计算机视觉期末报告

题　　目:基于卷积神经网络的 FashionMNIST 分类

姓　　名：洪玉杰

学　　号：2511100136

日　　期：2026 年 1 月 10 日

目录

1 卷积神经网络算法简介

1.1 算法原理概述

卷积神经网络（Convolutional Neural Network, CNN）是一种受生物视觉系统启发的深度学习模型，特别擅长处理图像、音频等网格结构数据，相较于传统全连接神经网络，它凭借局部连接、权值共享和池化操作三大核心设计，大幅减少模型参数数量，既提升了计算效率，又能有效保留数据的空间信息，其典型结构包含特征提取和分类决策两大模块，输入层会接收高度、宽度、通道数的三维张量数据并进行归一化等预处理，卷积层则通过不同尺寸的卷积核在输入特征图上滑动做逐元素相乘求和运算，结合 ReLU 等激活函数提取边缘、纹理等局部特征，池化层常用最大池化或平均池化，通过下采样降低特征图维度，减少计算量的同时增强特征的鲁棒性，批归一化层可对批次数据标准化以加速模型训练收敛，之后全连接层会将高维特征图展平为一维向量，通过神经元全连接实现高级特征的组合与分类决策，最后输出层借助 softmax 等激活函数输出各类别的概率，这种层次化的特征提取方式让 CNN 能够从低级特征逐步学习到高级特征，还具备良好的平移不变性，不仅解决了传统模型手动设计特征的弊端，还显著提升了模型的泛化能力，成为计算机视觉领域的核心技术支撑。

