学校代码：10284
分 类 号：TP181
密　　级：公开
U D C：004.8
学　　号：502022330064

南京大学

硕 士 学 位 论 文

论 文 题 目　面向边缘计算平台的高能效神经网络研究

作 者 姓 名　俞诗航
专 业 名 称　计算机科学与技术
研 究 方 向　边缘智能
导 师 姓 名　申富饶　教授

2025 年 5 月 16 日

答辩委员会主席 武港山 教授

评　　阅　　人 张剡 高级工程师

徐明华 教授

论文答辩日期 2025 年 5 月 16 日

研究生签名：

导师签名：

Research on High-Energy-Efficient Neural Networks for Edge Computing Platforms

by

Yu Shi-hang

Supervised by

Professor Shen Fu-rao

A dissertation submitted to

the graduate school of Nanjing University

in partial fulfilment of the requirements for the degree of

MASTER

in

Computer Science and Technology

School of Chemistry and Chemical Engineering

Nanjing University

May 16, 2025

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：面向边缘计算平台的高能效神经网络研究

计算机科学与技术 专业 2022 级硕士生姓名：俞诗航

指导教师（姓名、职称）：申富饶 教授

摘 要

随着深度学习的飞速发展和物联网技术的大规模普及，越来越多的边缘设备被赋予神经网络的推理能力，边缘智能已成为最火热的话题之一。在追求更高性能的同时，神经网络模型日益复杂，功耗与资源占用不断攀升，如何将神经网络模型高效地部署在资源功耗受限的边缘计算平台上也成为了一个值得重点关注的领域。针对神经网络模型的边缘部署，设计高能效的硬件加速器是提升部署效率的有效方法。通过软硬件协同优化，神经网络模型既可以达到更高的性能，又可以突破边缘计算平台的资源功耗瓶颈，助力实现万物智能互联。

本文主要针对边缘智能领域以及神经网络关键技术进行描述与回顾，再分别基于第二代神经网络的代表——卷积神经网络和第三代神经网络的代表——脉冲神经网络，在 FPGA 平台和 NPU 平台设计低功耗高能效的硬件加速器，并利用提出的加速器设计方法，实现了一个基于边缘计算平台的实时目标检测系统。本文的主要工作包括以下几个方面：

1. 本文提出了一种基于 FPGA 平台的低功耗卷积神经网络加速器设计方法。该方法通过软硬件协同优化，在模型层面对网络模型进行轻量化处理和算子重构，并结合 FPGA 平台优化数据流，可以得到高性能低功耗的 YOLOv4-tiny 模型。
2. 本文提出了一种基于国产 NPU 平台的高能效脉冲神经网络加速器设计方法。该方法开创性地将 SNN 模型迁移到国产 NPU 平台上，并针对 NPU 平台进行模型适配优化，有效提升了 SNN 在边缘计算平台的推理性能。
3. 本文设计并实现了一套基于边缘计算平台的实时目标检测系统。该系统应用了本文提出的卷积神经网络加速器设计方法，高效地将目标检测模型部

署在边缘计算平台并设计成拥有良好交互性的实时系统，允许用户灵活地根据功能需要完成目标检测任务。

关键词：边缘计算；卷积神经网络；脉冲神经网络；高能效神经网络

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Research on High-Energy-Efficient Neural Networks for Edge Computing Platforms

SPECIALIZATION: Computer Science and Technology

POSTGRADUATE: Yu Shi-hang

MENTOR: Professor Shen Fu-rao

ABSTRACT

With the rapid development of deep learning and the large-scale proliferation of IoT technologies, an increasing number of edge devices are being equipped with neural network inference capabilities, making edge intelligence one of the most prominent topics. While pursuing higher performance, neural network models are becoming increasingly complex, leading to rising power consumption and resource demands. Efficiently deploying these models on resource- and power-constrained edge computing platforms has thus emerged as a critical area of focus. Designing high-energy-efficient hardware accelerators is an effective approach to enhance deployment efficiency for neural network models on edge devices. Through hardware-software co-optimization, neural network models can achieve superior performance while overcoming resource and power bottlenecks on edge platforms, facilitating the realization of smart everything.

This paper provides an overview and review of edge intelligence and key neural network technologies. It then designs low-power, high-energy-efficient hardware accelerators for FPGA and NPU platforms, focusing on convolutional neural networks (CNNs, representative of second-generation neural networks) and spiking neural networks (SNNs, representative of third-generation neural networks). Using the proposed accelerator design methods, a real-time object detection system based on edge computing platforms is implemented. The main contributions of this work are as follows:

1. A Low-Power CNN Accelerator Design for FPGA Platforms: This method em-

ploys hardware-software co-optimization to lightweight and reconstruct network operators at the model level, while optimizing dataflow for FPGA platforms. This achieves a low-power YOLOv4-tiny model with maintained high performance.

2. A High-Energy-Efficient SNN Accelerator Design for Domestic NPU Platforms: This method innovatively migrates SNN models to domestic NPU platforms and optimizes model adaptation for NPUs, significantly improving SNN inference performance on edge computing platforms.
3. A Real-Time Object Detection System for Edge Computing Platforms: This system applies the proposed CNN accelerator design method to efficiently deploy object detection models on edge platforms, creating a real-time system with user-friendly interactivity that allows flexible adaptation to functional requirements.

KEYWORDS: Edge Computing; Convolutional Neural Networks; Spiking Neural Networks; High-Energy-Efficient Neural Networks

目　录

插图目录

表格目录

第一章　引言

1.1　研究背景与意义

人工智能 (Artificial Intelligence, AI) 是最重要的科学研究领域之一，AI 技术以我们过去认为不可能的方式改变了世界。神经网络 (Neural Network, NN) 作为人工智能领域的核心技术，是一种模拟人类大脑神经元结构和功能的计算模型，其通过大量神经元之间的相互连接和信息传递，实现对数据的学习、处理和模式识别。自 20 世纪 40 年代以来，神经网络在理论研究和实际应用方面都取得了显著进展。从早期的 MP 模型[1]与感知机[2]，到近期的深度神经网络及大模型，神经网络经历了漫长且曲折的发展历程，被广泛应用在图像分类，语音识别，智能驾驶，医学诊断，金融投资，国防军事等各个领域[3]。与此同时，随着物联网技术的普及与发展，大量的 AI 模型被部署在边缘计算平台中以满足实际场景实时应用的需求，呈现出“万物智能互联”的趋势。

随着人工智能物联网 (Artificial Internet of Things, AIoT) 的飞速发展，连接到网络的设备数量呈指数级增长，产生了海量的数据，给数据中心带来了沉重的计算负担、较长的响应延迟和过高的运行能耗。在许多应用场景中，如自动驾驶、工业自动化、智能安防等，需要对数据进行实时处理和分析，以便及时做出决策。传统的云计算架构在处理大规模数据和实时性要求高的应用时存在一些局限性。一方面，将大量数据传输到云端进行处理会导致较高的延迟，无法满足实时性要求；另一方面，云计算中心的计算资源有限，难以应对突发的大规模数据处理需求。此外，在一些应用场景中，数据的隐私和安全至关重要。例如在医疗领域，患者的个人健康数据需要得到严格的保护，将数据传输到云端进行处理可能会带来数据泄露的风险。

边缘智能将数据处理的位置从云端下移至靠近数据源的边缘设备，极大地缩短了数据处理的响应时间。通过在边缘设备进行数据的初步筛选、分析和处理，能极大程度地降低敏感数据在传输过程中的暴露风险，而仅将关键数据传输

至云端，可大幅减少数据传输量，提高网络资源的利用效率。分布在各个终端的边缘计算设备，能有效分担数据中心的计算负载，为智能应用提供更高效，更可靠的服务。随着神经网络在边缘智能中扮演的角色越来越重要，如何将各类神经网络模型有效地部署在边缘硬件设备上已经成为了一个重点关注的领域。

1.2 研究现状与挑战

1.2.1 深度人工神经网络的边缘部署

根据计算单元的特点，神经网络可以分为 3 代。第 1 代神经网络由感知机等阈值单元组成，执行阈值运算并输出二值结果。但由于其结构过于简单，被证明仅具备有限的功能，只能进行线性分类，甚至无法解决最简单的“异或”逻辑问题[4]。

为了解决线性不可分问题，第 2 代神经网络是通过在计算单元的输出上应用连续激活函数构造的，反向传播 (Back Propagation, BP) 机制使其能够计算一组连续的输出值，且对于连续型前馈神经网络，只要有足够数量的隐藏神经元，就可以以任意精度逼近任何复杂的连续映射[5]。这类拥有隐藏层和激活函数的基于联结主义的多层人工神经网络被称为第 2 代神经网络。作为第 2 代神经网络最出色的代表，深度神经网络 (Deep Neural Network, DNN) 出现于 20 世纪 80 年代中期[6]，自 2006 年以来引领了人工智能的发展，被广泛地应用到各种领域的各种任务上[7]。然而，在一些应用场景下，深度神经网络必须部署在移动设备上，例如智能手机上的人脸识别算法或汽车上的自动驾驶算法。因此，为了将深度神经网络成功部署在资源受限的边缘设备上，如何将现有的神经网络模型进行优化，使其能够在边缘设备上高效运行，是需要解决的关键问题。首先，深度神经网络中有着数以百万计的权值参数，部署一个训练好的深度神经网络模型需占用设备巨量的内存。以 GPT-3 为例，该模型拥有超过 1700 亿个参数和 45 TB 的训练数据，内存占用可达 700GB[8]。即使是较为简单的 ResNet 模型，也需要将近 100MB 内存[9]。但是对绝大部分边缘计算设备而言，其内存往往非常有限，除去本身系统所占用的内存后所剩无几，再部署一个深度神经网络模型是非常困难的。其次，边缘计算设备受限于自身面积，无法搭载足够的高性能计算模块，例如大型 GPU，因此其计算能力十分有限。然而，深度神经网络模型对

一张图像进行计算所需要的浮点运算次数是十分庞大的。以所需浮点运算次数相对较少的 AlexNet 网络[10]为例，该模型对单张图像进行处理时仍然需要大约 7×10^8 次浮点运算，而其他更复杂的深度神经网络模型则需要更多的浮点运算次数，进而和边缘计算设备所能提供的算力产生巨大缺口，导致部署困难。最后，由于部署深度学习模型应用需要庞大的计算量，所以使用这些模型时往往会产生大量的能量消耗，这对于一些电池容量有限的边缘计算平台来说是致命的隐患，很可能影响设备的正常续航。综上三点所述，如果要在边缘设备上部署神经网络模型，就必须降低神经网络的规模。为此研究人员提出多种模型压缩方法，在尽量不影响模型的准确率的前提下，降低模型的参数量和计算量。模型压缩可以分为量化和剪枝两种不同的策略，用于降低数据的位宽或直接将数据置为 0。

量化方法通过降低数据位宽来减少资源占用。根据量化后的数据位宽和数据分布特点，量化方法可以分为低位宽量化、线性量化和非线性量化。

低位宽量化将数据量化为极低的位宽，如 1 比特或者 2 比特，实现极高的模型压缩率。二值化神经网络[11]中的模型参数仅占 1 比特位宽，取值只能为 ± 1，可以实现 32 倍的压缩率。然而，实验结果表明二值化量化方法仅适用于相对简单的模型和小规模数据集，当网络模型变复杂时，例如对 GoogleNet[12]这样的网络进行二值化后，精度会有明显的下降。相比于二值化神经网络，三值化网络[13]能够有效提升模型分类精度，此时模型权值参数占 2 比特位宽，取值包括 $-1, 0, 1$。尽管三值化网络能够有效提升模型分类精度，但也与大规模数据集适配不佳。还有一些研究提出了根据训练过程或输入数据的特点动态切换权重的方法。例如，在训练过程中，当模型遇到简单的训练样本时，使用三值化权重进行快速计算和更新，以提高训练效率。而当遇到复杂的样本或在模型验证阶段，为了保证准确性，则切换到全精度权重，兼具三值化权重和全精度权重的优势。

线性量化是一种将高位宽的权值转换为低位宽（如 8 位）的技术。其基本原理是在一个给定的数值范围内，通过线性映射的方式将原始的高精度数值映射到低精度的数值空间。一个具有大量参数的深度学习模型，经过线性量化后，其存储大小可能会降低到原来的 $\frac{1}{4}$，这对于在资源受限的设备上部署神经网络非常有利。此外，低精度的数值运算通常比高精度的浮点数运算更快。Wang 等人[14]提出的 HAQ 框架，利用增强学习自动化搜索量化方法，可以在最优配置情况下将经过线性量化后的深度神经网络模型高效地部署在各类硬件平台实现加

速。

非线性量化采用非线性的函数关系来进行转换，通常是为了更好地适应数据的分布特性，包括对数量化，指数量化和基于概率分布的量化。Daisuke 等人[15]提出基于对数的以 2 为底的权值表示法，实现了 3 比特量化。PoT 量化方法[16]将待量化的值映射到由 2 的幂次方组成的离散值集合上，硬件实现相对简单，能够加速神经网络的推理过程，更适合部署在边缘设备上。针对数据集中在某些特定区间或者具有长尾分布的情况。Jain 等人[17]提出的 Biscaled-DNN 量化方法使用两个不同的缩放因子按比例缩放较大权值和较小权值，进而将长尾分布的头尾分别量化到不同的数据范围，最终成功部署在 FPGA 平台上。

神经网络剪枝是指减去神经元中对最终性能没有影响或者影响较小的连接，其本质是一种去冗余的操作，可以在保证精度损失不变或较小的前提下降低模型的资源需求，是一种普遍性的模型压缩方法。根据剪枝粒度不同，可以分为非结构化剪枝和结构化剪枝两大类。

非结构化剪枝以细粒度方式移除单个权值或连接，通过将低于设定阈值的权值置为 0 来生成稀疏的权重矩阵，能极大降低模型的参数量和理论计算量。LeCun 等人[18]首次提出对权值剪枝的概念，根据权值的影响程度移除不重要的权值来提高神经网络模型的泛化性。Guo 等人[19]提出一种动态网络手术的方法，通过边剪枝边恢复的策略，在非结构化剪枝中动态调整网络连接，避免传统一次性剪枝导致的不可逆性能损失。尽管非结构化剪枝能达到非常高的剪枝率，但无法在通用的硬件架构上直接部署，在计算过程中会引起不规则访存行为，影响硬件工作效率，因此往往需要设计特定的硬件才能实现加速。

结构化剪枝以粗粒度方式移除网络中的结构化组件，例如整个神经元、通道、层或块，这种方法会直接改变模型的架构，但是模型的结构以及权值矩阵的结构没有被破坏，因此，仍然能够通过通用硬件平台来加速。Li 等人[20]提出卷积核粒度的剪枝算法，根据卷积层中卷积核的重要性进行裁剪。Liu 等人[21]使用 BN 层中的缩放因子作为衡量通道重要性的指标来决定剪去哪些通道。Yu 等人[22]提出一种基于神经元重要性分数传播的结构化剪枝方法，通过评估并传播神经元对最终输出的重要性指导网络剪枝，实现了高压缩率与低精度损失的平衡。相比于非结构化剪枝，结构化剪枝基于硬件特性可设定特定的剪枝粒度，尽管剪枝的稀疏度相对较低，但对于硬件部署更加友好。

1.2.2 脉冲神经网络的边缘部署

神经形态计算是一种受到生物神经系统启发的计算范式，其试图基于大脑中神经元的脉冲事件，在时间和空间分布上模仿神经元和突触的功能，实现高能效的计算。作为神经形态计算的典型代表，第 3 代神经网络——脉冲神经网络 (Spiking Neural Network, SNN) 与生物神经系统非常相似，使用离散值 (脉冲) 来编码和处理数据，提供了一种高效且低功耗的计算方案[23]。在 SNN 中，脉冲神经元只有当新的输入脉冲到来时才会进行处理，这使得 SNN 在本质上更符合生物学原理，表现出和真实神经回路中相同的有利特性，如模拟计算能力、低功耗、快速推理、事件驱动、在线学习和大规模并行性等。然而，由于传统的深度学习平台无法彻底发挥出 SNN 高能效的潜力，神经形态硬件平台的出现为 SNN 的边缘部署打开了一扇窗户。不同于传统的冯 • 诺依曼架构，神经形态硬件通常采用去中心化的可扩展架构，通过路由网络交换中间结果数据，允许多计算核心同时工作。同时，在神经形态硬件中，外部存储器将不复存在，取而代之的是存储单元与计算单元的高度贴合，从而带来极高的并行性和访存效率。神经形态硬件旨在最大限度地降低能耗和成本，与 SNN 的理念十分契合, 可以看作是 SNN 及其应用部署的理想硬件平台。目前，神经形态硬件平台主要包括现场可编程门阵列 (Field Programmable Gate Array, FPGA), 神经形态芯片和非易失性存储器 (Non-Volatile Memory, NVM)。

尽管 FPGA 本身并非专门设计用于神经形态计算，但它具有高度的灵活性和不俗的并行计算能力，同时具备高能效和位级操作的特点，因此在一定程度上被认为是可以模拟生物神经网络自然可塑性的神经形态硬件平台。目前，SNN 在 FPGA 平台有非常多的部署案例，表 X 简单统计了部分基于 FPGA 实现的 SNN 模型在 MNIST 数据集的表现。

截至目前，在 FPGA 上部署 SNN 仍存在许多的挑战与困难。其中最大的问题出现在内存方面。因为当部署大规模 SNN 时，它需要大量的神经元和突触来缓冲大权重矩阵。考虑到商用 FPGA 设备中的资源数量，瓶颈往往出现在块存储器 (Block Random Access Memory, BRAM) 上，这导致网络中的神经元数量受到限制。另一方面，硬件上的实现面临着准确性和成本或资源使用之间的权衡。实现高准确率虽然性能优异，但功耗较大，如何在边缘环境中获得最合适的性能与

表 1-1 基于 FPGA 实现的 SNN 在 MNIST 数据集的表现情况

作者	年份	FPGA 平台	SNN 模型/算法	准确率 (%)	推理时间/工作频率	功耗
Neil 等人[24]	2014	Xilinx Spartan-6	事件驱动 SNN	92	0.53s/张	1.2 μJ/张
Wang 等人[25]	2017	Xilinx Virtex-6	STDP	89.1	8.4 s/张	1.76 μJ/张
Mostafa 等人[26]	2017	Xilinx Spartan6-LX150	前馈 SNN	96.98	N/A	N/A
Zhang 等人[27]	2019	Terasic DE2-115	前馈 SNN	96.26	100 MHz	293 mW
Zhang 等人[28]	2019	Xilinx VC707 FPGA	事件驱动 SNN	98	1.1 ms/张	360 mW
Abderrahmane 等人[29]	2019	Altera Cyclone V	前馈 SNN	98.15	50 MHz	N/A
Kuang 等人[30]	2019	Altera Stratix III	STDP	93	N/A	N/A
Guo 等人[31]	2019	Xilinx V7 690T	ANN-to-SNN	98.98	100 MHz	745 mW
Losh 等人[32]	2019	Xilinx Zynq XC7Z010	前馈 SNN	97.70	125 MHz	161 mW
Ju 等人[33]	2020	Xilinx Zynq ZCU102	ANN-to-SNN	98.94	6.11 ms	4.6 W
Han 等人[34]	2020	Xilinx ZC706	前馈事件驱动 SNN	97.06	200 MHz	477 mW
Fang 等人[35]	2020	Xilinx ZCU102	基于脉冲的梯度下降	99.2	7.53 ms	4.5 W
Wang 等人[36]	2020	Xilinx XCVU440	CSNN	99.16	200 MHz	1.562 5 TOPS
Aung 等人[37]	2021	Xilinx XCVU440	CSNN	99.16	200 MHz	1.5625 TOPS
Li 等人[38]	2021	Xilinx Virtex-7 VCU118	STDP	92.93	3.15 ms/张	5.04 mJ/张
Zheng 等人[39]	2021	Xilinx ZCU102	STDP	90.53	200 MHz	782 mW
Gerlinghoff 等人[40]	2021	Xilinx XCKU3P	CSNN	99.1	294 μs	3.4 W
Zhang 等人[41]	2021	Xilinx Virtex-7	BP-STDP	95.3	0.27 ms/张	0.34 mJ/张
Panchapakesan 等人[42]	2022	Xilinx ZCU102	ANN-to-SNN	99.3	200 MHz	32.7 kFPS/W
Liu 等人[43]	2022	Xilinx Kintex-7	事件驱动 SNN	97.70	4.17 ms/张	2.23 mJ/张
Ye 等人[44]	2022	Xilinx Kintex-7	CSNNP	99.10	1.21 ms/张	1.19 mJ/张
Chen 等人[45]	2022	Xilinx XC7Z045	CSNN	98.5	22.6 GSOPS	19.3 GSOPS/W
Sommer 等人[46]	2022	Xilinx XCZU7EV	CSNN	98.3	0.04 ms	2.1 W
Liu 等人[47]	2023	Zynq XA7Z020	CSNN	99.00	0.27 ms	0.28 W
Wang 等人[48]	2023	Xilinx KCU115	前馈 SNN	99.4	65.7 GSOPS	41.7 GSOPS/W
Li 等人[49]	2023	Xilinx XCZU3EG	CSNN	98.12	5.53 TOPS	2.55 W

功耗配置将一直是该领域需要面对的问题。当使用 FPGA 实现 SNN 时，另一个挑战是确定适当的模型和模型组合，以及部署在逻辑阵列上的互连和架构问题，以便网络可以进行训练和片上学习。此外，与 CPU 和 GPU 相比，FPGA 在算力上具有明显劣势，因此在 FPGA 上实现 SNN 非常耗时。最后，由于 FPGA 的可编程性相对较低，缺乏诸如 TensorFlow, PyTorch 等软件开发框架的支持，因此需要开发人员对硬件细节有更深入的了解，开发难度较高。

神经形态芯片从大脑的结构和功能中汲取灵感，其主体是一个类似于人脑神经网络的近似 SNN。与传统计算机取指-执行的循环工作方式不同，神经形态芯片遵循并行工作和分布式处理机制，完成学习、记忆、推理等认知任务。现有的神经形态芯片根据电路实现技术的不同，可以分为数模混合神经形态芯片和全数字神经形态芯片。数模混合神经形态芯片的优势在于它可以将数字和模拟电路集成在同一芯片上，减少了电路的面积、功耗和成本，在一些仿生学应用实现了全面的性能优势，非常适合实时研究系统与环境的交互问题。全数字实现的电路具有很强的可编程性与灵活性，支持 SNN 模型各种参数的配置，大大缩短了开发时间，而且不受电源、热噪声或器件不匹配的影响。在片上通信方面，数电实现也能提供相当的稳定性和可靠性。然而，全数字实现的电路会带来更多的硅片面积占用以及单位功能功耗。表 X 简单总结了当下神经形态芯片的情况。

表 1-2 神经形态芯片总结

名称	类型	核心	神经元	突触	工艺 (nm)	面积 (mm^2)	能耗	片上学习
Neurogrid[50]	数模混合	1	65k	100M	180	168	2.7W	否
BrainScaleS[51]	数模混合	352	512 (单核)	130k (单核)	180	50 (单核)	174 pJ/SOP	是
ROLLS[52]	数模混合	1	256	128k	180	51.4	4 mW	是
DYNAPs[53]	数模混合	4	1k	64k	180	43.79	17 pJ/SOP	否
TrueNorth[54]	数字电路	4096	1M	256M	28	430	65 mW (单核)	否
SpiNNaker[55]	数字电路	18	1k (单核)	1M (单核)	130	102	1 W (单核)	是
Loihi[56]	数字电路	128	131 040	130M	14	60	45 W	是
Tianjic[57]	数字电路	156	1-40000	1-10M	28	14.44	937 mW	否
Darwin[58]	数字电路	576	32768	1B	180	25	0.84 mW/MHz	否

目前主流的神经形态芯片通常采用数模混合电路或全数字电路实现，在架构设计上通过片上网络连接多个神经形态计算核心，形成大规模集成芯片。虽然这两种方法都利用先进的技术模拟出了上亿神经元规模的 SNN, 但是在制造工艺上，由于 CMOS 电路受限于二维连接和有限的互连金属及路由协议，复现真正的三维生物大脑结构仍然存在巨大困难，神经元之间的大规模突触连接和突触自适应的物理实现是目前主流架构的最大瓶颈。为了克服传统 CMOS 工艺存在的问题，选择具有理想性能的忆阻器正成为部署大规模 SNN 的解决方案之一[59]。

