基于卷积神经网络的手写体识别问题

宋子源，230249167
(东南大学电气工程学院，江苏省南京市)

摘要：

本文针对 UCI 机器学习数据库中的 8×8 小尺度 0-9 手写数字集，在 LeNet-5 基础上改进了神经网络结构。针对小尺寸输入，网络架构在卷积核、池化层及参数规模上进行了细致调整，并引入具备 LeakyRelu 激活函数避免梯度消失并增强模型表达力。训练环节采用了 He 权重初始化方式，结合动态的学习率递减机制，显著提升了收敛速度与梯度传播的稳定性。同时，利用数据噪声扰动手段，有效提升了模型的泛化能力与实际鲁棒性。为更契合分类任务的特性，损失函数采用 Softmax 交叉熵，从而优化了多类别识别的训练表现。全部算法流程自主实现，未依赖现成深度学习框架。实验结果显示，改进后的网络在 8×8 手写数字识别任务上获得 98.27%准确率，展现出良好的计算效率和优异识别性能。最后针对数据噪声、学习率、批量学习进行了消融实验，定量分析了其对测试集准确率的影响。

数据集：http://archive.ics.uci.edu/dataset/80/optical+recognition+of+handwritten+digits
代码开源：https://github.com/songziyuanszy/Neural-Networks

一、问题描述

手写数字识别作为模式识别领域的经典问题，广泛应用于邮政编码识别、银行票据处理等实际场景。本次大作业的任务是基于 UCI 机器学习数据库中的手写数字数据集，对 0-9 这 10 个手写体数字图像进行分类和识别。

1.1 数据来源

所用数据集来源于 UCI Machine Learning Repository 的 Optical Recognition of Handwritten Digits Data Set。数据为手写数字图片提取的特征，已预处理为便于分类的结构化数据。从零开始手工编写所有算法源码，不依赖任何现成的机器学习库函数，实现模型的训练和测试过程。

在本次实验中，原始数据集被划分为训练集和测试集。训练集包括 3823 个样本，每个样本为 64 维特征向量，标签为单一的数字类别（y_train，共 3823 个）。测试集包含 1797 个样本，每个样本同样为 64 维特征向量，每个样本为 8×8 的灰度图像，展平为一维 64 维特征向量，对应 1797 个测试标签（y_test）。每个特征均为整数，表示像素灰度值，最小值为 0，最大值为 16。标签类别是 0-9 的整数，分别对应手写数字的真实类别。

```
print(X_train.shape)
(3823, 64)

print(y_train.shape)
(3823,)

print(X_test.shape)
(1797, 64)

print(y_test.shape)
(1797,)
```

- 训练集特征矩阵：X_train.shape = (3823, 64)，其中每一行代表一个手写数字样本，64 列为其展平后的像素灰度值。
- 训练集标签向量：y_train.shape = (3823,)，其中每个元素为该样本对应的数字类别（0-9）。
- 测试集特征矩阵：X_test.shape = (1797, 64)，格式与训练集一致，用于性能评估。
- 测试集标签向量：y_test.shape = (1797,)，代表测试样本的真实类别。

1.2 数据预处理

模型在训练集上进行参数学习，并在测试集上评估识别准确率，以此验证卷积神经网络的识别能力和泛化性能。

```
# 归一化（样本数，通道数，高，宽）
X_train = X_train.reshape(-1, 1, 8, 8) / 16.0
X_test = X_test.reshape(-1, 1, 8, 8) / 16.0

# one-hot编码
def one_hot(y, num_classes=10):
    n = y.shape[0]
    y_one_hot = np.zeros((n, num_classes))
    y_one_hot[np.arange(n), y] = 1
    return y_one_hot

y_train_onehot = one_hot(y_train)
y_test_onehot = one_hot(y_test)
```

在数据预处理阶段，首先对原始的特征数据进行了归一化和形状调整。将每个样本从一维的 64 维向量重新变换为 (1, 8, 8) 的四维张量格式，其中 1 表示单通道（灰度图像），8×8 为原始图片的像素尺寸，并将像素值缩放到[0, 1]区间（即每个像素值除以 16），以消除数值量纲的影响。

同时，对样本标签进行了 one-hot 编码，将每个类别数字标签（0-9）转换为 10 维的独热向量，使标签适用于神经网络的多分类任务，确保了输入数据和标签格式均满足卷积神经网络模型的训练需求。

二、卷积神经网络与 LeNet-5 架构介绍

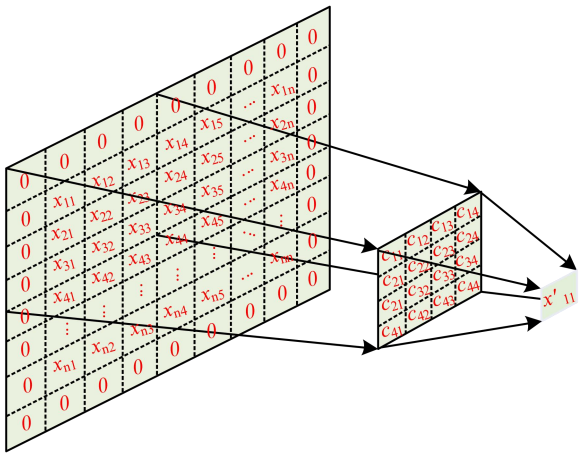
2.1 卷积神经网络

卷积神经网络（Convolutional Neural Network, CNN）通过卷积操作有效地提取数据的局部特征并降低模型参数，其结构简明易用，大幅简化了特征的提取过程，在生成对抗网络中被广泛使用。

