CNN

> 2013551 雷贺奥

原始版cnn

网络结构

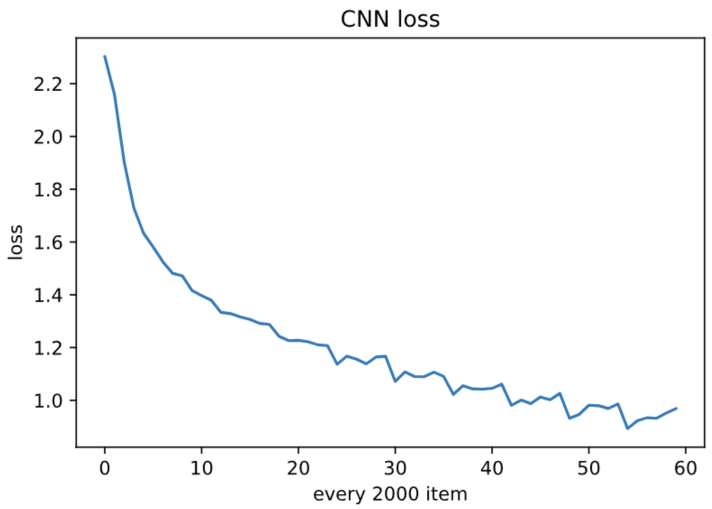
```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)   # kernel_size=5, padding=2, stride=1
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.conv3=nn.Conv2d(16,16,1)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.short_cut1=nn.Conv2d(3, 6, 19)

    def forward(self, x):
        res1=x
        x = self.conv1(x)
        x = F.relu(x)
        x = self.pool(x)
        x=self.conv2(x)
        x = F.relu(x)
        x = self.pool(x)
        x=self.conv3(x)
        x = F.relu(x)
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
```

```
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

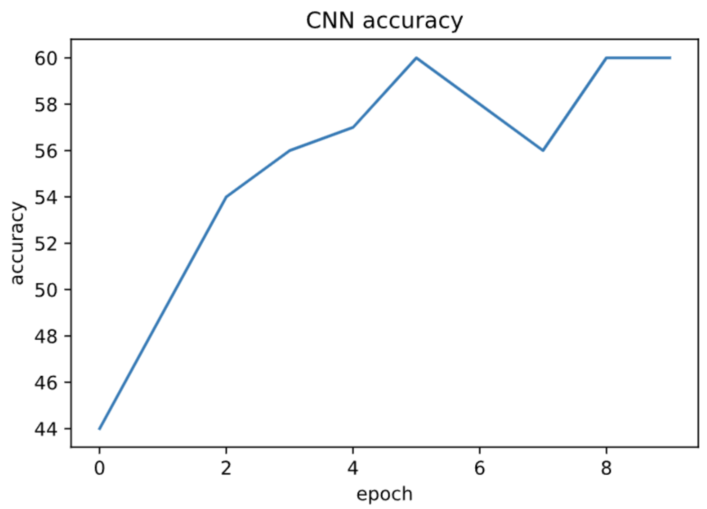
使用**torchsummary**打印网络结构，其功能特别强大，各层网络均给出shape，如下所示:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             456
         MaxPool2d-2            [-1, 6, 14, 14]               0
            Conv2d-3           [-1, 16, 10, 10]           2,416
         MaxPool2d-4             [-1, 16, 5, 5]               0
            Conv2d-5             [-1, 16, 5, 5]             272
            Linear-6                  [-1, 120]          48,120
            Linear-7                   [-1, 84]          10,164
            Linear-8                   [-1, 10]             850
================================================================
```

