人工智能训练和推理一体化轻量级部署实现方法的研究

何宾，郑阳扬

（北京化工大学信息科学与技术学院，北京，100029）

## 摘要

近年来，人工智能的不断发展由算法和算力两方面共同驱动。本文主要从算力这个角度来讨论实现人工智能的训练和推理的方法。解决算力问题，需要从性能、成本、功耗、灵活性、鲁棒性等多个方面综合考虑。目前，人工智能模型的训练大多采用GPU平台，虽然GPU的运算性能很高，但是其功耗也比较大，成本较高。在一些对功耗和成本要求比较苛刻的应用场景中就不适合使用GPU作为实现的平台。而高性能异构架构器件的出现为人工智能训练和推理的一体化实现提供了一种新的途径。典型的，在Xilinx和Intel推出的多核异构架构中，将多个高性能处理器和FPGA集成在单芯片中。较目前采用分离的训练和推理方法相比，采用单芯片实现AI训练和推理的一体化，较好的平衡了训练和推理的不同目标，进一步降低了实现AI训练和推理的成本和功耗，从而达到了算力上的轻量化目标，并提高了整个系统的灵活性和鲁棒性。

在本文中，我们以LeNet-5网络结构为基础，首先介绍了在Xilinx最新多核异构架构器件MPSoC中使用多核CPU实现对网络训练的过程；然后，研究将该网络模型转换为硬件逻辑实现的方法，并通过片内提供的总线接口AXI，将训练得到的模型参数从器件内的处理系统传递到由可编程逻辑构成的硬件加速器结构中；最后，在Xilinx MPSoC内对该一体化实现方法进行了测试和验证。

根据测试结果可知，该轻量化部署方案在MNIST数据集和CIFAR10数据集上的识别准确率分别达到了99.5%和75.4%，而图片的平均单帧处理时间仅为2.2ms。此外，该网络在SoC内硬件加速器运行功耗仅1.363W@100MHz。

关键词：深度学习、MPSoC、FPGA、卷积神经网络

## 1 引言

近年来，CNN广泛应用于人工智能中，尤其是在图像分类应用方面表现出巨大的优势[1-2]。但是，由于卷积神经网络运算量超大，因此在传统上使用GPU进行训练，然后再进行部署。这种方法虽然解决了CNN中运算量大的问题，但是并不能很好地解决CNN应用中的灵活性、成本和功耗的问题[3-4]。在特定的场景下，如天文设备与物联网设备上很难负担起GPU的能耗[5]。因此，研究人员相继提出了使用FPGA、ASIC等实现CNN加速的方法，并与GPU进行了对比[3][6-7]。除此之外，训练与部署相分离的情况让神经网络对环境的适应性变得很差，如在一些情况复杂的应用场景中，网络模型的参数也是需要动态调整，而不是一成不变的。而在基于FPGA的异构架构器件中，一方面，集成了高性能的多核处理器，使得可以对网络进行训练，并在器件内将神经网络的参数进行传输；另一方面，在器件内的可编程逻辑部分实现CNN的硬件加速器结构，使得一个神经网络可以在功耗、成本、性能方面达到最优[8-10]。同时，克服了传统部署方式采用训练和推理分离方式所带来的灵活性差的问题，显著改善了AI的响应能力和适应不同场景需求的能力。图1为本文提出的轻量化部署实现方法架构。

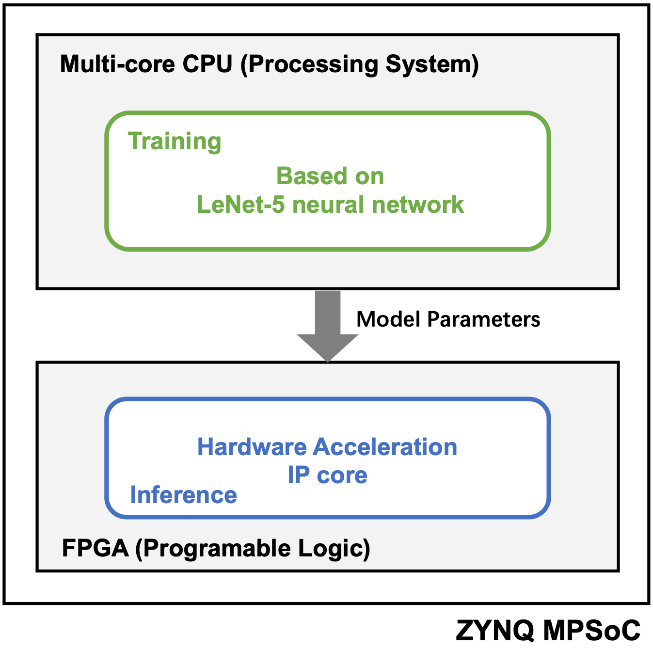


图1 基于异构架构的人工智能轻量化部署实现方法

综上所述，本文的主要贡献列举如下：

1. 利用异构架构的特点，我们提出了一种新的神经网络模型轻量化部署的方案，该方案使得在不同场景的人工智能应用可以在灵活性、性能、成本、功耗和抗干扰能力之间进行更好的平衡。实现了神经网络的训练与可灵活调整权值的前向推理加速的一体化部署。

