中国科学技术大学

学士学位论文

神经网络处理器编译器的测试框架

姓　　名：吴凡迪

院　　系：少年班学院

学　　号：PB13000675

导　　师：郭崎 副教授

完成时间：二〇一七年五月

University of Science and Technology of China

A dissertation for bachelor's degree

USTC Thesis Template for Bachelor, Master and Doctor User's Guide(Beta)

Author :	Fandi Wu
Department :	
Student ID :	PB13000675
Supervisor :	A/Prof. Qi Guo
Finished Time :	May 2017

致　谢

感谢原本科模板的作者 XPS、硕博模板的作者刘青松以及它们的维护者的辛勤工作！

感谢大家对本模板更新工作的支持！

本模板以及本示例文档还存在许多不足之处，欢迎大家测试并及时提供反馈。

ywg@USTC

在中国科技大学完成本科和硕博连读学业的九年里，我所从事的学习和研究工作，都是在导师以及系里其他老师和同学的指导和帮助下进行的。在完成论文之际，请容许我对他们表达诚挚的谢意。

首先感谢导师 XXX 教授和 XXX 副教授多年的指导和教诲，是他们把我带到了计算机视觉的研究领域。X 老师严谨的研究态度及忘我的工作精神，X 老师认真细致的治学态度及宽广的胸怀，都将使我受益终身。

感谢班主任 XXX 老师和 XX 老师多年的关怀。感谢 XXX、XX、XX 等老师，他们本科及研究生阶段的指导给我研究生阶段的研究工作打下了基础。

感谢 XX、XXX、XXX、XX、XXX、XXX、XXX、XX 等师兄师姐们的指点和照顾；感谢 XXX、XX、XXX 等几位同班同学，与你们的讨论使我受益良多；感谢 XXX、XX、XXX、XX、XXX 等师弟师妹，我们在 XXX 实验室共同学习共同生活，一起走过了这段愉快而难忘的岁月。

感谢科大，感谢一路走过来的兄弟姐妹们，在最宝贵年华里，是你们伴随着我的成长。

最后，感谢我家人一贯的鼓励和支持，你们是我追求学业的坚强后盾。

赵钱孙

2017 年 5 月 22 日

目 录

摘 要

随着深度学习算法的复杂化和应用规模的不断增长，传统的计算机 CPU、GPU 芯片在进行神经网络运算时，受限于功耗与性能，不能满足计算的需求。我们的寒武纪深度学习处理器（以下简称神经网络处理器），通过定制专用的运算部件、设计高效的片上存储，有效地提高了处理器的运算、访存效率，从而获得相对通用处理器数量级提升的性能优势。在学术界和工业界获得了广泛关注。

为了方便用户编程，减少开发难度，一个可靠、安全、稳定的神经网络处理器编译器是不可或缺的。而为了达到这个目的，就必须要由完善的测验流程与高效的测验框架来对其进行验证。然而，受深度学习指令集的特性所限，传统的编译器测验框架无法实现对神经网络处理器的测验，而仅依靠随机指令产生器进行测验效率过于低下。本文针对神经网络处理器指令以及编译器的特性进行了分析，提出并实现了一套完整的测验流程，为此我们设计了，随机网络生成器，寒武纪测验框架。

通过我们的神经网络处理器测验框架，我们大大加速了对神经网络处理器编译器的测验进程，保证了寒武纪系列神经网络处理器编译器的正确性及可靠性。

关键词： 深度学习 神经网络处理器 Caffe 编程框架 测验框架

ABSTRACT

With the complexity of the deep learning algorithm and the growing application scale,limited by power consumption and performance,the traditional CPU, GPU chip can not meet the needs of computing in the neural network operation. Our Cambricon deep learning processor (neural network processor) effectively improves the processor's operation by customizing the dedicated computing components, the design of efficient on-chip storage.It gains superior performance advantages over the number of general purpose processors,so it has received extensive attention in academia and industry.

In order to facilitate user programming, reduce the difficulty of development, a reliable, safe and stable neural network processor compiler is indispensable.In order to achieve this goal, we needs a perfect test process and efficient test framework to verify it. However, the traditional compiler test framework can not achieve the test of the neural network processor.Only rely on the random command generator to test the efficiency is too low. In this paper, we analyze the characteristics of neural network processor and the characteristics of the compiler, and propose and achieve a complete set of test process. We alse designed the random network generator and the Cambrian test framework.

We have greatly accelerated the neural network processor compiler test process to ensure that the Cambrian series of neural network processor compiler correctness and reliability by using our neural network processor test framework.

Keywords: Deep learning,Neural network processor,Caffe,Test framework

第一章　绪论

1.1　研究背景

随着大数据时代的到来，计算机朝着智能化的趋势不断发展。深度学习等新兴方法已经卓有成效的应用在各个领域。特别是深度神经网络的出现，让很多领域都取得了突破性的进展。在每年最大的数据集 ImageNet 图像识别大赛中 [1]，微软研究团队所设计的图像识别系统通过运用深度神经网络技术使得识别图像的正确率达到了 97%，定位正确率达到了 91%；而在国际多通道语音识别大赛上，科大讯飞团队采用深度神经网络，使得识词错误率达到了 2.24%（六麦克风），相比于原来的传统机器学习算法，其语音识别错误率下降了接近六成。除此之外，在 2016 年 3 月举世瞩目的人机大战中 [2]，AlphaGo 战胜围棋高手李世石，深度学习逐渐走上神坛，成为计算机科学中最为炙手可热的研究方向之一。

由于深度学习的应用都包含大量参数且需要大量计算，需要相应的处理器作为支撑。因此，针对深度学习的硬件、芯片等行业飞速发展。

通常，研究者们将深度学习算法部署在 CPU、GPU 或者 FPGA 上。传统的中央处理器 CPU 虽然通用性较好，但是它单次只能处理少量任务，并行性较低，不适合与神经网络的高并行性计算，对于深度学习来讲，只能起到辅助作用；GPU 拥有高并行的计算的特点，且在深度学习所需求的浮点数计算上性能极其出众，在神经网络的训练阶段大放异彩。但是在庞大的需求规模下，数据中心的 GPU 集群的能耗相当恐怖，对移动端的神经网络前向运算实用性差。FPGA 由于其可编程性的特征可以灵活适应飞速发展的神经网络算法的变迁，且性能功耗比出色，但是想达到领先 GPU、CPU 计算性能的高端 FPGA 芯片成本太高，不适合数据中心的大规模集群布置。

在这种情况下，专用于深度学习，神经网络的神经网络处理器成为了目前热门的研究方向。神经网络处理器，是指具有模仿人的大脑判断能力和适应能力、可并行处理多种数据功能的处理器。相较于 CPU、GPU，神经网络处理器结合了神经网络模型的数据局部性特点以及计算特性，进行存储体系以及专用硬件设计，从而具有更好的性能加速比以及计算功耗比。目前，国际上流行的神经网络处理器有：IBM 与美国国防部共同研发的类脑芯片 TrueNorth[3]，谷歌公司推出的与编程框架 Tensorflow[4] 高度契合的神经网络处理器 TPU[5]，以及中科院计算所所推出的寒武纪系列神经网络处理器 (Cambricon-IPU)[6]。

