

DenseNet

DenseNet 为 CVPR2017 年的 Best Paper, DenseNet 脱离了加深网络层数(ResNet)和加宽网络结构(Inception)来提升网络性能的定式思维, 从特征的角度考虑, 通过特征重用和旁路(Bypass)设置, 既大幅度减少了网络的参数量, 又在一定程度上缓解了 gradient vanishing 问题的产生. 结合信息流和特征复用的假设, DenseNet 当之无愧成为 2017 年计算机视觉顶会的年度最佳论文。DenseNet 模型, 它的基本思路与 ResNet 一致, 但是它建立的是前面所有层与后面层的密集连接(dense connection), 它的名称也是由此而来。DenseNet 的另一大特色是通过特征在 channel 上的连接来实现特征重用(feature reuse)。这些特点让 DenseNet 在参数和计算成本更少的情形下实现比 ResNet 更优的性能。本篇文章首先介绍 DenseNet 的原理以及网路架构, 然后讲解 DenseNet 在 Pytorch 上的实现。ResNet 的一个最主要的优势便是梯度可以流经恒等函数到达靠前的层. 但恒等映射和非线性变换输出的叠加方式是相加, 这在一定程度上破坏了网络中的信息流. 为了进一步优化信息流的传播, DenseNet 提出了图示的网络结构。

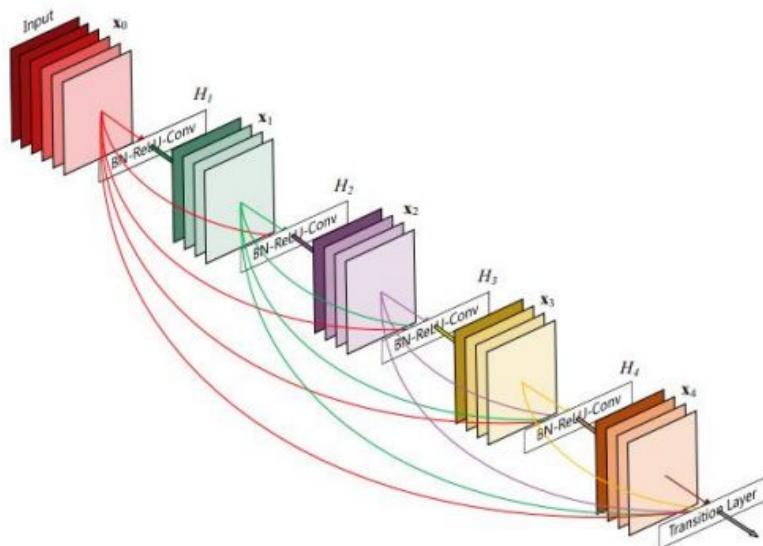
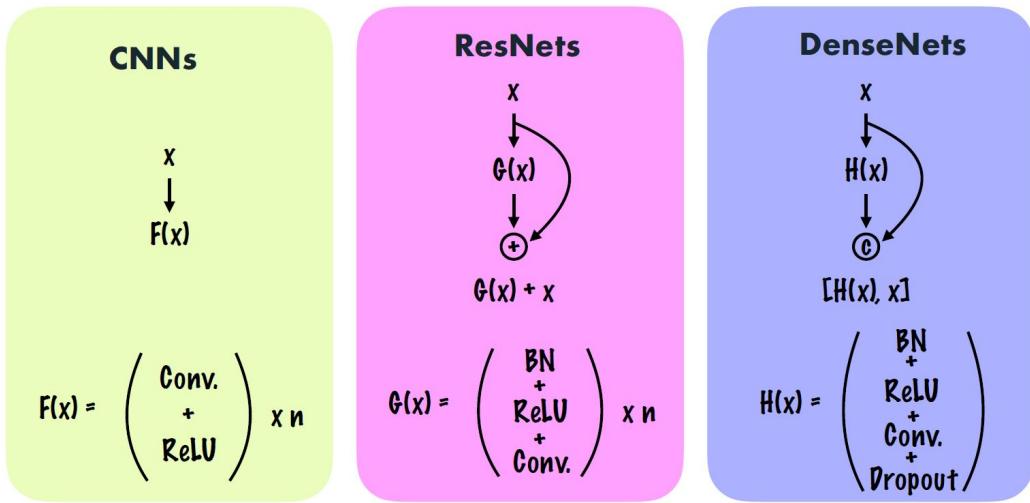


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.



简单的说，传统的 CNN 是的 layers 是一种将输入 x 映射为 $F(x)$ ，再让 $F(x)$ 去拟合目标分布的过程。而 ResNet 的 layer 则是通过在输入和输出之间的一条旁路，将输入直接连到 layer 的输出，这样 layer 所要拟合的对象就从 $F(x)$ 变成了 $G(x)$ ，抽象地说，这样的连接使得 layer 只需要拟合 0 便好（而不是去拟合赋值更大的 x ），这大大降低了深层网络在学习过程中的梯度消失的问题，因为在 0 附近，任何变动都是相对来说较大的（针对这一问题的解决方法还有 batch_norm，输入归一化等等，但都没有这个简单直接而又精巧）。而 DenseNet（见图-2）则一不做二不休，既然输入与输出之间要有连接，我们就连多一点：在一个 DenseBlock（图中黄、红、蓝、深蓝色代表不同的 Denseblock）模块内，每一层的输入都来自这一层之前所有层的输入。而对 DenseBlock 而言，其输入则同样来自于在其之前所有的 Denseblock 和原始输入 x 。

一、设计理念

DenseNet 的一个优点是网络更窄，参数更少，很大一部分原因得益于这种 dense block 的设计，后面有提到在 dense block 中每个卷积层的输出 feature map 的数量都很小（小于 100），而不是像其他网络一样动不动就几百上千的宽度。同时这种连接方式使得特征和梯度的传递更加有效，网络也就更加容易训练。

原文的一句话非常喜欢：Each layer has direct access to the gradients from the loss function and the original input signal, leading to an implicit deep supervision. 直接解释了为什么这个网络的效果会很好。前面提到过梯度消失问题在网络深度越深的时候越容易出现，原因就是输入信息和梯度信息在很多层之间传递导致的，而现在这种 dense connection 相当于每一层都直接连接 input 和 loss，因此就可以减轻梯度消失现象，这样更深网络不是问题。另外作者还观察到这种 dense connection 有正则化的效果，因此对于过拟合有一定的抑制作用，博主认为是因为参数减少了（后面会介绍为什么参数会减少），所以过拟合现象减轻。

相比 ResNet，DenseNet 提出了一个更激进的密集连接机制：即互相连接所有的层，具体来说就是每个层都会接受其前面所有层作为其额外的输入。图 1 为 ResNet 网络的连接机制，作为对比，图 2 为 DenseNet 的密集连接机制。可以看到，ResNet 是每个层与前面的某层（一般是 2~3 层）短路连接在一起，连接方式是通过元素级相加。而在 DenseNet 中，每个层都会与前面所有层在 channel 维度上连接（concat）在一起（这里各个层的特征图大小是相同的，后面会有说明），并作为下一层的输入。对于一个 L 层的网络，DenseNet 共包含 $\frac{L(L+1)}{2}$ 个连接，相比 ResNet，这是一

种密集连接。而且 DenseNet 是直接 concat 来自不同层的特征图，这可以实现特征重用，提升效率，这一特点是 DenseNet 与 ResNet 最主要的区别。

