**DNNDK学习笔记(1)**

论坛：https://forums.xilinx.com/t5/Deephi-DNNDK/bd-p/Deephi

DNNDK：

1. SDK的文档结构。

host\_x86：包含了需要拷贝到ubuntu16.04的文件。

common：用于保存一些示例的图片和文件等。

docs：用户手册。

“board name”: 实用工具、DP驱动、DPU运行时和开发库、设备说明书。

HOST SETUP：

1. host\_x86中包含了DECENT（深度压缩工具）和DNNC（深度神经网络编译器），他们可以优化和加速

DPU的推断引擎.

2. 下载并解压了DNNDK后, 复制"host\_x86" 文件夹到一个64bit 的ubuntu14.04或16.04系统中，同时需要

使用Nvidia GPU card。

3. DNNDK需要安装一些其他的软件来运行。

ubuntu14.04 LTS CUDA8.0(GA2), cuDNN 7.0.5

ubuntu16.04 LTS CUDA 8.0 (GA2), cuDNN 7.0.5

CUDA 9.0, cuDNN 7.0.5

CUDA 9.1, cuDNN 7.0.5

4. 安装一些Caffe v1.0 的依赖库

sudo apt-get install -y --force-yes build-essential autoconf libtool libopenblas-dev libgflags-dev libgoogle-glog-dev libopencv-dev libprotobuf-dev protobuf-compiler libleveldb-dev liblmdb-dev libhdf5-dev libsnappy-dev libboost-all-dev libyaml-cpp-dev libssl-dev

5. 安装CUDA

从英伟达官网 https://developer.nvidia.com/cuda-toolkit，下载ubuntu系统所用的CUDA,并安装到"/usr/local/"目录下面。

6. 安装cuDNN

从官网下载对应于CUDA版本的cuDNN。 https://developer.nvidia.com/cudnn 解压缩并拷贝到

“usr/local/"目录下。使用下面的命令：

$ sudo tar -xzvf cudnn-9.1-linux-x64-v7.tgz

$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include

$ sudo cp cuda/lib64/libcudnn\* /usr/local/cuda/lib64

$ sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn\*

最后，在“host\_x86”目录下面执行命令：

./install.sh board\_name #安装软件到主机上。

#board name只能用 DP-8020 / DP-N1 / Ultra96 / ZCU104 / ZCU102其中之一。

-------------------------------------------------------------------------------

CUDA Toolkit：

1. CUDA Toolkit包含了GPU-accelerated库、调试和优化工具、C/C++编译器和运行时库，来进行应用开发。

GPU-accelerated库，支持多个领域的加速。如现行代数、图像和音频处理、深度学习和图像分析领域。

2. 从CUDA体系结构的组成来说，包含了三个部分：开发库、运行期环境和驱动

3. CUDA 的核心有三个重要抽象概念： 线程组层次结构、共享存储器、屏蔽同步（barriersynchronization），

可轻松将其作为C语言的最小扩展级公开给程序员。

4. CUDA 软件堆栈由几层组成，一个硬件驱动程序，一个应用程序编程接口（API）和它的Runtime，还有二个高级的通用数学库，CUFFT 和CUBLAS。硬件被设计成支持轻量级的驱动和Runtime 层面，因而提高性能。

安装CUDA：

Installation Instructions:

`sudo dpkg -i cuda-repo-ubuntu1604-10-0-local-10.0.130-410.48\_1.0-1\_amd64.deb`

`sudo apt-key add /var/cuda-repo-<version>/7fa2af80.pub`

`sudo apt-get update`

`sudo apt-get install cuda`

更新升级：

sudo apt-get install cuda

sudo apt-get install cuda-drivers

-------------------------------------------------------------------------------

深度学习是人工智能和机器学习的一个子集，它使用多层人工神经网络在目标检测、语音识别、语言翻译等任务中提供最高(state of the art)的准确性。深度学习不同于传统的机器学习技术，它可以自动从图像、视频或文本等数据中学习表示，而无需引入手工编码的规则或人类领域知识。它们高度灵活的体系结构可以直接从原始数据中学习，并在提供更多数据时提高预测精度。

