北京大学

本科生毕业论文

题目： **Caffe针对手机ARM架构的OpenMP优化**

姓　　名：杨博文

学　　号：1300011443

院　　系：物理学院

专　　业：大气与海洋科学

研究方向：深度学习应用

导师姓名：孙广宇

二〇一七年五月

Caffe针对手机ARM架构的OpenMP优化

杨博文　大气与海洋科学

导师姓名：孙广宇

摘要

物体识别、图像分类、语义处理等计算机视觉领域中的基本任务目前广泛依赖各种机器学习方法。卷积神经网络(Convolutional Neural Network, CNN)凭借其精度优势，逐渐在图像方面的任务中占据主导地位。但由于CNN所含卷积层、全连接层有着庞大的浮点数计算量，CNN模型长期以来在计算性能较弱的便携设备上难以得到广泛应用。随着ARM架构引入对浮点数更强的硬件性能提升，手机厂商开始考虑尝试利用CNN模型完成多媒体任务，以期显著提升用户体验。

各种深度学习框架为灵活快捷地实现不同结构的CNN模型提供了极大的便利。这些框架中，伯克利视觉与学习中心(Berkeley Vision and Learning Center, BVLC) 推出的开源深度学习框架Caffe 因其在PC端的优异性能被我们选中，作为手机上的首选深度学习解决方案。本文基于OpenMP并行化技术，针对Arm64-v8a 这一新一代手机支持的ARM架构CPU提出了Caffe的一个优化版本。本文提出优化版Caffe运行AlexNet [1]网络和GoogleNet [2]网络时，成功在Arm64-v8a架构的Open-Q 820开发板上分别获得了约18%和约12%的性能提升。

关键词：深度学习，OpenMP并行化，ARM架构，Caffe

Caffe Optimization for Smartphone ARM Architecture with OpenMP

Bowen Yang (Physics)
Directed by Prof. Guangyu Sun

Abstract

Machine learning methods are widely being used in solving basic objectives in computer vision such as object detection, image classification and image semantic analysis. Among numerous machine learning algorithms, convolutional neural networks became the dominant one in image processing for its splendid precision. However, implementing these CNN related models on embedded devices with relatively poorer performance was generally considered impractical, due to its high volume of floating-point computation introduced by its convolutional layers and fully-connected layers. With ARM architecture's hardware floating-point computation capabilities enhancing, nowadays smart phone manufacturers are seeking solutions for implementing CNN models on smart phones to accomplish multi-media related tasks, to significantly boost their user experience.

The emerge of deep learning frameworks made it possible to implement various CNN models with different structures with ease. Among all these frameworks, Caffe, an open-source deep learning framework developed and maintained by Berkeley Vision and Learning Center(BVLC) was chose as the subject of this paper for mobile

phone deep learning solution due to its superior performance on PC. This paper introduces an optimized version of Caffe, parallelized with OpenMP specifically for Arm64-v8a, an ARM architecture commonly seen among latest generations of smart phone chips. The optimization proposed by this paper managed to boost the performance of AlexNet [1] and GoogleNet [2] on Arm64-v8a based Open-Q 820 development kit by approximately 18% and 12% respectively.

Keywords: Deep Learning, OpenMP Parallelism, ARM architecture, Caffe

目录

第一章　背景

近些年来，深度学习技术得到了长足的发展，为图像分类、机器翻译、语音识别等多媒体问题提供了高精度高性能的解决方案。自高精度的图像分类CNN网络AlexNet [1]问世以来，卷积神经网络凭借其精度优势，逐渐成为物体识别、图像分类、图像语义分析等图像、视频处理领域问题的主流选择。在AlexNet [1]被提出之后，学界发展了大量精度更高、结构更复杂的卷积神经网络模型，如R-CNN [3]、Fast R-CNN [4]、Faster R-CNN [5]、YOLO [6]。基于卷积神经网络的图像处理技术在业界也得到了广泛应用，并渗透到了每个人的日常生活中。

如，在自动驾驶领域，Google基于图像物体检测技术启动了智能自动驾驶汽车计划 [7]，截止至2012年，其推出的Waymo自动驾驶系统已经装上Lexus RX450h试验车已经完成了30万英里的无人驾驶旅程。2017 年，Waymo 项目已经推出了其第一款基于量产车型的自动驾驶汽车，并且在亚利桑那州开始召集体验用户为他们提供回馈；

在医疗方面 [8]，深度学习技术也可以通过预训练的神经网络对CT影像、组织切片影像等图像进行分析，为医护人员做出诊断提供可靠的参考。日前即将落幕的Kaggle大数据竞赛Data Science Bowl 2017 大赛 [9]的主题即为根据CT图像等数据集准确预测肺癌。

标榜“智能”的智能手机厂商们也在利用深度学习技术，为手机用户开发更加智能的应用、提供更加智能的用户体验。从较为简单基础的相机识别人脸、自动聚焦，到复杂的云相册服务（识别并分类相册中的人脸，自动根据照片中含有“谁”来进行整理归类；识别各照片中的物体并标记，以便用户检索相册中的特定照片，等）。

随着更新的卷积神经网络模型在深度、广度上进一步变复杂，这类任务的计算量对服务器的压力也进一步增大。一些智能手机厂商为了减少服务器的负担，决定采取直接在手机上运行卷积神经网络模型的做法。如最近更新的iOS10系统中引入的智能相册就应用了运行在手机本地的CNN 模型，苹果公司近期也对iOS开发者开放了神经网络、深度学习相关的应用程序接口（API），让开发者能够调用这些现有函数开发更加智能的应用软件。

然而这种闭源的深度学习接口无法满足很多智能手机的开发需求：首先，它是闭源的，开发者无法根据需求调整，只能调用现成的函数；其次，这套接口是苹果公司开发的，只能用于iOS系统，安卓开发者、Windows Phone开发者只能望洋兴叹；第三，学界的CNN模型正不断推陈出新，很多近期较为复杂的CNN模型的网络结构、损失函数等已经无法用常规的CNN操作（如卷积层、全连接层、Max Pooling层、ReLU激活函数等）描述。如，R-FCN [10]模型中引入了非常规的Pooling操作（ROI Pooling与PSROI Pooling）；YOLO [6]模型中为了统一物体识别与分类，引入了之前从未被现成框架实现过的损失函数。可以预见的是，CNN模型的复杂度会随着学界的对精度的更高要求而进一步提升，这类无法用常规操作描述的模型也将变得越来越多。

在手机上，尤其是安卓手机上进行深度学习开发，目前可替代苹果公司神经网络API的选项中，最容易想到的当然是将各种开源的神经网络框架，如Caffe、TensorFlow、Torch等移植到手机上。这些框架一般允许用户直接通过预定格式的文本文件去定义较为基本的网络结构，并规定训练、测试、实际使用时所执行的动作。统计学习的数学研究者甚至无需掌握一定的编程技巧即可在这些框架上测试自己的研究想法。由于其开源性质，一些含有自定义复杂操作而无法用原版框架去描述的模型，也可以通过修改源码的方式实现。

在业界诸多开源深度框架中，Caffe比较适合移植到手机上供开发者进行深度学习相关开发：首先，Caffe的主体代码构成是C++，不使用NVIDIA CUDA进行GPU计算的话，其依赖库也均为开源项目。也就是说，理论上讲Caffe可以移植到任何支持C++编译工具链的平台上。安卓系统的主要开发语言虽然是Java，但拥有完整可靠的Native Development Kit工具链，可以交叉编译C++等底层语

言到指定手机上，并封装native接口供Java语言写成的安卓应用调用。相比之下，以Theano等框架为代表的深度学习框架对外开放的接口主要是Python等高级脚本语言，缺乏底层支持，很难移植到没有原生Python 支持的大部分手机上；其次，Caffe得益于其靠近底层的实现，虽然没有官方支持ARM手机架构，但在PC端主流CPU架构（x86-32，x86-64）上的性能表现比较突出；第三，Caffe因为设计上考虑了可扩展性，自定义的操作可以参考文档，规范化地实现。Caffe在底层封装的大量接口可以让开发者非常方便地直接访问到底层数据，加入自定义的操作抽象成层非常方便。如近期因高精度、高性能瞩目的CNN物体识别论文SSD [11]、R-FCN [10]等，其作者所在实验室放出的项目源码即是自定义版的深度学习框架Caffe。

