指导教师：杨涛　　提交时间：2016/3/25

CVPR2015 Paper Translation

No: 01

姓名：李傲

学号：2012300842

班号：HC001210

稀疏卷积神经网络

刘宝元 王民 哈桑·佛罗士 马歇尔·塔彭 玛丽安娜·彭琪
中佛罗里达大学 Amazon.com
{bliu, mwang, forooshg}@cs.ucf.edu, tappenm@amazon.com,
Marianna.Pensky@ucf.edu

摘要

在以大量的参数和计算复杂度为代价情况下，深度神经网络已经在图像分类及目标物体检测问题方面取得了显著的成果。在本项工作中，我们将展示如何使用稀疏分解来减少参数冗余。最大稀疏性由利用信道间和信道内冗余，结合微调步骤，最小化由稀疏性最大化导致的损失得到。在ILSVR2012 数据集上的实验显示，这个过程在精确度损失小于 1%的情况下消去了超过 90%的参数。在这样的环境下，我们还提出了一个基于稀疏卷积神经网络(SCNN)模型的高效稀疏矩阵乘法 CPU 实现算法。我们的 CPU 实现表现出的效率远远高于传统稀疏矩阵库及基于原始密集网络实现显著加速的算法。此外，我们在目标物体检测问题中将稀疏卷积神经网络与级联模型和稀疏全连接层连接，取得了非常显著的速度提升。

1. 简介

本文展示如何利用稀疏分解在一个卷积神经网络中表示过滤步骤能够在保证系统精确度的前提下显著的降低计算复杂度。深度神经网络已经在图像分类及目标物体检测问题方面取得了显著的成果 [14][8]。近些年来，ImageNet LSVRC [2] 的比赛结果已经展示出网络大小和分类精确度之间有着很强的关联。在 ILSRVR 2014 上 VGG[20]方面的一项研究构建了一个多达 16 个卷积层的神经网络，在以 4 个高端 GPU 网络训练大约一个月的时间代价下，将上 5 层分类错误率降至 7.4%。

根据这些网络的结构，我们可以合理的推测：这些巨大的网络中存在严重的冗余。由于神经网络高度的非凸性，过参数化及随机初始化在克服网络训练中局部最小的负面影响中异常重要。此外，在训练阶段没有独立约束施加在每层卷积核上的事实也表明网络存在高度的冗余。

本文中我们将展示利用这种冗余性我们可以通过对卷积核稀疏分解来显著降低图像处理所需的计算量。如图 1 所示，可以利用两阶段分解来探索卷积核信道内及信道间冗余。我们首先基于内核权重的重构错误进行一次初分解，然后微调网络同时施加稀

疏约束。在微调阶段，我们通过最小化一个稀疏组内目标函数来同时最优化网络训练错误，卷积核的稀疏性及卷积基的数量。我们的模型可以取得令人惊讶的高稀疏性。我们有能力在保证准确度降低少于 1%的前提下用相关的少数基数消去超过 90%的卷积核参数。

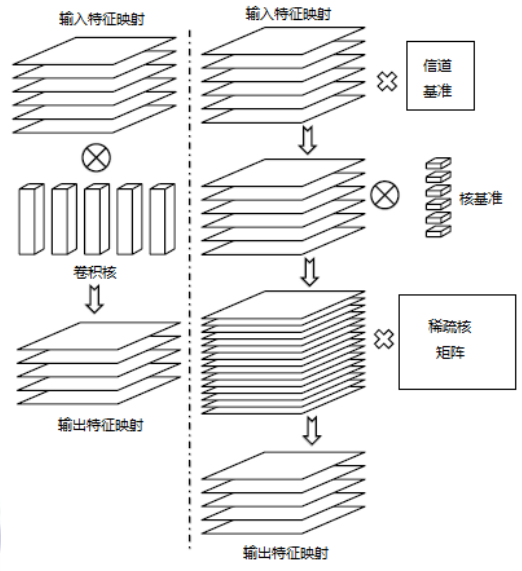


图 1：我们的稀疏卷积神经网络总览。左：传统卷及神经网络对卷积层的操作，它计算大量卷积核与输入特征映射的卷积。右：我们提出的稀疏卷积神经网络模型。我们应用两个阶段的信道和卷积核分解，获得显著（超过 90%）的稀疏核矩阵并且将卷积层操作转换为稀疏矩阵乘法。

在我们的稀疏卷积神经网络（SCNN）模型中，每个稀疏卷积层可以表现为带有一个稀疏矩阵乘法的一些卷积核。可以假设稀疏矩阵化自然能够提高计算效率。然而，计算稀疏矩阵乘法涉及到严重的开销这使得很难实际取得显著的加速。因此，我们也提出了一个有效的稀疏矩阵乘法算法。基于稀疏卷积核可以在训练后修正的事实，我们编码输入到我们程序中稀疏矩阵的结构为寄存器的索引来避免间接和不连续内存读取的可能性。我们基于 CPU 的实现方法表现出比传统稀疏矩阵库和通过原始稠密网络实现显著加速的方法大得多的效率。然而卷积网络系统被基于 GPU 方法主导，基于 CPU 系统的优势是有用的，因为它们可以被部署在没有特定 GPU 节点的商业集群中。

