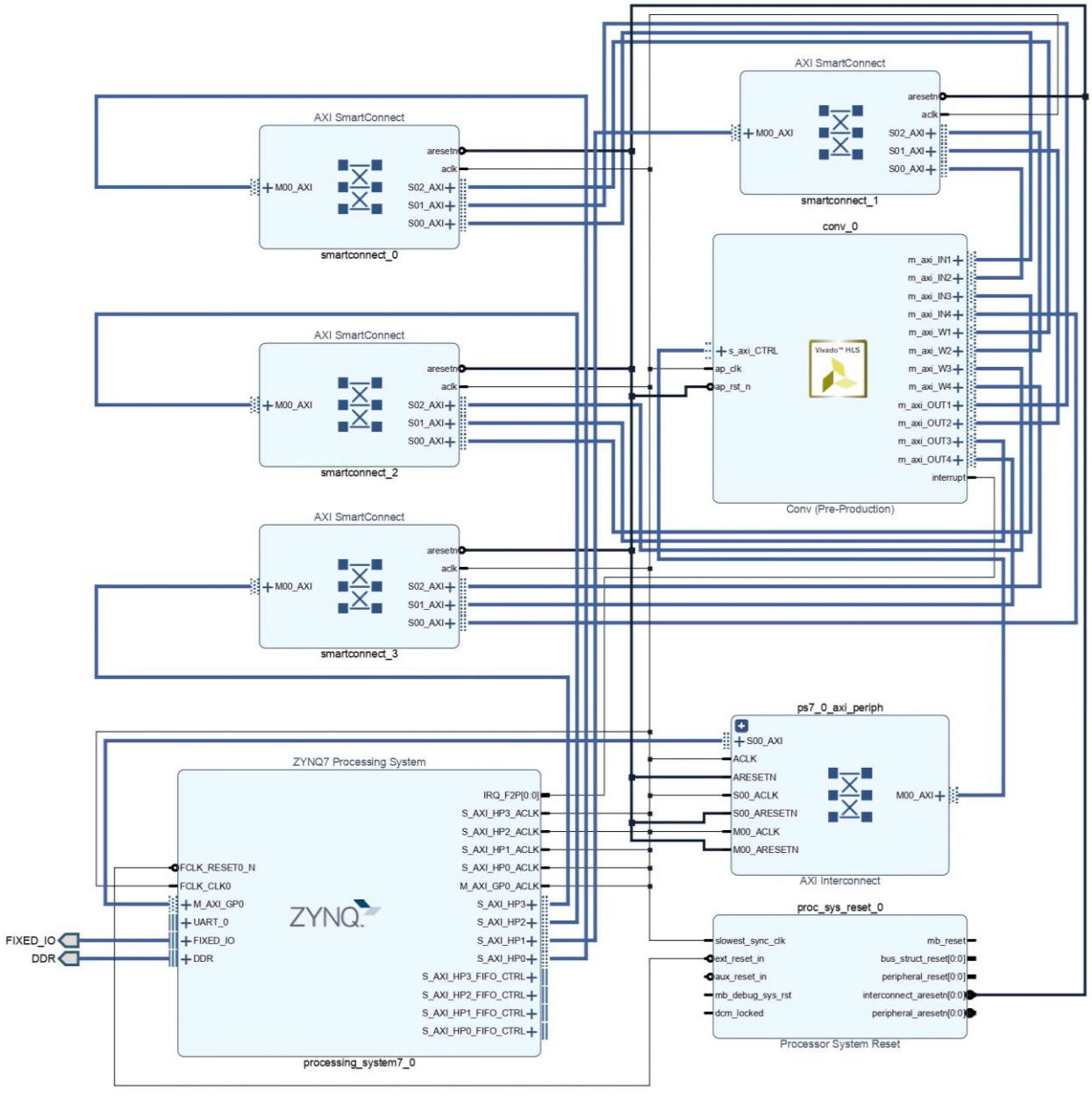


图 5.3 加速器系统 Block Design

给图5.3所示的Block Design添加一个wrapper，以这个wrapper为顶层文件进行综合，通过对硬件描述语言代码进行语法分析、优化和映射，将描述的电路转换为逻辑网表。完成综合后进行Implementation，将综合出的资源相互连接以实现布局布线。最后生成比特流文件，即将电路的物理实现映射为与目标FPGA器件相关的二进制文件。Vivado综合工具给出的本文AI加速器需要消耗的资源如表5.1所示。

