《机器学习导论》
课程设计报告

题目： 基于卷积神经网络的衣服图像分类模型

院　　系	计算机科学与技术学院
专业班级	
姓　　名	
学　　号	
指导教师	李钦宾

计算机科学与技术学院

目　　录

1 问题定义与理解

1.1 问题描述

本次大作业的任务是基于 Fashion-MNIST 数据集构建一个图像分类模型，实现对 10 类服饰（如 T 恤、外套、鞋子等）的自动识别。Fashion-MNIST 是一个替代 MNIST 手写数字集的图像数据集。 它是由 Zalando（一家德国的时尚科技公司）旗下的研究部门提供。其涵盖了来自 10 种类别的共 7 万个不同商品的正面图片。Fashion-MNIST 的大小、格式和训练集/测试集划分与原始的 MNIST 完全一致。60000/10000 的训练测试数据划分，28x28 的灰度图片。

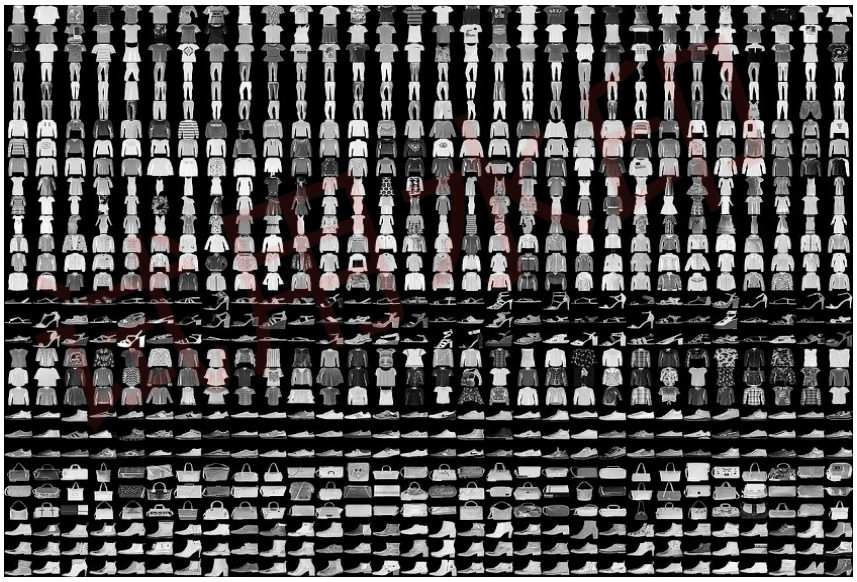

图 1-1 数据集示意图

本任务属于典型的监督学习中的多分类问题，输入为图像像素构成的二维张量，输出为 10 类之一的标签。模型需学习从图像中提取有效特征并进行准确分类。根据题目建议和项目实现，我采用了基于预训练 ResNet-18 的卷积神经网络（CNN）作为主干模型，由于通常用于处理三通道 RGB 图像，因此需对图像进行预处理，包括尺寸调整和通道扩展。为提升模型性能，本项目采用基于 ResNet-18 的迁移学习方法：保留其强大的特征提取能力，替换分类头以适应 Fashion-MNIST 的输出需求。

本项目的核心目标包括：完成数据分析与处理、构建并训练模型、评估分类效果，并可视化训练过程与测试结果（如混淆矩阵和分类报告）以用来分析。本

项目旨在增强我们对机器学习核心概念和算法的理解与应用能力，帮助我们掌握图像分类的基本流程，理解模型适配、训练策略与性能分析等关键步骤，同时为后续更复杂的机器学习任务打下基础。

1.2 相关理论基础

卷积神经网络(Convolutional Neural Networks, CNN)是一种深度学习算法，主要用于图像识别和处理领域。CNN 的设计模仿了人类视觉系统的工作机制，能够通过层次化的方式对图像中的局部特征和全局模式进行提取与分析，其应用已扩展到视频分析、自然语言处理等多个领域。

