课程名称：图像处理与机器视觉

报告题目：OpenCV-CNN手写汉字分割与识别

xxxxx

指导教师：xxx

提交日期： 2025年 5月 11 日

一、摘要

手写汉字识别是人机交互的一种重要形式，在票据处理、文件录入等领域具有很高的实用价值。但因汉字的字符数量众多、每个人书写风格各异等原因，手写汉字识别的准确率一直不高。近年来，深度学习在图像识别、目标检测等领域取得了突破性的进展，逐渐成为了模式识别问题中常用的方法。本文基于深度学习的方法，综合利用特征提取技术对脱机手写汉字的识别进行了研究，使其识别准确率得到提升[1]。本项目针对手写汉字识别任务，设计并实现了一个基于OpenCV和卷积神经网络的汉字分割与识别系统。系统首先对扫描输入的文本图像进行预处理和字符分割，提取出单个汉字图像；然后使用卷积神经网络进行分类识别。通过引入图像增强、图像变换、图像分割等机制，模型在测试集上取得了较高的识别准确率。同时，为了验ResNet的优势，我们将其与设计的和LeNet等经典网络在同一数据集上进行对比训练和测试。 实验结果表明，ResNet模型在汉字识别任务中具有更高的准确率和收敛速度。我们记录了训练集与测试集准确率随迭代的变化曲线，并对识别结果进行了可视化展示。从结果分析可以看出，ResNet网络更深的结构和残差连接有助于提取复杂笔画特征，提高了对相似字符的区分能力，验证了其在大规模类别识别任务中的稳定性和有效性，具有较强的实用推广价值。

二、引言

1、背景介绍

随着社会信息化的发展，大量文档需要数字化处理。汉字作为使用最广泛的文字系统之一， 其手写识别问题具有重要意义。手写汉字的结构复杂、类别众多，而且不同书写者风格差异大，因此准确识别手写汉字仍面临较高难度。汉字识别通常包括字符分割和字符分类两个阶段：字符分割将连写文本拆分为单个汉字图像，字符分类则将每个汉字图像映射到对应类别。卷积神经网络在图像分类领域取得了显著效果。

2、研究目标

本项目旨在构建一个高效的手写汉字分割与识别系统，实现从原始图像到字符分类的完整流程。具体目标包括：1）利用OpenCV技术实现对.gnt格式图像中汉字的自动分割与预处理；2）构建基于深度学习（LeNet、VGG、ResNet）的分类模型，对分割后的汉字进行准确识别；3）通过对图片进行分割后通过模型进行预测，得到识别后的汉字。

3、项目意义

手写汉字识别在文档数字化、智能输入、教育辅助等场景中具有广泛应用价值。相比传统的 特征工程方法，深度学习具有端到端学习能力，可以自动提取表征汉字结构的特征，提高识别准确率。项目通过结合OpenCV字符分割和深度神经网络分类，验证了现代深度学习模型在手写汉字识别方面的有效性，为后续开发更高性能的汉字识别系统奠定基础。

三、相关工作

1、文献资料

脱机手写体字符识别常用的方法一般分为三类：模板匹配法[2]、特征提取与识别[3]和深度学习[4]的方法。模板匹配法是OCR早期最常用的方法，它首先要对所识别的字符建立一个标准模板集,其次将要识别的对象依次与模板集中的模板进行匹配，相似度最高的模板所对应的字符即为识别结果；特征提取与识别法则由两部分构成，第一步使用某种算法（如Scale-Invariant Feature Transform，SIFT、Speeded Up Robust Features，SURF、Histogram of Oriented Gradient，HOG等）来提取文字图像的特征；第二步使用一种分类器（如贝叶斯分类器、支持向量机、K近邻算法、人工神经网络算法等）对所获取的特征进行分类；深度学习的方法是当前最流行的方法，它几乎己经取代了传统的字符识别方法。自LeNet-5以来卷积神经网络在图像识别领域中取得了巨大成就，它越来越多地应用于图像分类、字符识别等任务中并取得了很好的效果。因此，将深度学习的方法应用到脱机手写体汉字识别[5]中对汉字识别的研究和发展有极其重要的意义。

2、方法比较

自20世纪80年代以来，脱机手写体汉字识别（Handwritten Chinese Character Recognition, HCCR）在模式识别领域得到了广泛的研究和关注。由于汉字类别数目庞大、相似字符较多、书写者的随意性和不规范性、硬件条件的限制等，使得手写体汉字识别（HCCR）一直是一个极具挑战性的模式识别及机器学习问题。经过40多年来无数研究者的不懈努力，脱机手写体汉字识别取得了极大进展。

文献[6]针对小样本脱机手写体汉字数据集，采用了弹性网格方向像素的概率分布特征来获取样本特征，然后将获取的样本特征作为极限学习分类器的输入。在分类器的设计上，使用了基于最优投票机制的极限学习机优化算法，利用OAO（One-Against-One）或OAA（One-Against-All）的思想将一个多分类问题降解为多个二分类问题来提高分类的准确率。文献作者仅在少量类别的样本上进行了实验，识别率为95.40%。文献[7]提出了一种基于粒子群和支持向量机（Support Vector Machine, SVM）的研究方法，即利用粒子群算法找出最优的SVM参数，从而解决了SVM参数选择的困难。该方法最终的识别率为87.67%。

文献[8]提出了一种基于重要抽样的大规模分类任务判别学习方法，该方法根据贝叶斯分类规则的样本重要性权重函数，对不同权重的样本进行赋值，并采用多种方法计算改进的二次判别函数（Modified Quadratic Discriminant Function, MQDF）的样本重要性权重。这种策略允许MQDF在保持较低计算复杂性的同时获得更高的准确性，文献作者在三种脱机手写体汉字数据集上验证了该方法的有效性。

近年来，传统的模式识别方法几乎没有较大的研究进展，随着深度学习技术的兴起及不断发展，传统方法（如基于统计特征、结构分析的模型）逐渐被取代，深度学习为脱机手写体汉字识别带来了新的活力和非常有效的解决方法。特别是卷积神经网络（Convolutional Neural Network, CNN）、深度置信网络（Deep Belief Network, DBN）、层叠自动编码机（Stacked Autoencoder, SAE）、残差网络（Residual Networks）等深度神经网络模型在语音识别、图像分类及计算机视觉领域取得了突破性进展，因此这些方法也被逐步应用到OCR领域中。

