西安电子科技大学

Python 网络数据处理
大作业报告

学院 机电工程学院 专业 机械

任课教师 李隐峰 开课年份 2024 年

班级 04 班

学号 24041212672

姓名 涂威海

邮箱 17304013484@163.com

手机 17304013484 完成日期 2024-12-20

成绩

目录

题目:手写数字识别 4
一、 题目要求....4
二、 设计思路....5
2.1 问题分析与解决方案....5
2.2 所用 Python 库和版本介绍 5
2.2.1 核心数据处理与科学计算....5
2.2.2 深度学习与图像处理....5
2.2.3 数据可视化....6
2.3.4 交互与调试....6
2.3.5 系统工具....6
2.3.6 依赖管理....7
2.3.7 其他辅助工具....7
2.3.8 库的关键用途....7
2.3 编程算法与方法....8
2.3.1 数据加载与预处理：....8
2.3.2 模型设计与训练：....8
2.3.3 模型训练与验证：....8
2.3.4 性能评估：....9
2.3.5 结果可视化：....9
2.3.6 测试集预测：....9

2.4 流程图.....10
三、 结果展示.....11
3.1 数据集介绍.....11
3.2 数据分析.....11
3.3 数据处理与封装.....13
3.3.1 数据集划分.....13
3.3.2 将数据转为 tensor 张量.....13
3.3.3 数据封装.....13
3.4 模型训练.....14
3.4.1 resnet18 模型.....14
3.4.2 CNN 模型.....15
3.4.3 FCNN 模型.....16
3.5 结果分析.....17
3.5.1 混淆矩阵.....17
3.5.2 查看错误分类样本.....19
3.6 加载最佳模型.....19
四、 课程收获建议.....21
4.1 收获.....21
4.2 建议.....21
五、 参考文献.....22
附录.....23

题目:手写数字识别

一、题目要求

MNIST（“修改版美国国家标准与技术研究所”）是事实上的计算机视觉“hello world”数据集。自 1999 年发布以来，这个经典的手写图像数据集，一直是对分类算法进行基准测试的基础。随着新的机器学习技术的出现，MNIST 仍然是研究人员和学习者的可靠资源。

在本次比赛中，您的目标是从数万张手写图像的数据集中正确识别数字。您将使用从 Ignite 计划中学到的核心机器学习和计算机视觉概念来实现这一目标！

题目来源：https://www.kaggle.com/competitions/exploretechla-ignite-digit-recognition-challenge/overview

