

## 《机器学习基础》



## 卷积神经网络模型

# 卷积神经网络模型的发展

---

- ▶ LeNet-5
- ▶ AlexNet-8
- ▶ VGG
- ▶ ResNet
- ▶ InceptionNet (GoogLeNet)

# AlexNet

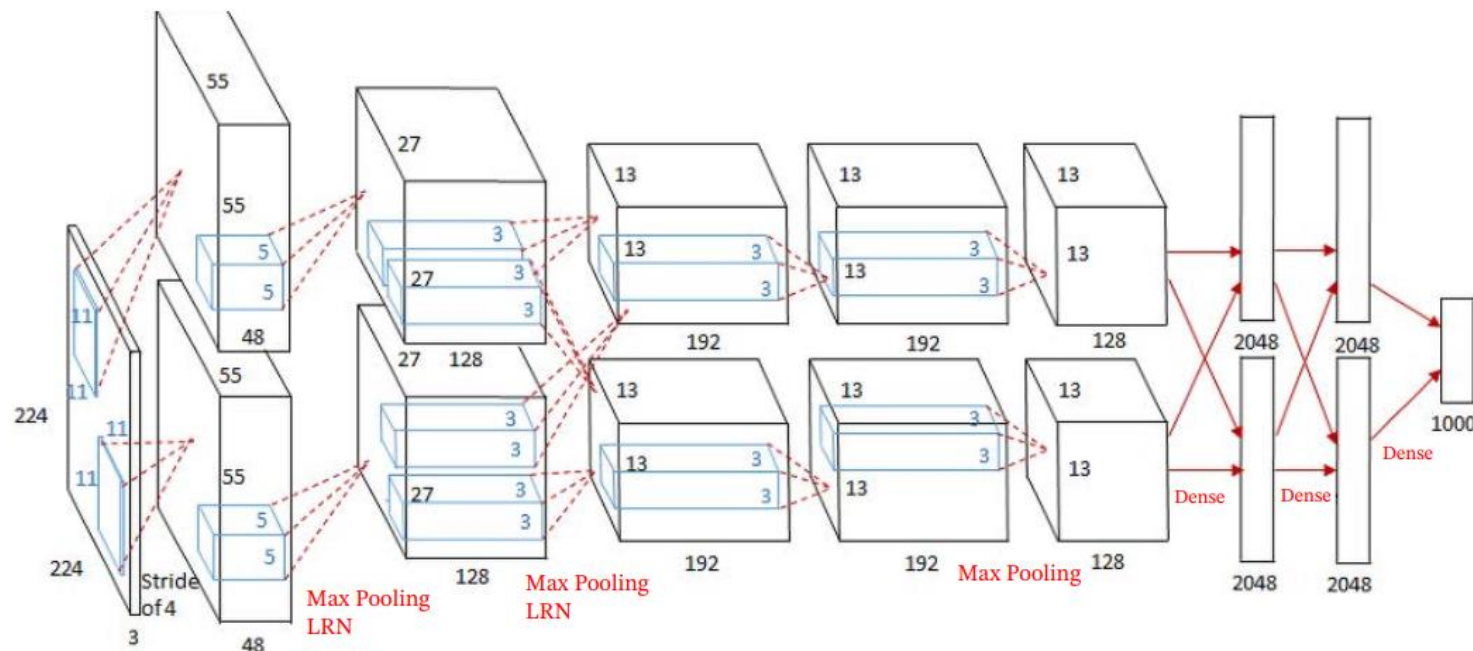
## ► 2012 ILSVRC winner

► (top 5 error of 15.3% compared to runner-up with 26.2% error)

► 第一个现代深度卷积网络模型

► 首次使用了很多现代深度卷积网络的一些技术方法

□ 使用GPU进行并行训练，采用了ReLU作为非线性激活函数，使用Dropout防止过拟合，使用数据增强



# AlexNet

## 经典网络-Alexnet:

[227x227x3] INPUT  
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0  
[27x27x96] MAX POOL1: 3x3 filters at stride 2  
[27x27x96] NORM1: Normalization layer  
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2  
[13x13x256] MAX POOL2: 3x3 filters at stride 2  
[13x13x256] NORM2: Normalization layer  
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1  
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1  
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1  
[6x6x256] MAX POOL3: 3x3 filters at stride 2  
[4096] FC6: 4096 neurons  
[4096] FC7: 4096 neurons  
[1000] FC8: 1000 neurons (class scores)



# AlexNet

---

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax



# AlexNet

---

- ▶在AlexNet神经网络中，整个网络包含大约6000万个参数
- 第1个卷积层（C1）：34,944个参数
- 第2个卷积层（C2）：614,656个参数
- 第3个卷积层（C3）：886,176个参数
- 第4个卷积层（C4）：1,327,488个参数
- 第5个卷积层（C5）：885,952个参数
- 第1个全连接层（FC6）：37,752,832个参数
- 第2个全连接层（FC7）：16,781,312个参数
- 输出层（Output layer）：4,097,000个参数
- 将这些参数数量相加，可以得到AlexNet网络的总参数数量大约是60,964,224个

# AlexNet

---

## ▶ 训练策略

- ▶ 优化算法：带动量的随机梯度下降（SGD with momentum）。
- ▶ 使用的momentum参数设置为0.9。
- ▶ 初始学习率为0.01。
- ▶ AlexNet还采用了学习率衰减的策略，当验证集误差不再下降时除以10，共调整3次，最终学习率0.0001。

## ▶ 数据增强

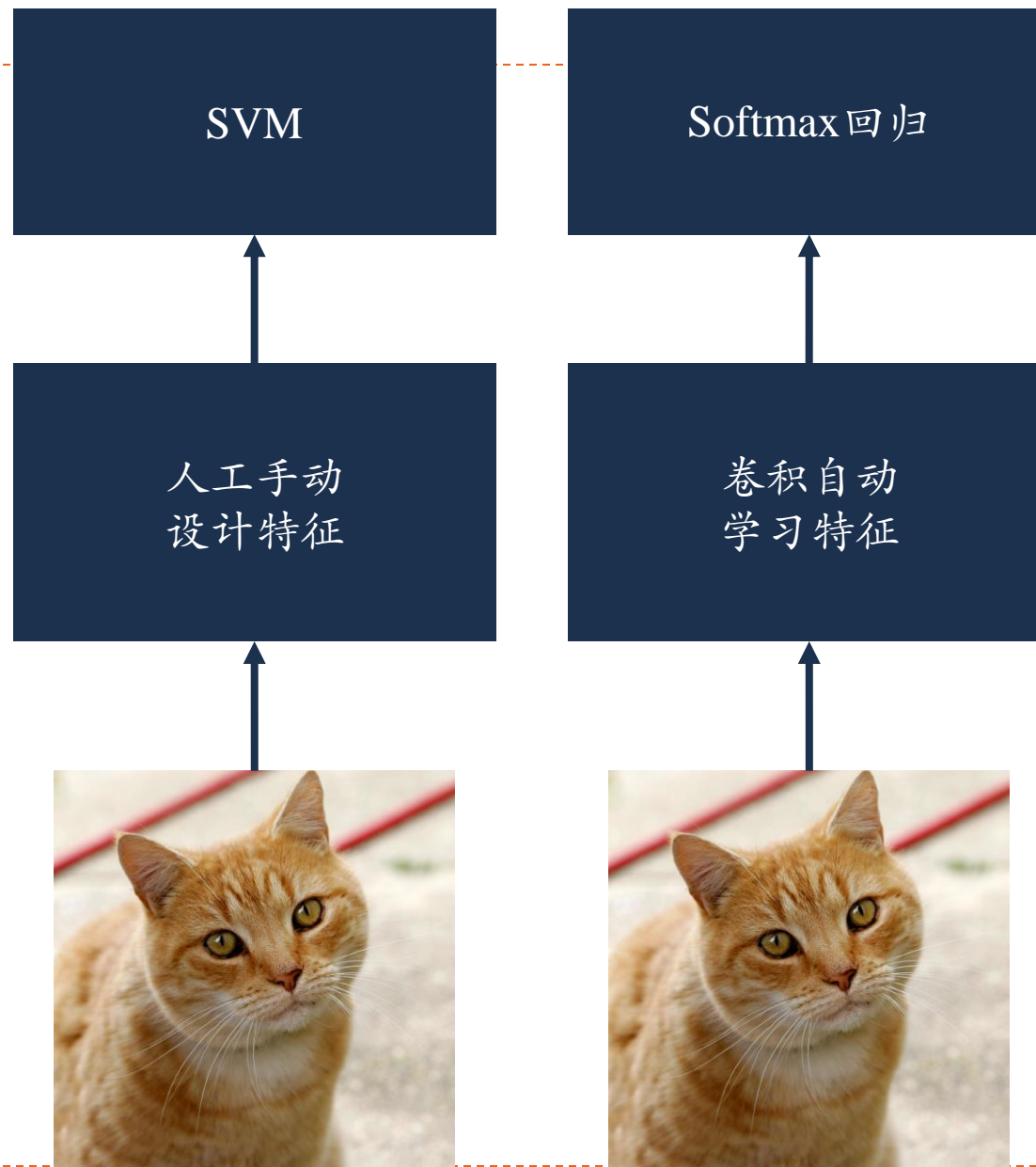
- ▶ 从  $256 \times 256$  图像中随机截取  $227 \times 227$  区域（包括水平翻转）
- ▶ 调整图像 RGB 通道的亮度、对比度和饱和度

## ▶ 训练周期

- ▶ 在 2 块 GTX 580 GPU 上训练约 6 天

# AlexNet

- ▶ AlexNet 在 2012 年赢得了 ImageNet 竞赛
- ▶ 更深更大的 LeNet
- ▶ 计算机视觉的范式转变
- ▶ 主要修改
  - ▶ 丢弃法 (正则化)
  - ▶ ReLu 激活函数 (训练)
  - ▶ 最大池化法