四、方法

1、数据集

在本实验中，采用的是中文手写字符识别领域权威的HWDB1.1数据集（Handwritten Chinese Character Database Version 1.1），该数据集由中国科学院自动化研究所国家模式识别实验室（NLPR）提供，广泛用于离线手写汉字识别算法的研究与评估。HWDB1.1数据集涵盖了GB2312一级字表中的全部3755个常用汉字，由300位书写者使用电子笔在电子纸上书写标准样本采集而成。每个汉字约包含240个样本，训练集总计约898,000张灰度图像，测试集约224,000张图像，由另外60位不同的书写者书写，确保训练与测试集风格互异、无重叠，增强了模型泛化评估的科学性。

数据以.gnt二进制格式存储，包含每个字符图像及其标签编码。图像原始尺寸不一，通常需进行统一预处理（如灰度化、二值化、大小归一化、去噪等）。为了适应模型训练的需要，实验将原始数据按照训练集、测试集按8:2的比例划分，其中验证集从训练集中划出，用于调整模型超参数和防止过拟合。该数据集字符类别覆盖面广、书写风格多样，具备良好的代表性与挑战性，因此在评估卷积神经网络（如LeNet、ResNet、DenseNet）等模型在中文字符识别任务上的性能时具有高度参考价值。整体来看，HWDB1.1的规模、分布及划分设计为本实验提供了扎实、可靠的实验基础。

1. 数据预处理

在本实验中，为了提高模型的训练效果与泛化能力，我们对手写汉字图像数据进行了系统性的预处理，主要包括图像归一化、二值化、尺寸标准化以及多种数据增强策略。具体步骤如下：

首先，在预处理阶段，我们遍历整个数据目录，依次打开每个 .gnt 文件，并对其中的每一条样本记录进行读取与解析。.gnt 文件的每个样本由10字节的头部信息和后续的图像数据组成，我们通过读取头部内容获取图像的宽高、标签编码以及样本长度等信息。图像的标签以GB2312编码存储，我们将其解析为对应的汉字字符，并构建一个去重后的字符集合，用于生成统一的字符索引字典，该字典在整个训练和测试过程中起到了关键作用，使模型输出与实际标签能够一一对应。

接着，为了适应后续模型训练，我们对图像进行一系列标准化处理。原始图像是灰度图像，我们首先将其进行二值化处理，阈值设为200，将背景变为白色、前景字变为黑色；同时，为更符合常见图像输入格式，我们还执行了颜色反转处理，使汉字变为黑底白字，这样更有利于深度卷积网络提取边缘和纹理信息。此外，所有图像被统一调整为指定大小（64×64），从而保证输入维度一致，避免模型因尺寸不统一而出错。

在此基础上，我们进一步引入了数据增强机制，尤其在训练集加载过程中启用。增强手段包括随机的亮度与对比度扰动、椒盐噪声与高斯噪声添加、模糊与锐化处理、以及随机旋转等。这些策略有效模拟了现实中可能出现的字迹轻重不一、笔迹抖动、扫描模糊等干扰因素，从而提升了模型在实际应用场景中的鲁棒性和泛化性能。实验中也观察到，适当的数据增强策略能够缓解过拟合问题，使模型在测试集上取得更为稳定的表现。

上述过程具体如图1所示：

|  |
| --- |
|  |
| 图1：数据预处理流程 |

3、模型结构

1、Residual Block（残差块）

Residual Block是ResNet的基本构建模块，它引入了恒等映射，将输入直接添加到输出中，构成如下形式的残差函数：

其中，为输入，是待学习的残差函数，通常由两个3×3的卷积层及其对应的Batch Normalization和ReLU组成。通过这种设计，网络不再直接学习复杂的映射，而是学习输入和目标输出之间的“残差”，更容易优化。当输入和输出维度不一致时，采用1×1卷积进行维度变换，保证残差加法的可行性。利用这种shortcut短路连接机制，有效解决了深层网络训练的性能退化问题，提高了模型的收敛速度和泛化能力。

残差块的具体实现如图2所示。

|  |
| --- |
|  |
| 图2：包含以及不包含1×1卷积层的残差块 |

3、ResNet 网络结构

ResNet的前两层跟GoogLeNet中的一样：在输出通道数为64、步幅为2的7 × 7卷积层后，接步幅为2的3 × 3的最大汇聚层。不同之处在于ResNet每个卷积层后增加了批量规范化层。GoogLeNet在后面接了4个由Inception块组成的模块。ResNet则使用4个由残差块组成的模块，每个模块使用若干个同样输出通道数的残差块。第一个模块的通道数同输入通道数一致。由于之前已经使用了步幅为2的最大汇聚层，所以无须减小高和宽。之后的每个模块在第一个残差块里将上一个模块的通道数翻倍，并将高和宽减半。最后，在ResNet中加入全局平均汇聚层，以及全连接层输出。每个模块有4个卷积层（不包括恒等映射的1 × 1卷积层），4 个模块 × 4个卷积层 = 16 层。加上第一个7 × 7卷积层和最后一个全连接层，共有18层。因此，这种模型通常被称为ResNet‐18。通过配置不同的通道数和模块里的残差块数可以得到不同的ResNet模型，例如更深的含152层的ResNet‐152。虽然ResNet的主体架构跟GoogLeNet类似，但ResNet架构更简单，修改也更方便。这些因素都导致了ResNet迅速被广泛使用。图3描述了完整的ResNet‐18结构。

|  |
| --- |
|  |
| 图3：ResNet‐18 架构 |

4、实验过程

（1）实验配置

笔记本电脑（i5-12500H+RTX2050-4G）

云容器配置：Ubuntu 20.04、10vCPU、RTX3090、1卡 \* 24 GB显存

python3.8、pytorch 2.4.1等。

（2）模型超参数

|  |  |
| --- | --- |
| 超参数 | 数值 |
| Learning rate | 0.01 |
| Batch\_size | 512 |
| epochs | 50 |
| optimizer | SGD |

