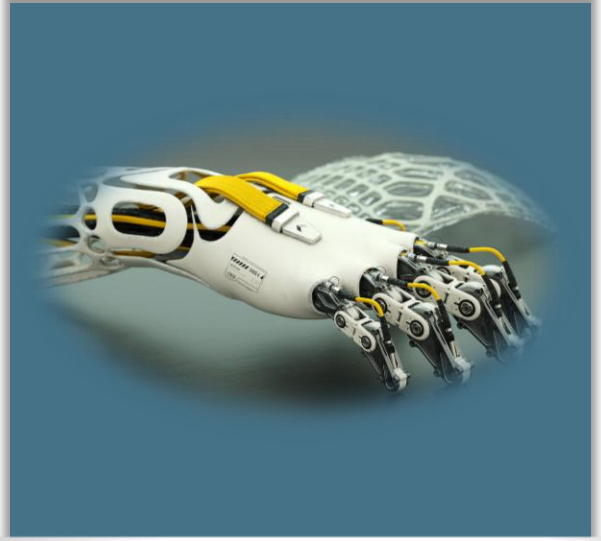
机器学习 第六讲

授课人：王闻博

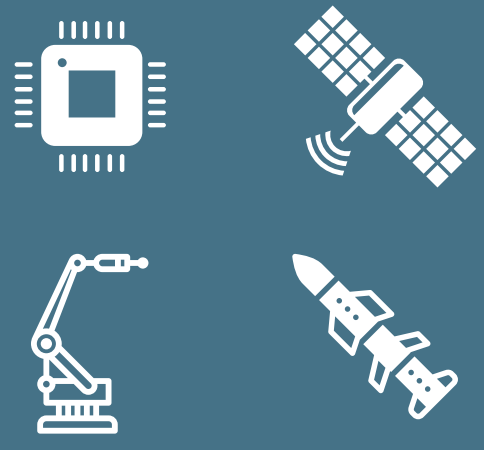
Email: wenbo_wang@kust.edu.cn

昆明理工大学 机电工程学院

2026年04月10-17日



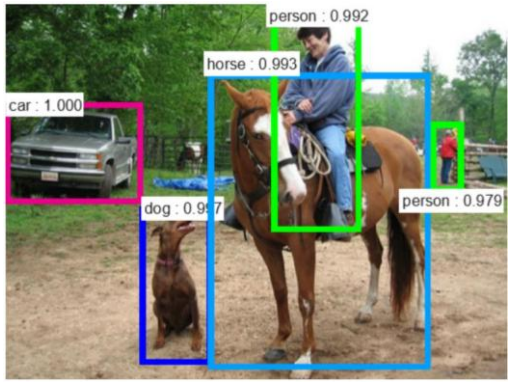
卷积神经网络

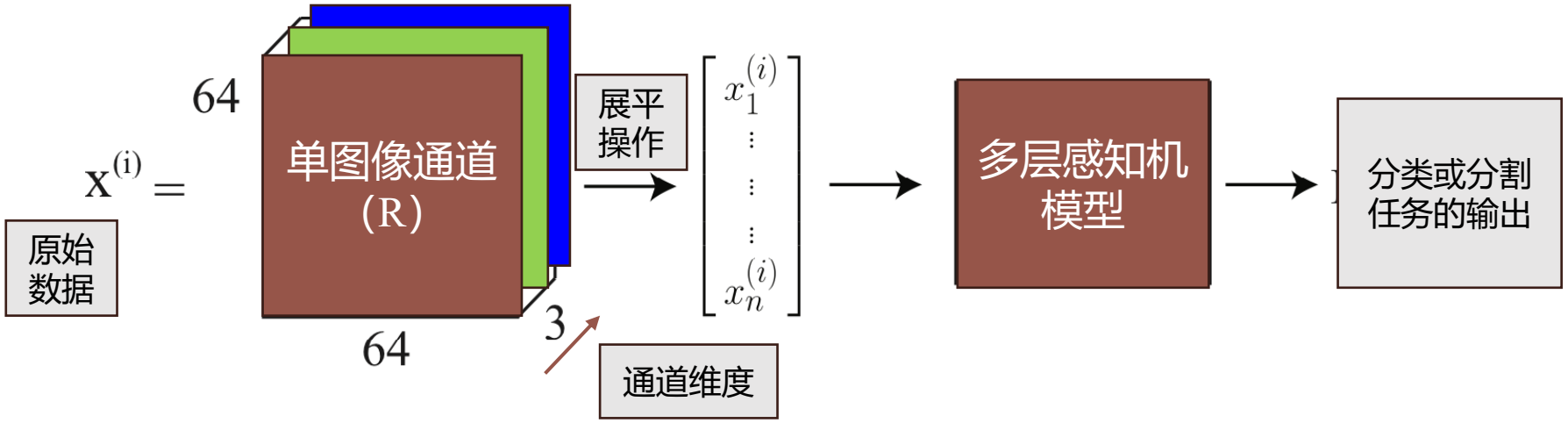


1. 从多层感知机到卷积神经网络
2. 卷积神经网络的基本组成和计算机制
3. 卷积神经网络的反向传播
4. 以卷积网络为骨干的早期深度学习模型概览
5. 卷积神经网络应用于不同任务

前馈神经网络（多层感知机）的局限

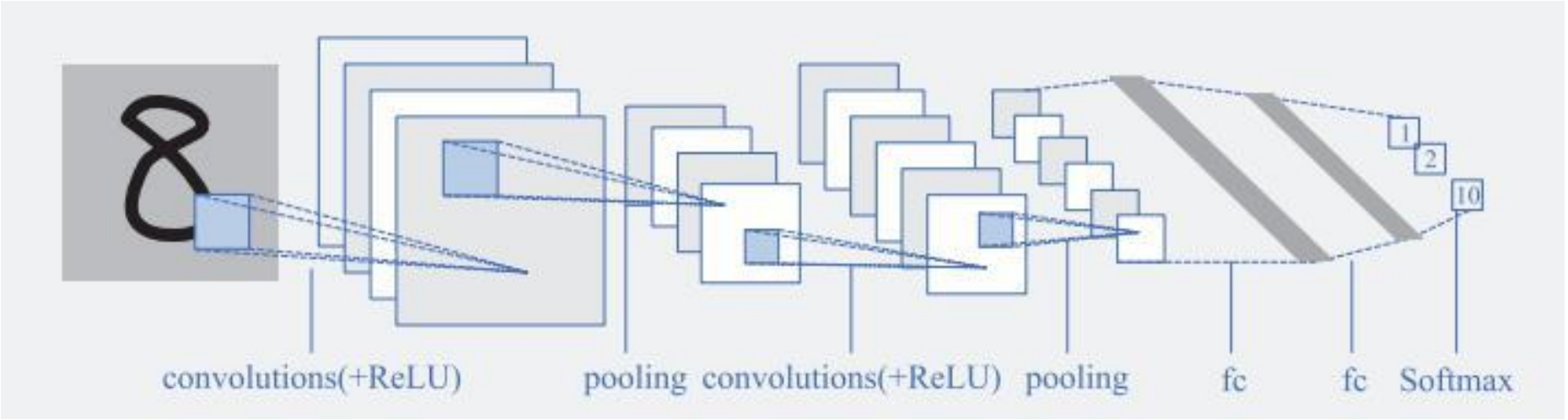
- **图像识别/分类领域的信号处理问题**
  - 任务1：识别右图1（三通道RGB图像）中的不同物体（人、马、狗、车辆等）；
  - 任务2：分割右图2 （三通道RGB图像）中的不同物体；
  - 基于前馈神经网络的设计：
    - 64x64x3=12288个输入层参数，引入多个隐藏层面临参数爆炸问题；
    - 多层感知机模型不善于处理通道间和通道内局部区域图像特征的相关问题；





卷积神经网络（Convolutional Neural Network，CNN）初窥

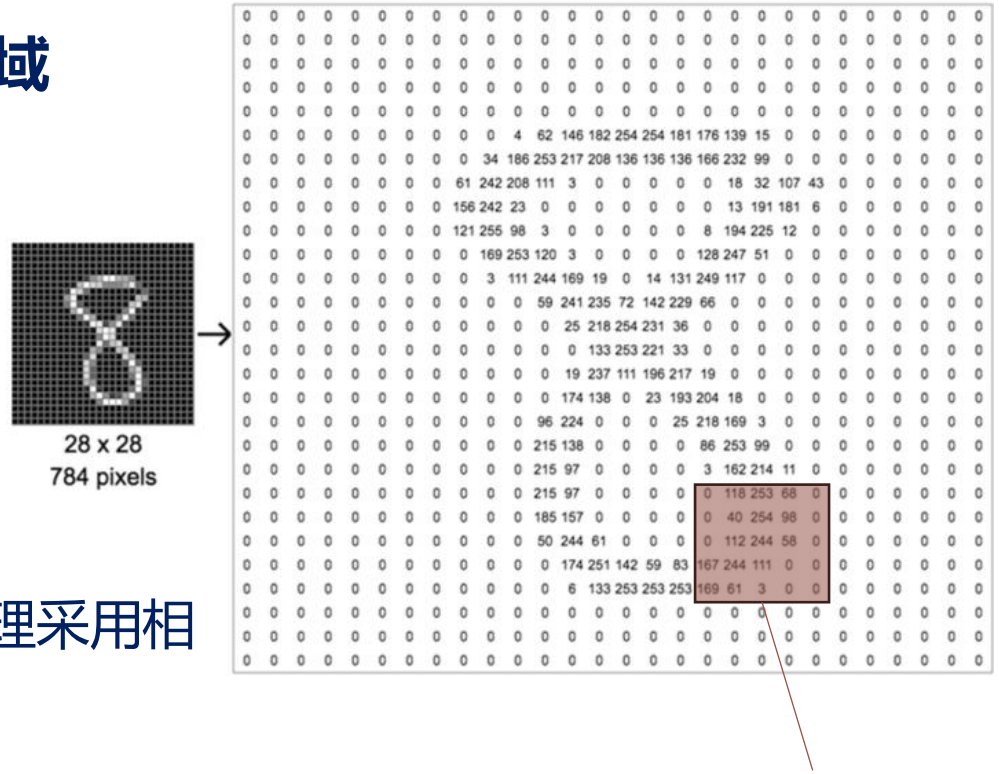
- **计算输入图像的特征图（Feature Map）：特征层次**
  - 深度卷积神经网络通过逐层组合的方式把低级特征组合成高级特征以构建“**特征层次**”；
  - 特征图中的每个特征层的每个元素都来自输入图像（或前一层）的一些局部像素（或特征）；



  - 卷积神经网络的典型信号处理层次
    - 卷积层；
    - 激活函数；
    - 池化层；
    - 全连接层；

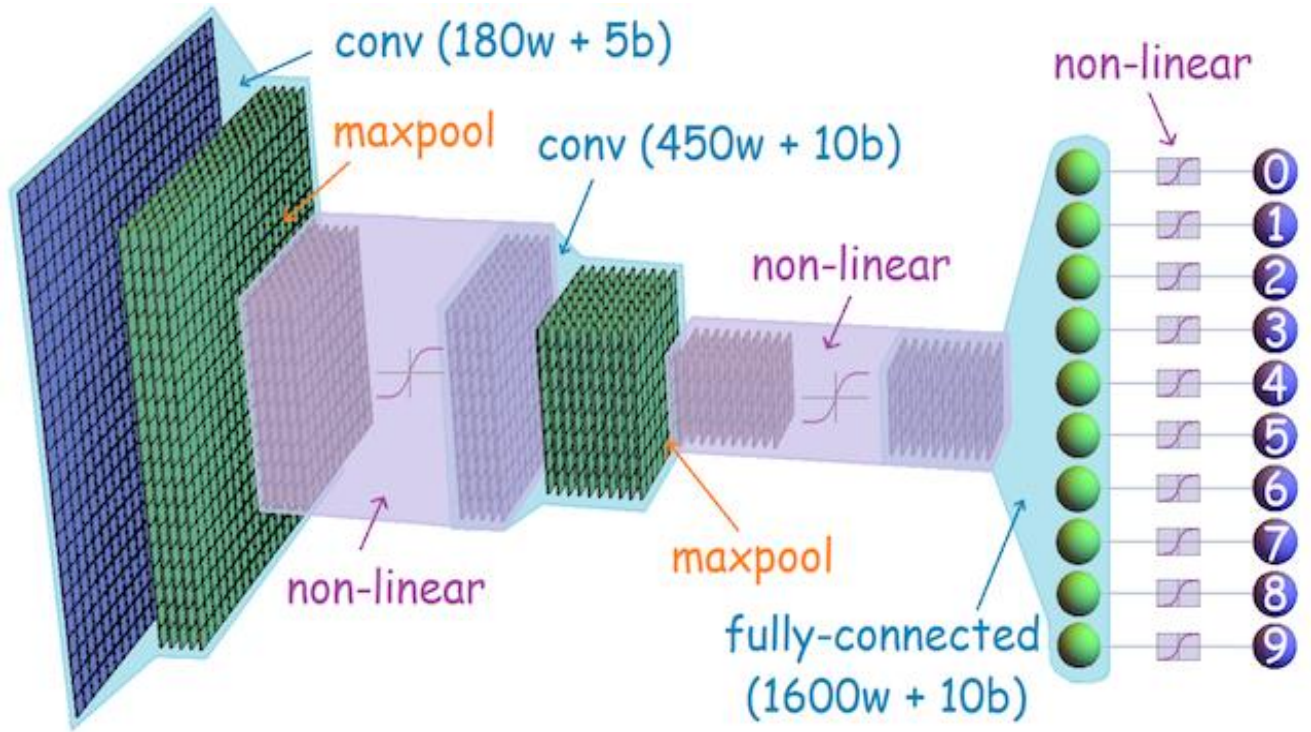
CNN网络的构建思想

- **局部感受野（Local Reception Field）：局部像素区域**
- **两个重要设计思路**
  - **（a）稀疏连接**：特征映射中的单个元素只连接到局部像素区域
    - 比较：多层感知机连接整个输入（全连接）。
  - **（b）参数共享**：对于输入图像的不同区域的信息处理采用相同的权重。
- **设计特性**：用卷积层替代全连接层能够显著减少网络中的权重参数的数量
- **此种设计的可行性分析**：图像数据中，距离相近的像素一般比距离较远的像素更为相关。

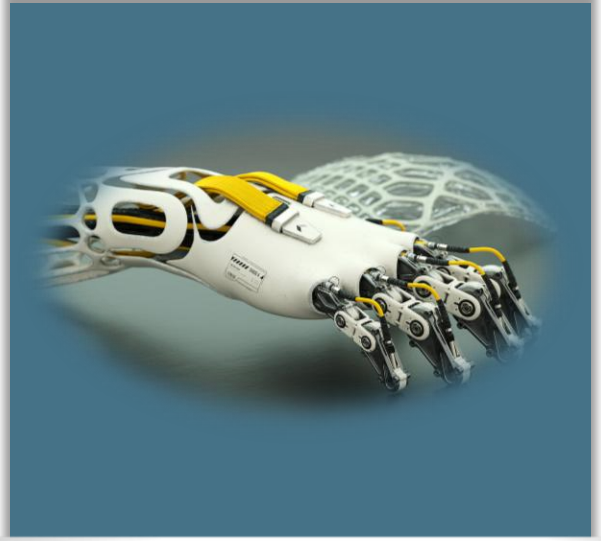


一个完整的CNN网络

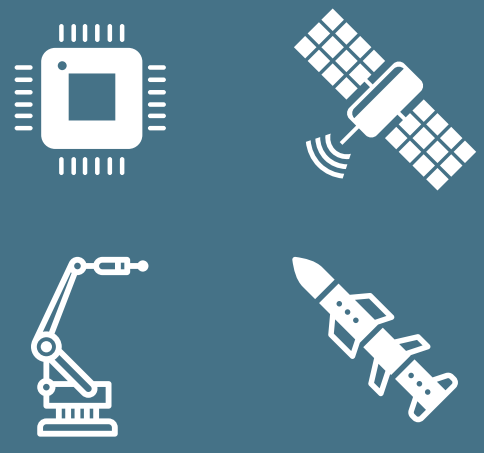
一个典型的卷积神经网络



图片来源：Stanford课程cs229-deep learning



卷积神经网络

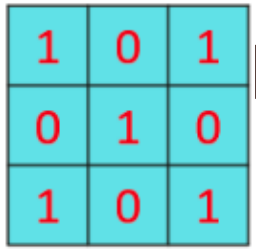


1. 从多层感知机到卷积神经网络
2. 卷积神经网络的基本组成和计算机制
3. 卷积神经网络的反向传播
4. 以卷积网络为骨干的早期深度学习模型概览
5. 卷积神经网络应用于不同任务

基本组件：卷积

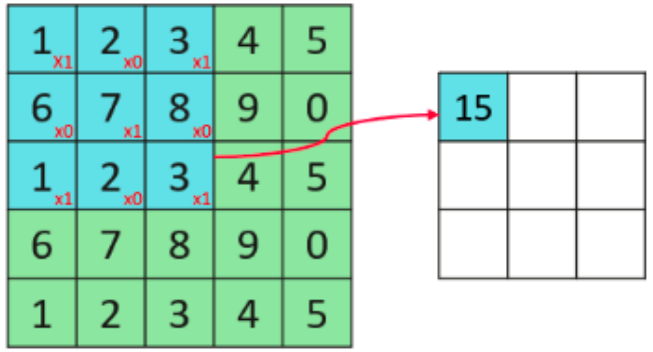
- **定义**
  - 通过一个特定大小的矩阵与上层感受野区域矩阵做点积运算从而得到下层神经元的特征输出，这个特定大小的矩阵叫做卷积核（convolution kernel），也称滤波器(convolution filter)。
  - 操作解析：在前向传播的时候，让每个卷积核都在输入数据的宽度和高度上作滑动窗口卷积（在卷积核的大小上做掩模运算），计算这个卷积核和输入数据对应每一个区域的矩阵内积，最终通过激活函数生成一个 2 维的激活映射输出。

