中山大学硕士学位论文

基于深度学习智能标注图片关键字系统的设计与实现

The Design and Implementation of Deep Learning-based Picture Auto Tagging System

专业：　　软件工程

学位申请人：

导师姓名：

论文答辩委员会主席 ：

成员：

年　　日

论文原创性声明

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

学位论文作者签名：

日期：　年　月　日

学位论文使用授权声明

本人完全了解中山大学有关保留、使用学位论文的规定，即：学校有权保留学位论文并向国家主管部门或其指定机构送交论文的电子版和纸质版，有权将学位论文用于非赢利目的的少量复制并允许论文进入学校图书馆、院系资料室被查阅，有权将学位论文的内容编入有关数据库进行检索，可以采用复印、缩印或其他方法保存学位论文。

学位论文作者签名：　　导师签名：

日期：　年　月　日　日期：　年　月　日

论文题目：基于深度学习智能标注图片关键字系统的设计与实现

专业：软件工程

硕士生：XXX

指导教师：XXX

摘　要

在现今这个信息爆炸的时代，每时每刻人们都在拍摄大量的数码图片。在这其中，有很大一部分图片都是来自用户的智能移动设备。在这样大规模的数据量下，为了满足互联网用户对查找图片和归档图片的需求，对拍摄的图片进行关键字标注变得非常必要。而传统对于拍摄图片的标注，往往仍旧在使用基于图片上下文中的文字中的关键字来指定，并非使用图片内容本身。一些现存的基于图像内容的检索引擎，并不是很适合解决上述问题：一方面并非提供图片标注，而是提供相似图片，不是很符合关键字标注的需求；另一方面需要依靠网络传输和服务器计算资源，在移动端使用并非很便利。

本文针对以上问题，提出了一种基于深度学习方法的解决方案，使用户即时拍摄的图片可以在本地计算进行分类标注，并且将 PC 端的计算和移动端的计算分离，从而缓解时刻产生的大量图片对服务器计算造成的负荷。深度学习不同于以往的机器学习方法，不需要人工去指定给计算机对于数据的抽象表示方法，而是通过给定庞大的样本集，让计算机自己去找出对数据的抽象方法。因此本解决方案的核心就是在 PC 端寻找数据的抽象表示方法，并将该抽象方法使用在移动端从而实现本地化分类。

论文的主要工作包括：（1）实现了基于深度学习方法的训练计算和分类计算分离开的多平台系统，各平台间定义了通用的分类配置；（2）在 PC 平台使用 Python 实现了一个可借由 gpu 进行加速的完整的卷积神经网络训练器；（3）在 iOS 平台使用 objetive-c 实现了一个使用卷积神经网络方法进行分类的分类器；（4）在 iOS 平台上开发了一个完整的包括即时拍照、导入照片、下载分类配置、进行分类标注各功能的 APP 应用；（5）使用不同的图像库对卷积神经网络的训

练过程进行测试和调优，最终优化出了使准确率和神经网络规模较为平衡的神经网络结构。

本论文根据软件工程的开发流程，对本系统进行了详细的分析、设计以及实现。目前，本系统以初步完成一个原型系统，实现了从提供的图片集提取图片数据生成分类器，到移动端使用该分类器进行分类标注的一系列完整功能。

关键字：深度学习，卷积神经网络，图片分类

Title: The Design and Implementation of Deep Learning-based Picture Auto Tagging System

Major: Software Engineering

Name: XXX

Supervisor: XXX

Abstract

In this modern era of information explosion, people are taking a lot of digital pictures all the time. And there are a large part of the pictures are taken from the user's smart mobile devices. In such a massive amount of data, in order to meet Internet user's demand for finding and achiving pictures, keyword tagging for pictures becomes very necessary. The traditional method for tagging picture, often only use context-based text keywords to tag the picture, not the picture itself. And the existing content-based image retrieval systems are not suitable for solving the above problem: on the one hand, they are not providing tags, but providing the similiar pictures; on the other hand, they are rely on network and server computing resources, are not convenient on the mobile device.

Aiming at the problem, we proposed a solution based on deep learning approach. This solution can provide tagging suggest locally just after user taking a picture, the classify caculation are processed locally. It separates the server caculation and the device caculation, thus easing the heavy computing on the servers caused by the large quantities of pictures. Unlike the previous machine learning approaches, deep learning does not need to be assigned a method for data representation, but will find out its own data representation method by giving it a huge set of samples. This solution's keypoint is to find a data representation on PC platform, and use this representation on mobile platform for classification computing locally.

The main work of this paper includes:(1)Achieved a system based on deep learning which separates the caculation of training and classifying to different platforms, connected by sharing a property configuration;(2)Implemented a convolutional neural network trainer on PC platform, which written in Python and can be accelerated by using gpu computing;(3)Implemented a convolutional neural network classifier on iOS platform, which written in objective-c;(4)Developed a applicatoin on iOS platform, which can download classification configurations, taking pictures and tag the pictures using downloaded configurations;(5)Used different images datasets to test and tune the convolutional neural network's training process, figured out a balanced, optimized neural network architecture.

In this paper, based on software engineering development process, we carried out a detailed analysis, design and implmentation explanation for the system. Currently the system is initially completed as a prototype system, which has achieved a series of full functionality from generate image sets to use the classifier to tag a picture on mobile device.

Keywords: Deep Learning, Convolutional Neural Network,
Image Classification

目录

第一章 前言

在现今这个信息爆炸的时代，每时每刻人们都在拍摄大量的数码图片。在这其中，有很大一部分图片都是来自用户的智能移动设备。在这样大规模的数据量下，为了满足互联网用户对查找图片和归档图片的需求，在移动端使用自动关键字标注是一个合适的解决方法。

本章主要介绍了图片关键字标注问题的背景及现状，以及应用深度学习处理该问题的意义，之后介绍了深度学习的历史和研究现状，最后阐述了本文的工作结果和论文的组织结构。

1.1 图片关键字标注问题的现状以及应用深度学习的意义

近年来随着互联网引起的信息爆炸扩散至我们生活的各个角落，整个世界每日产生的信息量仍在以惊人的速度剧增。在这些大量的信息下，有很多的信息是对我们没用的，冗余的。相应的，有很多信息以原信息的状态存在着，对其加以分析和提取出对我们有用的信息的工作慢慢地变得至关重要。这其中占很大比例的同时也是很重要的一种信息就是，图片信息。

在现今这个信息爆炸的时代，每时每刻人们都在拍摄大量的数码图片。人们现今每天产生的数码照片数目，比摄影 1839 年最初发明后 100 年内拍摄的照片数目都要多。根据 2013 年 Facebook 提供的数据，平均每天用户上传了 3 亿 8 千万张图片至 Facebook，他们累计已经为用户保存了 2500 亿张图片[1]。在这其中，有很大一部分图片都是来自用户的智能移动设备。由于现今的智能移动设备上普遍配置摄像头，大家早已养成了随身拍、随手拍的拍照习惯。用户除了单纯的上传外，也有着对图片进行归档、以供日后查找的需求。在这样大规模的数据量下，用户就连对自己产生的照片进行检索管理也是个难题，更不用说加上用户

附近相关的庞大的图片集。

面对以上问题，使用传统的方法进行基于图像内容的检索，即 CBIR（Content-based image retrieval），来尝试解决，似乎并不特别合适。首先基于图像内容的检索一直以来都是一个极其消耗计算资源和存储资源的难题，难以在用户的移动设备上直接运行；其次如果依靠网络和服务器资源，一方面在如此大的数据规模下加重了服务器的负担，另一方面也要依靠网络传输，损失了移动端的便利性；而且传统的基于图像内容的检索方法，需要提前针对所要处理的对象的特性作分析，提取出其相应的模板，才能有效地用于检索，例如人脸匹配算法等等，不具有解决普遍标注问题的能力。

图 1-1　　带有人工标注关键字的照片

为了满足互联网用户对查找图片和归档图片的需求，现今较好的解决方法仍是依靠文字，对拍摄的图片进行关键字标注，从而实现对图片的标签化分类管理。传统的照片分享网站，大多数仍是基于人工进行关键字标注和管理（如图 1-1）。传统的图像搜索引擎，如 Google 和百度的图片搜索（如图 1-2），对于拍摄图片的关键字标注，往往仍旧在使用图片上下文中的文字中的关键字来指定，并非使用图片内容本身来进行。

图 1-2　基于上下文关键字的图片搜索

本文尝试使用机器学习中近年来新兴的深度学习的方法来解决上述图片标注的问题。深度学习是指深层神经网络学习方法，是近年来机器学习的一个新突破点。以往的机器学习算法，往往是配合一些人类预先定义的特征提取表达方法，在对原始数据进行一定的抽象和分析后，提取出用以表达原始数据的主要信息的特征（如在图像领域著名的 SIFT 特征[2]），然后才对该特征使用机器学习方法来进行所需要的分类等分析工作。而人类所定义的特征表达，总会存在着一些缺点和问题，很难表达过于复杂的结构，并且有时无法找出原始信息中的关键信息的好的表达方式。因此，人们认为机器学习的工作范畴不应该仅仅限于对提取的特征进行分析归类，而应该进一步扩展至对复杂结构的构造表达上。

而随着需要表示和构造的计算过程越来越复杂，以往热门的基于浅层的机器学习方法，例如 SVM、Boosting，自身所包含的节点数与能够表达的结构的复杂性是呈算术级数增长的，如果要表示过于复杂的结构，相应的节点代价过高。而深层的神经网络，自身所包含的节点数与能够表达的结构的复杂性是呈几何级数增长的，因此表达代价更低，也可以处理更大的数据。

基于以上所述的优势，深度学习不同于以往的机器学习方法，甚至不需要人工去指定给计算机对于数据的抽象表示方法，而是通过给定庞大的样本集，让计

算机自己去找出对数据的抽象方法。由于深度学习可以面向未经处理的原数据，自动抽象出数据中的关键信息，因此非常适合应用在图片标注上，免去了传统的人工分析各个标注的特性，从而为每个标注订立模板的过程。一切都由计算机自动分析，自动训练计算获得。并且经过计算机自动分析获得的分类器数据，并不受平台和计算能力的限制，可以以较低的性能要求完成分类计算的需要，从而可以将分析与实际应用的场景分开，并不完全依靠服务器资源，这与传统的 SIFT 等方法也是非常不同（因为该类方法的应用也需要较高计算量）。

综上，因此本文使用深度学习作为解决图像分类标注问题的算法理论基础，并将训练器与分类器进行分离，在不同的平台上进行实现，充分利用了 PC 端的计算能力和移动端的便捷性，构成了完整的图片标注系统的原型。

1.2 深度学习方法的历史和现状

深度学习即是指深层神经网络，一般指层数大于 3 层的神经网络结构。

人工神经网络（artificial neural network），是一种模仿生物神经网络而功能和结构的数学模型兼计算模型。它由大量的节点（神经元，unit）相互连接组成，节点之间进行联结计算，计算结果通过连接传向上方的节点。每个节点都代表一种特定的输出函数，称为激活函数（activation function）。每两个节点间的连接代表对通过该连接的数据的权值（weight），用以表示神经网络的记忆。外界信息通过传感器流入第一层神经网络，经过一层层神经网络的计算，最后得出计算结果（如图 1-3）。在这个过程中，通过对神经网络中的权值参数进行调整，使得输出结果尽可能贴近原本经过现实验证的比对结果，就是神经网络的训练过程，从而使得神经网络尽可能模拟原本未知的算法。

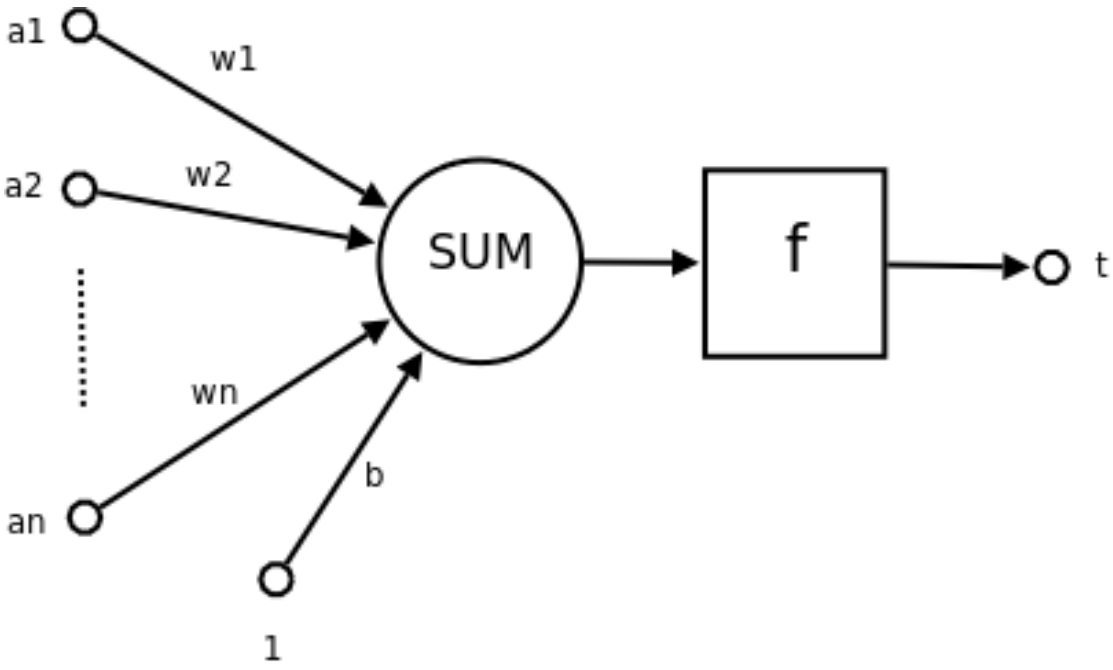

图 1-3　一个人工神经元的示意图

人工神经网络理念的最初提出是受到人和其他动物神经网络功能的运作原理的启发而产生的。在上世纪 40 年代初，人工神经网络最初由心理学家 W.S.McCulloch 和数理逻辑学家 W.Pitts 提出[3]，通过单个神经元执行逻辑功能，从而使构成的神经网络能够执行更复杂的功能。在之后多年的发展中，神经网络模型不断地被改进和完善。直至今日，神经网络被认为是一种非线性统计性数据建模工具[4]，用来对输入和输出间的复杂关系进行建模和拟合，从而探索数据的模式。然而由于神经网络的不稳定性和参数的众多，调整训练神经网络变得非常困难，并且在过去的计算能力下几乎是不可能的。神经网络方法在当时并没有太多的应用空间。

上世纪 90 年代初，由 Vapnik 等提出的支持向量机(Support vector machines, SVM)方法[5]，大大改变了这一现状。他们通过使用退化为 1 层的浅层神经网络（如图 1-4），使神经网络的训练变得更加简单和可行，并可应用于许多现有解决框架中。以及同一时代由 Freund 等提出的 Boosting 算法[6]，实质上也为浅层神经网络的变种。一时间，特征提取+浅层神经网络的解决框架几乎成为了面对各种问题的通解和最优解，并带来了人工智能和机器学习的飞速发展。

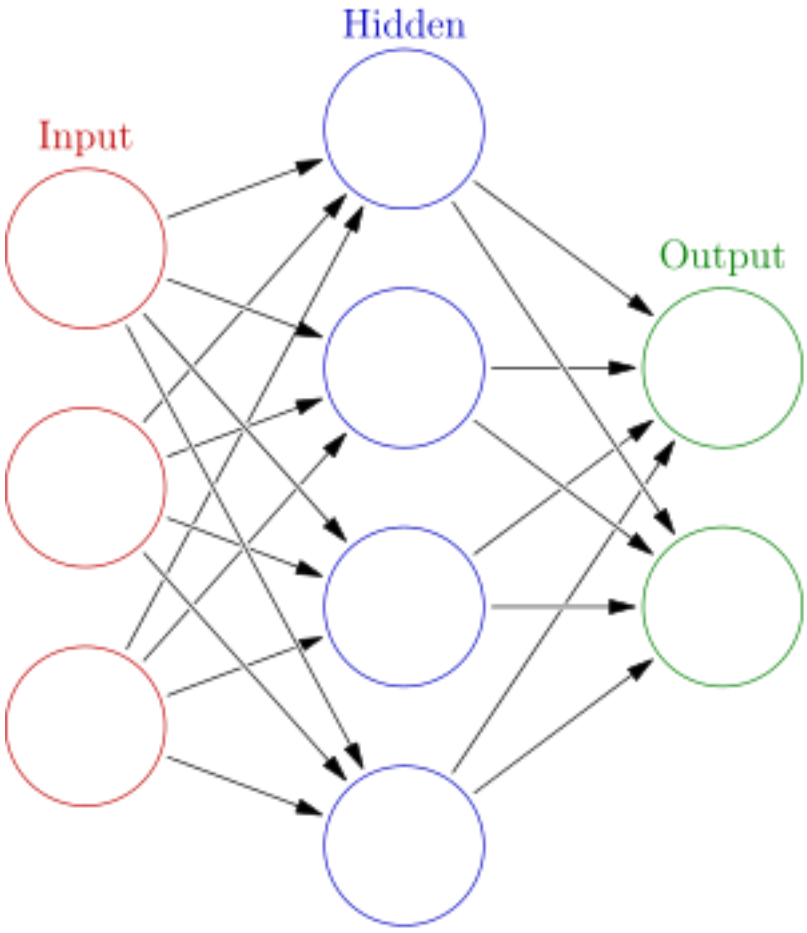

图 1-4　一个拥有一层隐藏层的浅层人工神经网络

然而，浅层的神经网络不得不面临一些问题：特征提取方法有时很难找到行之有效的，并且其所包含的节点数和能表达的多项式复杂度呈算术级数增长，即如果要表达更复杂的模型，需要增加大量的参数和计算量。

而与之相对的，深层神经网络恰好是这两种问题的答案：深层神经网络能表达的复杂度为各层的节点数的乘积，即所包含的节点数和能表达的多项式复杂度呈几何级增长。因此可以以较少的参数轻易的表达一些复杂的模型，并且由于可以表示复杂模型，甚至可以直接把原始数据当做神经网络的输入，把特征提取表达的功能也囊括在神经网络的工作范畴内。

但是深层神经网络仍旧面临着不稳定和训练难的问题，传统的反向传播算法[7]很难在其上进行高效的调整。在近些年内出现两种成功的解决方式：一个是本文中所使用的卷积神经网络（Convolutional Neural Network）方法，在处理图像

数据上有突出的效果；另一个是基于无监督训练的深信度网络（Deep Belief Network）。

卷积神经网络[8]来源于对猫的视觉皮层细胞结构的模拟，根据视觉区域逐层抽象图像[9]。在其中定义了卷积层，在卷积层中使用卷积的方法使数据间稀疏连接，且使用共享权值大大减少了所需训练的参数数量。因此使得反向传播算法训练变得可行，且收到了不错的成效[10]。最初成功的模型为 LeCun 于 1998 年提出的 LeNet5[11]（如图 1-5），后来被美国大多数银行广泛用于支票上的手写数字识别。

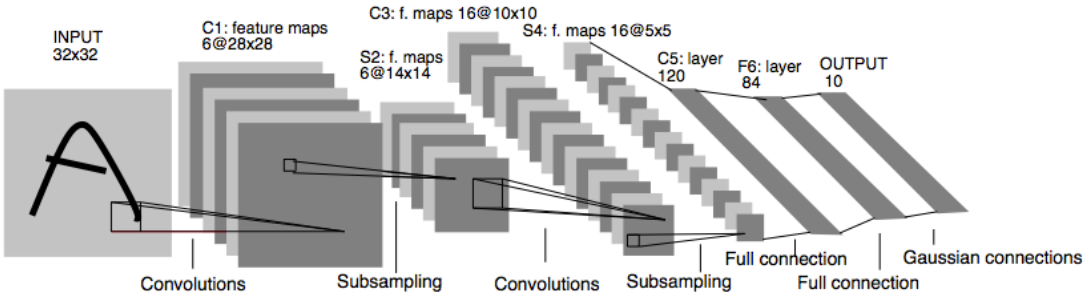

图 1-5　　LeNet5 的结构示意图

深信度网络[12]是由加拿大多伦多大学的 Hinton 提出的一种新的深层神经网络的无监督训练方法[13]（如图 1-6）的深度网络。其摒弃了传统的训练方法，通过定义局部的代价函数，一层一层间各自分别训练，每一层都是一个受限玻尔兹曼机（RBM）[14]，并在最后使用反向传播对整个神经网络进行调优从而实现对神经网络的训练。由于其层间的训练是使用局部的代价函数，其训练过程几乎是生成的，训练时间大大减少；而后来使用反向传播进行调整时，由于有之前的预训练的权值，因此计算性能也比单纯的反向传播算法要好，并且收敛速度要快，是近年来深度学习研究最初的突破点[15]。与该算法相关的研究有堆叠自动编码器[16]等，该算法成为无监督学习的重要理论基础。

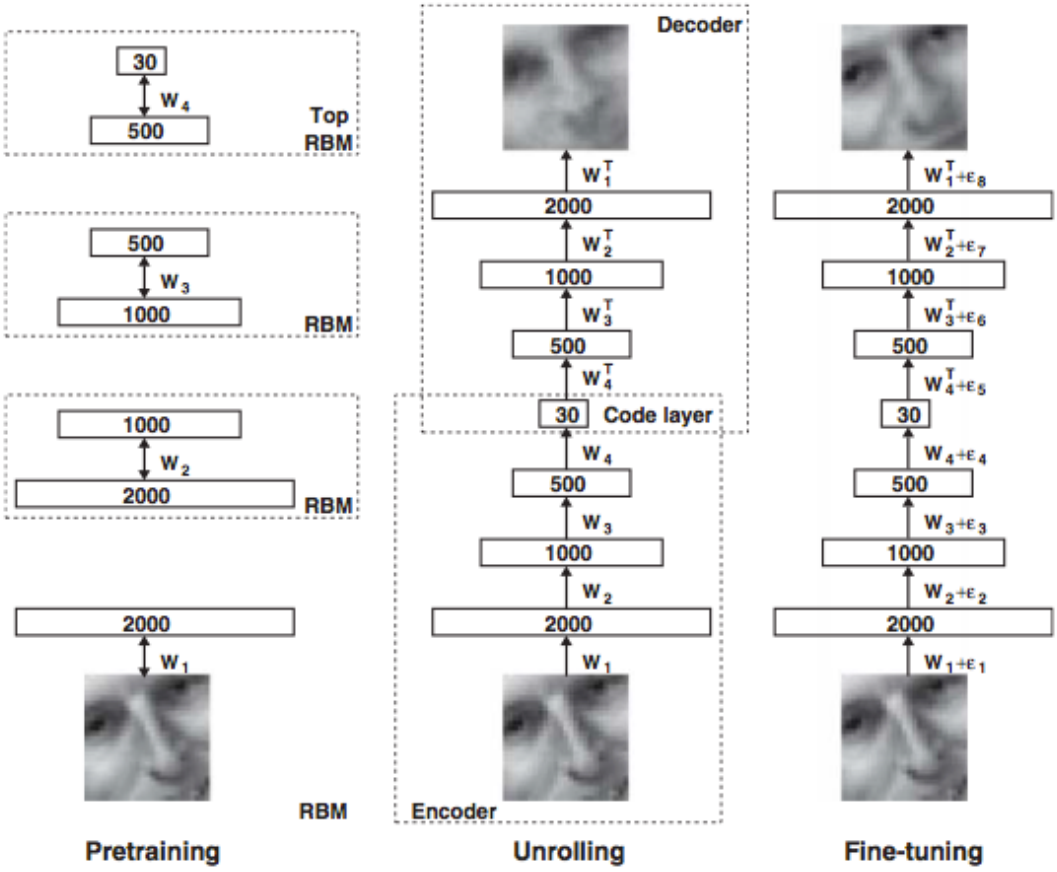

图 1-6　　深信度网络的训练方法

深度学习方法提供了与传统的特征提取表达不同的崭新道路，完全由机器自己学习特征，并以此为分类或匹配的标准，从而大大减少了寻找特征的工作量，且对各种问题找到了一个可复用的通解。深度学习作为近些年来新兴的一种机器学习架构，受到了越来越多的研究机构以及企业的重视，并在多方面彰显出了不凡的潜力，诸如语音识别、图像匹配等。深度学习领域的泰斗 Hinton 教授的学生使用卷积神经网络参加了 ImageNet 的图片分类比赛，获得了惊人的成绩，在 120 万张的图片库中，将前五分类错误率由当前技术的 25%减少到了 17%[17]。微软使用深度学习方法，研发出了一个英汉间的即时翻译系统，识别率已可达到 80%-90%，且几乎可达到实时口译发音的效果[18]。谷歌使用深度学习研发的 DistBelief 系统，可以在未事先获取猫的特征描述信息的情况下，在给定了大量含有猫的视频后，通过自学习自行总结出猫的特征[19]。国内走在深度学习前沿的公司百度，也推出了基于深度学习的以图找图的百度识图产品。其他诸如 Facebook，IBM，淘宝等公司也在纷纷加入这一技术的竞争。