训练loss曲线



准确度曲线图



准确度最终稳定在60%，可见简单CNN的局限性。

Resnet

作者在此处复现的是Resnet18，因为若网络层数过少，最终的准确率与CNN相差无几，因此选择18层的Resnet。

网络结构

先定义残差连接 Reslink，用于连接不同网络层数，以防止因为网络过深，而导致的过拟合等问题。

再定义Resnet18，为了方便起见，直接把各个block中卷积层的参数写定，如下所示：

```
class Reslink(nn.Module):
    def __init__(self,inchannel,outchannel,mykernel_size,mystride,mypadding):
        super().__init__()

 self.link=nn.Conv2d(in_channels=inchannel,out_channels=outchannel,kernel_size=my
kernel_size,stride=mystride,padding=mypadding)
    def forward(self, x):
        x=self.link(x)
        return x

class ResNet(nn.Module):
    def __init__(self,inchannel,outchannel,stride=1):
        super().__init__()
        self.conv1=nn.Sequential(
            nn.Conv2d(3,64,kernel_size=7,stride=2,padding=3),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(kernel_size=3,stride=2,padding=1)
        )
        self.layer1=nn.Sequential(
            nn.Conv2d(64,64,kernel_size=3,stride=1,padding=1),
            nn.BatchNorm2d(64),
            nn.Conv2d(64,64,kernel_size=3,stride=1,padding=1),
            nn.BatchNorm2d(64),
        )
        self.layer2=nn.Sequential(
            nn.Conv2d(64,128,kernel_size=3,stride=2,padding=1),
            nn.BatchNorm2d(128),
            nn.Conv2d(128,128,3,1,1),
            nn.BatchNorm2d(128),
        )
        self.layer2p2=nn.Sequential(
            nn.Conv2d(128,128,kernel_size=3,stride=1,padding=1),
            nn.BatchNorm2d(128),
            nn.Conv2d(128,128,3,1,1),
            nn.BatchNorm2d(128),
        )
        self.layer3=nn.Sequential(
            nn.Conv2d(128,256,3,2,1),
            nn.BatchNorm2d(256),
            nn.Conv2d(256,256,3,1,1),
            nn.BatchNorm2d(256),
        )
        self.layer3p2=nn.Sequential(
            nn.Conv2d(256,256,3,1,1),
            nn.BatchNorm2d(256),
            nn.Conv2d(256,256,3,1,1),
            nn.BatchNorm2d(256),
        )
```

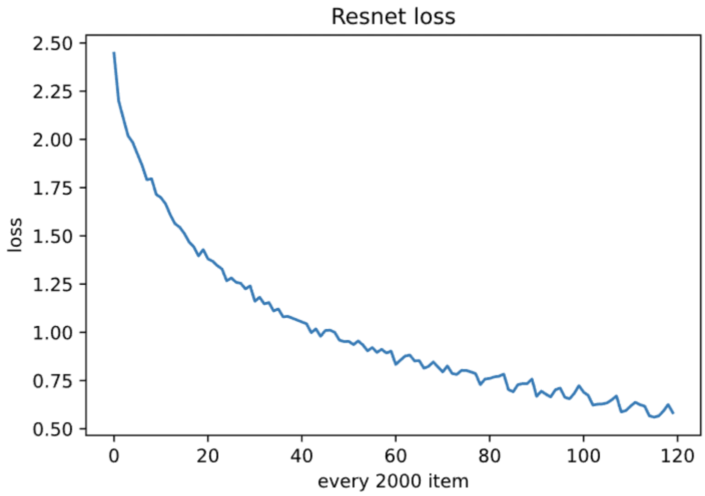
```python
        self.layer4=nn.Sequential(
            nn.Conv2d(256,512,3,2,1),
            nn.BatchNorm2d(512),
            nn.Conv2d(512,512,3,1,1),
            nn.BatchNorm2d(512),
        )
        self.layer4p2=nn.Sequential(
            nn.Conv2d(512,512,3,1,1),
            nn.BatchNorm2d(512),
            nn.Conv2d(512,512,3,1,1),
            nn.BatchNorm2d(512),
        )
        self.avgpool = nn.AdaptiveAvgPool2d(output_size=(1, 1))
        self.fc = nn.Linear(512, outchannel)
    def forward(self, x):
        output=self.conv1(x)
        #res1 64->64
        res=output
        output=self.layer1(output)
        output=F.relu(output+res)
        #res2 64->64
        res=output
        output=self.layer1(output)
        output=F.relu(output+res)
        #res3 64->128
        res=output
        reslink1=Reslink(64,128,mykernel_size=1,mystride=2,mypadding=0)
        res=reslink1(res)
        output=self.layer2(output)
        output=F.relu(output+res)
        #res4 128->128
        res=output
        output=self.layer2p2(output)
        output=F.relu(output+res)
        #res5 128->256
        res=output
        reslink1=Reslink(128,256,mykernel_size=1,mystride=2,mypadding=0)
        res=reslink1(res)
        output=self.layer3(output)
        output=F.relu(output+res)
        #res6 256->256
        res=output
        output=self.layer3p2(output)
        output=F.relu(output+res)
        #res7 256->512
        res=output
        reslink1=Reslink(256,512,mykernel_size=1,mystride=2,mypadding=0)
        res=reslink1(res)
        output=self.layer4(output)
        output=F.relu(output+res)
        #res8 512->512
        res=output
        output=self.layer4p2(output)
        output=F.relu(output+res
        output=self.avgpool(output)
        #转化为二维矩阵
        output=output.reshape(x.shape[0], -1)
        #线性展开
```

```
        output=self.fc(output)
        return output
```

