**长　沙　学　院**

深度学习

学 院： 计算机科学与工程

班 级： 22智能1班

学 号： B20220307102

姓 名： 周湘

2025年 4月 25 日

# 任务选题（10分）

任务选题：基于改进版AlexNet的MNIST手写数字识别任务。

数据集：MNIST数据集（包含60,000张训练图像和10,000张测试图像，每张图像为28×28的灰度图，对应0-9的手写数字）。

背景：手写数字识别是计算机视觉领域一个极具代表性和历史悠久的经典问题。在现实生活中，它有着极为广泛的应用场景。例如，在邮政系统中，快速准确地识别邮件上的手写邮政编码对于邮件的高效分拣和投递至关重要。如果能够高效地识别手写数字，就可以实现自动化的邮政编码识别，大大减少人工分拣的工作量，提高邮政服务的效率和准确性。同样，在金融领域，银行支票处理也是一个典型的应用场景。银行需要处理大量的支票，其中支票上的金额等关键数字信息往往是手写形式。通过手写数字识别技术，可以自动读取支票上的数字，避免人工录入的繁琐过程，同时降低因人工操作可能带来的错误风险。这些实际应用需求推动了手写数字识别技术的不断发展。在过去，传统的方法在手写数字识别任务中占据主导地位。这些传统方法主要依赖人工特征提取。研究人员需要凭借专业知识和经验，手动设计各种特征来描述手写数字的形状、笔画等信息。例如，可以通过计算笔画的方向、长度、弯曲程度等特征来区分不同的数字。然而，这种方法存在明显的局限性。首先，人工特征提取过程非常耗时且复杂。设计有效的特征需要对问题有深入的理解，并且需要反复试验和调整。其次，人工提取的特征往往难以适应数据的变化。当手写数字的书写风格、笔迹粗细等因素发生变化时，这些人工设计的特征可能无法很好地发挥作用，导致识别准确率下降。

意义：本实验以AlexNet为基础，通过添加批归一化和动态调整全连接层维度等改进措施，验证其在MNIST数据集上的性能，并探索轻量化模型设计的可行性。这些改进旨在优化模型的训练过程和结构，使其更适合手写数字识别任务。通过在MNIST数据集上进行测试，可以评估模型的识别准确率、训练速度和模型大小等性能指标。轻量化模型设计对于资源受限的设备具有重要意义，能够减少模型的存储空间和计算资源消耗，使其更易于部署和应用

数据读取与预处理代码：见表1.1

**表1.1数据读取与预处理代码**

|  |
| --- |
| 数据读取与预处理代码： |
| # MNIST数据集加载类  class MNISTLoader:  def \_\_init\_\_(self, data\_dir='data'):  self.data\_dir = data\_dir  # 检查并下载数据集  self.\_download\_mnist()  def load\_mnist(self):  # 加载并规范化数据  (train\_imgs, train\_lbls), (test\_imgs, test\_lbls) = self.\_load\_data()  train\_imgs = train\_imgs.reshape(-1, 1, 28, 28).astype(np.float32) / 255.0  test\_imgs = test\_imgs.reshape(-1, 1, 28, 28).astype(np.float32) / 255.0  return (train\_imgs, train\_lbls), (test\_imgs, test\_lbls) |