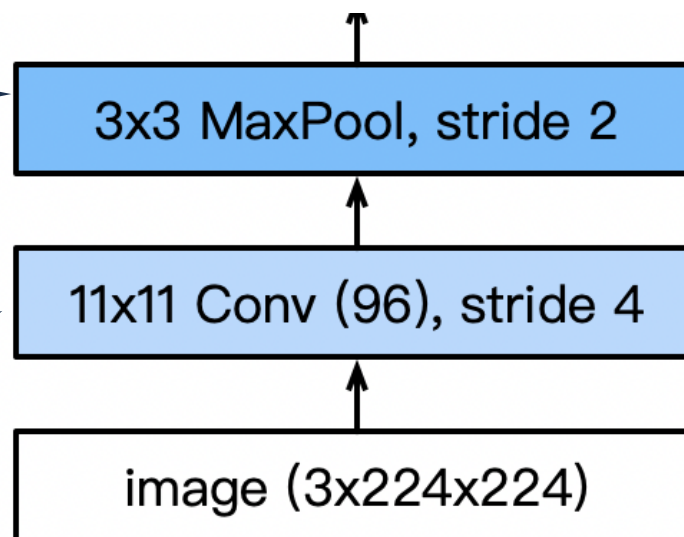


# AlexNet 架构

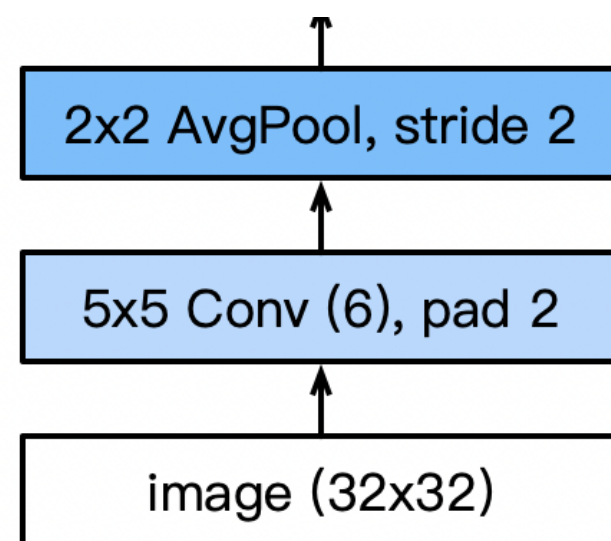
池化窗口更大，更改为最大池化

更大的卷积核，更大的步长，由于图像尺寸增大，输出通道更多。

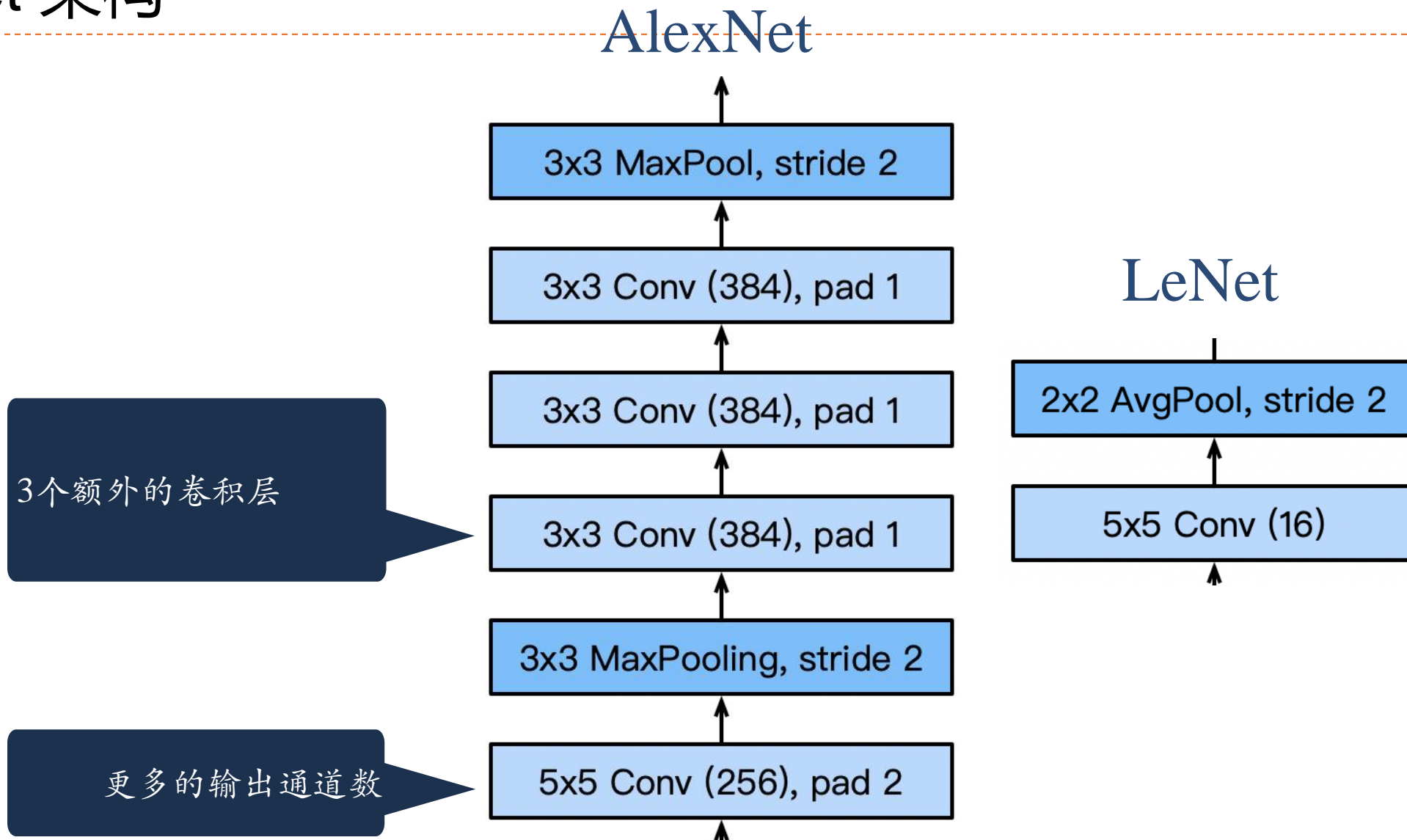
## AlexNet



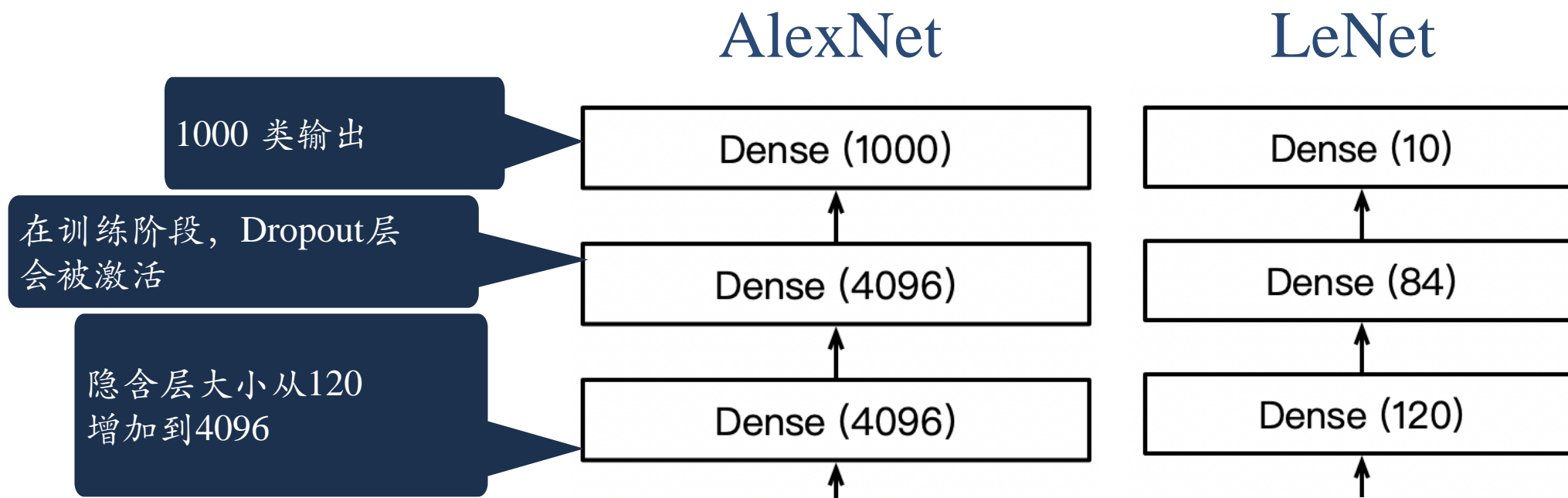
## LeNet



# AlexNet 架构



# AlexNet 架构



# Dropout层

---

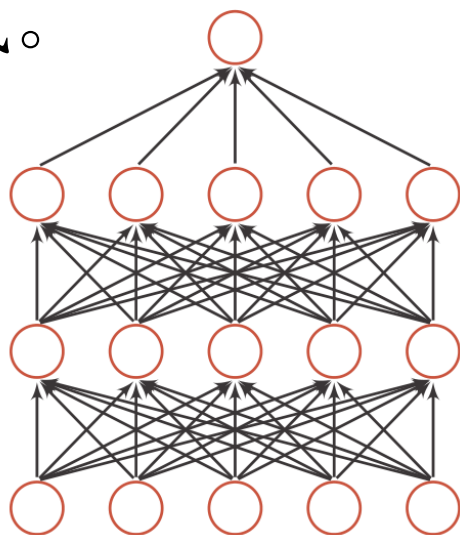
- ▶ 一种常用的正则化技术，由 Geoffrey Hinton 等人在 2012 年提出
- ▶ 用于缓解模型过拟合问题，提高模型的泛化能力
- ▶ 其核心思想是在训练过程中随机“关闭”一部分神经元，打破神经元之间的“共适应”（Co-adaptation）现象。即避免某些神经元过度依赖特定其他神经元的存在，迫使网络学习更通用、不依赖局部结构的特征。
- ▶ 相当于在训练过程中生成多个“简化版”子网络，最终模型可视为这些子网络的集成（类似 Bagging 思想），从而减少过拟合。

# Dropout层

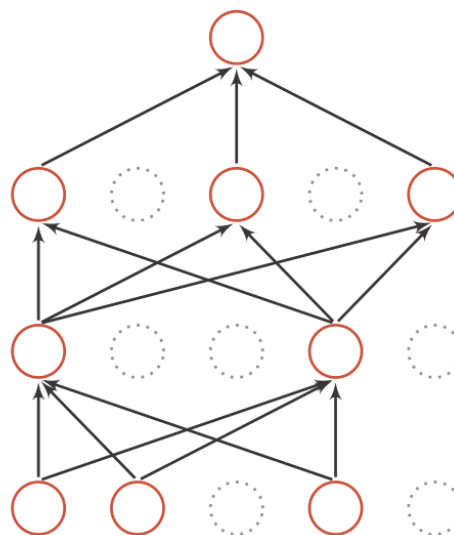
► 对于一个神经层  $y = f(Wx + b)$ ，引入一个丢弃函数  $d(\cdot)$  使得  $y = f(Wd(x) + b)$ 。

$$d(\mathbf{x}) = \begin{cases} \mathbf{m} \odot \mathbf{x} & \text{当训练阶段时} \\ p\mathbf{x} & \text{当测试阶段时} \end{cases}$$

► 其中  $m \in \{0,1\}^d$  是丢弃掩码 (dropout mask)，通过以概率为  $p$  的伯努利分布随机生成。



(a) 标准网络



(b) Dropout 后的网络

# 局部响应归一化

---

- ▶ 局部响应归一化是对邻近的特征映射进行局部归一化，是一种受生物学启发的归一化方法，即活跃神经元对相邻神经元具有抑制作用

$$\hat{\mathbf{Y}}^p = \mathbf{Y}^p / \left( k + \alpha \sum_{j=\max(1, p-\frac{n}{2})}^{\min(P, p+\frac{n}{2})} (\mathbf{Y}^j)^2 \right)^\beta$$
$$\triangleq \text{LRN}_{n,k,\alpha,\beta}(\mathbf{Y}^p),$$

- ▶  $n, k, \alpha, \beta$  为超参， $n$  为局部归一化的特征窗口大小。在 AlexNet 中，这些超参的取值为  $n=5, k=2, \alpha=10\text{e-}4, \beta=0.75$ 。



# 局部响应归一化

---

- ▶ 局部响应归一化应用在激活函数之后，只是对邻近的神经元进行局部归一化，并且不减去均值。
- ▶ AlexNet 的网络结构前两个卷积层（Conv1 和 Conv2）后均使用了 LRN：

**Conv1 → ReLU → LRN → MaxPool**

**Conv2 → ReLU → LRN → MaxPool**

# 局部响应归一化

---

- ▶ 当使用ReLU作为激活函数时，神经元的活性值是没有限制的，局部响应归一化可以起到平衡和约束作用。
- ▶ 如果一个神经元的活性值非常大，那么和它邻近的神经元就近似地归一化为 0，从而起到抑制作用，增强模型的泛化能力。
- ▶ 最大汇聚也具有侧抑制作用。但最大汇聚是对同一个特征映射中的邻近位置中的神经元进行抑制，而局部响应归一化是对同一个位置的邻近特征映射中的神经元进行抑制。

# 局部响应归一化

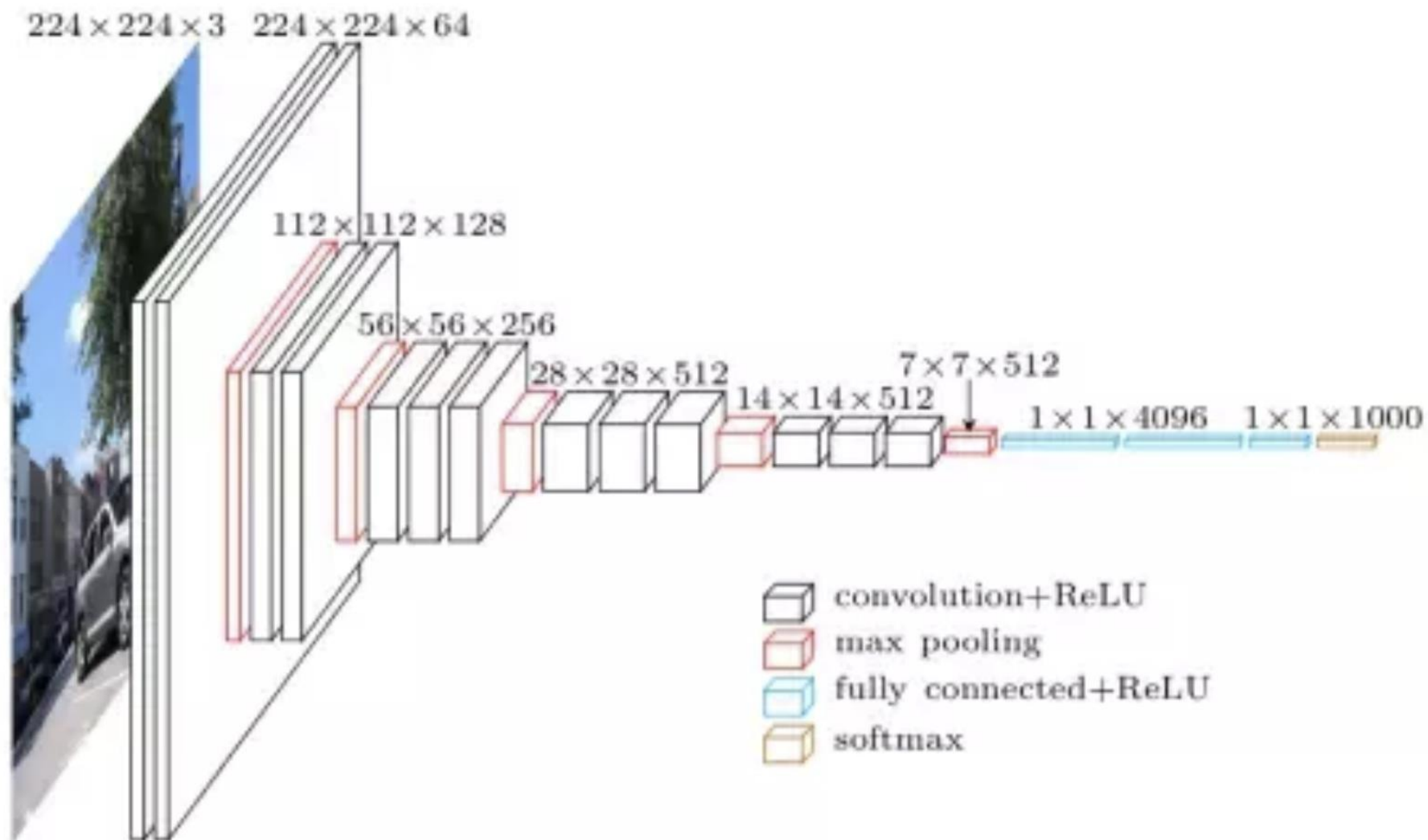
---

- ▶ 在2015年 Very Deep Convolutional Networks for Large-Scale Image Recognition. 提到LRN基本没什么用。

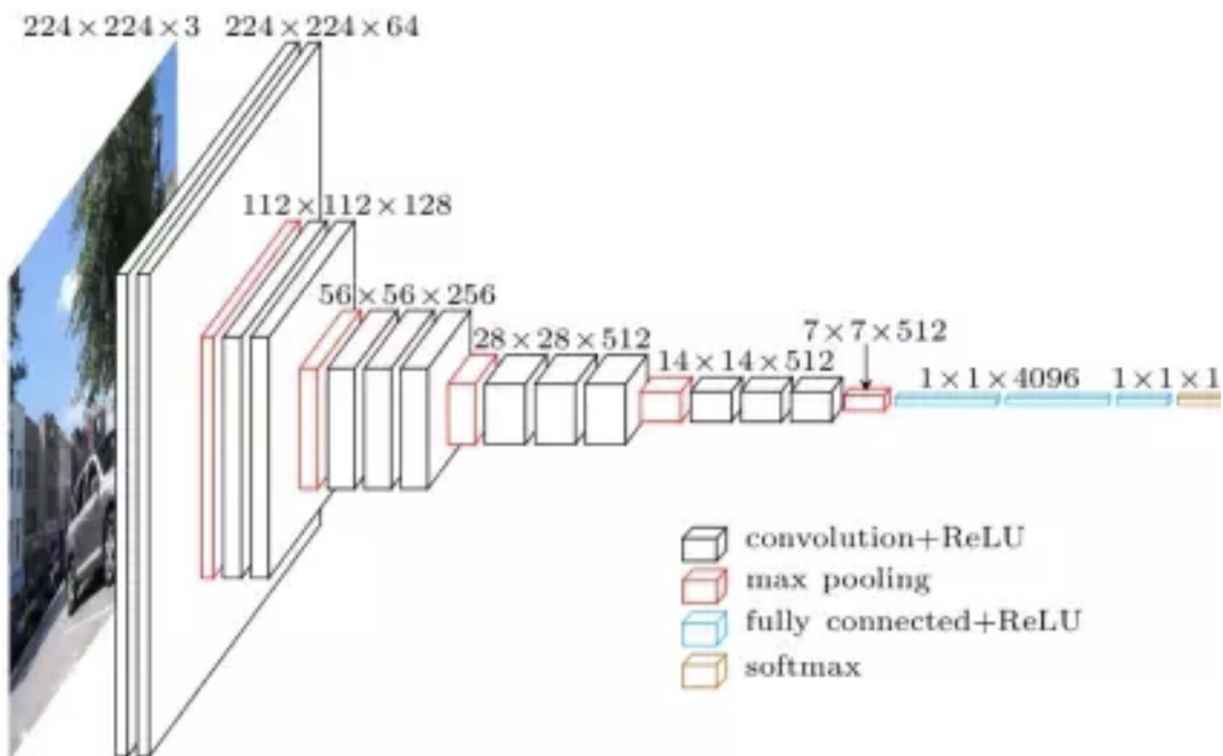
All hidden layers are equipped with the rectification (ReLU (Krizhevsky et al., 2012)) non-linearity. We note that none of our networks (except for one) contain Local Response Normalisation (LRN) normalisation (Krizhevsky et al., 2012): as will be shown in Sect. 4, such normalisation does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time. Where applicable, the parameters for the LRN layer are those of (Krizhevsky et al., 2012).

