中国矿业大学

CHINA UNIVERSITY OF MINING AND TECHNOLOGY

中国矿业大学计算机学院

2017级本科生课程大作业

基于卷积神经网络的验证码识别

课程名称	计算智能
报告时间	2020.03.31
学生姓名	
学　　号	
专　　业	计算机科学与技术
任课教师	

序号	课程教学目标	考查方式与考查点	占比	得分点	应得分	实际得分	目标汇总
1	目标1：了解计算智能在实际应用领域中的应用背景、现状以及趋势。	针对某个实际问题，能够总结适用该问题计算智能方法的背景和应用领域	20%	0	10		
				1	5		
				8	5		
2	目标2：能够将实际问题抽象成数学模型，并通过具体的计算智能方法原理分析，对此模型进行求解。	针对具体问题，抽象出数学模型，并能够求解。	40%	2	15		
				3	10		
				9	15		
3	目标3：根据所选计算智能方法，对实际问题对应的模型求解方案加以实验验证，并对方法的有效性、合理性以及优缺点进行分析，并与其他方法进行对比，得到有效结论。	对所选方法以仿真方式进行实现，并分析方法的优缺点，与其他方法对比，得出结论。	40%	4	15		
				5	10		
				6	5		
				7	10		
总分			100%	-	100		

基于卷积神经网络的验证码识别

摘要：验证码在设计之初只是一个用来区分人和计算机程序的图灵测试，由于验证码的简单、易用，现在作为维护互联网安全的主要工具。验证码一般会出现在用户注册、登录网站、信息的查询等场景，在这些场景中希望用户是一个真正的人，而不是一个计算机程序。文本型的验证码对于人来说是很容易识别的，但是对于计算机来说就很困难，因此验证码的能一定程度的维护一个良好的网络环境。为保证验证码的安全性和可靠性，验证码识别技术应运而生，作为学术研究去识别验证码，可以发现现有验证码存在的漏洞，为设计出更加有效，而不易于机器识别的验证码。

本文利用深度学习框架TensorFlow对验证码进行识别。针对易于分割的验证码，采用灰度化、二值化、去噪与分割处理验证码，然后利用卷积神经网络来对验证码进行识别。不易分割的验证码，采用端到端的识别方法。两种识别方式都取得了很好的效果，具体内容与成果如下：

(1) 介绍了验证码的用途以及验证码识别国内外现状，详细介绍了卷积神经网络中卷积、池化的原理，激活与损失函数的计算方法。

(2) 在验证码的处理过程中，使用加权平均值法对图片进行灰度化，设置阈值进行二值化，利用8邻域的方式去噪处理，最后利用垂直投影进行验证码的分割。在数据集处理过程中，针对分割的字符进行大小归一化处理，使用one-hot对样本进行归类编码，最后随机分割出训练集与测试集。

(3) 在模型训练与测试中，采用分割处理的验证码识别率达到99%以上，采用端到端的验证码识别率达到98.75%。根据不同参数的训练结果分析，最后得到了使用Adam优化器与学习率设置为0.01时候，模型的训练速度和准确率有较好的结果。

关键字：卷积神经网络；Tensorflow；验证码

CAPTCHA Recognition Based On Convolutional Neural Network

Abstract: The verification code was only a Turing test to distinguish people from computer programs at the beginning of the design. Because the verification code is simple and easy to use, it is now used as the main tool for maintaining Internet security. The verification code generally appears in scenarios such as user registration, website login, and information query. In these scenarios, the user is expected to be a real person, not a computer program. Text-based verification codes are easy for humans to recognize, but difficult for computers, so verification codes can maintain a good network environment to a certain extent. In order to ensure the security and reliability of verification codes, verification code identification technology came into being. As an academic research to identify verification codes, you can find the vulnerabilities of existing verification codes, and to design more effective verification that is not easy for machine identification code.

This paper uses the deep learning framework TensorFlow to identify the verification code. For verification codes that are easy to segment, gray-scale, binarization, denoising and segmentation verification codes are used, and then convolutional neural networks are used to identify the verification codes. The verification code, which is not easy to divide, adopts the end-to-end identification method. Both recognition methods have achieved good results. The specific contents and results are as follows:

(1) Introduced the use of verification codes and the current status of verification code recognition at home and abroad, introduced in detail the principles of convolution and pooling in convolutional neural networks, and the calculation methods of activation and loss functions.

(2) In the process of verification code, the weighted average method is used to grayscale the image, the threshold is set to binarize, and the denoising process is performed using the 8-neighbourhood method. Finally, the vertical projection is used to divide the verification code. In the process of data set processing, the size of the segmented characters is normalized, one-hot is used to classify the samples, and finally the training set and the test set are randomly split.

(3) In model training and testing, the recognition rate of verification codes using segmentation processing reached more than 99%, and the recognition rate of end-to-end verification codes reached 98.75%. According to the analysis of the training results of different parameters, finally, when the Adam optimizer is used and the learning rate is set to 0.01, the training speed and accuracy of the model have better results.

Keywords: Convolutional neural network; Tensorflow; CAPTCHA

1 验证码识别问题描述

1.1 验证码简介

全自动区分计算机和人类的公开图灵测试（Completely Automated Public Turing test to tell Computers and Humans Apart，简称CAPTCHA），俗称验证码，是一种区分用户是计算机或人的公共全自动程序[1]。验证码由于简单高效的特征，被广泛应用于网站注册、登陆中，用来区分是人类还是机器进行访问，抵挡恶意软件的自动化攻击，但是这个验证码可以让人类很轻松的通过，但是可以有效的阻止机器进行访问。比如，12306购票中，防止黄牛刷票、抢票，这样就能较好的保证用户能够购买到所需的票；网络平台加入验证码，防止“撞库攻击”，有效的减少服务资源的乱用，造成用户信息的泄露等。

1.2 研究意义

当互联网应用越来越多时，如何保证互联网的安全成为一个很重要的问题，若一类验证码被机器识别的概率较高，则说明使用这类验证码的网站或者服务存在风险。验证码作为基础的保护措施，以学术目的研究去识别验证码，可以发现验证码中的漏洞，并提出一些改进意见，有利于开发人员设计出更加安全、有效的验证码。

自从2000年9月，卡内基梅隆大学研究团队设计的，第一个用来抵御雅虎聊天室被非法程序恶意攻击的Gimpy系列商业验证码以来，研究验证码的设计和识别也开始了[2]。验证码识别涉及图像处理、机器学习、深度学习等各种技术，随着验证码与识别技术的发展，越来越多的识别技术也推广到了其他方面，比如，车牌识别，证件识别等，掌握验证码识别的技术也有利于运用在其他方向。

1.3 研究现状

由于验证码应用的广泛，国内外学者对于验证码的识别也有很多研究。早期的验证码识别主要以分割技术为主，因为所识别的验证码几乎没有背景干扰或者噪点，字符也大都清晰，没有扭转或者重叠，所以识别率很高。经过不断的发展，验证码越来越复杂，分割效果不好，也无法很好的识别复杂的验证码，所以出现了一些其他的分割识别方法。王璐提出极小值分割法、K均值聚类算法对粘连字符验证码进行分割[3]，Lu P等人提出了不同的字符分割算法并结合SVM分类算法及BP神经网络进行字符识别[4]，对于粘连较少的验证码识别的效果都比较好。

常见的验证码识别方法有模版匹配法、支持向量机（SVM）以及神经网络法[5]。现如今，作为图像识别的热门技术，又因为学习计算显卡的发展，深度学习网络方法在研究中被广泛应用。卷积神经网络（Convolution NeuralNetwork）采用随机梯度下降[6]（Stochastic Gradient Descent）和GPU（Graphics Processing Unit）加快训练过程从而使得训练大量的图像数据更为方便，在图像分类、目标检测、文本识别[7]等方面都取得了巨大的成功。

