摘要：深度神经网络在计算机视觉、语音识别等方面有广泛应用，然而深度神经网络出色的准确率是基于很强的计算能力并且对硬件资源占用极大，所以它很难嵌入智能手机、智能手表和机器人等移动设备。所以在不降低准确率和不增加硬件开销的情况下，降低功耗和加快处理速度是急需的。

本文旨在分析比较目前的加速技术，包括dataflow和量化。

关键词：神经网络、加速、低功耗、dataflow、量化

Abstract：

Keywords:

//设计说明：

//任务来源：

引言：深度神经网络在AI的许多领域有着广泛的应用，自从深度神经网络在语音和图像识别方面取得重大突破后，深度神经网络的应用呈现爆发式增长，不管是自动半自动驾驶、癌症细胞检测还是消费电子，深度神经网络都能胜任，甚至在某些领域例如，已经超过了人的准确率。它的极佳的性能利益于对数据特征的高层次抽象的能力，数据量越大网络越深，特征抽象得越高层越准确。近期神经网络的发展利益于大数据的计算机算力的飞速提升。

然而，现阶段的深度神经网络的算力是基于高的计算复杂度，大量的计算能耗和大量占用内存资源。然而，目前，嵌入式视觉系统广泛应用于自动半自动驾驶、消费电子（相机和移动电话）、电动玩具，它们硬件资源少，电池容量小且要求准确率高和实时性，不能直接运行深度学习网络，需要一个低功耗、轻量化、低成本的系统。，功耗主要由计算功耗、时钟功耗、内存访问功耗组成。深度神经网络的参数动辄上万计，计算量更是高于两个数量级，对DRAM访问，比Buffer访问，PE和Regfile访问消耗的能量高两个数量级，如何减小对存储的数据的访问次数，尤其是减少对Dram访问，增加对数据的复用，不管是在RF内部，PE之间，还是从BUFFER获取得将极大降低功耗。Dataflow是本文分析了目前主流的4种dataflow：OS WS NLR RS,它们结构因都是为了降低功耗减少内存占用而有诸多共同点，对数据的存储交换和处理方式而差别很大。从对数据的访问量和不同存储位置功耗差别来比较4种结构的功耗，并与实验数据相比较。

功耗和另一个来源是计算，我们知道乘法比加法消耗的功耗是数量级的，尤其是训练神经网络时用的32位浮点运算，乘法比加法多，时间多，如果我们用8位量化的话，在可接受精度下降范围内，硬件支持更加友好，速度更快，功耗更低。本文将在谷歌的tensorflow框架下，对训练过的lenet网络进行量化，准确率只下降0.6%,而模型参数变为1/4。更容易移植到小型移动设备上，比如做成app在手机上运行。

国内外相关研究：【】提出了一种基于行固定的dataflow，能减少数据交换的能耗，通过探索对权值输入数据的复用并且减少部分和的数据迁移。

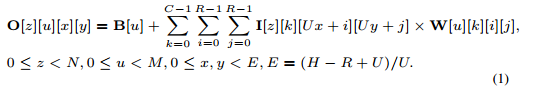
深度神经网络理论基础

深度神经网络的结构针对同的应用场景差别很大，比如： ，而且针对同一种应用为了提高准确率和效率结构也在不断变化，比如从ALEXNET到，都是对，，由于，，它的输入是要被分析的数据，代表可以是图像的像素，音频的幅值，。。，本文主要讨论前向网络，所有计算都是对前一层网络的操作，最后一层的操作产生网络的输出。

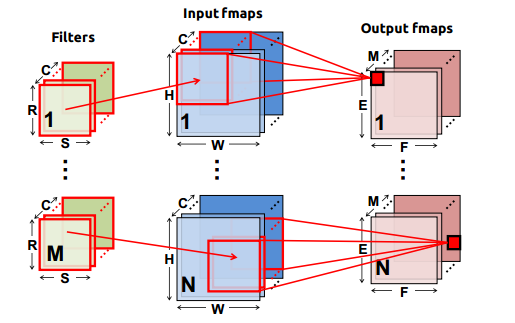
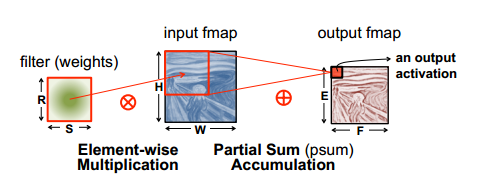
深度神经网络的一种结构是全连接，如图，从图中可以看出每一个输出都通过“线”连接到所有的输入，而这“线”就是权重，也就是说输出是所有输入的加权和，权值的数量是输入输出的积，需要大量的计算和占用大量的存储资源，如果能够减少计算一个输出所需要的权值的数量，并且让每个输出都用相同的权值，那么就将权值的数量极大减少，权值的共享和复用做到了极致。卷积就是这样一种操作。