忆阻器与传统 CMOS 晶体管在基本功能上的主要区别在于它们是非易失性可编程模拟电阻器，这使得研究者们能够在底层器件的电阻状态中直接模拟神经形态计算中的神经元与突触行为，实现更具生物合理性的第 3 代神经网络。忆阻器 (Memristor) 的概念由 Chua 等人于 1971 年被提出[60]，它是电路中假设的第 4 种无源元件，将磁通量的变化与流经该元件的电荷变化联系起来。在数学上，它等同于一个非线性电阻，会根据电流的变化改变其电阻值，因此，它被称为忆阻器，即记忆电阻器。目前，已经有一些小型网络模型展示了基于忆阻器 NVM 阵列的高效运算，这种横杆阵列的结构为实现大规模并行和高能效的神经形态硬件平台提供了一种有效的途径。然而，受限于忆阻器系统本身的集成难度和仿生物突触学习规则的限制，基于忆阻器的神经形态计算仍处于利用器件进行原理探索与验证的阶段，大规模的忆阻器 SNN 暂时仍未被报道。

最后，根据选择的边缘计算平台，往往还需要设计相应的硬件映射方法，进而高效地将 SNN 模型部署在各类硬件平台上。尽管过去 20 年来，人们一直在研究将大脑运作原理映射到神经形态硬件上，但目前的主流硬件映射方法在资源分配、脉冲延迟，通信拥塞和能耗等性能指标上，仍然存在一系列瓶颈。

1.3 研究内容与贡献

本文以神经网络模型的边缘部署与推理为核心，通过模型层面的详细设计与一系列结合具体边缘计算平台的优化方法，实现了第二代神经网络代表——卷积神经网络和第三代神经网络代表——脉冲神经网络在低功耗边缘设备上的高性能推理设计。结合硬件平台特性的神经网络模型设计能有效提高模型的推理效率，实现智能化应用的高性能边缘部署，为 AI 技术普适化做出贡献。具体而言，本文的主要内容以及创新点如下：

1) 本文提出了一种基于 FPGA 平台的低功耗卷积神经网络加速器设计。针对人工智能领域经典的目标检测任务，将 YOLOv4-tiny 这一轻量化卷积神经网络模型部署在 PYNQ-z2 型号的 FPGA 开发板上。通过对模型进行剪枝，量化等轻量化操作，并针对 YOLOv4-tiny 模型中的核心卷积算子进行适合硬件部署的重构，再结合对 FPGA 硬件平台的内存，通信，带宽等机制的优化，实现了基于 FPGA 的低功耗卷积神经网络加速器设计。
2) 本文提出了一种基于国产 NPU 的高能效脉冲神经网络加速器设计。为了有效解决我国面临的高端 AI 设备断供风险，实现 AI 模型的高能效边缘部署，本设计基于华为昇腾 Atlas 200I DK A2 开发者套件，开创性地将第三代神经网络——脉冲神经网络部署在开发板上，结合开发板搭载的国产 NPU，实现高能效推理。
3) 本文设计了一个基于边缘计算平台的实时目标检测系统。该系统具有良好的交互性，操作简易，以 PYNQ-z2 为硬件平台，使用本文设计的 YOLOv4-tiny 加速器进行模型边缘部署，兼具速度与精度，能在毫秒级完成图片，视频，实时摄像头数据流的目标检测任务。

1.4 论文的组织结构

本文主要研究基于边缘计算平台的高能效神经网络加速器的设计。针对较为成熟的 FPGA 平台和卷积神经网络加速器，提出了一种性能表现优异的基于 FPGA 平台的低功耗卷积神经网络加速器设计方法。针对较为新颖的 NPU 平台和具有低功耗特点的脉冲神经网络，提出了一种基于国产 NPU 的高能效脉冲神

经网络加速器设计方法。最后，结合神经网络硬件加速器设计中提出的优化方案，搭建了基于边缘计算平台的实时目标检测系统。

全文一共分为六章，其结构如图1-1所示: 第一章内容为引言，主要介绍边缘智能的研究背景以及研究意义；第二章内容介绍了边缘智能领域的相关工作，包括神经网络基本原理和模型的轻量化部署方法，并简单介绍了神经网络加速器的基本原理；第三章内容介绍了一种基于 FPGA 平台的低功耗卷积神经网络加速器设计，通过软硬件协同优化在 FPGA 平台上高效部署了卷积神经网络模型；第四章内容介绍了基于 NPU 的高能效脉冲神经网络加速器设计，利用模型迁移技术将 SNN 模型从通用计算平台适配到国产 NPU 平台；第五章内容介绍了基于本文所提出的加速器设计方法实现的边缘实时目标检测系统；第六章内容总结全文并展望未来工作。

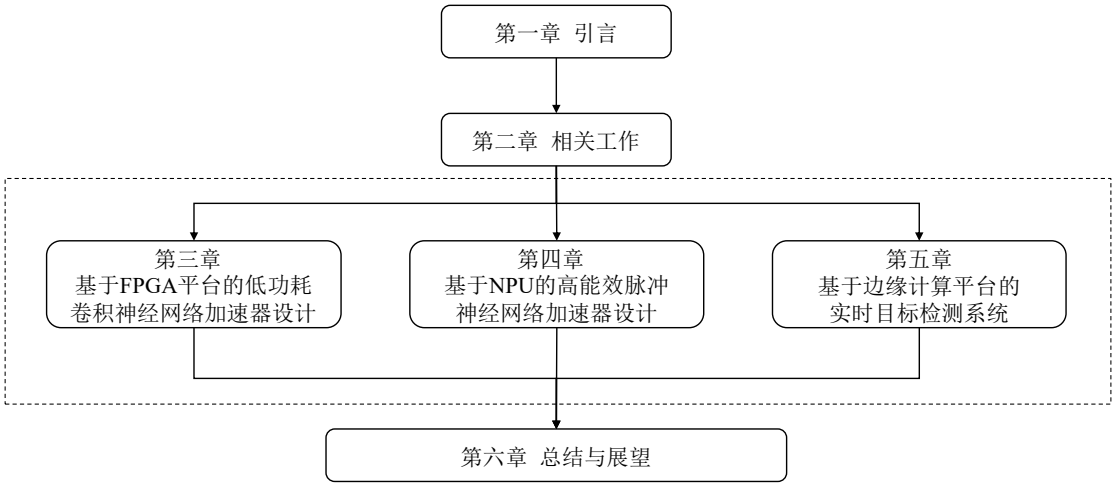

图 1-1 论文的组织结构

第二章　相关工作

2.1　相关神经网络模型结构

神经网络是一种模仿人脑神经元结构的机器学习模型，通过多层非线性变换从数据中学习复杂特征，广泛应用于图像识别、自然语言处理、语音识别等领域。

神经网络的基本组成结构是神经元，神经元是一个信息整合和加工的装置，完成信息处理后，通过神经元之间的突触实现信息的传递。如图2-1，每个神经元都会接受来自其他若干神经元的信号 x，每一个输入信号都有一个与之对应的突触，组成对应的 n 维向量 w，突触权值的大小反映了该输入信号对神经元的重要程度。对加权后的输入信号进行累加求和，即可得到激活电压，给定神经元激活阈值 θ，当激活电压大于激活阈值时，神经元将产生兴奋信号，反之产生抑制信号。最后，通过激活函数 σ 将输出信号映射到合适的范围内，再传递给其他神经元。

单个神经元的功能是有限的，扩展神经元的数量可以处理更加复杂的任务。多个神经元组合在一起便形成了神经网络的层，根据层的功能，可以分为输入层，隐藏层，输出层三种。输入层神经元接收来自外部的输入信息，并传递给隐

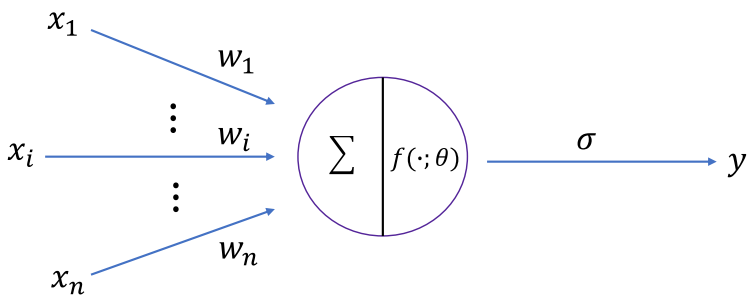

图 2-1　神经元结构示例

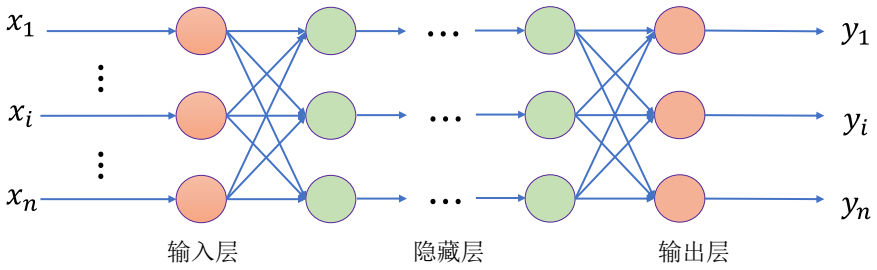

图 2-2　神经网络结构示例

藏层。隐藏层完成对信息的复杂处理，若只有一层隐藏层，则该神经网络可称为浅层神经网络。为了提高模型的表征能力，可以增加隐藏层的数量，即构造深度神经网络，此时的模型能容纳更多样本串联的知识。输出层完成对处理后信息的整合，最终将模型的结果输出到外部。神经网络的结构图如图2-2所示。

如果没有激活函数，神经网络将退化为线性模型，只能解决线性可分问题，无法处理实际问题中的复杂模式。因此，神经网络层之间还存在非线性激活函数，使其能够表示复杂的非线性关系。常见的激活函数包括 Sigmoid，Tanh，ReLU，他们的表达式如下所示：

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

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

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{当} x \geqslant 0, \\ 0 & \text{当} x < 0. \end{cases} \tag{2-3}$$

神经网络在运行时，包括前向传播和反向传播两个过程。前向传播通过层层前馈，计算模型的最终输出，反向传播则根据输出结果和期望结果的差异，反向传播更新神经元之间的连接权值，给予模型学习模式的能力。

目前，根据神经元的激活方式，可以将神经元分为基于乘积求和的人工神经元与基于脉冲激活的脉冲神经元两种。人工神经元是第二代神经网络的组成单

元，脉冲神经元是第三代神经网络的组成单元。

2.1.1 卷积神经网络

卷积神经网络 (Convolutional Neural Networks, CNNs) 是最具代表性的第二代神经网络，在 2012 年的 ImageNet 大规模视觉识别挑战赛中，AlexNet 深度卷积神经网络脱颖而出，以远超传统算法的优势赢得了图像分类任务的冠军，引发了 10 年代的人工智能热潮。相较于最基础的多层感知机神经网络，CNN 引入了“卷积层”这一核心结构，卷积层利用卷积核从输入特征图数据中提取信息特征。根据卷积核大小的不同，卷积层可以提取不同尺度的特征。为了提取多个维度的信息，卷积层使用多通道的卷积计算进行整合，累加所有输入通道的输入与卷积核的乘积得到最终输出，单个卷积核进行计算的过程如图2-3所示。

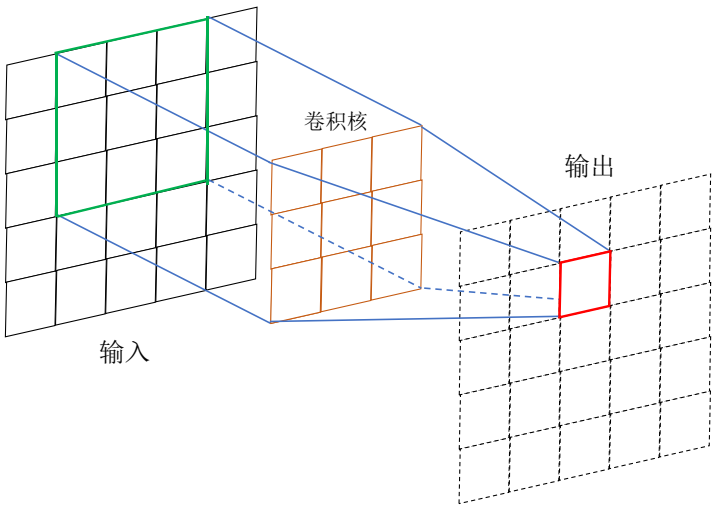

图 2-3 卷积计算过程示例

目前，卷积层普遍使用 3*3 大小的卷积核，既能减少模型计算量又能提取特征图中较大尺度的信息特征。随着 CNN 的广泛应用和理论发展，卷积计算也出现了许多变种，例如分组卷积 (Group Convolution)、转置卷积 (Transposed Convolution) 和空洞卷积 (Dilated/Atrous Convolution) 等。

池化层用于减少模型中特征图的尺寸并保留重要的特征信息，主要方法包括最大池化，平均池化和随机池化等。假设输入数据为 $X \in R^{H \times W}$，池化核大小为 $k \times k$，步长为 s，那么输出特征图的尺寸如式2-4所示：

$$H' = \left\lfloor \frac{H-k}{s} \right\rfloor + 1, W' = \left\lfloor \frac{W-k}{s} \right\rfloor + 1. \tag{2-4}$$

最大池化（Max Pooling）：对于特征图的每个局部区域，选择其中的最大值作为该区域的输出。

$$Y_{i,j} = \max_{m=0}^{k-1} \max_{n=0}^{k-1} X_{i\times s+m, j\times s+n}, \tag{2-5}$$

平均池化（Average Pooling）：对于特征图的每个局部区域，计算其平均值作为该区域的输出。

$$Y_{i,j} = \frac{1}{k^2} \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} X_{i\times s+m, j\times s+n}, \tag{2-6}$$

随机池化（Stochastic Pooling）：在特征图中基于激活值的概率分布随机采样，可根据各元素的数值大小生成概率分布，通过采样确定输出值。

$$P_{i,j} = \frac{X_{i,j}}{\sum_{m=0}^{k-1} \sum_{n=0}^{k-1} X_{i,j}}, \tag{2-7}$$

池化操作可以有效降低数据的维度，减少计算量，并防止过拟合，随机池化引入的随机性还能在一定程度上提高模型的泛化能力。

全连接层通过特征提取实现最终输出的分类或回归。在全连接层中，所有的输入均和神经元相连接，每个连接都是一个可以学习的权值参数，学习特征之间的复杂关系。

综上所述，卷积神经网络的整体结构可以用图2-4表示。

CNN 能通过局部感知和权值共享提取图像中的局部特征，并通过多层结构逐渐学习到从低级到高级的特征。这种组合的层级结构能使 CNN 捕捉到图像中的复杂模式。此外，池化操作能有效降低计算量并增强对平移、旋转等变化的鲁棒性。目前，CNN 是解决计算机视觉问题的核心工具之一。

2.1.2　脉冲神经网络

脉冲神经网络是一种受到生物神经系统启发的高能效计算模型，其模型基本原理包括脉冲神经元、脉冲编码机制、SNN 的学习训练方式以及网络整体的拓扑结构。

脉冲神经元是构成 SNN 的基本单元，使用离散脉冲序列进行神经元间通信，

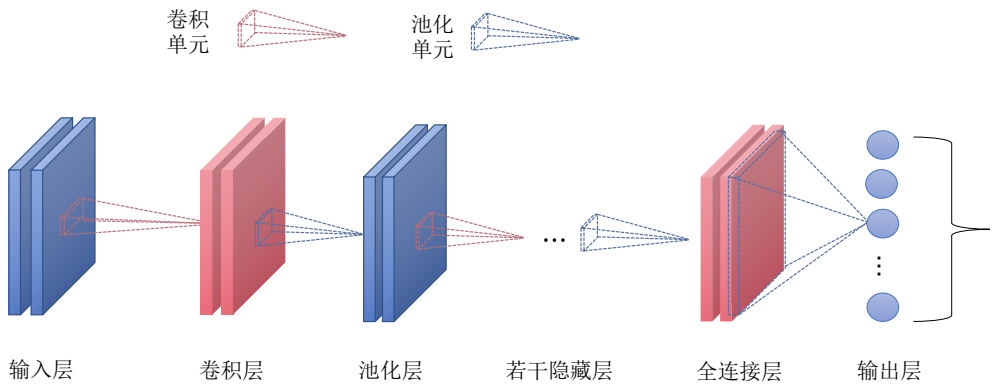

图 2-4 卷积神经网络结构示例

在机制上和生物神经元是类似的，拥有优秀的生物合理性。脉冲神经元之间的突触连接并不总是活跃的，因为神经元只能发射或不发射脉冲。这种现象被称为“脉冲稀疏性”，能让计算次数减少，进而降低功耗和延迟。Hodgkin-Huxley (H-H) 模型是第 1 个描述脉冲神经元动作电位如何启动和传播的生物模型[61]，也是生物学上最合理的脉冲神经元模型，能准确捕捉许多真实神经元的动态，H-H 模型给出了通过膜电位的电流的数学描述，可以用以下公式进行计算:

$$I = C\frac{\mathrm{d}V}{\mathrm{d}t} + G_{\mathrm{Na}}m^3h\left(V - V_{\mathrm{Na}}\right) + G_{\mathrm{k}}n^4\left(V - V_{\mathrm{k}}\right) + G_{\mathrm{L}}\left(V - V_{\mathrm{L}}\right). \tag{2-8}$$

式中，I 为外部电流，C 为电路的电容；V_{Na}、V_k 和 V_L 称为逆电位；G_{Na}、G_k 和 G_L 分别是模拟钠、钾和泄漏通道电导的参数。门控参数 n 控制钾通道，m 和 h 控制钠通道。这些参数可由公式2-9得到, 其中 α 和 β 代表对应的粒子向膜内 (外) 移动的速率。

$$\begin{aligned}
\frac{\mathrm{d}m}{\mathrm{d}t} &= \alpha_m(V)(1-m) - \beta_m(V)m, \\
\frac{\mathrm{d}n}{\mathrm{d}t} &= \alpha_n(V)(1-n) - \beta_n(V)n, \\
\frac{\mathrm{d}h}{\mathrm{d}t} &= \alpha_h(V)(1-h) - \beta_h(V)h.
\end{aligned} \tag{2-9}$$

虽然 H-H 模型在生物学上非常还原，但目前的学习算法往往在更简单的脉冲神经元模型上表现更好。Integrated & fire (IF) 脉冲神经元是一种针对阈下电位

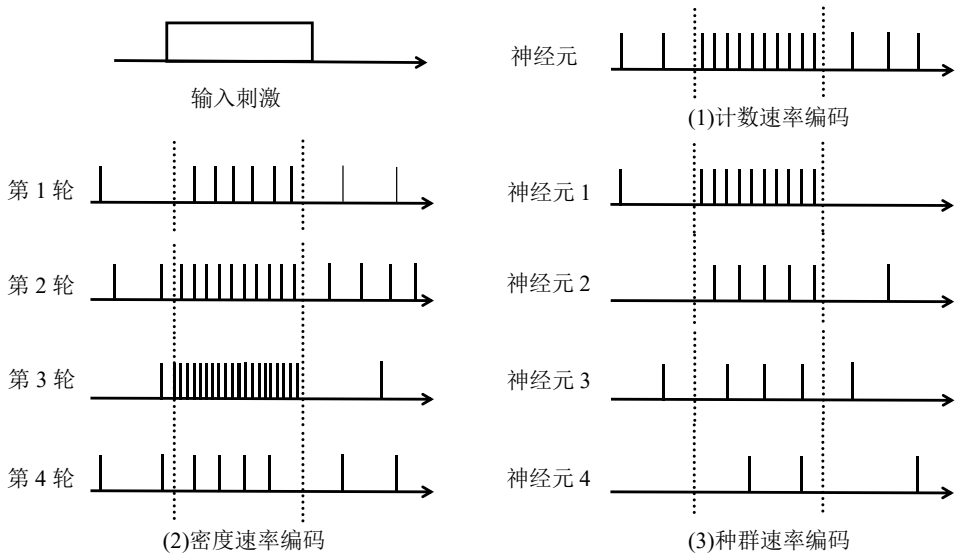

图 2-5　速率编码

的变化规律进行描述的简单模型，它将输入脉冲信号整合到膜电位中，如果达到定义的阈值，则产生输出脉冲，膜电位再恢复到静息状态。Izhikevich 模型是对 H-H 模型简化后的二维模型[62]，可以再现各种各样的脉冲行为，在生物合理性和计算效率之间达到了很好的平衡。脉冲响应模型 (Spike Response Model, SRM) 是一种生物激发的脉冲神经元，它更精确地描述了输入脉冲对膜电位的影响。与 LIF 模型类似，SRM 模型在其内部膜电位达到阈值时产生脉冲[63]，不过，SRM 模型包含了对于不应期的模拟，且使用的是滤波器而非微分方程来描述神经元行为。

SNN 与第二代神经网络最显著的区别除了神经元的不同，还在于信息的编码和解码方式。在 SNN 中，数据都是以脉冲序列的形式进行传输，因此需要将图像像素或实数这样的模拟量用二进制信号进行编码。编码机制决定了用脉冲表示模拟值时的量化或转换误差，目前使用最广泛的编码方法是速率编码[64]和时间编码[26]。

速率编码使用相应记录时间内脉冲序列的发射速率来编码信息，实际输入数字被转换成频率与输入值成正比的脉冲序列，被视为对神经元输出的一种量化衡量。速率编码可以进一步分为 3 种类型: 计数速率编码、密度速率编码和种群速率编码，图2-5展示了速率编码的可视化案例。

时间编码通过单个脉冲的相对时间对信息进行编码，输入值被转换成具有精确时间的脉冲序列，常用于时间序列处理。时间编码方案包括首脉冲触发时间

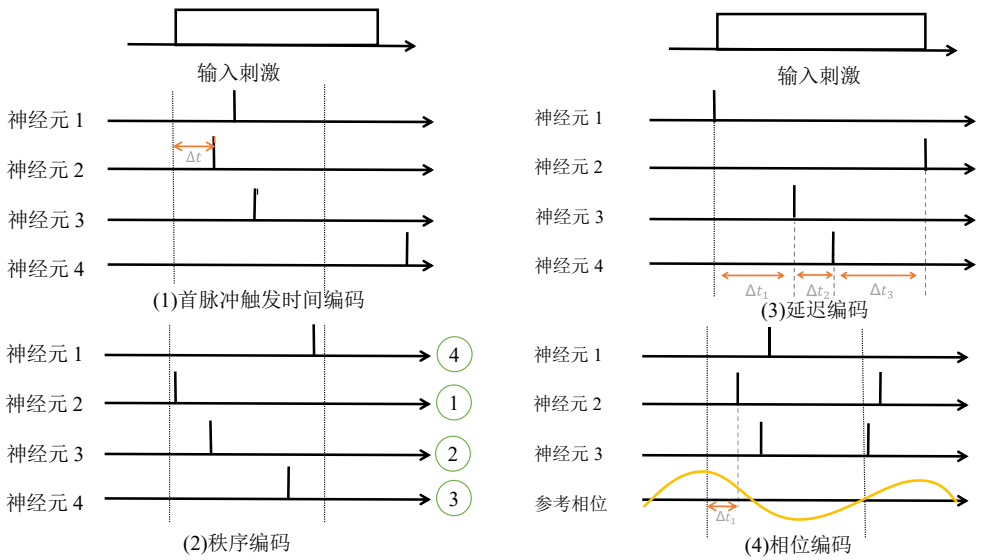

图 2-6 时间编码

表 2-1 速率编码和时间编码在硬件实现中的性能比较

编码方案	复杂度	鲁棒性	信息量	功耗	速度	反应时间	训练难度
速率编码	低	高	少	高	慢	长	低
时间编码	高	低	多	低	快	短	高

编码，秩序编码，延迟编码，相位编码等。图2-6展示了时间编码的可视化案例。

表2-1简单给出了速率编码和时间编码在边缘部署时的性能比较。总的来说，速率编码以神经元的发射速率对信息进行编码，不依赖于每个脉冲事件的精确时间，也不需要额外的训练信息，方法相对简单，有更好的鲁棒性和抗扰性。时间编码通过神经元发出脉冲的精确定时进行编码，可以提供更大的信息容量、更快的反应时间和更高的传输速度，有更低的功耗和更快的计算速度，但是结构更为复杂，且拥有更高的训练门槛。