（3）模型训练中gpu资源占用情况

|  |
| --- |
|  |
| 图4: ResNet训练时GPU资源占用情况 |

4、结果分析

（1）数据集预处理结果

|  |
| --- |
|  |
| 图5：char\_list结果输出 |

（2）MyDataset类处理结果

在MyDataset类中，我们加入了颜色抖动（在二值化图像上调整对比度和亮度）、椒盐噪声、高斯模糊，锐化、随机旋转（-5到5度），来模拟现实中可能会遇到的情况。下面展示的是部分图像变化结果。

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
| 处理前 | 处理后 |

（3）模型训练结果

|  |  |  |
| --- | --- | --- |
|  | ResNet增强前 | ResNet增强后 |
| 训练集准确率 | 99.92% | 99.69% |
| 测试集准确率 | 84.89% | 87.62% |
| 召回率 | 84.89% | 87.62% |
| F1分数 | 84.60% | 87.64% |
| Loss  变化 |  |  |
| 准确率  变化 |  |  |
| 耗时 | 01:09:51（时:分:秒） | 01:34:55（时:分:秒） |

训练集准确率下降（由99.92%降至99.69%），表明数据增强引入了更多样本变化，抑制了模型对训练集的过拟合；增强后训练集准确率下降，说明模型的过拟合程度减轻了，相对来说，泛化能力更好。测试集准确率略有提升（从84.89%到87.62%），说明增强策略有效地提高了模型在未见样本上的泛化能力；召回率和 F1 分数均有所提升，进一步验证了增强后模型在查全查准方面的综合性能增强；训练耗时显著增加（由约1小时提升至1.5小时），因为每个batch额外执行了随机旋转、噪声、模糊、锐化处理等多种增强操作，增加了数据加载与预处理开销。

综合来看，虽然数据增强带来了训练时间的开销，但通过多样化样本模拟，模型的泛化性能得到了提升，尤其是在测试集的召回率和F1分数上都有所增益，符合本项目对鲁棒性和准确性的双重追求。

同时，在本项目中，还测试了其他的卷积神经网络对汉字识别的效果，其中，通过手写了一个简单的只有3个卷积层和3个全连接层的简单CNN来训练，为了测试不同的卷积神经网络的效果，我们使用了经典的神经网络LeNet，在LeNet上做了改进并进行训练，得到的实验结果如下：

|  |  |  |
| --- | --- | --- |
|  | 简单CNN | 改进的LeNet |
| 测试集准确率 | 66.00% | 81.00% |
| Loss  变化 |  |  |
| 准确率变化 |  |  |
| 耗时 | 01:53:11（时:分:秒） | 00:29:10（时:分:秒） |

实验结果表明，ResNet在识别准确率和泛化能力方面明显优于简单CNN模型。简单CNN模型在训练50轮后，虽然训练准确率达到了95.82%，但测试准确率仅为 66.00%，存在一定程度的过拟合现象；表明该浅层CNN结构简单，提取特征、表达能力受限。

实验结果表明，ResNet在识别准确率和泛化能力方面明显优于简单CNN模型。简单CNN模型在训练50轮后，虽然训练准确率达到了95.82%，但测试准确率仅为 66.00%，存在一定程度的过拟合现象；表明该浅层CNN结构简单，提取特征、表达能力受限。而改进版的LeNet神经网络在汉字识别任务中的效果。该模型结构其他卷积神经网络来说，准确率提高巨大，训练耗时较短，仅为29分钟，训练过程稳定，收敛速度较快。尽管模型较浅，参数量较少，但在处理简单样本时具有一定优势，适合资源受限的场景。

为了测试参数量更大的模型，我们还进行了VGG11和ResNet34的训练，实验结果如下：

|  |  |  |
| --- | --- | --- |
|  | ResNet34 | VGG11 |
| 测试集准确率 | 85.14% | 90.00% |
| Loss  变化 |  |  |
| 准确率变化 |  |  |
| 耗时 | 02:00:16（时:分:秒） | 01:34:55（时:分:秒） |

HWDB1.1的图像为 64×64，是单通道灰度图，结构上比较接近于手写字（笔画细节密集）。ResNet是为ImageNet设计的（224×224，RGB三通道），其初始卷积核是 7×7 步长 2 的大卷积，步幅大，下采样太快，可能容易丢失细节。VGG11的初始卷积核是 3×3，小步长保留细节更充分，可能更适合识别笔画细微变化。残差连接本质上是y = F(x)+x，适合处理深层网络中的梯度消失问题。但在“字符图像识别”这类对纹理、局部结构敏感的任务中：残差结构有时可能抑制了一些重要的笔画变化。这里可能需要进行对ResNet进行改进来适应我的数据集，由于时间关系（训练一次需要2-3小时，以上神经网络设计加训练时长长达30+小时），所以暂时没有尝试。

综上所述，在超参数（Learning rate = 0.01，Batch\_size = 512，epochs = 50，optimizer = SGD）的情况下，效果较好的卷积神经网络为改进版的VGG11和ResNet34。其中，对图像进行颜色抖动、椒盐噪声、高斯模糊，锐化、随机旋转（-5到5度）之后，会使得模型的测试集准确率更高，泛化性能更优。

（4）字符分割结果

为了验证我们字符分割的有效性，我们分别找两位同学在A4纸上进行写字，然后拍照，保存为png格式图片，运行代码后获得如下的分割结果：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 原始图像 | 分割结果 | | | | | | | |
|  |  |  | |  | |  | |  |
|  |  | |  | |  | |  | |

