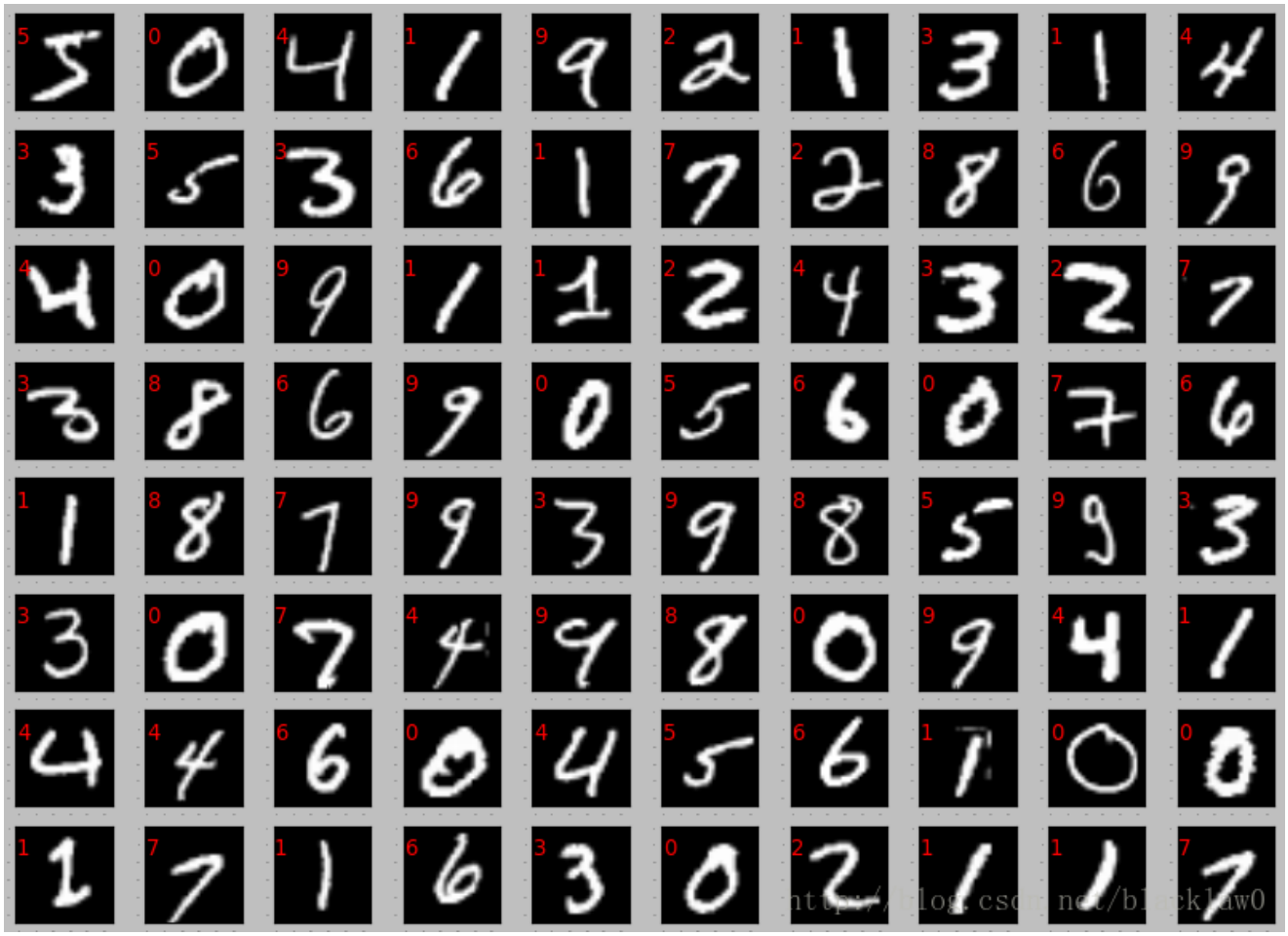
五、数字识别



姓名：游霄童

学号：21009200158

created：2024/05/05

实验目的

1. 了解 MNIST 数据集。
2. 掌握 LeNet-5 网络的结构。
3. 完成手写数字识别任务。

实验环境

- 硬件：Nvidia Jetson TX1 嵌入式系统
- 软件：PyTorch 深度学习框架

实验数据

- 数据集：MNIST 手写数字数据集

导入数据集

首先，通过 `torchvision` 模块加载 MNIST 数据集：

```
#导入MNIST数据集
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('data', train=True, download=True,
transform=transforms.Compose([
        transforms.ToTensor(), transforms.Normalize((0.1307,),
(0.3081,))#对数据进行归一化处理
    ])),
    batch_size=batch_size, shuffle=True
)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('data', train=False, transform=transforms.Compose([
        transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))
    ])),
    batch_size=test_batch, shuffle=True
)
```