现阶段验证码的安全性不断的提高，字符的复杂变换使得整体识别准确率也难以保证，因此本文使用卷积神经网络对两类验证码进行分割识别和端到端识别。

2 卷积网络算法理论介绍

卷积神经网络(Convolutional Neural Networks，CNN)本质上是将卷积运算作为网络层之一的神经网络[8]，具有通过卷积操作进行分层抽象表示的能力。因此，CNN在计算机视觉应用领域效果显著。两个关键设计推动着CNN发展，首先，CNN利用图像二维结构以及像素在一个邻域内高度相关性，使用分组连接，避免所有像素单元之间一对一连接，实现网络稀疏性。其次，CNN权值共享，每个通道都是根据相同的滤波器进行卷积生成的。CNN的这些特征使得其与普通神经网络相比，参数更少[9]。

2.1 与传统多层神经网络对比

1. 传统意义上的多层神经网络是只有输入层、隐藏层与输出层。其中隐藏层的层数根据需要而定，没有明确的理论推导来说明到底多少层合适。
2. 卷积神经网络CNN，在原来多层神经网络的基础上，加入了更加有效的特征学习部分，具体操作就是在原来的全连接层前面加入了卷积层与池化层。卷积神经网络出现，使得神经网络层数得以加深，“深度”学习由此而来。

2.2 CNN结构

- 卷积层——通过在原始图像上平移来提取特征。
- 激活层——增加非线性分割能力。
- 池化层——减少学习的参数，降低网络的复杂度（最大池化和平均池化）。
- 全连接层——为了能够达到分类效果。

卷积神经网络的输入数据为没有经过预处理的原始数据，比如验证码原图、原始动画以及音频等数据。输入数据通过卷积、池化、和非线性激活函数映射等一系列操作，将数据特征从输入层中抽象提取出来，这个过程称为前馈运算。CNN的最后一层将图像处理任务转化为计算预测值与真实值之间误差的目标函数，反向传播算法将前馈运算预测出的结果与真实结果之间的误差再由后一层向前传递，通过误差传递调整每层参数，并在更新参数后再次前反馈，前向传播与反向传播往复进行，当预测结果收敛时，完成模型训练[10]，图1展示了简单的CNN结构图。

在图像处理应用领域中，一般将RGB彩色图像作为卷积网络的输入，有H行、W列、RGB通道，将输入x^1与参数w^1做一次卷积运算输出x^2，然后再将x^2与w^2做卷积运算，输出x^3，…，一直进行$L-1$次，得到数据x^L，通过公式1计算输入数据y与真实标记x^1之间的误差，结束整个网络。损失函数表示为[11]：

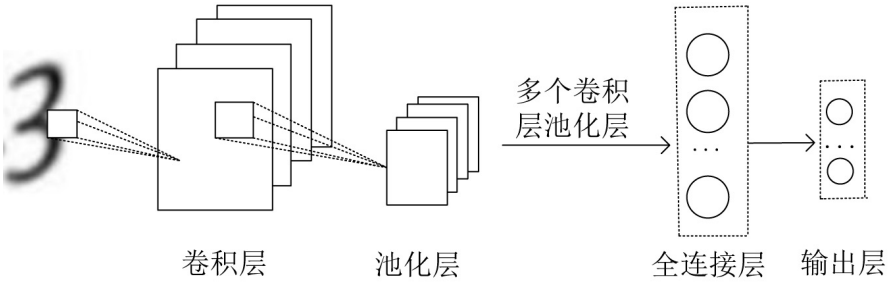

图 1: 简单的CNN结构图

$$z = (x^L, y) \tag{1}$$

公式中，w^L是$L(x^L, y)$的参数。在实际应用中，对于不同的任务，损失函数的形式不同。交叉熵损失函数被用作分类任务中CNN目标函数$z = L_{classification}(x^L, y) = -\sum_i y_i log(p_i)$，其中$p_i = |\frac{exp(x_i^L)}{\sum_{j=1}^{C}(i=1,2,...,C)}|$，$c$为分类任务的类别[9]。$L_2$损失函数通常作为回归问题中CNN的目标函数，则有$z = L_{regression}(x^L, y) = \frac{1}{2}||x^L - y||^2$。无论回归任务还是分类任务，计算$z$之前，都需要一系列运算得到与$y$同维度的$x^L$。

2.3 卷积层

卷积层是CNN的核心构件，它承担了大部分计算量。卷积具有平移不变性，主要负责特征提取。卷积核的参数可以通过学习得到，每个卷积核在空间上较小，对输入数据执行局部加权组合，根据所选择的权重集合，揭示了输入数据的不同特性。网络每一层的输出特征与当前层卷积核相乘后累加，加上偏置项，并通过激活函数非线性映射得到当前层输出。输入数据中包含很多重要信息，根据这些信息可以对输入内容做出强有力的推断，因此选择正确的卷积核来获取最显著特征非常重要。如公式2所示[23]：

$$x_j^l = f(\sum_{i \in M_j} x_i^{l-1} \cdot k_{ij}^l + b_j^l) \tag{2}$$

x为该层输入，k为卷积核，b表示偏置项。

讨论论单通道输入$c_{in} = 1$，如灰度图片只有灰度值一个通道，单个卷积核$c_{out} = 1$情况。输入X为5×5，卷积核为3×3，如图2。与卷积核同大小的感受野(输入X上方的方框)首先移动至输入X最左上方，选中输入X上3×3的感受野元素，与卷积核(图片中间3×3方框)对应元素相乘：

$$\begin{bmatrix} 1 & -1 & 0 \\ -1 & -2 & 2 \\ 1 & 2 & -2 \end{bmatrix} \begin{bmatrix} -1 & 1 & 2 \\ 1 & -1 & 3 \\ 0 & -1 & -2 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 2 & 6 \\ 0 & -2 & 4 \end{bmatrix} \tag{3}$$

运算后得到3×3的矩阵，这9个数值全部相加：$-1-1+0-1+2+6+0-2+4 = 7$，得到标量7，写入输出矩阵第一行、第一列的位置如图2所示。

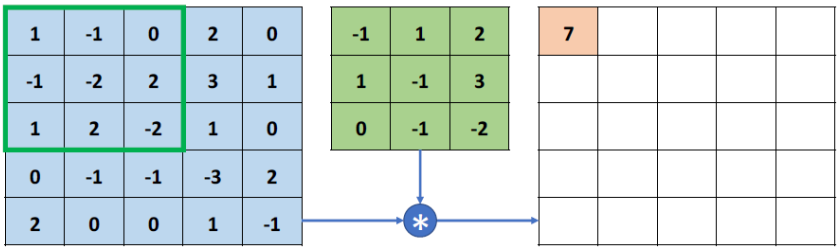

图 2: 3×3卷积运算-1

完成第一个感受野区域的特征提取后，感受野窗口向右移动一个步长单位(Strides，记为s，默认为1)，选中图3中方框中的9个感受野元素，按照同样的计算方法，与卷积核对应元素相乘累加，得到输出10，写入第一行、第二列位置。

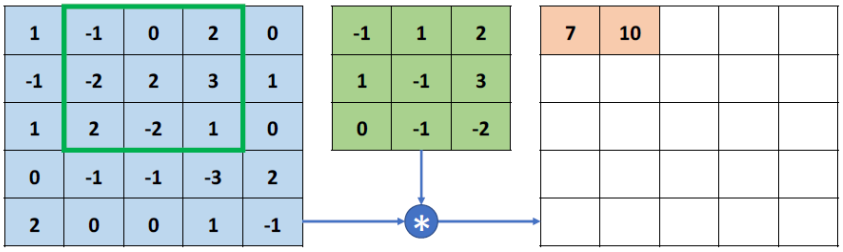

图 3: 3×3卷积运算-2

按照上述方法，每次感受野向右移动$s = 1$个步长单位，若超出输入边界，则向下移动$s = 1$个步长单位，并回到行首，直到感受野移动至最右边、最下方位置，如下图4所示。