1.2 网络结构图

实验中使用到的卷积神经网络结构如下图 1 所示：

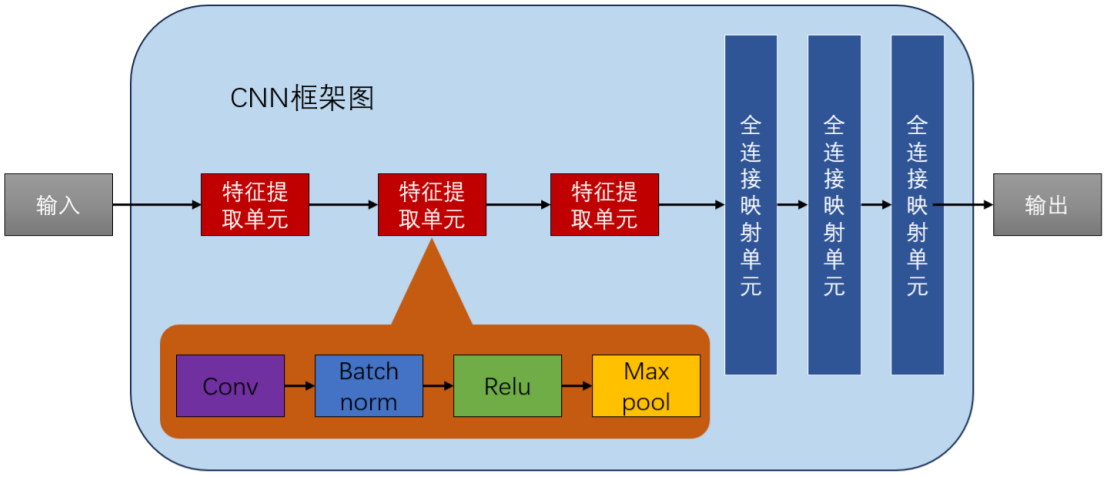

图 1: CNN 网络结构示意图

CNN 网络采用” 特征提取-分类映射” 的经典架构设计，针对 FashionMNIST

数据集的特点进行了结构优化。网络从输入层接收 28×28 的单通道灰度图像后，通过三个级联的特征提取单元逐层进行深度特征提取，每个特征提取单元均由” 卷积-归一化-激活-池化” 的经典组合构成：第一层使用 32 个 5×5 卷积核，第二层扩展至 64 个 5×5 卷积核，第三层进一步增加到 128 个 5×5 卷积核，这种通道数逐层倍增的设计使网络能够提取更丰富的图像特征；每层卷积后均引入批量归一化层，通过对每批次数据进行标准化处理，有效加速了训练收敛速度并缓解了梯度消失问题；激活函数采用 ReLU 非线性变换，为网络引入非线性特征表达能力；2×2 最大池化层则通过下采样操作降低特征图空间维度，减少计算量的同时增强特征的空间不变性。特征提取完成后，通过 Flatten 操作将三维特征图展平为一维向量，随后接入三层全连接映射单元进行分类决策：第一层全连接层设置 256 个神经元，实现高维特征的初步映射；第二层缩减至 64 个神经元，进行特征降维和抽象；第三层为 10 个神经元的输出层，对应 FashionMNIST 的 10 个服装类别。为防止过拟合，在全连接层间引入 Dropout 层，通过随机失活部分神经元降低网络复杂度，最终实现对 10FashionMNIST 图像的高效分类预测。

2 实验设置及结果分析

2.1 数据集

2.1.1 数据集简介

FashionMNIST 是一个广泛应用于深度学习图像分类任务的基准数据集，常被视为经典 MNIST 手写数字数据集的替代与扩展，其设计初衷是解决 MNIST 数据集过于简单、难以有效验证模型泛化能力的问题，由 Zalando Research 团队发布并开源供全球研究者使用。该数据集包含总计 70000 张 28×28 像素的灰度图像，其中 60000 张被划分为训练集，10000 张作为独立测试集，所有图像均对应 10 个不同类别的日常服装配饰，具体涵盖 T 恤、裤子、套头衫、连衣裙、外套、凉鞋、衬衫、运动鞋、包包以及短靴，每个类别均包含 6000 张训练样本和 1000 张测试样本，且数据集中的图像都经过了标准化预处理，像素值被归一化至 0-1 的区间，同时图像中的服装主体被居中对齐，能够降低模型训练的前期预处理成本。相较于以简单手写数字为对象的 MNIST 数据集，FashionMNIST 的图像具有更丰富的纹理、边缘和形状特征，更贴近真实世界的视觉任务场景，能够更精准地评估卷积神经网络等深度学习模型的特征提取能力与分类性能，因此被广泛用于深度学习入门教学、新算法的初步验证、不同模型结构的性能对比实验等场景，无论是初学者用来掌握图像分类模型的训练流程，还是研究者用来快速测试改进后的网络架构，FashionMNIST 都是一个便捷高效且极具参考价值的基准数据集，FashionMNIST 数据集如图 2 所示。

