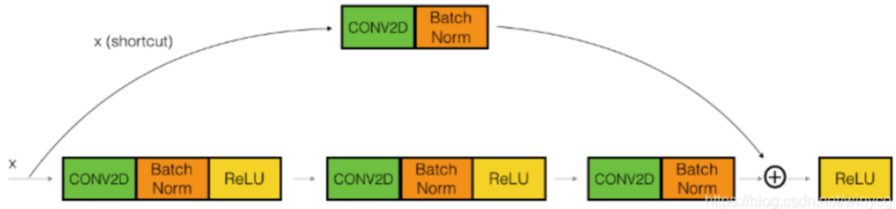
0421ResNet

一、整体架构

1. 构建思路：

   先构建残差块（类），再用残差块构建残差网络（Resnet 类）



2. 残差块

   残差块与普通网络的不同之处：

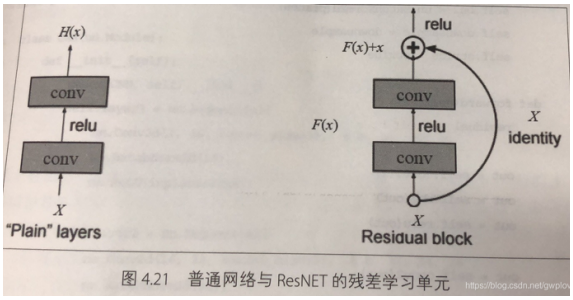
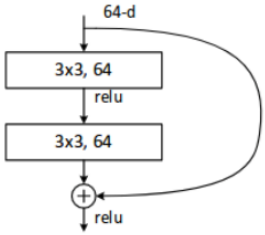


图 4.21　普通网络与 ResNET 的残差学习单元

BasicBlock 类：（用于 34 层及以下的残差网络）

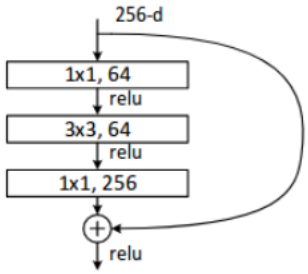



```python
# 定义一个残差块
class ResidualBlock(nn.Module):
    def __init__(self, in_channel, out_channel, stride=1, downsample=None):
        super(ResidualBlock, self).__init__()
        # self.in_channel = in_channel
        # self.out_channel = out_channel
        # self.stride = stride

        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=in_channel, out_channels=out_channel, kernel_size=3, stride=stride, padding=1),
            nn.BatchNorm2d(out_channel),
            nn.ReLU(),
            nn.Conv2d(in_channels=out_channel, out_channels=out_channel, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(out_channel),
        )
        self.downsample = downsample
        self.relu = nn.ReLU()

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        # print(out.shape)
        if self.downsample:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out
```

BottleNeck 类：（用于 50 层及以上的残差网络）



3. 残差网络

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

BasicBlock 构建了单个残差块，而 Resnet 类下的_make_layer_函数则是将多个残差块堆叠起来，形成一个残差层。(一个残差层中，相同的残差块会用上好多次，所以这个函数本质上是将工作简单化，避免反复调用造成的书写累赘。)

```python
# 定义网络
class Resnet(nn.Module):
    def __init__(self,block,num_block):
        super(Resnet, self).__init__()
        self.in_channel = 64
        self.conv0 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=7, stride=2),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.layer1 = self._make_layer(block, 64, num_block[0])
        self.layer2 = self._make_layer(block, 128, num_block[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_block[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_block[3], stride=2)

        self.avgpool = nn.AvgPool2d(kernel_size=4)
        self.fc = nn.Sequential(
            nn.Linear(512, 2),
            nn.Softmax(dim=1)
        )

    def _make_layer(self, block, out_channel, num_block, stride=1):
        downsample = None
        if stride != 1 or self.in_channel != out_channel:
            downsample = nn.Sequential(
                nn.Conv2d(self.in_channel, out_channel, kernel_size=3, stride=stride, padding=1, bias=False),
                nn.BatchNorm2d(out_channel)
            )
        layers = []
        layers.append(block(self.in_channel, out_channel, stride, downsample))
        self.in_channel = out_channel

        for i in range(1, num_block):
            layers.append(block(out_channel, out_channel))
        return nn.Sequential(*layers)
```

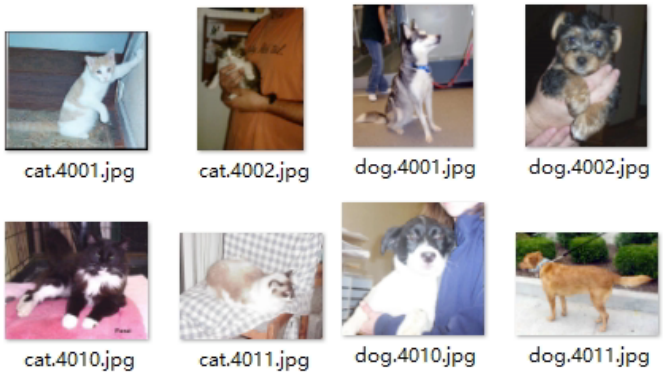
```python
    def forward(self, x):
        x = self.conv0(x)
        # print(x.shape)
        x = self.layer1(x)
        # print(x.shape)
        x = self.layer2(x)
        # print(x.shape)
        x = self.layer3(x)
        # print(x.shape)
        x = self.layer4(x)
        x = self.avgpool(x)
        # print(x.shape)
        x = x.view(x.size(0), -1)
        # print(x.shape)
        x = self.fc(x)
        return x
```