**DNNDK学习笔记(2)**

ZCU104 Introduction：

1. zcu系列适合大吞吐量且需要低延时的深度学习推断需求。DNNDN支持的开发板上的SD卡系统镜像可以

从网站下载， http://www.deephi.com/technology/boards。在使用DNNDK v2.08前，需要下载最新版本的镜像文件。

2. zcu104开发板，使用mid-rang的ZU7 UltraScale+ 器件，使用户可以快速进行机器学习应用。更多zcu104

的信息，请访问Xilinx网站. https://www.xilinx.com/products/boards-and-kits/zcu104.html

双B4096F DPU核心在程序逻辑中实现，为深度学习推理加速提供2.4 top INT8峰值性能。在Xilinx的ZU7上实现主频333Mhz的双B4096F DPU。

3. 在DNNDK v2.08 中不包含多线程的ResNet-50、ADAS 检测和Segmentation三个网络。因为他们对电源的

要求更高，zcu104默认配置不支持。有单独的补丁包可以解决这个问题。如果需要，可以联系DNNDK技术

支持团队。

下面使用DP-8020演示，如何在评估板上设置DNNDK运行环境：

note：推荐使用ETCHER，将img镜像文件写入TF卡中。

Boot the evaluation board：

1. 对于DP-8020板子来说，先连接电源12V/2A；

2. 连接UART调试接口到host和其他外设；

3. 打开电源，等待系统启动；

4. 登入系统；

5. 第一次登陆，需要修改一下配置，然后重启生效。

有3种方法登录开发板：

One： UART

Linux 系统中可以安装Minicom或者类似的软件。

Two：Ethernet

可以通过网口，SSH进行连接登录目标板。

Three：Standalone

可以接上显示器、鼠标和键盘，直接单机运行。

Copy DNNDK tools to the evaluation board：

1. 连接建立后可以将DNNDK包中的文件从host复制到目标板上，为了减小DNNDK包的大小，board\_name/

common中包含了示例程序需要的图片的link链接。

Note：如果在windows 上进行了解压，会破坏这种链接文件。

2. 在windows系统中需要先安装 MobaXterm。

然后点击“Start local terminal”，这个位置可以被访问；

假设DNNDK包位于"D"盘的根目录，使用下面命令将Windows中的DNNDK包复制到，IP地址为192.168.0.10的

目标板中；

cd d:

tar -xzvf deephi\_dnndk\_v2.08\_beta.tar.gz

cd deephi\_dnndk\_v2.08\_beta/

scp -r ./DP-8020 root@192.168.0.10:~/

最后，在目标板上，切换到“～/DP-8020”文件夹，运行 install.sh后出现，安装成功，并reboot。

reboot后可以自动替代前一版本。

**DNNDK学习笔记(3)--Windows下进入目标板boot**

《ug1267-zcu104-eval-bd.pdf》

-------------------------------------------------------------------------------

BOOT：

需要配置从SD卡启动，见zcu104的板子说明：SW6拨码进行设置：

Boot Mode SW6[4:1]

-------------------------------------------

jtag ON,ON,ON,ON （0x00）

QSPI32 ON,ON,OFF,ON（0x02）默认

SD1 OFF,OFF,OFF,ON（0x0E）

JTAG:

通过FT4232，4组USB-->UART可以建立Vivado\SDK\第三方工具与Zynq UltraScale+MPSoc器件的jtag连接。

Quad SPI:

1. 先将boot image存到Quad SPI flash中，连接MIO Quad SPI接口线。

2. 设置SW6，从QSPI进行启动。

3. 重新启动电源，按电源或SW4按键。

SD:

基本同上。

--------------------------------------------------------------------------

启动：

1. 连接USB-->UART线到FT4232芯片接口处的micro USB接口。

2. 插入写入image的SD卡

3. 设置SW6的拨码。

4. PC端可以通过Windows中的Teraterm软件，通过串口连接zcu104evb。其中的第一个USB Serial Port(COMx)

5. 插入电源线，并开机。查看启动信息。

用户名：**root**

密码：**root**

#ifconfig -a：

eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500

inet 192.168.200.158 netmask 255.255.255.0 broadcast 192.168.200.255

ether 86:a3:7a:80:a7:42 txqueuelen 1000 (Ethernet)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

device interrupt 15

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

inet 127.0.0.1 netmask 255.0.0.0

inet6 ::1 prefixlen 128 scopeid 0x10<host>

loop txqueuelen 1000 (Local Loopback)

RX packets 22 bytes 2030 (1.9 KiB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 22 bytes 2030 (1.9 KiB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

网口的配置：

Network Configuration

=====================

1.1 Using DHCP to automatically configure the interface

If you're just using DHCP then all you need is creating the configuration entry

in the "/etc/network/interfaces" file as the following.