CNN 的核心思想在于通过局部感知野(Local Receptive Fields)和权值共享(Shared Weights)实现特征提取和数据的高效表示。CNN 因其强大的特征提取能力，被广泛应用于图像分类、目标检测、人脸识别和视频分析等任务中。例如，在 ImageNet 大规模图像识别挑战赛中，基于 CNN 的模型展现了超越传统方法的性能。此外，CNN 结构在研究中不断进化，衍生出了深度残差网络(ResNet)、密集连接网络(DenseNet)等变种，这些改进进一步增强了 CNN 对复杂数据的处理能力。本项目采用的正是深度残差网络(ResNet)。

残差神经网络(ResNet)是由微软研究院的何恺明、张祥雨、任少卿、孙剑等人提出的。残差神经网络的主要贡献是发现了“退化现象（Degradation）”，并针对退化现象发明了 “快捷连接（Shortcut connection）”，极大的消除了深度过大的神经网络训练困难问题。

ResNet-18 是 ResNet 系列中较轻量的一个版本，包含 18 个权重层（17 个卷积层和 1 个全连接层）。 相比更深的 ResNet（如 ResNet-50、ResNet-101），ResNet-18 参数量较少，适合计算资源有限的场景。在本项目中，使用的是预训练的 ResNet-18 模型作为特征提取器，并根据 Fashion-MNIST 的分类任务需求，替换了原始模型的输出层以适配 10 类服饰分类问题。这种方式属于典型的迁移学习方法，可以在较少训练数据和较短训练时间的前提下，获得较好的分类性能。

通过结合 CNN 的特征提取能力与 ResNet 的深层结构优势，实验模型能够有效处理 Fashion-MNIST 数据集中的图像特征，从而实现对服饰类别的准确分类。

2 数据分析及处理

2.1 数据集分析

本实验使用的 Fashion-MNIST 数据集由 Zalando 提供，包含 10 类服饰图像，每类图像均为 28×28 像素的灰度图像。训练集包含 60,000 张图像，测试集包含 10,000 张图像，图像标签的类别包括：T 恤/上衣、裤子、套头衫、连衣裙、外套、凉鞋、衬衫、运动鞋、包和短靴。各类别图像数量基本平衡，适合用于监督学习中的多分类任务。

表 2-1 标签与描述对照表

Label	Description
0	T 恤（T-shirt/top）
1	裤子（Trouser）
2	套头衫（Pullover）
3	连衣裙（Dress）
4	外套（Coat）
5	凉鞋（Sandal）
6	衬衫（Shirt）
7	运动鞋（Sneaker）
8	包（Bag）
9	靴子（Ankle boot）

2.2 数据处理

由于实验中采用的是基于预训练 ResNet-18 的迁移学习方法，而该网络原始设计用于处理 224×224 尺寸的 RGB 图像，因此需要对输入图像进行了如下预处理：

尺寸调整：使用 transforms.Resize((224, 224)) 将 28×28 的图像上采样至 224×224。虽然可能导致图像模糊，但有助于适配模型结构。

通道转换：使用 transforms.Grayscale(num_output_channels=3) 将单通道灰度图复制为 3 通道图像，以满足 ResNet 对输入维度的要求。

张量转换：使用 transforms.ToTensor() 将图像转为 PyTorch 张量，并自动将像素值归一化至 [0, 1] 区间。

未使用标准化（如 transforms.Normalize）或数据增强（如旋转、裁剪），原因在于图像本身较小，过多增强操作可能丢失关键图像结构，经过尝试发现，复杂的数据处理并不会提高最终效果，反而会有负面影响，也提升了训练时间和设备负担。数据处理的具体代码如下：

```
# 数据预处理：将图片调整为224x224、转为3通道，并转为Tensor格式
transform = transforms.Compose([
    transforms.Resize((224, 224)),                # 将28x28图像调整为224x224（适配ResNet输入）
    transforms.Grayscale(num_output_channels=3),  # 将单通道灰度图转为3通道（适配ResNet预训练模型）
    transforms.ToTensor()                         # 转换为张量
    # 可选添加：transforms.Normalize((0.5,), (0.5,))
])
```

