毕 业 设 计

**题目：基于深度学习的手写数字识别算法实现**

**学 生：**

**学 号：**

**学 院：**

**专 业：**

**指导教师：**

**年** **月** 日

基于深度学习的手写数字识别算法实现

摘 要

深度学习是传统机器学习下的一个分支，得益于近些年来计算机硬件计算能力质的飞跃，使得深度学习成为了当下热门之一。手写数字识别更是深度学习入门的经典案例，学习和理解其背后的原理对于深度学习的理解有很重要的作用。

本文将采用深度学习中的卷积神经网络来训练手写数字识别模型。使用卷积神经网络建立合理的模型结构，利用卷积层中设定一定数目的卷积核（即滤波器），通过训练数据使模型学习到能够反映出十个不同手写提数字特征的卷积核权值，最后通过全连接层使用softmax函数给出预测数字图对应每种数字可能性的概率多少。

本文以学习基于深度学习的手写数字识别算法的过程为线索，由简入深，从最基础的感知器到卷积神经网络，学习和理解深度学习的相关基本概念、模型建立以及训练过程。在实现典型LeNet-5网络结构的同时，通过更改超模型结构、超参数进一步探索这些改变对模型准确率的影响。最后通过使用深度学习框架Keras以MNIST作为训练数据集训练出高识别率的模型并将其与OpenCV技术结合应用到摄像头上实现实时识别数字，使用合理的模型结构，在测试集上识别准确率达到99%以上，在与摄像头结合实际应用中的识别效果达到90%以上。

关键词：深度学习，卷积神经网络，MNIST，OpenCV

**Realization of Handwritten Numeral Recognition Algorithm Based on Deep Learning**

ABSTRACT

Depth learning is a branch of traditional machine learning, thanks to the recent years, computer hardware computing power of the quality of the leap, making the depth of learning has become one of the popular. Handwritten digital recognition is the classic case of advanced learning, learning and understanding the principles behind the depth of learning for the understanding of a very important role.

In this paper, the convolution neural network in depth learning will be used to train the handwritten numeral recognition model. The convolution neural network is used to establish a reasonable model structure. A certain number of convolution cores (ie, filters) are set in the convolution layer. The training data are used to study the convolution of the model to reflect ten different handwritten digital features Kernel weight, and finally through the full connection layer using softmax function gives the predicted digital map corresponding to the probability of each number of the probability of how much.

In this paper, we study the basic concepts, model establishment and training process of the depth learning based on the process of learning the handwritten numeral recognition algorithm based on the depth learning. The basic concepts, the model establishment and the training process are studied and understood from the most basic sensor to the convolution neural network. In the realization of the typical LeNet-5 network structure at the same time, by changing the super-model structure, super-parameters to further explore the impact of these changes on the model accuracy. Finally, by using the depth learning framework Keras to MNIST as a training data set to train a high recognition rate model and combine it with OpenCV technology to apply real-time identification numbers to the camera, using a reasonable model structure, the recognition accuracy is achieved on the test set More than 99%, with the camera in the practical application of the recognition effect of more than 90%.

**Key words:** deep Learning, convolution neural network, MNIST, OpenCV

目 录

[摘要 I](#_Toc485553973)

[ABSTRACT II](#_Toc485553974)

[1 绪论 1](#_Toc485553975)

[1.1 数字识别研究现状 1](#_Toc485553976)

[1.2 深度学习的发展与现状 1](#_Toc485553977)

[1.3 研究意义 2](#_Toc485553978)

[1.4 论文结构 3](#_Toc485553979)

[2 卷积神经网络基本原理 4](#_Toc485553980)

[2.1 卷积神经网络 4](#_Toc485553981)

[2.1.1 卷积神经网络概述 4](#_Toc485553982)

[2.1.2 卷积神经网络的重要组成部分 4](#_Toc485553983)

[2.1.3 权值共享和局部连接 5](#_Toc485553984)

[2.2 神经网络的前向传播和反向传播 6](#_Toc485553985)

[2.2.1 神经元 6](#_Toc485553986)

[2.2.2 神经网络的连接形式 7](#_Toc485553987)

[2.2.3 神经网络的前向传播 8](#_Toc485553988)

[2.2.4 神经网络的反向传播算法 8](#_Toc485553989)

[2.3 优化方法——梯度下降 9](#_Toc485553990)

[2.3.1 批梯度下降 9](#_Toc485553991)

[2.3.2 随机梯度下降 11](#_Toc485553992)

[2.3.3 小批量梯度下降 11](#_Toc485553993)

[2.4 小结 11](#_Toc485553994)

[3 Keras深度学习框架 12](#_Toc485553995)

[3.1 Keras简介 12](#_Toc485553996)

[3.2 Keras编程 12](#_Toc485553997)

[3.2.1 Keras模型构建 12](#_Toc485553998)

[3.2.2 Keras常用函数及用法 12](#_Toc485553999)

[3.3 Keras环境配置 13](#_Toc485554000)

[3.4 小结 14](#_Toc485554001)

[4 经典LeNet-5实验探究 15](#_Toc485554002)

[4.1 数据集MNIST介绍 15](#_Toc485554003)

[4.2 LeNet-5实现 16](#_Toc485554004)

[4.2.1 LeNet-5介绍 16](#_Toc485554005)

[4.2.2 LeNet实现 16](#_Toc485554006)

[4.3 模型探究 18](#_Toc485554007)

[4.3.1 不同网络结构 18](#_Toc485554008)

[4.3.2 卷积核大小数量 20](#_Toc485554009)

[4.3.2 权值初始化 22](#_Toc485554010)

[4.3.3 激活函数选择 24](#_Toc485554011)

[4.4 小结 26](#_Toc485554012)

[5 手写数字识别算法应用实践 27](#_Toc485554013)

[5.1 OpenCV图像处理部分 27](#_Toc485554014)

[5.1.1 OpenCV介绍 27](#_Toc485554015)

[5.1.2 OpenCV安装配置 27](#_Toc485554016)

[5.1.3 寻找数字 28](#_Toc485554017)

[5.1.4 数字处理 32](#_Toc485554018)

[5.2 实现摄像头的手写数字实时识别 33](#_Toc485554019)

[5.2.1 手写数字识别模型训练 33](#_Toc485554020)

[5.2.2 模型的保存与调用 35](#_Toc485554021)

[5.2.3 结果 36](#_Toc485554022)

[5.3小结 36](#_Toc485554023)

[6 总结 38](#_Toc485554024)

[致 谢 39](#_Toc485554025)

[参考文献 40](#_Toc485554026)

1 绪论

1.1 数字识别研究现状

早期的研究人员在数字识别[1]这一方向已经取得了不错的成果，如使用K-邻近分类方法，SVM分类方法，Boosting分类方法等。但这些方法多少都会有不足之处，例如K-邻近方法在预测时需要将所有的训练数据集加载至内存，然后用待测数字图片与训练集作对应像素点差的和，最后得出的差值最小的则为预测结果。显然这样的方法在正常的图片准确度上并不可靠，对于待测手写数字的要求也很高。目前识别率最好的模型应该还属基于深度学习的CNN，最典型的例子LeNet-5，美国最早将其商用到识别银行支票上得手写数字[2]。可见基于深度学习的手写数字识别在准确率上是相当可靠。

1.2 深度学习的发展与现状

机器学习[3]发展大致分为两个阶段，起源于浅层学习，发展于深度学习，深度学习是机器学习的一个重要分支。

在20世纪80年代末期，反向传播算法的应用给机器学习带来了希望的同时也掀起了基于统计模型的机器学习热潮。通过实践，人们成功发现利用反向传播算法可以使一个人工神经网络模型在大量有标签训练样本中学习统计一定的规律，在此之上进而对无标签事物进行预测。该阶段的的网络模型因为只含有一层隐含层的缘故被称之为浅层模型，浅层模型在参数个数、计算单元以及特征表达上有一定瓶颈。90年代，学术界相继提出各种各样的浅层学习模型，如支持向量机（SVM，support Vector Machine）、Boosting、最大熵方法等。这些模型相比当时的神经网络模型不论是在效率上还是在准确率上都有一定提升[4]。

直到2006年，加拿大多伦多大学教授、机器学习领域泰斗Geoffrey Hinton和他的学生在Ruslan Salakhutdinov在《科学》上发表的一篇关于“deep learning”的综述文章，正式开启了深度学习的浪潮[5]。深度学习火 起来的标志事件是2012年Geoff Hinton的博士生Alex Krizhevsky、Ilya Sutskever使用深度学习在图片分类的竞赛ImageNet上取得了识别结果第一名的好成绩，并且在识别精度上领先于同样使用深度学习进行识别的Google团队。这里的谷歌团队不是一般的团队，而是由机器学习领域的领跑者Andrew Ng和Google神人 Jeff Dean带领下有着他人团队无法企及的硬件资源和数据资源支持的团队，而打败这个顶级团队的仅仅是两个研究深度学习不久的“小毛孩”，这令学术界和工业届哗然的同时，也吸引了工业界对深度学习的大规模投入。Google收购了Hinton的DNN初创公司，并邀请Hinton加入了Google；LeCun加盟Facebook并出任AI实验室主任；百度成立了自己的深度学习研究所，并邀请到了原负责Google Brain的吴恩达。深度学习发展之快令人吃惊，在2016年初谷歌建立的AlphaGo系统在与围棋世界冠军李世石的对弈中，最终AlphaGo以4:1的大比分应得了比赛的胜利。可以看出，人工智能的第三波浪潮也是和深度学习密不可分的。深度学习里最经典的模型是全连接的神经网络、卷积神经网络CNN以及循环神经网络RNN。还有一个非常重要的技术就是深度强化学习技术，而AlphaGo所采用的就是该技术。

深度学习[4]的成功归功于三大因素——大数据、大模型、大计算。同时这个三个方向也是当前研究的热点。受益于计算能力的提升和大数据的出现，深度学习在现在的条件下可以将模型层次加深到上亿神经元的计算量的等级。GPU对深度学习计算的支持也放低了进行深度学习研究的门槛，使更多的初学研究人员可以踏足这个领域，为这个领域不断注入新鲜血液，不断提出新的思考，开阔新的应用方向。

深度学习的本质是通过构建多层隐藏层的机器学习模型和大量的训练数据，在训练中不断调整参数以寻找能反应数据集特点的特征，从而提升类似数据的分类或预测准确性。“深度模型”是手段，而“特征学习”是目的。有别于传统的浅层学习，深度学习强调了模型除输入输出层外的隐藏层数量，通常有大于2层以上的隐藏节点。除此之外深度学习突出了特征学习的重要性，通过逐层特征变换，将样本在原空间的特征表示变换到一个新特征空间，从而使分类或预测更加容易。

深度学习算法中重要的算法之一就是卷积神经网络算法，目前主要应用在图像分类、图像分割、目标检测等相关计算机视觉领域。上文中提及到的2012年取得图片分类的竞赛ImageNet冠军的团队就是使用改进后的卷积神经网络，使得识别准确率达到了质的飞跃。

1.3 研究意义

数字识别已经应用到了生活中的点滴，如停车场停车按车牌号计费，交通电子眼抓拍违章，大规模数据统计，文件电子化存储等。

阿拉伯数字作为一种全球通用的符号，跨越了国家、文化以及民族的界限，在我们的身边应用非常广泛。数字的类别数目适当，仅有10类，方便对研究方法进行评估和测试。通过研究基于深度学习的手写数字识别方法有助于对深度学习的理解，具有很大的理论实践价值。手写数字识别的方法经验还可以推广应用到其它字符识别问题上，如英文字母的识别。

本文设计将训练好的卷积神经网络模型与摄像头相结合，实现对摄像头画面中出现的数字实时识别。

1.4 论文结构

在本文基于深度学习的手写数字识别算法实现中，第一章主要对数字识别的研究现状、深度学习的发展与现状及本文的研究意义作以介绍；第二章内容为本文数字识别核心技术卷积神经网络的基本原理；第三章内容为本文采用的深度学习框架Keras相关使用；第四章内容为本文在经典LeNet5结构的基础上进行单一变量改动，以探究不同因素对模型识别率的影响，总结调参经验；第五章内容为对训练好的手写数字识别模型的实际应用，包含了图像处理和数字图像识别两部分。

2 卷积神经网络基本原理

本文采用深度学习中的卷积神经网络实现对手写数字的识别，卷积神经网络是被设计用来处理多维组数据的，如常见的彩色图像就是三个颜色通道组合。手写数字图片是典型的2D型图像数据，使用卷积神经网络可以有效通过训练提取去手写提数字的特征，本章对卷积神经网络的基本原理作以分析。

2.1 卷积神经网络

2.1.1 卷积神经网络概述

卷积神经网络（CNN），是人工神经网络的一种。它是一种特殊的对图像识别的方式，属于非常有效的带有前向反馈的网络。

常规的神经网络不能很好地适应所有的图像，例如在CIFAR-10的训练集中，图片的大小只有32 \* 32 \*3(32宽32高3颜色通道)，那么通过输入层后的第一个隐藏层的神经元将达到3072个。看似可以接受的数字，当隐藏层由一层上升到两层三层甚至更多时，后面隐藏层每一个神经元全连接权值与输入积的加和计算量将膨胀到无法想象。与此同时，除了效率的低下外，大量的参数还会导致过拟合的发生。与常规神经网络不同，卷积神经网络的各层中的神经元是3维排列的：宽度、高度和深度（此处的深度不是网络结构的层数），是一种立体结构。在卷积网络最后的输出层里，会把三维结构的数据转换为在深度方向的一维分类值。

卷积神经网络诞生的主要目的是为了识别二维图形，它的网络结构对平移、比例缩放、倾斜或其他形式的变形具有高度不变性。卷积神经网络是近些年来发展迅速，备受器重的一种高效识别算法。它的应用范围也不仅仅局限于图像识别领域，也应用到了人脸识别、文字识别等方向。