"

auto eth0

allow-hotplug eth0

iface eth0 inet dhcp

"

or

1.2 Using static IP to configure the interface

The network interface served by the static IP is configured by creating the

configuration entry in the "/etc/network/interfaces" file as the following.

"

auto lo

iface lo inet loopback

auto eth0

allow-hotplug eth0

iface eth0 inet static

address 192.168.200.158

netmask 255.255.255.0

gateway 192.168.200.1

dns-nameservers 114.114.114.114

"

Caution:

The IP addresses used in the above example are not meant to be copied literally.

You have to adjust IP numbers to your actual network configuration.

1.3 Restart Network

Restart network services using the following command via serial console.

"

# /etc/init.d/networking restart

"

or by reboot the board

"

#reboot

"

1.4 Install App

Apt uses a file that lists the 'sources' from which packages can be obtained.

This file is /etc/apt/sources.list. User need update apt's internal

database of available packages as the following after network configure done:

Below is an example of how to update a /etc/apt/sources.list for Debian 9/Stretch.

"

deb http://deb.debian.org/debian stretch main

deb-src http://deb.debian.org/debian stretch main

"

Caution:

To speed up the package download, user could choose to the best server in your country.

then run the following command to install the package

"apt-get update"

"apt-get install APP"

**DNNDK学习笔记(4)--复制SDK到目标板上,运行sample**

安装DNNDKv2.08到开发板：

1. 通过网口可以将HOST端的DNNDK包，复制到评估板上。有2中方法：

a. 在Windows上，通过终端的scp命令可以将文件复制到开发板。

假设开发板 IP为：192.168.200.158，且压缩文件放在D盘根目录中，即d:/deephi\_dnndk\_v2.08\_beta.tar.gz

cd /d d:\ #cmd中切换到d盘中

cd deephi\_dnndk\_v2.08\_beta/ #tar.gz解压后的文件夹deephi\_dnndk\_v2.08\_beta

scp -r ./ZCU104 root@192.168.200.158:~/ #使用scp命令 -r 递归复制文件到(用户名)目标板的～/目录下。

进入“board name”文件夹，然后执行install.sh，安装。

b. 可以使用MobaXterm，使用ssh建立连接，可以直接将压缩包上传到zcu104 evb中。

解压# tar -xzvf deephi\_dnndk\_v2.08\_beta.tar.gz

进入“board name”文件夹，然后执行install.sh，安装。

NOTE:

安装DNNDKv2.08将会自动替换之前版本，不需要手动卸载之前版本。

安装完之后reboot就行了。

运行sample程序：

1. 成功安装完DNNDK后，就可以编译并运行所有示例程序了。samples放在“$deephi\_dnndk\_package/ZCU104/samples/“

文件夹下面。下面用“$dnndk\_sample\_base”来代替“$deephi\_dnndk\_package/DP-8020”。

2. 假设主机的 IP地址为192.168.200.111，使用SSH终端登录开发板时，确保用下面命令使能X11转发功能，因为所有示例

需要运行在Linux窗口系统下运行。

export DISPLAY=192.168.200.111:0.0

3. 每个sample程序，进入相应的文件夹，运行”make“生成hybrid DPU可执行文件，然后像正常Linux程序一样运行就行了。

NOTE:

A. DNNDK示例程序不能在串口连接中工作，因为需要Linux窗口环境；

B. 有时候显示器(monitor)会闪烁flicker(卡)，是因为DPU需要大量内存带宽；

可以使用“X11”进行转发，将SSH sever通过 X11转发到host的SSH client上，在ssh client端，使能 -X

#ssh -X root@192.168.200.158可以将sample的图像相关的新结果转发到Linux VM上。

\*-----------------------------------------------总结：---------------------------------------------------------------------\*