2.3 数据加载

训练集与测试集通过 torchvision.datasets.FashionMNIST 下载和加载，并由 DataLoader 进行批处理。训练集设置 shuffle=True 打乱样本顺序，以提升模型泛化能力，测试集设置 shuffle=False 保证评估结果稳定。批大小设为 64，兼顾训练效率与显存开销。

```
# 加载FashionMNIST训练集和测试集，自动下载数据到本地./data文件夹
train_set = torchvision.datasets.FashionMNIST(
    root='./data',
```

```
    train=True,
    download=True,
    transform=transform
)
test_set = torchvision.datasets.FashionMNIST(
    root='./data',
    train=False,
    download=True,
    transform=transform
)
# 使用DataLoader包装数据集，支持批量加载与打乱顺序
batch_size = 64  # 批大小，影响训练速度与显存占用
train_loader = DataLoader(train_set, batch_size=batch_size, sh
uffle=True)
test_loader = DataLoader(test_set, batch_size=batch_size, shuf
fle=False)
```

3 模型构建与训练

3.1 模型定义

本实验采用基于 ResNet-18 的迁移学习方案，使用 torchvision 提供的预训练 ResNet-18 模型作为特征提取器，ResNet-18 具有 18 层深度，包含 8 个残差块，在 ImageNet 上预训练的参数能有效捕捉通用图像特征，相比更深的 ResNet 变体，ResNet-18 在计算效率和性能之间取得较好平衡。

ResNet-18 主要由以下部分组成：

输入层：一个 7×7 的大卷积核卷积层（stride=2），后接批量归一化（BatchNorm）和 ReLU 激活函数，用于快速降采样输入图像。

最大池化层：紧随卷积层之后，3×3 最大池化，步幅为 2，进一步减少特征图尺寸，降低计算量。

残差块（Residual Blocks）：共 4 个阶段，每组由 2 个残差单元组成，每组残差块的第一个单元可能会通过步幅为 2 的卷积进行下采样（降低分辨率，同时增加通道数），每个残差单元包括两个 3×3 卷积层和跳跃连接（skip connection）。跳跃连接将输入直接添加到输出，实现“恒等映射”，帮助梯度直接传递，避免梯度消失。

全局平均池化：对最后一层的特征图进行全局平均池化，将空间维度降为 1×1，减少参数量。

全连接层（FC）：将全局平均池化的结果展平，并通过一个全连接层输出分类结果，默认输出 1000 类（ImageNet），本实验中替换为输出 10 类。

根据网络结构，我们对输入和输出进行以下适配和改造，虽然原始 ResNet 的输入层已支持 3 通道，显式重定义可确保与后续修改的一致性，保持 kernel_size=7 和 stride=2 的初始下采样策略，快速压缩特征图尺寸，将原 1000 类分类头替换为 10 类输出，适配 Fashion-MNIST 任务，保留原始特征维度（512 维）作为全连接层输入。

```
# 修改 ResNet 的第一个卷积层以接受 3 通道输入（默认是 3 通道，但防止灰度图输入报错）
        self.base.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
```

```
        # 获取原 fc 层输入特征维度
        num_features = self.base.fc.in_features
        # 替换 fc 层，使其输出 10 个类别的预测值
        self.base.fc = nn.Linear(num_features, 10)
```

3.2 训练过程设计

本实验使用微调后的 ResNet-18 模型对 Fashion-MNIST 数据集进行训练，训练周期设为 20 轮。优化器选用 Adam，损失函数为交叉熵损失函数 CrossEntropyLoss，这在多分类任务中应用广泛。

```
del = FashionResNet().to(device)
criterion = nn.CrossEntropyLoss()  # 多分类交叉熵损失
optimizer = optim.Adam(model.parameters(), lr=0.001)  # 使用 Adam 优化器
```

每轮训练开始前，首先调用 model.train() 方法，将模型切换至训练模式。在 PyTorch 中，该设置确保如 Batch Normalization（BN）层 和 Dropout 层 等结构能够正常工作。具体来说，BN 会根据当前 batch 的统计量进行特征归一化，Dropout 则会随机丢弃部分神经元，以提高模型的泛化能力。这一步是深度学习训练流程的标准启动操作。

```
for epoch in range(epochs):
        model.train()  # 设置为训练模式
        running_loss = 0.0
        correct = 0
        total = 0
```