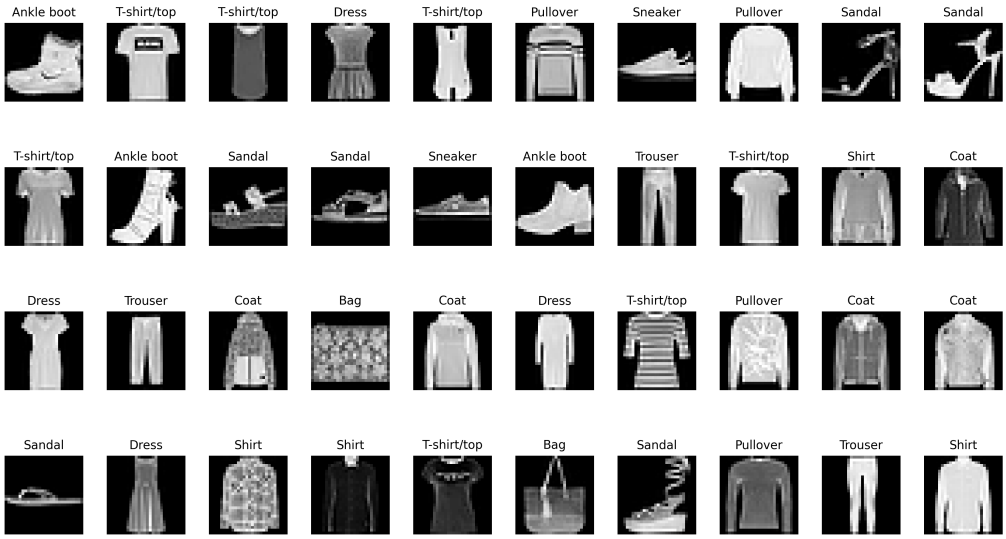

图 2: FashionMNIST 数据集样本展示

2.1.2 数据集划分

本实验采用的 FashionMNIST 数据集遵循深度学习领域的标准数据划分范式，同时结合实验需求进行了精细化调整。数据集原始划分为 60,000 张训练样本和 10,000 张测试样本，比例约为 85.7%:14.3%，确保了充足的训练数据和独立的性能评估基准。在实验实施过程中，为实现科学的模型验证与调优，进一步将原始训练集按 9:1 的比例拆分为 54,000 张实验训练集和 6,000 张验证集，形成了" 训练-验证-测试" 三层数据体系，划分情况如表 1 所示。

这种三层划分策略具有明确的功能定位：实验训练集用于网络参数的迭代更新与深度特征学习；验证集则在训练过程中实时监控模型性能，用于超参数调整如学习率、批处理大小等、模型早停与选择最优模型版本，防止过拟合；而原始测试集始终保持独立，仅用于最终评估模型在完全未见数据上的泛化能力，确保实验结果的客观性与可靠性。

从类别分布来看，数据集在各类别间保持严格的平衡状态，10 个服装类别（T 恤、裤子、套头衫、连衣裙、外套、凉鞋、衬衫、运动鞋、包包、短靴）在训练集和测试集中的样本数量比例完全一致，避免了因类别不平衡导致的模型偏倚。所有图像样本均经过标准化预处理，且服装主体已居中对齐，为卷积神经网络的高效训练奠定了坚实基础。

表 1: FashionMNIST 数据集划分

数据集类型	样本数量	用途
训练集	54,000	模型训练
验证集	6,000	超参数调优
测试集	10,000	最终性能评估

2.2 性能指标

本实验采用以下核心指标评估模型性能：

1. 准确率（Accuracy）：正确分类的样本数占总样本数的比例，反映整体分类效果：

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

其中，TP 是真阳性、TN 是真阴性、FP 是假阳性、FN 是假阴性。

2. 平均准确率（Macro-Accuracy）：各类别准确率的平均值，适用于类别均衡数据集：

$$\text{Macro-Accuracy} = \frac{1}{K} \sum_{i=1}^{K} \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i + \text{FN}_i}$$

其中，$K = 10$ 为 FashionMNIST 的类别数。

3. 交叉熵损失（Cross-Entropy Loss）：是分类任务中最常用的损失函数之一，用于衡量模型预测的概率分布与真实标签的概率分布之间的差异。对于多分类问题，交叉熵损失的计算公式为

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

其中，N 表示训练样本数量，C 表示类别总数，y_{ic} 为真实标签的独热编码，如果第 i 个样本属于第 c 类则为 1，否则为 0，$\hat{y}_{ic}$ 为模型预测第 i 个样本属于第 c 类的概率值。

4. 训练耗时：模型完成全部训练轮次的总时间。

2.3 不同网络参数/结构对性能的影响

2.3.1 实验设计

本实验采用控制变量法系统探究不同网络参数与结构对模型性能的影响，以 FashionMNIST 数据集为实验对象，统一设置训练轮次为 20 轮，通过逐一改变单一变量并保持其他参数不变的方式，全面评估各因素对分类准确率的影响规律；具体实验设计为：在标准三层 CNN 基础上，分别探究卷积层数量（2 层/3 层/4 层）对特征提取能力的影响，对比不同激活函数（ReLU/Sigmoid/Tanh）在梯度流动与非线性表达上的差异，验证优化器（SGD/Adam/RMSprop）对模型收敛速度与最终精度的作用，分析学习率（0.001/0.01/0.1）对训练稳定性与收敛效果的影响，并进一步比较标准 CNN 与经典深度网络结构（ResNet18/DenseNet121）在特征复用与梯度传播方面的性能表现。