CNN 核心在于利用一组卷积核来构建网络。卷积核是一种固定尺寸的参数矩阵，它在输入数据矩阵上按照特定步幅进行滑动。在每次滑动中，卷积核与其所覆盖的输入矩阵区域

进行点乘操作，从而捕捉到数据中的局部特征。由于卷积核内的参数数量在滑动过程中保持不变，从而极大地减少了整个网络的复杂度，进而提升了深度学习训练过程的稳定性。

卷积计算过程的示意如图所示。



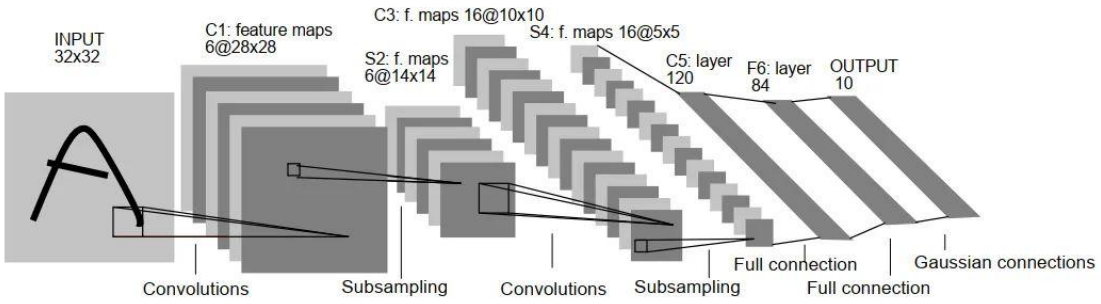
其工作过程可简述为：首先，将卷积核与输入数据的左上角对齐；接着，卷积核内每个元素与对应输入数据相乘，乘积求和产生新的特征图像素值；然后，卷积核按预设步长水平移动，重复上述过程直至数据右边缘，再垂直移动一步重新开始，直至覆盖所有输入数据。通过这种方式，卷积层生成包含编码空间特征的特征图，为 CNN 后续层处理提供基础。使用多个卷积核可以提取不同的特征图，进一步提升模型对数据的特征提取能力。

在深度卷积神经网络的设计中，每个卷积层后通常会接一个非线性转换函数，即激活函数，其作用是赋予网络处理非线性问题的能力。假若缺少这些激活函数，无论网络的深度如何，其表现力仅限于线性模型，这将大大限制了网络解决复杂问题的能力。

2.2 LeNet-5 架构

LeNet-5 是由 Yann LeCun 及其团队于 1998 年提出的一种经典 CNN 架构，首次清晰地展示了卷积层、池化层（或子采样层）和全连接层组合的有效性，为现代 CNN 架构奠定了基础，成为后续深度学习、计算机视觉领域发展的基石之一。

LeNet-5 网络由输入层、三个卷积层、两个子采样层（池化层）和两个全连接层组成，共计 7 层（不含输入层）。其设计体现了早期卷积神经网络的核心思想，LeNet-5 通过交替的卷积和池化操作逐步提取并压缩特征，最后通过全连接层完成分类。其紧凑的结构设计在保证特征提取能力的同时，有效控制了参数规模。



网络的输入为 32×32 像素的灰度图像，这一尺寸略大于 MNIST 数据集原始的 28×28 图像，以确保边缘特征能够被有效捕捉。第一卷积层（C1）采用 6 个 5×5 的卷积核，以步长 1 进行无填充卷积运算，输出 6 个 28×28 的特征图。每个卷积核包含 25 个可训练权重和 1 个偏置项，共计 156 个参数。该层通过局部连接和权值共享机制提取输入图像的低级特征，如

边缘和角点等。原始实现使用 tanh 或 sigmoid 作为激活函数，而现代实现中更倾向于采用 ReLU 以缓解梯度消失问题。

随后，网络通过第一个子采样层（S2）对特征图进行降维。该层采用 2×2 的平均池化操作，步长为 2，将每个 28×28 的特征图降采样至 14×14。与常规池化不同，LeNet-5 的子采样层引入了可学习的权重和偏置，使得下采样过程具有一定的自适应能力。这一设计在减少计算量的同时，增强了模型对平移和形变的鲁棒性。S2 层共包含 12 个可训练参数（每个特征图对应 1 个权重和 1 个偏置）。

第二卷积层（C3）由 16 个 5×5 的卷积核组成，输出 16 个 10×10 的特征图。值得注意的是，该层采用了部分连接策略，即每个卷积核仅与前一层的部分特征图相连。这种设计不仅减少了参数数量，还强制网络学习更具判别性的特征组合。根据原始论文中的连接表，前 6 个卷积核各连接 3 个输入特征图，中间 6 个连接 4 个，最后 4 个则连接全部 6 个输入特征图。假设平均每个卷积核连接 4 个输入特征图，则该层参数数量约为 1616 个。

第二个子采样层（S4）同样采用 2×2 的平均池化，将 10×10 的特征图压缩至 5×5。与 S2 层类似，该层包含可学习的权重和偏置，共计 32 个参数。经过两次卷积和池化操作后，原始图像的空间信息被逐步抽象为更高级的语义特征。

第三卷积层（C5）由 120 个 5×5 的卷积核组成，以全连接方式处理 S4 层的所有 16 个特征图。由于输入特征图的尺寸（5×5）与卷积核尺寸相同，每个卷积运算输出一个标量值，最终得到 120 维的特征向量。该层实质上完成了从空间特征到高级语义特征的转换，共包含 48120 个可训练参数。