相对于以前工作的贡献

在下面第二节将讨论，以前的工作比如[4][12]已经使用低秩估计去表达小数量基准过滤方面的网络计算。展示在这儿的这些方法除低秩估计之外，还使用稀疏分解通过表达过滤步骤来获得额外的效率。在 6.2 节将展示，这导致只用低秩分解不能取得的效率和精确度两方面的提升。

2. 相关工作

在研究深度神经网络冗余度方面，已经有一些尝试。Denil et al. [3] 利用低秩矩阵因式分解降低了普通神经网络参数数量。他们减少了基于 MNIST 数据库的多层神经网络 95%的参数。JaderBerg et al. [12] 和 Denton et al. [4] 都使用张量低秩扩张技术来加速卷积神经网络。Jaderberg et al. [12] 在一个训练在场景字符分类数据集上的 4 层

卷积神经网络上以精确度损失低于 1%的前提下获得 4.5 倍加速。Denton et al. [4] 在训练在 ILSVRC 数据集上的卷积神经网络的前两个卷积层上取得了 2 倍加速。显而易见，[12] 和 [4] 仅仅在相对大的卷积核上表现出加速。他们都没有在被广阔应用在最先进的卷积神经网络模型上小至 3×3 的卷积核上展示他们的方法。

也有一些工作尝试从其他方面优化卷积神经网络的速度。Vanhoucke et al. [22] 研究基于 CPU 的普通神经网络加速优化法。他们讨论了 SIMD 指令的使用，内存的对齐以及网络中定点数的量化。Mathieu et al. [17] 提出在傅里叶域使用快速傅里叶变换来计算卷积。他们在 Alex net 上取得了 2 倍的加速。由于快速傅里叶变换的开销，他们的方法更倾向于使用在相对较大的卷积核中。Farabet et al. [6] 基于 FPGA 架构实现了一个大规模的卷积神经网络，它能完成嵌入式实时识别任务。

以前在稀疏矩阵计算方面的工作主要聚焦于稀疏矩阵稠密向量乘法(SpMV)问题。稀疏矩阵因为效率问题被以多种格式存储，例如 CSR[1] 和 ESB [15]。在寄存器和缓存级别，分块被应用于改善稀疏矩阵的空间局部性。为进一步减少带宽需求，包括矩阵重新排序，值和索引压缩等多种技术被提出来。我们引导[9]读者进一步的回顾。

3. 我们的方法

3.1 稀疏卷积神经网络

考虑在$\mathbb{R}^{h\times w\times m}$中输入特征映射 I，这里 h, w 和 m 分别表示高度、宽度和输入特征映射的信道数量，及在$\mathbb{R}^{s\times s\times m\times n}$中的卷积核 $\mathbf{K}$，这里 s 是卷积核的大小，n 是输出信道的数量。我们假设计算卷积时没有填充 0 并且以 1 为步幅。然后，卷积层的输出特征映射 $\mathbf{O}\in\mathbb{R}^{(h-s+1)\times(w-s+1)\times n}=\mathbf{K}*\mathbf{I}$ 被给出：

$$\mathbf{O}(\mathrm{y},\mathrm{x},\mathrm{j})=\sum_{i=1}^{m}\sum_{u,v=1}^{s}\mathbf{K}(u,v,i,j)\mathbf{I}(y+u-1,x+v-1,i) \tag{1}$$

我们的目标是用基于稀疏矩阵乘法的快速稀疏版本代替公式(1)中计算代价昂贵的卷积操作 $\mathbf{O}=\mathbf{K}*\mathbf{I}$。

为达到这个目的，我们首先使用一个矩阵$\mathbf{P}\in\mathbb{R}^{m\times m}$转换张量 $\mathbf{I}$ 到$\mathbf{J}\in\mathbb{R}^{h\times w\times m}$，卷积核 $\mathbf{K}$ 到$\mathbf{R}\in\mathbb{R}^{s\times s\times m\times n}$得到$\mathbf{O}\approx\mathbf{R}*\mathbf{J}$，这里

$$\mathbf{K}(u,v,i,j)\approx\sum_{k=1}^{m}\mathbf{R}(u,v,k,j)\mathbf{P}(k,i)$$

$$\mathbf{J}(y,x,i)=\sum_{k=1}^{m}\mathbf{P}(i,k)\mathbf{I}(y,x,k) \tag{2}$$

下一步，对于每个信道$i=1,\dots,m$，我们分解张量$\mathbf{R}(\cdot,\cdot,\mathrm{i},\cdot)\in\mathbb{R}^{s\times s\times n}$为矩阵$\mathcal{S}_i\in\mathbb{R}^{q_i\times n}$和张量$\mathcal{Q}_i\in\mathbb{R}^{s\times s\times q_i}$的乘积，这里$q_i$是基数的数量：

$$\mathbf{R}(u,v,i,j) \approx \sum_{k=1}^{q_i} \mathcal{S}_i(k,j)\,\mathcal{Q}_i(u,v,k)$$

$$\mathcal{T}_i(y,x,k) = \sum_{u,v=1}^{s} \mathcal{Q}_i(u,v,k)\mathbf{J}(y+u-1, x+v-1, i). \tag{3}$$

所以：

$$\mathbf{O}(y,x,j) \approx \sum_{i=1}^{m}\sum_{k=1}^{q_i} \mathcal{S}_i(k,j)\mathcal{T}_i(y,x,k) \tag{4}$$