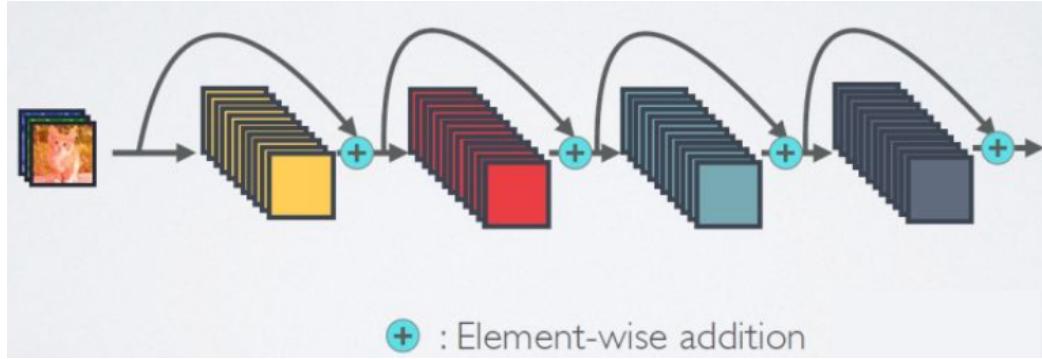


图 1 ResNet 网络的短路连接机制（其中+代表的是元素级相加操作）

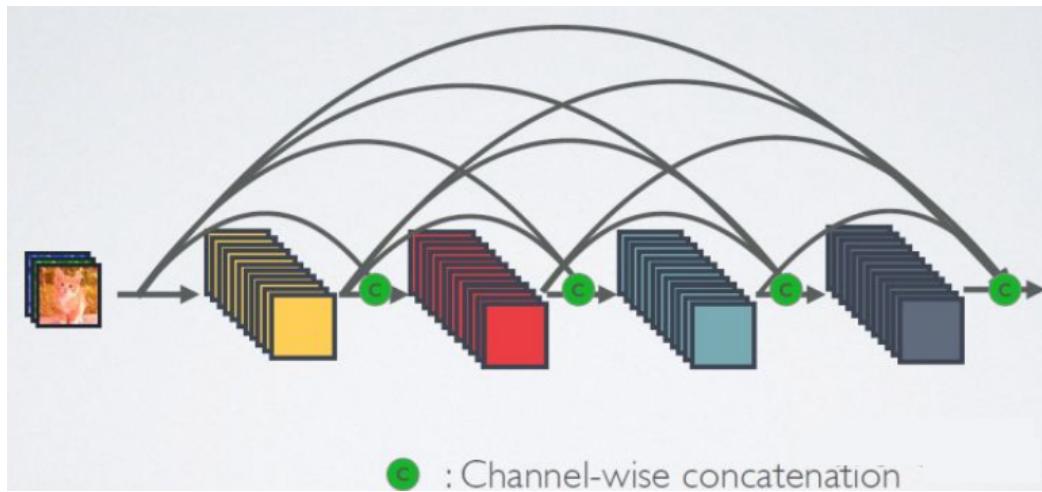


图 2 DenseNet 网络的密集连接机制（其中 c 代表的是 channel 级连接操作）

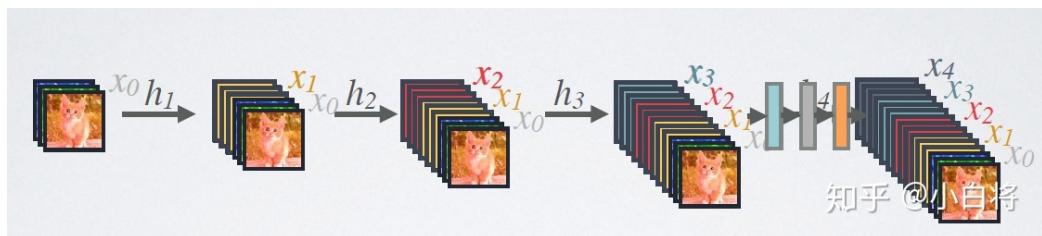
如果用公式表示的话，传统的网络在 l 层的输出为： $x_l = H_l(x_{l-1})$

而对于 ResNet，增加了来自上一层输入的 identity 函数： $x_l = H_l(x_{l-1}) + x_{l-1}$

在 DenseNet 中，会连接前面所有层作为输入： $x_l = H_l([x_0, x_1, \dots, x_{l-1}])$

其中， $H_l(\cdot)$ 代表是非线性转化函数 (non-linear transformation)，它是一个组合操作，其可能包括一系列的 BN(Batch Normalization), ReLU, Pooling 及 Conv 操作。注意这里 l 层与 $l-1$ 层之间可能实际上包含多个卷积层。

DenseNet 的前向过程如图 3 所示，可以更直观地理解其密集连接方式，比如 h_3 的输入不仅包括来自 h_2 的 x_2 ，它们是在 channel 维度上连接在一起的。



CNN 网络一般要经过 Pooling 或者 $stride > 1$ 的 Conv 来降低特征图的大小，而 DenseNet 的密集连接方式需要特征图大小保持一致。为了解决这个问题，DenseNet 网络中使用 DenseBlock+Transition 的结构，其中 DenseBlock 是包含很多层的模块，每个层的特征图大小相同，层与层之间采用密集连接方式。而 Transition 模块是连接两个相邻的 DenseBlock，并且通过 Pooling 使特征图大小降低。下图 DenseNet 的

网路结构，它共包含 4 个 DenseBlock，各个 DenseBlock 之间通过 Transition 连接在一起。

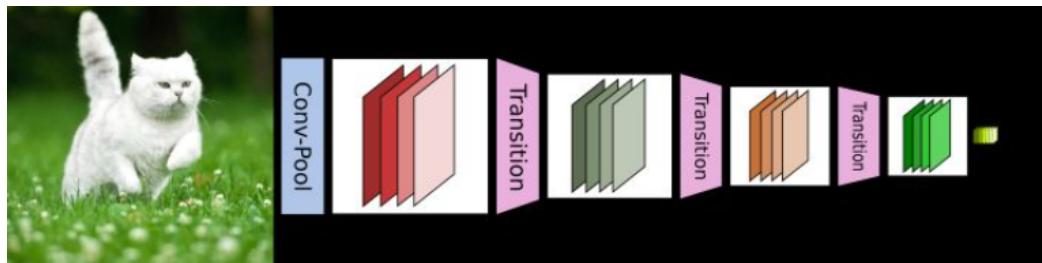
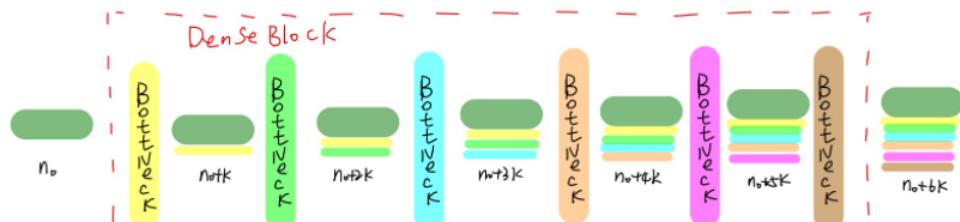
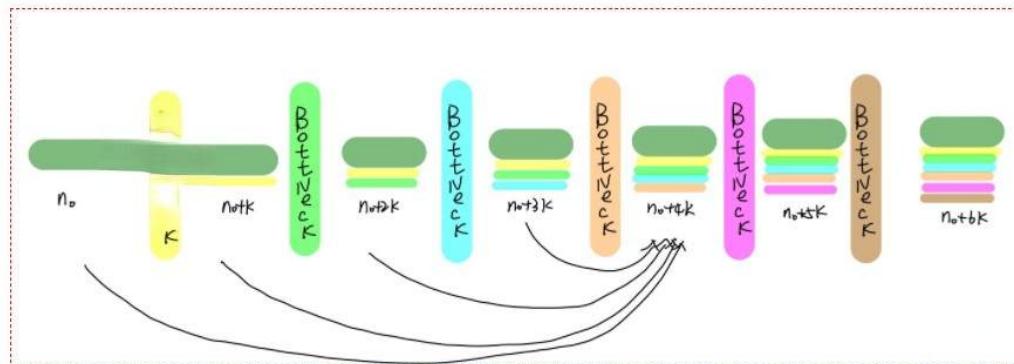


图 4 使用 DenseBlock+Transition 的 DenseNet 网络

DenseNet-121 的第一个 DenseBlock 包含了 6 个 BottleNeck, BottleNeck 之间是串联在一起的。



以粉红色 BottleNeck 为例,说明整个 DenseBlock 为密集连接.



仔细观察粉红色 BotteNeck 的输入,其实是来自于前面每一层 BotteNeck 输出和原始输入的堆叠,而且每一个 BotteNeck 的输入都是其前面所有层输出的堆叠,这就是 DenseNet 为什么是密集连接的原因,也是 DenseNet 取得良好效果的原因:

- 传递到粉红色 BotteNeck 的梯度,能直接传递到其前面各层 BotteNeck 中,一方面避免了梯度消失,另一方面加快了参数的迭代速度,提高了训练效率
- 网络在前向传播过程中,每个 BotteNeck 利用其前面所有网络层的输出结果作为输入,产生 k 个特征图.作者认为这是一个利用当前"集体成果"(前面所有层的输出),产生新特征,同时再将新特征加入"集体成果"中,不断成长壮大的过程.
- ResNet 中 BotteNeck 的输出和原始输入的融合方式采用的是相加,作者认为这种方式会破坏已经学到的特征,因此采用 concat 的方式.

