实验报告

从零开始构建三层神经网络分类器，实现图像分类

姓 名: 余 星 磊
学 号: 23300290012
DATA130051.01 计算机视觉
(春季, 2025)

复旦大学
大数据学院

2025 年 4 月 13 日

摘 要

本次实验旨在从零开始构建一个三层神经网络分类器,用于实现基于 CIFAR-10 数据集的图像分类任务。实验内容包括数据预处理、模型设计、训练过程优化及性能评估。通过对 CIFAR-10 数据集进行归一化、数据重塑和 one-hot 编码等预处理操作，设计了一个包含两层卷积层和一层全连接层的神经网络，结合 ReLU 激活函数、批归一化、最大池化及 Softmax 输出层实现分类功能。采用随机梯度下降（SGD）优化算法，结合 L2 正则化和指数衰减学习率策略进行训练，通过网格搜索调节超参数以优化模型性能。实验结果表明，模型在测试集上达到了 61.3% 的分类准确率，验证了其在图像分类任务中的有效性。参数可视化进一步揭示了模型的特征提取能力，为后续优化提供了方向。

目 录

1 概述

本次实验的主要内容是从零开始构建一个三层神经网络分类器，针对 CIFAR-10 数据集实现图像分类任务。实验涵盖以下核心内容：

- **数据预处理**：对 CIFAR-10 数据集进行处理，包括数据重塑（从 3072 维向量到 3×32×32 三维数组）、归一化（像素值从 [0, 255] 到 [0, 1]）以及 one-hot 编码，确保数据适合模型输入。
- **模型设计**：设计三层神经网络，包括输入层、两个卷积隐藏层（含卷积、批归一化、ReLU 激活和最大池化）和一个全连接输出层（含 Softmax 函数），用于提取图像特征并输出分类概率。
- **训练与优化**：使用随机梯度下降（SGD）算法进行训练，结合 L2 正则化防止过拟合，采用指数衰减学习率策略加速收敛，并通过梯度裁剪增强训练稳定性。
- **超参数调节**：通过网格搜索优化学习率、正则化系数、卷积核数量等超参数，分析不同配置对模型性能的影响。
- **性能评估**：在训练集、验证集和测试集上记录损失和准确率曲线，测试集准确率达 61.3%，并通过类别准确率分析模型在不同类别上的表现。
- **参数可视化**：展示卷积核和全连接层权重的可视化结果，分析模型学习的特征模式，揭示边缘检测和高级特征提取能力。

实验使用 Python 和 NumPy 实现，结合 CuPy 加速计算，依赖 xelatex 编译 LaTeX 文档以生成报告。

2 引言

2.1 实验背景

随着深度学习技术的快速发展，卷积神经网络（Convolutional Neural Networks, CNN）在图像分类、目标检测等领域展现了强大的性能。CIFAR-10 数据集作为一个经典的图像分类基准，包含 10 类 32×32 像素的彩色图像，广泛用于验证神经网络模型的有效性。传统机器学习方法在处理高维图像数据时面临特征提取的瓶颈，而深度学习通过端到端的训练方式自动学习图像特征，显著提升了分类精度。然而，设计和实现一个高效的神经网络需要深入理解其结构、优化方法及超参数调节策略。本实验基于 CIFAR-10 数据集，从零开始构建一个三层神经网络分类器，旨在探索深度学习模型的设计与优化过程，为后续复杂任务提供基础。

2.2 实验目的与意义

本实验的目的是通过从零开始实现一个三层神经网络分类器，掌握卷积神经网络的设计、训练和评估的全流程。具体目标包括：

- 理解并实现数据预处理、模型构建、优化算法及性能评估等关键步骤。
- 通过实验分析不同超参数对模型性能的影响，探索优化模型泛化能力的方法。
- 深入学习卷积层、池化层、全连接层及批归一化等模块的原理及其在图像分类中的作用。

实验的意义在于通过动手实现一个完整的神经网络，加深对深度学习核心概念的理解，同时通过可视化参数和性能分析，揭示模型的学习机制，为优化复杂神经网络提供经验参考。此外，本实验培养了从理论到实践的能力，为后续研究和应用奠定了基础。

2.3 实验环境与工具

本实验在以下环境和工具支持下完成：

- **硬件环境**：配备 NVIDIA GPU 4060 的计算设备），用于加速矩阵运算和模型训练；CPU 为 Intel Core i7，内存 16GB，保证数据处理和代码运行的效率。

- **软件环境**：操作系统为 Windows 11 下的 WSL2，Python 版本为 3.9.21，深度学习相关库包括 NumPy（用于矩阵运算）、CuPy（GPU 加速计算）和 Matplotlib（可视化结果）。
- **开发工具**：使用 VS Code 作为代码编辑器，python 调试和可视化实验结果。
- **文档工具**：实验报告采用 LaTeX 编写，结合`ctex`包支持中文排版，使用 XeLaTeX 编译生成 PDF 文档。

实验代码基于 Python 实现，所有模块（包括数据预处理、模型定义和训练逻辑）均为自主编写，未依赖现有的深度学习框架（如 PyTorch 或 TensorFlow），以加深对底层原理的理解。

3 数据集介绍

3.1 CIFAR-10 数据集概述

CIFAR-10 数据集由 10 个类的 60000 张 32x32 彩色图像组成，每个类有 6000 张图像。有 50000 张训练图像和 10000 张测试图像。

CIFAR-10 数据集一共包含 10 个类别的 RGB 彩色图片：飞机（airplane ）、汽车（automobile ）、鸟类（bird ）、猫（cat ）、鹿（deer ）、狗（dog ）、蛙类（frog ）、马（horse ）、船（ship ）和卡车（truck ）。

CIFAR-10 数据集被划分为 5 个训练批次和 1 个测试批次，测试批次包含每个类中随机选择的 1000 张图像。训练批次包含按随机顺序排列的剩余图像。在 python 中导出文件后返回一个字典，字典的键为 b'batch_label', b'labels', b'data', b'filenames'。

本次实验中使用到的 b'labels' 和 b'data' 分别代表图像的标签与图像的像素颜色。

- b'labels' 为范围 0-9 的 10000 个数字的列表。索引 i 处的数字表示数组数据中第 i 张图像的标签。
- b'data' 为一个数据类型为 uint8 的 10000x3072 numpy 数组。数组的每一行存储一个 32x32 彩色图像。前 1024 个条目包含红色通道值，接下来的 1024 个条目包含绿色通道值，最后 1024 个条目包含蓝色通道值。图像按行优先顺序存储，因此数组的前 32 个条目是图像第一行的红色通道值。

3.2 数据预处理方法

3.2.1 数据重塑

CIFAR-10 数据集中加载的图像数据是一个 NumPy 数组，形状为 (10000,3072)。通过将每个图像的 3072 个像素值重新排列为一个 3×32×32 的三维数组，经过这一步，数据的形状从 (10000,3072) 变为 (10000,3,32,32)，即（图像序号，通道数，宽度，高度）.

