电子科技大学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

专业学位硕士学位论文

MASTER THESIS FOR PROFESSIONAL DEGREE

论 文 题 目	基于 FPGA 的深度学习算法加速
专业学位类别	工 程 硕 士
学 号	201722040411
作 者 姓 名	黄磊
指 导 教 师	荣健 教授

分类号 _____ 密级 _____

UDC [注1] _____

学 位 论 文

基于 FPGA 的深度学习算法加速

（题名和副题名）

黄磊

（作者姓名）

指导教师 荣健 教 授

电子科技大学 成 都

（姓名、职称、单位名称）

申请学位级别 硕士 专业学位类别 工 程 硕 士

工程领域名称 电子与通信工程

提交论文日期 **2020.05.29** 论文答辩日期 **2020.05.31**

学位授予单位和日期 电子科技大学 2020 年 **6** 月

答辩委员会主席 洪劲松，教授

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

Deep Learning Algorithm Acceleration Based On FPGA

A Master Thesis Submitted to

University of Electronic Science and Technology of China

Discipline: Master of Engineering

Author: Lei Huang

Supervisor: Jian Rong

School: School of Physics

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

作者签名：黄磊　　日期：2020 年 5 月 29 日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名：黄磊　　导师签名：

日期：2020 年 5 月 29 日

摘要

在近些年，深度学习算法有着飞速的发展，其核心为卷积神经网络。卷积神经网络应用及其广泛，在图像识别分类、自然语言处理、情感分析等领域均有应用。神经网络的主要实现方式还是在 GPU 平台上，各项指标也是在 GPU 上完成改进的。但是在实际应用中，经常会面临各种低功耗的场景，此时 GPU 的巨大功耗成为了神经网络算法落地的障碍。

神经网络一个很重要的应用领域是目标检测，在目标检测算法中，YOLOV2（You Only Look Once）是一个极具代表性的算法，其结构简单，检测速度非常快。由于在低功耗场景下的特殊需求，神经网络在嵌入式设备上也有很多的应用，但是有传统的嵌入式设备绝大部分是基于 ARM 的平台，神经网络在 ARM 上部署时存在的依据巨大的问题就是算力不足的问题，所以基于 FPGA 的硬件加速平台就应运而生了。FPGA 由于其独特的架构，被广泛应用于实时信号处理、图像处理等领域，其并行性也为卷积运算提供了巨大的算力。

本文中采用 YOLOV2 来做为核心算法，先分析了 YOLOV2 的原理，然后根据其原理来对算法中的参数进行量化，以减少运算和传输的带宽消耗，从而加速算法。本设计中，采用 ZYNQ 系列的 FPGA 芯片进行算法实现，根据 ZYNQ 系列芯片的特点，再结合 YOLOV2 算法的层次结构，将 YOLOV2 算法中的部分层使用硬件来实现，另一部分层使用软件来实现，采用软硬件结合的方式来提高计算效率。本设计中采用 HLS 的方式来实现算法中的加速 IP 核，采用 C 来实现算法，使用 C 编写 Testbench，从而保证算法的正确性，加速算法的验证。对于硬件加速过程中算法的并行性，在本设计中主要采用两个方式。对于层内的运算的并行化，将数据进行分块，每一块分别进行运算，最后将结果拼接在一起。对于每个模块的运算，采用 HLS 并行优化来进行，分别对数组和循环添加优化指令进行并行优化。整个系统采用 PYNQ 的软件框架来实现，完成数据的预处理和以及最终的结果分析。本设计为深度学习算法 YOLOV2 提供了很好的硬件加速方案，充分发挥了 FPGA 的并行性，对于实际工程实践有着重要参考意义。

关键词：卷积神经网络，YOLOV2，HLS，FPGA，PYNQ

ABSTRACT

In recent years, deep learning algorithm has developed rapidly, and its core is the convolutional neural network. Convolutional neural network is widely used in image recognition and classification, natural language processing, emotion analysis and other fields. The main way to realize neural network is still on GPU platform, and the indicators are also improved on GPU(Graphics Processing Unit). However, in practical application, it is often faced with various low-power scenarios, at which time the huge power consumption of GPU becomes an obstacle to the landing of neural network algorithm.

An important application field of neural network is target detection. Among the target detection algorithms, YOLOV2 (You Only Look Once) is a very representative algorithm with simple structure and very fast detection speed. Due to the low power consumption under the scenario of special demand, the neural network also has a lot of application in the embedded devices, but there are most of the traditional embedded devices based on ARM platform, neural network deployment on the ARM of force by huge problem is the problem of insufficient, so the hardware acceleration platform based on FPGA(Field Programmable Gate Array) was born. Because of its unique architecture, FPGA is widely used in real time signal processing, image processing and other fields, Its parallelism also provides great computational power for convolution operation.

In this thesis, YOLOV2 is used as the core algorithm. The principle of YOLOV2 is first analyzed, and then the parameters in the algorithm are quantified according to the principle to reduce the bandwidth consumption of calculation and transmission, thereby speeding up the algorithm. In this design, the ZYNQ series FPGA chip is used for algorithm implementation. According to the characteristics of the ZYNQ series chip, combined with the YOLOV2 algorithm hierarchy, part of the YOLOV2 algorithm is implemented in hardware, and the other part is implemented in software Combination of software and hardware to improve computing efficiency. In this design, HLS is used to implement the accelerated IP core in the algorithm, C is used to implement the algorithm, and C is used to write Testbench to ensure the accuracy of the algorithm and accelerate the verification of the algorithm. For the parallelism of the algorithm during the hardware acceleration process, two methods are mainly adopted in this design. For the parallelization of operations within the layer, the data is divided into blocks, each block

is operated separately, and finally the results are spliced together. For the operation of each module, HLS parallel optimization is used, and optimization instructions are added to the array and the loop for parallel optimization. The entire system is implemented using the PYNQ software framework to complete data preprocessing and final result analysis. This design provides a good hardware acceleration solution for the deep learning algorithm YOLOV2, and fully utilizes the parallelism of FPGA, which has important reference significance for practical engineering practice.

key words: CNN，YOLOV2，HLS，FPGA，PYNQ

目录

第一章 绪论

1.1 课题研究的背景及意义

卷积神经网络来源于人工神经网络，上个世纪以来，神经网络经过了三起三落，最后被广泛应用于各个领域。神经网络独特的特征提取方式，使得神经网络代替了传统的算法[1]。另一方面，卷积神经由于其能够适应图像平移旋转的特征，在目标检测、目标跟踪领域逐渐成为了主流算法。目标检测算法在人脸识别、车牌识别、智能拍照等领域有着广泛的应用，在这些领域中往往是由嵌入式产品搭建的平台来运行目标检测算法。基于卷积神经网络的目标检测算法中包含大量的卷积运算，卷积运算需要进行大量的累乘加运算，且这些累乘加运算很多都是并行执行的，由于其嵌入式设备很多都是基于 CPU 平台的，主打低功耗策略，在性能上无法满足卷积神经网络运算的需要[2]。英伟达公司基于此需求研发出了嵌入式 GPU，但是由于嵌入式 GPU 的能耗比太低，也不适宜使用在移动平台上进行深度学习算法的部署实现。

FPGA 是 Xilinx 公司在上个世纪发明的，FPGA 实现方式是基于查找表的，FPGA 中包含查找表、触发器、可编程 IO、可编程布线资源、嵌入式存储器以及 IP 硬核。FPGA 独特的不依赖于指令集的架构，以及并行的流水线运算方式，使得 FPGA 有着极大的运算带宽以及数据传输速率[3]。由于 FPGA 的可重配置性，使得用户能够自定义算法架构以及接口，将深度学习算法部署在 FPGA 上，且可以重复修改而不需要更换 FPGA 芯片，适合于不断迭代更新的深度学习算法的部署和开发。另一方面，GPU 也在嵌入式平台上有所作为，但是基于指令集的冯诺依曼架构使 GPU 在实现算法时无法达到高度的并行性。FPGA 天然的并行性让 FPGA 成为了深度学习加速器领域的不二选择。

深度学习算法在实际生活有着广泛的应用，例如目标检测、语音识别、文字识别等领域。目标检测算法常用于监控、智能驾驶等领域，这些领域在进行运算的时候需要及时对采集到的数据进行运算处理和分析，以便设备做出反应，所以对实时性和准确率要求很高，所以 FPGA 非常适合这些领域的深度学习算法部署实现[4]。

1.2 课题研究的现状

在深度学习的一些应用中，例如机器人和自动驾驶领域，为了提供更精确的目标检测结果，通常会添加更多的网络层，这会导致网络的尺寸非常庞大[5]。这种方

式不仅使得网络的深度变深了，也会使网络的结构变得更加复杂。所以网络中有大量的参数以及运算操作，并且还需要大量的计算资源。这些对于通用处理器来说是非常大的挑战[6]。所以为了提高深度学习算法的吞吐量和性能，一系列的加速方案应运而生，ASIC、FPGA 以及 GPU 等等加速方案，用来推理或者训练深度学习算法[6]。训练过程是在 off-line 的情况下实现的，所以相比训练而言，推理过程的速度具有更大的重要性[7]。

在训练和分类过程中，GPU 是使用最广泛的硬件加速器之一[8]。原因在于 GPU 具有很高的带宽以及吞吐率，同时还具有高效的浮点操作能力[8]。但是，GPU 加速会产生很大的功耗[9]。所以，CNN 在云端服务器或者是电池供电的设备上实现的难度是很大的[10]。此外，GPU 可以并行处理大批量的图片。在某些应用中，例如视频流的处理，图像被逐帧的处理，这对处理的速度要求是很高的。对于一些跟踪算法来说，本帧的处理会对下一帧的处理产生影响。卷积神经网络中每个特征图神经单元与其他神经单元共享权重，这是一个非常有用的属性[13]。有研究表明，从片外内存中读写数据的功耗，比起运算过程中的功耗要大得多[14]。所以，深度学习加速器需要谨慎考虑时间和功耗，以获得更好的架构[15]。现有的研究表明，对于深度学习加速领域而言，FPGA 比 GPU 的能耗比更高。与 GPU 相比，FPGA 和 ASIC 在内存和 IO 带宽上有一定的限制。但是，它们可以在很低的功耗下呈现出稳定的性能[11]。但是，对于 ASIC 加速来说，在开发周期、成本和灵活性等方面是无法满足需求的。另一方面，FPGA 作为一个可重构的加速器，可以在很低的功耗下提供一个很高的带宽。另一方面，FPGA 开发方式也有了很大的进化，基于 HLS（高层次综合）工具的开发方式提高了 FPGA 的开发效率，缩短了 FPGA 的开发周期[12]。

1.3 论文的主要工作和章节安排

本文使用 YOLOV2 作为代表算法进行深度学习算法加速研究。在算法层面使用量化的方式减少了深度学习算法中参数的数量，使用 HLS 进行算法的运算加速。分析了算法的数据传输，采用多缓存结构的方式改善了算法硬件实现过程中的内存带宽不足的问题，根据卷积池化算法的运算特点，使用数据分块以及层内并行的方式提升了算法加速核的性能。首先在 vivado 中实现了整个硬件系统的搭建和加速 IP 的封装，在 PYNQ 框架中实现的整个算法系统的数据前处理和后处理以及显示与分析。最后对算法系统进行了系统的评估和分析，得出了加速性能结果。将本文中的算法优化方式和实现方式与主流方式进行对比，得出了本文中所提到的系统方案的诸多优势。由上述得出本文主要安排如下：

第一章介绍本文的研究背景和深度学习算法加速的方法和研究现状，概述本

文的主要工作和章节安排。

第二章主要介绍深度神经网络的基本原理和网络结构，简要介绍了目前的集中主流深度学习算法。重点分析了深度学习算法中卷积层池化层以及激活层的结构，对比了当前三种主流的深度学习加速平台，GPU、ASIC 和 FPGA。

第三章详细介绍了 YOLOV2 算法的基本原理，详细分析了 YOLOV2 的网络特点。根据 YOLOV2 的网络特点做了数据量化以及硬件架构方案的优化，数据量化包括权重数据的量化和偏移数据的量化，硬件架构的优化包括运算结构优化和缓存优化。

第四章详细介绍了 Xilinx 的异构 FPGA 开发套件 ZCU104 的基本结构和开发流程，讲解了 HLS 加速的开发流程和开发优势。详细介绍如何将 HLS 的优势与算法的特性结合起来，从而开发出高效的加速 IP，基于 HLS 的加速 IP 主要在三个方面做了优化，分别是层内并行、循环优化和数组优化。算法加速 IP 实现之后需要对 IP 进行集成，此过程是在 PYNQ 框架下进行的，此框架首先实现 PL 部分的 IP 核封装以及驱动生成，然后将 bitfile 作为一个子模块集成到 PYNQ 中，实现一个软硬件协同的 SOC。最终是在整个系统之上运行操作系统以及 python 解释器，将整个算法运行在平台之上，完成在环测试分析以及性能评估，得出本文最终的加速效果。

第五章对本文中所作的所有工作进行归纳总结，分析整个系统存在的其他瓶颈，并且提出了优化的方案，最后对整个算法加速行业做出了美好的展望。

第二章 深度学习算法原理及其硬件实现

2.1 深度学习算法的简介

在 1950 年，Arthur Samuel 首次使用机器学习这个词，使用这个词来描述计算机的一种能力，这种能力就是从原始数据中提取和学习到有用的模式信息[16]。这使得机器能够解决很多的问题，这些问题都是使用数学公式或者是传统的方法难以解决的。例如，素数在数学上被定义为一类数，这类数的因数只有 1 和它自己。所以我们可以使用传统的方法来编写代码实现判断一个数是否为素数。但是考虑到其他情况，例如判断一张图片中是否有人存在，这种情况很难使用数学方式来表达。有一些任务，类似于图像理解和自然语言处理，这些任务都是人类所擅长的东西。为机器指定序列，让他们来找出解决问题的方法，这是非常麻烦的，有时候甚至是不可能实现的。对于这类问题，机器学习可以提供高效准确的解决办法。机器学习通常分为三种，监督学习、非监督学习和强化学习。在监督学习中，通过打上标签的包含正确结果的示例，机器可以得到输入输出映射的相关参数。在训练阶段，主要的目标是减少损失函数和提高精度，损失函数代表的是训练样本的真实值和预测值之间的误差。在训练中可以得到函数，这个函数可以将模型中从未出现过的输入映射到输出。有很多这种监督学习算法，例如线性逻辑回归和支持向量机。

另一方面，和监督学习不一样的是，无监督学习中的训练数据是一系列的观测值，而不是像监督学习中那种输入和输出的对应序列值。模型通过推理输入观测值的联合密度来从数据中获取有用的模式。和监督学习不同的是，无监督学习中没有失败的概念，也没有成功的衡量标准，而且也没有衡量模型效率的方法。无监督学习的一个很重要的用途就是聚类和数据分割。在这些任务中，最主要的目标就是将这些数据划分为特定个数的组，它们与组内的数据相比相似度明显高于组外的数据。典型的聚类方法有 K-means 聚类和高斯混合聚类两种方式。聚类也可以分层执行，表现为在每个组中创建子组，或者是将每个小组联合为更大的组。

在强化学习中，每一个样本都不会有一个明确的输出。模型通过经验的学习，首先发出一个动作，然后会收到关于这个问题的奖励和惩罚[16]。通过收集这些经历，在不需要准确分析每种情形的正确动作情况下，模型可以在不懂的情形下做出动作[17]。对于机器需要怎样玩游戏这种场景来说，这种学习方法特别适用。在这种情况下，要让模型对于游戏的每一个阶段都有很好的反馈，其实这对于监督学习来说是一件很困难的事情，但是强化学习在这个时候就可以派上用场了。

2.2 从线性回归到深度神经网络

线性回归是最简单和最基础的无监督学习方法。如公式(2-1)所示，线性回归表征的是 n 维的输入特征向量$X' \in R^n$和预测的输出值$\hat{y} \in R$，权重$W' \in R$和偏移值 b 是从模型中学习训练得到的。为了更好的简化模型，我们可以在输入向量中添加额外 1，并且在权重向量中加入额外的 b 来获取$X, W \in R^{n+1}$。训练的过程可以看作是一个优化问题，这个问题是为了减少损失函数J(w)。损失函数用来表征 m 个训练样本的真实输出和预测输出的均方误差，其具体公式如公式(2-2)所示[18]。如果我们将输入的 m 矩阵定义为$X \in R^{m\times(n+1)}$,$\hat{y}$和y是预测输出和真实输出的。损失函数也可以表示为公式(2-3)中那样。线性回归可以用来预测连续值，例如公司的销售业绩、房价以及学生的分数等等。

$$\hat{y} = w'^T X' + b = W^T X \tag{2-1}$$

$$\mathrm{J(w)} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y_i} - y_i)^2 \tag{2-2}$$

$$\mathrm{J(w)} = \frac{1}{m}||\hat{y} - y||_2^2 = \frac{1}{m}||X_w - y||_2^2 \tag{2-2}$$

为了找到 W 的值使得损失函数最小，可以通过解析小规模的输入，解析时候可以通过求解J关于W的梯度，其具体求解过程如公式(2-4)、公式(2-5)、公式(2-6)、公式(2-7)、公式(2-8)、公式(2-9)所示，整个过程目的是为了求解 W 的梯度。

$$\nabla_w J(w)=0 \tag{2-4}$$

$$\frac{1}{m}\nabla_w||X_w - y||_2^2=0 \tag{2-5}$$

$$\frac{1}{m}\nabla_w(X_w - y)^T(X_w - y) = 0 \tag{2-6}$$

$$\frac{1}{m}\nabla_w(w^T X^T X_w - 2w^T X^T y + yy^T) = 0 \tag{2-7}$$

$$\frac{1}{m}(2X^T X_w - 2X^T y) = 0 \tag{2-8}$$

$$\mathrm{w} = (X^T X)^{-1} X^T y \tag{2-9}$$

但是，随着输入特征图的尺寸增大，这种方式并不能很好的适配，因为当输入$(\mathrm{n}+1)\times(\mathrm{n}+1)$的特征值，其结果为$X^T X$，计算复杂度为$\mathrm{O}(n^3)$。使用如图 2-1[19]中所描述的梯度下降迭代的方式，可以解决这个问题。在梯度下降函数中，w 沿着

使损失函数下降最快的方式变化，变化的步进由学习率决定。如果学习率α太小，那么要想达到 J 的局部最小值，那么需要更多次的下降才可以完成。相反，如果学习率太大，那么就会出现三种情况：震荡、过冲以及无法收敛到 J 的全局最小值。通过对每个函数求权重的偏导，得到损失函数的斜率，由此可以得到损失下降最快的方向。如图 2-2[19]所示，为了简单起见，表示为$W = [\ w_1\ w_2]^T$。考虑到w_1和w_2，这些值在每次迭代时候就改变，改变的方向是超平面上最损失函数减小最快的方向，直达达到局部最小值。

```
function [w] = gradientDescent (n, alpha, m, Y, X, threshold, maxIterations)
  %Concatenate a feature of value 1 to each input feature vector
  X = [ones(m,1) X];
  %Initialize the weights to random values and calculate the initial error
  w = rand(n+1, 1);
  err = 1/m * norm((X * w) - Y);
  iter = 0;

  %Iterate until max no. of iterations is reached or error is below the given threshold
  while(iter < maxIterations && err > threshold)
    %Calculate the derivative of cost function w.r.t. w where X' is the transpose of X
    dw = 1/m * (X' * ((X * w) - Y));
    %Update the weight values in the direction of steepest descend
    w = w - (alpha * dw);
    %Calculate the error using updated weight values
    err = 1/m * norm((X * w) - Y);
    iter = iter + 1;
  end
end
```

图 2-1 梯度下降算法 matlab 代码[19]

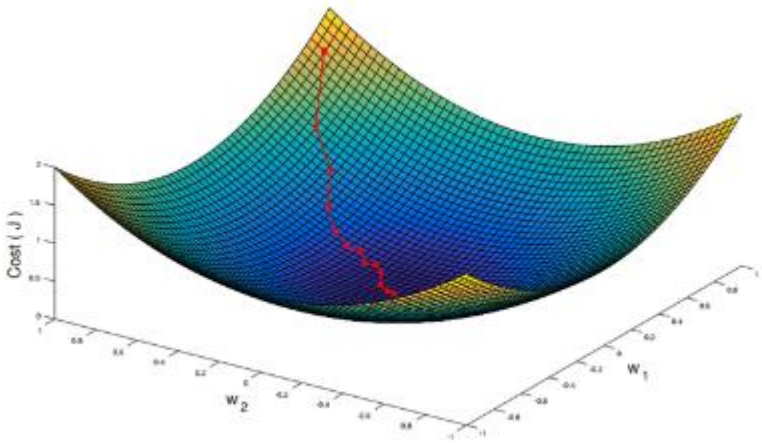

图 2-2 梯度下降算法示意图[19]

另一个模型就是逻辑回归，逻辑回归常被用来解决二值分类问题。在二值分类问题中，与线性回归中不同的是，模型的输出是离散的值。逻辑回归可以被看作是一种线性回归，其后跟上如公式(2-10)所示的激活函数[20]，在该式中σ是 sigmoid 激

活函数[21]。Sigmoid 函数的输出是介于 0 和 1 之间的实数，其代表的是 y 为 1 的概率。这个模型经过优化之后，通过输出值来代表类别，其中接近于 0 的值代表其中一类，接近于 1 的值代表另一类。与线性回归不一样的是，最优参数 W 没有解析解。另一方面，线性回归中的均方误差成本函数，将会在逻辑回归的情况下，产生一个非凸的成本函数。这个成本函数意味着梯度下降无法获得局部最优解。解决这个问题的办法是，在逻辑回归中，通过最小化如公式(2-11)中所示的负对数似然代价函数，再通过梯度下降的方式，迭代的找到最优解。如何计算 err 和 dw 是公式 2.1 中梯度下降算法的唯一改变。

$$\hat{y} = P(y = 1|x; w) = \sigma(w^T) \quad \alpha(z) = \frac{1}{1+e^{-z}} \tag{2-10}$$

$$J(w) = -\frac{1}{w}[y^T \log(Xw) + (1-y)^T \log(1-Xw)] \tag{2-11}$$

人工神经网络是一个有向无环图，这个有向无环图被称为神经元。这些神经元通过网络连接在一起，这些网络代表更复杂的非线性映射函数，这个函数代表输入与输出之间的映射关系。与逻辑回归很相似的一点是，每个神经元之后都跟着一个非线性激活函数，但是这个非线性激活函数不一定是 sigmoid 函数。人工神经网络包含着一个或者多个由神经元构成的隐藏层。这些层存在于输入和输出之间，代表输入特征图和输出预测值。深度神经网络使用十分广泛，通常用于描述人工神经网络，这些人工神经网络通常不止一层。

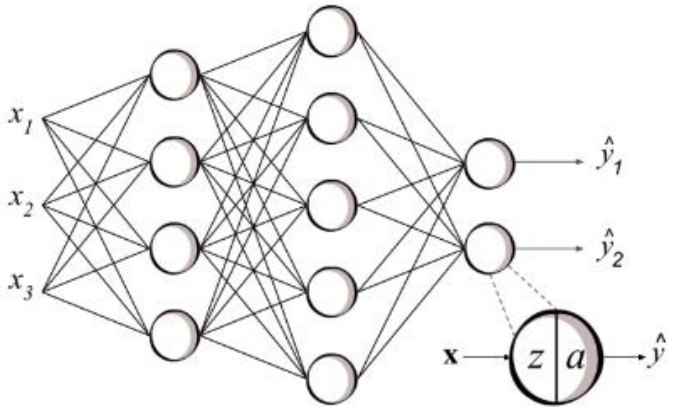

图 2-3 人工神经网络示意图[22]

如图 2-3[22]中所示，DNN 的产生是受到生物神经系统中的信息处理方法的启发。平均每个人脑大约含有 860 亿个神经元，这些神经元通过复杂的网络连接在一起，这些神经元从周围的神经元接收到输入，当输入超过一定的阈值时，神经元

