1绪论（简述研究背景、意义等。简述FPGA的发展，CNN的发展，结构相似性的启发）

1.1研究背景和意义

随着集成电路工艺和技术的飞速发展，各种高性能、高密度的现场可编程门阵列芯片也发展迅速，其芯片中海量的可编程逻辑、运算资源和芯片本身的阵列结构，让这种产品具备灵活的并行运算能力，从而适合对某些计算并行性较高的模型或者算法在实现上进行优化加速。如今，集成电路生产工艺持续进步，借助最新的14纳米三栅极工艺，各大FPGA厂商的芯片产品快速迭代，而且不断针对不同应用，如车载电子系统等，提供了比较成熟的解决方案。如ALTERA公司通过FPGA提供的车载娱乐系统在视频输入、显示接口、图形加速、射频信号处理等方面都具有巨大的优势。[[[1]](#endnote-1)] 同时，FPGA的开发工具也向着提高与软件开发的兼容性而进步。Xilinx公司提出的高层次综合工具和All Programmable抽象化计划可以加速复杂的FPGA和SoC的开发[[2]](#endnote-2)【车载电子解决方案的图ALTERA】

人工神经网络，特别是卷积神经网络，是一种受到生物视觉神经网络启发而发展起来的计算机视觉模型，它通过模仿生物（人类）大脑通过视觉识别物体的方法，设计多层神经元模型，让计算机视觉识别系统能够提高识别率，并达到较强的抗位移和形变干扰的能力。[[3]](#endnote-3)卷积神经网络通过对大量有标签样本数据的学习，而实现对图像中各种物体的识别。卷积神经网络这种计算机视觉模型的一个特点是其中很多层内的运算相互独立，模型有较高的并行特性，适合通过使用FPGA，充分挖掘卷积神经网络的并行特性，从而以较低的功耗，加速算法的运行，优化网络的实现。

比较成熟的卷积神经网络根据规模可以大致分为两类，大规模的网络用于识别较为庞大的图片数据库中图片的内容，比如某种在ImageNet数据库中做识别测试的网络能够识别数据库中1000类共一百二十万张图片，其前五选错误率约为17%。另一种规模较小的网络，如LeNet-5有更为明确的应用意义，专门用于识别手写数字图片。较高的正确率使得这种网络可以实际应用在现实生活中，完成某些实际任务。

特定功能的卷积神经网络应用在智能家居设备或者手持智能设备上可以提供一种全新的交互体验，实现更直观自然的交互方式。然而，上述智能设备一方面无法提供足够的功率，以满足用高性能并行计算平台（如GPU）实现的大规模网络；另一方面，这些平台所搭载的普通处理器通过串行方式实现小规模网络仍然需要耗费较多的时间。因此，使用FPGA实现的卷积神经网络芯片，能够充分发挥并行特性，高校地独立处理卷积神经网络的全部运算，从硬件角度，实现卷积神经网络在上述智能设备中的普遍应用，从而实现全新的交互体验。

1.2研究现状

本节将介绍卷积神经网络以及其硬件实现方面的研究现状。

自从1998年的LeNet-5[[4]](#endnote-4)被成功应用在交易支票读取系统中以来，越来越多的卷积神经网络被应用到实际问题的解决中来。微软的手写识别系统和光学字符识别系统中应用了Simard和 Chillapilla等人的三种卷积神经网络[[5]](#endnote-5) 啊[[6]](#endnote-6)。谷歌街景中的人脸检测和车牌识别应用了Frome等人提出的神经网络[[7]](#endnote-7)。而法国电信视频会议中的人脸识别系统应用到了Garcia等人提出的神经网络[[8]](#endnote-8)

早在20世纪90年代，贝尔实验室就开发了人工神经网络芯片ANNA，该芯片通过数-模混合架构，实现了同时完成64个8\*8卷积的任务。这时对神经网络的硬件实现相对初级，运算能力较低，精度也无法达到较高水平，其状态表示用4bit量化，权重用6bit量化[[9]](#endnote-9)。Antony等人[[10]](#endnote-10)通过FPGA实现了多层感知器网络，并详细分析了定点数格式表示和浮点数格式表示以及串行实现和并行实现对硬件实现的神经网络的性能的影响。详细介绍一种用Xilinx Virtex5 FPGA实现的CNN向量处理器单元。该处理单元采用流水线式的缓存乘、加模块实现二维卷积操作，通过并行排列多个处理单元组成CNN协处理器，配合与上位机共用的片外存储单元块，FPGA可以完成CNN多个完整卷积层的运算，每层运算包含一个二维卷积运算，非线性激活函数运算和子采样运算。该系统可以完成对640\*480分辨率的图像中的人脸识别工作。

1.3本文的研究方向

【核心思路，目前CNN主要用于大规模的计算机视觉应用，如图片内容识别。涉及到的具体问题通常是输入海量的现实图片数据，或者从现实视频中提取的图片帧，利用训练好的神经网络来识别图片中的物体，从而达到计算机理解现实场景的目的。对于此类应用，对于科研机构来说，大多数可以使用大规模的服务器、大量的显卡等先进设备来实现卷积神经网络的高度并行化训练和识别。其可行性不会受到太多挑战，且不太需要考虑功耗等问题。另一方面，对于手写数字识别这类简单应用，其应用场合是银行等金融机构，其平台多为普通的PC机，对于LeNet-5这样的小规模网络，普通PC的运算能力完全可以在耗费较少运算资源的条件下，较快地完成任务。】

