PyTorch 深度学习实践 第10讲

第10讲  卷积神经网络(基础篇) 源代码

B站 刘二大人 ，传送门PyTorch深度学习实践——卷积神经网络(基础篇)

视频中截图：

说明 1、每一个卷积核它的通道数量要求和输入通道是一样的。这种卷积核的总数有多少个和你输出通道的数量是一样的。

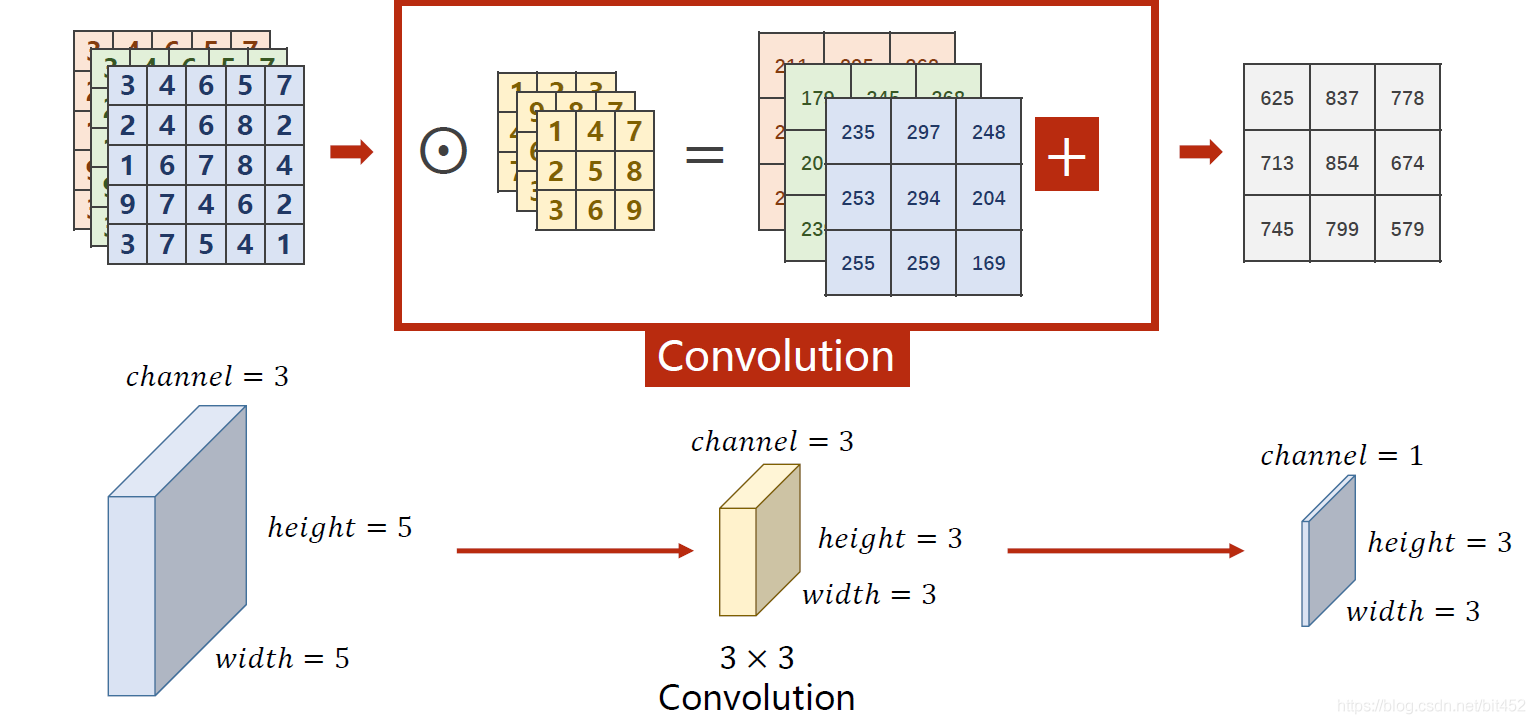
2、卷积(convolution)后，C(Channels)变，W(width)和H(Height)可变可不变，取决于是否padding。subsampling(或pooling)后，C不变，W和H变。

