Python程序设计——第七次实验报告

2311095 宋卓伦 计算机科学与技术 2024.10.25

一、实验目的

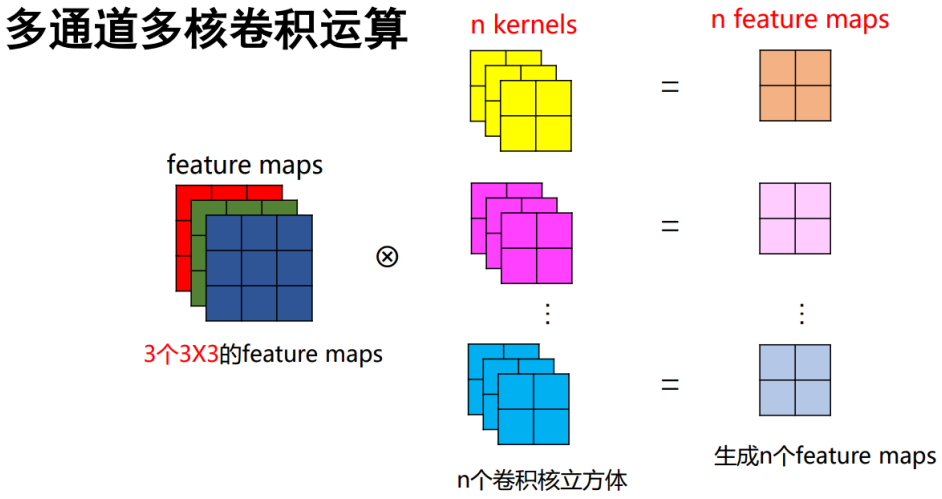
安装Python工具包，基于Python工具包编写程序，掌握Python工具包的安装和使用方法，理解Python工具包在解决实际问题中的作用。

二、实验描述

修改下面程序，实现如下图所示的多通道多卷积核的卷积运算

（指定输入数据、卷积核形状及卷积核数量后，随机生成运算数据）。

```python
#原始代码.py
import numpy as np  #导入numpy工具包

F=np.array(# 创建ndarray类对象F(对应一个3*3矩阵，待做卷积计算的二维数据)
    [[1, 2,-1],
     [-2,-3, -4],
     [3, 4,5]]
)

C =np.array(# 创建ndarray类对象C(对应一个2*2矩阵，卷积核)
    [[0.3, 0.1],
     [0.2, 0.4]]
)

R =np.zeros((2,2))
# 创建2行*2列、所有元素值都为零的ndarray类对象
for row in range(2):
    # 依次获取切片操作的起始行索引
    for col in range(2):
        #依次获取切片操作的起始列索引

        R[row,col]= np.sum(F[row:row+2,col:col+2]*C)

        # 先对F切片操作得到2行*2列元素并将切片结果与C做哈达玛积
        # 再调用np.sum函数计算哈达玛积结果矩阵的元素之和
        # 最后将求得的和保存到结果矩阵相应位置

print("卷积结果:\n",R)

```



三、实验代码

```python
#实验代码.py
#导入相关的包
import numpy as np
from numpy import random

#随机生成一个p*p的矩阵，每个元素值在0-1之间
def given_value(p):
    Fs = random.random(size=(p, p))
    return Fs

#通道数
channel_num = int(input("请输入通道数："))
#feature map的个数
numbers = int(input("请输入 feature map 的个数："))
#每个feature map的尺寸
fm_size = int(input("请输入 feature map 每个的大小："))
#每个kernel的尺寸
kernel_size = int(input("请输入 kernel 每个的大小："))

#生成矩阵的尺寸计算公式（这里步长为1，简化了计算过程）
tmp = fm_size - kernel_size + 1

#储存feature_maps矩阵
feature_maps = []
for index in range(numbers):
    feature_maps.append(given_value(fm_size))

#储存kernels矩阵
kernels = []
for index in range(channel_num):
    kernels.append([])
    for index0 in range(numbers):
        kernels[index].append(given_value(kernel_size))

#储存最终results矩阵
results = []
for index in range(channel_num):
    results.append([])
    for index0 in range(numbers):
```

```
        results[index].append(np.zeros((tmp, tmp)))

#计算多通道卷积，首先提取数组中的矩阵元素
for i in range(channel_num):
    for j in range(numbers):

        #提取
        F = feature_maps[j]
        G = kernels[i][j]
        R = results[i][j]

        #正式计算多通道卷积
        for row in range(tmp):
            for col in range(tmp):
                R[row, col] =
                np.sum(F[row:row + kernel_size, col:col + kernel_size] * G)

#打印计算结果
for i in range(channel_num):
    R = sum(results[i])
    print('卷积', i + 1, ': \n', R)