从 2008 年起，中科院计算技术研究所智能处理器研究中心开展了寒武纪系列神经网络处理器的研发，这也是国际上首个深度学习处理器结构。当前寒武纪系列已包含 3 种原型处理器结构：DianNao[7]，单核神经网络处理器结构；DaDianNao[8]，面向超大规模神经网络的多核处理器结构；PuDianNao[9]，面向多种机器学习算法。在若干代表性深度神经网络上的实验结果表明，DianNao 的

平均性能超过主流 CPU 核的 100 倍，但是面积仅为 1/30 而功耗则仅为 1/5, 效能提升可达 3 个数量级；DianNao 的平均性能与主流 GPU 相当，但面积和功耗仅为主流 GPU 百分之一量级。2016 年，中科院计算所的研究团队又提出了了深度学习指令集 Cambricon[10]，试图在更为泛化的层面来完成 AI 加速器的设计。

为了方便用户编程，减少开发难度，我们设计了一套包括库和编译器的软件，为了保证这套软件的正确性和可靠性，我们需要对其进行测验。然而，徒手检查代码工作量大，而仅靠随机生成的指令又无法覆盖到实际使用需求的指令序列，而且对于专用的神经网络处理器来说，由于在运算部件和数据存储结构等多方面创新性的不同，导致了神经网络处理器编译器的验证方式与传统编译器很有大差别。因此，设计一个能用于检测神经网络处理器编译器的测试框架势在必行。

1.2 论文的组织结构

第一章绪论部分介绍了由于深度学习近年来的流行，更快更低功耗的硬件设备被人们所需求，这导致了神经网络处理器的出现与发展。而为了更有效可靠的使用神经网络处理器来将深度学习应用在学术研究和产品中，需要在开发过程中，对神经网络处理器编译器的正确性进行验证。

第二章相关工作部分首先介绍了传统软件的验证方法，接着介绍了传统编译器的验证手段，通过分析深度学习指令集的特性，阐明了神经网络处理器编译器与传统编译器的差别，证明了神经网络处理器编译器验证框架的重要性。

第三章首先先从神经网络处理器的整体构架说起，而后详细的介绍了神经网络处理器的软件构架并着重点出了我们验证的主要目标。接着，从 prototxt 的生成到 caffemodel 的建立，再到 caffe 的的重载直到最后指令的调用，层层深入，详细介绍了神经网络处理器编译器的验证流程，并简要介绍了使用该测验框架后，测验工作的进程。

第四章作为本文的重点，介绍了神经网络处理器编译器验证框架中最重要的随机网络生成器的实现。吸取了编程框架 caffe 中数据结构的优点并对其结构进行改良，将层的连接方式以及网络的构建融入网络生成器的创建当中，并实现了一系列方法用于满足用户对结构以及参数的要求，最后，建立了一个数据库用于保证测试的高效性，同时阐述了数据库的设计思想。

第五章总结了论文的研究工作，并针对现有的神经网络处理器编译器测试框架提出了几点改进，并展望了未来神经网络处理器验证的工作的开展及研究方向。

第二章　研究背景

2.1　测试的传统方法

2.1.1　软件测试的传统验证方法

随着计算机的普及，软件系统已经深入到生活的各个方面，从计算机软件，到智能手机上的各种应用。超市的终端系统，银行的安保系统，甚至航空航天工程的控制系统，软件的应用在各个领域发挥着重要作用。然而，在软件给我们生活带来便利的同时，也存在着风险——软件的实效将会产生严重的后果，2011 年 7 月 23 日，由于 LKD-T1 型列控中心设备存在严重设计缺陷和重大安全隐患，在浙江省温州市内，两辆动车追尾，造成大量人员伤亡，这便是由于软件存在问题所酿成的惨剧。在这种情况下，人们对软件的正确性、可靠性、可靠安全性和保密性等可信性质给予了十分的关注，如何在软件的开发和运行中保证软件具有高可信性质也成为软件理论和技术的重要研究方向。

早期的验证方法分成黑盒测试和白盒测试 [11]。

白盒测试是指将测试对象看做一个打开的盒子，是考虑系统的内部结构，重点测试系统的每一个动作是否符合定义，因此被称为基于结构的测试。白盒覆盖对测试用例的选择主要看是否能达到对系统内部结构的覆盖而不需考虑软件的功能，有不同的逻辑覆盖准则：语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、路径覆盖等。白盒测试的主要方法由逻辑驱动、基路测试等等，常用工具为 jtest、VcSmith、C++Test 等。

而黑盒测试则是指不考虑系统的内部结构，只按照规格说明测试已定义的功能，因此被称为基于功能的测试。黑盒测试则系统看成一个黑盒子，只关心系统的输入输出，所以测试方法的重点在于如何从输入域中选择待测的测试用例，而不关心程序具体如何实现。黑盒测试的一般方法主要包括：等价类划分、边界值分析、判定表、因果图。黑盒测试的常用工具为 AutoRunner、winrunner。

后来，随着测试验证领域的不断发展，测试理论也不断产生。又产生了如下几种软件测试方法。[12]

基于模型的测试。近年来，由于模型化的方法被广泛用于工程领域，因此模型测试的思想被广泛用于软件测试。测试用例的选择问题可以看做是从庞大的输入、状态组合中，搜寻那些可以发现错误的状态及组合。基于模型的测试主要考虑的是系统的功能，因为模型是系统功能的形式化或半形式化的表示，代表了被测系统的本质，因此比起系统来说，更容易被分析。但采取基于模型的测试必须保证模型能够完全准确地展现测试对象的所有特征，而这样的模型不易建立，因此具有一定局限性。

验收测试。验收测试是指系统开发生命周期方法论的一个阶段，是部署软件之前的最后一个测试操作，因此也被称作交付测试。验收测试的目的是确保软件

准备就绪，并且可以让最终用户将其用于执行软件的既定功能及任务。验收测试一般有三种策略：正式验收、非正式测试和 Beta 测试。

错误驱动测试。错误驱动测试是指，在用户实际使用的过程中，面对用户的非法输入，测试系统的稳定性。功能测试仅能测试系统已实现功能的完备性，而对系统缺少的部分无能为力，因此，需要非法操作或错误的测试来保证系统的稳定性。

回归测试。回归测试就是一个不断发现测试和不断改正错误的过程。由于程序的复杂性，各个模块及元素（变量、函数、类）之间存在着相关联性，所以对于改正的错误，还要进行在测试。一方面检查此错误是否真的被修改了，另一方面检查此错误修改是否引入了新的错误，这就需要将测过的样例拿来重新进行测试，这就是回归测试。理论上，软件产生新版本，都需要进行回归测试，验证以前发现和修复的错误是否在新版本中再次出现。在我们神经网络测试框架的设计中，便采取了回归测试的思想。

