三层神经网络图像分类器实验报告

黄乐妍 23307130460

2025 年 4 月 12 日

1 项目介绍

1.1 数据集介绍

本项目使用 CIFAR-10 数据集，这是一个广泛应用于图像分类任务的基准数据集。它包含 10 个类别的彩色图片，每张图像的大小为 32×32 像素，包括飞机、汽车、鸟、猫、鹿、狗、青蛙、马、船和卡车。数据集分为 50,000 张训练图像和 10,000 张测试图像，每个类别在训练集和测试集中均匀分布。

我们知道，彩色图像由三个颜色通道（红、绿、蓝）组成，每个通道的像素值范围为 0 到 255。对于 CIFAR-10 数据集中的 32×32 彩色图像，输入维度为 32×32×3=3072，参数数量较多。在实验中我发现，经过多次调参 FCNN 的 accuracy 最多只能达到 0.5 左右，训练 5 个 epoch 之后上升缓慢；而且存在较为严重的过拟合现象，即使加入 l2 正则化和随机 dropout 仍无太大改善。前者可能因为这个数据集中的物体具有较为丰富的局部纹理和形状特征，这些特征对于正确分类至关重要（比如猫和狗、狗和马的分类），但 FCNN 对输入数据的空间结构不敏感，仅将输入视为一维向量进行处理，难以有效捕捉局部特征；后者可能是因为输入维度高且参数数量多。

所以本项目最终选择 CNN 作为网络架构。

模型链接: Google Drive Link

代码链接: Github Repo Link.

1.2 神经网络架构概览

在不使用 Pytorch,Tensorflow 的条件下，利用 numpy 与 sklearn（数据处理部分），设计并实现了一个包含三层卷积的卷积神经网络（CNN），后接全连接层完成分类。

层	过滤器数量	尺寸	输出形状
输入	3	32×32	3×32×32
Conv	16	32×32	16×32×32
AvgPool	16	16×16	16×16×16
Conv	20	16×16	20×16×16
AvgPool	20	8×8	20×8×8
Conv	20	8×8	20×8×8
AvgPool	20	4×4	20×4×4
Output (FC)	10	-	10（分类输出）

表 1: 网络架构表

1.3 报告结构

本文共分为五个部分：项目介绍、代码实现、运行结果、结果分析与改进方向。

2 代码实现

2.1 代码结构说明

项目采用模块化设计，主要包括：

`main.py`（训练主程序）

`neuron.py`（定义神经元）

`layer.py`（定义 Layer 抽象基类，确保所有神经网络层都具有一致的接口，支持前向传播，反向传播，权重更新等方法）

`linearlayer.py`（线性层，用于全连接层和 output 层）

`convlayer.py`（卷积层）

`poollayer.py`（池化层）

`neuralnet.py`（完整的层封装）

`utils.py`（神经网络的工具函数）：主要包括损失函数，卷积层的相关操作，对神经元权重的不同更新方式，激活函数及其导数。

`tools.py`（训练和可视化的工具函数）：主要包括保存和加载模型，绘制指标变化，可视化卷积层。

2.2 数据处理

采用 He normal 初始化方法（适用于 ReLU 激活函数），并对训练集进行随机打乱（shuffle），避免样本顺序对模型训练产生影响。

在 He Normal 初始化中，权重是从均值为 0、标准差为 $\sqrt{\frac{2}{n}}$ 的高斯分布中随机采样得到的，其中 n 是输入神经元的数量。具体公式为：

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n}}\right)$$

在传播过程中，ReLU 的非线性特性导致其输出的方差可能会随着网络层数增加而逐渐减小。He Normal 初始化通过将权重的方差设置为 $\sqrt{\frac{2}{n}}$，确保了在前向传播过程中，激活值的方差保持稳定；其次，ReLU 神经元在输入值为负时输出为零，如果权重初始化过小，可能会导致神经元在训练初期“死亡”（即输出始终为零）。He Normal 初始化通过适当设置权重的初始值，避免了这一问题。

