卷积神经网络

**卷积神经网络**（convolutional neural network，CNN）是一类强大的、为处理图像数据而设计的神经网络，主要用于图像处理、计算机视觉和自然语言处理等任务。以下是对卷积神经网络的详细介绍：

1. 基本结构

卷积神经网络通常由以下几种层组成：

* **卷积层（Convolutional Layer）**：通过卷积操作提取特征，使用多个滤波器（卷积核）对输入数据进行处理。每个滤波器可以捕捉不同的特征（如边缘、纹理等）。
* **激活层（Activation Layer）**：通常使用ReLU（Rectified Linear Unit）函数，增加模型的非线性。
* **池化层（Pooling Layer）**：通过下采样减少特征图的尺寸，常用的有最大池化（Max Pooling）和平均池化（Average Pooling）。这有助于降低计算复杂度和防止过拟合。
* **全连接层（Fully Connected Layer）**：在网络的最后阶段，将特征映射转换为分类结果，类似于传统神经网络。

2. 工作原理

卷积神经网络的工作流程如下：

1. **输入层**：输入图像被转化为一个多维数组（张量）。
2. **卷积操作**：通过卷积层中的滤波器对输入图像进行卷积，生成特征图（Feature Map）。
3. **激活函数**：对特征图应用激活函数，增强模型的表达能力。
4. **池化操作**：通过池化层减少特征图的维度，保留重要特征。
5. **重复**：以上过程可以重复多次，以提取更高层次的特征。
6. **全连接层**：将经过多层卷积和池化后的特征图展平，输入到全连接层，进行最终分类。

3. 优势（特征）

* **局部感知**：卷积操作只关注局部区域，减少了参数数量，提高了计算效率。
* **权重共享**：同一卷积核在整张图像上滑动，减少了模型的复杂度。
* **平移不变性**：CNN对图像的平移具有一定的鲁棒性，能够识别不同位置的特征。（**不管检测对象出现在图像中的哪个位置，神经网络的前面几层，应该对相同的图像区域具有相似的反应，即为“平移不变性”。**）
* **局部性**（locality）：神经网络的前面几层应该只探索输入图像中的局部区域，而不过度在意图像中相隔较远区域的关系，这就是“局部性”原则。最终，可以聚合这些局部特征，以在整个图像级别进行预测。

4. 应用

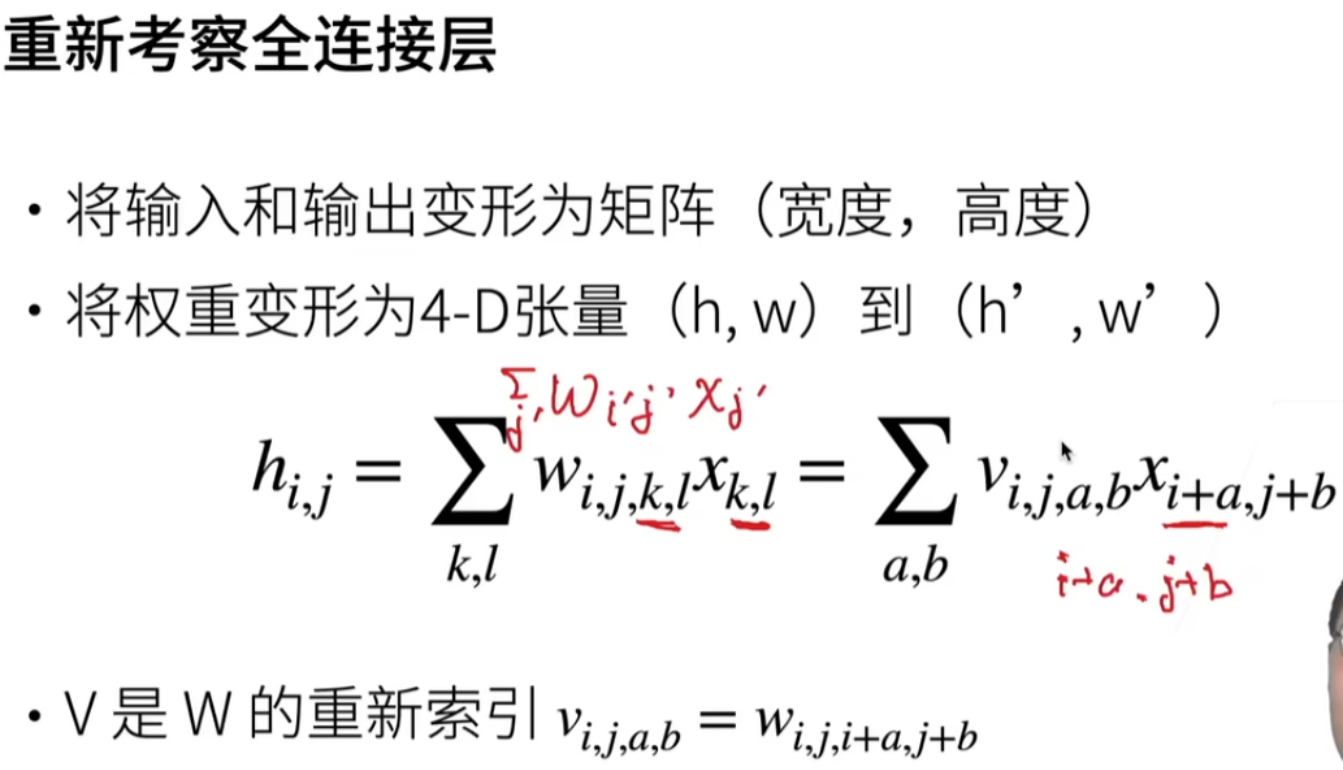
* **图像分类**：如图片识别、物体检测等。
* **语义分割**：将图像中的每个像素标记为特定类。
* **目标检测**：识别图像中的特定对象并标记其位置。
* **视频分析**：处理视频帧以进行动作识别等任务。