2.1.2 编译器的传统测试方法

编译器作为计算机软件中最基础的软件之一，与操作系统、数据库系统一起被列为构成计算机系统关键性基础设施。而编译器作为任何软件的产生器，它的安全性、可靠性和稳定性更是至关重要，特别是在那些软件的可靠性要求很高的特殊环境里面，我们必须保证编译器编译出来的代码是对程序源代码正确、真实的反应，保证编译器在编译过程中逻辑上正确性以及行为上的透明性。

编译器系统可信验证主要包括两方面，一个是编译器的逻辑正确性，即编译器编译的程序在逻辑上符合程序源代码的描述，与程序源代码的逻辑一致；另一方面是编译器的安全性和可靠性，指编译器在编译过程中不会因人为地插入恶意代码，而导致目标程序运行不可靠或者达到某些其他恶意的目的。

下面主要介绍编译器的逻辑正确性。

早期的编译器测试主要以人工为主，而后逐渐转向自动测试 [13]。自动测试的基本原理为随机生成程序，然后用不同的方式编译出运行等价的程序，然后作对比，具体的做法包括：用两个不同的编译器编译，对比运行结果；用一个编译器的不同优化选项编译，对比运行结果；对原程序中不被运行的语句进行修改，然后和原程序的结果进行对比。

Leroy 等人 [14] 为 C 语言编译器提出了一种开发编译器以及在开发过程中形式化验证编译器可信性的方法，他们设计了一种叫作 Coq proof assistant 的工具，在开发编译器过程中验证该编译器是否是正确的。他们的方法侧重于对编译器后端（即中间语言到代码生成过程）的形式化验证，Sandrine[15] 等人扩充了 Leroy 等人的研究，提出了一种形式化的 C 语言编译器可信验证方法，该方法支持 C 语言的一个子集的验证，而且它能够验证编译器的前端过程（即从源代码到中间语言转换过程）的正确性。

如果不能验证整体编译器的正确性，另一种思路则是对每一个编译步骤的正确性进行检查。翻译验证 [16]（translation validation）就是一种基于此思路的编译器正确验证方法，其目标是检查每一个编译步骤的结果，将其与源程序比较，检测可能存在的编译错误，该方法绕开了验证编译器整体的复杂性，在每一步验证编译的正确性，这一步骤主要是用于语法层面的测试。另外，还有一种称为可信验证（credible compilation）[17] 的技术也是采用类似思路，编译产生转换后的代码的同时，生成额外的上下文信息，使用一个简单的验证器来检查编译步骤的正确性。然而，这些分布验证方法只保证了局部却无法顾及整体，还是无法保证编译器本身是没有缺陷的。

为了测验神经网络处理器，我们将结合传统测验方法及深度学习指令集的特性进行设计。

2.2 神经网络处理器编译器测试框架的特性

一般而言，编译器是将一种抽象的高级语言翻译成另一种低级的语言的软件，类似的，神经网络处理器编译器是将一种抽象的神经网络处理器编程语言翻译成底层的神经网络处理器指令序列。同时，通用编译器的编译流程包括：源代码 $\rightarrow$ 预处理器 $\rightarrow$ 编译器 $\rightarrow$ 目标代码 $\rightarrow$ 链接器 $\rightarrow$ 可执行文件。对于编译过程，神经网络处理器编译器与传统编译器类似。但是，由于深度学习指令集的特殊性以及神经网络处理器结构的复杂性，在结构与储存层次方面，神经网络处理器编译器与传统编译器有很多不同。

2.2.1 深度学习指令集的特性

深度学习指令集 (Cambricon) 的特性主要是以下几点 [10]：

1. 数据并行度高。在大多数神经网络技术中，神经元和突触的数据被组织成层，然后以一个统一、对称的方式进行操作。为了适应这些操作，在数据集并行度方面，向量-矩阵 (Vector/Matrix) 指令比传统的标量指令的并行效率更高，因此 Cambricon 采用了矢量-矩阵指令进行设计，提高了数据的并行度。典型的例子就是应用于漏失 (drop-out) 的随机向量 (Random-Vector) 指令，用于在一条指令内部为一个向量进行快速随机初始化，以及应用于激活层的指数向量操作（Vector-Expotential），用于在一条指令内部为一个向量进行快速的非线性变换，而分支跳转的逻辑在神经网络计算任务里，并不像常规计算任务那么复杂，所以指令集的设计上并不需要提供丰富的分支跳转逻辑的支持。

2. 定制矢量-矩阵指令。虽然有许多的线性代数库可以用于科学计算，但是对于神经网络计算来说，这些代数库中定义的操作并不一定是有效且高效的（有的甚至是多余的）。更重要的是，对于传统的线性代数库来说，很多操作不涉及神经网络技术，例如，BLAS 库不支持元素的指数计算，也不支持随机向量在突

触中的初始化、退出，因此，Cambricon 以现有的线性代数库为基础，考虑神经网络技术，重新定制了一个具有小而具有代表性的向量-矩阵指令集。

3 采用片上暂存存储器 (On-chip Scratchpad memory) 而不是寄存器堆来作为计算数据的主存储。由于神经网络技术的计算任务与常规的多媒体计算任务不同，往往需要访问并操作密集、连续、可改变长度的方式访问向量-矩阵数据。因此，使用固定宽度的寄存器不再是最有效且节约成本的选择。因此，Cambricon 将向量寄存器替换为片上暂存缓存器，为每个数据的访问提供灵活的宽度。由于神经网络突触的数据量巨大，重复使用率高。而也会减少向量登记文件所带来的性能损失，使得网络数据并行度更加高效。

这些特性使得 Cambricon 指令集能正确的描述现在大部分的网络，同时，每条指令的描述能力高，对于描述同样的网络来说，所需的指令数量相比通用处理器要少很多。但，这也带来了缺陷，即便是同一条指令，对于不同的参数，指令的生存周期变化很大，且需要访问的存储方位也变化很大，不同类型的指令更是如此，更别说设计不同层次的存储访问。

2.2.2 神经网络编译器与传统编译器的比较

由于深度学习指令集的众多特性，这也使得神经网络编译器与传统编译器有很多不同。

首先，神经网络编译器对应编译的指令粒度较大，对于神经网络处理器来说，一条卷积 (Conv) 指令可以自己设置卷积核的大小、所需要的卷积图像大小，但是传统处理器的指令操作是固定的，这也给我们的验证带来了很多困难。

其次，神经网络编译器具有操作实现多样性的特点。同样的卷积，根据芯片资源不同，调度方式也不同，可能通过一条卷积指令实现，或者是多条卷积指令组合形式实现，十分复杂。

最后，神经网络编译器和传统编译器设计的存储层次也有很大不同，传统的编译器中，其计算指令都是针对寄存器的操作，只有 load 或者 store 才会涉及到寄存器数据和 Cache 或者 RAM 之间的数据交互，但神经网络编译器加入了中间存储这一层，改变了整个编译器的存储层次。

