vDNN:用于可扩展、内存高效的神经网络设计的虚拟化深度神经网络

Minsoo RhuNatalia GimelsheinJason ClemonsArslan ZulfiqarStephen W. Keckler
英伟达
Santa Clara, CA 95050
{mrhu, ngimelshein, jclemons, azulfiqar, skeckler}@nvidia.com

摘要--
最广泛使用的机器学习框架要求用户仔细调整他们的内存使用，以便将深度神经网络（DNN）纳入GPU的DRAM容量。这一限制阻碍了研究人员研究不同机器学习算法的灵活性，迫使他们要么使用不太理想的网络架构，要么将处理过程并行化。跨越多个GPU。我们提出了运行时一个内存管理器虚拟化DNN的内存使用，从而使GPU和CPU内存可以同时用于训练更大的DNN。我们的虚拟化DNN（vDNN）将AlexNet的平均GPU内存使用量减少了89%，OverFeat减少了91%，GoogLeNet减少了95%，大大降低了DNN的内存需求。在VGG-16上进行了类似的实验，其中包括vDNN使批处理量为256的VGG-16（需要28GB的内存）能够在含有12GB内存的单个NVIDIA Titan X GPU卡上进行训练，而与拥有足够内存的假设的高级GPU相比，性能损失为18%。

I. 简介

深度神经网络（DNNs）最近已经成功地全面部署在各种应用领域，如计算机视觉[1]、语音识别[2]和自然语言处理[3]，这要归功于它们与传统的最先进的方法相比的卓越性能。这种深度学习技术的扩散导致近年来开发了一些软件框架来分析和促进神经网络的设计[4, 5, 6, 7]。可用的框架列表继续扩大，开发人员不断增加更多的功能，提高计算效率，以促进深度学习领域的研究。由于图形处理单元（GPU）提供了巨大的计算能力，这些框架为GPU软件库（如cuDNN）提供了强大的后台支持[8]。事实上，今天几乎所有参与训练神经网络的团体都在部署GPU来加速深度学习[9]。虽然这些流行的机器学习（ML）框架有助于DNN的研究，但使用这些框架的一个主要限制是，系统中GPU的DRAM容量限制最终限制了可以训练的DNN的大小（第II-C节）。为了绕过内存容量瓶颈[10, 11]，ML从业者必须使用不太理想的DNN架构（例如，较少的层数、较小的批处理量、性能较差但内存效率较高的卷积算法），或者将DNN并行化。

978-1-5090-3508-3/16/$31.00 ©2016年IEEE

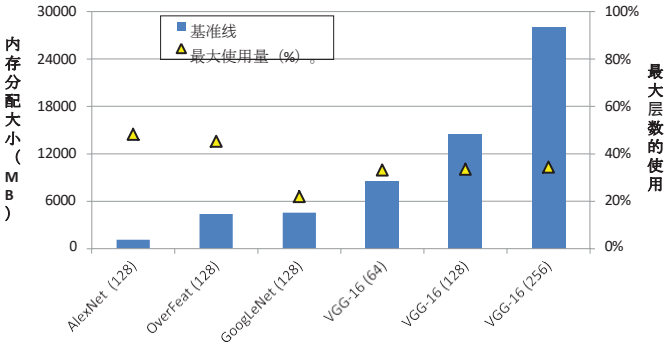

图1：使用基线、全网分配策略时的GPU内存用量（左轴）。右轴显示了在逐层遍历网络时实际利用的该基线分配的最大部分。在本文中，每个网络名称旁边的数字指的是批次大小。所研究的DNN详见第四节C。

跨越多个GPU[12]。图1强调了ImageNet[13]获奖DNN的内存消耗趋势是如何随着时间的推移而演变的。例如，AlexNet[1]只包含5个卷积层和2个全连接层，训练时需要"仅仅 "1.1GB的内存分配，这远远低于最先进的NVIDIA Titan X的12GB内存容量。另一方面，最近的VGG-16[14]包含16个卷积层和3个全连接层，在批次大小为256时，总共产生28GB的内存使用。由于单个GPU只能满足VGG-16的批量大小为64，所以批量256的训练需要在多个GPU上进行并行化，或者必须以较小的批量顺序多次执行网络。随着最近的ImageNet获胜网络采用了超过一百个卷积层[15]，深度学习的趋势是走向更大更深的网络设计[14, 16, 17, 18]。因此，缓解GPU僵化的物理内存限制变得越来越重要。

在本文中，我们提出了*虚拟化的深度神经网络*（vDNN），这是一个运行时内存管理解决方案，它虚拟了深度神经网络在GPU和CPU内存中的使用。我们的vDNN允许ML从业者部署更大和更深的网络，超过现有GPU的物理容量，使他们能够更专注于他们的算法，而系统结构和运行---。

时间系统透明地管理其数据的分配、放置、移动和释放。vDNN背后的动机是基于以下三个关键观察。

1) 通过随机梯度白化（SGD）训练的DNN是

2）这些神经网络的训练涉及一系列的*分层*计算，其顺序是静态固定的，并在整个训练过程中重复数百万至数十亿次的迭代。3）尽管GPU在任何时候都只能处理单一层的计算（由于基于SGD的DNN训练的逐层计算特性），但流行的ML框架采用了*全网*内存分配政策，因为DNN训练需要在GPU内存中备份网络中所有层的中间特征图，以便进行梯度更新（第二节C）。换句话说，现有的内存管理方案*过度配置*内存分配，以适应*整个*网络层的使用，即使GPU只使用该分配的一个子集来满足层的要求。我们观察到，这种内存利用不足的问题对于更深的网络来说变得更加严重，导致53%到79%的分配内存在任何时候都没有被使用（图1）。vDNN的目标是*保守地*将GPU内存分配给特定层计算的*即时*使用，这样最大和平均的内存使用量就会大大减少，允许重新搜索人员训练更大的网络。为了实现这一目标，vDNN利用分配的数据结构的数据依赖性，特别是占大部分内存使用的中间特征图（第二节C），并在GPU和CPU内存之间释放或移动这些中间数据。具体来说，vDNN要么1）积极地从GPU内存中*释放*这些特征图，如果不存在进一步的重复使用；要么2）如果确实存在进一步的重复使用，但不是立即需要的，则将其*载入*（和后来的*预取*）到CPU内存。通过利用DNN的层间内存访问和重用模式，我们的vDNN内存管理器智能地将正常的DNN计算与ffload/refetch/release操作重叠，有效地将DNN的内存使用虚拟化，几乎没有性能损失。vDNN的操作对程序员来说是完全透明的，使他们能够训练更大更深的神经网络，其内存消耗远远超过目前GPU的物理内存的限制。我们工作的主要贡献是。

- 这项工作是首次对基于GPU的DNN*训练进行*详细的定量分析，而不是针对DNN*推理的*节能加速器的再厘米文献[20, 21, 22, 23, 24, 25, 26, 27, 28, 29]。
- 据我们所知，我们的工作是第一个从架构角度对DNN的内存访问特性及其对GPU内存系统的影响进行深入的特征研究。
- 这项工作确定了当前ML框架的内存管理政策的主要局限性，因为它们重新要求目标DNN的全网内存使用量在GPU的物理容量内单片化。我们表明，当现有框架的内存分配大小（14GB至67GB）超过GPU内存预算（英伟达Titan X的12GB）时，10个研究的DNN中有6个训练失败。
- 我们提出、实施并评估了一个名为vDNN的运行时内存管理器，它将神经网络的内存使用在CPU和GPU内存之间进行虚拟化。我们的vDNN解决方案将这6个内存饥渴的网络的平均GPU内存使用量减少了73%到98%，使它们可以在一块Titan X卡上进行训练。与一个假设的、包含足够的内存来容纳整个DNN的GPU相比，vDNN产生了1%到18%的性能开销。

II. 背景和动机

本节概述了现代DNN、当前ML框架的内存管理策略，以及它们的主要局限性，这也是本项工作的动力。

A. DNN架构

卷积神经网络是用于高精度计算机视觉任务的最流行的ML算法之一。虽然其他类型的网络也在不断发展（例如，用于自然语言处理的递归神经网络），但所有这些DNN都是通过*随机梯度白化*（SGD）使用*反向传播*算法[19]进行训练的。为了清楚地说明问题，并且由于它们在ImageNet竞赛中的先进表现，本文主要关注AlexNet[1]、OverFeat[30]、GoogLeNet[17]和VGG[14]中常见的前馈式卷积神经网络。然而，我们工作的关键直觉同样适用于*任何*表现出*层级*计算特征并通过SGD训练的神经网络，本节后面将详细介绍。