卷积神经网络是包含卷积层的一类特殊的神经网络。在深度学习研究社区中，V被称为卷积核（convolution kernel）或者滤波器（filter），亦或简单地称之为该卷积层的权重，通常该权重是可学习的参数。当图像处理的局部区域很小时，卷积神经网络与多层感知机的训练差异可能是巨大的：以前，多层感知机可能需要数十亿个参数来表示网络中的一层，而现在卷积神经网络通常只需要几百个参数，而且不需要改变输入或隐藏表示的维数。**参数大幅减少的代价是，我们的特**

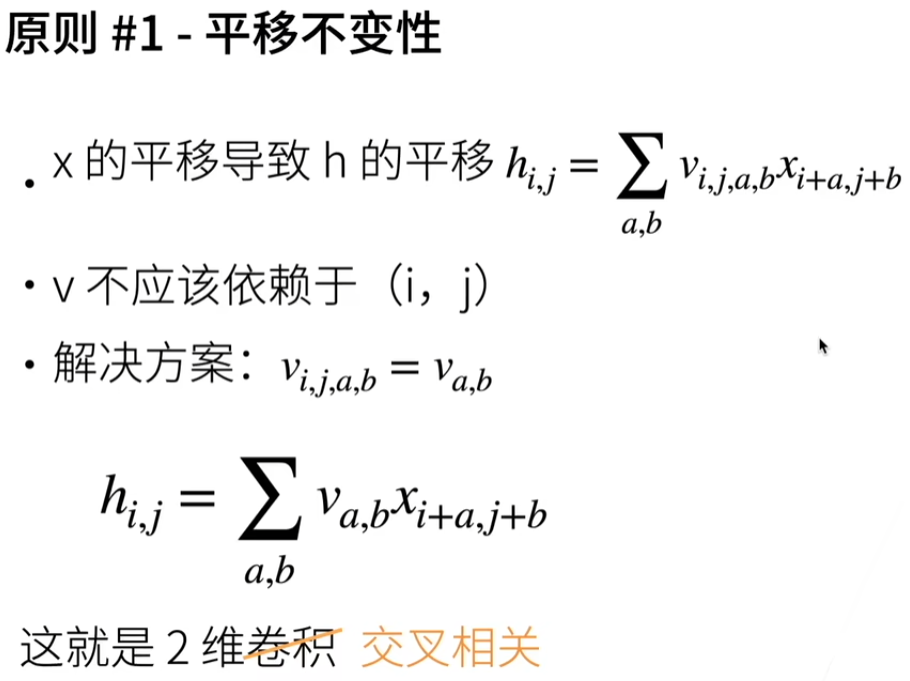
**征现在是平移不变的，并且当确定每个隐藏活性值时，每一层只包含局部的信息。**以上所有的权重学习都将依赖于归纳偏置。当这种偏置与现实相符时，我们就能得到样本有效的模型，并且这些模型能很好地泛化到未知数据中。但如果这偏置与现实不符时，比如当图像不满足平移不变时，我们的模型可能难以拟合我们的

