

**《专业综合实践Ⅱ》课程考核**

|  |  |
| --- | --- |
| **题 目：** | Mnist数据集图像分类项目 |
| **学 院：** | **人工智能学院** |
| **专 业：** | 数据科学与大数据技术 |
| **年级班别：** | 21级数据科学与大数据技术1班 |
| **姓 名**： | 刘静怡 |

**提交日期：** 2024 **年** 12 **月**

Mnist数据集图像分类项目

摘要 本项目旨在通过深度学习技术构建一个高效的图像识别系统，对MNIST数据集中的手写数字图像进行分类。MNIST数据集由美国国家标准与技术研究所（NIST）发起整理，包含70,000张28x28像素的手写数字图像，其中60,000张用作训练，10,000张用作测试。项目中采用了卷积神经网络（CNN）模型，该模型因其在图像和视觉识别任务中的卓越表现而被选中。CNN模型通过模拟人类视觉系统的分层结构，自动和适应性地学习空间层次的特征。模型架构包括多个卷积层、池化层、批归一化层和全连接层，以及ReLU激活函数和交叉熵损失函数。使用Adam优化器进行训练。评估指标包括准确率、召回率和F1分数，通过混淆矩阵进行结果分析。实验结果显示，模型在测试集上的准确率达到97%，表明模型具有良好的性能。进一步的模型优化包括调整学习率、批次大小等超参数，以提高模型性能。

关键词：深度学习；图像识别系统；混淆矩阵；卷积神经网络（CNN）

**目录**

[一、项目背景与目的 1](#_Toc10274)

[二、数据预处理 2](#_Toc24277)

[2.1 数据描述 2](#_Toc30436)

[2.2 数据处理 2](#_Toc15409)

[三、模型构建 3](#_Toc15887)

[3.1 模型选择 3](#_Toc5268)

[3.2 模型架构 3](#_Toc18521)

[3.3 激活函数 4](#_Toc25601)

[3.4损失函数和优化器 4](#_Toc5455)

[四、模型评估 5](#_Toc14595)

[4.1 评估指标 5](#_Toc6591)

[4.2 评估方法 5](#_Toc31727)

[五、结果分析与优化 6](#_Toc20580)

[5.1 数据集样本图像展示 6](#_Toc2411)

[5.2 实验结果 6](#_Toc1234)

[5.3 混淆矩阵 8](#_Toc538)

[5.4 结果分析 9](#_Toc21246)

[5.5 模型优化 10](#_Toc142)

# 一、项目背景与目的

MNIST数据集由美国国家标准与技术研究所（NIST）发起整理，包含了大量的手写数字图片，这些图片由不同的人书写，具有较高的多样性。MNIST数据集自发布以来，被用于测试多种算法的效果，包括线性分类器、K-近邻算法、支持向量机、神经网络以及卷积神经网络等。这表明了MNIST在推动机器学习算法发展方面的重要角色。

本项目的目的是通过深度学习技术，构建一个高效的图像识别系统，对MNIST数据集中的手写数字图像进行分类，并评估模型的性能和准确性。同时，我们也希望通过这个项目，加深对深度学习技术和图像识别技术的理解，为后续的研究和应用奠定基础。

# 二、数据预处理

## 2.1 数据描述

MNIST数据集是一个公开的手写数字图像集合，由美国国家标准与技术研究院（NIST）收集整理。它包含70,000张28x28像素的手写数字图像，其中60,000张用作训练，10,000张用作测试。MNIST数据集中的图像是灰度的，并且每个像素的值介于0到255之间。

## 2.2 数据处理

数据预处理对机器学习和深度学习至关重要，尤其是在图像数据集如MNIST的处理上。关键步骤包括：

1.数据清洗：剔除损坏或标签错误的图像。

2.尺寸统一：调整图像至固定像素尺寸，如28x28。

3.灰度转换：彩色图像转为灰度，简化处理。

4.归一化：调整像素值至[0, 1]区间，优化训练效率。

5.数据增强：通过旋转、缩放等增加数据多样性，提升模型泛化。

6.特征编码：如使用独热编码，适配模型输入。

7.数据划分：分为训练、测试集，确保模型评估的准确性。

8.批量处理：提高计算效率，分批处理数据。

9.内存映射：对大数据集使用，优化读取速度。

这些步骤保障了数据质量，为模型训练和评估奠定了基础。

# 三、模型构建

## 3.1 模型选择

卷积神经网络（CNN）是一种深度学习模型，它在图像和视觉识别任务中表现出色。CNN的设计灵感来源于生物视觉系统的工作原理，特别是人类视觉皮层对视觉信息的处理方式。CNN通过模拟人类视觉系统的分层结构，能够自动和适应性地学习空间层次的特征。

