蘭州大學
LANZHOU UNIVERSITY

卷积神经网络（花书6-7章）

指导教师：　学生：@dearRongerr

2024年03月18日

本章目录

本章目录

计算机视觉

- 图像分类
- 目标检测
- 图像分割
- 目标跟踪
- OCR文字识别
- 图像滤波与降噪
- 图像增强
- 风格迁移
- 三维重建
- 图像检索
- GAN

图像获取

提取二维图像、三维图组、图像序列或相关的物理数据，如声波、电磁波或核磁共振的深度、吸收度或反射度

预处理

对图像做一种或一些预处理，使图像满足后继处理的要 求，如：二次取样保证图像坐标的正确，平滑、去噪等

特征提取

从图像中提取各种复杂度的特征，如：线，边缘提取和脊侦测，边角检测、斑点检测等局部化的特征点检测

检测/分割

对图像进行分割，提取有价值的内容，用于后继处理， 如：筛选特征点，分割含有特定目标的部分

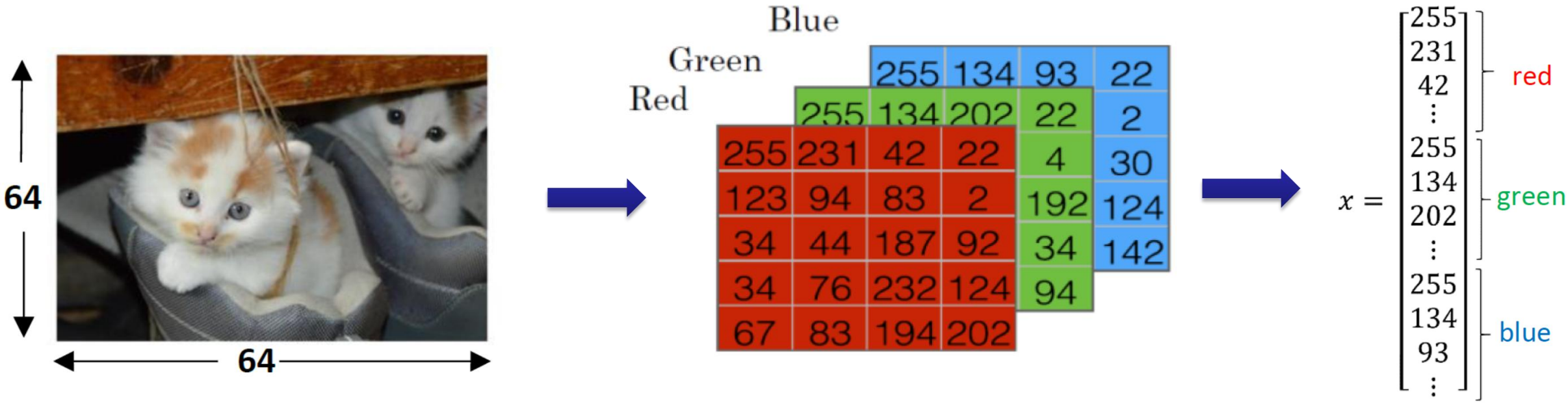
高级处理

验证得到的数据是否匹配前提要求，估测特定系数，对 目标进行分类

图像分类

计算机视觉

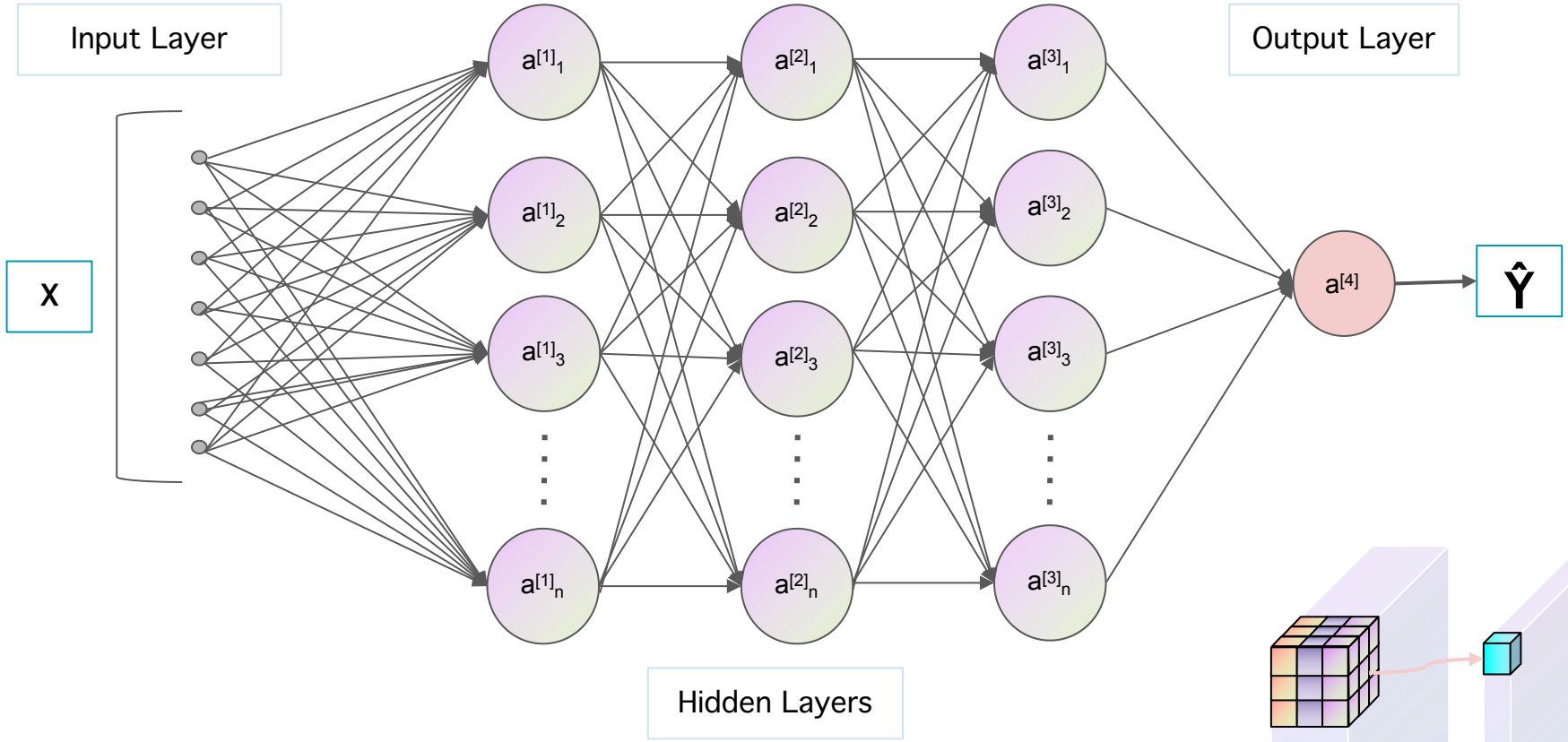
图像的数字表示

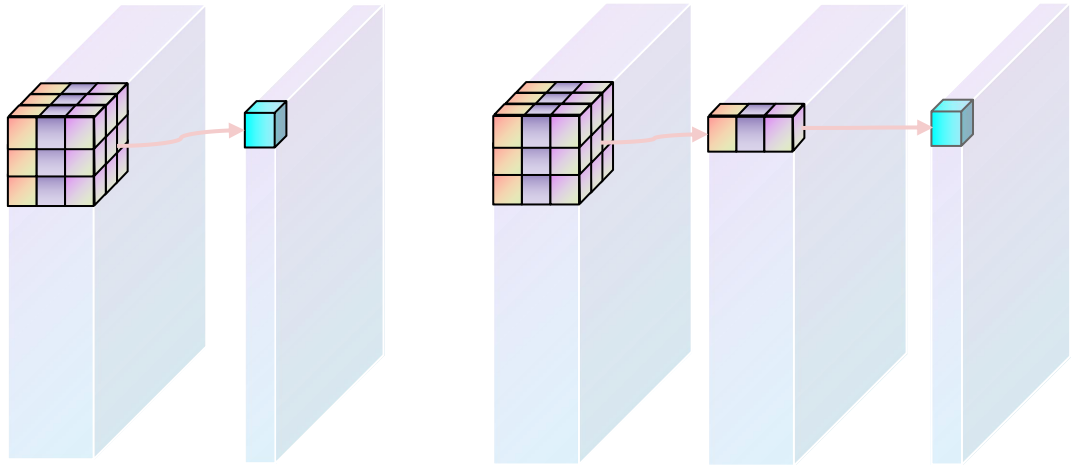


一张图片数据量是64×64×3，因为每张图片都有3个颜色通道。如果计算一下的话，可得知数据量为12288。如果用全连接网络实现分类，那么网络的参数会急剧增加。

本章目录

深层神经网络和卷积神经网络

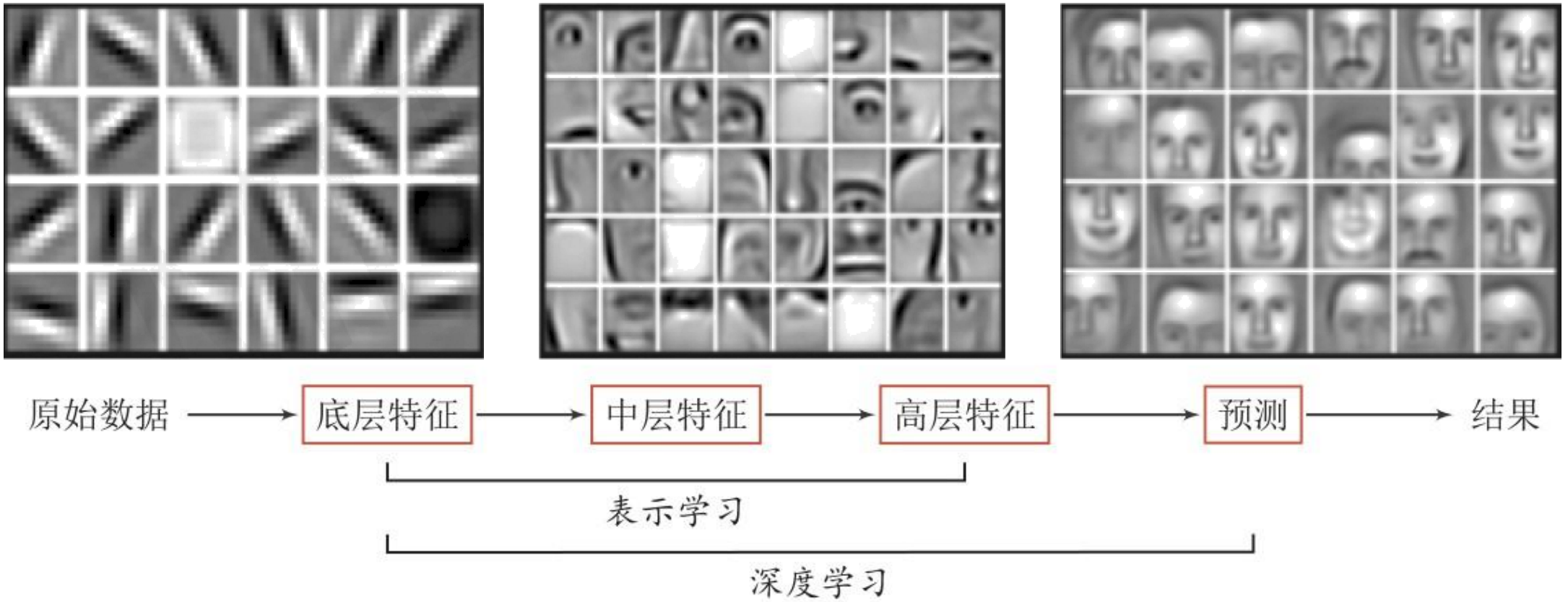




进行卷积计算，处理大量特征

卷积神经网络

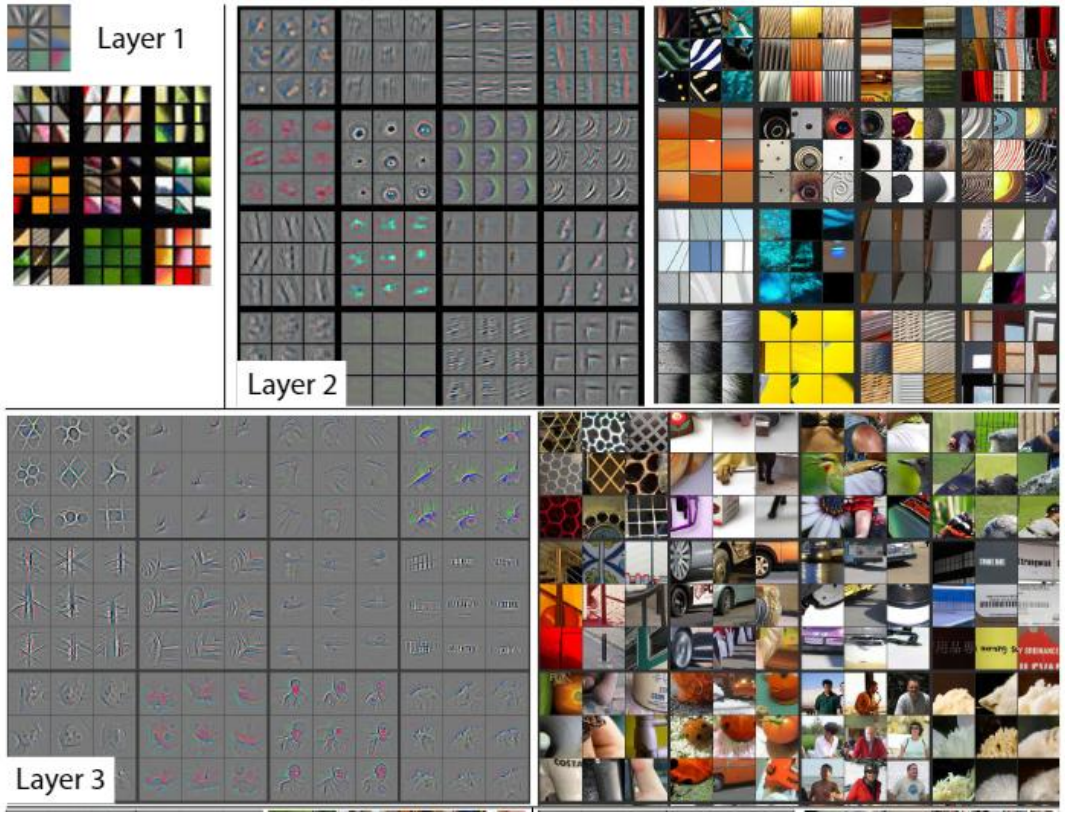
深度学习=表示学习+浅层学习

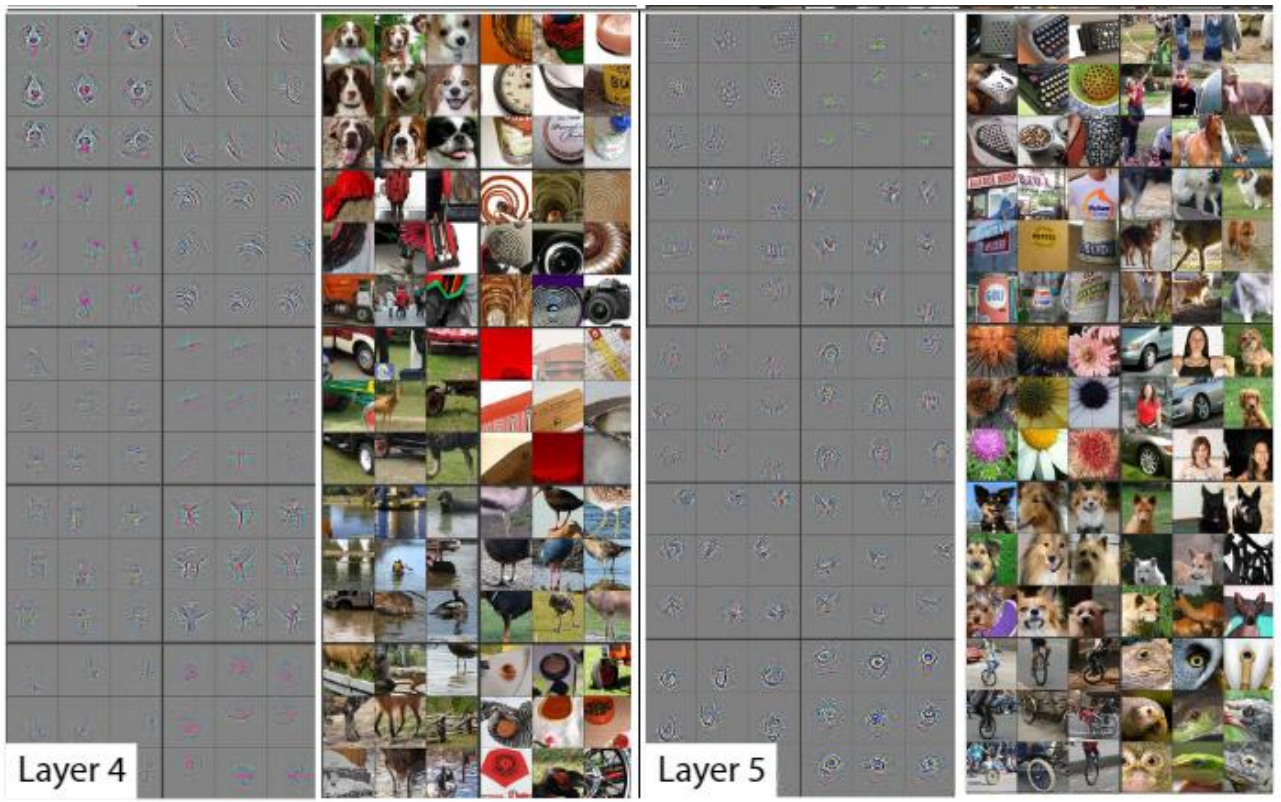


多层卷积能抽取复杂特征

浅层学到的特征为简单的边缘、角点、纹理、几何形状、表面等

深层学到的特征则更为复杂抽象，为狗、人脸、键盘等等





本章目录

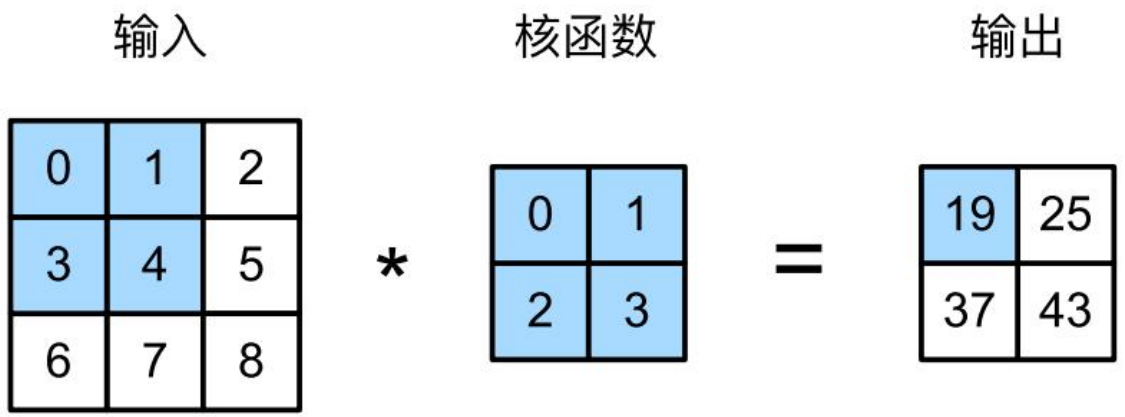
内容

卷积运算

卷积运算

首先，暂时忽略通道（第三维）这一情况，看看如何处理二维图像数据和隐藏表示。

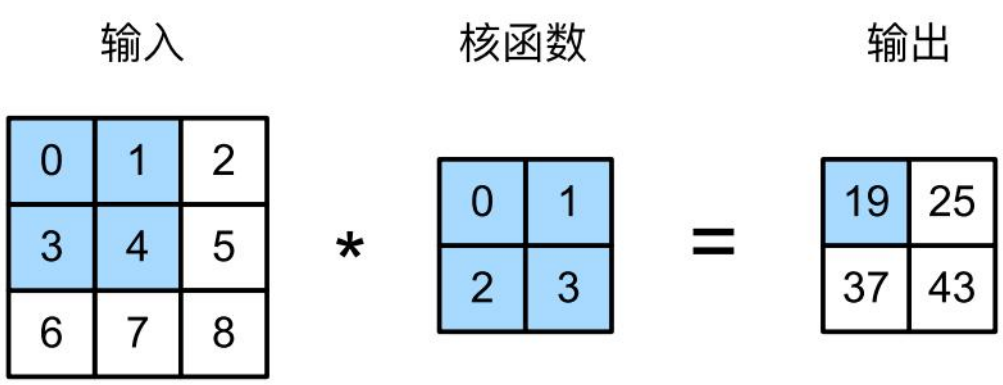
- 输入是高度为3、宽度为3的二维张量（即形状为3 × 3）
- 卷积核的高度和宽度都是2（即形状为2 × 2）



- 阴影部分是第一个输出元素的计算：0 × 0 + 1 × 1 + 3 × 2 + 4 × 3 = 19.

卷积运算

- 卷积窗口从输入张量的**左上角开始，从左到右、从上到下滑动**。
- 当卷积窗口滑动到新一个位置时，包含在该窗口中的**部分张量**与**卷积核张量**进行**按元素相乘**，得到的张量**再求和**得到一个**单一的标量值**，由此得出了这一位置的**输出值**。
- 如上例，输出张量的四个元素由卷积运算得到，这个输出高度为2、宽度为2，



$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$

$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$

$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$

$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

卷积运算输出大小的讨论

note

- **输出大小略小于输入大小**

这是因为卷积核的宽度和高度大于1，而卷积核只与图像中每个大小完全适合的位置进行卷积运算。

- 输出大小 = 输入大小 - 卷积核大小

$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

- 为保证输出大小保持不变：padding（填充）

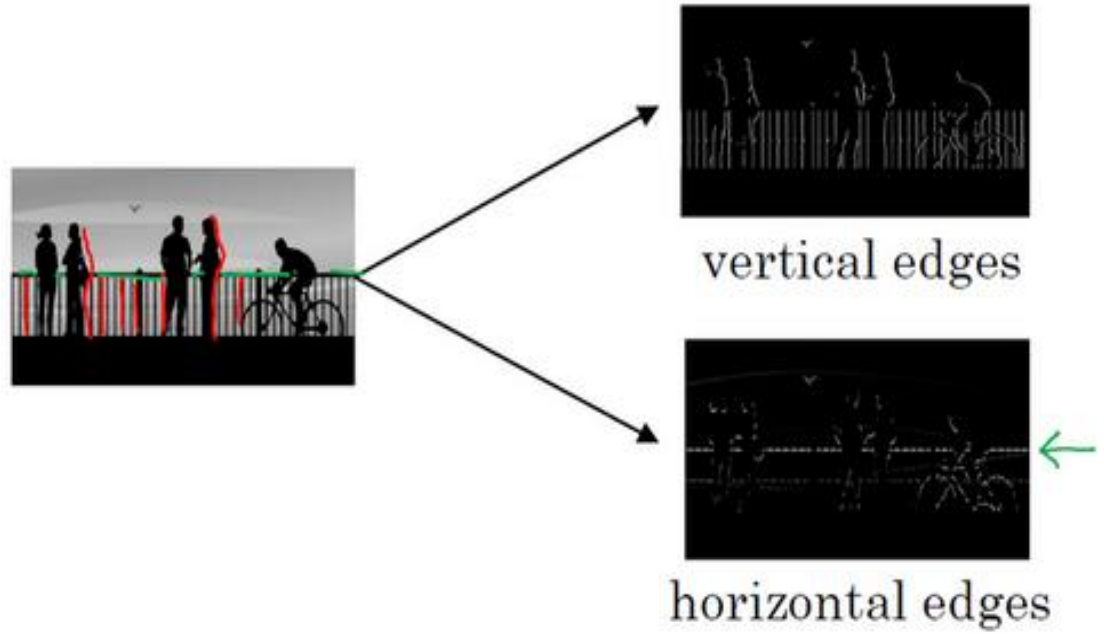
卷积运算输出大小的讨论

- 在上例中，输入的高度和宽度都为 3，卷积核的高度和宽度都为 2，输出维数为 2×2
- 假设输入形状为 $n_h \times n_w$，卷积核形状为 $k_h \times k_w$，那么输出形状将是 $(n_h - k_h + 1) \times (n_w - k_w + 1)$
- 卷积的输出形状取决于输入形状和卷积核的形状

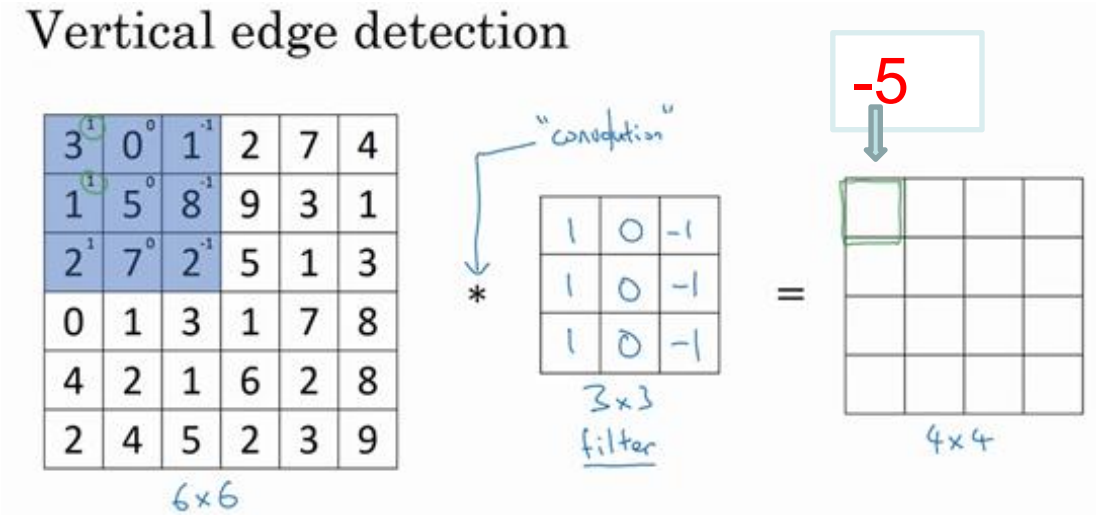
边缘检测

边缘检测

神经网络的前几层是通常检测边缘的，然后，后面的层有可能检测到物体的部分区域，更靠后的一些层可能检测到完整的物体



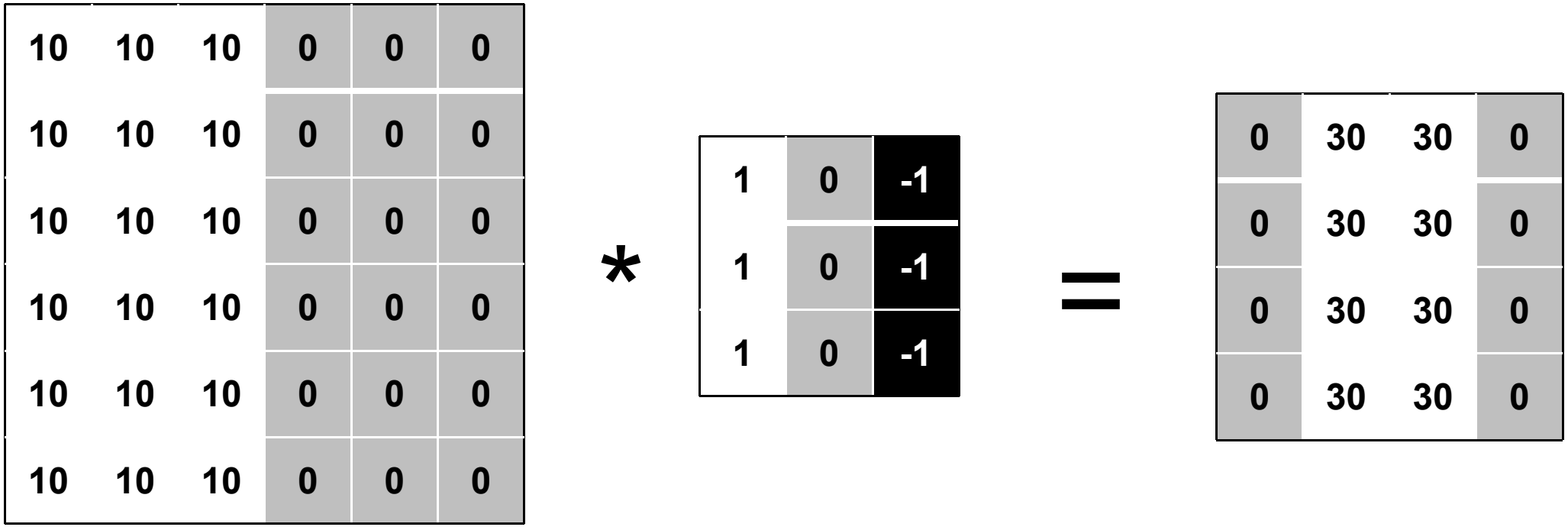
Vertical edge detection



$$\begin{bmatrix} 3\times1 & 0\times0 & 1\times(1) \\ 1\times1 & 5\times0 & 8\times(-1) \\ 2\times1 & 7\times0 & 2\times(-1) \end{bmatrix} = \begin{bmatrix} 3 & 0 & -1 \\ 1 & 0 & -8 \\ 2 & 0 & -2 \end{bmatrix}$$

$$3+1+2+0+0+0+(-1)+(-8)+(-2)=-5$$

图像目标中的边缘检测


10 10 10 0 0 0
10 10 10 0 0 0
10 10 10 0 0 0
10 10 10 0 0 0
10 10 10 0 0 0
10 10 10 0 0 0
*
1 0 -1
1 0 -1
1 0 -1
=
0 30 30 0
0 30 30 0
0 30 30 0
0 30 30 0

图像目标中的边缘检测

10	10	10	10	10	10
10	10	10	10	10	10
10	10	10	10	10	10
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

填充与步幅

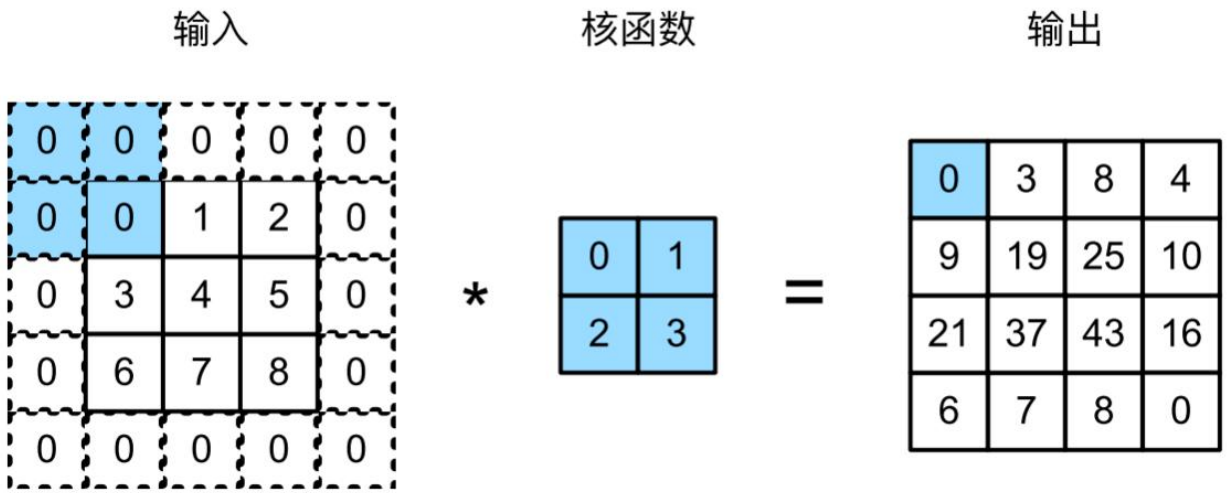
填充与步幅：为什么需要填充和步幅

- 由于卷积核的宽度和高度通常大于 1，所以在应用了连续的卷积之后，我们最终得到的输出远小于输入大小
- eg：一个 240×240 像素的图像，经过 10 层 5×5 的卷积后，将减少到 200×200 像素
- 原始图像的边界丢失了许多有用信息，So“填充”
- 如果我们发现原始的输入分辨率十分冗余，我们希望大幅降低图像的宽度和高度。So“步幅”

什么是填充

- 在输入图像的边界填充元素（通常填充元素是 0）
- 我们将 3×3 输入填充到 5×5，那么它的输出就增加为 4×4。阴影部分是第一个输出元素以及用于输出计算的输入和核张量元素：

$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$。



填充与输入大小的讨论(1/2)

- 如果添加 p_h 行填充（一半在顶部，一半在底部）和 p_w 列填充（左侧一半，右侧一半），则输出形状将为

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- 意味着输出的高度和宽度将分别增加 p_h 和 p_w

填充与输入大小的讨论(2/2)

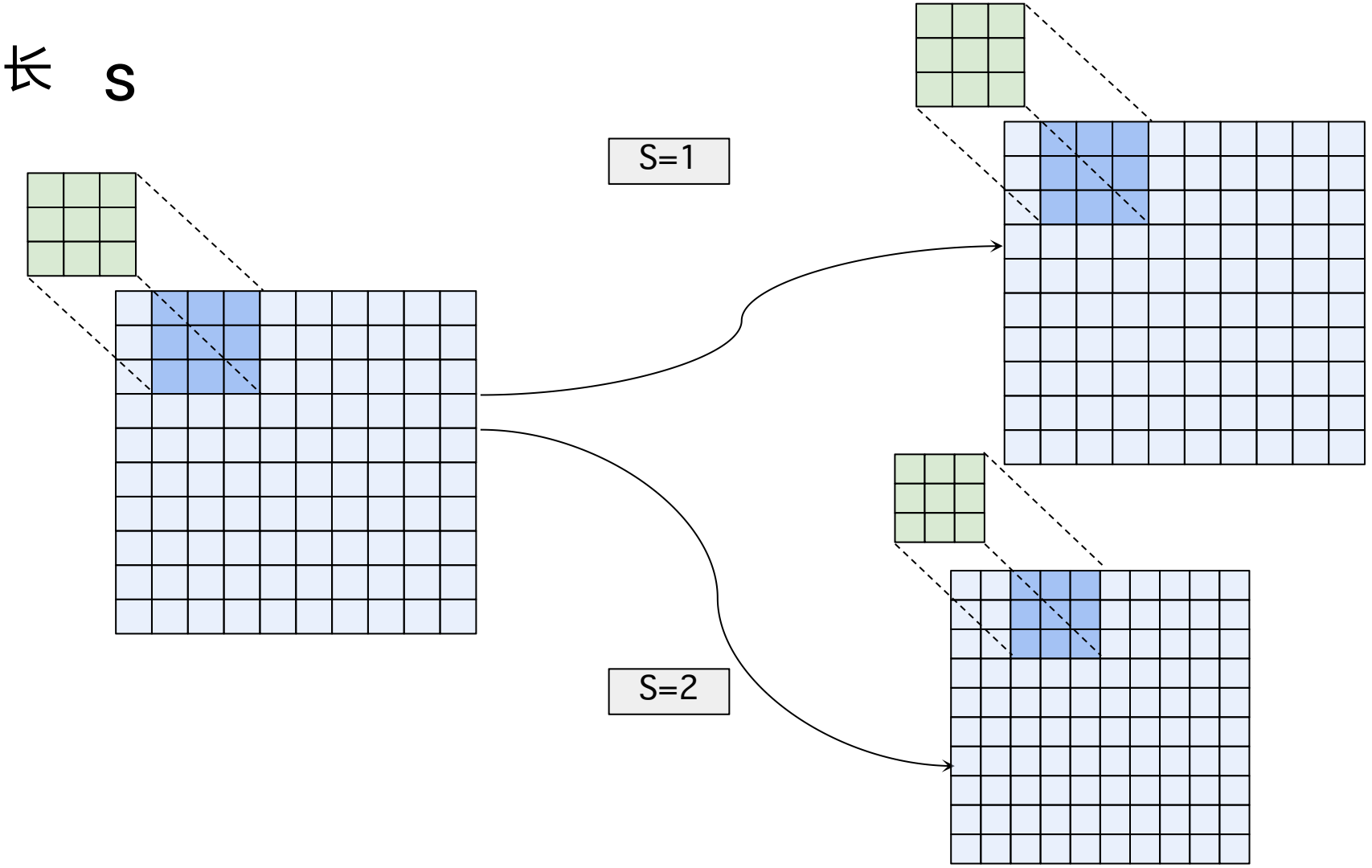
- 在许多情况下，我们设置 $p_h = k_h - 1$ 和 $p_w = k_w - 1$，使输入和输出具有相同的高度和宽度。这样可以在构建网络时更容易地预测每个图层的输出形状。
- 假设 k_h 是奇数，我们将在高度的两侧填充 $p_h/2$ 行。
- 卷积神经网络中卷积核的高度和宽度通常为奇数，例如 1、3、5 或 7。选择奇数的好处是，可以在顶部和底部填充相同数量的行，在左侧和右侧填充相同数量的列。

什么是步幅？

- 在卷积运算时，卷积窗口从输入张量的左上角开始，向下、向右滑动。
- 在前面的例子中，我们默认每次滑动一个元素。
- 但是，有时候为了高效计算或是缩减采样次数，卷积窗口可以跳过中间位置，每次滑动多个元素。
- 将每次滑动元素的数量称为步幅（stride）

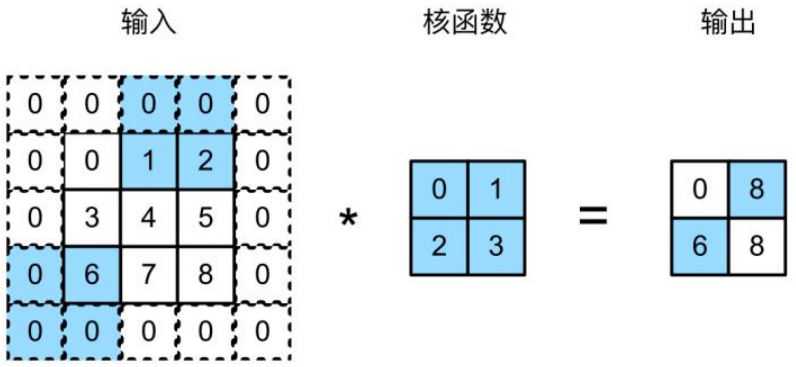
卷积步长的直观理解

卷积步长 S



步幅的应用举例

- 如图是垂直步幅为 3，水平步幅为 2 的卷积运算
- 着色部分是输出元素以及用于输出计算的输入和内核张量元素：
  - $0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$
  - $0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$
- 可以看到，为了计算 输出阵$_{12}$ 和 输出阵$_{21}$，卷积窗口分别向右滑动两列和向下滑动三行。



填充与步幅小结

- 填充可以增加输出的高度和宽度。这常用来使输出与输入具有相同的高和宽。
- 步幅可以减小输出的高和宽，例如输出的高和宽仅为输入的高和宽的 1/n（n 是一个大于 1 的整数）。
- 填充和步幅可用于有效地调整数据的维度。