输入矩阵

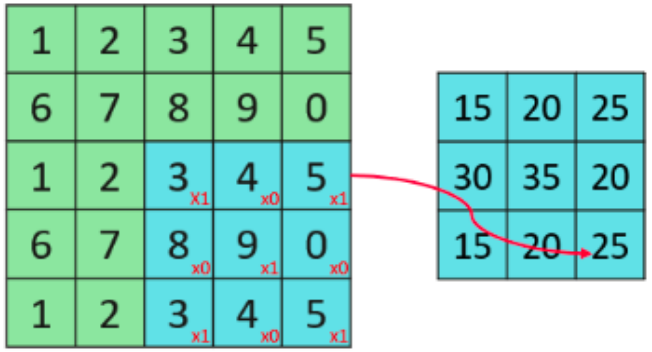


卷积核

输入矩阵与卷积核



(a) 第1次卷积操作及得到的卷积特征



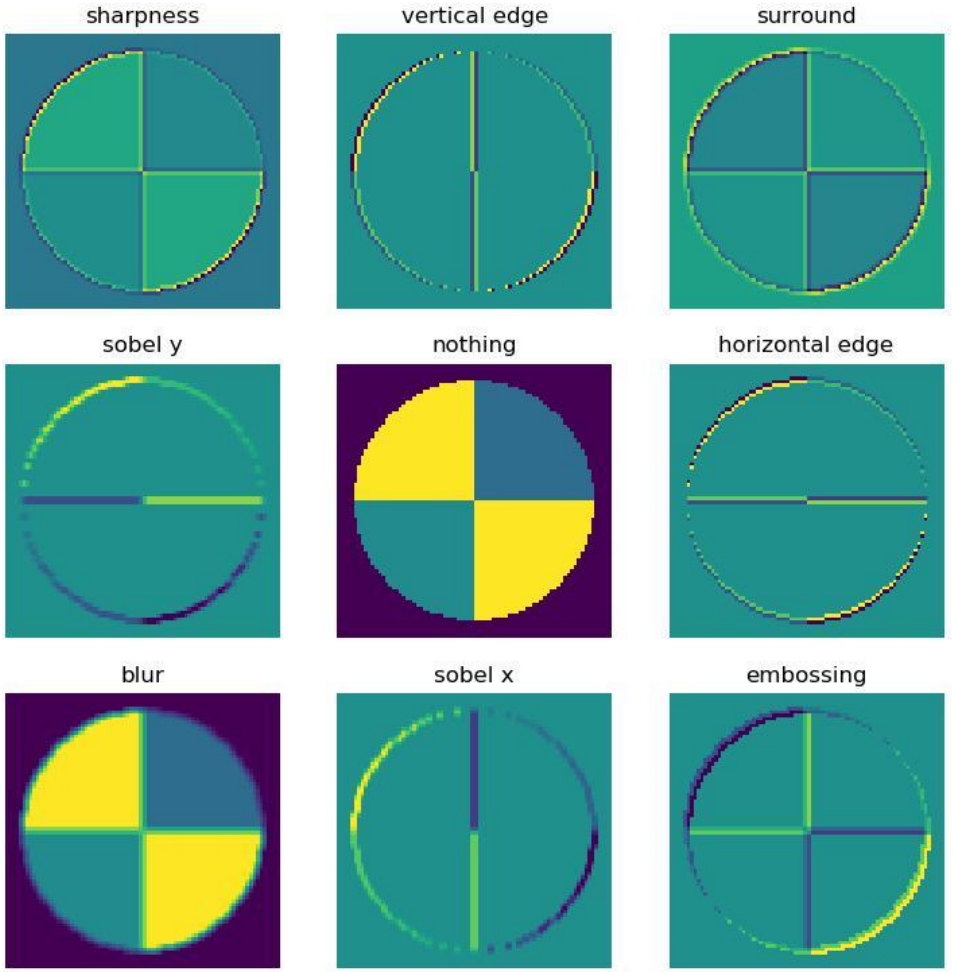
(b) 第9次卷积操作及得到的卷积特征

卷积操作示例

对比：传统图像处理中的滤波操作

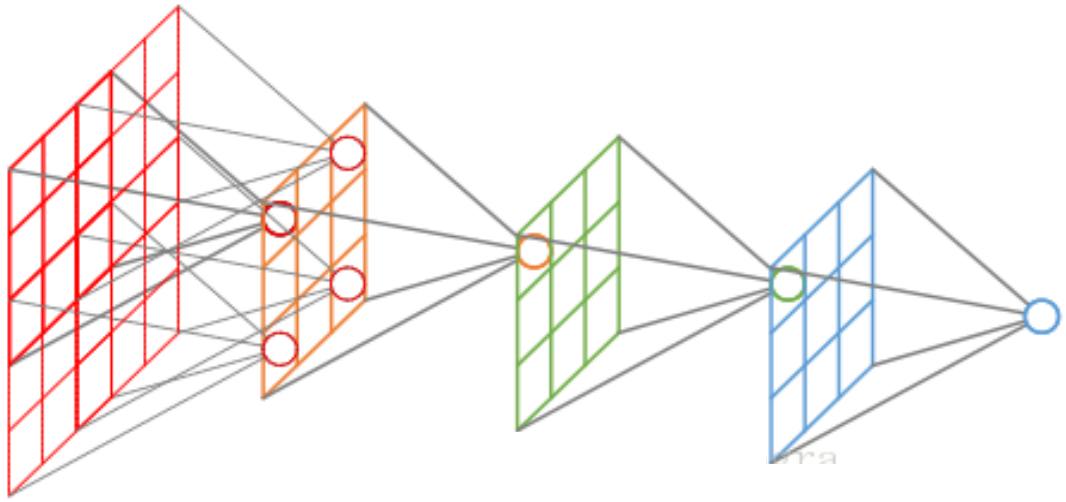
- 固定卷积核对应的滤波效果：锐化、边缘增强、周边增强或模糊等

	1	2	3
1	0,-1, 0 -1, 5,-1 0,-1, 0	0, 0, 0 -1, 2,-1 0, 0, 0	1, 1, 1 1,-9, 1 1, 1, 1
	sharpness	vertical edge	surround
2	-1,-2, -1 0, 0, 0 1, 2, 1	0, 0, 0 0, 1, 0 0, 0, 0	0,-1, 0 0, 2, 0 0,-1, 0
	sobel y	nothing	horizontal edge
3	0.11,0.11,0.11 0.11,0.11,0.11 0.11,0.11,0.11	-1, 0, 1 -2, 0, 2 -1, 0, 1	2, 0, 0 0,-1, 0 0, 0,-1
	blur	sobel x	embossing

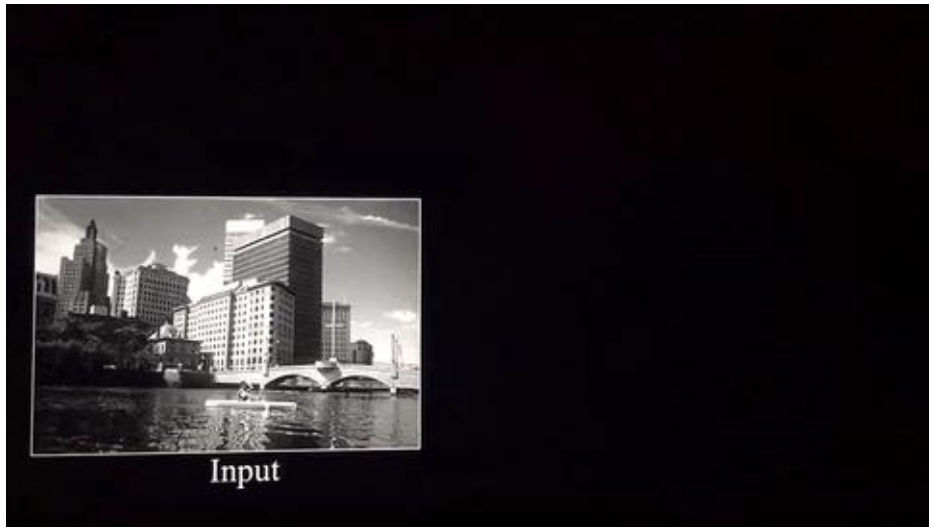


基本组件：感受野

- **定义**
  - CNN每一层输出的特征图(feature map)上每个像素点在输入特征图上映射的区域大小。或，局部连接的空间大小叫做神经元的感受野。
  - 在卷积神经网络中普遍使用卷积层和池化层，层与层之间均为局部连接，因此，单个的神经元无法对原始图像的所有信息进行感知。
  - 在输入图像中移动局部感受野，每移动一次，对应一个隐层的神经元，不断重复构成隐层所有神经元。



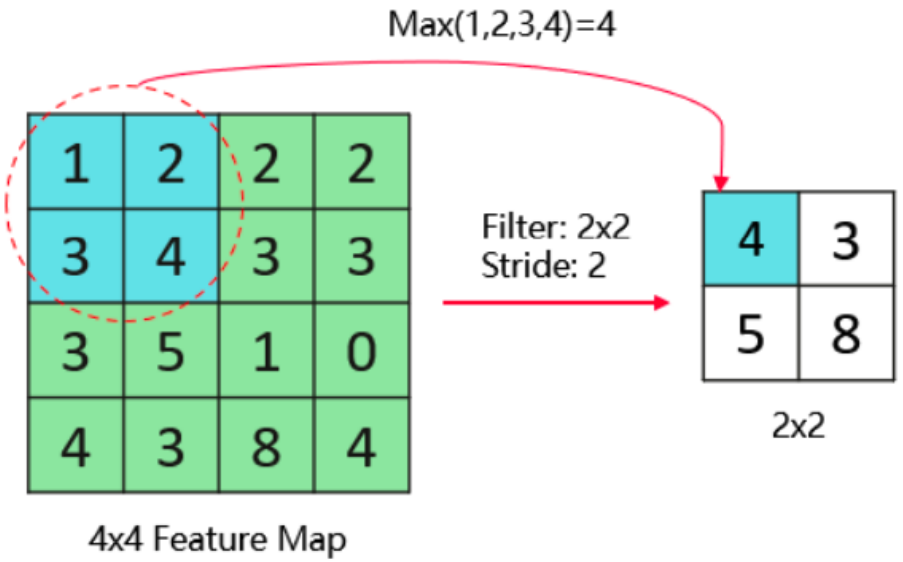
不同层的特征图上的一个特征点对应其前一层输入特征图的区域



不同卷积核在局部感受野上的掩模操作

基本组件：池化（Pooling）

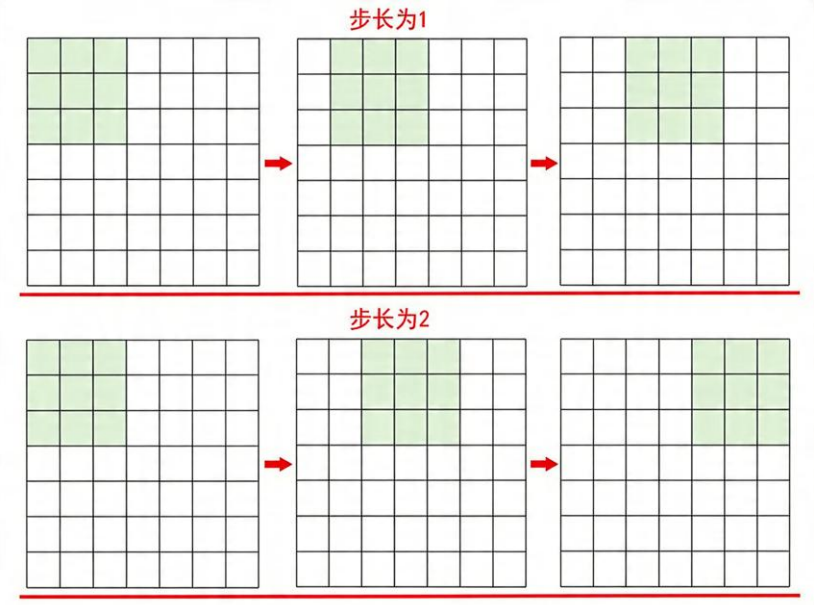
- **定义**
  - 为了只用一个像素点就能表达图像某个领域内整个区域的信息，可以采用类似于图像压缩的思想，对图像进行卷积之后，通过一个下采样（Downsampling）过程，来调整特征图的大小。
  - 从某个池化窗口内进行采样的规则主要有取最大值、取最小值和取平均值三种，所对应的池化操作分别称之为最大池化、最小池化和均值池化。

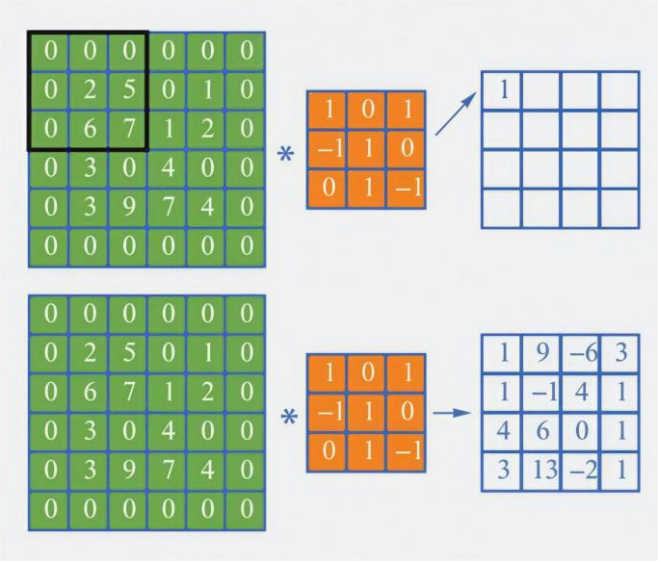


通过最大池化，完成特征图压缩，
特征图规模减少至1/4

卷积/池化滑动窗口的重要超参数：步长和零填充

- **步长**
  - 步长是卷积核在输入数据（如图像或特征图）上滑动的间隔距离。例如，步长为1时，卷积核逐像素移动；步长为2时，每次移动两个像素。
- **零填充**
  - 零填充是在输入数据周围添加若干层零值的操作，例如在图像边缘补零以扩展输入尺寸。
  - 作用：
    - **保持输入输出尺寸一致**（Same卷积）：通过填充使输出特征图尺寸与输入相同。、；
    - **防止边缘信息丢失**：确保卷积核能覆盖输入边缘区域，避免边缘像素仅参与一次计算；
    - **灵活设计网络结构**：通过调整填充量，控制特征图尺寸的缩减速度。





基本特性：输出特征矩阵的维度

- **对一个卷积核而言，其输出特征矩阵的特征维度由卷积核大小、步长和填充值（圈数）决定**
  - 假设数据矩阵大小为$M \times N$，卷积核大小为$m \times n$，填充的圈数为p，水平方向和竖直方向的步长分别为d_1, d_2，则输出特征矩阵的行数和列数分别为：

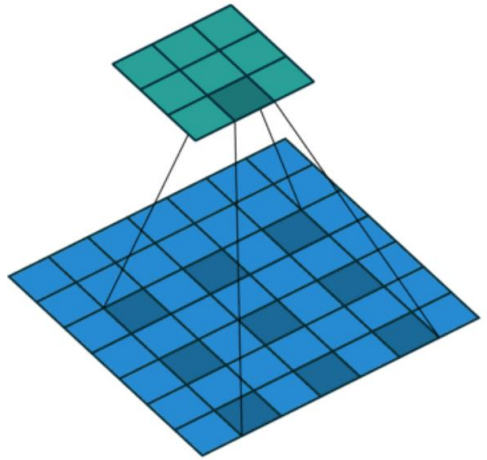
$$W = \lfloor \frac{M + 2p - m}{d_1} \rfloor + 1$$

$$H = \lfloor \frac{N + 2p - n}{d_2} \rfloor + 1$$

其中，$\lfloor \cdot \rfloor$表示向下取整。

说明：如果有扩张(空洞卷积，Dilation，如下图)操作，则

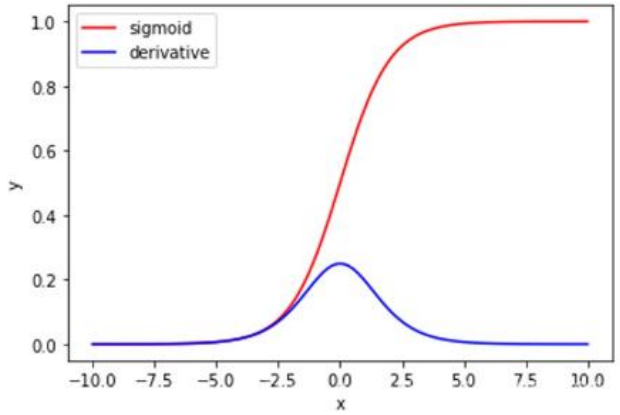
$$W = \lfloor \frac{M + 2p - \mathrm{d}(m-1) - 1}{d_1} \rfloor + 1$$



基本组件：激活函数

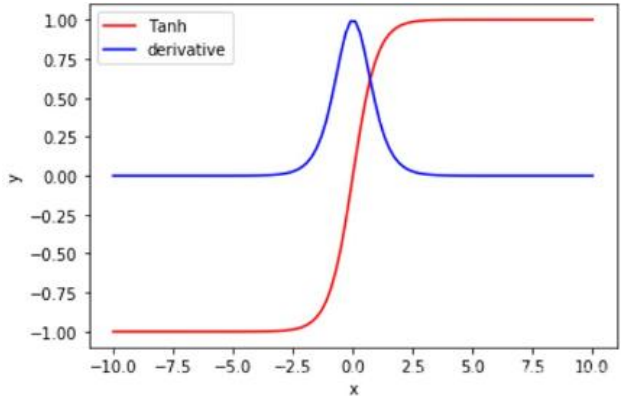
- **卷积运算后的特征图中的元素通过激活函数加入非线性因素，用以提高模型表达力**

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$



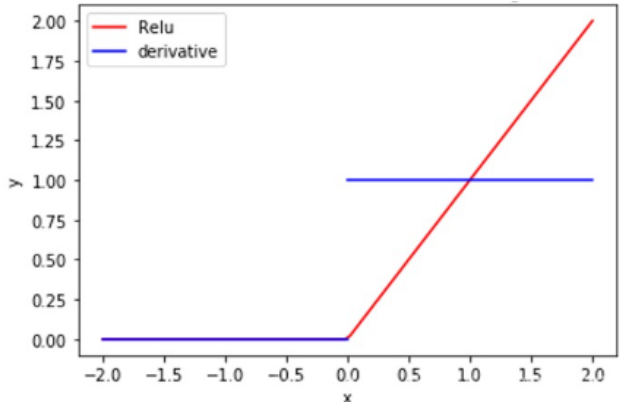
值域：(0, 1)
反向传播中，
容易就会出现梯度消失

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$



值域：(-1, 1)
减轻梯度消失问题

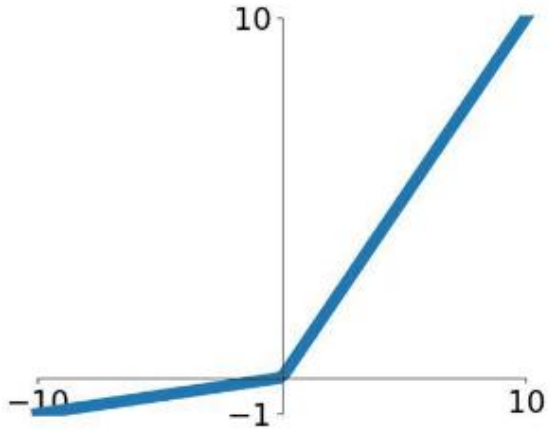
$$relu(x) = \max(0,x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$



值域：(0, x)
有助于随机梯度下降收敛

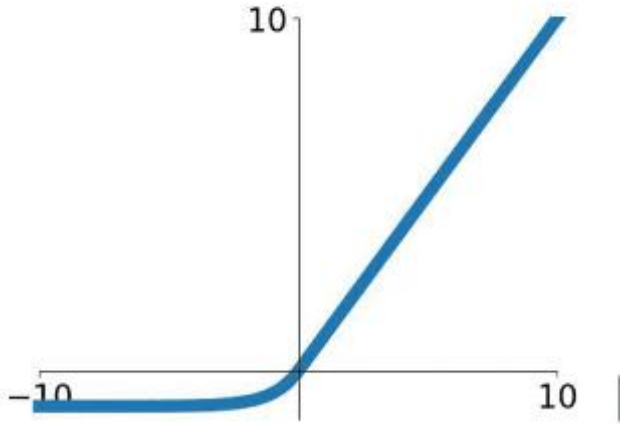
其他激活函数

(1) $\text{LeakyReLU}(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$



特点：解决了ReLU单元的问题，$x < 0$时给出一个很小的梯度值

(2) $\text{ELU}(x) = \begin{cases} x & ,x > 0 \\ \alpha(\exp(x) - 1), other \end{cases}$