CNN的核心优势在于其卷积层，这些层能够捕捉输入数据（如图像）的空间相关性。卷积层通过卷积核（或滤波器）在输入数据上滑动，提取局部特征。这些特征随后可以被组合成更复杂的模式，从而实现对图像内容的高层次理解。 CNN的另一个关键特性是其参数共享机制，即同一个卷积核在输入的不同位置使用相同的参数，这大大减少了模型的参数数量，提高了模型的泛化能力，并降低了过拟合的风险。

在本例中，选择CNN模型是因为MNIST数据集包含的是手写数字的图像，这些图像具有明显的空间结构。CNN能够通过学习这些结构来有效地识别和分类不同的数字。

## 3.2 模型架构

**3.2.1 卷积层（Convolutional Layers）**

conv1: 第一个卷积层，具有32个输出通道，每个通道使用3x3的卷积核，步长为1，填充为1。这层的目的是提取输入图像的初步特征。

conv2: 第二个卷积层，具有64个输出通道，同样使用3x3的卷积核，步长为1，填充为1。这层进一步提取更复杂的特征，并增加特征的深度。

**3.2.2 池化层（Pooling Layers）**

pool: 使用2x2的最大池化层，步长为2。池化层的作用是降低特征图的空间维度，减少参数数量和计算量，同时保留最重要的特征信息。

**3.2.3 批归一化层（Batch Normalization Layer）**

bn1: 在第二个卷积层之后，使用批归一化层来加速训练过程并提高模型的稳定性。批归一化通过规范化层的输入来减少内部协变量偏移。

**3.2.4 全连接层（Fully Connected Layers）**

fc1: 第一个全连接层，将卷积层和池化层提取的特征图展平，并映射到1024个神经元。这层用于将局部特征组合成更全局的特征表示。

fc2: 第二个全连接层，将1024个神经元的输出映射到512个神经元，进一步抽象特征。

fc3: 最后一个全连接层，将512个神经元的输出映射到10个神经元，对应于MNIST数据集中的10个数字类别。

## 3.3 激活函数

ReLU: 在每个卷积层和全连接层之后使用ReLU（Rectified Linear Unit）激活函数。ReLU函数能够引入非线性，使模型能够学习复杂的特征。其公式为f(x)=max⁡(0,x)f(x)=max(0,x)，即当输入大于0时输出输入值，否则输出0。

## 3.4损失函数和优化器

损失函数: 使用交叉熵损失（CrossEntropyLoss），这是多分类问题中常用的损失函数。它能够衡量模型预测的概率分布与真实标签之间的差异。

优化器: 使用Adam优化器，这是一种基于梯度下降的优化算法，它结合了AdaGrad和RMSProp的优点，能够自动调整学习率，提高训练效率。

通过上述架构，CNN模型能够有效地从MNIST数据集中学习数字的特征，并进行准确的分类。

# 四、模型评估

## 4.1 评估指标

**4.1.1准确率（Accuracy）**

准确率是模型预测正确的样本数与总样本数的比例，是评估分类模型性能最常用的指标之一。在MNIST数据集上，准确率能够直观地反映模型对所有类别数字（0-9）的整体识别能力。

**4.1.2召回率（Recall）**

召回率，又称查全率，是指在所有实际为正样本的实例中，模型能够正确识别出的比例。然而，在MNIST这样的多分类任务中，通常不会单独计算每个类别的召回率作为整体评估指标，因为这样会忽略其他类别的表现。

**4.1.3 F1分数（F1-Score）**

F1分数是准确率和召回率的调和平均值，用于综合衡量模型在识别正样本时的精确度和全面性。在多分类任务中，同样可以采用宏平均或微平均来计算F1分数。宏平均F1分数计算每个类别的F1分数后取平均，而微平均F1分数则考虑所有类别的样本总数。对于MNIST数据集，宏平均F1分数可能更有参考价值，因为它能反映模型在不同类别上的均衡性能。