二、设计思路

2.1 问题分析与解决方案

本代码的主要目标是使用深度学习模型对手写数字识别任务进行分类。具体来说：

1、数据处理：加载、分析、预处理手写数字数据集（MNIST 格式）。

2、模型训练：训练不同深度学习模型，包括 ResNet18、自定义的 CNN 和 FCNN，评估在验证集上的性能。

3、性能评估：使用混淆矩阵、准确率和预测错误样本可视化等方法评估模型表现。

4、测试集预测：使用训练好的最佳模型，对测试数据进行预测，并输出结果。

5、网络结构可视化：通过图表展示 CNN 网络的结构以及训练结果。

2.2 所用 Python 库和版本介绍

2.2.1 核心数据处理与科学计算

1、pandas==2.2.3：高性能数据操作和分析库，用于加载 CSV 文件、数据预处理、以及结果保存。

2、NumPy==2.0.2：科学计算库，支持多维数组操作，广泛用于数值计算。

3、scikit-learn==1.6.0：用于数据划分、评估指标（如混淆矩阵和准确率）、以及简单机器学习功能。

4、SciPy==1.13.1：提供优化、积分、插值、线性代数等高级科学计算工具。

2.2.2 深度学习与图像处理

1、torch==2.5.1：PyTorch 深度学习框架，用于模型构建、训练和评估。

2、torchvision==0.20.1：PyTorch 的计算机视觉工具包，用于加载预训练模型（如 ResNet18）。

3、pillow==11.0.0：图像处理库，用于图像加载和保存。

4、joblib==1.4.2：用于模型的参数保存和加载。

2.2.3 数据可视化

1、matplotlib==3.9.4：主要的绘图库，用于绘制训练曲线、网络结构图和结果可视化。

2、seaborn==0.13.2：高级统计数据可视化库，用于绘制混淆矩阵和数据分布图。

3、matplotlib-inline==0.1.7：在交互式环境（如 Jupyter Notebook）中内嵌绘图。

2.3.4 交互与调试

1、ipython==8.18.1：增强的交互式 Python 环境，支持代码高亮和自动补全。

2、jupyter_client==8.6.3 和 jupyter_core==5.7.2：支持 Jupyter Notebook 的核心组件。

3、debugpy==1.8.11：Python 调试工具，用于代码调试。

2.3.5 系统工具

1、psutil==6.1.0：用于监控和管理系统资源（如 CPU 和内存使用情况）。

2、threadpoolctl==3.5.0：用于控制线程池，优化多线程计算性能。

3、 platformdirs==4.3.6 和 filelock==3.16.1：用于跨平台的文件路径操作和文件锁定。

2.3.6 依赖管理

1、packaging==24.2：解析和比较版本信息，用于依赖管理。

2、importlib_metadata==8.5.0 和 importlib_resources==6.4.5：提供 Python 的导入库功能支持。

3、tzdata==2024.2 和 pytz==2024.2：处理时区相关问题。

2.3.7 其他辅助工具

1、fonttools==4.55.3 和 kiwisolver==1.4.7：用于绘图库（如 Matplotlib）的字体和布局支持。

2、pyparsing==3.2.0 和 cycler==0.12.1：支持 Matplotlib 的配置和绘图属性循环。

3、 sympy==1.13.1 和 mpmath==1.3.0：符号数学计算库，用于数学公式解析和高精度计算。

2.3.8 库的关键用途

库名称	用途
pandas	数据加载与预处理
numpy	数值计算与数组操作
scikit-learn	数据划分与评估指标计算
torch	模型构建、训练与评估
torchvision	加载预训练模型与图像处理

