《人工智能基础》第一次作业实验报告

姓名： 苏祎成
学号： 23307090051
日期： 2025 年 11 月 11 日

---

1 摘要

本实验基于 PyTorch 框架，对 CIFAR-10 数据集上卷积神经网络（CNN）的训练、优化与改进方法展开系统性探索。以经典 LeNet 网络为基准，逐步引入并验证现代深度学习技术，包括正则化方法、超参数调优及先进网络架构设计。

实验探索围绕四个关键步骤推进：（1）实现训练过程可视化，通过追踪损失函数变化捕捉并分析模型的学习动态；（2）引入 L2 正则化与 Dropout 技术，对比二者在抑制过拟合方面的实际效果；（3）系统性调整学习率、训练轮数、动量等核心超参数，揭示其对模型最终性能与训练效率的影响规律；（4）设计并实现具有现代特征的轻量级残差网络（ResNet），与 LeNet 进行全方位性能对比。

实验结果表明：（1）两种正则化策略未能显著提升模型泛化能力，其原因可能在于模型基础设定；（2）学习率是影响模型性能的关键因素，需重点优化；（3）现代 CNN 架构（如本次实现的 ResNet）相较于 LeNet，测试准确率提升约 30%，且在实验配置下未显著增加训练时间成本。

此次实验显著深化了我对卷积神经网络内在运作机制的理解。

---

2 引言

图像分类作为计算机视觉领域的基础性任务，核心目标是使机器具备类人视觉理解能力。随着深度学习的发展，卷积神经网络（CNN）已成为解决此类问题的主流方法，广泛应用于图像识别、目标检测及图像生成等领域。从 LeNet、AlexNet 到 VGG，经典网络的发展不仅推动图像识别性能持续突破，更为后续模型设计与训练策略提供了重要参考。

为确保实验的基准性与可操作性，本研究选用 CIFAR-10 数据集作为实验对象。该数据集是计算机视觉领域常用基准数据集，包含 10 个类别的 60000 张 32×32 彩色图像（训练集 50000 张、测试集 10000 张），涵盖飞机、汽车、鸟类、猫狗等日常事物；其数据规模适中，适合深入分析与评估中小型 CNN 模型的性能。

本实验基于 PyTorch 框架开展，首先以杨立昆（Yann LeCun）的开山之作—— LeNet 网络为基础，构建并训练首个 CIFAR-10 分类模型。针对神经网络训练中普遍面临的过拟合问题，本文引入 L2 正则化（权重衰减）与 Dropout 两种主流正则化策略，通过实验量化二者在提升模型泛化性能上的实际作用。**需要重点指出的是，本实验并未将这一量化过程中 task2 中呈现，而是将正则化项看作连续变化的超参数，与第三部分超参数调优放置在一起讨论；换言之，本实验的基准模型已经引入了两种正则化项，并在基础上绘制损失函数图像。**

事实上，模型性能不仅取决于网络结构，超参数调优亦是关键环节。本研究的核心部分之一，是深入探索超参数对模型的影响：通过调节正则化强度、学习率、权重衰减系数、训练轮数等变量，细致观察其对训练损失曲线变化趋势、模型最终性能及收敛速度的作用规律。**本文在此部分同样进行了细致的可视化呈现，可以看作是对任务要求中 task1&2 部分的细化和深入呈现。**

为突破经典架构的局限，本研究进一步将目光投向现代网络结构，设计并实现更深层的

CNN 架构 —— 轻量化 ResNet，引入其标志性的残差连接机制。通过对比分析训练速度、测试准确率、过拟合程度等指标，具体衡量现代架构相较于 LeNet 的性能优势与技术改进点。

贯穿本实验的核心目标包括下述几点：（1）深度学习模型训练过程中的真实动态特征；（2）正则化技术抑制过拟合的作用机制；（3）超参数对模型性能的调控规律；（4）现代 CNN 架构相较于经典架构的改进原理与实际成效。

最后需要说明的是，为了更好地呈现板块之间相对独立的代码逻辑，并尽量避免单个代码文件及其输出结果过于冗长不方便阅读，本实验将 task1 &2、task3 以及 task4 整合为三份源代码分别提交，力图清晰展现我对 CNN 训练与评估的完整理解。

---

3 实验方法

3.1 Task1：绘制损失函数曲线

损失函数的演化特征是量化模型训练动态的核心指标，其可视化分析旨在建立参数更新方向、训练稳定性与收敛趋势间的关联，为模型优化提供直观依据。本实验通过设计数据记录、持久化与可视化流程，实现对神经网络训练过程的量化解析。

3.1.1 数据记录方案设计

为同时捕获训练过程的瞬时波动与长期收敛趋势，设计 loss_history 字典作为数据存储结构，包含三类核心字段：

- batch_losses：存储每批次（batch）训练的瞬时损失值，用于表征训练过程的微观动态，支撑训练稳定性评估；
- epoch_losses：通过计算单轮次（epoch）内 batch_losses 的均值得到，用于反映模型的长期收敛趋势，辅助判断收敛平台期或过拟合前兆；
- batch_indices：存储各批次的全局索引（如第 2 轮次第 3 批次索引为 “单轮次总批次数 + 3”），解决局部索引导致的曲线重叠问题，确保损失值与训练进程的精准映射。

3.1.2 训练循环中的数据记录实现

将数据记录逻辑嵌入训练循环，核心解决 PyTorch 张量内存冗余问题：

PyTorch 计算的损失值以张量（Tensor）形式存储，附带计算图信息以支持反向传播，直接存储会导致内存累积。采用 loss.item()方法提取张量中的 Python 标量值，在保留数值信息的同时剥离计算图数据，降低内存消耗。具体流程为：