可以看到，我们可以有效的把手写汉字分割开。

（5）模型识别结果

针对上面的两个图片，我们分别传入训练好的模型进行预测，得到的识别结果如下：

|  |
| --- |
|  |
| “计算机视觉”识别结果 |
|  |
| “霜天黄叶”识别结果 |

本次测试的两个结果均正确，验证了我们训练得到的模型十分有效。

五、总结

本项目基于OpenCV与深度学习技术，构建了一套完整的手写汉字分割与识别系统，成功实现了从文本图像预处理、字符分割到分类识别的全流程自动化。通过系统化的数据增强策略和多种卷积神经网络模型的对比实验，验证了残差网络（ResNet）在汉字识别任务中的显著优势。实验结果表明，改进后的ResNet34模型在HWDB1.1数据集上取得了87.62%的测试集准确率，较传统方法（如LeNet）提升超过20%。数据增强策略通过引入随机旋转、噪声干扰和几何变换，有效缓解了模型过拟合问题，使测试集泛化性能提升近3%。此外，基于OpenCV的字符分割算法结合轮廓合并与中值滤波技术，解决了手写汉字粘连与断裂的难题，并通过实际拍摄样本验证了分割流程的鲁棒性。这些成果不仅验证了深度学习在复杂文字识别中的有效性，也为文档数字化、教育辅助等场景提供了技术支撑。

研究目标通过三方面实现：首先，利用OpenCV技术完成了.gnt格式图像的高效解析与字符分割，支持复杂书写场景下的单字提取；其次，通过对比ResNet、VGG等模型的性能差异，揭示了深层网络对汉字笔画特征的捕捉能力；最后，在实际测试中实现了“计算机视觉”“霜天黄叶”等手写样本的准确识别。项目通过可视化训练曲线与混淆矩阵分析，验证了残差连接在缓解梯度消失、提升模型收敛速度方面的作用，同时发现了传统网络（如简单CNN）因结构限制导致的过拟合问题，为后续模型优化提供了数据支持。

展望未来，研究可从多维度深入探索。在模型设计上，可针对汉字笔画密集的特性，引入轻量化网络（如MobileNet）或注意力机制（如Transformer），优化局部特征的提取效率；同时，结合生成对抗网络（GAN）合成多样化手写样本，或通过风格迁移技术模拟不同书写风格，进一步提升模型对复杂场景的适应性。在系统集成方面，可探索分割与识别模块的端到端联合训练，减少两阶段误差传递的影响；针对连笔字、倾斜文本等实际场景，需优化分割算法的抗干扰能力，并引入图神经网络（GNN）建模汉字间的上下文语义关联。随着边缘计算技术的发展，模型轻量化与移动端部署将成为重要方向，推动教育辅助、档案数字化等应用的落地。此外，跨语言迁移学习可扩展至多语种汉字（如日文、韩文）的联合识别框架，促进OCR技术的跨文化应用。这些探索将进一步提升手写汉字识别的实用价值，助力文化遗产保护与信息化社会建设。

六、参考文献

[1]袁柱. 基于深度学习的脱机手写汉字识别的研究与应用[D]. 广州：广东工业大学, 2020.Yuan Zhu. Research and Application of Offline Handwritten Chinese Character Recognition Based on Deep Learning[D]. Guangzhou: Guangdong University of Technology, 2020. University of Technology,2020.

[2]高聪,王福龙,GAOCong,等.基于模板匹配和局部HOG特征的车牌识别算法[J].计算机应用,2017,26（1）:122-128

[3]李建林.一种基于PCA的组合特征提取文本分类方法[J].计算机应用研究,2013,30(8):2398-2401

[4] Lecun Y, Bengio Y, Hinton G, Deep learn[J]. Nature,2015,521(7553):436.

[5]赵继印，郑蕊蕊，吴春宝,等.脱机手写体汉字识别综述[J].电子学报,2010,38(2):405-415.

[6]杨弋.基于极限学习机的脱机手写体汉字识别研究[D].东北大学,2012.

[7]陈永春.基于粒子群算法和支持向量机的脱机手写体汉字识别研究[D].天津师范大学,2013.

[8]Wang,Yangwei,Fu,Qiang,Ding,Xiaoqing,et al.Importance sampling based discriminative learning for large scale offline handwritten Chinese character recognition[J].Pattern Recognition: The Journal of the Pattern Recognition Society, 2015,48(4):1225-1234

六、附录

1、核心代码

（1）从测试集中解析所有.gnt文件，逐样本读取图像与标签，通过GB2312 解码提取汉字字符，构建字符集合并生成字符到索引的映射字典，最终将该字典使用pickle序列化保存，用于模型训练和预测时的标签映射。

import os  
import pickle  
import struct  
import numpy as np  
from config import args

train\_data\_dir = os.path.join(args.root, 'HWDB1.1trn\_gnt')  
test\_data\_dir = os.path.join(args.root, 'HWDB1.1tst\_gnt')

*# 获取字典*char\_set = set()  
data\_path = test\_data\_dir  
*#从.gnt文件中逐样本读取图像和对应标签的二进制数据，并解析为灰度图像和字符编码。*for file\_name in os.listdir(data\_path):  
 if file\_name.endswith('.gnt'):  
 file\_path = os.path.join(data\_path, file\_name)  
 with open(file\_path, 'rb') as f:  
 header\_size = 10  
 while True:  
 header = np.fromfile(f, dtype='uint8', count=header\_size)  
 if not header.size:  
 break  
 sample\_size = header[0] + (header[1] << 8) + (header[2] << 16) + (header[3] << 24)  
 tagcode = header[5] + (header[4] << 8)  
 width = header[6] + (header[7] << 8)  
 height = header[8] + (header[9] << 8)  
 if header\_size + width \* height != sample\_size:  
 break  
 image = np.fromfile(f, dtype='uint8', count=width \* height).reshape((height, width))  
 *#将 tagcode 按大端方式打包成2字节，再按 gb2312 解码为对应汉字。  
 #加入到 char\_set 中* tagcode\_unicode = struct.pack('>H', tagcode).decode('gb2312')  
 char\_set.add(tagcode\_unicode)  
  
char\_list = sorted(list(char\_set)) *# set是随机的，需要排一下序  
#创建字符→索引的字典*char\_dict = dict(zip(char\_list, range(len(char\_list))))  
*#使用pickle序列化保存该字典到本地*f = open(args.root + '/char\_dict', 'wb')  
pickle.dump(char\_dict, f) *# 将码表写入到数据集*f.close()  
*# print(char\_list)*

