实验名称：**MNIST 手写数字识别：传统机器学习与深度学习模型性能对比实验报告**

完 成 人：苏 栋

摘要

本实验基于 MNIST 手写数字数据集，采用传统机器学习算法（逻辑回归、KNN、SVM、随机森林）与深度学习模型（CNN、ResNet 及其优化版本）开展手写数字分类研究。解析 IDX 格式数据，经归一化与独热编码预处理后，训练各模型并对比分析准确率、精确率等指标。结果表明：传统模型中 SVM 表现最优（准确率 0.9566），深度学习模型优势显著，CNN+（epoch 加深 10 倍）与 ResNet+（epoch 为基础模型 4 倍）在测试集准确率分别达 0.9927 和 0.9915，且在手写数据集 1 上均实现 100%预测准确率。研究发现，残差连接（ResNet）可加速深度网络收敛，但过度增加 epoch 可能引发过拟合；深度学习模型对复杂手写体特征提取能力显著优于传统模型。本实验为图像分类任务的模型选择与优化提供了实践参考，未来可探索数据增强与轻量化架构进一步提升泛化能力。

关键词：MNIST 数据集、机器学习、深度学习

目录

一、实验要求与实现

本次实验基于 MNIST 手写数字数据集，利用监督学习算法完成模型训练、预测及手写数字识别。

1. https://yann.lecun.org/exdb/mnist/index.html 或 https://github.com/cvdfoundation/mnist
2. 在该网站下载4组数据(2组训练，2组测试)
3. 使用matlab提取数据(图像及标签：0-9)
4. 选择一个合适监督学习算法，对以上训练数据进行训练，得到分类模型。
5. 使用上述模型对测试数据进行预测，得到预测准确率。
6. 自己手写一个数字，代入所得模型进行预测。
7. 将整个过程，用word编辑为一个完整实验报告(模板请自行查找)
8. 6.15前提交给学习委员。

注：以上过程均建议使用matlab，可选任意所学或未学过算法。

图 1 实验要求

二、数据集说明与探索

（一）数据简介

MNIST 是一个入门级别的计算机视觉数据库，也是机器学习领域最有名的数据集之一。当我们开始学习编程的时候，第一件事往往就是打“Hello world”。而在机器学习中，识别 MNIST 就相当于编程中的“Hello world”。

MNIST 中包含了手写数字 0~9 的图片以及他们对应的标签。下载后包含四个压缩包：

- train-images-idx3-ubyte.gz
- train-labels-idx1-ubyte.gz
- t10k-images-idx3-ubyte.gz
- t10k-labels-idx1-ubyte.gz

解压后得到四个文件：

- 训练集图像：t10k-images.idx3-ubyte
- 训练集标签：t10k-labels.idx1-ubyte
- 测试集图像：train-images.idx3-ubyte
- 测试集标签：train-labels.idx1-ubyte

训练集中有 60,000 个样本，测试集中有 5,000 个样本。所有图像都被标准化为 28*28 像素，像素值在 0~255 之间，0 表示背景，255 表示前景。

表 1 文件格式说明表

图片文件格式说明：

[字节位置]	[类型]	[值]	[描述]
0000	32 位整型	2051	幻数
0004	32 位整型	60000	图片数
0008	32 位整型	28	行数
0012	32 位整型	28	列数
0016	无符号字节	??	像素
0017	无符号字节	??	像素
.....			
xxxx	无符号字节	??	像素

标签文件格式说明：

[字节位置]	[类型]	[值]	[描述]
0000	32 位整型	2049	幻数
0004	32 位整型	60000	标签数
0008	无符号字节	??	标签
0009	无符号字节	??	标签
.....			
xxxx	无符号字节	??	标签

注：这里的整形指的都是无符号整型

上述的 32 位整形遵循“MSB first”，即高位字节在左边，如十进制 8，二进制储存形式为 1000。

幻数是一个固定值，它占据文件的前 4 个字节，实际上表示的是这个文件储存的是图片还是标签，没有具体用处，我们可以忽略它。

图片数与标签数占据文件 4~7 个字节的位置，在训练集中，它为 60,000，表示这个文件有 60,000 个图片或标签，在测试集中，它为 5,000。

行数和列数描述的是每张图片的大小，它们也是固定值，都为 28。每张图片有 28*28=784 个像素，所以从图片文件第 16 个字节位置开始，每隔 784 个字节为一张新图片，其中每个像素的像素值为 0~255。

从标签文件的第 8 个字节位置开始，每个字节都对应着一张图片的数字，标签的值为 0~9。

（二）数据探索

根据以上数据基本信息，对数据进行初步探索。通过频数统计发现，训练集中 10 个类别（0-9）的样本分布呈现高度均衡特征，如下图。

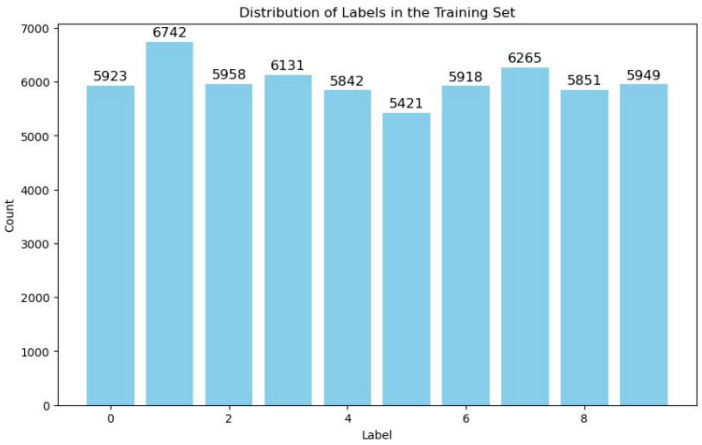

图 2 标签分布图

为了确认标签与图片的对应关系，可以看到标签与图片是一一对应的，以下是前十的训练样本，如下图。

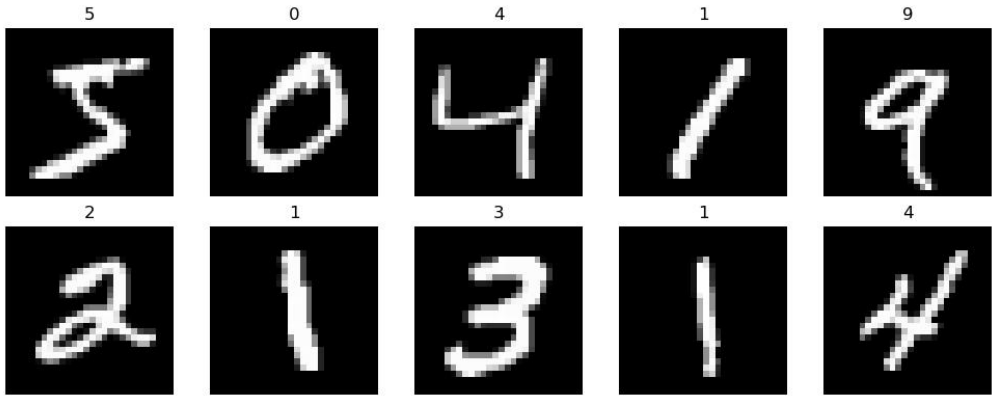

图 3 前 10 个样本

三、模型选择与构建

研究使用多种传统机器学习模型（逻辑回归、KNN、支持向量机、随机森林）与深度学习模型（CNN、ResNet）进行预测，已达到对比与寻找更优的解决方案的目的。

（一）逻辑回归

对于一个机器学习方法，通常由模型、策略和算法 3 个要素构成。模型是假设空间的形式，如是线性函数还是条件概率；策略是判断模型好坏的依据，寻找能够表示模型好坏的数学表达式，将学习问题转化为一个优化问题。一般策略对应着一个代价函数（Cost Function）；算法是上述优化问题的求解方法，有多种形式，如梯度下降法、直接求导、遗传算法等。对于逻辑回归也一样。首先，它依然是基于线性模型的，但是为了解决分类问题，需要把线性模型的输出做一个变换，这就用到了 Sigmoid 函数，它能够把实数域的输出映射到（0，1）区间。这就为输出提供了很好的概率解释。但是从本质上来说，逻辑回归还是一种广义的线性模型。对于策略来说，经过推导可以知道，它采用了交叉熵损失函数。最后为了最小化损失函数，逻辑回归采用了梯度下降方法。综合这 3 个因素，就构成了逻辑回归算法。

1．Sigmoid 函数

Sigmoid 函数的表达式如下：

$$g(z)=\frac{1}{1+e^{-z}}$$

2．梯度下降方法

梯度下降法（Gradient Descent）是一种一阶最优化算法，广泛应用于机器学习和深度学习中，旨在通过迭代过程找到目标函数（通常是损失函数或成本函数）的局部最小值。这种方法的核心思想是沿着函数梯度的负方向逐步调整参数，因为函数的梯度指向了函数值增长最快的方向，那么它的相反方向——梯度的负方向，则是函数值减少最快的方向。因此，通过连续的小步长（学习率）沿着负梯度方向移动，最终可以到达函数的一个局部最小点，从而实现模型参数的优化。

梯度下降得到的结果可能是局部最优值，如果 F(x)是凸函数，则可以保证梯度下降得到的结果是全局最优值。

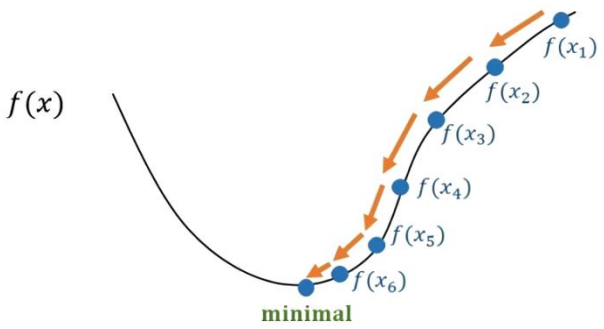

图 4 梯度下降示意图

（二）KNN

KNN 的全称是 K Nearest Neighbors，意思是 K 个最近的邻居，从这个名字我们就能看出一些 KNN 算法的蛛丝马迹了。K 个最近邻居，毫无疑问，K 的取值肯定是至关重要的。KNN 的原理就是当预测一个新的值 x 的时候，根据它距离最近的 K 个点是什么类别来判断 x 属于哪个类别。

