密级:_____

中国科学院大学
University of Chinese Academy of Sciences

硕士学位论文

卷积神经网络加速器研究

作者姓名: 张泽鲲

指导教师: 乔树山 研究员 中国科学院微电子研究所

学位类别: 工程硕士

学科专业: 集成电路工程

培养单位: 中国科学院大学微电子学院

2021 年 6 月

Research on Convolutional Neural Network Accelerator

A thesis submitted to

University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Master of Engineering

In Integrated Circuit

By

Zekun Zhang

Supervisor: Prof. Shushan Qiao

School of Microelectronics

University of Chinese Academy of Sciences

June 2021

中国科学院大学

研究生学位论文原创性声明

本人郑重声明：所呈交的学位论文是本人在导师的指导下独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明或致谢。

作者签名：张泽鑫

日　　期：2021年5月18日

中国科学院大学

学位论文授权使用声明

本人完全了解并同意遵守中国科学院有关保存和使用学位论文的规定，即中国科学院有权保留送交学位论文的副本，允许该论文被查阅，可以按照学术研究公开原则和保护知识产权的原则公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名：张泽鑫　　导师签名：

日　　期：2021年5月18日.　　日　　期：2021年5月18日

摘 要

随着人工智能领域的蓬勃发展，随之而来的是超大数据量的运算需求对硬件的更新和提升提出了更大的挑战。神经网络在人工智能领域充当着十分重要的角色，也让人工智能进入了重要的发展阶段。卷积神经网路为神经网络的一种，随着目标识别，人脸识别等应用逐渐司空见惯，卷积神经网络真正进入了人们的视野。为了提高网络识别的准确度，网络变得越来越复杂，规模也愈来愈大，这随之而来的便是大幅提高计算成本以及存储需求，导致通用计算平台面临巨大的吞吐、能效挑战。至此，卷积神经网络加速器逐渐成为学术界和工业界的研究热点。

然而目前的设计仍存在问题。研究者通常通过提高频率或者扩大阵列单元来提升算力，这并不是最优的加速方案，比如阵列单元个数过多会导致时序很难收敛，应该从适配算力和带宽的方面入手做整体的架构优化。针对这类问题，本文提出一种具有可配置、高并行高吞吐、高能效比的机器视觉领域的神经网络加速器。本文的主要工作如下：

针对卷积层的大数据量问题，采用数据切分的方式，将卷积操作分为若干子运算，通过输入输出通道并行的计算方式，提高计算效率；另外，利用切分后的数据在通道方向做数据复用，减少数据的重复搬运；对于数据存储，采用顺序存储的方式，不仅可以提高 DMA 工作效率又可以提高阵列单元工作效率；针对池化层的滑窗运算需要用到较多的移位寄存器，采用先横向池化后纵向池化的算法，使得控制简单，效率大幅提高；针对阵列单元进行了整体的优化，在阵列内部采用权重寄存器堆缓存的方式来做乒乓操作，减少数据搬运时延。

经实验得出：以 VGG-16 为目标网络，基于 FPGA 在 100MHz 的时钟频率下，峰值算力 409.6GOPS，功耗 1.107W，可以达到 242.75GOPS 的吞吐量，219.29GOPS/W 的能效比。在 SMIC 55nm 的工艺下，时钟频率可以达到 300MHz，峰值算力约 1.2T，综合后的功耗为 1.7W，可达到约 722.8GOPS/W 的能效比。

关键词： FPGA，卷积神经网络加速器，卷积切分，可配置，高性能计算

Abstract

With the rapid development of artificial intelligence field, the following large amount of data computing requirements have posed greater challenges to the update and upgrading of hardware. Neural network plays a very important role in artificial intelligence, which also makes artificial intelligence enter an important stage of development. Convolutional neural network is a kind of neural network. With the application of target recognition and face recognition becoming more and more common, convolutional neural network has really entered people's vision. As networks grow in size to improve accuracy, computing costs and storage requirements will increase dramatically, leading to significant throughput and energy efficiency challenges for common computing platforms. So far, Convolutional Neural Network Accelerator has gradually become a research hotspot in academia and industry.

However, there are still problems with the current design. Researchers improve computing power by increasing frequency or expanding array units, which is not the optimal acceleration scheme. For example, too many array units will lead to difficult convergence of timing sequence. Therefore, overall architecture optimization should be carried out from the aspects of adaptive computing power and bandwidth. To solve this problem, this thesis proposes a neural network accelerator with configurable, high parallel, high throughput and high energy efficiency ratio in the field of machine vision. The main work of this thesis is as follows:

To solve the problem of large amount of data in the convolution layer, the convolution operation is divided into several sub-operations by means of data cutting, and the computing method of parallel input-output channels is adopted to improve the computational efficiency. In addition, the data after cutting is used for data reuse in the channel direction to reduce the repeated handling of data. The sequential data storage method can not only improve the working efficiency of DMA but also improve the working efficiency of array cell. The sliding window operation of the pooling layer needs to use more shift registers. The algorithm of horizontal pooling first and then vertical pooling is adopted, which makes the control simple and the efficiency greatly improved. The whole array unit is optimized, and the weight register heap cache is used to perform ping-pong operation inside the array to reduce the data transfer delay

This thesis presents a configurable convolutional neural network accelerator. The experimental results show that VGG-16 as the target network, based on FPGA, at the clock frequency of 100MHz, the peak computing power is 409.6GOPS, and the power consumption is 1.107W, can achieve a throughput of 242.75GOPS and an energy efficiency ratio of 219.29GOPS/W. Under the SMIC 55nm process, the clock frequency can reach 300MHz, the peak computing power is about 1.2T, the integrated power consumption is 1.7W, and the energy efficiency ratio reaches about 722.8GOPS/W

Key Words: FPGA, Convolutional neural network accelerator, Convolution segmentation, Configurable, High performance computing

目 录

第 1 章 绪论 1
1.1 研究背景与意义 1
1.1.1 卷积神经网络应用 1
1.1.2 卷积神经网络加速 4
1.1.3 基于 FPGA 的卷积神经网络加速器的应用前景 6
1.2 研究进展 7
1.3 The Roofline Mode:可视化性能模型 7
1.4 本文主要研究内容及章节安排 9
第 2 章 卷积神经网络和加速器概述 11
2.1 卷积神经网络原理 11
2.1.1 常规卷积层 12
2.1.2 深度卷积层 15
2.1.3 激活层 15
2.1.4 池化层 16
2.1.5 全连接层 17
2.2 卷积神经网络结构 18
2.2.1 LeNet 18
2.2.3 VGGNet 19
2.3 相关卷积神经网络加速器架构 20
2.4 本章小结 22
第 3 章 卷积神经网络算法模型优化 23
3.1 基本结构概述 23
3.2 卷积层算法优化 23
3.3 卷积运算单元 25
3.4 全连接层算法优化 27
3.5 其它运算 29
3.5.1 批归一化 29
3.5.2 Element-wise 运算 30
3.5.3 非线性运算方案优化 30
3.6 池化层运算优化 31
3.7 数据存储方式优化 32

3.7.1 片上缓存上的存储方式 33
3.7.2 外部存储器上的存储方式 35
3.8 数据量化 35
3.9 本章小结 36
第 4 章 卷积神经网络加速器硬件架构 39
4.1 加速器整体架构 39
4.2 卷积运算通路 40
4.3 池化运算通路 41
4.4 批归一化以及偏置运算 42
4.5 Element-wise 运算 42
4.6 非线性运算 43
4.7 时钟与复位系统设计 43
4.8 通信接口定义 44
4.9 本章小结 45
第 5 章 验证与性能分析 47
5.1 实验环境和验证平台 47
5.2 实验结果 48
5.3 LeNet 结果分析与对比 50
5.3.1 精度对比 50
5.3.2 结果分析 50
5.3.3 吞吐量及能量效率对比 51
5.4 VGG 结果分析与对比 51
5.4.1 结果分析 51
5.4.2 吞吐量以及能量效率对比 53
5.4.3 Roofline 模型分析 54
5.5 ASIC 综合及性能评估 55
5.6 本章小结 55
第 6 章 总结与展望 57
6.1 全文总结 57
6.2 未来工作 58
参考文献 59
附录 65
致谢 67

作者简历及攻读学位期间发表的学术论文与研究成果 69

图目录

图 1.1 FPGA 内部结构示意图 5
图 1.2 The Roofline 模型 8
图 2.1 典型卷积神经网络结构 11
图 2.2 卷积层运算过程 12
图 2.3 加 Pad 的卷积运算过程 13
图 2.4 常规卷积层 13
图 2.5 深度卷积层 15
图 2.6 激活函数 15
图 2.7 池化层 16
图 2.8 池化运算过程 17
图 2.9 LeNet 网络结构 18
图 2.10 VGG 网络结构 19
图 3.1 优化的卷积层算法数据分组 24
图 3.2 卷积操作在乘法阵列执行过程 24
图 3.3 乘法阵列单元整体结构 25
图 3.4 数据流水载入乘法阵列 26
图 3.5 加法阵列 27
图 3.6 全连接运算转为卷积运算 27
图 3.8 全连接运算映射到阵列的数据流水方式 28
图 3.9 指数函数的几种逼近拟合方法 30
图 3.10 传统的滑窗池化操作 31
图 3.11 池化算法优化 32
图 3.12 权重在片上缓存的存储方式 33
图 3.13 特征在片上缓存的存储方式 34
图 3.14 定点数据格式 36
图 4.1 加速器整体架构 39
图 4.2 卷积运算通路 40

图 4.3 池化运算通路.....41
图 4.4 批归一化及偏置运算模块.....42
图 4.5 非线性运算模块.....43
图 4.6 时钟与复位系统.....44
图 4.7 通信接口.....45
图 5.1 基于 XCKU115 FPGA 定制的加速卡.....47
图 5.2 验证平台示意图.....48
图 5.3 实际验证硬件平台.....48
图 5.4 FPGA 布局布线后的版图.....49
图 5.5 LeNet 各层的运行时间.....50
图 5.6 VGG-16 的 FPGA 识别结果.....52
图 5.7 VGG-16 各层的运行时间.....52
图 5.8 Roofline 模型对性能评估.....54

表目录

表 2.1 卷积层参数.....13
表 2.2 卷积层数据规模与数据量.....15
表 2.3 池化层参数.....17
表 2.4 池化层数据规模及数据量.....17
表 2.5 LeNet 每层参数.....18
表 2.6 VGG-16 各层参数.....19
表 5.1 加速器物理资源占用情况.....48
表 5.2 加速器各部分功耗.....49
表 5.3 LeNet 准确率对比.....50
表 5.4 LeNet 卷积各层 MAC 里利用率.....51
表 5.5 加速器运行 LeNet 与其它研究成果性能对比.....51
表 5.6 VGG-16 卷积各层 MAC 利用率.....53
表 5.7 加速器运行 VGG-16 与其它研究成果性能对比.....54
表 5.8 加速器在 SMIC 55nm 工艺下的性能参数.....55

第1章　绪论

1.1　研究背景与意义

随着人工智能领域的蓬勃发展，随之而来的是超大数据量的运算需求对硬件的更新和提升提出了更大的挑战。神经网络在人工智能领域充当着十分重要的角色，也让人工智能进入了重要的发展阶段。卷积神经网路为神经网络的一种，随着目标识别，人脸识别等应用逐渐司空见惯，卷积神经网络真正进入了人们的视野。

卷积神经网络最早于 1998 由 LeCun 等人提出，名为 LeNet[8]，起初用于识别邮件上的手写数字。LeNet 奠定了卷积神经网络的网络基本机构，为之后的网络结构发展提供了发展基础。LeNet 由多层过滤器组成，以最大程度的激活某种类型的视觉结构。在 LeNet 后，各种的优秀的网络结构层出不穷，为了提高网络识别的准确度，网络变得越来越复杂，规模也愈来愈大，这随之而来的便是大幅提高计算成本以及存储需求，导致通用计算平台面临巨大的吞吐、能效挑战。

卷积神经网络是通过反向传播来进行前向数据的修正，本质上，反向传播是导致网络学习的原因。这是一个反馈过程，网络产生的输出与它原本应该产生的输出进行比较修正。卷积神经网络目前主要基于 GPU、CPU。对于前向传播，每层之间独立，计算独立，也无数据反馈，而 CPU 基于执行指令的处理特性并不适合卷积神经网络计算的并行性。经过研究人员的探索，发现 FPGA 具有天生适合做神经网络加速的优势。

为了充分发挥卷积神经网络本身固有的并行特点，很多研究者采用 FPGA 来加速神经网络，本文将针对这些问题设计一种高并行流水的卷积神经网络加速器，以达到提高计算效率的目的，并和已有研究成果进行实验比较。

1.1.1　卷积神经网络应用

直到 20 世纪 80 年代，神经网络的研究逐渐进入人们的视野，当时，传统的冯·诺依曼体系结构主导了计算领域。1989 年，LeCun 等提出卷积神经网络 LeNet，这种网络拓扑结构是现代神经网络的原型。直到近些年一些研究引入了一种合理有效的无监督学习算法，才从经验上缓解了与深度模型相关的优化难题。近年来，

深度学习技术被广泛应用于越来越具有挑战性和前瞻性的应用中，其算法性能是传统技术从未达到的。

1. 在图像方面的应用

卷积神经网络用于识别图像的层次或概念结构。常见的计算机视觉任务有人脸识别、图像分类、目标检测、文本分析、姿态识别、行为识别等。下面将分类介绍。

(1) 图像分类

图像识别是深度学习的一个重要应用领域。ImageNet 是基准测试，被称为最具挑战性的图像识别任务之一。Grigory 等总结了传统计算机视觉模型精度与基于 ImageNet 的分类技术的比较。2015 年又提出 leaky-ReLU 激活函数以及 PReLU-net，成为第一个超越人类水平的卷积神经网络模型。与此同时，基于深度学习的图像识别的应用开始越来越多进入到人们的生活中。例如，谷歌公司宣布其公司旗下的软件产品“谷歌翻译”集成了基于卷积神经网络的实时翻译。

在图像分类方面，神经网络在 GPU 上的处理速度比对应的 CPU 快两个数量级。研究者提出这两种网络结构，并取得很高的性能：其一为 MCDNN (Multi-column deep neural networks)可以比以往所有的图像分类方法有更好的性能，而且经过验证，错误率减少 30-40%；其二为 HDCNN (Hierarchical Deep Convolutional neural Networks)是基于图像分类中某些类别比其他类别更容易混淆而提出的神经网络，它建立在传统 N 向分类器的基础上，遵循由粗到细的分类策略和设计原则。在 CIFAR100-NIN 构造块的 HD-CNN 模型的测试精度达到 65.33%。在 CIFAR100 数据集上，此结构高于其他标准深度模型和 HD-CNN 模型。

(2) 目标检测

目标检测同样为卷积神经网络的另一个应用领域，特别是在自动驾驶，公路摄像机中最为体现。然而，在实际应用中，导致物体变形的因素很多，如姿态和视角的变化，以及其他物体的遮挡等，都会使物体检测变得非常困难。其中最重要的是提高特征表达能力和抗变形能力以及提高分类器的准确率和速度。

近年来，工业界的无人驾驶技术发展十分迅速，大量的科技公司和起初公司都在致力于无人驾驶技术的研究，且取得了突破性进展。Google-X 实验室于 2015 年，其公司研发的无人驾驶车上路测试。

以深度学习技术为发展基础，自动驾驶算法逐渐被提出，比如直行、加速或者减速这些操作指令，都是通过神经系统从不同的传感器和摄像头采集原始输入数据，进行实时网络模型计算的。NVIDIA 的 Bojarski 等发布了基于卷积神经网络的自动驾驶，端到端学习解决方案，名为 NVIDIA DAVE-2 System。在这项工作中，他们使用 NVIDIA DevBox 和 Torch 7 进行训练，并使用 NVIDIA DRIVEPX 自动驾驶汽车电脑运行 Torch 7，得到以每秒 30 帧的速度进行数据运算的吞吐量，这被认为是实时视频处理 AI 算法性能的标准。学术界和工业界一直致力于自动驾驶汽车的应用，使这一领域成为最重要的深度学习领域成果之一。

2. 在语音方面的应用

卷积神经网络近年来在语音识别中得到了广泛的应用，并取得了比深度神经网络更好的识别效果。2015 年，微软公司的研究人员指出，卷积神经网络在噪声鲁棒性、远距离语音识别、低足迹模型以及通道不匹配训练测试条件的结果优于深度神经网络。

研究人员表示，在 1000 小时的 Kinect 远程训练中，卷积神经网络相对于同样大小的深度神经网络单词错误率减少 4%。研究者使用卷积神经网络和最大输出单元部署足迹模型到设备，比深度神经网络单词错误率减少 9.3%。卷积神经网络在语音识别中有一定的优势。在局部频率区域池化增强了卷积神经网络的鲁棒性，并通过使用较少的参数提取低阶特征避免了过拟合。

远程语音支持 SDM (Single remote Microphone)和 MDM (Multiple remote Microphone)两种方式。远距离语音识别中需要解决的主要问题是多声源和恒定混响。2014 年，Pawel Swietojanski 等人发现，在远距离语音识别中，使用卷积神经网络的单词错误减少率比深度神经网络提高了 6.5%，使用高斯混合模型的单词错误减少率比深度神经网络提高了 15.7%。在跨通道卷积的情况下，单词错误减少率比深度神经网络提高了 3.5%，比高斯混合模型提高了 9.7%。研究人员推断，SDM 中产生的单词错误减少率与通过 8 个麦克风进行波束形成训练的深度神经网络相当。在 MDM 中，最大通道池比多通道卷积效果更好。为了获得准确的语音识别结果，人们普遍采用递归神经网络进行远距离语音识别。卡耐基梅隆大学的 Suyoun Kim 等人在循环神经网络中嵌入了一种注意机制，以便在一个步骤中获得期望的输出。斯坦福大学的研究人员结合了循环神经网络和卷积神经网络的方法来获得更好的输出。