1.3 本文的工作

本文提出了一种基于深度学习方法的解决方案，使用户即时拍摄的图片可以在本地计算进行分类标注。在该方案中，将 PC 端的计算和移动端的计算分离，从而缓解时刻产生的大量图片对服务器计算造成的负荷。

论文的主要工作包括：

对卷积神经网络的原理和特性进行了探讨和研究，并提出了一种将训练器和分类器分开在不同的平台上的解决方案。将训练器在 PC 平台上实现，而分类器在移动平台上实现，其中训练器训练得到的结果通过封装为配置文件下载至移动端的分类器使用。这样一方面可以利用 PC 端的高性能进行训练，另一方面将实际进行分类标注的计算量分摊在移动端，不必给云端增加不必要的负荷。该解决方案主要分为以下四部分：

1. PC 平台的训练器：该部分主要使用 Python 进行开发，实现出一个可借由 gpu 进行加速的完整的包括训练部分的卷积神经网络，并包括其中的参数封装导出。
2. 移动平台的分类器：该部分主要使用 Objective-C 开发，实现出使用现有参数计算和传递激活值的卷积神经网络。
3. 移动平台完整的 APP：在 iOS 平台上开发了一个完整的包括即时拍照、导入照片、下载分类配置、进行分类标注各功能的 APP 应用，用以展示最终系统的结果。
4. 样本图像集的生成器：在 PC 端将任意分类标注好的图片作为输入，转化为可被训练器用于训练的样本集数据。

由上，本文最终实现了一个基于深度学习方法的训练计算和分类计算分离开的多平台原型系统，并对其使用不同样本图像库进行测试和调优，最终优化出了使准确率和神经网络规模较为平衡的神经网络结构。

本文的主要贡献主要为：将深度学习推至应用阶段，用以解决即时拍摄的图片标注问题，将运算量庞大的训练处理和不需很高运算量的分类处理分开，将云端训练所得的配置文件下载至移动端，从而在移动端进行分类标注时不需依靠云端服务器运算，只需移动端的处理能力即可完成。

1.4 论文结构简介

本文的组织结构如下：

第一章阐述和分析了图像标注问题的背景和现状、深度学习的相关现状，以及介绍了本文的主要工作。

第二章论述了本文的理论基础，对卷积神经网络的原理进行了深入的剖析，研究了各种训练深度神经网络的优化方法，并介绍了本文在实现过程中所使用的关键技术和库。

第二章也包括了本文在理论方面的核心贡献，分析了在不同情况下时配置卷积神经网络的结构要点，以及参数训练时的优化经验等等。

第三章是系统需求分析，分别对该系统所要实现的主要功能加以分析，并转化为用例进行详细描述，以及提取出系统的领域模型。

第四章和第五章包含了本文的主要贡献，即设计和实现了一套完整的基于深度学习方法的训练计算和分类计算分离开的多平台关键字标注系统。

第四章是系统总体设计，介绍了系统的整体架构和设计思路，以及各主要功能的实现思路，并根据用例转化出相应的业务流程图，最后对系统包结构、类结构进行详细的分析与设计。

第五章是系统的详细设计与实现，详细介绍了各模块的设计和实现方式，包括其中包含的类、重要类中包括的重要函数以及模块与模块间的共享配置文件规格等。

第六章是部署与应用，主要对系统的开发和运行环境进行简单介绍，并分析了各模块的测试结果，最后结合主要界面对最终 APP 应用的关键输入输出进行了说明。

最后一章是全文的总结，提出该系统后续仍需完成的工作，以及未来应用前景的展望。

第二章 卷积神经网络原理和关键技术框架

本章包含了本文的主要理论贡献。本论文的主要工作是利用卷积神经网络实现一个多平台的、训练与分类分离开的关键字标注系统，因此本章将阐述实现该系统时所需用到的核心技术，即卷积神经网络的主要原理，并归纳了在本系统的研究使用过程中对卷积神经网络结构设计上的具体理解和调整修改，以及卷积神经网络在训练时的优化经验。此外，本章还介绍了在实现的过程中所用到的一些关键技术框架。

2.1 卷积神经网络原理

卷积神经网络（CNNs）属于深层神经网络，其深度一般在 4 层以上。在面临传统定义的深层神经网络时，由于每一层的节点间存在 O(n*m)级别的参数（n 为输出节点数，m 为输入节点数），难以解决由于其众多的参数所引起的梯度扩散、反向传播算法性能慢的问题，使得训练过程极其缓慢，且其效果并不比浅层的神经网络（SVM）好。而卷积神经网络是第一个真正成功训练多层网络结构的学习算法。其利用空间关系减少所需要学习的参数数量，使得梯度扩散和反向传播算法的性能问题得以抑制。

卷积神经网络来源于对猫的视觉皮层细胞结构的模拟，根据视觉区域逐层抽象图像。在其中定义了卷积层，在卷积层中使用卷积的方法使数据间稀疏连接，且使用共享权值大大减少了所需训练的参数数量。由于其对输入数据无特殊要求，因此还最小化了数据的预处理要求。因此，卷积神经网络成为了最初成功的深层神经网络结构，且直至今日仍为非常有潜力的视觉识别方面的有效的机器学习方式[20]。

2.1.1 卷积神经网络结构

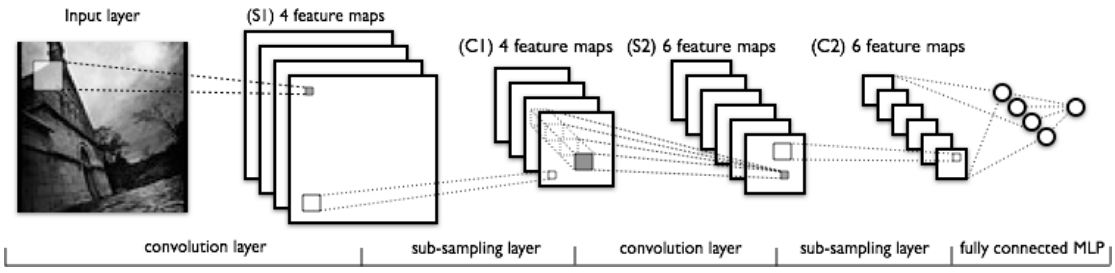

图 2-1　卷积神经网络

如图 2-1，本文中所实现的卷积神经网络自左至右由以下四种神经网络层组成：

卷积层、池化层（或称为下采样层）、全连接层、分类层。

整个数据流向大致为：最左端的输入样本数据首先流入若干个卷积层和池化层。经过卷积层和池化层的计算提取后，将所得的样本特征向量流入若干个全连接的神经网络层，进行进一步的计算拟合。经过全连接层的计算后，将输出得到的样本特征向量传入一个传统的分类器（逻辑回归），并得到该样本为各个分类的概率。

整个训练的过程就是对于所给的样本集，调整各层参数，使得经过神经网络计算得到的结果尽可能接近样本标注。

各层参数调整方式为梯度下降法，且使用反向传播算法，下层的梯度由上层的梯度计算获得。

以下我们依次讲解本文所使用到的各种算法和定义。

2.1.2 神经网络层的通式

由于神经网络的目的在与拟合一个非线性函数，因此神经网络的每一层的本质即为一个非线性变换，整个神经网络为这些变换的乘积。

因此，神经网络层的通式大体可以写为公式（2-1）：

$$f(x) = s(b + Wx) \quad (2-1)$$

其中 x 表示原向量。f(x)表示 x 经过变换后所得的新向量。W 为变换方式，例如在多数情况下为一个变换矩阵。b 为变换后所要加的偏移量。s 为一个选取的非线性函数，又称为激活函数，从而实现变换的非线性化，常用的有 sigmoid、tanh 等。

我们用以上公式表示一层神经网络层所代表的变换。

2.1.3 梯度下降算法

梯度下降算法[21]是最常用的神经网络训练方法，其使用计算代价函数的梯度的方法来调整参数，使得变换尽可能拟合原来的变换。

对于以上公式表示的神经网络，对于分类问题给定的一个样本(X,Y)，其中 X 代表数据，Y 代表标注，我们就可以依此定义一个在该样本上的代价函数（2-2）：

$$C(f,X,Y) = cost(f(X),Y) \quad (2-2)$$

其中 f 的参数为 b、W。

由于在给定确定的样本时，其中 X、Y 都是确定的，因此代价函数就变为以参数 b、W 为自变量的函数。令确定的样本集为 D，则原代价函数可以转变为以下函数（2-3）：

$$C(\theta = \{W,b\},D) = \sum_{i=0}^{|D|} cost(f(x_i),y_i) \quad (2-3)$$

依照以下公式针对代价函数分别求各个参数的梯度（偏导），公式（2-4）：

$$\frac{\partial C(\theta)}{\partial \theta} = \lim_{\delta\theta \to 0} \frac{C(\theta+\delta\theta) - C(\theta)}{\delta\theta} \quad (2-4)$$

将得到的偏导数作为使代价函数变小的调整方向，根据以下的公式（2-5）更新参数：

$$\theta^{k+1} = \theta^k - \varepsilon_k \frac{\partial C(\theta^k)}{\partial \theta^k} \quad (2-5)$$

其中ε_k为每次改变的步长，直到梯度趋近为 0 时，找到该代价函数的一个局

部极小值点，从而得到使得拟合尽可能贴合的一组参数 b、W。

样本集的代价即为各个样本的代价之和。

传统的梯度下降算法指对整个样本集的代价做计算，而在样本集规模较大时，其性能很受影响。因此在传统梯度下降的基础上，梯度下降也有新的变种。

随机梯度下降是指，每次只取一个样本来计算梯度，这样在大量的样本计算后，其梯度的方向会有一个平衡，即统计信息近似于原样本集的梯度。该方法可以极大缩短每一步移动的计算时间，并得到不错的结果。

批量随机梯度下降是随机梯度下降的另一个变种，其每次不是取一个样本，而是一批固定数量的样本，这样一方面减少了每一步的计算量，另一方面也减少了个别样本对参数调整的影响量，使得移动更加稳定更具有统计学效力。

本文中所使用的就是批量随机梯度下降算法来进行训练。

2.1.4 反向传播算法

由于神经网络有许多层，而标注数据只可以与最后一层输出层进行比对，因此对于整个神经网络：

在分类层（即逻辑回归），其代价函数可以定义为如下公式（2-6）：

$$C(\theta = \{W, b\}, D) = -\sum_{i=0}^{|D|} \log\big(P(Y = y_i | x_i, W, b)\big) \tag{2-6}$$

而在其它层，其代价函数为参数变化对最终的代价的影响，因此需要计算当层和以后所有层直到输出，才能计算出梯度。

在这种情况下，对于前面层的参数进行调整时，需要计算很多后续层的数据，调整性能大大降低。

因此为了解决这种传统前向传导（Feedforward Pass）调整时的性能问题，出现了反向传播（Backpropagation Pass）调整算法[22]，用于快速计算深层神经网络各层的梯度。

其主要核心思想为，后面层的参数的梯度，事实上是由前面层的参数的梯度组成的，且与前一层的输出值有一定联系。因此，只要知道后一层的各参数的梯度值，再加上前一层到后一层的输出结果，就可以反推出前一层各参数的梯度值。

这样就可以去除掉反复计算后续层的值和梯度的过程，大大减少了计算量。

其具体过程如下：

首先使用前向传导方法，计算出网络中的所有节点的输出值，在这里称为激活值a；

然后对于最后一层（即输出层）计算该层各个节点的输出值变化对最终代价影响的残差值$\delta^{(n)}$，如公式（2-7），其中 p 为对激活值标准化后的概率，n 为总神经网络层数，i 为当前输出节点；

$$\delta_i^{(n)} = \frac{\partial C}{\partial a_i^{(n)}} = (p_i - 1_{y=i}) \quad (2-7)$$

之后从后往前，依层计算各层各个节点的激活值变化对最终代价影响的残差值$\delta^{(l)}$，由它的下一层的残差值$\delta^{(l+1)}$加权获得，权值即为变换矩阵 W 相应的权值，如公式（2-8），其中 s'为该层的激活函数的导数；

$$\delta_i^{(l)} = \left(\sum_{j=0}^{m_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) s'(a_i^{(l)}) \quad (2-8)$$

最后，计算各层参数的梯度，b 即为其下一层的残差值，W 为该层激活值与下一层残差值的积，如公式（2-9）、（2-10）。

$$\frac{\partial C}{\partial W_{ij}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)} \quad (2-9)$$

$$\frac{\partial C}{\partial b_i^{(l)}} = \delta_i^{(l+1)} \quad (2-10)$$

由此，计算出各层各参数的梯度，使用梯度下降法进行调整。

2.1.5 卷积层

卷积神经网络的核心就在于卷积层的定义。

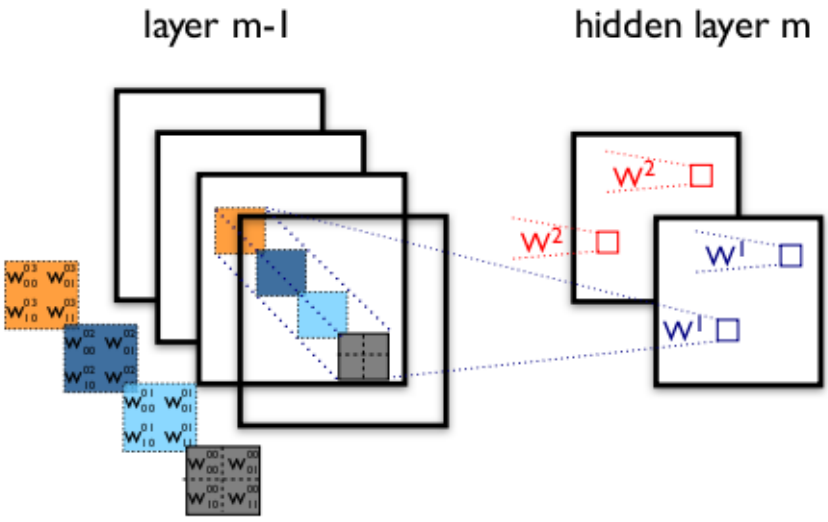

图 2-2　卷积层

如图 2-2，卷积层的计算过程就是卷积的过程，使用若干个三维的卷积核在该三维矩阵上做二维卷积（因为第三个维度大小是相等的），把各个卷积核的结果以矩阵的第三维度为区分，再经过非线性函数，输出为一个三维矩阵，其中将这些卷积核的矩阵作为这一层的参数。卷积层的激活值计算公式（2-11）为：

$$a_{ij}^{k} = s((W^{k} * x)_{ij} + b_{k}) \quad (2-11)$$

该层拥有稀疏连接、共享权值的特点，一方面大大减小了计算量，计算时只使用了附近的数据；另一方面大大减小了参数数量，只把有限个卷积核当做参数。

举例说明，假设输入输出皆为 100*100，则全连接层的变换矩阵为 10^4*10^4，计算量为 10^8，参数数量为 10^8 个；而如果是包括 20 个 5*5 的卷积核的卷积层，则计算量为 10^4*25*20，参数数量为 25*20。可见卷积层在前几层面对巨大的节点数时，仍保持较少的参数和计算量，并保留了其中的关键信息，使得训练深层神经网络变得可行[23]。

该层的梯度定义为：对下一层各个激活值的影响之和，即在反向传播时，该层的梯度为其导出的所有激活值联合求和起来计算得到的梯度。

2.1.6 池化层

在卷积层之间存在着一些池化层，用于在保持多数信息的情况下，减小传递至下一层计算的数据量。

池化的意思即为，在一个区域中（例如 2x2），抽取出其中一个值当做该区域的代表值。

这样可以快速减少数据量，且会保持大部分信息不丢失，并可在一定程度上减少图片平移、旋转变换（小范围）的影响[24]。

该层不存在需要调整的参数，其目的仅仅为减少数据量。

一般使用的方法有最大池化、平均池化等。

2.1.7 全连接层

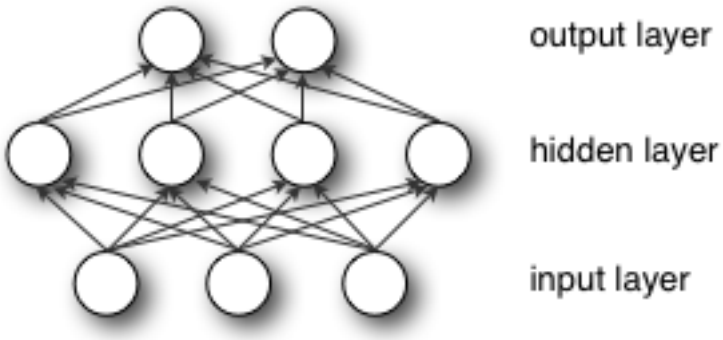

图 2-3　全连接层

如图 2-3，全连接层顾名思义即为输入节点与输出节点全连接的层，其依靠一个 n*m 的矩阵 W 作为变换矩阵，一个 n 维的向量 b 作为偏差，其中 m 为输入节点数，n 为输出节点数。将输入向量右乘矩阵 W，再加上偏差 b，最后再对计算所得到的向量使用非线性函数，从而得到该层的输出结果。全连接层的激活值计算公式（2-12）为：

$$a^k = s(W^k x + b^k) \quad (2-12)$$

该层的梯度定义为：与反向传播算法中原定义的类似，从下一层的偏差值中计算获得。

2.1.8 分类层

该层作为输出层，可以使用各种分类器（SVM 等），在本文中我们使用最简单的逻辑回归。

逻辑回归与线性回归类似，也是用线性变换的方法拟合函数。但与线性回归最主要的不同为，线性回归拟合的是连续函数，而逻辑回归所拟合的函数得到的结果只为 0 到 1 的一个概率。在经过前面计算得到各个结果的比重后，使用一个 softmax 函数来将总概率进行归一从而得到各结果的概率[25]。

定义 n 个类的逻辑回归函数如公式（2-13）：

$$P(Y=i|x,W,b)=softmax_i(Wx+b)$$
$$=\frac{e^{W_i x+b_i}}{\sum_j e^{W_i x+b_j}} \quad (2-13)$$

该层的梯度可以直接由结果与样本的标注比较计算获得，即偏差代价的公式（2-14）为：

$$C(\theta=\{W,b\},D)=-\sum_{i=0}^{|D|}\log[P(Y=y_i|x_i,W,b)] \quad (2\text{-}14)$$

2.1.9 卷积神经网络的结构设计分析及优化经验

由于卷积神经网络的复杂性，其结构设计一直是一个复杂的问题。由于深层神经网络结构的复杂性，以及参数数量的众多，如何合适的定义各层的参数量、初始值及调整步长，对训练的结果影响很大，而且一直也是个难题。当设计的参数过多时训练速度会极大下降，且收敛速度也会变慢；当设计的参数过少时，又很容易遇到收敛瓶颈，无法得到最优结果。

在本文的实验研究和对比参考过程中，总结出了以下经验。

首先是针对不同的图片集所需使用的结构策略。

（1） 图片尺寸较大的图片集。该情况下使用较大的池化步长，使得前几层卷积结果的尺寸快速缩小。可适当提高卷积核尺寸，但不宜超过 10x10。

（2） 图片尺寸较小的图片集。该情况下使用较小的池化步长或不使用池化层。卷积核尺寸在 3x3 或 5x5 较好。

（3） 图片分类较多的图片集。该情况下应使用较多卷积层和卷积核，从而增大计算时可分布的空间，从而增加可拟合的范围，找到更优解。

（4） 图片分类较少的图片集。该情况下应使用较少卷积层，且后面的层可以使用较少的卷积核，从而加快训练速度且不会过拟合。

（5） 图片分类间辨识度不高的图片集。该情况下应该加大前一二层的卷积核数目，凸显图像细节特点的比重。

除了上述各种特点的图片集的不同处理策略外，卷积神经网络设计还有以下的共同策略：1. 经过所有的卷积层后，输出结果尺寸大致为 10x10 以内，连入全连接层；2. 全连接层的单元数要与分类层所要分的分类数有关，不宜过大，否则；3. 除了辨识度不高的图片集外，其他情况下每层卷积层的卷积核数目大致相同，但不宜过大；4. 一般由于训练速率等关系，在一般的设备上，卷积层不宜超过 5 层。

其次是训练经验和遇到的问题。

前面的卷积层要快速将节点数规模缩小，可以借助大卷积窗口、池化等方法，后面的则不需要池化，卷积窗口也不宜过大。

全连接层的节点数不要太多，节点多了容易过拟合，并且训练时所需内存大小和所需时间也大幅提升。

在处理大规模数据及大规模的神经网络时，有些特殊的激活函数可以加速收敛，例如 f(x)=max(0,x)，也称为限制的线性化单元[26]（Rectified Linear Units，它也是个非线性函数，简称 ReLU）。而也有些其他的可以一定程度解决过拟合的影响，例如 f(x)=|tanh(x)| [27]。具体在针对不同的样本集时多进行尝试或许会找到最好的结果。

池化中最大池化往往效果较好，能提取出最具代表性的细节数据，而平均池

化没什么太大意义。

对于一些数据集应有较简单的预处理，比如白色背景很多的数量级，会由于白色的像素值为（255，255，255）产生很多干扰效果，使训练难度加大，若将外围的白色替换为黑色，或者对图片做反色处理，效果会好很多。

2.2 关键技术框架

本文在实现的过程中，为高性能完成神经网络的矩阵操作，引入了一些为完成关键技术而使用的框架，并且分为 PC 平台和移动平台。在 PC 平台上包括 Numpy 库和 Theano 框架，在移动平台上包括 Accelerate 框架。下面我们将简单介绍各个库或框架的功能和特点。

2.2.1 Numpy 库

Numpy 是一个开源的 Python 科学计算扩展包[28]（如图 2-4），它为用户提供了大规模的 N 维数组、矩阵的表示方法以及针对这些数组的数学处理函数，并且其较高的运行效率甚至可以与 C 语言的代码相当。

图 2-4 Numpy

在 Python 下，由于代码是未经编译而是由解释器转换为字节码解释执行的，因此在进行大规模运算时效率会比起类似 C 语言的编译执行的语言慢好几个数量级。而 Python 由于其易用性和代码的清晰性，现今又被广大科学研究者青睐，很多研究者使用 Python 能很容易的实现他们所要的数学算法，但由于其运行效率低下，使得大计算量的工作扔需要很多人去重写晦涩难懂的 C 语言代码。因此，使用 Python 进行数学计算是广泛存在的需求。

Numpy 即是为了解决 Python 在大规模科学计算时过慢的问题而出现的。在 Numpy 中，N 维数组在内存中储存的方式不同于一般 Python 数组的储存每个对