特点：具有ReLU的优点，有负饱和问题，对噪声有较强的鲁棒性。

(3) $\text{maxout}(x) = \max\left(w_1^T x + b_1, w_2^T x + b_2\right)$

特点：对权重和数据的内积结果进行直接比较两个线性函数，可以线性操作，没有饱和现象

信号在卷积神经网络的前向传播

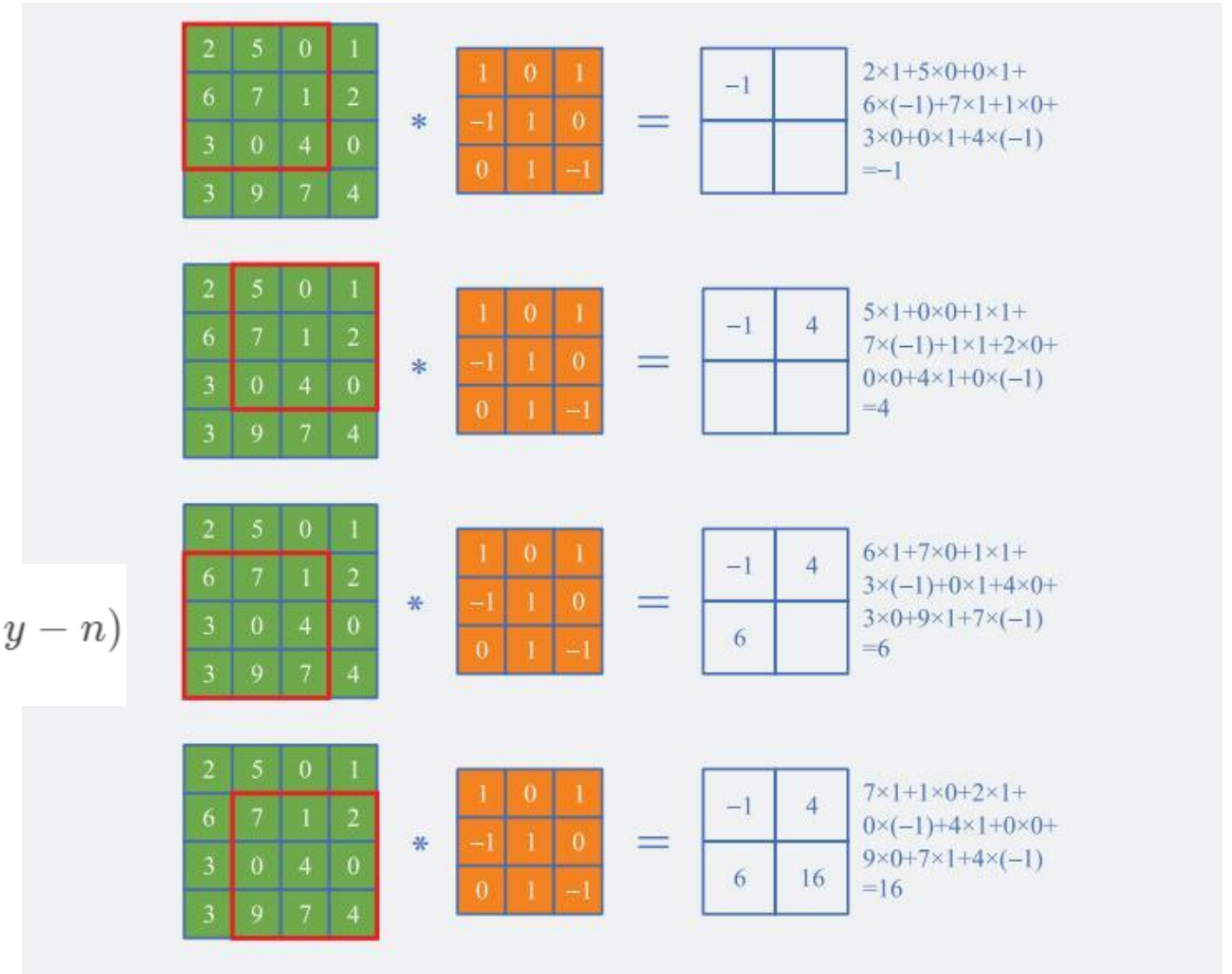
- **卷积的前向计算**

$$F(i,j)=\sum_{m=0}^{k-1}\sum_{n=0}^{k-1}I(i+m,j+n)\cdot K(m,n)$$

  - 注意：此操作实际上是互相关函数的计算，并不是传统意义上的二维信号的卷积，比较传统卷积：

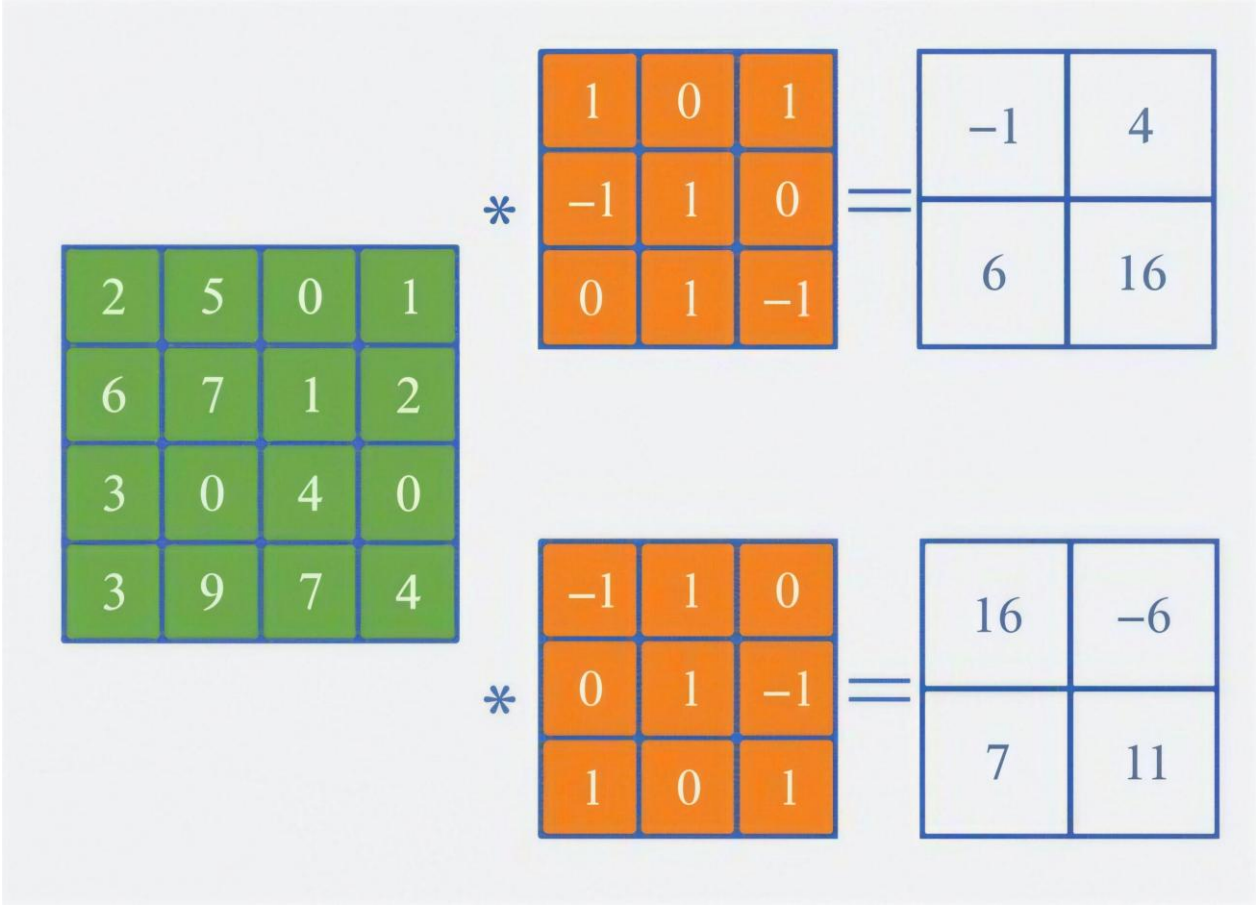
$$(f*g)(x,y)=\sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty}f(m,n)\cdot g(x-m,y-n)$$

  - 传统卷积：需先对卷积核进行水平和垂直翻转（即核旋转180度），再与输入信号逐点相乘求和。



卷积的前向计算：单入多出

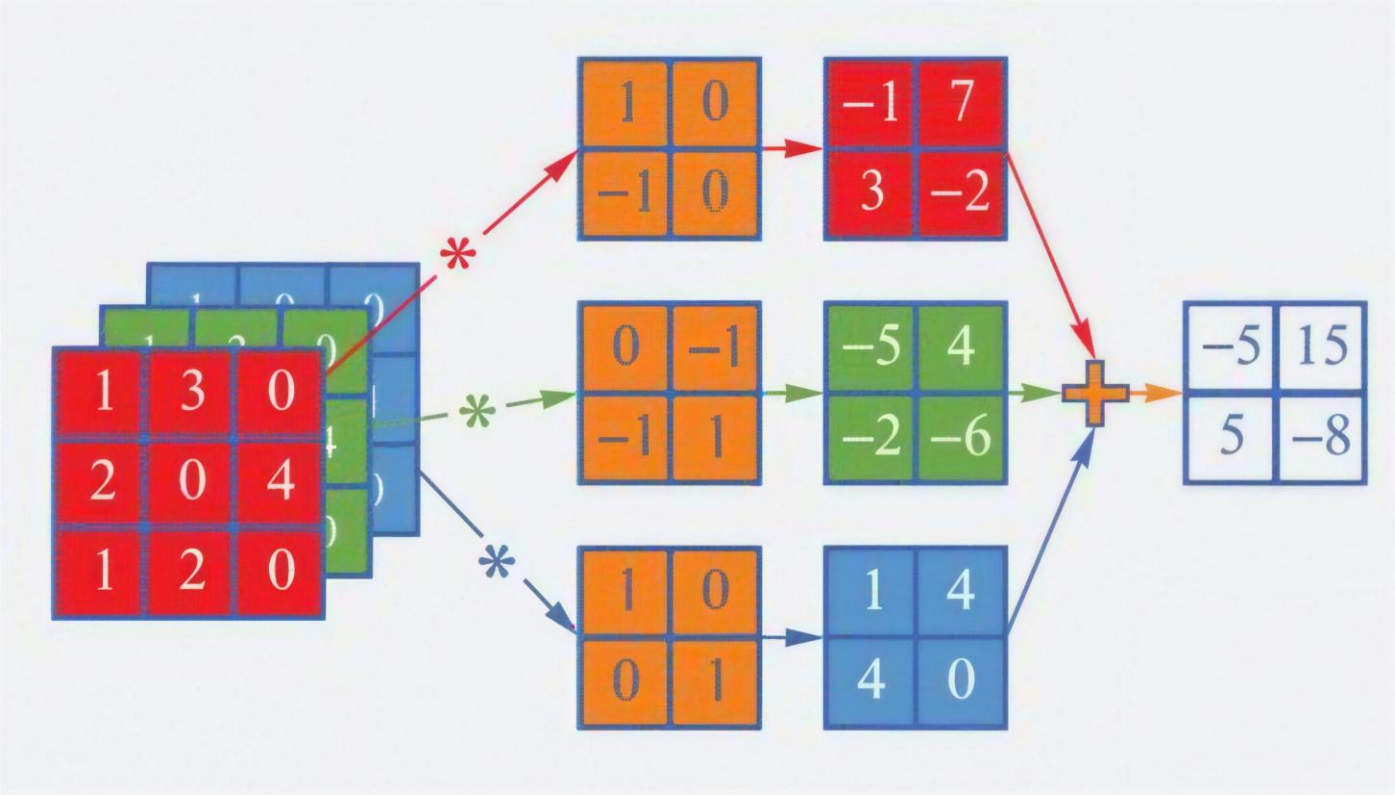
- **对单通道输入张量（单通道图像），可以用多个卷积核分别对其卷积，得到多个特征输出。**