3. 在语言分析方面的应用

自然语言处理(NLP)是人工智能和语言学的一个跨学科领域深度学习技术的重要应用。自然语言处理可以进行语音识别、情感分析、问答、机器翻译等应用。许多基于深度卷积神经网络的自然语言处理技术已经成功实现商业化。例如，大多数主流的电脑和手机操作系统手机有自己的问答软件，比如亚马逊的 Alexa，微软的 Cortana，谷歌 Assistant 等。深度学习在自然语言处理应用中优于传统的基于特征的模型主要是因为深度神经网络不考虑声学特征，因此在设计时不那么费力模型适用于具有复杂和可变特征的真实数据。

Yoon Kim 的研究评估了卷积神经网络在 7 个数据集上的句子级分类任务。这项工作进一步证明了词向量的无监督预处理训练是自然语言处理深度学习的一个重要因素。Kalchbrenner 等人也提出了类似的模型，该模型提出一个称为 k-Max pooling 的全局池操作，使得该模型误差减少了 25%。Wang 等在该网络中加入了语义聚类，其结果验证了该模型的有效性。

1.1.2 卷积神经网络加速

深度学习算法是基于大量数据运算的，包含大量的计算操作以及数据搬运。典型的硬件平台如 CPU 很难很好的完成这种数据密集型操作，因此，人们在应用深度学习算法时通常使用其他硬件来加速。目前主流的硬件加速器有：GPU、ASIC、FPGA。在卷积神经网络硬件加速的发展历程中，学术界和工业界为了达到最优的效率，进行了多维度的研究。本节将一一做介绍。

1. GPU

GPU（Graphic Processing Unit）起初专门用于处理图像任务，其主要由控制器、寄存器和逻辑单元构成。与传统的 CPU 结构不同，GPU 的内部结构包含了大量的逻辑计算单元，计算单元与共享内存之间的数据传输速度远快于全局内存。此外，GPU 具有相对高速的全局内存和较大的内存带宽，例如 GDDR5。目前，一些深度学习框架(如 Caffe、TensorFlow 等)可以更好地应用于 GPU，英伟达等公司也为 GPU 提供了更好的深度学习环境。由于其强大的计算能力，目前主要用于处理大规模的计算任务。在卷积神经网络中，大多数计算都是矩阵的线性运算，以及实际大量数据运算，但控制逻辑简单。对于这些庞大的计算任务，GPU 的并行处理器表现出极大的优势。自从 AlexNet 在 2012 的 ImageNet 比赛中取得

优异成绩以来，GPU 被广泛应用于深层神经网络的训练和推理。大量依赖 GPU 运算的软件框架极大地降低了 GPU 的使用难度。

2. FPGA

FPGA(Field-Programmable Gate Array)现场可编程门阵列，是基于 CLB，包含查找数组(LUTs)的芯片，内部有固定功能块，信号路由和时钟网络，如图 1.1 所示。通过配置 LUTs 的内容以及相互连接固定功能块，任何数字电路都可以在 FPGA 中构建，并且可以根据需要更改。一般来说，用 FPGA 实现的电路总是比用 ASIC 实现的同类专用电路慢，且效率低。然而，对于可以利用固定功能块的应用程序来说，这种效率差距更小，而且现成的可重构芯片不会带有禁止的掩模成本或 ASIC 设计失败成本。FPGA 可以根据神经网络的计算过程实现高并行性和简化逻辑，并针对具体的模型进行硬件设计。因此，FPGA 可以实现比 CPU 和 GPU 更高的能量效率。图 1.1 摘自文献[75]。

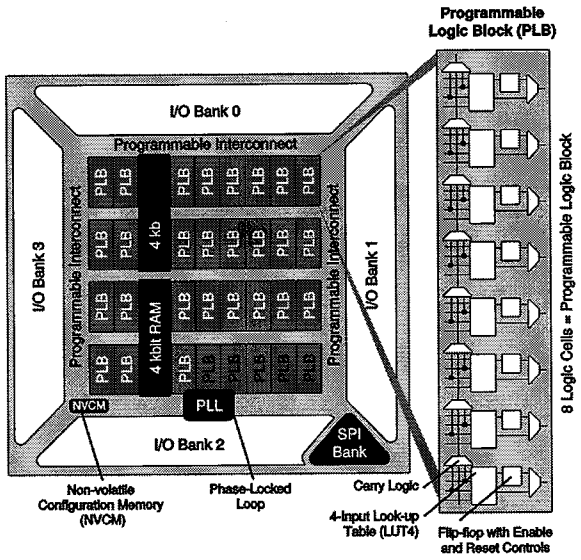

图 1.1　FPGA 内部结构示意图

Figure 1.1 The organization of a FPGA

由于其内部组件的重复和结构规则，FPGA 受益于半导体工艺技术的改进，如今的 FPGA 拥有大量的片上存储容量、带宽和计算能力。在过去的几年里，FPGA 作为一种新的异构计算受到了广泛的关注。

FPGA 的可编程特性使其有更快的开发时间。因此，经常被用来验证卷积神经网络训练和推理算法。例如，FPGA 已被用于实现低精度和二进制网络、新的训练机制，以及通过权重修剪的模型压缩。

3. ASIC

ASIC(Application Specific Integrated Circuit)专用集成电路，与 FPGA 一样，基于 ASIC 在构建卷积神经网络推理方面已经有了广泛的进展。一般来说，这些方法主要在预先训练好的卷积神经网络，如 LeNet-5、AlexNet 和 VGG。这些系统主要考虑因素之一是尽量减少存储和获取卷积神经网络权重、输入样本以及减少中间层结果与 DDR 访问次数，因为这会消耗大量的功耗。在合理的内存假设下，从 DDR 加载 32 位浮点数比浮点乘法操作要多消耗三个数量级的功耗。因此，这类设计的主要关注点之一是映射卷积神经网络的计算，使得在所有在层内使用的数据只获取一次输入和权值。

不同于深度学习算法适应 CPU 或 GPU 硬件结构实现加速的方式，使用 ASIC 加速深度学习算法的主要方式是定制专用的硬件加速算法。由于专用电路是专门为加速某一种或某一类算法而设计的，因此加速效果通常较好，功耗较低，但同时可重构性较差，开发成本较高，硬件设计和开发周期更长。

最小化数据移动的一个有效方法是层融合，即其中多个卷积神经网络层以流水线的方式同时处理，不需要外部内存访问，可以直接使用中间数据。我们将这种类型的方法称为跨层流水线。假设有足够的硬件资源以流水线的方式跨几个层执行计算，跨层流水线可以显著减少推理的总运行时间。一般来说，压缩架构可以高效地执行卷积神经网络，因为它们被设计为复用相同的数据元素，这样可以减少从内存读取数据的次数。

1.1.3 基于 FPGA 的卷积神经网络加速器的应用前景

与 CPU、GPU 神经网络加速结构相比，FPGA 更接近底层，通常具有更好的能量效率，更适合算法的推理阶段。而 ASIC 加速性能优势在性能方面，基于 FPGA 的加速提供了更大的灵活性、更低的开发成本和更少的开发周期。而且，目前基于 FPGA 的开发环境越来越轻量高效，高级语言 C 和 Python 语言给开发过程带来了极大的便利。FPGA 在卷积神经网络加速方面具有可重构、可定制和节能的特点，已成为国内外研究机构的研究热点与应用方向。FPGA 最初并不是为神经网络而开发的，而是为快速发展的电子硬件原型而开发的。神经网络自诞生以来一直在探索新的改进和应用，但尚未形成统一的研究方向。虽然在 1994 年 D.S. Reay 首次使用 FPGA 来加速神经网络，但由于神经网络本身的发展，并没有引起人们的注意，直到 ILSVRC 2012 中 AlexNet 的诞生。研究界正在向更

深入、更复杂的网络研究方向发展，如后期的 CNN、RNN、DNN 等。随之而来的 VGGNet、GoogleNet、ResNet 等优秀的网络结构层出不穷，这充分标志着复杂神经网络的发展趋势。

1.2 研究进展

随着人工智能领域的蓬勃发展，随之而来的是超大数据量的运算需求对硬件的更新和提升提出了更大的挑战。神经网络在人工智能领域充当着十分重要的角色，也让人工智能进入了重要的发展阶段。卷积神经网路为神经网络的一种，随着目标识别，人脸识别等应用已经习以为常，卷积神经网络真正进入了人们的视野。大多数基于 FPGA 的卷积神经网络加速器的研究都是通过优化体系结构或优化体系结构中的一个或几个步骤的速度来提高加速器的速度。

研究者在追求更好运算效率的同时，神经网络模型层数和参数数量也在不断的增加，从而对硬件的计算能力、内存带宽及数据存储等提出越来越高的要求。

Chen 等(2016)[39]提出了多节点神经网络加速器 DaDianNao，同时也验证了多芯片架构的可行性，其使用 64 个节点。在 28nm 处实现了节点到节点和路由，包含自定义存储和计算单元的组合。该加速器比单个 GPU 的性能提高达 450.65 倍，功耗降低达 150.31 倍。在 28nm 工艺节点下，每个节点的面积为 67.73mm^2。

因此，在计算能力强、可并行加速、数据吞吐高的高性能硬件平台对模型训练来说显得尤为重要。面对复杂的实际应用需求和不断加深的神经网络结构，多样化的深度神经网络硬件平台不断发展，逐步形成了以通用性芯片(CPU、GPU)、FPGA、ASIC、SoC 等为主的硬件平台。

1.3 The Roofline Mode:可视化性能模型

The Rooline[67]是使性能可视化的模型，它展示了运行在特定计算机体系结构上的特定计算内核的性能上限。Roofline 提出一种使用 Operational Intensity（计算强度）进行性能分析的方法。计算强度就是从存储器中取每个字节所完成的计算操作数，即计算量与从片外存储器读取的每字节的比率，运算量与访存量的比值。如图 1.2

“如果我们把计算强度想象成一根柱子撞到屋顶，它要么撞到屋顶平坦的部分，这说明性能受到算力的限制；要么撞到屋顶的倾斜部分，意味着性能最终受到内存带宽的限制。”

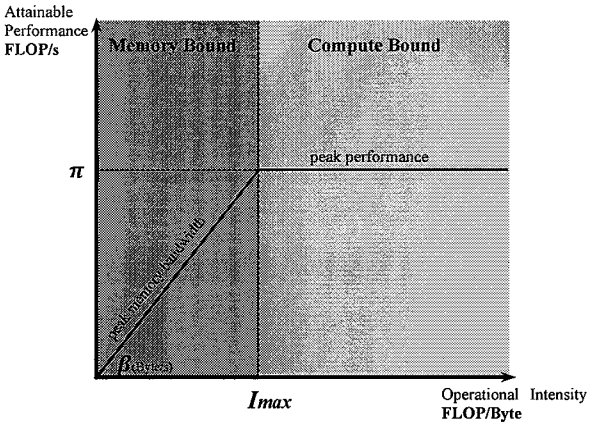

图 1.2 The Roofline 模型

Figure 1.2 The Roofline Model

计算强度上限为峰值算力与最大带宽的比值，描述在此计算平台上，单位内存交换最多用来计算的次数。

$$I_{max} = \frac{\pi}{\beta} \qquad \ldots(1.1)$$

The Rooline 模型主要考虑两个性能边界：计算能力和访存带宽，并依据计算强度，确定其中哪个将成为系统瓶颈，依此评估系统性能可以达到的峰值性能。由于 Rooline 模型可以在抽象的层次上去考虑系统平台与内核特性，所以也可以很好的评估加速器的性能。

1. 峰值算力(π)：计算能力上限，即硬件平台每秒钟所能完成的浮点运算的最大值。也就是屋顶的平坦部分。
2. 最大访存带宽(β)：访存带宽上限，即硬件平台每秒所能完成的内存存取的最大值。也就是屋顶的倾斜部分。

通过 Rooline 模型可以估算出系统或者模型可达到的理论性能：

$$Attainable\ Performance = min(\frac{\text{访存量}}{\text{访存带宽}}, \frac{\text{运算量}}{\text{系统算力}})$$

计算强度将 Roofline 模型划分出两个区域：其一为因访存带宽所限制的区域 Memory Bound，即访存带宽为系统瓶颈；另一个为因算力限制的区域 Compute Bound，即算力为系统瓶颈。

$$P=\begin{cases}\beta\cdot I, & when\ I<I_{max} \quad 带宽边界\\ \pi, & when\ I\geqslant I_{max} \quad 计算边界\end{cases} \qquad ...(1.2)$$

1.4　本文主要研究内容及章节安排

本论文主要研究内容和整体结构如下：

第一章为绪论。介绍了神经网络的发展历程，接着介绍了卷积神经网络所带动各领域的发展及在各领域发挥的作用。然后简述了硬件平台的出现促进着神经网络的进一步发展，以及各硬件平台的优缺点以及差异。接着分析了基于 FPGA 的加速器的应用前景。最后着重介绍了性能评估模型 The Roofline Model。

第二章为卷积神经网络的相关背景知识。首先介绍了卷积神经网络的基本组成，然后对卷积神经网络的识别原理进行简要阐述，接着介绍了每个种类层在卷积神经网络中的发挥的作用以及其运算方式。然后详细介绍了两种经典卷积神经网络的背景及其整体结构。最后介绍了本领域的研究者所取得的研究成果及相关工作。

第三章为对传统的卷积神经网络算法模型优化。首先介绍了本方案的基本结构。接着对各类型层的算法进行优化，并比较优化后的性能。针对卷积层运算的大数据量问题，采用卷积切分的方式对卷积层数据进行分组进行运算；针对池化层滑窗需要用到很多的移位寄存器，采用先进行横行池化后进行纵向池化的优化算法。针对非线性函数中的超越函数不易用硬件实现，采用二次逼近的方法拟合，并达到很高的精度。之后介绍了本文所采取的数据存储方案。最后介绍了本文采取的量化方案。

第四章为卷积神经网络加速器的总体硬件架构。我们逐一的描述每个模块的硬件设计思路、实现方案，以及每个通路的硬件结构。接着从时钟系统设计、复位系统设计、对外通信接口定义以及内部模块的通信接口定义分别对本加速器进行更完善的介绍。

第五章为本设计的验证和性能分析。首先对 FPGA 平台的选型和硬件平台的搭建。然后用 LeNet 与 VGGnet 作为目标网络，对本文所设计的加速器进行验证。从资源占用、功耗分析、精度对比等方面分析结果。并对吞吐量、能效比等方面和目前的研究成果对比，对加速器进行评估。同时也根据 Roofline 模型对加速器

做性能评估。最后对本文所设计的加速器进行 ASIC 综合，大致评估其在 ASIC 下的性能以及能效比。

第六章对本文的研究工作进行总结并对后续工作进行了展望。

第2章　卷积神经网络和加速器概述

本章首先介绍了卷积神经网络的基本组成，然后对卷积神经网络的识别原理进行简要阐述，接着介绍了每个种类层在卷积神经网络中的发挥的作用以及其运算方式。然后详细介绍了两种经典卷积神经网络的背景及其整体结构。最后介绍了本领域的研究者所取得的研究成果及相关工作。

2.1　卷积神经网络原理

卷积神经网络通常由卷积层、池化层和全连接层以及非线性激活层构成，如图 2.1。卷积神经网络的计算过程可以表示为公式 2.1。在卷积层，$\boldsymbol{W}$ 为卷积核，$\boldsymbol{I}$ 为输入特征，$\boldsymbol{B}$ 为偏置，f 为非线性激活，$\boldsymbol{O}$ 为输出特征图

$$\boldsymbol{O}[m][r][c]=f\{\sum_{n=0}^{N-1}\sum_{i=0}^{k-1}\sum_{j=0}^{k-1}\boldsymbol{I}[n][r\cdot s+i][c\cdot s+j]\times\boldsymbol{W}[m][i][j]+\boldsymbol{B}[m]\} \quad ...(2.1)$$

其中，n 为输入通道数、m 为输出通道数、r 为输出特征图行数、c 为输出特征图列数、k 为卷积核尺寸、s 为跨步长[27]。

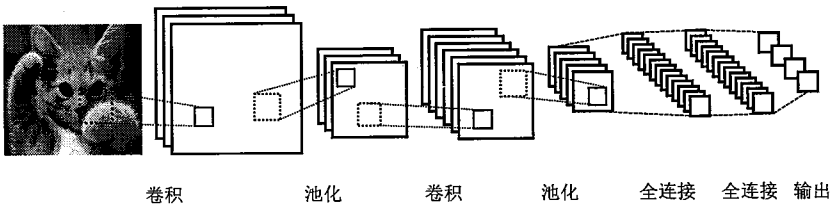

图 2.1　典型卷积神经网络结构

Figure 2.1 Typical layer structure in a CNN

卷积层对输入数据进行点乘运算，用来提取图像特征，然后通常会将卷积结果进行非线性操作，即经过激活函数后得到输出结果值。常用的激活函数由 ReLU、Sigmoid、Tanh 等。

子抽样层即池化层与卷积层相连，池化是对卷积卷积计算完的特征图依据特征不变性进行压缩，可以减少后面层的计算量。

全连接层一般在网络的最后几层，将这层的每一个神经元与前面层的神经元全部连接[25]。

2.1.1 常规卷积层

卷积运算是卷积神经网络中最重要的计算之一。输入图像可以看作是三维的像素值的矩阵，分别为高度(*H*)、宽度(*W*)和通道(*CH*)。卷积运算的过程为首先从输入特征取出与卷积核尺寸相同的滑动窗口，然后与卷积核做点乘加运算，得到的结果即为此输出特征的一个输出神经元，即产生一个映射。然后用同样的方式进行下一次卷积，得到另一个结果，依次顺序，直至此输入特征全部与卷积核做运算即为此次卷积层计算结束。当由多个卷积核生成的多个这样的特征映射堆叠时，便形成了此卷积层的输出。例如，假设有一个 7×7 的图像，还有另外一个 3×3 矩阵，其卷积计算如图 2.2 所示:

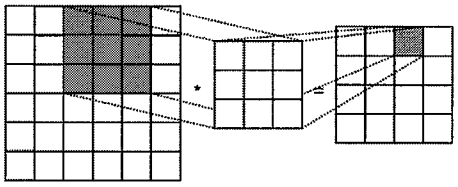

图 2.2 卷积层运算过程

Figure 2.2 Convolutional layer calculation process

目前可用的卷积神经网络拓扑结构中使用的内核的大小可以是不同的。典型的尺寸是 1×1、3×3、5×5 和 7×7。使用特定维度的规则依赖于需要捕获的特征的相对大小:要提取的特征的大小越小，过滤器就越小。

跨步长为卷积层中的一个参数，它指的是卷积核沿着输入图像滑动的长度。通常，在执行卷积操作时，滤波器按行或列移动一个像素，并输出一个结果，这个过程一直进行，直到完成输出特征。跨步是指这个位移的长度。也可以使用大于一个像素的步长或非统一步幅，以此减少激活量和计算工作量的。例如，步长为 2 时产生一个输入尺寸一半的输出特征。Kong 和 Lucey 对使用不同步长进行了有趣的讨论，他们证明了任何非统一步长的卷积都可以被统一步长的卷积所替代，而不会损失性能。

另一个参数是 Pad，即特征填充，以保持输出特征的尺寸与输入特征的大小相同。通常使用零填充，即用零填充特征的边界。如果不应用填充，卷积结果的空间维度(高度和宽度)将在每次卷积后逐步降低，就会迅速丢失包含在边界中的所有信息数据。下面的例子可以看到这种效果：一个 5×5 矩阵与一个 3×3 核进行卷积。如果没有填充，结果是一个 3×3 矩阵。但是，如果原始的 5×5 矩阵的所有

边界都被 0 填充，结果将是另一个 5×5 矩阵，此时尺寸保持相同的大小，通过再次填充这个矩阵，可以执行任意多的 3×3 卷积。如图 2.3。

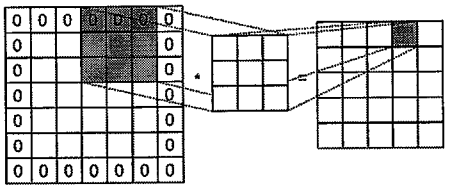

图 2.3　加 Pad 的卷积运算过程

Figure 2.3 Convolutional layer calculation process with pad

一般来说，一个卷积层可以用表 2.1 参数描述：

表 2.1　卷积层参数

Table 2.1 Convolutional layer parameter

参数名称	描述
CH_{in}/CH_{out}	输入/输出图像的通道数
W_{in}/W_{out}	输入/输出图像的宽度
H_{in}/H_{out}	输入/输出图像的高度
K_x	卷积核的宽度
K_y	卷积核的高度
S_x	卷积核滑动时宽度方向上的步长
S_y	卷积核滑动时高度方向上的步长
P_{up}	对输入图像上方填充 0 的行数
P_{down}	对输入图像下方填充 0 的行数
P_{left}	对输入图像左边填充 0 的列数
P_{right}	对输入图像右边填充 0 的列数

如图 2.4 所示，从示意图中可以看到左边为输入特征，其尺寸为 $CH_{in}\times H_{in}\times W_{in}$，中间为卷积核，尺寸 $CH_{out}\times CH_{in}\times K_x\times K_y$。右边为输出特征，尺寸 $CH_{out}\times H_{out}\times W_{out}$。输入特征通道和卷积核通道一样，均为 CH_{in}。CH_{out} 个卷积核，对应输出特征通道 CH_{out}，即有多少个卷积核，就有多少输出特征。

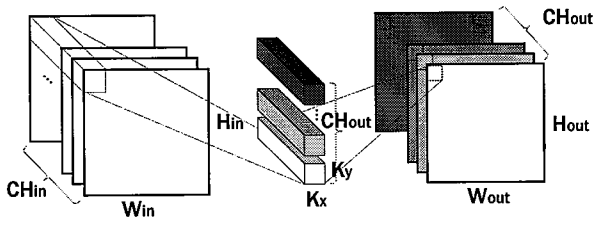

图 2.4　常规卷积层

Figure 2.4 Standard convolution layer

卷积层的计算过程就是首先从输入特征取 $CH_{in}\times K_x\times K_y$ 尺寸大小的窗口，然后与 $CH_{in}\times K_x\times K_y$ 卷积核做点乘加运算，便得到一个输出结果，即完成依次卷积映射。然后与卷积核通道的另一个卷积核继续做卷积运算，即可得到下个通道的输出结果，依次循环，直到 CH_{out} 个卷积核全部与特征窗口进行卷积操作，便可得到 $1\times1\times CH_{out}$ 个输出。然后将特征窗口按照从左往右、从上向下依次移动固定步长的节奏，行列依次进行，按照上面所述的运算方式一直到整个输入特征图与所有的卷积核进行卷积操作，意味着一层特征的卷积计算结束。接着进行另一层特征的运算，运算方式同理，直至所有特征运算完成，即卷积层的操作结束。卷积层的计算过程由算法 1 的 6 层循环嵌套完成。

算法 1 Convolution Layer

Require: feature map R_{in} of size $H_{in}\times W_{in}\times CH_{in}$;convolution kernel K of size $CH_{out}\times K_x\times K_y\times CH_{in}$

Ensure: feature map R_{out} of size $(H_{in}-K+1)\times(W_{in}-K+1)\times CH_{out}$

```
1:  procedure ConvolutionLayer
2:      for y ← 0, H_in − K + 1 do                          ▷Feature map height loop
3:          for x ← 0, W_in − K + 1 do                      ▷Feature map width loop
4:              for n ← 0, CH_out do                        ▷Output channel loop
5:                  for m ← 0, CH_in do                     ▷Iutput channel loop
6:                      for k_y ← 0, K_y do                 ▷Kernel height loop
7:                          for k_x ← 0, K_x do             ▷Kernel width loop
8:                              R_out[y][x][n] += F_in[y + k_y][x + k_x][m]×K[n][k_y][k_x][m]
9:                          end for
10:                     end for
11:                 end for
12:             end for
13:         end for
14:     end for
15:     return R_out
16: end procedure
```

算法 1 为一种典型的循环次序[3]。

在卷积运算中，一般用三维数组表示输入输出特征图像，用四维数组表示卷积核，如表 2.2：

表 2.2　卷积层数据规模与数据量

Table 2.2 Data size and volume of convolutional layer

	数据规模	数据量
输入特征	$[CH_{in}][H_{in}][W_{in}]$	$CH_{in}\times H_{in}\times W_{in}$
权重	$[CH_{out}][CH_{in}][K_y][K_x]$	$CH_{out}\times CH_{in}\times K_y\times K_x$
输出特征	$[CH_{out}][H_{out}][W_{out}]$	$CH_{out}\times H_{out}\times W_{out}$

2.1.2　深度卷积层

深度卷积层和常规卷积层的不同点在于深度卷积层的卷积方式：输入特征只与与之相对应的卷积核进行点乘加运算。如图 2.5，输入特征与卷积核的通道数均为，每个特征对应一个卷积核，并只与其进行卷积。

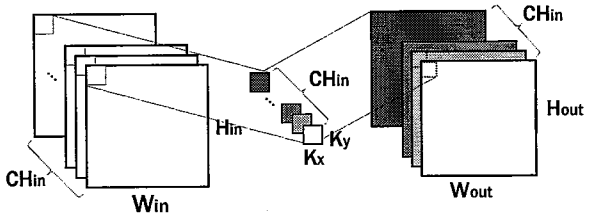

图 2.5　深度卷积层

Figure 2.5 Depthwise convolution layer

从示意图可以看到，左边为输入特征，尺寸大小为 $CH_{in}\times H_{in}\times W_{in}$，中间为卷积核，尺寸为 $CH_{in}\times K_x\times K_y\times 1$，右边为输出特征，尺寸为 $CH_{in}\times H_{out}\times W_{out}$。由于深度卷积为对应做卷积，所以输入特征通道、卷积核通道以及输出特征通道均相等，一一对应，均为 CH_{in}。计算过程为输入特征与卷积核对应做点乘，所以输出的通道数依然也为 CH_{in}。

2.1.3　激活层

激活层一般在卷积层后，卷积运算后会经过非线性操作，因为卷积神经网络的数据多数是非线性的，而卷积运算本身是通过数据的运算得来的，是线性操作，所以就需要做非线性处理。图 2.6 为目前常用的激活函数，摘自文献[65]，包括 Sigmoid、Tanh 和 ReLU 的各种变体。

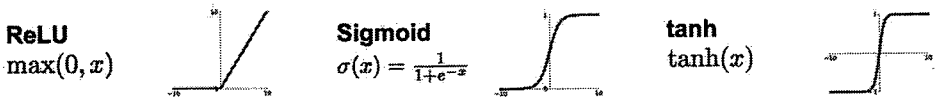

图 2.6　激活函数

Figure 2.6 Non-linear functions

Sigmoid 函数具有特征性的 S 形，并且当输入达到一定水平时，这些功能的输出趋于饱和。Sigmoid 将输入压缩或映射为(−1,+1)或(0,+1)。由此，神经网络可以建模为决策树，提供基于其输入值的不同输出。但是由于其计算复杂性，且被研究者发现很难训练，逐渐被 ReLU 函数所取代。对于输入的线性组合，ReLU 的输出对于负值将为零，对于正值，就是其本身。

ReLU 函数是一个简单，优雅，并且非常容易在硬件中实现。本质上，Sigmoid 以及 ReLU，均是在通过模仿生物神经元的行为输入信号从非活动状态转变为活动状态达到一定的阈值水平。ReLU 的效果要好得多，因为它可以更快地训练网络而没有在精度上有显著差异。

关于激活功能，偏置项的目的是转移激活输出，以便在预测和输入数据。 在神经网络训练阶段，每次迭代均会优化偏差和权重，从而实现更好地拟合并为神经元提供学习能力。权重和偏置也称为的可训练参数网络。

2.1.4 池化层

池化层是卷积神经网络中重要层之一，一般在卷积层之后。池化层的目的其一是减少特征体的空间大小（高度以及宽度），减少后面层的计算量；其二是可以控制过拟合。除了最大池化之外，池化层还可以执行其他函数，有最大池化、最小池化和平均池化。图 2.8 为典型的最大池化和平均池化，它的过滤器大小为 2×2，步长为 2。在这种情况下，最大池化操作将获取由过滤器分隔字段上 4 个元素的最大值。下面介绍最大值池化。

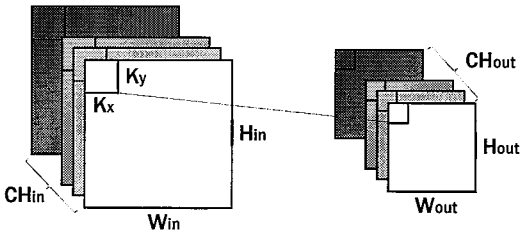

图 2.7 池化层

Figure 2.7 Pooling layer

如图 2.7 所示，左边为池化层的输入特征，尺寸大小为 $CH_{in}\times H_{in}\times W_{in}$，滑动窗口的尺寸为 $K_x\times K_y$，右边为输出特征，尺寸大小为 $CH_{out}\times H_{out}\times W_{out}$。在池化运算中，输入特征通道 CH_{in} 和输出特征通道 CH_{out} 相等。

池化操作的计算过程为从输入特征中取出 $K_x\times K_y$ 尺寸大小的窗口，比较出这个窗口中的最大值，这个最大值就是池化结果，然后去比较下一个通道，得到另

一个输出神经元，依次循环直至所有通道结束，共 $1\times1\times CH_{in}$ 个输出结果。然后从左往右、从上向下滑动固定的步长，行列依次进行，直到完整比较出整个输入特征，此时池化计算结束。

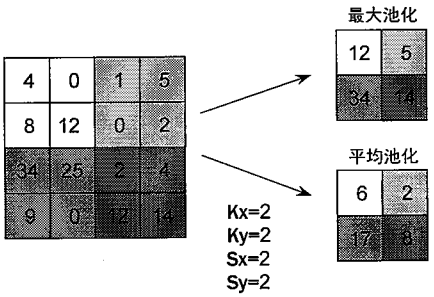

图 2.8　池化运算过程

Figure 2.8 Pooling calculation process

池化层可以用表 2.3 参数来描述：

表 2.3　池化层参数

Table 2.3 Pool layer parameters

参数名称	描述
CH_{in}/CH_{out}	输入/输出特征的通道数
H_{in}/H_{out}	输入/输出特征的高度
W_{in}/W_{out}	输入/输出特征的宽度
K_x	池化模板的宽度
K_y	池化模板的高度
S_x	池化模板滑动时宽度方向的步长
S_y	池化模板滑动时高度方向的步长

因此，在池化层运算中，通常用三维数组表示输入与输出特征：

表 2.4　池化层数据规模及数据量

Table 2.4 Size and volume of data in the pool layer

	数据规模	数据量
输入特征	$[CH_{in}][H_{in}][W_{in}]$	$CH_{in}\times H_{in}\times W_{in}$
输出特征	$[CH_{out}][H_{out}][W_{out}]$	$CH_{out}\times H_{out}\times W_{out}$

2.1.5　全连接层

在典型的卷积神经网络中，全连接层通常出现在架构的末端，

全连接层的操作可以表示为简单的矩阵乘法，加上偏置向量，然后执行逐元素方式的非线性函数$f(x)$:

$$Y = f(W * x + b) \qquad \ldots(2.2)$$

其中，x 为输入向量，y 输出激活向量，W 表示层间各单元之间连接的权重的矩阵，b 表示偏置项向量。

2.2 卷积神经网络结构

ILSVRC 2012 中 AlexNet 的诞生，标志着神经网络的发展方向逐渐被明确，研究界向更深入、更复杂的神经网络研究方向发展，如后期的卷积神经网络、循环神经网络以及深度神经网络等。VGGNet、GoogleNet、ResNet 等优秀的网络模型层出不穷，这也充分标志着复杂神经网络的发展趋势。下面以 LeNet 和 VGGNet 为代表介绍卷积神经网络。

2.2.1 LeNet

LeNet(LeCun 等，1998)[8]是卷积神经网络最早和最基本的形式之一。由于总共包含 5 个权重层，也被称为 LeNet-5。如图 2.9，图 2.9 摘自文献[8]。

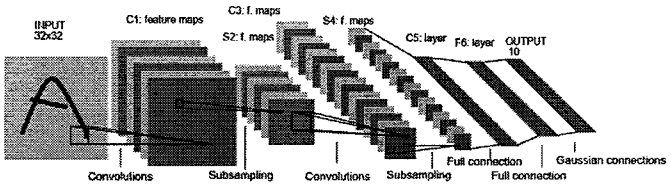

图 2.9 LeNet 网络结构

Figure 2.9 Network structure of LeNet

L eNet 的网络结构由一组卷积池化层和两个全连接层组成，LeNet 因为网络本身易收敛，所以有各种各样的变体，但基本都是由卷积池化以及全连接层构成。通过卷积层对输入进行特征提取，池化层通过特征不变性进行特征缩小，然后经过全连接层进行分类输出，LeNet 每层的参数见表 2.5.

表 2.5 LeNet 每层参数

Table 2.5 LeNet parameters per layer

层类型	输入尺寸	卷积核尺寸	填充(P)	步长(S)	运算量
CONV	28×28×1	5×5×1×32	2	1	0.6272
POOL	28×28×32	2×2	0	2	—
CONV	14×14×32	5×5×32×64	2	1	10.0352
POOL	14×14×64	2×2	0	2	—
CONV	7×7×64	5×5×64×64	12	2	20.0704
FC	14×14×64	[14×14×64]×[1024]	—	—	12.845
FC	1×1×1024	[1×1×1024]×[10]	—	—	0.01024
总计	0.061M	13.01M	—	—	0.087GOP

表注：输入尺寸对应$[H_{in}][W_{in}][CH_{in}]$，卷积核尺寸对应 $[K_x][K_y][CH_{in}][CH_{out}]$

2.2.3　VGGNet

VGGNet[73]于 2014 年由牛津大学计算机视觉组和 Google DeepMind 公司的研究员研究出来，并取得了当年比赛定位项目的第一名。图 2.10 为摘自文献[65]的 VGG 网络结构。

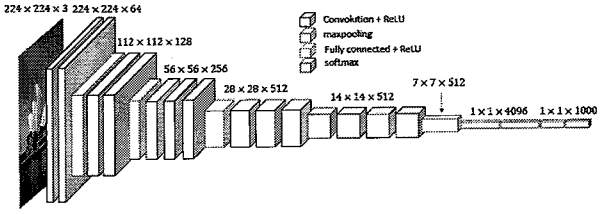

图 2.10　VGG 网络结构

Figure 2.10 Network structure of VGG

VGGNet 的网络结构很规整，在卷积层严格使用 3×3 尺寸的卷积核，池化层一律采用最大池化，在网络末端由一组三个全连接层用于分类，VGG 网络的每个卷积层后面跟 ReLU 激活层。由于使用较小的卷积核，相当于进行了更多的非线性映射，可以有效的训练和测试。表 2.6 为 VGG-16 的各层参数：

表 2.6　VGG-16 各层参数

Table 2.6 VGG-16 parameters of each layer

层类型	输入尺寸	卷积核尺寸	填充（P）	步长（S）	运算量(M)
CONV	224×224×3	3×3×64	1	1	86.704
ReLU	224×224×64	—	—	—	—
CONV	224×224×64	3×3×64	1	1	1849.688
ReLU	224×224×64	—	—	—	—
POOL	224×224×64	2×2	0	2	—
CONV	112×112×64	3×3×128	1	1	924.844
ReLU	112×112×128				—
CONV	112×112×128	3×3×128	1	1	1849.688
ReLU	112×112×128	—	—	—	—
POOL	112×112×128	2×2	0	2	—
CONV	56×56×128	3×3×256	1	1	924.844
ReLU	56×56×256	—	—	—	—
CONV	56×56×256	3×3×256	1	1	1849.688
ReLU	56×56×256	—	—	—	—
CONV	56×56×256	3×3×256	1	1	1849.688
ReLU	56×56×256	—	—	—	—
POOL	56×56×256	2×2	0	2	—
CONV	28×28×256	3×3×512	1	1	924.844