matplotlib	绘制训练曲线、混淆矩阵与结果可视化
seaborn	高级统计图表绘制
pillow	图像读取与保存

2.3 编程算法与方法

2.3.1 数据加载与预处理：

从 CSV 文件中加载训练集和测试集。

使用 train_test_split 划分训练集和验证集。

将数据转换为 PyTorch Tensor 格式，便于模型训练和评估。

2.3.2 模型设计与训练：

1、ResNet18：

（1）通过 torchvision.models.resnet18() 加载预训练的 ResNet18 网络。

（2）将输入图像扩展为 3 通道以适配 ResNet18 结构。

2、自定义 CNN：

（1）使用两个卷积层和两个最大池化层进行特征提取。

（2）使用全连接层进行分类。

3、FCNN（全连接神经网络）：

（1）设计多层全连接结构，使用激活函数（ReLU、Tanh 和 ELU）进行非线性映射。

2.3.3 模型训练与验证：

1、使用交叉熵损失函数 nn.CrossEntropyLoss。

2、使用随机梯度下降优化器 optim.SGD 进行参数更新。

3、分别定义训练函数 model_train 和验证函数 model_validate，实现训练和评估过程。

2.3.4 性能评估：

1、准确率：使用 sklearn 中的 accuracy_score 计算验证集的准确率。

2、混淆矩阵：通过 confusion_matrix 和 seaborn.heatmap 绘制混淆矩阵，分析分类结果。

3、错误样本展示：展示验证集中被错误分类的图像及其真实标签和预测标签。

2.3.5 结果可视化：

1、绘制训练和验证的损失曲线、准确率曲线。

2、使用 matplotlib 和 patches 绘制 CNN 网络结构图。

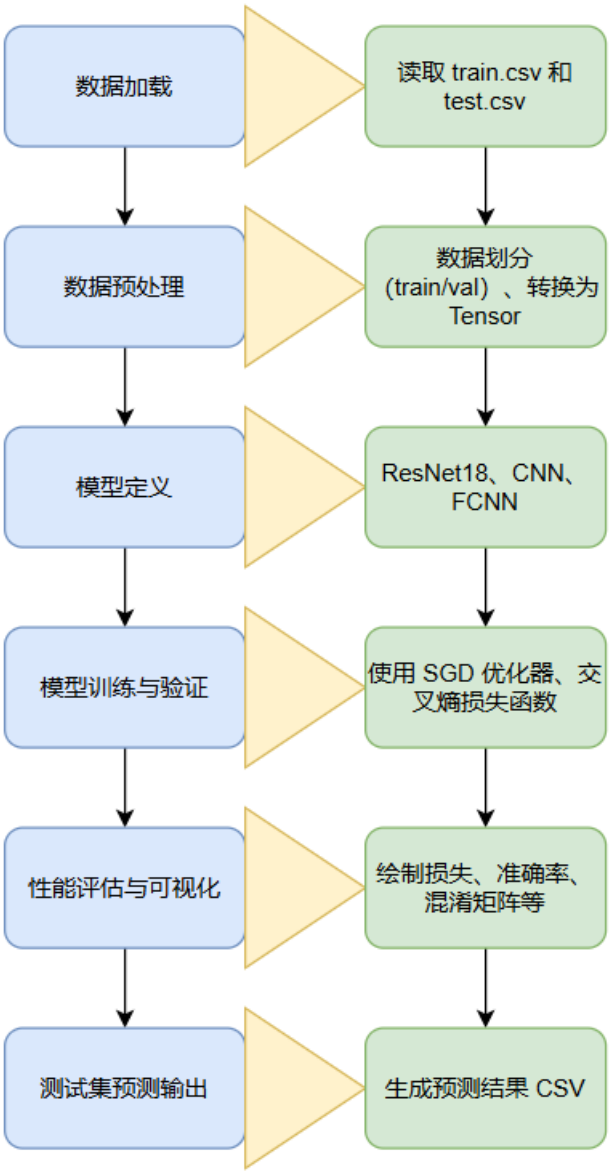
3、显示训练过程中的滤波器（卷积核）权重分布。

2.3.6 测试集预测：

1、加载最佳模型权重，对测试数据集进行预测。

2、输出结果，并保存为 CSV 文件。

2.4 流程图


数据加载
读取 train.csv 和 test.csv
数据预处理
数据划分 (train/val) 、转换为 Tensor
模型定义
ResNet18、CNN、FCNN
模型训练与验证
使用 SGD 优化器、交叉熵损失函数
性能评估与可视化
绘制损失、准确率、混淆矩阵等
测试集预测输出
生成预测结果 CSV

三、结果展示

3.1 数据集介绍

竞赛使用的是 MNIST (Modified National Institute of Standards and Technology，美国国家标准与技术研究院修改版) 手写图像数据集，其中训练集 42000 条，测试集 28000 条，每条数据有 784 个像素点，即原始图像的像素为 28 * 28。训练集中的 Label 列表示手写数字的类别（共 10 个类别，0-10）。

3.2 数据分析

```
# %%
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import warnings
from torchvision import models
import torch.optim as optim
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import torch
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import seaborn as sns
from sklearn.model_selection import train_test_split

train = pd.read_csv("./dataset/train.csv")
test = pd.read_csv("./dataset/test.csv")
```

train.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

test.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28000 entries, 0 to 27999
Columns: 784 entries, pixel0 to pixel783
dtypes: int64(784)
memory usage: 167.5 MB
```

查看空缺值