# VGG

原文: <https://arxiv.org/abs/1409.1556>



# VGG



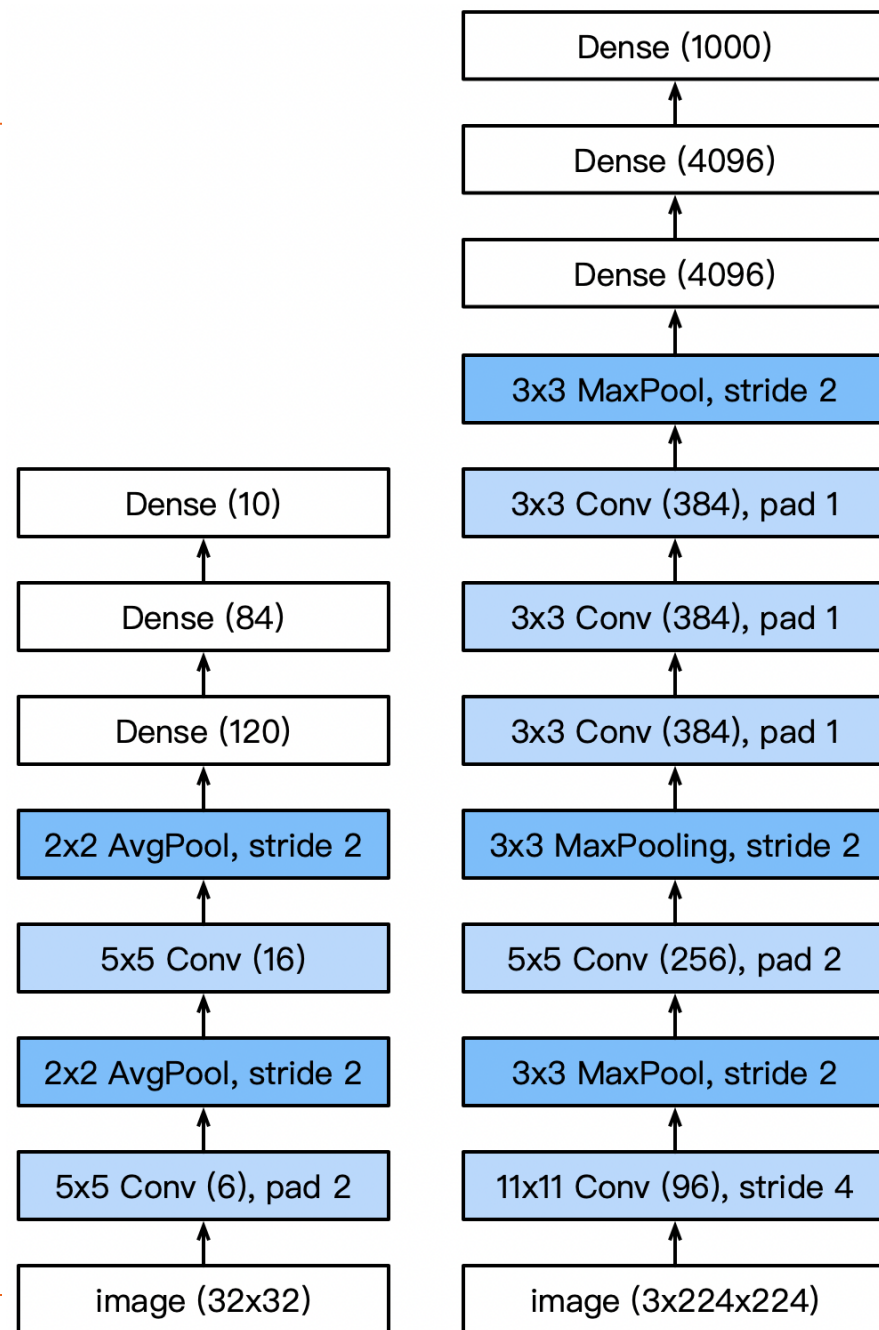
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

# VGG

- ▶ AlexNet 比 LeNet 更深入更大，以获得更强性能
- ▶ 怎么更大更深？

## ▶ 选项

- ▶ 更多稠密层(开销太大)
- ▶ 更多的卷积层
- ▶ 分组成块





# VGG 块

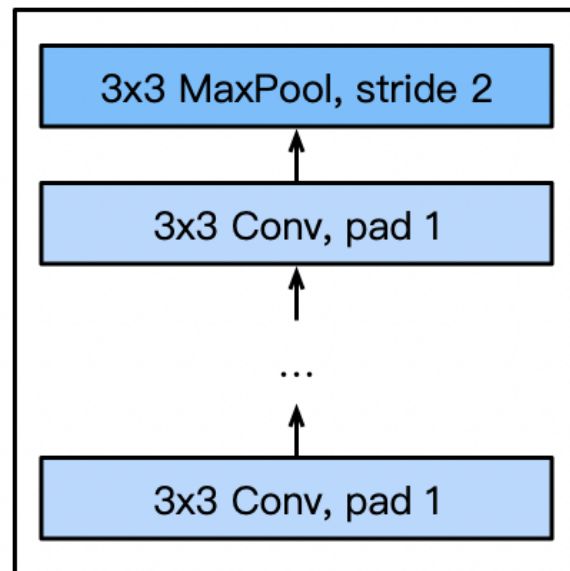
## ► 更深还是更宽?

- 5x5 卷积
- 3x3 卷积 (更多)
- 更深和更窄更好

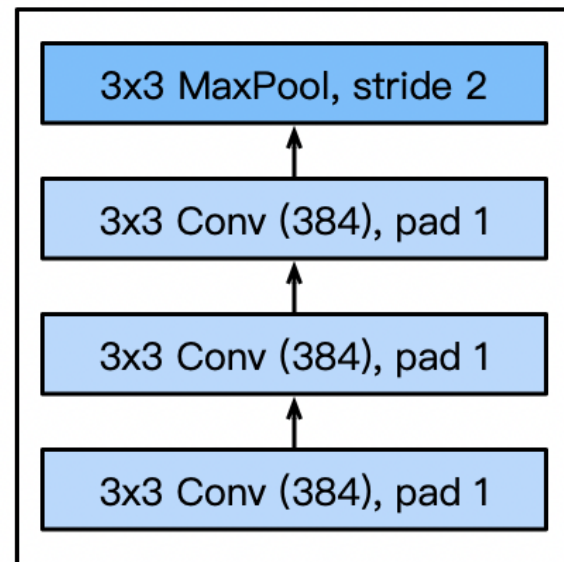
## ► VGG 块

- 3x3 卷积 (填充=1) (n层, m个通道)
- 2x2 最大池化层 (步长=2)

VGG 块



AlexNet 一部分



# VGG 结构

## ► 模块化设计:

► 多个VGG块后加稠密层

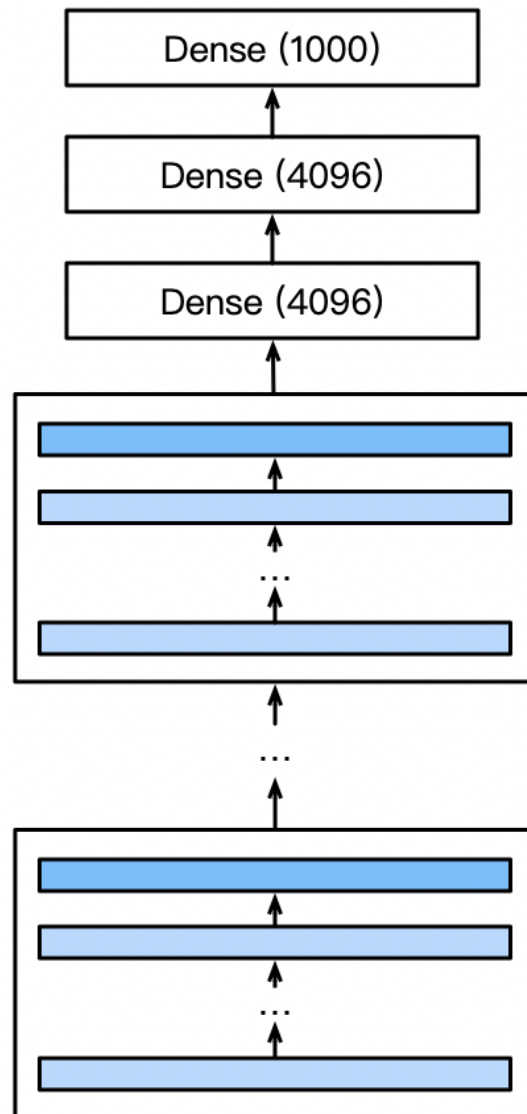
## ► 不同数目重复VGG块

► 可获得不同的架构,

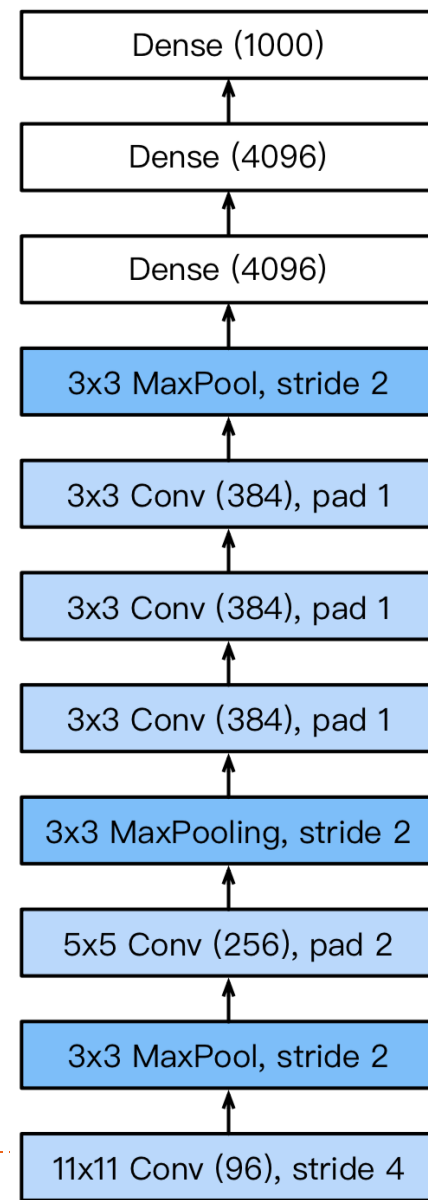
► 例如VGG-16, VGG-19,

.....

VGG



AlexNet



# 进程

---

## ▶ LeNet (1995)

- ▶ 卷积层 + 池化层
- ▶ 隐含层

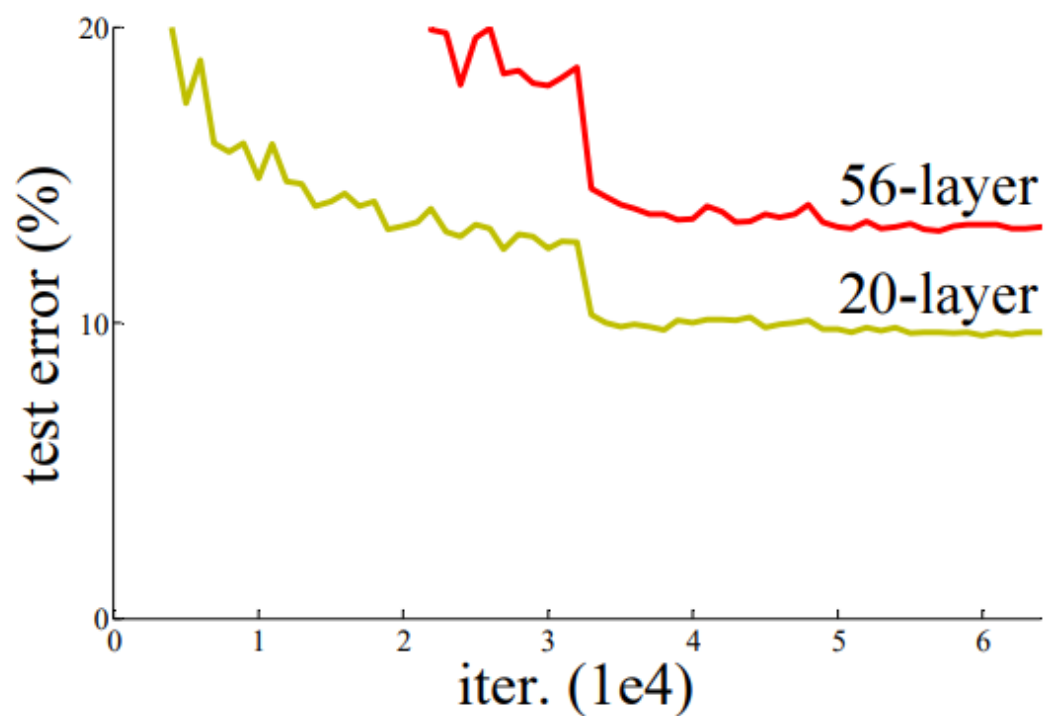
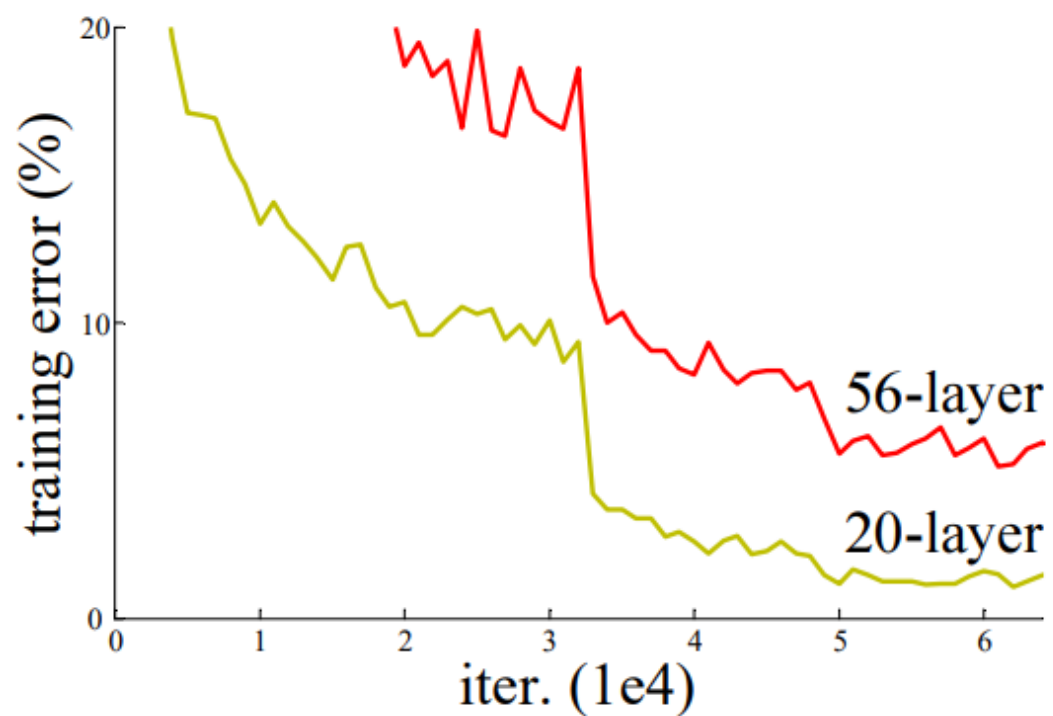
## ▶ AlexNet

- ▶ 更大更深的 LeNet
- ▶ ReLu 激活, 丢弃法, 预处理

## ▶ VGG

- ▶ 更大更深的 AlexNet (重复的 VGG 块)
- ▶ 使用可重复的卷积块来构建深度卷积神经网络

# VGG的“退化”现象



# 残差网络

- ▶ 是一种深度卷积神经网络（CNN）架构，由微软研究院提出，2015年在ImageNet竞赛中获得了冠军。
- ▶ 假设在一个深度网络中，我们期望一个非线性单元（可以为一层或多层的卷积层） $f(x, \theta)$ 去逼近一个目标函数为 $h(x)$ 。
- ▶ 将目标函数拆分成两部分：**恒等函数**和**残差函数**

$$h(\mathbf{x}) = \underbrace{\mathbf{x}}_{\text{恒等函数}} + \underbrace{(h(\mathbf{x}) - \mathbf{x})}_{\text{残差函数}} \rightarrow f(\mathbf{x}, \theta)$$

# 残差网络

---

- ▶ 是一种深度卷积神经网络（CNN）架构，由微软研究院提出，2015年在ImageNet竞赛中获得了冠军。
- ▶ ResNet的核心思想是引入了“残差学习”来解决深度网络训练中的退化问题，即随着网络层数的增加，网络的性能反而下降的问题。
- ▶ 在传统的深度网络中，如果网络层数增加到一定程度，网络将难以训练，并且准确率先是增加然后饱和，再增加层数甚至会导致准确率下降。这是因为，随着层数的增加，网络中的梯度在反向传播过程中会逐渐消失或爆炸，导致网络无法学习到有效的特征表示。
- ▶ ResNet通过引入跳跃连接（skip connections）或称为shortcut connections来解决这个问题。这些连接允许网络中的信号绕过一些层直接传播，从而使得网络可以学习到恒等映射（identity mapping），即直接传递输入到输出而不经任何变化。这样，即使网络非常深，也可以保证网络至少能够学习到一个有效的恒等映射，从而保证了网络性能不会随着层数增加而下降。