综上，这篇论文选择Caffe作为基础，将其移植到ARM架构的手机芯片上，并在它的基础上进行优化，以提供一个高性能、可扩展的深度学习套件。

第二章　神经网络

人工神经网络ANN（Artificial Neural Network）是一种有监督学习的机器学习模型。神经网络由人工神经单元连接成网络，每个神经单元接收与其相连的各个单元的输入信号，乘以对应的连接权重并累加。这个值经过激活函数（通常是非线性函数，如S型Sigmoid函数、ReLU函数等）之后成为该单元的输出信号。神经元之间的连接权重即是需要拟合的参数。目前的神经网络从小到大，有几千至几百万之众，连接参数可以达到上千万甚至上亿的量级。

如图2.1所示是一个人工神经元。该神经元的输出为$\varphi(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}})$，其中$\varphi(x)$为激活函数。

通过这种线性变换（权重与输入点积）与非线性变换（激活函数）结合的方式，高维空间中线性不可分的输入特征被变换到了易于分割的空间中，从而完成对输入的分类、识别等信息提取过程。

应用中的人工神经网络会把人工神经单元组织成层，每层之间互相连接，信号从（一般经过预处理之后的）输入层开始，逐层地传递到输出层，得到有用的信息。最简单的神经网络可由三层构成：第一层为输入层，第二层为隐层，第三层为输出层。数学上已经得到证明的是[12]，随着隐层神经元数量的增多，一个三层（含有一层隐藏层，一层输入层，一层输出层）的神经网络理论上可以任意逼近一个未知的连续函数。如图2.2是一个这样的前馈神经网络。

但神经元增多同时会带来训练上的难度，故目前大部分高精度的CNN模型选择增多网络层数（即“深度学习”概念中的“深度”）来提高拟合的逼近程度。如ILSVRC的ImageNet图片分类物体探测挑战[13]中，2017 年的精度冠军得主MSRA 使用了一个多达128层的神经网络。

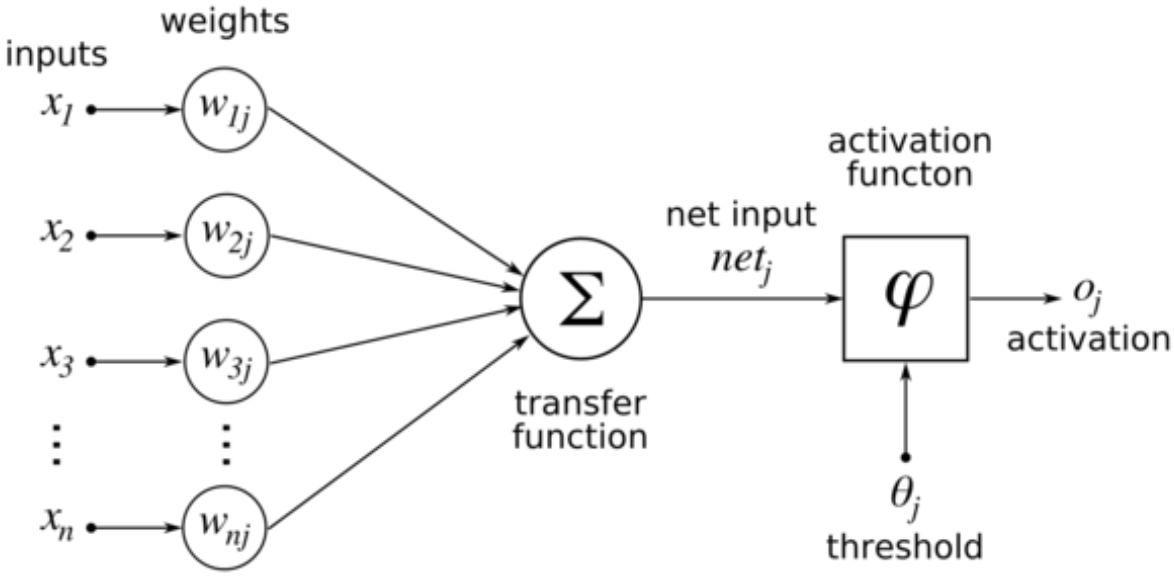

图 2.1: 人工神经元示意图

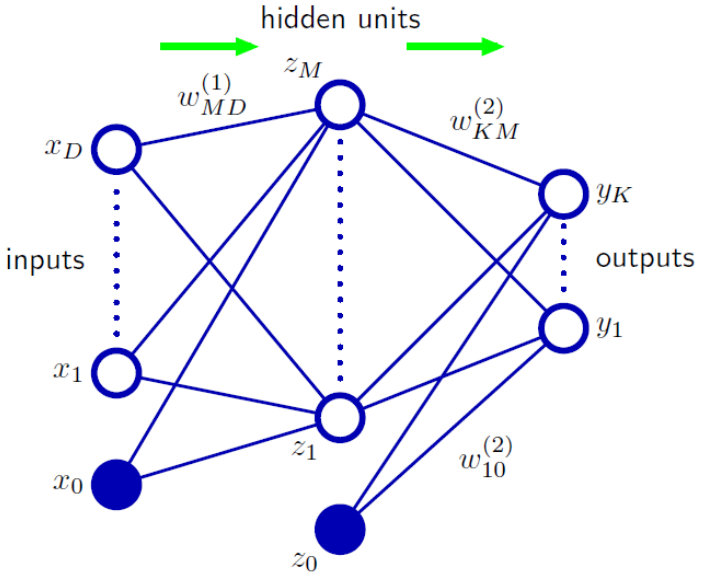

图 2.2: 一个三层的人工神经网路

为了解决特定的问题，神经网络需要根据可供拟合的已有数据集对其大量参数调整，即“训练”过程。根据实际问题，我们规定一定的损失函数。这些损失函数是参数的函数，表征模型的性能如何，训练的目的就是以训练数据集为根据，最小化损失函数。类比最小二乘法线性拟合的过程，各个样本即训练数据集，线性函数的两个系数即需要训练的参数，损失函数即误差的平方和。

和最小二乘等简单的模型不同，神经网络的损失函数最值通常无法表达为参数和训练集的解析表达式形式。故损失函数的最小化要通过数值方法，在训练集上反复迭代近似到其极小值。实用的方式是梯度下降法调整参数，逐步迭代直至损失函数收敛到极小值。当激活函数可导时，易证[14]损失函数对参数的数值梯度恰好可以写成由输出向输入端反向传播信号的形式，故训练过程为反向传播。

模型训练效果达标后，参数固定下来，网络即可使用。显然，反向传播训练网络的过程会在模型得到应用之前在提供商的服务器上完成，使用时网络只需根据确定的参数对输入做前向传播即可。故，本文侧重于优化Caffe的前向计算过程。

第三章　卷积神经网络

卷积神经网络是一种特殊的人工神经网络，在图像处理方面有广泛应用。大部分应用场景下，图像对神经网络来说是非常高维的输入。如，一张1920*1080的RGB三通道图片，组织成列向量输入给网络的话就有622万维。如果使用全连接的神经网络，每相邻层之间的神经网络均互相相连，则权重参数的个数将随神经元数量平方增长。对于图像这种高维输入，权重参数过多，巨大的计算量使训练和应用都变得不现实。

卷积神经网络利用了图像本身的局部性（相距较近像素的相关性强），提出了新的网络连接方式——每一层的神经元只和上一层的局部相连，一般格子取3x3、5x5、7x7等尺寸，称为卷积核。

则，前向计算相当于以卷积核为滤波器，对上一层提供的输入（称特征图，feature map）作二维离散卷积操作。

基于如下假设：如果某个滤波器接受一片二维的输入特征图区域后卷积出了有用的信息，那么这个滤波器一定也能在别的区域得到有用信息，可以在滑动过程中使用同一个滤波器。遍历过程中卷积核的权重保持不变，实现权重共享。

卷积神经网络靠局部连接和权重共享，相比全连接网络计算量大幅减少，而且因为利用了图像本身的局部性，图像内部的物体发生平移等变换不会引起网络的性能下降，故在计算机视觉领域有优秀的性能。