2.1.2 卷积神经网络的重要组成部分

（1）卷积层

卷积神经网络的核心是卷积层，它产生了网络中大部分的计算。卷积层里最重要的组成就是卷积核。卷积核也可以理解为是一些滤波器集合。每个卷积核在空间上的尺寸都比较小，但是深度和输入数据是一致的。卷积核的大小是个超参数，可以自行选择。卷积核的内容，相当于上一节中全连接神经网络中药更新的权值w，即卷积核就是我们训练卷积神经网络要学习的东西。训练的过程实际上是在寻找能够反映训练数据特征的滤波器。以下图2-1为例，左边的是卷积层的输入image，中间的为卷积核（filter），右边的为卷积后产生的特征图谱（feature map）。卷积层卷积的过程是如图所示从输入的左上角开始，按照卷积核的大小3 3的方框括起相应的元素，与卷积核元素对应位置的元素作积后加和得到特征图谱的第一个元素。方框按一定步幅从左向右从上到下，依次完成卷积，即可得到特征图谱所有的内容。

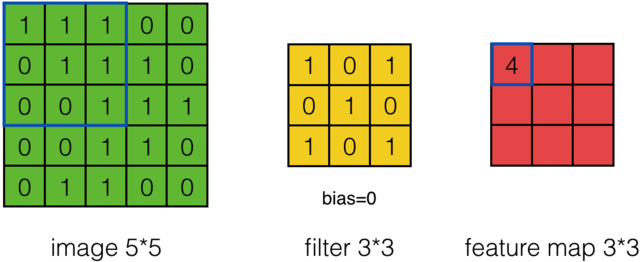


图2-1 卷积过程示意图

（2）子采样层

子采样层也称作池化层在有些文献中也称之为下采样层，在卷积层之间加入子采样层，可以有效减少输入网络中的数据大小，减小规模，控制网络中参数的数量，进而节约计算资源，减少训练所需时间。同时，采样层还能够有效地控制过拟合的出现。最常用的采样方式有两种，一种是Max Pooling，另外一种是Mean Pooling。Max Pooling 是指在算选定的N N尺寸中保留最大的那个作为采样后得样本值。Mean Pooling 是指在算选定的N N尺寸中取样本的平局值作为采样值。研究人员通过不断试验发现，使用MaxPooling的效果好于Mean Pooling，图2-2为2 \* 2 大小的Max Pooling采样过程示意。

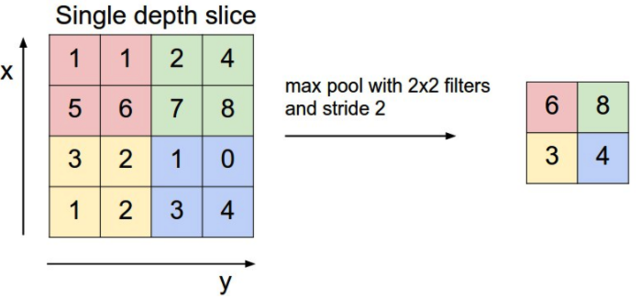


图2-2 下采样过程示意图

2.1.3 权值共享和局部连接

权值共享：图像的一部分的统计特性与其他部分是一样的。这也意味着在这一部分学习的特征也能用在另一部分上，所以对于这个图像上的所有位置，都能使用同样的学习特征。

局部连接：在处理图像这样的高维度输入时，让每个神经元都与前一层中的所有神经元进行全连接是不现实的。相反，让每个神经元只与输入数据的一个局部区域连接。该连接的空间大小叫做神经元的感受野（receptive field），它的尺寸是一个超参数（其实就是滤波器的空间尺寸）。在深度方向上，这个连接的大小总是和输入量的深度相等。需要再次强调的是，对待空间维度（宽和高）与深度维度是不同的：连接在空间（宽高）上是局部的，但是在深度上总是和输入数据的深度一致。

2.2 神经网络的前向传播和反向传播

所有神经网络在训练过程中都存在这两个过程，向前传播计算节点输出，反向传播更新连接权值。

2.2.1 神经元

神经元是组成神经网络的基本单位，如图2-3所示，神经元和感知器在结构上是基本相似的，同样的它们在激活函数上得不同，决定了其输出结果的不同。对于神经元来说，激活函数一般选择为sigmoid函数或者tanh函数或者REUL函数。传统的神经网络激活函数一般选择sigmoid函数或者双曲正切函数。sigmoid函数的值域是（0,1），其函数图像如图2-4所示。

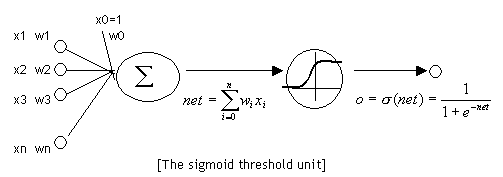


图2-3 神经元结构图

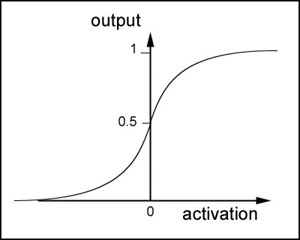


图2-4 sigmoid函数图

sigmoid函数的导数是：，可以看到sigmoid函数的导数可以用sigmoid函数自身来表示。方便计算，这也解释了早期的神经网络会选择sigmoid函数作为激活函数的原因，这是对于早期计算资源的一种妥协。

2.2.2 神经网络的连接形式

神经网络就是多个神经元按一定规则连接在一起。图2-5是一个简单的全连接神经网络。最左边的层叫做输入层，负责接收输入数据；最右边的层叫输出层，可以从这层获取神经网络输出数据；输入层和输出层之间的层叫做隐藏层，因为它们对于外部来说是不可见的。通过上图可以观察到，神经网络一般有以下结构规则：

（1）同一层的神经元之间没有连接。

（2）第N层的每个神经元和第N-1层的所有神经元相连(这就是full connected的含义)，第N-1层神经元的输出就是第N层神经元的输入。

（3）每个连接都有一个权值。



图2-5 全连接结构示意图

上面这些规则定义了全连接神经网络的结构。事实上还存在很多其它结构的神经网络，比如卷积神经网络(CNN)、循环神经网络(RNN)，他们都具有不同的连接规则。

2.2.3 神经网络的前向传播

神经网络的前向传播中，每一层的输入依赖于前一层的输出和两层两层之间连接的权值，如图2-6所示连接的箭头指向为前向传播方向。

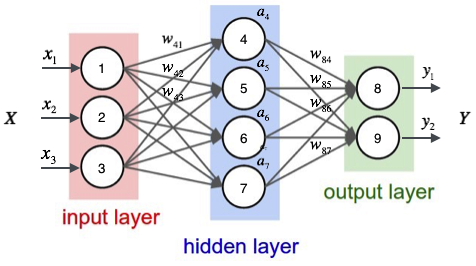


图2-6 全连接网络前向传播示意图

在图2-7中标出以数字1、2、3……对神经元进行编号，输入用x表示，输出用y表示，每条连接的权重用w表示，隐藏层的神经元输出用a表示。前向传播中，要计算当前层的节点输出必须得到前一层的输出，即前层的输出是后层的输入。以激活函数sigmoid为例，由输入层神经元1、2、3获取输入仍旧输出。如计算节点的输出,式中是节点的偏置项。同理可以得，，的输出值，得到了隐藏层所有节点的输出值后可以得到最后输出层的输出值，如。卷积神经网络所采用的前向传播方式与之相似，卷积层与其它层之间的权值在传播过程中改为卷积层的卷积核权值。

2.2.4 神经网络的反向传播算法

神经网络的反向传播是对网络层之间权值得更新过程，层与层之间的权值更新依赖于后一层的输出，反向传播的名字也由此而来。如图2-7所示，反向传播是有输出层向输入层的计算过程。

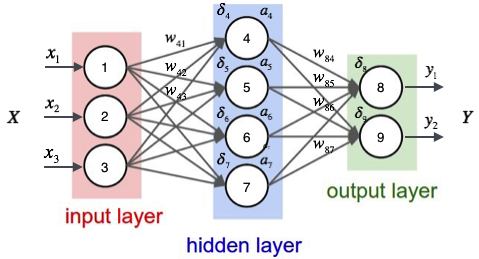


图2-7 全连接网络反向传播示意图

反向传播的计算方法以图2-7为例，在上一小节的基础上按序号标注出对应节点的误差项。反向传播算法，字如其名从反方向开始计算，对权值进行更新。在上一节中已经计算了全连接网络的每个节点的输出。这一节从输出层开始，依次计算输出层、和隐藏层每一节点的误差项。输出节点误差项，i对应相应的节点标号，t表示目标值，y表示节点实际输出值。隐藏节点误差项 ，其中是节点i到下一层节点k的连接权重。权重值得更新依赖于当前节点的输出和下一层对应节点的误差项，。

反向传播算法的本质是对链式求导法则的应用，上述的例子中sigmoid函数的求导可以用函数结果本身表示。

2.3 优化方法——梯度下降

梯度下降的方法是目前训练神经网络最常用的一种更新权值的计算方法，其巧妙应用了数学上利用导数求最值的概念。梯度下降具体细分为三种，批梯度下降、随机梯度下降、小批量梯度下降，本节以线性单元为例介绍梯度下降的计算过程。

线性单元结构如图2-8所示与感知器相比较，替换了激活函数，线性单元将返回一个实数值而不是0,1分类，因此线性单元可以用来解决回归问题，而非分类问题。

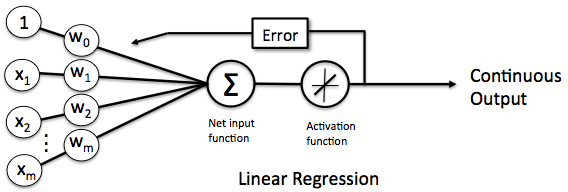


图2-8 线性单元结构图

2.3.1 批梯度下降

以线性单元为例，设置激活函数为，和感知器的输出结果相似的，因为激活函数得到什么值输出什么值，所以其输出为 其中仍旧等于b。

在数学上可以用两个数的方差来表示其相差程度的大小，在这里使用来训练数据表示通过激活函数后的输出值，用来表示训练数据所对应的真实标签值，则有：

（2-1）

把e称作单个样本误差，这里的是用来消除求导后系数2的，并不影响最终结果。当有N个训练数据时，用所有样本的误差和来表示模型的误差，即

（2-2）

将上式2代入式3中结合输出函数y化简后得

（2-3）

此时称之为目标函数，此函数是关于w的多元二次式。在数学上，我们要求一个函数的极值点，是对该函数进行求导，当时，求得极值点。对于计算机来说，它可以凭借超强的数据计算能力，通过一次一次怎加或减少x的值把函数的极值点试出来。

如图2-9所示，随机选取一个点，每次迭代更新为直到找到极小值点。这里要引入梯度的概念，梯度是指向函数值上升最快方向的一个向量。那么我们对梯度取反方向，就能找到函数值下降最快的方向。

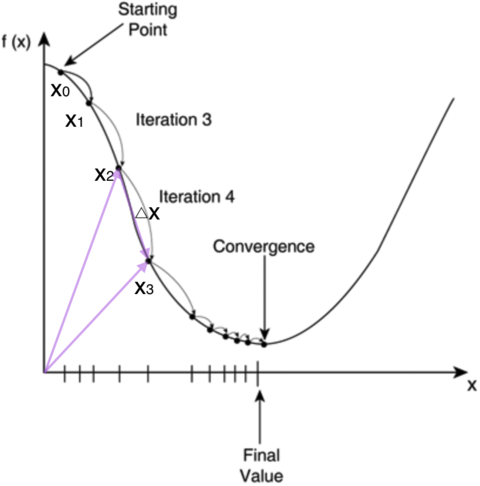


图2-9 梯度下降示意图

由此得出梯度下降算法的公式

（2-4）

对于目标函数来说则为

（2-5）

其中，，所以最后得

（2-6）

根据上式来更新w，每次要遍历训练数据中的所有样本进行计算，这种算法叫做批梯度下降。批梯度下降的不足之处在于面对数据量巨大的样本时，由于每次更新都学要遍历所有样本数据，纵使现在的计算机有再强大的计算能力，这也将使得因为计算量异常庞大，而让训练花费更多时间。虽然卷积神经网络的结构特点可以有效降低计算量，但从硬件性能上的限制考虑，本文不采用批梯度下降的方式对权值进行优化。

2.3.2 随机梯度下降

对于精度要求不高的的模型来说，可以使用随机梯度下降，随机梯度下降每次更新w值时只计算一个样本，由于样本的噪音和随机性，并不能保证每一次都是沿着下降的方向更新，但是总体看上去是沿着下降的方向更新的，最后收敛到最小值的附近。随机梯度下降大大提升了训练大规模样本数据的效率。尽管随机梯度下降可以大幅提高模型训练效率，但其波动较大，在有限迭代次数内不能得到稳定的训练结果，故本文不采用该方法对权值进行更新。

2.3.3 小批量梯度下降

随机梯度下在迭代的过程中并不是每次都朝着整体优化的方向，在迭代开始的时候可以很快收敛，但是训练一段时间后收敛会变得很慢。小批量梯度下降结合了批梯度下降和随机梯度下降的优点，使算法的训练过程在提高速度的同时，也保证了最终参数的准确率。本文采用小批量梯度下降的方法对权值进行更新，通过设置合理的批量大小和迭代次数能够在短时间内使模型快速得到优化。