（2）自定义数据集类用于加载 .gnt 文件并支持图像增强与预处理，生成训练所需图像与标签数据。

import cv2  
import numpy as np  
import random  
from PIL import Image, ImageEnhance, ImageFilter  
from torch.utils.data import Dataset  
import struct  
import pickle  
from config import args  
import os  
  
class MyDataset(Dataset):  
 def \_\_init\_\_(self, data\_path, transforms, augment=False):  
 super(MyDataset, self).\_\_init\_\_()  
 self.images = []  
 self.labels = []  
 self.transforms = transforms  
 self.augment = augment *# 是否启用数据增强* self.get\_data(data\_path)  
  
 def \_\_getitem\_\_(self, index):  
 image = self.images[index]  
 label = self.labels[index]  
  
 *# 数据增强* if self.augment:  
 image = self.apply\_augmentations(image)  
  
 *# 应用基础变换* image = self.transforms(image)  
 return image, label  
  
 def \_\_len\_\_(self):  
 return len(self.labels)  
  
 def get\_data(self, data\_path): *# 读取并解析.gnt文件* with open(args.root + '/char\_dict', 'rb') as f:  
 char\_dict = pickle.load(f)  
  
 for file\_name in os.listdir(data\_path):  
 if file\_name.endswith('.gnt'):  
 file\_path = os.path.join(data\_path, file\_name)  
 with open(file\_path, 'rb') as f:  
 *# .gnt文件的每个样本头占用10个字节。* header\_size = 10  
  
 while True:  
 header = np.fromfile(f, dtype='uint8', count=header\_size)  
 if not header.size: break  
 *# 解析样本字节数（低位在前，高位在后)* sample\_size = header[0] + (header[1] << 8) + (header[2] << 16) + (header[3] << 24)  
 *# 解析汉字标签（使用GB2312编码转为字符），并转为分类编号。* tag\_code = header[5] + (header[4] << 8)  
 tag\_code = struct.pack('>H', tag\_code).decode('gb2312')  
 label = char\_dict[tag\_code]  
 *# 获取图像宽高* width = header[6] + (header[7] << 8)  
 height = header[8] + (header[9] << 8)  
 *# 确保样本大小合理* if header\_size + width \* height != sample\_size: break  
 *# 读取图像数据* image = np.fromfile(f, dtype='uint8', count=width \* height).reshape((height, width))  
 *# 将图像二值化并反色（黑底白字 → 白底黑字)* \_, image = cv2.threshold(image, 200, 255, cv2.THRESH\_BINARY\_INV)  
 *# 统一图像尺寸为模型输入需要的大小* image = cv2.resize(image, (args.image\_size, args.image\_size))  
 *# 存入图像与对应标签。* self.images.append(image)  
 self.labels.append(label)  
  
 def apply\_augmentations(self, image):  
 *"""应用随机数据增强"""  
 # 转换为PIL Image进行增强处理* pil\_img = Image.fromarray(image)  
  
 *# 颜色抖动（在二值化图像上调整对比度和亮度）* if random.random() < 0.5:  
 enhancer = ImageEnhance.Contrast(pil\_img)  
 pil\_img = enhancer.enhance(random.uniform(0.8, 1.2))  
  
 if random.random() < 0.5:  
 enhancer = ImageEnhance.Brightness(pil\_img)  
 pil\_img = enhancer.enhance(random.uniform(0.8, 1.2))  
  
 *# 转换为numpy数组进行OpenCV处理* image = np.array(pil\_img)  
  
 *# 椒盐噪声* if random.random() < 0.1:  
 noise\_type = random.choice(['gaussian', 'saltpepper'])  
 if noise\_type == 'saltpepper':  
 prob = 0.02  
 output = np.copy(image)  
 r = np.random.rand(\*image.shape)  
 output[r < prob / 2] = 0  
 output[r > 1 - prob / 2] = 255  
 image = output  
 else: *# 高斯噪声（需临时转换为浮点型）* gauss = np.random.normal(0, random.uniform(0, 15), image.shape)  
 noisy = image.astype(np.float32) + gauss  
 image = np.clip(noisy, 0, 255).astype(np.uint8)  
  
 *# 高斯模糊，锐化* if random.random() < 0.5:  
 if random.random() < 0.5:  
 image = cv2.GaussianBlur(image, (3, 3), 0)  
 else:  
 kernel = np.array([[-1, -1, -1],  
 [-1, 9, -1],  
 [-1, -1, -1]])  
 image = cv2.filter2D(image, -1, kernel)  
  
 *# 几何变换  
 # 随机旋转（-5到5度）* if random.random() < 0.7:  
 angle = random.uniform(-5, 5)  
 h, w = image.shape  
 center = (w // 2, h // 2)  
 M = cv2.getRotationMatrix2D(center, angle, 1.0)  
 image = cv2.warpAffine(image, M, (w, h),  
 borderMode=cv2.BORDER\_REPLICATE)  
  
 return image

（3）实现了六个经典卷积神经网络模型：CNN、ResNet（手写改进）、VGG11、LeNet、ResNet18和ResNet34，用于汉字分类（3755类），输入图像大小为 1×64×64。：

CNN：两层卷积池化，接三层全连接，结构简洁。

ResNet18（手写改进）：多层残差模块堆叠，缓解梯度消失，性能稳定。

VGG11：堆叠小卷积核，特征提取能力强，参数量大。

LeNet：早期经典网络，结构简单，但分类能力有限。

ResNet18：多层残差模块堆叠，缓解梯度消失，性能稳定。

ResNet34：多层残差模块堆叠，缓解梯度消失，性能稳定。

from torch import nn  
import Residual\_block  
import vgg\_block  
from d2l import torch as d2l  
from torchvision.models import resnet18  
  
class CNN(nn.Module):  
 def \_\_init\_\_(self):  
 super(CNN, self).\_\_init\_\_()  
 self.features = nn.Sequential(  
 nn.Conv2d(1, 16, 3, stride=1, padding=1), *# 16\*64\*64* nn.ReLU(),  
 nn.MaxPool2d(2, 2), *# 16\*32\*32* nn.Conv2d(16, 32, 3, stride=1, padding=0), *# 32\*30\*30* nn.ReLU(),  
 nn.MaxPool2d(2, 2)) *# 32\*15\*15* self.flatten = nn.Flatten()  
  