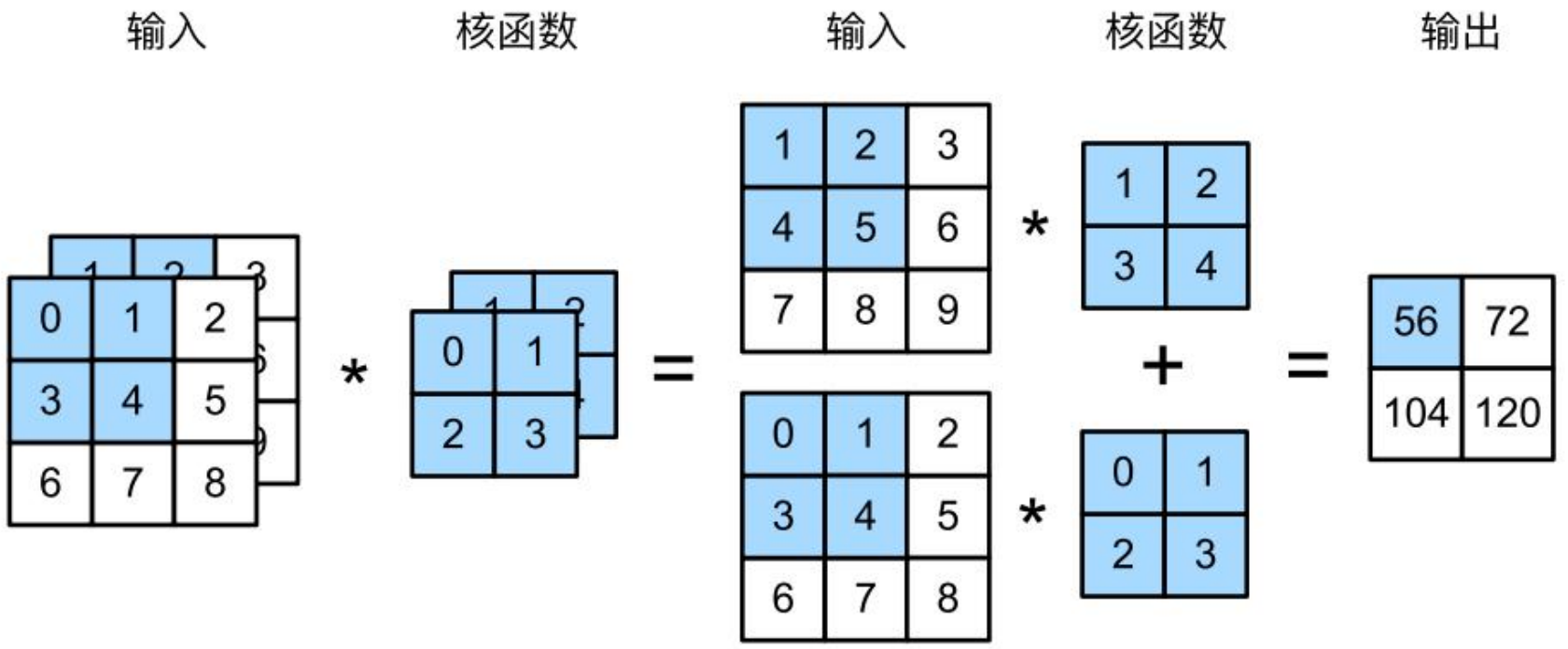
内容

从单输入输出通道→多输入输出通道

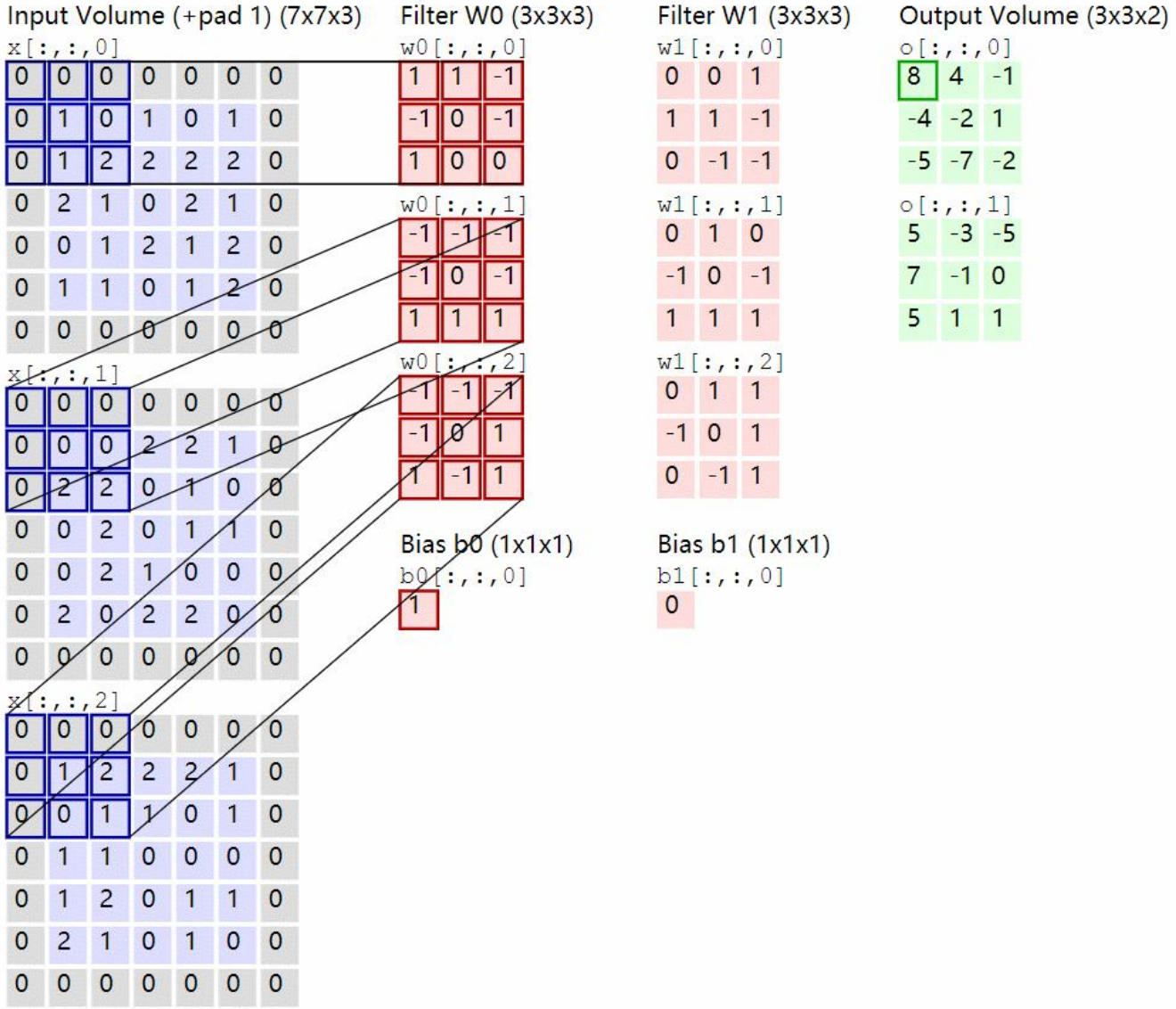
从单输入输出通道→多输入输出通道

- 是到目前为止，仅展示了单个输入和单个输出通道的简化例子，即将输入、卷积核和输出看作二维张量。
- 当添加通道时，输入和隐藏的表示都变成了三维张量。例如，每个 RGB 输入图像具有 $3 \times h \times w$ 的形状。将这个大小为 3 的轴称为通道（channel）维度。

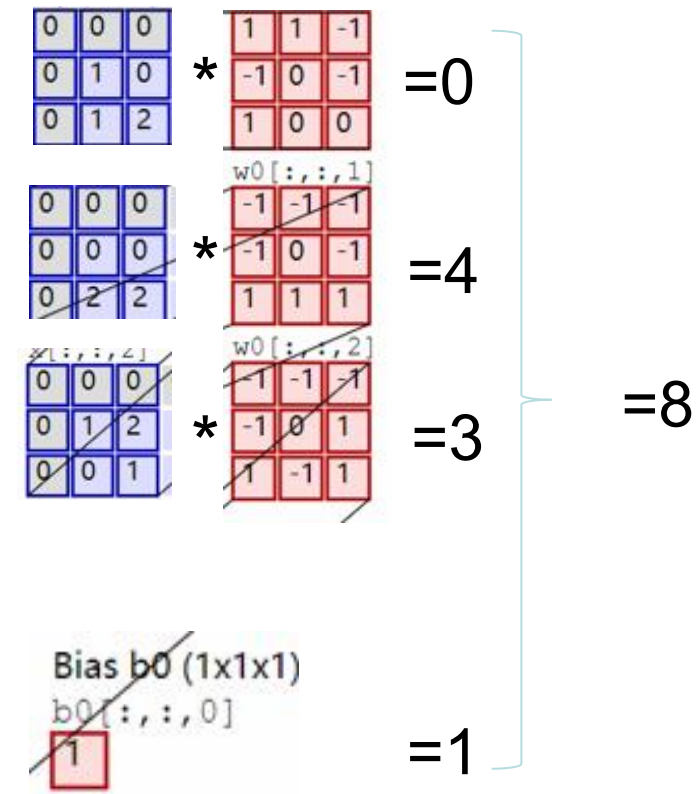
多输入通道举例


输入
核函数
输入
核函数
输出
0 1 2
3 4 5
6 7 8
*
0 1
2 3
=
1 2 3
4 5 6
7 8 9
*
1 2
3 4
+
0 1 2
3 4 5
6 7 8
*
0 1
2 3
=
56 72
104 120

多输出通道举例

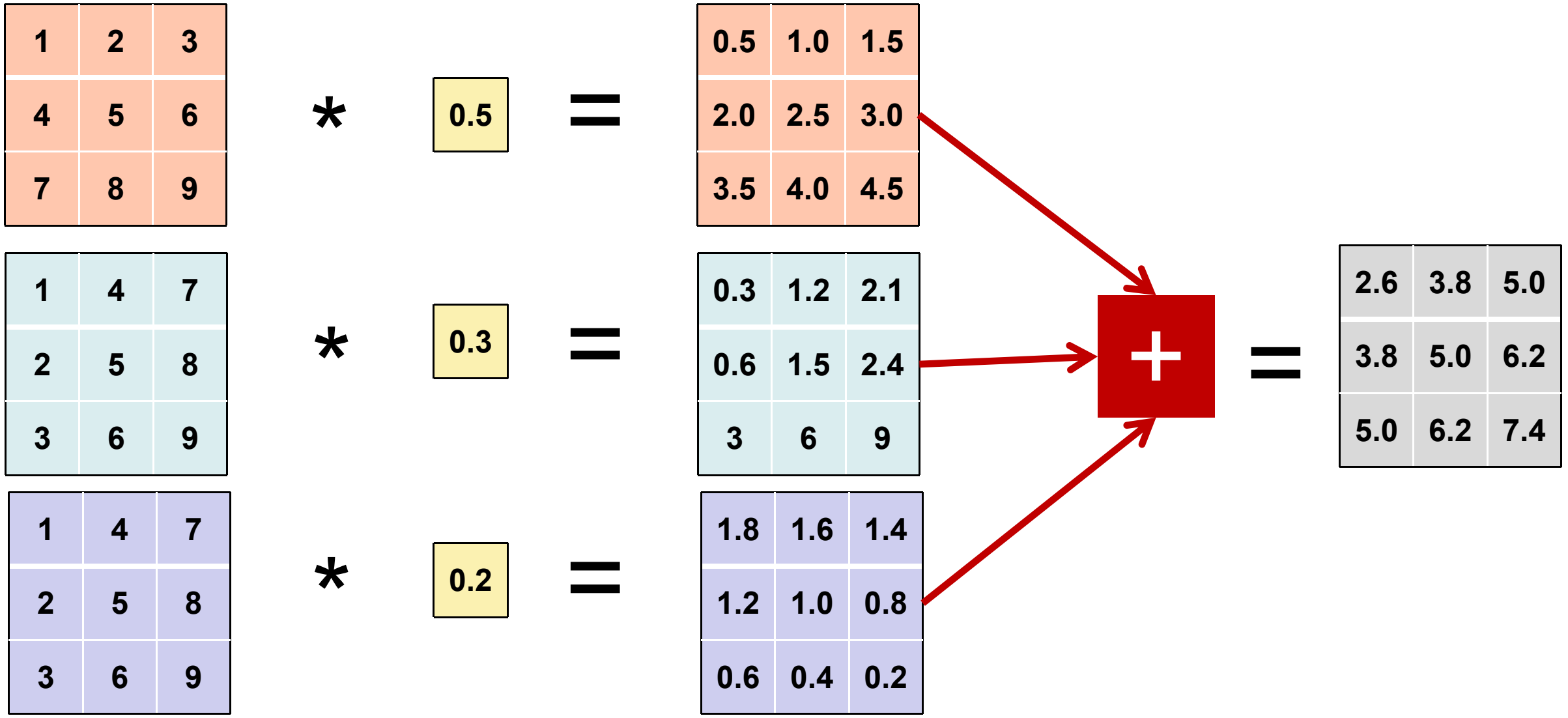


8=0+4+3+1

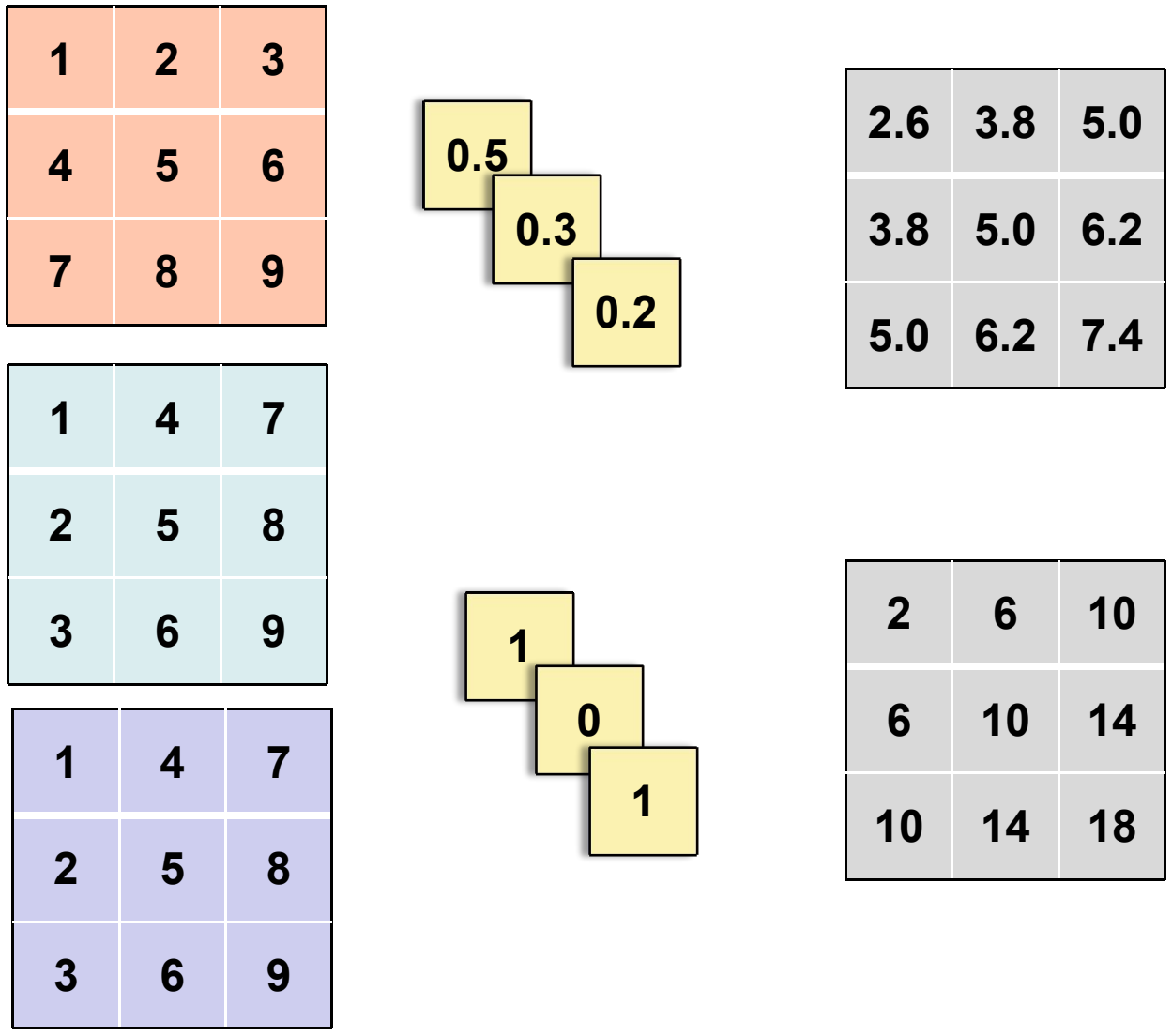


1*1卷积核

1*1 的卷积核是什么？


1 2 3
4 5 6
7 8 9
*
0.5
=
0.5 1.0 1.5
2.0 2.5 3.0
3.5 4.0 4.5
1 4 7
2 5 8
3 6 9
*
0.3
=
0.3 1.2 2.1
0.6 1.5 2.4
3 6 9
1 4 7
2 5 8
3 6 9
*
0.2
=
1.8 1.6 1.4
1.2 1.0 0.8
0.6 0.4 0.2
+
=
2.6 3.8 5.0
3.8 5.0 6.2
5.0 6.2 7.4

1*1 的卷积核是什么？ ⇔ 全连接层

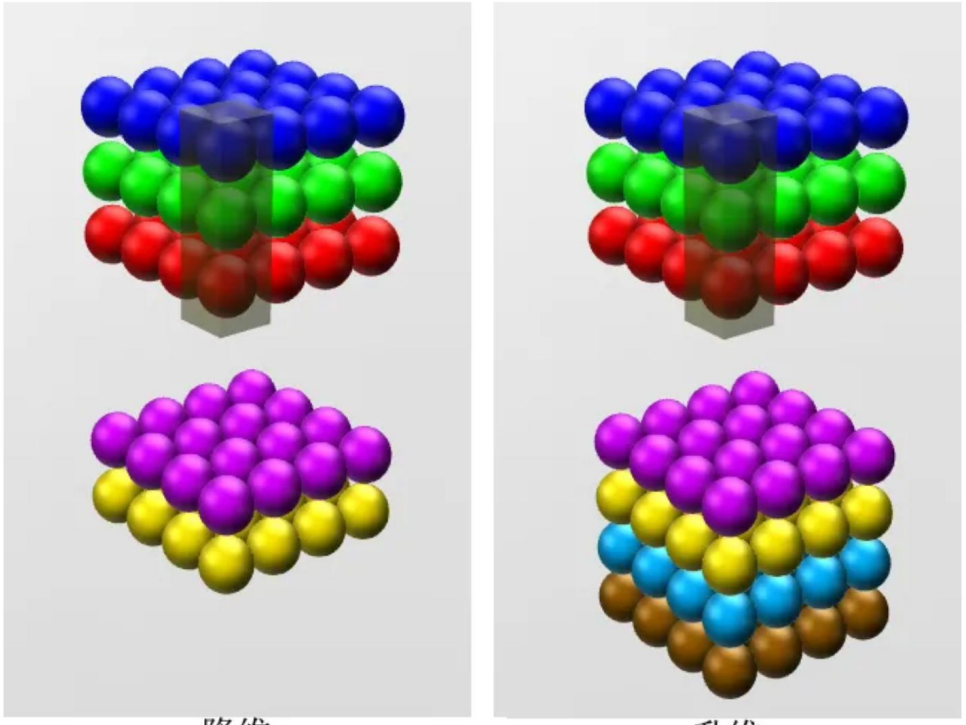


1*1 的卷积核有什么用？

- 升维/降维： 可以改变输出特征图通道数
- 增加非线性映射次数
- 减少卷积核参数： 控制模型复杂度

1*1 的卷积核有什么用？

- 输入特征图的通道数**决定了**卷积核的通道数
- 卷积核的个数**决定了**输出特征图的通道数
- 1*1 的卷积核对维度的改变主要体现在卷积核的个数



特征映射与感受野

特征映射与感受野

特征映射 feature map

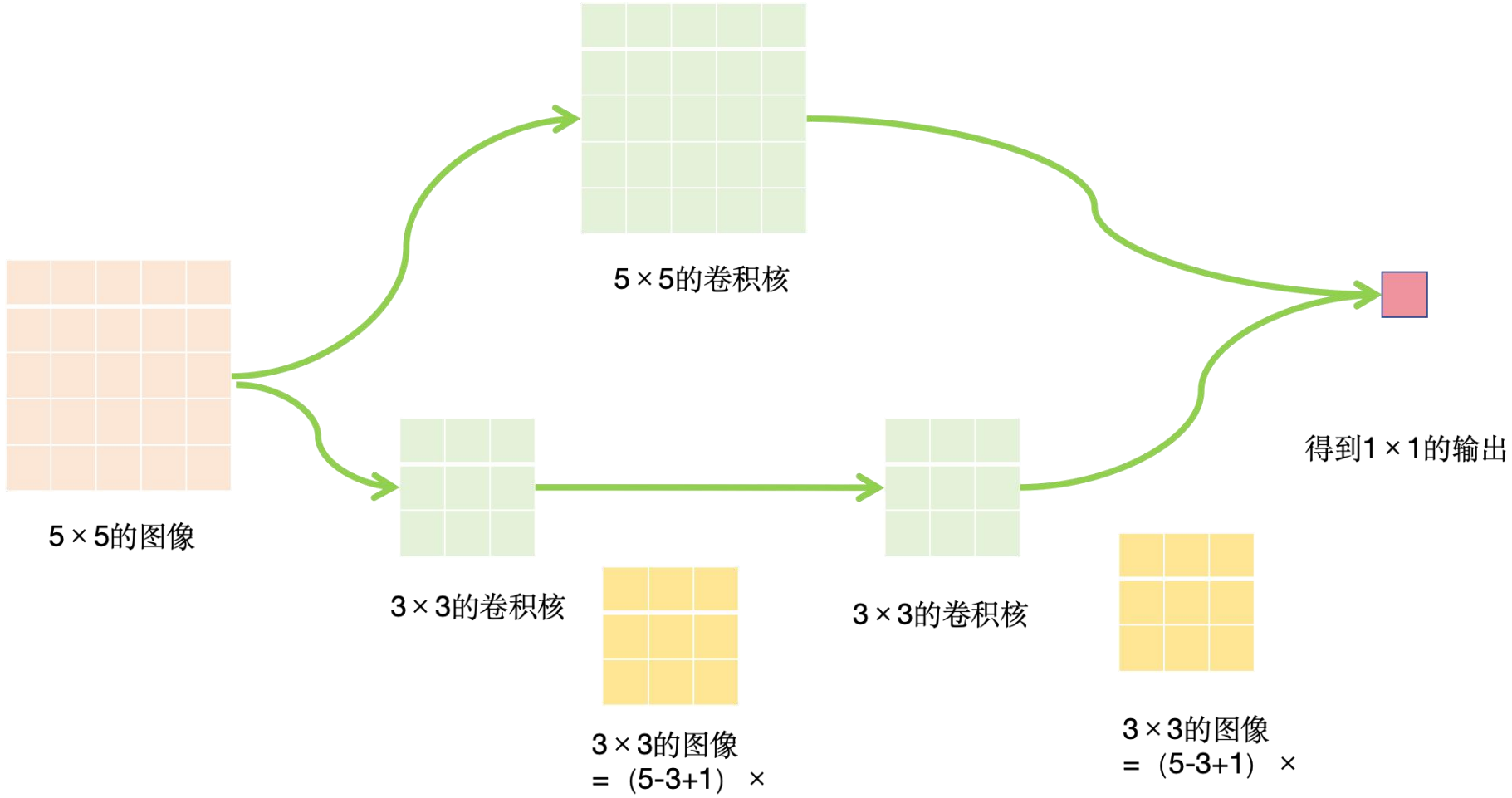
- 卷积层 = 特征映射
- 卷积层可以看做是一个输入映射到下一层的空间维度的转换器

感受野 receptive field

- 来自先前层
- 指在前向传播期间可能影响运算的所有元素

特征映射与感受野 举例

连续的3×3卷积核的使用可以模拟5×5、7×7的感受野



通过上例想说明什么？

- 上面的参数个数：
  - 1个5×5的卷积核＋1个截距项 = 26个参数 || 同时是一个ReLu变换
- 下面的参数个数：
  - 2个3×3的卷积核＋2个截距项 = 20个参数 || 同时是两个ReLu变换

更小的卷积核 好处在于：

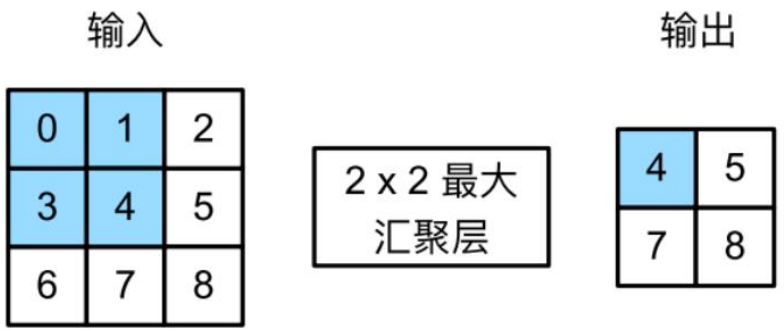
- 可以增加非线性变换
- 可以感受更小的物体
- W的参数可以减少

有关感受野的说明

- 感受野可能大于输入的实际大小
- 举例
  - 给定 2×2 卷积核，阴影输出元素值 19 的感受野是输入阴影元素部分的 4 个元素
  - 设之前的输出为 Y，大小为 2×2，我们在其后附加一个卷积层
  - 该卷积层以 Y 为输入，输出单个元素 z
  - 此时，输出的 z 的感受野包括 Y 的所有四个元素，而输入的 Y 的感受野包括最初所有 9 个输入元素
- 因此当一个特征图中的任意元素需要检测更广区域的输入特征时，我们可以构建一个更深的网络

池　化　层

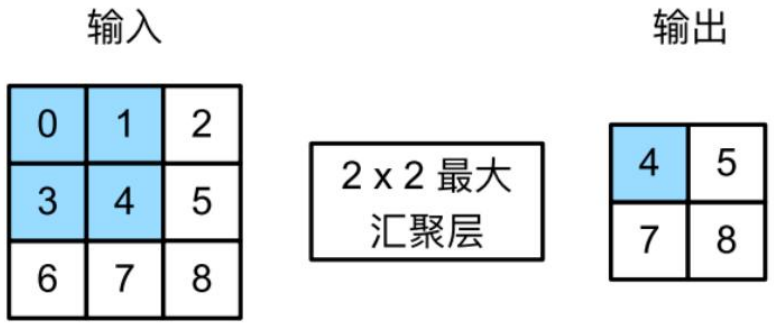
池化层:什么是池化层？



池化层之后的四个元素为每个汇聚窗口的最大值：

- $\max(0,1,3,4) = 4$
- $\max(1,2,4,5) = 5$
- $\max(3,4,6,7) = 7$
- $\max(4,5,7,8) = 8$

池化层:什么是池化层？



池化层之后的四个元素为每个汇聚窗口的最大值：

- $\max(0,1,3,4) = 4$
- $\max(1,2,4,5) = 5$
- $\max(3,4,6,7) = 7$
- $\max(4,5,7,8) = 8$

常用的池化层：①MaxPool②AvgPool

池化层

为什么提出池化层

- **减少过拟合**：Pooling 层可以降低模型对训练数据的过度拟合的风险。通过减少数据维度和参数数量，Pooling 层有助于提高模型的泛化能力，使其更好地适应新的数据。
- **平移不变性**：Pooling 层可以提高模型对输入数据的平移不变性。即使输入数据在空间位置上发生了微小的变化，Pooling 层仍然可以提取出相同的特征，从而使模型更具鲁棒性。
- **提取特征**：Pooling 层能够保留输入数据中最重要的特征，通过池化操作，将原始数据中的冗余信息过滤掉，保留最显著的特征，有助于提高模型的表达能力。

池化层需要注意的点

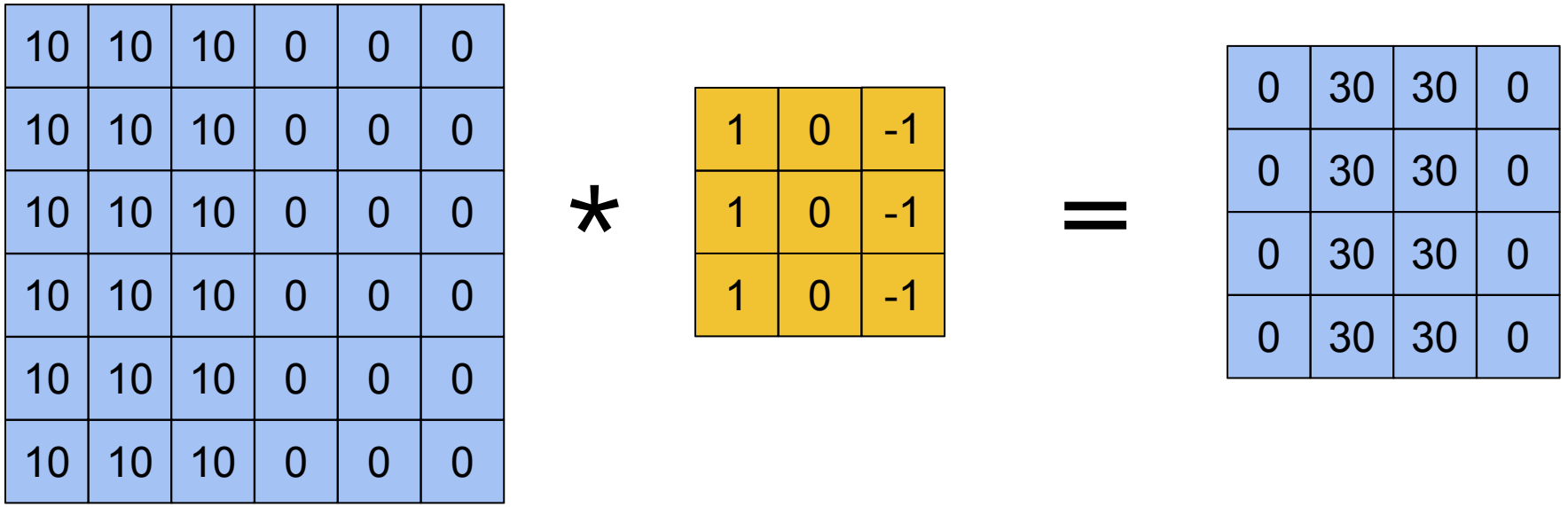
- 对于给定输入元素，最大汇聚层会输出该窗口内的最大值，平均汇聚层会输出该窗口内的平均值
- 汇聚层的主要优点之一是减轻卷积层对位置的过度敏感
- 汇聚层也有和卷积层一样的填充和步幅
- 汇聚层的输出通道数与输入通道数相同!!!!!!（尤其要与卷积层区分）
  - 在处理多通道输入数据时，汇聚层在每个输入通道上单独运算
  - 而不是像卷积层一样在通道上对输入进行汇总
  - 这意味着汇聚层的输出通道数与输入通道数相同。

本章目录

卷积神经网络的作用

卷积神经网络作用

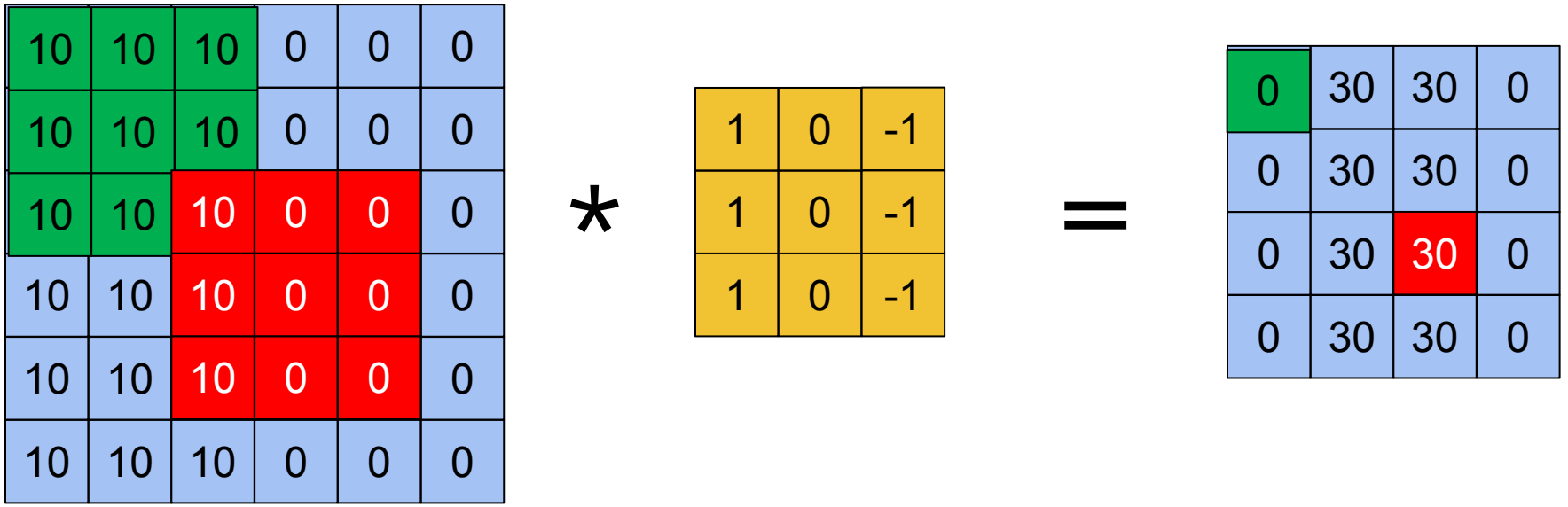
参数共享



如果你用一个3×3的过滤器检测垂直边缘，那么图片的左上角区域，以及旁边的各个区域（左边矩阵中蓝色方框标记的部分）都可以使用这个3×3的过滤器。即使减少参数个数，这9个参数同样能计算出16个输出。

卷积神经网络作用

稀疏连接



右图这个绿色格子的0是通过3×3的卷积计算得到的，它只依赖于这个3×3的输入的单元格，右边这个输出单元（元素0）仅与36个输入特征中9个相连接。而且其它像素值都不会对输出产生任影响，这就是稀疏连接的概念。

卷 积 神 经 网 络 时 间 线

Title TimeLine

时间	网络	
1998	LeNet	基础图像识别网络
2012	AlexNet	深度卷积网络
2013	ZFNet	大型卷积网络
2014	VGG	使用块的网络
2014	GoogLeNet	含并行连接的网络
2015	ResNet	残差网络
2017	DenseNet	稠密连接网络
2016	Darknet-19	
2018	DarkNet-53	
2019	EfficientNetV1	
2021	EfficientNetV2	
2020	CSPNet	跨阶段局部网络

经典网络

轻量化网络

时间	网络
2016	SqueezeNet
2017，2018，2019	MobileNetv1/2/3
2017	ShffleNet
2017	Xception
2020	GhostNet

常用的卷积神经网络架构

- LeNet
- AlexNet
- VGG
- NiN
- GoogLeNet
- ResNet
- DenseNet

常用的卷积神经网络架构

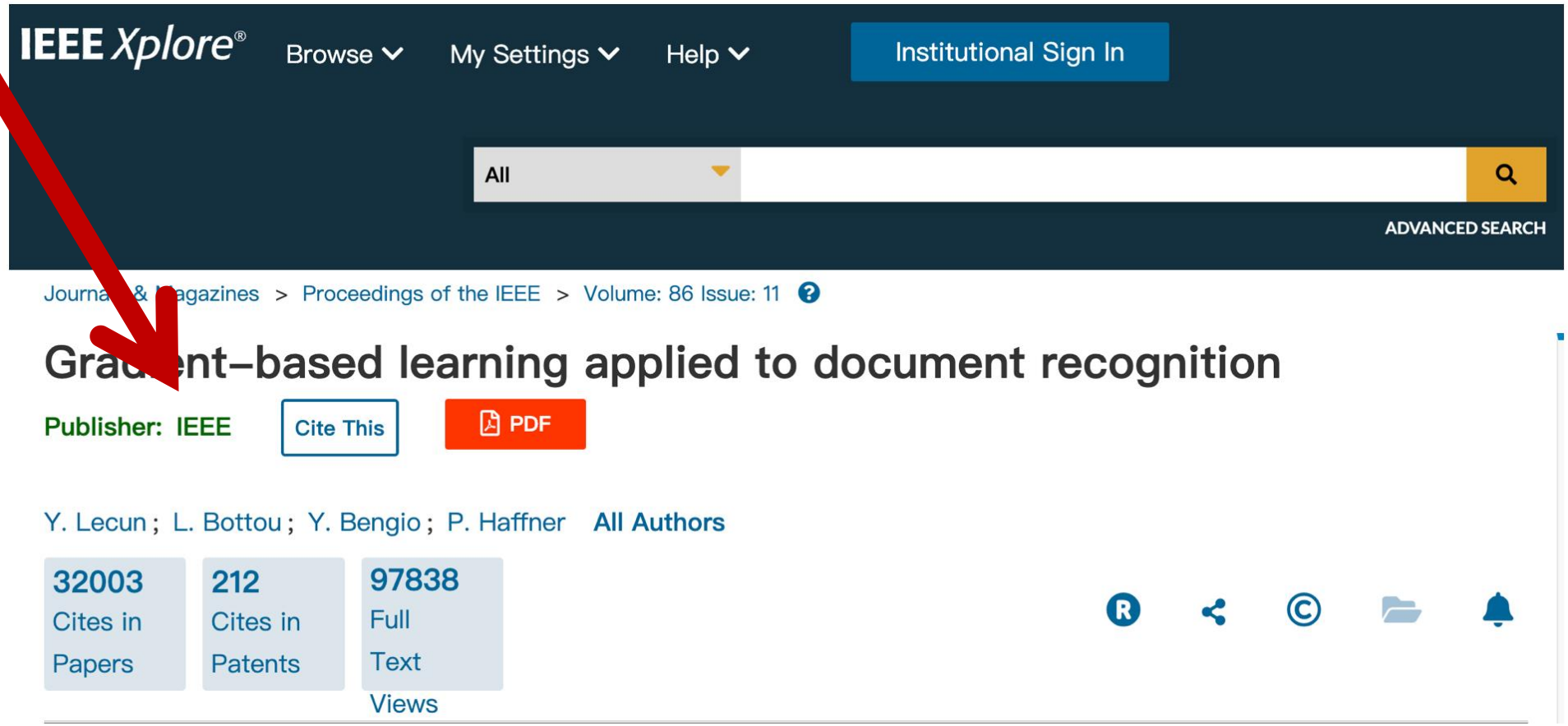
- LeNet：1998年Yann LeCun等《Gradient-Based Learning Applied to Document Recognition》发表在IEEE
- AlexNet：2012年Alex Krizhevsky等《ImageNet Classification with Deep Convolutional Neural Networks》 赢得了2012年ImageNet图像识别挑战赛
- VGG：2014年牛津大学计算机视觉组合和Google DeepMind公司研究员等研发表《Very Deep Convolutional Networks for Large-Scale Image Recognition》

常用的卷积神经网络架构

- NiN：2014年新加坡国立大学的MinLin 等《Network In Network》GoogLeNet：2014年Google 团队 《Going Deeper with Convolutions》
- ResNet：2015年微软实验室中的何恺明等《: Deep Residual Learning for Image Recognition》
- DenseNet：2017年 《Densely Connected Convolutional Networks》作者来自康奈尔大学、清华大学、_x0008_FaceBook AI研究员