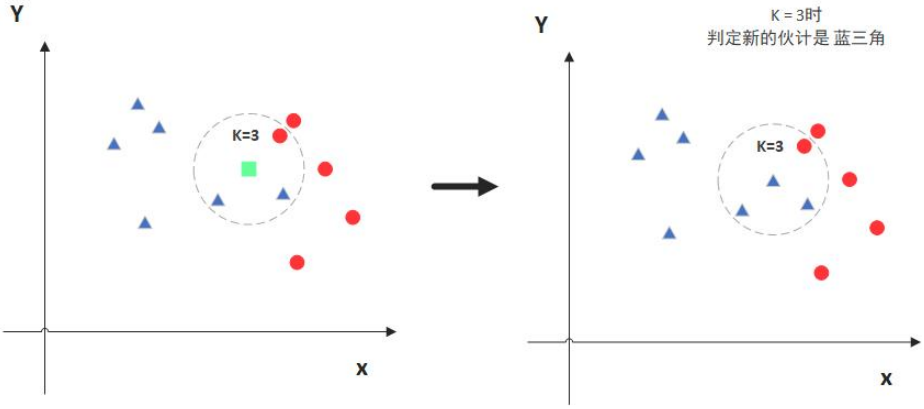

图 5 K=3 示意图

图中绿色的点就是我们要预测的那个点，假设 K=3。那么 KNN 算法就会找到与它距离最近的三个点（这里用圆圈把它圈起来了），看看哪种类别多一些，比如这个例子中是蓝色三角形多一些，新来的绿色点就归类到蓝三角了。

1．距离计算

要度量空间中点距离的话，有好几种度量方式，比如常见的曼哈顿距离计算，欧式距离计算等等。不过通常 KNN 算法中使用的是欧式距离，拓展到多维空间，则公式变成这样：

$$d(x,y)=\sqrt{(x_1-y_1)^2+\ldots+(x_n-y_n)^2}=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

这样我们就明白了如何计算距离，KNN 算法最简单粗暴的就是将预测点与所有点距离进行计算，然后保存并排序，选出前面 K 个值看看哪些类别比较多。但其实也可以通过一些数据结构来辅助，比如最大堆，这里就不多做介绍。

2．K 值选择

通过上面那张图我们知道 K 的取值比较重要，通过交叉验证（将样本数据按照一定比例，拆分出训练用的数据和验证用的数据，比如 6：4 拆分出部分训练数据和验证数据），从选取一个较小的 K 值开始，不断增加 K 的值，然后计算验证集合的方差，最终找到一个比较合适的 K 值。

通过交叉验证计算方差后你大致会得到下面这样的图：

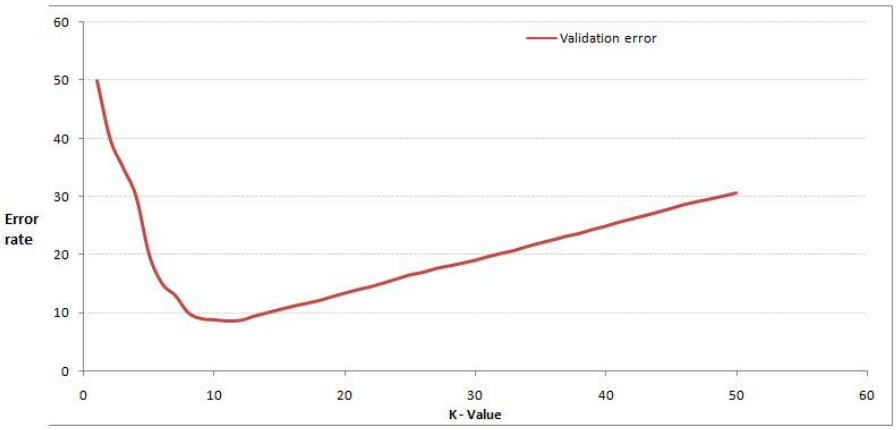

图 6 交叉验证示例图

当你增大 k 的时候，一般错误率会先降低，因为有周围更多的样本可以借鉴了，分类效果会变好。但注意，和 K-means 不一样，当 K 值更大的时候，错误率会更高，比如说你一共就 35 个样本，当你 K 增大到 30 的时候，KNN 基本上就没意义了。

所以选择 K 点的时候可以选择一个较大的临界 K 点，当它继续增大或减小的时候，错误率都会上升，比如图中的 K=10。具体如何得出 K 最佳值的代码，下一节的代码实例中会介绍。

（三）支持向量机

SVM 的全称是 Support Vector Machine，即支持向量机，主要用于解决模式识别领域中的数据分类问题，属于有监督学习算法的一种。SVM 要解决的问题可以用一个经典的二分类问题加以描述。

如下图所示，红色和蓝色的二维数据点显然是可以被一条直线分开的，在模式识别领域称为线性可分问题。然而将两类数据点分开的直线显然不止一条。图(b)和(c)分别给出了 A、B 两种不同的分类方案，其中黑色实线为分界线，术语称为“决策面”。每个决策面对应了一个线性分类器。虽然在目前的数据上看，这两个分类器的分类结果是一样的，但如果考虑潜在的其他数据，则两者的分类性能是有差别的。

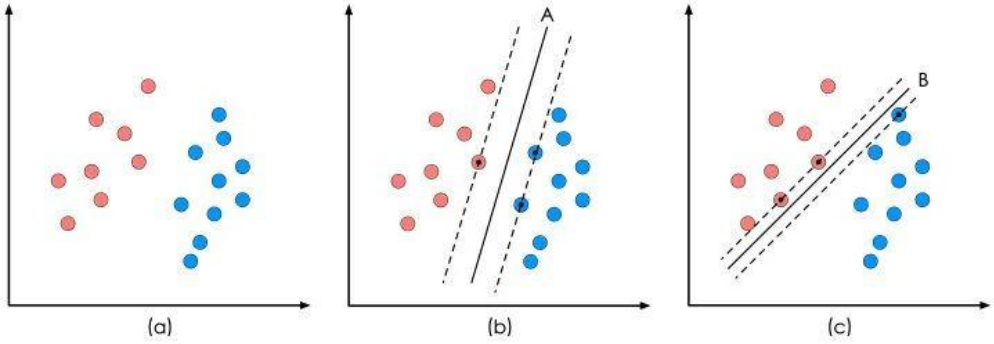

图 7 二分类问题描述

SVM 算法认为图中的分类器 A 在性能上优于分类器 B，其依据是 A 的分类间隔比 B 要大。这里涉及到第一个 SVM 独有的概念“分类间隔”。在保证决策面方向不变且不会出现错分样本的情况下移动决策面，会在原来的决策面两侧找到两个极限位置（越过该位置就会产生错分现象），如虚线所示。虚线的位置由决策面的方向和距离原决策面最近的几个样本的位置决定。

而这两条平行虚线正中间的分界线就是在保持当前决策面方向不变的前提下的最优决策面。两条虚线之间的垂直距离就是这个最优决策面对应的分类间隔。显然每一个可能把数据集正确分开的方向都有一个最优决策面（有些方向无论如何移动决策面的位置也不可能将两类样本完全分开），而不同方向的最优决策面的分类间隔通常是不同的，那个具有“最大间隔”的决策面就是 SVM 要寻找的最优解。

而这个真正的最优解对应的两侧虚线所穿过的样本点，就是 SVM 中的支持样本点，称为“支持向量”。对于图中的数据，A 决策面就是 SVM 寻找的最优解，而相应的三个位于虚线上的样本点在坐标系中对应的向量就叫做支持向量。

从表面上看，我们优化的对象似乎是这个决策面的方向和位置。但实际上最优决策面的方向和位置完全取决于选择哪些样本作为支持向量。而在经过漫长的公式推导后，你最终会发现，其实与线性决策面的方向和位置直接相关的参数都会被约减掉，最终结果只取决于样本点的选择结果。

到这里，我们明确了 SVM 算法要解决的是一个最优分类器的设计问题。既然叫作最优分类器，其本质必然是个最优化问题。

（四）随机森林

随即森林模型是基于决策树算法的改进，是一种集成学习方法（Bagging），主要用于分类和回归任务。它由多个决策树组成，通过集成这些决策树的预测结果来提高模型的准确性和稳定性。

如下图所示，随机森林模型会在原始数据集中随机抽样，构成 n 个不同的样本数据集，然后根据这些数据集搭建 n 个不同的决策树模型，最后根据这些决策树模型的平均值（针对回归模型）或者投票（针对分类模型）情况来获取最终结果。

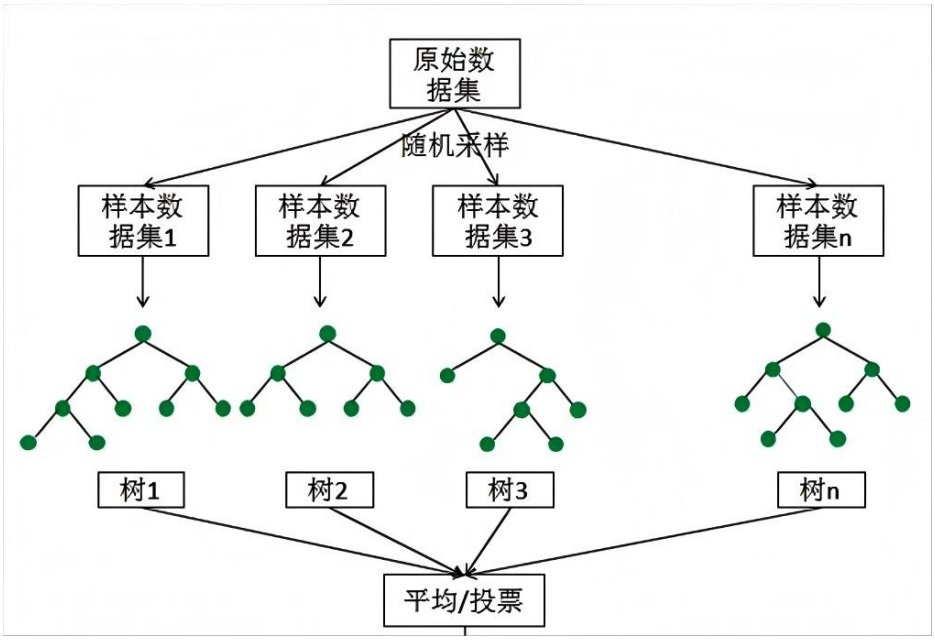

图 8 决策树模型图

随机森林在 bagging 的基础上，再次对特征做了一次随机选择，比如对于自助采样后的每一个子数据集（总共 m 个子数据集），我们并不会像决策数那样用到所有的特征，随机森林会从所有的特征中随机选择一个包含 k（k<n）个特征的子集。当有一条新数据进来，在随机森林的 m 棵树会各自给出一个答案，如果是分类任务，我们就选择投票法，如果是回归任务则一般选择平均值作为输出。不像决策树，越靠近根节点的特征重要性越高，在随机森林中，在每个特征都是有可能成为“主角”的，也不容易出现过拟合的问题。

