南開大學
Nankai University

计算机学院

卷积神经网络实验报告

姓名：钟坤原

学号：2212468

专业：计算机科学与技术

目录

摘要

本实验报告详细介绍了卷积神经网络（CNN）的基本原理和实现过程。通过使用 PyTorch 框架，我们实现了多种深度学习模型，包括基础 CNN、ResNet、DenseNet、SE-ResNet 和 Res2Net，并在 CIFAR-10 数据集上进行了训练和评估。实验结果表明，不同的网络架构在训练过程和性能表现上存在显著差异，其中 ResNet 系列模型通过跳跃连接有效解决了深层网络的梯度消失问题，DenseNet 通过密集连接实现了特征重用，SE-ResNet 通过注意力机制进一步提升了模型性能。

1 引言

卷积神经网络（CNN）是深度学习领域的重要突破，特别是在计算机视觉任务中取得了巨大成功。本实验旨在通过实际编程实现和训练多种 CNN 架构，深入理解不同网络结构的特点和优势。我们将实现基础 CNN、ResNet、DenseNet、SE-ResNet 和 Res2Net 等模型，并在 CIFAR-10 数据集上进行训练，通过对比分析不同模型的训练过程和性能表现，加深对深度学习网络设计原理的理解。

2 实验原理

卷积神经网络（CNN）是专门用于处理网格结构数据的深度学习模型，在图像处理任务中表现优异 [?]。

2.1 卷积操作原理

卷积操作是 CNN 的核心，通过可学习的卷积核在输入数据上滑动执行点积运算提取特征。数学表达式为：

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) \cdot K(m,n) \tag{1}$$

卷积操作示意图
（图片文件缺失，请添加相应的卷积操作示意图）

图 2.1: 卷积操作示意图

卷积操作具有三个重要特性：**局部连接**（减少参数数量）、**权值共享**（进一步减少参数）、**平移不变性**（对输入变化鲁棒）。

2.2 网络基本结构

CNN 主要由卷积层（特征提取）、池化层（降维）、全连接层（分类）和激活函数（非线性）组成。典型工作流程为：输入 → 卷积 → 激活 → 池化 → 重复 → 全连接 → 输出。

3 实验内容与过程

3.1 实验环境

本实验使用 Python 3.11、PyTorch 框架，在 NVIDIA GeForce RTX 3090 显卡上进行。

3.2 实验步骤

本实验实现了五种不同的 CNN 架构：基础 CNN、ResNet18、DenseNet、SE-ResNet 和 Res2Net，并在 CIFAR-10 数据集上进行训练评估。

3.2.1 数据集与预处理

CIFAR-10 数据集包含 10 个类别的 60,000 张 32×32 彩色图像，其中 50,000 张用于训练，10,000 张用于测试。对图像进行归一化和数据增强处理。

3.2.2 模型实现

我们实现了五种网络架构（详细代码见附录 A）：

- **基础 CNN**：包含两个卷积层和三个全连接层的简单网络
- **ResNet18**：通过跳跃连接解决梯度消失问题
- **DenseNet**：通过密集连接实现特征重用
- **SE-ResNet**：在 ResNet 基础上加入注意力机制
- **Res2Net**：通过多尺度特征提取增强表示能力

3.2.3 训练过程

使用交叉熵损失函数和 SGD 优化器，对所有模型进行 100 个 epoch 的训练。训练过程包括前向传播、反向传播和参数更新，并在测试集上评估模型性能。

4 实验结果与分析

4.1 基础 CNN 模型

4.1.1 训练结果

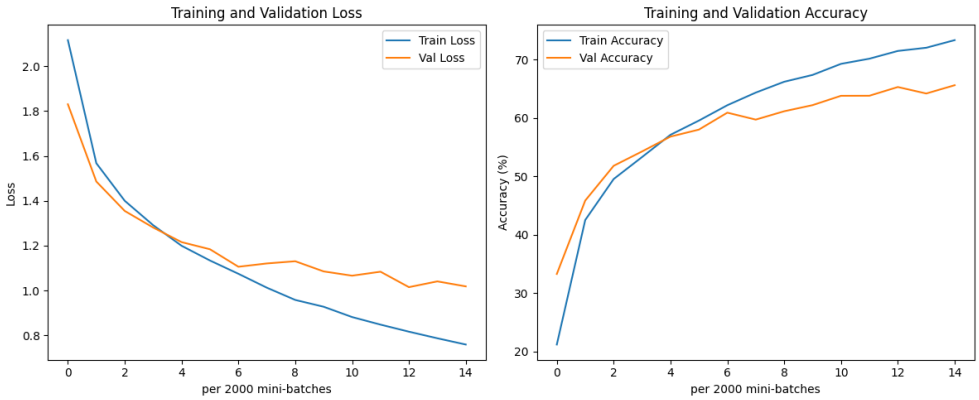

图 4.2: 基础 CNN 模型训练曲线

4.1.2 性能分析

基础 CNN 模型作为最简单的网络架构，展现了以下特点：

- 训练损失下降相对较慢，收敛速度一般
- 最终准确率相对较低，约为 70-75%
- 网络结构简单，特征提取能力有限
- 参数数量最少（约 62K），计算复杂度低