象的指针这样的处理方式，而与 C 等编译执行的程序类似，是以连续内存分类来实现的，其中每个数只用内存储存而并不表示为 Python 对象，并加以数组形状的概念来实现将其表示为高维数组。从而 Numpy 提供的函数可以在操作这些连续内存时，可以使用一些局部的基于 C 语言实现的关键函数，从而使计算效率得到极大的提高。在使用 Numpy 进行大规模矩阵运算时，其运行效率基本与同功能的 C 代码效率相当，并且其保留了 Python 数组对象的操作方式，保持了语言的便利特点。

Numpy 包含了许多与 MATLAB 相似的数学计算函数，从而支持科学计算算法的实现。其主要提供了以下几方面功能：1. 提供了一个强大的 N 维数组对象；2. 提供了比较完整成熟的操作数组对象的重载函数；3. 包含了线性代数、傅里叶变换、随机数生成等常用的科学计算函数；4. 可以与 C/C++或 Fortran 代码进行集成的工具包。

在本文中 PC 平台的运算上，主要使用了 Numpy 的矩阵作为处理时的操作对象。

2.2.2 Theano 框架

Theano 是一个开源的 Python 下的基于 Numpy 的数学计算框架[29]（如图 2-5），其主要为用户提供了数学表达式的符号表示，即定义了数学表达式的函数对象。利用其提供的函数对象，可以实现在定义了一个数学公式或表达式的对象后，代入不同的自变量从而计算该公式的结果。由于其表达式和公式是以对象的性质存在的，因此可以在运行时动态调整该函数对象所对应的功能，而不需重新编写相应的函数。这极大的解耦了众多科学计算中各式各样的计算要求，使得定义、实现算法更加便利。

图 2-5　theano

Theano 内部对该功能的实现方法为保留一个完整的关于表达式的图，其中包含了一些不变量和自变量，从该图中生成出该表达式对应的计算函数，从而实

现表达式计算。在生成的过程中，Theano 也可以进一步优化计算次序等，使得表达式的计算更加高效。Theano 同时又包含了一些 C 和 CUDA 代码，其最终生成执行的代码为基于 C 编译的到的，在生成时可以选择使用 gcc 或者 nvcc 来编译，从而实现使用 CPU 进行计算或 GPU 进行计算。

在本文的 PC 平台的神经网络训练上，各调整操作函数主要基于 Theano 计算框架实现。

2.2.3 Accelerate 框架

Accelerate 框架是苹果公司在 Mac 和 iOS 系统上提供的计算框架，该框架集成在苹果公司提供的开发环境 Xcode5 中（如图 2-6），主要用于优化各种科学计算操作在苹果设备上的性能。

图 2-6 Xcode5

Accelerate 提供了上百个针对 Mac 和 iOS 设备进行优化过的数学函数，其中包括了信号处理、快速傅里叶变换、基本矢量和矩阵运算等多个方面，以及例如分解矩阵因子和线性方程组求解系统等线性代数计算的函数。并且 Accelerate 针对不同设备中存在的不同硬件配置进行了优化，只需编写一次代码，它就可确保代码能在所有设备上高效运行。

Accelerate 框架中提供的函数为宏定义的 C 语言函数，在 iOS 的 objective-c 下可直接编译执行，且其中处理的矩阵等数据并不需要包装为类，而是直接提供连续储存的浮点数数组和相应的形状参数即可。

在本文的移动平台的计算上，使用了 Acclerate 的 vDSP 库中的函数来进行相应的矩阵运算。

2.3 本章小结

本章介绍了本系统中所使用的卷积神经网络算法的基本原理，该网络为一个包含了卷积层、池化层、全连接层和逻辑回归分类层的一个多层的神经网络，使用反向传播的梯度下降法进行调整，从而实现分类功能。下一章从需求分析开始，

完整的以软件工程分析、设计及实现的框架，来介绍本系统的具体实现方式。

第三章 需求分析

从本章开始，本文将以软件工程分析、设计和实现的角度来具体阐述本系统完成的过程和方式。本章将从需求分析开始，详细对整个系统的需求进行分析，并分离出相应的用例、领域模型等，为下一章进一步的设计工作打下基础。另外，也对卷积神经网络算法实现过程中的数据流向进行分析，以便以后章节进一步的算法实现。

3.1 系统概述

本系统的主要目的是利用图片分类技术在移动端对即时拍摄的照片进行关键字标注。本论文尝试为以上需求提出解决方案。该方案将移动端分类器与 PC 端训练器分开，构建了一个通过配置文件将分类器与训练器联系起来的图片标注系统。本系统的完整功能如图 3-1 所示：

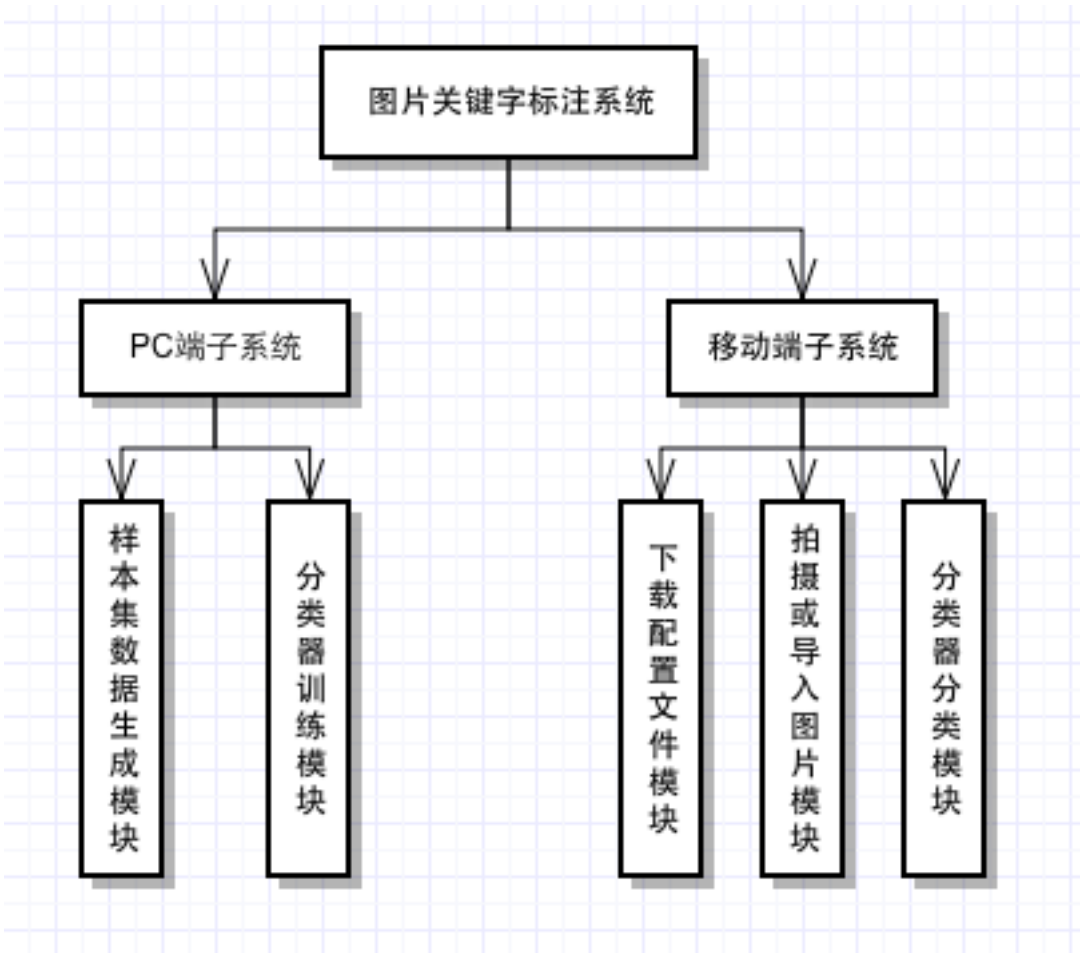

图 3-1　　系统需求说明图

本系统的整体设计思路如下:(1)通过使用网上搜集的图片组成分类图片库，或使用现有的图片库，生成训练器所需的训练集数据；（2）构建训练器模块，在 PC 端使用卷积神经网络算法进行训练，从中获得分类器的配置文件，在此过程中使用带有 GPU 加速的库进行加速计算；（3）构建分类器模块，在移动端使用训练获得的配置文件，使用卷积神经网络实现在本地对本地数据进行分类；（4）设计并开发移动端 APP，并实现拍照功能、下载配置文件及使用分类器对即时拍到的图片进行标注。

通过以上解决方案，使得移动端仅需使用训练得到的配置文件在本地进行分类计算，从而将计算负荷由服务器分摊到各移动端设备上，且并不需要与服务端的即时网络传输。

3.2 用例分析

根据以上整个方案的整体设计思路，系统的各模块间的关系与用例总览如以下用例图 3-2 所示：

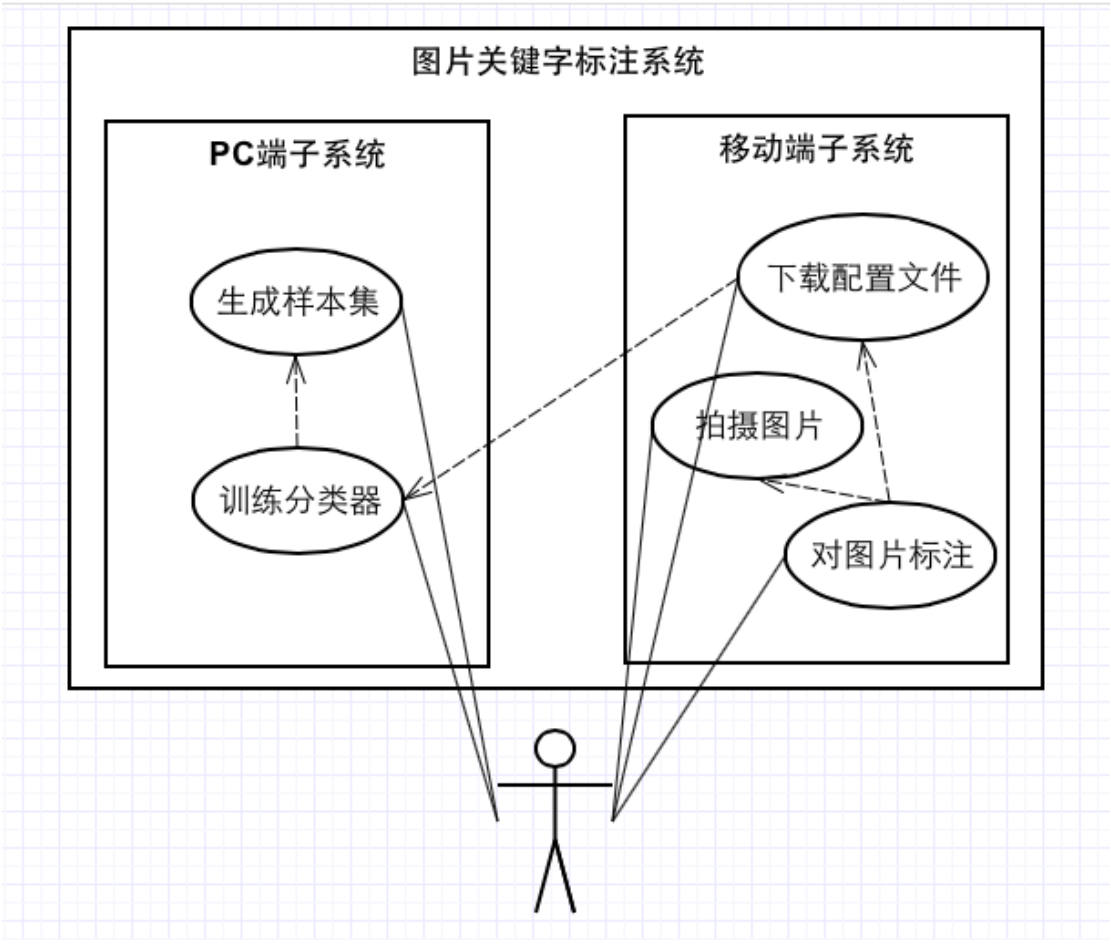

图 3-2　　系统用例图

整个系统分为 PC 端和移动端两大部分。其中 PC 端又包括生成样本集数据、训练分类器这两个样例，移动端包括下载配置文件、拍摄或选取图片、使用配置文件进行分类三个样例。如表 3-1 所示，采用用例列表的形式对每个用例的功能做简要的介绍：

表 3-1　　用例列表

用例号	用例名称	用例描述
UC.1	生成样本集数据	根据用户提供好的图片生成用于训练集的样本集数据
UC.2	训练分类器	根据所提供的样本集数据，训练卷积神经网

		络，并保存配置文件
UC.3	下载配置文件	在移动端下载之前 PC 端训练好的配置文件
UC.4	拍摄或选取图片	在移动端拍摄新的图片或选择已有图片，转换为分类器可用的输入数据
UC.5	使用分类器分类	调用下载得到的配置文件，对输入数据进行分类标注

由于本文篇幅限制，在此不对以上用例一一详细列举。下面使用用例详述表来对其中三项用例的具体流程及扩展点细节进行详细描述。

3.2.1 生成样本集数据

生成样本集数据为训练神经网络的前置工作。对于所要解决的不同问题，可以通过设立不同的样本集来实现。用户首先准备一定的样本图片，再按所需处理的分类分开存储，从而构成样本集。用户所分类储存的图片，由于性能和解耦原因，无法被分类器直接读取。因此，首先要对用户所储存的图片，通过将其转换为 numpy 直接可读取的格式，从而建立样本集数据。下表 3-2 描述了该用例的用例详述：

表 3-2　用例 1 的用例详述

用例编号	UC.1
用例名称	生成样本集数据
用例概述	根据用户提供好的图片生成用于训练集的样本集数据
参与者	用户
前置条件	用户拥有本文所描述系统的程序
基本流程	1. 用户准备若干的样本图片和其相应的分类。 2. 用户建立一个工作目录，并把所需的 python 文件拷贝到该目录下。 3. 用户在该目录下新建一个 property.txt 文件，并在其中填写生成图片数据的信息（分辨率、通道数等）。 4. 用户在该工作目录下根据样本图片的分类建立文件夹，每一个

	文件夹代表一个分类，并把该分类的图片放置在该文件夹下。 5. 用户打开命令提示符，并通过“cd”命令定位到该工作目录下。 6. 用户在命令提示符中输入执行 python 程序的指令，对该组样本集生成样本集数据。 7. 命令提示符显示出该组样本集的相应数据（类数，图片数等），并将样本集数据保存为该目录下的 data 文件。 8. 完成。
扩展点	1a. 用户还未准备图片集。 1. 可以使用本文提供的简易爬虫程序抓取淘宝店铺内该页面的所有商品的图片。 2. 重复进行上一步操作从而获得多个分类。 2-5a. 提示用户权限不够。 1. 请用户在有权限的目录下进行操作。 6a. 提示无法进入路径。 1. 请根据命令提示符的错误提示进行操作。 7a. 程序执行出错。 1. 请确认环境中安装了 python 和 numpy，如果未安装请安装。 2. 请确认命令提示符拥有该目录的写入权限。 3. 请确认 property.txt 按规则填写。
后置条件	使用生成的样本集数据，进行神经网络训练。
补充说明	用户使用的图片类型为 jpg 格式

3.2.2 训练分类器

该样例为本系统在 PC 端的核心工作。通过样例 1 我们已经生成了卷积神经网络训练器所要使用的样本数据集，接下来就是最重要也是计算量最大的工作，训练卷积神经网络。使用用户设定好的神经网络结构，生成并训练神经网络参数从而使得其尽可能达到分类要求，并将训练获得的参数生成并封装为配置文件，

从而为移动端所使用。下表 3-3 描述了该用例的用例详述：

表 3-3　用例 2 的用例详述

用例编号	UC.2
用例名称	训练分类器
用例概述	根据所提供的样本集数据，训练卷积神经网络，并保存配置文件。
参与者	用户
前置条件	用户已生成样本集数据。
基本流程	1. 用户将样本集数据重命名为 inputdata。 2. 用户建立一个工作目录，并把所需的 python 文件及样本集数据拷贝到该目录下。 3. 用户在该工作目录下建立一个命名为 property.txt 的文件，并根据卷积神经网络的结构信息填写该文件（有多少个层，每层的类别、节点数等）。 4. 用户打开命令提示符，并通过“cd”命令定位到该工作目录下。 5. 用户在命令提示符中输入执行 python 程序的指令，对该组样本集生成样本集数据。 6. 命令提示符显示出该组训练的情况（迭代数，错误率等），并将训练所得的配置文件打包保存为该目录下的 data 文件。 7. 完成。
扩展点	1-3a. 提示用户权限不够。 1. 请用户在有权限的目录下进行操作。 4a. 提示无法进入路径。 1. 请根据命令提示符的错误提示进行操作。 5a. 程序执行出错。 1. 请确认环境中安装了 python、numpy 和 theano，如果未安装请安装。 2. 请确认命令提示符拥有该目录的写入权限。 3. 请确认 property.txt 的层与层之间关系配置正确，且按规

	则填写。
后置条件	从移动端下载生成的配置文件。
补充说明	第 6 步的训练速度往往较慢，如果环境中有带有 cuda 处理核的显卡则可以大大提高性能。

3.2.3 使用分类器分类

该样例为本系统在移动端上的核心工作。通过样例 3、样例 4 我们已经在移动端下载了分类器的配置文件，且拍摄或选取了将要进行分类操作的图片。接下来就是本系统的终极目标，对图片数据进行分类标注。使用之前训练好的卷积神经网络参数，将待分类的图片数据当做输入，根据神经网络最终层的激活值来决定分类结果。下表 3-4 描述了该用例的用例详述：

表 3-4　用例 5 的用例详述

用例编号	UC.5
用例名称	使用分类器分类
用例概述	调用下载得到的配置文件，对输入数据进行分类标注。
参与者	用户
前置条件	用户已下载配置文件，用户已导入图片数据。
基本流程	1. 用户打开移动端 APP。 2. 用户选择将要使用的配置文件。 3. 用户选择将要进行标注的图片。 4. 点击标注按钮。 5. 用户在经过等待后，查看得到的标注信息。 6. 完成。
扩展点	1a. APP 崩溃。 1. 请重新安装移动端 APP。 2a. 提示加载失败。 1. 请尝试重新下载配置文件，见 UC.3。

	4a. APP 崩溃。 1. 请重新启动设备再尝试。
后置条件	无。
补充说明	用户需安装移动端 APP 并配置完成。

3.3 领域模型分析

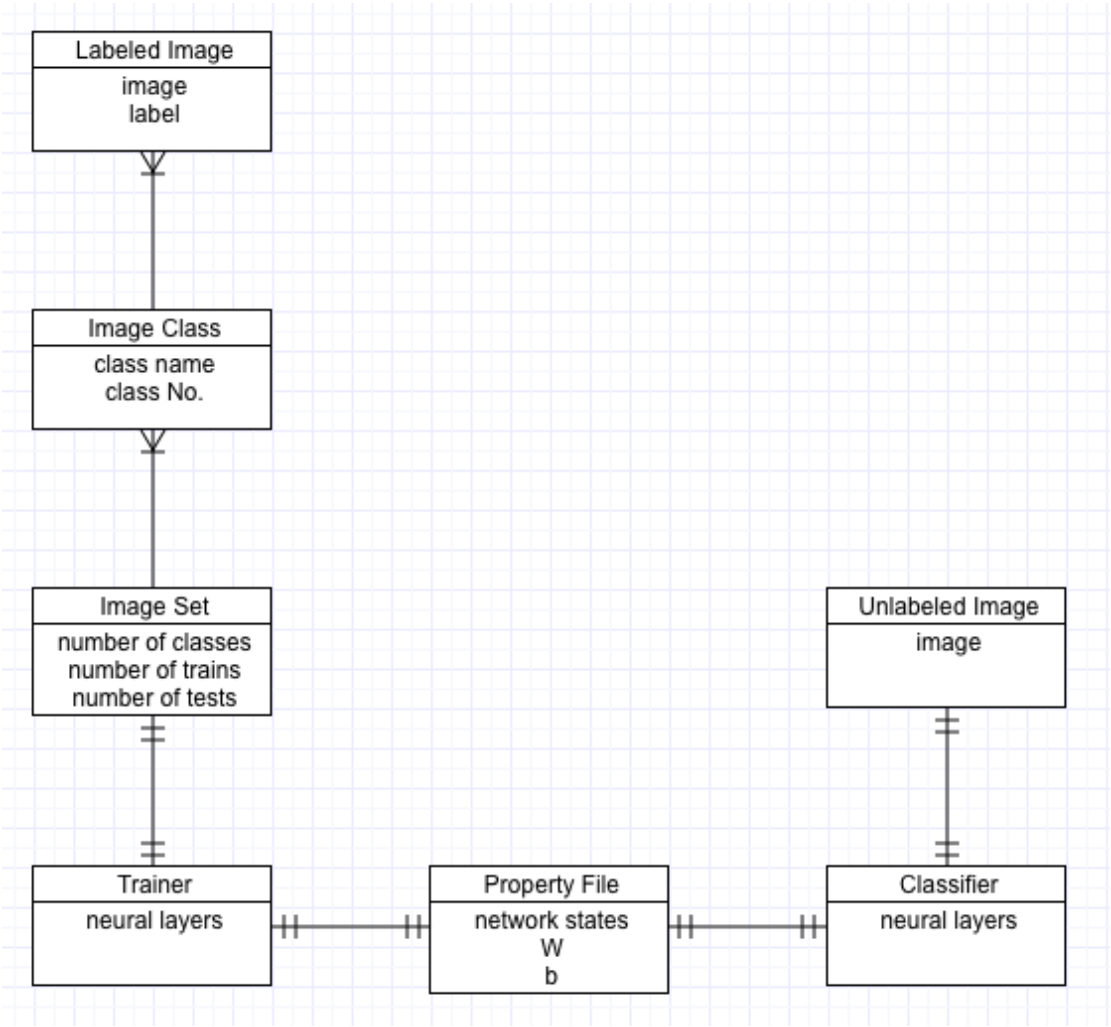

图 3-3　领域模型

由于本系统中出现的概念和组成元素之间比较松散，下面将以可视化的形式描述该系统中的各个要素以及它们之间的联系。系统的领域模型如图 3-3 所示。PC 端的结构主要如下：每一个图片集由若干个图片分类组成，一个图片分类又

由若干个同类别的带标注的图片组成。经过训练器的训练，每一个图片集又对应一个训练好的神经网络，该神经网络又对应一个配置文件。移动端的结构主要如下：一个分类器可以包含若干个配置文件，再分别载入了不同的配置文件时，分类其表示了不同的神经网络。移动端又包括了若干未标注的图片以供标注。

3.4 卷积神经网络的数据流分析

卷积神经网络的算法是本系统的最核心所在，其结构表示、运算规则和训练方法都是本系统中最复杂的组成部分。在卷积神经网络中存在着四种不同的神经元层，他们之间互相连接互相影响，从而得到了最终的分类结果。下面我们用可视化形式的数据流图来表述一下卷积神经网络中的数据流向关系。