将会被激活。前向传播被应用于 DNN 推理中，通过前向传播可以计算出公式(2-10)中的结果，此计算对于每个神经元是从输入开始，通过隐藏层，直到获取到预测的输出值。但是，非线性激活函数有很多选择，不能直接采用 sigmoid 函数。例如正切双曲线函数也被用作非线性激活函数。在众多非线性激活函数中，使用的最近的就是 ReLu 函数，它的具体公式是$a(x) = \max(0, x)$，采用 ReLu 函数作为激活函数可以使得训练速度变得更快。L 层的前向传播计算可以看作是矩阵向量乘法，两个乘数中一个是权重矩阵W^l，另一个是前一层的结果。这个结果来源于公式(2-12)。在层权重矩阵$W^{(l)} = [w^{ij}]$中，$[w^{ij}]$是连接$l-1$层的第 j 个神经元和第 l 层的第 i 个神经元。公式(2-12)可以被看做一系列的累乘加操作，这些运算是 CNN 推理和训练的核心操作。

$$x^{(l)} = W^{(l)}x^{(l-1)} = \begin{bmatrix} w_{10}^{(l)} & \cdots & w_{1d^{(l-2)}}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{d^{(l)}0}^{(l)} & \cdots & w_{d^{(l)}d^{(l-1)}}^{(l)} \end{bmatrix} \begin{bmatrix} x_0^{(l-1)} \\ \vdots \\ x_{d^{(l-1)}}^{(l-1)} \end{bmatrix} \tag{2-12}$$

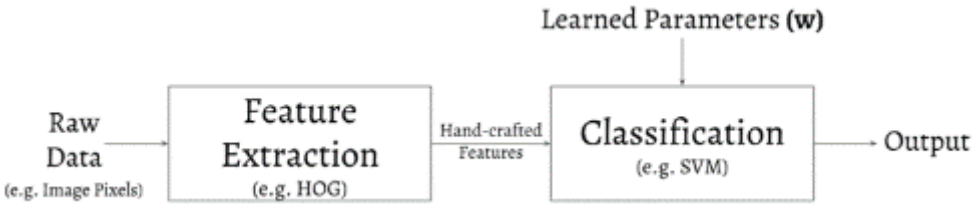

图 2-4 随机梯度下降示意图[23]

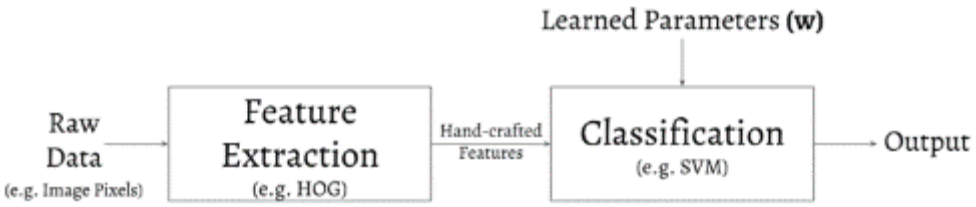

图 2-5 小批量梯度下降示意图[23]

DNN 最常用的训练方法有两种[23]，如图 2-4[23]所示为随机梯度下降，另一种为如图 2-5[23]所示的小批量梯度下降。其中小批量梯度下降是梯度下降算法的一个改进算法。在这些改进算法中，在随机梯度下降算法中，权重在评估单个训练样本之后更新，在小批量梯度算法中，权重在评估一个小批量训练样本之后个更新。权重的每一次改变都是随着损失函数的梯度的改变而改变的。因为 DNN 的尺寸比较

大，所以梯度的定量计算非常消耗算力，由此引出另一种算法——反向传播算法。反向传播算法使用链式法则来进行梯度的计算，这种方式既简便又快捷。

2.3 深度学习算法的结构

2.3.1 卷积层

卷积操作通过将一个k × k的窗(卷积核)在图片（特征图）上滑动[24]，从一个大的矩阵中得到一个小的矩阵，从而得到输出的特征值，用于滑动的特征值被称为权重或者参数。这些权重是在训练的过程中得到的。当卷积核在特征图上滑动时，首先和特征图上对应点的值相乘，然后乘积相加，得到一个结果。输入和输出形成了卷积层的一系列特征矩阵。

卷积操作从特征图的左上角开始进行，在随着不断移动滑窗的过程中重复操作，首先使将滑窗向右移动，直到到达特征图的右侧边缘。然后将滑窗下一行，从左向右进行滑动，知道整个特征图都被滑窗覆盖完之后才结束操作。这种滑窗运算方式被称为卷积。一般来说，卷积核的尺寸都非常小，通常小于等于11×11。每一组输入输出特征图都有与之对应的权重，权重值的个数与卷积核的尺寸相等。输出的特征图源于输入特征图的卷积运算之和。在相同的 CNN 模型中不同的卷积层尺寸不同。

如图 2-6[25]所示,卷积神经网络包含了四种层次的循环，输出特征循环(Loop-4),穿过特征图的循环(Loop-3)，沿单个输入特征图维度的循环(扫描操作，Loop-2)，卷积核窗口内部循环(累乘加运算，Loop-1)。卷积层是 CNN 中的主要部分，占据了 CNN 的 90%的运算，很多研究都致力于通过展开循环来加速卷积运算。循环展开将卷积运算并行度最大化，这种展开依赖于特殊的架构，这种架构包含大量的处理单元和寄存器阵列。

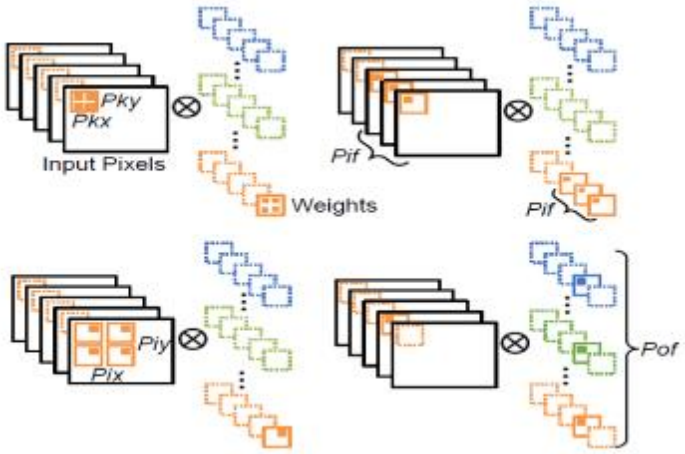

图 2-6 神经网络循环示意图[25]

2.3.2 激活函数

激活函数函数在卷积神经网络中的作用如同动物细胞中的神经元一样。当发出一个动作电位时神经元被激活。如公式(2-13)所示的 sigmoid 函数常被作为激活函数[26]。

$$f(x) = 1/(1 + e^{-x}) \tag{2-13}$$

其中 x 代表特征图的输入与权重相乘求和之后的结果，如果他是一个无穷大的数，那么 sigmoid 函数的结果接近于 1。如果 sigmoid 函数中 x 的大小为负无穷大，那么 sigmoid 函数接近于 0，如公式(2-14)所示为另一种常用的激活函数。

$$f(x) = \tanh(x) \tag{2-14}$$

以上所提到的 sigmoid 函数和 tanh 这些非线性函数都需要很长的训练时间。近些年来比较流行和常用的 CNN 激活函数是修正线性单元 (ReLU)，如公式(2-15)所示。

$$f(x) = \max(x, 0) \tag{2-15}$$

ReLU 激活函数在训练时速度更快，与 sigmoid 和 tanh 等函数比起来计算复杂度更低。此外，它不需要输入规范化来阻止它饱和。

2.3.3 Normalization 层

在现实生活中，有一种现象被称为侧向抑制，在这种现象中非常活跃的神经元拥有一种抑制相邻神经元的能力，这会是的这块区域的对比度变得很大。在 CNN 中，为了实现这个目的，通常使用局部归一化和简单归一化，尤其是在处理 ReLU 神经元时，因为他们当中存在不受控制的激活，这个需要归一化来处理[27]。它通过很大的相应检测出高频。如果我们在兴奋的神经元的局部领域标准化，与它的领域相比，它的灵敏性会变得更强。同时，它会抑制在任何区域同样大的反映。如果值很大，那么标准化会使他们都变得很小，所以它表现出了某种抑制作用，并且提高了神经元的活跃度。通过相邻的神经元，可以实现相同特征和相邻特征的归一化。

2.3.4 Pooling 层

池化层也被称作下采样层，用来减少所表示的空间大小，从而减少神经网络中的参数大小以及计算复杂度。池化层被周期的加入到连续的卷积层中间[28]。池化层对每一个深度切片进行操作，并且使用最大池化的方式来对其大小进行调整。最

常用的就是 2 乘 2 的最大池化，在 4 个数中选择最大的一个留下，其余的百分之 75 丢掉。除了最大池化以外，在 CNN 中也有很多的其他池化操作方式，例如平均池化和最小池化。

2.3.5 全连接层

卷积神经网络的结构通常是由一些卷积层和 ReLU 构成的栈，在这之后紧接着就是池化层了，在图片被缩小到一个很小的尺寸之前，网络的结构都是如此。在这之后是一个或者多个全连接层，这也被成为内积层，这个层的神经元全连接在前一层的所有神经元上，正如这个层的名字一样。最后一层是分类层，他输出了类的分数。

2.4 典型深度学习网络

深度学习算法是计算机视觉领域最有创新性的技术之一。2012 年，Krizhevsky 等人在计算机视觉领域著名的比赛——ILSVRC 中获奖，自此开始，卷积神经网络为人们所熟知。他们使用 AlexNet 将图像分类识别的准确率从 2011 年的 26%降到了 15%，这个进步是非常大的。ImageNet 是一个标准的数据集，可以用它来衡量图像分类以及图像识别算法的性能，其中包含了几百万张图片，这些图片分为几万种不同种类。CNN 图像识别和分类等计算机视觉领域有着更好的准确率。从 2013、2014、2015 年以来，在 ILSVRC 中图像的分类准确率从 88.8% 到 93.3%再到 96.4。很多公司例如 Google、Facebook 以及亚马逊等等开始将 CNN 作为他们的业务核心，这些业务包括图片搜索、目标检测、推荐算法等。但是 CNN 最典型的应用还是在图像视觉领域以及和语音处理领域。典型的深度学习 CNN 网络有 AlexNet、VGG、ResNets 等。

2.4.1 AlexNet

与以前的计算机视觉领域的网络例如 LeNet 相比，AlexNet 要更大。在 AlexNet 中有 6 千万个参数以及 65 万个神经元，训练过程需要在内存为 3G 的 GTX580 显卡上花费 5 到 6 天。到现在为止，即使数据非常大的情况下，有些非常复杂的神经网络依然能够在 GPU 上运行的很快。但是在 2012 年的时候，这些网络是非常庞大的。

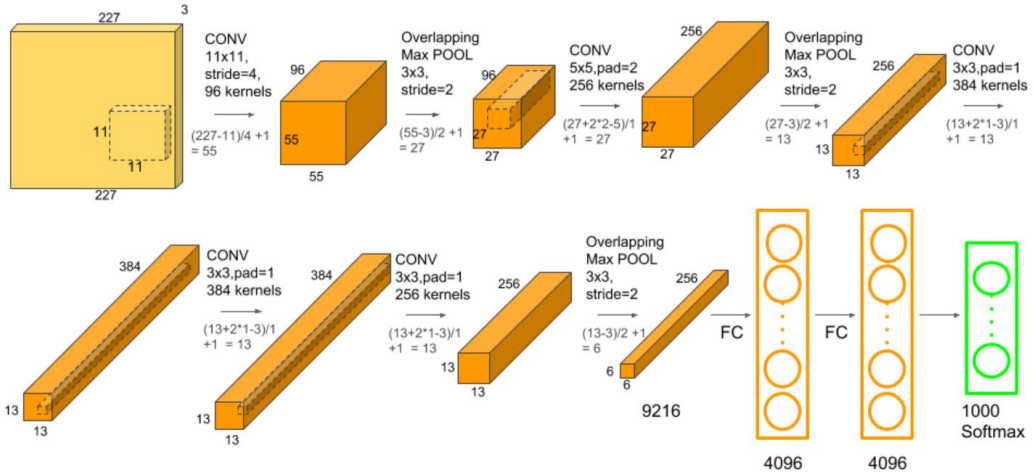

图 2-7 AlexNet 结构示意图[29]

如图 2-7[29]所示，AlexNet 中包含五个卷积层和 3 个全连接层。通过多个卷积核，可以提取出图像中感兴趣的特征。在单个神经网络层中，通常包含大量相同尺寸的卷积核。例如，AlexNet 的第一层，包含了 96 个尺寸为$11 \times 11 \times 3$的卷积核。需要注意的是，卷积核的高度和宽度是相同的，卷积核的深度和通道的数目也是相同的。最开始的两个卷积层之后，紧接着是两个相互重叠的最大池化层。第三个卷积层和第四个卷积层是直接相连的。在第五个卷积层后面是一个重叠的池化层，这个层的输出是两个全连接层的输入。第二个全连接层输出到 softmax 层中，这个 softmax 层中有 1000 个标签，这 1000 个标签代表 1000 个类。在所有卷积核全连接操作结束后，就需要非线性激活层来处理。在第一个和第二个卷积层和池化层之后紧接着的是一个非线性激活层。这个层之后是一个局部归一化的处理，局部归一化之后将数据输出到池化层进行处理，但是研究发现归一化的作用并没有那么大。

2.4.2 VGG16

第一个卷积层的输入是一个尺寸为224×224的 RGB 图像，如图 2-8[30]所示，这个图像经过一系列的卷积层处理，这些卷积层中使用的是3×3的卷积核。卷积核可以捕获到左/右，上/下以及中间的信息。在 VGG 的结构中，也会使用1×1的卷积核。这种卷积操作可以看作是将输入通道进行了一个线性变换。

卷积的步进为一个像素，卷积的输入通过填充来经过卷积操作之后，输出的尺寸不再改变。例如3×3的卷积采用 1 个像素的填充，如图 2-9 所示，在卷积层之后采用五层最大池化层。最大池化操作在2×2的窗口中进行，池化操作的步进为 2。在一系列全连接层之后是一个卷积层，这些全连接层的深度和架构都不同。前

面两个卷积层每个都有 4096 个通道，第三个包含 100 个通道，其中每个通道也代表一个类。最后一层是 softmax 层，VGG 中的全连接层结构都相同。所有隐藏层都有 ReLU 层，需要注意的是 VGG 中没有本地归一化，因为对于 ILSVRC 来说本地归一化不会提升性能。

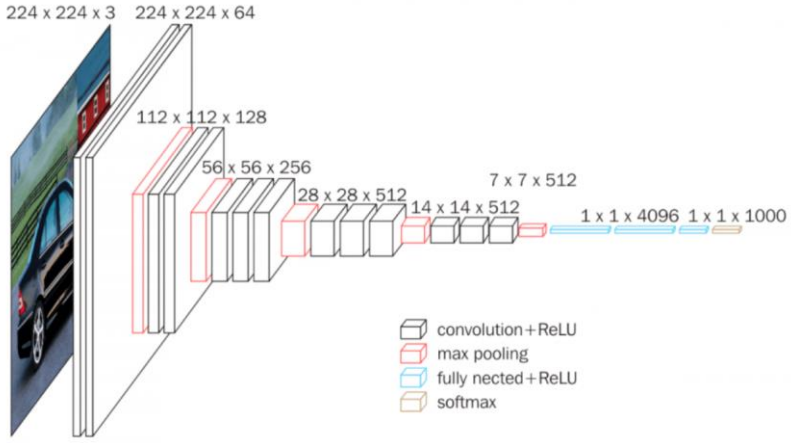

图 2-8 VGG16 结构尺寸示意图[30]

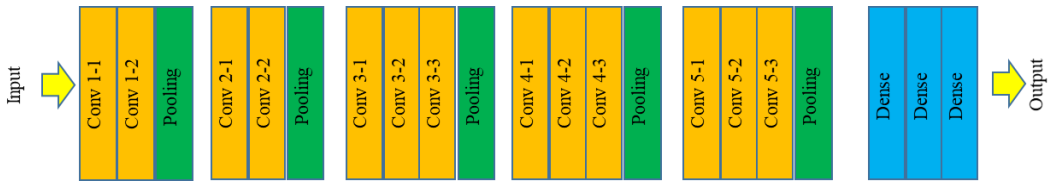

图 2-9 VGG16 结构层次示意图

VGG 各网络的结构参数如表 2-1[31]所示，可以看出其基本层包括卷积层、池化层、全连接层和 softmax 层，其中池化层采用的是最大池化方法。从表 2-1 中可以看出，此网络的配置方式有很多种，针对于不同尺寸的图片，其所适合的配置方式不同。

表 2-1 VGG 结构参数[31]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input(224*224 RGB image)					
conv3-64	con3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv2-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

2.4.3 ResNets

微软的研究发现，将深度神经网络拆分为三个层块，将输入数据输入到每一个块并且传递到下一个块，其结构如图 2-10 所示，块的剩余输出减去重新引入的块的输入,如图 2-11 所示。这些都可以减少信号消失的问题。对于学习算法来说并没有新的参数和其他的改变。换句话说，ResNets 将简单的深度神经网络分为多个块，然后通过跳过某些块或者是快捷的连接这些块来建立新的连接，从而构成一个更大的新的网络结构，其网络结构图如图 2-12 所示。

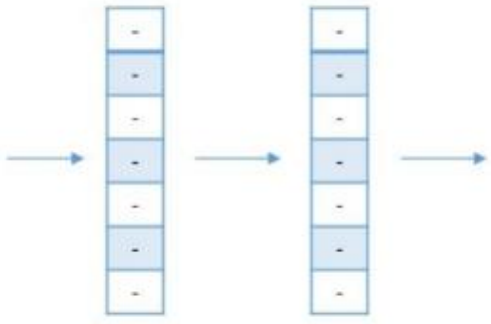

图 2-10 典型神经网络连接图

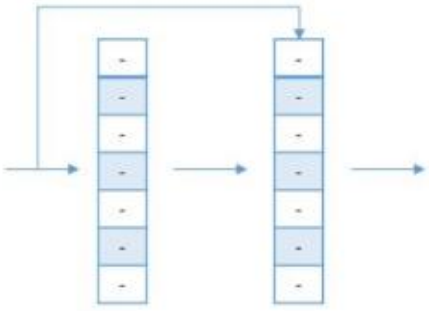

图 2-11 ResNets 连接图

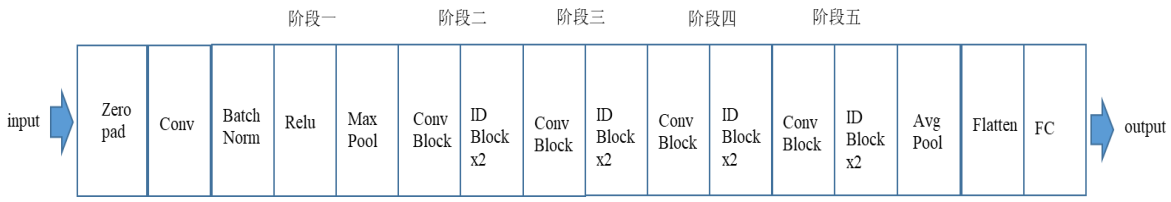

图 2-12 ResNet 结构图

2.4.4 GoogleNet

GoogleNet 是在 2014 年的 ILSVRC 比赛中首先进入人们的视野中的。GoogleNet 的结构与 VGGNet,ZFNet 以及 AlexNet 有很大的区别。GoogleNet 的具体结构如表 2-2[32]所示,它在网络的中间包含了1×1的卷积层。并且全局平均池化层在网络的最后使用，而不是在全连接层之后使用。这两种技术被称为”Network in Network(NIN)”。另一种技术叫做 inception module，指的是对于相同的输入采用不同类型或者是不同尺寸的卷积网络，并且将输出堆叠在一起。

表 2-2 GoogleNe 结构参数[32]

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

2.4.5 ZF Net

ZF Net 的关键点有三个，主要是反卷积、迁移学习以及模拟退火学习速率。在 2013 年时，这个架构曾经获得了 2013 年的 ImageNet 的冠军，它的错误率为 14.8%，相对于以前的 15.4%有了很大的改善。很多人容易将深度学习中的反卷积和信号 处理中的反卷积混淆，其实这两个是有很大的不同的。在深度学习中，反卷积也被称为转置卷积或者是部分步进卷积。反卷积是用在误差反向传播的过程中的，这个过程是通过卷积层来实现的。

卷积允许通过特定的输入到特定的输出。一个特定的卷积的输入通常是由以下参数来决定的：输入尺寸、卷积尺寸、步进以及填充值。但是反卷积的方向与卷积的方向的相反的：从输出维度到出入维度。卷积的反向传播是一个反卷积过程，这个过程是用于解决卷积的输入梯度问题的。如图 2-13[33]所示，如果将卷积操作看作是矩阵乘法的话，那么这个乘法的两个乘数是输入和矩阵C相乘，其中C是卷积的权重。然后计算反向传播的误差，其中反向传播的误差等于输入梯度乘以C的转置。需要注意的是乘上的是C的转置，转置意味着卷积核旋转 180 度。

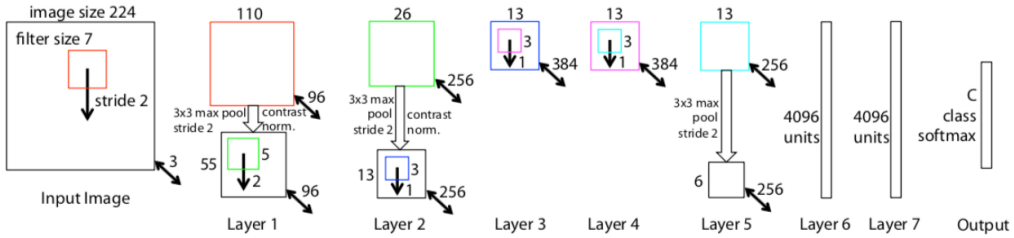

图 2-13 ZF Net 结构图[33]

总的来说，这个架构是 AlexNet 的升级版本，这个架构的发明者找出了 AlexNet 的瓶颈，并且进行了突破，实现了更好的性能。这个架构中将第一层的卷积核尺寸从11×11改为7×7，这样可以减少从第一层中获取的无效特征。这些无效特征对于系统来说是不需要的。这些特征都是单色图像，它们和其他的特征差别不大。除了改变卷积核的尺寸以外，ZFnet 还将卷积层中的卷积核的个数变成了双倍。与 AlexNet 相比，ZFNet 中的神经元数量是 AlexNet 中的神经元数量的两倍。在 AlexNet 中，神经元的数量为 48-128-192-192-128-2048-2048。在 ZFNet 中，神经元的数量是两倍，为 96-256-384-384-256-4096-4096。这个改变让输出的模型表示变得更复杂了。2012 年时，imagenet 比赛的准确率为 14.8%，到了 2013，由于有了这种优化方式，使得准确率达到了 15.4%，也就顺利成章的称为了冠军。

2.5 深度学习硬件实现

2.5.1 GPU 实现

GPU 是一种多核处理器，这种多核处理器最开始是用来进行并行图形处理任务的，图 2-14[34]为其发展历程，可以看出 GPU 的发展是日趋发杂的，而且针对不同领域也有所不同。但是现在 GPU 也用来进行一些通用的计算，这种也被称为 GPGPU，这种 GPU 上支持一些通用的计算框架，例如 OpenCL 和 CUDA。有一些高端的 GPU 例如英伟达的 GTX Titan 系列的 GPU，其中包含了超过 3000 个浮点运算处理核。这些核均可以工作在 1GHZ 的频率下，并且 330GB/的带宽。它们可以提供每秒 6600GFLOP 的算力，但是功耗也达到了 250W。嵌入式 GPU，例如英伟达 Tegra X1，这个也用于英伟达 Jetson TX1 核英伟达的 Drive PX 平台上。Tegra X1 中包含了 256 个处理核，这些核可以工作在 1GHZ 的频率下，并且提供约 25GB/S 的带宽。它可以提供每秒 512GFLOP 的算力，功耗为 10 瓦。GPU 非常适合 CNN 的并行计算，并且也有很好的深度学习框架支持。这些都为 GPU 在科研领域奠定了良好的基础。但是由于 GPU 的功耗问题导致 GPU 在推理端无法发挥更大的优势。