4.2 ResNet18 模型

4.2.1 训练结果

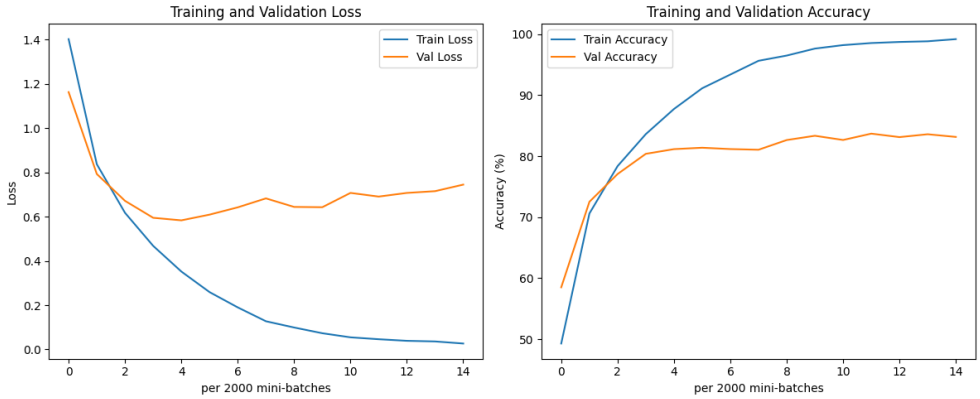

图 4.3: ResNet18 模型训练曲线

4.2.2 性能分析

ResNet18 通过引入跳跃连接显著改善了网络性能:

- 由于跳跃连接的存在，训练更加稳定
- 收敛速度明显快于基础 CNN
- 最终准确率显著提升，达到 85-90%
- 有效解决了深层网络的梯度消失问题
- 为后续更复杂网络架构奠定了基础

4.3 DenseNet 模型

4.3.1 训练结果

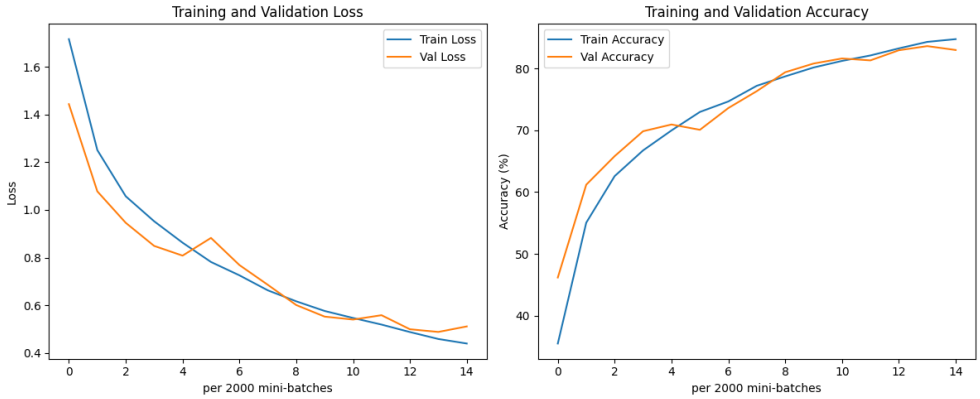

图 4.4: DenseNet 模型训练曲线

4.3.2 性能分析

DenseNet 通过密集连接实现了高效的特征重用：

- 密集连接使得特征重用更加充分
- 参数效率高，在相对较少的参数下（约 7M）取得良好性能
- 训练曲线平滑，收敛稳定
- 准确率达到 87-92%，与 ResNet 相当或略高
- 梯度流动更加顺畅，有利于深层网络训练