# ResNet

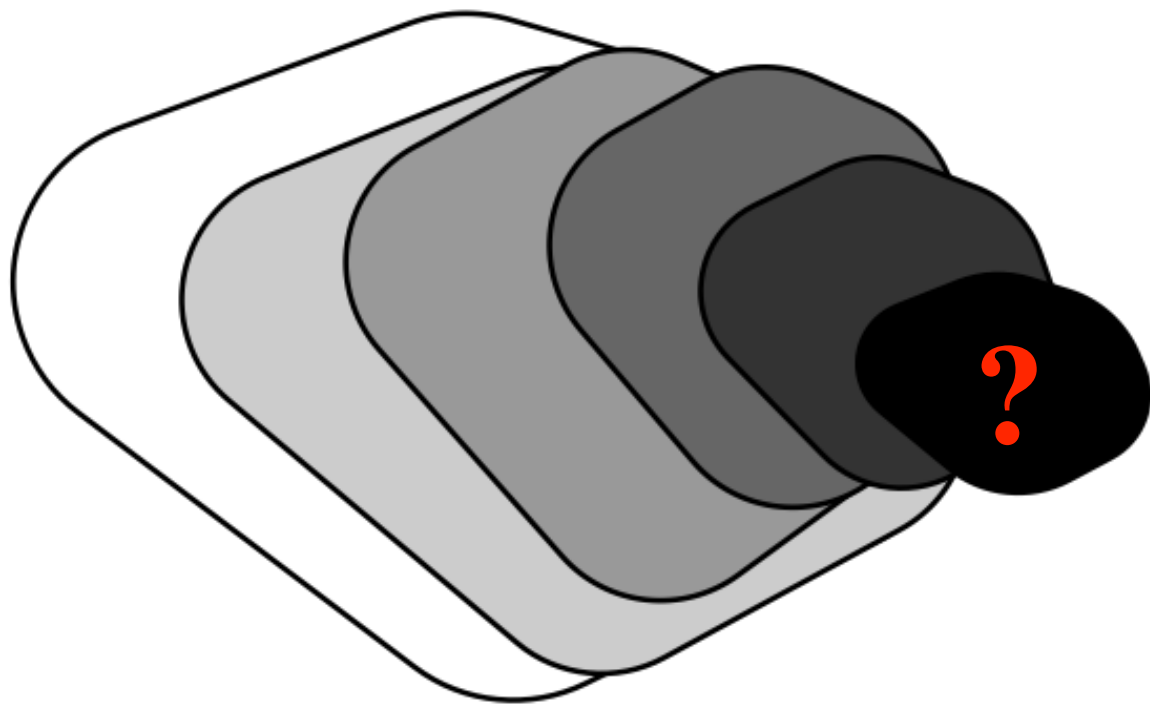
---

## ▶ 2015 ILSVRC winner (152层)

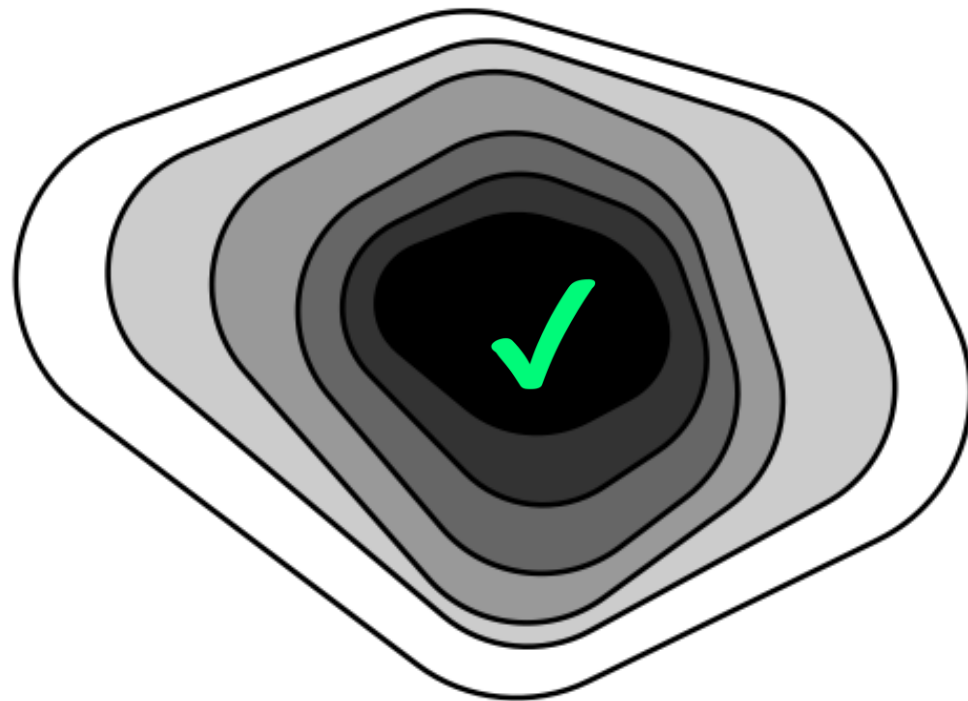
- ▶ 错误率: 3.57%
- ▶ 超深的网络结构(突破1000层)
- ▶ 提出residual模块
- ▶ 使用Batch Normalization加速训练(丢弃dropout)

# 添加层会提高准确性吗？

---

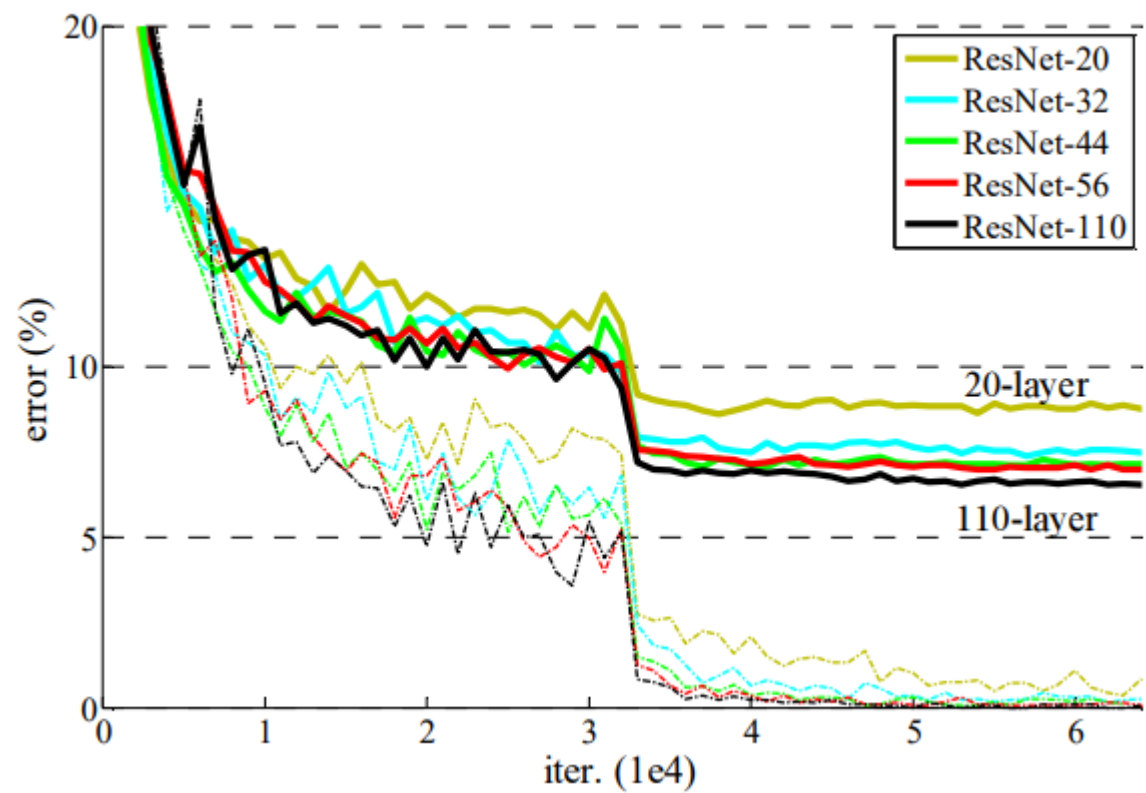
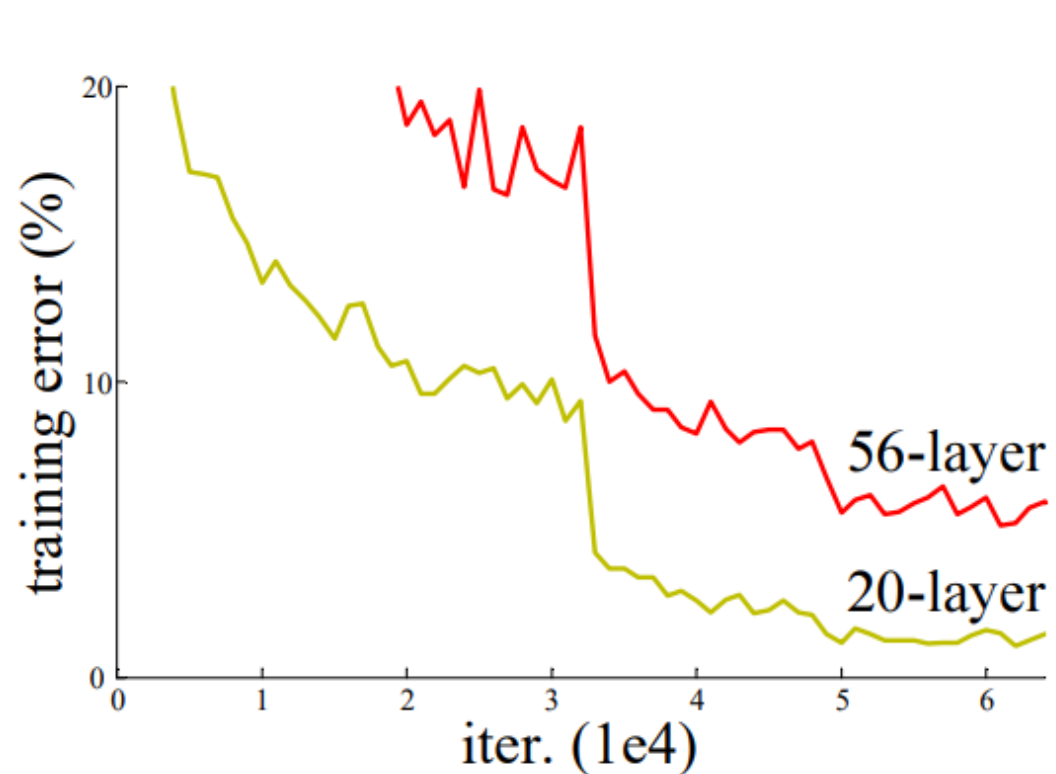


generic function classes



nested function classes

# 添加层会提高准确性吗？

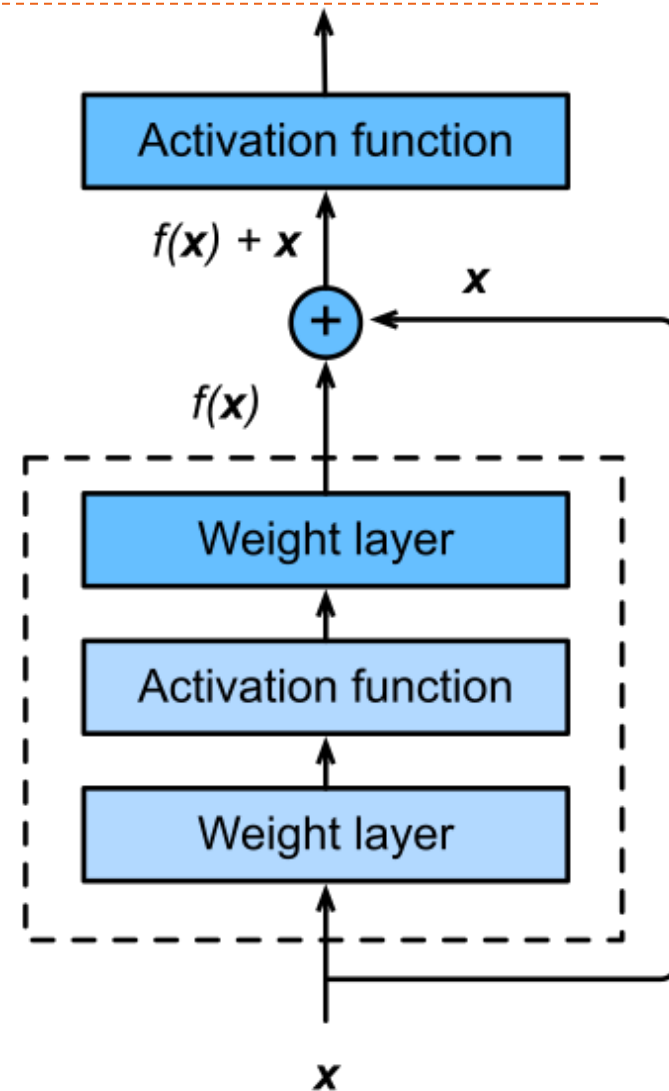
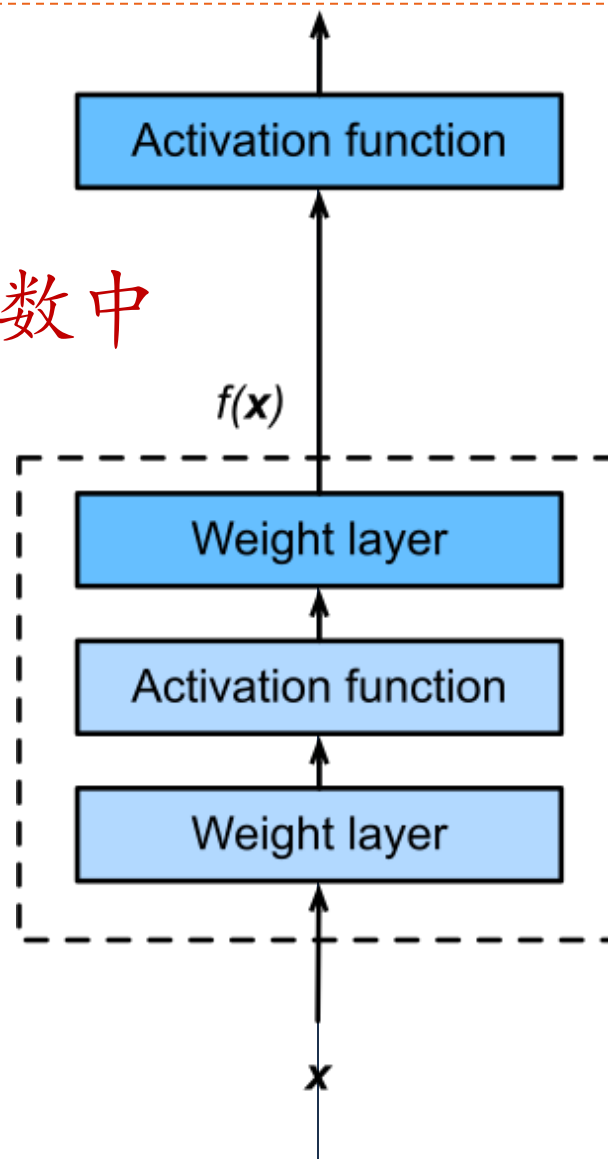


# 残差网络 (ResNet)

► 添加层会更改原特征类

► 我们想要“加”到原函数中  
“泰勒展开”的参数

$$f(x) = x + g(x)$$



# 1x1的卷积核

---

▶ 经卷积后的矩阵尺寸大小计算公式为：

$$\text{Out} = \frac{\text{Inp} - W + 2P}{S} + 1$$

▶

- ① 输入图片大小  $\text{Inp} \times \text{Inp}$
- ② Filter大小  $W \times W$
- ③ 步长  $S$
- ④ padding的像素数  $P$