2. 利用MPSoC中的PL (Programable Logic) 部分，根据有限的资源在神经网络层内实现了流水线与并行性之间的平衡，优化神经网络前向推理的性能。并将各层优化封装后构建为灵活的可定制硬件加速IP库。

3. 在MPSoC的PS (Processing System) 部分部署神经网络训练框架，支撑神经网络的训练。并编写自动化网络权值参数移植脚本，实现了轻量化部署一体化的数据处理自动化。

本文其余的部分如下所示：第二章介绍了一些相关的前沿性工作。第三章描述了神经网络轻量化部署的方法，从整体的角度介绍了我们的架构特点以及在PS端实现LeNet-5网络训练与在PL端推理加速IP核的构建方式。第四章介绍了推理加速IP核的实验，介绍了整体SoC的设计并验证加速IP核功能。第五章针对实验结果进行了加速器IP核的性能分析。最后，我们在第六章中总结了我们的贡献与展望未来的研究方向。

## 3本文方法

### 3.1人工智能一体化架构

从总体上来说，人工智能轻量化部署包含两个方面，其一是将需要训练的模型用异构架构器件内的多核处理器部分进行训练，然后得到训练后的模型参数；其二是在异构架构器件的可编程逻辑部分构建并优化网络的推理模型，该推理模型通过SoC内部的总线接口接收训练得到的模型参数。

图2为轻量级人工智能一体化整体实现结构。从该结构可知，MPSoC的PS端部署神经网络训练框架支持深度学习网络的训练；同时在PL端构建神经网络加速器IP核。通过将模型权值和待识别的特征图数据传递给IP核执行神经网络的运算加速，最终实现了神经网络的训练与可灵活调整权值的前向推理加速的一体化实现。该实现方法具体体现在以下几个方面：

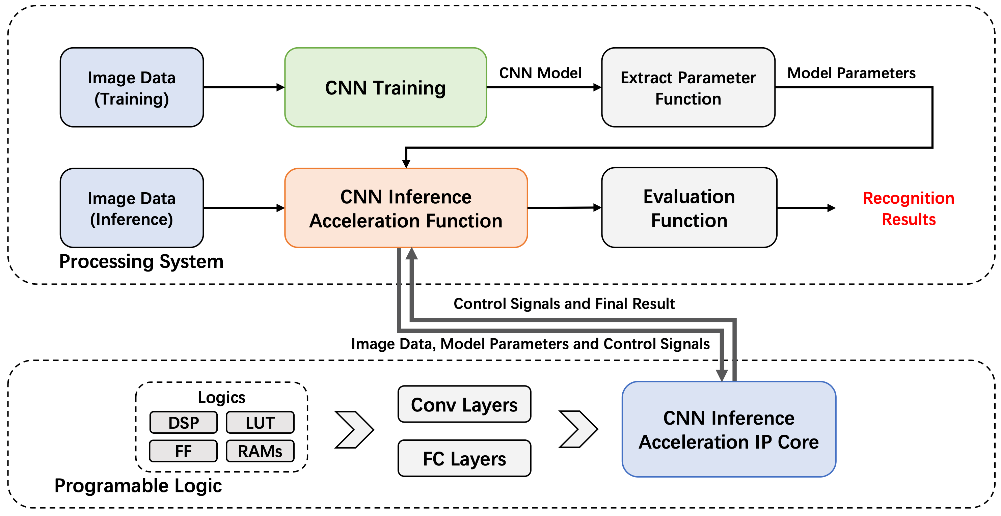


图2 系统整体架构

（1）首先，在SoC内部的PS端使用多核处理器进行CNN的训练，将大量的图像数据送入网络中，并进行多次迭代，最终得到在测试集上表现最优的CNN模型；然后，通过脚本导出模型权值并将其传递给CNN加速器IP核。

（2）在SoC内部的PL一侧实现CNN的基础网络层，并按网络结构连接这些基础网络层以构成完整的神经网络结构。

（3）通过高级可扩展接口（Advanced eXtensible Interface，AXI）规范，将PS与PL连接在一起构成训练和推理的完整硬件结构[28]。在该硬件结构上，通过Linux操作系统将CNN加速器IP核封装为可直接调用的应用程序接口（Application Program Interface，API）函数，以构成训练和加速的完整实现结构。

### 3.2神经网络的训练方法

通过在MPSoC内的ARM多核处理器上运行神经网络训练框架-PyTorch，实现对人工智能轻量化部署的训练，这种方法可以显著降低训练网络时的系统总体功耗。对于小样本以及对训练模型收敛速度要求不高的应用场景下，使用MPSoC内的高性能多核CPU执行对网络的训练，可以显著降低系统的整体成本以及功耗。

