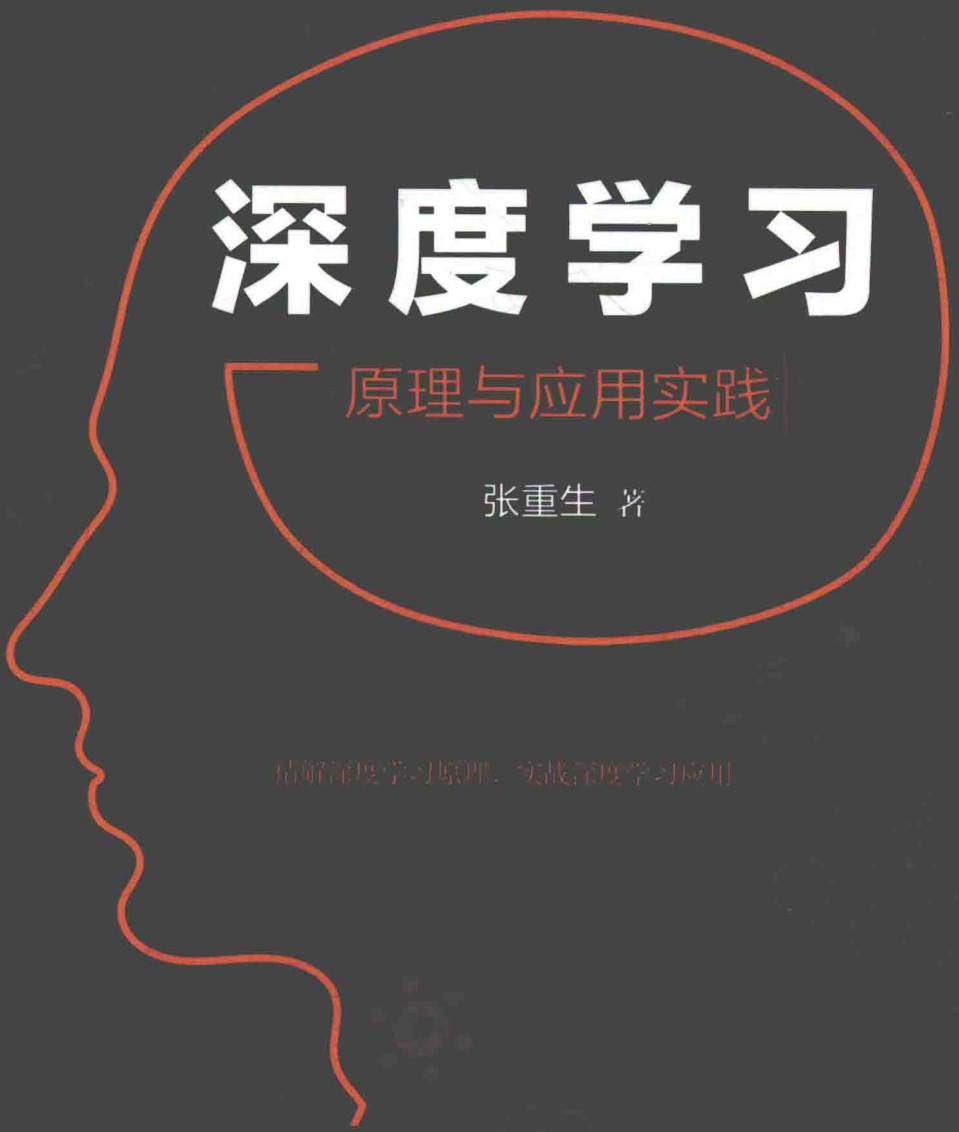
深度学习

原理与应用实践

张重生 著



中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

深度学习
原理与应用实践

张重生 著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书全面、系统地介绍深度学习相关的技术，包括人工神经网络，卷积神经网络，深度学习平台及源代码分析，深度学习入门与进阶，深度学习高级实践。所有章节均附有源程序，所有实验读者均可重现，具有高度的可操作性和实用性。通过学习本书，研究人员、深度学习爱好者，能够在 3 个月内，系统掌握深度学习相关的理论和技术。

本书内容新颖、层次清晰，适合高校教师、研究人员、研究生、高年级本科生、深度学习爱好者使用。

图书在版编目（CIP）数据

深度学习：原理与应用实践 / 张重生著. —北京：电子工业出版社，2016.12
ISBN 978-7-121-30413-2

Ⅰ. ①深… Ⅱ. ①张… Ⅲ. ①学习系统 Ⅳ. ①TP273

中国版本图书馆 CIP 数据核字（2016）第 280651 号

责任编辑：赵 娜
印 刷：北京季蜂印刷有限公司
装 订：北京季蜂印刷有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：720×1 000 1/16 印张：14.5 字数：259 千字
版 次：2016 年 12 月第 1 版
印 次：2017 年 4 月第 2 次印刷
定 价：48.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）88254694。

序言 1

如今，深度学习是国际上非常活跃、非常多产的研究领域，它被广泛应用于计算机视觉、图像分析、语音识别和自然语言处理等诸多领域。在多个领域上，深度神经网络已大幅超越了已有算法的性能。

本书是深度学习领域的一本力作。它对深度神经网络尤其是卷积神经网络进行介绍，且注重深度学习的实际应用。而且，本书还对深度学习研发现状进行总结和阐述，包括对 Google 和 Facebook 的研究与总结。

本书通过示例的方式详解深度学习的具体应用，包括手写数字识别，物体识别，及以人为中心的计算（包括人脸识别、人脸表情识别、年龄估计、人脸关键点定位等）。

本书也介绍了深度学习 Caffe 和 Pylearn2 两个平台，并给出具体示例，介绍如何使用。

本书的所有实验均可重现，对初学者、研究生和工程师有重要参考价值，能够帮助读者掌握深度学习的实战技能。

我在访问河南大学期间与本书作者相识。作者和他的学生为本书的出版投入了 1 年多的时间，付出了巨大的心血。我相信，本书将会被中国科技界所认可。

Ioannis Pitas
IEEE Fellow
亚里士多德大学

序言 2

深度学习是当前最炙手可热的研究领域，可是缺乏一本系统深入的著作。今天，这本书来了！

本书是目前国内为数不多的深度学习原著之一，更是最接地气的、同时兼顾深度学习理论与应用开发实践的深度学习著作。本书面向深度学习领域的研究人员、工程师、爱好者，能够帮助人们把握国内外深度学习的研究现状和产业化趋势，快速理解深度学习的基础理论，牢固掌握深度学习应用开发的实用技术。无论是本书的深度学习理论介绍还是应用开发实例详解，作者都采用实例驱动的方式，使得抽象的理论变得容易理解、难懂的图像处理与模式识别问题变得具体、落地。尤其难能可贵的是，本书还对深度学习平台 Caffe 的源代码进行了精彩解析。

本书的 8 个应用及对应实验都经过作者的反复验证，只需按照书中的相关步骤进行实验，读者就可以重现相关的实验结果。通过本书的学习，读者不但能掌握深度学习相关的基础理论，更能切实掌握深度学习的应用开发技能。因此，对于有意从事深度学习研究与应用实践的研究人员、工程师、爱好者，本书是首选之作。

教授

中国信息协会大数据分会副会长

中国最畅销的《云计算》书籍作者

前 言

深度学习是当今最流行和最受关注的信息技术之一。本书的特点是理论与实践兼备，以举例的方式介绍神经网络和卷积神经网络的原理，然后，通过实例讲解、实战操作的方式，讲解深度学习的具体应用，有利于读者动手能力的培养和解决实际问题能力的提升。

本书旨在降低深度学习的门槛，成为通用、普及性强的深度学习书籍，方便深度学习爱好者快速上手，帮助读者快速入门、进阶、精通深度学习相关的技术，适用于研究人员、高校教师、深度学习爱好者、研究生、高年级本科生。

本书共 15 章，分为 6 篇，第 1 篇是深度学习基础篇，包括第 1～2 章，分别是绪论和国内外深度学习技术研发现状及其产业化趋势。第 2 篇是深度学习理论篇，包括第 3～4 章，分别讲解神经网络和卷积神经网络。第 3 篇是深度学习工具篇，包括第 5～6 章，分别是深度学习工具 Caffe 和 Pylearn2 的介绍和使用详解。第 4 篇是深度学习实践篇（入门与进阶），包括第 7～10 章，分别讲解基于深度学习解决手写数字识别、图像识别、物体图像识别、人脸识别等常见的应用问题。第 5 篇是深度学习实践篇（高级应用），包括第 11～14 章，分别讲解基于深度学习的高级人脸识别算法 DeepID、表情识别、年龄估计和人脸关键点检测。第 6 篇是深度学习总结与展望篇，包括第 15 章，介绍对深度学习的总结和展望。

本书所有章节中的示例内容具有高度的可重现性，读者按照本书的步骤进行操作，编写对应的程序，就能重现相应的实验结果。

本书有助于读者快速入门深度学习的研究和应用实践，帮助读者在较短的时间内掌握相关的技术，并从事基于深度学习的应用开发。然而，深度学习的理论和技术发展极其迅速，尤其是深度学习的理论，需要长期的研究和积淀才能全面理解、深入掌握。本书仅仅介绍了深度学习的基本原理，更多全面、深入、前沿的原理和技术，需要读者自行钻研、主动探究。

本书由张重生和王朋友合著。硕士研究生裴宸平、于珂珂也参与了部分章节的撰写工作，在此致谢。本书的出版，得到了国家自然科学基金（41401466）、省属高校基本科研业务费（xxjc20140005）、河南省科技攻关项目（132102210188）、河南大学研究生院“大数据汇聚与分析”双语课程建设项目（Y1513004）的资助。

感谢中国信息协会大数据分会副会长刘鹏教授，IEEE Fellow、亚里士多德大学 Ioannis Pitas 教授为本书作序。感谢电子工业出版社董亚峰先生对本书的大力支持和无私帮助。

笔者自认才疏学浅，仅略知深度学习之皮毛。书中错谬之处在所难免，如蒙读者诸君不吝告知（本书作者邮箱：chongsheng.zhang@yahoo.com，微信号：A13938613173），将不胜感激。

张重生

2016 年 11 月

目 录

深度学习基础篇

深度学习理论篇

深度学习工具篇

深度学习实践篇（入门与进阶）

深度学习实践篇（高级应用）

深度学习基础篇

第 1 章

绪 论

1.1 引言

深度学习（Deep Learning）[1]，是当今人工智能/机器学习领域研究和应用开发的热点。在“大数据+深度学习”的共同推动下，深度学习在 ImageNet 图像分类竞赛、语音理解、图像识别、视频分析、无人驾驶汽车领域都取得了重要突破。与传统方法不同，深度学习首先通过大规模的迭代实验（调参实验）逼近所能达到的最高识别准确率，然后使用对应的（参数和）模型对新样本（图像、声音等）提取关键特征，并基于该特征，利用已训练好的分类模型预测新样本的类别。目前，Google、Facebook、Microsoft 等国际巨头，以及百度、腾讯、阿里巴巴、京东等国内互联网巨头都已投入巨资布局深度学习，并将其作为重要的研发方向。

1.1.1 Google 的深度学习成果

2015 年 10 月，Google（谷歌）旗下 DeepMind 公司研发了人工智能围棋程序，该程序主要使用深度学习的技术，整体上包含离线学习和在线对弈两个过程，其中离线学习主要利用大量已有棋谱进行训练“价值网络”去计算局面优劣，训练“策略网络”去选择下子位置；在线对弈主要利用“价值网络”计算当前棋面优劣，利用“策略网络”计算当前应该选择的下子位置。2015 年，阿尔法围棋（AlphaGo）以 5: 0 的总比分击败欧洲围棋冠军樊麾；紧接着，2016 年 3 月，以

4:1 的总比分击败世界围棋冠军、职业九段选手李世石[2]。

而在此之前，2011 年谷歌就成立了由人工智能和机器学习顶级学者吴恩达（Andrew Ng）领衔的“Google Brain”项目，这个项目利用谷歌的分布式计算框架训练深度人工神经网络。该项目的主要成果是使用包含 16000 个 CPU 核的并行计算平台，使用基于深度学习算法训练超过 10 亿个神经元的深度神经网络[3]，该系统能够在没有任何先验知识的前提下，自动学习 YouTube 网站上海量的视频数据，训练深度神经网络。吴恩达目前是斯坦福大学计算机科学系和电子工程系副教授、人工智能实验室主任，并担任百度公司首席科学家，负责百度研究院的百度大脑计划。

1.1.2　Microsoft 的深度学习成果

2012 年，微软首席研究官 Rick Rashid 在“21 世纪计算大会”上的英文演讲被实时翻译成与他音色很接近的中文演讲，该功能主要借助于基于深度学习技术实现的自动同声传译系统[4]，自动同声传译过程主要是语音识别、机器翻译和语音合成。

1.1.3　国内公司的深度学习成果

2013 年，百度成立了由知名学者余凯领导的百度深度学习研究院（Institute of Deep Learning，IDL），主要目标是将深度学习应用于语音识别和图像识别、智能检索等领域。现在，基于深度学习，百度的图像搜索更加准确，百度翻译更加专业，语音识别效果令人十分满意。目前，许多基于深度学习的产品已经面市，例如百度识别 APP，该 APP 主要功能是图像识别和智能检索，其中拍照购物和通过照片匹配度来交友都是该 APP 中比较有特色的功能。百度在“小度机器人”和无人驾驶汽车领域等都取得了重要进展。小度机器人能够通过对话等自然的交互方式，准确理解用户意图，并与用户进行信息和服务等的交流。

阿里巴巴的“拍立淘”是基于“大数据+深度学习+图像处理”的构思开发的，网购用户通过手机拍照，利用“拍立淘”就能在淘宝中找到非常类似的产品，其搜索准确度和用户满意度非常高。

LinkFace（脸云科技）在 2014 年开创了基于深度学习的人脸检测算法，支持人脸检测、人脸识别、人脸关键点检测等全套技术，在 FDDB 数据集上的人

脸识别准确率高达 99.5%。图森（http://www.tusimple.com/）通过深度学习引擎，研发了图像识别和语义分析技术，为企业搭建了自己的图片识别服务，根据企业的实际业务设计了分类标签系统，精准描述企业图片分类需求。该公司还研发了基于摄像头的智能驾驶解决方案。

1.2 深度学习技术的发展历程

深度学习是基于人工神经网络发展起来的一项技术，而神经网络的起源可以追溯到 20 世纪 40 年代。神经网络的基本组成单元为神经元，1943 年心理学家 W.S.McCulloch 和数理逻辑学家 W.Pitts 首次提出神经元计算模型，并讲述了神经元的形式化数学描述和网络结构方法，验证了单个神经元具有逻辑功能[5]，从此打开了人工神经网络研究领域的大门。

人工神经网络在全世界科研工作者的持续努力下不断发展，1957 年，Frank Rosenblatt 首次提出“感知器”的概念[6]，该项工作在当时得到了广泛应用，在许多研究领域都取得了一定的成果。但由于当时计算条件的限制，并没有能够持续很长时间。此外，还有一个主要原因是 1969 年 Marvin Minsky 和 Seymour Papert 出版的书籍 *Perceptron*，该书主要论证了“感知器”不能解决异或问题（也就是“异或门”），而且当时的计算机硬件满足不了神经网络的复杂计算。此后，神经网络的发展陷入低潮，在很长一段时间都没有得到重视。

尽管如此，Geoffrey Hinton 一直在坚持神经网络领域的研究，1986 年，DE Rumelhart，G.E. Hinton 和 R.J. Williams 共同在 *Nature* 上发表论文[7]，提出误差后向传播（Error Back Propagation，EBP）算法，该算法解决了“感知器”的“异或”问题，降低了神经网络的计算复杂度，在解决相同问题的前提下，基于误差后向传播算法的神经网络的效果要优于简单“感知器”。误差后向传播算法在各个领域得到广泛应用，并延续至今，重新掀起了神经网络高潮。1989 年，Yann Lecun 成功将误差后向传播算法应用到手写邮政编码的识别任务上[8]。Yann Lecun 在人工神经网络的基础上提出了“卷积神经网络”，目前卷积神经网络在图像和视频分析等领域都取得了很好的成果。

深度神经网络是模仿人脑机制构建的具有学习和分析解决问题的神经网络，它是由输入层、若干隐层和输出层构成的多层网络结构。相邻层之间的神经元通

过权重相互连接，同层或跨层之间的神经元没有连接，权重值的大小表示对整体输出的贡献值，神经元采用非线性激励函数（Activation Function）获得输出值。深度学习之所以被称为“深度”，是相对于支持向量机（Support Vector Machine，SVM），Boosting 提升方法等“浅层学习”方法而言的，深度学习可以自动学习数据特征，不需要手工设计特征，省去了很多人工设计的工作，能够更好地表达数据的本身特征。但深度神经网络需要大量的调参、迭代学习来寻找最佳的特征表达模型。

深度学习一般包含多个网络层，利用海量数据进行训练，通过逐层特征变换，形成对数据本身更抽象的特征，得到更具有代表性的特征，从而提高分类或预测的准确率。“深度学习”是手段，“特征学习”是目的，与“浅层学习”的区别主要在于：

（1）深度模型包含更多的网络层，通常是 5 层、7 层，甚至是 10 层以上。

（2）强调特征学习的重要性，通过逐层特征变换，得到更具有代表性的特征[9]。

2006 年，机器学习泰斗、多伦多大学计算机系教授 Geoffery Hinton 在 *Science* 上发表的文章[10]，采用基于深度信念网络（Deep Belief Networks，DBN）的非监督的逐层训练算法，解决了深度神经网络训练难度大的问题。

随着深度学习技术的不断发展，逐渐出现了许多深度学习开发框架，比较典型的有 Torch [11]，Caffe [12]，Theano [13]，Pylearn2 [14]等，这些工具都可以让开发者很轻松地学习使用深度学习相关技术，并提供了各种不同的接口供不同的开发人员使用，其中 Caffe 是基于 C++实现的开源库，提供了 Python 和 Matlab 外部接口，通过修改配置文件，修改网络参数可以轻松实现训练深度模型，支持 CPU 和 GPU 两种模式的无缝切换；Torch 是基于 Lua 实现的开源库，该开源库在使用的过程中，并不像 Caffe 那样直接修改配置文件就可以实现所需功能，它需要开发人员自己写训练过程的代码，但这样可以更好地理解网络的整个训练过程的原理；Theano 是基于 Python 实现的开源库，是用来有效地定义、优化和计算关于多维数组数学表达式的 Python 类库；Pylearn2 是基于 Theano 开发的深度学习工具，可以方便地定义参数，快速训练模型。2016 年，Google 开发了其深度学习平台 TensorFlow，Facebook 开发了其基于 Torch 的深度学习框架 Torchnet。

1.3 深度学习的应用领域

目前，深度学习在越来越多的领域表现出优越的性能，尤其体现在图像识别、语音识别和自然语言处理等领域。

1.3.1 图像识别领域

在物体识别问题上，深度学习的优势主要体现在ImageNet ILSVRC竞赛上，该竞赛是计算机视觉领域高度权威的竞赛，主要对1000类的物体图像进行识别。2012年，Geoffery Hinton和他的学生针对分类问题将分类Top-5错误率从原来的26.2%降低至15.3%，取得了当时领先的结果[15]。2013年，在ImageNet ILSVRC2013竞赛中，Clarifai模型[16]将分类Top-5错误率降低至11.197%；2014年，在ImageNet ILSVRC 2014竞赛中，GoogleNet[17]通过使用更深的卷积神经网络将分类Top-5错误率降低至6.67%；2015年，在ImageNet ILSVRC 2015竞赛中，微软亚洲研究院（MSRA）的深度网络Deep Residual Network [18]将分类Top-5错误率降低至3.567%。

在人脸识别领域，深度学习的优势主要体现在LFW[19]（Labeled Faces in the Wild）竞赛上的识别准确率。LFW是目前最著名的人脸识别数据库，用来测试非可控条件下的人脸识别准确率，该数据库中的图片是从互联网中获得的，大部分图片在表情、光照、姿态等方面表现出不同的特性，香港中文大学汤晓鸥教授领导的团队设计的DeepID[20]算法取得高达99.53%的识别准确率。

1.3.2 语音识别领域

语音识别要解决的问题首先就是将语音中的音节识别出来，其次将合适的音节组成文字。上述过程构成了语音识别的两大组成部分：声学模型、语言模型[21]。在很长一段时间内，声学模型使用的是自动机的方法进行划分，最经典的建模方法是隐马尔可夫模型。而在语言模型方面一般分为规则模型和统计模型两种，统计语言模型是用概率统计的方法来揭示语言单位内在的统计规律，其中N-Gram简单有效，被广泛使用。最近，基于深度神经网络技术，百度和科大讯飞在语音

识别领域都取得了重要突破，百度的 Deep Speech 采用深度学习技术对语音进行识别，它可以在饭店等嘈杂环境下实现将近 81%的辨识准确率。而同类商业版语音识别系统如 Microsoft Bing、Google 等公司的最高识别率只有 65%[22]。

1.3.3　自然语言理解领域

应用深度学习模型进行自然语言处理，目前主流的做法是应用 Recursive Neural Network（递归神经网络）和 Recurrent Neural Network（循环神经网络）。其中，Recurrent Neural Network 是非常有名的应用于情绪分析的树状神经网络模型，它是包含循环的网络，允许信息的持久化，更加适用于自然语言处理。

1.4　如何开展深度学习的研究和应用开发

进行深度学习的研究和应用开发，首先，应该确定具体的问题和解决问题的目标；其次，需要采集或通过公开途径获得大量相关的有标注的数据集；然后选择合适的深度学习平台进行深度神经网络模型的训练；最后，得到特征提取模型和目标分类模型。

本节以深度学习的主流平台——Caffe 为例，概要讲解如何使用深度学习平台在自己的数据集上训练模型，如何设计网络结构，如何调整网络参数的相关技术。Caffe 的具体讲解将在本书第 5 章进行。

目前使用 Caffe 训练深度模型主要有两种方法：①利用当前数据从零开始重新训练深度模型；②加载已有模型，利用加载到的模型的参数作为训练新模型的初始化参数。Caffe 中提供了许多基于大量数据训练的深度模型，读者可以参考文献[23]下载自己需要的深度模型。至于如何选择这两种方法，如果你的数据量较小或者数据与文献[23]中的某个模型所用的数据很相似，就可以优先选择第二种方法。当然，也可以尝试第一种方法。

下面主要参考文献[24]讲解如何加载 Caffe 中的已有模型来训练自己的数据，得到针对自己数据集的特有模型。该例使用在 ImageNet 数据集上训练得到的 CaffeNet 模型。ImageNet 是一个数据量达到百万级的数据库，CaffeNet 模型

本身解决的是物体分类问题，而这里[24]主要是解决“Flickr Style”[25]数据库问题，即图片风格分类问题，该数据库包含 8 万张图片。选择加载 CaffeNet 模型的主要原因是“Flickr Style”数据库中的图片与 ImageNet 很相似。

具体步骤如下（假设所有操作都在配置好的 Caffe 根目录下执行）。

1. 准备自己的数据

```
python examples/finetune_flickr_style/assemble_data.py --workers=-1 --images= 2000 --seed 831486
```

使用上面的命令，下载“Flickr Style”数据库中的 2000 张图片，并自动将数据分为训练和测试数据，保存在 data/flickr_style 文件夹中，训练文件为 train.txt，测试文件为 test.txt，这两个文件中的数据格式：图片路径、该图片对应的标签，路径和标签之间用空格隔开。

2. 下载 Caffe 中已有模型

```
./scripts/download_model_binary.py models/bvlc_reference_caffenet
```

使用上面的命令，下载 Caffe 中已有的 CaffeNet 模型，保存在 models 文件夹中，模型的文件名为 bvlc_reference_caffenet.caffemodel，该文件中保存了该模型对应的权重和偏置等相关参数值。

由于在训练新的模型的时候，需要对应的均值文件，可以通过下面的命令得到，对应文件保存在 data/ilsvrc12/文件夹下，名称为 imagenet_mean.binaryproto。

```
data/ilsvrc12/get_ilsvrc_aux.sh
```

3. 修改网络结构

首先需要强调的是，由于使用的是 CaffeNet 模型，所以使用的网络结构必须是 CaffeNet 网络结构，这些可以从 models\bvlc_reference_caffenet 文件夹下获得。由于“Flickr Style”数据库中的图片风格共有 20 类，而 ImageNet 数据库中图片共有 1000 类，所以需要将 CaffeNet 网络中的最后一个全连接层的输出数量从 1000 修改为 20，这个至关重要。另外，加载已有模型的参数到现有网络中对应的网络层主要依据网络中的层名称，需要将最后一个全连接层的层名称从 fc8 修改为其他不一样的名称，例如 fc8_flickr，这样就可以实现从随机初始化的权重开始学习这一层。

网络中最后一个全连接层修改如下：

```
layer {
  name: "fc8_flickr"
  type: "InnerProduct"
```

```
    bottom: "fc7"
    top: "fc8_flickr"

    # lr_mult 设置的值要高于其他层，主要因为该层从随机初始化权重开始学习
# 而其他层则是从已有模型中直接加载获得的初始值
    param {
      lr_mult: 10
      decay_mult: 1
    }
    param {
      lr_mult: 20
      decay_mult: 0
    }

    inner_product_param {
  #"Flickr Style" 数据库中图片风格共有 20 类
      num_output: 20
      weight_filler {
        type: "gaussian"
        std: 0.01
      }
      bias_filler {
        type: "constant"
        value: 0
      }
    }
  }
```

此外，还需要修改 models/finetune_flickr_style 文件夹下 solver.prototxt 文件的内容，修改如下：

```
net: "models/finetune_flickr_style/train_val.prototxt"
test_iter: 100
test_interval: 1000
#学习率应该设置为小于已有模型的初始学习率
base_lr: 0.001
lr_policy: "step"
gamma: 0.1

# 步长也应该适当减小
```

```
stepsize: 20000
display: 20
max_iter: 100000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000

#模型文件保存的路径及文件前缀名称
snapshot_prefix: "models/finetune_flickr_style/finetune_flickr_style"
# solver_mode: CPU
```

4. 加载已有模型训练自己的数据

```
./build/tools/caffe train -solver models/finetune_flickr_style/solver.prototxt -weights models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel -gpu 0
```

使用上面的命令，加载已有模型的参数，主要通过-weights 来指定已有模型对应的文件路径，训练针对自己的数据的深度模型；如果不想通过加载已有模型的方式训练自己的模型，可以通过去掉-weights 参数来实现。最后模型保存在 models/finetune_flickr_style/文件夹中，随着迭代的进行，会保存多个模型文件。而 finetune_flickr_style _100000.caffemodel 是完成所有迭代后得到的最终模型文件。

紧接着，就是调整参数的过程，这里，需要调整的参数较多，目前的技术不能全自动地确定调某一个参数就能得到最好的模型。当前都是通过逐个尝试的方法来调参。具体而言，调参主要包括调整 solver.prototxt 文件中的学习率（base_lr），权重衰减值（weight_decay）等。当然也可以尝试使用不同的学习率改变策略（lr_policy），具体可以参考本书第 5 章中不同的学习率改变策略。

调整学习率的完整过程如下：首先固定一个学习率的值，然后训练模型，直到验证准确率不再上升，可以尝试将学习率降低继续训练，反复调整学习率的值，直至训练出一个较好的深度模型。

上述四个主要步骤非常重要，它们是深度学习业内人士进行深度学习研究和应用开发时，为了在自己的具体数据上训练特征提取模型时，经常使用的解决方案。

参考文献

[1] Deep learning. Wikipedia. https://en.wikipedia.org/wiki/Deep_learning.

[2] 谷歌 AlphaGo 大战李世石首局：机器人胜！http://www.chinarobots.cn/XingYeDongTai/143.html.

[3] Q.V. Le, M.A. Ranzato, R. Monga, M. Devin, K. Chen, G.S. Corrado, J. Dean, A.Y. Ng. Building high-level features using large scale unsupervised learning. ICML, 2012.

[4] Rick Rashid, Speech Recognition Breakthrough for the Spoken, Translated Word. http://www.youtube.com/watch?v=Nu-nlQqFCKg.

[5] 人工神经网络. http://baike.baidu.com/.

[6] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain.[J]. Psychological Review, 1958, 65(6):386-408.

[7] Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors[C]// MIT Press, 1986:533-536.

[8] Lecun Y, Boser B, Denker J S, et al. Backpropagation APPlied to Handwritten Zip Code Recognition[J]. Neural Computation, 1989, 1(4):541-551.

[9] 余凯，贾磊，陈雨强，等. 深度学习的昨天、今天和明天[J]. 计算机研究与发展，2013，50（9）：1799-1804.

[10] Geoffery E. Hinton, Salakhutdinov RR. Reducing the dimensionality of data with neural networks[J]. Science. 2006 Jul 28;313(5786):504-507.

[11] Torch. A scientific computing framework. http://torch.ch/.

[12] Caffe. Deep Learning Framework. http://caffe.berkeleyvision.org/.

[13] Theano. http://deeplearning.net/software/theano/.

[14] Pylearn2. Machine learning library. http://deeplearning.net/software/pylearn2/.

[15] ImageNet Classification with Deep Convolutional Neural Networks, Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, NIPS 2012.

[16] http://www.clarifai.com/.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. arXiv:1409.4842, 2014.

[18] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition[J]. Computer Science, 2015.

[19] Huang G B, Mattar M, Berg T, et al. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments[J]. Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition, 2007.

[20] Sun Y, Liang D, Wang X, et al. DeepID3: Face Recognition with Very Deep Neural Networks[J]. Computer Science, 2015.

[21] http://cslt.riit.tsinghua.edu.cn/~fzheng/THESES/201506-M-LC.pdf.

[22] http://36kr.com/p/217970.html.

[23] http://caffe.berkeleyvision.org/model_zoo.html.

[24] http://caffe.berkeleyvision.org/gathered/examples/finetune_flickr_style.html.

[25] Karayev S, Trentacoste M, Han H, et al. Recognizing Image Style[J]. Eprint Arxiv, 2013.

第 2 章

国内外深度学习技术研发现状及其产业化趋势

本章主要介绍国内外 IT 巨头在深度学习领域的研发现状及其产业化趋势。包括 Google、Facebook、百度、阿里巴巴、京东、腾讯、NVIDIA、Face++等知名公司。

2.1 Google 在深度学习领域的研发现状

2.1.1 深度学习在 Google 的应用

2012 年谷歌发起了一个项目，这个项目当时是由吴恩达领导的（吴恩达现任百度首席科学家），该项目让一个神经网络使用 16000 个 CPU 服务器在 1000 万个 YouTube 视频进行训练，算法自己学会了识别猫脸；谷歌在 2014 年设计了 GoogLeNet，在 ILSVRC 中将分类错误率低至 6.66%。谷歌早期的深度学习基础平台是建立在大规模 CPU 集群的 DistBelief（由 16000 个 CPU 计算节点构成），现在使用的深度学习平台是建立在超过 8000 个 GPU 组成的集群上的 TensorFlow。

谷歌深度学习的应用包括：①语音识别，谷歌基于深度神经网络训练语音识别模型，而非传统的隐马尔科夫模型。2016 年 3 月，谷歌开放了其语音识别接口——Google Cloud Speech API[1]。②图像识别、图像类别识别。③图像搜索。④街景图像识别（含文字识别）。⑤自然语言理解、智能回答。⑥谷歌翻译等[2, 3]。

2016 年 6～7 月，谷歌旗下的 DeepMind 公司宣布与英国国家卫生服务体系（NHS）合作，准备利用机器学习来进行眼疾诊断，其目标是仅通过一次视网膜扫描来识别影响视力的症状。NHS 的 Moorfields 眼科医院将向 DeepMind 提供 100 万份匿名的眼球扫描资料，DeepMind 使用机器学习、深度学习技术在该数据上进行训练，以便更好地发现与显性年龄相关黄斑变性以及糖尿病视网膜病变等眼疾的早期迹象[4, 5]。

2.1.2 Google 的 TensorFlow 深度学习平台

2015 年 11 月，Google 开源了人工智能学习系统 TensorFlow[6]（见图 2-1），成为 2015 年最受关注的六大开源项目之一。TensorFlow 是 Google Brain 团队开发的一款深度学习软件。TensorFlow 的计算过程是数据流图的方式，最初的开源版本不支持分布式计算，只能在单机上运行[7]。2016 年 4 月，谷歌发布了分布式 TensorFlow 0.8，它最突出的特征是增加分布式计算的支持，能够在不同的机器上同时运行，从而在不同的机器上（集群中）分布式地训练模型，能够把部分模型的训练过程从数周缩短到几个小时[8]。

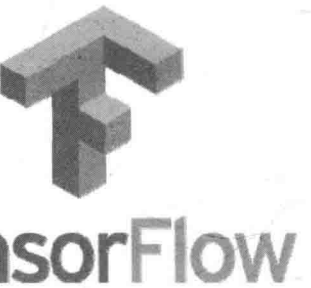


图 2-1　Google 的 TensorFlow 深度学习系统

在谷歌内部，TensorFlow 已经成功地应用到了谷歌搜索、广告、翻译、图像识别、语音识别、自然语言处理等众多产品之中。

2.1.3　Google 的深度学习芯片 TPU

2016 年 5 月，谷歌发布了用于深度学习芯片 TPU（Tensor Processing Unit），该芯片支持了深度学习框架 TensorFlow（见图 2-2）。TPU 优化了硬件架构的设计，允许芯片降低计算精度，这意味着每个操作需要的晶体管数量更少，可以在每秒时间内加入更多的操作，可以使用更加复杂和强大的深度学习模型[9, 10]。据称，TPU 的性能是普通 GPU 的近 10 倍。

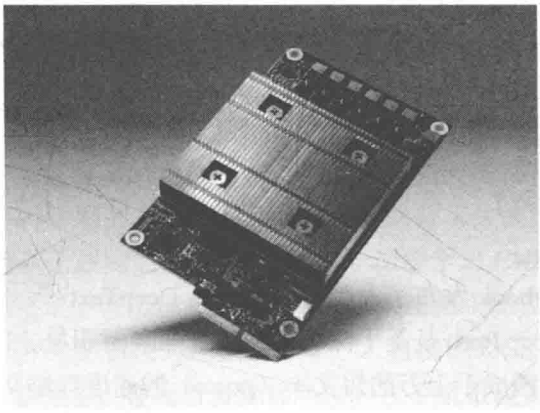


图 2-2　Google 的深度学习芯片 TPU

2.2　Facebook 在深度学习领域的研发现状

2016 年，Facebook 公开了其深度学习平台 Torchnet 和基于深度学习技术的自然语言理解引擎 DeepText。

2.2.1　Torchnet

2016 年 6 月，Facebook 开源了其深度学习框架 Torchnet[11]。Torchnet 的主要聚焦不在于进行高效的推断，也不在于使得 Torch 神经网络中的梯度计算更快[12]。相反，Torchnet 旨在建立一个在深度学习框架 Torch 之上的外层框架

（Facebook 选择的是在 Torch 上编译一个开源库[13]，而并非建立一个全新的深度学习框架），该外层框架通过提供样板代码和模块化的设计，以便于代码重用，使得深度学习的实验和应用变得容易、简单、快速，大大减少了从零开始设计复杂的深度学习代码和模块所需的时间。Torchnet 可以用于图像识别、自然语言处理。而且，Facebook 的科学家 Van der Maaten 写道[13]：Torchnet 可能并不会一直局限在 Torch 上使用，Torchnet 是抽象的，可以轻松应用到其他框架中，例如 Caffe 还有谷歌的 TensorFlow 框架。

Torchnet 使用 Lua 脚本语言开发，可以运行在标准的 x86 芯片或图形处理单元（GPUs）上，提高了代码的重复使用率。这意味着在以后的开发中开发者可以使用这个框架来简化工作，并且降低引入 Bug 的概率[14]。

2.2.2 DeepText

2016 年，Facebook 发布了人工智能新产品 DeepText[15]，它能够准确识别人类的聊天内容。DeepText 是基于深度学习的文本理解引擎，它能以接近人类的准确度，以每秒处理成千上万的短文本（posts）的速度理解文本内容。

DeepText 使用了卷积神经网络（Convolutional Neural Nets），递归神经网络（Recurrent Neural Nets），使用 FbLearner Flow 和 Torch 用于模型训练，这是一个分布式的、可扩展、高可信的架构。通过 DeepText 的架构，Facebook 的工程师可以轻而易举地建立自然语言处理的模型。

Facebook 机器学习团队的工程技术主管 Hussein MeHanna 表示："我们希望将 Deep Text 运用在对 Facebook 平台上的海量内容进行分类，从而让搜索内容变得更加容易，同时为用户呈现他们感兴趣的内容"[16~18]。Facebook 表示，Deep Text 目前已经在一些方面开始发挥作用了。例如，Facebook Messenger 上的一些聊天机器人现在已经能够基于用户发的聊天内容理解他是否需要叫出租车[16]。除识别聊天内容，自动搜索给出建议之外，Deep Text 还会用于 Facebook 上的垃圾消息清理[16]。

DeepText 还可以进一步改善 Facebook 的用户体验，例如，很多名人和公众人物会使用 Facebook 与大家进行交流，这些交流通常可能获得数百条，甚至上千条评论。此时就可以借助 DeepText 找出最相关的内容，同时确保评论始终维持较高质量[17, 18]。

2.3　百度在深度学习领域的研发现状

2.3.1　光学字符识别

传统的 OCR 技术包括版面分析、行分割、字分割、单字识别、语言模型解码等步骤，其中行分割涉及二值化、连通域分析等技术。这种技术在印刷体文档扫描识别等传统应用场合中取得了巨大成功，但并不适用于自然场合中的文字识别（自然场合中，背景复杂，噪声多，光照、拍摄角度等都有很多影响）。

百度对光学字符识别的系统流程和技术框架进行了大幅改造，舍弃了传统的二值化和连通域等基于规则的方法，形成了基于深度学习 CNN-RNN 的光学字符识别流程，不考虑每个字符出现的具体位置，只关注整个图像序列对应的文字内容，使得单字分割和单字识别问题融为一体[19]。百度的 OCR 技术已用于百度街景，可识别店铺、商家等。使用百度翻译 APP，在手机摄像头聚焦菜单后，可以实现对菜单中文字的定位、识别与翻译[20]。

2.3.2　商品图像搜索

类似于淘宝的“拍立淘”，百度支持全新的拍照购物方式。百度商品搜索支持对服装、箱包、鞋类等商品的拍照检索。针对服装，百度结合图像的颜色和纹理，引入深度学习技术，可以找到图片中人物与衣服的语义表示（如修身无袖、女装、棉麻连衣裙）。另外，百度设计了一套人物与服装对齐算法，较好地解决了用户拍照时图片的大小、背景各不相同，拍摄姿态、角度也与图片库中的模特存在差异的问题，极大地提升了用户购物体验[21]。对于箱包，百度开发了一套精确箱包物体检测和分割算法。该算法可以精确定位出包的位置，并精确分割出包的图像。然后，百度使用深度学习技术，训练众多箱包的识别模型，针对在实拍中可能出现的诸多复杂情况，如背景、姿态、遮挡、光照环境等，做到了精确识别和检索[21]。

2.3.3 在线广告

广告数据特征维度非常高，有上百亿个稀疏特征。基于深度学习，百度设计了 DANOVA 算法，由底层到高层，逐层贪婪学习，从底层（第一层）的单特征开始，到更高层的不同数量的特征组合（第二层的 2 个特征组合，到第三层的 3 个特征组合……），构造高阶组合特征。 DANOVA 算法使得特征挖掘效率提升上千倍，CTR 显著增长[22, 23]。据称，百度基于深度学习将 CTR 提升了 3.7%。

2.3.4 以图搜图

以图搜图，就是基于内容的图像搜索（Content-Based Image Retrieval，CBIR），输入的是图片，需要返回与之相关的图片。这其中包括人脸检索、相似性检索等[24]。百度基于深度学习的人脸识别技术在 LFW 评测中达到 99.77%的准确率[25, 26]。另外，百度开发了世界上第一款基于深度学习的拍照实物搜索 APP[25]. 目前，通过公开渠道尚无法获知百度以图搜图的技术细节。

2.3.5 语音识别

百度开发出深度语音识别系统 Deep Speech，其语音识别准确率高于谷歌，在噪声环境中的表现更为突出[27, 28]。Deep Speech 基于深度学习技术，收集了 9600 个人长达 7000 小时语音，并在其中增加了 15 类噪声，最后将数据扩容成一个 10 万小时的数据。百度语音识别系统基于上千块 GPU 硬件进行训练，让语音识别模型训练速度变得更快。

百度语音识别的基本原理并非是基于音节的技术，而是直接在音频和其对应文字内容之间建立映射关系，训练深度学习模型。目前，百度的准确率已经能达到 97%，每天的语音识别请求超过 1 亿[29]。百度硅谷研究人员 Greg Diamos 设计了持久 RNN（Persistent recurrent Neural Networks）技术，减少训练 RNN 的内存使用量以及每块 GPU 小批量的大小，加快了深度递归神经网络的训练速度[30]。

2.3.6　百度开源深度学习平台 MXNet 及其改进的深度语音识别系统 Warp-CTC

百度于 2015 年 5 月通过“深盟”（Deep Machine Learning in Common[31, 32] DMLC）开源了其深度机器学习开源平台 MXNet[33~35]（见图 2-3），又在 2016 年 1 月开源了 Warp-CTC 系统。CTC（Connectionist Temporal Classification，链结式时间分类算法）是 Jurgen Schmidhuber 团队在 2006 年提出来的，较优于 RNN 方案。经过百度方面优化之后的 Warp-CTC 可以提高原来 CTC 运算效率 10 到 400 倍[36]。Warp-CTC 系统本质上是根据几年前开发的深度学习算法改进实现的，算法使用 C 语言编译，并做了集成化处理，应用于语音识别技术中[37]。

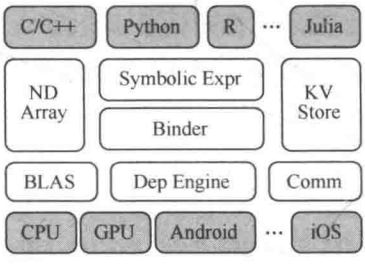


图 2-3　百度深度机器学习开源平台 MXNet 的系统架构

据称，百度的深度学习硬件平台包括 6000 台 PC 服务器和 2000 余台 GPU 服务器，能够构建世界上最大的包含千亿级参数的人工神经网络。

2.4　阿里巴巴在深度学习领域的研发现状

2.4.1　拍立淘

阿里巴巴的“拍立淘”允许用户以手机拍照，然后在淘宝或天猫中搜索网上相同或者相似的商品[38, 39]。拍立淘包括以下三个主要步骤[38]。

第一步是确定一个图片的大致类型，例如，一个商品需要知道它是帽子，上衣，裙子，还是鞋子等。

第二步是主体检测，从图片中检测感兴趣的商品的区域。为了减少计算量，

该步骤还需要过滤候选窗口的步骤。

第三步是图像特征提取。对第二步中确定的窗口中的图像区域，选择合适的特征来描述商品。该步骤中，拍立淘利用深度学习工具使神经网络收敛到一个地方，使得特征输出能够反映出这个商品的特性，如种类、风格、图案、颜色等[38]。

2.4.2 阿里小蜜——智能客服 Messenger

2015 年 7 月，阿里巴巴在手机淘宝客户端推出了新一代智能客服产品——阿里小蜜，基于语音识别、语义理解、个性化推荐、深度学习等人工智能技术的应用，在每天应对百万级服务量的情况下，阿里小蜜的智能解决率达到了接近 80%[40]。除了客服，阿里小蜜还支持充话费、查天气、买机票、导购等功能[41]。阿里小蜜的基本功能和技术原理与京东 JIMI 有很多相似之处，使用深度学习实现自然语言理解。

2.5 京东在深度学习领域的研发现状

2014 年 9 月，京东挂牌成立了京东深度神经网络实验室（JD DNN Lab）。

京东客服机器人 JDIMI（JD Instant Messaging Intelligence）是一款由京东 DNN Lab 开发的、用于京东客服工作的即时聊天软件，可用于京东商城的售前咨询、售后服务等自动在线问答软件[42~44]（见图 2-4）。