注意如果我们通过合并前两维，然后沿着维度q_i将$\mathcal{S}_i$和$\mathcal{T}_i$连接起来，我们可以将张量 **O** 和$\mathcal{T}_i$表示为矩阵形式，公式(4)就可以通过一个简单的矩阵乘法实现。

这里，我们应该寻找矩阵$P, \mathcal{Q}_i$和$\mathcal{S}_i, i = 1, \ldots, m$，从而$q_i$远小于$s^2$，矩阵$\mathcal{S}_i$有大量的零元素和列，同时我们新的稀疏卷积核 **R** 提供接近于从原始核 **K** 得到的输出。

3.2 计算复杂度

我们通过测量乘法的次数来分析我们方法的理论复杂度。原始卷积所需要的乘法次数如下：

$$mns^2(h-s+1)(w-s+1) \tag{5}$$

我们的方法通过稀疏化卷积核降低了复杂度，但引入了两个矩阵分解的开销：

$$\left(\gamma mn + \sum_{i=1}^{n} q_i\right)\mathrm{s}^2(h-s+1)(w-s+1) + m^2hw \tag{6}$$

这里γ是稀疏矩阵非零项的比率。当(1)q_i的平均值远小于γm，(2) m 远小于$\gamma n s^2$时分解开销小。

3.3 学习参数

稀疏卷积神经网络的参数通过两种方式学习：初分解和微调。如果我们沿着分解维将卷积核从张量转换为矩阵，公式(2)和公式(3)的因式分解都可以看作为矩阵因式分解问题。对于初始化来说，稀疏矩阵分解算法是一个直观的选择。然而，关于目前的稀疏字典学习算法有一些值得关注的地方：(a)它们是非凸的因此不能取得一个全局最优解；(b)学习过程精确度和稀疏度之间的折中，因此不能提供精确的分解。由于这些原因，我们选择下列分解方法并且在 6.4 节比较它们的效能：

- 以 **P**，$\mathcal{Q}_i$作为基准，使用[16]中稀疏字典学习算法分解 **K** 和 **R**
- 以 **P**，$\mathcal{Q}_i$作为主成分，使用主成分分析(PCA)分解 **K** 和 **R**
- 初始化 **P**，$\mathcal{Q}_i$为单位矩阵，保持 **K** 和 **R** 不变，以这种方式，稀疏性仅仅靠微调取得。

作为初分解，上述方法仅仅能获得非常有限的稀疏性，但是提供低或者零重构误差的非常有意义的起点。进一步的微调是获得高度稀疏网络的关键步骤。在微调步骤，继续训练整个网络的同时，我们引入了在网络参数上的稀疏约束。微调能增加稀疏性

的主要原因是训练原始网络时没有施加任何稀疏约束。因为网络为我们所知为避免对于初始值敏感是过参数化的，它应该在不损失分类精确度情况下有一些修改的自由。初始化步骤仅仅能基于重构误差获得稀疏性，然而微调通过引入网络亏损作为直接误差测量能够充分的开发我们模型的稀疏性潜能。我们的实验通过展示由于微调显著性增加稀疏性证明了这个观点。

正式的，我们在微调步骤最小化下列目标函数：

$$\underset{P,\mathcal{Q}_i,\mathcal{S}_i}{\text{minimize}}\mathbf{L}_{net}+\lambda_1\sum_{i=1}^{m}||\mathcal{S}_i||_1+\lambda_2\sum_{i=1}^{m}\sum_{j=1}^{q_i}||\mathcal{S}_i(j,\cdot)||_2$$

$$\text{s.t. } ||\mathbf{P}(\cdot,\mathrm{j})||_2\le 1, j=1,\dots,m$$

$$||\mathcal{Q}_i(\cdot,\cdot,k)||_2\le 1, i=1,\dots,m, k=1,\dots q_i \quad (7)$$

这里$\mathbf{L}_{net}$网络[14]输出层的逻辑损失函数，$||\cdot||_1$和$||\cdot||_2$表示一个矩阵的元素及l_1和l_2标准。公式(7)第二项对$\mathcal{S}_i$每一个元素引入了套索约束，第三项以组套索公式化的精神作为组变量对待$\mathcal{S}_i$的每一行。需要计算的$\mathcal{Q}_i$每一列的效率等于$\mathcal{S}_i$相应非零行的数量。通过这种方式，我们能进一步减少在特征映射上卷积$\mathcal{Q}_i$的开销。

3.4 我们方法与低秩分解法比较

像[12][4]，我们的模型使用低秩分解，但是通过在卷积层中鼓励用于表达过滤去进一步降低冗余性的权重稀疏性进而超越前人工作。我们通过一个l_1标准和组套索惩罚组合同时引入稀疏约束和低秩约束。

考虑分解矩阵$\mathbf{M}\in\mathbb{R}^{m\times n}$为$\mathbf{S}\in\mathbb{R}^{m\times n}$和$\mathbf{P}\in\mathbb{R}^{\mathrm{n}\times n}$的乘法。使用稀疏分解，提升速度与 S 中非零元素占比成比例，同时复杂度降低程度与非零列占比成比例。稀疏约束可以把 S 中特定条目作为目标，然而低秩分解必须消除整列。这就使得我们的方法更精确的以冗余度为目标成为可能。6.2 节中结果将展示，使用乘法算法可以导致显著的性能提升。

