

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

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，感谢一路走来过的兄弟姐妹们，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 19 日

目 录

致 谢	I
目 录	III
摘 要	V
ABSTRACT	VII
第一章 绪论	1
1.1 验证的传统方法	1
1.1.1 软件验证的传统验证方法	1
1.1.2 编译器的传统验证方法	2
1.2 神经网络处理器编译器测试框架的必要性	3
1.2.1 深度学习指令集的特性	3
1.2.2 神经网络编译器与传统编译器的比较	4
1.2.3 使用验证框架带来的好处	4
1.2.4 本节总结及研究意义	4
第二章 神经网络处理器编译器验证思路	5
2.1 protext 的生成	6
2.2 caffe_model 的创建	6
2.3 caffe 重载	6
2.4 caffe 运行 ipu 的过程	6
第三章 随机网络生成器的设计思路	9
3.1 prototxt 的概述	9
3.2 随机网络生成器的数据结构	9
3.3 网络的建立	13
3.4 随机网络生成器的运行	13
3.5 caffe_model 的生成	14
3.6 生成已有的常用网络	14
3.7 测试框架数据库的建立	15
第四章 验证框架能的支持及改进	17
4.1 如今能支持的需求	17
4.2 还需要改进的地方	17
第五章 展望与总结	19

摘 要

本文是中国科学技术大学本硕博毕业论文模板示例文件。本模板由 ywg@USTC 创建，适用于撰写学士、硕士和博士学位论文，本模板由原来的本科模板和硕博模板整合优化而来。本示例文件除了介绍本模板的基础用法外，本文还是一个简要的学位论文写作指南。

关键词： 中国科学技术大学 学位论文 L^AT_EX 通用模板 学士 硕士 博士

ABSTRACT

This is USTC thesis template for bachelor, master and doctor user's guide. The template is created by ywg@USTC and a derivative of USTC Bachelor and Master-PhD templates. Besides that the usage of the template, a brief guideline for writing thesis is also provided.

Keywords: University of Science and Technology of China (USTC), Thesis, Universal L^AT_EX Template, Bachelor, Master, PhD

第一章 绪论

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。

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，单核神经网络处理器结构；DaDianNao，面向超大规模神经网络的多核处理器结构；PuDianNao，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深度学习指令集 Cambricnn，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，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。

本节，我们将先回顾软件验证的传统方法以及编译器的传统验证方法，比较神经网络处理器编译器以及传统编译器的不同，分析传统编译器验证的局限性，最后推导出神经网络处理器编译器验证框架的必要性。

1.1 验证的传统方法

1.1.1 软件验证的传统验证方法

随着计算机软件在一些关键领域，例如航空航天、军事武器以及其他对软件可靠性要求极为严格的领域应用越来越广泛，其复杂程度和功能越来越复杂，集成程度也越来越高。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，黑盒测试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，所以又被成为基于功能的测试。黑盒测试则将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。黑盒测试的一般方法主要有：等价类划分（包括有效等价类和无效等价类）、边界值分析（包括有效边界内和边

界外)、判定表(系统输入输出的有效组合)、因果图(系统输入输出的制约关系图)。

而白盒测试则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,因此又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,有不同的多级覆盖准则:语句覆盖、判定覆盖、条件覆盖、判定——条件覆盖、路径覆盖等。

后来,随着软件验证领域的不断发展,验证理论也不断产生。又产生了如下几种软件测试方法。

基于模型的测试。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。如果不适用抽象的手段,有效的测试是不可能达到的。模型化的方法被广泛应用于工程领域,模型是系统功能的形式化或半形化的表示,必须支持输入、状态系统组合的系统枚举,虽然不能产生所有的输入、状态组合,但是模型可以帮助实现这一目的。

错误驱动测试。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。在用户实际使用的过程中,会有大量的非法输入,此时系统表现如何,是否会崩溃,基于非法操作或错误的测试就是错误驱动测试。

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)之间存在着相关性,所以对于改正的错误,还要进行再测试。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。

1.1.2 编译器的传统验证方法

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,这个指编译器在编译过程中不会人为地插入恶意代码,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。

Leroy 等人为 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 等人扩充了 Leroy

等人的研究，也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（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。另外，还有一种称为可信验证（credible compilation）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，这些分布验证方法还是无法保证编译器本身是没有缺陷的。

1.2 神经网络处理器编译器测试框架的必要性

1.2.1 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：

1. 采用基于负载存储 (load-store) 访存模式的 RISC 指令集。具体指令的选取，根据工作负荷 (workload) 的类型进行计算层面的抽象得出。对于深层神经网络来说，主要的计算和控制任务有几种：向量计算、矩阵计算、标量计算、分支跳转。其中向量计算、矩阵计算、标量计算都属于标准计算工作，形式上和通用处理器没有什么区别，主要的差别在于细节的支撑上，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-Exponential)，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. 不引入复杂的 Cache 体系和相关控制逻辑。这跟 AI 算法的 workload 类型有强关联，对于机器学习算法来说算法来说，数据逻辑性并不强，Cache 对性能的影响不像常规计算任务那么大，所以把用于实现 cache hierarchy 的控制逻辑精简掉，对于提升芯片的计算功耗比会有很大的助益。