2.4 小结

本章从卷积神经网络的组成结构、模型权值共享和局部连接的特点、网络的前向反向传播过程、以及模型的优化方法等基本原理作以分析。在后续的章节里，将理论应用于实践。

3 Keras深度学习框架

Keras使用简单易上手，只要有Python编程经验即可快速将理论付诸于实现。本文使用深度学习Keras来实现LeNet5的经典模型结构，并在实现经典结构的基础上改变参数，探究不同参数对模型训练的影响，本章对Keras的使用和配置作以介绍。

3.1 Keras简介

Keras是一个有Python编写而成以TensorFlow或Theano 作为后台的深度学习框架。当前作为深度学习框架对于GPU运算的支持是必不可少的，Keras同时支持CPU和GPU。Keras在linux下安装起来非常方便，不像caffe那样需要各种各样的支持库，有一点问题就得重新编译安装。Keras对用户的使用体验支持的相当好，它高度模块化的同时还具有可扩充性。因为是用Python编写的，所以Keras同时适用于Python2和Python3两个版本。Keras的使用更像是搭积木，深度学习里大多数你需要的东西，如网络层、激活函数、优化器、等，它都有对应的API可以进行直接调用。Keras没用单独的模型配置文件，模型的配置、运行、保存都可以写在同一个Python文件中。

3.2 Keras编程

3.2.1 Keras模型构建

Keras提供了两种构建模型的方法，一种是序贯模型，另一种是函数式模型。函数式模型应用的范围更加广泛，确切的说序贯模型是函数式模型的一种特殊情况。因此两种模型在有些API的方法上是一样的。序贯模型最容易上手操作，它就像盖楼房那样，从第一层，一层一层按照你所设计的模型结构按顺序从输入层到输出层，依次进行声明即可。这也就意味着模型只有层与层之间的关系，不能存在跨层连接。函数式模型则没有对跨层连接的限制。

3.2.2 Keras常用函数及用法

本文使用的是序贯模型，主要用到的函数有下面这些。

（a）要开始使用序贯模型建立网络，需要先在Python文件的头部引入Sequential模块，创建Sequential的对象。

（b）为模型添加层，通过创建的对象调用Sequential的add方法依次添加层。使用add添加的第一层需要指定有关输入数据shape的参数，而其后层的可以自动推导出中间数据的shape。

（c）对训练过程进行配置，使用compile，接收三个参数，分别是优化器、损失函数、指标列表。

（d）开始训练，使用fit函数，其参数有

x：输入数据。如果模型只有一个输入，那么x的类型是numpy array，如果模型有多个输入，那么x的类型应当为list，list的元素是对应于各个输入的numpy array

y：标签，numpy array

batch\_size：整数，指定进行梯度下降时每个batch包含的样本数。训练时一个batch的样本会被计算一次梯度下降，使目标函数优化一步。

epochs：整数，训练的轮数，每个epoch会把训练集轮一遍。

verbose：日志显示，0为不在标准输出流输出日志信息，1为输出进度条记录，2为每个epoch输出一行记录

callbacks：list，其中的元素是Keras.callbacks.Callback的对象。这个list中的回调函数将会在训练过程中的适当时机被调用。

validation\_split：0~1之间的浮点数，用来指定训练集的一定比例数据作为验证集。验证集将不参与训练，并在每个epoch结束后测试的模型的指标，如损失函数、精确度等。注意，validation\_split的划分在shuffle之前，因此如果你的数据本身是有序的，需要先手工打乱再指定validation\_split，否则可能会出现验证集样本不均匀。

validation\_data：形式为（X，y）的tuple，是指定的验证集。此参数将覆盖validation\_spilt。

shuffle：布尔值或字符串，一般为布尔值，表示是否在训练过程中随机打乱输入样本的顺序。若为字符串“batch”，则是用来处理HDF5数据的特殊情况，它将在batch内部将数据打乱。

class\_weight：字典，将不同的类别映射为不同的权值，该参数用来在训练过程中调整损失函数（只能用于训练）。

sample\_weight：权值的numpy array，用于在训练时调整损失函数（仅用于训练）。可以传递一个1D的与样本等长的向量用于对样本进行1对1的加权，或者在面对时序数据时，传递一个的形式为（samples，sequence\_length）的矩阵来为每个时间步上的样本赋不同的权。这种情况下请确定在编译模型时添加了sample\_weight\_mode='temporal'。

initial\_epoch: 从该参数指定的epoch开始训练，在继续之前的训练时有用。

3.3 Keras环境配置

Keras在Ubuntu下的环境配置如下,下列命令均在终端中执行。

（1）系统升级

Sudo apt update

Sudo apt upgrade

（2）安装Python 基础开发包

sudo apt install -y python-dev python-pip python-nose gcc g++ git gfortran vim

（3）安装运算加速库

sudo apt install -y libopenblas-dev liblapack-dev libatlas-base-dev

（4）Keras及相关开发包安装

sudo pip install -U --pre pip setuptools wheel

sudo pip install -U --pre numpy scipy matplotlib scikit-learn scikit-image

sudo pip install -U --pre tensorflow-gpu

sudo pip install -U --pre tensorflow

sudo pip install -U --pre Keras

（5）打开Python解释器 import TensorFlow import Keras 无报错则完成安装

3.4 小结

本章主要对本文采用的深度学习框架Keras两种模型建立方法贯序模型和函数模型、模型的编译和训练所用到的函数参数、使用Keras环境配置中所需的Python基础开发包以及后台TensorFlow的安装作以介绍。本文在下一章中使用Keras实现经典LeNet5结构，并在其基础上作以改动探究不同参数对模型识别率的影响，学习调参经验。

4 经典LeNet-5实验探究

本章以MNIST数据集作为训练和测试数据来源，使用Keras深度学习框架在实现LeNet5的基础上对LeNet5模型结构进行改动以探究不同参数对模型识别率的影响，总结调参经验。

4.1 数据集MNIST介绍

MNIST[7]数据集是一个手写体数字数据集，通过了解这个数据集由四个部分组成。可以分为两大类，一类是训练数据，另一类是测试数据。每种数据包含了数据本身和对应的数据标签。训练集中有60000个用例，测试集中有10000个用例。

MNIST是由NIST的手写体数字二值化图片的数据库SD3和SD1构成的。NIST最初的设计中，SD3是训练集，SD1是测试集。然而SD3比SD1更清晰，并且更容易识别。原因是SD3是从人口统计局的雇员收集的，而SD1是从中学的学生中收集的。从学习经验中得到有效的结论需要测试集是独立于训练集，并且测试集在完整的样本之中。因此，有必要通过混合NIST的数据来建造一个新的数据库。

MNIST的训练集是由从SD3中的3万张图片和SD1中的3万张图片组成的。测试集是由从SD3中的5000张图片和SD1中的5000张图片。6万张训练集包含了近似250位写手。并且保证了训练集和测试集的写手是不相交的。

SD1包含了由500位不同的写手写的58527张图片。比较而言，SD3的数据块是一次排列的，而SD1的数据是杂乱无章的。可以识别出SD1中的写手信息，我们根据识别出来的信息，把500位写手的数据分为两部分，前250位分到训练集中，后250位分到测试集。这样训练集和测试集我们现在都有大概3万张图片。在训练集中再加入SD3的数据，从0开始，使其凑够6万。类似的，在测试集中从SD3第35000张图开始，补充测试集到6万张。

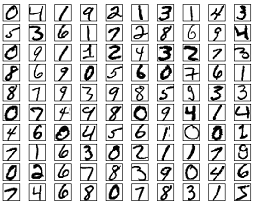


图4-1 MNIST部分数据展示图

4.2 LeNet-5实现

4.2.1 LeNet-5介绍

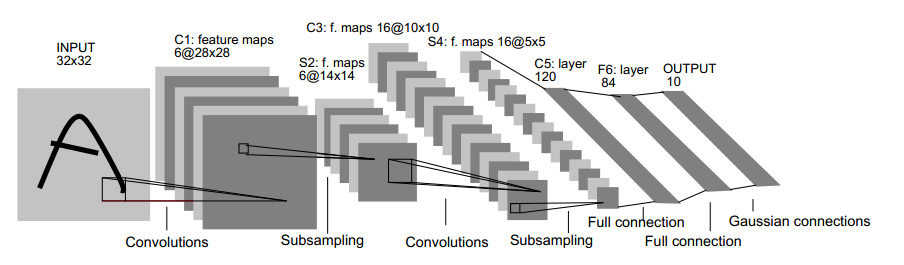


图4-2 LeNet5结构图

经典LeNet-5[9]包含输入输出层共计有8层，除输入输出两层外的每一层都包含需要通过训练学习的参数。每层有多个特征图谱，每个特征图谱都是通过一种卷积核（滤波器）卷积体现输入的一种特征，每个特征图谱有多个神经元。

C1层是第一个卷积层，得到的时32\*32的数据，包含了6中不同的5\*5大小的卷积核，输出28\*28的特征图谱。可训练参数共计个（每个卷积核有25个参数和一个偏置项参数，共6个卷积核）

S2层是一个下采样层，输入为28\*28的，采样大小为2\*2,通过采样后得到6个14\*14的图，加上偏置项每个特征图谱有2个可训练参数，共计12个。

C3层是第二个卷积层，有16种5\*5的卷积核，C3的前6个特征图以S2中3个相邻的特征图子集为输入。接下来6个特征图以S2中4个相邻特征图子集为输入。然后的3个以不相邻的4个特征图子集为输入。最后一个将S2中所有特征图为输入。输出16个10\*10的特征图谱。

S4层是下采样层，采样大小为2\*2，通过采样得到16个5\*5的图。这层共计32个训练参数。

C5层是卷积层，卷积核大小5\*5，卷积过后形成共计120种卷积结构。每个都与S4的16个输出相连。

F6层是全连接层，84个节点，共计10164个可训练参数。

4.2.2 LeNet实现

原LeNet-5采用的是32\*32的图片数据大小，本文采用的是MNIST数据集，该数据集的单张图片大小为28\*28。本文在不改变LeNet-5的整体结构下将LeNet加以实现，模型细节与原paper作者有细微不同,保留了整体层次结构，因为数据输入尺寸大小的不同导致经过两次下采样后的特征图谱尺寸小于5\*5无法进行卷积，所以改变C5层卷积核的大小为3\*3。卷积层采用的权值初始化方法采用Keras官方推荐的截尾高斯分布初始化，激活函数采用sigmoid函数。最后得全连接层权值初始化采用正态分布初始化方法。所有初始化方法的随机种子采用固定值，方便结果重现和随后探究中控制单一变量进行结果比较。

代码实现如下:

# 输入数据的维度

img\_rows, img\_cols = 28, 28

# 使用的卷积滤波器的数量

nb\_filters = 6

# 用于 max pooling 的池化面积

pool\_size = (2, 2)

# 卷积核的尺寸

kernel\_size = (5, 5)

# C1 卷积层1 卷积核6个 5\*5

model.add(Convolution2D(nb\_filters, kernel\_size[0], kernel\_size[1],

border\_mode='valid',

input\_shape=input\_shape, kernel\_initializer='TruncatedNormal'))

model.add(Activation('sigmoid'))

# S2 下采样

model.add(MaxPooling2D(pool\_size=pool\_size))

# C3卷积层2 16个卷积核 5\*5

model.add(Convolution2D(

16, kernel\_size[0], kernel\_size[1], kernel\_initializer='TruncatedNormal'))

model.add(Activation('sigmoid'))

# S4 下采样

model.add(MaxPooling2D(pool\_size=pool\_size))

# C5 卷积层3 120个卷积核 3\*3

model.add(Convolution2D(120, 3, 3, kernel\_initializer='TruncatedNormal'))

model.add(Activation('sigmoid'))

# 转化为一维

model.add(Flatten())

# F6 全连接层 输出层

model.add(Dense(nb\_classes, kernel\_initializer='random\_normal'))

model.add(Activation('softmax'))

模型训练采用克服了批梯度下降和随机梯度下降缺点的小批量梯度下降的参数更新方法。Keras在不设定的情况下，默认对训练数据进行打乱，因为所采用的平台硬件资源限制，该模型采用迭代40批次，每批次更新使用128个样本。