2.3.2 结果展示与分析

本实验通过控制变量法系统探究了网络参数与结构对 FashionMNIST 分类性能的影响，结果如表 2 所示。在卷积层数方面，模型呈现 2 层 (89.1%)→3 层 (89.6%)→4 层 (90.2%) 的准确率递增趋势，验证了增加卷积层数可有效提升特征提取能力，但同时也需注意过度增加层数可能导致的过拟合风险与计算成本激增之间的平衡问题。关于激活函数的选择，ReLU(89.6%) 显著优于 Tanh(89.4%) 和 Sigmoid(51.2%)，充分证明其在缓解梯度消失问题、加速训练收敛方面的独特优势，尤其适用于深度卷积网络架构。在优化器性能对比中，Adam(90.4%) 和 RMSprop(89.9%) 等自适应优化器在相同配置下表现优于 SGD(89.6%)，体现了自适应学习率机制在提升训练稳定性与最终分类准确率上的显著优势。学习率作为关键超参数，设置为 0.01 时模型表现最优，过高 (0.1) 会导致训练过程不稳定，过

低 (0.001) 则易造成收敛不足，凸显了选择合适学习率对模型整体性能的重要影响。在网络结构性能对比中，DenseNet121(89.4%) 展现出比 ResNet18(88.1%) 更好的适配性，其密集连接结构在小尺寸图像上的特征复用优势得到验证，但两者与精心调优的标准 CNN 差异并不明显，说明在简单任务中复杂网络结构的优势并不显著；考虑到 FashionMNIST 数据集图像尺寸较小是 28×28 灰度图且分类任务相对简单，三层 CNN 已能有效提取关键特征，而深层网络由于参数过多，反而容易在小数据集上发生过拟合现象，导致模型泛化能力下降。

表 2: 不同参数/结构的性能对比

模型名称	卷积层数	激活函数	优化器	学习率	测试准确率
基准模型	3	ReLU	SGD	0.01	89.6%
2 层卷积模型	2	ReLU	SGD	0.01	89.1%
4 层卷积模型	4	ReLU	SGD	0.01	90.2%
Sigmoid 激活模型	3	Sigmoid	SGD	0.01	51.2%
Tanh 激活模型	3	Tanh	SGD	0.01	89.4%
Adam 优化器模型	3	ReLU	Adam	0.001	90.4%
RMSprop 优化器模型	3	ReLU	RMSprop	0.001	89.9%
高学习率 (0.1) 模型	3	ReLU	SGD	0.1	89.7%
低学习率 (0.001) 模型	3	ReLU	SGD	0.001	83.6%
ResNet18 模型	-	ReLU	SGD	0.001	88.1%
DenseNet121 模型	-	ReLU	SGD	0.001	89.4%

2.4 计算复杂度分析

表 3: 计算复杂度指标对比

模型类型	参数量	模型大小	训练时长	单个样本平均测试时长
CNN（3 层卷积）	0.57M	2.18MB	4.01min	0.02ms
ResNet18	11.18M	42.71MB	6.97min	0.06ms
DenseNet121	6.96M	27.11MB	42.64min	0.14ms

根据实验结果，DenseNet121 虽然在参数量（6.96M）和模型大小（27.11MB）上均低于 ResNet18（11.18M 参数量、42.71MB 模型大小），但其训练时长（42.64min）和单样本测试时长（0.14ms）却分别是 ResNet18（6.97min 训练时长、0.06ms 测试时长）的约 6 倍和 2.3 倍，这一现象主要由两者的网络结构差异导致。DenseNet