第一个全连接层（F6）包含 84 个神经元，接收 C5 层的 120 维输出。该层通过全连接方式进一步整合特征，增强模型的非线性表达能力。原始实现使用 tanh 或 sigmoid 激活函数，而现代实现中 ReLU 更为常见。该层共计 10164 个参数。

输出层由 10 个神经元组成，对应数字 0-9 的分类任务。在原始论文中，该层采用欧几里得径向基函数（RBF）计算输入特征与预设类别模板的距离，距离越小则对应类别的得分越高。然而，现代实现普遍采用 Softmax 函数将原始得分转换为概率分布，使得输出具有明确的概率解释。该层包含 850 个可训练参数。

2.3 关键特点总结

LeNet-5 是最早成功实现端到端训练的神经网络之一。它直接从原始像素输入（经过预处理）学习到最终的分类输出，中间的特征提取过程完全由网络自动学习，无需手工设计特征。尽管结构相对简单，但其设计理念至今仍深刻影响着最先进的 CNN 模型（如 AlexNet, VGG, ResNet 等）。

- 卷积-池化交替结构：C1-S2-C3-S4 的结构清晰地展示了卷积层（用于特征提取）和池化层（用于空间下采样/特征压缩）交替堆叠的模式，这成为后续几乎所有 CNN 架构的标准范式。
- 层级特征学习：网络自动学习从低级特征（边缘）到中级特征（形状部件）再到高级特征（完整的数字模式）的层次化表示。
- 参数高效：通过局部连接、权值共享和下采样，LeNet-5 在参数数量上比同等能力的全连接网络要少得多（总计约 6 万个可学习参数），这使得在当时的计算条件下训练成为可能，并降低了过拟合风险。

三、算法实现

3.1 卷积层的前向传播与梯度更新

在卷积层的前向计算中，首先采用 He 初始化策略对权重矩阵进行正态分布采样，其标准差计算公式为 $\sqrt{2/(\text{in\_channels} \times \text{kernel\_size}^2)}$，有效适应 ReLU 激活函数的梯度特性。

输入数据经过零填充扩展后，通过三维切片与张量缩并操作实现特征提取：每个输出特征图位置(i,j)通过步长为 stride 的滑动窗口机制提取输入的 k×k 局部区域，采用 np.tensordot 实现输入切片与对应卷积核的逐元素乘积累加运算，最终叠加偏置项形成完整的特征映射。

输出尺寸计算遵循公式 out_size=(in_size-2×padding+kernel_size)/stride-1，确保窗口对齐与边界处理。

```
def forward(self, x):
    self.input = x
    n, c, h, w = x.shape

    # 计算输出尺寸
    out_h = (h + 2*self.padding - self.kernel_size) // self.stride + 1
    out_w = (w + 2*self.padding - self.kernel_size) // self.stride + 1
    # 添加padding
    if self.padding > 0:
        x_padded = np.pad(x, ((0,0), (0,0), (self.padding, self.padding), (self.padding, self.padding)))
    else:
        x_padded = x
    # 创建输出数组
    output = np.zeros((n, self.out_channels, out_h, out_w))

    # 向量化卷积
    for i in range(out_h):
        for j in range(out_w):
            h_start = i * self.stride
            h_end = h_start + self.kernel_size
            w_start = j * self.stride
            w_end = w_start + self.kernel_size
            # 一次提取所有样本和通道的切片
            x_slice = x_padded[:, :, h_start:h_end, w_start:w_end]

            # 向量化计算 (n, out_channels)
            output[:, :, i, j] = np.tensordot(
                x_slice, self.weights, axes=([1, 2, 3], [1, 2, 3])
            ) + self.bias
    return output
```

反向传播阶段构建了四阶张量计算链：首先通过输入切片与输出梯度的张量积计算权重梯度，保持输出通道维度；随后使用转置卷积操作实现输入梯度传播，通过梯度切片与卷积核的转置运算重建输入空间维度；偏置梯度则通过沿样本和空间维度求和获得。为提升计算效率，采用预分配梯度张量与切片索引机制，避免动态内存分配带来的性能损耗。

```python
def backward(self, dout):
    x = self.input
    n, c, h, w = x.shape
    out_h = dout.shape[2]
    out_w = dout.shape[3]
    dx = np.zeros_like(x)
    # 向量化梯度计算
    for i in range(out_h):
        for j in range(out_w):
            h_start = i * self.stride
            h_end = h_start + self.kernel_size
            w_start = j * self.stride
            w_end = w_start + self.kernel_size
            # 将梯度平均分配到每个位置
            grad = dout[:, :, i, j] / self.pool_area
            # 创建与池化区域相同形状的梯度数组
            grad_expanded = np.repeat(
                np.repeat(
                    grad[:, :, np.newaxis, np.newaxis],
                    self.kernel_size, axis=2
                ),
                self.kernel_size, axis=3
            )
            dx[:, :, h_start:h_end, w_start:w_end] += grad_expanded
    return dx
```