DNN以CNN为基础，加入池化层（Pooling layer），内积层（Inner product）、激活层（Activation layer）、标准化层（Normalization layer）、

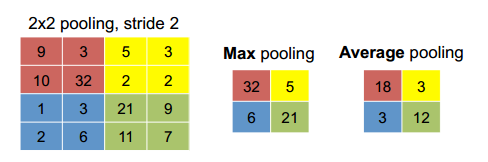
卷积层：如图所示，通过将一个权值滤波器在输入上面滑动并与相应位置上的输入值相乘，所有的积再相加得到这个滤波器当前所在中心位置的输出，二维权值滤波器以一定的步长滑过输入特征图产生的所有输入值按位置排列形成的阵列就是二维输出特征图，二维权值滤波器一般为1 x 1 、3 x 3、7 x 7等奇数，所以输出值的位置一般在滤波器的中心。高维卷积的输入是多个二维特征图堆叠而成的，每一个特征图称为一个通道，每一个通道都和相应的高维滤波器的一层卷积，再把所有通道的卷积值相加，得到输出特征图的一个通道，多个高维滤波器通过相同的运算能产生相等数量的输出通道，再把这些通道堆叠起来生成输出特征图。



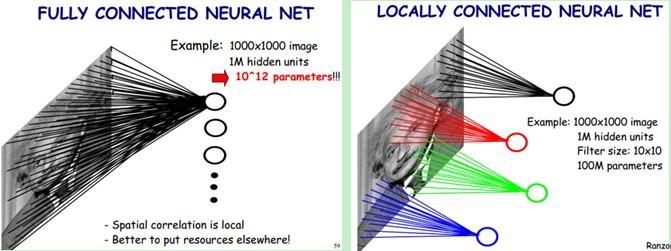
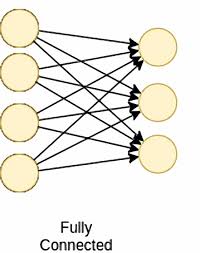
O I W B 是输出特征图矩阵、输入特征图矩阵、滤波器矩阵、偏差矩阵。

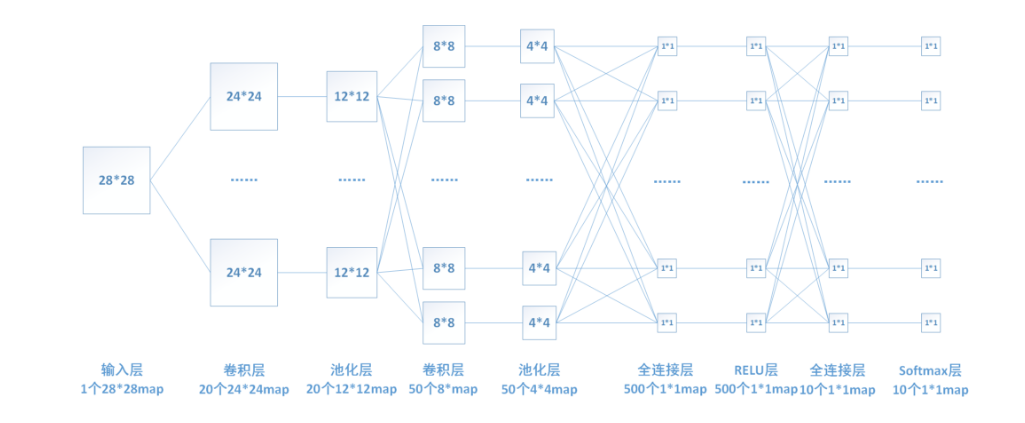


池化层：池化层让网络更加稳定，不受微扰的影响，还能缩小网络的大小。如图，在输入特征图的每一个通道，用一定大小的方框，（比如2x2），以一定的规则（比如，选取方框中的平均值、最大值、最小值）得到一个输出值替代方框中的所有输入值，类似于卷积，当方框滑过整个输入通道时，得到输出通道，所有输出通道堆叠形成输出特征图。

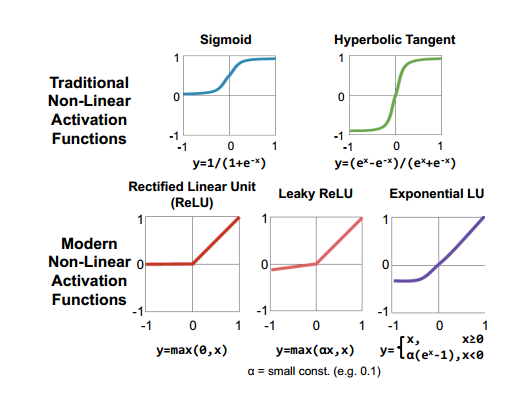


内积层：在Caffe中全连接层（Fully Connected ）又被称为内积层（Inner product）,顾名思义，内积就是做输入向量和权重矩阵的内积，本质是由一个特征空间线性变换到另一特征空间，输出层的每一个元素都会受到所有输入元素的影响。在DNN中，内积常常出现在最后几层，用于对前面设计的特征做加权和。