卷积的计算过程见图3.1。

3.1　卷积层

卷积层是CNN的主要组成层，其原理如上述。计算过程的表达式可形式化表

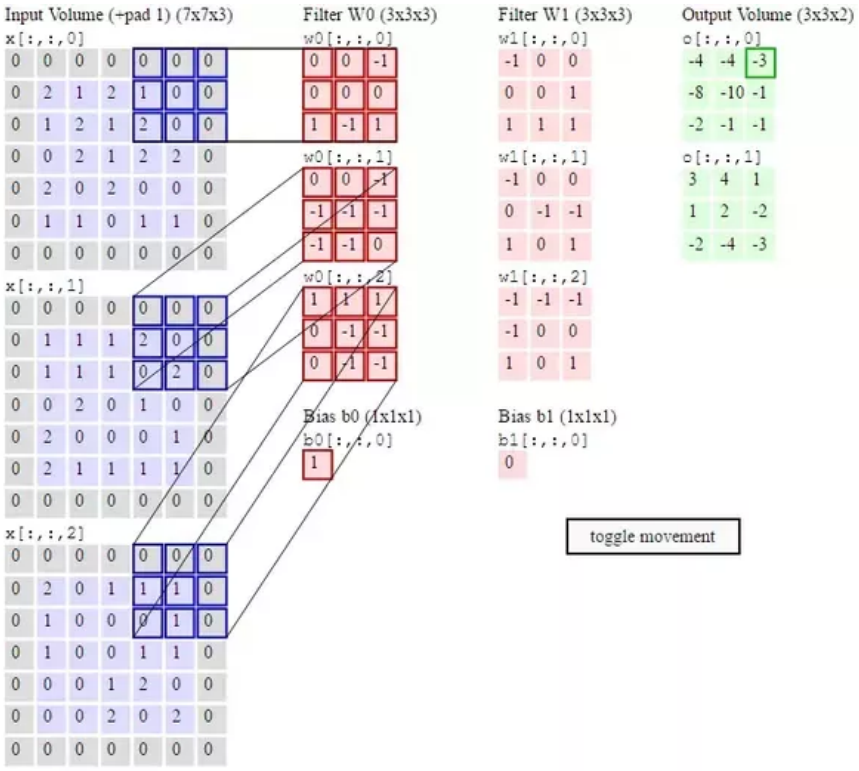

图 3.1: 卷积层的计算过程

示为 [15]

$Out[m][r][c] = \sum_{n=0}^{N} \sum_{i=0}^{K_1} \sum_{j=0}^{K_2} W[m][n][i][j] * In[n][S_1 * r + i][S2 * c + j]$

描述该层如何卷积需要如下几个预定的超参数：

深度（Depth） 又称频道数（Channels），控制该层每个神经元与多少个上一层的神经元区域相连。如，CNN输入层接受RGB三通道图像时，输入层的Depth=3；

Kernel Size 规定了连接区域的大小，即卷积滤波器的大小。常用的区域大小有3x3，5x5，7x7。AlexNet [1]原论文的网络使用3x3的卷积核；

步长（Stride） 规定了卷积核在输入特征图上每次滑动移过多少像素，常用值为1、2，AlexNet取1；

填充（Padding） 规定卷积核在滑动到图像边缘，滤波器部分格子在图像以外时，采用什么值在图像边界外扩充一段，使卷积计算合法。AlexNet取图片边缘外为零。显然，这几个超参数和卷积层神经元数量不是互相独立的。卷积层神经元个数为$(W - K + 2P)/S + 1$。

由于卷积层在维数较高的空间上操作，计算量也很大。2017年的论文 [16]显示AlexNet [1]92%的浮点数集中在卷积层，故该层是我们的优化重点。

3.2 ReLU激活函数

ReLU [18]函数的定义：

$$f(x) = \begin{cases} x & : \quad x \leq 0 \\ ax & : \quad x < 0 \end{cases} \tag{3.1}$$

AlexNet [1]论文中指出，使用不同的激活函数时，损失函数收敛到极小值的速度不同，见图3.2。对卷积神经网络，ReLU激活函数值没有上界，收敛较快，故得到了包括AlexNet [1]在内的许多CNN模型的应用。

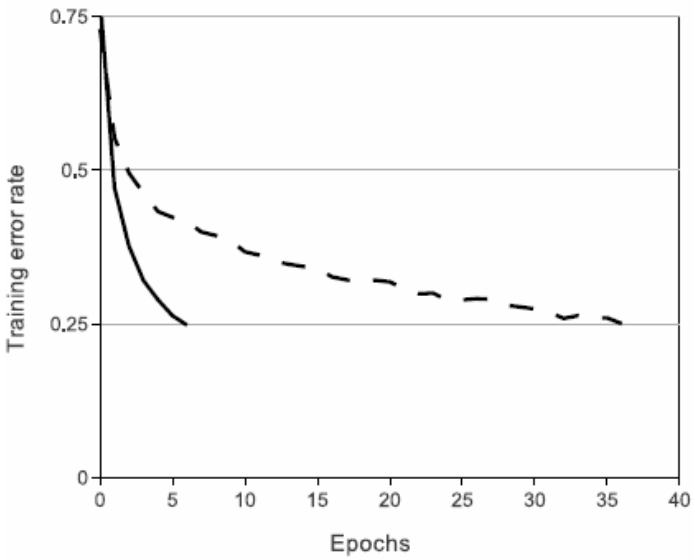

图 3.2: 不同激活函数的收敛速度。本图反应一个四层卷积神经网络在CIFAR-10[17]数据集上的训练收敛过程，横轴为训练集所有图片的训练迭代世代数，纵轴为模型的错误率。实线代表激活函数为ReLU，虚线代表激活函数为$tanh$。由图可见ReLU的收敛速度快6倍左右。

3.3 池化层

池化层（Pooling Layer）对输入进行下采样操作，以达到对输入的降维，同时可以提高网络对输入图像的空间不变性。最常用的池化方法是Max Pooling（每个下采样窗口保留值最大的）和Average Pooling （每个下采样窗口取包含特征图像素的平均值）

3.4 全连接层

全连接层接受前面卷积层的输出，最后得到图像的类别，和普通全连接神经网络的分类问题一样。AlexNet [1]使用5层卷积层接3层全连接层的网络。由于全连接层的输入是已经经过卷积层和池化层充分降维的数据，引入全连接层不会增加过多的计算量。

第四章　Caffe的移植

4.1　Caffe

Caffe是一个开源的深度学习框架，主体由贾扬清在伯克利分校读PhD期间完成，目前在开源社区GitHub [19]上由二百多名开发者维护。

本文选择Caffe进行移植并优化的一个原因是其灵活性。Caffe通过Prototxt协议来定义神经网络结构，只需写好配置文件就能实现不同的卷积神经网络。其主体代码由C++写成，也有封装好的Python和MATLAB 接口，底层高层接口均比较完备，不仅适合灵活开发比较常规的网络，也可以很方便地加入自定义的操作，开发原创的CNN模型。在Caffe推出后一年内，就有超过1000个fork分支出去的自定义Caffe版本 [20]。

本文选择Caffe进行移植的另外一个原因是其性能。在Caffe刚推出时，它在NVIDIA K40 GPU平台上就可以达到每天6000万张图片的处理速度（每张图片前向传播需要1毫秒，训练迭代平均需要4毫秒），其在CPU上的速度同样惊人，而且优化潜力巨大。Intel曾针对Caffe推出了适用于Intel x86架构的CPU版Caffe，相比原版Caffe在纯CPU模式下可以达到13.5倍的速度提升 [21]。

这主要得益于其底层运算操作的设计。在卷积神经网络的特殊语境下，卷积的计算过程伪代码逻辑上是一个六重循环（逻辑上六层循环指标w, h, x, y, m, d分别为：输入特征图长，输入特征图宽，卷积核长，卷积核宽，输出频道数，输入频道数）。Caffe原作者贾扬清在文档 [22]中将伪码实现如下：

```
for w in 1..W
    for h in 1..H
```

```
        for x in 1..K
            for y in 1..K
                for m in 1..M
                    for d in 1..D
                        output(w, h, m) += input(w+x, h+y, d) * filter(m, x,
y, d)
                    end
                end
            end
        end
    end
end
```

如果真的按六重循环嵌套的方式实现，优化将根本无从谈起，因为嵌套循环实在太深，无法对任意的输入维度保证良好的性能。而且，由于滤波器需要在输入特征图上滑动，这种实现的空间局部性很差，无法有效利用缓存。

Caffe将难以优化的卷积操作简化为已经高度优化的问题——矩阵相乘。自然科学计算和计算机科学中的很多问题都可以用线性代数表达，基础线性代数库（Basic Linear Algebra Subprograms, BLAS）已经被高度优化，具有良好性能。如果将输入特征图原地展开为列向量，将滤波器权重对应展开为矩阵，则卷积层计算可以写成稀疏矩阵与列向量相乘的形式。这就是Caffe底层运算的机制。同理，全连接层也可写作矩阵相乘的形式。目前Caffe支持的BLAS有开源跨平台的ATLAS和OpenBLAS，以及Intel公司为其处理器优化过的MKL。在NVIDIA生产的GPU上，Caffe也支持NVIDIA的cuBLAS。

将滤波器展开为矩阵、输入原地展开为列向量的方式虽然能利用高度优化的BLAS库，获得良好性能，但缺点是对空间要求很大。如果在GPU上并行实现，手机芯片上GPU和CPU间的通信时间将取代计算时间成为性能瓶颈（见第八章讨论），故本文选择在CPU上使用OpenMP优化Caffe。因MKL闭源，且只能在Intel CPU上使用，在CPU上可用的BLAS只有ATLAS和OpenBLAS两个选项。