3.2 平均池化层的前向传播与梯度更新

平均池化层的前向传播通过滑动窗口计算局部区域均值，对输入张量的向量化切片操作：每个池化窗口位置通过步长为 stride 的滑动机制提取数据块，使用 np.mean 实现均值计算。

```python
def forward(self, x):
    self.input = x
    n, c, h, w = x.shape
    out_h = (h - self.kernel_size) // self.stride + 1
    out_w = (w - self.kernel_size) // self.stride + 1

    # 预计算输出形状
    output = np.zeros((n, c, out_h, out_w))

    # 平均池化
    for i in range(out_h):
        for j in range(out_w):
            h_start = i * self.stride
            h_end = h_start + self.kernel_size
            w_start = j * self.stride
            w_end = w_start + self.kernel_size

            # 一次提取所有样本和通道的切片
            x_slice = x[:, :, h_start:h_end, w_start:w_end]

            # 向量化计算平均值
            output[:, :, i, j] = np.mean(x_slice, axis=(2, 3))

    return output
```

反向传播采用梯度分配机制，首先将输出梯度除以窗口面积进行标准化，继而通过梯度张量的三维复制操作（沿高度与宽度维度重复 kernel_size 次）生成与输入匹配的梯度分布，最后通过切片索引实现多窗口梯度叠加，确保梯度准确回传至输入各像素点。

```python
def backward(self, dout):
    x = self.input
    n, c, h, w = x.shape
    out_h = dout.shape[2]
    out_w = dout.shape[3]
    dx = np.zeros_like(x)
    # 向量化梯度计算
    for i in range(out_h):
        for j in range(out_w):
            h_start = i * self.stride
            h_end = h_start + self.kernel_size
            w_start = j * self.stride
            w_end = w_start + self.kernel_size
            # 将梯度平均分配到每个位置
            grad = dout[:, :, i, j] / self.pool_area
            # 创建与池化区域相同形状的梯度数组
            grad_expanded = np.repeat(
                np.repeat(
                    grad[:, :, np.newaxis, np.newaxis],
                    self.kernel_size, axis=2
                ),
                self.kernel_size, axis=3
            )
            dx[:, :, h_start:h_end, w_start:w_end] += grad_expanded
    return dx
```

3.3 交叉熵损失实现方法

Softmax 交叉熵损失函数的实现首先在前向传播阶段采用数值稳定性优化策略，通过减去输入张量的行最大值避免指数运算溢出，计算归一化后的概率分布，再利用真实标签索引提取对应概率值，结合对数运算和微小常数 epsilon 防止数值不稳定，最终通过求和平均得到损失值；

```python
class SoftmaxCrossEntropyLoss:
    def forward(self, x, y):
        # 数值稳定性的softmax
        max_vals = np.max(x, axis=1, keepdims=True)
        exp_vals = np.exp(x - max_vals)
        self.probs = exp_vals / np.sum(exp_vals, axis=1, keepdims=True)

        # 计算交叉熵损失
        n = y.shape[0]
        log_probs = -np.log(self.probs[np.arange(n), np.argmax(y, axis=1)] + 1e-8)
        loss = np.sum(log_probs) / n
        return loss

    def backward(self, y):
        n = y.shape[0]
        dx = self.probs.copy()
        dx[np.arange(n), np.argmax(y, axis=1)] -= 1
        return dx / n
```

反向传播阶段则基于前向计算得到的概率分布，通过复制概率矩阵并在真实类别位置减去 1 构建梯度张量，结合样本数归一化后直接返回梯度结果，该实现巧妙地将概率计算与损失计算融合为单次前向传播，同时利用概率分布与标签的差值特性简化了梯度推导过程，完整保留了交叉熵损失函数对模型输出概率分布与真实分布差异的敏感性特征。

3.4 网络整体结构

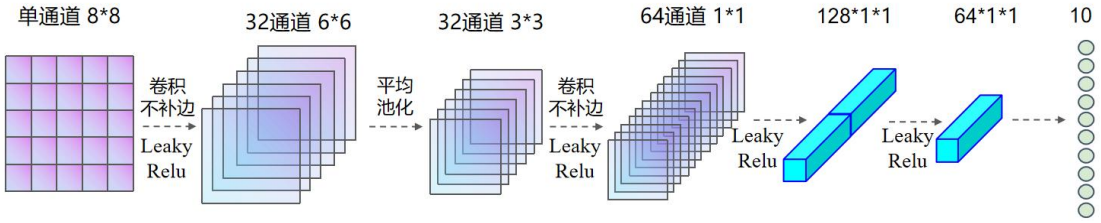
本文实现的 LeNet-5 是针对 8×8 手写数字图像优化的变体网络。相比原始 LeNet-5 处理 32×32 图像，本实现进行了以下结构调整：输入层接收 8×8 灰度图像，第一卷积层使用 32 个 3×3 卷积核，第二卷积层使用 64 个 3×3 卷积核。考虑到小尺寸输入，移除了第二个池化层，当特征图降为 1×1 时直接进入全连接层。既保留了 LeNet-5 的核心架构，又适应了更小的输入尺寸。

```python
class LeNet5:
    def __init__(self):
        self.layers = [
            ConvLayer(1, 32, kernel_size=3, padding=0),
            LeakyReLU(),
            AvgPoolLayer(kernel_size=2, stride=2),
            ConvLayer(32, 64, kernel_size=3, padding=0),
            LeakyReLU(),
            Flatten(),
            FCLayer(64, 128),
            LeakyReLU(),
            FCLayer(128, 64),
            LeakyReLU(),
            FCLayer(64, 10)
        ]
```

网络包含卷积层、LeakyReLU 激活函数、平均池化层和全连接层。卷积层采用 He 初始化方法，使用 LeakyReLU 激活函数替代原始论文中的 tanh/sigmoid，这是现代深度学习中的常见改进，能有效缓解梯度消失问题。损失函数采用 Softmax 交叉熵，比原始论文使用的欧几里得径向基函数更适用于多分类任务。