采用密集连接设计，每一层都会接收前面所有层的特征图作为输入并进行拼接，这种设计虽然提高了参数复用率，减少了总参数量，但随着网络深度增加，每一层的输入通道数会快速累积增长，导致卷积操作的计算量与输入通道数呈平方级关系膨胀；相比之下，ResNet 使用残差连接，通过简单的元素级加和组合特征，计算开销显著更低。此外，DenseNet 中特征图数量随深度线性增长的特点不仅增加了内存占用，还大幅提高了后续卷积层的计算复杂度和内存访问成本，同时密集连接带来的大量特征图拼接操作以及每层 Batch Normalization 的累积计算开销，共同导致了 DenseNet 在参数量和模型大小更优的情况下，运行时间却大幅增加的现象，这也反映了深度学习模型中参数效率与计算效率之间的权衡关系。

2.5 训练过程的 Loss 曲线变化

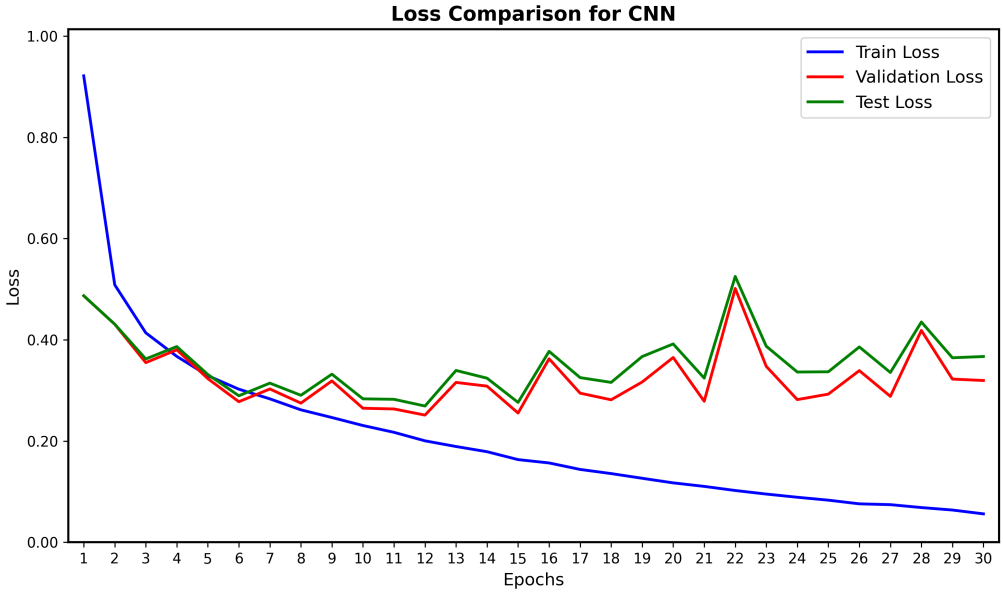

图 3: 训练过程 Loss 曲线变化

图 3 是 CNN 模型在 FashionMNIST 数据集上的训练、验证和测试的 Loss 曲线。可以看出，训练 Loss 呈现出显著的下降趋势，从初始的接近 1.0 迅速下降并稳定在 0.1 以下，这种快速收敛的趋势表明模型具有良好的学习能力，能够有效地从训练数据中捕捉特征模式，优化过程高效。验证 Loss 和测试 Loss 高度重合，始终保持在 0.3-0.4 之间小幅波动，没有出现明显的上升趋势，这表明模型具有不错的泛化能力，没有发生过拟合现象，能够稳定地对未见过的数据进行预测。训练 Loss 明显低于验证和测试 Loss 是因为模型是直接在训练数据上进行优化的，但三者的整体趋势保持一致，说明模型的训练过程稳定可靠。曲线后期的微小波动属于正常现象，与数据的随机性和模型的学习动态有关。