（五）CNN

CNN 模型通常包括这几层：输入层（input layer）、卷积层（convolutional layer）、池化层（pooling layer）以及输出层（全连接层+softmax layer）。CNN 的模型结构图如下所示。

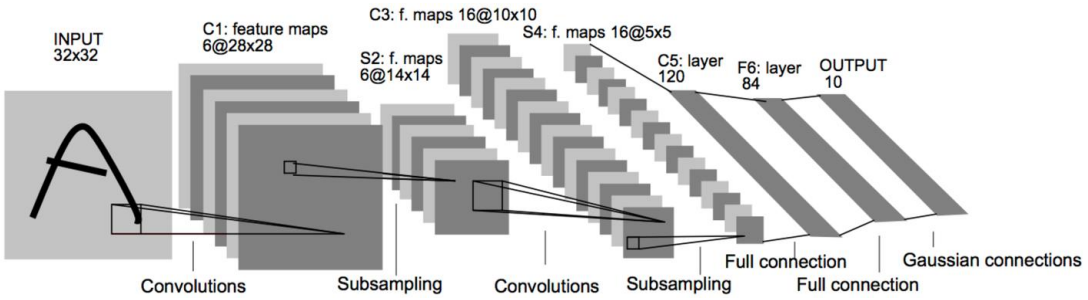

图 9 CNN 模型架构图

卷积是一种局部操作，通过一定大小的卷积核作用于局部图像区域获取图像的局部信息。图像中不同数据窗口的数据和卷积核做 inner product（内积）的操作叫做卷积，其本质是提纯，即提取图像不同频段的特征。卷积层是卷积神经网络的核心基石。在图像识别里我们提到的卷积是二维卷积，即卷积核与二维图像做卷积操作，简单讲是卷积核滑动到二维图像上所有位置，并在每个位置上与该像素点及其领域像素点做内积。不同卷积核可以提取不同的特征，在深层卷积神经网络中，通过卷积操作可以提取出图像低级到复杂的特征。要注意的是，层与层之间会有若干个卷积核，同时，卷积核深度与初始图片的通道数一致。

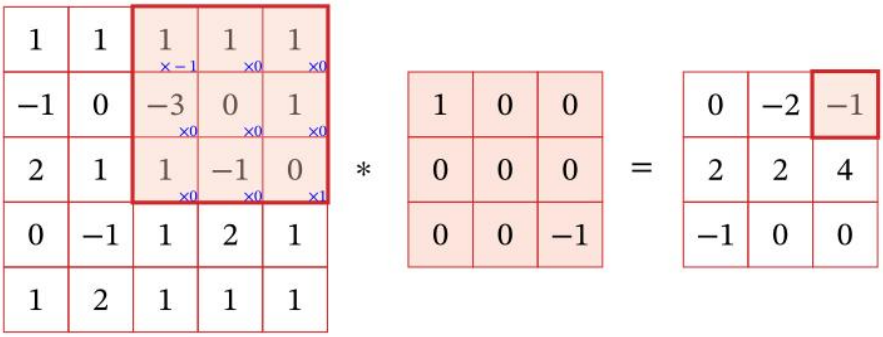

图 10 二维卷积示例图

（六）ResNet

ResNet（Residual Network，残差网络）是深度学习领域中非常重要且具有影响力的一种卷积神经网络（CNN）架构，由何恺明等人于 2015 年提出，在图像识别、目标检测等诸多计算机视觉任务中取得了巨大成功。

ResNet 的核心创新是残差块（Residual Block）。一个标准的残差块包含两个或更多的卷积层，以及一个从输入直接连接到块输出的“捷径连接”（Shortcut Connection）。这个捷径连接允许输入信号直接传递到块的输出端，与经过卷积层处理后的信号相加。这样，残差块的目标就变成了学习一个残差函数，即输入到输出之间的差异，而不是整个映射函数。

在 ResNet 中，残差块是构建整个网络的基础单元。残差块主要由两部分构成：一个或多个卷积层和一个捷径连接（直连边，Shortcut Connection）。

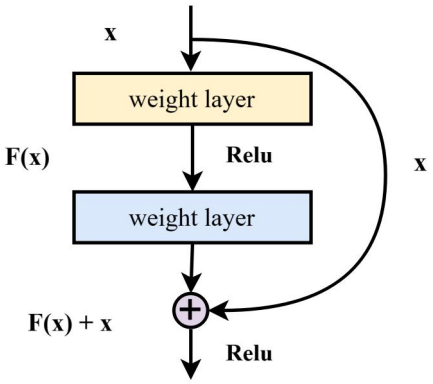

图 11 ResNet 网络架构

在标准残差块中，输入 x 通过一系列卷积层（通常是两个 3x3 的卷积层）进行特征变换，得到输出 F(x)。与此同时，输入 x 直接通过直连边传递，然后与 F(x)相加得到最终的输出 y。即：

$$y = F(x, W_i) + x$$

其中，W_i是残差块中卷积层的权重参数。这种结构只有在输入和输出的维度相同的情况下才能成立，因为只有维度相同，输入和输出才能直接相加。

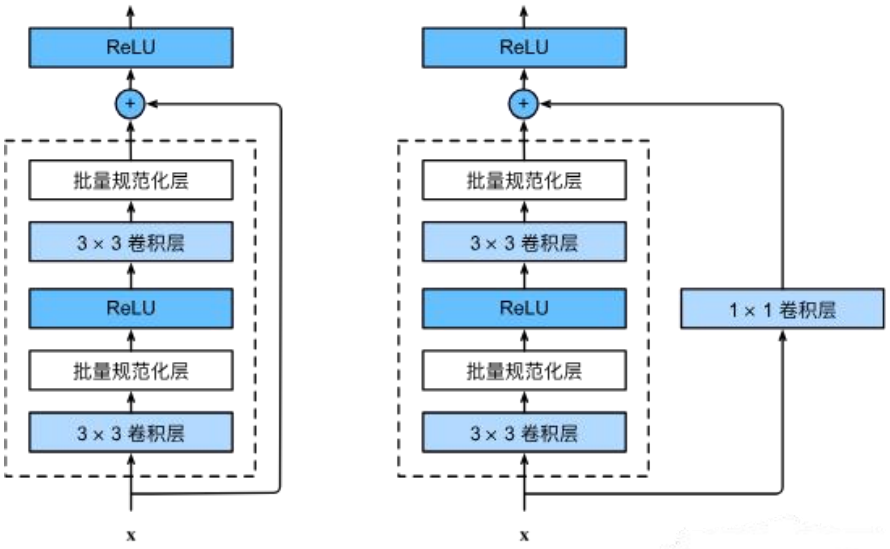

图 12 ResNet 网络架构（左为标准残差块，右为降采样残差块）

在网络的某些层，我们可能需要减小特征图的尺寸或者改变通道数，这时输入和输出的维度就不一致了。为了解决这个问题，ResNet 引入了降采样残差块，它在直连边上增加了一个 1x1 的卷积层，用于调整输入的维度，使之与输出维度匹配，以便进行相加操作。降采样残差块的公式可以表示为：

$$y = F(x, W_i) + W_s(x)$$

其中，$W_s(x)$表示 1x1 卷积层的权重参数，它负责对输入 x 进行降维或升维，以

匹配 $F(x, W_i)$的维度。

ResNet 的设计使得整个网络可以进行端到端的反向传播训练。通过残差块的这种结构，即使在网络非常深的情况下，反向传播算法也可以有效地更新所有层的权重，因为每一步的梯度计算都不会因为深度的增加而显著衰减。

四、模型结果

（一）模型指标

本次实验对多种机器学习和深度学习模型在 MNIST 数据集上的性能进行了评估，主要通过准确率、精确率、召回率和 F1 值四个指标衡量模型效果，各模型具体表现如下表所示。

表 2 模型结果表

模型名称	准确率	精确率	召回率	F1 值
knn	0.9425	0.9442	0.9416	0.9422
logistic	0.8972	0.8959	0.8957	0.8956
random	0.9456	0.9453	0.9452	0.9452
svm	0.9566	0.9563	0.9563	0.9562
CNN	0.9899	0.9899	0.9899	0.9899
CNN+	0.9927	0.9927	0.9926	0.9926
ResNet	0.9873	0.9874	0.9872	0.9872
ResNet+	0.9915	0.9915	0.9914	0.9914
ResNet++	0.9913	0.9912	0.9913	0.9912

从结果来看，传统机器学习模型中，支持向量机（SVM）表现最佳，其准确率达到 0.9566，在精确率、召回率和 F1 值上也保持在较高水平，说明 SVM 在处理 MNIST 数据时能有效区分不同数字类别，误判和漏判情况较少。K 近邻（KNN）和随机森林（random）模型准确率也突破 0.94，性能较为稳定；而逻辑回归（logistic）模型准确率仅为 0.8972，相对其他模型表现较弱，可能是由于其线性分类特性难以捕捉 MNIST 数据复杂的非线性关系。

深度学习模型展现出显著优势，其中基础卷积神经网络（CNN）准确率已高达 0.9899，四个指标均接近 1，表明该模型对 MNIST 数据的特征提取和分类能力极强。在此基础上优化得到的 CNN+模型，准确率进一步提升至 0.9927，说明通过调整网络结构或训练策略，能够有效增强模型性能。基于残差网络的系列模型中，ResNet、ResNet+和 ResNet++准确率均超过 0.98，尤其是 ResNet+模型，以 0.9915 的准确率证明了残差连接在解决深度网络训练难题、提升分类精度上

的有效性。

对比两类模型，深度学习模型在准确率、精确率、召回率和 F1 值上均优于传统机器学习模型，更适合处理 MNIST 这类图像数据。同时，在同一模型架构下进行改进和优化，能够进一步提升模型性能，这为后续模型优化提供了重要参考。

