基于 Resnet 的图像分类实验报告

实验目标：

1、 参考 ResNet 论文[He et al., 2016a]中的表 1，以实现不同的变体

2、 在 Fashion-MNIST 数据集上训练你实现的 ResNet 变体，形成实验报告

实验内容：

ResNet 使用 4 个由残差块组成的模块，每个模块使用若干个同样输出通道数的残差块。由于每个模块有 4 个卷积层（不包括恒等映射的 1×1 卷积层）。加上第一个 7×7 卷积层和最后一个全连接层，共有 18 层。因此，这种模型通常被称为 ResNet-18。

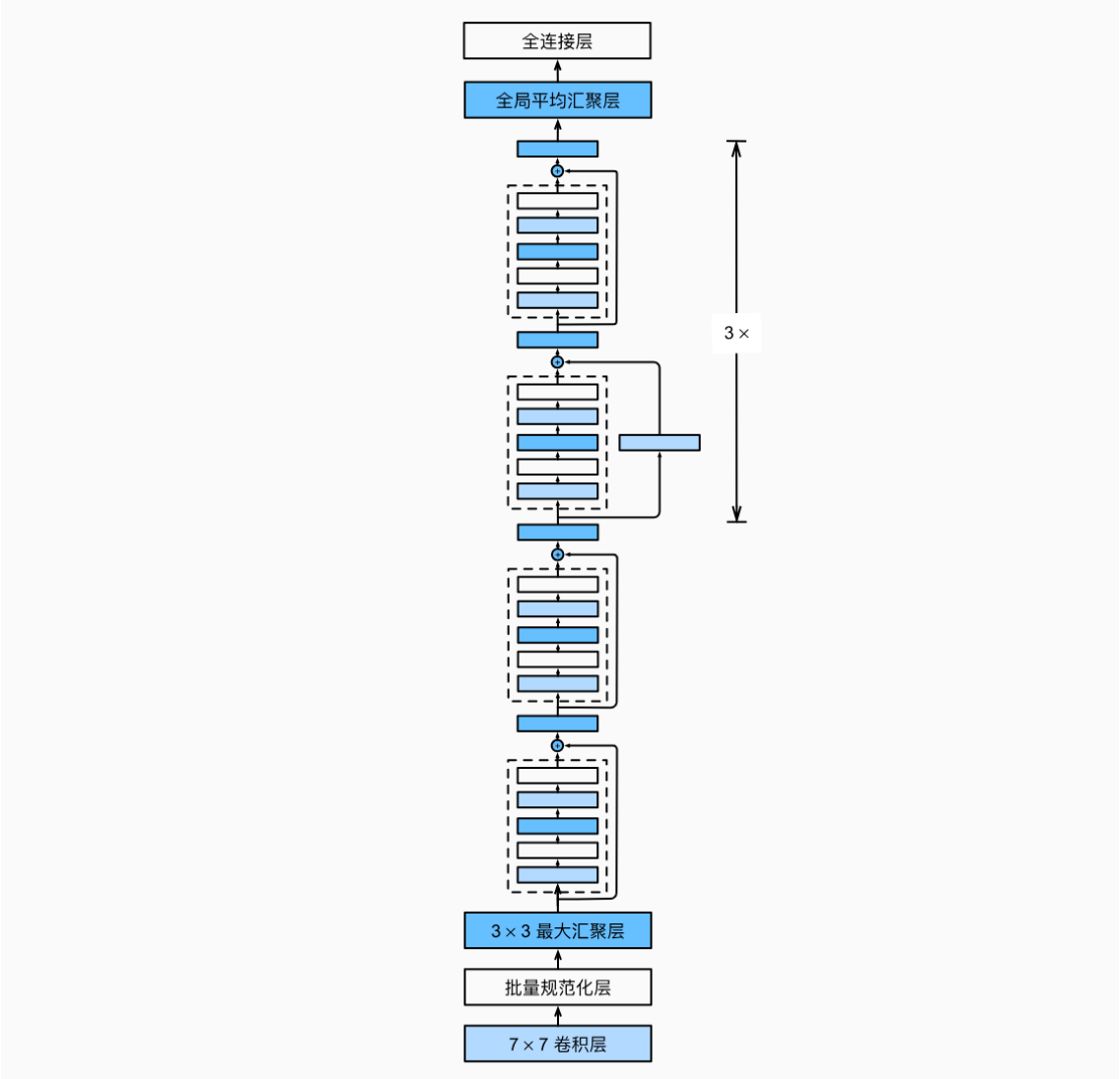

图 1 ResNet-18 组成架构

通过配置不同的通道数和模块里的残差块数可以得到不同的 ResNet 模型，例如更深的含 152 层的 ResNet-152。根据 ResNet 论文中的表 1，我们可以实现 ResNet 的不同变体，如 ResNet-18、ResNet-34、ResNet-50、ResNet-101 和 ResNet-152。

参考文献的变体架构如图 2 所示。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

图 2 各类 ResNet 变体的组成架构

定义残差块类，调用父类初始化方法后，设计第一个卷积层用于减少输入特征图的尺寸和通道数，设计第二个卷积层用于保持特征图尺寸不变，同时对于批归一化层进行设置。然后再设置前向传播函数 forward。

```python
# 参考ResNet论文 (He et al., 2016)中的表 1，以实现不同的变体

import torch
from torch import nn
from torch.nn import functional as F
from d2l import torch as d2l

# 定义残差块类
class Residual(nn.Module):
    def __init__(self, input_channels, num_channels, use_1x1conv=False, strides=1):