3. 使用高速暂存存储器 (Scratchpad Memory) 而不是寄存器堆来作为计算数据的主存储。因为机器学习算法的计算任务与常规的多媒体计算任务不同，指令所操作的数据长度往往是不定长的，所以应用于多媒体指令优化 (SIMD) 的寄存器堆就不如 Scratchpad Memory 灵活。

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。

1.2.2 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。

1.2.3 使用验证框架带来的好处

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。

1.2.4 本节总结及研究意义

第二章 神经网络处理器编译器验证思路

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构架，如图 2.1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，是一种编程工具。

3.runtime library：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。

4.driver、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。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)

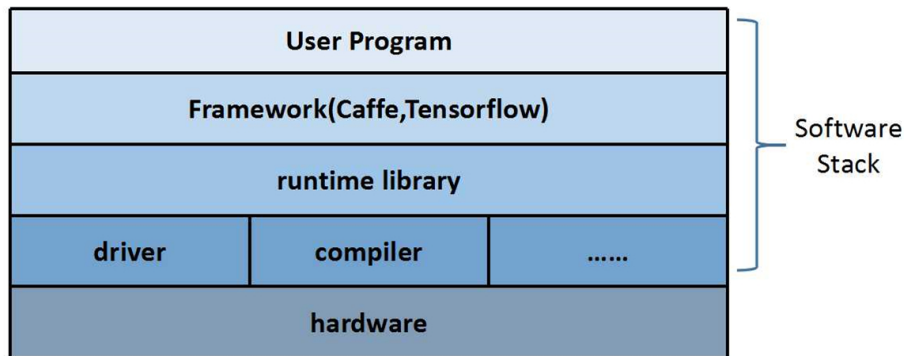


图 2.1 神经网络处理器的软件架构

验证需要对变量进行控制，由于我们需要验证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验证流程，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 caffemodel，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，以达到验证目的。

2.1 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。

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。

2.2 caffe_model 的创建

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, 我们便生成一个具有随机权值的真实网络。

2.3 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。

2.4 caffe 运行 ipu 的过程

caffe 运行神经网络处理器的流程如图 2.2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)
3. 准备数据 (nalloc CPU 数据和神经网络处理器数据,CPU 到神经网络处理器的数据拷贝)
- 4.IPU 运算 (xxx_Forward 函数调用)
5. 读结果

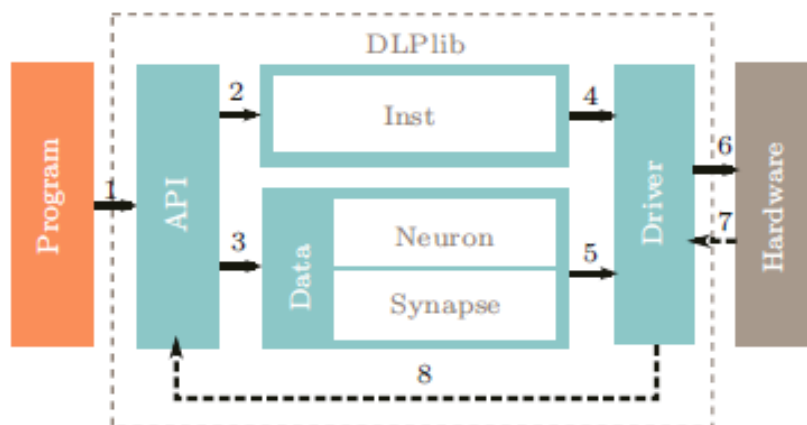


图 2.2 库的执行模型

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。

第三章 随机网络生成器的设计思路

3.1 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 3.1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
  }
}

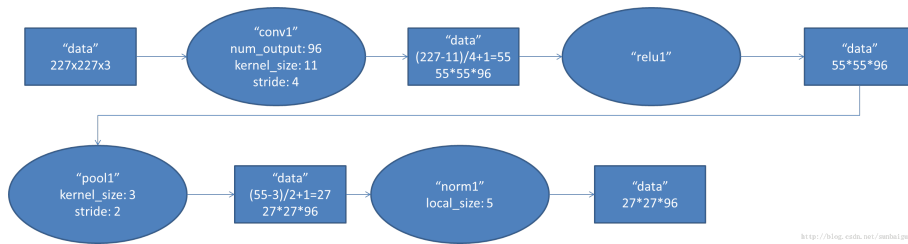
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}

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: "conv2"
  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 3.1 AlexNet 网络图

3.2 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 $(N \times C \times H \times W)$ 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 进行输入，输出一个二维 Blob。

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 3.2 blob shape 的参数

blob_shape 描述的属性和参数如图 3.2 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 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，我们也可以采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。