手机屏幕的截图

中度可信度描述已自动生成

图3 LeNet网络模型

当采用一体化人工智能轻量化部署的实现方法时，为了降低MPSoC内PS的训练负载，可以在计算机上首先对神经网络进行训练，然后得到预训练模型。该模型可高效运行在MPSoC的PS中，以完成最后的训练并提取网络模型的参数。

本文使用的LeNet-5网络具有五个网络层，前三个为卷积层，输入层为单通道的32x32像素的特征图，在每个卷积层后跟随最大池化层，用于控制网络规模；后面两个为全连接层，两个全连接层的最后一层完成图像的十个分类的输出。LeNet-5网络使用ReLU函数作为激活函数。通过log\_softmax函数，对输出层的输出进行归一化处理。

在使用CIFAR10数据集训练网络模型时，将LeNet-5网络第一个卷积层的输入从单通道扩展到了三个通道；在使用MNIST数据集训练网络模型时，需要将一维形式的MNIST数据集复制为三维形式，然后传给LeNet-5网络的第一个卷积层的输入通道。

为了训练修改后的LeNet-5模型，需要在MPSoC的PS端安装嵌入式Linux操作系统来运行PyTorch框架。在训练模型的过程中，如果出现收敛不理想的情况时，通过调整学习率、优化器等超参数，来提高模型的收敛速度与优化模型在测试集上的表现。

CPU与GPU均可以支撑神经网络的训练，由Nvidia公司推出的CUDA加速平台加速了神经网络的训练。目前绝大部分神经网络的训练都是在GPU上完成的。我们在PyTorch框架下对两类平台的训练速度与功耗进行了对比。如表1所示，GTX1050完成一个epoch的训练时间为MPSoC集成的多核的训练时间的0.08倍，但是功耗也约为PS端的30多倍。在边缘计算系统等功耗要求较苛刻的场景中，较低功耗的CPU也能为神经网络的训练提供可靠的支持，尽管它没有GPU的速度快。

表1 训练性能对比

|  |  |  |
| --- | --- | --- |
|  | **GPU** | **ZYNQ UltraScale+ MPSoC** |
| **型号** | **GTX1050** | **quad-core Arm Cortex-A53** |
| **功耗** | **75W** | **Less than 2W** |
| **训练时长/epoch** | **22.6s** | **286s** |

与GPU相比，因为MPSoC中的多核处理器的运算性能有限，因此需要降低batch size来减少训练时多核处理器的运行负载。从图4.b给出的实验结果可知，在500个epoch内，不管batch size的值如何变化，在训练集上的loss（损失）曲线始终保持正常下降的趋势，最终都会使模型正确收敛且不会出现过拟合情况。再结合图4.a给出的实验结果可知，batch size值越大，识别准确率的收敛速度越快，这就意味着训练模型所需的时间越短，这样就会显著降低多核处理器的整体功耗。因此，从准确率和损失两个角度考虑，在实际的模型训练中可以将batch size的值调小，这样在进行训练的时候不会显著影响模型的收敛速度，也不会使CPU负载过重。

图表

描述已自动生成图表

中度可信度描述已自动生成

(a) 识别准确率(acc)随epoch的变化 (b) 损失(loss)随epoch的变化

图4 不同的batch size对模型收敛速度的影响

当训练完成后，就可以通过Python脚本从训练好的模型中提取LeNet-5网络的权值参数，模型参数由key值和value值组成，其中key为网络层名字，value为对应网络层的参数值，数据类型为tensor。

在该设计中，首先根据网络层名字key提取出对应的value值，并将其数据类型从tensor转化为数据类型为fp32的数组numpy；然后，将其保存到以各层名字命名的文件中；最后，通过AXI规范将训练得到模型参数传递给PL内进行前向推理的硬件加速器。

### 3.3 卷积层硬件加速器结构的设计

前面提到，LeNet-5网络具有五个网络层，主要包括三个卷积层和两个全连接层，因此在设计神经网络加速器时可以针对这两种网络层的结构进行优化。前三个网络层均为卷积层，实现的方法相同。该设计中，将卷积层、激活函数和池化层放在同一个模块中，减少了接口的设计难度，并提高了代码的可读性和可维护性。