\* 使用SSH连接，然后 expor DISPLAY=192.168.200.111:0.0，进入相应“sample/”中，#make 运行 #./\*\*\* 。 \*

\*---------------------------------------------------------------------------------------------------------------------------\*

为什么得到5组数据，就退出SSH了？

如何使用CUDA和cuDNN与DNNDK进行工作？

使用standalone直接接上HDMI显示器为什么没有输出桌面系统? 不是使用HDMI进行显示，可能是使用DP(Display Port)口

进行显示。

**DNNDK学习笔记(5)--DNNDK的组件说明**

第3章：版本

版本查看：

1. DNNDK中的“host\_x86”文件夹包含了主机端的工具DECENT 和 DNNC

decent --version

dnnc --version

DNNDK包含的组件：DExplore、DSigh、DPU driver、 runtime N2Cube.运行命令查看版本

dexplorer -v

升级与移植：

1. N2Cube runtime APIs增加了新的中断处理模式，可以在不退出运行的情况下打印log信息。使用dpuGetExceptionMessage()

打印log信息。为了兼容之前版本，默认是不处理error并推出运行的，可以使用 dpuSetExceptionMode()使用新的模式。

2. 增加了4个API函数到 libn2cube中：

dpuRunSoftmax()

dpuSetExceptionMode()

dpuGetExceptionMode()

dpuGetExceptionMessage()

DNNDK

1. DNNDK为DPU(Deeplearning Processor Unit)提供了全栈的深度学习SDK。它为深度神经网络推断应用提供了裁剪、量化、

编译、优化、运行时支持。有如下的特点：

\* 提供全套的工具链，包括压缩、编译、运行时

\* 轻量级C/C++ 编程APIs

\* 易于使用与循序渐进的学习曲线

DNNDK使没有FPGA知识的开发者，使用轻量级C/C++ APIs开发深度学习推断应用变得简单。摒弃了底层FPGA的复杂。

2. DPU

DPU用来加速计算深度学习推断算法计算，广泛用于向图像/视频分类、语义分割、物体检测和跟踪等计算机视觉应用。

高效张量级指令集被用于支持流行的卷积神经网络，像VGG、ResNet、GoogLeNet、YOLO、SSD、MobileNet等。DPU可以

被裁减，适用于多款Xilinx Zynq/MPSoC 器件。从边缘到云端，满足多种应用需求。

DPU包含：

DECENT(Deep Compression Tool )、

DNNC(Deep Neural Network Compiler)、

DNNAS(Deep Neural Network Assembler)、

N2Cube(Neural Network Runtime)、

DPU Simulator、

DPU Profiler、

\*\* DECENT

推理过程计算量大，需要较高的内存带宽来满足边缘应用程序的低延迟和高吞吐量要求。DECENTTM (Deep Compression Tool)

，采用粗粒度的修剪、经过训练的量化和权重共享来解决这些问题，同时以非常小的精度降低实现高性能和高能效。

\*\* DNNC

DNNC?(深层神经网络编译器)是DPU专用编译器设计。它将神经网络算法映射到DPU指令，通过平衡计算工作负载和内存访问

来实现DPU资源的最大利用率。

\*\* N2Cube

N2Cube(中立网络的多维数据集)是DPU运行时引擎。它充当DNNDK应用程序的加载程序，处理资源分配和DPU调度。

其核心组件包括DPU驱动程序、DPU加载程序、跟踪程序和应用程序开发的编程api。

N2Cube 通过一个库提供了一组轻量级编程接口，该库抽象出底层硬件实现的细节。

DPU Driver 在Linux操作系统的内核空间中运行，包括任务调度和高效的内存管理等DPU功能，以避免DPU和CPU之间的

内存复制开销。

DPU Loader 负责将DPU代码和数据动态加载到DPU专用内存空间，并执行运行时地址重定位。

\*\* DNNAS

DNNAS (Deep Neural Network Assembler)负责将DPU指令汇编成ELF格式的二进制代码。它属于DNNC代码生成后端组件

，不能单独调用。

\*\* Profiler

DPU Profiler 由两部分组成:DPU Tracer和DSight。DPU Tracer是在DNNDK runtime N2cube中实现的，它负责在DPU上

运行神经网络时收集原始剖析数据。使用提供的原始分析数据，DSight可以帮助生成用于性能分析的可视化图表。

DPU Profiler 使程序员能够分析DPU代码的效率和资源的利用率。

**DNNDK学习笔记(6)--网络部署概述**

第4章：网络部署概述

1.