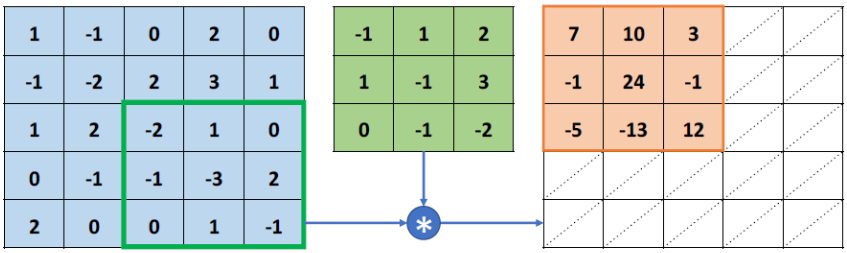

图 4: 3×3卷积运算-3

2.4 池化层

任何CNN模型，无论是生物学启发或者纯粹基于学习，都包含一个合并步骤称为池化操作。池化的作用是保持位置和规模在一定程度上不变的情况下对卷积层提取出的特征进行二次提取，从而减少网络原始输入数据抽取出的特征，

降低网络后续的计算量，有效避免了过拟合情况的发生。池化操作是对某一位置的邻域进行特征统计，然后用统计值取代原来的特征值[12]。

一些早期生物学启发的CNN依赖于均值池化(ave- pooling)[13]。均值池化将相邻区域所有值的均值作为池化结果，用均值池化进行二次特征抽取主要是为了降低网络对位置变化的敏感性。另一方面，HMAX[12]类网络依赖于最大值池化(max-pooling)。最大池化将相邻区域所有值的最大值作为池化结果，当池化特征非常稀疏时，选择最大池化更合适。均值池化与最大值池化表示如下[13]：

$$y_{i^{l+1},j^{l+1},d} = \frac{1}{HW} \sum_{0 \le i < H, 0 \le j < W} x_{i^{l+1} \times H + i, j^{l+1} \times W + j, d^l} \tag{4}$$

其中$0 \le i^{l+1} < H^{l+1}, 0 \le j^{l+1} < W^{l+1}, 0 \le d < D^{l+1} = D^l$。区别于卷积操作，池化层只需要指出池化函数类型、池化操作的窗口大小与步长等超参数。最常见的池化形式是池化层带有大小为2×2的过滤器，沿宽度和高度对输入进行下采样，丢弃36%的激活。图5用最大值池化以2×2的过滤器对5×5的特征数据，进行二次特征抽取。

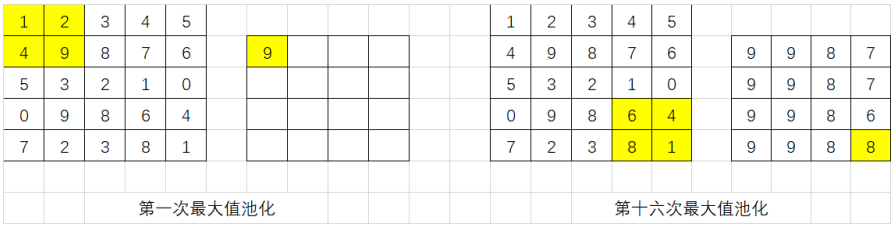

图 5: 最大值池化过程

池化步骤的本质是模仿生物观察物体时对观测到的物体抽象表示的过程。从图5可以发现，池化后的结果相比输入特征减少，起到降维作用。

2.5 激活函数

神经网络中的常见激活函数，与阶跃函数和符号函数不同，这些函数都是平滑可导的，适合于梯度下降算法。

2.5.1 Sigmoid

Sigmoid 函数也叫 Logistic 函数，定义为：

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

它的一个优良特性就是能够把$x \in R$的输入“压缩”到$x \in (0, 1)$区间，这个区间的数值在机器学习常用来表示以下意义：

- 概率分布——(0,1)区间的输出和概率的分布范围[0,1]契合，可以通过Sigmoid函数将输出转译为概率输出。

- 信号强度——一般可以将0、1理解为某种信号的强度，如像素的颜色强度，1代表当前通道颜色最强，0代表当前通道无颜色；抑或代表门控值（Gate）的强度，1代表当前门控全部开放，0代表门控关闭。

Sigmoid函数连续可导，如图6所示，可以直接利用梯度下降算法优化网络参数，应用的非常广泛。

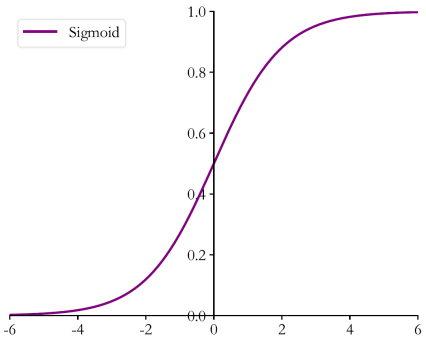

图 6: Sigmoid 函数曲线

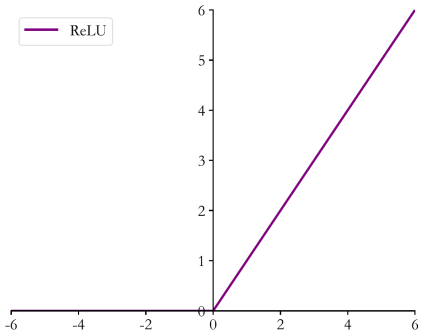

图 7: ReLU 激活函数

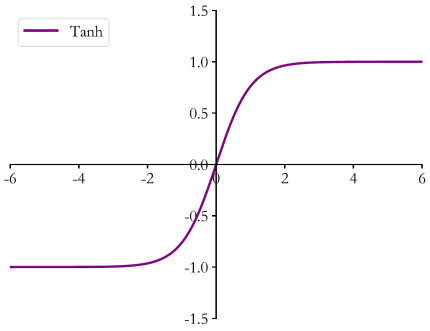

图 8: tanh 函数曲线

2.5.2 ReLU

在 ReLU(REctified Linear Unit，修正线性单元)激活函数提出之前，Sigmoid函数通常是神经网络的激活函数首选。但是Sigmoid函数在输入值较大或较小时容易出现梯度值接近于0的现象，称为梯度弥散现象。出现梯度弥散现象时，网络参数长时间得不到更新，导致训练不收敛或停滞不动的现象发生，较深层次的网络模型中更容易出现梯度弥散现象。2012年提出的8层AlexNet模型采用了一种名叫ReLU的激活函数，使得网络层数达到了8层，自此ReLU函数应用的越来越广泛。ReLU函数定义为：

$$ReLU(x) = max(0, x) \tag{6}$$

函数曲线如图7所示。可以看到，ReLU对小于0的值全部抑制为0；对于正数则直接输出，这种单边抑制特性来源于生物学。

2.5.3 Tanh

Tanh函数能够将$x \in R$的输入“压缩”到$(-1,1)$区间，定义为：

$$tanh(x) = \frac{(e^x - e^{-x})}{e^x + e^{-x})} \tag{7}$$

可以看到tanh激活函数可通过Sigmoid函数缩放平移后实现，函数曲线如图8所示。

2.6 损失函数

2.6.1 均方差函数

均方差（Mean Squared Error，简称MSE）函数把输出向量和真实向量映射到笛卡尔坐标系的两个点上，通过计算这两个点之间的欧式距离（欧式距离的平方）来衡量两个向量之间的差距：

$$MSE(y,o) = \frac{1}{d_{out}} \sum_{i=1}^{d_{out}} (y_i - o_i)^2 \tag{8}$$

其中，y_i为第i个样本的真实值，o_i为第i个样本的输出值，即预测值，d_{out}为样本数量。MSE误差函数的值总是大于等于0，当MSE函数达到最小值0时，输出等于真实标签，此时神经网络的参数达到最优状态。