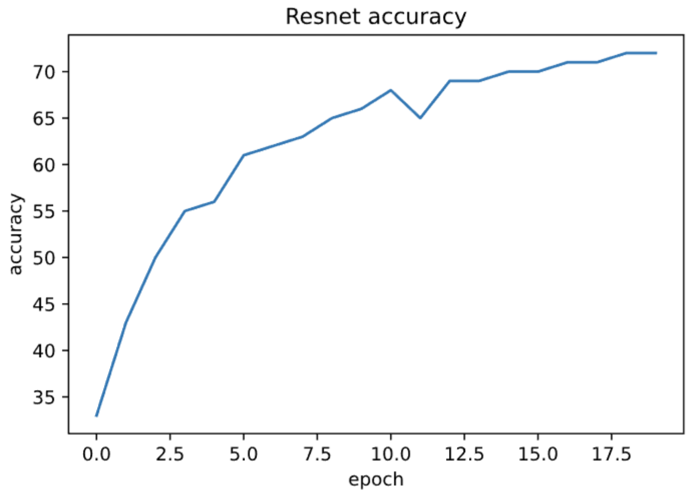
使用**torchsummary**打印网络结构，其功能特别强大，各层网络均给出shape，如下所示:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 16, 16]           9,472
       BatchNorm2d-2           [-1, 64, 16, 16]             128
         MaxPool2d-3             [-1, 64, 8, 8]               0
            Conv2d-4             [-1, 64, 8, 8]          36,928
       BatchNorm2d-5             [-1, 64, 8, 8]             128
            Conv2d-6             [-1, 64, 8, 8]          36,928
       BatchNorm2d-7             [-1, 64, 8, 8]             128
            Conv2d-8             [-1, 64, 8, 8]          36,928
       BatchNorm2d-9             [-1, 64, 8, 8]             128
           Conv2d-10             [-1, 64, 8, 8]          36,928
      BatchNorm2d-11             [-1, 64, 8, 8]             128
           Conv2d-12            [-1, 128, 4, 4]          73,856
      BatchNorm2d-13            [-1, 128, 4, 4]             256
           Conv2d-14            [-1, 128, 4, 4]         147,584
      BatchNorm2d-15            [-1, 128, 4, 4]             256
           Conv2d-16            [-1, 128, 4, 4]         147,584
      BatchNorm2d-17            [-1, 128, 4, 4]             256
           Conv2d-18            [-1, 128, 4, 4]         147,584
      BatchNorm2d-19            [-1, 128, 4, 4]             256
           Conv2d-20            [-1, 256, 2, 2]         295,168
      BatchNorm2d-21            [-1, 256, 2, 2]             512
           Conv2d-22            [-1, 256, 2, 2]         590,080
      BatchNorm2d-23            [-1, 256, 2, 2]             512
           Conv2d-24            [-1, 256, 2, 2]         590,080
      BatchNorm2d-25            [-1, 256, 2, 2]             512
           Conv2d-26            [-1, 256, 2, 2]         590,080
      BatchNorm2d-27            [-1, 256, 2, 2]             512
           Conv2d-28            [-1, 512, 1, 1]       1,180,160
      BatchNorm2d-29            [-1, 512, 1, 1]           1,024
           Conv2d-30            [-1, 512, 1, 1]       2,359,808
      BatchNorm2d-31            [-1, 512, 1, 1]           1,024
           Conv2d-32            [-1, 512, 1, 1]       2,359,808
      BatchNorm2d-33            [-1, 512, 1, 1]           1,024
           Conv2d-34            [-1, 512, 1, 1]       2,359,808
      BatchNorm2d-35            [-1, 512, 1, 1]           1,024
AdaptiveAvgPool2d-36            [-1, 512, 1, 1]               0
           Linear-37                   [-1, 10]           5,130
================================================================
```

训练loss曲线



准确度曲线图



准确度最终稳定在72%，可见Resnet相较于简单的CNN，准确率提升巨大。

Densenet

网络结构

参考资料（[DenseNet代码复现+超详细注释（PyTorch）]），从而实现一个简单的Dense net，不同于论文中的121层，这里，简易Dense net 结构如下：

> input->denseblock1->transistion1->denseblock2->线性连接层，denseblock中的numlayer均设置为6，最终网络层数为16层，与上文实现的Resnet18 复杂度类似，更好比较。

- DenseBlock模块: 堆叠一定数量的layer（设置为6），这些layer本质上就是两个卷积层，关键的是参数growth_rate，用于逐次增加通道数量（设置为32）。
- Transition模块：1×1卷积核负责降低通道数，2×2AvgPool负责降低特征层宽度，可以起到压缩模型的作用。

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class denseBasic(nn.Module):
    def __init__(self,in_channels,growth_rate):
        super(denseBasic, self).__init__()
        self.layer=nn.Sequential(
            nn.BatchNorm2d(in_channels),
```

```
            nn.ReLU(),
            # 手动规定 k=4
            nn.Conv2d(in_channels,growth_rate*4,kernel_size=(1,1)),
            nn.BatchNorm2d(growth_rate*4),
            nn.ReLU(),
            nn.Conv2d(growth_rate*4,growth_rate,kernel_size=(3,3),padding=1),
        )

    def forward(self,x):
        x=torch.cat((x,self.layer(x)),dim=1)
        return x


class denseBlock(nn.Module):
    def __init__(self,in_channels,growth_rate,num_layer):
        super(denseBlock, self).__init__()
        block=[]
        # 随着layer层数的增加，每增加一层，输入的特征图就增加一倍growth_rate
        for i in range(num_layer):
            block.append(denseBasic(in_channels,growth_rate))
            in_channels+=growth_rate
        self.denseblock=nn.Sequential(*block)

    def forward(self,x):
        return self.denseblock(x)

class transistion(nn.Module):
    def __init__(self,in_channels,out_channels):
        super(transistion, self).__init__()
        self.mytransistion = nn.Sequential(
            nn.BatchNorm2d(in_channels),
            # 默认设置为（3x3）,padding=1,stride = 1
            nn.Conv2d(in_channels,out_channels,kernel_size=
(3,3),stride=1,padding=1)
        )

    def forward(self,x):
        x = self.mytransistion(x)
        return x


class denseNet(nn.Module):
    def __init__(self):
        super(denseNet, self).__init__()
        #处理input 3*32*32
        nn.sovleinput = nn.Sequential(
            nn.Conv2d(3,32,kernel_size=(7,7),stride=2,padding=3),
            nn.BatchNorm2d(32),#防止过拟合
            # 32*32*32
            nn.MaxPool2d(2),
            # 32*16*16
            nn.ReLU(),
        ),
        self.denseblock1=denseBlock(32,32,6)
        #(32+32*6)*16*16
        self.transistion1=transistion(32+32*6,64)
        #64*8*8
        self.denseblock2=denseBlock(64,64,6)
```

```
        self.fc = nn.Sequential(
            #(64+64*6)*4*4
            nn.Linear((64+64*6)*4*4,128),
            nn.ReLU(),
            nn.Linear(128,10),
        )

    def forward(self,x):
        x = self.sovleinput(x)
        x=self.denseblock1(x)
        x=self.transistion1(x)
        x=self.denseblock2(x)
        x=x.reshape(x.shape[0],-1)
        x = self.fc(x)
        return x
```

