基于 PyTorch 的 MNIST 手写数字识别

摘要

手写数字识别是计算机视觉领域的经典问题之一。MNIST 数据集[1]是一个常用的手写数字识别数据集，包含了大量的手写数字图像及其对应的标签。本项目旨在使用深度学习技术构建一个准确、高效的模型，对 MNIST 数据集中的手写数字进行识别。项目使用了 PyTorch[2]构建了模型来实现目标[3]：全连接神经网络和卷积神经网络（CNN）模型[4]。实验通过优化网络结构和超参数设置，实现了 99% 左右的测试准确率，验证了 CNN 在图像分类任务中的有效性。

关键词：深度学习，卷积神经网络，PyTorch

装 订 线

目 录

1 算法介绍及流程分析

1.1 算法流程

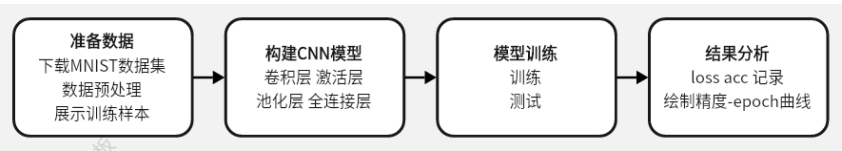

图 1.1 算法流程图

1.2 代码分析

1.2.1 准备数据与样本展示

数据加载采用 `torchvision.datasets.MNIST`，利用 `transforms.ToTensor()` 将数据转换为张量，并进行标准化处理（均值 0.1307，标准差 0.3081）。训练数据集和测试数据集分别采用 `DataLoader` 进行批量处理。通过 `matplotlib` 可视化部分数据样本，以了解数据分布情况。

```
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])
train_dataset = datasets.MNIST(root='./Assignment1/mnist_data', train=True, transform=transform, download=True)
test_dataset = datasets.MNIST(root='./Assignment1/mnist_data', train=False, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

1.2.2 卷积神经网络 (CNN) 模型的构建

卷积神经网络（Convolutional Neural Network, CNN）是一种专门用于处理具有类似网格结构的数据的神经网络，如图像数据。CNN 的基本结构包括卷积层、激活层、池化层和全连接层:

（1）**卷积层：**卷积层采用 `nn.Conv2d`，每一个卷积核的通道数量要求和输入通道数量一样，卷积核的总数和输出通道的数量一样。

（2）**激活层：**激活层采用 `nn.ReLU`，通过 ReLU 激活函数，增加网络的非线性表达能力。

（3）**池化层：**池化层采用 `nn.MaxPool2d`，通过 `MaxPool2d` 进行下采样，以减少计算量并提取更具判别性的特征。

（4）**全连接层：**全连接层采用 `nn.Linear`，将卷积层提取的特征映射到 10 个类别，并使用 `softmax` 进行分类。

模型的处理过程如图1.2所示：

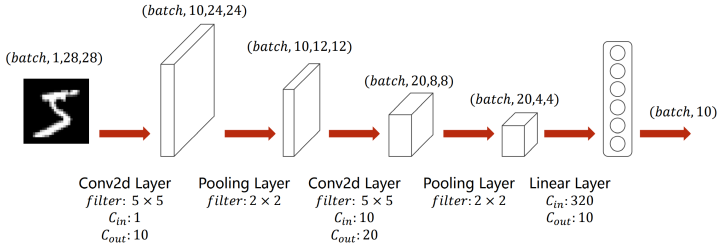

图 1.2 CNN 模型处理过程

CNN 模型的构建代码如下：

```
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 10, kernel_size=5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = torch.nn.Sequential(
            torch.nn.Conv2d(10, 20, kernel_size=5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2),
        )
```

```
        self.fc = torch.nn.Sequential(
            torch.nn.Linear(320, 50),
            torch.nn.Linear(50, 10),
        )

    def forward(self, x):
        batch_size = x.size(0)
        x = self.conv1(x)  # 一层卷积层，一层池化层，一层激活层（图是先卷积后激活再池化，差别不大）
        x = self.conv2(x)  # 再来一次
        x = x.view(batch_size, -1)
        x = self.fc(x)
        return x  # 最后输出的是维度为 10 的，也就是（对应数学符号的 0~9）

# 实例化模型
model = Net()
```

1.2.3 模型训练

模型训练采用 torch.optim.SGD 优化器，损失函数采用 nn.CrossEntropyLoss。

```
criterion = torch.nn.CrossEntropyLoss()  # 交叉熵损失
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=momentum)  # lr 学习率，momentum 冲量
```

训练过程中，每个 epoch 都会对训练集进行一次遍历，每次遍历都会对数据集进行随机打乱，以增加模型的泛化能力。训练过程中，记录训练集和测试集的损失值和准确率，以便后续分析。

```
def train(epoch):
    running_loss = 0.0  # 这整个 epoch 的 loss 清零
    running_total = 0
    running_correct = 0
    for batch_idx, data in enumerate(train_loader, 0):
        inputs, target = data
        optimizer.zero_grad()