# 1x1的卷积核

---

## ► Padding: Valid

►  $Out = \frac{Inp - W + 1}{S}$  向上取整

- ① 输入图片大小  $Inp \times Inp$
- ② Filter大小  $W \times W$
- ③ 步长  $S$

## Padding: Same

$Out = \frac{Inp}{S}$  向上取整

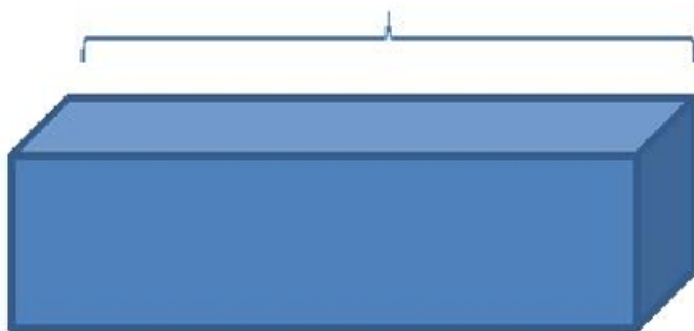


# 1x1的卷积核

## ►GoogLeNet 详解

### ►不使用1x1卷积核降维

channels: 512



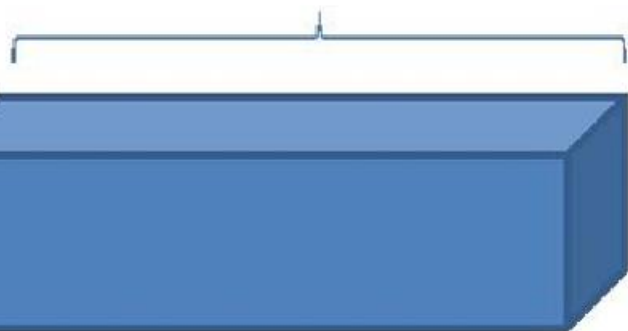
使用64个5x5的卷积核进行卷积



$$5 \times 5 \times 512 \times 64 = 819\,200$$

### ►使用1x1卷积核降维

channels: 512



使用24个1x1的卷积核进行卷积



channels: 24



使用64个5x5的卷积核进行卷积

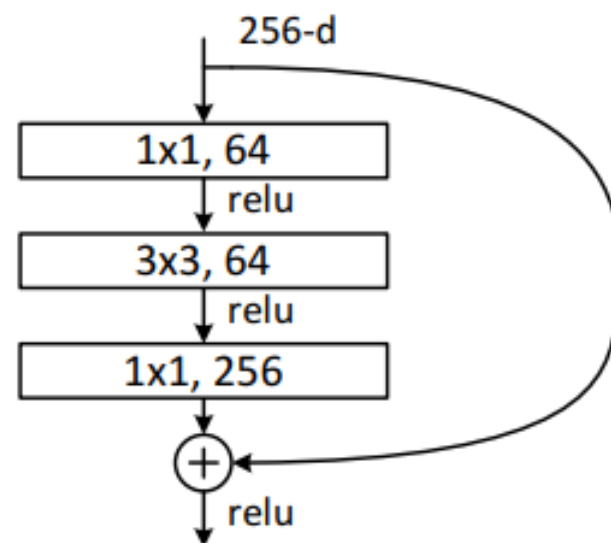
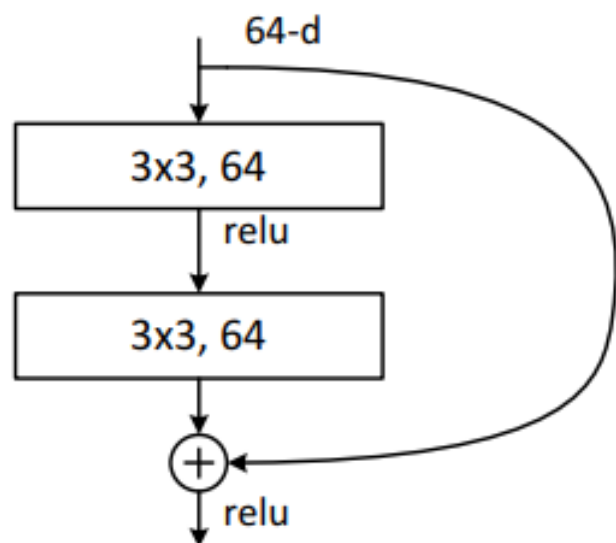


$$1 \times 1 \times 512 \times 24 = 12\,288$$

$$5 \times 5 \times 24 \times 64 = 38\,400$$

$$12\,288 + 38\,400 = 50\,688$$

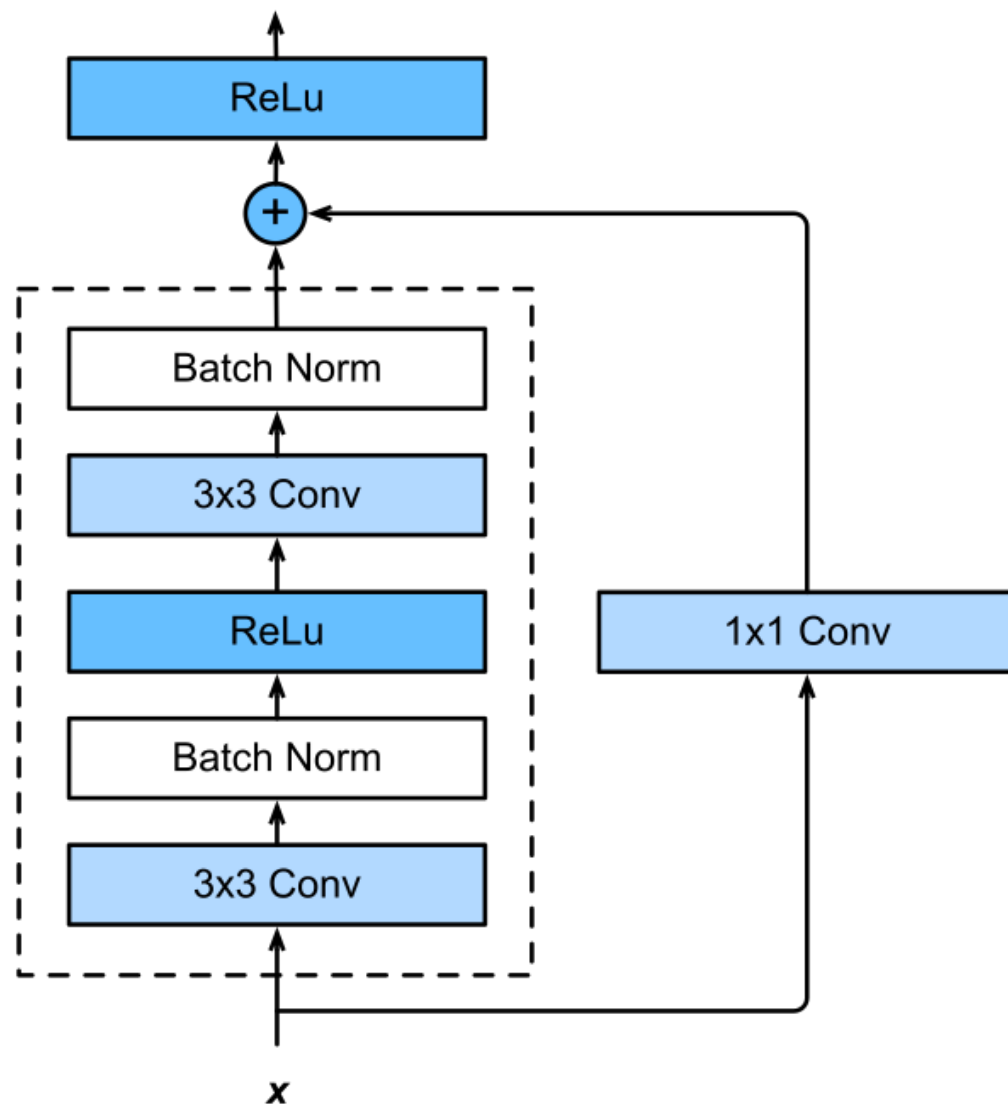
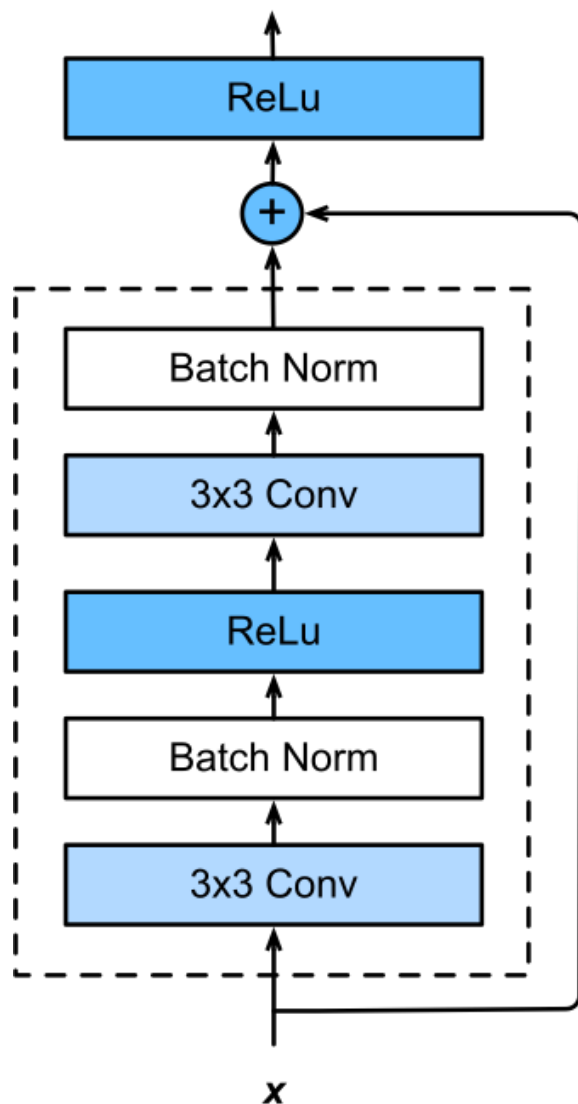
# 残差块



$$3 \times 3 \times 256 \times 256 + 3 \times 3 \times 256 \times 256 = 1,179,648$$

$$1 \times 1 \times 256 \times 64 + 3 \times 3 \times 64 \times 64 + 1 \times 1 \times 64 \times 256 = 69,632$$

# 残差块

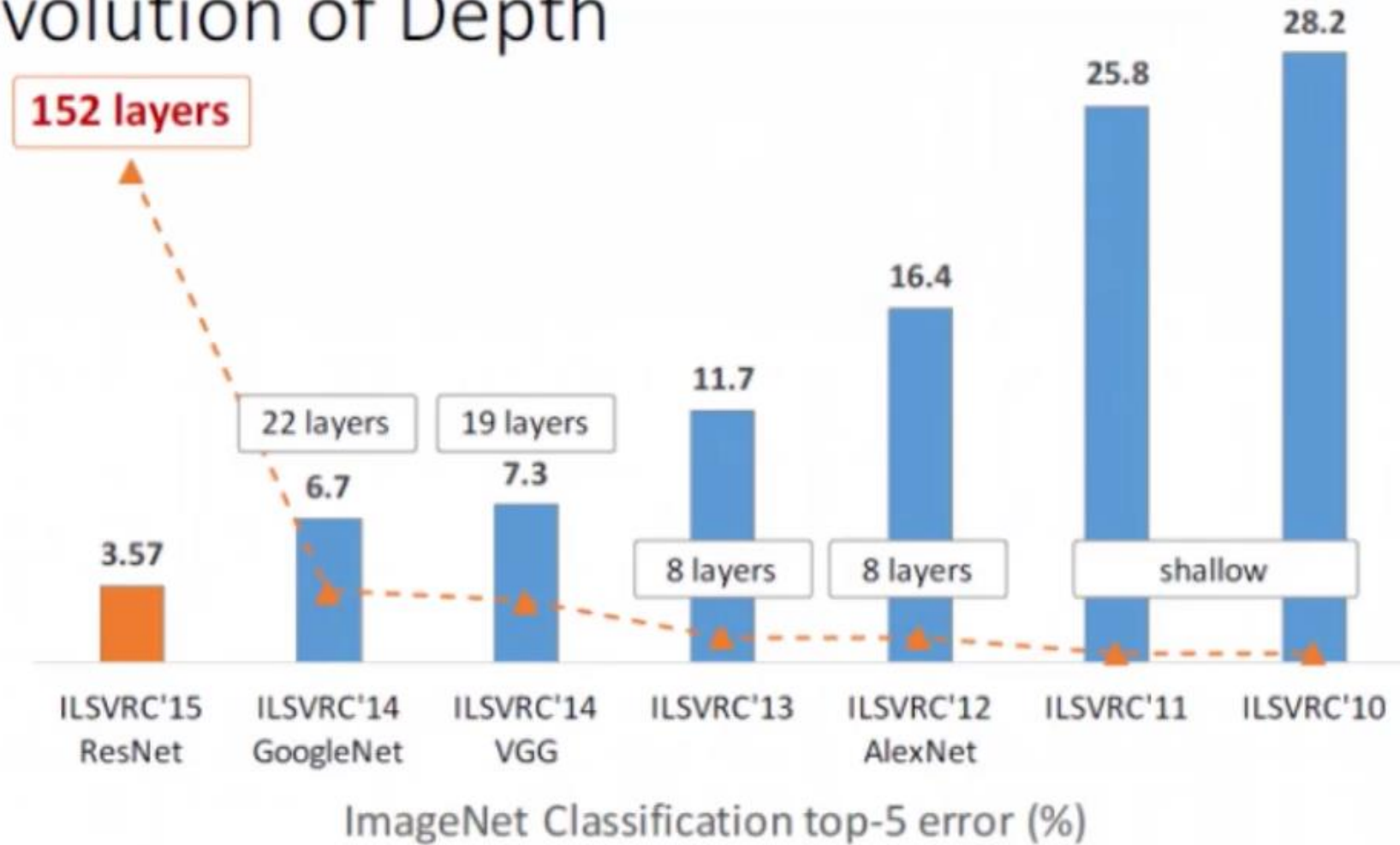


# ResNet详解

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				
		$\begin{bmatrix} 3\times 3, 64 \\ 3\times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3, 64 \\ 3\times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 64 \\ 3\times 3, 64 \\ 1\times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 64 \\ 3\times 3, 64 \\ 1\times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 64 \\ 3\times 3, 64 \\ 1\times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times 3, 128 \\ 3\times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3, 128 \\ 3\times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times 1, 128 \\ 3\times 3, 128 \\ 1\times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times 1, 128 \\ 3\times 3, 128 \\ 1\times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times 1, 128 \\ 3\times 3, 128 \\ 1\times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times 3, 256 \\ 3\times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3, 256 \\ 3\times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times 1, 256 \\ 3\times 3, 256 \\ 1\times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times 1, 256 \\ 3\times 3, 256 \\ 1\times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times 1, 256 \\ 3\times 3, 256 \\ 1\times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times 3, 512 \\ 3\times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3, 512 \\ 3\times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 512 \\ 3\times 3, 512 \\ 1\times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 512 \\ 3\times 3, 512 \\ 1\times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times 1, 512 \\ 3\times 3, 512 \\ 1\times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