深度学习应用基本分两步：训练、推断。训练阶段是利用大量训练数据设计为特定任务(如图像分类)设计神经网络。

推理阶段包括部署之前神经网络处理之前没有见过的新输入数据。

DNNDK工具链提供了一个创新的工作流，可以有效地在DPU上部署深度学习推理应用程序，只需5个简单步骤:

1. 压缩 神经网络模型

2. 编译 神经网络模型(分析网络，生成DPU支持和不支持的指令。)

3.DNNDK api 编程

4. 编译混合DPU应用程序

5. 运行混合DPU可执行文件

NOTE:

SDK中不包含DECENT的prune工具\*\*只有quantization工具。

2. 以ResNet为例

---------------------Table 8：DECENT的输入文件--------------------------

1 float.prototxt Floating-point model for ResNet-50

2 float.caffemodel Pre-trained weights file for ResNet-50

3 calibration dataset A subset of the training set containing 100 to 1000 images

a. Network compression

路径$deephi\_dnndk\_package/host\_x86/models/resnet50下的，脚本文件“decen.sh”用于调用(invoke) DECENT工具用近似参数实现量化。

NOTE:

运行量化ResNet-50程序前，DECENT使用的校准数据集dataset需要先准备好。可以从下面网址中下载100到1000张图片， http://www.image-net.org/an 。然后修改ResNet-50 prototxt中image\_data\_param的“source” “root\_folder”设置。

脚本执行后会在“/decent\_output/”文件夹下生成两个文件：

“deploy.prototxt” ：Quantized network description file

“deploy.caffemodel” ：Quantized Caffe model parameter file (non-standard Caffe format)

他们会用于之后编译工程。

b. Network compilation

在文件夹$deephi\_dnndk\_package/host\_x86/models/resnet50有脚本文件“dnnc.sh”。它调用DNNC工具，用合适的选项

对模型进行编译。

运行脚本将ResNet-50模型编译，生成两个ELF格式DPU内核文件。其中包含ResNet-50网络的DPU指令和参数，

还显示了哪些是DPU不支持的层，如图34所示。将ResNet-50网络模型编译成4个不同的内核：

\* Kernel 0： resnet50\_0 （ run on DPU）

\* Kernel 1： resnet50\_1 （ deploy on the CPU）

\* Kernel 2： resnet50\_2 （ run on DPU）

\* Kernel 3： resnet50\_3 （ deploy on the CPU）

DNNC为每个kernel在“/output\_dir/”文件夹下生成了名字为：“dpu\_resnet50\_0.elf”和“dpu\_resnet50\_2.elf”的可执行文件。

“resnet50\_1”和“resnet50\_3” 是为了 “Average Pooling”和“Softmax”操作，他们不能部署到DPU中，只能部署到CPU

中进行运行。

3. 使用DNNDK编程

在DPU上开发深度学习应用，需要3类工作：

\* 使用DNNDK api管理DPU内核

- DPU内核的创建和销毁

- DPU任务创建

- 管理输入和输出张量

\* 实现DPU不支持的kernels

\* 添加预处理和后处理例程来读取数据或计算结果

a. ResNet-50示例:

对于ZCU102，ResNet-50的示例代码存放在“$deephi\_dnndk\_package/ZCU102/samples/resnet50/”文件夹下，

DPU kernels 和 tasks的代码写在main函数中。（kernel应该理解成网络）。

int main(void)

{

/\* DPU Kernels/Tasks for running ResNet-50 \*/

DPUKernel\* kernelConv;

DPUKernel\* kernelFC;

DPUTask\* taskConv;

DPUTask\* taskFC;

/\* Attach to DPU driver and prepare for running \*/ 调用dupOpen()开启一个DPU 设备；

dpuOpen();

/\* Create DPU Kernels for CONV & FC Nodes in ResNet-50 \*/ 调用dupLoadKernel()来装入DPU kernels

kernelConv = dpuLoadKernel(KERNEL\_CONV);

kernelFC = dpuLoadKernel(KERNEL\_FC);