值得一提的是，速率编码和时间编码拥有各自的独特优势，组合这些方案可能对系统性能产生巨大影响，这种组合方案被称为混合编码。混合方式可以是神经编码方案在网络层之间变化，也可以是神经元在编码方案之间切换，文献[65]提出了一种多层编码方案，脉冲序列根据时间尺度在不同编码方案的不同通道中表示信息。另一方面，由于没有一种适用于所有神经形态应用的通用编码方法，因此在实际环境下所使用的编码方案也视情况而定。为了解决多种编码方法的兼容问题，文献[66]提出了第 1 个基于硬件的脉冲编码器，支持多种脉冲编码方案和运行时配置，并允许集成到各类神经形态硬件中，为 SNN 在边缘部署

时的脉冲编码方案选择提供了新的思路。

在 SNN 中，学习是一项艰巨的任务，由于脉冲事件的不可微性，SNN 无法直接应用传统的 BP 训练算法进行学习，也没有公认的核心训练算法。目前，训练 SNN 主要有 3 种策略: 无监督学习、监督学习和 ANN-to-SNN 转换法。

SNN 的无监督学习基于 Hebb 规则[67]，该规则会使网络的突触连接适应神经元接收到的数据。脉冲时间相关的可塑性 (Spike-Timing Dependent Plasticity, STDP) 算法[68]是 Hebb 规则的一种实现，基于 STDP 的学习规则根据各自脉冲时间的相关程度，修改连接一对突触前和突触后神经元的突触权重。其中，对神经元发射脉冲有贡献的突触应该得到加强，对那些没有贡献或以消极方式贡献的突触应该被削弱。最常见的 STDP 规则由公式2-10描述。

$$\begin{cases} \Delta\omega = \begin{cases} +A_{+}\exp\left(\frac{-\Delta t}{\tau}\right), & \Delta t > 0 \\ -A_{-}\exp\left(\frac{+\Delta t}{\tau}\right), & \Delta t \leqslant 0 \end{cases} . \\ \Delta t = t_{\text{post}} - t_{\text{pre}} \end{cases} \tag{2-10}$$

其中，ω 为突触权重，τ 为时间常数，A_{+} 和 A_{-} 为表示增强和抑制强度的参数。

由于脉冲神经元不可微的特性，在 SNN 中直接实现基于梯度的反向误差传播训练方法十分具有挑战性。为了规避脉冲神经元动力方程不可微的问题，有研究将不连续导数近似为连续函数的代理梯度并用于训练端到端反向传播的 SNN[69]，这种代理梯度的思想是用连续伪导数逼近脉冲函数的不连续梯度。

除了代理梯度法外，SNN 的有监督训练还包括基于脉冲的梯度下降法，该方法通过时间反向传播 (Back-Propagation Through Time, BPTT) 执行信用分配，结合脉冲序列中的时间信息实现更低的推理延迟[70]。

ANN-to-SNN 转换法从拥有成熟训练方法的 ANN 中学习，快速得到性能良好的 SNN。在 ANN-to-SNN 转换法中，因为 ReLU 神经元在功能上等同于没有任何泄漏和折射期的 IF 脉冲神经元，所以通常会选择 ReLU 神经元作为待转换 ANN 的基本神经元。在转换过程中，首先训练 ReLU 神经元组成的 ANN, 训练完成后，用 ANN 的权重初始化具有 IF 神经元和等结构的 SNN, 图2-7展示了转换过程中 ReLU 神经元映射为 IF 脉冲神经元的原理。

SNN 的拓扑结构直接反映了神经元和突触之间的连接方式。根据训练时网

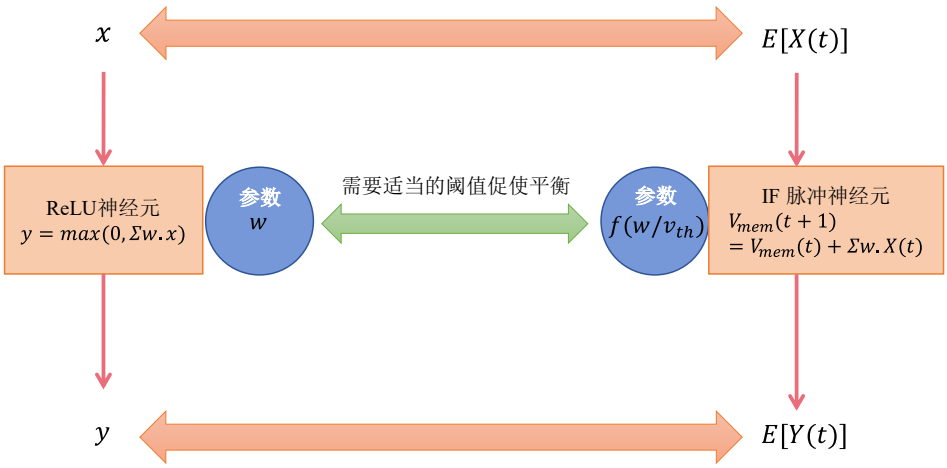

图 2-7 将 ANN 神经元转化为 SNN 神经元

络拓扑是否发生变化，可以将 SNN 的拓扑分为静态拓扑和动态拓扑。静态拓扑中脉冲神经元数量和突触连接方式在训练时一直保持不变，改变的只有突触之间的连接权重。常见的静态结构包括前馈网络结构和循环网络结构。动态拓扑中脉冲神经元数量和突触连接方式在训练时会动态变化，典型代表为进化脉冲神经网络 (Evolutionary Spiking Neural Network, ESNN)[71]。ESNN 的设计思想来源于生物学的互联进化系统，它能够以自适应、自组织、在线连续的方式动态改变系统的结构和功能。

2.2 模型轻量化部署方法

2.2.1 模型压缩算法

模型压缩是深度神经网络模型的重要优化方法，其核心目标是在保证模型性能的前提下，降低模型的存储开销和计算复杂度，使其能够在资源受限的设备上高效部署。目前的两大主流方法分别是量化和剪枝。

量化通过降低模型参数和激活值的数值精度，减少内存占用并加速计算。其核心思想是将高精度浮点数（如 32 位浮点数）转换为低精度整数（如 8 位整数），从而显著降低模型的存储和计算需求。

以将 32 位浮点数量化为 8 位整数为例，量化操作的基本原理如式2-11所示，其中，Δ 为量化步长，Z 为零点偏移。

$$Q(x) = \text{round}\left(\frac{x}{\Delta}\right) + Z, \tag{2-11}$$

剪枝通过移除模型中冗余的权重或神经元，生成稀疏化网络，从而减少参数数量和计算量。剪枝方法包括非结构化剪枝和结构化剪枝两类，非结构化剪枝是细粒度的方法，针对权值进行操作；结构化剪枝是更粗粒度的方法，针对滤波器甚至整个隐藏层进行剪枝。

假设原始的权值矩阵为 $\boldsymbol{W}$, 非结构化剪枝通过逐元素掩码操作保留重要权值，保持维度但引入稀疏性，设掩码矩阵为 $\boldsymbol{M} \in \{0,1\}^{m\times n}$, 则经过非结构化剪枝后的权值矩阵可表示为：

$$\boldsymbol{W}_{\text{pruned}} = \boldsymbol{W} \odot \boldsymbol{M}. \tag{2-12}$$

其中 $\odot$ 表示逐元素相乘，掩码矩阵可根据设定的阈值 θ 生成，例如：

$$\boldsymbol{M}_{i,j} = \begin{cases} 1 & \text{if } \left|\boldsymbol{W}_{i,j}\right| \geqslant \theta, \\ 0 & \text{otherwise}, \end{cases} \tag{2-13}$$

结构化剪枝通过移除网络模型中某个整体结构来缩减简化计算图，直接改变了网络维度。以对深度卷积神经网络模型进行结构化剪枝为例，选择滤波器这一结构级别为剪枝对象，并假设网络中的某层计算结构包括输入数据集 $\boldsymbol{D} = \left\{\left(x_i, y_i\right)\right\}_{i=1}^{N}$, 损失函数 $\boldsymbol{L}(\cdot)$ 和 $\boldsymbol{N}$ 个三维滤波器 $\boldsymbol{F}$, 那么结构化剪枝的目标可表示为：

$$\begin{gathered} \min_{\boldsymbol{F}} \boldsymbol{L}(\boldsymbol{F}; \boldsymbol{D}) = \min_{\boldsymbol{F}} \frac{1}{\boldsymbol{N}} \sum_{i=1}^{N} \boldsymbol{L}\left(\boldsymbol{F}; \left(x_i, y_i\right)\right) \\ \text{s.t. } \text{Card}(\boldsymbol{F}) \leqslant \alpha \end{gathered}. \tag{2-14}$$

其中 Card(·) 是滤波器集合的基数，α 是设定的剪枝稀疏率，即最多可保留的滤波器数量。

对神经网络模型进行剪枝的基本原理包括重要性评估，剪枝策略的选择和剪枝后微调。重要性评估根据权值幅度、梯度信息或特征贡献度，判断参数重要性。剪枝策略需要决定剪枝时的具体操作，是移除不重要的参数或是直接删除对应结构。微调则对剪枝后的模型重新训练以恢复性能。

在实际应用中，量化与剪枝常联合使用以进一步提升压缩效率，通常先剪枝减少参数量，再对稀疏模型进行量化。例如 TinyBERT[72]通过剪枝 + 量化可将

BERT 模型压缩至 1/7 大小，精度损失小于 3%。

2.2.2 网络算子重构

网络算子重构是一种通过优化网络基础算子的计算方式或结构设计，降低模型复杂度的方法。通过重新设计计算密集型的算子，可以在保持模型性能的前提下，减少参数量、计算开销或内存访问次数，进而提升硬件部署时的执行效率。典型的算子重构方法包括低秩分解，分组卷积，算子融合等。

低秩分解使用矩阵分解的思想，将高秩张量分解成低秩张量，来减少模型的大小和内存的访问次数。这种方法可以应用到卷积层和全连接层上，并且能与量化方法结合。对于给定的权值矩阵 $W \in R^{m\times n}$，如果可以按照式2-15将其表示为若干个低秩矩阵的和，即

$$W = \sum_{i=1}^{n} M_i, \tag{2-15}$$

其中 $M_i \in R^{m\times n}$ 为低秩矩阵，其秩为 r_i，并满足 $r_i \leqslant \min(m,n)$，那么每个低秩矩阵都可以分解两个小规模矩阵的乘积，如式2-16所示，其中 $G_i \in R^{m\times r_i}$，$H_i \in R^{r_i\times n}$。如果 r_i 取到较小值，那么能大幅减少矩阵计算过程的资源占用。

$$M_i = G_i H_i, \tag{2-16}$$

低秩分解方法数学可解释性强，参数压缩率高，但计算和过程较为复杂，尤其是在处理大规模数据时。

分组卷积将输入通道分组，每组独立进行卷积操作，从而显著减少单次计算量。式2-17展示了标准卷积的计算复杂度，用乘加运算次数 (Multiply-Accumulate Operations, MACs) 来表示。若分为 G 组，则每组计算复杂度将变为原先的 $\frac{1}{G}$。

$$\text{MACs} = H_{\text{out}} \cdot W_{\text{out}} \cdot C_{\text{in}} \cdot C_{\text{out}} \cdot K^2, \tag{2-17}$$

深度可分离卷积是一种典型的分组卷积方法，将标准卷积分解为逐通道卷积（Depthwise Conv）与逐点卷积（Pointwise Conv）两步。逐通道卷积仅在单个输入通道进行卷积计算，而逐点卷积在所有输入通道进行卷积计算并累加。式2-18给

出了逐通道卷积和逐点卷积的复杂度。

$$\begin{aligned} \mathrm{MACs}_{\mathrm{depthwise}} &= H \times W \times C_{\mathrm{in}} \times k^2, \\ \mathrm{MACs}_{\mathrm{pointwise}} &= H \times W \times C_{\mathrm{in}} \times C_{\mathrm{out}}. \end{aligned} \tag{2-18}$$

目前，深度可分离卷积算子是 MobileNet 系列、EfficientNet 的核心组件。

算子融合技术将多个连续算子合并为单一计算单元，减少内存访问与中间结果存储，非常适合资源有限的边缘计算平台部署神经网络模型。如图2-8所示，以卷积神经网络为例，其隐藏层中包含卷积层，批归一化层和激活函数，通过数学等价变换，可以得到单一融合层，将内存占用减少 30%~50%，推理速度提升 20% 以上。

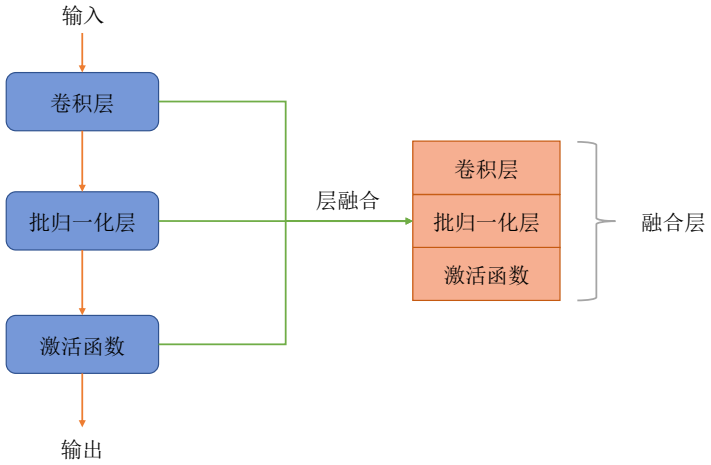

图 2-8　层融合原理示例

2.3　神经网络加速器设计

神经网络加速器是专为高效执行神经网络计算任务而设计的软硬件协同系统，旨在提升计算速度、降低功耗并优化资源利用率。相比于通用计算平台，定制化的神经网络加速器可以针对网络模型的具体细节进行硬件级定制优化，从而成为高效的深度学习应用加速平台。相比于传统的 GPU 加速器，定制化的神经网络加速器通常具有更低的功耗，在资源受限时提供高性能，从而将智能应用部署在边缘设备上。

2.3.1 加速器基本架构

在神经网络加速器的设计中，需要通过并行计算单元最大化吞吐量，结合硬件平台的实际特性，优化数据局部性，减少片外内存访问，将神经网络各算子高效结合，在性能与功耗之间达到合适的平衡。目前，神经网络加速器主要的硬件部署平台包括 FPGA, NPU 等。

基于 FPGA 的神经网络加速器是目前一种主流的加速器，FPGA 平台具有大量的可编程逻辑单元、乘法器和存储块，能大规模并行支持神经网络的计算。相比于传统的 GPU 加速器，FPGA 具有更低的功耗，并能在功耗受限的情况下提供高性能。此外，由于 FPGA 的可编程性，基于 FPGA 的加速器能够适应不同的神经网络模型和算法，适合对网络结构进行迭代优化，通过定制化设计，获得更好的性能。

基于 NPU（Neural Processing Unit，神经处理单元）的加速器是另一种新兴的神经网络加速器。NPU 是一种专为神经网络计算设计的专用硬件芯片，其核心目标是通过定制化架构优化矩阵乘加、张量运算等深度学习核心操作，显著提升能效比和计算吞吐量。相较于 CPU，GPU 等通用处理器，NPU 在架构设计上深度融合了神经计算特性，已成为移动端及边缘计算场景的主流加速方案。目前，NPU 在许多边缘设备上都有搭载，例如智能手机，智能汽车，智能家居，开发板等。NPU 芯片在设计时通常会采用大规模并行阵列的思路，并针对神经网络算子进行专门的优化，以便为神经网络模型提供更高的计算效率。此外，NPU 采用定制的指令集和优化的电路设计，在保持高性能的同时，能够降低能耗，达到较低的计算延迟，部分 NPU 还在硬件级别支持非结构化剪枝后的稀疏权值压缩与零值跳过。

图2-9给出了神经网络加速器的基本架构，该架构既包含神经网络模型，输入输出数据和中间计算结果等软件层面的要素，也包括片上存储空间，并行处理阵列和中央处理器等硬件层面的基本组成原件，是一种软硬件协同运行的系统。神经网络模型和数据作为输入进入硬件模块，在中央处理器的整体调度下，经过输入缓冲区被加载到硬件平台的片上存储空间中。当进行具体的运算任务时，作为计算核心的并行处理阵列会不断与片上的存储块进行实时的内存交换。待一轮计算推理结束后，最终的推理结果将经过输出缓冲区，并采取必要的后处理

操作，把本次运算的推理结果展示出来。

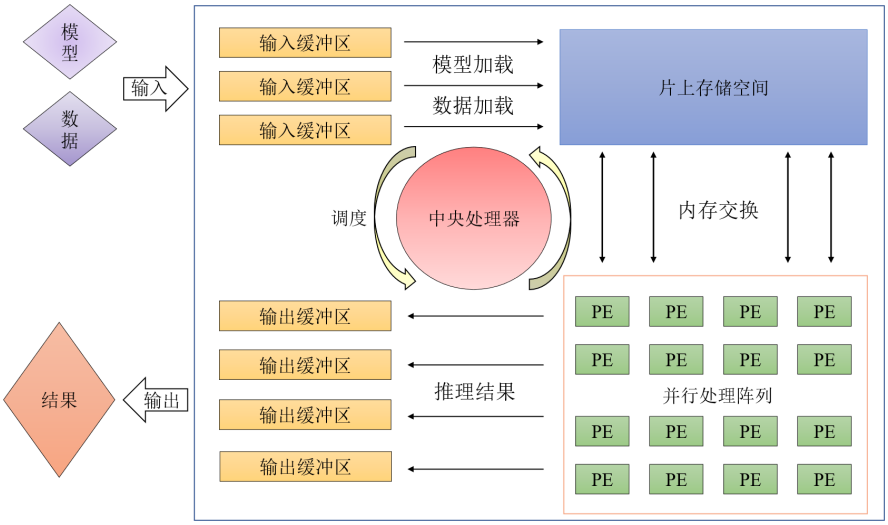

图 2-9　神经网络加速器的基本架构

2.3.2　数据流架构

从图2-9还可以看出，神经网络加速器的基本架构中包含了大量的数据流。由于神经网络计算的核心是密集的矩阵乘加操作，因此其性能高度依赖数据流的优化，高效的数据流处理方式能够显著提升加速器的计算效率并降低功耗。目前，主流的数据流处理方式包括流式数据流和重叠式数据流。

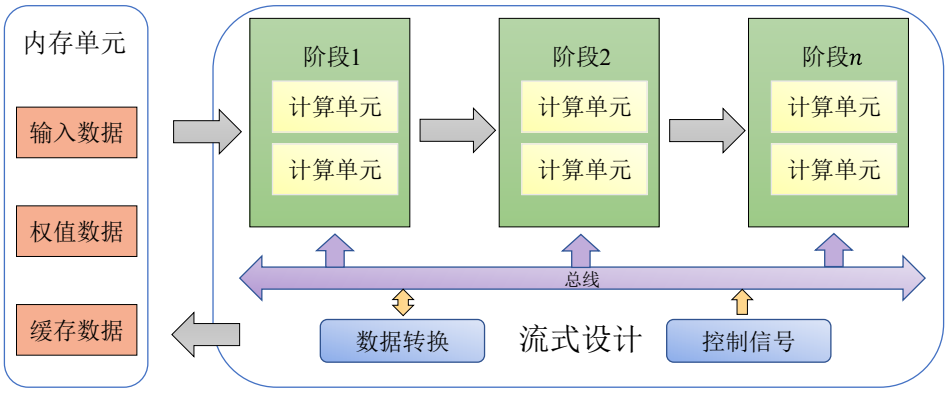

图 2-10　流式数据流

流式数据流的设计架构如图2-10所示，这种数据流处理方式为神经网络模型的计算流程中的每个计算部分实现了不同的硬件单元，并分别对其进行优化，以

利用块间并行性。计算单元按照模型算法的不同阶段顺序连接，每个阶段处理完成后立即将结果传递给下一阶段，实现数据的流式传输。这种设计模式的优点在于可以将复杂的模型分而治之，根据不同计算部分的特性差异，进行针对性的优化设计。然而，考虑到神经网络模型硬件部署时可能遇到的环境和配置等方面的棘手问题，这种设计模式会导致灵活性低，难以适配。

重叠式数据流的设计架构如图2-11所示，这种数据流处理方式的核心思想是设计一个大型可重用引擎来执行模型的运算步骤，并由上层软件执行硬件控制和操作调度。由于所有的运算步骤都交给可重用引擎处理，因此加速器可以根据输入模型和可用的硬件资源进行动态扩展，具备更好的灵活性和适配性。这种设计模式的缺点则在于数据流控制方式类似于通用处理器，因此不容易实现计算的最大效率。

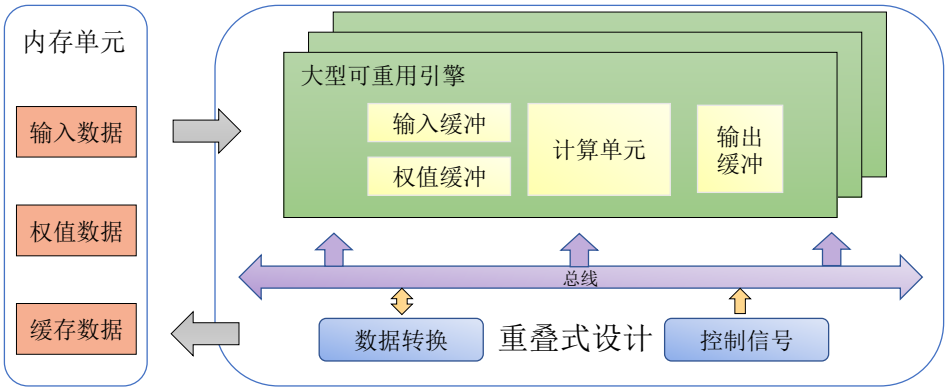

图 2-11 重叠式数据流

根据深度神经网络模型拥有大量结构相似的隐藏层的特点，可以考虑将流式数据流和重叠式数据流的优点相结合，即为神经网络模型的每个计算步骤 (算子) 设计独立的处理单元，不同算子对应的处理单元之间是流式传输，对应神经网络推理时的前向传播，可实现各算子的高效执行。而对于网络模型中结构相似的隐藏层，则可以将这些算子对应的处理单元进行跨层复用，提高加速器的兼容性。以卷积神经网络为例，这种混合式数据流的结构如图2-12所示。

由于深度卷积神经网络中的隐藏层大部分都是卷积算子、池化算子和全连接算子的组合，因此存在大量的算子可重用空间。对于网络的前向传播推理，实时处理的数据流在不同算子之间以流式高效传输，而在不同隐藏层之间，则可根

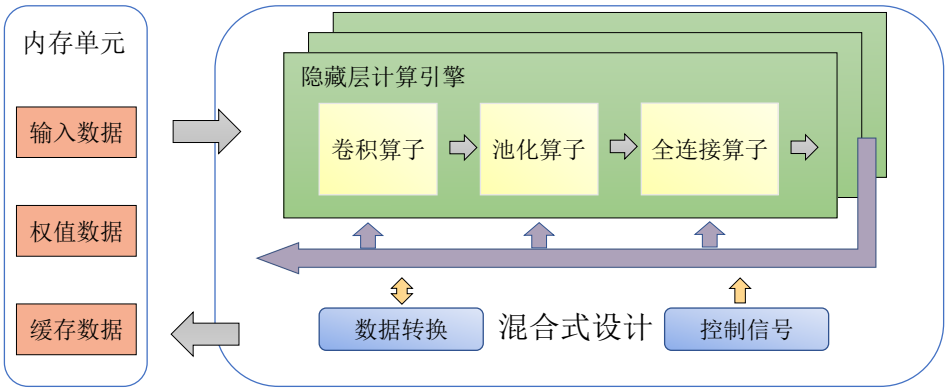

图 2-12　混合式数据流

据网络模型的隐藏层配置，重叠使用相同的算子。

2.4　本章小结

本章主要介绍了构建高能效神经网络加速器时采取的神经网络模型结构，硬件部署时涉及的轻量化方法以及加速器设计的基本架构。

对于相关神经网络模型结构，本章简单介绍了第二代神经网络的代表——卷积神经网络和第三代神经网络的代表——脉冲神经网络的基本原理；对于模型轻量化部署方法，主要介绍了模型压缩算法和网络算子重构技术；对于神经网络加速器设计，概述了加速器的基本架构与数据流架构。