通过 shuffle 操作，防止数据集中任何固有的顺序或模式影响训练过程，增强了训练过程的随机性和模型的泛化能力。

2.3 训练方法

模型结构设定 使用 3 层卷积结构，每层卷积核大小为 5×5。第一层使用 16 个卷积核，第二、三层使用 20 个卷积核。

激活函数与初始化 所有卷积层均采用 ReLU 激活函数，与 He 初始化配合，缓解梯度消失。

优化策略选择 卷积层使用 NAG 优化器以加速收敛，全连接层使用 Adam 优化器（项目中选择了一般默认的超参数）。

在本项目中，针对卷积神经网络中不同类型的层，采用了差异化的优化器策略：**卷积层使用 Nesterov 加速梯度（NAG）**，而**全连接层使用 Adam**。

我们知道，卷积层主要负责图像特征的抽象与提取，参数相对稀疏，结构稳定，对训练中梯度更新的稳定性要求较高。NAG 在传统 Momentum 的基础上引入“提前量”机制，即在计算当前梯度前，预估未来位置，从而避免局部振荡，提升收敛平稳性，适合卷积层中浅层特征的学习。其参数更新规则如下：

$$v_t = \mu v_{t-1} - \eta \nabla f(\theta_{t-1} + \mu v_{t-1})$$
$$\theta_t = \theta_{t-1} + v_t$$

其中，μ 为动量系数（手动试验了几个常用值，最后设置默认为 0.9，可在 utils.py 里进行修改），η 为学习率，∇f 表示在提前点计算的梯度。

相比之下，全连接层具有更大量级的参数和更加明确的目标导向，尤其接近网络输出端，梯度波动大。为提升训练效率与稳定性，在 output 层采用了 Adam 优化器，其自适应调整每个参数的学习率，同时引入了梯度的一阶矩估计与二阶矩估计，可加速收敛并减少对初始学习率的敏感性。跟据这篇文章的试验，在 cifar-10 数据集上效果不错。其更新规则如下：

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla f(\theta_t)$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla f(\theta_t))^2$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

其中，α 为学习率，β_1, β_2 分别控制一阶、二阶矩估计的指数衰减速率，ϵ 是防止除零的小常数。

常用默认超参数为：

$$\alpha = 0.001, \quad \beta_1 = 0.9, \quad \beta_2 = 0.999, \quad \epsilon = 10^{-8}$$

正则化与 Dropout 为防止过拟合，在全连接层引入 L2 正则化与 Dropout（仅用于 FC 层，以保留卷积层全部特征），dropout 在 layer 类的默认值为 1，即不进行随机掉落，由于在训练过程中通过观察曲线发现和 FCNN 相比并未出现明显过拟合，由于时间较为紧张且性能已经不错了就没有调。

验证集划分 从训练集中划分出 4000 个样本作为验证集，用于调参与早停策略评估。

学习率衰减 每个 epoch 之后对学习率进行指数衰减，以保证在收敛后期能更稳定。在训练初期使用较大的学习率（如初始值 η_0），可以快速逃离局部极小值或平坦区域，加速初期收敛。随着训练步数 t 增加，学习率按指数规律衰减：

$$\eta(t) = \eta_0 \cdot \gamma^t \quad (\gamma \in (0,1))$$

其中：

- η_0 为初始学习率（调优后设置为 0.0001）
- γ 为衰减系数（调优后设置为 0.96）
- t 为训练步数

3 运行结果

3.1 超参数调优

因为我的电脑性能不太好，在完整数据集上跑完一个 epoch 要 40Min, 所以调参时选取了 1/10 的数据集进行调试, 大概 3-5 个 epoch 就比较接近最终收敛值了, 所以每个参数跑了三个 epoch。如下为部分超参数搜索实验结果：

表 2: 不同 Batch Size 对测试集准确率的影响

Batch Size	**Test Accuracy (%)**
1	40.75
4	22.75
8	19.38
32	18.63
64	17.25
128	15.25

表 3: 不同学习率（Learning Rate）对测试集准确率的影响

Learning Rate	**Test Accuracy (%)**
1	11.88
0.1	9.75
0.01	11.88
0.001	41.38
0.0001	38.38