3.2.2 数据归一化

将像素值从 [0, 255] 归一化到 [0, 1]。归一化是数据预处理中常见的步骤，有助于加速模型的收敛速度。具体公式为：

$$normalized\_pixel = \frac{255.0}{pixel}$$

3.2.3 one-hot 编码处理

在中间步骤计算梯度的反向传播时，将 labels 标签转化为 one-hot 编码，例如，标签 2 将被转换为 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]。

3.3 数据集划分（训练集、验证集、测试集）

CIFAR-10 数据集中已经划分好训练集和测试集，测试集包括每个类中随机选择的 1000 张图像，共 10000 张图像。我们只需要手动划分验证集。我们通过对合并五个训练批次后的训练集进行随机划分，按照 9:1 的比例划分训练集和验证集，即训练集为 45000 张图像，验证集为 5000 张

4 模型设计

4.1 三层神经网络结构

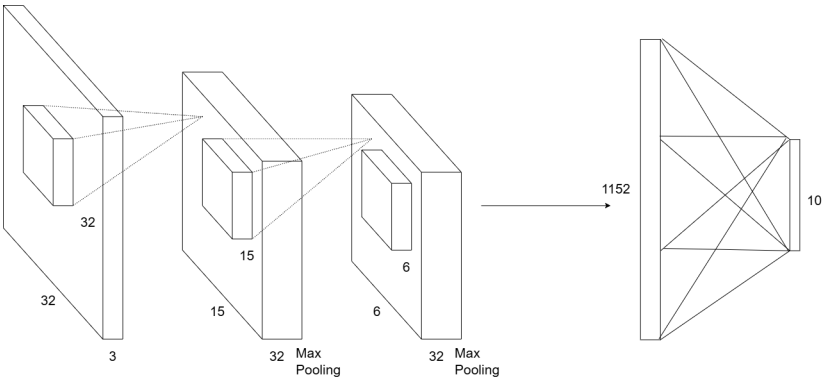

图 1: 神经网络结构图

这个神经网络由以下几个主要部分组成：

- 输入层：接受图像数据，形状为 $(N, H_{in}, W_{in}, C_{in})$，其中 N 是批量大小，H_{in} 和 W_{in} 是图像的高度和宽度，C_{in} 是输入通道数（例如 RGB 图像的 $C_{in} = 3$）。
- 卷积层 1 (Conv1)：第一层卷积操作，包含卷积、批归一化（Batch Normalization）、ReLU 激活和最大池化（Max Pooling）。
- 卷积层 2 (Conv2)：第二层卷积操作，结构与第一层类似。
- 全连接层 (Fully Connected Layer)：将池化后的特征展平并通过全连接层输出分类概率。
- 输出层：使用 Softmax 函数将全连接层的输出转换为概率分布。

整个网络通过前向传播计算预测概率，通过反向传播计算梯度并更新参数，最终实现图像分类。

4.2 ReLU 激活函数

我们在每层卷积层后添加了激活函数 ReLU，表达式如下

$$\mathbf{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

而通常的神经元的输出 f 表示为其输入 x 的函数的方法是使用饱和非线性函数 $f(x) = tanh(x)$ 或 $f(x) = \mathbf{sigmoid}(x) = (1 + e^{-x})^{-1}$。相比以往的方法，ReLU 的优势在于其

1. 不会发生梯度消失问题。下图分别为 sigmoid 和 tanh 的导数图像，可以发现，在 $x > 5$ 的条件下，导数值 $\frac{df}{dx} \to 0$ 即梯度消失。而对于 $max(x)$ 函数，其导数为分段函数，在 $x > 0$ 的范围内始终为 1 为固定的梯度，因此不会出现梯度消失问题

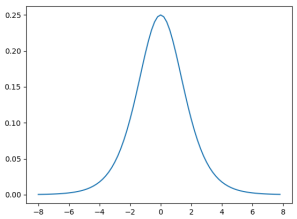

图 2: sigmoid 导数图像

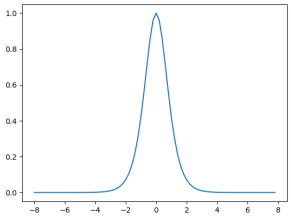

图 3: tanh 导数图像
w

2. ReLU 函数简单，梯度计算计算量小，速度快，而 sigmoid 和 tanh 均存在指数梯度，计算量较大。

4.3 批正则化层

介绍 批正则化（Batch Normalization）是一种有效的技术，用于缓解深度神经网络训练中的内部协变量偏移（Internal Covariate Shift）。通过对每一层的输入进行归一化，使其具有零均值和单位方差，批归一化可以加速训练、稳定梯度并增强网络鲁棒性。本节介绍批归一化的前向传播和反向传播过程，基于其在卷积神经网络中的实现。

在深度神经网络中，每一层的输入分布可能会随着训练过程而发生变化，这种现象被称为“内部协变量偏移”（Internal Covariate Shift）。例如，如果输入数据的分布发生变化，那么网络的权重和偏置参数需要不断调整以适应新的分布，这会导致训练过程变得非常缓慢且不稳定。

前向传播 批归一化接收输入 $x \in \mathbb{R}^{N \times H \times W \times C}$，其中 N 为批量大小，H, W 为特征图高宽，C 为通道数。归一化过程沿批量维度计算每个通道的均值和方差，然后对输入进行标准化，最后通过可学习的缩放参数 $\gamma \in \mathbb{R}^{1 \times H \times W \times C}$

和偏移参数 $\beta \in \mathbb{R}^{1\times H\times W\times C}$ 调整输出。计算公式如下：

$$\begin{aligned}
\mu_c &= \frac{1}{N}\sum_{n=1}^{N} x_{n,h,w,c},\\
\sigma_c^2 &= \frac{1}{N}\sum_{n=1}^{N}(x_{n,h,w,c}-\mu_c)^2,\\
\hat{x}_{n,h,w,c} &= \frac{x_{n,h,w,c}-\mu_c}{\sqrt{\sigma_c^2+\varepsilon}},\\
y_{n,h,w,c} &= \gamma_{h,w,c}\hat{x}_{n,h,w,c}+\beta_{h,w,c},
\end{aligned}$$

其中：- μ_c 和 σ_c^2 分别为第 c 个通道的均值和方差；- ε 是一个小常数（例如 10^{-5}），用于避免除零；- $\hat{x}$ 为标准化后的输入，y 为最终输出。

实现中，代码沿批量轴 (N) 计算均值和方差，并对每个通道独立应用归一化和缩放操作。

反向传播 反向传播计算输入梯度 $\frac{\partial L}{\partial x}$，缩放参数梯度 $\frac{\partial L}{\partial \gamma}$ 和偏移参数梯度 $\frac{\partial L}{\partial \beta}$. 给定输出梯度 $\frac{\partial L}{\partial y} \in \mathbb{R}^{N\times H\times W\times C}$，梯度计算公式如下：

$$\begin{aligned}
\frac{\partial L}{\partial \gamma_{h,w,c}} &= \sum_{n=1}^{N}\frac{\partial L}{\partial y_{n,h,w,c}}\hat{x}_{n,h,w,c},\\
\frac{\partial L}{\partial \beta_{h,w,c}} &= \sum_{n=1}^{N}\frac{\partial L}{\partial y_{n,h,w,c}},\\
\frac{\partial L}{\partial \hat{x}_{n,h,w,c}} &= \frac{\partial L}{\partial y_{n,h,w,c}}\gamma_{h,w,c},\\
\frac{\partial L}{\partial \sigma_c^2} &= \sum_{n=1}^{N}\frac{\partial L}{\partial \hat{x}_{n,h,w,c}}(x_{n,h,w,c}-\mu_c)\left(-\frac{1}{2}\right)(\sigma_c^2+\varepsilon)^{-3/2},\\
\frac{\partial L}{\partial \mu_c} &= \sum_{n=1}^{N}\frac{\partial L}{\partial \hat{x}_{n,h,w,c}}\left(-\frac{1}{\sqrt{\sigma_c^2+\varepsilon}}\right)+\frac{\partial L}{\partial \sigma_c^2}\left(-\frac{2}{N}\sum_{n=1}^{N}(x_{n,h,w,c}-\mu_c)\right),\\
\frac{\partial L}{\partial x_{n,h,w,c}} &= \frac{\partial L}{\partial \hat{x}_{n,h,w,c}}\frac{1}{\sqrt{\sigma_c^2+\varepsilon}}+\frac{\partial L}{\partial \sigma_c^2}\frac{2(x_{n,h,w,c}-\mu_c)}{N}+\frac{\partial L}{\partial \mu_c}\frac{1}{N}.
\end{aligned}$$