二、网络结构

DenseNet 的整体结构主要包含稠密块 (Dense Blocks) 和过渡块 (transition layers)。

1. 网络参数解释

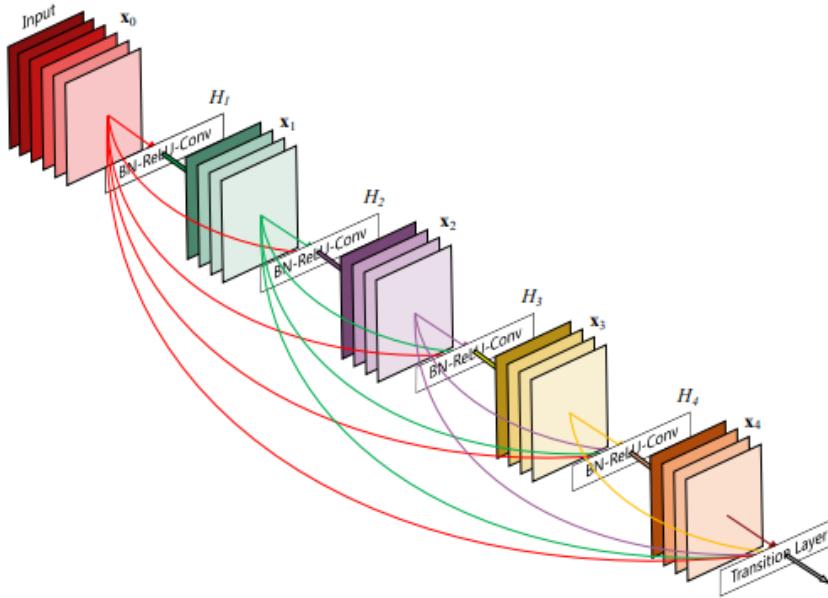
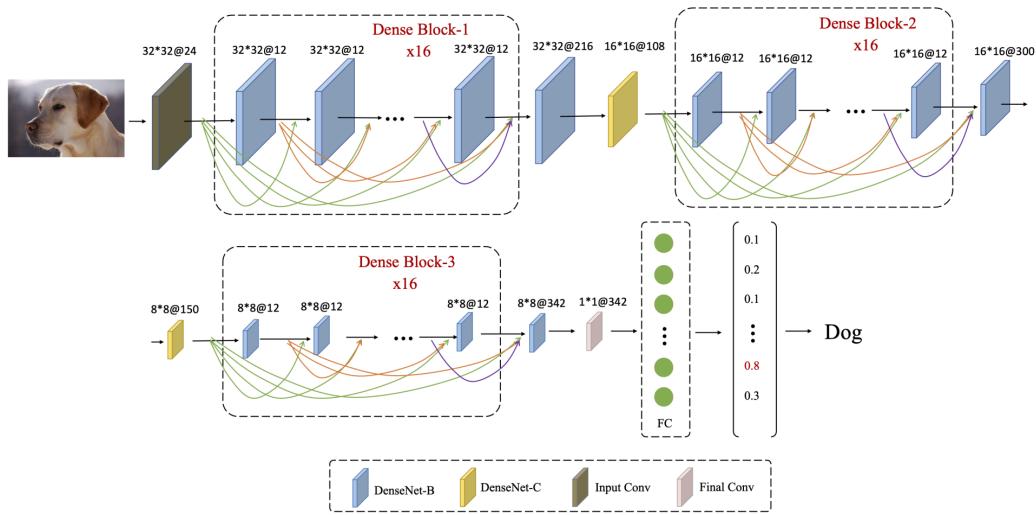


Figure 1: A 5-layer dense block with a growth rate of $k = 4$.
Each layer takes all preceding feature-maps as input.

- **Growth rate:** 如果每个 H_l 函数产生 k 个 feature-maps, 那么第 l 层产生的 $k_0 + k(l - 1)$ 个 feature-maps。 k_0 是输入层的通道数。将超参数 k 称为网络的增长率。 k 表示每个 dense block 中每层输出的 feature map 个数。
- **DenseNet Basic Composition Layer:** BatchNorm(BN)–ReLU–3*3 Conv.
- **DenseNet-B(Bottleneck Layers):** 在 BN–ReLU–3*3 Conv 之前进行 BN–ReLU–1*1 Conv 操作, 减少 feature maps size。
- **Transition Layer(过渡层):** 采用 1*1 Conv 和 2*2 平均池化作为相邻 Dense Block 之间的转换层, 减少 feature map 数和缩小 feature map size, size 指 width*height。在相邻 Dense Block 中输出的 feature map size 是相同的, 以便它们能够很容易的连接在一起。
- **DenseNet-BC:** 如果 Dense Block 包含 m 个 feature-maps, 则 Transition Layer 生成 θm 输出 feature maps, 其中 $0 < \theta < 1$ 称为压缩因子。当 $\theta = 1$ 时, 通过 Transition Layers 的 feature-maps 数保持不变。当 $\theta < 1$ 时, 称为 DenseNet-C, 在实验中 $\theta = 0.5$ 。当同时使用 Bottleneck 和 $\theta < 1$ 的 Transition Layers 时, 称为 DenseNet-BC。



下图是一个 DenseNet 结构图，来自于论文：使用了 3 个 Dense Blocks。

DenseNet 由多个 Desne Block 组成。每个 Dense Block 中的 feature-map size 相同。两个相邻 Dense Block 之间的层称为 Transition Layers。通过卷积和池化来更改 feature-map size。

Denseblock :

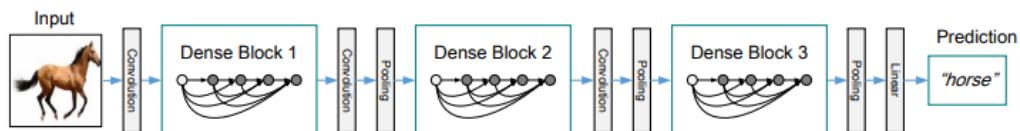
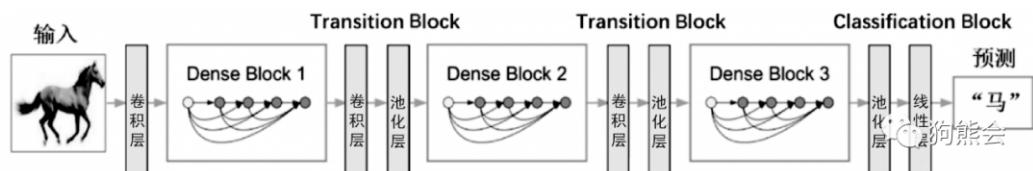


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.



向前：每一层都可以看到所有的之前的输入，对于网络已经学到的『知识』（即已有 feature map），以及原始输入，都可以直接 access 到，然后再添加自己的『知识』到全局知识库。鼓励了特征的重用，特征重用就可以减少不必要的计算量。另外，多层之间可以很好地进行交互，每一层都接受前面所有层的输出，具有多层特征融合的特性；**向后**：跳跃结构，可以很近地连接到最后的 loss，训练起来很容易，直接接受最终 loss 的监督，深层监督，解决梯度消失的问题，并且，能起到正则化的作用缓解过拟合；但是这样的结构，有一个明显的缺点，就是特征会爆炸，每一层的输入都连接到后面的所有层，这样特征越来越多越来越多，作者用一个增长率的概念来量化这个过程，定义为每一层输出的 feature map 数量，即每一层增加的 feature map 数量。因为特征数量会快速增加，所以作者用了很『轻』的层，不想通常的每层输出上百个通道，DenseNet 每一层只输出比如 4、16 这样很少的层，每一层都很轻。并且，每一层也使用了 1×1 来降维。

同一个 dense blocks 中要求 feature size 保持相同大小，在不同 dense blocks 之间设置 **transition layers**（卷积和池化来改变特征地图的大小）实现降维。dense Blocks 内部必须特征图大小一致，每层的输入是 concat 连接，而不是 ResNet 的 element-wise 连接，内部的每一个节点代表 BN+ReLU+Conv（参考 ResNet V2），每个卷积层都是 $3 \times 3 \times k$ 的 filter，其中 k 被称为 growth rate。**Transition layers** 中包含的 Pooling 层会改变特征图的大小。若每个 Dense Block 有 12 层，输入到该 block 的 feature map 数为 16， $k=12$ ，则第一个 Dense Block 所有输出的 concat 起来的 feature map 数是 $16+12 \times 12 = 160$ ，*transition layer* 节点由 BN-Conv-Pool 组成，卷积由 11 构成，num_out 数保持和输入一致，第二、三个的输出 feature map 数量分别是 $160+12 \times 12 = 304$ ， $304+12 \times 12 = 448$ 。