内容

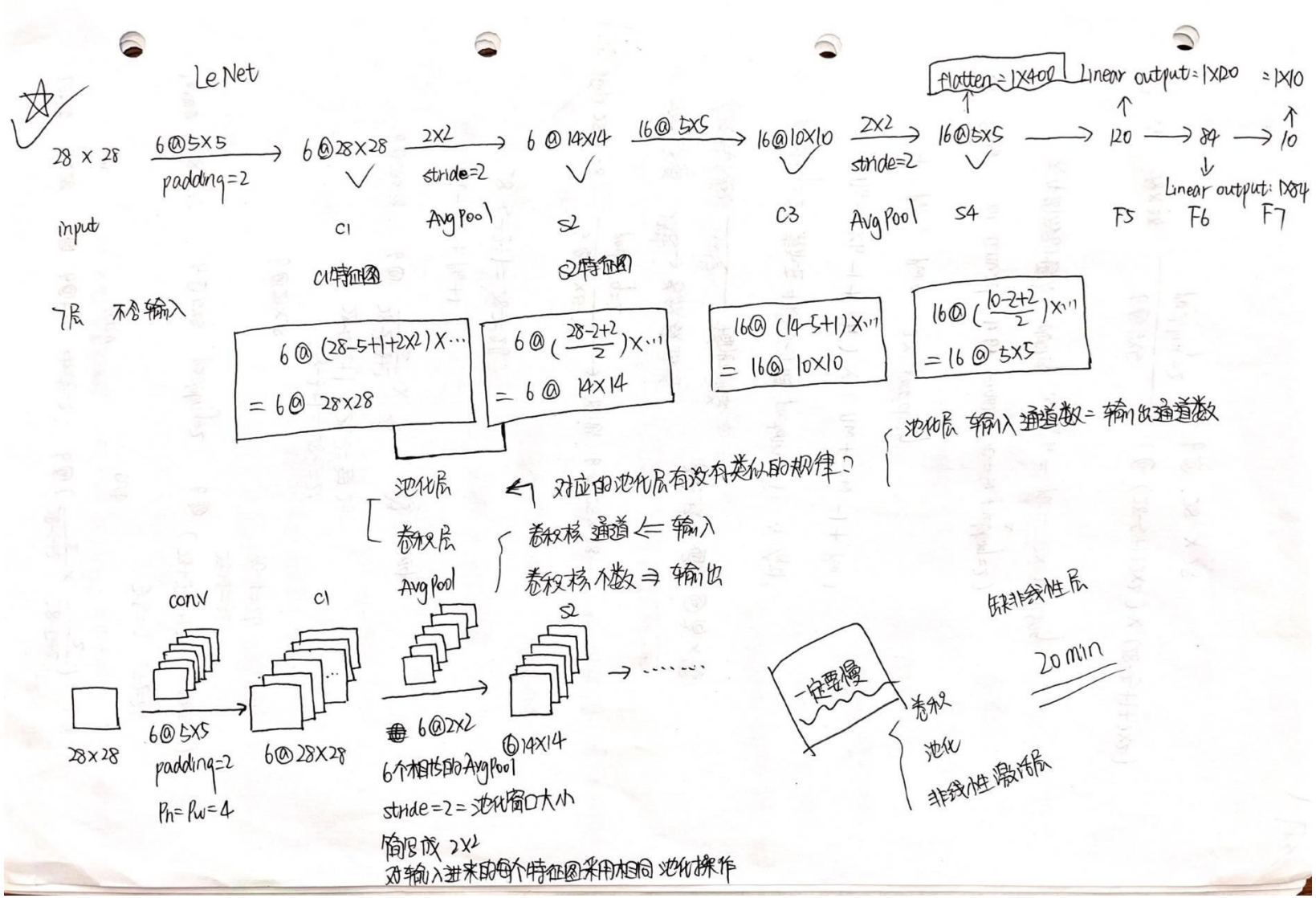
L e N e t


IEEE Xplore®
Browse
My Settings
Help
Institutional Sign In
All
ADVANCED SEARCH
Journals & Magazines > Proceedings of the IEEE > Volume: 86 Issue: 11
Gradient-based learning applied to document recognition
Publisher: IEEE
Cite This
PDF
Y. Lecun ; L. Bottou ; Y. Bengio ; P. Haffner All Authors
32003
Cites in Papers
212
Cites in Patents
97838
Full Text Views

内容

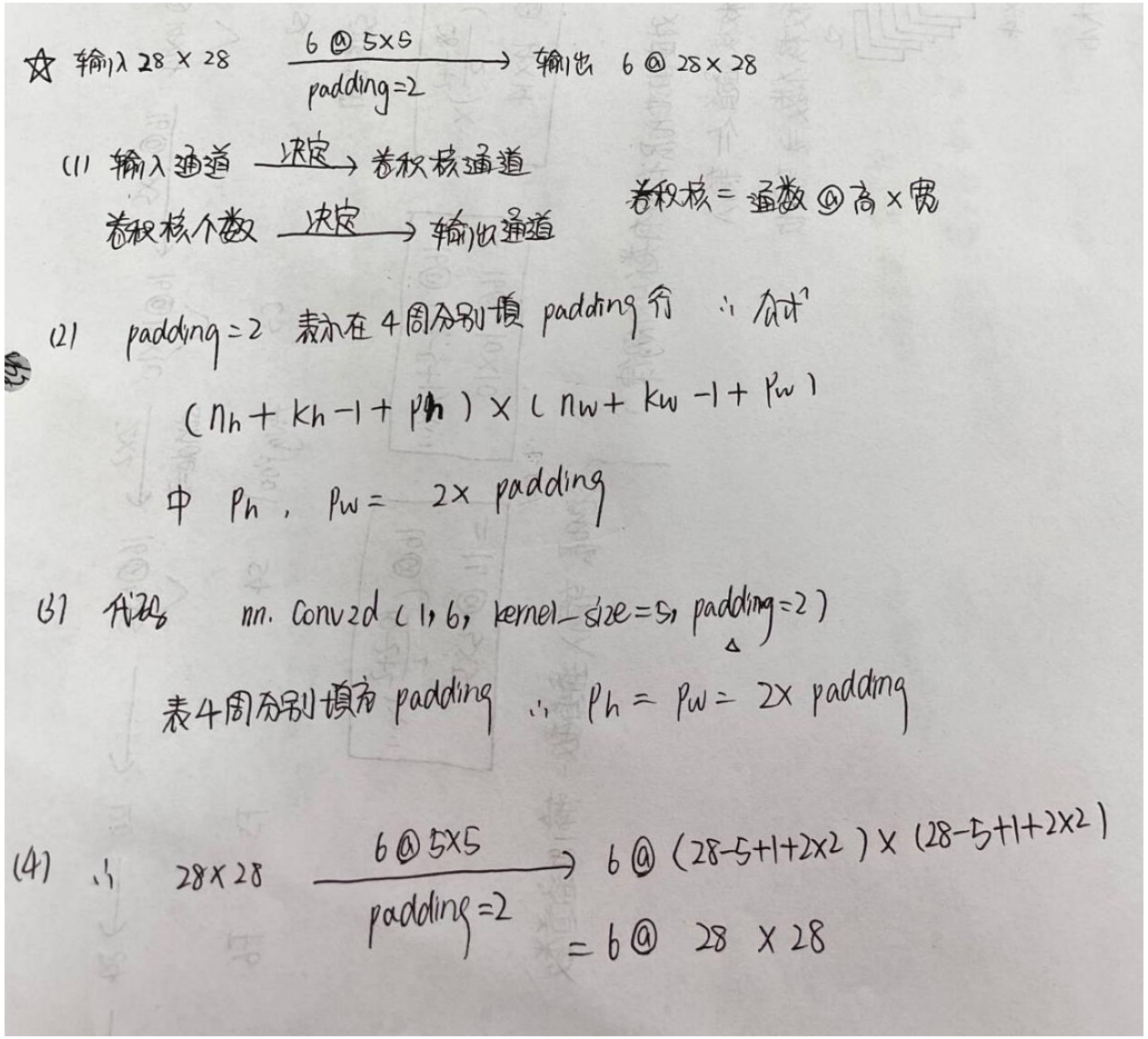
LeNet是什么？

LeNet是什么？（网络架构1/4）

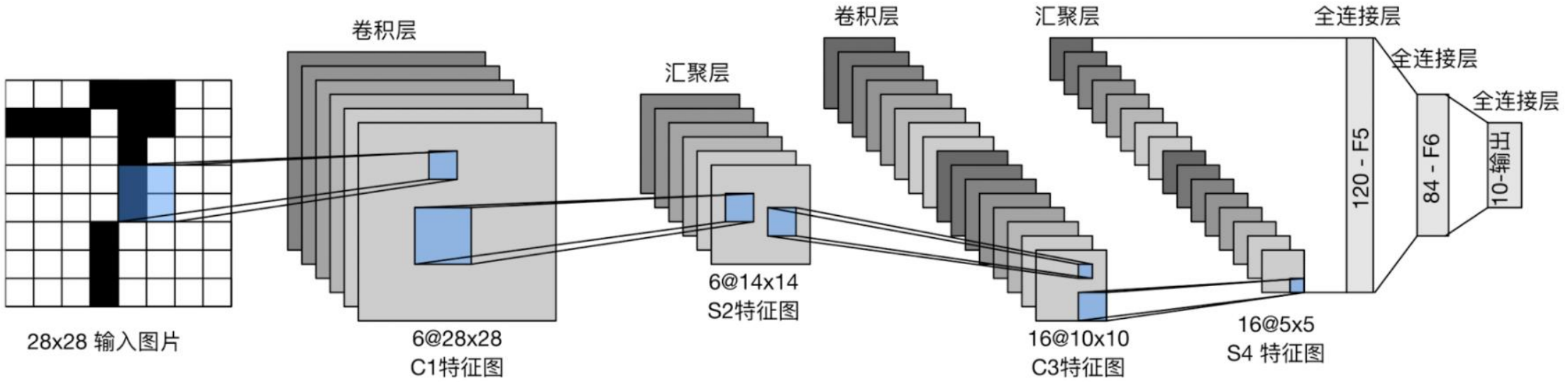


LeNet是什么？（网络架构2/4）

LeNet第一个卷积层详解



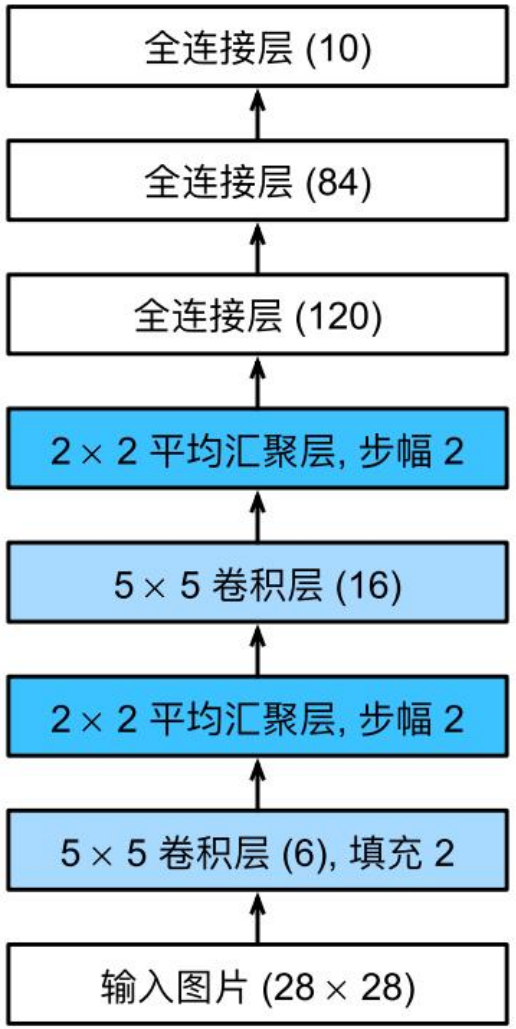
LeNet是什么？（网络架构3/4）



- LeNet（LeNet-5）由两个部分组成：
- 卷积编码器：由两个卷积层组成
- 全连接层密集块：由三个全连接层组成

note：这个网络是5层，池化层和输入层不计入层数（∵不需要计算参数

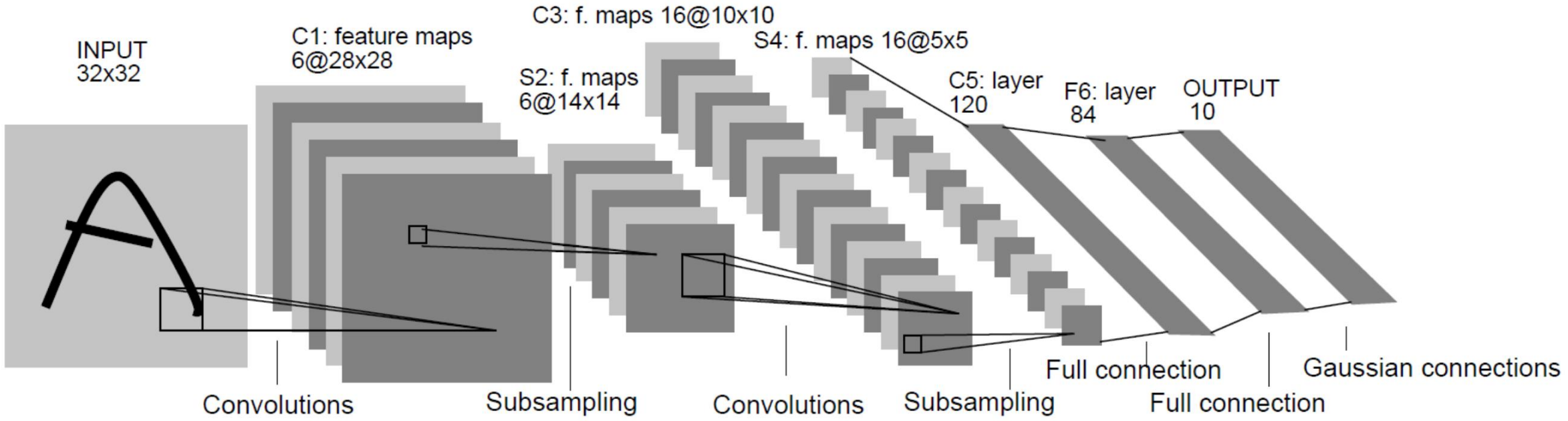
LeNet是什么？（网络架构4/4）



通过以上几张图片想说明的问题：

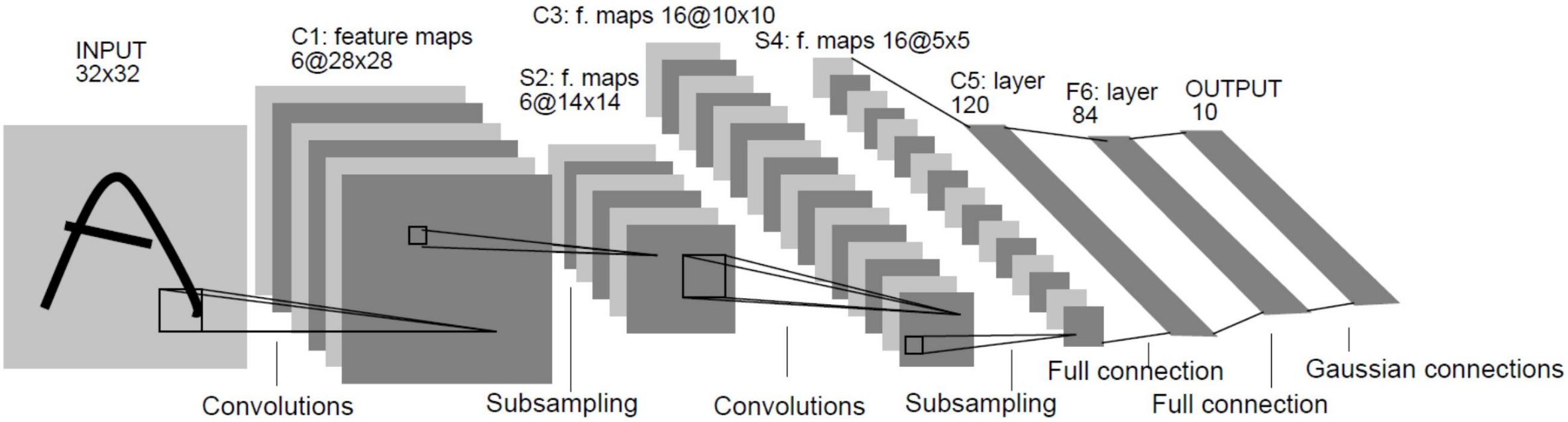
- 完整的卷积神经网络包括：输入→操作→输出
  - 但都简化
  - 要么省略 操作
  - 要么省略 输出
- 操作：Conv、Pooling、非线性激活层

LeNet举例



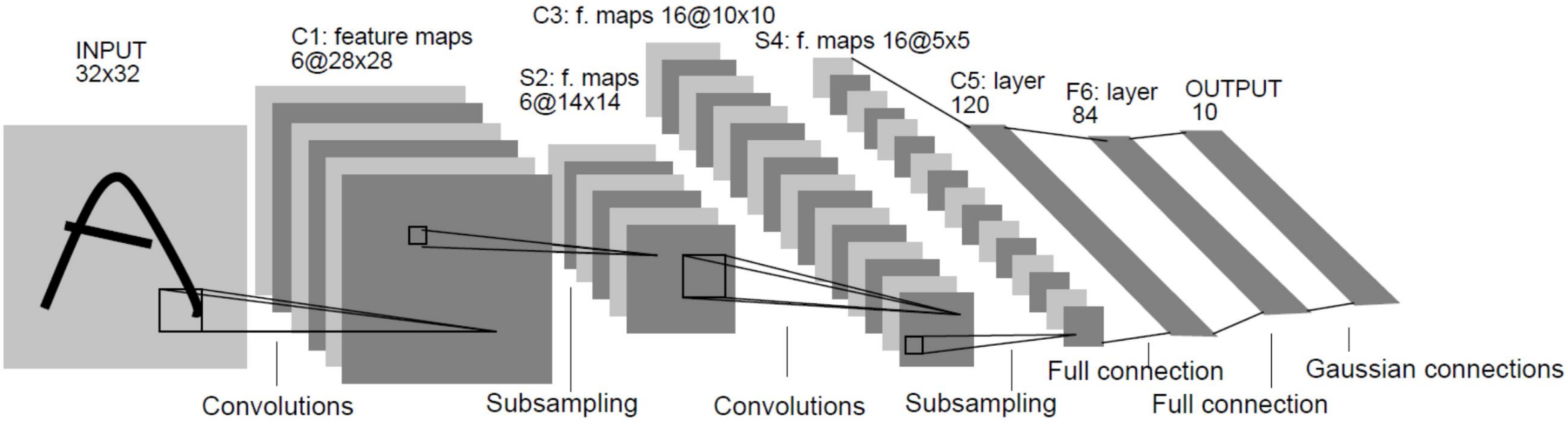
- 输入层：32x32的灰度图像。
- 卷积层C1：6个大小为5x5的卷积核，步长为1，输出为28x28x6。
- 池化层S2：2x2大小的池化层，使用的是平均池化，步长为2，结果通过Sigmoid非线性化，输出为14×14x6。

LeNet举例



- 卷积层C3：16个大小为5x5的卷积核，步长为1。输出为10x10x16。注意，这16个卷积核并不是扫描前一层所有的6个通道。
- 池化层S4：2x2大小的池化层，使用的是平均池化，步长为2，结果通过Sigmoid非线性化，输出为5x5×16。

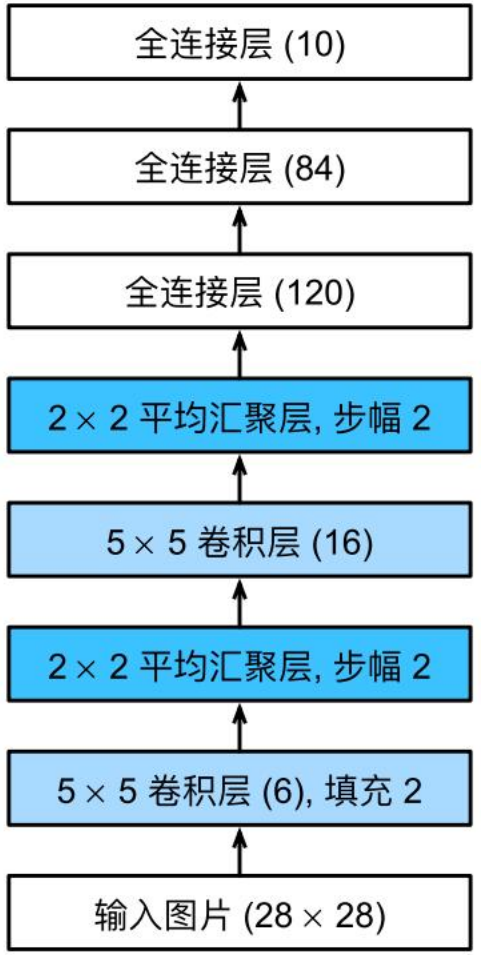
LeNet举例



- 卷积层C5：120个大小为5x5的卷积核。步长为1，输出为1x120。相当于一个全连接层，实现时可用全连接层代替。
- 全连接层F6：84个神经元。
- 输出层：10个神经元 （10分类问题）

LeNet怎么实现？


```
net = nn.Sequential(
nn.Conv2d(1, 6, kernel_size=5, padding=2), nn.Sigmoid(),
nn.AvgPool2d(kernel_size=2, stride=2),
nn.Conv2d(6, 16, kernel_size=5), nn.Sigmoid(),
nn.AvgPool2d(kernel_size=2, stride=2),
nn.Flatten(),
nn.Linear(16 * 5 * 5, 120), nn.Sigmoid(),
nn.Linear(120, 84), nn.Sigmoid(),
nn.Linear(84, 10))
```



LeNet的优缺点

LeNet优点

- **卷积结构**：LeNet 是最早引入卷积层和池化层的神经网络之一，利用卷积操作和参数共享的特性有效地减少了需要学习的参数数量，降低了模型复杂度。
- **层级结构**：LeNet 通过多层卷积和池化层的叠加，实现了对输入数据特征的逐层提取和抽象，从而能够捕获不同层次的特征信息，提高了模型的表征能力。
- **平移不变性**：由于 LeNet 中采用了卷积操作和池化操作，使得网络对于输入数据的平移具有一定的不变性，即无论对象在图像中的位置如何变化，网络都可以识别相同的特征。

LeNet 作为卷积神经网络的先驱之一，具有卷积结构、层级结构、平移不变性等优点，为深度学习和图像处理领域的发展做出了重要贡献。

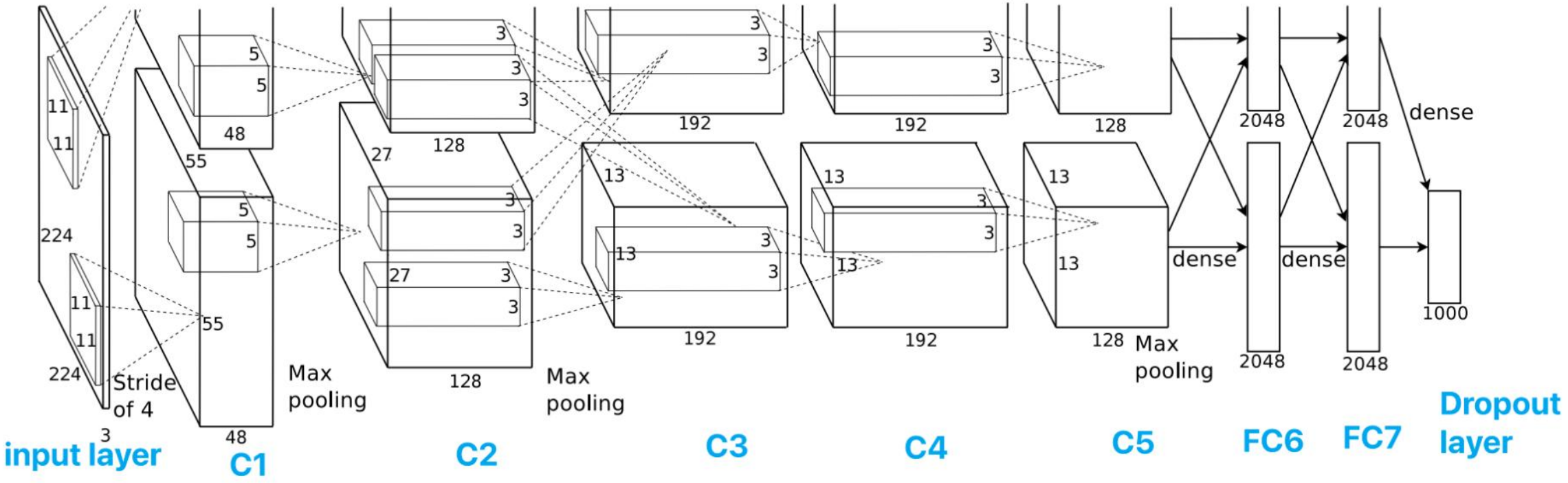
LeNet的不足

	LeNet	后来的CNN网络
网络层连接方式	conv → pool → 激活	conv → 激活 → pool
激活函数	Sigmoid	tanh、relu（most）、leakly
POOL	AvgPooL	MaxPooL
全连接层的连接方式	Gaussian Connection	Softmax

内容

AlexNet

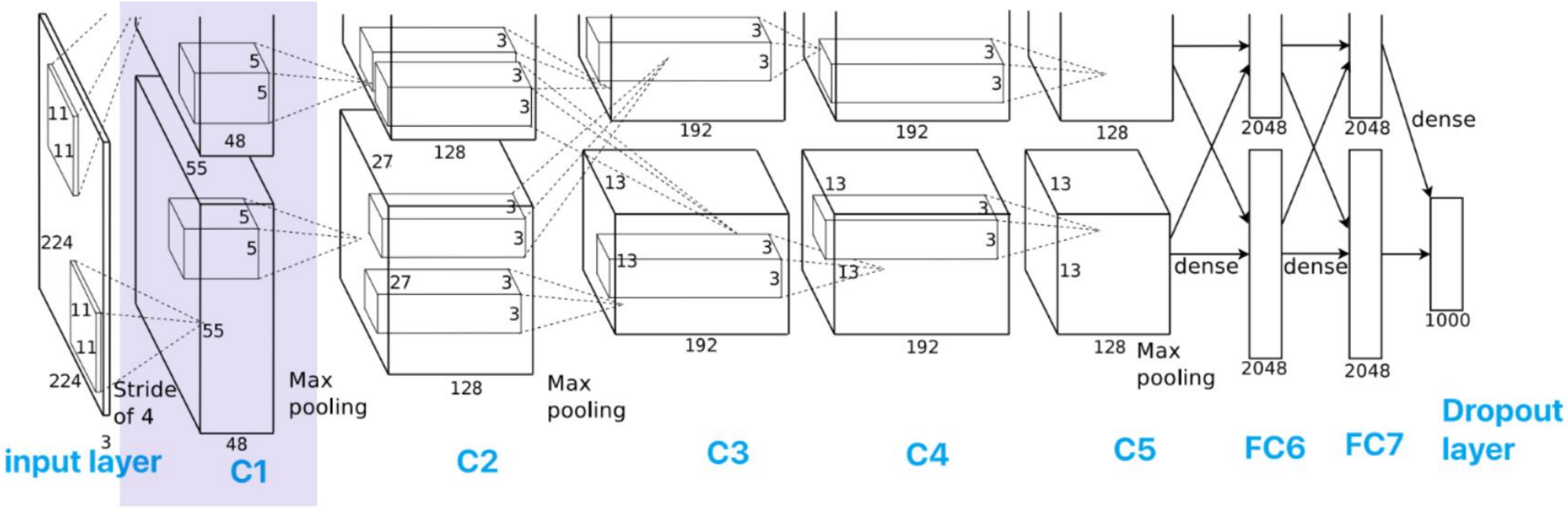
AlexNet 是什么？



- 在两个GPU上跑
- 卷积层 C2、C4、C5中的卷积核只和位于同一GPU的上一层的FeatureMap相连，C3的卷积核与两个GPU的上一层的FeautureMap都连接

AlexNet 是什么？（第一层讲解C1）

■ 第一层讲解 C1



AlexNet 是什么？（第一层讲解C1）

这一层要做：

conv→ReLU→LRN→Pooling

输入： 227*227*3

操作：

（卷积）96@11*11*3 padding=0 stride=4

计算输出：(227-11+2*0+4)/4 = 55

输出：55*55*96

（ReLu）卷积层输出的特征图输入到ReLu函数

（LRN）

- 对局部神经元创建竞争机制，s.t.响应比较大的值变得相对更大，并抑制其他反馈比较小的神经元，增强模型的泛化能力
- **LRN的输出：55*55*96（输出不变）**
- 输出分为2组，每组大小：55*55*48，分别位于单个GPU上

AlexNet 是什么？（第一层讲解C1）

（池化）

- 输入：**55*55*96（分为两组：**55*55*48）
- 使用3*3，stride=2的池化单元
- （重叠池化）stride＜pooling_size
- **根据公式计算输出**：

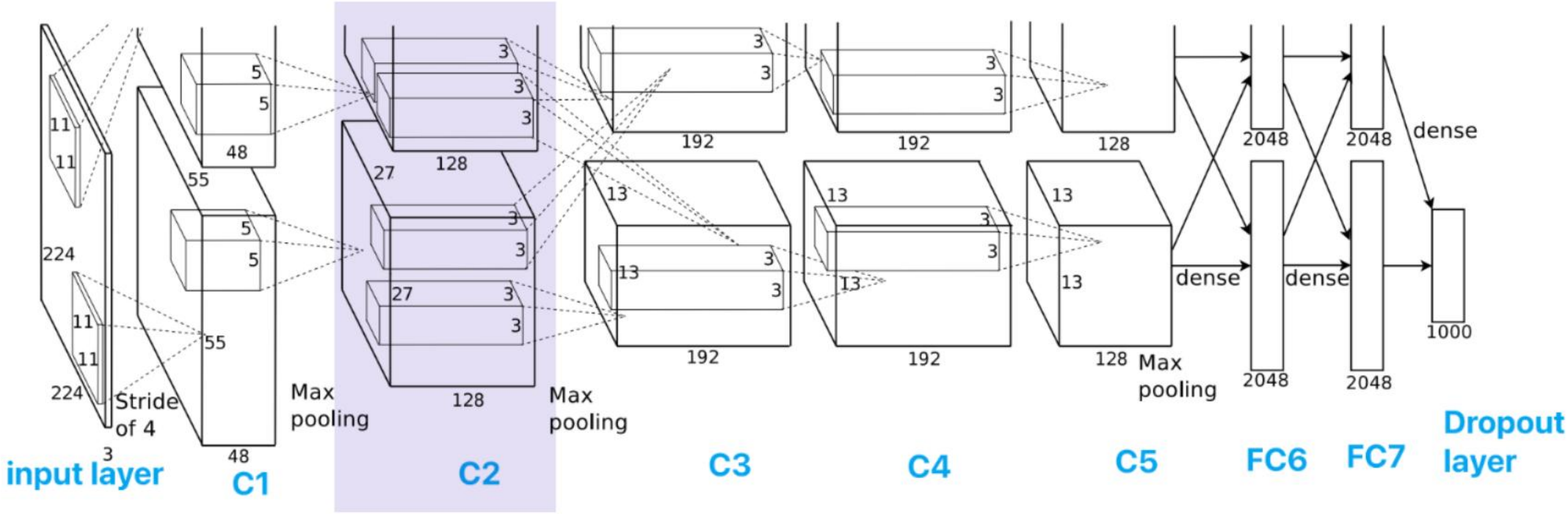
(55-3+2*0+2)/2=27

27*27*96（输出两组：27*27*48）

小结论吧：pool_size=3 stride=2 的池化层，池化以后输出尺寸的长宽各减少一半

AlexNet 是什么？（第二层讲解C2）

第二层讲解 C2



该层的处理流程是：**卷积-->ReLU-->局部响应归一化（LRN）-->池化。**

AlexNet 是什么？（第二层讲解C2）

我的理解是，图中只画出了一次卷积之后的输出结果。

卷积：两组输入均是27x27x48，各组分别使用128个5x5x48的卷积核进行卷积，padding=2，stride=1，根据公式：(input_size + 2 * padding - kernel_size) / stride + 1=(27+2*2-5)/1+1=27，得到每组输出是27x27x128。

ReLU：将卷积层输出的FeatureMap输入到ReLU函数中。

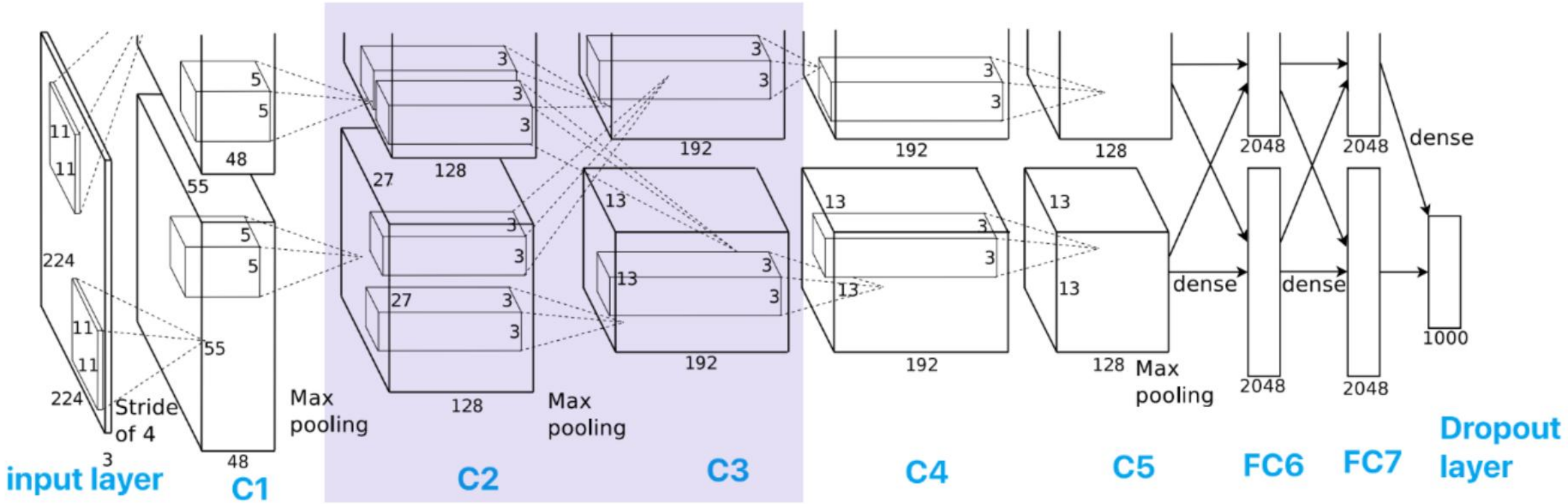
局部响应归一化：使用参数k=2，n=5，α=0.0001，β=0.75进行归一化。每组输出仍然是27x27x128。

池化：使用3x3，stride=2的池化单元进行最大池化操作（max pooling）。注意这里使用的是重叠池化，即stride小于池化单元的边长。根据公式：(27+2*0-3)/2+1=13，每组得到的输出为13x13x128。

它这边的池化也好理解，pool_size=3，stride=2，padding=0,输出的图像，高宽各减少一半，通道数不变。所以只画出了卷积的结果，也算情有可原。

AlexNet 是什么？（第三层讲解C3）

第三层 C3



该层的处理流程是： 卷积 → ReLU

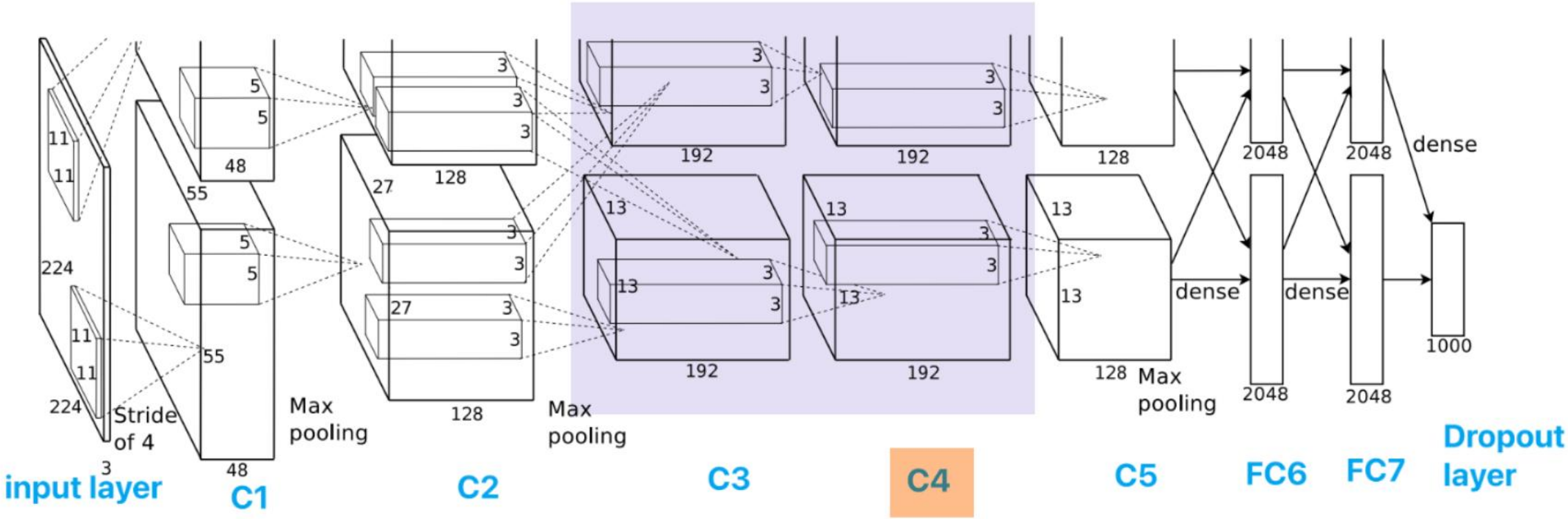
Title：AlexNet 是什么？（第三层讲解C3）

- 卷积
  - 输入是13x13x256
  - 使用384个3x3x256的卷积核进行卷积，padding=1，stride=1
  - 根据公式：(input_size + 2 * padding - kernel_size) / stride + 1=(13+2*1-3)/1+1=13，得到输出是13x13x384

Title：AlexNet 是什么？（第三层讲解C3）

- ReLU
  - 将卷积层输出的FeatureMap输入到ReLU函数中。将输出其分成两组，每组FeatureMap大小是13x13x192，分别位于单个GPU上。

Title：AlexNet 是什么？（第四层讲解C4）

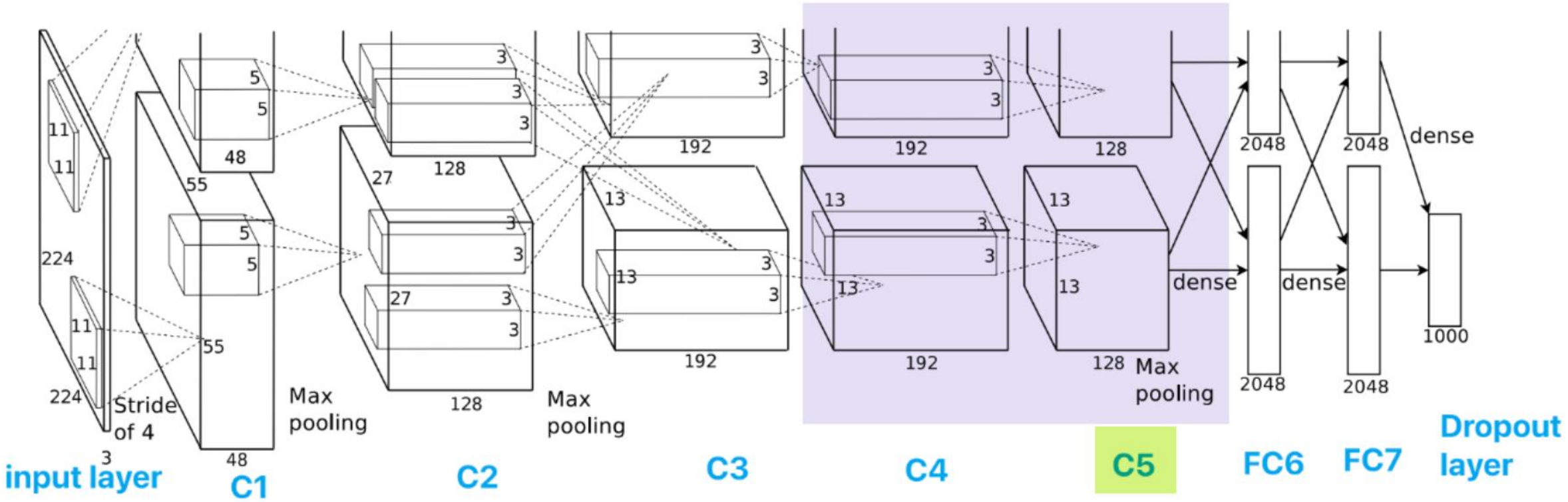


该层的处理流程是：卷积→ReLU

Title：AlexNet 是什么？（第四层讲解C4）

- 卷积
  - 两组输入均是13x13x192
  - 各组分别使用192个3x3x192的卷积核进行卷积，padding=1，stride=1
  - 根据公式：
  - (input_size + 2 * padding - kernel_size) / stride + 1=(13+2*1-3)/1+1=13
  - 得到每组FeatureMap输出是13x13x192。（分析的是一个组的情况）
- ReLU：将卷积层输出的FeatureMap输入到ReLU函数中。