1. 每批次训练中，模型完成前向传播后，与真实标签计算得到损失张量；
2. 调用 loss.item()提取标量损失，追加至 batch_losses；
3. 单轮次结束后，将该轮次 batch_losses 转换为 numpy 数组，通过 np.mean()计算平均损失并存入 epoch_losses；
4. 同步更新 batch_indices，记录当前累计训练批次数（初始值为 0，每完成 1 批次递增 1）。

3.1.3 可视化策略

采用双子图（subplots）架构，实现微观波动与宏观趋势的同步观测：

- 上方子图：以 batch_indices 为横轴、batch_losses 为纵轴，绘制批次级损失曲线；叠加 100 批次窗口的移动平均线，滤除随机噪声以凸显关键波动特征，支撑训练稳定性评估（如学习率不当导致的参数更新震荡）；
- 下方子图：以 epoch 序号为横轴、epoch_losses 为纵轴，绘制轮次级平均损失曲线，

用于分析长期收敛趋势，辅助判断收敛平台期或过拟合前期的损失异常。

关键代码结构

```
loss_history = {
    'batch_losses': [],
    'epoch_losses': [],
    'batch_indices': []
}

# 训练循环中记录
current_loss = loss.item()
loss_history['batch_losses'].append(current_loss)

# 保存数据
np.savez('loss_history.npz', **loss_history)

# 可视化
plt.plot(loss_history['batch_losses'])
```

3.2 Task2：加入正则化

3.2.1 问题重述

深度学习模型在训练过程中易因模型复杂度高于数据分布复杂度，出现“过拟合”现象——即模型对训练集噪声过度拟合，导致训练集性能优异但测试集泛化能力显著下降。其典型表现为：训练过程中训练损失持续下降，但测试损失下降至一定程度后开始回升。针对该问题，正则化技术通过“约束模型复杂度”或“引入随机性打破过度依赖”两类思路，提升模型泛化能力。本实验重点引入两种正则化策略：

1. L2 正则化（权重衰减）：通过对模型权重施加惩罚，限制权重幅值以降低模型复杂度；
2. Dropout：通过训练时随机丢弃部分神经元，打破神经元间的“共适应”依赖，增强模型稳健性。

3.2.2 实验思路

（1）L2 正则化实现

L2 正则化基于以下核心直觉：更简单的模型（权重分布更平缓）具有更强的泛化能力。其核心逻辑是在原始损失函数中引入“权重平方和惩罚项”，迫使模型权重倾向于较小值，避免因权重过大导致的函数拟合过度陡峭（即对输入微小变化敏感）。

$$L_{\{total\}} = L_{\{original\}} + \lambda \backslash sum_{\{i\}} w_i^2$$

在 PyTorch 框架中，L2 正则化无需手动修改损失函数，可通过优化器的 weight_decay 参数直接实现，本质是优化器在参数更新时自动对权重梯度进行调整：

```
optimizer = optim.SGD(net.parameters(),
                      lr=0.001,
                      momentum=0.9,
                      weight_decay=0.01)  # L2正则化系数
```

（2）Dropout 实现

Dropout 的提出源于对 “神经元共适应” 问题的解决 —— 训练过程中，神经元易过度依赖相邻神经元的输出特征，导致模型对特定神经元组合过度敏感（即 “过拟合”）。其核心逻辑是：训练时以概率 \(p\) 随机将部分神经元的输出设为 0，迫使模型学习 “不依赖特定神经元” 的鲁棒特征；同时为保证输出期望不变，对未被丢弃的神经元输出乘以 1-p（即“缩放补偿”）。