实现中，代码首先计算 γ 和 β 的梯度，然后通过链式法则逐步推导标准化输入、方差和均值的梯度，最终得到输入梯度 $\frac{\partial L}{\partial x}$. 计算过程利用向量化操作，避免显式循环。

4.4 卷积层

介绍 卷积层是卷积神经网络（CNN）的核心组件，用于提取输入数据的空间特征。卷积操作通过滑动窗口机制，将卷积核与输入数据进行局部加权求和，生成特征图。本节介绍卷积层的前向传播和反向传播过程，包括其数学原理和实现方法。

前向传播 卷积层的前向传播通过卷积核对输入数据进行卷积运算，并添加偏置项，生成输出特征图。对于输入数据 $X \in \mathbb{R}^{N\times H_{\text{in}}\times W_{\text{in}}\times C_{\text{in}}}$，卷积核 $W \in \mathbb{R}^{F\times kH\times kW\times C_{\text{in}}}$，偏置 $b \in \mathbb{R}^{F}$，输出特征图 $Z \in \mathbb{R}^{N\times H_{\text{out}}\times W_{\text{out}}\times F}$，其计算公式如下：

$$
\begin{aligned}
H_{\text{out}} &= \left\lfloor \frac{H_{\text{in}} + 2\cdot\text{padding} - kH}{\text{stride}} \right\rfloor + 1, \\
W_{\text{out}} &= \left\lfloor \frac{W_{\text{in}} + 2\cdot\text{padding} - kW}{\text{stride}} \right\rfloor + 1, \\
Z[n,h,w,f] &= \sum_{i=0}^{kH-1}\sum_{j=0}^{kW-1}\sum_{c=0}^{C_{\text{in}}-1} X_{\text{padded}}[n, h\cdot\text{stride}+i, w\cdot\text{stride}+j, c] \\
&\qquad\qquad \cdot W[f,i,j,c] + b[f],
\end{aligned}
$$

其中：

- N 为批量大小，$H_{\text{in}}, W_{\text{in}}$ 为输入高宽，C_{in} 为输入通道数；
- F 为卷积核数量，kH, kW 为卷积核高宽；
- stride 为步幅，padding 为填充大小；
- X_{padded} 为填充后的输入，形状为 $\mathbb{R}^{N\times(H_{\text{in}}+2\cdot\text{padding})\times(W_{\text{in}}+2\cdot\text{padding})\times C_{\text{in}}}$；
- $b[f]$ 为第 f 个卷积核的偏置。

实现中，代码通过 `as_strided` 提取滑动窗口，并利用 `tensordot` 高效计算卷积，优化了性能。

反向传播 反向传播计算输入梯度 $\frac{\partial L}{\partial X}$, 卷积核梯度 $\frac{\partial L}{\partial W}$ 和偏置梯度 $\frac{\partial L}{\partial b}$. 给定输出梯度 $\frac{\partial L}{\partial Z} \in \mathbb{R}^{N\times H_{\text{out}}\times W_{\text{out}}\times F}$, 梯度计算公式如下:

$$\begin{aligned}
\frac{\partial L}{\partial b[f]} &= \sum_{n=0}^{N-1}\sum_{h=0}^{H_{\text{out}}-1}\sum_{w=0}^{W_{\text{out}}-1}\frac{\partial L}{\partial Z[n,h,w,f]},\\
\frac{\partial L}{\partial W[f,i,j,c]} &= \sum_{n=0}^{N-1}\sum_{h=0}^{H_{\text{out}}-1}\sum_{w=0}^{W_{\text{out}}-1}\frac{\partial L}{\partial Z[n,h,w,f]}\\
&\qquad \cdot X_{\text{padded}}[n, h\cdot\text{stride}+i, w\cdot\text{stride}+j, c].\\
\frac{\partial L}{\partial X_{\text{padded}}[n,h',w',c]} &= \sum_{f=0}^{F-1}\sum_{i=0}^{kH-1}\sum_{j=0}^{kW-1}\frac{\partial L}{\partial Z[n,h,w,f]}\cdot W[f,i,j,c],
\end{aligned}$$

其中 $h' = h\cdot\text{stride}+i$, $w' = w\cdot\text{stride}+j$, 且需考虑滑动窗口的对应关系。最终, $\frac{\partial L}{\partial X}$ 从 $\frac{\partial L}{\partial X_{\text{padded}}}$ 中裁剪掉填充部分。也可以简写作

$$\begin{aligned}
\frac{\partial L}{\partial F} &= \text{Convolution}\left(\text{Input_}X, \text{Loss_gradient_}\frac{\partial L}{\partial O}\right)\\
\frac{\partial L}{\partial X} &= \text{Full_Convolution}\left(180°\text{rotated_}F_{\text{Filter}}, \text{Loss_gradient_}\frac{\partial L}{\partial O}\right)
\end{aligned}$$

实现中, 偏置梯度通过 `sum` 沿 $N, H_{\text{out}}, W_{\text{out}}$ 轴求和; 卷积核梯度通过 `tensordot` 计算窗口与输出梯度的乘积; 输入梯度通过旋转卷积核 (180 度翻转) 与填充后的输出梯度进行卷积计算。

4.5 池化层

介绍 最大池化层通过在输入特征图上应用固定大小的滑动窗口, 提取每个窗口内的最大值, 降低空间分辨率, 减少计算量并增强平移不变性。本节介绍最大池化层的前向传播和反向传播过程。

前向传播 最大池化层接收输入 $X \in \mathbb{R}^{N\times H_{\text{in}}\times W_{\text{in}}\times C}$, 输出 $Z \in \mathbb{R}^{N\times H_{\text{out}}\times W_{\text{out}}\times C}$, 其中:

$$\begin{aligned}
H_{\text{out}} &= \left\lfloor \frac{H_{\text{in}} - \text{pool_size}}{\text{stride}} \right\rfloor + 1,\\
W_{\text{out}} &= \left\lfloor \frac{W_{\text{in}} - \text{pool_size}}{\text{stride}} \right\rfloor + 1,\\
Z[n,h,w,c] &= \max_{i=0}^{\text{pool_size}-1}\ \max_{j=0}^{\text{pool_size}-1} X[n, h\cdot\text{stride}+i, w\cdot\text{stride}+j, c].
\end{aligned}$$

其中:

- pool_size 为池化窗口大小；
- stride 为步幅，通常等于 pool_size。

实现中，代码通过 `as_strided` 提取池化窗口，沿窗口维度使用 `max` 计算最大值，并记录最大值位置（掩码）以供反向传播使用。

反向传播 最大池化层的反向传播将输出梯度 $\frac{\partial L}{\partial Z} \in \mathbb{R}^{N\times H_{\text{out}}\times W_{\text{out}}\times C}$ 传递到输入梯度 $\frac{\partial L}{\partial X} \in \mathbb{R}^{N\times H_{\text{in}}\times W_{\text{in}}\times C}$. 由于最大池化仅保留窗口内最大值位置的梯度，其余位置梯度为零，计算公式为：

$$\frac{\partial L}{\partial X[n,h'+i,w'+j,c]} = \begin{cases} \frac{\partial L}{\partial Z[n,h,w,c]}, & \text{if } (i,j) \text{ 为窗口内最大值位置,} \\ 0, & \text{otherwise,} \end{cases}$$

其中 $h' = h \cdot \text{stride}$, $w' = w \cdot \text{stride}$.