```
#-------------------------------------------------------------------------------------------------------------
# train_data.isnull(): 返回一个与 train_data 相同维度的布尔值数据框，其中 True 表示该位置存在缺失值，False 表示没有缺失值
# any(): 对每一列进行操作，如果某列中存在至少一个 True 那么这一列的结果就是 True；否则就是 False，结果是一个布尔类型的 Serie
# describe(): 统计摘要，包括总列数、唯一值个数、最频繁出现的值（top）及其出现频率（freq）
#-------------------------------------------------------------------------------------------------------------
train.isnull().any().describe()
```

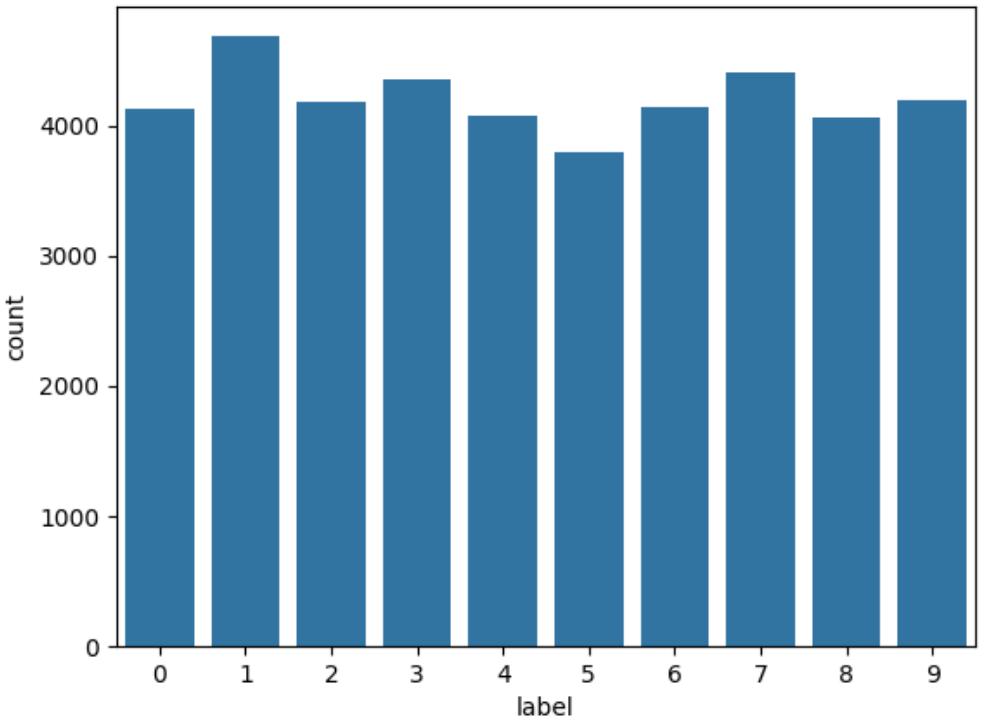
```
count       785
unique        1
top       False
freq        785
dtype: object
```

由结果可知，仅有一个唯一值 False，且出现 785 次，故训练集中无缺失值

查看类别统计

sns.countplot(x=train['label']);

plt.show()



3.3 数据处理与封装

3.3.1 数据集划分

将训练集划分为训练集和验证集。

```
# 分割特征和标签
train_labels = train["label"]
train= train.drop(labels=["label"], axis=1)

# 划分训练集和验证集
X_train, X_val, y_train, y_val = train_test_split(train, train_labels, test_size = 0.2, random_state=41)
print("训练集大小：{}，验证集大小：{}".format(len(X_train), len(X_val)))
```

训练集大小：33600，验证集大小：8400

3.3.2 将数据转为 tensor 张量

dataFrame 和 Series 类型需要先转为 numpy 类型，

```
import torch
from torch.utils.data import DataLoader, TensorDataset

X_train_tensor = torch.tensor(X_train.values, dtype = torch.float32)
y_train_tensor = torch.tensor(y_train.values)

X_val_tensor   = torch.tensor(X_val.values, dtype = torch.float32)
y_val_tensor   = torch.tensor(y_val.values)

test_tensor = torch.tensor(test.values, dtype = torch.float32)
```

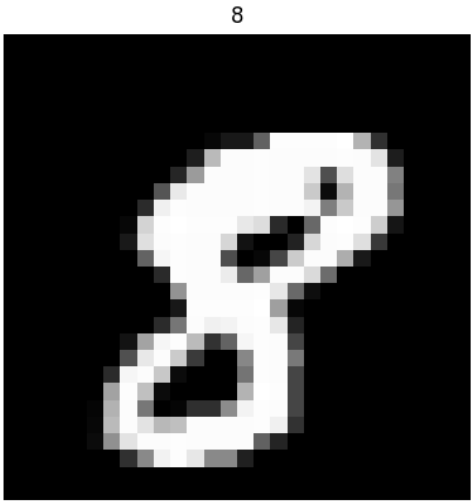
3.3.3 数据封装

使用 TensorDataset 创建包含数据特征和数据类别的 tensor 数据集，再用 DataLoader 划分封装数据集。封装数据集时，训练集中的 shuffle 参数设置为 True（随机打乱数据），可以防止模型学习到数据的顺序，从而提高模型的泛化能力；验证集和测试集 shuffle 参数设置为 False，能够保证测试集,预测结果的一致性和可比性。