模型构建

本实验采用 LeNet-5 网络架构。LeNet-5 是一种经典的卷积神经网络（CNN），主要用于图像分类任务。

LeNet-5 网络结构

LeNet-5 的网络结构由 7 层组成，不包括输入层，每一层都包含可训练的参数（权重），具体结构如下：

1. **输入层**：
    - 输入：尺寸为 28×28 的灰度图像。
2. **第一卷积层（C1）**：
    - 输入：1×28×28（1 个通道的 28×28 像素图像）。
    - 卷积核：6 个 5×5 卷积核。
    - 输出：6×28×28。
    - 激活函数：ReLU。
    - 计算过程：每个卷积核对输入图像进行卷积操作，得到 6 个特征图。由于 padding=2，卷积后特征图尺寸不变。

3. **第一池化层（S2）**：
    - 输入：6×28×28。
    - 池化方式：平均池化（Avg Pooling）。
    - 池化窗口：2×2，步长为 2。
    - 输出：6×14×14。
    - 计算过程：每个 2×2 区域取平均值，降低特征图的分辨率。
4. **第二卷积层（C3）**：
    - 输入：6×14×14。
    - 卷积核：16 个 5×5 卷积核。
    - 输出：16×10×10。
    - 激活函数：ReLU。
    - 计算过程：每个卷积核对输入特征图进行卷积操作，得到 16 个特征图。
5. **第二池化层（S4）**：
    - 输入：16×10×10。
    - 池化方式：平均池化。
    - 池化窗口：2×2，步长为 2。
    - 输出：16×5×5。
    - 计算过程：每个 2×2 区域取平均值。
6. **全连接层（C5）**：
    - 输入：16×5×5，展平为 1×400。
    - 全连接层：将输入连接到 120 个神经元。
    - 激活函数：ReLU。
    - 输出：1×120。
7. **全连接层（F6）**：
    - 输入：1×120。
    - 全连接层：将输入连接到 84 个神经元。
    - 激活函数：ReLU。
    - 输出：1×84。
8. **输出层**：
    - 输入：1×84。
    - 全连接层：将输入连接到 10 个神经元，对应 10 个类别（0-9）。
    - 激活函数：Softmax。
    - 输出：1×10。

LeNet-5 是一种结构简单但功能强大的卷积神经网络，通过卷积层和池化层的组合，它能够有效地提取图像中的特征，经过全连接层后完成分类任务。LeNet-5 在手写数字识别任务中表现出色，是卷积神经网络应用于图像识别的经典示例。

```
#定义网络结构LeNet-5
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 6, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(6, 16, 5),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.fc1 = nn.Sequential(
            nn.Linear(16*5*5, 120),
            nn.ReLU()
        )
        self.fc2 = nn.Sequential(
            nn.Linear(120, 84),
            nn.ReLU()
        )
        self.fc3 = nn.Linear(84, 10)


    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(-1, 16*5*5)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        return x
```

训练网络

训练网络采用随机梯度下降（SGD）优化算法，损失函数为交叉熵损失。

测试模型

为了评估模型的准确率，需要在测试集上进行验证。

```
# 定义测试模型函数
def test_model():
    examples = enumerate(test_loader)
    batch_idx, (example_data, example_targets) = next(examples)
    with torch.no_grad():
        output = model(example_data)
        pred = output.argmax(dim=1, keepdim=True)

        # 打印前10个预测结果和对应的真实标签
        print("Predicted labels:", pred[:10].view(-1))
        print("Actual labels:", example_targets[:10])

        # 显示预测结果的图像
        fig = plt.figure(figsize=(10, 6))
        for i in range(6):
            plt.subplot(2, 3, i+1)
            plt.tight_layout()
            plt.imshow(example_data[i][0], cmap='gray',
interpolation='none')
            plt.title("Prediction: {}".format(pred[i].item()))
            plt.xticks([])
            plt.yticks([])
```

```
# 计算模型的整体准确率
def calculate_accuracy():
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            output = model(data)
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
            total += target.size(0)
    accuracy = 100. * correct / total
    print('Accuracy: {}/{} ({:.0f}%)'.format(correct, total, accuracy))
```