根据这篇基准测试[23]的结果，目前OpenBLAS的性能较ATLAS更好，故本文选择使用OpenBLAS。

4.2 OpenMP

OpenMP是一种开放标准的编译器并行化指导方案，只需在一些合适的地方输入预编译指令，就可以暗示编译器在共享内存的多处理器系统上自动产生并行化的代码。在一些串行的代码上应用#pragma omp编译指令表明意图，即可令代码多线程并行执行。新一代手机ARM处理器是多核的，属于共享内存的多处理器系统，C++写成的Caffe在ARM上可以被OpenMP并行优化。

手机安卓系统的应用开发语言是Java，C++写成的代码需要借助安卓开发套件NDK，编译成动态库，在Java中封装为native接口使用。Caffe官方没有对ARM架构和对安卓系统的支持，本文借助GitHub的第三方项目Caffe for Android[24]的生成脚本进行移植，通过脚本指导编译器使用合适的套件，将Caffe以及其依赖库编译成指定架构、指定安卓版本的库。

BVLC的Caffe原版代码中并没有考虑加入OpenMP并行化，其多线程并行化主要体现在其依赖库上。如，OpenBLAS、ATLAS等CPU运行的BLAS支持OpenMP并行化，GPU上使用NVIDIA的CUDA通用计算库可以支持GPU的多线程并行。故，我们如果要在Caffe本身的代码中加入OpenMP并行化，需要在编译脚本中加入对OpenMP的支持。

Caffe借助CMake寻找依赖包，生成编译脚本。我们需要在caffe-android-lib中的Caffe CMakelist.txt中加入对OpenMP的支持：

```
find_package(OpenMP)
if (OPENMP_FOUND)
    message("OPENMP FOUND")
    set (CMAKE_C_FLAGS "$CMAKE_C_FLAGS $OpenMP_C_FLAGS")
    set (CMAKE_CXX_FLAGS "$CMAKE_CXX_FLAGS $OpenMP_CXX_FLAGS")
endif()
```

为支持OpenMP的软件指定线程数主要有两种方法：

显式： 在每处OpenMP预编译宏中，显式地给出受这个宏控制的并行化代码应当具有的线程数量，如：#pragma omp parallel num_threads(8)

隐式： 在代码中不指定并行化代码，运行之前设定环境变量OMP_NUM_THREADS来控制线程数。如，export OMP_NUM_THREADS=8

本文主要使用显式方式指定线程数量，根据卷积层本身的维数性质计算出合适的线程数。缺省线程数按隐式方式给出，取4。

由于Caffe最后被编译为动态链接库caffe.so供可执行文件caffe-time.bin运行时链接，我们还需要在环境变量LD_LIBRARY_PATH中加入动态链接库caffe.so的路径，以确保库能够被可执行文件找到。综上，在执行caffe-time.bin之前，我们需要通过adb shell运行如下脚本设定环境变量：

```
#!/system/bin/sh
export LD_LIBRARY_PATH=./:$LD_LIBRARY_PATH
export OMP_NUM_THREADS=4
```

4.3 基准测试

本文采用AlexNet [1]网络和GoogleNet [2]网络作为基准，评估Caffe的性能。AlexNet [1]的网络结构如图4.1，其主体由五层卷积层连接三层全连接层构成。为方便开发者测试网络性能，Caffe封装了time函数，对外以argument 的形式开放。我们将time函数劫持出来，写到自定义源文件caffe-time.cpp的入口，放到/tools下随Caffe的其他源文件一起编译得到caffe-time.bin。编译完成后，我们采用这个命令作为性能评估基准：

```
./build/tools/caffe-time.bin –model=./models/bvlc_alexnet/deploy.prototxt – iterations
50
```

caffe-time.bin代表含有time函数的可执行文件；model参数赋值为AlexNet的ProtoTXT格式网络描述；iterations为测试的迭代次数。由于第一次迭代开始之前，内存

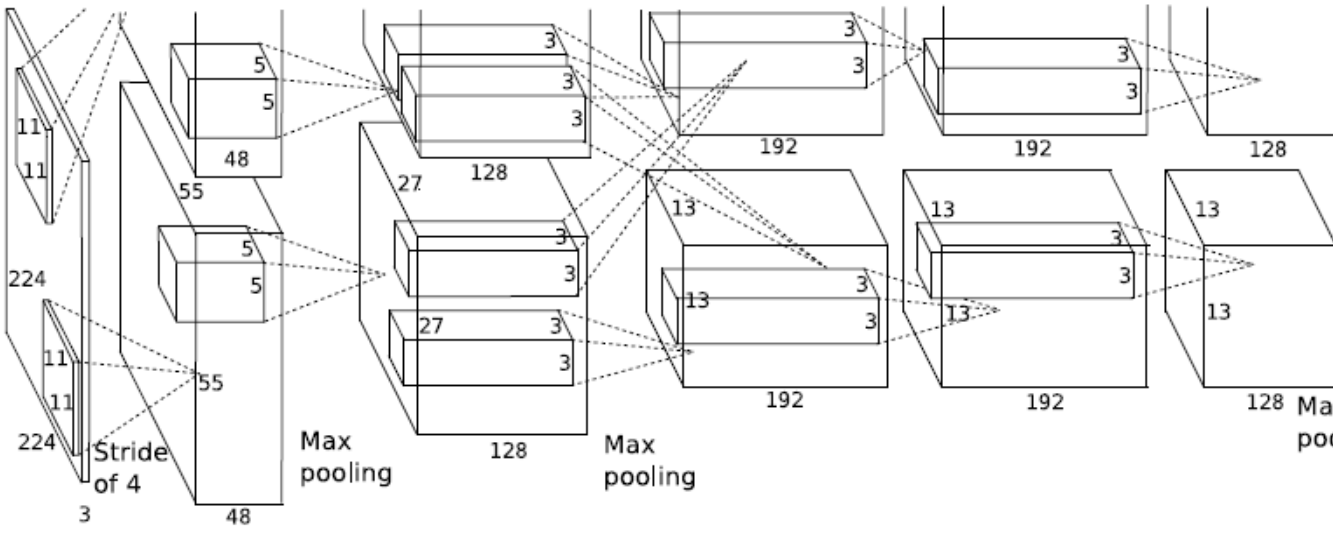

图 4.1: AlexNet的结构示意图

和CPU缓存是冷的，所以需要多次迭代测试取平均值来消除这种误差。“一次迭代”在这个语境下指对整个网络进行一次前向计算和反向传播。该命令等价于 ./build/tools/caffe time –model=./models/bvlc_alexnet/deploy.prototxt – iterations 50

执行命令之后，Caffe的输出范例如下 [21]：

I0101 01:08:22.013758 3686 caffe-time.cpp:56] Testing for 50 iterations.

I0101 01:08:25.120884 3686 caffe-time.cpp:84] Iteration: 1 forward-backward time: 3107 ms.

I0101 01:08:27.764709 3686 caffe-time.cpp:84] Iteration: 2 forward-backward time: 2643 ms.

……

I0101 01:10:34.847990 3686 caffe-time.cpp:84] Iteration: 48 forward-backward time: 2973 ms.

I0101 01:10:37.622038 3686 caffe-time.cpp:84] Iteration: 49 forward-backward time: 2773 ms.

I0101 01:10:40.450104 3686 caffe-time.cpp:84] Iteration: 50 forward-backward time:

2827 ms.

I0101 01:10:40.450280 3686 caffe-time.cpp:87] Average time per layer:

I0101 01:10:40.450315 3686 caffe-time.cpp:90] dataforward: 0.00376 ms.

I0101 01:10:40.450361 3686 caffe-time.cpp:93] databackward: 0.00446 ms.

I0101 01:10:40.450405 3686 caffe-time.cpp:90] conv1forward: 158.396 ms.

I0101 01:10:40.450447 3686 caffe-time.cpp:93] conv1backward: 157.189 ms.

I0101 01:10:40.450488 3686 caffe-time.cpp:90] relu1forward: 14.4338 ms.

I0101 01:10:40.450694 3686 caffe-time.cpp:93] relu1backward: 0.00406 ms.

……

I0101 01:10:40.451578 3686 caffe-time.cpp:93] fc8backward: 10.1705 ms.

I0101 01:10:40.451598 3686 caffe-time.cpp:90] probforward: 0.45548 ms.

I0101 01:10:40.451619 3686 caffe-time.cpp:93] probbackward: 0.1202 ms.

I0101 01:10:40.451649 3686 caffe-time.cpp:98] Average Forward pass: 1591.12 ms.