DNN的设计采用了多种类型的层的组合，大致分为卷积层（CONV）、激活层（ACTV）、池化层（POOL）和全连接层（FC）。一个神经网络的结构是这些层的多个实例的序列。尤其是用于计算机视觉任务的DNN，其结构大致分为以下两个模块。1）*特征提取层*，检测输入图像的可区分特征；2）*分类层*，分析提取的特征并将图像分类到一个给定的图像类别。特征提取层一般采用CONV/ACTV/POOL层设计，被定位为DNN的初始部分。分类层是使用FC层建立起来的，并在DNN计算序列的最后发现。深度学习的一般趋势是用大量的特征提取层来设计网络，以便为稳健的图像分类训练一个深层次的特征结构[14, 15, 17]。

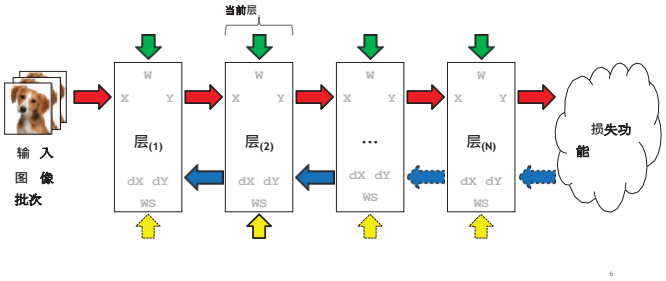

图2：使用基线内存管理器的线性网络所需的内存分配（粗体箭头）。对于推理，所有绿色（W）和红色（X）箭头的总和被分配。对于训练，需要两个额外的数据结构dX和dY：这两个结构的大小是所有蓝色（dY）箭头的最大值，并且在向后传播期间回溯各层时被重复使用。在某些卷积算法中需要一个可选的临时缓冲区，在cuDNN[8]中称为*工作区*（黄色箭头，WS）。工作区缓冲区的大小与所有层中的最大工作区要求一致，并在向后传播过程中被重复使用。

B. DNN训练与推理

一个神经网络在被部署到*推理*或*分类*任务之前需要被*训练*。训练需要通过执行前向和后向传播算法的操作来学习和更新神经网络各层的*权重*[19]。前向传播和后向传播的遍历方向以及必须执行的数学运算都有所不同。

正向传播。前向传播是从第一层（输入）到最后一层（输出）进行的，而后向传播的方向正好相反（最后一层到第一层），在图2中从右到左。直观地说，前向传播*逐层*遍历网络，在给定的输入上完成上述特征提取和分类任务，从而实现图像分类。在前向传播过程中，每一层都应用了一个数学模型，并在此基础上进行分类。对其输入特征图（X）进行操作，并将结果存储为输出特征图（Y）。对于线性前馈的DNN来说。层$_{(n-1)}$的结果Y直接被层$_{(n)}$作为输入X（图2）。因此，前向传播的计算流程是一个*序列化的*过程，因为层$_{(n)}$，只有在前一个层$_{(n-1)}$，完成其计算并将其输出Y转发到层$_{(n)}$，输入X。非线性网络拓扑结构可以包含一对多(fork)和多对一(join)的相互关系。层的依赖性，但前向传播仍然涉及一个一系列的逐层计算，详见图3。注由于这种层间数据的依赖性，GPU在任何时候都只能处理单一层的计算。因此，每层所需的最小内存分配是由该层的输入-输出关系和它的数学函数[1]。例如，一个CONV层使用

[1]流行的激活函数（sigmoid/tanh/ReLU[1]）可以使用元素明智的计算方式重构为*就地的*算法。Caffe和Torch都利用了这种就地内存优化，只为Y和dY分配内存空间，用于前向（Y）和后向（Y和dY）传播[31]。本文对基线和vDNN都采用了这种就地优化，以进行保守的评估。

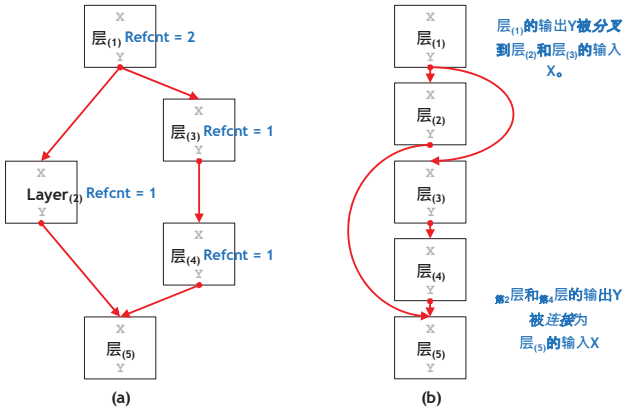

图3：（a）GoogLeNet风格的非线性前馈网络在前向传播过程中的计算图及其层间依赖关系。Refcnt指的是取决于当前生产者层Y的消费者层的数量。GPU处理每层前向计算的顺序显示在（b）中，从(1)到(5)，突出了DNN训练的逐层计算。在向后传播过程中，生产者和消费者的关系是相反的。

最具内存效率的卷积算法（如cuDNN中的隐式GEMM[8][2]）需要三个数据结构，输入/输出特征图（X和Y）和用于前向传播的层的权重（W）。然而，采用基于快速傅立叶变换（FFT）的卷积算法需要一个额外的临时工作区（WS）缓冲区来管理转换后的地图。

后向传播。对于没有经过充分训练的DNN，推断出的图像类别可能是不正确的。因此，*损失函数*被用来得出前向传播结束时推断错误的大小。具体来说，损失函数的*梯度*是相对于最后一层$_{(N)}$"的输出而言的。

$$\frac{\partial 损失}{\partial Y_{(N)}} \tag{1}$$

方程1中的数值被转发给最后一层$_{(N)}$，作为其输入梯度图（dY），而输出梯度图（dX）是根据*链式规则*[19]得出的。

$$\frac{\partial 损失}{\partial X_{(N)}} \quad \frac{\partial 亏损}{\partial Y_{(N)}} \cdot \frac{\partial Y_{(N)}}{\partial X_{(N)}} \tag{2}$$

因为输出dX（$\frac{\partial Loss}{\partial X_{(N)}}$）是输入的乘积。dY（$\frac{\partial Loss}{\partial Y_{(N)}}$）与$\frac{\partial Y_{(N)}}{\partial X_{(N)}}$，推导出层$_{(N)}$的dX的值一般来说，它的输入/输出梯度图（dY和dX）和该层的输入/输出特征图（X和Y）都需要内存。对于线性网络，计算出的层$_{(N)}$的dX直接传递给前面的层$_{(N-1)}$，作为dY用于层$_{(N-1)}$的dX推导（图2）。

[2]cuDNN（4.0版）提供六种不同的卷积算法。隐式GEMM需要的内存分配最少，因为不需要额外的工作空间。另一方面，基于FFT的卷积算法由于需要额外的数据结构来存储转化为频域的特征图，因此产生了较大的内存分配。更多细节可参见[8，32]。

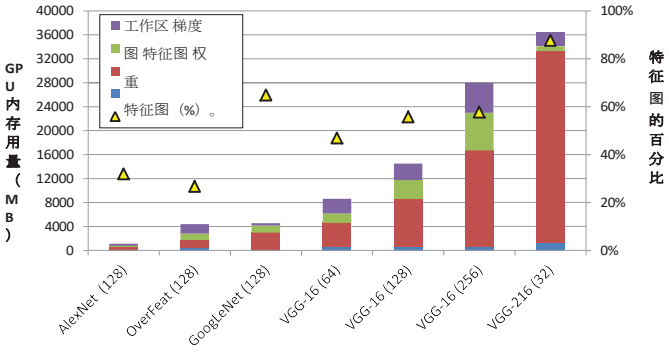

图4：基于功能的GPU内存使用细分（左轴）。右轴显示特征图所消耗的分配内存的比例。

这个链式规则同样被用来得出权重的梯度以更新网络模型。

与前向传播类似，后向传播也是*按层*对各自传入的梯度图，`dYs`进行的。一旦后向传播到达第一层，权重就会利用权重梯度进行调整，从而减少下一个分类任务的预测误差。因此，*训练*一个网络涉及到前向和后向传播，重复进行了数百万次。