实验结果

Train Epoch: 1 [57600/60000 (96%)] Loss: 0.311384

Test set: Average loss: 0.1468, Accuracy: 9570/10000 (96%)
Train Epoch: 2 [57600/60000 (96%)] Loss: 0.101105

Test set: Average loss: 0.0803, Accuracy: 9733/10000 (97%)
Train Epoch: 3 [57600/60000 (96%)] Loss: 0.049905

Test set: Average loss: 0.0749, Accuracy: 9760/10000 (98%)
Train Epoch: 4 [57600/60000 (96%)] Loss: 0.164802

Test set: Average loss: 0.0559, Accuracy: 9801/10000 (98%)
Train Epoch: 5 [57600/60000 (96%)] Loss: 0.050568

Test set: Average loss: 0.0479, Accuracy: 9833/10000 (98%)
Train Epoch: 6 [57600/60000 (96%)] Loss: 0.013274

Test set: Average loss: 0.0428, Accuracy: 9862/10000 (99%)
Train Epoch: 7 [57600/60000 (96%)] Loss: 0.012090

Test set: Average loss: 0.0411, Accuracy: 9859/10000 (99%)
Train Epoch: 8 [57600/60000 (96%)] Loss: 0.008077

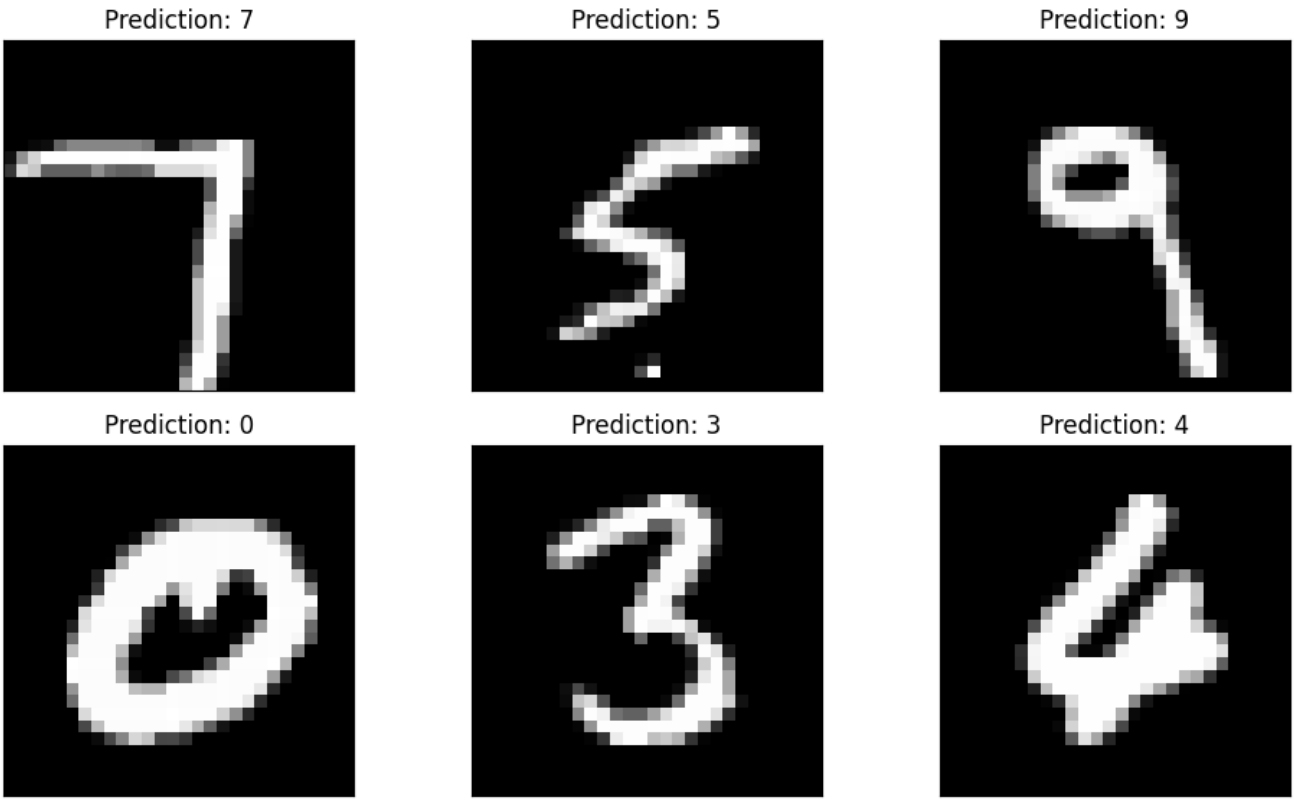
Test set: Average loss: 0.0371, Accuracy: 9872/10000 (99%)
Train Epoch: 9 [57600/60000 (96%)] Loss: 0.014966