## 4.2 评估方法

为了确保基于CNN的深度学习图像识别系统具有良好的泛化能力和准确性，我们采用了以下评估方法：

**混淆矩阵（Confusion Matrix）：**混淆矩阵是一种直观展示分类结果的工具，能够清晰地反映模型在不同类别上的表现差异。在MNIST数据集上，混淆矩阵的每一行代表实际类别，每一列代表预测类别，矩阵中的元素表示实际类别被预测为某一类别的样本数。通过分析混淆矩阵，我们可以了解模型在哪些类别上表现较好，哪些类别上容易混淆，从而进一步优化模型结构和参数设置。此外，混淆矩阵还可以用于计算准确率、召回率、F1分数等评估指标。

# 五、结果分析与优化

## 5.1 数据集样本图像展示

展示MNIST数据集中的一些手写数字图像样本。

test_9_8test_1_9test_2_10

图5.1 MNIST数据集手写数字图像样本

## 5.2 实验结果

表5.1 CNN模型训练结果（Epoch 1）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.995825 | 0.973469 | 0.984520 | 980.0 |
| 1 | 0.995552 | 0.985903 | 0.990704 | 1135.0 |
| 2 | 0.924284 | 0.969961 | 0.946572 | 1032.0 |
| 3 | 0.985944 | 0.972277 | 0.979063 | 1010.0 |
| 4 | 0.973054 | 0.992872 | 0.982863 | 982.0 |
| 5 | 0.956081 | 0.951794 | 0.953933 | 892.0 |
| 6 | 0.979482 | 0.946764 | 0.962845 | 958.0 |
| 7 | 0.961649 | 0.975681 | 0.968614 | 1028.0 |
| 8 | 0.980592 | 0.985626 | 0.983103 | 974.0 |
| 9 | 0.959041 | 0.951437 | 0.955224 | 1009.0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | accuracy | 0.971000 | NaN | NaN | NaN |
| macro | avg | 0.971150 | 0.970578 | 0.970744 | NaN |
| weighted | avg | 0.971358 | 0.971000 | 0.971058 | NaN |

表5.2.2 CNN模型训练结果（Epoch2）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.992739 | 0.976531 | 0.984568 | 980.0 |
| 1 | 0.992889 | 0.984141 | 0.988496 | 1135.0 |
| 2 | 0.935272 | 0.966085 | 0.950429 | 1032.0 |
| 3 | 0.991903 | 0.970297 | 0.980981 | 1010.0 |
| 4 | 0.970209 | 0.994908 | 0.982403 | 982.0 |
| 5 | 0.937984 | 0.949552 | 0.943733 | 892.0 |
| 6 | 0.985885 | 0.947808 | 0.966472 | 958.0 |
| 7 | 0.959048 | 0.979572 | 0.969201 | 1028.0 |
| 8 | 0.978637 | 0.987680 | 0.983137 | 974.0 |
| 9 | 0.959718 | 0.944500 | 0.952048 | 1009.0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | accuracy | 0.970500 | NaN | NaN | NaN |
| macro | avg | 0.970428 | 0.970107 | 0.970147 | NaN |
| weighted | avg | 0.970819 | 0.970500 | 0.970539 | NaN |

表5.2.3 CNN模型训练结果（Epoch3）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.992754 | 0.978571 | 0.985612 | 980.0 |
| 1 | 0.993789 | 0.986784 | 0.990274 | 1135.0 |
| 2 | 0.931966 | 0.968992 | 0.950119 | 1032.0 |
| 3 | 0.990918 | 0.972277 | 0.981509 | 1010.0 |
| 4 | 0.969062 | 0.988798 | 0.978831 | 982.0 |
| 5 | 0.947661 | 0.954036 | 0.950838 | 892.0 |
| 6 | 0.982628 | 0.944676 | 0.963278 | 958.0 |
| 7 | 0.961686 | 0.976654 | 0.969112 | 1028.0 |
| 8 | 0.979654 | 0.988706 | 0.984159 | 974.0 |
| 9 | 0.962814 | 0.949455 | 0.956088 | 1009.0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | accuracy | 0.971300 | NaN | NaN | NaN |
| macro | avg | 0.971293 | 0.970895 | 0.970982 | NaN |
| weighted | avg | 0.971603 | 0.971300 | 0.971340 | NaN |