【然而对于智能电视、机顶盒等家用智能产品或者手机、平板等手持智能设备，它们对产品体积和整体功耗都有比较苛刻的要求，而且它们需要完成的任务也并不像手写数字识别那样简单。因此，它们对具备一定并行运算能力，并且功耗和体积都较小的专用神经网络运算芯片有较高的需求】

【本文将针对家居智能平台或手持智能设备对新型人机交互方式的需求，对静态手势识别这一特定任务，设计类似LeNet-5规模的网络，并将整个网络实现在FPGA芯片中，从而实现快速、低功耗的交互应用。并提出面向个人应用场景的卷积神经网络应用芯片的概念，从应用和实现角度，拓展人工神经网络和FPGA相结合的现实意义。】【如图所示的是本文的研究思路示意图】

本文的结构按照如下思路组织：第一章，绪论：介绍FPGA与卷积神经网络的背景，两者相结合的意义和一些现有的应用情况。同时介绍本文设计的基于FPGA的卷积神经网络在现实中的应用价值。第二章，详细介绍FPGA的发展和应用；卷积神经网络的发展和应用，分析并挖掘卷积神经网络的并行特性，以及该特性与FPGA并行架构的结合点。第三章，详细介绍FPGA实现的卷积神经网络的技术要点，各个模块的设计方案，详细设计思路和设计过程。同时，对每一模块的实验和优化工作做了详细介绍。第五章，结论：总结本文设计的基于FPGA的卷积神经网络，同时提出对该网络的进一步设想。

2.1FPGA的基本情况（对FPGA做综合性介绍）

2.1.1FPGA的产生和发展（详细介绍FPGA的产生背景、发展历程等）

FPGA技术是从可编程只读存储器和可编程逻辑器件这两项技术中发展出来的。可编程只读存储器和可编程逻辑器件这两项技术都要求产品能成批地在工厂中或者应用现场实现可编程功能，而可编程逻辑需要让逻辑门之间达成可编程的硬件连线。早在二十世纪80年代，可编程逻辑阵列、逻辑门、逻辑块等相关技术概念的专利就已经出现。Steve Casselman就曾开发过60万可编程门的计算器件并成功获得专利。

成立于1983年的Altera公司在1984年就生产了第一块可编程逻辑器件，该器件通过紫外照射可擦除编程只读存储器单元来实现器件的配置。Xilinx的Ross Freeman and Bernard Vonderschmitt于1985年发明了第一块商业化的FPGA。这块名为XC2064的FPGA拥有可编程逻辑门和逻辑门之间的可编程互连线，它的可编程逻辑门组成64个可配置的逻辑块，逻辑块中包含两个三输入查找表。这样的设计结构也成为后来FPGA的主流结构，各FPGA芯片均以类似结构的逻辑块为核心单元，辅以触发器、存储器，以及更高级的数值运算单元和嵌入式处理器等。

近年来，FPGA架构发生了翻天覆地的变化。在原有可编程逻辑块和可编程互连线的基础上，Altera的Nios II将软核用FPGA实现，以完成处理器与FPGA的结合。新一代的FPGA将嵌入式微处理器和相关外围模块融合到芯片中,形成一个完整的系统——system on a programmable chip（SoPC）。这一结构的代表产品是Xilinx 的Zynq-7000 All Programmable SoC。该系统将一枚1.0GHz双核ARM Cortex-A9 MPCore处理器与FPGA芯片融合，这使得嵌入式设计师们能够更轻松地使用这款产品。这种可扩展的处理器平台让系统架构师和嵌入式软件开发者能够使用串、并行结合的处理方式来实现其日益复杂的软件设计。另外Altera Arria V FPGA也将一枚800MHz的双核ARM Cortex-A9 MPCore处理器与FPGA芯片融合。也有其他厂商的产品将处理器核以及诸如多通道模-数/数-模转换器与FPGA融合。这种高度集成的产品能够降低功耗和冗余。而且减少了元器件数量，这就会避免了采用分立FPGA和CPU可能带来的成本高，系统庞大的缺点。而且有研究表明，现代电子产品中多数问题发生在芯片与PCB之间，而不是芯片内部，因此高度的集成化可以提高系统可靠性。

2.1.2FPGA的应用情况

FPGA最原始的应用是作为PCB胶合逻辑，与复杂可编程逻辑器件竞争。随着FPGA的规模、能力和速度的提升，才开始胜任越来越庞大的任务，进而形成完整的片上系统SoC。特别是到了二十世纪九十年代，专用乘法器等DSP模块被融入FPGA架构，FPGA开始承担越来越多的数字信号处理工作。

理论上讲，FPGA可以用于任何计算领域，其内部可以集成微处理器就是最直接的证明。而FPGA的真正优势在于，针对某些特定的应用场合，它们的运算速度会非常快，因为它们本身具有大量逻辑门，以及其并行性的本质。具体的FPGA应用场合有： ASIC原型设计，计算机视觉，医学影像等。FPGA的另一重要应用趋势就是对算法的硬件加速。这种加速是指设计者可以使用FPGA来加速某一算法的某些具体步骤，或者是将这种高性能的运算模块在FPGA和通用处理器之间实现共享。