激活层：在卷积和内积计算之后往往有非线性激活函数，如果没有非线性激活函数或激活函数是线性的，那么多个内积层可以用一个层来代替，失去了深度学习神经网络在更高层次提取特征的意义。非线性化函数有传统的Sigmoid、Tanh和ReLU，其中ReLU由于其实现方式简单能够快速训练网络，在DNN中有广泛运用，为了提高准确率，ReLU也有许多变种，比如Leaky ReLU、Parametric ReLU和Exponential LU。

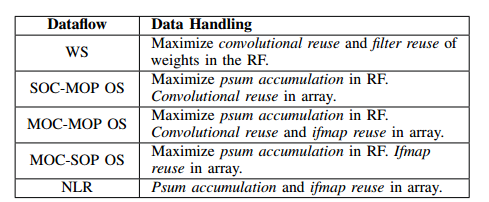


标准化：为了加速训练速度和提高准确率，必须控制输入值的分布。标准化先计算出滑动窗口中元素的平方和，再将平方和非线性化，然后乘以输入元素得到相应的输出

流行的深度神经网络

图说。

* 对于深度神经网络，处理的瓶颈是内存访问，。每一个乘加操作需要三次内存读取，分别是权值、输入数据和部分和，一次写内存用于更新部分和或者输出。最坏的情况是所有的内存访问都对片外DRAM访问，将严重降低速度和增大功耗，片外DRAM的访问的功耗是计算的几个数量级（释）。图。显示了抓取数据从DRAM比RF或者邻近的PE高2个数量级，所以为了减少数据迁移的能量消耗，必然要使得数据尽可能多的从RF或者PE获取，从DRAM获取的数据尽可能地在内部复用，于是提出了图，一种空间结构，加速器要设计一种数据流能决定何时从哪个内存层次取数据，最优的数据流是最少访问能耗高的内存层次比如DRAM，DRAM虽然能够存储GB级别的数据，但是访问的功耗比只有KB片上的存储高两个数量级，所以，每次从高能耗层次取的数据尽可能多的在低能耗层次复用，但是低能耗层次的容量是十分有限的，因此我们需要在功耗和容量的限制下探索不同的数据流结构对数据的复用。加速器这样的空间敏感架构，我们讨论数据流如何能低成本地从存储器中被调用，以减少能耗。

(中文)

几乎所有的神经网络加速芯片都有如下共性的模块:

一个计算单元，主要完成深度学习中矩阵乘/卷积操作；这个计算单元的实现大致分为两种: 1. 以TPU为代表的脉动阵列(systoic array); 2. 点乘器(dot-product)。

片上存储单元，存储每一层的输入filter map/输出filter map及权值(weights)：目前的设计中，输入 filter map和输出 filter map一般都是共享一块片上存储资源；权值(weights)可以很大，因此有时候不能全部存储在片上；

其实神经网络加速芯片这两个共性的模块一点也不难理解，神经网络加速芯片不就是为了完成每个网络层的计算嘛，因此需要一个高吞吐量的计算单元；为了能喂饱这个计算单元，就必须想办法高效的给其喂数据，因此就有了片上存储单元

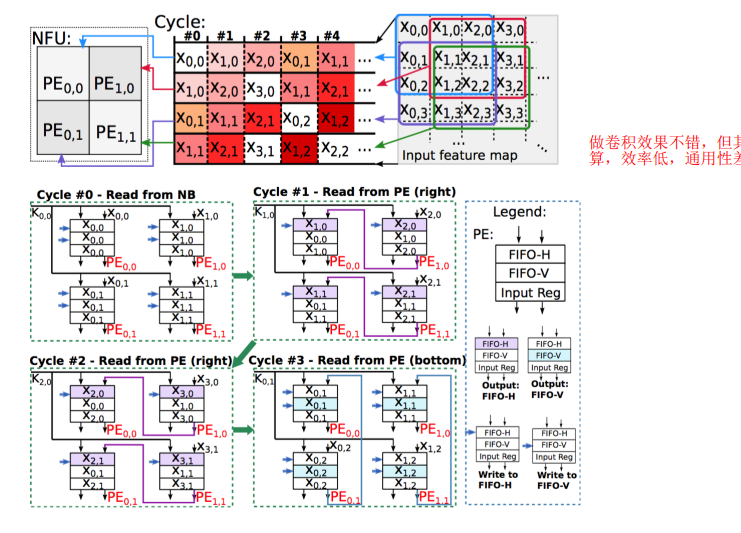
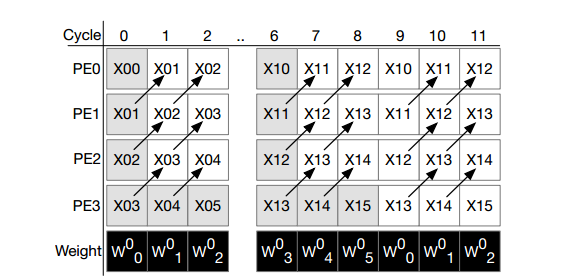
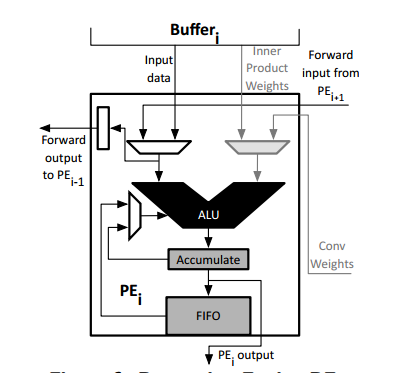
已经存在许多实现对DNN加速的数据流结构，根据对数据的处理方式为列出下列三类：