现在，人工智能正在完成着过去无法完成的任务，对资源的彻底理解会对执行造成很大的影响。GPU(Graphics Processing Unit)被认为是深度学习的核心，它也是人工智能的一部分。GPU 是一颗单芯片，这颗芯片被广泛的用于图形和数学计算，这样就可以让 CPU 有空闲时间做其它的事情。CPU 中通过少量复杂的核来运行多个独立的任务，这个通过多线程来完成。GPU 中则含有大量的简单的核，这些核同时进行运算，可以处理同时处理多个任务。在深度学习中，主要的代码运行在 CPU 上，但是 cuda 的代码运行在 GPU 上。CPU 在运算时将 3D 图形渲染和向量运算等任务分配给 GPU 来进行。CPU 用来执行优化过的并且耗时较长的复杂任务。GPU 有一个带宽瓶颈，将大量数据传输到 GPU 中耗时较长。GPU 的带宽是经过优化的，GPU 的延时也是经过了优化的。CPU 可以将少量的数据快速搬运到 RAM 中，而 GPU 则可以快速将大量的数据从内存中进行搬移。从一些参数数据中可以知道是否应该采用 GPU。是否采用 GPU，带宽是一个主要的原因。对于比较大的数据集来说，在训练模型的过程中，cpu 会花费大量的内存。对于独立的 GPU 来说，有专用的 VRAM 内存。这样，CPU 的内存就可以用来做其他的事情了。但是从 CPU 中搬运数据到 GPU 中是个巨大的挑战。在 CPU 中进行复杂的运算会消耗大量的时钟周期。这是因为 CPU 处理事务是串行执行的，而且 CPU 中只有少量的核。但是，虽然 GPU 的速度很快，但是 由于处理器的原因，大量的数据从 CPU 到 GPU 中会消耗大量的内存，这也会导致处理时间过长。最好的 CPU 带宽是 5GB/S,而最好的 CPU 的带宽是 750GB/S。

训练一个深度学习模型需要大量的数据集，所以大量的运算操作需要访问内存。为了提高运算速度，GPU 是一个很好的选择。运算量越大，更能凸显出 GPU 相对于 CPU 的优势。打个比方，当使用多个卡车运输货物时，就可以等待卡车以及装车的时间。因此，因为卡车装车需要更多的时间，所以装车时间(延迟)可以隐藏（线程级并行）。因为这个可以隐藏延时，所以 GPU 可以获取更高的带宽，而且在线程级的并行中，可以隐藏延时。然而，虽然 CPU 中也有多线程，但是和 GPU 中的多线程就没有太大的可比性了。CPU 中优化任务比 GPU 中更容易，即使 CPU 比 GPU 中核数更少，但是核的性能很强。每个 CPU 核都可以在不同的指令集(MIMD)下工作，因为 GPU 是由 32 个核组成的，所以可以并行的执行相同的指令。由于需要巨大的算力，所以在密集的神经网络中这种并行是非常困难的。所以相对于 CPU 来说，这种复杂的运算的部署在 GPU 中是更困难的。对于小网络和小数据集来说，如果在不考虑时间的情况下，可以考虑使用 CPU 而不是 GPU，而且 GPU 的功耗相对于 CPU 来说更高。

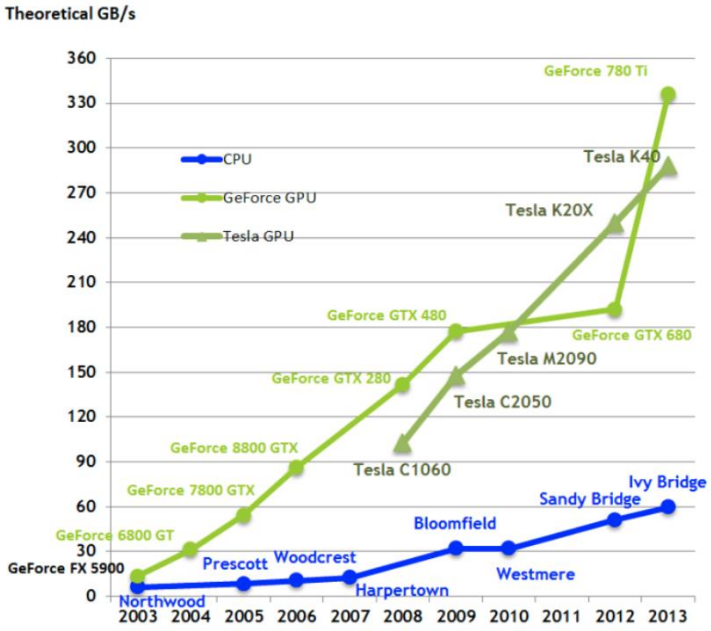

图 2-14 GPU 发展发展历程图[34]

GPU 有着比 CPU 更快的速度，因为 GPU 的高带宽、通过并行隐藏延迟以及 GPU 中有更容易编程的寄存器。由于这些原因，CPU 在训练网络的时候要求数据集要比较小才可以。但是 GPU 可以针对大型的深度学习网络进行长时间的训练。CPU 训练网络的速度相比 CPU 来说更慢，GPU 可以加速网络的训练。所以相对 CPU 来说 GPU 更加适合训练深度学习模型[35]。

GPU 是目前使用最广泛的深度学习硬件加速器之一，它被普遍的用来进行 CNN 的分类和训练。主要是因为 GPU 的高带宽以及高效率的浮点运算单元。但是，GPU 在运算的时候功耗非常大。所以，在大型云端服务器和电池供电的设备上使用 GPU 部署 CNN 是很困难的。此外 GPU 可以并行的处理大批量的图片数据。对于像视频流这种数据来说，输入图像需要逐帧的被处理，因为每一帧图片之间的间隔延时会对应用的结果有很大的影响。对于一些跟踪算法来说，一帧的结果会影响到下一帧的运算结果。所以对于 GPU 来说，推理端劣势还是非常明显的。

2.5.2 ASIC 实现

现如今，全球有超过 100 家公司在研发 AI 芯片，如图 2-15[36]所示，其中包括 ASIC(Application Specific Integrated Circuit)和 SOC(System on Chip)。这其中也不乏一些大公司开始研发 AI 芯片，包括 Google(TPU)、Facebook、Amazon(Inferentia)、Tesla 等等。一般都分为两类：

1、训练和推理。这些芯片既被用来进行 DNN 的训练，也被用来进行 DNN 的

推理。训练是计算密集型的任务，其中包含了求解梯度和反向传播，和训练比起来推理相对来说更加简单并且需要更少的计算。英伟达的 GPU，现在对于深度学习来说非常受欢迎，它既可以进行推理也可以进行训练。除此之外，也有一些其他的硬件平台，例如 Graphcore IPU、Google TPU V3 以及 Cerebras 等等。OpenAI 的大量数据显示，在训练大型时需要更强大的算力。

2、推理。这些 ASICS 用来运行在 GPU 或者其他的 ASICs 上已经训练好的深度学习模型，在训练的过程中要对被训练的网络进行一些修改，例如剪枝压缩等等，这样可以使得网络可以运行在不同的 ASICs（Google Coral Edge TPU,NVdia Jetson Nano 等等）上。大部分人认为深度学习的推理市场需求远远比训练的需求大。任何一个基于 ARM Cortex-M0,M3,M4 内核的微控制器(MCU)都可以进行推理。

研发任何芯片(ASIC,SOC 等)都是一件高成本、高难度以及耗时很久的事情，通常根据芯片的大小和复杂程度，一个芯片团队研发人员规模从 10 到 1000 人不等。Habana Goya 是一家初创芯片公司 Habana labs 的产品，其架构如图 2-16[36]所示，其中 GEMM Engine 时通用矩阵乘法引擎。矩阵乘法时 DNN 中的核心操作。卷积就是典型的矩阵乘法，全连接层就是前向矩阵乘法。TPC 是张量处理核，这个单元是实际进行乘法运算或者是累乘加操作的。局部存储器和共享存储器都是用 SRAM(静态随机可读取内存)和寄存器来实现的。Eyeriss 是 MIT 的团队研发的深度学习推理加速器，其架构如图 2-17[37]所示，并且已经经过了两代的迭代，分别是 Eyeriss V1 和 Eyeriss V2。另外，AI 芯片中比较有代表性的就是谷歌的 TPU 架构了，其结构如图 2-18[38]所示，图 2-19[38]所示为其布局图，从图中可以看出,由于 ALU 的结构是直接连接的，所以在运算的过程中不需要使用内存，信息直接由上一级传输到下一级，运算的延迟被大大减小了。

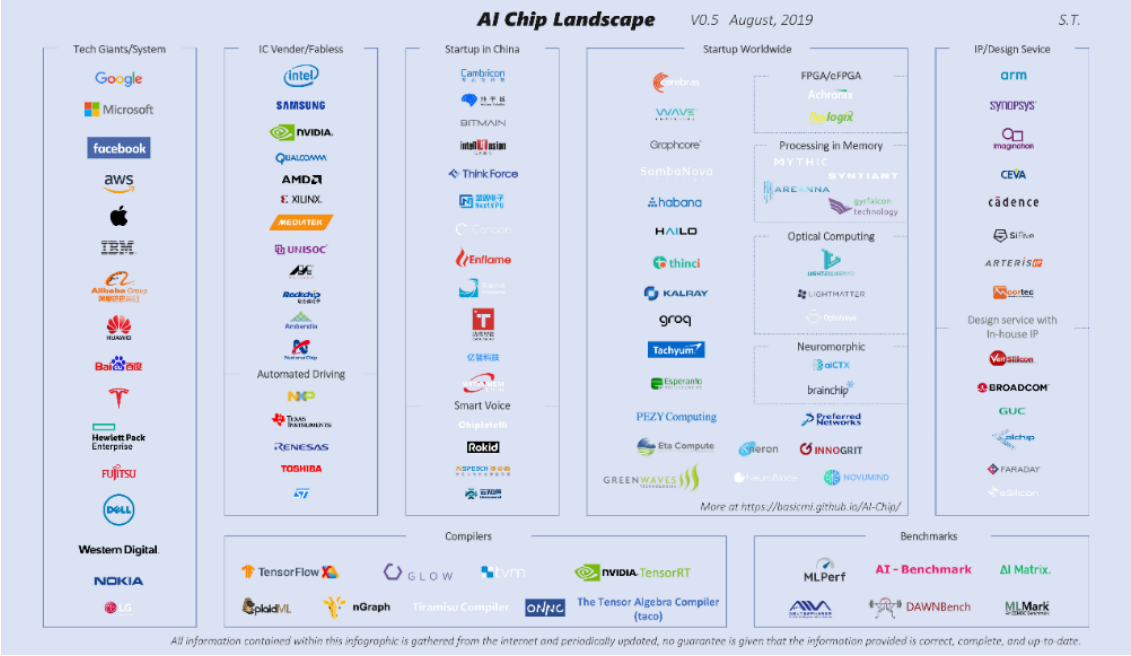

图 2-15 AI ASIC 厂家[38]

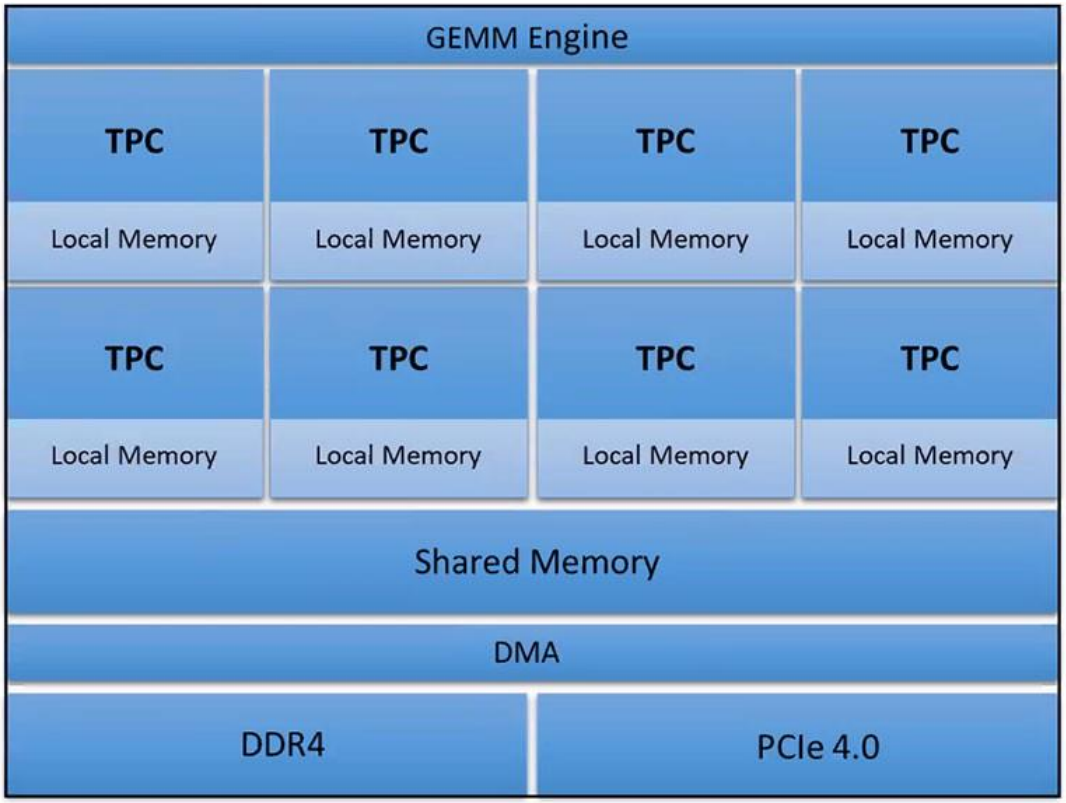

图 2-16 Habana 芯片架构[39]

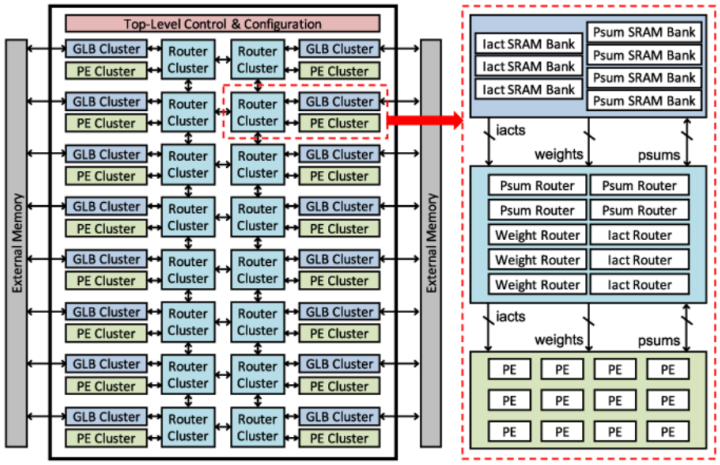

图 2-17 Eyeriss 芯片架构图[40]

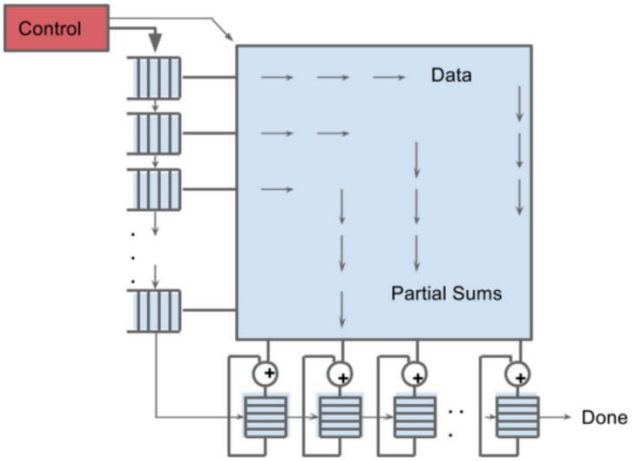

图 2-18 谷歌 TPU 芯片架构图[41]

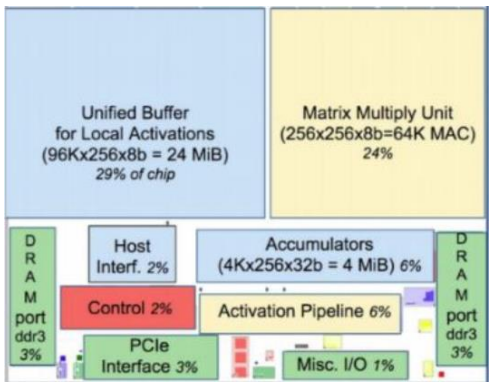

图 2-19 谷歌 TPU 芯片布局图[42]

2.5.3 FPGA 实现

如果要实现某种运算，通常使用的方法就是基于某种架构写出对应指令集的软件代码，然后再对应的硬件平台上来运行，对应的硬件平台一般有 CPU 以及 GPU 等。但是更加困难的是重新研发一种特殊的电路用来计算，然后在电路中运行特殊的指令集，这种运行方式与 CPU 以及 GPU 类似。如果你实现了这种电路，那么就需要通过一些办法来实现设计，从而进行运算。其中一种方式就是，将这种设计以电路的方式来实现，这种被称为专用集成电路，那么就需要消耗大量的资金。更简单的方式，是使用 FPGA，一种可重复配置的电路，你可以将 FPGA 配置为你所需要的电路。这和大部分编程者使用的基于指令集的硬件实现有很大的不同。基于指令集的硬件是通过软件来进行配置的，但是 FPGA 是通过专用硬件来进行配置的[43]。

当为智能驾驶或者是高频交易算法进行编程时，低延迟是一个必须要考虑的问题，应该减少从数据输入到进行处理之间的反应时间。在延迟方面，FPGA 比 GPU 要更好。这就是 FPGA 比 CPU 和 GPU 更适合的原因，GPU 还要使用 CPU 进行通信，否则无法在一些场景下进行使用。当使用 FPGA 时，可以将延迟控制在 1 微秒或者时 1 微秒之外，但是对于 CPU 来说，延迟低于 50 微秒性能就很好了。除此之外，FPGA 的延迟能够认为的控制，是更加明确的。原因就是 FPGA 的定制化更强，FPGA 不需要操作系统，内部也不需要像 CPU 一样通过总线(例如 USB 或者是 PCIE)来进行通信。

在 FPGA 中，你可以连接任何数据源，例如网口或者是传感器，直接通过芯片的引脚就可以连接。这就和 CPU 以及 GPU 形成了鲜明的对比，因为 CPU 和 GPU 与外界的数据交互需要使用标准总线进行连接，例如 USB 或者是 PCIE 等，只有通过操作系统才能将数据送入到应用中。FPGA 的直连技术可以为数据提供很高的带宽，同时也降低了时延。对于某些场景，例如射电天文(LOFAR 和 SKA)，高带宽是必须的。在这些场景中存在着大量的传感器，这些传感器采集了海量的信息。在这些数据被处理之前，需要进行进行数据压缩，将大量的数据压缩成小批量的数据以便后续处理。出于这种目的，荷兰的射电天文阻止 ASTRON 设计了 Uniboard 来进行射电天文的数据压缩，Uniboard 是基于 FPGA 的设备，它每秒钟处理的数据比阿姆斯特丹的交换网络都多[44]。

在高性能计算领域，例如在深度学习中，通常依赖于浮点运算单元，在浮点运算上，GPU 是很有优势的。在过去，FPGA 在浮点运算上是没有优势的，因为浮点运算单元必须由逻辑块组成，这会消耗大量的资源。但是随着 FPGA 的日益改进，

FPGA 中也加入了浮点运算单元，使得 FPGA 在浮点运算上也有了优势。当前可使用的速度最快的 GPU 是 Tesla V100,Tesla V100 的最大理论算力达到了 15 TFLOPS(衡量浮点运算速率的标准)，功耗是 250 瓦特。现在最好的 FPGA 的最大理论算力是 9.2TFLOPS，功耗是 225 瓦特。如果单从功耗效率来看的话，GPU 看起来更加节约功耗，因为从理论上来说，GPU 的能效比是 56GFLOPS/W。而 FPGA 的能效比是 40.9GFLOP/W。但是，不同的是，现在最新的 FPGA 中集成了浮点运算单元，使它的能效比更高。所以深度学习的边缘端大多采用 FPGA 来实现[45]。

在某些领域中，要绕开 FPGA 是非常困难的。在很多应用中，例如在制导系统中，因为 FPGA 的低延时的优势，FPGA 被广泛使用。在射电天文应用中，输入输出接口中使用 FPGA 来处理海量的数据。在现在非常火的加密矿机应用中，为了节省功耗，FPGA 也被用于进行顶点运算和逻辑操作[46]。

但是英特尔和赛灵思如今在 FPGA 中投入了大量的资金并不是为了做射电天文、制导以及加密货币，他们有更大的计划，他们的计划就是高性能计算领域，其中一个非常重要的领域就是深度学习领域。

2.6 本章小结

本章介绍了深度学习算法的发展历程，简要阐述了深度学习算法的特性和巨大优势。也对深度学习算法中的关键部分(卷积层、激活函数、Pooling 层、全连接层和 Normalization 层)做了详细的介绍，还介绍了几种主流的深度学习网络。文章节还对三种主流深度学习加速方案进行了对比，分析了三种实现平台的优缺点，最终确立了深度学习 FPGA 加速方案的优势。

第三章 YOLOV2 算法加速研究

3.1 YOLOV2 算法的基本理论

3.1.1 YOLOV2 概述

人的眼睛当看到图像时，只需要看一眼就可以知道图像是什么，图像的位置在哪里，以及图像中的物体的相互关系。人的视觉系统很快，而且非常的准确。这可以让我们在很短的时间内完成很复杂的任务，比如驾驶。使用快速精确的目标检测算法可以实现智能辅助驾驶甚至是自动驾驶，这种自动驾驶不需要太多的传感器，并且可以将信息反馈给用户。当然这种算法在通用领域，例如机器人这些领域也爆发出了很大的潜力。现在的很多目标检测系统都是根据分类算法改进的。为了检测到目标，系统将整个图像分为很多部分，然后对每一个部分进行分类。DFM 使用滑窗法，一个滑窗在图像上滑动，在窗口区域内进行分类。大部分算法都采用候选区域的方式创建一个原始的锚盒子，然后在对应的盒子中进行分类。在分类之后，对锚盒子进行后处理，从而消除重复检测。然后对锚盒子中的目标进行重新评分。这种复杂的算法很慢，而且很难被优化，因为这些部件被分开训练的。我们将目标检测的问题重新定义为了单一的回归问题。通过这种方式可以直接从图像像素中获得边界框的坐标值以及分类的概率值。使用 YOLO 我们可以从图像中得出目标物体的坐标以及类别[47]。

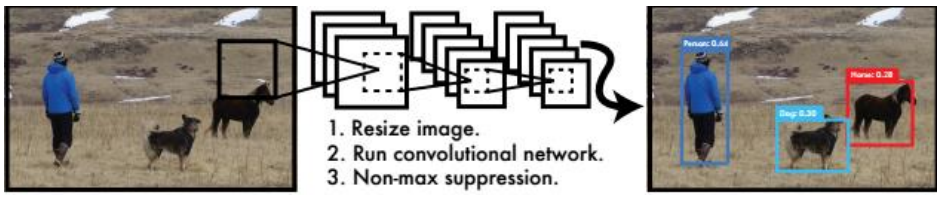

图 3-1 YOLO 效果图

YOLO 是非常简单直接的，如图 3-1 所示，通过 CNN 可以预测出目标框的坐标以及类别概率。YOLO 对整张图片进行训练并且优化目标检测的性能。与传统目标检测方法相比，这种方式有更多的优势。首先 YOLO 将目标检测问题重新定义为单一的回归问题，所以算法变得更加简介，速度也更加快，在 Titan X 的 GPU 上可以运行到 45 帧每秒的速度。这意味着在处理视频流时的延迟小于 25 毫秒。而且 YOLO 的性能是其他的目标检测的平均精度的两倍。和滑窗法不同的是，YOLO 在训练和测试时候是对整个图片进行操作的，所以可以直接检测出目标的种类以