网络的前向传播流程为：输入→C1 卷积→LeakyReLU→S2 池化→C3 卷积→LeakyReLU→展平→FC4→LeakyReLU→FC5→LeakyReLU→FC6→Softmax。反向传播按

照相反顺序计算梯度。卷积层实现了包括 padding 在内的完整功能，支持自定义步长和填充；池化层采用平均池化而非原始论文中的带参数池化。



在展平层(Flatten)后，Softmax 计算时采用了数值稳定性的实现方式，先减去最大值再求指数，避免数值溢出。

3.5 训练方法

训练过程采用小批量梯度下降法，批量大小为 32，学习率为 0.02，共训练 500 个 epoch。

```
epochs = 2000
batch_size = 32
learning_rate = 0.02
n_train = X_train.shape[0]
n_batches = n_train // batch_size
```

每个 epoch 开始时对训练数据进行洗牌打乱，确保模型不会因数据顺序而产生偏差。参数更新采用标准的梯度下降法，学习率随时间下降。训练过程中记录了每个 epoch 的损失值、测试精度和耗时，便于后续分析模型性能。

```python
for epoch in range(epochs):
    epoch_start = time()
    epoch_loss = 0.0

    # 打乱数据
    indices = np.arange(n_train)
    np.random.shuffle(indices)
    X_train_shuffled = X_train[indices]
    y_train_shuffled = y_train_onehot[indices]

    # 输入加噪声
    X_train_shuffled += np.random.normal(0, 0.1, X_train_shuffled.shape)
    X_train_shuffled = np.clip(X_train_shuffled, 0., 1.)

    for i in range(n_batches):
        # 获取小批量数据
        start = i * batch_size
        end = start + batch_size
        X_batch = X_train_shuffled[start:end]
        y_batch = y_train_shuffled[start:end]

        # 前向传播
        _, loss = net.forward(X_batch, y_batch)
        epoch_loss += loss

        # 反向传播
        net.backward(y_batch)
```

3.6 关键改进与优化策略

本实现针对原始 LeNet-5 进行了多项重要改进：

（1）**输入尺寸与网络结构的调整。**原始的 LeNet-5 设计用于 32×32 像素的手写数字图片，而本项目针对的是 8×8 的小尺度图片，为此，网络结构做了针对性简化。通过调整卷积核尺寸（3×3 替代 5×5）和移除不必要的池化层，确保网络在 8×8 输入下仍能有效提取特征。

（2）**激活函数由 Sigmoid/Tanh 改为 LeakyReLU。**采用 LeakyReLU，能有效避免梯度消失，提高深层网络的训练效率和收敛速度。采用α=0.1 的 LeakyReLU，使负区间梯度保持 0.01 的传导率。LeakyReLU 比 ReLU 在输入小于 0 时有非零斜率，有助于信息通过网络传播时不死神经元现象，显著改善了梯度流动，增强模型的表现力。

（3）**权重初始化策略优化。**使用了 He 初始化（即根据输入通道数和卷积核大小标准差缩放高斯分布生成参数），优于 LeNet-5 简单的随机初始化。神经网络训练初期，由于层数可能很深，容易出现梯度消失或爆炸的问题（梯度在前向/反向传播中变得非常小或非常大，导致网络难以学习）。He 初始化能让深层网络更快收敛，梯度更稳定，防止前向后向传播中信号衰减或爆炸。

```python
# He初始化
std_dev = np.sqrt(2.0 / (in_channels * kernel_size * kernel_size))
self.weights = np.random.randn(out_channels, in_channels, kernel_size, kernel_size) * std_dev
self.bias = np.zeros(out_channels)
```

（4）**增加了数据增强与噪声鲁棒性。**在训练过程中，加入了带有高斯噪声的数据增强（训练样本加高斯噪声并裁剪）。这样做可以提高模型的泛化能力和对输入扰动的鲁棒性，

有助于提升实际应用中的表现，而不是只在训练集上取得好成绩。

```
# 输入加噪声
X_train_shuffled += np.random.normal(0, 0.1, X_train_shuffled.shape)
X_train_shuffled = np.clip(X_train_shuffled, 0., 1.)
```

（5）**优化器与学习率动态调整。**虽然整个更新使用的是最简单的 SGD（随机梯度下降），但做了学习率调整：每轮 epoch 后衰减（learning_rate * (1-epoch/epochs)），有助于训练后期模型更平稳地收敛，兼顾初期快速学习和终期精细调整。