目前，针对高能效神经网络的研究意义非凡，这类研究有助于将前沿的 AI 模型部署在资源功耗受限的边缘设备上，进而实现万物智能。针对不同的边缘计算平台和神经网络模型，神经网络加速器的设计方法也不尽相同，相关研究的主要问题仍聚焦于性能与资源的权衡，以达到更高的能效。此外，随着硬件设备的不断更新迭代，如何有效地将成熟的神经网络模型迁移部署到新兴的硬件平台上，也是一个值得关注的问题。后文详细介绍的加速器设计方法将以这些相关工作为基础，为这些问题提供解决思路。

第三章　基于 FPGA 的低功耗卷积神经网络加速器设计

在卷积神经网络的推动下，计算机视觉任务逐步从图像分类拓展至目标检测与实例分割。然而，传统两阶段检测框架因区域建议生成与特征重复计算导致的效率问题，严重限制了其在实时场景中的应用。为解决这一矛盾，Redmon 等人于 2016 年提出 YOLO 模型，首次将目标检测任务简化为单次网格化回归问题。YOLO 以 CNN 骨干网络提取的层次化特征为基础，通过在特征图上直接预测边界框坐标、置信度及类别概率，实现了检测速度的数量级提升。

以 YOLOv4 模型为例，模型包含多个卷积层，每个卷积层都有大量的卷积核参数需要存储。从最初的 YOLOv1 模型发展而来, 模型经过了多次迭代和改进, 逐渐提高了检测的精度和速度。然而，模型的复杂度也随之逐渐增加，卷积核的数量和大小也相应增大，导致模型参数的存储量大幅增加，很难直接部署在资源有限的边缘计算平台上。

YOLOv4-tiny 是 YOLOv4 的精简版，属于轻量化模型，参数只有 6M 左右，相当于原来的约十分之一，非常适合边缘部署。考虑到模型中的卷积、池化等操作需要大量的乘法和加法运算，而 FPGA 平台具有大规模的并行逻辑单元，因此非常适合同时对多个卷积核进行卷积计算或者对不同通道的特征图进行并行处理，能有效提高计算效率。

为了进一步提高视觉模型在边缘计算平台的推理性能，在不影响 YOLOv4-tiny 模型检测效果的前提下，本章结合软件模型层面和硬件部署层面的联合优化，以推理前轻量化，推理时高性能为核心思想，通过模型压缩，算子重构和数据流优化等方法，设计了一种基于 FPGA 平台的低功耗卷积神经网络加速器。

3.1　网络结构与加速器设计

图3-1展示了 YOLOv4-tiny 的模型结构，整体分为三部分：

骨干网络（Backbone）：骨干网络采用轻量化的 CSPDarknet-tiny 结构，共包

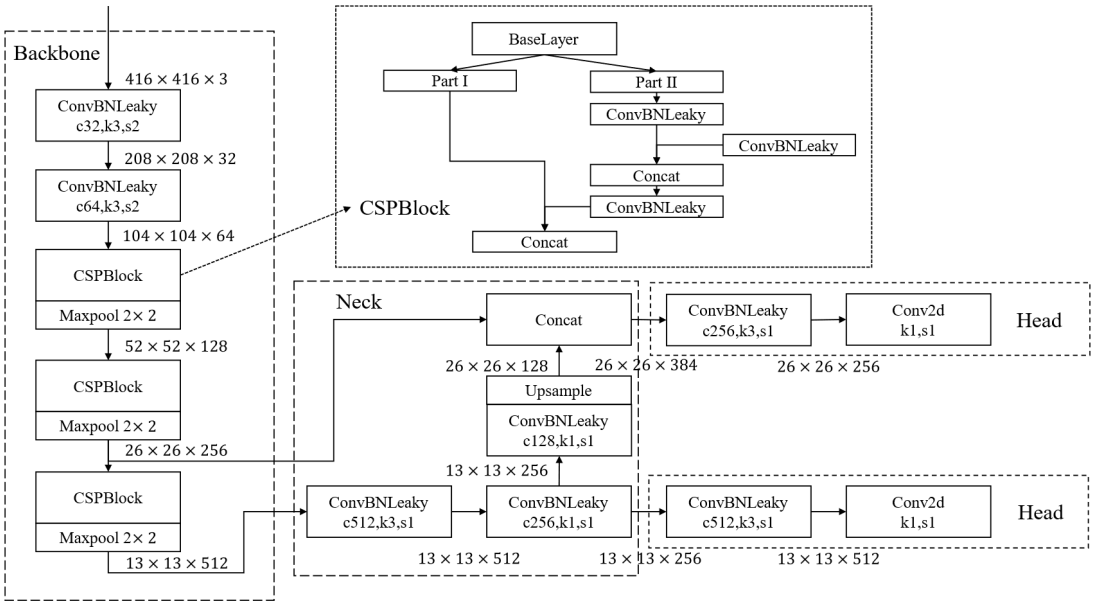

图 3-1　yolov4-tiny 结构图

含 38 层，其中集成 3 个残差单元，每个单元由多个卷积块构成，每个卷积块采用 Conv+BN+LeakyReLU 组合，计算时通过分治策略将特征图划分为独立路径，分别进行卷积和残差操作，最后通过通道拼接实现特征复用。

颈部（Neck）：颈部作为特征增强网络，采用特征金字塔网络融合不同层级的语义信息，通过上采样操作将高层语义特征与底层细节特征结合，提升小目标检测能力 。

检测头（Head）：检测头使用两个不同尺度的特征层，分别负责中小目标的检测，每个检测层包含分类与回归分支，通过 1×1 卷积输出目标类别概率和边界框坐标。

针对 YOLOv4-tiny 原始模型结构，设计了包括数据处理模块，通信调度模块，计算引擎和输出后处理模块四部分的基于 FPGA 的低功耗卷积神经网络加速器。根据所选 FPGA 平台的硬件资源分布，加速器在硬件架构上可分为逻辑计算资源 (PL) 端和通用处理器 (PS) 端两部分。根据加速器的逻辑功能划分，整体设计架构如图3-2所示，数据处理模块使用量化，剪枝等操作后，将经过轻量化处理的模型与数据送入硬件平台。在推理过程中，通信调度模块负责数据通信与控制通信，计算引擎作为加速器的核心部分，利用优化后的网络算子完成数据计算，并把推理结果交给后处理模块完成可视化。各模块的详细功能如下：

数据处理模块：该模块负责数据的预处理和传输，完成输入数据和权值从片

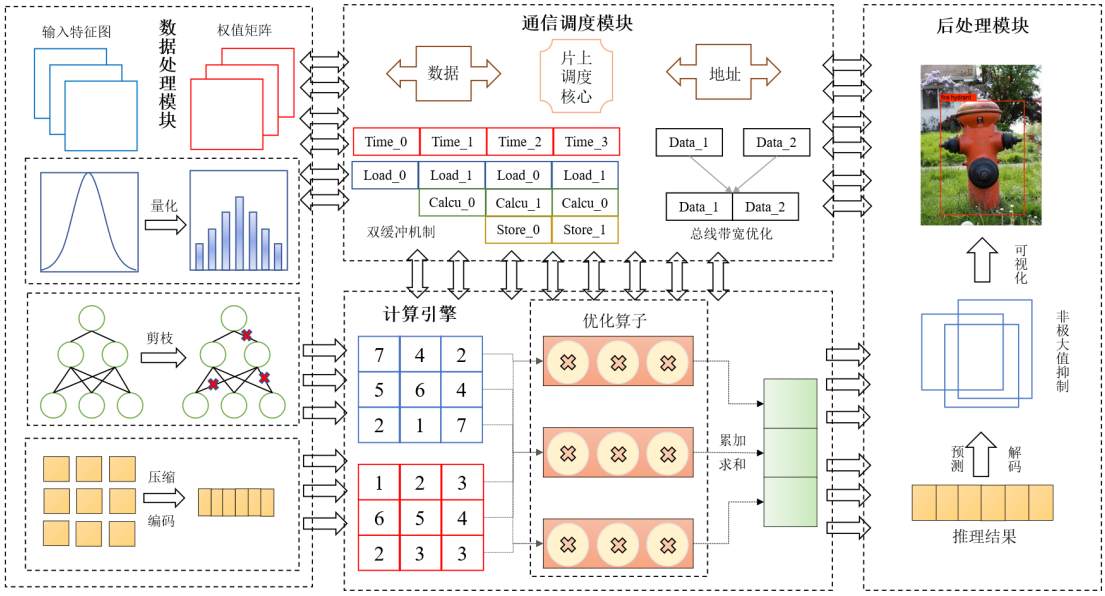

图 3-2 加速器整体设计架构

外存储到片上缓存的搬运，在传输时通过对各类数据的轻量化处理，优化整体的数据传输效率。

通信调度模块：该模块负责数据流在加速器中的调度，通过对片上缓冲单元的高效利用和对总线带宽使用的优化，提高数据流在计算引擎和片外存储间的通信效率。

计算引擎：该模块为加速器的核心处理部分，将原始模型的骨干网络配置为可以重用的卷积算子，利用 FPGA 高度并行的逻辑资源执行稠密的乘累加操作，完成高效的特征提取与计算。

后处理模块：对计算引擎输出的原始预测结果进行解码、筛选和优化，最终生成符合需求检测框和类别信息。

3.2 模型轻量化处理

在加速器的整体设计中，数据处理是计算前的关键步骤。对原始模型进行轻量化处理，使模型整体的资源占用更少，进而降低加速器的整体功耗。在轻量化处理部分，包括模型剪枝和权值量化两个步骤。

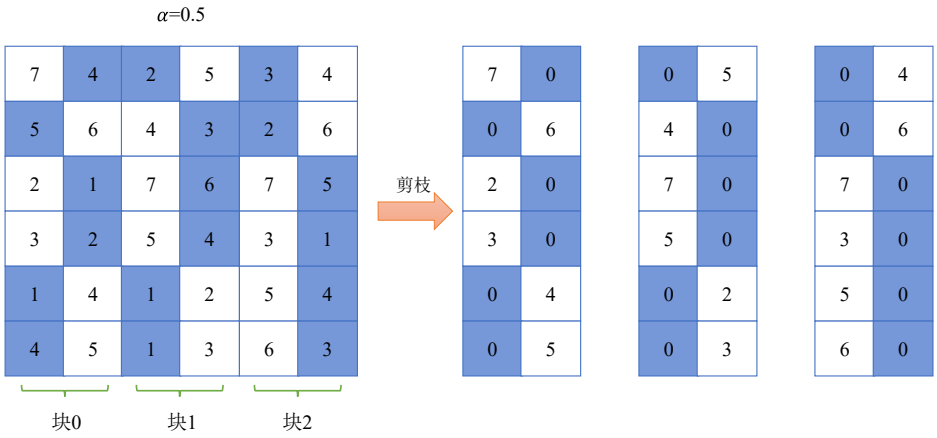

图 3-3　块均衡结构化剪枝算法

3.2.1　模型剪枝

模型剪枝能有效地裁剪掉冗余或不重要的权值，进而减少乘法操作次数。为了使模型减少资源占用，本设计使用文献[73]提出的块均衡结构化剪枝算法进行推理前的轻量化处理。

常规的非结构化剪枝算法在模型训练时，根据设定的重要性度量选择未超过阈值的权值参数置为 0，后续则通过忽略 0 数值参与的乘加运算来使计算效率提高。但是考虑到卷积算子会应用加速器的大规模并行计算资源进行计算，这个处理过程会分为访存和计算的流水线，因此当权值矩阵的稀疏率不规则时，会导致卷积算子的负载不均衡，进而影响流水线的填充。块均衡的结构化剪枝方法对所有的权值矩阵设定相同的稀疏率，其原理如图3-3所示。以稀疏率 $\alpha = 0.5$ 为例，在分块后的权值矩阵中，相邻两列数据较小者会被置为 0，最终在每个数据块均达到相同的稀疏率，保证每个时钟周期内权值矩阵的稀疏率是一致的。通过这种均衡的剪枝方式，后续加速器中每个累加树处理的权值个数相同，并且拥有更好的局部性。

剪枝完成后，此时的权值矩阵中存在大量的 0 元素，较为稀疏。为了提高内存单元的利用率，需要对稀疏化权值矩阵进行编码压缩。由于此前采用列维度的剪枝规则，所以稀疏化矩阵中各行 0 元素的数量相同，于是可以使用基于列索引的位置编码。对于每一行中的非零元素，只需要记录其对应的列索引，即可确定相应的权值。若此前稀疏矩阵的列数为 n，则压缩后的稠密矩阵列数为 $log_2 n$, 能

大幅提高内存利用率。

3.2.2 定点数量化

在加速器的结构中，网络模型的权值，偏置和预处理后的图像数据都是以32 位浮点数的形式存储在片外存储器中，为了减少模型推理时的功耗并提升推理速度，本设计采用了定点数量化的方式，将读取到的高精度浮点数转为低精度定点数。

对于 32 位浮点数而言，其小数点可根据其指数部分不同大小的移码来决定位置，表现为小数点可以" 浮动"。在这种方法中，32 位浮点数的表示如式3-1所示：

$$V_{float} = (-1)^{Sign} \times 2^{Exp-127} \times (1 + \sum_{i=1}^{23} b_{23-i} \cdot 2^{-i}), \tag{3-1}$$

其中，$Sign$ 表示符号位，可取 0, 1，Exp 为指数部分，减去固定偏移量可得到移码，决定浮点数小数点的位置，最后为尾数部分，决定了浮点数表示的值与精度。

在定点数的数据格式中，小数点的位置固定不变，数值表示的范围和精度都是有限的。采用低精度定点数可以降低数据位宽，减少片上存储空间占用和传输时间消耗。定点数的表示如式3-2所示：

$$V_{fixed} = \sum_{i=0}^{Length-1} b_i \times 2^{-Exp} \times 2^i, \tag{3-2}$$

其中 $Length$ 是定点数位宽，b_i 是第 i 位上的二进制数字，Exp 是阶码, 用于指示定点数的小数点位置。

定点数和浮点数之间的相互转换如式3-3和式3-4所示：

$$V_{fixed} = (int)(V_{float} \times 2^{Exp}), \tag{3-3}$$

$$V_{float} = (float)(V_{fixed} \times 2^{-Exp}), \tag{3-4}$$

在定点数量化时，量化的数据位宽越短，所需的内存资源就越少，但是截断误差会增大，导致精度损失更大。因此，定点数量化的阶码值 Exp 是决定量化性能的关键。本设计采用一种动态多精度定点量化策略，搜索最优的 Exp，以实

现性能和资源的平衡。

在定点数的表示中，定点数据的表示精度由阶码 Exp 决定。传统的静态量化策略会选定一个 Exp 值，并将所有数据都按照固定的比例系数进行量化放缩。然而，在卷积算子的计算过程中，不同的输入输出特征图，不同层之间的权值分布范围都是不一样的，采取统一固定的量化系数会导致较高的截断误差。因此，根据参与卷积算子计算的各类数据范围，采用动态多精度定点量化能有效降低转换时的精度损失进而提升推理效果，算法3.1展示了搜索最优阶码的过程。

算法 3.1 搜索最优阶码值

输入: 浮点数据 V_{float}，定点数位宽 bl
输出: 最优阶码值 Exp_{best}

```
确定原始数据来源 //权值矩阵，特征图，中间结果
for 阶码值 Exp_i ∈ bl do
    for 浮点数值 V_float(j) ∈ V_float do
        将 V_float(j) 转换为定点数值 V_fixed(bl, Exp_i, j)
        计算本次截断误差并累加
    end for
    更新最小截断误差绝对值之和并记录对应的阶码值 Exp_i
end for
得到最优阶码值 Exp_best
```

3.3　算子重构

对神经网络模型进行算子重构可以在保持模型性能的前提下有效降低复杂度，本设计针对 YOLOv4-tiny 模型中的核心部分——卷积算子进行更适合硬件部署的重构。

3.3.1　循环优化

据统计，卷积神经网络中 90% 的计算开销都来源于卷积计算[74]。卷积层接收 N 个特征图作为输入, 大小为 $K*K$ 的卷积核在输入特征图上滑动进行卷积，并得到输出特征图中的一个像素。移位窗口的步长为 S，通常小于 K，最终输出的 M 个输出特征图将作为下一个卷积层的输入特征图。常规卷积计算的伪代码如算法3.2所示。

可以看到伪代码中一共有 6 层循环，其中 i 代表卷积窗口内行，j 代表卷积

算法 3.2 常规卷积运算伪代码

输入: 输入特征图 *Input_fm*，权值矩阵 *Weight*
输出: 输出特征图 *Output_fm*

```
for i = 1, 2, ..., K do
    for j = 1, 2, ..., K do
        for to = 1, 2, ..., M do
            for ti = 1, 2, ..., N do
                for row = 1, 2, ..., R do
                    for col = 1, 2, ..., C do
                        遍历输入特征图 Input_fm 和权值矩阵 Weight
                        通过乘加运算得到输出特征图 Output_fm
                    end for
                end for
            end for
        end for
    end for
end for
```

窗口内列，*to* 代表卷积核编号，*ti* 代表通道，*row* 代表行循环，*col* 代表列循环，*K* 代表卷积核尺寸，*M* 代表输出特征图数量，*N* 代表输入特征图数量，*R* 代表输出特征图的行数，*C* 代表输出特征图的列数。

常规的卷积运算复杂度高，计算时需要存储的中间结果多，不利于在资源有限的低功耗平台上直接部署。因此，本设计采用循环展开，循环平铺和循环交换的方法对卷积算子中的循环进行优化。

循环展开是一种编译器层面的优化技术，通过减少循环迭代的次数来减少循环控制开销。通过对各个维度的循环进行展开，再将展开后的计算单元组成并行化流水线，能优化卷积算子的计算时延。设某层循环的原始循环次数为 n，展开因子为 k，则优化后循环次数为：

$$n' = \lfloor \frac{n}{k} \rfloor + +(N mod k), \tag{3-5}$$

由于每次每次迭代处理 k 个元素，于是循环条件判断次数减少 k 倍，展开后的连续运算可以消除指令间依赖，进而能提高流水线利用率。

循环平铺通过将原始循环分解为小块来优化数据局部性。对于卷积计算中嵌套的多维循环，平铺操作将大矩阵划分为尺寸为 $T_w \times T_h$ 的子块，使得每个小块的数据能够有效地放入片上缓存中，提供更高的内存访问效率, 其原理如图3-4所示。

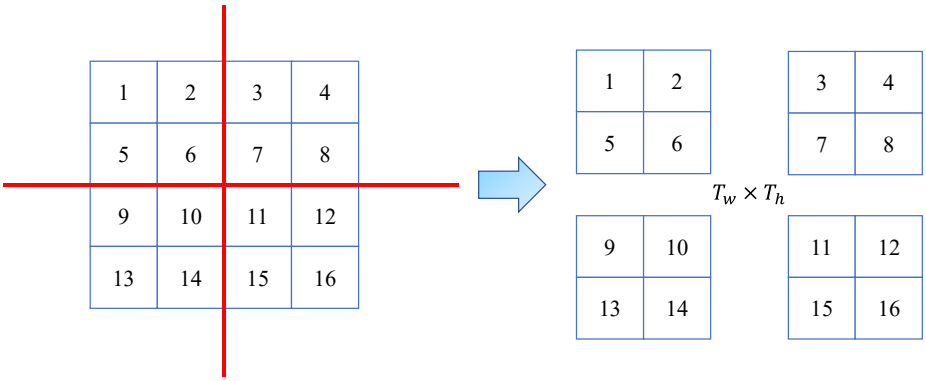

图 3-4　循环平铺原理

循环交换通过调整循环嵌套顺序以优化数据访问模式，加强对资源的复用程度，进而提高缓存命中率。在卷积算子中，由于卷积核的尺寸相对于特征图和权值矩阵尺寸小很多，可以一次性读取到片上，因此一般将卷积核的行和列两个维度的遍历放在最内层的循环中，将输出特征图的数量，行，列和输入特征图数量放在外层循环。

经过循环展开，循环平铺和循环交换的优化操作后，卷积运算可以在硬件平台上得到更高效的执行，此时的伪代码如算法3.3所示：

3.3.2　层融合

FPGA 平台的内存资源非常有限，往往是计算任务中的瓶颈。为了减少模型推理时的内存占用，本设计采用层融合技术进行推理时内存优化。层融合通过将多个连续的计算步骤合并为单一操作，以减少中间结果存储和内存访问次数，从而提升效率。

分析 YOLOv4-tiny 的模型结构，其骨干网络中的核心操作包括卷积计算，批归一化和 ReLU 函数激活输出。融合前，卷积层的数学表达如下：

$$C = wx + b, \tag{3-6}$$

BN 层对卷积输出归一化，使用训练阶段统计的均值 μ、方差 σ^2，以及可学

算法 3.3 循环优化卷积运算伪代码

输入: 输入特征图 *Input_fm*，权值矩阵 *Weight*
输出: 输出特征图 *Output_fm*

```
循环平铺将大矩阵划分为小矩阵
得到平铺后的输出特征图数量 tm，输入特征图数量 tn，输出特征图的行数 tr，输出特征图的列数 tc
for row = 1, 2, ..., tr do
    for col = 1, 2, ..., tc do
        #pragma HLSPIPELINE
        //pragma 指令可以实现编译器级并行和流水
        for to = 1, 2, ..., M do
            for ti = 1, 2, ..., N do
                //循环交换，将卷积核的遍历放在最内层
                for i = 1, 2, ..., K do
                    for j = 1, 2, ..., K do
                        遍历输入特征图 Input_fm 和权值矩阵 Weight
                        通过乘加运算得到输出特征图 Output_fm
                    end for
                end for
            end for
        end for
    end for
end for
```

习参数 γ（缩放）和 β（平移）：

$$B = \gamma \cdot \frac{C - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{3-7}$$

层融合将卷积与 BN 的线性部分合并为一个等效的卷积运算，总输出可表示为：

$$Y = \gamma \cdot \frac{(wx + b) - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{3-8}$$

整理可得到新的权值 w' 和偏置 b'：

$$Y = w'x + b' \tag{3-9}$$

$$w' = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} \cdot w \quad b' = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}}(b - \mu) + \beta, \tag{3-10}$$

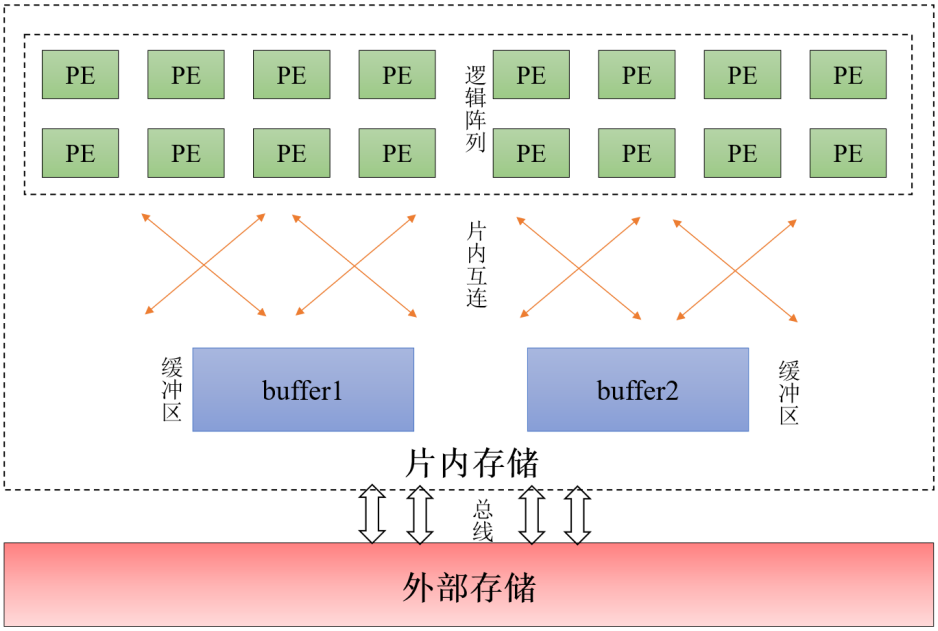

图 3-5　双缓冲结构原理

最后，再将合并后的线性运算结果通过 LeakyReLU 函数激活：

$$\text{LeakyReLU}(Y)=\begin{cases} Y & \text{if } Y>0, \\ \alpha \cdot Y & \text{if } Y \leqslant 0. \end{cases} \tag{3-11}$$