实现中，代码利用前向传播生成的掩码（`pool_mask`），将输出梯度直接分配到最大值位置，其余位置保持零，避免显式循环，提高效率。

4.6 全连接层

介绍 全连接层（Fully Connected Layer）通常位于卷积神经网络的末端，用于将卷积层或池化层提取的特征映射到分类或回归任务的输出空间。全连接层将输入展平为一维向量，并通过矩阵乘法和偏置加法生成输出。本节介绍全连接层的前向传播和反向传播过程，包括其数学原理和实现方法。

前向传播 全连接层接收输入 $X \in \mathbb{R}^{N\times D}$, 其中 N 为批量大小，D 为输入特征维度。权重矩阵为 $W \in \mathbb{R}^{D\times K}$, 偏置为 $b \in \mathbb{R}^{1\times K}$, 其中 K 为输出维度（例如分类任务中的类别数）。输出 $Z \in \mathbb{R}^{N\times K}$ 的计算公式如下：

$$Z = XW + b, \tag{1}$$

其中：- XW 表示矩阵乘法，生成 $N \times K$ 的矩阵；- b 通过广播机制逐元素加到每一行。

实现中，代码通过 `dot` 函数高效计算矩阵乘法，并确保偏置的形状正确以支持广播。

反向传播 反向传播计算输入梯度 $\frac{\partial L}{\partial X}$, 权重梯度 $\frac{\partial L}{\partial W}$ 和偏置梯度 $\frac{\partial L}{\partial b}$. 给定输出梯度 $\frac{\partial L}{\partial Z} \in \mathbb{R}^{N\times K}$, 梯度计算公式如下：

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Z} W^T, \tag{2}$$

$$\frac{\partial L}{\partial W} = \frac{1}{N} X^T \frac{\partial L}{\partial Z}, \tag{3}$$

$$\frac{\partial L}{\partial b} = \frac{1}{N} \sum_{n=0}^{N-1} \frac{\partial L}{\partial Z[n,:]}, \tag{4}$$

其中：- $\frac{\partial L}{\partial X}$ 的形状为 $N \times D$, 通过输出梯度与权重转置的矩阵乘法计算；- $\frac{\partial L}{\partial W}$ 的形状为 $D \times K$, 通过输入转置与输出梯度的矩阵乘法计算，并除以批量大小 N 以平均梯度；- $\frac{\partial L}{\partial b}$ 的形状为 $1 \times K$, 通过沿批量维度求和并平均。

实现中，代码使用 `dot` 进行矩阵乘法，`sum` 沿批量轴计算偏置梯度，并通过形状调整确保偏置梯度的正确性。

4.7 Softmax 层与交叉熵损失

介绍 Softmax 层将全连接层的输出转换为概率分布，常用于多分类任务。结合交叉熵损失（Cross-Entropy Loss），它衡量模型预测概率与真实标签之间的差异。本节介绍 Softmax 层的前向传播、交叉熵损失的计算及其反向传播过程，包括正则化项的影响。

Softmax 前向传播 Softmax 层接收输入 $z \in \mathbb{R}^{N\times K}$, 其中 N 为批量大小，K 为类别数。输出 $a \in \mathbb{R}^{N\times K}$ 为概率分布，计算公式如下：

$$a_{n,k} = \frac{\exp(z_{n,k} - \max_k z_{n,k})}{\sum_{j=1}^{K} \exp(z_{n,j} - \max_k z_{n,k})}, \tag{5}$$

其中：- $\max_k z_{n,k}$ 减去输入最大值以提高数值稳定性；- $a_{n,k}$ 表示第 n 个样本属于第 k 类的预测概率，且 $\sum_{k=1}^{K} a_{n,k} = 1$。

实现中，代码通过 `cp.exp` 和 `cp.sum` 计算指数和归一化，利用向量化操作高效生成概率分布。

交叉熵损失计算 交叉熵损失接收 Softmax 层的输出 $a \in \mathbb{R}^{N\times K}$ 和真实标签 $y \in \{1, \ldots, K\}^N$, 其中 y_n 表示第 n 个样本的类别。损失函数包括数据损

失和正则化损失，计算公式如下：

$$L_{\text{data}} = -\frac{1}{N}\sum_{n=1}^{N}\log(a_{n,y_n}+\varepsilon), \tag{6}$$

$$L_{\text{reg}} = \frac{\lambda}{2}\sum_{l}\sum_{i,j}W_{l,i,j}^2, \tag{7}$$

$$L = L_{\text{data}} + L_{\text{reg}}, \tag{8}$$

其中：- ε 是一个小常数（例如 10^{-10}），用于避免对数运算的数值不稳定；- λ 为正则化系数，W_l 表示网络中所有权重矩阵；- L_{data} 为交叉熵损失，L_{reg} 为 L2 正则化项。

实现中，代码使用 `cp.clip` 限制概率值以确保数值稳定性，并通过 `cp.sum` 计算所有权重矩阵的平方和。

反向传播 反向传播计算损失对 Softmax 输入的梯度 $\frac{\partial L}{\partial z}$. 对于交叉熵损失结合 Softmax，梯度具有简洁形式。令 $\delta_{n,k} = \frac{\partial L}{\partial z_{n,k}}$, 则：

$$\delta_{n,k} = \frac{\partial L_{\text{data}}}{\partial z_{n,k}} = a_{n,k} - \mathbb{I}\{y_n = k\}, \tag{9}$$

其中 $\mathbb{I}\{y_n = k\}$ 为指示函数，当 $y_n = k$ 时为 1，否则为 0。正则化损失对 z 无直接贡献，但对权重 W_l 的梯度为：

$$\frac{\partial L_{\text{reg}}}{\partial W_{l,i,j}} = \lambda W_{l,i,j}. \tag{10}$$

实现中，代码通过复制 Softmax 输出并对正确类别减 1 （`delta3 = probs.copy(); delta3[range(N), y] -= 1`）计算梯度，利用向量化操作高效传递到前一层。正则化梯度在权重更新时加入。

5 训练过程

5.1 优化算法

介绍 网络参数的优化采用随机梯度下降（Stochastic Gradient Descent, SGD），通过小批量数据计算梯度并更新参数，以最小化损失函数。本节介绍 SGD 的更新规则及其实现。

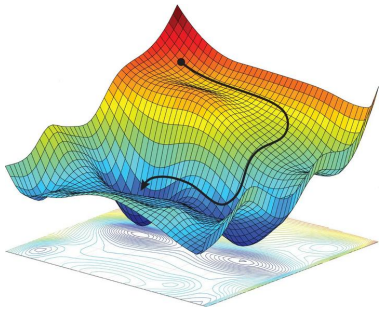

图 4: SGD 随机梯度下降

参数更新 SGD 基于损失函数 L 对参数 θ（包括权重 W 和偏置 b）的梯度进行更新。对于小批量数据，参数更新规则如下：

$$\theta \leftarrow \theta - \eta \left(\frac{\partial L_{\text{data}}}{\partial \theta} + \lambda \theta \right), \tag{11}$$

其中：

- η 为学习率，控制更新步长；
- $\frac{\partial L_{\text{data}}}{\partial \theta}$ 为数据损失的梯度；
- $\lambda\theta$ 为 L2 正则化项，λ 为正则化系数。