在训练过程中，模型使用 DataLoader 按照 batch size = 64 的配置加载训练数据，避免一次性将所有样本送入内存，提升计算效率。每一个 batch 内，完成标准的四个优化步骤：梯度清零、前向传播、损失计算、反向传播与参数更新。整个训练过程由 Adam 优化器驱动，结合交叉熵损失函数作为目标函数。

```
for images, labels in train_loader:
            images, labels = images.to(device), labels.to(devi
ce)
            optimizer.zero_grad()           # 梯度清零
            outputs = model(images)         # 前向传播
            loss = criterion(outputs, labels)  # 计算损失
            loss.backward()                 # 反向传播
```

```
            optimizer.step()                    # 更新参数
```

通过 loss.item() 获取每个 batch 的损失值，并累计用于后续每个 epoch 的平均损失计算。同时使用 torch.max(outputs.data, 1) 获取模型的分类预测结果，并与真实标签进行比对，用于计算训练准确率。

```
running_loss += loss.item()     # 累计损失
            _, predicted = torch.max(outputs.data, 1)  # 获取预测标签
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
```

每轮训练结束后，模型立即切换至评估模式 model.eval()，此时 BN 层和 Dropout 层的行为将被冻结，确保测试结果不受训练策略干扰。为了进一步提升推理速度和节省显存，在验证阶段使用 torch.no_grad() 上下文管理器关闭梯度计算功能，仅进行前向传播，获取每张验证图像的分类结果。

```
# 验证阶段
        model.eval()
        val_correct = 0
        val_total = 0
        with torch.no_grad():  # 禁用梯度计算，加快推理
            for images, labels in test_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                _, predicted = torch.max(outputs.data, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()
```

为全面监控训练过程，代码实时记录和输出三类核心训练指标，用于分析模型的学习进展和泛化能力。

训练损失通过统计当前 epoch 所有 batch 的损失均值计算得到。采用 CrossEntropyLoss 交叉熵损失函数，这是分类任务中最常见的目标函数，能够有效度量模型预测分布与真实标签之间的差异。在模型初始阶段，训练损失通常在 2.3 左右，随着学习逐步降低至 0.2 以下，表明模型在训练集上逐渐收敛。

为了同时评估模型在训练集与测试集上的表现，代码在每个 epoch 结束时分别计算训练准确率与验证准确率。前者用于衡量模型的记忆能力，后者则反映其泛化能力。若两者差距过大，往往提示模型出现过拟合风险，需考虑使用数据增强、正则化等方法进行改进。

每个训练周期结束后，通过标准格式的日志语句输出当前轮次的完整训练状

态，包括训练损失、训练准确率与验证准确率，便于开发者实时观测训练曲线变化趋势，为后续的调参与模型选择提供依据。

```
 # 记录每轮的指标
        train_loss.append(running_loss / len(train_loader))
        train_acc.append(100 * correct / total)
        val_acc.append(100 * val_correct / val_total)
        print(f'Epoch [{epoch+1}/{epochs}] | Loss: {train_loss
[-1]:.4f} | Train Acc: {train_acc[-1]:.2f}% | Val Acc: {val_ac
c[-1]:.2f}%')
    return train_loss, train_acc, val_acc
# 训练模型
train_loss, train_acc, val_acc = train_model(model, epochs=20)
```

