gooleNet ResNet

笔记本:	pytorch		
创建时间:	2023/4/2 10:12	**更新时间:**	2023/4/2 10:29
作者:	Kaiyuecui		
URL:	about:blank		

1、这里的网络不是对照论文中的

只是为了自己理解，所编写的一个简单的代码

2、GooleNet

①提出了Inception,因为在网络学习中，卷积核的大小是一个参数，我们应该使用适当大小的卷积核，此时就有了Inception,Inception的思想是，对于输入x，我们可以同时使用不同的卷积来进行操作，最后将不同卷积得到的结果来进行一个拼接cat操作（是在通道维度上进行拼接），我们应该保证通过不同卷积后得到的宽和高相同，但是通道数可以不同。

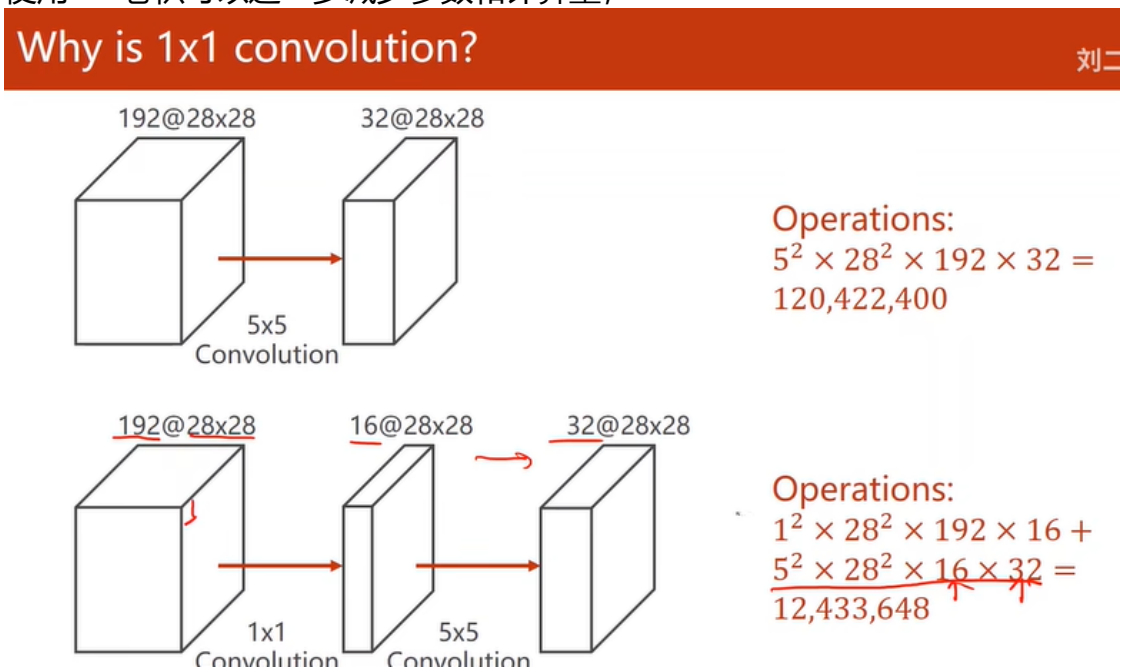
Inception：由于使用了不同卷积，对于不同路上的权重也不同。如果某个卷积的效果好，则含有该卷积的路上的权重就会大，反之亦然。