2.2.3 使用测试框架带来的好处

由于神经网络编译器的特殊性，若采用随机指令可能无法覆盖到实际使用需求的指令序列，所以需要实际的使用者，也就是编译器生成的指令来做二次检查，来寻找所有网络集合里那些不易覆盖的“Corner Case”。而测试框架是保证编译器和库这一整套系统正确性的环节，不可或缺。

2.3 本节总结

本节首先介绍了传统软件的测试方法，接着介绍了传统编译器的测试手段，通过分析深度学习指令集的特性，阐明了神经网络处理器编译器与传统编译器的差别，证明了神经网络处理器编译器测试框架的重要性。

第三章　神经网络处理器编译器测试流程

3.1　神经网络处理器的整体架构

神经网络处理器是针对大型神经网络相关计算设计的专用加速硬件。神经网络处理器提供的高效运算包括了卷积，池化，激活，基本矩阵乘法，归一化等操作。通过专门设计的特有物理结构和指令集，可以实现高效的与神经网络相关的计算。物理结构包括特有的物理存储结构和计算核心结构。指令集则是指一系列特殊定义的用于操作底层硬件的二进制编码的集合。神经网络处理器可以是处理器之外的专用加速设备。神经网络处理器可以以不同的形式与现有的通用处理器进行集成，形成神经网络加速系统。按照与现有通用处理器耦合程度（从紧耦合至松耦合），可行的集成方式至少包括以下几种：1. 将神经网络处理器作为 CPU 的功能部件，集成到 CPU 的流水线中；2. 将神经网络处理器与 CPU 集成到系统芯片 SoC 中；3. 神经网络处理器设计为兼容 PCIe 协议的板块，通过 PCIe 与 CPU 进行连接。

在我们现阶段的处理器实现过程中，我们所采用的方式，是通过 PCI Express 高速总线与 CPU 实现互联。PCIe 总线速度快，并且在现在的各种处理器互联中广泛运用。

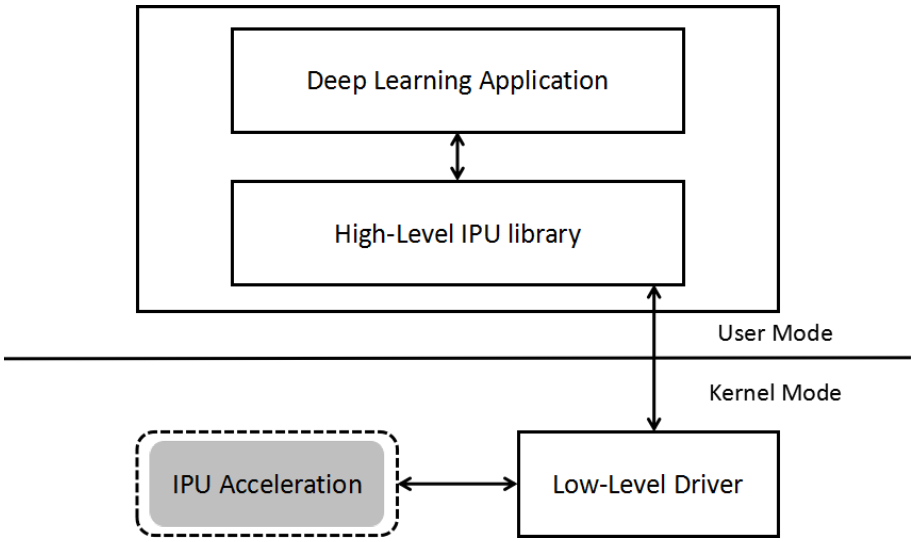


图 3.1　神经网络处理器的架构

在本文测试过程中，使用的已实现的神经网络处理器架构图如图 3.1所示。

在用户的应用程序中，为了调用神经网络处理器去准确方便的完成神经网络相关计算，一套稳定可靠的应用程序接口也是不可或缺的。接下来我们将详细介绍神经网络处理器的软件架构。

3.2 神经网络处理器的软件架构

在阐明验证思路之前，我们应首先来熟悉一下神经网络处理器的软件架构，如图 3.2所示，神经网络处理器的软件部分自上而下主要分成如下几个层次：

1.User Program：上层应用，即程序员基于编程框架，定义特定的神经网络结构来实现特定的神经网络功能。

2.Framework：编程框架，相当于连接的桥梁。例如（Caffe、TensorFlow），为用户提供神经网络基本操作，减轻程序员的编程负担，是一种编程软件。

3.runtime library：Combricon-IPU 库是一个屏蔽掉底层硬件，编译器、启动程序的具体信息，对上层编程框架提供神经网络原子（即神经网络的某一层，这是神经网络最基本的组成单元）操作的编程接口的系统软件，它具有内存分配和释放、基本运算、指令生成和数据传输等功能。

Combricon-IPU 库集成封装了神经网络基本的运算操作，如 Conv（卷积）、Pooling（池化）、Active（激活函数）、BatchNorm（批规范化）、FullyConnected（全连接）等等常见操作，还有更详细矩阵的数学运算如乘法、加法、减法、缩放等等，这样可以直接调用神经网络处理器的接口，就能直接实现神经网络，同时将优化 IPU 的工作封装在 Combricon-IPU 库这一个层次中，而不用直接向应用程序展露。

4.compile：编译器，将上层的库传来的指令描述符按照神经网络处理器的指令集翻译成能接受的机器指令。

driver 驱动：对上层库提供调用底层硬件最基本的操作支持，例如读取数据、启动关闭、读取寄存器等。

5.hardware 硬件：即神经网络处理器 (DaDianNao)

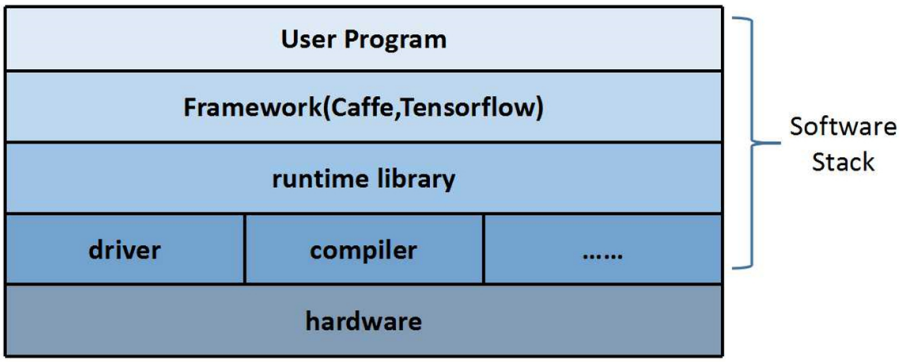


图 3.2 神经网络处理器的软件架构

3.3 测试流程

在还没有神经网络处理器测试框架的时候，我们曾采取随机生成单个指令的方式来进行验证。然而，由于神经网络处理器指令粒度较大，随机指令无法覆

盖到实际使用需求的指令序列，所以需要实际的使用者，也就是编译器生成的指令来做二次审查。这实际上也对应了传统软件测试方法中基于模型测试的思想。

在我们的测试过程中，需要对变量进行控制，由于我们需要测试的是神经网络处理器的编译器，因此主要关注的是库 (runtime library) 以及编译器 (compiler) 的正确性。