如前所示，DenseNet 的网络结构主要由 DenseBlock 和 Transition 组成，如图 5 所示。下面具体介绍网络的具体实现细节。

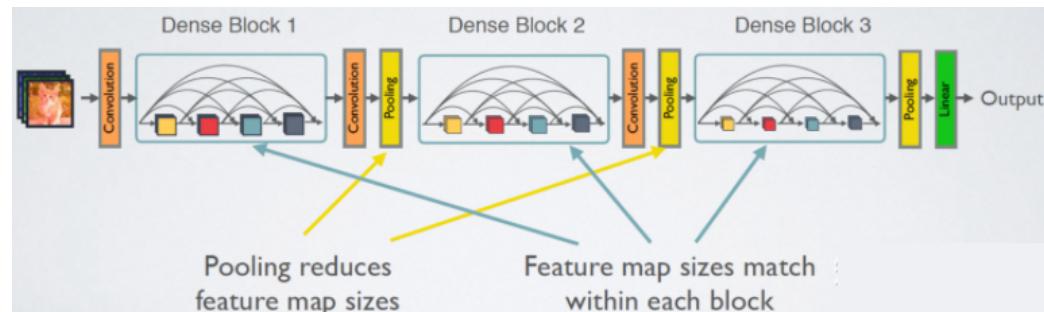


图 6 DenseNet 的网络结构

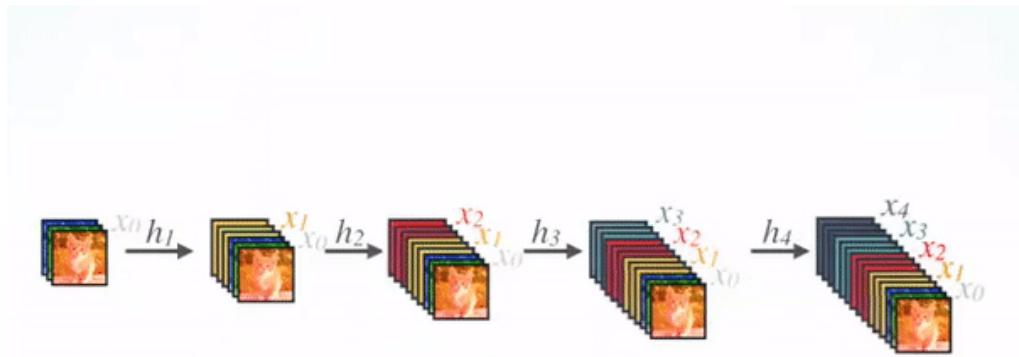
在 DenseBlock 中，各个层的特征图大小一致，可以在 channel 维度上连接。

DenseBlock 中的非线性组合函数 $H_l(\cdot)$ 采用的是 BN+ReLU+3x3 Conv 的结构，如图 6 所示。另外值得注意的一点是，与 ResNet 不同，所有 DenseBlock 中各个层卷积之后均输出 k 个特征图，即得到的特征图的 channel 数为 k ，或者说采用 k 个卷积核。 k 在 DenseNet 称为 growth rate，这是一个超参数。一般情况下使用较小的 k （比如 12），就可以得到较佳的性能。假定输入层的特征图的 channel 数为 k_0 。那么 l 层输入的 channel 数为 $k_0 + k(l - 1)$ ，因此随着层数增加，尽管 k 定得较小，DenseBlock 的输入会非常多，不过这是由于特征重用所造成的，每个层仅有 k 个特征是自己独有的。



前向传播中的级联

基础 DenseNet 组成层



对于每个组成层使用 Pre-Activation Batch Norm (BN) 和 ReLU , 然后用 k 通道的输出特征映射进行 3×3 卷积 , 例如 , 将 x_0 、 x_1 、 x_2 、 x_3 转换为 x_4 。这是 Pre-Activation ResNet 的想法。

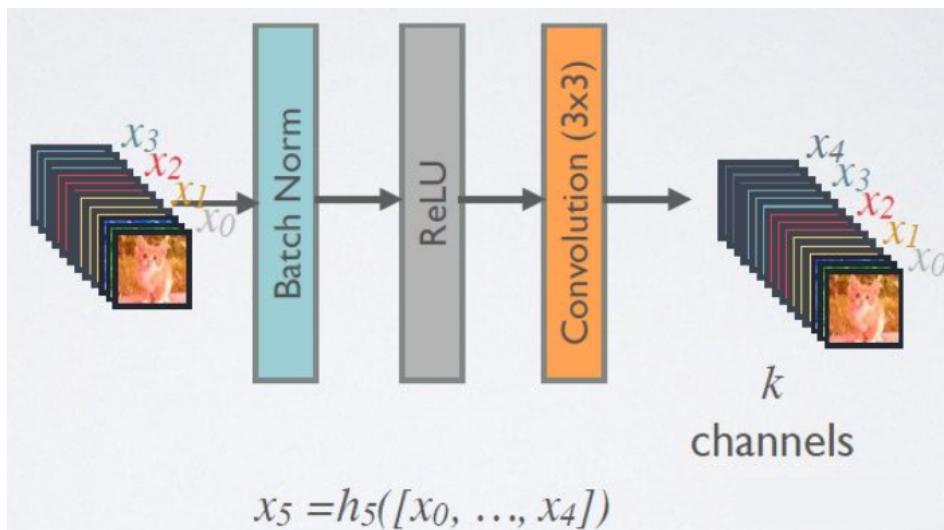


图 6 DenseBlock 中的非线性转换结构

由于后面层的输入会非常大 , DenseBlock 内部可以采用 bottleneck 层来减少计算量 , 主要是原有的结构中增加 1×1 Conv , 如图 7 所示 , 即 BN+ReLU+ 1×1 Conv+BN+ReLU+ 3×3 Conv , 称为 DenseNet-B 结构。其中 1×1 Conv 得到 $4k$ 个特征图它起到的作用是降低特征数量 , 从而提升计算效率。

DenseNet-B (Bottleneck 层)

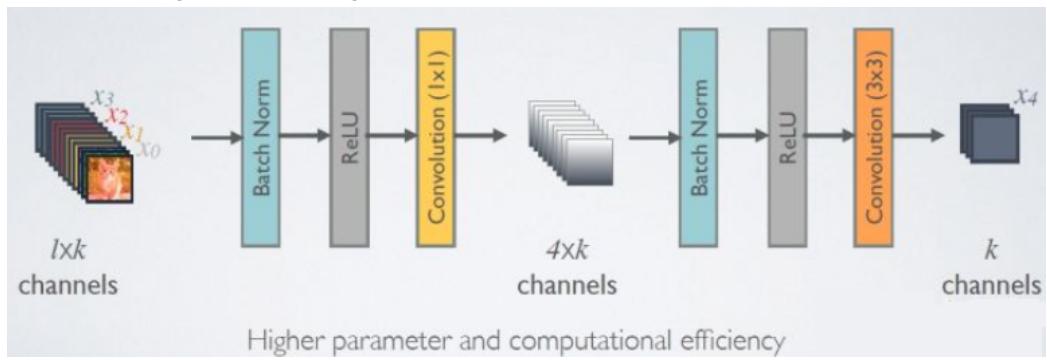
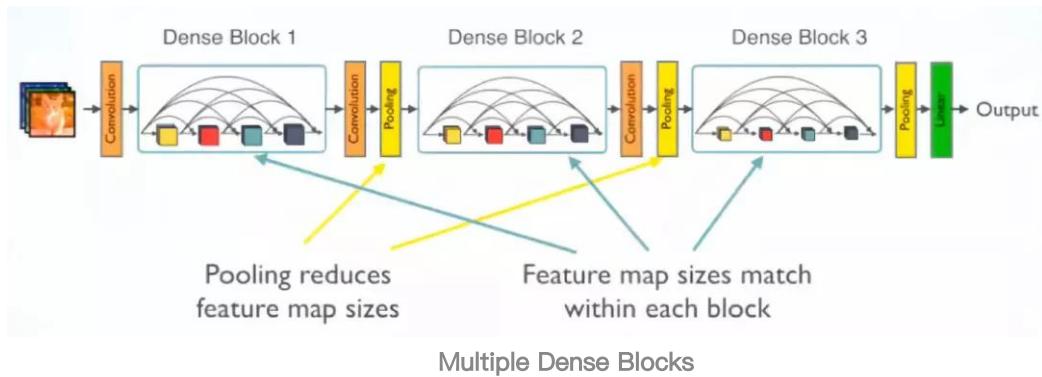


图 7 使用 bottleneck 层的 DenseBlock 结构

为了降低模型的复杂度和规模，在 BN-ReLU- 3×3 conv 之前进行了 BN-ReLU- 1×1 conv.