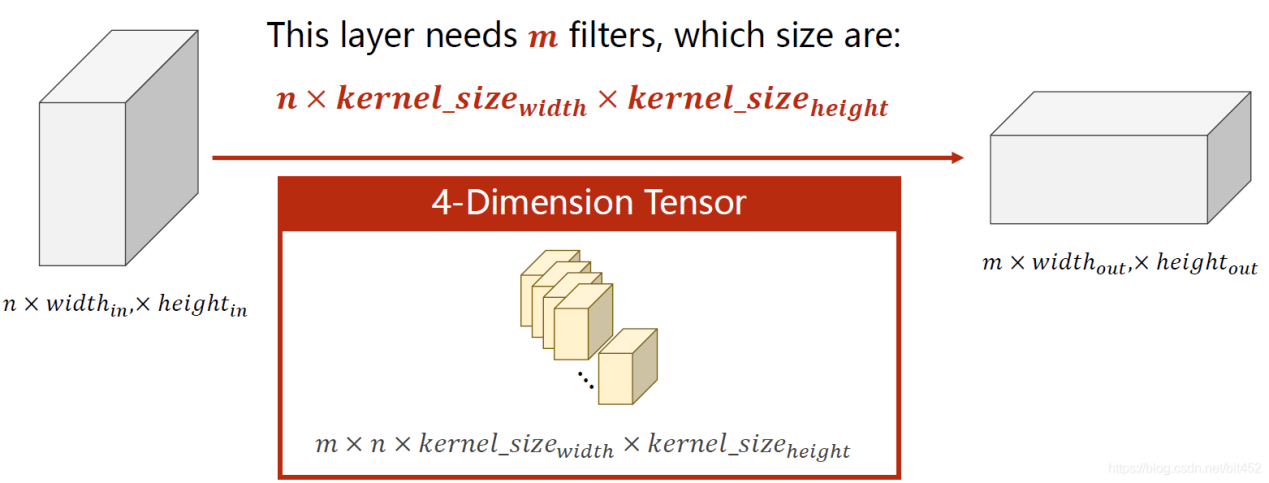
3、卷积层：保留图像的空间信息。

4、卷积层要求输入输出是四维张量(B,C,W,H)，全连接层的输入与输出都是二维张量(B,Input\_feature)。

传送门 PyTorch的nn.Linear（）详解

5、卷积(线性变换)，激活函数(非线性变换)，池化；这个过程若干次后，view打平，进入全连接层~





代码说明：

1、torch.nn.Conv2d(1,10,kernel\_size=3,stride=2,bias=False)

      1是指输入的Channel，灰色图像是1维的；10是指输出的Channel，也可以说第一个卷积层需要10个卷积核；kernel\_size=3,卷积核大小是3x3；stride=2进行卷积运算时的步长，默认为1；bias=False卷积运算是否需要偏置bias，默认为False。padding = 0，卷积操作是否补0。

2、self.fc = torch.nn.Linear(320, 10)，这个320获取的方式，可以通过x = x.view(batch\_size, -1) # print(x.shape)可得到(64,320),64指的是batch，320就是指要进行全连接操作时，输入的特征维度。