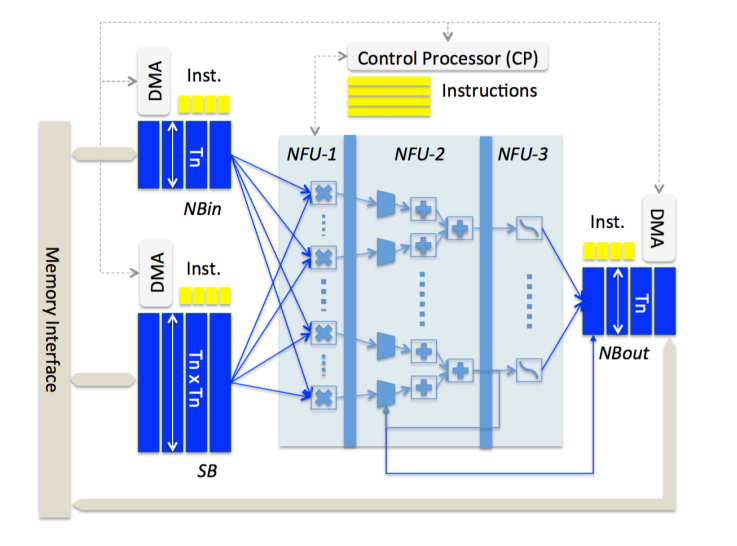
有三种数据流：RS不比较。

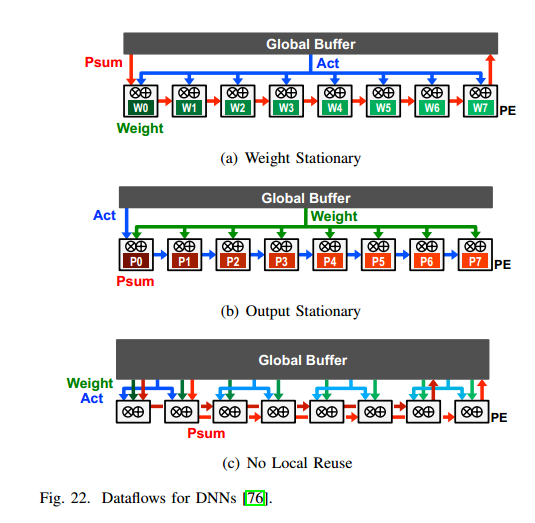
1. 权重固定（WS），顾名思义，权重固定dataflow是将权值固定在RF里面，最大化权值的复用，使得读取权值的功耗最小。Kxk个权值被分配到kxk个PE，并且固定下来，kxk个输入的数据被依次分配给相应kxk个PE，与权值相乘，通过PE阵列将积累加，PE之间传输的前面所有PE的部分积的累加（Psum）。权值一旦从DRAM被读到RF里面，就会被多个MAC复用。由于权值占据了RF，因此输入数据被分配给了所有的PE，部分和通过PE阵列累加。（像DNNweaver的图？代码算法）。内存访问量公式？

最后一个PE输出的部分和存入全局缓冲器，如果加完了所有输入通道的部分和，则可以直接写入DRAM或者作为下一层的输入，否则作为部分和进入下一下通道的部分和累加。如图：

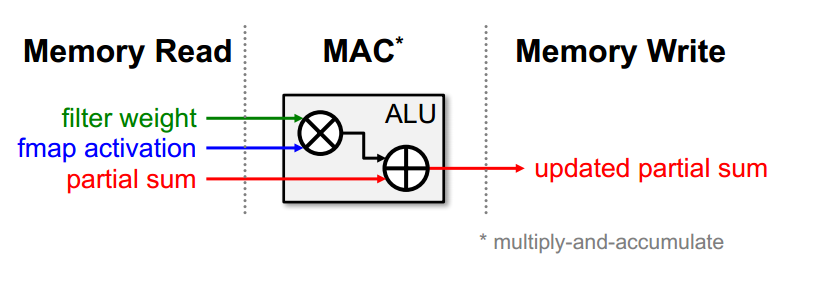
1. 输出固定（OS），（用DNNWeaver例子来说明），输入固定dataflow通过将部分和存入同一个RF，在此RF内部累加来最小化读写部分和的功耗。输入数据在PE阵列间传递，而权值被分配给所有的PE阵列，内部简单结构如图（DNNW）,代码：