具有转换层 (transition layer) 的多 Dense 块



采用 1×1 Conv 和 2×2 平均池化作为相邻 dense block 之间的转换层。特征映射大小在 dense block 中是相同的，因此它们可以很容易地连接在一起。在最后一个 dense block 的末尾，执行一个全局平均池化，然后附加一个 Softmax 分类器。

对于 Transition 层，它主要是连接两个相邻的 DenseBlock，并且降低特征图大小。

Transition 层包括一个 1×1 的卷积和 2×2 的 AvgPooling，结构为 BN+ReLU+ 1×1 Conv+ 2×2 AvgPooling。

DenseNet-BC (进一步压缩)

另外，Transition 层可以起到压缩模型的作用。假定 Transition 的上接 DenseBlock 得到的特征图 channels 数为 m ，Transition 层可以产生 $\lfloor \theta m \rfloor$ 个特征（通过卷积层），其中 $\theta \in (0, 1]$ 是压缩系数 (compression rate)。当 $\theta = 1$ 时，特征个数经过 Transition 层没有变化，即无压缩，而当压缩系数小于 1 时，这种结构称为 DenseNet-C，文中使用 $\theta = 0.5$ 。对于使用 bottleneck 层的 DenseBlock 结构和压缩系数小于 1 的 Transition 组合结构称为 DenseNet-BC。

DenseNet 共在三个图像分类数据集 (CIFAR , SVHN 和 ImageNet) 上进行测试。对于前两个数据集，其输入图片大小为 32×32 ，所使用的 DenseNet 在进入第一个 DenseBlock 之前，首先进行一次 3×3 卷积 (stride=1)，卷积核数为 16 (对于 DenseNet-BC 为 $2k$)。DenseNet 共包含三个 DenseBlock，各个模块的特征图大小分别为 32×32 ， 16×16 和 8×8 ，每个 DenseBlock 里面的层数相同。最后的 DenseBlock 之后是一个 global AvgPooling 层，然后送入一个 softmax 分类器。注意，在 DenseNet 中，所有的 3×3 卷积均采用 padding=1 的方式以保证特征图大小维持不变。对于基本的 DenseNet，使用如下三种网络配置： $\{L = 40, k = 12\}$ ，

$\{L = 100, k = 12\}$ ， $\{L = 40, k = 24\}$ 。而对于 DenseNet-BC 结构，使用如下三种网络配置： $\{L = 100, k = 12\}$ ， $\{L = 250, k = 24\}$ ， $\{L = 190, k = 40\}$ 。这里的 L 指的是网络总层数 (网络深度)，一般情况下，我们只把带有训练参数的层算入其中，而像 Pooling 这样的无参数层不纳入统计中，此外 BN 层尽管包含参数但是也不单独统计，而是可以计入它所附属的卷积层。对于普通的 $L = 40, k = 12$ 网络，除去第一个卷积层、2 个 Transition 中卷积层以及最后的 Linear 层，共剩余 36 层，均分到三个 DenseBlock 可知每个 DenseBlock 包含 12 层。其它的网络配置同样可以算出各个 DenseBlock 所含层数。对于 ImageNet 数据集来说，图片输入大小为 224×224 ，网络结构采用包含 4 个 DenseBlock 的 DenseNet-BC，其首先是一个 stride=2 的

7x7 卷积层(卷积核数为 $2k$)，然后是一个 stride=2 的 3x3 MaxPooling 层，后面才进入 DenseBlock。ImageNet 数据集所采用的网络配置如表 1 所示：

| Layers | Output Size | DenseNet-121 | DenseNet-169 | DenseNet-201 | DenseNet-264 |
|----------------------|-------------|--|--|--|--|
| Convolution | 112 × 112 | | 7 × 7 conv, stride 2 | | |
| Pooling | 56 × 56 | | 3 × 3 max pool, stride 2 | | |
| Dense Block (1) | 56 × 56 | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 6$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 6$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 6$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 6$ |
| Transition Layer (1) | 56 × 56 | | 1 × 1 conv | | |
| Dense Block (2) | 28 × 28 | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 12$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 12$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 12$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 12$ |
| Transition Layer (2) | 28 × 28 | | 1 × 1 conv | | |
| Dense Block (3) | 14 × 14 | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 24$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 32$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 48$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 64$ |
| Transition Layer (3) | 14 × 14 | | 1 × 1 conv | | |
| Dense Block (4) | 7 × 7 | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 16$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 32$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 32$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 48$ |
| Classification Layer | 1 × 1 | | 7 × 7 global average pool | $\left[\begin{array}{l} \text{COPY} \\ \text{---} \end{array} \right]$ | 1000D fully-connected, softmax |

表 1 ImageNet 数据集上所采用的 DenseNet 结构

DenseNet-121、DenseNet-169 等中的数字 121、169 是如何计算出来的：以 121 为例，1 个卷积(Convolution)+6 个 Dense Block*2 个卷积(1*1、3*3)+1 个 Transition Layer(1*1 conv)+12 个 Dense Block*2 个卷积(1*1、3*3)+1 个 Transition Layer(1*1 conv)+24 个 Dense Block*2 个卷积(1*1、3*3)+1 个 Transition Layer(1*1 conv)+16 个 Dense Block*2 个卷积(1*1、3*3)+最后的 1 个全连接层=121。这里的层仅指卷积层和全连接层，其它类型的层并没有计算在内。

| Layers | Output Size | DenseNet-121($k = 32$) | DenseNet-169($k = 32$) | DenseNet-201($k = 32$) | DenseNet-161($k = 48$) |
|----------------------|-------------|--|--|--|--|
| Convolution | 112 × 112 | | 7 × 7 conv, stride 2 | | |
| Pooling | 56 × 56 | | 3 × 3 max pool, stride 2 | | |
| Dense Block (1) | 56 × 56 | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 6$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 6$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 6$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 6$ |
| Transition Layer (1) | 56 × 56 | | 1 × 1 conv | | |
| Dense Block (2) | 28 × 28 | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 12$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 12$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 12$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 12$ |
| Transition Layer (2) | 28 × 28 | | 2 × 2 average pool, stride 2 | | |
| Dense Block (3) | 14 × 14 | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 24$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 32$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 48$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 36$ |
| Transition Layer (3) | 14 × 14 | | 1 × 1 conv | | |
| Dense Block (4) | 7 × 7 | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 16$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 32$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 32$ | $\left[\begin{array}{l} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{array} \right] \times 24$ |
| Classification Layer | 1 × 1 | | 7 × 7 global average pool | | 1000D fully-connected, softmax |

Table 1. DenseNet architectures for ImageNet. The growth rate for the first 3 networks is $k = 32$, and $k = 48$ for DenseNet-161. Note that each “conv” layer shown in the table corresponds to the sequence BN-ReLU-Conv.

这个 Table1 就是整个网络的结构图。这个表中的 $k=32$, $k=48$ 中的 k 是 growth rate, 表示每个 dense block 中每层输出的 feature map 个数。为了避免网络变得很宽, 作者都是采用较小的 k , 比如 32 这样, 作者的实验也表明小的 k 可以有更好的效果。根据 dense block 的设计, 后面几层可以得到前面所有层的输入, 因此 concat 后的输入 channel 还是比较大的。另外这里每个 dense block 的 3*3 卷积前面都包含了一个 1*1 的卷积操作, 就是所谓的 bottleneck layer, 目的是减少输入的 feature map 数量, 既能降维减少计算量, 又能融合各个通道的特征, 何乐而不为。另外作者为了进一步压缩参数, 在每两个 dense block 之间又增加了 1*1 的卷积操作。因此在后面的实验对比中, 如果你看到 DenseNet-C 这个网络, 表示增加了这个 Translation layer, 该层的 1*1 卷积的输出 channel 默认是输入 channel 到一半。如果你看到 DenseNet-BC 这个网络, 表示既有 bottleneck layer, 又有 Translation layer。

再详细说下 bottleneck 和 transition layer 操作。在每个 Dense Block 中都包含很多个子结构，以 DenseNet-169 的 Dense Block (3) 为例，包含 32 个 1*1 和 3*3 的卷积操作，也就是第 32 个子结构的输入是前面 31 层的输出结果，每层输出的 channel 是 32 (growth rate)，那么如果不做 bottleneck 操作，第 32 层的 3*3 卷积操作的输入就是 31×32 (上一个 Dense Block 的输出 channel)，近 1000 了。而加上 1*1 的卷积，代码中的 1*1 卷积的 channel 是 $growth\ rate \times 4$ ，也就是 128，然后再作为 3*3 卷积的输入。这就大大减少了计算量，这就是 bottleneck。至于 transition layer，放在两个 Dense Block 中间，是因为每个 Dense Block 结束后的输出 channel 个数很多，需要用 1*1 的卷积核来降维。还是以 DenseNet-169 的 Dense Block (3) 为例，虽然第 32 层的 3*3 卷积输出 channel 只有 32 个 (growth rate)，但是紧接着还会像前面几层一样有通道的 concat 操作，即将第 32 层的输出和第 32 层的输入做 concat，前面说过第 32 层的输入是 1000 左右的 channel，所以最后每个 Dense Block 的输出也是 1000 多的 channel。因此这个 transition layer 有个参数 reduction (范围是 0 到 1)，表示将这些输出缩小到原来的多少倍，默认是 0.5，这样传给下一个 Dense Block 的时候 channel 数量就会减少一半，这就是 transition layer 的作用。文中还用到 dropout 操作来随机减少分支，避免过拟合，毕竟这篇文章的连接确实多。

假如输入图像大小为 $n \times n$ ，过滤器(filter)为 $f \times f$ ，padding 为 p ，步长(stride)为 s ，则输出大小为：计算卷积层大小，如果商不是整数，向下取整，即 floor 函数；计算池化层大小，如果商不是整数，向上取整，即 ceil 函数。参考：