为防止梯度爆炸，采用梯度裁剪：

$$\text{if } \|\nabla\theta\|_2 > \tau, \quad \nabla\theta \leftarrow \nabla\theta \cdot \frac{\tau}{\|\nabla\theta\|_2}, \tag{12}$$

其中 τ 为最大梯度范数。

实现中，代码对每个参数（如权重和偏置）独立应用上述规则，利用向量化操作高效更新，并通过正则化项和梯度裁剪增强训练稳定性。

与传统 GD 的区别 Mini-batch GD（小批量梯度下降）为 SGD 的特殊的一种。与传统 GD（梯度下降）的区别在于，Mini-batch GD 每次迭代只使用小批量数据进行梯度计算和参数更新，而传统 GD 使用整个训练数据集，导致 Mini-batch GD 在计算效率、更新频率和内存需求上更有优势，适合大数据集和 GPU 并行计算，且其梯度噪声有助于跳出局部最优，提高泛化能力，但传统 GD 的梯度估计更准确，适合小数据集或追求精确梯度的场景。

5.2 学习率下降策略

介绍 学习率是优化算法（如随机梯度下降）中的关键超参数，控制参数更新的步长。过高的学习率可能导致训练不稳定，而过低的学习率会减慢收敛速度。学习率下降策略通过动态调整学习率，平衡训练初期的快速收敛和后期的精细调整。本节介绍一种常见学习率下降策略的原理及其实现。

策略原理 假定初始学习率为 lr, 学习率下降策略根据训练进度调整学习率 lr_t。本实验采取阶梯衰减（Step Decay），公式为：

$$\eta_t = \eta_0 \cdot \gamma^{\lfloor \frac{t}{S} \rfloor} \tag{13}$$

其中 $lr\_delay > 0$ 为衰减率，控制学习率下降的速度。

在准确率和损失曲线中，都可以看出在 $epoch = 5, 10, 15$ 时，曲线都有呈阶梯状的趋势

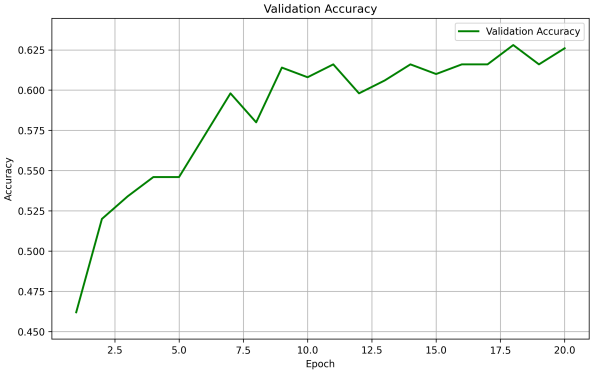

图 5: 验证集准确率

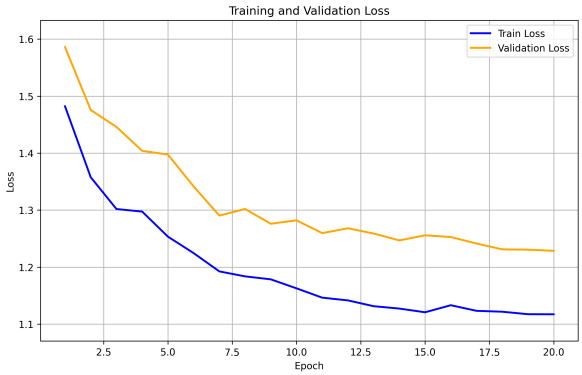

图 6: loss 曲线

5.3 L2 正则化实现

介绍 L2 正则化通过在损失函数中加入权重平方和的惩罚项，限制模型复杂度，防止过拟合。结合随机梯度下降，L2 正则化在参数更新时引入额外的梯度项。本节介绍 L2 正则化的原理及其在代码中的实现。

原理 L2 正则化在原始损失函数 L_{data} 上添加正则化项，定义总损失为：

$$L = L_{\text{data}} + \frac{\lambda}{2} \sum_{l} \sum_{i,j} W_{l,i,j}^2, \tag{14}$$

其中：

- L_{data} 为数据损失（交叉熵损失）；
- λ 为正则化系数，控制正则化强度；
- W_l 表示第 l 层权重矩阵。

对权重 $W_{l,i,j}$ 的梯度为：

$$\frac{\partial L}{\partial W_{l,i,j}} = \frac{\partial L_{\text{data}}}{\partial W_{l,i,j}} + \lambda W_{l,i,j}. \tag{15}$$

在随机梯度下降中，权重更新规则为：

$$W_{l,i,j} \leftarrow W_{l,i,j} - \eta \left(\frac{\partial L_{\text{data}}}{\partial W_{l,i,j}} + \lambda W_{l,i,j} \right), \tag{16}$$

其中 η 为学习率。

5.4 训练过程中的超参数设置

介绍 超参数决定了模型的架构和训练行为，直接影响性能和收敛性。合理的超参数设置需要在实验中权衡计算成本和模型效果。本节总结训练过程中使用的关键超参数及其设置方法。

超参数列表 基于代码实现，训练涉及以下超参数：

- **学习率** η: 控制参数更新步长，代码中默认设置为 `lr=0.01`。
- **正则化系数** λ: 控制 L2 正则化强度，代码中默认设置为 `reg=0.001`。
- **最大梯度范数** τ: 用于梯度裁剪，防止梯度爆炸，代码中默认设置为 `max_grad_norm=5.0`。
- **批量大小** N: 每次迭代处理的样本数，通常根据内存和计算资源选择（例如 32 或 64）。
- **训练轮数** T: 整个数据集的遍历次数（epoch），需通过实验确定。
- **卷积核大小** (kH, kW): 代码中第一层和第二层卷积核默认设置为 $(3, 3)$。
- **卷积核数量** F: 代码中第一层和第二层分别设置为 64(`conv1_filters=64`, `conv2_filters=64`)。

5.5 训练过程中的损失值记录与分析

介绍 损失值是衡量模型训练效果的核心指标，反映了预测与真实标签之间的差距。记录和分析损失值有助于监控训练进度、诊断模型性能并调整超参数。本节介绍损失值的记录方法及其分析过程。

记录方法 训练过程中，损失值通过 `compute_loss` 方法计算，包括数据损失和正则化损失：

$$L = -\frac{1}{N}\sum_{n=1}^{N}\log(a_{n,y_n} + \varepsilon) + \frac{\lambda}{2}\sum_{l}\sum_{i,j}W_{l,i,j}^2, \tag{17}$$

实现中，训练和验证损失通常存储在列表或数组中(例如 `loss_history`)，结合可视化工具（如 Matplotlib）绘制曲线，辅助超参数调优和模型选择。

5.6 最优模型权重的保存与加载机制

介绍 训练过程中，模型权重会随迭代更新，保存最优权重可确保在验证性能最佳时保留模型状态。加载保存的权重则允许恢复训练或用于推理。本节介绍最优模型权重的保存与加载机制。

保存机制 最优模型权重基于验证集性能（例如最低验证损失或最高验证准确率）保存。保存流程如下：

- 在每个 epoch 结束时，计算验证集上的准确率。
- 如果当前准确率高于历史最高值，保存所有参数($W_1, b_1, W_2, b_2, W_3, b_3, \gamma_1, \beta_1, \gamma_2, \beta_2$)。
- 参数存储为 NumPy 的 `.npy` 格式，记录为检查点（checkpoint）。

保存的检查点包括：

$$\text{checkpoint} = \{W_1, b_1, W_2, b_2, W_3, b_3, \gamma_1, \beta_1, \gamma_2, \beta_2\}. \tag{18}$$