②1*1卷积。

卷积有局部相关，权值共享的优点。

使用卷积操作，在全连接的基础上进行对比，已经减少了我们的计算量

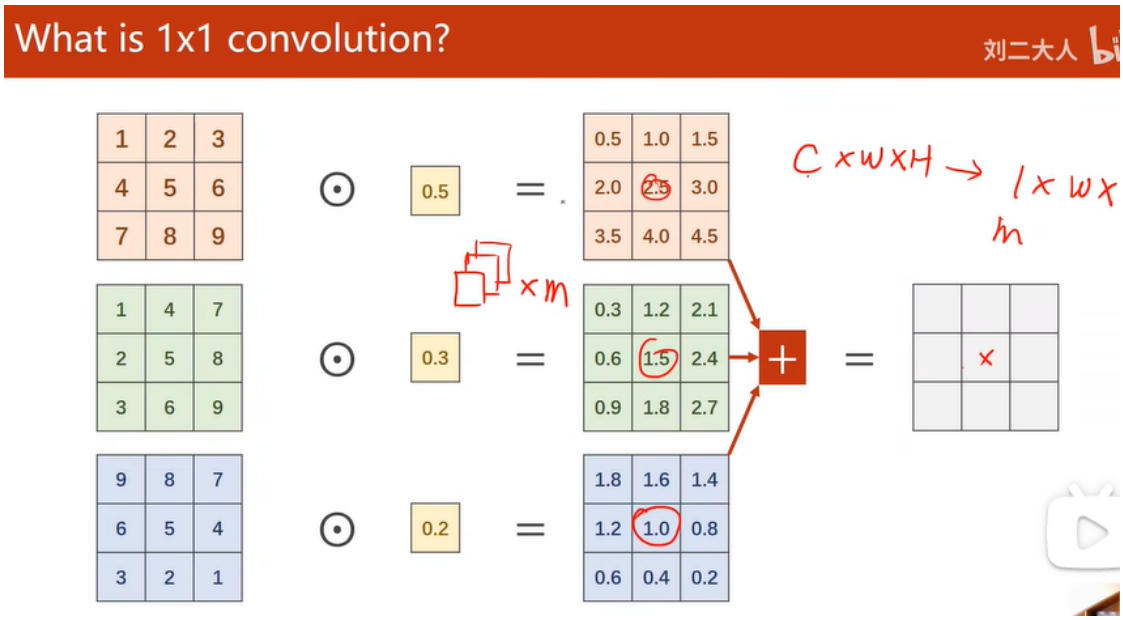
使用1*1卷积可以进一步减少参数和计算量；



1*1卷积可以用来进行升维和降维的作用。

1*1卷积之后得到的结果，并不会改变图像的长和宽，之后改变通道。

同时有图像融合的作用，1*1卷积之后得到的图像，对应像素点的值时是原始图像不同通道像素值融合的结果。



③代码

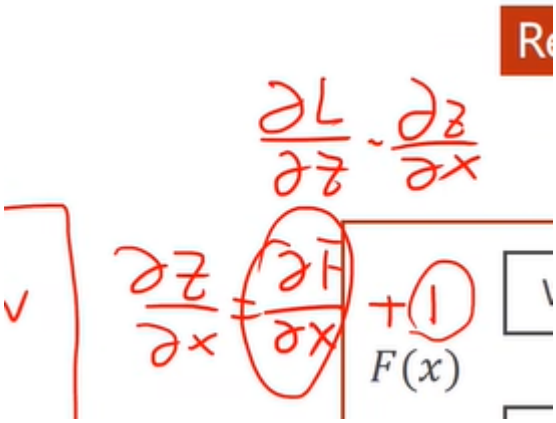
```
# 时间：2023/3/31 21:57
# cky
from torch.utils.data import DataLoader
from torchvision import transforms
from torchvision import datasets
#parpare datasets
batch_size=64
transform=transforms.Compose([
    transforms.ToTensor(),#将pil图像转化为tensor类型 whc-》cwh
    transforms.Normalize((0.1307,),(0.3801,)) #归一化，平均值，方差
])
train_data=datasets.MNIST(root='../dataset/minst',train=True,download=True,transform=transform)
test_data=datasets.MNIST(root='../dataset/minst',train=False,download=True,transform=transform)
test_len=len(test_data)  #保存一下我们的测试数据集样本数，在测试时我们需要用它来求准确
率
train_load=DataLoader(train_data,batch_size=batch_size,shuffle=True)
test_load=DataLoader(test_data,batch_size=batch_size,shuffle=False)
#design model using class
import  numpy as np
import torch.nn as nn
import torch
import torch.nn.functional as F
class Goolenet(nn.Module):
    def __init__(self):
        super(Goolenet, self).__init__()
        self.conv1=nn.Conv2d(1,10,kernel_size=5)
        self.conv2=nn.Conv2d(88,20,kernel_size=5)
        self.incp1=Inception(channel=10)
        self.incp2=Inception(channel=20)
        self.mp=nn.MaxPool2d(2)
        self.fc=nn.Linear(1408,10)
        self.relu=nn.ReLU()
    def forward(self,x):
        batch=x.size(0)
        x=self.conv1(x)
        x=self.relu(self.mp(x))
        x=self.incp1(x)
        x=self.relu(self.mp(self.conv2(x)))
        x=self.incp2(x)
        x=x.view(batch,-1)
        x=self.fc(x)
        return x


class Inception(nn.Module):
    def __init__(self,channel):
```

```
        super(Inception, self).__init__()
        self.pool=nn.Conv2d(channel,24,kernel_size=1)
        self.con1=nn.Conv2d(channel,16,kernel_size=1)
        self.con5=nn.Conv2d(16,24,kernel_size=5,padding=2)
        self.con3_1=nn.Conv2d(16,24,kernel_size=3,padding=1)
        self.con3_3=nn.Conv2d(24,24,kernel_size=3,padding=1)
        self.relu=nn.ReLU()
    def forward(self,x):
        branch_pool=F.avg_pool2d(x,kernel_size=3,stride=1,padding=1)
        branch_pool=self.pool(branch_pool)
        branch_1=self.con1(x)
        branch_5=self.con1(x)
        branch_5=self.con5(branch_5)
        branch_3=self.con1(x)
        branch_3=self.con3_1(branch_3)
        branch_3=self.con3_3(branch_3)


        outputs=[branch_pool,branch_1,branch_5,branch_3]
        return torch.cat(outputs,dim=1)
#construct loss and optim
model=Goolenet()
device=torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)
print("device:",device)
loss=nn.CrossEntropyLoss()
optim=torch.optim.SGD(params=model.parameters(),lr=0.1,momentum=0.5)


#training cycle and test
#将训练和测试分别定义为一个函数
def train(epoch):
    loss_sum=0
    for i,data in enumerate(train_load,1):
        inputs,outputs=data
        inputs,outputs=inputs.to(device),outputs.to(device)
        y_pre=model(inputs)
        cost=loss(y_pre,outputs)
        optim.zero_grad()
        cost.backward()
        optim.step()
        loss_sum+=cost.item()
        if i%300==0:  #不需要每次迭代都输出，在这里我们让每300次迭代输出一次
            print('epoch:',epoch,'i:',i,'loss:',loss_sum/300)
def test():  #测试函数
    correct=0
    with torch.no_grad():  #在测试中不需要计算梯度，这样可以不用保留一些为了计算梯度的
值，可以节省内存空间
        for data in test_load:
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            y_pre=model(images)
            _,predicts=torch.max(y_pre,dim=1)  #找出维度1 即一行中的哪一列值最大，返
回最大值和相对应的索引
            correct+=(predicts==labels).sum().item()
    print('Accuracy on test data: %d %%' % (100*(correct/test_len)))
if __name__ == '__main__':
    for i in range(10):
        train(i)
        test()
```