Title：AlexNet 是什么？（第五层讲解C5）



该层的处理流程是：卷积 → ReLU → 池化

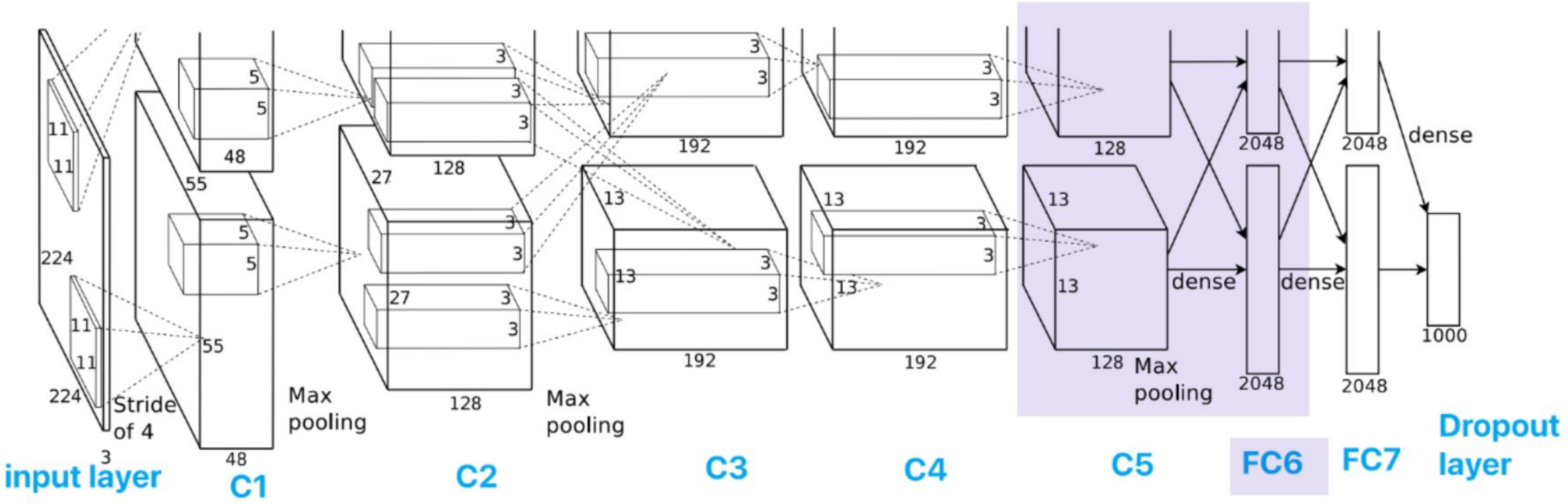
Title：AlexNet 是什么？（第五层讲解C5）

- 卷积：
  - 两组输入均是13x13x192
  - 各组分别使用128个3x3x192的卷积核进行卷积，padding=1，stride=1
  - 根据公式：(input_size + 2 * padding - kernel_size) / stride + 1=(13+2*1-3)/1+1=13，得到每组FeatureMap输出是13x13x128。（用单个组的说）
  - ReLU：将卷积层输出的FeatureMap输入到ReLU函数中。

Title：AlexNet 是什么？（第五层讲解C5）

- 池化：
  - 使用3x3，stride=2的池化单元进行最大池化操作（max pooling）
  - 注意这里使用的是重叠池化，即stride小于池化单元的边长。
  - 根据公式：(13+2*0-3)/2+1=6
  - 每组得到的输出为6x6x128
  - （这边，pool_size=3，stride=2 池化之后的窗口 减半）

Title：AlexNet 是什么？（第六层讲解C6）



该层的流程为：（卷积）全连接 → ReLU → Dropout （卷积）

Title：AlexNet 是什么？（第六层讲解C6）

- 全连接：
  - 输入为6×6×256（这里的输入是因为C5→FC6 两个GPU交互了 所以把第三维合并作为输入）
  - 使用4096个6×6×256的卷积核（注意这里的卷积核大小与输入图像完全相同 所以1个卷积核输出1个值）进行卷积

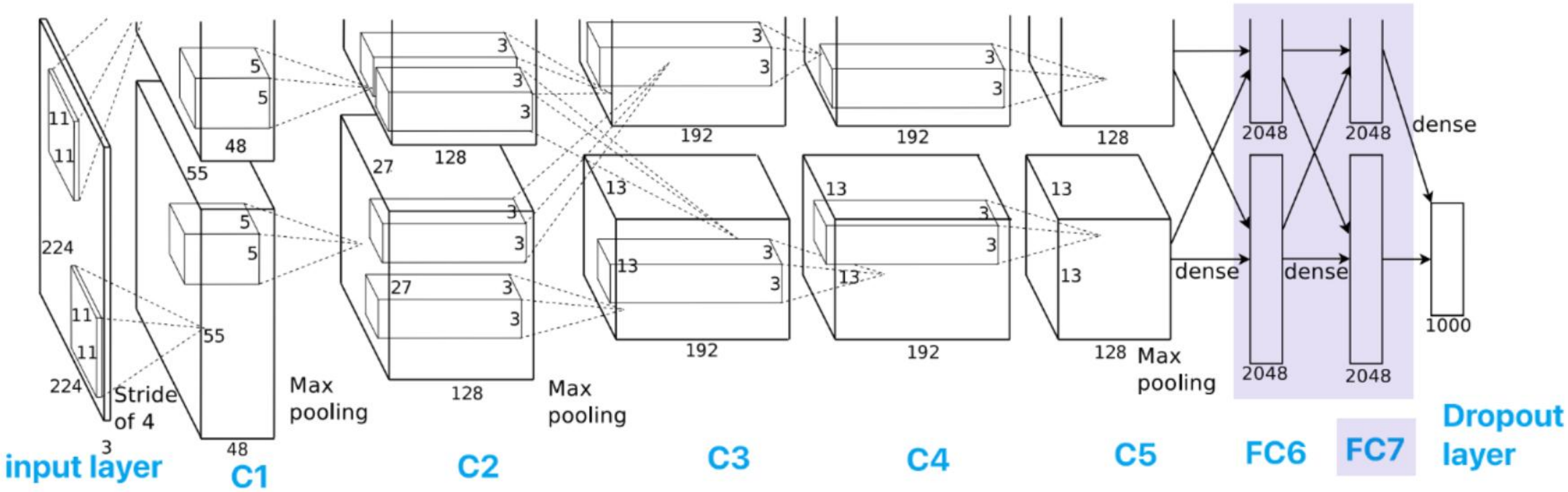
Title：AlexNet 是什么？（第六层讲解C6）

- 由于卷积核尺寸与输入的尺寸完全相同，即卷积核中的每个系数只与输入尺寸的一个像素值相乘一一对应
- 根据公式：(input_size + 2 * padding - kernel_size) / stride + 1=(6+2*0-6)/1+1=1，得到输出是1x1x4096。
- 即有4096个神经元，该层被称为全连接层。

Title：AlexNet 是什么？（第六层讲解C6）

- ReLU
  - 这4096个神经元的运算结果通过ReLU激活函数中。
- Dropout
  - 随机的断开全连接层某些神经元的连接，通过不激活某些神经元的方式防止过拟合。4096个神经元也被均分到两块GPU上进行运算。

Title：AlexNet 是什么？（第七层讲解C7）

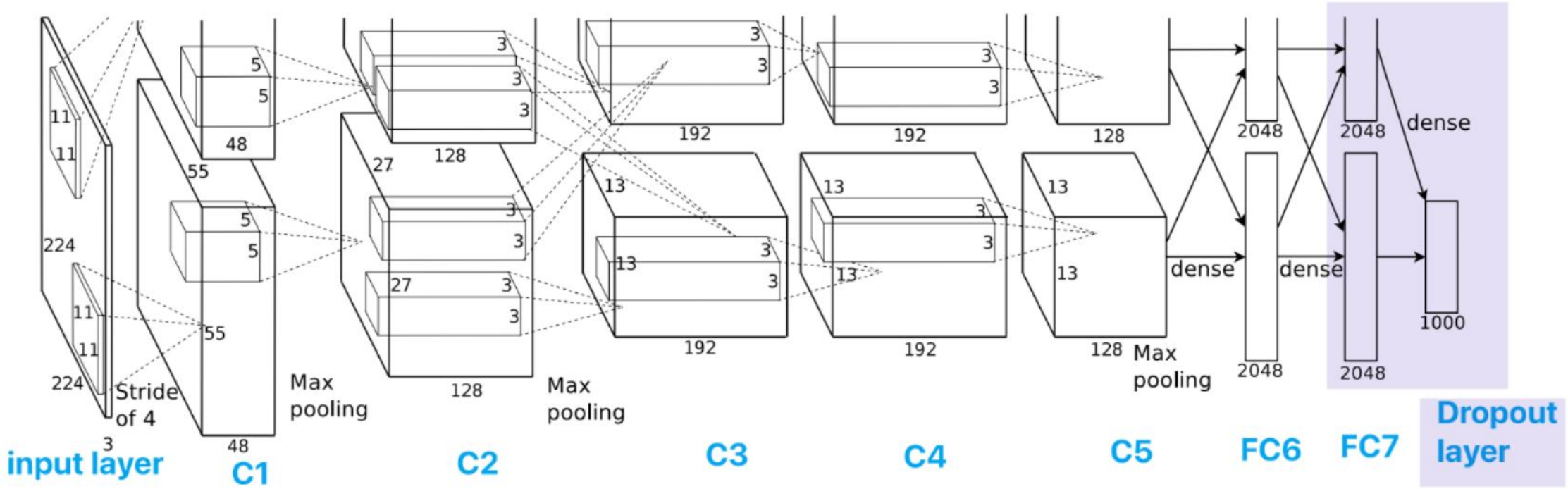


该层的流程为：（卷积）全连接 → ReLU → Dropout

Title：AlexNet 是什么？（第七层讲解C7）

- 全连接：
  - 输入为4096个神经元，输出也是4096个神经元（作者设定的）。
- ReLU：
  - 这4096个神经元的运算结果通过ReLU激活函数中。
- Dropout：
  - 随机的断开全连接层某些神经元的连接，通过不激活某些神经元的方式防止过拟合。
  - 4096个神经元也被均分到两块GPU上进行运算。

Title：AlexNet 是什么？（第八层讲解C8）



该层的流程为：（卷积）全连接 → Softmax

Title：AlexNet 是什么？（第八层讲解C8）

- 全连接
  - 输入为4096个神经元，输出是1000个神经元。这1000个神经元即对应1000个检测类别。
- Softmax
  - 这1000个神经元的运算结果通过Softmax函数中，输出1000个类别对应的预测概率值。

Title:

AlexNet神经元数量

Title：AlexNet神经元数量

层数	定义	数量
C1	C1层的FeatureMap的神经元个数	55x55x48x2=290400
C2	C2层的FeatureMap的神经元个数	27x27x128x2=186624
C3	C3层的FeatureMap的神经元个数	13x13x192x2=64896
C4	C4层的FeatureMap的神经元个数	13x13x192x2=64896
C5	C5层的FeatureMap的神经元个数	13x13x128x2=43264
FC6	FC6全连接层神经元个数	4096
FC7	FC7全连接层神经元个数	4096
Output layer	输出层神经元个数	1000

Title：

整个AlexNet网络包含的神经元个数为：

290400 + 186624 + 64896 + 64896 + 43264 + 4096 + 4096 + 1000 = 659272

大约65万个神经元。

Title:

AlexNet参数数量

Title：AlexNet参数数量

层数	定义	数量
C1	卷积核11x11x3，96个卷积核，偏置参数	(11x11x3+1)x96=34944
C2	卷积核5x5x48，128个卷积核，2组，偏置参数	(5x5x48+1)x128x2=307456
C3	卷积核3x3x256，384个卷积核，偏置参数	(3x3x256+1)x384=885120
C4	卷积核3x3x192，192个卷积核，2组，偏置参数	(3x3x192+1)x192x2=663936
C5	卷积核3x3x192，128个卷积核，2组，偏置参数	(3x3x192+1)x128x2=442624
FC6	卷积核6x6x256，4096个神经元，偏置参数	(6x6x256+1)x4096=37752832
FC7	全连接层，4096个神经元，偏置参数	(4096+1)x4096=16781312
Output layer	全连接层，1000个神经元	1000x4096=4096000

Title：AlexNet参数数量

整个AlexNet网络包含的参数数量为：

34944 + 307456 + 885120 + 663936 + 442624 + 37752832 + 16781312 + 4096000 = 60964224

大约6千万个参数。

Title:

AlexNet的创新点

Title： AlexNet的创新点

- 激活函数ReLU
- 局部响应归一化
- Dropout
- 重叠池化
- 双GPU
- 端到端的训练

Title： AlexNet的创新点

- 激活函数ReLU
- 局部响应归一化LRN
  - LRN 有助于增强特征图中的对比度
  - 增强激活值较大的特征，抑制激活值较小的特征
  - 有助于模型更好地学习到数据中的关键特征
  - LRN 在一些现代的深度学习架构中已经不太常用，取而代之的是 Batch Normalization 等更有效的正则化和归一化技术

Title： AlexNet的创新点

- Dropout
  - 用于减少神经网络的过拟合
  - 使用Dropout时，在训练过程中随机将一部分神经元的输出置为0
  - 从而减少神经元之间的依赖关系，提高模型的泛化能力
  - AlexNet中很大的创新，现在很多的神经网络依然使用

Title： AlexNet的创新点

- 重叠池化
  - 指：池化层的池化窗口大小小于步幅大小，导致池化窗口之间有重叠部分的情况
  - **更好的特征提取：**重叠池化可以减少信息丢失，有助于更好地保留特征。
  - **提高空间信息的保留：**重叠池化可以保留更多的空间信息。

Title： AlexNet的创新点

- 双GPU
  - 硬件资源有限
  - AlexNet结构复杂，参数量庞大，难以在单个GPU上训练
  - 因此AlexNet采用两个GTX 580 GPU并行训练
  - 也就是把原先的卷积层平分成两部分FeatureMap分别在两块GPU上进行训练
  - 例如：卷积层55×55×96分成两个FeatureMap55×55×48

Title： AlexNet的创新点

- 端到端的训练
  - 其中输入数据从开始到结束直接传递给模型
  - 模型负责处理所有的预处理、特征提取和最终的预测
  - 而无需人为介入或手动进行特征工程

Title：

VGG

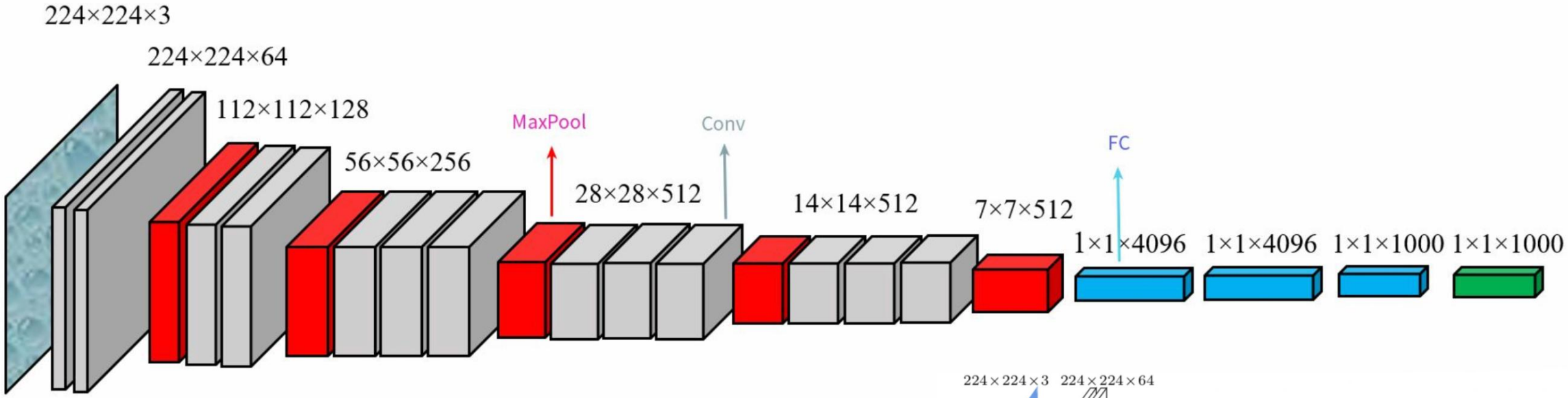
牛津大学的视觉几何组提出

2015年的ICLR大会上发表

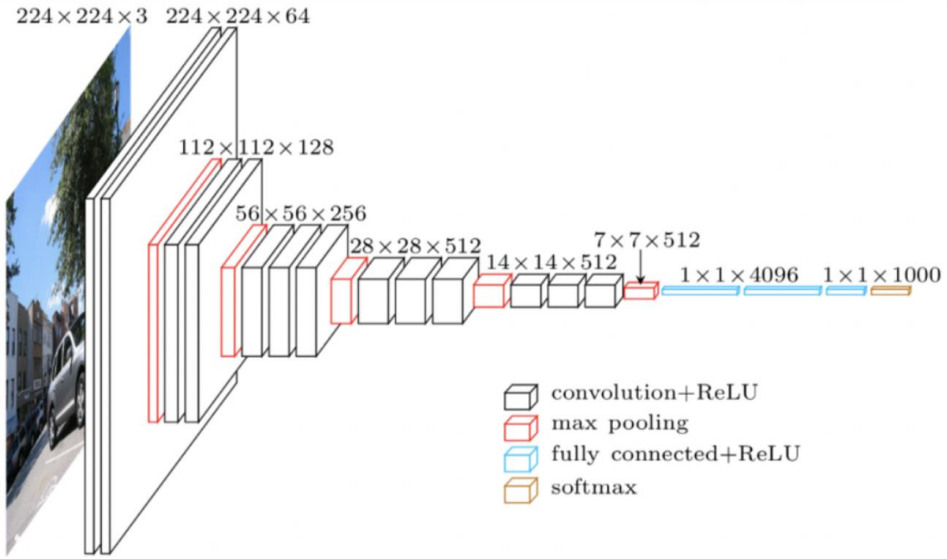
在ImageNet 2014 年图像分类竞赛的第二名VGG

VGG 全称Visual Geometry Group

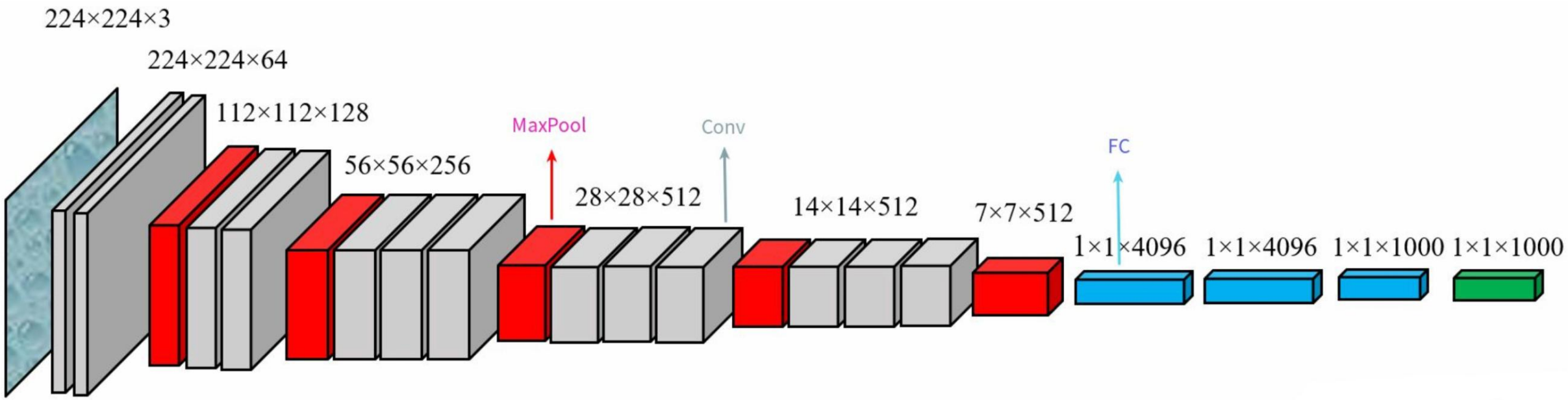
Title：VGG是什么？（网络架构图）



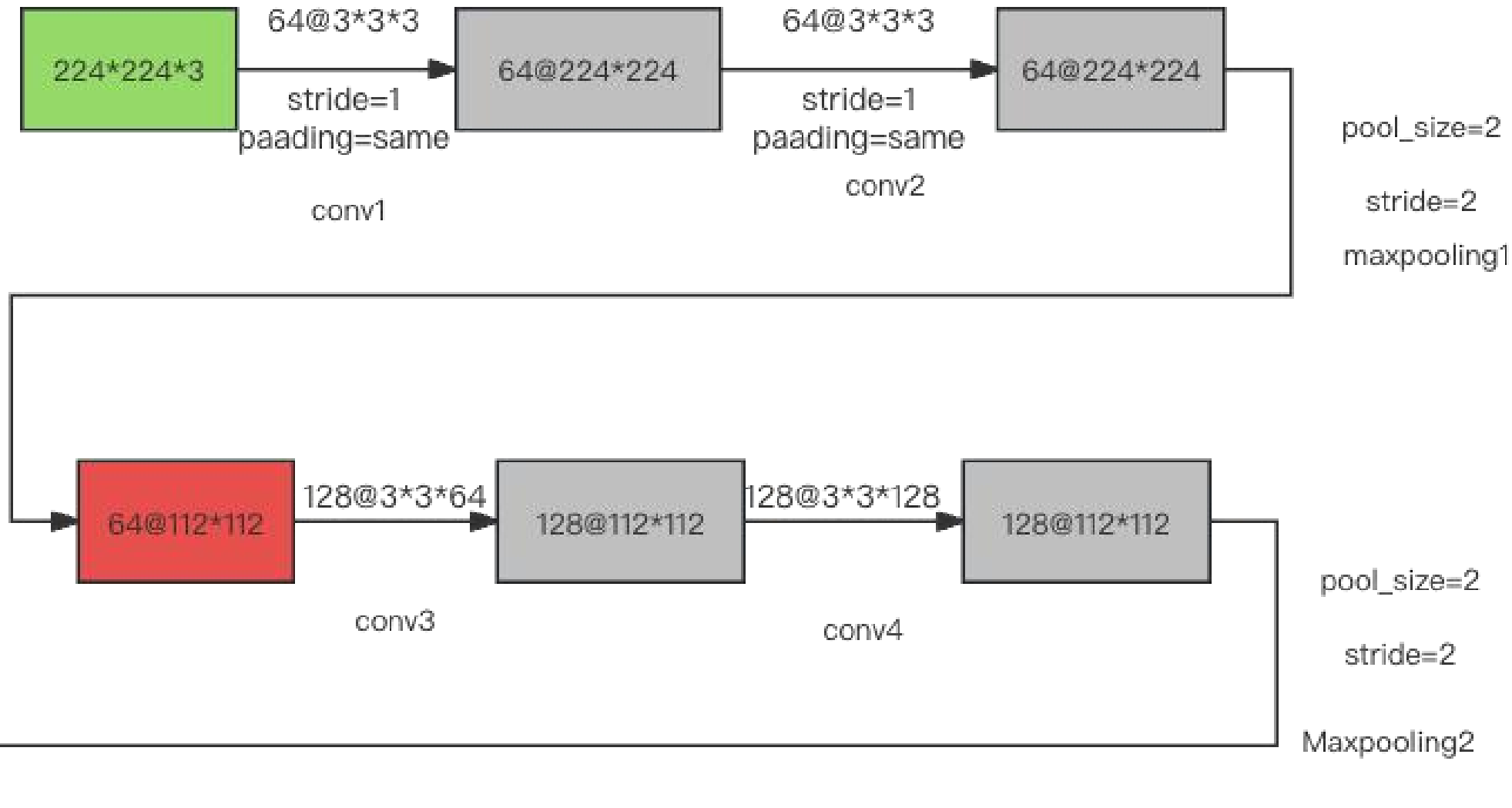
- 每个块的高度和宽度减半
- 最终的高度和宽度都为7
- 最后展平表示，送入全连接层

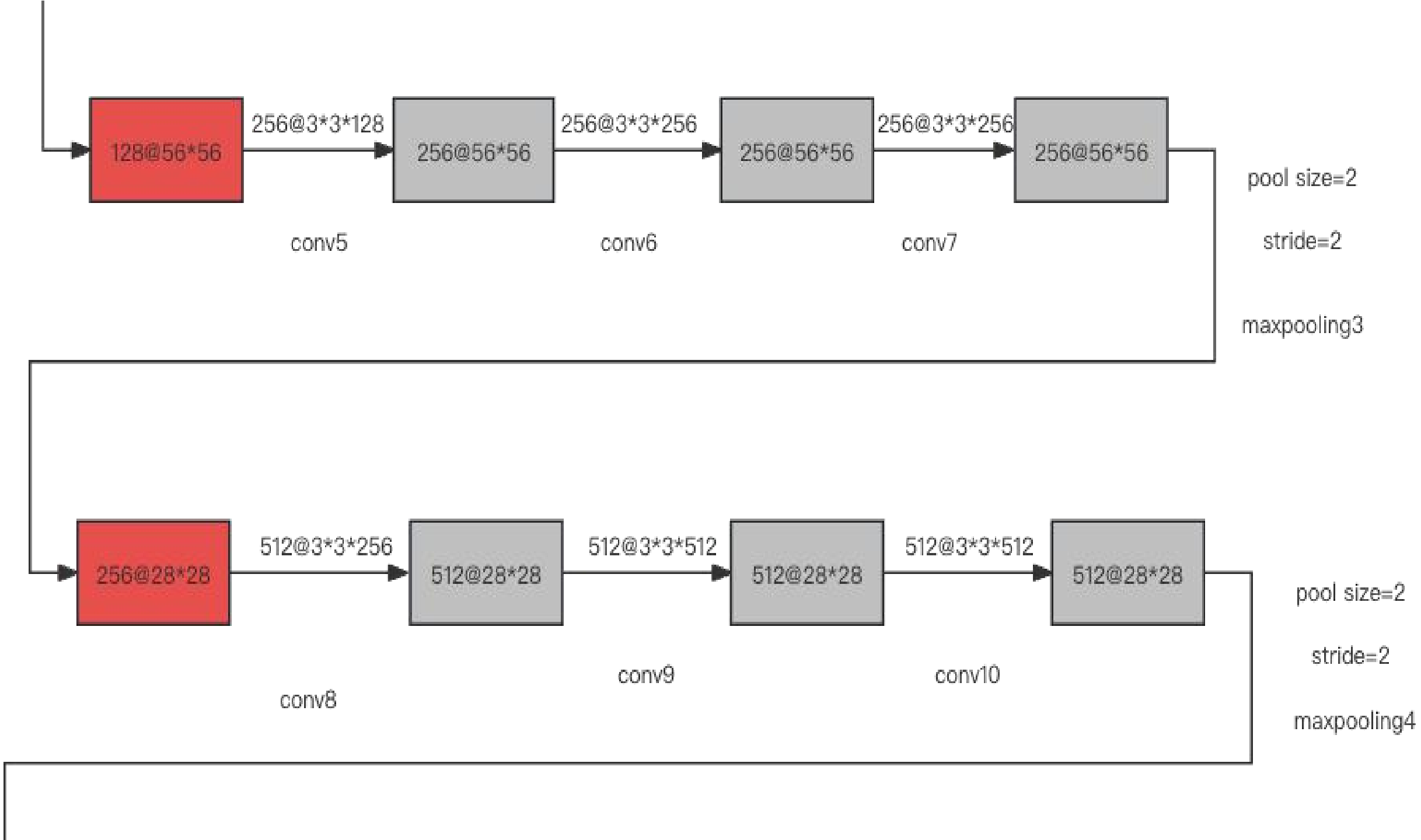


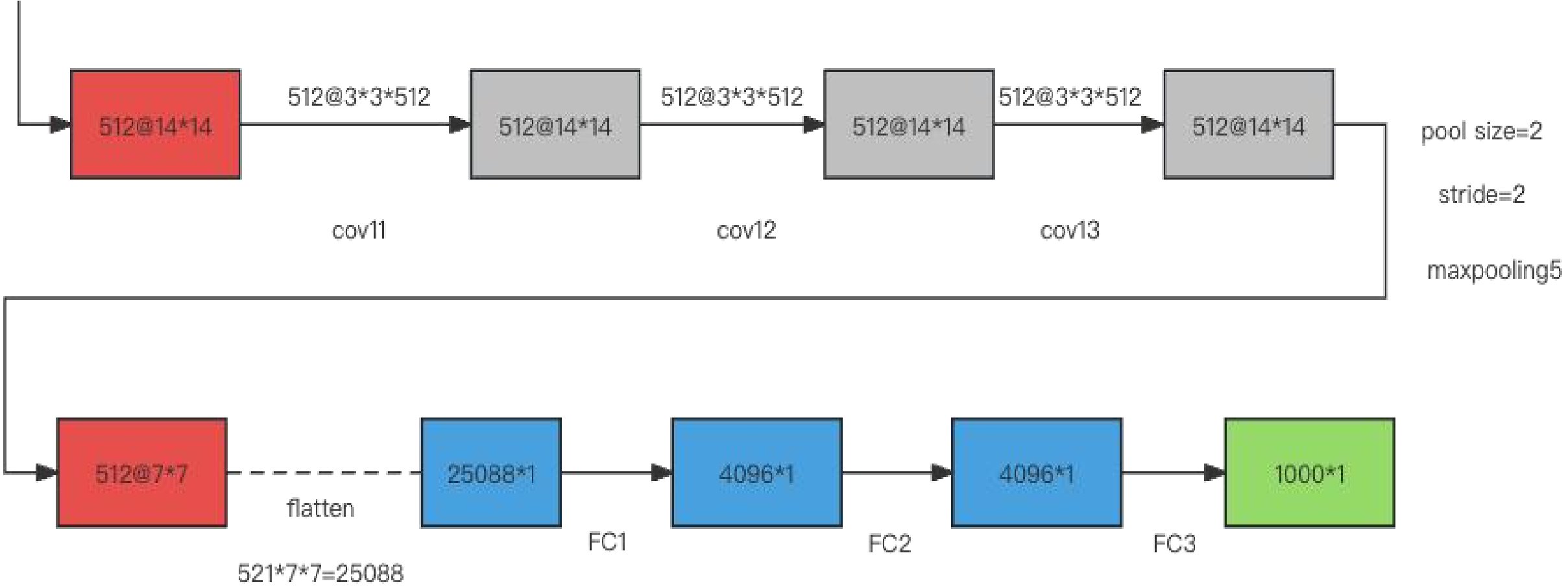
Title：VGG是什么？（网络架构图）



Title：VGG是什么？（网络架构图）




128@56*56
256@3*3*128
256@56*56
256@3*3*256
256@56*56
256@3*3*256
256@56*56
conv5
conv6
conv7
pool size=2
stride=2
maxpooling3
256@28*28
512@3*3*256
512@28*28
512@3*3*512
512@28*28
512@3*3*512
512@28*28
conv8
conv9
conv10
pool size=2
stride=2
maxpooling4


512@14*14
512@3*3*512
512@14*14
512@3*3*512
512@14*14
512@3*3*512
512@14*14
pool size=2
stride=2
maxpooling5
cov11
cov12
cov13
512@7*7
flatten
25088*1
4096*1
4096*1
1000*1
FC1
FC2
FC3
521*7*7=25088

Title：VGG是什么？（网络架构图）

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 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	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 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Title：VGG是什么？（网络架构图）

第1层输入层：输入224×224×3的图像

- 第2层vgg block层：
  - 输入为224×224x3
  - 经过64个kernel size 为3×3x3的filter,stride = 1， padding=same卷积后
  - 得到shape 为224x224×64的block层（指由conv构成的vgg-block）

Title：VGG是什么？（网络架构图）

- 第3层Max-pooling层：
  - 输入为224×224×64，
  - 经过pool size=2,stride=2的减半池化后
  - 得到尺寸为112x112x64的池化层

Title：VGG是什么？（网络架构图）

- 第4层vgg block层：
  - 输入尺寸为112x112×64
  - 经128个3×3×64的filter卷积
  - 得到112x112x128的block层。

Title：VGG是什么？（网络架构图）

- 第5层Max-pooling/层：
  - 输入为112×112×128
  - 经pool size=2,stride=2减半池化后
  - 得到尺寸为56×56x128的池化层。
- 第6层vgg block层：
  - 输入尺寸为56x56x128
  - 经256个3x3x128的filter卷积
  - 得到56×56x256的block层

Title：VGG是什么？（网络架构图）

- 第7层Max-pooling层：
  - 输入为56×56×256
  - 经pool size=2,stride =2减半池化后
  - 得到尺寸为28x28x256的池化层
- 第8层vgg block层：
  - 输入尺寸为28x28×256
  - 经512个3×3x256的filter卷积
  - 得到28×28×512的block层

Title：VGG是什么？（网络架构图）

- 第9层Max-pooling层：
  - 输入为28×28x512
  - 经pool size =2,stride =2减半池化后
  - 得到尺寸为14×14×512的池化层
- 第10层vgg block层：
  - 输入尺寸为14×14×512
  - 经512个3x3x512的filter卷积
  - 得到14×14×512的block层

Title：VGG是什么？（网络架构图）

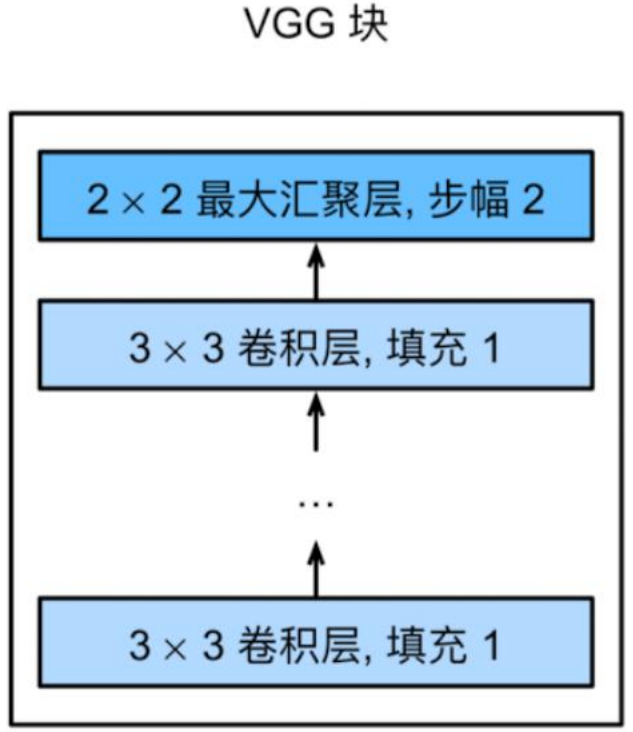
- 第11层Max-pooling层：
  - 输入为14×14×512
  - 经pool size =2,stride =2减半池化后
  - 得到尺寸为7x7x512的池化层：
    - 该层后面还隐藏了flatten操作
    - 通过展平得到7x7×512=25088个参数后
    - 与之后的全连接层相连

Title：VGG是什么？（网络架构图）

- 第12~14层Dense层：
  - 第12~14层神经元个数分别为4096，4096，1000
  - 其中前两层在使用relu后还使用了Dropout对神经元随机失活
  - 最后一层全连接层用softmax输出1000个分类

Title：VGG的特点：提出VGG块的思想

- 经过每一个VGG块的处理，尺寸减半、层数翻倍
- 使用kernel_size=3，padding=1 (p_h = p_w = 2)（保持高度和宽度）的**卷积层**
- 使用pooling_size = 3，padding = 1的最大汇聚层(池化层把上一层的卷积层特征图尺寸减半)
- 使用较小的kernel size



Title：VGG的思想

- 在VGG论文中，Simonyan和Ziserman尝试了各种架构。特别是他们发现深层且窄的卷积（即3 × 3）比较浅层且宽的卷积更有效。
  - 堆叠两个3×3的卷积来替代一个5×5的卷积，它们的感受野相同
  - 堆叠三个3×3的卷积来替代一个7×7的卷积，它们的感受野相同

Title：

NIN

2014年新加坡国立大学(颜水成)

2014年ICLR的一篇paper

1×1的卷积

Global Average Pooling

Title：intro

- LeNet→AlexNet→VGG→NiN→GoogLeNet→ResNet
- LeNet→AlexNet→VGG
  - 卷积层模块充分抽取空间特征
  - 全连接层输出分类结果
  - AlexNet & VGG 改进在于把两个模块加宽 、加深（加宽指增加通道数，加深指网络层数不断增加
- NiN：串联（多个卷积层和”全连接层“构成的小网络）来构建一个深层网络

Title：

NIN是什么？

Title：NIN网络结构

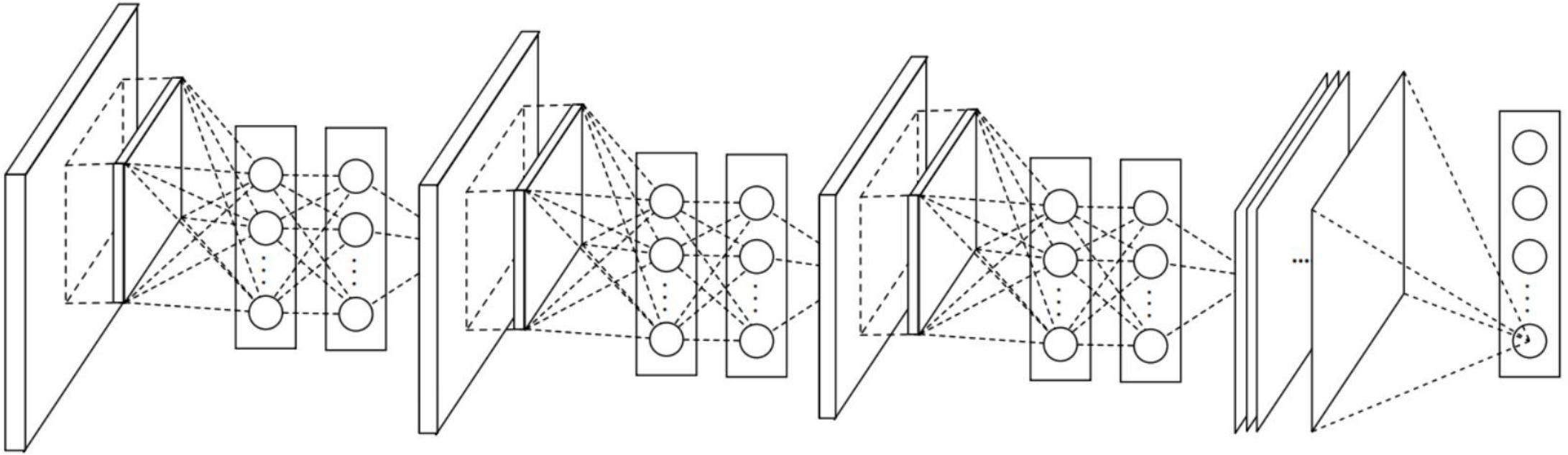


Figure 2: The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer.