对于神经网络处理器编译器的测试流程，可以概括为如下几个步骤，首先，生成包含有网络参数与结构信息的配置文件 prototxt 。接着，根据 prototxt 调用 caffe 提供的接口生成具有训练参数的神经网络模型 caffe_modrel，然后调用 caffe 框架，经过一系列的内存分配，指令生成的操作后，调用库的接口，获得神经网络处理器结果，将这个结果 caffe 中生成的 cpu 结果进行比对，以达到测验目的。

接下来将详细论述流程的步骤。

3.3.1 protext 的生成

要运行 caffe，首先需要先创建一个模型 (model)，caffe 里自带了不少常用的神经网络模型，而一个模型由多个层（layer）构成，每一个层又包含多个参数，每一个层的参数都定义在 caffe.proto 这个文件内。因此，想要生成一个可以用于测试的模型，我们首先得生成一个符合结构要求并含有参数定义的 prototxt。

我们自己编写了一套随机网络生成器，通过这个随机网络生成器，我们可以根据用户的需求，生成结构随机，参数随机且具有正确合理的连接方式的神经网络。同时，这个生成器也可以根据用户需求，生成包含有指定层序列、甚至是结构固定的网络，同时，我们也搭建了一个数据库，分三个层次将一些网络的样例加入数据库中。关于随机网络生成器的细节将在第三章详细概述。

3.3.2 caffe_model 的创建

在得到了包含有网络结构及参数的配置文件 prototxt 后，我们调用 caffe 提供的 caffe_param 接口 (caffe_param 是 caffe 提供的一个记录网络结构与权值并能直接生成 caffe_model 的类)，以此初始化一个 Net 类，在 Net 中我们将权值初始化，再赋以随机权值传回 param 类中，而后 net_param 便可以直接将网络结构以及权值写入 caffe_model 中，我们就生成一个具有随机权值的真实网络。

在权值赋值的过程中，我们既可以手动设置权值的大小，也可以随机赋予。这样子保证了权值能遍历所有大小，也不会忽略一些特殊的情况，例如稀疏权值。

接着，我们先用 caffe 调用 cpu 计算数据通过这个网络的结果，而后调用我们的神经网络处理器。

3.3.3 caffe 重载

caffe 是一个快速的深度学习框架，现在只能调用 CPU 与 GPU 进行计算。因此，为了让 caffe 能够调用神经网络处理器进行运算，我们对 caffe 进行了移植，在框架下加入对神经网络处理器硬件的支持。把原来的 CPU 操作，重载成为神经网络处理器的指令，成为库能接受的操作和运算。

从实现上来看，即将操作的 xxx_Forward_CPU 改写为支持神经网络处理器运算的 xxx_Forward_ipu。同时，我们也在内存调度与空间管理等细节上进行了完善，使得 caffe 可以支持神经网络处理器的运算。

3.3.4 caffe 调用库的过程

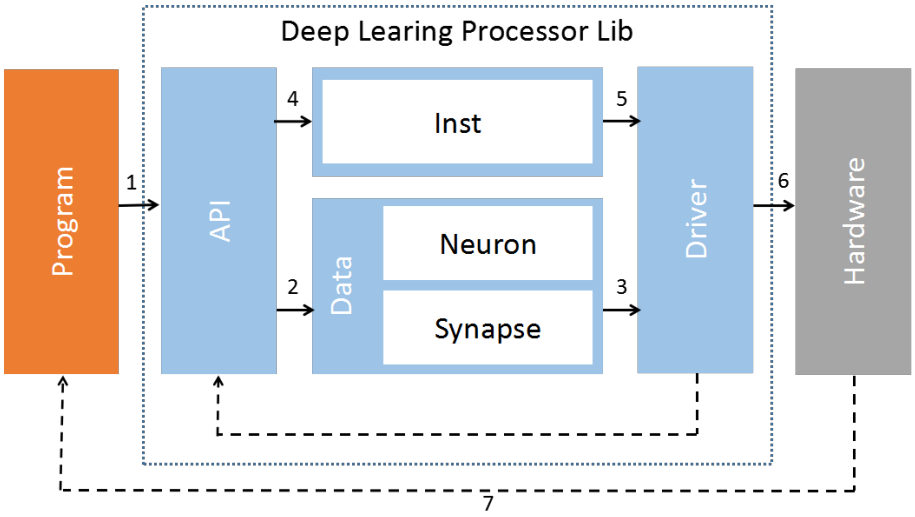


图 3.3 库的执行模型

caffe 运行神经网络处理器的流程如图 3.3所示，库的调用大致分为以下流程：

1. 初始化神经网络处理器的库

2. 声明和设置描述符 (tensor 描述符，算法描述符，操作描述符) 并准备 CPU 端数据

3. 先 Malloc CPU 数据和神经网络处理器数据，再将数据从 CPU 拷贝到神经网络处理器上

4. 生成 Combricon-IPU 指令

5. 通过驱动，将 Combricon-IPU 指令从 CPU 传入神经网络处理器

6. 神经网络处理器进行运算（xxx_Forward 函数调用）

7. 读取结果

至此，我们已经得到了神经网络处理器结果，将这个结果 caffe 中生成的 CPU 结果进行比对，便可以达到验证目的。

3.4 测验工作的进程

对于随机网络生成器来说，现在已经能支持包括卷积，池化，激励等层在内的 18 种层类型的生成。依靠神经网络处理器测验框架，我们基本完成了库的测试工作，保证了寒武纪系列神经网络处理器编译器的的正确性及可靠性。

3.5 本节总结

本节首先先从神经网络处理器的整体构架说起，而后详细的介绍了神经网络处理器的软件构架并着重点出了我们验证的主要目标。接着，从 prototxt 的生成到 caffe_model 的建立，再到 caffe 的的重载直到最后指令的调用，层层深入，详细介绍了神经网络处理器编译器的验证流程，并简要介绍了使用该测验框架后，测验工作的进程。

第四章　随机网络生成器的设计及实现

要运行 caffe，首先需要先创建一个模型 (model)，caffe 里自带了不少常用的神经网络模型，而一个模型由多个层（layer）构成，每一个层又包含多个参数，每一个层的参数都定义在 caffe.proto 这个文件内。因此，想要生成一个可以用于验证的模型，我们必须首先生成一个符合结构要求并含有参数定义的 prototxt。

4.1　随机网络生成器的设计

4.1.1　prototxt 的概述

prototxt 是配置文件，是 caffe 中用于描述网络参数以及网络结构的文件。这个文件里不包含对网络训练的权值，相当于一个网络构架。注明的 AlexNet 网络的 prototxt 的前四层以及其可视化如图 4.1所示。因此，为了得到一个完整的随机网络，我们应该考虑先生成一个 prototxt 文件。

