```
# %%
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"
# 每张图像的像素为 28 * 28 像素，总共为 784 像素点
train.head()
train.info()
test.info()
# ------------------------------------------------------------------------------
--------------------------#
# 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 Series
# 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()
# %%
# 数字类别统计
sns.countplot(x=train['label'])
plt.show()
# %%
# 分割特征和标签
train_labels = train["label"]
train = train.drop(labels=["label"], axis=1)
# %%
type(train_labels)
# %%
# 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)))
# %%
X_train.head()
# %%
# 将数据转为tensor张量,dataFrame和Series类型需要先转为numpy类型
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)
# %%
# ------------------------------------------------------------------------------
-----------------#
# 创建一个 TensorDataset 对象和一个 DataLoader 对象，用于按量载入dataset中的数据
# ------------------------------------------------------------------------------
-----------------#
train_tensor = TensorDataset(X_train_tensor, y_train_tensor)
# 将训练数据设置随机打乱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
train_loader = DataLoader(train_tensor, batch_size=100, shuffle=True)
val_tensor = TensorDataset(X_val_tensor, y_val_tensor)
# 确保验证集预测结果的一致性和可比性，shuffle设置为Flase（不打乱顺序）
val_loader = DataLoader(val_tensor, batch_size=100, shuffle=False)
# 确保测试集预测结果的一致性和可比性，shuffle设置为Flase（不打乱顺序）
test_loader = DataLoader(test_tensor, batch_size=100, shuffle=False)
# %%
plt.imshow(train.values[10].reshape(28, 28), cmap='gray')
plt.axis("off")
plt.title(str(train_labels.values[10]))
plt.show()
# %%
# ------------------------------------------------模型训练-------------------------
-------------------------#
# %%
"""
模型训练函数
Params:
    epoch:          训练轮次
    model:          预定义模型
    dataloader:     批处理数据
    criterion:      损失函数（交叉熵）
    optimizer:      优化器
Returns
    running_loss/len(train_loader)：本轮次（遍历一遍训练集）的平均损失
    sum_correct/train_num：本轮次（遍历一遍训练集）准确率
"""
def model_train(epoch, model, model_name, dataloader, criterion, optimizer):
    #     print("-------------------------Training-------------------------")
    # 设置模型为训练模式
    model.train()
    running_loss = 0.0
    # 训练集大小
    train_num = len(X_train)
    # 记录遍历一轮数据集后分类正确的样本数
    sum_correct = 0
    for step, data in enumerate(dataloader):
        images, labels = data
        if model_name == 'resnet18':
            # ------------------------------------------------------------------
-------------------------#
            # ResNet18 期望输入的形状为 [batch_size, channels, height, width]，其中
channels 为 3（RGB 图像）
            # expand(): 沿指定维度扩展张量(但不复制数据，只改变视图)
```

```
            # ------------------------------------------------------------------
---------------------------#
            images = images.view(-1, 1, 28, 28).expand(-1, 3, -1, -1)
        if model_name == 'cnn':
            # 自定义CNN的输入维度为 1
            images = images.view(-1, 1, 28, 28)
        # 模型为FCNN时无需转换
        images = images.to(device)
        labels = labels.to(device)
        # 清除上一次迭代的梯度信息，防止梯度累积
        optimizer.zero_grad()
        # ------------------------------------------------------------------
---------------------------#
        # outputs的尺寸[每次输入的样本数（batch_size），类别数]
        # 表示的含义：对应样本被分为某一类别的概率
        # ------------------------------------------------------------------
---------------------------#
        outputs = model(images)
        # 计算损失值
        loss = criterion(outputs, labels)
        # ------------------------------------------------------------------
---------------------------#
        # 计算损失函数相对于模型参数的梯度，并将这些梯度存储在每个参数的 .grad 属性中。
        # 随后，优化器会使用这些梯度来更新模型参数，从而逐步最小化损失函数，实现模型的训练
        # ------------------------------------------------------------------
---------------------------#
        loss.backward()
        # 使用优化器 optimizer 更新模型参数
        optimizer.step()
        running_loss += loss.item()
        # ------------------------------------------------------------------
---------------------------#
        # torch.max()函数返回两个值：每行的最大值和最大值的索引
        # _：表示忽略了第一个返回值（每行的最大值）
        # 1：寻找每行的最大值和索引
        # ------------------------------------------------------------------
---------------------------#
        predicted = torch.max(outputs, 1)