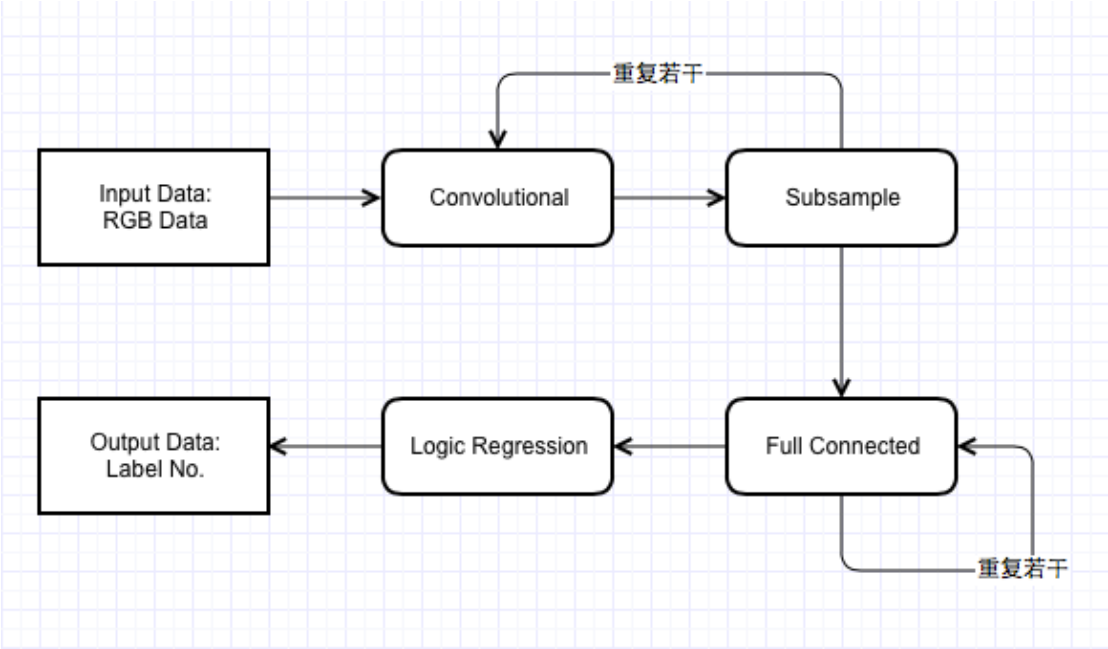

图 3-4　卷积过程的数据流图

由图 3-4 所示，首先输入的数据流入卷积层，获得卷积层的激活值。其后数据由卷积层流入池化层，进行池化缩减数据量。其后经过了若干个卷积层和池化层的连接后，数据流入了全连接层，使用参数矩阵计算该层神经元的激活值。而又经过了若干个全连接层的连接后，最终流入了逻辑回归层，输出了判断为各个分类的概率。

3.5 本章小结

本章从系统概述、用例分析、领域模型分析和卷积神经网络的数据流分析等多个方面对系统所要实现的功能进行了分析和阐述。下一章将开始进行系统的总体设计，并对各个功能分开相应的模块，以及对每个模块进行初步设计，并找到合适的解决方案。

第四章 系统架构与总体设计

本章和下一章是本文的主要贡献，即设计和实现了一套完整的基于深度学习方法的训练计算和分类计算分离开的多平台关键字标注系统。通过前一章对系统各个方面的需求分析，确立了系统所要完成的总体功能。本章承接上一章的需求分析，对系统的总体设计方案进行介绍。首先对系统的总体架构设计进行介绍，接下来根据之前的用例进行主要用例对应的实现，之后设计了系统的静态结构，如系统包、各模块的类组成等等，最后设计了在本系统中 PC 端和移动端连接起来所用的配置文件的组成结构。

4.1 系统架构设计

本系统属于 C/S 架构，但又与典型的 C/S 架构略有不同。其中客户端和服务端的功能相对独立，服务端中包括生成训练样本数据和使用样本集进行训练两部分功能，客户端中包括从服务端下载配置文件和在本地使用配置文件进行标注两部分功能。在这个过程中，服务端与客户端互相有交流的仅仅为下载配置文件部分。本系统的架构设计图如图 4-1 所示：

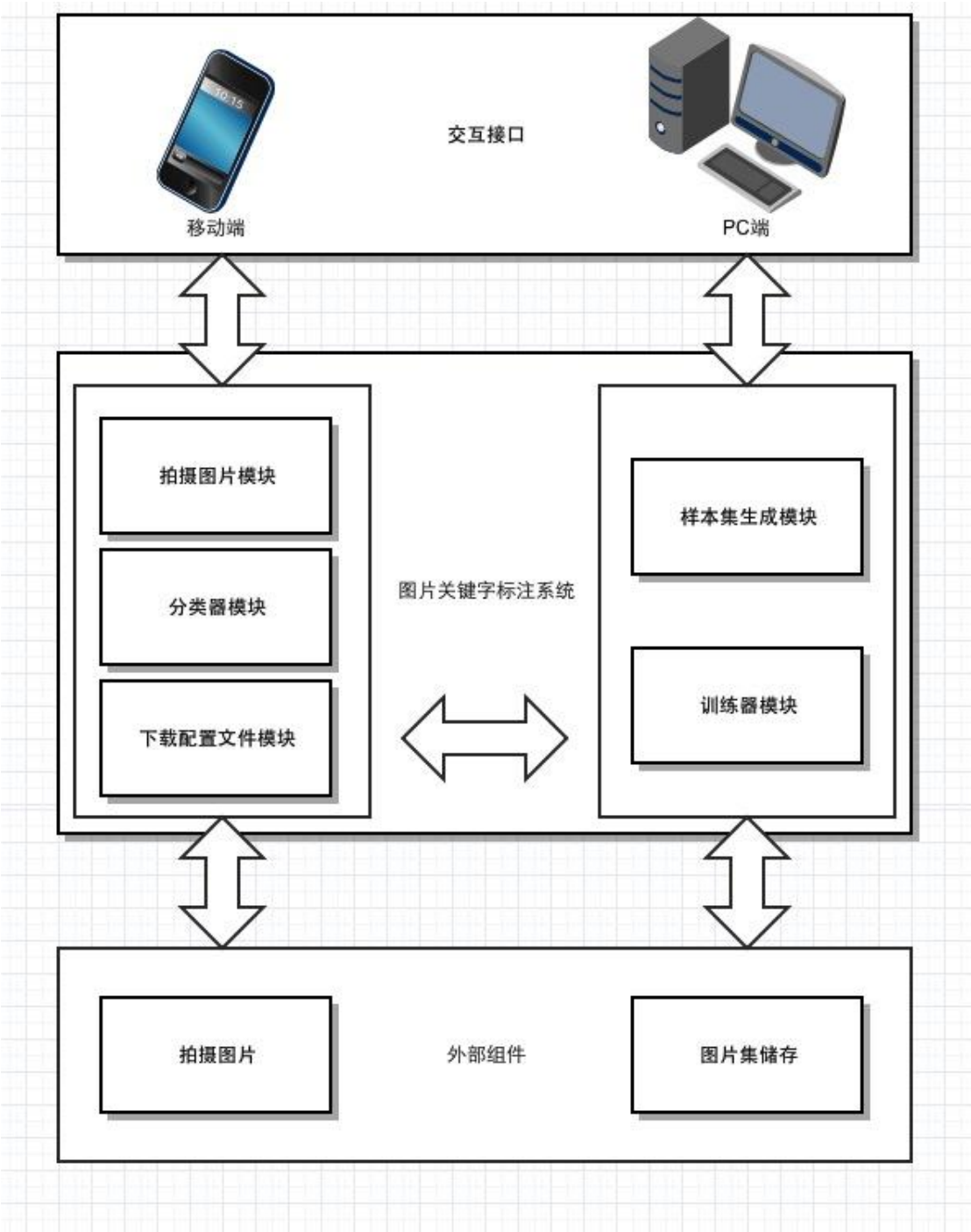

图 4-1　架构设计图

在此基础上，下面对作为服务端的 PC 端与作为客户端的移动端进行分别介绍。

PC 端包含两个模块：样本集数据生成模块和分类器训练模块

样本集数据生成模块，其主要目的为将用户整理好的样本集图片进行预处理，并打包成为 python 可载入的 cpickle 数据文件。其中预处理主要指以下一系列操作：缩放、通道调整等，将图片的像素值规范化到 0-1 的实数域，并转化为三维

矩阵形式的数据（第三维为通道）。该模块处理的主要流程为：1. 读取目录下的property.txt 配置文件，将其中的分辨率、通道设置读取出来；2. 依次扫描各子目录，对每个目录设定一个唯一的分类编号，以 0 开始计数；3. 依次扫描各目录下的图像文件，对图像进行预处理，并将预处理后的数据及相应的分类编号分别存储于两个数组中；4.将最后所得的两个数组进行元素对应的乱序操作，并以cpickle 格式保存为 data 文件。

分类器训练模块，主要基于卷积神经网络，实现一个包括卷积层、池化层、全连接层、逻辑回归层的深度神经网络。其中包括各层神经网络激活值的计算和传导，以及使用反向传播算法和梯度下降法对神经网络的参数进行训练的功能。其中训练的大体流程如下：1. 首先读取目录下 property.txt 文件和样本及文件inputdata；2. 根据神经网络结构配置，构建神经网络的参数矩阵，并进行初始化；3. 使用一批样本集中的数据进行激活值计算，并根据结果计算当前错误率；4. 根据激活值和最终生成的分类结果，计算各层参数的梯度；5. 将梯度应用于神经网络各层参数上，进行调整；6. 重复以上 234 步，直至错误率达到要求或已近似收敛；7. 将神经网络的配置以及各层参数的结果保存，并打包封装为 data。

1.根据本地图片生成训练所需的样本数据集

2.使用样本数据集训练 CNNs，并生成配置文件

移动端包含三个模块：下载配置文件模块、拍摄及导入图片模块和分类器分类模块。

下载配置文件模块，其主要目的为从服务端读取配置文件列表，并可从中选择配置文件进行下载。其中主要流程为：1. 向服务端发起一个 http 访问，读取其 list 文件；2. 根据解析 list 文件的内容，获取各个配置文件的储存地址；3. 根据用户选择的配置文件，将配置文件下载至本地；4. 对下载得到的配置文件进行解压。

拍摄及导入图片模块，其主要功能是将即时拍摄的图片或导入的本地图片加入程序中的图片库，并可根据各个分类配置文件对其数据进行提取转换，该提取转换类似于样本集数据生成模块中的预处理操作。其操作指以下一系列操作：缩放、通道调整等，将图片的像素值规范化到 0-1 的实数域，并转化为三维矩阵形

式的数据（第三维为通道）。通过得到这样的数据，从而可以使得直接被分类器模块读取使用。

分类器分类模块，其目的为使用分类器的配置信息，对图片数据进行分类标注。在这里也要实现一个完整的卷积神经网络，即包括卷积层、池化层、全连接层、逻辑回归层的激活值计算。其主要处理流程如下：1. 读取本地选择的配置文件和图片数据；2. 从配置文件中读取神经网络的结构信息，生成一个神经网络并从配置文件中读取各层参数；3. 将图片数据当做输入数据传入神经网络，依次计算各层的激活值并进行传导；4. 根据神经网络最后一层的输出数据，确定该图片的分类标注。

在以上所述的系统架构中，主要有如下特点：

1. 移动端与服务端联系并不紧密，从 PC 端下载配置文件是客户端与服务端唯一有接触的地方。
2. 充分发挥了 PC 端和移动端的各自的优势，PC 端负责大规模的计算，而移动端保持其灵活性，只需在本地快速计算得出结果，不需依赖与 PC 端的网络连接。

以上即为本系统的主要架构设计和其中各项模块的功能。

4.2 用例实现

在上一节中，给出了本系统的整体架构设计，并对各个模块的相应功能及相互联系进行了详细的描述。在本节中，将对第三章中所详述的三个用例进行初步的分析和实现。

4.2.1 PC 端生成样本集数据样例实现

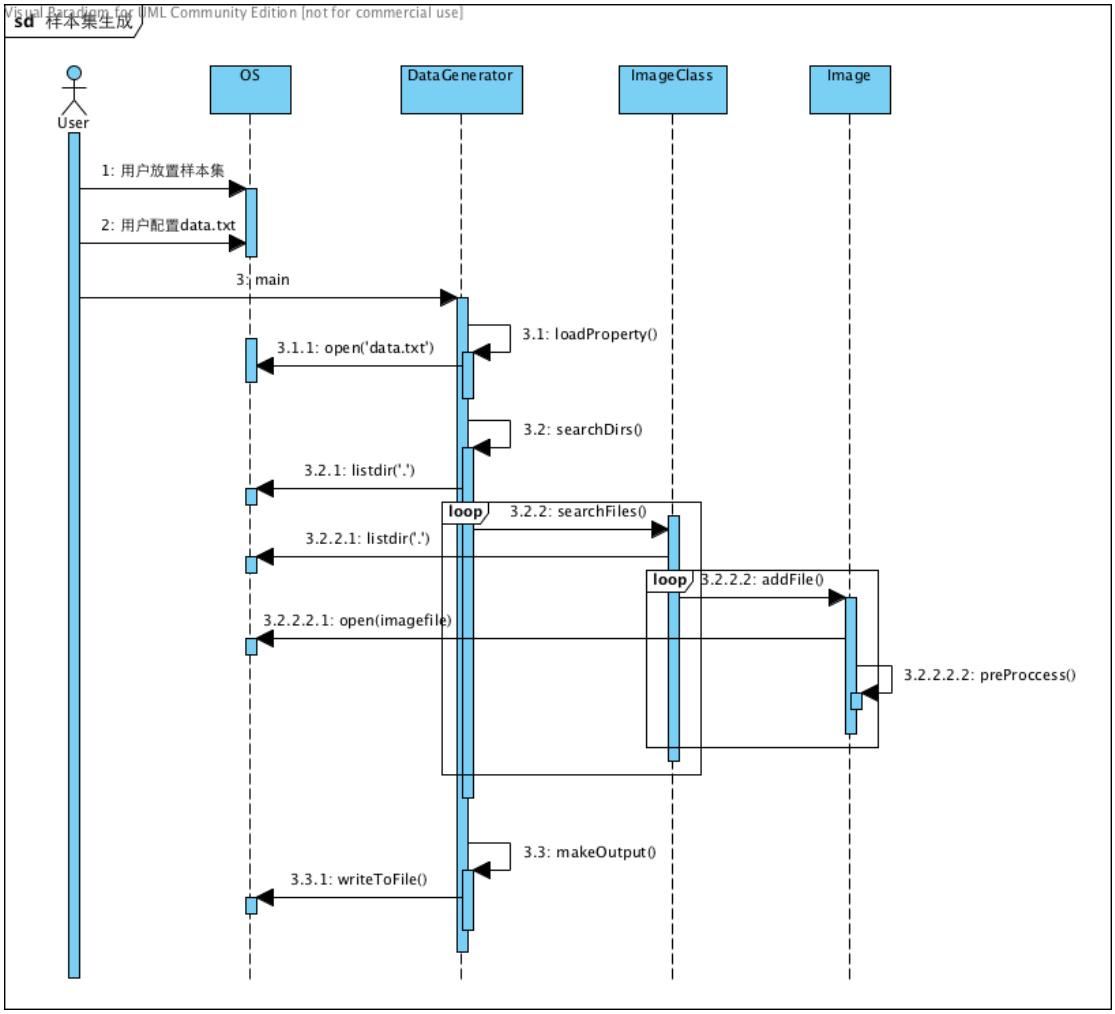

图 4-2　样本集生成序列图

PC 端生成样本集数据样例中，如图 4-2 所示，用户与系统的交互和系统内的交互过程如下：

1. 用户将准备好的样本库按格式放置好。
2. 用户建立并填写 property.txt 配置文件。
3. 用户执行样本集数据生成程序。
4. 程序首先访问 property.txt 配置文件，读取其中的配置信息。
5. 程序接下来向 os 请求当前目录的所有子目录名称，并对每个目录依次编号。
6. 程序开始依次迭代进入每个目录，并向 os 请求该目录下的所有图片名称。
7. 程序依次读取每个图片。
8. 程序对读取到的图片数据进行预处理。

9. 程序对每个图片得到的数据和分类编号，分别保存至内存中的两个数组中。

10. 程序完成所有迭代后，将数组进行乱序，储存至硬盘的 data 文件，终止运行。

4.2.2 PC 端训练分类器样例实现

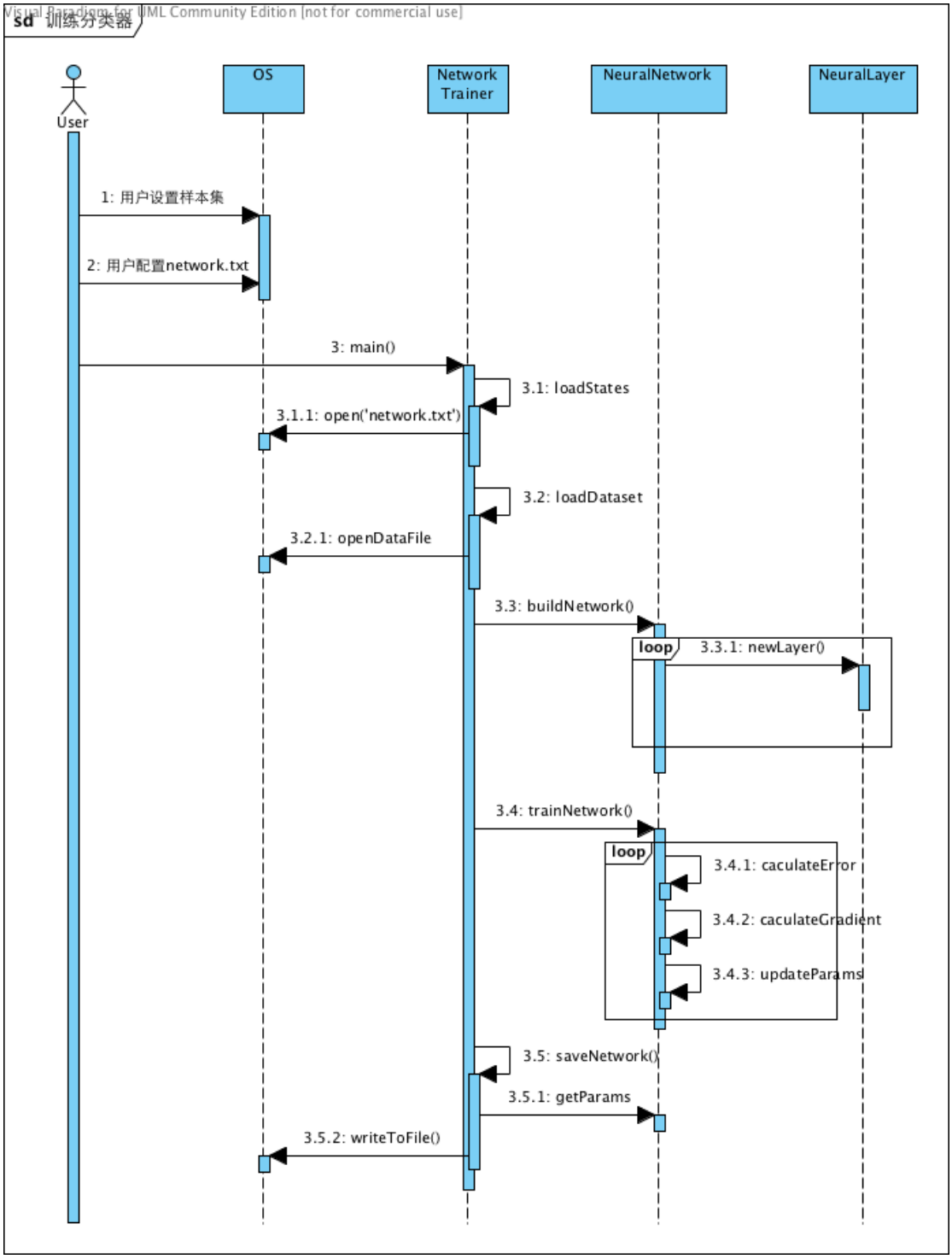

图 4-3　　训练分类器序列图

PC 端训练分类器样例中，如图 4-3 所示，用户与系统的交互和系统内的交互过程如下：

1. 用户将所要使用的样本集数据放置在工作目录下。
2. 用户建立并填写 property.txt 配置文件。
3. 用户执行训练分类器程序。
4. 程序首先访问 property.txt 配置文件，读取其中的配置信息。
5. 程序将样本集数据读入，并将样本集数据分为若干个处理批次。
6. 程序使用配置信息生成一个卷积神经网络，并初始化其中的参数。
7. 程序依次使用一批数据进行神经网络激活值计算。
8. 程序根据神经网络的激活值计算该批数据下的错误率和梯度。
9. 程序将该梯度应用至神经网络。
10. 重复若干上述 7-9 操作，直至达到收敛条件，打包保存神经网络配置和参数，并终止运行。

4.2.3 移动端使用分类器分类样例实现

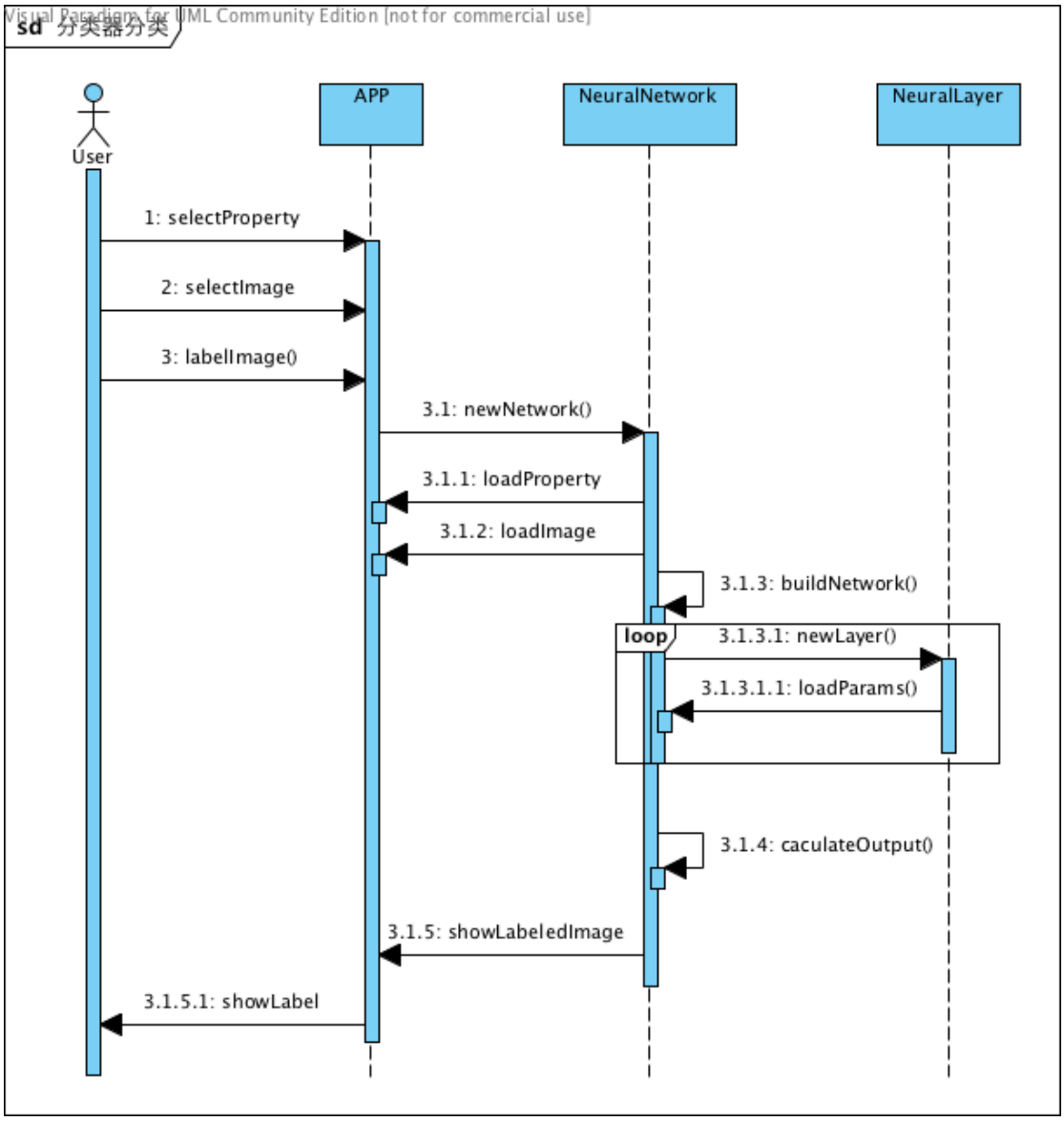

图 4-4　　分类器分类序列图

移动端使用分类器样例中，如图 4-4 所示，用户与系统的交互和系统内的交互过程如下：

1. 用户在列表中选择所要使用的配置文件。
2. 用户在列表中选择所要标注的图片。
3. 用户点击分类标注按钮，调用分类函数。
4. 程序读取配置文件。
5. 程序读取图片数据。
6. 程序生成根据读取的参数初始化卷积神经网络。