（二）手写数字预测

1．手写数据集构建

本研究邀请了三个同学，三个人利用平板电脑进行书写，得到了从 0-9 的手写数字，如下图。

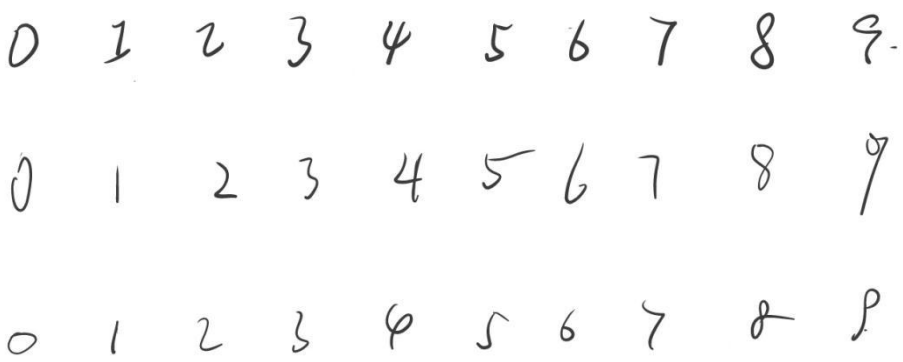

图 13 手写数字数据集

分别对数据集进行划分，得到单一数字，划分成三个小数据集方便后续测试。

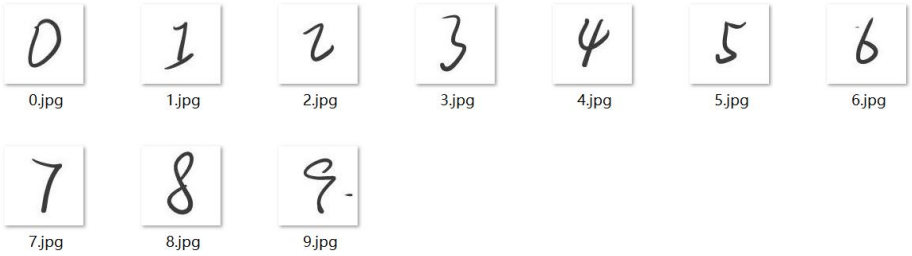

图 14 手写数字数据集 1

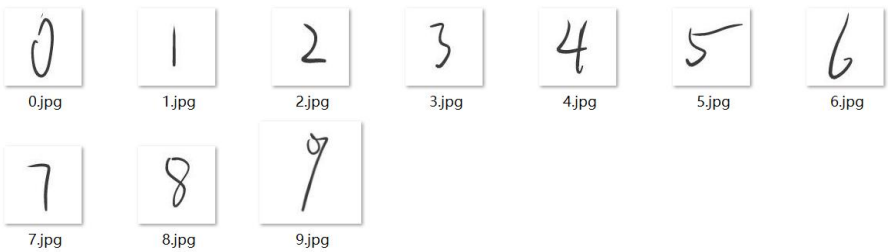

图 15 手写数字数据集 2

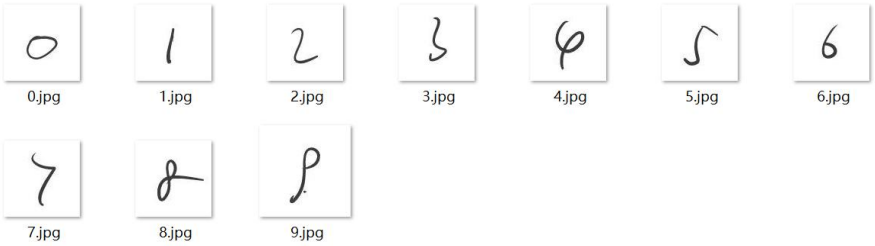

图 16 手写数字数据集 3

2．传统机器学习模型预测结果

下图为手写数据集 1 在各个传统机器学习模型的预测结果，我们可以发现，对于数字 0，knn 与逻辑回归预测有误，分别将其预测为 9 和 7，剩下的随机森林与支持向量机预测正确；对于数字 1、数字 6、数字 7，四个模型均预测准确；对于数字 2，四个模型均预测不正确，knn、逻辑回归、随机森林、支持向量机分别将其预测为 1、5、6、1；对于数字 3，只有随机森林预测准确；对于数字 4，四个模型均预测不正确，knn、逻辑回归、随机森林、支持向量机分别将其预测为 1、7、8、9，对于数字 5、数字 8，只有随机森林与支持向量机预测正确；对于数字 9，四个模型均预测不正确，knn、逻辑回归、随机森林、支持向量机分别将其预测为 1、7、8、7。

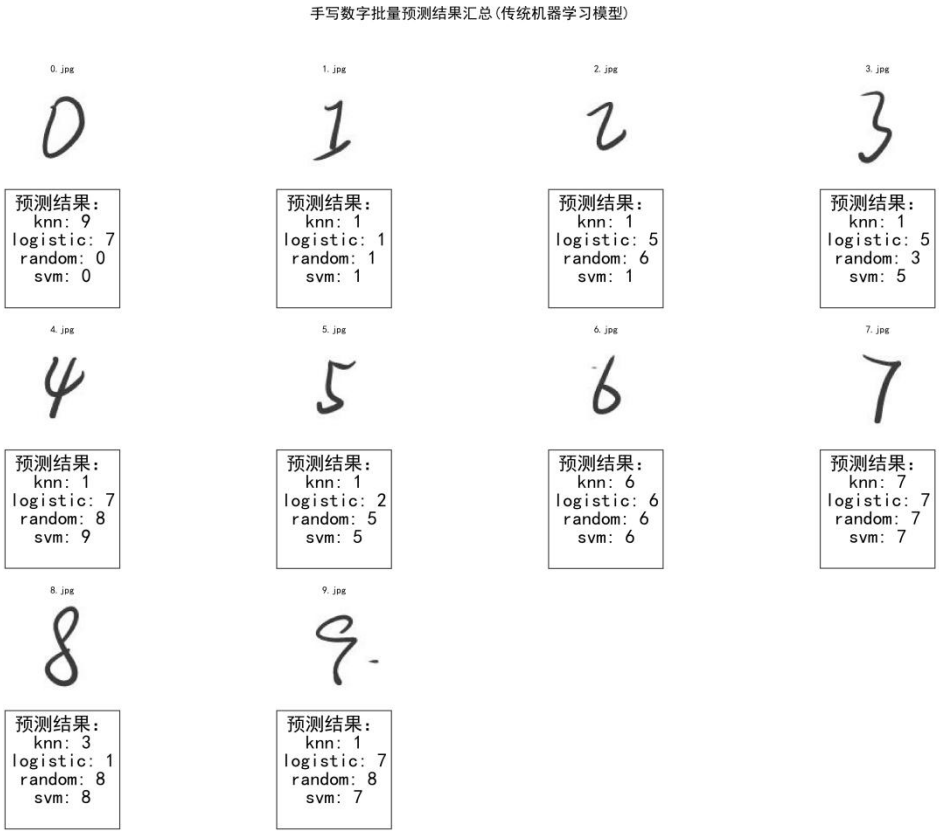

图 17 手写数字数据集 1 预测结果

下图为手写数据集 2 在各个传统机器学习模型的预测结果，我们可以发现，对于数字 0、数字 6、数字 8、数字 9，四个模型预测均有误；对于数字 4、数字 5，只有随机森林与支持向量机预测正确；对于数字 2、数字 3，只有随机森林预测准确；对于数字 7，只有逻辑回归和随机森林预测准确。

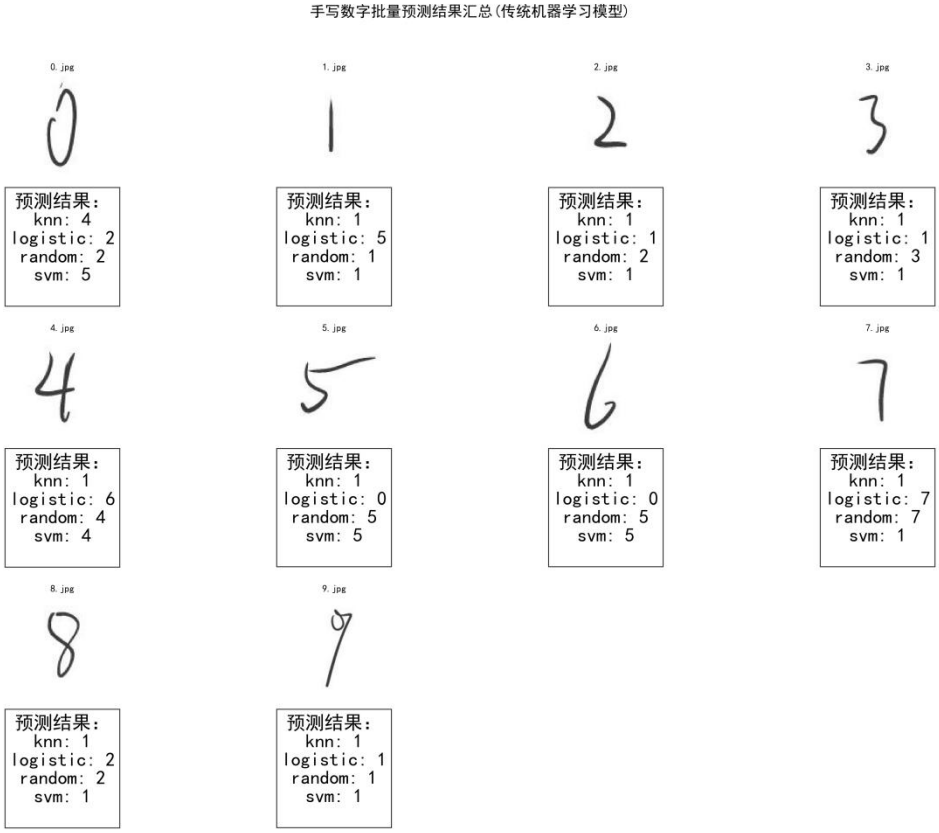

图 18 手写数字数据集 2 预测结果

下图为手写数据集 3 在各个传统机器学习模型的预测结果，我们可以发现，对于数字 0、数字 2、数字 3、数字 6、数字 8、数字 9，四个模型预测均有误；对于数字 1，只有逻辑回归预测错误；对于数字 4，只有 knn 预测错误；对于数字 5，只有随机森林和支持向量机预测准确；对于数字 7，四个模型预测均准确。