从另一个角度讲，FPGA还是较多地适用于小产量的纵向应用。因为对于小产量应用，每个可编程芯片的单位硬件成本是比生产专用芯片却无法形成市场规模时的单位硬件成本更低。而且，如今的成本与性能优势，正在不断地拓宽FPGA的应用领域。

2.2CNN的基本情况（对CNN做综合性介绍）

2.2.1CNN的产生和发展

1962年Hubel和Wiesel研究了猫视觉皮层细胞，并且提出了感受野(receptive field)的概念，1984年日本学者Fukushima在设计手写阿拉伯数字识别系统的时候，根据感受野的概念，提出了神经认知机(neocognitron)这个概念。这种神经认知机甚至能够识别形变比较严重的手写数字。福岛提出的神经认知机是一个层级结构的多层神经元网络，输入信息逐层通过网络被处理。网络中每个神经元选择性地对特定的激励模式做出反应。福岛的这个神经认知机是卷积神经网络的第一个现实网络，同时他也首次将感受野的概念应用在了人工神经网络中。近几年来卷积神经网络在许多领域不断取得进展，在语音识别、人脸识别、通用物体识别、运动分析、自然语言处理等方面均有突破。

卷积神经网络以待识别的图像直接作为输入，经过多个卷积层、子采样层和全链接层处理，得出识别结果。其中卷积层将输入的图像（这些图像可能是输入层的原始图像，也可能是前一个卷积层或子采样层处理后的特征映射图）与若干个卷积核分别做卷积，输出相应数量该层的特征映射图。【这里需要一个卷积层示意图，1-to-n&m-to-n两种】子采样层的操作一般被称为池化，该层对卷积层输出的若干个特征映射图做减采样操作。一般情况下是对输入图像分为2\*2像素的小块，每块对应的输出图像素为这一小块四个像素的和或最大值，根据取值的方法不同，分别被称为“和子采样”或“最大值子采样”。【示意图】卷积层或子采样层输出的每个特征映射图的每个像素，一般会加上一个与生成该特征映射图的卷积核相对应的的偏置，并且将每个像素带入一个激活函数，函数的输出作为该层最终输出的特征映射图的像素值。这里激活函数的选择可以是多种多样的，它们分别有各自的优缺点。如【图A】的Sigmoid函数曾被认为是神经网络的核心所在，从数学上来看，非线性的Sigmoid函数对中央区的信号增益较大，对两侧区的信号增益小，在信号的特征空间映射上，有很好的效果。从神经科学上来看，中央区酷似神经元的兴奋态，两侧区酷似神经元的抑制态，因而在神经网络学习方面，可以将重点特征推向中央区，将非重点特征推向两侧区。如【图B】的双曲正切函数形状与Sigmoid函数基本一致，只是小于零的部分会趋近于-1，适用于数据标签是-1到1的情况。如【图C】的ReLU函数对比Sigmoid之类的函数，主要变化有三点：单侧抑制；相对宽阔的兴奋边界；稀疏激活性。另外，相比于普通的Sigmoid函数，采用ReLU激活函数的神经网络应用有更快的特征学习能力。经过多个卷积层和子采样层的处理，每个特征映射图的尺寸最终会很小，而其数量会很多，这就是原始输入图像经过特征提取后得到的抽象特征。最后这些特征将被送入一个或几个全连接的传统神经网络进行分类，得到最终的识别结果。

2.2.2CNN的应用情况

以图像本身直接作为神经网络的输入，由于图像像素数量大，因此全连接的传统神经网络就需要数量非常巨大的权值参数。以100\*100的图像（共1万个像素）为例，全连接网络需要1亿个参数，使得每个神经元与输入图像的每个像素相连接。然而我们知道，图像的空间联系往往是局部的，每个神经元并不需要对图像全局做出反应，而通常只需要对图像的一小块做出反应。结合感受野的概念，假设我们只需要对图像中一个5\*5的小块做出反应，而且，每个特征映射图对输入图像做出相同的反应，也就是一个特征映射图共享一组25个权值，这样多个特征映射图（这里假设有6个）就只需要150个参数。这就是卷积神经网络中权值共享所带来的好处——大大减小了网络的规模。总结下来，结合局部感受野、权值共享以及子采样这三大主要思路，就获得了卷积神经网络的图像识别能力，并且，这种识别能力在某种程度上达到了位移、尺度、形变不变性。

正因为网络规模的缩小和对位移、尺度变化与形变的一定程度的抵抗能力，使得卷积神经网络成为目前应用最广泛的深度神经网络之一。作为最经典的手写数字识别网络之一，于20世纪九十年代由LeCun提出的LeNet-5，其识别准确率已经达到相当高的水平，已经在美国银行系统内用于识别支票上的手写数字。该网络结构示意图如下，不包括输入层的情况下，共有7层，输入32\*32像素的图像。每个卷积层卷积核大小为5\*5像素；每个子采样层做加和子采样，各个子采样窗口之间没有重叠部分，子采样结果乘以系数并加偏置；激活函数采用sigmoid函数，在子采样层之后，卷积层不设置激活函数。网络第一个卷积层输出6个特征映射图，第二个卷积层输出16个特征映射图，然而，第二个卷积层的输入是与前一层输出的6个特征映射图中的部分连接的，其连接情况可以用【表格】来表示。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 0 | √ |  |  |  | √ | √ | √ |  |  | √ | √ | √ | √ |  | √ | √ |
| 1 | √ | √ |  |  |  | √ | √ | √ |  |  | √ | √ | √ | √ |  | √ |
| 2 | √ | √ | √ |  |  |  | √ | √ | √ |  |  | √ |  | √ | √ | √ |
| 3 |  | √ | √ | √ |  |  | √ | √ | √ | √ |  |  | √ |  | √ | √ |
| 4 |  |  | √ | √ | √ |  |  | √ | √ | √ | √ |  | √ | √ |  | √ |
| 5 |  |  |  | √ | √ | √ |  |  | √ | √ | √ | √ |  | √ | √ | √ |