及出现的位置[48]。

YOLO 学习的对象可以被概括的表示。当在自然图片上训练且在样本上进行测试时，YOLO 的性能比 DPM 和 R-CNN 等性能好很多。因为 YOLO 的概括能力更强，所以当输入未知量或者是新的图片时候，性能依然很好。通常目标检测的目标就是快速而精准的从大量物体中识别出目标物体。因为神经网络的存在，使得目标检测算法越来越精准。但是大部分目标检测算法都有局限性。与其他任务例如分类和标注这些任务相比，现在的数据集也有局限性。大部分目标检测的数据集都包含了大量的带有标签的图片。分类数据集包含了很多种类别的图片。我们都希望检测的规模更小一些。但是用于检测的样本图片比分类的训练样本图片成本更高。所以我们希望检测的数据集在规模大小能够像分类的数据集规模那样。

我们有一种新的办法来利用分类数据集，通过这种方法可以扩大分类数据集的使用范围。通过使用分层试图的方式，我们可以将各种各样的数据集结合起来。YOLOV2 可以使用检测和分类的数据集来进行训练。YOLOV2 通过使用分类图像进行训练，可以提高鲁棒性，从而精准的预测出物体的位置。通过这种方式来训练网络，可以检测到超过 9000 多种不同的物体。将 YOLO 改进为 YOLOV2 之后提升了算法的性能，然后采用 ImageNet 和 COCO 中数据集相结合的方式来训练 YOLOV2 的网络模型，可以训练出很好的目标检测算法。

3.1.2 算法原理

传统目标检测需要多个部件才可以检测到目标，YOLO 中只需要单一的神经网络就可以检测到目标了。通过这个网络可以预测图片中的每一个锚盒子。同时也可以检测到图片中的目标的类别。YOLO 中使用端到端的训练，并且在保证很高的精度的情况下也能保证实时性。如图 3-2[49]所示，系统将输入图像划分为$S \times S$的窗格。当某个物体的中心落入窗格的中间时，那么此窗格就负责检测该物体。每个窗格预测B个锚盒子，并且为这个盒子进行评分。这个分表示这个窗格中包含有物体的几率有多大，并且这个物体是目标物体的概率有多大。如果窗格中没有物体的话，那么信任度就是 0。

每个窗格中包含五个预测的结果信息：x, y, w, h以及信任度。窗格的中心坐标用(x, y)来表示。相对于整个图像的高度采用w, h来表示。置信度代表预测窗格和真是窗格之间的差距。每一个窗格代表一个条件概率。这些条件概率是窗格中包含目标物体的概率[50]。如公式(3-1)所示，在测试时候，将类的条件概率和单个窗格概率相乘，这个得分代表了窗格中包含物体的概率以及物体是目标的概率。

$$\Pr(Class_i|\text{Object}) * \Pr(\text{Object}) * IOU_{pred}^{truth}$$

$$= \Pr(Class_i) * IOU_{pred}^{truth} \tag{3-1}$$

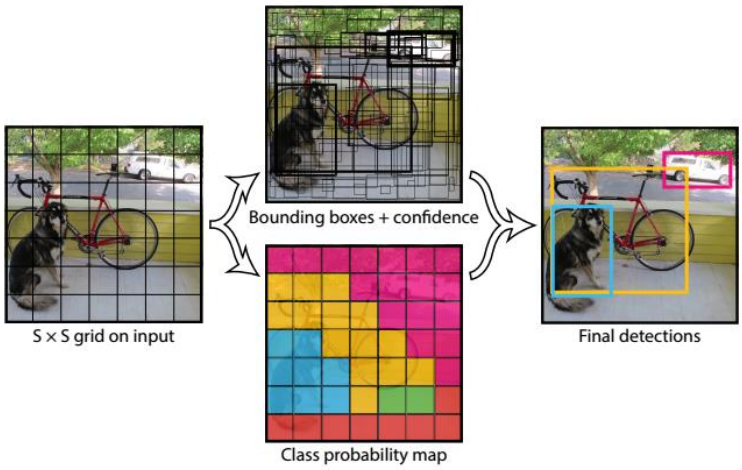

图 3-2 YOLO 原理图

系统将检测问题变为一个回归问题。它将图片分为$S * S$个格子，每个格子负责预测 B 个锚盒子。并且检测出 C 个种类。这个检测产生了$S * S * (B * 5 + C)$的张量。当使用 PASAL VOC 来评估 YOLO 时，我们使用 S=7，B=2，PASCAL VOC 有 20 个标签，所以 C 为 20。最终结果张量为$7 * 7 * 30$。

3.1.3 网络结构

这个算法是以卷积神经网络的方式来实现的，并且使用 PASCAL VOC 数据集来进行评估的。算法前面的卷积层用来从图像中提取特征，而最后的全连接层用来获取概率以及类别，从而输出。这个网络最初的创意来自于 GoogLeNet，GoogLeNet 在分类识别比赛中取得了非常好的成绩。YOLO 中有 24 个卷积层以及两个全连接层，其网络结构如表 3-1 所示。在 GoogLeNet 中使用的是 inception 模块，在 YOLO 中则不是采用的这种方式，而是采用 1*1 的约简层以及 3*3 的卷积层。YOLOV2 中有一些原创工作，也有一些是从其他地方借鉴的。改进之后的 YOLOV2 性能非常好，并且也在最新的检测数据集例如 PASCAL VOC 和 COCO 上检测过。通过多尺度的训练之后，YOLOV2 可以运行不同的尺寸的图片，但是需要在精度和速度之间做出权衡。精度和速度二者不可兼得，例如在 VOC207 上以每秒 67 帧的速率运行时，其精度只有 76.8mAP，但是以每秒 40 帧的速率运行时，其精度可以达到 78.6mAP，这个精度甚至比 FasterR-CNN 和 SSD 的性能更好。

表 3-1 YOLO 网络结构参数表

Tpye	Filters	Size/Stride	Output
Convolutional	32	3× 3	224× 224
Maxpool		2× 2/2	112× 112
Convolutional	64	3× 3	112× 112
Maxpool		2× 2/2	56× 56
Convolutional	128	3× 3	56× 56
Convolutional	64	1 × 1	56× 56
Convolutional	128	3× 3	56× 56
Maxpool		2× 2/2	28 × 28
Convolutional	256	3× 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3× 3	28 × 28
Maxpool		2× 2/2	14 × 14
Convolutional	512	3× 3	14 × 14
Convolutional	256	1× 1	14 × 14
Convolutional	512	3× 3	14 × 14
Convolutional	256	1× 1	14 × 14
Convolutional	512	3× 3	14 × 14
Maxpool		2× 2/2	7 × 7
Convolutional	1024	3× 3	7 × 7
Convolutional	512	1× 1	7 × 7
Convolutional	1024	3× 3	7 × 7
Convolutional	512	1× 1	7 × 7
Convolutional	1024	3× 3	7 × 7
Convolutional	1000	1× 1	7 × 7
Avgpool		Global	7 × 7
Softmax			

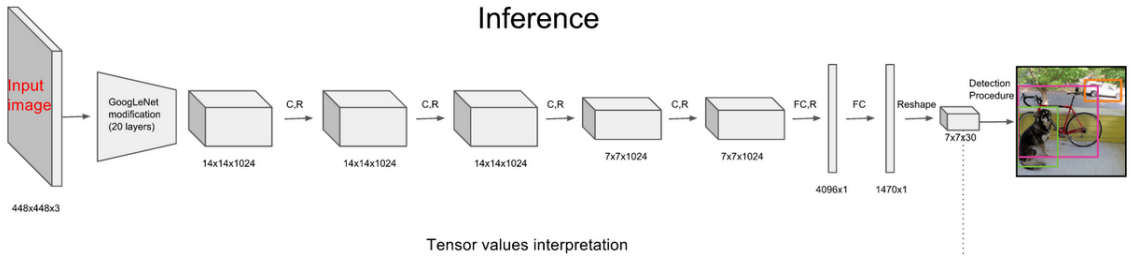

图 3-3 YOLO 结构图[51]

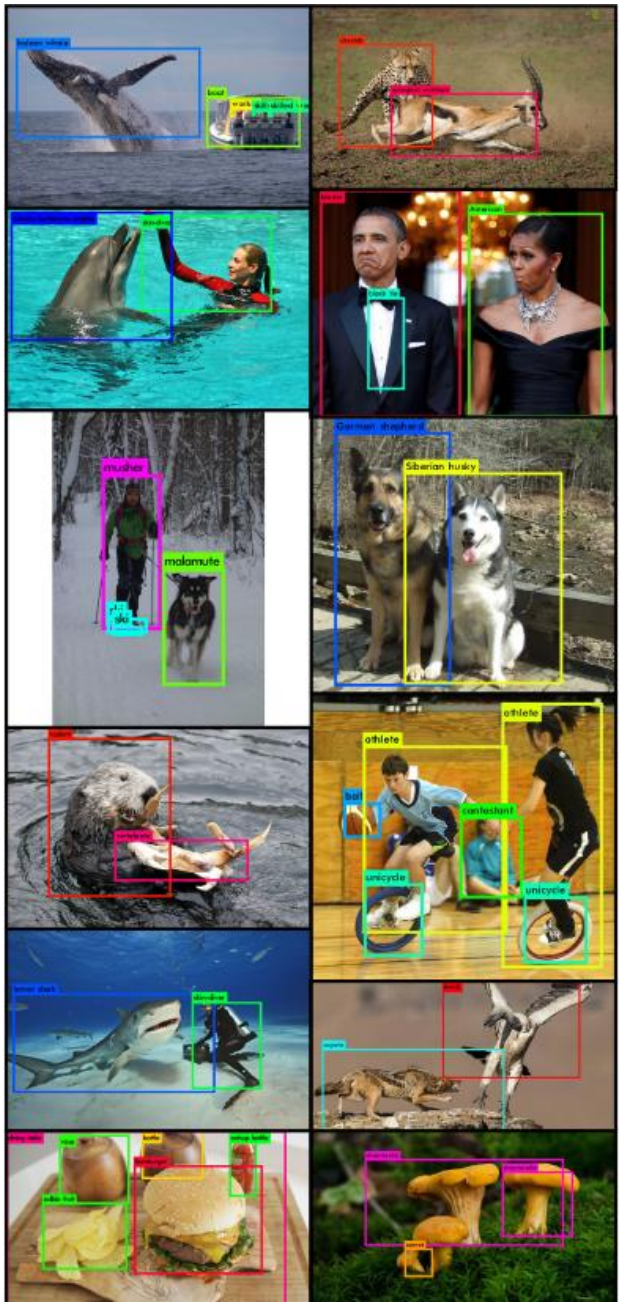

图 3-4 YOLO 检测效果图

如图 3-3[51]所示，YOLOV2 在 YOLO 的基础上做了改进，YOLOV2 中使用的训练方法是将分类和检测融合在一起来进行的。使用这种方式可以同时在 COCO 检测数据集和 ImageNet 分类数据集上进行训练。YOLOV2 在 ImageNetz 上的准确度是 19.7mAP，虽然只从 200 个类中检测 40 个物体。YOLOV2 在 COCO 数据集上的准确度为 16mAP，如图 3-4 所示，YOLO 可以检测超过 200 类物体，检测物体超过 9000 种，并且它的实时性也很强。

3.1.4 网络特点

YOLO 和最新的目标检测算法相比有很多缺点。从错误率上来看，和 Fast R-CNN 相比，YOLO 的目标检测位置错误率更高。和基于区域检测的目标检测算法相比，YOLO 在召回率上处于明显的劣势。所以，在保持分类精度较高的情况下，应该着重于提高位置准确率和召回率。计算机视觉的目标是更深更大的网络。更好的性能通常需要有更大的网络，或者是将不同的模型结合起来。但是 YOLOV2 的最终目标是取得速度和精度的平衡。传统方式优化网络都是将网络的层次加深，让网络变得越来越大，从而改善网络的性能。如表 3-2 所示为 YOLOV2 与 YOLO 的性能参数对比图，可以看出 YOLOV2 是通过精简网络，然后让有代表性的特征更容易被识别。从过去的研究中产生一些新的构想，将 YOLO 改进为 YOLOV2，并且提升了性能。

表 3-2 YOLO 与 YOLOV2 性能对比表

	YOLO								YOLOV2
batch norm?		√	√	√	√	√	√	√	√
hi-res classifier?			√	√	√	√	√	√	√
convolutional?				√	√	√	√	√	√
anchor boxes?				√	√				√
new network?					√	√	√	√	√
dimension priors?						√	√	√	√
location prediction?						√	√	√	√
passthrough							√	√	√
multi-scale?								√	√
hi-res detector?									√
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	78.6

在需要消除其他正规化的时候，采用批处理的方式可以很好的改善收敛性。在对 YOLO 中所以的层进行批处理化之后，可以将 YOLO 的性能提高 2%mAP。批处理也可以对 YOLO 模型进行归一化处理。在过度拟合的时候，通过使用批处理的方式，可以从模型中移除 dropout。对于所有先进的检测算法来说，都需要经过 ImageNet 的训练才可以得到。开始的时候使用的模型是 AlexNet，这个网络只能对尺寸小于256×256的图片进行检测。但是从 YOLO 开始，就采用224×224的图片进行训练，在推理的时候就可以使用448×448的图片进行检测，可以对尺寸为 448 的图片进行检测。这就意味着，在训练目标检测的同时，网络需要切换到新的输入分辨率才可以。对于 YOLOV2 来说，要将448×448的图片分为 10 个 epochs，

对分类网络进行微调。通过这种方式，可以让网络有调整的时间，在调整时间中，可以对网络的卷积部分进行调整，经过调整之后，网络就可以在高分辨率的输入中更好的工作。在这之后，需要对检测的结果层进行微调。这样处理之后，对于一个高分辨率的网络，它的精度可以提升 4%mAP。

在经过卷积层提取特征之后，YOLO 通过全连接层来获取 bounding box 中的坐标。YOLO 中是直接预测坐标的，但是在 Faster R-CNN 中却不是这样的，Faster R-CNN 中是手动通过先验概率来预测 bounding box。在只使用卷积神经网络的情况下，Faster R-CNN 可以为锚盒子预测偏移和置信度。与预测坐标相比，预测偏移将问题简化了，并且让网络训练变得更加容易。将全连接层从 YOLO 中去除之后，使用锚盒子来预测 bounding box。在去除池化层之后，可以增加网络卷积层的分辨率。YOLOV2 中通过将网络的输入图像大小从448×448减小到416×416。这样做的目的是希望在特征图中得到一个奇数的位置，这样就可以找到单个的中心格子。一个物体，尤其是较大的物体，通常会占据图像的中心，所以我们希望能够在图像的中心获得单个的中心窗格，而不是在中心有四个中心窗格。YOLO 的卷积层有下采样的功能，其下采样的因子是 32，所以通过下采样之后，分辨率为416×416的图像在经过卷积层之后会输出一个13×13的特征图。

在使用锚盒子时，我们可以从空间位置中解耦出目标检测机制，而不是对于每一个锚盒子都进行类和物体的检测。使用 YOLO，目标检测可以测出实际位置的 IOU，目标的锚盒子和预测出的类的条件概率指示出物体是否存在。使用锚盒子使精度有了小幅度的提高。YOLO 平均在每张图片中只预测出 98 个盒子，但是每个盒子可以预测超过 1000 多种物体。在没有锚盒子的情况下，模型的精度为 69.5mAP，此时召回率为 88%。虽然 mAP 得到了提高，但是召回率的增加意味着我们所设计的模型还需要进一步提升。

在 YOLO 中使用锚盒子时有两个问题。第一个是盒子的尺寸需要手动挑选。网络可以学习去调整盒子的大小，但是如果我们实现选择适合的大小时，网络可以更容易的检测到物体了。处理手动去选择之外，我们可以使用 k-means 聚类的方式来训练 bounding box 的数据集，从而自动的找出更好的参数。如果我们使用标准的 k-means 方式，那么相对于欧几里得距离较小的盒子来说，欧几里得距离较大的盒子就会产生更多的错误。但是，我们的目标就是得到更多的 IOU 分数，IOU 分数和盒子的大小是不相关的。如图 3-5[52]所示，在使用 k-means 算法时，我们通常使用不同的 K 值，并且使用最近的中心去计算 IOU。

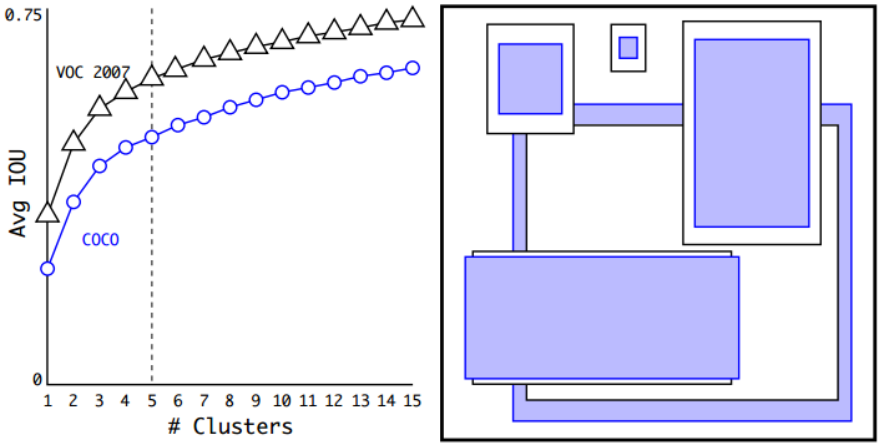

图 3-5 YOLO 中 K 值选择曲线图[52]

从图中可以看出，当使用 5 的时候，可以很好地满足召回率和模型复杂度二者的平衡。这种得到聚类中心的方式和手动处理锚盒子的方式有很大的不同。从形状上来看，盒子中短粗的盒子很少，相比之下，长且细的盒子增加了很多。下图中是两种方式获取的 IOU 的对比图，这两种 IOU 分别对应的是集群方式和手动处理锚盒子。从图中可以看出，有五个先验时，采用集群方式获得的模型 IOU 为 61.0，当手动处理锚盒子时候，需要 9 个先验才能达到 IOU 为 60.9 的水平。使用集群方式处理的时候，当先验为 9 时，可以看到平均 IOU 更高。可以看出，当使用 9 个矩形重心时，IOU 可以达到更高。从表 3-3[53]可以看出，当采用 k-means 方式生成锚盒子时，生成的模型具有更好的性能，并且更容易学习。

表 3-3 Box 参数选取表[53]

Box Generation	#	Avg IOU
Cluster SSE	5	58.7
Cluster IOU	5	61.0
Anchor Boxes	9	60.9
Cluster IOU	9	67.2

在 YOLO 中使用锚盒子时会有另一个问题，就是模型不稳定。这种不稳定尤其是在早期迭代过程中表现明显。造成这种不稳定的原因大部分来自于预测框坐标（x，y）的这一个过程，网络中的框的中心坐标计算方式如公式(3-2)和公式(3-3)所示，其中 x 为横坐标，y 为纵坐标。

$$\mathrm{x} = (t_x * w_a) - x_a \tag{3-2}$$

$$\mathrm{y} = (t_y * h_a) - y_a \tag{3-3}$$

从公式(3-3)可以看出，当$t_x = 1$时，框将会向右移动一些距离，这个距离就是锚盒子的宽度。同样，当$t_x = -1$时，就会把框向左移动一个锚盒子的距离。这个公式是没有约束的，所以在图像中的任何地方都能够预测锚盒子的位置。由于初始化是随机的，所以需要很长的时间才能够使模型稳定下来，并且有稳定的补偿值预测结果。

有一种方式是预测偏移量的方式，但是我们没有采用这种方式，而是采用YOLO 中的预测窗格中的位置坐标的方式。这种方式使真实值保持在 0 和 1 之间。我们采用逻辑激活的方式，从而来约束网络，从而使其预测值在范围之内。

在特征图中，一个网格中可以预测出五个锚盒子。对于每一个锚盒子来说，预测出了五个坐标，分别是t_x, t_y, t_w, t_h, t_o。公式(3-4)、公式(3-5)、公式(3-6)、公式(3-7)、公式(3-8)为其计算公式，如果图像相对于左上角的偏移是(c_x, c_y)且锚盒子对于高度和宽度先验概率是p_w和p_h。

$$b_x = \sigma(t_x) + c_x \tag{3-4}$$

$$b_y = \sigma(t_y) + c_y \tag{3-5}$$

$$b_w = p_w e^{t_w} \tag{3-6}$$

$$b_h = p_h e^{t_h} \tag{3-7}$$

$$\Pr(\text{object}) * \mathrm{IOU}(\mathrm{b}, \text{object}) = \sigma(t_o) \tag{3-8}$$

在算法中，因为我们约束了位置的预测，所以模型中的参数更容易学习到，这样也使模型变得更加简单。在预测窗格中心位置时，通过使用维度聚类的方式，可以将 YOLO 的性能提升 5%，这个比直接使用锚盒子的性能更加优越。

改进的 YOLO 可以预测尺寸为13×13的特征图，因为这对于大的物体来说是很适用的，这得益于对于局部对象的细粒度特性。Faster R-CNN 和 SSD 都运行的是分辨率大小不同的特征图，从图 3-8 可以看出，这些算法与 YOLOV2 相比有很大的差距，其具体数据如表 3-4 所示。如图 3-7 所示，YOLOV2 中采取了另一种方式，就是通过添加一些 passthrough 层来将特征从之前的分辨率为26×26的层去引入。

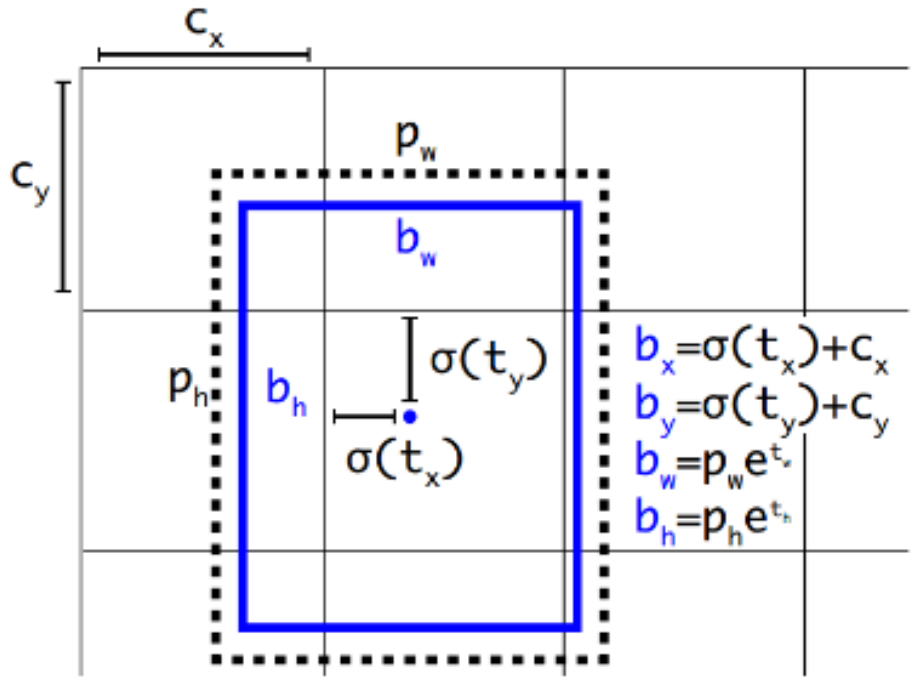

图 3-7 YOLOV2 中 bounding box 结构图

表 3-4 YOLOV2 性能对比表

Detection Frameworks	Train	mAP	FPS
Fast R-CNN	2007+2012	70.0	0.5
Faster R-CNN VGG-16	2007+2012	73.2	7
Faster R-CNN ResNet	2007+2012	76.4	5
YOLO	2007+2012	63.4	45
SSD300	2007+2012	74.3	46
SSD500	2007+2012	76.8	19
YOLOV2 228 × 228	2007+2012	69.0	91
YOLOV2 352 × 352	2007+2012	73.7	81
YOLOV2 416 × 416	2007+2012	76.8	67
YOLOV2 480 × 480	2007+2012	77.8	59
YOLOV2 544 × 544	2007+2012	78.6	40