4. 稀疏矩阵乘法算法

当稀疏惩罚能更精确的以冗余度为目标，性能效益只能被招致的开销不至于覆盖稀疏矩阵好处的稀疏矩阵乘法算法所实现。在这一节，我们展示乘法可以被实现得更有效率。

4.1 动机

为了避免零值的存储和计算，稀疏矩阵中非零元素根据它们的位置索引被典型地连续存储在一些特定的结构中。这就导致在遍历矩阵时，只能间接跳转读取内存，这比稠密情况下连续的内存读取要慢得多。另外，这种不规则的输入矩阵模式也很难充分利用单指令多数据(SIMD)微架构的能力，而

这在稠密矩阵算法中是个提升性能的关键方法。

我们为执行稀疏卷积核提出一个高效的，稀疏-稠密矩阵乘法算法。我们的主意基于下列两个关键的观察：

- 一旦我们的网络被完全训练，卷积核是常数而输入特征映射随着输入图像变化。因此，非零元素的位置是已知的并且在编译后的乘法代码中能够被直接编码。
- 仅仅卷积核被当作稀疏矩阵对待。输入特征映射为一些层处理温和稀疏，但是将被当作稠密矩阵对待。

尽管我们期待这种方法能够被拓展到 GPU 架构上，我们在带有高级矢量拓展(AVX)的 x86_64 微架构 CPU 上实现了我们的方法，在 2011 年之后 Intel 和 AMD 的 CPU 上都具有这种架构。我们的方法基于开源的稠密线性代数库 OpenBLAS [24]。以下部分我们先简要介绍 OpenBLAS 的矩阵相乘算法，然后介绍我们的稀疏矩阵方法。

4.2 OpenBLAS 稠密矩阵乘法

设计一个有效的矩阵乘法算法有两个主要考虑因素：(a)利用 SIMD 指令从而实现更高的计算吞吐量；(b)最大化利用缓存从而减少内存读取延迟。在最新的 OpenBLAS 库中，矩阵乘法被用 8 个浮点数能够被存储在一个 256 位 AVX 寄存器中的 AVX 指令集实现。也就是说，8 对浮点数能够在每个 CPU 核的一个时钟周期内被同时乘及加。为了最大化的减少内存延迟，输入矩阵首先被分成能够适应 CPU L2 缓存的块。然后一个输入矩阵的每块以下列方式与其他输入矩阵的每块相乘：一个块被分成 8 元素宽的行条，其他块被分成 8 元素宽的列条。然后每两个条相乘生成一个可以被存储在 8 个 AVX 寄存器中的8×8微方形。图 2 给出 OpenBLAS 矩阵相乘算法的一个图形描述。

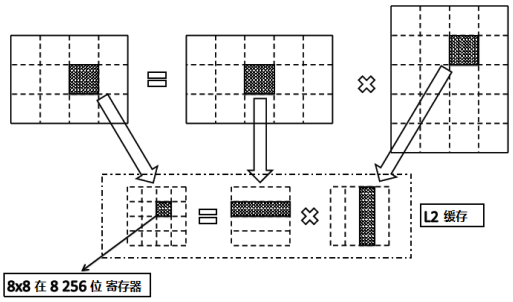


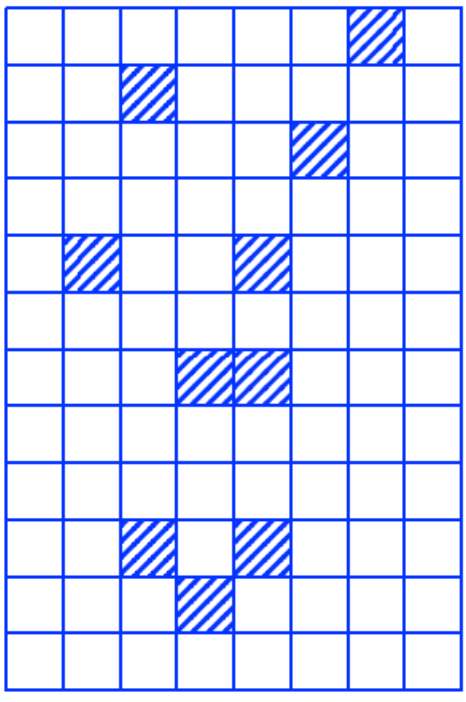
图 2：OpenBLAS 中矩阵相乘算法。输入矩阵首先被划分成能够适应 L2 缓存的块，然后每个块被划分成 8 个元素的条，在计算时，两个块的乘法输出被放在 8 个 AVX 寄存器中。

4.3 稀疏矩阵乘法

我们聚焦于稀疏-稠密矩阵乘法问题$\mathbf{C} = \mathbf{A} \times \mathbf{B}$。$\mathbf{A} \in \mathbb{R}^{m \times k}$是一个稠密矩阵，而$\mathbf{B} \in \mathbb{R}^{k \times n}$是一个固定的稀疏矩阵。**A** 和 **B** 被以上面讨论的相同方式划分成块和条。现在我们考虑一个行条和一个列条的乘法$\bar{\mathbf{C}} = \bar{\mathbf{A}} \times \bar{\mathbf{B}}, \bar{\mathbf{A}} \in \mathbb{R}^{8 \times k}, \bar{\mathbf{B}} \in \mathbb{R}^{k \times 8}, \bar{\mathbf{C}} \in \mathbb{R}^{8 \times 8}$。对任何一个矩阵 M，令$m_{i,*}$为 M 的第$i$行，$m_{*,j}$为 M 的第$j$列，矩阵乘法可以表示为：

$$\bar{c}_{*,j} = \sum_{i=1}^{k} \bar{a}_{*,j}\bar{b}_{i,j}, 1 \le j \le 8 \tag{8}$$