        # forward + backward + update
        outputs = model(inputs)
        loss = criterion(outputs, target)

        loss.backward()
        optimizer.step()

        # 把运行中的 loss 累加起来，为了下面 300 次一除
        running_loss += loss.item()
        # 把运行中的准确率 acc 算出来
        _, predicted = torch.max(outputs.data, dim=1)
        running_total += inputs.shape[0]
        running_correct += (predicted == target).sum().item()

def test():
    correct = 0
    total = 0
    with torch.no_grad():  # 测试集不用算梯度
        for data in test_loader:
            images, labels = data
            outputs = model(images)
            _, predicted = torch.max(outputs.data, dim=1)
            total += labels.size(0)  # 张量之间的比较运算
            correct += (predicted == labels).sum().item()
    acc = correct / total
    print('[%d / %d]: Accuracy on test set: %.1f %% ' % (epoch+1, EPOCH, 100 * acc))  # 求测试的准确率
    return acc

if __name__ == '__main__':
    acc_list_test = []
    for epoch in range(EPOCH):
        train(epoch)
        # if epoch % 10 == 9:  # 每训练 10 轮 测试 1 次
        acc_test = test()
        acc_list_test.append(acc_test)

    plt.plot(acc_list_test)
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy On TestSet')
    plt.show()
```

2 实验结果与分析

2.1 训练集与测试集数据

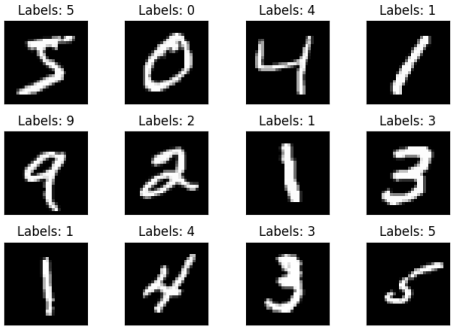

图 2.1 训练集与测试集数据

2.2 模型结果分析

共进行 20 轮次的训练和测试：每一轮的训练中，每 300 小批量数据输出一次损失值和准确率；每一轮训练结束后进行一次测试，并打印其在测试集上的准确率（图2.2）。20 轮后，在训练集上的平均识别准确率达到 98.8%，在测试集上的准确率达到 99.1%，其中训练集上准确率如图2.3所示，测试集上准确率如图2.4所示。

```
[1,   300]: loss: 0.719 , acc: 80.12 %
[1 / 20]: Accuracy on test set: 94.7 %
[2,   300]: loss: 0.168 , acc: 95.07 %
[2 / 20]: Accuracy on test set: 96.7 %
[3,   300]: loss: 0.118 , acc: 96.39 %
[3 / 20]: Accuracy on test set: 97.1 %
[4,   300]: loss: 0.095 , acc: 97.21 %
[4 / 20]: Accuracy on test set: 97.9 %
[5,   300]: loss: 0.079 , acc: 97.59 %
[5 / 20]: Accuracy on test set: 97.9 %
[6,   300]: loss: 0.071 , acc: 97.76 %
[6 / 20]: Accuracy on test set: 98.1 %
[7,   300]: loss: 0.063 , acc: 98.02 %
[7 / 20]: Accuracy on test set: 98.2 %
[8,   300]: loss: 0.057 , acc: 98.29 %
[8 / 20]: Accuracy on test set: 98.3 %
[9,   300]: loss: 0.056 , acc: 98.22 %
[9 / 20]: Accuracy on test set: 98.6 %
[10,   300]: loss: 0.052 , acc: 98.36 %
[10 / 20]: Accuracy on test set: 98.4 %
[11,   300]: loss: 0.048 , acc: 98.53 %
[11 / 20]: Accuracy on test set: 98.6 %
[12,   300]: loss: 0.044 , acc: 98.68 %
[12 / 20]: Accuracy on test set: 98.6 %
[13,   300]: loss: 0.043 , acc: 98.66 %
[13 / 20]: Accuracy on test set: 98.7 %
[14,   300]: loss: 0.042 , acc: 98.69 %
[14 / 20]: Accuracy on test set: 98.6 %
[15,   300]: loss: 0.039 , acc: 98.78 %
[15 / 20]: Accuracy on test set: 98.6 %
[16,   300]: loss: 0.036 , acc: 98.87 %
[16 / 20]: Accuracy on test set: 98.7 %
[17,   300]: loss: 0.035 , acc: 98.89 %
[17 / 20]: Accuracy on test set: 98.7 %
[18,   300]: loss: 0.033 , acc: 98.96 %
[18 / 20]: Accuracy on test set: 98.9 %
[19,   300]: loss: 0.033 , acc: 98.91 %
[19 / 20]: Accuracy on test set: 98.6 %
[20,   300]: loss: 0.030 , acc: 99.09 %
[20 / 20]: Accuracy on test set: 98.8 %
```