/\* Create DPU Tasks for CONV & FC Nodes in ResNet-50 \*/ 调用dpuCreateTask()来为每个DPU kernel创建task

taskConv = dpuCreateTask(kernelConv, 0);

taskFC = dpuCreateTask(kernelFC, 0);

/\* Run CONV & FC Kernels for ResNet-50 \*/ 分别调用“计算平均池化”和“计算Softmax”

runResnet50(taskConv, taskFC);

/\* Destroy DPU Tasks & release resources \*/ 调用两个函数解除kernel和task

dpuDestroyTask(taskConv);

dpuDestroyTask(taskFC);

/\* Destroy DPU Kernels & release resources \*/

dpuDestroyKernel(kernelConv);

dpuDestroyKernel(kernelFC);

/\* Detach DPU driver & release resources \*/ 最后关闭DPU设备

dpuClose();

return 0;

}

b. 图像分类示例:

在函数 runResnet50()中完成，它执行下面的操作：

- 使用 OpenCV的 imread()获取图像，并调用 dpuSetInputImage2() API将图像设为DPU的resnet50\_0的输入；

- 调用 dpuRunTask()在ResNet-50模型中运行taskConv卷积操作；

- 对前一个卷积操作的输出进行平均池化，并将taskFC操作的输入设置为输出；

- 调用 dpuRunTask在DPU上运行完全连接的操作taskFC；

- 使用完全连接操作的输出作为输入，在CPU上执行softmax；

- 输出前5个分类类别和相应的概率；

Mat image = imread(baseImagePath + imageName);

dpuSetInputImage2(taskConv, CONV\_INPUT\_NODE, image);

dpuRunTask(taskConv);

CPUCalcAvgPool(taskConv, taskFC);

dpuRunTask(taskFC);

/\* Get FC result and convert from INT8 to FP32 format \*/

dpuGetOutputTensorInHWCFP32(taskFC, FC\_OUTPUT\_NODE,

FCResult, channel);

CPUCalcSoftmax(FCResult, channel, softmax);

TopK(softmax, channel, 5, kinds);

4. 混合编译

切换到相应目录下，“$deephi\_dnndk\_package/samples/resnet50”执行 make，将生成CPU的二进制代码，并连接到

DPU kernels “dpu\_resnet50\_0.elf” and “dpu\_resnet50\_2.elf”中。

5. 运行。

**DNNDK学习笔记(7)--网络压缩&量化**

第5章

1.

DECENT包含两个功能，粗颗粒的裁剪和经过训练的量化(less bit param)。这两步减少了执行操作并量化的权重。

图35显示了示意图，在这个版本中只包含了量化工具，不包括prune工具。

通常，在训练神经网络时使用32位浮点权值和激活值。DECENT通过转换32bit的浮点权重为INT8

可以降低计算复杂度，且不降低预测精度。int8权重降低了内存带宽需求，可以提供更快的速度和更高的效率。Quantization工具

支持神经网络中常用的layer，像卷积、池化、fully-connected全连接层、batch norm批标准化。

量化处理后不需要再进行训练，只需要一小组图像数据，分析激活值的分布，进行校准。

量化的时间从几秒到几分钟，取决于神经网络的大小。

DECENT基于Caffe v1.0 并且需要GPU来运行它。这个版本只支持Caffe format。

NOTE:

注意，DECENT的输出文件是一个扩展Caffe模型，只能当作DNNC 编译器的输入。

2. DECENT使用

DECENT支持的选项如下，见Table 10.

3. 流程

a. 准备神经网络模型

在运行DECENT前，需要以（浮点格式）和（校准set）准备Caffe模型，包括：

\* Caffe浮点网络模型 prototxt文件；

\* 预训练的Caffe 浮点网络模型 caffemodel文件；

\* 校准数据集 校准set通常是实际使用图像或训练set的一个子集，至少100张。确保在image\_data\_param

中设置正确的（root\_folder）和（源）为实际校准图像和图像文件夹路径，如图36中：

source："./data/imagenet\_256/calibration.txt”

root\_folder: "./data/imagenet\_256/calibration\_images/"

NOTE: DECENT只支持3平均值，需要转换为3-mean-value。

b. 运行

运行命令行，来生成定点模型：

$decent quantize -model float.prototxt -weights float.caffemodel [options] #其中[options]表示可选参数。有