```

```
"C:\Program Files\Python312\python.exe"
请输入通道数：3
请输入 feature map 的个数：3
请输入 feature map 每个的大小：3
请输入 kernel 每个的大小：2
卷积 1 ：
 [[3.49001661 3.77692827]
 [3.14991584 2.26693036]]
卷积 2 ：
 [[3.64056554 3.97555746]
 [3.25361907 2.33783569]]
卷积 3 ：
 [[2.99454047 2.80562123]
 [2.55259055 1.97752759]]

进程已结束，退出代码为 0
```

四、实验反思

- 时间复杂度相当高

  其中后半段存在四个循环，最大复杂度近似为channel_num的四次方，即$o(n^4)$；

- 代码长度过大

有很多重复的地方，比如为各矩阵数组赋值等处；

- 可以导入torch

  pytorch专门用于机器学习、深度学习等领域，里面含有大量处理卷积的函数可以尝试使用。

五、实验改进

下面的程序我们导入了pytorch里面的库进行计算：

```
import torch
import torch.nn as nn
import numpy as np

# 设置随机种子以确保结果的可重复性
torch.manual_seed(0)
np.random.seed(0)

# 获取输入参数
channel_num = int(input("请输入通道数："))
numbers = int(input("请输入 feature map 的个数："))
fm_size = int(input("请输入 feature map 每个的大小："))
kernel_size = int(input("请输入 kernel 每个的大小："))

# 初始化feature maps和kernels
feature_maps = [torch.rand(1, fm_size, fm_size) for _ in
                range(numbers)]  # 每个feature map的形状为[1, fm_size, fm_size]

kernels = [torch.rand(1, kernel_size, kernel_size) for _ in
           range(channel_num)]  # 每个kernel的形状为[1, kernel_size,
kernel_size]

# 使用PyTorch的Conv2d模块进行卷积操作
conv_layers = nn.ModuleList(
    [nn.Conv2d(1, 1, kernel_size=(kernel_size, kernel_size), padding=
(kernel_size - 1) // 2) for _ in
     range(channel_num)])

# 应用卷积
results = []
for i in range(channel_num):
    # 更新卷积层的权重为当前的kernel
    conv_layers[i].weight.data = kernels[i].view(1, 1, kernel_size,
kernel_size)
    # 应用卷积到每个feature map
    result = []
    for fm in feature_maps:
        fm = fm.view(1, 1, fm_size, fm_size)  # 添加batch和channel维度
        conv_result = conv_layers[i](fm)  # 应用卷积
        result.append(conv_result.squeeze().detach().numpy())  # 移除batch和
channel维度并转换为numpy
    results.append(result)

# 打印结果
for i in range(numbers):
    print('原矩阵', i + 1, '：\n', feature_maps[i].squeeze().numpy())

```

```
for i in range(channel_num):
    print('卷积', i + 1, ': \n', np.array(results[i]).sum(axis=0))  # 将所有feature map的结果相加

```

```
"C:\Program Files\Python312\python.exe"
请输入通道数：3
请输入 feature map 的个数：3
请输入 feature map 每个的大小：3
请输入 kernel 每个的大小：2
卷积 1 ：
 [[-0.42883754 -0.25691563]
 [-0.0117801   0.44036308]]
卷积 2 ：
 [[1.2030253 1.352327 ]
 [1.1940564 1.377857 ]]
卷积 3 ：
 [[2.160815  2.6061153]
 [3.298646  4.198612 ]]

进程已结束，退出代码为 0
```

在PyTorch中，nn.ModuleList 是一个容器，用于包含子模块（nn.Module）的列表。与普通的Python列表不同，nn.ModuleList 会被正确地注册到网络中，使得所有的子模块都能被正确地管理和访问。这对于构建复杂的神经网络结构，特别是那些包含多个相同或相似组件的结构时，非常有用。

对于Conv2d：

```
class torch.nn.Conv2d(in_channels, out_channels, kernel_size,
                      stride=1, padding=0, dilation=1, groups=1, bias=True)
```

可以看出，这个类将步长默认为1，没有填充。同时定义了一个卷积核，方便后面进行二维卷积操作。

最后没有解决的小问题：为什么在打印矩阵的过程中会出现一些空格?