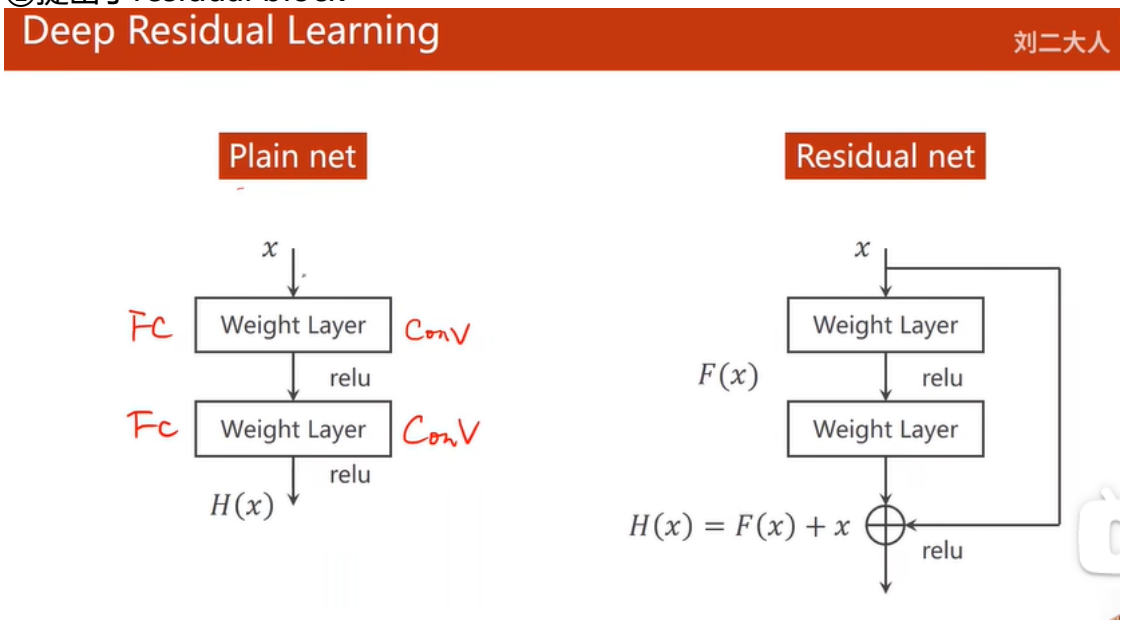
3、Resnet

对于我们普通的网络，如果网络层数越多，则我们模型的性能也会下降。
一般来说，如果网络层数深，我们预期的是网络的后几层几乎不起作用，起到恒等学习的作用，但是实际情况并非如此，由于层数过多，那么求梯度时，就