Title：NIN网络结构

上图的网络架构：

- NiN = 3 × mplconv层 + 1 × GAP全局池化层
  - 1个mplconv层 = 1个微型神经网络 ∴ NiN（网络中的网络）
  - 1个mplconv层 内部由多层感知机实现 = （1个conv + 2个fc层）
  - mpl感知机的层数是可以调整的
  - mlpconv代替了传统的卷积层
  - GAP代替了传统CNN模型中末尾的全连接层

Title：

NIN块

Title：NIN块

- NiN 块是 NiN 中的基础块
  - NIN块=1个卷积层+2个1×1的卷积层
    - 串联而成
    - 每一次卷积之后都会进行非线性激活
  - 第一个卷积层的超参数可以自行设置
  - 第二个和第三个卷积层的超参数是固定的

Title：NIN块

- 卷积窗口（有AlexNet的影子）
  - 卷积窗口形状为11×11、5×5、3×3的卷积层、输出通道数与AlexNet一致
  - 每个NIN块后接一个stride=2 pool_size=3×3的最大池化层
- 去掉了AlexNet最后的3个全连接层
  - 使用了输出通道数=标签类别数的NIN块
  - GAP对每个通道中的元素求平均并直接用于分类 （GAP 全局平均池化 Global Average Pooling）

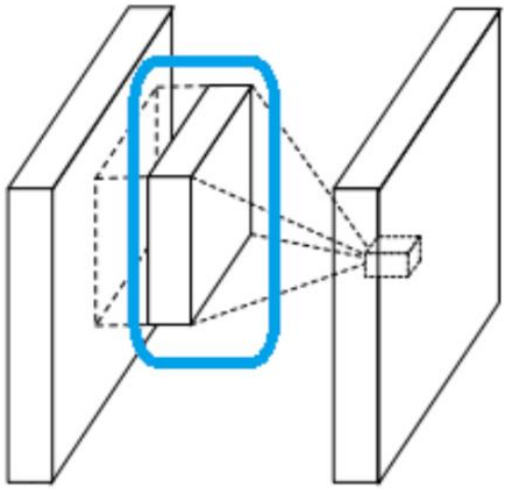
Title：

NIN的特点

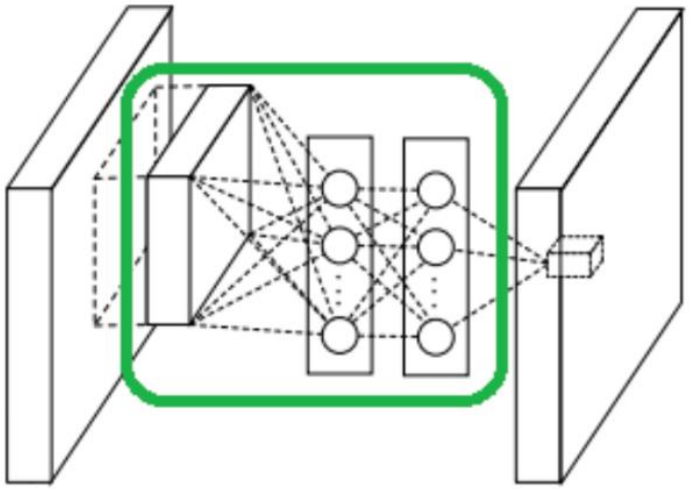
Title：NIN的特点

改进1：

先前CNN中 简单的 线性卷积（篮框部分）被替换为了多层感知机MLP（多层全连接和非线性函数的组合）（绿框部分）



(a)Linear convolution layer



(b)Mlpconv layer

Title：mlpconv的特点

- NiN使用由一个卷积层和多个1 × 1卷积层组成的块。该块可以在卷积神经网络中使用，以允许更多的每像素非线性
- 提供了网络层间映射的一种新可能
- 增加了网络卷积层的非线性能力

Title：全局池化层 global average pooling(:

改进2：

- 先前CNN中 FC 被替换层了 GAP
  - 假设分类任务共有C个类别
  - 先前CNN中最后一层为特征图层数（共计N）的全连接层，要映射到C个类别上
  - 改成全局池化层，最后一层特征图层数（共计C）的全局池化层，恰好对应分类任务的C个类别

Title：GAP的优点

GAP优点

- 传统CNN结构，卷积以后全连接层，经过softmax输出分类，最后的全连接层有过拟合的风险
- GAP
  - 最后的特征图层数 = 输出类别数
  - 没有全连接层，需要学习的参数大大减少，避免了FC层过拟合的发生

Title： NIN

Summary

Title：关于NIN两个比较重要的点

- mlpconv = MLP（1×1的conv+ 2×Conv） 增加非线性变换，更好的学习局部特征
- GAP 全局平均池化 防止过拟合

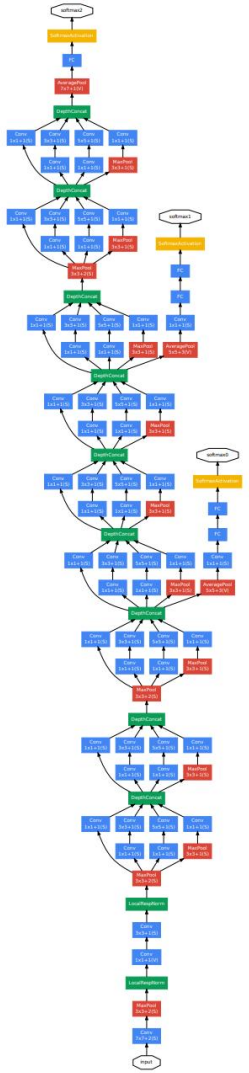
Title

GoogLeNet

Title：什么是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								

Table 1: GoogLeNet incarnation of the Inception architecture

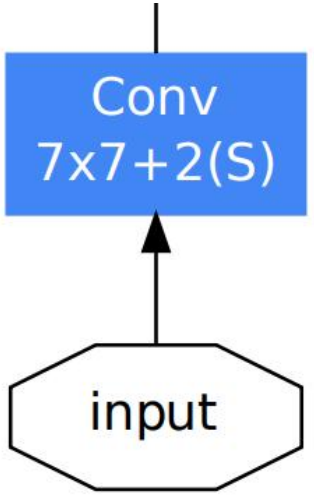


Title：GoogLeNet网络架构

原文地址链接：https://arxiv.org/pdf/1409.4842v1

- 0、输入

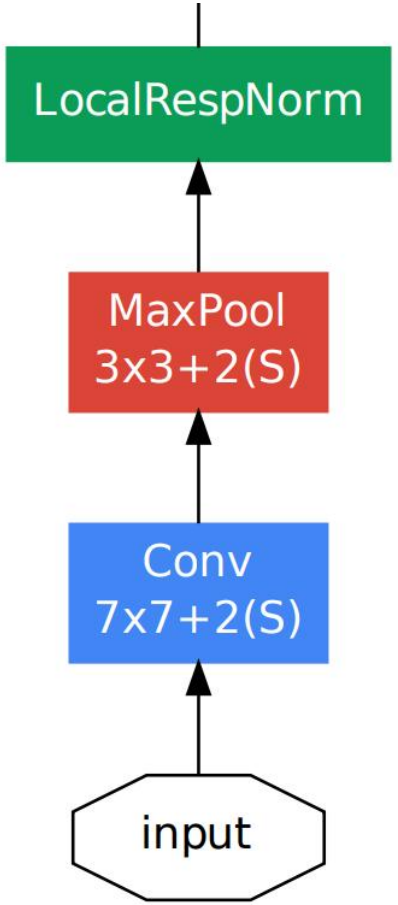
  原始输入图像为224x224x3，且都进行了零均值化的预处理操作（图像每个像素减去均值）。



Title：GoogLeNet网络架构

- 1、第一层（卷积层）
  - 使用7x7的卷积核（滑动步长2，padding为3），64通道，**输出**为112x112x64，卷积后进行ReLU操作
  - 经过3x3的max pooling（步长为2），输出为((112 - 3+1)/2)+1=56，即56x56x64，再进行ReLU操作

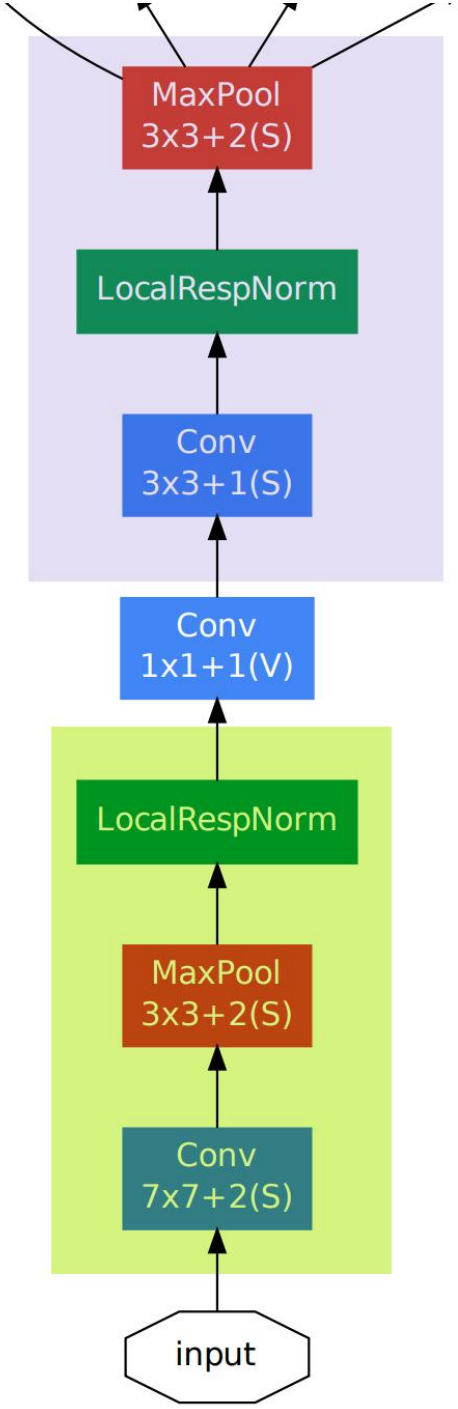
需要训练的参数量：64*7*7*3



Title：GoogLeNet网络架构

- 2、第二层（卷积层）
  - 输入：56*56 *64
  - 操作：64@kernel size=1*1 *64
  - 输出：64@56*56

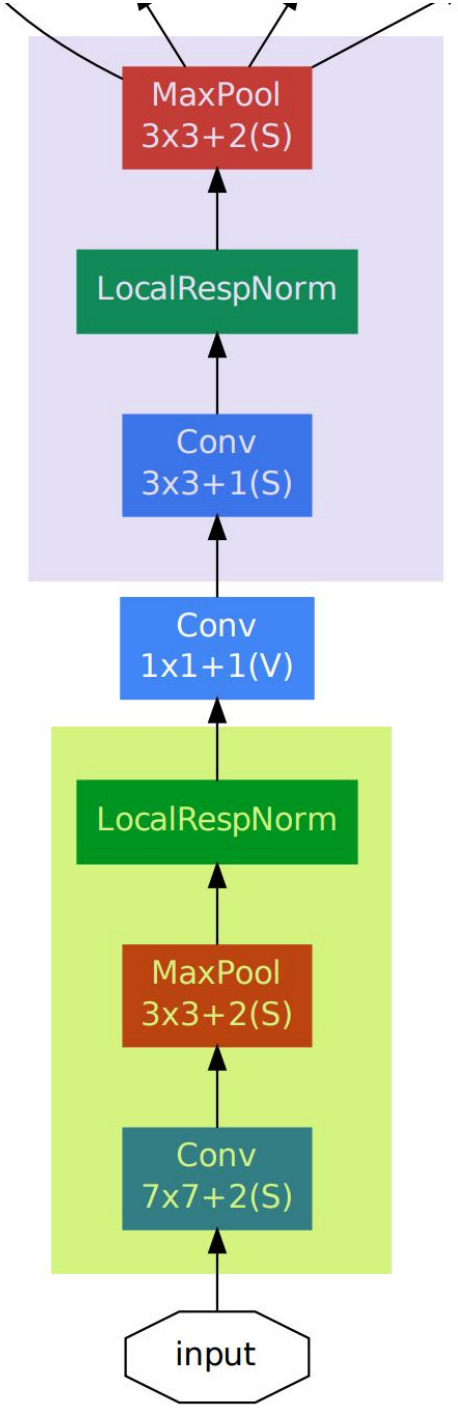
需要训练的参数量=1*1*64*64=4096



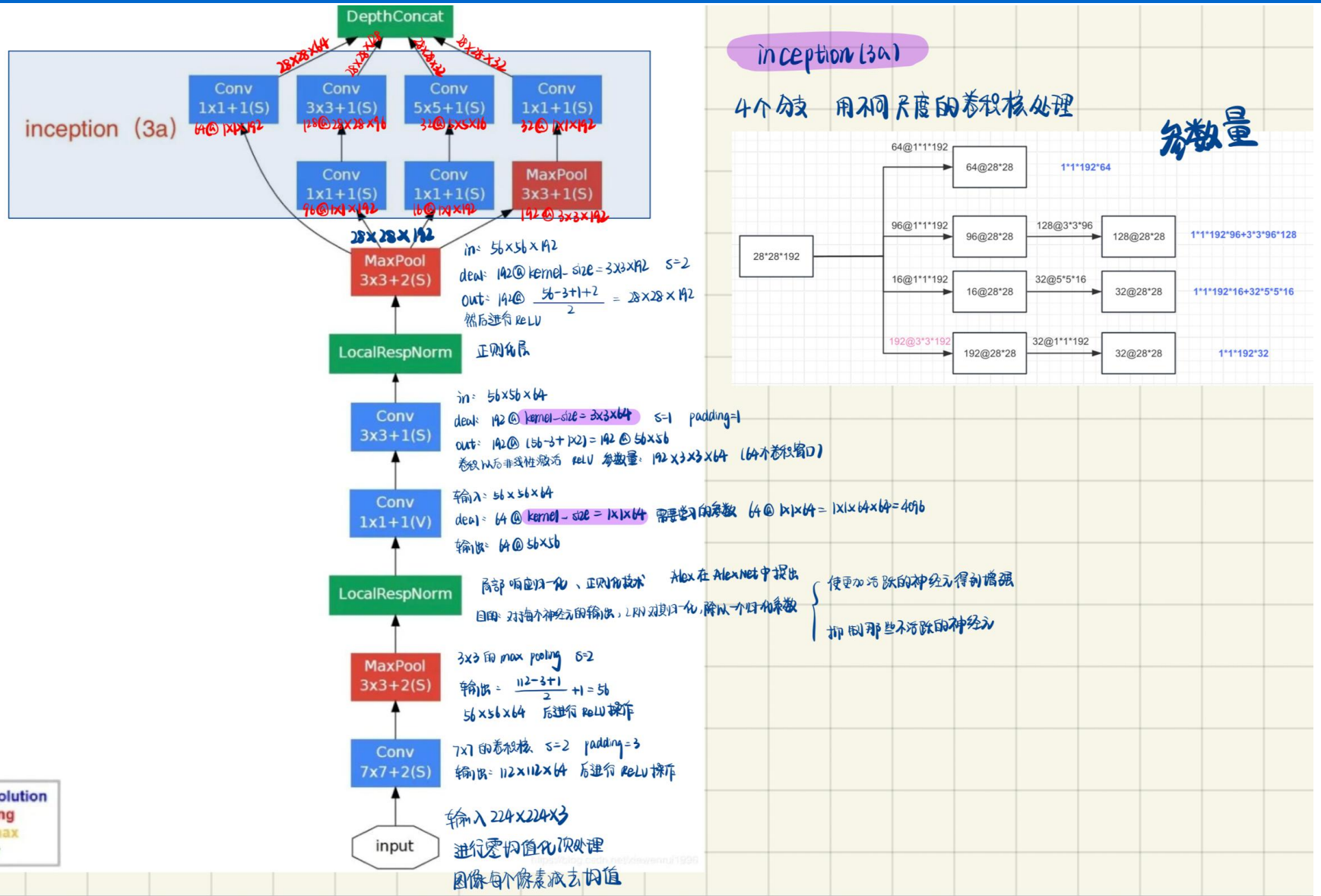
Title：GoogLeNet网络架构

- 3、第三层（卷积层）
  - 使用3x3的卷积核（滑动步长为1，padding为1），192通道，输出为56x56x192，卷积后进行ReLU操作
  - 经过3x3的max pooling（步长为2），输出为((56 - 3+1)/2)+1=28，即28x28x192，再进行ReLU操作

需要训练的参数量=3*3*64*192=110592



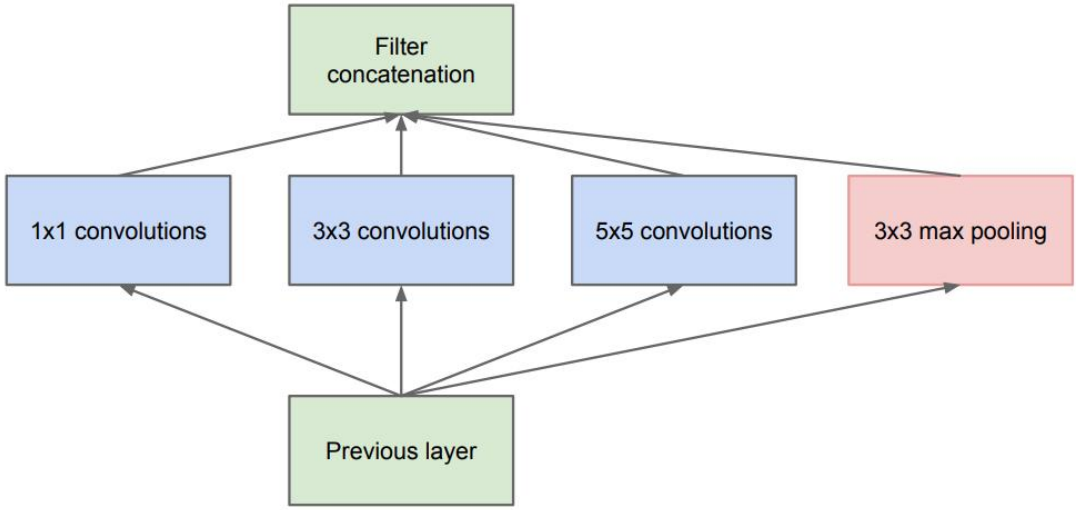
Title：GoogLeNet网络架构



Inception块

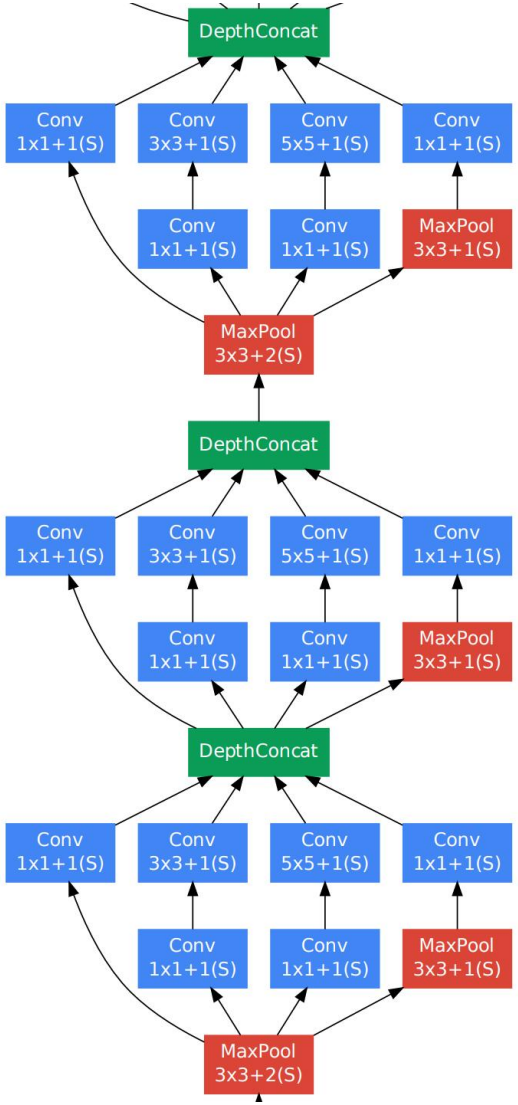
Title：Inception块

- 思想：利用不同大小的卷积核实现不同尺度的感知，最后进行融合，可以得到图像更好的表征；
- 基本组成结构：1*1卷积，3*3卷积，5*5卷积，3*3最大池化，最后对四个成分运算结果进行通道上组合。



(a) Inception module, naïve version

Title：GoogLeNet网络架构



Inception（4a）

Inception（3b）

Inception （3a）

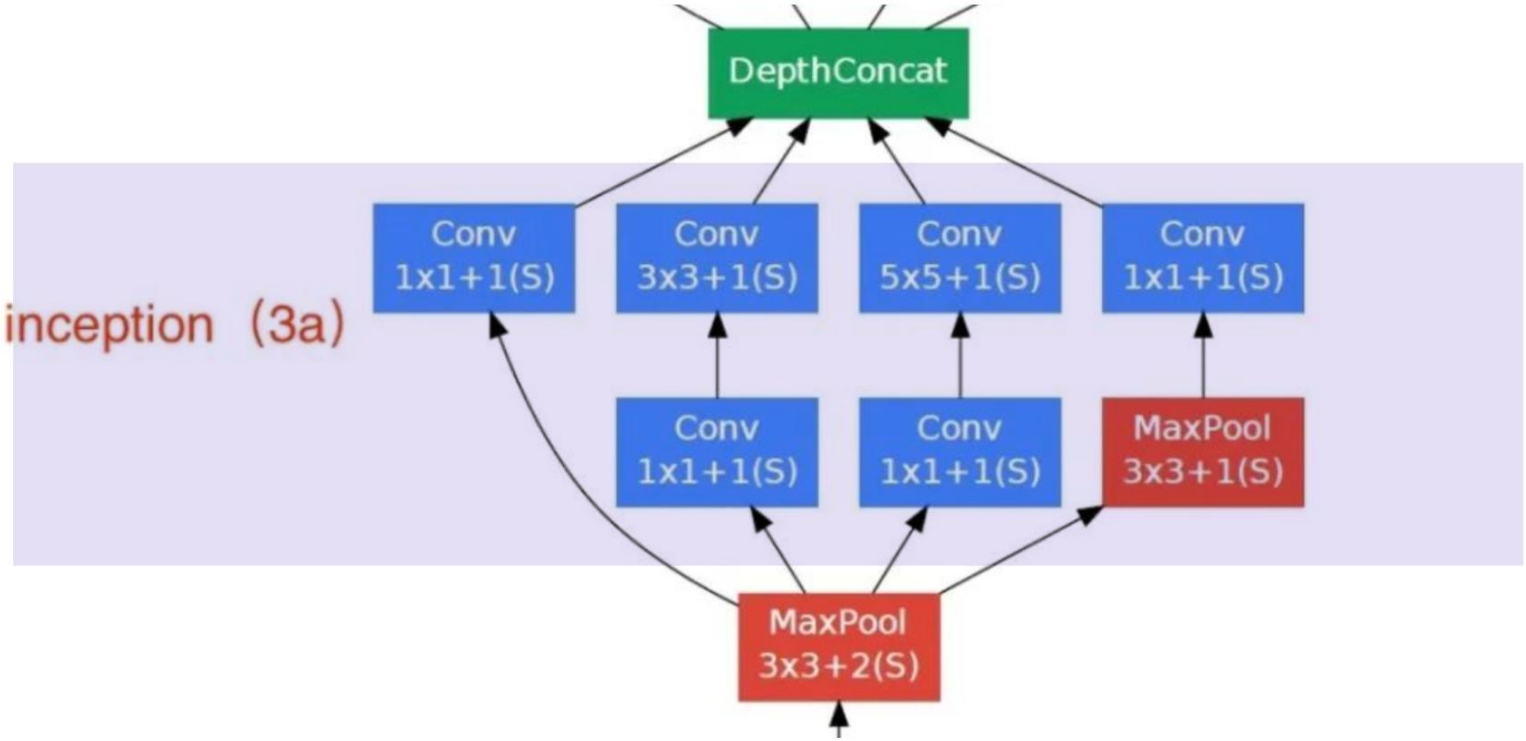
- Inception 3a层，分为四个分支，采用不同尺度；
- Inception 3b层，分为四个分支，采用不同尺度；
- (Inception 4a、4b、4c、4d、4e)与Inception3a，3b类似

Inception(3a)

Inception(3a)长什么样?

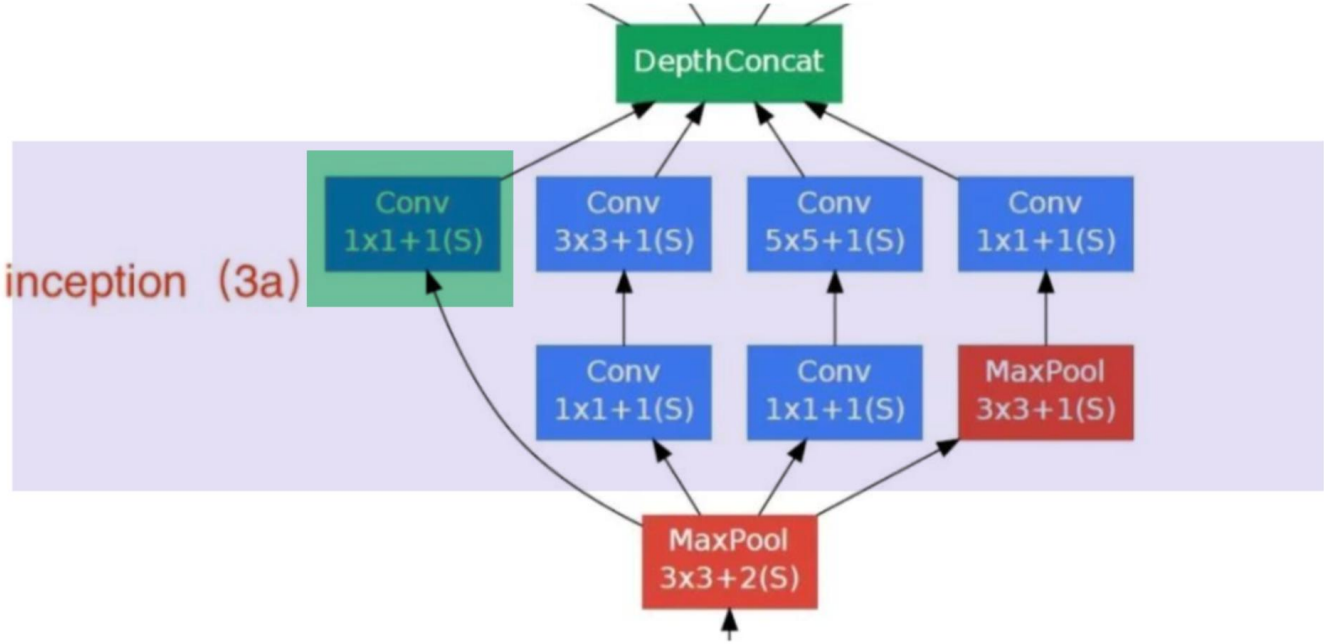
Title： Inception(3a)

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
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



图的方式理解

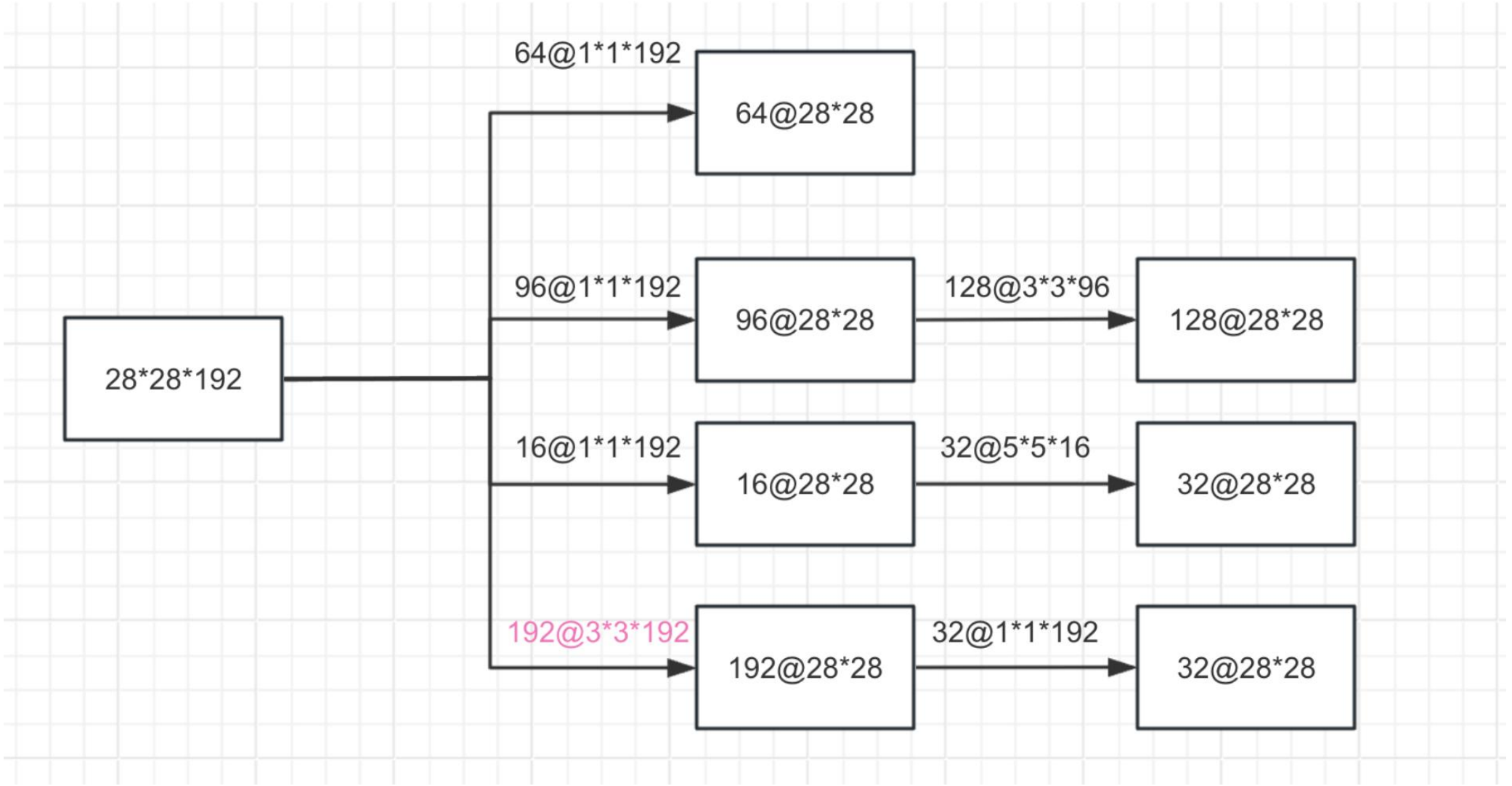
Title：Inception(3a)



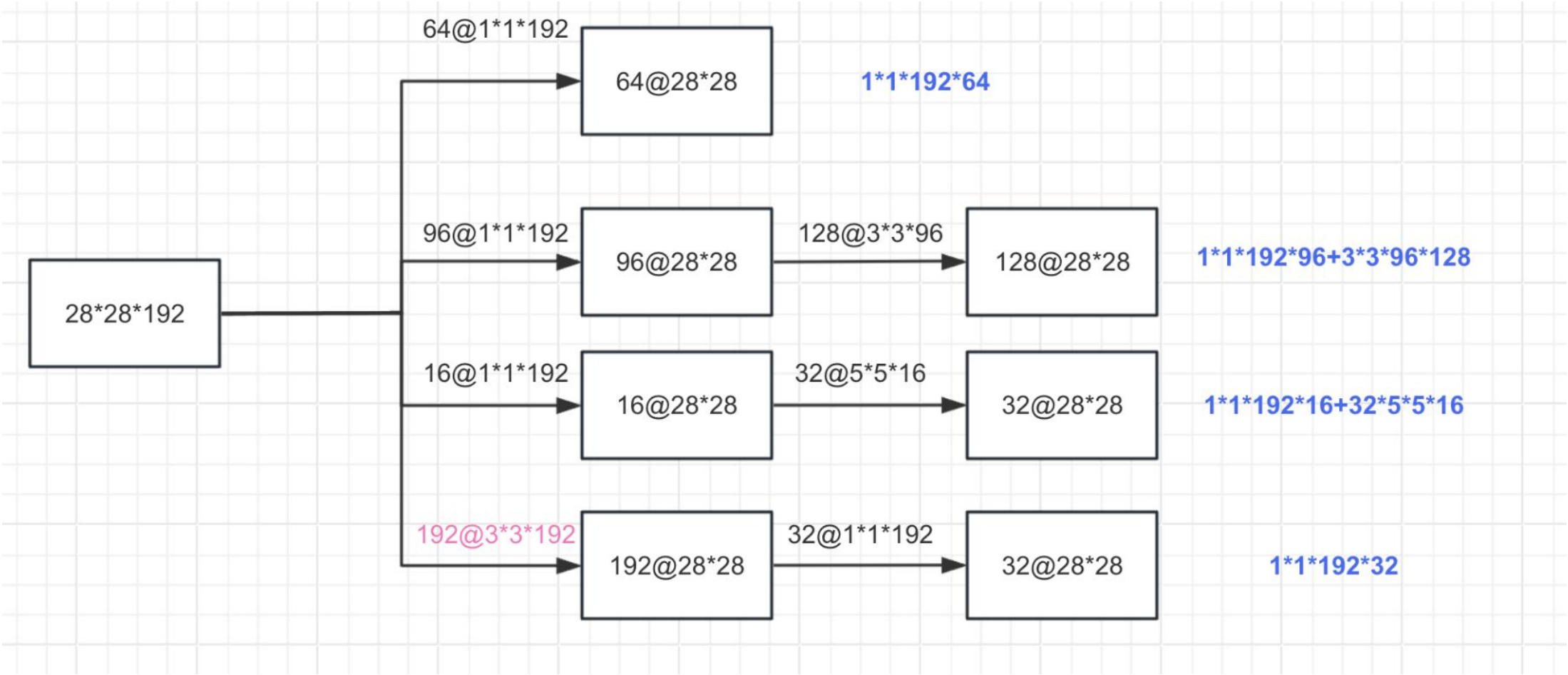
输入：192@28*28 → 28*28*192

(从左往右数操作)(省略所有卷积操作后的ReLu操作)

Title： Inception(3a)



Title： Inception(3a)



Title：Inception(3a)

参数量统计：

- 1*1*192*64=12288
- 1*1*192*96+3*3*96*128=129024
- 1*1*192*16+32*5*5*16=15872
- 1*1*192*32=6144
  - 12288+129024+15872+6144=163328
  - 163328/1024=159.5K

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
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

文字描述理解

Title：Inception(3a)

（1）使用64个1x1的卷积核

- 运算后特征图输出为28x28x64
- 然后ReLU操作
- 参数量1*1*192*64=12288

（2）96个1x1的卷积核（3x3卷积核之前的降维）

- 运算后特征图输出为28x28x96
- 进行ReLU计算
- 再进行128个3x3的卷积，输出28x28x128。
- 参数量1*1*192*96+3*3*96*128=129024

Title：Inception(3a)

（3）16个1x1的卷积核（5x5卷积核之前的降维）

- 将特征图变成28x28x16
- 进行ReLU计算
- 再进行32个5x5的卷积
- 输出28x28x32
- 参数量1*1*192*16+5*5*16*32=15872

Title：Inception(3a)

（4）pool层，使用3x3的核，输出28x28x192

- 然后进行32个1x1的卷积，输出28x28x32。
- 总参数量1*1*192*32=6144

Title：Inception(3a)

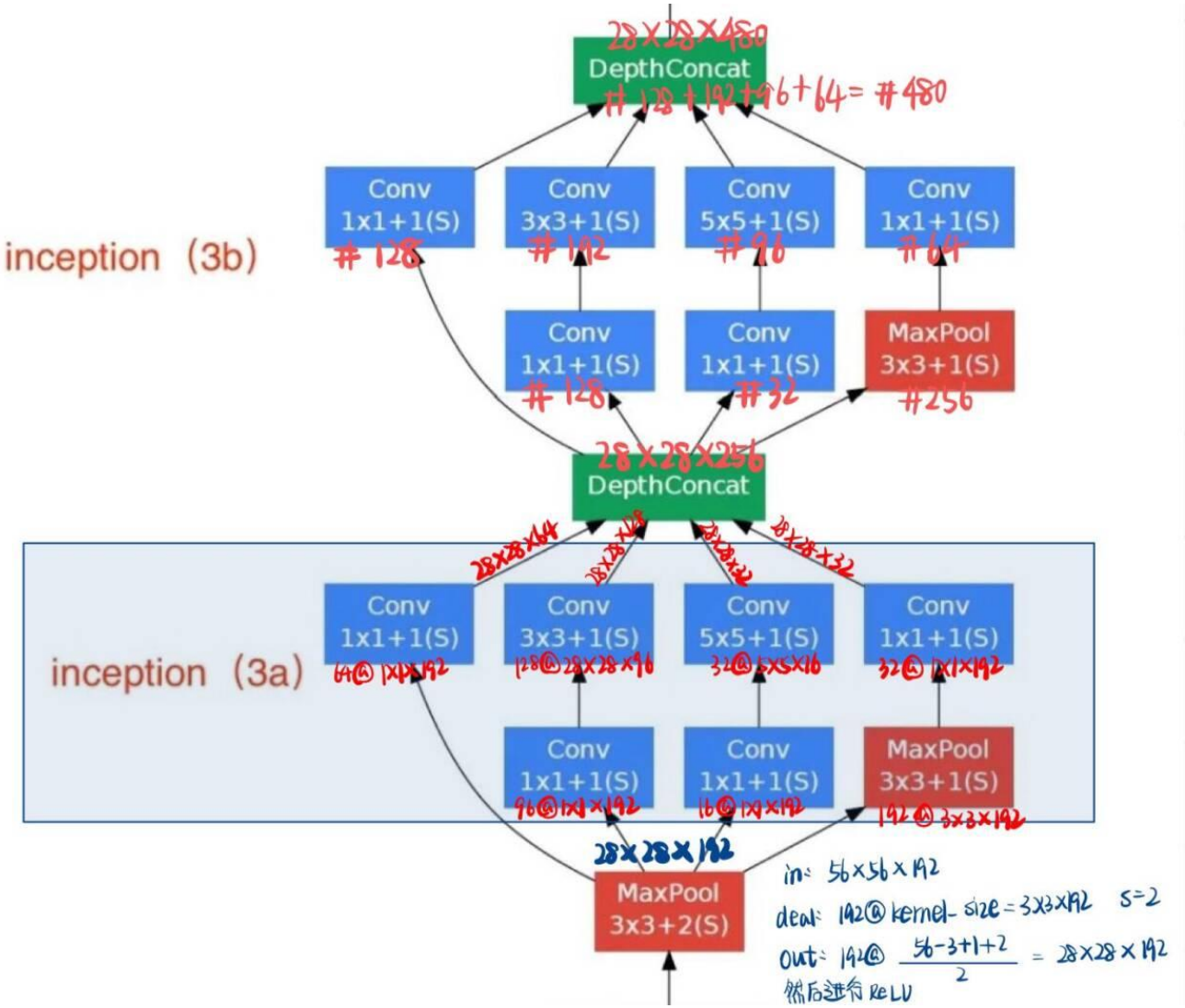
（5）将四个结果进行连接，

- 对这四部分输出结果的第三维并联
- 即64+128+32+32=256，最终输出28x28x256
- 总的参数量是12288+129024+15872+6144=163328，即163328/1024=159.5K，约等于159K。

Inception(3b)

Inception(3b)长什么样?