2.6.2 交叉熵函数

交叉熵（cross entropy）描述的是两个概率分布之间的距离，距离越小表示这两个概率越相近，越大表示两个概率差异越大。

$$H(p,q) = -\sum_i p(i) \log_2 q(i) \tag{9}$$

其中$p(i)$是真实值，$q(i)$为预测值。交叉熵可以很好地衡量2个分布之间的“距离”，当使用sigmoid作为激活函数的时候，常用交叉熵损失函数而不用均方误差损失函数，因为它可以完美解决平方损失函数权重更新过慢的问题，具有“误差大的时候，权重更新快；误差小的时候，权重更新慢”的良好性质。

3 卷积网络算法解决验证码识别的步骤

验证码识别过程大致分为：验证码获取、图片处理、整合数据集、建立卷积网络与识别。

3.1 验证码获取

在深度神经网络的训练中，需要大量的样本作为保障，这样才能使训练后的模型更加精确。对于大量的图片验证码，可以利用程序自动生成，对于简单的数字字母验证码，其生成原理是将一串随机产生的数字或字母展示在图片，然后在图片里加上一些干扰像素。如python中的captcha库，这样就可以短时间获取大量的样本，并且样本标签也进行了记录。本文使用学习过程中收集的两类验证码进行研究，如图9-10所示，两类验证码均为三通道。

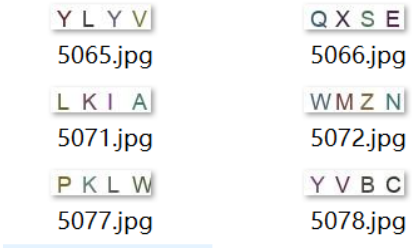

图 9: 验证码类型1

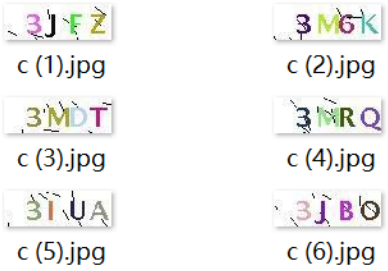

图 10: 验证码类型2

验证码1字符集为大写字母，并且没有干扰，字母间距也很平均；验证码2字符集为数字和大写字母，但是存在干扰，并且存在一些验证码存在粘连。本文使用分割预处理识别验证码1，使用端到端识别验证码2。

3.2 验证码预处理

3.2.1 灰度化

彩色图片中每个像素值是由R、G、B三个通道的像素值组合而来。每个通道像素值的取值范围为0-255，灰度图像中每个通道像素值都是相同的灰度图像可以完整的反映原图的色度和亮度等级的分布和特征，而彩色验证码图像在设计时会增加各种颜色干扰全自动化机器对字符的识别，因此把待识别验证码图像转化为灰度图，既可以保证充足的图像细节同时还能减少背景噪声过多的干扰，一定程度上减少了图像处理计算量，降低验证码识别的复杂度，程序运行时间缩短，因此将验证码灰度图作为验证码后续处理的基础。

设验证码原图为$f(x,y)$，灰度化处理以后图像为$g(x,y)$，则灰度化过程表示为$g(x,y) = T(x,y)$。高效的灰度化处理算法，对图像处理的发展非常重要。以下是三种常用的灰度化处理方法[14]：

(1) 最大值法：取RGB三者中最大值作为该点的灰度值，即：

$$R = G = B = \max(R, G, B) \tag{10}$$

最大值法进行灰度化处理会形成亮度很高的图像。

(2) 平均值法：取RGB三者的平均值作为该点的灰度值，即：

$$R = G = B = \frac{R + G + B}{3} \tag{11}$$

平均值法进行灰度化处理会形成较为柔和的灰度图像。

(3) 加权平均值法：将RGB的加权平均值作为该点的灰度值，即

$$R = G = B = W_R \times R + W_G \times G + W_B \times B \tag{12}$$

其中的W_R，W_G和W_B分别指RGB的权值。W_R，W_G和W_B取不同的值，加权平均值法进行灰度化处理会形成不同的灰度图像。

本文采用$W_R = 0.587$，$W_G = 0.299$和$W_B = 0.114$的权重对图片进行灰度化，灰度化结果为图11-12所示：

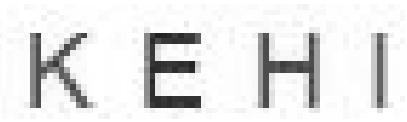

图 11: 验证码类型1灰度化

图 12: 验证码类型2灰度化

3.2.2 二值化

任意图像的像素一般在二维空间中都会对应一个特定存在的位置，并且包含一个或者多个与该像素相关的采样值组成数值。验证码识别研究中，在分割与识别之前需要对验证码字符进行二值化处理，将输入的彩色验证码图像$g(x, y)$转换为灰度图像$b(x, y)$，然后通过专用的阈值处理进行二值化，验证码图像上像素点的灰度值全部设置为255或0。把验证码图像分成目标图像和背景两个区域，将验证码中字符部分从复杂的背景图像中凸显出来，尽可能的去除背景噪声。二值化的关键是选择合适的阈值，设定某一阈值T，处理方法如下公式：

$$b(x, y) = \begin{cases} 0 & g(x, y) \leq T \\ 255 & g(x, y) > T \end{cases} \tag{13}$$

通过验证码的直方图选取合适的阈值，验证码1使用的阈值T为210，验证码2使用的阈值T为235，经过二值化处理后的验证码如图15-16所示。

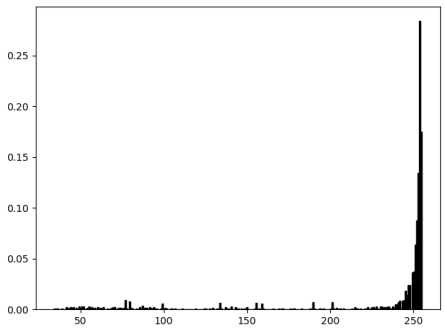

图 13: 验证码类型1直方图

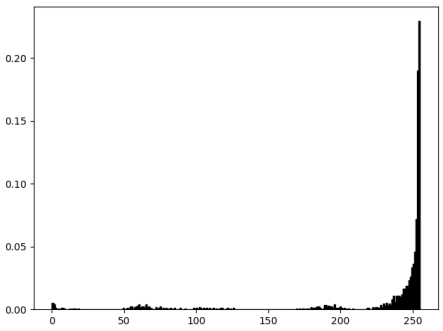

图 14: 验证码类型2直方图

图 15: 验证码类型2二值化

图 16: 验证码类型2二值化

3.2.3 去噪处理

二值化后的图像不仅有目标，有时还混有噪声。噪声一般有杂点噪声、线噪声和块噪声等。这些噪声会影响分割效果和特征提取，所以在预处理阶段要对这些噪声一一进行处理。

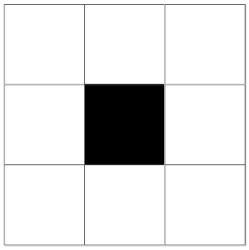

图 17: 孤立点噪声

如图17所示，对于这种孤立点噪声，通过对它的八邻域进行统计，如果一个黑色像素点的周围存在的白色像素点超过一定阈值p，就认为该像素点为噪声，然后可以把它的灰度值设为255。计算公式14如下：

$$d(x,y)=\begin{cases}0 & K\le p\\ 255 & K>p\end{cases} \tag{14}$$

$$\begin{aligned}K=&b(x-1,y+1)+b(x,y+1)+b(x+1,y+1)\\&+b(x-1,y)+b(x+1,y)\\&+b(x+1,y-1)+b(x,y-1)+b(x+1,y-1)\end{aligned}$$