```
train_tensor = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_tensor, batch_size=100, shuffle=True)

val_tensor   = TensorDataset(X_val_tensor, y_val_tensor)
val_loader   = DataLoader(val_tensor, batch_size=100, shuffle=False)

test_loader  = DataLoader(test_tensor, batch_size = 100, shuffle=False)
```

可视化训练集中的一张图像

```
plt.imshow(train.values[10].reshape(28,28), cmap='gray')
plt.axis("off")
plt.title(str(train_labels.values[10]));
plt.show()
```



3.4 模型训练

3.4.1 resnet18 模型

训练模型：

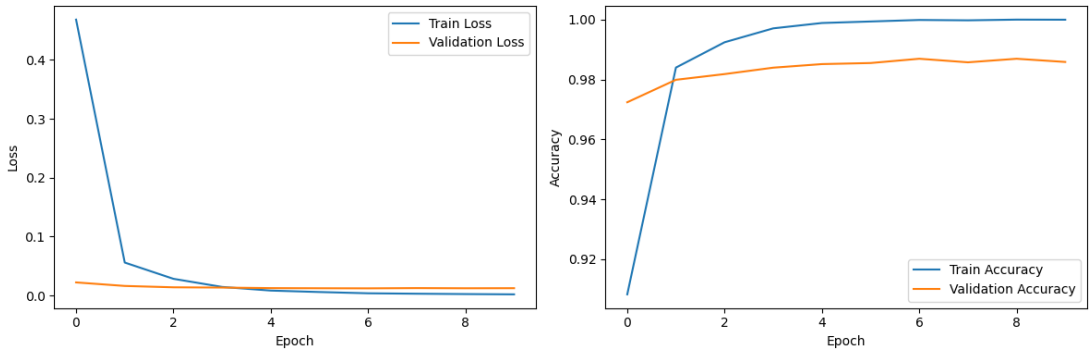
```
from torchvision import models
# 使用GPU训练模型（如果GPU可用的话）
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 调用resnet18
resnet_model = models.resnet18()
resnet_model = resnet_model.to(device)

# 记录训练集和验证集的损失值和准确率
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

train_val(resnet_model, "resnet18")
```

```
[第1轮训练完成，训练集中 Loss：0.48536758923104834，Accuracy：0.9060714285714285]
[第2轮训练完成，训练集中 Loss：0.05270050720095502，Accuracy：0.985]
[第3轮训练完成，训练集中 Loss：0.02555189496238849，Accuracy：0.9938392857142857]
[第4轮训练完成，训练集中 Loss：0.015233770400560129，Accuracy：0.9965773809523809]
[第5轮训练完成，训练集中 Loss：0.007979269749263213，Accuracy：0.9988690476190476]
[第6轮训练完成，训练集中 Loss：0.005160370017706771，Accuracy：0.9996428571428572]
[第7轮训练完成，训练集中 Loss：0.0035936778385803336，Accuracy：0.9998511904761904]
[第8轮训练完成，训练集中 Loss：0.0028507261213235324，Accuracy：0.9999107142857143]
[第9轮训练完成，训练集中 Loss：0.002293311161511589，Accuracy：0.9998809523809524]
[第10轮训练完成，训练集中 Loss：0.0019566422187857653，Accuracy：0.9998511904761904]
训练完成！最佳训练轮次：6，该轮次验证集上的准确率：0.9858333333333333
```

可视化损失值和准确率：

loss_acc_plot(train_losses, val_losses, train_accuracies, val_accuracies)

plt.show()



3.4.2 CNN 模型

训练模型：

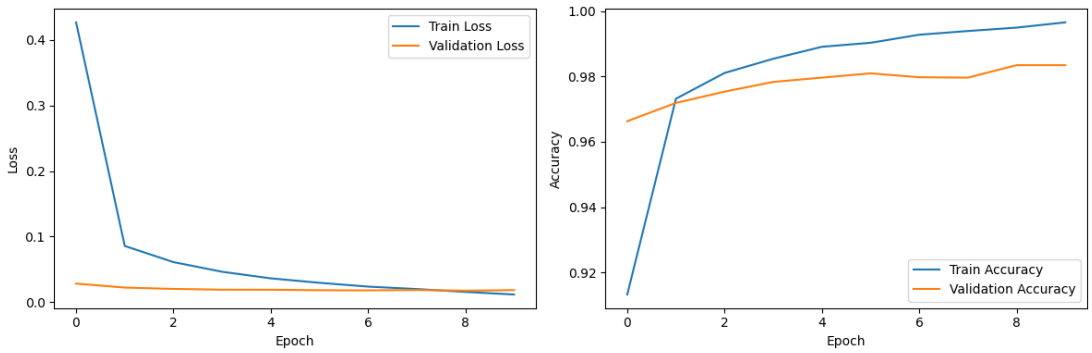
```
cnn_model = CNNModel()
cnn_model = cnn_model.to(device)