数十亿次的迭代。由于基于SGD的反向传播的随机性，网络输入一般都是用数百张图像分批输入的（例如，性能最好的AlexNet和VGG-16的128和256张图像），这增加了内存分配大小，但有助于网络模型更好地收敛到最优解。

C. 动机。可扩展和内存高效的DNN设计

为了帮助神经网络的设计和部署，近年来开发了大量的ML框架，包括Caffe、Torch、Neon、TensorFlow和Theano[9]。这些框架提供了丰富的功能，再加上它们使用GPU加速DNN训练和推理的能力，大大简化了实现神经网络的过程。尽管它们具有灵活性，但流行的ML框架在分配和管理内存的方式上受到严重限制。

为了说明ML框架在管理内存方面的缺陷，请考虑图2中的例子。当使用现有的ML框架训练DNN时，*所有*网络层所需的内存必须符合GPU物理内存的容量。这种GPU端、全网络内存分配策略的关键原因是为了获得性能上的好处。更具体地说，基于页面迁移的虚拟化解决方案将CPU和GPU内存用于页面分配（无论虚拟化功能是否由未来的CUDA运行时扩展或OpenMP（4.0）[33]等编程模型提供），都必须通过PCIe传输页面，这涉及几个延迟密集型过程，如CPU中断系统调用、页面表更新、TLB更新/关闭以及实际页面传输。之前的工作[34]报告说

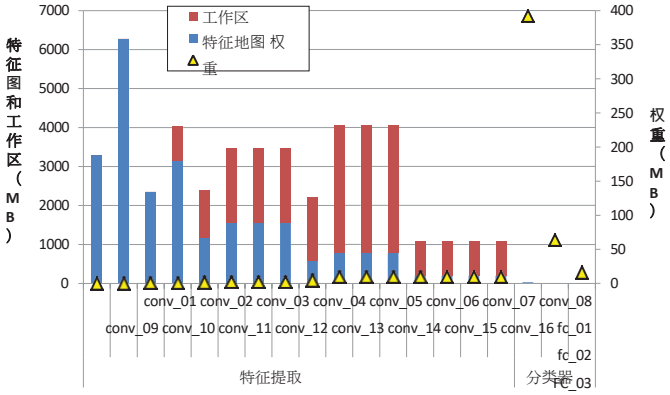

图5：VGG-16（256）的每层内存使用情况。为简洁起见，我们只显示前向传播过程中和包含权重的层（CONV和FC）的内存使用情况。左轴对应的是工作区和每层输入/输出特征图的总和。右轴对应的是存储权重的内存消耗。后向传播期间的内存使用情况与此图类似。

将一个4KB的页面分页到GPU的延迟是20到50μs，这意味着使用分页迁移的PCIe带宽利用率是80到200MB/秒，而DMA启动的`cudaMemcpy`平均达到12.8GB/秒。在16GB/秒的最大PCIe带宽之外。由于对于非常深的网络来说，通过PCIe分页进出的数据量可以达到10几GB（图15），当依靠分页迁移来训练DNN时，ML框架将遭受巨大的性能损失。

请注意，由于后向传播算法的逐层梯度更新规则（链式规则的属性，第II-B节），每一层的特征图（`X`）后来在其自身的后向传播过程中被*重复使用*。这意味着在后向计算完成之前，*所有的*`X`必须仍然在GPU内存中可用。图4显示了基于其功能的内存使用量，以及随着网络的深入，特征图的重要性越来越大。因为更深的网络需要跟踪更多的`X`，所以分配给特征图的内存比例随着层数的增加而单调地增长。然而，无论神经网络的深度如何，网络本身的训练仍然是按层进行的。因此，基线网络范围内的内存分配策略既非常浪费，又无法扩展，因为它没有考虑到逐层的DNN训练。图5显示了VGG-16在前向传播过程中每层的内存使用情况，它提供了以下关键观察。首先，与每层的权重（右轴）相比，中间特征图和工作区（左轴）的内存用量要高一个数量级。第二，这些中间数据结构大多集中在特征提取层，在后面的分类器层中不那么重要。第三，与这些中间数据相比，权重的大小较小，但由于其完全连通性，权重主要集中在分类器层。最后，每层的内存使用量要比28层小得多。

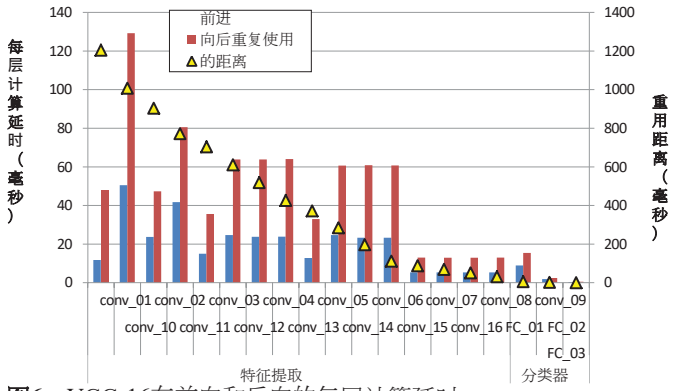

图6：VGG-16在前向和后向的每层计算延时。左轴为各层输入特征图X的重用距离。右轴显示了各层输入特征图的重用距离，X。我们将一个层(n)，X的重用距离定义为完成层(n)的前向传播和同一层(n)的后向传播的开始之间的延迟。

基准策略所需的内存为GB（图1），这表明细化的、分层的内存管理策略有很大的机会来节省内存。

III. 虚拟化的DNN

我们的虚拟化DNN（vDNN）内存管理器的设计目标是*将*DNN的内存使用*虚拟化*，同时使用GPU和CPU内存，并将其最小化。对性能的影响。透明的到由于数据的分配、放置、移动和释放是由系统架构和运行时系统无缝协调的，因此，程序员的工作也是如此。vDNN主要优化特征提取层的内存使用，因为大部分内存使用集中在这些层，在AlexNet上占81%的内存使用，在VGG-16上占96%(256)。更具体地说，我们以这些特征提取层的特征图为目标，因为这些中间数据结构占了GPU内存使用的大部分（图4和图5）。vDNN的直觉也可以应用于权重和分类层，但在节省内存方面的好处较少。

A. 设计原则

vDNN采用了一种基于滑动窗口的*分层*内存管理策略，运行时内存管理器从其内存池中保守地分配内存，供GPU当前正在处理的层立即使用。当前层不需要的中间数据结构将作为内存释放的目标，以减少内存使用。

前向传播。正如II-C节中所讨论的，深度网络必须跟踪大量的相互关系。

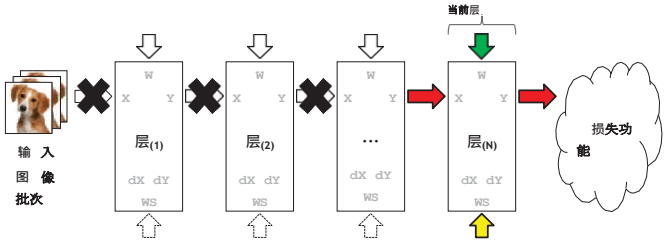

图7：线性网络在前向传播过程中的执行流程。该图假设图层(N)，目前正在由GPU处理。在该层的前向计算中，与标有黑色*X的*箭头相关的数据（所有前面的层的输入特征图）没有被使用，可以安全地从内存池中释放。

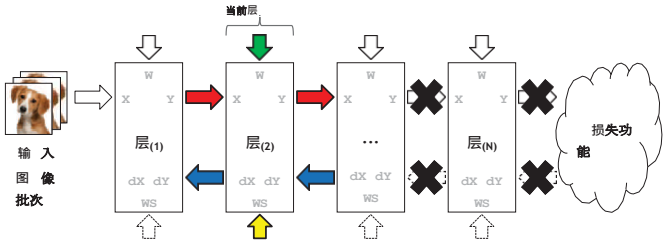

图8：线性网络在后向程序中的执行流程。该图假设图层(2)，目前正在由GPU处理。与箭头相关的数据用黑色的*X*标记。可以安全地被释放，因为它们不会被重新使用。训练这个输入图像批次。