表 5.1 工程占用资源

FPGA 资源	LUT	FF	DSP	BRAM
片上资源	53200	106400	220	140
占用资源	26283	28968	142	117
资源占用率	50.25%	27.22%	64.55%	83.57%

其中 DSP 资源的消耗主要用于卷积计算模块，BRAM 资源的消耗主要用于缓存数据和权重，FF 资源的消耗主要用于PS 端的控制信号以及状态配置，LUT资源的消耗主要用于各种模块中的其它电路结构。本文设计AI加速器BRAM消耗117个，占可用总量的83.57%，是所有资源中占比最大的，设计的主要压力也来自于此，因此本文对数据进行分块以优化BRAM的资源使用。

完成FPGA PL端的硬件设计后将生成的比特流文件导入Vitis软件，还需要编写Arm PS端的网络模型描述文件和调度程序控制网络的前向传播，控制流如下：首先初始化硬件电路的所有参数，并将电路设置为起始工作状态。然后根据每一层的输入地址，将特征图和卷积权重从DDR写入加速器进行计算，待计算完成后将输出特征图根据地址写回DDR中，接着执行下一层的运算。待网络所有层都执行运算完成后，对网络输出结果进行后处理运算，输出最终结果。编写完成网络模型的描述文件和调度控制软件后即可生成系统可执行文件。

由于待检测的图片大小各不相同，而本文的网络模型输入图片大小规定为416×416，因此首先需要对图片进行预处理，经过缩放将原图转化为416×416大小的图片。随后将包括经过预处理后的待测试的数据集图片，生成的可执行文件以及量化后的权重和偏置文件的文件夹从PC拷贝至SD卡中。

电脑通过USB接口与开发板上的JTAG接口和UART串口进行相连，开发板上电，将比特流下载到FPGA上并使用串口通信软件运行可执行文件。此时，存储在SD卡中的各网络层的结构参数和测试用图片都将被加载到DDR上，网络前向推理开始启动，软硬件相互配合对测试图片进行检测，当检测完成后，结果图片存回SD卡中。测试平台实物图如图5.4所示。

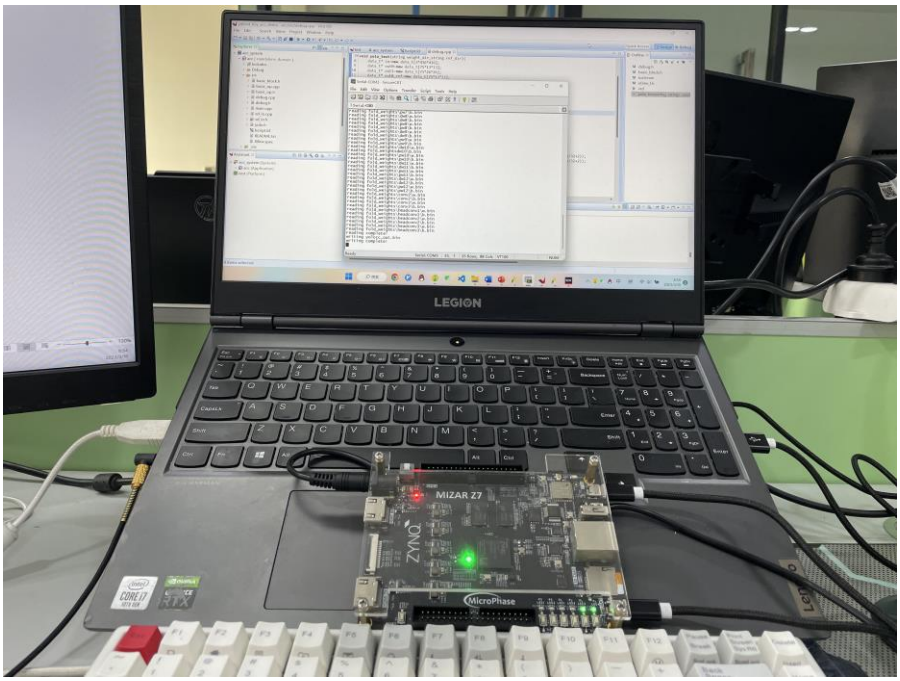
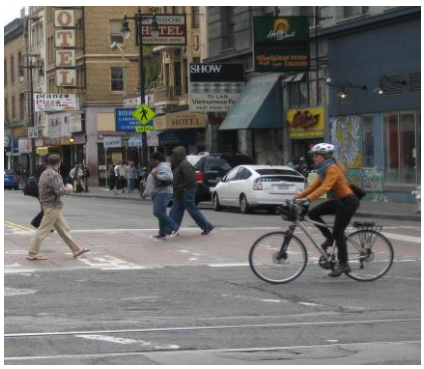