续表 VGG-16 各层参数

ReLU	28×28×512	—	—	—	—
CONV	28×28×512	3×3×512	1	1	1849.688
ReLU	28×28×512	—	—	—	—
CONV	28×28×512	3×3×512	1	1	1849.688
ReLU	28×28×512	—	—	—	—
POOL	28×28×512	2×2	0	2	—
CONV	14×14×512	3×3×512	1	1	462.422
ReLU	14×14×512	—	—	—	—
CONV	14×14×512	3×3×512	1	1	462.422
ReLU	14×14×512	—	—	—	—
CONV	14×14×512	3×3×512	1	1	462.422
ReLU	14×14×512	—	—	—	—
POOL	14×14×512	2×2	0	2	—
FC	7×7×512	[7×7×512]×[4096]	—	—	102.76
ReLU	1×1×4096	—	—	—	—
FC	1×1×4096	[1×1×4096]×[4096]	—	—	16.7772
ReLU	1×1×4096	—	—	—	—
FC	1×1×4096	[1×1×4096]×[1000]	—	—	4.096
总计	15.24M	138.34M	—	—	30.95GOP

表注：输入尺寸对应$[H_{in}][W_{in}]$ $[CH_{in}]$，卷积核尺寸对应 $[K_x][K_y][CH_{in}][CH_{out}]$

2.3 相关卷积神经网络加速器架构

大多数基于 FPGA 的卷积神经网络加速器的研究都是通过优化体系结构或优化体系结构中的一个或几个步骤的速度来提高加速器的速度。接下来将介绍若干基于 FPGA 或者 ASIC 的卷积神经网络加速器的相关研究工作。

Chen 等(2014)[39]提出了多节点神经网络加速器 DaDianNao，同时也验证了多芯片架构的可行性，其使用 64 个节点。在 28nm 处实现了节点到节点和路由，包含自定义存储和计算单元的组合。该加速器比单个 GPU 的性能提高达 450.65 倍，功耗降低达 150.31 倍。在 28nm 节点下，每个节点的面积为 67.73mm^2。

Chen 等(2016)[30]提出了加速器 Eyeriss，Eyeriss 通过配置，优化了整个加速器芯片和 DRAM，以适应各种卷积神经网络。Eyeriss 设计 168 个处理单元的阵列架构，使用称为 RS 的处理数据流完成数据搬运以及运算。该工作在测试 AlexNet 网络时，帧率达到 35 帧/秒 278 mW 的功耗，在跑 VGG16 时，功耗为为 236 mW，帧率 0.7 帧/秒。

Guo 等(2017)[6]提出了卷积神经网络的加速器 Angel-Eye，该工作同时考虑了数据量化策略和编译工具。数据量化策略有助于将位宽减小到 8 位，而精度损失可以忽略不计。编译工具将特定的卷积神经网络模型有效地映射到硬件上。该工作在 Zynq XC7Z045 平台上进行评估，比同平台上 FPGA 实现快 6 倍，功耗方面低 5 倍。使用 NIVIDA TK1 和 TX1 平台进行比较。Angel-Eye 实现了类似的性能，并提供多达 16 更好的能效比。

Jouppi 等(2017)提出了 TPU(Tensor Processing Unit)[21]，为谷歌公司的研究者提出的基于 ASIC 的神经网络加速器，对神经网络的推理阶段加速。TPU 是一款应用于 PC 领域的加速卡，其采用对指令译码的方式进行运算配置，由此可使得控制逻辑非常简单。其采用脉动阵列的架构做核心运算阵列单元，由 256×256 大小的阵列以及激活池化等模块完成推理阶段的所有运算。TPU 在访存带宽上也做了许多优化，比如对于权重的缓冲以及输入数据和输出数据采用同一缓冲区间。该工作最终可以达到 92 TOPS 的峰值吞吐量，并提供了 28MiB 软件管理片上内存。

Zhang 和 Prasanna (2017)[43]提出该工作在 Intel QuickAssist QPI FPGA 平台上实现了 VGG16、AlexNet 和 GoogLeNet。与 AlexNet 实现相比，该工作使用更少的 3.33 倍倍增器和更少的 1.1 倍内存实现了 1.35 倍 GFLOPs/sec 的改进。与 VGG16 实现相比，在不影响分类准确性的情况下使用 3.48 倍的倍增器，具有 0.66 倍 GFLOPs/sec。对于 GoogLeNet 实现，实现了 5.56 倍的性能，相比于 16 个线程运行在 10 核 Intel Xeon 处理器 2.8 GHz。

Zhao 等(2017)[44]首次提出了针对二值卷积神经网络设计的专用加速器。二值化神经网络的主要计算是位逻辑操作，并且减少了内存需求。经试验得出，以 Zynq XC7Z020 为运行平台，算力可达到 207.8GOPS。

Yang 等(2019)[45]提出了 Synetgy 卷积神经网络加速器。该工作开发 Synetgy 卷积神经网络加速器和 DiracDeltaNet 的新型卷积神经网络模型。加速器和 ConvNet 是根据 FPGA 要求定制的。DiracDeltaNet 在 ImageNet 上实现了很高的准确性。该工作量化 DiracDeltaNet 的权值为 1 位，激活为 4 位，精度损失小于 1%。这些量化很好地利用了 FPGA 硬件。DiracDeltaNet 的小模型尺寸、低计算 OP 计数、超低精度和简化操作，为 FPGA 共同设计高度定制的计算单元。该工作在 Ultra96 SoC 系统上通过高级综合实现 DiracDeltaNet 的计算单元。加速器最

终在 ImageNet 上排名前 5 的精度为 88.2%。此外，该加速器在 ImageNet 分类任务上达到了 96.5FPS 的推理速度，在类似的精度上至少超过了以前的工作 16.9 倍。

2.4 本章小结

本章主要介绍卷积神经网络的网络结构、原理以及加速器的相关设计概述。首先介绍了卷积神经网络的基本组成，然后对卷积神经网络的识别原理进行简要阐述，接着介绍了每个种类层在卷积神经网络中的发挥的作用以及其运算方式。然后详细介绍了两种经典卷积神经网络的背景及其整体结构，分别为 LeNet 和 VGG-16。最后介绍了本领域的研究者所取得的研究成果及相关工作。

第3章　卷积神经网络算法模型优化

本章针对传统的卷积神经网络算法模型进行优化。首先介绍本文设计的加速器的基本模型结构，然后分别从卷积运算、池化运算、全连接层运算以及非线性激活的算法进行优化，并介绍每个模块算法的优化方案。最后介绍了本文所采用的数据顺序存储方案。

3.1　基本结构概述

由前面章节所述，我们将全连接层的运算转化为卷积层运算，并整合到卷积运算模块来实现全连接操作。卷积层的结果不返回外部存储器，而是进行偏置或非线性运算，完成后将结果返回外部存储器。由于池化层不需要权重，数据流和执行操作单一，将池化运算单独成为一个模块，直接通过访存模块访存外部存储器。

3.2　卷积层算法优化

在第二章已经介绍了卷积操作的传统算法，本小节对卷积算法进行优化。由于卷积操作的每个循环之间没有数据依赖，因此，可以调整循环顺序，以充分使用乘法阵列，使利用效率达到最大限度，将可以将循环拆分为多个子循环。

为了将卷积运算更好的映射至乘法器阵列上，我们重新优化循环顺序，并对数据流分组、分批，以流水的方式并行输入乘法器阵列，将大卷积转换为小卷积去送给乘法阵列运算，这样即可以减小数据的缓存压力，也可以减轻带宽的影响。在输入特征通道对其作输入并行宽度为 M，即输入并行因子为 M；在输出通道作输出并行为 N，即输出并行因子为 N，则输入特征数据被切割为 CH_{in}/M 个通道组，输出数据被切割为 CH_{out}/N 个输入通道组，输出通道对应于权重，即权重同样被切分为 CH_{out}/N 个权重组。最后一个通道组中的通道数量可能不足 M 或者 N。如图 3.1。

如果将切分后的子卷积直接并行输入进乘法阵列中依次计算，按通道组为单位进行卷积会使得输出缓存的压力巨大，且不容易控制。所以在通道组中以计算

块为单位作为一次单次卷积运算，并将单次计算块的卷积结果提交缓存，以每个计算块的结果进行缓存提交。

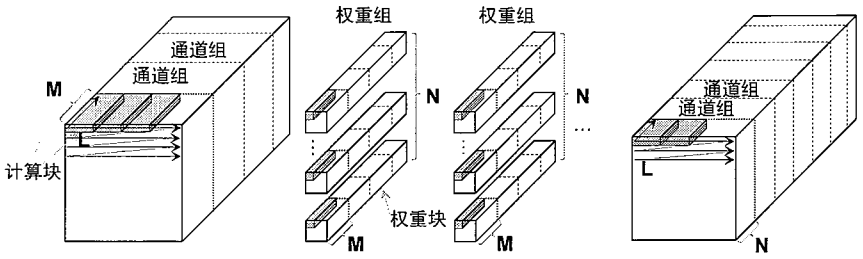

图 3.1　优化的卷积层算法数据分组

Figure 3.1 Optimized convolutional layer algorithm data grouping

在图 3.1 中可以看到，输入特征沿着通道方向切分为 CH_{in}/M 个输入通道组，每个通道组沿深度方向将 M 个数据搬进乘法阵列中做运算($0\rightarrow M$)，第一个 M 数据结束后沿着宽度方向进行 L 次这样的操作($0\rightarrow L$)，依次将 L 个 M 长度的数据输入给乘法阵列。L 为最大计算块宽度，每层有$(H_{out}\times W_{out})/L$ 个计算块。当所有计算块计算结束后，意味着此次通道组计算结束，开始下一输入通道组的运算。依次计算，直至 CH_{in}/M 个输入通道组全部计算结束。

输出特征沿着通道方向切分为 CH_{out}/N 个输出通道组，每个通道组将乘法阵列的计算结果依次填充，N 沿深度方向，L 沿宽度方向。直至所有计算块全部填充，意味着此通道组输出特征结束，进行下一个通道组的输出特征填充。依次计算，直至 CH_{out}/N 个输出通道组全部计算结束。

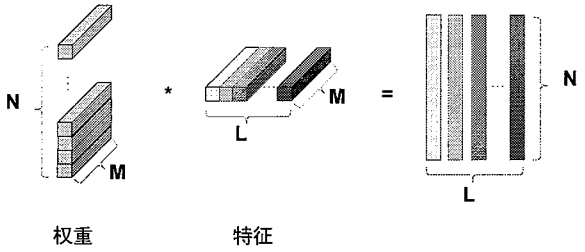

图 3.2　卷积操作在乘法阵列执行过程

Figure 3.2 Convolution operation performed in PE array

权重同样被分为 CH_{out}/M 个权重组，每个权重沿深度方向分为 CH_{in}/M 个权重块。在执行卷积操作时，取第一个权重块中每个权重相同位置的权重作为乘法阵列的输入，如图 3.1，依次沿着 K_x 方向，K_y 方向，直至所有的权重均参与运算，意味着此权重块计算结束；然后取下一个权重块的数据参与运算，直至完成 CH_{in}/M 个权重块的计算；接着取下一个权重组的数据，直至完成 CH_{out}/M 个权重

组的计算操作。由于权重数据量大，会浪费大量的时间去搬运，而且输入特征会重复用到同样的权重，所以我们可以固定权重，让输入特征变化，这样会大大提高复用效率，节省时间。如图 3.2，首先输入 N 个 M 长度的权重数据，然后输入依次输入 L 个 M 长度的特征，每次乘法结果进入加法阵列中求和，并将最终的卷积结果填充至输出通道。本文所采用的卷积优化算法参考英伟达公司的开源项目 NVDLA 加速器[82](NVIDIA Deep Learning Accelerator)中的 Atomic Operation、Block Operation、Channel Operation 以及 Group Operation 等。

3.3　卷积运算单元

卷积运算是神经网络最重要的计算之一，卷积层的计算效率也是影响整个加速器峰值算力的重要因素。在设计卷积运算单元时，需要着力优化阵列单元的访存和计算效率。由于外部存储器存在读写延迟，这会导致数据输入进阵列单元工作效率的大幅降低，这也是大多数时候带宽成为系统瓶颈的缘故之一，所以要尽可能的减少外部存储器的访存。为了解决这一问题，一般采用片上缓存做缓冲，通过 DMA 将数据搬进片上缓存，然后在从片上缓存将数据搬进阵列，可以减小访存带宽的压力。另外，阵列大小不是越大越好，阵列越大会导致更长的逻辑延时，时序不容易收敛，需要更深的流水来减少关键路径延时，继而需要更多的寄存器，使得面积、功耗都变大。

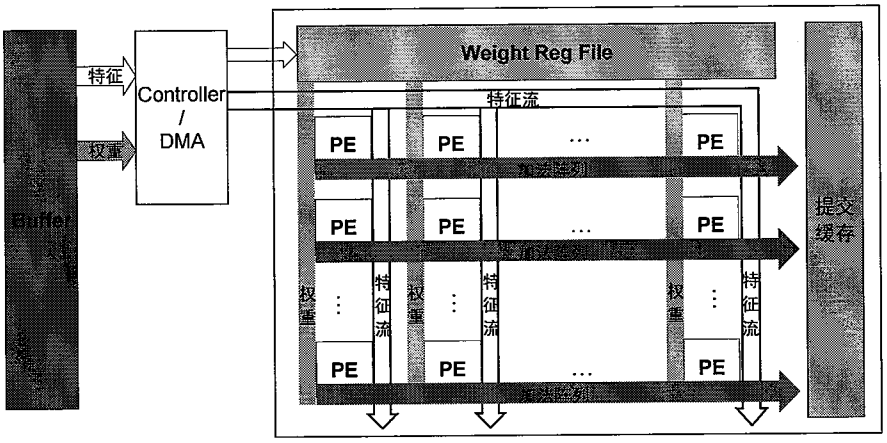

图 3.3　乘法阵列单元整体结构

Figure 3.3 Structure of PE array

图 3.3 为本设计得乘法阵列单元的整体结构图。本设计使用 64×32 大小的阵列单元，由于切分的卷积操作和乘法阵列需要最优匹配，则切分的输入并行因子 M 为 64，输出并行因子 N 为 32，这样子卷积操作便可以像图 3.2 所的执行过程

在乘法阵列中计算；另外本文做采取的最大计算块的宽度 L 为 32，即单次卷积为进行 32 次输入特征的计算。数据流从片上缓存流入，输出给提交缓存。由于卷积运算以首先加载权重至阵列单元，然后加载特征的方式进行计算，加载权重需要 32 个周期，加载完一次特征需要 32 个周期，这样单次加载会浪费加载权重的 32 个周期，带来很长的时序浪费，所以对权重进行预加载可以“隐藏”加载权重所需要时间，在阵列单元中设计一块权重存储区，通过乒乓操作的方式提高数据带宽，比如原本需要权重和特征数据分开一次进阵列，每个特征计算结束后才会载入下一组权重，而在乒乓模式下，在进行载入特征的同时在权重存储区也载入下一组权重，这样就省去了下一次单独输入权重所需的时间，如图 3.4。这样可以使得乘法阵列满载。

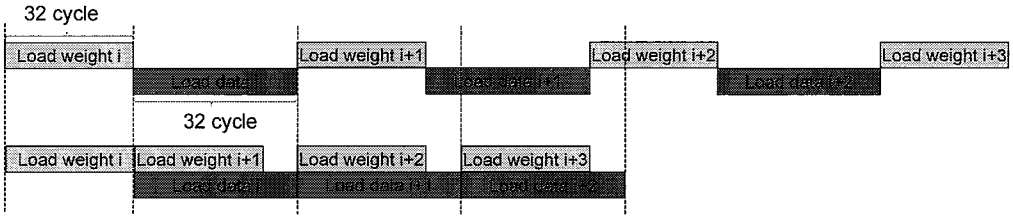

图 3.4　数据流水载入乘法阵列

Figure 3.4 Data pipelining into the PE array

阵列单元主要有三部分构成：其一为 64×32 个 16bit 的乘法器组成的乘法阵列，其二为加法器组成的加法阵列，另外还有缓存部分。

由于乘法阵列是在一拍完成所有的计算，这就使得到权重存储区中的所有数据在一个时刻均参与运算，所以采用寄存器堆的方式来做权重存储区，即使用 2×64×32×16bit 的寄存器。当权重加载完毕后，开始同样以 64 的并行度流入第一个特征，直至 32 个全部计算结束，并存放在提交缓存。在这期间第二组权重已经放进权重寄存器堆中，可直接开始下一次 32 个 64 并行度的输入特征计算，在进行此次计算的期间，下一组权重已经存放完成。依次循环，直至结束。

由上一节所述，优化后的卷积算法由计算块以及各通道组的分别运算完成，这些运算共同完成了整个卷积层在各个部分的循环运算，包含不同位置的输入特征和权重所参与的乘加运算，这些均需要控制逻辑来协调和配置，使其在正确的位置进行运算。本设计用状态机来控制这些循环，每次循环的数据由 DMA 来从片上缓存搬运。

为了减小阵列计算的逻辑延时，使用粗粒度插五级流水的方式减短关键路径长度，减少时序违例的可能性，将阵列做成流水结构，也达到时序快速收敛的目的。如图 3.5 所示。

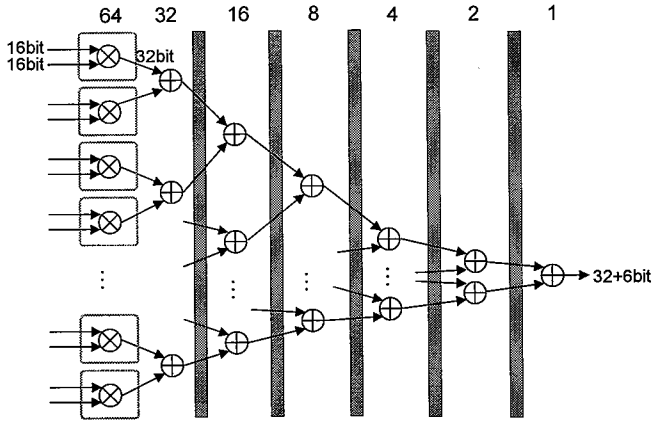

图 3.5　加法阵列

Figure 3.5 Addition tree of PE array

图 3.5 的顶端为每级需要的加法器数目，共需 63 个加法器，即每个输出并行通道需要 63 个加法器，由于本设计的输出并行度为 32，即计算阵列中还需 32×63 大小的加法阵列作为卷积运算的最终结果。另外，插流水共需要 5×64×32/2 个(32+6)bit 的寄存器，加法树的输入为 64×32×(32+6)bit，因此，共需要 5×64×32/2×(32+6)bit 的寄存器。