左图：两个卷积核分别对输入进行操作

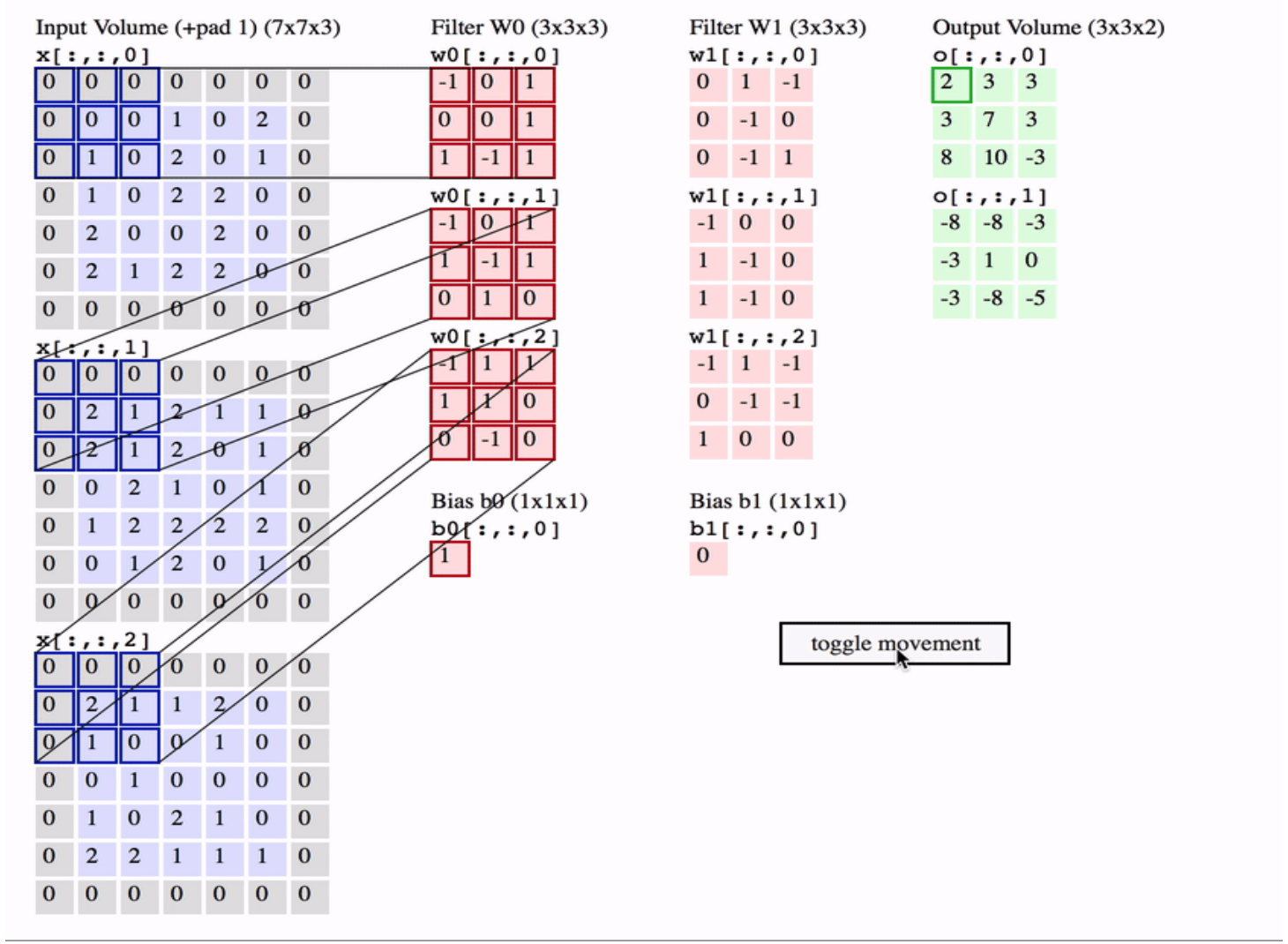
卷积的前向计算：多入单出

- 对多通道输入（如RGB三通道图像），每个通道共用一个卷积核操作，输出加在一起
- 或，三个通道各自通过同尺寸不同卷积和操作后，再进行不加权和操作，如下图：



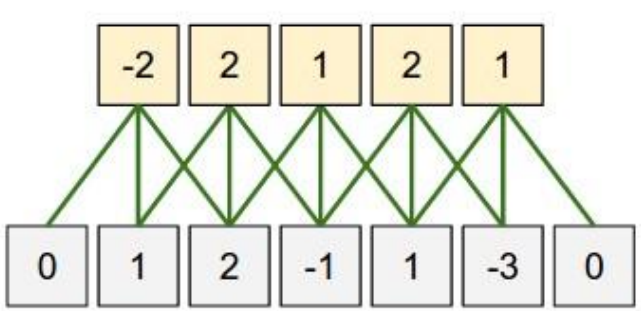
卷积的前向计算：多入多出的一般情况

- 假设有两组3×3×3的卷积核，则卷积运算如右图所示。

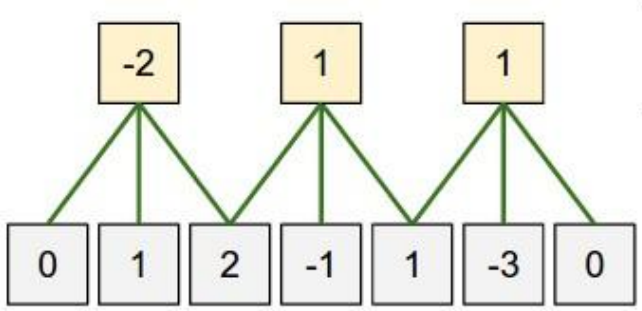


基本特性：参数共享

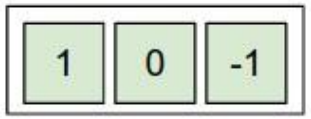
- 以一维卷积为例



卷积操作

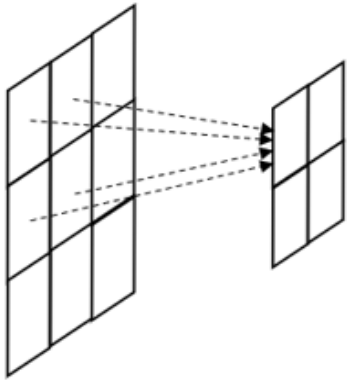


输入输出相关性（局部连接）

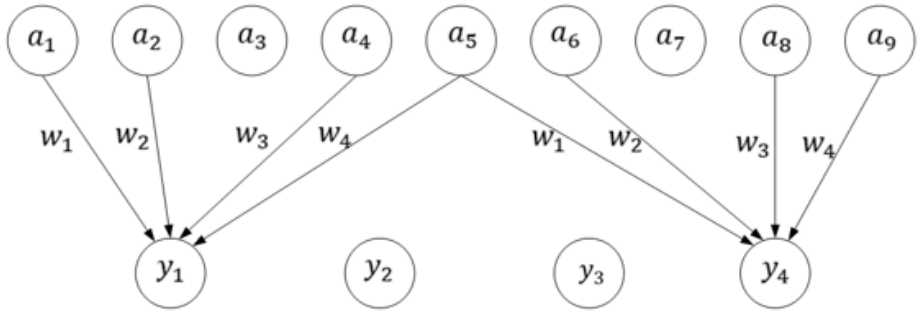


卷积核

- 二维卷积的参数共享：



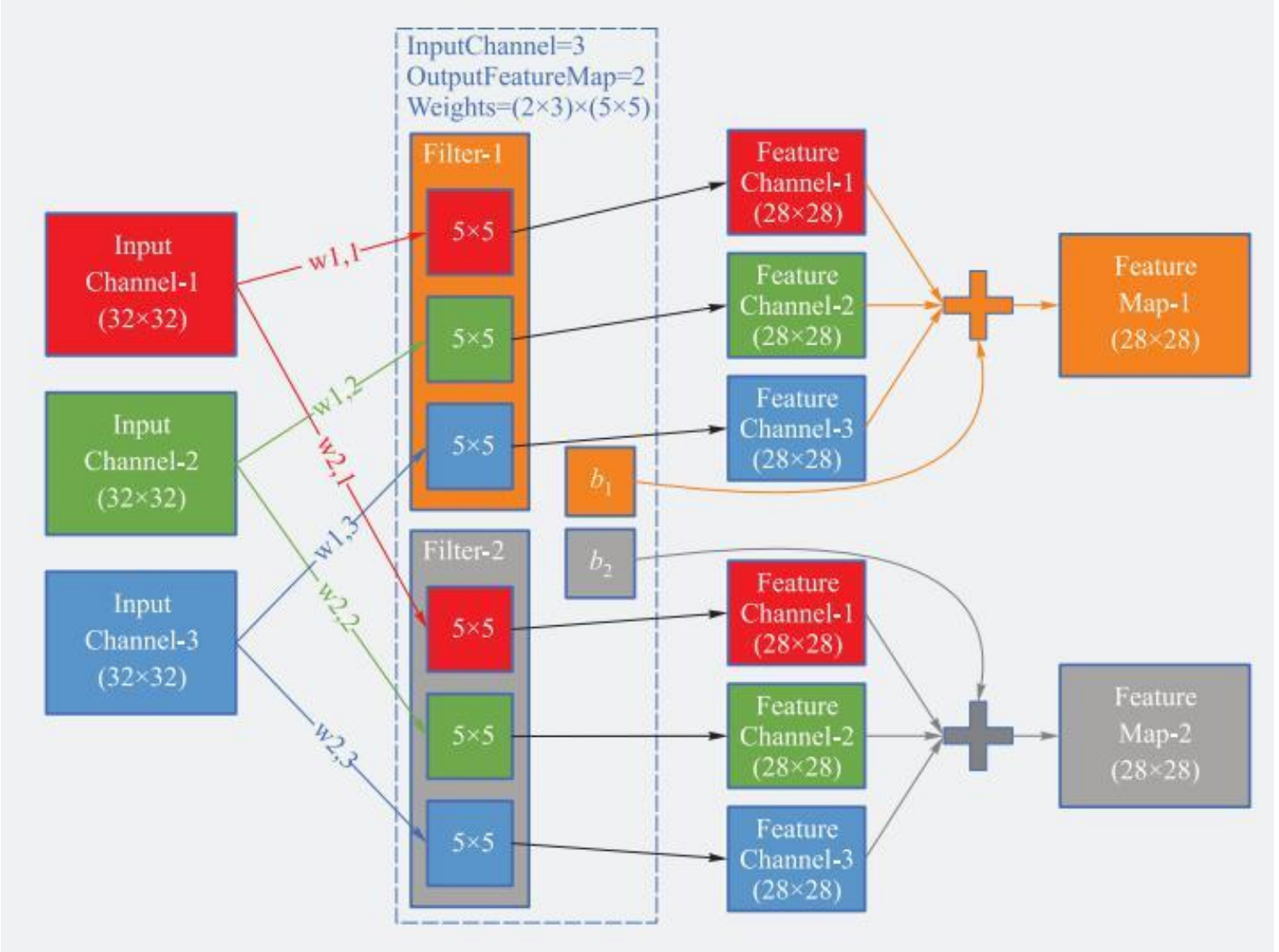
元素y_1相关性关系图



展开的元素y_1, y_4相关性关系图

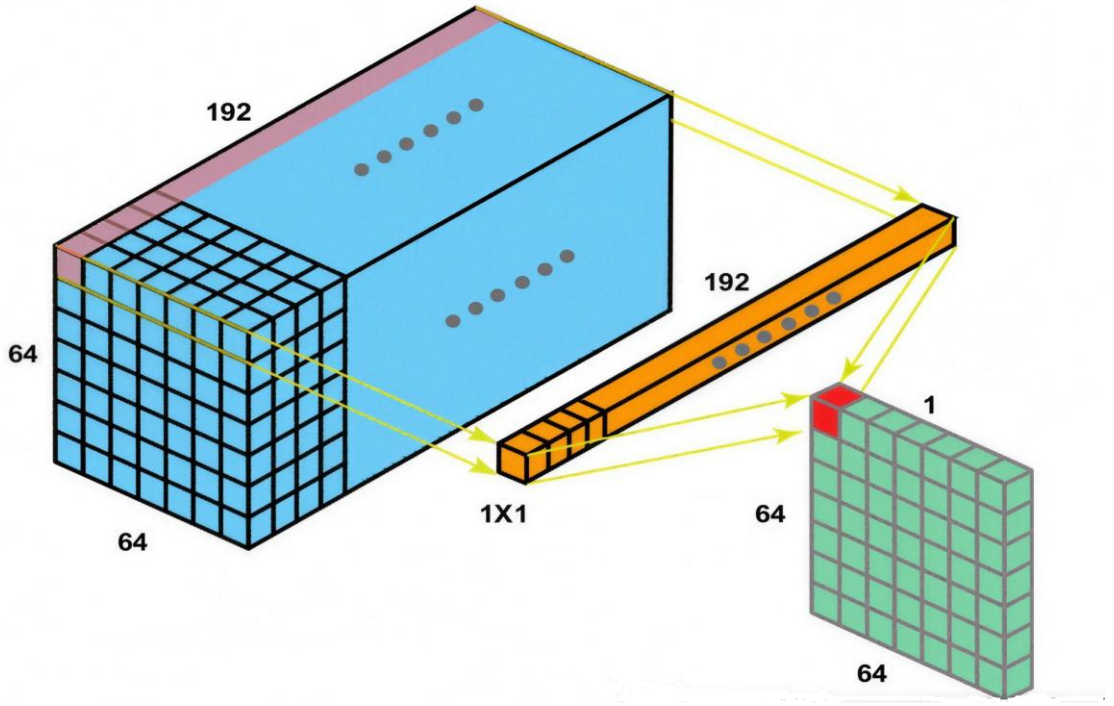
小总结：卷积的前向计算

- **卷积运算的主要元素**
  - 输入：Input（Width×Height×Channel）；
  - 卷积核Kernel组（含偏置项bias）；
  - 步长（Stride）；
  - 填充（Padding）；
  - 输出：Feature Map（Width×Height×Channel）。
- **右图**
  - 32×32×3的输入，在两组5×5×3的卷积核操作下，输出28×28×2的特征图（Feature Map）。



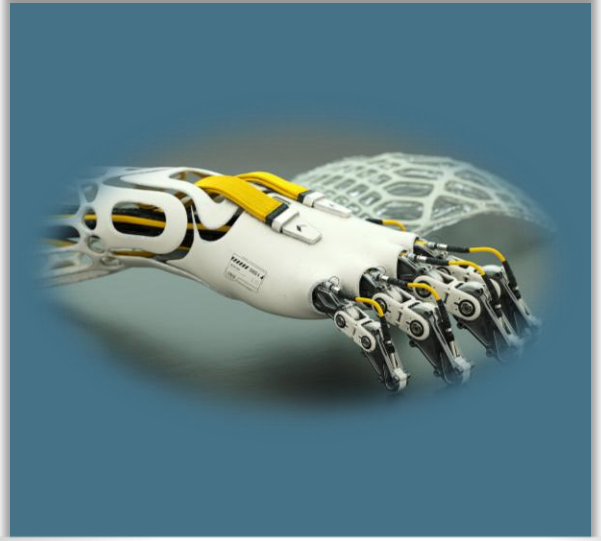
拓展：1×1卷积+卷积核数对输出的影响

- **1×1卷积通过调整卷积核的数量用于升维和降维**
  - 1×1卷积实际上是在每个空间位置（像素点）上，对输入特征图的所有通道进行加权求和，生成新的通道表示。
  - 当1×1卷积核的数量 **小于输入通道数** 时，输出特征图的通道数会被压缩。
    - 举例：输入为28×28×256的特征图，使用16个1×1卷积核时，输出可能变为28×28×16，通道数从256降至16。
  - 当1×1卷积核的数量 **大于输入通道数** 时，输出通道数增加。
    - 举例：输入为16×16×32的特征图，使用128个1×1卷积核后，输出可能变为16×16×128。

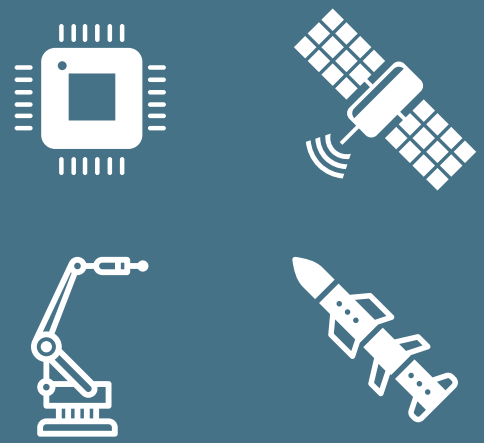


一层卷积的参数量由输入通道数 C_{in}、卷积核尺寸 K，和输出通道数 C_{out} 决定：

$$\text{参数量} = C_{in} \times K \times K \times C_{out} + C_{out} \quad \text{(含偏置)}$$



卷积神经网络



1. 从多层感知机到卷积神经网络
2. 卷积神经网络的基本组成和计算机制
3. 卷积神经网络的反向传播
4. 以卷积网络为骨干的早期深度学习模型概览
5. 卷积神经网络应用于不同任务

回顾：前馈神经网络中的反向传播算法

- **损失函数：**

$$L = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

- **基于递归定义的每层误差（标量形式）**

$$\delta_j^l = \frac{\partial L}{\partial a_j^l} = \sum_k \frac{\partial L}{\partial a_k^{l+1}} \cdot \frac{\partial a_k^{l+1}}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial a_j^l} = \sum_k \delta_k^{l+1} \cdot w_{kj}^{l+1} \cdot \sigma'(a_j^l)$$

下一层子节点传播的误差（左乘部分）

子节点激活值对当前节点激活的偏导

- 注意：上式中 a_j^l 是第l层第j个神经元的激活值。

- **基于递归定义的每层误差的向量形式**

$$\delta^l = \left((\mathbf{W}^{l+1})^\top \delta^{l+1}\right) \odot \sigma'(\mathbf{a}^l)$$

- 其中 $\mathbf{W}^{l+1} \in \mathbb{R}^{n^{l+1} \times n^l}$ 为第l+1层的权重矩阵；
- $\delta^{l+1} \in \mathbb{R}^{n^{l+1} \times 1}$ 为第 l+1 层的误差向量；

卷积层的反向传播

- 卷积层对图像的运算是部分连接运算。
- 卷积层的反向传播运算与全连接层类似。
- 假设l和$l+1$层都是卷积层，则对l层的(i,j)节点而言，其误差依旧定义为损失函数对其激活值（未通过激活函数的输出的梯度）：

$$\delta_{i,j}^{l}=\frac{\partial L}{\partial a_{i,j}^{l}}$$

- 上式展开有（标量公式）

$$\delta_{i,j}^{l}=\left(\sum_{p=0}^{K-1}\sum_{q=0}^{K-1}w_{p,q}^{l+1}\cdot\delta_{i+p,j+q}^{l+1}\right)\odot\sigma'(a_{i,j}^{l})$$

因为： $\boldsymbol{a}^{l+1}=\boldsymbol{\sigma}(\boldsymbol{a}^{l})*\boldsymbol{W}^{l+1}$

- 左侧公式写为矩阵形式有：

$$\delta^{l}=\mathrm{rot180}(W^{l+1})\star\delta^{l+1}\odot\sigma'(a^{l})$$

- 解释说明：由于CNN网络含有卷积结构，求导时卷积核被旋转了180度，即式中$\mathrm{rot180}(W^{l+1})$，具体操作是将卷积核W^{l+1}上下翻转一次，然后左右翻转一次。
- $\star$：表示卷积操作。

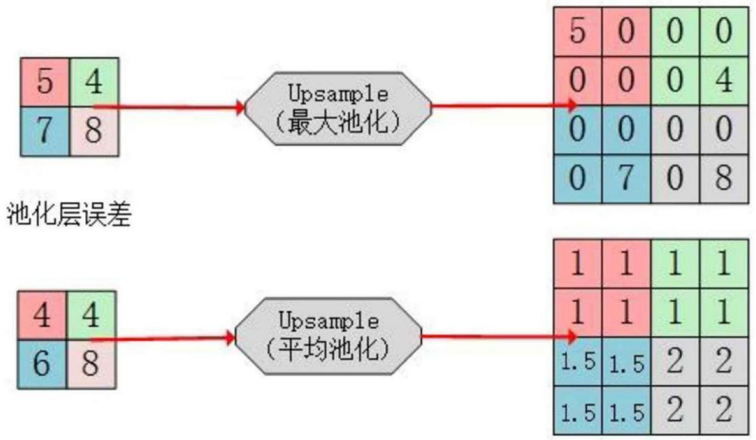
卷积神经网络的反向传播：处理池化层

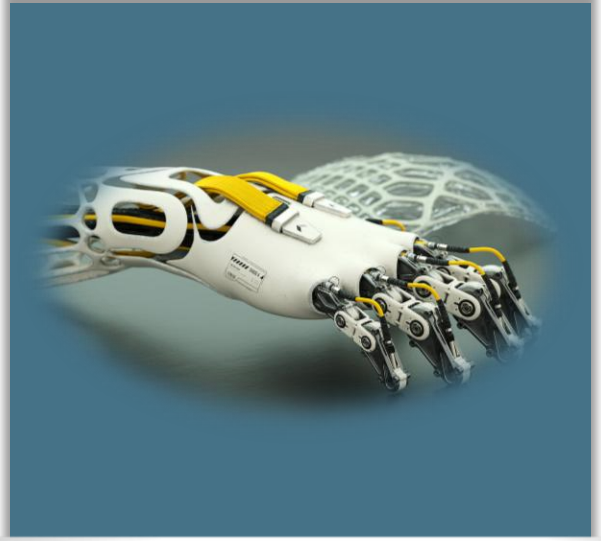
- **池化层的特性：**
  - 池化层没有要学习的参数，也不连接激活函数。但是造成了数据维度的变化 （下采样）。
  - 应对维度改变：误差反向传播时，需要一个上采样过程让数据维度恢复到下采样前。上采样方法与池化方法有关 。
- **假设l 层是卷积层，l+1层是池化层。反向传播规则为将后一层的误差映射回前一层更高分辨率特征图**
  - 最大池化：仅将下一层Feature Map某点的误差传递给前向传播中上一层Feature Map最大值所在的位置，其他位置误差置零：

$$\delta^l(x,y)=\begin{cases}\delta^{l+1}(i,j) & \text{if } (x,y)=\arg\max_{(x',y')\in \text{Window}(i,j)} a^l(x',y') \\ 0 & \text{otherwise}\end{cases}$$

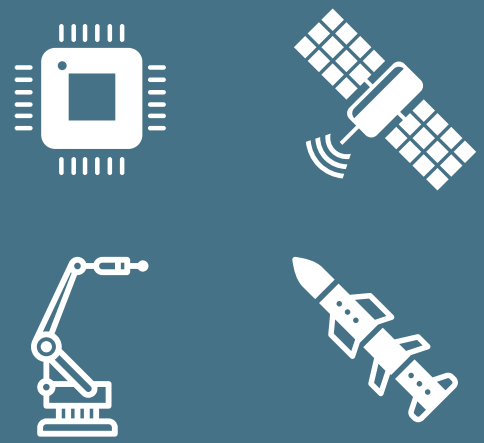
  - 平均池化：将误差均匀分配给前向池化窗口内的所有位置

$$\delta^l(x,y)=\frac{\delta^{l+1}(i,j)}{k\times k}$$





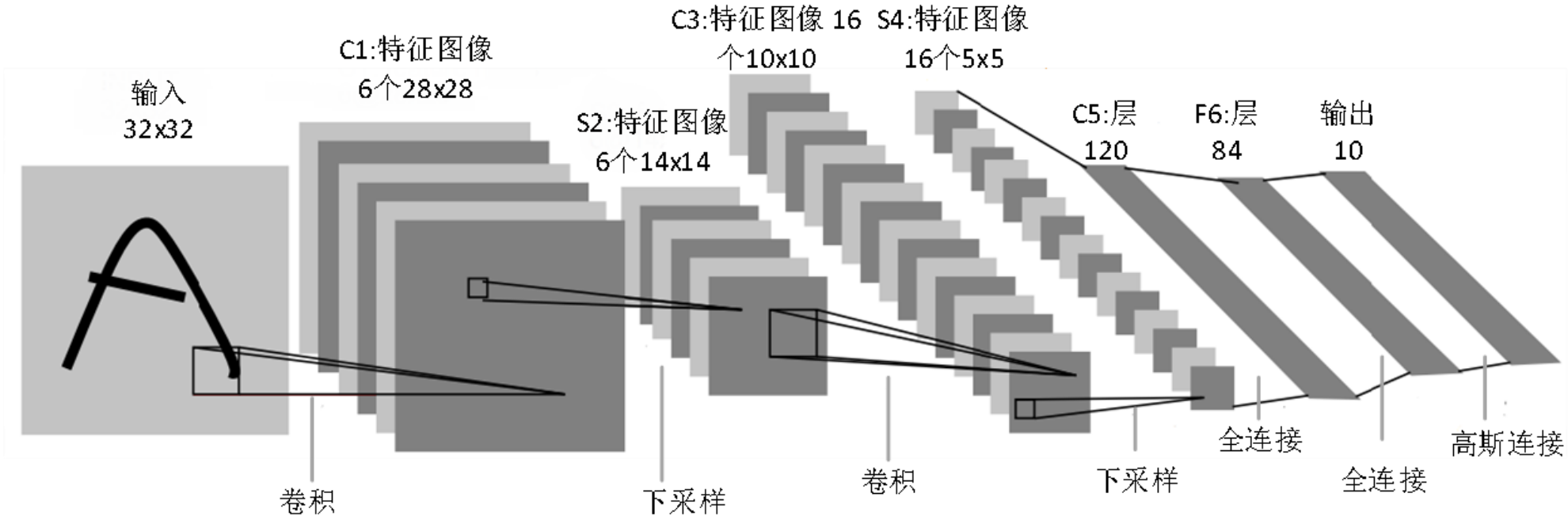
卷积神经网络



1. 从多层感知机到卷积神经网络
2. 卷积神经网络的基本组成和计算机制
3. 卷积神经网络的反向传播
4. 以卷积网络为骨干的早期深度学习模型概览
5. 卷积神经网络应用于不同任务

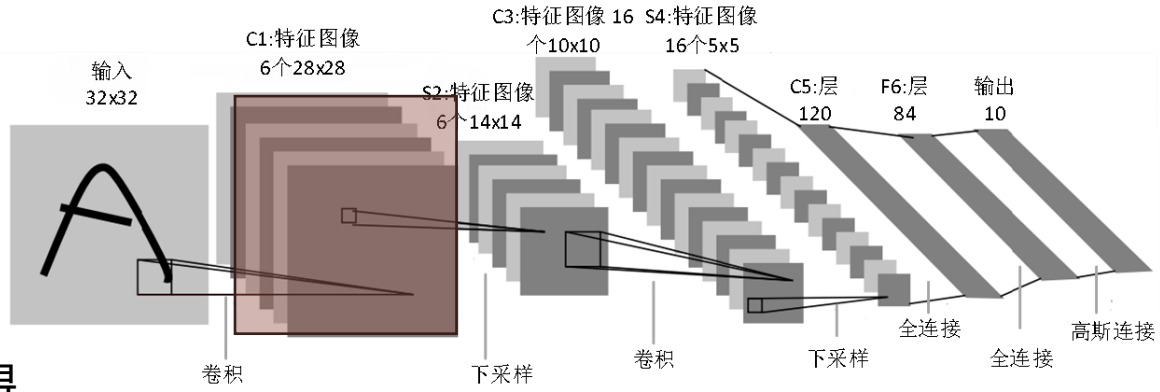
早期网络：LeNet5

- **LeNet（Yann LeCun，1998）：第一个成功取得应用效果的卷积神经网络**
  - 最初设计用于手写数字识别的卷积神经网络，当年大多数银行就是用LeNet5来识别支票上面的手写数字的，是早期卷积神经网络中最有代表性的实验系统之一。
- **LeNet5：**
  - 骨干网络结构：除输入层之外共有七层，分别为三个卷积层、两个池化层、一个全连接层和一个输出层。