7. 程序将图片数据输入神经网络，并计算网络的激活值。
8. 程序根据激活值确定分类标注，并显示给用户。

4.3 系统静态结构设计

根据以上用例中描述的系统需要的功能，本系统可以拆分为若干个子系统，其中包含类较多关系较复杂的有以下三个部分：PC 端的训练器部分，移动端的分类器部分，和移动端的 APP 整体结构。下面分别使用系统包图和类图对系统中的包结构和类之间关系进行直观视觉上的描述。

4.3.1 系统包图

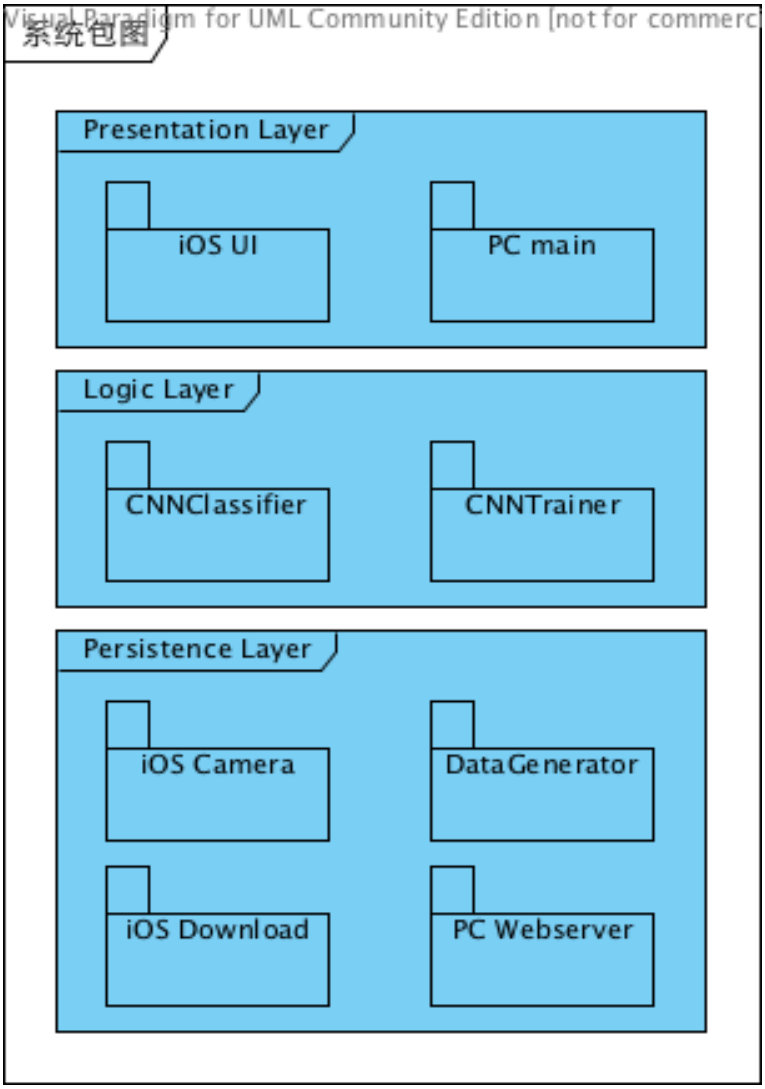

图 4-5　系统包图

如图 4-5 系统包图所示，本系统的 PC 端和移动端，都各主要由三个层组成：表示层、逻辑层、持久层。

在移动端上，这三个层分别包含如下的包：

表示层：跟 UI 交互及 APP 显示相关的类组成的 iOS UI 包；

逻辑层：处理神经网络功能和网络各层逻辑的 CNNClassifier 包；

持久层：用于拍摄和保存图片的 iOS Camera 包和用于下载和读取配置文件的 iOS Download 包；

在 PC 端上，这三层分别包含以下的包：

表示层：各个 Python 类里供系统直接执行程序的各 main 函数组成的 PC main 包，其分散于各个类中。

逻辑层：包括生成和训练神经网络逻辑的 CNNTrainer 包；

持久层：用于处理图片文件生成样本集的 DataGenerator 包，和保存供客户端下载配置文件的 Web Server 包。

4.3.2 训练器类图设计

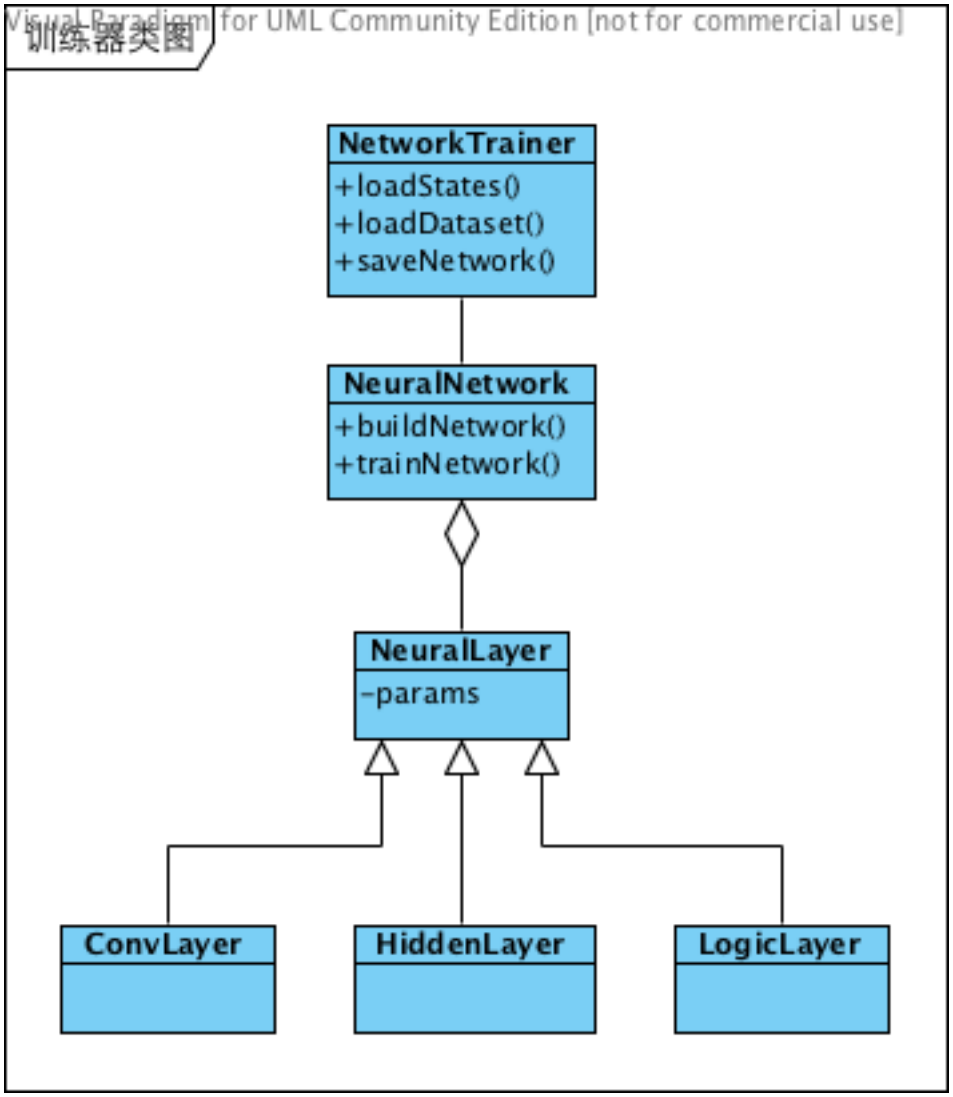

图 4-6　训练器类图

训练器的设计类图如图 4-6 所示，训练器所要实现的功能主要是用例 UC1，根据以上用例实现里的序列图分析，抽象出 6 个类：其中一个用于表示抽象神经网络层的 NeuralLayer 类，三个用于表示不同种类的神经网络层的 ConvLayer、HiddenLayer、LogicLayer 类，一个用于表示组合了的神经网络的 NeuralNetwork 类，和一个表示神经网络训练起的 NetworkTrainer 类。

其中每一个神经网络层包含一定的参数 params 作为实例变量，神经网络有建立网络和训练网络两个功能的函数：buildNetwork()和 trainNetwork()，神经网络训练类有读取神经网络配置、读取样本集和保存神经网络参数的功能。

4.3.3 分类器类图设计

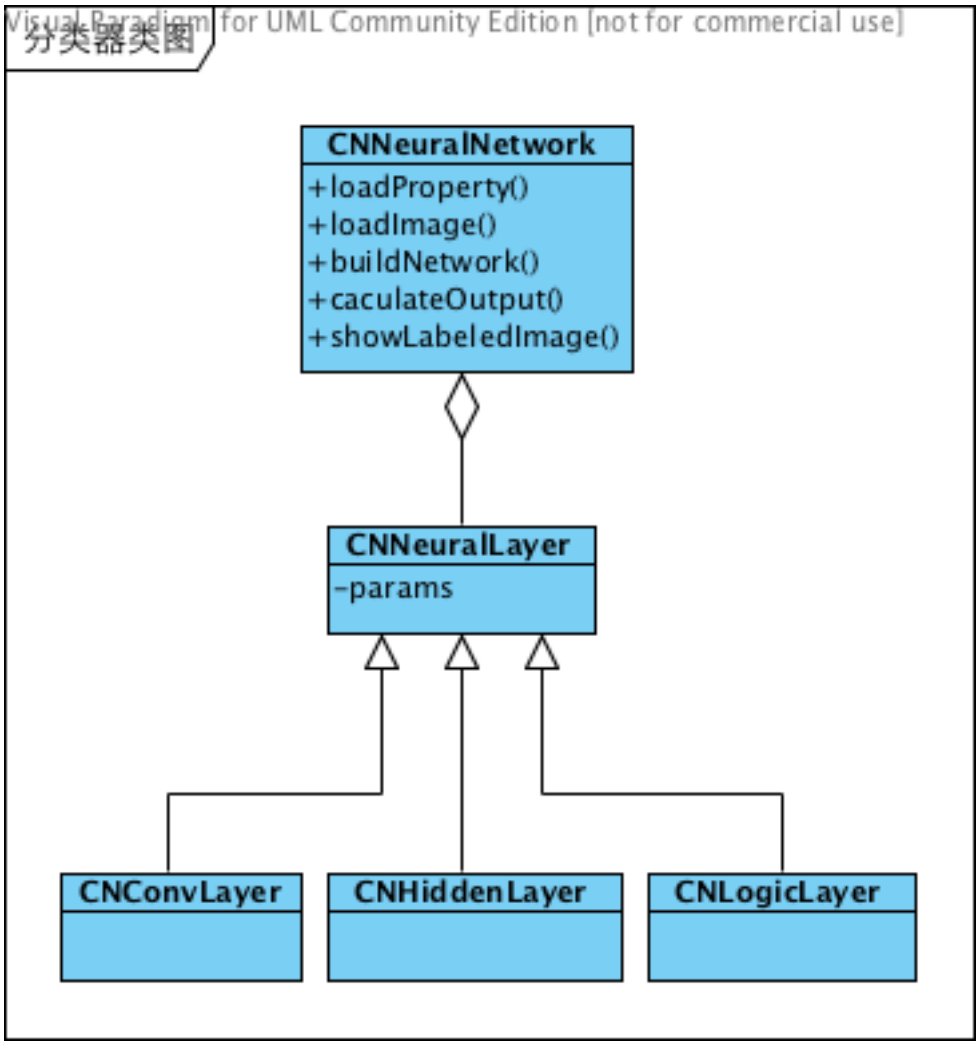

图 4-7　　分类器类图

分类器的设计类图如图 4-7 所示，分类器所要实现的功能主要是用例 UC5，根据以上用例实现里的序列图分析，与上一节的训练器类似，该部分功能主要抽象出 5 个类：用于表示抽象的神经网络层的 CNNeuralLayer、三个用于表示各种不同的神经网络层的 CNConvLayer、CNHiddenLayer、CNLogicLayer，以及一个用于表示整个神经网络的 CNNeuralNetwork 类。

其中每一个神经网络层包含一定的参数作为 params 变量，神经网络有读取配置、读取输入图像、构建网络、计算输出值和输出标注结果等功能。

4.3.4 移动端 APP 类图设计

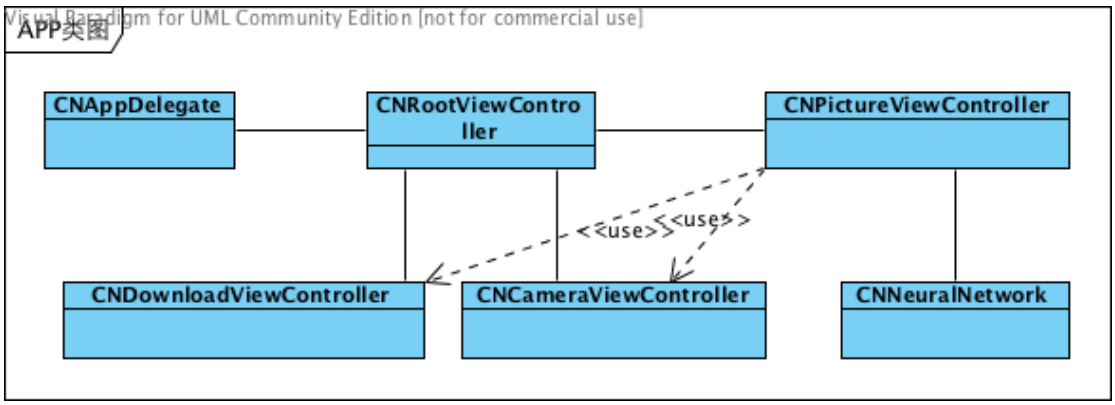

图 4-8　　APP 类图

移动端 APP 的设计类图如图 4-8 所示，移动端 APP 所实现的功能主要是用例 UC3 和用例 UC4，根据对用例的分析，根据其主要功能和职责，抽象出 4 个类：用于 APP 主界面的 CNRootViewController、用于下载配置文件的 CNDownloadViewController 、用于进行拍照和导入图片的 CNCameraViewController、和用于分类和显示结果的 CNPictureViewController。

其中 CNRootViewController 使用了另外两个页面，显示结果的类 CNPictureViewController 使用了类 CNNeuralNetwork。

4.4 分类器配置文件结构设计

在本系统中，分类器的配置文件作为服务端与客户端之间唯一的连接点，起着非常重要的作用。其中需要包括关于样本集和卷积神经网络各方面的信息，才能在客户端完整的重现出服务端训练完成的神经网络，从而用于分类标注计算。而由于这里需要服务端与客户端之间进行传输，如果包含了过多的冗余信息，会使传输速率过慢。因此，此处配置文件的设计需要谨慎且完整。

经过分析卷积神经网络的结构和原理，其中配置文件主要需要包括以下各部分：

1. 卷积神经网络层数。
2. 神经网络每层的类型。

3. 神经网络每层的输入连接数、输出连接数、参数数目及组成结构。
4. 神经网络每层的激活函数。
5. 神经网络每层的具体参数。
6. 样本集数据的尺寸和通道数。
7. 样本集的分类数，以及每类的名称。

因此组成结构主要为：

1. network.txt 文件，表示了神经网络的层数，以及各层的配置信息。
2. class.txt 文件，表示了各分类编号所对应的标注的名称。
3. layerXparaW.txt 文件，表示了第 X 层神经网络的 W 参数。
4. layerXparab.txt 文件，表示了第 X 层神经网络的 b 参数。

将以上各文件使用 zip 压缩打包为 zip 文件。

4.5 本章小结

本章中对本系统的整体设计进行了描述，首先介绍了系统的整体架构，然后采用序列图对上一章中分析的重要用例进行了用例实现，之后分别使用系统包图和类图对系统的主要组成结构进行了全面的介绍，最后对本系统中比较重要的分类器配置文件的结构组成进行了设计。下一章中将对本系统进行更加细致的设计以及同时介绍系统的实现。

第五章 系统模块详细设计与实现

上一章对本系统的整体结构进行了分析和设计，本章中将接下来实际对本系统进行更加详细的设计以及实现。本章主要以各个关键模块为单位进行组织，从样本集数据模块、分类器训练模块、移动端分类器分类模块、移动端 APP 等方面分别进行详细描述。

5.1 样本集数据文件生成器的详细设计与实现

样本集数据文件的生成是神经网络训练的前置工作，需要从用户提供的普通的已分类的图片文件中将图片数据进行简单地预处理，转换并提取出来，并根据用户配置的训练集和测试集各占的百分比，使其分别存入六个数据文件中以供以后使用。该模块的主要功能由 DataGenerator 类来实现。

如图 5-1，其主要处理流程如下：

1. 从文件中读取 property 信息。
2. 向系统获取 data 目录下的子目录列表，并初始化训练集和测试集相应的 6 个数组。
3. 迭代进入每一个子目录，为其分配相应的分类编号。
4. 向系统获取当前子目录下的图片文件列表，并初始化两个临时数组用以暂时保存该类

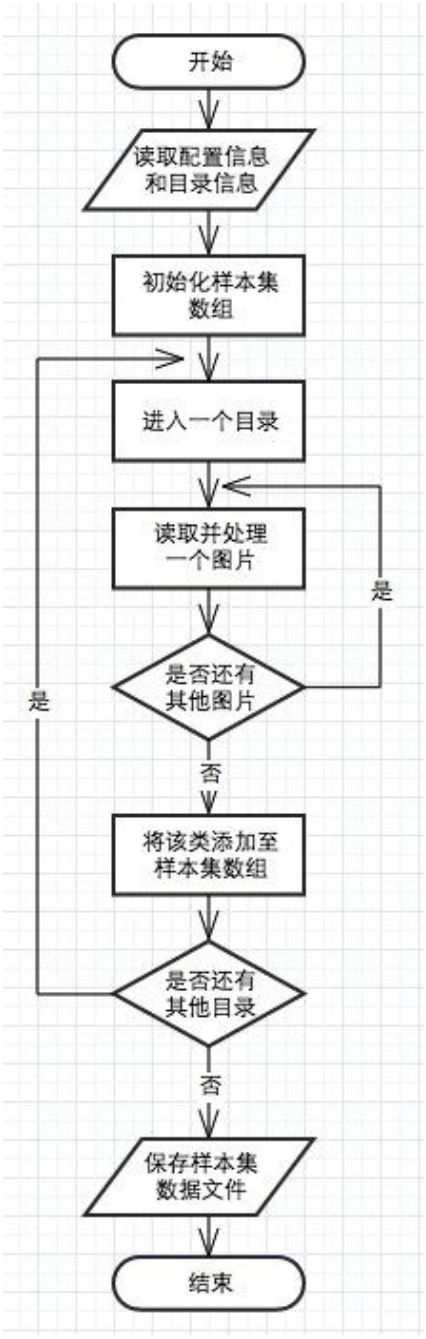

图 5-1　样本集数据生成的流程图

的数据。

5. 迭代读取每一个文件，并根据其通道数进行图像色彩空间转换，之后根据其分辨率进行缩放处理，并将其 0-255 的像素值转换至 0-1 的实数，得到该图片的数据。
6. 将得到的图片加入临时数组，当迭代完该类别所有图片时，得到该类的数据数组。
7. 对该类的数据数组进行乱序，并根据 property 所定义的训练集和测试集的百分比，将该类数据数组分拆加入相应的各个数组中。
8. 当迭代完所有分类时，即得到了该组样本集的训练集和测试集数据，对其分别进行训练，写入文件。

类 DataGenerator 包含以下主要函数，如表 5-1：

表 5-1　DataGenerator 类的主要函数

函数名	函数功能
loadProperty()	从文件“property.txt”中读取样本集的配置信息
searchDirs()	查找“data”目录下的所有子目录，并各自给其分类标号
searchFiles()	查找当前子目录下的所有文件，并将数据分类保存至数组
addFile()	读取当前图片，并对其进行预处理，转化为图片数据数组
addToLibrary()	将给定数组的数据根据百分比配置分别加入各全局数组
makeOutput ()	将全局数组中保存的数据写入文件输出
main()	类的主函数，通过其调用其它函数

类 DataGenerator 还包含了一些辅助函数，如表 5-2：

表 5-2　DataGenerator 类的辅助函数

函数名	函数功能
addToArray()	将给定的数据和分类标号加入给定数组
shuffleArray()	将给定的数组乱序排序
writeToFile ()	将给定的数组写入给定名称的文件

样本集的配置文件“property.txt”中，主要包含以下信息，如表 5-3：

表 5-3　dataGenerator 类的变量信息

变量名	用途
imageSize	缩放后的图片宽度和高度（正方形图片）
channel	图片的通道数（3 表示 RGB，1 表示灰度）
trainPercent	得到的数据文件中训练集占得半分比
validPercent	得到的数据文件中测试集占得半分比

5.2 基于卷积神经网络的训练器详细设计与实现

卷积神经网络的训练是作为服务器的 PC 端的最核心模块，并且也是全系统中计算量最大的部分。在此处性能的处理至关重要，并且会占用极高的 CPU 资源和内存资源（使用 gpu 加速时，也占用 cuda 核心和显存资源）。该部分主要是实现出卷积神经网络的训练算法，因此在这里我们分结构上和算法上分别对该模块进行详述。

5.2.1 训练器的结构

该模块的类图如图 5-2：

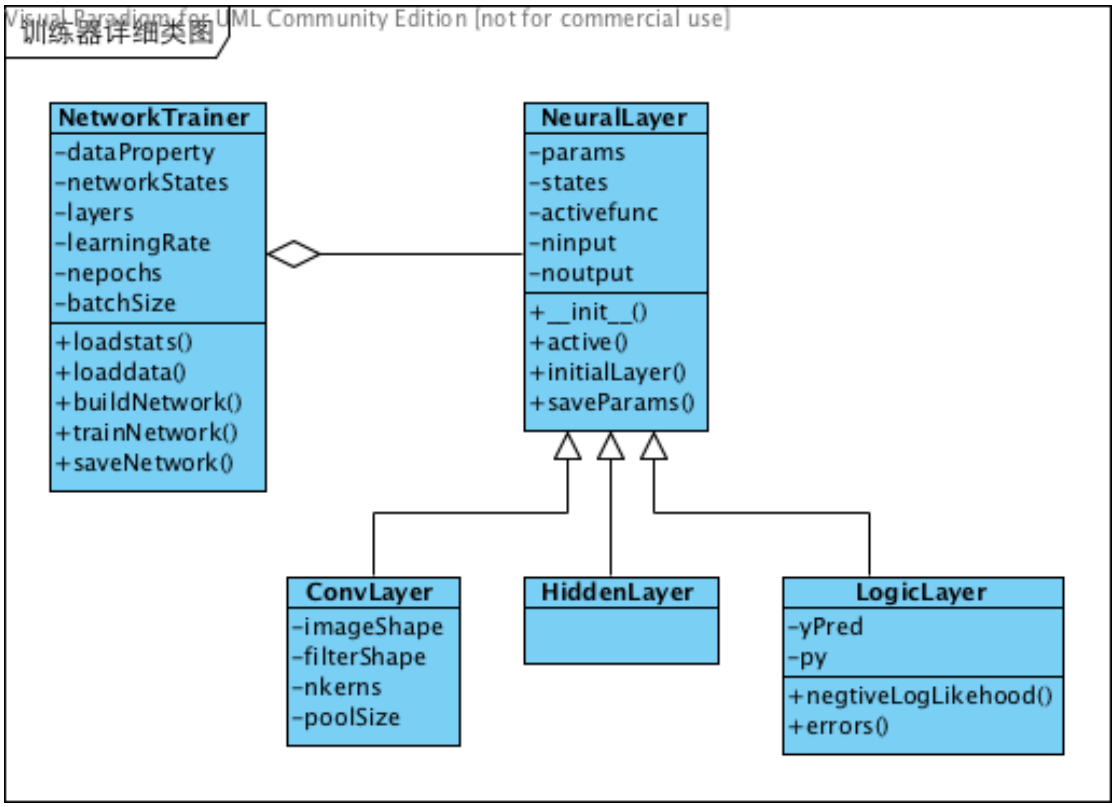

图 5-2　　训练器详细类图

该模块主要由 5 个类组成，其中类 ConvLayer、类 HiddenLayer 和类 LogicLayer 分别表示卷积层、全连接层和逻辑回归层，NeuralLayer 类表示 ConvLayer、HiddenLayer 和 LogicLayer 类的抽象父类，NetworkTrainer 类表示一个整个神经网络系统。我们在这里着重分析父类 NeuralLayer 和表示整个神经网络系统的 NetworkTrainer 类。