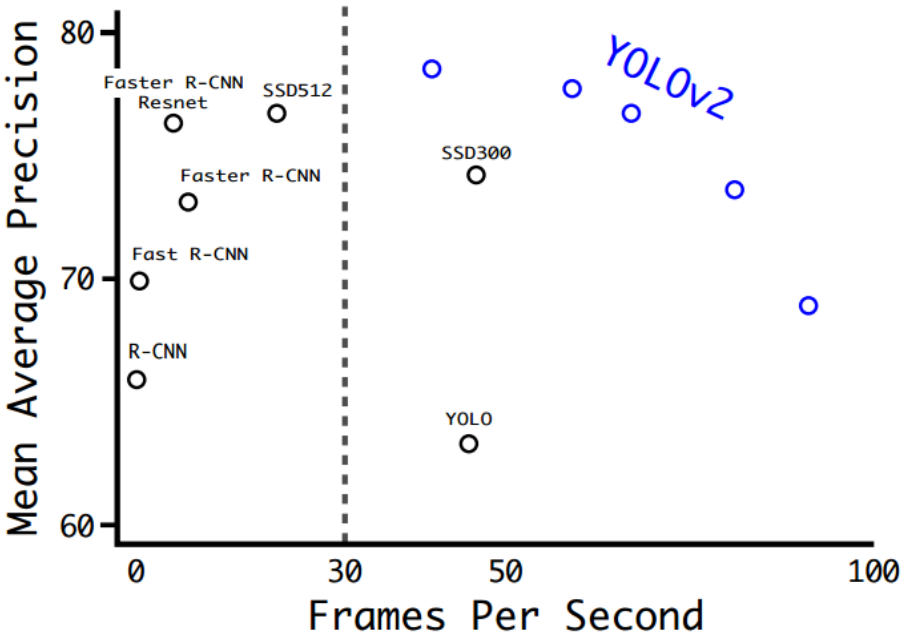

图 3-8 YOLOV2 与其他算法性能对比图

3.2 网络数据的量化

在神经网络的计算中，数据量是十分巨大的，这些数据主要来源于卷积层的输入输出数据，输入和输出数据都是特征图或者是原始图片。另外一些数据是在网络训练的过程中得到的模型的 weight 和 bias，这些数据决定着模型的性能。从图 3-9 中可以看出，在运算过程中，一方面要保证模型能够正确的得出预测结果，另一方面也要保证模型的速度能够更快。在 FPGA 中进行深度学习运算时，数据宽度较大时不仅会使运算速度变小，而且在数据进行搬移的过程中会造成带宽过大，整个系统的工作速度就会减慢。所以在保证运算精度的情况下，应该尽可能让数据的宽度变窄，加快算法系统的运算速度。离散卷积计算方式如公式(3-9)所示，通过对 YOLOV2 中的网络结构进行分析可以发现，第一个卷积层中的输入数据就是原始图片的归一化之后的数据，其尺寸是 416*416。

$$(f * g)(n) = \sum_{\tau=-\infty}^{\infty} f(\tau) g(n-\tau)_n \tag{3-9}$$

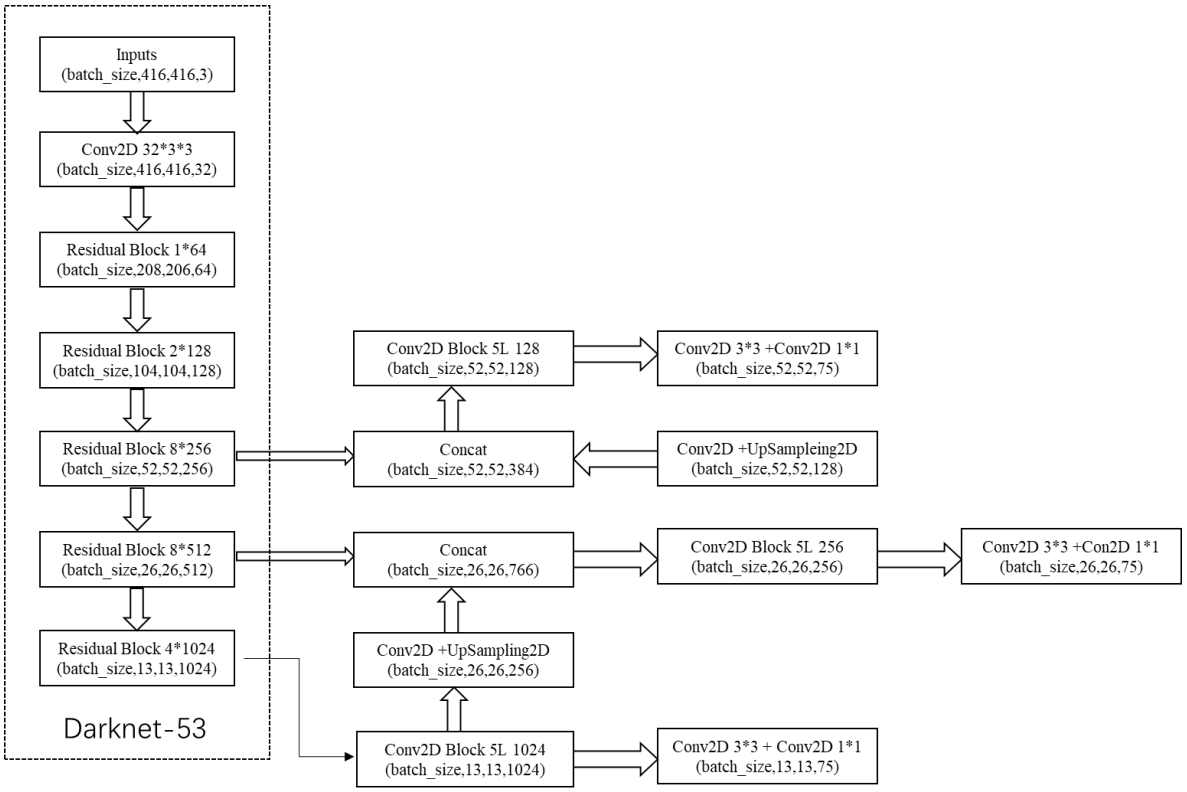

图 3-9 YOLOV2 运算流程图

3.2.1 量化权重数据

权重是卷积核中的重要参数，对权重进行量化时，既要保证运算的精度也要保证运算的速度。从图 3-10、3-11 和 3-12 可以看出，卷积层 0 的权重值分布略集中，主要是集中在 0 值附近。根据散点图中的点分布可以看出，可以将权重数据量化到-127 到+127 之间，可以保证实际误差较小。通过量化之后的频率分布图对比可以看出，量化前后权重频率图基本一样，量化没有对精度造成太大影响。同理，从图 3-13、3-14 和 3-15 可以看出，卷积层 1 的量化依据是合理的，量化前与量化后数据精度没有发生下降。通过两个频率分布图可以看出，量化对权重的频率影响可以忽略不计。从前两级卷积层来看，采用从-127 到 127 范围的量化，可以满足网络对精度的要求，而且通过将位宽量化之后，可以让网络在运算过程中的带宽和速度得到提升。所以网络中的权重数据都按照此种方式来进行量化。

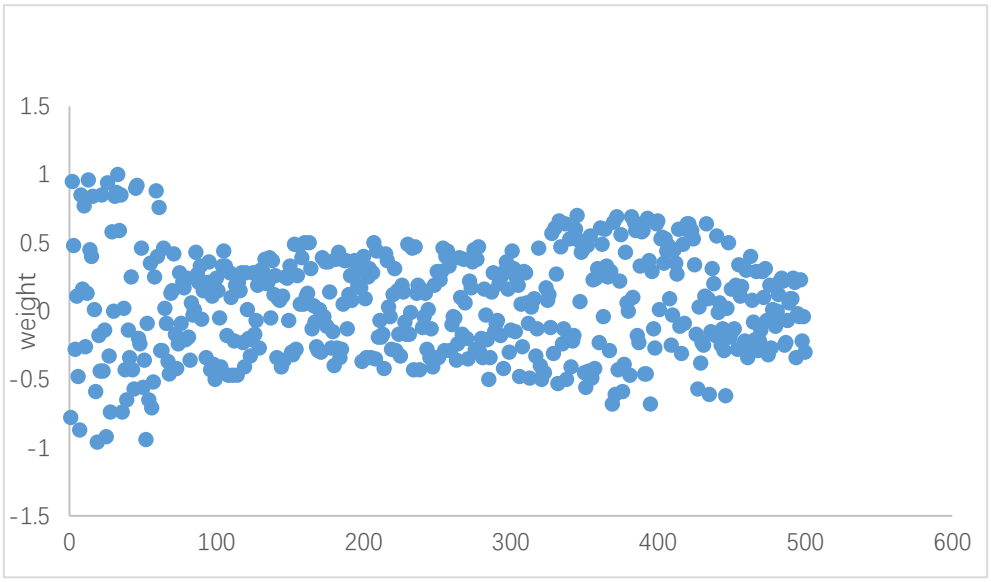

图 3-10 卷积层 0 量化前权重分布图

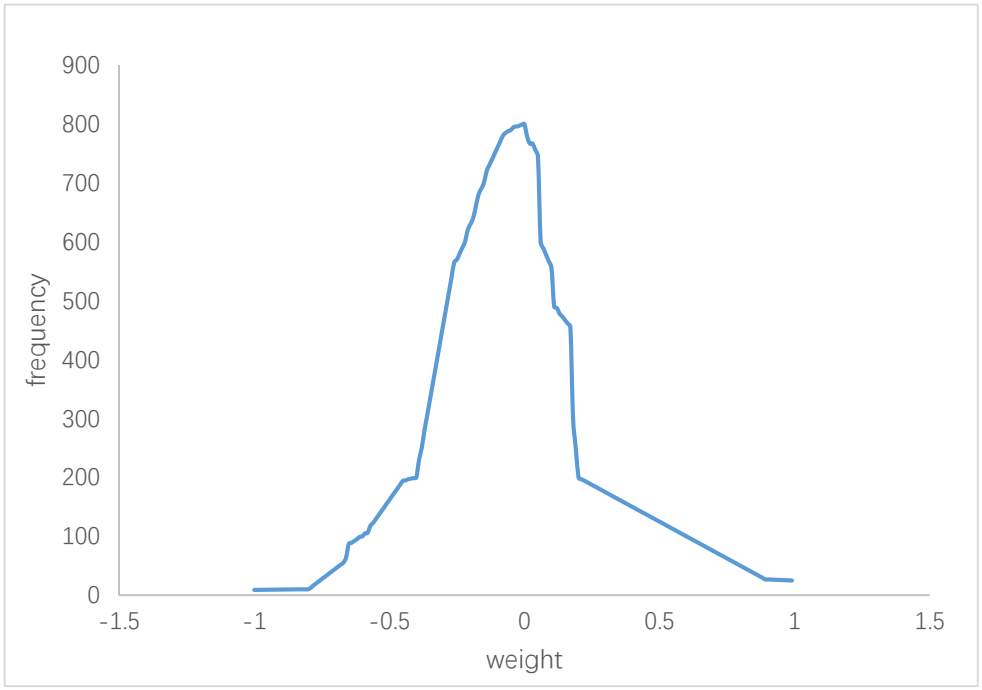

图 3-11 卷积层 0 量化前权重数据频率分布图

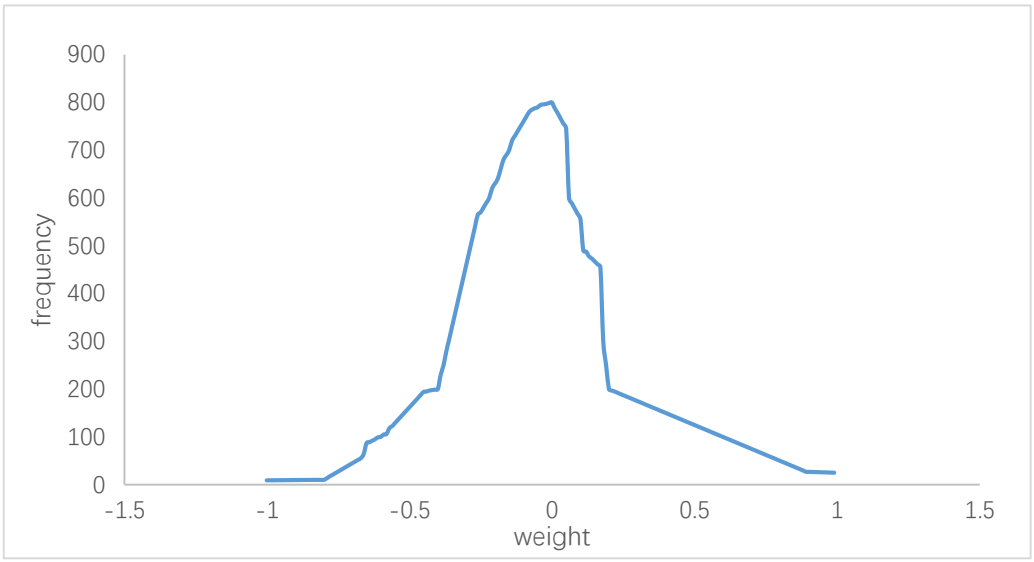

图 3-12 卷积层 0 量化后权重数据频率分布图

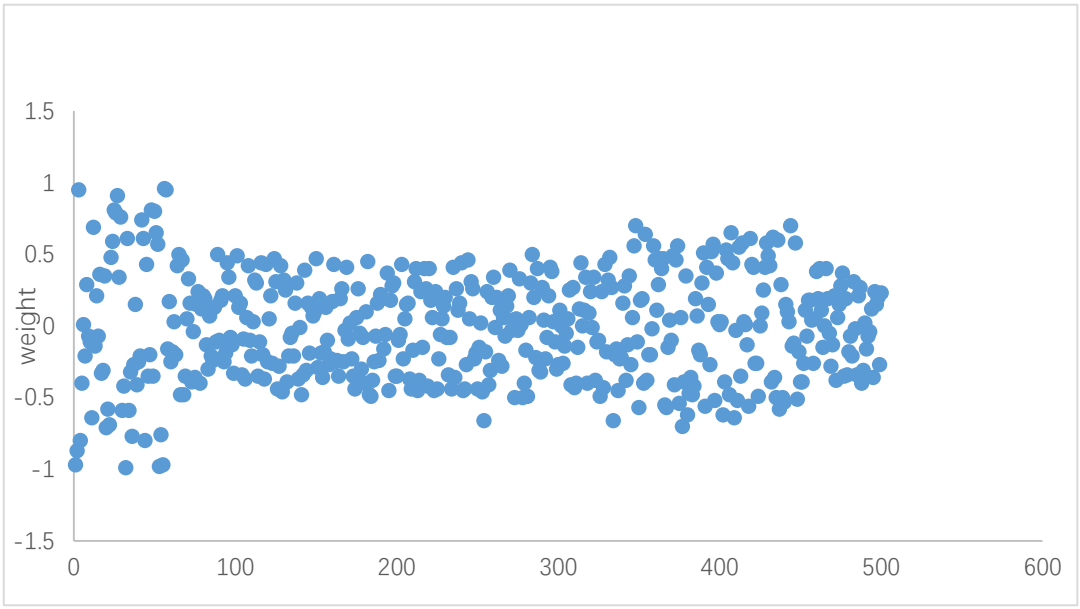

图 3-13 卷积层 1 量化前权重分布图

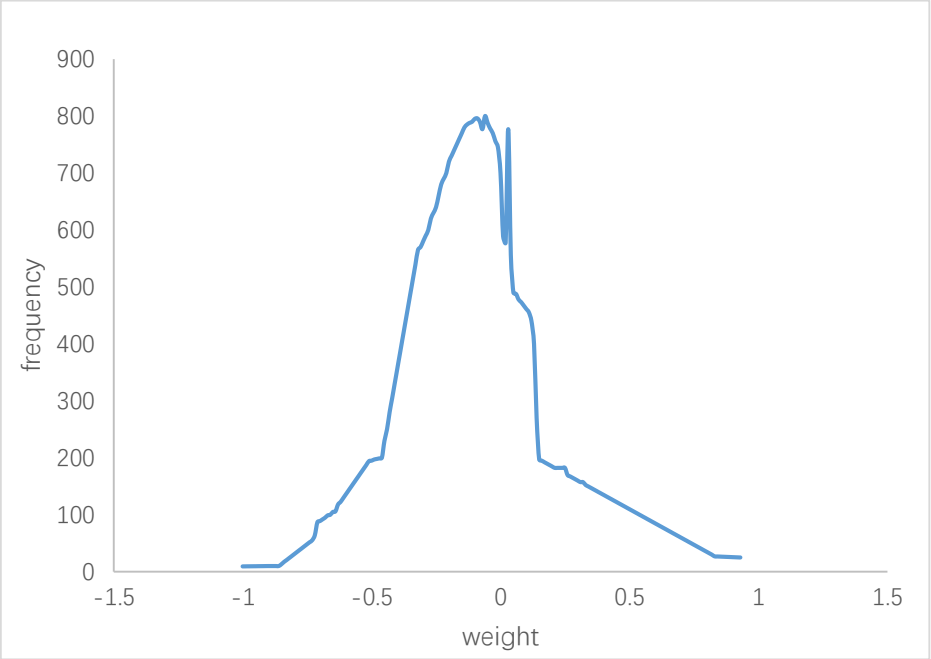

图 3-14 卷积层 1 量化前权重频率分布图

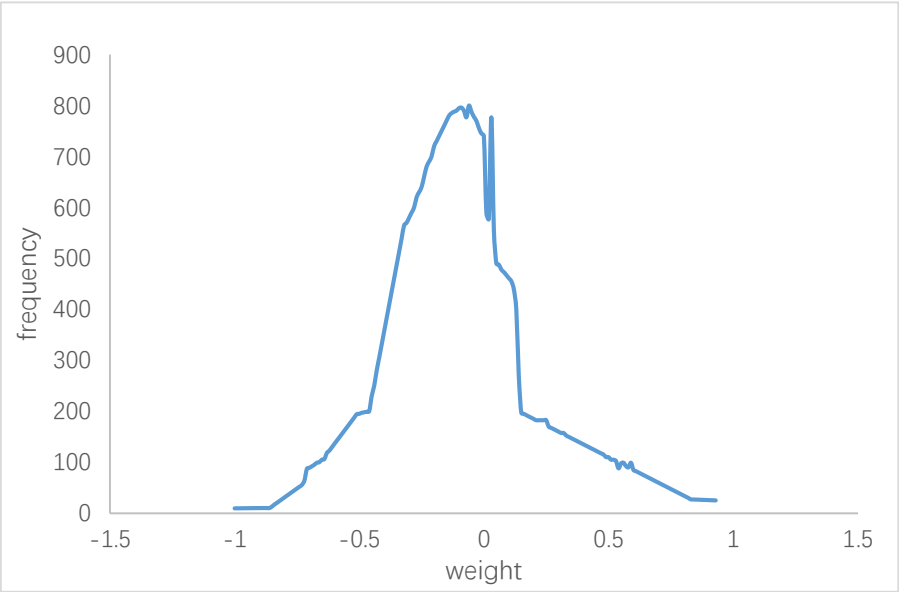

图 3-15 卷积层 1 量化后权重频率分布图

3.2.2 量化偏移数据

偏移数据和权重数据共同构成了一个网络，所以偏移数据对于网络来说也十分重要，所以在量化偏移数据时也需要考虑量化对数据分布的影响。图 3-16 所示为偏移数据的分散点图，通过此图与 3-17 和 3-18 综合对比来看，采用从区间[-127：127]的量化方式可以满足数据要求，综合来看采用与权重相同的量化方式是可行的。

网络中的其它层均可以采用这种方式，但是量化的区间需要依据数据的分布来进行选择。

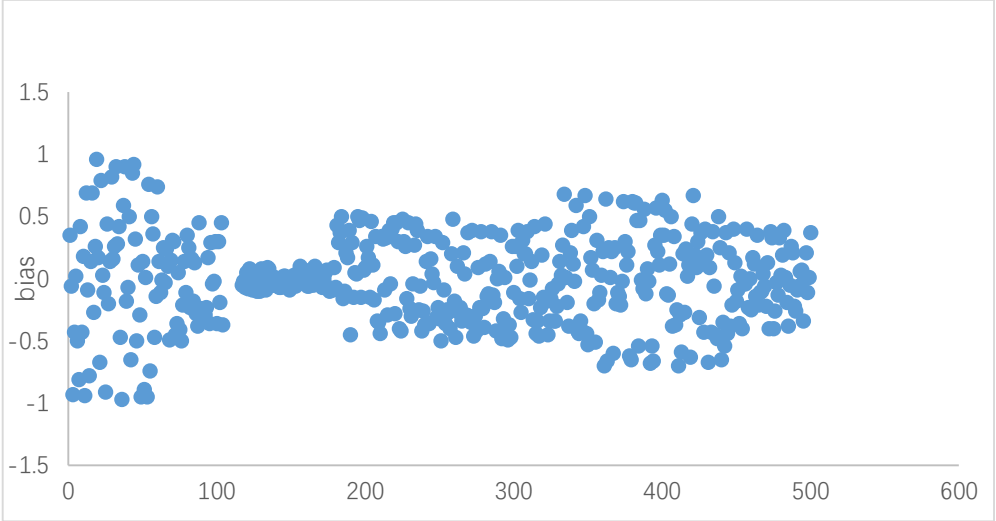

图 3-16 卷积层 0 量化前偏移数据分布图

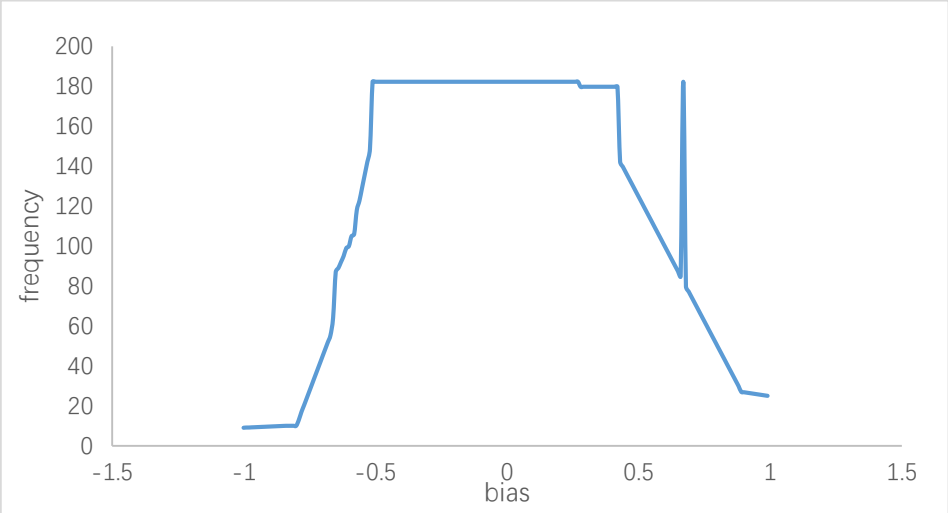

图 3-17 卷积层 0 量化前偏移数据频率分布图

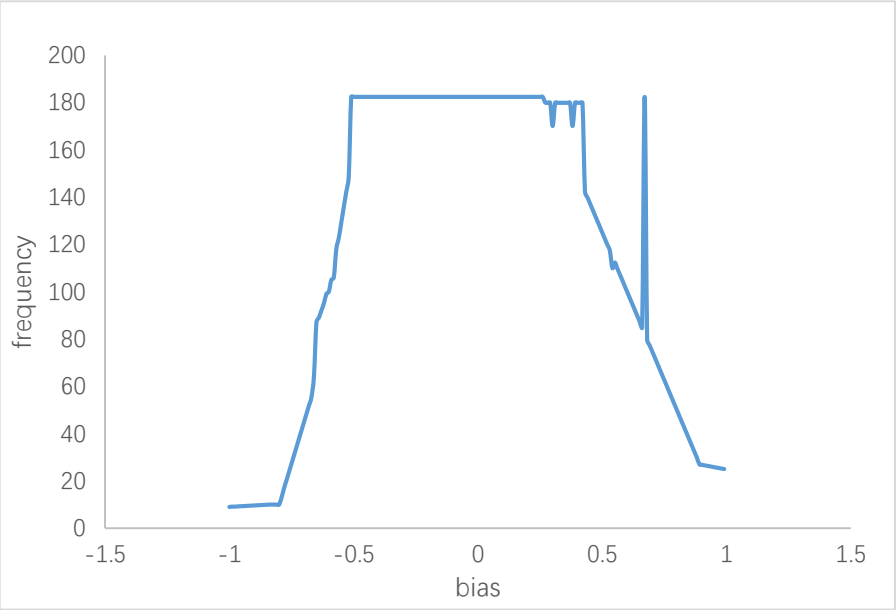

图 3-18 卷积层 0 量化后偏移数据频率分布图

3.3 硬件架构优化

3.3.1 运算结构优化

对于卷积和池化的运算来说，由于是矩阵运算，所以消耗的算力十分巨大。而且由于卷积和池化运算都是流式运算，所以将它们放在 FPGA 中实现从而进行加速是一个很好的选择。如表 3-5 所示，对于 YOLOV2 中部分运算，例如计算坐标，图像的预处理等等，都是一些非流式运算，或者是标准运算，这些运算适合放在 ARM 中进行。所以将整个 YOLOV2 中的运算做一下划分，可以充分利用软硬件协同的优势，从而对整个算法进行加速。将卷积层和池化层放在 FPGA 中运行，因为这两种层都是流式运算，这两种运算都适合在 FPGA 中进行加速。而对于最后一层 Softmax 来说，并不是流式运算，所以用 FPGA 来运算并不合适，所以在 ARM 中进行运算。另外，对于输入图像来说，还需要进行一些图像的预处理，这些由于 ARM 中有对应的软件栈支持，这些软件栈中的相应算法都进行过优化，十分稳定，直接进行调用即可。

