PyTorch 深度学习实践 第11讲

第11讲  卷积神经网络(高级篇) 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(高级篇)

视频中截图：

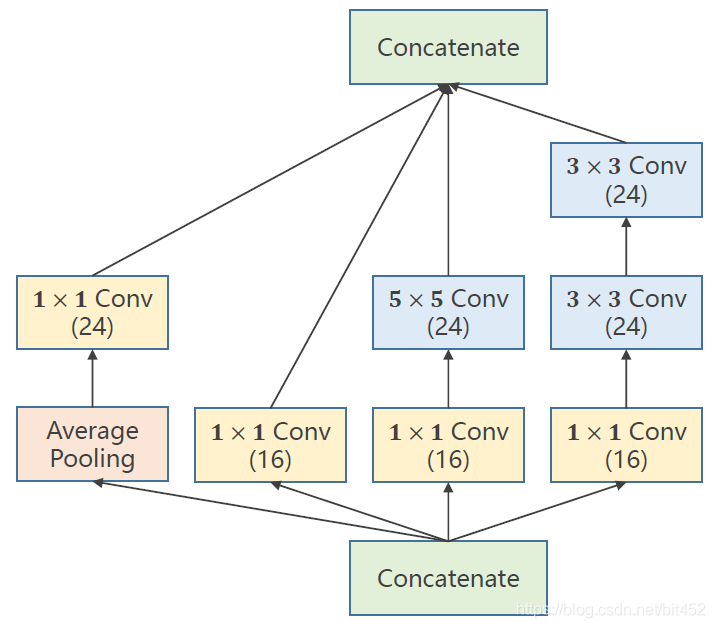
说明：Inception Moudel

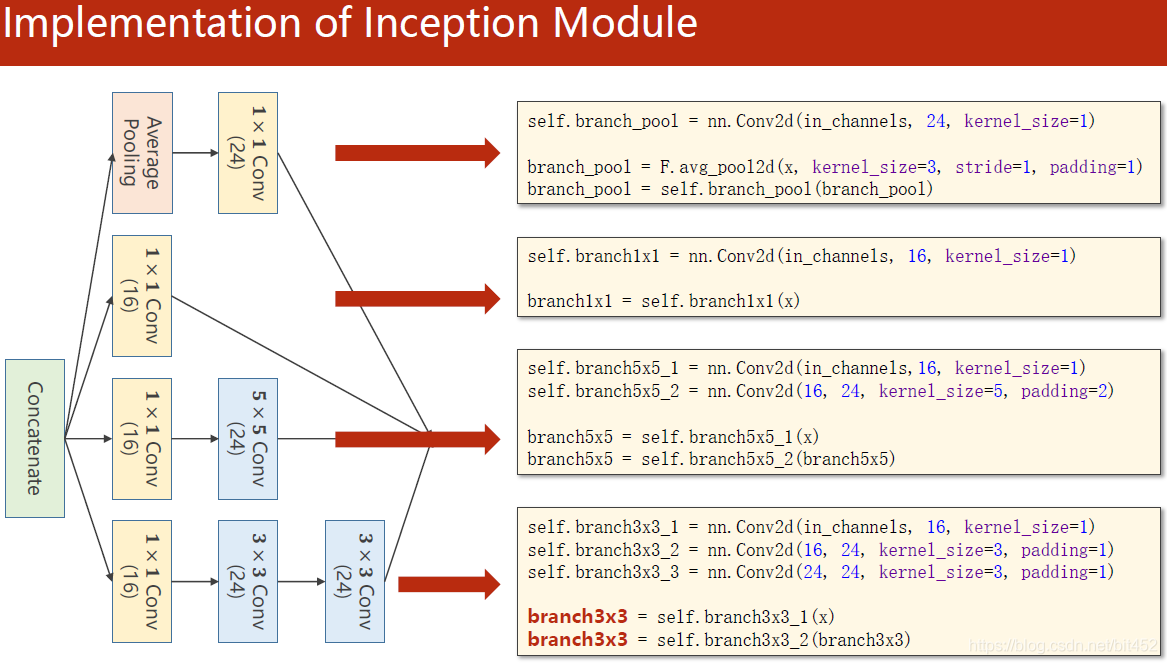
1、卷积核超参数选择困难，自动找到卷积的最佳组合。

2、1x1卷积核，不同通道的信息融合。使用1x1卷积核虽然参数量增加了，但是能够显著的降低计算量(operations)