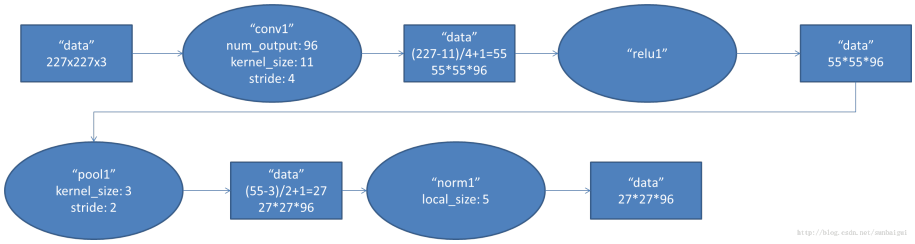
```
layer {
 name: "conv1"
 type: "Convolution"
 bottom: "data"
 top: "conv1"
 param {
  lr_mult: 1
  decay_mult: 1
 }
 param {
  lr_mult: 2
  decay_mult: 0
 }
 convolution_param {
  num_output: 96
  kernel_size: 11
  stride: 4
 }
}
```

```
layer {
 name: "pool1"
 type: "Pooling"
 bottom: "norm1"
 top: "pool1"
 pooling_param {
  pool: MAX
  kernel_size: 3
  stride: 2
 }
}
```

```
layer {
 name: "conv2"
 type: "Convolution"
 bottom: "pool1"
 top: "conv2"
 param {
  lr_mult: 1
  decay_mult: 1
 }
 param {
  lr_mult: 2
  decay_mult: 0
 }
 convolution_param {
  num_output: 256
  pad: 2
  kernel_size: 5
  group: 2
 }
}
```

```
layer {
 name: "pool2"
 type: "Pooling"
 bottom: "norm2"
 top: "pool2"
 pooling_param {
  pool: MAX
  kernel_size: 3
  stride: 2
 }
}
```

(a) AlexNet 网络的 prototxt



(b) AlexNet 网络的可视化

图 4.1　AlexNet 网络图

4.1.2 随机网络生成器的数据结构

在我们神经网络生成器当中，有三个主要的数据结构，其一是用于描述网络中最基本粒子“层”的数据结构 layer，另一个则是用于描述数据在层之间传递方法的数据结构 blob_shape。第三个则是用于读取，判断用户对网络结构需求的数据结构 struct_limit。

下面，我们分别介绍这三个数据结构。首先，对于 blob_shape，这是一个一个密集的 n 维（$n \leq 4$）阵列，在神经网络中，这代表神经元和偏置 (bias)。

在 caffe 中也拥有类似的数据结构 Blob，用于封装了并在层与层之间传递运行时的信息，提供了 CPU 和 GPU 的同步。从数学上来说，Blob 就是一个 N 维数组，是 caffe 中的数据操作基本单位，就像 matlab 中以矩阵为基本操作对象一样。只是矩阵是二维的，而 Blob 是 N 维的。N 可以是 2，3，4 等等。对于图片数据来说，Blob 可以表示为（N×C×H×W）这样一个 4 维数组。其中 N 表示图片的数量，C 表示图片的通道数，H 和 W 分别表示图片的高度和宽度。当然，除了图片数据，Blob 也可以用于非图片数据。比如传统的多层感知机，就是比较简单的全连接网络，用 2 维的 Blob，调用 innerProduct 层来计算即可。

而为了与 caffe 提供的接口相对应，也为了保证生成网络连接方式的合理性，我们的 blob_shape 也拥有和 caffe 中 Blob 类的类似结构。在我们随机网络生成器中，网络内层与层之间的数据，是用 blob_shape 来表示和运算。它的维度会根据网络参数与结构的类型不同而不同。例如，卷积操作取一个四维的 blob_shape 进行输入，而输出一个四维的 blob_shape，而全连接操作则取一个四维 blob_shape 进行输入，输出一个二维 blob_shape。

Parameter	Description
Dim	Dimension of tensor
N	Number of samples
C	Number of feature maps
H	Height of sample
W	Width of sample
Data format	NCHW
Data type	Float32,Float 16

图 4.2　blob shape 的参数

blob_shape 描述的属性和参数如图 4.2所示，N,C,H,W 表示四个维度的大小，其中 N 表示样本数，C 表示特征映射的数量，也就是图片中通道数，而 H,W 分别表示特征映射的高与宽，每一个数据的数据类型是一个枚举类型变量，用于表示张量的数据格式，这些字母的顺序代表了的 blob_shape 数据排列。

值得注意的是，和 caffe 中的 Blob 相比，我们的 blob_shape 还有一个参数 (Dim) 用于描述 blob_shape 的维度。对于维度小于 4 的 blob_shape，我们也可以用相同的格式来表示。例如，一个二维的 blob_shape，我们将其 H 与 W 设为 1，而依靠 Dim 参数区分 H 与 W 都为 1 的四维 blob_shape，这样的好处是，我们可以保证数据的传递的有效性，同时也保证了所有的 blob_shape 在网络的传递过程中计算形式上都具有相同的维度，而 (Dim) 这一参数可以让我们有效地分辨出该数据的实际数值，有效避免了误差。

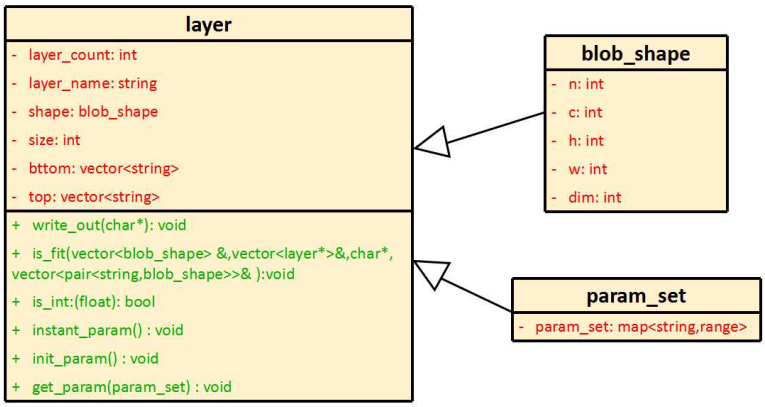


图 4.3　layer 与 blobshape 的类图

对于另一个主要的数据结构——层（layer）来说，神经网络的层是一个独特的概念，这不但代表了一个操作，同时也包含了这个操作的参数，是网络模型的组成要素和基本单位。在 caffe 中，proto.prototxt 文件记录了每个层的参数种类，但 caffe 中 layer 类用作层与层的运算，而非网络的生成。于是，在我们的随机网络生成器中，层 (layer) 被声明为一种新的类。这个类里面的变量与方法如图 4.3所示。

其中 write_out 方法用来打印网络结构，is_fit 方法则用于判断单个层（操作）所连接的位置。instant 方法和 get_param 方法用于初始化参数范围，前者的初始化是根据设计者估计的参数来设置的，而后者的初始化则是由用户设定的，后者的优先级高于前者，最后由 instant 方法在范围内随机得到参数，随机参数的类型（整数或者浮点数）由 is_int 控制。

