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

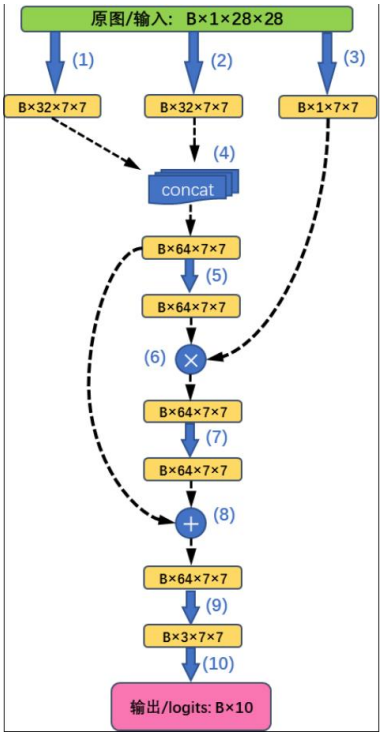
实验项目编号 02 实验项目类型 实验地点

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

（一） 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



（二） 主要仪器设备

仪器： PC

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

（三） 源程序

源程序在实验步骤与调试中给出。

（四） 实验步骤与调试

1. 学习整理“6.6.1 LeNet”的 notebook 的代码上下文

1) 第一卷积层：6 个 5×5 卷积核，填充 2，输出 6 个特征图。
2) Sigmoid 激活。
3) 平均池化：2×2，步长 2。
4) 第二卷积层：16 个 5×5 卷积核，无填充，输出 16 个特征图。
5) Sigmoid 激活。
6) 平均池化：2×2，步长 2。
7) 展平层。
8) 全连接层：输入 400，输出 120，接 Sigmoid。
9) 全连接层：输入 120，输出 84，接 Sigmoid。
10) 全连接层：输入 84，输出 10（分类输出）。

2. 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)
```

（五） 实验结果与分析

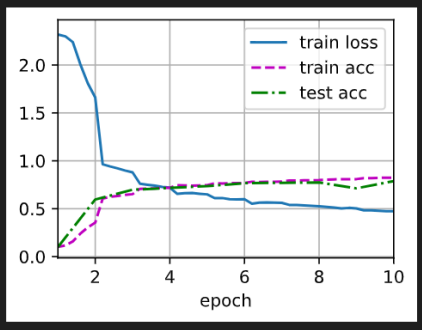
1. Notebook 代码执行结果

```
Conv2d output shape:     torch.Size([1, 6, 28, 28])
Sigmoid output shape:    torch.Size([1, 6, 28, 28])
AvgPool2d output shape:          torch.Size([1, 6, 14, 14])
Conv2d output shape:     torch.Size([1, 16, 10, 10])
Sigmoid output shape:    torch.Size([1, 16, 10, 10])
AvgPool2d output shape:          torch.Size([1, 16, 5, 5])
Flatten output shape:    torch.Size([1, 400])
Linear output shape:     torch.Size([1, 120])
Sigmoid output shape:    torch.Size([1, 120])
Linear output shape:     torch.Size([1, 84])
Sigmoid output shape:    torch.Size([1, 84])
Linear output shape:     torch.Size([1, 10])
```

```
lr, num_epochs = 0.9, 10
train_ch6(net, train_iter, test_iter, num_epochs, lr, d2l.try_gpu())
```

[6] ✓ 1m 30.2s

```
loss 0.475, train acc 0.823, test acc 0.788
68758.2 examples/sec on cuda:0
```



2. DummyNet 模型替换替换 LeNet 之后的性能