# 记录训练集和验证集的损失值和准确率
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

train_val(cnn_model, "cnn")
```

```
[第1轮训练完成，训练集中 Loss：1.004740373009727，Accuracy：0.8109226190476191]
[第2轮训练完成，训练集中 Loss：0.17581947764293068，Accuracy：0.945625]
[第3轮训练完成，训练集中 Loss：0.13953285299551985，Accuracy：0.9569345238095238]
[第4轮训练完成，训练集中 Loss：0.12599757561526662，Accuracy：0.9607738095238095]
[第5轮训练完成，训练集中 Loss：0.11612535938842311，Accuracy：0.9638095238095238]
[第6轮训练完成，训练集中 Loss：0.10294126443720113，Accuracy：0.9671726190476191]
[第7轮训练完成，训练集中 Loss：0.09651396153051228，Accuracy：0.9698214285714286]
[第8轮训练完成，训练集中 Loss：0.09004475945229864，Accuracy：0.9717857142857143]
[第9轮训练完成，训练集中 Loss：0.08583687311537298，Accuracy：0.9727083333333333]
[第10轮训练完成，训练集中 Loss：0.08039018868779142，Accuracy：0.9748214285714286]
训练完成！最佳训练轮次：9，该轮次验证集上的准确率：0.9721428571428572
```

可视化损失值和准确率：

loss_acc_plot(train_losses, val_losses, train_accuracies, val_accuracies)

plt.show()



3.4.3 FCNN 模型

训练模型：

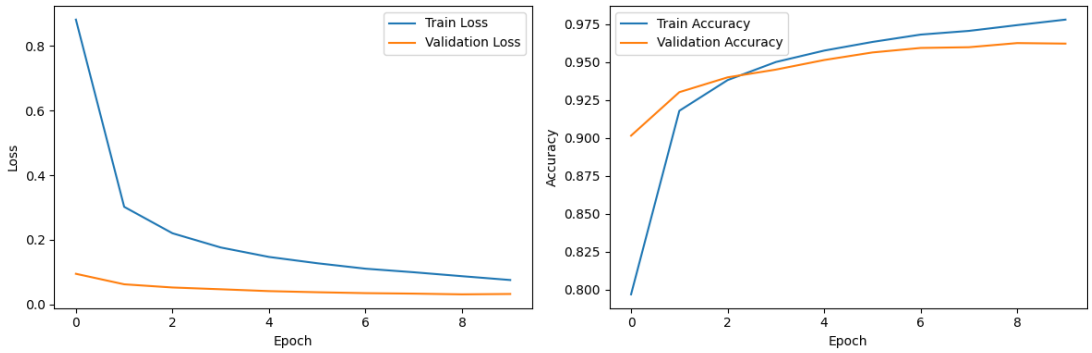
```python
# 记录训练集和验证集的损失值和准确率
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

input_dim = 28*28
# 可微调
hidden_dim = 150
output_dim = 10

fcnn_model = FCNNModel(input_dim, hidden_dim, output_dim)
fcnn_model = fcnn_model.to(device)
train_val(fcnn_model, "fcnn")
```