I0101 01:10:40.451672 3686 caffe-time.cpp:100] Average Backward pass: 1177.4 ms.

I0101 01:10:40.451692 3686 caffe-time.cpp:102] Average Forward-Backward: 2768.74 ms.

I0101 01:10:40.451713 3686 caffe-time.cpp:104] Total Time: 138437 ms.

I0101 01:10:40.451732 3686 caffe-time.cpp:105] *** Benchmark ends ***

可以从输出中得知：

- 指定次数迭代中，每次迭代使用总时间、每次迭代使用的平均时间；
- 多次迭代中各层的平均耗费时间；
- 多次迭代中总的前向反向计算时间；
- 总共所用时间。

我们对Arm64-v8a ARM架构进行测试，使用的测试平台为Open-Q 820开发板[25]。该开发板基于Qualcomm snapdragon 820芯片，CPU为Qualcomm®Kryo™ CPU, 四核64位, 主频2.2GHz。

使用目前最新提交版本[26]的caffe-android-lib脚本进行交叉编译。commit号为2f48b30f98547013f2cc56899ceebceb31108f2b。

使用的gcc,g++编译器版本为6.3.0。

CMake版本3.7.2。

使用adb shell通过USB数据线向开发板发送命令。考虑到对Arm64-v8a编译所需GCC依赖库的兼容性，使用NDK版本为NDK r11c[27]。

第五章　代码优化

Caffe的设计将数据抽象为四维的Blob，将所有对数据的操作（包括卷积、全连接、Pooling、激活函数、Softmax）抽象为层（Layer），Blob数据在Layer中被处理，在Layer间流动。各种Layer的实现存放在src/caffe/layers/下，CPU版本的源文件后缀为.cpp，GPU版本的源文件后缀为.cu（NVIDIA CUDA 通用计算技术）。本文在CPU上执行OpenMP并行化优化，故优化对象主要为src/caffe/layers/下的各.cpp文件。

5.1　热点分析

为了确定优化目标，我们需要分析Caffe中占资源较高的热点函数。根据Intel公司对Caffe的完整基准测试[21]图5.1（测试基于CIFAR-10[17]网络），Caffe运行时占用计算资源最大、运行总时间最长的几个热点，由高到低分别为：

gemm_omp_driver_v2:

该函数是MKL库libmkl_intel_thread.so中的一个函数。GEMM为General Matrix-Matrix的缩写，即“通用矩阵-矩阵相乘”。如前文所述，Caffe 将难以直接优化的卷积层的计算过程以线性代数矩阵相乘的形式表达，而全连接层计算过程本身即可写作矩阵相乘形式，故该函数是Caffe进行卷积、全连接计算的核心函数，占用时间最多。当选取的底层BLAS改为OpenBLAS时，该函数相应变为cblas_gemm()。

caffe::im2col_cpu<float>:

该函数名称im2col_cpu是“image to column on cpu”的缩写，即“从特征图图像到列向量的CPU操作”。如前文所述，Caffe将滤波器对图像的卷积表达为矩阵与

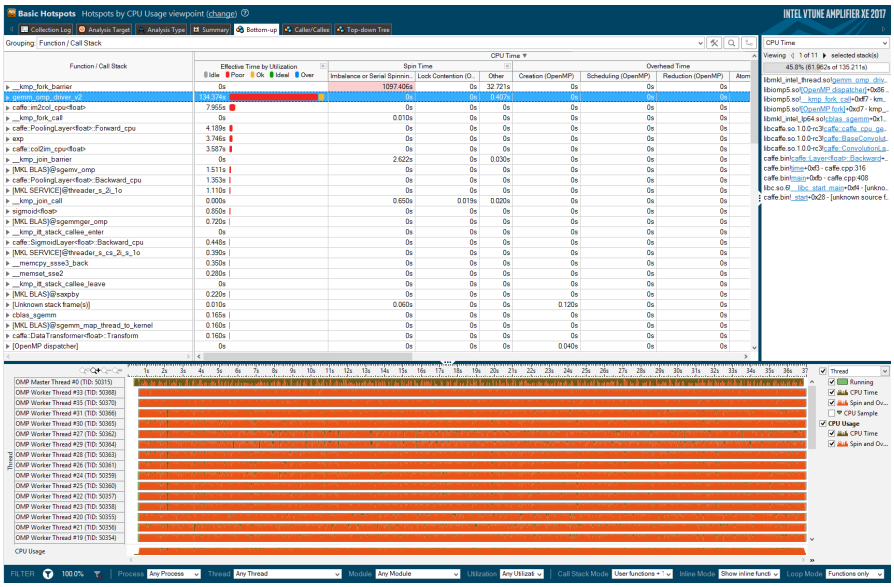

图 5.1: Intel的基准测试

列向量相乘的形式，二维图像需要转换到可供稀疏矩阵相乘的列向量。该过程将图像原地展开，补0填充为非常长的列向量，运算量较大。

caffe::PoolingLayer<float>::Forward_cpu:

该函数为Caffe执行Pooling层前向计算的函数。Pooling是一个对特征图下采样降维的过程，在特征图较大、特征图频道数较多时，该层的运算量不可忽略。

caffe::col2im_cpu<float>:

im2col的逆操作，将列向量转回图片。运算量同理。

5.2 对卷积层的优化

卷积层使用训练好的滤波器，在输入的特征图上做卷积操作。卷积神经网络的学习过程使用梯度下降法，让待学习的参数向着数值梯度的反方向前进，从而使参数达到损失函数取极小值点。但训练数据集很大时，整个数据集无法全部装入内存，这时计算训练中某一时刻的数值梯度的时间空间代价太高，难以承受。实际应用中的做法是规定超参数“Batch Size”，认为从训练集中的一小批样本计算出的梯度可以代表当前整个网络整个训练集的梯度。同时处理成批图片也有助

于提高矩阵运算的局部性，将计算资源充分利用起来。

在定义卷积层OpenMP并行化计算的线程数时，我们必须充分考虑Batch Size参数的影响。因为如果同批处理的样本数量少于所开线程，那么过多线程将引入冗余的计算量，造成计算资源的浪费。线程数多于所需实际数量时，线程之间的通信协调也将拖慢整体的计算时间。

故，我们在基类BaseConvolutionLayer中维护成员变量num_of_threads_为设定的线程数，并设定其初始值为1。解析网络结构配置文件之后，我们取num_of_threads_为OpenMP在平台上最多支持的线程数和Batch Size二者之间较小者。即

$$num\_of\_threads = min(omp\_get\_max\_threads(), BatchSize)$$

具体到代码上的表达为：

```
num_of_threads_ = 1;
#ifdef _OPENMP
num_of_threads_ = omp_get_max_threads() <bottom[0]->shape(0) ?
omp_get_max_threads() : bottom[0]->shape(0);
#endif
```

继承BaseConvolutionLayer的ConvolutionLayer在前向计算中即可使用继承的成员变量num_of_threads_推断线程数，进行OpenMP并行化：

```
#ifdef _OPENMP
#pragma omp parallel for num_threads(this->num_of_threads_)
#endif
for (int n = 0; n <this->num_; ++n)
{
    this->forward_cpu_gemm(bottom_data + n * this->bottom_dim_, weight,
top_data + n * this->top_dim_);
    ......
```

反向传播同理：

```
#ifdef _OPENMP
    #pragma omp parallel for num_threads(this->num_of_threads_)
#endif
    for (int n = 0; n <this->num_; ++n)
    {// gradient w.r.t. bottom data, if necessary.
        this->backward_cpu_gemm(top_diff + n * this->top_dim_, weight, ……
```

通过并行化GEMM矩阵相乘操作，我们完成了对卷积层的并行优化。

5.3 对im2col、col2im函数的优化

im2col函数将输入特征图原地展开为列向量，以将计算过程转化为稀疏矩阵与列向量相乘的形式。col2im是im2col的逆操作，将列向量还原为特征图。Caffe在实现这个函数时并未用到BLAS库，而是以五重循环的形式直接实现。由于循环过深，难以从代码层面直接进行优化。且Intel提出的代码优化 [21]已经被整合到了GitHub [19]上更新后的Caffe中。而且循环中包含较为复杂的条件判断（需要通过下标判定列向量该维补零填位还是填入特征图像素），并行化带来的代价得不偿失。我们保持这两个函数原样，不做优化。

5.4 对Pooling层的优化

Pooling层将特征图做下采样处理，达到降维的目的。Caffe目前只实现了两种较为常用的Pooling操作：Average Pooling与Max Pooling。Average Pooling取下采样格子中各特征图像素的平均值作为输出，Max Pooling取下采样格子中各特征图像素的最大值作为输出。这种Pooling操作可以降维，同时大大增加卷积神经网络的鲁棒性、平移不变性。 Caffe使用四重循环实现这两种Pooling操作，循环指标分别为图片批次、图片频道、长、宽。其中，后两重循环对每个下采样格子进行操作，各下采样格子之间完全独立。故，每个线程对应一个下采样格子，即可