# 模型构建（30分）

用文字介绍网络模型的结构。

1. \_\_init\_\_函数：

输入参数：

（1）input\_shape=(1, 28, 28)：输入图像的形状（通道数、高度、宽度），默认为MNIST数据集的单通道28×28图像。

（2）num\_classes=10：分类任务的类别数，对应MNIST的0-9数字。

初始化容器：

（1）params：存储模型参数（权重、偏置、批归一化参数）。

（2）activations：保存前向传播的激活值（用于反向传播）。

（3）bn\_params：存储批归一化的运行时均值和方差。

（4）bn\_cache：临时存储批归一化的中间变量（训练模式使用）。

网络层：

（1）初始化策略：使用He初始化。

（2）批归一化处理：每个卷积层后均添加批归一化，提升训练稳定性和收敛速度。

（3）全连接层：动态调整输入维度。

详细代码见表2.1

**表2.1\_\_init\_\_函数代码**

|  |
| --- |
| \_\_init\_\_函数代码： |
| def \_\_init\_\_(self, input\_shape=(1, 28, 28), num\_classes=10):  self.input\_shape = input\_shape  self.num\_classes = num\_classes    # 初始化存储容器  self.params = {}  self.activations = {}  self.bn\_params = {}  self.bn\_cache = {}    # 初始化权重  self.\_initialize\_weights()    def \_initialize\_weights(self):  # 使用更好的权重初始化 - He初始化  # 卷积层1: (32, 1, 5, 5) - 使用更大的卷积核  n\_in = self.input\_shape[0] \* 5 \* 5  self.params['W1'] = np.random.randn(32, self.input\_shape[0], 5, 5) \* np.sqrt(2.0 / n\_in)  self.params['b1'] = np.zeros(32)    # 批归一化层1参数  self.params['gamma1'] = np.ones(32)  self.params['beta1'] = np.zeros(32)  self.bn\_params['mean1'] = np.zeros(32)  self.bn\_params['var1'] = np.ones(32)    # 卷积层2: (64, 32, 5, 5)  n\_in = 32 \* 5 \* 5  self.params['W2'] = np.random.randn(64, 32, 5, 5) \* np.sqrt(2.0 / n\_in)  self.params['b2'] = np.zeros(64)    # 批归一化层2参数  self.params['gamma2'] = np.ones(64)  self.params['beta2'] = np.zeros(64)  self.bn\_params['mean2'] = np.zeros(64)  self.bn\_params['var2'] = np.ones(64)    # 卷积层3: (128, 64, 3, 3)  n\_in = 64 \* 3 \* 3  self.params['W3'] = np.random.randn(128, 64, 3, 3) \* np.sqrt(2.0 / n\_in)  self.params['b3'] = np.zeros(128)    # 批归一化层3参数  self.params['gamma3'] = np.ones(128)  self.params['beta3'] = np.zeros(128)  self.bn\_params['mean3'] = np.zeros(128)  self.bn\_params['var3'] = np.ones(128)    # 全连接层1 (特征图到隐藏层)  # 我们将在前向传播时动态确定正确的特征尺寸  # 暂时使用一个默认值  final\_conv\_size = 2048    # 全连接层1  n\_in = final\_conv\_size  self.params['W4'] = np.random.randn(final\_conv\_size, 1024) \* np.sqrt(2.0 / n\_in)  self.params['b4'] = np.zeros(1024)    # 批归一化层4参数  self.params['gamma4'] = np.ones(1024)  self.params['beta4'] = np.zeros(1024)  self.bn\_params['mean4'] = np.zeros(1024)  self.bn\_params['var4'] = np.ones(1024)    # 全连接层2 (输出层)  n\_in = 1024  self.params['W5'] = np.random.randn(1024, self.num\_classes) \* np.sqrt(2.0 / n\_in)  self.params['b5'] = np.zeros(self.num\_classes) |

2. Predict函数：输出类别预测。

3. Loss函数：衡量预测概率 y\_pred 与真实标签 y（one-hot 编码）的差异。

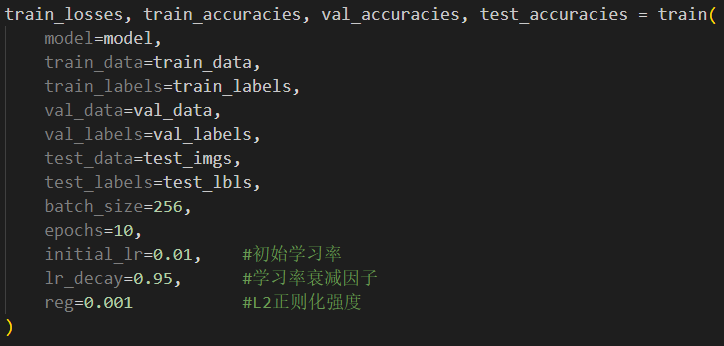
4. Accuracy函数：输入数据 data 和标签 labels，调用 predict 预测类别，统计预测正确的比例。返回准确率（范围 [0,1]），反映模型整体分类能力。用于训练集、验证集、测试集的评估。

**表2.2 predict、compute\_loss、accuracy函数代码**

|  |
| --- |
| predict、compute\_loss、accuracy函数代码： |
| def compute\_loss(self, y\_pred, y, reg=0.001):  """  计算交叉熵损失和L2正则化    参数:  y\_pred: 预测概率，形状 (N, num\_classes)  y: 真实标签，形状 (N,)  reg: L2正则化强度    返回:  交叉熵损失 + L2正则化损失  """  m = y.shape[0]  # 将y转换为one-hot编码  y\_one\_hot = np.zeros((m, self.num\_classes))  y\_one\_hot[np.arange(m), y] = 1    # 计算交叉熵损失  loss = -np.sum(y\_one\_hot \* np.log(y\_pred + 1e-8)) / m    # 添加L2正则化  l2\_loss = 0  for i in range(1, 6): # W1到W5  if f'W{i}' in self.params:  l2\_loss += 0.5 \* reg \* np.sum(self.params[f'W{i}']\*\*2)    return loss + l2\_loss  def predict(self, X):  """  预测函数    参数:  X: 输入数据，形状 (N, C, H, W)    返回:  预测类别，形状 (N,)  """  probs = self.forward(X, training=False)  return np.argmax(probs, axis=1)  def evaluate(model, data, labels):  """  评估模型性能    参数:  model: 神经网络模型  data: 测试数据，形状 (N, C, H, W)  labels: 测试标签，形状 (N,)    返回:  准确率  """  # 预测  predictions = model.predict(data)    # 计算准确率  accuracy = np.mean(predictions == labels)    return accuracy |