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor * \left\lceil \frac{n+2p-f}{s} + 1 \right\rceil$$

这里描述下 DenseNet-121 架构：k=32，与上表中“DenseNet-121”有所差异

(1) Feature Block 的计算。DenseNet 的第一部分为 Feature Block，它介于输入层和第一个 Dense Block 之间，包含卷积和池化的过程。输入图像是 224 像素×224 像素×3，使用 64 个 7×7 的卷积核进行 same 卷积，步长是 2，此时输出为 $112 \times 112 \times 64$ ，进行 Batch Normalization 操作并使用 ReLU 函数激活，通过一次池化，最终输出的结果为 $56 \times 56 \times 64$ 。

Feature Block 的计算过程如下。

- ① 输入为图像 ($224 \times 224 \times 3$)。
- ② 卷积为 7×7 (conv) $\times 64$, stride=2, 输出为 $112 \times 112 \times 64$, 参数个数为 $7 \times 7 \times 3 \times 64 + 64$ 。
- ③ Batch Normalization 计算。
- ④ 激活函数 ReLU 计算。
- ⑤ 池化为 3×3 (max pool), stride=2, 输出为 $56 \times 56 \times 64$ 。

(2) Dense Block 内部的计算。在每一个 Dense Block 的内部，每一层的输入是前面所有层输出的拼接，这里的拼接是指通道层面上的拼接。例如，将一个 $56 \times 56 \times 64$ 的数据和一个 $56 \times 56 \times 32$ 的数据拼接在一起，结果就是 $56 \times 56 \times 96$ ，这里的 96 是 64 和 32 的和。定义等于 growth rate (增长率)，表示每一层的输出都是一个确定的通道数。以第一个 Dense Block 的第一层为例，输入为前面 Feature Block 的输出，即 $56 \times 56 \times 64$ ，经过 Batch Normalization 和 ReLU，进入 Bottleneck 层，该层是可选层，其目的是减少 feature-maps 的数量。在该层进行 1×1 的卷积，通道数目设定为，这里取 = 32，因此就是 $32 \times 4 = 128$ ，若输出大于 128，则经过 Bottleneck 层之后，通道数都变为 128。最后进入卷积层，因为卷积核大小为 3×3 ，输出通道数 = 32，所以经过第一个 Dense Block 之后，最终的输出为 $56 \times 56 \times 32$ 。

例如，第一个 Dense Block 中第一层的计算过程如下。

- ① 输入为 Feature Block 的输出 $56 \times 56 \times 64$ 或者上一层 Dense Layer 的输出。
- ② Batch Normalization 的输出为 $56 \times 56 \times 64$ 。
- ③ ReLU 的输出为 $56 \times 56 \times 64$ 。
- ④ Bottleneck 为可选层，为了减少 feature-maps 的数量。采用一个 1×1 的卷积核，则输出为 $56 \times 56 \times 128$ 。对大于的，则都变成，参数个数为 $1 \times 1 \times 64 \times 128 + 128$ 。
- ⑤ 卷积层，采用一个 3×3 的卷积核，输出为 $56 \times 56 \times 32$ ，参数个数为 $3 \times 3 \times 128 \times 32 + 32$ 。总参数个数为 $1 \times 1 \times 64 \times 128 + 128 + 3 \times 3 \times 128 \times 32 + 32 = 45, 216$ 。

最后将 Dense Block 的结构总结如下，这里需要注意的是，每层的输出都不变，而输入通道数都在增加，因为根据 Dense Block 的设计，每层的输入是前面所有层的拼接，Dense Block 由 L 层 Dense Layer 组成。

- layer0: 输入 $(56 \times 56 \times 64) \rightarrow$ 输出 $(56 \times 56 \times 32)$ 。
layer1: 输入 $(56 \times 56 \times (32 \times 1)) \rightarrow$ 输出 $(56 \times 56 \times 32)$ 。
layer2: 输入 $(56 \times 56 \times (32 \times 2)) \rightarrow$ 输出 $(56 \times 56 \times 32)$ 。
.....
layerL: 输入 $(56 \times 56 \times (32 \times L)) \rightarrow$ 输出 $(56 \times 56 \times 32)$ 。

(3) Transition Block 的计算。Transition Block 介于两个 Dense Block 之间，起连接作用，由一个卷积层和一个池化层组成。它的计算过程如下。

- ① 输入为 Dense Block 的输出为 $56 \times 56 \times 32$ 。
- ② Batch Normalization 的输出为 $56 \times 56 \times 32$ 。
- ③ ReLU 的输出为 $56 \times 56 \times 32$ 。
- ④ Bottleneck 为可选层，采用一个 1×1 的卷积核，此处可以根据预先设定的压缩系数 θ ($0 \sim 1$) 对进行压缩，以减小参数，输出为 $56 \times 56 \times (32 \times \theta)$ ，参数个数为 $1 \times 1 \times 32 \times 32 \times \theta + 32 \times \theta$ 。
- ⑤ 池化层，采用 2×2 的平均值池化，输出为 $28 \times 28 \times (32 \times \theta)$ 。

至此介绍了 DenseNet 中每一个单独的层。循环进行 Dense Block 层和 Transition Block 层，最后是 Classification 层，可以得到一个完整的 DenseNet 神经网络结构，总结如下。

循环 Dense Block 和 Transition 层。

DenseBlock 1: 输入为 $56 \times 56 \times 64$ ，输出为 $56 \times 56 \times 32$ 。

Transition 1: 输入为 $56 \times 56 \times 32$ ，输出为 $56 \times 56 \times 32$ 。

DenseBlock 2: 输入为 $28 \times 28 \times 32$ ，输出为 $28 \times 28 \times 32$ 。

Transition 2: 输入为 $28 \times 28 \times 32$ ，输出为 $14 \times 14 \times 32$ 。

DenseBlock 3: 输入为 $14 \times 14 \times 32$ ，输出为 $14 \times 14 \times 32$ 。

Transition 3: 输入为 $14 \times 14 \times 32$ ，输出为 $7 \times 7 \times 32$ 。

DenseBlock 4: 输入为 $7 \times 7 \times 32$ ，输出为 $7 \times 7 \times 32$ 。

Classification 层。

输入为 Dense Block4 的输出，即 $7 \times 7 \times 32$ 。

Batch Normalization 的输出为 $7 \times 7 \times 32$ 。

ReLU 的输出为 $7 \times 7 \times 32$ 。

池化层采用大小为 7×7 ，步长为 1 的平均值池化，输出为 $1 \times 1 \times 32$ 。

Flatten 的输出为 1×32 。

全连接层的输出为 $1 \times 1, 000$ 。

三、实验结果

(1) 数据集

CIFAR: 两个数据集 CIFAR-10 和 CIFAR-100，均由 32×32 像素的彩色自然图像组成。CIFAR-10 (C10) 包含 10 个类别的图像，CIFAR-100 (C100) 包含 100 个类别的图像。训练集和测试集分别包含 50,000 张和 10,000 张图像，我们保留 5,000 张训练图像作为验证集。我们采用了标准的数据增强方案（镜像/移位），该方案广泛用于这两个数据集。我们在数据集名称（例如 C10 +）的末尾用“+”号表示此数据增强方案。对于预处理，我们使用通道平均值和标准偏差对数据进行归一化。对于最终运行，我们使用所有 50,000 张训练图像，并在训练结束时报告最终测试错误。

SVHN: 街景门牌号码 (SVHN) 数据集包含 32×32 彩色数字图像。训练集中有 73257 张图像，测试集中有 26032 张图像，而其他训练有 531131 张图像。按照常规做法，我们使用所有训练数据而没有做任何数据增强，并从训练集中分配出 6,000 张图像的验证集。我们选择训练期间验证误差最小的模型，并报告测试误差。我们遵循《Wide residual networks》并将像素值除以 255，因此它们在 [0,1] 范围内。

ImageNet: ILSVRC 2012 分类数据集包含来自 1,000 个类别的 120 万张用于训练的图像和 50,000 张用于验证的图像。我们采用与《Training and investigating residual nets》中相同的数据增强方案来训练图像，并在测试时应用大小为 224×224 的 single-crop 或 10-crop。按照《Deep residual learning for image recognition》中的方式，我们报告了验证集上的分类错误。

(2) 训练

所有网络均使用随机梯度下降 (SGD) 进行训练。在 CIFAR 和 SVHN 上，我们分别设置 batch_size=64 进行训练 300 和 40 个 epochs。初始学习率设置为 0.1，然后在训练时期总数的 50% 和 75% 阶段，除以 10。在 ImageNet 上，我们训练了 90 个 epochs 的模型，批量大小为 256。最初将学习速率设置为 0.1，然后在第 30 和 60 个 epochs 的阶段将其降低 10 倍。请注意，DenseNet 的简单实施可能会导致内存效率低下。为了减少 GPU 上的内存消耗，请参考我们有关 DenseNets 的内存高效实现的技术报告《Memory-efficient implementation of densenets》。