上式中每个$\bar{c}_{*,j}$和$\bar{a}_{*,j}$都被放在一个AVX矢量中。在我们的情况下因为$\bar{\mathbf{B}}$是稀疏的，我们需要知道非零元素的位置然后跳过零元素。为了避免间接内存读取，我们提出把$\bar{\mathbf{B}}$的结构编码到程序中。对每个非零值$\bar{b}_{i,j}$，i表明$\bar{a}_{*,i}$去与之相乘j表明$\bar{c}_{*,j}$去储存。因为它们都相当于单个AVX寄存器，我们在我们的程序中能简单的编码i和j成寄存器的索引。图 3 展示了一个我们的方法如何从给定稀疏矩阵生成代码的小例子。



(a)一个稀疏矩阵 **B** 例子。阴影部分方块代表非零元素空白方块代表零元素。

输入：

A: 8×12稠密矩阵

B: 12×8稀疏矩阵

输出：

$\mathbf{C} = \mathbf{A} \times \mathbf{B}$

操作：

$c_7 += a_1 \times b_{1,7}$

$c_3 += a_2 \times b_{2,3}$

$c_6 += a_3 \times b_{3,6}$

$c_2 += a_5 \times b_{5,2}$

$c_5 += a_5 \times b_{5,5}$

$c_4 += a_7 \times b_{7,4}$

$c_5 += a_7 \times b_{7,5}$

$c_3 += a_{10} \times b_{10,3}$

$C_5 += a_{10} \times b_{10,5}$

$c_4 += a_{11} \times b_{11,4}$

(b)为计算$\mathbf{C} = \mathbf{A} \times \mathbf{B}$生成的伪代码。$c_i$是 **C** 的第$i$列，$a_j$是 **A** 的第$j$列。$b_{i,j}$是 **B** 的第$j$行第$i$列的元素。

图 3: 描述我们的算法计算稠密矩阵和稀疏矩阵乘法生成代码的一个例子。

5. 物体检测的应用

我们现在把稀疏卷积神经网络(SCNN)应用到物体检测问题。Cirshick et al. [8] 首先提出使用卷积神经网络来解决物体检测问题。他们把通过选择搜索(Selective Search)方法生成的候选窗弯曲到固定大小，使用卷积神经网络生成高度区别特性，使用带有高负挖掘的修改线性支持向量机来训练物体检测模型。He et al. [10] 通过使用一个空间金字塔池(SPP)结构显著得提

升速度，在这种结构中，卷积层在整个图像上仅仅需要执行一次(单标度)或者多次(多标度)并且只有全连接层执行在每个窗口上。由于每个图像上有大量的候选窗(~2000)，[10]中全连接层总的运行时间相当或者超过卷积层之一。

我们应用稀疏卷积神经网络加速 SPP 模型中的卷积层。尽管我们的方法能够大大提升卷积层的速度，但由于全连接层的相对耗时，总运行时间不会减少太多。这部分，我们提出两种方案来降低全连接层的复杂度从而取得总体较高效率。

首先，我们提议网络采用类似于[7]的级联方案。因为上一卷积层的输出已经高度区别化，直接在其上应用一个线性支持向量机(SVM)分类器就可以取得良好的效果。因此，我们使用上一卷积层作为我们级联模型的第一阶段去修剪大量的候选窗口，然后使用最终全连接层的输出作为第二阶段分类器来生成最后的检测结果。第一阶段检测阈值决定是效率和精确度的折中。低的阈值保持高的记忆以至于总体精确度不受影响然而高的阈值移除了取得更高效率的更多候选者。在我们的情况中，我们发现阈值相当精度等于 0.5 是一个平衡的折中。以前在级联模型的工作仅仅被应用在单类检测中。多类检测，通常来说会取得更少的效益，因为随着类数量的增加候选者数量急剧增加。然而在我们的模型中，我们在多类情况下仍然能够移除相当大部分的候选者。

第二，我们以我们对卷积层做过的相似的方式分解全连接层。全连接层的操作可以表示为神经元函数之后的矩阵-向量乘法。我们把权重矩阵分解为一个稀疏矩阵和一个稠密矩阵的乘积$\mathbf{M} \approx \mathbf{PS}$，这里$\mathbf{M} \in \mathbb{R}^{m\times n}, \mathbf{P} \in \mathbb{R}^{m\times k}$, $\mathbf{S} \in \mathbb{R}^{k\times n}$。如果$m > n$我们选择在 $\mathbf{P}$ 上强加稀疏性，否则在 $\mathbf{S}$ 上加稀疏性。我们同时施加l_1标准和组套索代价函数以至k也减少。

6. 实验结果

6.1 设置

我们在 ImageNet LSVRC 2012 [2]数据集上训练我们的模型。我们从一个预训练的 Caffe 参考卷积神经网络模型开始，这个模型几乎和[14]中描述的相同。模型包含 5 个卷积层和 2 个全连接层，与二次抽样层，局部标准化层，最大分摊层，修正线性单位层和降落层交织。第一个卷积层有11×11相对大核和仅仅 3 个输入信道；第二个卷积层有5×5核；第三，四和五卷积层有非常小的3×3核。核的大小差异及输入核的数量影响能够取得的稀疏度可能大小。