 self.classifier = nn.Sequential(  
 nn.Linear(32 \* 15 \* 15, 4096),  
 nn.Linear(4096, 4096),  
 nn.Linear(4096, 3755))  
  
 def forward(self, x):  
 x = self.features(x)  
 x = self.flatten(x)  
 x = self.classifier(x)  
 return x  
  
  
class ResNet(nn.Module):  
 def \_\_init\_\_(self):  
 super(ResNet, self).\_\_init\_\_()  
  
 self.layer1 = nn.Sequential( *# bs\*1\*64\*64* nn.Conv2d(1, 64, kernel\_size=7, stride=2, padding=3), *# 64\*32\*32* nn.BatchNorm2d(64),  
 nn.ReLU(inplace=True),  
 nn.MaxPool2d(kernel\_size=3, stride=2, padding=1) *# 64\*16\*16 (32+2-3/2)+1* )  
 *# 第2层64\*16\*16(2个resnet\_block，4个卷积层）* self.layer2 = nn.Sequential(\*Residual\_block.resnet\_block(64, 64, 2, first\_block=True))  
 *# 第3层通道数翻倍，高宽减半128\*8\*8（2个resnet\_block，4个卷积层）* self.layer3 = nn.Sequential(\*Residual\_block.resnet\_block(64, 128, 2))  
 *# 第4层通道数翻倍，高宽减半256\*4\*4（2个resnet\_block，4个卷积层）* self.layer4 = nn.Sequential(\*Residual\_block.resnet\_block(128, 256, 2))  
 *# 第5层通道数翻倍，高宽减半512\*2\*2（2个resnet\_block，4个卷积层）* self.layer5 = nn.Sequential(\*Residual\_block.resnet\_block(256, 512, 2))  
  
 self.avgpool = nn.AdaptiveAvgPool2d((1, 1)) *# 平均池化512\*1\*1* self.flatten = nn.Flatten()  
 self.fc = nn.Linear(512, 3755) *# 512->3755* self.\_initialize\_weights()  
  
 def \_initialize\_weights(self):  
 for m in self.modules():  
 if isinstance(m, nn.Conv2d):  
 nn.init.kaiming\_normal\_(m.weight, mode='fan\_out', nonlinearity='relu')  
 if m.bias is not None:  
 nn.init.constant\_(m.bias, 0)  
 elif isinstance(m, nn.Linear):  
 nn.init.normal\_(m.weight, 0, 0.01)  
 nn.init.constant\_(m.bias, 0)  
  
 def forward(self, X):  
 X = self.layer1(X)  
 X = self.layer2(X)  
 X = self.layer3(X)  
 X = self.layer4(X)  
 X = self.layer5(X)  
 X = self.avgpool(X)  
 X = self.flatten(X)  
 net = self.fc(X)  
 return net  
  
class ResNet18(nn.Module):  
 def \_\_init\_\_(self):  
 super(ResNet18, self).\_\_init\_\_()  
 self.model = resnet18(weights=None) *# 或 weights='DEFAULT' 使用预训练* self.model.conv1 = nn.Conv2d(1, 64, kernel\_size=7, stride=2, padding=3, bias=False) *# 单通道适配* self.model.fc = nn.Linear(512, 3755)  
  
 def forward(self, x):  
 return self.model(x)  
from torchvision.models import resnet34  
import torch.nn as nn  
  
class ResNet34(nn.Module):  
 def \_\_init\_\_(self):  
 super(ResNet34, self).\_\_init\_\_()  
 self.model = resnet34(weights=None) *# 如果用预训练，可写 weights='DEFAULT'* self.model.conv1 = nn.Conv2d(1, 64, kernel\_size=7, stride=2, padding=3, bias=False) *# 单通道适配* self.model.fc = nn.Linear(512, 3755) *# 修改输出类别数* def forward(self, x):  
 return self.model(x)  
  
  
class VGG(nn.Module):  
 def \_\_init\_\_(self):  
 super(VGG, self).\_\_init\_\_()  
  
 *# 配置：每个元组是 (卷积层数量, 输出通道数)* conv\_arch = ((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))  
  
 layers = []  
 in\_channels = 1  
 *# 1\*64\*64->64\*32\*32->128\*16\*16->256\*8\*8->512\*4\*4->512\*2\*2  
 # 1\*128\*128->64\*64\*64->128\*32\*32->256\*16\*16->512\*8\*8->512\*4\*4* for (num\_convs, out\_channels) in conv\_arch:  
 layers.append(vgg\_block.vgg\_block(num\_convs, in\_channels, out\_channels))  
 in\_channels = out\_channels  
  
 self.features = nn.Sequential(\*layers) *# 卷积层部分* self.flatten = nn.Flatten()  
  
 self.classifier = nn.Sequential( *# 全连接层部分  
 # nn.Linear(512 \* 2 \* 2, 4096),  
 # nn.ReLU(True),  
 # nn.Dropout(0.5),  
 # nn.Linear(4096, 4096),  
 # nn.ReLU(True),  
 # nn.Dropout(0.5),  
 # nn.Linear(4096, 3755) # 3755分类* nn.Linear(512 \* 2 \* 2, 1024),  
 nn.ReLU(True),  
 nn.Dropout(0.5),  
 nn.Linear(1024, 1024),  
 nn.ReLU(True),  
 nn.Dropout(0.5),  
 nn.Linear(1024, 3755) *# 3755分类* )  
 self.\_initialize\_weights()  
  
 def \_initialize\_weights(self):  
 for m in self.modules():  
 if isinstance(m, nn.Conv2d):  
 nn.init.kaiming\_normal\_(m.weight, mode='fan\_out', nonlinearity='relu')  
 if m.bias is not None:  
 nn.init.constant\_(m.bias, 0)  
 elif isinstance(m, nn.Linear):  
 nn.init.normal\_(m.weight, 0, 0.01)  
 nn.init.constant\_(m.bias, 0)  
  