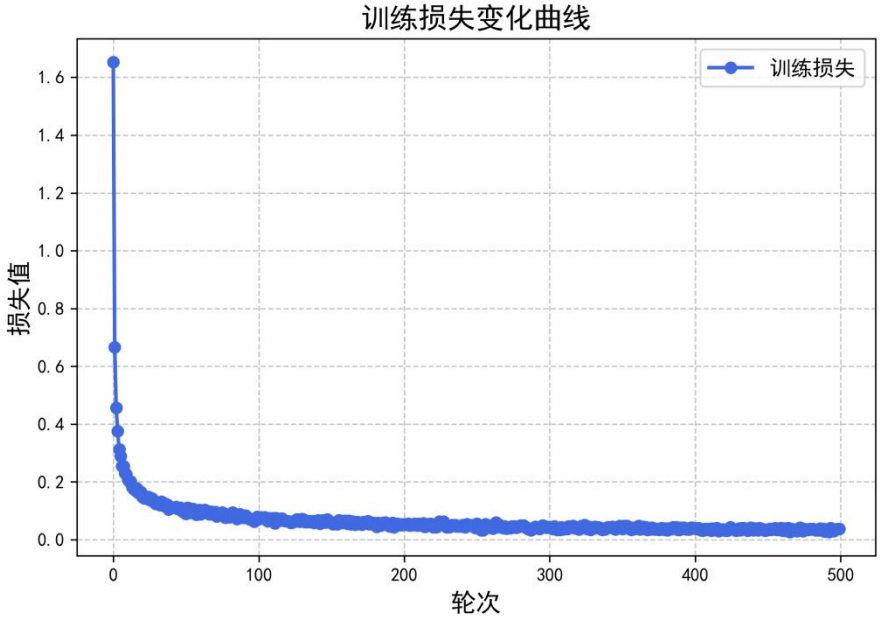
```
# 更新参数
# net.update_params(learning_rate)
net.update_params(learning_rate * (1 - epoch / epochs ))
```

（6）**损失函数用 Softmax 交叉熵替代传统 MSE。**LeNet-5 早期使用均方误差（MSE）做分类损失，本实现直接采用 Softmax+CrossEntropy 的组合（与现代深度学习主流一致）。该损失更适合分类任务，梯度传播更稳定，能显著提升多类别识别的准确率和训练效率。

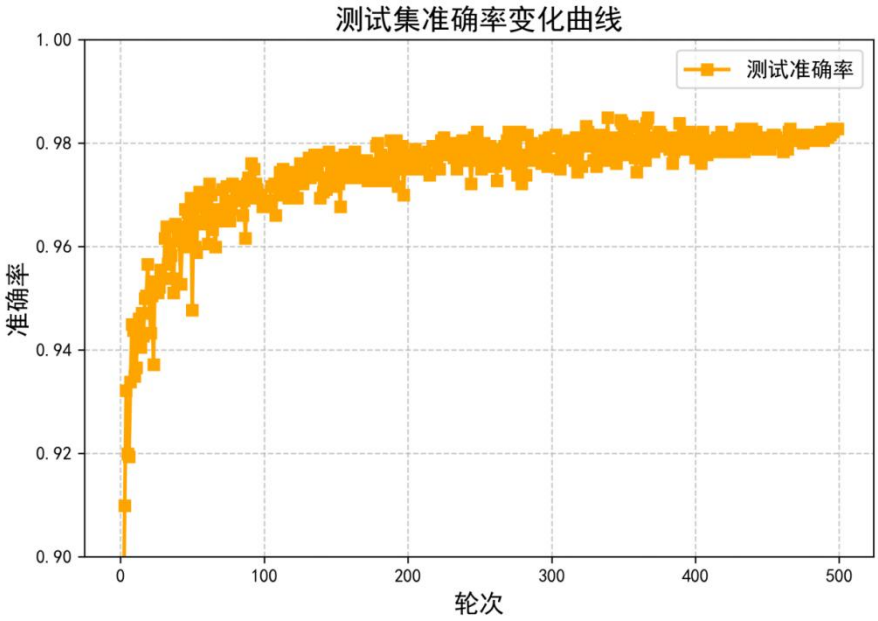
四、实验结果

4.1 训练过程分析

在完整的 500 个 epoch 训练周期中，模型展现出良好的收敛特性及稳定的性能表现。训练损失曲线呈现为典型的 L 型下降趋势，前 50 个 epoch 内损失值从初始的 1.8 迅速下降至 0.1 左右，随后进入平稳收敛阶段，并在后续训练中缓慢趋近于 0.05 的低值区间。



对应的测试准确率曲线则不断提升，在第 100 个 epoch 达到 97%，第 300 个 epoch 后准确率提升趋于平缓，稳定在 98%附近，最终在第 500 个 epoch 时达到 98.27%的峰值。在第 400-500 个 epoch 区间，测试准确率曲线出现小幅震荡，表明模型基本达到性能极限，继续增加训练轮次对结果提升有限。