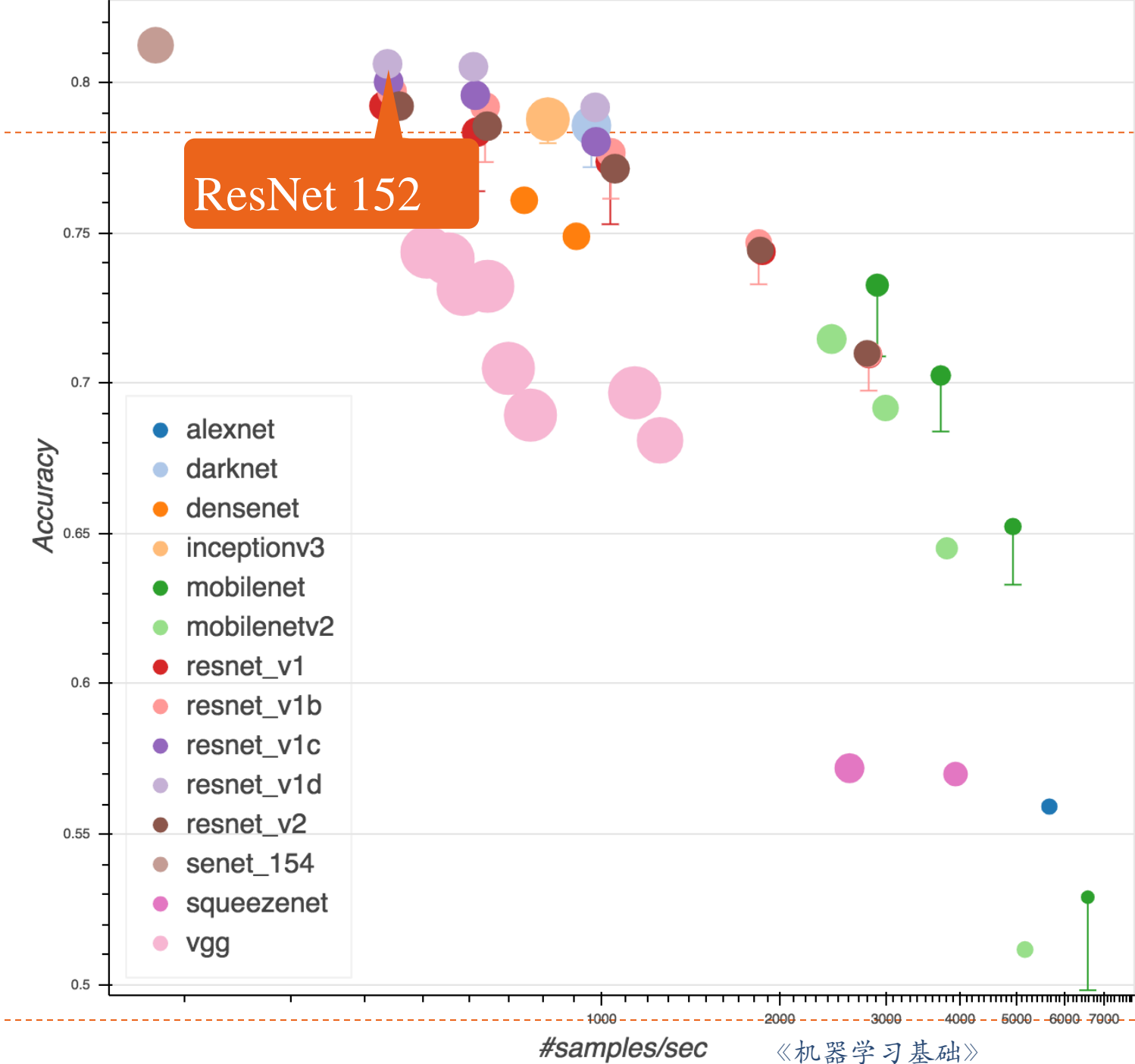
# ResNet

## Revolution of Depth





GluonCV 模型 “动物园”  
[https://gluon-cv.mxnet.io/model\\_zoo/classification.html](https://gluon-cv.mxnet.io/model_zoo/classification.html)





# ResNet

---

- ▶ 残差块使得很深的网络更加容易训练
  - ▶ 甚至可以训练一千层的网络
- ▶ 残差网络对随后的深层神经网络设计产生了深远影响，无论是卷积类网络还是全连接类网络。

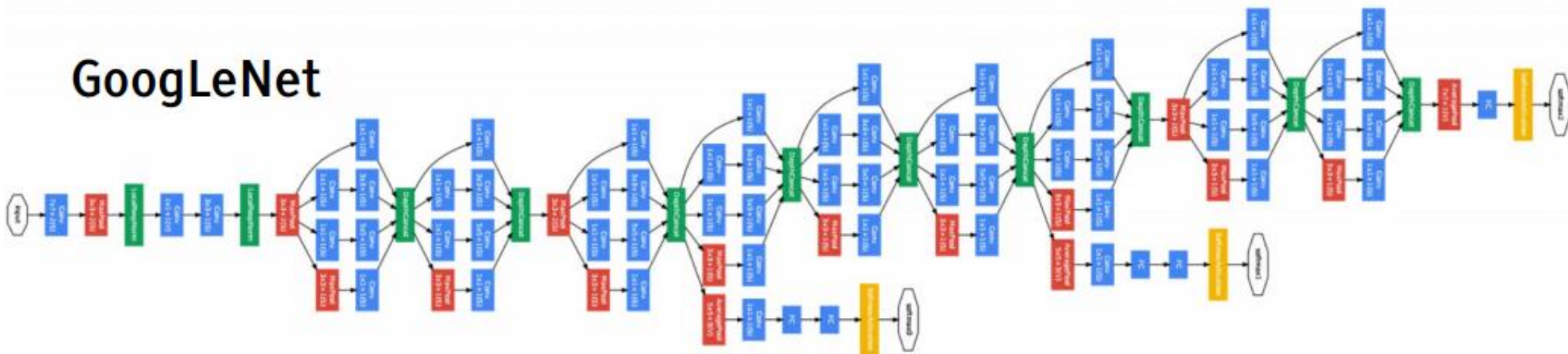
# Inception网络

## ► 2014 ILSVRC winner (22层)

► 参数: GoogLeNet: 4M VS AlexNet: 60M

► 错误率: 6.7%

► Inception网络是由有多个inception模块和少量的汇聚层堆叠而成。



# GoogLeNet

---

- ▶ 引入了Inception结构（融合不同尺度的特征信息）
- ▶ 使用 $1 \times 1$ 的卷积核进行降维以及映射处理
- ▶ 添加两个辅助分类器帮助训练
- ▶ 丢弃全连接层，使用平均池化层（大大减少模型参数）

## 选最合适的卷积 ...

1x1

# 3x3

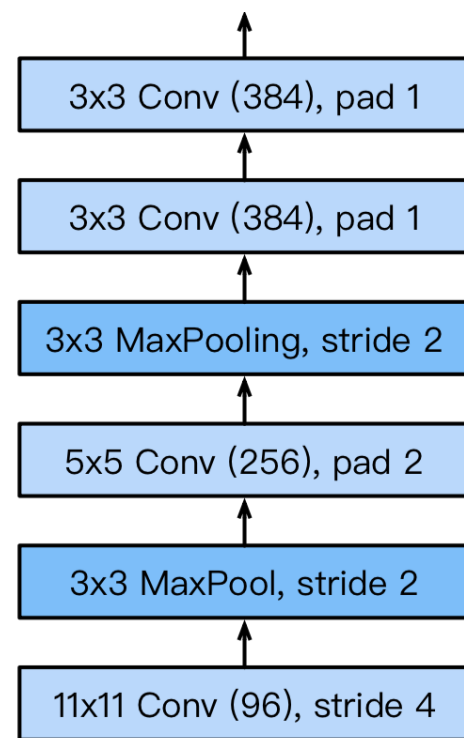
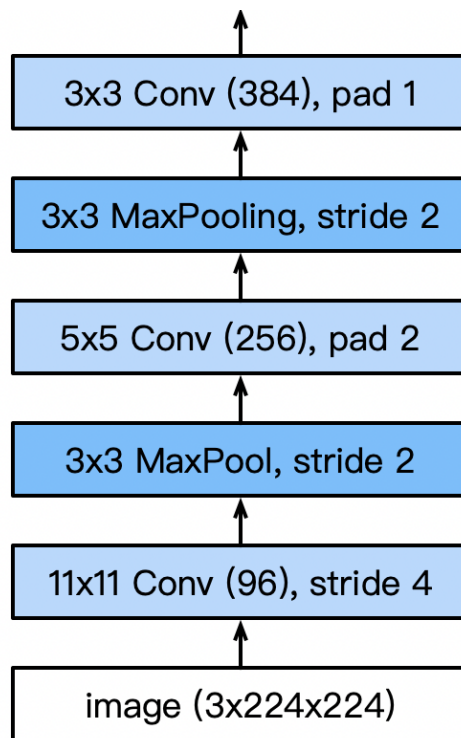
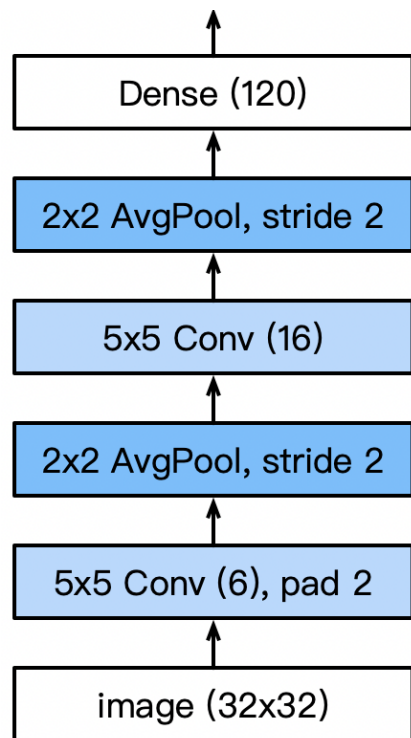
# 5x5