        # ------------------------------------------------------------------
---------------------------#
        # sum(): 将布尔张量转换为整数张量并对其进行求和，得到正确预测的总数。
        # 布尔值 True 计算为 1，False 计算为 0。
        # item(): 将单元素张量转换为 Python 标量值，便于计算
        # ------------------------------------------------------------------
---------------------------#
        correct = (predicted == labels).sum().item()
        sum_correct += correct
        train_acc = correct / len(labels)
        print("[Epoch {}, step: {}] Train Loss: {:.4f}, Train Acc:
{:.2f}%".format(epoch + 1, step+1, loss, train_acc*100))

        print("-------------------------Training-------------------------")
    return running_loss/len(train_loader), sum_correct/train_num
"""
模型评估函数
```

```
Params:
    epoch:          训练轮次
    model:          预定义模型
    dataloader:     批处理数据
    criterion:      损失函数（交叉熵）
Returns
    running_loss/len(train_loader)：本轮次（遍历一遍验证集）的平均损失
    sum_correct/train_num：本轮次（遍历一遍验证集）准确率
"""
def model_validate(epoch, model, model_name, dataloader, criterion):
    #     print("-----------------------Validating-----------------------")
    # 设置模型为测试模式
    model.eval()
    val_loss = 0.0
    # 训练集大小
    val_num = len(X_val)
    sum_correct = 0
    # 禁止梯度反传
    with torch.no_grad():
        for step, data in enumerate(dataloader):
            images, labels = data
            if model_name == 'resnet18':
                # ---------------------------------------------------------------
--------------------------#
                # ResNet18 期望输入的形状为 [batch_size, channels, height, width]，
其中 channels 为 3（RGB 图像）
                # expand(): 沿指定维度扩展张量(但不复制数据，只改变视图)
                # ---------------------------------------------------------------
--------------------------#
                images = images.view(-1, 1, 28, 28).expand(-1, 3, -1, -1)
            if model_name == 'cnn':
                # 自定义CNN的输入维度为 1
                images = images.view(-1, 1, 28, 28)
            # 模型为FCNN时无需转换
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            # 计算损失值
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            # ---------------------------------------------------------------
--------------------------#
            # torch.max()函数返回两个值：每行的最大值和最大值的索引
            # _：表示忽略了第一个返回值（每行的最大值）
            # 1：寻找每行的最大值和索引
            # ---------------------------------------------------------------
--------------------------#
            predicted = torch.max(outputs, 1)
            correct = (predicted == labels).sum().item()
            sum_correct += correct
            total = len(labels)
            val_acc = correct / total
            print("[Epoch {}, step: {}] Val Loss: {:.4f}, Val Acc:
{:.2f}%".format(epoch + 1, step+1, loss, val_acc*100))
```

```
            print("------------------------Validating-------------------------")
    return val_loss/len(train_loader), sum_correct/val_num
# %%
"""
模型整体训练与验证函数
Params:
    model:          预定义模型
"""
def train_val(model, model_name):
    # 定义损失函数（交叉熵）和优化器
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
    # 验证集上的最佳准确率和最佳轮次
    best_val_acc = 0.0
    best_epoch = 0
    for epoch in range(10):
        # 模型训练
        train_loss, train_acc = model_train(
            epoch, model, model_name, train_loader, criterion, optimizer)
        train_losses.append(train_loss)
        train_accuracies.append(train_acc)
        # 模型验证
        val_loss, val_acc = model_validate(
            epoch, model, model_name, val_loader, criterion)
        val_losses.append(val_loss)
        val_accuracies.append(val_acc)
        if val_acc > best_val_acc:
            best_val_acc = val_acc
            best_epoch = epoch + 1
            torch.save(model.state_dict(), model_name+'_best_model.pth')
        print("[第{}轮训练完成，训练集中 Loss：{}，Accuracy：{}]".format(
            epoch+1, train_loss, train_acc))
    print("训练完成！最佳训练轮次：{}，该轮次验证集上的准确率：{}".format(best_epoch,
best_val_acc))
"""
可视化损失值和准确率
"""
def loss_acc_plot(train_losses, val_losses, train_accuracies, val_accuracies):
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    # 默认情况下，plt.plot 会将 train_losses 的索引作为 X 轴的值
    plt.plot(train_losses, label='Train Loss')