3.4　全连接层算法优化

全连接层的计算过程就是矩阵向量的乘积，所以全连接运算可以转化成卷积计算，直接使用卷积运算模块来实现全连接层的计算。

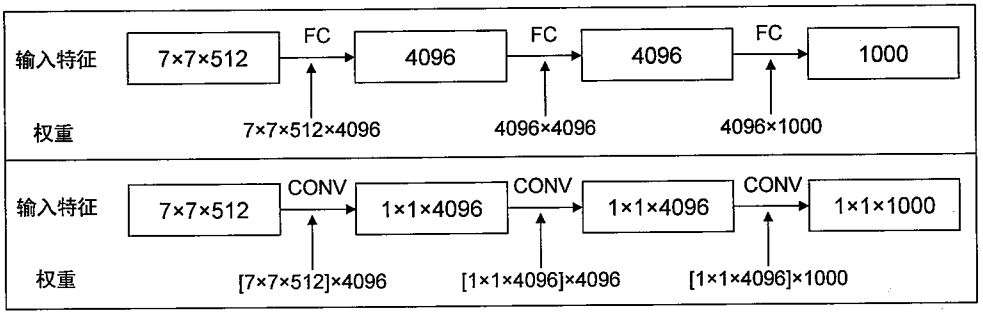

图 3.6　全连接运算转为卷积运算

Figure 3.6 Fully connected operation is converted to convolution operation

如图 3.6 所示，若 7×7×512 的层要跟 4096 个神经元的层做全连接，传统的全连接层计算需要用[7×7×512][4096]大小的二维矩阵去和输入向量进行矩阵乘

法，结果为 4096 的一维矩阵输出向量。本设计将权重重新调整为[7][7][512][4096]的卷积核和输入特征做卷积，最终得到[1][1][4096]三维输出向量。

可以得到全连接运算就是做这样的卷积操作：$K_x=W_{in}$，$K_y=H_{in}$，$S_x=S_y=1$，$P_{up}=P_{down}=P_{left}=P_{right}=0$，$H_{out}=W_{out}=1$。这样做不仅可以出色的完成全连接运算，还不会破坏特征图像的位置关系，不需要用专门的逻辑去完成全连接层的运算。

输入特征和输出特征用三维矩阵表示，权重，即卷积核用四维矩阵表示。全连接层的输入特征尺寸为 $CH_{in}\times H_{in}\times W_{in}$，权重为 $H_{in}\times W_{in}\times CH_{in}\times CH_{out}$，输出特征尺寸为 $1\times 1\times CH_{out}$。每个输出神经元与所有输入神经元相连接。如图 3.7 所示。

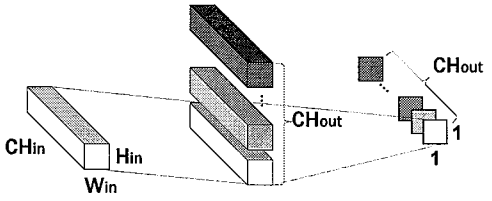

图 3.7　全连接层运算

Figure 3.7 Fully connected operation

另外，1×1 的卷积操作在一些网络发挥着十分重要的作用，可达到了很好的效果，如 NiN[79]。1×1 卷积操作具有降维的作用，其输出可以看作是在不影响输入输出的维度情况下，对输入进行形变，再通过 ReLU 进行非线性处理。

在全连接运算中，数据规模和数据量见表 3.1。

表 3.1　全连接层数据规模与数据量

Table 3.1 Full connection layer data size and data volume

	数据规模	数据量
输入特征	$[CH_{in}][H_{in}][W_{in}]$	$CH_{in}\times H_{in}\times W_{in}$
权重	$[H_{in}][W_{in}][CH_{in}][CH_{out}]$	$H_{in}\times W_{in}\times CH_{in}\times CH_{out}$
输出特征	$[1][1][CH_{out}]$	$1\times 1\times CH_{out}$

将卷积操作映射到阵列的过程中，当计算块长度大于加载权重的时间，便可以“隐藏”掉加载权重的时间，阵列利用率也会大幅提高；但是全连接计算用卷积代替后，会有 1×1 的输入特征，即搬运一个数据的过程中加载权重，这将无法“隐藏”加载权重的时间，阵列的利用率也会大幅降低。如图 3.8 所示。

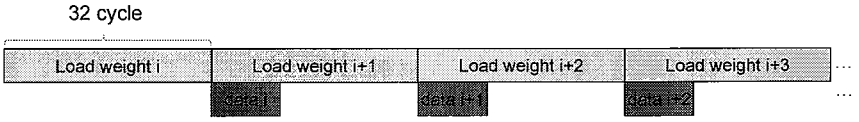

图 3.8　全连接运算映射到阵列的数据流水方式

Figure 3.8 Pipeline of mapping the fully connected operation to the PE array

3.5　其它运算

卷积神经网络中还包括偏置运算、批归一化、非线性运算和 Element-wise 运算。偏置运算和批归一化是针对通道，Element-wise 运算为单元素求积或者求和运算。在常见的卷积神经网络中，卷积层的输出通常会进行一系列的单通道或者单元素运算，因此本设计将卷积运算模块的输出结果至此类运算的数据输入端，这样可以很大程度上减少对外部存储器(DDR)访问次数，减少带宽需求，可以减少外部存储器的访问功耗和数据通路的延迟，最大程度的减少访存对系统的影响，也使得数据通路的控制逻辑设计简单。因为偏置运算和批归一化运算需要卷积的结果还需要从外部存储器中取数据，所以我们根据数据需求种类的数目将运算的先后顺序设计为偏置运算、批归一化运算、Element-wise 运算以及非线性运算。偏置运算比较简单，本文不再赘述。

3.5.1　批归一化

批归一化(Batch Normalization)(Loffe 等，2015)也是神经网络中重要的一层。批归一化是对输入特征节点的线性操作，让数据的分布永远处于激活函数的敏感部分，这样不仅可以加快模型的收敛速度，也可以有效地减少训练阶段的梯度弥散现象。比如某个神经元输入为 x，权重为 W，输出为 $y=f(Wx+b)$，激活函数 f 为 Tanh 函数。当 x 在[-1，1]之间变化时，输出随着输入变化，但是在此区间之外输出几乎没变化，即无论输入怎么扩大，激活函数的输出一直接近 1，也就是说，输出对比较大的输入值不敏感了。

我们可以对数据进行矫正，让输入值的分布永远处于激活函数的敏感部分。一旦训练结束，每个归一化层都有一组特定的参数，这意味着在推理过程中，归一化层就是对上一层的结果做简单的线性转换。

Batch Normalization 在推理阶段比较简单，但是反向比较复杂，一般在训练中会加入更多的子层来加速网络的收敛。Batch Normalization 的运算公式为公式 3.1，$\boldsymbol{O}$ 为输出结果，$\boldsymbol{I}$ 为输入的特征量，输入特征尺寸为 $CH\times H\times W$，$\boldsymbol{\mu}$是在训练阶段导出的偏置因子，尺寸可表示为 $1\times1\times CH$，$\boldsymbol{\sigma}$是在训练阶段导出的缩放因子，尺寸可表示为 $1\times1\times CH$，在公式 3.1 中，用[x][y][z]代表其三个维度。Batch

Normalization 运算对每个通道进行线性变换，使输出结果的数值分布更加贴近于正态分布。

$$\boldsymbol{O}[x][y][z] = \frac{I[x][y][z] - \mu[x][y][z]}{\sigma[x][y][z]} \quad \ldots(3.1)$$

3.5.2 Element-wise 运算

Element-wise 是针对两个张量之间的操作，是对两个输入特征中的对应元素做基本运算，输入特征的尺寸可表示为 $CH \times H \times W$，输出特征的尺寸为 $CH \times H \times W$。本设计 Element-wise 支持求积、求和以及做差运算。

3.5.3 非线性运算方案优化

激活函数为高度线性的神经网络引入非线性运算，让神经网络具备强大的拟合能力。ReLU 函数由于其简单高效，很适合硬件实现得到广泛的应用。但是其它的函数不适合硬件实现，比如 Sigmoid 函数，它们属于超越函数的范畴。

$$Sigmoid(x) = \frac{1}{1 + e^{x}} \quad \ldots(3.2)$$

我们采用基于查找表的等距分段逼近来高效实现这类超越函数。

首先，需要在不影响硬件实现难度的情况下，选取计算精度最高的逼近算法。我们分别使用 $y = kx + b$ 线性逼近，$y = ax^2 + bx + c$ 二次逼近以及三阶泰勒展开三种多项式展开逼近拟合 $y = e^x$，并对比其误差，如图 3.9 所示。

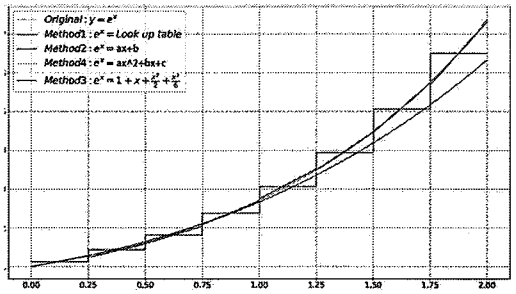

图 3.9 指数函数的几种逼近拟合方法

Figure 3.9 Several Approximation Methods of Exponential Function

可以明显对比出，泰勒展开在[0,1]范围拟合度是很高的，但是在后面的区间偏离越来越大。Sigmoid 在[-10,10]的区间，虽然会在[-10,-6][6,10]趋于水平，泰勒展开的拟合误差仍然无疑会是最大的。在其他条件相同的情况下，对上图误差量化分析见表 3.2。

表 3.2　指数函数逼近误差

Table 3.2 The exponential function approximation error

算法	平均误差
线性逼近	0.07
二次逼近	1.2e-5
泰勒展开	0.3

所以我们使用二次逼近 $y=e^x$，等距分为 16 段，通过 MATLAB 将拟合出的每段二次函数的系数,并量化为 8bit 定点，并存放进一块 RAM 中供计算读取。由于二次函数中有平方项，所以我们进一步对其简化，使其达到最优的计算效果：

$$y=ax^2+bx+c \rightarrow y=(ax+b)x+c$$

这样仅仅需要乘法器和加法器就可以计算出e^x的结果。根据e^x的计算结果，便可以计算出 Sigmoid 的函数值。

为了减少激活层对卷积神经网络最终的识别精度的影响，我们需要对硬件计算的精度进行分析。以 Sigmoid 函数为例，得出与软件计算的平均误差为 2.3460e-05，最大误差为 4.3830e-05。

3.6　池化层运算优化

池化层根据特征不变性减少特征大小，减少后面层的运算压力。池化层的输入特征的高为 H_{in}，宽为 W_{in}，池化算子的高为 K_x，宽为 K_y，通过横向为 S_x，纵向为 S_y 的跨步，输出高为 H_{out}，宽为 W_{out} 的特征。

传统的计算方法，需要对输入特征的每一个数据进行 $K_x \times K_y$ 次重复的读取，一般会使用一块片上缓存来存放输入特征，这样最明显的代价就是使用更大的面积；而且这样做会增加逻辑的控制难度。另外，当步长小于池化算子的长度，会多次取矩阵框中同样的数据，重复的数据会继续进行多次重复的运算（平均、最大和最小运算），从而降低了运算的效率。如图 3.10 所示。

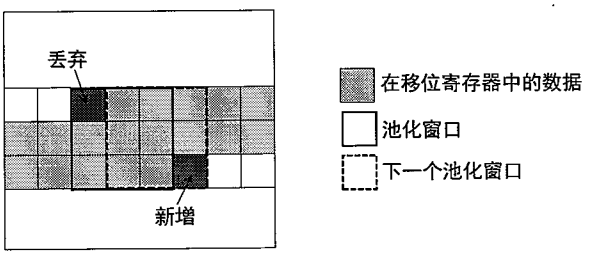

图 3.10　传统的滑窗池化操作

Figure3.10 Traditional sliding window pool operation

二维的操作本身比一维操作复杂，控制也更复杂，我们采用将二维操作分解为两次一维操作的方式进行池化运算，分别从宽度方向和高度方向进行池化操作。即就是 $K_x \times K_y$ 个数的这些操作，可以类比成将这 $K_x \times K_y$ 个数划分为 K_y 个集，每个集合中有 K_x 个数，计算每组的最大、最小或者均值，得到 K_y 个数；然后计算这 K_y 个数的最大、最小或者均值。这样可以有效的避免多次重复的运算，也大大减少了控制难度，更加有利于硬件实现。

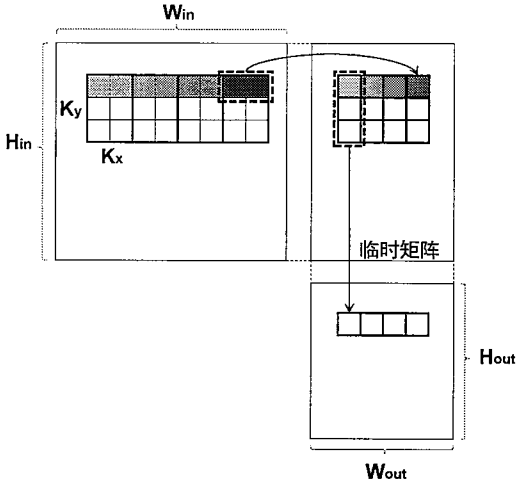

图 3.11　池化算法优化

Figure 3.11 Pooling algorithm optimization

首先进行宽度方向的池化，可以得到高仍为 H_{in}，宽为 W_{out} 的中间矩阵。中间矩阵中的第 i 行对应输入特征的第 i 行，第 j 行对应输入特征的第 $S_x \times j$ 列至第 $S_x \times j + K_x - 1$ 列的数据进行池化运算后的结果。然后对临时矩阵进行高度方向的池化，可以得到$[H_{out}][W_{out}]$的输出矩阵，即输出特征，其中，输出矩阵中的第 i 行对应临时矩阵的第 $S_y \times j$ 行至第 $S_x \times j + K_x - 1$ 行的数据进行池化运算后的结果，第 j 列对应临时矩阵中的第 j 列。如图 3.11 所示。此优化算法参考于文献[82]。

3.7　数据存储方式优化

根据前面小节介绍阵列单元的数据输入并不是数据原本的存储方式，是经过重新排布的，且针对阵列计算所定制的最优的数据流方式，这就需要在送进阵列之前将数据重新存储成有利于阵列计算的方式。数据从外部存储器搬进阵列经过片上缓存，所以首先用 DMA 将数据搬进片上缓存，然后再经过 DMA 将数据送给阵列，并且在这中间将数据存储完毕。显然，特征和权重在外部存储器或者片上缓存的存储方式紧密的关系着 DMA 的设计复杂度以及工作效率。本文的数据

存储方案参考英伟达公司的开源项目 NVDLA 加速器[82](NVIDIA Deep Learning Accelerator)中的存储方案，即子卷积运算的特征以及权重按照固定的格式存储。

3.7.1　片上缓存上的存储方式

1. 权重在片上缓存的存储方式

根据前文中优化的卷积算法，每次需要从片上缓存中将 32 个卷积核取出[1×1×64]的子块输入进阵列单元，且在 32 个时钟周期完成，即每个时钟周期都会从片上缓存中取出[1×1×64]的子块。在完成第一块后，沿 K_x 方向 K_y 方向依次按顺序取出每个权重块中的小子块。接着依次按顺序取出 CH_{in}/64 个权重块的每个子块。这样依次按顺序取出，直到完成所有的权重组。

如果数据在片上缓存中已经按照阵列需要的数据存储方式排列，那么 DMA 可以通过地址自增突发读的方式从片上缓存中将数据搬进阵列中，这样会让 DMA 的工作效率高很多。每完成一次搬运，将地址计数器复位，继续搬运数据，使得 DMA 的设计简单许多，访存成为系统的瓶颈的可能性减小。

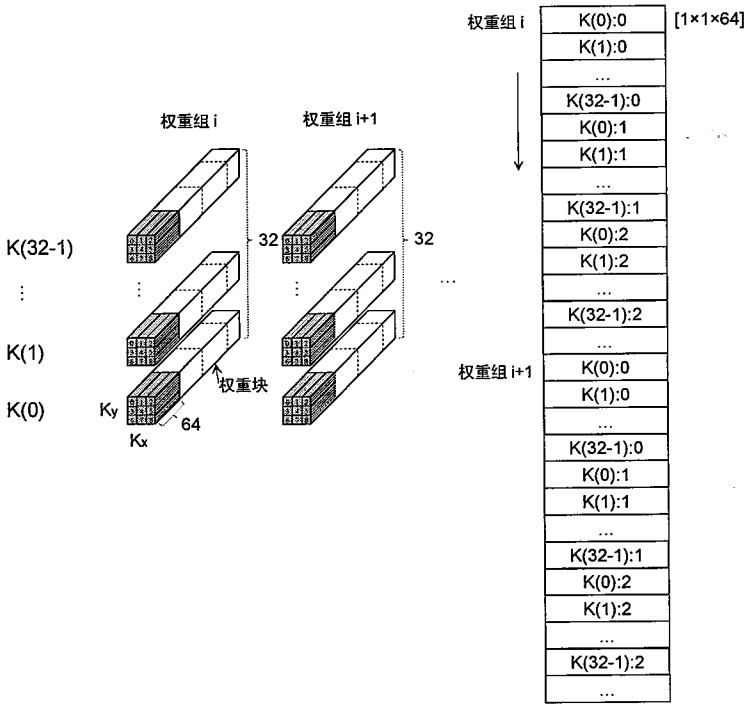

图 3.12　权重在片上缓存的存储方式

Figure 3.12 Mapping of weight on BRAM

如图 3.12 所示，将权重分为 CH_{out}/32 个权重组，每组有 32 个权重，分别为 $K(0)$、$K(1)$…$K(32\text{-}1)$，然后将每个权重分为 CH_{in}/64 个权重块，每个权重块分为 $K_x \times K_y$ 个权重子块，每个子块大小为[1×1×64]。每次将权重组中 32 个权重的相