加载机制 加载机制从检查点恢复模型状态，流程如下：

- 读取保存的二进制文件，获取参数值。
- 将加载的参数赋值给模型的对应变量（例如 `self.W1 = loaded_W1`）。
- 确保模型架构（如卷积核数量、输入尺寸）与保存时一致。

实现中，保存和加载通过 `np.save` 和 `np.load` 完成，支持训练中断后的恢复或部署时的快速推理。

6 参数查找与模型性能分析

6.1 超参数调节

介绍 超参数调节是优化神经网络性能的关键步骤，直接影响模型的收敛速度和泛化能力。超参数包括学习率、正则化系数、隐藏层大小等，需要通过实验系统地搜索最优组合。本节介绍超参数调节的通用方法及其应用。

调节方法 超参数调节采用**网格搜索**（Grid Search）：定义超参数的离散候选值集合（如学习率 $\{0.1, 0.01, 0.001\}$），穷举所有组合，训练模型并评估验证集性能

性能评估通常基于验证集的损失或准确率，选出泛化能力最强的超参数组合。注意，在训练过程中发现，如果超参数调节过程中的 epochs 选取过小，例如只有 5 时，容易选取到过拟合的超参数（即前几轮收敛较快，但之后收敛能力有限的情况，所以 epochs 要尽量与最终训练的 epochs 相同）

6.2 学习率调节

学习率 η 控制参数更新的步长，是影响训练收敛的关键超参数。过高的学习率可能导致损失震荡或发散，过低的学习率则会减慢收敛。本节介绍学习率调节的方法及其对模型性能的影响。

6.3 正则化强度调节

介绍 正则化强度通过控制 L2 正则化系数 λ 限制模型复杂度，防止过拟合。合适的 λ 需在拟合不足和过拟合之间取得平衡。本节介绍正则化强度的调节方法及其影响。

调节方法 正则化系数 λ 的调节采用以下步骤：

- **范围选择**：测试对数尺度上的候选值（如 $\lambda \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$）。
- **验证集评估**：对每个 λ 训练模型，比较训练和验证损失的差距。较小的 λ 可能导致过拟合，较大的 λ 可能导致欠拟合。
- **交叉验证**：在多个数据划分上评估 λ，确保泛化性能稳定。

正则化强度的选择需结合任务复杂度和数据集大小。

调节过程通过测试不同 λ（如 $\{10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}\}$），记录验证集损失和准确率，选择使验证性能最佳的值。实现利用 `reg_lambda` 参数灵活调整正则化强度。

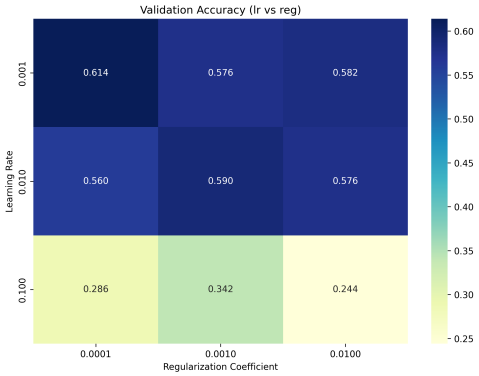

图 7: lr—reg 参数调节

如图，可以得到当正则化系数 $\lambda = 0.0001$，学习率为 $lr = 0.001$ 时最好

6.4 隐藏层大小调节

介绍 隐藏层大小决定了模型的容量，直接影响特征提取能力和计算成本。在卷积神经网络中，隐藏层大小包括卷积核数量。本节介绍隐藏层大小的调节方法及其对性能的影响。

调节方法 隐藏层大小调节需考虑以下因素：

- **卷积核数量** F：增加 F（如从 32 到 64）可增强特征提取能力，但增加计算成本和过拟合风险，在具体实现中发现，当 $F = 32$ 时，训练一个 epoch 只需要花 30s，但当 $F = 128$ 时，训练一个 epoch 就需要花费 4min 的时间，这极大的提高了计算成本。若选择卷积层为 256 则会出现 GPU 内显存爆满的情况。
- **权衡复杂性**：较大隐藏层适合复杂任务（如高分辨率图像分类），较小隐藏层适合简单任务或受限计算资源。

调节通常或倍增法（如 $F \in \{16, 32, 64, 128\}$）确定。

结果通过表格或图表总结，分析超参数对性能的权衡。例如，较高的 F 可能提升准确率但增加时间，较大的 λ 可能降低过拟合但牺牲训练损失。最终选择验证准确率最高且训练时间合理的配置。

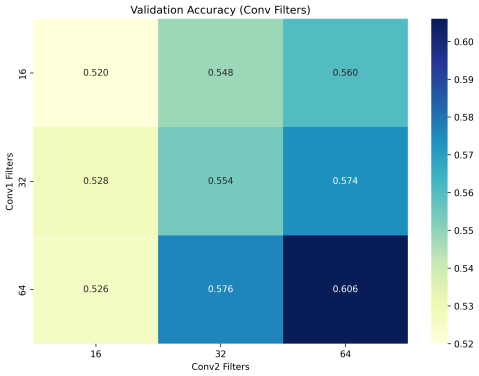

图 8: 卷积层数参数调节

7 测试与评估

7.1 训练集和验证集上的损失曲线

训练和验证损失曲线用于评估模型的收敛性和泛化能力。损失函数为交叉熵损失加 L2 正则化项：

$$L = -\frac{1}{N}\sum_{n=1}^{N}\log(a_{n,y_n}+\varepsilon) + \frac{\lambda}{2}\sum_{l}\sum_{i,j}W_{l,i,j}^2, \tag{19}$$

其中 N 为批量大小，a_{n,y_n} 为 Softmax 输出对应真实类别的概率，ε 防止数值不稳定。

绘制方法：每 epoch 记录训练集损失 L_{train} 和验证集损失 L_{val}，使用 Matplotlib 绘制曲线，横轴为 epoch 数，纵轴为损失值。

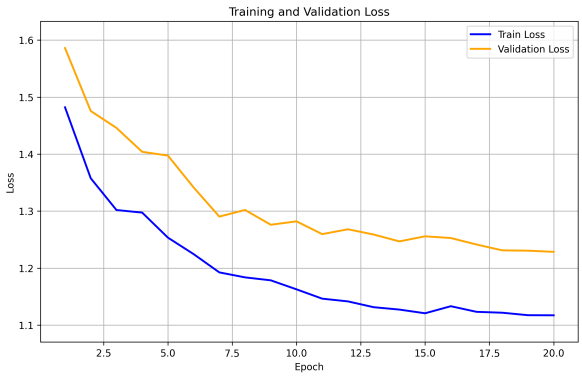

图 9: 训练集和验证集上的损失曲线

结果与分析：

- L_{train} 持续下降，表明模型在训练数据上收敛良好。
- L_{val} 初始下降后趋于平稳，验证集损失略高于训练集，提示轻微过拟合。
- 未观察到损失震荡，学习率设置合理。

7.2 验证集上的准确率曲线

验证集准确率反映模型分类性能，计算公式为：

$$A_{\text{val}} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{I}\{\hat{y}_m = y_m\}, \tag{20}$$