该模型完成训练后测试准确率达到了98.58%，每次迭代训练和测试率的识别率和损失变化曲线如图4-3所示。

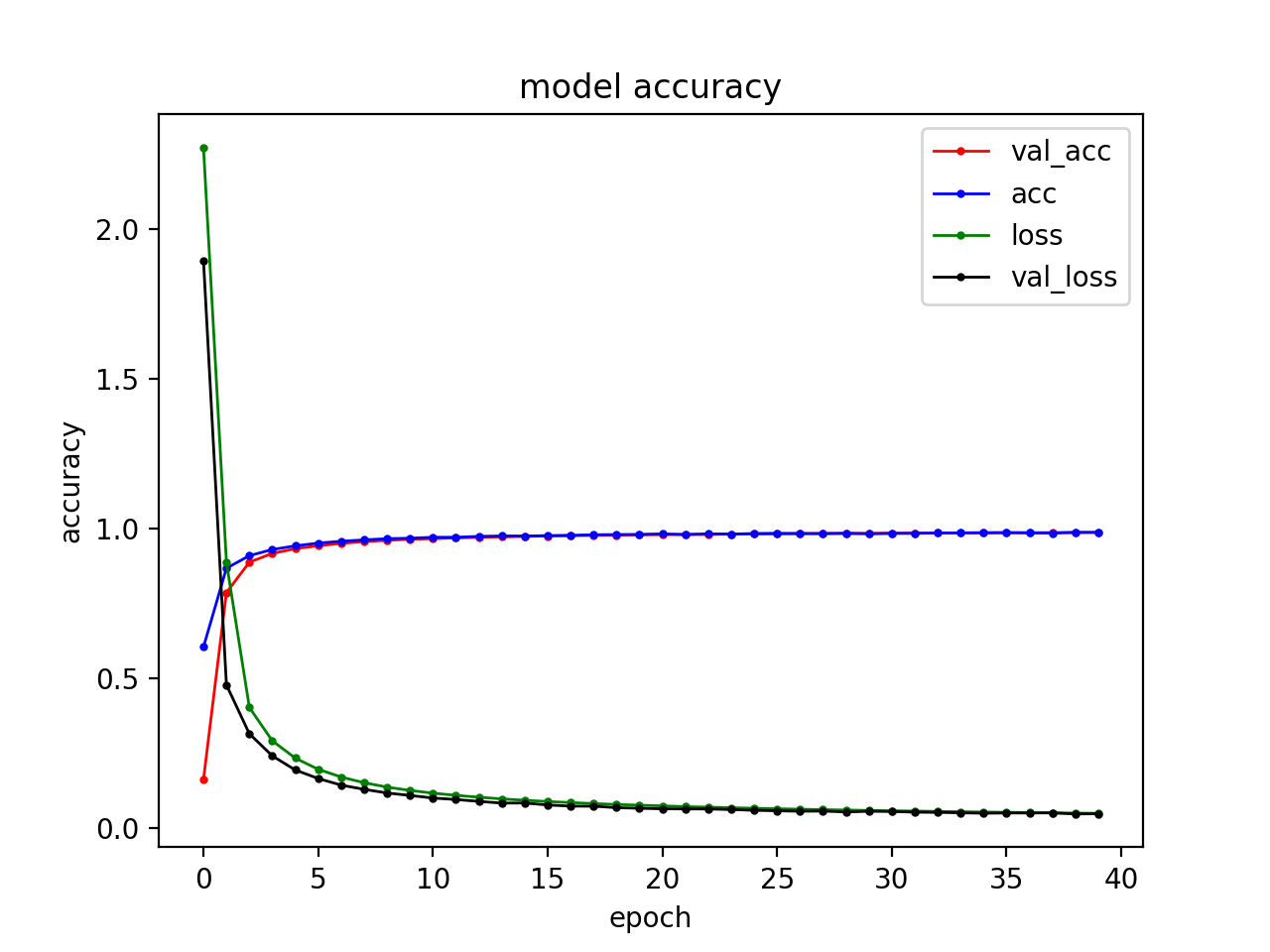


图4-3 LeNet 识别率损失折线图

4.3 模型探究

本文将在使用Keras实现经典结构LeNet-5的同时，控制单一变量，在LeNet-5的基础上分别从改变网络结构，卷积核大小和数量、权值初始化方式、激活函数选择这几方面与LeNet5进行比较试验，以掌握一定的调参方法和规则。

4.3.1 不同网络结构

（1）实验设计

在上一节实现的LeNet5的基础上，对模型结构进行改动。直接在原结构S4层的基础上增加5\*5的卷积层会导致因为前四层两次下采样而使S4层的输出特征图谱尺寸大小小于5\*5无法再次进行卷积，所以CNN1-1直接以新的卷积层代替了S4层的下采样层；CNN1-2在原Lenet5结构上在C5层后添加了按百分之五的概率断开输入神经元的Drop层；CNN1-3在原LeNet5的结构基础上在C5层后添加了新的全连接层。通过使用改变网络结构，探究不同结构的组合对识别率的影响，不同结构对比如表4-1所示。

表4-1 网络模型结构对比

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 网络名称 | 层结构 | | | | | | |
| LeNet5  CNN1-1  CNN1-2  CNN1-3 | Con  Con  Con  Con | Pooling  Pooling  Pooling  Pooling | Con  Con  Con  Con | Pooling  Con Pooling  Pooling | Con  Con  Con  Con | Full  Full  Drop Full | /  /  Full  Full |

（2）实验结果

表4-2为不同模型结构经过40轮迭代后，使用测试集测试后的识别率大小，原LeNet5结构的识别率为98.58%，CNN1-1的识别率为98.05%，CNN1-2的识别率为98.49%，CNN1-3的识别率为98.19%。

表4-2 不同网络结构经过训练后的网络识别率

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 网络名称 | LeNet5 | CNN1-1 | CNN1-2 | CNN1-3 |
| 测试识别率 | 98.58% | 98.05% | 98.49% | 98.19% |

如图4-4所示为不同结构与LeNet5识别准确率对比折线图，通过对折线图的分析比较，可以看出原LeNet5的识别准确率起点最高，添加了Drop层的CNN1-2其次，添加了卷积层的CNN1-1要通过迭代2次后识别率大幅上升，添加全链接层的CNN1-3因为增加了全连接导致的链接权值数量增加需要迭代5次后才能大幅提高识别的准确率。

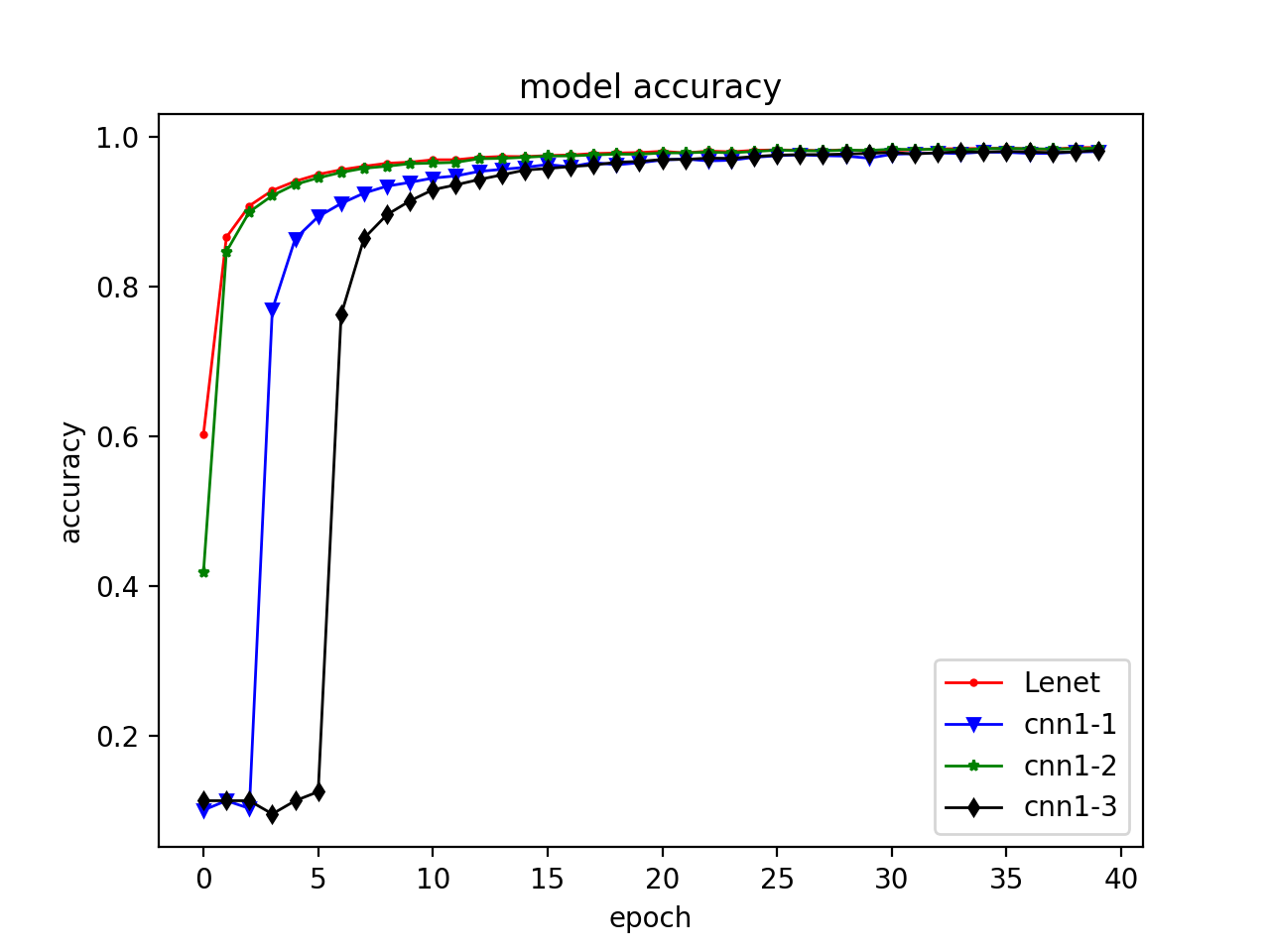


图4-4 不同模型结构识别率折线图

如图4-5所示为不同结构模型与LeNet5损失对比折线图，几种不同模型的损失率与识别率情况相对应，总体上均能在10次迭代以内快速收敛。

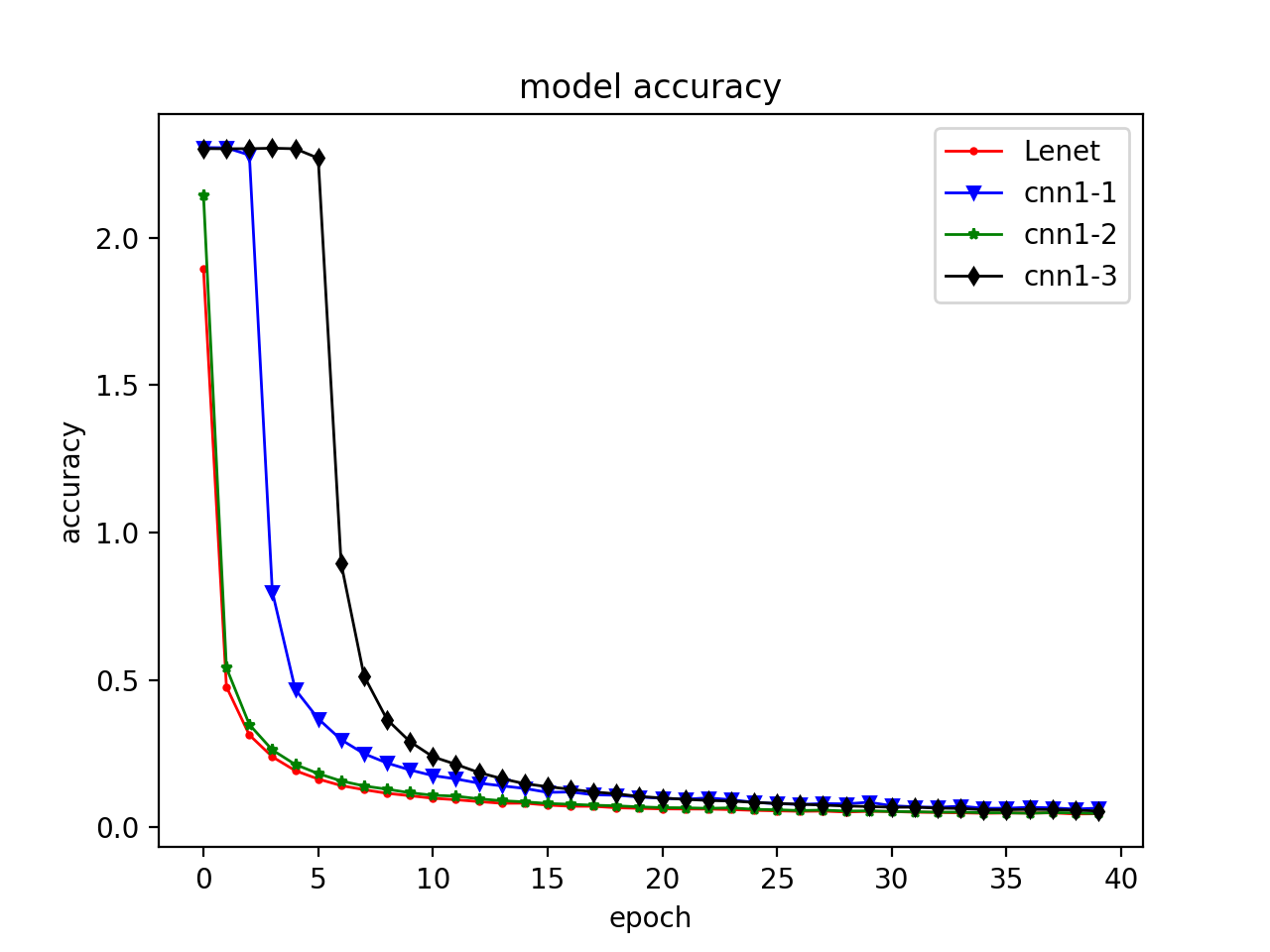


图4-5 不同模型结构损失折线图

通过表的比较，可以看出不论是添加卷积层、全连接层、还是Drop层，在相同其他条件下，特别是均迭代更新40次的前提下，CNN1-1、CNN1-2、CNN1-3的识别准确率并没有比原LeNet5模型准确率高，均有不同程度的下降。结论是在有限迭代次数内，添加卷积层、Drop层和全连接层，会不同程度降低对测试集识别率，原LeNet5模型结构经典，识别率最高。

4.3.2 卷积核大小数量

（1）实验设计

在LeNet5的基础上进行改动，主要改动卷积核大小和数量。CNN2-1在原LeNet5的基础上将C1和C3层卷积核尺寸大小由5\*5改为3\*3；CNN2-2在原LeNet5的基础上将C3层的卷积核数量由6个改为16个。通过改变卷积核参数，探究卷积核对模型识别率的影响，不同卷积层参数对比如表4-3所示。

表4-3 卷积层参数对比

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 网络名称 | C1 | S2 | C3 | S4 | C5 | F6 |
| LeNet5  CNN2-1  CNN2-2 | 6\*(5\*5)  6\*(3\*3)  6\*(5\*5) | （2\*2）  （2\*2）  （2\*2） | 6\*(5\*5)  6\*(3\*3)  16\*(5\*5) | （2\*2）  （2\*2）  （2\*2） | 120(3\*3)  120(3\*3)  120(3\*3) | 10  10  10 |