4 实验结果与分析

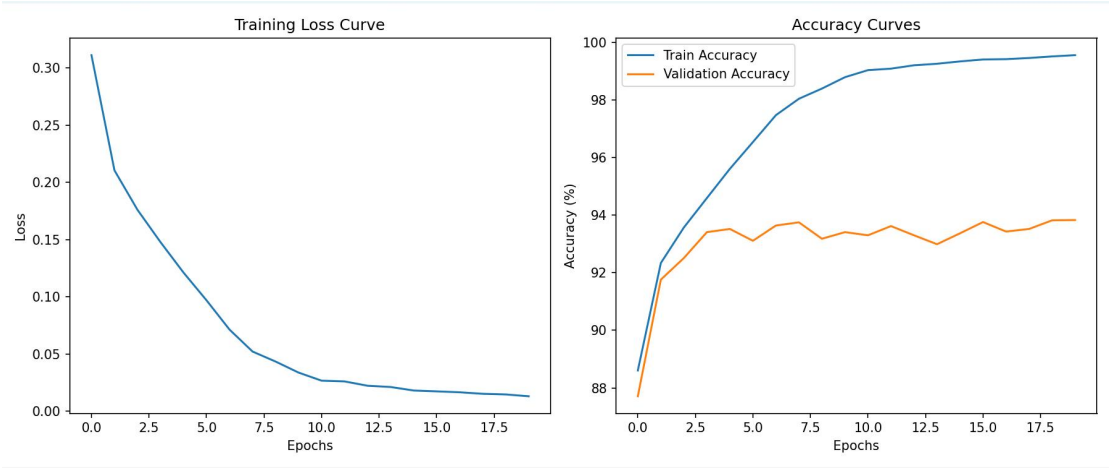

图 4-1 演化图

4.1 训练过程分析

(1) 损失函数演化：

训练损失在整个训练周期中呈现持续下降的趋势。从初始的 0.3178 降至第 10 轮的 0.0337，最终在第 20 轮收敛至 0.0157。损失下降速度在前 5 轮尤为迅速，之后逐步趋于平稳，表明模型在初期快速学习到大部分判别性特征，随后进入细节优化阶段。

此外，损失曲线无明显震荡或反弹，说明模型训练过程稳定，未出现过拟合或欠拟合的波动。

(2) 准确率演化：

训练准确率从第 1 轮的 88.39% 稳步提升至第 20 轮的 99.48%，展现出极强的拟合能力。验证集准确率从 89.45% 提高到 94.30%，变化趋势与训练准确率大致同步，但提升幅度略小，稳定在 93%以上，说明模型泛化能力较强。

值得注意的是，从第 6 轮开始，训练集准确率已超过 96%，但验证集准确率提升趋缓，表明模型在后期进一步训练主要针对训练集微调，而验证集的提升空间有限。

(3) 精确度（Precision）：

虽然训练日志未显式记录每轮的平均精确度变化，但从最终测试结果可知，

模型各类别的 Precision 值均在 0.85 以上，多数类别如 Trouser、Sandal、Bag 等已接近 0.99，说明模型在输出正类样本时具有极高的置信度。

可以推测，精确度随训练轮数增长而不断提升，尤其是在早期损失大幅下降阶段，模型已快速学习到能够准确区分正类的关键特征。

(4) 召回率（Recall）：

与精确度类似，召回率也未在每轮中单独显示。但从测试结果看，除 Shirt 类稍低（0.80）外，其他各类别的 Recall 均维持在 0.93–0.99 区间，说明模型对多数目标类别具有较强的覆盖能力。

结合验证集准确率的变化，推断模型召回率在训练后期持续提升，且未出现明显波动，表明其在不同类别间的均衡性表现良好。

```
(fashion) C:\Users\muke\Documents\CursorCode\Fashion>python main.py
Using device: cuda
Epoch [1/20] | Loss: 0.3178 | Train Acc: 88.39% | Val Acc: 89.45%
Epoch [2/20] | Loss: 0.2118 | Train Acc: 92.30% | Val Acc: 92.11%
Epoch [3/20] | Loss: 0.1782 | Train Acc: 93.54% | Val Acc: 93.31%
Epoch [4/20] | Loss: 0.1482 | Train Acc: 94.60% | Val Acc: 92.84%
Epoch [5/20] | Loss: 0.1218 | Train Acc: 95.63% | Val Acc: 93.69%
Epoch [6/20] | Loss: 0.0954 | Train Acc: 96.62% | Val Acc: 93.53%
Epoch [7/20] | Loss: 0.0754 | Train Acc: 97.27% | Val Acc: 93.77%
Epoch [8/20] | Loss: 0.0567 | Train Acc: 97.96% | Val Acc: 93.50%
Epoch [9/20] | Loss: 0.0442 | Train Acc: 98.35% | Val Acc: 93.44%
Epoch [10/20] | Loss: 0.0337 | Train Acc: 98.78% | Val Acc: 93.19%
Epoch [11/20] | Loss: 0.0308 | Train Acc: 98.88% | Val Acc: 93.76%
Epoch [12/20] | Loss: 0.0243 | Train Acc: 99.11% | Val Acc: 94.13%
Epoch [13/20] | Loss: 0.0238 | Train Acc: 99.15% | Val Acc: 93.55%
Epoch [14/20] | Loss: 0.0209 | Train Acc: 99.31% | Val Acc: 93.98%
Epoch [15/20] | Loss: 0.0179 | Train Acc: 99.37% | Val Acc: 93.78%
Epoch [16/20] | Loss: 0.0208 | Train Acc: 99.27% | Val Acc: 93.71%
Epoch [17/20] | Loss: 0.0137 | Train Acc: 99.50% | Val Acc: 93.93%
Epoch [18/20] | Loss: 0.0156 | Train Acc: 99.45% | Val Acc: 93.79%
Epoch [19/20] | Loss: 0.0123 | Train Acc: 99.56% | Val Acc: 94.30%
Epoch [20/20] | Loss: 0.0157 | Train Acc: 99.48% | Val Acc: 93.84%
```