        super().__init__()  # 调用父类初始化方法
        # 第一个卷积层，用于减少输入特征图的尺寸和通道数（如果需要的话）
        self.conv1 = nn.Conv2d(input_channels, num_channels, kernel_size=3, padding=1, stride=strides)
        # 第二个卷积层，保持特征图尺寸不变
        self.conv2 = nn.Conv2d(num_channels, num_channels, kernel_size=3, padding=1)

        # 如果需要使用 1x1 卷积层来匹配输入和输出的通道数，则创建该层
        if use_1x1conv:
            self.conv3 = nn.Conv2d(input_channels, num_channels, kernel_size=1, stride=strides)
        else:
            self.conv3 = None

        # 批归一化层，分别用于两个卷积层后的输出
        self.bn1 = nn.BatchNorm2d(num_channels)
        self.bn2 = nn.BatchNorm2d(num_channels)

    def forward(self, X):
        # 输入经过第一个卷积层、批归一化层和 ReLU 激活函数
        Y = F.relu(self.bn1(self.conv1(X)))
        # 上一步结果再经过第二个卷积层和批归一化层
        Y = self.bn2(self.conv2(Y))

        # 如果使用了 1x1 卷积层，则对输入 X 进行相同操作
        if self.conv3:
            X = self.conv3(X)
        # 将两部分结果相加，并通过 ReLU 激活函数
        Y += X
        return F.relu(Y)
```