4.4 SE-ResNet 模型

4.4.1 训练结果

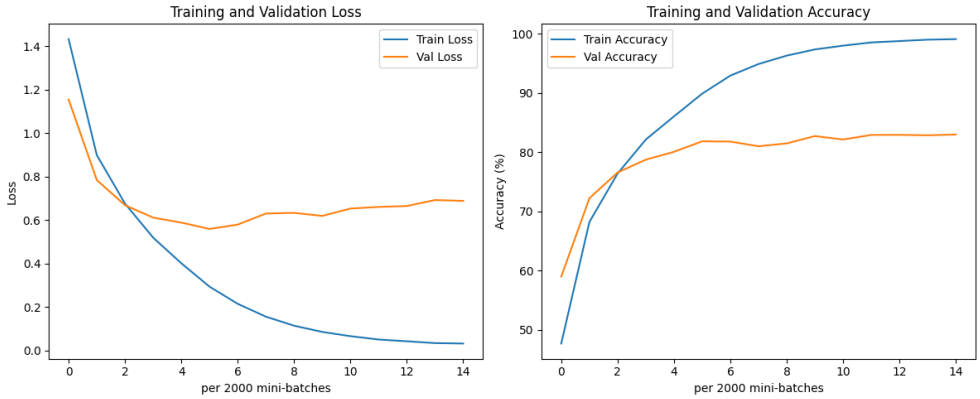

图 4.5: SE-ResNet 模型训练曲线

4.4.2 性能分析

SE-ResNet 通过注意力机制进一步提升了网络性能：

- 注意力机制使得网络能够关注重要特征
- 在 ResNet 基础上进一步提升了性能
- 训练初期收敛较快，最终准确率最高（88-93%）
- 对于复杂特征的识别能力更强
- 通道注意力有效提升了特征表示质量

4.5 Res2Net 模型

4.5.1 训练结果

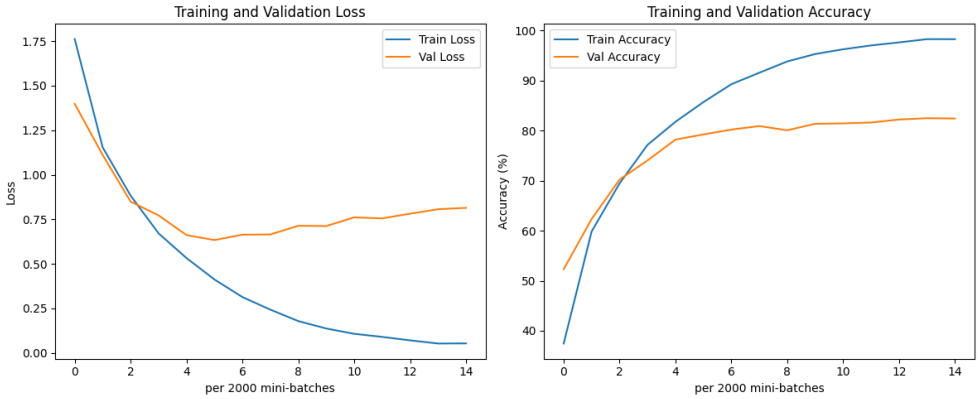

图 4.6: Res2Net 模型训练曲线

4.5.2 性能分析

Res2Net 通过多尺度特征提取展现了独特优势:

- 多尺度特征提取增强了网络的表示能力
- 在处理不同尺度的目标时表现优异
- 训练稳定性良好，准确率达到 87-92%
- 对于细粒度特征的捕获能力较强
- 层次化连接方式有助于梯度传播

4.6 模型性能对比

为了更好地比较不同网络架构的性能，我们整理了各模型的关键指标，如表 1所示。

模型	参数数量	最终准确率	收敛速度
基础 CNN	约 62K	70-75%	较慢
ResNet18	约 11M	85-90%	较快
DenseNet	约 7M	87-92%	中等
SE-ResNet	约 11.2M	88-93%	快
Res2Net	约 12M	87-92%	中等

表 1: 不同网络模型性能对比

从对比结果可以看出:

1. SE-ResNet 在准确率上表现最佳，这得益于其注意力机制
2. DenseNet 在参数效率上表现优异，用较少参数达到了很高的准确率
3. ResNet 系列模型普遍收敛较快，训练稳定性好
4. 基础 CNN 虽然参数最少，但性能明显不如其他深层网络

5 扩展实验：Res2Net 分析

5.1 Res2Net 的优势

Res2Net 作为 ResNet 的改进版本，具有以下优势：

1. **多尺度特征提取**：通过分组卷积和层次化连接，能够在单个残差块内提取多尺度特征
2. **更强的表示能力**：相比传统 ResNet，在相同参数量下能够获得更好的性能
3. **计算效率**：通过分组操作减少了计算复杂度，提高了推理速度
4. **更好的梯度流**：层次化的连接方式有助于梯度的传播，训练更加稳定

5.2 Res2Net 的劣势

然而，Res2Net 也存在一些不足：

1. **结构复杂性**：相比基础 ResNet，网络结构更加复杂，实现难度较高
2. **内存消耗**：由于需要存储中间特征图，内存消耗相对较大
3. **超参数敏感**：scale 参数的选择对性能影响较大，需要仔细调优
4. **训练时间**：虽然推理速度有所提升，但训练时间可能略长于标准 ResNet