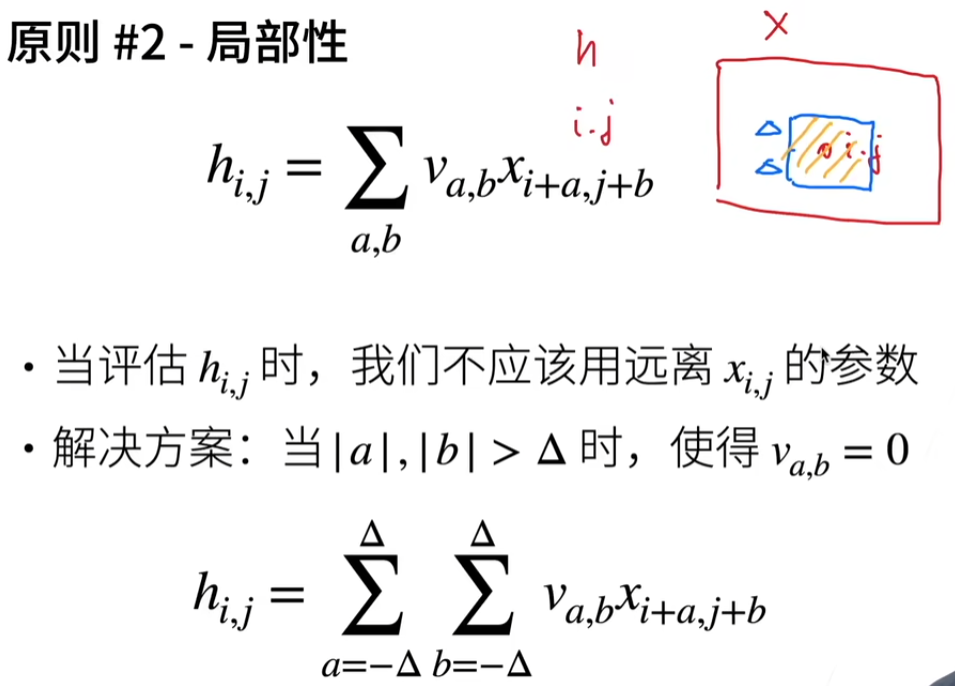
训练数据。

一、卷积层

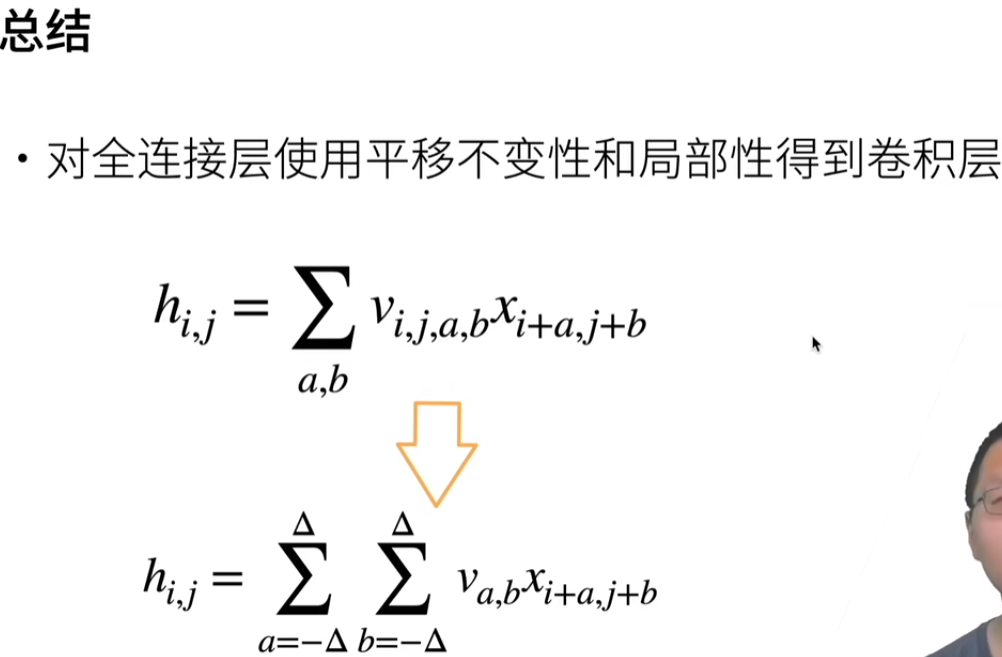


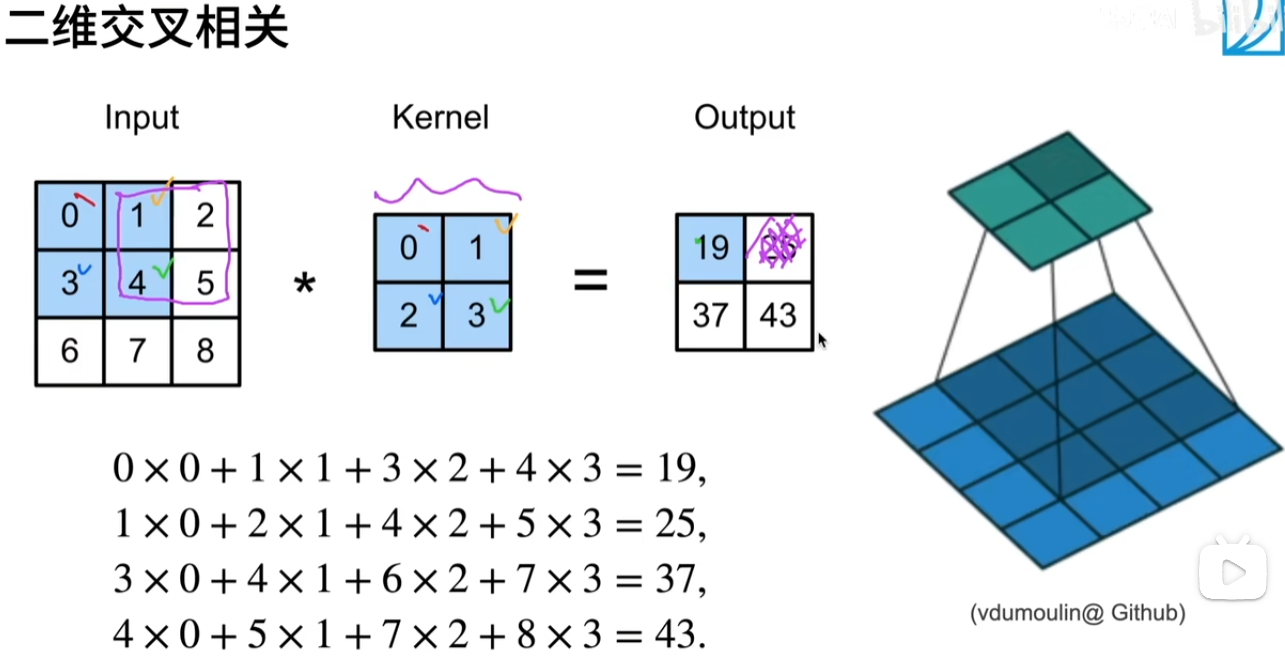
考虑到空间信息，必须用矩阵。  
二维到四维

对下标做变换引出卷积。v不发生变化，不论识别器移到哪里位置i j，识别器都是不变的。平移不变性，对权重做了限制，不是每个元素都可以自由变换，降低模型复杂度，不用存储太多元素。干掉 i j 维度。