NeuralLayer 类抽象表示着神经网络的一层，因此其的主要功能就是计算该层的激活值，以及为了训练的原因，计算该层当前的梯度。NeuralLayer 类中又包含了一些变量，用以表示神经网络该层的各项参数，也就是该层神经的记忆。由于神经网络的一层一般表示着一个基于线性变换的非线性变换，因此其参数的主要组成为转换矩阵 W，转换后的偏移量 b，以及一个非线性函数 s。该层中还包括了一些函数，主要用于计算便利以及初始化所用。

对于神经网络的一个层，有以下的主要函数，如表 5-4：

表 5-4　　NeuralLayer 的主要函数

函数名	函数功能
__init__()	神经层类的构造函数，用于将传递来的变量赋值给实例变

	量，以及调用 initialLayer 函数
initialLayer()	初始化该层，在每个继承的子类中都有各自不同的实现
active()	根据本身的 activefunc 实例变量，反应该层相应的激活函数
saveParams()	保存该层的参数值，在各类中实际的功能不同

对于神经网络的一个层，又包含以下变量作为其参数，如表 5-5：

表 5-5　NeuralLayer 的变量信息

变量名	用途
params	该层包含的参数，不同类中保留的形式不同
states	是一个字典，保存着该层相关的信息，在不同类中条目不同
activefunc	该层激活函数的名字
ninput	该层一个样本输入的节点数
noutput	该层一个样本输出的节点数

下面使用表 5-6，来分别列出在卷积层、全连接层和逻辑回归层，params 变量都保存了哪些参数：

表 5-6　params 在各层的意义

类名称	params 包含的内容
ConvLayer	该层各个卷积核的 W 和 b，W 为 4 维而 b 为 1 维，W 具体形状与卷积核形状与卷积核数量有关
HiddenLayer	该层的转换矩阵 W 和 b，W 为 2 维而 b 为 1 维，其中 W 行数等于输入节点数，列数等于输出结点数
LogicLayer	该层的转换矩阵 W 和 b，W 为 2 维而 b 为 1 维，其中 W 行数等于输入节点数，列数等于要分的类数

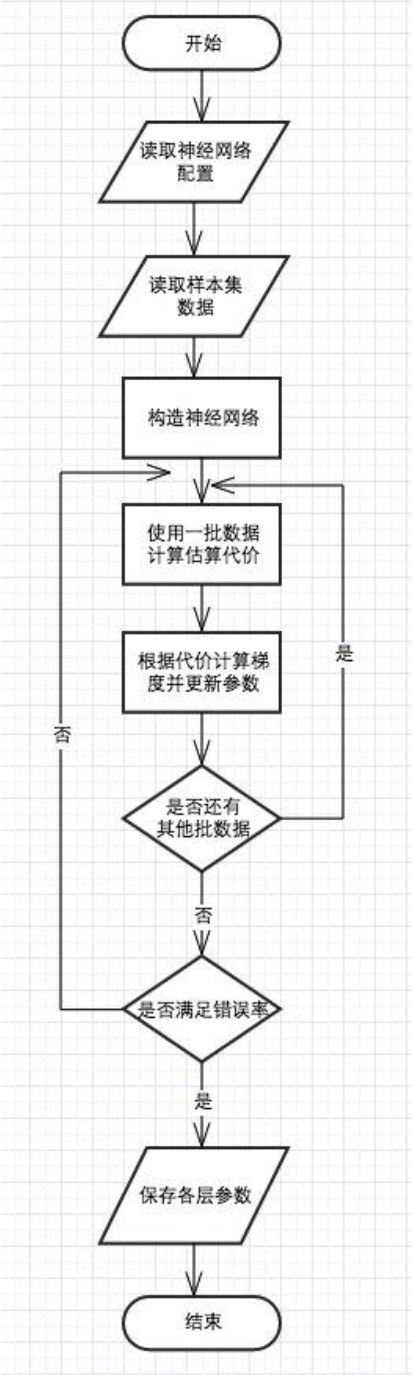

图 5-3　分类器训练的流程图

NetworkTrainer 类作为表示整个神经网络的类，包含了一些以 NeuralLayer 为基类的表示神经层的类，因此其主要功能是负责对各个层的训练。由于其又是该模块的根类，因此 NetworkTrainer 类也负责读入配置、读入数据、构建及初始化神经网络、输出保存各层参数等很多其他相应功能。

如图 5-3，NetworkTrainer 类的主要工作流程如下：

1. 从文件中读取输入文件配置信息。
2. 从文件中读取神经网络配置信息。
3. 读取样本集数据文件，并将训练集数据根据神经网络配置文件中给定的批次大小分批次。
4. 根据神经网络配置信息，构建神经网络并初始化参数初值。
5. 使用一批数据文件，计算神经网络的激活值和各层各参数的梯度。
6. 使用该梯度和神经网络配置文件中设定的步长更新神经网络的各参数。
7. 计算当前情况下的验证错误率。
8. 重复 5-7，直至错误率达到要求，停止计算。
9. 将神经网络各层的参数和配置文件一同打包，保存为 zip 文件。

类 NetworkTrainer 包含以下主要函数，如表 5-7：

表 5-7　NetworkTrainer 的主要函数

函数名	函数功能
loadstats()	读取该神经网络的配置信息

loaddata()	读取该次训练所用的样本集数据
buildNetwork()	逐层建立神经层，并组成神经网络
trainNetwork()	根据配置设置训练神经网络
saveNetwork()	保存神经网络中的各项参数信息并打包

类 NetworkTrainer 中的神经网络配置文件主要包含以下信息，如表 5-8：

表 5-8　NetworkTrainer 的变量信息

变量名	用途
dataPreperty	该次训练所用的样本集数据的信息
networkStates	为一个字典，保存着整个神经网络的各项属性信息
layers	为一个数组，保存着该神经网络中的各神经层
learningRate	学习速率，为每次更新时所乘的系数
nepochs	训练最多使用全部数据的次数
batchSize	每一批次训练图像的数量

5.2.2 训练算法的实现

深层神经网络使用梯度下降法来进行参数值的更新和训练，训练器中所涉及到的算法主要有两个，一个是根据输入数据计算各层的激活值，另一个是根据最终层输出数据和样本的比对，计算各层个参数的梯度。其中最重要和困难的工作就是计算各层各参数的梯度。本处使用反向传播算法来计算各层各参数梯度。

对于激活值的传导顺序如图 5-4 所示：

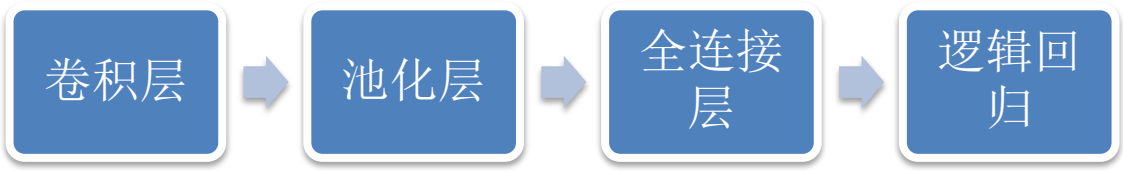

图 5-4　卷积神经网络的激活值传导

激活值的传播计算过程如下：

1. 读取输入数据矩阵。

2. 从前往后依层将数据矩阵进行神经网络的传导计算，卷积层为卷积计算，其他层为矩阵计算。
3. 将计算得到的结果上加上偏差值，并对每个数据应用非线性函数，得到下一层的输入数据。
4. 依层计算直至最后一层，输出值即为整个神经网络的输出值。

由其直观表示，激活值的计算也成为前向传导计算。

对于反向传播算法，主要示意图如图 5-5：

图 5-5　反向传播算法

反向传播算法的主要计算过程如下：

1. 计算整个神经网络的激活值。
2. 根据最后一层的激活值计算出预估结果，与样本给定的结果一起计算出逻辑回归层的梯度。
3. 从最后一层往前，依次计算各层各参数的残差值。
4. 针对各层的激活值和残差值，加权计算出各层各参数的梯度。
5. 使用该梯度更新参数，从而训练神经网络。

5.3 基于卷积神经网络的分类器详细设计与实现

该模块为移动端的核心功能，根据所训练出的神经网络的配置文件，重现出其对应的神经网络，并将其用于分类标注。由于该处所要实现的功能即是前一节

中所述神经网络的激活值计算的功能，因此与上一节所述 PC 端的分类器结构类似。

由于该模块需要在移动端执行，因此与 PC 端使用了完全不同的开发环境、工具和语言，将模块实现为以移动端处理性能最优为目的的形式。该节实现除结构上类似、配置文件定义相同外，其他均为完全不同的实现。

该模块的类图如图 5-6：

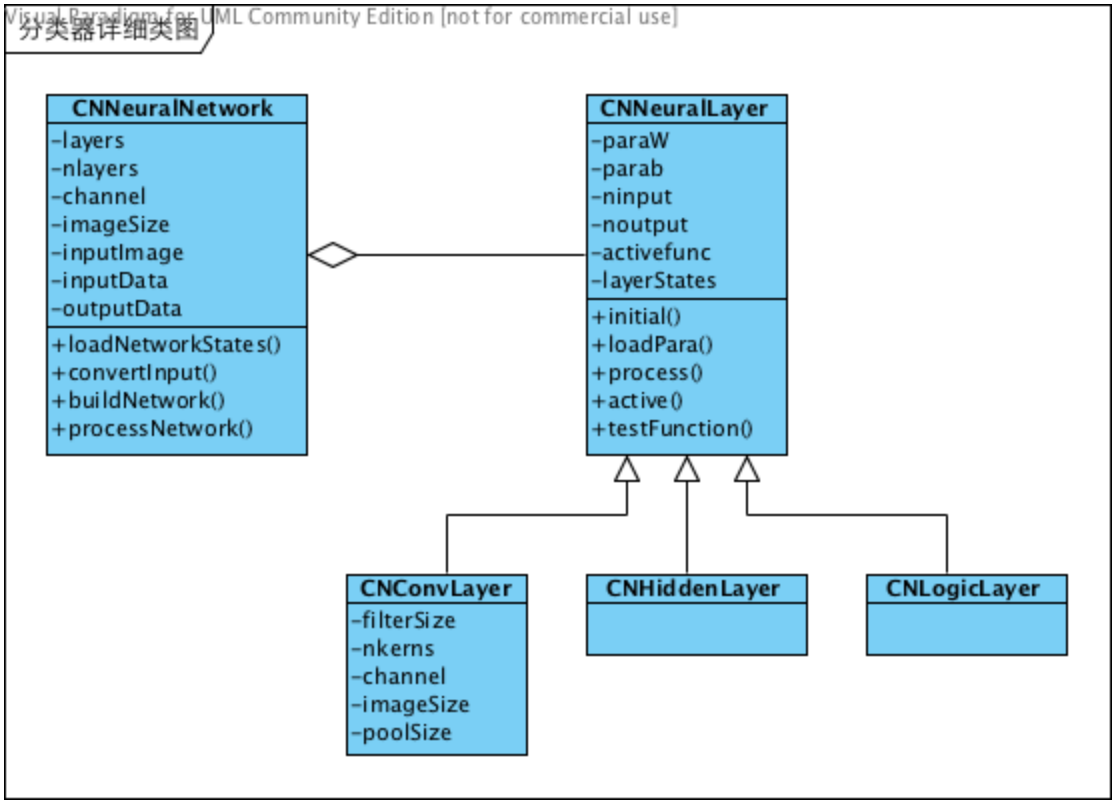

图 5-6　分类器详细类图

该模块与上一节类似，主要由 5 个类组成：类 CNConvLayer、类 CNHiddenLayer 和类 CNLogicLayer 分别表示卷积层、全连接层和逻辑回归层，CNNeuralLayer 类表示 CNConvLayer、CNHiddenLayer 和 CNLogicLayer 类的抽象父类，CNNeuralNetwork 类表示整个神经网络系统。我们在这里也着重分析父类 CNNeuralLayer 和表示整个神经网络系统的 CNNeuralNetwork 类。

在这里的 CNNeuralLayer 类，比起上一节中所述的功能有所缩减，在这里只需要计算激活值而不需要计算梯度值。但是又比起上一节有一个新的需要的功能，即是根据配置文件读取重现出该层的状态。

类 CNNeuralLayer 主要包含以下函数，如表 5-9：

表 5-9　CNNeuralLayer 的主要函数

函数名	函数功能
initial()	初始化该神经层，并从 layerStates 中读出该层的配置信息
loadPara()	读入该层训练好的各项参数
process()	计算该层神经网络的激活值
active()	该层使用的激活函数

类 CNNeuralLayer 包含以下主要变量作为参数，如表 5-10：

表 5-10　CNNeuralLayer 的参数信息

变量名	用途
paraW	保存着该层神经网络的 W 参数
parab	保存着该层神经网络的 b 参数
ninput	表示该层神经网络输入元的个数
noutput	表示该层神经网络输出元的个数
activefunc	该层所使用的激活函数的名称
layerStates	为一个字典，保存着该层神经层的各项信息

CNNeuralNetwork 类的功能比起上一节所述也有所缩减，不需要调度神经网络的训练，只需要计算其输出结果。也比上一节多了一个新的功能，即是调度各层参数的读取。

类 CNNeuralNetwork 的工作流程如下：

1. 将配置文件解包。
2. 读取数据配置文件。
3. 读取神经网络配置文件。
4. 调度各层读取各自相应的参数。
5. 读取输入数据给予输入层。
6. 依次依层计算输出值。
7. 根据最后一层的输出值计算分类标注结果。

类 CNNeuralNetwork 包含的主要函数如下，如表 5-11：

表 5-11　CNNeuralNetwork 的主要函数

函数名	函数功能
loadNetworkStates()	读取该神经网络的配置信息
convertInput()	将 UIImage 转化为可用于神经网络的输入数据
buildNetwork()	构造神经网络，并读取相应各层的参数
processNetwork()	使用输入数据计算该神经网络的激活值

类 CNNeuralNetwork 还包含了一些辅助函数，如表 5-12：

表 5-12　CNNeuralNetwork 的辅助函数

函数名	函数功能
reSizeImage()	将图像缩放为合适的尺寸
getRawFromImage()	从图像中获取 RGBA 像素值数组
getRGBChannels()	从像素值数组中提取出 RGB 数组，并归一化到(0,1)
getGreyChannel()	从像素值数组中计算出灰度数组，并归一化到(0,1)

类 CNNeuralNetwork 下所包含的变量为，如表 5-13：

表 5-13　CNNeuralNetwork 的变量信息

变量名	用途
layers	表示保存着该网络各层的数组
nlayers	表示该网络拥有的层数
channel	表示该神经网络输入数据的频道数
imageSize	表示该神经网络输入图像数据的尺寸
inputImage	保存着神经网络的输入图像
inputData	保存着输入图像转化后的输入数据
outputData	保存着经过网络后计算得到的输出值

5.4 基于移动端的 APP 的详细设计与实现

该模块为该系统最后成品的表示层，包括了完整的拍摄图片、下载配置文件及利用配置文件进行分类的完整功能。其中使用到了上一节所述的移动端的分类器进行分类计算，并将拍摄和下载的功能联结起来。根据第四章中的用例分析和 APP 的设计类图，该模块的最终类图如下：

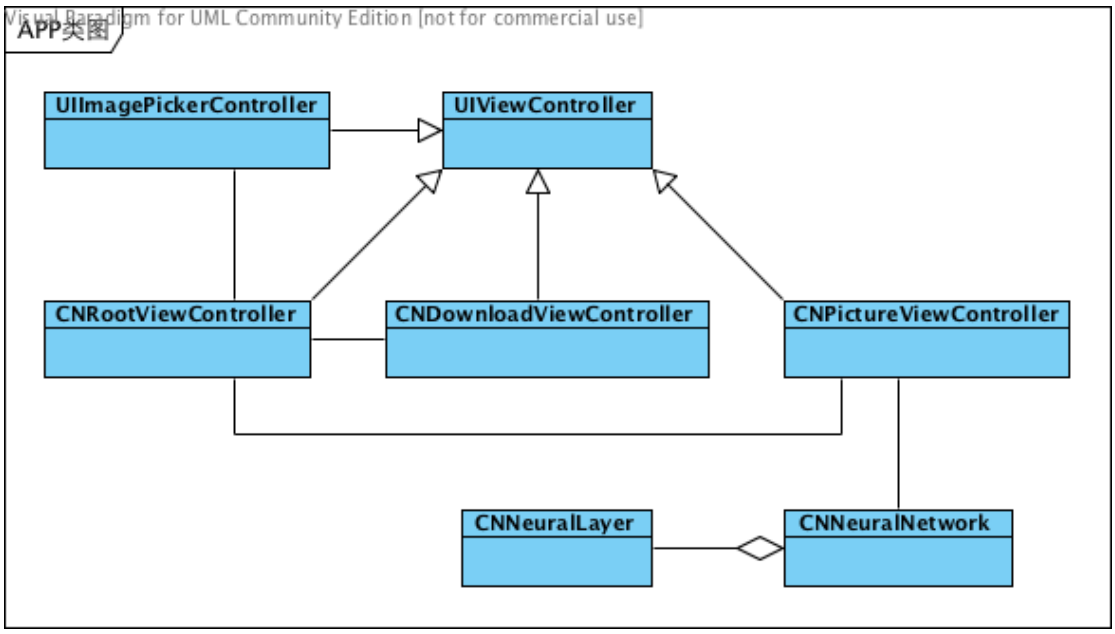

图 5-7　　APP 详细类图

如图 5-7，移动端 APP 主要包括了 3 个用于交互界面的类：CNRootViewController、CNDownloadViewController、CNPictureViewController，其中针对交互中的每个页面分配了一个控制类，并将分类器当做一个公共类来实现调用。

本 APP 中主要包含了以下界面设计：

1. 主界面

如图 5-8，该界面主要包含一个用于选择配置文件及测试配置文件的列表，和三个分别转入配置下载、拍摄照片、导入照片页面的按钮。

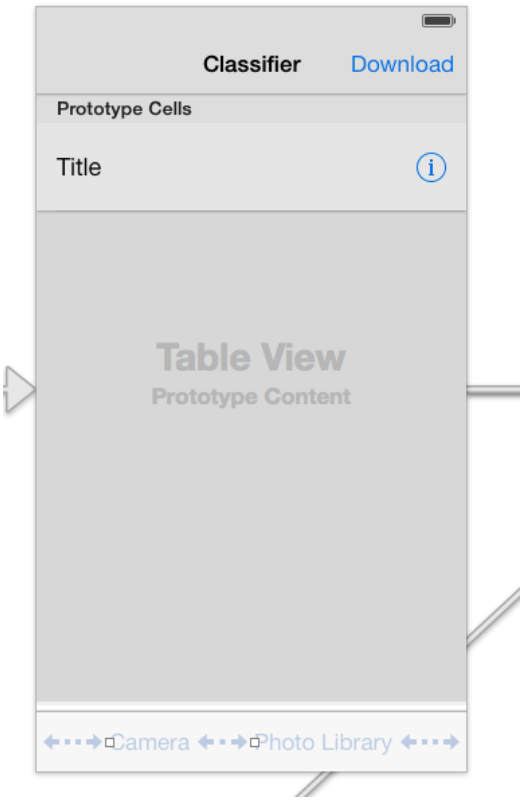

图 5-8　主界面界面设计

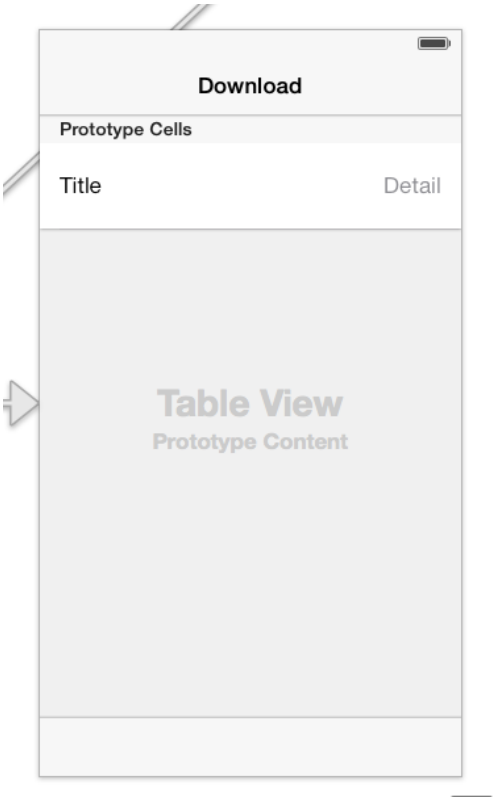

图 5-9　配置下载界面设计

2. 下载页面

如图 5-9，该页面主要包含一个显示可用的配置文件的列表，点击其中的条目即进行下载。

3. 标注结果页面

如图 5-10，该页面包括一个显示使用的图片的 ImaveView 和一个用于显示分类结果的 ResultLabel。

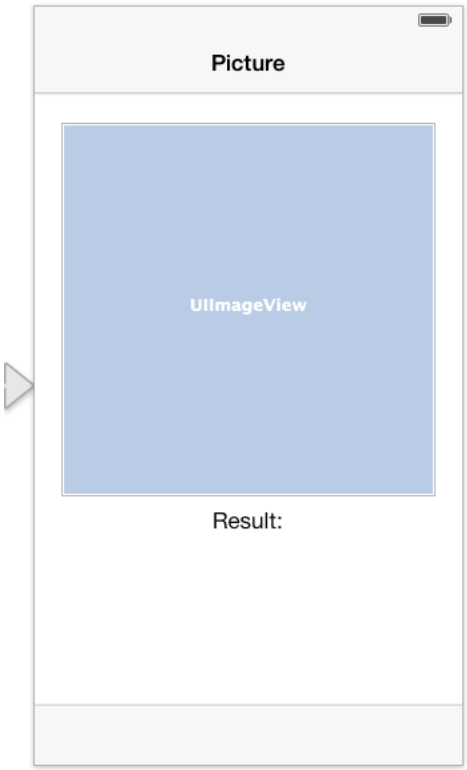

图 5-10　标注结果界面设计

本 APP 的主要操作流程如下图：

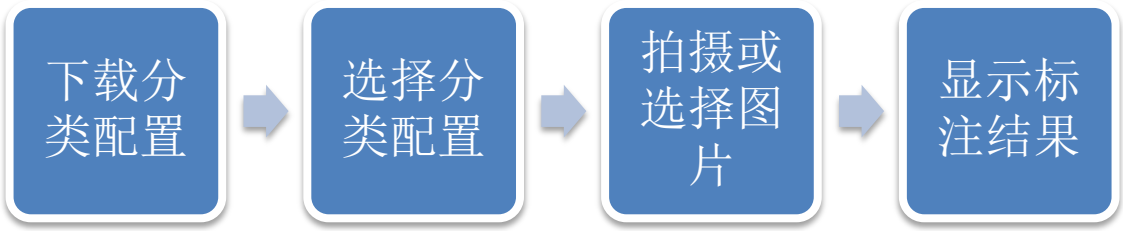

图 5-11　APP 操作流程图

如图 5-11，使用 APP 进行图片标注工作流程如下：

1. 点击主界面右上角的 Download 按钮，进入下载配置文件页面。
2. 下载配置文件类开始去获取配置文件列表，并显示在列表中。
3. 通过点击列表中的条目开始下载相应的配置文件，并在下载完成后自动解压并提示完成。
4. 点击左上角的 Back 按钮返回主界面，选择相应的分类配置后，使用下方的按钮开始拍摄图片或导入图片。
5. 拍摄图片或导入图片完成后，进入显示结果页面，该页面的控制器首先初始化了一个使用该分类配置还原的神经网络，之后将结果显示在该结果页面上。