在前向传播过程中提取的特征图（Xs）进行调解。一旦一个给定的层(n)"的前向计算完成，然而，层(n)"的X不会被重用，直到GPU回到同一层(n)"的相应后向计算。由于(n)"层的X的重用距离是以毫秒到秒为单位的（例如，AlexNet和VGG-16（64）的第一层分别超过60毫秒和1200毫秒），深层网络最终分配了大量的X，它们有效地在GPU内存中安营扎寨，没有立即使用（图6）。因此，解决这些Xs的内存优化对有效利用GPU内存至关重要，因为这些中间数据占内存分配的相当大的一部分（图4）。第III-C节详细介绍了vDNN的内存传输策略，该策略决定选择哪些层来加载其X。一旦加载操作完成，vDNN*将*从内存池中*释放*加载的X，以减少GPU内存的使用。

然而，在评估一个层的输入X的floading的可行性时必须小心。这是因为，对于非线性网络拓扑结构，多个层可以是一个先前计算的层的输出特征图（Y）的消费者。(2)例如，图3中的层(2)和层(3)都使用层(1)的输出Y作为其输入X。

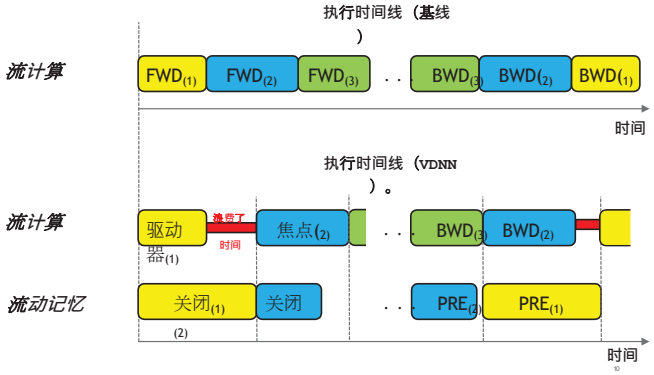

图9：加载和预取的性能影响。$FWD(n)$ 和 $BWD(n)$ 分别是层(n)的前向和后向计算。$OFF(n)$ 是层(n) 's X的ffloading，$PRE(n)$是层(n) 的相应预取操作。

(3)因此，vDNN以数据流图（如图3中的Refcnt）的形式跟踪层间的依赖关系，并且只允许在当前处理层是其输入特征图的最后一个消费者时启动加载/释放操作。图7是一个线性DNN在前向传播过程中的执行流程的例子，强调了何时释放一个层的X是安全的。

后向传播。与前向传播类似，vDNN积极地释放训练其余层的后向计算所不需要的数据结构。在(n)层的向后传播期间，不再需要$(n+1)$ 层的 Y 和 dY，因为GPU 已经完成了该层的梯度更新（图8）。同样，通过利用逐层的DNN向后传播，一旦该层的向后计算完成，vDNN会立即释放该层的Y和dY。X和dX不会被释放，因为前一层的后向传播将需要这些值来进行梯度推导。需要注意的是，如果一个层将其X载入主机内存，vDNN应该保证在梯度更新开始前将载入的数据拷贝回GPU内存。因此，vDNN为图层(n) 's offloaded feature maps启动*预取*操作，该操作与图层(m) 's backward computation重叠，$n < m$，因此预取在其实际使用之前启动，隐藏了预取的延迟。

B. 核心业务及其设计

vDNN的原型是cuDNN[8]之上的一个层。每一层都跟踪输入/输出特征图的跨层数据依赖关系，以便正确安排vDNN的加载和释放操作。vDNN采用了两个独立的CUDA流[36]，将正常的DNN操作与vDNN的内存分配、移动和释放操作重叠起来。

```
00 // currLayerId: 该类方法的调用层的ID。
01 // layers[n]->offloaded: 当一个层卸载其输入的特征图时设置为true 02 //
layers[n]->prefetched: 最初对所有层设置为false
03
04 int Network::findPrefetchLayer(int currLayerId) {
05       // 搜索所有前面的层
06       for(int id=(currLayerId-1); id>=0; id--){
07            // 找到离当前层最近的层，需要预取 08            if( (layer[id]-
>offloaded=true)&&(layer[id]->prefetched=false) ){ 09                  // 标记该
层为被当前层预取的层
10                 layers[id]->prefetched = true。
11                 返回id。
12            }
13            // 在到达搜索窗口的末端之前无法找到预取层。
14            else if(layer[id]->layerType==CONV) {
15                 return -1; // 找不到要预取的层ID
16            }
17       }
18 }
```

图10：解释vDNN如何寻找预取层的伪代码。请注意，搜索操作只到下一个最近的CONV层（第14行），保证预取的X在未来不会被用得太远，因为它限制了预取层在搜索窗口的层内。

streammemory管理VDN的三个关键部分：内存分配/释放、加载和预取。

内存分配/释放。CUDA库只支持*同步*内存（去）分配，这意味着对cudaMalloc()或cudaFree() 的任何调用都会在一个节点内的所有GPU上执行额外的同步。为了安全地实现VDN 的内存操作，同时不落入同步CUDA API的陷阱，我们采用了NVIDIA[37]发布的开源*异步*内存分配/释放API库。当程序启动时，VDNN内存管理器被分配了一个内存池，其大小与物理GPU内存容量相当。每当VDNN分配（和释放）数据结构时，底层内存管理器将从这个内存池中保留（和释放）内存区域，而不必调用cudaMalloc()或cudaFree()。

内存加载。对输入特征图的加载是vDNN节省内存的关键因素之一。当一个图层被选择用于加载时，vDNN首先使用cudaMallocHost()分配一个钉住的主机端内存区域，然后streammemory使用cudaMemcpyAsync()通过PCIe启动该图层的X到钉住内存的非阻塞内存传输，与同一图层的cuDNN的正向计算重叠。目前vDNN的实施在每一层的前向计算结束时，如果streammemory已经加载了其特征图，则会同步streamcompute和streammemory。这种方法保证了在下一层开始前向计算之前，已加载的数据被安全地从内存池中释放出来，使加载的内存节约优势最大化。因为CONV和POOL层的X是只读数据结构，所以(n)，与同一层的前向传播重叠的层的offload操作不会产生任何正确性问题。ACTV层已经被重构为就地算法，只使用Y和dY进行梯度更新，避免了内存加载的需要（第二节B）。图9提供了一个vDNN的加载操作的概述。这里，基线系统能够

一旦完成了(1) ， 就立即启动(2) ，进行前向计算。(2)层的执行对于vDNN来说是停滞的，因为streamcompute必须等到streammemory的加载操作完成，从而阻碍了(2) 层的计算。然而，(3) 层的计算并没有延迟，因为(2)层的加载延迟完全隐藏在计算同一层的前向传播的延迟中。

内存预取。与offloading类似，使用cudaMemcpyAsync()将offloaded的Xs预取回GPU内存，使数据传输与后向传播的计算重叠。然而，相对于前向传播的加载操作而言，streammemory以相反的顺序启动预取操作（图9）。正如在III-A节中提到的，预取的一般规则是将层(n) "加载数据 "的内存拷贝操作与层(m) "后向计算"重叠，层ID m总是高于n，以最大化预取和延时隐藏的好处。换句话说，当GPU开始对(m) ，vDNN确定在前面的层中预取的最佳层（因为$n < m$）。

如果预取层(n) 和超限层(m)之间的距离太远，那么vDNN的内存节约优势将被削弱，因为这个预取数据的重用时间将在未来很遥远。换句话说，过早地预取数据将再次次优地利用GPU内存，因为预取的数据将再次停留在GPU内存中而不能立即使用。我们精心设计了vDNN的预取算法，以避免这一缺陷，并平衡加载的内存节约优势和预取的及时性。图10是vDNN预取算法的一个伪代码，它可以确定预取的最佳候选层。在streamcompute开始一个层的后向计算之前，vDNN首先搜索需要预取其X的潜在层。如果搜索操作成功（第11行），要预取的层ID由findPrefetchLayer例程返回，并用于通过streammemory启动其预取操作。与加载类似，vDNN将streamcompute和streammemory同步，这样下一层的后向计算就会停滞，直到预取操作结束。因此，在(n) "层的后向计算中启动的任何预取操作都能保证在$(n-1)$ "层的计算之前准备就绪。当然，当预取延迟长于重叠计算时，这种好处是以潜在的性能损失为代价的，我们将在第V-C节详细说明。