3 总结与展望

3.1 工作总结

本实验系统完成了基于卷积神经网络的 FashionMNIST 图像分类任务，通过构建并优化标准 CNN 模型，同时深入探究了卷积层数、激活函数、优化器等关键参数对模型性能的影响规律，验证了 ReLU 激活函数、3 层卷积结构以及 SGD 优化器（学习率 0.01）在该任务上的有效性。此外，实验进一步对比了 ResNet18 和 DenseNet121 等经典深度网络结构的表现，完成了参数量、模型大小、训练时长及单样本测试时长等计算复杂度指标的全面分析，并通过 Loss 曲线验证了模型训练过程的稳定性和良好泛化能力，为卷积神经网络在图像分类任务中的应用提供了完整的实验参考和性能评估。

3.2 存在问题

尽管实验取得了较为理想的分类效果，但仍存在一些需要改进的地方：标准 CNN 模型在特征提取深度和语义理解能力上仍有局限，导致分类准确率未达到更高水平；实验过程中未对数据集进行数据增强处理如旋转、裁剪、水平翻转等，可能影响模型对图像形变和视角变化的鲁棒性；超参数调优采用了简单的控制变量法，未运用网格搜索或贝叶斯优化等更系统的方法，可能错失更优的参数组合；深度网络结构如 ResNet18 和 DenseNet121，在小尺寸图像分类任务中未充分发挥其优势，参数量和计算复杂度的增加与性能提升不成比例。

3.3 未来改进方向

针对上述问题，未来可以从以下几个方面进行改进：引入数据增强技术如随机裁剪、水平翻转、高斯噪声添加等，通过增加训练数据多样性提升模型的泛化能力和鲁棒性；采用迁移学习策略，基于在大型数据集上预训练的深度网络模型如 ResNet50、EfficientNet 等进行微调，充分利用预训练模型学习到的通用特征，进一步提升分类准确率；尝试结合注意力机制，让模型能够自动聚焦于图像的关键区域，增强特征表示能力；探索模型压缩技术例如权重量化、通道剪枝等，在保证模型性能的前提下降低计算复杂度和内存占用，提升推理速度；运用更先进的超参数优化方法，系统性地寻找最优参数组合，进一步挖掘模型性能潜力。