表 4: 不同学习率衰减因子（Decay）对测试集准确率的影响

Learning Rate Decay	**Test Accuracy (%)**
1.0	39.00
0.98	39.25
0.96	**42.50**

但是实际试跑时发现，选用这组参数时，loss 快速下降并且开始震荡，出现了欠拟合现象，搜索后发现可能是调参时出现错误，小数据集上出现了“假收敛”，导致小数据集上最优的学习率对大数据集不够精细。首先是小数据集噪声大、样本分布不完整，模型可能仅需少量迭代（如 3 个 epoch）就能快速收敛，偏大的 LR 在小数据上表现“最优”（快速收敛），但到大数据的复杂分布时，沿用小数据调出的 LR，模型在大数据上前几轮仅学到粗浅特征，更新步长过大，跳过全局最优解，导致欠拟合，而偏小的 LR 在小数据上因 epoch 不足（仅 3 轮）未能充分学习，误判为效果差，实际可能更适合大数据。不过根据试验结果，学习率被调大的可能性更大（只有相对的轻微下降），试验之后的确如此。故最后改用 1e-4 的学习率训练模型。

表 5: 不同 L2 正则化系数对测试集准确率的影响

L2 Regularization	Test Accuracy (%)
0.001	39.88
0.0001	39.38
1e-5	42.15

表 6: 当前实验中最佳超参数配置

超参数	值
Batch Size	1
Learning Rate	0.001
Learning Rate Decay	0.96
L2 Regularization	0.001
Test Accuracy	**42.50%**

3.2 准确率与损失变化曲线

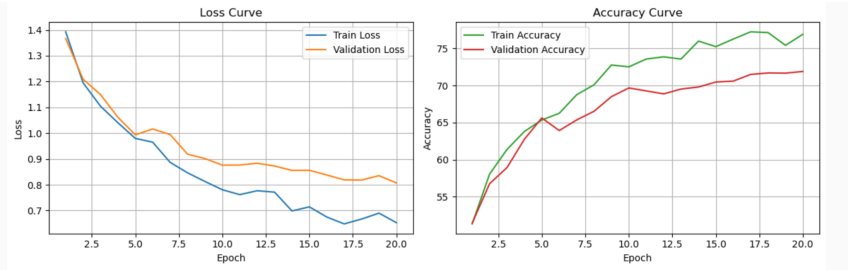

图 1: 训练与验证集上的准确率与损失曲线

图 2: 每个小批量训练的损失曲线

最终测试结果：

- 测试准确率：71.40%

表 7: 模型训练过程记录（20 个 Epoch）

Epoch	学习率	训练损失	训练准确率 (%)	验证损失	验证准确率 (%)
1	0.000100	1.3927	51.33	1.3664	51.43
2	0.000096	1.1950	58.05	1.2089	56.75
3	0.000092	1.1044	61.38	1.1496	58.93
4	0.000088	1.0403	63.80	1.0613	62.75
5	0.000085	0.9797	65.35	0.9937	65.63
6	0.000082	0.9649	66.25	1.0160	63.93
7	0.000078	0.8868	68.75	0.9946	65.35
8	0.000075	0.8463	70.10	0.9186	66.53
9	0.000072	0.8128	72.78	0.9016	68.50
10	0.000069	0.7806	72.53	0.8761	69.68
11	0.000066	0.7618	73.58	0.8765	69.28
12	0.000064	0.7768	73.88	0.8834	68.88
13	0.000061	0.7714	73.58	0.8726	69.53
14	0.000059	0.6986	76.00	0.8556	69.80
15	0.000056	0.7143	75.25	0.8560	70.48
16	0.000054	0.6750	76.28	0.8379	70.60
17	0.000052	0.6483	77.25	0.8189	71.50
18	0.000050	0.6671	77.15	0.8183	71.70
19	0.000048	0.6901	75.43	0.8353	71.68
20	0.000046	0.6530	76.93	0.8070	71.90

- 测试损失：0.8417
- 最佳模型保存为 `best_model.pkl`（验证准确率 71.90%）

3.3 卷积核可视化

图 3: 第一层卷积核可视化（RGB 三通道）

图 4: 第二层卷积核可视化（RGB 三通道）

图 5: 第三层卷积核可视化（RGB 三通道）

4 结果分析

4.1 关于参数设置的解释

第一个感觉比较奇怪的是 batch size 测试出来单个表现似乎最好。搜索过后发现，小 batch size 的梯度噪声更大，但在非凸优化中，可能可以帮助模型跳出局部最小值或鞍点，特别是在像 CIFAR-10 这种多类图像任务中，目标函数很复杂，容易卡在 sub-optimal 的点，所以小 batch size 虽训练时间较长，但能提升泛化能力。其次，小 batch size 具有一定的正则化效果，因为小 batch size 下，不同批次的数据分布差异更大，导致梯度方向更多样化，相当于对数据分布进行了隐式数据增强，减少模型对特定样本的依赖，降低过拟合风险。