其中 α 为负区间的固定斜率。

3.4　数据流优化

当模型部署在 FPGA 平台上时，考虑到数据在硬件单元间的流动情况，本设计针对数据的传输和存储进行硬件设计的优化。

3.4.1　双缓冲机制

在算子设计时，如果在片上存储所有计算时需要的中间数据，那么能达到很高的性能。但是考虑到边缘硬件平台的资源有限和带宽限制，直接将全部计算数据缓存到片上是无法实现的。因此在设计时，采用双缓冲机制，使数据的加载和计算操作同时进行，形成两级流水线，提高计算周期的覆盖率，进而达到更高的吞吐量。双缓冲机制的结构如图3-5所示：

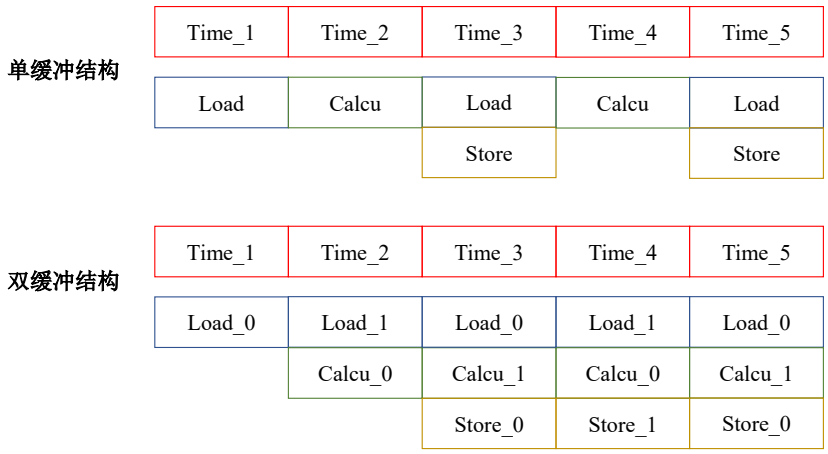

图 3-6 缓冲对比

采用双缓冲机制前的流水线和采用后的流水线如图3-6所示：

分析流水线，可以发现采用双缓冲机制前，数据流是完全串行的，每一次计算操作必须等到数据加载完毕才可以进行，等到计算完毕后，再将计算结果写回，总时延较高。设数据的读取时延，计算时延和写入时延分别为 $T_{load}, T_{compute}, T_{store}$，需要计算的数据块数量为 K，那么可以将单缓冲时延 T_{single} 表示为：

$$T_{single} = K * (T_{load} + T_{compute} + T_{store}), \tag{3-12}$$

双缓冲机制通过并行加载与计算，重叠数据搬运与计算时间，分为流水线填充阶段和流水线稳定阶段。当流水线完成填充后，后续块的计算与写入交替进行，读取与计算重叠。双缓冲时延 T_{double} 可表示为：

$$T_{double} = T_{load} + K * max(T_{compute}, T_{load} + T_{store}) + T_{store}, \tag{3-13}$$

当卷积算子中计算时延占主导时，即 $T_{compute} \geqslant T_{load} + T_{store}$, 双缓冲时延可表示为：

$$T_{double} = T_{load} + K * T_{compute} + T_{store}, \tag{3-14}$$

此时的加速比为：

$$S = \frac{K(T_{\text{load}} + T_{\text{compute}} + T_{\text{store}})}{T_{\text{load}} + K(T_{\text{load}} + T_{\text{store}}) + T_{\text{compute}}} \approx \frac{T_{\text{load}} + T_{\text{compute}} + T_{\text{store}}}{T_{\text{load}} + T_{\text{store}}}, \tag{3-15}$$

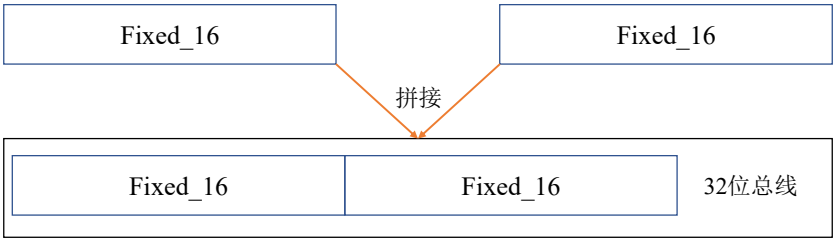

图 3-7　带宽优化

当 $K \to \infty$ 时，加速比趋近于 $(1+\frac{T_{\text{load}}+T_{\text{store}}}{T_{\text{compute}}})$。

对于卷积算子这样的计算密集型任务，当采取低位宽数据运算减少计算时延后，通过双缓冲机制充盈流水线，可以达到更高的硬件计算效率。

3.4.2　总线带宽优化

在硬件平台上，总线是系统中连接各个组件的通信线路，不同组件之间的数据传输都需要经过总线。以 32 位位宽总线为例，如果需要传输的数据是 32 位，那么总线在一个数据周期内可以搬运 1 个数据。考虑到本设计已经对网络模型的参数进行了量化，因此在实际传输中存在很多的低位宽数据。当数据块大小与总线带宽不匹配时，总线可能会出现空闲状态，导致数据传输效率低下。为了解决这一问题，本设计针对量化后的低位宽数据，实现一个总线周期内运输多个数据，以达到更高的总线带宽利用率。

图3-7展示了 32 位总线一次运输两个 16 位定点数的处理流程。

3.5　实验与分析

为了验证本文提出的加速器设计方法的性能，本节以目标检测任务为载体，通过实验与分析，对基于 FPGA 平台的 YOLOv4-tiny 加速器设计进行详细的性能评估。

3.5.1 实验设置

软件环境

- 开发语言：Python，C++。Python 用于训练原始的 YOLOv4-tiny 模型，C++ 用于编写 YOLOv4-tiny 模型映射在 FPGA 平台上的各类算子。
- 卷积神经网络模型：YOLOv4-tiny，网络框架为 CSPdarknet53-tiny。
- 开发工具：Xilinx HLS，Xilinx Vivado。前者是 Xilinx 官方提供的高层次综合仿真工具，用于重构神经网络算子，后者可以集成各算子的 IP 核形成顶层 bdf 文件，再编译得到 FPGA 可识别的比特流文件。

硬件环境

- FPGA：Xilinx PYNQ-z2。FPGA 芯片型号为 XC7Z020-1CLG400C，搭载双核 ARM Cortex-A9。
- CPU：Intel，ARM Cortex-A9，频率 650MHz，512M 内存。
- GPU：NVIDIA GTX 2080Ti。

数据集

Pascal VOC 数据集来源于世界级的计算机视觉竞赛 Pascal(The Pattern Analysis, Statical Modeling and Computational Learning) VOC。该比赛的数据集按年份发布，在所有数据集发布版本中，以 2007 版本和 2012 版本最受欢迎，是计算机视觉领域最经典的数据集之一，在目标检测、图像分类和语义分割任务中被广泛应用。VOC2007 的数据规模包含 9963 张图片（训练集：2501 张；验证集：2510 张；测试集：4952 张），涵盖了 20 个常见类别，标注对象总数为 24640 个。

COCO 数据集是计算机视觉领域最具影响力的大规模、多任务数据集之一，由微软团队于 2014 年首次发布并持续更新，主要应用于目标检测、实例分割、图像描述生成等任务。COCO2014 的数据规模包含约 33 万张图片 (训练集：118k，验证集：5k，测试集：41k)，涵盖了 80 个常见物体类别，标注目标数超过 250 万个目标实例。

3.5.2　实验过程

本设计的实验过程主要包括三个步骤，分别是原始模型的训练，将网络模型用硬件实现和基于不同平台的模型推理。

1. 模型训练：在 GPU 服务器上分别使用 VOC 2007 数据集和 COCO 数据集训练 YOLOv4-tiny 模型，并将得到的模型文件保存下来。
2. 硬件实现：使用 Xilinx HLS 和 Xilinx Vivado 针对加速器进行硬件设计，包括算子 IP 设计和加速器整体 Block 设计。
3. 模型推理：将 YOLOv4-tiny 模型分别在 CPU，GPU 和 FPGA 环境下进行推理实验，记录推理时的功耗，时间和准确率等指标。

下面将详细讲述本设计的硬件实现过程。

硬件实现过程

高层次综合工具 (High-level Synthesis tool, HLS) 是 Xilinx 在 2012 年发布硬件开发工具。它提供了一种抽象层，通过仿真，优化，综合，导出等过程将高级语言代码转化为硬件级别的描述，自动生成硬件电路，使开发者能够使用熟悉的高级语言进行硬件设计，并将设计的模块作为硬件电路设计的 IP 核导出。

HLS 工具大大简化了 FPGA 的设计流程，其抽象层级高，无需手动编写寄存器传输级（RTL）代码，通常可以减少 10 倍以上的代码量。同时，HLS 还提供了自动化优化能力，在设计时可通过并行化与流水线，资源与时序权衡，自动接口生成等操作优化硬件实现。最后，HLS 设计的硬件电路描述具有良好的可移植性，其内含的 C 语言仿真快速验证功能速度比传统的 RTL 仿真快百倍以上，支持灵活的功能修改。

本设计使用 Xlinx HLS 工具和 Vivado 软件进行硬件实现的具体过程如下：

1. 算子分析：对 YOLOv4-tiny 模型的 CSPDarkNet53-tiny 框架进行详细分析，经过层融合后，需硬件实现的算子一共有 7 种，分别是 3 种不同配置的卷积算子 Conv2D，最大池化算子 Maxpool2d，上采样算子 Upsample，激活函数 LeakyReLU 和拼接算子 Concat。
2. 算子实现：使用 HLS 工具支持的高级编程语言 C++ 编写上述分析的算子。

3. 算子仿真：为了确认实现的算子功能正常，需要对算子进行功能仿真，通过编写 HLS 工具支持的 testbench，将算子测试结果与原始模型的结果进行比对，如果误差在可接受范围内，即可认为算子实现功能正确。
4. 综合设计：对算子仿真完毕后，利用 HLS 工具提供的 C 语言综合功能，对加速器的整体设计进行综合分析，得到加速器整体硬件设计结果。
5. 导出 IP 核：当 HLS 完成对加速器的综合分析后，可以将加速器整体作为 IP 核导出，用于之后的顶层 SoC 电路设计，至此完成了 HLS 工具的所有实现流程。
6. SoC 设计：得到加速器的 IP 核后，使用 Vivado 创建工程，添加加速器 IP 核与 PYNQ-z2 的处理器核心，对整体的 Block 设计进行布线，配置各模块之间的接口。
7. 综合实现：在 Vivado 中经过仿真，综合，实现等步骤后，可以得到最终的硬件设计文件以及硬件设计的各类指标。该硬件设计文件作为 FPGA 逻辑资源可以直接识别的比特流文件，需要导出供后续 FPGA 推理使用。至此完成了 Vivado 的所有实现流程。
8. 硬件加速:将导出的比特流文件下载到板卡上,板卡加载设计好的 YOLOv4-tiny 模型 (作为 IP 核)，执行利用 FPGA 并行逻辑资源的硬件加速。

整个加速器硬件实现的顶层 Block 连接图如图3-8所示。

3.5.3 实验结果

本小节将展示实验过程中的详细实验结果，并根据相关结果进行分析，进而验证本加速器设计的有效性。

参数分析

首先对训练得到的 YOLOv4-tiny 模型进行参数分析，如表3-1所示，模型总参数量为 6056606 个，模型大小 (占用的存储空间) 为 73.52MB，需要的点积运算 (乘-加操作) 为 6.93GMAdd，浮点运算次数为 3.47GFlops，运行内存包括网络运行时从内存中读取的内存大小和写入到内存中的内存大小两部分，其总和为 176.34MB。由于 PYNQ-z2 的板上内存仅为 512MB，再考虑到开发板需要永久

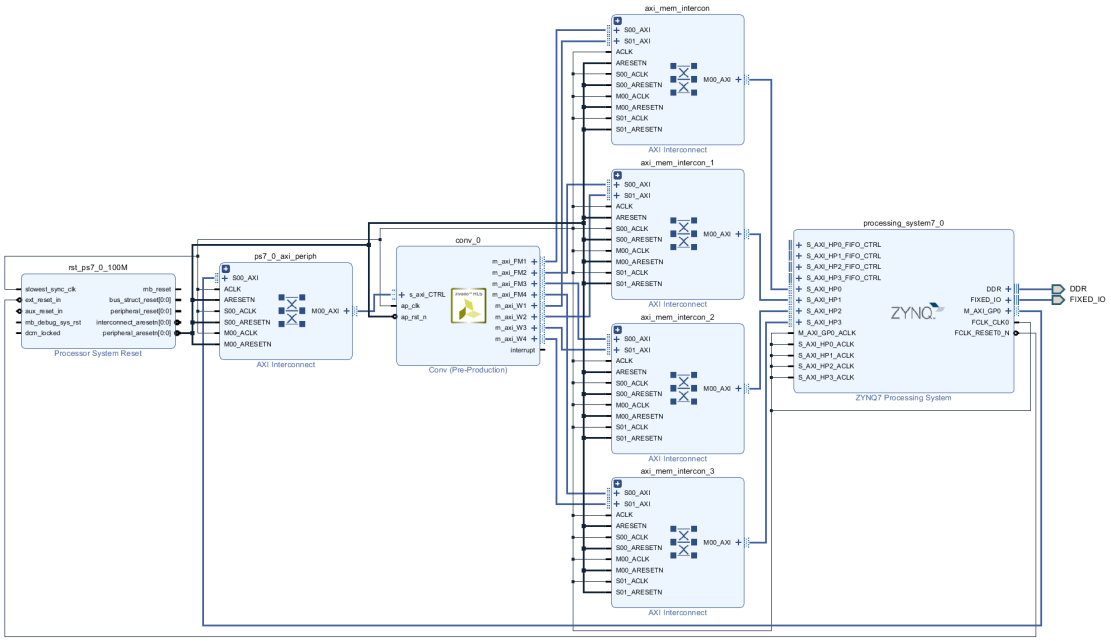

图 3-8　加速器顶层设计图

保留一部分内存空间运行 ARM 核心处理器以及其他系统进程，因此若直接将原始网络模型加载到 FPGA 开发板上，会面临严重的资源紧缺问题。

表 3-1　原始模型参数分析

参数个数	模型大小	点积运算	浮点运算	运行内存
6056606	73.52MB	6.93GMAdd	3.47GFlops	176.34MB

剪枝分析

针对本设计采用的剪枝方法，图3-9展示了在不同的剪枝稀疏率设定下，YOLOv4-tiny 模型在 VOC 2007 数据集和 COCO 数据集表现的平均精度均值 (mean Average Precision, mAP) 的损失程度。对于两类数据集而言，若稀疏率 $\alpha \leqslant 0.5$，此时 mAP 的精度损失都很小，说明剪枝方法具有不错的效果，当稀疏率超过 0.5 时，模型性能会出现明显下降，说明此时稀疏率过大剪去参数过多，影响了模型的推理能力。最后，将精度与模型大小进行综合考虑，本设计采用 $\alpha = 0.5$ 对模型权值进行剪枝以适用 YOLOv4-tiny 模型在 FPGA 平台的部署。

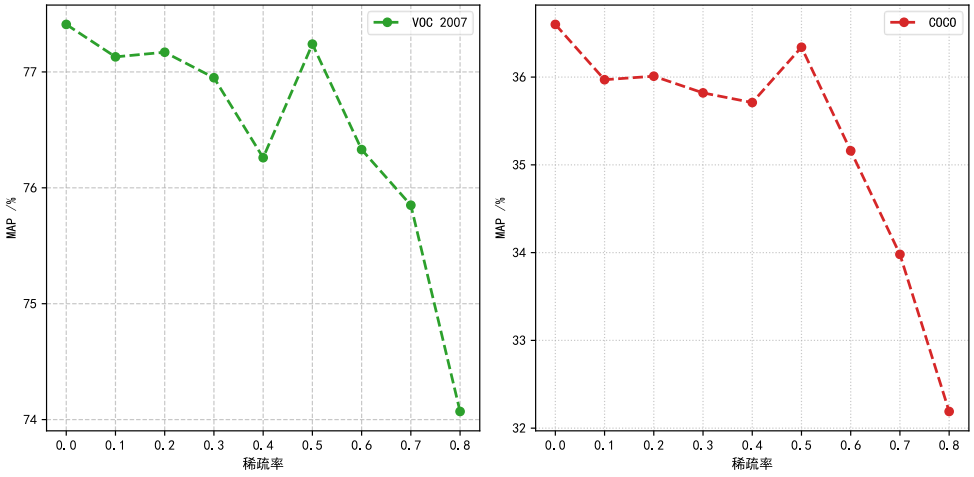

图 3-9 不同剪枝率下的精度损失

量化分析

针对本设计的定点数量化方法,表3-2给出了量化前后的资源消耗变化。LUT(Look-Up Table) 查找表是 FPGA 中实现组合逻辑的基本单元，在量化前，LUT 占用率为 185%，通过 16 位定点数量化可降低至 68.2%. Register(寄存器) 是 FPGA 中用于存储数据的基本单元，量化操作可实现 2.76 倍的资源占用优化。BRAM(块随机访问内存) 是 FPGA 内部集成的大容量存储资源，16 位定点数量化可以将资源占用从 100% 减少至 66.4%。DSP(数字信号处理模块) 是 FPGA 中专门用于高效执行数字信号处理运算的硬件资源，可以快速地进行乘法、加法、累加等运算，从表中可以看到，量化前的模型无法利用板上拥有的 DSP 资源完成推理，其占用率远远超出总资源，量化后可以将 DSP 占用率降低至 98.6%，既保证模型能被正常加载推理，也保证了计算资源的高效利用。

表 3-2 16 位定点数量化分析

	LUT	Register	BRAM	DSP
总资源	53200	106400	140	220
量化前 (float32)	98508(185%)	94330(88.6%)	140(100%)	815(370%)
量化后 (AP16)	36325(68.2%)	34064(32.0%)	93(66.4%)	217(98.6%)

推理分析

在 CPU,GPU,FPGA 三种不同硬件平台下的实验结果如表3-3所示，表中给出了不同硬件平台的具体型号，工作频率，推理使用的数据精度等基本信息。推理速度定义为每张图片推理需要的时间，CPU 和 GPU 的平均功耗通过查阅其性能参数得到，FPGA 的平均功耗通过加速器设计的实现阶段报告得出。能效定义为在该平台上每焦耳能量能推理的图片数量。

在实验结果中，GPU 拥有最高的推理速度，但其平均功耗较高，相比通用处理器 CPU，其在计算任务上有更好的能效表现。FPGA 平台尽管工作频率较低，但本设计拥有不俗的单张推理速度，且功耗仅 2.33W，能效达到了 10.05 张/J，约为 GPU 平台的 6.6 倍。

表 3-3　不同硬件平台下的实验结果

硬件平台	CPU	GPU	FPGA
硬件型号	Intel(R)core(TM)i5-9400	NVIDIA RTX 2080Ti	Zynq XC7Z020
工作频率 (Hz)	2.9G	1.35G	100M
数据精度	float32	float32	AP16
推理速度 (s/张)	0.3721	0.0025	0.0427
平均功耗 (W)	65	260	2.33
能效 (张/J)	0.041	1.53	10.05

与其他基于 FPGA 平台的目标检测模型效果对比如表3-4所示，表中未标注的 mAP 指标是基于 VOC 数据集得到的结果。整体而言，所有工作选择的 FPGA 平台在性能上不存在量级差异，实际表现主要由所选的神经网络模型在 FPGA 平台的适配程度和加速器设计方法决定。文献[75]以 Tiny-YOLO 为基础，通过低位宽量化将所有特征图权值量化至 3 bit, 结合并行流水技术提出了 Tincy-YOLO 并部署在 Zynq Ultrascale+(XCZU3EG) 上，实现了较低的资源占用，但由于量化位宽过低，导致推理精度不佳。文献[76]将二元 CNN(用于特征提取) 与并行 SVM（用于精细检测）相结合，提出了轻量权值 YOLOv2，在 FPGA 平台上实现了较高的能效表现，但缺乏对模型卷积算子的优化。文献[77]采用流式架构设计，以深度流水线方式在 FPGA 平台实现了完整的 YOLOv3-tiny 模型，拥有不俗的性能表现，但是缺少轻量化处理，使得 DSP 消耗过高。文献[78] 将 YOLOv4-tiny 模型针对 Kria KV260 平台进行了结构化剪枝，减少模型资源占用的同时达到了不俗的准确率，但是缺乏对网络模型算子针对平台的进一步优化，导致能效表现不

佳。通过实验分析，本设计在 COCO 数据集和 VOC 数据集上的准确率表现均优于之前的工作的同时，能以较少的 DSP 消耗和较低的功耗，达到中等表现的吞吐量，并实现最优的能效。

表 3-4 与其他基于 FPGA 的相关工作对比

	Preußer 等人[75]	Nakahara 等人[76]	Montgomerie 等人[77]	Heller 等人[78]	本设计
网络模型	Tincy-YOLO	Lightweight-YOLOv2	YOLOv3-tiny	YOLOv4-tiny	YOLOv4-tiny
FPGA 平台	Zynq Ultrascale+	Zynq Ultrascale+	VCU110	Kria KV 260	PYNQ-z2
工作频率 (Hz)	N/A	300M	220M	N/A	100M
准确率 (mAP)(%)	48.5	67.6	33.9^{COCO}	73.8	36.34^{COCO} 77.24
DSP 消耗	N/A	377	1780	N/A	217
功耗 (W)	6	4.5	15.4	8	2.33
吞吐量 (FPS)	16	40.81	69	15	23.42
能效 (FPS/W)	2.67	9.06	4.48	1.88	10.05

3.6 本章小结

本章通过软件模型层面和硬件部署层面的联合优化，设计了一种基于 PYNQ-z2 FPGA 开发板的 YOLOv4-tiny 模型加速器。本设计使用了剪枝与量化等轻量化操作，降低了模型部署时的资源消耗，使模型能在资源有限的 FPGA 平台上高效运行。此外，还针对 YOLO 模型核心的卷积算子使用了融合，重构等优化手段，进一步提高了模型在 FPGA 平台的推理能力。最后还结合 FPGA 平台的特性，优化了推理时的数据流，实现了更高的硬件效率。通过实验对比，验证了本设计能以最优的能效实现精度最高的边缘智能推理。

第四章　基于 NPU 的高能效脉冲神经网络加速器设计

脉冲神经网络（SNN）通过仿生脉冲时序编码机制，在动态信息处理与能效表现上显著超越传统人工神经网络，具备在神经形态硬件上高能效推理的巨大部署潜力。

NPU 是一种专为神经网络计算设计的处理器，通过架构级定制化突破传统通用处理器（CPU/GPU）在能效比与并行计算上的瓶颈。其核心思想是将神经网络的计算模式（如矩阵乘法、非线性激活、稀疏连接等）映射为硬件原生支持的操作。通过架构级创新与软硬件深度协同，NPU 正在成为边缘 AI 部署的核心引擎。

目前，主流的神经网络训练——推理流程还是在通用图形处理器 (GPU) 平台上进行。然而，全球高端 GPU 市场由英伟达和 AMD 主导，他们占据了 80% 以上的市场，由于现代 AI 应用高度依赖 GPU 的大规模并行计算能力，所以我国 AI 行业面临着关键设备断供的风险。例如，BIS（美国商务部工业安全局）通过《出口管理条例》（EAR）限制先进制程（>16nm）、算力阈值（≥4800 TOPS）或带宽（≥600 GB/s）的 GPU 对华出口。这样的断供风险隐藏着诸多危机，可能会导致我国 AI 研发链受到冲击，包括硬件计算平台的缺乏，大量代码需要迁移重构等。因此，为了解决断供风险，在国产化硬件计算平台上进行 AI 应用的落地刻不容缓。