PyTorch 实现：在第一个全连接层（fc1）和第二个全连接层（fc2）之间插入 Dropout 层

```
class Net(nn.Module):
    def __init__(self, dropout_rate=0.5):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        # *** 新增：Dropout层 ***
        self.dropout = nn.Dropout(p=dropout_rate)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(x.size()[0], -1)
        x = F.relu(self.fc1(x))
        # *** 应用Dropout ***
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

3.2.3 两种正则化方法对比

特性	L2 正则化	Dropout
作用对象	权重参数	神经元激活
实现位置	优化器参数	网络结构
训练/测试差异	无差异	训练时激活，测试时关闭
典型取值	0.001 ~ 0.1	0.3 ~ 0.5
主要效果	权重平滑，防止过大	减少神经元共适应
计算开销	低	中等

3.3 Task3：调整超参数

3.3.1 问题重述

超参数的选择对模型性能有重要影响。本任务要求系统性地调节关键超参数（学习率、

训练轮数、正则化系数等），观察并分析其对训练损失和测试准确率的影响。

3.3.2 实验思路

实验设计： 采用控制变量法，设计 7 组对照实验：

1. 基准配置：lr=0.001, epochs=5, momentum=0.9, weight_decay=0.01, dropout=0.5
2. 高学习率：lr=0.01（10 倍增加）
3. 低学习率：lr=0.0001（0.1 倍）
4. 更多 epoch：epochs=10（2 倍）
5. 无正则化：weight_decay=0, dropout=0
6. 强正则化：weight_decay=0.1, dropout=0.7
7. 无动量：momentum=0

关键超参数分析：

1. 学习率（Learning Rate）
    - 影响：控制参数更新的步长，是最关键的超参数
    - 过大：训练不稳定，损失震荡，可能发散
    - 过小：收敛缓慢，需要更多训练时间
    - 最优范围：通常在 [0.0001, 0.01] 之间
2. 训练轮数（Epochs）
    - 影响：决定模型学习的充分程度
    - 过少：模型未充分学习，欠拟合
    - 过多：浪费计算资源，可能过拟合
    - 策略：使用 early stopping 或验证集监控
3. 正则化系数
    - 无正则化：容易过拟合，训练准确率高但测试准确率低
    - 适度正则化：平衡拟合能力和泛化能力
    - 过度正则化：限制模型学习能力，导致欠拟合
4. 动量（Momentum）
    - 作用：累积历史梯度，加速收敛，减少震荡
    - 典型值：0.9 或 0.95
    - 无动量：训练不稳定，容易陷入局部最优

评估指标：

- 最终测试准确率
- 训练损失曲线的平滑度
- 收敛速度（达到目标准确率所需的 epoch 数）
- 总训练时间

3.4 Task4：实现现代网络架构

经典 CNN 架构的演进始终围绕 “如何在提升模型表达能力的同时，解决深层训练难题” 展开。本任务首先回顾经典 LeNet 的定位与局限，进而引入残差网络（ResNet）的核心设计思想，并基于算力约束实现轻量级残差网络（LightweightResNet），验证现代架构对模型性能的提升价值。

3.4.1 经典 LeNet 的背景与时代局限

LeNet 由 LeCun 于 1998 年提出，是首个成功应用于手写数字识别（MNIST 数据集）的 CNN 架构，其核心贡献在于确立了 “卷积 - 池化 - 全连接” 的经典 CNN 流程，为后

续网络设计奠定了基础。但受限于当时的硬件算力与数据集规模，LeNet 存在显著的时代局限，难以适配现代图像分类任务（如 CIFAR-10 的彩色 32×32 图像）：

1. 网络层数浅：仅包含 2 个卷积层与 3 个全连接层，参数量约 6 万，模型表达能力有限，无法捕捉复杂图像的多层级特征（如物体的纹理、形状组合）；
2. 缺乏现代训练组件：未引入批量归一化（BatchNorm）与正则化机制，训练过程易出现收敛缓慢、过拟合问题；
3. 架构适配性差：针对灰度单通道图像设计，且全连接层依赖固定输入尺寸（如 MNIST 的 28×28），无法灵活处理通道数更多、尺寸变化的图像；