Title： Inception(3b)



Title：Inception(3b)

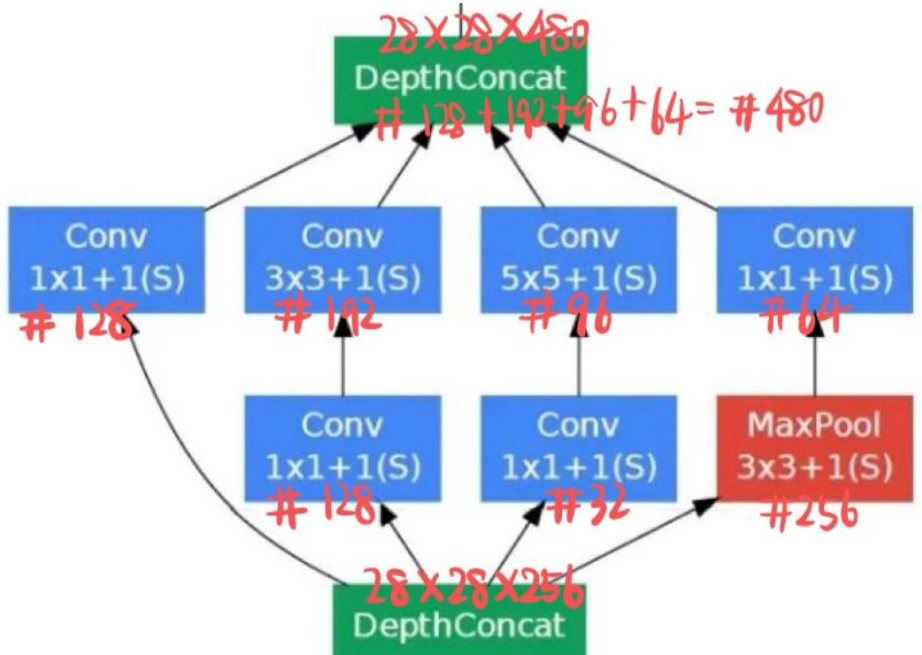
图　解

Title： Inception(3b)

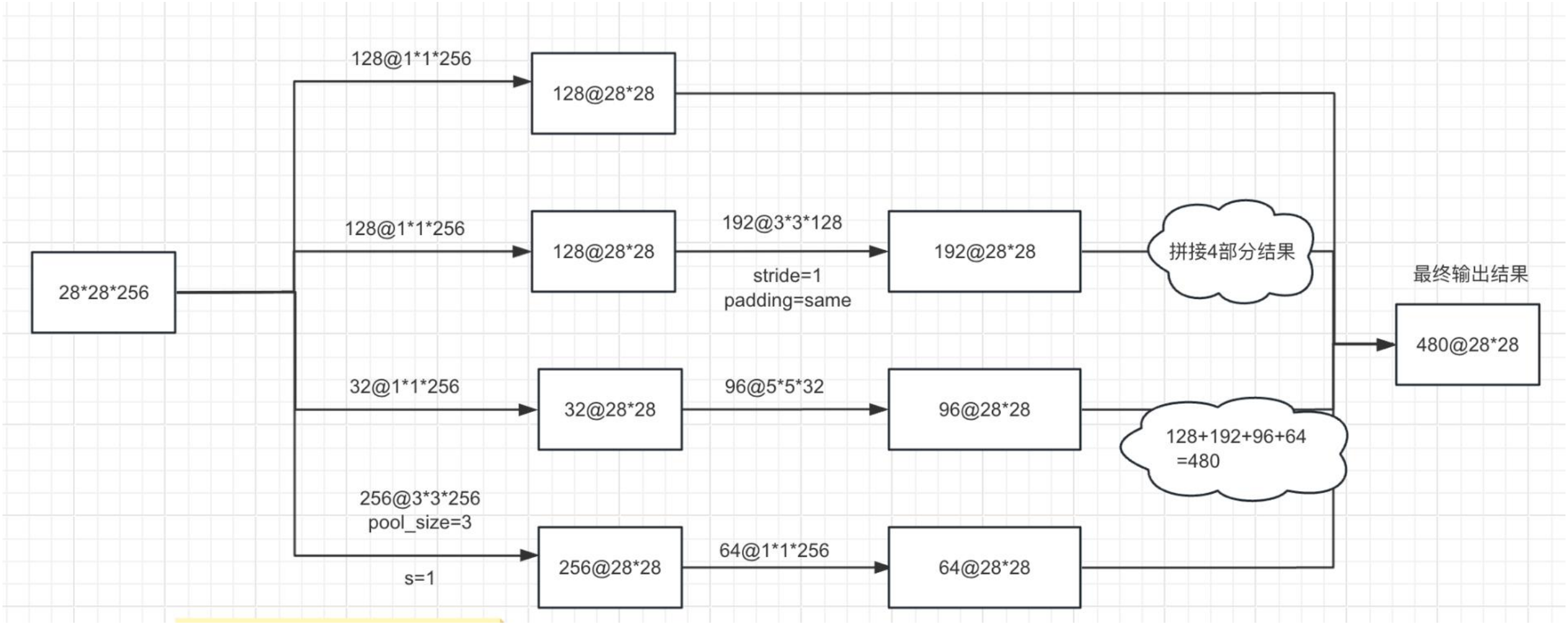
GoogLeNet

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

inception (3b)



Title： Inception(3b)



Title：Inception(3b)

文字描述

Title：Inception(3b)

Inception 3b层，分为四个分支，采用不同尺度。

（1）128个1x1的卷积核，然后ReLU，输出28x28x128

（2）128个1x1的卷积核（3x3卷积核之前的降维）变成28x28x128，进行ReLU，再进行192个3x3的卷积，输出28x28x192

（3）32个1x1的卷积核（5x5卷积核之前的降维）变成28x28x32，进行ReLU，再进行96个5x5的卷积，输出28x28x96

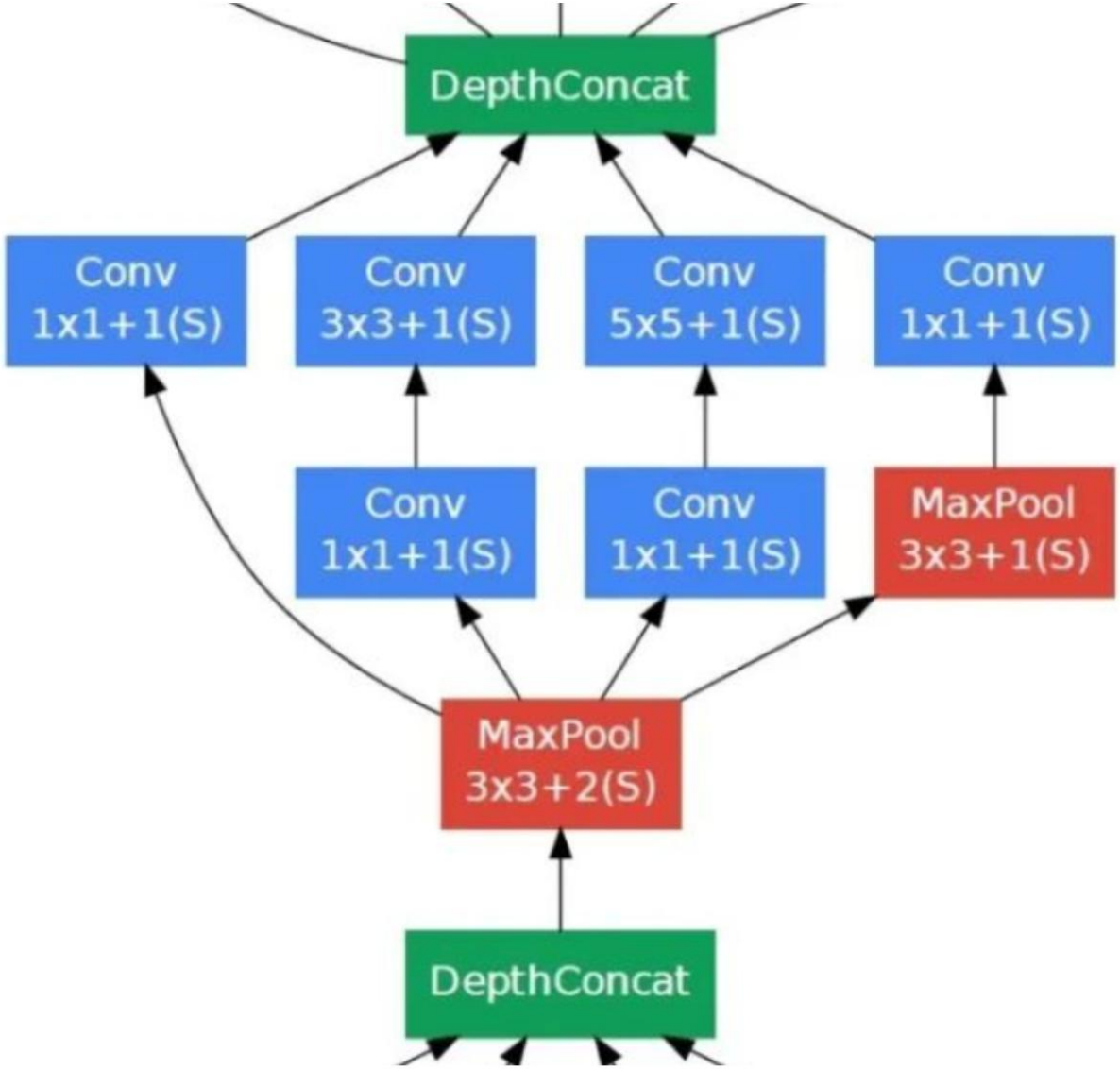
（4）pool层，使用3x3的核，输出28x28x256，然后进行64个1x1的卷积，输出28x28x64

将四个结果进行连接，对这四部分输出结果的第三维并联，即128+192+96+64=480，最终输出为28x28x480。

Inception(4a)

Title： Inception(4a)

inception（4a）



Title： Inception(4a)

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

理解Inception(4a)

Title：Inception(4a)

- 输入：28*28*480；总的输出尺寸 14*14*512
- 进行192个1*1的卷积 输出14*14*192
- 使用3*3的卷积核之前的降维，进行96个1*1的卷积，输出14*14*96，进行208个3*3的卷积，输出14*14*208
- 使用5*5的卷积核之前的降维，进行16个1*1的卷积，输出14*14*16，进行48个5*5的卷积，输出14*14*48
- 池化之后进行64个1*1的卷积，输出14*14*64
- 最后汇聚成512个通道

第四模块、第五模块

Title：第四模块、第五模块

GoogLeNet

第四模块(Inception 4a、4b、4c、4e)

与Inception3a，3b类似

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								

第五模块(Inception 5a、5b)

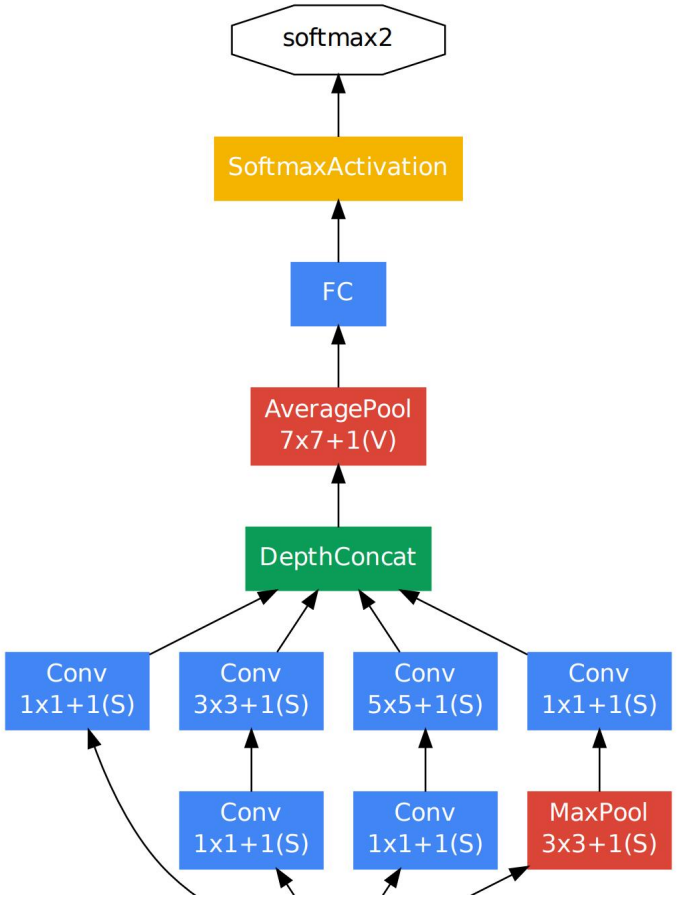
与Inception3a，3b类似

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

输　出　层

Title

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								

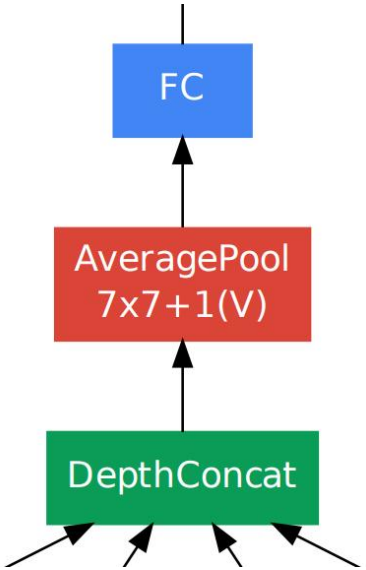
Title：输出层

- 在输出层GoogLeNet与AlexNet、VGG采用3个连续的全连接层不同
- GoogLeNet采用的是全局平均池化层，得到的是高和宽均为1的卷积
- 然后添加丢弃概率为40%的Dropout
- 输出层激活函数采用的是softmax

理 解 G A P

（全 局 平 均 池 化）

Title：全局平均池化



- in:7*7*1024
- deal:pool_size=7*7， stride=1
- out:1*1*1024

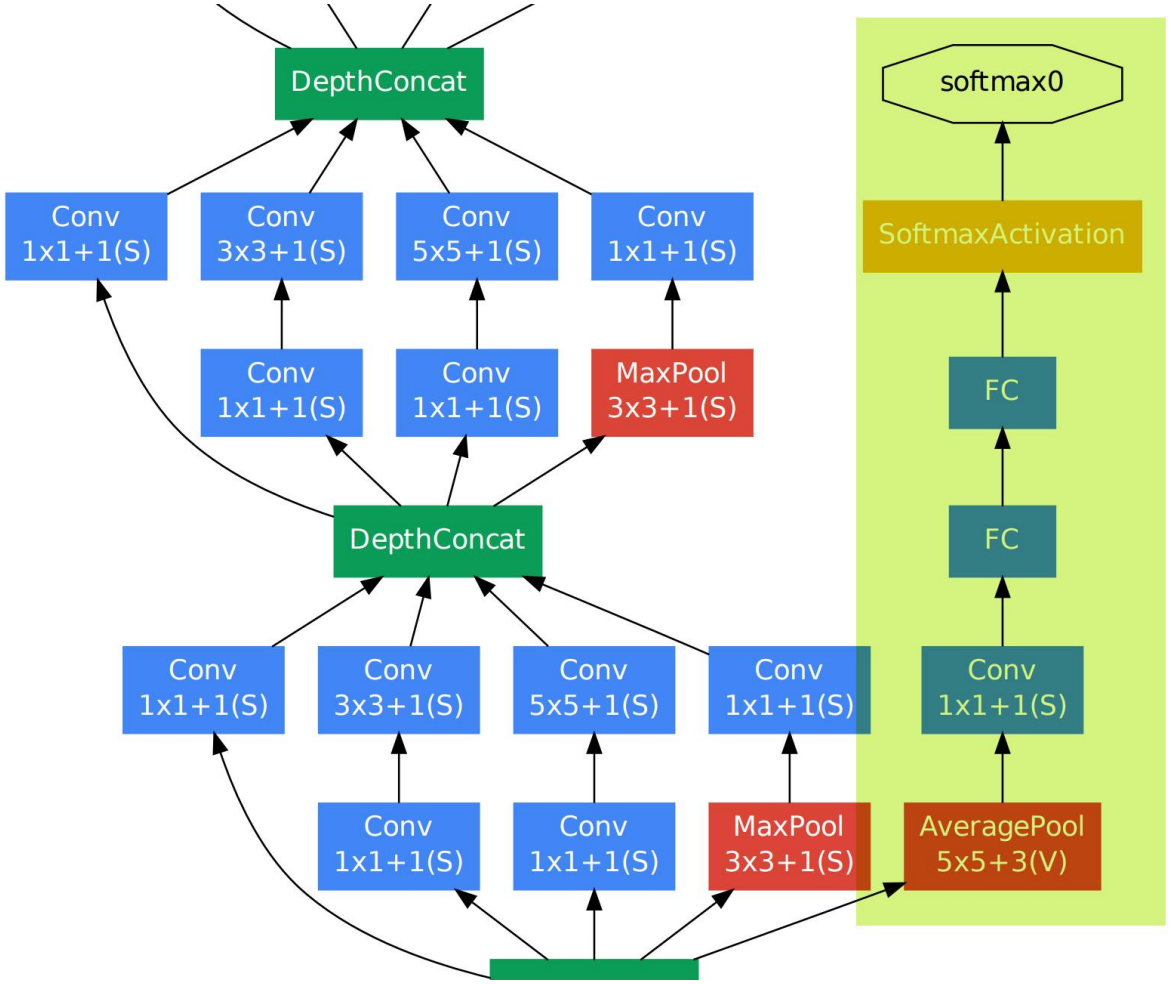
■ 在输出层GoogLeNet为什么采用全局平均池化？

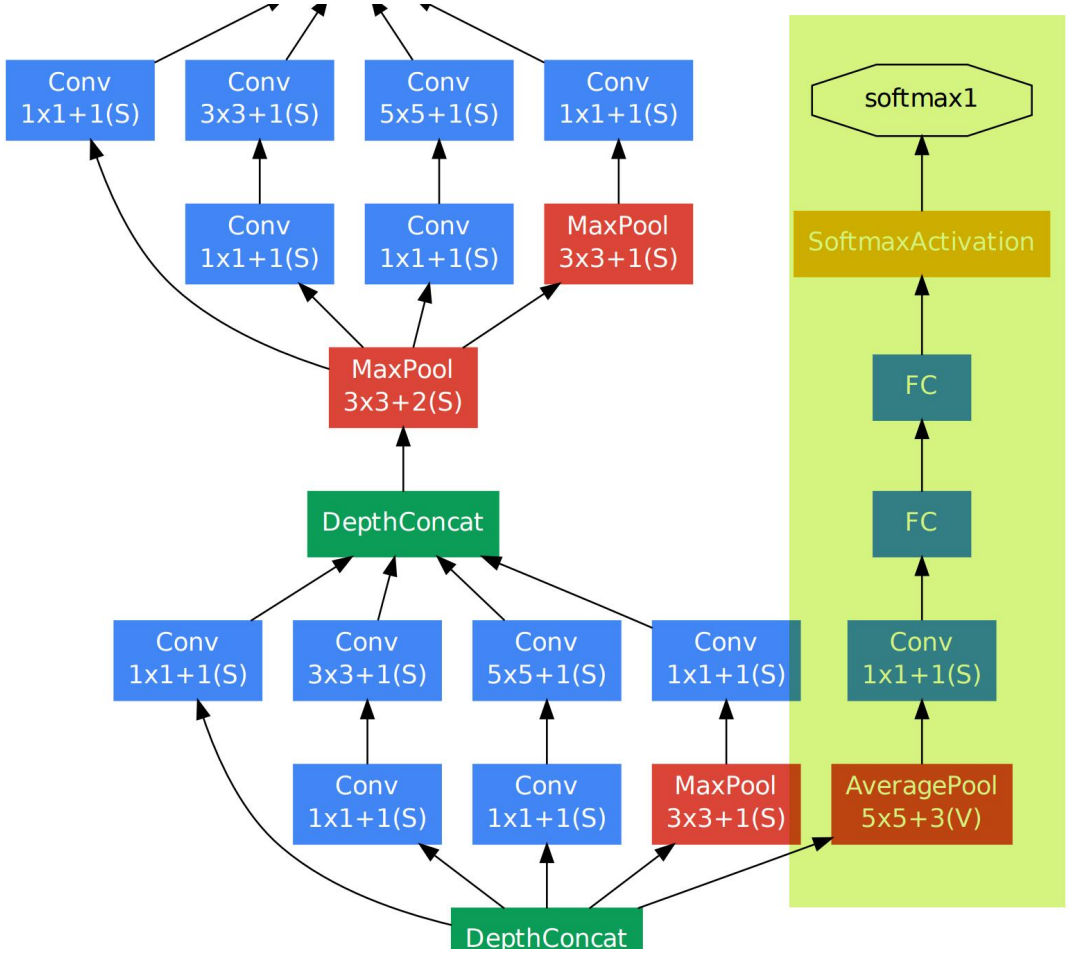
（1）减少过拟合：全局平均池化通过将整个特征图的值平均化，减少了模型的参数数量。

（2）降低参数数量：传统的全连接层会引入大量的参数，而全局平均池化之后的输出直接与分类任务相关，省去了大量全连接层的参数

辅助分类器

Title 辅助分类器长什么样





Title 为什么要设置辅助分类器

- 因为除了最后一层的输出结果，中间节点的分类效果也可能是很好的
- GoogleNet将中间的某一层作为输出，并以一个较小的权重加入到最终分类结果中。
- 相当于一种变相的模型融合，同时给网络增加了反向传播的梯度信号，起到了一定的正则化的作用。

$$Loss = loss_2 + 0.3 * loss_1 + 0.3 * loss_0$$

GoogLeNet的创新点

Title GoogLeNet的创新点

1. Inception模块：

① 多分支结构：使用不同大小的卷积核和池化操作来捕获不同尺度的特征；

② 使得网络能够同时学习到局部细节和全局特征。

2. 全局平均池化：

① 输出层采用了全局平均池化代替传统的全连接层；

② 降低了参数数量，减轻了过拟合的风险。

Title GoogLeNet的创新点

3. 稀疏连接：

① 将部分卷积操作替换为1x1的卷积核，减少特征图的通道数

② 降低计算复杂度

4. 辅助分类器：

① 辅助分类器在训练期间引入额外的梯度信号

② 有助于加速网络收敛，解决深层网络训练过程中的梯度消失问题

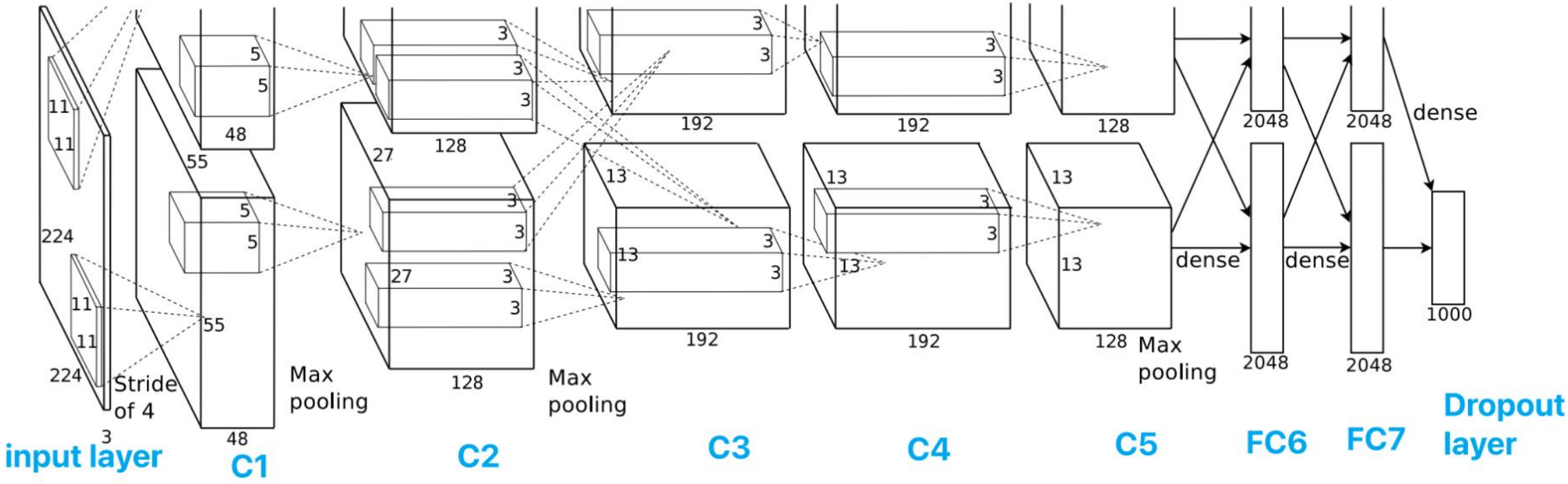
Title：GoogLeNet的评价

GoogLeNet在网络模型方面与AlexNet、VGG还是有一些相通之处的，

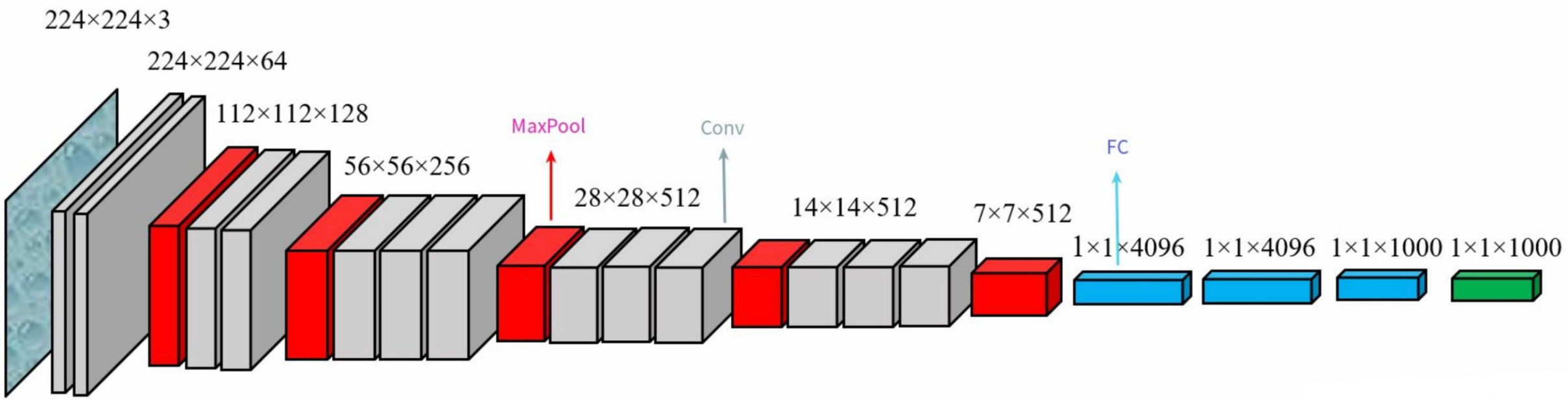
它们的主要相通之处就体现在卷积部分：

- AlexNet采用5个卷积层
- VGG把5个卷积层替换成5个卷积块
- GoogLeNet采用5个不同的模块组成主体卷积部分

Title AlexNet、VGG、GoogleNet



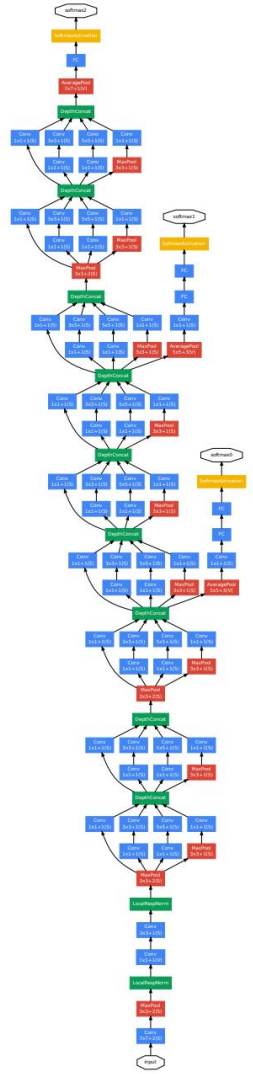
Title AlexNet、VGG、GoogleNet



Title AlexNet、VGG、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								

Table 1: GoogLeNet incarnation of the Inception architecture



Title

ResNet

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

paper: https://arxiv.org/pdf/1512.03385v1

提出背景

Title ResNet提出的背景

退化问题

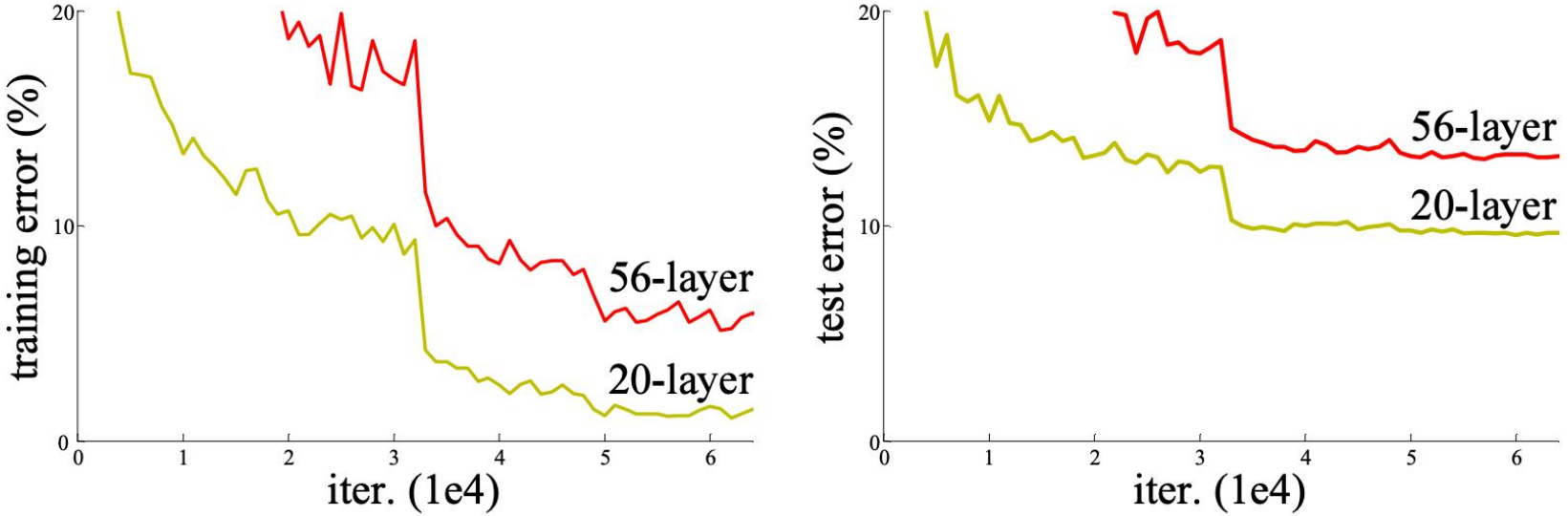


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

解决方法

跳跃连接

- 随着层数的增加，预测效果反而越来越差。

Title ResNet提出的背景

梯度问题

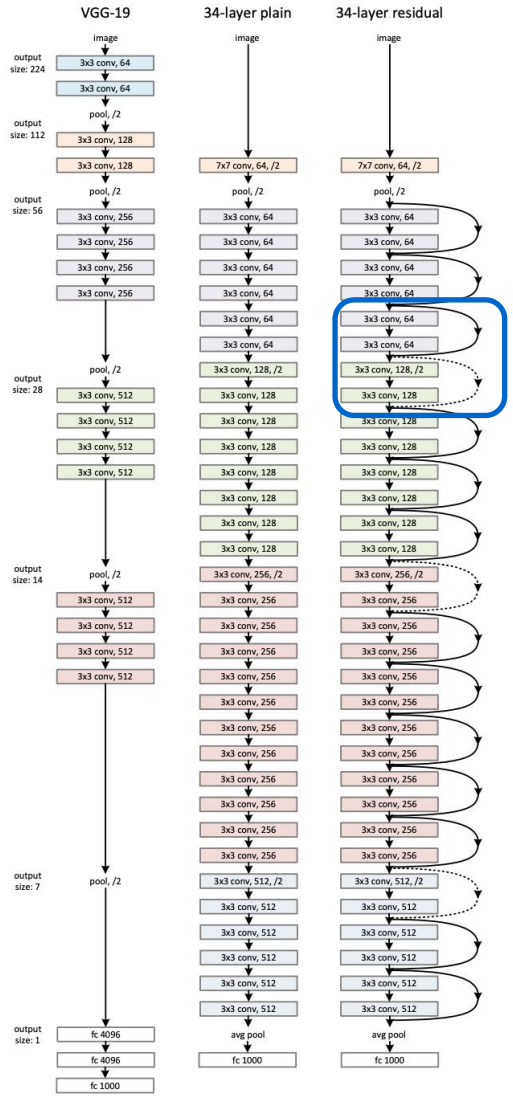
梯度消失和梯度爆炸

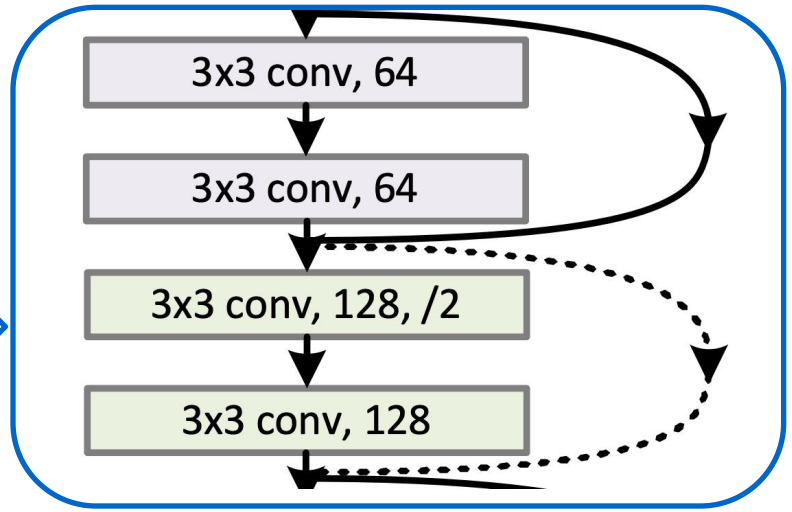
- 梯度消失：若每一层的误差梯度小于1，反向传播时，网络越深，梯度越趋近于0
- 梯度爆炸：若每一层的误差梯度大于1，反向传播时，网络越深，梯度越来越大

解决方法 **Batch Normalization**

网络架构

Title ResNet网络架构





基本残差块

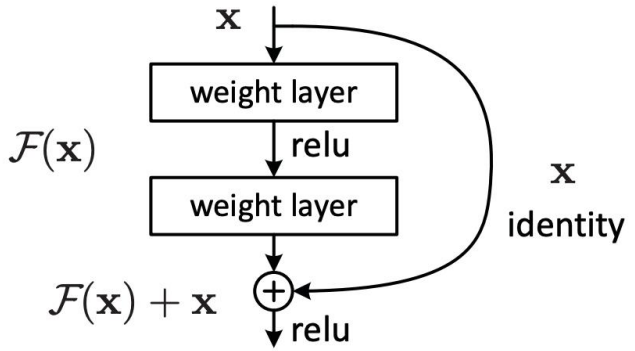
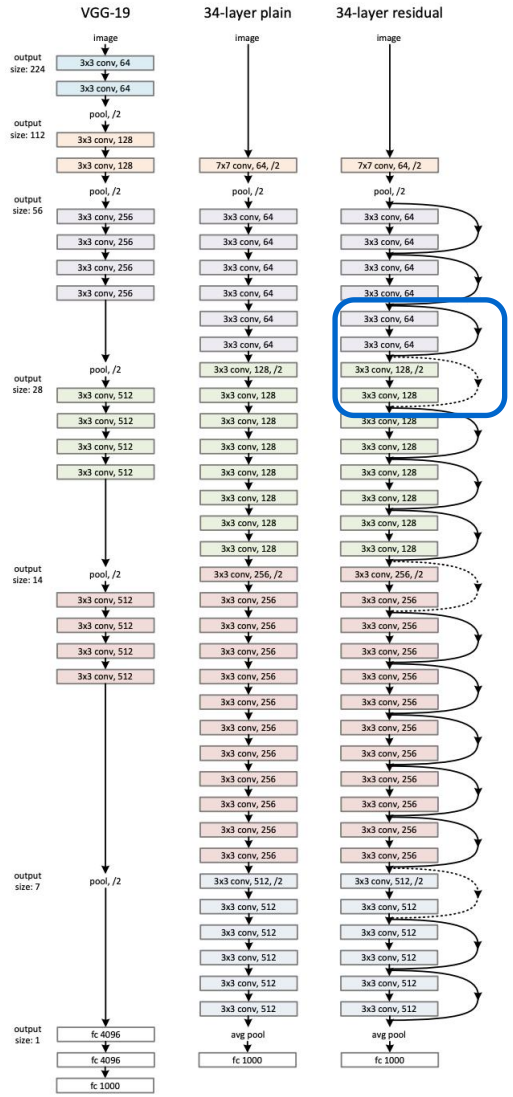


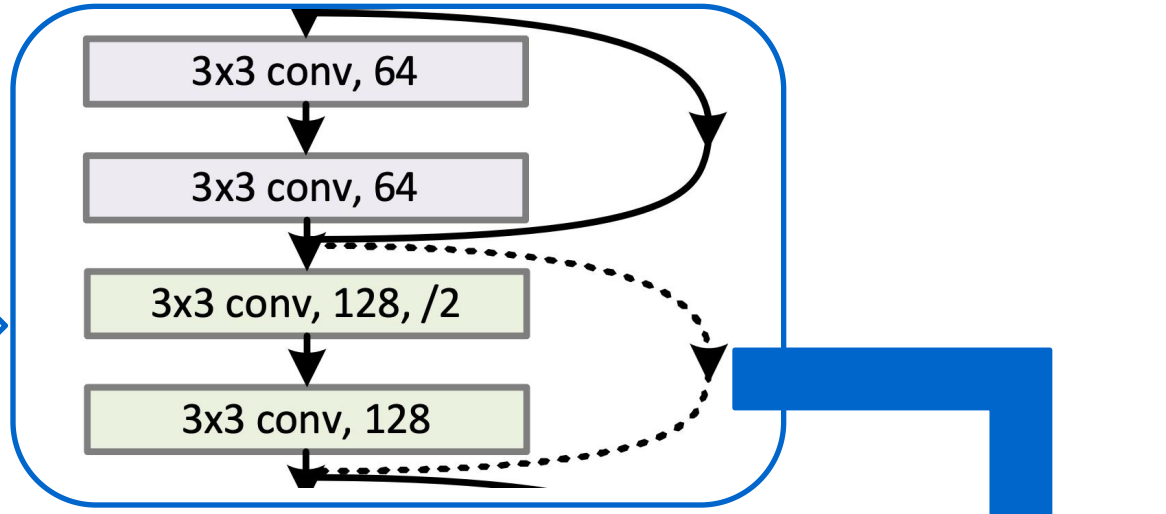
Figure 2. Residual learning: a building block.