然而，将其他平台训练好的模型部署在 NPU 平台时，会面临严峻的适配性问题。由于硬件架构和编程开发库的不同，从其他平台得到的网络模型无法直接部署在 NPU 平台上，需要进行模型迁移工作。在迁移过程中，还应考虑到各类算子在 NPU 平台的支持度，并进行针对性优化。迁移完成后，也需要进一步在推理过程中考虑结合硬件特性的优化，以充分发挥计算平台的性能。

为了在边缘端部署高效的 SNN 模型，并在一定程度上解决关键设备的断供风险，本设计选择华为 Atlas 200I DK A2 开发者套件作为硬件计算平台，开创性地将 SNN 模型部署到开发板上，并利用开发板搭载的国产 NPU，完成对 SNN

的推理加速。

4.1　脉冲神经网络模型搭建

4.1.1　IF 脉冲神经元

Integrated & Fire (IF) 脉冲神经元是一种针对阈下电位的变化规律进行描述的简单模型，它将输入脉冲信号整合到膜电位中，如果达到定义的阈值，则产生输出脉冲，膜电位再恢复到静息状态。IF 模型的神经学原理可以用公式4-1来描述:

$$C_m \frac{dV}{dt} = I(t), \tag{4-1}$$

式中 C_m 为膜电容，V 为膜电位，$I(t)$ 为当前电流。IF 脉冲神经元的动作包括整合，激发，重置三个步骤，计算功耗非常低的同时仍能捕捉到一些重要的神经元特性，可以很好地逼近真实神经元的特性和行为，也能适应计算资源有限的边缘硬件。在实际研究中，IF 模型往往是各类新型脉冲神经元模型的基础，也是 ANN-to-SNN 转换训练方法首选的脉冲神经元，拥有广泛的应用场景。

在搭建 SNN 模型时，IF 脉冲神经元可以按如下方式定义，如果第 l 层的 IF 神经元接受了来自上一层 IF 神经元的输入 $\boldsymbol{x}^{l-1}(t)$，那么 IF 神经元的时间电位可以定义为：

$$\boldsymbol{u}^l(t) = \boldsymbol{v}^l(t-1) + \boldsymbol{W}^l \boldsymbol{x}^{l-1}(t), \tag{4-2}$$

其中 $\boldsymbol{u}^l(t)$ 和 $\boldsymbol{v}^l(t)$ 表示在时间步长 t 触发脉冲事件之前和之后的膜电位，$\boldsymbol{W}^l$ 代表第 l 层的权值。当 $\boldsymbol{u}^l(t)$ 的任何元素 $\boldsymbol{u}_i^l(t)$ 超过放电阈值 θ^l，神经元将发送脉冲并更新膜电位 $\boldsymbol{v}_i^l(t)$。

当脉冲神经元完成整合与激发步骤后，将采取减法重置机制[79]恢复膜电位，其规则如下式所示：

$$\begin{aligned} \boldsymbol{o}^l(t) &= H\left(\boldsymbol{m}^l(t) - \boldsymbol{\Theta}^l\right), \\ \boldsymbol{v}^l(t) &= \boldsymbol{m}^l(t) - \boldsymbol{o}^l(t)\theta^l \end{aligned}. \tag{4-3}$$

其中，$\boldsymbol{o}^l(t)$ 是时间步长 t 中第 l 层中所有脉冲神经元的输出脉冲，若存在脉冲则其值为 1，否则为 0,$H(\cdot)$ 是 Heaviside 阶跃函数,$\boldsymbol{\Theta}^l$ 是放电阈值 θ^l 的向量形式。

4.1.2 ANN-to-SNN 转换法

ANN-to-SNN 转换法从拥有成熟训练方法的 ANN 中学习，快速得到性能良好的 SNN，其核心思想是将 ANN 中人工神经元的激活值映射到 SNN 中脉冲神经元的发射率 (或平均突触后电位)。

采用 ANN-to-SNN 转换法有几个好处。第一，在大规模网络中模拟精确的脉冲动作可能会在计算上很昂贵，特别是在资源严格受限的边缘硬件设备中。因此，ANN-to-SNN 转换法可以让 SNN 避免高代价的大规模训练，同时与转换前的 ANN 相比，得到的 SNN 精度损失很小。第二，神经网络的训练过程可以在 ANN 上进行，从而使用算力资源更充足的 GPU 等设备，无需在线学习，降低边缘部署的硬件要求。最后，对 ANN 进行训练的方法已经经过了长久的发展，技术方面更成熟，这能为转换得到的 SNN 提供更高的性能上限。

根据公式4-2和4-3，可以将 ANN-to-SNN 转换得到的脉冲神经元的基本功能表示为：

$$\boldsymbol{v}^l(t) - \boldsymbol{v}^l(t-1) = \boldsymbol{W}^l \boldsymbol{x}^{l-1}(t) - \boldsymbol{o}^l(t)\theta^l, \tag{4-4}$$

同时记录下从 0 到 T 期间的平均突触后电位：

$$\boldsymbol{\sigma}^{l-1}(T) = \frac{\sum_{i=1}^{T} \boldsymbol{x}^{l-1}(i)}{T}, \tag{4-5}$$

再根据4-4和4-5, 将方程对时间步长进行求和，然后除以总时间 T，可以得到：

$$\boldsymbol{\sigma}^l(T) = \boldsymbol{W}^l \boldsymbol{\sigma}^{l-1}(T) - \frac{\boldsymbol{v}^l(T) - \boldsymbol{v}^l(0)}{T}, \tag{4-6}$$

式4-6描述了相邻层脉冲神经元的平均突触后电位关系，可以发现，若将第 0 层膜电位置为 0，并将总时间 $T \to \infty$, 则可以忽略掉尾项 $\frac{\boldsymbol{v}^l(T)-\boldsymbol{v}^l(0)}{T}$，此时脉冲神经元几乎等价于传统人工神经元，即：

$$\boldsymbol{x}^l = h\left(\boldsymbol{W}^l \boldsymbol{x}^{l-1}\right), \tag{4-7}$$

其中 $\boldsymbol{x}^l$ 代表人工神经网络中第 l 层的人工神经元状态。

然而，过高的总时间 T 会带来较长的计算延迟，并不适合 SNN 模型在边缘

端的部署与推理。因此，在 SNN 训练时应选择达到低延迟的同时能保证高性能的 ANN-to-SNN 转换方法。

本设计采用文献[80]所提出的量化向下取整移位 (Quantization Clip Floor Shift, QCFS) 激活函数来完成对 SNN 模型的训练。QCFS 的原理可由公式4-8表示：

$$\boldsymbol{x}^l = \widehat{h}\left(\boldsymbol{z}^l\right) = \lambda^l \operatorname{clip}\left(\frac{1}{L}\left\lfloor\frac{\boldsymbol{z}^l L}{\lambda^l} + \varphi\right\rfloor, 0, 1\right), \tag{4-8}$$

其中，超参数 $\boldsymbol{L}$ 表示 ANN 的量化步长，可训练参数 λ^l 决定了 ANN 中 $\boldsymbol{x}^l$ 的最大值与映射到 SNN 中 $\boldsymbol{\sigma}^l(\boldsymbol{T})$ 的最大值，$\boldsymbol{z}^l = \boldsymbol{W}^l\boldsymbol{\sigma}^{l-1}(\boldsymbol{T}) = \boldsymbol{W}^l\boldsymbol{x}^{l-1}$ 。利用这个新的激活函数，可以证明 SNN 和 ANN 之间估计的转换误差为零。

为了能直接训练 SNN 的权值，本节使用直通式估计器[81]对向下取整函数进行求导，即 $\frac{\mathrm{d}\lfloor x\rfloor}{\mathrm{d}x} = 1$, 权值更新的过程如式4-9, 4-10所示：

$$\frac{\partial \widehat{h}_i\left(\boldsymbol{z}^l\right)}{\partial z_i^l} = \begin{cases} 1, & -\frac{\lambda^l}{2L} < z_i^l < \lambda^l - \frac{\lambda^l}{2L} \\ 0, & \text{otherwise} \end{cases} \tag{4-9}$$

$$\frac{\partial \widehat{h}_i\left(\boldsymbol{z}^l\right)}{\partial \lambda^l} = \begin{cases} \frac{\widehat{h}_i\left(\boldsymbol{z}^l\right) - z_i^l}{\lambda^l}, & -\frac{\lambda^l}{2L} \leqslant z_i^l < \lambda^l - \frac{\lambda^l}{2L} \\ 0, & z_i^l < -\frac{\lambda^l}{2L} \\ 1, & z_i^l \geqslant \lambda^l - \frac{\lambda^l}{2L} \end{cases} \tag{4-10}$$

对于 ANN-to-SNN 的转换过程，算法4.1给出了使用 QCFS 激活函数的训练过程伪代码。

通过基于 QCFS 激活函数的 ANN-to-SNN 转换法，本设计得到了训练好的 SNN 模型，后续将把该模型部署在国产化边缘计算平台上进行优化推理。

4.2　模型迁移

模型迁移是将原本设计用于 GPU 或其他硬件平台的深度学习模型训练代码，通过模型代码修改，算子重构适配等操作，对 NPU 平台的架构进行针对性适应，使得模型可以在 NPU 平台进行高性能计算。

在将模型从其他三方平台迁移到 NPU 平台时，由于硬件架构和编程开发库

算法 4.1 使用 QCFS 激活函数将 ANN 转化为 SNN[80]

输入： ANN 模型 $M_{\text{ANN}}(x; W)$; 始权值 W; 数据集 D; 量化步长 L; 初始动态阈值 λ; 学习率 ϵ

输出： SNN 模型 $M_{\text{SNN}}(x; \hat{W})$

for $l = 1$ **to** M_{ANN}.layers **do**
  if ReLU 激活函数 **then**
    用 QCFS(x; L, λ^l) 替代 ReLU(x)
  end if
  if 最大池化层 **then**
    用平均池化替代最大池化
  end if
end for
for $e = 1$ **to** epochs **do**
  for 数据集大小 D **do**
    样本小批次 (x^0, y) from D
    for $l = 1$ **to** M_{ANN}.layers **do**
      $x^l = \text{QCFS}(W^l x^{l-1}; L, \lambda^l)$
    end for
    Loss = CrossEntropy(x^l, y)
    for $l = 1$ **to** M_{ANN}.layers **do**
      $W^l \leftarrow W^l - \epsilon \frac{\partial \text{Loss}}{\partial W^l}$
      $\lambda^l \leftarrow \lambda^l - \epsilon \frac{\partial \text{Loss}}{\partial \lambda^l}$
    end for
  end for
end for
for $l = 1$ /textbfto M_{ANN}.layers **do**
  $M_{\text{SNN}}.\hat{W}^l \leftarrow M_{\text{ANN}}.W^l$
  $M_{\text{SNN}}.\theta^l \leftarrow M_{\text{ANN}}.\lambda^l$
  $M_{\text{SNN}}.v^l(0) \leftarrow M_{\text{SNN}}.\theta^l/2$
end for
return M_{SNN}

的不同，涉及到一系列从底层驱动到顶层接口的适配操作。以 GPU 平台为例，将神经网络模型迁移至 NPU 平台需要适配的原因可分为三方面：

- **硬件特性差异**：由于 NPU 和 GPU 的硬件特性和性能特点不同，从 GPU 训练得到的模型在 NPU 上推理需要进一步的性能调试和优化，以充分发挥 NPU 的潜力。
- **计算架构差异**：GPU 平台通常使用 CUDA（Compute Unified Device Architecture）并行计算框架，而本设计选择的昇腾开发板使用的是华为的 CANN（Compute Architecture for Neural Networks）NPU 异构计算架构。
- **深度学习框架差异**：NPU 平台的深度学习框架与 GPU 平台上主流的 Py-

Torch，TensorFlow 等框架具有明显差别。为了更好地支持 NPU 硬件，需要对其他平台的深度学习框架进行适配，可以通过适配张量运算、自动微分等方式完成框架迁移。

本设计采取的模型迁移适配方法，可以分为四个阶段：迁移分析、迁移适配、精度调试与性能调优，总体流程如图4-1所示。

迁移分析包括对模型约束进行说明和对支持度进行分析。模型约束需要评估选择模型迁移的可行性，完成迁移前的准备，并在 GPU 或 CPU 平台上输出精度和性能基线。支持度分析需要分析模型算子是否支持昇腾 NPU，如果存在 NPU 平台尚未提供支持的算子，需要通过修改模型脚本，使用等价算子进行替换等方法解决。

迁移适配可分为模型移植、环境配置和关键特性适配三部分流程。在模型移植中，需要将其他深度学习框架训练得到的模型进行移植，使其能在 NPU 平台被正常加载完成推理。本设计使用昇腾张量编译器（Ascend Tensor Compiler，ATC）工具，将 4.1 节中在 GPU 平台上训练好的 SNN 模型转换为昇腾 NPU 支持的.om 格式离线模型。在环境配置中，完成 CANN 软件基础环境变量的配置，为基于 NPU 平台的推理提供环境支持。在关键特性适配中，开启检测开关和故障预警，对推理过程的中间计算结果进行保障。

精度调试包括对输入数据进行预处理和对模型的超参数配置进行调试，以得到精度更优的推理结果。

性能调优会用到并行策略，IO 调度优化，NPU 亲和适配和内存优化等技术，通过 ATC 工具进行算子调度优化、权重数据重排、等具体操作，对推理过程进行进一步的调优，完成对 NPU 的更优适配，使其能够在昇腾 NPU 上高效执行。

4.3　基于 NPU 平台的软硬件协同优化

随着深度学习模型在边缘计算和端侧设备中的广泛应用，NPU 凭借其高效的神经网络计算能力成为关键硬件载体。过去单纯依赖软件算法层面的更新或是硬件平台的升级的优化思路均难以实现最佳推理效率。为了充分发挥 NPU 平台的计算效率，本节结合硬件平台和软件算法的特性进行软硬件协同优化。

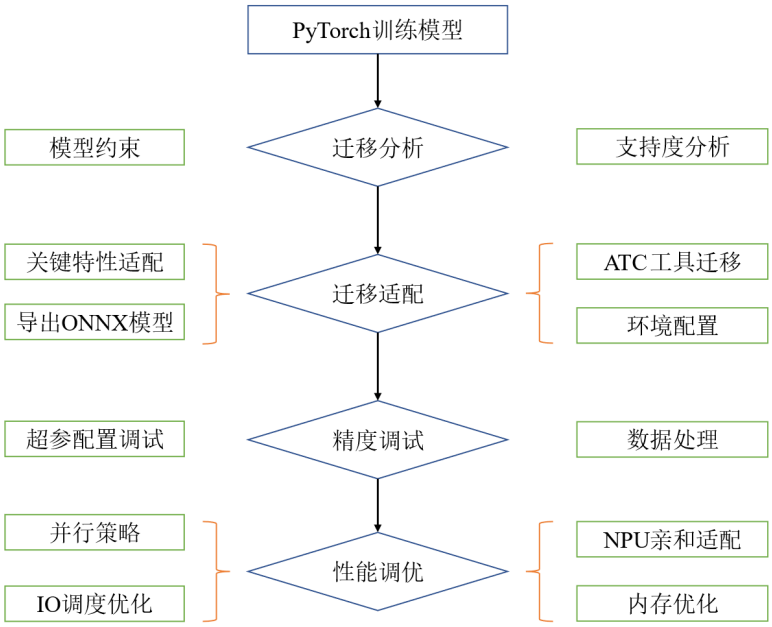

图 4-1 模型迁移总体流程

4.3.1 算子调度优化

神经网络模型可以看成是若干算子的有机组合。具体在昇腾 NPU 平台中，按照网络模型中算子计算单元的差异，可分为 TBE（Tensor Boost Engine）算子与 AI CPU 算子。TBE 算子于开发板的 AI Core 上运行，而 AI CPU 算子则在开发板的 AI CPU 上运行。算子调度优化工作会在模型迁移适配期间开展，其目的在于针对 NPU 平台，在算子层面优化借助其他训练框架获得的神经网络模型。TBE 算子和 AI CPU 算子的转换过程包括图准备、图拆分、图优化、图编译等节点，如图4-2所示。

在算子调度优化过程中，首先使用 ATC 工具的 Parser 对原始 PyTorch 框架训练好的 SNN 模型进行解析，将其模型格式 (.ONNX) 转换成 CANN 模型格式。在图准备阶段，会执行原图的优化工作以及 Infershape 推导等操作。当进行原图优化时，昇腾平台的 GE（Graph Engine，图引擎）会针对模型计算图开展一系列操作，涵盖图的准备、拆分、优化、编译、加载、执行以及管理等。之后，GE 会把中间图传递给 FE（Fusion Engine，融合引擎）。FE 会按照融合规则对图进行融合操作，选取优先级最高的算子类型来进行算子匹配，最终将优化好的完整图反馈给 GE。当 GE 完成图编译操作后，原始的 SNN 模型文件 (.ONNX) 将转化为适配昇腾 AI Core 的离线模型文件 (.om)。

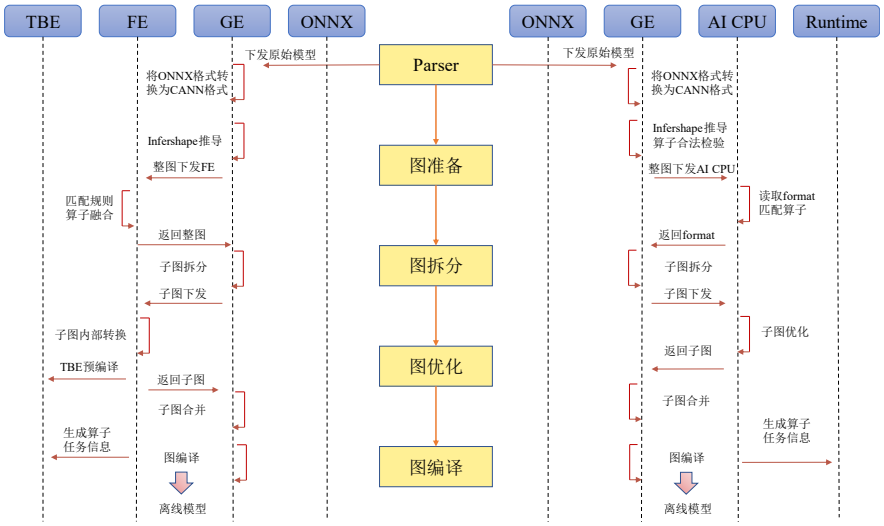

图 4-2　算子调度优化

4.3.2　权值数据重排

在原始 SNN 模型中，隐藏层的特征图通过多维数组存储，包含四个维度(4D)，其格式如下：

- N：批次 (Batch) 的数量。
- C：Channels，特征图通道，彩色图像为 3，灰度图像为 1。
- H：Height，特征图高度。
- W：Width，特征图宽度。

在实际的硬件布局中，由于内存单元是连续的，因此权值数据只能线性存储，上述四个维度在内存中的存储有固定的顺序。在模型训练阶段，经过 Pytorch 框架中的 ToTensor() 处理之后，图片数据通道顺序将变成 N，C，H，W。不论是 GPU 平台还是昇腾 NPU 平台，即使存储的权值数据相同，但不同的存储顺序会导致不一致的数据访问结果，进而得到截然不同的计算效率。

在昇腾 NPU 平台的计算架构中，为了提升通用矩阵乘法运算时数据块的访问效能，将所有张量数据统一调整为 NC_1HWC_0 的五维数据格式。其中 C_0 由平台的核心处理器特性决定，一般情况下，该值等于平台内部 AI Core 中矩阵计算单元的大小。C_1 的计算方法由式4-11得到：

$$C_1 = \lfloor \frac{C + C_0 - 1}{C_0} \rfloor, \tag{4-11}$$

将 Pytorch 框架中的 N，C，H，W 存储顺序转换为 NC_1HWC_0 的过程如图4-3所示，最开始的数据三维尺寸分别为 C，H，W，通过将数据在通道维度上进行分割，可以得到下方所示的 C_1 份 NC_0HW 数据块。将分块后的权值数据进行重新排列，如图右方所示，C1 份 NC_0HW 数据库可在内存中连续排列成 NC_1HWC_0 块重组后的权值数据，此时的数据三维尺寸将更适合平台的计算架构。

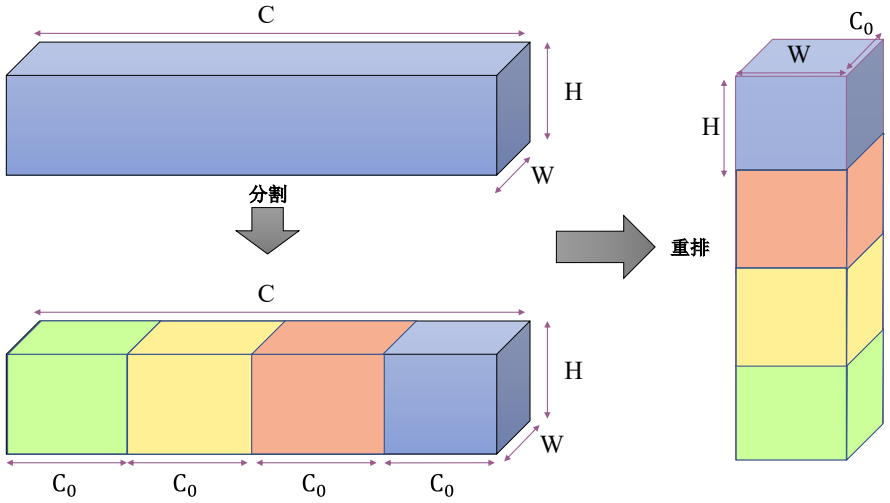

图 4-3 权值数据重排

经过权值数据重排后，片上总线的传输带宽利用率更高，从而可以减少访问共享内存的次数，进而提高计算效率。

4.4 实验与分析

为了验证本文提出的基于 NPU 的 SNN 加速器设计的性能，本节将通过实验与分析，对 SNN 模型在国产 NPU 平台的部署进行评估，并验证本设计的效果。

4.4.1 实验环境

软件环境

- 开发语言：Python. 使用 Python 语言完成 SNN 模型的训练，转换和在 NPU 平台的推理。
- 编程框架：PyTorch, AscendCL. 使用 PyTorch 框架完成对 SNN 模型的训练，使用 AscendCL 完成模型的转换，优化和推理。

硬件环境

本设计选择的 NPU 平台为华为昇腾 Atlas 200I DK A2 开发者套件，该套件搭载昇腾 NPU，包括 1 个 DaVinciV300 AI core（主频 500MHz）和 4 个 TAIS-HANV200M 处理器核（主频 1.0GHz），AI 算力最高可达 8 TOPS。板卡的内存大小为 4GB，类型为 LPDDR4X，支持 ECC，并内置内置 SPI flash 存储，工作电压为 12V，典型功耗 24W。Atlas 200I DK A2 的展示图如4-4所示。

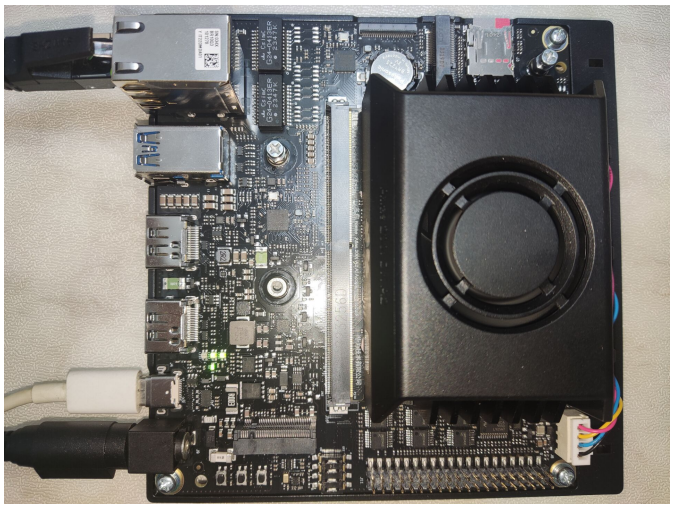

图 4-4　Atlas 200I DK A2

数据集

CIFAR-10 数据集是计算机视觉领域中一个非常著名的基准数据集，主要用于图像分类任务的性能评估。该数据集总共包含 60000 张彩色图像 (训练集 50000 张，测试集 10000 张)，分为 10 个不同的类别，每张图像的尺寸为 32×32 像素。