    plt.plot(val_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(train_accuracies, label='Train Accuracy')
    plt.plot(val_accuracies, label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
# %%
```

```
# 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")
# %%
# 可视化resnet18的损失值和准确率
loss_acc_plot(train_losses, val_losses, train_accuracies, val_accuracies)
plt.show()
# %%
# 创建CNN模型
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        # 卷积层 1
        self.cnn1 = nn.Conv2d(in_channels=1, out_channels=16,
                              kernel_size=5, stride=1, padding=0)
        self.relu1 = nn.ReLU()
        # 最大池化层 1
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)
        # 卷积层 2
        self.cnn2 = nn.Conv2d(in_channels=16, out_channels=32,
                              kernel_size=5, stride=1, padding=0)
        self.relu2 = nn.ReLU()
        # 最大池化层 2
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        # 全连接层
        self.fc1 = nn.Linear(32 * 4 * 4, 10)

    def forward(self, x):
        # 卷积层 1
        out = self.cnn1(x)
        out = self.relu1(out)
        # 最大池化层 1
        out = self.maxpool1(out)
        # 卷积层 2
        out = self.cnn2(out)
        out = self.relu2(out)
        # 最大池化层 2
        out = self.maxpool2(out)
        # flatten层
        out = out.view(out.size(0), -1)
        # 全连接层
        out = self.fc1(out)

        return out
# %%
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")
# %%
# 可视化CNN的损失值和准确率
loss_acc_plot(train_losses, val_losses, train_accuracies, val_accuracies)
plt.show()
# %%
# 创建FCNN模型
class FCNNModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(FCNNModel, self).__init__()
        # 784 --> 150
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        # 激活函数
        self.relu1 = nn.ReLU()

        # 150 --> 150
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        # 激活函数
        self.tanh2 = nn.Tanh()

        # 150 --> 150
        self.fc3 = nn.Linear(hidden_dim, hidden_dim)
        # 激活函数
        self.elu3 = nn.ELU()

        # 150 --> 10
        self.fc4 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        # 784 --> 150
        out = self.fc1(x)
        out = self.relu1(out)

        # 150 --> 150
        out = self.fc2(out)
        out = self.tanh2(out)

        # 150 --> 150
        out = self.fc3(out)
        out = self.elu3(out)

        # 150 --> 10
        out = self.fc4(out)

        return out
# %%
# 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")
# %%
# 可视化FCNN的损失值和准确率
loss_acc_plot(train_losses, val_losses, train_accuracies, val_accuracies)
plt.show()
# %%
warnings.filterwarnings('ignore')
# 加载已保存的最佳CNN模型，验证其在验证集上的准确率
best_resnet_model = models.resnet18()
best_resnet_model.load_state_dict(torch.load("./resnet18_best_model.pth"))
# 设置模型为测试模式
best_resnet_model.eval()
# 计算验证集准确率
all_predictions = []
all_labels = []
# 禁止梯度反传
with torch.no_grad():
    for step, data in enumerate(val_loader):
        images, labels = data
        images = images.view(-1, 1, 28, 28).expand(-1, 3, -1, -1)
        outputs = best_resnet_model(images)
        # ------------------------------------------------------------------------
-------------------------#
        # torch.max()函数返回两个值：每行的最大值和最大值的索引
        # _：表示忽略了第一个返回值（每行的最大值）
        # 1：寻找每行的最大值和索引
        # ------------------------------------------------------------------------
-------------------------#
        _, predicted = torch.max(outputs, 1)
        all_predictions.extend(predicted)
        all_labels.extend(labels)
accuracy = accuracy_score(all_labels, all_predictions)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")
# %%
# ----------------------------------------------------------------------------------
-----------------#
# 计算混淆矩阵
# 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
# （1，4）中的值为1，表示实际类别为1的样本中有1个样本被预测为4
# ----------------------------------------------------------------------------------
-----------------#
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")
# %%
cm
# %%
# 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()
# %%
incorrect_images[1].shape
# %%
# 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})
# submission.to_csv('/kaggle/working/submission.csv', index=False)
print('Submission file created!')
# %%
```