1. 没有本地复用(NLR)。分布在PE中的RF虽然能减少功耗，却占用了很多的面积，成本是面积的？关系，为了最大化存储容量，最小化片外存储带宽，PE里面没有本地存储RF，所有的RF被归结于全局Buffer，增加了容量。但是极大增加了PE之间，PE和全局Buffer之间的数据交换。没有在RF级的复用，PE间的数据交换实现了输入数据复用和部分和累加。NLR把PE阵列分组，组内的PE读取相同的输入数据但是权值不同。不同组的PE从不同的通道读取输入数据和权值。部分和在不同组之间累加。

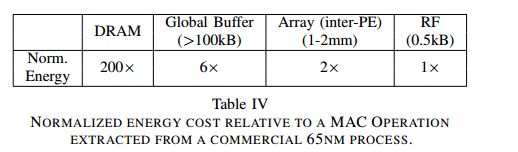


MAC操作的能耗由两个因素决定：输入数据复用和部分和在哪里累加；MAC的输入是从哪个存储层次获取的。



低功耗dataflow的目标是让更多的数据的在低能耗的路径上交换。基于数据的路径和每级存储的能耗，提出一种计算总体存储访问量及访问功耗的模型。加速器有四个存储层次，按照能耗从高到低排序为，DRAM,全局缓冲器，阵列（PE之间），RF。ALU从越高层次获取数据，消耗的能耗越高。在任意两个存储层次之间移动数据的能耗是这两个层次到ALU能耗的差额：如下公式

表：



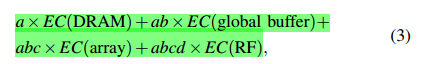
展示了每个存储层次到ALU的能耗。

分析方法：

正如图：MAC的数据分为两个部分：1.输入的数据，数据是单向流动的，取了数据，用完即丢，不会再存入存储层次里面。2.部分和，表现为循环流动，通过与新值累加不断更新自己，在多个存储层次读写。能耗由两个方面决定：1.每个存储层次有多少次访问，2.存储层次一次访问的能耗，如表所示。有如下公式：总能耗=输入数据能耗+部分和能耗

输入数据能耗= 访问量X 能耗/次

1）输入数据访问能耗：根据表，要使输入数据的能耗最低，理想情况下是，从DRAM读一次到RF，然后每次都从RF读取数据。然而，由于有限的RF存储资源，不能把所有的输入数据都存入RF, 一旦被踢出RF，就需要从更高能耗的存储层次再次获取数据。据此，数据复用分为四个层次，复用数定义为在一个数据的生命期内，它从高能耗层次被读到低能耗层次的次数。如表：DRAM ->Buffer：

假设一个数据总的复用次数是axbxcxd，其中a为，，，，b为，，，,举个例子，如图，复用总数34，分为a=1， 能耗估计公式为

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | DRAM<->Buffer | Buffer<->PE | PE<->RF | RF<->ALU |
|  | a | b | c | d |
|  | DB | BP | PR | RA |

2）部分和累加能耗：部分和会穿过4级存储层次，在ALU之间累加，理想情况下，每个部分和存储在RF里面用来累加，然而，这常常是不可行的，由于全局操作执行的需要，部分和必须被存入更高层次，然后读回来，因此，累加总数axbxcxd能被分成4个层次。每个层次的累加数定义为每个数据在它的生命期内，写入读出的次数。例如图9，累加总数为36，a= ,能耗：



1. 获取参数：对于每种dataflow，都有一组a,b,c,d对于三种数据类型，输入数据，权重，部分和，通过方程3.4.在给定CNN结构下，有最优的abcd最低的能耗，约束条件是硬件资源，包括全局buffer，RF，PE阵列。

注意点：虽然我们假设在不同dataflow同一个存储层次的访问能耗相同，但是实际上会因dataflow的具体实现方式有差异，比如大的全局buffer有更大的访问能耗。在阵列层次，与相邻的PE通信比更远的PE有更低的能耗，因为有小的线电容更简单的NoC设计。RF与buffer和PE类似。

卷积层：WS:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | a | b | c | d |
| Weight | Nink^2nout | 1 | Nin\*r\*k\*nout | Nin\*r\*r\*k\*nout | 1 |
| Data | Nin\*r^2\*nout | 1 | 1 | K^2 | 1 |
| Psum | Nin\*r^2\*nout | 1 | 1 | 1 | K^2-1 |

OS:

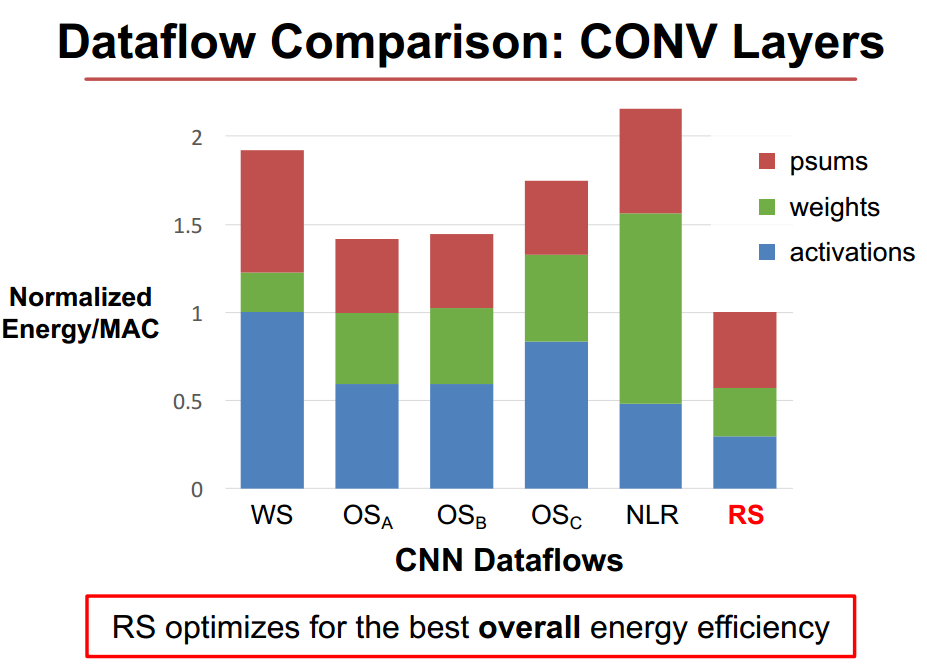
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | a | b | c | d |
| Weight | Nink^2nout | 1 | 1 | Ne/k^2 | 1 |
| Data | Nin\*r^2\*nout | 1 | 1 | K^2 | 1 |
| Psum | Nin\*r^2\*nout | 1 | 1 | K^2-1 | 1 |

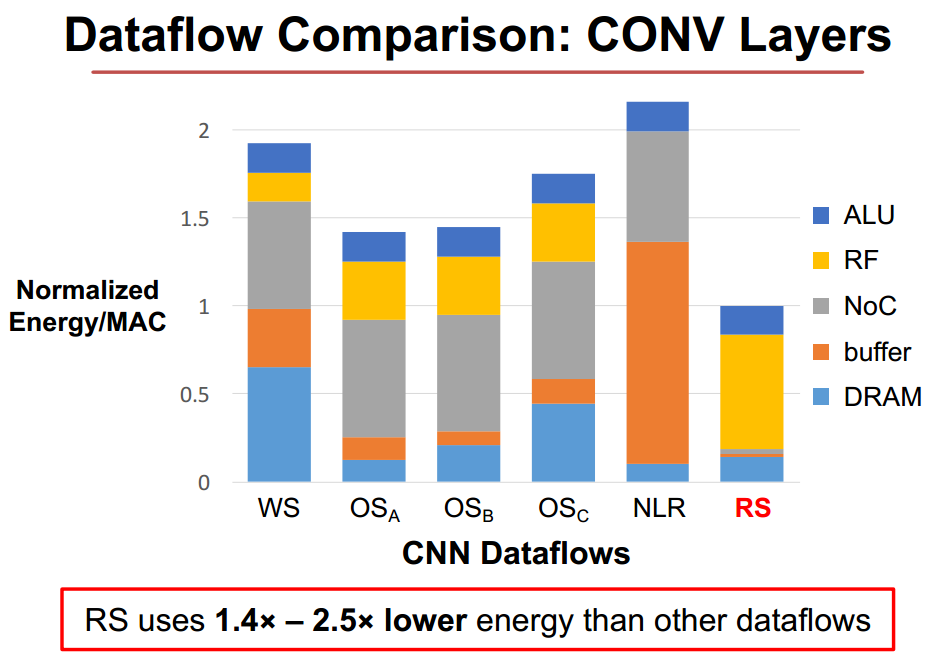
NLR:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | a | b | c | d |
| Weight | Nink^2nout | 1 | R^2 | 1 | 1 |
| Data | Nin\*r^2\*nout | 1 | K^2 | p | 1 |
| Psum | Nin\*r^2\*nout | 1 | 1 | g | 1 |