图 4-2 训练过程图

4.2 混淆矩阵分析

混淆矩阵显示了模型在每个类别上的预测情况(真实标签 vs. 预测标签)。

从图中可见，大多数类别如 Trouser、Sandal、Bag、Sneaker 等均能被模型高精度识别，对角线值接近 1000，误分类数量极少。其中：

Trouser 类预测最为准确，几乎无误（TP=991，误判仅 9 例），说明模型对裤子类图像具有高度判别力。

Shirt 类的混淆较为严重，尤其易被错分为 T-shirt/top、Pullover 和 Coat，可能由于这些类别在图像纹理与轮廓上较为接近。

T-shirt/top 存在部分被误识别为 Shirt 和 Pullover 的情况，这种混淆也符合

人类主观经验。

Pullover 与 Coat 之间有部分交叉误判，反映出模型对这类结构相似服饰的识别能力仍有提升空间。

模型整体分类效果优秀，仅在部分服饰之间（如 Shirt 与 Top/Pullover）存在可解释的混淆。进一步提升精度可考虑引入注意力机制或更深层网络结构以加强特征辨别能力，对 Shirt/T-shirt 类别增加局部特征增强（如领口、袖口特写）或使用更高的 loss 权重。

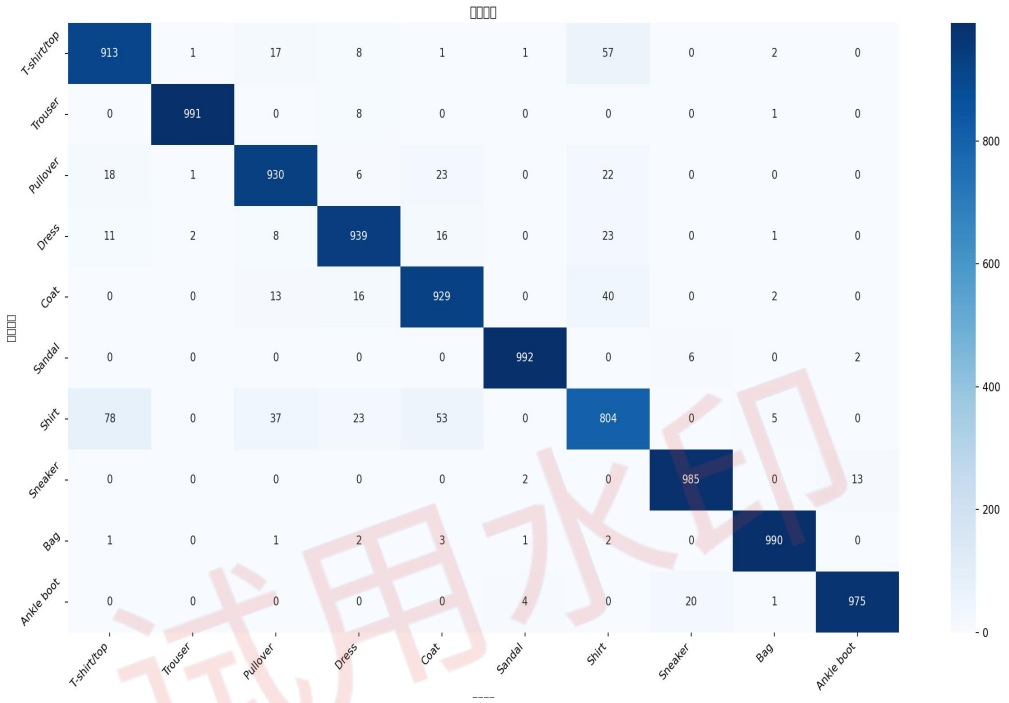

图 4-3 混淆矩阵图

4.3 性能分析

模型在测试集上整体达到了 94% 的准确率，10 个类别的精度、召回率与 F1 分数均衡良好，详见下表：

表 4-1 分类性能表

类别	Precision	Recall	F1-score	Support
T-shirt/top	0.89	0.91	0.90	1000
Trouser	1.00	0.99	0.99	1000
Pullover	0.92	0.93	0.93	1000
Dress	0.94	0.94	0.94	1000
Coat	0.91	0.93	0.92	1000
Sandal	0.99	0.99	0.99	1000

Shirt	0.85	0.80	0.83	1000
Sneaker	0.97	0.98	0.98	1000
Bag	0.99	0.99	0.99	1000
Ankle boot	0.98	0.97	0.98	1000
平均值	0.94	0.94	0.94	10000

从宏平均（macro avg）和加权平均（weighted avg）来看，模型在类别不平衡的情况下仍保持高度均衡的识别能力。尤其是 F1-score 均为 0.94，说明模型在精度与召回之间达成了良好平衡。下面继续对类别性能进行分析。

(1) 类别 0 (T-shirt/top)：

精确度（Precision）： 0.89

召回率（Recall）： 0.91

F1 分数： 0.90

表现： 模型对 T-shirt/top 类别具有较高的区分能力，但仍存在部分样本被误分类为 Shirt（混淆矩阵中有 57 例）。这是由于两者外观相近，易导致判别困难。