4. 深层训练瓶颈：未解决“梯度消失”问题 —— 当网络层数增加时，反向传播过程中梯度会因多次矩阵乘法逐渐衰减至接近 0，导致深层参数无法有效更新。

这些局限使得 LeNet 在 CIFAR-10 等复杂数据集上的性能天花板较低，也推动了后续深层网络（如 ResNet）的研发。

3.4.2 ResNet 的提出与核心设计思想

2015 年，He 等人提出残差网络（ResNet），其核心目标是突破“深层网络训练难”的瓶颈，首次实现了 1000 层以上 CNN 的稳定训练。ResNet 的提出源于对“梯度消失”问题的深刻洞察，其核心设计思想可拆解为以下三部分：

核心直觉：残差连接解决梯度回传问题

传统 CNN 的前向传播中，某一层的输出可表示为 $H(x) = f(x)$（f 为卷积、激活等操作的组合，即“映射函数”）；而 ResNet 引入“残差连接”（Shortcut Connection），将前向传播改写为：$H(x) = f(x) + x$，其中，x 为该层的原始输入（即“残差项”），$f(x)$为“残差映射”（卷积层需学习的差异部分）。

这一设计的关键价值在于梯度回传效率的提升：反向传播时，梯度不仅可通过 $f(x)$的路径传递，还能通过残差连接直接传递（梯度为 1，无衰减），有效避免了深层网络中梯度因多次乘法而趋近于 0 的问题，使深层参数可被有效更新。

核心组件：ResidualBlock 的设计逻辑

ResNet 的基本构建单元为“残差块（ResidualBlock）”，本任务中的实现采用“预激活”变体（更稳定的训练方式），其结构与逻辑如下：

1. 基础结构：由 2 个 3×3 卷积层堆叠而成，每个卷积层后紧跟 BatchNorm2d（批量归一化）——BatchNorm 可标准化层输出，加速收敛并缓解过拟合；
2. 激活函数顺序：采用“预激活”策略 ——ReLU 激活函数置于第一个 BatchNorm 之后（而非卷积层之后），且仅在残差映射与残差项相加后，才施加第二次 ReLU，避免激活函数对原始输入（残差项）的“破坏”；

LightweightResNet 的实现逻辑（适配算力约束）

为适配有限算力，本任务设计“轻量级残差网络（LightweightResNet）”，在保留 ResNet 核心优势的前提下，通过精简层数、控制通道数降低计算成本，其整体架构分为“茎干（Stem）- 主体（Body）- 头部（Head）”三部分：

茎干（Stem）：轻量特征初始化

不同于标准 ResNet 的“7×7 大卷积 + 最大池化”重型茎干，本设计采用轻量方案：

- 结构：3×3 卷积（conv1）→ BatchNorm2d → ReLU；
- 功能：将 CIFAR-10 的 3 通道彩色图像转换为 32 通道特征图，无额外池化操作（减少信息损失），参数量仅为标准 ResNet 茎干的 1/5。

主体（Body）：精简残差块堆叠

主体由 3 个“单残差块阶段”组成（标准 ResNet 每个阶段含多个残差块），实现特征逐步抽象与维度调整：

- layer1：32 通道输入→32 通道输出，stride=1（保持 32×32 空间尺寸），1 个 ResidualBlock；
- layer2：32 通道输入→64 通道输出，stride=2（空间尺寸减半至 16×16），1 个 ResidualBlock（shortcut 含 1×1 卷积适配维度）；
- layer3：64 通道输入→128 通道输出，stride=2（空间尺寸再减半至 8×8），1 个 ResidualBlock（shortcut 含 1×1 卷积适配维度）；
- 特点：总卷积层数仅 7 层（1 茎干 +2×3 主体），最大通道数 128，计算量仅为标准 ResNet-18 的 1/10。

头部（Head）：现代分类层设计

摒弃 LeNet 的传统全连接层（参数量大、易过拟合），采用现代分类头设计：

1. AdaptiveAvgPool2d ((1,1))：全局平均池化（GAP）—— 无视输入空间尺寸（如 8×8），对每个通道的空间维度取平均，输出固定形状 [BatchSize, 128, 1, 1]，避免全连接层对输入尺寸的依赖；
2. 展平操作：x.view (x.size (0), -1) → 将 GAP 输出展平为 [BatchSize, 128] 的特征向量；
3. Dropout 层：引入 Dropout（概率 0.5）进行正则化，降低过拟合风险；
4. 分类层：1 个 Linear 层（128→10）—— 直接将 128 维特征映射到 CIFAR-10 的 10 个类别，参数量仅 1290（远少于 LeNet 的全连接层）。

3.4.4 架构总结

LightweightResNet 是 ResNet 核心思想与算力约束平衡的产物，其设计亮点可概括为：