C. vDNN内存传输策略

确定最佳层来加载其特征图是一个必须考虑的多维优化问题。

1）GPU内存容量，2）使用的卷积算法和整体层的内存使用，以及3）整个网络的性能。前两个因素决定了我们是否能够对网络进行训练（我们的目标是："训练"）。指的是网络的*可训练性*），而最后一个因素决定了整体的训练*效率*。如果vDNN对所有层都使用最节省内存的算法（例如cuDNN中的隐式GEMM[8]，不需要任何WS分配），同时所有层都有fload/refetch，那么GPU内存的使用量将是最低的。然而，与各层采用最快的卷积算法的基线相比，性能可能会受到影响；性能损失主要来自于：1）可能由于flarge/refetch而产生的额外延迟，以及2）内存最优的隐式GEMM和性能最优的卷积算法之间的性能差异。使用最快的算法，而不使用任何offload/refetch，将导致尽可能高的性能，但是更快的算法的工作区的潜在内存开销和在GPU内存内的累积Xs可能会超出GPU内存的范围。鉴于优化层级内存使用及其性能本身就是一个多维的优化问题，在整个网络中选择最优化的超参数是不容易的。因此，我们采用了以下基于启发式的内存传输策略，缩小了参数的选择范围，简化了优化问题，同时在实践中仍然表现稳健。

静态的vDNN。特征提取层主要由CONV和ACTV层组成，间歇性的POOL层可以降低特征图的维度。然而，对于深层神经网络来说，超过70%到80%的（前向/后向）运算时间是花在CONV层上的。因此，我们评估了两种利用这种计算特性的*静态*vDNN内存传输方案。我们探索的第一个选项是让vDNN内存管理器加载所有层的所有Xs。这个策略，vDNN$_{all}$，是我们最节省内存的解决方案，因为所有的X都被加载并从GPU中释放，大大减少了设备内存的使用。第二个vDNN策略是只为CONV层加载X，而将其余层的X留在GPU内存中（vDNN$_{conv}$ ）。vDNN$_{conv}$策略是基于这样的观察，即CONV层的计算延迟比ACTV/POOL层长得多，更有可能有效地隐藏FLOAD/Prefetch的延迟。毫不奇怪，vDNN$_{conv}$ 的性能通常高于vDNN$_{all}$。但vDNN$_{all}$的优势是消耗的GPU内存最少，大大增强了DNN的训练能力。我们随后评估了这两种静态策略的内存使用情况和性能，并采用了内存最优和性能最优的卷积算法。

动态的VDNN。虽然静态的VDNN很简单，也很容易它没有考虑到决定DNN可训练性和性能的系统结构组件（例如，最大计算FLOPs和内存带宽、内存大小、有效PCIe带宽等）。对于可以舒适地在GPU内存中运行的DNN来说，vDNN$_{all}$ 和vDNN$_{conv}$都不是最佳选择，因为最好的方法是让所有的内存分配都驻留在GPU中，而不用

任何floading和采用最快的卷积算法。另一方面，大型的深度网络可能无法使用更快的卷积算法。因此，能够在GPU上对这样的网络进行优化是vDN可以做的最好的优化。因此，我们开发了一个*动态的*vDNN策略，在运行时自动确定加载层和采用的卷积算法，以平衡DNN的训练性和性能。动态虚拟DNN利用了DNN训练的几个特性。首先，我们利用了训练所需的同一前向/后向传播通道的数百万至数十亿次迭代。英伟达公司的cuDNN提供了一个运行时API，该API对某一特定层的所有可用卷积算法进行实验，评估每种算法的性能及其内存使用情况。目前的ML框架利用这个API来进行最初的排序阶段，以确定为每个CONV层部署最佳性能的最佳算法。这种程序设计的开销只有几十秒，相对于DNN训练所需的几天到几周的时间来说，是可以忽略不计的。我们的动态虚拟数字网络通过一些额外的程序来选择最好的层来加载和最好的每层算法，从而增强了这个程序阶段。一旦基线程序阶段完成，所有CONV层的最快卷积算法就会产生，动态vDN会采用以下额外的程序。

1) 首先，静态的vDNN$_{all}$，*使用*内存最优、无WS发生的算法对所有CONV层进行一次训练测试。这个初始通道决定了目标DNN是否可以被训练，因为它需要最少的GPU内存。
2) 如果vDNN$_{all}$通过，则启动另一个训练阶段，所有CONV层采用最快的算法，但没有任何floading。这样的配置，如果成功通过，将在剩下的全部训练过程中采用，因为它提供了高的性能，同时保证了训练的可行性。如果此阶段由于内存超载而失败，则用同样最快的算法测试另外两个训练通道，但在vDNN$_{conv}$和vDNN$_{all}$中分别启用vDNN的加载功能。如果成功，vdnN将在剩下的训练中使用成功的配置。如果vDNN$_{conv}$和vDNN$_{all}$都失败了，我们就进入下一个程序，以进一步减少内存使用。
3) 最后一个阶段是基于一个*贪婪的*算法，它试图在本地减少一个层的内存使用，在可训练性和性能方面寻求一个全球最佳状态。在遍历每一层时，vDNN首先计算使用最快的算法是否会超出GPU的内存预算。如果是这样，那么给定层的卷积算法将被*局部*降级为性能较差但更节省内存的算法，直到达到内存最优的隐式GEMM。这种基于贪婪的方法，首先 尝试了vDNN$_{conv}$，每个CONV层最初使用自己的性能最优算法。如果vDNN$_{conv}$ 失败了，那么就用更节省内存的vDNN$_{all}$开始另一次训练。如果vDNN$_{all}$在这种贪婪的算法下也失败了，那么vDNN又回到了最开始的vDNN$_{all}$的解决方案，在整个网络中使用内存最优、无WS的算法。

虽然其他可能的设置可能会更好地平衡性能和可训练性，但我们发现，我们的动态vDN的性能是有竞争力的，不需要详尽地搜索全局最佳参数选择。

IV. 方法论

A. vDNN内存管理器

我们实现了一个主机端内存管理器，与最新最快的cuDNN 4.0版本[8]互动，作为GPU后端。构成DNN特征提取层的所有层都是用cuDNN实现的，每个层的执行都是用两个CUDA流、streamcompute和streammemory来协调的，如第三节B所讨论的。分类层保持不变，使用Torch中使用的相同的cuBLAS程序。vDNN的API与Torch和Caffe的API非常相似，提供了目标DNN和其每一层组成的高级抽象。

虽然Torch、Caffe和Theano的内存分配方案有细微的差别，但之前的工作[9]定量地证明了这三个框架都表现出相当的性能和内存需求[3]。因此我们选择Torch的内存管理策略作为基线，与vDNN进行比较，因为它在学术界和工业界都有广泛的部署（例如Facebook和Google DeepMind）。这个基线政策采用了II-C节中讨论的全网分配政策。然而，我们使用以下策略进一步改进这个基线策略，以减少后向传播阶段的内存消耗[38, 39]：我们不是为所有单独的层分配单独的dY和dX，而是只分配这些数据结构的最小需要数量，并在每个层的后向计算完成后重新使用它们（图2）。

B. GPU节点拓扑结构

我们在NVIDIA的Titan X[40]上进行了实验，它提供了Maxwell GPU家族中最高的数学吞吐量（单精度吞吐量为7 TFLOPS）、内存带宽（最大336 GB/秒）和内存容量（12 GB）。该GPU通过PCIe开关（gen3）与英特尔i7-5930K（包含64GB的DDR4内存）进行通信，它提供了最大16GB/秒的数据传输带宽。