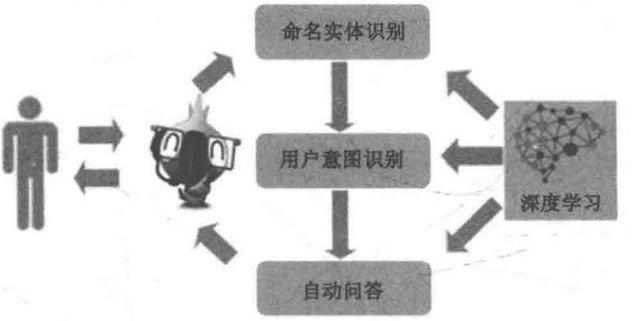


图 2-4 京东客服机器人 JIMI 基于深度学习的系统架构

目前，JIMI 已服务超过 1600 万的用户，其中，JIMI 在 2014 年“双十一”期间接待了近百万用户，有效缓解了人工客服的压力[43]。JIMI 中涉及如下技术：①自然语言处理：剖析用户语言，分析语义、情感、意图。②用户画像：分析用户基本资料、历史行为、动作轨迹，构造出用户的个人信息库，精准了解每个用户的性格、爱好、习惯，以便做出更好的服务[44]。③知识图谱：抽取知识相关性的关键词或商品。④机器学习。⑤深度学习：构建大型深层神经网络集群，模拟出人类的思维过程，通过上万神经节点的交叉计算达成高精度的智能意图识别及应答效果[44]。

2.6　腾讯在深度学习领域的研发现状

腾讯深度学习平台已在微信语音识别、微信图像识别中得到深入应用[45]。腾讯深度学习平台以 GPU 服务器为主，每台服务器配置 4 或者 6 块 NVIDIA Tesla GPU 卡，该深度学习平台重点研究多 GPU 卡的并行化技术，实现 DNN 的数据并行框架，以及 CNN 的模型并行和数据并行框架，在单机 6 GPU 卡配置下获得相比单卡 4.6 倍的加速，可在数日内完成数十亿高维度训练样本的 DNN 模型训练[45]（见图 2-5）。

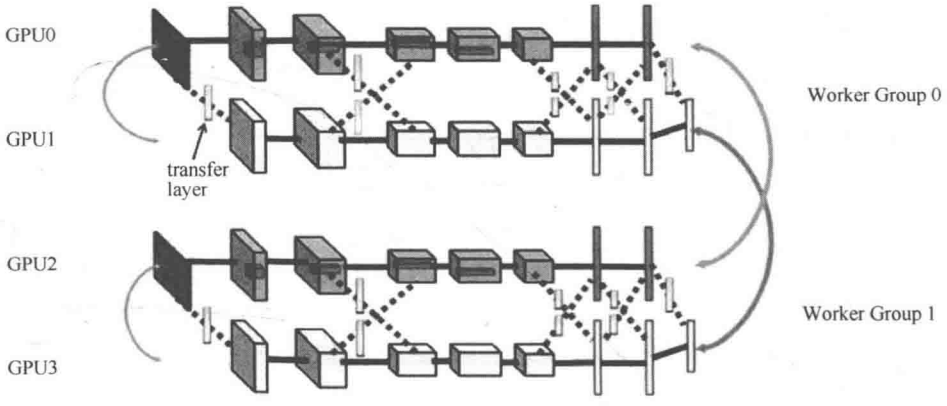


图 2-5　腾讯 Mariana 深度学习平台的 CNN GPU 框架的模型并行和数据并行架构

腾讯将其深度学习平台命名为 Mariana。针对腾讯多种应用，打造了三套框架[46]。

（1）Mariana DNN。深度神经网络的多 GPU 数据并行框架，主要用于微信语音识别。微信的语音包括超过 1 万个小时的训练数据、40 亿个样本，超过 5000 万个参数需要进行深度神经网络训练。

（2）Mariana CNN。深度卷积神经网络的多 GPU 并行框架，主要用于微信图像识别，对 2000 个类别、300 万个样本，6000 万个参数训练卷积神经网络，其 4GPU 模型+数据并行的加速比是单 GPU 的 2.5 倍[46]。

（3）Mariana Cluster。深度神经网络的 CPU 集群框架，用于广告点击率预估模型的训练，目前得到了初步应用[46]。

2.7 科创型公司（基于深度学习的人脸识别系统）

Face++[47]是北京旷视科技有限公司旗下的云端人脸识别平台，公司成立于 2011 年。支付宝人脸支付使用的技术就是 Face++（见图 2-6）。Face++在人脸检测 FDDB 评测、人脸关键点定位评测和人脸识别 LFW 评测上，拿到了这三项的世界第一。而 Face++的人脸关键点检测和人脸特征表示都是基于深度学习技术的。

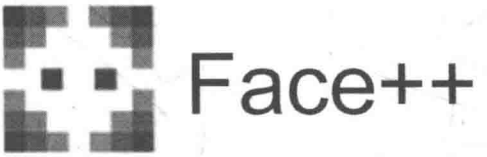


图 2-6 Face++：基于深度学习的云端人脸检测平台

Linkface[48]成立于 2014 年，是由四个 90 后“美女学霸”成立的人脸识别的创业公司，2014 年曾取得人脸检测 FDDB 评测第一名的成绩。Linkface 开发了基于深度学习的人脸检测创新算法，在人群、侧脸、遮挡、模糊等情景中，均能进行精准检测[49]。2016 年，Linkface 与 Sensetime（商汤科技）[50]合并。Sensetime 是专注于计算机视觉和深度学习的创新型公司，以基于深度学习的人脸识别为核心技术，该公司的产品已经应用到京东钱包、借贷宝、融 360、公安部第三研究所[50]等实际应用中。

2.8　深度学习的硬件支撑——NVIDIA GPU

Facebook、百度、京东、腾讯、阿里巴巴等公司通常都是使用英伟达的 GPU 作为其深度学习的硬件平台，尤其是 NVIDIA Tesla K40 是这些大公司广泛采用的 GPU 产品。谷歌在推出自己的深度学习芯片 TPU（Tensor Processing Unit）之前，使用的也是 NVIDIA 的 GPU。

2016 年 6 月，英伟达（NVIDIA）开始销售全球首款深度学习超级计算机 NVIDIA® DGX-1™[51]（见图 2-7）。NVIDIA DGX-1 深度学习系统包括 8 块 NVIDIA Tesla® P100 GPU，而该 GPU 基于全新的 NVIDIA Pascal™ GPU 架构。NVIDIA DGX-1 深度学习系统的吞吐量相当于 250 台基于 CPU 的服务器，拥有每秒高达 170 万亿次的半精度浮点运算峰值性能。所有这些 GPU 以及相应的网络、线缆和机架等都封装在一个机箱里，使得 GPU 间的通信更快[52]。

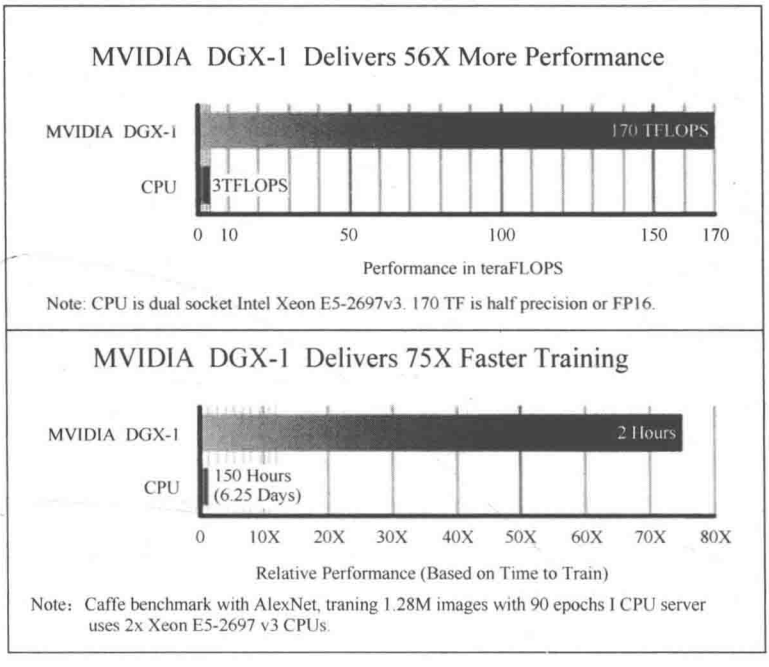


图 2-7　NVIDIA DGX-1 的浮点计算速度和深度学习训练速度分别是 CPU 的 56 倍和 75 倍

而在 2015 年，曙光联合 NVIDIA 发布了 X System 平台——包括深度学习软件 XSharp 和 XMachine 系列深度学习一体机。作为整个系统的硬件平台，XMachine 深度学习一体机特别为深度学习定制开发，提供多种类型 GPU 服务器供选择，原生态支持 NVIDIA DIGITS 开发环境，可大大降低用户进入深度学习领域的软件投入成本。借助 X System 深度学习一体机，用户可以快速进入深度学习领域[53]。

2015 年年底，Facebook 将其人工智能硬件平台 Big Sur 进行开源[54]（见图 2-8）。“Big Sur”装有 8 个 NVIDIA 的 Tesla M40，主板易于维修，十分适用于人工智能软件（尤其是深度学习）所需要的海量计算。

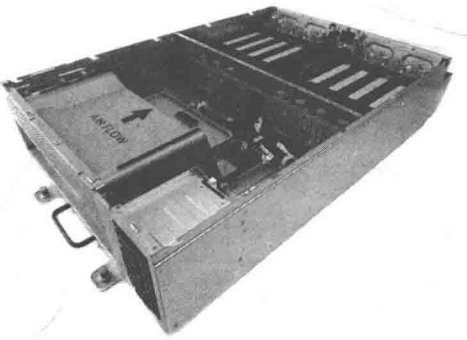


图 2-8　Facebook“Big Sur”服务器

参考文献

[1] 谷歌开放语音识别 API 发布机器学习云平台.http://tech.sina.com.cn/it/2016-03-24/doc-ifxqswxn6327980.shtml.

[2] Jeff Dean. Large-Scale Deep Learning for Intelligent Computer Systems. http://www.wsdm-conference.org/2016/slides/WSDM2016-Jeff-Dean.pdf.

[3] Google 首席科学家谈 Google 是怎么做深度学习的. http://www.huxiu.com/article/143219/1.html.

[4] DeepMind 联合创始人：AlphaGo 之后，AI 拯救落后医疗. http://chuansong.me/n/429408051962.

[5] 谷歌深度学习 DeepMind 与 NHS 合作用人工智能学习诊断眼部疾病. http://www.fromgeek.com/latest/36793.html.

[6] Tensorflow, GitHub. https://github.com/tensorflow/tensorflow.

[7] 如何评价Google发布的第二代深度学习系统TensorFlow? https://www.zhihu.com/question/37243838.

[8] Google 推出分布式深度学习系统 TensorFlow，能像 Android 一样带来 AI 复兴? http://36kr.com/p/5045960.html.

[9] 谷歌专为深度学习设计芯片 TPU. http://toutiao.com/i6286429867810488834.

[10] 相较传统 CPU，Google 的这款 AI 芯片能带来什么? http://dataunion.org/24275.html.

[11] Torchnet, Github. https://github.com/torchnet/torchnet.

[12] Torchnet: An Open-Source Platform for (Deep) Learning Research. https://lvdmaaten.github.io/publications/papers/Torchnet_2016.pdf.

[13] Facebook 开源深度学习框架 Torchnet，加快 A.I 研究步伐. http://36kr.com/p/5048573.html.

[14] Facebook 开源的 Torchnet 是什么? http://mt.sohu.com/20160706/ n458108904.shtml.

[15] Ahmad Abdulkader, Aparna Lakshmiratan, Joy Zhang. Introducing DeepText: Facebook's text understanding engine.

[16] Facebook 发布 Deep Text 是搜索引擎的一次进化. http://news.hexun.com/2016-06-04/184238150.html.

[17] 浅析 Facebook 文字理解引擎 DeepText. http://www.d1net.com/uc/company/421115.html.

[18] DeepText：Facebook 的文本解析引擎. http://www.iteye.com/news/31627.

[19] 都大龙，余轶南，罗恒. 基于深度学习的图像识别进展：百度的若干实践[J]. 中国计算机学会通讯. 2015（4）.

[20] 百度推出自然场景 OCR. http://idl.baidu.com/IDL-news-1.html.

[21] 商品图像搜索. http://idl.baidu.com/IDL-news-23.html.

[22] 大数据时代 百度机器学习算法受追捧. http://big5.xinhuanet.com/gate/big5/news.xinhuanet.com/info/2014-03/21/c_133204365.htm.

[23] 百度技术沙龙—广告数据上的大规模机器学习. http://blog.csdn.net/three_body/article/details/24913343.

[24] 百度识图. http://idl.baidu.com/IDL-news-22.html.

[25] 百度人脸识别“准冠全球”获双料世界第一. https://developer.baidu.com/announcement/182.

[26] 吴恩达：百度人脸识别技术已超越谷歌. http://tech.ifeng.com/a/20150322/41019389_0.shtml.

[27] 百度开发出深度语音识别系统 Deep Speech. http://tech2ipo.com/100100.

[28] 吴恩达谈百度深度学习：为什么要建中文神经网络. http://m.leiphone.com/news/201502/lWC2uwpDwJire12I.html.

[29] 李彦宏：大数据和云计算都不是互联网的下一幕，人工智能才是. http://weibo.com/ ttarticle/p/show?id=2309351000553984086938058220.

[30] 百度 PRNN：增强 GPU 伸缩性，RNN 训练最高提速 30 倍. http://geek.csdn.net/news/detail/84745.

[31] Distributed (Deep) Machine Learning Community. https://github.com/dmlc.

[32] DMLC 深盟分布式深度机器学习开源平台解析. http://www.csdn.net/article/2015-05-21/2824742.

[33] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. In Neural Information Processing Systems, Workshop on Machine Learning Systems, 2015.

[34] MXNet, Github. https://github.com/dmlc/mxnet.

[35] MXNet. http://mxnet.rtfd.org.

[36] 百度开源的 Warp-CTC 人工智能技术，到底是什么鬼? http://chengzhe.baijia.baidu.com/article/300536.

[37] 百度开源人工智能代码 Warp-CTC 机器训练提速百倍. http://it.sohu.com/20160118/n434910586.shtml.

[38] 阿里研究员华先胜：图像搜索的前世今生. http://www.36dsj.com/archives/50681.

[39] 华先胜. 图像验证码和大规模图像识别技术. http://www.infoq.com/cn/articles/CAPTCHA-image-recognition?from=timeline&isappinstalled=1.

[40] 阿里小蜜：语音识别、语义分析、深度学习在手机淘宝的实战分享. http://www.techweb.com.cn/news/2016-05-12/2331402.shtml.

[41] 阿里推人工智能客服“小蜜”对着它说话还能充话费. http://www.admin5.com/article/20160331/655269.shtml.

[42] 京东 DNN 实验室：大数据、深度学习与计算平台的实践. http://www.csdn.net/article/2015-08-04/2825376.

[43] 京东 DNN Lab 首席科学家：用深度学习搞定 80%的客服工作. http://www.csdn.net/article/1970-01-01/2823378.

[44] 深度学习与京东智能客服机器人 JIMI. http://www.useit.com.cn/thread-8356-1-1.html.

[45] 腾讯深度学习平台亮相机器学习顶级会议 ICML2014. http://data.qq.com/article?id=1553.

[46] Mariana——腾讯深度学习平台的进展与应用. http://www.36dsj.com/archives/20222.

[47] Face++: Leading Face Recognition on Cloud. http://www.faceplusplus.com/.

[48] Linkface. https://www.linkface.cn/.

[49] 世界第一的人脸检测技术背后的四个 90 后美女学霸. http://tech2ipo.com/100100.

[50] 商汤科技. http://www.sensetime.com/.

[51] The World's First Deep Learning Supercomputer in a box. http://www.nvidia.com/object/deep-learning-system.html.

[52] NVIDIA 发布全球首款深度学习超级计算机. http://www.nvidia.cn/object/world-s-first-deep-learning-supercomputer-cn.html.

[53] 联手 NVIDIA 曙光推深度学习整体解决方案. http://www.d1net.com/server/vendor/359857.html.

[54] Facebook releases design for souped-up artificial intelligence server, 'Big Sur'. http://www.pcworld.com/article/3014321/hardware/facebook-makes-its-big-sur-ai-server-design-available-to-anyone.html.

深度学习理论篇

第 3 章

神经网络

神经网络是一种经典的机器学习算法，随着对神经网络研究的不断深入，目前，它在模式识别、物体检测、视频分析和图像识别等领域发挥着越来越重要的作用。

3.1 神经元的概念

神经元是构成神经网络的基本单元，其基本模型（见图 3-1）。可以表示为

$$y_k = f\left(\sum_{i=1}^{n} w_{ik} \times x_i + b_k\right) \tag{3.1}$$

其中 $f(\cdot)$ 可以是 sigmoid 函数（$f(x)=\dfrac{1}{1+e^{-x}}$，域值范围为[0，1]），也可以双曲线正切（$f(x)=\dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$，域值范围为[−1，1]）等非线性函数。sigmoid 函数被看成一个挤压函数，它可以将一个较大输入范围挤压到较小的 0～1 的区域。sigmoid 和双曲线正切函数的函数图像（见图 3-2）。y_k 的值是由输入和对应的权值进行线性求和，然后再加上偏置值 b_k，再使用激活函数 $f(\cdot)$ 获得的。

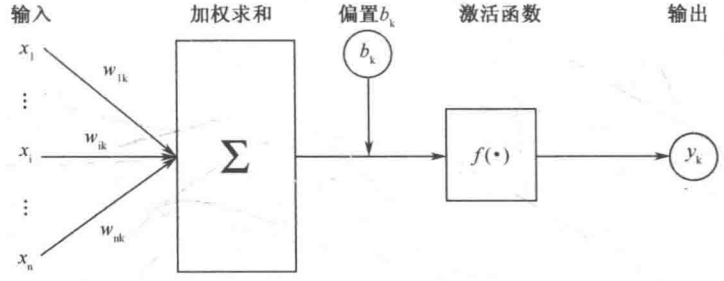


图 3-1　神经元模型

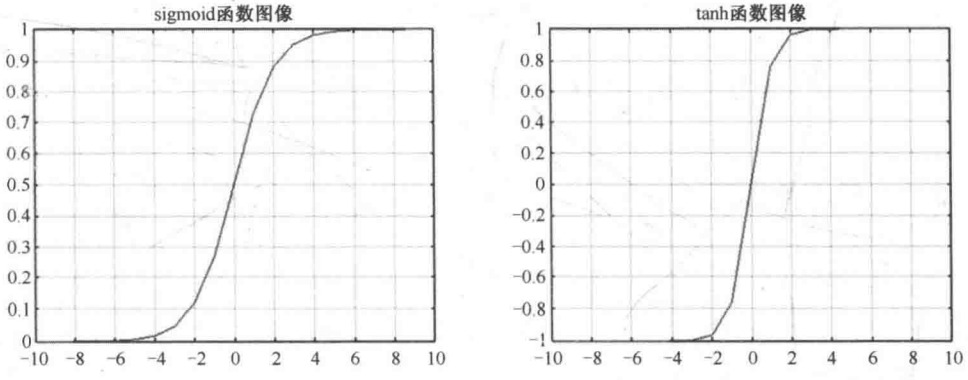


图 3-2　sigmoid 和双曲线正切函数的函数图像

3.2　神经网络

人工神经网络（Artificial Neural Network，ANN）是多层感知器（MultiLayer Perception，MLP）。神经网络包括输入层、一层或若干隐藏层以及输出层（见图 3-3），该多层网络结构包括 1 个输入层、2 个隐藏层和 1 个输出层，其基本组成单元为神经单元，如 x_1 等。

如果神经网络中的隐藏层足够多，该神经网络可以逼近任何函数。但由于网络结构比较复杂，会造成过拟合现象，或在误差后向传播过程中产生梯度弥散现象。

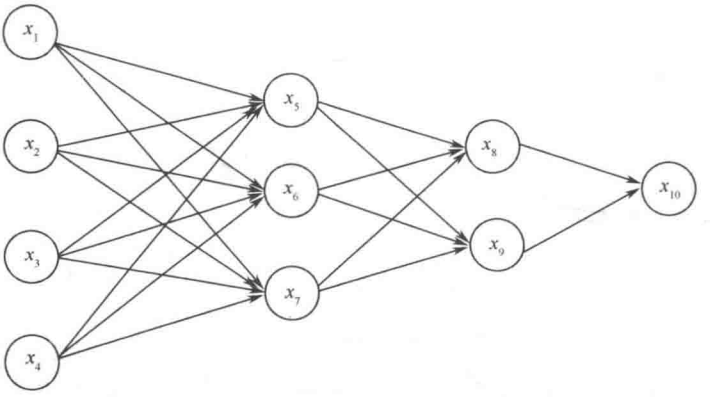


图 3-3　多层网络结构示意

3.2.1　后向传播算法

神经网络的训练包括正向传播和反向传播两个过程。正向传播是输入信号从输入层，经过若干隐层，从输出层输出结果（预测）的过程；误差后向传播是将误差信号从输出层反向传播至输入层的过程。反向传播主要使用误差后向传播（Error Back Propagation，EBP）算法和梯度下降对网络各层调整权重，通过比较输出信号和期望信号得到误差信号，利用链式求导将误差信号逐层向前传播得到各层误差信号，根据各层误差信号调整各层权重和相关参数。而不断调整权重和相关参数的过程就是人工神经网络训练学习的过程。

假设第 i 个单元与第 j 个单元有连接，则相关符号说明如下：

b_j 表示与第 j 个单元相关的偏置值；

$I_j = \left(\sum_i w_{ij} O_i + b_j\right)$ 表示第 j 个单元的输入值；

O_j 表示第 j 个单元的输出值；

$f(.)$ 表示神经元激活函数；

w_{ij} 表示第 i 个单元与第 j 个单元之间的权值；

b_i 表示与第 i 个神经单元相关的偏置值；

λ 表示学习率；

δ_i 表示与第 i 个神经单元相关的误差项；

$\text{nextlayer}(j)$ 表示下一层中与第 j 个神经单元相连的神经单元集合。

使用 sigmoid 激活函数的神经网络的后向传播算法整体思想如下[1]。

While 不满足终止条件：{
遍历训练集中的每一个样例：
#1. 将每个样例输入沿着网络向前传播计算：
输入层没有计算操作，所以输入层的输出等于输入：

$$O_j = I_j \tag{3.2}$$

隐藏层或输出层输出：

$I_j = \sum_i w_{ij} \times O_i + b_j$ //根据与单元 j 相连的单元，计算单元 j 的输入

$O_j = f(I_j)$ //使用激活函数（sigmoid 函数）计算单元 j 的输出　(3.3)

#2. 使用误差项后向传播
输出单元 j 误差项的计算：

$$\delta_j = O_j \times (1 - O_j) \times (T_j - O_j) \tag{3.4}$$

隐藏单元 j 误差项的计算：

$\delta_j = O_j \times (1 - O_j) \times \sum_{k \in \text{nextlayer}(j)} \delta_k \times w_{jk}$ //k 为下一层与隐藏单元 j 相连的单元　(3.5)

权重更新：

$$\Delta w_{ij} = \lambda \times \delta_j \times O_i$$
$$w_{ij} = w_{ij} + \Delta w_{ij} \tag{3.6}$$

偏置更新：

$$\Delta b_j = \lambda \times \delta_j$$
$$b_j = b_j + \Delta b_j \tag{3.7}$$

}

上面算法中的终止条件为人为设定的条件，可以是达到指定的迭代次数，也可以是网络收敛到一定程度等。终止条件的选择至关重要，例如终止条件为迭代次数时，迭代次数过少，网络误差降低幅度很小；迭代次数过多，有可能会导致网络出现过拟合现象。上面算法的推导过程将在 3.2.2 节详细讲解。

3.2.2　后向传播算法推导

本节主要介绍后向传播算法的推导过程，让读者能更清晰地了解权值和偏置更新。假设某个样例为 d，该样例对应的误差项设为 E_d，则 E_d 的计算为式（3.2）。

$$E_d = \frac{1}{2} \sum_{m \in \text{outs}} (T_m - O_m)^2 \tag{3.8}$$

其中，outs 表示输出层中输出单元的集合。O_m 表示第 m 个单元的输出值。T_m 表

示样例 d 的第 m 个单元对应的真实目标值。

权重改变量 Δw_{ij} 和偏置改变量 Δb_j 为：

$$\begin{aligned}\Delta w_{ij} &= -\lambda \times \frac{\partial E_{\mathrm{d}}}{\partial w_{ij}} \\ &= -\lambda \times \frac{\partial E_{\mathrm{d}}}{\partial I_j} \times \frac{\partial I_j}{\partial w_{ij}} \\ &= -\lambda \times \frac{\partial E_{\mathrm{d}}}{\partial I_j} \times \frac{\partial \left(w_{ij} \times O_i + b_j\right)}{\partial w_{ij}} \\ &= -\lambda \times \frac{\partial E_{\mathrm{d}}}{\partial I_j} \times O_i\end{aligned} \tag{3.9}$$

$$\begin{aligned}\Delta b_j &= -\lambda \times \frac{\partial E_{\mathrm{d}}}{\partial b_j} \\ &= -\lambda \times \frac{\partial E_{\mathrm{d}}}{\partial I_j} \times \frac{\partial I_j}{\partial b_j} \\ &= -\lambda \times \frac{\partial E_{\mathrm{d}}}{\partial I_j} \times \frac{\partial \left(w_{ij} \times O_i + b_j\right)}{\partial b_j} \\ &= -\lambda \times \frac{\partial E_{\mathrm{d}}}{\partial I_j}\end{aligned} \tag{3.10}$$

（1）输出单元的权值和偏置更新策略

由链式求导法则可知：

$$\frac{\partial E_{\mathrm{d}}}{\partial I_j} = \frac{\partial E_{\mathrm{d}}}{\partial O_j} \times \frac{\partial O_j}{\partial I_j} \tag{3.11}$$

式（3.4）中，只有当 $k=j$ 时，$\dfrac{\partial\left(\dfrac{1}{2}\sum\limits_{k\in \mathrm{outs}}\left(T_k - O_k\right)^2\right)}{\partial O_j} \neq 0$，则

$$\begin{aligned}\frac{\partial E_{\mathrm{d}}}{\partial O_j} &= \frac{\partial\left(\frac{1}{2}\sum\limits_{k\in \mathrm{outs}}\left(T_k - O_k\right)^2\right)}{\partial O_j} \\ &= \frac{1}{2}\frac{\partial\left(T_j - O_{\mathrm{j}}\right)^2}{\partial O_j} \\ &= -\left(T_j - O_j\right)\end{aligned} \tag{3.12}$$

由于 $O_j = f(I_j)$，则

$$\begin{aligned}\frac{\partial O_j}{\partial I_j} &= f(I_j)\times\left[1-f(I_j)\right] \\ &= O_j\times(1-O_j)\end{aligned} \tag{3.13}$$

由式（3.7）和式（3.8）可得，

$$\begin{aligned}\frac{\partial E_{\mathrm{d}}}{\partial I_j} &= \frac{\partial E_{\mathrm{d}}}{\partial O_j}\times\frac{\partial O_j}{\partial I_j} \\ &= -O_j\times(1-O_j)\times(T_j-O_j)\end{aligned} \tag{3.14}$$

从而可得，第 j 个单元的误差项的值为：

$$\begin{aligned}\delta_{\mathrm{j}} &= -\frac{\partial E_{\mathrm{d}}}{\partial I_j} \\ &= O_j\times(1-O_j)\times(T_j-O_j)\end{aligned} \tag{3.15}$$

（2）隐藏单元的权值和偏置更新策略

隐藏单元 j 的误差项主要由与它连接的下一层的所有单元的误差项决定。因此可通过以下方式进行计算。

$$\begin{aligned}\frac{\partial E_{\mathrm{d}}}{\partial I_j} &= \sum_{k\in\mathrm{nextlayer}(j)}\frac{\partial E_{\mathrm{d}}}{\partial I_k}\times\frac{\partial I_k}{\partial I_j} \\ &= \sum_{k\in\mathrm{nextlayer}(j)}(-\delta_k)\times\frac{\partial I_{\mathrm{k}}}{\partial I_j} \\ &= \sum_{k\in\mathrm{nextlayer}(j)}(-\delta_k)\times\frac{\partial I_k}{\partial O_j}\times\frac{\partial O_j}{\partial I_j} \\ &= \sum_{k\in\mathrm{nextlayer}(j)}(-\delta_k)\times\frac{\partial(w_{jk}O_j+b_k)}{\partial O_j}\times\frac{\partial O_j}{\partial I_j} \\ &= \sum_{k\in\mathrm{nextlayer}(j)}(-\delta_k)\times w_{jk}\times\frac{\partial O_j}{\partial I_j} \\ &= \sum_{k\in\mathrm{nextlayer}(j)}(-\delta_k)\times w_{jk}\times O_j\times(1-O_j)\end{aligned} \tag{3.16}$$

从而可得，第 j 个单元的误差项的值为：

$$\delta_j = -\frac{\partial E_d}{\partial I_j} = O_j \times (1 - O_j) \times \sum_{k \in \text{nextlayer}(j)} \delta_k \times w_{jk} \tag{3.17}$$

由式（3.9）、式（3.10）、式（3.15）和式（3.17）可知：

$$\Delta w_{ij} = \lambda \times \delta_j \times O_i$$

$$\Delta b_j = \lambda \times \delta_j$$

3.3 神经网络算法示例

本节通过两个示例讲解神经网络前向传播和后向传播的详细计算过程：例 3.3.1 讲解单样本输入时的计算过程。多样本的处理方式可参考本书 3.2.1 节，采用迭代训练和梯度下降来实现神经网络的整个训练过程。

例 3.3.1　设输入样例为$\{x_1, x_2, x_3, x_4\}=\{1, 0, 1, 1\}$，学习率 $l=0.8$，真实标签为 1，神经网络结构示例（见图 3-4），网络中相关权重和偏置信息如表 3-1 所示[1]。

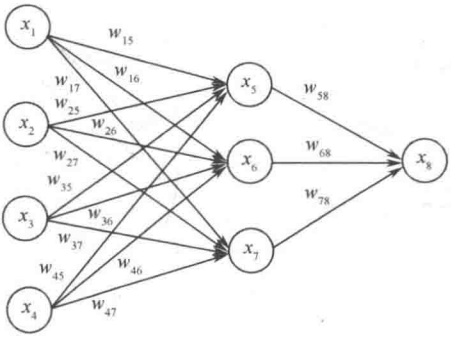


图 3-4　神经网络结构示例

表 3-1　网络中相关权重和偏置信息

w_{15}	0.5	w_{46}	0.5
w_{16}	0.2	w_{47}	−0.1
w_{17}	−0.5	w_{58}	0.6
w_{25}	−0.2	w_{68}	−0.2

续表

w_{26}	0.3	w_{78}	0.1
w_{27}	0.4	b_5	0.2
w_{35}	0.8	b_6	−0.5
w_{36}	−0.6	b_7	0.4
w_{37}	0.4	b_8	0.6
w_{45}	0.6		

根据式（3.2）和式（3.3）可以计算网络中每个单元的输入和输出值，如表 3-2 所示。

表 3-2　网络中每个单元的输入和输出值

神经单元	输入 I	输出 O
x_1	1	1
x_2	0	0
x_3	1	1
x_4	1	1
x_5	$1\times0.5+0\times(-0.2)+1\times0.8+1\times0.6+0.2=2.1$	$1/(1+e^{-2.1})=0.8909$
x_6	$1\times0.2+0\times0.3+1\times(-0.6)+1\times0.5-0.5=-0.4$	$1/(1+e^{0.4})=0.40131$
x_7	$1\times(-0.5)+0\times0.4+1\times0.4+1\times(-0.1)+0.4=0.2$	$1/(1+e^{-0.2})=0.54983$
x_8	$0.8909\times0.6+0.40131\times(-0.2)+0.54983\times0.1+0.6=1.1093$	$1/(1+e^{-1.1093})=0.752$

根据式（3.4）和式（3.5）可以计算网络中每个单元的误差项的值，如表 3-3 所示。

表 3-3　网络中每个单元的误差项的值

神经单元	误差项 δ
δ_{x8}	$0.752\times(1-0.752)\times(1-0.752)=0.04625$
δ_{x7}	$0.54983\times(1-0.54983)\times(0.1\times0.04625)=0.0011448$
δ_{x6}	$0.40131\times(1-0.40131)\times(-0.2\times0.04625)=-0.0022224$
δ_{x5}	$0.8909\times(1-0.8909)\times(0.6\times0.04625)=0.0026972$

根据式（3.6）和式（3.7）可以计算网络中更新后的权重值和偏置值，如表 3-4 所示。

表 3-4　网络中更新后的权重值和偏置值

权重和偏置	更新后的值
w_{15}	0.5+0.8×0.0026972×1=0.50216
w_{16}	0.2+0.8×(−0.0022224)×1=0.19822
w_{17}	(−0.5)+0.8×0.0011448×1=−0.49908
w_{25}	(−0.2)+0.8×0.0026972×0=−0.2
w_{26}	0.3+0.8×(−0.0022224)×0=0.3
w_{27}	0.4+0.8×0.0011448×0=0.4
w_{35}	0.8+0.8×0.0026972×1=0.80216
w_{36}	(−0.6)+0.8×(−0.0022224)×1=−0.60178
w_{37}	0.4+0.8×0.0011448×1=0.40092
w_{45}	0.6+0.8×0.0026972×1=0.60216
w_{46}	0.5+0.8×(−0.0022224)×1=0.49822
w_{47}	(−0.1)+0.8×0.0011448×1=−0.099084
w_{58}	0.6+0.8×0.04625×0.8909=0.63296
w_{68}	(−0.2)+0.8×0.04625×0.40131=−0.18515
w_{78}	0.1+0.8×0.04625×0.54983=0.12034
b_5	0.2+0.8×0.0026972=0.20216
b_6	(−0.5)+0.8×(−0.0022224)=−0.50178
b_7	0.4+0.8×0.0011448=0.40092
b_8	0.6+0.8×0.04625=0.637

参考文献

[1] Jiawei Han,Micheline Kamber, Jian Pei. Data Mining:Concepts and Techniques (Third Edition), Morgan Kaufmann Publisher. 2011.

第 4 章

卷积神经网络

随着计算机视觉的发展，深度学习技术也日益成熟，其中，卷积神经网络最为常见。卷积神经网络在图像识别、语音识别等领域发挥着越来越重要的作用。

4.1 卷积神经网络特性

卷积神经网络是一种经典的前馈神经网络，主要受生物学中的感受野的概念提出的，一个神经元由特定区域控制，只有这个特定区域才能激活该神经元[1]，包括正向传播和反向传播两个过程。正向传播是输入信号从输入层，经过若干隐层，从输出层输出的过程；反向传播是将误差信号从输出层反向传播至输入层的过程。反向传播主要使用误差后向传播（Error Back Propagation，EBP）算法和梯度下降对网络各层调整权重，通过比较输出信号和期望信号得到误差信号，利用链式求导将误差信号逐层向前传播得到各层误差信号，根据各层误差信号调整各层权重和相关参数，调整权重和相关参数的过程就是训练学习的过程。

卷积神经网络结构一般包括卷积层（Convolution 层）、下采样层（Pooling 层）和全连接层（Fully-Connection 层）；下采样层一般连在卷积层之后，与卷积层交替出现，最后连接全连接层。卷积神经网络采用局部连接、权值共享和空间或时间相关下采样方法，从而获得很好的平移、缩放和扭曲不变性，使提取的特征更具有区分性。

讲解卷积神经网络特性之前，首先说明几个常用概念之间的关系，网络结构中每层包含若干特征图（Feature Map）。特征图是图像在网络的不同层之间进行（下采样或卷积操作）运算后保存的结果（矩阵）。每个特征图中包含若干神经元，每个特征图对应一个过滤器（Filter），并对应一种特征。

4.1.1 局部连接

局部连接是卷积神经网络中比较重要的一个特性，由于图像本身的统计特性，图像一个区域的统计特性跟其他区域的统计特性相同，所以可利用同一个核对图像各个区域分别进行卷积运算。通过局部连接，能更好地抽象出图像的局部特征，例如图像的一些边缘信息。与全连接相比，局部连接能有效减少网络中相邻层的连接数，减少网络运算复杂度（见图 4-1）。

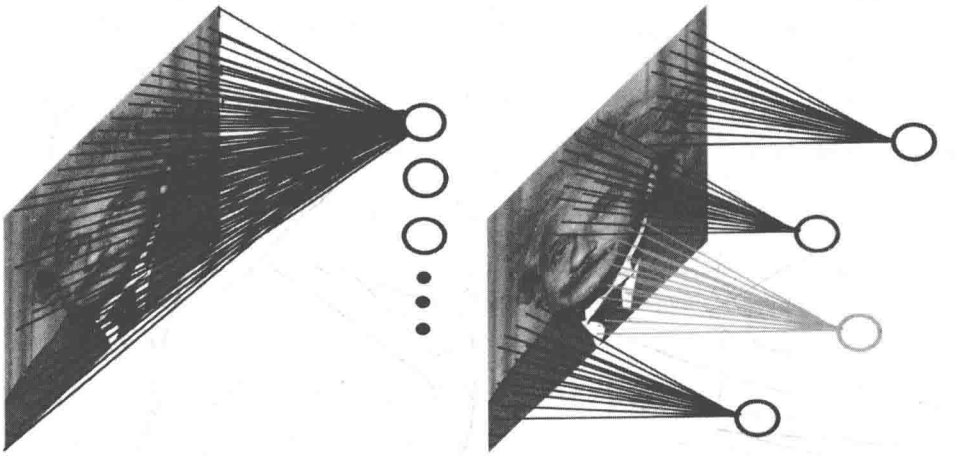


图 4-1　网络全连接方式（左）与局部连接方式（右）

假定图像尺寸为 400×400 像素，图 4-1（左）表示的是网络的全连接方式，图 4-1（右）表示的是网络的局部连接方式，局部连接方式使用一个一定尺寸的过滤器分别与下一层的神经元相连。设网络中有 10^6 个隐藏单元，那么图 4-1 左侧表示的全连接网络则需要 $400\times400\times10^6=1.6\times10^{11}$ 个连接；设局部连接网络中使用的局部区域尺寸为 10×10 像素，则图 4-1（右）的局部连接网络只需要 $10\times10\times10^6=10^8$ 个连接。

局部连接减少了网络结构中的连接数，有助于提高计算速度。如果图片尺寸很大，采用全连接方式对计算硬件要求较高，而采用局部连接不仅可以有效降低

计算复杂度，还可以降低对计算硬件的要求。从而可见对于大尺寸图片，局部连接的效果应该会更明显。

4.1.2　权值共享

为了减少可训练参数数目，卷积神经网络采用了权值共享的机制，同一个特征图使用相同的过滤器对前一层进行计算操作。使用一个过滤器会生成对应的一个特征图，也就相当于学习到图像的其中一个特征，为了学习图像的更多特征，可以通过使用多个不同的过滤器实现（见图 4-2）。

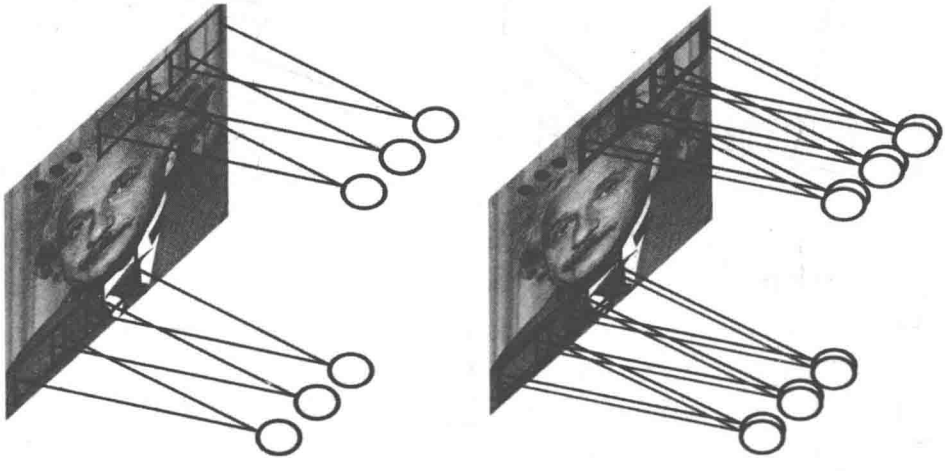


图 4-2　同一过滤器（左）与不同过滤器（右）[2]

假如图像尺寸为 400×400 像素，过滤器尺寸为 10×10 像素，图 4-2 左侧表示的是使用同一个过滤器学习图像特征，图 4-2 右侧表示的是使用不同过滤器学习图像特征。设网络中有 10^6 个隐层单元，那么图 4-1 左侧表示的全连接网络则需要 $400\times400\times10^6=1.6\times10^{11}$ 个参数；而图 4-2 左侧的局部连接网络只需要 $10\times10=100$ 个参数，从而可以看出权值共享机制大大减少了网络结构中的可训练参数的数目，由于该图中只使用了一个过滤器，并不能很好地表示图像特征，因此，需要通过使用不同的过滤器学习更丰富的图像特征；假设图 4-2 右侧中使用的过滤器数目为 10 个，则该层需要 $10\times10\times10=1000$ 个参数（过滤器数目与过滤器尺寸的乘积）。

权值共享机制可以大大减少网络结构中可训练参数的数目，减小了网络学习难度，同时可以实现并行训练。

4.1.3 空间相关下采样

基于图像空间局部相关性原理，卷积神经网络加入了下采样机制，下采样一般是紧跟在卷积层之后的操作，由于卷积层虽然通过局部连接的方式减少了网络结构中的连接数，但由于特征图数目的增加，实际上并没有减少网络中的神经元的个数，这使得特征维度过大，网络训练难度加大，并容易产生过拟合的现象，加入下采样机制后，可以有效降低特征维度并保留图像有效信息，加快网络训练速度，一定程度上避免过拟合现象。

目前，下采样机制策略有多种，例如最大值区域采样、平均值区域采样、求和区域采样和随机区域采样等。

4.2 卷积神经网络操作

4.2.1 卷积操作

卷积层是卷积神经网络中较为核心的网络层，主要进行卷积操作，基于图像的空间局部相关性分别抽取图像局部特征，通过将这些局部特征进行连接，可以形成整体特征。本节分单通道卷积和多通道卷积分别讲解卷积层操作，使读者对卷积能有清晰的理解。

1. 单通道卷积

如图 4-3 所示，假设卷积过程中滑动步长为 1，左侧表示图片大小为 5×5，设为矩阵 A；中间表示过滤器大小为 3×3，设为矩阵 B，右侧表示过滤器在图片上的卷积结果，设为矩阵 C，卷积后图片尺寸应为 [(5-3)/1+1] × [(5-3)/1+1] =3×3。假设矩阵 A、B、C 的下标都是从 1 开始的，则卷积计算过程为：

$C_{11}=A_{11}\times B_{11}+A_{12}\times B_{12}+A_{13}\times B_{13}+A_{21}\times B_{21}+A_{22}\times B_{22}+A_{23}\times B_{23}+A_{31}\times B_{31}+A_{32}\times B_{32}+A_{33}\times B_{33}=1\times 0+1\times 1+0\times 0+1\times 0+0\times 1+1\times 1+1\times 1+1\times 0+1\times 1=4$。其余结果，读者可以自行计算。