图 5.4 测试平台实物图

为验证软硬件平台的有效性，本文选取了以下四张例图进行测试，每张图片都包含有多个 VOC 数据集中的待检测目标，如图 5.5 所示。



测试例图(a)

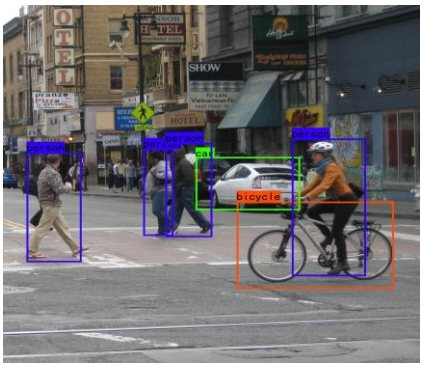
测试例图(b)

测试例图(a)

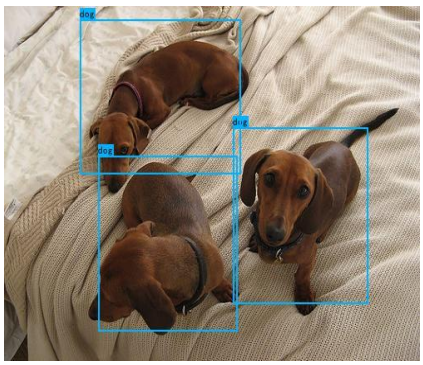
测试例图(b)

图 5.5 测试例图

经过前文所述的实验流程，将以上图片和网络模型放入本文设计的 AI 硬件加速器和 PC 平台上的 RTX 2060 GPU 中进行检测，得到的结果分别如图 5.6 和图 5.7 所示。

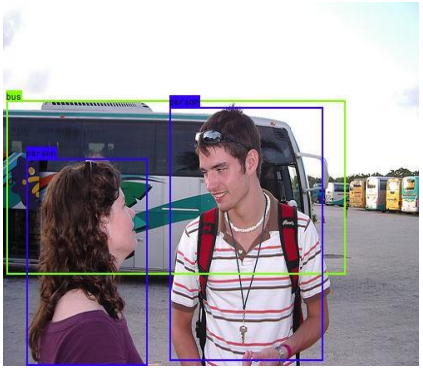


测试结果(a)



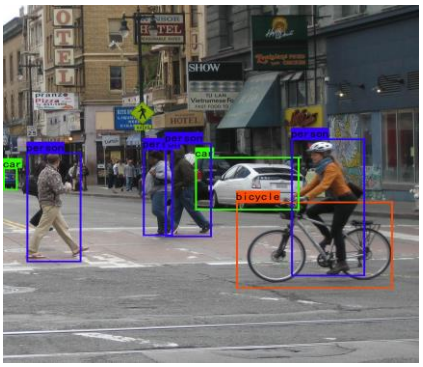
测试结果(b)

测试结果(c)

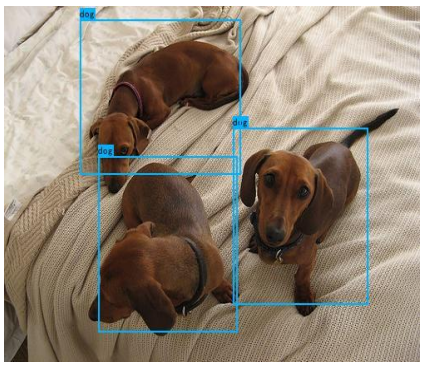


测试结果(d)

图 5.6 加速器测试结果

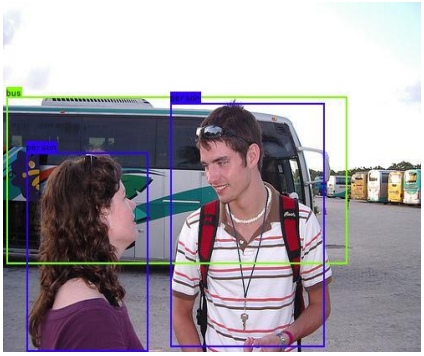


测试结果(a)



测试结果(b)