[3]由于TensorFlow在GPU内存使用和训练速度方面表现最差[9]，我们在本文中不进一步讨论其内存管理策略。

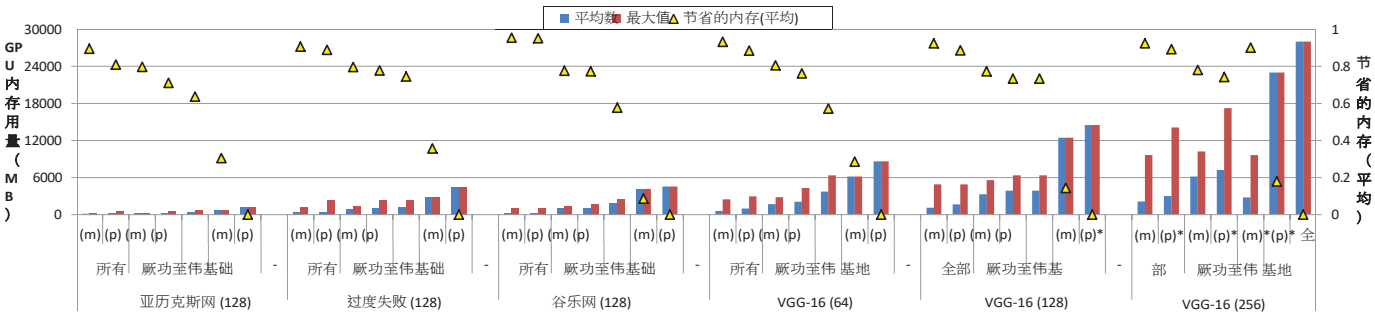

图11：平均和最大内存用量（左轴）。右轴对应的是平均内存使用量的节省。

C. DNN基准测试

传统的DNNs。首先，我们评估了现有的、最先进的ImageNet获奖DNN。AlexNet[1]、OverFeat[30]、GoogLeNet[17]，以及VGG-16（具有16个CONV层和3个FC层的最深网络）[14]的三种不同批量大小。这些DNN的网络构架（如层的类型、批量大小等）与Facebook[41]的研究人员所保留的参考模型完全相同。虽然AlexNet、OverFeat和GoogLeNet的内存使用量已经低于Titan X的12GB内存容量（图4），但我们用它来评估这些网络与vDNN的性能回归。VGG-16是迄今为止最大和最深的DNN架构之一，需要大量的内存容量来实现可训练性（当以256的最佳性能批次大小进行训练时，使用高达28GB的内存）。因此，Simonyan和Zisserman[14]将VGG-16（256）在四个GPU上并行化，每个GPU训练VGG-16（64），在一个GPU的内存预算内完成。因此，我们研究了具有三种批处理规模（64/128/256）的VGG-16，并将其作为一个具有代表性的、未来的DNN架构，强调当今GPU的内存容量限制。

非常深的网络。为了突出vDNN在训练深度网络方面的可扩展性，我们收集了第二组通过扩大VGG的CONV层数，从16个CONV层扩大到416个CONV层，从而达到了基准。该原始的VGG网络具有同质化的结构，只使用3×3的卷积运算（跨度1和垫1）和2×2的池化运算（跨度2），从从第一层到最后一层的特征提取层。特征提取层在概念上分为五组CONV层，由中间的POOL层分开。这些CONV层组之间的唯一区别是，从第一层组到最后一层组，输出特征图的数量从64增长到512。Simonyan和Zisserman[14]研究了层深度对分类精度的影响，他们在这些层组中逐步增加CONV层，从8个CONV层到16个CONV层。我们采取类似的措施来加深VGG的层深度，在VGG-16中逐渐增加100个CONV层，从而形成VGG-116/216/316/416的构架。每增加100个CONV层是通过在五个CONV层组中的每个组增加20个CONV层来完成的。增加的CONV层的输出特征图的数量与该层组采用的相同。我们使用这些四个VGG风格的网络，对vDNN在训练需要更多内存的非常深的网络上的可扩展性进行了案例研究。与输入批量大小为数百张图片的传统DNN相比，我们研究了这些批量大小为32张的非常深的网络，以突出层*深度*对DNN的内存扩展效应。

V. 结果

本节评估了vDNN对GPU内存使用率、片外内存带宽利用率、GPU功耗和整体性能的影响。在本节讨论的所有图表中，静态vDNN$_{all}$ 和 vDNN$_{conv}$ 政策分别表示为 `all` 和 `conv`，并在整个网络中用内存最优和性能最优（表示为（m）和（p））的卷积算法进行评估。基线内存管理器（base）也是用内存最优和性能最优算法进行评估的。如第III-C节所述，这些算法是为vDNN$_{dyn}$（表示为dyn）动态选择的。在训练网中失败的内存管理策略由于内存超限，在工作时，用(*)标记。

A. GPU内存使用情况

由于vDNN采用了分层的内存分配策略，在前向/后向传播过程中，GPU的内存使用量会根据所选择的内存加载策略和给定层采用的卷积算法而变化（图5）。因此，我们讨论最大和平均内存使用量，如图11所示。最大内存使用量对应于整个运行过程中分配的最大内存，它决定了目标DNN应用是否可以被训练。另一方面，平均内存反映了平均使用了多少内存，反之，在前向/后向传播期间释放了多少内存。平均内存使用量越小，vDN就越有可能通过以下方式提高性能。1）采用需要更大工作空间的性能高效的卷积算法，以及2）减少负载层的总数，防止因负载而导致的潜在性能下降（图9）。

由于基线策略规定了内存分配，以适应整个网络的使用，最大和平均的内存使用是相同的。基线策略

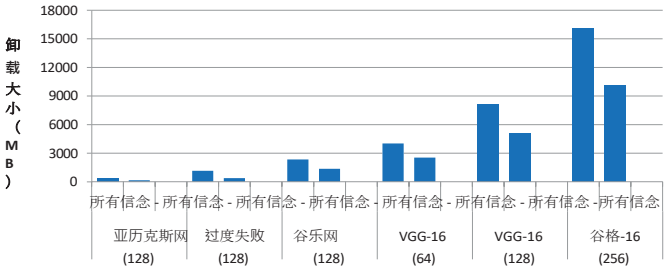

图12：使用cudaMallocHost()将GPU内存分配floaded到主机侧的pined内存的大小。

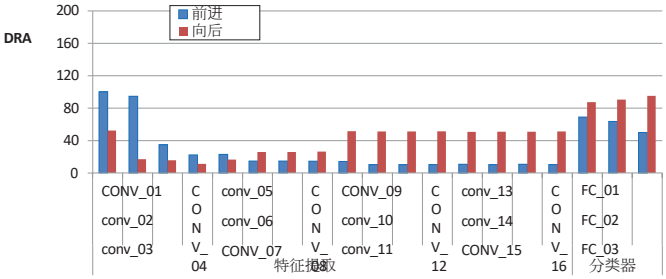

图13：每个CONV层的前向和后向传播的最大DRAM带宽利用率。

因此，它不能训练像VGG-16这样的批处理128和256的网络，这需要超过物理上可用的12GB的内存。我们的vDNN通过大幅降低内存需求来增强网络的可训练性。总的来说，内存最优的vDNN$_{all}$ (m)显示了最小的平均和最大的内存使用量，因为它总是在使用最节省内存的算法时加载一个层的输入特征图。因此，vDNN$_{all}$表现出被发送到主机内存的最高流量，对于VGG-16 (256)，GPU内存节省达到16 GB（图12）。这种积极的加载方式极大地提高了内存的效率，在图11所示的六个网络的最大和平均内存使用量上，平均减少了73%和93%。当采用性能最优算法时，vDNN$_{all}$的平均内存节省率略微降低到64%和90%，最大和平均内存使用量。由于vDNN$_{conv}$只加载了CONV层的特征图，它的内存节省没有vDNN$_{all}$那么高。然而，即使在整个网络中采用了性能最优的算法，vDNN$_{conv}$仍然将最大和平均的内存使用量减少了52%和76%。vDNN$_{dyn}$在三种vDNN策略中，分配的内存最大，相比之下，最大和平均内存消耗分别减少了49%和69%。基准线。这是因为vDNN$_{dyn}$试图平衡其内存使用和性能，寻求在GPU内存中实现网络的优化，同时通过最小化加载层的数量和采用最快的卷积算法来优化性能。另一方面，静态的vDNN$_{all}$和vDNN$_{conv}$，在选择加载层时不考虑整体性能。例如，VGG-16 用内存最优的vDNN$_{all}$训练出来的(128)只使用了12GB的可用内存中的4.8GB。这种配置导致了61%的性能损失（第V-C节），因为vDNN$_{all}$未能利用剩余的7.2GB内存进行性能优化。vDNN$_{dyn}$试图通过动态推导出负载层以及每层采用的最佳卷积算法来弥补这一差距。我们在第五节C部分进一步讨论了vDNN对性能的影响。