根据《Training and investigating residual nets》，我们使用 $10 - 4 \cdot 10^{-4}$
 $10 - 4$

的权重衰减和 0.9 的 Nesterov 动量而不进行阻尼。我们采用《Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.》引入的权重初始化。对于没有数据增强的三个数据集，即 C10, C100 和 SVHN，我们在每个卷积层（第一个卷积层除外）之后添加一个 Dropout 层，并将丢失率设置为 0.2。对于每个任务和模型设置，仅对测试错误进行一次评估。

(3)评估

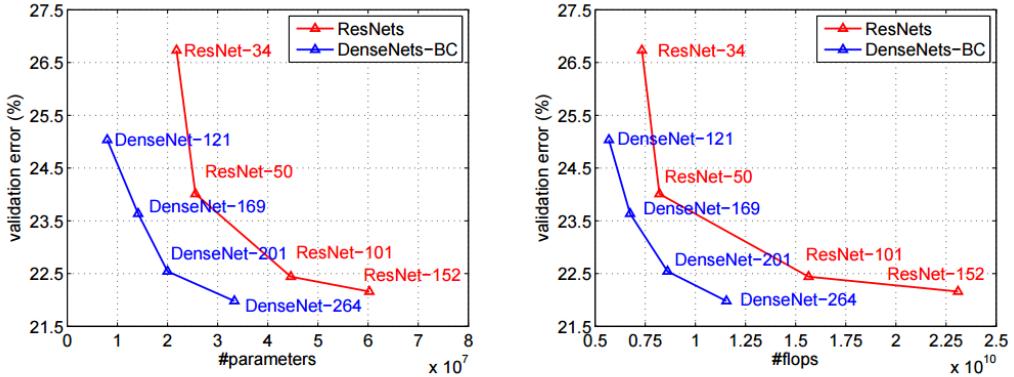


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*). <https://blog.csdn.net/u013841196>

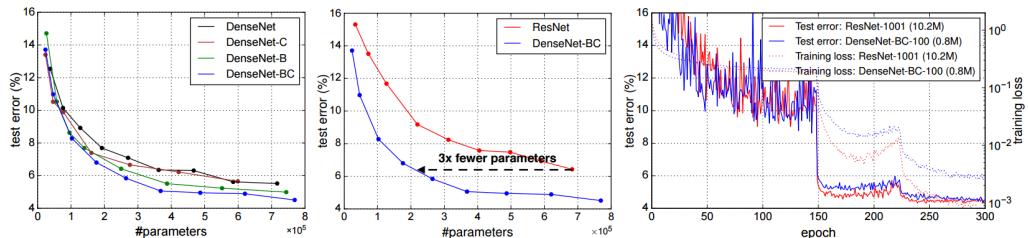


Figure 4: *Left:* Comparison of the parameter efficiency on C10+ between DenseNet variations. *Middle:* Comparison of the parameter efficiency between DenseNet-BC and (pre-activation) ResNets. DenseNet-BC requires about 1/3 of the parameters as ResNet to achieve comparable accuracy. *Right:* Training and testing curves of the 1001-layer pre-activation ResNet [12] with more than 10M parameters and a 100-layer DenseNet with only 0.8M parameters. <https://blog.csdn.net/u013841196>

DenseNet 核心思想在于建立了不同层之间的连接关系，充分利用了 feature，进一步减轻了梯度消失问题，加深网络不是问题，而且训练效果非常好。另外，利用 bottleneck layer, Translation layer 以及较小的 growth rate 使得网络变窄，参数减少，有效抑制了过拟合，同时计算量也减少了。DenseNet 优点很多，而且在和 ResNet 的对比中优势还是非常明显的。

作者在不同数据集上采用的 DenseNet 网络会有一点不一样，比如在 Imagenet 数据集上，DenseNet-BC 有 4 个 dense block，但是在别的数据集上只用 3 个 dense block。其他更多细节可以看论文 3 部分的 Implementation Details。训练的细节和超参数的设置可以看论文 4.2 部分，在 ImageNet 数据集上测试的时候有做 224*224 的 center crop。

Table2 是在三个数据集 (C10, C100, SVHN) 上和其他算法的对比结果。

ResNet[11]就是 kaiming He 的论文，对比结果一目了然。DenseNet-BC 的网络参数和相同深度的 DenseNet 相比确实减少了很多！参数减少除了可以节省内存，还能减少过拟合。这里对于 SVHN 数据集，DenseNet-BC 的结果并没有 DenseNet(k=24)的效果好，作者认为原因主要是 SVHN 这个数据集相对简单，更深的模型容易过拟合。在表格的倒数第二个区域的三个不同深度 L 和 k 的 DenseNet 的对比可以看出随着 L 和 k 的增加，模型的效果是更好的。

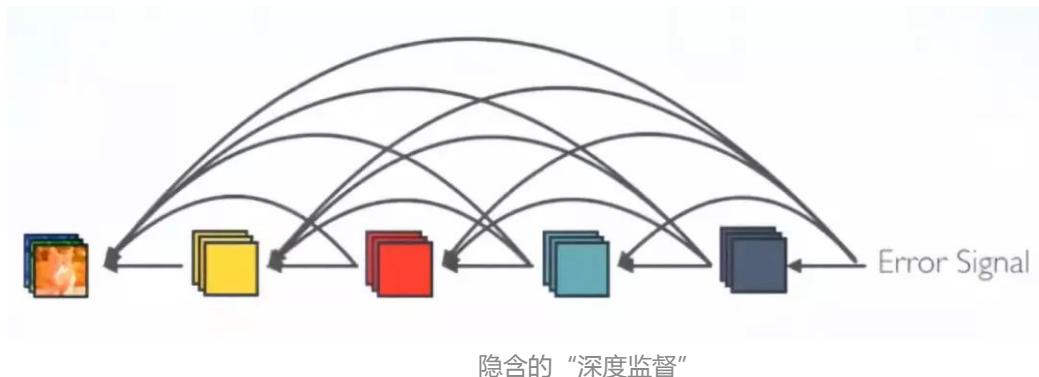
| Method | Depth | Params | C10 | C10+ | C100 | C100+ | SVHN |
|--|-------------------|------------------------|----------------------|-----------------------------|----------------------------|---------------------------------------|-----------------------------|
| Network in Network [22] | - | - | 10.41 | 8.81 | 35.68 | - | 2.35 |
| All-CNN [31] | - | - | 9.08 | 7.25 | - | 33.71 | - |
| Deeply Supervised Net [20] | - | - | 9.69 | 7.97 | - | 34.57 | 1.92 |
| Highway Network [33] | - | - | - | 7.72 | - | 32.39 | - |
| FractalNet [17] with Dropout/Drop-path | 21 21 | 38.6M 38.6M | 10.18 7.33 | 5.22 4.60 | 35.34 28.20 | 23.30 23.73 | 2.01 1.87 |
| ResNet [11] | 110 | 1.7M | - | 6.61 | - | - | - |
| ResNet (reported by [13]) | 110 | 1.7M | 13.63 | 6.41 | 44.74 | 27.22 | 2.01 |
| ResNet with Stochastic Depth [13] | 110 1202 | 1.7M 10.2M | 11.66 - | 5.23 4.91 | 37.80 | 24.58 | 1.75 |
| Wide ResNet [41] with Dropout | 16 28 16 | 11.0M 36.5M 2.7M | - - - | 4.81 4.17 - | - - - | 22.07 20.50 1.64 | - |
| ResNet (pre-activation) [12] | 164 1001 | 1.7M 10.2M | 11.26* 10.56* | 5.46 4.62 | 35.58* 33.47* | 24.33 22.71 | - |
| DenseNet ($k = 12$) DenseNet ($k = 12$) DenseNet ($k = 24$) | 40 100 100 | 1.0M 7.0M 27.2M | 7.00 5.77 5.83 | 5.24 4.10 3.74 | 27.55 23.79 23.42 | 24.42 20.20 19.25 | 1.79 1.67 1.59 |
| DenseNet-BC ($k = 12$) DenseNet-BC ($k = 24$) DenseNet-BC ($k = 40$) | 100 250 190 | 0.8M 15.3M 25.6M | 5.92 5.19 - | 4.51 3.62 3.46 | 24.15 19.64 - | 22.27 17.60 17.18 | 1.76 1.74 - |

Table 2. Error rates (%) on CIFAR and SVHN datasets. L denotes the network depth and k its growth rate. Results that surpass all competing methods are **bold** and the overall best results are **blue**. “+” indicates standard data augmentation (translation and/or mirroring). * indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