图 18: 处理前

图 19: 阈值为4处理一次结果

由于验证码类型1不存在噪点，并且在二值化过程中通过合适的阈值达到了很好的背景与字符的区分，所以这里只展示验证码类型2去噪效果，如图19所示，可以看出，在阈值p为4的情况下，大部分的噪点都被除去了，但是字符存在部分的缺失，为减少此情况，选择阈值p为3处理原图片，结果为图20所示，可以看出字符特征被完整的保留下来，但是噪点为处理完全，再次去噪此验证码，

结果为图21所示，这里噪点大部分被去除了，并且字符的特征也很好的保存了下来，达到了预计的效果。

图 20: 阈值为3处理一次结果

图 21: 阈值为3处理二次结果

3.2.4 图像分割

图像分割技术就是根据图像表现出的一些特征，比如灰度、纹理、颜色、型状等特征对图片的内容进行分割，使得图像各部分之间互不干涉，比如前景和背景之间，前景中各个元素之间。常用的方法有基于阈值的分割、基于区域的分割、基于边缘检测的分割。

阈值分割方法和前面提到的二值化操作类似，都是要寻找一个比较合适的阈值，这个阈值可以是基于全局灰度统计下的全局阈值，也可以是基于局部的动态阈值，两种阈值都有它们各自的特性，所以要根据实际情况进行选择。

验证码垂直投影分割法将图像垂直分割成几块，每块可能包含一个或多个字符。通过投影直方图统计验证码图像前景像素的数量，切割完全没有前景像素的列将图像分割成块[15]。对于一个$M \times N$的二值化字符图像I，用0表示背景像素1表示字符像素，首先对二值化验证码图像进行垂直方向投影，统计图像垂直方向上的黑色像素数量，将统计值作为该位置的值，待分割图像映射成像素统计特征，基于统计特征判定图像的分割坐标，以此来分割验证码图像，得到单个字符，这里使用验证码类型1作为研究对象，投影直方图如下所示。

图 22: 验证码二值化结果

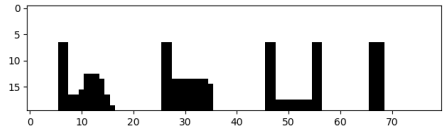

图 23: 投影直方图

图 24: 验证码1分割结果

由图23所得，可以根据阈值0将此验证码分割为部分，正好对应验证码的四个字符，对此类验证码分割的结果如图24所示，每个文件夹下面保存被分割出来的字符（为加快处理速度，在分割图片的完成的时候提前做了大小归一化处理）。

3.3 数据集处理

3.3.1 归一化处理

归一化是图像预处理中必不可少的一步，其原理就是通过一系列的变化使图像转化为某一种标准的形式给。图像归一化使得图像可以抵抗几何变换的攻击，它能够找出图像中的那些不变量，好的归一化算法可以提高后续特征提取在同一类内的一致性。在神经网络模型中归一化可以加快训练网络的收敛性。这里只是对图片的大小进行归一化处理，可以使用python中的resize函数，因为验证码类型1被分割后宽度不一致，所以这里将它们统一大小为20×20。

3.3.2 归类编码

在机器学习中，常常使用one-hot作为分类问题的编码。在手写数字识别问题中，使用一个长度为10的向量来表示标签值，如：$[1,0,0,0,0,0,0,0,0,0]$来表示数字1，$[0,1,0,0,0,0,0,0,0,0]$来表示数字2，等等，也就是说这个长度为10的向量不同位置为1。

对于验证码类型1，只存在大写字母，所以one-hot编码长度为26，向量位置与字母排序顺序一致，如，向量第一个位置为1，则代表字母A。

对于验证码类型2，存在大写字母与数字，所以one-hot编码长度为36，前十位为1分别代表数字0-9，后面为1依次代表A-Z。

3.4 建模及分析

3.4.1 CNN模型

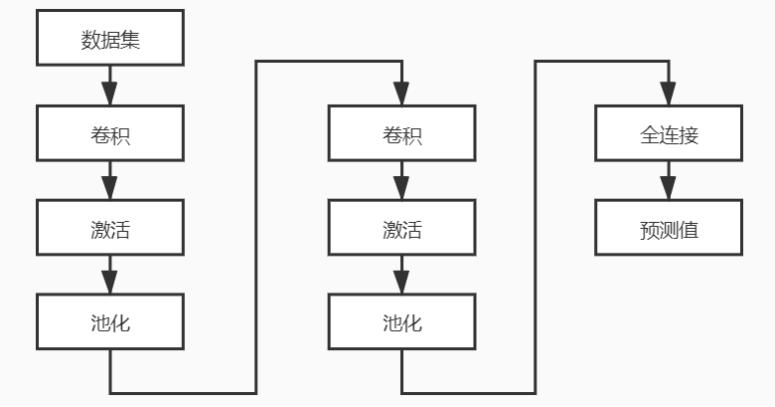

图 25: CNN模型结构

CNN 网络模型结构如图25所示，模型包括两个卷积层、两个池化层、两个激活函数和一个全连接层，其中全连接层包括softmax交叉熵函数。池化操作使得特征对噪声和变形有更强的抗干扰能力，各层所提取的特征以增强的方式从不同角度表现原始图像，并且随着层数的增加，其表现形式越来越抽象。全连接层相连可以把前期所提取的各种局部特征综合起来，这样就提高了对事物的表达能力。

优化函数：由于Adam优化器具有很多显著的优点，比如：超参数具有很好的解释性，且通常无需调整或仅需很少的微调、更新的步长能够被限制在大致的范围内和适用于梯度稀疏或梯度存在很大噪声的问题，等等，所以这里使用Adam优化器，设置学习率learning_rate为0.001。

损失函数：先sigmoid再求交叉熵，最后再求平均，这里使用Tensorflow中的sigmoid_cross_entropy_with_logits方法和reduce_mean求平均损失。

计算准确率：这里自定义评估函数，由于真实标签与预测标签都为one-hot编码，所以判断预测与真实one-hot向量的最大值的位置是否一样，一样返回True，否则返回False。由于验证码类型1被分割为单个字符，所以准确率直接将前面返回的bool类型转换为float32，再求批量数的平均值，就是准确率。对于验证码类型2，需要在上面的基础上比较四个字符是否对应准确，然后与类型1的方法继续计算就可以得到准确率。

3.4.2 模型分析

以验证码类型2为例分析网络参数：

第一卷积大层：输入shape为[None, 30, 90, 1]，含义依次是批量数（可调节，这里设置为None，具体值由封装的数据集给出）、图片的高度、图片的宽度和图片通道数（因为已经将图片二值化了，所以这里为1）。卷积核大小为3×3，32个，步长为1，输出$H_2 = (30-3+0)/1+1 = 28, W_2 = (90-3+0)/1+1 = 88, D_2 = 32$，激活函数使用 ReLU；池化窗口大小$2 \times 2$，步长为2，输出$H_3 = (28-2+0)/2+1 = 14, W_3 = (88-2+0)/2+1 = 44, D_3 = 32$。

在卷积层中含有参数，这里输入参数的个数为$3 \times 3 \times 1 = 9$，即卷积核大小的平方与输入通道数的乘积，然后根据公式$weight*x+bias$（权重的参数个数与输入的参数个数乘积，加上偏置的个数），来计算总的参数个数，这里权重与偏置的参数个数均为卷积核个数32，所以这个卷积层参数为：$32 \times 9 + 32 = 320$。