调用print()函数，打印网络结构。

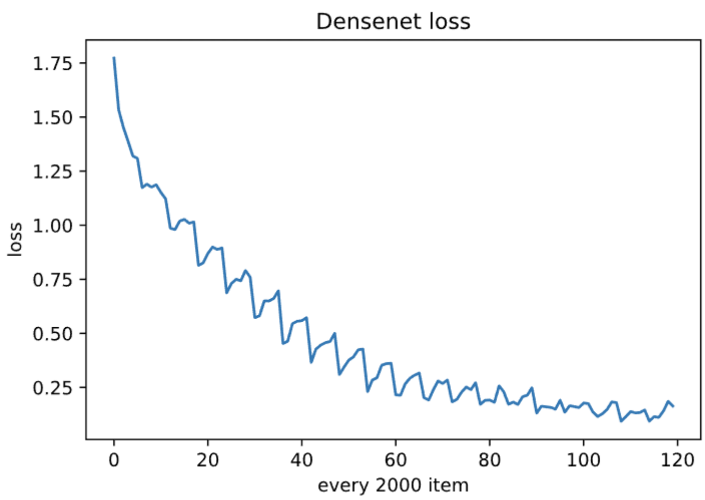
```
denseNet(
  (sovleinput): Sequential(
    (0): Conv2d(3, 32, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): ReLU()
  )
  (denseblock1): denseBlock(
    (denseblock): Sequential(
      (0): denseBasic(
        (layer): Sequential(
          (0): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (1): ReLU()
          (2): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))
          (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (4): ReLU()
          (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        )
      )
      (1): denseBasic(
        (layer): Sequential(
          (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (1): ReLU()
          (2): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1))
          (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (4): ReLU()
          (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        )
      )
      (2): denseBasic(
        (layer): Sequential(
          (0): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
          (1): ReLU()
          (2): Conv2d(96, 128, kernel_size=(1, 1), stride=(1, 1))
          (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (4): ReLU()
          (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        )
      )
      (3): denseBasic(
        (layer): Sequential(
          (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (1): ReLU()
          (2): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1))
          (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (4): ReLU()
          (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        )
      )
      (4): denseBasic(
        (layer): Sequential(
          (0): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (1): ReLU()
          (2): Conv2d(160, 128, kernel_size=(1, 1), stride=(1, 1))
          (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (4): ReLU()
          (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        )
      )
      (5): denseBasic(
        (layer): Sequential(
          (0): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (1): ReLU()
          (2): Conv2d(192, 128, kernel_size=(1, 1), stride=(1, 1))
          (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (4): ReLU()
          (5): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        )
      )
    )
  )
  (transistion1): transistion(
    (mytransistion): Sequential(
      (0): BatchNorm2d(224, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (1): Conv2d(224, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
  )
  (denseblock2): denseBlock(
    (denseblock): Sequential(
      (0): denseBasic(
        (layer): Sequential(
```