在保证代码正确性的前提下实现并行优化。具体到代码上的表达为：

```
#ifdef _OPENMP
#pragma omp parallel for collapse(2)
for (int n = 0; n <bottom[0]->num(); ++n)
{
    for (int c = 0; c <channels_; ++c)
    {
    ......
```

其中，预编译宏“collapse(2)”意为对下列代码的两重for循环都做展开，进行并行。由于外面两重循环分别代表对批次中的不同图片、图片中的不同频道迭代，这个预编译宏能够达到令每个下采样格子互相并行计算的效果。

5.5 对ReLU的优化

ReLU是目前卷积神经网络较为常用的神经单元激活函数。如前文所述，Caffe将数据组织为Blob在层之间流动，ReLU层每次处理一批神经元，计算它们的激活程度，作为输出传导到下一层。显然，每个神经元之间的激活程度只和其激活函数的输入有关，互相完全独立。故我们对ReLU层的前向和后向计算的for循环直接并行化：

```
#ifdef _OPENMP
#pragma omp parallel for
#endif
for (int i = 0; i <count; ++i)
{
    top_data[i] = std::max(bottom_data[i], Dtype(0)) + negative_slope * std::min(bottom_data[i
Dtype(0));
}
```

和

```
#ifdef _OPENMP
#pragma omp parallel for
#endif
for (int i = 0; i <count; ++i)
{
    bottom_diff[i] = top_diff[i] * ((bottom_data[i] >0) + negative_slope * (bottom_data[i] <= 0));
}
```

第六章　优化结果

经过上一节中的并行化优化，Caffe在支持Arm64-v8a平台上的执行速度得到了明显提升。

6.1　AlexNet [1]基准的优化结果

测试平台为Open-Q 820开发板 [25]。CPU为Qualcomm®Kryo™ CPU, 四核64位, 主频2.2GHz，网络定义选取BVLC提供的AlexNet [1] ProtoTXT配置文件 [28]，进行50次迭代。执行命令为：./build/tools/caffe-time.bin –model=./models/bvlc_alexnet/deplo
– iterations 50

未经优化，直接移植的版本给出的输出（节选）：

I0101 01:03:32.593662 3664 caffe-time.cpp:98] Average Forward pass: 1971.22 ms.

I0101 01:03:32.593865 3664 caffe-time.cpp:100] Average Backward pass: 1418.77 ms.

I0101 01:03:32.594115 3664 caffe-time.cpp:102] Average Forward-Backward: 3390.52 ms.

I0101 01:03:32.594271 3664 caffe-time.cpp:104] Total Time: 169526 ms.

经过优化之后的输出如下：（节选）

I0101 01:10:40.451649 3686 caffe-time.cpp:98] Average Forward pass: 1591.12 ms.

I0101 01:10:40.451672 3686 caffe-time.cpp:100] Average Backward pass: 1177.4 ms.

I0101 01:10:40.451692 3686 caffe-time.cpp:102] Average Forward-Backward: 2768.74 ms.

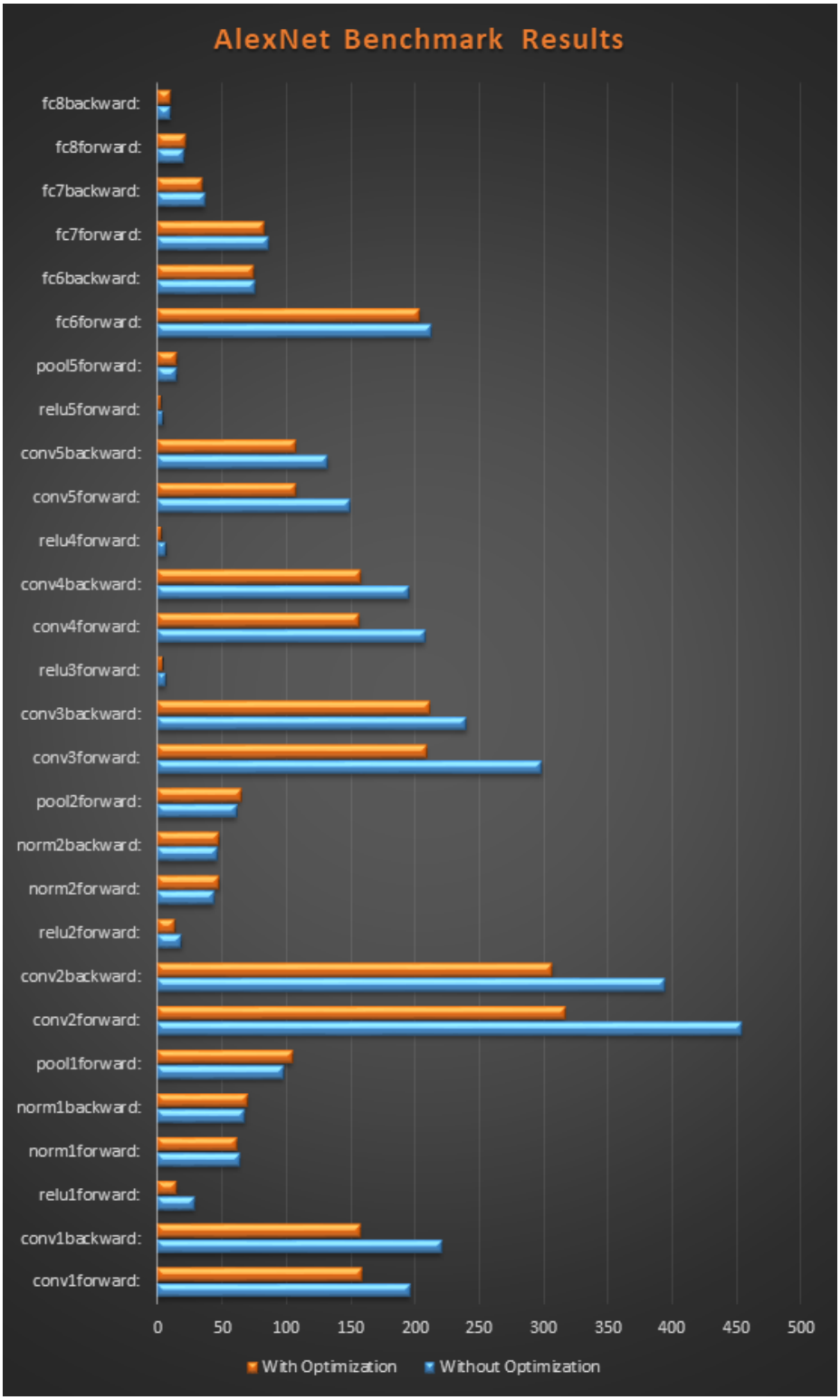

图 6.1: AlexNet的测试结果对比。

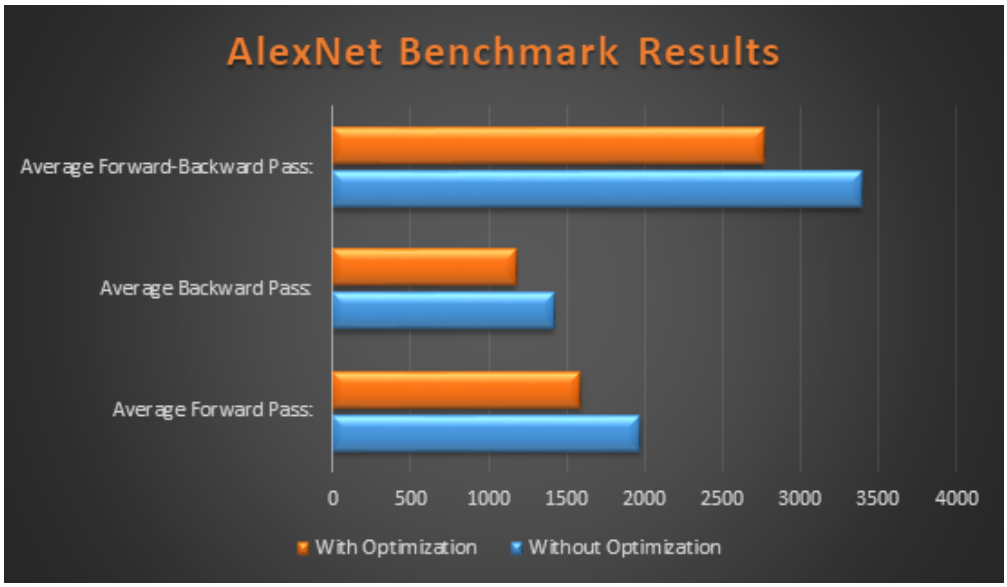

图 6.2: 分层表示的AlexNet测试结果对比。一些操作数极少的层因时间太短被忽略。

I0101 01:10:40.451713 3686 caffe-time.cpp:104] Total Time: 138437 ms.