图 3 定义残差类

定义 ResNet 的基本块函数和 ResNet 的网络架构。

```
# 定义ResNet的基本块函数
def resnet_block(input_channels, num_channels, num_residuals, first_block=False):
    blk = []  # 初始化空列表用于存储残差块
    for i in range(num_residuals):
        # 对于每个残差块，根据是否为第一块以及是否需要下采样来决定参数
        if i == 0 and not first_block:
            blk.append(Residual(input_channels, num_channels, use_1x1conv=True, strides=2))
        else:
            blk.append(Residual(num_channels, num_channels))
    return blk  # 返回包含多个残差块的列表


# 定义ResNet网络架构
class ResNet(nn.Module):
    def __init__(self, num_classes, block_sizes):
        super().__init__()
        # 网络的第一层，包括一个 7x7 的卷积层，一个批量归一化层，一个 ReLU 激活函数和一个最大池化层
        self.b1 = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )

        # 定义四个残差块，每个残差块由多个残差单元组成
        self.b2 = nn.Sequential(*resnet_block(64, 64, block_sizes[0], first_block=True))  # 第一个残差块
        self.b3 = nn.Sequential(*resnet_block(64, 128, block_sizes[1]))  # 第二个残差块
        self.b4 = nn.Sequential(*resnet_block(128, 256, block_sizes[2]))  # 第三个残差块
        self.b5 = nn.Sequential(*resnet_block(256, 512, block_sizes[3]))  # 第四个残差块

        # 最后一层包括自适应平均池化层、展平层和全连接层
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(512, num_classes)

    def forward(self, X):
        # 前向传播过程，依次通过各个层
        X = self.b1(X)
        X = self.b2(X)
        X = self.b3(X)
        X = self.b4(X)
        X = self.b5(X)
        X = self.avgpool(X)
        X = self.flatten(X)
        X = self.fc(X)
        return X
```

图 4 基本块函数与网络架构

通过参考文献中对 ResNet 的不同变体，如 ResNet-18、ResNet-34、ResNet-50、ResNet-101 和 ResNet-152 不同残差块的设计数量，修改 ResNet 模型中的残差块数量，直接定义不用层数的 ResNet 模型即可。

```
# 定义不同层数的ResNet模型
def resnet18(num_classes):
    return ResNet(num_classes, [2, 2, 2, 2])  # ResNet18

def resnet34(num_classes):
    return ResNet(num_classes, [3, 4, 6, 3])  # ResNet34
```

图 5 定义 ResNet-18 和 ResNet-34

```python
def resnet50(num_classes):
    return ResNet(num_classes, [3, 4, 6, 3])  # ResNet50

def resnet101(num_classes):
    return ResNet(num_classes, [3, 4, 23, 3])  # ResNet101

def resnet152(num_classes):
    return ResNet(num_classes, [3, 8, 36, 3])  # ResNet152
```

图 6 定义 ResNet-50、ResNet-101 和 ResNet-152

对网络 net 选择 ResNet 模型，并对数据集路径进行设置、对数据进行预处理和加载器（此处主要因为无法直接使用，因此将数据集下载到本地进行运行），然后使用 0.1 的学习率和 10 轮 epoch 来训练网络。

```python
# 示例用法
num_classes = 10  # 输出类别数
net = resnet50(num_classes)  # 创建一个 ResNet50实例

# 数据集路径设置
data_dir = './data'
if not os.path.exists(data_dir):
    os.makedirs(data_dir)

# 数据预处理
transform = transforms.Compose([transforms.Resize(96), transforms.ToTensor()])
mnist_train = datasets.FashionMNIST(root=data_dir, train=True, transform=transform, download=True)
mnist_test = datasets.FashionMNIST(root=data_dir, train=False, transform=transform, download=True)

# 数据加载器
batch_size = 256  # 设置批次大小
train_iter = DataLoader(mnist_train, batch_size=batch_size, shuffle=True)
test_iter = DataLoader(mnist_test, batch_size=batch_size, shuffle=False)

# 训练网络
lr, num_epochs = 0.1, 10
d2l.train_ch6(net, train_iter, test_iter, num_epochs, lr, d2l.try_gpu())
```