第二卷积大层：输入shape为[None, 14, 44, 32]，卷积核大小为3×3，64个，步长为1，输出$H_4 = (14-3+1)/1+1 = 13, W_4 = (44-3+1)/1+1 = 43, D_4 = 64$，激活函数使用 ReLU；池化窗口大小$2 \times 2$，步长为2，输出：$H_5 = (13-2+3)/2+1 = 8, W_5 = (43-2+1)/2+1 = 22, D_5 = 64$。

这里也有一个卷积层，参数计算方法同上一层，即$3 \times 3 \times 32 = 288$，与$64 \times 288 + 64 = 18496$。

第三全连接层：输入shape为[None, 8, 22, 64]，首先打平操作，转为[None, 8 * 22 * 64]，然后再根据sigmoid和交叉熵来计算损失。

全连接层中的参数可以直接计算，即：$4 \times 36 \times (8 \times 22 \times 64) + 4 \times 36 =$

1695888，这里4×36是标签值的one-hot编码长度，因为是端到端4个字符，每个字符的编码长度为36。

整个模型中的参数个数为：$320 + 18496 + 1695888 = 1714704$。

4 仿真运行结果

4.1 运行环境

- 操作系统——Windows 10 家庭中文版、AMD R-5 3550H 2.10 GHz；
- 编程环境——PyCharm、Python 3.7.6、TensorFlow 2.1。

4.2 分割识别结果

对于易分割的验证码识别（验证码类型1，由26个大写英文字母组成），首先需要分割，经过处理后成为单个字符组成的图片。由于分割后的图片数量较多，这里选择每个字母150张图片构造数据集，共$150 \times 26 = 3900$张图片。以$8:2$分割数据集，得到3120张训练图片，780张测试图片。学习率设置为0.001，batch_size设置为32，epochs设置为4，一次迭代训练50次batch，经过训练，损失曲线如图26所示，准确率曲线如图27所示。

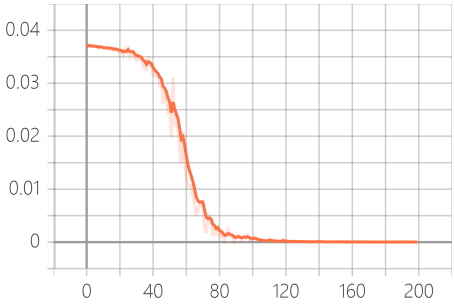

图 26: 分割识别损失曲线

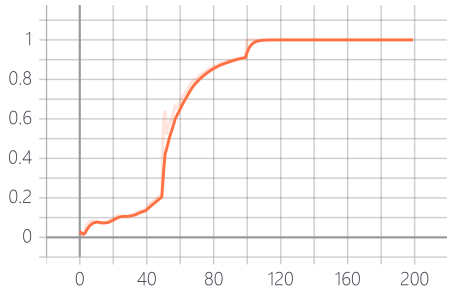

图 27: 分割识别准确率曲线

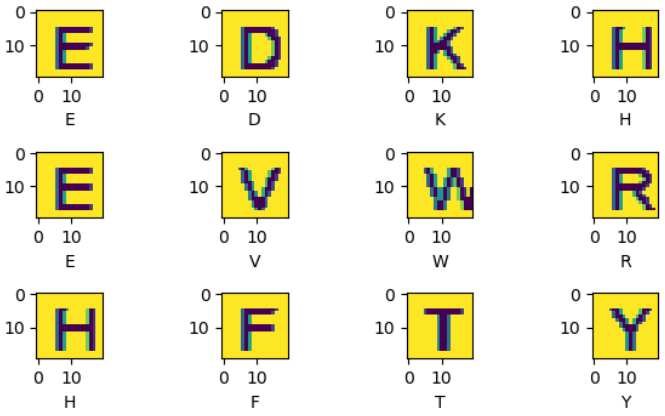

图 28: 分割识别随机预测结果

同过曲线可以发现在训练90次以后，损失值接近于0.002，准确率接近于90%，再次迭代训练，由于字符的标准、简单，准确率达到了100%，使用此模型对测试集进行评估，损失为0.00054，准确率为100%。随机选取部分测试集中的字符进行测试，结果如图28所示。

如果识别整体验证码，首先对验证码进行分割，然后依次识别分割出来的单个字符，输出结果即可，经过测试，识别率能达到99%以上。

4.3 端到端识别结果

对于不易分割的验证码使用端到端识别（验证码类型2，由26个大写英文字母与0-9数字组成），学习率设置为0.001，batch_size 设置为32，epochs 设置为6，一次迭代训练50次batch，训练样本1600个，测试样本150个，经过训练，损失曲线如图29所示，准确率曲线如图30所示。

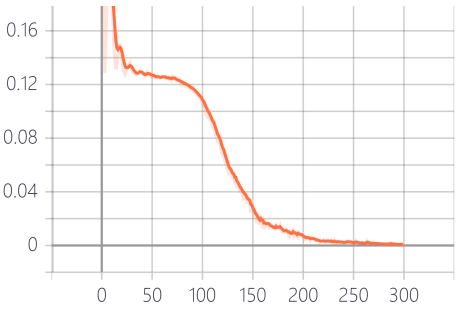

图 29: 端到端识别损失曲线

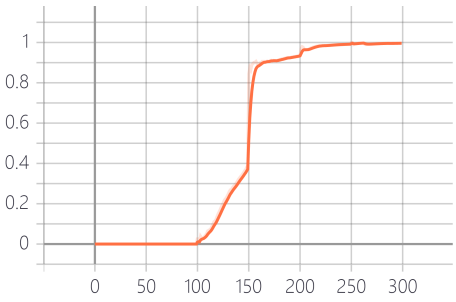

图 30: 端到端识别准确率曲线

通过曲线可以看出在训练200次左右，损失值已经降低到0.01左右，准确率已提高至90%以上，使用此模型对测试集进行评估，损失为0.0043，准确率为98.75%。随机选择部分验证码进行预测，结果如图28所示（标题左侧为真实值，右侧为预测值）。

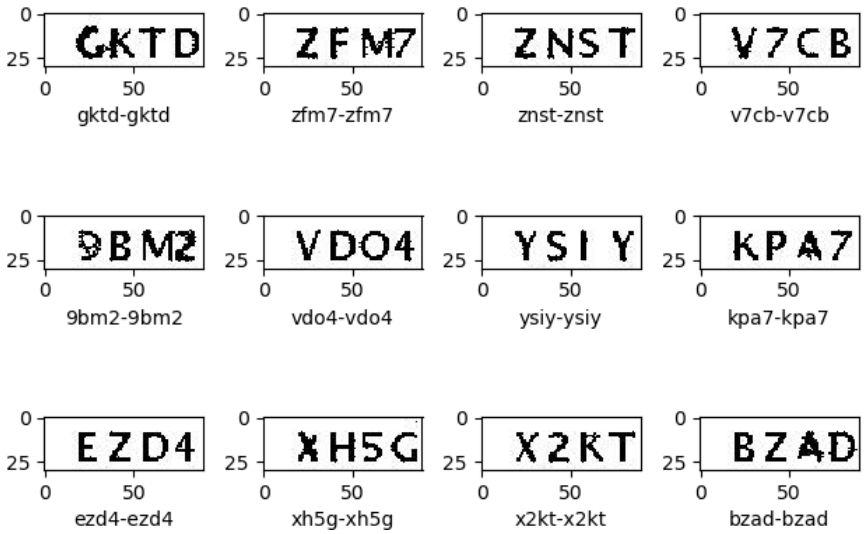

图 31: 端到端随机预测结果

4.4 改变参数训练结果

为便于叙述，以识别验证码类型2为例，将学习率由0.001改为0.01，在相同环境下，重新训练，损失曲线如32所示，准确率曲线如图33所示，可以看出学习率改为0.01，损失值的下降速率比0.001的快，准确率的上升速度也比0.001的时候快，所以此模型学习率为0.01的时候更好。