例如，第二个卷积层的第一个卷积核就是将前一层输出的前三个特征映射图分别卷积一次，并且将卷积结果叠加起来得到本层的第一个特征映射图。网络最 后的传统神经网络是两个全连接层。这种全连接层的输入是最后的卷积层输出的特征映射图，一般情况是将最后所有的特征映射图的每个像素顺序排列成一列，输入给全连接层。而LeNet-5最后一层卷积层稍微特殊，其最后一个卷积层输出的特征映射图都只有一个像素。网络的输出是一个10元向量，通过选取10个元素中最大或最小的一个值来确定网络识别的结果。

【另一个应用介绍：ImageNet Classification with deep convolutional neural networks一文中设计的卷积神经网络在imagenet数据库上训练了一个CNN。该网络有六千万个参数，六十五万个神经元，网络包含5个卷积层，其中的3个卷积层之后设置了最大值子采样层，网络最后有三个全连接层和一个一千类的softmax层。网络中使用的激活函数是ReLU函数。【表格详细地说明了这个网络各层的配置情况】。下面结合表格所示的各个卷积神经网络层，说明该网络的连接情况。第一层224\*224像素\*3（RGB）图像通过卷积1层（该层包含最大值子采样层）之后分为两组，每组输出48个特征映射图。第二层将前层的两组输出分别通过卷积2层（该层包含最大值子采样层），各组之内的卷积是全连接的，组之间无连接，每组输出128个特征映射图。第三层将前两组输出全连接地通过卷积3层，无视分组，卷积全连接，输出则仍然分为两组，每组192个特征映射图。第四层和第五层都分组做卷积，组内全连接，组间无连接，在第五层最后通过一个最大值子采样层，并将两组输出合并，拉成4096个元素的列向量。第六层和第七层是全连接层，不分组，每层都输出4096个元素的列向量。第八层是softmax层，输入前一层的4096个元素，输出1000个元素的分类识别结果。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 层序号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 层名称 | 卷积1 | 卷积2 | 卷积3 | 卷积4 | 卷积5 | 全连接6 | 全连接7 | 全连接8 |
| 输入数据 | 224\*224像素\*3（RGB）图像 | 48\*2个55\*55尺寸的特征映射图 | 128\*2个27\*27尺寸的特征映射图 | 192\*2个13\*13尺寸的特征映射图 | 192\*2个13\*13尺寸的特征映射图 | 4096个数据 | 4096个数据 | 4096个数据 |
| 操作1 | 卷积（核尺寸：11\*11\*3，步长：4，共96个） | 卷积（核尺寸：5\*5\*48，步长：1，共256个） | 卷积（核尺寸：3\*3\*128，步长：2，共384个） | 卷积（核尺寸：3\*3\*192，步长：1，共384个） | 卷积（核尺寸：3\*3\*192，步长：1，共256个） | 点乘 | 点乘 | 点乘（softmax） |
| 操作2 | ReLU | ReLU | ReLU | ReLU | ReLU | ReLU | ReLU | ReLU |
| 操作3 | 最大值子采样 | 最大值子采样 | 无 | 无 | 最大值子采样 | 无 | 无 | 无 |
| 操作4 | 归一化 | 归一化 | 无 | 无 | 无 | 无 | 无 | 无 |
| 中间数据 | 48\*2个55\*55尺寸的特征映射图 | 128\*2个27\*27尺寸的特征映射图 | 192\*2个13\*13尺寸的特征映射图 | 192\*2个13\*13尺寸的特征映射图 | 4096个数据 | 4096个数据 | 4096个数据 | 1000个识别结果 |

这个网络在LSVRC-2010 contest测试集上得到了前五选错误率17%的结果，这个测试集包含了一百二十万张图片，共分为一千类。】

1.3FPGA与CNN的结合

1.3.1CNN计算的并行特点

从以上的介绍我们可以总结出，CNN的计算由大量的二维图像卷积构成。分析整个网络的数据流和运算特点可以将CNN涉及的并行运算分为一下四类：位计算并行，卷积运算内的并行，同一层中各卷积运算之间的并行，各层之间的并行。文献【基于FPGA的卷积神经网络并行结构研究】中，提出了五层并行层次结构，详细区分了神经元节点之间的并行结构与层内并行结构。其神经元节点之间的并行结构主要针对网络输出层的传统全连接层中的各个节点，提出这类数量众多的节点对网络并行性有至关重要的影响。然而本文认为，网络输出层的传统全连接网络运算方式与卷积层的卷积运算有极大的相似之处，完全可以通过控制卷积器的输入，几乎不加修改地将卷积器移植到全连接层，完成全连接层的各种运算。因此在FPGA实现过程中，并不需要专门区分这一层次的并行结构，该层的并行性，可以归纳到卷积运算内的并行结构这一层次。由于卷积神经网络属于深度网络，其规模都相对较大，在现有的软硬件条件下，是不可能完全并行实现，选择合适的并行层次结构和并行规模，设计并行运算单元，才能在FPGA等嵌入式平台上实现CNN运算加速，并且得到比较优秀的性能。下面就从上述四类并行层次入手，详细讨论各类并行层次的计算特点。