测试结果(c)



测试结果(d)

图 5.7 RTX 2060 GPU 测试结果

返回的测试结果将检测到的不同种类目标用不同颜色的框线框出，并在左上角注明了该

类别。可以从测试结果图片看出，网络模型在 AI 加速器和 RTX 2060 GPU 中均能正常完成推理计算工作，返回计算结果。本文设计 AI 加速器相比起 RTX 2060 GPU，在测试例图(a)中会少检测到一个最左侧的车辆目标，这是由于硬件计算时的精度损失导致的，而其它目标均能正常被检出。对于测试例图(b)，测试例图(c)和测试例图(d)三张图片，检测返回的结果均完全相同。因此从整体上来说，可以认为本文设计的卷积神经网络模型和 AI 加速器能够有效地对目标进行检测。

5.3 加速性能对比

为了充分证明本设计在加速性能和功耗方面存在优势，本设计将从横向和纵向两个方面进行对比，横向对比就是将硬件加速结果与PC端常用的CPU与GPU进行对比和分析，纵向对比就是将硬件加速结果与其他人设计的目标检测AI加速器的结果进行对比分析。

5.3.1 横向性能对比

分别在 PC 平台上的 Intel i7-10750H CPU，Nvidia RTX2060 GPU 和本文设计的 AI 加速器上测试本文设计的轻量化 YOLO 网络的检测性能，为消除单张图片造成的误差，使用 100 张图片进行检测后算出运行时间的平均值。通过测试得到的不同硬件平台的速度和能耗表现如表 5.2 所示。

表 5.2 不同硬件平台的速度和能耗表现

平台	Intel i7-10750H	Nvidia RTX2060	本 AI 加速器
100 张图片用时	62.8s	2.7s	20.7s
帧率	1.5 FPS	37.0 FPS	2.1 FPS
工作频率	2.6GHz	1650MHz	150Mhz
功耗	45W	100W	2.38W
能效比	0.034	0.370	0.870

表中可以看出，使用Intel i7-10750H CPU对100张图片进行批量检测需要用时62.8秒，帧率为1.5 FPS，而使用NVIDIA RTX2060 GPU 对100张图片进行批量检测需要用时2.7秒，帧率为37.0fps。本文设计的轻量化YOLO网络AI加速器和Intel i7-10750H CPU相比用时减少了40%，不过和NVIDIA RTX2060 GPU相比，用时就要长很多。

虽然与NVIDIA RTX2060 GPU相比，本文设计的AI加速器在速度上不占优势，但是该加速器的优点在于功耗较低，仅为2.38W，其中动态功耗为2.17W，静态功耗为0.21W，而NVIDIA RTX2060 GPU的功耗高达100W，两者相差悬殊。为了更好的衡量各设计的性能，本文引入了能效比指标，其代表的意义是处理一帧图片所需要消耗的能量，其数值越高就说明对在相同能效下卷积神经网络的加速效果越好，能效比的计算如式5.1所示：

$$能效比 = \frac{帧率}{功率} \tag{5.1}$$

图5.8以柱状图的形式分别列出了本文设计的轻量化YOLO网络AI加速器、Intel i7-10750H CPU和NVIDIA RTX2060 GPU能效比的表现。从图中可以非常直观的看出，本文设计的轻量化YOLO网络AI加速器在能效比方面的表现要优于NVIDIA RTX2060 GPU和Intel i7-10750H CPU，在具体数值上，本设计的能效比为Intel i7-10750H CPU的25.59倍，NVIDIA RTX2060 GPU的2.35倍。