其中 $\hat{y}_m = \arg\max_k a_{m,k}$ 为预测类别，M 为验证集样本数。

绘制方法：每 epoch 计算 A_{val}，使用 Matplotlib 绘制曲线，横轴为 epoch 数，纵轴为准确率。

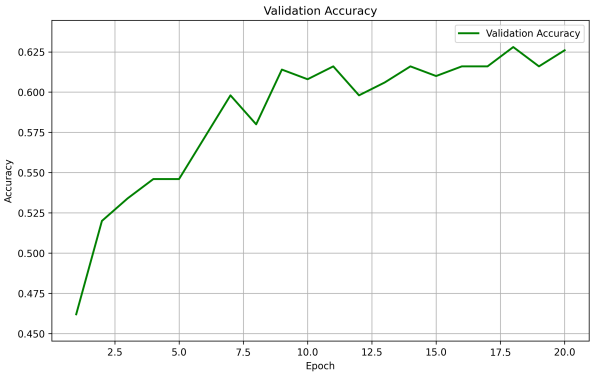

图 10: 验证集上的准确率曲线

结果与分析：

- 准确率随 epoch 增加稳步提升，最终稳定在约 62.6%。
- 未出现明显下降，表明模型泛化能力较好。

7.3 测试集上的分类准确率

测试集准确率评估模型在未见过数据上的性能，计算公式为：

$$A_{\text{test}} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{I}\{\hat{y}_m = y_m\}, \tag{21}$$

其中 $\hat{y}_m = \arg\max_k a_{m,k}$。

实现方法：模型对测试集 $X \in \mathbb{R}^{M\times H\times W\times C}$ 进行前向传播，生成 Softmax 输出 a，通过 cp.argmax$(a, \text{axis} = 1)$ 得到 $\hat{y}$，再计算准确率。

结果：测试集准确率 $A_{\text{test}} = 62.6\%$，表明模型具有较好的分类能力。

7.4 测试结果分析

总体准确率：$A_{\text{test}} = 62.6\%$，模型在测试集上表现良好，适合实际应用。

类别准确率：对每个类别 k 计算：

$$A_k = \frac{\sum_{m:y_m=k} \mathbb{I}\{\hat{y}_m = y_m\}}{\sum_{m:y_m=k} 1}. \tag{22}$$

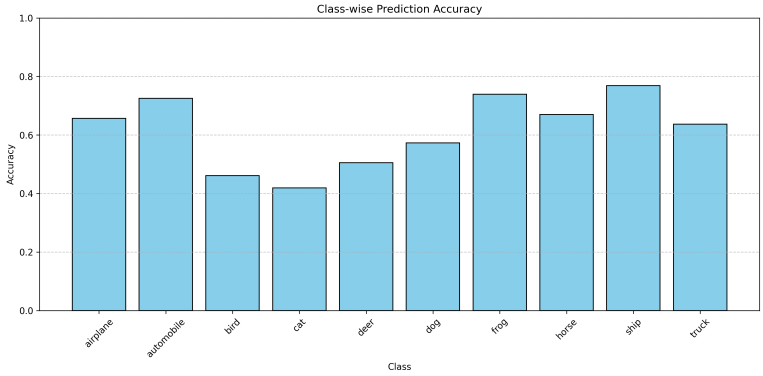

图 11: 测试集上各类别的准确率

结果显示：

- 类别 frog 和 ship 的准确率较高（约 70%），特征区分明显。
- 类别 cat 准确率较低（约 40%），可能因样本不平衡或特征混淆。

分析：模型整体性能稳定，但需优化对少数类别的分类效果，可考虑数据增强或调整损失权重。

7.5 模型性能总结

优点：

- 卷积神经网络有效提取图像特征，分类性能较强。
- 模型训练稳定，泛化能力良好。

缺点：

- 对某些类别性能不足，可能受限于模型容量或数据分布。
- 计算复杂度较高，推理时间需优化。

改进方向：增加模型深度、引入注意力机制或优化数据预处理。

8 模型网络参数可视化

8.1 网络参数的可视化方法

网络参数可视化通过图形展示权重、偏置和批归一化参数的分布与模式，帮助理解模型特征提取能力和训练状态。针对卷积神经网络的参数，可视化方法包括：

- **卷积核可视化**：将卷积层权重 $W_l \in \mathbb{R}^{F\times kH\times kW\times C}$ 的每个卷积核（$kH\times kW\times C$）绘制为图像，揭示低级和高层次特征。

实现方法：参数存储为 CuPy 数组，转换为 NumPy 后使用 Matplotlib 和 Seaborn 绘制：

8.2 参数可视化结果展示与分析

可视化结果（基于 20 个 epoch 训练）：

- **第一层卷积核**：权重 W_1（$64\times 3\times 3\times 3$）显示为 32 张图像。约 70% 卷积核呈现边缘或颜色检测模式（如水平线、垂直线），其余接近均匀。
- **第二层卷积核**：权重 W_2（$64\times 3\times 3\times 64$）显示复杂纹理或形状组合，表明学习到高级特征。

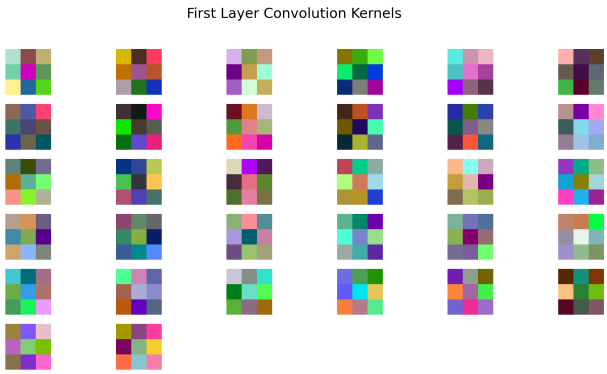

图 12: 第一层卷积核可视化

- **全连接权重热图**：对全连接层权重 $W_3 \in \mathbb{R}^{D\times K}$ 使用热图，分析特征与类别的关联。
- **参数分布直方图**：绘制所有参数（如 $W_l, b_l, \gamma_l, \beta_l$）的值分布，检查训练稳定性。

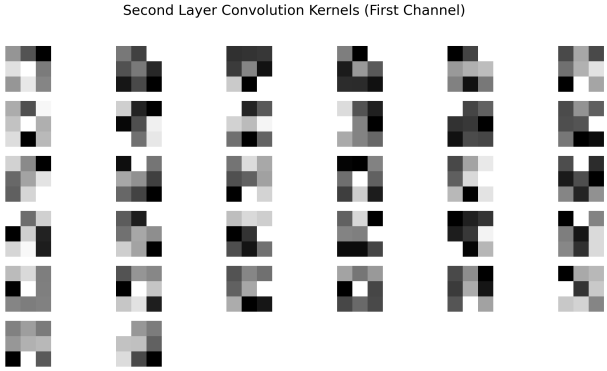

图 13: 第二层卷积核可视化

8.3 参数可视化结果展示与分析

可视化结果（基于 20 个 epoch 训练）：

- **全连接权重**：权重 W_3（$512 \times K$）热图显示部分类别与特定特征强关联。
- **参数分布**：权重的直方图近似正态，均值约 0，方差较小；偏置 b_l 和 β_l 负值

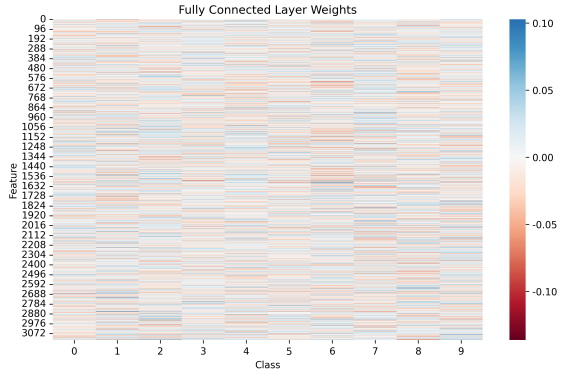

图 14: 全连接层权重热图

分析：

- **卷积核**：边缘特征表明模型适合图像任务，但部分卷积核低活性提示需延长训练或调整初始化。
- **全连接权重**：类别权重不均可能源于数据不平衡，需数据增强。