相比优化之前，总体的执行时间少18%。各层的性能见6.1，总的结果对比见6.2。

6.2 GoogleNet基准的优化结果

GoogleNet [2]是2014年ImageNet ILSVRC图像识别大赛的冠军网络，由22层网络构成。其网络结构较AlexNet大很多，计算量也更贴近近些年来新提出的卷积神经网络模型计算量。使用GoogleNet [2]网络对我们的优化效果做基准测试很有必要。测试平台同9.1，网络定义选取BVLC提供的GoogleNet [2] ProtoTXT配置文件 [29]，进行50次迭代。执行命令为：./build/tools/caffe-time.bin –model=./models/bvlc_googlenet/deploy.prototxt – iterations 50
未经优化，直接移植的版本给出的输出：
I0101 01:36:44.662608 3781 caffe-time.cpp:98] Average Forward pass: 7039.31 ms.

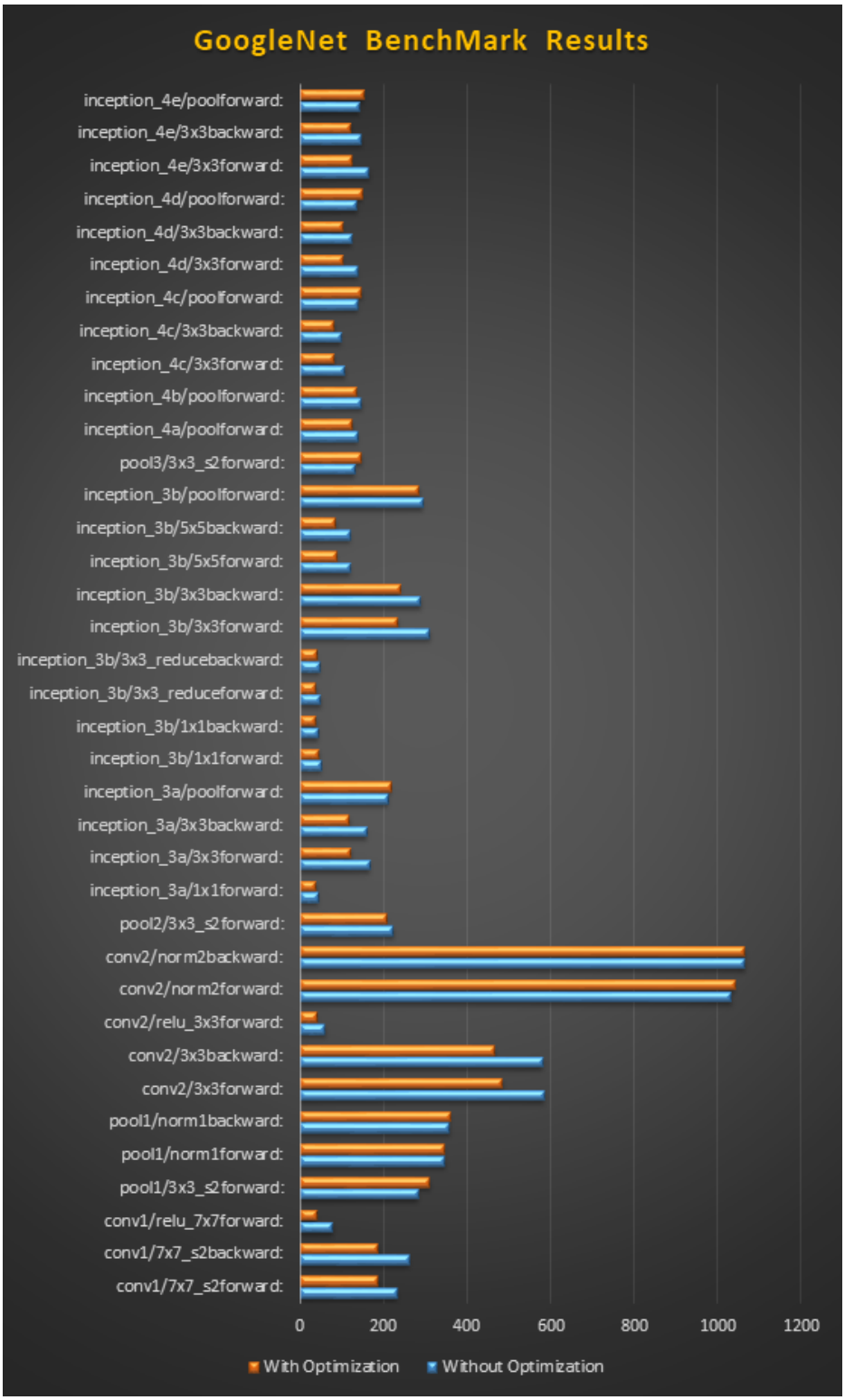

图 6.3: 分层表示的GoogleNet测试结果对比。一些操作数极少的层因时间太短被忽略。

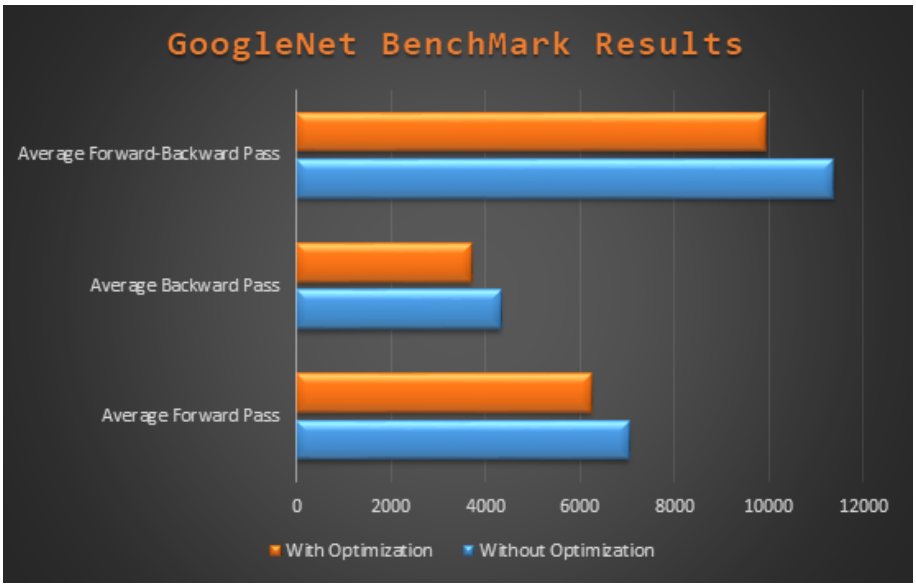

图 6.4: GoogleNet测试结果对比。

I0101 01:36:44.662745 3781 caffe-time.cpp:100] Average Backward pass: 4331.88 ms.

I0101 01:36:44.662867 3781 caffe-time.cpp:102] Average Forward-Backward: 11372.7 ms.

I0101 01:36:44.662983 3781 caffe-time.cpp:104] Total Time: 568637 ms.

经过优化之后的输出如下：

I0101 01:48:43.451630 3803 caffe-time.cpp:98] Average Forward pass: 6251.23 ms.

I0101 01:48:43.451734 3803 caffe-time.cpp:100] Average Backward pass: 3699.81 ms.

I0101 01:48:43.451839 3803 caffe-time.cpp:102] Average Forward-Backward: 9952.54 ms.

I0101 01:48:43.451954 3803 caffe-time.cpp:104] Total Time: 497627 ms.