1. 保留现代架构优势：通过残差连接解决梯度消失，BatchNorm 加速收敛，GAP 简化分类头，确保深层训练稳定性与模型表达能力；
2. 极致轻量化：7 层卷积、最大 128 通道、无重型茎干，在 CIFAR-10 任务中实现“低算力消耗 - 高泛化能力”的平衡；
3. 适配性强：针对 32×32 彩色图像优化，避免了 LeNet 对小尺寸、单通道图像的依赖，为后续性能对比（与 LeNet）提供了公平且有代表性的现代架构基准。

---

4 实验结果与分析

4.1 基准模型的损失函数可视化

在设定好模型形式（包括惩罚项、卷积层与池化层设置，以及优化器）并在 CIFAR-10 之上完成训练后，本部分对基准 LeNet 模型在训练集上的性能表现进行了分析。具体而言，通过代码可视化交叉熵（Cross-Entropy Loss）形式损失函数，本部分尝试细致识别并分析基准模型在不同训练轮次呈现出的动态特征。

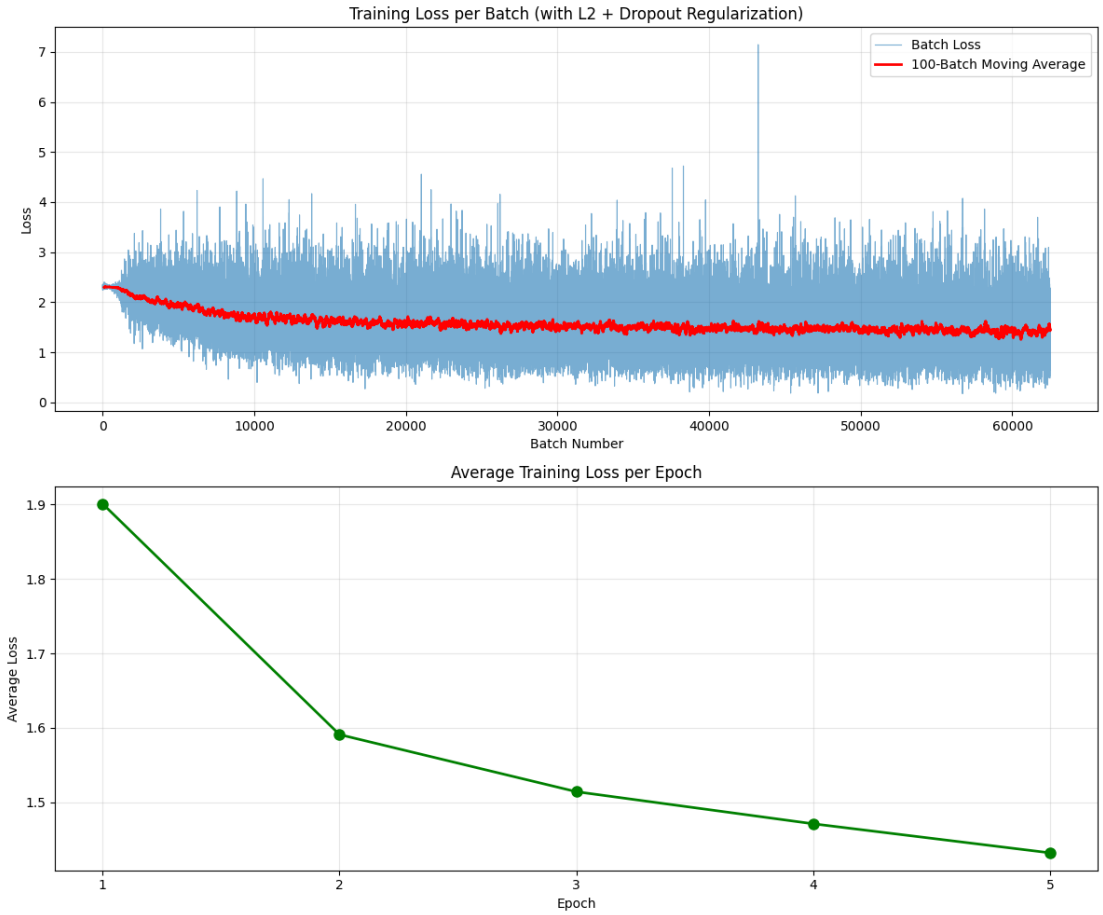

图 1 基准模型损失函数动态可视化

经验误差（Empirical Error）分析

实证结果（见下图“Average Training Loss per Epoch”）表明，模型的经验误差随着训练周期的增加呈现出严格的单调递减趋势。

1. 初始阶段（Epoch 1 -> Epoch 2）： 训练的第一个周期结束后，平均损失约为 1.90。在进入第二个周期后，损失经历了最显著的下降，降至约 1.59。这表明模型在训练初期迅速学习到了数据中的宏观且易于区分的特征模式，优化算法（如 SGD 或 Adam）在此阶段的梯度下降效率最高。
2. 收敛阶段（Epoch 2 -> Epoch 5）： 从第二个周期开始，损失曲线的斜率明显变缓，呈现边际效益递减的特征。平均损失从 Epoch 2 的~1.59 稳步下降至 Epoch 5 的~1.43。这一下降速率的放缓是深度学习训练中的典型现象，表明模型已基本拟合了主要的数据分布，现正处于参数的精细调整（fine-tuning）阶段，试图捕捉数据中更细微、更复杂的模式。
3. 模型拟合状态： 截至 Epoch 5，损失曲线仍处于下降通道，表明模型尚未在训练集上达到完全收敛。这（以及上图“Training Loss per Batch”所示的正则化策略）证实了优化过程正在有效地最小化目标函数，模型成功地*拟合*了训练数据。

训练周期 (Epoch)	观测平均损失 (Approx.)	现象解释（模型拟合）	对泛化能力的推论与待验证项
Epoch 1	~1.90	模型完成对数据的首次完整遍历。损失值反映了参数随机初始化（或预训练）后的初始性能。	初始基线。
Epoch 2	~1.59	损失大幅下降。模型快速学习了数据中的主要特征和最易区分的模式。	**拟合能力得到验证**。模型正在快速收敛。
Epoch 3	~1.51	损失下降速率开始放缓（边际递减）。	模型进入微调阶段，开始学习更精细的特征。
Epoch 4	~1.47	损失持续缓慢下降。	优化过程仍在继续，但接近局部最优解。
Epoch 5	~1.43	损失继续下降，但斜率趋于平缓。	**过拟合风险开始增加**。此时是监控测试集损失的关键时期。

泛化能力（Generalization Ability）讨论

当前图表显示训练损失持续降低。这是模型良好拟合的必要条件，但非充分条件。在实际应用中，必须警惕一种可能性：即训练损失持续下降的同时，测试损失（Test Loss）开始停滞甚至上升。这种现象即为过拟合，表明模型开始“记忆”训练数据中的噪声和特异性，而不是学习可泛化的底层规律。事实上，此基准模型中正则化策略的引入，正是为了*提升*模型的泛化能力、*抑制*过拟合。因此，尽管训练损失在下降，但这些正则化手段正在积极地限制模型的复杂度，以期缩小训练误差与验证误差之间的差距。

然而，这一部分的输出未必真正体现了模型的泛化能力。泛化能力，即模型在未见数据（如验证集或测试集）上的表现，无法*仅*通过经验误差（训练损失）来独立评估。评估泛化能力需要引入泛化误差（Generalization Error），即训练损失与验证损失之间的差距。**事实上，模型在保留的测试集（Test Set）上进行了评估，其最终预测准确率（Accuracy）为 49%。**

这一结果揭示了模型的严重缺陷。尽管训练损失（如前一部分所示）在迭代中稳步下降，且模型部署了 L2 正则化与 Dropout 等技术，但最终的泛化性能（Generalization Performance）远未达到可接受水平。

原因分析如下：

1. **正则化效果有限：** 49%的准确率（假设在 10 分类任务如 CIFAR-10 上，随机猜测为 10%）表明，模型确实学习到了一些超越随机噪声的、具有统计意义的数据模式。正则化项在一定程度上防止了模型在训练初期彻底发散或完全“记忆”噪声。