对神经网络来说，每一个操作 (operator) 都对应了大量的参数，网络中的 layer 与每一个 operation 一一对应。因此也会有众多的参数来描述层的特性。

我们以卷积层 (conv_layer) 为例，说明卷积层内的参数。如图 4.4所示，卷积层内有 10 个参数（后期还可以根据用户的需求进行添加）, 其中 num_output，kernel_size 用于描述核的通道数与长和宽，而 pad 即 stride 则用于描述卷积操作中，核 (kernel) 的具体操作，sparse_percentage 用于定量描绘稀疏图的概率，

Convolution		
num_output	通道数	(20,50)
bias	偏置	(0,1)
position	(用于定义数据的bite)	0
offset		(-15,0)
stride	stride的高与宽	1
pad	pad的高与宽	(1,2)
kernel_size	kernel的高与宽	(3,4)
sparse_percentage	稀疏率	50
num_bottom	bottom的数量	1
num_top	top的数量	1

图 4.4　卷积层的参数类图

这里的每一个参数都在神经网络的卷积操作中有意义，使得程序具有较强的可读性。

第三个数据结构 struct_limit 是用于处理随机神经网络生成器中的结构要求，第一个作用是读取用户单层结构的概率分布，从而按照概率生成所需要的网络，第二个作用则是读取用户输入的串（即给定的 layer 序列、多层结构）以及串的生成概率，在判断串合法的情况下，生成具有特定连接方式的神经网络。struct_limit 类里面的变量与方法如图 4.5所示。

其中 rule 是用于读入用户定义规则的结构体，而 snake 类则定义了一系列规则，其中 add 方法用于添加新的规则，而 dadj_snake 方法则用于调整连接规则（例如 conv-mlp 与 mlp-pooling 规则可以合成 conv-mlp-pooling 一条规则）而 struct_limit 继承了两者，用于调配规则，其中 series 参数用于记录串的形式信息，而 rule 参数则用来记录用户所提供的生成规则，而 check_rule 方法用于判断合法性。

4.1.3　网络的建立

网络是由层 (layer) 构成，而 blob_shape 封装了数据从 layer 一层一层输入再输出。而数据数据能否作为一个层的输入是取决于数据的维度、数值以及层的参数，而数据的输出则在层中根据层的种类与参数进行计算与调配。

卷积层是卷积神经网络（CNN）最重要的一种层的类型，它需要一个四维 blob_shape 作为输出，并输出一个四维 blob_shape。张量的操作通过层来进行，卷积层的参数类型如图 4.4所示，一个 blob_shape 传入卷积层后，N 值保持不变，

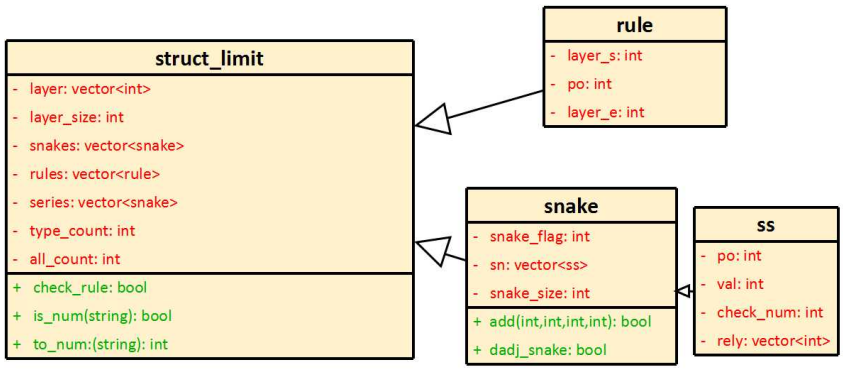


图 4.5　struct limit 的类图

C 值被卷基层的 number_output 所取代，而 H 和 W 则由式 4.1与式 4.2计算：

$$H = \lceil \frac{H_i - K_h + 1 + 2P_h}{S_h} \rceil \tag{4.1}$$

$$W = \lceil \frac{W_i - K_w + 1 + 2P_w}{S_w} \rceil \tag{4.2}$$

其中 H_i 和 W_i 代表输入张量的 H 和 W，K_h 与 K_w 则代表了 kernel 的长和宽，而 P_h,P_w 则代表了 pad 的长和宽，这些都能在层的参数中得到，之后将得到的新的张量存储在 shape 中传向网络里更深的位置（层）内。

在 blob_shape 的传入与输出过程中，is_fit 函数能控制输入的合法性，例如对于平常的卷积层，我们需要保证输入的 blob_shape 是四维的，同时，输出的 blob_shape 里，每一个成员变量都需要大于 0，即还应该满足式 4.3与式 4.4：

$$\lceil H_i - K_h + 1 + 2P_h \rceil > 0 \tag{4.3}$$

$$\lceil W_i - K_w + 1 + 2P_w \rceil > 0 \tag{4.4}$$

对于一些操作复杂的层来说 is_fit 函数还取到了生成补充层的作用，例如 eltwise 层需要输入多个 blob_shape，同时保证输入 blob_shape 的四个维度都相等，而 lstm_unit 层则既输入多个 blob_shape，同时也输出多个 blob_shape，进一步的，我们还可以根据库的要求、乃至硬件的要求调整 is_fit 函数，使得该程序具有很好的可扩展性。

4.2　随机网络生成器的实现

4.2.1　随机网络生成器的运行

主函数首先读取 struct_limit 以及 param_limit 两个文件，前者用于描述网络的结构分布，即网络的深度以及每一类型的层在本次随机网络生成中所占有的

比例，后者则用于描述不同层中参数的限制。这些结构分布与参数限制都根据用户的验证需求自行限制。另外，除了每次进行单个层的生成与连接外，还可以把多个层构成串一并连接并生成。例如，我们可以要求每一个卷积层 (convolution) 后面，连接的都是一个全连接层 (mlp)，这样子尽可能满足了用户的验证需求，也给程序的维护带来了便利。

```
input_dim  1  16  10  10

conv
num_output  10  20
bias  0
position  0
offset  1  16
pad  1  2
kernel_size  3  4
sparse_percentage  90
number_bottom  1
num_top  1

lrn
local_size  0  3

pooling
pool  0  1
pad  1  2
kernel_size  3  4
stride  1  3
number_bottom  1
num_top  1

mlp
sparse_percentage  80
bias  0
number_output  10  20

concat
axis  0  1
num_bottom  2  4

exp
base  1  10
```

(a) 网络结构限制

```
layer_size  50

distribution
conv  10
pooling  2
lrn  2
mlp  5
exp  1
batchnorm  1
softmax  1
scale  1
power  1
reshape  2
relu  1
concat  2
absval  1
bnll  0
deconv  0
eltwise  1
lstm_unit  2
proposal  0
roipooling  0
```