早期网络：LeNet5（续）

- **LeNet5：**
  - 骨干网络结构：除输入层之外共有七层，分别为三个卷积层、两个池化层、一个全连接层和一个输出层。

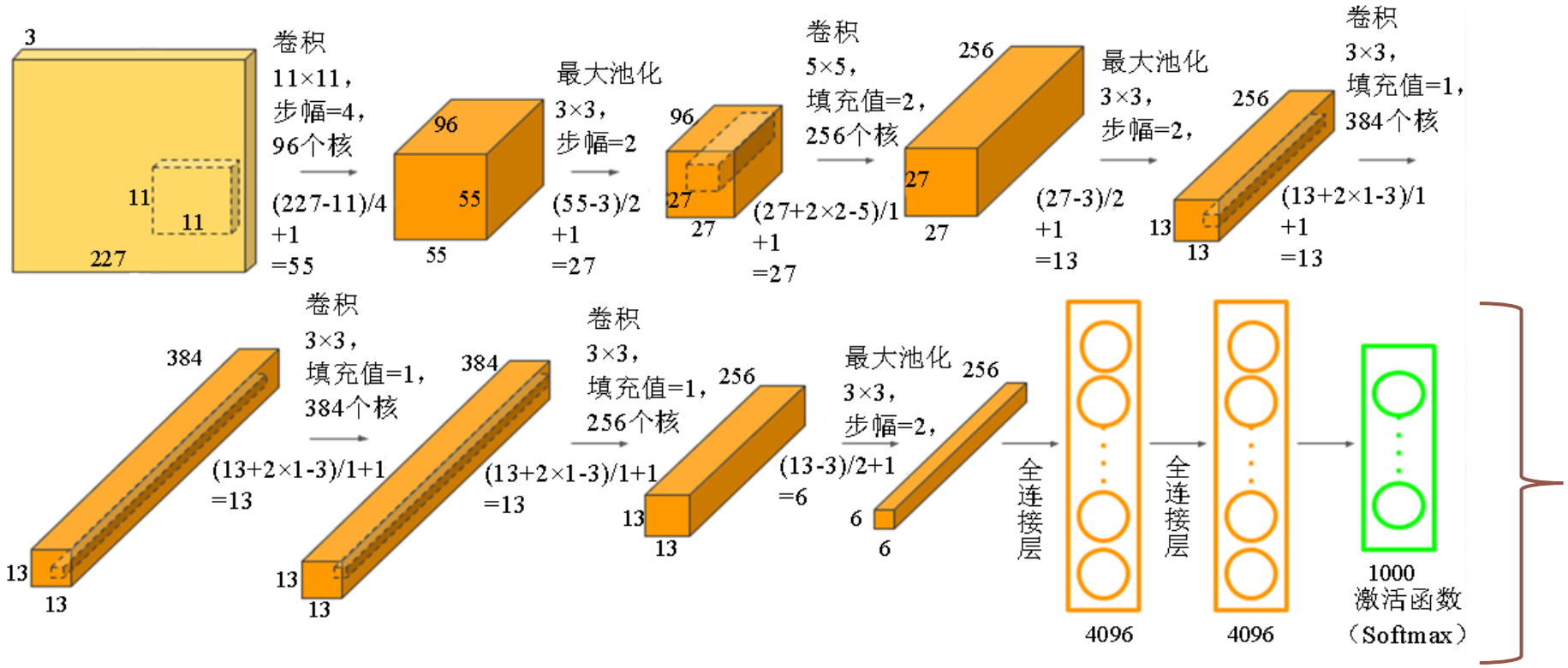


  - 分别使用6个大小为5×5的卷积核对X进行步长为1的卷积操作再加上偏置，并通过Sigmoid激活函数输出得到卷积层C1的6通道、大小为28×28的特征图。

  - 下图展示了一个训练好的LeNet5中卷积层C1对输入图像进行特征提取所得到的6个通道的特征图：

早期网络：AlexNet

- **AlexNet（Alex Krizhevsky，Ilya Sutskever，Geoffrey Hinton，2012）：CNN成为图像处理网络的骨干结构的里程碑**
  - AlexNet网络一共分为11层，含5个卷积层以及3个全连接层，除此之外还有3个池化层。在每一个卷积层中包含了激活函数ReLU以及局部响应归一化（LRN）处理，然后经过降采样（池化处理）。



1000个取值区间为(0,1)的输出，即输入样本的所属类别的概率预测。

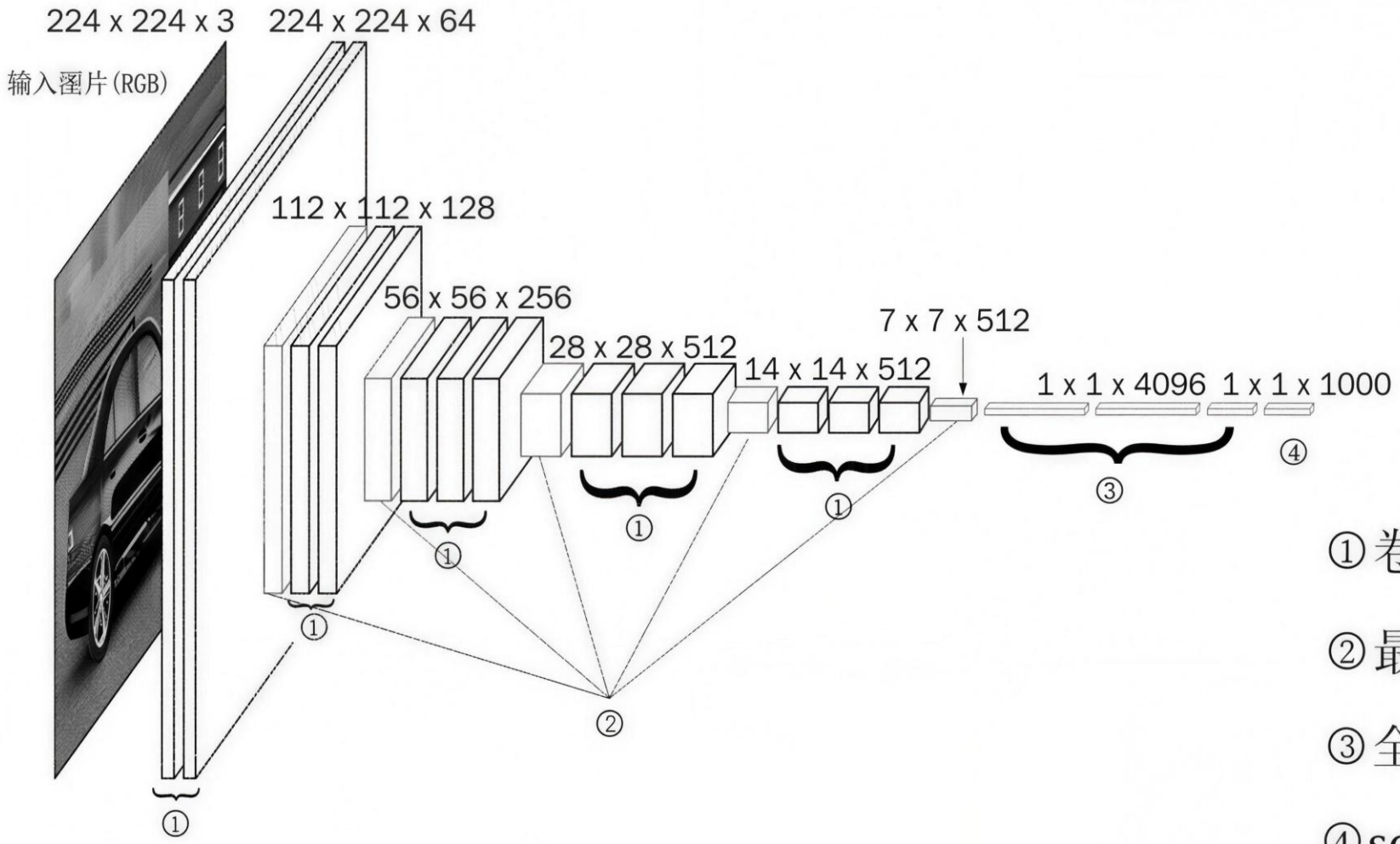
AlexNet相对于LeNet的技术进步

- **与LeNet5相比**
  - AlexNet除了使用模型容量更大的架构方式之外，还对激活函数进行了改进，使用ReLU激活代替Sigmoid激活。
  - AlexNet采用了许多当时看来的革命性训练技巧以加快训练速度并提升模型性能：
    - 采用GPU训练
    - 引入了数据增强与Dropout正则化；
    - 采用更高效的激活函数；
    - 采用最大池化方式；
    - 引入局部响应归一化。

早期网络：VGGNet——对AlexNet的改进

- **VGGNet（Visual Geometry Group Net，Karen Simonyan，2014）：构造更深层次的网络模型**
  - 简洁的网络构造：网络的模组一律采用步长（stride）为1的3×3的卷积核（filter），以及步长为2的2×2的最大池化（MaxPooling）。
  - 更深的网络深度：VGGNet能够达到19层（在当时是一个了不起的成就）。
  - VGGNet一共有6种不同的网络结构，每种结构都含有5组卷积，每组卷积都使用3×3的卷积核，每组卷积后进行一个2×2的最大池化，最后是3个全连接层。其中网络结构D就是著名的VGG16，网络结构E就是著名的VGG19。

VGGNet结构示意图



①卷积+ReLU函数

②最大池化

③全连接+ReLU函数

④softmax

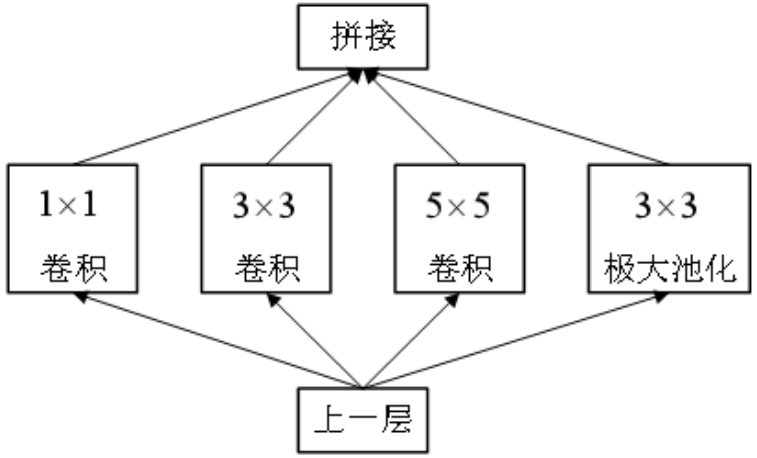
VGGNet的训练技术概览

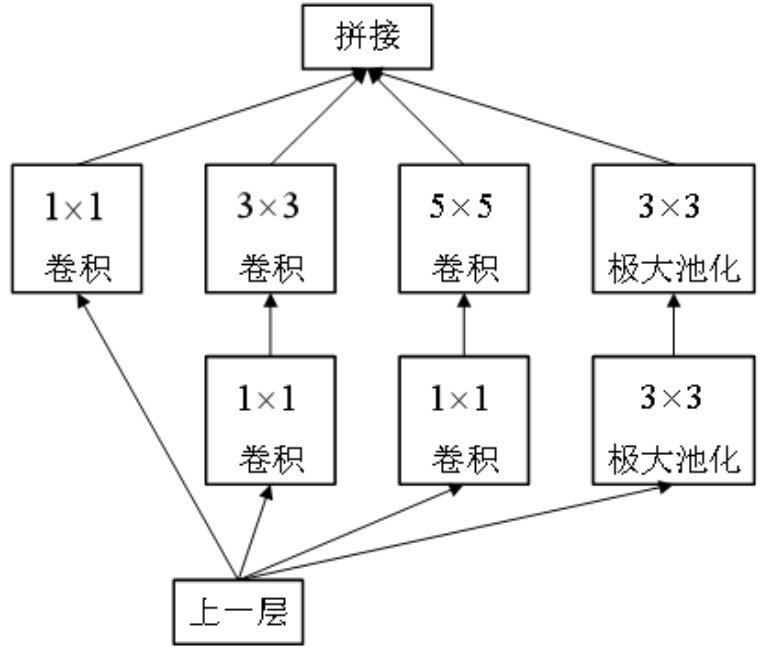
- **VGGNet的不同构型级别如左表所示**
  - 在训练时， VGGNet 先训练 A 级别的简单网络，再复用A网络的权重，来初始化后面的复杂模型，以加快训练收敛的速度。
  - 在预测时，VGGNet采用Multi-Scale（多尺幅目标检测）的方法，先将图像的尺寸变换为Q，并将变换后的图片输入卷积网络计算；然后在最后一个卷积层使用滑窗的方式进行分类预测，将不同窗口的分类结果平均，并将不同尺寸Q的结果平均后得到最后结果，能够提高图片数据的利用率并提升预测准确率。
  - 在训练过程中，VGGNet也使用了Multi-Scale的方法做数据增强，将原始图像缩放到不同尺寸S，然后再随机裁切成224×224的图片增加数据量，防止模型过拟合。
  - 左表中网络结构D就是著名的VGG16，网络结构E就是著名的VGG19。

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 权重	11 权重	13 权重	16 权重	16 权重	19 权重
输入(224 x 224 RGB 图片)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
最大池化层					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
最大池化层					
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
最大池化层					
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**
最大池化层					
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**
最大池化层(maxpooling)					
全连接层-4096(FC-4096)					
全连接层-4096(FC-4096)					
全连接层-1000 (FC-1000)					
激活函数 (Softmax)					

早期网络： GoogLeNet

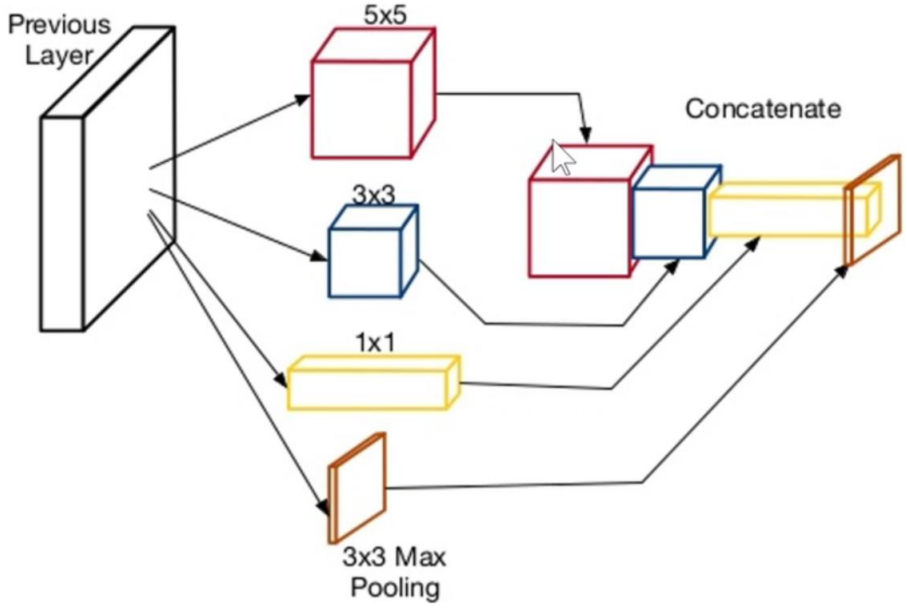
- **GoogLeNet（Christian Louboutin，2014）：引入著名的Inception结构**
  - 在GoogleLeNet之前的AlexNet、VGG等结构都是通过增大网络的深度（层数）来获得更好的训练效果，但层数的增加会带来很多负作用，如过拟合、梯度消失、梯度爆炸等。
  - GoogLeNet直接使用一个如下图所示的多路小型网络代替了网络中部分卷积层加池化层的架构方式，从而提取到更好的特征图，而该小型网络称之为Inception结构。
  - 网络宽度（Inception）的引入从另一种角度来提升训练结果，能更高效地利用计算资源，在相同的计算量下能提取到更多的特征。





GoogLeNet的Inception结构

- **多尺度卷积核与感受野**
  - 使用并行的不同尺度的1x1、3x3、5x5大小的卷积核和池化操作，分别提取具有不同感受野特征信息，提升了网络了宽度。同时多尺度信息的提取也增加了网络对不同尺度特征的适应性，有助于下一层提取不同尺度的特征信息。
- **Max-Pooling支路**
  - 原始论文还提到一条Max-Pooling（3×3，步长1）分支，随后通过1×1卷积调整通道数。但Max-Pooling能保留特征图的显著信息，并通过池化操作扩大感受野，同时参数量几乎不增加。
- **1×1卷积的降维作用**
  - 在3×3和5×5卷积前加入1×1卷积，是为了减少通道数（降维），从而降低后续大卷积核的计算量。
  - 例如，若输入通道数为256，直接使用5×5卷积会带来 256×5×5=6400 个参数，而先通过1×1卷积将通道数降至64，则参数量减少至 64×5×5=1600，显著降低了计算复杂度。



加入Maxpooling支路的Inception结构

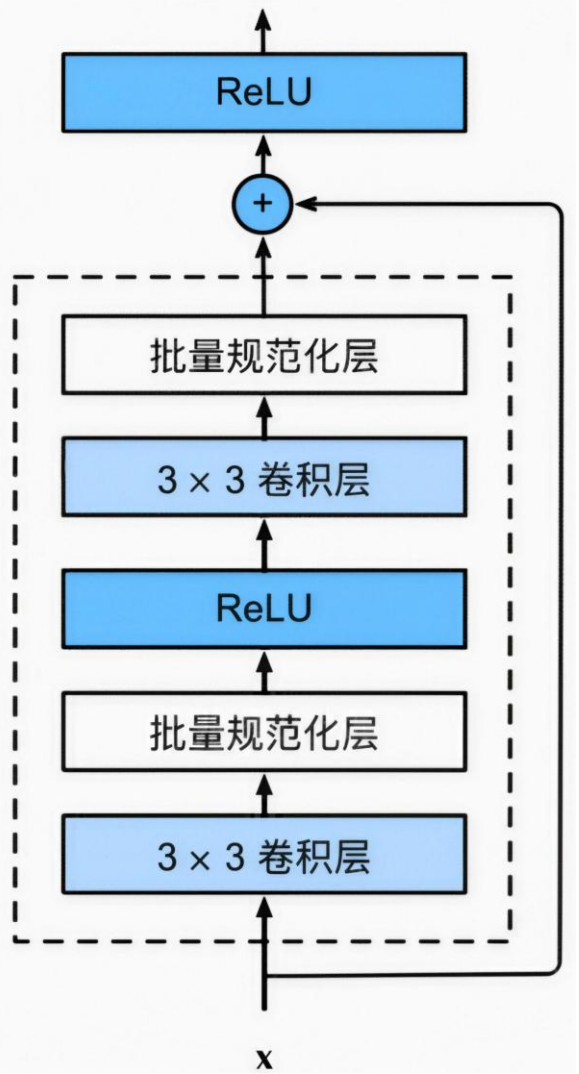
早期网络的改进：ResNet——残差网络

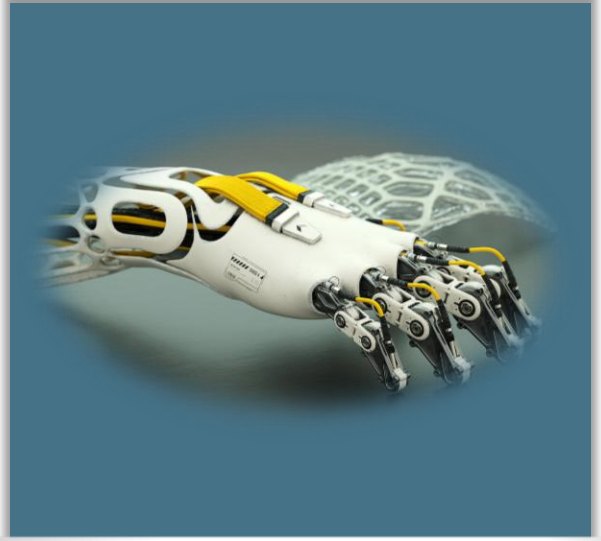
- **ResNet（何恺明等，2015）：通过引入残差块（residual block），解决了深层网络训练中的梯度消失问题，使网络能够高效地堆叠上百层甚至上千层**
  - 残差学习：通过跳跃连接（skip connection）将输入直接传递到后续层，即

$$y = F(x, \{W_i\}) + x$$

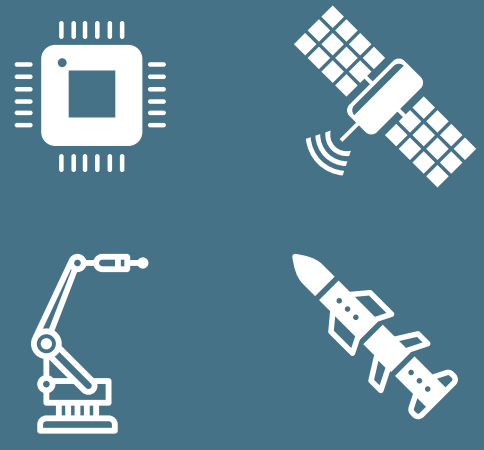
其中，F为之前所有网络层对于样本x的非线性变换。

  - 分阶段设计：网络分为多个阶段（stage），每个阶段包含多个残差块，逐步降低空间分辨率并增加通道数。残差可以设计为跨多个卷积层连接（一般只跨越2至3层，但跨越更多层的连接方式也是可行的）。