## 最大池化

# LeNet

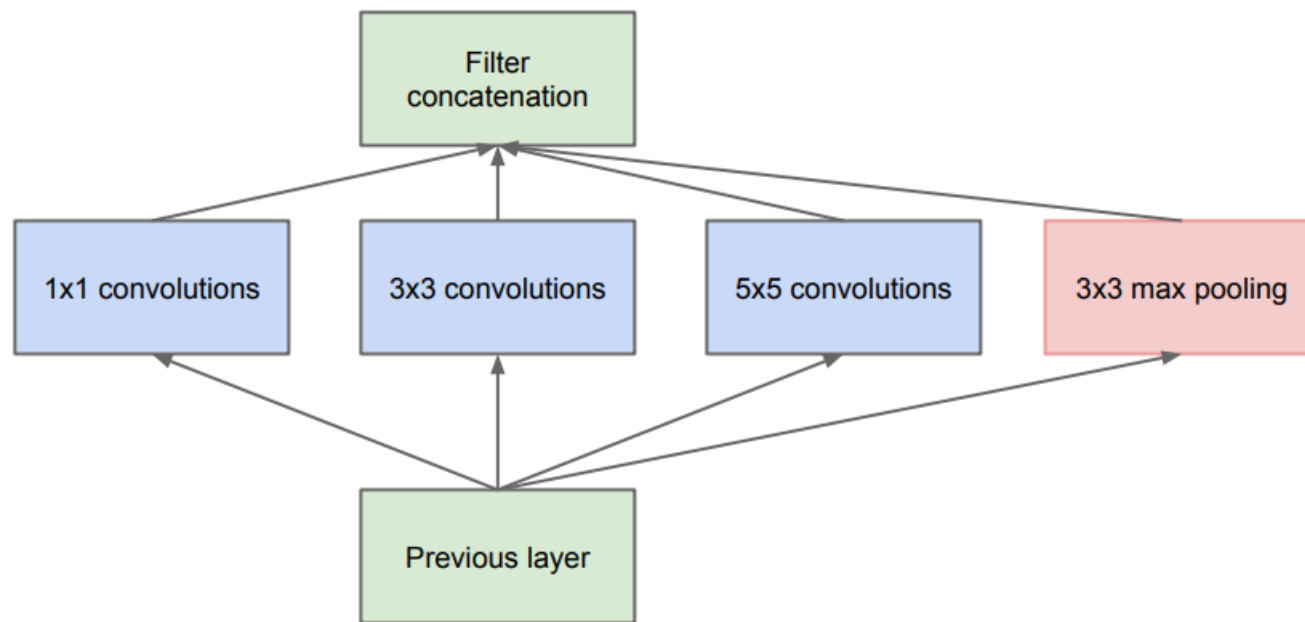
# AlexNet

VGG



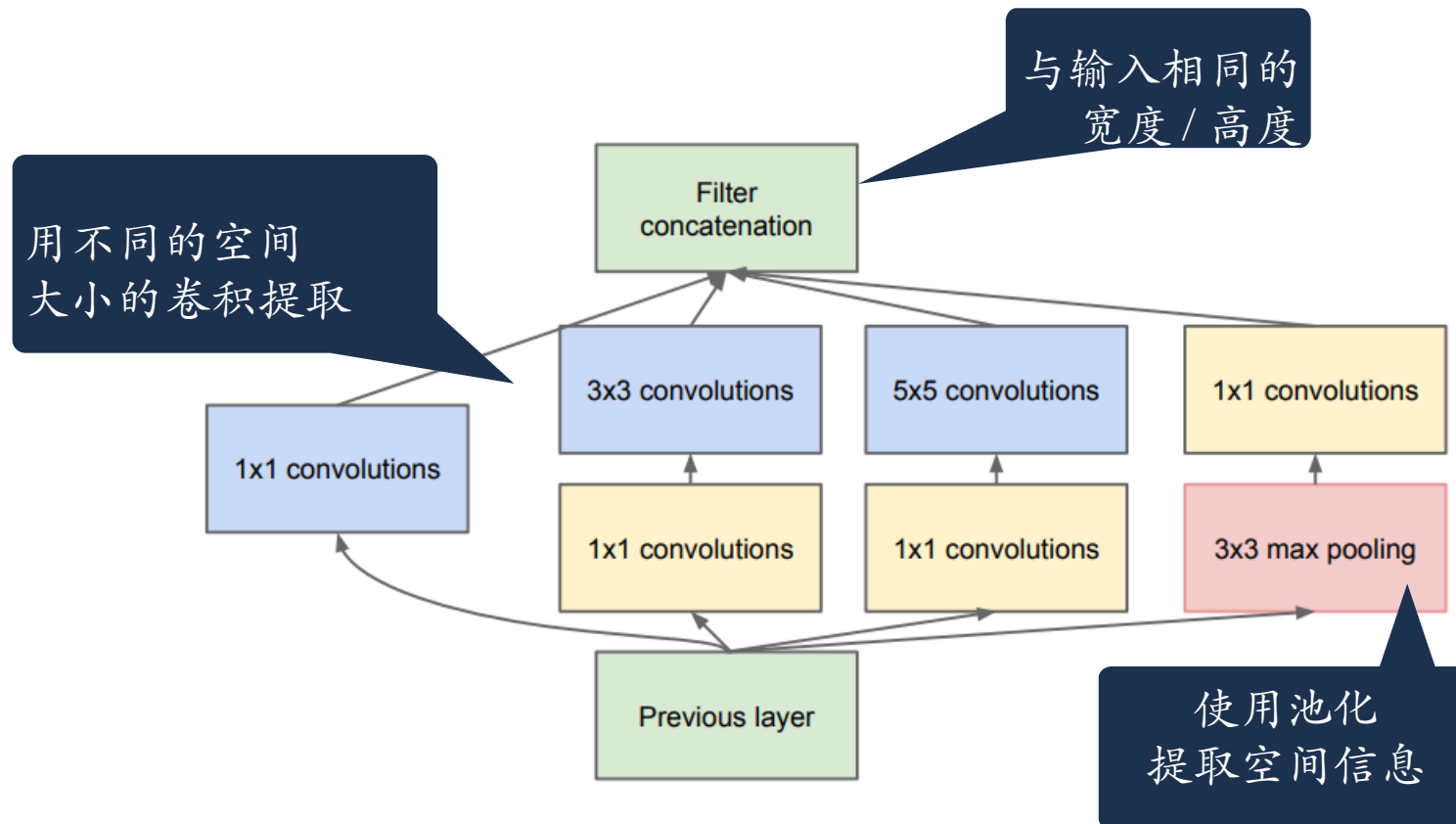
# Inspection结构

---



(a) Inception module, naïve version

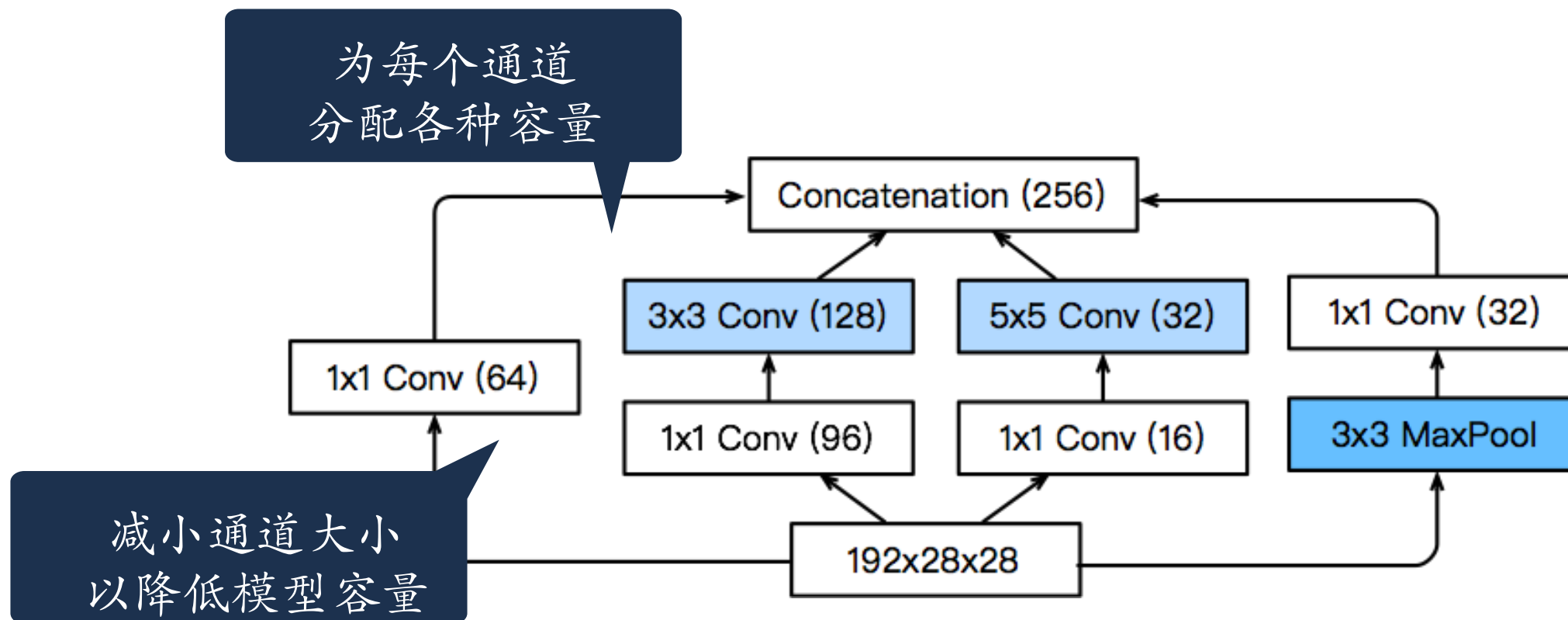
# Inspection结构



(b) Inception module with dimension reductions

# Inception 块

(第一个初始块) 指定的通道大小



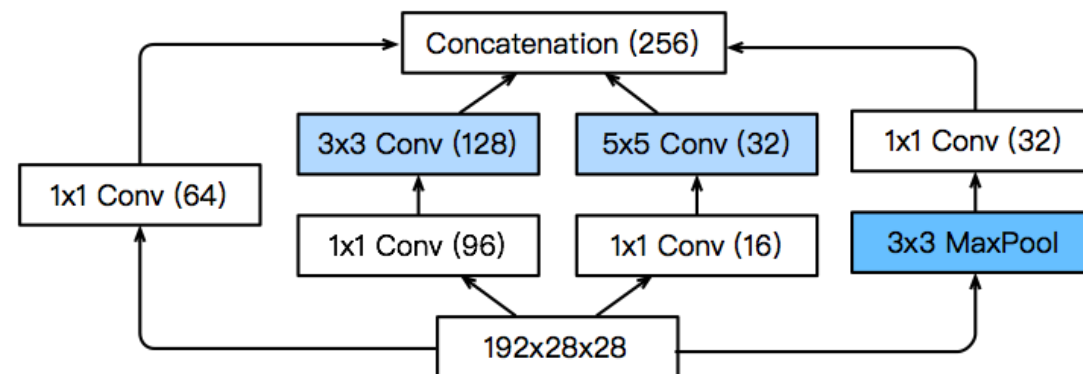


# Inception 块

与单个3x3或5x5卷积层相比，初始块具有更少的参数和更低的计算复杂度

- ▶ 不同功能混合（多样的功能类）
- ▶ 卷积核计算高效（良好的泛化）

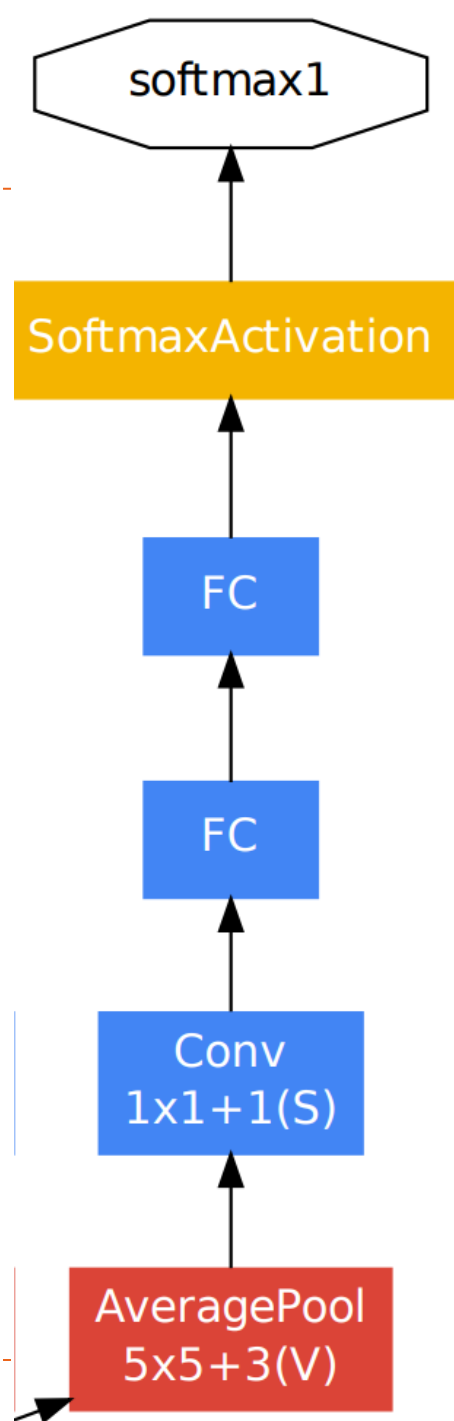
	# 参数	浮点运算 FLOPS
<b>Inception</b>	0.16 M	128 M
<b>3x3 卷积</b>	0.44 M	346 M
<b>5x5 卷积</b>	1.22 M	963 M



# 辅助分类器 (Auxiliary Classifier)

▶ 辅助分类器是一个附加网络结构，其具体结构如下：

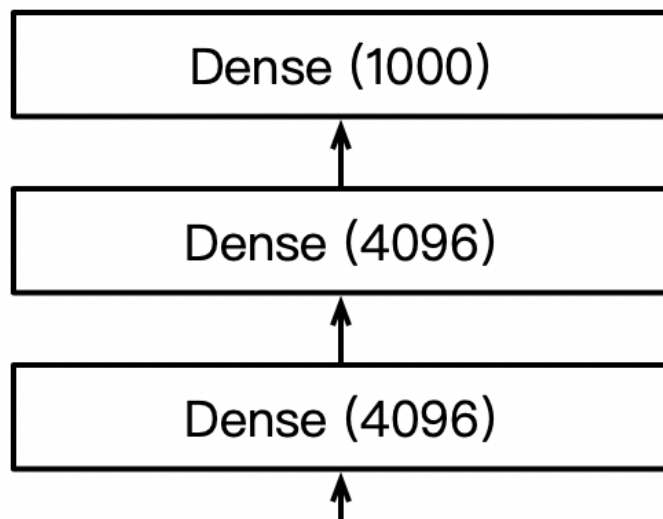
- ▶ 平均池化层：使用 $5 \times 5$ 的过滤器尺寸和步长为3，对于(4a)阶段产生一个 $4 \times 4 \times 512$ 的输出，对于(4d)阶段产生一个 $4 \times 4 \times 528$ 的输出。
- ▶  $1 \times 1$ 卷积层：使用128个过滤器进行维度缩减，并使用修正线性激活函数（Rectified Linear Activation，即ReLU）。
- ▶ 全连接层：包含1024个单元，同样使用修正线性激活函数。
- ▶ 丢弃层（Dropout Layer）：以70%的概率丢弃输出，这是一种防止过拟合的正则化技术。
- ▶ 线性层：使用softmax损失作为分类器，这个分类器预测与主分类器相同的1000个类别，但在推理（inference）时会被移除。



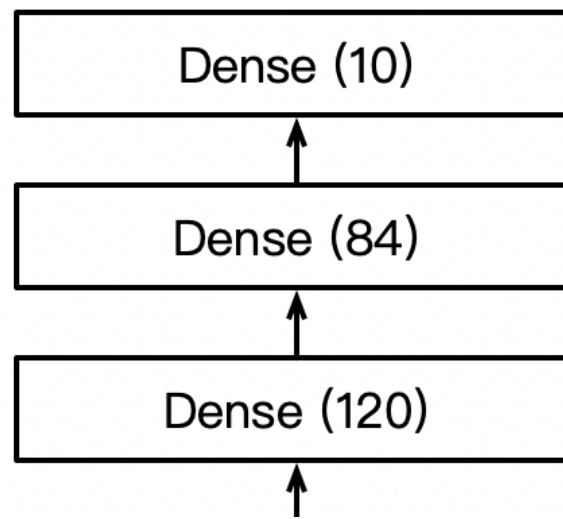
# 全连接架构

---

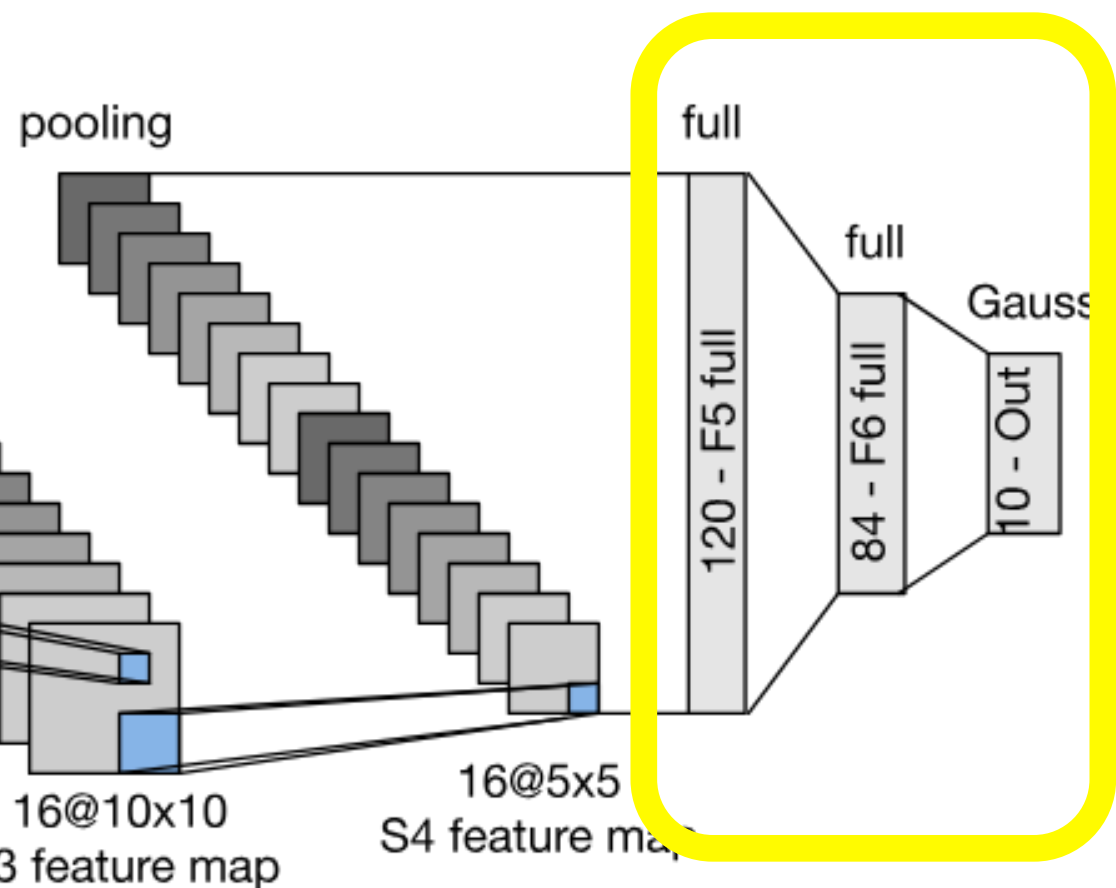
AlexNet/VGG



LeNet



# 最后一层的诅咒



► 卷积层需要相对较少的参数

$$c_i \times c_o \times k^2$$

► 最后一层(稠密层)对于n个类的需要许多参数

$$c \times m_w \times m_h \times n$$

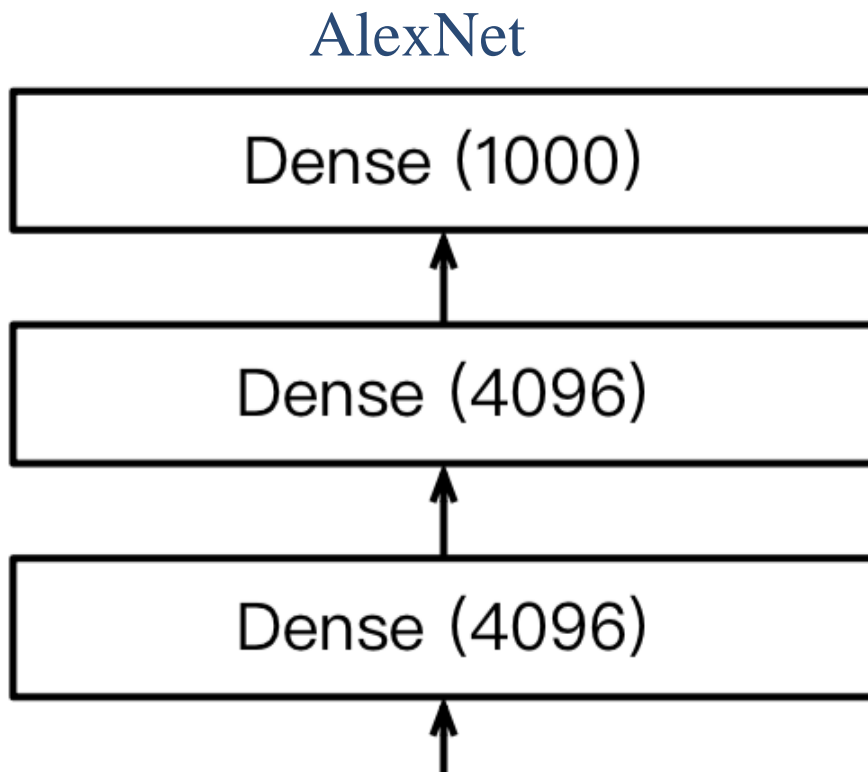
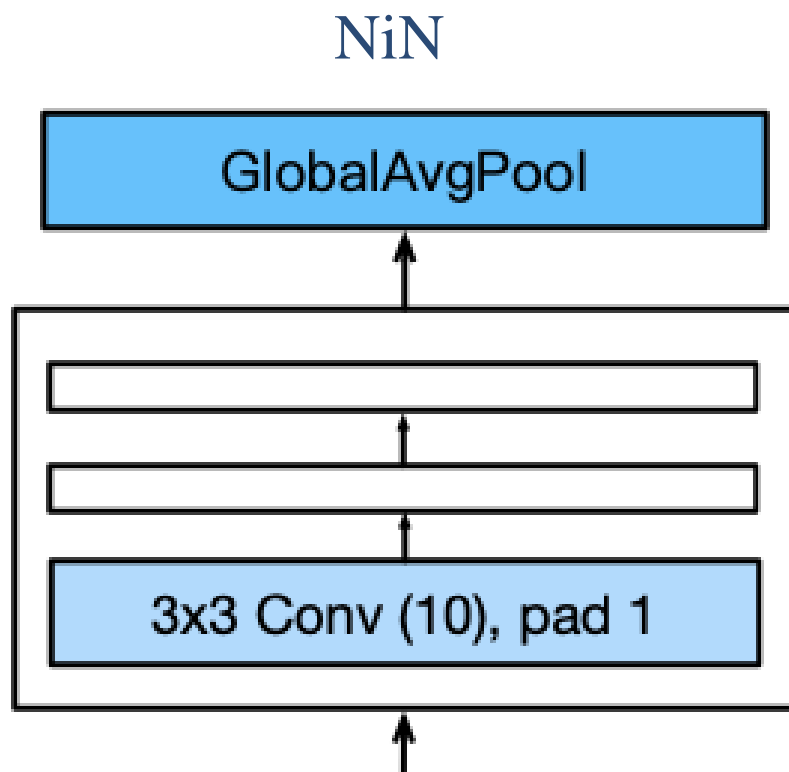
► LeNet  $16 \times 5 \times 5 \times 120 = 48k$