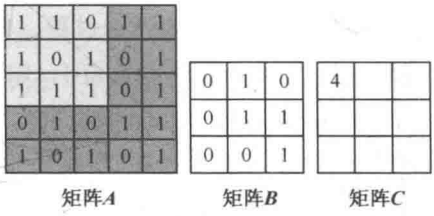


图 4-3　图片卷积过程

2. 多通道卷积

图 4-4 中网络结构由两层构成。第 m−1 层有 4 个特征图构成，第 m 层有 2 个特征图构成，卷积核大小为 2×2。W_{ij}^{kl} 表示第 m 层中第 k 个特征图，l 表示第 m−1 层中第 1 个特征图，i、j 表示过滤器中的值的索引位置。第 m 层中的 W^1 的值就是第 m−1 层中四个特征图中利用对应卷积核对相应区域进行卷积操作的总和加上偏置值。计算公式如式（4.1）所示，bias 表示偏置大小。

$$W^1 = \sum_{l=0}^{3} W_{ij}^{1l} \times x + \text{bias} \tag{4.1}$$

设输入图像尺寸为 $W_i \times H_i$，过滤器尺寸为 $w \times h$，滑动步长为 s，则输出图像尺寸 W_o 和 H_o 的计算公式如式（2.2）所示。

$$W_o \times H_o = [(W_i - w)/s + 1] \times [(H_i - h)/s + 1] \tag{4.2}$$

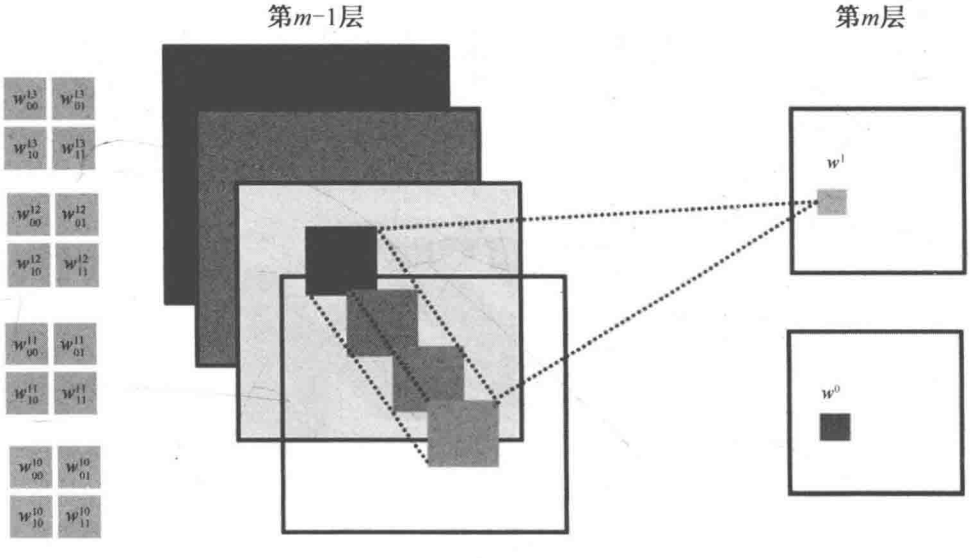


图 4-4　多通道卷积网络结构[3]

下面通过图示的方式介绍多通道卷积的详细过程。假设输入特征图个数为3，尺寸为5×5，过滤器尺寸为3×3，滑动步长为1，输出特征图个数为2。则可知过滤器组数应为2，过滤器总数应为2×3（输出特征图个数×输入特征图个数）=6，输出特征图的尺寸应为［(5-3) /1+1］×［(5-3) /1+1］=3×3。

如图4-5所示，输出组中第一个输出out[0,:,:]的值由输入和过滤器组中的第一个过滤器 w[0,:,:,:]卷积加上偏置值获得的，输出组中第二个输出 out[1,:,:]的值由输入和过滤器组中的第二个过滤器 w[1,:,:,:]卷积加上偏置值获得的。有兴趣的读者，可以手动演算、验证结果。

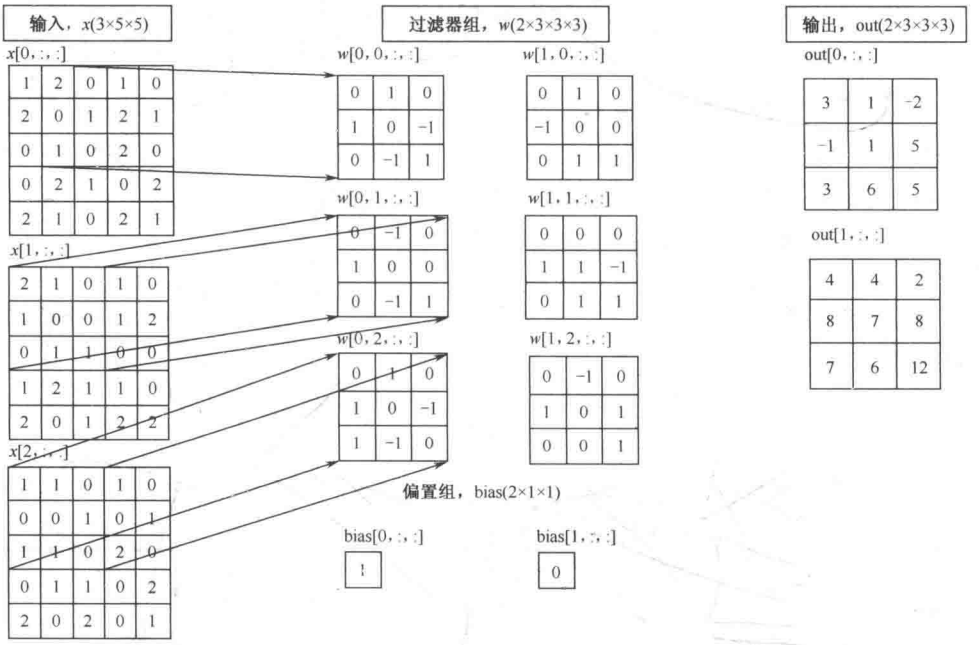


图4-5　多通道卷积计算[4]

4.2.2　下采样操作

下采样层主要是对卷积层的卷积结果进行下采样操作，主要用来降低维度并保留有效信息，一定程度上避免过拟合。

采样的策略有最大值区域采样、平均值区域采样、求和区域采样和随机区域采样等，此处操作中使用的策略是最大值区域采样。

在采样过程中，对图片边界的处理方式有两种，一种是考虑图片边界，另一种是不考虑图片边界。如图 4-6 所示，左侧表示图片大小为 5×5，采样区域大小为 2×2，无重叠区域采样，即在该例中采样滑动步长为 2，中间图片表示考虑边界的情况下的采样结果，右侧图片表示不考虑边界的情况下的采样结果。

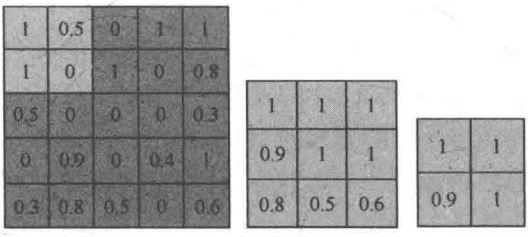


图 4-6　采样过程

4.3　卷积神经网络示例：LeNet-5

LeNet-5 是一种比较经典的卷积神经网络结构，在字符识别领域取得很好的效果，且美国多家银行都使用该系统用来识别银行支票上的数字。

本节主要讲解 LeNet-5 网络结构中各层所需的连接数以及可训练参数数目。网络结构图 4-7 中 C_x 代表卷积层，S_x 代表下采样层，F_x 代表全连接层，其中 x 表示各层的编号。除了输入层，该网络结构总共包含 7 层。网络结构中输入图片尺寸为 32×32 像素。各层的特征图尺寸都可根据式（4.2）计算出来。

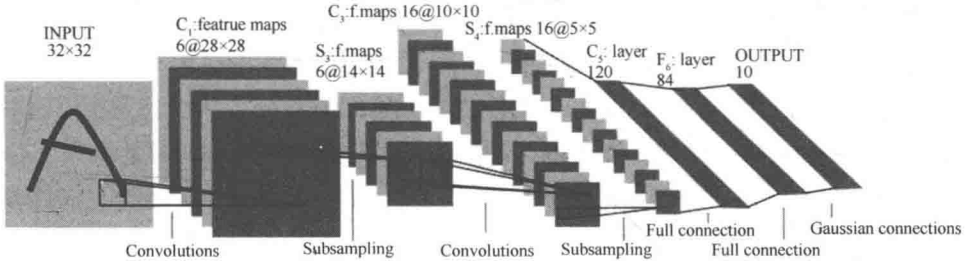


图 4-7　Lenet-5 网络结构[5]

C_1 层：该层是卷积层，包含 6 个特征图，特征图尺寸为 28×28，过滤器尺寸为 5×5，滑动步长为 1，由于该层中有 6 个特征图，每个特征图有 28×28 个

神经元，每个神经元与 5×5 个过滤器元素和 1 个偏置参数相连，因此该层包含 6×(28×28)×(5×5+1)=122 304 个连接数；由权值共享的特性可知，每个特征图中的所有神经元所使用的参数是相同的，因此每个特征图所需参数包括 5×5 个过滤器元素和 1 个偏置参数，包含 6×(5×5+1)=156 个可训练参数[5]。

S_2层：该层是下采样层，包含 6 个特征图，特征图尺寸为 14×14，过滤器尺寸为 2×2，滑动区域不重叠，滑动步长为 2。该层中的每个特征图中的每个神经元与 C_1 层中对应特征图中的 2×2 的采样区域连接，该层中每个神经元的值的计算过程为首先对与该神经元的连接的 C_1 层中的 2×2 的采样区域求和，然后再乘以一个可训练系数，再加上一个可训练偏置，最后将所得结果输入到 sigmoid 函数中得到最终结果。由于该层中有 6 个特征图，每个特征图有 14×14 个神经元，每个神经元与 2×2 个过滤器元素和 1 个偏置参数相连，因此该层包含 6×(14×14)×(2×2+1)=5880 个连接数；由权值共享的特性可知，每个特征图中的所有神经元所使用的参数是相同的，因此每个特征图所需参数包括 1 个可训练系数和 1 个偏置参数，包含 6×(1+1)=12 个可训练参数[5]。

C_3层：该层是卷积层，包含 16 个特征图，特征图尺寸为 10×10，过滤器尺寸为 5×5，滑动步长为 1。需要注意的是，该层中的每个特征图中的每个神经元并不是与 S_2 层中的所有特征图中的 5×5 的区域连接，而是 S_2 层中所有特征图、或者其中一部分特征图中的 5×5 的区域连接，C_3 层特征图与 S_2 层中特征图的连接情况如表 4-1 所示：C_3 层中第 0～5 个特征图与 S_2 层中连续的三个特征图相连，C_3 层中第 6～11 个特征图与 S_2 层中连续的四个特征图相连，C_3 层中第 12～14 个特征图与 S_2 层中不连续的四个特征图相连，C_3 层中第 15 个特征图与 S_2 层中所有特征图相连，稀疏连接可以有效减少网络中的连接数，通过不同的输入组合，可以得到更多不同的特征。由于该层中有 16 个特征图，每个特征图有 10×10 个神经元，每个特征图与 S_2 层中特征图连接的个数不同，则需要分别计算，该层包含 6×（10×10）×［3×（5×5）+1］+6×（10×10）×［4×（5×5）+1］+3×（10×10）×［4×（5×5）+1］+1×（10×10）×［6×（5×5）+1］=45600+60600+30300+15100=151600 个连接数和 6×［3×（5×5）+1］+6×［4×（5×5）+1］+3×［4×（5×5）+1］+1×［6×（5×5）+1］=1516 个可训练参数，偏置参数也属于可训练参数。

表 4-1　C_3 层中特征图与 S_2 层中特征图的连接说明

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

S_4 层：该层是下采样层，包含 16 个特征图，特征图尺寸为 5×5，过滤器尺寸为 2×2，滑动区域不重叠，滑动步长为 2，该层中的每个特征图中的每个神经元与 C_3 层中对应特征图中的 2×2 的采样区域连接，由于该层中有 16 个特征图，每个特征图有 5×5 个神经元，每个神经元与 2×2 个过滤器元素和 1 个偏置参数相连，因此该层包含 16×（5×5）×（2×2+1）=2000 个连接数；由权值共享的特性可知，每个特征图中的所有神经元所使用的参数是相同的，因此每个特征图所需参数包括 1 个可训练系数和 1 个偏置参数，包含 16×（1+1）=32 个可训练参数[5]。

C_5 层：该层是卷积层，包含 120 个特征图，特征图尺寸为 1×1，过滤器尺寸为 5×5，滑动步长为 1，虽然特征图尺寸为 1×1，但该层仍是卷积层，如果增大输入图像尺寸，该层的特征图尺寸也会随着发生改变。该层每个特征图中的每个神经元与 S_4 层中所有特征图中相应区域相连。由于该层中有 120 个特征图，每个特征图有 1×1 个神经元，每个神经元与 16×（5×5）个过滤器元素和 1 个偏置参数相连，因此该层包含 120×（1×1）×［16×（5×5）+1］=48120 个连接数；由权值共享的特性可知，每个特征图中的所有神经元所使用的参数是相同的，因此每个特征图所需参数包括 16×（5×5）个过滤器元素和 1 个偏置参数，包含 120×［16×（5×5）+1］=48120 个可训练参数[5]。

F_6 层：该层是全连接层，包含 84 个神经单元，该层中神经元的值是通过双曲正切函数计算输入向量和权重向量的点积得到的。由于该层中的每个神经元与 C_5 层中的所有特征图和 1 个偏置参数相连，因此该层包含 84×（120×1×1+1）=10164 个连接数；该层是全连接层，则包含 84×（120×1×1+1）=10164 个可训练参数[5]。

输出层：该层是输出层，包含 10 个神经元，该层也是分类层，每个神经元代表一类，每个神经元的输入为 F_6 层中所有 84 个神经元，该层的每个神经元的计算函数为欧式径向基函数（Euclidean Radial Basis Function），计算公式如式（4.3）：

$$y_i = \sum_j \left(x_j - w_{ij}\right)^2 \tag{4.3}$$

从公式中可以看出该层中每个神经元表示输入向量和参数向量之间的欧式距离，y_i 表示输出层的第 i 个神经元，x_j 表示 F_6 层中第 j 个神经元，w_{ij} 表示 F_6 层中第 j 个神经元与输出层的第 i 个神经元之间的连接权重。

参考文献

[1] Hubel D. H., Wiesel T. N. Receptive fields and functional architecture of monkey.

[2] http://yann.lecun.com.

[3] http://deeplearning.net/tutorial/lenet.html#lenet.

[4] http://cs231n.github.io/convolutional-networks/.

[5] Lécun Y., Bottou L., Bengio Y., et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11):2278-2324.

深度学习工具篇

第 5 章

深度学习工具 Caffe

Caffe [1]（Convolutional Architecture for Fast Feature Embedding）是最流行的深度学习框架之一，具有使用方便、层次分明、运行速度快等优点。本章将详细讲解 Caffe 安装、Caffe 中所涉及的网络数据层（数据输入）、网络层（线性操作）、网络结构（由许多层构成的网络结构）、网络参数（定义网络的相关参数）。更重要的是，本章将对 Cafee 各层对应的源代码进行解析。

5.1 Caffe 的安装

本节将详细讲解如何安装深度学习工具 Caffe，所用系统为 Ubuntu14.04 64bit（位），CUDA 版本为 CUDA7.0。Caffe 平台的安装方法请参考链接为 http://caffe.berkeleyvision.org/installation.html 和 http://blog.csdn.net/wangpengfei163/article/details/50488079。

注：本书中第 7～10 章使用 Caffe 的版本链接为：https://github.com/BVLC/caffe；第 11～13 章使用 Caffe 版本的链接为：https://github.com/BVLC/caffe/archive/v0.999.tar.gz。

5.1.1 安装依赖包

首先安装所需的基本依赖包：

```
sudo apt-get install build-essential # 安装必须包
sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libboost-all-dev lib
hdf5-serial-dev libgflags-dev libgoogle-glog-dev liblmdb-dev protobuf-compiler
# 安装 Caffe 依赖包
```

5.1.2 CUDA 安装

在安装 CUDA 的过程中，会同时安装 NVIDIA 显卡驱动。

1. CUDA 安装

1）离线*.deb 安装方法（推荐）

（1）下载对应系统的离线 CUDA 安装包（×.deb），链接：https://developer.nvidia.com/cuda-toolkit

（2）安装下载到后缀名为.deb 的 CUDA 离线包（cuda-repo-ubuntu1404-7-0-local_7.0-28_amd64.deb）

然后执行如下命令：

```
#添加软件源
sudo dpkg -i cuda-repo-<distro>_<version>_<architecture>.deb

#更新软件源
sudo apt-get update

#安装 CUDA
sudo apt-get install cuda

#重启计算机（通过 boot 设置独立显卡支持）
sudo reboot
```

2）离线 *.run 安装方法

（1）下载对应系统的离线 CUDA 安装包 （*.run），链接：https://developer.nvidia.com/cuda-toolkit

（2）安装下载到后缀名为.run 的 CUDA 离线包（cuda_7.0.28_linux.run）

```
sudo sh cuda_7.0.28_linux.run
```

运行过程中一直选择 accept，直至安装完成。

2. 修改环境变量和库路径

（1）在 /etc/profile 文件中添加以下内容

export PATH=/usr/local/cuda-7.0/bin:$PATH

命令：sudo vim /etc/profile

使环境变量生效

命令：source /etc/profile

（2）添加 lib 库路径

在 /etc/ld.so.conf.d/文件夹下添加 cuda.conf 文件，内容如下：

/usr/local/cuda-7.0/lib64

使库路径立即生效

命令：sudo ldconfig [-v, 可选]

3. 编译 CUDA Samples

（1）进入 samples 文件夹

命令：cd /usr/local/cuda-7.0/samples

（2）编译

命令：sudo make

（3）编译验证

进入 release 文件夹

命令：cd /usr/local/cuda-7.0/samples/bin/x86_64/linux/release

验证

命令：./deviceQuery

输出以下信息，则说明 CUDA 已安装。

```
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)
```

```
Detected 1 CUDA Capable device(s)

Device 0: "Tesla K40c"
  CUDA Driver Version / Runtime Version          7.0 / 7.0
  CUDA Capability Major/Minor version number:    3.5
  Total amount of global memory:                 11520 MBytes (12079136768 bytes)
  (15) Multiprocessors, (192) CUDA Cores/MP:     2880 CUDA Cores
  GPU Clock rate:                                745 MHz (0.75 GHz)
  Memory Clock rate:                             3004 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 1572864 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096,
4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Bus ID / PCI location ID:           1 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 7.0, CUDA Runtime Versio
n = 7.0, NumDevs = 1, Device0 = Tesla K40c
Result = PASS
```

5.1.3 MATLAB 和 Python 安装

1. MATLAB 安装

假设 MATLAB 安装路径为：/home/andywang/Tools/R2014b

假设 ISO 文件挂载路径为：/media/matlab2014b

（1）挂载 ISO 文件

把 MATLAB 的 ISO 文件挂载上去，为防止出现权限不够的问题，建议使用 root 用户。

创建 ISO 文件挂载路径，命令格式如下：

命令：sudo mkdir /media/matlab2014b

文件挂载命令格式如下：

命令：sudo mount -o loop,rw /home/R2014b_glnxa64.iso /media/matlab2014b/

media/matlab2014b 是要挂载的文件夹目录，建议读者新建一个文件夹。

（2）安装 MATLAB

进入挂载后的 MATLAB 文件夹/media/matlab2014b 中，运行 sudo ./install 将出现 Matlab 安装界面，如图 5-1 所示。

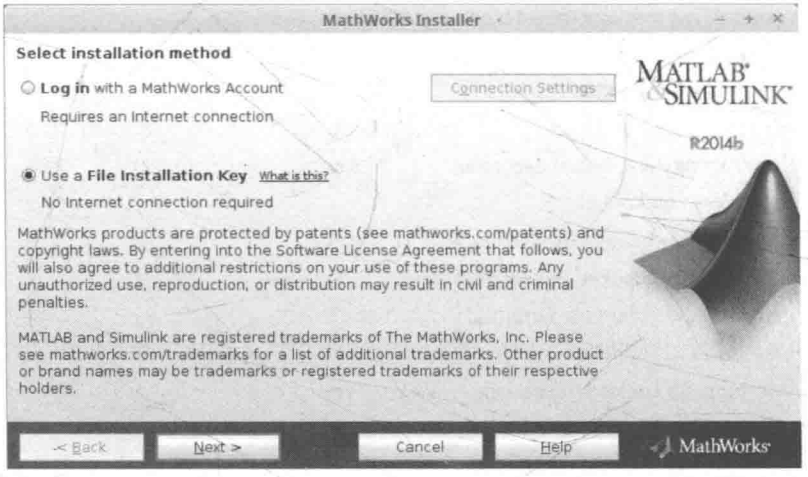


图 5-1 MATLAB 安装界面

两种方式可供选择，第一种需要连接互联网验证 MathWorks 用户名和密码。第二种方式不需要连接互联网，点击“Next”进入下一步操作，如图 5-2 所示。

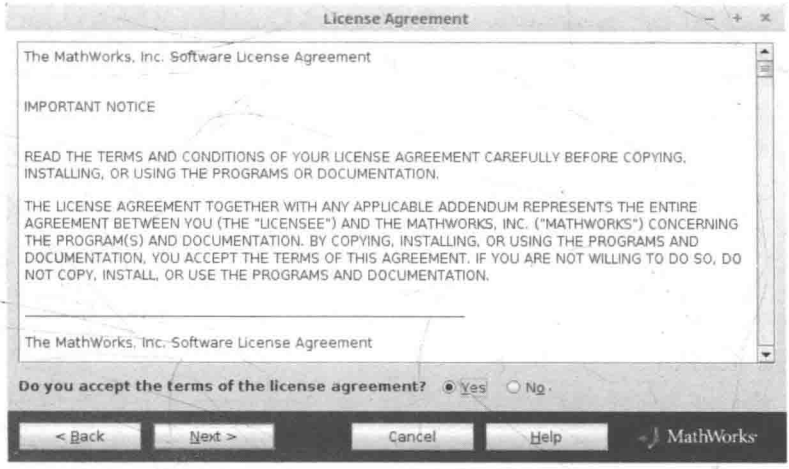


图 5-2　MATLAB R2014b 安装中的许可证协议选项

选择“Yes”选项，然后点击“Next”进入下一步操作。选择第一项，输入从官方获得的安装密钥进行安装，然后点击“Next”进入下一步操作，如图 5-3 所示。

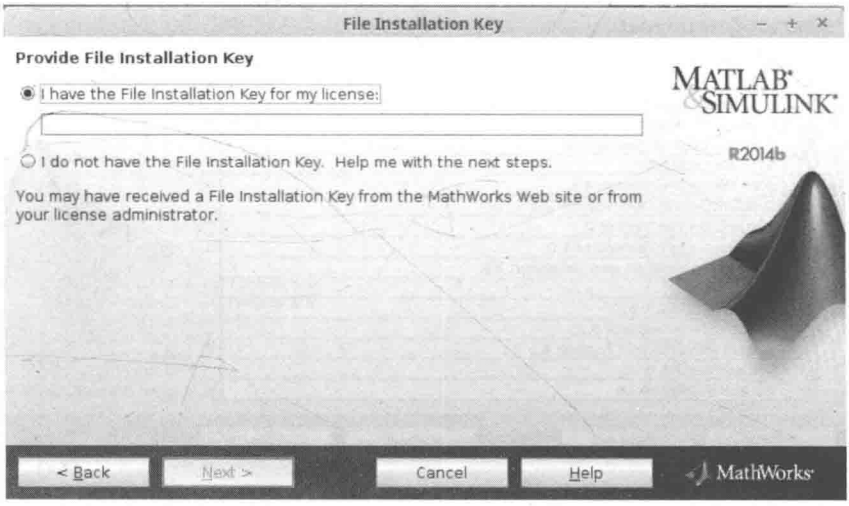


图 5-3　MATLAB R2014b 安装中的安装密钥选项

点击“Browser”按钮，选择 Matlab 软件的安装目录/home/andywang/Tools/R2014b，然后点击“Next”进入下一步操作，如图 5-4 所示。

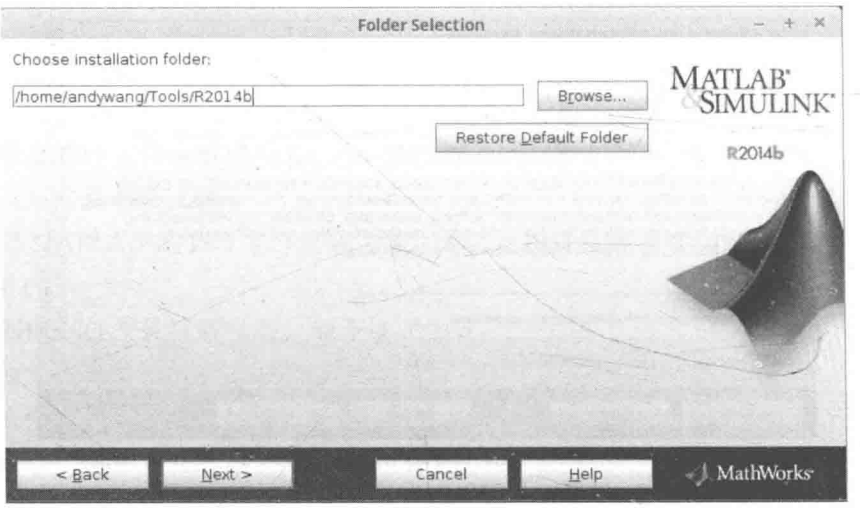


图 5-4　MATLAB R2014b 安装中选择安装目录的选项

该界面显示了该软件安装包所包含的所有功能，选择自己需要的功能进行安装，然后点击“Next”进入下一步操作，如图 5-5 所示。

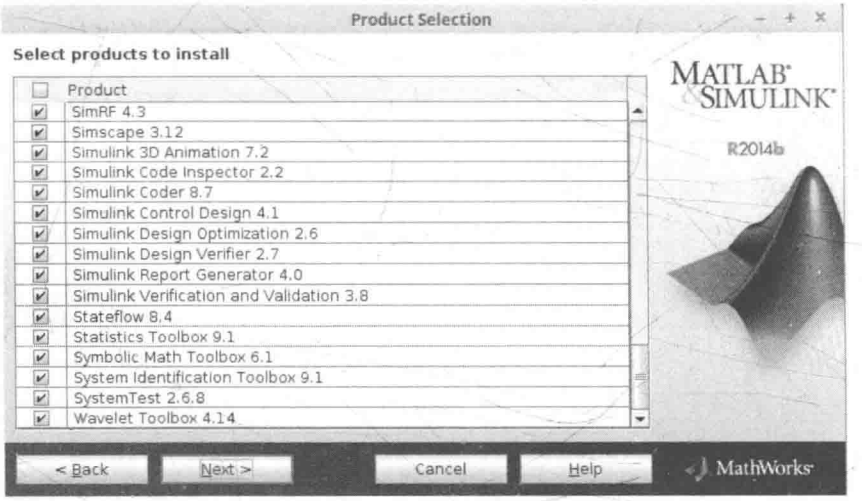


图 5-5　MATLAB R2014b 的所有产品列表，按需要自行选择

点击“Browser”按钮，选择可用的“License File”，然后点击“Next”进入下一步操作，如图 5-6 所示。

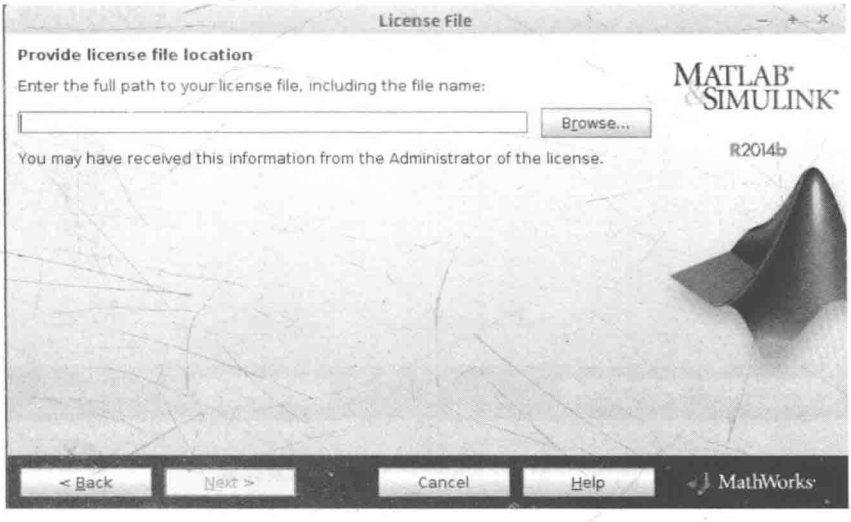


图 5-6　MATLAB R2014b 安装中的 License File 选项

该界面显示了软件安装目录，安装所需空间以及所安装的相关功能，然后点击“Install”安装软件，等待软件安装完成，如图 5-7 所示。

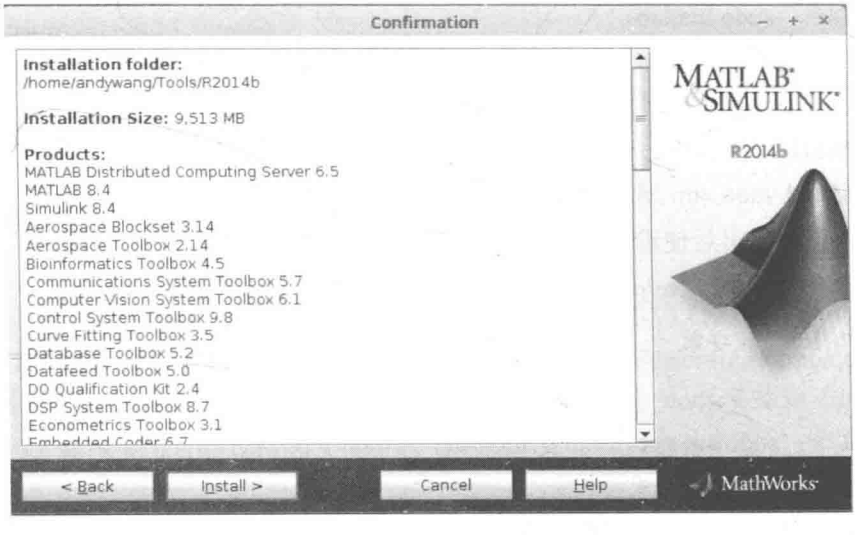


图 5-7　开始安装 MATLAB R2014b

至此，MATLAB R2014b 软件安装完成，如图 5-8 所示。

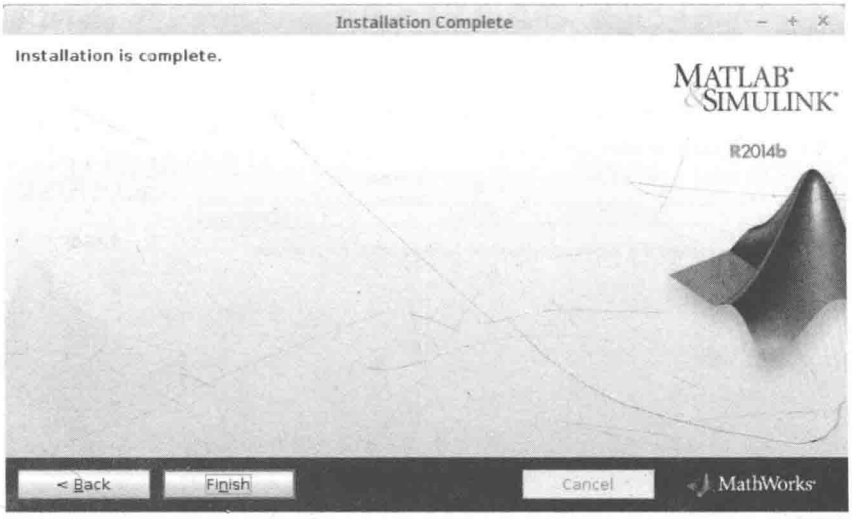


图 5-8　MATLAB R2014b 软件安装完成

（3）MATLAB R2014b 软件安装成功后，启动 Matlab R2014b 软件

进入/home/andywang/Tools/R2014b/bin 文件夹

命令：cd /home/andywang/Tools/R2014b/bin

启动 Matlab 软件

命令：sudo matlab

（4）配置环境变量，使得在命令行终端种直接输入 matlab 即可启动 Matlab 软件在/etc/profile 文件末尾添加 export PATH=/home/andywang/Tools/R2014b/bin:$PATH

命令：sudo vim /etc/profile

保存并退出后使设置生效

命令：source /etc/profile

2. Python 安装

（1）安装 Python

命令：sudo apt-get install python-dev python-pip

（2）安装 Python 依赖包

命令：sudo apt-get install python-numpy python-scipy python-matplotlib ipython ipython-notebook python-pandas python-sympy python-nose

5.1.4　OpenCV 安装（可选）

（1）下载安装脚本，链接为：https://github.com/bearpaw/Install-OpenCV

（2）进入目录 Install-OpenCV/Ubuntu/2.4

（3）执行脚本

命令：chmod +x *.sh

命令：sh ./opencv2_4_10.sh

5.1.5　Intel MKL 或者 BLAS 安装

如果没有申请到 Intel MKL，可以使用以下命令安装免费的 ATLAS

命令：sudo apt-get install libatlas-base-dev

如果申请到了 Intel MKL，解压下载到的文件，运行 install_GUI.sh，然后按照图形界面步骤安装即可。

安装完成后，需要执行下列两个步骤：

（1）在/etc/ld.so.conf.d/文件夹下添加 intel_mkl.conf 文件，内容如下：

/opt/intel/lib

/opt/intel/mkl/lib/intel64

（2）使库路径立即生效

sudo ldconfig [-v, 可选]

5.1.6　Caffe 编译和测试

（1）下载 Caffe 源码包，链接为：https://github.com/BVLC/caffe

（2）进入 caffe-master 文件夹目录，复制一份 Makefile.config.examples

命令：cp Makefile.config.example Makefile.config

（3）修改 Makefile.config 文件中相关路径，该文件内容如下：

```
## Refer to http://caffe.berkeleyvision.org/installation.html
# Contributions simplifying and improving our build system are welcome!

# cuDNN 加速开关
```

```
# cuDNN acceleration switch (uncomment to build with cuDNN).
# USE_CUDNN := 1

#是否只是用 CPU 模式
# CPU-only switch (uncomment to build without GPU support).
# CPU_ONLY := 1

# To customize your choice of compiler, uncomment and set the following.
# N.B. the default for Linux is g++ and the default for OSX is clang++
# CUSTOM_CXX := g++

#CUDA 目录
# CUDA directory contains bin/ and lib/ directories that we need.
CUDA_DIR := /usr/local/cuda-7.0
# On Ubuntu 14.04, if cuda tools are installed via
# "sudo apt-get install nvidia-cuda-toolkit" then use this instead:
# CUDA_DIR := /usr

# CUDA architecture setting: going with all of them.
# For CUDA < 6.0, comment the *_50 lines for compatibility.
CUDA_ARCH := -gencode arch=compute_20,code=sm_20 \
        -gencode arch=compute_20,code=sm_21 \
        -gencode arch=compute_30,code=sm_30 \
        -gencode arch=compute_35,code=sm_35 \
        -gencode arch=compute_50,code=sm_50 \
        -gencode arch=compute_50,code=compute_50

# BLAS choice:
# atlas for ATLAS (default)
# mkl for MKL
# open for OpenBlas
BLAS := atlas
# Custom (MKL/ATLAS/OpenBLAS) include and lib directories.
# Leave commented to accept the defaults for your choice of BLAS
# (which should work)!
# BLAS_INCLUDE := /path/to/your/blas
# BLAS_LIB := /path/to/your/blas
BLAS_INCLUDE := /usr/include/atlas
BLAS_LIB := /usr/lib/atlas-base
```

```
# This is required only if you will compile the matlab interface.
# MATLAB directory should contain the mex binary in /bin.
# MATLAB_DIR := /usr/local
# MATLAB_DIR := /APPlications/MATLAB_R2012b.app
#安装 MATLAB 目录
MATLAB_DIR := /stor/Matlab2014b

# NOTE: this is required only if you will compile the python interface.
# We need to be able to find Python.h and numpy/arrayobject.h.
PYTHON_INCLUDE := /usr/include/python2.7 \
        /usr/lib/python2.7/dist-packages/numpy/core/include
# Anaconda Python distribution is quite popular. Include path:
# Verify anaconda location, sometimes it's in root.
# ANACONDA_HOME := $(HOME)/anaconda
# PYTHON_INCLUDE := $(ANACONDA_HOME)/include \
        # $(ANACONDA_HOME)/include/python2.7 \
        # $(ANACONDA_HOME)/lib/python2.7/site-packages/numpy/core/include \

# We need to be able to find libpythonX.X.so or .dylib.
PYTHON_LIB := /usr/lib
# PYTHON_LIB := $(ANACONDA_HOME)/lib

# Uncomment to support layers written in Python (will link against Python libs)
# WITH_PYTHON_LAYER := 1

# Whatever else you find you need goes here.
INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include
LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib

# Uncomment to use `pkg-config` to specify OpenCV library paths.
# (Usually not necessary -- OpenCV libraries are normally installed in one of the above
$LIBRARY_DIRS.)
# USE_PKG_CONFIG := 1

BUILD_DIR := build
DISTRIBUTE_DIR := distribute

# Uncomment for debugging. Does not work on OSX due to
```

```
# https://github.com/BVLC/caffe/issues/171
# DEBUG := 1

# The ID of the GPU that 'make runtest' will use to run unit tests.
TEST_GPUID := 0

# enable pretty build (comment to see full commands)
Q ?= @
```

说明：如果选择使用 Intel MKL，则需要将 BLAS、BLAS_INCLUDE 和 BLAS_LIB 修改为

```
BLAS := mkl
BLAS_INCLUDE := /opt/intel/mkl/include
BLAS_LIB := /opt/intel/mkl/lib/intel64
```

（4）编译 Caffe

```
make all
make test
make runtest
```

（5）编译 MATLAB 接口

```
make matcaffe
```

（6）编译 Python 接口

```
make  pycaffe
```

至此，Caffe 工具的安装过程已经讲解完毕，希望读者能根据操作系统和软件版本解决安装过程中碰到的问题。

5.1.7 Caffe 安装问题分析

由于 CUDA 安装包中 NVIDIA 驱动的版本并不能保证是最新的，也不一定适合个人计算机的显卡版本，可以去 NVIDIA 官网下载对应的显卡驱动的最新版，至少要高于 CUDA 安装包中自带的 NVIDIA 驱动版本，重新安装 NVIDIA 显卡驱动。

5.2　Caffe 框架与源代码解析

本节以 AlexNet 网络结构[2]为例，它主要包括 5 个卷积层、3 个全连接层、下采样层（Pooling 层）、线性修正单元（ReLU 层[3]）以及标志所属类别的输出层。网络最后一个全连接层的输出需要设置为类别数，输入图片尺寸为 227×227 像素，输出为属于每个类别的概率值，如图 5-9 所示。我们选择网络结构中的倒数第二个全连接层作为要提取的深度特征层，由于该层后跟有线性修正单元（ReLU 层），所以提取出的深度特征向量都是非负值。

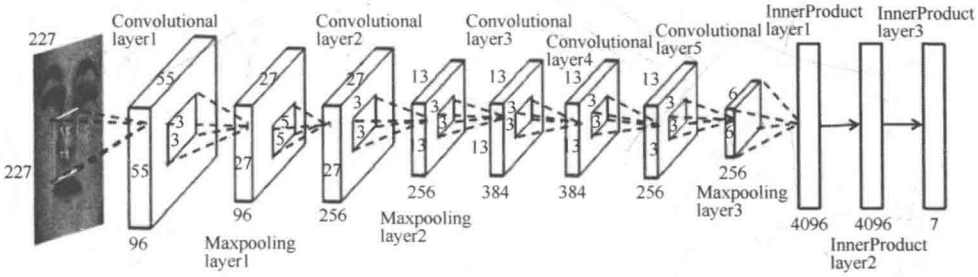


图 5-9　AlexNet 网络结构

每个立方体的长、宽和高表示特征图的数目和维度，在立方体中的正方形表示核尺寸，最后三层为全连接层，全连接层旁边的数字分别代表该层的神经元数目[2]。

Caffe 中对应各层的源文件一般包含后缀名为.cpp 和对应的后缀名为.hpp 文件，对于可以在 GPU 模式下计算的层，有对应的后缀名为.cu 文件。.cu 文件的定义跟.cpp 文件主要功能基本一致，只是不同的实现方式，本书只对.cpp 文件中的内容进行解读，有兴趣的读者可以尝试去阅读.cu 文件，这需要一些 CUDA 编程的基础知识。本部分主要参考文献[4]，有兴趣的读者可以详细阅读。

5.2.1　数据层解析

Caffe 中数据层是整个网络结构中的最底层，也是最关键的一层，它是数据

输入层，用来提供数据。Caffe 进行数据传递和处理的基本数据类型是 Blob 类型，数据来源可以是数据库文件，也可以是磁盘上的文件列表，常用的数据类型有 Data 和 ImageData。

1. Data 层

Data 层是按照 LMDB/LEVELDB 数据格式输入数据的，该层所利用的数据需要提前将图片文件数据转换成 LMDB/LEVELDB 格式。该层数据的保存结构为的四维结构（批处理大小、通道数、数据高度、数据宽度）。

（1）主要参数

source：表示数据文件来源，即文件保存位置。

batch_size：表示一次迭代过程所处理的图片数目。

backend：表示数据文件的格式是 LMDB 或 LEVELDB，默认是 LEVELDB。

（2）示例

```
layer {
  name: "data" #层属性：层名称
  type: "Data" #层属性：层类型
  top: "data" #层属性：上一层名称
  top: "label" #层属性：上一层名称
  include { #层参数：数据在训练阶段使用
    phase: TRAIN
  }
  transform_param { #层参数：数据转换参数
    mirror: true
    crop_size: 227
    mean_file: "data/ilsvrc12/imagenet_mean.binaryproto"
  }
  data_param { #层参数
    source: "examples/imagenet/ilsvrc12_train_lmdb" #层参数：数据来源文件
    batch_size: 256 #层参数：数据批量大小
    backend: LMDB #层参数：数据文件格式
  }
}
```

（3）源码解析

整体简介

该层对应的源代码文件为 data_layer.cpp，类定义文件为 data_layers.hpp，对应类名为 DataLayer，DataLayer 为 BasePrefetchingDataLayer，BaseDataLayer，

Layer 的继承类。该层的主要功能是首先调整数据尺寸与网络结构保持一致，数据处理（可采用批量处理的方式），将当前数据当作该层的输出。

主要函数

① DataLayerSetUp(const vector<Blob<Dtype>*>& bottom,vector<Blob <Dtype>*> *top)

该函数的主要功能初始化数据层的输入和输出。其中 7～14 行检查是否需要跳过一些数据，22～32 行进行图片剪切处理并设置顶层数据的维度。

```
1.  template <typename Dtype>
2.  void DataLayer<Dtype>::DataLayerSetUp(const vector<Blob<Dtype>*>& bottom,
3.          const vector<Blob<Dtype>*>& top) {
    // 初始化文件后缀处理
4.    db_.reset(db::GetDB(this->layer_param_.data_param().backend()));
5.    db_->Open(this->layer_param_.data_param().source(), db::READ);
6.    cursor_.reset(db_->NewCursor());

    // 检查是否需要跳过一些数据
7.    if (this->layer_param_.data_param().rand_skip()) {
8.      unsigned int skip = caffe_rng_rand() %
9.                                this->layer_param_.data_param().rand_skip();
10.     LOG(INFO) << "Skipping first " << skip << " data points.";
11.     while (skip-- > 0) {
12.       cursor_->Next();
13.     }
14.   }
    // 读取第一个数据来初始化 top
15.   Datum datum;
16.   datum.ParseFromString(cursor_->value());

17.   bool force_color = this->layer_param_.data_param().force_encoded_color();
18.   if ((force_color && DecodeDatum(&datum, true)) ||
19.         DecodeDatumNative(&datum)) {
20.     LOG(INFO) << "Decoding Datum";
21.   }
    // 图片剪切处理
22.   int crop_size = this->layer_param_.transform_param().crop_size();
23.   if (crop_size > 0) {
```

```
24.       top[0]->Reshape(this->layer_param_.data_param().batch_size(),
              datum.channels(), crop_size, crop_size);
25.       this->prefetch_data_.Reshape(this->layer_param_.data_param().batch_size(),
              datum.channels(), crop_size, crop_size);
26.       this->transformed_data_.Reshape(1, datum.channels(), crop_size, crop_size);
27.     } else {
28.       top[0]->Reshape(
29.                 this->layer_param_.data_param().batch_size(), datum.channels(),
              datum.height(), datum.width());
30.       this->prefetch_data_.Reshape(this->layer_param_.data_param().batch_size(),
              datum.channels(), datum.height(), datum.width());
31.       this->transformed_data_.Reshape(1, datum.channels(),
              datum.height(), datum.width());
32.     }
33.     LOG(INFO) << "output data size: " << top[0]->num() << ","
          << top[0]->channels() << "," << top[0]->height() << ","
        << top[0]->width();
        // 标签处理
34.     if (this->output_labels_) {
35.       vector<int> label_shape(1, this->layer_param_.data_param().batch_size());
36.       top[1]->Reshape(label_shape);
37.       this->prefetch_label_.Reshape(label_shape);
38.     }
39.   }
```