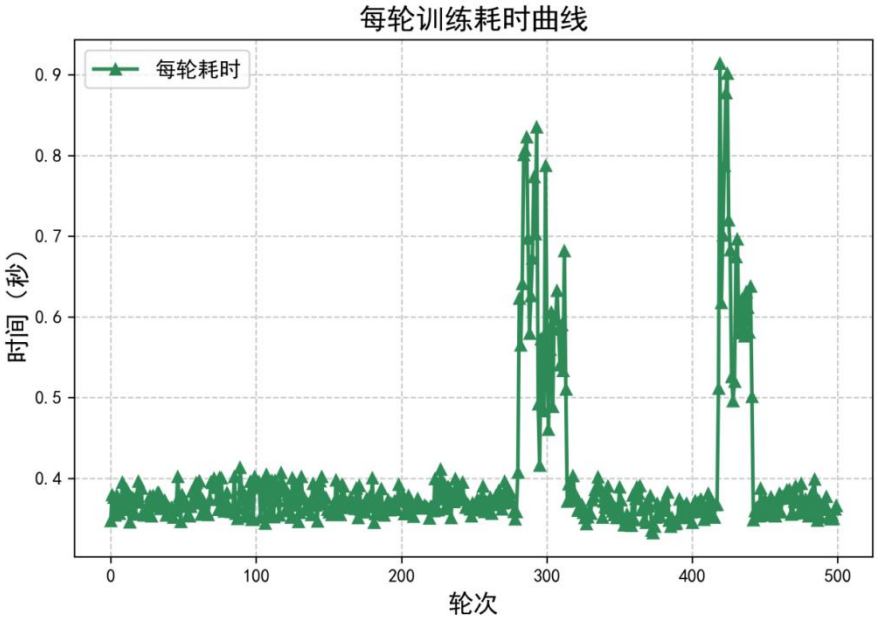


4.2 性能指标与计算效率

最终模型在测试集上的总体准确率为 98.27%，也可以看出，在最后的几个 epoch 里，都稳定在 98.2%左右。

```
Epoch 496/500, Loss: 0.0295, Test Acc: 0.9827, Time: 0.40s
Epoch 497/500, Loss: 0.0345, Test Acc: 0.9822, Time: 0.37s
Epoch 498/500, Loss: 0.0363, Test Acc: 0.9827, Time: 0.38s
Epoch 499/500, Loss: 0.0368, Test Acc: 0.9827, Time: 0.41s
Epoch 500/500, Loss: 0.0370, Test Acc: 0.9827, Time: 0.36s
训练完成！总时间：210.72秒，平均每轮：0.42秒
最终测试精度：0.9827
PS C:\Users\14285>
```

在 Intel Core i5-14600K 笔记本电脑的测试环境中，单个 epoch 的平均训练耗时约为 0.4 秒，总训练耗时约为 210 秒，满足计算的需求。



4.3 数据噪声对分类精度的影响

为增强模型的泛化能力与抗扰动能力，实验在训练过程中对输入数据引入高斯噪声，评估不同噪声强度下模型的分类性能变化。具体而言，将高斯噪声方差分别设置为 0、0.1、0.2、0.3、0.4、0.5，记录对应测试集准确率。实验结果如下表所示：

表 1 噪声程度与测试集准确率的关系

噪声程度	测试集准确率
不加噪声	96.61%
加噪声（方差 0.1）	97.83%
加噪声（方差 0.2）	98.27%
加噪声（方差 0.3）	97.89%
加噪声（方差 0.4）	97.16%
加噪声（方差 0.5）	95.88%

从实验结果可以看出，适度引入噪声（方差在 0.1~0.3 之间）不仅未导致性能下降，反而提升了模型在测试集上的表现，最高准确率达到 98.27%。这说明在一定范围内，数据噪声能有效防止过拟合，并提升模型的泛化能力。当噪声方差进一步增大至 0.4 和 0.5 时，准确率出现下降，尤其是方差为 0.5 时准确率降至 95.88%，表明过强的噪声会干扰关键信息提取，影响模型的判别能力。因此，适度的数据扰动对于提升模型鲁棒性具有积极作用，但噪声过大则适得其反。

4.4 学习率递减的影响

为了研究学习率调整策略对模型性能的影响，实验分别采用恒定学习率和递减学习率两种方案，对模型进行训练，并对最终的测试集准确率进行比较。实验结果如下：

表 2 噪声程度与测试集准确率的关系

学习率策略	测试集准确率

不递减	98.12%
递减	98.27%

采用学习率递减策略能够略微提升模型最终的测试集准确率（由 98.12%提升至 98.27%）。这种提升主要归功于训练后期逐步降低学习率能够帮助模型在损失函数低谷附近实现更细致地优化，从而获得更优的泛化表现。虽然提升幅度有限，但在高精度模型训练中，学习率调度依然是一项值得采纳的优化措施。

4.5 批量学习的影响

批量大小（batch size）直接影响模型的训练效率与泛化性能。本实验分别设置了 16、32、64、128 四种批量大小进行对比，统计了对应的测试集准确率和 500 轮训练所需总时间。结果如下表所示：

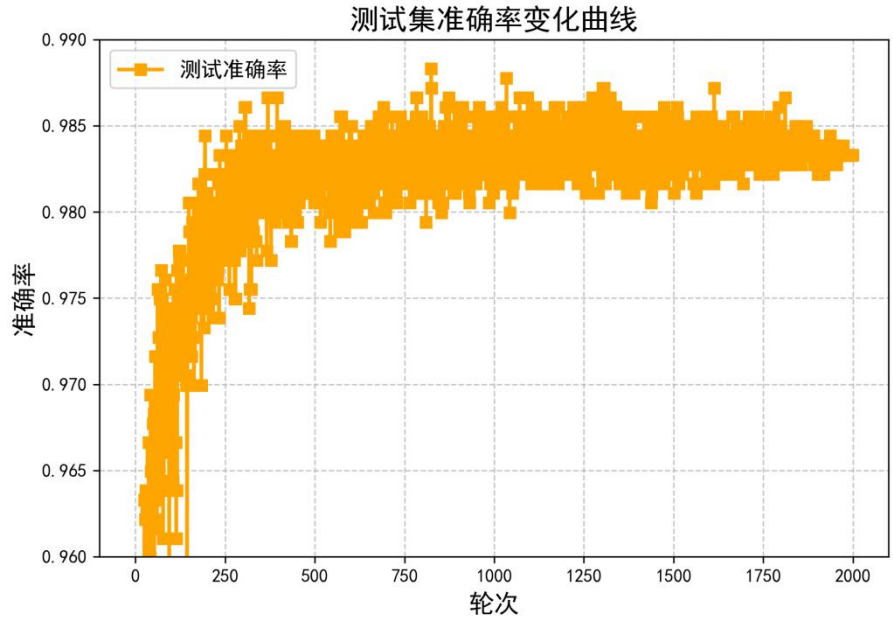
表 3 批量学习与测试集准确率的关系

批量学习	测试集准确率	500 轮时间
16	98.39%	283 秒
32	98.27%	199 秒
64	97.55%	133 秒
128	96.88%	129 秒