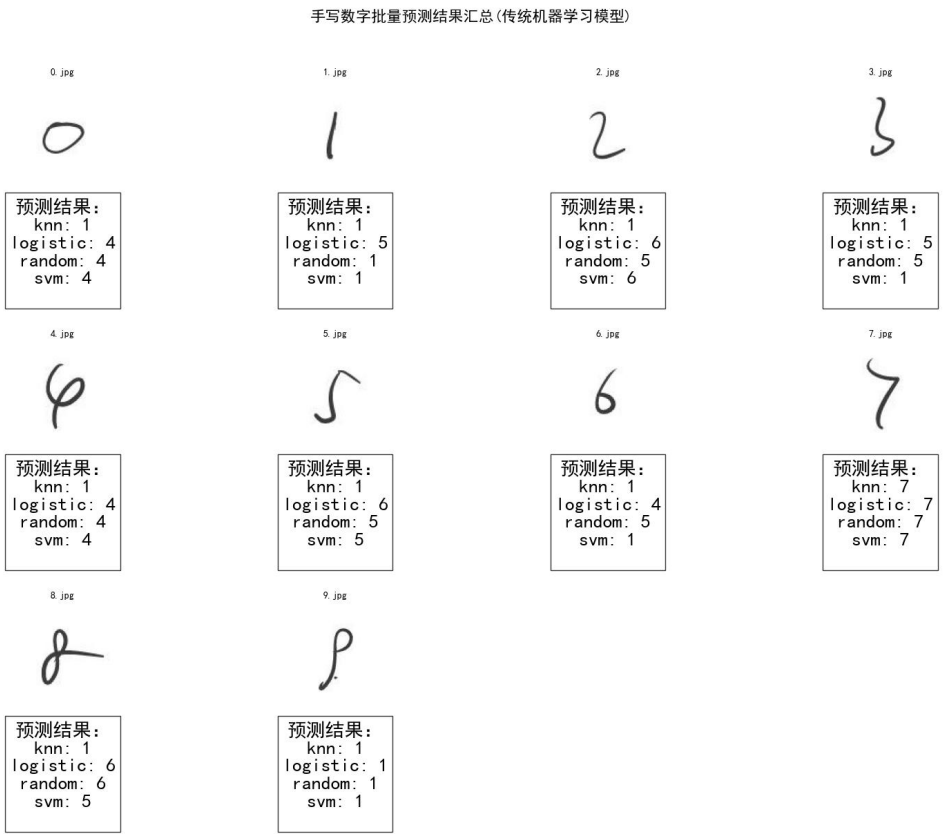

图 19 手写数字数据集 3 预测结果

总的来说，各类传统机器学习模型在手写数字数据集上的预测表现存在差异。对于特定数字如 1、6、7（数据集 1）、5（数据集 3）等，某些模型能准确预测，但对于数字 2、4、9（数据集 1）、0、6、8、9（数据集 2）、0、2、3、6、8、9（数据集 3）等，预测准确性普遍较低。其中，随机森林和支持向量机在多个数据集中对部分数字表现出了较好的预测能力，如在数据集 1 中对数字 0、5、8 的预测，在数据集 2 和 3 中对数字 4、5 的预测等。而 KNN 和逻辑回归在一些情况下容易出现误判。这表明不同模型在处理手写数字识别任务时，受数据特征和模型自身特性影响，各有优势与不足。

（三）深度学习模型预测结果

受篇幅限制，本节重点展示深度学习模型在手写数字数据集 1 上的预测表现。实验结果表明，相较于传统机器学习模型在该任务中不足 50%的准确率，深度学习模型展现出显著优势。以 CNN 模型为例，其在手写数据集 1 上的预测准确率达到 80%，仅出现两次预测错误，在图像特征提取与分类能力上远超传统模型。但需注意的是，部分预测正确的样本置信度较低，反映出模型对复杂手写体特征的泛化能力仍有提升空间。

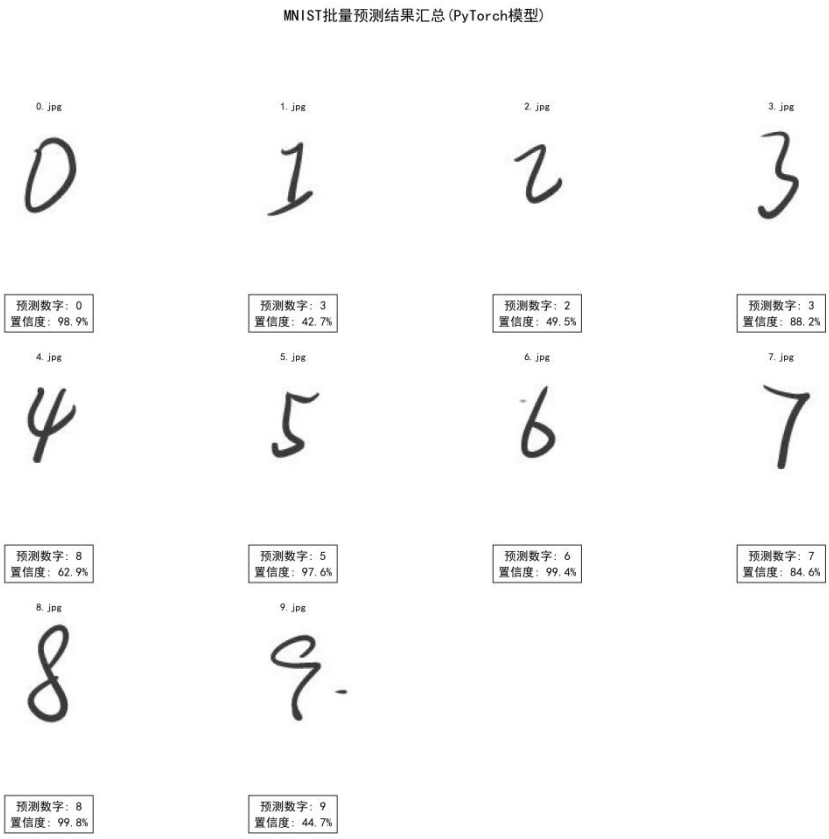

图 20 手写数字数据集 1 预测结果（CNN）

在 CNN 基础上优化的 CNN+模型，通过将训练轮次（epoch）提升 10 倍，实现了预测准确率的跨越式增长，达到 100%。更值得关注的是，该模型的预测置信度普遍接近 100%，说明加深训练深度不仅提升了模型的分类精度，还增强了对预测结果的确定性，有效解决了基础模型置信度不足的问题。

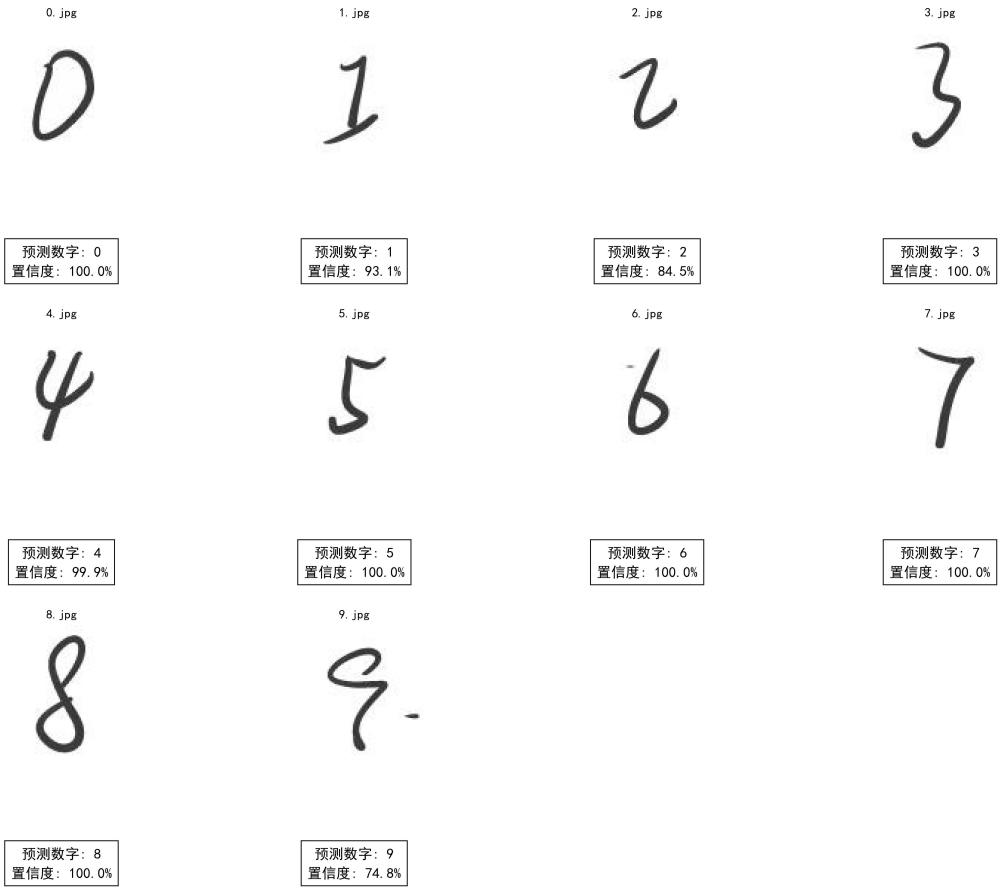

图 21 手写数字数据集 1 预测结果（CNN+）

基于残差网络的系列模型在手写数据集 1 上同样表现优异。ResNet、ResNet+与 ResNet++模型（见图 22、图 23）的预测准确率均达 100%，其中 ResNet+与 ResNet++分别将训练轮次设为基础 ResNet 模型的 4 倍与 10 倍。对比发现，在相同训练轮次下，ResNet 模型较 CNN 模型收敛速度更快，预测效果显著提升，充分验证了残差连接对深度网络训练效率的优化作用。此外，ResNet+模型的平均预测置信度高于 ResNet++，这一结果表明，过度增加训练轮次可能导致模型过拟合，并非 epoch 值越大，模型性能就越优，需在训练深度与泛化能力间寻求最佳平衡点。