表 3-5 YOLOV2 软硬件划分表

Type	Filters	Size/Stride	Output	HW/SW
Convolution	32	3×3	224×224	HW
Maxpool		$2 \times 2/2$	112×112	HW
Convolution	64	3×3	112×112	HW
Maxpool		$2 \times 2/2$	56×56	HW
Convolution	128	3×3	56×56	HW
Convolution	64	1×1	56×56	HW
Convolution	128	3×3	56×56	HW
Maxpool		$2 \times 2/2$	28×28	HW
Convolution	256	3×3	28×28	HW
Convolution	128	1×1	28×28	HW
Convolution	256	3×3	28×28	HW
Maxpool		$2 \times 2/2$	14×14	HW
Convolution	512	3×3	14×14	HW
Convolution	256	1×1	14×14	HW
Convolution	512	3×3	14×14	HW
Convolution	256	1×1	14×14	HW
Convolution	512	3×3	14×14	HW
Maxpool		$2 \times 2/2$	7×7	HW
Convolution	1024	3×3	7×7	HW
Convolution	512	1×1	7×7	HW
Convolution	1024	3×3	7×7	HW
Convolution	512	1×1	7×7	HW
Convolution	1024	3×3	7×7	HW
Convolution	1000	1×1	7×7	HW
Avgpool		Global	1000	HW
Softmax		Global	1000	SW

3.3.2 缓存优化

深度学习算法中需要使用大量的数据，其中包括权重数据和偏移数据以及输入输出数据。这些数据需要大量的带宽，但是由于这些数据在运算之前是存在于内存之中的，读取时候需要在内存和运算单元之间构建一个桥梁，这个桥梁就是OCM(On chip memory)，所以在运算的过程中需要进行下图所示的缓存优化。

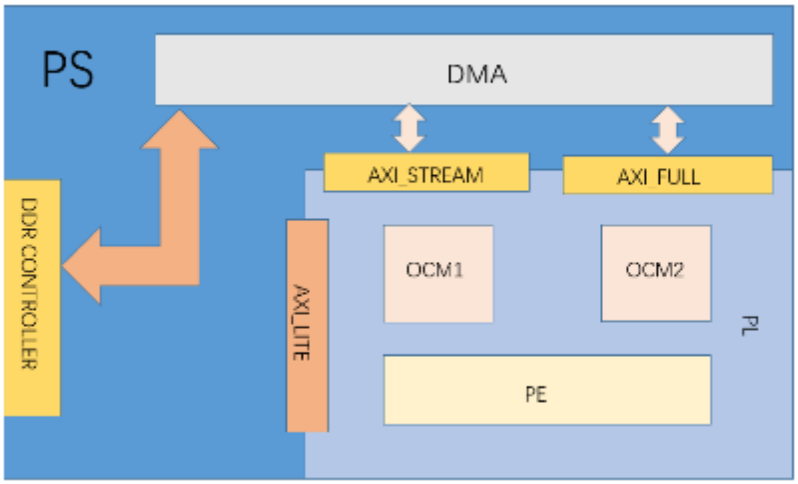

图 3-19 数据传输存储架构图

如图 3-19 所示为数据传输存储架构图，从图中可以看出，数据通过 DRAMC 之后可以通过 DMA 的方式进行数据访问，从而加快数据的相应速度，也可以将 PS 端的 CPU 从数据搬移任务中解脱出来，充分发挥 ARM 端 CPU 的性能。另一方面 PL 端在片上使用 BRAM 搭建了 OCM，从而可以搭建一个高速与低速期间的桥梁，满足运算的带宽需求。

3.4 本章小结

本章介绍了 YOLOV2 的算法原理和发展，重点介绍了网络结构，也介绍了网络特点。本章对 YOLOV2 的网络数据进行量化，包括权重数据和偏移数据的量化，保证了运算的性能和数据传输的带宽。还依据 YOLOV2 的结构制定了特定的微架构优化方案，优化主要集中在运算结构和缓存两个方面。

第四章 硬件加速实现与结果分析

4.1 FPGA 基本结构

FPGA 内部组成单元是可编程逻辑模块，这些模块之间可以根据用户的需求按照规则灵活的配置连接。如图 4-1[54]所示，和传统的 PLD 相比，FPGA 中的结构主要有三个部分：CLB(Configurable:可配置逻辑模块)、IOB(IO Block:输入/输出模块)以及 IR(Interconnect Resource)。FPGA 功能是由 SRAM 中的数据类配置的。

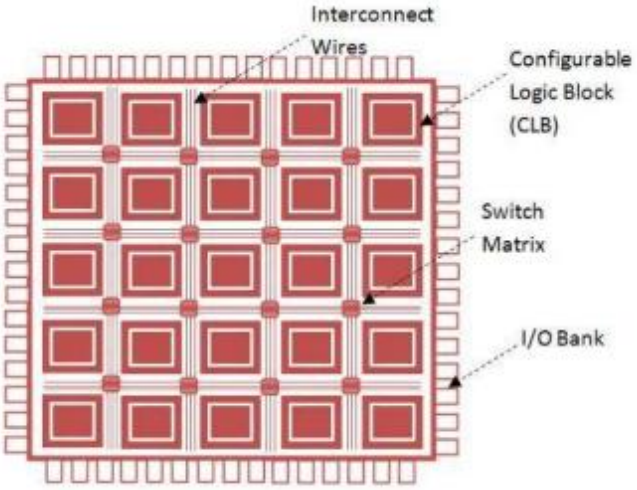

图 4-1 FPGA 结构图[54]

迄今为止，大部分 FPGA 芯片中都是采用查找表结构，这些查找表大部分采用 SRAM 来实现的。FPGA 中组合逻辑是用小型 LUT 来实现的，这些 LUT 的输出端连接到 D 触发器的输入端，D 触发器再连接到其他逻辑电路或者是驱动 IO 来对其进行驱动。这种连接结构构成了基本的逻辑单元模块，这种逻辑单元模块既可以完成组合逻辑运算也可以完成时序逻辑功能。这些模块时间通过金属连线进行连接，和外部通过 IO 口进行连接。FPGA 中的逻辑是通过加载编程数据来实现的，这些编程数据通过内部静态存储单元来进行加载。存储单元中的值可以配置逻辑单元各个模块的连接通路以及逻辑单元所实现的功能，也可以配置 IO 的功能以及电气属性等等，这些最终构成了一个可以实现目标功能的 FPGA 系统。由于 SRAM 是易失单元，所以在这个过程中，数据首先被存放在 FPGA 外部的存储芯片中，当芯片上电时数据首先被加载到 SRAM 中，然后 FPGA 被配置为我们所要的功能，然后 FPGA 可以进行工作。

与 DSP 以及 MCU 相比，FPGA 的运算速度更快，实现控制功能更加灵活。与

传统的 CPLD 相比，FPGA 更适合做一些规模更大、逻辑更加复杂的设计。FPGA 器件有如下特点：

(1) FPGA 有六部分构成：分别是可编程 CLB、可编程 IO、布线资源、嵌入式 RAM、专用硬核以及内嵌功能模块等六个部分构成。而 CPLD 的功能相对来说更加简单，其组成部分主要是：可编程 IO、基本逻辑单元、布线 pool 等构成。

(2) FPGA 由于其独特的结构，使得它更容易实现时序逻辑，但是对于 FPGA 实现组合逻辑来说，如果组合逻辑比较复杂的话需要几个 CLB 才能实现。CPLD 中包含有大量的与或阵列，所以 CPLD 更适合实现大规模组合逻辑。

(3) 由于 FPGA 中逻辑颗粒较小，导致 FPGA 中的连线资源非常丰富。另一方面，CLB 块非常小，所以其利用率非常高。CPLD 中由于其基本单元较大，所以会出现浪费，导致利用率过低。另一方面，针对连线来说由于宏单元是通过通道来连接的，所以灵活性和容量都有限。这两种原因导致 FPGA 的利用率比 CPLD 高。

(4) 由于 FPGA 是非连续式布线，所以 FPGA 的引脚的延时时无法预测的。对于 CPLD 而言，恰好与 FPGA 相反，CPLD 采用的是连续布线方式，这种方式导致 CPLD 的引脚的时延是可以预测的。

(5) 和专用集成电路(ASIC)相比，FPGA 比 ASIC 更加灵活，并且FPGA 的开发周期比 ASIC 更短，整个系统在出现 BUG 之后可以随时进行维护。一个复杂的系统使用 FPGA 之后可以将器件的种类降至最低，也可以将设计的成本降低到最低。由于 FPGA 的特殊结构，使用 FPGA 实现设计也能保证设计的保密性和可靠性。

4.2 ZCU104 开发平台介绍

4.2.1 ZCU104 硬件平台介绍

本文中实现的设计采用 Xilinx Zynq UltraScale+ MPSoC 系列 FPGA 芯片实现。如图 4-2 所示，Zynq UltraScale+ MPSoC 中的 PS 部分集成了四个核的 64 位 ARM Cortex-A53 应用处理器、Cortex-R5 双核实时处理器、多核 GPU 等硬核。对于性能来说，和 28nm 器件所设计的类似系统相比较而言，Zynq UltraScale+ MPSoC 中所构建系统的性能比前者强 2~5 倍。在安全性、保密性、可靠性以及电源控制的创新型上面具有更强大的优势。至于异构性能方面更是优势明显。如下图所示，Zynq UltraScale+ MPSoC 集成了 ARM 和 FPGA，ARM 端被称为是 PS 部分(处理器系统部分)，FPGA 端被称为是 PL 部分(可编程逻辑)。PS 部分的 ARM 处理器是四核 64 位的 ARM Cotex-A53 处理器，此处理器的运行最高频率为 1.5GHz。PL 端也有大量的 I/O 口和可编程逻辑资源，这样的资源配置更有利于实现高性能、低功耗以及

高复杂性的系统。此设计中使用 Xilinx ZCU104 开发套件完成系统实现与测试，此套件中的主控芯片为 Zynq UltraScale+ XCZU7EV-2FFVC1156 MPSOC。在 FPGA 中经常会出现片上 SRAM 存储器资源不足的问题，而 XCZU7EV 芯片上具有很丰富的 Ultra RAM 资源，刚好可以弥补这一不足。在设计中使用较大的本地存储器模块时，可以提高设计的系统性能，降低系统的功耗。XCZU7EV 中还具有很多 debug 接口，例如 USB、JTAG 以及 ARM Trace 这些接口，这些接口是用于 debug 使用的。ZCU104 开发套件中的芯片资源模块如图 4-3 所示，其资源十分丰富，本设计中所使用的板卡实物图片如图 4-4 所示。另外，Xilinx 还提供了用户体验十分良好的开发工具，vivado design suite 以及 PYNQ 开发框架，便于用户快速实现设计，验证以及调试。此开发方式广泛应用于智能监控、智能驾驶、无人机和生物医学成像等领域。

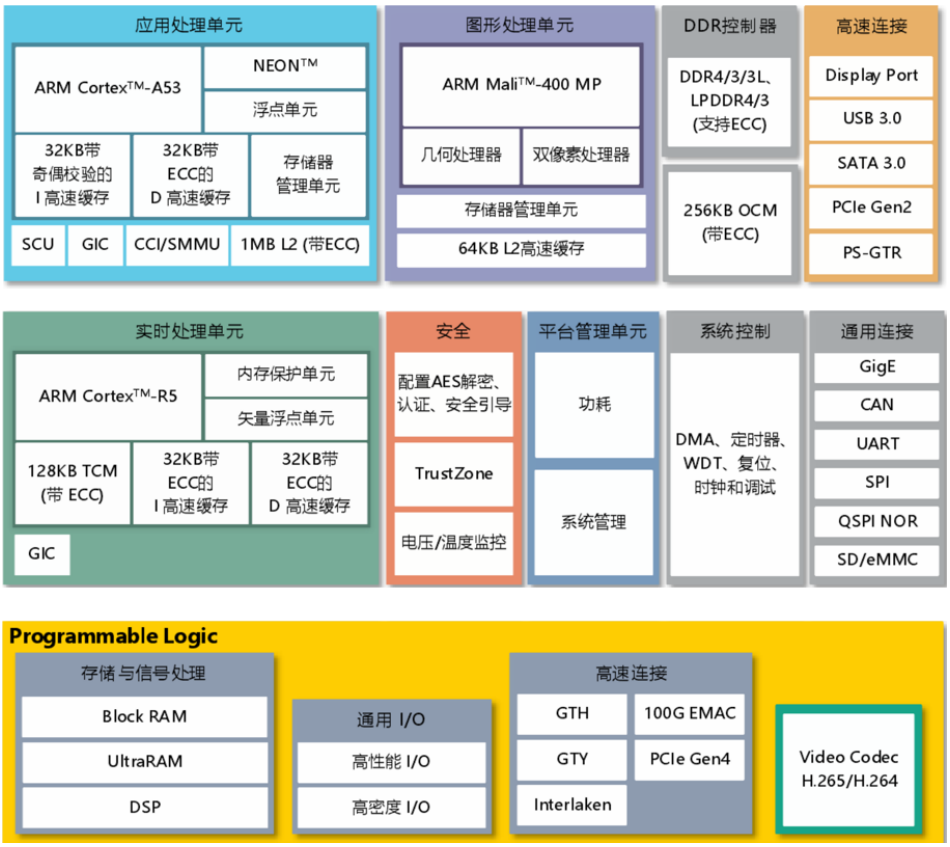

图 4-2 Zynq UltraScale+ MPSoC 内部结构图

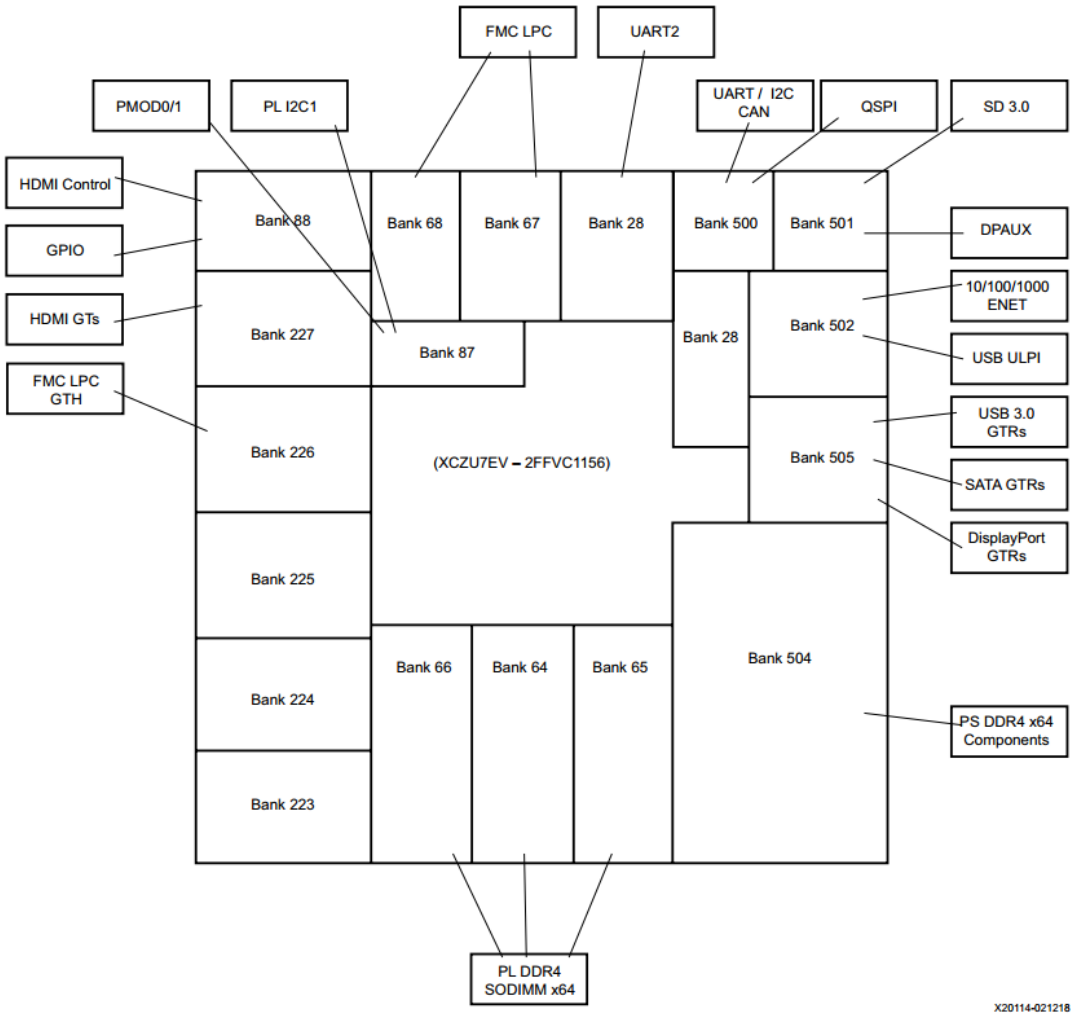

图 4-3 Zynq UltraScale+ MPSoC 内部资源图

图 4-4 Zynq UltraScale+ MPSoC 板卡实物图

4.3 PYNQ 开发框架介绍

传统的 FPGA 架构有两种，一种是 FPGA 与 CPU 互联的，另一种是 FPGA 与 RAM 互联的方式。这两种方式需要设计人员关注硬件的实现细节以及硬件中个各模块的交互。这两种开发方式对于软件人员来说很不友好，软件人员试图将算法在 FPGA 中实现时会遇到障碍。这种障碍使得复杂的系统在 FPGA 上实现起来很困难，算法的加速工作量也非常大。PYNQ 的出现很好的解决了这一问题，基于 PYNQ 框架的开发方式将开发人员从繁重的硬件调试过程中解放出来，让开发人员能够专注于算法层面的优化与创新。PYNQ 可以运行在 ZYNQ 上，ZYNQ 中包含 PL 和 PS 部分，PS 部分是 ARM 的处理器，其上可以运行 Linux 操作系统，操作系统之上运行 python 解释器。PL 部分为可编程逻辑资源，在开发过程开发者先在 PL 端设计 IP 核，将 ip 核配置为基于 AXI 总线的形式，然后在开发工具中生成驱动，最后在 PS 中对驱动函数进行调用即可。

PYNQ 是一个全新的开发框架，无论是 FPGA 工程师还是嵌入式工程师都能够使用 python 进行快速的 FPGA 算法部署，而且在部署的过程中不用去深入研究硬件的实现细节。PYNQ 的框架在 ZYNQ 上内置了 Linux Ubuntu 15.10 操作系统，通过使用此系统可以调用大量的 IP 核，让整个设计的实现和测试更加方便。并且 PYNQ 中将大量的 PL 端 IP 核内置为”Overlays”，可以方便用户调用 PL 端 IP 核。ZCU104 开发套件中的芯片为 Zynq UltraScale+ MPSoC ZU7EV,如表 4-1 所示，芯片中的资源及其丰富，配合 PYNQ 开发框架，非常适合进行复杂系统设计以及算法加速。

表 4-1 Zynq UltraScale+ MPSoC ZU7EV 资源表

Feature	Resource Count
Quad core Arm Cortex-A53 MPCore	1
Dual core Arm Cortex-R5 MPCore	1
Mali-400 MP2 GPU	1
H.264/H.265 VCU	1
HD banks	Two banks, total of 48 pins
HP banks	Six banks, total of 312 pins
MIO banks	Three banks, total of 78 pins
PS-GTR transceivers (6 Gb/s)	Four PS-GTR transceivers
GTH transceivers (16.3 Gb/s)	20 GTH transceivers
System logic cells	504K
CLB flip-flops	461K
Maximum distributed RAM	6.2 Mb
Total block RAM	11 Mb
UltraRAM	27 Mb
DSP slices	1,728

4.4 HLS 加速基本理论

传统的 FPGA 开发流程中，是采用基于 RTL 的设计方式来进行逻辑设计的，这种开发方式有优点也有缺点。其优点在于，可以进行自定义的接口协议的开发，在进行逻辑黏合时候就有很强的优势，设计者可以在开发的过程中从底层的硬件出发，设计相应逻辑接口。也可以根据算法来构建行为级模型，然后在 FPGA 上进行 RTL 设计仿真以及验证。这种开发方式对于软件人员来说十分不友好，而且对于硬件人员来说构建大规模的系统耗时较长，系统内部各个模块之间进行通信也需要大量的设计仿真验证。Xilinx 提供了 Vivado design suite 开发工具，支持 C/C++/Systemverilog 综合为 RTL，通过这种方式可以快速实现算法加速。在高级综合的过程中，设计代码中的数组、结构体和循环体等都会通过编译器编译映射为存储器、计数器和状态机等硬件结构。这种方式明显缩短了开发周期，减少了开发人员成本。

4.4.1 HLS 开发流程

HLS 的开发流程对于用户来说十分友好，基于 C 的开发方式可以节约用户大量的时间，C simulation 相比 RTL 级的仿真来说能够提高验证设计的效率。如图 4-4 所示，HLS 开发流程主要包括 C 开发、C 仿真、C 综合以及 RTL 综合等等。