4 附录：实验代码

Listing 1: PyTorch 实现 FashionMNIST 分类完整代码

```
# 利用3层卷积神经网络来实现对FashionMNIST 图像分类任务的代码

import torch
import torchvision
from torch import nn
from torch.utils.data import DataLoader
from torchvision.transforms import transforms
from rich.console import Console
from icecream import ic
import matplotlib.pyplot as plt
import os

# 定义训练的设备
device = torch.device("cuda:0" if torch.cuda.is_available() else
    "cpu")

# 创建控制台对象，用于终端显示
console = Console()

# 下载数据集
current_dir = os.path.dirname(os.path.abspath(__file__))
dataset_root = os.path.join(current_dir, "dataset")

# 加载完整训练集和测试集
train_full_dataset = torchvision.datasets.FashionMNIST(root=
    dataset_root, train=True, transform=transforms.ToTensor(),
    download=False)
test_dataset = torchvision.datasets.FashionMNIST(root=
    dataset_root, train=False, transform=transforms.ToTensor(),
    download=False)

# 划分训练集和验证集：54,000用于训练，6,000用于验证
train_size = 54000
val_size = 6000
train_dataset, val_dataset = torch.utils.data.random_split(
```

```
    train_full_dataset, [train_size, val_size], generator=torch.
    Generator().manual_seed(42))

ic("数据集为FashionMNIST")
console.print(f"[bold yellow]训练数据集长度为: {len(
    train_dataset)}[/bold yellow]")
console.print(f"[bold yellow]验证数据集长度为: {len(val_dataset)
    }[/bold yellow]")
console.print(f"[bold yellow]测试数据集长度为: {len(test_dataset
    )}[/bold yellow]")

# 查看数据集图片大小
# 获取第一个训练样本
first_img, first_label = train_dataset[0]
console.print(f"[bold red]图片张量形状: {first_img.shape}[/bold
    red]")

# 加载数据集
train_data_loader = DataLoader(dataset=train_dataset, batch_size
    =64)
val_data_loader = DataLoader(dataset=val_dataset, batch_size=64)
test_data_loader = DataLoader(dataset=test_dataset, batch_size
    =64)

# 搭建神经网络
class network(nn.Module):
    def __init__(self):
        super(network, self).__init__()
        self.model = nn.Sequential(
            # 第一层卷积
            nn.Conv2d(in_channels=1, out_channels=32,
                kernel_size=5, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),

            # 第二层卷积
            nn.Conv2d(in_channels=32, out_channels=64,
```

```
                kernel_size=5, padding=2),
60              nn.BatchNorm2d(64),
61              nn.ReLU(),
62              nn.MaxPool2d(kernel_size=2),
63
64              # 第三层卷积
65              nn.Conv2d(in_channels=64, out_channels=128,
                    kernel_size=5, padding=2),
66              nn.BatchNorm2d(128),
67              nn.ReLU(),
68              nn.MaxPool2d(kernel_size=2),
69
70              nn.Flatten(),
71              # 全连接层: Dropout层防止过拟合
72              nn.Linear(in_features=1152, out_features=256),
73              nn.ReLU(),
74              nn.Dropout(p=0.5),
75              nn.Linear(in_features=256, out_features=64),
76              nn.ReLU(),
77              nn.Dropout(p=0.5),
78              nn.Linear(in_features=64, out_features=10)
79          )
80
81      def forward(self, x):
82          return self.model(x)
83
84  # 创建网络模型
85  my_net = network()
86  my_net = my_net.to(device)# 转移到我们设置的网络上去训练
87
88  # 损失函数
89  loss_fn = nn.CrossEntropyLoss()
90  loss_fn = loss_fn.to(device)
91
92  # 优化器
93  learning_rate = 1e-2  # 0.01
94  optimizer = torch.optim.SGD(my_net.parameters(), lr=
        learning_rate)
```

```
95
96  # 设置训练网络的一些参数
97  # 训练次数
98  epoch = 30
99  # 记录训练的次数
100 total_train_step = 0
101 # 记录测试的次数
102 total_test_step = 0
103 # 记录训练loss
104 train_loss_list = []
105 # 记录验证loss
106 val_loss_list = []
107 # 记录测试loss
108 test_loss_list = []
109 # 记录测试准确率
110 test_accuracy_list = []
111
112 # 训练开始
113 for i in range(epoch):
114     my_net.train()
115     total_train_loss = 0
116     total_train_accuracy = 0
117
118     for data in train_data_loader:
119         imgs, targets = data
120         imgs, targets = imgs.to(device), targets.to(device)
121         outputs = my_net(imgs)
122         loss = loss_fn(outputs, targets)
123
124         # 优化器优化模型
125         optimizer.zero_grad()
126         loss.backward()
127         optimizer.step()
128
129         total_train_step += 1
130         total_train_loss += loss.item()
131         accuracy = (outputs.argmax(1) == targets).sum()
132         total_train_accuracy += accuracy.item()