很容易出现梯度消失的情况，导致我们的模型在一开始就不能学习到很好的参数，因此导致模型的性能下降。
使用了残差快，就会很好的抑制我们梯度消失的情况。
原因：



加上x则对x求导时就会加上一个1.
①提出了residual block



残差网络相加时应该注意，我们应该保证在两条线路上到的维度以及图像的长和宽都应该相同。
②代码：

```
# 时间：2023/4/2 9:58
# cky
import numpy as np
import matplotlib.pyplot as plt
import torch.nn
from torch.utils.data import DataLoader
from torchvision import transforms
from torchvision import datasets
import torch.nn as nn


#parpare datasets
batch_size=64
transform=transforms.Compose([
    transforms.ToTensor(),#将pil图像转化为tensor类型 whc-》cwh
    transforms.Normalize((0.1307,),(0.3801,)) #归一化，平均值，方差
])
train_data=datasets.MNIST(root='../dataset/minst',train=True,download=True,transform=transform)
test_data=datasets.MNIST(root='../dataset/minst',train=False,download=True,transform=transform)
```

```
test_len=len(test_data)  #保存一下我们的测试数据集样本数，在测试时我们需要用它来求准确率
train_load=DataLoader(train_data,batch_size=batch_size,shuffle=True)
test_load=DataLoader(test_data,batch_size=batch_size,shuffle=False)
#prepare datasets
#deaign model
class ResNet(nn.Module):
    def __init__(self):
        super(ResNet, self).__init__()
        self.conv1=nn.Conv2d(1,16,kernel_size=5)
        self.conv2=nn.Conv2d(16,32,kernel_size=5)
        self.relu=nn.ReLU()
        self.linear=nn.Linear(512,10)
        self.r1=rblock(16)
        self.r2=rblock(32)
        self.mp=nn.MaxPool2d(2)
    def forward(self,x):
        batch=x.size(0)
        x=self.mp(self.relu(self.conv1(x)))
        x=self.r1(x)
        x=x=self.mp(self.relu(self.conv2(x)))
        x=self.r2(x)
        x=x.view(batch,-1)
        #print(x.size(1))
        x=self.linear(x)
        return x
class rblock(nn.Module):
    def __init__(self,channel):
        super(rblock, self).__init__()
        self.con1=nn.Conv2d(channel,channel,kernel_size=3,padding=1)
        self.con2=nn.Conv2d(channel,channel,kernel_size=3,padding=1)
        self.relu=nn.ReLU()
    def forward(self,x):
        y=self.relu(self.con1(x))
        y=self.con2(y)
        return x+y
model=ResNet()
device=torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)
print("device:",device)
loss=nn.CrossEntropyLoss()
optim=torch.optim.SGD(params=model.parameters(),lr=0.1,momentum=0.5)


#training cycle and test
#将训练和测试分别定义为一个函数
def train(epoch):
    loss_sum=0
    for i,data in enumerate(train_load,1):
        inputs,outputs=data
        inputs,outputs=inputs.to(device),outputs.to(device)
        y_pre=model(inputs)
        cost=loss(y_pre,outputs)
        optim.zero_grad()
        cost.backward()
        optim.step()
        loss_sum+=cost.item()
        if i%300==0:  #不需要每次迭代都输出，在这里我们让每300次迭代输出一次
            print('epoch:',epoch,'i:',i,'loss:',loss_sum/300)
def test():  #测试函数
    correct=0
    with torch.no_grad():  #在测试中不需要计算梯度，这样可以不用保留一些为了计算梯度的值，可以节省内存空间
        for data in test_load:
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            y_pre=model(images)
            _,predicts=torch.max(y_pre,dim=1)  #找出维度1 即一行中的哪一列值最大，返回最大值和相对应的索引
            correct+=(predicts==labels).sum().item()
```

```
    print('Accuracy on test data: %d %%' % (100*(correct/test_len)))
if __name__ == '__main__':
    for i in range(10):
        train(i)
        test()
```