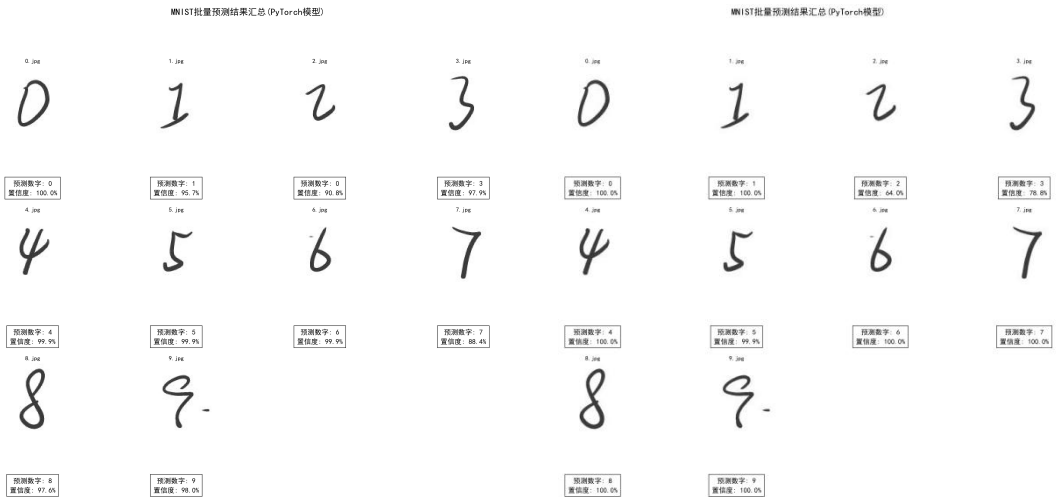

图 22 手写数字数据集 1 预测结果（左为 ResNet，右为 ResNet+）

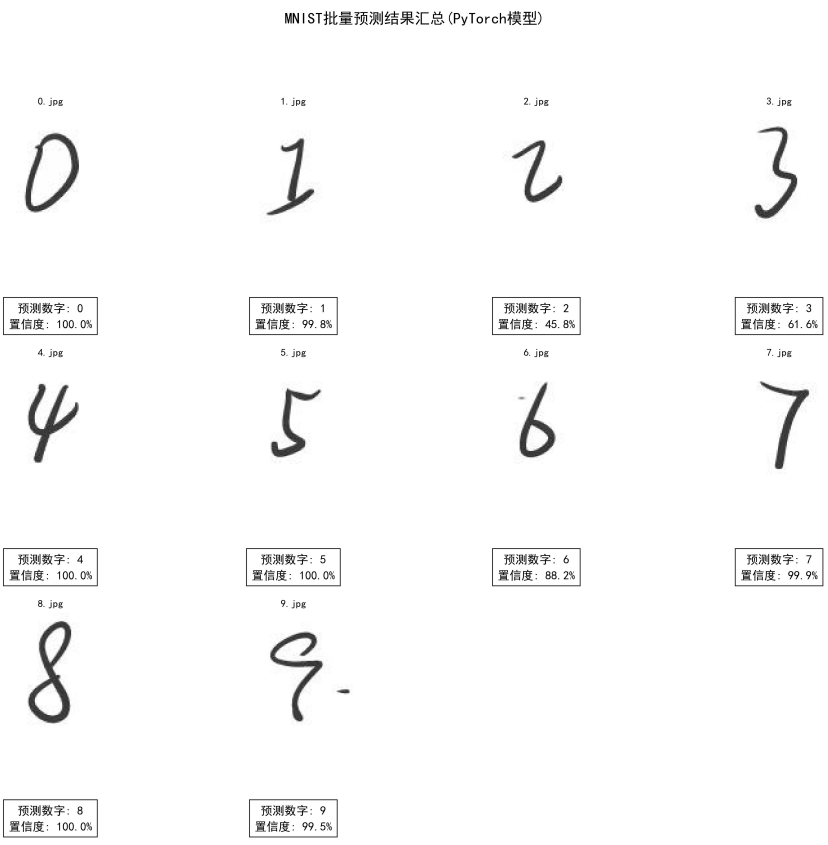

图 23 手写数字数据集 1 预测结果（ResNet++）

五、总结

本次实验围绕 MNIST 手写数字识别任务展开，对比分析了多种传统机器学习与深度学习模型的性能差异。在传统机器学习模型中，支持向量机（SVM）表现最优，准确率达 0.9566，其非线性分类能力有效区分了数字特征；K 近邻（KNN）和随机森林（random）准确率突破 0.94，性能稳定；而逻辑回归（logistic）因线性分类特性限制，准确率仅 0.8972，难以捕捉数据复杂关系。深度学习模

型优势显著，基础卷积神经网络（CNN）准确率达 0.9899，优化后的 CNN + 模型通过将 epoch 加深 10 倍，准确率提升至 0.9927 且置信度接近 100%；基于残差网络的 ResNet 系列中，ResNet+（epoch 为基础模型 4 倍）准确率 0.9915，优于 ResNet++（epoch 10 倍），表明过度增加 epoch 可能引发过拟合，需平衡训练深度与泛化能力。

在手写数字预测方面，传统机器学习模型在手写数据集上准确率普遍不足 50%，对数字 2、4、9 等复杂手写体误判率高，仅随机森林和支持向量机对部分数字（如 5、7）表现较好；深度学习模型则展现出强大性能，CNN 在手写数据集 1 上准确率 80%，CNN + 与 ResNet 系列均达 100%，且 ResNet 因残差连接收敛速度快于 CNN，验证了其对深度网络训练效率的优化作用。

实验结果表明，深度学习模型更适合处理 MNIST 图像数据，其特征提取能力显著优于传统模型；同一架构下合理优化训练策略（如调整 epoch）可提升性能，但需避免过拟合。未来可探索数据增强、引入 Transformer 等复杂架构或轻量化模型，进一步提升手写数字识别的泛化能力与效率。

六、参考文献

[1] https://github.com/cvdfoundation/mnist?tab=readme-ov-file

[2] https://blog.csdn.net/weixin_40522523/article/details/82823812

[3] https://blog.csdn.net/weixin_42393362/article/details/147897953

[4] 《深度学习入门：基于 Python 的理论与实现》，斋藤康毅

[5] https://cloud.tencent.com/developer/article/1618598

[6] https://cloud.tencent.com/developer/article/1574868

[7] https://blog.csdn.net/m0_62224692/article/details/136565454

[8] 《机器学习入门到实战-MATLAB 实践应用》，冷雨泉等

[9] https://blog.csdn.net/qq_51872445/article/details/140467567

[10] https://cloud.tencent.com/developer/article/2109487

七、附录

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import struct
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, accuracy_score,
precision_recall_fscore_support
import os
# ---------------------- 数据读取部分 ----------------------
def read_idx_images(file_path):
    """读取 IDX 格式的图像文件"""
    with open(file_path, 'rb') as f:
        magic, num, rows, cols = struct.unpack('>IIII', f.read(16))
        images = np.fromfile(f, dtype=np.uint8).reshape(num, rows, cols)
    return images  # shape: (样本数, 28, 28)

def read_idx_labels(file_path):
    """读取 IDX 格式的标签文件"""
    with open(file_path, 'rb') as f:
        magic, num = struct.unpack('>II', f.read(8))
        labels = np.fromfile(f, dtype=np.uint8)
    return labels  # shape: (样本数,)

class MNISTDataset(Dataset):
    def __init__(self, images_path, labels_path, transform=None):
        self.images = read_idx_images(images_path)
        self.labels = read_idx_labels(labels_path)
        self.transform = transform

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        image = self.images[idx].astype(np.float32) / 255.0  # 归一化到
[0,1]
        label = self.labels[idx]
        if self.transform:
            image = self.transform(image)
        return torch.tensor(image).unsqueeze(0), torch.tensor(label)  #
```

增加通道维度

```
# ---------------------- 模型定义部分 ----------------------
class MNISTClassifier(nn.Module):
    def __init__(self):
        super(MNISTClassifier, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),  # 输入(1,28,28) ->
(32,28,28)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),       # (32,14,14)
            nn.Conv2d(32, 64, kernel_size=3, padding=1), # (64,14,14)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)        # (64,7,7)
        )
        self.classifier = nn.Sequential(
            nn.Linear(64 * 7 * 7, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)  # 展平
        x = self.classifier(x)
        return x

# ---------------------- 训练流程部分 ----------------------
def train_model(train_loader, model, criterion, optimizer, epochs, device):
    model.train()
    for epoch in range(epochs):
        running_loss = 0.0
        for batch_idx, (images, labels) in enumerate(train_loader):
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            if (batch_idx+1) % 100 == 0:
```

```
                print(f'Epoch [{epoch+1}/{epochs}], Batch
[{batch_idx+1}/{len(train_loader)}], Loss: {running_loss/100:.4f}')
                running_loss = 0.0
    return model

# ---------------------- 测试评估部分 ----------------------
def evaluate_model(test_loader, model, device):
    model.eval()
    all_labels = []
    all_preds = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)

            all_labels.extend(labels.cpu().numpy())
            all_preds.extend(preds.cpu().numpy())

    # 计算核心指标
    accuracy = accuracy_score(all_labels, all_preds)
    precision, recall, f1, _ = precision_recall_fscore_support(
        all_labels, all_preds,
        average='macro',  # 宏平均（各类别等权重）
        zero_division=0
    )

    # 生成详细分类报告（包含各类别指标）
    class_report = classification_report(
        all_labels, all_preds,
        target_names=[str(i) for i in range(10)],
        output_dict=True
    )

    return {
        'accuracy': round(accuracy, 4),
        'precision': round(precision, 4),
        'recall': round(recall, 4),
        'f1': round(f1, 4),
        'class_report': class_report,  # 各类别详细指标
        'true_labels': all_labels,
        'pred_labels': all_preds
    }
```

```python
# ---------------------- 主程序 ----------------------
if __name__ == "__main__":
    # 参数设置
    DATA_DIR = ''  # 数据文件存放路径
    TRAIN_IMAGES = os.path.join(DATA_DIR, 'train-images.idx3-ubyte')
    TRAIN_LABELS = os.path.join(DATA_DIR, 'train-labels.idx1-ubyte')
    TEST_IMAGES = os.path.join(DATA_DIR, 't10k-images.idx3-ubyte')
    TEST_LABELS = os.path.join(DATA_DIR, 't10k-labels.idx1-ubyte')

    BATCH_SIZE = 128
    EPOCHS = 50
    LEARNING_RATE = 0.001
    MODEL_PATH = '1/mnist_classifier_50.pth'
    RESULT_PATH = '1/prediction_results_50.csv'
    METRICS_PATH = '1/evaluation_metrics_50.csv'  # 新增指标保存路径

    # 设备选择
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    # 数据加载
    train_dataset = MNISTDataset(TRAIN_IMAGES, TRAIN_LABELS)
    train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)

    test_dataset = MNISTDataset(TEST_IMAGES, TEST_LABELS)
    test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)

    # 模型初始化
    model = MNISTClassifier().to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)

    # 训练模型
    print("开始训练...")
    model = train_model(train_loader, model, criterion, optimizer, EPOCHS,
device)

    # 保存模型
    torch.save(model.state_dict(), MODEL_PATH)
    print(f'模型已保存至 {MODEL_PATH}')
```

```python
# 加载模型（演示后续调用）
loaded_model = MNISTClassifier().to(device)
loaded_model.load_state_dict(torch.load(MODEL_PATH))

# 评估测试集
print("开始评估测试集...")
eval_results = evaluate_model(test_loader, loaded_model, device)

# 保存预测结果（样本级）
results_df = pd.DataFrame({
    'sample_index': range(len(eval_results['true_labels'])),
    'true_label': eval_results['true_labels'],
    'pred_label': eval_results['pred_labels']
})
results_df.to_csv(RESULT_PATH, index=False)
print(f'预测结果已保存至 {RESULT_PATH}')