$H(x) = F(x) + x$

- 每次都至少保证不比以前差
- 如果权重层不好，极端的情况下就是设置为0，那么下一层就等于上一层

Title ResNet网络架构





虚线

- 理由
  - 前后维度不匹配
- 解决
  - 虚线部分做1*1的卷积，特征图个数翻倍

残差网的贡献

Title ResNet的主要贡献

- 超深的网络结构（超过1000层）；
- 提出residual（残差结构）模块；
- 使用Batch Normalization 加速训练（丢弃dropout）。

Title

DenseNet

CVPR2017年的Best Paper

原文地址:
https://arxiv.org/pdf/1608.06993

Title DenseNet网络架构

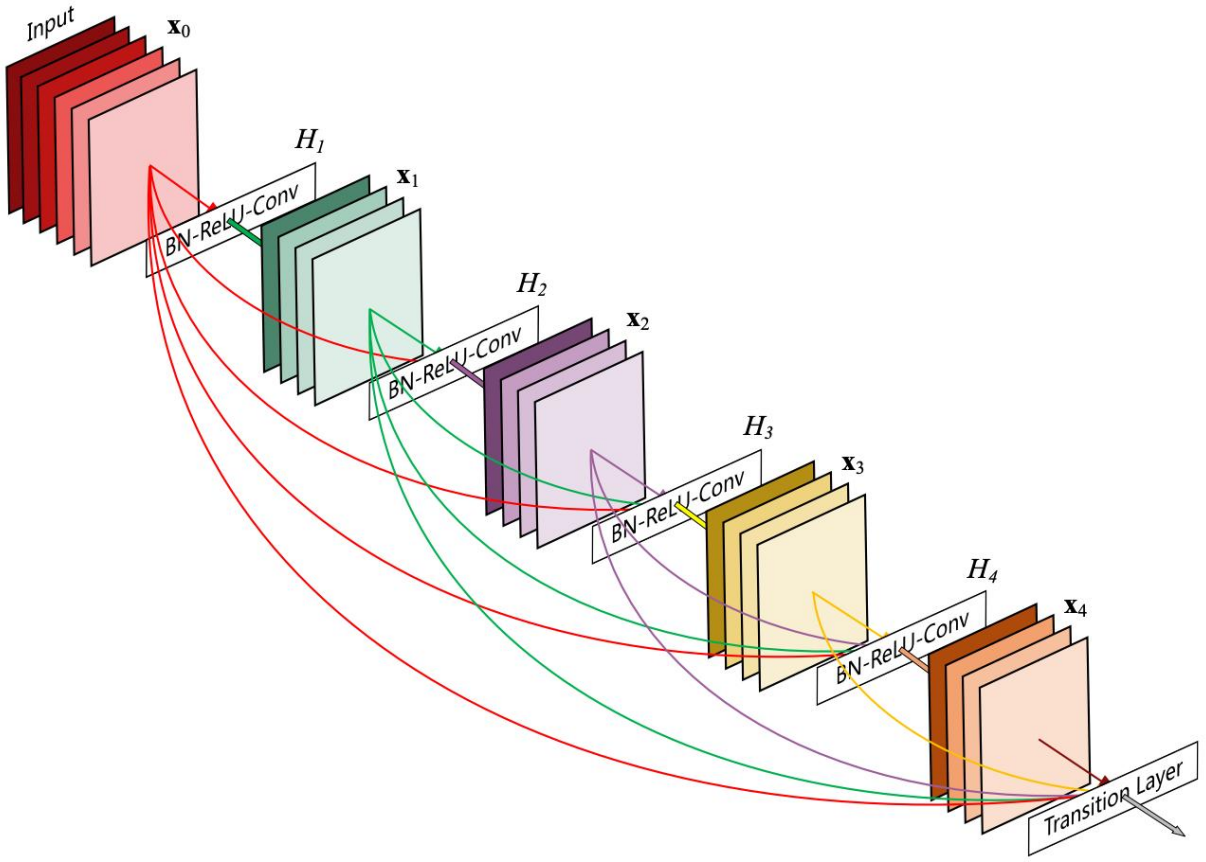


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

读图：

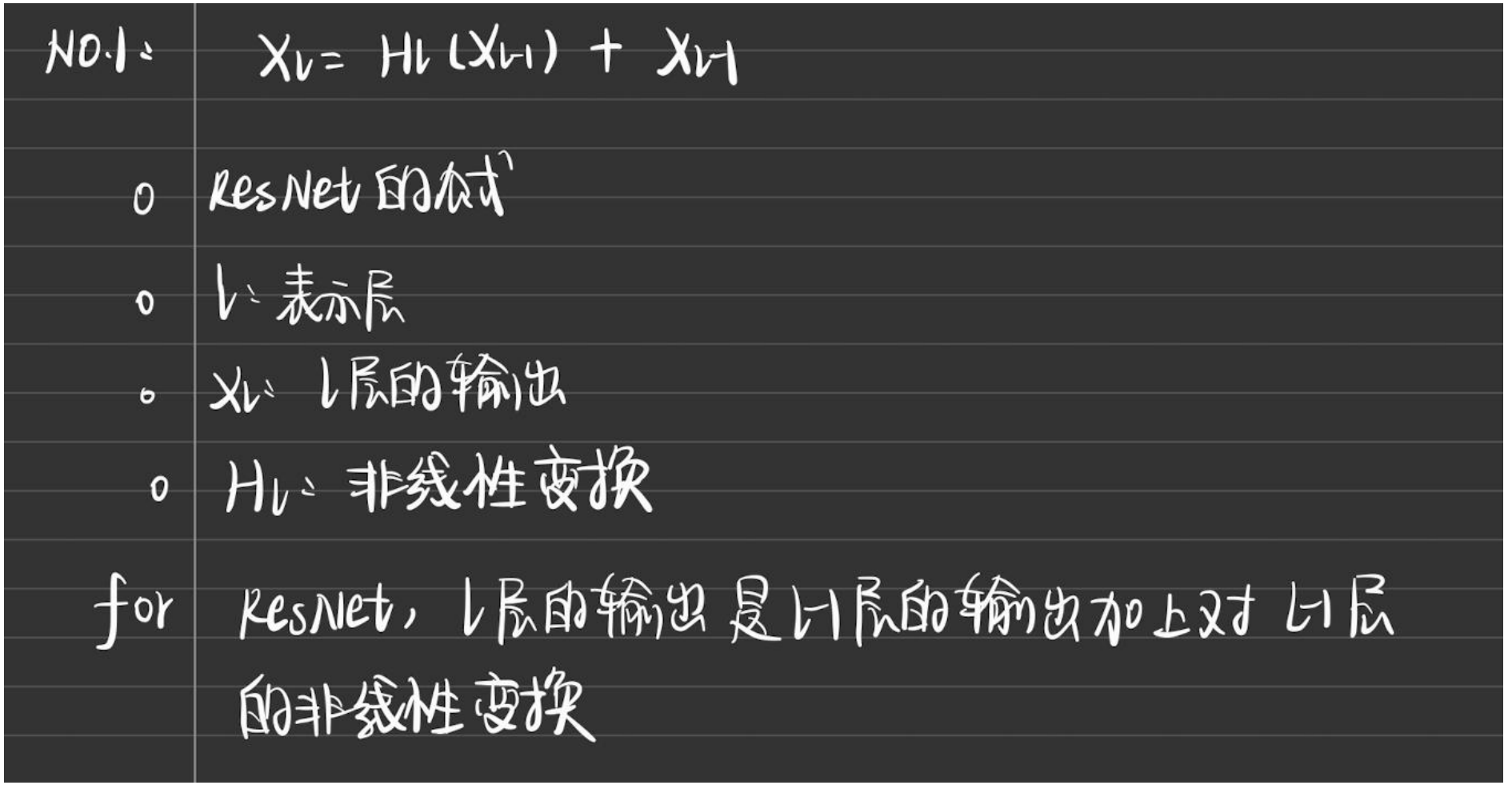
- x0是input
- H1的输入是x0 (input)
- H2的输入是x0和x1 (x1是H1的输出)

Title DenseNet

- 传统卷积网，网络有L层，就会有L个连接
- DenseNet中，会有$\frac{L(L+1)}{2}$个连接↔每一层的输入来自前面所有的层的输出

Title DenseNet

- 原文仅有的两个公式，通过这两个公式理解 ResNet 和 DenseNet



Title DenseNet

NO2.

$X_l = H_l([X_0, X_1 \cdots X_{l-1}])$

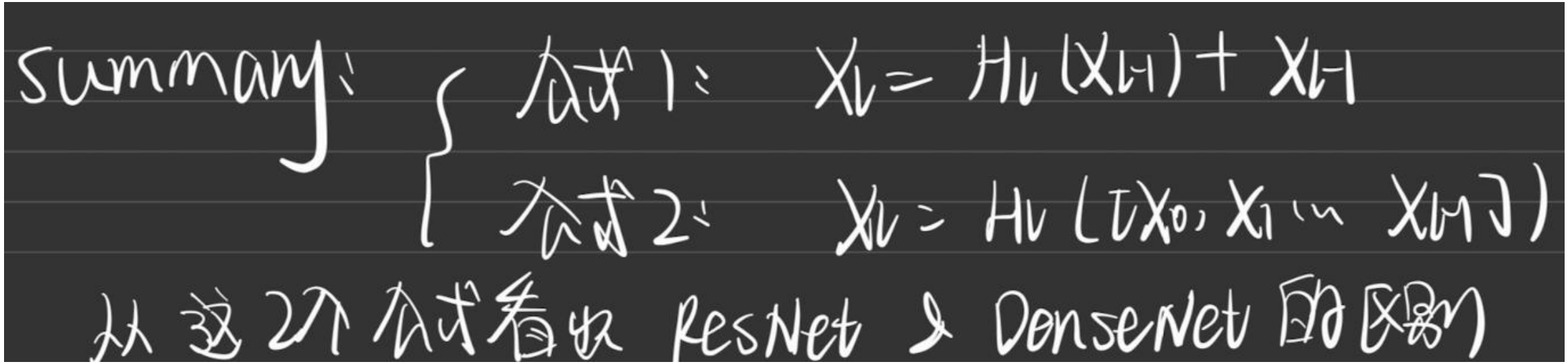
- DenseNet 的公式

- $[X_0, X_1 \cdots X_{l-1}]$：将 0～l-1 层的输出 feature map 做 concatenation

- what concatenation?

  - 做通道的公并 类似 inception
  - ResNet 做值的相加，通道数不变
  - H_l：包括 BN，ReLU 和 3*3 的 conv

Title DenseNet



DenseNet主要贡献

Title DenseNet评价

- 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

ResNet v.s. DenseNet

Title ResNet v.s. DenseNet

- 何恺明 ResNet 的假设：若某一较深的网络多出另一较浅网络的若干层有能力学习到恒等映射，那么这一较深网络训练得到的模型性能一定不会弱于该浅层网络
  - = 如果对某一网络中增添一些可以学到恒等映射的层组成新的网络，那么最差的结果也就是新网络中的这些层在训练后成为恒等映射而不会影响原网络的性能
- DenseNet假设：（特征复用） 与其多次学习冗余的特征，特征复用是一种更好的特征提取方式

蘭州大學
LANZHOU UNIVERSITY

时序模型

指导教师：　学生：@dearRongerr

2024年5月6日

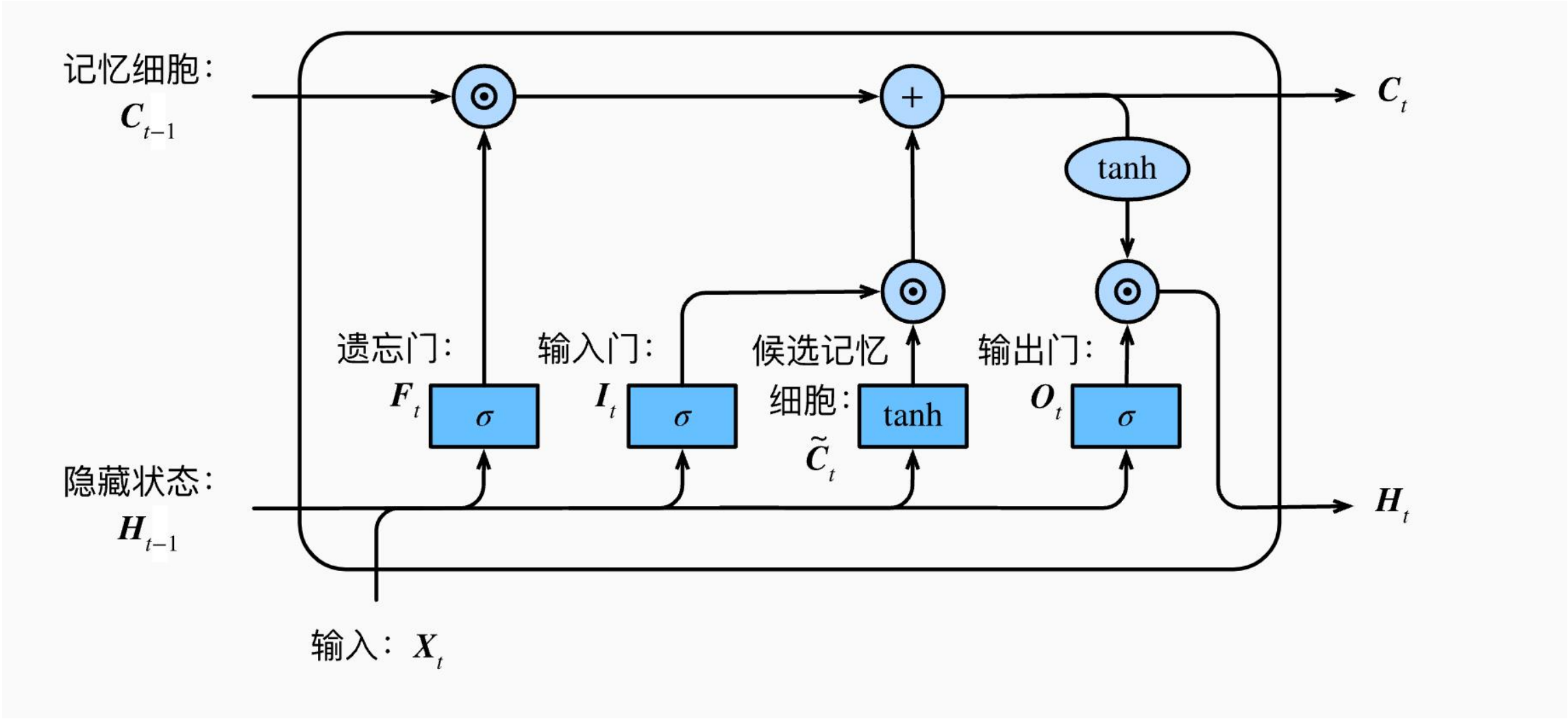
Title

R N N

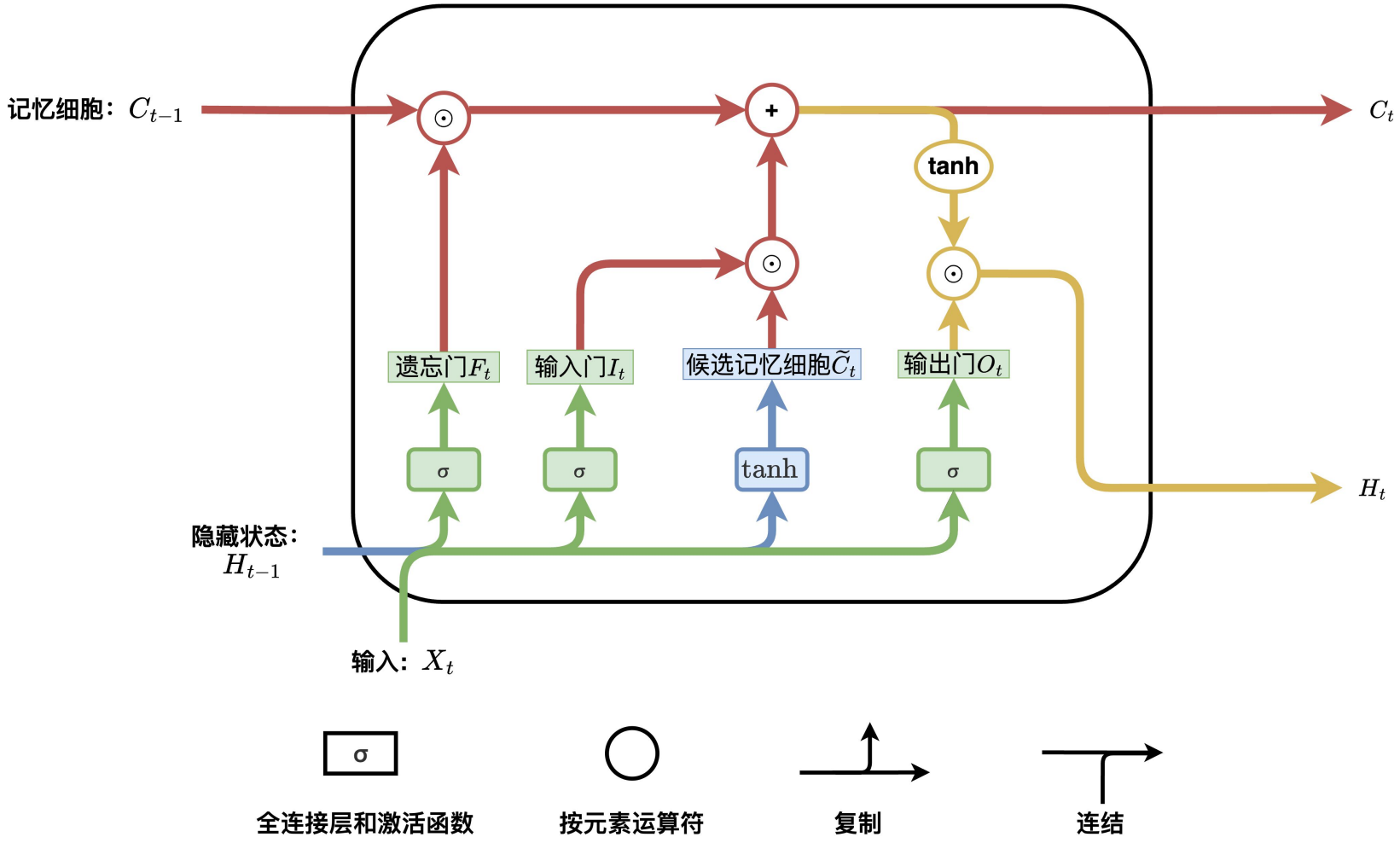
Title

L S T M

Title LSTM长什么样？

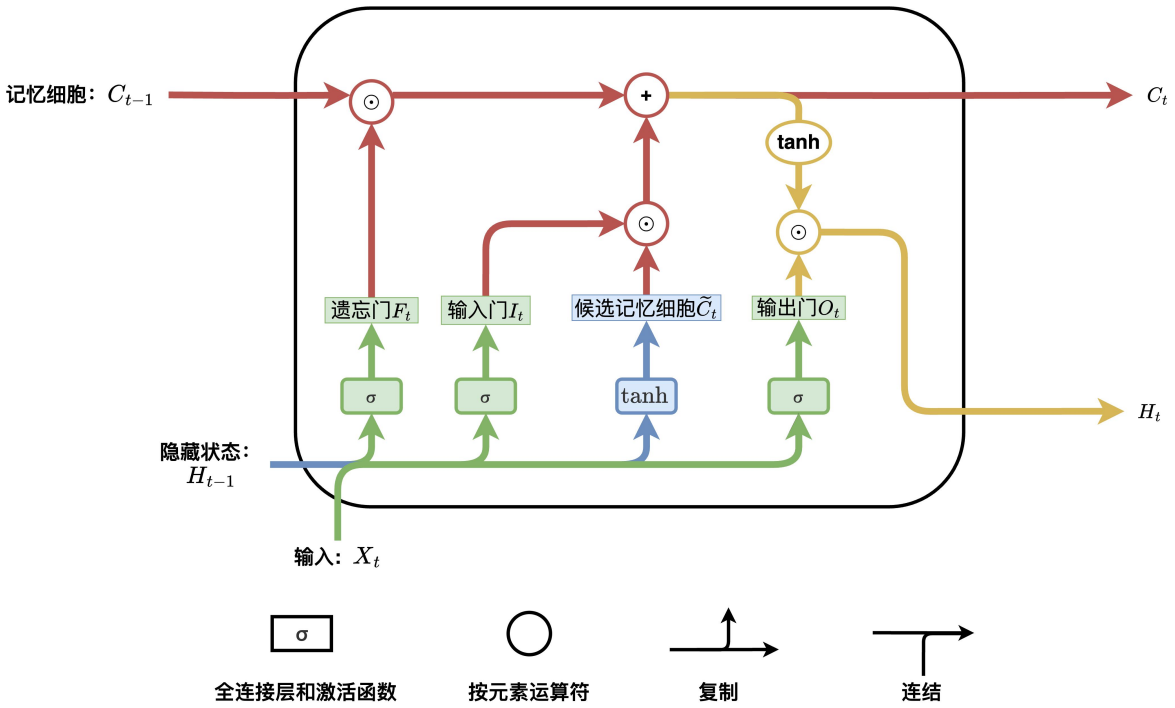


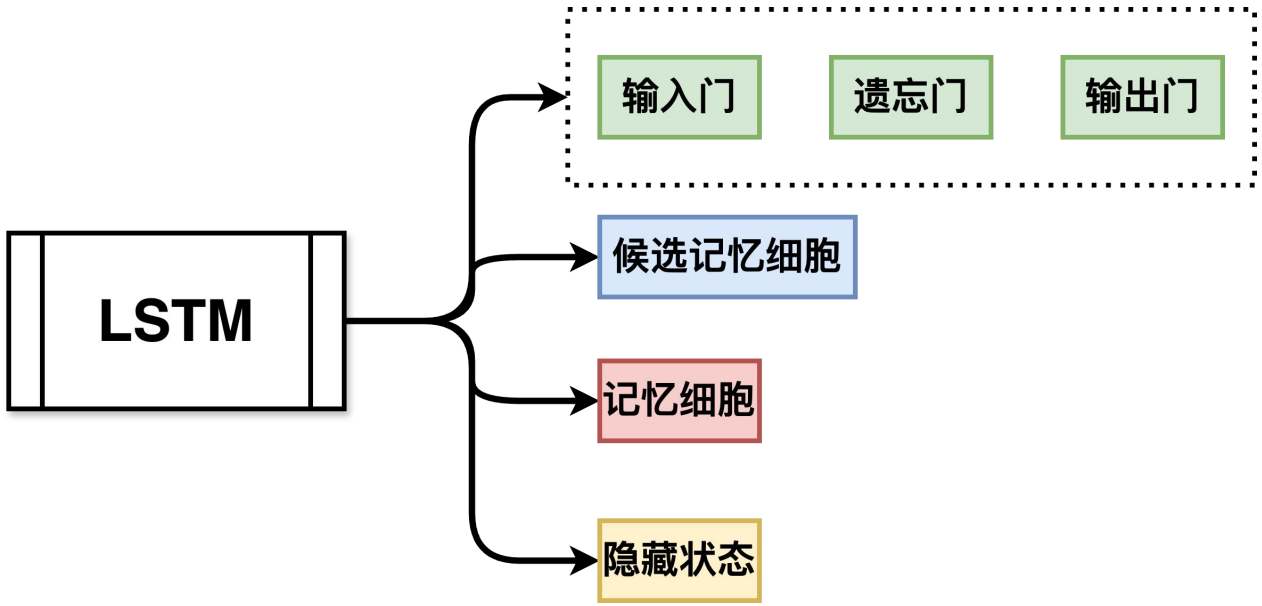
Title 数据流动


记忆细胞：C_{t-1}
C_t
tanh
遗忘门F_t
输入门I_t
候选记忆细胞$\tilde{C}_t$
输出门O_t
σ
σ
tanh
σ
H_t
隐藏状态：
H_{t-1}
输入：X_t
σ
全连接层和激活函数
按元素运算符
复制
连结

Title LSTM

各部分拆解





Title

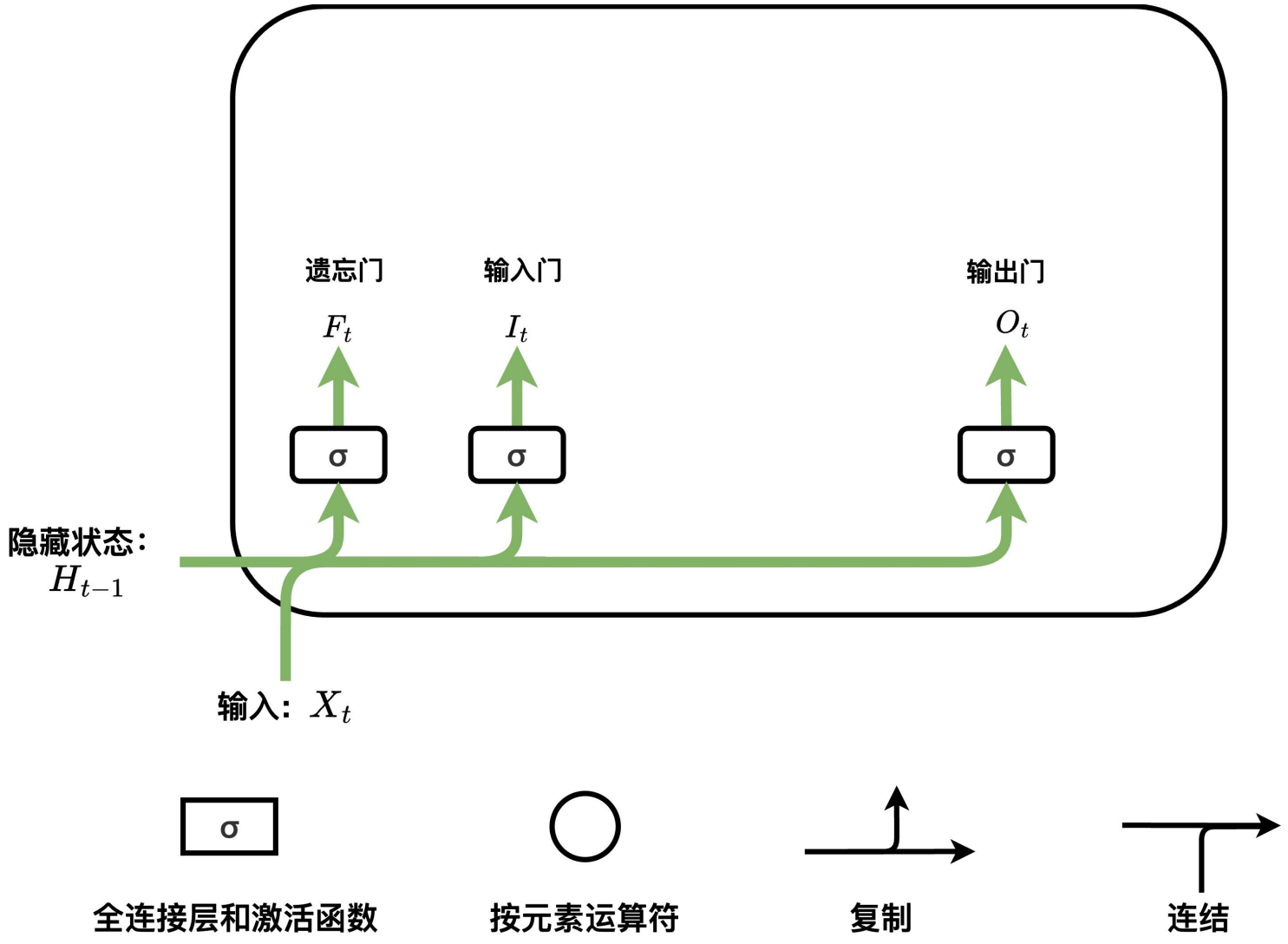
输入门、遗忘门、输出门

Title 输入门、遗忘门、输出门

LSTM网络结构

基本组件：

- 即输入门（input gate）
- 遗忘门（forget gate）
- 输出门（output gate）



Title 输入门、遗忘门、输出门

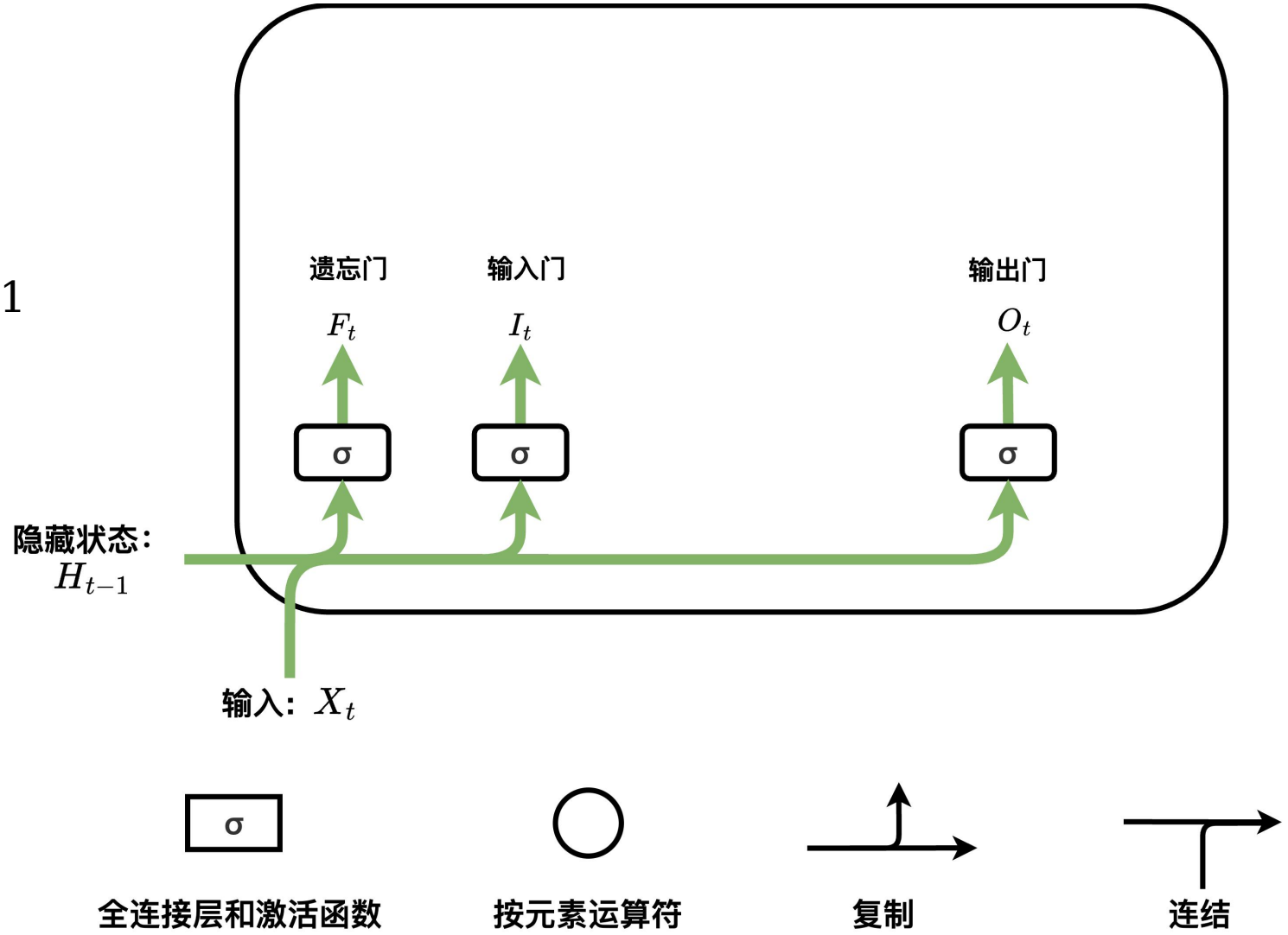
LSTM网络结构

- in：
  - 当前时间步输入 $\mathrm{x_t}$
  - 上一时间步隐藏状态 $\mathrm{H_{t-1}}$
- deal：
  - 激活函数sigmoid

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$

$$\boldsymbol{F}_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_0)$$



Title

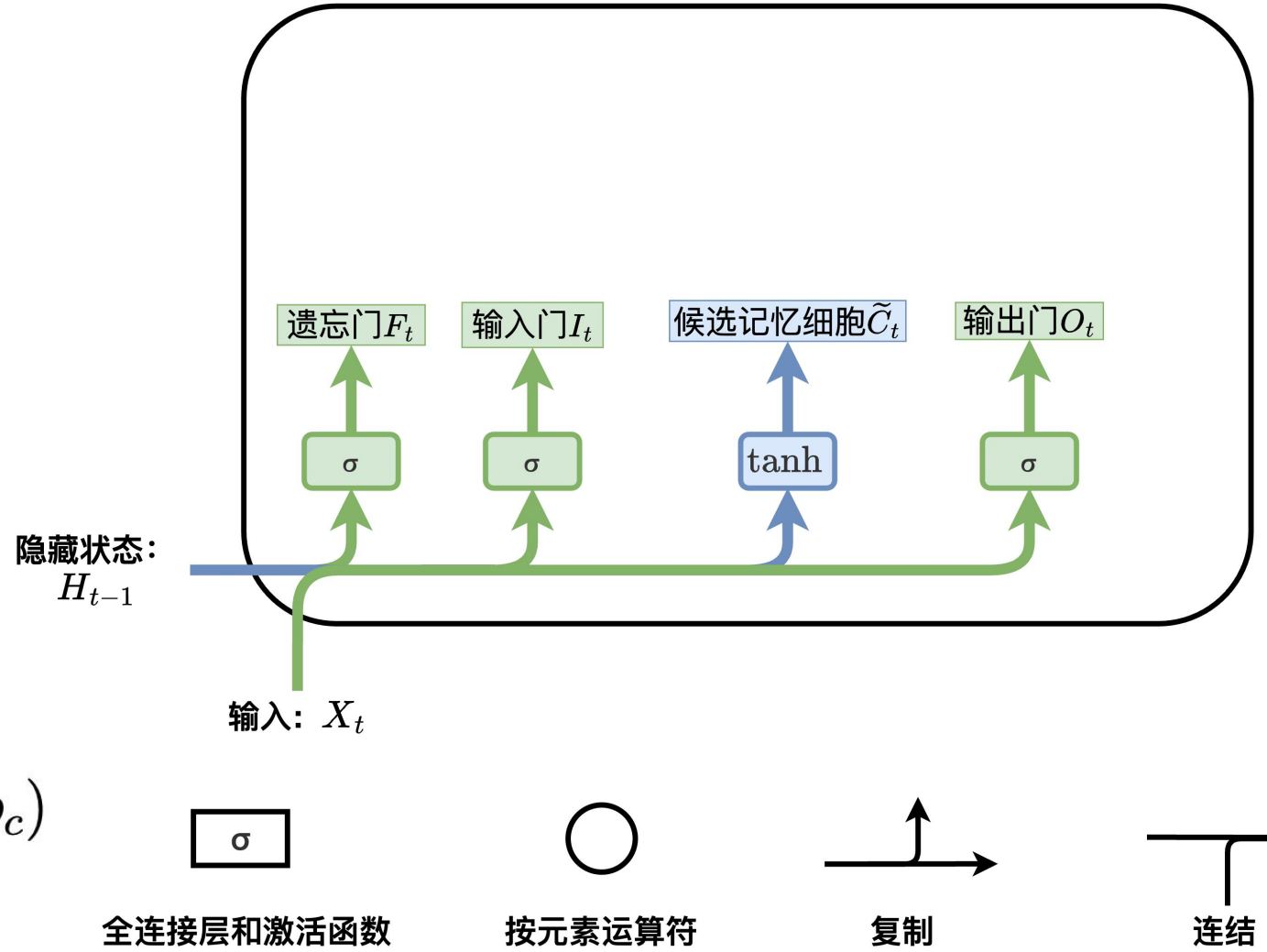
候选记忆细胞

Title 候选记忆细胞

LSTM网络结构

- in：
  - 当前时间步输入 $\mathrm{x_t}$
  - 上一时间步隐藏状态 $\mathrm{H_{t-1}}$
- deal：
  - 激活函数tanh

$$\tilde{C}_t = tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$



Title

记忆细胞

Title 记忆细胞

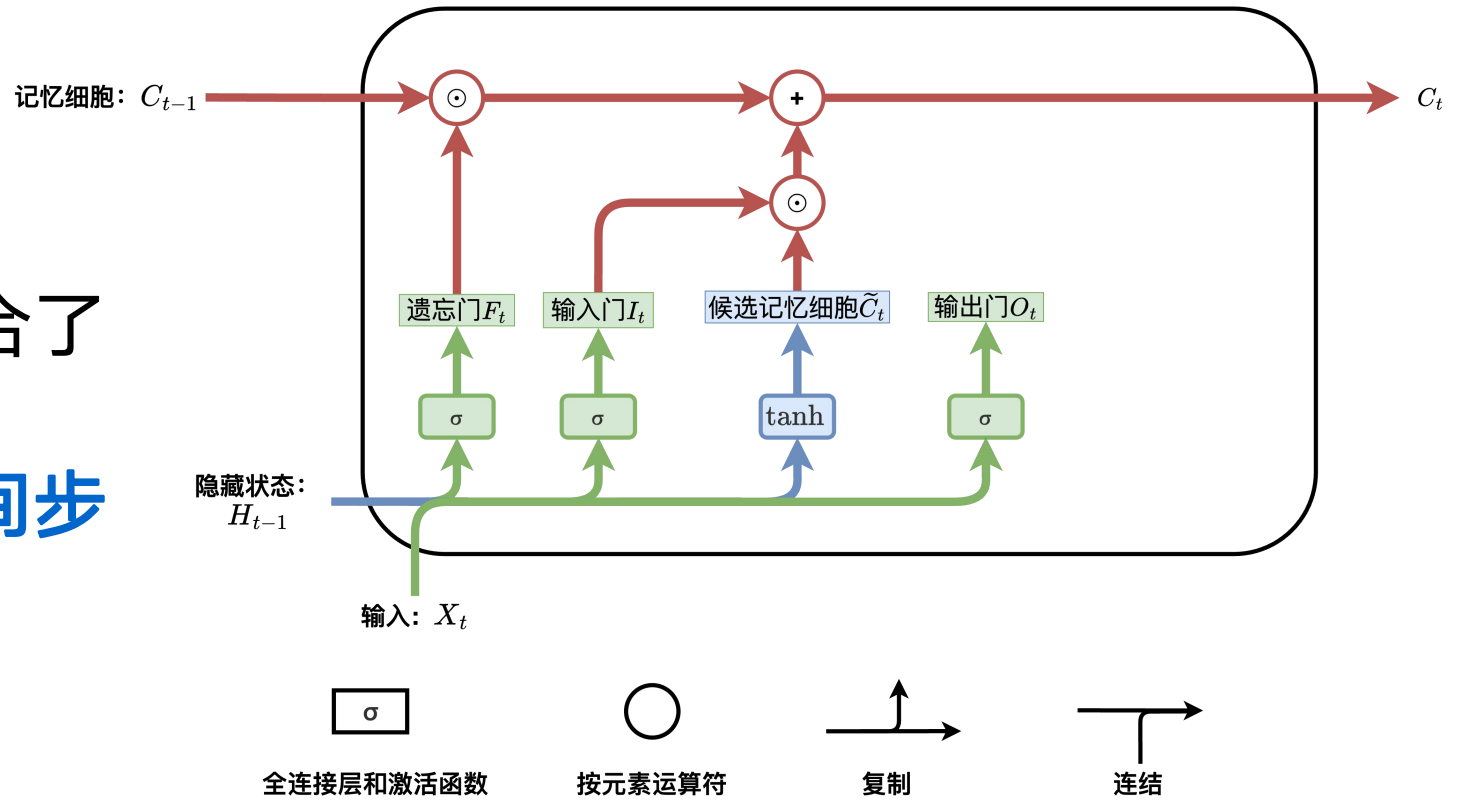
LSTM网络结构

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

当前时间步记忆细胞的计算组合了

上一时间步记忆细胞和当前时间步

候选记忆细胞的信息



上一时间步记忆细胞

- 遗忘门控制着前一个时间步的记忆是否被遗忘或保留。
- 当遗忘门的输出接近1时，意味着几乎不会丢弃过去的记忆，而是保留它们传递到当前时间步。
- 这可以帮助网络捕捉长期的依赖关系和记忆模式。