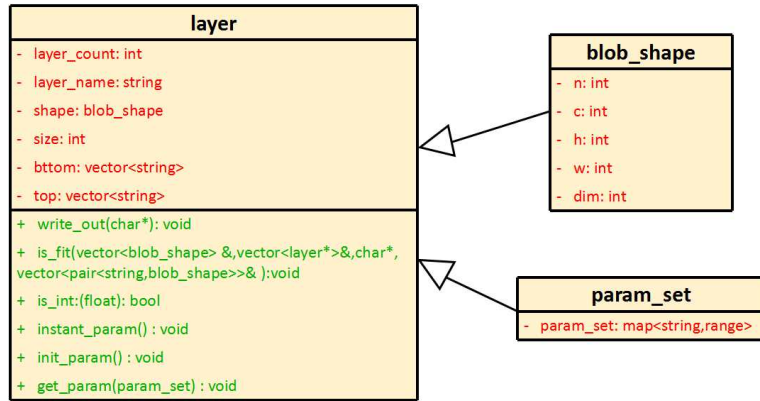


图 3.3 layer 与 blobshape 的类图

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 3.3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 3.4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，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 3.5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

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 3.4 卷积层的参数类图

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 继承了两类，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。

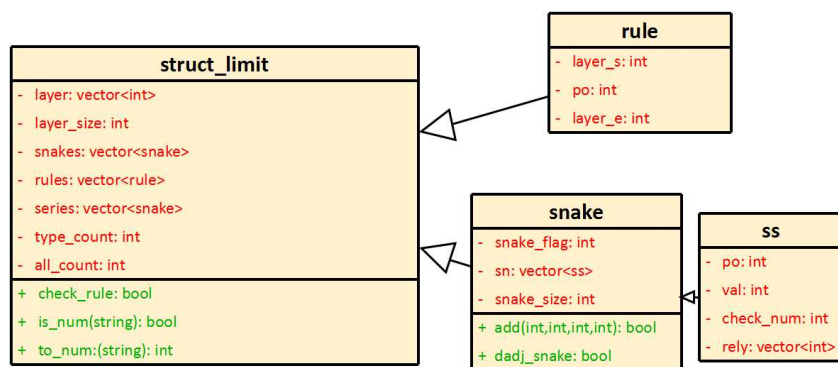


图 3.5 struct limit 的类图

3.3 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 3.4 所示，一个 `blob_shape` 传入卷积层后，N 值保持不变，C 值被卷基层的 `number_output` 所取代，而 H 和 W 则由式 3.1 与式 3.2 计算：

$$H = \lceil \frac{H_i - K_h + 1 + 2P_h}{S_h} \rceil \quad (3.1)$$

$$W = \lceil \frac{W_i - K_w + 1 + 2P_w}{S_w} \rceil \quad (3.2)$$

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 3.3 与式 3.4：

$$\lceil H_i - K_h + 1 + 2P_h \rceil > 0 \quad (3.3)$$

$$\lceil W_i - K_w + 1 + 2P_w \rceil > 0 \quad (3.4)$$

对于一些操作复杂的层来说 `is_fit` 函数还起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。

3.4 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。

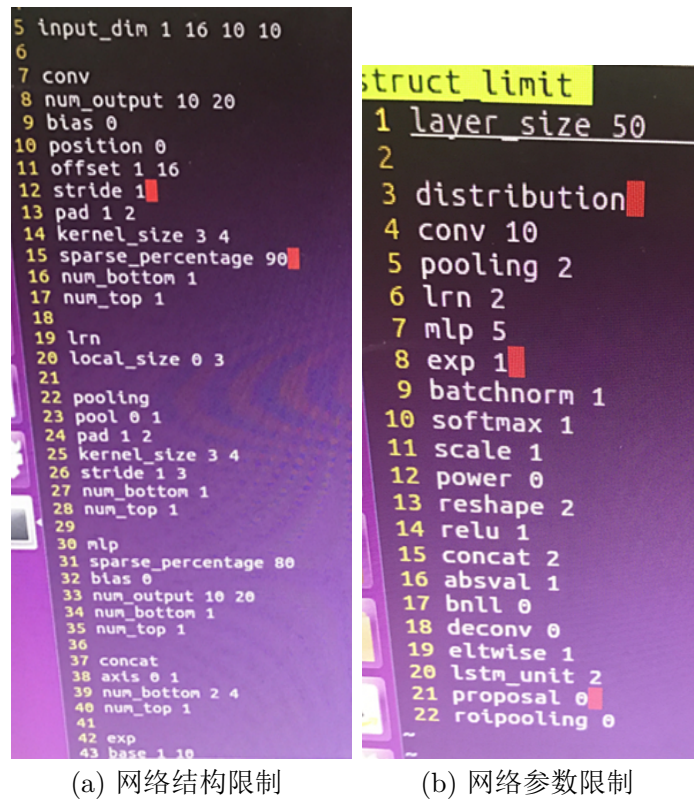


图 3.6 proto 的运行

3.5 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。

3.6 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。

3.7 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 (DLPLib) 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对其他错误, 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。

第四章 验证框架能的支持及改进

4.1 如今能支持的需求

4.2 还需要改进的地方

第五章 展望与总结