卷积层硬件加速器的数据流操作过程按顺序分为卷积操作、ReLU激活函数和池化操作。

（1）卷积操作表示为：

式中， 为对应第 *j* 个输入特征图和第 *i* 个输出特征图的卷积核， 为第 *i* 个输出特征图的偏置。

（2）激活函数ReLU表示为：

式中，为输入的特征图的值，为输出的特征图的值。

（3）池化操作表示为：

式中， 为最大池化的采样核大小。通过池化层操作来消除特征图中的冗余信息，这样可减少运算量，但不会影响到识别的精度。

通过对式（3-1）、（3-2）和（3-3）的分析，将卷积层硬件加速器操作过程表示为：

将式（3-4）所表示的卷积层操作模型映射为最基本的硬件加速器结构，如图5所示。通过对该基本结构的优化，进一步提高卷积层硬件加速器的整体处理性能。

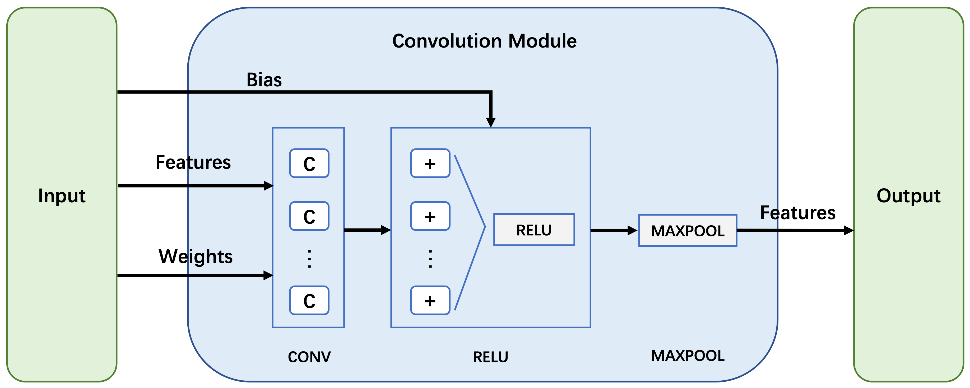


图5 卷积层的实现

当软件模型通过HLS工具转化为硬件结构时，软件模型的表达方式可能会对最终转化的硬件加速器结构产生不利的影响。典型的，式（3-1）给出的卷积操作通常嵌套循环实现。为了将重复读取特征图而产生的高时延影响降到最低，按如下的方式描述卷积操作的先后顺序。

|  |  |
| --- | --- |
| **Algorithm 1: Convolution** | |
| OUT: Output  IN: Input  W: Weight  R: OutputFeatureMapRow  C: OutputFeatureMapColumn  K: KernelSize  CHin: InChannelSize  CHout: OutChannelSize | |
|  | 1. for(kr = 0; kr < K; kr++)  2. for(kc = 0; kc < K; kc++)  3. for(r = 0; r < R; r++)  4. for(c = 0; c < C; c++)  5. for(chi = 0; chi < Chin; chi++)  6. for(cho = 0; cho < CHout; cho++)  7. OUT[cho][r][c] = W[cho][chi][kr][kc] \* IN[chi][r + kr][c + kc]; |

在转化为硬件加速器实现时，使用Pipeline命令可以很好的缩短for循环之内的指令触发间隔，从而提高吞吐量和降低时延。如表2所示，在不添加Pipeline命令的情况下，完成一次图像识别的延迟（latency）和间隔（Interval）均为7709158个时钟周期；在上面给出算法的第5行代码处添加Pipeline命令后，显著减少了一次图像识别的延迟和间隔，从7709158减少为353263个时钟周期，性能约有22倍的提升。但是，资源利用率也有了显著的增加，比如数字信号处理块DSP48E的使用量增加了约27倍，触发器（Flip-Flop，FF）和查找表（Look-Up Table，LUT）的使用量也增加了约2倍，这符合硬件加速器通过增加逻辑资源的使用率来换取并行处理数据能力提高的规律。

表2 优化前后卷积层资源使用对比

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **No\_optimization** | | | | **Pipelined** | | | |
| **Latency (clock cycles)** | **Latency** | | **Interval** | | **Latency** | | **Interval** | |
| **7709158** | | **7709158** | | **353263** | | **353263** | |
| **Utilization Estimates** | **BRAM\_18K** | **DSP48E** | **FF** | **LUT** | **BRAM\_18K** | **DSP48E** | **FF** | **LUT** |
| **79** | **9** | **29061** | **24651** | **83** | **244** | **53049** | **41842** |

当使用Pipeline命令时，存在流水线层数过高可能导致资源的使用量迅速增加或时延达不到预期值时，可以按照如下方式将较大的循环拆分开为两个较小的循环（两个较小的循环次数相乘等于较大的循环次数），从而降低流水线层数。

|  |  |
| --- | --- |
| **Algorithm 2: For Loop Optimization** | |
| OUT: Output  IN: Input  KL: InnerLoop  KH: OuterLoop  K = KL \* KH : OriginalLoopCount | |
|  | 1. for(h = 0; h < KH; h++)  2. for(l = 0; l < KL; l++)  3. OUT[h \* KL + l] = *f* ( IN[h \* KL + l]); |