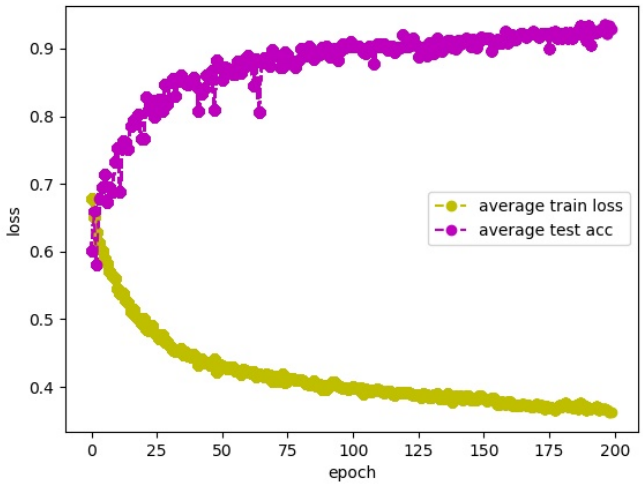
注意：
downsample 只是作为参数传进 BasicBlock 类中，不需要在 BasicBlock 类中定义，只需要初始化即可。因为需要在 Resnet 类下的_make_layer_函数中调用 BasicBlock 类，所以 downsample 是在 Resnet 类下的_make_layer_函数中定义的。

当输入输出的维度不匹配时，或者 stride 不为 1 时，就要用 1*1 的卷积核对输入进行维度的改变，再与输出相加。

二、网络训练

总览：

算法	Resnet34
日期	0421
文件夹	0421resnet
数据集	猫狗网络图 猫（4000+1000） 狗（4000+1000） 
超参数	Epoch200/200 Batchsize=16
网络定义	Resnet34 BasicBlock [3,4,6,3]
图像处理	随机中心翻转 随机剪裁 转张量 归一化
输出	[epoch 1, batch 20] loss: 0.853 [epoch 1, batch 40] loss: 0.682 [epoch 1, batch 60] loss: 0.708 [epoch 1, batch 80] loss: 0.611 [epoch 1, batch 100] loss: 0.826 [epoch 1, batch 120] loss: 0.682 [epoch 1, batch 140] loss: 0.713 [epoch 1, batch 160] loss: 0.660 [epoch 1, batch 180] loss: 0.750 [epoch 1, batch 200] loss: 0.621 [epoch 1, batch 220] loss: 0.665 [epoch 1, batch 240] loss: 0.771 [epoch 1, batch 260] loss: 0.561 [epoch 1, batch 280] loss: 0.591 [epoch 1, batch 300] loss: 0.699 [epoch 1, batch 320] loss: 0.702 [epoch 1, batch 340] loss: 0.732 [epoch 1, batch 360] loss: 0.657 [epoch 1, batch 380] loss: 0.480 [epoch 1, batch 400] loss: 0.818 [epoch 1, batch 420] loss: 0.705 [epoch 1, batch 440] loss: 0.706 [epoch 1, batch 460] loss: 0.689 [epoch 1, batch 480] loss: 0.704 [epoch 1, batch 500] loss: 0.596 epoch 1 av_train_loss: 0.678 lr: 0.00100 epoch 1 av_test_acc: 0.601

训练曲线图	
曲线分析	Loss: 0.853->0.36 上下 Acc：0.601->0.93 上下 最佳：epoch197，loss0.367，acc0.935，lr0.00049
保存模型参数	epoch2+loss0.651+acc0.658+lr0.00100.pth epoch4+loss0.613+acc0.678+lr0.00100.pth epoch5+loss0.601+acc0.695+lr0.00100.pth epoch6+loss0.590+acc0.715+lr0.00100.pth epoch10+loss0.561+acc0.733+lr0.00100.pth epoch11+loss0.545+acc0.754+lr0.00100.pth epoch13+loss0.538+acc0.763+lr0.00100.pth epoch16+loss0.511+acc0.785+lr0.00100.pth epoch17+loss0.514+acc0.793+lr0.00100.pth epoch19+loss0.501+acc0.802+lr0.00100.pth epoch22+loss0.485+acc0.829+lr0.00100.pth epoch29+loss0.467+acc0.847+lr0.00100.pth epoch31+loss0.460+acc0.851+lr0.00100.pth epoch32+loss0.455+acc0.856+lr0.00100.pth epoch35+loss0.456+acc0.861+lr0.00100.pth epoch47+loss0.436+acc0.867+lr0.00100.pth epoch49+loss0.423+acc0.883+lr0.00100.pth epoch61+loss0.422+acc0.889+lr0.00100.pth epoch70+loss0.409+acc0.898+lr0.00100.pth epoch81+loss0.410+acc0.900+lr0.00100.pth epoch84+loss0.403+acc0.903+lr0.00100.pth epoch97+loss0.399+acc0.906+lr0.00100.pth epoch100+loss0.398+acc0.909+lr0.00100.pth epoch120+loss0.391+acc0.920+lr0.00100.pth epoch159+loss0.374+acc0.925+lr0.00070.pth epoch170+loss0.371+acc0.926+lr0.00070.pth epoch187+loss0.368+acc0.928+lr0.00070.pth epoch188+loss0.375+acc0.934+lr0.00070.pth epoch197+loss0.367+acc0.935+lr0.00049.pth
预测	this picture might be: dog , score: 1.0 this picture might be: cat , score: 1.0 this picture might be: dog , score: 1.0 this picture might be: dog , score: 1.0 this picture might be: cat , score: 1.0

			
备注	由训练图可知，在 200 epochs 训练完之后，loss 和 acc 并没有达到一个平稳的状态，说明仍然可以训练。		

三、相关内容链接

https://blog.csdn.net/winycg/article/details/86709991

https://blog.csdn.net/gwplovekimi/article/details/83578473