全部 5 个卷积层都被根据方程 7 使用动量随机梯度下降同时优化。当稀疏化网络参数时，基础学习率最初被设置为 0.001。为了使训练过程稳定，

我们引入了阈值函数设置在训练过程中小于$1e^{-4}$的参数为 0。一旦训练过程收敛，我们移除稀疏度约束但是保留阈值函数。最后，我们逐渐降低基础学习率去微调网络从而获得最好的精确度。

6.2 ILSVRC12 上的结果

表 1 展示了 ILSVRC12 上的结果。对于全部 5 个卷积层，我们获得了超过 90%的稀疏度。每一层基准平均数量和全秩分解相似，显著小于s^2(核大小的平方)。这个理论上的加速是我们稀疏卷积神经网络层运行时间和原始卷积网络的比例。表 1 的最后一列展示了实际取得的加速倍数。因为稀疏矩阵乘法的开销，能够预料到实际的性能提升不会和理论结果相匹配。

卷积层	1	2	3	4	5
核大小	11	5	3	3	3
输入信道数	3	96	256	384	384
输出信道数	96	256	384	384	256
复杂度 %	15.8	33.6	22.5	16.8	11.2
稀疏度 %	0.927	0.95	0.951	0.942	0.938
平均值 qi	29	7.91	5.23	4.32	3.95
理论加速	2.61	7.14	16.12	12.42	10.77
实际加速	2.47	4.52	6.88	5.18	3.92

表 1: 我们稀疏卷积神经网络每一层相应的稀疏度，基数平均数量，理论和实际加速。q_i是每一层基数平均数量。结果展示我们高度稀疏模型在理论和实际方面都能够产生显著的计算加速。

由于核大输入输出信道少导致的冗余度有限，相比较其他层来说卷积层 1 的加速倍数不是特别显著。因为输出信道数量比较少，卷积层 1 基数的数量尽管在后面有所减少，仍然在运行时间中占据大部分比例。这个问题也出现在以前的低秩过程[12]中。我们实验使用可分离过滤器的组合分解基准过滤器，但是基准展示出用可分离过滤器表达的太多的变量。

6.3 与仅使用低秩估计比较

为更直接的与前人工作相比较，我们也训练了一个类似于使用在[4]中模型相似的使用在[10]中模型的一个修改版本。主要差异位于一个空间金字塔层，不会影响卷积层的空间性能。表 2 展示了[4]和我们基于[10]的修改版本理论加速倍数的比较。由于上述提到的原因，我们在卷积层 1 有相似的速度提升，而在卷积层 2 却获得更大的加速倍数[4]。需要注意的是[4]不尝试加速后面的层，这可能是由于过滤器比较小(3×3)，而我们的方法有能力取得显著的加速。

卷积层	1	2	3	4	5
核大小	7	5	3	3	3
稀疏度 %	0.84	0.956	0.893	0.904	0.89
平均值 qi	21	9.06	6.76	6.86	6.98
理论加速倍数	2.62	7.06	8.03	8.78	7.29
低秩方法[4]	2.4	2.5	-	-	-

表 2: 类似于[10]的一个模型和[4]描述的模型在稀疏度，加速倍数等方面的比较。

6.4 初始化方法的比较

图 4 展示了我们采用的不同初始化方法的比较。PCA 和稀疏编码都获得了超过 90%的稀疏度，然而随机初始化展示差得多的稀疏度。值得一提的是，仅仅靠初始化所有的方法都获得非常有限的稀疏度。这些结果表明

初始化和微调都具有重要的地位。所有这些方法都显示出微调过程只有非常小的精确度损失。由于我们在稀疏化网络过程中设置的学习率高于最后快速收敛的学习率，图 4 中显示的精确度数字通常低于最后的精确度。在这三种方法中，PCA 的精确度略胜一筹。稀疏编码方法尽管在理论上感觉很好，实际很差。我们指出主要原因是它的非凸性和非精确重构。尽管 PAC 也是一个非凸性问题，但可通过 SVD 获得全局唯一最优化。

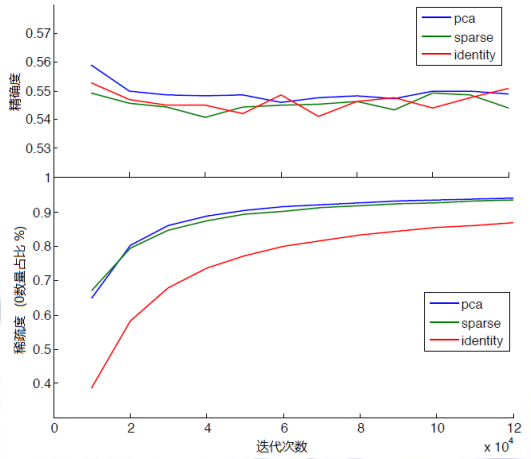


图 4：初分解方法的比较。我们在训练过程中展示我们稀疏卷积神经网络精确度和平均稀疏度的变化。

6.5 基准可视化

图 6 显示我们稀疏卷积核中非零元素的平均数量与我们在信道和每个信道中的核分解基准一致。基准依照他们 PCA 初始化的特征值分类。稀疏度和 PCA 特征值之间有高度的相关性，这就证明了 PCA 初始化的重要性。对基准进行有意义的划分，系数和几乎全变为 0。全零基准等同于可以被低秩分解估计的那些。对于相对较大的核来说，像卷积层 1 和卷积层 2，全零基准的比例完全足够取得温和级别的加速。然而，对于像卷积层 3,4,5 这样的小核来说，全零基准的比例非常有限。通过低秩分解甚至取得 2 倍的加速就要损失非常可观的精确度。如图 6 所示，我们非零基准获得的稀疏性方法的优势非常明显。