同位置的权重子块按顺序搬进片上缓存，比如将权重组 i 的 $K(0)$、$K(1)$...$K(32\text{-}1)$ 每个权重的第一个子块 $K(0)$：0、$K(1)$：0、...、$K(32\text{-}1)$：0 依次按照地址自加的方式存进片上缓存，完成后继续将 $K(0)$：1、$K(1)$：1、...、$K(32\text{-}1)$：1 存进缓存，依次按顺序存；在第一个权重块结束后，进行下一个权重块，存储方式和第一个权重组的方式相同。完成后将下一个权重组按顺序存进片上缓存，当片上缓存空间不足时，使用分组替换。

2. 特征在片上缓存的存储方式

卷积运算中，需要读取不同位置的输入特征数据，且重复读取，由于网络本身有步长和加 pad 等参数，造成在读取数据时，检索数据地址没有固定的规律可依，因此在每次读取数据时需要提前计算出所需数据的三维坐标 H 方向、W 方向以及 CH 方向的索引，并映射到片上缓存的地址。

为了简化索引转换至片上缓存地址这一问题，特征在片上缓存的存储越简单越好。

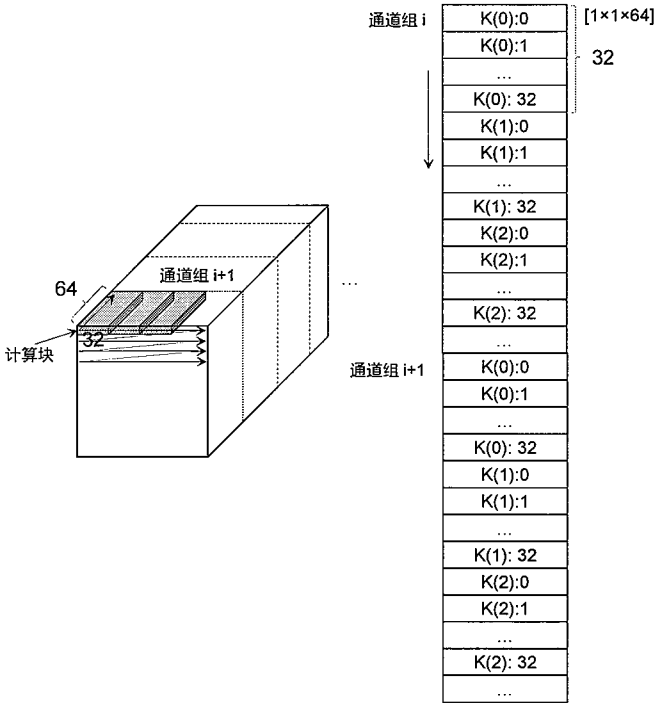

图 3.13　特征在片上缓存的存储方式

Figure 3.13 Mapping of feature on BRAM

输入特征依然根据有利于阵列输入计算的方式进行存储。将特征分为 $CH_{in}/64$ 个通道组，每个通道组又分为多个计算块，每个计算块分为 32 个[1×1×64]个子块，片上缓存的读位宽 64×16bit。依次将每个子块按顺序存储到片上缓存，

子块完成后进行下一个子块，计算块完成后进行下一个计算快，通道组完成后进行下一个通道组。依次按顺序完成。如图 3.13 所示。

3.7.2　外部存储器上的存储方式

1. 权重在外部存储器的存储方式

通过 DMA 从外部存储器中将数据搬运至片上缓存。DMA 在突发读写时的效率是最高的，所以本设计发挥 DMA 的最优效率，将所有的数据搬运均通过突发读的方式搬运。当外部存储器中的数据和片上缓存中的数据存储方式一样的话，就可以利用突发读的方式将数据依次搬进片上缓存。

所以我们在软件编译网络的时候需要通过 CPU 将数据重新存储成前文所述的方式，并存进 PC 内存中，通过 PCIe 将数据搬进 FPGA 的外部存储器中。

2. 特征在外部存储器的存储方式

由于输入特征不是只读，比如说当前层的计算结果，也是下一层计算的原始数据，因此特征数据在外部存储器(DDR)中的存储方式将直接影响到写回逻辑的设计复杂度。因此我们在软件编译网络的时候需要通过 CPU 将数据存储成前文所述在片上缓存相同的方式，并存进 PC 内存中，通过 PCIe 将数据搬进 FPGA 的外部存储器中。但是当卷积输入特征过大或者反卷积操作写回时，当前层可能无法瞬间做到最优的空间分配，所以在每完成一行数据的搬运后在外部存储器中预留 32×16bit 的整数倍大小的空间来缓冲大卷积的特征写回，在每完成单次卷积操作时，即就是完成一次计算块的卷积操作后在存储空间中预留块扩大存储空间；在完成一行的卷积操作后同样预留行扩大存储空间；同理，通道组的完成也可能造成卷积扩大后的数据无法存储，所以在通道组的卷积操作完成后，也预留组扩大存储空间。特征在写回外部存储器时只需给定首地址并突发地写回，这样简化了特征写回控制电路的复杂性，提升了特征写回的工作效率。

3.8　数据量化

定点数就是包含符号位、整数位以及小数为的数据格式，通过一个共享的比例因子来确定数据表示范围以及数据精度，比如缩小比例因子后，数据可表示范围缩小，而数据精度会提高。图 3.14 为 8bit 的定点数格式。

定点数因小数点位置固定，且表数范围有限，而卷积神经网络中各个层的数据（特征以及权重）范围相差比较大，统一且固定的小数点位置并不适合。

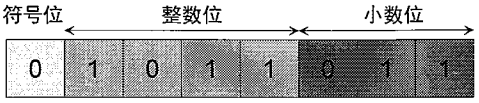

图 3.14　定点数据格式

Figure 3.14 Fixed point data format quantization

Williamson 在 1991 年提出动态定点数[83][84]的概念，即每个特征或者权重单独有一个缩放因子，不同特征或者权重的缩放因子可以不同，每层的数据格式也可不同。在网络训练完后，通过高级语言对权重的所有张量进行检索，得出其数据范围，然后便可以确定数据的整数位数，余下的即为小数位数。量化是对训练完成后导出的模型进行定点等效，量化过程由软件完成。由于量化的定点数的表数范围精度无法和浮点数相比，所以在硬件所完成的定点数的结果会有精度损失。动态定点数表示的具体数值为公式 3.3，其中 s 为符号位，fl 为小数位，B 为数据位宽，x_i 为尾数。

$$(-1)^{s}\cdot 2^{-fl}\cdot\sum_{i=0}^{B-2}2^{i}\cdot x_i \qquad \ldots(3.3)$$

3.9　本章小结

本章主要对卷积神经网络算法模型进行优化。首先介绍了本文设计的加速器的基本模型。然后对介绍了本文所使用的卷积计算的优化方法，并对阵列单元进行了详细的介绍：针对卷积层的大数据量问题，采用卷积切分的方式，将卷积操作分为若干子运算，通过输入输出通道并行的计算方式，提高计算效率；另外，利用切割后的数据在通道方向做数据复用，减少数据的重复搬运；采用顺序数据存储方式不仅可以提高 DMA 工作效率又可以提高阵列单元工作效率。对加速器的核心阵列单元进行了整体的优化，采用在阵列内部加权重寄存器堆的方式来做乒乓操作，减少数据搬运时延。然后针对非线性激活函数中难以用硬件实现的超越函数，采用二次逼近的方式实现，而且精度损失很小，同时进行了批归一化层的融合。接着对传统的池化层的滑窗运算进行了分析，得出传统的滑窗池化运算需要用到较多的移位寄存器，而且二维运算的控制逻辑复杂，针对这一问题，采用先横向池化后纵向池化的算法，少用了大量的寄存器，而且一维运算的控制逻辑要简单很多。然后着重分析了常见的加速器所面临的带宽限制问题，采用先进

的数据存储方案，利用 DMA 的突发传输搬运，不仅可以达到高效传输的目的，而且可以大大提高 DMA 的工作效率。最后介绍了本文使用的量化方法。

第4章　卷积神经网络加速器硬件架构

本章将根据第三章优化后的模型做硬件设计与实现。首先介绍了本设计的总体硬件架构；然后逐一的描述每个模块的设计思路以及实现方案；接着将从时钟系统设计、复位系统设计、对外通信接口定义以及内部模块的通信接口定义分别对本加速器进行更完善的介绍。

4.1　加速器整体架构

卷积神经网络加速器的整体架构如图 4.1 所示，为 CPU+FPGA 的计算架构，其中 CNNPU 为本文所设计的卷积神经网络加速器 IP。CPU 和 FPGA 通过 PCIe x8 通信。数据通路有两条：其一为从 CPU 的内存中将数据搬进 FPGA 的外部存储器 DDR 供加速器计算，并从外部存储器通过 PCIe 写回 PC；另一条通路为对加速器的寄存器配置。两条数据通路经过总线互联由 PCIe 通信，或者访存外部存储器 DDR。

工作流程为 CPU 将软件编译过的数据存储到内存中，并将网络参数提取哟用于配置加速器寄存器。然后通过 PCIe 的 AXI 接口将数据搬进 FPGA 的 DDR 中，同时通过 PCIe 的 AXI-Lite 接口配置加速器的寄存器。配置结束后便可以可是工作，加速器的三个计算模块通过对访存模块的请求/有效信号来收发数据包，访存模块会通过仲裁的方式发送或者接收数据包，最后将加速器计算完成的数据通过 PCIe 搬回 PC。

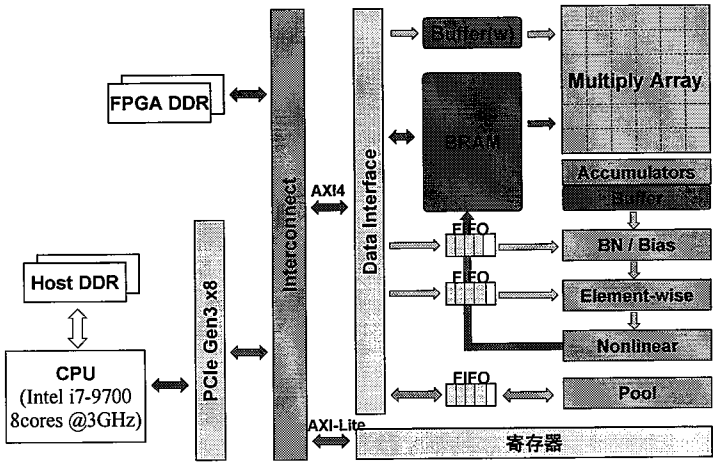

图 4.1　加速器整体架构

Figure 4.1 Accelerator overall framework

4.2 卷积运算通路

卷积运算模块由片上缓存、DMA 以及阵列单元组成，其中阵列单元又包括乘法阵列、加法树、权重寄存器堆以及运算结果提交缓存。片上缓存会占用整个卷积运算模块的大量存储器资源。阵列单元占用整个卷积运算模块的大量逻辑资源。

DMA 与片上缓存之间除了特征和权重数据包的传输，还有由于控制数据包正确输出的控制信号。在片上缓存内部设计有一对权重以及特征的地址指针，以计数器的方式实现，用来做新旧数据的区别。当来新的卷积操作时，这组地址指针计数器将被复位，以指针计数器来区别前一次卷积操作所使用的权重以及特征数据为旧数据，片上缓存上的数据要被本次所需的新数据覆盖；当 DMA 在向片上缓存中更新数据时，也会同时更新地址指针计数器，以区别这次数据为新数据。而当上一组权重或者特征可以复用时，仅复位特征或者权重中的某一个地址指针。前文所述的每个计算块的子运算就是通过这两个地址指针来做控制的。同时，第二个 DMA 会根据优化后的卷积算法循环顺序向片上缓存发送数据请求，并将返回的有效数据输入进阵列单元进行卷积运算。

第二个 DMA 与片上缓存也是采用 ready/valid 接口，若所请求的数据存在于片上缓存中，并且地址指针计数器标记为此次所需的新数据，则片上缓存会响应这一请求，同时将数据打包发送给 DMA；若所请求的数据未被地址指针计数器标记为新数据，片上缓存不会对这一请求应答，直到第一个 DMA 将数据搬进片上缓存。这样不仅可以保证输入进阵列的数据为此次卷积层运算所需的数据，也保证此次卷积结果的正确性；而且两个 DMA 可以同时工作，数据搬运和阵列计算可以同时进行。片上缓存数据位宽为 64×16bit，使用 16 块 1024 深度的缓存，即 2M。在 100MHz 的时钟频率下，带宽可以跑到 204.8G/s，带宽优于同样存储大小的单块片上缓存。

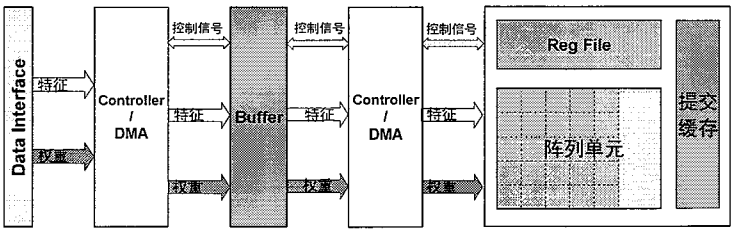

图 4.2 卷积运算通路

Figure 4.2 Convolution path

卷积运算模块的工作流程为：首先会复位片上缓存中的特征与权重的地址指针，然后启动两个 DMA，从 DDR 中读取出数据，由 DMA 将数据搬进片上缓存；然后再由第二个 DMA 将返回的有效数据搬进乘法阵列来做卷积运算，并将运算结果输出给提交缓存，供计算模块做下一步运算。如图 4.2 所示。

4.3　池化运算通路

根据前文所述的优化算法，池化运算通路的结构如图 4.3 所示。输入特征通过访存模块被送入横向池化运算模块，接着将输出给纵向池化运算。由于输入特征数据是由宽度方向→高度方向→通道方向的顺序输入，而纵向需要从高度方向→宽度方向→通道方向计算，所以需要将临时数据缓存。

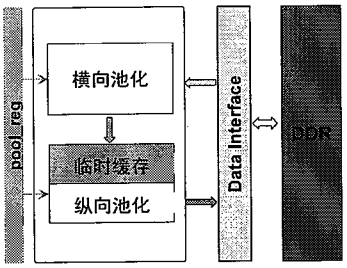

图 4.3　池化运算通路

Figure 4.3 Pooling path

横向池化模块计算结构简单，就是根据数据的输入，从行到列，再到下一层的行列，依次做池化运算，并根据 S_y 和 K_y 等参数在寄存器中做控制信号，不需如传统池化那样缓存很多的数据，输出同样为从行到列，再到通道方向。

除了卷积以及池化运算偏置运算、批归一化、Element-wise 以及非线性运算的模块。数据交互来自卷积层的输出提交缓存、DDR 以及将最终的结果输出给 DDR，之间均通过 ready/valid 接口来确定数据的有效性，保证计算数据的匹配和结果的正确性。另外，每个模块的内部寄存器均通过 cfg_reg 来配置输入。如前文所述，批归一化以及偏置运算是对卷积操作的后续处理，所以数据来源一部分来自卷积运算模块的输出，另一部分来自 DDR 中训练好的参数。计算结束后将数据输出给 Element-wise 运算模块，最后输出给非线运算模块，然后将最终结果输出 DDR。

4.4 批归一化以及偏置运算

偏置运算是将输出通道加一个参数，而批归一化是对输出通道与一个参数相减再与另一个参数相乘，将这两个参数记为 data1_from_ddr、data1_from_ddr。

由于卷积层的输出结果为 32bit，需要将输出结果左移至 16bit 传输。另外，data1_from_ddr、data1_from_ddr 均通过 FIFO 做缓冲。由于网络配置不同，有的网络不需要做通道运算或者其它的运算，所以本设计全部通过 MUX 分发或者 MUX 旁路的方法解决这一问题，只需要配置寄存器将配置信息输入 MUX 的 op 端口，来操作数据路径的选择。如图 4.5 所示。当网络不需要做通道运算时，可以经过 MUX 旁路输出直接给 Element-wise 做运算。

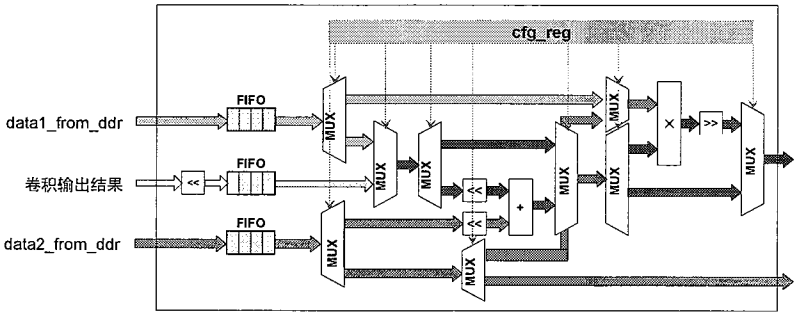

图 4.4 批归一化及偏置运算模块

Figure 4.4 BN/Bias operation

4.5 Element-wise 运算

Element-wise 运算是将输出特征的每个数据（元素）做单点乘法、加法以及减法。Element-wise 运算通路与上节所介绍“通道运算”的实现方法相同，同样是通过配置寄存器配置 MUX 旁路的方式来分发数据。