ImageNet 数据集是一个在计算机视觉领域被广泛应用的大型图像数据集，由斯坦福大学的教授带领创建。其目标是将 WordNet 的大量同义词集用大量带注释的图像填充，助力图像识别算法的发展。该数据集拥有 14197122 张图片，涵盖 2 万多个类别。其中，ImageNet 大规模视觉识别挑战赛（ILSVRC）常用的子集包含 1000 个对象类别，本章将采用 ILSVRC2012 作为实验数据集。

4.4.2 实验过程

本设计的实验过程主要包括三个步骤，分别是原始模型的训练，将原始模型转换成 NPU 平台适配的模型和基于 NPU 平台的模型推理。

1. 模型训练：在 GPU 服务器上分别训练 SNN 模型，并将得到的模型以.pth 格式的文件保存下来。根据前文的分析，基于 QCFS 激活函数的 ANN-to-SNN 转换法训练得到的 SNN 模型精度依赖于时间步长 T 的选择。

   表4-1给出了在 CIFAR-10 数据集上不同时间步长 T 训练得到的 SNN 模型的精度情况。以原始 ANN 模型为基准，整体而言转换得到的 SNN 模型精度会随着 T 的增加逼近 ANN 模型的精度。当 $T = 16$ 时，SNN 模型精度已经和 ANN 模型精度接近，且时间步长适中。

   表4-2给出了在 ImageNet 数据集上不同时间步长 T 训练得到的 SNN 模型的精度情况。整体而言，SNN 模型的精度随着 T 的增加逐渐逼近 ANN 模型。当 $T = 64$ 时，SNN 模型精度和 ANN 模型精度较为接近，且时间步长适中。

表 4-1 不同时间步长训练得到的 SNN 模型精度对比：CIFAR-10

模型结构	ANN	T=2	T=4	T=8	T=16	T=32	T=64
VGG-16	95.49%	91.07%	93.82%	94.77%	95.37%	95.42%	95.43%
ResNet-18	96.02%	78.74%	90.75%	94.56%	95.89%	95.97%	95.96%
ResNet-50	96.76%	84.61%	91.29%	94.18%	96.14%	96.50%	96.47%

表 4-2 不同时间步长训练得到的 SNN 模型精度对比：ImageNet

模型结构	ANN	T=16	T=32	T=64	T=128	T=256	T≥1024
VGG-16	74.27%	51.05%	69.12%	72.81%	73.46%	74.07%	74.25%
ResNet-34	74.30%	59.13%	68.94%	72.34%	73.09%	73.41%	73.44%

2. 模型转换：使用 AscendCL 工具完成模型迁移，在迁移过程中进行针对后续推理过程的优化，迁移命令如下所示：

   atc –model=model.onnx –framework=5 –output=om_model –soc_version=<soc_version>

3. 模型推理：基于 NPU 平台完成对 SNN 模型的推理，并将推理性能与其他平台和模型进行比较。图4-5展示了昇腾 NPU 的推理效果，对于输入图片，

给出其 Top-5 的类别置信度，并计算平均推理延时。

图 4-5　SNN 模型在 NPU 上的推理效果

4.4.3　实验结果

本小节将展示实验过程中的详细实验结果，并根据相关结果进行分析，以验证本设计的有效性。

转换精度的对比

在将模型迁移至 NPU 平台之前，需要先训练好 SNN 模型。由于本设计采取的是 ANN-to-SNN 转换法，因此需要将转换后的 SNN 模型精度分别与转换前 SNN 模型精度以及 ANN 模型精度进行对比。在将 ANN 转换训练得到 SNN 的过程中，根据本节之前的分析，若时间步长 T 较小，则 ANN 与 SNN 之间的转换误差会略大一些，但是会得到更快的推理速度，若时间步长 T 较大，则转换误差会随着 T 增大逐渐趋近于 0，但与此同时，推理延迟会越来越大。

表4-3给出了在 CIFAR-10 数据集上从 ANN 模型训练得到 SNN 模型，再将 SNN 模型迁移适配过程中转换精度的对比。综合考虑转换误差与推理速度，针对 CIFAR-10 数据集训练统一设置 $T = 16$。当原始 ANN 模型结构为 VGG-16 时，最终迁移后的 SNN 模型精度损失为 0.74%；当原始 ANN 模型结构为 ResNet-18 时，最终迁移后的 SNN 模型精度损失为 1%；当原始 ANN 模型结构为 ResNet-50 时，最终迁移后的 SNN 模型精度损失为 1.88%。

表4-4给出了在 ImageNet 数据集上从 ANN 模型训练得到 SNN 模型，再将 SNN 模型迁移适配过程中转换精度的对比。综合考虑转换误差与推理速度，针对 ImageNet 数据集训练统一设置 $T = 64$。当原始 ANN 模型结构为 VGG-16 时，最终迁移后的 SNN 模型精度损失为 3.03%；当原始 ANN 模型结构为 ResNet-34 时，最终迁移后的 SNN 模型精度损失为 3.27%。

表 4-3 转换精度的对比: CIFAR-10

模型结构	ANN	SNN	迁移后的 SNN	总精度损失
VGG-16	95.49%	95.37%	94.75%	0.74%
ResNet-18	96.02%	95.89%	95.02%	1.00%
ResNet-50	96.76%	96.14%	94.88%	1.88%

表 4-4 转换精度的对比: ImageNet

模型结构	ANN	SNN	迁移后的 SNN	总精度损失
VGG-16	74.27%	72.81%	71.24%	3.03%
ResNet-34	74.30%	72.34%	71.03%	3.27%

从实验结果可以发现，迁移后的 SNN 模型相比迁移前的 SNN 模型存在精度损失，这说明在将使用其他框架训练好的模型迁移至 NPU 平台的过程中，由于模型在昇腾 NPU 上的算子适配、数据读写等问题，导致模型推理的性能会出现小幅下降，但在 CIFAR-10 数据集上损失不超过 2%，在 ImageNet 数据集上损失约为 3%。考虑到不同计算平台的硬件架构和编程开发库存在一定差异，且目前相关的模型适配技术还存在相当大的技术空白，这样的精度损失尚在可接受范围内。

不同平台的对比

基于 NPU 平台的 SNN 加速器推理性能与在其他平台上的推理性能对比如表4-5所示，实验统一采用由 ResNet-50 转换训练得到的 SNN 模型进行推理。从结果可以发现，SNN 模型在 NPU 平台上适配后，在速度，功耗，能效各方面均优于 CPU 平台，其推理速度可接近 GPU 平台的一半，为 1.98ms/张，但在能效上约为 GPU 平台的 5.7 倍，达到了 21.04 张/J，实现了非常可观的计算效率。由

于 GPU 平台的硬件架构专门为大规模并行计算进行了优化，并凭借大量计算核心的数据并行处理实现了极快的推理速度，因此，面积有限的 NPU 平台无法在算力上与 GPU 平台做到旗鼓相当。然而，在资源功耗有限的情况下，NPU 平台可以凭借自身的并行计算单元，针对部署的神经网络模型进行加速优化，实现更高的推理能效。

表 4-5　不同平台推理性能

硬件平台	CPU	GPU	NPU
硬件型号	Intel(R)core(TM)i5-9400	NVIDIA RTX 2080Ti	华为 Atlas 200I DK A2
工作频率 (Hz)	2.9G	1.35G	500M
推理速度 (ms/张)	22.97	1.04	1.98
平均功耗 (w)	65	260	24
能效 (张/J)	0.67	3.70	21.04

与国外相关工作的对比

以 CIFAR-10 数据集为应用场景，表4-6展示了本设计与国外其他相关工作的对比情况，所有工作均通过 ANN-to-SNN 转换法训练得到 SNN 模型。Gerlinghoff 等人[40]的工作以 Xilinx Virtex UltraScale+ XCVU13P 这一 FPGA 平台为硬件部署环境，并选择了 AlexNet 作为原始 CNN 模型，实现了在 FPGA 平台的优异性能表现。尽管 FPGA 平台可提供大量的并行逻辑资源，但受限于工作频率和计算架构等因素，在处理神经网络模型推理时的效率不如 NPU 等专门为神经计算优化过的硬件平台，且原始 CNN 模型 AlexNet 属于较为落后的早期模型，因此整体表现不如其他设计。Rueckauer 等人[82]的工作选择了 Loihi 数字电路神经形态芯片作为硬件平台，以 MobileNet 为原始 CNN 模型，取得了在神经计算平台上优秀的性能表现。然而，该工作未针对网络模型的权重共享进行专门优化，且未完全解决模型转换后在硬件层面的脉冲拥堵问题，缺少进一步对算子的优化。本设计采用由 ResNet-50 转换训练得到的 SNN 模型进行推理，通过模型迁移至国产 NPU 平台时的优化，充分发挥硬件架构的计算能力，结合算子调度优化与权值数据重排等方法，实现了最佳的性能表现，展示了国产 NPU 平台的应用前景。

表 4-6 与国外相关工作的对比

	Gerlinghoff 等人[40]	Rueckauer 等人[82]	本设计
硬件平台	Xilinx Virtex UltraScale+ XCVU13P	Loihi	华为 Atlas 200I DK A2
原始模型	AlexNet	MobileNet	ResNet-50
推理速度 (ms/张)	69.93	4.35	1.98
准确率 (%)	80.60	91.48	94.88
能效 (张/J)	3.04	9.80	21.04

4.5 本章小结

本章使用基于 ATC 工具的模型迁移适配方法，将利用 PyTorch 框架训练得到的脉冲神经网络模型迁移至华为昇腾 NPU 平台，并结合针对 NPU 平台的软硬件协同优化，实现了 SNN 模型在 NPU 平台的高能效部署。此外，本设计也是首次将第三代神经网络的代表模型——SNN 部署在国产 NPU 平台上，具有一定的攻关意义。实验证明，适配后的 SNN 模型在 NPU 平台上的推理性能可比肩 GPU 平台的同时，拥有更高的计算能效。

第五章　基于边缘计算平台的实时目标检测系统

为了验证本文提出的加速器设计方法在边缘智能任务中的应用效果，本章节构建了一个基于边缘计算平台的实时目标检测系统，并在该系统中应用了本文提出的卷积神经网络加速器设计方法。实时目标检测系统是计算机视觉领域的重要技术落地应用，从日常生活到工业应用、公共安全领域均具有深远影响。在实际应用中，实时检测通常要求模型在毫秒级响应的速度要求下保持高准确率，这要求模型的设计兼具速度与精度，同时，检测系统的实时性也要求模型必须部署在边缘端实现更自然的协作交互，最后，考虑到边缘计算平台的资源有限性，在设计系统时也应考虑系统的整体资源消耗不能过大，避免系统无法正常运行。本章将从研发背景，需求分析，系统设计与整体效果四个方面来详细阐述本文设计的基于边缘计算平台的实时目标检测系统。

5.1　研发背景

在深度学习技术取得重大突破之前，传统目标检测方法存在一系列局限性，包括但不限于以下几点：

1) 算法效率低：早期方法（如 Viola-Jones、HOG+SVM）依赖手工设计特征，需要经过滑动窗口 + 特征提取 + 分类等多阶段特征处理，计算复杂度高。
2) 精度与泛化能力不足：基于手工特征设计的系统对光照变化、遮挡等场景鲁棒性差，复杂场景下误检率较高。
3) 应用场景受限：由于早期方法的算法效率较低，使用手工特征设计的系统在识别速度指标上存在瓶颈，导致其无法满足自动驾驶、视频监控等应用的实时性需求。

近年来，随着自动驾驶的兴起，移动端应用爆发和工业自动化突破，目标检测系统的主流应用场景对实时性的要求随之水涨船高。深度学习技术的发展使得深度神经网络在一系列任务中都取得了惊人的精度与速度突破，但随之而来

的是模型的愈发复杂与庞大，无法直接部署在资源功耗受限的边缘计算平台上，与实现边缘智能应用场景的普适化应用还存在一定距离。

算法层面与硬件层面的协同进化使得边缘实时目标检测系统的实现落地成为可能。在算法层面，以 YOLO 系列为代表的目标检测模型将目标检测任务简化为单次网络推理，速度较 Faster R-CNN 提升 10 倍以上。知识蒸馏、剪枝、量化等模型压缩技术的进步在保持精度的同时，可将模型尺寸压缩压缩至原先的$\frac{1}{10}$，对边缘计算平台更加友好。在硬件层面，NVIDIA Jetson、Xilinx FPGA 与华为昇腾计算 NPU 等边缘计算芯片的推出与升级，使实时检测模型能在低功耗设备（如摄像头、机器人）上顺利落地。

本章旨在实现一个基于边缘计算平台的实时目标检测系统，能通过实时输入的视频流，在低延迟条件下完成对关键物体的准确识别。

5.2　需求分析

本节从用户的实际使用需求出发，基于用户体验与应用场景进行系统的用户需求分析，再结合用户需求分析的结果与系统的软硬件设计情况，对系统的功能需求进行分析。

5.2.1　用户需求分析

从用户使用体验与实际应用场景出发，边缘实时目标检测系统应考虑的用户需求应当包括以下几个方面：

1) 体验友好的交互模块：系统界面的设计应注重用户的实际体验。考虑到实际操作系统的人员往往为不具备专业计算机知识的工业质检员或交通管理员等其他职业人员，系统的交互模块应尽可能去专业化，保留简洁直接的说明与指引，为用户操作降低门槛。
2) 安全隐私的认证控制：系统的认证模块包括用户信息的注册，登陆，修改密码等，确保用户能够安全、便捷地访问和使用系统。此外，对于实时识别，还应该考虑数据采集开关（如关闭人脸检测）、结果匿名化展示（模糊敏感信息）等需求，充分保证用户的安全隐私。

3) 方便调整的检测视角：系统的检测效果受采集视角影响，如果视角固定不变，那么检测场景十分受限，且不便于用户的实际使用。为了方便用户更好地使用系统进行目标检测，系统应支持实时的视角调整，让用户随时随地可以方便地使用。
4) 快速准确的可视结果：系统的检测结果应当支持实时的可视化显示，并保证执行速度快，推理结果准和显示界面明确。通过简洁明了的可视化界面，让用户清晰地了解系统的检测结果，对任务的效果做出及时准确的判断与反馈。

5.2.2 功能需求分析

功能需求分析是连接用户期望与技术实现的桥梁，本章实现的基于边缘计算平台的实时目标检测系统的功能需求应包括以下几个方面：

1) 系统的启动与关闭：为了便于实时目标检测系统的正常使用，应当基于边缘计算平台为系统的启动与关闭提供友好的接口调用，让系统能根据实际使用需要方便快捷地启动与关闭，并根据认证模块完成用户的登陆，保证系统安全性。
2) 灵活化的功能定制：考虑到系统的使用可能有多种应用场景，不一定仅用于实时检测，在实际使用中还可能存在录像分析，图片分析等任务。在设计时还应保留灵活的模式切换接口，使系统能灵活地应对不同要求的任务。
3) 保证目标检测效果：目标检测是系统的核心功能，保证目标检测任务能正常进行是系统的最基本功能要求。本系统基于前文提出的目标检测模型加速器设计方法，保证核心功能的边缘部署实现。
4) 持久化的待机部署：考虑到目标检测任务往往需要系统长时间待机运行，且边缘计算平台往往受到内存等资源的限制，应当为系统提供定时的内存清理功能，以实现目标检测功能的持久化运行。

基于以上详细的需求分析，本研究旨在开发一个边缘端实时目标检测系统，尽可能地满足各类需要目标检测功能的场景，为 AI 应用的边缘落地提供更多选项。

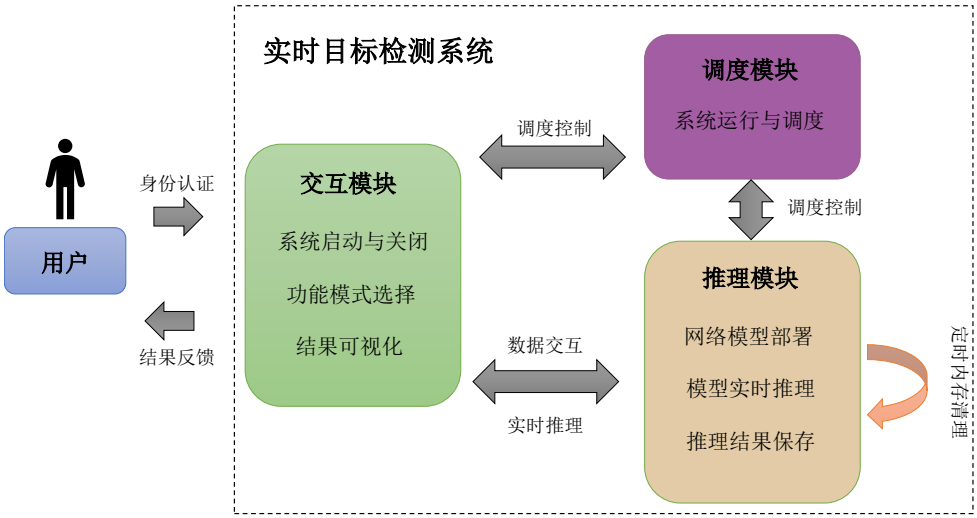

图 5-1　实时检测系统整体结构

5.3　系统设计

基于上节详细分析的需求，本节给出了系统的详细结构设计，如图5-1所示，实时目标检测系统主要由三个关键模块组成，分别是交互模块、调度模块和推理模块。

交互模块是用户操作访问检测系统的接口，用户通过身份认证获取进入系统的许可，通过设计好的接口对系统的启动与关闭进行操作，同时可以选择系统的功能模式 (实时检测，视频检测，图片检测)。当系统完成推理后，会实时将推理结果可视化，并通过交互模块将结果反馈给用户。

调度模块属于边缘计算平台的一部分。实时检测系统搭载在开发板上，需要接受开发板处理核心的控制。在处理器核心中，调度模块完成对整个实时目标检测系统的运行调度。

推理模块：推理模块是检测系统的核心模块。在该模块，将前文设计的 YOLOv4-tiny 模型通过一系列加速优化后部署在开发板的计算核心中，完成实时推理，并将推理结果临时保存下来。

5.4　系统实现

基于前文的需求分析和系统设计，本节将详细介绍系统的具体实现过程。在本节中，将介绍系统的硬件平台，开发环境和关键功能的实现细节。

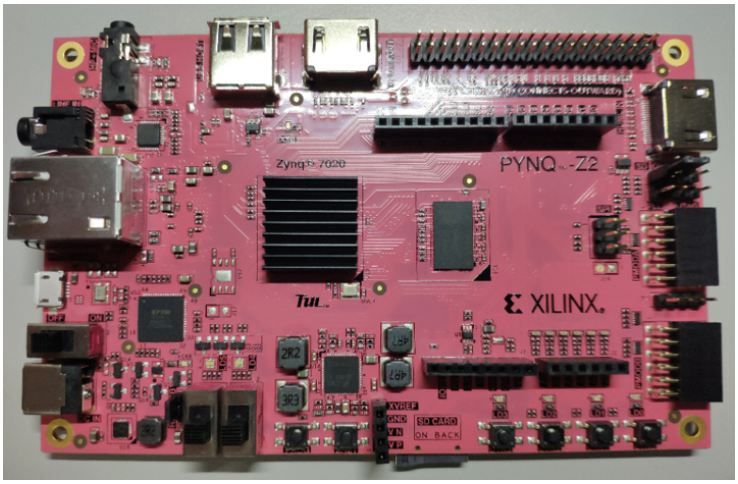

图 5-2 PYNQ-z2 开发板实物图

5.4.1 硬件平台

实时目标检测系统搭载的边缘计算平台为 PYNQ-z2 开发板，PYNQ-z2 是由 Xilinx 公司推出的一款基于 Zynq-7000 SoC 的可编程嵌入式开发板，结合了 FPGA 的并行计算能力和 ARM 处理器的灵活控制。平台板载有 650MHz 的双核处理器，内含 DDR3 内存控制器，具有 8 个 DMA 通道和 4 个高性能 AXI3 从端口，FPGA 逻辑资源包括 85K 逻辑单元（约 13300 个 LUT），220 个 DSP 片，4.9Mb Block RAM，外设接口包括千兆以太网口，HDMI 输入/输出，音频编解码器和 USB 接口等。由于开发板包括 ARM 处理器和 Linux 操作系统，可以利用该处理器在操作系统层面对实时目标检测系统进行控制。PYNQ-z2 开发板实物图5-2所示：

5.4.2 开发环境

实时目标检测系统基于 Linux 操作系统，使用多种编程语言 (Python，C++) 和多种开发环境 (vitis HLS,Vivado, Dash) 进行设计。目标检测模型使用基于 PyTorch 和 OpenCV 框架训练得到的 YOLOv4-tiny 模型，经过轻量化操作后部署在开发板上。系统的调度模块在 Linux 环境下进行操作，推理模块通过板卡加载定制的神经网络算子 IP 核集成实现，交互模块通过 Python Dash 框架实现，以 Web 应用的形式供用户交互，并将结果可视化展现。

5.4.3　功能实现

本系统的主要模块包括三部分，分别是交互模块，调度模块和推理模块，本节将详细介绍模块的实现方法。

交互模块是用户接触系统的直接接口，基于前文提到的交互模块设计需求，本系统采用 Python Dash 框架进行前端交互模块的实现。

Dash 是一个用于构建数据分析应用程序的 Python 框架，应用程序在 Web 浏览器中运行，使用 Flask 作为 Web 服务器，并且通过 HTML、CSS 和 JavaScript 来呈现前端页面。

图5-3展示了系统的登录界面，用户需要在此进行身份信息的注册，注册完成后，可以通过账号密码登录系统。

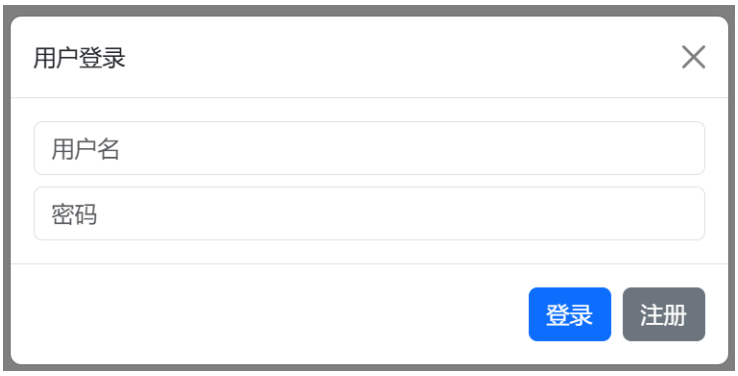

图 5-3　登录界面

当完成登录后，用户将进入功能选择界面，如图5-4所示，用户可以从图片检测，视频检测，实时监测三种模式中选择一种。图片检测支持 jpg/png 格式的图片上传进行检测，视频检测支持 mp4/avi 格式的视频上传进行检测，实时检测则通过外接的摄像头进行实时的目标识别检测。

图 5-4　功能选择界面

调度模块不对用户展示，在开发板运行的 Linux 操作系统的后台提供服务。开发者可以借助终端界面完成基于 Linux 命令行的调度控制操作。

推理模块中包含经过软硬件协同设计优化后的目标检测神经网络模型，其详细设计流程如图5-5所示，首先基于 Pytorch 训练框架得到训练好的 YOLOv4-tiny 模型，再经过轻量化处理，由 HLS 工具得到各类 IP 算子并映射到开发板的并行逻辑资源上，最终完成模型部署。推理结果会临时保存在片上的缓冲区中，等待处理器调度将推理结果返回给交互模块。