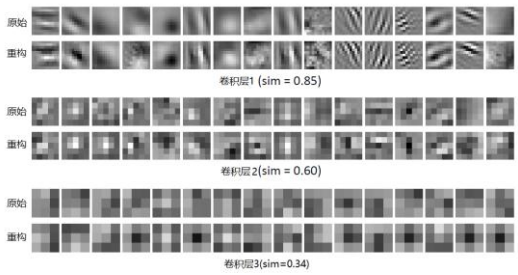


图 5：原始卷积核和我们稀疏核重构的卷积核之间的比较。这里我们展示了随机取样核卷积层 1，卷积层 2 和卷积层 3。层 4 和层 5 与层 3 非常相似。对于每一层，第一行表示原始核而第二行表示重构核。他们之间平均余弦相似度于图下给出。

为了证明微调的必要性，我们比较原始模型与图 5 从我们微调稀疏模型重构的卷积核。由于空间有限，我们仅仅展示卷积层 1，卷积层 2 和卷积层 3。我们忽略卷积层 3 到卷积层 5 中非常普遍的全零核。我们也测量经过第一次减少内核平均值后原始核重构内核的平均相似性，然后计算余弦相似度测量，它被定义为：$\mathrm{Sim}_{cos}(x,y) = \frac{x \cdot y}{\|x\|\|y\|}$。从图 5 我们可以看出从卷积层

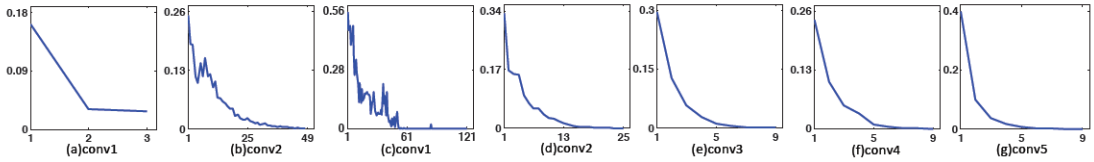


图 6：我们稀疏卷积核中非零元素平均比率与通过信道和过滤器基数分解一致。(a)和(b)展示了信道基数，(c)到(g)展示了过滤器基数。每个图的基数以他们的 PCA 初始化特征值降序排序。

1 到卷积层 3 平均相似度迅速减少。卷积层 3 的核看起来与原始核非常不同。考虑到我们模型中精确度的些许降低，这证明了我们讨论中原始网络是极度过参数化并且在不影响性能的情况下可以施加有意义的规则。因此，仅通过尝试估计网络而不是使用我们方法中的基于微调的网络损失方法很难充分开发网络的稀疏度潜能。

6.6 稀疏矩阵乘法算法评估

我们首先用一个随机生成的矩阵分析我们的稀疏稠密矩阵乘法算法的性能。我们随机生成一个1024×1024稠密矩阵和一个1024×1024稀疏矩阵，然后测量我们算法计算它们相乘运行的时间。我们的实验进行在一个 Intel i7-3930k CPU 上。为简单起见，我们计算这种情况下的单线程表现。多线程表现结果应当一致。图 7 展示了我们评估的结果。从这个图我们能够得到以下结论：(a)算术运算时间与输入矩阵的稠密度严格成比例，并且非常接近理论限制；(b)输入/输出时间随着稀疏度非线性增加。从内存加载到缓存与储存结果的时间是一个常量与稀疏度无关，从缓存加载稠密矩阵到 CPU 的时间取决于稀疏矩阵连续 8 个 0 的比例；(c)当稀疏度增加时输入/输出时间显著增加，当稠密度少于 10%时，输入/输出操作占据总运行时间的 80%；(d)注意算术运算时间和输入/输出操作时间的和显著高于总时间。这是由于 CPU 流水线策略引入的并行化。因为这个原因，算术运算和输入/输出操作同时进行，从而使得总体性能显著改善。

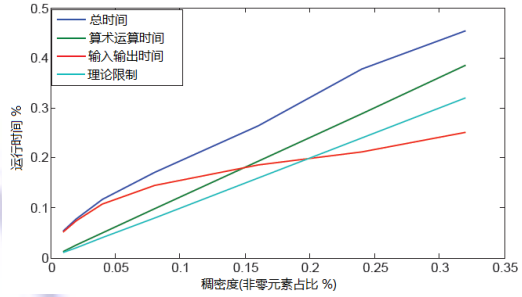


图 7：我们稀疏矩阵乘法算法的运行时间分析。横轴代表输入稀疏矩阵中非零元素的比例，纵轴是与 OpenBLAS 中稠密矩阵乘法代码相比较的相对运行时间。算术运算时间是执行乘法和加法的运行时间，输入/输出时间包括从内存加载输入矩阵到缓存，从缓存加载到 CPU 及写入结果到内存。理论时间是我们可能取得的最好时间，和