3个常用的可选参数：

\* weights\_bit: 量化权重和偏执的位宽（默认是8）

\* data\_bit: 量化激活值的位宽 （默认是8）

\* method: 量化方法。0代表OVER\_FLOW；1代表DIFF\_S。（默认为0，减少执行时间。）

c. 输出

运行上面命令后，在默认文件夹". /quantize\_results/"下生成2个文件。他们用作DNNC的输入文件：

\* 定点网络模型 （deploy.prototxt）

\* 定点权重值 （deploy.caffemodel）

**DNNDK学习笔记(8)--网络分析编译**

第6章

网络编译

1. DNNC概览

DNNC编译器的体系结构如图37所示。前端解析器sparser负责对Caffe模型进行解析，并生成输入模型的中间表示(IR)，优化器基于IR进行优化，代码生成器将优化后的IR转换map为DPU指令。

2. DNNC用法

DNNC需要选项来控制对神经网络模型的编译处理。这些操作分为两类：1. 必须的 2. 选需。

Table11

1. 必须选项：

--prototxt Path of Caffe prototxt file

--caffemodel Path of caffemodel file

--output\_dir Path of output directory

--net\_name Name of neural network

--dpu DPU target (supported list: 1024FA, 1152FA, 4096FA)

--cpu\_arch CPU target (supported list: arm32, arm64)

2. 选需选项：

--help Show all options of DNNC

--version Show DNNC version

--save\_kernel Whether save kernel description in file or not

--abi DNNC生成的abi版本，0生成以前elf版本，1生成最新的abi版本。

--mode debug/nomal

--dump

3. 编译ResNet50

编译一个网络的时候，需要给DNNC必须的option选项。为了方便，在DNNDK包中已经包含了用于编译ResNet50

的脚本文件。他们有助于用户熟悉DNNC的选项使用。

编译成功后，在参数“output\_dir”指定的路径下，将生成 “elf文件”和“kernel部署信息”。

由于DPU支持的运算有限，当有不支持DPU运算的时，DNNC自动将目标神经网络分成不同的kernels。用户负责使用

由N2Cube提供的APIs，来解决数据传输和不同kernel间的通信，APIs可以基于kernel节点的输入输出获取输入输出地址。

DNNC生成的kernel描述信息包含两部分：

第一部分：描述了kernel的个数和拓扑：

\* Kernel number：根据DPU中支持的operator不同，不同网络生成的kernel数也不同。每个kernel将在第2部分详述。

\* Kernel topology: kernel拓扑图，描述编译完成时kernel graph视图中的内核。名为“kernel\_graph”被保存为扩展名为.gv的标准DOT文件，存放在当前DNNC工作文件夹下。下面的命令可以将DOT文件转换成JPEG文件。

$dot -Tjpg -o kernel\_graph.jpg kernel\_graph.gv

----------------------------------------------------------

例如：Inception-v1的JPEG图

如下所示，kernel图形节点包含kernel id和kernel type，箭头edge描述两个kernel间的

关系，第一个项item表示source kernel的输出张量tensor，第二个项item表示destination kernel的输入张量tensor。

tuple包含两部分：

绑定到tensor的输入/输出节点名 name；

输入/输出节点的tensor索引 index；

使用tuple提供的名字和索引，用户可以通过N2Cube提供的APIs获取输入/输出tensor address。

第二部分：详细描述每个kernel：

\* Kernel id: 当前kernel的id，每个kernel都有DNNC分配的唯一的id。

\* Kernel name: 当前kernel的名字，DPU支持的每个kernel都有一个相应的elf文件。前缀“dpu\_ "+KernelName.elf

\* Type：kernel的类型，DNNC支持3种kernel类型。（DPU kernel、CPU kernel、ParamKernel）

\* Nodes：所有当前kernel包含的node。对于DPU支持的kernel，“NA”用于代表所有nodes名字。

\* Input nodes：当前kernel的所有输入节点。对于DPU不支持的kernel，用户必须通过输出节点获得前一个kernel的

输出，并使用N2Cube提供的api，将这个输出作为当前节点的输入节点。

\* Output nodes：当前kernel的所有输出节点。输出节点的地址和大小可以用APIs提取出来。

----------------------------------------------------------

例如，dpu\_resnet50\_0.elf 和 dpu\_resnet50\_2.elf 对应的kernel名为 “resnet50\_0”和“resnet50\_2”。kernel