如上即为该移动端 APP 实现过程的主要介绍。

5.5 本章小结

本章中分别对系统组成的各个关键模块进行了详细的分析设计和实现，完成了完整系统的一个原型系统。下一章将对本系统的各个方面进行全面的测试。

第六章 系统部署与应用

本章中对本文所实现的原型系统所需的开发环境及运行环境进行了介绍，接下来分别针对系统中的各个关键模块进行了性能、正确性等方面的测试。最后对测试结果进行了分析。

6.1 开发环境及运行环境

开发环境如表 6-1

表 6-1　开发环境

软件环境	
操作系统	Mac OSX 10.9.2, Ubuntu 13.10
开发环境	Xcode 5.2, TextMate 2.0-alpha, Sublime Text 3-beta
开发语言	Python, objective-c
语言库	Python 2 以上
Python 类库	Scipy, Numpy, Theano
iOS 计算框架	Acclerate.framework
测试用图像库	Cifar-10, Caltech-101
CUDA 版本	5.5
硬件环境	
CPU	Intel Core i5 2.4GHz
内存	8G

运行环境如表 6-2

表 6-2　运行环境

PC 端	
软件环境	
操作系统	Ubuntu 13.10
语言库	Python 2 以上
Python 类库	Scipy, Numpy, Theano
CUDA 版本	CUDA 5.5
硬件环境	
CPU	Intel Xeon E5630 2.53GHz
内存	8G
显卡	NVIDIA Quadro 4000
CUDA 流处理器	256 个
显存	256bit 2G GDDR5
移动端	
软件环境	
操作系统	iOS 7.0.4
iOS 计算框架	Acclerate.framework
硬件环境	
型号	iPhone5s
处理器	A7 双核 1.3GHz

6.2 系统测试

6.2.1 数据集的选取

在本系统的测试中，我们使用 cifar-10 图片库[30]和 caltech-101 图片库[31]作为测试数据。选择这两组差异较大的数据集，其中一组作为小图片的图片集，另一组作为大图片的图片集；一组作为分类较少的图片集，另一组作为分类较多的

图片集，从而进行对比测试。

Cifar-10 图片集由多伦多大学提供，包含了 10 个分类的 60000 张图片，其中每个类别各有 6000 张图片，图片大小为 32x32 的彩色图片。由于 cifar-10 提供了可供 Python 直接读入的数据文件，因此我们只需要对其进行预处理的转换后即可在以后的测试中使用，因此训练集生成模块测试中并未使用 cifar-10 图片集作为测试数据。

Caltech-101 图片集包含了 101 种不同类别的 8677 张彩色图片，图片种类涵盖各个方面，每个种类包含 40-800 张图片，图片文件为 jpg 文件，图片大小各不相同，在 300x200 左右。

6.2.2 训练集生成模块测试

训练集生成模块的测试主要是测试训练集生成模块工作是否正常，且运行性能如何，处理图片后所生成的数据集大小等。

在这里我们使用 caltech-101 图片集作为测试数据。

我们将生成的样本集的规格设定为 100*100 的三通道数据，使用 70%的数据作为训练集，30%的数据作为测试集，将 caltech-101 的数据集放入 data 文件夹中，配置 data.txt 的参数如表 6-3：

表 6-3　data.txt 的参数

参数名	值
imageSize	100
channel	3
trainPercent	70
validPercent	30

运行程序结果如图 6-1、图 6-2：

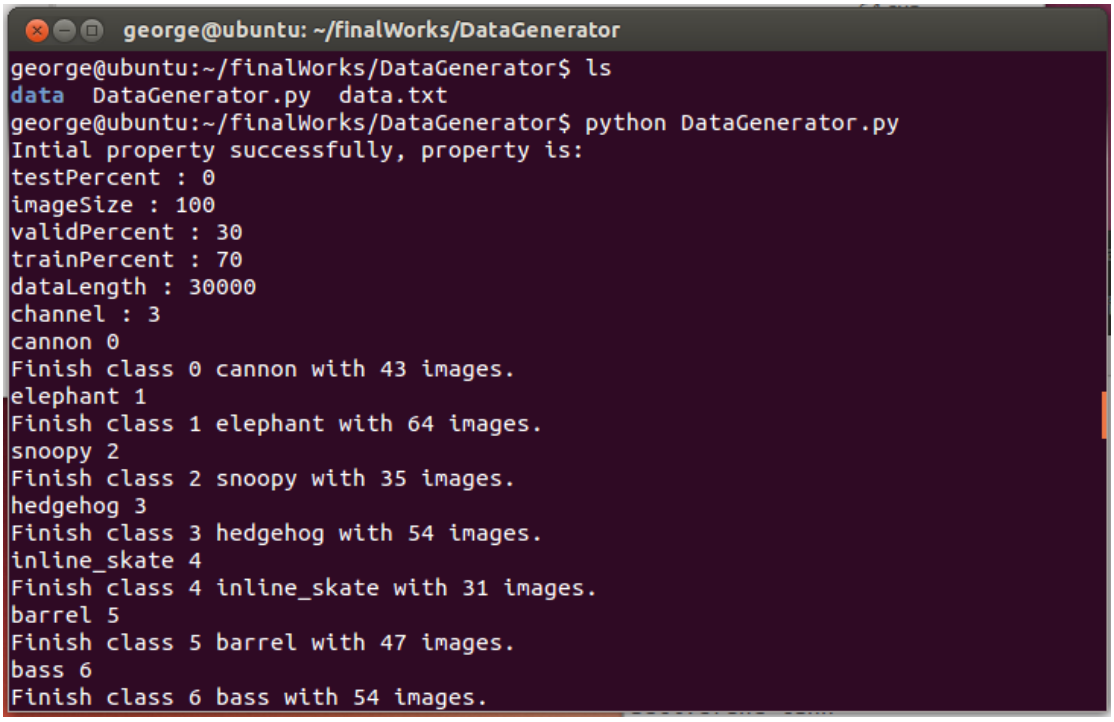

图 6-1　样本集生成结果 1

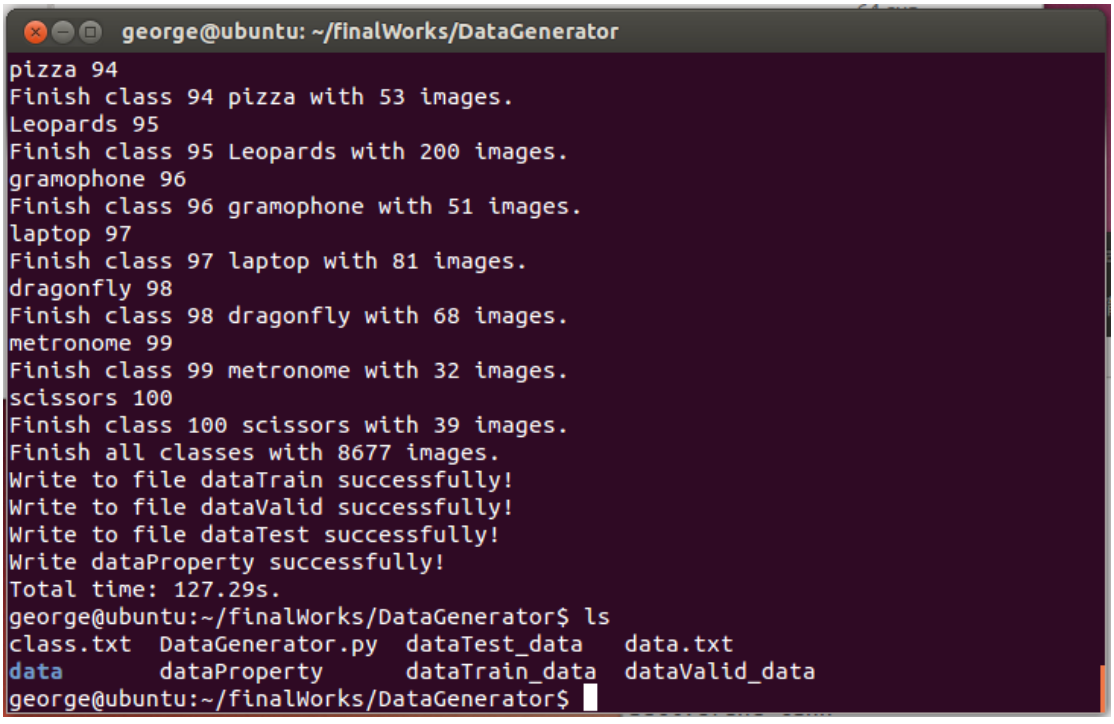

图 6-2　样本集生成结果 2

处理 8677 张图片花费时间为 127s，查看生成的输出文件，训练文件 trainData 大小为 734m，测试文件 validData 大小为 307m。

6.2.3 PC 端训练器模块测试

PC 端训练器的测试为本系统中最重要的部分之一，主要测试该模块的读入数据、构造神经网络、训练神经网络、保存配置文件的各个方面，以及运行的性能效率、收敛速度等。

在这里我们分别使用 cifar-10 和 caltech-101 作为测试数据进行测试。分别根据两个数据集的特点构造神经网络配置，并进行测试，在 100 次迭代后的最佳错误率。

1. Cifar-10

在训练时，我们配置的神经网络配置如表 6-4：

表 6-4　Cifar10 的配置参数

全局配置	
网络层数	4
学习速率	0.1
最大训练迭代	50
期望错误率	0.2
批处理大小	500
图片尺寸	32
图片通道数	3
网络层 1	
类型	卷积层
卷积核尺寸	5x5
卷积核数量	30
池化尺寸	2x2
激活函数	tanh
网络层 2	
类型	卷积层
卷积核尺寸	5x5

卷积核数量	60
池化尺寸	2x2
激活函数	tanh
网路层 3	
类型	全连接层
单元数	500
激活函数	tanh
网络层 4	
类型	逻辑回归层
分类数	10

运行程序结果如图 6-3、图 6-4、图 6-5：

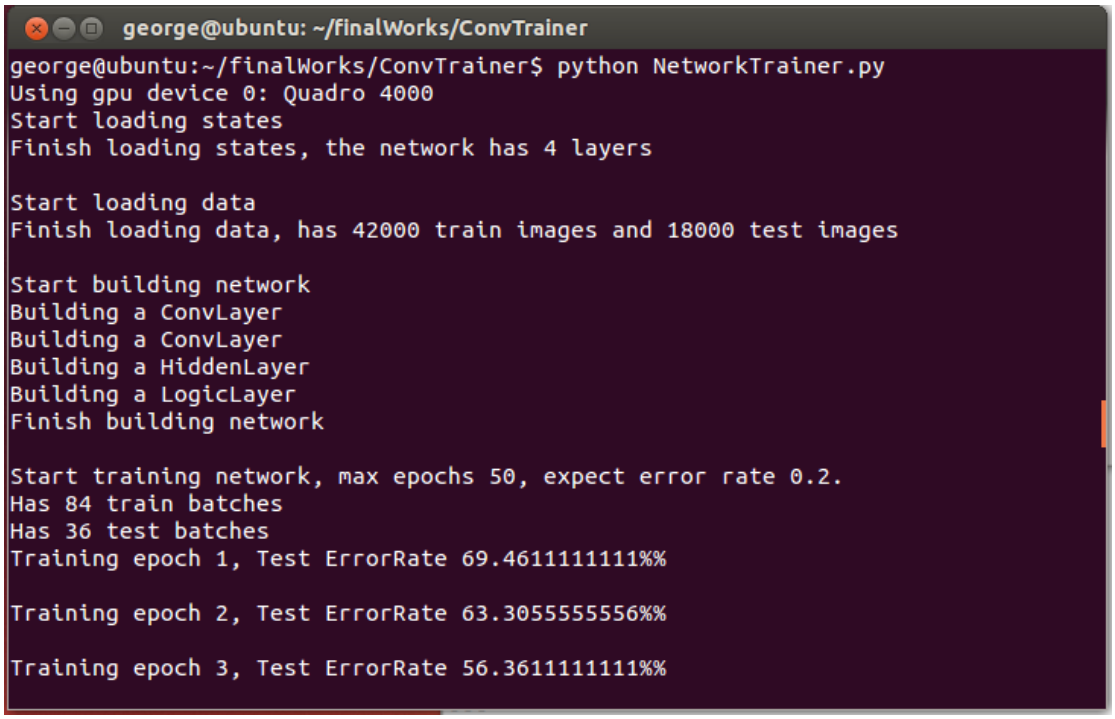

图 6-3　　Cifar10 运行结果 1

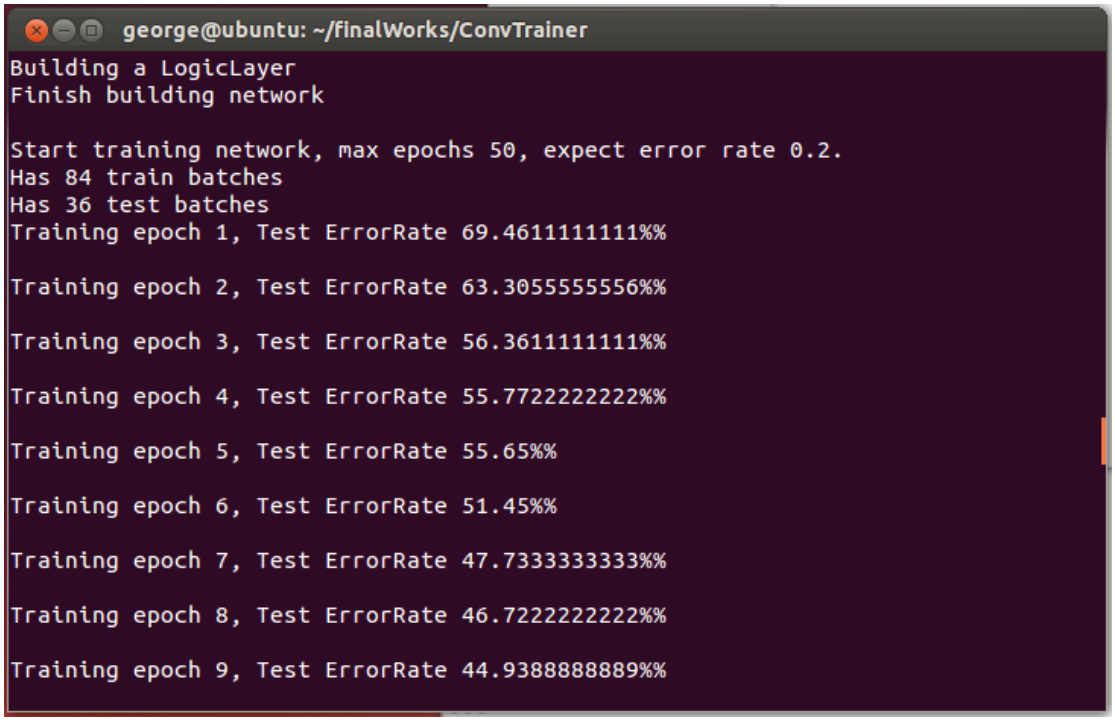

图 6-4　Cifar10 运行结果 2

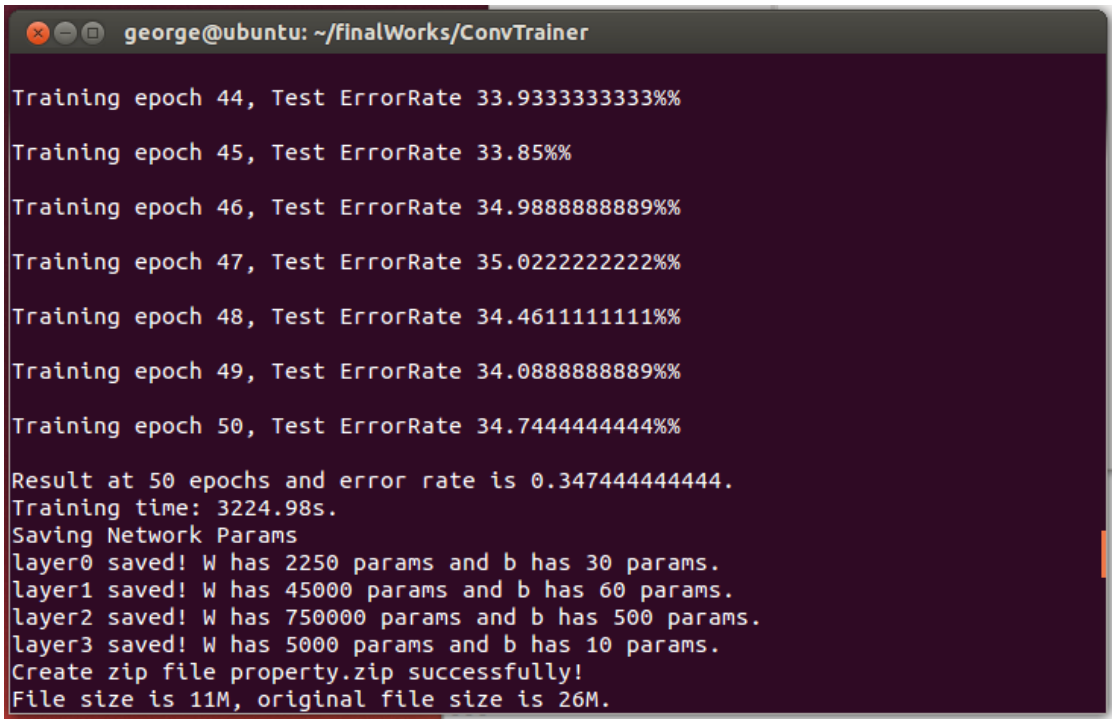

图 6-5　Cifar10 运行结果 3

最终保存的神经网络未达到期望错误率，最好错误率为 33%，在第 37 次迭代时获得，训练了 50 个迭代所需的训练时间为 3225 秒，配置文件大小为 11m。

2. Caltech-101

在训练时，我们配置的神经网络配置如表 6-5：

表 6-5　Caltech101 的配置参数

全局配置	
网络层数	6
学习速率	0.1
最大训练迭代	50
期望错误率	0.35
批处理大小	100
图片尺寸	100
图片通道数	3
网络层 1	
类型	卷积层
卷积核尺寸	5x5
卷积核数量	30
池化尺寸	2x2
激活函数	tanh
网络层 2	
类型	卷积层
卷积核尺寸	5x5
卷积核数量	60
池化尺寸	2x2
激活函数	tanh
网络层 3	
类型	卷积层
卷积核尺寸	5x5
卷积核数量	30
池化尺寸	2x2
激活函数	tanh
网络层 4	

类型	卷积层
卷积核尺寸	5x5
卷积核数量	30
池化尺寸	2x2
激活函数	tanh
网路层 5	
类型	全连接层
单元数	500
激活函数	tanh
网络层 6	
类型	逻辑回归层
分类数	101