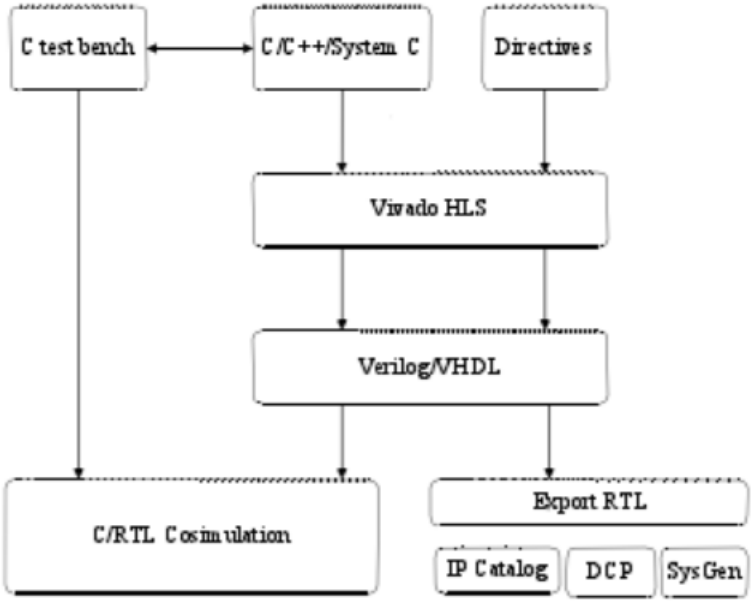

图 4-4 HLS 开发流程图

(1) 建立工程。在新建工程的过程中，需要 C 源码、C Testbench,并且为设计规定相应的时钟周期以及目标器件等等。对于多个设计文件来说，需要指定顶层文件，便于生成多层次的 RTL 文件。

(2) 执行 C Simulation。对于设计来说，首先应该保证其功能的正确性。通过编写 C Testbench 来调用设计，输入待测数据给 DUT，然后将 DUT 处理后的数据与算法模型的正确数据比对，从而检查 DUT 功能是否正确。当 DUT 输出结果与算法模型的输出结果不一致时，可以观察 DUT 的中间结果，然后对中间结果进行分析。通过 Debug 来修正 DUT 的源代码，以改善其功能。

(3) 执行 C synthsize。执行 C synthsize，可以将 C 代码按照预先设定的规则，例如时钟周期等等，转化为 RTL 代码。转化过程中也要考虑接口协议以及转化后的 RTL 的性能，这些都是根据代码中的编译指令或者是工程中的编译设置来决定的。

(4) 执行 Cosimulation。采取 C Simulation 的方式，使用 C 语言编写主函数来调用待测设计，实际上待测设计这个时候就是一个子函数。这个过程采取打印信息比对信息的方式来进行，所以不需要 dump 波形，所以需要的验证时间很短。

(5) 封装 ip 并导出。通过 Cosimulation、C synthsize 之后，可以到处 IP 核，这种导出方式在 IP 核之外添加了驱动程序、以及相应的 ip doc 文件。增强了 IP 的易用性，同时 IP 中也可以设置可以参数修改接口，方便复用 IP。

4.4.2 HLS 开发优势

通过 HLS 工具可以综合出模块级的 RTL 代码，然后封装为 IP，IP 可以在设计中复用。HLS 高层次综合工具将软件以硬件的方式实现，提高了软件执行的效率，另一方面采用 HLS 高层次综合工具也加快了硬件的开发速度。HLS 的功能主要有以下几点：

(1) 使用 C 语言实现设计，然后使用 C Simulation 进行仿真，使用 C 语言调用设计输入激励进行仿真，验证设计的功能的正确性。这种 C Simulation 免去了 dump 波形的时间，大大缩短了开发周期。

(2) 采用 HLS 的开发方式可以将算法的软件代码综合为硬件的逻辑状态机和数据通路。

(3) HLS 开发工具可以将高层次综合之后的设计模块封装为用户自定义接口的 IP 核。

4.5 加速 IP 的 HLS 实现

在网络加速的过程中，需要考虑面积、速度以及功耗的问题。所以在本章里面针对 FPGA 这一特殊平台提出了相应的实现方法，利用 C 语言的高层次抽象能力以及 HLS 编译器的高层次综合功能来加速推断，最后在那个测试系统的性能。

4.5.1 层内并行化实现

对于深度学习算法来说，通常由很多层构成，如果这些层全部展开之后在 FPGA 上运行的话会耗费大量的资源，且不易复用 IP 核。所以针对这种场景，制定了相应的加速方案，即在 FPGA 中实现深度学习算法中的一层，计算时候对这一层进行复用，计算完一层之后将数据缓存到片外的 DDR 中，当进行下一层的计算的时候将数据再次读入运算单元中。如图 4-5 所示，在这个过程中需要异构 FPGA 中的 ARM 来对 IP 核进行配置，配置的内容包括输入输出通道的数量、卷积核的尺寸等等参数。当配置结束之后，IP 就可以进行相应层的运算了。图 4-5 为实现卷积 IP 核的经典架构，这个系统中包含片外 DDR、ARM 处理器、控制器、运算单元以及各种缓冲器，其中缓冲器包括输出输出缓冲、权重缓冲等等。输入的图像数据首先加载到输入的寄存器上，然后通过运算单元执行卷积操作，卷积操作是通过多个运算单元来运算的，这样可以保证运算的速度。在进行卷积运算时，第一级输出缓存中的数据会被输出到第二级输出缓存中。在当前层运算完之后，其运算结果就会称为下一级运算的输入，用这样的方式实现网络每一层的加速。

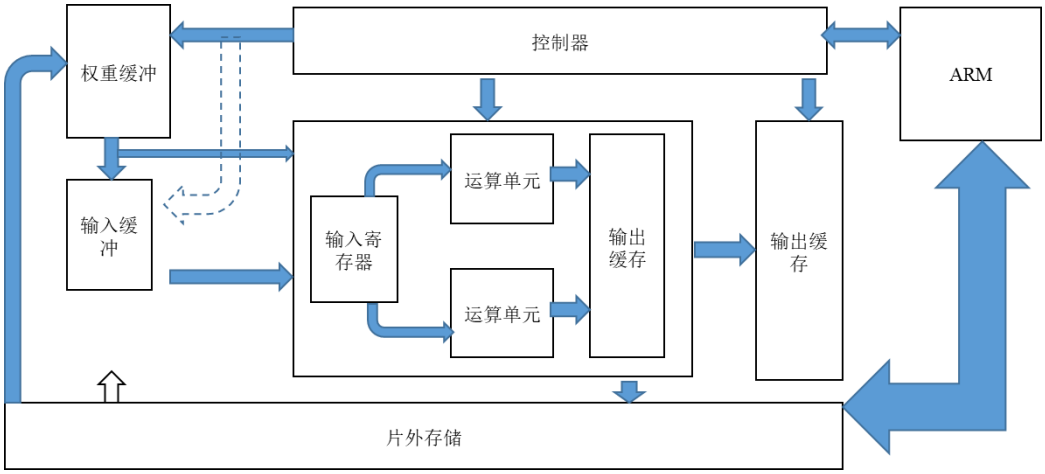

图 4-5 层内并行结构图

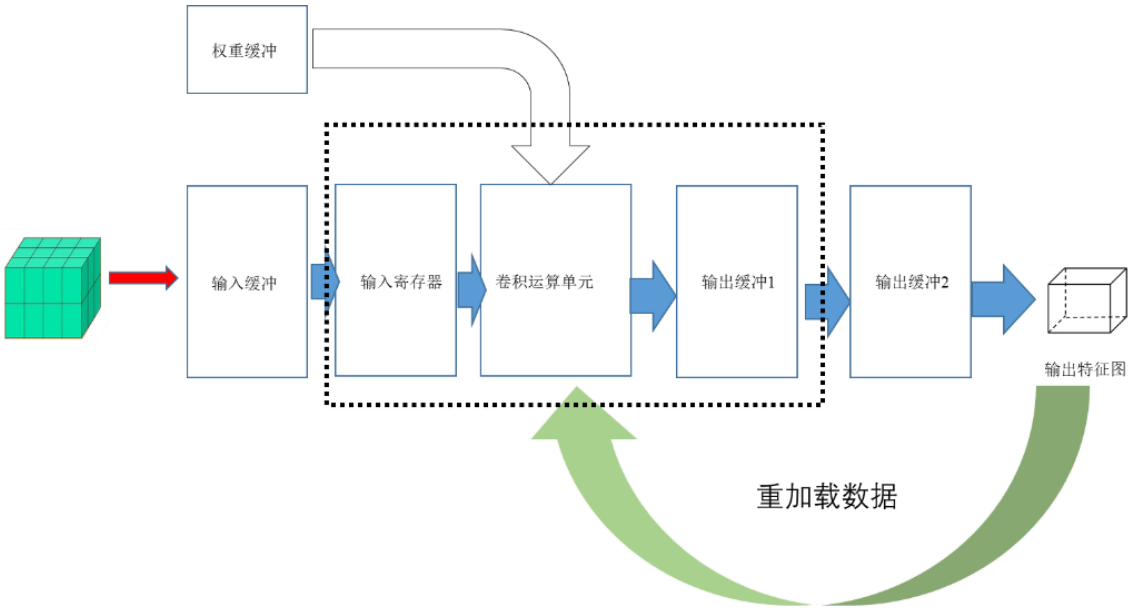

图 4-6 层内并行结构图

```
for(int h = 0; h < H; h += Th) //Loop H
  for(int l = 0;l < L;l += Tl) //Loop L
    for(int m = 0;m < M; n += Tn) //Loop N
      for(int tn = 0;tn < Min(Tn,N-n);tn++)  //Loop Tn
        for(int th = 0;th < Min(Th,H-h);th++)  //Loop Th
          for(int t1 = 0; tl < Min(Tl, L -1);tl++) //Loop T1
            for(int tm = 0; tm < Min(Tm, M - m);tm++)//Loop  Tm
              o[m][h][l] += ∑_{i=0}^{K-1} ∑_{j=0}^{K-1} W[m][n][i][j] * I[n][h * s + i][l * s + j]; //MAC
```

图 4-7 层内并行伪代码

这种方式可以实现网络在单层内部运算的并行优化，不必将整个网络全部展开，这样就降低了整个网络的资源占用率，功耗也随之降低了，其数据流如图 4-6 所示，这种实现方式实现了性能面积以及功耗的平衡。数据分块来进行读写会造成数据访问次数的增加，所以在这个地方要做一些优化，因为将数据从片外搬运到片内时会产生大量的功耗。但是若要追求单层网络的加速效果，那就需要在层的内部实现流水。所以就由一种数据拆分机制，将数据拆分为多个小块，然后并行的去处理这些数据。如图 4-7 所示，将图像分割为多个小块，每个小块的尺寸为$T_r \times T_c \times T_n$，这一小块经过计算之后所得出的结果应该是卷积计算的部分和，其尺寸为$T_h \times T_l \times T_m$，在此运算过程中，所需要的权重尺寸为$\mathrm{K} \times K \times T_n \times T_m$。这一块数据处理之后，再处理这个特征图的下一小块数据。这样按块处理，直到本层的数据处理完即可。

4.5.2 循环优化实现

HLS 张针对于循环的优化指令一共有 22 个，但是本设计中使用的有两种，分别是 Loop Pipeline 和 Unroll。

(1) Loop Pipelining 的作用是对循环进行流水线化的并行处理。这种方式可以让两轮循环执行时间重叠，也就是说本轮循环执行的过程中下一轮循环也可以同时执行。图 4-8 中是未经过流水线优化的循环步骤，图 4-9 中是经过流水线优化的步骤。未经过流水线处理时，需要按照步骤进行串行的处理，即先进行数据的读取，然后进行运算，最后再进行数据的写入。如果每次迭代两次，那么一共需要 6 个时钟周期才能完成运算，在写入数据之前需要五个时钟周期才能够完成运算。但是经过流水线化的处理之后，只需要一个时钟周期就可以完成读取本轮计算的数据，同时准备开始下一次的迭代的数据读取。两轮迭代同时进行，这样完成两轮运算只需要消耗三个时钟周期，仅仅是前面的一半。通过优化指令可以改善吞吐率、减小时延以及缩小初始化间隔。

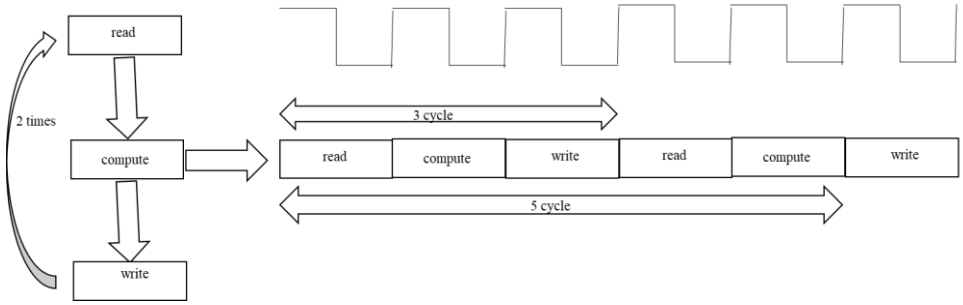

图 4-8 原始算法结构示意图

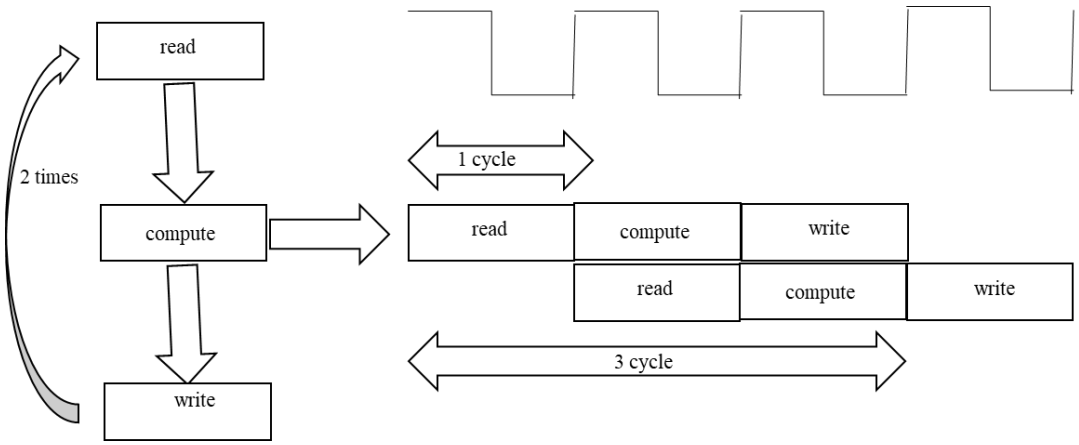

图 4-9 Loop Unrolling 结构示意图

(2) Loop Unrolling，在没有进行循环展开优化之前，循环的运行是按照默认设置来操作的，即按照按照循环的本意来进行运算。当进行循环展开后，循环的电路会被设置为 N 份，N 一般由 HLS 指令中的 factor 来指定。例如当 factor 为 2 时，此时如果迭代的次数为 8，那么迭代会被分为 4 次来进行，每次是 2 个循环一起实现的。如果 factor 为 4 时，循环会被分为 2 次，每次 4 个循环一起执行。这种方式最终实现的电路性能是不一样的，资源也是不一样的。

4.5.3 数组优化实现

在 vivado 中顶层中使用数组时候，vivado 会对数组使用优化。如果内存是在芯片之外，HLS 会把数组综合为 memory 接口。memory 在芯片之内时，数据会根据对应的地址将数据送入运算单元中，其 latency 是一个时钟周期。

用户可以通过使用 Resource 指令来指定 RAM 的接口类型，单端口或者是双端口，RAM 的实现方式(是 BRAM 还是 LUT RAM)。数组一般最终综合为 RAM、ROM 或者是 FIFO。数组的优化方式是数组分割，分割方式有三种，分别是基于 block、cyclic 以及 complete 的三种方式。如图 4-10 所示，factor 代表拆分的因子，block 代表连续拆分，其个数为 N/factor 个块(其中 N 为数组的个数)。cylic 对应的数组拆分方式与前面的 block 方式类似，但是是非连续间隔拆分的。

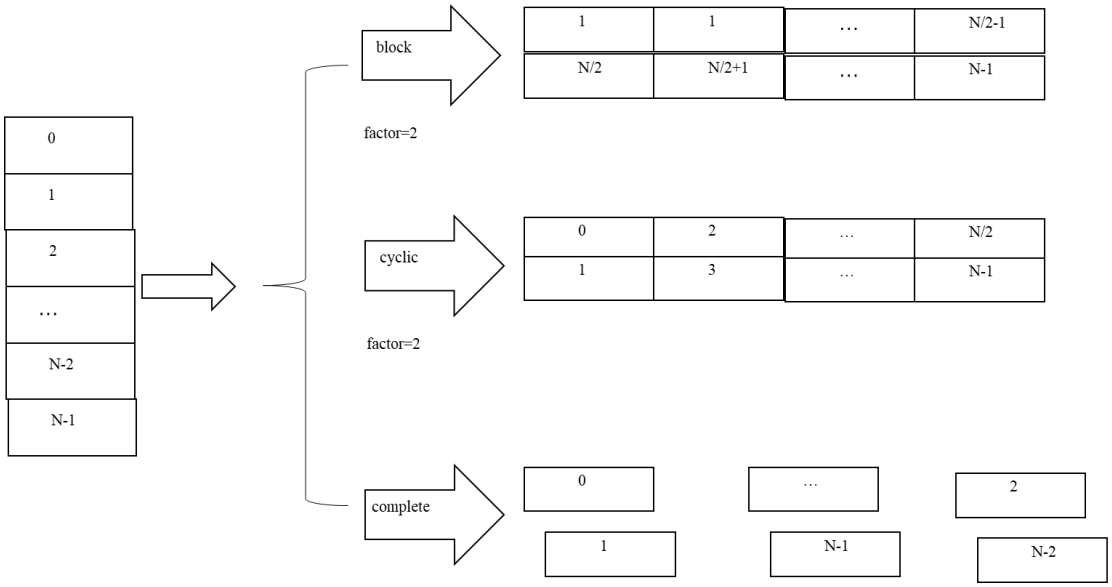

图 4-10 数组优化结构示意图

4.6 硬件系统的构建

4.6.1 PL 部分实现

PL 部分采用 HLS 来实现卷积神经网络，其卷积核循环部分示意代码如下图-11 所示。PL 端的硬件系统配置表格如下所示，图中可以看到，PL 端使用了硬件加速 IP，ZYNQ processor 以及 AXI 的接口，其主要配置如表 4-2 所示。ZYNQ processor 中的也按照工程的需求来进行配置，，其 IO 口配置如图 4-12 所示,时钟配置如图 4-13 所示，最终完成一整个硬件系统。

```
for(i=0;i<k;i++) {
                for(j=0;k<k;j++) {
                                for(trr=row;trr<min(row+Tr,R);trr++) {
                                                for(tcc=col;tcc<min(col+Tc,C);tcc++) {
#pragma HLS pipeline
                                                                for(too=to;too<min(to+Tm,M);too++) {
#pragma HLS UNROLL
                                                                                for(tii=ti;tii<min(ti+Tn,N);tii++) {
#pragma HLS UNROLL
                                                                L: output_fm[too][trr][tcc] +=weights[too][tii][i][j]*input_fm[tii][S*trr+i][S*tcc+j];
}}}}}}
```

图 4-11 卷积核心部分伪代码

表 4-2 实际工程资源配置

资源名称 / 特性	具体参数
ZYNQ Processor	四核
Clock wizard	150Mhz 主时钟
accelerator	150Mhz 主时钟、axi_lite 接口、axi_full 接口
Processsor system reset	Ip reset、interconnect reset
axi interconnect	axi_lite、axi_full

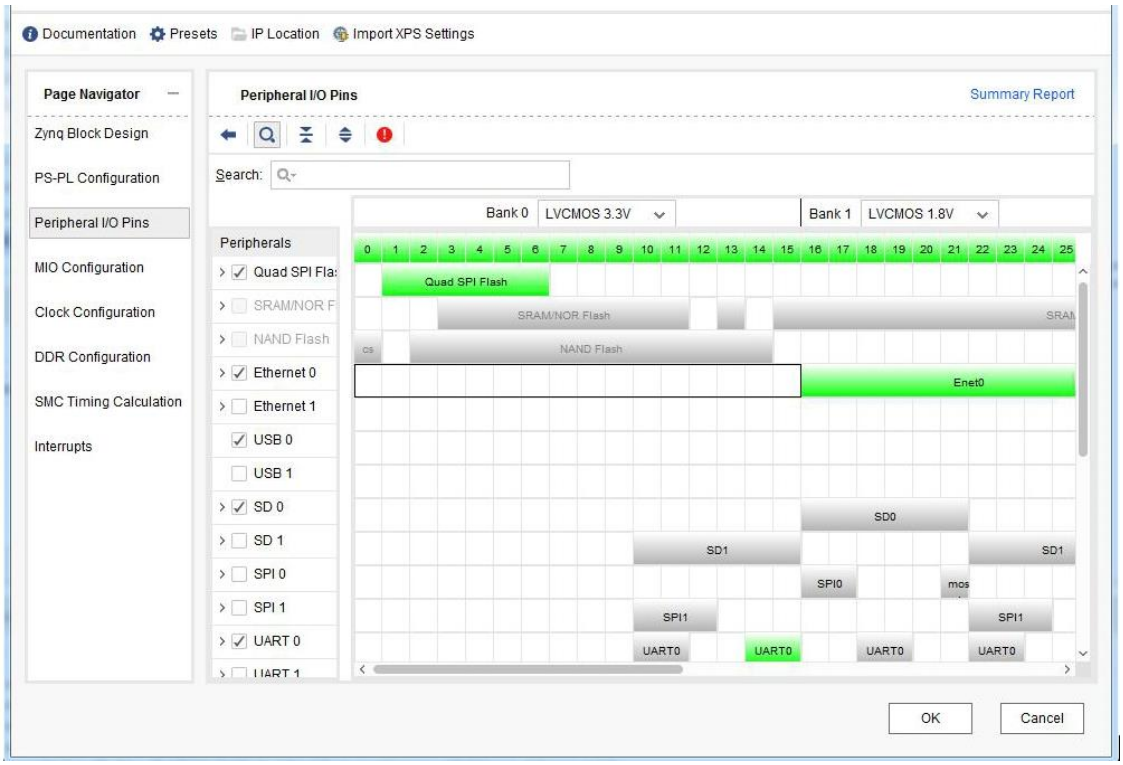

图 4-12 processor io 配置

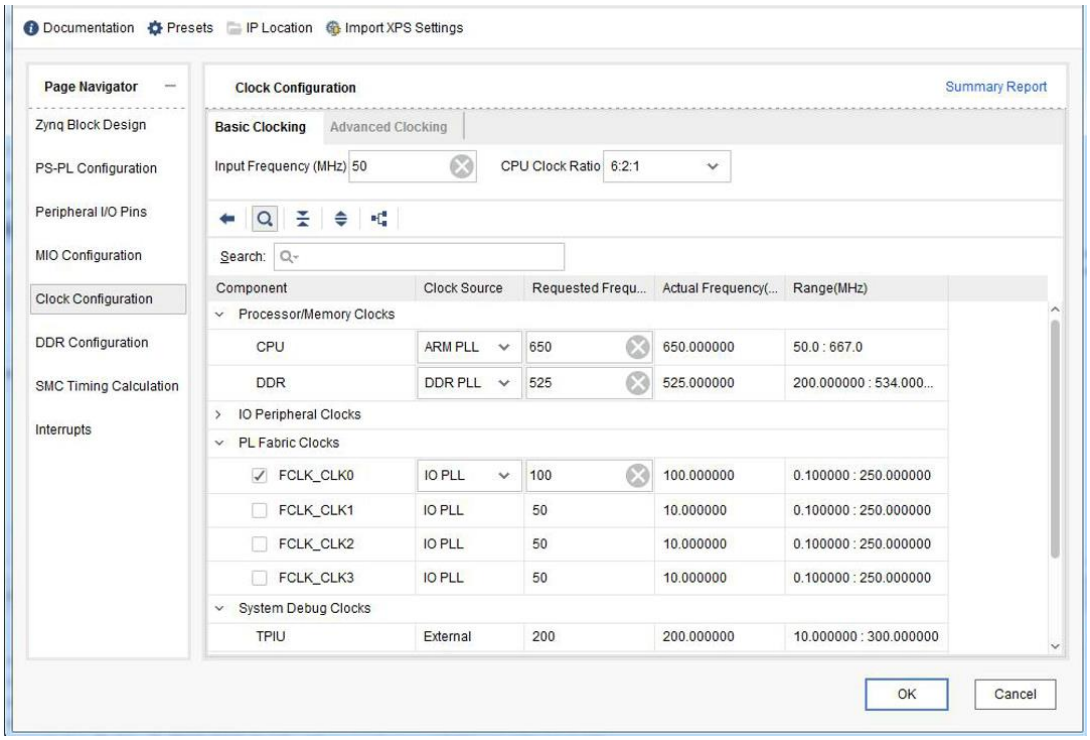

图 4-13 时钟配置

4.6.2 软硬件系统实现

本文中使用的是 ZCU104 开发套件，如下图所示，开发套件上运行的是 PYNQ 框架，在使用前需要对开发套件的软硬件进行配置。其配置的步骤如下：

图 4-14 开发套件实物图

(1) 设置为 SD 卡启动；

(2) 连接供电线；

(3) 拷贝镜像到 SD 卡中，并且将 SD 卡插入到板卡插槽中；

(4) 将 PC 机与板卡通过 USB 连接线连接起来，将板卡上的串口通信线与 PC 机连接起来；

(5) 将 PC 机与板卡进行逻辑上的互联。有两种方式可以达到这种目的，第一种方式通过以太网连接，即将板卡上的以太网接口与 PC 机上的网络接口连接起来，PC 机上分配 IP 地址，然后通过访问网址 http://192.168.2.99:9090 来访问 Jupter Notebook 来将 PC 作为上位机来调试板卡。通过这种方式来连接板卡与 PC 机会导致 PC 机无法访问网络，PYNQ 框架在开发的过程中需要调用一些开源的库，例如图像处理库等等，所以无法上网会给开发增加困难。第二种方式是将开发板与 PC 连在同一个局域网之下，即连接到同一个路由器下，然后登陆到网址 http://pynq:9090 进入到 Jupter Notebook 中进行在线调试。本论文中采用连接到同一个路由器的方式来连接 PC 和开发套件。