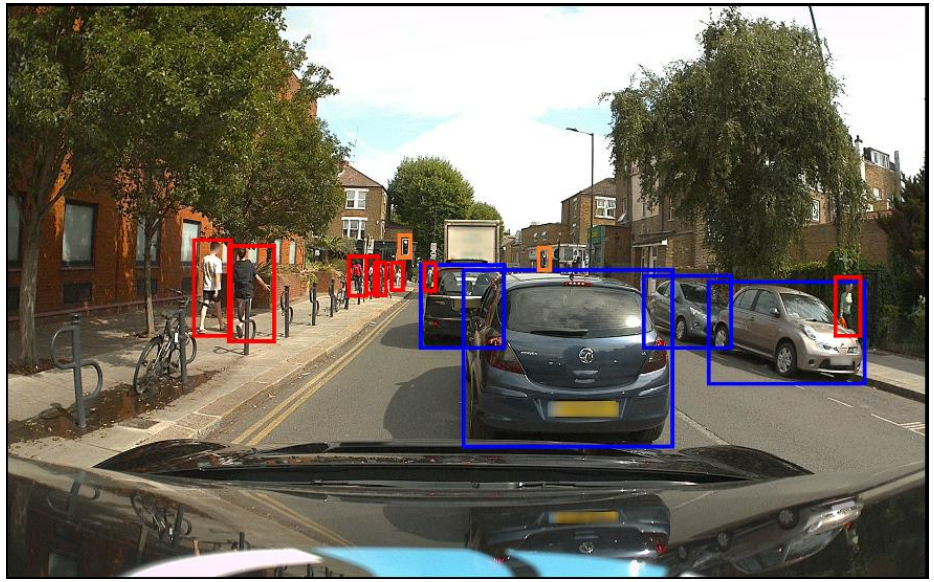
卷积神经网络



1. 从多层感知机到卷积神经网络
2. 卷积神经网络的基本组成和计算机制
3. 卷积神经网络的反向传播
4. 以卷积网络为骨干的早期深度学习模型概览
5. 卷积神经网络应用于不同任务

目标检测任务

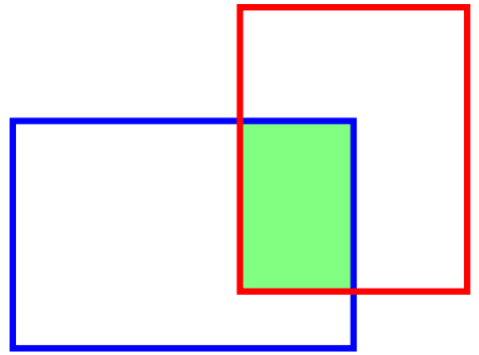
- **边界框（Bounding Box）**
  - 常见场景：同一幅图像中有多个对象属于一个或多个类别。
  - 处理目标：检测图像中任何对象的位置。
- **边界框的定义**
  - 一个紧密贴合对象边界的矩形，形式为向量$b = (b_x, b_y, b_W, b_H)$。
  - b中的元素可以以像素为单位，也可以用连续数字。
  - 惯例：图像左上角对应坐标（0,0），右下角对应坐标（1,1）。
- **同时输出物体分类和位置边界（假设只有一个类别对象）**
  - 分类部分：网络有C个输出单元（对应C个类别），使用Softmax激活函数来预测物体属于哪个类别。
  - 定位部分：网络额外增加4个输出单元，使用线性激活函数，来直接预测边界框的四个坐标（通常是中心点x, y，以及宽度w和高度h）。由于坐标是连续值，通常使用均方误差（Sum-of-Squares Error）来训练这部分。



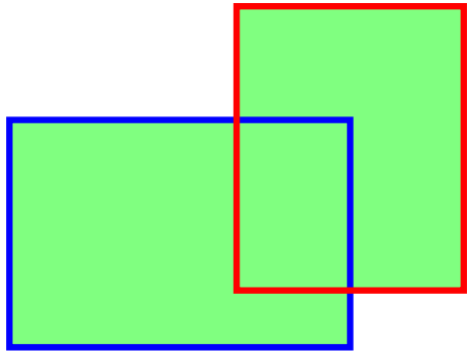
图来源：D.L.F.C. by Bishop

交并比（Intersection-over-Union，IoU）

- **图像分类准确性测量：标签的对数似然**
  - 图像分类的网络输出是（Softmax输出）类别标签的概率分布。
  - 可以通过查看测试集上的真实类别标签的对数似然衡量性能。
- **对象定位准确性测量：交并比**
  - 在训练和测试集中，目标框（真实对象位置）可通过人工标注的方式获得。
  - 交并比定义：$\mathrm{IoU} = \dfrac{\text{预测框与真实框的“交集”面积}}{\text{预测框与真实框的“并集”面积}}$
  - 惩罚过度预测：IoU值在0到1之间，1表示完全重合，0表示没有重叠。并集面积包含了预测框超出真实框的部分，因此IoU会自然地对“超出”进行惩罚。
  - 注意：由于计算负杂性和梯度消失问题，标准IoU通常不直接用于训练的损失函数构成。



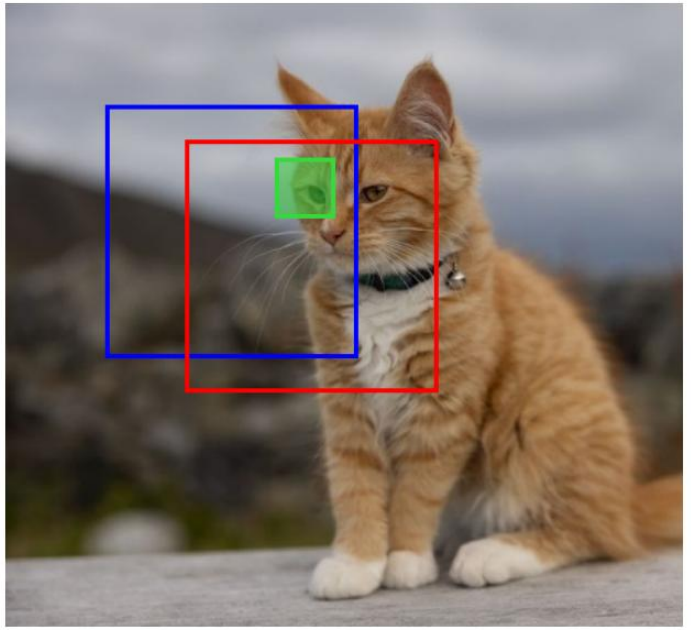
交集区域（绿色）



并集区域（绿色）

滑动窗口检测和卷积

- **传统滑动窗口训练阶段：**
  - 准备一个数据集，里面包含裁剪好的物体图片和背景图片，先用来训练一个分类器（如CNN）。
- **传统滑动窗口检测阶段：**用训练好的分类器去扫描新图片。
  - 用一个固定大小的“窗口”在图片上从左到右、从上到下依次滑动。
  - 每滑动到一个位置，就把窗口内的图像块裁剪出来，输入分类器判断其中是否有物体。
  - 如果分类器给出高置信度，当前窗口的位置就构成了一个预测的边界框。
- **传统滑动窗口检测的缺陷：计算成本极高（如右图）**
  - 位置数量多：一张图片上有成千上万个可能的窗口位置。
  - 尺度变化多：为了检测不同大小的物体，必须用不同尺寸的窗口重复整个滑动过程，计算量成倍增加。

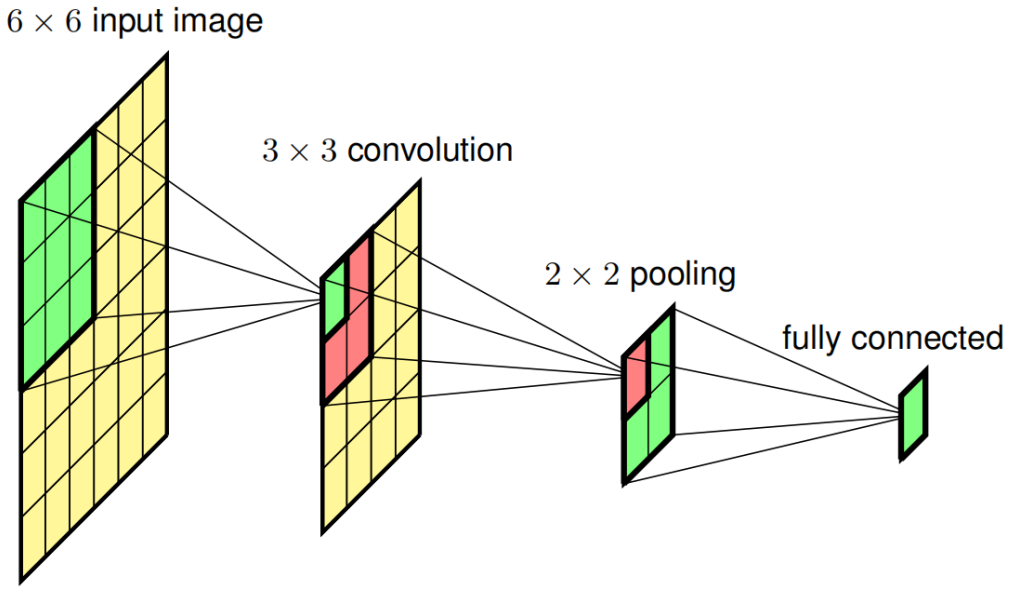


使用CNN处理滑动输入窗口数据时的重复计算情况：

- 图中红色和蓝色框展示了输入滑动窗口的两个重叠位置。
- 绿色框代表第一个卷积层中某个隐藏单元感受野的一个位置。该隐藏单元激活的计算可以在两个窗口位置之间共享。

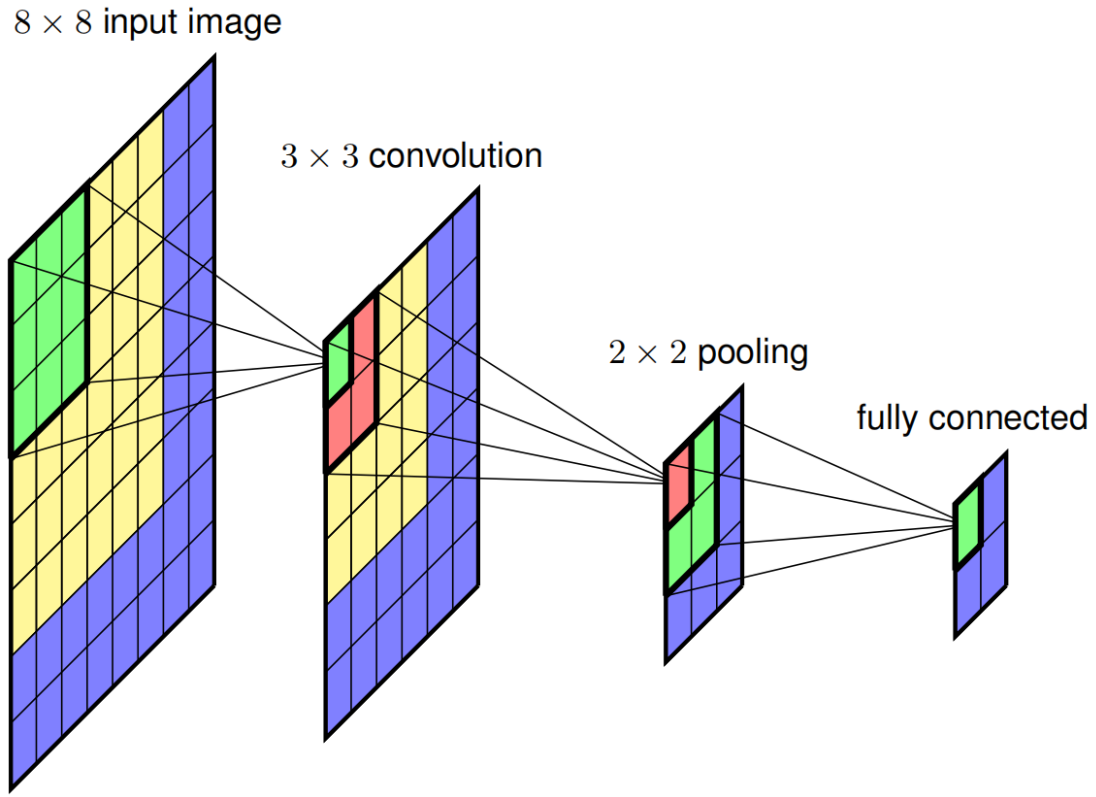
图来源：D.L.F.C. by Bishop

通过卷积（计算共享）完成滑动窗口检测



- 一个在6 × 6裁剪小图上训练好的分类网络，传统上需要被反复调用（滑动窗口）来处理大图。
- 图中，最后的全连接层可视为一个卷积层。这个最终全连接层连接到一个2 × 2的输入区域（池化输出），因此可以等价地看作一个滤波器尺寸为2 × 2 、通道数匹配的卷积层。在训练网络中，这个“卷积核”只在2 × 2输入上有一个位置，因此只产生1个输出值。

- 检测阶段网络：输入变为更大的 8 × 8图像。
- 计算共享：最终4个输出值，正好对应原始大图中4个不同位置的6 × 6滑动窗口的分类结果。
- 蓝色方块表示，在计算不同窗口时，只有顶层的“卷积化全连接层”需要在新位置进行少量计算，而底层庞大的计算量被完全共享。

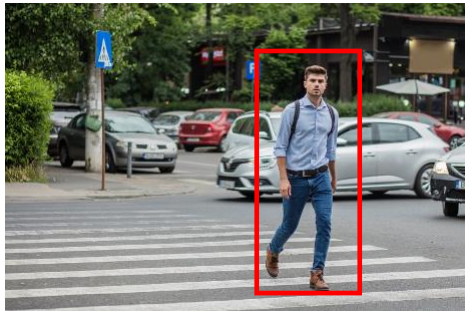


跨尺度检测

- **核心问题：**
  - 一个固定大小的检测窗口无法有效检测不同大小的物体（如近处的大猫和远处的小猫）或不同形状的物体（如站立的猫（高瘦）和趴着的猫（矮胖））。
- **检测方案：图像金字塔 + 固定检测器输入窗口。**
  - 构建图像金字塔（特征金字塔的前身）：将原始图像以不同的水平与垂直缩放因子进行多次缩放，生成一系列不同尺寸的图像副本。
  - 单尺度滑动检测：用同一个训练好的固定窗口检测器，分别在每个缩放后的图像副本上进行完整的滑动窗口扫描。

原图像

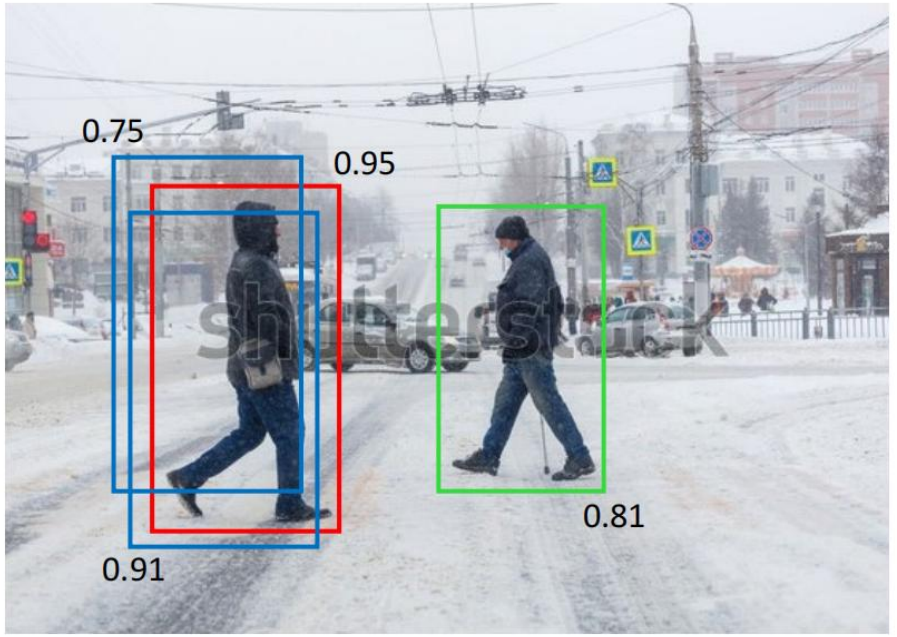
水平方向缩放后的检测结果



检测窗口投影回到原图像

非极大值抑制（Non-Max Suppression）

- **NMS按物体类别逐一处理，获取所有检测框：**
  - 用检测器扫描完整图像，得到某个类别所有可能的检测框及其置信度（概率）。
- **过滤低置信度框**
  - 设定一个置信度阈值（例如0.7），丢弃所有低于此阈值的检测框。此时剩下的都是“可能”的物体。
- **迭代选择与抑制**
  - 选择：从剩余的框中，选出置信度最高的那个框，将其标记为一个“成功检测”，并记录其边界框。。
  - 抑制：计算其他所有框与这个“成功检测框”的交并比（IoU）。设定一个IoU阈值（例如0.5），丢弃所有IoU超过此阈值的框。（如果其他框与当前最佳框高度重叠，则认为它们检测的是同一个物体，因此只保留最好的一个。）
  - 重复：在剩下的框中，再次选择置信度最高的框作为有效检测，重复上述操作。



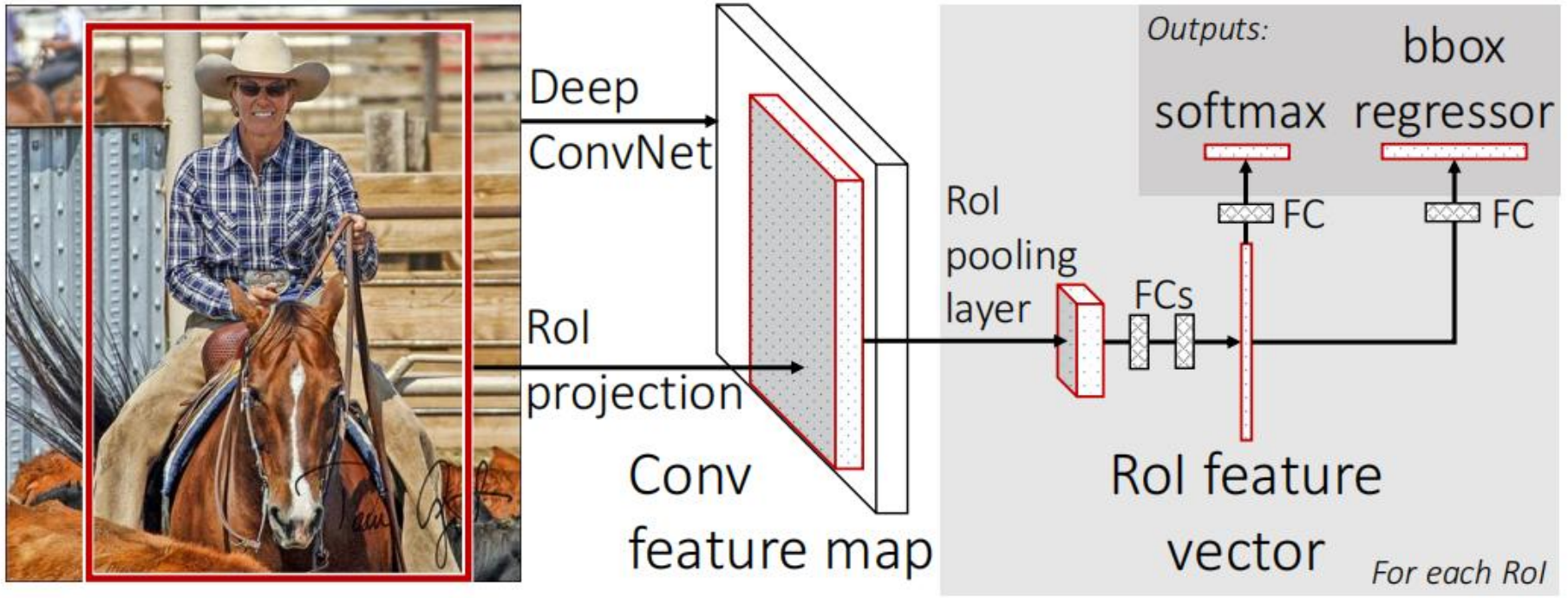
同一对象在临近位置被多次检测及关联概率的示意图（红色边界框对应最高的分类概率）。