（一）位计算的并行。由于FPGA是数字化的信号处理方式，运算中各个操作数都是二进制表示，通过分析各操作数本身的二进制表示，可以从位运算的角度实现并行计算。具体来讲，就是设计性能更高的数字信号处理单元，如多位全加器、乘法器等。也就是说，这个层面的并行化是数字电路底层设计的范畴。由于现代化的FPGA中大量使用嵌入式DSP单元实现高性能的多位全加和乘法，该层面的并行化不是本文讨论的重点。

（二）卷积运算内的并行。一个典型的【二维卷积运算由图】所示，卷积核扫过被卷积的图片，3\*3大小的卷积核内的每一个点，与图像中被卷积核覆盖的3\*3大小的范围内的每一个像素分别做乘法，并将9个乘法结果加起来得到一个卷积输出。由此可以看出，这就个乘法相互独立，可以同时进行，这是卷积运算内部的并行实现依据。

（三）同一层中各卷积运算之间的并行。由前文所述，在同一卷积层内，可能存在着两种卷积情况：一、同一个输入图片被不同的卷积核卷积得到不同的输出特征映射图；二、不同的输入图片（前一层输出的多个特征映射图）被相同的卷积核卷积，并将结果相叠加。这样两类层内卷积运算都是相互独立的，这里将得到一个特征映射图的整个过程看作一个卷积运算，则每个卷积运算顺序的改变，只影响中间数据缓存的形式，不影响最终结果。因此，这些卷积运算也是可以同时进行的，这便是同一层中各卷积运算之间的并行计算依据。

（四）层间的并行。一般来说，CNN是一种多层的网络结构，各层在结构上也有较大的相似性。但是，由于整个网络中的数据是层层递进地传播和处理，前后层之间的运算有很强的相关性，因此层间计算的并行性很低，基本不存在前后层并行处理的可能。然而，由于训练好的网络参数在各层之间基本没有任何相关性，层间的并行主要可以体现在系统初始化时装载网络参数的阶段，网络各层的各种参数可以同时并行加载。

1.3.2FPGA与CNN相似的阵列结构

再次审视CNN的运算结构，可以清楚地发现，CNN的运算由大量以“乘-累加”为基本单元的神经元运算组成，每个单元所做的运算操作高度重复，且同一个卷积层内的各个“乘-累加”运算相互独立，可以同时进行，不分先后。这种由大量重复基本单元组成复杂系统的结构，与FPGA中以逻辑单元阵列和可编程互连线组成复杂数字系统的结构高度相似，因此可以很直观地想到，FPGA芯片适合于实现CNN，有FPGA实现的CNN能够充分发挥FPGA芯片的高度并行特性。【绘制CNN运算单元模块图】然而，目前FPGA基本逻辑单元的结构主要由一个多输入（四输入或六输入）查找表，一个可编程寄存器和一些可编程互连线组成，用于实现简单的组合逻辑或者比特的寄存。

Xilinx公司的FPGA中的Slice比上述基本逻辑单元稍加复杂，由两个4输入的函数、进位逻辑、算术逻辑、存储逻辑和函数复用器组成。算术逻辑包括一个异或门（XORG）和一个专用与门（MULTAND），一个异或门可以使一个Slice实现2bit全加操作，专用与门用于提高乘法器的效率；进位逻辑由专用进位信号和函数复用器（MUXC）组成，用于实现快速的算术加减法操作；4输入函数发生器用于实现4输入LUT、分布式RAM或16比特移位寄存器（Virtex-5系列芯片的Slice中的两个输入函数为6输入，可以实现6输入LUT或64比特移位寄存器）；进位逻辑包括两条快速进位链，用于提高模块的处理速度。ALTERA公司的FPGA在基本逻辑单元的基础上，用16个逻辑单元配合以内部控制信号和互联同路，组成逻辑阵列，以实现稍微复杂的逻辑功能。【逻辑单元结构图】

1.3.3FPGA与CNN的结合

从上述对FPGA基本逻辑单元结构和规模的描述中，我们可以看出，一般情况下，为了保证较高的运算精度，一个CNN神经元运算单元是难以用单个逻辑单元甚至Slice或者逻辑阵列实现的。然而，当我们重新审视目前业界普遍推广的FPGA产品时不难发现，FPGA厂商为了提高其产品对大规模乘法、乘-加运算的性能，以便在数字信号处理领域有更高的竞争力，普遍将一定规模的多位乘法器以及专用的DSP单元以硬核的方式，嵌入到FPGA芯片中，构成数字信号处理单元。以ALTERA的Stratix IV为例，该系列FPGA内拥有多达1360个18\*18乘法器，其DSP单元可以完成高分辨率视频处理，数字上、下变频和多采样率滤波等众多数字信号处理功能。【DSP单元结构图】而这种高性能乘法、乘-加功能的实现，正是人工神经网络特别是卷积神经网络运算最需要的。从FPGA的内部结构看，这些内嵌的硬件乘法器仍然是以整列方式，排列在整个芯片中，能够方便地实现与片内其他逻辑资源的互联。因此，围绕着FPGA中的硬件乘法器来设计卷积神经网络的神经元和网络层次结构，规避了当前FPGA产品中单个逻辑单元或Slice资源不足的瓶颈，在运算精度和网络实现的并行度上取得了很好的折衷。