(2) 类别 1 (Trouser)：

精确度： 1.00

召回率： 0.99

F1 分数： 0.99

表现： 本类别表现最优，几乎无误分类情况，模型能够稳定准确地识别该类，是训练中最易区分的类别之一。

(3) 类别 2 (Pullover)：

精确度： 0.92

召回率： 0.93

F1 分数： 0.93

表现： 模型对 Pullover 的识别表现良好，但仍有部分被误分为 Shirt 和 Coat，说明模型对上装类服饰的边界还存在一定模糊。

(4) 类别 3 (Dress)：

精确度： 0.94

召回率： 0.94

F1 分数： 0.94

表现： Dress 识别精度高，召回率均衡，说明模型在该类上的判别特征较为充分，属于泛化能力较强的类别。

(5) 类别 4 (Coat)：

精确度： 0.91

召回率： 0.93

F1 分数： 0.92

表现： 与 Pullover 和 Shirt 类之间存在一定混淆，模型对衣物外套类仍需进一步增强区分性特征学习。

(6) 类别 5 (Sandal)：

精确度： 0.99

召回率： 0.99

F1 分数： 0.99

表现： Sandal 类别几乎无误判，是模型最擅长识别的类别之一，表明鞋类与衣物类之间的边界清晰且易于建模。

(7) 类别 6 (Shirt)：

精确度： 0.85

召回率： 0.80

F1 分数： 0.83

表现： 这是模型表现最弱的类别，误分类情况较多，尤其容易与 T-shirt/top、Pullover 和 Coat 混淆，说明模型对中性上衣类特征提取仍需优化。

(8) 类别 7 (Sneaker)：

精确度： 0.97

召回率： 0.98

F1 分数： 0.98

表现： 与 Sandal 类似，Sneaker 识别准确度极高，说明模型对鞋类具有良好的鲁棒性和特征稳定性。

(9) 类别 8 (Bag)：

精确度： 0.99

召回率： 0.99

F1 分数： 0.99

表现： Bag 类别识别极为精准，误判样本极少，是模型泛化效果最强的类别之一。

(10) 类别 9 (Ankle boot)：

精确度： 0.98

召回率： 0.97

F1 分数： 0.98

表现： 类似于 Sneaker，模型能稳定识别 Ankle boot，偶有误判为 Sneaker 但整体影响较小，表现优异。

5 结论与可能改进

本项目成功地将预训练的 ResNet-18 模型应用于 Fashion-MNIST 数据集的图像分类任务。通过适当的数据预处理和训练策略，模型在测试集上达到了 **94.3%** 的准确率，在另一次运行时达到了 **94.7%**准确率(但忘记保存该模型)显示出强大的特征提取和分类能力。训练过程中，模型的损失函数持续下降，准确率稳步提升，验证了训练过程的有效性。