②InternalThreadEntry()

该函数的主要功能是创建一个批量读取数据并将数据进行一些简单的转换操作（如映射、缩放、剪切等）的线程，读取数据过程中会计算批处理所需时间，读数据所需时间，转换数据所需时间。

```
1.  template <typename Dtype>
2.  void DataLayer<Dtype>::InternalThreadEntry() {
3.    CPUTimer batch_timer;
4.    batch_timer.Start();
5.    double read_time = 0;
6.    double trans_time = 0;
7.    CPUTimer timer;
8.    CHECK(this->prefetch_data_.count());
9.    CHECK(this->transformed_data_.count());
```

```
    // 调整 batch_size=1 和 crop_size=0 时的图片尺寸
10.  const int batch_size = this->layer_param_.data_param().batch_size();
11.  const int crop_size = this->layer_param_.transform_param().crop_size();
12.  bool force_color = this->layer_param_.data_param().force_encoded_color();
13.  if (batch_size == 1 && crop_size == 0) {
14.    Datum datum;
15.    datum.ParseFromString(cursor_->value());
16.    if (datum.encoded()) {
17.      if (force_color) {
18.        DecodeDatum(&datum, true);
19.      } else {
20.        DecodeDatumNative(&datum);
21.      }
22.    }
23.    this->prefetch_data_.Reshape(1, datum.channels(),
24.        datum.height(), datum.width());
25.    this->transformed_data_.Reshape(1, datum.channels(),
26.        datum.height(), datum.width());
27.  }

28.  Dtype* top_data = this->prefetch_data_.mutable_cpu_data();
29.  Dtype* top_label = NULL;  // 使用 NULL 初始化变量

30.  if (this->output_labels_) {
31.    top_label = this->prefetch_label_.mutable_cpu_data();
32.  }
    // 遍历批处理中的所有数据
33.  for (int item_id = 0; item_id < batch_size; ++item_id) {
34.    timer.Start();
      // 获得当前数据块
35.    Datum datum;
36.    datum.ParseFromString(cursor_->value());

37.    cv::Mat cv_img;
38.    if (datum.encoded()) {
39.      if (force_color) {
40.        cv_img = DecodeDatumToCVMat(datum, true);
41.      } else {
42.        cv_img = DecodeDatumToCVMatNative(datum);
```

```
43.           }
44.           if (cv_img.channels() != this->transformed_data_.channels()) {
45.             LOG(WARNING) << "Your dataset contains encoded images with mixed "
              << "channel sizes. Consider adding a 'force_color' flag to the "
              << "model definition, or rebuild your dataset using "
              << "convert_imageset.";
46.           }
47.         }
48.         read_time += timer.MicroSeconds();
49.         timer.Start();

          // 数据转换 (映射、缩放、剪切等)
50.         int offset = this->prefetch_data_.offset(item_id);
51.         this->transformed_data_.set_cpu_data(top_data + offset);
52.         if (datum.encoded()) {
53.           this->data_transformer_->Transform(cv_img, &(this->transformed_data_));
54.         } else {
55.           this->data_transformer_->Transform(datum, &(this->transformed_data_));
56.         }
57.         if (this->output_labels_) {
            // 将当前数据块对应的标签保存在 top_label 变量中
58.           top_label[item_id] = datum.label();
59.         }
          // 记录转换所需时间
60.         trans_time += timer.MicroSeconds();
          // 使用 cursor_指针访问下一个数据块
61.         cursor_->Next();
62.         if (!cursor_->valid()) {
63.           DLOG(INFO) << "Restarting data prefetching from start.";
64.           cursor_->SeekToFirst();
65.         }
66.       }
67.       batch_timer.Stop();
68.       DLOG(INFO) << "Prefetch batch: " << batch_timer.MilliSeconds() << " ms.";
69.       DLOG(INFO) << "Read time: " << read_time / 1000 << " ms.";
70.       DLOG(INFO) << "Transform time: " << trans_time / 1000 << " ms.";
71.     }
```

2. ImageData 层

ImageData 层在输入该层中不需要将图片文件数据进行转换，这些转换操作可以在网络定义文件中指定，可以直接处理图片列表文件，图片列表文件格式为图片路径文件名和该图片对应的标签。

（1）主要参数

source：表示数据文件来源，即文件保存位置。

batch_size：表示一次迭代过程所出来的图片数目。

shuffle：表示是否打乱训练集或测试集的数据顺序，默认是 false。

（2）示例

```
layer {
  name: "data" #层属性：层名称
  type: "ImageData " #层属性：层类型
  top: "data" #层属性：上一层名称
  top: "label" #层属性：上一层名称
  include { #层参数：数据在训练阶段使用
    phase: TRAIN
  }
  transform_param { #层参数：数据转换参数
    mirror: true
    crop_size: 227
    mean_file: "data/ilsvrc12/imagenet_mean.binaryproto"
  }
  data_param { #层参数
    source: "examples/imagenet/image.txt" #层参数：数据来源文件
    batch_size: 256 #层参数：数据批量大小
  }
}
```

（3）源码解析

整体简介

该层对应的源代码文件为 image_data_layer.cpp，类定义文件为 data_layers.hpp，对应类名为 ImageDataLayer，ImageDataLayer 为 BasePrefetchingDataLayer，BaseDataLayer，Layer 的继承类。该层的主要功能是首先通过网络定义调整数据尺寸，数据处理（可采用批量处理的方式），将当前数据当作该层的输出。

主要函数

①DataLayerSetUp(const vector<Blob<Dtype>*>& bottom,vector<Blob <Dtype>

> top)

该函数的主要功能是通过网络定义调整数据尺寸，处理成（批处理大小、通道数、剪切尺寸、剪切尺寸），初始化数据层的输入和输出。

```
template <typename Dtype>
void ImageDataLayer<Dtype>::DataLayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top) {
  //获得设置的图片新的高度和宽度
  const int new_height = this->layer_param_.image_data_param().new_height();
  const int new_width  = this->layer_param_.image_data_param().new_width();
  const bool is_color  = this->layer_param_.image_data_param().is_color();
  string root_folder = this->layer_param_.image_data_param().root_folder();

  CHECK((new_height == 0 && new_width == 0) ||
      (new_height > 0 && new_width > 0)) << "Current implementation requires "
      "new_height and new_width to be set at the same time.";
  // 将文件中的图片名称和标签存入 lines_中
  const string& source = this->layer_param_.image_data_param().source();
  LOG(INFO) << "Opening file " << source;
  std::ifstream infile(source.c_str());
  string filename;
  int label;
  while (infile >> filename >> label) {
    lines_.push_back(std::make_pair(filename, label));
  }
  //判断是否将数据打乱顺序
  if (this->layer_param_.image_data_param().shuffle()) {
    // 随机将数据打乱顺序
    LOG(INFO) << "Shuffling data";
    const unsigned int prefetch_rng_seed = caffe_rng_rand();
    prefetch_rng_.reset(new Caffe::RNG(prefetch_rng_seed));
    ShuffleImages();//调用将数据打乱顺序的函数
  }
  LOG(INFO) << "A total of " << lines_.size() << " images.";

  lines_id_ = 0;
  // 检查是否需要跳过一些数据
  if (this->layer_param_.image_data_param().rand_skip()) {
```

```
25.        unsigned int skip = caffe_rng_rand() %
               this->layer_param_.image_data_param().rand_skip();
26.        LOG(INFO) << "Skipping first " << skip << " data points.";
27.        CHECK_GT(lines_.size(), skip) << "Not enough points to skip";
28.        lines_id_ = skip;
29.      }
         // 读取一个图片，并用它来初始化 top
30.      cv::Mat cv_img = ReadImageToCVMat(root_folder + lines_[lines_id_].first,
                                                  new_height, new_width, is_color);
31.      const int channels = cv_img.channels();
32.      const int height = cv_img.rows;
33.      const int width = cv_img.cols;
         // 图片处理
34.      const int crop_size = this->layer_param_.transform_param().crop_size();//图片剪切
尺寸
35.      const int batch_size = this->layer_param_.image_data_param().batch_size();//图片批
处理大小
36.      if (crop_size > 0) {
37.        top[0]->Reshape(batch_size, channels, crop_size, crop_size);
38.        this->prefetch_data_.Reshape(batch_size, channels, crop_size, crop_size);
39.        this->transformed_data_.Reshape(1, channels, crop_size, crop_size);
40.      } else {
41.        top[0]->Reshape(batch_size, channels, height, width);
42.        this->prefetch_data_.Reshape(batch_size, channels, height, width);
43.        this->transformed_data_.Reshape(1, channels, height, width);
44.      }
45.      LOG(INFO) << "output data size: " << top[0]->num() << ","
46.          << top[0]->channels() << "," << top[0]->height() << ","
47.          << top[0]->width();
         // 标签处理
48.      vector<int> label_shape(1, batch_size);
49.      top[1]->Reshape(label_shape);
50.      this->prefetch_label_.Reshape(label_shape);
51.    }
```

②ShuffleImages()

该函数的主要功能是将数据随机打乱顺序。

```
1.   template <typename Dtype>
2.   void ImageDataLayer<Dtype>::ShuffleImages() {
```

```
    caffe::rng_t* prefetch_rng =
        static_cast<caffe::rng_t*>(prefetch_rng_->generator());
    shuffle(lines_.begin(), lines_.end(), prefetch_rng);//随机打乱数据顺序
}
```

③InternalThreadEntry()

该函数的主要功能是创建一个批量读取数据并将数据进行一些简单的转换操作（如映射、缩放、剪切等）的线程，读取数据过程中会计算批处理所需时间，读数据所需时间，转换数据所需时间。

```
template <typename Dtype>
void ImageDataLayer<Dtype>::InternalThreadEntry() {
  CPUTimer batch_timer;
  batch_timer.Start();
  double read_time = 0;
  double trans_time = 0;
  CPUTimer timer;
  CHECK(this->prefetch_data_.count());
  CHECK(this->transformed_data_.count());
  ImageDataParameter image_data_param = this->layer_param_.image_data_param();
  const int batch_size = image_data_param.batch_size();//批处理大小
  const int new_height = image_data_param.new_height();//图片新的高度
  const int new_width = image_data_param.new_width();//图片新的宽度
  const int crop_size = this->layer_param_.transform_param().crop_size();//图片剪切后的尺寸
  const bool is_color = image_data_param.is_color();//图片是否为 RGB 图片
  string root_folder = image_data_param.root_folder();//图片文件根目录

  // 调整 batch_size=1 和 crop_size=0 时的图片尺寸
  if (batch_size == 1 && crop_size == 0 && new_height == 0 && new_width == 0) {
    cv::Mat cv_img = ReadImageToCVMat(root_folder + lines_[lines_id_].first,
        0, 0, is_color);
    this->prefetch_data_.Reshape(1, cv_img.channels(),
        cv_img.rows, cv_img.cols);
    this->transformed_data_.Reshape(1, cv_img.channels(),
        cv_img.rows, cv_img.cols);
  }

  Dtype* prefetch_data = this->prefetch_data_.mutable_cpu_data();
  Dtype* prefetch_label = this->prefetch_label_.mutable_cpu_data();
```

```
    // 数据转换（如映射、缩放、剪切等）
24.   const int lines_size = lines_.size();
25.   for (int item_id = 0; item_id < batch_size; ++item_id) {
        // 获得一个数据块
26.     timer.Start();
27.     CHECK_GT(lines_size, lines_id_);
28.     cv::Mat cv_img = ReadImageToCVMat(root_folder + lines_[lines_id_].first,
            new_height, new_width, is_color);
29.     CHECK(cv_img.data) << "Could not load " << lines_[lines_id_].first;
30.     read_time += timer.MicroSeconds();
31.     timer.Start();
        // 数据转换（映射、缩放、剪切等）
32.     int offset = this->prefetch_data_.offset(item_id);
33.     this->transformed_data_.set_cpu_data(prefetch_data + offset);
34.     this->data_transformer_->Transform(cv_img, &(this->transformed_data_));
35.     trans_time += timer.MicroSeconds();

36.     prefetch_label[item_id] = lines_[lines_id_].second;
        // 使用 lines_id_ 访问下一个数据块
37.     lines_id_++;
38.     if (lines_id_ >= lines_size) {
          // 访问到文件结尾，重新从文件第一行开始访问数据
39.       DLOG(INFO) << "Restarting data prefetching from start.";
40.       lines_id_ = 0;
41.       if (this->layer_param_.image_data_param().shuffle()) {
42.         ShuffleImages();
43.       }
44.     }
45.   }
46.   batch_timer.Stop();
47.   DLOG(INFO) << "Prefetch batch: " << batch_timer.MilliSeconds() << " ms.";
48.   DLOG(INFO) << "Read time: " << read_time / 1000 << " ms.";
49.   DLOG(INFO) << "Transform time: " << trans_time / 1000 << " ms.";
50. }
```

5.2.2 网络层解析

网络层是深度网络结构中重要的组成部分，本节将详细介绍卷积层、下采样层和全连接层。

1. 卷积层

卷积层是卷积神经网络中较为核心的网络层，主要进行卷积操作，基于图像的空间局部相关性分别抽取图像局部特征，通过将这些局部特征进行连接，可以形成整体特征。

（1）主要参数

num_output：表示该层的输出通道数。

kernel_size（或者 kernel_h 和 kernel_w）：表示在输入上使用的过滤器尺寸。

stride（或者 stride_h 和 stride_w）：表示在输入上使用过滤器（filters）进行卷积时的间隔，默认值为 1。

pad（或者 pad_h 和 pad_w）：表示在输入的边界上添加的像素数，默认值为 0。

（2）输入和输出

输入：$n \times c\_i \times h\_i \times w\_i$，$n$ 为输入图片数，c_i 为输入通道数，h_i 和 w_i 分别为输入图片高度、宽度。

输出：$n \times c\_o \times h\_o \times w\_o$，$n$ 为输出图片数，c_o 为输出通道数，h_o 和 w_o 分别为输出图片高度、宽度。其中 c_o 为 num_output 的数值，$h\_o=(h\_i+2 \times pad\_h-kernel\_h)/stride\_h+1$，$w\_o=(w\_i+2 \times pad\_w-kernel\_w)/stride\_w+1$。

（3）层定义

```
layer {
  name: "conv1" #层属性：层名称
  type: "Convolution" #层属性：层类型
  bottom: "data" #层属性：下一层名称
  top: "conv1" #层属性：上一层名称
  param { #层参数：权重相关参数
    lr_mult: 1
    decay_mult: 1
  }
  param { #层参数：偏置相关参数
    lr_mult: 2
```

```
    decay_mult: 0
  }
  convolution_param { #层参数
    num_output: 96 #层参数：输出通道数
    kernel_size: 11 #层参数：卷积核尺寸，11×11
    stride: 4 #层参数：卷积步长
    weight_filler { #层参数：利用标准差为 0.01 的高斯分布初始化权重
      type: "gaussian"
      std: 0.01
    }
    bias_filler { #层参数：初始化偏置为常量 0
      type: "constant"
      value: 0
    }
  }
}
```

该层主要进行卷积计算，经过该层计算后，输出为 96 个通道数，每个通道中数据尺寸变为：[(227-11)/4+1]×[(227-11)/4+1] =55×55。

（4）源码解析

整体简介

该层 CPU 计算模式下对应的源代码文件为 conv_layer.cpp，GPU 计算模式下对应的源代码文件为 conv_layer.cu。类定义文件为 vision_layers.hpp，对应类名为 ConvolutionLayer，ConvolutionLayer 为 BaseConvolutionLayer，Layer 的继承类。

主要函数

①LayerSetUp(const vector<Blob<Dtype>*>& bottom,vector<Blob<Dtype>*> * top)

该函数的主要功能是通过网络结构定义（当前卷积层的尺寸、填充尺寸、卷积核尺寸、步长尺寸），计算当前卷积层的输出特征图的尺寸。

```
1.  template <typename Dtype>
2.  void ConvolutionLayer<Dtype>::compute_output_shape() {
3.    this->height_out_ = (this->height_ + 2 × this->pad_h_ - this->kernel_h_)
        / this->stride_h_ + 1;// 计算当前卷积层的输出特征图的高度
4.    this->width_out_ = (this->width_ + 2 × this->pad_w_ - this->kernel_w_)
        / this->stride_w_ + 1;// 计算当前卷积层的输出特征图的宽度
5.  }
```

② Forward_cpu(const vector<Blob<Dtype>*>& bottom,const vector<Blob

<Dtype>*>& top)

该函数的主要功能是在 CPU 模式下从输入层到输出层的前向计算，遍历所有图片，第 8 行代码实现对每张图片数据和对应的权重信息利用 forward_cpu_gemm 函数进行矩阵相乘操作，利用第 9～10 行代码将得到的乘积加上对应的偏置信息，最后的结果就是该层的输出结果。

```
1.  template <typename Dtype>
2.  void ConvolutionLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
3.          const vector<Blob<Dtype>*>& top) {
      //获得权重信息
4.    const Dtype* weight = this->blobs_[0]->cpu_data();
5.    for (int i = 0; i < bottom.size(); ++i) {
6.      const Dtype* bottom_data = bottom[i]->cpu_data();
7.      Dtype* top_data = top[i]->mutable_cpu_data();
        for (int n = 0; n < this->num_; ++n) { // 遍历所有图片，计算每张图片经过卷积之后的结果
          //矩阵相乘操作
8.        this->forward_cpu_gemm(bottom_data + bottom[i]->offset(n), weight,
              top_data + top[i]->offset(n));
          //如果有偏置项，则添加偏置项
9.        if (this->bias_term_) {
10.         const Dtype* bias = this->blobs_[1]->cpu_data();
11.         this->forward_cpu_bias(top_data + top[i]->offset(n), bias);
12.       }
13.     }
14.   }
15. }
```

③ Backward_cpu(const vector<Blob<Dtype>*>& top,const vector<bool>& propagate_down, vector<Blob<Dtype>*>* bottom)

该函数的主要功能是在 CPU 模式下从输出层到输入层的后向计算，遍历所有数据，第 17～22 行代码根据顶层数据梯度（top_diff）直接计算出偏置梯度（bias_diff）；第 25～27 行代码根据底层数据（bottom_data）和顶层数据梯度（top_diff）作矩阵相乘计算出权重梯度（weight_diff）；第 28～30 行代码根据顶层数据梯度（top_diff）和对应权重（weight）作矩阵相乘计算出底层数据梯度（bottom_diff）。

```
1.  template <typename Dtype>
2.  void ConvolutionLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
```

```
3.          const vector<bool>& propagate_down, const vector<Blob<Dtype>*>& bottom) {
4.      const Dtype* weight = this->blobs_[0]->cpu_data();// 获得权重信息
5.      Dtype* weight_diff = this->blobs_[0]->mutable_cpu_diff(); // 保存权重梯度
6.      if (this->param_propagate_down_[0]) {
        // 初始化权重梯度变量 weight_diff 为 0
7.        caffe_set(this->blobs_[0]->count(), Dtype(0), weight_diff);
8.      }
9.      if (this->bias_term_ && this->param_propagate_down_[1]) {
        // 初始化偏置梯度为 0
10.       caffe_set(this->blobs_[1]->count(), Dtype(0),
11.           this->blobs_[1]->mutable_cpu_diff());
12.     }
13.     for (int i = 0; i < top.size(); ++i) {
14.       const Dtype* top_diff = top[i]->cpu_diff();
15.       const Dtype* bottom_data = bottom[i]->cpu_data();
16.       Dtype* bottom_diff = bottom[i]->mutable_cpu_diff();
        // 如果需要，计算偏置梯度
17.       if (this->bias_term_ && this->param_propagate_down_[1]) {
18.         Dtype* bias_diff = this->blobs_[1]->mutable_cpu_diff();
19.         for (int n = 0; n < this->num_; ++n) {
20.           this->backward_cpu_bias(bias_diff, top_diff + top[i]->offset(n));
21.         }
22.       }
23.       if (this->param_propagate_down_[0] || propagate_down[i]) {
24.         for (int n = 0; n < this->num_; ++n) {// 遍历所有数据
            // 计算权重梯度
25.           if (this->param_propagate_down_[0]) {
26.             this->weight_cpu_gemm(bottom_data + bottom[i]->offset(n),
                  top_diff + top[i]->offset(n), weight_diff);
27.           }
            // 计算底层数据梯度
28.           if (propagate_down[i]) {
29.             this->backward_cpu_gemm(top_diff + top[i]->offset(n), weight,
                  bottom_diff + bottom[i]->offset(n));
30.           }
31.         }
32.       }
33.     }
34.   }
```

2. 下采样层

下采样层主要是对卷积层的卷积结果进行下采样操作，可以起到降低维度的作用。

（1）主要参数

kernel_size（或者 kernel_h 和 kernel_w）：表示在输入上使用的过滤器尺寸。

stride（或者 stride_h 和 stride_w）：表示在输入上使用过滤器（filters）进行卷积时的间隔，默认值为 1。

pool：表示采样方法，目前有最大值池化、平均值池化和随机池化三种方法。

（2）输入和输出

输入：$n \times c\_i \times h\_i \times w\_i$，$n$ 为输入图片数，c_i 为输入通道数，h_i 和 w_i 分别为输入图片高度、宽度。

输出：$n \times c\_o \times h\_o \times w\_o$，$n$ 为输出图片数，c_o 为输出通道数，h_o 和 w_o 分别为输出图片高度、宽度。其中 c_o 为 num_output 的数值，$h\_o=(h\_i+2 \times pad\_h-kernel\_h)/stride\_h+1$，$w\_o=(w\_i+2 \times pad\_w-kernel\_w)/stride\_w+1$。

（3）层定义

```
layer {
  name: "pool1" #层属性：层名称
  type: "Pooling" #层属性：层类型
  bottom: "conv1" #层属性：下一层名称
  top: "pool1" #层属性：上一层名称
  pooling_param { #层参数
    pool: MAX #层参数：池化类型，取最大值
    kernel_size: 3 #层参数：池化核尺寸，3×3
    stride: 2 #层参数：池化步长
  }
}
```

该层主要进行下采样计算，在上一个卷积层的基础上经过该层计算后，数据尺寸变为：[(55-3)/2+1]×[(55-3)/2+1] =27×27。

（4）源码解析

整体简介

该层 CPU 计算模式下对应的源代码文件为 pooling_layer.cpp，GPU 计算模式下对应的源代码文件为 pooling_layer.cu，类定义文件为 vision_layers.hpp，对应类名为 PoolingLayer，PoolingLayer 为 Layer 的继承类。

主要函数

①LayerSetUp(const vector<Blob<Dtype>*>& bottom,vector<Blob<Dtype>*>* top)

该函数的主要功能是初始化采样层过滤器尺寸，填充尺寸、步长和采样策略。

```
template <typename Dtype>
void PoolingLayer<Dtype>::LayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top) {
  // 配置过滤器尺寸，填充尺寸、步长
  PoolingParameter pool_param = this->layer_param_.pooling_param();
  if (pool_param.global_pooling()) {
    CHECK(!(pool_param.has_kernel_size() ||
      pool_param.has_kernel_h() || pool_param.has_kernel_w()))
      << "With Global_pooling: true Filter size cannot specified";
  } else {
    CHECK(!pool_param.has_kernel_size() !=
      !(pool_param.has_kernel_h() && pool_param.has_kernel_w()))
      << "Filter size is kernel_size OR kernel_h and kernel_w; not both";
    CHECK(pool_param.has_kernel_size() ||
      (pool_param.has_kernel_h() && pool_param.has_kernel_w()))
      << "For non-square filters both kernel_h and kernel_w are required.";
  }
  CHECK((!pool_param.has_pad() && pool_param.has_pad_h()
      && pool_param.has_pad_w())
      || (!pool_param.has_pad_h() && !pool_param.has_pad_w()))
      << "pad is pad OR pad_h and pad_w are required.";
  CHECK((!pool_param.has_stride() && pool_param.has_stride_h()
      && pool_param.has_stride_w())
      || (!pool_param.has_stride_h() && !pool_param.has_stride_w()))
      << "Stride is stride OR stride_h and stride_w are required.";
  global_pooling_ = pool_param.global_pooling();
  if (global_pooling_) {// 配置过滤器尺寸
    kernel_h_ = bottom[0]->height();
    kernel_w_ = bottom[0]->width();
  } else {
    if (pool_param.has_kernel_size()) {
      kernel_h_ = kernel_w_ = pool_param.kernel_size();
    } else {
      kernel_h_ = pool_param.kernel_h();
```

```
22.          kernel_w_ = pool_param.kernel_w();
23.        }
24.      }
25.      CHECK_GT(kernel_h_, 0) << "Filter dimensions cannot be zero.";
26.      CHECK_GT(kernel_w_, 0) << "Filter dimensions cannot be zero.";
27.      if (!pool_param.has_pad_h()) {// 填充尺寸
28.        pad_h_ = pad_w_ = pool_param.pad();
29.      } else {
30.        pad_h_ = pool_param.pad_h();
31.        pad_w_ = pool_param.pad_w();
32.      }
33.      if (!pool_param.has_stride_h()) {// 配置步长
34.        stride_h_ = stride_w_ = pool_param.stride();
35.      } else {
36.        stride_h_ = pool_param.stride_h();
37.        stride_w_ = pool_param.stride_w();
38.      }
39.      if (global_pooling_) {
40.        CHECK(pad_h_ == 0 && pad_w_ == 0 && stride_h_ == 1 && stride_w_ == 1)
             << "With Global_pooling: true; only pad = 0 and stride = 1";
41.      }
42.      if (pad_h_ != 0 || pad_w_ != 0) {
43.        CHECK(this->layer_param_.pooling_param().pool()
                 == PoolingParameter_PoolMethod_AVE
                 || this->layer_param_.pooling_param().pool()
                 == PoolingParameter_PoolMethod_MAX)
                 << "Padding implemented only for average and max pooling.";
44.        CHECK_LT(pad_h_, kernel_h_);
45.        CHECK_LT(pad_w_, kernel_w_);
46.      }
47.    }
```

② Reshape(const vector<Blob<Dtype>*>& bottom,vector<Blob<Dtype>*>* top)

该函数的主要功能是设置采样后尺寸，控制采样区域和对不同采样策略的处理操作。

```
1.  template <typename Dtype>
2.  void PoolingLayer<Dtype>::Reshape(const vector<Blob<Dtype>*>& bottom,
        const vector<Blob<Dtype>*>& top) {
```

```
3.    CHECK_EQ(4, bottom[0]->num_axes()) << "Input must have 4 axes, "
          << "corresponding to (num, channels, height, width)";
4.    channels_ = bottom[0]->channels();//底层通道数
5.    height_ = bottom[0]->height();//底层数据高度
6.    width_ = bottom[0]->width();//底层数据宽度
7.    if (global_pooling_) {
8.      kernel_h_ = bottom[0]->height();
9.      kernel_w_ = bottom[0]->width();
10.   }
11.   pooled_height_ = static_cast<int>(ceil(static_cast<float>(
          height_ + 2 * pad_h_ - kernel_h_) / stride_h_)) + 1;//采样后数据高度
12.   pooled_width_ = static_cast<int>(ceil(static_cast<float>(
          width_ + 2 * pad_w_ - kernel_w_) / stride_w_)) + 1;//采样后数据宽度
13.   if (pad_h_ || pad_w_) {
        //如果使用填充，需保证采样时，采样区域的起始点在原始图像内部
14.     if ((pooled_height_ - 1) * stride_h_ >= height_ + pad_h_) {
15.       --pooled_height_;
16.     }
17.     if ((pooled_width_ - 1) * stride_w_ >= width_ + pad_w_) {
18.       --pooled_width_;
19.     }
20.     CHECK_LT((pooled_height_ - 1) * stride_h_, height_ + pad_h_);
21.     CHECK_LT((pooled_width_ - 1) * stride_w_, width_ + pad_w_);
22.   }
23.   top[0]->Reshape(bottom[0]->num(), channels_, pooled_height_,
          pooled_width_);
24.   if (top.size() > 1) {
25.     top[1]->ReshapeLike(*top[0]);
26.   }
      // 最大化采样处理
27.   if (this->layer_param_.pooling_param().pool() ==
          PoolingParameter_PoolMethod_MAX && top.size() == 1) {
28.     max_idx_.Reshape(bottom[0]->num(), channels_, pooled_height_,
            pooled_width_);
29.   }
      // 随机采样处理
30.   if (this->layer_param_.pooling_param().pool() ==
          PoolingParameter_PoolMethod_STOCHASTIC) {
31.     rand_idx_.Reshape(bottom[0]->num(), channels_, pooled_height_,
```

```
            pooled_width_);
    32.     }
    33.   }
```

③ Forward_cpu(const vector<Blob<Dtype>*>& bottom,vector<Blob<Dtype>*>* top)

该函数的主要功能是在 CPU 模式下从输入层到输出层的前向计算，按不同的采样策略分别处理，第 19～54 行代码表示最大值采样策略中使用采样窗口采样整个特征图并保存每个采样窗口中的最大值；第 60～86 行代码表示平均值采样策略中使用采样窗口采样整个特征图并保存每个采样窗口中的平均值。

```
    1.  template <typename Dtype>
    2.  void PoolingLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
            const vector<Blob<Dtype>*>& top) {
    3.    const Dtype* bottom_data = bottom[0]->cpu_data();
    4.    Dtype* top_data = top[0]->mutable_cpu_data();
    5.    const int top_count = top[0]->count();
          // 如果上一层的大小大于 1，则 top[1]用来保存 mask 信息
    6.    const bool use_top_mask = top.size() > 1;
    7.    int* mask = NULL;  // 使用 NULL 初始化变量
    8.    Dtype* top_mask = NULL;
          // 处理不同的采样策略
    9.    switch (this->layer_param_.pooling_param().pool()) {
    10.   case PoolingParameter_PoolMethod_MAX:
            // 初始化操作
    11.     if (use_top_mask) {
    12.       top_mask = top[1]->mutable_cpu_data();
    13.       caffe_set(top_count, Dtype(-1), top_mask);
    14.     } else {
    15.       mask = max_idx_.mutable_cpu_data();
    16.       caffe_set(top_count, -1, mask);
    17.     }
    18.     caffe_set(top_count, Dtype(-FLT_MAX), top_data);
            // 主循环：使用采样窗口采样整个特征图并保存每个采样窗口中的最大值
    19.     for (int n = 0; n < bottom[0]->num(); ++n) {
    20.       for (int c = 0; c < channels_; ++c) {
    21.         for (int ph = 0; ph < pooled_height_; ++ph) {
    22.           for (int pw = 0; pw < pooled_width_; ++pw) {
    23.             //hstart ， wstart ， hend ， wend 为采样窗口在特征图中对应起始点
坐标
```

```
24.                 int hstart = ph * stride_h_ - pad_h_;
25.                 int wstart = pw * stride_w_ - pad_w_;
26.                 int hend = min(hstart + kernel_h_, height_);
27.                 int wend = min(wstart + kernel_w_, width_);
28.                 hstart = max(hstart, 0);
29.                 wstart = max(wstart, 0);
30.                 const int pool_index = ph * pooled_width_ + pw;
31.                 for (int h = hstart; h < hend; ++h) {
32.                   for (int w = wstart; w < wend; ++w) {
33.                     const int index = h * width_ + w;
34.                     if (bottom_data[index] > top_data[pool_index]) {
35.                       top_data[pool_index] = bottom_data[index];
36.                       if (use_top_mask) {
37.                         top_mask[pool_index] = static_cast<Dtype>(index);
38.                       } else {
39.                         mask[pool_index] = index;
40.                       }
41.                     }
42.                   }
43.                 }
44.               }
45.             }
                // 计算偏移量
46.             bottom_data += bottom[0]->offset(0, 1);
47.             top_data += top[0]->offset(0, 1);
48.             if (use_top_mask) {
49.               top_mask += top[0]->offset(0, 1);
50.             } else {
51.               mask += top[0]->offset(0, 1);
52.             }
53.           }
54.         }
55.         break;
56.       case PoolingParameter_PoolMethod_AVE:
57.         for (int i = 0; i < top_count; ++i) {
58.           top_data[i] = 0;
59.         }
            // 主循环：使用采样窗口采样整个特征图并保存每个采样窗口中的平均值
60.         for (int n = 0; n < bottom[0]->num(); ++n) {
```

```
61.          for (int c = 0; c < channels_; ++c) {
62.            for (int ph = 0; ph < pooled_height_; ++ph) {
63.              for (int pw = 0; pw < pooled_width_; ++pw) {
64.                //hstart , wstart , hend , wend 为采样窗口在特征图中对应起始点
坐标信息
65.                int hstart = ph * stride_h_ - pad_h_;
66.                int wstart = pw * stride_w_ - pad_w_;
67.                int hend = min(hstart + kernel_h_, height_ + pad_h_);
68.                int wend = min(wstart + kernel_w_, width_ + pad_w_);
69.                int pool_size = (hend - hstart) * (wend - wstart);
70.                hstart = max(hstart, 0);
71.                wstart = max(wstart, 0);
72.                hend = min(hend, height_);
73.                wend = min(wend, width_);
74.                for (int h = hstart; h < hend; ++h) {
75.                  for (int w = wstart; w < wend; ++w) {
76.                    top_data[ph * pooled_width_ + pw] +=
77.                        bottom_data[h * width_ + w];
78.                  }
79.                }
80.                top_data[ph * pooled_width_ + pw] /= pool_size;
81.              }
82.            }
               // 计算偏移量
83.            bottom_data += bottom[0]->offset(0, 1);
84.            top_data += top[0]->offset(0, 1);
85.          }
86.        }
87.        break;
88.      case PoolingParameter_PoolMethod_STOCHASTIC:
89.        NOT_IMPLEMENTED;
90.        break;
91.      default:
92.        LOG(FATAL) << "Unknown pooling method.";
93.      }
94.    }
```

④ Backward_cpu(const vector<Blob<Dtype>*>& top,const vector<bool>& propagate_down, vector<Blob<Dtype>*>* bottom)

该函数的主要功能是在 CPU 模式下从输出层到输入层的后向计算，按不同的采样策略分别处理，第 14～36 行代码表示最大值采样策略中根据顶层数据梯度（top_diff）直接计算出底层数据对应的采样窗口中最大值的数据梯度（bottom_diff）；第 39～63 行代码表示平均值采样策略中根据顶层数据梯度（top_diff）直接计算出底层数据对应的采样窗口中每一个数值的数据梯度（bottom_diff）为顶层数据梯度（top_diff）与采样窗口大小的比值。

```
1.  template <typename Dtype>
2.  void PoolingLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
        const vector<bool>& propagate_down, const vector<Blob<Dtype>*>& bottom) {
3.    if (!propagate_down[0]) {
4.      return;
5.    }
6.    const Dtype* top_diff = top[0]->cpu_diff();//获得 top 层的残差
7.    Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
      // 处理不同的采样策略
8.    caffe_set(bottom[0]->count(), Dtype(0), bottom_diff);
      // 如果上一层的大小大于 1，则 top[1]用来保存 mask 信息
9.    const bool use_top_mask = top.size() > 1;
10.   const int* mask = NULL;  // suppress warnings about uninitialized variables
11.   const Dtype* top_mask = NULL;
12.   switch (this->layer_param_.pooling_param().pool()) {
13.   case PoolingParameter_PoolMethod_MAX:
        // 主循环
14.     if (use_top_mask) {
15.       top_mask = top[1]->cpu_data();
16.     } else {
17.       mask = max_idx_.cpu_data();
18.     }
19.     for (int n = 0; n < top[0]->num(); ++n) {
20.       for (int c = 0; c < channels_; ++c) {
21.         for (int ph = 0; ph < pooled_height_; ++ph) {
22.           for (int pw = 0; pw < pooled_width_; ++pw) {
23.             const int index = ph * pooled_width_ + pw;
24.             const int bottom_index =
                    use_top_mask ? top_mask[index] : mask[index];
25.             bottom_diff[bottom_index] += top_diff[index];//根据 top_diff 获得对应
的 bottom_diff
26.           }
```

```
27.             }
                // 计算偏移量
28.             bottom_diff += bottom[0]->offset(0, 1);
29.             top_diff += top[0]->offset(0, 1);
30.             if (use_top_mask) {
31.               top_mask += top[0]->offset(0, 1);
32.             } else {
33.               mask += top[0]->offset(0, 1);
34.             }
35.           }
36.         }
37.         break;
38.       case PoolingParameter_PoolMethod_AVE:
            // 主循环
39.         for (int n = 0; n < top[0]->num(); ++n) {
40.           for (int c = 0; c < channels_; ++c) {
41.             for (int ph = 0; ph < pooled_height_; ++ph) {
42.               for (int pw = 0; pw < pooled_width_; ++pw) {
43.                 //hstart , wstart , hend , wend 为采样窗口在特征图中对应起始点
坐标信息
44.                 int hstart = ph * stride_h_ - pad_h_;
45.                 int wstart = pw * stride_w_ - pad_w_;
46.                 int hend = min(hstart + kernel_h_, height_ + pad_h_);
47.                 int wend = min(wstart + kernel_w_, width_ + pad_w_);
48.                 int pool_size = (hend - hstart) * (wend - wstart);
49.                 hstart = max(hstart, 0);
50.                 wstart = max(wstart, 0);
51.                 hend = min(hend, height_);
52.                 wend = min(wend, width_);
53.                 for (int h = hstart; h < hend; ++h) {
54.                   for (int w = wstart; w < wend; ++w) {
55.                     bottom_diff[h * width_ + w] +=
                          top_diff[ph * pooled_width_ + pw] / pool_size; //根据 top_diff 和
采样窗口大小获得对应的 bottom_diff
56.                   }
57.                 }
58.               }
59.             }
                // 计算偏移量
```

```
60.                 bottom_diff += bottom[0]->offset(0, 1);
61.                 top_diff += top[0]->offset(0, 1);
62.               }
63.             }
64.             break;
65.           case PoolingParameter_PoolMethod_STOCHASTIC:
66.             NOT_IMPLEMENTED;
67.             break;
68.           default:
69.             LOG(FATAL) << "Unknown pooling method.";
70.         }
71.     }
```

3. 全连接层

全连接层主要将多维特征向量连接成单一特征向量，该层中每个神经元与前一层中所有神经元之间都有连接。

（1）主要参数

num_output：表示该层的输出通道数。

（2）输入和输出

输入：$n \times c\_i \times h\_i \times w\_i$，$n$ 为输入图片数，c_i 为输入通道数，h_i 和 w_i 分别为输入图片高度、宽度。

输出：$n \times c\_o \times 1 \times 1$，$n$ 为输出图片数，c_o 为输出通道数。

（3）层定义

```
layer {
  name: "fc6" #层属性：层名称
  type: "InnerProduct" #层属性：层类型
  bottom: "pool5" #层属性：下一层名称
  top: "fc6" #层属性：上一层名称
  param { #层参数：权重相关参数
    lr_mult: 1
    decay_mult: 1
  }
  param { #层参数：偏置相关参数
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param { #层参数
```

```
    num_output: 4096 #层参数：输出通道数
    weight_filler { #层参数：利用标准差为 0.005 的高斯分布初始化权重
      type: "gaussian"
      std: 0.005
    }
    bias_filler { #层参数：初始化偏置为常量 1
      type: "constant"
      value: 1
    }
  }
}
```

该层主要进行全连接计算，将输入当作简单向量进行处理，经过该层计算后，输出单个向量，数据尺寸变为：1×1。

（4）源码解析

整体简介

该层 CPU 计算模式下对应的源代码文件为 inner_product_layer.cpp，GPU 计算模式下对应的源代码文件为 inner_product_layer.cu，类定义文件为 common_layers.hpp，对应类名为 InnerProductLayer，InnerProductLayer 为 Layer 的继承类。

主要函数

①LayerSetUp(const vector<Blob<Dtype>*>& bottom,vector<Blob<Dtype>*>* top)

该函数的主要功能是初始化卷积层过滤器尺寸、填充尺寸、步长和输入及初始化权重和偏置值。

```
1.  template <typename Dtype>
2.  void InnerProductLayer<Dtype>::LayerSetUp(const vector<Blob<Dtype>*>& bottom,
3.          const vector<Blob<Dtype>*>& top) {
      // 得到输出单元数目
4.    const int num_output = this->layer_param_.inner_product_param().num_output();
5.    bias_term_ = this->layer_param_.inner_product_param().bias_term();
6.    N_ = num_output;
7.    const int axis = bottom[0]->CanonicalAxisIndex(
8.          this->layer_param_.inner_product_param().axis());
      // 将 bottom 层展开为长度为 K_的向量。例如，如果 bottom[0]的尺寸是 (N, C, H, W),
      // 并且 axis == 1, 最终得到 N 个长度为 CHW 的向量
```

```
9.      K_ = bottom[0]->count(axis);
        // 检查是否需要设置权重参数
10.     if (this->blobs_.size() > 0) {
11.       LOG(INFO) << "Skipping parameter initialization";
12.     } else {
13.       if (bias_term_) {
14.         this->blobs_.resize(2);
15.       } else {
16.         this->blobs_.resize(1);
17.       }
          // 初始化权重值
18.       vector<int> weight_shape(2);
19.       weight_shape[0] = N_;
20.       weight_shape[1] = K_;
21.       this->blobs_[0].reset(new Blob<Dtype>(weight_shape));
          // 使用权重过滤器填充权重变量
22.       shared_ptr<Filler<Dtype> > weight_filler(GetFiller<Dtype>(
              this->layer_param_.inner_product_param().weight_filler()));
23.       weight_filler->Fill(this->blobs_[0].get());
          // 如果需要，初始化并填充偏置变量
24.       if (bias_term_) {
25.         vector<int> bias_shape(1, N_);
26.         this->blobs_[1].reset(new Blob<Dtype>(bias_shape));
27.         shared_ptr<Filler<Dtype> > bias_filler(GetFiller<Dtype>(
                this->layer_param_.inner_product_param().bias_filler()));
28.         bias_filler->Fill(this->blobs_[1].get());
29.       }
30.     }  // 参数初始化
31.     this->param_propagate_down_.resize(this->blobs_.size(), true);
32.   }
```

② Reshape(const vector<Blob<Dtype>*>& bottom,vector<Blob<Dtype>*>* top)

该函数的主要功能是计算输出尺寸大小。

```
1.    template <typename Dtype>
2.    void InnerProductLayer<Dtype>::Reshape(const vector<Blob<Dtype>*>& bottom,
            const vector<Blob<Dtype>*>& top) {
        // 计算输出尺寸
3.      const int axis = bottom[0]->CanonicalAxisIndex(
            this->layer_param_.inner_product_param().axis());
```

```
4.      const int new_K = bottom[0]->count(axis);
5.      CHECK_EQ(K_, new_K)
            << "Input size incompatible with inner product parameters.";
        // M_表示所有数据数目
6.      M_ = bottom[0]->count(0, axis);
        // top 层的尺寸为 num_output (N_)
7.      vector<int> top_shape = bottom[0]->shape();
8.      top_shape.resize(axis + 1);
9.      top_shape[axis] = N_;
10.     top[0]->Reshape(top_shape);
        // 设置偏置乘子
11.     if (bias_term_) {
12.       vector<int> bias_shape(1, M_);
13.       bias_multiplier_.Reshape(bias_shape);
14.       caffe_set(M_, Dtype(1), bias_multiplier_.mutable_cpu_data());
15.     }
16.   }
```