运行程序结果如图 6-6、图 6-7、图 6-8：

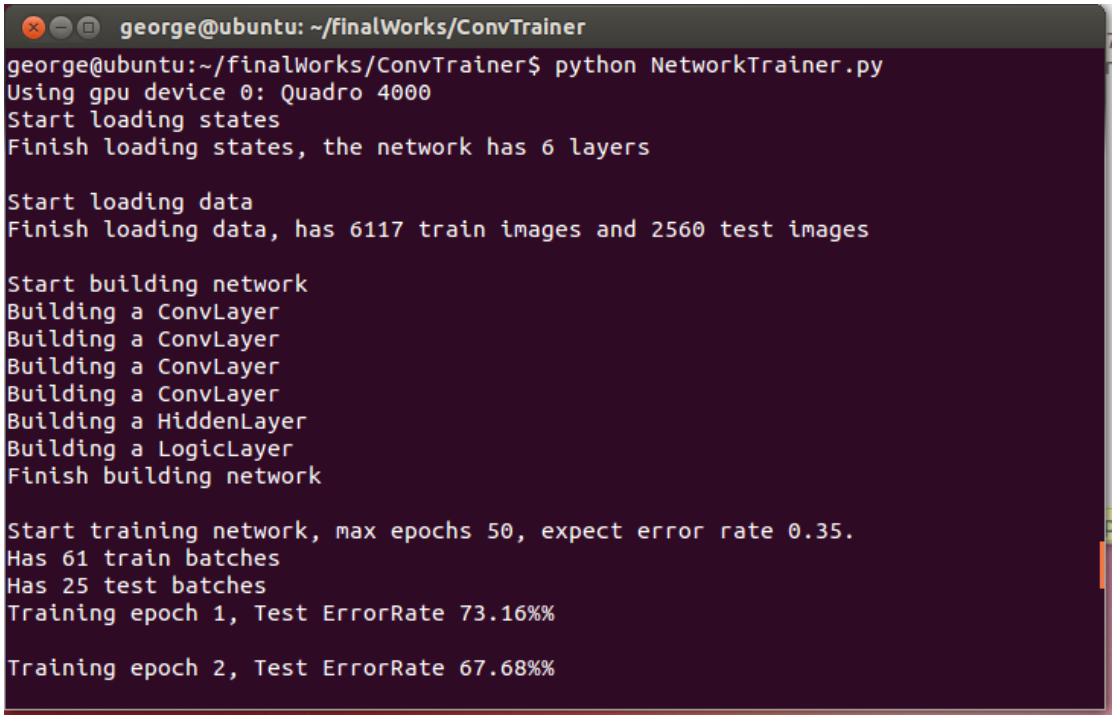

图 6-6　Caltech101 运行结果 1

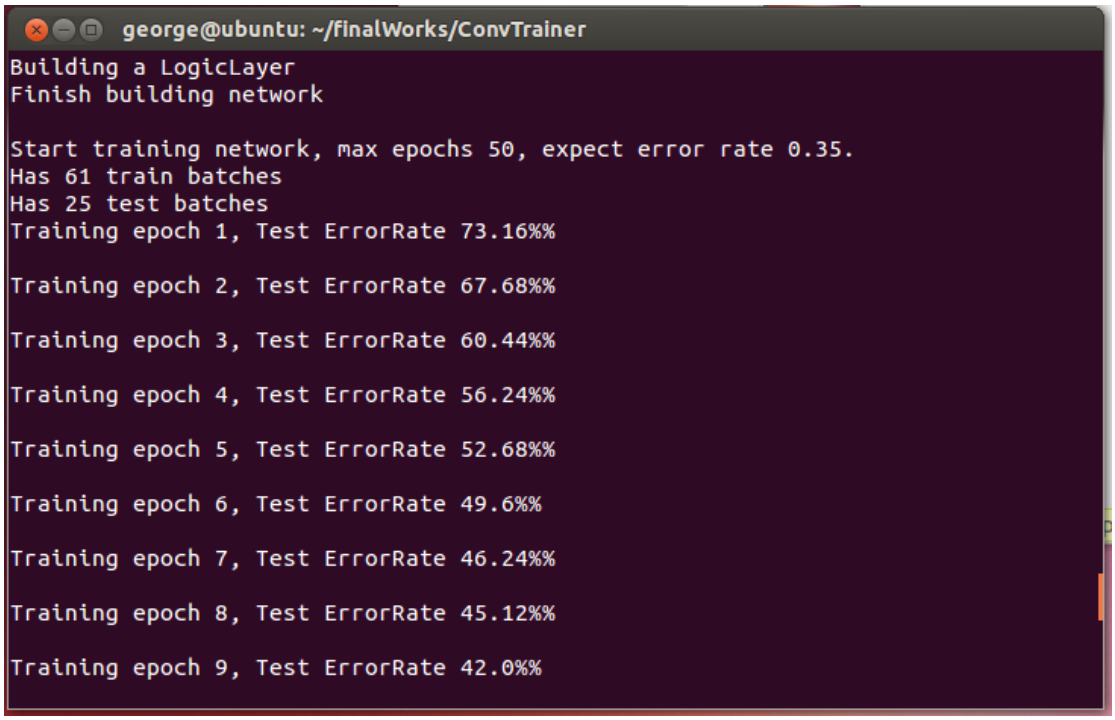

图 6-7　Caltech101 运行结果 2

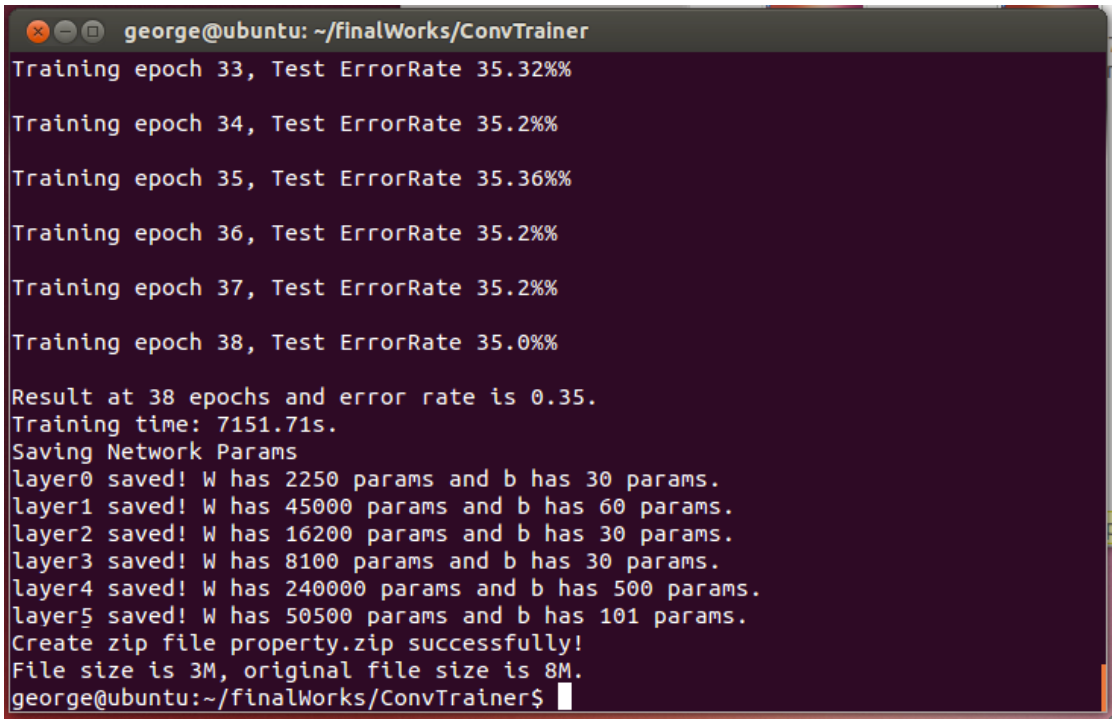

图 6-8　Caltech101 运行结果 3

最终保存的神经网络达到期望错误率 35%所需的训练时间为 7151 秒，配置文件大小为 3m。

将训练过程绘制成折线图，如图 6-9 所示。

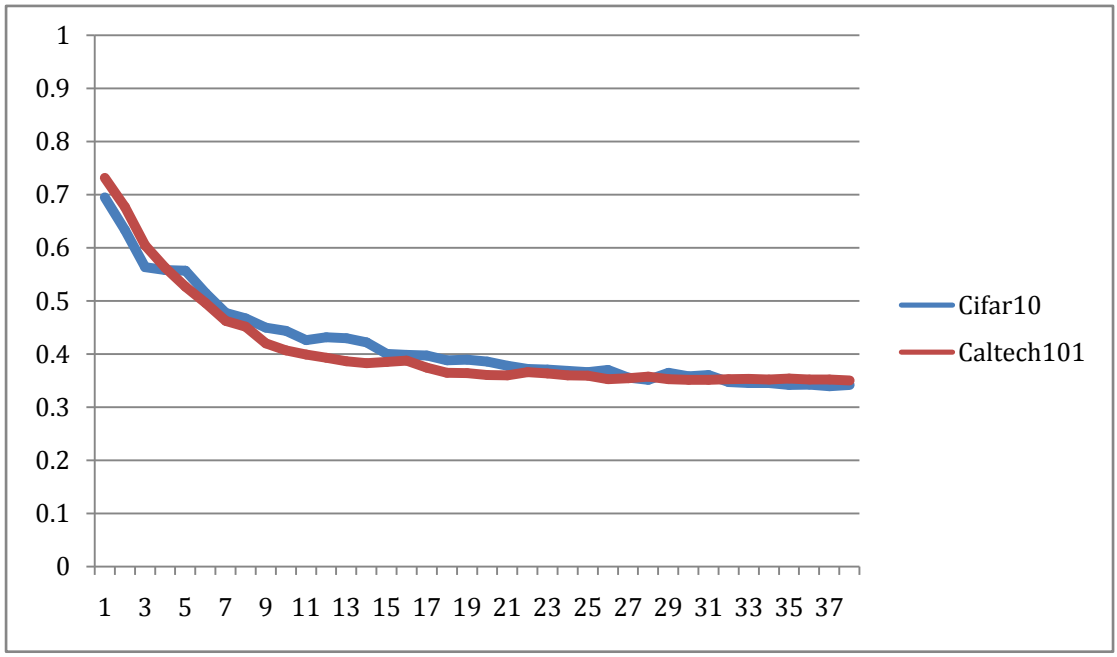

图 6-9　　错误率收敛对比图

6.2.4 移动端分类器模块测试

移动端分类器模块的测试是本系统中另一个重要的部分，其主要测试 PC 端和移动端共享配置文件是否正确，以及移动端的运行性能水平。

在分类器模块的测试中，我们将原测试集中的 100 张图片的数据随配置文件打包，以供在移动端进行测试。

在 APP 的主页面的选择配置文件列表处，点击右边的按钮可以进入配置文件测试，在短暂时间后显示结果。

1. Cifar-10

运行结果如图 6-10：

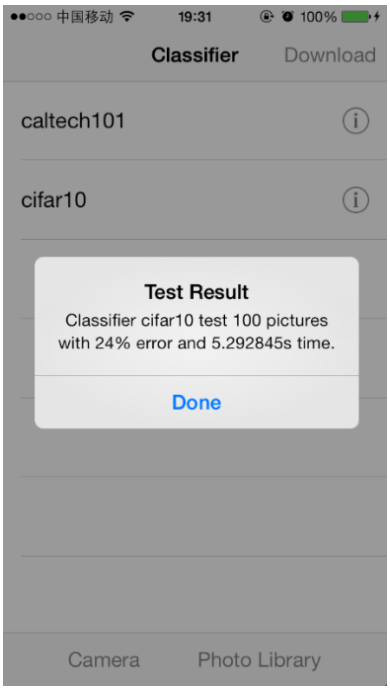

图 6-10　Cifar10 移动端测试

100 个样本的计算时间为 5.29 秒，错误率为 24%。

2. Caltech-101

运行结果如图 6-11：

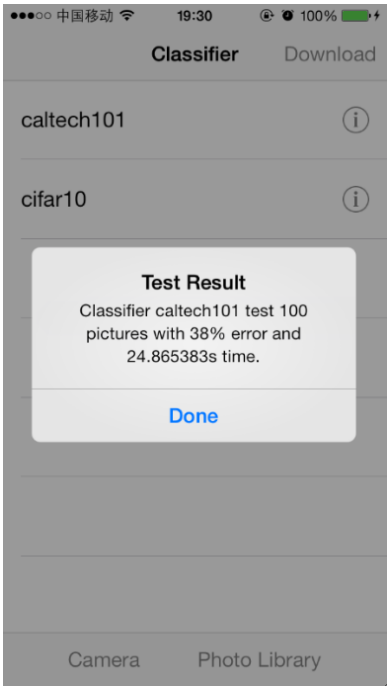

图 6-11　Caltech101 移动端测试

100 个样本的计算时间为 24.86 秒，错误率为 38%。

6.2.5 移动端 APP 使用测试

此处为本章最终的成品 APP 的展示测试，包括下载配置、拍摄或选择图片、对图片进行关键字标注三个测试关键点。下面将逐一进行功能测试和结果分析。

1. 下载配置

测试步骤为：

（1） 在主界面点击右上角的 Download 按钮；

（2） 在列表中点击要下载的分类器配置文件；

（3） 等待下载完成后，确认提示框下载成功；

（4） 点击左上角的 Back 按钮回到主菜单，看到配置文件已下载成功。

APP 运行结果如图 6-12 到 6-15：

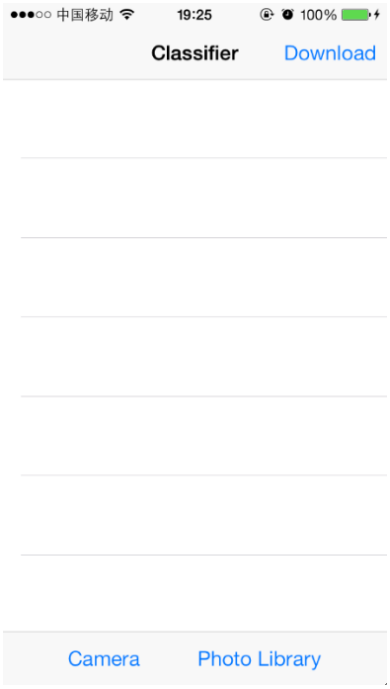

图 6-12 下载配置 1

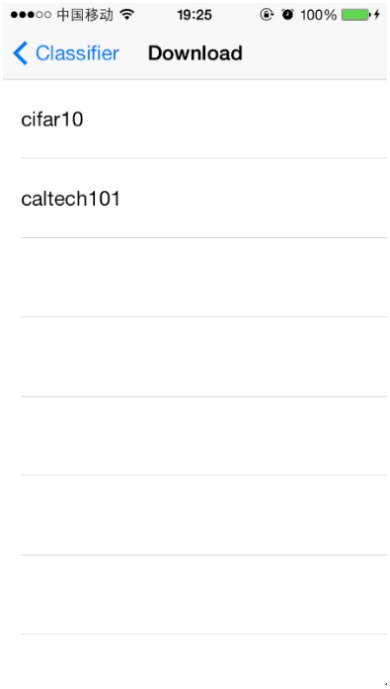

图 6-13 下载配置 2

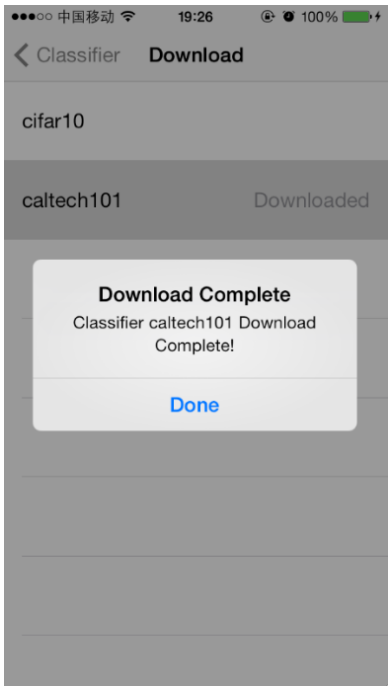

图 6-14　下载配置 3

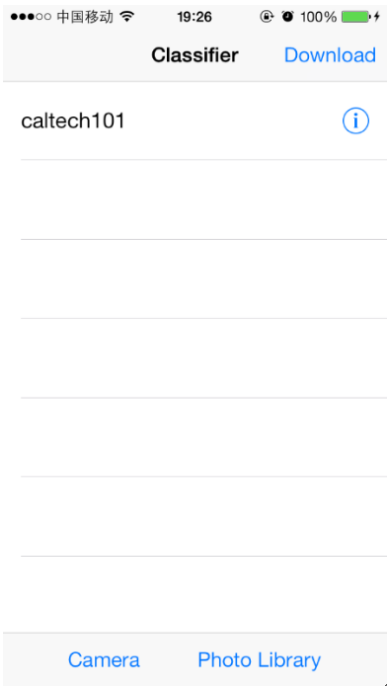

图 6-15　下载配置 4

程序工作正常。

2. 拍摄或选择图片

测试步骤：

（1） 在主界面点击 Camera 按钮；

（2） 在拍照界面拍摄完成后，确认得到的照片；

（3） 点击 Retake，再点击 Cancel；

（4） 在主界面点击 Library 按钮；

（5） 选择了要导入的图片后，确认得到的结果图片。

APP 运行结果如图 6-16 到 6-18：

图 6-16　拍摄测试 1

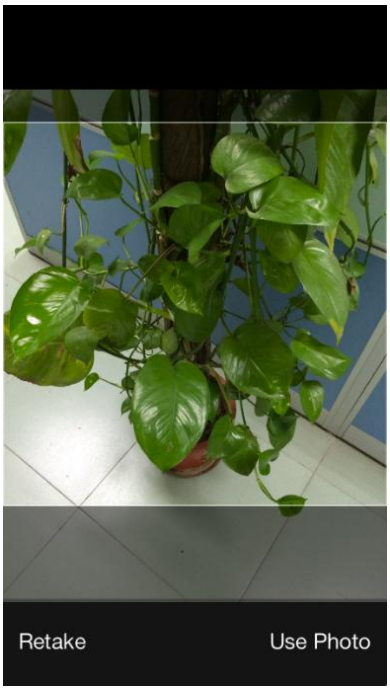

图 6-17　拍摄测试 2

图 6-18　导入测试

程序工作正常。

3. 对图片进行关键字标注

测试步骤：

（1） 在主界面选择所要使用的配置文件；

（2） 点击拍照或导入图片后，点击 Use Photo；

（3） 在结果画面中确认分类结果。

当使用 cifar-10 的配置文件时，APP 运行结果如图 6-19 到 6-22：

图 6-19 拍摄标注 1

图 6-20 拍摄标注 2

图 6-21 导入标注 1

图 6-22 导入标注 2

程序工作正常。

当使用 caltech-101 的配置文件时，APP 运行结果如图 6-23 到 6-26：

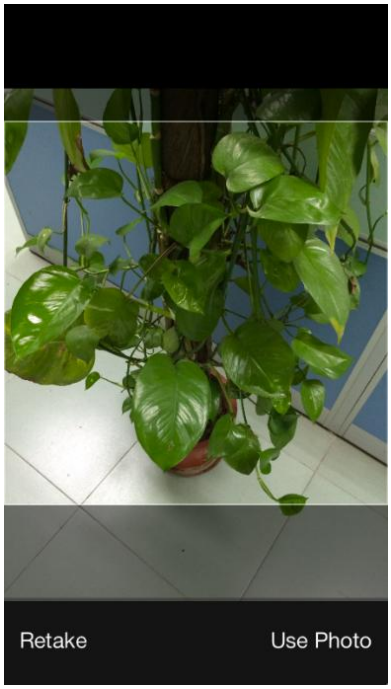

图 6-23　拍摄标注 1

图 6-24　拍摄标注 2

图 6-25　导入标注 1

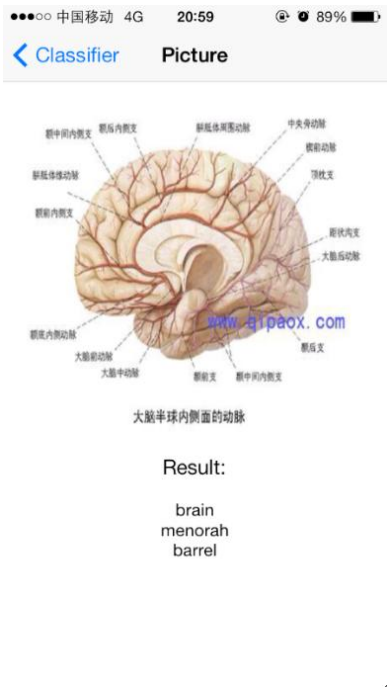

图 6-26　导入标注 2

程序工作正常。

6.3 测试结果分析

经过本章对各个模块的分别测试，测试结果达到预期设想，各模块工作正常、性能满足需求，每张图片的处理时间在 0.3 秒以下。

6.4 本章小结

本章中分别从样本集生成模块、分类器训练模块、移动端分类器分类模块、移动 APP 进行了全面的测试，并在最后进行了结果分析。下一章中将对全文进行总结，并对实现过程中的不足和未来的应用前景进行分析。

第七章 总结与展望

本章为本文的最后一章，将对全文进行总结，对本文中所实现的系统的不足进行分析，并对本应用未来的应用前景进行展望。

7.1 总结

本文针对图片关键字标注问题，基于深度学习的方法，提出了一种使训练器和分类器分离在不同的平台的解决方案，通过传输配置文件使得移动端可以不断获取新的标注分类能力，分类计算在移动端本地运行，不需借助网络与云端进行连接。这样一方面利用了 PC 端强大的计算能力可以扩充分类器的分类种类，另一方面保持了移动端的可移动性和灵活性。

基于以上解决方案，本文首先对其中利用的关键技术卷积神经网络的原理进行了详细的介绍，之后使用软件工程设计的流程，分别从需求分析、系统设计、详细设计和实现、测试分析等方面，对本系统进行了完整的实现，并最终实现出了一个满足远功能需求的原型系统。

本文中所完成的原型系统主要分为两部分，其中分类样本集和神经网络运算在 PC 平台执行，在训练完成后得到一个神经网络的配置文件，将其传输至移动端的 APP 上，即可在移动端 APP 上使用该分类对即时拍摄的图片进行关键字的标注。

7.2 展望

本文中所提供的对于图片关键字标注的解决方案，使用 PC 平台机器学习训练生成分类参数，在移动平台使用较小的计算量即可实现对图片进行关键字标注。但是，本文中最终的实现结果上仍有一些不足和可供改进的地方，主要有以下三点：

第一，本文中所提供的卷积神经网络算法，对于不同的分类样本集和分类要

求，整个网络需要重新配置、生成、训练，消耗了较多的训练时间。在以后的改进中可以引入深信度网络等方法，使得神经网络的前几层辨识度提高，在不同网络中使用共同的前几层配置和参数，再加入新的类的训练时只需要训练最后几层，从而提高训练效率。

第二，本文中所生成的神经网络配置文件，在神经网络规模较庞大时容量较大，再加入较多分类配置文件时呈线性增长。在以后的改进中可根据上一条，将若干分类配置使用共用的前几层，从而减小各神经网络中不同的参数量。

第三，本文中最终完成的移动端 APP 只有 iOS 平台的应用，且仍有较大丰富空间。接下来的工作中可以开发出更多移动平台的通用计算模块，从而实现更加广泛的可适用范围。

参考文献

[1] Internet.org. "A Focus on Efficiency. " *A whitepaper from Facebook, Ericsson and Qualcomm* (2013): 5-6.

[2] Lowe, David G. "Distinctive image features from scale-invariant keypoints." *International journal of computer vision* 60.2 (2004): 91-110.

[3] McCulloch, Warren S., and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." *The bulletin of mathematical biophysics* 5.4 (1943): 115-133.

[4] Bengio, Yoshua, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives." (2013): 1-1.

[5] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." *Machine learning* 20.3 (1995): 273-297.

[6] Freund, Yoav, and Robert E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting." *Journal of computer and system sciences* 55.1 (1997): 119-139.

[7] Bryson, Arthur Earl. *Applied optimal control: optimization, estimation and control*. CRC Press, 1975.

[8] Fukushima, Kunihiko. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." *Biological cybernetics* 36.4 (1980): 193-202.

[9] Hubel, David H., and Torsten N. Wiesel. "Receptive fields and functional architecture of monkey striate cortex." *The Journal of physiology* 195.1 (1968): 215-243.

[10] LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." *Neural computation* 1.4 (1989): 541-551.

[11] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

[12] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning

algorithm for deep belief nets." *Neural computation* 18.7 (2006): 1527-1554.

[13] Hinton, Geoffrey E. "Learning multiple layers of representation." *Trends in cognitive sciences* 11.10 (2007): 428-434.

[14] Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507.

[15] Bengio, Yoshua. "Learning deep architectures for AI." *Foundations and trends® in Machine Learning* 2.1 (2009): 1-127.

[16] Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." *Proceedings of the 25th international conference on Machine learning*. ACM, 2008.

[17] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." *NIPS*. Vol. 1. No. 2. 2012.

[18] Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

[19] Dean, Jeffrey, et al. "Large Scale Distributed Deep Networks." *NIPS*. 2012.

[20] Kavukcuoglu, Koray, et al. "Learning Convolutional Feature Hierarchies for Visual Recognition." *NIPS*. Vol. 1. No. 2. 2010.

[21] Snyman, Jan. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*. Vol. 97. Springer, 2005.

[22] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. *Learning representations by back-propagating errors*. MIT Press, Cambridge, MA, USA, 1988.

[23] Korekado, Keisuke, et al. "A convolutional neural network VLSI for image recognition using merged/mixed analog-digital architecture." *Knowledge-Based Intelligent Information and Engineering Systems*. Springer Berlin Heidelberg, 2003.

[24] Cireşan, Dan C., et al. "Flexible, high performance convolutional neural

networks for image classification." *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*. AAAI Press, 2011.

[25] LeCun, Yann A., et al. "Efficient backprop." *Neural networks: Tricks of the trade*. Springer Berlin Heidelberg, 2012. 9-48.

[26] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010.

[27] Jarrett, Kevin, et al. "What is the best multi-stage architecture for object recognition?." *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009.

[28] Oliphant, Travis E. "Python for scientific computing." *Computing in Science & Engineering* 9.3 (2007): 10-20.

[29] Bergstra, James, et al. "Theano: a CPU and GPU math expression compiler." *Proceedings of the Python for scientific computing conference (SciPy)*. Vol. 4. 2010.

[30] Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." *Computer Science Department, University of Toronto, Tech. Rep* (2009).

[31] Fei-Fei, Li, Rob Fergus, and Pietro Perona. "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories." *Computer Vision and Image Understanding* 106.1 (2007): 59-70.

附录

致谢

经过将近一年的忙碌，本论文的工作已经接近尾声。在这个过程中，我从对这个领域一无所知，到完成毕业设计以及论文的过程中，学到了很多，收获了很多，当然也离不开老师和同学们方方面面的帮助。如果没有导师的指导与同学们的鼓励与支持，我就不可能接触到深度学习这个奇妙的研究领域，在此，本人深感感谢。

在这里首先要感谢我的导师。虽然平日工作繁忙，但每周仍定期召开例会了解我们的工作进展和情况，并予以亲切的指导。在我们毕业设计工作的各个阶段，从查阅资料、确定论文方向，到后期修改、终稿完成，都给予了我悉心的指导。本人在此深感感谢。

其次要感谢实验室的师兄和同学们，在我涉足新领域遇到困难时，同我进行密切的讨论与建议，帮助我渡过了一个又一个难关。

最后，感谢我的家人和朋友，一直鼓励我在这条路上坚持向前。