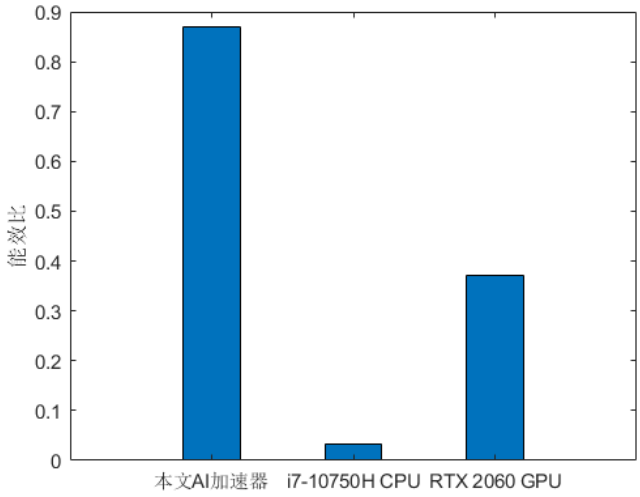


图 5.8 不同平台实现轻量化 YOLO 网络能效比对比

5.3.2 纵向性能对比

为了进一步评估AI加速器的性能，将本文提出的硬件与近年来其他学者在关于目标检测的AI加速器研究成果进行对比分析，如表5.3所示。

表 5.3 本文方法与近年来其他学者的研究成果进行对比

网络	YOLOv2-tiny[60]	YOLOv3-Tiny[61]	YOLOv2[62]	MobileNet-SSD[63]	YOLOv2[64]	本文
开发板	7020	Pynq-z2	pynq-z2	ZYNQ 7100	ZCU102	7020
时钟频率(Mhz)	142	150	150	100	300	150
数据精度	int8	int16	int16	fp32	int2	int16
DSP	110	32	-	1944	377	142
帧率(FPS)	24.50	2.22	0.62	0.95	24.50	2.07
功耗(W)	-	2.80	2.75	8.53	4.50	2.38
能效比	-	0.79	0.23	0.11	5.44	0.86
运算量(GOP)	0.30	5.57	62.64	28.21	14.18	2.57
吞吐率(GOPS)	7.46	12.25	39.15	26.67	347.44	5.32

文献[59]主要是通过对YOLOv2-tiny模型进行剪枝以缩小参数量，提出在PL端乘数与被乘数成对存入同一BRAM单元的存储办法，解决了BRAM数量和端口稀缺带来流水线结构和

数据冒险问题，从而提高了流水效率，从而达到了高帧率的检测效果，但是模型本身精度较差；文献[60]主要是采用了循环展开、循环分块和循环交换三种循环优化策略以及参数重排序、乒乓缓冲和多通道数据传输三种时延优化策略，实现了YOLOv3-tiny的加速，但是其网络模型仍有待优化空间，以实现更高效的加速；文献[61]针对PL侧数据读写效率低的问题，引入了参数重排序和outstanding技术以降低DDR读写时的时延，以加速运行，但是网络模型太大，导致运算量也很大，成为了上表中帧率最低的一个。文献[62]使用Roofline模型对目标检测算法和硬件参数进行了计算分析，得到了最适合在其硬件平台上部署的模型以及相应的硬件参数，使开发板的资源得到了最大利用，但是模型没有经过量化，仍然采用32位浮点进行运算，资源占用太多，导致帧率功耗比很低；文献[63]将卷积核内的权重直接二值化，极大地节省了硬件资源，但是导致检测精度很低，不能满足复杂环境下的检测任务。

本文设计的AI加速器首先针对检测网络模型进行了包括替换主干网络、IOU以及激活函数的修改，达到了在参数量不增长太多的前提下有效提高网络检测精度的目标，然后又对模型进行了剪枝、BN层融合和量化操作，达到了轻量化模型的目的，再根据模型的特点设计了三种不同的卷积运算模块，以及采用了循环分块、并行运算、双缓存设计和流水化等优化操作，完成了硬件对算法的加速。虽然由于网络模型深度较深以及将硬件资源分配给三种卷积，导致硬件的吞吐率不高，但是其能效比非常优秀，高于文献[59-62]，仅次于文献[63]。而文献[63]的硬件加速器是通过二值化的方法，以牺牲大量精度为代价换取了优异的其它性能指标。综上所述，本文的网络和硬件在达到相同检测速度的条件下拥有较低的能耗，符合业界与生活实际的需要。

5.4 本章小结

本章首先对AI加速器设计和测试的软件平台和硬件平台进行了介绍，随后对上板测试实验的步骤进行了详细讲解，通过返回的检测结果图可以看出其结果是符合预期的。最后又根据AI加速器的各个参数进行了横向和纵向对比，证明了本文设计的AI加速器的有效性和优越性。

第六章　总结与展望

6.1 本文总结

目标检测网络是运用深度学习中的神经网络技术实现对输入图像目标位置的定位和类别检测的一种计算机视觉算法，在军事和民用领域有着广泛的应用。其中YOLO系列算法是当前最热门的目标检测算法之一。但是当前学界的主流的方向是朝着精度的不断提升进行的，网络结构越来越复杂，参数也越来越多，导致模型难以部署到边缘端设备中，而且运行速度也很慢，不能满足生活中的需求，因此轻量化网络和AI加速器有着极强的现实意义，本文以面向轻量化YOLO网络的AI加速器设计与实现为主题，做出了以下创新性工作：