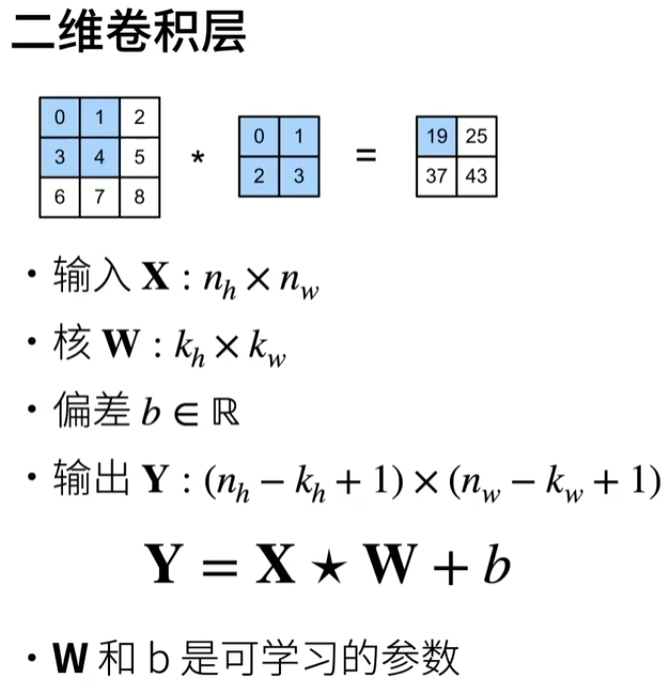


不看远离输入x i , j太远的元素，只看附近的。超过delta位置的元素就不看了。

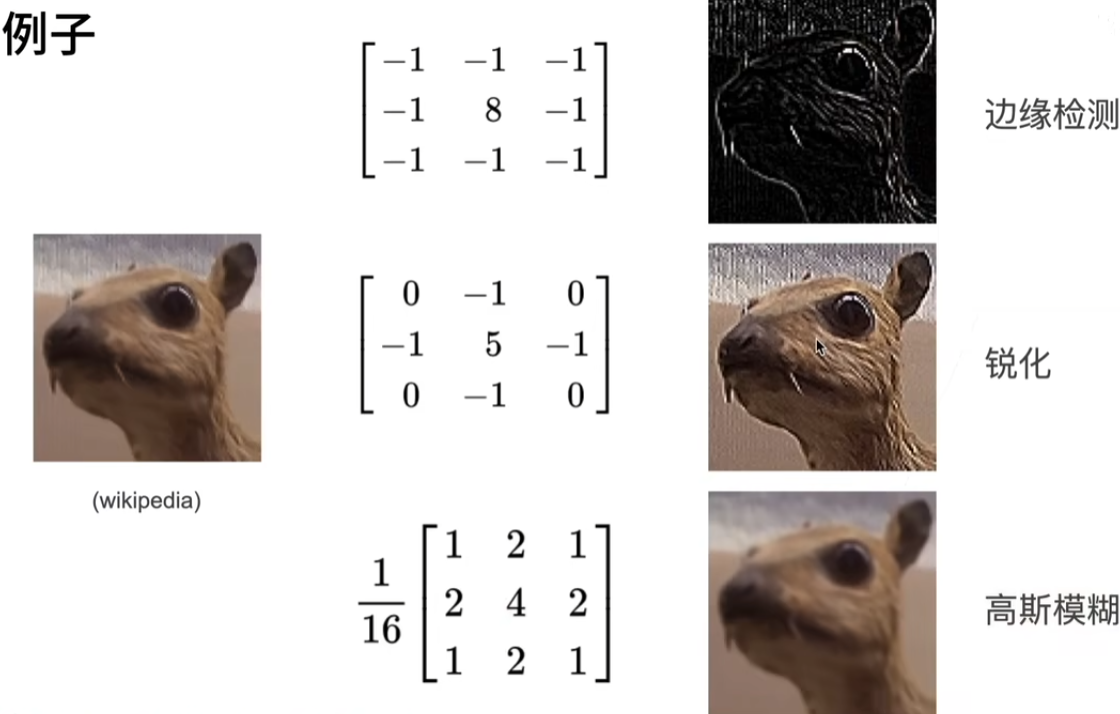




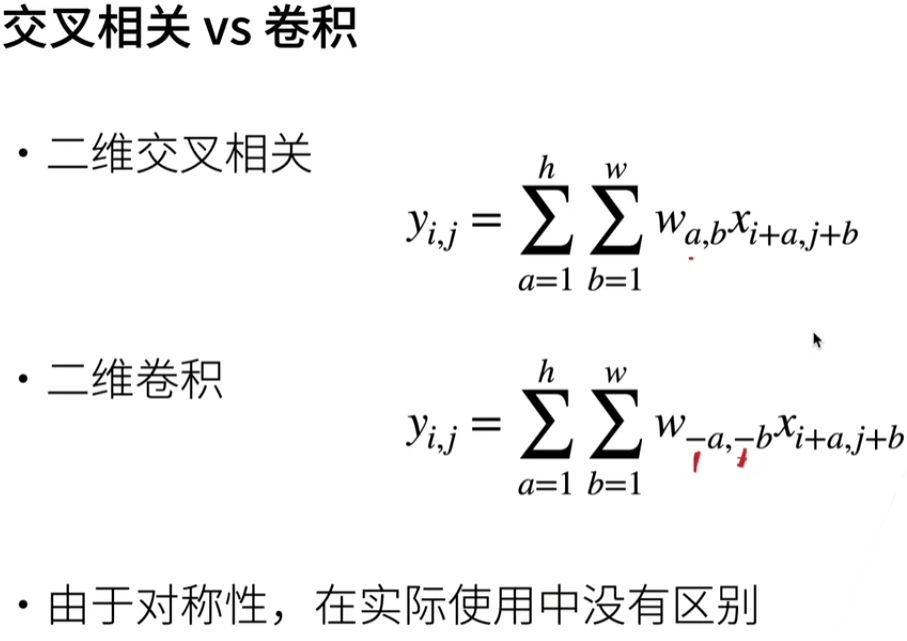
**平移不变性**，kernel（卷积核）是不变的。输出看局部，局部和kernel对应位置计算。kernel窗口不断的往右移往下移，不断计算。