 def forward(self, x):  
 x = self.features(x)  
 x = self.flatten(x)  
 x = self.classifier(x)  
 return x  
 *# def forward(self, x):  
 # print("Input shape:", x.shape)  
 # for i, layer in enumerate(self.features):  
 # x = layer(x)  
 # print(f"Features[{i}] output shape: {x.shape}")  
 # x = self.flatten(x)  
 # print("Flatten shape:", x.shape)  
 # x = self.classifier(x)  
 # return x*class LeNet(nn.Module): *# 不稳定* def \_\_init\_\_(self):  
 super(LeNet, self).\_\_init\_\_()  
 self.net = nn.Sequential( *# bs\*1\*64\*64* nn.Conv2d(1, 6, kernel\_size=5, padding=2), nn.ReLU(), *# 6\*64\*64（sigmoid->relu）* nn.AvgPool2d(kernel\_size=2, stride=2), *# 6\*32\*32* nn.Conv2d(6, 16, kernel\_size=5), nn.ReLU(), *# 16\*28\*28* nn.AvgPool2d(kernel\_size=2, stride=2), *# 16\*14\*14* nn.Flatten(),  
 *# nn.Linear(16 \* 14 \* 14, 120), nn.ReLU(),#3136->120  
 # nn.Linear(120, 84), nn.ReLU(),#120->84  
 # nn.Linear(84, 3755))#84->3755* nn.Linear(16 \* 14 \* 14, 1024), nn.ReLU(), *# 从 3136 → 1024，减缓降维* nn.Dropout(0.5), *# 不加会出现严重的过拟合（训练到30epoch）acc为0.1左右！* nn.Linear(1024, 512), nn.ReLU(), *# 1024 → 512* nn.Dropout(0.5),  
 nn.Linear(512, 3755))  
  
 def forward(self, x):  
 out = self.net(x)  
 return out

（4）主程序训练根据命令行参数加载汉字图像数据集，选择四种模型（LeNet、VGG、ResNet、CNN）之一进行训练，采用SGD优化器与交叉熵损失，记录训练与测试精度，支持动态学习率、模型断点续训，并可保存训练过程图表。

import torch  
from matplotlib import pyplot as plt  
from torch import nn, optim  
from torch.utils.data import DataLoader  
from torchvision import transforms  
from MyDataset import MyDataset  
from config import args  
import time  
from model import LeNet, VGG, ResNet,CNN  
from test import test as ts  
  
*# 获取训练集*transform = transforms.ToTensor()  
train\_set = MyDataset(args.root + '/HWDB1.1trn\_gnt', transforms=transform,augment=True)  
test\_set = MyDataset(args.root + '/HWDB1.1tst\_gnt', transforms=transform)  
train\_loader = DataLoader(train\_set, batch\_size=args.batch\_size, shuffle=True,num\_workers=8)  
  
*# 定义model 、loss 、optimizer*torch.manual\_seed(args.seed) *# 设置随机数种子，确保结果可重复*device = torch.device('cuda' if args.cuda else 'cpu')  
  
model\_dict = {  
 'LeNet': LeNet,  
 'VGG': VGG,  
 'ResNet': ResNet,  
 'CNN':CNN,  
}  
  
model = model\_dict[args.model]().to(device)  
  
if args.load\_state:  
 model.load\_state\_dict(torch.load(args.result + '/param/model.pth', map\_location=device)) *# 继续训练*model.train()  
loss\_fn = nn.CrossEntropyLoss()  
optimizer = optim.SGD(model.parameters(), lr=args.lr)  
scheduler = optim.lr\_scheduler.MultiStepLR(optimizer, milestones=[10, 20, 50], gamma=0.1)  
  
start\_time = time.time()  
train\_losses, train\_accuracies, test\_accuracies = [], [], []  
*# 开始训练*for epoch in range(1, args.epochs + 1):  
 running\_loss = 0.0  
 running\_acc = 0.0  
  
 for data in train\_loader:  
 image, label = data  
 image, label = image.to(device), label.to(device)  
  
 *# 前向传播* out = model(image)  
 loss = loss\_fn(out, label)  
 running\_loss += loss.item() \* label.size(0)  
 \_, pred = torch.max(out, 1)  
 num\_correct = (pred == label).sum()  
 running\_acc += num\_correct.item()  
  
 *# 反向传播* optimizer.zero\_grad() *# 梯度清零，以免影响其他batch* loss.backward() *# 后向传播，计算梯度* optimizer.step() *# 利用梯度更新W、b的参数  
 # scheduler.step() # 动态调整学习率  
  
 # 每5个epoch保存一遍模型权重* if epoch % 5 == 0:  
 torch.save(model.state\_dict(), args.result + '/param/model.pth')  
  
 test\_acc, f1, recall = ts(model)  
  
 train\_loss = running\_loss / (len(train\_set))  
 train\_acc = running\_acc / (len(train\_set))  
  
 train\_losses.append(train\_loss)  
 train\_accuracies.append(train\_acc)  
 test\_accuracies.append(test\_acc)  
  
 *# 打印一个循环后，训练集上的loss和正确率* print('Train {} epoch, Loss: {:.4f}, Train\_Acc: {:.4f}, Test\_Acc:{:.4f}'  
 .format(epoch, train\_loss, train\_acc, test\_acc))  
  
end\_time = time.time()  
total\_time = end\_time - start\_time  
*# 转换为时分秒*hours = int(total\_time // 3600)  
minutes = int((total\_time % 3600) // 60)  
seconds = int(total\_time % 60)  
print(f"\n训练总耗时：{hours:02d}:{minutes:02d}:{seconds:02d}（时:分:秒）")  
  
*# 画图*epochs = range(1, args.epochs + 1)  
plt.plot(epochs, train\_losses, label='Train Loss')  
plt.plot(epochs, train\_accuracies, label='Train Accuracy')  
plt.plot(epochs, test\_accuracies, label='Test Accuracy')  
plt.xlabel('Epoch')  
plt.ylabel('Value')  
plt.title('Training & Test Metrics')  
plt.legend()  
plt.grid(True)  
plt.savefig(args.result + "/fig/" + args.model)  
plt.close()