当前时间步候选记忆细胞的信息

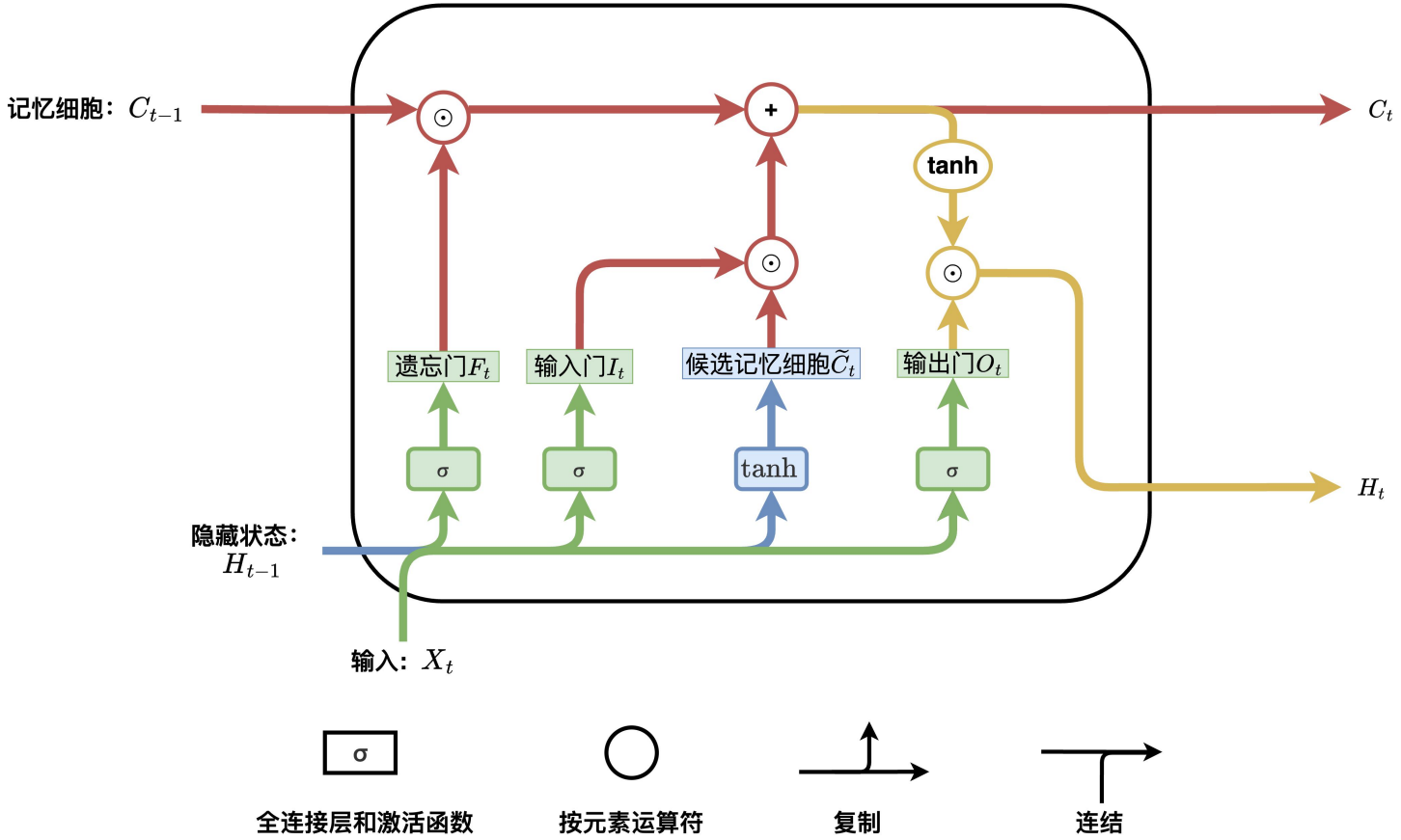
- 输入门控制着新的输入信息对当前时间步的影响。当输入门的输出接近0时，意味着几乎不会接受新的输入信息，因此过去的记忆细胞可以持续传递至当前时间步，而不被新信息所替代。

- 如果遗忘门一直近似 1 且输入门一直近似 0，过去的记忆细胞将一直通过时间保存并传递至当前时间步；
- 优点：更好地捕捉时间序列中时间步距离较大的依赖关系。

Title

隐 藏 状 态

Title 隐藏状态



- 当输出门近似1时，记忆细胞信息将传递到隐藏状态供输出层使用；
- 当输出门近似0时，记忆细胞信息只自己保留。

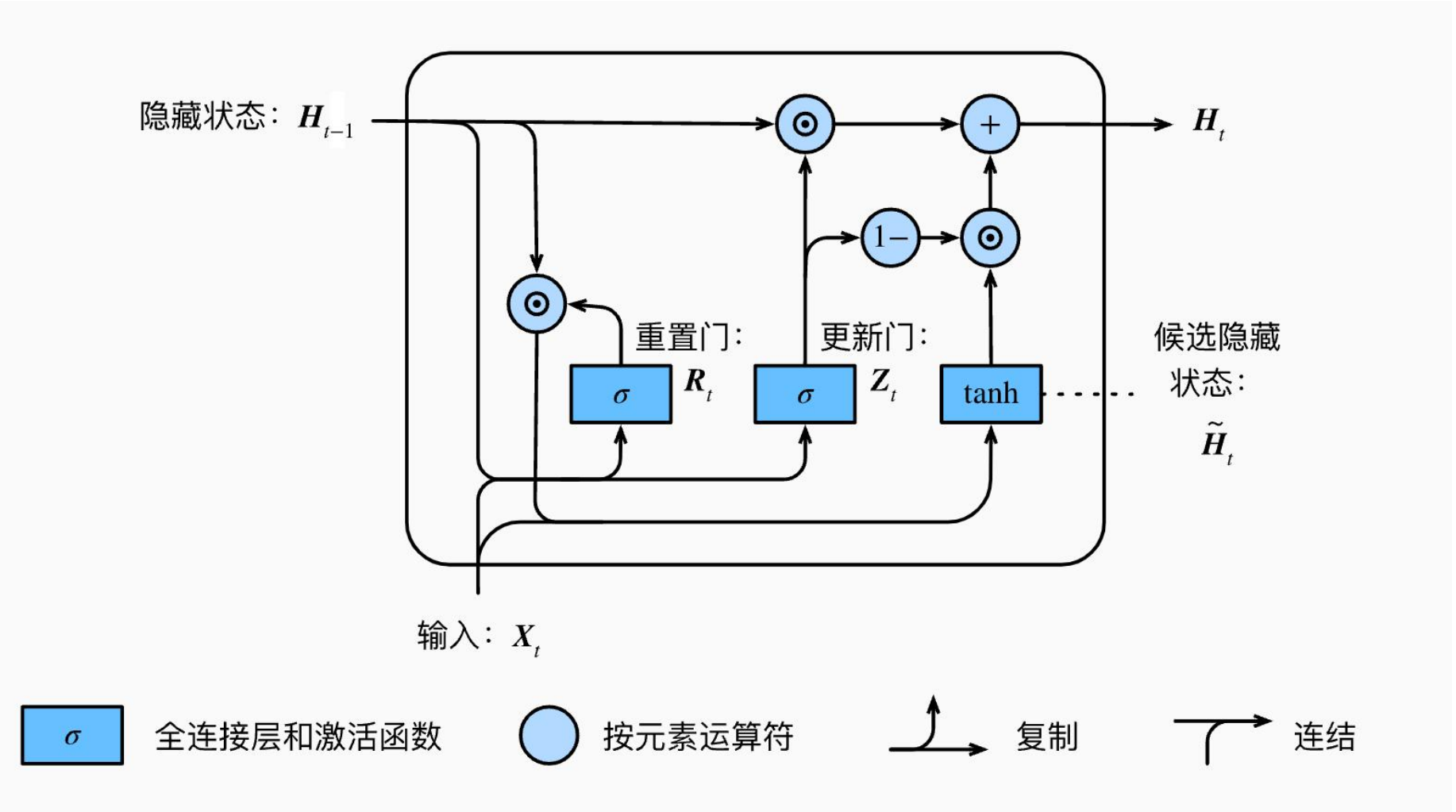
$$H_t = O_t \odot tanh(C_t)$$

Title

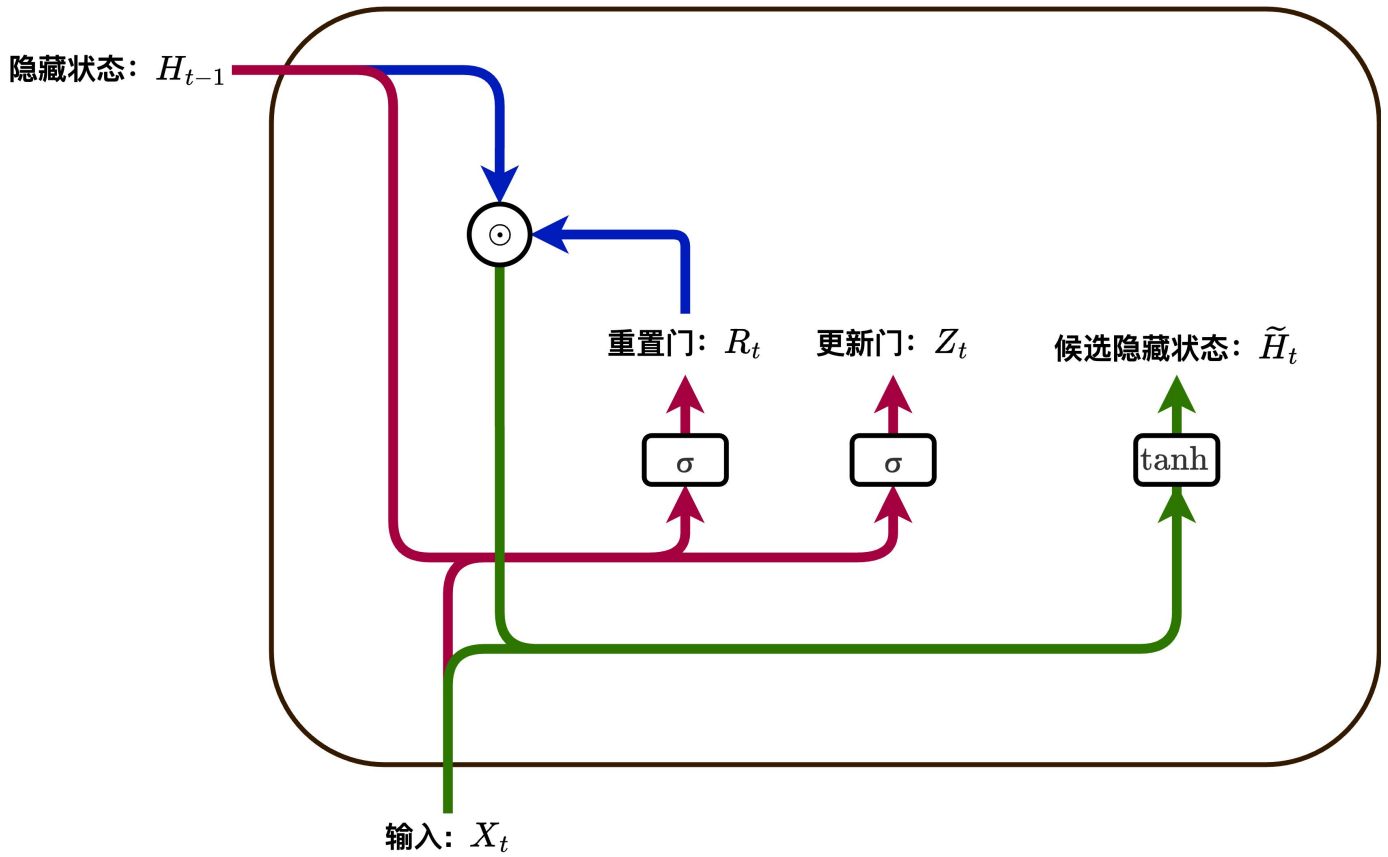
G R U

（gated recurrent neural network）

Title GRU长什么样？

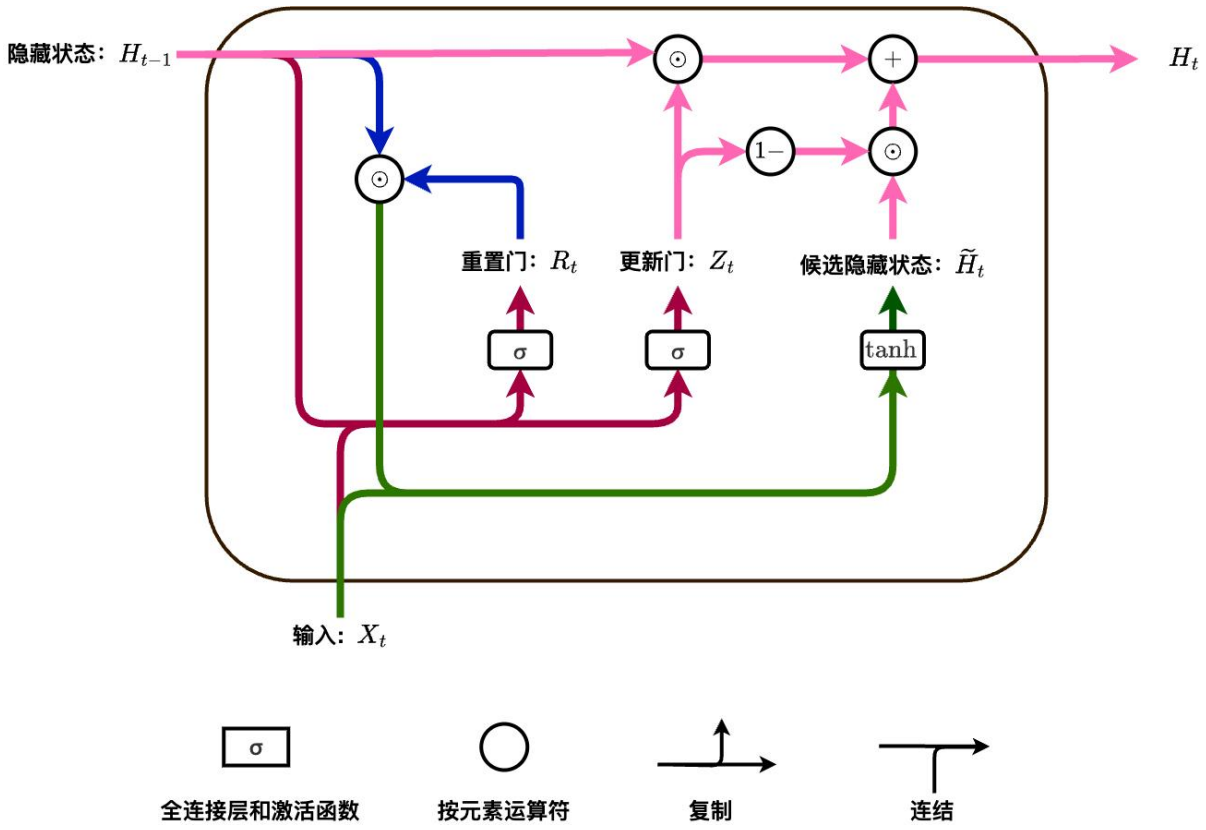


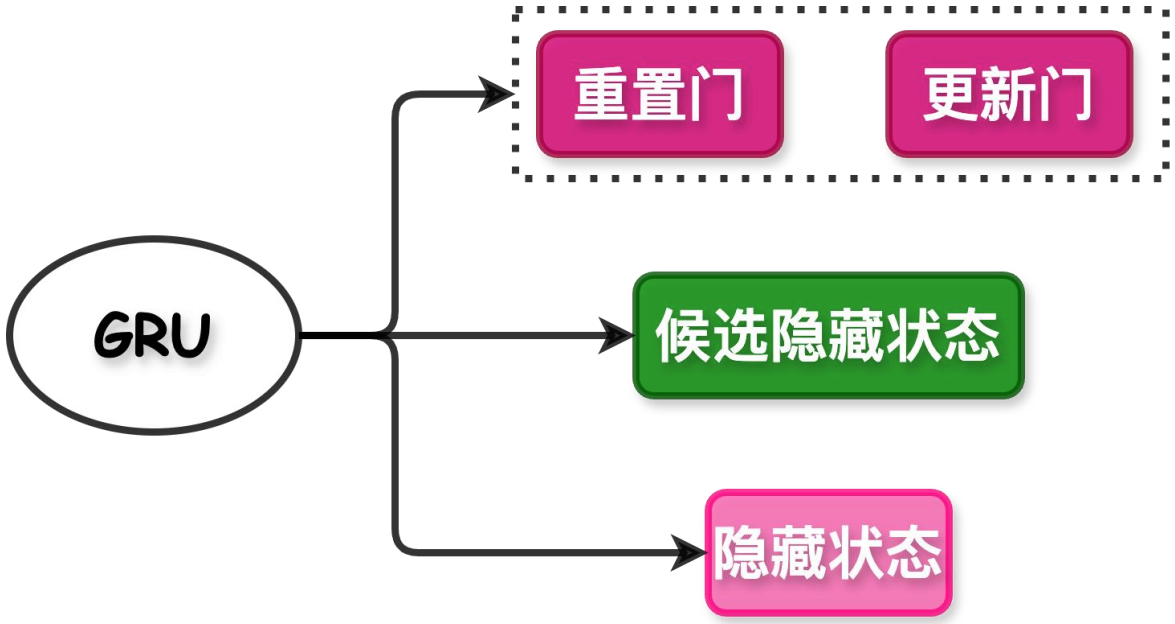
Title 数据流动


隐藏状态：H_{t-1}
重置门：R_t
更新门：Z_t
候选隐藏状态：$\widetilde{H}_t$
σ
σ
tanh
输入：X_t
σ
全连接层和激活函数
按元素运算符
复制
连结

Title GRU

各部分拆解

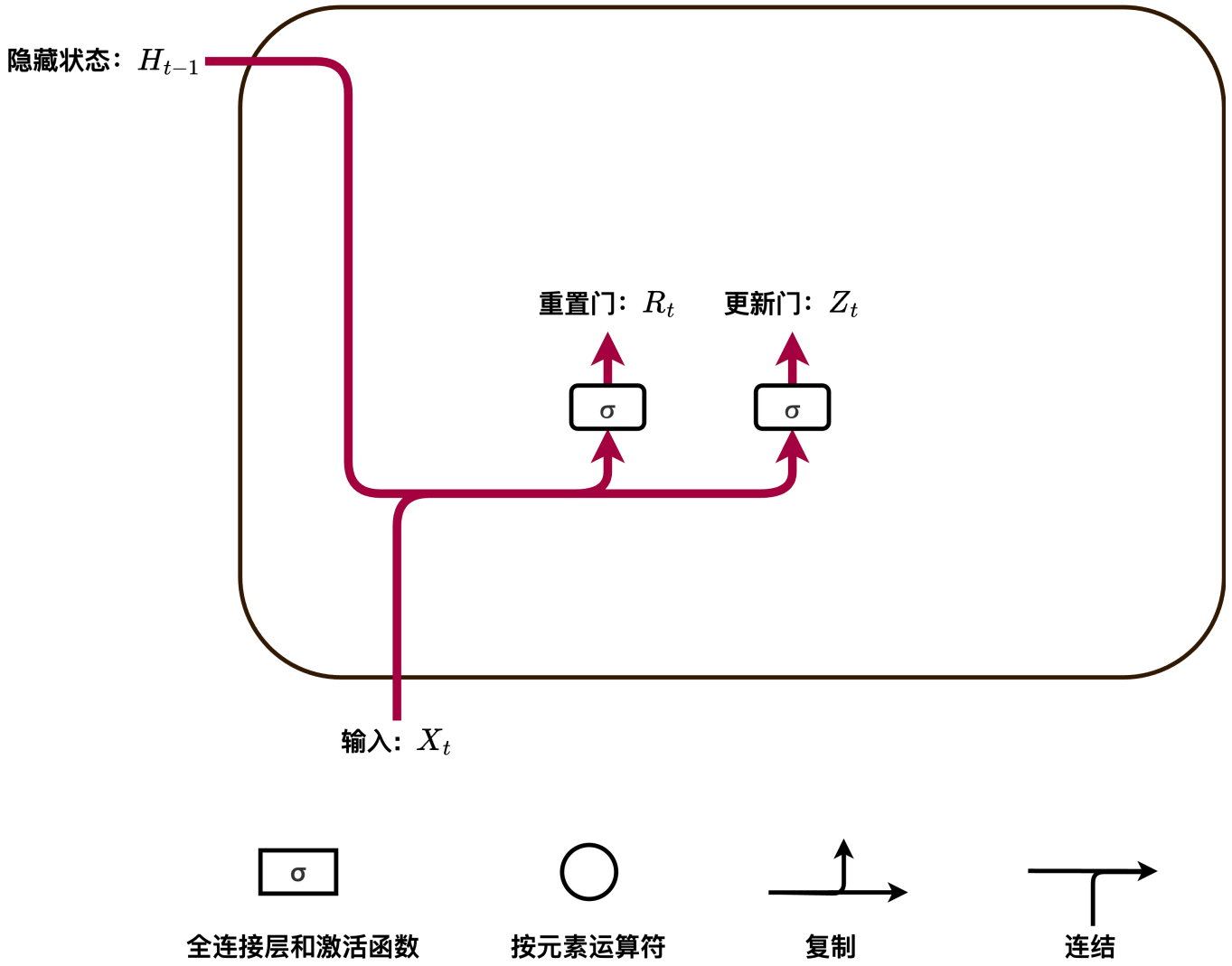




GRU

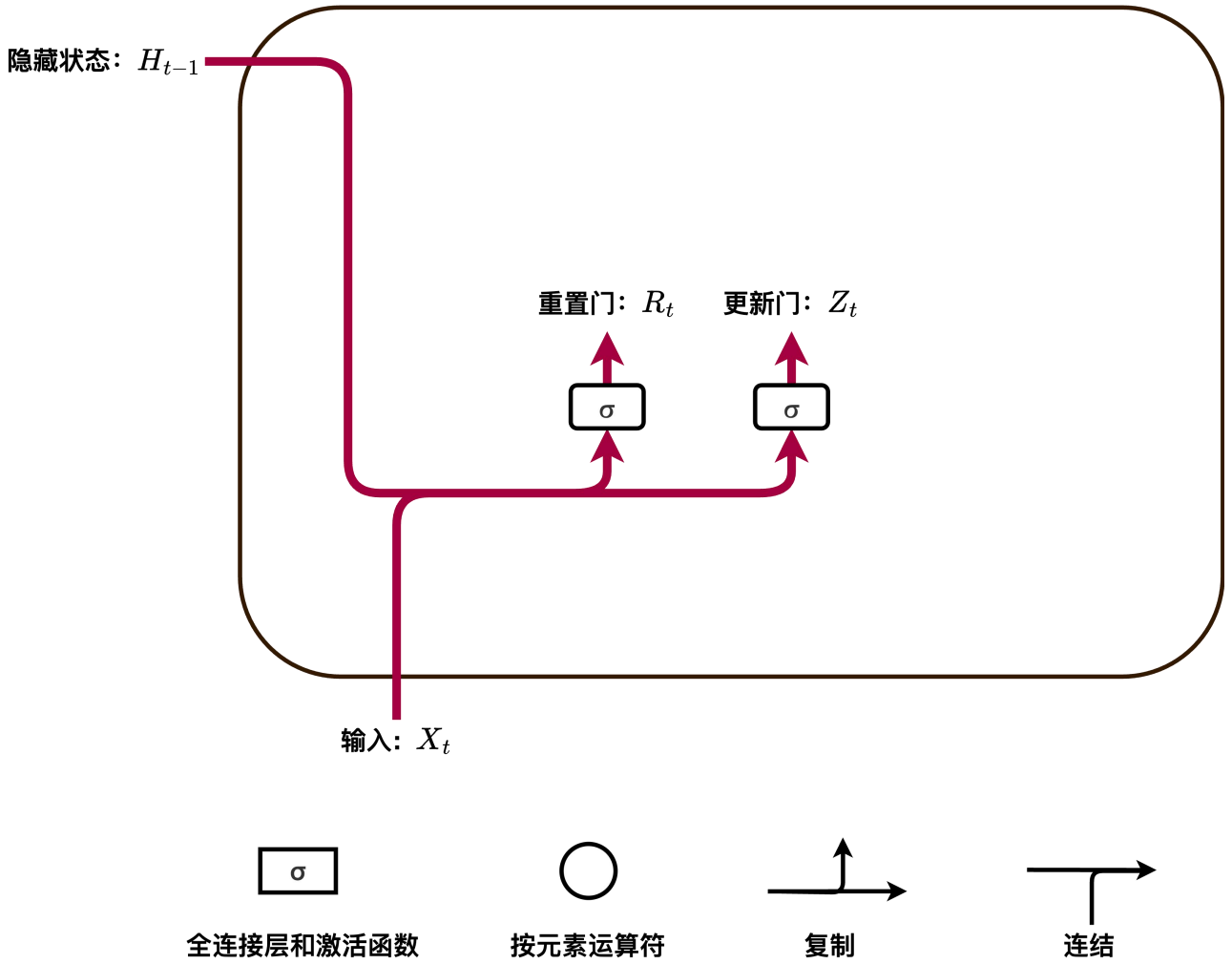
重置门和更新门

Title 重置门和更新门



- 门控循环单元（gated recurrent unit，GRU）是一种常见的门控循环神经网络；
- 通过引入重置门（reset gate）和更新门（update gate）的概念，从而修改了循环神经网络中隐藏状态的计算方式。

Title 重置门和更新门

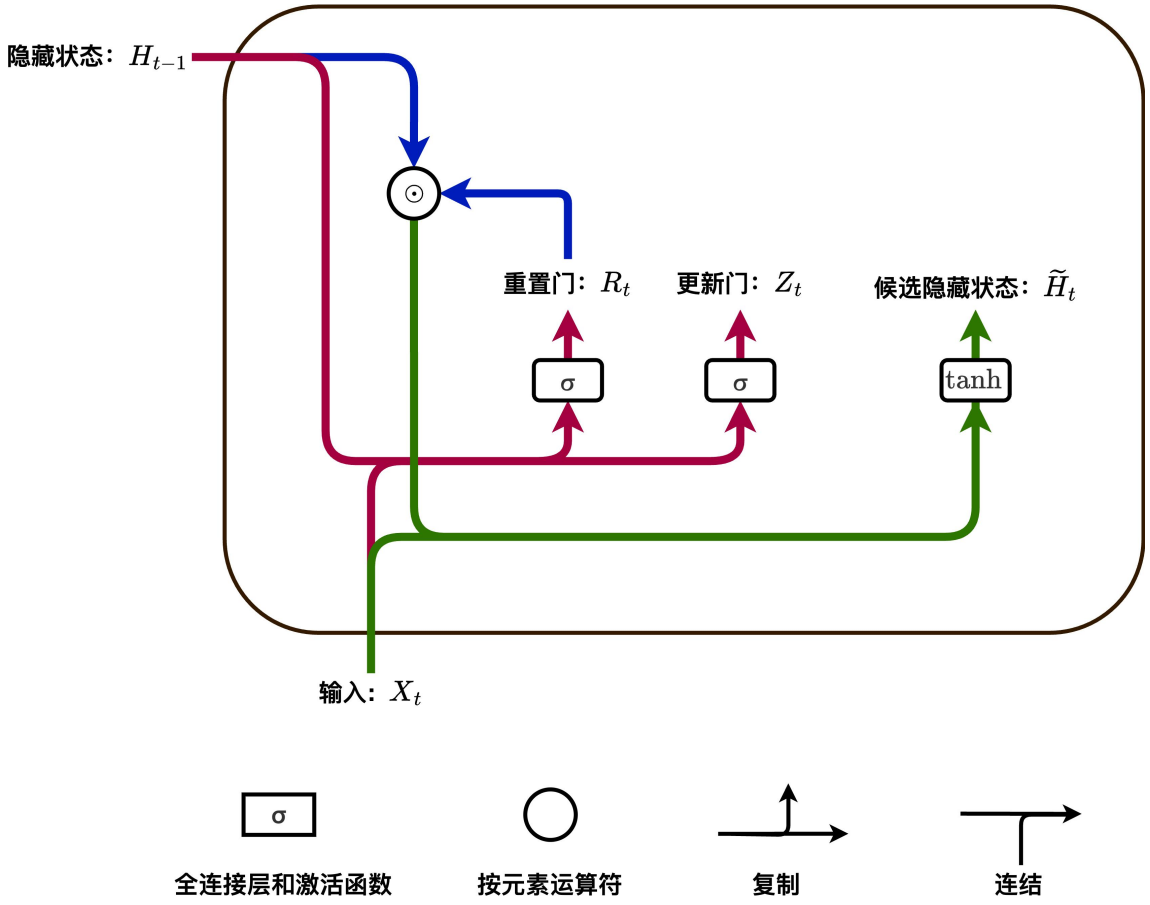


- in：
  - 当前时间步输入X_t
  - 上一时间步H_{t-1}
- deal：$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r)$
- out：$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$
  - 激活函数为sigmoid函数的全连接层计算得到

GRU

候选隐藏状态

Title 候选隐藏状态



$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$$

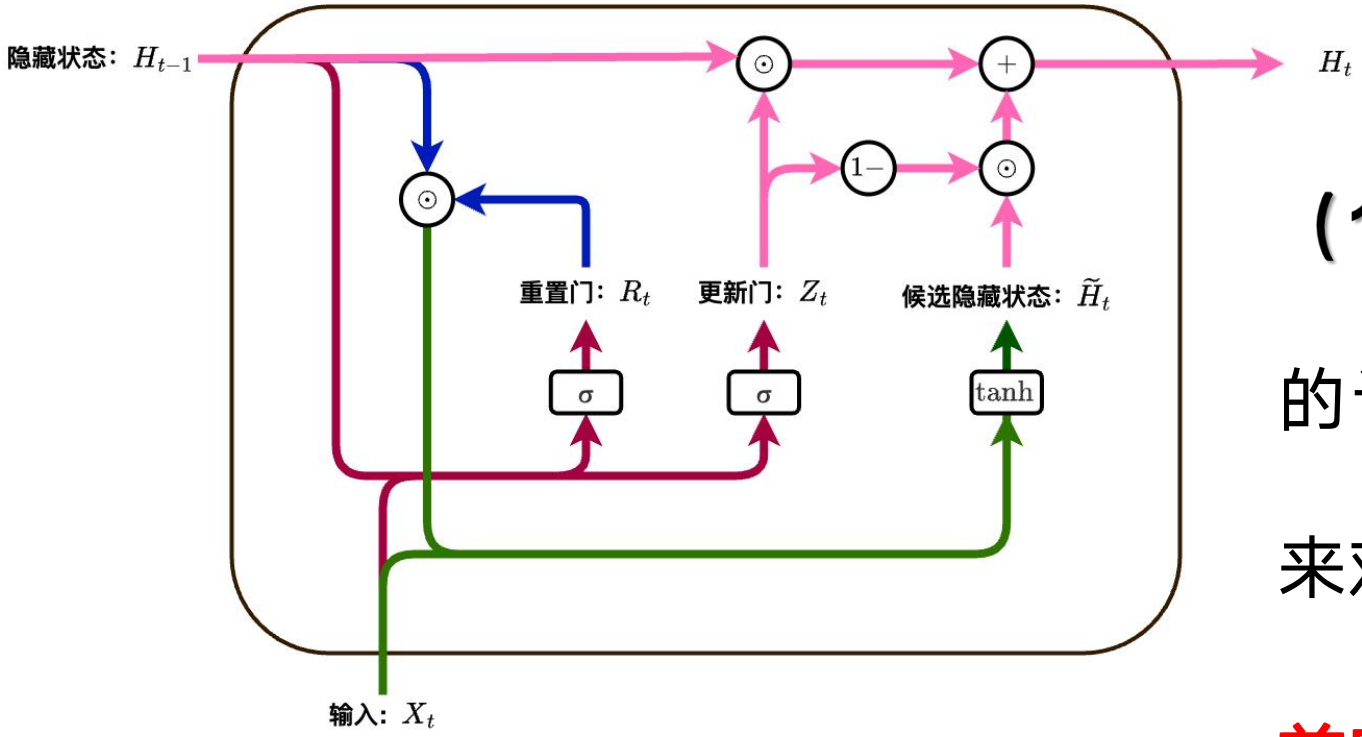
- 当前时间步重置门的输出与上一时间步隐藏状态按元素乘法（符号为⊙_x000C_）。
  - 如果重置门中元素值接近 0，意味着重置对应隐藏状态元素为 0，即丢弃上一时间步的隐藏状态；
  - 如果元素值接近1，那么表示保留上一时间步的隐藏状态。

- 然后，将按元素乘法的结果与当前时间步的输入连结，再通过含激活函数 tanh

隐藏状态

Title 隐藏状态

GRU



$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$

（1）时间步t的隐藏状态$H_t \in R^{n \times h}$

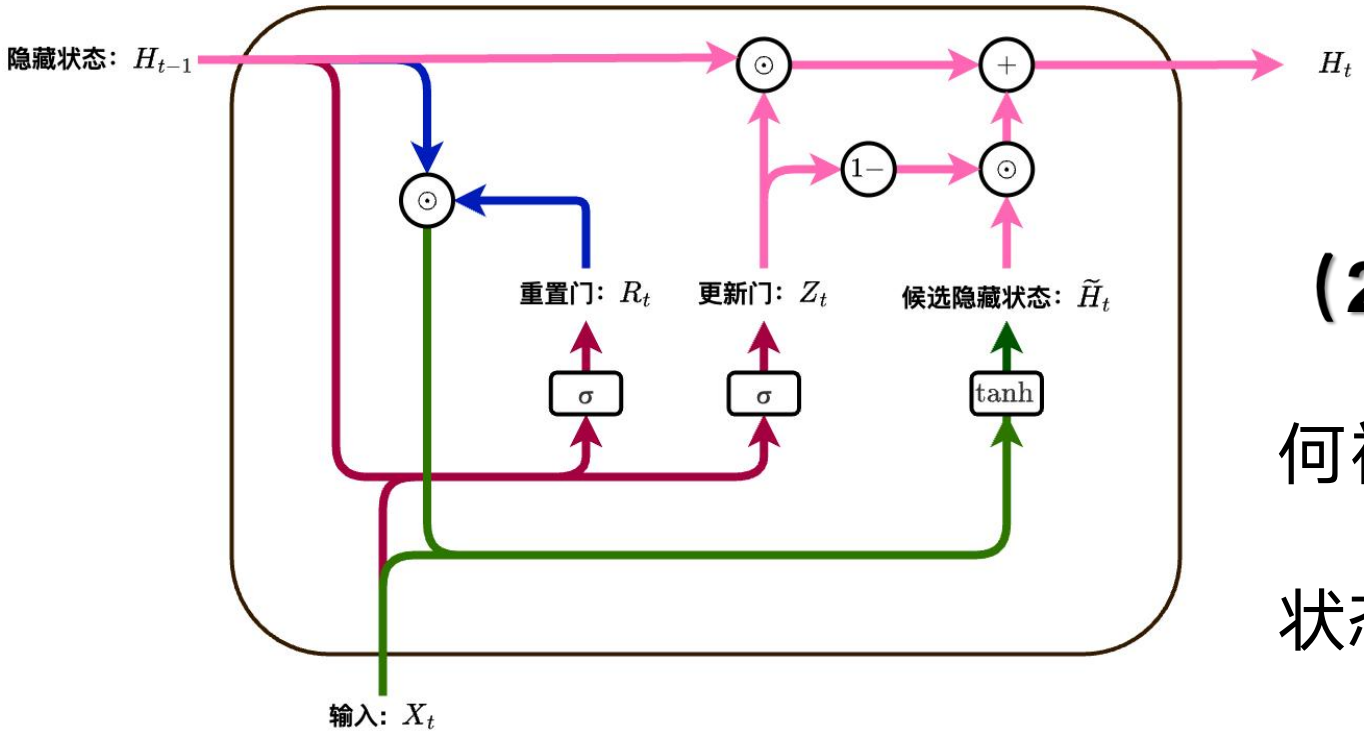
的计算使用当前时间步的更新门 Z_t

来对上一时间步的隐藏状态H_{t-1}和当

前时间步的候选隐藏状态 $\tilde{H}_t$ 做组合

Title 隐藏状态

GRU

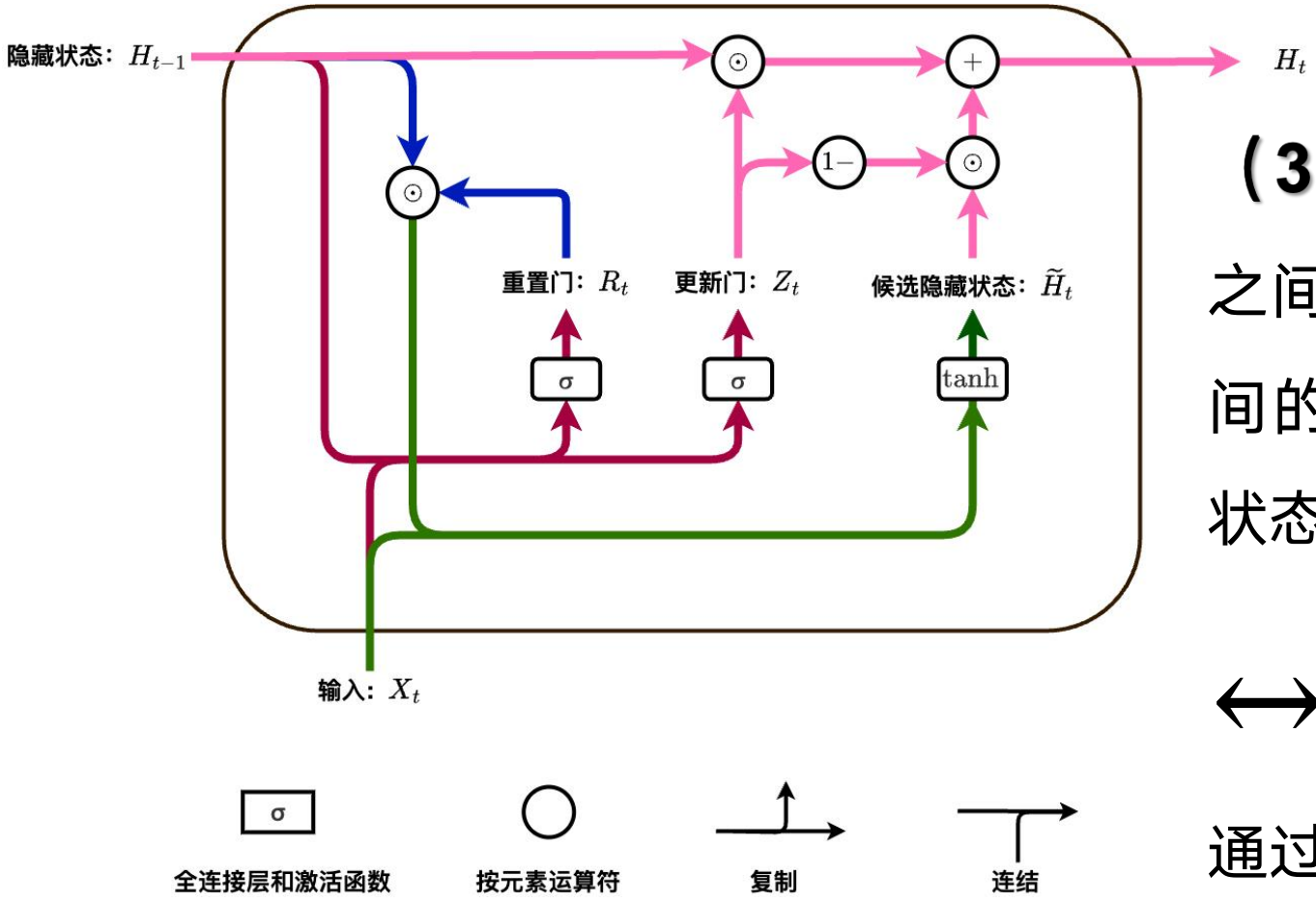


（2）更新门可以控制隐藏状态应该如何被包含当前时间步信息的候选隐藏状态所更新

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$

Title 隐藏状态

GRU



$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$

（3）假设更新门在时间步 t 到 t’(t’ ≤ t) 之间一直近似 1。那么，在时间步 t'到 t 之间的输入信息几乎没有流入时间步t的隐藏状态H_t

↔相当于较早时刻的隐藏状态 $H_{t'-1}$一直通过时间保存并传递至当前时间步t

Title 隐藏状态

（解释这句话）假设更新门在时间步 t 到 t’(t’ ≤ t) 之间一直近似 1。那么，在时间步 t'到 t 之间的输入信息几乎没有流入时间步t的隐藏状态H_t。

① 更新门控制着过去记忆的保留程度以及新信息的输入。当更新门接近1时，意味着模型更倾向于保留过去的记忆而忽略新的输入信息。这意味着在时间步 t 到 t’ （ t’ ≤ t）之间，更新门一直很接近1，这表示模型在这段时间内更加注重保留过去的记忆而忽略了新的输入信息。

Title 隐藏状态

（解释这句话）假设更新门在时间步 t 到 t’(t’ ≤ t) 之间一直近似 1。那么，在时间步 t'到 t 之间的输入信息几乎没有流入时间步t的隐藏状态H_t。

② 那么，那么，在时间步 t'到 t 之间的输入信息几乎没有流入时间步t的隐藏状态H_t。这句话的意思是，在这段时间内，由于更新门几乎保持关闭状态，导致在时间步 t 时刻的隐藏状态H_t几乎没有受到时间步t' 到 t 之间的输入信息的影响。因此，时间步 t 的隐藏状态主要由之前的记忆决定，而不是最近的输入信息。

谢　谢！