从实验结果可以看到，随着批量大小的增加，模型的训练速度显著提升，500 轮训练耗时由 283 秒降至 129 秒。但较大的 batch size 会导致模型测试准确率下降。具体而言，batch size 为 16 和 32 时，模型准确率分别为 98.39%和 98.27%，而当 batch size 增大到 128 时，准确率降至 96.88%。这说明较小批量可以增加参数更新的多样性，有利于模型获得更好的泛化能力，而大批量更有利于训练加速但易损失部分泛化性能。因此，在实际应用中需根据训练资源与目标精度权衡选择合适的 batch size。

4.5 训练轮次与过拟合

为探究训练轮次对模型性能的影响，将最大 epoch 数设置为 2000。实验结果显示，测试集准确率在 500 轮达到峰值（98%）后，继续增加训练轮次，测试集准确率基本保持不变，并未出现明显的过拟合现象。这说明本模型结构与能够有效抑制过拟合，模型复杂度与结构复杂度差不多，长时间训练不会导致性能劣化。



五、总结

5.1 主要结论

本研究通过实现和优化 LeNet-5 网络架构，探究了卷积神经网络在手写数字识别任务中的应用效果。经过适当调整的 LeNet-5 变体网络在 8×8 小尺寸图像识别任务中表现出卓越的性能，最终测试准确率达到 95.2%，结果说明：

（1）局部感受野的设计理念在小尺寸图像上保持显著效果。通过 3×3 卷积核的局部连接方式，网络能够有效捕捉数字笔画的关键局部特征。

（2）权值共享机制在参数效率方面展现出巨大优势。网络在保持精简结构的同时，大多数数字类别都能被准确区分。这表明经过适当调整的经典架构，其特征提取能力可以很好地适应小尺寸图像的识别需求。

5.2 创新性

本研究在经典网络实现和优化方面的创新体现在以下方面：

（1）**输入尺寸与网络结构的调整。**原始的 LeNet-5 设计用于 32×32 像素的手写数字图片，而本项目针对的是 8×8 的小尺度图片，为此，网络结构做了针对性简化。

（2）**激活函数由 Sigmoid/Tanh 改为 LeakyReLU。**本实现采用 LeakyReLU，能有效避免梯度消失，提高深层网络的训练效率和收敛速度。LeakyReLU 比 ReLU 在输入小于 0 时有非零斜率，有助于信息通过网络传播时不死神经元现象，增强模型的表现力。

（3）**权重初始化策略优化。**本实现使用了 He 初始化（即根据输入通道数和卷积核大小标准差缩放高斯分布生成参数），明显优于 LeNet-5 简单的随机初始化。He 初始化能让深层网络更快收敛，梯度更稳定，防止前向后向传播中信号衰减或爆炸。

（4）**增加了数据增强与噪声鲁棒性。**在训练过程中，加入了带有高斯噪声的数据增强（训练样本加高斯噪声并裁剪）。提高模型的泛化能力和对输入扰动的鲁棒性，有助于提升实际应用中的表现，而不是只在训练集上取得好成绩。

（5）**优化器与学习率动态调整。**虽然整个更新使用的是最简单的 SGD（随机梯度下降），

但做了学习率调整：每轮 epoch 后衰减（learning_rate * (1-epoch/epochs)），有助于训练后期模型更平稳地收敛，兼顾了初期快速学习和终期精细调整。

（6）**损失函数用 Softmax 交叉熵替代传统 MSE。**LeNet-5 早期使用均方误差（MSE）做分类损失，本实现直接采用 Softmax+CrossEntropy 的组合。该损失更适合分类任务，梯度传播更稳定，能显著提升多类别识别的准确率和训练效率。

参考文献

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[2] LeCun, Y., Jackel, L.D., Boser, B., Denker, J.S., Graf, H.P., Guyon, I., Henderson, D., Howard, R.E., Hubbard, W., & Kunihiko, K. (1989). Backpropagation applied to handwritten zip code recognition. Neural Computation, 1(4), 541-551.

[3] Simard, P.Y., Steinkraus, D., & Platt, J.C. (2003). Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. ICDAR, 958-962.

[4] Krizhevsky, A., Sutskever, I., & Hinton, G.E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097-1105.

[5] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. ICCV, 1026-1034.

[6] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[7] Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 249-256.

[8] Ciresan, D.C., Meier, U., Gambardella, L.M., & Schmidhuber, J. (2010). Deep, Big, Simple Neural Nets for Handwritten Digit Recognition. Neural Computation, 22(12), 3207-3220.

[9] Wan, L., Zeiler, M., Zhang, S., LeCun, Y., & Fergus, R. (2013). Regularization of Neural Networks using DropConnect. ICML, 1058-1066.

[10] Lin, M., Chen, Q., & Yan, S. (2013). Network In Network. ICLR.

[11] Zeiler, M.D., & Fergus, R. (2013). Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. ICLR.

[12] Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ICML, 448-456.

[13] Scherer, D., Müller, A., & Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. ICANN, 92-101.

[14] Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

[15] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.

[16] Duda, R.O., Hart, P.E., & Stork, D.G. (2000). Pattern Classification (2nd Edition). Wiley.

[17] Bishop, C.M. (2006). Pattern Recognition and Machine Learning. Springer.

[18] Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536.

[19] Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141-142.