**暨南大学本科实验报告专用纸**

课程名称 深度学习实验 成绩评定

实验项目名称 构建CNN模型 指导教师 林聪

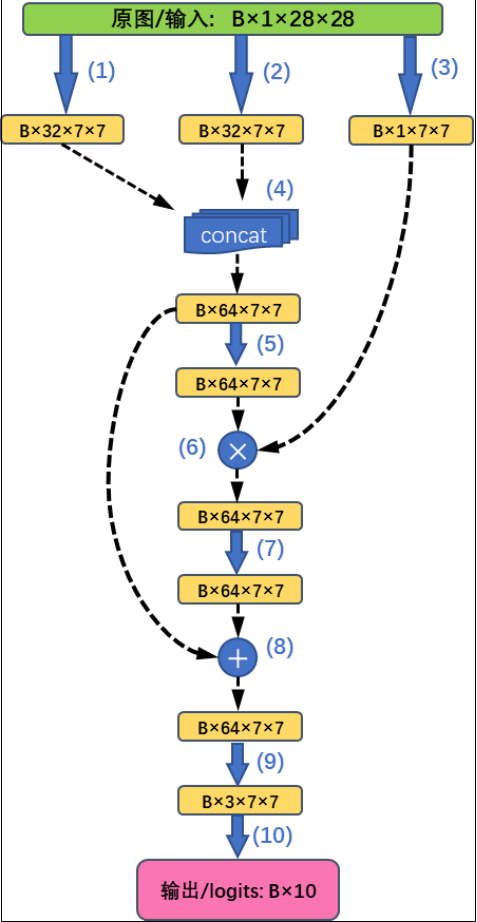
实验项目编号 02 实验项目类型 实验地点

学生姓名 赵俊文 学号 2022104002

学院 智能科学与工程学院 系 人工智能 专业 人工智能

1. **实验目的**

* 学习并整理“6.6.1 LeNet”小节的 notebook 的代码上下文
* 根据图 1 结构, 构建 DummyNet 模型, 替换 LeNet, 测试其性能



1. **主要仪器设备**

**仪器：** PC

**实验环境：** Windows11,Python1.10,Pytorch11.8

1. **源程序**

源程序在实验步骤与调试中给出。

1. **实验步骤与调试**
2. **学习整理“6.6.1 LeNet”的 notebook 的代码上下文**
3. 第一卷积层：6个5×5卷积核，填充2，输出6个特征图。
4. Sigmoid激活。
5. 平均池化：2×2，步长2。
6. 第二卷积层：16个5×5卷积核，无填充，输出16个特征图。
7. Sigmoid激活。
8. 平均池化：2×2，步长2。
9. 展平层。
10. 全连接层：输入400，输出120，接Sigmoid。
11. 全连接层：输入120，输出84，接Sigmoid。
12. ​​全连接层​​：输入84，输出10（分类输出）。
13. **根据图 1 结构, 构建 DummyNet 模型, 替换 LeNet, 测试其性能。**

构建DummyNet模型

import torch

from torch import nn

from d2l import torch as d2l

# 自定义 DummyNet 模型

class DummyNet(nn.Module):

    def \_\_init\_\_(self):

        super(DummyNet, self).\_\_init\_\_()

        # (1) 两组 Conv2d + LeakyReLU（步幅2，核3）

        self.branch1 = nn.Sequential(

            nn.Conv2d(1, 16, kernel\_size=3, stride=2, padding=1), nn.LeakyReLU(),

            nn.Conv2d(16, 32, kernel\_size=3, stride=2, padding=1), nn.LeakyReLU()

        )

        # (2) 一组 Conv2d + LeakyReLU（步幅4，核5）

        self.branch2 = nn.Sequential(

            nn.Conv2d(1, 32, kernel\_size=5, stride=4, padding=2), nn.LeakyReLU()

        )

        # (3) MaxPool + Sigmoid（步幅4，核4）

        self.branch3 = nn.Sequential(

            nn.MaxPool2d(kernel\_size=4, stride=4), nn.Sigmoid()

        )

