首先导入torch.nn，pytorch的网络模块多在此内，然后导入model\_zoo，作用是根据下面的model\_urls里的地址加载网络预训练权重。后面还对conv2d进行了一次封装，个人觉得有些多余。

|  |
| --- |
| import torch.nn as nn  import torch.utils.model\_zoo as model\_zoo  \_\_all\_\_ = ['ResNet', 'resnet18', 'resnet34', 'resnet50', 'resnet101',             'resnet152']  model\_urls = {      'resnet18': 'https://download.pytorch.org/models/resnet18-5c106cde.pth',      'resnet34': 'https://download.pytorch.org/models/resnet34-333f7ec4.pth',      'resnet50': 'https://download.pytorch.org/models/resnet50-19c8e357.pth',      'resnet101': 'https://download.pytorch.org/models/resnet101-5d3b4d8f.pth',      'resnet152': 'https://download.pytorch.org/models/resnet152-b121ed2d.pth',  }  def conv3x3(in\_planes, out\_planes, stride=1):      """3x3 convolution with padding"""      return nn.Conv2d(in\_planes, out\_planes, kernel\_size=3, stride=stride,                       padding=1, bias=False) |

这里定义了最重要的残差模块，这个是基础版，由两个叠加的3x3卷积组成，与之相对应的bottleneck模块在下面定义

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | class BasicBlock(nn.Module):      expansion = 1        def \_\_init\_\_(self, inplanes, planes, stride=1, downsample=None):          super(BasicBlock, self).\_\_init\_\_()          self.conv1 = conv3x3(inplanes, planes, stride)          self.bn1 = nn.BatchNorm2d(planes)          self.relu = nn.ReLU(inplace=True)          self.conv2 = conv3x3(planes, planes)          self.bn2 = nn.BatchNorm2d(planes)          self.downsample = downsample          self.stride = stride        def forward(self, x):          residual = x            out = self.conv1(x)          out = self.bn1(out)          out = self.relu(out)            out = self.conv2(out)          out = self.bn2(out)            if self.downsample is not None:              residual = self.downsample(x)            out += residual          out = self.relu(out)            return out |

　　与基础版的不同之处只在于这里是三个卷积，分别是1x1,3x3,1x1,分别用来压缩维度，卷积处理，恢复维度，inplane是输入的通道数，plane是输出的通道数，expansion是对输出通道数的倍乘，在basic中expansion是1，此时完全忽略expansion这个东东，输出的通道数就是plane，然而bottleneck就是不走寻常路，它的任务就是要对通道数进行压缩，再放大，于是，plane不再代表输出的通道数，而是block内部压缩后的通道数，输出通道数变为plane\*expansion。接着就是网络主体了。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | class Bottleneck(nn.Module):      expansion = 4        def \_\_init\_\_(self, inplanes, planes, stride=1, downsample=None):          super(Bottleneck, self).\_\_init\_\_()          self.conv1 = nn.Conv2d(inplanes, planes, kernel\_size=1, bias=False)          self.bn1 = nn.BatchNorm2d(planes)          self.conv2 = nn.Conv2d(planes, planes, kernel\_size=3, stride=stride,                                 padding=1, bias=False)          self.bn2 = nn.BatchNorm2d(planes)          self.conv3 = nn.Conv2d(planes, planes \* self.expansion, kernel\_size=1, bias=False)          self.bn3 = nn.BatchNorm2d(planes \* self.expansion)          self.relu = nn.ReLU(inplace=True)          self.downsample = downsample          self.stride = stride        def forward(self, x):          residual = x            out = self.conv1(x)          out = self.bn1(out)          out = self.relu(out)            out = self.conv2(out)          out = self.bn2(out)          out = self.relu(out)            out = self.conv3(out)          out = self.bn3(out)            if self.downsample is not None:              residual = self.downsample(x)            out += residual          out = self.relu(out)            return out |

　　resnet共有五个阶段，其中第一阶段为一个7x7的卷积处理，stride为2，然后经过池化处理，此时特征图的尺寸已成为输入的1/4，接下来是四个阶段，也就是代码中的layer1,layer2,layer3,layer4。这里用make\_layer函数产生四个layer，需要用户输入每个layer的block数目（即layers列表)以及采用的block类型（基础版还是bottleneck版）

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | class ResNet(nn.Module):        def \_\_init\_\_(self, block, layers, num\_classes=1000):          self.inplanes = 64          super(ResNet, self).\_\_init\_\_()          self.conv1 = nn.Conv2d(3, 64, kernel\_size=7, stride=2, padding=3,                                 bias=False)          self.bn1 = nn.BatchNorm2d(64)          self.relu = nn.ReLU(inplace=True)          self.maxpool = nn.MaxPool2d(kernel\_size=3, stride=2, padding=1)          self.layer1 = self.\_make\_layer(block, 64, layers[0])          self.layer2 = self.\_make\_layer(block, 128, layers[1], stride=2)          self.layer3 = self.\_make\_layer(block, 256, layers[2], stride=2)          self.layer4 = self.\_make\_layer(block, 512, layers[3], stride=2)          self.avgpool = nn.AvgPool2d(7, stride=1)          self.fc = nn.Linear(512 \* block.expansion, num\_classes)            for m in self.modules():              if isinstance(m, nn.Conv2d):                  nn.init.kaiming\_normal\_(m.weight, mode='fan\_out', nonlinearity='relu')              elif isinstance(m, nn.BatchNorm2d):                  nn.init.constant\_(m.weight, 1)                  nn.init.constant\_(m.bias, 0)        def \_make\_layer(self, block, planes, blocks, stride=1):          downsample = None          if stride != 1 or self.inplanes != planes \* block.expansion:              downsample = nn.Sequential(                  nn.Conv2d(self.inplanes, planes \* block.expansion,                            kernel\_size=1, stride=stride, bias=False),                  nn.BatchNorm2d(planes \* block.expansion),              )            layers = []          layers.append(block(self.inplanes, planes, stride, downsample))          self.inplanes = planes \* block.expansion          for i in range(1, blocks):              layers.append(block(self.inplanes, planes))            return nn.Sequential(\*layers)        def forward(self, x):          x = self.conv1(x)          x = self.bn1(x)          x = self.relu(x)          x = self.maxpool(x)            x = self.layer1(x)          x = self.layer2(x)          x = self.layer3(x)          x = self.layer4(x)            x = self.avgpool(x)          x = x.view(x.size(0), -1)          x = self.fc(x)            return x |