5.3 适用场景

Res2Net 特别适用于以下场景：

- 需要处理多尺度目标的图像分类任务
- 对模型精度要求较高的应用
- 计算资源充足的环境
- 需要提取细粒度特征的任务

6 结论

通过本次卷积神经网络实验，我们成功实现了五种不同的网络架构在 CIFAR-10 数据集上的分类任务。主要结论如下：

1. 不同网络架构性能差异显著：SE-ResNet 表现最佳（88-93%），DenseNet 参数效率最高，基础 CNN 性能相对较低（70-75%）
2. 跳跃连接、密集连接、注意力机制等技术显著提升了网络性能和训练稳定性
3. 深层网络架构能够更好地解决梯度消失问题，实现更高的准确率

本次实验使我们掌握了 CNN 的基本原理和实现方法，深入理解了现代深度学习网络设计的核心思想。

A 附录：网络代码实现

A.1 基础 CNN 模型

Listing 1: 基础 CNN 模型结构

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)       # 第一个卷积层: 3->6通道, 5x5卷积核
        self.pool = nn.MaxPool2d(2, 2)        # 最大池化层: 2x2窗口
        self.conv2 = nn.Conv2d(6, 16, 5)      # 第二个卷积层: 6->16通道, 5x5卷积核
        self.fc1 = nn.Linear(16 * 5 * 5, 120) # 全连接层1
        self.fc2 = nn.Linear(120, 84)         # 全连接层2
        self.fc3 = nn.Linear(84, 10)          # 输出层: 10个类别

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))  # 卷积->ReLU->池化
        x = self.pool(F.relu(self.conv2(x)))  # 卷积->ReLU->池化
        x = torch.flatten(x, 1)               # 展平特征图
        x = F.relu(self.fc1(x))               # 全连接->ReLU
        x = F.relu(self.fc2(x))               # 全连接->ReLU
        x = self.fc3(x)                       # 输出层
        return x
```

A.2 ResNet 模型

Listing 2: ResNet18 模型结构

```python
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, planes, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)  # 跳跃连接
        out = F.relu(out)
        return out
```

A.3 SE-ResNet 模型

Listing 3: SE 模块结构

```python
class SEModule(nn.Module):
    def __init__(self, channel, reduction=16):
        super(SEModule, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
```

```
            nn.Linear(channel, channel // reduction, bias=False),
            nn.ReLU(inplace=True),
            nn.Linear(channel // reduction, channel, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avg_pool(x).view(b, c)
        y = self.fc(y).view(b, c, 1, 1)
        return x * y.expand_as(x)  # 通道注意力加权
```

A.4 DenseNet 模型

Listing 4: DenseNet 基本块结构

```
class DenseBlock(nn.Module):
    def __init__(self, in_channels, growth_rate):
        super(DenseBlock, self).__init__()
        self.bn1 = nn.BatchNorm2d(in_channels)
        self.conv1 = nn.Conv2d(in_channels, 4 * growth_rate,
            kernel_size=1, bias=False)
        self.bn2 = nn.BatchNorm2d(4 * growth_rate)
        self.conv2 = nn.Conv2d(4 * growth_rate, growth_rate,
            kernel_size=3, padding=1, bias=False)

    def forward(self, x):
        out = self.conv1(F.relu(self.bn1(x)))
        out = self.conv2(F.relu(self.bn2(out)))
        return torch.cat([x, out], 1)  # 密集连接：拼接输入和输出
```

A.5 Res2Net 模型

Listing 5: Res2Net 基本块结构

```
class Res2NetBlock(nn.Module):
    def __init__(self, inplanes, planes, stride=1, scale=4, stype='
        normal'):
        super(Res2NetBlock, self).__init__()
```

```
        self.scale = scale
        self.conv1 = nn.Conv2d(inplanes, planes * scale, kernel_size=1,
            bias=False)
        self.bn1 = nn.BatchNorm2d(planes * scale)

        if scale == 1:
            self.nums = 1
        else:
            self.nums = scale - 1

        convs = []
        bns = []
        for i in range(self.nums):
            convs.append(nn.Conv2d(planes, planes, kernel_size=3,
                stride=stride, padding=1, bias=False))
            bns.append(nn.BatchNorm2d(planes))
        self.convs = nn.ModuleList(convs)
        self.bns = nn.ModuleList(bns)

    def forward(self, x):
        out = self.conv1(x)
        out = self.bn1(out)
        out = F.relu(out)

        spx = torch.split(out, out.size(1) // self.scale, 1)
        for i in range(self.nums):
            if i == 0:
                sp = spx[i]
            else:
                sp = sp + spx[i]
            sp = self.convs[i](sp)
            sp = F.relu(self.bns[i](sp))
            if i == 0:
                out = sp
            else:
                out = torch.cat((out, sp), 1)

        return out
```