输入矩阵的稠密度一致。

6.7 运行时间分析

表 3 展示了我们代码以及每个部分比例实际速度。所有 5 层都取得了显著的速度提升。最后三层耗时主要在稀疏矩阵乘法方面，然而在前两层主要是基准卷积时间，这与理论分析一致。实际运行时间和理论运行时间之间的差距主要来自以下两个因素：(a)稀疏矩阵乘法的开销；(b)基准卷积的低效率。Caffe 通过矩阵乘法实现卷积，这对一些像我们例子中的那种过滤器很少的情况来说是低效的。我们实现了快速版本但是没有完全优化。另外，卷积及稀疏矩阵乘法的联合缓存优化将进一步提升效率。

卷积层	1	2	3	4	5
通道分解	0.06	0.14	–	–	–
基准卷积	0.78	0.41	0.21	0.3	0.44
矩阵乘法	0.16	0.45	0.79	0.7	0.56
原始方法	2.47	4.52	6.88	5.18	3.92

表 3: 与原始稠密网络的运行时间分析和比较。每一层的所有数字都用我们的方法被以这一层所有运行时间标准化。最后一行是我们的方法相对于原始方法的加速倍数。

6.8 物体检测方面的结果

我们使用表 2 的微调模型去在 PASCAL VOC2007[5]数据集上执行物体检测。我们方法相对于传统方法[10]的精确度被展示在表 4 中。我们使用公布在[10]中的代码无法复制精确度，所以我们放置我们得到的数字代替它。我们的方法比原始 SPP 模型大约差 2%，然而却得到几次更快的执行速度。我们推测比分类问题更高的精确度损失可能是因为我们的卷积层被训练在 ILSVRC 数据集上这个事实，同时只有全连接层被微调以适应[10]中的 PASCAL 数据集。因此，尽管我们用微调稀疏卷积层，网络的损失和检测问题的损失不相当。一个基于微调的完全网络应该能进一步减少我们方法的精确度损失。

	fc7(1s)	fc7bb(1s)	fc7(5s)	fc7bb(5s)
spp[10]	52.47	54.19	54.75	57.19
我们的方法	50.16	52.64	52.58	55.13

表 4: 物体检测方面我们的方法相比较[10]中方法平均精度。“bb”代表边界框回归分析，“1s”意味着 1 刻度“5s”意味着 5 刻度。我们的稀疏模型次于[10]大约 2%。

在我们的级联模型中，我们设置了第一阶段的阈值以至于每一类的精确度等于 0.05。大约 80%的候选窗为每一幅图像修剪，因此给全连接层带来了大约 5 倍的速度但几乎没有精确度损失。另外，第一和第二全连接层分别取得了 85%和 68%的稀疏度，进一步提供了超过 2 倍的速度。全连接层的运行时间因此降低到远小于卷积层的地步。

参考文献

[1] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. Templates for the solution of linear systems: building blocks for iterative methods, volume 43. Siam, 1994.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 248–255. IEEE, 2009.

[3] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al. Predicting parameters in deep learning. In Advances in Neural Information Processing Systems, pages 2148–2156, 2013.

[4] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Advances in Neural Information Processing Systems, 2014.

[5] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. International journal of computer vision, 88(2):303–338, 2010.

[6] C. Farabet, Y. LeCun, K. Kavukcuoglu, E. Culurciello, B. Martini, P. Akselrod, and S. Talay. Large-scale fpga-based convolutional networks. Machine Learning on Very Large Data Sets, 2011.

[7] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. Cascade object detection with deformable part models. In Computer vision and pattern recognition (CVPR), 2010 IEEE conference on, pages 2241–2248. IEEE, 2010.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

[9] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis, and N. Koziris. Performance evaluation of the sparse matrixvector multiplication on modern architectures. The Journal of Supercomputing, 50(1):36–77, 2009.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In Computer Vision–ECCV 2014, pages 346–361. Springer, 2014.

[11] E.-J. Im, K. Yelick, and R. Vuduc. Sparsity: Optimization framework for sparse matrix kernels. International Journal of High Performance Computing Applications, 18(1):135–158, 2004.

[12] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank

expansions. In Proc. BMVC, 2014.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

[15] X. Liu, M. Smelyanskiy, E. Chow, and P. Dubey. Efficient sparse matrix-vector multiplication on x86-based many-coreprocessors. In Proceedings of the 27th international ACM conference on International conference on supercomputing, pages 273–282. ACM, 2013.

[16] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In Proceedings of the 26th Annual International Conference on Machine Learning, pages 689–696. ACM, 2009.

[17] M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through ffts. In International Conference on Learning Representations (ICLR2014). CBLS, April 2014.

[18] R. Nishtala, R. W. Vuduc, J. W. Demmel, and K. A. Yelick. When cache blocking of sparse matrix vector multiply works and why. Applicable Algebra in Engineering, Communication and Computing, 18(3):297–311, 2007.

[19] L. Oliker, X. Li, P. Husbands, and R. Biswas. Effects of ordering strategies and programming paradigms on sparse matrix computations. Siam Review, 44(3):373–393, 2002.

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

[21] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. International journal of computer vision, 104(2):154–171, 2013.

[22] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. In Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop, 2011.

[23] J. Willcock and A. Lumsdaine. Accelerating sparse matrix computations via data compression. In Proceedings of the 20th annual international conference on Supercomputing, pages 307–316. ACM, 2006.

[24] Z. Xianyi, W. Qian, and Z. Chothia. Openblas.URL:http://xianyi.github.io/OpenBLAS,2012.