Test set: Average loss: 0.0402, Accuracy: 9875/10000 (99%)
Train Epoch: 10 [57600/60000 (96%)] Loss: 0.033665

Test set: Average loss: 0.0347, Accuracy: 9891/10000 (99%)

结论

通过本次实验，我们成功地实现了手写数字识别任务，掌握了MNIST数据集的使用以及LeNet-5网络的构建和训练方法。这为后续更复杂的深度学习任务打下了坚实的基础。



代码附录

```
#%%
#导入必要的库
import torch
import torch.nn as nn
import torch.optim.optimizer
import torch.nn.functional as F
from torchvision import datasets, transforms
import numpy as np
import matplotlib.pyplot as plt
#%%
#定义超参数
batch_size = 64
test_batch = 1000
epochs = 10
lr = 0.01
momentum = 0.5
seed = 1
log_interval = 10
save_model = True
torch.manual_seed(seed)
#%%
#导入MNIST数据集
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('data', train=True, download=True,
transform=transforms.Compose([
```

```
        transforms.ToTensor(), transforms.Normalize((0.1307,),
(0.3081,))#对数据进行归一化处理
    ])),
    batch_size=batch_size, shuffle=True
)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('data', train=False, transform=transforms.Compose([
        transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))
    ])),
    batch_size=test_batch, shuffle=True
)
#%%
#定义网络结构LeNet-5
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 6, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(6, 16, 5),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.fc1 = nn.Sequential(
            nn.Linear(16*5*5, 120),
            nn.ReLU()
        )
        self.fc2 = nn.Sequential(
            nn.Linear(120, 84),
            nn.ReLU()
        )
        self.fc3 = nn.Linear(84, 10)


    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(-1, 16*5*5)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        return x
#%%
#定义模型、优化器、损失函数
model = LeNet()
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=momentum)
```

```
#%%
#定义训练函数
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)  # 使用交叉熵作为损失函数
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss:
{:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()
            ))
#%%
#定义测试函数
def test():
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            print(data.shape, target.shape)  # 添加此行以检查批次大小
            output = model(data)
            test_loss += F.cross_entropy(output, target,
reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{}
({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)
    ))

#%%
#训练模型
for epoch in range(1, epochs + 1):
    train(epoch)
    test()
#%%
#保存模型
if save_model:
    torch.save(model.state_dict(), 'mnist_cnn.pt')
#%%
#加载模型
model = LeNet()
```

```
model.load_state_dict(torch.load('mnist_cnn.pt'))
model.eval()
#%%
# 定义测试模型函数
def test_model():
    examples = enumerate(test_loader)
    batch_idx, (example_data, example_targets) = next(examples)
    with torch.no_grad():
        output = model(example_data)
        pred = output.argmax(dim=1, keepdim=True)

        # 打印前10个预测结果和对应的真实标签
        print("Predicted labels:", pred[:10].view(-1))
        print("Actual labels:", example_targets[:10])

        # 显示预测结果的图像
        fig = plt.figure(figsize=(10, 6))
        for i in range(6):
            plt.subplot(2, 3, i+1)
            plt.tight_layout()
            plt.imshow(example_data[i][0], cmap='gray',
interpolation='none')
            plt.title("Prediction: {}".format(pred[i].item()))
            plt.xticks([])
            plt.yticks([])
#%%
#测试模型
test_model()
#%%
# 计算模型的整体准确率
def calculate_accuracy():
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            output = model(data)
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
            total += target.size(0)
    accuracy = 100. * correct / total
    print('Accuracy: {}/{} ({:.0f}%)'.format(correct, total, accuracy))

#%%
# 计算并打印准确率
calculate_accuracy()
```