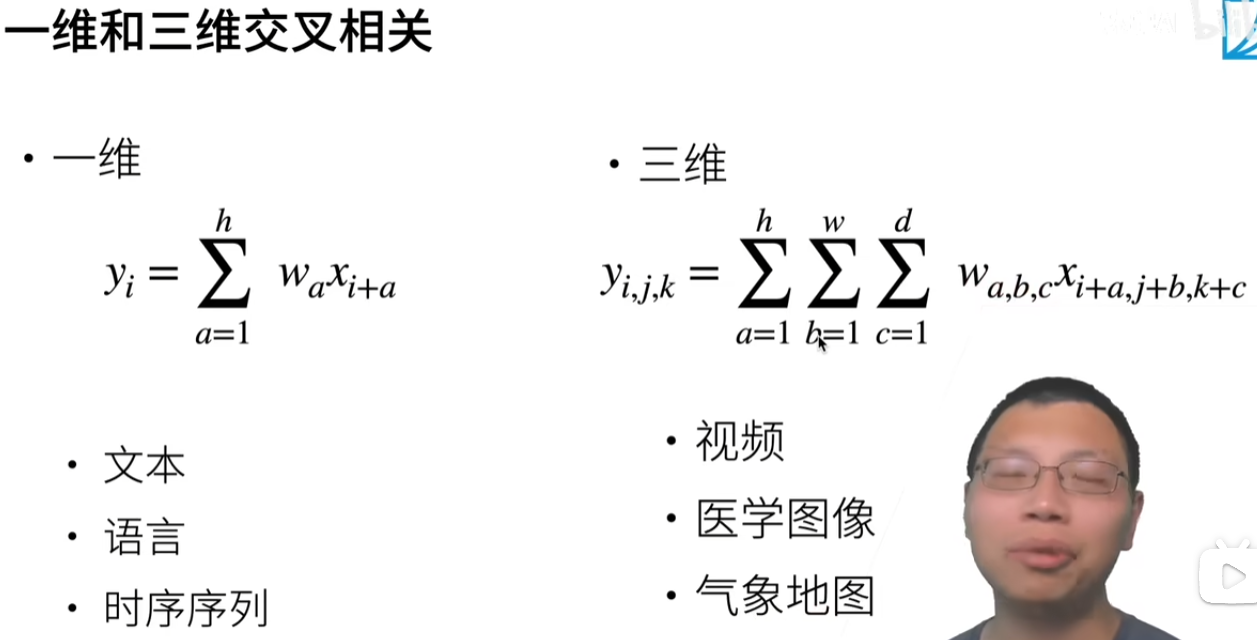
当kernel移动到边界，没有数据了就不再移动，这样就会丢掉一些数据（ k h − 1 , k w − 1 ） **星号**：表示二维交叉相关计算操作子



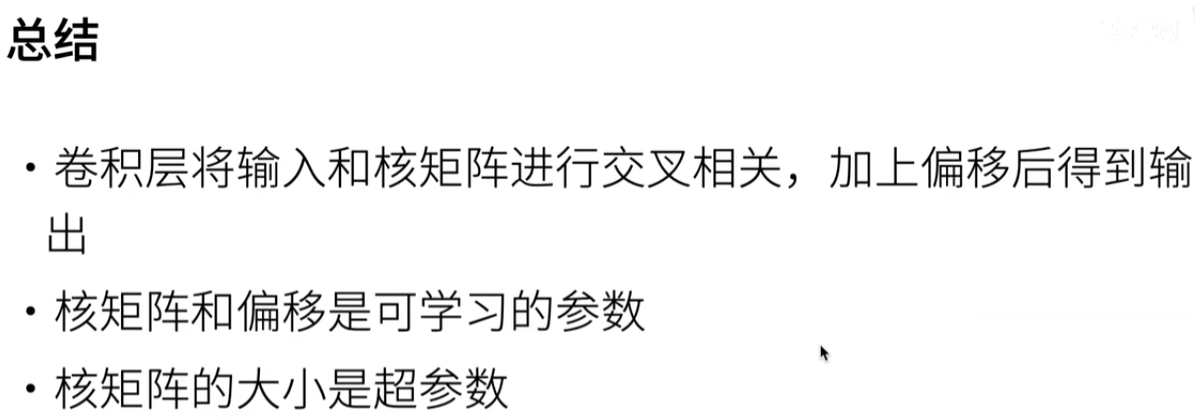
不同的卷积核会带来不同的效果。神经网络可以学习一些核，达到我们想要的效果。



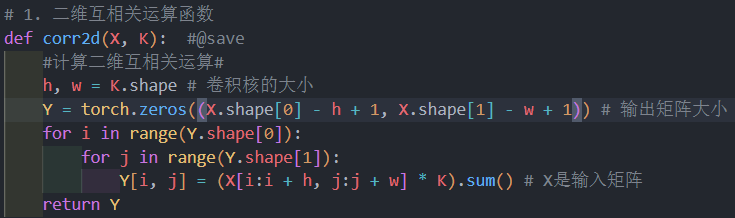
交叉相关和卷积没有太大区别，区别是卷积在索引处有负号，在索引w的时候是反过来走的，所以是负号。又因为是对称的，在实际使用中正反没有区别，学到的w正反，左右上下反一下是一样的东西。说是卷积层，实际计算实现的是交叉相关。