总体所用时间减少12.5%。各层的性能见6.3，总的结果对比见6.4。

6.3 对结果的讨论

经过优化之后，我们的解决方案运行速度得到了长足的提升。卷积层的计算速度平均提升约19%（AlexNet [1]为19.1%，GoogleNet [2]为19.2%），全连接层提升约5%。对AlexNet和GoogleNet，整体性能分别提升了18%和12.5%。

然而，池化层(Pooling Layer)的表现并不尽人意，个别层甚至有消耗时间反常变多的现象。这可能是因为优化时的粒度太细，直接细分到了多重循环中的最外两重循环，导致线程数过多，线程间协调通信的overhead抵消了并行处理的优势。

第七章　讨论

经过上一节中的并行化优化，Caffe在支持Arm64-v8a平台上的执行速度得到了明显提升。

7.1　如果引入GPU优化，可行的优化方案有哪些？

将Caffe移植到手机端并使用GPU计算具有可行性，但难度较大：首先，Caffe原版的GPU端并行实现仅支持NVIDIA CUDA通用并行计算技术，也就只能在NVIDIA出品的显卡上运行。但目前广泛使用的高通手机端芯片普遍使用Adreno等非NVIDIA显卡。

这就意味着Caffe的GPU实现无法直接使用，只能令Caffe在CPU模式下运行，同时将开放标准的并行化架构OpenCL实现的OpenBLAS替换进项目依赖库，并指定GPU为OpenCL运行设备，以达到将大部分运算转移到GPU上进行并行操作的效果。

但根据前文所述，Caffe将卷积操作转换为线性代数矩阵相乘的形式，借助BLAS的性能提升运算速度。这种做法显著增多了空间消耗，因为卷积层中，滤波器与特征图局部连接，滤波器权重转写所得的矩阵必然是稀疏的。与矩阵维数相匹配的列向量在原地展开过程中，也引入了大量填零补位。

Caffe因此一直因内存空间消耗巨大为用户诟病。这一劣势在GPU上体现得尤其突出——CPU端执行时，Caffe所需的数据存储在内存中，CPU可以直接访问到，并利用CPU的缓存机制提升性能。但GPU无法直接访问内存，所有滤波器权重和特征图等数据必须经CPU调度，从总线复制到GPU的显存中，才能被GPU访问并计算。桌面端GPU计算能力较强，PCI-E总线带宽虽然是瓶颈，但由于桌

面端GPU显存同样较大，可以一次容纳大量数据，数据无需频繁在内存和显存间流动，GPU的计算速度优势不至被数据复制的瓶颈抵消；

与之相对，手机端GPU计算能力很弱，而且GPU显存较小，计算卷积过程中需要频繁地从内存中更新显存中的数据，数据复制所需时间远远盖过GPU端实际进行计算的时间，计算资源无法充分得到利用。

如果希望实现手机GPU端的并行优化，Caffe基于线性代数操作的底层计算模型是不适合的。可行的方案是直接替换掉其底层计算方式，和Google [16]日前在Arria 10 FPGA上实现AlexNet [1] 一样，换为直接并行进行卷积，引入Winograd变换 [30]以进一步减少滤波操作的浮点数运算，压缩所需空间，以充分发挥其性能。

7.2 本文优化方式的可能提升空间

本文采用OpenMP进行优化。将特征图展开为列向量的im2col操作因为循环层次太深，且存在复杂的逻辑判断，不便于分析独立性以通过OpenMP并行化，没有进行优化。由前文所述，该操作是Caffe的热点函数之一，其实现仍存在性能提升空间。

如果能重构其代码，通过其他操作代替复杂的逻辑判断，将其转化为简单的多重嵌套循环，那么这段代码各层循环执行的独立性将变高，分析这段代码存在的并行性也将变得容易许多，可以通过OpenMP多线程并行优化该段代码，使其外层循环被并行执行。

另，底层计算库OpenBLAS在本文发表时已经发布了OpenBLAS optimized for deep learning分支版本 [31]，引入了针对深度学习的专用优化。如能将本文实现中的OpenBLAS依赖库替换为该版本，对性能应当也有一定的提升。

池化层(Pooling Layer)的并行化效果比较差，按6.3中的讨论，应当适当放粗并行化的分割粒度，避免线程数过多导致overhead时间抵消计算速度。

Armeabi-v7a以及更新架构的指令集引入了浮点指令集与NEON单指令多数据扩展，不再使用性能低下的软模拟方式实现浮点运算。如果能从汇编层面上对OpenBLAS的核心作进一步优化，将非向量化的汇编指令替换为等效的向量化

汇编指令，充分发挥单指令多数据汇编的优势，那么Caffe将得到超出OpenMP 多线程并行化层次之外的性能提升。

结论

本文对业界学界广泛使用的深度学习框架Caffe实现了到ARM架构安卓手机的移植，满足了深度学习开发者在手机上灵活实现神经网络模型的需求，并且使用了OpenMP多线程并行技术对其进行了优化。基于Open-Q 820开发板的测试表明，Caffe经本文所述方式优化后，AlexNet总体执行时间减少18%；GoogleNet总体执行时间减少12%。本文提出的优化方式成功使Caffe在AlexNet、GoogleNet等主流卷积神经网络架构上得到了明显的性能提升。

参考文献

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger, (Editors) Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012. 1097–1105. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. Going Deeper with Convolutions[J]. CoRR. 2014, **abs/1409.4842**

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation[C]. Computer Vision and Pattern Recognition. 2014

[4] Ross Girshick. Fast R-CNN[C]. International Conference on Computer Vision (ICCV). 2015

[5] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[C]. Advances in Neural Information Processing Systems (NIPS). 2015

[6] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection[J]. CoRR. 2015, **abs/1506.02640**

[7] Google. Inc. Google Autodriving cars[Z], 2017. URL https://www.google.com/selfdrivingcar/

[8] NVIDIA. Deep learning in medicine[Z], 2017. URL http://www.nvidia.com/object/deep-learning-in-medicine.html

[9] Kaggle. Data science bowl[Z], 2017. URL https://www.kaggle.com/c/data-science-bowl-2017

[10] Kaiming He Jian Sun Jifeng Dai, Yi Li. R-FCN: Object Detection via Region-based Fully Convolutional Networks[J]. arXiv preprint arXiv:160506409. 2016

[11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector[J]. CoRR. 2015, **abs/1512.02325**

[12] KURT HORNIK. Approximation Capabilities of Muitilayer Feedforward Networks[J]. 1990

[13] ImageNet. ILSVRC Challenge[Z]. URL http://www.image-net.org/

[14] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams. Learning representations by back-propagating errors[J]. Cognitive modeling. 1988, **5**(3):1

[15] Chen Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, Jason Cong. Caffeine: Towards Uniformed Representation and Acceleration for Deep Convolutional Neural Networks[C]. Proceedings of the 35th International Conference on Computer-Aided Design. ICCAD '16, New York, NY, USA: ACM, 2016, 12:1–12:8. URL http://doi.acm.org/10.1145/2966986.2967011

[16] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C. Ling, Gordon R. Chiu. An OpenCL(TM) Deep Learning Accelerator on Arria 10[J]. CoRR. 2017, **abs/1701.03534**

[17] Toronto University. CIFAR-10 Dataset[Z]. URL http://www.cs.toronto.edu/~kriz/cifar.html

[18] Vinod Nair, Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines[C]. Proceedings of the 27th international conference on machine learning (ICML-10). 2010, 807–814

[19] BVLC. Caffe GitHub repositories[Z]. URL https://github.com/BVLC/caffe

[20] BVLC. Caffe[Z]. URL http://caffe.berkeleyvision.org/

[21] Jacek Czaja Mariusz Moczala Vadim Karpusenko, Andres Rodriguez. Caffe Optimized for Intel Architecture: Applying Modern Code Techniques[J]

[22] Yangqing Jia. Convolution in Caffe: a memo[Z]. URL https://github.com/Yangqing/caffe/wiki/Convolution-in-Caffe:-a-memo

[23] Bogdan Oancea, Tudorel Andrei, Raluca Mariana Dragoescu. Accelerating R with high performance linear algebra libraries[J]. CoRR. 2015, **abs/1508.00688**

[24] sh1r0. Caffe Android Lib[Z]. URL https://github.com/sh1r0/caffe-android-lib

[25] Intrinsyc. Qualcomm snapdragon 820 development kit[Z]. URL https://www.intrinsyc.com/snapdragon-embedded-development-kits/snapdragon-820-development-kit/

[26] sh1r0. Caffe Android Lib Latest commit[Z]. URL https://github.com/sh1r0/caffe-android-lib/commit/2f48b30f98547013f2cc56899ceebceb31108f2b

[27] Google. NDK version r11c[Z]. URL https://dl.google.com/android/repository/android-ndk-r11c-linux-x86_64.zip

[28] BVLC. BVLC AlexNet[Z]. URL https://github.com/BVLC/caffe/blob/master/models/bvlc_alexnet/deploy.prototxt

[29] BVLC. BVLC GoogleNet[Z]. URL https://github.com/BVLC/caffe/blob/master/models/bvlc_googlenet/deploy.prototxt

[30] S. Winograd. Arithmetic Complexity of Computations[J]. 1980, **33**

[31] xianyi. OpenBLAS[Z]. URL https://github.com/xianyi/OpenBLAS

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　日期：　　年　　月　　日

学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校 □ 一年 /□ 两年 /□ 三年以后在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：　　导师签名：　　日期：　　年　　月　　日