在转化为硬件加速器结构时，在上面给出算法的第2行代码处添加Pipeline命令后，高层次综合工具只会展开第2行的循环，显著减少了流水线层数以及资源消耗。

### 3.4 全连接层硬件加速器结构的设计

正如前面所提到的那样，LeNet-5网络的最后两层均为全连接层。全连接层硬件加速结构相对简单，其操作模型可表示为：

式中，为全连接层的权值，对应第个输入特征图与第个输出特征图；为第个输入特征图的偏置。

通过激活函数ReLU后，对式3-2和式3-5进行组合，得到全连接层的操作模型如下所示：

由式3-6可知，在实现全连接模块时，将特征图与权值进行加权运算后，通过ReLU激活函数就得到了全连接模块的输出结果。全连接模块的最基本的硬件加速器结构如图6所示。

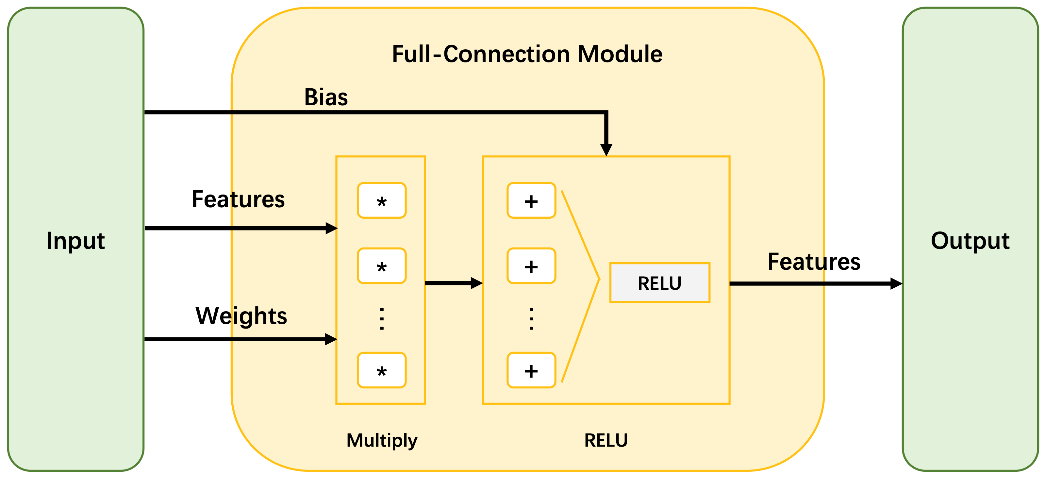


图6 全连接层的实现

对于全连接层，输入 与输出 都是一维的数组，权值 为二维数组。全连接层的模型可用如下的表示方式。

|  |  |
| --- | --- |
| **Algorithm 3: Full-Connection** | |
| OUT: Output  IN: Input  W: Weight  CHin: InChannelSize  CHout: OutChannelSize | |
|  | 1. for(chi = 0; chi < Chin; chi++)  2. for(cho = 0; cho < CHout; cho++)  3. OUT[cho] = W[cho][chi] \* IN[chi]; |

使用Unroll命令可以将使用循环表示的全连接层软件模型转化成高效的硬件加速器结构。如表3所示，在上面给出算法的第2行代码处添加Unroll命令后，可以将全连接层硬件加速器的时延（latency）和间隔（interval）从128856个时钟周期降低为45696个时钟周期。很明显，吞吐量提高了大约3倍。同样的，这也是以增加总体逻辑资源的利用率来提升全连接层的数据处理能力。

表3 优化前后全连接层资源使用对比

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **No\_optimization** | | | | **Full-connection** | | | |
| **FC layers′ Latency (clock cycles)** | **Latency** | | **Interval** | | **Latency** | | **Interval** | |
| **128856** | | **128856** | | **45696** | | **45696** | |
| **Utilization Estimates** | **BRAM\_18K** | **DSP48E** | **FF** | **LUT** | **BRAM\_18K** | **DSP48E** | **FF** | **LUT** |
| **79** | **9** | **29061** | **24651** | **88** | **14** | **29259** | **28196** |

## 4 实验和验证

### 4.1 验证平台的构建

在对一体化人工智能轻量化部署的设计进行验证时，采用了Xilinx公司的XCZU3EG-SBVA484 MPSoC作为核心器件，在该器件内的PS一侧集成了Arm四核Cortex-A53应用处理单元和Arm双核Cortex-R5F实时处理单元，在该器件的PL一侧集成了多达7.6MB的块存储器资源 (Block RAM，BRAM)，以及多达360个数字信号处理模块DSP48E等逻辑资源。基于该器件的硬件平台上搭载了2GB的LPDDR4存储器[29]。

基于修改的LeNet-5网络以及IP封装和复用技术，该设计在MPSoC内构建了一个可实现图像识别的系统[30-31]。如图7所示，通过AXI规范将用于实现图像识别功能的硬件加速器IP核连接到片内的PS一侧。图中，各个模块IP核的功能如下：