        # (5) 偏卷积 Conv2d 核为(3,1)，padding=(1,0)

        self.conv5 = nn.Sequential(

            nn.Conv2d(65, 64, kernel\_size=(3, 1), padding=(1, 0)), nn.LeakyReLU()

        )

        # (6, 7) 偏卷积 Conv2d 核为(1,3)，padding=(0,1)

        self.conv6 = nn.Sequential(

            nn.Conv2d(64, 64, kernel\_size=(1, 3), padding=(0, 1)), nn.LeakyReLU()

        )

        # (9) 1x1 卷积

        self.conv9 = nn.Conv2d(64, 3, kernel\_size=1)

        # (10) 全连接层

        self.fc = nn.Linear(3 \* 7 \* 7, 10)

    def forward(self, x):

        out1 = self.branch1(x)  # Bx32x7x7

        out2 = self.branch2(x)  # Bx32x7x7

        out3 = self.branch3(x)  # Bx1x7x7

        # (4) concat -> Bx65x7x7

        out = torch.cat((out1, out2, out3), dim=1)

        # (5)

        out = self.conv5(out)

        # (6) 残差乘法

        residual = out.clone()

        out = self.conv6(out) \* residual

        # (7) 继续卷积

        out = self.conv6(out)

        # (8) 残差加法

        out = out + residual

        # (9)

        out = self.conv9(out)

        # (10)

        out = out.view(out.shape[0], -1)

        out = self.fc(out)

        return out

# 评估精度（使用 GPU）

def evaluate\_accuracy\_gpu(net, data\_iter, device=None):

    if isinstance(net, nn.Module):

        net.eval()

        if not device:

            device = next(iter(net.parameters())).device

    metric = d2l.Accumulator(2)

    with torch.no\_grad():

        for X, y in data\_iter:

            if isinstance(X, list):

                X = [x.to(device) for x in X]

            else:

                X = X.to(device)

            y = y.to(device)

            metric.add(d2l.accuracy(net(X), y), y.numel())

    return metric[0] / metric[1]

# 训练函数

def train\_ch6(net, train\_iter, test\_iter, num\_epochs, lr, device):

    def init\_weights(m):

        if type(m) in [nn.Linear, nn.Conv2d]:

            nn.init.xavier\_uniform\_(m.weight)

    net.apply(init\_weights)

    net.to(device)

    print("training on", device)

    optimizer = torch.optim.SGD(net.parameters(), lr=lr)

    loss = nn.CrossEntropyLoss()

    animator = d2l.Animator(xlabel='epoch', xlim=[1, num\_epochs],

                            legend=['train loss', 'train acc', 'test acc'])

    timer, num\_batches = d2l.Timer(), len(train\_iter)

    for epoch in range(num\_epochs):

        metric = d2l.Accumulator(3)

        net.train()

        for i, (X, y) in enumerate(train\_iter):

            timer.start()

            optimizer.zero\_grad()

            X, y = X.to(device), y.to(device)

            y\_hat = net(X)

            l = loss(y\_hat, y)

            l.backward()

            optimizer.step()

            with torch.no\_grad():

                metric.add(l \* X.shape[0], d2l.accuracy(y\_hat, y), X.shape[0])

            timer.stop()

            if (i + 1) % (num\_batches // 5) == 0 or i == num\_batches - 1:

                animator.add(epoch + (i + 1) / num\_batches,

                             (metric[0] / metric[2], metric[1] / metric[2], None))

        test\_acc = evaluate\_accuracy\_gpu(net, test\_iter, device)

        animator.add(epoch + 1, (None, None, test\_acc))

    print(f'loss {metric[0] / metric[2]:.3f}, train acc {metric[1] / metric[2]:.3f}, '

          f'test acc {test\_acc:.3f}')

    print(f'{metric[2] \* num\_epochs / timer.sum():.1f} examples/sec '

          f'on {str(device)}')

# 主函数

if \_\_name\_\_ == "\_\_main\_\_":

    batch\_size = 256

    train\_iter, test\_iter = d2l.load\_data\_fashion\_mnist(batch\_size=batch\_size)