```
[第1轮训练完成，训练集中 Loss：0.8977700323753414，Accuracy：0.7811904761904762]
[第2轮训练完成，训练集中 Loss：0.3077347204089165，Accuracy：0.9172916666666666]
[第3轮训练完成，训练集中 Loss：0.2244828560034789，Accuracy：0.9372619047619047]
[第4轮训练完成，训练集中 Loss：0.18338089338725522，Accuracy：0.9476488095238095]
[第5轮训练完成，训练集中 Loss：0.15651956990006424，Accuracy：0.9541071428571428]
[第6轮训练完成，训练集中 Loss：0.1355396158483234，Accuracy：0.9603869047619048]
[第7轮训练完成，训练集中 Loss：0.11753073033122789，Accuracy：0.965625]
[第8轮训练完成，训练集中 Loss：0.10319345946135443，Accuracy：0.9705059523809524]
[第9轮训练完成，训练集中 Loss：0.09024346410296857，Accuracy：0.974047619047619]
[第10轮训练完成，训练集中 Loss：0.07875394061695606，Accuracy：0.9776785714285714]
训练完成！最佳训练轮次：10，该轮次验证集上的准确率：0.963452380952381
```

可视化损失值和准确率：

loss_acc_plot(train_losses, val_losses, train_accuracies, val_accuracies)

plt.show()

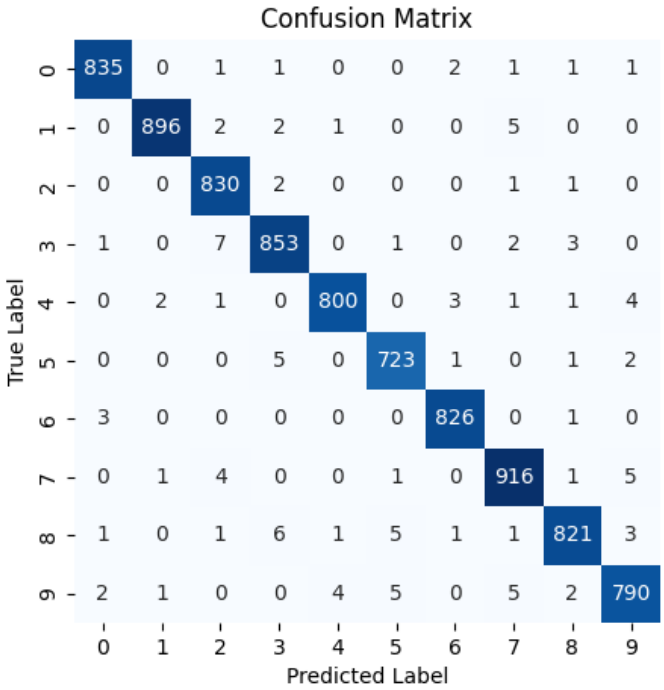


3.5 结果分析

3.5.1 混淆矩阵

计算混淆矩阵：

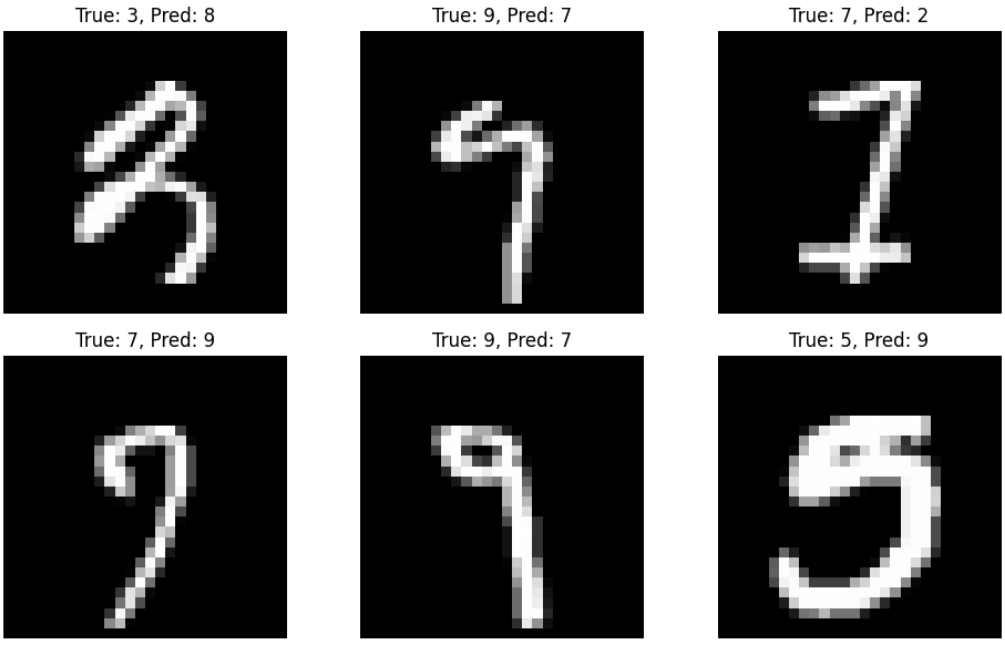
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(all_labels, all_predictions)
plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=range(10), yticklabels=range(10))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix");
```