2. **泛化能力不足：** 尽管如此，49%的准确率在 LeNet 这类成熟架构上不够理想。这暴露出训练损失和测试准确率之间的巨大鸿沟（Gap）。模型虽然在训练集上表现出良好的*拟合能力*，但未能将这种能力*泛化*到未见过的数据上。
3. **诊断：** 这种现象（低训练损失与低测试准确率并存）指向一个核心问题：**当前模型的配置（Configuration）处于次优状态**。虽然正则化策略被引入，但其强度可能设置不当。更广泛地说，模型的学习率、批量大小（Batch Size）或网络结构本身的容量，可能均未达到一个平衡点，导致模型或陷入了不良的局部最优，或正则化过度/不足。

因此，仅仅*加入*正则化项并不足以保证模型的泛化能力。模型的最终性能高度依赖于这些配置的精确校准。这必然地引出了下一个关键步骤：对超参数（Hyperparameters）进行系统性的调优。

4.2 Task3：超参数调优分析

4.2.1 实验配置

基于基准 LeNet 模型（实验 1）所展现的次优泛化能力（最终测试准确率 50.68%），本部分启动了一系列(共 7 组)对照实验，旨在通过系统的超参数调优(Hyperparameter Tuning）来诊断模型瓶颈并提升其泛化性能。调优范围覆盖了优化器关键参数（学习率、动量）、训练轮次（Epochs）以及正则化策略（L2 权值衰减、Dropout 率）。

表格 1 超参数调优配置

实验编号	名称	学习率	Epoch	动量	Weight Decay	Dropout
1	基准配置	0.001	5	0.9	0.01	0.5
2	高学习率	0.01	5	0.9	0.01	0.5
3	低学习率	0.0001	5	0.9	0.01	0.5
4	更多 epoch	0.001	10	0.9	0.01	0.5
5	无正则化	0.001	5	0.9	0	0
6	强正则化	0.001	5	0.9	0.1	0.7
7	无动量	0.001	5	0	0.01	0.5

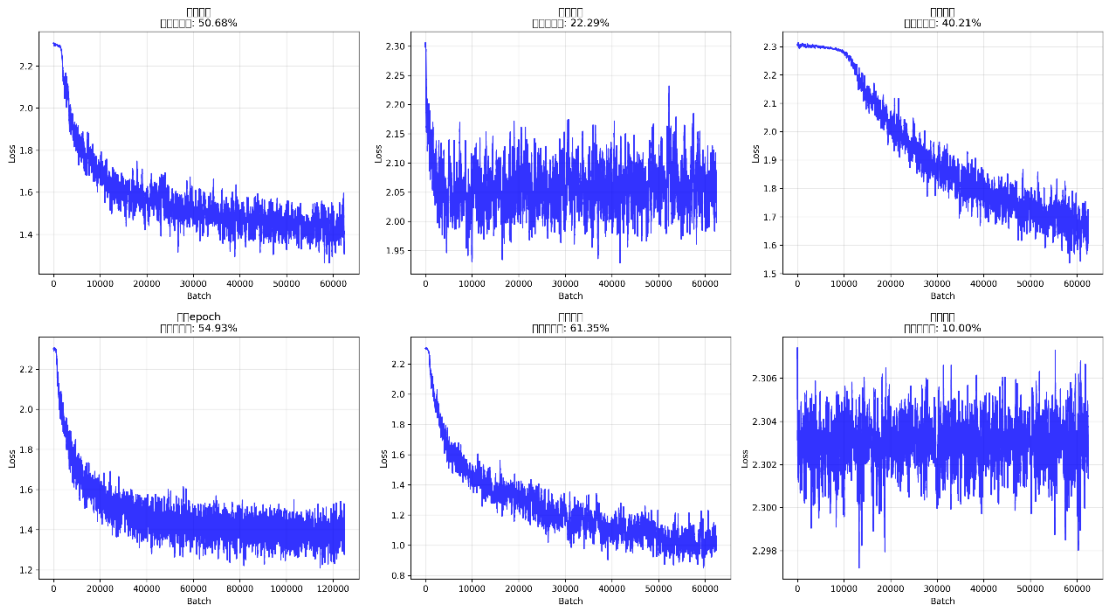

图 2 六种超参数调优配置损失函数动态 [1]

4.2.2 实验结果与分层讨论

通过对比分析 7 组实验的训练日志与最终性能指标，可从以下几个维度对模型行为进行深入解构：

优化器参数（学习率与动量）的敏感性

- 学习率 (Learning Rate, lr): 学习率的设置对模型收敛性具有决定性影响。
  - 高学习率 (实验 2, $lr=0.01$): 导致优化失败。训练损失在整个训练过程中停滞于高位（~2.06），最终准确率仅为 22.29%。这表明学习率过大，导致优化器在损失函数的“峡谷”中剧烈振荡（oscillating）或直接发散（diverging），无法找到有效的局部最优解。
  - 低学习率 (实验 3, $lr=0.0001$): 导致收敛极其缓慢。尽管损失在 5 个 Epoch 内确实下降（从 2.29 降至 1.69），但其最终训练损失（1.692）远高于基准（1.434），导致准确率低下（40.21%）。这显现了严重的欠拟合（Underfitting），模型在有限的训练步数内未能充分学习数据分布。
- 动量 (Momentum): 动量项（Momentum=0.9）对加速收敛至关重要。
  - 无动量 (实验 7, $momentum=0$): 导致收敛效率大幅降低。其最终训练损失（1.745）和准确率（36.51%）均显著劣于基准配置。这证实了动量机制在克服损失曲面上的平坦区域和鞍点（saddle points）方面的重要作用。

训练时长（Epochs）的影响

- 更多 Epoch (实验 4, $epochs=10$): 在保持基准配置不变的情况下，将训练周期从 5 延长至 10，使得最终准确率从 50.68% 提升至 54.93%。训练损失也进一步从 1.434 降至 1.379。
- 分析: 这有力地证明了基准模型（实验 1）处于欠训练（Undertrained）状态。在第 5 个 Epoch 时，模型尚未在训练集上充分收敛，延长训练时间使其得以继续优化参数，从而同时提升了训练集拟合度和测试集泛化能力。

[1] 由于没有配置 plotly 中文显示包，诸示意图标题缺失；但是限于模型训练时长与算力有限，没有进行重新绘制。依次为：高学习率、低学习率、更多轮次、无正则化、强正则化与无动量。

4.2.3 可视化综合分析

此外，本实验绘制了四象限可视化图表，从训练过程、泛化表现、最终性能及核心指标关联四个维度，全面直观地呈现了不同超参数配置对模型的影响，各子图的绘图逻辑与直观结论具有清晰的递进与呼应关系。