3、Inception Moudel由4个分支组成，要分清哪些是在Init里定义，哪些是在forward里调用。4个分支在dim=1(channels)上进行concatenate。24+16+24+24 = 88

4、GoogleNet的Inception(Pytorch实现)





代码说明：1、先使用类对Inception Moudel进行封装

2、先是1个卷积层(conv,maxpooling,relu)，然后inceptionA模块(输出的channels是24+16+24+24=88)，接下来又是一个卷积层(conv,mp,relu),然后inceptionA模块，最后一个全连接层(fc)。

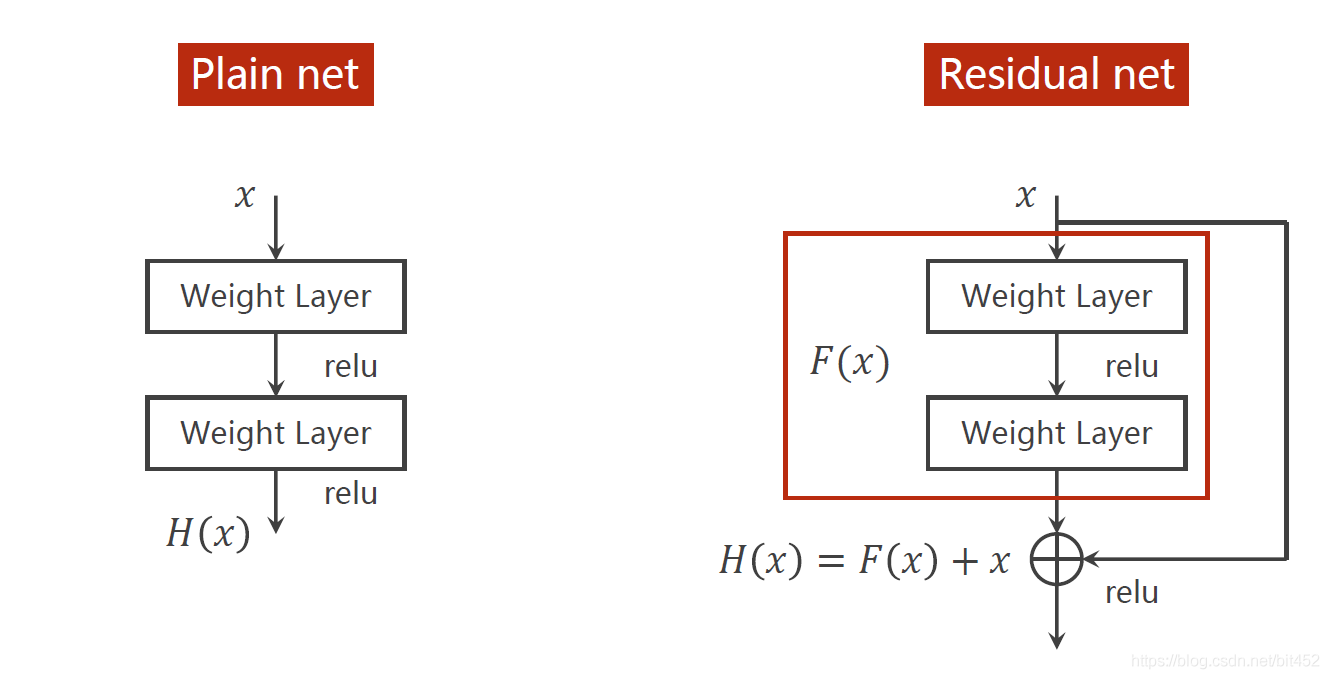
3、1408这个数据可以通过x = x.view(in\_size, -1)后调用x.shape得到。