横轴为预测类别，纵轴为实际类别。对标线上的值表示模型正确预测的样本数量，非对角线上的值表示模型错误预测的样本数量。对角线(1, 1)中的值 896 表示实际类别为 1 的样本中有 896 条被正确预测为 1；(1, 4)中的值为 1 表示实际类别为 1 的样本中有 1 个样本被错误预测为 4。

3.5.2 查看错误分类样本

```
# 查看预测错误的样本
incorrect_images = []
incorrect_labels = []
predicted_labels = []

with torch.no_grad():
    for step, data in enumerate(val_loader):
        images, labels = data
        images = images.view(-1, 1, 28, 28)
        outputs = best_resnet_model(images.expand(-1, 3, -1, -1))
        _, predicted = torch.max(outputs, 1)
        for i in range(len(predicted)):
            if predicted[i] != labels[i]:
                incorrect_images.append(images[i])
                incorrect_labels.append(labels[i])
                predicted_labels.append(predicted[i])
 # 展示部分预测错误的样本
num_samples = 6
fig, axes = plt.subplots(nrows=2, ncols=num_samples // 2, figsize=(10, 6))
axes = axes.flatten()

for i in range(num_samples):
    ax = axes[i]
    img = incorrect_images[i].reshape(28, 28)
    ax.imshow(img, cmap='gray')
    ax.set_title(f"True: {incorrect_labels[i]}, Pred: {predicted_labels[i]}")
    ax.axis('off')
plt.tight_layout()
plt.show()
```



3.6 加载最佳模型

保存的最佳模型中，resnet18、CNN 和 FCNN 在验证集中的准确率分别为

98.58%、97.21%和 96.35%，因此选择 resnet18 模型来预测测试集。

```
# 选择最佳模型对测试集进行预测
best_model = models.resnet18()
best_model.load_state_dict(torch.load("./resnet18_best_model.pth"))
predictions = []
with torch.no_grad():
    for data in test_loader:
        images = data.view(-1, 1, 28, 28).expand(-1, 3, -1, -1)
        outputs = best_model(images)
        _, predicted = torch.max(outputs, 1)
        predictions.extend(predicted.numpy())
# 保存预测结果
submission = pd.DataFrame(
    {'ImageId': range(1, len(test) + 1), 'Label': predictions})
print('Submission file created!')
```

四、 课程收获建议

4.1 收获

1、熟悉了 Python 的基本语法

2、掌握了使用 Pandas、NumPy 等库进行数据操作的技巧

3、从课堂上学到了一些计算机的拓展知识

4.2 建议

1、希望老师可以加入实际案例的讲解，让我们真正体会到 Python 能进行哪些数据处理

2、建议增加课堂讨论、答疑环节，帮助解决学习中的问题

五、 参考文献

[1]. kaggle：Digit Recognizer《手写数字识别》你的第一个图像识别竞赛项目

[2]. Pytorch Tutorial for Deep Learning Lovers

[3]. 面向深度学习爱好者的 Pytorch 教程

[4]. 嵩天. 程序设计基础:Python 语言[M].程序设计基础:PYTHON 语言. 2017.

附录