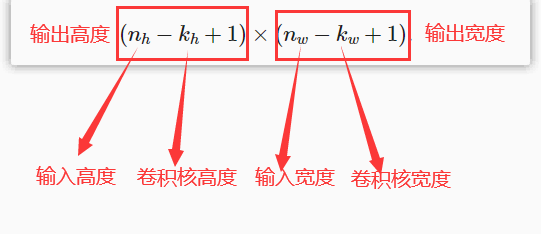
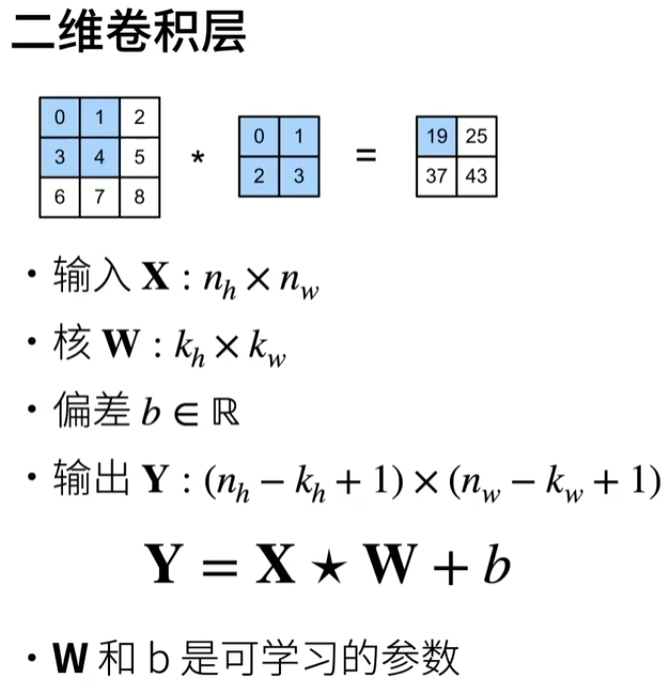


kernel卷积核大小控制的是局部性，大看的局部多，小看的局部少。  
卷积层是个特殊的全连接层。卷积核不会随着输入数据大小而改变，可以固定。



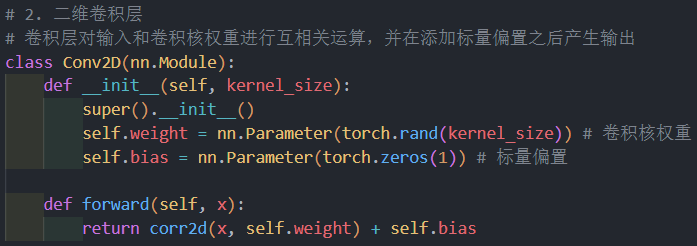
代码：





这段代码原理如上图所示，实际上丢失了一些数据。

**Y[i, j] = (X[i:i + h, j:j + w] \* K).sum()** ：取出输入矩阵里的小方块与核矩阵做点积然后求和，得到输出矩阵Y



定义卷积层Conv2D：（继承nn.Module实现的一个类）

1. 卷积层对输入和卷积核权重进行互相关运算；
2. 并在添加标量偏置之后产生输出。

**kernel\_size**：卷积核的大小，通常是一个元组，表示卷积核的宽度和高度。

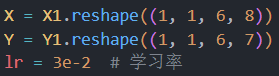
* **self.weight**：卷积核的权重，使用 nn.Parameter 定义，这意味着这些权重是模型的可训练参数。初始值是随机生成的。
* **self.bias**：卷积层的偏置，同样使用 nn.Parameter 定义，初始值是全零。偏置也是模型的可训练参数。

**corr2d(x, self.weight) + self.bias**：调用 corr2d 函数进行二维互相关运算，并加上偏置 self.bias，这就是前向传播运算

简单应用

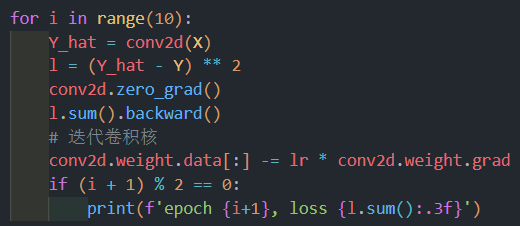
 白变黑 黑变白 第一列第二列数值相同就相加为0，不同则等于正1或者负1.