► AlexNet  $256 \times 5 \times 5 \times 4096 = 26M$

► VGG  $512 \times 7 \times 7 \times 4096 = 102M$

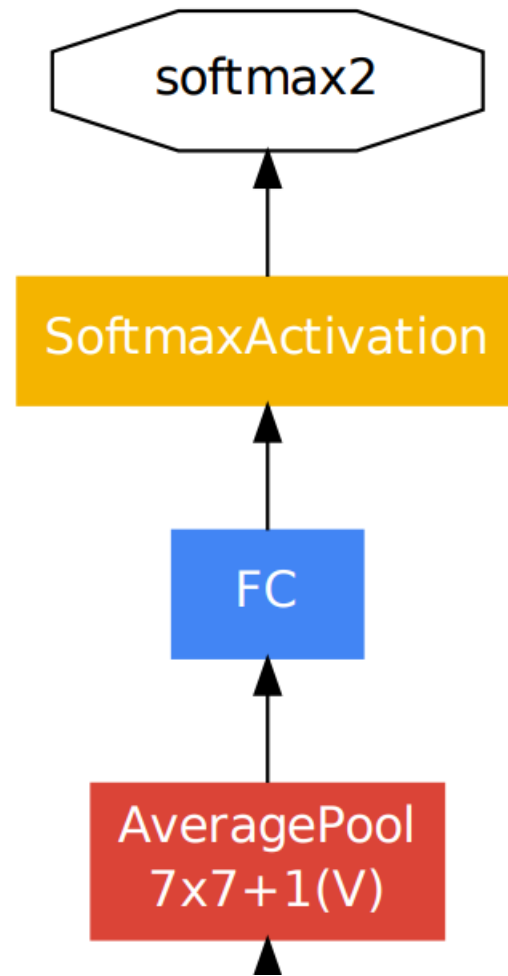
# NiN 最后一层

- ▶ 用 NiN 块替换了 AlexNet 的稠密层
- ▶ 输出：全局平均池化层



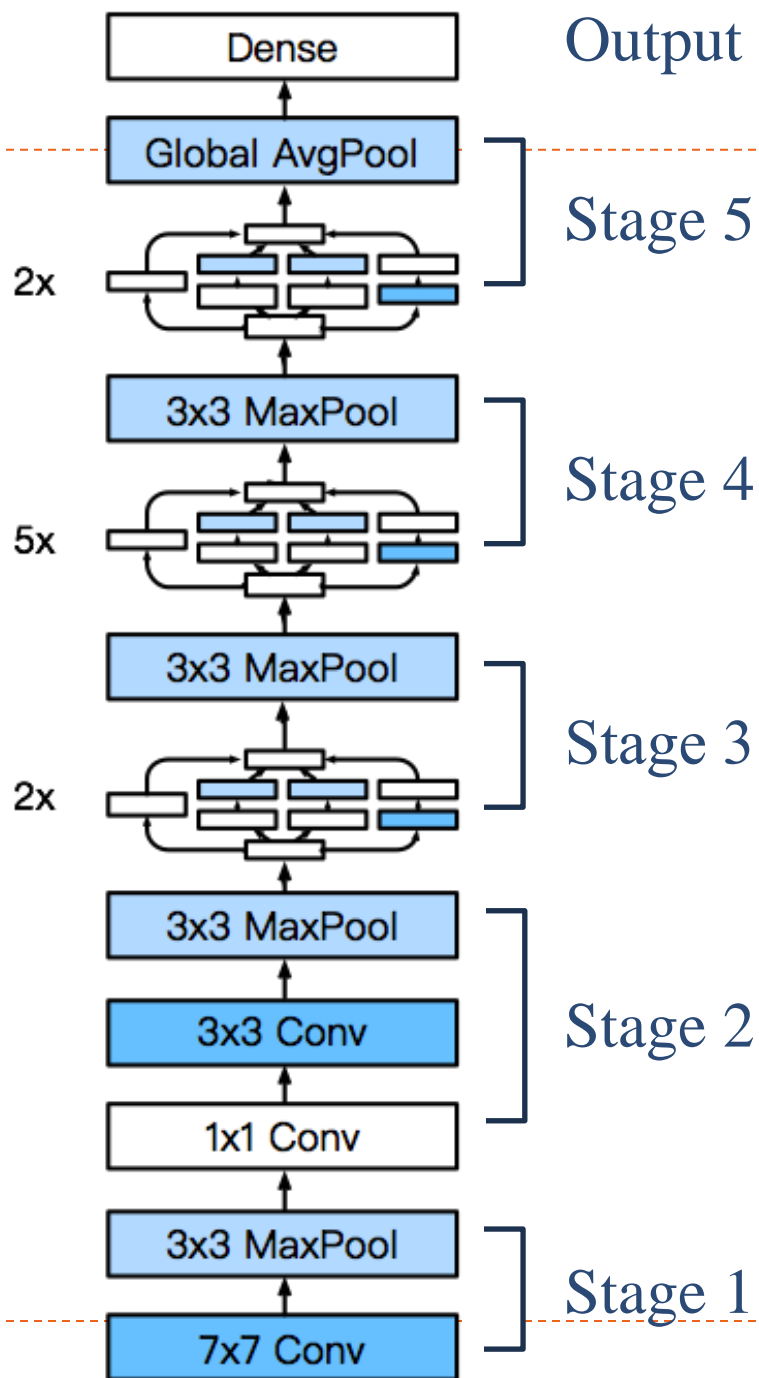
# GoogLeNet

---



# GoogLeNet

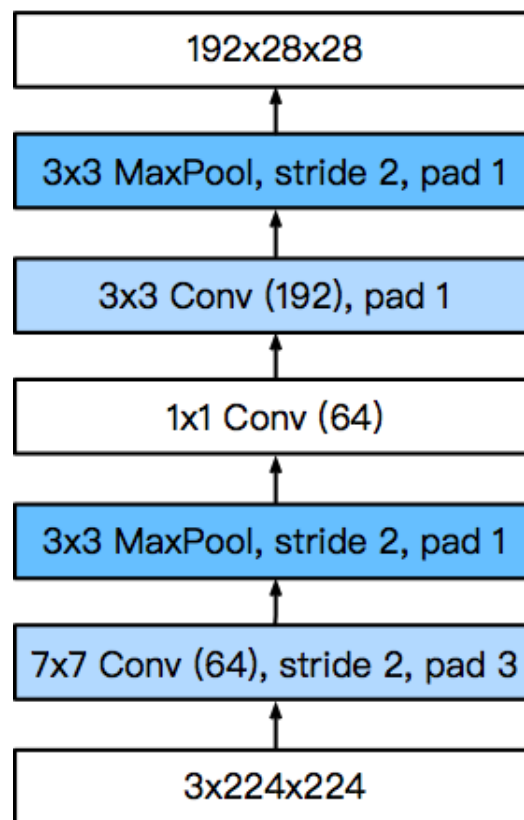
- 5 个阶段
- 9 个 Inception 块



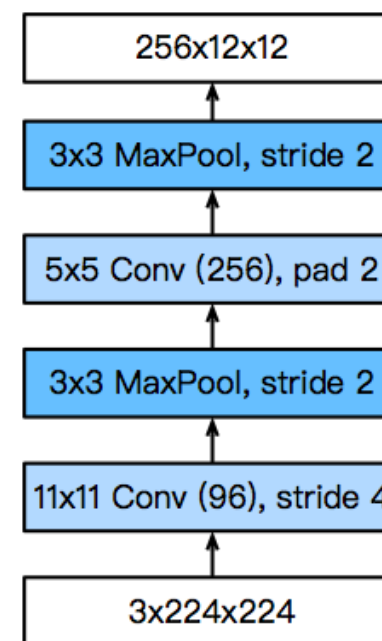
# 阶段 1 & 2

- ▶ 由于更多层:
- ▶ 更小的内核
- ▶ 更小的输出通道

GoogLeNet

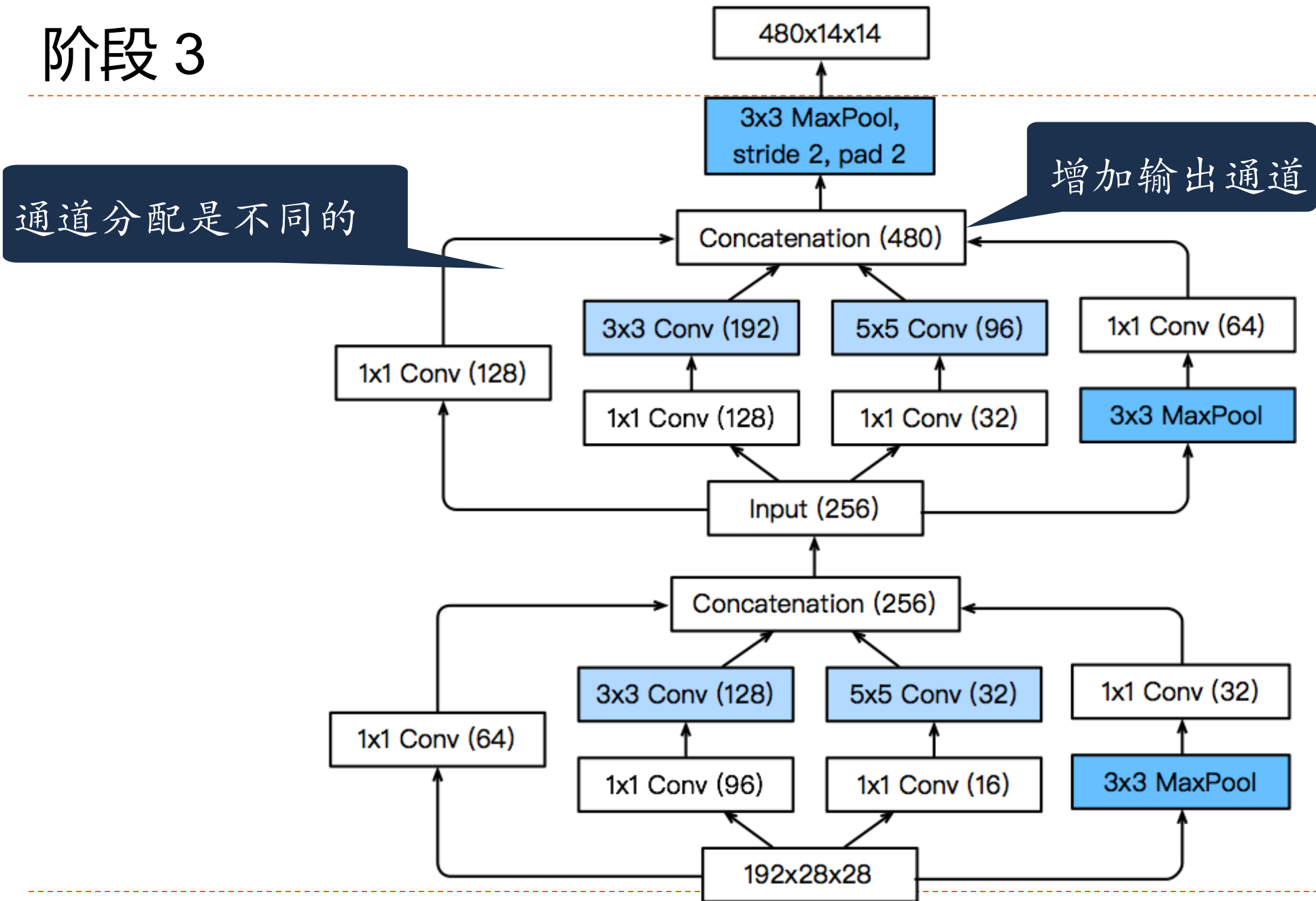


AlexNet





## 阶段 3



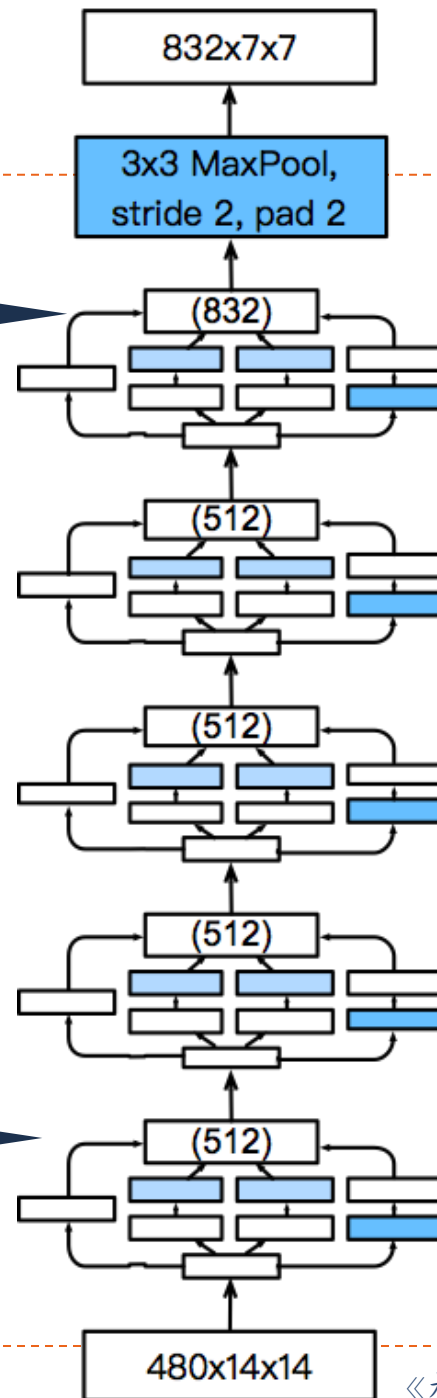
## 阶段 4 & 5

增加输出通道

1024个输出通道

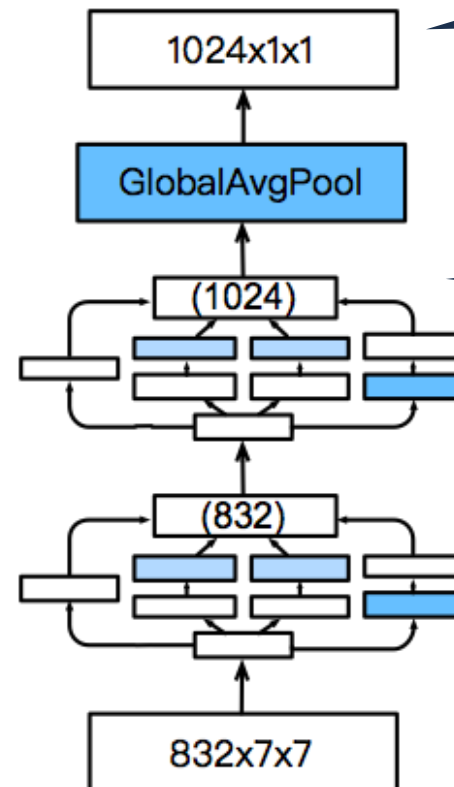
增加输出通道

增加输出通道



3x3 MaxPool,  
stride 2, pad 2

$832 \times 7 \times 7$



$1024 \times 1 \times 1$

GlobalAvgPool

(1024)

(832)

$832 \times 7 \times 7$

# GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

# 许多种类的 Inception 网络

---

- ▶ Inception-BN (v2) - 添加批量归一化
- ▶ Inception-V3 - 修改了初始块
  - ▶ 用多个 3x3 卷积替换 5x5
  - ▶ 用 1x7 和 7x1 卷积替换 5x5
  - ▶ 用 1x3 和 3x1 卷积替换 3x3
  - ▶ 通常用更深的堆
- ▶ Inception-V4 - 添加残差块连接

GluonCV 模型 “动物园”  
[https://gluon-cv.mxnet.io/model\\_zoo/classification.html](https://gluon-cv.mxnet.io/model_zoo/classification.html)