G: num\_group of Pes p:num of pes of each group

行比较：





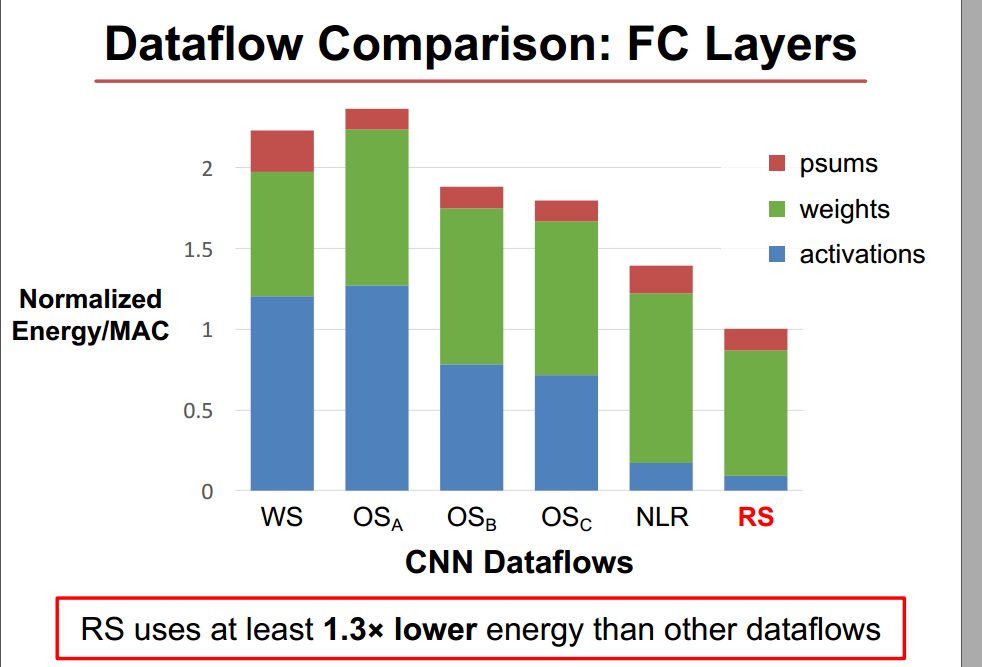
列比较：ALU相似，因为对于ALEXNET，虽然数据流向不同，但计算过程相同。

RF：由表中WS的RF为： NLR没有RF

NOC:OS略大于WS和NLR

BUFFER: NLR由于没有RF，所有的数据都由Buffer给，因此开销巨大

FC：



线性量化

一般在设计深度神经网络的时候，追求更高的准确率，不重视实现的复杂度，但是实现起来就很难，随着神经网络算法在图像、语音等领域都大幅度超越传统算法，但在应用到实际项目中却面临两个问题：计算量巨大及模型体积过大，不利于移动端和嵌入式的场景；模型内存占用过大，导致功耗和电量消耗过高。因此，如何对神经网络模型进行优化，在尽可能不损失精度的情况下，减小模型的体积，并且计算量也降低，就是我们将深度学习在更广泛的场景下应用时要解决的问题。在移动端或者嵌入式设备上应用深度学习，有两种方式：一是将模型运行在云端服务器上，向服务器发送请求，接收服务器响应；二是在本地运行模型。

一般来说，采用后者的方式，也就是在PC上训练好一个模型，然后将其放在移动端上进行预测。使用本地运行模型原因在于，首先，向服务端请求数据的方式可行性差。移动端的资源（如网络、CPU、内存资源）是很稀缺的。例如，在网络连接不良或者丢失的情况下，向服务端发送连续数据的代价就变得非常高昂。其次，运行在本地的实时性更好。但问题是，一个模型大小动辄几百兆，且不说把它安装到移动端需要多少网络资源，就是每次预测时需要的内存资源也是很多的。

那么，要在性能相对较弱的移动/嵌入式设备（如没有加速器的ARM CPU）上高效运行一个CNN，应该怎么做呢？这就衍生出了很多加速计算的方向，其中重要的两个方向是对内存空间和速度的优化。采用的方式一是精简模型，既可以节省内存空间，也可以加快计算速度；二是加快框架的执行速度，影响框架执行速度主要有两方面的因素，即模型的复杂度和每一步的计算速度。

精简模型主要是使用更低的权重精度，如量化（quantization）或权重剪枝（weight pruning）。

量化（Quantization）又称定点，用更少的数据位宽进行神经网络存储和计算。它的优势在于节省存储，并进行更快地访存和计算。

量化是一个总括术语，用比32位浮点数更少的空间来存储和运行模型，并且TensorFlow量化的实现屏蔽了存储和运行细节。

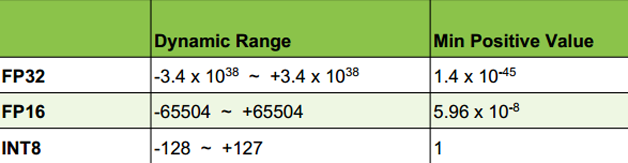
神经网络训练时要求速度和准确率，训练通常在GPU上进行，所以使用浮点数影响不大。但是在预测阶段，使用浮点数会影响速度。量化可以在加快速度的同时，保持较高的精度。