图来源：D.L.F.C. by Bishop

两阶段检测方法：Fast R-CNN（快速区域提议卷积网络）

- **区域提议网络：解决“全图扫描”的低效问题**
  - 不做滑动窗口深度扫描，先用一个低计算成本的方法快速找出可能包含物体的“候选区域”，再只对这些区域进行精细检测。

- **方法**
  - Fast R
  - Faste
    专门
- **方法特点**
  - 使用
  - 通过
    图。



卷积网络，

）。

7）的小特征

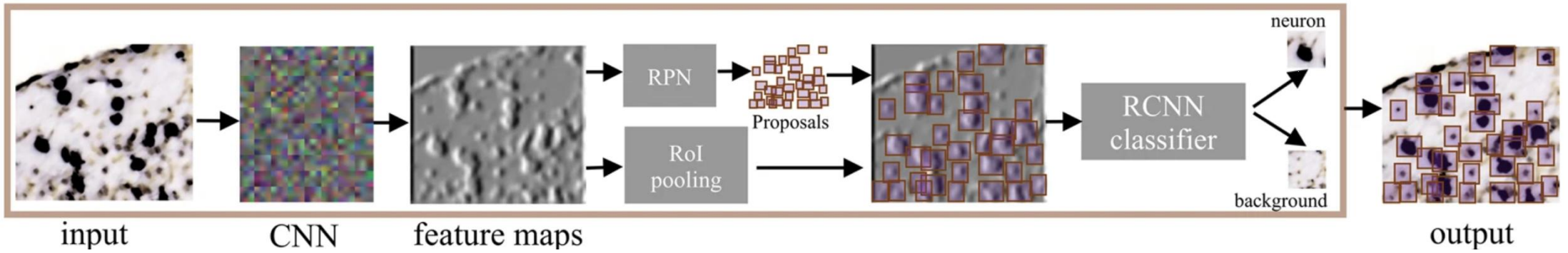
两阶段检测方法：以Faster R-CNN为例

- **两阶段损失联合训练：**区域提议网络（RPN）阶段损失 + 检测头阶段的损失
  - 原理：在骨干网络输出的特征图上滑动一个小的窗口，在每个位置上预设多个不同尺度和长宽比的锚点框，判断每个锚点框是“前景”（有物体）还是“背景”，并对其位置进行初步回归修正，输出候选区域提议。
  - **RPN阶段任务**：判断候选目标框（锚框）是前景还是背景（二分类任务）；以及初步位置回归。

损失函数：$L_{RPN} = L_{RPN_{cls}} + \lambda_{RPN}\ L_{RPN_{reg}}$　　二分类损失（交叉熵）+边界框回归损失

  - **检测头阶段任务**：在RPN输出的候选区域上，需要完成具体类别分类和位置精修。

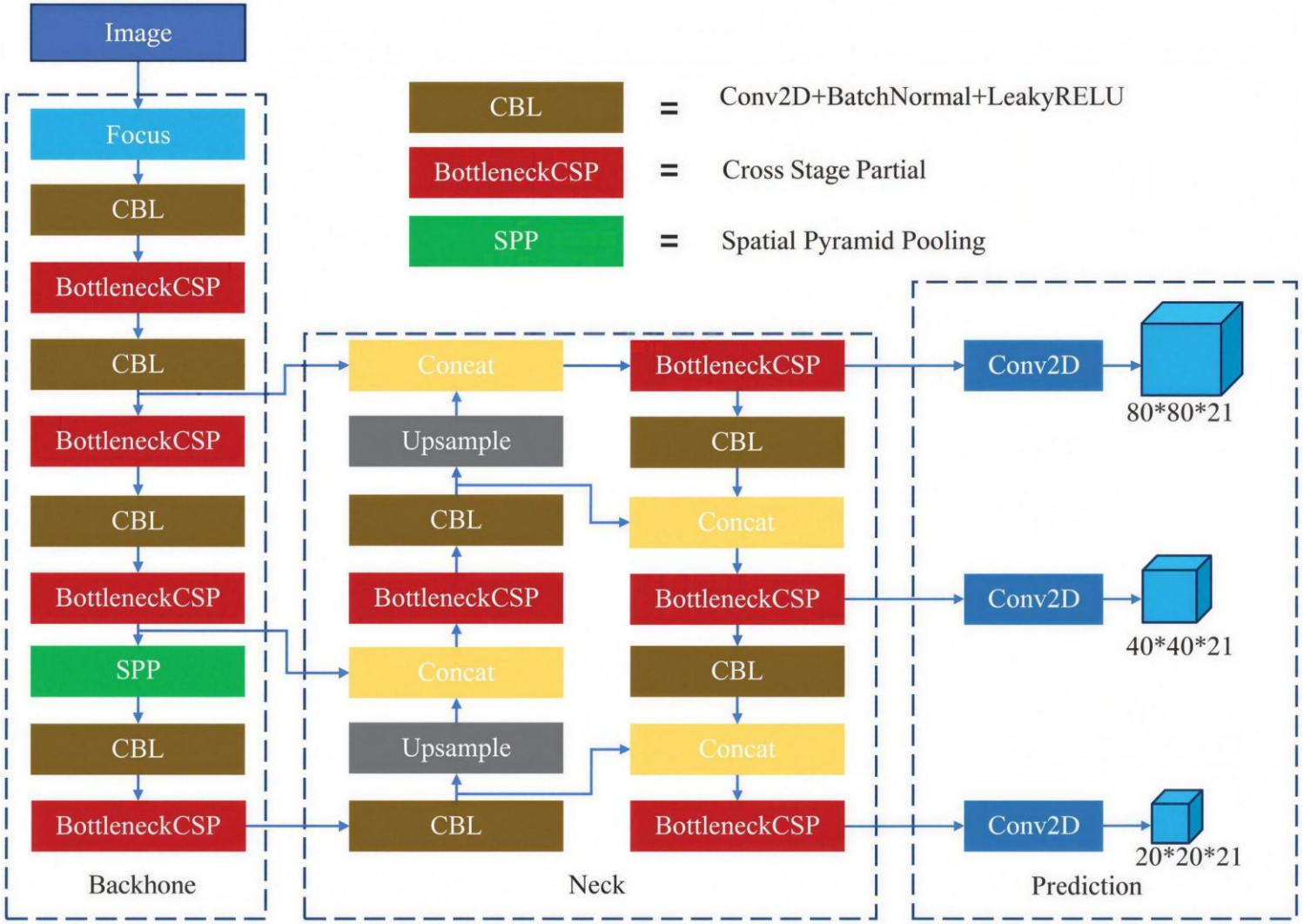
损失函数：$L_{\text{Detect}} = L_{\text{Detect}_{cls}} + \lambda_{\text{Detect}}\ L_{\text{Detect}_{reg}}$　　多分类损失（交叉熵）+边界框回归损失



图来源: “DeNeRD: high-throughput detection of neurons for brain-wide analysis with deep learning”, Scientific Report

单阶段检测方法：以YOLO为例

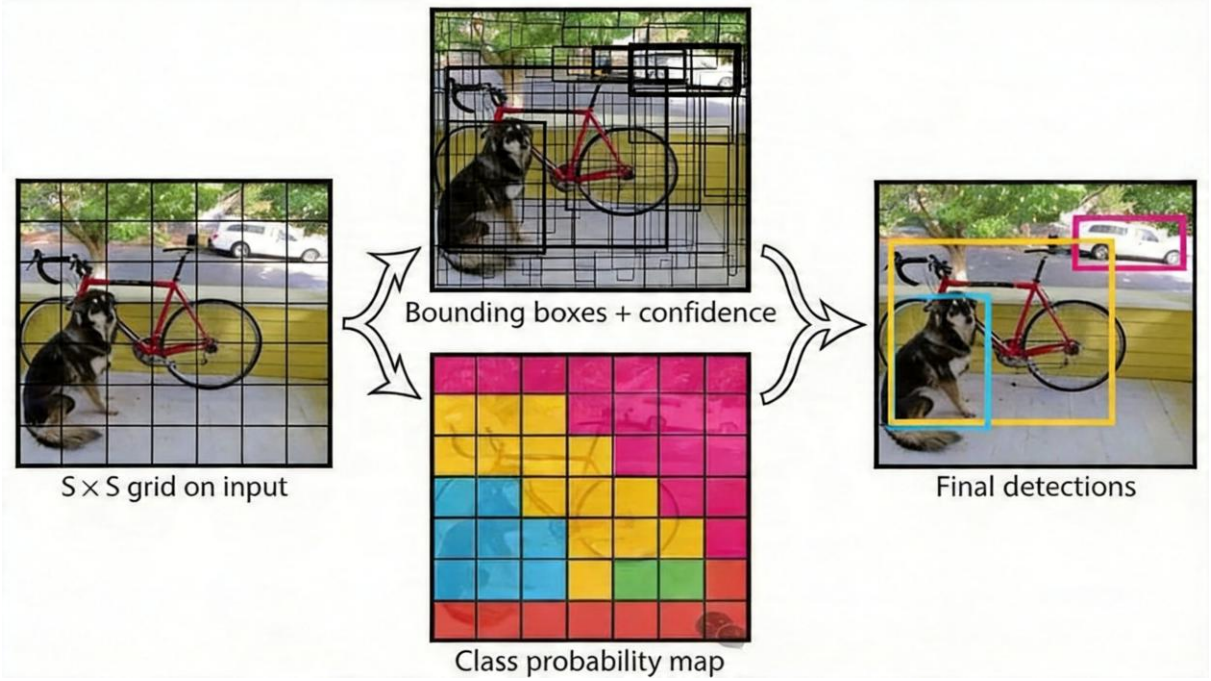
- **三模块架构设计**
  - 骨干网（Backbone）：从输入图像中提取多尺度特征，主要实现有CSPDarknet53（YOLOv4/v5）等。
  - 颈部网络（Neck）：整合不同层次的特征信息，常见结构有特征金字塔网络（金字塔池化、FPN\BiFPN，PANet）等。
  - 头部网络（Head）：预测输出，生成边界框、类别和置信度。



YOLOv5网络中的三模块架构

YOLO的检测过程

- **核心思想**
  - 早期（YOLOv1，见右图）：输入图像划分为S×S的网格（S=7），每个网格单元负责预测中心点落在该网格内的目标。
  - 后期（YOLOv5）：在特征图上以多尺度形式划分网格（特征金字塔的P3/P4/P5三层，对应不同stride），每个网格单元负责预设多个Anchor框，不仅检测中心点落在网格内，同时通过Anchor尺寸匹配策略确定正样本，每个正样本Anchor独立预测边界框、目标置信度和类别概率。



YOLO的网格划分机制

- **损失函数设计（YOLOv5）**

$$L_{\text{YOLOv5}} = \lambda_{box} L_{box} + \lambda_{obj} L_{obj} + \lambda_{cls} L_{cls}$$

左侧公式符号含义：

- L_{box}：边界框回归损失，使用CIoU损失；
- L_{obj}：锚框二分类损失（有/无物体），对三个尺度计算交叉熵损失再加权。
- L_{cls}：分类损失，使用二元交叉熵（BCE）损失，因此允许一个目标属于多个类别。

图像分割任务

- **语义分割（Semantic Segmentation，见右图）**
  - 每个像素都被分配给预设的类别之一：每个像素按最高概率的类别着色。
  - 输出空间维度与输入空间维度相同。
- **简单的CNN方法构想（不经济）**
  - 构建一个分类器：训练一个CNN，其输入是一个以目标像素为中心的小图像块，输出是一个Softmax，用于预测该中心像素的类别。
  - 滑动应用：将这个训练好的分类器像滑动窗口一样，依次应用到图像的每一个像素位置上，为所有像素生成分类标签。
  - 利用卷积的权值共享特性，将整个流程整合成一个单一的、全卷积的网络：将网络最后的全连接层全部替换为卷积层（例如，将全连接层视为 1×1卷积）。为保证输出维度，移除所有的池化层，并设置所有层的步长为1，使用相同填充参数。

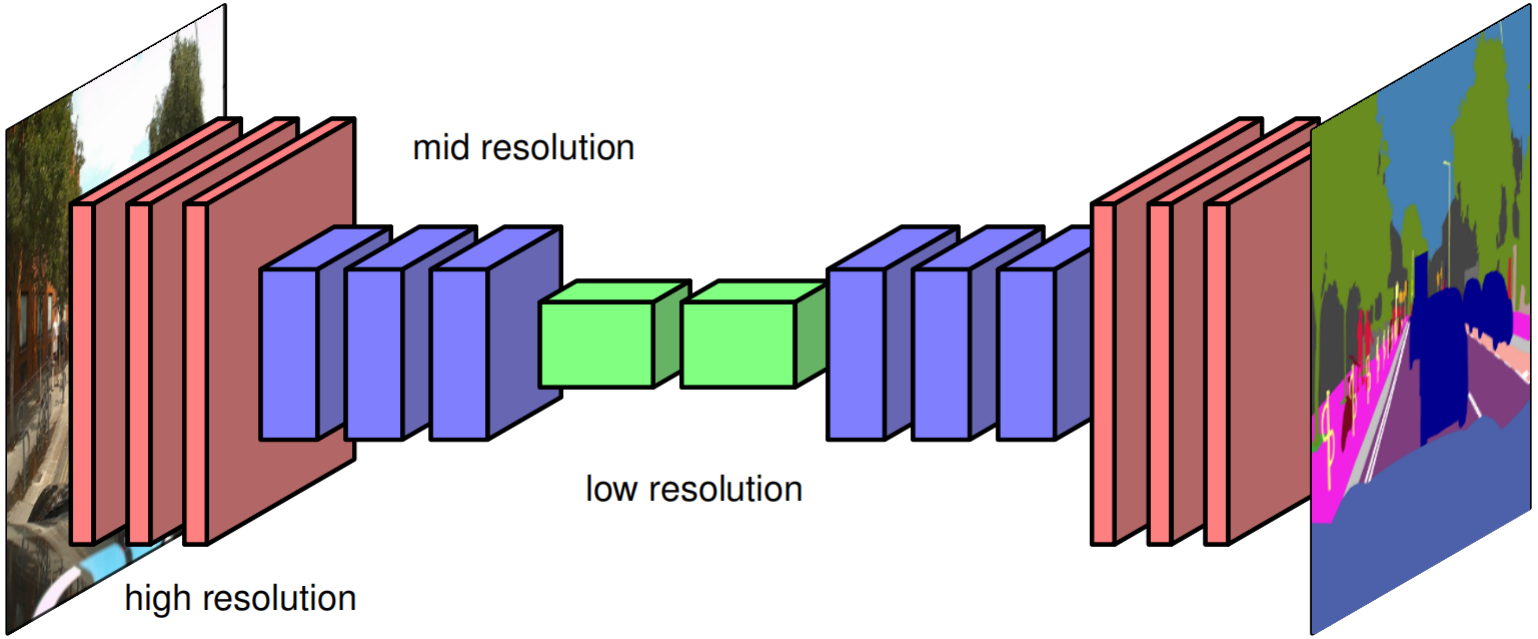
分割结果

图来源：D.L.F.C. by Bishop

图像分割任务：上采样（Upsampling）

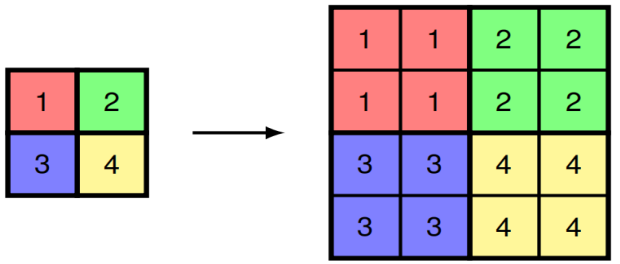
- **为什么需要下采样（Downsampling）？**
  - 卷积层的一般操作：随着网络加深，通道数逐渐增加（提取更复杂的特征），而特征图尺寸逐渐减小。
  - 网络能够从图像中提取出语义上有意义的高阶特征，同时控制网络的总计算量和内存占用。
- **语义分割任务和下采样**
  - **由任务特性决定：**语义的最终输出分辨率应与原始图像一致。
  - 先使用一个标准的深度卷积网络（如分类网络）进行下采样，得到低维的内部特征表示。
  - 然后……如何处理维度gap?
- **上采样：下采样的逆过程**
  - 提供额外的（可学习）层，将这些经由下采样获得的低维特征上采样回原始图像分辨率。
  - 方法：利用池化操作的逆过程，如带位置记录的反最大池化操作，恢复图像尺寸。