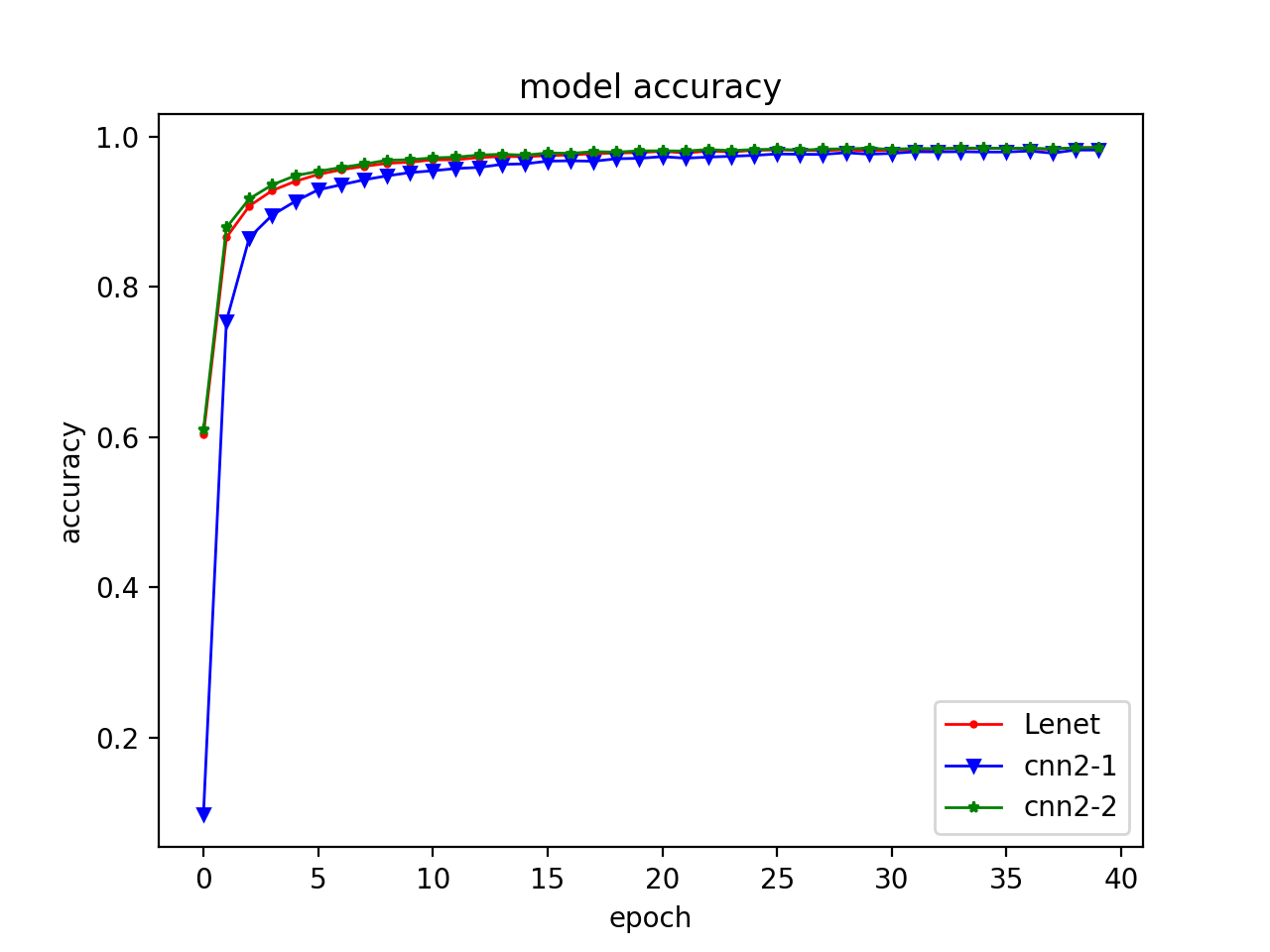
（2）实验结果

表4-4为相同模型结构下不同卷积层参数经过40轮迭代后，使用测试集测试后的识别率大小，原LeNet5结构的识别率为98.58%，CNN2-1的识别率为98.25%，CNN2-2的识别率为98.57%。

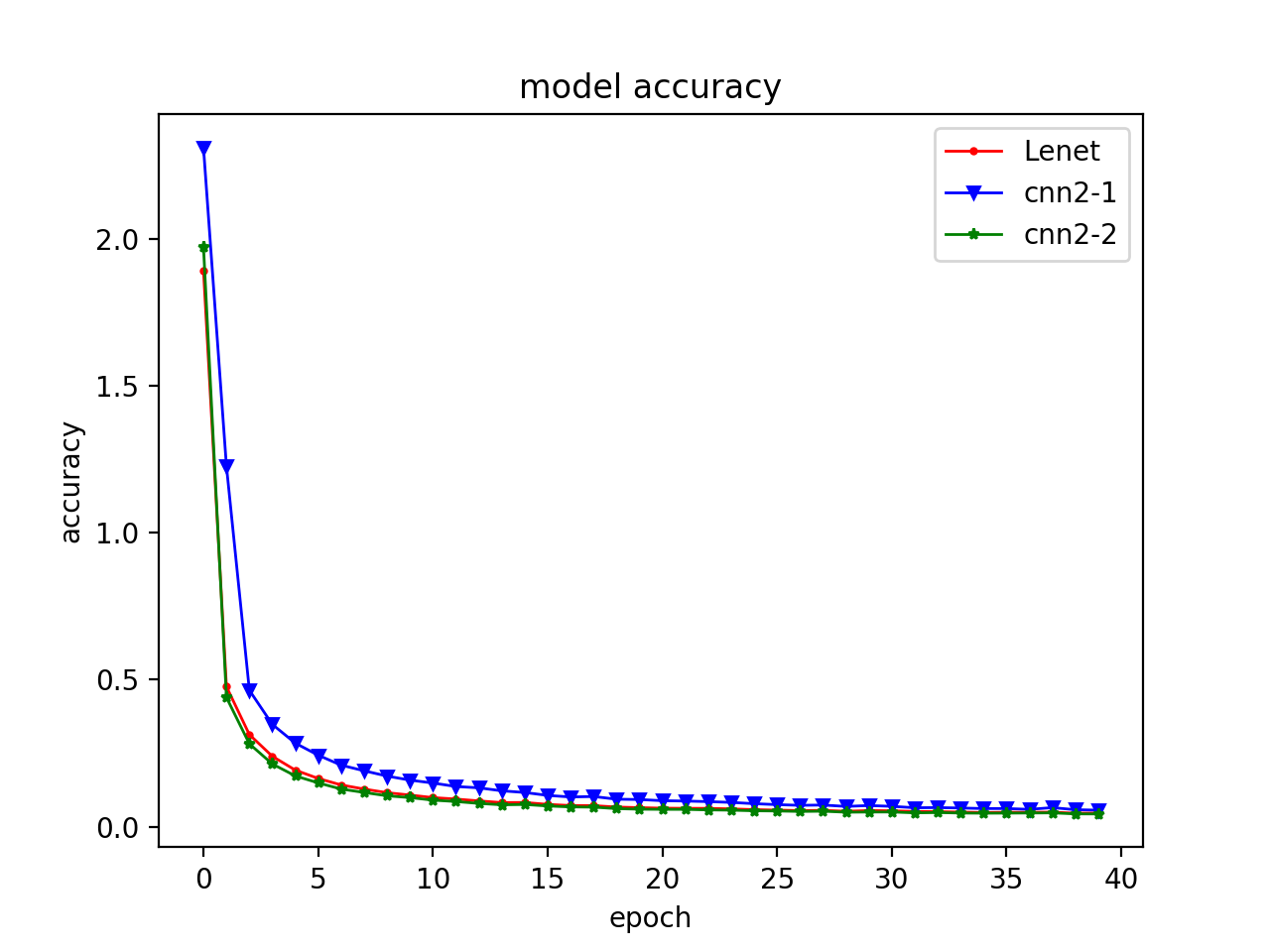
表4-4 卷积层不同参数经过训练后的网络识别率

|  |  |  |  |
| --- | --- | --- | --- |
| 网络名称 | LeNet5 | CNN2-1 | CNN2-2 |
| 测试识别率 | 98.58% | 98.25% | 98.57% |

如图4-6所示为卷积层不同参数模型识别率变化折线图，通过对折线图的分析，减小卷积核尺寸的大小，会使识别率收敛的起点降低。增加卷积核的数量会使识别率收敛的更快。

图4-6 卷积层不同参数模型识别率变化折线图

如图4-7所示为卷积层不同参数模型损失变化折线图，通过对折线图的分析，减小卷积核尺寸的大小，会使损失收敛的起点升高；增加卷积核的数量会使损失收敛的更快。

图4-7 卷积层不同参数模型损失变化折线图

通过对测试识别率结果进行分析，CNN2-1与LeNet5对比，在其他条件不变的前提下，减小卷积核的大小，明显会使识别率明显下降。CNN2-2与LeNet5对比，在其他条件不变的前提下，增加卷积核的数量，对识别率的影响几乎可以忽略不计，仅降低了0.01%。结论是增加卷积核的数量可以有效提取更多训练集样本的特征，从而提高识别率的收敛速度。

4.3.2 权值初始化

（1）实验设计

如表所示，原LeNet5卷积层采用的权值初始化方式是Keras推荐滤波器使用的TruncatedNormal，CNN3-1采用的是权值全0初始化方式；CNN3-2采用的是权值全1初始化方式；CNN3-3采用的是；CNN3-4采用的是。通过改变卷积层的权值初始化方式，探究不同初始化方式对模型识别率收敛速度和准确率的影响，卷积层不同权值初始化方式对比如表4-5所示。

表4-5 卷积层不同权值初始化方式对比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 网络名称 | LeNet5 | CNN3-1 | CNN3-2 | CNN3-3 | CNN3-4 |
| 权值初始化方式 | TruncatedNormal | Zero | Ones | RandomNormal | RandomUniform |

（2）实验结果

表4-6为相同模型结构下不同卷积层权值初始化方式经过40轮迭代后，使用测试集测试后识别率大小，原LeNet5结构的识别率为98.58%，CNN3-1的识别率为97.28%，CNN3-2的识别率为8.92%，CNN3-3的识别率为98.41%，CNN3-4的识别率为98.38%。

表4-6 卷积层不同权值初始化方式经过训练后的模型识别率

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 网络名称 | LeNet5 | CNN3-1 | CNN3-2 | CNN3-3 | CNN3-4 |
| 测试识别率 | 98.58% | 97.28% | 8.92% | 98.41% | 98.38% |

如图4-8所示为卷积层不同权值初始化方式模型识别率折线图，通过对折线图的分析，在其他参数不改变的前提下，CNN3-1的识别率需要经过6次迭代后才快速收敛；CNN3-2的识别率，在40次迭代中一直没有收敛。CNN3-3的识别率与LeNet5识别率收敛过程几乎一致，仅收敛起点比LeNet5略高；CNN3-4的识别率也与LeNet5识别率收敛过程极其相似。