的名字在代码中用于识别kernel。

DPU支持的运算：

Convolution Support kernel-w and kernel-h values ranging from 1to 8 in any combination

ReLU No limitations

Pooling Only max-pooling is supported, and kernel size must be 2x2 or 3x3

Concat No limitations

Element-wise No limitations

InnerProduct No limitations

NOTE：根据FPGA资源的不同，支持的运算也会不同。

**DNNDK学习笔记(9)--使用DNNDK编程**

第7章

1. 编程模型

理解DNNDK的编程模型后，将更容易的在DPU平台上开发和部署深度学习网络。相应的概念包括：

“DPU Kernel、DPU Task、DPU Node、DPU Tensor. 其中前两个是DNNDK编程的核心概念。

2. DPU Kernel：

通过DNNC编译后，将神经网络model转换为等效DPU汇编文件，他会被DNNAS汇编为可执行文件。

DPU elf可执行文件被称为”DPU kernel“。

在调用 dupLoadKernel（）后，从runtime N2Cube角度看，DPU kernel将变成一个可执行unit。

N2Cube将会装载DPU kernel（包括DPU指令和网络参数）到特定的内存空间并分配硬件资源。

然后，通过调用 dpuCreateTask()来启用多线程编程，可以将每个DPU内核实例化为几个DPU任务。

3. DPU Task：

每个DPU Task是DPU kernel的一个运行实体。Task有自己的“私有内存空间”，所以多线程应用可用于同时

处理多个tasks，来提高效率和吞吐量。

4. DPU Node：

DPU Node，被认为是部署在DPU上的网络模型的基本元素。每个DPU节点都与输入、输出和一些参数相关联。

每个DPU节点都有唯一的名称，允许DNNDK导出的exported api访问它的信息。

有3种节点类型：

\* 边界输入节点：在DPU kernel拓扑中，它没有任何precursor的节点；通常是第一个node，有时一个kernel

有多个边界输入节点。

\* 边界输出节点：在DPU kernel中，它后面没有后继的节点。

\* 其他不适input 和 output boundary node 的节点，都可以叫internal nodes。

编译之后，DNNC将会给出关于kernel的信息和kernel 的边界输入输出节点信息。下面是编译Iception-v1的示例:

“conv1\_7x7\_s2”是DPU kernel 0的边界输入节点，“inception\_5b\_output” 是DPU kernel 0的边界输出节点。

使用dpuGetInputTensor\* / dpuSetInputTensor\*时，需要参数“nodeName”来指定边界输入节点。如果参数

”nodeName“不是一个有效的边界输入节点，DNNDK会给出报错信息。

同样的，使用dpuGetOutputTensor\* / dpuSetOutputTensor\*时，参数如果不是正确的边界输出节点，也会报错。

5. DPU Tensor：

DPU Tensor是一组多维数据集，用于存储运行信息。Tensor的属性信息(长、宽、通道...)可以通过DNNDK的APIs获取。

6. 编程接口

DNNDK提供了一组封装在几个库中的轻量级C/ C++ api，以平滑DPU的深度学习程序开发。详见12章。在进行DPU

开发时，经常会进行CPU与GPU的数据交互。如，例如，CPU预处理的数据可以发送到DPU进行加速，DPU的输出可能

需要复制回CPU进一步处理。DNNDK提供了一组APIs，使这些工作变得简单。一些示例：

APIs to set input tensor for a computation layer or node: 为层or节点设置输入数据：

- dpuSetInputTensor()

- dpuSetInputTensorInCHWInt8()

- dpuSetInputTensorInCHWFP32()

- dpuSetInputTensorInHWCInt8()

- dpuSetInputTensorInHWCFP32()

APIs to get output tensor from a n a computation layer or node: 获取计算or节点的输出：

- dpuGetOutputTensor()

- dpuGetOutputTensorInCHWInt8()

- dpuGetOutputTensorInCHWFP32()

- dpuGetOutputTensorInHWCInt8()

- dpuGetOutputTensorInHWCFP32()