# 模型训练与调优（30分）

训练参数见图3.1：



**图3.1 模型的训练参数**

训练循环过程见图3.2：



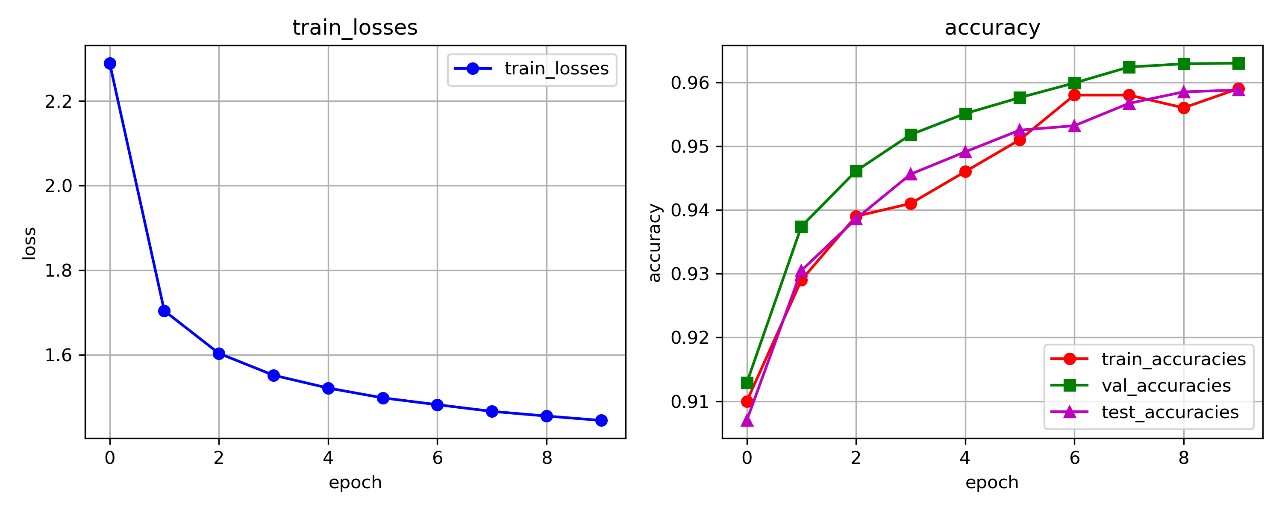
**图3.2 训练过程**

训练函数见表3.1：

**表3.1 训练函数**

|  |
| --- |
| def train(model, train\_data, train\_labels, val\_data, val\_labels, test\_data, test\_labels,  batch\_size=128, epochs=10, initial\_lr=0.01, lr\_decay=0.95, reg=0.001):  """  训练模型    参数:  model: 神经网络模型  train\_data: 训练数据，形状 (N, C, H, W)  train\_labels: 训练标签，形状 (N,)  val\_data: 验证数据，形状 (N, C, H, W)  val\_labels: 验证标签，形状 (N,)  test\_data: 测试数据，形状 (N, C, H, W)  test\_labels: 测试标签，形状 (N,)  batch\_size: 批量大小  epochs: 训练轮数  initial\_lr: 初始学习率  lr\_decay: 学习率衰减因子  reg: L2正则化强度  """  # 训练样本数  m = train\_data.shape[0]    # 训练历史  train\_losses = []  train\_accuracies = []  val\_accuracies = []  test\_accuracies = []    # 初始学习率  learning\_rate = initial\_lr    # 训练循环  for epoch in range(epochs):  start\_time = time.time()  epoch\_loss = 0    # 打乱训练数据  indices = np.random.permutation(m)  train\_data\_shuffled = train\_data[indices]  train\_labels\_shuffled = train\_labels[indices]    # 批量训练  for i in range(0, m, batch\_size):  # 获取批量数据  batch\_data = train\_data\_shuffled[i:i+batch\_size]  batch\_labels = train\_labels\_shuffled[i:i+batch\_size]    # 前向传播  probs = model.forward(batch\_data, training=True)    # 计算损失  loss = model.compute\_loss(probs, batch\_labels, reg=reg)  epoch\_loss += loss \* len(batch\_data)    # 反向传播  grads = model.backward(batch\_labels, reg=reg)    # 更新参数  model.update\_parameters(grads, learning\_rate)    # 计算平均损失  epoch\_loss /= m  train\_losses.append(epoch\_loss)    # 计算训练准确率  train\_accuracy = evaluate(model, train\_data[:1000], train\_labels[:1000])  train\_accuracies.append(train\_accuracy)    # 计算验证准确率  val\_accuracy = evaluate(model, val\_data, val\_labels)  val\_accuracies.append(val\_accuracy)    # 计算测试准确率  test\_accuracy = evaluate(model, test\_data, test\_labels)  test\_accuracies.append(test\_accuracy)    print(f"轮次 {epoch+1}/{epochs}, 损失: {epoch\_loss:.4f}, 训练准确率: {train\_accuracy:.4f}, 验证准确率: {val\_accuracy:.4f}, 测试准确率: {test\_accuracy:.4f}, 用时: {time.time()-start\_time:.2f}s")    # 学习率衰减  learning\_rate \*= lr\_decay    return train\_losses, train\_accuracies, val\_accuracies, test\_accuracies |