目前，高端水平【ALTERA Stratix10】的百万门级FPGA可以提供5,510,000基本逻辑单元，137Mbits的块状存储空间，以及接近两千个数字信号处理模块和接近四千个18\*19位乘法器硬核。这样规模的FPGA芯片，足以胜任以一个完整的CNN运算层（卷积层、激活函数和子采样层一起）为单位来完成硬件实现。对于LeNet-5这样规模比较小的网络，通过适当的串行结构配合，在单个芯片内高性能地实现完整的网络运算，不再是一个难以达到的要求。

然而，考虑到规模、性能、功耗等问题的折衷，应对各种限制的设计方法仍然应该被具体考虑，从而在设计过程中达到相应的指标。因此，本文的CNN各层模块设计中，均讨论多种设计方案，并比较各方案在性能、规模等方面的优劣，方便读者能根据具体设计要求，挑选设计方案。

3基于FPGA的CNN应用

3.1静态手势识别的CNN实现

3.2结合FPGA设计对具体运算的分析

4关键电路模块的设计

4.1卷积器

卷积神经网络涉及的一个核心运算是图像的卷积运算，图像卷积运算将图像处理和传统神经网络巧妙地结合起来。卷积神经网络正是借助卷积核和卷积运算实现了感受野的概念。同时，也正是借助卷积核和卷积运算，实现了卷积神经网络权值共享，减少了网络参数数量的。如【公式】所示的图像卷积运算即二维卷积运算，在数字图像处理领域有着非常广泛和成熟的应用。图像的空间域滤波处理就大量地使用到了二维卷积运算，通过调整卷积核的内容，二维卷积运算可以实现图像的高、低通滤波，边缘检测，形态学滤波等多种实用的处理。而在卷积神经网络中，多种卷积运算的本质也是对输入图像的各种空间域滤波和边缘检测，从而实现对图像数据中提取多种特征，以便网络最后的全连接层实现对图像的分类识别。

在卷积神经网络中，大量的二维卷积运算成为整个识别算法中运算最为密集的部分。二维卷积运算虽然在数学上非常简单，但是对于硬件的实现，则有非常多的内容需要考虑，才能以较高的效率完成这种简单的运算。这些需要考虑的内容包括前端数据流馈入的形式（这种形式包括数据流量，数据格式和数据流时序等），数据处理对运算时间和硬件资源的要求等。根据实际情况，具体分析上述条件，是设计高效硬件卷积器的必然要求。下面将结合本文提出的卷积神经网络应用芯片，从数据流的缓存和卷积运算两个角度详细地讨论卷积器的设计与实现。

4.1.1卷积器的实现

如【前文公式】所示，在计算一个像素点的卷积结果Ym,n时，计算单元需要获得输入像素点Xm,n以及它周围K2-1邻域的K2个数据，然后与卷积核中的K2个权值对应相乘再累加。首先考虑系统前端输入数据是以数据流形式给进的，即像素数据从前端缓存中只能每个时钟一个像素按照从左到右从上到下的顺序逐个流入，要同时获得一个卷积窗口中的K2个数据，必须对输入数据进行缓存，通过【如图】的行缓存结构，在寄存器id0~id24上面，可以获得一个大小为5\*5的卷积窗口的图像数据。不考虑图像边缘的情况下，在初始等待(K-1)\*M+K个时钟之后，卷积窗口可以每个时钟向后（从左到右，从上到下）移动一个像素的位置，从而实现每个时钟都能得到一个新的卷积窗口的全部数据。这种情况下，后续的卷积运算单元（乘-累加单元）若能在一个时钟内完成K2个乘法和一个K2路并行加法，则可以实现每个时钟一个卷积结果的数据吞吐量，同时通过简单的数据使能标志，这个卷积层模块便可以高效地完成一张特征映射图的运算。通过对权值寄存器的更新或者对该卷积层模块的复制，即可串行或者并行地计算一个输入图像的多个输出特征映射图。这样的卷积层架构对输入数据的带宽要求最低，但有着在此限制下最高的并行性能。其主要缺点在于对硬件资源的大量需求，随着卷积核尺寸的增大，资源消耗呈平方关系增长，同时，大量的并行数据流对芯片内时序同步的要求大幅提高，极大地限制了系统时钟的频率，不利于硬件并行优势的充分发挥，反而凸显了并行数据流所带来的缺陷。

为了避免上述并行数据流时序同步问题，本节提出串行-并行结合的设计思路。在计算卷积核尺寸为K2的卷积运算时，【如图】将乘法运算分成等量的K组，每组的K个乘法和加法运算通过乘-累加（MAC）单元串行完成，最后将K组乘-累加结果并列相加。次方案通过牺牲一定的并行程度，换取了硬件资源的节省和时序要求的降低，进而提升了系统时钟的频率限制，在不考虑其他要求的条件下，可以通过提高系统时钟，达到更高的性能。

在实现该串行-并行结合方案的时候，通过对卷积运算模块时钟频率的K倍频，即可保证输入、输出数据流的速度不变。而实际情况下，由于完成卷积运算后需要将乘-累加结果代入激活函数，实际倍频频率会略大于K倍。