左上角“训练损失对比（每轮次）”子图以训练轮次（Epoch）为横轴、平均训练损失为纵轴，核心目的是观察不同配置下模型的收敛速度与最终训练损失水平。从图中可直接看出，不同配置的训练收敛差异显著：部分配置（如“无正则化”，紫色曲线）的训练损失下降迅速，且最终稳定在较低水平（~1.02），表明模型能快速学习训练数据中的规律；而另一部分配置（如“强正则化”，棕色曲线）的训练损失全程处于高位（~2.3）且几乎无下降趋势，说明该配置下模型未实现有效学习。

与之对应，右上角“测试准确率对比”子图以训练轮次为横轴、测试准确率（%）为纵轴，聚焦模型泛化能力的动态变化，其趋势与训练损失形成鲜明呼应：训练损失下降快的配置（“无正则化”，紫色），测试准确率上升也更为迅速，最终能达到接近 60% 以上的水平；而训练损失无改善的配置（“强正则化”，棕色），测试准确率始终徘徊在 10% 左右，与随机猜测效果（10%）相近，反映出模型严重欠拟合。

为了更清晰地量化各配置的最终性能，左下角“最终测试准确率对比”子图采用柱状图形式，以实验配置为横轴、最终测试准确率（%）为纵轴，直观呈现了各配置的性能天花板：最优配置（“无正则化”）的最终测试准确率高达 61.35%，而最差配置（“强正则化”）仅为 10.00%，悬殊的数值差异直接凸显了超参数调优对模型泛化性能的决定性作用。进一步地，右下角“最终训练损失与最终测试准确率关联分析”子图以最终训练损失为横轴、最终测试准确率（%）为纵轴，旨在探究训练效果与泛化性能的内在联系。从图中散点分布可明确观察到显著的负相关趋势：最终训练损失较低的配置（点位靠左），对应的最终测试准确率普遍更高（点位靠上）；而最终训练损失处于高位的配置（点位靠右），其测试准确率也大多处于较低水平（点位靠下）。这说明在本次超参数调优实验中，训练损失的有效降低（即更优的拟合）对模型泛化能力的提升具有直接的正向作用。

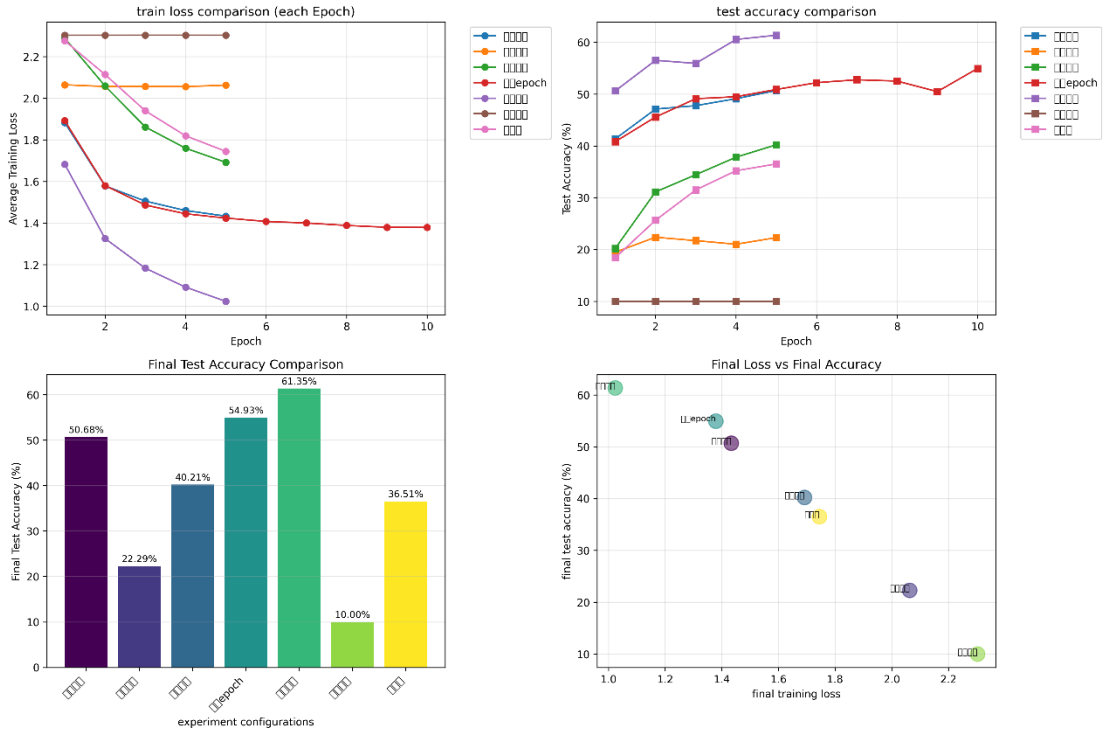

图 3 超参数调优对比示意图

4.2.4 正则化强度对泛化能力的核心影响

本次调优中最具启发性的发现集中在正则化策略的对比上。实验结果颠覆了“正则化总是有益”的直觉，揭示了模型当前的根本问题。

- 强正则化 (实验 6, $L2=0.1, Dropout=0.7$): 导致了灾难性的极端欠拟合。极高的 L2 惩罚系数和 Dropout 率完全压制了模型的学习能力。训练损失在 5 个 Epoch 中始终停滞在~2.30（接近随机猜测的损失值），最终准确率仅为 10.00%，等同于随机猜测。这表明正则化强度远超模型所需，使其无法拟合任何有效模式。
- 基准正则化 (实验 1, $L2=0.01, Dropout=0.5$): 提供了 50.68% 的准确率。如前所述，该模型表现不佳。
- 无正则化 (实验 5, $L2=0, Dropout=0$): 取得了 61.35% 的最佳准确率，显著超越所有其他配置。与此同时，其训练损失也降至所有实验中的最低点（1.024）。

深入分析：

这一对比（实验 6、1、5）清晰地表明，基准模型（实验 1）的核心问题并非过拟合（Overfitting），而是由不当正则化引起的欠拟合（Underfitting）。

在基准配置中，L2 惩罚（0.01）和 Dropout（0.5）的组合对于当前架构（LeNet）和数据集而言过于严苛。它过度限制了模型的有效容量（Effective Capacity），使其在训练数据上的拟合能力都受到了不必要的抑制（训练损失 1.434）。

当移除所有正则化项后（实验 5），模型得以充分利用其全部参数容量来拟合训练数据，训练损失大幅降低（1.024）。关键在于，这种更优的训练集拟合*同时*转化为了更强的测试集泛化能力（准确率 61.35%）。这说明在该任务中，模型（LeNet）本身的结构简单性可能已经是一种隐性的正则化，额外的强正则化反而阻碍了其学习复杂数据分布的能力。

结论汇总

超参数调优实验表明，基准模型（50.68%）的性能瓶颈主要源于欠拟合，具体归因于：1) 训练周期不足（Undertraining）；2) 正则化强度过高（Over-regularization）。

通过移除所有正则化项（实验 5），模型性能大幅提升至 61.35%。这揭示了在当前配置下，模型的首要任务是提升其拟合能力，而非抑制过拟合。基准模型所采用的正则化策略（L2=0.01, Dropout=0.5）被证明是适得其反的。

未来的优化方向应以“无正则化”（实验 5）配置为新基准，并结合“更多轮次”（实验 4）的结论，即延长训练时间，以探索模型在充分训练下的性能极限。

表格 2 正则化强度与模型性能之间的权衡关系

实验配置	L2 (Weight Decay)	Dropout	最终训练损失	最终测试准确率	结论
强正则化 (Exp 6)	0.1000	0.70	2.3029	10.00%	**极端欠拟合** (无法学习)