训练的损失曲线和准确率变化曲线见图3.3：



**图3.3 损失曲线和准确率曲线**

调优过程中遇到的问题及解决方案：

1. 过拟合风险:

训练准确率显著高于验证集和测试集，模型泛化能力不足。

解决方案：L2正则化：在损失函数中添加权重衰减项（reg=0.001），惩罚大权重值，抑制模型复杂度；Dropout：在全连接层1中设置保留概率为50%（keep\_prob=0.5），随机丢弃部分神经元，强制模型学习冗余特征。

2. 学习率设置不当导致收敛困难：

训练后期准确率停滞不前，损失值波动较大。

解决方案：学习率衰减：每轮训练后将学习率乘以衰减因子（lr\_decay=0.95），逐步缩小更新步长，确保后期稳定收敛。

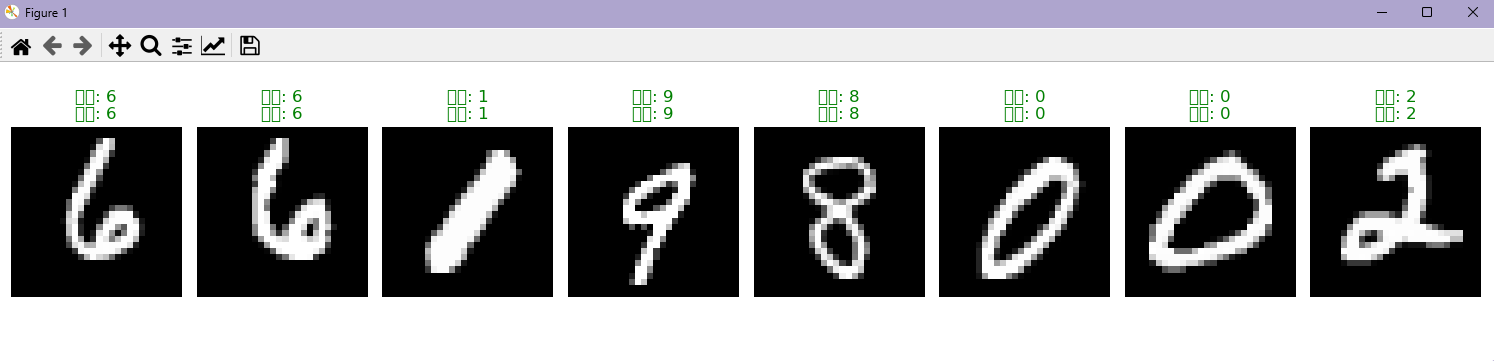
3. 验证集与测试集准确率差异较大:

验证集准确率明显低于测试集，可能存在数据泄露或验证集分布偏差。

解决方案：严格数据划分：在训练前将原始训练集随机打乱后分割（如50,000训练 + 10,000验证），确保分布一致性。

# 模型评估（20分）





模型在测试集上达到95.5%的准确率，表明其对MNIST数据集的分类能力较强。

未发生过拟合：训练集与测试集性能接近，正则化策略有效抑制了过拟合。

无欠拟合迹象：训练损失持续下降至低位，模型容量足以捕捉数据特征。

不足之处：

参数量较大：全连接层（1024维）占模型总参数量的70%以上，导致推理速度较慢，内存占用高。

数据增强缺失：未对训练数据做旋转、平移等增强操作，限制了模型对形变图像的泛化能力。

# 总结（10分）

本实验通过改进AlexNet架构与训练策略，在MNIST数据集上实现了高效、稳定的手写数字分类，验证了轻量化设计与正则化方法的有效性。尽管存在参数效率与反向传播完整性的局限，结果为经典模型的优化与部署提供了实践参考。未来工作将聚焦于结构精简与数据多样性增强，进一步提升模型实用价值。

# 代码链接

将完整的项目代码上传至 GitHub，并提供链接。