③Forward_cpu(const vector<Blob<Dtype>*>& bottom,vector<Blob<Dtype>*> * top)

该函数的主要功能是在 CPU 模式下从输入层到输出层的前向计算，利用底层数据（bottom_data），相关权重（weight）进行线性加权和得到求和结果，如果有偏置项，则需要在求和结果上加上对应的偏置值，最后计算的结果即为顶层数据（top_data）。

```
1.    template <typename Dtype>
2.    void InnerProductLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
          const vector<Blob<Dtype>*>& top) {
3.      const Dtype* bottom_data = bottom[0]->cpu_data();
4.      Dtype* top_data = top[0]->mutable_cpu_data();
5.      const Dtype* weight = this->blobs_[0]->cpu_data();
        // 利用 bottom_data, weight 线性加权和求解 top_data
6.      caffe_cpu_gemm<Dtype>(CblasNoTrans, CblasTrans, M_, N_, K_, (Dtype)1.,
            bottom_data, weight, (Dtype)0., top_data);
        // 如果有偏置项，需计算偏置
7.      if (bias_term_) {
8.        caffe_cpu_gemm<Dtype>(CblasNoTrans, CblasNoTrans, M_, N_, 1, (Dtype)1.,
              bias_multiplier_.cpu_data(),
              this->blobs_[1]->cpu_data(), (Dtype)1., top_data);
```

```
9.      }
10.  }
```

④ Backward_cpu(const vector<Blob<Dtype>*>& top,const vector<bool>& propagate_down, vector<Blob<Dtype>*>* bottom)

该函数的主要功能是在 CPU 模式下从输出层到输入层的后向计算，第 6 行代码表示根据底层数据（bottom_data）和顶层数据梯度（top_diff）作矩阵相乘计算出权重梯度（weight_diff）；第 10 行代码表示根据顶层数据梯度（top_diff）直接计算出偏置梯度（bias_diff）；第 14 行代码表示根据顶层数据梯度（top_diff）和对应权重信息作矩阵相乘计算出底层数据梯度（bottom_diff）。

```
1.   template <typename Dtype>
2.   void InnerProductLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
         const vector<bool>& propagate_down,
         const vector<Blob<Dtype>*>& bottom) {
3.     if (this->param_propagate_down_[0]) {
4.       const Dtype* top_diff = top[0]->cpu_diff();
5.       const Dtype* bottom_data = bottom[0]->cpu_data();
       // 利用顶层数据梯度 top_diff 和底层数据 bottom_data 计算权重梯度
6.       caffe_cpu_gemm<Dtype>(CblasTrans, CblasNoTrans, N_, K_, M_, (Dtype)1.,
             top_diff, bottom_data, (Dtype)0., this->blobs_[0]->mutable_cpu_diff());
7.     }
8.     if (bias_term_ && this->param_propagate_down_[1]) {
9.       const Dtype* top_diff = top[0]->cpu_diff();
       // 利用顶层数据梯度 top_diff 计算偏置梯度
10.      caffe_cpu_gemv<Dtype>(CblasTrans, M_, N_, (Dtype)1., top_diff,
             bias_multiplier_.cpu_data(), (Dtype)0.,
             this->blobs_[1]->mutable_cpu_diff());
11.    }
12.    if (propagate_down[0]) {
13.      const Dtype* top_diff = top[0]->cpu_diff();
     // 利用顶层数据梯度 top_diff 和权重信息计算每个单元的输出的梯度，
     // 其中 this->blobs_[0]->cpu_data()表示权重信息
14.      caffe_cpu_gemm<Dtype>(CblasNoTrans, CblasNoTrans, M_, K_, N_, (Dtype)1.,
             top_diff, this->blobs_[0]->cpu_data(), (Dtype)0.,
             bottom[0]->mutable_cpu_diff());
15.    }
16.  }
```

5.2.3 网络结构解析

网络结构以层为基本单位，是由数据输入层、各种类型网络层、全连接层等共同构成的。

1. 示例

网络结构以 LeNet 网络为例，如下所示：

```
name: "LeNet" // 网络名称
layer {  // 训练阶段数据层
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
layer {  // 测试阶段数据层
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    scale: 0.00390625
  }
```

```
  data_param {
    source: "examples/mnist/mnist_test_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {  // 卷积层，卷积核尺寸为 5×5，步长为 1
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {  // 采样层，采样窗口尺寸为 2×2，步长为 2
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
```

```
}
layer {  // 卷积层，卷积核尺寸为 5×5，步长为 1
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {  // 采样层，采样窗口尺寸为 2×2，步长为 2
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {  // 全连接层
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
```

```
    top: "ip1"
    param {
      lr_mult: 1
    }
    param {
      lr_mult: 2
    }
    inner_product_param {
      num_output: 500
      weight_filler {
        type: "xavier"
      }
      bias_filler {
        type: "constant"
      }
    }
  }
  layer {  // 神经元激活函数层
    name: "relu1"
    type: "ReLU"
    bottom: "ip1"
    top: "ip1"
  }
  layer {  // 全连接层
    name: "ip2"
    type: "InnerProduct"
    bottom: "ip1"
    top: "ip2"
    param {
      lr_mult: 1
    }
    param {
      lr_mult: 2
    }
    inner_product_param {
      num_output: 10
      weight_filler {
        type: "xavier"
      }
```

```
    bias_filler {
      type: "constant"
    }
  }
}
layer {  // 输出层，输出正确率
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {  // 网络损失层，输出网络损失
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

2. 源码解析

整体简介

该层对应的源代码文件为 net.cpp，类定义文件为 net.hpp，对应类名为 Net。该类包含函数较多，这里只列举以下其中的几个函数，有兴趣的读者，可以自己去查找对应的源码理解。

主要函数

①Init(const NetParameter& in_param)

该函数的主要功能是使用网络参数初始化整个网络结构，构建所有网络层并将它们连接起来，主要分前向设置和后向设置两部分，其中前向设置中设置每层的输入、相关层参数和输出，计算所需存储空间大小并记录需要后向传播的参数；后向设置中从输出层到输入层后向判断哪些 Blobs 对网络损失有影响，跳过那些对网络损失没有影响的 Blobs；最后读取网络学习率和权重衰减值求解网络。

```
1.  template <typename Dtype>
2.  void Net<Dtype>::Init(const NetParameter& in_param) {
```

```
    // 设置网络状态的阶段
3.  phase_ = in_param.state().phase();
    // 将规则应用到网络结构中，使用网络参数初始化网络结构
4.  NetParameter filtered_param;
5.  FilterNet(in_param, &filtered_param);
6.  LOG(INFO) << "Initializing net from parameters: " << std::endl
                  << filtered_param.DebugString();
    // 创建 filtered_param 变量的副本
7.  NetParameter param;
8.  InsertSplits(filtered_param, &param);
    // 构建所有网络层并将它们连接起来
9.  name_ = param.name();
10. map<string, int> blob_name_to_idx;
11. set<string> available_blobs;
12. CHECK(param.input_dim_size() == 0 || param.input_shape_size() == 0)
        << "Must specify either input_shape OR deprecated input_dim, not both.";
13. if (param.input_dim_size() > 0) {
      // Deprecated 4D dimensions.
14.   CHECK_EQ(param.input_size() * 4, param.input_dim_size())
          << "Incorrect input blob dimension specifications.";
15. } else {
16.   CHECK_EQ(param.input_size(), param.input_shape_size())
          << "Exactly one input_shape must be specified per input.";
17. }
18. memory_used_ = 0;
    // 设置输入数据块
19. for (int input_id = 0; input_id < param.input_size(); ++input_id) {
20.   const int layer_id = -1;  // 输入层的默认层 ID 为 -1
21.   APPendTop(param, layer_id, input_id, &available_blobs, &blob_name_to_idx);
22. }
23. DLOG(INFO) << "Memory required for data: " << memory_used_ * sizeof(Dtype);
    // 设置每层的输入和输出
24. bottom_vecs_.resize(param.layer_size());
25. top_vecs_.resize(param.layer_size());
26. bottom_id_vecs_.resize(param.layer_size());
27. param_id_vecs_.resize(param.layer_size());
28. top_id_vecs_.resize(param.layer_size());
29. bottom_need_backward_.resize(param.layer_size());
30. for (int layer_id = 0; layer_id < param.layer_size(); ++layer_id) {
```

```
    // 如果没有设置该层使用阶段（训练阶段或测试阶段），
    // 则从网络的阶段状态（训练阶段或测试阶段）中继承
31.     if (!param.layer(layer_id).has_phase()) {
32.       param.mutable_layer(layer_id)->set_phase(phase_);
33.     }
      // 设置层参数
34.     const LayerParameter& layer_param = param.layer(layer_id);
35.     layers_.push_back(LayerRegistry<Dtype>::CreateLayer(layer_param));
36.     layer_names_.push_back(layer_param.name());
37.     LOG(INFO) << "Creating Layer " << layer_param.name();
38.     bool need_backward = false;
      // 计算当前层的输入和输出
39.     for (int bottom_id = 0; bottom_id < layer_param.bottom_size();
            ++bottom_id) {
40.       const int blob_id = APPendBottom(param, layer_id, bottom_id,
41.                                        &available_blobs, &blob_name_to_idx);
        // 保存需要后向传播的块存储
42.       need_backward |= blob_need_backward_[blob_id];
43.     }
44.     int num_top = layer_param.top_size();
45.     for (int top_id = 0; top_id < num_top; ++top_id) {
46.       APPendTop(param, layer_id, top_id, &available_blobs, &blob_name_to_idx);
47.     }
      // 如果当前层设置 AutoTopBlobs() 为 true 并且 LayerParameter 指定 Top 块
      // 的数量少于需要的数量（由 ExactNumTopBlobs()或 MinTopBlobs()指定）
      // ，需要通过以下代码分配
48.     Layer<Dtype>* layer = layers_[layer_id].get();
49.     if (layer->AutoTopBlobs()) {
50.       const int needed_num_top =
51.           std::max(layer->MinTopBlobs(), layer->ExactNumTopBlobs());
52.       for (; num_top < needed_num_top; ++num_top) {
          // 不修改 available_blobs or blob_name_to_idx，
          //  防止这些块被用作其他层的输入
53.         APPendTop(param, layer_id, num_top, NULL, NULL);
54.       }
55.     }
      // 层层连接之后，设置每层的输入和输出
56.     LOG(INFO) << "Setting up " << layer_names_[layer_id];
57.     layers_[layer_id]->SetUp(bottom_vecs_[layer_id], top_vecs_[layer_id]);
```

```
58.       for (int top_id = 0; top_id < top_vecs_[layer_id].size(); ++top_id) {
59.         if (blob_loss_weights_.size() <= top_id_vecs_[layer_id][top_id]) {
60.           blob_loss_weights_.resize(top_id_vecs_[layer_id][top_id] + 1, Dtype(0));
61.         }
62.         blob_loss_weights_[top_id_vecs_[layer_id][top_id]] = layer->loss(top_id);
      //输出 Top 层的尺寸信息
63.         LOG(INFO) << "Top shape: " << top_vecs_[layer_id][top_id]->shape_string();
64.         if (layer->loss(top_id)) {
65.           LOG(INFO) << "    with loss weight " << layer->loss(top_id);
66.         }
          // 计算当前层 Top 所用存储空间大小
67.         memory_used_ += top_vecs_[layer_id][top_id]->count();
68.     }
        // 输出所需存储空间大小
69.     DLOG(INFO) << "Memory required for data: " << memory_used_ * sizeof(Dtype);
70.     const int param_size = layer_param.param_size();
71.     const int num_param_blobs = layers_[layer_id]->blobs().size();
72.     CHECK_LE(param_size, num_param_blobs)
            << "Too many params specified for layer " << layer_param.name();
73.     ParamSpec default_param_spec;
74.     for (int param_id = 0; param_id < num_param_blobs; ++param_id) {
75.       const ParamSpec* param_spec = (param_id < param_size) ?
              &layer_param.param(param_id) : &default_param_spec;
76.       const bool param_need_backward = param_spec->lr_mult() > 0;
          // 记录需要后向传播的参数
77.       need_backward |= param_need_backward;
78.       layers_[layer_id]->set_param_propagate_down(param_id,
79.                                                   param_need_backward);
80.     }
81.     for (int param_id = 0; param_id < num_param_blobs; ++param_id) {
82.       APPendParam(param, layer_id, param_id);
83.     }
        // 设置后向传播标记
84.     layer_need_backward_.push_back(need_backward);
85.     if (need_backward) {
86.       for (int top_id = 0; top_id < top_id_vecs_[layer_id].size(); ++top_id) {
87.         blob_need_backward_[top_id_vecs_[layer_id][top_id]] = true;
88.       }
89.     }
```

```
90.     }
        // 从输出层到输入层后向判断哪些 Blobs 对网络损失有影响。
        // 跳过那些对网络损失没有影响的 Blobs。
91.     set<string> blobs_under_loss;
        // 遍历网络中所有层
92.     for (int layer_id = layers_.size() - 1; layer_id >= 0; --layer_id) {
93.       bool layer_contributes_loss = false;
94.       for (int top_id = 0; top_id < top_vecs_[layer_id].size(); ++top_id) {
95.         const string& blob_name = blob_names_[top_id_vecs_[layer_id][top_id]];
96.         if (layers_[layer_id]->loss(top_id) ||
                (blobs_under_loss.find(blob_name) != blobs_under_loss.end())) {
              // 设置当前层对网络损失有影响
97.           layer_contributes_loss = true;
98.           break;
99.         }
100.      }
101.      if (!layer_contributes_loss) { layer_need_backward_[layer_id] = false; }
102.      if (layer_need_backward_[layer_id]) {
103.        LOG(INFO) << layer_names_[layer_id] << " needs backward computation.";
104.      } else {
105.        LOG(INFO) << layer_names_[layer_id]
                      << " does not need backward computation.";
106.      }
107.      for (int bottom_id = 0; bottom_id < bottom_vecs_[layer_id].size();
              ++bottom_id) {
108.        if (layer_contributes_loss) {
109.          const string& blob_name =
                  blob_names_[bottom_id_vecs_[layer_id][bottom_id]];
110.          blobs_under_loss.insert(blob_name);
111.        } else {
112.          bottom_need_backward_[layer_id][bottom_id] = false;
113.        }
114.      }
115.    }
        // Handle force_backward if needed.
116.    if (param.force_backward()) {
117.      for (int layer_id = 0; layer_id < layers_.size(); ++layer_id) {
118.        layer_need_backward_[layer_id] = true;
119.        for (int bottom_id = 0;
```

```
                bottom_id < bottom_need_backward_[layer_id].size(); ++bottom_id) {
120.            bottom_need_backward_[layer_id][bottom_id] =
                    bottom_need_backward_[layer_id][bottom_id] ||
                    layers_[layer_id]->AllowForceBackward(bottom_id);
121.            blob_need_backward_[bottom_id_vecs_[layer_id][bottom_id]] =
                    blob_need_backward_[bottom_id_vecs_[layer_id][bottom_id]] ||
                    bottom_need_backward_[layer_id][bottom_id];
122.        }
123.        for (int param_id = 0; param_id < layers_[layer_id]->blobs().size();
                ++param_id) {
124.            layers_[layer_id]->set_param_propagate_down(param_id, true);
125.        }
126.    }
127. }
    // 最后，所有其他剩余的 blobs 都是输出 blobs
128. for (set<string>::iterator it = available_blobs.begin();
129.        it != available_blobs.end(); ++it) {
130.    LOG(INFO) << "This network produces output " << *it;
131.    net_output_blobs_.push_back(blobs_[blob_name_to_idx[*it]].get());
132.    net_output_blob_indices_.push_back(blob_name_to_idx[*it]);
133. }
134. for (size_t blob_id = 0; blob_id < blob_names_.size(); ++blob_id) {
135.    blob_names_index_[blob_names_[blob_id]] = blob_id;
136. }
137. for (size_t layer_id = 0; layer_id < layer_names_.size(); ++layer_id) {
138.    layer_names_index_[layer_names_[layer_id]] = layer_id;
139. }
    // 获得学习率和权重衰减值
140. GetLearningRateAndWeightDecay();
141. debug_info_ = param.debug_info();
142. LOG(INFO) << "Network initialization done.";
143. LOG(INFO) << "Memory required for data: " << memory_used_ * sizeof(Dtype);
144. }
```

②APPendTop(const NetParameter& param, const int layer_id, const int top_id, set<string>* available_blobs, map<string, int>* blob_name_to_idx)

该函数的主要功能是将新的输入Blob（layer_id = -1）或者顶层Blob（layer_id >= 0）添加到网络结构中，与网络现有结构结合在一起，主要在网络构建过程中使用。

```
template <typename Dtype>
void Net<Dtype>::APPendTop(const NetParameter& param, const int layer_id,
                           const int top_id, set<string>* available_blobs,
                           map<string, int>* blob_name_to_idx) {
  shared_ptr<LayerParameter> layer_param((layer_id >= 0) ?
    (new LayerParameter(param.layer(layer_id))) : NULL);
  const string& blob_name = layer_param ?
      (layer_param->top_size() > top_id ?
          layer_param->top(top_id) : "(automatic)") : param.input(top_id);
  // 检查是否是同址计算
  if (blob_name_to_idx && layer_param && layer_param->bottom_size() > top_id &&
      blob_name == layer_param->bottom(top_id)) {
    // 同址计算
    LOG(INFO) << layer_param->name() << " -> " << blob_name << " (in-place)";
    top_vecs_[layer_id].push_back(blobs_[(*blob_name_to_idx)[blob_name]].get());
    top_id_vecs_[layer_id].push_back((*blob_name_to_idx)[blob_name]);
  } else if (blob_name_to_idx &&
             blob_name_to_idx->find(blob_name) != blob_name_to_idx->end()) {
    // 如果不是同址计算并且含有多个重复 Blobs，则提示错误
    LOG(FATAL) << "Duplicate blobs produced by multiple sources.";
  } else {
    // 输出信息
    if (layer_param) {
      LOG(INFO) << layer_param->name() << " -> " << blob_name;
    } else {
      LOG(INFO) << "Input " << top_id << " -> " << blob_name;
    }
    shared_ptr<Blob<Dtype> > blob_pointer(new Blob<Dtype>());
    const int blob_id = blobs_.size();
    blobs_.push_back(blob_pointer);
    blob_names_.push_back(blob_name);
    blob_need_backward_.push_back(false);
    if (blob_name_to_idx) { (*blob_name_to_idx)[blob_name] = blob_id; }
    if (layer_id == -1) {
      // 设置输入 Blob 的尺寸
      if (param.input_dim_size() > 0) {
        blob_pointer->Reshape(param.input_dim(top_id * 4),
                              param.input_dim(top_id * 4 + 1),
                              param.input_dim(top_id * 4 + 2),
```

```
                                        param.input_dim(top_id * 4 + 3));
27.          } else {
28.            blob_pointer->Reshape(param.input_shape(top_id));
29.          }
30.          net_input_blob_indices_.push_back(blob_id);
31.          net_input_blobs_.push_back(blob_pointer.get());
32.        } else {
33.          top_id_vecs_[layer_id].push_back(blob_id);
34.          top_vecs_[layer_id].push_back(blob_pointer.get());
35.        }
36.      }
37.      if (available_blobs) { available_blobs->insert(blob_name); }
38.    }
```

③ APPendBottom(const NetParameter& param, const int layer_id, const int bottom_id, set<string>* available_blobs, map<string, int>* blob_name_to_idx)

该函数的主要功能是将新的底层 Blob 添加到网络结构中，与网络现有结构结合在一起，主要在网络构建过程中使用。首先判断要添加的底层是否属于可用层，如果属于可用层，则将该层添加到现有网络结构中。

```
1.   template <typename Dtype>
2.   int Net<Dtype>::APPendBottom(const NetParameter& param,
         const int layer_id, const int bottom_id,
         set<string>* available_blobs, map<string, int>* blob_name_to_idx) {
3.     const LayerParameter& layer_param = param.layer(layer_id);
4.     const string& blob_name = layer_param.bottom(bottom_id);
       // 判断要添加的底层是否属于可用层
5.     if (available_blobs->find(blob_name) == available_blobs->end()) {
6.       LOG(FATAL) << "Unknown blob input " << blob_name
                    << " (at index " << bottom_id << ") to layer " << layer_id;
7.     }
       // 将 id 为 blob_id 的层作为 id 为 layer_id 层的底层 Blob。
8.     const int blob_id = (*blob_name_to_idx)[blob_name];
9.     LOG(INFO) << layer_names_[layer_id] << " <- " << blob_name;
10.    bottom_vecs_[layer_id].push_back(blobs_[blob_id].get());
11.    bottom_id_vecs_[layer_id].push_back(blob_id);
12.    available_blobs->erase(blob_name);
13.    const bool need_backward = blob_need_backward_[blob_id];
14.    bottom_need_backward_[layer_id].push_back(need_backward);
15.    return blob_id;
16.  }
```

5.2.4 网络求解解析

网络求解中需要设置网络相关求解参数，如网络学习率、网络权重衰减值、网络迭代最大次数等。

1. 示例

网络求解需要设置许多参数，一般保存在 solver.prototxt 文件中，具体参数如下：

```
# 训练和测试网络结构文件路径
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter 指定每次测试时，迭代 100 次
test_iter: 100
# test_interval 指定迭代训练 500 次之后进行一次测试
test_interval: 500
# 网络学习率
base_lr: 0.01
# 网络动量值
momentum: 0.9
# 网络权重衰减值
weight_decay: 0.0005
# 网络学习率策略
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# 每迭代 100 次，输出一次信息
display: 100
# 最大迭代次数为 10000
max_iter: 10000
#保存中间结果
snapshot: 5000
#保存的中间结果的前缀路径
snapshot_prefix: "examples/mnist/lenet"
# 设置 CPU 或者 GPU 模式
solver_mode: GPU
```

学习率改变有多种策略，目前经常使用的有如下几种：

```
fixed: 返回 base_lr
step: 返回 base_lr * gamma ^ (floor(iter / step))
exp: 返回 base_lr * gamma ^ iter
inv: 返回 base_lr * (1 + gamma * iter) ^ (- power)
multistep:和 step 模式相似，但该模式允许使用 stepvalue 设置多个不同的步长
poly: 多项式衰减，利用 max_iter 将学习率趋于 0。返回 base_lr *(1 - iter/max_iter) ^ (power)
sigmoid: sigmod 函数衰减模式。返回 base_lr *( 1/(1 + exp(-gamma * (iter - stepsize))))
```

其中 base_lr、max_iter、gamma、step、stepsize、stepvalue 和 power 定义在 solver.prototxt 文件中, iter 表示当前迭代的次数。

2. 源码解析

整体简介

该层对应的源代码文件为 solver.cpp，类定义文件为 solver.hpp，对应类名为 Solver。该类包含函数较多，这里只列举其中的几个函数，有兴趣的读者，可以自己去查找对应的源码理解。

主要函数

①Init(const SolverParameter& param)

该函数的主要功能是初始化求解网络相关参数，利用相关参数分别初始化训练网络和测试网络。

```
1.  template <typename Dtype>
2.  void Solver<Dtype>::Init(const SolverParameter& param) {
3.    LOG(INFO) << "Initializing solver from parameters: " << std::endl
                << param.DebugString();
4.    param_ = param;
5.    CHECK_GE(param_.average_loss(), 1) << "average_loss should be non-negative.";
6.    if (param_.random_seed() >= 0) {
7.      Caffe::set_random_seed(param_.random_seed());
8.    }
      // 初始化训练网络
9.    InitTrainNet();
      // 初始化测试网络
10.   InitTestNets();
11.   LOG(INFO) << "Solver scaffolding done.";
12.   iter_ = 0;
13.   current_step_ = 0;
```

```
14. }
```

②InitTrainNet()

该函数的主要功能是初始化训练网络，训练网络可以使用 train_net_param、train_net、net_param、net 四个参数指定，然后设置准确的网络状态。

```
1.  template <typename Dtype>
2.  void Solver<Dtype>::InitTrainNet() {
3.    const int num_train_nets = param_.has_net() + param_.has_net_param() +
          param_.has_train_net() + param_.has_train_net_param();
4.    const string& field_names = "net, net_param, train_net, train_net_param";
      // 判断 SolverParameter 是否设置指定训练网络的参数
5.    CHECK_GE(num_train_nets, 1) << "SolverParameter must specify a train net "
          << "using one of these fields: " << field_names;
6.    CHECK_LE(num_train_nets, 1) << "SolverParameter must not contain more than "
          << "one of these fields specifying a train_net: " << field_names;
7.    NetParameter net_param;
8.    if (param_.has_train_net_param()) { // 从 train_net_param 中创建训练网络
9.      LOG(INFO) << "Creating training net specified in train_net_param.";
10.     net_param.CopyFrom(param_.train_net_param());
11.   } else if (param_.has_train_net()) {// 从 train_net 文件中创建训练网络
12.     LOG(INFO) << "Creating training net from train_net file: "
                  << param_.train_net();
13.     ReadNetParamsFromTextFileOrDie(param_.train_net(), &net_param);
14.   }
15.   if (param_.has_net_param()) { // 从 net_param 中创建训练网络
16.     LOG(INFO) << "Creating training net specified in net_param.";
17.     net_param.CopyFrom(param_.net_param());
18.   }
19.   if (param_.has_net()) {// 从 net 文件中创建训练网络
20.     LOG(INFO) << "Creating training net from net file: " << param_.net();
21.     ReadNetParamsFromTextFileOrDie(param_.net(), &net_param);
22.   }
      // 设置准确的网络状态，网络状态优先级顺序从低到高分别为 sovler 默认值，net_param 参数指定值，train_state 参数指定值
23.   NetState net_state;
24.   net_state.set_phase(TRAIN);
25.   net_state.MergeFrom(net_param.state());
26.   net_state.MergeFrom(param_.train_state());
27.   net_param.mutable_state()->CopyFrom(net_state);
```

```
28.     net_.reset(new Net<Dtype>(net_param));
29.   }
```

③InitTestNets()

该函数的主要功能是初始化测试网络，测试网络可以使用 test_net_param、test_net、net_param、net 四个参数指定，然后保证为每个测试网络都指定了 test_iter 参数，通过统计 test_net_param、test_net、net_param、net 四个参数分别指定的网络个数，使用不同的方式创建测试网络，最后分别设置准确的网络状态。

```
1.    template <typename Dtype>
2.    void Solver<Dtype>::InitTestNets() {
        // 检查是否通过 net_param 指定网络参数或通过 net_file 指定网络文件
3.      const bool has_net_param = param_.has_net_param();
4.      const bool has_net_file = param_.has_net();
5.      const int num_generic_nets = has_net_param + has_net_file;
6.      CHECK_LE(num_generic_nets, 1)
            << "Both net_param and net_file may not be specified.";
7.      const int num_test_net_params = param_.test_net_param_size();
8.      const int num_test_net_files = param_.test_net_size();
9.      const int num_test_nets = num_test_net_params + num_test_net_files;
10.     if (num_generic_nets) {
          // 保证为每个测试网络都指定了 test_iter 参数
11.       CHECK_GE(param_.test_iter_size(), num_test_nets)
              << "test_iter must be specified for each test network.";
12.     } else {
            // 保证为每个测试网络都指定了 test_iter 参数
13.       CHECK_EQ(param_.test_iter_size(), num_test_nets)
              << "test_iter must be specified for each test network.";
14.     }
          // 如果存在通用网络（通过 net 或者 net_param 指定，
          // 而不是 test_net 或者 test_net_param），可能会有多个测试网络，
          // net 或者 net_param 指定的测试网络的真实数目为
          // 除了 test_net 或者 test_net_param 指定所需的 test_iters 数目之外，剩余的 test_iters 的数目

          // 通过 net 或者 net_param 指定的测试网络的真实数目
          const int num_generic_net_instances = param_.test_iter_size() - num_test_nets;
          // 测试网络的总数目（通过 test_net 或者 test_net_param 指定的测试网络的真实数目+通过 net 或者 net_param 指定的测试网络的真实数目）
15.       const int num_test_net_instances = num_test_nets + num_generic_net_instances;
```

```
16.     if (param_.test_state_size()) {
17.       CHECK_EQ(param_.test_state_size(), num_test_net_instances)
              << "test_state must be unspecified or specified once per test net.";
18.     }
19.     if (num_test_net_instances) {
20.       CHECK_GT(param_.test_interval(), 0);
21.     }
22.     int test_net_id = 0;
23.     vector<string> sources(num_test_net_instances);
24.     vector<NetParameter> net_params(num_test_net_instances);
        // 从 test_net_param 中创建测试网络
25.     for (int i = 0; i < num_test_net_params; ++i, ++test_net_id) {
26.         sources[test_net_id] = "test_net_param";
27.         net_params[test_net_id].CopyFrom(param_.test_net_param(i));
28.     }
        // 从 test_net 文件中创建测试网络
29.     for (int i = 0; i < num_test_net_files; ++i, ++test_net_id) {
30.         sources[test_net_id] = "test_net file: " + param_.test_net(i);
31.         ReadNetParamsFromTextFileOrDie(param_.test_net(i),
              &net_params[test_net_id]);
32.     }
33.     const int remaining_test_nets = param_.test_iter_size() - test_net_id;
34.     if (has_net_param) {   // 从 net_param 中创建测试网络
35.       for (int i = 0; i < remaining_test_nets; ++i, ++test_net_id) {
36.         sources[test_net_id] = "net_param";
37.         net_params[test_net_id].CopyFrom(param_.net_param());
38.       }
39.     }
40.     if (has_net_file) {   // 从 net 文件中创建测试网络
41.       for (int i = 0; i < remaining_test_nets; ++i, ++test_net_id) {
42.         sources[test_net_id] = "net file: " + param_.net();
43.         ReadNetParamsFromTextFileOrDie(param_.net(), &net_params[test_net_id]);
44.       }
45.     }
46.     test_nets_.resize(num_test_net_instances);
47.     for (int i = 0; i < num_test_net_instances; ++i) {
        // 设置准确的网络状态，网络状态优先级顺序从低到高分别为 sovler 默认值，
net_param 参数指定值，test_state 参数指定值
48.       NetState net_state;
```

```
49.         net_state.set_phase(TEST);
50.         net_state.MergeFrom(net_params[i].state());
51.         if (param_.test_state_size()) {
52.           net_state.MergeFrom(param_.test_state(i));
53.         }
54.         net_params[i].mutable_state()->CopyFrom(net_state);
55.         LOG(INFO)
              << "Creating test net (#" << i << ") specified by " << sources[i];
56.         test_nets_[i].reset(new Net<Dtype>(net_params[i]));
57.         test_nets_[i]->set_debug_info(param_.debug_info());
58.       }
59.     }
```

参考文献

[1] Jia Y, Shelhamer E, Donahue J, et al. Caffe: Convolutional Architecture for Fast Feature Embedding[J]. Eprint Arxiv, 2014:675-678.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton.Imagenet classification with deep convolutional neural networks. In Proceedings of NIPS, 2012.

[3] Nair V, Hinton G E. Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair[C]// International Conference on Machine Learning. 2010:807-814.

[4] Caffe. http://caffe.berkeleyvision.org/

第 6 章

深度学习工具 Pylearn2

Pylearn2 [1]是基于 Theano [2]开发的深度学习工具，可以方便地定义参数，快速训练模型，Theano 是用来有效定义、优化和计算关于多维数组数学表达式的 Python 类库。

6.1 Pylearn2 的安装

本节将以 Ubuntu14.04-64 位操作系统为例，详细介绍如何安装使用 Pylearn2 深度学习工具。

6.1.1 相关依赖安装

（1）Python 2 >= 2.6 or Python 3 >= 3.3

Python 是一种方便使用的程序设计语言，且是面向对象、解释型语言。

安装命令：sudo apt—get install python—dev

（2）NumPy >= 1.7.1

Numpy 是使用 Python 进行科学计算的基础开源工具包，包括 *N* 维数组对象、成熟的函数库、集成 C++和 Fortran 代码的工具和实用的线性代数、傅里叶变换和随机数。

安装命令：sudo apt-get install python-numpy

（3）SciPy >= 0.11

SciPy 是进行科学计算的 Python 开源工具包，为 Python 提供强大的科学计算能力。

安装命令：sudo apt-get install python-scipy

（4）Nose >= 1.3.0

Nose 是一款方便安装及维护的 Python 测试工具包。

安装命令：sudo apt-get install python—nose

（5）Git

Git 是一个高效的分布式版本管理系统，并且是免费、开源的，具有使用方便，高效管理的特点。

安装命令：sudo apt-get install git

（6）BALS（ATLAS 或 OpenBLAS）

Basic Linear Algebra Subprograms(BLAS)是基础线性代数子程序库，提供了许多线性代数操作函数。

Automatically Tuned Linear Algebra Software(ATLAS)是基于 BLAS 实现的计算平台。

安装命令：sudo apt-get install libatlas-base-dev

OpenBLAS 是基于 BLAS 实现的计算平台，计算效率要比 ATLAS 高。

安装命令：sudo apt-get install libopenblas-dev

（7）Theano

Theano 是用来有效地定义、优化和计算关于多维数组数学表达式的 Python 类库。

安装命令：

（1）下载 Theano 安装包文件

执行命令：git clone git://github.com/Theano/Theano.git

（2）进入 Theano 文件夹

执行命令：cd Theano

（3）安装 Theano 软件

执行命令：python setup.py install

6.1.2 安装 Pylearn2

（1）下载 Pylearn2 安装包文件
执行命令：git clone git://github.com/lisa-lab/pylearn2.git
（2）安装 Pylearn2
进入 Pylearn2 文件夹
执行命令：cd Pylearn2
安装 Pylearn2
执行命令：sudo python setup.py install
（3）设置环境变量 PYTHON2_DATA_PATH, PYLEARN2_VIEWR_COMMAND
安装图形显示软件 gwenview
执行命令：sudo apt—get install gwenview
在/etc/profile 中添加环境变量：
执行命令：sudo vim /etc/profile
在文件中最后一行添加内容：
export PYLEARN2_DATA_PATH=/home/wpf/cnn/pylearn2-master/data
export PYLEARN2_VIEWER_COMMAND=/usr/bin/gwenview
使设置的环境变量生效
执行命令：source /etc/profile

6.2 Pylearn2 的使用

本节主要讲解如何使用 Pylearn2 训练深度模型，该例主要参考文献[3]，以 cifar10 数据[4]为例，假设测试机器上的 Pylearn2 的安装根目录为/home/wpf/cnn/pylearn2-master/，本例的根目录为/home/wpf/cnn/pylearn2-master/pylearn2/scripts/tutorials/grbm_smd/，使用 Binary Gaussian RBM（GRBM）模型，denoising score matching（SMD）目标函数。

（1）下载 cifar10 测试数据

进入数据文件夹

执行命令：cd /home/wpf/cnn/pylearn2-master/data/cifar10

下载 cifar10 数据压缩包

执行命令：wget http://www.cs.utoronto.ca/~kriz/cifar-10-python.tar.gz

解压 cifar 数据压缩包

执行命令：tar xvf cifar-10-python.tar.gz

注：下载数据的另外一种方式，sh /home/wpf/cnn/pylearn2-master/pylearn2/scripts/datasets/download_cifar10.sh。

（2）对下载的数据进行预处理

数据预处理：cifar10 数据集[4]包含 50000 张训练样本，10000 张测试样本，样本为 32×32 像素的彩色图片；该实验中将从每张图片中抽取 8×8 的切片，并对这些切片进行正则化和 ZCA 白化预处理操作，共有 15 万张切片作为训练样本，保存在 cifar10_preprocessed_train.pkl 文件中。

操作流程：

进入/home/wpf/cnn/pylearn2-master/pylearn2/scripts/tutorials/grbm_smd 文件夹；

执行命令：cd /home/wpf/cnn/pylearn2-master/pylearn2/scripts/tutorials/grbm_smd

预处理 cifar10 数据

执行命令：python make_dataset.py

（3）使用数据进行训练模型

使用预处理数据文件 cifar10_preprocessed_train.pkl 和参数定义文件 cifar_grbm_smd.yaml 训练模型参数，最终会得到名称为 cifar_grbm_smd.pkl 的模型文件。

执行命令：sudo python /home/wpf/cnn/pylearn2-master/pylearn2/scripts/train.py cifar_grbm_smd.yaml

cifar_grbm_smd.yaml 文件内容如下：

```
#与 pylearn2 进行交互的主要文件
!obj:pylearn2.train.Train {
    #数据集文件
    dataset: !pkl: "cifar10_preprocessed_train.pkl",
    #初始化模型参数
model: !obj:pylearn2.models.rbm.GaussianBinaryRBM {
    #由于图片为 8×8 的 RGB 图片，RBM 模型中可视单元数应为 192
```

```
            nvis : 192,
            #设置 RBM 模型中隐藏单元数
            nhid : 400,
            irange : 0.05,
        energy_function_class : !obj:pylearn2.energy_functions.rbm_energy.grbm_type_1 {},
            learn_sigma : True,
            #sigma 的初始值
            init_sigma : .4,
            #隐层的偏置的初始值
            init_bias_hid : -2.,
            mean_vis : False,
            sigma_lr_scale : 1e-3
    },
    #初始化算法参数
        algorithm: !obj:pylearn2.training_algorithms.sgd.SGD {
            learning_rate : 1e-1,
            batch_size : 5,
            monitoring_batches : 20,

            monitoring_dataset : !pkl: "cifar10_preprocessed_train.pkl",
            cost : !obj:pylearn2.costs.ebm_estimation.SMD {
                corruptor : !obj:pylearn2.corruption.GaussianCorruptor {
                        stdev : 0.4
                },
            },

            termination_criterion : !obj:pylearn2.termination_criteria.MonitorBased {
                prop_decrease : 0.01,
                N : 1,
            },
            },
        extensions : [!obj:pylearn2.training_algorithms.sgd.MonitorBasedLRAdjuster {}],
        save_freq : 1
    }
```

可以看出 cifar_grbm_smd.yaml 文件主要包含数据、模型和算法三个方面相关的参数，能更方便地使用现有模型和算法进行实验。

（4）查看模型参数信息

使用模型文件 cifar_grbm_smd.pkl 和 cifar_grbm_smd.pkl 文件查看学习到的权重信息，结果如图 6-1 所示。

执行命令：python /home/wpf/cnn/pylearn2-master/pylearn2/scripts/show_weights.py --out weights.jpg/home/wpf/cnn/pylearn2-master/pylearn2/scripts/tutorials/grbm_smd/cifar_grbm_smd.pkl

注：--out weights.jpg 可以让生成的图片保存到本地磁盘，保存的图片文件名为 weights.jpg。

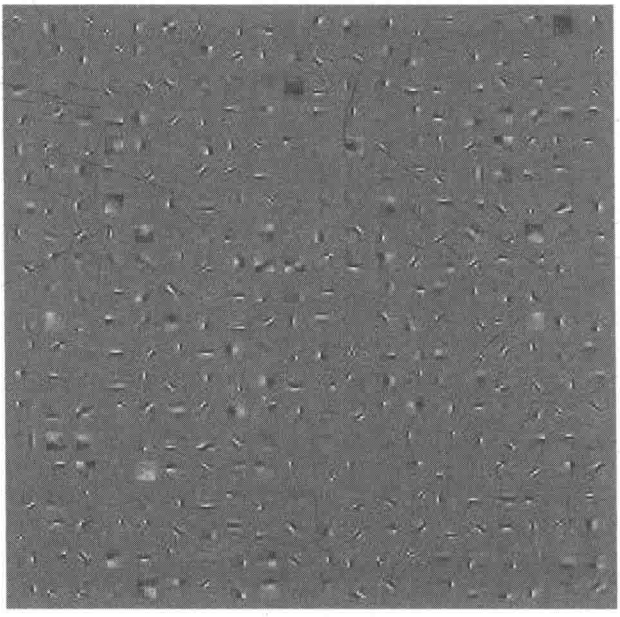


图 6-1　经过训练学习到的权重信息（参考文献[3]）

（5）查看模型文件 cifar_grbm_smd.pkl 的结构信息

在 /home/wpf/cnn/pylearn2-master/pylearn2/scripts/tutorials/grbm_smd/ 目录下输入：python

```
>>> from pylearn2.utils import serial
>>> cifar10_model = serial.load('cifar_grbm_smd.pkl')
```

获取权重参数值

```
>>> cifar10_model.get_weights()
```

将显示出该模型训练得到的权重参数值

参考文献

[1] Pylearn2. http://deeplearning.net/software/pylearn2/.

[2] Theano. http://deeplearning.net/software/theano/.

[3] http://deeplearning.net/software/pylearn2/tutorial/index.html#tutorial.

[4] http://www.cs.toronto.edu/~kriz/cifar.html.

深度学习实践篇
（入门与进阶）

第 7 章

基于深度学习的手写数字识别

7.1 数据介绍

7.1.1 MNIST 数据集

MNIST 数据集是手写数字的数据库，其中训练集 60000 例，测试集 10000 例。其中数字大小已经标准化，并且集中在一个固定大小的图像内，图像大小为 28×28 像素。在其官网中有 4 个文件可供下载，分别为：训练数据集、训练数据集标签、测试数据集、测试数据集标签。其中标签文件对应数据集中每一例表示的具体内容（打印字体数字）。

MNIST 数据集是以多维矢量矩阵的形式存储。

（1）训练数据集文件格式如表 7-1 所示。

表 7-1 训练数据集文件格式[1]

offset	type	value	description
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
...	...	...	...
xxxx	unsigned byte	??	pixel

表 7-1 中，从 0016 号位移（offest）开始，表示的是每个像素的信息，对应的??表示的是该像素的对应灰度值，其范围在 0～255 之间。

（2）训练数据集标签文件格式如表 7-2 所示。

表 7-2　训练数据集标签文件格式[1]

offset	type	value	description
0000	32 bit integer	0x00000801(2049)	magic number
0004	32 bit integer	60000	number of items
0016	unsigned byte	??	label
0017	unsigned byte	??	label
...	...	...	...
xxxx	unsigned byte	??	label

表 7-2 中，从 0016 号位移（offest）开始，表示的是每个像素对应的标签信息，对应的??表示的是该像素的标签值，在本例中，其取值是 0～9 之间的一个整数。

（3）测试数据集文件格式如表 7-3 所示。

表 7-3　测试数据集文件格式[1]

offset	type	value	description
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	10000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
...	...	...	...
xxxx	unsigned byte	??	pixel

表 7-3 中，从 0016 号位移（offest）开始，表示的是测试数据集中每个像素的信息，对应的??表示的是该像素的对应灰度值，其范围在 0～255 之间。

（4）测试数据集标签文件格式如表 7-4 所示。

表 7-4　测试数据集标签文件格式[1]

offset	type	value	description
0000	32 bit integer	0x00000801(2049)	magic number
0004	32 bit integer	10000	number of items
0016	unsigned byte	??	label
...	...	...	...
0017	unsigned byte	??	label

表 7-4 中，从 0016 号位移（offest）开始，表示的是对测试数据集中的每个像素对应的预测结果，对应的??表示的是该像素的标签值，在本例中，其取值是 0～9 之间的一个整数。

测试数据集第一张图片数据，如图 7-1 所示。

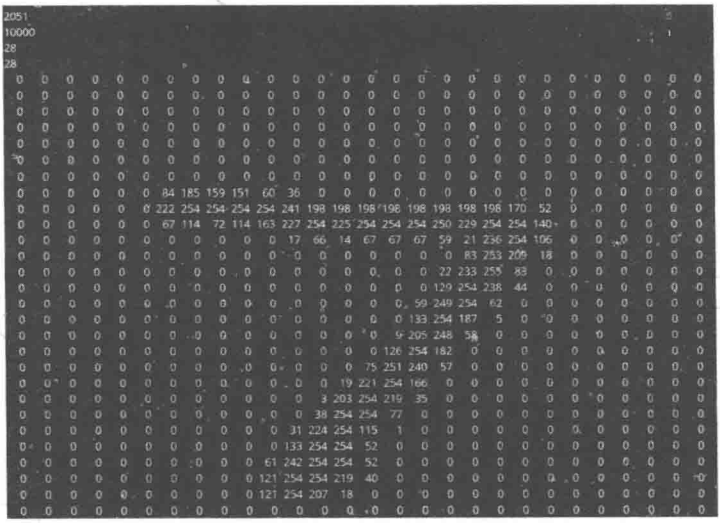


图 7-1　测试数据集第一张图片数据

测试数据集前十行数据，如图 7-2 所示。

图 7-2　测试数据集前十行数据

7.1.2　提取 MNIST 数据集图片

根据前一节所讲述的文件格式设计程序提取数据集中图片，程序流程如下：

（1）打开目标文件；

（2）判断文件类型（magic number）；

（3）跳过文件定义部分数据（前 4 个字）；

（4）新建图片文件；

（5）循环将像素值写入图片文件；

（6）关闭图像文件；

（7）重复步骤（4）直至完成。

提取后训练集 60000 张图片，测试集 10000 张图片。其效果如图 7-3 所示。

图 7-3　从测试数据集中提取图片

7.2　手写数字识别流程

7.2.1　模型介绍

使用 Caffe 在 MNIST 数据集对网络进行训练，使用的网络模型为 LeNet 模型。其网络模型文件内容如下：