（1）通过对卷积神经网络的基本组成和 YOLO 系列目标检测算法架构的分析，提出了轻量化 YOLO 网络：使用 0.5 MobileNet 替换 YOLOv3-tiny 网络的 7 层卷积主干网络，并使用 GIOU 回归度量函数和 LeakyReLU 激活函数进一步提高精度。接着在 VOC 数据集下进行了训练，并将训练完成的模型与现有的一些网络模型对比，证明了本文提出的 YOLO 网络在满足精度较高要求的同时实现了轻量化，易于在边缘端部署。

（2）使用本文提出的基于相邻层权重的卷积神经网络裁剪方法对训练完成的模型进行了剪枝，以减少模型内参数的冗余。随后又进行了 BN 层融合，减少参数并优化硬件实现需要的资源，然后将模型量化到 16bit，进一步减少了模型的参数，完成了网络模型的轻量化。

（3）根据上文设计的卷积神经网络模型，使用高层次综合技术设计了对应的硬件加速器，其中针对网络中存在的标准卷积、深度卷积和点卷积三种不同的运算分别设计了对应的硬件加速模块，可根据不同卷积实现专用的加速。还使用了循环分块、并行计算、双缓存模块和流水线等优化设计以加速运行。

最后，将本文设计的轻量化 YOLO 网络模型和 AI 加速器部署在搭载 Zynq-7020 核心的 FPGA 平台上进行了测试和验证，证明其能有效运行。随后又进行了进一步的分析，将加速器和 CPU、GPU 进行横向对比，本文加速器的能效比达到了 Intel i7-10750H CPU 的 28.44 倍，NVIDIA RTX2060 GPU 的 2.61 倍，随后又和其它文献设计的目标检测 AI 加速器进行了纵向对比，也有着相对优良的效果，证明了该加速器设计的有效性和优越性。

6.2 未来研究展望

本文虽然成功完成了面向轻量化YOLO网络的AI加速器设计与实现任务，但仍然存在一些可以改进的问题：

（1）网络模型仍可以进一步优化。本文为了网络结构简单使用0.5 MobileNet对主干网络进行替换，但是MobileNet如今已经有了性能更加优秀的v2、v3版本，进一步的研究可以

使用MobileNetv2或者MobileNetv3尝试替换主干网络以在轻量化的同时获取更高的检测精度。

（2）加速器的参数仍可以改进。受限于时间与知识，本文未对加速器的参数选取进行进一步的讨论，最终实现的工程也未利用到FPGA开发板的全部资源，没有做到最优化设计，后续研究可以对加速器的参数进行优化。

（3）硬件实现系统仍可以改进。本文设计的硬件加速器实现是将图片放入SD卡中进行处理和运算，然后将结果再写回SD卡中，操作繁琐且不直观，后续考虑在开发板上接入显示器和摄像头，实现实时的监看与目标检测。

参考文献

[1] Pei J , Deng L , Song S , et al. Towards artificial general intelligence with hybrid Tianjic chip architecture[J]. Nature, 2019, 572(7767):106.

[2] Russakovsky O , Deng J , Su H , et al. ImageNet Large Scale Visual Recognition Challenge[J]. International Journal of Computer Vision, 2014:1-42.

[3] Krizhevsky A , Sutskever I , Hinton G . ImageNet Classification with Deep Convolutional Neural Networks[J]. Advances in neural information processing systems, 2012, 25(2).

[4] Zhe L , Wei Q Y , Yang M L . Image denoising based on a CNN model[C].2018 4th International Conference on Control, Automation and Robotics (ICCAR). 2018.

[5] Collobert R, Weston J. A unified architecture for natural language processing: Deep neural networks with multitask learning[C].Proceedings of the 25th international conference on Machine learning. 2008: 160-167.

[6] Graves A, Mohamed A, Hinton G. Speech recognition with deep recurrent neural networks[C].2013 IEEE international conference on acoustics, speech and signal processing. Ieee, 2013, 6645-6649.

[7] Arena P, Basile A, Bucolo M, et al. Image processing for medical diagnosis using CNN[J]. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 2003, 497(1): 174-178.

[8] Yan C, Coenen F, Zhang B. Driving posture recognition by convolutional neural networks[J]. IET Computer Vision, 2016, 10(2): 103-114.