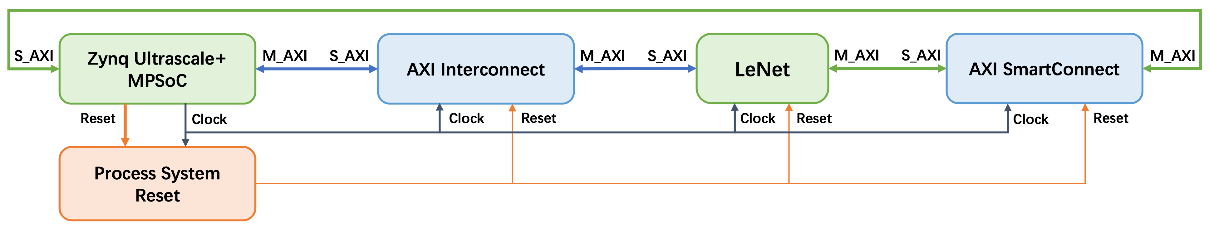


图7 MPSoC内搭建的片上系统设计图

(1) LeNet模块。该模块基于修改的LeNet-5网络结构实现图像识别硬件加速器结构。

(2) Zynq Ultrascale+ MPSoC（PS）模块。是对MPSoC内PS的抽象，它包含了四核Cortex-A53 APU和双核Cortex-R5F RPU。

(3) AXI Interconnect模块。遵从AXI规范，通过开关结构将多个AXI存储器映射的主设备连接到多个存储器映射的从设备，在该结构中主要作为连接S\_AXI外设的桥。

(4) AXI SmartConnect模块。类似于AXI Interconnect，用于将AXI外设连接到PS，在该结构中主要作为连接M\_AXI的桥。

(5) Process System Reset模块。处理器系统复位模块，在该结构中用于为PS和其他三个模块产生复位信号。

在使用Xilinx Vivado软件对图7给出的图像识别系统的整体结构进行综合与实现后，所使用PL的逻辑资源如表4所示。

表4 Vivado综合实现后资源消耗报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Utilization Estimates** | **BRAMs** | **DSP** | **FF** | **LUT** | **LUTRAM** |
| **41.5** | **248** | **51434** | **48966** | **3288** |
| **Utilization (%)** | **19.2** | **68.89** | **36.45** | **69.40** | **11.42** |

根据Vivado软件中的功耗分析工具可知，该图像识别体系统的总功耗仅为3.22W，而用于实现图像识别体前向推理的硬件加速器模块LeNet的功耗仅为1.363W。

### 4.2 验证方法的设计

基于PYNQ框架，使用Python语言编写图像识别系统的软件代码[32]。在该框架内提供的Overlay编程库为IP核生成可调用的Python API，使得该系统的软硬件协同设计更加便捷。该系统的数据流如图8所示。

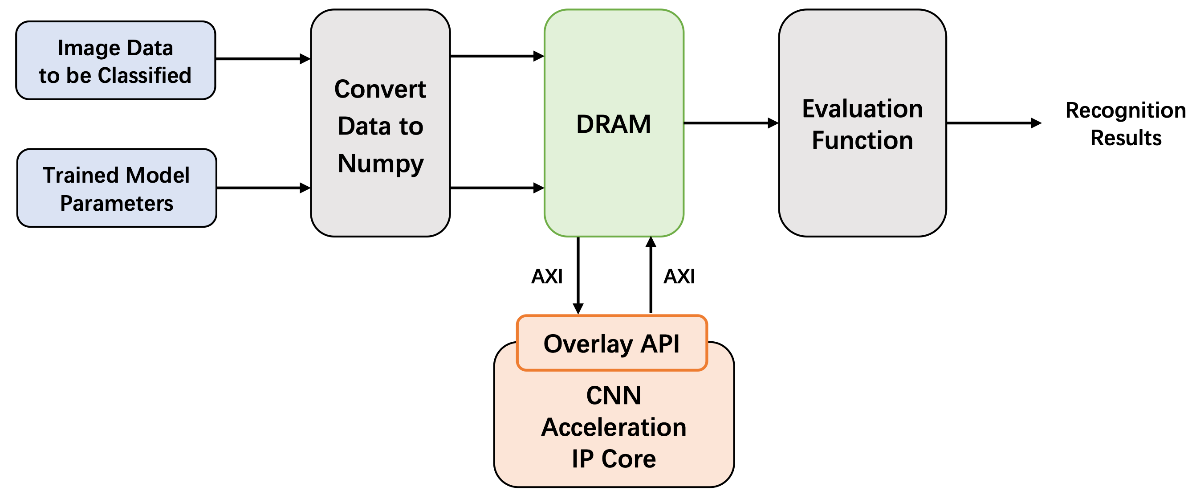


图8 PL端加速推理数据流

在PS端将待识别的图像数据和已训练完成的模型参数导入后，均转化为fp32类型的numpy数组。这些numpy类型的数组会被存储到DRAM中，等待着CNN加速器读取。在PL端构建的CNN加速器IP核的接口会被Overlay编程库封装为可调用的Python API。当运行程序调用加速器时，存储在DRAM中的numpy数组将通过AXI总线被传入加速器中进行前向推理的计算。当加速器执行完成时，返回的结果将通过AXI总线传输回DRAM中。最后经过评估函数处理后就得到了分类的结果。