<http://blog.csdn.net/u014380165>

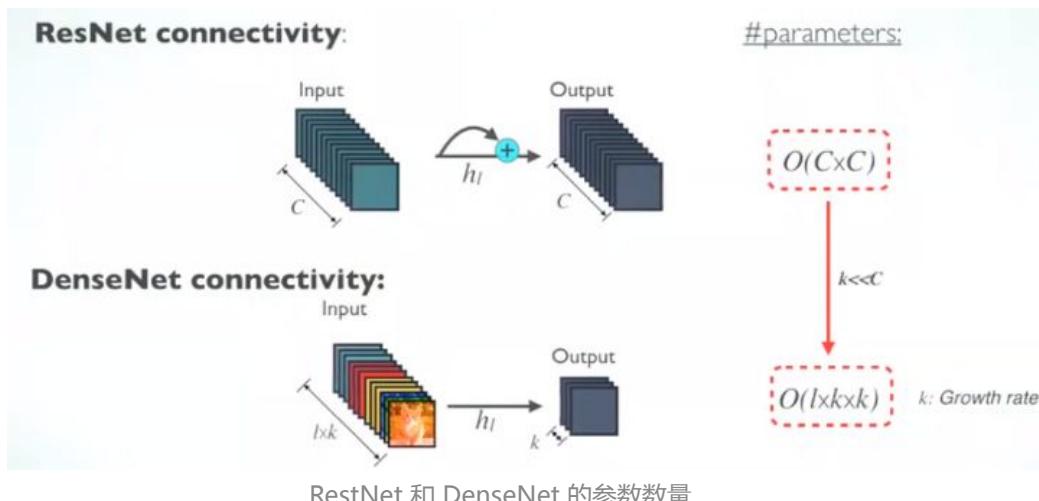
四、DenseNet 的优势

1. 强梯度流



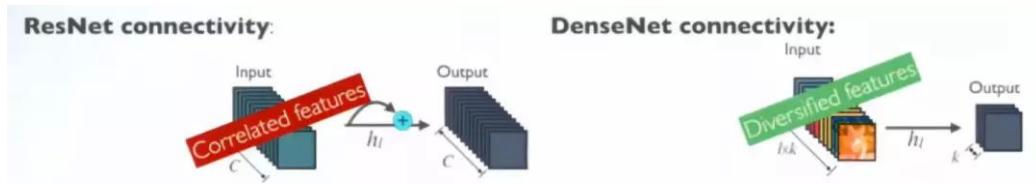
误差信号可以更直接地传播到早期的层中。这是一种隐含的深度监督，因为早期的层可以从最终的分类层直接获得监督。

2. 参数和计算效率



对于每个层，ResNet 中的参数与 $C \times C$ 成正比，而 DenseNet 中的参数与 $1 \times k \times k$ 成正比。由于 $k \ll C$ ，所以 DenseNet 比 ResNet 的 size 更小。

3. 更加多样化的特征

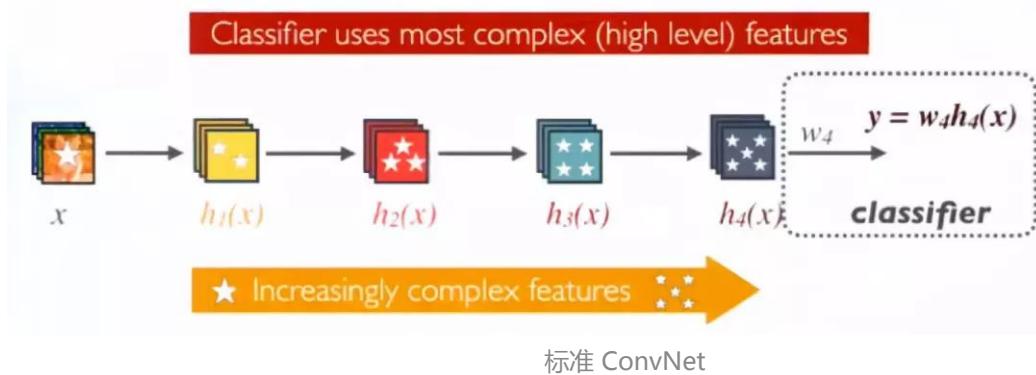


DenseNet 中更加多样化的特征

由于 DenseNet 中的每一层都接收前面的所有层作为输入，因此特征更加多样化，并且倾向于有更丰富的模式。

4. 保持低复杂度特征

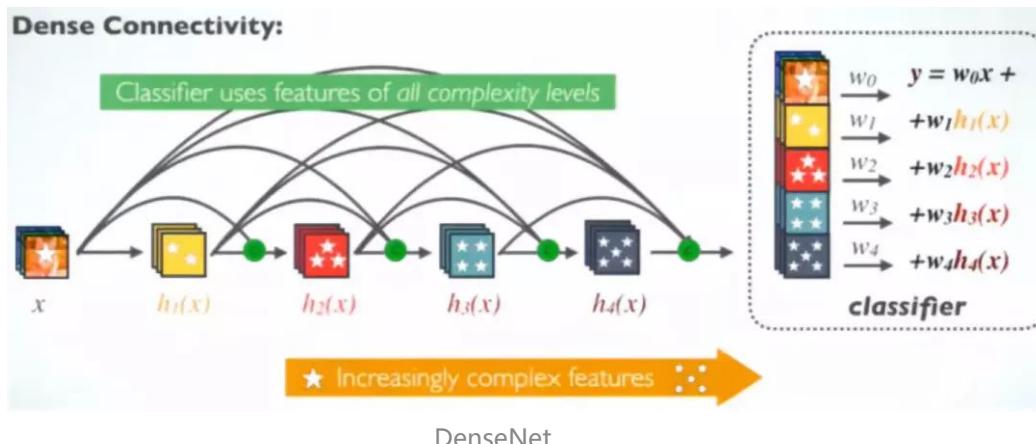
Standard Connectivity:



标准 ConvNet

在标准 ConvNet 中，分类器使用最复杂的特征。

Dense Connectivity:



DenseNet

在 DenseNet 中，分类器使用所有复杂级别的特征。它倾向于给出更平滑的决策边界。它还解释了为什么 DenseNet 在训练数据不足时表现良好。

五、总结分析

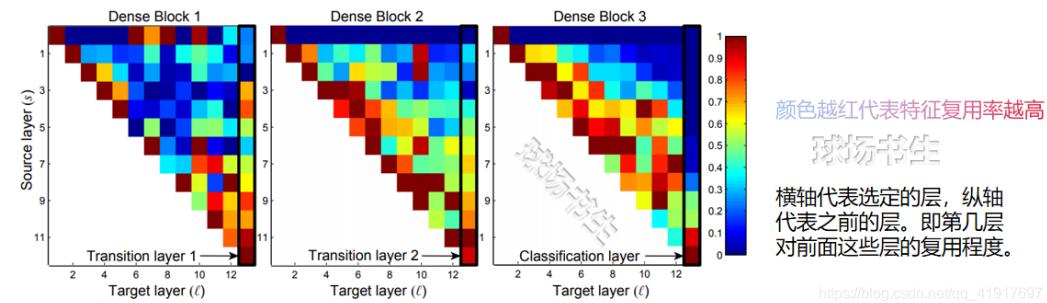
- 1) DenseNet 模型更加简洁，节约参数和计算量。
- 2) DenseNet 具有非常好的抗过拟合性能，尤其适合于训练数据相对匮乏的应用（或者不需要预训练的方法，医学图像，卫星图像），类似于 DenseNet 的网络结构具有更小的泛化误差。神经网络每一层提取的特征都相当于对输入数据的一个非线性变换，而随着深度的增加，变换的复杂度也逐渐增加。相比于一般神经网络的分类器直接依赖于网络最后一层的特征，DenseNet 可以综合利用浅层复杂度低的特征。

3) 隐式深度监督 **implicit Deep supervision**, 因为有 shortcut 的存在, 各层从损失函数

获得额外的监督。我们可以将 DenseNets 理解为一种“深度监督”, 它将分类器附加到每个隐藏层, 强制中间层学习区分特征。

4) 密集连接还具有冗余性吗? 实际上 DenseNet 比其他网络效率更高, 其关键就在于网络每层计算量的减少以及**特征的重复利用**。DenseNet 的每一层只需学习很少的特征, 使得参数量和计算量显著减少。

针对“如此多的密集连接, 是不是全部都是必要的”疑问, 作者给出了实验:



可见, 一些较早层提取出的特征仍可能被较深层直接使用, 即使是 Transition layer 也会使用到之前 Denseblock 中所有层的特征。最后的分类层虽然使用了之前 Denseblock 中的多层信息, 但更偏向于使用最后几个 feature map 的特征, 说明在网络的最后几层是有助于分类的更高层特征。

六、好文

- [CNN深度卷积神经网络-DenseNet](#)
- [综述 : DenseNet—Dense卷积网络 \(图像分类 \)](#)
- [Keras中add和 concatenate 操作的不同](#)
- [DenseNet论文翻译及pytorch实现解析 \(上 \)](#)