[9] Zhang Z , Lin X , Li M , et al. A customized deep learning approach to integrate network-scale online traffic data imputation and prediction[J]. Transportation Research Part C: Emerging Technologies, 2021, 132:103372.

[10] Jiao L , Zhang F , Liu F , et al. A Survey of Deep Learning-based Object Detection[J]. IEEE Access, 2019, 7:128837-128868.

[11] 邵延华, 张铎, 楚红雨, 等. 基于深度学习的 YOLO 目标检测综述[J]. 电子与信息学报, 2022, 44: 0.

[12] Oltean G, Florea C, Orghidan R, et al. Towards real time vehicle counting using yolo-tiny and fast motion estimation[C].2019 IEEE 25th international symposium for design and technology

in electronic packaging (SIITME). IEEE, 2019: 240-243.

[13] Dibbo S V, Cheung W, Vhaduri S. On-phone CNN model-based implicit authentication to secure IoT wearables[C].The Fifth International Conference on Safety and Security with IoT: SaSeIoT 2021. Cham: Springer International Publishing, 2022: 19-34.

[14] 雷杰, 高鑫, 宋杰,等. 深度网络模型压缩综述[J]. 软件学报, 2018, 29(2):16.

[15] Szerwinski R, Güneysu T. Exploiting the power of GPUs for asymmetric cryptography[C].International Workshop on Cryptographic hardware and embedded systems. Springer, Berlin, Heidelberg, 2008: 79-99.

[16] Shawahna A, Sait S M, El-Maleh A. FPGA-based accelerators of deep learning networks for learning and classification: A review[J]. ieee Access, 2018, 7: 7823-7859.

[17] Fukushima K . Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position[J]. Biological Cybernetics, 1980, 36(4):193-202.

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

[19] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84-90.

[20] Simonyan K , Zisserman A . Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. arXiv e-prints, 2014.

[21] He K， Zhang X， Ren S , et al. Deep Residual Learning for Image Recognition[C]. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2016.

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

[23] Girshick R. Fast r-cnn[C].Proceedings of the IEEE international conference on computer vision. 2015: 1440-1448.

[24] Ren S, He K, Girshick R, et al. Faster r-cnn: Towards real-time object detection with region proposal networks[J]. Advances in neural information processing systems, 2015, 28.

[25] Dai J， Li Y， He K , et al. R-FCN: Object Detection via Region-based Fully Convolutional

Networks[J]. Curran Associates Inc. 2016.

[26] Liu W, Anguelov D, Erhan D, et al. Ssd: Single shot multibox detector[C].Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14. Springer International Publishing, 2016: 21-37.

[27] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C].Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 779-788.

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

[29] Farhadi A, Redmon J. Yolov3: An incremental improvement[C].Computer vision and pattern recognition. Berlin/Heidelberg, Germany: Springer, 2018, 1804: 1-6.

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

[31] Wu W, Liu H, Li L, et al. Application of local fully Convolutional Neural Network combined with YOLO v5 algorithm in small target detection of remote sensing image[J]. PloS one, 2021, 16(10): e0259283.

[32] 李江昀, 赵义凯, 薛卓尔, 等. 深度神经网络模型压缩综述[J]. 工程科学学报, 2019, 41(10): 1229-1239.

[33] Li H, Kadav A, Durdanovic I, et al. Pruning filters for efficient convnets[J]. arXiv preprint arXiv:1608.08710, 2016.

[34] Vanhoucke V， Senior A， Mao M Z . Improving the speed of neural networks on CPUs[C]// Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011. 2011.

[35] Hinton G， Vinyals O， Dean J . Distilling the Knowledge in a Neural Network[J]. Computer Science, 2015, 14(7):38-39.

[36] 霍天娇.基于轻量级神经网络的智能汽车目标检测[D].吉林大学,2022.

[37] Iandola F N, Han S, Moskewicz M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size[J]. arXiv preprint arXiv:1602.07360, 2016.

[38] Zhang X, Zhou X, Lin M, et al. Shufflenet: An extremely efficient convolutional neural network for mobile devices[C].Proceedings of the IEEE conference on computer vision and pattern

recognition. 2018: 6848-6856.

[39] Howard A G, Zhu M, Chen B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J]. arXiv preprint arXiv:1704.04861, 2017.

[40] 魏少军. 可重构芯片的方法学原理[J]. 科技导报, 2019(3):8.

[41] Chen T， Du Z， Sun N , et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning[J]. Acm Sigplan Notices, 2014, 49(4):269-284.