（5）对输入图像进行字符分割，采用中值滤波降噪，并融合相近轮廓防止字符断裂或粘连，确保分割字符完整、顺序正确。

import cv2  
import numpy as np  
import os  
from config import args  
  
def charSeperate(src\_img, filter\_size=3):  
 *"""函数功能：字符分割  
 @param src\_img: 输入图像  
 @param filter\_size: 中值滤波核大小  
 @return dst\_img: 分割出的字符图像列表"""  
  
 # 灰度图* gray = cv2.cvtColor(src\_img, cv2.COLOR\_BGR2GRAY)  
  
 *# 二值化* \_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)  
 *#颜色反转（黑变白，白变黑），便于轮廓提取* binary\_inv = cv2.bitwise\_not(binary)  
  
 *# 中值滤波* binary\_f = cv2.medianBlur(binary\_inv, filter\_size)  
  
 *# 查找字符区域* contours, \_ = cv2.findContours(binary\_f, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 *# 遍历所有区域，寻找最大宽度* w\_max = 0  
 for cnt in contours:  
 \_, \_, w, \_ = cv2.boundingRect(cnt)*# （矩形左上角的横坐标，纵坐标，矩形的宽度，高度）* if w > w\_max:  
 w\_max = w  
  
 *#根据轮廓的横向位置将字符区域合并在一起，避免一个字符被分成多个小轮廓  
 # 遍历所有区域，拼接x坐标接近的区域* char\_dict = {}  
 for cnt in contours:  
 x, y, w, h = cv2.boundingRect(cnt)  
 x\_mid = x + w // 2 *# 计算中点位置  
  
 # 如果char\_dict是空的（第一个字符），直接添加。  
 # 对于已有的所有字符区域中点z，是否都与当前中点x\_mid的距离大于半个最大字符宽度（w\_max // 2），说明为新字符。* if not char\_dict or all(np.abs(z - x\_mid) > w\_max // 2 for z in char\_dict):  
 char\_dict[x\_mid] = cnt  
 else:  
 *# 否则，当前轮廓距离某个已有字符区域很近，说明它们可能是同一个字符被切成多个小块  
 #找到接近的字符区域（通过中点坐标比较），用 np.concatenate() 把它们合并成一个更大的轮廓。* for z in list(char\_dict.keys()):  
 if np.abs(z - x\_mid) <= w\_max // 2:  
 char\_dict[z] = np.concatenate((char\_dict[z], cnt), axis=0) *# 拼接两个区域  
  
 # 按照中点坐标，对字符进行排序* char\_dict = dict(sorted(char\_dict.items(), key=lambda item: item[0]))  
  
 *# 遍历所有区域，提取字符* dst\_img = []  
 for \_, cnt in char\_dict.items():  
 x, y, w, h = cv2.boundingRect(cnt)  
 roi = binary[y:y + h, x:x + w]  
 dst\_img.append(roi)  
  
 return dst\_img  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 *# 读取本地图像* input\_path = "./sentence\_img/yyy.png"  
 image = cv2.imread(input\_path)  
  
 if image is None:  
 print(f"无法读取图像: {input\_path}")  
 exit()  
  
 *# 字符分割* characters = charSeperate(image, filter\_size=3)  
  
 *# 输出目录* output\_dir = args.result+"/output\_chars"  
 os.makedirs(output\_dir, exist\_ok=True)  
  
 *# 保存字符图像* for idx, char\_img in enumerate(characters):  
 output\_path = os.path.join(output\_dir, f"char\_{idx+1}.png")  
 cv2.imwrite(output\_path, char\_img)  
 print(f"字符{idx+1}保存至：{output\_path}")

（6）基于训练好的深度学习模型，实现字符图片的自动识别，通过字符编号与字典映射复原文本，批量图像识别并保持字符顺序。

import pickle  
import cv2  
import torch  
import os  
from torch import nn  
from torchvision import transforms  
from model import LeNet, VGG, ResNet  
from config import args  
  
*# 加载字符识别模型*device = torch.device('cuda' if args.cuda == 'true' else 'cpu')  
model\_dict = {  
 'LeNet': LeNet,  
 'VGG': VGG,  
 'ResNet': ResNet  
}  
  
model = model\_dict[args.model]().to(device)  
model.load\_state\_dict(torch.load(args.result + '/param/model.pth', map\_location=device))  
model.eval()  
loss\_fn = nn.CrossEntropyLoss()  
  
*# 加载字符映射字典*with open(args.root + '/char\_dict', 'rb') as f:  
 char\_dict = pickle.load(f)  
 char\_dict = {v: k for k, v in char\_dict.items()} *# 转换为{编号: 字符}的格式*def process\_image(image\_path):  
 *"""处理单张图片并返回预测结果"""  
 # 读取图片并进行预处理* image = cv2.imread(image\_path, 0)  
 if image is None:  
 return None  
  
 *# 二值化和反色处理* \_, image = cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY\_INV)  
  
 *# 提取有效区域* x, y, w, h = cv2.boundingRect(image)  
 image = image[y:y + h, x:x + w]  
  
 *# 调整尺寸并转换为Tensor* image = cv2.resize(image, (args.image\_size, args.image\_size))  
 transform = transforms.ToTensor()  
 image\_tensor = transform(image).unsqueeze(0).to(device)  
  
 *# 预测结果* with torch.no\_grad():  
 output = model(image\_tensor)  
 \_, pred = torch.max(output, 1)  
 return char\_dict[int(pred[0])]  
  
  
def recognize\_chars(folder\_path):  
 *#识别文件夹中图片的所有字符并拼接结果* files = sorted(  
 [f for f in os.listdir(folder\_path) if f.lower().endswith(('.png', '.jpg', '.jpeg'))],  
 key=lambda x: int(x.split('\_')[1].split('.')[0]) *# 按char\_后面的数字排序* )  
  
 *# 逐个处理文件* result = []  
 for filename in files:  
 file\_path = os.path.join(folder\_path, filename)  
 char = process\_image(file\_path)  
 if char:  
 result.append(char)  
 print(f"识别文件 {filename} -> 字符: {char}")  
  
 return ''.join(result)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 *# 配置参数* input\_folder = args.result+"/output\_chars"  
 output\_string = recognize\_chars(input\_folder)  
 print("\n最终识别结果:", output\_string)

1. 作者贡献

本项目全部由本人独立实现。