```
name: "LeNet"
layer {
  name: "mnist"
```

```
    type: "Data"
    top: "data"
    top: "label"
    include {
      phase: TRAIN
    }
    transform_param {
      scale: 0.00390625
    }
    data_param {
      source: "examples/mnist/mnist_train_lmdb"
      batch_size: 64
      backend: LMDB
    }
  }
  layer {
    name: "mnist"
    type: "Data"
    top: "data"
    top: "label"
    include {
      phase: TEST
    }
    transform_param {
      scale: 0.00390625
    }
    data_param {
      source: "examples/mnist/mnist_test_lmdb"
      batch_size: 100
      backend: LMDB
    }
  }
  layer {
    name: "conv1"
    type: "Convolution"
    bottom: "data"
    top: "conv1"
    param {
      lr_mult: 1
```

```
    }
    param {
      lr_mult: 2
    }
    convolution_param {
      num_output: 20
      kernel_size: 5
      stride: 1
      weight_filler {
        type: "xavier"
      }
      bias_filler {
        type: "constant"
      }
    }
  }
  layer {
    name: "pool1"
    type: "Pooling"
    bottom: "conv1"
    top: "pool1"
    pooling_param {
      pool: MAX
      kernel_size: 2
      stride: 2
    }
  }
  layer {
    name: "conv2"
    type: "Convolution"
    bottom: "pool1"
    top: "conv2"
    param {
      lr_mult: 1
    }
    param {
      lr_mult: 2
    }
    convolution_param {
```

```
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
```

```
        type: "constant"
      }
    }
  }
  layer {
    name: "relu1"
    type: "ReLU"
    bottom: "ip1"
    top: "ip1"
  }
  layer {
    name: "ip2"
    type: "InnerProduct"
    bottom: "ip1"
    top: "ip2"
    param {
      lr_mult: 1
    }
    param {
      lr_mult: 2
    }
    inner_product_param {
      num_output: 10
      weight_filler {
        type: "xavier"
      }
      bias_filler {
        type: "constant"
      }
    }
  }
  layer {
    name: "accuracy"
    type: "Accuracy"
    bottom: "ip2"
    bottom: "label"
    top: "accuracy"
    include {
      phase: TEST
```

```
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

该网络结构包含 2 个卷积层、2 个下采样层和 2 个全连接层。

7.2.2 操作流程

（1）下载数据集

使用如下命令下载数据集：

```
cd $CAFFE_ROOT/data/mnist/get_mnist.sh./examples/mnist/create_mnist.sh
```

在命令运行完后 examples/mnist 路径下会出现两个文件夹：mnist_test_lmdb（测试数据集）和 mnist_train_lmdb（训练数据集）。

（2）调整网络的参数

Caffe 中网络参数的定义以文本形式存储在 lenet_solver.prototxt 中，其具体内容如下：

```
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500                                    //训练迭代 500 次，测试一次
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.0325                                       //学习率
momentum: 0.9                                         //动量
weight_decay: 0.0005                                  //权重的衰减
# The learning rate policy                  //学习策略：有固定学习率和每部递减学习率
lr_policy: "inv"
gamma: 0.0001
```

```
power: 0.75
# Display every 100 iterations                    //每 100 次迭代显示一次
display: 100
# The maximum number of iterations                //最大迭代次数
max_iter: 10000
# snapshot intermediate results                   //每 5000 次迭代存储一次数据
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
# solver mode: CPU or GPU                         //使用 CPU 或 GPU 进行运算
solver_mode: GPU
```

本次实验仅对 base_lr（学习率）参数进行调整。

（3）训练网络

使用如下命令：

./examples/mnist/train_lenet.sh

7.3　实验结果分析

如图 7-4 所示在 base_lr（学习率）为 0.01 时 accuracy（准确率）为 0.9902。

```
ary proto file examples/mnist/lenet_iter_10000.solverstate
I0421 20:32:49.326586  7262 solver.cpp:317] Iteration 10000, loss = 0.0032048
I0421 20:32:49.326611  7262 solver.cpp:337] Iteration 10000, Testing net (#0)
I0421 20:32:50.773252  7262 solver.cpp:404]     Test net output #0: accuracy = 0
.9902
I0421 20:32:50.773285  7262 solver.cpp:404]     Test net output #1: loss = 0.029
7917 (* 1 = 0.0297917 loss)
I0421 20:32:50.773293  7262 solver.cpp:322] Optimization Done.
I0421 20:32:50.773298  7262 caffe.cpp:222] Optimization Done.
```

图 7-4　准确率为 0.9902

实验中，共使用了 14 个不同参数，其结果如表 7-5 所示。

表 7-5　不同的 base_lr 参数对实验结果的影响

base_lr	准确率
0.01	0.9902
0.001	0.9832
0.0001	0.9415

续表

base_lr	准确率
0.05	0.9903
0.02	0.9915
0.002	0.9869
0.000 2	0.9624
0.03	0.9915
0.025	0.9929
0.04	0.9918
0.015	0.991
0.035	0.9921
0.027 5	0.9913
0.032 5	0.9911

从表 7-5 可以看出，当 base_lr（学习率）为 0.025 时，准确率最高为 0.9929。因此，最终选定的 base_lr（学习率）参数值应为 0.025。

参考文献

[1] http://yann.lecun.com/exdb/mnist/.

第 8 章

基于深度学习的图像识别

8.1 数据来源

8.1.1 Cifar10 数据集介绍

Cifar10 [1]由 60000 张 32×32 像素的 RGB 彩色图片构成，共 10 个分类。50000 张图片用于训练，10000 张用于测试（交叉验证）。这个数据集最大的特点在于将识别迁移到了普适物体，而且应用于多分类问题（姊妹数据集 Cifar-100 达到 100 类）。

8.1.2 Cifar10 数据集格式

Cifar10 数据集由训练集和测试集两部分构成，分别为 50000 张和 10000 张彩色图片。其大致内容如图 8-1 所示。

图 8-1　Cifar-10 数据集中的样例[1]

8.2　Cifar10 识别流程

8.2.1　模型介绍

使用 Caffe 在 Cifar10 数据集对网络进行训练，其网络模型文件 cifar10_full_train_test.prototxt 内容如下：

```
name: "CIFAR10_full"
layer {
  name: "cifar"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mean_file: "examples/cifar10/mean.binaryproto"
  }
  data_param {
    source: "examples/cifar10/cifar10_train_lmdb"
```

```
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "cifar"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mean_file: "examples/cifar10/mean.binaryproto"
  }
  data_param {
    source: "examples/cifar10/cifar10_test_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
```

```
        std: 0.0001
      }
      bias_filler {
        type: "constant"
      }
    }
  }
  layer {
    name: "pool1"
    type: "Pooling"
    bottom: "conv1"
    top: "pool1"
    pooling_param {
      pool: MAX
      kernel_size: 3
      stride: 2
    }
  }
  layer {
    name: "relu1"
    type: "ReLU"
    bottom: "pool1"
    top: "pool1"
  }
  layer {
    name: "norm1"
    type: "LRN"
    bottom: "pool1"
    top: "norm1"
    lrn_param {
      local_size: 3
      alpha: 5e-05
      beta: 0.75
      norm_region: WITHIN_CHANNEL
    }
  }
  layer {
    name: "conv2"
    type: "Convolution"
```

```
    bottom: "norm1"
    top: "conv2"
    param {
      lr_mult: 1
    }
    param {
      lr_mult: 2
    }
    convolution_param {
      num_output: 32
      pad: 2
      kernel_size: 5
      stride: 1
      weight_filler {
        type: "gaussian"
        std: 0.01
      }
      bias_filler {
        type: "constant"
      }
    }
  }
  layer {
    name: "relu2"
    type: "ReLU"
    bottom: "conv2"
    top: "conv2"
  }
  layer {
    name: "pool2"
    type: "Pooling"
    bottom: "conv2"
    top: "pool2"
    pooling_param {
      pool: AVE
      kernel_size: 3
      stride: 2
    }
  }
```

```
layer {
  name: "norm2"
  type: "LRN"
  bottom: "pool2"
  top: "norm2"
  lrn_param {
    local_size: 3
    alpha: 5e-05
    beta: 0.75
    norm_region: WITHIN_CHANNEL
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "norm2"
  top: "conv3"
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
}
layer {
  name: "pool3"
```

```
    type: "Pooling"
    bottom: "conv3"
    top: "pool3"
    pooling_param {
      pool: AVE
      kernel_size: 3
      stride: 2
    }
  }
  layer {
    name: "ip1"
    type: "InnerProduct"
    bottom: "pool3"
    top: "ip1"
    param {
      lr_mult: 1
      decay_mult: 250
    }
    param {
      lr_mult: 2
      decay_mult: 0
    }
    inner_product_param {
      num_output: 10
      weight_filler {
        type: "gaussian"
        std: 0.01
      }
      bias_filler {
        type: "constant"
      }
    }
  }
  layer {
    name: "accuracy"
    type: "Accuracy"
    bottom: "ip1"
    bottom: "label"
    top: "accuracy"
```

```
    include {
      phase: TEST
    }
  }
  layer {
    name: "loss"
    type: "SoftmaxWithLoss"
    bottom: "ip1"
    bottom: "label"
    top: "loss"
  }
```

该网络结构包含 3 个卷积层、3 个下采样层和 2 个全连接层。

8.2.2 操作流程

1. 使用 CIFAR10 网络训练 CIFAR10 数据集

（1）下载数据集

使用如下命令下载数据集：

```
cd $CAFFE_ROOT
./data/cifar10/get_cifar10.sh
./examples/cifar10/create_cifar10.sh
```

在命令运行完后 examples/cifar10 路径下会出现两个文件夹：cifar10_test_lmdb（测试数据集）和 cifar10_train_lmdb（训练数据集）。并且建立图片的均值文件（mean.binaryproto）。

（2）调整网络的参数

Caffe 中网络参数的定义以文本形式存储在 cifar10_full_solver.prototxt 中，其具体内容如下：

```
# 训练和测试定义文件
net: "examples/cifar10/cifar10_full_train_test.prototxt"
test_iter: 100
# 每训练迭代 1000 次，测试一次
test_interval: 1000
# 学习率
base_lr: 0.001
# 动量
```

```
momentum: 0.9
# 权重的衰减
weight_decay: 0.004
# 学习率策略
lr_policy: "fixed"
# 每 200 次迭代显示一次
display: 200
# 最大迭代次数
max_iter:17000
# 每 10000 次迭代存储一次数据
snapshot: 10000
snapshot_format: HDF5
snapshot_prefix: "examples/cifar10/cifar10_full"
# 使用 CPU 或 GPU 进行运算
solver_mode: GPU
```

本节使用固定学习率的方式设置学习率为 0.001。

（3）训练网络

使用如下命令：

```
./ examples/cifar10/train_full.sh
```

2. 使用 AlexNet 网络训练 CIFAR10 数据集

AlexNet 网络是 2012 年 Imagenet 上的图像分类的冠军，其网络结构在 Caffe 上有详细代码，在此不做赘述。需要注意的是 AlexNet 网络的输入图像大小必须为 227×227 像素，还需要修改网络的训练测试文件路径及网络最后一层的输出神经元个数。

```
#修改网络的训练文件路径
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
 transform_param {
  mirror: true
  crop_size: 32
  mean_file: "examples/cifar10/mean.binaryproto"
```

```
  }
  data_param {
    source: "examples/cifar10/cifar10_train_lmdb"
    batch_size: 256
    backend: LMDB
  }
}

#修改网络的测试文件路径
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
  mirror: false
  crop_size: 32
  mean_file: "examples/cifar10/mean.binaryproto"
 }
 data_param {
  source: "examples/cifar10/cifar10_test_lmdb"
  batch_size: 50
  backend: LMDB
 }
}
 #修改网络最后一层的输出神经元个数
layer {
  name: "fc8"
  type: "InnerProduct"
  bottom: "fc7"
  top: "fc8"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
```

```
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

8.3 实验结果分析

1. 使用 CIFAR10 网络训练 CIFAR10 数据集

CIFAR 10 网络训练中具体参数及准确率结果，如表 8-1 所示。

表 8-1　CIFAR10 网络训练中具体参数及准确率结果

lr_policy	base_lr	momentum	weight_decay	power	gamma	stepsize	max_iter	accuracy
fixed	0.001	0.9	0.004	—	—	—	17000	75.94%
step	0.01	0.9	0.004	0.75	0.1	1000	60000	52.34%

2. 使用 AlexNet 网络训练 CIFAR10 数据集

AlexNet 网络训练中具体参数及准确率结果，如表 8-2 所示。

对比表 8-1 和表 8-2 可知，在 CIFAR10 数据集上，使用 AlexNet 网络结构，base_lr 参数的取值为 0.01，weight_decay 为 0.0005，stepsize 等于 10000 时，使用深度学习训练得到的模型的准确率最高，其准确率为 0.955562。

表 8-2　AlexNet 网络训练中具体参数及准确率结果

lr_policy	base_lr	momentum	weight_decay	power	gamma	stepsize	max_iter	accuracy
step	0.01	0.9	0.0005	0.75	0.1	10000	51000	0.955562
inv	0.01	0.9	0.0005	0.75	0.5	—	—	0.925942
inv	0.01	0.9	0.0005	0.75	0.1	—	—	0.931561

在深度学习的应用中，调参是一个非常重要也极为耗时的工作，同时还需要考虑不同的网络结构：对不同的参数，需要迭代地尝试各种参数取值，并进行实验验证，直到找到最优的参数，以取得最高的识别准确度。

参考文献

[1]　http://www.cs.toronto.edu/~kriz/cifar.html.

第 9 章

基于深度学习的物体图像识别

9.1 数据来源

9.1.1 Caltech101 数据集

Caltech101[1]数据集将物体照片分为 101 类（不包括背景类），每个类别有 40～800 个图片，大多数类别的图片为 50 个左右。每张图片的尺寸约为 300×200 像素。图片格式为 JPG 格式。其样例图片如图 9-1 所示。

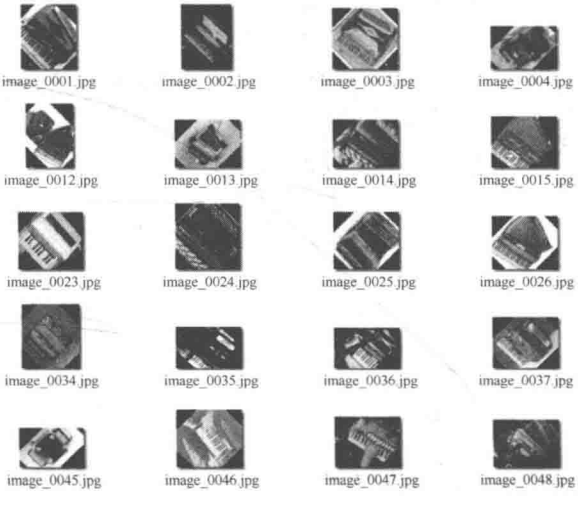


图 9-1 Caltech101 数据集样例图片[1]

9.1.2 Caltech101 数据集处理

同前面实验一样，同样需要先将图片格式转换为数据库格式（LMDB）从而起到提升训练速度的作用。首先将所有图片转移到一个文件夹中，如图 9-2 所示。

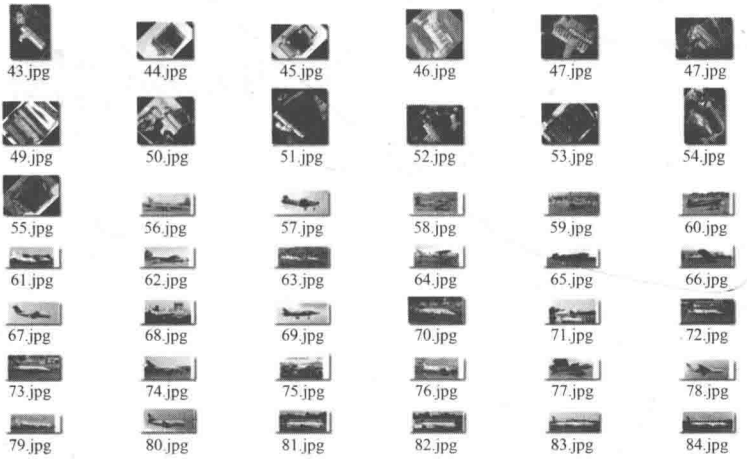


图 9-2 Caltech101 数据集保存到同一个文件夹下的样例图片[1]

建立一个 TXT 文件在文件中按行写入图片名和标签，如图 9-3 所示。

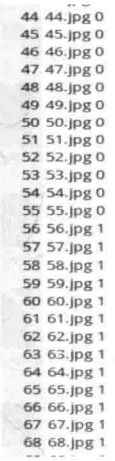


图 9-3 TXT 文件中的图片名和标签

将所有的图片名和标签写入后需要将图片自主分为训练集和测试集，划分的策略是每次选取总图片数的 40%作为测试集（当然也可以按其他策略进行训练）。

9.2　物体图像识别流程

9.2.1　模型介绍

使用 Caffe 在 Caltech101 数据集对网络进行训练，使用的网络模型为 LeNet 模型。其网络模型在第 5 章已经介绍，其中需要修改的文件内容如下：

```
修改训练数据路径
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/O101/img_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
修改测试数据路径
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    scale: 0.00390625
  }
```

```
    data_param {
      source: "examples/O101/img_test_lmdb"
      batch_size: 100
      backend: LMDB
    }
  }
  将输出层数目修改为物体类别数 101
  layer {
    name: "ip2"
    type: "InnerProduct"
    bottom: "ip1"
    top: "ip2"
    param {
      lr_mult: 1
    }
    param {
      lr_mult: 2
    }
    inner_product_param {
      num_output: 101
      weight_filler {
        type: "xavier"
      }
      bias_filler {
        type: "constant"
      }
    }
  }
```

在定义结束网络模型后，将文件命名为 lenet_train_test.prototxt。需要注意的是在训练网络的 source 中需要指定训练图片的 LMDB 文件的路径。同理，在测试网络中的 source 中需要指定测试图片的 LMDB 文件的路径。

9.2.2 操作流程

这里使用 LeNet 和 GoogLeNet 两种不同的网络结构，训练深度学习模型。

1. 使用 LeNet 网络训练 Caltech101 数据集

（1）转换数据集格式

建立 convert_imageset.sh 文件，内容如下：

```
#!/usr/bin/env sh
DATA=examples/O101
rm -rf $DATA/img_train_lmdb
rm -rf $DATA/img_test_lmdb
build/tools/convert_imageset --shuffle \
--resize_height=32 --resize_width=32 \
/home/cz/Downloads/caffe-master/examples/O101/train/ $DATA/ImgSrc.txt  $DATA/img_train_lmdb
build/tools/convert_imageset --shuffle \
--resize_height=32 --resize_width=32 \
/home/cz/Downloads/caffe-master/examples/O101/test/ $DATA/ImgSrc.txt  $DATA/img_test_lmdb
```

最后使用 convert_imageset.sh 将图片文件转换为数据库格式（LMDB）。

格式转换完毕后会在目录下出现两个文件夹：img_train_lmdb，img_test_lmdb 分别存放训练文件和测试文件。

（2）调整网络的参数

Caffe 中网络参数的定义以文本形式存储在 lenet_solver.prototxt 中，其具体内容如下：

```
# 训练和测试网络结构定义文件
net: "examples/O101/lenet_train_test.prototxt"
test_iter: 100
test_interval: 500
# 学习率
base_lr: 0.001
# 动量
momentum: 0.9
# 权重衰减
weight_decay: 0.05
# 学习率策略
lr_policy: "step"
gamma: 0.1
#gamma: 0.0001
power: 0.75
stepsize: 1000
# 迭代 100，显示输出一次
display: 100
# 最大迭代次数
max_iter: 10000
```

```
# 每 5000 次迭代，保存一次中间结构
snapshot: 5000
snapshot_prefix: "examples/O101/lenet"
# 使用 CPU 或 GPU 进行运算
solver_mode: GPU
```

（3）训练网络

使用如下命令：./examples/mnist/train_lenet.sh

该文件内容如下：

```
#!/usr/bin/env sh
./build/tools/caffe train --solver=examples/O101/lenet_solver.prototxt
```

2. 使用 GoogLeNet 网络训练 Caltech101 数据集

本部分内容主要讲解如何使用 GoogLeNet 网络对 Caltech101 数据集进行训练。GoogLeNet 网络是 2014 年 Imagenet 上的图像分类的冠军，其网络结构在 Caffe 上有详细代码，在此不做赘述。需要注意的是 GoogLeNet 网络的输入图像大小必须为 224×224 像素。

在原有网络结构上需要对训练测试的路径及输出神经元数量做对应修改。修改如下：

```
transform_param {
  mirror: true
  crop_size: 224
  mean_value: 104
  mean_value: 117
  mean_value: 123
}
data_param {
  source: "examples/O101/img_train_lmdb"
  batch_size: 32
  backend: LMDB
}
transform_param {
  mirror: false
  crop_size: 224
  mean_value: 104
  mean_value: 117
  mean_value: 123
}
data_param {
  source: "examples/O101/img_test_lmdb"
  batch_size: 50
  backend: LMDB
}
```

```
#修改网络的训练文件路径
layer {
  name: "data"
```

```
    type: "Data"
    top: "data"
    top: "label"
    include {
      phase: TRAIN
    }
    transform_param {
      mirror: true
      crop_size: 224
      mean_value: 104
      mean_value: 117
      mean_value: 123
    }
    data_param {
      source: "examples/O101/img_train_lmdb"
      batch_size: 32
      backend: LMDB
    }
  }
  #修改网络的测试文件路径
  layer {
    name: "data"
    type: "Data"
    top: "data"
    top: "label"
    include {
      phase: TEST
    }
    transform_param {
      mirror: false
      crop_size: 224
      mean_value: 104
      mean_value: 117
      mean_value: 123
    }
    data_param {
      source: "examples/O101/img_test_lmdb"
      batch_size: 50
      backend: LMDB
```

```
    }
  }
  #修改网络的第一个分类器的最后一个全连接层的输出神经元个数
  layer {
    name: "loss1/classifier"
    type: "InnerProduct"
    bottom: "loss1/fc"
    top: "loss1/classifier"
    param {
      lr_mult: 1
      decay_mult: 1
    }
    param {
      lr_mult: 2
      decay_mult: 0
    }
    inner_product_param {
      num_output: 101
      weight_filler {
        type: "xavier"
      }
      bias_filler {
        type: "constant"
        value: 0
      }
    }
  }
  #修改网络的第二个分类器的最后一个全连接层的输出神经元个数
  layer {
    name: "loss2/classifier"
    type: "InnerProduct"
    bottom: "loss2/fc"
    top: "loss2/classifier"
    param {
      lr_mult: 1
      decay_mult: 1
    }
    param {
      lr_mult: 2
```

```
    decay_mult: 0
  }
  inner_product_param {
    num_output: 101
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
#修改网络的第三个分类器的最后一个全连接层的输出神经元个数
layer {
  name: "loss3/classifier"
  type: "InnerProduct"
  bottom: "pool5/7x7_s1"
  top: "loss3/classifier"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 101
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

9.3 实验结果分析

1. 使用 LeNet 网络训练 Caltech101 数据集

运行 train_lenet.sh 后得到准确率为 68.88%。

2. 使用 GoogLeNet 网络训练 Caltech101 数据集

在此网络下对三组参数进行实验。

在 base_lr: 0.01 lr_policy: "step" stepsize: 2500 时其准确率如下所示：

```
Test net output #0: loss1/loss1 = 1.44593 (* 0.3 = 0.433779 loss)
Test net output #1: loss1/top-1 = 0.77188
Test net output #2: loss1/top-5 = 0.929062
Test net output #3: loss2/loss1 = 1.43903 (* 0.3 = 0.43171 loss)
Test net output #4: loss2/top-1 = 0.77016
Test net output #5: loss2/top-5 = 0.929002
Test net output #6: loss3/loss3 = 1.65384 (* 1 = 1.65384 loss)
Test net output #7: loss3/top-1 = 0.731219
Test net output #8: loss3/top-5 = 0.905281
```

其对应的最高准确率为 77.188%，top-5 准确率为 92.9062%。

在 base_lr: 0.02 lr_policy: "step" stepsize: 2500 时其准确率如下所示：

```
Test net output #0: loss1/loss1 = 1.00128 (* 0.3 = 0.300383 loss)
Test net output #1: loss1/top-1 = 0.75166
Test net output #2: loss1/top-5 = 0.922341
Test net output #3: loss2/loss1 = 0.839655 (* 0.3 = 0.251897 loss)
Test net output #4: loss2/top-1 = 0.783821
Test net output #5: loss2/top-5 = 0.927461
Test net output #6: loss3/loss3 = 1.05035 (* 1 = 1.05035 loss)
Test net output #7: loss3/top-1 = 0.7314
Test net output #8: loss3/top-5 = 0.902002
```

其对应的最高准确率为 75.166%，top-5 准确率为 92.2341%。

在 base_lr: 0.03 lr_policy: "step" stepsize: 2500 时其准确率如下所示：

```
Test net output #0: loss1/loss1 = 1.58179 (* 0.3 = 0.474538 loss)
Test net output #1: loss1/top-1 = 0.71782
Test net output #2: loss1/top-5 = 0.907042
Test net output #3: loss2/loss1 = 1.46957 (* 0.3 = 0.440872 loss)
Test net output #4: loss2/top-1 = 0.75494
Test net output #5: loss2/top-5 = 0.930761
Test net output #6: loss3/loss3 = 1.75556 (* 1 = 1.75556 loss)
Test net output #7: loss3/top-1 = 0.722919
Test net output #8: loss3/top-5 = 0.907121
```

其对应最高的准确率为 71.782%，top-5 准确率为 90.7042%。

由于 GoogLeNet 包含三个分类器，所以会出现三组准确率结果。其中 top-5 表示网络测试时一次输出五个标签，如果其中包含正确标签则判断测试准确。

对比 LeNet 和 GoogLeNet 上的实验结果，可知，使用 GoogLeNet，当 base_lr: 0.01 lr_policy: “step” stepsize: 2500 时，通过深度学习训练得到的模型，能够达到最高的识别准确度。

参考文献

[1] Li Fei-Fei, R. Fergus and P. Perona. Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. IEEE. CVPR 2004, Workshop on Generative-Model Based Vision. 2004.

第 10 章

基于深度学习的人脸识别

10.1 数据来源

10.1.1 AT&T Facedatabase 数据库

AT&T Facedatabase 数据库包含 40 个人物，每个人有 10 张图片，其中某些人的图片在不同时间采集，有微弱的光照化和面部表情变化。图片文件格式为 PGM 格式。每个图像大小为 92×112 像素，每像素为 256 灰度级。其中样例图片如图 10-1 所示。

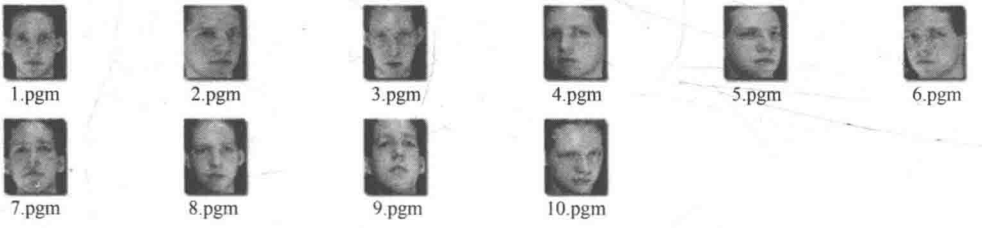


图 10-1 AT&T Facedatabase 数据库中的样例图片[1]

10.1.2 数据库处理

由于Caffe不能直接使用PGM格式图片，所以需要先把图片格式转化为PNG格式。为了使数据能够得到快速处理我们再次把 PNG 格式的图片转化为 LMDB

的数据库格式。在 Caffe 中自带能够将图片格式转化为数据库格式的程序：convert_imageset.sh 其路径为：cafferoot/build/tools/。在相应的文件夹下建立 create_img.sh，其内容如下：

```
#!/usr/bin/env sh
DATA=examples/Face
rm -rf $DATA/img_train_lmdb
rm -rf $DATA/img_test_lmdb
build/tools/convert_imageset --shuffle \
--resize_height=28 --resize_width=28 \
/home/cz/Downloads/caffe-master/examples/Face/face_train/ $DATA/DataSrc/ImgSrc2.txt
$DATA/img_train_lmdb
build/tools/convert_imageset --shuffle \
--resize_height=28 --resize_width=28 \
/home/cz/Downloads/caffe-master/examples/Face/face_test/ $DATA/DataSrc/ImgSrc2.txt
$DATA/img_test_lmdb
```

由上述代码可以看出将图片归一化为 28×28 像素的大小。其中 /home/cz/Downloads/caffe-master/examples/Face/face_train/代表 PNG 文件存放路径。$DATA/img_test_lmdb 和$DATA/img_train_lmdb 即为转换格式后 LMDB 文件的存放地址。

$DATA/DataSrc/ImgSrc2.txt 内容，如图 10-2 所示。

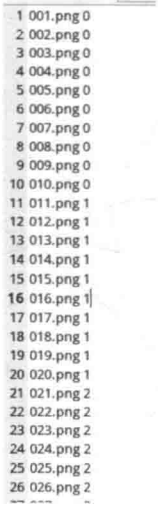


图 10-2　TXT 文件中的图片文件名称和标签

第一列为图片名，第二列为图片标签。

10.2 人脸识别流程

10.2.1 模型介绍

使用 Caffe 在 AT&T Facedatabase 数据集对网络进行训练，使用的网络模型为 LeNet 模型。其网络模型在第 5 章已经介绍，修改文件内容如下：

```
修改训练数据路径
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/Face/img_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
修改测试数据路径
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
```

```
    scale: 0.00390625
  }
  data_param {
    source: "examples/Face/img_test_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
将输出层数目修改为人数 40
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 40
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

在定义结束网络模型后，将文件命名为 lenet_train_test.prototxt。需要注意的是在训练网络的 source 中，需要指定训练图片 LMDB 文件的路径。同理，在测试网络中的 source 中需要指定测试图片 LMDB 文件的路径。

10.2.2　操作流程

本部分将分别使用 LeNet 和 AlexNet 网络结构训练深度学习模型。

1. 使用 LeNet 网络训练 AT&T Facedatabase 数据集

（1）创建 LMDB 数据集

运行 10.1.2 节中创建的文件 create_img.sh，在命令运行完后 examples/Face 路径下会出现两个文件夹：img_test_lmdb（测试数据集）和 img_train_lmdb（训练数据集）。

（2）调整网络的参数

Caffe 中网络参数的定义以文本形式存储在 lenet_solver.prototxt 中，其具体内容如下：

```
# 训练和测试网络定义文件路径
net: "examples/mnist/lenet_train_test.prototxt"
test_iter: 100
#训练迭代 500 次，测试一次
test_interval: 500
#学习率
base_lr: 0.0325
#动量
momentum: 0.9
#权重的衰减
weight_decay: 0.0005
# 学习策略
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# 每 100 次迭代显示一次
display: 100
# 最大迭代次数
max_iter: 10000
# 每 5000 次迭代存储一次数据
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
# 使用 CPU 或 GPU 进行运算
solver_mode: GPU
```

（3）训练网络

使用如下命令：./examples/Face/train_lenet.sh

该文件内容如下：

```
#!/usr/bin/env sh
./build/tools/caffe train --solver=examples/Face/lenet_solver.prototxt
```

2. 使用 AlexNet 网络训练 AT&T Facedatabase 数据集

AlexNet 网络是 2012 年 Imagenet 上的图像分类的冠军，其网络结构在 Caffe 上有详细代码，在此不做赘述。需要注意的是 AlexNet 网络的输入图像大小必须为 227×227 像素。同时 AlexNet 网络需要在输入时减去每张图片的均值。所以还需要计算训练集的均值文件，Caffe 中有命令可以完成此操作，其代码如下：

```
EXAMPLE=examples/Face
DATA=examples/Face
TOOLS=build/tools

$TOOLS/compute_image_mean $EXAMPLE/img_train_lmdb \
 $DATA/mean.binaryproto

echo "Done."
```

生成的均值文件名为 mean.binaryproto。

最后需要修改网络的训练测试文件路径及输出神经元个数，修改如下：

```
#修改网络的训练文件路径
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
 transform_param {
  mirror: true
  crop_size: 227
  mean_file: "examples/Face/mean.binaryproto"
 }
 data_param {
  source: "examples/Face/img_train_lmdb"
  batch_size: 256
  backend: LMDB
 }
}
```

```
#修改网络的测试文件路径
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
  mirror: false
  crop_size: 227
  mean_file: "examples/Face/mean.binaryproto"
 }
 data_param {
  source: "examples/Face/img_test_lmdb"
  batch_size: 50
  backend: LMDB
 }
}
 #修改网络的最后一层的输出神经元个数
layer {
  name: "fc8"
  type: "InnerProduct"
  bottom: "fc7"
  top: "fc8"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 40
    weight_filler {
      type: "gaussian"
      std: 0.01
```

```
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

修改 lenet_solver.prototxt 文件内容，测试不同参数下的准确率。其中一个 lenet_solver.prototxt 的内容如下：

```
net: "examples/Face/AlexNet_train_val.prototxt"
test_iter: 1000
test_interval: 1000
base_lr: 0.01
lr_policy: "inv"
gamma: 0.5
power: 0.75
stepsize: 1000
display: 20
max_iter: 5000
momentum: 0.9
weight_decay: 0.0005
snapshot: 1000
snapshot_prefix: "examples/Face/caffe_alexnet_train"
solver_mode: GPU
```

10.3　实验结果分析

使用 LeNet 网络训练 AT&T Facedatabase 数据集训练结果，如表 10-1 所示。

表 10-1　LeNet 网络训练 AT&T Facedatabase 数据集训练结果

lr_policy	base_lr	Momentum	weight_decay	power	gamma	stepsize	accuracy	max_iter
step	0.01	0.9	0.05	0.75	0.1	1000	0.8959	5000
inv	0.01	0.9	0.0005	0.75	0.0001	—	0.9183	—
inv	0.0001	0.9	0.0005	0.75	0.0001	—	0.9032	—

使用 AlexNet 网络训练 AT&T Facedatabase 数据集训练结果，如表 10-2 所示。

表 10-2　AlexNet 网络训练 AT&T Facedatabase 数据集训练结果

lr_policy	base_lr	momentum	weight_decay	power	gamma	stepsize	accuracy	max_iter
step	0.01	0.9	0.0005	0.75	0.1	100000	0.955382	4000
step	0.01	0.9	0.0005	0.75	0.1	1000	0.903701	5000
inv	0.01	0.9	0.0005	0.75	0.5	—	0.91112	—

对比 LeNet 和 AlexNet 网络结构上的实验结果可知，使用 AlexNet，当 base_lr: 0.01 lr_policy 为 "step"，stepsize: 100000 时，通过深度学习训练得到的模型，能够达到最高的识别准确度为 95.5382%，即为使用深度学习，借助 AlexNet 网络结构，在 AT&T Facedatabase 数据集上，进行人脸识别的最高准确度。

参考文献

[1] Samaria F S, Harter A C. Parameterisation of a Stochastic Model for Human Face Identification[C]// APPlications of Computer Vision, 1994, Proceedings of the Second IEEE Workshop on. IEEE, 1994:138 - 142.

深度学习实践篇
（高级应用）

第 11 章

基于深度学习的人脸识别——DeepID 算法

11.1 问题定义与数据来源

近年来，人脸识别（主要是正脸识别）技术越来越成熟，已经从学术界应用到工业界，与人们的日常生活也越来越紧密。但由于人群、场景、姿态、光照等各种条件的影响，人脸识别仍然需要更多更全面和更深入的研究。

本节主要讲解使用 DeepID 算法[1]解决人脸识别的任务。DeepID 算法是基于深度卷积神经网络的人脸识别算法，该算法由 Y.Sun 等人在 2014 年提出，在 LFW 数据集上，DeepID 算法具有 99%的识别准确度。由于 CelebFaces 和 CelebFaces+ 数据库并未公开，所以本书中的实验是在 CASIA-WebFace[2]数据库上进行训练的，在 LFW[3]图像库上进行实验验证。LFW 图像库包含 10575 个人，共有 494414 张图片，该数据库中每人均有多张图片。本文只基于 Caffe 深度学习平台实现了其中一个 DeepID 深度神经网络，并用于提取人脸的特征。然后，通过计算 LFW 图像库上 6000 对图片特征向量的夹角余弦距离，进行进行人脸识别的验证［计算每对人脸（深度特征之间）的夹角余弦距离，并求解最优的夹角余弦距离阈值］。

11.2　算法原理

本书实现的 DeepID 算法主要包括数据预处理、模型训练策略、算法验证和结果评估。

11.2.1　数据预处理

首先使用 Y.Sun 等人[4]提出的面部检测软件，检测出所使用的图像库中每张图片中的人脸边框信息，根据人脸边框信息检测面部关键点，包括 2 个眼部中点、鼻尖和 2 个嘴角共五个面部关键点。根据这五个关键点首先产生 10 个切片（包括 5 个全局区域和 5 个以关键点为中心的局部区域），如图 11-1（上）所示。每张切片有 3 种尺寸，如图 11-1（下）所示。又分为灰度图和 RGB 图两种，因此每张检测出来的人脸被处理成了对应的 60 个切片。

图 11-1　人脸切片

上部：左边五个为全局区域，右边五个为以检测到的五个关键点（两个眼睛中心，鼻尖和两个嘴角）产生的局部区域；

下部：其中两个切片的三个不同的尺寸。

分别对这 60 种切片（图像库中每个图像上得到的 60 个切片，分别放到这 60 种切片中）训练深度模型，共训练得到 60 个深度网络结构模型（每种切片对

应一种深度模型）。然后，对每张图像的 60 个切片，就可以分别使用这 60 个深度模型对应切片提取深度特征了。这样，每个切片上可以得到一个 160 维的特征（DeepID 特征向量）。然后，对每个切片进行水平翻转，然后使用相同的深度模型提取特征。最后，每张人脸图片对应总的 DeepID 特征向量维度为 19200（160×2×60）。

11.2.2 模型训练策略

LFW 数据库包含了 5749 个人，只有 85 个人有超过 15 张的图片，4069 个人只有一张图片。可以看出 LFW 数据库并不适合训练模型（因为 5749 个人中，只有 1680 个人有两张及以上的图像，故不适合训练机器学习模型）。于是本节采用了 CelebFaces 数据库，随机选取 80%（4349）个人来训练 DeepID 模型，剩余的 20%用于训练人脸验证模型。在深度特征学习的过程中，使用了切片和该切片对应的水平翻转切片，随机选择每个人 10%的图片来作为训练 DeepID 模型的验证数据。

11.2.3 算法验证和结果评估

为了验证本节的算法和评估结果，选择使用了扩展数据库 CelebFaces+，该数据库包含 10177 个人，共有 202599 张图片。随机选取 8700 个人训练 DeepID 模型，剩下的 1477 个人用来训练联合贝叶斯模型做人脸验证。通过使用 5 种不同尺寸将每张图像的切片数量提高到 100 个，每张图像的特征向量维度提高到 32000，然后，使用 PCA 将其降维到 150 维。实验结果表明，本书设计的方法在 LFW 数据库上得到 97.20%的准确率（对 LFW 图像库中的 6000 对图片，分别计算每对人脸图像的特征向量的夹角余弦距离，来进行人脸识别的验证）。考虑到数据分布的影响，最终选择使用迁移学习的算法（在数据集 A 上训练模型，在数据集 B 上测试），并进行 10 折交叉验证（10-fold cross-validation），在 LFW 数据库上得到平均为 97.45%的准确率。

11.3　人脸识别步骤

本节主要介绍人脸识别整体步骤，主要包括数据预处理，训练深度网络结构模型和提取深度特征与人脸验证三个模块。

对应源码整体操作步骤。

（1）运行源码中 process_data/Face_Detect_Master 文件夹下的 process_data_master.m 文件剪切图片；运行源码中 preprocess_data/Face_Detect_Master 文件夹下的 split_data.m 文件将数据划分成训练集和验证集。

（2）运行源码中 extract_feature\deepID_webFace\model 文件夹下的 run_deepIDnet.sh 文件，最终会得到后缀名为 caffemodel 的模型文件。

（3）源码中 extract_feature 文件夹下，getDeepID.m 文件可以实现对指定文件列表中的所有图片进行提取深度特征，classify.m 可以利用提取到的深度特征和夹角余弦距离进行人脸验证。

11.3.1　数据预处理

本部分主要是进行数据预处理，代码保存在源码文件夹 code/preprocess_data 文件夹下。

（1）首先使用 Sun 等人[4]提出的面部检测的方法，检测出所使用的数据库中每张图片中包含的人脸边框信息，如图 11-2 所示。

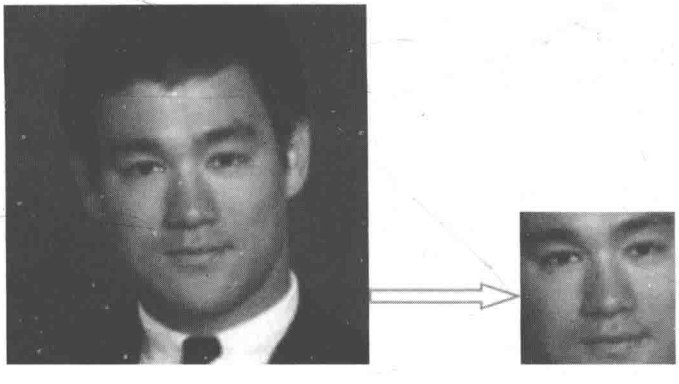


图 11-2　原始图片和裁剪出的面部区域

运行所提供的源码文件 process_data/Face_Detect_Master 工具，在 process_data_master.m 文件中指定原始图片路径、剪切后图片保存路径和图片剪切后尺寸，运行 process_data_master.m，process_data/Face_Detect_Master/Output 文件夹中会生成检测到的图片边框和关键点信息的文件，会在指定的剪切路径下生成对应剪切尺寸的图片。process_data_master.m 文件内容如下：

```
%原始图片路径
root_dir='G:\paper\book\code\face_recognize_deepid\process_data\CASIA-WebFace-images';
%剪切后图片保存路径
crop_dir='G:\paper\book\code\face_recognize_deepid\process_data\CASIA-WebFace-cropped
-images';
image_list='Output/imageList.list';
all_bbox_list='Output/all_bbox.list';
%是否对剪切后的图片进行缩放
flag_scale=false;
%对剪切后的图片缩放的尺寸
crop_size=[48,48];
process_data(root_dir,crop_dir,image_list,all_bbox_list,flag_scale,crop_size);

% 将数据集划分为训练集和验证集

split_data(crop_dir);

% 剪切 LFW 数据库
root_dir='F:\paper\book\code\face_recognize_deepid\extract_feature\deepID_webFace\data\lfw';
crop_dir='F:\paper\book\code\face_recognize_deepid\extract_feature\deepID_webFace\data\lfw_
cropped_images';
image_list='Output/imageList.list';
all_bbox_list='Output/all_bbox.list';
flag_scale=false;
crop_size=[48,48];
process_data(root_dir,crop_dir,image_list,all_bbox_list,flag_scale,crop_size);
%处理没有检测到人脸的图片，使用原始图片代替
miss_bbox_fid=fopen('Output/miss_bbox.list','r+');

while(~feof(miss_bbox_fid))
    line=fgetl(miss_bbox_fid);
    img=imread(fullfile(root_dir,line));
    %得到图片文件名称
```

```
        filesep_index=strfind(line,filesep);
        dot_index=strfind(line,'.');
        image_name=line(max(filesep_index)+1:max(dot_index)-1);
        image_extension=line(max(dot_index)+1:end);
        pre_path=line(1:max(filesep_index)-1);

        %创建保存剪切后图片的路径
        crop_image_dir=strcat(crop_dir,filesep,pre_path);
        if ~exist(crop_image_dir,'dir')
            mkdir(crop_image_dir);
        end

        image_full_path=strcat(crop_image_dir,filesep,image_name,'.',image_extension);
        imwrite(img,image_full_path);
    end

    fclose(miss_bbox_fid);
```