## 5 实验结果与分析

经过测试集上10000张图片的验证，加速器IP核对单帧的推理速度为2.2ms，平均每秒可识别帧数约450余帧，而模型在Inter i5-4300U CPU上单帧处理时间为8.1ms，不仅推理速度得到了约4倍的提升，且在功耗表现上更有优势，更符合轻量化部署的要求。两种器件的性能对比如表4所示。

表4 性能对比

|  |  |  |
| --- | --- | --- |
|  | **CPU** | **ZYNQ UltraScale+ MPSoC** |
| **型号** | **Inter i5-4300U** | **XCZU3EG-SBVA484** |
| **工作频率** | **2.5GHz** | **100MHz** |
| **功耗** | **44W** | **3.22W** |
| **推理速度** | **8.1ms** | **2.2ms** |

最终加速器IP核对MNIST手写体数字体验证集的识别准确率为99.5%，对Cifar10验证集分类的识别准确率为75.4%，准确率均与模型在计算机上的表现一致。下图为部分实验结果截图的展示：

加载MNIST网络权值并使用测试集验证的结果如图9所示：

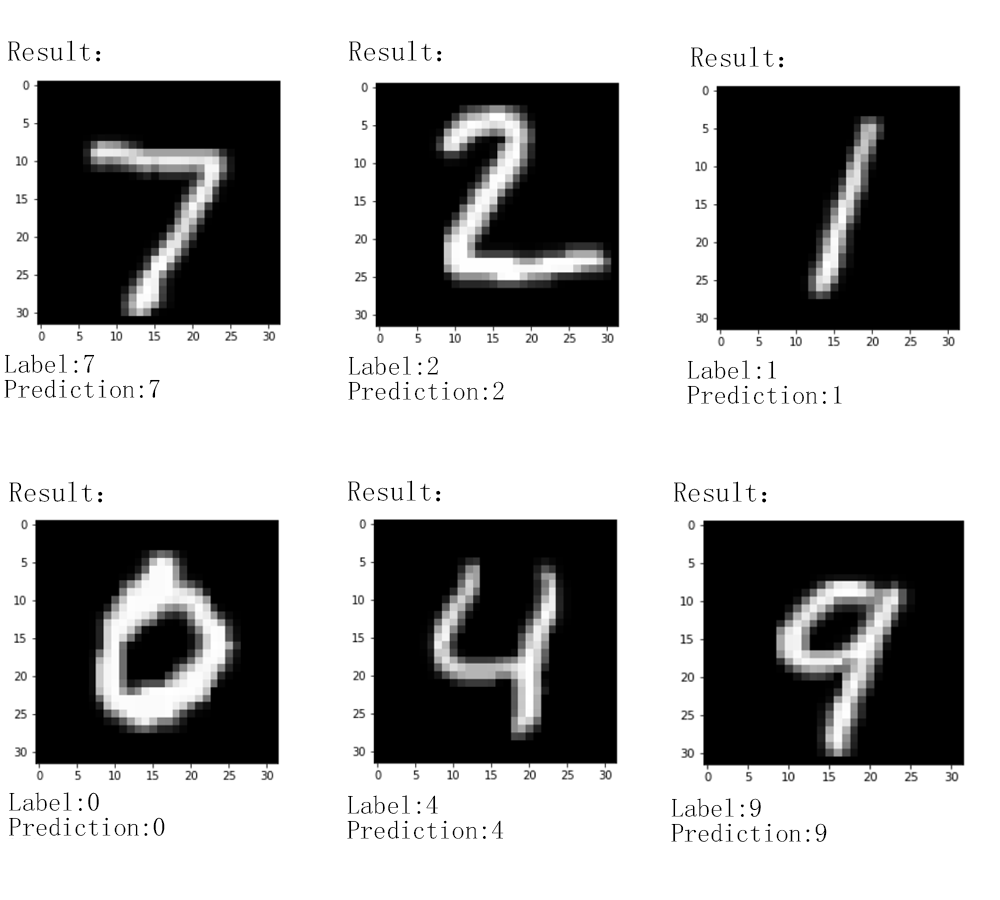


图9 加速器在MNIST数据集上的识别结果

加载Cifar10网络权值并使用测试集验证的结果如图10所示：

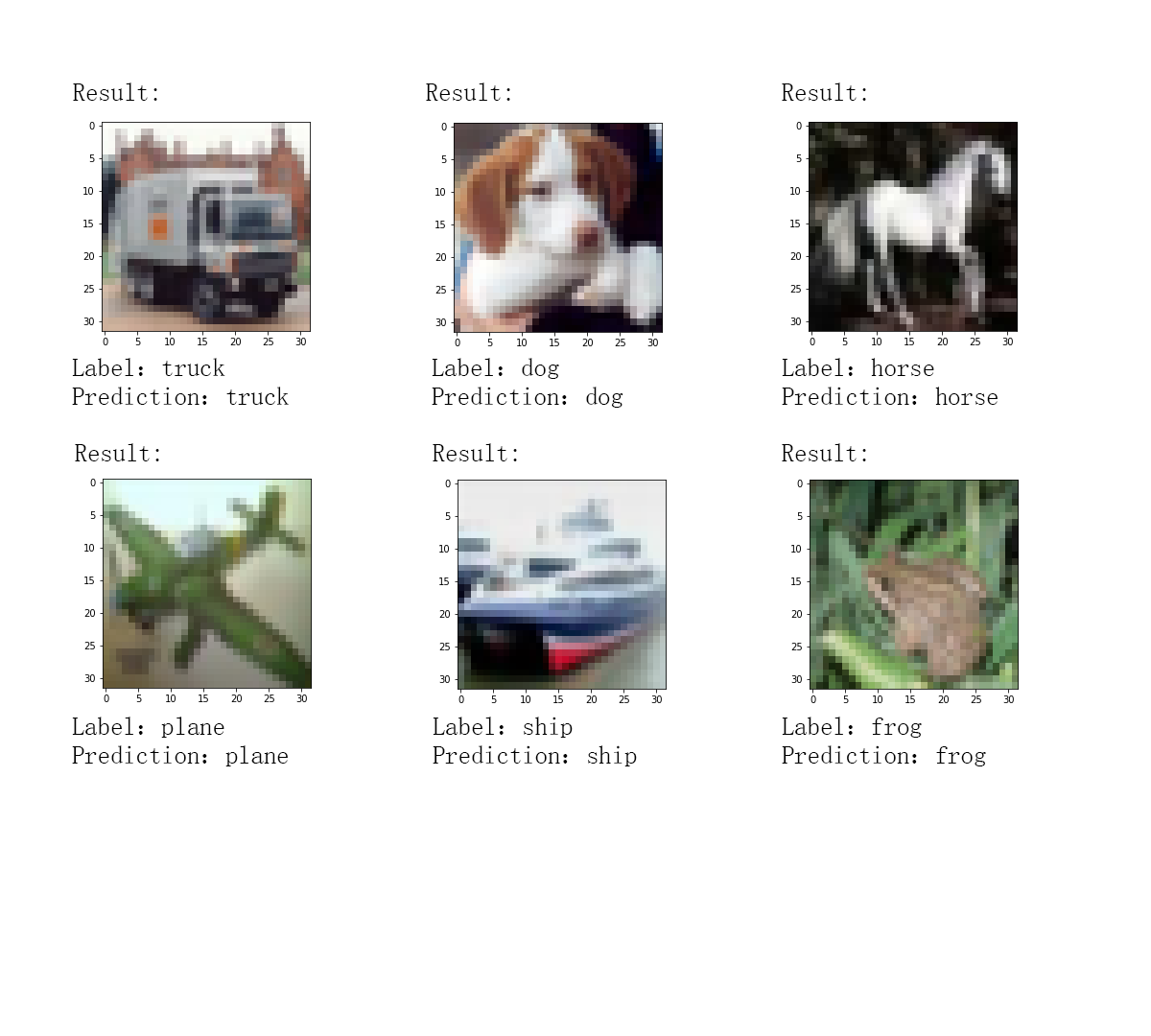


图10 加速器在CIFAR10数据集上的识别结果

## 6.讨论

在本文中，采用了MPSoC实现一体化的LeNet-5网络的训练和推理模型的构建，较好的满足了训练和推理的不同需求目标，显著降低了实现成本和功耗。从算力上解决了训练和推理之间的数据处理的灵活性问题。由于在采用了单芯片的实现方法，使得从算力上显著了整个系统的鲁棒性，从而为人工智能在对成本、性能和功耗要求比较苛刻的应用场景中，找到了一个新的解决方法。

相信，随着半导体集成电路技术的不断发展，未来会有越来越多的高性能异构架构器件出现，从而为人工智能的发展提供更好的算力平台，从而推动人工智能技术的不断发展，使之能更好的为人类的文明进步服务。之后的研究中，我们将致力于将这种架构应用到更复杂的应用场景中，并探索更复杂的网络模型的实现。

正如本文开头所提到的那样，半导体发展，集成度提高，高算力器件集成在单芯片中。GPU算法算力