```
[1,   300]: loss: 0.848 , acc: 77.18 %
[1 / 20]: Accuracy on test set: 94.0 %
[2,   300]: loss: 0.173 , acc: 94.95 %
[2 / 20]: Accuracy on test set: 96.3 %
[3,   300]: loss: 0.122 , acc: 96.41 %
[3 / 20]: Accuracy on test set: 97.0 %
[4,   300]: loss: 0.098 , acc: 97.11 %
[4 / 20]: Accuracy on test set: 97.5 %
[5,   300]: loss: 0.083 , acc: 97.56 %
[5 / 20]: Accuracy on test set: 97.6 %
[6,   300]: loss: 0.072 , acc: 97.82 %
[6 / 20]: Accuracy on test set: 98.1 %
[7,   300]: loss: 0.064 , acc: 98.05 %
[7 / 20]: Accuracy on test set: 98.3 %
[8,   300]: loss: 0.059 , acc: 98.23 %
[8 / 20]: Accuracy on test set: 98.2 %
[9,   300]: loss: 0.056 , acc: 98.32 %
[9 / 20]: Accuracy on test set: 98.6 %
[10,   300]: loss: 0.051 , acc: 98.42 %
[10 / 20]: Accuracy on test set: 98.6 %
[11,   300]: loss: 0.048 , acc: 98.46 %
[11 / 20]: Accuracy on test set: 98.7 %
[12,   300]: loss: 0.047 , acc: 98.59 %
[12 / 20]: Accuracy on test set: 98.6 %
[13,   300]: loss: 0.043 , acc: 98.73 %
[13 / 20]: Accuracy on test set: 98.8 %
[14,   300]: loss: 0.043 , acc: 98.70 %
[14 / 20]: Accuracy on test set: 99.0 %
[15,   300]: loss: 0.041 , acc: 98.74 %
[15 / 20]: Accuracy on test set: 98.9 %
[16,   300]: loss: 0.037 , acc: 98.88 %
[16 / 20]: Accuracy on test set: 98.9 %
[17,   300]: loss: 0.037 , acc: 98.88 %
[17 / 20]: Accuracy on test set: 99.0 %
[18,   300]: loss: 0.036 , acc: 98.91 %
[18 / 20]: Accuracy on test set: 99.0 %
[19,   300]: loss: 0.033 , acc: 99.03 %
[19 / 20]: Accuracy on test set: 99.1 %
[20,   300]: loss: 0.032 , acc: 98.98 %
[20 / 20]: Accuracy on test set: 99.1 %
```

图 2.2 训练集与测试集准确率

（1）**损失变化趋势：**训练过程中，损失随着训练轮数的增加逐渐下降。从图2.2可以看出，初始时 `loss` 较大，随着训练的进行，损失逐步降低，并最终趋于稳定。这说明模型在不断学习特征，并收敛到较优解。

（2）**准确率变化趋势：**从 `accuracy` 曲线来看，CNN 训练的准确率随着 `epoch` 的增加而逐步

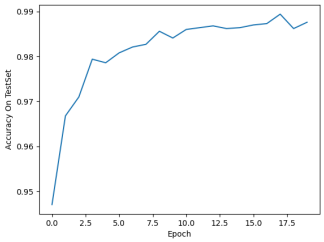

图 2.3 训练集结果

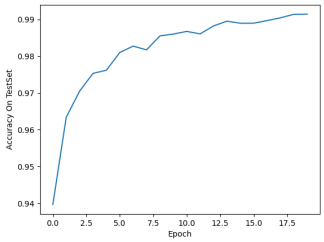

图 2.4 测试集结果

上升。实验表明，在 10 轮之后，模型的准确率已基本趋于稳定，表明模型已经很好地学习到了手写数字的特征。

2.3 实验结果分析

通过实验验证，CNN 在 MNIST 手写数字分类任务上表现优异，能够有效提取图像特征，并达到 99% 左右的准确率。CNN 通过卷积和池化操作有效提取图像特征，相比传统全连接神经网络（FNN），具备更强的识别能力。

3 总结

本实验成功构建了一个基于 CNN 的手写数字识别系统，并在 MNIST 数据集上取得了较高的分类准确率。实验结果进一步验证了 CNN 在图像分类任务中的有效性和优越性：（1） **高效特征提取：**CNN 通过局部感受野、共享权重和池化操作，有效提取了数字图像的空间特征，减少了计算复杂度，并增强了模型的泛化能力。（2） **高准确率：**模型在手写数字识别任务上表现优异。（3） **快速收敛：**CNN 能够有效学习数据特征并快速收敛。（4） **鲁棒性强：**CNN 能够适应不同书写风格的数字，具有较强的泛化能力。未来的研究方向可围绕数据增强、网络结构优化和优化方法改进，以进一步提高模型的性能。

参考文献

[1] ZHU W. Classification of MNIST handwritten digit database using neural network[J]. Proceedings of the Research School of Computer Science, Australian National University, 2018: 2601.

[2] PASZKE A, GROSS S, MASSA F, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library[A/OL]. 2019. https://pytorch.org.

[3] CSDN. 用 PyTorch 实现 MNIST 手写数字识别[EB/OL]. 2021. https://blog.csdn.net/qq_45588019/article/details/120935828.

[4] GOODFELLOW I, BENGIO Y, COURVILLE A. Deep Learning[M]. MIT Press, 2016.