[42] Chen Y, Luo T, Liu S, et al. Dadiannao: A machine-learning supercomputer[C].2014 47th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE, 2014: 609-622.

[43] Ardavan, Pedram, Song, et al. EIE: Efficient Inference Engine on Compressed Deep Neural Network[J]. Computer architecture news, 2016, 44(3):243-254.

[44] Cass S. Taking AI to the edge: Google's TPU now comes in a maker-friendly package[J]. IEEE Spectrum, 2019, 56(5): 16-17.

[45] Sengupta J, Kubendran R, Neftci E, et al. High-speed, real-time, spike-based object tracking and path prediction on google edge TPU[C].2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS). IEEE, 2020: 134-135.

[46] Deng L， Wang G， Li G , et al. Tianjic: A Unified and Scalable Chip Bridging Spike-Based and Continuous Neural Computation[J]. IEEE Journal of Solid-State Circuits, 2020, 55(8):2228-2246.

[47] 苑福利. 基于动态硬件重构的卷积神经网络加速器[D]. 中国科学技术大学,2021.

[48] 刘书勇. 面向 FPGA 的可重构编译器研究[D]. 哈尔滨工程大学,2017.

[49] ZHANG C， LI P， SUN G, et al. Optimizing fpga-based accelerator design for deep convolutional neural networks [C].Proceedings of the 2015 ACM/SIGDA international symposiumon field-programmable gate arrays. 2015: 161-170.

[50] Chen Y H, Krishna T, Emer J S, et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks[J]. IEEE journal of solid-state circuits, 2016, 52(1): 127-138.

[51] LIANG Y, LU L, XIAO Q, et al. Evaluating fast algorithms for convolutional neural net-works on fpgas [J].IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019,39(4): 857-870.

[52] Nguyen D T， Nguyen T N， Kim H , et al. A High-Throughput and Power-Efficient FPGA

Implementation of YOLO CNN for Object Detection[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2019:1-13.

[53] Xiong Q, Liao C, Yang Z, et al. A Method for Accelerating YOLO by Hybrid Computing Based on ARM and FPGA[C].2021 4th International Conference on Algorithms, Computing and Artificial Intelligence. 2021: 1-7.

[54] 基于深度学习的行车障碍物目标检测[D]. 山东大学, 2022

[55] Rezatofighi H , Tsoi N , Gwak J Y , et al. Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression[C]。2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2019.

[56] 杨岸青, 王彬, 徐凯,等. 一种基于相邻层权重的卷积神经网络裁剪方法: CN113935485[P].2022.03.04

[57] Winterstein F , Bayliss S , Constantinides G A . High-level synthesis of dynamic data structures: A case study using Vivado HLS[C].International Conference on Field-programmable Technology. IEEE, 2013:362-365.

[58] 庄琼. 基于 AXI 总线的 DMA 高速通道及驱动的设计与实现[D]. 电子科技大学, 2019.

[59] 付凌浩. 面向移动嵌入式设备的 MobileNet 卷积神经网络计算加速研究[D]. 武汉理工大学, 2021.

[60] 张雲轲, 刘丹. 基于小型 Zynq SoC 硬件加速的改进 TINY YOLO 实时车辆检测算法实现[J]. 计算机应用, 2019, 39(1):7.

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

[62] 姚沛东. 基于 FPGA 的目标检测硬件加速技术及其应用研究[D].江苏大学,2022.

[63] 邹丹音. 基于深度学习的目标检测算法 FPGA 实现[D]. 哈尔滨工业大学,2019.

[64] Nakahara H , Yonekawa H , Fujii T , et al. A Lightweight YOLOv2: A Binarized CNN with A Parallel Support Vector Regression for an FPGA[C].the 2018 ACM/SIGDA International Symposium. ACM, 2018.

毕业/学位论文答辩委员会名单

毕业/学位论文题目		面向轻量化 YOLO 网络的 AI 加速器设计与实现		
作　者		杨岸青		
专　业		电子信息		
研究方向		电路与系统		
导　师		徐建		
答辩委员会组成	**姓　名**	**职　称**	**学科专业**	**工作单位**
	汪小军 **（主席）**	高工		紫金山实验室
	唐路	副教授	电路与系统	东南大学
	徐欧	副教授	电磁场与微波技术	东南大学
	李伟 **（秘书）**	讲师	电路与系统	信息科学与工程学院