图 7 模型选择、数据处理和网络训练

使用本机 GPU 对不同变体 ResNet-18、ResNet-34、ResNet-50、ResNet-101 和 ResNet-152 进行训练得到以下结果。

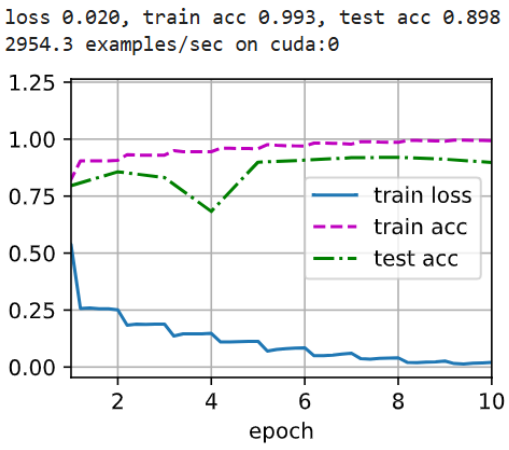

图 8 ResNet-18 训练结果

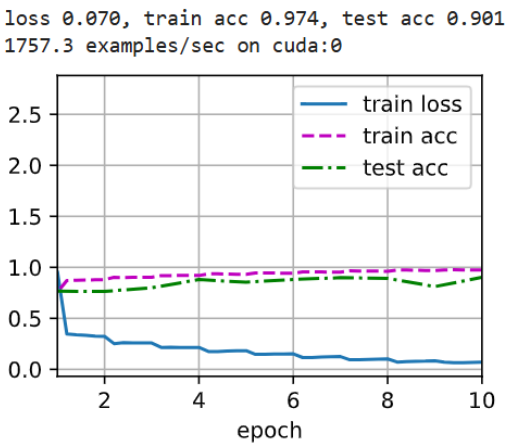

图 9 ResNet-34 训练结果

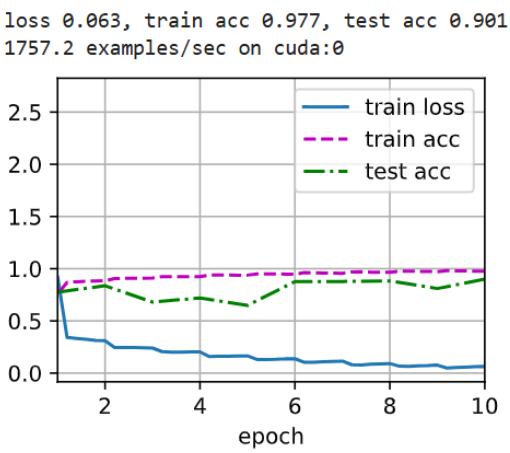

图 10 ResNet-50 训练结果

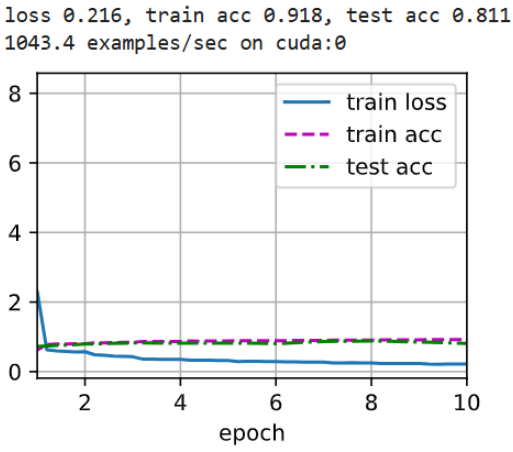

图 11 ResNet-101 训练结果

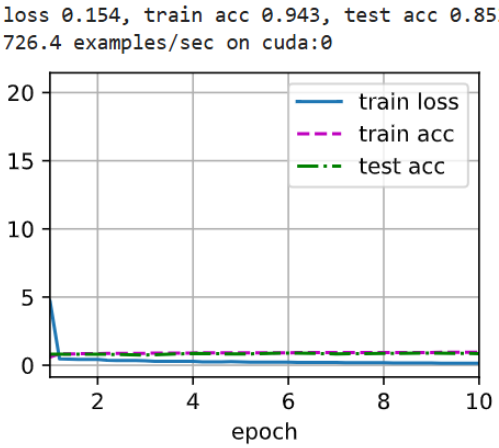

图 12 ResNet-152 训练结果