（2）运行源码中 preprocess_data/Face_Detect_Master 文件夹下的 split_data.m，随机选取图片数的 10%作为验证集，剩余图片作为训练集，需要指定图片数据的路径，生成训练集文件 train.txt 和测试集文件 test.txt，文件中每行包含对应的图片路径和标签。split_data.m 文件内容如下：

```
    function split_data(crop_dir)
    crop_dir='G:\paper\book\code\face_recognize_deepid\process_data\CASIA-WebFace-cropped
-images';
    %保存训练集
    train_list_fid=fopen('Output/train.txt','w+');
    %保存验证集
    val_list_fid=fopen('Output/val.txt','w+');

    label=1;
    sub1_crop_dir=dir(crop_dir);
    for i=1:length(sub1_crop_dir)
        sub1_crop_dir_i=sub1_crop_dir(i);
        if(isequal(sub1_crop_dir_i.name,'.')||isequal(sub1_crop_dir_i.name,'..'))
            continue;
        end

        %得到图片路径下的所有图片
```

```
        images=dir(fullfile(crop_dir,sub1_crop_dir_i.name,'*.jpg'));
        images_num=length(images);
    randNum=randperm(images_num);
    %随机选取图片数的 10%作为验证集
        randIndex=randNum(1:floor(images_num*0.1));

        for m=1:images_num
            image=images(m);
            image_path=fullfile(sub1_crop_dir_i.name,image.name);
            image_path=strrep(image_path,'\','/');

            if find(randIndex==m)
                % 将图片路径和标签写入验证集文件
                fprintf(val_list_fid,'%s %d\n',image_path,label);
            else
                % 将图片路径和标签写入训练集文件
                fprintf(train_list_fid,'%s %d\n',image_path,label);
            end
        end

        label=label+1;
    end
    fclose(train_list_fid);
    fclose(val_list_fid);

    end
```

11.3.2 深度网络结构模型

本节使用 DeepID 网络结构，主要包括 4 个卷积层、3 个下采样层（Pooling 层）和 2 个全连接层、线性修正单元（ReLU 层）以及标志所属类别的输出层。网络最后一个全连接层的输出需要设置为类别数，输入图片尺寸：长方形切片尺寸为 $39\times31\times k$，正方形切片尺寸为 $31\times31\times k$，其中 k=3 表示彩色图片，k=1 表示灰度图像。输出属于每个类别的概率值。我们选择网络结构中的倒数第二个全连接层作为要提取的深度特征层（DeepID 层，维度为 160），由于该层后跟有线性修正单元（ReLU 层），所以提取出的深度特征向量都是非负值。使用的网

络结构如图 11-3 所示。

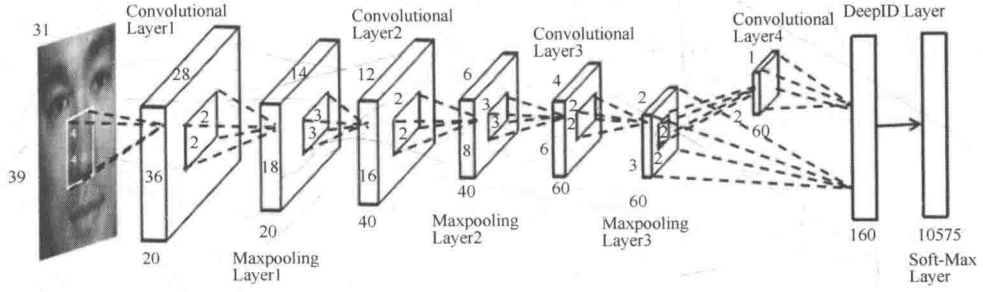


图 11-3　DeepID 网络结构

该网络结构包括 4 个卷积层、3 个下采样层和 2 个全连接层。每一个立方体的维度分别表示输入层、卷积层、下采样层的长度、宽度和高度。内部的小正方形表示卷积核的尺寸。DeepID 层连接了第四个卷积层和第三个下采样层，实现了整体特征和局部特征的结合，称为多尺度网络结构。

本书使用的图片尺寸为 48×48，训练模型所需源码 extract_feature/deepID_webFace/model 文件夹下，所含文件如图 11-4 所示。

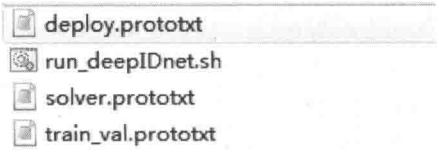


图 11-4　model 文件夹下的文件列表

图 11-4 中，train_val.prototxt 文件定义了 DeepID 网络结构、训练集和验证集，solver.prototxt 文件定义了训练网络所需的网络参数，deploy.prototxt 用于提取深度特征。

在安装有 Caffe 环境的 Ubuntu 系统中，运行 run_deepIDnet.sh 文件，最终会得到后缀名为 caffemodel 的模型文件。run_deepIDnet.sh 文件内容如下：

```
# 创建用于训练深度模型的 lmdb 文件
EXAMPLE=/research/res/face_recognize/code/extract_feature/deepID_webFace/model
#数据文件 train.txt 和 val.txt 所在路径
DATA=/research/res/face_recognize/code/extract_feature/deepID_webFace/data
#caffe 工具所在路径
TOOLS=/stor/caffe/caffe-master/build/tools
#图片所在路径
```

```
    TRAIN_DATA_ROOT=/research/res/face_recognize/code/extract_feature/deepID_webFace/data/
CASIA_WebFace_crop_images/
    VAL_DATA_ROOT=/research/res/face_recognize/code/extract_feature/deepID_webFace/data
/CASIA_WebFace_crop_images/

    # 缩放图片尺寸到 48×48
    RESIZE=true
    if $RESIZE; then
      RESIZE_HEIGHT=48
      RESIZE_WIDTH=48
    else
      RESIZE_HEIGHT=0
      RESIZE_WIDTH=0
    fi

    if [ ! -d "$TRAIN_DATA_ROOT" ]; then
      echo "Error: TRAIN_DATA_ROOT is not a path to a directory: $TRAIN_DATA_ROOT"
      echo "Set the TRAIN_DATA_ROOT variable in create_imagenet.sh to the path" \
           "where the ImageNet training data is stored."
      exit 1
    fi

    if [ ! -d "$VAL_DATA_ROOT" ]; then
      echo "Error: VAL_DATA_ROOT is not a path to a directory: $VAL_DATA_ROOT"
      echo "Set the VAL_DATA_ROOT variable in create_imagenet.sh to the path" \
           "where the ImageNet validation data is stored."
      exit 1
    fi

    echo "Creating train lmdb..."

    GLOG_logtostderr=1 $TOOLS/convert_imageset \
        --resize_height=$RESIZE_HEIGHT \
        --resize_width=$RESIZE_WIDTH \
        --shuffle \
        $TRAIN_DATA_ROOT \
        $DATA/train.txt \
        $EXAMPLE/deepID_webFace_train_lmdb
```

```
echo "Creating val lmdb..."

GLOG_logtostderr=1 $TOOLS/convert_imageset \
    --resize_height=$RESIZE_HEIGHT \
    --resize_width=$RESIZE_WIDTH \
    --shuffle \
    $VAL_DATA_ROOT \
    $DATA/val.txt \
    $EXAMPLE/deepID_webFace_val_lmdb

echo "Done."

# Compute the mean image from the imagenet training lmdb

$TOOLS/compute_image_mean $EXAMPLE/deepID_webFace_train_lmdb \
  $EXAMPLE/deepID_webFace_mean.binaryproto

echo "Done."
#使用 caffe 工具开始训练深度模型
/stor/caffe/caffe-master/build/tools/caffe train \

--solver=/research/res/face_recognize/code/extract_feature/deepID_webFace/model/solver.prototxt
```

说明：源码中涉及的路径问题，请更改为读者自己的对应路径。

11.3.3　提取深度特征与人脸验证

上文已经提到本实验进行人脸验证时，对 LFW 图像库中的 6000 对图片，分别计算每对人脸图像的特征向量的夹角余弦距离，根据该距离的大小判断两张图像是否属同一个人。

本实验选择图 11-3 中的 DeepID 网络结构中的倒数第二个全连接层作为要提取的深度特征层。本实验中，对 LFW 图像集中的每张图片，只使用了一个切片（人脸区域），因此每张图片上最后得到的特征向量维度为 160。我们穷举不同的阈值，分别获得对应的准确率。最后采用最大的准确率对应的阈值，作为本实验的最终结果。

该部分对应的代码包含在源码中 extract_feature 文件夹下，getDeepID.m 文

件可以实现对指定文件列表中的所有图片进行提取深度特征，classify.m 可以利用提取到的深度特征和夹角余弦距离进行人脸验证。

getDeepID.m 文件内容如下：

```
function [ deepID] = getDeepID(imagepath,use_gpu,model_def_file,model_file,featurepath)
%    使用训练得到的模型提取图片列表深度特征

disp('extract DeepID feature Begin');
extractFeature=tic;

% 重置 caffe 网络结构
caffe('reset');
%将图片列表输入网络中，提取得到深度特征，输入参数分别为图片列表文件路径，是否使用 GPU 模式，模型网络文件，模型参数文件
[ori_deepID,list_im,weights]=deepFeature(imagepath,use_gpu,model_def_file,model_file);

% 所有图片的深度特征，转置为每行对应一张图片的深度特征
deepID=ori_deepID';

%保存特征到 csv 文件中
for i=1:length(list_im)
    %获得图片文件的名称
    line=list_im{i,1};
    filesep_index=strfind(line,filesep);
    dot_index=strfind(line,'.');
    image_name=line(max(filesep_index)+1:max(dot_index)-1);
    image_extension=line(max(dot_index)+1:end);
    pre_path=line(1:max(filesep_index)-1);

    if ~exist(feature_path,'dir')
        mkdir(feature_path);
end
%保存特征到对应的文件中
    saveFeatureToCSV(feature_path,image_name,deepID(i,:));
end

disp('extract DeepID feature Done');

toc(extractFeature);
```

```
end
```

classify.m 文件内容如下：

```
function classify(model_def_file,model_file,metric,feature_root,resultFile)
%输入：网络结构文件（deploy.prototxt），训练得到的模型参数文件(deepIDnet_train_iter_450000.caffemodel)，距离方式，保存特征的根目录，保存结果文件 clc;
%将 caffe 路径添加到 matlab 路径中
path('/stor/caffe/caffe-master/matlab/caffe',path);

disp('get the accuracy');

datestr(now,31);
use_gpu=0;
%待提取特征的图片文件列表
left_imagepath='/research/res/face_recognize/code/extract_feature/deepID_webFace/data/left.list';
right_imagepath='/research/res/face_recognize/code/extract_feature/deepID_webFace/data/right.list';
%提取图片列表文件的深度特征
[left_feature]=getDeepID(left_imagepath,use_gpu,model_def_file,model_file,strcat(feature_root,'left/'));
[right_feature]=getDeepID(right_imagepath,use_gpu,model_def_file,model_file,strcat(feature_root,'right/'));
%6000 对人脸对的标签，同一个人标为 1，否则标为 0
label_path='/research/res/face_recognize/code/extract_feature/deepID_webFace/data/label.list';
label_fid=fopen(label_path,'r+');
label_cell=textscan(label_fid,'%d');
label=label_cell{1,1};
fclose(label_fid);

image_num=size(left_feature,1);
%计算 6000 对特征向量之间的距离
distance = pdist2(left_feature,right_feature,metric);

%获得每对之间的距离
predicts=zeros(image_num,1);
for i=1:image_num
    predicts(i,1)=distance(i,i);
end
```

```
%将距离归一化到 0～1，保持跟标签在同一范围
max_predict=max(predicts);
min_predict=min(predicts);

predict_scale=zeros(image_num,1);
for i=1:image_num
    predict_scale(i,1)=(predicts(i,1)-min_predict)/(max_predict-min_predict);
end
%设置初始阈值，如果距离大于该阈值则认为是同一个人，否则不是同一个人
threshold=0.2;
[index,acc,thre]=getaccuracy(predict_scale,label,image_num,threshold,resultFile);

fprintf('index=%d;threshold=%g;the max acc=%g\n',index,thre,acc);
end
```

11.4 实验结果分析

本部分介绍实验结果。我们在 CASIA-WebFace 数据库上训练 DeepID 模型，进行特征学习；在 LFW 数据库上进行实验验证。由于 CASIA-WebFace 数据库每人对应的图片并不相等，因此在该数据库中获得的数据是不均衡的，而且存在图片错误归类的情况，会对实验结果造成一定的影响。

11.4.1 实验数据

（1）我们在 CASIA-WebFace 数据库上进行训练 DeepID 模型，进行特征学习，该数据库包含 10575 个人，共有 494414 张图片，该数据库中每人均有多张图片。图 11-5 显示了 CASIA-WebFace 数据库中的一些样例图片。

（2）我们主要在 LFW 图像库上进行人脸验证，该图像库中一共有 5749 个人物，但 4069 个人物只有一张对应的图片，而只有 85 个人物的图片多于 15 张。图 11-6 显示了 LFW 数据库中的一些样例图片。

图 11-5　CASIA-WebFace 数据库中的样例图片

图 11-6　LFW 数据库中的样例图片

11.4.2　实验结果分析

实验中，数据预处理方法使用 11.3.1 中介绍的方法，生成每个原始图片样本对应的 1 个切片图。使用 11.3.3 中介绍的方法提取深度特征，获得每个原始图片样本对应的 160 维的特征向量。

通过数据预处理得到数据集，随机选取每个人 10%的图片作为训练 DeepID 模型的验证数据，然后使用训练得到的模型提取 LFW 中 6000 对数据的深度特

征，计算 6000 对特征向量的夹角余弦（cosine）距离，通过穷举不同的阈值，得到不同的准确率，选择最大准确率作为最终的验证准确率。最终准确率为 0.823167。得到的准确率并没有达到 90%以上，主要是因为：

（1）CASIA-WebFace 数据库每人对应的图片并不相等，因此在该数据库中获得的数据是不均衡的，而且存在图片错误归类的情况，会对实验结果造成一定的影响。

（2）该实验中没有使用一些数据增强的方法（增加切片数目的方法）。有兴趣的读者，可以尝试筛选 CASIA-WebFace 数据库，去除错误标记的图片和使用一些数据增强技术（切片数目增加的方法，如每个图像产生 60×2 个切片）来提高最终验证准确率。

我们尝试了不同的参数组合，例如逐渐减小学习率、调整批处理大小、测试不同的迭代次数。最终选择较好的参数组合来调优深度学习模型，用于提取深度特征向量。

参考文献

[1] Sun Y, Wang X, Tang X. Deep Learning Face Representation from Predicting 10,000 Classes[C]// 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society, 2014:1891-1898.

[2] Dong Yi, Zhen Lei, Shengcai Liao and Stan Z. Li, "Learning Face Representation from Scratch". arXiv preprint arXiv:1411.7923. 2014.

[3] Huang G B, Mattar M, Berg T, et al. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments[J]. Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition, 2007.

[4] Sun Y, Wang X, Tang X. Deep Convolutional Network Cascade for Facial Point Detection[C]// Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. IEEE, 2013:3476-3483.

第 12 章

基于深度学习的表情识别

本节将详细介绍如何使用深度卷积神经网络进行面部表情识别实验，介绍所使用的深度神经网络结构、提取深度特征的方法以及使用 LIBSVM[1]进行分类的过程。

12.1 表情数据

本书主要使用了扩展的 Cohn-Kanade（CK+）数据库[2]和 JAFFE 数据库[3]。

12.1.1 Cohn-Kanade（CK+）数据库

扩展的 Cohn-Kanade（CK+）数据库中一共出现了 123 个人物，共 593 个表情序列，所有的表情序列都是从无表情状态逐渐向峰值表情变化。每个人物表现出 23 个面部表情序列，这些都是基于 6 种基本的原型表情。由于 CK+数据库中有少部分为彩色图，我们在实验中将这些彩色图转换成了灰度图。图 12-1 显示了 CK+数据库中的样例图片。

图 12-1　CK+数据库中的样例图片

12.1.2　JAFFE 数据库

JAFFE 数据库包括 213 张日本女性面部表情图片。该数据库中图片的尺寸都是 256×256 像素。所有的 213 张图片都用作七类表情识别任务。图 12-2 显示了 JAFFE 数据库中的样例图片。

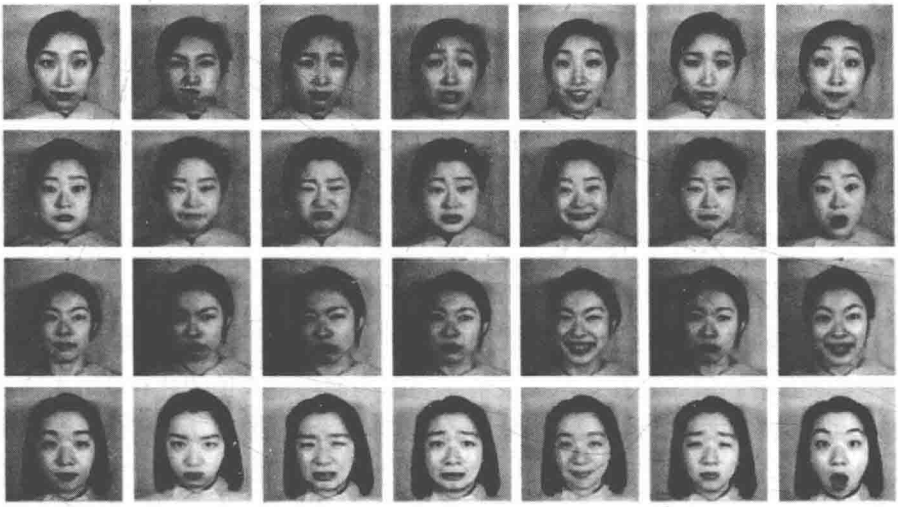


图 12-2　JAFFE 数据库中的样例图片

12.2　算法原理

我们使用 AlexNet 网络结构[4]，它主要包括 5 个卷积层、3 个全连接层、下采样层（Pooling 层）、线性修正单元（ReLU 层）以及标志所属类别的输出层。网络最后一个全连接层的输出需要设置为类别数，输入图片尺寸为 227×227 像素，输出为属于每个类别的概率值，如图 12-3 所示。我们选择网络结构中的倒数第二个全连接层作为要提取的深度特征层，由于该层后跟有线性修正单元（ReLU 层），所以提取出的深度特征向量都是非负值。为防止过拟合现象，网络中使用了 Drop-Out 层，过滤学习到的一些冗余信息。

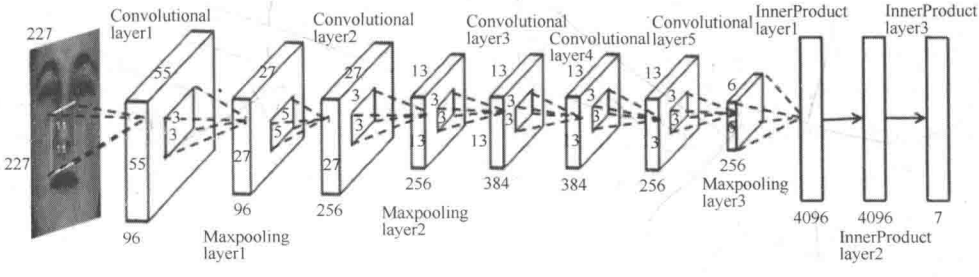


图 12-3　AlexNet 网络结构

每个立方体的长、宽和高表示特征图的数目和维度，在立方体中的正方形表示核尺寸，最后三层为全连接层，全连接层旁边的数字分别代表该层的神经元数目。

首先使用 Sun 等人[5]提出的面部检测的方法，检测出所使用的数据库中每张图片中的人脸边框信息。根据人脸边框信息检测面部关键点，包括 2 个眼部中点、鼻尖和 2 个嘴角共五个面部关键点。根据这五个关键点首先产生 10 个切片（包括 5 个全局区域和 5 个以关键点为中心的局部区域），如图 12-1 所示，基于每个切片产生 3 个尺寸大小切片，最后对产生的切片进行左右翻转处理，最后每张图片产生对应的 60 个切片。

选择网络结构中的倒数第二个全连接层作为要提取的深度特征层，因此对于每个切片，都能得到一个 4096 维的特征向量，由于每张图片对应 60 个切片，因此每张图片可以获得的特征向量维度为 245760（4096×60）。

12.3 表情识别步骤

本节主要介绍面部表情识别整体步骤，主要包括数据预处理、训练深度网络结构模型和提取深度特征与分类三个模块。

对应源码的整体操作步骤如下。

（1）运行源码中 process_data/Face_Detect_Master 文件夹下的 process_data_master.m 文件剪切图片；运行源码中 process_data/Face_Detect_Master 文件夹下的 generate10GroupSubjects.m 文件将数据划分为训练图像集和验证图像集。

（2）运行源码中 extract_feature\model 文件夹下的 train_expressionIDnet_imagenet_file.sh 文件，最终会得到后缀名为 caffemodel 的模型文件。

（3）源码中 extract_feature 文件夹下，getDeepID.m 文件可以实现对指定文件列表中的所有图片进行提取深度特征，crossval_ck_subjects.m 可以利用提取到的深度特征和夹角余弦距离进行面部表情识别。

12.3.1 数据预处理

（1）首先使用 Sun 等人[5]提出的面部检测的方法，检测出所使用的数据库中每张图片中包含的人脸边框信息，如图 12-4 所示。

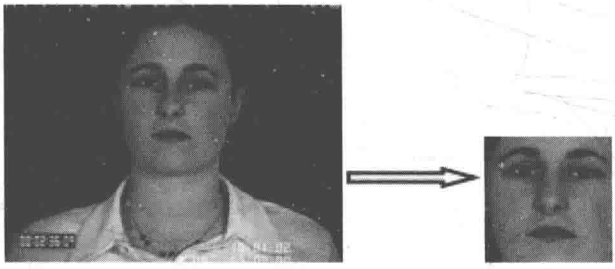


图 12-4 原始图片和裁剪出的面部区域

（2）根据人脸边框信息检测面部关键点，包括 2 个眼部中点、鼻尖和 2 个嘴角共五个面部关键点。根据这五个关键点首先产生 10 个切片（包括 5 个全局区域和 5 个以关键点为中心的局部区域），如图 12-1 所示，基于每个切片产生 3 个尺寸大小切片，最后对产生的切片进行左右翻转处理，最后每张图片产生对应的

60 个切片。

运行所提供的源码文件 process_data/Face_Detect_Master 工具，在 process_data_master.m 文件中指定原始图片路径、剪切后图片保存路径和图片剪切后尺寸，process_data/Face_Detect_Master/Output 文件夹中会生成检测到的图片边框和关键点信息的文件，会在指定的剪切路径下生成对应剪切尺寸的图片。将每张图片处理成对应的 120 个切片。process_data_master.m 文件内容如下：

```
%原始图片路径
root_dir='G:\paper\book\code\facial_expression\process_data\jaffe_image';
%剪切后图片保存路径
crop_dir='G:\paper\book\code\facial_expression\process_data\jaffe_cropped_image';
image_list='Output/imageList.list';
all_bbox_list='Output/all_bbox.list';
%是否对剪切后的图片进行缩放
flag_scale=false;
%对剪切后的图片缩放的尺寸
crop_size=[48,48];
process_data(root_dir,crop_dir,image_list,all_bbox_list,flag_scale,crop_size);
```

（3）运行源码中 process_data/Face_Detect_Master 文件夹下的 generate10 GroupSubjects.m，随机将数据集分成 10 组，每组作 1 次验证集，共进行 10 轮生成数据集文件的操作。针对每轮，选 1 组作为验证集，从其余组中随机选 1 组作为测试集，其余组作为训练集。生成的文件中每行包含对应的图片路径和标签。

12.3.2　深度神经网络结构模型

本节使用的 AlexNet 网络结构[4]，主要包括 5 个卷积层、3 个全连接层、下采样层（Pooling 层）、线性修正单元（ReLU 层）以及标志所属类别的输出层。网络最后一个全连接层的输出需要设置为类别数，输入图片尺寸为 227×227 像素，输出为属于每个类别的概率值。我们选择网络结构中的倒数第二个全连接层作为要提取的深度特征层，由于该层后跟有线性修正单元（ReLU层），所以提取出的深度特征向量都是非负值。为了防止过拟合现象，网络中使用了 Drop-Out 层，过滤学习到的一些冗余信息。

训练模型所需源码在 extract_feature/model 文件夹下，所含文件如图 12-5 所示。

图 12.5　训练深度模型所用到的相关配置文件与模型

图 12.5 中，train_val_imagenet_file.prototxt 文件定义了 ImageNet 网络结构、训练集和验证集，solver_imagenet_file.prototxt 文件定义了训练网络所需的网络参数，deploy_imagenet.prototxt 文件用于提取深度特征，bvlc_reference_caffenet.caffemodel 是参考文献[4]中训练出来的深度模型。

在安装有 Caffe 环境的 Ubuntu 系统中，运行 train_expressionIDnet_imagenet_file.sh 文件，参考文献[4]中训练的 bvlc_reference_caffenet.caffemodel 深度模型进行初始化网络参数，最终得到后缀名为 caffemodel 的模型文件。train_expressionIDnet_imagenet_file.sh 文件内容如下：

```
#使用 caffe 工具开始训练深度模型
/stor/caffe/caffe-master/build/tools/caffe train \
--solver=/wpf/face_expression/paperfinal/sourcecode/ckplus/8class_imagenet/model/crossval1
/solver_imagenet_file.prototxt
--weights=/stor/caffe/caffe-master/models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemo
del
```

说明：源码中涉及的路径问题，请根据实际情况进行更改。

12.3.3　提取深度特征及分类

本节选择网络结构中的倒数第二个全连接层作为要提取的深度特征层，因此对于每个切片，都能得到一个 4096 维的特征向量，由于每张图片对应 60 个切片，因此每张图片可以获得的特征向量维度为 245760（4096×60）。提取到的特征如图 12-6 所示。

左边为切片，右边为对应的特征图，对应的为网络结构中倒数第二层的特征显示，维度为 64×64，由于在深度特征层后边加了线性修正单元（ReLU 层），因此特征向量值都是非负的，特征图中亮度越大，表明该处代表的特征值越大。可以看出，对于同一个人的不同表情的特征具有很强的可区分性。

该部分对应的代码包含在源码中 extract_feature 文件夹下，getDeepID.m 文

件可以实现对指定文件列表中的所有图片进行提取深度特征，crossval_ck_subjects.m 可以利用提取到的深度特征和 LIBSVM 进行 10 折交叉验证。

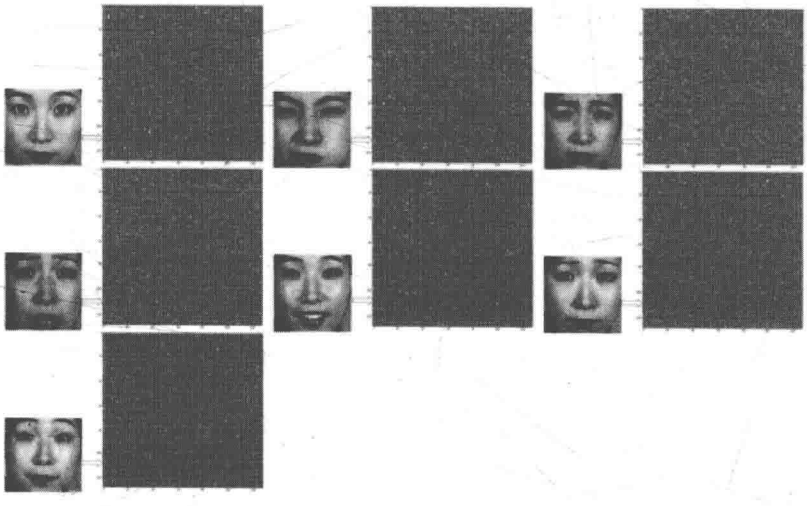


图 12-6　切片和提取到的对应特征图

getDeepID.m 文件的内容如下：

```
function [ deepID,label ] = getDeepID( imagepath,use_gpu,model_def_file,model_file )
%使用训练的模型提取文件列表图片的特征

disp('extract DeepID feature Begin');
extractFeature=tic;

%重置 caffe 网络
caffe('reset');
%将图片列表输入网络中，提取得到深度特征，输入参数分别为图片列表文件路径，是否使用 GPU 模式，模型网络文件，模型参数文件
[ori_deepID,list_im,weights,imageLabel]=deepFeature(imagepath,use_gpu,model_def_file,model_file);

%获得所有图片对应的特征向量
deepID=reshape(ori_deepID,4096*60,[]);
%所有图片的深度特征，转置为每行对应一张图片的深度特征
deepID=deepID';
deepID=double(deepID);
% 获得所有图片对应的标签
```

```
labelRows=size(imageLabel,1);
label=zeros(labelRows/60,1);
j=1;
for i=1:60:labelRows
    label(j,1)=imageLabel(i,1);
    j=j+1;
end

disp('extract expressionID feature Done');
toc(extractFeature);
end
```

12.4 实验结果分析

本节将介绍作者所做的实验以及实验结果对比分析。主要考虑六类［生气（Anger）、厌恶（Disgust）、恐惧（Fear）、喜悦（Happy）、悲伤（Sadness）、惊讶（Surprise）］识别和包括面部无表情（Neutral）在内的七类识别任务。

本书主要测试了扩展的 Cohn-Kanade（CK+）数据库[2]和一个跨库测试七类表情识别的实验（在 CK+数据库上训练，在 JAFFE 数据库[3]上测试），验证模型的泛化能力（迁移学习能力）。

12.4.1 实现细节

本实验中，数据增强方法使用 12.3.1 中介绍的方法，生成每个原始图片样本对应的 60 个切片图。使用 12.3.3 中介绍的方法提取深度特征，获得每个原始图片样本对应的 245760 维的特征向量。

（1）我们从 CK+数据库中选择了 309 个表情序列，选择标准是这个表情序列属于六种基本原型表情。对于每个表情序列，选择无表情图片和前三的峰值表情图片作为表情识别样本，因此总共有 1236（309×4）个原始图片样本。使用 10-fold（十折）交叉验证模式来测试泛化能力。按实验中的人物随机分成十组，其中 1 组作为测试集，剩余 9 组作为训练集和验证集，直接使用 Caffe 类库、12.3.2

中的网络结构和预训练模型 ImageNet 模型对训练集和验证集进行调优训练，使用 12.3.3 中提取特征的方法和调优训练得到的模型对训练集、验证集和测试集分别提取深度特征，训练集和验证集的深度特征向量及标签作为训练数据，测试集的深度特征向量及标签作为测试数据，最终在 LIBSVM 训练分类模型，预测测试集上图像的表情，最终得到本轮的识别率。分别将每一组作为一次测试集测试 10 次，取测试识别率的平均值作为最终结果。

由于使用了 10 折交叉验证分别训练模型，所以在训练过程中使用了不同的迭代次数、不同的学习率和批处理大小（batch size），针对每一轮，我们尝试了不同的参数组合，例如逐渐减小学习率、调整批处理大小、测试不同的迭代次数。最终选择较好的参数组合来调优训练模型，进而提取深度特征向量。

（2）将 CK+数据库按实验者随机分成 10 组，随机取 1 组作为验证集，其余 9 组作为训练集，将 JAFFE 数据库作为测试集，直接使用 Caffe 类库、12.3.2 中的网络结构和预训练模型 ImageNet 模型对训练集和验证集进行调优训练，使用 12.3.3 中提取特征的方法和调优训练得到的模型对训练集、验证集和测试集分别提取深度特征，训练集和验证集的深度特征向量及标签作为训练数据，测试集的深度特征向量及标签作为测试数据，最终在 LIBSVM 得到识别率。

12.4.2　实验结果对比

本部分我们主要对我们的方法得到的结果和目前比较好的表情识别结果进行对比分析。

（1）在 CK+数据库上，6 类整体识别结果为 98.2904%，7 类整体识别结果为 96.24%，如表 12-1 所示。6 类表情识别结果的混淆矩阵如表 12-2 所示，7 类表情识别结果的混淆矩阵如表 3 所示。从表 12-1 中可以看出我们的识别结果已经超过了其他方法，在 6 类表情识别上相比其他方法最好的结果提高了 3.35%，在 7 类表情识别上相比其他方法最好的结果提高了 5.295%。从表 12-2 中可以看出 Fear 比较容易被识别为其他表情，而其他表情都能达到 96%以上。从表 12-3 中可以看出 Fear 和 Sadness 比较容易被识别为其他表情，而其他表情都能达到 96%以上。

表 12-1　CK+数据库表情识别结果对比

	六类 6-class(%)	七类 7-class(%)
Our Method	98.2904	96.24
SVM(linear)+Boosted-LBP [6]	95.0	91.1
SVM(polynomial)+Boosted-LBP [6]	95.0	91.1
SVM(RBF)+Boosted-LBP [6]	95.1	91.4
LDA+Boosted-LBP [6]	84.2	77.6
SVM(linear)+LBP [6]	91.5	88.1
SVM(polynomial)+LBP [6]	91.5	88.1
SVM(RBF)+LBP [6]	92.6	88.9
LDA+LBP [6]	79.2	73.4
SVM(linear)+Gabor [6]	89.4	86.6
SVM(polynomial)+Gabor [6]	89.4	86.6
SVM(RBF)+Gabor [6]	89.8	86.8
SVM(linear)+Gabor [7]	—	84.8
SVM(polynomial)+Gabor [7]	—	Worse than RBF/linear
SVM(RBF)+Gabor [7]	—	86.9
Geometric features+TAN [8]	73.2	—

注："—"表示未做实验，没有相应结果。

表 12-2　6 类表情识别结果的混淆矩阵

	方法	生气 (%)	厌恶 (%)	恐惧 (%)	喜悦 (%)	悲伤 (%)	惊讶 (%)
生气	我们的方法	98.5	0.8	0	0	0.7	0
	SVM(RBF)+LBP [6]	89.7	2.7	0	0	7.6	0
厌恶	我们的方法	0	99.4	0.6	0	0	0
	SVM(RBF)+LBP [6]	0	97.5	2.5	0	0	0
恐惧	我们的方法	0	0.7	89.3	10.0	0	0
	SVM(RBF)+LBP [6]	0	2.0	73.0	22.0	3.0	0
喜悦	我们的方法	0	0	0	100.0	0	0
	SVM(RBF)+LBP [6]	0	0.4	0.7	97.9	1.0	0
悲伤	我们的方法	0	0	3.3	0	96.7	0
	SVM(RBF)+LBP [6]	10.3	0	0.8	0.8	83.5	4.6
惊讶	我们的方法	0	0	0	1.0	0	99.0
	SVM(RBF)+LBP [6]	0	0	1.3	0	0	98.7

表 12-3　7 类表情识别结果的混淆矩阵

	方法	无表情 (%)	生气 (%)	厌恶 (%)	恐惧 (%)	喜悦 (%)	悲伤 (%)	惊讶 (%)
无表情	我们的方法	99.3	0.4	0.3	0	0	0	0
	SVM(RBF)+LBP [6]	90.0	1.6	0.4	0	1.6	6.0	0.4
	SVM(RBF)+Boosted-LBP [6]	95.2	0	0	0.8	0.4	3.6	0
生气	我们的方法	2.0	98.0	0	0	0	0	0
	SVM(RBF)+LBP [6]	7.5	85.0	2.7	0	0	4.8	0
	SVM(RBF)+Boosted-LBP [6]	20.4	66.6	3.7	2.0	0	7.3	0
厌恶	我们的方法	3.0	0.6	96.4	0	0	0	0
	SVM(RBF)+LBP [6]	0	0	97.5	2.5	0	0	0
	SVM(RBF)+Boosted-LBP [6]	5.0	0	92.5	2.5	0	0	0
恐惧	Our Method	0	0	0	81.7	10.0	0	8.3
	SVM(RBF)+LBP [6]	7.0	0	2.0	68.0	22.0	1.0	0
	SVM(RBF)+Boosted-LBP [6]	10.0	0	0	70.0	17.0	3.0	0
喜悦	我们的方法	0	0	0	0	100.0	0	0
	SVM(RBF)+LBP [6]	3.5	0	0	0.7	94.7	1.1	0
	SVM(RBF)+Boosted-LBP [6]	7.4	0	0	2.5	90.1	0	0
悲伤	我们的方法	18.3	5.0	0	0	0	76.7	0
	SVM(RBF)+LBP [6]	19.6	8.6	0	0	0	69.5	2.3
	SVM(RBF)+Boosted-LBP [6]	31.6	6.4	0	0	0	61.2	0.8
惊讶	我们的方法	1.2	0	0	0	1.0	0	97.8
	SVM(RBF)+LBP [6]	0.5	0	0	1.3	0	0	98.2
	SVM(RBF)+Boosted-LBP [6]	5.7	0	0	1.3	0	0.5	92.5

（2）在 CK+数据库上，如果仅使用一张人脸全局切片进行实验，6 类整体识别结果为 92.246%，7 类整体识别结果为 89.6778%。每张图片使用 60 张切片后，6 类整体识别结果提高至 98.2904%，7 类整体识别结果提高至 96.24%。从而可以看出，进行适当的数据增强对提高准确率有一定的帮助作用。

（3）我们的方法在 JAFFE 数据库测试的最终准确率为 49.77%。如表 12-4 所示。

该结果有待于进一步提高，但高于参考文献[9]中的实验结果，显示了深度学习进行表情识别的实际效果。

表 12-4 JAFFE 数据库表情泛化结果对比

	七类(%)
我们的方法	49.77
SVM(linear)+Boosted-LBP[9]	40.4
SVM(polynomial)+Boosted-LBP[9]	40.4
SVM(RBF)+Boosted-LBP[9]	41.3

参考文献

[1] LibSVM. https://www.csie.ntu.edu.tw/~cjlin/libsvm/.

[2] Lucey, P., Cohn, J. F., Kanade, T., Saragih, J., Ambadar, Z., & Matthews, I. (2010). The Extended Cohn-Kanade Dataset (CK+): A complete expression dataset for action unit and emotion-specified expression. Proceedings of the Third International Workshop on CVPR for Human Communicative Behavior Analysis (CVPR4HB 2010), San Francisco, USA, 94-101.

[3] M.J. Lyons, J. Budynek, S. Akamatsu, Automatic classification of single facial images, IEEE Transactions on Pattern Analysis and Machine Intelligence 21 (12) (1999) 1357–1362.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton.Imagenet classification with deep convolutional neural networks. In Proceedings of NIPS, 2012.

[5] Sun Y, Wang X, Tang X. Deep Convolutional Network Cascade for Facial Point Detection[C]// Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. IEEE, 2013:3476-3483.

[6] Shan C, Gong S, Mcowan P W. Facial expression recognition based on Local Binary Patterns: A comprehensive study[J]. Image & Vision Computing, 2009, 27(6):803-816.

[7] M. Bartlett, G. Littlewort, I. Fasel, R. Movellan, Real time face detection and facial expression recognition: development and application to human–computer interaction, in: CVPR Workshop on CVPR for HCI, 2003.

[8] I. Cohen, N. Sebe, A. Garg, L. Chen, T.S. Huang, Facial expression recognition from video sequences: temporal and static modeling, Computer Vision and Image Understanding 91 (2003) 160-187.

[9] Shan C, Gong S, Mcowan P W. Facial expression recognition based on Local Binary Patterns: A comprehensive study[J]. Image & Vision Computing, 2009, 27(6):803-816.