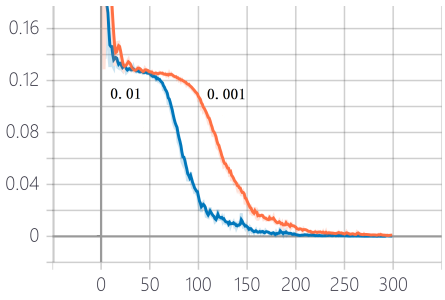

图 32: 不同学习率损失曲线

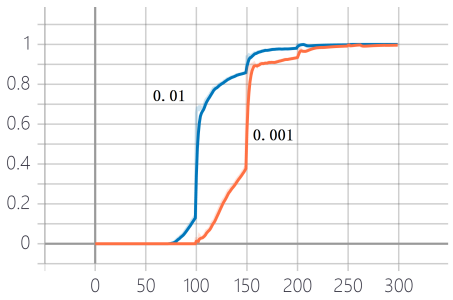

图 33: 不同学习率准确率曲线

通过上面改变学习率可以知道，学习率为0.01更好，下面来改变优化器，由Adam改为SGD，损失曲线如34所示，准确率曲线如图35所示。可以看出使用 SGD 训练的效果大不如Adam，原因是Adam自适应学习率算法对于稀疏数据具有优势，且收敛速度很快。

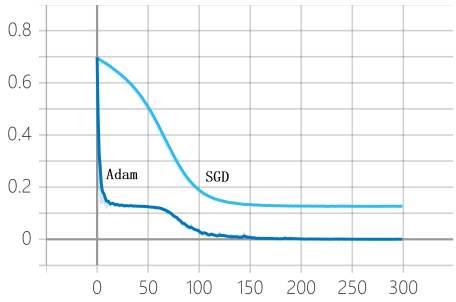

图 34: 不同优化器损失曲线

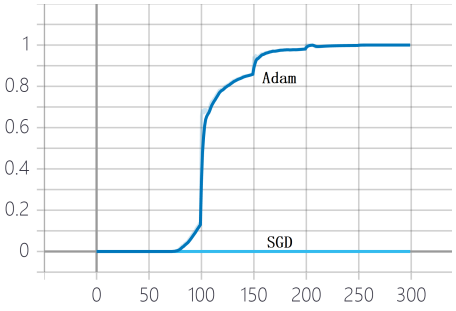

图 35: 不同优化器准确率曲线

5 与其他算法的比较

虽然在本次研究中，验证码的识别效果很好，但是还是有很多的方面值得优化。

5.1 迁移学习训练模型

迁移学习(Transfer Learning)是机器学习的一个研究方向，主要研究如何将任务A上面学习到的知识迁移到任务B上，以提高在任务B上的泛化性能。

例如任务A为猫狗分类问题，需要训练一个分类器能够较好的分辨猫和狗的样本图片，任务B为牛羊分类问题。可以发现，任务A和任务B存在大量的共享知识，比如这些动物都可以从毛发、体型、形态、发色等方面进行辨别。因

此在任务A训练获得的分类器已经掌握了这部份知识，在训练任务B的分类器时，可以不从零开始训练，而是在任务A上获得的知识的基础上面进行训练或微调(Fine-tuning)。通过迁移任务A上学习的知识，在任务B上训练分类器可以使用更少的样本和更少的训练代价，并且获得不错的泛化能力。如果使用迁移学习的方法，在训练验证码样本较复杂或者很多的时候，可以使用此方法快速训练，并且也可以得到很好的效果。

5.2 滴水算法分割

由于大部分的国内外大部分验证码都存在粘连，利用已有的垂直投影分割已无法很好的解决此类问题，需要使用其他算法，如：滴水分割算法。

滴水算法是一种经典的字符分割方法，基本思想是模拟水滴从高处向下滴落的滴落轨迹对粘连字符进行切分[16]。水滴从字符串顶端沿着字符轮廓向下滴落或者水平左右滚动，当水滴不能进一步下降或者水平滚动时，将竖直向下穿透字符笔画，然后继续向下滴落，最终水滴滴落轨迹即为字符的切分路径。针对粘连的验证码，使用垂直投影分割可能得不到较好的结果，若是选择滴水算法进行分割，解决就会好很多，分割出来的字符也比较完整。

6 结论

6.1 优缺点

6.1.1 优点

1) 卷积神经网络相较于全连接网络，对图像识别有更好的性能；
2) 对验证码图片进行了预处理，减少模型的训练时间与次数；
3) 对不同验证码使用不同的识别方法，有利于研究更多的识别方法。

6.1.2 缺点

1) 只是对两种验证码进行了识别分析，还有很多更加复杂的验证码需要更深入的研究，比如粘连性较多的验证码；
2) 由于计算机性能的限制，只使用CPU训练模型和预测所用时间较长，可以适当采用GPU加速训练；
3) 有些验证码的字符数目不是固定的，也有一些验证码是汉字组成，本文只对固定个数字符的验证码与数字、字母组成的验证码进行了分析。

6.2 展望

当前AlexNet、VGG、GooleNet及 ResNet等模型都在图像是被领域取得了很好的效果，如果能把这些模型的优势利用到验证码的识别中去，识别的正确

率也会有所上升，使用GPU加速训练和识别过程，尝试对不同种类的验证码进行识别。

验证码一直在更新换代，它的破解技术也一直在发展，只要有验证码存在，就会一直有人去研究它的破解技术。随着人工智能的发展，在将来，传统的验证码终将会被识别，对于设计者而言，基于图像和行为共同验证的高可靠性验证码将是未来的发展趋势。对于研究者而言，也要合理利用这一技术，不能用它去破坏现有的网络环境。

通过这次研究，在设计验证码时有效的安全特征有如下几个：字符之间相互粘连、较多的字符数量、采用特殊的字体和不影响人类识别的噪点与线条。无效的安全策略有：相似的背景图案、仅存在零星的噪点。

参考文献

[1] Von Ahn L,Blum M,Langford J. Telling humans and computers apart automatically[J]. Communications of the ACM,2004,47(2):56-60.

[2] 宋琦悦.扭曲粘连验证码识别算法研究[D]. 西安电子科技大学,2019.

[3] 王璐,张荣,尹东,詹金春,吴陈洋.粘连字符的图片验证码识别[J].计算机工程与应用,2011,47(28):150-153.

[4] Lu P,Shan L,Li J. A new segmentation method for connected characters in CAPTCHA[C]. 2015 International Conference on Control,Automation and Information Sciences (ICCAIS).IEEE,2015:128-131.

[5] 王枫,陈小平.CNN深度学习的验证码识别及Android平台移植[J].单片机与嵌入式系统应用,2019,19(07):20-22+73.

[6] Bottou L.Large-scale machine learning with stochastic gradient descent[M]. Proceedings of COMPSTAT'2010. Physica-Verlag HD,2010:177-186.

[7] Goodfellow I J,Bulatov Y,Ibarz J. Multi-digit number recognition from street view imagery using deep convolutional neural networks[J]. arXiv preprint arXiv:1312.6082,2013.

[8] LECUN Y,BENGIO Y,HINTON G.Deep learning[J]. Nature,2015,521(7553): 436-444.

[9] GANG L. Recognition of multi-fontstyle characters based on convolutional neural network[J]. Journal of Zhejiang Normal University,2011,2(1):223-225.

[10] COLLINS ACHEPSAH L,TSHILIDZI M. Introduction to Deep Learning[M]. Berlin:Springer,2017.

[11] MONTAVON G,Mü LLER K R. Big Learning and Deep Neural Networks[M]. Berlin:Springer,2012.

[12] Hamker F H. Predictions of a model of spatial attention using sum-and max-pooling functions[J]. Neurocomputing,2004,56:329-343.

[13] 李彦冬,郝宗波,雷航. 卷积神经网络研究综述[J]. 计算机应用,2016,36(9): 2508-2515.