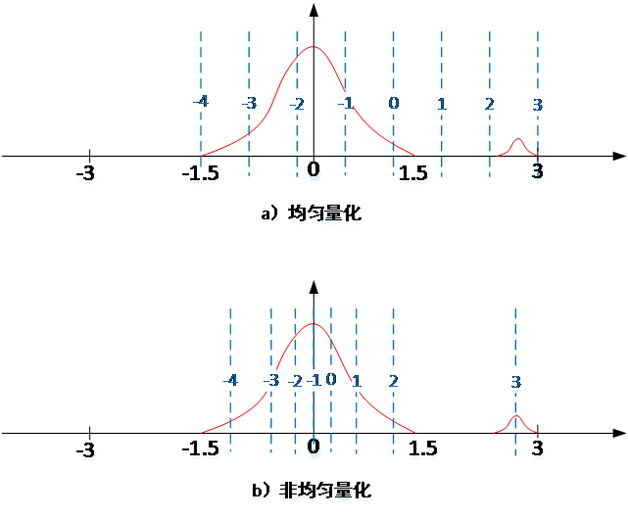
量化网络的动机主要有两个。最初的动机是减小模型文件的大小。模型文件往往占据很大的磁盘空间，例如，上一节介绍的网络模型，很多模型都接近200MB，模型中存储的是分布在大量层中的权值。在存储模型的时候用8位整数，模型大小可以缩小为原来32位的25%左右。在加载模型后运算时转换回32位浮点数，这样已有的浮点计算代码无需改动即可正常运行。

量化的另一个动机是降低预测过程需要的计算资源。这在嵌入式和移动端非常有意义，能够更快地运行模型，功耗更低。从体系架构的角度来说，8位的访问次数要比32位多，在读取8位整数时只需要32位浮点数的1/4的内存带宽，例如，在32位内存带宽的情况下，8位整数可以一次访问4个，32位浮点数只能1次访问1个。而且使用SIMD指令，可以在一个时钟周期里实现更多的计算。另一方面，8位对嵌入式设备的利用更充分，因为很多嵌入式芯片都是8位、16位的，如单片机、数字信号处理器（DSP芯片），8位可以充分利用这些。

此外，神经网络对于噪声的健壮性很强，因为量化会带来精度损失（这种损失可以认为是一种噪声），并不会危害到整体结果的准确度。

那能否用低精度格式来直接训练呢？答案是，大多数情况下是不能的。因为在训练时，尽管前向传播能够顺利进行，但往往反向传播中需要计算梯度。例如，梯度是0.2，使用浮点数可以很好地表示，而整数就不能很好地表示，这会导致梯度消失。因此需要使用高于8位的值来计算梯度。因此，正如在本文一开始介绍的那样，在移动端训练模型的思路往往是，在PC上正常训练好浮点数模型，然后直接将模型转换成8位，移动端是使用8位的模型来执行预测的过程。



最简单的量化方式是线性量化，

tensorflow的量化是通过将预测的操作转换成等价的8位版本的操作来实现。量化操作过程如图7所示

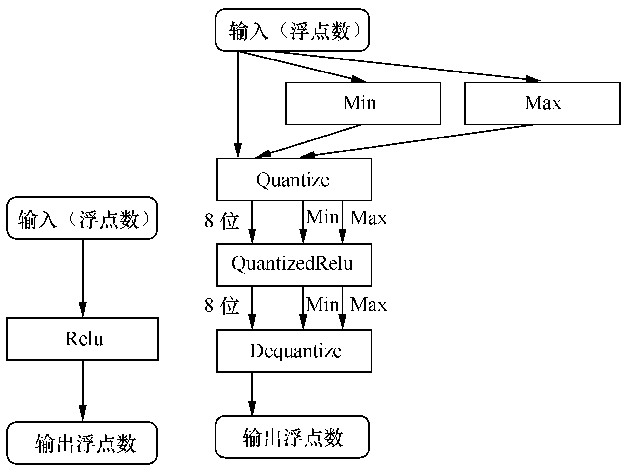


图7中左侧是原始的Relu操作，输入和输出均是浮点数。右侧是量化后的Relu操作，先根据输入的浮点数计算最大值和最小值，然后进入量化（Quantize）操作将输入数据转换成8位。一般来讲，在进入量化的Relu（QuantizedRelu）处理后，为了保证输出层的输入数据的准确性，还需要进行反量化（Dequantize）的操作，将权重再转回32位精度，来保证预测的准确性。也就是整个模型的前向传播采用8位段数运行，在最后一层之前加上一个反量化层，把8位转回32位作为输出层的输入。

在这个项目中，我使用了 TensorFlow 中的量化工具来进行模型压缩。目前我只使用权重量化来减小模型大小

参考文献：