# 保存总体评估指标
metrics_df = pd.DataFrame({
    'metric': ['accuracy', 'precision', 'recall', 'f1'],
    'value': [
        eval_results['accuracy'],
        eval_results['precision'],
        eval_results['recall'],
        eval_results['f1']
    ]
})
metrics_df.to_csv(METRICS_PATH, index=False)
print(f'评估指标已保存至 {METRICS_PATH}')

# 打印详细指标
print("\n===== 总体评估指标 =====")
print(f"准确率: {eval_results['accuracy']}")
print(f"宏平均精确率: {eval_results['precision']}")
print(f"宏平均召回率: {eval_results['recall']}")
print(f"宏平均 F1 值: {eval_results['f1']}")

print("\n===== 各类别详细指标 =====")
print(classification_report(
    eval_results['true_labels'],
    eval_results['pred_labels'],
    target_names=[str(i) for i in range(10)]
```

```
    ))
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import struct
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, accuracy_score,
precision_recall_fscore_support
import os

# ---------------------- 数据读取部分（保持不变） ----------------------
def read_idx_images(file_path):
    """读取 IDX 格式的图像文件"""
    with open(file_path, 'rb') as f:
        magic, num, rows, cols = struct.unpack('>IIII', f.read(16))
        images = np.fromfile(f, dtype=np.uint8).reshape(num, rows, cols)
    return images  # shape: (样本数, 28, 28)

def read_idx_labels(file_path):
    """读取 IDX 格式的标签文件"""
    with open(file_path, 'rb') as f:
        magic, num = struct.unpack('>II', f.read(8))
        labels = np.fromfile(f, dtype=np.uint8)
    return labels  # shape: (样本数,)

class MNISTDataset(Dataset):
    def __init__(self, images_path, labels_path, transform=None):
        self.images = read_idx_images(images_path)
        self.labels = read_idx_labels(labels_path)
        self.transform = transform

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        image = self.images[idx].astype(np.float32) / 255.0  # 归一化到
[0,1]
        label = self.labels[idx]
```

```
        if self.transform:
            image = self.transform(image)
        return torch.tensor(image).unsqueeze(0), torch.tensor(label)  # 增加通道维度

# ---------------------- ResNet 核心组件 ----------------------
class ResidualBlock(nn.Module):
    """残差块实现"""
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResidualBlock, self).__init__()
        # 主路径卷积
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        # 跳跃连接（当通道数或尺寸变化时调整）
        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        out += self.shortcut(residual)  # 残差连接
        out = self.relu(out)
        return out

# ---------------------- ResNet 模型定义 ----------------------
class MNISTResNet(nn.Module):
    def __init__(self, num_classes=10):
        super(MNISTResNet, self).__init__()
```

```
        # 初始卷积层（适应 MNIST 28x28 输入）
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, stride=1, padding=1,
bias=False)  # 输入(1,28,28)->(16,28,28)
        self.bn1 = nn.BatchNorm2d(16)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)  # (16,14,14)

        # 残差块堆叠（2 个残差块组）
        self.layer1 = self._make_layer(16, 32, num_blocks=2, stride=2)  #
(32,7,7)
        self.layer2 = self._make_layer(32, 64, num_blocks=2, stride=2)  #
(64,3,3) 注：实际计算(7-2)/2+1=3

        # 全局平均池化 + 全连接层
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))  # (64,1,1)
        self.fc = nn.Linear(64, num_classes)

    def _make_layer(self, in_channels, out_channels, num_blocks, stride):
        """构建残差块组"""
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(ResidualBlock(in_channels, out_channels,
stride))
            in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)

        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x


# ---------------------- 训练流程部分（保持不变） ----------------------
def train_model(train_loader, model, criterion, optimizer, epochs, device):
```

```
    model.train()
    for epoch in range(epochs):
        running_loss = 0.0
        for batch_idx, (images, labels) in enumerate(train_loader):
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            if (batch_idx+1) % 100 == 0:
                print(f'Epoch [{epoch+1}/{epochs}], Batch
[{batch_idx+1}/{len(train_loader)}], Loss: {running_loss/100:.4f}')
                running_loss = 0.0
    return model

# ---------------------- 测试评估部分（保持不变） ----------------------
def evaluate_model(test_loader, model, device):
    model.eval()
    all_labels = []
    all_preds = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)

            all_labels.extend(labels.cpu().numpy())
            all_preds.extend(preds.cpu().numpy())

    accuracy = accuracy_score(all_labels, all_preds)
    precision, recall, f1, _ = precision_recall_fscore_support(
        all_labels, all_preds,
        average='macro',
        zero_division=0
    )

    class_report = classification_report(
        all_labels, all_preds,
        target_names=[str(i) for i in range(10)],
```

```
        output_dict=True
    )

    return {
        'accuracy': round(accuracy, 4),
        'precision': round(precision, 4),
        'recall': round(recall, 4),
        'f1': round(f1, 4),
        'class_report': class_report,
        'true_labels': all_labels,
        'pred_labels': all_preds
    }


# ---------------------- 主程序（仅修改模型类名） ----------------------
if __name__ == "__main__":
    DATA_DIR = ''  # 数据文件存放路径
    TRAIN_IMAGES = os.path.join(DATA_DIR, 'train-images.idx3-ubyte')
    TRAIN_LABELS = os.path.join(DATA_DIR, 'train-labels.idx1-ubyte')
    TEST_IMAGES = os.path.join(DATA_DIR, 't10k-images.idx3-ubyte')
    TEST_LABELS = os.path.join(DATA_DIR, 't10k-labels.idx1-ubyte')

    BATCH_SIZE = 128
    EPOCHS = 50  # ResNet 收敛更快，可适当减少训练轮数
    LEARNING_RATE = 0.001
    MODEL_PATH = '2/mnist_resnet_classifier_50.pth'
    RESULT_PATH = '2/resnet_prediction_results_50.csv'
    METRICS_PATH = '2/resnet_evaluation_metrics_50.csv'

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    # 数据加载（保持不变）
    train_dataset = MNISTDataset(TRAIN_IMAGES, TRAIN_LABELS)
    train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)

    test_dataset = MNISTDataset(TEST_IMAGES, TEST_LABELS)
    test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)

    # 模型初始化（改为 ResNet）
    model = MNISTResNet().to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

```
    # 训练模型（保持不变）
    print("开始训练...")
    model = train_model(train_loader, model, criterion, optimizer, EPOCHS,
device)

    # 保存模型（保持不变）
    torch.save(model.state_dict(), MODEL_PATH)
    print(f'模型已保存至 {MODEL_PATH}')

    # 加载模型（保持不变）
    loaded_model = MNISTResNet().to(device)
    loaded_model.load_state_dict(torch.load(MODEL_PATH))

    # 评估测试集（保持不变）
    print("开始评估测试集...")
    eval_results = evaluate_model(test_loader, loaded_model, device)

    # 保存结果（保持不变）
    results_df = pd.DataFrame({
        'sample_index': range(len(eval_results['true_labels'])),
        'true_label': eval_results['true_labels'],
        'pred_label': eval_results['pred_labels']
    })
    results_df.to_csv(RESULT_PATH, index=False)
    print(f'预测结果已保存至 {RESULT_PATH}')

    metrics_df = pd.DataFrame({
        'metric': ['accuracy', 'precision', 'recall', 'f1'],
        'value': [
            eval_results['accuracy'],
            eval_results['precision'],
            eval_results['recall'],
            eval_results['f1']
        ]
    })
    metrics_df.to_csv(METRICS_PATH, index=False)
    print(f'评估指标已保存至 {METRICS_PATH}')

    # 打印指标（保持不变）
    print("\n===== 总体评估指标 =====")
    print(f"准确率: {eval_results['accuracy']}")
```

```
print(f"宏平均精确率: {eval_results['precision']}")
print(f"宏平均召回率: {eval_results['recall']}")
print(f"宏平均 F1 值: {eval_results['f1']}")

print("\n===== 各类别详细指标 =====")
print(classification_report(
    eval_results['true_labels'],
    eval_results['pred_labels'],
    target_names=[str(i) for i in range(10)]
))
```

```
数据观察

# 数据集\train-images.idx3-ubyte

# 数据集\train-labels.idx1-ubyte


import struct

import numpy as np

import matplotlib.pyplot as plt


# 读取训练集数据

with open('数据集/train-images.idx3-ubyte', 'rb') as f:

    magic, num, rows, cols = struct.unpack('>IIII', f.read(16))

    train_images = np.fromfile(f, dtype=np.uint8).reshape(num, rows, cols)


# 读取训练集标签

with open('数据集/train-labels.idx1-ubyte', 'rb') as f:

    magic, num = struct.unpack('>II', f.read(8))

    train_labels = np.fromfile(f, dtype=np.uint8)


# 显示前 10 个样本
```

```
fig, axes = plt.subplots(2, 5, figsize=(10, 4))
for i, ax in enumerate(axes.flatten()):
    ax.imshow(train_images[i], cmap='gray')
    ax.set_title(train_labels[i])
    ax.axis('off')
plt.tight_layout()
plt.show()
import matplotlib.pyplot as plt


def show_sample(image, label):
    plt.imshow(image, cmap='gray')
    plt.title(f"Label: {label}")
    plt.axis('off')
    plt.show()


# 查看第一个训练样本
show_sample(train_images[0], train_labels[0])
# 统计各个标签数量
label_counts = np.bincount(train_labels)


# 绘制标签分布图（加上具体数字，颜色）


fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(range(10), label_counts,color='skyblue')
for i, v in enumerate(label_counts):
    ax.text(i, v + 100, f'{v}', ha='center', fontsize=12)
ax.set_xlabel('Label')
ax.set_ylabel('Count')
```

```
ax.set_title('Distribution of Labels in the Training Set')
plt.show()
传统机器学习方法


    # 示例：后续调用模型预测（可单独使用）
    # loaded_model = load('models/random_forest.pkl')
    # sample_pred = loaded_model.predict(X_test[:5])
    # print("示例预测结果:", sample_pred)
import numpy as np
import struct
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import joblib
import os


def read_idx_file(file_path):
    """读取 MNIST IDX 格式二进制文件"""
    with open(file_path, 'rb') as f:
        # 读取文件头（魔数+维度信息）
        if 'images' in file_path:
            magic, num_images, rows, cols = struct.unpack('>IIII', f.read(16))
            data = np.fromfile(f, dtype=np.uint8).reshape(num_images, rows*cols)
            return data.astype(np.float32) / 255.0  # 归一化到[0,1]
        else:
```

```
                magic, num_labels = struct.unpack('>II', f.read(8))
                return np.fromfile(f, dtype=np.uint8)

    def train_and_save_models(X_train, y_train, model_dir='saved_models'):
        """训练多种传统模型并保存"""
        os.makedirs(model_dir, exist_ok=True)

        models = {
            'logistic_regression': LogisticRegression(max_iter=1000, n_jobs=-1),
            'svm_rbf': SVC(kernel='rbf', gamma='scale'),
            'random_forest': RandomForestClassifier(n_estimators=100, n_jobs=-1),
            'knn': KNeighborsClassifier(n_neighbors=5, n_jobs=-1)
        }

        for name, model in models.items():
            print(f'训练 {name} 模型中...')
            model.fit(X_train, y_train)
            joblib.dump(model, os.path.join(model_dir, f'{name}_mnist.pkl'))
        print('所有模型训练并保存完成\n')

    def evaluate_and_save(X_test, y_test, model_dir='saved_models',
result_path='evaluation_results.csv'):
        """评估模型并保存结果到表格"""
        results = []

        for model_file in os.listdir(model_dir):
            if not model_file.endswith('.pkl'):
                continue
```

```
model_path = os.path.join(model_dir, model_file)
model = joblib.load(model_path)
model_name = model_file.split('_')[0]

# 模型预测
y_pred = model.predict(X_test)

# 收集评估指标
acc = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, output_dict=True)

# 提取关键指标（宏观平均）
metrics = {
    '模型名称': model_name,
    '准确率': acc,
    '精确率（宏平均）': report['macro avg']['precision'],
    '召回率（宏平均）': report['macro avg']['recall'],
    'F1 值（宏平均）': report['macro avg']['f1-score'],
    '样本量': report['macro avg']['support']
}

# 收集混淆矩阵对角线（正确分类数）
cm = confusion_matrix(y_test, y_pred)
metrics.update({f'类别{i}_正确数': cm[i,i] for i in range(10)})

results.append(metrics)
```

```
    # 保存为表格
    df = pd.DataFrame(results)
    df.to_csv(result_path, index=False)
    print(f'评估结果已保存至 {result_path}')
    return df

if __name__ == '__main__':
    # 数据路径（根据实际存储位置调整）
    data_paths = {
        'train_images': '数据集/t10k-images.idx3-ubyte',  # 用户指定的训练集图像
        'train_labels': '数据集/t10k-labels.idx1-ubyte',  # 用户指定的训练集标签
        'test_images': '数据集/train-images.idx3-ubyte',   # 用户指定的测试集图像
        'test_labels': '数据集/train-labels.idx1-ubyte'     # 用户指定的测试集标签
    }

    # 读取数据（注意用户数据命名可能与常规相反）
    print('读取训练数据...')
    X_train = read_idx_file(data_paths['train_images'])
    y_train = read_idx_file(data_paths['train_labels'])

    print('读取测试数据...')
    X_test = read_idx_file(data_paths['test_images'])
    y_test = read_idx_file(data_paths['test_labels'])

    # 训练并保存模型
    train_and_save_models(X_train, y_train)

    # 评估并保存结果
```

```
    evaluate_and_save(X_test, y_test)
```