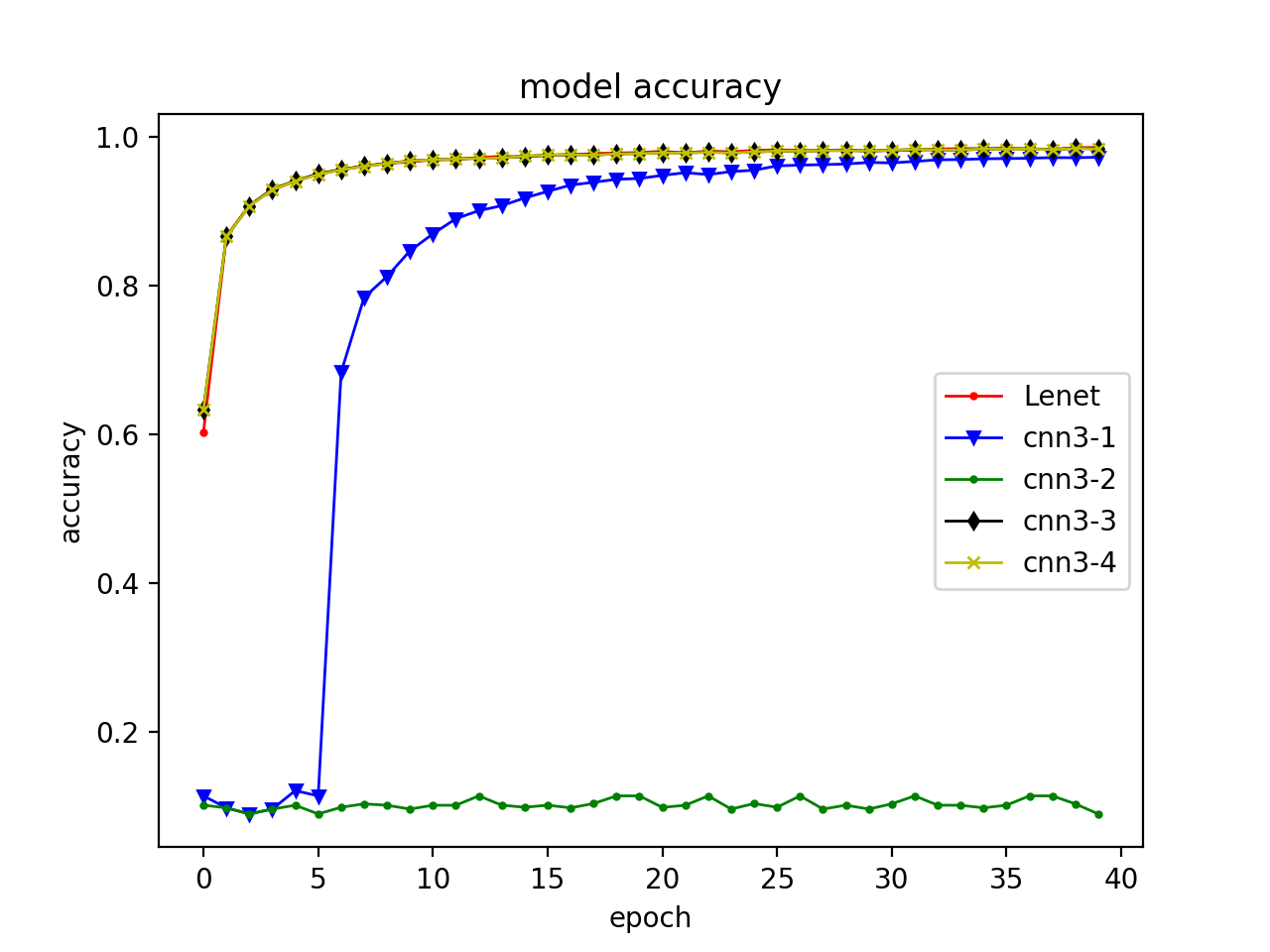


图4-8 卷积层不同权值初始化方式模型识别率折线图

如图4-9所示为卷积层不同权值初始化方式模型损失折线图，通过对折线图的分析，

与识别率的收敛过程相似，采用全1的初始化方式损失居高不下。

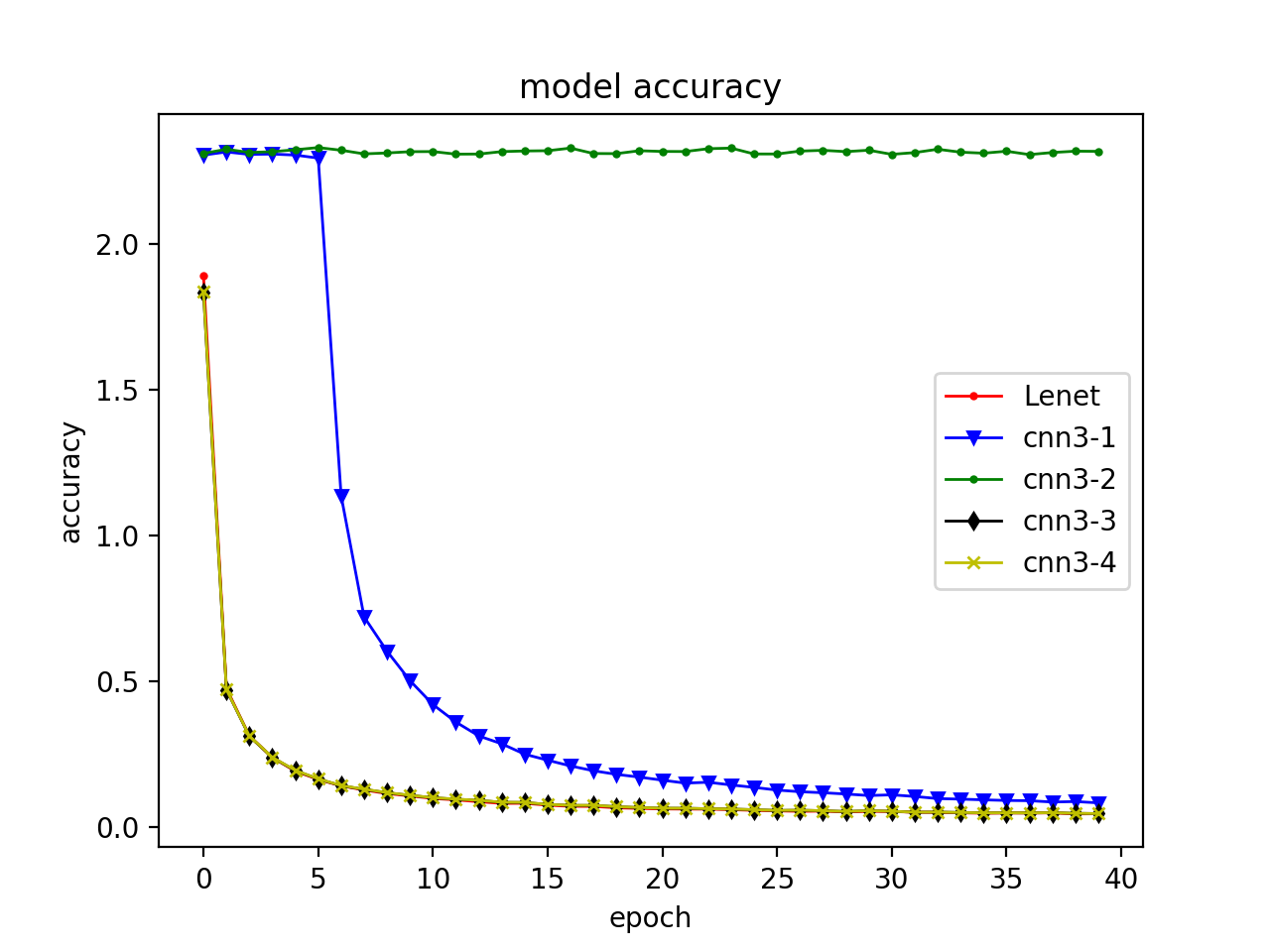


图4-9 卷积层不同权值初始化方式模型损失折线图

通过对表4-6的分析，CNN3-1卷积层采用权值全0初始化，相比LeNet5在识别准确率上1.3个百分点；CNN3-2卷积层采用全1初始化，相比LeNet5在识别率上大幅降低；CNN3-3卷积层采用初始化方式，相比LeNet5在识别准确率上有略微降低；CNN3-4卷积层采用初始化方式，相比LeNet5在识别准确率上也略微降低。结论是不同于LeNet5的权值初始化方式在其他参数条件不变的前提下会使模型识别率有所降低。在迭代次数规模较小的情况下，不能采用全1对权值进行初始化。

4.3.3 激活函数选择

（1）实验设计

如表所示，LeNet5卷积层采用的是Sigmoid的函数，CNN4-1采用的是Tanh函数，CNN4-2采用的是Reul函数，CNN4-3采用的是Softplus函数，卷积层不同激活函数对比如表4-7所示。

表4-7 卷积层不同激活函数对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 网络名称 | LeNet5 | CNN4-1 | CNN4-2 | CNN4-3 |
| 卷积层激活函数 | Sigmoid | Tanh | Reul | Softplus |

（2）实验结果

表4-8为相同模型结构卷积层使用不同激活函数经过40轮迭代后，使用测试集测试后识别率大小，原LeNet5结构的识别率为98.58%，CNN4-1的识别率为98.89%，CNN4-2的识别率为98.44%，CNN4-3的识别率为98.62%。

表4-8 卷积层不同激活函数经过训练后的模型识别率

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 网络名称 | LeNet5 | CNN4-1 | CNN4-2 | CNN4-3 |
| 测试识别率 | 98.58% | 98.89% | 98.44% | 98.62% |

如图4-10所示为卷积层不同激活函数模型识别率折线图，通过对折线图的分析，采用tanh函数作为卷积层激活函数的CNN4-1识别率的收敛起点相当高，通过第一次迭代就已经达到90%以上；采用reul函数作为卷积层激活函数的CNN4-2识别率收敛起点币LeNet5低，但在经过一次迭代后迅速收敛，收敛速度大于LeNet5；采用新激活函数softplus作为卷积层激活函数的CNN4-3，收敛起点和速度都高于LeNet5。

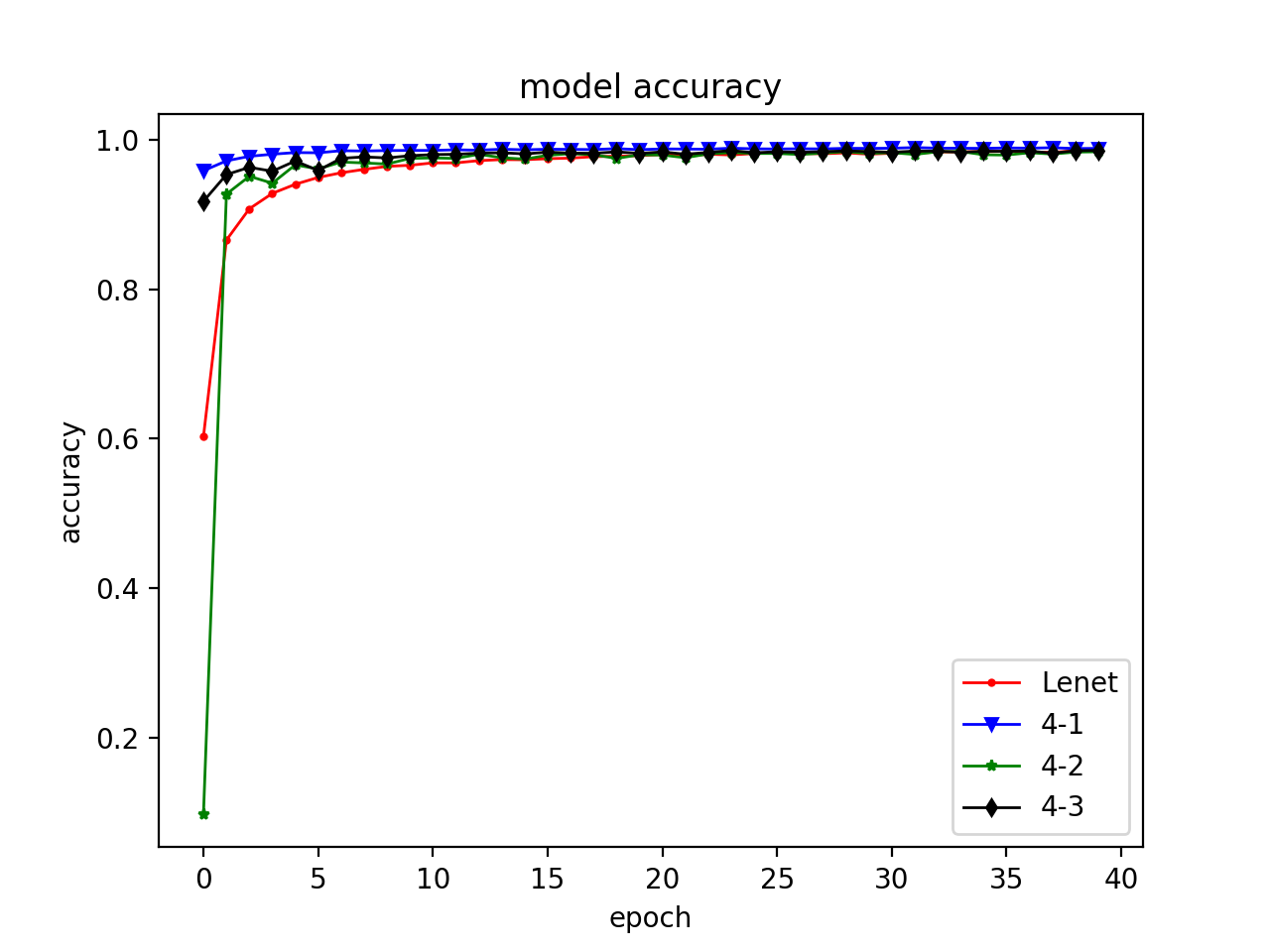


图4-10 卷积层不同激活函数模型识别率折线图

如图4-10所示为卷积层不同激活函数模型识别率折线图，通过对折线图的分析，卷积层采用Tanh和Softplus作为激活函数的模型损失收敛起点低，卷积层采用Reul函数作为激活函数的模型损失收敛起点高。