基准配置 (Exp 1)	0.0100	0.50	1.4336	50.68%	**显著欠拟合** (过度约束)
更多 Epoch (Exp 4)	0.0100	0.50	1.3792	54.93%	缓解欠拟合 (训练更久)
无正则化 (Exp 5)	**0.0000**	**0.00**	**1.0241**	**61.35%**	**当前最优** (容量释放)

4.3 Task4：现代网络架构对比

在上一阶段（超参数调优实验）中，基准 LeNet 模型的最佳泛化性能被锁定在 61.35%（“无正则化”配置）。尽管这一成绩相较于初始的基准配置（50.68%）有所提升，但其绝对性能在 CIFAR-10 这类复杂彩色图像数据集上仍显不足。

本实验在此认为，这一性能在某种程度上是 LeNet 架构的内在缺陷的体现。如前文（3.4.1 节）所分析，LeNet 作为早期架构，在此基础上继续进行超参数调试，所能带来的边际收益已极其有限。因此，为了寻求性能的突破，我们不再局限于对 LeNet 的修补，而是转向采用为解决“深层训练难题”而生的现代精细模型。基于 3.4.2 至 3.4.4 节的分析，本任务将部署所设计的轻量级残差网络（ResNet），以验证残差连接、批量归一化及现代分类头设计对模型泛化能力的真实提升价值。

结果与分析

本实验在相同的训练配置下（10 个 Epochs，SGD 优化器，L2=0.01，Dropout=0.5），分别训练了“LeNet（基准）”与“LightweightResNet”模型，并在 CIFAR-10 测试集上评估其性能。

4.3.1. 最终性能：ResNet 压倒性优势

最核心的结论见于“final test accuracy”图表。在 10 轮训练后：

- **Lightweight ResNet：达到 63.37% 的测试准确率。**
- **LeNet (benchmark)：仅达到 36.30% 的测试准确率。**

二者之间存在高达 27% 的绝对准确率差距。这无可辩驳地证明了，在处理 CIFAR-10 这类彩色图像任务时，LightweightResNet 的架构设计远优于 LeNet。LeNet 的浅层结构（参数量仅 6.2 万）显然已构成其表达能力的严重瓶颈。

4.3.2. 收敛动态：ResNet 更快、更强

test accuracy comparison 动态曲线图（左图）揭示了差距形成的过程：

- **起点差异**：在 Epoch 1 结束时，LeNet 准确率仅为 15.01%，而 ResNet 已达到 44.68%。这归功于 ResNet 的 BatchNorm（批量归一化）层，它极大地稳定了初始训练，使模型能从一个更优的起点开始收敛。
- **收敛速率**：ResNet 的准确率（橙线）攀升斜率远大于 LeNet（蓝线）。ResNet 在第 3 轮时（53.02%）已超越 LeNet 训练 10 轮的最终性能。这表明，残差连接（Residual Connections）有效解决了深层训练中的梯度传播问题，使优化器能高效地更新参数。
- **拟合能力**：从训练日志可知，ResNet 的最终训练损失（0.936）远低于 LeNet（1.768），证明其能更充分地拟合训练数据。

4.3.3. 训练效率

就参数量**而言**：ResNet 的参数量（30.9 万）约为 LeNet（6.2 万）的 **5 倍**；就**训练时间**：ResNet 的总用时（53.0 秒）与 LeNet（49.1 秒）**几乎持平。换言之，参数量增加 5 倍，ResNet 在训练所用的 ecs.gn7i-c8g1.2xlarge/NVIDIA a1 配置上训练时间没有显著改变。**

这其中的因果机制可能在于：LeNet 的参数大量集中在效率低下的全连接层（FC Layers）；而 ResNet 摒弃了庞大的 FC 层，采用全局平均池化（GAP）和更深的 3x3 卷积堆叠。这使得 ResNet 在拥有更高模型容量（表达能力更强）的同时，其计算效率（特别是 GPU 上的并行效率）反而与 LeNet 相当。

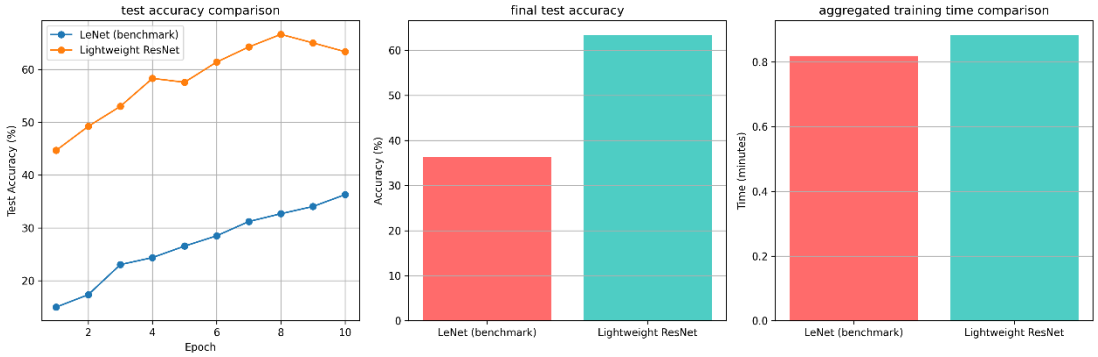

图 4LeNet 与残差网络性能对比

表格 3 架构对比总结表

评估指标	LeNet (benchmark)	Lightweight ResNet	分析
最终测试准确率	36.30%	63.37%	ResNet 性能提升 +27.07% (压倒性优势)
Epoch 1 准确率	15.01%	44.68%	ResNet (BatchNorm) 初始收敛极快
最终训练损失	1.768	0.936	ResNet 拟合能力更强
参数量	约 6.2 万	约 30.9 万	ResNet 模型容量 (×5)
总训练用时	49.1 秒	53.0 秒	几乎持平 (效率极高)