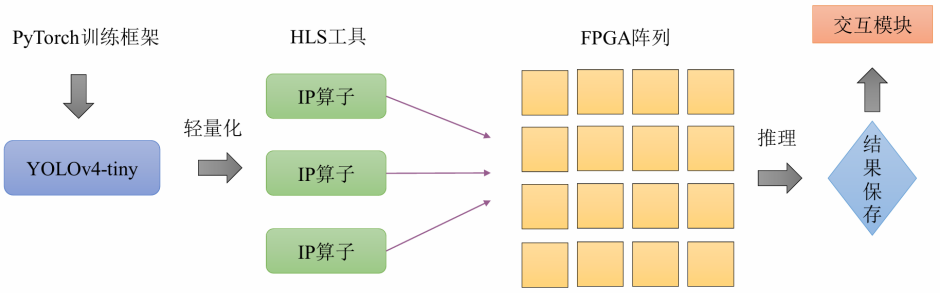

图 5-5　推理模块设计

5.5　效果展示

本节将通过直观的应用案例展示系统的实际效果。用户在登录系统后，选择对应的功能模式，按照提示框进行操作，系统便可返回对应的检测结果。

图 5-6　图片检测效果

图5-6展示了图片检测模式的效果，用户上传需要检测的图片，系统会对图片中的关键物体进行识别，以检测框的形式将物体圈出，并给出类别和置信度。

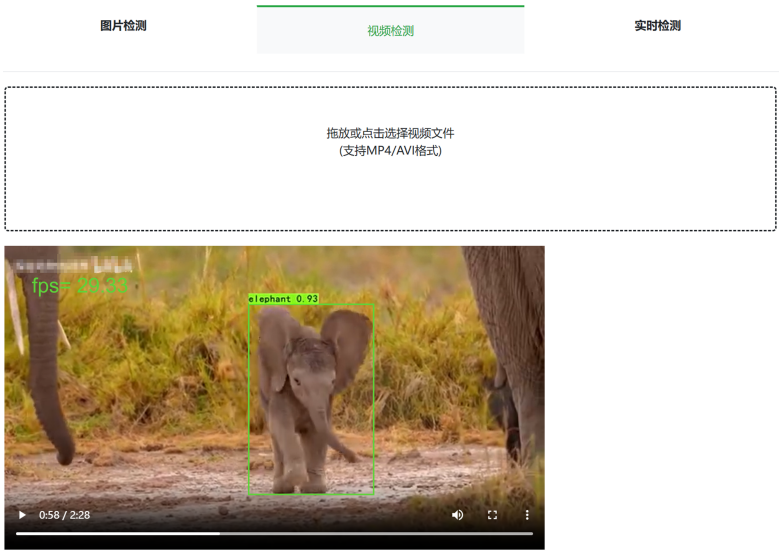

图 5-7　视频检测效果

图5-7展示了视频检测模式的效果，将待检测视频通过交互模块上传后，系统会按照一定的帧率 (视频左上角) 对视频中的关键物体进行识别，以检测框的形式将物体圈出，给出类别和置信度。

图 5-8　实时检测效果

图5-8展示了实时检测模式的效果，系统采用 USB3.0 接口接入高清摄像头，实现实时视频流的低延迟传输，并保持 30FPS 的稳定帧率。系统接收实时输入后立刻反馈检测结果，给出视频帧中的关键物体识别种类和置信度。

5.6 本章小结

本章详细介绍了基于 PYNQ-z2 边缘计算平台的实时目标检测系统。首先介绍了实时目标检测系统的研发背景，强调了在边缘计算平台部署边缘智能任务的重要意义。再通过需求分析，明确了系统的设计方向和整体架构。然后通过模块化的结构设计，结合前文的加速器设计方法，将 YOLOv4-tiny 模型部署在开发板上。并借助一系列软件开发工具，完成了拥有交互模块，调度模块和推理模块的完整系统设计。最后，通过一个实际应用案例，展示了系统的实时检测效果。

第六章　总结与展望

随着人工智能与物联网技术的普及，越来越多的边缘场景都被配以智能模型的辅助决策。边缘智能是一种融合了边缘计算和人工智能的新兴研究方向，它将神经网络模型的推理过程从云端下放到网络边缘终端，使边缘设备能在靠近数据源的位置进行数据处理和智能决策。本文围绕边缘智能这一核心话题，讲述了第二代神经网络和第三代神经网络在边缘场景的部署现状与技术挑战，阐明了在边缘计算设备有限的资源条件下，设计低功耗高能效的神经网络加速器是有效的破局之法。

在设计神经网络加速器时，必须考虑网络模型本身的结构特点与边缘计算平台的硬件特性，利用软硬件协同优化的方法，提升加速器的性能。本文从两种极具代表性的神经网络模型出发，分别在两种不同的边缘计算平台进行模型适配，以充分发挥硬件平台的潜力，实现高性能推理。结合本文的研究内容，主要贡献包括以下几点：

1. 本文基于 FPGA 平台提出了一种低功耗的卷积神经网络加速器设计。该设计以 YOLOv4-tiny 模型为基础，通过剪枝量化的轻量化处理，使模型更加适应 FPGA 平台有限的资源。此外，通过算子重构方法，将 YOLOv4-tiny 模型最核心的卷积算子进行了更适配硬件平台的转换。同时，针对硬件平台特性进行了数据流相关的优化。最后，通过实验验证了该设计在 FPGA 平台的高性能表现。
2. 本文基于国产 NPU 平台提出了一种高能效的脉冲神经网络加速器设计。该设计采用 ANN-to-SNN 转换法得到训练好的 SNN 模型，再利用 ATC 工具实现模型从原始硬件平台到 NPU 平台的迁移适配，开创性地将 SNN 成功部署到国产 NPU 计算平台上。通过实验证明，SNN 在 NPU 平台拥有超越 GPU 的推理能效。
3. 本文基于提出的低功耗卷积神经网络加速器设计方法，实现了一个落地在

FPGA 开发板上的边缘实时目标检测系统。该系统拥有良好的交互性，支持多种功能，能以流畅的帧率完成实时视频流的关键物体检测，是具有代表性的边缘智能应用。

在本文提出的方法基础上，未来的边缘智能研究还有很多可以拓展和深入研究的方向，包括但不限于以下几个方面：

1. 降低设计门槛：以 FPGA 平台为例，在 FPGA 平台上的编程往往需要 AI 领域的研究人员对硬件描述语言以及电路原理有所了解，且 FPGA 平台缺乏像 TensorFlow, PyTorch 这样成熟的软件开发框架的支持，开发门槛非常高。尽管目前已经有 Vitis HLS 这样较高抽象层级的工具出现，但是整体开发难度还是比较高，期待未来有更便捷的开发工具出现。
2. 国产生态攻关：目前最主流的 AI 部署平台仍然是以英伟达和 AMD 主导的 GPU 平台。虽然目前已经有一些国产化 AI 计算平台出现，但这些平台在性能，开发框架，整体生态方面仍和国外主流平台存在差距。通过国产化攻关，期待未来能实现更全面，更完整的国产 AI 计算生态。
3. 边缘智能落地：对于许多 AI 应用，只有真正落地才能让全人类享受到科技发展带来的红利。期待未来可以进一步加强前沿 AI 应用的落地与普及，为人类文明的发展贡献更多智慧。

参考文献

[1] MCCULLOCH W S, PITTS W. A logical calculus of the ideas immanent in nervous activity[J]. The bulletin of mathematical biophysics, 1943, 5: 115-133.

[2] ROSENBLATT F. The perceptron: a probabilistic model for information storage and organization in the brain.[J]. Psychological review, 1958, 65(6): 386.

[3] KHAN S, RAHMANI H, SHAH S A A, et al. A guide to convolutional neural networks for computer vision[J]., 2018.

[4] MINSKY M, PAPERT S A. Perceptrons, reissue of the 1988 expanded edition with a new foreword by Léon Bottou: an introduction to computational geometry[M]. MIT press, 2017.

[5] CYBENKO G. Approximation by superpositions of a sigmoidal function[J]. Mathematics of control, signals and systems, 1989, 2(4): 303-314.

[6] RUMELHART D E, HINTON G E, WILLIAMS R J. Learning representations by back-propagating errors[J]. Nature, 1986, 323(6088): 533-536.

[7] HINTON G E, OSINDERO S, TEH Y W. A fast learning algorithm for deep belief nets[J]. Neural computation, 2006, 18(7): 1527-1554.

[8] BROWN T, MANN B, RYDER N, et al. Language models are few-shot learners[J]. Advances in neural information processing systems, 2020, 33: 1877-1901.

[9] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[10] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[J]. Advances in neural information processing systems, 2012, 25.

[11] HUBARA I, COURBARIAUX M, SOUDRY D, et al. Binarized neural networks[J]. Advances in neural information processing systems, 2016, 29.

[12] SZEGEDY C, LIU W, JIA Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 1-9.

[13] LI F, LIU B, WANG X, et al. Ternary weight networks[J]. ArXiv preprint arXiv:1605.04711, 2016.

[14] WANG K, LIU Z, LIN Y, et al. Haq: Hardware-aware automated quantization with mixed precision[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019: 8612-8620.

[15] MIYASHITA D, LEE E H, MURMANN B. Convolutional neural networks using logarithmic data representation[J]. ArXiv preprint arXiv:1603.01025, 2016.

[16] LI Y, DONG X, WANG W. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks[J]. ArXiv preprint arXiv:1909.13144, 2019.

[17] JAIN S, VENKATARAMANI S, SRINIVASAN V, et al. Biscaled-dnn: Quantizing long-tailed datastructures with two scale factors for deep neural networks[C]//Proceedings of the 56th Annual Design Automation Conference 2019. 2019: 1-6.

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

[19] GUO M H, XU T X, LIU J J, et al. Attention mechanisms in computer vision: A survey[J]. Computational visual media, 2022, 8(3): 331-368.

[20] LI H, KADAV A, DURDANOVIC I, et al. Pruning filters for efficient convnets[J]. ArXiv preprint arXiv:1608.08710, 2016.

[21] LIU Z, LI J, SHEN Z, et al. Learning efficient convolutional networks through network slimming[C]//Proceedings of the IEEE international conference on computer vision. 2017: 2736-2744.

[22] YU R, LI A, CHEN C F, et al. Nisp: Pruning networks using neuron importance score propagation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 9194-9203.

[23] TAVANAEI A, GHODRATI M, KHERADPISHEH S R, et al. Deep learning in spiking neural networks[J]. Neural networks, 2019, 111: 47-63.

[24] NEIL D, LIU S C. Minitaur, an event-driven FPGA-based spiking network accelerator[J]. IEEE transactions on very large scale integration (VLSI) systems, 2014, 22(12): 2621-2628.

[25] WANG Q, LI Y, SHAO B, et al. Energy efficient parallel neuromorphic architectures with approximate arithmetic on FPGA[J]. Neurocomputing, 2017, 221: 146-158.

[26] MOSTAFA H. Supervised learning based on temporal coding in spiking neural networks[J]. IEEE transactions on neural networks and learning systems, 2017, 29(7): 3227-3235.

[27] ZHANG C, QIAO G, HU S, et al. A versatile neuromorphic system based on simple neuron model[J]. AIP Advances, 2019, 9(1).

[28] ZHANG J, WU H, WEI J, et al. An asynchronous reconfigurable SNN accelerator with event-driven time step update[C]//2019 IEEE Asian Solid-State Circuits Conference (A-SSCC). 2019: 213-216.

[29] ABDERRAHMANE N, MIRAMOND B. Information coding and hardware architecture of spiking neural networks[C]//2019 22nd Euromicro Conference on Digital System Design (DSD). 2019: 291-298.

[30] KUANG Z, WANG J, YANG S, et al. Digital implementation of the spiking neural network and its digit recognition[C]//2019 Chinese Control And Decision Conference (CCDC). 2019: 3621-3625.

[31] GUO S, WANG L, WANG S, et al. A systolic SNN inference accelerator and its co-optimized software framework[C]//Proceedings of the 2019 Great Lakes Symposium on VLSI. 2019: 63-68.

[32] LOSH M, LLAMOCCA D. A low-power spike-like neural network design[J]. Electronics, 2019, 8(12): 1479.

[33] JU X, FANG B, YAN R, et al. An FPGA implementation of deep spiking neural networks for low-power and fast classification[J]. Neural computation, 2020, 32(1): 182-204.

[34] HAN J, LI Z, ZHENG W, et al. Hardware implementation of spiking neural networks on FPGA[J]. Tsinghua Science and Technology, 2020, 25(4): 479-486.

[35] FANG H, MEI Z, SHRESTHA A, et al. Encoding, model, and architecture: Systematic optimization for spiking neural network in FPGAs[C]//Proceedings of the 39th International Conference on Computer-Aided Design. 2020: 1-9.

[36] WANG S Q, WANG L, DENG Y, et al. SIES: A novel implementation of spiking convolutional neural network inference engine on field-programmable gate array[J]. Journal of Computer Science and Technology, 2020, 35: 475-489.

[37] AUNG M T L, QU C, YANG L, et al. DeepFire: Acceleration of convolutional spiking neural network on modern field programmable gate arrays[C]//2021 31st International Conference on Field-Programmable Logic and Applications (FPL). 2021: 28-32.

[38] LI S, ZHANG Z, MAO R, et al. A fast and energy-efficient SNN processor with adaptive clock/event-driven computation scheme and online learning[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2021, 68(4): 1543-1552.

[39] ZHENG H, GUO Y, YANG X, et al. Balancing the cost and performance tradeoffs in SNN processors[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2021, 68(9): 3172-3176.

[40] GERLINGHOFF D, WANG Z, GU X, et al. E3NE: An end-to-end framework for accelerating spiking neural networks with emerging neural encoding on FPGAs[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 33(11): 3207-3219.

[41] ZHANG J, WANG R, PEI X, et al. A fast spiking neural network accelerator based on BP-STDP algorithm and weighted neuron model[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2021, 69(4): 2271-2275.

[42] PANCHAPAKESAN S, FANG Z, LI J. SyncNN: Evaluating and accelerating spiking neural networks on FPGAs[J]. ACM Transactions on Reconfigurable Technology and Systems, 2022, 15(4): 1-27.

[43] LIU Y, CHEN Y, YE W, et al. FPGA-NHAP: A general FPGA-based neuromorphic hardware acceleration platform with high speed and low power[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2022, 69(6): 2553-2566.

[44] YE W, CHEN Y, LIU Y. The implementation and optimization of neuromorphic hardware for supporting spiking neural networks with MLP and CNN topologies[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 42(2): 448-461.

[45] CHEN Q, GAO C, FANG X, et al. Skydiver: A spiking neural network accelerator exploiting spatio-temporal workload balance[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 41(12): 5732-5736.

[46] SOMMER J, ÖZKAN M A, KESZOCZE O, et al. Efficient hardware acceleration of sparsely active convolutional spiking neural networks[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 41(11): 3767-3778.

[47] LIU H, CHEN Y, ZENG Z, et al. A low power and low latency FPGA-based spiking neural network accelerator[C]//2023 International Joint Conference on Neural Networks (IJCNN). 2023: 1-8.

[48] WANG Z, ZHONG Y, CUI X, et al. A spiking neural network accelerator based on ping-pong architecture with sparse spike and weight[C]//2023 IEEE International Symposium on Circuits and Systems (ISCAS). 2023: 1-5.

[49] LI J, SHEN G, ZHAO D, et al. Firefly: A high-throughput hardware accelerator for spiking neural networks with efficient dsp and memory optimization[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2023, 31(8): 1178-1191.

[50] BENJAMIN B V, GAO P, MCQUINN E, et al. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations[J]. Proceedings of the IEEE, 2014, 102(5): 699-716.

[51] SCHEMMEL J, BRÜDERLE D, GRÜBL A, et al. A wafer-scale neuromorphic hardware system for large-scale neural modeling[C]//2010 ieee international symposium on circuits and systems (iscas). 2010: 1947-1950.

[52] QIAO N, MOSTAFA H, CORRADI F, et al. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses[J]. Frontiers in neuroscience, 2015, 9: 141.

[53] MORADI S, QIAO N, STEFANINI F, et al. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)[J]. IEEE transactions on biomedical circuits and systems, 2017, 12(1): 106-122.

[54] MEROLLA P A, ARTHUR J V, ALVAREZ-ICAZA R, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface[J]. Science, 2014, 345(6197): 668-673.

[55] FURBER S B, GALLUPPI F, TEMPLE S, et al. The spinnaker project[J]. Proceedings of the IEEE, 2014, 102(5): 652-665.

[56] DAVIES M, SRINIVASA N, LIN T H, et al. Loihi: A neuromorphic manycore processor with on-chip learning[J]. Ieee Micro, 2018, 38(1): 82-99.

[57] YANG Y S, KIM Y. Recent trend of neuromorphic computing hardware: Intel's neuromorphic system perspective[C]//2020 International SoC Design Conference (ISOCC). 2020: 218-219.

[58] PEI J, DENG L, SONG S, et al. Towards artificial general intelligence with hybrid Tianjic chip architecture[J]. Nature, 2019, 572(7767): 106-111.

[59] LI Y, WANG Z, MIDYA R, et al. Review of memristor devices in neuromorphic computing: materials sciences and device challenges[J]. Journal of Physics D: Applied Physics, 2018, 51(50): 503002.

[60] CHUA L. Memristor-the missing circuit element[J]. IEEE Transactions on circuit theory, 1971, 18(5): 507-519.

[61] HODGKIN A L, HUXLEY A F. A quantitative description of membrane current and its application to conduction and excitation in nerve[J]. The Journal of physiology, 1952, 117(4): 500.

[62] IZHIKEVICH E M. Simple model of spiking neurons[J]. IEEE Transactions on neural networks, 2003, 14(6): 1569-1572.

[63] JOLIVET R, GERSTNER W. The spike response model: a framework to predict neuronal spike trains[C]//International Conference on Artificial Neural Networks. 2003: 846-853.

[64] DIEHL P U, COOK M. Unsupervised learning of digit recognition using spike-timing-dependent plasticity[J]. Frontiers in computational neuroscience, 2015, 9: 99.

[65] FAIRHALL A L, LEWEN G D, BIALEK W, et al. Efficiency and ambiguity in an adaptive neural code[J]. Nature, 2001, 412(6849): 787-792.

[66] ALAM S H, FOSHIE A, ROSE G. A runtime-reconfigurable hardware encoder for spiking neural networks[C]//Proceedings of the Great Lakes Symposium on VLSI 2023. 2023: 203-206.

[67] HEBB D O. The organization of behavior: A neuropsychological theory[M]. Psychology press, 2005.

[68] CAPORALE N, DAN Y. Spike timing–dependent plasticity: a Hebbian learning rule[J]. Annu. Rev. Neurosci., 2008, 31(1): 25-46.

[69] QIAO G, NING N, ZUO Y, et al. Direct training of hardware-friendly weight binarized spiking neural network with surrogate gradient learning towards spatio-temporal event-based dynamic data recognition[J]. Neurocomputing, 2021, 457: 203-213.

[70] QIAO G, NING N, ZUO Y, et al. Batch normalization-free weight-binarized SNN based on hardware-saving IF neuron[J]. Neurocomputing, 2023, 544: 126234.

[71] VAZQUEZ R. Izhikevich neuron model and its application in pattern recognition[J]. Australian Journal of Intelligent Information Processing Systems, 2010, 11(1): 35-40.

[72] JIAO X, YIN Y, SHANG L, et al. Tinybert: Distilling bert for natural language understanding[J]. ArXiv preprint arXiv:1909.10351, 2019.

[73] 谢坤鹏, 仪德智, 刘义情, 等. SAF-CNN：面向嵌入式 FPGA 的卷积神经网络稀疏化加速框架[J/OL]. 计算机研究与发展, 2023, 60(5): 1053-1072. https://crad.ict.ac.cn/cn/article/doi/10.7544/issn1000-1239.202220735. DOI: 10.7544/issn1000-1239.202220735.

[74] CONG J, XIAO B. Minimizing computation in convolutional neural networks[C]//International conference on artificial neural networks. 2014: 281-290.

[75] PREUßER T B, GAMBARDELLA G, FRASER N, et al. Inference of quantized neural networks on heterogeneous all-programmable devices[C]//2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). 2018: 833-838.

[76] NAKAHARA H, YONEKAWA H, FUJII T, et al. A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA[C]// Proceedings of the 2018 ACM/SIGDA International Symposium on field-programmable gate arrays. 2018: 31-40.

[77] MONTGOMERIE-CORCORAN A, TOUPAS P, YU Z, et al. SATAY: A streaming architecture toolflow for accelerating YOLO models on FPGA devices[C]//2023 International Conference on Field Programmable Technology (ICFPT). 2023: 179-187.

[78] HELLER D, RIZK M, DOUGUET R, et al. Marine objects detection using deep learning on embedded edge devices[C]//2022 IEEE International Workshop on Rapid System Prototyping (RSP). 2022: 1-7.

[79] RUECKAUER B, LUNGU I A, HU Y, et al. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification[J]. Frontiers in neuroscience, 2017, 11: 682.

[80] BU T, FANG W, DING J, et al. Optimal ANN-SNN Conversion for High-accuracy and Ultra-low-latency Spiking Neural Networks[C/OL]//International Conference on Learning Representations. 2022. https://openreview.net/forum?id=7B3IJMM1k_M.

[81] BENGIO Y, LÉONARD N, COURVILLE A. Estimating or propagating gradients through stochastic neurons for conditional computation[J]. ArXiv preprint arXiv:1308.3432, 2013.

[82] RUECKAUER B, BYBEE C, GOETTSCHE R, et al. NxTF: An API and compiler for deep spiking neural networks on Intel Loihi[J]. ACM Journal on Emerging Technologies in Computing Systems (JETC), 2022, 18(3): 1-22.

致　　谢

还没来得及从三年前初至南大的恍惚中缓过神来，却发现自己已经悄然走到了研究生时光的尽头。多希望时间可以倒转，让我重新再体验与南京大学的邂逅，与 RINC 研究组的相遇，与诸位友人的相识。在南京大学求学的三年，是幸福且满载回忆与收获的三年，在此，我由衷地向研究生期间给予我帮助的老师，同学，亲人们表达我最真挚的谢意。

感谢我的导师申富饶教授。记得在入学时，您曾说过每周讨论是我们不断进步的好方法，当我们经过一段时间再回头看曾经记录下的讨论内容，自会觉得收获满满。诚如您所言，学生现在回顾过去，总能通过与您讨论的内容拾起一些回忆，一些体验，一些历经沧桑却依旧发光的东西。感谢您对学生孜孜不倦的教诲，您的言传身教，将会是学生一生受用的宝贵财富。

感谢我的同门 RINC 研究组的各位。与你们一起生活学习的时光充满了幸福与快乐，你们是科研路上的好伙伴，我们曾一起勇攀高峰；你们是运动场上的好队友，我们曾一起挥洒汗水；你们是柏油路上的好搭子，我们曾一起踏过岁月。相见有时，友谊无限，感谢你们的陪伴，愿我们的友谊在未来跨越山海，超越时间。

感谢我的家人与朋友们。你们的支持就是我最大的底气，让我可以放心地把后背交给你们，勇敢向前冲。感谢你们的陪伴与关心，愿我们一直相互扶持，迈向更广阔的明天。

三年时光，很短，也很长。我想我一辈子也不会忘记在南大的这段经历，青春易逝，记忆长存，感谢南大，愿我能成为你的骄傲。

简历与科研成果

基本信息

俞诗航，男，汉族，2000 年 10 月出生，湖北省黄石人。

教育背景

2022 年 9 月 - 2025 年 6 月	南京大学计算机学院	硕士
2018 年 9 月 - 2022 年 6 月	湖南大学信息科学与工程学院	本科

攻读硕士学位期间完成的学术成果

俞诗航, 易梦军, 吴洲, 申富饶, 赵健. 神经形态计算: 从脉冲神经网络到边缘部署 [J]. 软件学报,2025,36(04):1758-1795.DOI:10.13328/j.cnki.jos.007298.

攻读硕士学位期间完成的竞赛成果

2024 华为软件精英挑战赛——江山赛区二等奖

攻读硕士学位期间参与的科研课题

1. 数字化安全管控边缘计算装置自主可控关键技术研究及应用 (项目编号 5700-202319302A-1-1-ZN，负责该项目研究的边缘计算部分)
2. 基于神经可塑性的脉冲网络高效学习机制与类脑智能系统 (科技创新 2030 项目，负责脉冲神经网络研究部分)