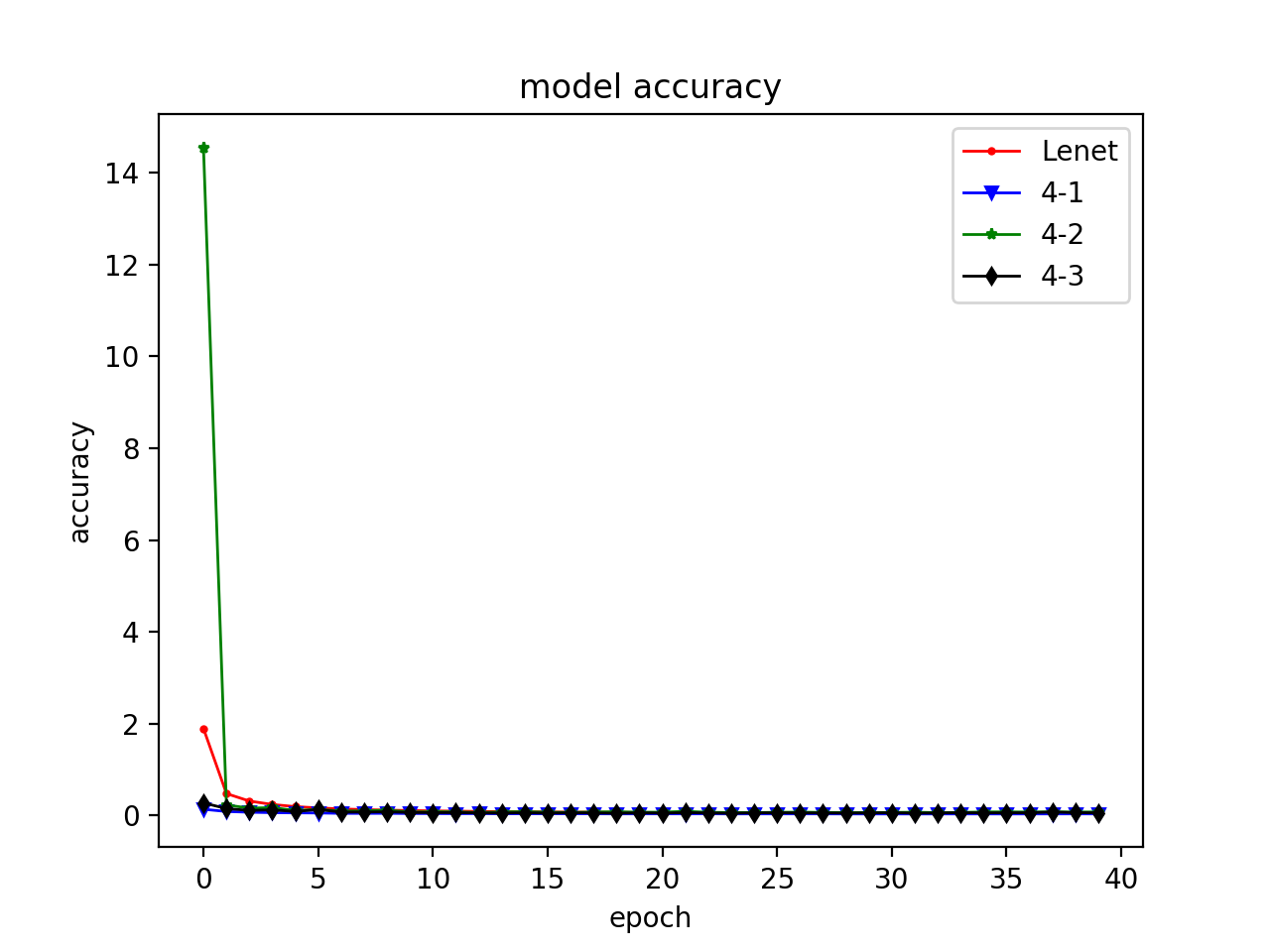


图4-11 卷积层不同激活函数损失折线图

通过表分析可知，在其他参数不改变的前提下，采用tanh函数作为卷积层激活函数的CNN4-1比卷积层采用sigmoid函数的LeNet5在识别率上高出了0.31%。这是出乎意料的结果。CNN4-2采用的是最近学术界推荐替代sigmoid和tanh的reul函数，理论上来说使用reul的效果应该会更好，出乎意料的是它比卷积层采用sigmoid函数的LeNet5识别率低了0.14%；CNN4-3采用的是近期出现的新函数softplus，其识别率比LeNet5高0.04%。结论是在有限迭代次数内，卷积层采用tanh或softplus，能够有效提高识别准确率，并提高识别率收敛的速度。

4.4 小结

本章通过深度学习框架Keras实现了以MNIST数据集作为训练数据的经典LeNet5结构，并通过设计实验探究得出以下结论，LeNet5模型在迭代次数为40次的前提条件下，添加卷积层、drop层、全连接层都会对模型识别率产生负影响；增加卷积层中的卷积核数量可以有效提高模型识别率收敛的速度；在有迭代限次数内采用全1的卷积层权值初始化方式会对模型识别率产生巨大的影响；卷积层激活函数采用Tanh函数可以有效提高模型识别率。

5 手写数字识别算法应用实践

本文将用训练好的高识别率手写数字模型应用到摄像头实现对镜头画面中数字的实时识别。本章将介绍如何使用计算机视觉库OpenCV调用电脑摄像头、找到帧画面中的数字并对数字进行识别前的处理，最后调用训练好的手写数字模型将识别结果在原帧画面中显示出来。

5.1 OpenCV图像处理部分

5.1.1 OpenCV介绍

OpenCV是一个开源的跨平台计算机视觉库，它可在Linux、Windows和Mac OS操作系统上运行。因为由C和C++编写，所以它很高效，并且提供了许多其他语言的接口如Python、Ruby、MATLAB等。OpenCV的功能有基本的图像处理如去燥、边缘检测、角点检测、色彩变化等。

本文将运用部分OpenCV的功能，调取电脑自带的摄像头，对每一帧的画面进行处理，在画面中找到数字并截取出来，调用数字识别模型进行识别，并用矩形框出每个数字把识别结果标注在旁边。

5.1.2 OpenCV安装配置

（1）安装依赖包

依赖包包含了linux程序在安装、运行中所必须的相关编译工具、安装工具等。程序本身并不包含这些需要的库，在不重复造轮子的前提下，程序在运行中相应的功能时会需要调用相应的库。如本文所使用的OpenCV需要对Python不同版本的支持、对图片视频读写的支持等。

sudo apt-get install --assume-yes build-essential cmake git

sudo apt-get install --assume-yes build-essential pkg-config unzip ffmpeg qtbase5-dev python-dev python3-dev python-numpy python3-numpy

sudo apt-get install --assume-yes libOpenCV-dev libgtk-3-dev libdc1394-22 libdc1394-22-dev libjpeg-dev libpng12-dev libtiff5-dev libjasper-dev

sudo apt-get install --assume-yes libavcodec-dev libavformat-dev libswscale-dev libxine2-dev libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev

sudo apt-get install --assume-yes libv4l-dev libtbb-dev libfaac-dev libmp3lame-dev libopencore-amrnb-dev libopencore-amrwb-dev libtheora-dev

sudo apt-get install --assume-yes libvorbis-dev libxvidcore-dev v4l-utils

（2）编译

在当前目录下创建build文件夹，进入build文件夹，配置编译选项，最后执行编译命令。

mkdir build

cd build/

cmake -D CMAKE\_BUILD\_TYPE=RELEASE -D CMAKE\_INSTALL\_PREFIX=/usr/local -D WITH\_TBB=ON -D WITH\_V4L=ON -D WITH\_QT=O -D WITH\_OPENGL=ON -D WITH\_CUBLAS=ON -D CUDA\_NVCC\_FLAGS="-D\_FORCE\_INLINES"

make -j $(($(nproc) + 1))

（3）安装

以root权限执行安装命令make install，利用输出重定向，创建OpenCV相关动态链接库配置文件，使用动态库管理命令ldconfig，让OpenCV的相关链接库被系统共享，最后更新系统源。

sudo make install

sudo /bin/bash -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/OpenCV.conf'

sudo ldconfig

sudo apt-get update

（4）检查安装

以root权限执行安装checkinstall，执行checkinstall命令检查安装。

sudo apt-get install checkinstall

sudo checkinstall

（5）确认安装

进入Python解释器输入“import cv2”无报错即成功安装。

5.1.3 寻找数字

（1）方案选择

要进行数字识别，首先要让程序能够寻找到画面中的数字。安装好的OpenCV中有自带的分类器，但是很不幸的是自带的分类器仅有关于人脸识别方向的，如果是做人脸识别方向的研究使用该分类器将会非常方便。至此，本文需要自己开发一个寻找数字的分类器或程序。有两种方案，一是训练一个关于数字识别的级联分类器，二是直接对画面中的元素进行寻找。方案一训练一个级联分类器并不容易，它需要准备正负两种样本。正样本中全是不同字体数字的图片集合，负样本要求是与数字可能出现的场景中非数字本身的物品图片集合。该方案适用于对识别要求严格的商业级软件开发。方案二相对于方案二适用于比较简单的测试环境，可以容忍一定识别误差。训练级联分类器对于负样本要求的范围广泛，数据量巨大。与深度学习目前所遇到的问题一样，虽然目前有大数据的支持，但是数据的标注代价是巨大的。本文以研究学习为目的且最终测试的环境简单，所以采用第二种方案。

（2）方案实施

在使用摄像头读取画面时，视频流中的每一帧可以看作为一张图片，如图5-1所示以单张写有数字的图片，代替摄像头读取到的单独一帧。

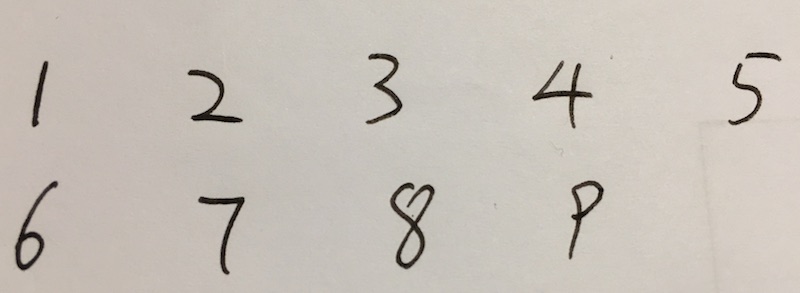


图5-1 手写数字图

先使用cvtcolor（）函数对图片进行灰度处理，然后对灰度图进行二次腐蚀处理，可以去除一些细微的无关纹理，处理后效果如图5-2所示可以看到经过二次腐蚀后的图像只能够依稀看到九个数字的痕迹。

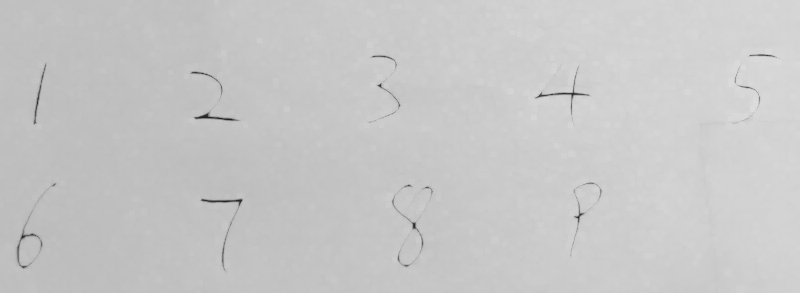


图5-2 手写数字经二次腐蚀处理

对图片进行二次膨胀处理，是数字在原图的基础上轮廓更加清晰，处理后效果如图5-3所示。

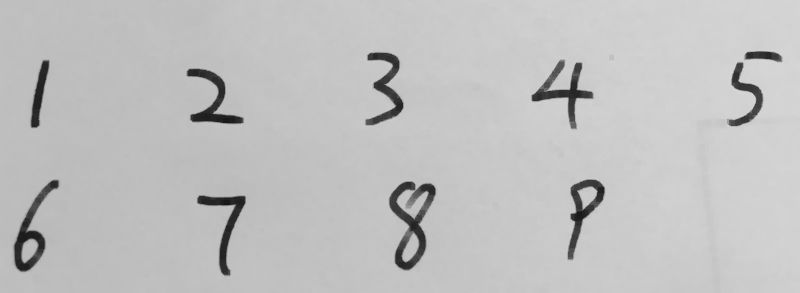


图5-3 手写数字经二次膨胀处理

用膨胀后的图和腐蚀后的图做差，可以得到清晰地数字轮廓，方便后续调用函数提取数字的轮廓信息，处理后如图5-3所示。

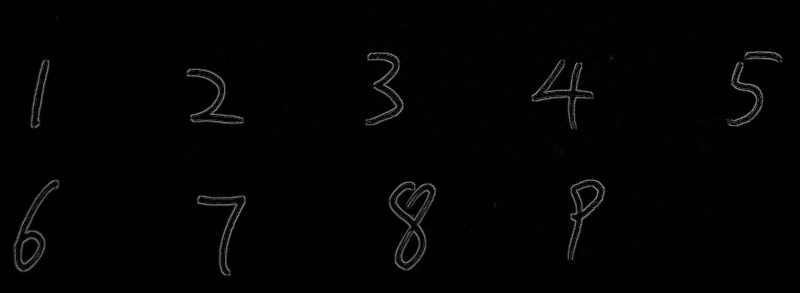


图5-3 手写数字轮廓图

对数字轮廓使用Sobel去噪处理，使轮廓更加清晰，将数字中未连接的部分连接在一起，减小轮廓寻找误差，处理后如图5-4所示。



图5-4 手写数字轮廓经算子去噪

利用findContours找到画面中所有轮廓，按设置的像素长宽寻找单个数字轮廓记录其相对坐标及长宽值，最后根据记录的信息在原图中将找到的数字标注出来如图5-5所示。

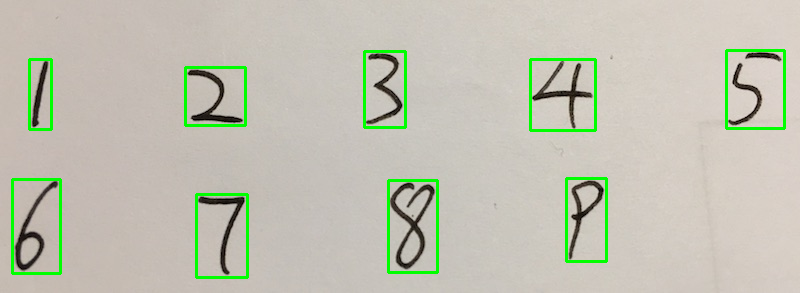


图5-5 单个数字标识

函数实现代码：

def where\_num(frame):

rois = []

# 灰度处理

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

element = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))

# 二次腐蚀处理

gray2 = cv2.dilate(gray, element)

gray3 = cv2.dilate(gray2, element)

# cv2.imshow("dilate", gray3)

# 二次膨胀处理

gray2 = cv2.erode(gray2, element)

gray2 = cv2.erode(gray2, element)

# cv2.imshow("erode", gray2)

# 膨胀腐蚀做差

edges = cv2.absdiff(gray, gray2)

# cv2.imshow("absdiff", edges)

# 使用算子进行降噪

x = cv2.Sobel(edges, cv2.CV\_16S, 1, 0)

y = cv2.Sobel(edges, cv2.CV\_16S, 0, 1)

absX = cv2.convertScaleAbs(x)

absY = cv2.convertScaleAbs(y)

dst = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

# cv2.imshow("sobel", dst)

# 选择阀值对图片进行二值化处理

ret\_1, ddst = cv2.threshold(dst, 50, 255, cv2.THRESH\_BINARY)