B. 对记忆系统的影响

虽然vDNN有助于虚拟化DNN的内存使用，但它的代价是增加了更多的读（offload）和写（Write）。(预取）流量到GPU内存子系统，可能会干扰正常的cuDNN操作。因为额外的vDNN内存流量可以达到PCIe的带宽（gen3最大为16GB/秒），它对性能的影响将由正常cuDNN操作的内存带宽强度决定。图13显示了VGG-16的基线的最大DRAM带宽利用率，这是为每个CONV层的前向和后向传播分别测量的。特征提取层很少使336 GB/秒的峰值内存带宽达到饱和，为vDN的加载/再取流量提供了足够的空间。即使一个假设的、未来的对话算法完全饱和了片外DRAM带宽，vDNN的额外traffic也会产生最坏情况下（16/336）=4.7%的性能开销，我们认为考虑到虚拟化内存的好处，这是合理的。

C. 业绩

图14总结了vDNN与基线的性能比较。为了进行保守的评估，我们只比较特征提取层产生的延迟，因为分类器层对基线和vDNN的执行是相同的。由于基线策略对VGG-16（128）和VGG-16（256）需要超过12GB的内存，并采用性能最优的算法（分别为15GB和28GB），因此在Titan X上 不可能训练这两个网络。这个基线的性能是通过用最快的算法配置所有的CONV层并单独评估每个层的延迟来估计的。后来，这些延迟被累积起来以估计整体性能。总的来说，与基线相比，采用内存最优算法的vDNN$_{all}$和vDNN$_{conv}$表现出平均58%和55%的性能损失（最大降幅为65%和63%），这是一个预期的结果，因为内存管理器没有努力平衡内存使用和整体性能。动态VDN$_{dyn}$在平衡内存效率和整体吞吐量方面做得更好，缩小了静态VDN和基线之间的性能差距，平均达到了基线吞吐量的97%（最坏的情况是，对于VGG-16(256)来说，是oracular基线的82%）。

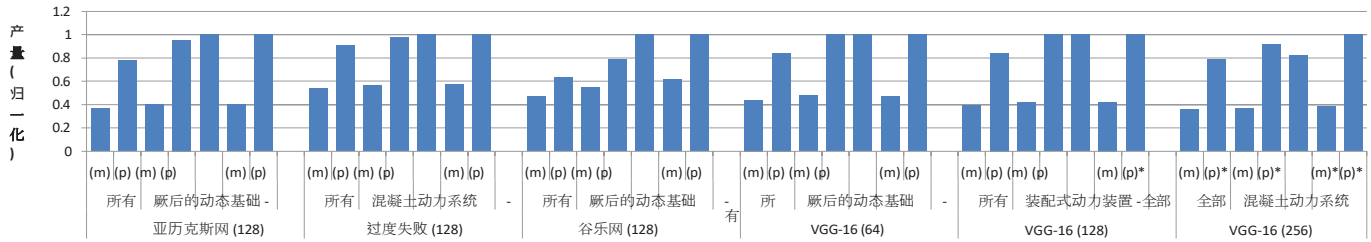

图14：总体性能（对基线进行归一化）。

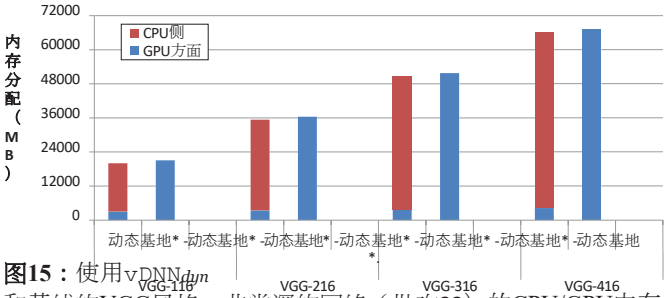

图15：使用vDNN*dyn*和基线的VGG风格、非常深的网络（批次32）的CPU/GPU内存分配情况。

D. 权力

本节讨论了vDNN*dyn*对GPU整体功耗的影响。我们使用nvprof[42]的系统描述工具来测量平均和最大GPU功耗。每个网络执行50次前向和后向传播的迭代，并对平均和最大功耗进行平均。除了VGG-16(128)之外，所有的网络都是用性能最优的卷积算法执行的，因为VGG-16(128)只能用基线下的内存最优算法进行训练（图11）。请注意，没有讨论VGG-16（256）的结果，因为这种构型只能用vdN训练，所以无法与基线进行比较。总的来说，VDN*dyn*，产生了1%到7%的最大功率开销。正如第V-B节所讨论的，VDNN的flad/refetch内存traffic是造成峰值功耗瞬间上升的最大因素之一。然而，由于以下两个因素，平均功耗（能量/时间）很少受到影响。1）对于这五个网络来说，VDN*dyn*，不会产生任何明显的性能开销；2）所研究的DNN很少使DRAM的峰值带宽达到饱和（图13），因此VDN内存轨迹的额外能量开销预计平均可以忽略不计（第五节B）。

E. 案例研究。训练非常深的网络

为了突出vDNN在训练深度网络方面的可扩展性，我们对四个包含数百个CONV层的VGG式网络进行了案例研究，并扩大了网络工作的内存需求。如IV-C节所述，批量大小被设定为比前几节所研究的要小得多（批量大小从128到256不等），以突出层深度的内存扩展效应，尽管其批量大小很小。图15显示了内存基线和vDNN的分配要求*dyn*对于这些非常深的神经网络。随着CONV层的数量从16增加到416，基线的内存需求单调地增加了14倍（从4.9GB到67.1GB），即使是32的小批量。由于其分层内存分配策略，vDNN*dyn*大大减少了所有四个网络的内存使用，只使用了4.2GB的GPU内存，其余81%至92%的内存分配都在CPU内存中。与白话基线相比，vDNN*dyn*也没有产生任何明显的性能下降，因为加载和预取延迟完全隐藏在该层的DNN计算中，同时仍然能够在整个网络中采用性能最优的算法。

VI. 相关工作

已有多种建议旨在减少神经网络的内存使用。网络修剪技术[43, 44, 45, 46, 47]从网络中移除小值权重连接，作为减少网络冗余的手段，导致内存消耗的 减少。另一种减少冗余的方法使用量化[48]或降低精度[49]来减少网络建模所需的比特数。Gong等人[50]也探索了多种网络压缩技术，以减少DNN的内存使用。

尽管这些先前的研究减少了DNN的内存再需求，但它们在几个方面还不够。首先，如图 4 所示，在最先进的DNN中，权重只占内存用量的一小部分。因此，优化权重的内存使用的建议，虽然在内存带宽利用和能源效率方面有好处，但只提供了有限的内存容量节省机会。其次，除非针对给定的网络和任务进行仔细调整，否则使用降低的精度有时会导致分类精度的损失。我们的建议优化了中间特征图的内存消耗，这是DNN中最主要的数据结构。

之前的一些工作讨论了在GPU上支持虚拟内存的机制。Pichai等人[51]和Power等人[52]提出了考虑GPU独特内存访问模式的TLB实现，改善了地址转换的吞吐量以及整个系统的吞吐量。Zheng等人[34]讨论了GPU硬件和软件堆栈所需的功能，以缩小GPU分页内存与传统程序员的性能差距。

有方向的内存管理技术。正如II-C节中所讨论的，基于页面迁移的虚拟化解决方案很可能会大大降低PCIe带宽的利用率，并在训练超额订购GPU内存的网络时产生性能开销。

虽然与vDNN没有那么直接的关系，但各种加速器架构也被提出来用于DNN[20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 53]。虽然这些定制的ASIC设计极大地提高了DNN的能效，但这些都没有解决DNN训练的内存容量瓶颈问题，这是我们工作的独特贡献。

VII. 总结

现有的机器学习框架要求用户仔细管理他们的GPU内存使用情况，以便使整个网络的内存需求符合物理GPU内存的大小。我们提出了vDNN，一个可扩展的、内存高效的运行时内存管理器，它将网络的内存使用在CPU和GPU内存之间虚拟化。我们的vDNN解决方案将AlexNet的平均GPU内存使用量减少了89%，OverFeat减少了91%，GoogLeNet减少了95%，大大改善了DNN的内存效率。与VGG-16（256）类似的实验结果是，与白话基线相比，内存使用量平均减少了90%，代价是性能下降了18%。我们还研究了vDNN对极深的神经网络的可扩展性，表明vDNN可以训练具有数百层的网络而没有任何性能损失。

鸣谢

我们感谢NVIDIA的同事对这项工作的反馈，特别是John Tran、Sharan Chetlur、Simon Layton和Cliff Woolley对vDNN概念和基础设施的贡献。

参考文献

[1] A.Krizhevsky, I. Sutskever, and G. Hinton, "Ima- geNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the Advances in Neural Information Processing Systems*, 2012.