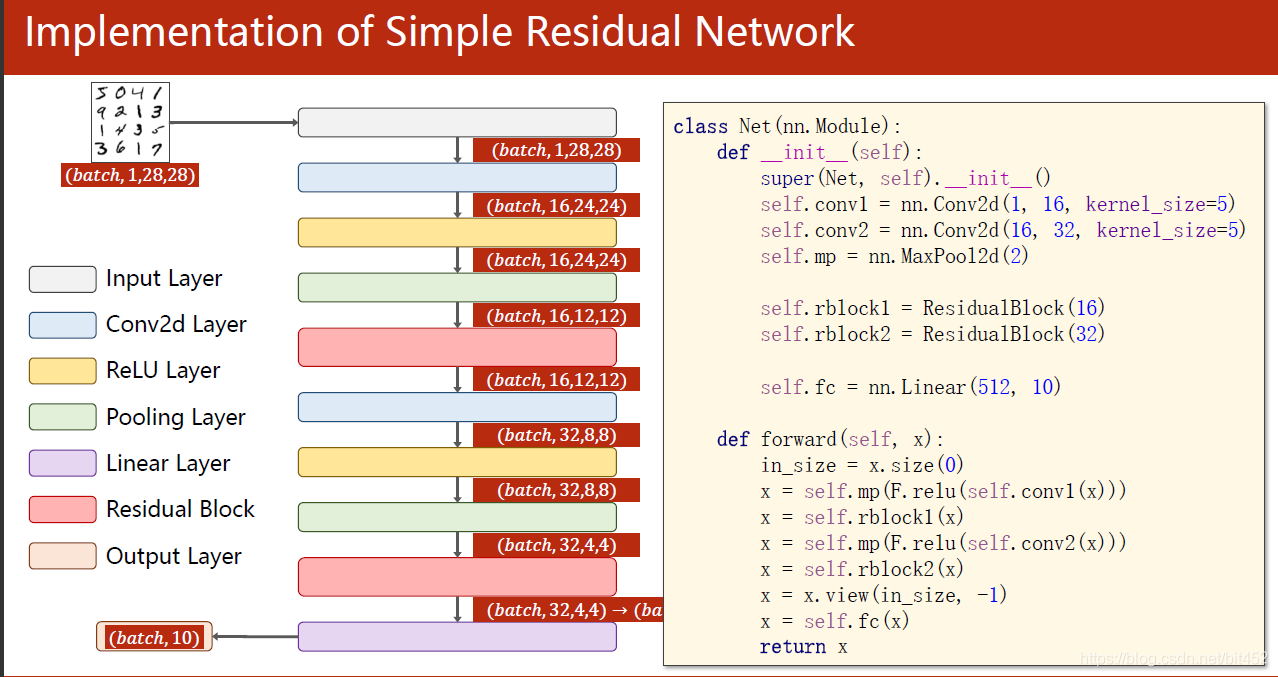
|  |
| --- |
| import torch  import torch.nn as nn  from torchvision import transforms  from torchvision import datasets  from torch.utils.data import DataLoader  import torch.nn.functional as F  import torch.optim as optim    # prepare dataset    batch\_size = 64  transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))]) # 归一化,均值和方差    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 InceptionA(nn.Module):  def \_\_init\_\_(self, in\_channels):  super(InceptionA, self).\_\_init\_\_()  self.branch1x1 = nn.Conv2d(in\_channels, 16, kernel\_size=1)    self.branch5x5\_1 = nn.Conv2d(in\_channels, 16, kernel\_size=1)  self.branch5x5\_2 = nn.Conv2d(16, 24, kernel\_size=5, padding=2)    self.branch3x3\_1 = nn.Conv2d(in\_channels, 16, kernel\_size=1)  self.branch3x3\_2 = nn.Conv2d(16, 24, kernel\_size=3, padding=1)  self.branch3x3\_3 = nn.Conv2d(24, 24, kernel\_size=3, padding=1)    self.branch\_pool = nn.Conv2d(in\_channels, 24, kernel\_size=1)    def forward(self, x):  branch1x1 = self.branch1x1(x)    branch5x5 = self.branch5x5\_1(x)  branch5x5 = self.branch5x5\_2(branch5x5)    branch3x3 = self.branch3x3\_1(x)  branch3x3 = self.branch3x3\_2(branch3x3)  branch3x3 = self.branch3x3\_3(branch3x3)    branch\_pool = F.avg\_pool2d(x, kernel\_size=3, stride=1, padding=1)  branch\_pool = self.branch\_pool(branch\_pool)    outputs = [branch1x1, branch5x5, branch3x3, branch\_pool]  return torch.cat(outputs, dim=1) # b,c,w,h c对应的是dim=1    class Net(nn.Module):  def \_\_init\_\_(self):  super(Net, self).\_\_init\_\_()  self.conv1 = nn.Conv2d(1, 10, kernel\_size=5)  self.conv2 = nn.Conv2d(88, 20, kernel\_size=5) # 88 = 24x3 + 16    self.incep1 = InceptionA(in\_channels=10) # 与conv1 中的10对应  self.incep2 = InceptionA(in\_channels=20) # 与conv2 中的20对应    self.mp = nn.MaxPool2d(2)  self.fc = nn.Linear(1408, 10)      def forward(self, x):  in\_size = x.size(0)  x = F.relu(self.mp(self.conv1(x)))  x = self.incep1(x)  x = F.relu(self.mp(self.conv2(x)))  x = self.incep2(x)  x = x.view(in\_size, -1)  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() |