测试

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

# 定义字体
font = {'family': 'SimHei', 'size': 14}  # 设置字体为黑体
plt.rc('font', **font)  # 应用字体设置

def preprocess_image(image_path, target_size=(28, 28)):
    """
    预处理图像以匹配模型输入要求
    参数:
        image_path: 实际图片路径（支持 png/jpg 等格式）
        target_size: 目标尺寸（MNIST 为 28x28）
    返回:
        预处理后的一维数组（784 个特征）
    """
    # 读取图像并转换为灰度图
    img = Image.open(image_path).convert('L')

    # 调整尺寸（保持宽高比，不足部分填充黑色）
    img = img.resize(target_size, Image.Resampling.LANCZOS)
```

```
    # 转换为 numpy 数组并归一化（0-255 → 0-1）
    img_array = np.array(img, dtype=np.float32) / 255.0

    # MNIST 是黑底白字（0 背景，255 数字），若实际图片是白底黑字需要反转
    # 可根据实际情况注释或取消注释下一行
    img_array = 1.0 - img_array  # 反转颜色（如果输入是白底黑字）

    # 展平为一维数组（与训练数据格式一致）
    return img_array.flatten().reshape(1, -1)

def load_models(model_dir='saved_models'):
    """加载所有保存的模型"""
    models = {}
    for model_file in os.listdir(model_dir):
        if model_file.endswith('.pkl'):
            model_name = model_file.split('_')[0]
            models[model_name] = joblib.load(os.path.join(model_dir, model_file))
    return models

def predict_image(image_path, models):
    """
    对单张图片进行多模型预测
    返回: 包含各模型预测结果的字典
    """
    # 预处理图像
    processed_img = preprocess_image(image_path)

    # 多模型预测
```

```
        predictions = {}
        for model_name, model in models.items():
            pred = model.predict(processed_img)
            predictions[model_name] = pred[0]  # 取出单个预测结果
        return predictions


def visualize_result(image_path, predictions):
        """可视化预测结果"""
        # 读取原始图像用于展示
        img = Image.open(image_path)

        # 创建展示窗口
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

        # 显示原始图像
        ax1.imshow(img)
        ax1.set_title('原始图片')
        ax1.axis('off')

        # 显示预测结果
        result_text = "\n 模型预测结果：\n"
        for model_name, pred in predictions.items():
            result_text += f"{model_name}: {pred}\n"



        # 更大
        ax2.text(0.3, 0.4, result_text, fontsize=18,
                 bbox=dict(facecolor='white', alpha=0.8))
```

```
    ax2.set_title('各模型预测结果')
    ax2.axis('off')

    plt.tight_layout()
    plt.show()

if __name__ == '__main__':
    # 配置参数（根据实际情况修改）
    IMAGE_PATH = '数据集/手写数字图像/1/7.jpg'  # 替换为你的实际图片路径
    MODEL_DIR = 'saved_models'        # 模型保存目录（与训练代码一致）

    # 加载所有训练好的模型
    print('加载模型中...')
    models = load_models(MODEL_DIR)

    # 对指定图片进行预测
    print(f'正在预测图片: {IMAGE_PATH}')
    predictions = predict_image(IMAGE_PATH, models)

    # 打印文本结果
    print('\n 预测结果：')
    for model_name, pred in predictions.items():
        print(f'{model_name.ljust(18)}: {pred}')

    # 可视化结果（需要图形界面支持）
    visualize_result(IMAGE_PATH, predictions)

import numpy as np
```

```
import joblib
from PIL import Image
import os
import matplotlib.pyplot as plt
from glob import glob


def preprocess_image(image_path, target_size=(28, 28)):
    """图像预处理（与单张预测版本保持一致）"""
    img = Image.open(image_path).convert('L')
    img = img.resize(target_size, Image.Resampling.LANCZOS)
    img_array = np.array(img, dtype=np.float32) / 255.0
    # 若需要颜色反转（白底黑字转黑底白字），取消下一行注释
    img_array = 1.0 - img_array
    return img_array.flatten().reshape(1, -1)


def load_models(model_dir='saved_models'):
    """加载所有保存的模型（与单张预测版本保持一致）"""
    return {
        model_file.split('_')[0]: joblib.load(os.path.join(model_dir, model_file))
        for model_file in os.listdir(model_dir) if model_file.endswith('.pkl')
    }


def predict_image(image_path, models):
    """单张图片预测（返回带图片路径的结果字典）"""
    processed_img = preprocess_image(image_path)
    return {
        'image_path': image_path,
        'predictions': {name: model.predict(processed_img)[0]
```

```
                for name, model in models.items()}
    }


def batch_predict(image_folder, models):
    """批量预测文件夹中的所有图片"""
    # 获取所有图片路径（支持常见图片格式）
    image_paths = glob(os.path.join(image_folder, '*.png')) + \
                  glob(os.path.join(image_folder, '*.jpg')) + \
                  glob(os.path.join(image_folder, '*.jpeg'))

    # 批量预测并收集结果
    results = []
    for img_path in image_paths:
        try:
            results.append(predict_image(img_path, models))
        except Exception as e:
            print(f'警告：处理 {img_path} 时出错: {str(e)}，已跳过')
    return results


def create_summary_plot(results, output_path='prediction_summary.png', cols=4):
    """
    创建批量预测结果汇总图
    参数:
        results: 批量预测结果列表（包含 image_path 和 predictions）
        output_path: 汇总图保存路径
        cols: 每行显示的图片数量（自动计算行数）
    """
    # 计算布局参数
```

```
rows = (len(results) + cols - 1) // cols
fig, axes = plt.subplots(rows, cols, figsize=(cols*4, rows*4))
fig.suptitle('手写数字批量预测结果汇总(传统机器学习模型)', fontsize=16, y=1.02)

# 遍历每个结果绘制子图
for idx, result in enumerate(results):
    row, col = idx // cols, idx % cols
    ax = axes[row, col] if rows > 1 else axes[col] if cols > 1 else axes

    # 显示原始图片
    img = Image.open(result['image_path'])
    ax.imshow(img)
    ax.axis('off')

    # 生成预测结果文本
    pred_text = "预测结果：\n"
    for model_name, pred in result['predictions'].items():
        pred_text += f"{model_name}: {pred}\n"

    # 添加结果标注（半透明背景框）
    ax.text(0.5, -0.15, pred_text,
            transform=ax.transAxes,
            ha='center', va='top',
            bbox=dict(facecolor='white', alpha=0.8, pad=5),
            fontsize=20)

    # 设置子图标题（原始文件名）
    ax.set_title(os.path.basename(result['image_path']), fontsize=10)
```

```
    # 隐藏多余的子图（当图片数不是 cols 整数倍时）
    for idx in range(len(results), rows*cols):
        row, col = idx // cols, idx % cols
        axes[row, col].axis('off')

    plt.tight_layout()
    plt.savefig(output_path, bbox_inches='tight', dpi=300)
    print(f'汇总图已保存至: {output_path}')

if __name__ == '__main__':
    # 配置参数（根据实际情况修改）
    IMAGE_FOLDER = '数据集/手写数字图像/3'    # 输入图片文件夹路径
    MODEL_DIR = 'saved_models'        # 模型保存目录
    OUTPUT_PATH = 'r3_t.png'   # 汇总图保存路径

    # 加载模型
    print('加载模型中...')
    models = load_models(MODEL_DIR)

    # 批量预测
    print(f'正在处理文件夹: {IMAGE_FOLDER}')
    results = batch_predict(IMAGE_FOLDER, models)
    print(f'完成 {len(results)} 张图片的预测')

    # 生成并保存汇总图
    if results:
        create_summary_plot(results, OUTPUT_PATH)
```

```
else:
    print('警告：未找到可处理的图片文件')
```