本设计的 Element-wise 运算模块支持求和、做差以及求积运算。输入数据中一组来自批归一化以及偏置运算的计算结果，另一组数据来自外部存储器中的权值。当网络不需要作 Element-wise 运算时，可在网络编译的过程中进行将数据配置给寄存器，然后使用同样的方式：寄存器来控制 MUX 的分发或者直接将结果传递给非线性运算。数据输入有两组：一组为卷积操作或者进行 BN/bias 操作的结果，另一组为从外部存储器中取得权值，Element-wise 运算便是对这两组数据进行求和、做差以及求积运算。

4.6　非线性运算

非线性运算的基本算法已经在前面章节介绍，并且详细阐述了对于复杂激活函数中的超越函数进行二次逼近拟合的算法进行硬件优化。由于 ReLU 的运算方式很简单，将ReLU 函数运算与其它非线性运算区分。前文已经已经介绍，Sigmoid 等函数对硬件实现来说比较复杂，占用资源大，计算速度也很慢，所以采用低级函数去拟合逼近来完成复杂函数的运算。当数据输入后，寄存器首先对其进行函数选择，低位“0”即网络需要进行 ReLU 激活，而高位“1”时网络需要进行 Sigmoid 或者 Tanh 进行非线性运算。如图 4.7 所示。

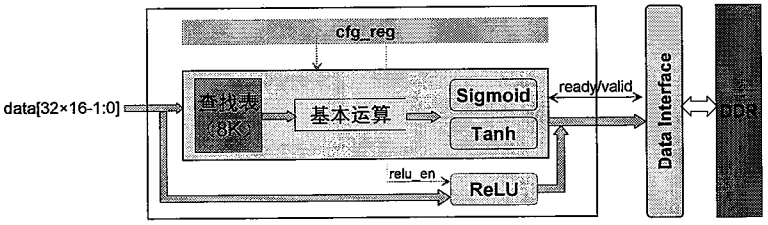

图 4.5　非线性运算模块

Figure 4.5 Non-linear operation

4.7　时钟与复位系统设计

加速器系统时钟主要分为两部分：第一部分为 PCIe xdma 的时钟域，第二部分为 DDR 的时钟域，虽然这两部分的参考时钟域均为 100MHz，但确实是分为不同的时钟域，我们在FPGA 中可以直接用总线互联 Interconnect 来做跨时钟域。

PCIe xdma 的由内部产生 100MHz 的时钟，输出数据在 125MHz 的时钟下。由于输出数据通过 AXI 给 DDR，通过 AXI lite 给 CNNPU，所以需要经过互联来做跨时钟域。另外 CNNPU 工作在 100MHz 的时钟频率下，所以需要做一次分频，输入 125MHz，输出 100MHz。CNNPU 也需要和 DDR 通信，同样需要互联来跨时钟。DDR 控制器工作的参考时钟为 100MHz，实际速度在 1200MHz。由于需要测试 DDR 的实际带宽，将 Timer 挂在 DDR 控制器上，由控制器提供 100MHz 的时钟。如图 4.8 所示。

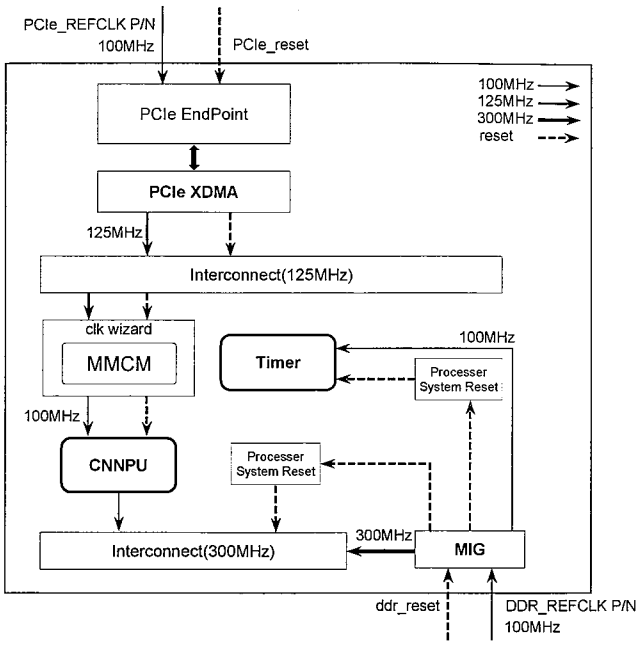

图 4.6　时钟与复位系统

Figure 4.6 Clock and reset system

复位系统同样分为两个部分：一部分为 PCIe xdma 的复位，另一部分为 DDR 控制器的复位端。PCIe xdma 为低电平复位，DDR 控制器为高电平复位。

4.8　通信接口定义

1. 外部通信接口

CNNPU 通过两个接口与外部通信。寄存器并不需要通过突发模式来进行高效配置，采用 AXI Lite 进行通信，所以 CPU 通过 AXI Lite 接口协议对寄存器进行配置。DDR 由于其器件本身有刷新、预充电、潜伏期等操作流程，读写数据不能在即刻完成，AXI 可以通过 Outstanding/Out-of-order 模式隐藏 DDR 读写的延迟，采用 AXI 总线协议进行访存数据通信。通信接口定义如图 4.9 所示。

AMBA AXI4[53]是 ARM 公司推出的第四代接口规范，高级可扩展接口。AXI4 协议共 5 个通道：读地址通道(AR)、读数据通道(R)、写地址通道(AW)、写数据通道(W)以及写相应通道(B)。读地址通道主要包括读请求的基地址、最后地址及突发长度等；读数据通道是从设备返回的数据；写地址通道主要包括写请求的基地址、最后地址以及突发长度等；写数据通道为向从设备写的数据；写响应通道是对本次写数据操作返回的响应。

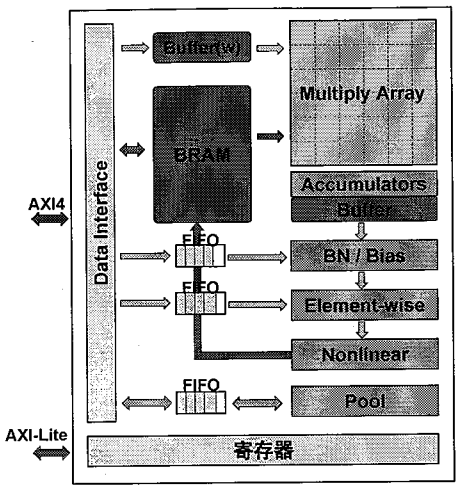

图 4.7　通信接口

Figure 4.7 Communication interface

2. 内部通信接口

内部模块的读通道分为两部分：读地址通道和读数据返回通道。由内部 DMA 主动对 DDR 发起读请求，若响应本次请求，则通过数据接口模块将本次读出的数据包搬进片上缓存，然后进行下一步的操作。读地址数据包包含了被读取的数据位于 DDR 中的基地址以及本次读取数据的突发长度总数。数据接口模块收到读请求后，通过 AXI4 总线与 DDR 控制器(MIG)进行通信，并将返回的读取的数据放在读数据返回通道，通过 ARID 信号将数据发送给请求数据的模块。

内部模块的写通道与读通道类似，DMA 对 DDR 进行写操作时，需要先发送写地址数据包，然后发送写数据。由于写地址与写数据均由内部发起，指向 DDR，所以写地址数据包和写数据可以共用一个通道。

4.9　本章小结

本章为卷积神经网络加速器硬件设计的核心。本章分别介绍了每个模块的设计方案以及具体实现方法，包括卷积运算通路、池化运算通路以及其它运算通路设计。并对每个模块的内部硬件设计进行充分的阐述。包括卷积运算的内部结构，池化运算的硬件电路；还有批归一化运算、偏置运算以及非线性运算等通过 MUX 分发的设计原理进行适配不同的网络。接着介绍了对外通信接口定义以及内部模块的通信接口定义。最后我们介绍了加速器的时钟以及复位系统的设计。

第5章　验证与性能分析

本章首先介绍 FPGA 的选型，仿真平台的搭建以及仿真流程。然后将 LeNet 与 VGG-16 作为目标网络在搭建的仿真平台上进行验证，并得到硬件的物理资源占用以及各部分功耗。接着分别对 LeNet 与 VGG-16 网络的每层运行延时，运算效率，以及能效比等所方面进行分析。最后对大型卷积网络 VGG-16 进行 Roofline 模型分析，对本文所设计的卷积神经网络加速器进行性能评估。

5.1　实验环境和验证平台

本文实验使用 XILINX 公司的 vivado18.3 作为硬件开发软件，采用 Xilinx Kintex UltraScale FPGA 定制的开发板作为 FPGA 平台，如图 5.1。具体型号为 xcku115-flvf1924 器件。Kintex UltraScale 系列器件以高性能、丰富的资源著称，并提供低成本、节省空间的倒装裸模封装。另外 Kintex UltraScale 系列器件在 20nm 的工艺节点上做到了最佳的性能、最优的功耗以及最高的性价比。

图 5.1　基于 XCKU115 FPGA 定制的加速卡

Figure 5.1 Accelerator card customized based on XCKU115 FPGA

FPGA 通过 PCIe Gen3 x8 和主机端 PC 的 CPU 通信，主机端 CPU 为 Intel i7-9700 8cores @3GHz。主机负责将需要运算的输入图片进行归一化，并按照前文所述的存储算法存放在主机内存中，通过 PCIe 将数据传输给 FPGA 的 DDR，然后 FPGA 做加速运算，完成运算后，通过 PCIe xdma 将分类结果返回给主机端。PCIe xdma 的理论最大传输速率为 8GT/s，本文实际传输速率为 7.8GB/s；FPGA DDR 的访存带宽经过测试为 6.4GB/s。图 5.2 为验证平台的搭建示意图，图 5.3 为实际验证平台。

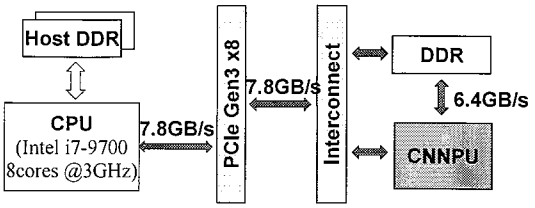

图 5.2 验证平台示意图

Figure 5.2 Schematic Diagram of Verification Platform

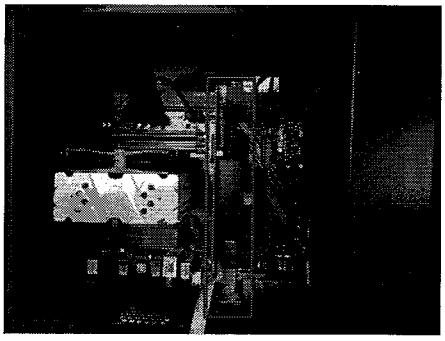

图 5.3 实际验证硬件平台

Figure 5.3 Actual verification of the hardware platform

一般的卷积神经网络加速器的设计流程为首先需要在 GPU 上训练网络，本文采用 tensorflow 框架，将精度最高的模型保存并导出，然后使用 C 或者 python 等高级语言对模型文件进行定点量化，最终得到推理阶段所需的参数文件，之后将此部署到 FPGA，本文所设计的加速器只进行推理阶段的加速。

5.2 实验结果

如前文所介绍，FPGA 最终工作在 100MHz 的时钟频率下。表 5.1 列出了 FPGA 资源占用情况。由于卷积通路使用的阵列由 64×32 个乘法阵列和 63×32 加法阵列实现所以占用大量 DSP 资源，使用率为 41.39%。另外中间有许多片上缓存，FIFO，占用许多片上 RAM 和 Block RAM。其余运算逻辑采用 LUT，占用 31.48%。

表 5.1 加速器物理资源占用情况

Table 5.1 Accelerator physical resource occupancy

	Used	Available	Utilization
LUTs	208819	663360	31.48%
LUTRAM	16977	293760	5.7 8%
FF	142196	1326720	10.72%
BRAM	595	2160	27.55%
DSP	2285	5520	41.39%

加速器各部分的功耗见表 5.2。搭建的硬件工程的总功耗为 5.271W，主要功耗来源分为三部分：其一为 PCIe xdma，消耗 2.578W，占比总功耗的 34%；其二为 DDR 控制器 MIG，消耗 1.43W，占比总功耗的 19%；其三为本文设计的加速器 CNNPU，消耗 1.107W，占比总功耗 15%。

表 5.2　加速器各部分功耗

Table 5.2 Power consumption of each part of the accelerator

On-Chip	Power(W)
Clock	0.184
Signals	0.354
Data	0.346
Logic	0.337
BRAM	0.213
DSP	0.019
MMCM	<0.001
PLL	<0.001
PCIe	<0.001
IO	<0.001
Total	1.107

图 5.4 为所搭建硬件工程经过综合以及布局布线后的版图。如图所示，大概占用一半的资源，其中 MIG 和 PCIe xdma 的布局版图由红框标出：左边红框为 MIG，右边红框为 xdma。由于 CNNPU 的使用大量 DSP 和 RAM，无法做到整块成规则布局，不易标记。

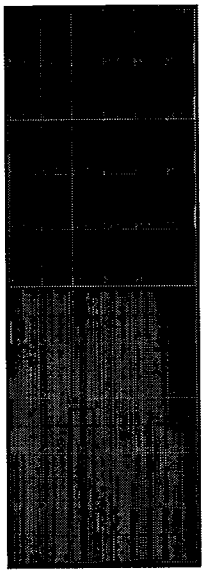

图 5.4　FPGA 布局布线后的版图

Figure 5.4 Layout after FPGA placement and routing

5.3 LeNet 结果分析与对比

5.3.1 精度对比

LeNet 网络一般采用 MNIST 进行测试。本文采用 MNIST 分别对 PC 和 FPGA 两种不同的实现方式进行测试以对比，测试集共 10000 张样本。FPGA 测试完 10000 张数据样本共需 58.58s，准确率为 99.25%。表 5.3 为 PC 和 FPGA 结果对比，PC 的准确率为 99.57%，可以看到有 PC 比 FPGA 实现精度高 0.32%，这主要是由于数据量化位宽所造成的精度损失。

表 5.3 LeNet 准确率对比

Table 5.3 Lenet accuracy comparison

	PC	FPGA
精度	99.57%	99.25%

5.3.2 结果分析

LeNet 的网络结构在第二章已经介绍，为了更好的对加速器进行分析，我们需要对网络的每层进行性能分析，得出卷积运算时阵列的利用率。阵列的利用率一方面和数据流的调度方式有关，另一方面就是网络结构本身的差异。比如有的网络输入通道多，在本加速器就会有很好的性能，而有的网络输入通道很小，就会影响加速器阵列的利用率。图 5.5 为 LeNet 各层运行所需的时间，显而易见，第一个全连接层的耗时最大，这也和它数据量最多呈正相关。

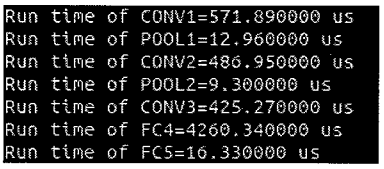

图 5.5 LeNet 各层的运行时间

Figure 5.5 Running time of each layer,LeNet

表 5.4 为 LeNet 中各卷积层运算的 MAC 利用率分析表。可以看到第一个卷积层和全连接层的 MAC 利用率很低，而中间的卷积层阵列利用率还可以。这是因为第一个卷积层是因为输入图像为灰度，为单通道，对于 64 的并行输入通道阵列，利用率大概 1/64。另外后两个全连接层 MAC 利用率低一方面时因为权重数据量巨大，另一方面时也是因为全连接层的输出为 1×1，导致阵列利用率迅速掉了下来。而中间层的输入特征为前层的结果，通道数会变多，MAC 的利用率也会提上来。

表 5.4　LeNet 卷积各层 MAC 里利用率

Table 5.4 LeNet convolves the utilization of MAC at each layer

层类型	运算量(M)	运行时间	MAC 利用率
CONV1	0.6272	571.89	0.54%
CONV2	10.0352	486.96	10.06%
CONV3	20.0704	425.27	23.04%
FC4	12.845	4260.34	1.47%
FC5	0.01024	16.33	0.306%
总计	0.087GOP	5783.05	—

5.3.3　吞吐量及能量效率对比

表 5.5 对比了其它具有突出研究成果的文献与本工作在运行 LeNet 的性能对比。由前一节介绍，本文所设计的加速器的功耗为 1.107W。

本文所设计的加速器在运行 LeNet 网络时的吞吐量为约 15.044GOPS，能效比为约 13.59GOPS/W，比去年发表的研究成果文献[3]的吞吐量提高约 16.36%，能效比提高约 34.23%。

表 5.5　加速器运行 LeNet 与其它研究成果性能对比

Table 5.5 Comparison of performance of accelerator running LeNet

Impl.	文献[1]	文献[2]	文献[3]	本工作
FPGA 型号	VC5VLX110T	ZYNQ7020	xc7k70t-1fbg676	Xilinx KCU115
工作频率(MHz)	75	200	50	100
数据精度	fixed 32bit	fixed 8-16bit	fixed 8bit	fixed 16bit
Latency/Image (ms)	—	—	—	5.783
吞吐量(GOPS)	1.83	10.07	12.928	15.044
能效比(GOPS/W)	1.008	3.596	10.124	13.59