第 13 章

基于深度学习的年龄估计

13.1 问题定义

年龄估计问题通常可以看做多分类问题或者回归问题。该问题具有非常广泛和新颖的应用，如著名的年龄估计网站 http://how-old.net/。本书中将年龄估计问题作为回归问题进行测试，并使用文献[1]中提供的数据集进行实验验证。

13.2 年龄估计算法

本节主要介绍年龄估计整体步骤：主要包括数据预处理、提取深度特征及 LBP 特征和训练回归模型三个模块。

13.2.1 数据预处理

使用文献[2]中的人脸检测工具进行数据预处理。运行源码 process_data\face-release1.0-basic[2]文件夹下的 process_data.m 文件，需要指定图片目录，剪切后图片保存目录和剪切后图片路径文件的保存目录三个参数。

process_data.m[2]文件内容如下：

```
function [ output_args ] = preprocess_data(datadir,crop_dir,filepath)
compile;
load face_p146_small.mat %加载检测所用模型
model.interval = 5;
model.thresh = min(-0.65, model.thresh);%设置阈值

if length(model.components)==13
    posemap = 90:-15:-90;
elseif length(model.components)==18
    posemap = [90:-15:15 0 0 0 0 0 0 -15:-15:-90];
else
    error('Can not recognize this model');
end
ims = dir(strcat(datadir,'*.jpg'));
non_fid=fopen(strcat(filepath,'non_bbox.list'),'a+t');
fid=fopen(strcat(filepath,'bbox.list'),'a+t');
crop_fid=fopen(strcat(filepath,'crop_file.list'),'a+t');
for i = 1:length(ims)
    fprintf('testing: %d/%d\n', i, length(ims));
    imageName=ims(i).name;
    picFilePath=[datadir imageName];
    cropFilePath=[crop_dir imageName];
    im = imread([datadir imageName]);
    imheight=size(im,1);
    imwidth=size(im,2);
    scale=1;
    scale_im=im;
    if(imwidth>1280)%对图像进行适当缩放，以便能更好地检测
        scale=1280/imwidth;
        scale_im=imresize(im,[imheight*scale,imwidth*scale]);
    end

    if(numel(size(scale_im))<3)%将灰度图转为 RGB 图像
        scale_im=cat(3,scale_im,scale_im,scale_im);
    end

    tic;
    bs = detect(scale_im, model, model.thresh);%使用模型参数检测输入图像
    bs = clipboxes(scale_im, bs);%防止边框越界
```

```
        bs = nms_face(bs,0.3);%对检测到的边框使用阈值 0.3 进行非最大值抑制处理
        dettime = toc;

        if(~isempty(bs))
            b=bs(1);
            partsize = fix((b.xy(1,3)-b.xy(1,1)+1)/2);

            x1=(min(b.xy(:,1))+partsize)/scale;%左边界
            y1=(min(b.xy(:,2))+partsize)/scale;%上边界
            x2=(max(b.xy(:,3))-partsize)/scale;%右边界
            y2=(max(b.xy(:,4))-partsize)/scale;%下边界

            boundingBox=[' ' num2str(x1) ' ' num2str(x2) ' ' num2str(y1) ' ' num2str(y2)];
            disp(boundingBox);
            fprintf(fid,'%s%s\n',picFilePath,boundingBox);%将边框信息保存到文件中

            bbox=[x1,x2,y1,y2];
            crop_images(cropFilePath,im,bbox);%根据检测到的边框信息剪切出人脸图片
            fprintf(crop_fid,'%s\n',cropFilePath);
            savekeyPoints(imageName,b,scale);%保存面部关键点信息
        else
            fprintf(non_fid,'%s\n',picFilePath);
        end
        fprintf('Detection took %.1f seconds\n',dettime);
        close all;
    end
    fclose(fid);
    fclose(non_fid);
    fclose(crop_fid);
    disp('done!');
    end
```

13.2.2 提取深度特征

使用文献[3]中的预训练模型提取图片深度特征。源代码 extract_feature\deepeval-encoder-1.0.3 来自文献 [3]，extractFeature.m 是本书在文献[4]的基础上进行了修改。

由于使用了多个不同的深度模型提取特征，这里介绍其中一种使用 CNN_F 模型提取深度特征的方法。运行源码中 extract_feature\deepeval-encoder-1.0.3[3] 文件夹下的 extractFeature.m 文件。

extractFeature.m 文件内容如下：

```
clear;

fid=fopen('/wpf/face_age/age/code/process_data/crop_data/crop_data_list.txt');
f=textscan(fid,'%s');
rows=f{1};
model_dir = './models/CNN_M_128';%模型文件
fprintf('%s\n', model_dir);
param_file = sprintf('%s/param.prototxt', model_dir);
model_file = sprintf('%s/model', model_dir);

average_image = './models/mean.mat';

use_gpu = false;%是否使用 GPU 模式
if use_gpu
    featpipem.directencode.ConvNetEncoder.set_backend('cuda');
end

% 初始化 encoder 实例
encoder = featpipem.directencode.ConvNetEncoder(param_file, model_file, ...
    average_image, ...
    'output_blob_name', 'fc7');

% 使用 CNN_M_1024 模型
model_dir_2 = './models/CNN_M_1024';

param_file_2 = sprintf('%s/param.prototxt', model_dir_2);
model_file_2 = sprintf('%s/model', model_dir_2);
encoder_2 = featpipem.directencode.ConvNetEncoder(param_file_2, model_file_2, ...
    average_image); % 返回 fc7 层向量

% 使用 CNN_M_2048 模型
model_dir_3 = './models/CNN_M_2048';

param_file_3 = sprintf('%s/param.prototxt', model_dir_3);
```

```
model_file_3 = sprintf('%s/model', model_dir_3);

encoder_3 = featpipem.directencode.ConvNetEncoder(param_file_3, model_file_3, ...
    average_image);%返回 fc7 层向量

% 使用 CNN_M 模型
model_dir_4 = './models/CNN_M';

param_file_4 = sprintf('%s/param.prototxt', model_dir_4);
model_file_4 = sprintf('%s/model', model_dir_4);

encoder_4 = featpipem.directencode.ConvNetEncoder(param_file_4, model_file_4, ...
    average_image); % 返回 fc7 层向量

% 使用 CNN_F 模型
model_dir_5 = './models/CNN_F';

param_file_5 = sprintf('%s/param.prototxt', model_dir_5);
model_file_5 = sprintf('%s/model', model_dir_5);

encoder_5 = featpipem.directencode.ConvNetEncoder(param_file_5, model_file_5, ...
    average_image); %返回 fc7 层向量
% 使用 CNN_S 模型
model_dir_6 = './models/CNN_S';

param_file_6 = sprintf('%s/param.prototxt', model_dir_6);
model_file_6 = sprintf('%s/model', model_dir_6);

encoder_6 = featpipem.directencode.ConvNetEncoder(param_file_6, model_file_6, ...
    average_image);% 返回 fc7 层向量

for i=1:length(rows)
    sample_image_path = rows{i};
    namestr=strsplit(sample_image_path,{'/','.'},'CollapseDelimiters',true);
    imageName=namestr{length(namestr)-1};

    im = imread(sample_image_path);
    im = featpipem.utility.standardizeImage(im); % 确保图片值为单精度，三通道
```

```
        tic;
        code = encoder.encode(im);%计算图片特征
        toc;

saveFeatureToCSV('/home/wpf/res/age/deepeval-encoder-1.0.3/featureFile/Train/CNN_M_128/',image
Name,code);%将图片特征保存到指定文件中

        tic;
        code_2 = encoder_2.encode(im);
        toc;

saveFeatureToCSV('/home/wpf/res/age/deepeval-encoder-1.0.3/featureFile/Train/CNN_M_1024/',
imageName,code_2);%将图片特征保存到指定文件中

        tic;
        code_3 = encoder_3.encode(im);
        toc;

saveFeatureToCSV('/home/wpf/res/age/deepeval-encoder-1.0.3/featureFile/Train/CNN_M_2048/',
imageName,code_3);%将图片特征保存到指定文件中

        tic;
        code_4 = encoder_4.encode(im);
        toc;
    saveFeatureToCSV('/home/wpf/res/age/deepeval-encoder-1.0.3/featureFile/Train/CNN_M/',
imageName,code_4);%将图片特征保存到指定文件中

        tic;
        code_5 = encoder_5.encode(im);
        toc;

saveFeatureToCSV('/home/wpf/res/age/deepeval-encoder-1.0.3/featureFile/Train/CNN_F/',image
Name,code_5);%将图片特征保存到指定文件中

        tic;
        code_6 = encoder_6.encode(im);
        toc;

saveFeatureToCSV('/home/wpf/res/age/deepeval-encoder-1.0.3/featureFile/Train/CNN_S/',image
```

```
Name,code_6);%将图片特征保存到指定文件中
    end
```

13.2.3 提取 LBP 特征

运行源码中 process_data\face-release1.0-basic 文件夹下的 extractLBP.m 文件。

extractLBP.m 文件内容如下：

```
function extractLBP(im,filename)
%将图片分成 4*4 的网格
divX=4;
divY=4;
%将 RGB 图像转成灰度图
if ndims(im)==3
    grayim=rgb2gray(im);
end

lbp=[];
[M,N] = size(grayim);
blockSizeM = floor(M/divX);
blockSizeN = floor(N/divY);
for i=1:divX,
for j=1:divY,
  %分别读取图片各个网格

blockImg=grayim((i-1)*blockSizeM+1:i*blockSizeM,(j-1)*blockSizeN+1:j*blockSizeN);
        %合并各个网格的 LBP 特征
        lbp=[lbp,extractLBPFeatures(blockImg)];
      end
end
%保存提取到的 LBP 特征
save (filename ,'lbp' );
end
```

13.2.4 训练回归模型

使用 LIBSVM 训练 Support Vector Regression(SVR)回归模型。运行源码中

classify_result\libsvm-3-0-2\trainval_deepeval 文件夹下的 trainAgeModelTune_CV_CNN_F.m 文件。需要修改特征保存的路径和标签文件的保存路径。

trainAgeModelTune_CV_CNN_F.m 文件内容如下：

```
clear;
close all;
clc;
logID=fopen('cnn_f.txt','a+');
fprintf(logID,'==========all
fprintf(logID,'load data\n');
dataroot='/home/wpf/res/age/deepeval-encoder-1.0.3/featureFile/Train/CNN_F/';
labelroot='/home/wpf/res/age/deepeval-encoder-1.0.3/featureFile/Train/';
[feature,label,standard_deviations] = loadAgeData_CNN(dataroot,labelroot);
fprintf(logID,'split data\n');
rows=size(feature,1);

error=zeros(0,0);
mae=zeros(0,0);
mse=zeros(0,0);

for i=1:5
    % 将数据划分为 80%作为训练集，剩余的作为验证集
    rowsIndex=randperm(rows);
    trainvalNums=ceil(rows*0.8);
    trainval_Index=rowsIndex(1:trainvalNums);

    trainval_label=label(trainval_Index);
    trainval_instance=feature(trainval_Index,:);
    trainval_standard_deviations=standard_deviations(trainval_Index);

    % 利用交叉验证和网格搜索寻找最优参数组合
    disp('begin search the bestc bestg');
    [bestc,bestg,min_mse]=
LibSvmCgForRegression_CNN(trainval_label,trainval_instance,-6,6,-6,6,10,2,2);
    cmd=['-c ',num2str(bestc),' -g ',num2str(bestg),' -s 3 -t 2'];
    model=svmtrain(trainval_label,trainval_instance,cmd);
    test_Index=setdiff(rowsIndex,trainval_Index);

    test_label=label(test_Index);
```

```
        test_instance=feature(test_Index,:);
        test_standard_deviations=standard_deviations(test_Index);
        [test_predicted_label, test_mse] = svmpredict(test_label,test_instance,model);

        disp('-------use u as the actual label of the samples [test]--------');
        len=length(test_predicted_label);

        error_sum=0;
        mae_sum=0;
        for j=1:len
            errorTemp=test_predicted_label(j)-test_label(j);
            mae_sum=mae_sum+abs(errorTemp);
            sd=test_standard_deviations(j);
            e=1-exp(-errorTemp^2/(2*sd^2));
            error_sum=error_sum+e;
        end
        test_error=error_sum/len;
        test_mae=mae_sum/len;%计算平均绝对误差

        error=[error,test_error];
        mae=[mae,test_mae];%平均绝对误差
        mse=[mse,test_mse(2)];%平均均方误差

fprintf('iteration=%g;test_error=%g;test_mae=%g;test_mse=%g\n',i,test_error,test_mae,test_mse(2));

    fprintf(logID,'iteration=%g;test_error=%g;test_mae=%g;test_mse=%g\n',i,test_error,test_mae,
test_mse(2));
        fprintf('bestc=%g;bestg=%g;min_mse=%g\n',bestc,bestg,min_mse);
        fprintf(logID,'bestc=%g;bestg=%g;min_mse=%g\n',bestc,bestg,min_mse);
    End

    fprintf('max_mae=%g;min_mae=%g;avg_mae=%g\n',max(mae),min(mae),mean(mae));
    fprintf(logID,'max_mae=%g;min_mae=%g;avg_mae=%g\n',max(mae),min(mae),mean(mae));
    fprintf(logID,'=====================all
end(loose->final)=====================\n');
    fclose(logID);
```

13.3　实验结果分析

该实验使用平均绝对误差作为评判标准，实验结果如表 13-1 所示。

表 13-1　年龄估计实验结果

方法	平均绝对误差（MAE）
CNN_M	6.75816
CNN_F	6.941
CNN_S	6.95691
LBP	8.63488
CNN_M+LBP	7.35418
CNN_F+LBP	7.0583
CNN_S+LBP	7.34369

CNN_M+LBP 表示 CNN_M 特征和 LBP 特征直接连接。

CNN_F+LBP 表示 CNN_F 特征和 LBP 特征直接连接。

CNN_S+LBP 表示 CNN_S 特征和 LBP 特征直接连接。

从表 13-1 可以看出，使用 CNN_M 模型提取的深度特征更具有区分性，它在 LIBSVM 上得到的 MAE 最小，表现出了最佳的准确度。

参考文献

[1] https://competitions.codalab.org/competitions/4711.

[2] X. Zhu, D. Ramanan. "Face detection, pose estimation and landmark localization in the wild" Computer Vision and Pattern Recognition (CVPR) Providence, Rhode Island, June 2012.

[3] http://www.robots.ox.ac.uk/~vgg/software/deep_eval/.

[4] http://cs231n.github.io/convolutional-networks/.

第 14 章

基于深度学习的人脸关键点检测

14.1 问题定义和数据来源

人脸关键点检测是通过图像或视频，定位到图像中人脸各关键点（如眉毛、眼睛、鼻子、嘴巴）的位置。该技术在人脸识别和表情识别中有着十分重要的应用。但是在自然条件下，由于受到光线、人脸姿态和面部表情等条件的影响，要想如何准确检测到人脸的关键点，一直都是十分有挑战性的课题。

本章主要讲解如何使用深度学习完成人脸关键点检测，这是深度学习中一种典型的多标签回归问题。本章主要参考文献[1]中的方法来实现人脸关键点的检测。文献[1]中的方法是基于 Caffe 平台实现深度卷积神经网络模型的训练和运用，但它所使用的 training.csv 和 testing.csv 文件不能直接下载使用，因此，本章使用 BioID 数据集（读者可以通过文献[2]下载）进行相关实验。该数据集有 1465 张可用的图像，每个图像中只有一张人脸，而且每张人脸所包含的 20 个关键点的坐标信息已在该数据集中给出。BioID 数据集的样本图像，如图 14-1 所示。

图 14-1　BioID 数据集的样本图像

14.2　基于深度学习的人脸关键点检测的步骤

本节主要讲解基于深度学习的人脸关键点检测的具体实现过程，该过程主要包括数据预处理、网络训练模型、预测结果处理三个模块。首先，在文献[3]中下载 caffe-facialkp-master.zip 文件，如图 14-2 所示。然后，在 caffe 根目录/examples 文件夹下创建 caffe-facialkp 文件夹，将 caffe-facialkp-master.zip 解压后的文件和文件夹保存在 caffe-facialkp 文件夹中。

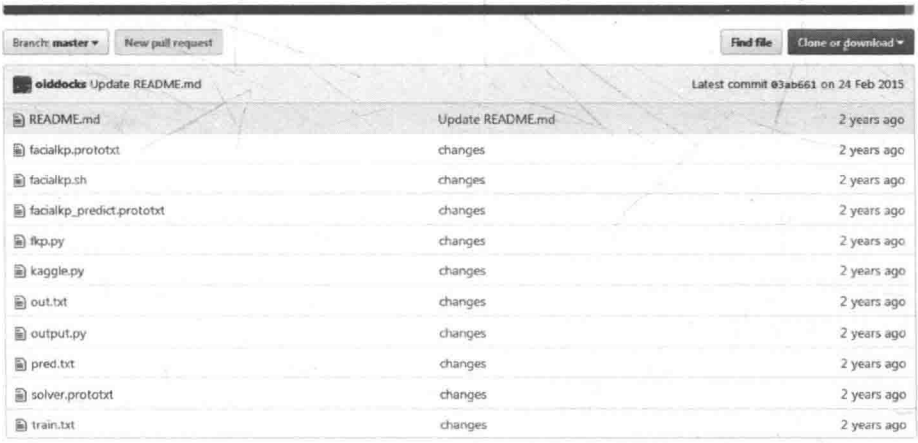


图 14-2　下载 caffe-facialkp-master.zip 文件

14.2.1　数据预处理

神经网络训练对训练数据的格式和类型都是有要求的，因此，在训练前需要对数据做预处理。该处理过程代码都保存在 caffe-facialkp 文件夹下，具体处理过程可分为两步。

（1）运行源文件夹下的 preprocessFace.py 文件。将人脸图像从原图像中剪切出来，并对剪切下的人脸做灰度化处理。接着，将灰度化人脸尺寸转变为 96×96 像素，并将转变后图像保存在 transform/images 文件夹下，在 BioID.txt 文件中保存尺寸为 96×96 像素人脸的真实关键点坐标。

preprocessFace.py 文件的具体代码如下：

```
from PIL import Image
import ImageDraw
from decimal import Decimal
import random

# read BioID file
imfiPath = 'BioID_Annotation.txt'
f = open(imfiPath)
lines = f.readlines()
f.close()

# disorder
random.shuffle(lines)

# save the data for testing
file_object = open("BioID_disorder.txt", 'w')
file_object.writelines(lines)
file_object.close()

# set the size of images
imgSize = 96
# save the result of process
file_object = open("BioID.txt", 'w')
for line in lines:
    data = line.strip().split(' ')
    imName = data[0]
    img = Image.open(imName)

    x = int(data[1])
    y = int(data[2])
    width = int(data[3])
    # the box of face
    box= [x, y, x + width, y + width]
    face =img.crop(box).convert('L')
    scale = imgSize / float(width)
    coordinate = [float(z) for z in data[5:]]
```

```
        # save face whose size is imgSize * imgSize
        file_object.writelines("transform/"+data[0] + ".jpg")
        for i in range(0, len(coordinate) / 2):
            # print(len(coordinate), type(coordinate[2*i]) , type(scale),type( x))
            x0 = (coordinate[2 * i] - x) * scale
            y0 = (coordinate[2 * i + 1] - y) * scale

            fx = Decimal(x0).quantize(Decimal('0.000'))
            fy = Decimal(y0).quantize(Decimal('0.000'))
            file_object.writelines(" " + str(fx) + " " + str(fy))
        file_object.writelines('\n')
        out = face.resize((imgSize, imgSize), Image.ANTIALIAS)
        out.save("transform/" + data[0] + '.jpg')

file_object.close()
```

该图像数据集中每张图像中的人脸，都有对应的人脸关键点坐标。上述代码对训练数据集进行预处理，将每张图像的人脸部分提取出来，并进行缩放，同时缩放对应的关键点坐标的值，最后保存下来。

（2）运行源文件夹下的 fkp.py 文件，该文件的主要功能是将输入输出数据转化成能被深度学习神经网络处理的 hd5 格式。神经网络输入数据是图像，输出数据人脸关键点坐标。首先，将所有的输入输出数据做规范化处理，将它们转化到 0～1 之间（图像数据/255，标签数据/96）的数值，然后，对图像进行直方图均衡化。最后，将上面所有数据分成 3 份。

① Training Set 包含 1100 张图像，用于训练深度学习网络；

② Val Set 包含 150 张图像，用于训练网络时检验 loss 大小；

③ Testing Set 包含 215 张图像，用于使用训练模型预测人脸关键点坐标，并显示网络模型的 loss 大小。

将上面的前两组分别保存在 hd5 格式的文件中。fkp.py 文件中的具体代码如下：

```
import numpy as np
import h5py

from PIL import Image

def dist(x,y):
```

```
        return np.sqrt(np.sum((x-y)**2))

# from http://www.janeriksolem.net/2009/06/histogram-equalization-with-python-and.html
def image_histogram_equalization(image, number_bins=256):

        # get image histogram
        image_histogram, bins = np.histogram(image.flatten(), number_bins, normed=True)
        cdf = image_histogram.cumsum() # cumulative distribution function
        cdf = 255 * cdf / cdf[-1] # normalize

        # use linear interpolation of cdf to find new pixel values
        image_equalized = np.interp(image.flatten(), bins[:-1], cdf)

        return image_equalized.reshape(image.shape), cdf

imgSize = 96
f = open('BioID.txt','r')
lines = f.readlines()
f.close();

name_xtrain = []
print(len(lines))
num = 40;
X = np.zeros([len(lines), imgSize, imgSize], np.float32)
y = np.zeros([len(lines),num], np.float32)

for i in range(0,len(lines)):
        line = lines[i]
        name_xtrain.append(line.strip().split(' ')[0])
        temp = [float(x) for x in line.strip().split(' ')[1:]]
        y[i] = temp[0: num]

        img = Image.open(line.strip().split(' ')[0]).convert('L')
        #Convert the input to an array
        X[i] =   img


y = y / float(imgSize)
#print(y)
```

```
    X = X.reshape(-1,imgSize,imgSize)
    X = X/255

    for i in range(len(X)):
        X[i, :, :] = image_histogram_equalization(X[i, :,:])[0]

    X = X.astype(np.float32)
    print(X)
    X = X.reshape(-1,1,imgSize,imgSize)
    print 'X:', X.shape

    print 'Shape', 'Labels', X.shape, y.shape

    X_train = X[:1100]
    y_train = y[:1100]
    X_val = X[1100:1250]
    y_val = y[1100:1250]
    X_test = X[1250:]
    y_test = y[1250:]

    print 'Train, Test shapes (X,y):', X_train.shape, y_train.shape, X_test.shape, y_test.shape,
X_val.shape, y_val.shape

    # Train data
    f = h5py.File("facialkp-train.hd5", "w")
    f.create_dataset("data", data=X_train,   compression="gzip", compression_opts=4)
    f.create_dataset("label", data=y_train,   compression="gzip", compression_opts=4)
    f.close()

    # Val data
    f = h5py.File("facialkp-val.hd5", "w")
    f.create_dataset("data", data=X_val,   compression="gzip", compression_opts=4)
    f.create_dataset("label", data=y_val,   compression="gzip", compression_opts=4)
    f.close()

    f = open('BioID_disorder.txt','r')
    lines = f.readlines()
    f.close();
```

```
# Test data
file_object = open("BioID_DLTest.txt", 'w')
file_object.writelines(lines[1250:])
file_object.close()
```

通过上述代码，Training Set 和 Val Set 将分别保存在 facialkp-train.hd5 和 facialkp-val.hd5 文件中。而 Testing Set，用于测试，不需要转成 hd5 格式，保存在 BioID_DLTest.txt 文件中。

14.2.2 训练深度学习网络模型

caffe-facialkp 文件夹下名为 facialkp.prototxt 文件定义了一个卷积神经网络。该网络包含 5 个卷积层（CONVOLUTION layer）、3 个下采样层（POOLING layer）、7 个线性修正单元（RELU layer）、2 个全连接层（INNER_PRODUCT layer）、2 个防止过拟合的 dropout 层等。神经网络中数据层的数据类型是 hd5，其中输入输出数据分别是上文提到的经预处理后大小为 96×96 像素人脸图像和 20 个人脸关键点坐标。

运行 caffe-facialkp 文件夹下的 facialkp.sh 文件，最后会得到后缀名为 caffemodel 的模型文件。facialkp.sh 的代码如下：

```
#!/usr/bin/env sh

./build/tools/caffe train --solver = examples/caffe-facialkp/solver.prototxt
```

上述代码中的 solver.prototxt 文件用于设置需要的网络结构文件和网络参数。执行完 facialkp.sh 脚本之后，命令窗体中会出现类似下面的内容：

```
I0223 19:10:08.230324  3702 solver.cpp:342] Snapshotting solver state to
examples/caffe-facialkp/tmp_iter_100000.solverstate
I0223 19:10:08.234257  3702 solver.cpp:264] Iteration 100000, Testing net (#0)
I0223 19:10:08.384732  3702 solver.cpp:315] Test net output #0: loss = 0.0294809 (* 1 =
0.0294809 loss)
```

上面输出的结果中，loss（loss = 0.0294809）就是训练损失函数（平均误差值），0.029×96 = 2.7 是关键点检测的误差。为了减少该误差，我们需要调优 solver.prototxt 中的网络参数，以降低 loss，提高检测精度。注意，该 loss 是对验证数据集 val 中的图像进行预测的结果，所对应的 loss。

执行完成 facialkp.sh 文件后，会得到后缀名为 caffemodel 的模型（默认会与该脚本在相同的文件夹下，但是可以设置）。

14.2.3　预测和处理关键点坐标

预测关键点数据需要用到上面已经训练好的 caffemodel 模型文件和预测网络文件。首先，需要将训练好的模型文件如 tmp_iter_100000.caffemodel 和预测网络结构文件，即 caffe-facialkp 文件夹下的 facialkp_predict.prototxt 文件存放到 [caffe 根目录]/python 文件夹下。然后，在该目录下运行 output.py 文件，将预测的结果保存在 fkp_output.csv 文件中。

output.py 的具体代码如下：

```
import numpy as np
from scipy.spatial import distance

from PIL import Image

import caffe


# Make sure that caffe is on the python path:
#caffe_root = './caffe/'  # this file is expected to be in {caffe_root}/examples
#import sys
#sys.path.append(caffe_root + 'python')


# Set the right path to your model definition file, pretrained model weights,
# and the image you would l1ike to classify.

# from http://www.janeriksolem.net/2009/06/histogram-equalization-with-python-and.html

def image_histogram_equalization(image, number_bins=256):

    # get image histogram
    image_histogram, bins = np.histogram(image.flatten(), number_bins, normed=True)
    cdf = image_histogram.cumsum() # cumulative distribution function
    cdf = 255 * cdf / cdf[-1] # normalize
```

```
    # use linear interpolation of cdf to find new pixel values
    image_equalized = np.interp(image.flatten(), bins[:-1], cdf)

    return image_equalized.reshape(image.shape), cdf

MODEL_FILE = './facialkp_predict.prototxt'
PRETRAINED = './tmp_iter_100000.caffemodel'
f = open('DLtest.txt','r')
tests = f.readlines()
f.close();
name_xtrain = []
print(len(tests))
X = np.zeros([len(tests), 96, 96], np.float32)
#y = np.zeros([len(tests),30], np.float32)
for i in range(0,len(tests)):
    line = tests[i]
    name_xtrain.append(line.strip().split(' ')[0])
    temp = [float(x) for x in line.strip().split(' ')[1:]]
    #y[i] = temp[0: 30]
    img = Image.open(line.strip().split(' ')[0]).convert('L')
    #Convert the input to an array
    arr = np.asarray(img,dtype='float32')
    X[i] =   img

X = X/255
X = X.reshape([-1,96,96])
X = X.astype(np.float32)

# Run Histogram equalization
for i in range(len(X)):
        X[i,   :, :] = image_histogram_equalization(X[i,:,:])[0]

# Scale between 0 and 1

X = X.reshape(-1,1,96,96)

num = 64
BATCH_SIZE = num
```

```
    total = len(X)
    max_value = num     #batch_size from .prototxt
    batches = abs(len(X) / BATCH_SIZE ) + 1

    data = np.zeros((batches*BATCH_SIZE,1,96,96),np.float32)

    data[0:len(X),:] = X

    print 'Input Data shape: ', data.shape
    print 'Total batches: ', batches

    net = caffe.Net(MODEL_FILE,PRETRAINED, caffe.TEST)
    #net.set_mode_gpu()

    predicted = []

    for b in xrange(batches):

        data4D = np.zeros([BATCH_SIZE,1,96,96]) #create 4D array, first value is batch_size, last
number of inputs
        data4DL = np.zeros([BATCH_SIZE,1,1,1]) # need to create 4D array as output, first value is
batch_size, last number of outputs
        data4D[0:BATCH_SIZE, :] = data[b*BATCH_SIZE:b * BATCH_SIZE + BATCH_SIZE, :] #
fill value of input xtrain

        #predict
        #print [(k, v[0].data.shape) for k, v in net.params.items()]
        net.set_input_arrays(data4D.astype(np.float32), data4DL.astype(np.float32))
        pred = net.forward()
        print 'batch ', b, data4D.shape, data4DL.shape

        predicted.append(pred['ip2']*96)

    predicted = np.asarray(predicted, 'float32')
    predicted = predicted.reshape(batches*BATCH_SIZE,40)
    predicted = predicted[:total,:40]

    print 'Total in Batches ', data4D.shape, batches
    print 'Predicted shape: ', predicted.shape
```

```
print 'Saving to csv..'

np.savetxt("fkp_output.csv", predicted, delimiter=",")
```

fkp_output.csv 中存放的是预测的结果值（即人脸关键点的坐标位置），但对应的图像是预处理之后的 96×96 大小的图像，这时我们需要将这些坐标对应到原图中（预处理之前的图像），recover.py 程序文件就是为了完成这一任务。具体代码如下：

```
from PIL import Image
from decimal import Decimal
from numpy import *
import csv

imgSize = 96.0
r = []
# read the result of testing
with open('fkp_output.csv', 'rb') as f:
    reader = csv.reader(f)
    for row in reader:
        temp = [float(x) for x in row]
        r.append(temp)
# read the real coordinates of facial infomation
f = open('BioID_DLTest.txt', 'r')
lines = f.readlines()
f.close()

imgList = []
file_object = open("test_output.txt", 'w')
for index in range(0,len(lines)):
    print(lines[index])
    data = lines[index].strip().split(' ')
    # image address
    imgName = data[0]
    x = int(data[1])
    y = int(data[2])
    # get the size of face
    width = int(data[3])
    # print(x,y,width)
```

```
        img = Image.open(imgName)

        scale = width / imgSize
        # print(scale)
        temp = r[index]
        file_object.writelines(imgName)
        # print(len(temp)/2)
        for k in range(0, len(temp)/2 ):
            fl = temp[2*k] * scale + x
            ft = temp[2*k + 1] * scale + y
            print(fl,ft, temp[2*k], temp[2*k+1])
            fx = Decimal(fl).quantize(Decimal('0.000'))
            fy = Decimal(ft).quantize(Decimal('0.000'))
            file_object.writelines(' ' + str(fx) +' '+str(fy))

        file_object.writelines('\n')

file_object.close()
```

经过上面的处理后，就能把预测结果在原图中正确输出。最终的检测效果如图 14-3 所示。

图 14-3　深度学习检测结果

总结而言，使用深度学习进行人脸特征点的定位时，在数据进行预处理之后（对图像进行缩放，转为 hd5 格式，将数据划分为 training、val、test 三部分），训练深度学习模型需要执行如下步骤：

（1）执行 facialkp.sh，该脚本将使用 Caffe 工具训练深度学习模型。该脚本定义了训练数据集的输入文件及其位置（如 facialkp-train.hd5 和 facialkp-val.hd5）。

该脚本也会读取 solver.prototxt 文件，该文件定了网络模型的相关参数，尤其是 base_lr 和 weight_decay 这两个参数。该步骤执行完毕后，最终会生成一个模型文件，如 tmp_iter_100000.caffemodel。

注意：本步骤会输出训练损失函数（平均误差值），这里，需要基于 val 数据集，反复调优 solver.prototxt 中的网络参数，尤其是 base_lr 和 weight_decay 这两个参数，以得到较小的 loss 值，提高检测精度。最后，使用最小的 loss 值对应的 base_lr 和 weight_decay 参数，及对应的模型（如命名为 tmp_iter_100000.caffemodel 的文件），该模型将用于（2）阶段的预测。

（2）使用步骤（1）中最终确定的模型文件，如 tmp_iter_100000.caffemodel，执行 output.py 脚本，预测 test 数据集中每个图像的人脸关键点坐标，并将该坐标缩放（对应）到原图中的坐标。

参考文献

[1] http://corpocrat.com/2015/02/24/facial-keypoints-extraction-using-deep-learning-with-caffe.

[2] https://www.bioid.com/About/BioID-Face-Database.

[3] https://github.com/olddocks/caffe-facialkp.

深度学习总结展望篇

第 15 章

总结与展望

本章将首先总结深度学习领域当前的主流技术/工具，以便读者有所选择地进行针对性的学习、研究和应用；然后，指出从事深度学习研究和应用开发面临的问题；最后，展望深度学习领域未来的发展方向。

15.1 深度学习领域当前的主流技术及其应用领域

15.1.1 图像识别

当前，图像识别（包括人脸识别）领域主流的深度学习工具是深度卷积网络 Deep CNN（DCNN）和 R-CNN 技术。

卷积神经网络（Convolutional Neural Network，ConvNet）是深度前馈网络 Deep Feedforward Network 的一种。相对于邻接层全连接的深度前馈网络，卷积神经网络更容易训练、更容易泛化。现在，Deep Convolutional Network（深度卷积网络，DCNN，一般有 10～20 层 ReLU 激活层、数百万个权重、数十亿个单元之间连接[1]）是计算机视觉领域广泛采用的神经网络，在人脸识别、图像识别领域中有极佳的应用效果。

R-CNN（Regions with Convolutional Neural Network Features）技术对图像中的子区域（候选框中的图像区域）卷积神经网络进行训练[2]，能够极大地提升物

体检测（定位）的精度。

15.1.2　语音识别与自然语言理解

当前，语音识别与自然语言理解领域主流的深度学习工具是循环神经网络 RNN 中的 LSTM 技术。

对于有顺序关系的输入（sequential inputs），主要是语音和语言文本，循环神经网络 RNN（Recurrent Neural Networks）通常是比较好的工具。RNN 尤其擅长预测文本中的下一个字或者是序列中的下一个单词。但是，RNN 的问题在于：RNN 很难学习很长的信息[1]。

Long ShortTerm Memory（LSTM）解决了 RNN 的这一问题。LSTM 是一种改进的 RNN 架构，尤其适用于事件（序列）之间有较长的时间滞后的情形，对序列进行分类、处理和预测。百度、搜狗就使用了 LSTM 模型，进一步提升了语音识别的正确率。双向 LSTM（Bi-direction LSTM）考虑了更多的上下文信息，性能普遍优于单向 LSTM[3]。

LSTM 的问题在于：LSTM 的训练很容易发散。因此，Google 提出了 LSTMP，它在传统的 LSTM 模型之上，引入了一个反馈层。这个反馈层会使运算的计算量大幅下降。例如，初始设定的神经元节点是 1024 个，LSTMP 反馈“采用 256 个神经元节点就可以了”，因此，能够大幅压缩计算量[4]。

15.2　深度学习的缺陷

任何事物都有两面性，包括深度学习。深度学习尽管在图像识别、语音识别领域取得了很多进步，但仍存在很多突出的问题、弊端。

15.2.1　深度学习在硬件方面的门槛较高

普通的办公 PC 或小型服务器，一般缺乏强悍的 GPU 硬件支持，且并不是所有的 GPU 都适合深度学习。GeForce Titan X 是入门级（简单、经济型的）的

深度学习 GPU。而 Tesla K40、K80 是标准型的深度学习 GPU，以一块 Tesla K40 为例，其价格是 2～3 万元。再配上合适的电脑主板，CPU，内存等，一台能够运行深度学习软件平台的计算机约需 3～4 万元。而且，随着 GPU 数量的增加，其价格也成倍增加。目前市场上有最高支持 8 块 Tesla K40 GPU 的主板，一台这样的服务器约需 30～40 万元。

而 NVIDIA（中国）相关部门的负责人表示，为了满足一个团队的在深度学习科研方面的需要，建议最少购置 40 块 Tesla K40 GPU，而购买这么多的 GPU 所需总费用约为 150～200 万元，甚至更多。

只有重点高校、大中型公司才有足够的资金，大规模的购置所需的 Tesla K40 或 K80 的 GPU，以加快深度学习计算、尤其是调参的效率。即便是一台装有一颗 Tesla K40 的计算机，也需要 3～4 万元，这让很多研究生、甚至研究人员在尝试深度学习入门时就望而却步。百度首席科学家吴恩达指出，深度学习的前沿正转向高性能计算。

以百度为例，其深度学习基础设施使用了 6000 台 PC 服务器和超过 2000 余台的 GPU 服务器。而谷歌在 2012 年发起让算法自己学会识别猫脸的项目，运行在 16000 个 CPU 处理器上，使用（学习）了 1000 万个 YouTube 上的视频。

15.2.2 深度学习在软件安装与配置方面的门槛较高

以 Caffe 深度学习平台为例，其安装、配置极其复杂、烦琐，依赖诸多其它安装包。有很多学生，花了一个月甚至更多的时间，都没能将 Caffe 正确安装。

即使安装成功了，初学者也往往需要 1～2 周的时间进行摸索、查资料，才能安装成功。而且，后期 Caffe 与 GPU 的显卡驱动的兼容性方面，容易出现无法进入操作系统界面的问题（只能进入命令行），需要重新安装 Caffe 才能解决这样的问题。有些时候，甚至需要重装操作系统才能解决问题（这是由 Ubuntu 操作系统与 NVIDIA Tesla K40 硬件兼容性，以及该显卡的驱动引起的）。深度学习软件平台的安装与配置方面的易失败性、复杂性，也让很多初学者望而却步。

基于 Docker 的快速环境部署有望大幅降低初学者学习深度技术的门槛。

15.2.3 深度学习最重要的问题在于需要海量的有标注的数据作为支撑

百度首席科学家吴恩达在斯坦福大学时，曾设计了一款咖啡机，需要识别出

不同的杯子。为了提高识别的准确度，满足产品的要求，他们跑遍了旧金山，买来所有他们可以买到的杯子，并从各个角度对这些杯子都拍了照片，总共获得了5 万张杯子的照片，并将这些照片作为机器学习算法的训练样本。只有这样，对杯子的识别模块最终才能以较高的准确度运行。而跑遍整个旧金山购买所有杯子就是为了获取标记数据。

尽管有 ImageNet、FDDB、LFW 等少数几个大规模开源数据集，但是从事深度学习的研究和开发，必须拥有更多种类、更大规模的、面向某一行业或某一主题的、有标注的数据集，才能有效地开展相关的研究。

百度深度语音识别系统的成功，很大程度上是因为百度拥有 10 万小时的声音语料作为训练数据，及其规模庞大的基于 GPU 的深度学习基础设施（使用了6000 台 PC 服务器和超过 2000 余台的 GPU+FPGA 服务器）。事实上，百度研究团队收集了 9600 个人长达 7000 小时语音，并在这些语音样本中增加了 15 类噪音。最后，他们将这项语音样本扩容成一个 10 万小时的数据。

以互联网色情图片过滤为例，百度的训练数据囊括了 1.2 亿幅色情图像，分类精度达 99.4%。

这种大规模的、有标注的数据，只有大公司或大型机构（如政府部门、电信）才可能拥有，而普通高校、学生、研究人员很难采集或获取到这样的数据。对于普通的科研人员而言，数据的规模远远比不过这些大公司，而计算设施也是远远不如这些大公司（几台 GPU V.S.上千台 GPU），因此，在高校里面，深度学习领域越来越难以做出超越大公司的算法、技术和成果。

15.2.4　深度学习的最后阶段竟然变成枯燥、机械、及其耗时的调参工作

著名机器学习专家周志华教授曾表达（或引用了）如下观点：“有点幽默，但很朴实，深度学习现在差不多就是民工活，调来调去，刷来刷去。文章发得飞快，貌似热闹，但有多少是能积淀下来的实质真进展，又有多少是换个数据就不靠谱了的蒙事撞大运？何曾见过调试 SVM 核函数 3 元一个。既缺乏清澈干净的内在美感，又不致力于去伪存真、正本清源，只图热闹好看，迟早把 arXiv 变成废纸堆。”

的确如此，在确定了深度网络结构后（自己设计或借用现有的网络结构），深度学习相关的工作只需要设计对数据进行预处理的方法；然后使用尽可能多的

数据输入到深度网络中进行训练；最后，需要反复地调参，直到找到一组最佳参数。很多时候，调参居然变成最耗时、最关键的工作。从事深度学习的工作，最后跟别人比的（竞争的）竟然仅仅是所使用的这一组参数而已。很多已经发表的论文中，很多作者都没有公布所使用的这一组最佳参数。这使得其他人员重现论文中的最佳结果时，耗费数周甚至数月的时间。

从事深度学习的研究时，像本书作者这样的普通研究人员很少能在理论证明、推演方面做出重要贡献。

夸张一点说，除了研究问题的定义和数据预处理这两方面之外，使用深度学习这个工具解决科学问题时，很多人基本上不进行理论推导，而只是反复、迭代地调参，直到找到一组最佳参数，如此而已。而且，在一个数据上调参得到的最佳参数，往往与另一个数据集上的最佳参数不同。于是，每一个数据集，都需要反复、迭代地调参，深度学习某种意义上就变成了机械、枯燥的调参工作。

深度学习尽管不需要人工干预特征的提取，却需要人工干预调参的过程，而且调参的过程耗时极长，少则数周，多则数月。

15.2.5 深度学习不适用于数据量较小的数据

在小规模数据集上，使用深度学习的技术，其效果往往较差。识别字迹、自然场景中的文字时，人类不需要大量的数据集，只看一眼就能够准确地识别字迹、文字。人类在小数据集上，也能取得极佳的识别准确度。

譬如，比较两个字迹是否出自同一个时，人类不需要提前观察、学习大量人的字迹，相反，他们往往只需要仔细观察、推演这两个字迹本身，就能准确地辨识字迹是否出自同一个人。

进行人脸识别时，人类也不需要提前学习、记忆成百上千个人脸图像，相反，他们一眼就能识别出目标人物的身份或名字，“小数据”就足够了。因此，深度学习与人类的智慧的差距仍然是十分巨大的。

15.2.6 深度学习目前主要用于图像、声音的识别和自然语言的理解

而在其它领域中，如数据管理、科学计算、算法设计、并行计算、数理统计、实际问题的解决思路构思等方面，深度学习也许并无用武之地。而本书作者在大

量“小数据”上的实验结果表明，在数值型或分类型数据的分类方面，如小规模数据集上的分类模型训练，深度学习的预测效果也要远远逊色于传统的机器学习算法（如 SVM，Random Forest，GBDT 等）。

15.2.7　研究人员从事深度学习研究的困境

目前，像本书作者这样的普通研究人员，通常缺乏在深度神经网络方面深层次的理论理解与推导方面的功力。深度学习平台以及应用开发的方法，普通的研究团队花费 1 年或更少的时间，应该能够熟练运用。但是发表学术论文的话，如果只有应用的结果而没有理论或算法的支撑，将很难发到好的期刊或会议上。

而本书作者在河南大学所指导的深度学习研究小组正是遭遇了这样的困境：我们在深度学习深层次理论理解和推导方面欠缺，只是用深度学习这个“工具”去解决一些传统的模式识别问题，如人脸表情识别；然后，将深度学习的结果和传统的非深度学习方法对应的结果进行对比，如果比人家好了，也许还可以发一个不太好的期刊。但如果结果比不过人家，就需要反复的调参。有时即使调参了 1～2 个月，还是比人家差 1%左右，这个过程，非常痛苦、非常枯燥、非常缺乏技术含量、非常浪费时间。最关键的问题是，如果深度学习方法的最终结果比不过已发表的论文，整个工作等于白做了，而且基本上无法沉淀技术和理论。

本书作者的研究生做的两篇论文，都是遭遇了上面类似的问题。作者现在甚至怀疑，让硕士研究生做深度学习是不是不合适，毕竟在理论推导方面，硕士研究生的功力基本不行，而用深度学习做应用的话，还得看实验结果，如果实验结果没人家非深度学习的方法好，最后就什么也得不到，正如本书作者的研究生所经历的。即便用深度学习的结果比人家好，那也是花了 1 个月甚至数月的时间调参调出来的、“刷”出来的，结果的提高并不是来自理论的创新或方法上的改进，而仅仅是调参的功劳。正如周志华老师所引用的话：“深度学习现在差不多就是民工活，调来调去，刷来刷去，文章发得飞快，貌似热闹，但并没有多少是能积淀下来的真进展、真原理、真算法、真技术”。

但不可否认，深度学习已经在图像识别、声音和自然语言的理解方面取得了空前的成功。

15.3 展望

目前，对深度学习的主要批评是许多深度学习方法缺乏理论支撑，这也是深度学习领域未来亟需解决的关键问题之一。如深度学习技术需要多少训练样本和计算资源才能学习到足够好的模型。又如，深度学习如何在理论方面获得支撑，以便将某个数据上的调参限制在一个较小的范围内，以加快深度学习的效率。

未来需要更多深度学习应用落地、百花齐放。例如，最近两年，国内的创业公司已经基于深度学习等技术开发出花朵识别的应用软件。越来越多的公司、研发人员也在尝试使用深度学习解决面向某一领域或某一主题的实际问题。而如何降低深度学习研究和应用开发的门槛，使其容易上手，使得深度学习的使用过程变得简单、容易操作、容易发现和解决问题、在应用实践方面给予具体理论指导，是深度学习领域必须解决的又一关键问题。

目前深度学习的成功大多来自于监督学习（有标注的数据上的训练、学习），而非监督学习将是深度学习未来的重点研究方向之一。

深度卷积网络 DCNN 与使用强化学习（Reinforcement Learning）的循环神经网络 RNN 的结合，将是未来的另一重点研究方向[1]。

参考文献

[1] Yann LeCun,Yoshua Bengio, Geoffrey Hinton. Deep learning. Nature 521, 436–444 (28 May 2015). doi:10.1038/nature14539.

[2] 基于 R-CNN 的物体检测. http://blog.csdn.net/hjimce/article/details/50187029.

[3] 深度学习与自然语言处理之五：从 RNN 到 LSTM. http://blog.csdn.net/malefactor/article/details/50436735

[4] 百度贾磊 LSTM+CTC 详解. http://blog.csdn.net/u014114990/article/details/49949075.

本书的特色是深度学习理论与应用实践并重，能够帮助读者在3个月时间内掌握深度学习的基本理论和实战技能。其特点在于：

1. [illegible]
2. [illegible]
3. [illegible]
4. [illegible]
5. [illegible]

对初学者、工程师和研究生而言，本书在手，深度学习理论与实践均无忧。

策划编辑：董亚峰（dyf@phei.com.cn）
责任编辑：赵娜
封面设计：朝天世纪

ISBN 978-7-121-30413-2

定价：48.00元