然而，实验中也暴露出一些问题和改进空间。例如，类别“Shirt”的 F1 分数仅为 0.83，明显低于其他类别，表明模型在区分某些相似类别时存在困难。此外，尽管训练准确率接近 99.5%，但验证准确率停留在 94% 左右，提示可能存在轻微的过拟合现象。

根据本实验结果以及 Fashion-MNIST 官方汇总中的算法对比分析，我们可以结合实际表现和社区最佳实践，总结出如下改进方向：

（1）增强数据增强策略

当前仅使用了基础的随机水平/垂直翻转与旋转。参考 Kyriakos Efthymiadis 的数据处理措施和 WRN 等高性能模型的做法，可以引入更强的数据增强策略：

随机裁剪（Random Crop）：模拟图像偏移，提升鲁棒性；

颜色抖动（Color Jitter）：虽然 Fashion-MNIST 是灰度图，但模拟对比度/亮度变化仍可提升泛化；

随机擦除（Random Erasing）：对模型隐藏部分信息，提高其全局特征理解能力（WRN + Random Erasing 达到 96.3%）；

Mixup 或 CutMix：增强数据分布平滑性，提高判别边界的泛化能力。

（2）尝试更深或更宽的模型架构

当前使用的 ResNet-18 架构在许多视觉任务上表现良好，但该模型为解决 imagenet 的 1000 分类问题而设计，用到这个入门级别的数据集上似乎过于庞大了，而且也容易过拟合但，针对 Fashion-MNIST 数据集，仍有更优结构可选：

WideResNet（WRN-28-10）：结构宽度大，减少残差连接阻断现象，在表格中准确率高达 95.9%~96.3%；

DenseNet：鼓励特征重用，参数更少、效果更稳（DenseNet-BC 768K 参数已达 95.4%）；

MobileNet/VGG16：如需部署轻量化模型，苏剑林提出的 MobileNet 可提供 95.0% 左右性能，适合边缘部署；

Google AutoML 或 DENSER 等神经架构搜索结果：在准确率和模型结构复杂度之间找到更优解。

（3）更长训练周期与更优超参数

虽然本项目已经训练 20 个 epoch，但多个公开高性能模型（如 SqueezeNet, WRN）训练了 100~200 个 epoch 并使用了学习率调度（如 Cosine Annealing、Cyclic LR）与早停策略，在提升准确率的同时防止过拟合。

在总结这些可能的改进时，我也尝试在原来代码上进行一些改进，但要么提升效果不明显，要么因处理过于复杂，训练周期过长导致个人电脑无法承担，目前公开的资料中，最高能实现的准确率也不过 96%，可见想要攻克这个经典的数据集真是任重道远。

参考资料

[1]Zalando Research. Fashion-MNIST 中文官方资料. GitHub 仓库, 访问于 2025 年 5 月 15 日, https://github.com/zalandoresearch/fashion-mnist/blob/master/README.zh-CN.md

[2]Mr.Ai_Wu. ResNet-18 超详细介绍！！！！. CSDN 博客, 最后更新于 2023 年 9 月 8 日. 访问于 2025 年 5 月 15 日, https://blog.csdn.net/m0_64799972/article/details/132753608

[3]Kyriakos Efthymiadis. Fashion-MNIST ResNet18 实现. GitHub 仓库, 访问于 2025 年 5 月 15 日, https://github.com/kefth/fashion-mnist

[4]苏剑林. 《Fashion MNIST 的一个 baseline (MobileNet 95%)》. 博客文章, 2017 年 8 月 27 日. 访问于 2025 年 5 月 15 日, https://kexue.fm/archives/4556

[5]韩晓. Fashion-MNIST：年度回顾. 博客文章, 2018 年 9 月 28 日. 访问于 2025 年 5 月 15 日, https://hanxiao.io/2018/09/28/Fashion-MNIST-Year-In-Review/