5.4　VGG 结果分析与对比

5.4.1　结果分析

为了测试加速器对大型卷积神经网络的支持，我们直接选取数据量最大，结构最复杂的网络——VGGNet，对加速器进行测试验证，采用 ILSVR2012 测试数据集。

同样，FPGA 的部署模型来自 tensorflow 模型的转化。并通过 C 模型初步得出大致性能。图 5.6 为 FPGA 运行后对图片进行预测的结果。

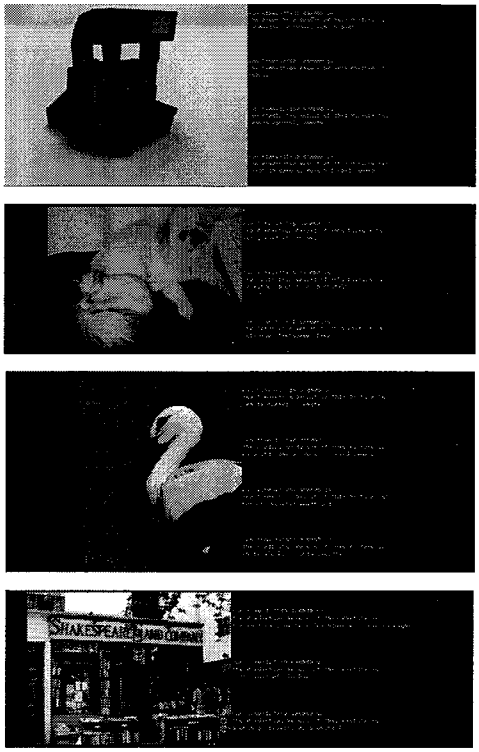

图 5.6　VGG-16 的 FPGA 识别结果

Figure 5.6 FPGA recognition result,VGG-16

VGG-16 的网络结构在第二章已经介绍，为了更好的对加速器进行分析，我们需要对网络的每层进行性能分析，得出每层卷积运算时阵列的利用率。图 5.7 为 VGG 网络各层运行所需的时间，显而易见，第一个全连接层的耗时最大，这也是由于数据量大，受访存带宽的牵制，使得运行时间较长。

```
Run time of CONV1_1=9084.020000 us
Run time of CONV1_2=9131.670000 us
Run time of POOL1   =1031.910000 us
Run time of CONV2_1=4535.190000 us
Run time of CONV2_2=9098.260000 us
Run time of POOL2   =519.490000 us
Run time of CONV3_1=4579.830000 us
Run time of CONV3_2=9127.630000 us
Run time of CONV3_3=9142.820000 us
Run time of POOL3   =261.570000 us
Run time of CONV4_1=4707.870000 us
Run time of CONV4_2=9351.380000 us
Run time of CONV4_3=9353.910000 us
Run time of POOL4   =134.330000 us
Run time of CONV5_1=2683.450000 us
Run time of CONV5_2=2683.450000 us
Run time of CONV5_3=2684.080000 us
Run time of POOL5   =36.460000 us
Run time of fc6     =33772.500000 us
Run time of fc7     =5508.070000 us
Run time of fc8     =1358.430000 us
The Prediction Result of This Picture is:
projectile, missile
```

图 5.7　VGG-16 各层的运行时间

Figure 5.7 Running time of each layer,VGG-16

表 5.6 为 VGG-16 中各卷积层运算的 MAC 利用率分析表。可以看到第一个卷积层和全连接层的 MAC 利用率很低，而中间的卷积层阵列利用率基本可以满载。这是因为第一个卷积层是输入图像为 RGB 三通道，对于 64 的并行输入通道

阵列，利用率大概只有 3/64。全连接层的 MAC 利用率低主要还是因为权重数据量巨大以及输出为 1×1，导致 MAC 利用率仅有 1/64，使得阵列利用率迅速掉了下来。而中间层的输入特征为前层的结果，通道数会变多，MAC 的利用率就会跑满，达到 100%。

表 5.6　VGG-16 卷积各层 MAC 利用率

Table 5.6 VGG-16 convolves the utilization of MAC at each layer

层类型	运算量(M)	运行时间(us)	MAC 利用率
CONV1_1	86.704	9084.02	4.66%
CONV1_2	1849.688	9131.67	98.9%
CONV2_1	924.844	4535.19	99.57%
CONV2_2	1849.688	9098.26	99.27%
CONV3_1	924.844	4579.83	98.6%
CONV3_2	1849.688	9127.63	98.9%
CONV3_3	1849.688	9142.82	98.78%
CONV4_1	924.844	4707.87	95.92%
CONV4_2	1849.688	9351.38	96.58%
CONV4_3	1849.688	9353.91	96.56%
CONV5_1	462.422	2683.45	84.14%
CONV5_2	462.422	2683.45	84.14%
CONV5_3	462.422	2684.08	84.12%
FC6	102.76	33772.5	1.49%
FC7	16.7772	5506.07	1.49%
FC8	4.096	1356.48	1.47%
总计	30.95GOP	128784.37	—

5.4.2　吞吐量以及能量效率对比

表 5.7 对比了其它具有突出研究成果的文献与本工作在运行 VGG-16 的性能对比。本文所设计的加速器的峰值算力为 204.8GMACS，即 409.6GOPS。本文主要进行吞吐量以及能量效率的比较。由前一节介绍，本文所设计的加速器的功耗为 1.107W。

本文所设计的加速器在运行 VGG-16 网络时的吞吐量为约 242.75GOPS，能效比为约 219.29GOPS/W，较文献[4]的吞吐量提高约 19.05%，较文献[5]提高约 106.07%，较文献[6]提高约 77.2%。

表 5.7 加速器运行 VGG-16 与其它研究成果性能对比

Table 5.7 Comparison of performance of accelerator running VGG-16

Impl.	文献[4]	文献[5]	文献[6]	本工作
FPGA 型号	Virtex-7-VX690t	Altera Stratix-V	ZYNQ	Xilinx KCU115
工作频率(MHz)	150	120	150	100
# Operation(GOP)	30.95	30.95	30.95	30.95
# of Parameters(M)	138.3	138.3	138.3	138.3
数据精度	fixed 16bit	fixed 8-16bit	fixed 16bit	fixed 16bit
Logic Utilization	—	153K(25%)	183K(84%)	208819
DSP Utilization	—	727(37%)	780(87%)	2285
BRAM Utilization	—	1500(58%)	486(89%)	595
Latency/Image(ms)	151.8	262.9	224.6	127.5
Throughput(GOPS)	203.9	117.8	136.97	242.75

5.4.3 Roofline 模型分析

在第一章已经介绍了 The Roofline Model，我们通过运行 VGG 网络所得到的数据对本加速器进行性能评估。本设计 FPGA 的 DDR 访存带宽为 6.4，加速器的峰值算力为 204.8GMAC，可以得到 Roofline 模型拐点的横坐标为 32MAC/Byte。

由图 5.8 可以看到，第一层卷积层和全连接层受到访存带宽的限制，处于 Memory Bound，其它层均处于 Compute Bound，不受访存带宽的限制。这也和硬件实现结果相一致。

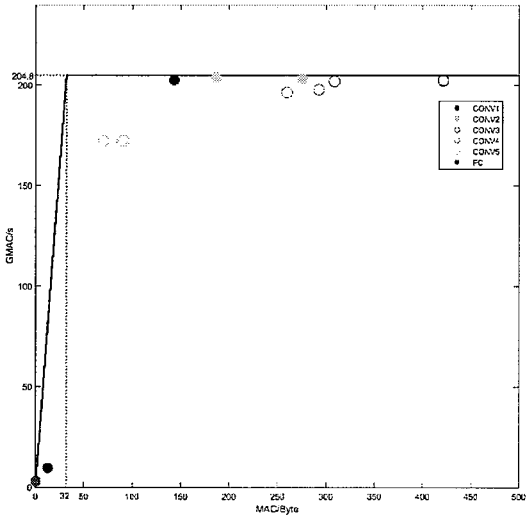

图 5.8 Roofline 模型对性能评估

Figure 5.8 Performance evaluation use The Roofline Model

Roofline 数据详见附录.

5.5　ASIC 综合及性能评估

由于 FPGA 中时钟产生为固定模块，并且处于固定的位置，这样就很难灵活的约束时钟，造成 FPGA 的时钟频率一般较低，很难收敛。而 ASIC 有天然的优势，通过模块的摆放，每条路径优化时序，综合中也有 timing、retime 等优化操作，使得 ASIC 的可以在很高的时钟频率下工作。

对本文所设计的可配置卷积神经网络加速器在 SMIC 55nm 的工艺节点进行逻辑综合，使用的综合工具为 synopsys 公司的 DesignCompiler2016。加速器可以运行在 300MHz 的时钟频率下，在 1.2v 的电源电压下综合后的功耗为 1.7W。

表 5.8 为本加速器在 SMIC55nm 的工艺节点下做能达到的性能。从表可以看到，峰值算力可以达到约 1.2TOPS，能效比可达 722.8GOPS/W。

表 5.8　加速器在 SMIC 55nm 工艺下的性能参数

Table 5.8 Performance parameters of the accelerator under the SMIC 55nm process

工艺	SMIC 55nm
电源电压	1.2v
标称频率	300MHz
峰值算力	1.2TOPS
精度	fixed 16bit
功耗	1.7W
能量效率	722.8GOPS/W

5.6　本章小结

本章对加速器进行验证。首先介绍硬件平台的搭建。然后将 LeNet 和 VGG-16 作为目标网络从精度、吞吐量、能效比等方面对加速器进行了性能评估。

以 LeNet 作为目标网络时，我们首先阐述了 FPGA 资源的占用情况以及功耗分布。其次验证在 FPGA 下的运行精度，实验证明 FPGA 由于量化位宽的缘故比软件精度低 0.32%。接着我们分别计算了 LeNet 各层的运行时间，并以此得出每层运算时阵列单元的的利用率。然后我们对加速器的吞吐量和能效比进行了评估。最后我们将本架构与三项别人的工作进行对比，本架构的吞吐量 15.044GOPS 和能量效率 13.59GOPS/W，显著优于其它三项对比工作。

以 VGG-16 作为目标网络时，我们首先介绍了 FPGA 的运行结果。接着我们分别计算了 LeNet 各层的运行时间，并以此得出每层运算时阵列单元的的利用率，可以明显的看到第一个卷积层与最后的全连接层的 MAC 利用率很低，这是因为带宽的限制。然后我们对加速器的吞吐量和能效比进行了评估。并与三项别人的工作进行对比，本架构的吞吐量为 242.75GOPS 和能量效率为 219.29GOPS/W，显著优于其它三项对比工作。最后对 VGG-16 的数据结果进行了 Roofline 模型分析，验证了 FPGA 的运行结果。

最后对本文设计的加速器在 SMIC 55nm 的工艺下进行逻辑综合，实验证明，在 ASIC 下可以达到 1.2T 的峰值算力，722.8GOPS/W 的能效比。

第6章　总结与展望

6.1　全文总结

随着人工智能领域的蓬勃发展，随之而来的超大数据量的运算需求对硬件的更新和提升提出了更大的挑战。神经网络在人工智能充当着不可比拟的角色，也让人工智能进入了重要的发展阶段。卷积神经网路为神经网络的一种，随着目标识别，人脸识别的应用逐渐司空见惯，卷积神经网络真正进入了人们的视野。

卷积神经网络的训练过程是通过反向传播来进行前向数据的修正，本质上，反向传播是导致网络学习的原因。这是一个反馈过程，网络产生的输出与它本来应该产生的输出进行比较修正。为了提高准确度，网络变得越来越大，这随之而来的就会大幅提高计算成本以及存储需求，导致通用计算平台面临巨大的吞吐、能效挑战。至此，卷积神经网络加速器逐渐成为学术界和工业界的研究热点。

然而目前的设计仍存在问题。研究者通过提高频率或者扩大阵列单元来提升算力，这不是最优的加速方案，比如阵列单元个数过多会导致时序很难收敛，应该从适配算力和带宽的方面入手做整体的架构优化。针对这类问题，本文提出一种具有可灵活配置、高并行高吞吐、高能效比的机器视觉领域的神经网络加速器。

本文首先对卷积神经网络进行详细的介绍，对每层运算过程、运算方式数据规模进行详细阐述；然后对相关的研究工作进行总结。接着开始详细的介绍本文的具体工作，从具体的算法优化到最后的硬件实现一一做阐述。最后对本文所设计的加速器在 FPGA 上进行验证，同时在 SMIC 55nm 的工艺下进行综合，均达到不错的效果。本文具体工作以及创新点总结如下：

1. 本文针对卷积层的大数据量问题，采用卷积切分的方式，将卷积操作分为若干子运算，通过输入输出通道并行的计算方式，提高计算效率；另外，利用切割后的数据在通道方向做数据复用，减少数据的重复搬运；采用顺序数据存储方式不仅可以提高 DMA 工作效率又可以提高阵列单元工作效率。

2. 本文对传统的池化层的滑窗运算进行了分析，得出传统的滑窗池化运算采用二维运算，通过移位的方式进行比较运算，需要用到较多的移位寄存器，而且二维运算的控制逻辑复杂，针对这一问题，采用先横向池化后纵向池化的算法，少用了大量的寄存器，且控制逻辑要简单很多。

3. 本文对加速器的核心阵列单元进行了整体的优化，采用在阵列内部加权重寄存器堆的方式来做乒乓操作，减少数据搬运时延。

4. 本文着重分析了常见的加速器所面临的带宽限制问题，采用先进的数据存储方案，利用 DMA 的突发传输搬运，不仅可以达到高效传输的目的，而且可以大大提高 DMA 的工作效率。

本文提出一种可配置卷积神经网络加速器。经实验得出，以 VGG-16 为目标网络，基于 FPGA 在 100MHz 的时钟频率下，峰值算力 409.6GOPS，功耗 1.107W，可以达到 242.75GOPS 的吞吐量，219.29GOPS/W 的能效比。在 SMIC 55nm 的工艺下，时钟频率可以达到 300MHz，峰值算力约 1.2T，综合后的功耗为 1.7W，达到 722.8GOPS/W 的能效比。

6.2 未来工作

本文提出的高能效比可配置卷积神经网络加速器虽然已经对算法模型和整体架构做了大量的优化，但仍然存在许多不足的地方，需要未来在本文设计的加速器的基础上继续做大量的优化。针对高能效比可配置卷积神经网络加速器的下一步的工作可以围绕以下几点开展：

1. 设计更优的访存模块或者数据存储方案，提高访存带宽。

2. 对量化数据格式进一步进行优化。量化方案一直是硬件与软件设计的核心区别，位宽的大小直接影响着系统的性能。将权重和神经元的位宽量化过大虽然可以大幅提高精度，但导致性能跟不上；将权重和神经元的位宽量化过小，又会带来精度损失的后果。在推理阶段的网络中进行每层量化，对每层的输出特征加量化层是一种不错的量化方案。

3. 支持更多种类的神经网络层。比如对紧凑型神经网络的优化与支持，进而支持更多的神经网络。

附录 VGG-16 的 Roofline 数据表

Type	Hin	Win	CHin	Ky	Kx	Sx	Sy	Px	Py	Hout	Wout	CHout	输入特征量	权重量	输出特征量	运算量 (GMAC)	访存量 (GByte)	运算量/访存量 (MAC/Byte)	受限于访存 (Y/N)	理论运算时间(us)	FPGA运行时间 (us)	实际性能 GMAC/s
CONV	224	224	3	3	3	1	1	1	1	224	224	64	150528	1728	3211264	0.086704	0.00672704	12.88889735	Y	1051.1	9084.02	9.544687044
CONV	224	224	64	3	3	1	1	1	1	224	224	64	3211264	36864	3211264	1.849688	0.01291878	143.1781864	N	9031.68	9131.67	202.5574801
POOL	224	224	64	2	2	2	2	0	0	112	112	64	3211264	0	802816	0	0.00802816	0	Y	1254.4	1031.91	0
CONV	112	112	64	3	3	1	1	1	1	112	112	128	802816	73728	1605632	0.924844	0.00496435	186.2970297	N	4515.84	4535.19	203.9261932
CONV	112	112	128	3	3	1	1	1	1	112	112	128	1605632	147456	1605632	1.849688	0.00671744	275.3560976	N	9031.68	9098.26	203.3012976
POOL	112	112	128	2	2	2	2	0	0	56	56	128	1605632	0	401408	0	0.00401408	0	Y	627.2	519.49	0
CONV	56	56	128	3	3	1	1	1	1	56	56	256	401408	294912	802816	0.924844	0.00299827	308.4590164	N	4515.84	4579.83	201.9385069
CONV	56	56	256	3	3	1	1	1	1	56	56	256	802816	589824	802816	1.849688	0.00439091	421.2537313	N	9031.68	9127.63	202.6471345
CONV	56	56	256	3	3	1	1	1	1	56	56	256	802816	589824	802816	1.849688	0.00439091	421.2537313	N	9031.68	9142.82	202.3104539
POOL	56	56	256	2	2	2	2	0	0	28	28	256	802816	0	200704	0	0.00200704	0	Y	313.6	261.57	0
CONV	28	28	256	3	3	1	1	1	1	28	28	512	200704	1179648	401408	0.924844	0.00356352	259.5310345	N	4515.84	4707.87	196.4463828
CONV	28	28	512	3	3	1	1	1	1	28	28	512	401408	2359296	401408	1.849688	0.00632422	292.4766839	N	9031.68	9351.38	197.7984067
CONV	28	28	512	3	3	1	1	1	1	28	28	512	401408	2359296	401408	1.849688	0.00632422	292.4766839	N	9031.68	9353.91	197.7449071
POOL	28	28	512	2	2	2	2	0	0	14	14	512	401408	0	100352	0	0.00100352	0	Y	156.8	134.33	0
CONV	14	14	512	3	3	1	1	1	1	14	14	512	100352	2359296	100352	0.462422	0.00512	90.3168	N	2257.92	2683.45	172.3236938
CONV	14	14	512	3	3	1	1	1	1	14	14	512	100352	2359296	100352	0.462422	0.00512	90.3168	N	2257.92	2683.45	172.3236938
CONV	14	14	512	3	3	1	1	1	1	14	14	512	100352	2359296	100352	0.462422	0.00512	90.3168	N	2257.92	2684.08	172.2832464
POOL	14	14	512	2	2	2	2	0	0	7	7	512	100352	0	25088	0	0.00025088	0	Y	39.2	38.46	0
FC	7	7	512							1	1	4096	25088	102760448	4096	0.10276	0.20557926	0.49985804	Y	32121.76	33772.5	3.042725531
FC	1	1	4096							1	1	4096	4096	16777216	4096	0.016777	0.03357082	0.499755979	Y	5245.44	5506.07	3.047040085
FC	1	1	4096							1	1	1000	4096	4096000	1000	0.004096	0.00820219	0.499378703	Y	1281.593	1356.48	3.01958009