视频中截图：

说明：1、要解决的问题：梯度消失

2、跳连接，H(x) = F(x) + x,张量维度必须一样，加完后再激活。不要做pooling，张量的维度会发生变化。





代码说明：

1、先是1个卷积层(conv,maxpooling,relu)，然后ResidualBlock模块，接下来又是一个卷积层(conv,mp,relu),然后esidualBlock模块模块，最后一个全连接层(fc)。

|  |
| --- |
| import torch  import torch.nn as nn  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,))]) # 归一化,均值和方差    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 ResidualBlock(nn.Module):  def \_\_init\_\_(self, channels):  super(ResidualBlock, self).\_\_init\_\_()  self.channels = channels  self.conv1 = nn.Conv2d(channels, channels, kernel\_size=3, padding=1)  self.conv2 = nn.Conv2d(channels, channels, kernel\_size=3, padding=1)    def forward(self, x):  y = F.relu(self.conv1(x))  y = self.conv2(y)  return F.relu(x + y)    class Net(nn.Module):  def \_\_init\_\_(self):  super(Net, self).\_\_init\_\_()  self.conv1 = nn.Conv2d(1, 16, kernel\_size=5)  self.conv2 = nn.Conv2d(16, 32, kernel\_size=5) # 88 = 24x3 + 16    self.rblock1 = ResidualBlock(16)  self.rblock2 = ResidualBlock(32)    self.mp = nn.MaxPool2d(2)  self.fc = nn.Linear(512, 10) # 暂时不知道1408咋能自动出来的      def forward(self, x):  in\_size = x.size(0)    x = self.mp(F.relu(self.conv1(x)))  x = self.rblock1(x)  x = self.mp(F.relu(self.conv2(x)))  x = self.rblock2(x)    x = x.view(in\_size, -1)  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() |

————————————————

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/109693790