　　接下来就是resnet18等几个模型的类定义

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  3 | def resnet18(pretrained=False, \*\*kwargs):      """Constructs a ResNet-18 model.      Args:          pretrained (bool): If True, returns a model pre-trained on ImageNet      """      model = ResNet(BasicBlock, [2, 2, 2, 2], \*\*kwargs)      if pretrained:          model.load\_state\_dict(model\_zoo.load\_url(model\_urls['resnet18']))      return model      def resnet34(pretrained=False, \*\*kwargs):      """Constructs a ResNet-34 model.      Args:          pretrained (bool): If True, returns a model pre-trained on ImageNet      """      model = ResNet(BasicBlock, [3, 4, 6, 3], \*\*kwargs)      if pretrained:          model.load\_state\_dict(model\_zoo.load\_url(model\_urls['resnet34']))      return model      def resnet50(pretrained=False, \*\*kwargs):      """Constructs a ResNet-50 model.      Args:          pretrained (bool): If True, returns a model pre-trained on ImageNet      """      model = ResNet(Bottleneck, [3, 4, 6, 3], \*\*kwargs)      if pretrained:          model.load\_state\_dict(model\_zoo.load\_url(model\_urls['resnet50']))      return model      def resnet101(pretrained=False, \*\*kwargs):      """Constructs a ResNet-101 model.      Args:          pretrained (bool): If True, returns a model pre-trained on ImageNet      """      model = ResNet(Bottleneck, [3, 4, 23, 3], \*\*kwargs)      if pretrained:          model.load\_state\_dict(model\_zoo.load\_url(model\_urls['resnet101']))      return model      def resnet152(pretrained=False, \*\*kwargs):      """Constructs a ResNet-152 model.      Args:          pretrained (bool): If True, returns a model pre-trained on ImageNet      """      model = ResNet(Bottleneck, [3, 8, 36, 3], \*\*kwargs)      if pretrained:          model.load\_state\_dict(model\_zoo.load\_url(model\_urls['resnet152']))      return model |

　　这里比较简单，就是调用上面ResNet对象，输入block类型和block数目，这里可以看到resnet18和resnet34用的是基础版block，因为此时网络还不深，不太需要考虑模型的效率，而当网络加深到52，101，152层时则有必要引入bottleneck结构，方便模型的存储和计算。另外是否加载预训练权重是可选的，具体就是调用model\_zoo加载指定链接地址的序列化文件，反序列化为权重文件。

 最后，不妨看一下resnet18和resnet50的网络结构，主要是为了看一下basic和bottleneck的区别。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | ResNet(    (conv1): Conv2d(3, 64, kernel\_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)    (relu): ReLU(inplace)    (maxpool): MaxPool2d(kernel\_size=3, stride=2, padding=1, dilation=1, ceil\_mode=False)    (layer1): Sequential(      (0): BasicBlock(        (conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (relu): ReLU(inplace)        (conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)      )      (1): BasicBlock(        (conv1): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (relu): ReLU(inplace)        (conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)      )    )    (layer2): Sequential(      (0): BasicBlock(        (conv1): Conv2d(64, 128, kernel\_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (relu): ReLU(inplace)        (conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (downsample): Sequential(          (0): Conv2d(64, 128, kernel\_size=(1, 1), stride=(2, 2), bias=False)          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        )      )      (1): BasicBlock(        (conv1): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (relu): ReLU(inplace)        (conv2): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)      )    ) |

　　这是resnet18,只贴出了前两层，其他层类似，第一层是没有downsample的，因为输入与输出通道数一样，其余层都有downsample。

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | ResNet(    (conv1): Conv2d(3, 64, kernel\_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)    (relu): ReLU(inplace)    (maxpool): MaxPool2d(kernel\_size=3, stride=2, padding=1, dilation=1, ceil\_mode=False)    (layer1): Sequential(      (0): Bottleneck(        (conv1): Conv2d(64, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (conv3): Conv2d(64, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (relu): ReLU(inplace)        (downsample): Sequential(          (0): Conv2d(64, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        )      )      (1): Bottleneck(        (conv1): Conv2d(256, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (conv3): Conv2d(64, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (relu): ReLU(inplace)      )      (2): Bottleneck(        (conv1): Conv2d(256, 64, kernel\_size=(1, 1), stride=(1, 1), bias=False)        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (conv2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (conv3): Conv2d(64, 256, kernel\_size=(1, 1), stride=(1, 1), bias=False)        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)        (relu): ReLU(inplace)      )    ) |  |  |

　　这是resnet50，只贴出了第一层，每一层都有downsample，因为输出与输入通道数都不一样。可以看在resnet类中输入的64，128，256，512，都不是最终的输出通道数，只是block内部压缩的通道数，实际输出通道数要乘以expansion，此处为4。