[2] A.A. Graves和J. Schmidhuber, "Framewise Phoneme Classification With Bidirectional LSTM and Other Neural Network Architectures," in *Neural Networks*, 2005.

[3] R. Collobert, J. Weston, L. Bottou, M. Karlen, K.Kavukcuoglu, and P. Kuksa, "Natural Language Processing (Almost) From Scratch," in *arxiv.org*, 2011.

[4] Tensorflow，"https://www.tensorflow.org"，2016。

[5] 火炬，"http://torch.ch"，2016年。

[6] Theano，"http://deeplearning.net/tutorial，"2016。

[7] Caffe，"http://caffe.berkeleyvision.org"，2016年。

[8] NVIDIA，"cuDNN。GPU加速的深度学习，" 2016年。

[9] S.Bahrampour, N. Ramakrishnan, L. Schott, and M.Shah, "Comparative Study of Caffe, Neon, Theano, and Torch for Deep Learning," in *arxiv.org*, 2016.

[10] The Next Platform (www.nextplatform.com), "Baidu Eyes Deep Learning Strategy in Wake of New GPU Options," 2016.

[11] G. Diamos, S. Sengupta, B. Catanzaro, M. Chrzanowski, A. Coates, E. Elsen, J. Engel, A. Hannun, and S.Satheesh, "Persistent RNNs:Stashing Recurrent Weights On-Chip," in *Proceedings of the International Conference on Machine Learning*, 2016.

[12] A.Krizhevsky, "One Weird Trick For Parallelizing Convolutional Neural Networks," in *arxiv.org*, 2014.

[13] ImageNet，"http://image-net.org，"2016。

[14] K.Simonyan和A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proceedings of the International Conference on Learning Representations*, 2015.

[15] K.He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *arxiv.org*, 2015.

[16] 连线》（www.wired.com），"微软神经网络显示深度学习可以变得更深入"，2016年。

[17] C.Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions, " in *arxiv.org*, 2014.

[18] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep Networks with Stochastic Depth, " in *arxiv.org*, 2016.

[19] Y.LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition, " in *Proceedings of the IEEE*, 1998.

[20] T.Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O.Temam, "DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[21] Y.Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li。
T.Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao:一台机器学习超级计算机"，载于《*ACM/IEEE国际微结构技术研讨会论文集*》，2014年。

[22] Y.Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efcient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *IEEE International Conference on SolidState Circuits*, 2016.

[23] S.Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proceedings of ACM/IEEE International Symposium on Computer Architecture*, 2016.

[24] Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proceedings of ACM/IEEE International Symposium on Computer Architecture*, 2016.

[25] R.LiKamWa, Y. Hou, M. Polansky, Y. Gao, and L.Zhong,
"RedEye:用于连续移动视觉的模拟ConvNet图像传感器结构"，在*Pro-Eye杂志上*。

ceedings of ACM/IEEE International Symposium on Computer Architecture, 2016.

[26] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H.Lee, S. Lee, J. Miguel, H. Lobato, G. Wei, and D.Brooks, "Minerva。Enabling Low-Power, High-Accuracy Deep Neural Network Accelerators," in *Proceedings of ACM/IEEE International Symposium on Computer Architecture*, 2016.

[27] P.Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "A Novel Processing-in-memory Architec- ture for Neural Network Computation in ReRAM-based Main Memory, " in *Proceedings of ACM/IEEE Interna- tional Symposium on Computer Architecture*, 2016.

[28] A.Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, 和 Srikumar, "ISAAC: A Convolutional Neural Net- work Accelerator with In-Site Analog Arithmetic in Crossbars," in *Proceedings of ACM/IEEE International Symposium on Computer Architecture*, 2016.

[29] J.Albericio, P. Judd, T. Hetherington, T. Aamodt, N.E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-Neuron-Free Deep Convolutional Neural Network Com- puting," in *Proceedings of ACM/IEEE International Symposium on Computer Architecture*, 2016.

[30] P.Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Lo- calization and Detection using Convolutional Networks," in *arxiv.org*, 2013.

[31] S.钦塔拉，"https://github.com/torch/nn/pull/235"，2015年。

[32] S.Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J.Tran, B. Catanzaro, and E. Shelhamer, "cuDNN:Efficient Primitives for Deep Learning," in *Proceedings of the Advances in Neural Information Processing Systems*, 2014.

[33] OpenMP架构审查委员会，《OpenMP应用程序接口（4.0版）》，2013年。

[34] T.Zheng, D. Nellans, A. Zulfiqar, M. Stephenson, and S. W. Keckler, "Toward High-Performance Paged-Memory for GPUs," in *Proceedings of IEEE International Symposium on High-Performance Computer Architecture*, 2016.

[35] 英伟达公司，"英伟达NVLINK高速互连网"，2016年。

[36] NVIDIA，"NVIDIA CUDA编程指南"，2016年。

[37] nvidia，"https://github.com/NVIDIA/cnmem，"2016。

[38] S.格罗斯和M.威尔伯，"训练和调查剩余网"，2016年。

[39] T.Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T.Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet:A Flexible and Efcient Machine Learning Library for Heterogeneous Distributed Systems," in *Proceedings of the 2015 Workshop on Machine Learning Systems*, 2015.

[40] NVIDIA，"GeForce GTX Titan X (Maxwell)"，2015年。

[41] S. 钦塔拉。 "https://github.com/soumith/convnet-benchmarks," 2016年。

[42] NVIDIA，"CUDA工具包7.5文档。Profiler," 2016年。
[43] S.Hanson和L. Pratt, "Comparing Biases for Mini-mal Network Construction with Back-propagation," in *Proceedings of the Advances in Neural Information Processing Systems*, 1989.
[44] Y.LeCun, S. Denker, and S. Solla, "Optimal Brain Damage," in *Proceedings of the Advances in Neural Information Processing Systems*, 1990.
[45] B.Hassibi and D. Stork, "Second Order Derivatives for Network Pruning:Optimal Brain Surgeon," in *Proceedings of the Advances in Neural Information Processing Systems*, 1993.
[46] S.Han, J. Pool, J. Tran, and W. Dally, "Learning Both Weights and Connections for Efcient Neural Networks," in *Proceedings of the Advances in Neural Information Processing Systems*, 2015.
[47] S.Han, H. Mao, and W. Dally, "Deep Compression:用剪枝、训练量化和Huffman编码压缩深度神经网络"，载于2016年*国际学习表征会议论文集*。
[48] V.Vanhoucke, A. Senior, and M. Mao, "Improving the Speed of Neural Networks on CPUs," in *Proceedings of Deep Learning and Unsupervised Feature Learning*, 2011.
[49] P.Judd, J. Albericio, T. Hetherington, T. Aamodt, N.E. Jerger, R. Urtasun, and A. Moshovos, "Reduced-Precision Strategies for Bounded Memory in Deep Neu- ral Nets, " in *arxiv.org*, 2016.
[50] Y.Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing Deep Convolutional Networks Using Vector Quantization," in *arxiv.org*, 2014.
[51] B.Pichai, L. Hsu, and A. Bhattacharjee, "GPU上地址转换的架构支持。Designing Memory Management Units for CPU/GPU with Uni-fied Address Spaces," in *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.
[52] J.Power, M. Hill, and D. Wood, "Supporting x86-64个地址转换为100个GPU通道"，载于*《IEEE高性能计算机架构国际研讨会论文集》*，2014年。
[53] Z.Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo。X.Feng, Y. Chen, and O. Temam, "ShiDianNao:将视觉处理转移到更接近传感器的地方，"*ACM/IEEE计算机架构国际研讨会论文集*，2015。