但是就计算效率来说，大 batch size 可以利用矩阵并行加速，整体计算效率远高于一个个地处理，在 NumPy 框架下这个优势不明显（没用 GPU），但迁移到 PyTorch/TensorFlow 就会体现。

卷积核大小设置为 5×5，在相对浅层网络中能提取更宽范围的空间特征。（比如 LeNet-5 当年就是用的 5×5 ，而且我的硬件条件也不太好（bushi））。（如果不考虑层数限制）可以考虑课堂上提到的把 5*5 的卷积核换成两个 3×3 的卷积堆叠，感受野就可以达到 5×5（还多了一个非线性激活函数），但参数量和计算量远小于直接用一个 5×5。

卷积核数量选择为 16/20/20，较好地平衡了计算资源与提取能力。

- **首层（16 个卷积核）**：负责捕捉低级特征（如边缘、纹理等），由于底层特征通用性强，无需过多通道数。
- **中间层（20 个卷积核）**：逐步增加复杂度，融合中级特征（如局部形状、部件组合等）。
- **末层（20 个卷积核）**：保持通道数稳定，避免参数量爆炸（尤其当输入尺寸较小时）。

4.2 损失曲线（左图）

- **训练损失**（蓝线）呈现持续下降趋势，表明模型在训练集上收敛良好；
- **验证损失**（橙线）整体下降，但在第 10 个 epoch 后下降速率减缓（相较全连接网络 FCNN 性能已有显著提升），提示模型可能出现轻微过拟合，但仍处于可控范围内。

4.3 精度曲线（右图）

- **训练精度**（绿线）稳定上升至 78%，证实训练过程有效；
- **验证精度**（红线）最终稳定在 72%，与训练精度存在 6% 差距，反映模型存在适度过拟合现象，但未达到失控程度。

4.4 小批量训练的损失曲线

- **Raw**表示原始的每个小批量数据对应的损失值。由于每个 mini-batch 的样本是随机抽取的，数据分布差异导致每次计算的 loss 有高有低，即使模型在“整体变好”，也可能有局部小批次突然“难学”或样本噪声较大，导致 loss 飙升，加上我的 mini-batch 选取的很小，所以该曲线波动非常剧烈，不同迭代次数下损失值起伏很大；
- **Smoothed**尽管原始损失波动大，但平滑后的损失总体呈下降趋势，意味着随着训练迭代不断进行，模型参数在整体上变得更优，学习过程有效，虽然 noisy，但收敛方向是对的。

4.5 卷积特征分析

第一层主要提取颜色与边缘特征，后续层能识别更复杂的局部模式，展示出从底层到高层的层级抽象能力。

第一层卷积核接收到的是原始图像像素输入，主要提取低级特征，如边缘、颜色、方向、纹理等，部分卷积核具有强烈的方向性（如第 1，6，7 个出现的带有明显水平/垂直模式的边缘检测）。第二层卷积核以第一层输出的特征图为输入，进一步组合低级特征，图像看起来比第一层更复杂，出现更多模糊的颜色混合和局部模式，说明开始形成对局部纹理、简单组合结构的响应。第三层是更深层的卷积操作，卷积核的可视化效果更加抽象，说明该层捕捉的是更高阶的抽象特征，有助于模型完成更复杂的分类决策。

5 改进方向

- 可以尝试引入 Batch Normalization ，在卷积计算之后、应用激活函数之前对各个通道的输出分别进行 Batch Normalization，加快收敛速度；
- 优化卷积结构，（如果不考虑层数限制）把 5*5 的卷积核换成两个 3×3 的卷积堆叠，感受野就可以达到 5×5（还多了一个非线性激活函数），减少参数量和计算量，增强灵活性；
- 扩展数据增强策略，如旋转、裁剪等，提升泛化能力；
- 对 dropout rate 进行超参数调优，更好地避免过拟合。