在实际应用中，由于资源成本的更严重限制等原因，也可能需要进一步牺牲并行特性，将K2个乘法和加法运算用【如图所示】的一个乘-累加单元架构，完全串行地实现，这种情况下，就需要对卷积运算模块的时钟做大于等于K2倍的倍频。而多个输出特征映射图的计算，仍然可以通过更新权值寄存器或复制该卷积层模块来按需求实现。根据面积-速度的折衷规律，灵活地组合完全并行卷积模块、串行-并行结合卷积模块和完全串行卷积模块，便可以设计出最符合实际需求的卷积层模块，灵活按需地解决嵌入式平台上卷积神经网络芯片的设计难题。

另外，【文献】中提出的全并行卷积运算单元，【如图】，则巧妙地将缓存结构和乘-累加运算单元相结合，避免了对K2路数据做并列加法，使用K2个乘法器和K2个加法器，规避了大量并行数据流时序同步难的问题。在如今FPGA内部计算单元数量较多的普遍现实下，这成为一个非常高效的并行卷积方案。

最后，如果前端输入的数据流可以【如图】所示那样，同时K路流入，则可以将二维卷积运算转化为K个一维卷积，该情况类似与上述串行-并行结合的卷积架构，只是此时前端同时流入K路数据，不需要缓存(K-1)\*M+K个像素，既可完成一个窗口的卷积运算，也不需要对卷积运算单元的时钟进行倍频。

4.1.2卷积器的实现与验证

上一节对卷积运算单元的详细分析，已经将多个硬件架构展现出来，本节根据上述讨论，设计实现了串行-并行结合的卷积运算单元。该单元以一路数据流形式输入每像素8bit的图像数据，卷积核尺寸为5\*5，权值数据用16bit定点数表示，1位符号位，15bit小数位，用于量化(-1,1)的权值数据。输出数据为29比特定点数，1位符号位，5比特整数，23bit小数位。设计中关键点乘-累加运算单元均使用FPGA中内嵌的DSP模块完成。使用Block RAM缓存输入数据流，实现卷积窗口的提取。模块使用硬件资源情况【如表格】所示。【对比全并行结构】

根据时序分析软件给出的结果，该模块最大时钟频率约为XXMHz。对该模块时序约束如下：【输入寄存器延时】……

本节还将介绍根据实际应用需求，对上述设计的卷积运算单元的实际验证情况。本文设计的卷积神经网络输入层需要完成对28\*28像素的图片做5\*5的卷积，这是该网络中输入图像尺寸最大的卷积运算，在此框架下设计的测试单元可以按数据流形式将图像数据送入待验证的卷积运算单元，在FPGA开发板上实际验证卷积运算单元的设计结果。验证输入的是【如图】的测试图像数据，经过XXX个时钟，卷积输出结果被完全得到。【图】显示了FPGA运算结果和MATLAB运算结果的对比，可以看到两者的归一化误差为XXX，带来这一误差的主要原因是对图像数据和权值数据的定点量化。

4.2激活函数

非线性的激活函数在各类人工神经网络中被普遍应用，将非线性因素引入人工神经网络，使得人工神经网络能够适用于更多的场合，更好地解决各种实际问题，其中sigmoid激活函数被广泛应用于各种神经网络中。Sigmoid函数是一个单极性激活函数，其表达式为【公式】。

函数图像【如图】。作为一个典型的非线性函数，sigmoid函数在数字电路中的实现成本是很高的，因为函数运算中涉及一个指数运算和一个除法运算。在数字电路硬件上实现这些非线性函数的方法主要有如下几种方案：查找表实现、比特映射实现、分段线性拟合和泰勒展开近似，以及上述方法的混合实现。本节将详细说明sigmoid函数的两种数字电路设计实现方式，它们分别采用分段线性拟合与查找表相结合的方案和【复用式分段线性拟合】方案。

4.2.1激活函数的实现

观察sigmoid函数图像我们发现如下两个事实：1）函数图像关于点(0,0.5)对称，即f(x)=1-f(-x)。2）函数图像在自变量x接近0的位置比较直，近似于一条过点(0,0.5)，斜率为0.25的直线，而在自变量稍微远离0的位置弧度比较明显，当自变量再大一些的时候，函数图像比较平，接近于f(x)=1或f(x)=0。由此我们可以得出结论，可以只计算自变量大于零的部分，小于零的部分可以很容易通过对称关系得到；在函数输入不同自变量的时候，应该采取不同的实现方案，从而达到最优的性能。这就是分段线性拟合与查找表相结合设计方案的理论基础。

在函数图像中线性度较好的区域采用线性拟合，在线性度相对较低的区域采用查找表实现，可以充分节省逻辑资源，提高精度。根据图像形状，这里将自变量大于零的部分分为三段： 0<=x<1和x>=8，这两段图像接近直线，用线性拟合可以直接将自变量简单地与固定的斜率相乘得到输出或者直接得到输出；1<=x<8，这一段图像弧度明显，采用查找表实现可以避免计算多段拟合直线，耗费过多逻辑资源用于乘、加运算和比较运算。具体设计思路如下：