# 寻找图片中出现过得轮廓

im, contours, hierarchy = cv2.findContours(

ddst, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

# 在保存的轮廓里利用宽度和高度进行筛选

for c in contours:

x, y, w, h = cv2.boundingRect(c)

if w > 12 and h > 24:

rois.append((x, y, w, h))

return rois

5.1.4 数字处理

在对寻找到的数字进行识别前，还需要进一步对单个数字进行处理。要使训练好的模型能够接收待预测的图片，需要将单个数字处理成与模型训练集数据一样的格式。单个数字截取图应被处理为28\*28大小，二值化黑底白字的形式。如图5-6所示为MNIST数据集中的一个样本，图5-7为本文经处理过后的单个数字示例。

C:\Users\lee\AppData\Local\Microsoft\Windows\INetCache\Content.Word\test7.jpeg

图5-6 MNIST单个数字示例

C:\Users\lee\AppData\Local\Microsoft\Windows\INetCache\Content.Word\num1_screenshot_03.06.2017.png

图5-7 经处理后单个数字示例

函数代码:

def resize\_image(image):

//单个数字灰度化

GrayImage = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

//二值化并翻转为黑底白字

ret, thresh2 = cv2.threshold(GrayImage, 120, 255, cv2.THRESH\_BINARY\_INV)

//给数字增加一圈黑色方框

constant = cv2.copyMakeBorder(

thresh2, 20, 20, 20, 20, cv2.BORDER\_CONSTANT, value=0)

//调整数字图片尺寸

image = cv2.resize(constant, (28, 28))

return image

5.2 实现摄像头的手写数字实时识别

5.2.1 手写数字识别模型训练

结合前文的实验，选择合适的模型结构与参数训练出来的模型测试集上的识别率达到了99%以上，具体结构如下图5-8所示。

代码如下：

//创建贯序模型对象

model = Sequential()

//添加卷积层，设置30个尺寸大小为5\*5的卷积核，激活函数为relu

model.add(Convolution2D(30, 5, 5,border\_mode='valid',input\_shape=input\_shape))

model.add(Activation('relu'))

//添加下采样层

model.add(MaxPooling2D(pool\_size=pool\_size))

//添加drop层，概率设为0.4

model.add(Dropout(0.4))

//添加卷积层，设置15个尺寸大小为3\*3的卷积核，激活函数为relu

model.add(Convolution2D(15, 3, 3))

model.add(Activation('relu'))

//添加下采样层

model.add(MaxPooling2D(pool\_size=pool\_size))

//添加drop层，概率设为0.4

model.add(Dropout(0.4))

//添加Flatten使数据一维化

model.add(Flatten())

//添加全连接层，设置128个节点，激活函数为relu

model.add(Dense(128))

model.add(Activation('relu'))

//添加drop层，概率设为0.4

model.add(Dropout(0.4))

添加全连接层，设置50个节点，激活函数为relu

model.add(Dense(50))

model.add(Activation('relu'))

//添加drop层，概率设为0.4

model.add(Dropout(0.4))

//添加全连接层，设置10个节点，激活函数为softmax

model.add(Dense(10))

model.add(Activation('softmax'))

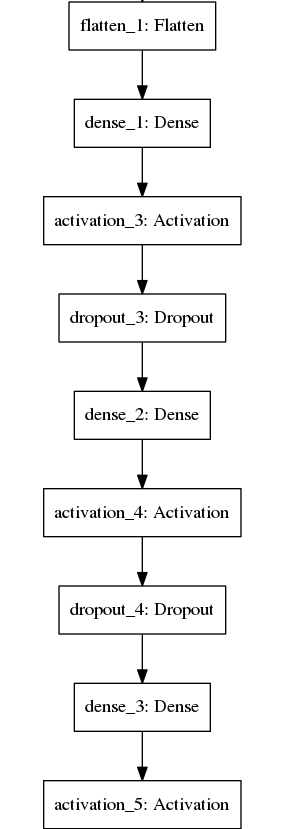
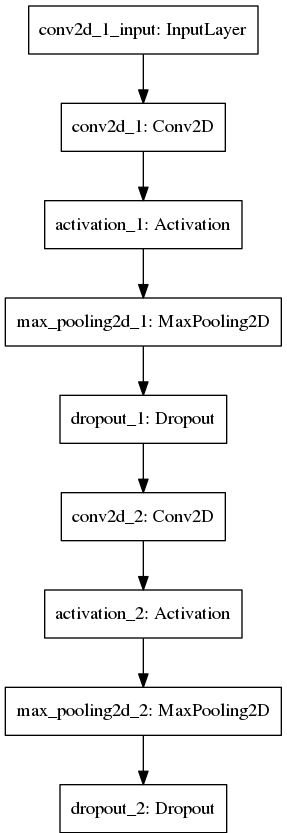


图5-8 手写数字识别模型结构

5.2.2 模型的保存与调用

（1）在训练模型的代码中使用model.save(filepath)将Keras训练好的模型和权重保存在一个HDF5文件中，该文件将包含：模型的结构，以便重构该模型；模型的权重训练配置（损失函数，优化器等）优化器的状态，以便于从上次训练中断的地方开始。

（2）使用load\_model(filepath)可以加载保存的HDF5文件，并直接对模型进行调用操作。

5.2.3 结果

调用训练好的模型，使用model.predict\_class()函数对截取出来的单个数字进行识别,将识别结果返回并在原帧画面中标注出来。如图5-9至图5-11所示，本文采取五个不同个人所书写的数字进行测试。

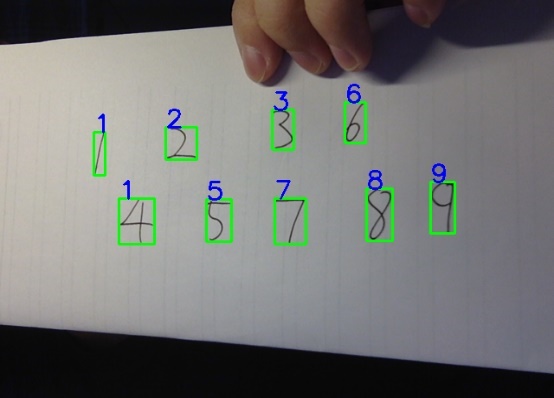
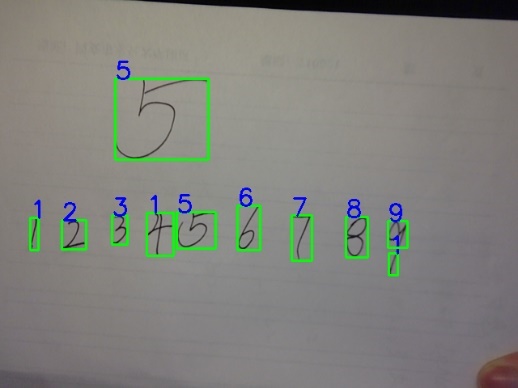
 

图5-9 手写数字识别程序运行截图1

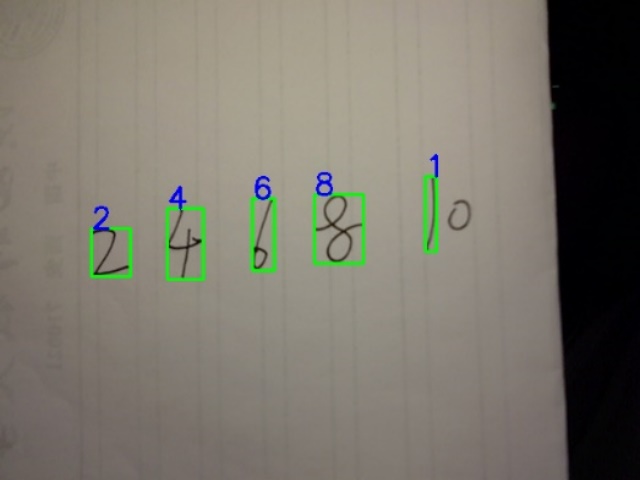
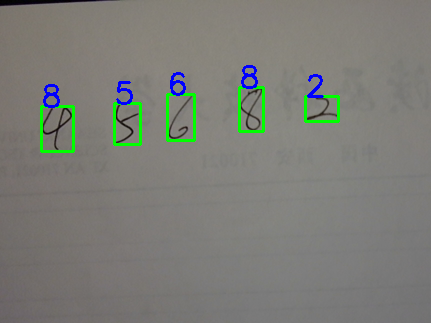
 

图5-10 手写数字识别程序运行截图2

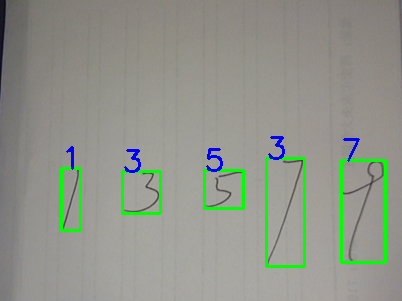


图5-11 手写数字识别程序运行截图3

经测试训练好的手写数字模型能够识别出一定程度畸变的数字，对于不常见的手写体风格不能够做出有效识别。

5.3小结

本章在前章的实验经验上设计出改进后的手写数字识别模型，在测试集的测试中准确率达到了99%以上。使用计算机视觉库OpenCV实现对摄像头的调用，利用寻找轮廓的方法在摄像头拍摄的帧画面中确定数字的位置并截取，调用训练好的手写数字识别模型对每个数字进行识别，将识别结果在原帧画面中标注返回。

6 总结

深度学习目前在大数据、大计算的支持下引领者新一波的AI热潮。使用MNIST数据集实现卷积神经网络的训练已经成为了深度学习入门的“hello word”。从经典的问题入手学习深度学习，一方面可以很好的入门，另一方面在学习的过程中所遇到的问题大部分通过浏览搜索互联网资源可以得到解决。本文以学习深度学习中重要分支卷积神经网络的过程为线索，由简入深从基本的神经元学起，学习和理解如何对卷积神经网络进行训练。在研究学习训练卷积神经网络的调参优化同时训练出高识别率的手写数字识别模型，并将该模型与OpenCV技术相结合，实现了基于电脑摄像头对画面中手写体数字进行实时识别。通过在对模型参数、结构的实验探究分析，发现并不是对训练集图像的特征提取的越多最终模型识别上得表现就越好，合理组合不同参数，选择合适的训练批次及批次数据大小才能训练出高识别率的模型。通过对模型的实际应用，反映出的问题是虽然在模型在测试集上表现达到近百分百的准确度，但是在应用过程中识别的效果并不完美，有个别数字会在两个结果中间来回跳，这说明理论到实际的应用过程中需要考虑到的实际因素还很多。在生活中我们对于数字精准要求不容闪失，任何一个数字的误差都会导致不可估量的损失。

在此次毕业设计中我学习了许多知识，从神经网络的原理推到中复习了高等数学的部分知识、从Keras、OpenCV的环境配置中扎实了Linux命令的使用、在使用Keras训练模型的过程中熟练掌握了Python面向对象编程以及使用Matlibplot库绘制实验图像。在将训练好的模型实际应用的过程中锻炼发现问题、解决问题独立思考的能力。通过这次毕业设计，让我对深度学习有了一个清晰地概念，能够将卷积神经网络应用到一些实际问题中，并取得了好的结果。

参 考 文 献

[1] 陈浩翔.手写数字深度特征学习与识别.计算机技术与发展26.7(2016):19-23.

[2] 张晓.手写数字识别的前景与难点.数码世界1(2016):69-70.

[3] Andrew Ng.Machine Learning[OL].2016.3.https://www.coursera.org/learn/machine-learning/

[4] 尹宝才,王文通,王立春.深度学习研究综述[J].北京工业大学学报, 2015(1):48-59.

[5] 郭丽丽,丁世飞.深度学习研究进展[J]. 计算机科学, 2015,42(5):28-33.

[6] 吴岸城.神经网络与深度学习[M].北京:电子工业出版社,2016.6

[7] The MNIST Database of Handwritten Digits[DB].http://yann.lecun.com/exdb/MNIST/

[8] 赵永科.深度学习21天实战Caffe[M].北京:电子工业出版社,2016.7

[9] 李丹."基于LeNet-5的卷积神经网络改进算法."计算机时代8(2016):4-6.

[10] Peter Harrington.Machine Learning in Action[M].Manning Publications,2012-4-19

[11] Andrew Ng,Jiquan Ngiam,Chuan Yu Foo,Yifan Mai,Caroline Suen.UFLDL教程[OL].2016.3.http://deeplearning.stanford.edu/wiki/index.php/UFLDL\_Tutorial

[12] Fei-Fei Li,Andrej Karpathy.CS231n Convolutional Neural Networks for Visual Recognition[OL].2016.1.http://cs231n.stanford.edu/

[13] 吴忠,朱国龙,黄葛峰,等.基于图像识别技术的手写数字识别方法[J]. 计算机技术与发展,2011,21(12):48-51.

[14] 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.

[15] 孙志军, 薛磊, 许阳明, 等. 深度学习研究综述[J]. 计算机应用研究, 2012, 29(8):2806-2810.

[16] LeCun Y, Bengio Y, Hinton G. Deep learning[J].Nature,2015,521(7553):436-444.

[17] Krizhevsky A,Sutskever I,Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems.2012:1097-1105.

致 谢

提笔至此，我突然意识到，大学四年时光转瞬即逝即将画上句号。这四年里有许多值得回忆的事。许多老师和同学不论是在生活中还是学习上都给了我无私的帮助。

在本次毕业设计完成过程中，特别要感谢老师的指导和帮助。老师治学态度严谨，有着深厚的专业功底，他创建ACM实验室，组织ACM比赛为同学们提供学习发展的平台，本人在ACM实验室期间收获颇多。同时也要感谢身边的同学们，在毕业设实现过程中出现问题时一起讨论并提出解决方案。最后，感谢父母对我的支持，希望不负他们的期望，顺利毕业。