```

```
133
134     # 计算每轮训练的平均loss和acc
135     train_loss = total_train_loss / len(train_data_loader)
136     train_accuracy = total_train_accuracy / len(train_dataset)
137     # 记录训练loss
138     train_loss_list.append(train_loss)
139
140     # 验证步骤开始
141     my_net.eval()
142     total_val_loss = 0
143     total_val_accuracy = 0
144
145     with torch.no_grad():  # 去除梯度即不进行权重更新
146         for data in val_data_loader:
147             imgs, targets = data
148             imgs, targets = imgs.to(device), targets.to(device)
149             outputs = my_net(imgs)
150             loss = loss_fn(outputs, targets)
151             total_val_loss += loss.item()
152             accuracy = (outputs.argmax(1) == targets).sum()
153             total_val_accuracy += accuracy.item()
154
155     val_accuracy = total_val_accuracy / len(val_dataset)
156     val_loss = total_val_loss / len(val_data_loader)
157     # 记录验证loss
158     val_loss_list.append(val_loss)
159
160     console.print(f"[bold magenta]第{i+1}轮验证数据集loss:{
        val_loss}, 准确率:{val_accuracy*100:.2f}%[/bold magenta]"
        )
161     total_test_step += 1
162
163     # 测试步骤开始
164     my_net.eval()  # 确保模型处于评估模式
165     total_test_loss = 0
166     total_test_accuracy = 0
167
168     with torch.no_grad():
```

```
169             for data in test_data_loader:
170                 imgs, targets = data
171                 imgs, targets = imgs.to(device), targets.to(device)
172                 outputs = my_net(imgs)
173                 loss = loss_fn(outputs, targets)
174                 total_test_loss += loss.item()
175                 accuracy = (outputs.argmax(1) == targets).sum()
176                 total_test_accuracy += accuracy.item()
177 
178         test_accuracy = total_test_accuracy / len(test_dataset)
179         test_loss = total_test_loss / len(test_data_loader)
180         # 记录每轮测试loss和准确率
181         test_loss_list.append(test_loss)
182         test_accuracy_list.append(test_accuracy)
183 
184         console.print(f"[bold green]第{i+1}轮测试数据集loss:{
                test_loss}, 准确率:{test_accuracy*100:.2f}%[/bold green]"
                )
185 
186 # 所有训练轮次结束后, 输出最终结果
187 console.print("\n[bold blue]所有训练轮次完成! [/bold blue]")
188 if len(test_loss_list) > 0:
189         avg_test_loss = sum(test_loss_list) / len(test_loss_list)
190         avg_test_accuracy = sum(test_accuracy_list) / len(
                test_accuracy_list) * 100
191         console.print(f"[bold green]测试集平均Loss:{avg_test_loss:.4
                f}, 平均准确率:{avg_test_accuracy:.2f}%[/bold green]")
192 else:
193         console.print("[bold green]训练完成! [/bold green]")
194 
195 # 绘制训练、验证、测试的loss曲线
196 plt.figure(figsize=(10, 6))
197 plt.title('Loss Comparison for CNN', fontweight='bold', fontsize
        =14)
198 
199 # 设置坐标轴范围
200 plt.ylim(bottom=0.0, top=max(max(train_loss_list), max(
        val_loss_list), max(test_loss_list))*1.1)
```

```
plt.xlim(left=0.5, right=epoch+0.5)

# 设置坐标轴标签
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Loss', fontsize=12)

# 设置x轴刻度为整数并从1开始
plt.xticks(range(1, epoch+1))

# 设置y轴刻度格式为最多两位小数
plt.ticklabel_format(style='plain', axis='y', useOffset=False)
plt.gca().yaxis.set_major_formatter(plt.FormatStrFormatter('%.2f
    '))

# 绘制三条曲线
plt.plot(range(1, epoch+1), train_loss_list, '-', color='blue',
    label='Train Loss', linewidth=2)
plt.plot(range(1, epoch+1), val_loss_list, '-', color='red',
    label='Validation Loss', linewidth=2)
plt.plot(range(1, epoch+1), test_loss_list, '-', color='green',
    label='Test Loss', linewidth=2)

# 添加图例和调整布局
plt.legend(fontsize=12, loc='upper right')
plt.tight_layout()

# 添加图表边框
ax = plt.gca()
ax.spines['top'].set_linewidth(1.5)
ax.spines['right'].set_linewidth(1.5)
ax.spines['bottom'].set_linewidth(1.5)
ax.spines['left'].set_linewidth(1.5)

# 保存图像到当前目录下的imgs文件夹
imgs_dir = './imgs'
if not os.path.exists(imgs_dir):
    os.makedirs(imgs_dir)

```

```
235 # 使用相对路径保存图像
236 plt.savefig(os.path.join(imgs_dir, 'all_loss_curves.png'), dpi
    =300, bbox_inches='tight')
237 console.print(f"[bold blue]图像已保存到: {os.path.abspath(os.
    path.join(imgs_dir, 'all_loss_curves.png'))}[/bold blue]")
238
239 # 显示图像
240 plt.show()
241 plt.close()
```