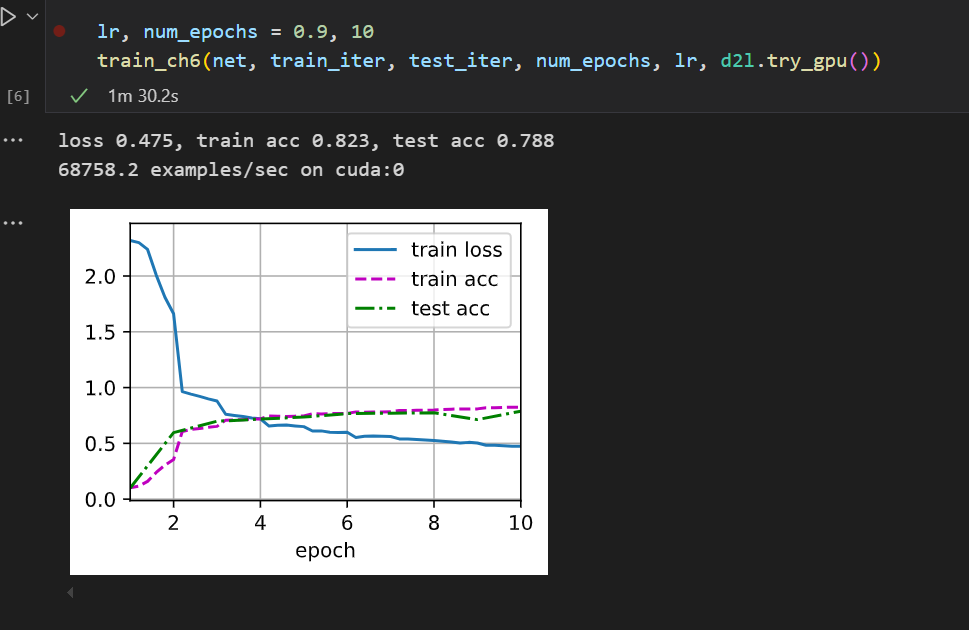
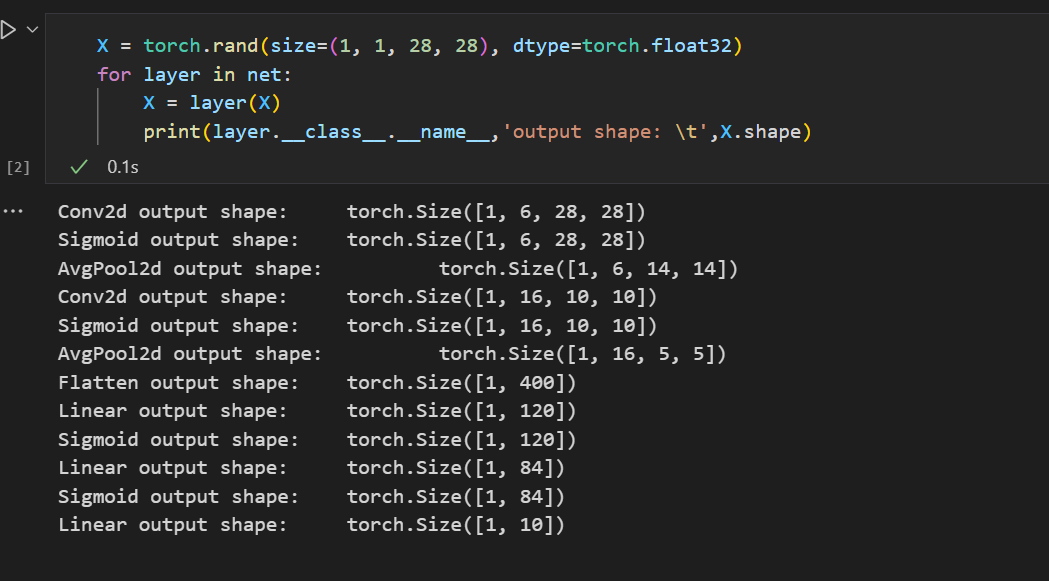
    device = d2l.try\_gpu()

    print(device.type)

    net = DummyNet()

    train\_ch6(net, train\_iter, test\_iter, num\_epochs=10, lr=0.1, device=device)

1. **实验结果与分析**
2. **Notebook代码执行结果**



1. **DummyNet模型替换替换 LeNet之后的性能**