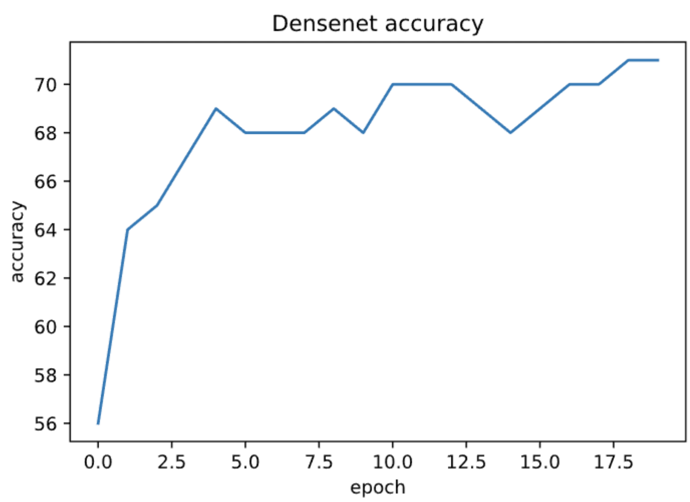
```
        (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
        (2): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1))
        (3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (4): ReLU()
        (5): Conv2d(256, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (1): denseBasic(
      (layer): Sequential(
        (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
        (2): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
        (3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (4): ReLU()
        (5): Conv2d(256, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (2): denseBasic(
      (layer): Sequential(
        (0): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
        (2): Conv2d(192, 256, kernel_size=(1, 1), stride=(1, 1))
        (3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (4): ReLU()
        (5): Conv2d(256, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (3): denseBasic(
      (layer): Sequential(
        (0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
        (2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (4): ReLU()
        (5): Conv2d(256, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
    (4): denseBasic(
      (layer): Sequential(
        (0): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
        (2): Conv2d(320, 256, kernel_size=(1, 1), stride=(1, 1))
        (3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (4): ReLU()
        (5): Conv2d(256, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
    )
```

```
      (5): denseBasic(
        (layer): Sequential(
          (0): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (1): ReLU()
          (2): Conv2d(384, 256, kernel_size=(1, 1), stride=(1, 1))
          (3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (4): ReLU()
          (5): Conv2d(256, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        )
      )
    )
  )
  (fc): Sequential(
    (0): Linear(in_features=28672, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

训练loss曲线



准确度曲线图



准确度最终稳定在71%，略低于Resnet18，也许是因为作者实现的简易版的Densenet 较Resnet18更加简单，只有十六层，使得最后的额准确度反而低了一些。

SENet

网络结构

参考资料（(pytorch——SENet详解及PyTorch实现)），在Resnet18 的基础上，加入SE模块，从而提高Resnet18的网络性能。

- **Resnet18：**原理同上。
- **SE模块：**经过一个卷积层，已知不同的卷积核会提取不同的特征，现在要给这些特征一定的权重，最后进行特征的拼接，得到更加优秀的特征。

  所以，先经过一个全局池化层，变为$1 \times 1 \times C$，接下来进行线性连接，最后由sigmoid选取出概率最大特征进行拼接。

```
class SeNetBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(SeNetBlock, self).__init__()
        # 准备工作
        self.prepare = nn.Sequential(
            nn.BatchNorm2d(in_channels),
            nn.ReLU(),
        )
        # 第一个卷积
        self.myconv1 = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride,
padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
        )
        # 传统卷积
        self.myconv2 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1,
padding=1, bias=False)
        )
        # stride = 2 或者 channel 改变 都需要使用卷积层
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1,
stride=stride, bias=False)
            )
        # SE layers
        self.fc1 = nn.Conv2d(out_channels, out_channels//16, kernel_size=1)
        self.fc2 = nn.Conv2d(out_channels//16, out_channels, kernel_size=1)

    def forward(self, x):
        out = self.prepare(x)
        shortcut = self.shortcut(out) if hasattr(self, 'shortcut') else x
        out = self.myconv1(out)
        out = self.myconv2(out)
        # Squeeze
        w = F.avg_pool2d(out, out.size(2))
        w = F.relu(self.fc1(w))
        w = F.sigmoid(self.fc2(w))
        # Excitation
        out = out * w
        out += shortcut
```