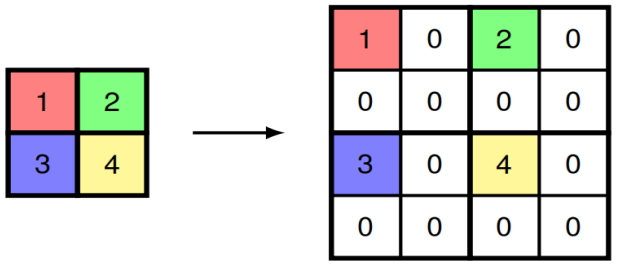
图像分割任务：上采样（Upsampling）



图来源：D.L.F.C. by Bishop

- **编码器-解码器结构（下采样再上采样）的卷积分割网络：**
  - 特征图维度首先通过一系列步长卷积和逐步恢复至原始图像尺寸的过程。

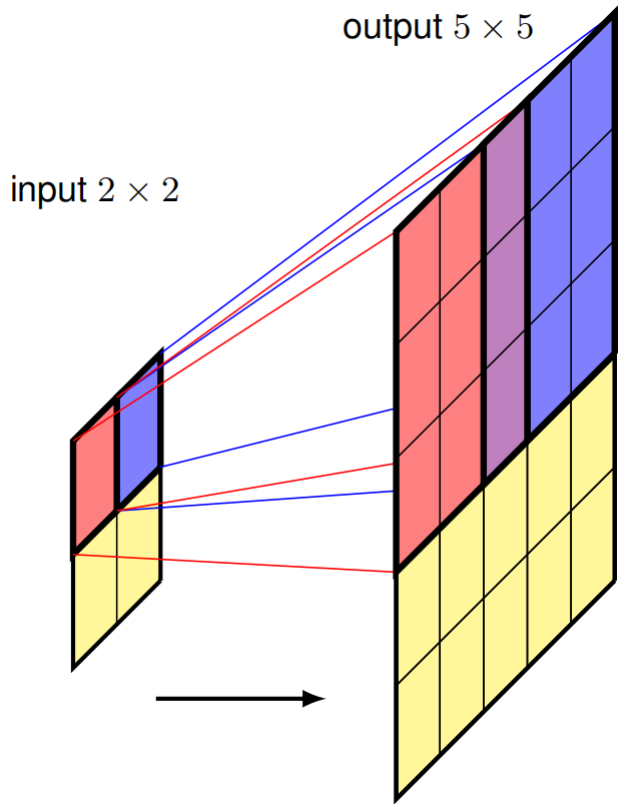




（a）平均池化和(b)最大池化的上采样

参数可学习上采样：转置卷积

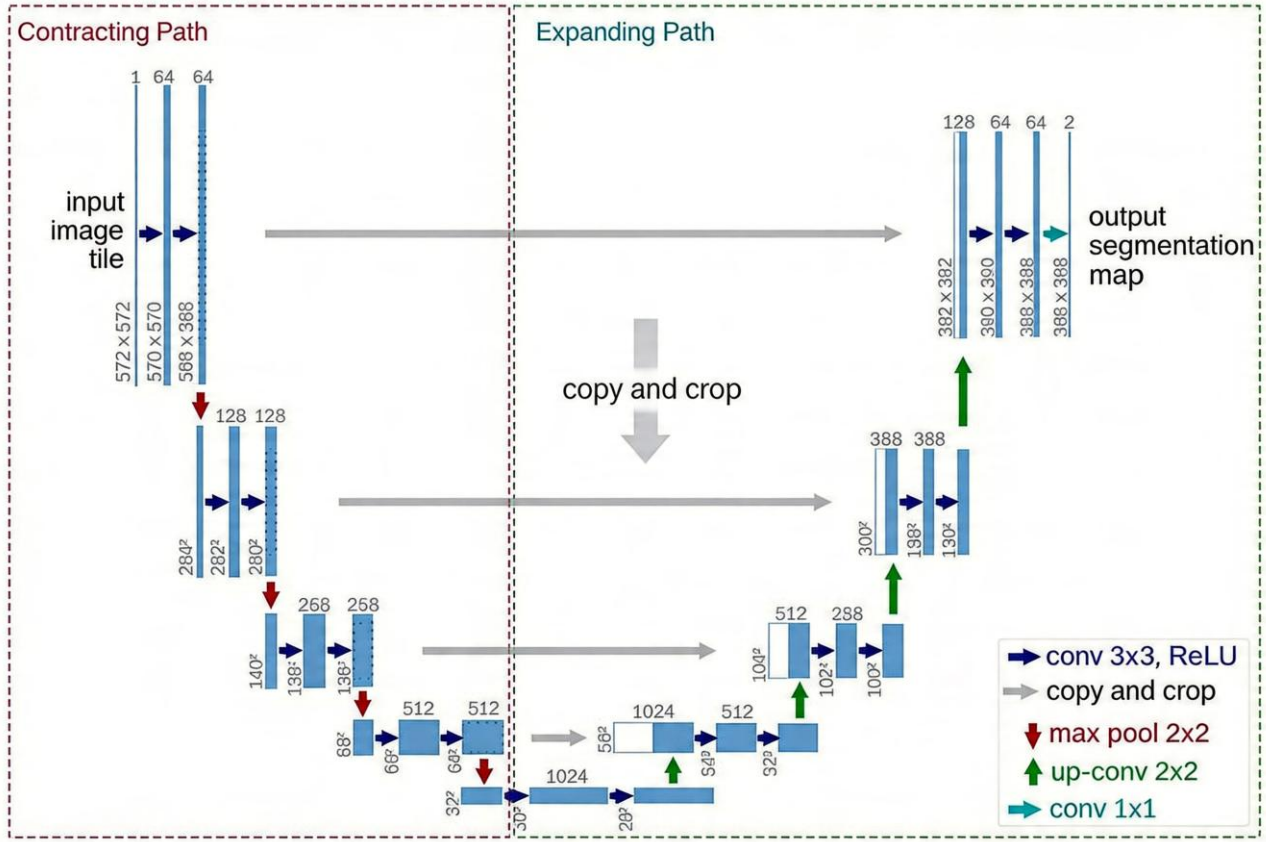
- **转置卷积是下采样“步长卷积”（Strided Convolution）的反操作**
  - 下采样（步长卷积）：一个小滤波器在输入特征图上以步长 > 1滑动。输出像素对应输入的一小块区域，由于步长大，输出尺寸变小。
  - 上采样（转置卷积）：与步长卷积相反。一个输入像素通过滤波器关联到输出特征图上的一个块（如3 × 3，见右图）。通过设置网络结构，使得输入移动1步时，滤波器在输出上移动 >1 步（如2步，见右图），从而输出尺寸变大。
- **转置卷积操作**
  - 插值：输入元素间插入（Stride-1）个零。
  - 填充：边界填充（Kernel_size-1-padding）个零。
  - 卷积：以 stride=1 进行标准卷积。



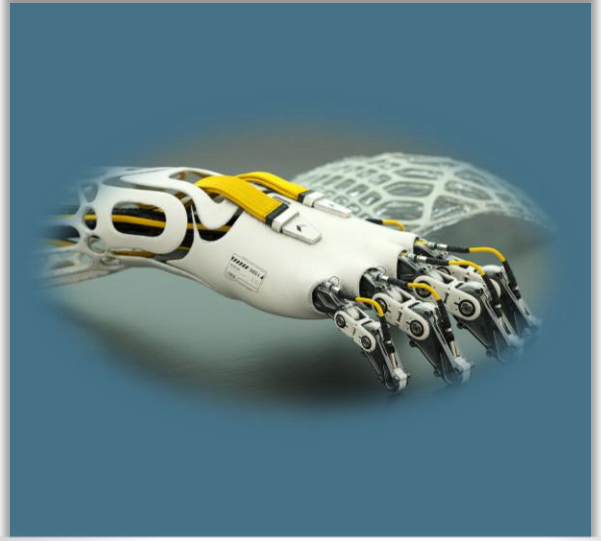
- 一个 3×3 卷积核、stride=2 的转置卷积操作；
- 当多个输出区域重叠时，重叠单元格的激活值 = 各区域贡献的求和或平均

语义分割网络：U-Net

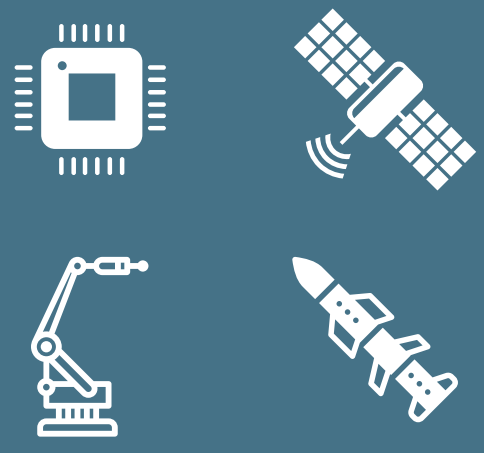
- **U-Net的对称的编码器-解码器结构**
  - **编码器**（下采样路径，U-Net的左半边）：由卷积和下采样层堆叠而成，负责提取深层语义特征。
  - **解码器**（上采样路径，U-Net的右半边）：由转置卷积等上采样操作和卷积层堆叠而成，负责将特征图尺寸逐步放大，恢复高分辨率。
  - **跳跃连接**：将编码器路径中同尺度特征图（在池化/下采样之前）直接拼接到当前上采样后的特征图上。
  - **输出层处理**：使用 1×1卷积将通道数映射到目标类别数，再对每个像素位置的所有通道值应用 Softmax激活函数，得到每个像素属于各个类别的概率分布。



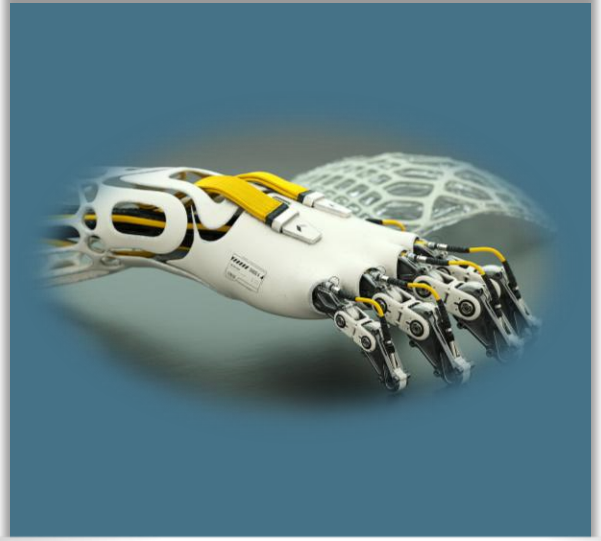
一个典型的二分类（见输出通道数）U-Net网络结构



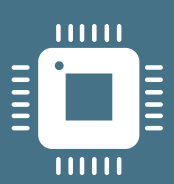
卷积神经网络



讨论



卷积神经网络




附录：以Pytorch为例构建神经网络

附录.1 Pytorch简介

附录.2 Pytorch数据的组织

附录.3 神经网络的构建

Pytorch简介：Pytorch的程序体的一般结构

- dataloader：数据载入例子

```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
```

- PyTorch提供特定领域的库，如TorchText、TorchVision和TorchAudio，它们都包含一些常用公开数据集如CIFAR, COCO等。以FashionMNIST（时装图像）为例

```
training_data = datasets.FashionMNIST(root="data", train=True,
    download=True, transform=ToTensor())
test_data = datasets.FashionMNIST(root="data", train=False,
    download=True, transform=ToTensor())
```

- 构建一个完整的dataloader

```
batch_size = 64
# 创建Dataloader，设定batch size
train_dataloader = DataLoader(training_data,
batch_size=batch_size)
test_dataloader = DataLoader(test_data,
batch_size=batch_size)

for X, y in test_dataloader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break
```

模型的创建

- 使用Pytorch创建一个神经网络：使用nn.Module库
  - 第一步：定义模型

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
      nn.Linear(28*28, 512), #网络层节点连接方式：线性连接
            nn.ReLU(), #激活函数选取为ReLU型
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)    )
    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

- 使用Pytorch创建一个神经网络：使用nn.Module库
  - 第二步：适配硬件

```
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
model = NeuralNetwork().to(device)
print(model) #打印神经网络模型信息
```

网络的训练

- 神经网络的训练是一个优化过程，需要定义网络的优化目标函数（损失函数Loss function），以及选择pytorch中提供的优化器（Optimizer）：

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
```

- 神经网络的一步训练，根据批处理采样数（batch size）进行梯度反向传播：

```
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
```

- （续左）def train()

```
for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)
        # 计算预测误差
        pred = model(X)  # 前向传播，得到预测结果
        loss = loss_fn(pred, y) #之前定义的损失函数的函数对象

        # Backpropagation
        optimizer.zero_grad() #清除之前的梯度（如果有）
        loss.backward() # 反向传播，计算梯度
        optimizer.step() # 使用优化器，更新模型的参数
        if batch % 100 == 0: #按条件打印到屏幕
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

网络的评估

- 神经网络的评估在测试集（test dataset）上进行，例如，统计平均损失和正确预测率：

```
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset) #计算test数据集有多少个采样
    num_batches = len(dataloader) #计算test数据集被分为多少个批次
    model.eval() # 切换到评估模式进行推理或测试，固定参数
    test_loss, correct = 0, 0
    with torch.no_grad(): #上下文管理器，确保不会计算梯度
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item() #计算损失函数和
            correct += (pred.argmax(1) == y).type(torch.float).sum().item() #统计正确预测/分类次数
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

- 训练迭代：训练过程在几个迭代中进行，一般而言，准确率将随着迭代的增加而提高，同时损失减少

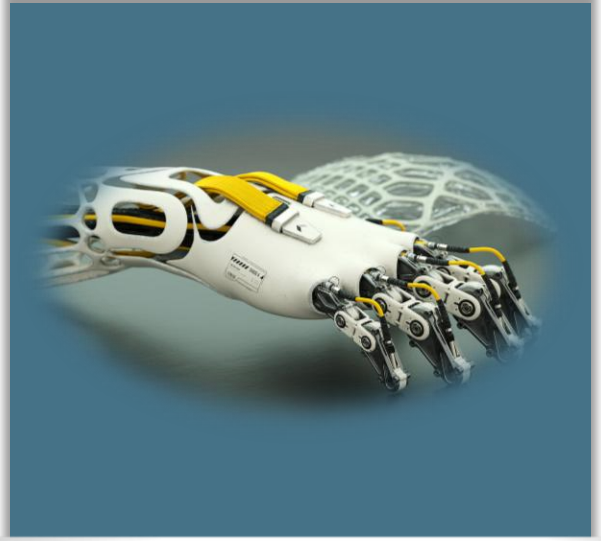
```
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Done!")
```

- 最后，保存训练好的模型（模型参数）

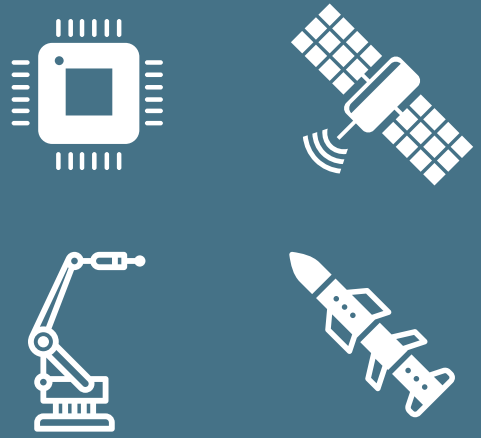
```
torch.save(model.state_dict(), "model.pth")
print("Saved PyTorch Model State to model.pth")
```

- 加载预训练的模型与保存相反

```
model = NeuralNetwork().to(device)
model.load_state_dict(torch.load("model.pth"))
```



卷积神经网络



附录：以Pytorch为例构建神经网络

附录.1 Pytorch简介

附录.2 Pytorch数据的组织

附录.3 神经网络的构建

Pytorch中数据的基本组织形式：Tensor

- Tensor可以看做是Pytorch对高维数组（一维为向量，二维为矩阵）的封装，其特性包括
  - Pytorch中tensor是一种数据对象的格式，与数学中的tensor定义不同。
  - tensor的操作基本兼容NumPy中的ndarrays，但是tensor操作可以由硬件加速。
  - tensor数据类型针对反向梯度传播（自动微分，Autograd）做了优化。
  - tensor和ndarrays之间需要显式的转换，如：

```
np_array = np.array(data)
x_np = torch.from_numpy(np_array)。
```

  - tensor的主要attributes有：shape，dtype，device

```
tensor = torch.rand(3,4)
print(f"Shape of tensor: {tensor.shape}") #Shape of tensor: torch.Size([3, 4])
print(f"Datatype of tensor: {tensor.dtype}") #Datatype of tensor: torch.float32
print(f"Device tensor is stored on: {tensor.device}") #Device tensor is stored on: cpu
```

Pytorch中样本集合的组织形式：Dataset

- Dataset是Pytorch对数据样本的封装
  - PyTorch 提供了两个数据封装类：torch.utils.data.DataLoader 和 torch.utils.data.Dataset。
  - Dataset 用于存储样本和它们对应的标签。
  - DataLoader 则将 Dataset 封装成可迭代的对象，以便于访问样本。
  - Pytorch提供了一些预加载的数据集（自动从网络加载），载入代码如：

```
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
training_data = datasets.FashionMNIST(  root="data",  train=True,  download=True,  transform=ToTensor())
```

构建用户自定义Dataset

- 用户自定义Dataset必须实现__init__, __len__, and __getitem__等三个函数，如：

```
import os
import pandas as pd
from torchvision.io import read_image


class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir,
                 transform=None,
target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform
```

- （续）：

```
def __len__(self): #得到数据集的大小
    return len(self.img_labels)


def __getitem__(self, idx): #加载并返回一个样本数据
    img_path = os.path.join(self.img_dir,
                self.img_labels.iloc[idx, 0])
    image = read_image(img_path)
    label = self.img_labels.iloc[idx, 1]
    if self.transform:
        image = self.transform(image)
    if self.target_transform:
        label = self.target_transform(label)
    return image, label
```

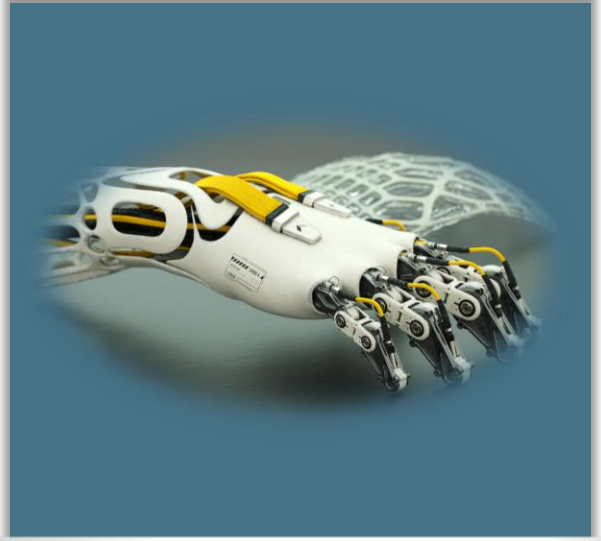
将Dataset封装成可迭代对象DataLoader

- Dataset和Dataloader的不同：
  - Dataset 一次得到数据集的一个样本的特征和标签。
  - DataLoader 是一个可迭代的对象，一次可以用小批量的方式（batch）传递样本。
  - 范例程序

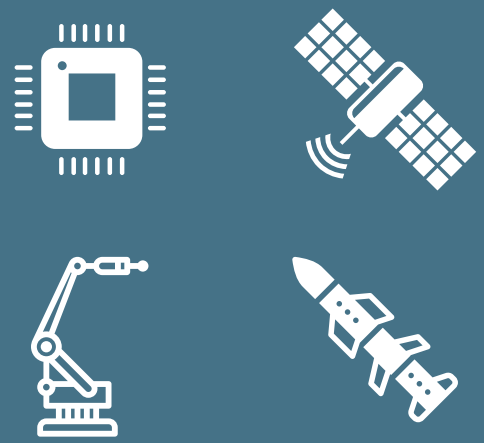
```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data,
batch_size=64, shuffle=True)

test_dataloader = DataLoader(test_data,
batch_size=64, shuffle=True)
```

- 使用DataLoader的迭代器：
  - 给定batch size=64后，每次迭代都会返回同样大小批次的训练特征和训练标签（分别包含 64 个特征和标签）

```
# 显式图像和标签

train_features, train_labels =
next(iter(train_dataloader))

print(f"Feature batch shape: {train_features.size()}")

print(f"Labels batch shape: {train_labels.size()}")

img = train_features[0].squeeze()

label = train_labels[0]

plt.imshow(img, cmap="gray")

plt.show()

print(f"Label: {label}")
```



卷积神经网络



附录：以Pytorch为例构建神经网络

附录.1 Pytorch简介

附录.2 Pytorch数据的组织

附录.3 神经网络的构建

神经网络的构建：模型类的定义

- Pytorch中所有神经网络类都继承自 nn.Module：
  - 神经网络的拓扑结构在__init__函数中构造。
  - 每个 nn.Module 子类都提供 forward 方法，实现对输入数据的操作。
  - 范例程序：使用Sequential方法构建网络

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten() #将2D矩阵（图像）输入展平为1D向量
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),  nn.ReLU(), nn.Linear(512, 512), nn.ReLU(),
            nn.Linear(512, 10),  )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

- 神经网络对象的实体化和使用：
  - 实例化神经网络：

```
model = NeuralNetwork().to(device)
print(model)
```

  - 输入数据直接传递给神经网络对象而不是直接model.forward()

```
X = torch.rand(1, 28, 28, device=device)
logits = model(X)
pred_probab = nn.Softmax(dim=1)(logits)
y_pred = pred_probab.argmax(1)
print(f"Predicted class: {y_pred}")
```

神经网络的训练：使用优化器和反向传播

- 超参数的选择：
  - 超参数是可调整的参数，用于控制模型优化过程。不同的超参数值可以影响模型训练和收敛速度。
  - 常用超参数：
    - 周期数 （Number of Epochs）- 遍历数据集的次数。
    - 批次大小（Batch Size）：在更新参数之前，通过网络反向梯度传播的数据样本数量。
    - 学习率 （Learning Rate）：在每个批次/周期中更新模型参数的程度。较小的值会导致学习速度慢，而较大的值可能会在训练过程中导致不可预测的行为。

- 范例程序

```
learning_rate = 1e-3

batch_size = 64

epochs = 5
```

- 网络训练：选择Loss Function，优化器，并迭代

```
loss_fn = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

神经网络的训练：使用优化器和反向传播（续）

- 定义函数train_loop()：

```
def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    # 显式地将网络设置为训练模式
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)
```

- (续左) def train_loop():

```
        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * batch_size + len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```