(b) 网络参数限制

图 4.6　proto 的运行

4.3　caffe_model 的生成

在读取参数后，主函数根据用户的需求按规则生成我们所需要的网络，并调用 write_out 方法从而打印生成具有网络信息的 prototxt 文件。我们现在得到了网络信息，然而这个网络还是一个“空架子”，我们接下来先根据每一个层的连接方式，为网络补充上权值，我们写了一个 create_model 函数，主要的作用就是调用 caffe 提供的 caffe_param 接口 (caffe_param 是 caffe 提供的一个记录网络结

构与权值并能直接生成 caffe_model 的类)，初始化一个 Net 类，在 Net 中我们将权值初始化，再赋值传回 param 类中，而后 net_param 可以直接将网络结构以及权值写入 caffe_model 中，至此，我们生成一个具有随机权值的真实网络，简单的说就是我们将一个“空架子”给“填充”了。这样子，便生成了我们所需要的 caffe_model。

4.3.1 生成已有的常用网络

在前文中，我们已经介绍了随机网络模型的生成流程，虽然已有的常用网络（例如 Lenet,Alex 等）是随机网络模型的子集，但常用网络的训练效果已得到同行的验证，因此在我们神经网络处理器验证框架的验证里，具有一定的代表性，同时将已有的常用网络单独验证也可以增强开发者的信心。

对于常用网络，caffe 里面已经给出了现有的 prototxt，甚至也有提供已经训练好的 caffe_model，因此，我们无需再去生成。（事实上，我们的随机网络模型生成器也可以按照给定的串生成常用网络），之后调用 caffe 提供的 caffe_param 接口，便可以生成具有随机权值的常用网络模型了。

4.3.2 测试框架数据库的建立

高效的测试需要有数据库来支持。对于神经网络处理器编译器的验证来说，由于 bug 主要出现在指令集中，由于神经网络处理器的特殊性，即使是同一条指令，不同的参数，指令的生存周期变化很大，且需要访问的存储范围变化也很大，所以我们的数据库里应该记载了完整的神经网络结构。我们的数据库应该由如下几个层次构成：

1. 单个层随机参数的指令，因为随着库能支持的网络越来越多，需要验证的网络种类也会越来越复杂，因此，我们需要对网络的最基本构造——层进行单独验证。因此，对于每一种新的层，我们应该把单层的网络加入到库中。

2. 已有的常用网络，常用的网络具有代表性，同时验证常用网络也会给予开发者信心，因此我们会收集其他同行使用过的网络，在库支持的前提下，对其进行验证。

3. 之前出过错的网络结构。由于神经网络结构的复杂性，因此修正了一个错误有可能会导致其他错误，将这些 corner case 收集，在每一次改良调试库之后，优先对这些网络结构进行验证，可以较容易的发现错误。这里也运用到了前文中介绍的回归测试的软件验证思想。

4.4 本节总结

第四章介绍了神经网络处理器编译器验证框架中最重要的随机网络生成器的实现。吸取了编程框架 caffe 中数据结构的优点并对其结构进行改良，将层的连接方式以及网络的构建融入网络生成器的创建当中，并实现了一系列方法用

于满足用户对结构以及参数的要求，最后，建立了一个数据库用于保证测试的高效性，同时阐述了数据库的设计思想。

第五章　展望与总结

5.1　工作总结

论文从深度学习入手，介绍了现有的 CPU、GPU 在面对大规模的深度学习神经网络应用时的困境，从而引入了专用于深度学习神经网络的神经网络处理器。为了能让用户正确稳定的使用神经网络处理器，神经网络处理器编译器的测试工作是很重要的，同时，由于神经网络处理器内部运算部件和数据存储结构等多方面创新性的不同，一套能用于检测神经网络处理器编译器的测试框架是必要的。

在调研了传统软件以及传统编译器的验证方式后，我们结合传统软件测试的方法和神经网络处理器编译器的独特性质，设计了一套专门用于检测神经网络处理器编译器的测试框架，通过该测试框架，我们提高了测试效率。

最后，我们详细地阐述了测试框架中最重要的一环——随机神经网络生成器，解释了随机网络生成器的功能和实现方法，最后，我们搭建了一个数据库来保证测试工作的高效性并解释了数据库的设计思想。

5.2　下一步研究方向

对于神经网络处理器编译器测试工作来说，我们接下来将继续完善我们的随机网络生成器，增加能支持的操作类型。除外，我们还将继续移植其他主流编程框架、并对新的编译器进行测验。

神经网络处理器编译器的复杂性，决定了我们测试工作的复杂性；而我们测试工作的高效性，则决定了编译器乃至整个神经网络处理器的高效性，为此，我们将会不断努力。

参考文献

[1] Russakovsky O, Deng J, Su H, et al. Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 2015, 115(3):211–252.

[2] Wang F Y, Zhang J J, Zheng X, et al. Where does AlphaGo go: from Church-Turing thesis to AlphaGo thesis and beyond. IEEE/CAA Journal of Automatica Sinica, 2016, 3(2):113–120.

[3] Akopyan F, Sawada J, Cassidy A, et al. TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 34(10):1537–1557.

[4] Abadi M, Agarwal A, Barham P, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467, 2016..

[5] Jouppi N P, Young C, Patil N, et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. arXiv preprint arXiv:1704.04760, 2017..

[6] Chen Y, Chen T, Xu Z, et al. DianNao family: energy-efficient hardware accelerators for machine learning. Communications of the ACM, 2016, 59(11):105–112.

[7] Chen T, Du Z, Sun N, et al. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. Proceedings of ACM Sigplan Notices, volume 49. ACM, 2014. 269–284.

[8] Chen Y, Luo T, Liu S, et al. Dadiannao: A machine-learning supercomputer. Proceedings of Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2014. 609–622.

[9] Liu D, Chen T, Liu S, et al. Pudiannao: A polyvalent machine learning accelerator. Proceedings of ACM SIGARCH Computer Architecture News, volume 43. ACM, 2015. 369–381.

[10] Liu S, Du Z, Tao J, et al. Cambricon: An instruction set architecture for neural networks. Proceedings of Proceedings of the 43rd International Symposium on Computer Architecture. IEEE Press, 2016. 393–405.

[11] Hetzel W C, Hetzel B. The complete guide to software testing. John Wiley & Sons, Inc., 1991.

[12] 许静, 陈宏刚, 王庆人. 软件测试方法简述与展望. 计算机工程与应用, 2003, 39(13):75–78.

[13] 俞甲子. GCC 编译器安全验证方法研究 [D]. 杭州: 浙江大学, 2008.

[14] Hannan J, Pfenning F. Compiler verification in LF. Proceedings of Logic in Computer Science, 1992. LICS'92., Proceedings of the Seventh Annual IEEE Symposium on. IEEE, 1992. 407–418.

[15] Berezin S. Model checking and theorem proving: a unified framework[D]. SRI International, 2002.

[16] Pnueli A, Siegel M, Singerman E. Translation validation. Tools and Algorithms for the Construction and Analysis of Systems, 1998. 151–166.

[17] Rinard M C. Credible compilation. 2003..