```
        return out


class SENet(nn.Module):
    def __init__(self, block=SENetBlock, num_blocks=[2,2,2,2], num_classes=10):
        super(SENet, self).__init__()
        self.in_channels = 64
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,
bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block,  64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512, num_classes)

    def _make_layer(self, block, out_channels, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_channels, out_channels, stride))
            self.in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out
```

使用print()函数打印网络结构：

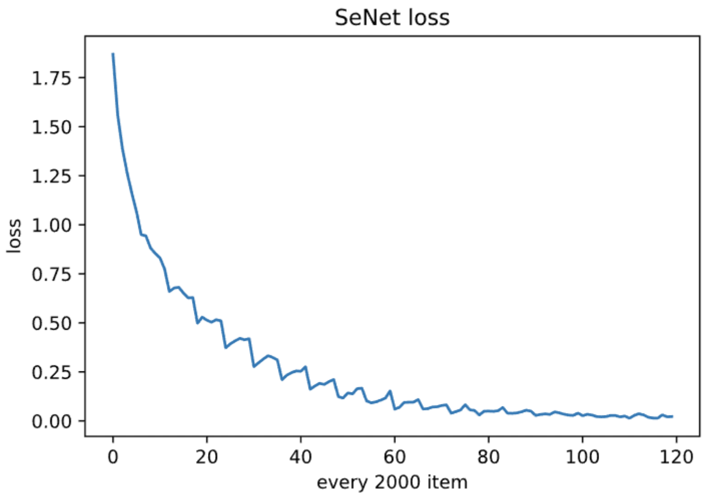
```
SENet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (layer1): Sequential(
    (0): SENetBlock(
      (prepare): Sequential(
        (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
      )
      (myconv1): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU()
      )
      (myconv2): Sequential(
```

```
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      )
      (fc1): Conv2d(64, 4, kernel_size=(1, 1), stride=(1, 1))
      (fc2): Conv2d(4, 64, kernel_size=(1, 1), stride=(1, 1))
    )
    (1): SeNetBlock(
      (prepare): Sequential(
        (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
      )
      (myconv1): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU()
      )
      (myconv2): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      )
      (fc1): Conv2d(64, 4, kernel_size=(1, 1), stride=(1, 1))
      (fc2): Conv2d(4, 64, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (layer2): Sequential(
    (0): SeNetBlock(
      (prepare): Sequential(
        (0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
      )
      (myconv1): Sequential(
        (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU()
      )
      (myconv2): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      )
      (shortcut): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
      )
      (fc1): Conv2d(128, 8, kernel_size=(1, 1), stride=(1, 1))
      (fc2): Conv2d(8, 128, kernel_size=(1, 1), stride=(1, 1))
    )
    (1): SeNetBlock(
      (prepare): Sequential(
        (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
      )
      (myconv1): Sequential(
```

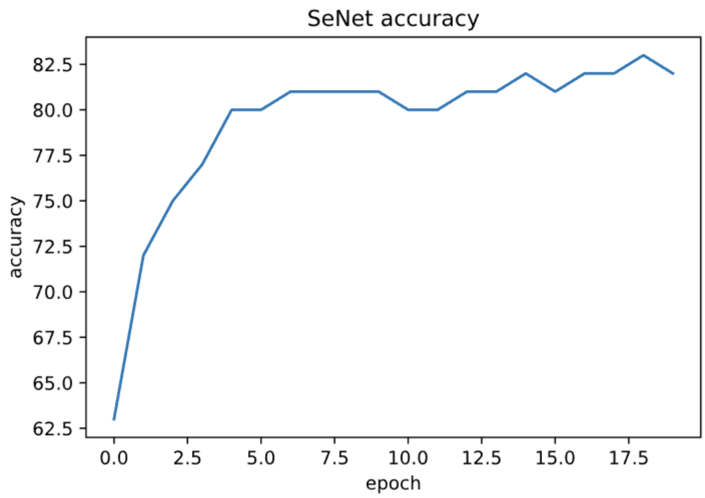
```
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU()
      )
      (myconv2): Sequential(
        (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      )
      (fc1): Conv2d(128, 8, kernel_size=(1, 1), stride=(1, 1))
      (fc2): Conv2d(8, 128, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (layer3): Sequential(
    (0): SeNetBlock(
      (prepare): Sequential(
        (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
      )
      (myconv1): Sequential(
        (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU()
      )
      (myconv2): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      )
      (shortcut): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
      )
      (fc1): Conv2d(256, 16, kernel_size=(1, 1), stride=(1, 1))
      (fc2): Conv2d(16, 256, kernel_size=(1, 1), stride=(1, 1))
    )
    (1): SeNetBlock(
      (prepare): Sequential(
        (0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
      )
      (myconv1): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU()
      )
      (myconv2): Sequential(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      )
      (fc1): Conv2d(256, 16, kernel_size=(1, 1), stride=(1, 1))
      (fc2): Conv2d(16, 256, kernel_size=(1, 1), stride=(1, 1))
    )
```