[14] GONZALEZ, RAFAEL C, WOODS R E. Digital image processing[M]. London:Addison Wesley,2010.

[15] GROETSCH C W, YOST S A. Vertical Projection in a Resisting Medium:Re ections on Observa-tions of Mersenne[J]. American Mathematical Monthly,2014,121(6):499-505.

[16] 李兴国,高炜. 基于滴水算法的验证码中粘连字符分割方法[J]. 计算机工程与应用,2014,50(1):163-166.

[17] 张乐乐. 基于深度学习的图片验证码识别算法研究[D]. 青岛科技大学,2018.

A 相关代码

A.1 处理图片

```
import csv
import os
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

basedir = os.path.abspath(os.path.dirname(__file__))
filepath = os.path.join(basedir, 'samples')
file_list = os.listdir(filepath)

def binary_processing(img, threshold):
    """
    图片二值化处理
    """
    bin_img = []
    shape = img.shape
    img = img.flatten()
    for i in img:
        if i < threshold:
            bin_img.append(0)
        else:
            bin_img.append(255)

    return np.reshape(bin_img, shape)

def neighbor8filter(img, p):
    """
    八邻域去噪
    """
    pixels = img
    w, h = img.shape
    for x in range(w):
        pixels[x, 0] = 255
        pixels[x, h - 1] = 255
    for y in range(h):
        pixels[0, y] = 255
        pixels[w - 1, y] = 255
    for y in range(1, h - 1):
```

```
        for x in range(1, w - 1):
            if not pixels[x, y]:
                count = 0
                for m in range(x - 1, x + 2):
                    for n in range(y - 1, y + 2):
                        if not pixels[m, n]:
                            count = count + 1
                if count <= p:
                    pixels[x, y] = 255
    return img

def img_change():
    img_path = os.path.join(basedir, 'imgs')
    if not os.path.exists(img_path):
        os.makedirs(img_path)
    for item in file_list:
        img = Image.open(os.path.join(filepath, item))
        img = img.convert('L')
        img = np.array(img)
        img = binary_processing(img, 235)
        img = neighbor8filter(img, 3)
        img = neighbor8filter(img, 3)
        img = img.astype(np.uint8)
        img = Image.fromarray(img)
        img.save(os.path.join(img_path, item))

if __name__ == '__main__':
    img_change()
```

A.2 模型训练

```
import datetime
import os
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers, models, optimizers
from sklearn.model_selection import train_test_split

class CNN:
```

```
    def __init__(self):
        model = models.Sequential()

        model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu'))
        model.add(layers.MaxPool2D((2, 2), strides=2, padding='same'))
        model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
        model.add(layers.MaxPool2D((2, 2), strides=2, padding='same'))
        model.add(layers.Flatten())  # 会将三维的张量转为一维的向量
        model.add(layers.Dense(4 * 36, activation=None))
        model.build(input_shape=[None, 30, 90, 1])
        model.summary()
        model.compile(optimizer=optimizers.Adam(learning_rate=0.01),
                      metrics=[self.custom_acc],
                      loss=self.custom_loss)
        self.model = model

    @staticmethod
    def custom_loss(y_true, y_pred):
        """
        自定义损失函数
        """
        loss_list = tf.nn.sigmoid_cross_entropy_with_logits(labels=y_true,
                                                            logits=y_pred)
        loss = tf.reduce_mean(loss_list)
        return loss

    @staticmethod
    def custom_acc(y_true, y_pred):
        """
        自定义评估函数
        """
        equal_list = tf.reduce_all(
        tf.equal(tf.argmax(tf.reshape(y_pred, shape=[-1, 4, 36]), axis=2),
        tf.argmax(tf.reshape(y_true, shape=[-1, 4, 36]), axis=2)), axis=1)
        accuracy = tf.reduce_mean(tf.cast(equal_list, tf.float32))
        return accuracy

class DataSource:
    def __init__(self):
        self.csv_data = pd.read_csv("./labels.csv", names=["file_num", "chars"],
                                    index_col="file_num")
```

```
    def filename2char(self, image_path):
        """
        图片路径到验证码值
        """
        # print(image_path)
        num = image_path.split('/')[-1].split('.')[0]
        chars = self.csv_data.loc[int(num), "chars"]
        return chars

    def filename2label(self):
        """
        解析csv文件，建立文件名对应的标签值
        """
        labels = []
        for label in self.csv_data["chars"]:
            tmp = []
            for letter in label:
                try:
                    tmp.append(int(letter))
                except ValueError:
                    tmp.append(ord(letter) - ord("a") + 10)
            labels.append(tmp)
        self.csv_data["labels"] = labels
        return labels

    @staticmethod
    def load_and_preprocess_from_path_label(path, label):
        """
        处理验证码与标签
        """
        image = tf.io.read_file(path)
        image_decode = tf.image.decode_jpeg(image, channels=1)
        image_decode = tf.image.resize(image_decode, [30, 90])
        # 归一化处理
        image_decode /= 255.0
        label_decode = tf.one_hot(label, depth=36)
        label_decode = tf.reshape(label_decode, shape=(4 * 36,))

        return image_decode, label_decode

    def read_picture(self):
        """
```

```
        读取验证码图片
        """
        basedir = os.path.abspath(os.path.dirname(__file__))
        filepath = os.path.join(basedir, 'imgs')
        all_image_paths = [os.path.join(filepath, i)
                           for i in os.listdir(filepath)]
        all_image_labels = self.filename2label()
        x_train, x_test, y_train, y_test = train_test_split(all_image_paths,
                            all_image_labels, test_size=150, random_state=32)
        dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
        dataset = dataset.map(self.load_and_preprocess_from_path_label)

        train_dataset = dataset.shuffle(1600)
        train_dataset = train_dataset.repeat()
        train_dataset = train_dataset.batch(32)
        test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test))
        test_dataset = test_dataset.map(self.load_and_preprocess_from_path_label)
        test_dataset = test_dataset.shuffle(150)
        return train_dataset, test_dataset

class Train:
    def __init__(self):
        self.cnn = CNN()
        self.data = DataSource()

    def train(self):
        train_dataset, test_dataset = self.data.read_picture()
        log_dir = "logs\\fit\\" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
        callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
                                                  update_freq='batch')
        if os.path.exists('simple.h5'):
            model = keras.models.load_model('simple.h5', custom_objects={
            'custom_loss': self.cnn.custom_loss,
            'custom_acc': self.cnn.custom_acc
            })
        else:
            model = self.cnn.model
            print(train_dataset)
            model.fit(train_dataset, epochs=6, steps_per_epoch=50,
                                callbacks=[callback])
            model.save('simple.h5')
        results = model.evaluate(test_dataset.batch(32))
```

```
        print('test loss, test acc:', results)

        for step, (x, y) in enumerate(test_dataset):
            pred = model.predict(tf.expand_dims(x, axis=0))
            # print(pred.shape)  # (1, 144)
            pred = tf.reshape(pred, shape=(-1, 4, 36))
            # print(pred.shape)  # (1, 4, 36)
            pred = tf.argmax(pred, axis=2).numpy()[0]
            pred = ''.join([str(i) if i < 10 else chr(ord("a") + i - 10)
                                        for i in pred])

            true_label = tf.argmax(tf.reshape(y, (4, 36)), axis=-1).numpy()
            true_label = ''.join([str(i) if i < 10 else chr(ord("a") + i - 10)
                                  for i in true_label])
            # print(true_label)
            plt.subplot(3, 4, step + 1)
            x = tf.squeeze(x, axis=-1)
            plt.imshow(x, cmap="gray")
            plt.xlabel(true_label+'-'+pred)
            if step == 11:
                plt.show()
                break

if __name__ == '__main__':
    app = Train()
    app.train()
```