将输入的二进制补码表示的数据转换成原码，即求其绝对值。

用两个比较器，将输入分为三类，即上述三个分段。

将绝对值数据乘以固定斜率，同时也将绝对值数据输入查找表，得到两个候选结果。

根据前一步的分类（分段）结果，选择候选结果作为中间结果。

用公式f(x)=1-f(-x)换算中间结果，得到第二步候选结果。

根据原始数据的符号选择最终的输出结果。

该原理的伪代码和设计框图则如【图】所示。

对于复用式分段线性拟合，因为段拟合都是使用二维直线，二维直线可以用一个乘法和一个有符号加法来实现，若在每段拟合运算中使用同一个硬件和相应不同的斜率、截距参数，则可以只用一个乘法器、一个加法器、一个多路选择器、一个编码器和若干比较器实现，实现精度要求从两方面影响资源使用情况：一，输入、输出的量化位数影响每一个运算器的规模；二，分段数量只影响多路选择器、编码器的规模和比较器的数量。分段数和各运算器数量关系【如表格所示】。该设计【模块框图如下】。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 分段数 | 多路选择器输入个数 | 多路选择器选择端位数个数 | 编码器输入比特数 | 编码器输出比特数 | 比较器个数 |
| N | N | log2(N) | N | log2(N) | N |

4.2.2激活函数实现结果的验证与分析

数字电路硬件实现非线性函数时需要考虑的一个非常重要的问题就是实现精度。对于卷积神经网络的应用来说，实现精度关系到识别正确性问题，也就是网络性能。精度太低必然影响网络的识别正确率，而过高的精度会增大硬件资源的耗费，同时也会提高系统的功耗。因此，在设计硬件电路之前，对实现误差做理论分析就非常必要。

函数实现中，若不考虑具体的实现细节，可以把该硬件系统看成对函数自变量和函数值的定点数量化。直观地可以将这个量化过程理解为将函数图像画入【如图的】网格中，用网格中离图像最接近的网格交点替代真实的函数值，且根据函数定义，要保证网格的每条纵格线上只取一个网格交点作为函数值的近似，这个过程即为自变量和函数值的量化。

前文所述的非线性函数的多种不同实现方案，便是用不同的方法来寻找网格中邻近真实函数曲线的网格交点来近似表达函数值，以实现量化的过程。【如局部细节图】所示，我们可以直观地看到这种量化过程是如何带来误差的，由于用网格交点近似实际函数值，我们可以看到，在横轴[i,i+k)区间上，真是的函数值随着自变量的增大而略微增大，但是由于使用了网格交点来近似真实的函数值，在整个输入区间，无论自变量具体是多少，都被量化为数值i，而输出都被量化为Di。因此这个区间里就有两方面误差：一），输入无论是多少，都被当成输入是i；二），输入即使恰好就是i，输出也不是真实的f(i)，而是其近似值Di。

通过上述分析，我们了解了误差的具体来源。现在，我们对分段线性拟合与查找表结合的设计方案进行MATLAB仿真和FPGA开发板的实际验证，分析在实际实现的过程中，误差的具体大小和表现形式。对分段线性拟合与查找表相结合的方案，MATLAB仿真从理论上揭示了误差的表现形式：在线性拟合段，误差由真实函数值和拟合直线之间的位置差异带来；在查找表段，误差由查找表对函数自变量和函数值的量化误差带来。在FPGA实现中，输入数据和输出都是16位定点数量化，输入数据格式为S3Q12，表示-8到8的数值，输出数据格式为Q16，表示0到1的数值。

将分段线性拟合与查找表结合的设计方案以verilogHDL建模，综合后资源使用情况如【表格】所示，该设计使用极少的逻辑资源，存储资源与精度需求相关。按照如下要求，进行时序约束：……。时序分析结果，关键路径XXns，出现在XX模块，系统最高时钟频率XXXMHz。

4.3子采样器

4.3.1子采用器的实现

4.3.2子采用器的验证

4.4设计总结

5结论

6参考文献

7致谢

1. [] 【参考：https://www.altera.com.cn/solutions/industry/automotive/applications/infotainment/aut-info.html】 [↑](#endnote-ref-1)
2. 【参考：http://china.xilinx.com/products/design-tools/all-programmable-abstractions.html】 [↑](#endnote-ref-2)
3. 【参考：神经认知机，CNN起源】 [↑](#endnote-ref-3)
4. 【参考：LeNet-5论文】 [↑](#endnote-ref-4)
5. 【Simard P.Y., Steinkraus D., Platt J.C. Best practices for convolutional neural networks applied to visual document analysis[C]. Proceedings of the seventh international conference on document analysis and recognition, 2003:958-962】 [↑](#endnote-ref-5)
6. 【Chellapilla K., Shilman M., Simard P. Optimally combining a cascade of classifiers[C]. Electronic Imaging, 2006:207-214】 [↑](#endnote-ref-6)
7. 【Frome A., Cheung G., Abdulkader A., et al. Large-scale privacy protection in google street view[C]. IEEE International Conference on Computer Vision, 2009:2373-2380】 [↑](#endnote-ref-7)
8. 【Garcia C., Delakis M. Convolutional face finder: A neural architecture for fast and robust face detection [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004, 26(11): 1408-1423】 [↑](#endnote-ref-8)
9. Boser B.E., Sackinger E., Bromley J., et al. An analog neural network processor with programmable topology [J]. IEEE Journal of Solid-State Circuits, 1991, 26(12): 2017-2025 [↑](#endnote-ref-9)
10. Antony W. Savich., Medhat Moussa., Shawki Areibi. The Impact of Arithmetic Representation on Implementing MLP-BP on FPGAs: A Study [J]. IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 18, NO. 1, JANUARY 2007: 240-252 [↑](#endnote-ref-10)