```
  )
  (layer4): Sequential(
    (0): SeNetBlock(
      (prepare): Sequential(
        (0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
      )
      (myconv1): Sequential(
        (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU()
      )
      (myconv2): Sequential(
        (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      )
      (shortcut): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      )
      (fc1): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1))
      (fc2): Conv2d(32, 512, kernel_size=(1, 1), stride=(1, 1))
    )
    (1): SeNetBlock(
      (prepare): Sequential(
        (0): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (1): ReLU()
      )
      (myconv1): Sequential(
        (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): ReLU()
      )
      (myconv2): Sequential(
        (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      )
      (fc1): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1))
      (fc2): Conv2d(32, 512, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (linear): Linear(in_features=512, out_features=10, bias=True)
)
```

训练loss曲线



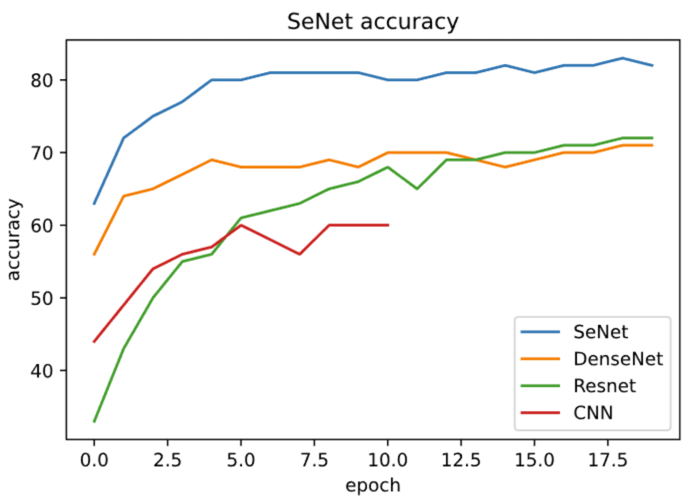
准确度曲线图



SeNet18 的准确率最终稳定在82%，明显优于Resnet18和简易的DenseNet。

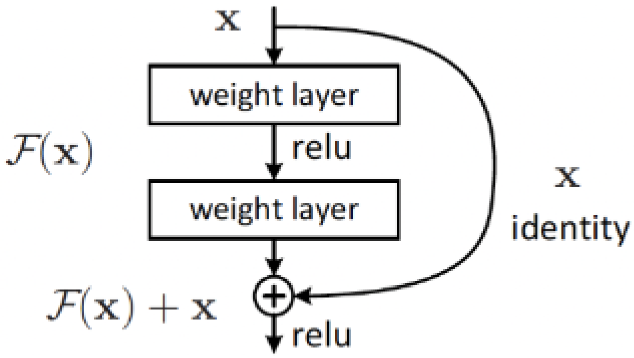
解释没有跳跃连接的卷积网络、ResNet、DenseNet、SE-ResNet在训练过程中有什么不同

结合四种网络的准确度曲线图，如下所示：



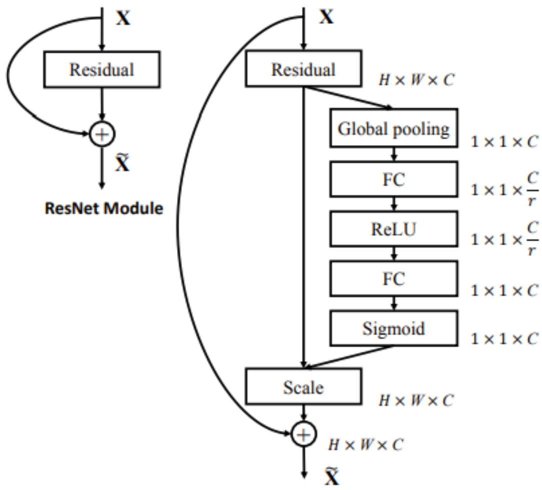
- 原始CNN：没有跳跃连接，当CNN的卷积层过多时，模型会过拟合。
- Resnet：残差连接，指的是在两个卷积层之间，加入了一个短路链接（case1:直接相连；case2: 输入和输出通道数不同，此时需要使用一层卷积），这样可以将上层提取取到的特征直接传递到下层，避免梯度消失的问题。