|  |
| --- |
| import torch  from torchvision import transforms  from torchvision import datasets  from torch.utils.data import DataLoader  import torch.nn.functional as F  import torch.optim as optim    # prepare dataset    batch\_size = 64  transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])    train\_dataset = datasets.MNIST(root='../dataset/mnist/', train=True, download=True, transform=transform)  train\_loader = DataLoader(train\_dataset, shuffle=True, batch\_size=batch\_size)  test\_dataset = datasets.MNIST(root='../dataset/mnist/', train=False, download=True, transform=transform)  test\_loader = DataLoader(test\_dataset, shuffle=False, batch\_size=batch\_size)    # design model using class      class Net(torch.nn.Module):  def \_\_init\_\_(self):  super(Net, self).\_\_init\_\_()  self.conv1 = torch.nn.Conv2d(1, 10, kernel\_size=5)  self.conv2 = torch.nn.Conv2d(10, 20, kernel\_size=5)  self.pooling = torch.nn.MaxPool2d(2)  self.fc = torch.nn.Linear(320, 10)      def forward(self, x):  # flatten data from (n,1,28,28) to (n, 784)  batch\_size = x.size(0)  x = F.relu(self.pooling(self.conv1(x)))  x = F.relu(self.pooling(self.conv2(x)))  x = x.view(batch\_size, -1) # -1 此处自动算出的是320  x = self.fc(x)    return x      model = Net()    # construct loss and optimizer  criterion = torch.nn.CrossEntropyLoss()  optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)    # training cycle forward, backward, update      def train(epoch):  running\_loss = 0.0  for batch\_idx, data in enumerate(train\_loader, 0):  inputs, target = data  optimizer.zero\_grad()    outputs = model(inputs)  loss = criterion(outputs, target)  loss.backward()  optimizer.step()    running\_loss += loss.item()  if batch\_idx % 300 == 299:  print('[%d, %5d] loss: %.3f' % (epoch+1, batch\_idx+1, running\_loss/300))  running\_loss = 0.0      def test():  correct = 0  total = 0  with torch.no\_grad():  for data in test\_loader:  images, labels = data  outputs = model(images)  \_, predicted = torch.max(outputs.data, dim=1)  total += labels.size(0)  correct += (predicted == labels).sum().item()  print('accuracy on test set: %d %% ' % (100\*correct/total))      if \_\_name\_\_ == '\_\_main\_\_':  for epoch in range(10):  train(epoch)  test() |

GPU版本

|  |
| --- |
| import torch  from torchvision import transforms  from torchvision import datasets  from torch.utils.data import DataLoader  import torch.nn.functional as F  import torch.optim as optim  import matplotlib.pyplot as plt    # prepare dataset    batch\_size = 64  transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])    train\_dataset = datasets.MNIST(root='../dataset/mnist/', train=True, download=True, transform=transform)  train\_loader = DataLoader(train\_dataset, shuffle=True, batch\_size=batch\_size)  test\_dataset = datasets.MNIST(root='../dataset/mnist/', train=False, download=True, transform=transform)  test\_loader = DataLoader(test\_dataset, shuffle=False, batch\_size=batch\_size)    # design model using class      class Net(torch.nn.Module):  def \_\_init\_\_(self):  super(Net, self).\_\_init\_\_()  self.conv1 = torch.nn.Conv2d(1, 10, kernel\_size=5)  self.conv2 = torch.nn.Conv2d(10, 20, kernel\_size=5)  self.pooling = torch.nn.MaxPool2d(2)  self.fc = torch.nn.Linear(320, 10)      def forward(self, x):  # flatten data from (n,1,28,28) to (n, 784)    batch\_size = x.size(0)  x = F.relu(self.pooling(self.conv1(x)))  x = F.relu(self.pooling(self.conv2(x)))  x = x.view(batch\_size, -1) # -1 此处自动算出的是320  # print("x.shape",x.shape)  x = self.fc(x)    return x      model = Net()  device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  model.to(device)    # construct loss and optimizer  criterion = torch.nn.CrossEntropyLoss()  optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)    # training cycle forward, backward, update      def train(epoch):  running\_loss = 0.0  for batch\_idx, data in enumerate(train\_loader, 0):  inputs, target = data  inputs, target = inputs.to(device), target.to(device)  optimizer.zero\_grad()    outputs = model(inputs)  loss = criterion(outputs, target)  loss.backward()  optimizer.step()    running\_loss += loss.item()  if batch\_idx % 300 == 299:  print('[%d, %5d] loss: %.3f' % (epoch+1, batch\_idx+1, running\_loss/300))  running\_loss = 0.0      def test():  correct = 0  total = 0  with torch.no\_grad():  for data in test\_loader:  images, labels = data  images, labels = images.to(device), labels.to(device)  outputs = model(images)  \_, predicted = torch.max(outputs.data, dim=1)  total += labels.size(0)  correct += (predicted == labels).sum().item()  print('accuracy on test set: %d %% ' % (100\*correct/total))  return correct/total      if \_\_name\_\_ == '\_\_main\_\_':  epoch\_list = []  acc\_list = []    for epoch in range(10):  train(epoch)  acc = test()  epoch\_list.append(epoch)  acc\_list.append(acc)    plt.plot(epoch\_list,acc\_list)  plt.ylabel('accuracy')  plt.xlabel('epoch')  plt.show() |

————————————————

版权声明：本文为CSDN博主「错错莫」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/bit452/article/details/109690712