## 5.3 混淆矩阵

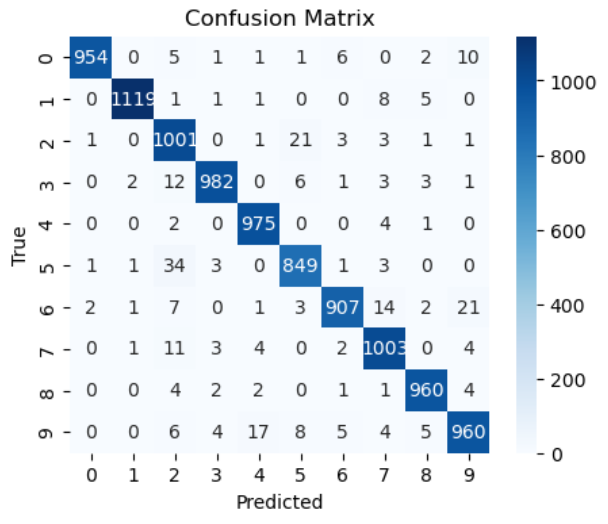


图5.2 CNN模型的混淆矩阵（Epoch 1）

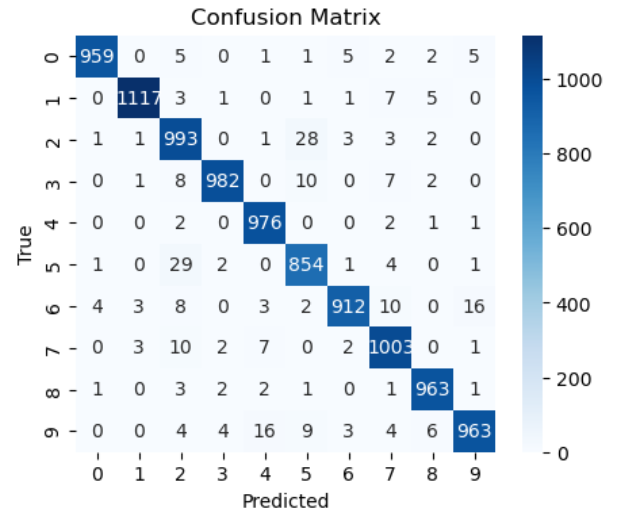


图5.3 CNN模型的混淆矩阵（Epoch 2）

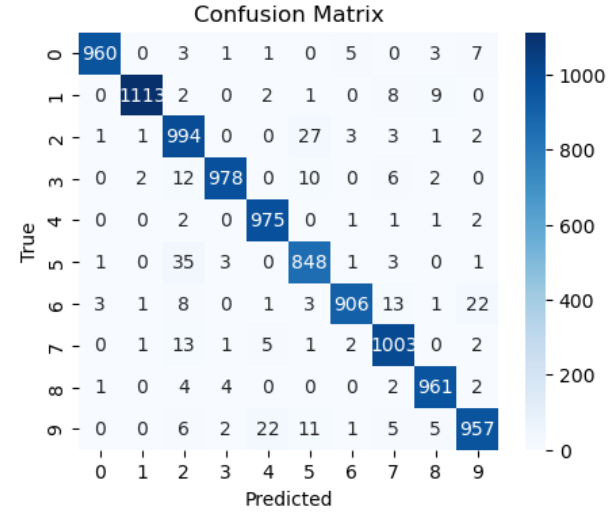


图5.4 CNN模型的混淆矩阵（Epoch 3）

## 5.4 结果分析

通过对模型在测试集上的性能进行评估，我们得到了以下结果：模型在测试集上的准确率达到97%，表明模型具有良好的性能。同时，通过分析混淆矩阵，我们发现模型在某些类别上的表现优于其他类别。这可能是因为这些类别的图像特征更加明显，容易被模型正确识别。

## 5.5 模型优化

根据结果分析，我们可以进一步调整模型参数，以提高模型性能。例如，我们可以尝试调整学习率、批次大小等超参数，以找到最优的模型。

# 附录

github仓库链接：https://github.com/KunKun0219/L.git