例如：在训练过程中上面的卷积层已经提取到了很好的特征，此时通过残差连接直接传导到下层，而非继续进行卷积操作从而导致过拟合，跳过一部分卷积层，维持已经提取到的很好的特征。Resnet可以很好的解决上述情况。



图片来源[1]，如图所示：$\mathcal{F}$表示卷积操作，x表示输入的特征，最终的输出为$y=\mathcal{F}(x)+x$。

- SE-ResNet：结合传统的Resnet和SE模块，从而形成一种新的残差连接，如下图所示：

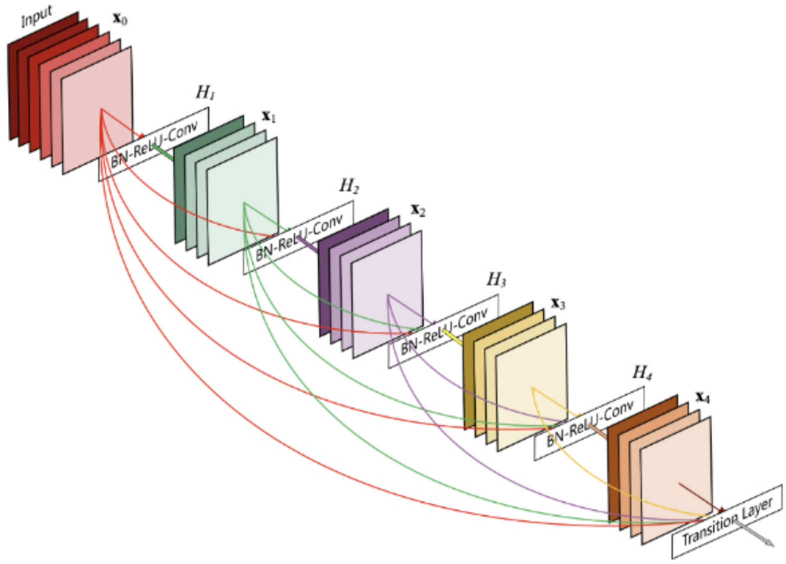


图片来源[3]，SE模块被添加到ResNet的基本块中，通过门控机制对特征进行重新加权。这可以使网络更加关注重要的特征，提高模型的性能。

SE模块：经过一个卷积层，已知不同的卷积核会提取不同的特征，现在要给这些特征一定的权重，最后进行特征的拼接，得到更加优秀的特征。所以，先经过一个全局池化层，变为$1\times1\times C$，接下来进行线性连接，最后由sigmoid选取出概率最大特征进行拼接。

Resnet拼接：保留Resnet中的残差连接，同样需要考虑两种情况（case1:直接相连；case2: 输入和输出通道数不同，此时需要使用一层卷积）

- DenseNet：DenseNet的连接方式与上述均不同， 为了进一步改善层之间的信息流，提出了一种不同的连接模式，引入了从任何层到所有后续层的直接连接。



图片来源[2],DenseNet通过密集连接（dense connection）改进了传统卷积网络的结构。

DenseBlock模块: 堆叠一定数量的layer，这些layer本质上就是两个卷积层，关键的是参数growth_rate，用于逐次增加通道数量。在DenseBlock中，各个层的特征图大小一致，可以在channel维度上连接。

Transition模块： 1×1卷积核负责降低通道数，2×2AvgPool负责降低特征层宽度，可以起到压缩模型的作用。

在DenseNet中，每一层的输出都与之前所有层的输出连接在一起，使得信息能够在网络中自由流动。这种密集连接的结构可以增强特征重用和梯度流动，从而提高模型的效果和训练速度。DenseNet的feature map比ResNet大很多，导致卷积过程的计算量ResNet大很多。

综上：ResNet通过跳跃连接解决梯度问题，DenseNet通过密集连接促进信息流动，SE-ResNet通过SE模块对特征进行自适应加权。这些改进使得网络能够更好地训练和利用特征，提高模型的性能和收敛速度。

从准确度曲线图也可以看出：

收敛速度：SE-ResNet>DenseNet>Resnet

准确度：SE-ResNet>Resnet≈DenseNet

训练时间：DenseNet>SE-ResNet>ResNet

参考文献

[1] K.He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings ofthe IEEE conference on computer vision and pattern recognition, pages 770-778,2016

[2] G.Huang,Z.Liu,L.VanDerMaaten,andK.Q.Weinberger. Densely connected convolutional networksIn Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700-47082017.

[3] Squeeze-and-Excitation Networks