(6) 在上述开发完成之后，将 PL 端的 bitfile 加载到 FPGA 中，再 PC 上进行在线调试，最后在 PC 上观察处理后的图像，进行结果分析与统计。

4.7 结果分析

最终实现结果如图 4-16 所示，整个系统能够处理图像，并且通过 PYNQ 中的 JupterNotebook 显示出图像以及框出被测目标。图 4-15 和图 4-16 分别为加速前和加速后的效果图，右侧的参数表示为预测概率，该参数表征物体的属于该类别的可能性为多大。加速前 person 的预测概率为 72%，加速后 person 预测概率为 70%，其相差为 2%，误差较小。对于 horse 和 dog 来说，加速前后的预测概率误差也在可接受范围之内，加速之后的效果与加速之前精度损失很小。如表 4-3 所示，共选用 2 万张图片，进行实验，加速之后预测概率误差很小，在可接受范围之内，在精度上基本达标。从图 4-17 的资源消耗图可以看出，芯片中的 DSP 资源消耗较大，占了总体的 78%，原因在于加速 IP 中需要大量的累乘加运算，这些运算需要消耗大量的 DSP 单元。加速 IP 中需要使用大量缓冲单元进行数据的缓冲，防止运算单元数据不足的情况发生，所以 BRAM 的消耗量是 312，占据了总体 BRAM 总体资源的 73%，另外对于大面积的缓存采用 Ultra RAM 来实现，占了总体 Ultra RAM 的 46%。另外在加速性能方面，在不进行展开加速的情况下，运行一张图片需要 19s。再进行 HLS 展开加速之后，运行一张图片只需要 1.92s，基础算力提高了近十倍。表 4-3 中，可以看到加速前后性能提升了十倍。未进行加速的时候整个系统

类似于一个嵌入式处理器，其运算是基于指令集的运算。

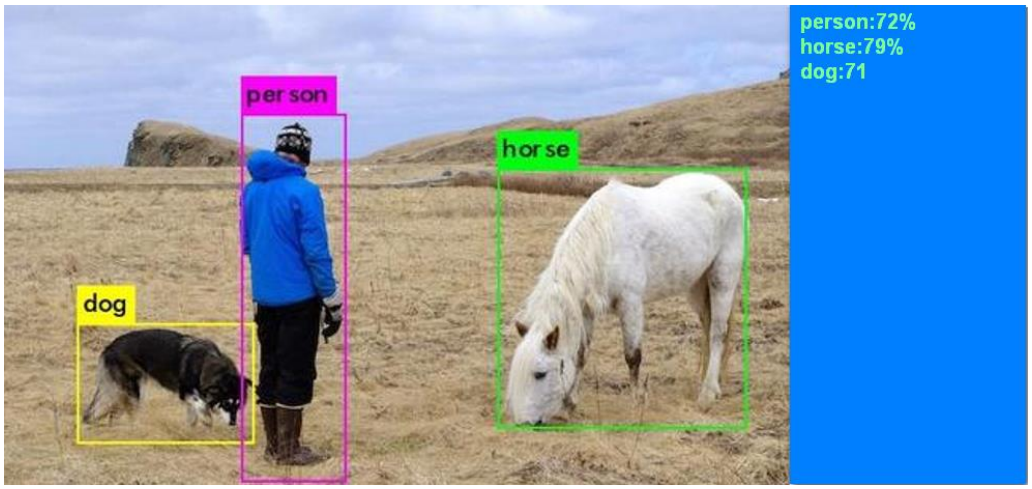

图 4-15 原始检测结果

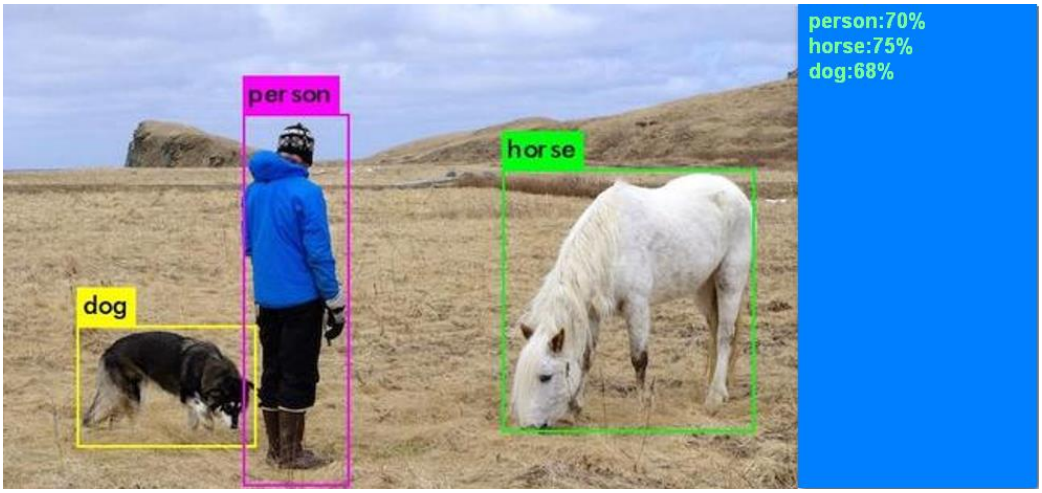

图 4-16 加速后的检测结果

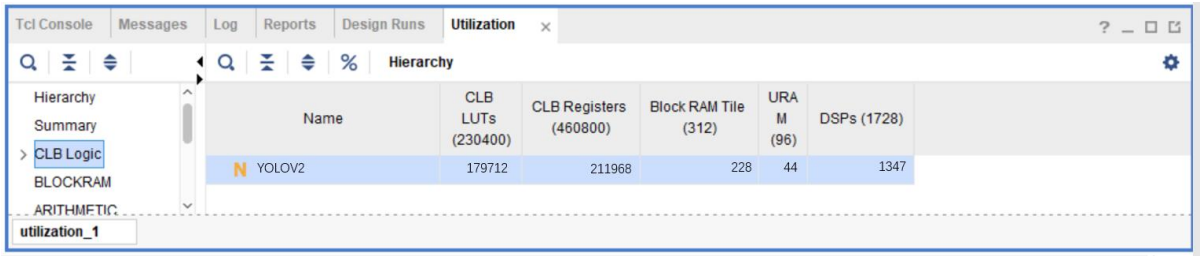

图 4-17 整体资源消耗

表 4-3 加速前后精度与性能对比

	dog	horse	person
加速前后预测概率平均误差	2%	1%	2%
加速前平均运行时间	1.92	1.90	1.87
加速后平均运行时间	0.18	0.19	0.18
加速倍数	10.67	10	10.39

4.8 本章小结

本章详细阐述的算法的硬件实现平台 FPGA，其中重点介绍了 Xilinx ZCU104 开发套件的硬件结构和开发框架。根据网络层的结构，结合硬件平台的特性和 HLS 开发方式提出了加速 IP 的实现方案，使用 HLS 实现了算法加速单元，完成了层内并行化、循环优化以及数组优化。本章还介绍了硬件系统的构建，即如何将 PL 部分的 HLS 封装为标准 IP 核，在 vivado design suite 中进行集成，最后在 PYNQ 框架下进行整个算法系统的运行和测试。

第五章 总结与展望

5.1 全文总结

深度学习算法有着广泛的需求，但是在深度学习算法落地的过程中有很多的困难，例如采用 GPU 来进行深度学习的推理时无法在嵌入式设备中使用，当采用 ARM来做深度学习推理时性能无法满足。基于 FPGA 的深度学习加速方案可以很好的解决这一问题，使用软硬件协同的方式，将流式运算放在异构 FPGA 的 PL 端进行，将数据的显示以及统计放在 PS 端进行。本文采用 Xilinx 的 ZCU104 板卡来进行开发，ZCU104 套件中 FPGA 芯片中有着丰富的查找表资源以及 dsp 运算单元，此运算单元负责进行 MAC 运算。另外芯片中还有丰富的 BRAM 资源，

其中的 Ultra RAM 资源可以进行大批量数据的缓存，保证数据的传输速度。采用加速 IP 核复用的方式可以提高系统的可移植性，而且达到了面积和速度上的均衡。加速 IP 采用 HLS 的开发方式，能够缩短开发时间，简化开发流程，快速将算法映射到 RTL 级电路中，在 FPGA 中实现算法的加速。

本论文首先分析了 YOLOV2 的算法基本理论，依据算法的基本结构将算法进行软硬件划分，流式运算在 FPGA 中进行，预处理以及后处理运算在 ARM 中进行。将硬件加速部分使用 HLS 来实现，其余部分在 ARM 中采用 PYNQ 框架中的 python 来实现。

本文所作的工作总结如下：

(1) 分析了 YOLOV2 的工作原理，根据国内外的论文确定了异构 FPGA 对于 YOLOV2 算法加速任务的优势。分析了 YOLOV2 的网络结构和异构 FPGA 的基本特征，得到了一种 YOLOV2 的 FPGA 加速软硬件划分方案。

(2) 对算法的软硬件方案与 Xilinx 的 ZCU104 开发套件进行适配，确定了采用 PYNQ 的软硬件协同方式来进行开发。特别研究了算法的结构，包括运算和存储，确定了硬件系统中的各个参数，软硬件交互方式以及交互接口，最终在 vivado 中搭建出整个系统。

(3) 研究了卷积算法和池化算法的结构，特别研究了数据的传输缓存以及卷积层的运算方式，得到了一种基于 IP 复用的加速方案。根据数据的传输方式确定了 IP 的控制接口和数据传输协议，也制定了 IP 内部多缓存的硬件结构。在对卷积的运算方式分析后，将层内部的算法进行改写，采用层内并行以及分块运算的方式进行卷积加速。

(4) 研究了 PYNQ 框架的开发原理与过程，利用 PYNQ 中特有的面向对象编

程语言 python 来进行数据的前处理后处理，采用 python 中的图像处理工具箱来进行图像的展示以及结果分析。

5.2 工作展望

本文中使用异构 FPGA 来进行深度学习加速，性能上和 GPU 版本的基本持平，在加速方面也有了很明显的效果。本文中所研究的加速是针对嵌入式 ARM 来说的，在实际工程中，有时候会使用 GPU，对比 FPGA 与 GPU 的速度和功耗比也是我将要研究的内容。对于本文的深度学习加速实验的工作，还有一些工作需要完善和优化：

1. 本文中使用的模型虽然成功实现了加速，但是若要让加速效果达到实时目标检测的标准，需要对模型进行二值化，这需要在算法层面进行大量的优化，不断进行训练迭代。通过二值化之后，算法模型中的参数会得到大幅度的减少，数据传输所需要的带宽也会相应减少，运算复杂度大大减小。因此，需要对算法进行二值化。

2. 本文中并未对原始算法的结构进行改变，原始的深度学习算法结构上存在一定的冗余，如果能够对原始的算法进行剪枝和压缩，那么会大大增加算法的运行速度。所以，对算法进行剪枝和压缩也是影响算法性能的一个重要因素。

通过研究，本文在实现深度学习算法 YOLOV2 之后，能够对算法进行优化，加快算法的运行速度，使其能够在移动设备上使用。随着低功耗设备需求的不断增大，本文中所使用的硬件加速方案将有望发挥重要作用。深度学习算法也将会在越来越多的移动设备上出现，低功耗深度学习加速方案终将促进整个目标检测领域的发展。

致谢

光阴荏苒，岁月如歌，转眼之间就到了挥手告别三年研究生生涯的日子了。值此毕业之际，我向给予我指导、帮助并关心我的老师、朋友和亲人致以最诚挚的谢意。

感谢我的导师荣健教授，三年中在学习和生活中荣老师都非常关心我们，教会了我们如何做人做事，让我受益匪浅。感谢教研室的师兄师姐、师弟师妹以及同级的伙伴们，感谢他们在生活和学习上给了我很大的帮助，让我能够在研究生阶段顺利完成学业，和大家一起成长的三年将会是我人生中非常美好的一段回忆。感谢我的同门董文辉，在研究生期间给了我很大的帮助，在生活上给予了我帮助，同时他自律自强、努力拼搏，是个很优秀的人。感谢我的朋友们，在我研究生时期给了我很大的帮助，让我能够乐观面对遇到的困难。

还有很多需要感谢的人，这里无法一一写出，在这里对他们表示最诚挚的谢意，在未来生活中我会不断努力，实现自我提升。

参考文献

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

[2] K. He, X. Zhang, S. Ren, et al. Deep residual learning for image recognition[C].Proceedings of the IEEE conference on computer vision and pattern recognition, New York, 2015, 770-778

[3] S. Asano, T. Maruyama, Y. Yamaguchi. Performance comparison of FPGA, GPU and CPU in image processing[C].2009 international conference on field programmable logic and applications, Prague, 2009, 126-131

[4] Q. Wu, Y. Ha, A. Kumar, et al. A heterogeneous platform with GPU and FPGA for power efficient high performance computing[C].2014 International Symposium on Integrated Circuits(ISIC), Singapore, 2014, 220-223

[5] S. K. Rethinagiri, O. Palomar, J. A. Moreno, et al. An energy efficient hybrid FPGA-GPU based embedded platform to accelerate face recognition application[C]. 2015 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XVIII), YOkohama, 2015, 1-3.

[6] Z. Wang, F. Qiao, Z. Liu, et al. Optimizing convolutional neural network on FPGA under heterogeneous computing framework with OpenCL[C].2016 IEEE Region 10 Conference (TENCON), Singapore, 2016, 3433-3438.

[7] R. Morcel, M. Ezzeddine, H. Akkary. Fpga-based accelerator for deep convolutional neural networks for the spark environment[C].2016 IEEE International Conference on Smart Cloud (SmartCloud), Trento, 2016, 126-133.

[8] 董振兴.基于 FPGA 平台的深度学习应用研究[D].西安：西安电子科技大学，2018，1-25

[9] 黄伟杰.基于 Zynq 的深度学习图像分类识别系统的设计[D].广州：广东工业大学，2018，30-50.

[10] 陆志坚.基于 FPGA 的卷积神经网络并行结构研究[D].哈尔滨：哈尔滨工程大学，2013，15-33

[11] 王江峰.基于 FPGA 的 CNN 自动代码生成设计与实现[D].天津：天津工业大学,，2018，35-60

[12] 王小雪.基于 FPGA 的卷积神经网络手写数字识别系统的实现[D].北京：北京理工大学，2016，28-50

[13] K. Simonyan, A. Zisserman. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:14091556, 2014, 18(7):256-270.

[14] B. Jacob, S. Kligys, B. Chen, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference[C]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake, 2018, 2704-2713.

[15] S. I. Venieris, C.-S. Bouganis. Latency-driven design for FPGA-based convolutional neural networks[C].27th International Conference on Field Programmable Logic and Applications(FPL), Ghent, 2017, 1-8.

[16] N. Mellempudi, A. Kundu, D. Mudigere, et al. Ternary neural networks with fine-grained quantization[J]. arXiv preprint arXiv:170501462, 2017, 36-43

[17] I. Hubara, M. Courbariaux, D. Soudry, et al. Binarized neural networks: Training neural networks with weights and activations constrained to +1 or -1[J]. arXiv preprint arXiv:160202830, 2016, 23-28

[22] 仇越. 基于 FPGA 的卷积神经网络加速方法研究及实现[D].苏州：江南大学，2018，20-30

[23] S. I. Venieris, A. Kouris, C. S. Bouganis. Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions[J]. Acm Computing Surveys, 2018, 51(3): 1-39

[24] K. Guo, S. Zeng, J. Yu, et al. A Survey of FPGA-Based Neural Network Accelerator[J]. ACM trans, 2018, 27-35

[25] S. Mittal. A survey of FPGA-based accelerators for convolutional neural networks[J]. Neural Computing and Applications, 2018, 1-31

[26] S. I. Venieris, C. S. Bouganis. fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs[C]. IEEE International Symposium on Field-programmable Custom Computing Machines, New York, 2011, 10-15

[27] K. Abdelouahab, M. Pelcat, J. Serot, et al. Tactics to Directly Map CNN graphs on Embedded FPGAs[J]. IEEE Embedded Systems Letters, 2017, 9(4): 113-116

[28] Y. Umuroglu, N. J. Fraser, G. Gambardella, et al. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference[C].Acm/sigda International Symposium on Field-programmable Gate Arrays,New york, 2017,120-131

[29] H. Nakahara, H. Yonekawa, T. Fujii, et al. A lightweight yolov2: A binarized cnn with a parallel support vector regression for an fpga[C].Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, toronto, 2018, 31-40

[30] Z. Chen, Z. Fang, P. Zhou, et al. Caffeine: Towards Uniformed Representation and Acceleration for Deep Convolutional Neural Networks[C].IEEE/ACM International Conference on Computer-aided Design, Austin, 2105, 641-648

[31] 单祥茹.加速云：用 FPGA 提高 AI 计算力，用 IP 库降低开发难度[J]. 中国电子商情(基础电子)，2018，(05)：21-22

[32] 张丽丽.基于 HLS 的 Tiny-yolo 卷积神经网络加速研究[D].重庆：重庆大学，2017，1-30

[33] 钟楠.基于 FPGA 的卷积神经网络关键技术研究与实现[D]. 北京：北京邮电大学，2018，35-40

[34] 余凯，贾磊，陈雨强，徐伟.深度学习的昨天、今天和明天[J].计算机研究与发展,，2013，50(09)：1799-1804.

[35] 李理，应三丛.基于 FPGA 的卷积神经网络 Softmax 层实现[J].现代计算机（专业版），2017(26)：21-24.

[36] Martens J. Deep learning via Hessian-free optimization[C]// International Conference on International Conference on Machine Learning. Omnipress, 2010:735-742.

[37] Chen Y, Sun N, Temam O, et al. DaDianNao: A Machine-Learning Supercomputer[C]//Ieee/acm International Symposium on Microarchitecture. IEEE, 2015:609-622.

[38] Farabet C, Martini B, Corda B, et al. NeuFlow: A Runtime-Reconfigurable Dataflow Processor for Vision[J]. 2011, 9(6): 109-116.

[39] Sankaradas M, Jakkula V, Cadambi S, et al. A Massively Parallel Coprocessor for Convolutional Neural Networks[C]// IEEE International Conference on Application-Specific Systems, Architectures and Processors. IEEE, 2009: 53-60.

[40] 高放，黄樟钦.基于异构多核并行加速的嵌入式神经网络人脸识别方法[J]. 计算机科学，2018，45(3)：288-293.

[41] Hubel D H, Wiesel T N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex.[J]. Journal of Physiology, 1962, 160(1):106-154.

[42] Fukushima K, Miyake S, Ito T. Neocognitron: a neural network model for a mechanism of visual pattern recognition[M]// Neurocomputing: foundations of research. MIT Press,1988:826-834.

[43] Hughes G. On the mean accuracy of statistical pattern recognizers[J]. Information Theory IEEE Transactions on, 1968, 14(1):55-63.

[44] Ross Girshick, Jeff Donahue, Trevor Darrell, et al. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation[J]. 2013:580-587.

[45] Redmon J, Farhadi A. YOLO9000: Better, Faster, Stronger[C].//CVPR 2017: Proceeding of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).. Piscataway, NJ:IEEE, 2017：6517-6525

[46] Liu W, Anguelov D, Erhan D, et al. SSD: Single Shot MultiBox Detector[C]// European Conference on Computer Vision. Springer, Cham, 2016:21-37.

[47] Schamm T, Von Carlowitz C, Zollner J M. On-road vehicle detection during dusk and at night[C]// Intelligent Vehicles Symposium. IEEE, 2010:418-423.

[48] Chang W C, Cho C W. Online Boosting for Vehicle Detection[J]. IEEE Transactions on Systems Man & Cybernetics Part B, 2010, 40(3):892-902.

[49] Deng Y, Liang H, Wang Z, et al. Novel approach for vehicle detection in dynamic environment based on monocular vision[C]// IEEE International Conference on Mechatronics and Automation. IEEE, 2014:1176-1181.

[50] Rezaei M, Terauchi M. Vehicle Detection Based on Multi-feature Clues and Dempster-Shafer Fusion Theory[C]// Pacific-Rim Symposium on Image and Video Technology, Psivt.2013:60-72.

[51] Niknejad H T, Takeuchi A, Mita S, et al. On-Road Multivehicle Tracking Using Deformable Object Model and Particle Filter With Improved Likelihood Estimation[J]. IEEE Transactions on Intelligent Transportation Systems, 2012, 13(2):748-758.

[52] Lin B F, Chan Y M, Fu L C, et al. Integrating Appearance and Edge Features for Sedan Vehicle Detection in the Blind-Spot Area[J]. IEEE Transactions on Intelligent Transportation Systems, 2012, 13(2):737-747.

[53] Sivaraman S, Trivedi M M. Vehicle Detection by Independent Parts for Urban Driver Assistance[J]. IEEE Transactions on Intelligent Transportation Systems, 2013,14(4):1597-1608.

[54] M. Motamedi, P. Gysel, V. Akella, et al. Design space exploration of FPGA-based Deep Convolutional Neural Networks[C].Design Automation Conference,California, 2018,576-580.

攻读硕士学位期间取得的成果

[1] 黄磊.基于 FPGA 的多路信号输出装置[P].中国，实用新型专利，201921353477.2，2019 年 8 月 2 日

[2] Jing S, Huang L, Rong J. A New Design and Feedback Control of a Fiber Michelson Interferometer to Detect High Frequency Vibration with Style of Noncontact [C]//2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC). IEEE, 2019: 1583-1586

[3] 井帅奇，黄磊，董文辉，荣健.基于 3×3 耦合器干涉仪的正负反馈判断及高频振动探测研究[J].光学学报，2019，39(07)：117-122.