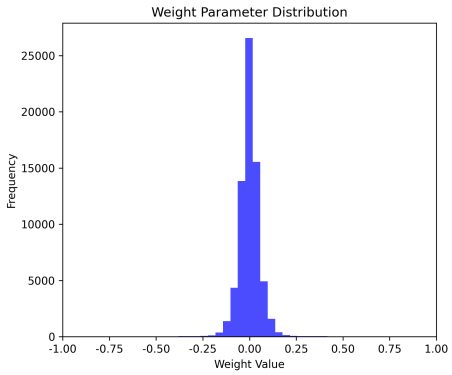

图 15: 权重参数分布直方图

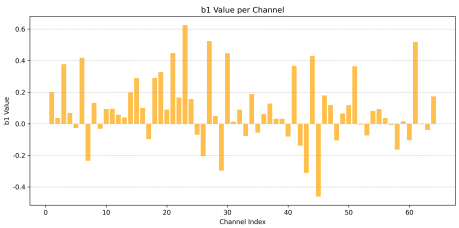

图 16: b1 参数直方图

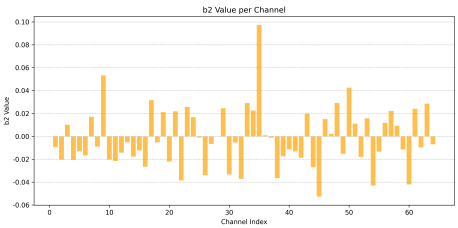

图 17: b2 参数直方图

- **参数分布**：正态分布反映训练稳定，偏置正偏可能与输入数据偏移相关。

优化建议：

- 增加训练轮数或调整权重初始化以激活低活性卷积核。
- 对类别权重稀疏问题，尝试加权损失或数据增强。
- 检查输入数据预处理，确保零均值。

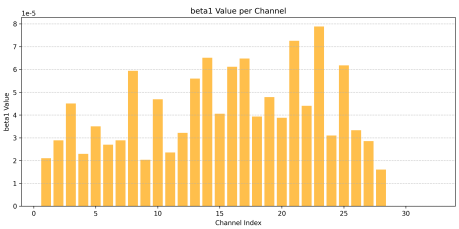

图 18: beta1 参数直方图

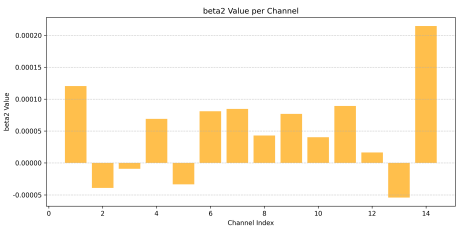

图 19: beta2 参数直方图

9 代码与资源

9.1 Github 链接

代码已上传至 Giuhub，链接为：Github

9.2 模型权重

模型权重已上传至百度网盘，best_model_weights.npz，data.npz，hyperparam_conv_filters.npz，hyperparam_lr_reg.npz，链接为：百度网盘 提取码: vyjv

附录 A 输入层、隐藏层、输出层的参数设计

模型概况

这是一个三层卷积神经网络（CNN），包括两个卷积隐藏层和一个全连接输出层，默认设计针对 CIFAR-10 数据集（32x32 的 RGB 图像，10 个类别）。以下分析基于默认参数：

- 输入形状：（32，32，3）
- 类别数：$num\_classes = 10$

A.1 输入层

- 输入维度 $(N, 32, 32, 3)$
  - N: 批量大小，表示一次处理的样本数量。
  - 32, 32：输入图像的高度和宽度（32x32 像素）。
  - 3：输入通道数，对应 RGB 图像的红、绿、蓝三通道。

A.2 隐藏层

- 卷积层 1 (Conv1)
  - **输入维度：** $(N, 32, 32, 3)$ （来自输入层）
  - **卷积核：** W_1，形状 $(32, 3, 3, 3)$ （32 个 3×3 的卷积核，输入通道数为 3）
  - **偏置：** b_1，形状 $(32, 1, 1)$

  - **参数：** 步幅 $= 1$，填充 $= 1$
  - **输出维度：** $(N, 30, 30, 32)$
    - **高度和宽度：** $(32 - 3 + 2 \times 1)/1 + 1 = 32$
    - **通道数：** 等于卷积核数量 conv1_filters $= 32$
  - **设计逻辑：**
    - 卷积核大小 3×3 是 CNN 中的常见选择，平衡了局部特征提取能力和计算成本。
    - 输出通道数 32 增加特征表达能力，同时保持计算复杂度可控。

- 批归一化 (Batch Normalization1) 与 ReLU1 层
  - **输入维度：** $(N, 32, 32, 32)$
  - **输出维度：** $(N, 32, 32, 32)$
  - **参数：** $gamma1$ 和 $beta1$，形状为 $(32, 32, 32)$

- 最大池化层 1 (Max Pooling1)
  - **输入维度：** $(N, 32, 32, 32)$
  - **参数：** 池化窗口 $=$ 2x2，步幅 $= 2$
  - **输出维度：** $(N, 16, 16, 32)$
    - **高度和宽度：** $(32 - 2)/2 + 1 = 16$
    - **通道数：** 保持不变，为 32

- 卷积层 2 (Conv2)
  - **输入维度：** $(N, 16, 16, 32)$
  - **卷积核：** W_2，形状 $(32, 3, 3, 32)$（32 个 3×3 的卷积核，输入通道数为 32）
  - **偏置：** b_2，形状 $(32, 1, 1)$
  - **参数：** 步幅 $= 1$，填充 $= 0$
  - **输出维度：** $(N, 14, 14, 32)$
    - **高度和宽度：** $(16 - 3 + 2 * 0)/1 + 1 = 14$
    - **通道数：** 等于卷积核数量 conv1_filters $= 32$
  - **设计逻辑：**

    - 卷积核大小 3×3 是 CNN 中的常见选择，平衡了局部特征提取能力和计算成本。
    - 输出通道数 32 增加特征表达能力，同时保持计算复杂度可控。

- 批归一化 2 (Batch Normalization2) 与 ReLU2 层
  - **输入维度：** $(N, 14, 14, 32)$
  - **输出维度：** $(N, 14, 14, 32)$
  - **参数：** $gamma1$ 和 $beta1$，形状为 $(14, 14, 32)$
- 最大池化层 2 (Max Pooling2)
  - **输入维度：** $(N, 14, 14, 32)$
  - **参数：**池化窗口 = 2x2，步幅 = 2
  - **输出维度：** $(N, 7, 7, 32)$
    - **高度和宽度：** $(14 - 2)/2 + 1 = 7$
    - **通道数：**保持不变，为 32

A.3 输出层

- 展平
  - **输入维度：** $(N, 7, 7, 32)$
  - **输出维度：** $(N, 1568)$
- 全连接层
  - **输入维度：** $(N, 1568)$
  - **输出维度：** $(N, 1568)$
  - **参数：**
    - **权重：**形状 (1568, 10)（输入特征数为 1568，输出类别数为 10）
    - **偏置：**形状 (1, 10)
- Softmax 激活层
  - **输入维度：** $(N, 10)$
  - **输出维度：** $(N, 10)$

参考文献

[1] X. Glorot, A. Bordes, and Y. Bengio, *Deep Sparse Rectifier Neural Networks*, Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS) 2011, Fort Lauderdale, FL, USA. Volume 15 of JMLR: W&CP 15, 2011, pp. 315–323.