**conv2d = nn.Conv2d(1, 1, kernel\_size=(1, 2), bias=False)**：这行代码创建了一个二维卷积层，它有一个输入通道（1）（黑白），一个输出通道（1）（黑白），卷积核的大小为（1, 2），并且没有偏置项。卷积核将沿着宽度方向进行卷积操作。只能检查纵向，不能检查横向。1\*2只能检测垂直的。



X1和Y1是原始的输入和输出数据。通过reshape函数，它们被转换为四维张量，其中：第一个维度是批量大小（batch size）。第二个维度是通道数（channels）。

第三个和第四个维度是高度和宽度。



**Y\_hat = conv2d(X)**：使用卷积层对输入X进行卷积操作，得到**预测输出Y\_hat**。

**l = (Y\_hat - Y) \*\* 2**：计算预测输出和真实输出之间的均方误差（MSE）。

**conv2d.zero\_grad()**：清空梯度。

**l.sum().backward()**：后向传播计算损失函数的梯度。

**conv2d.weight.data[:] -= lr \* conv2d.weight.grad**：根据梯度下降算法更新卷积核的权重。（手写梯度下降）



**特征映射和感受野**

1. 输出的卷积层有时被称为特征映射。
2. 在卷积神经网络中，对于某一层的任意元素x，其感受野是指在前向传播期间可能影响x计算的所有元素。

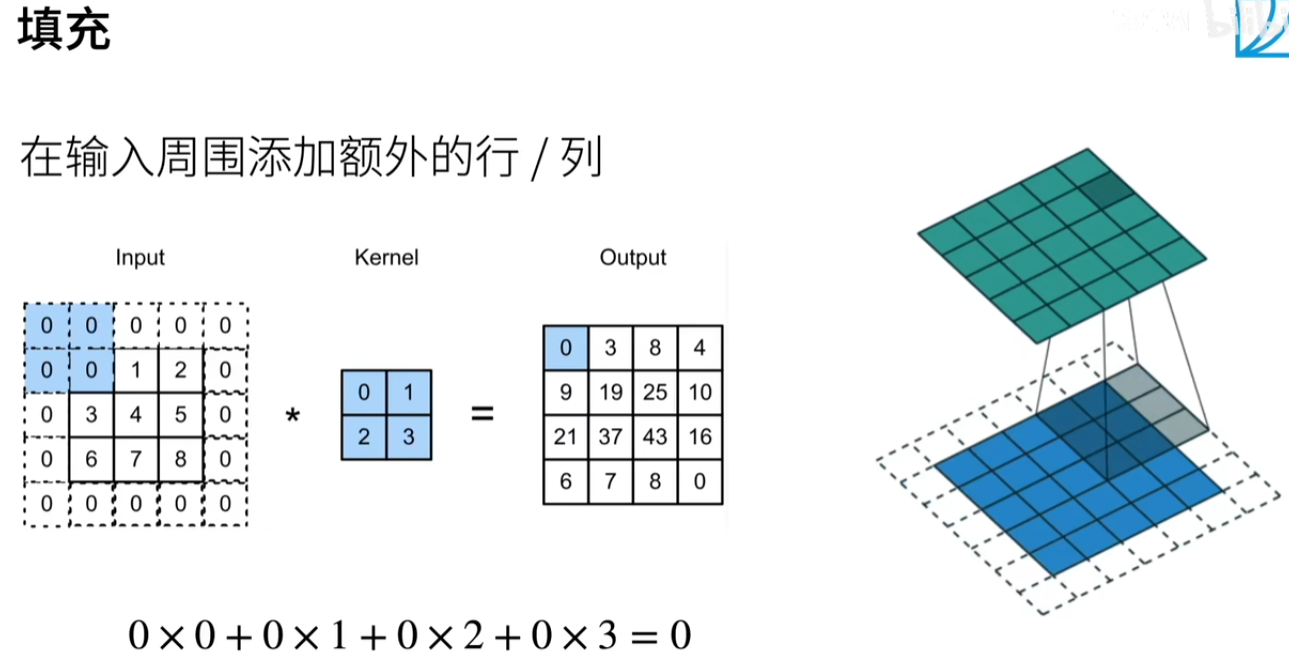
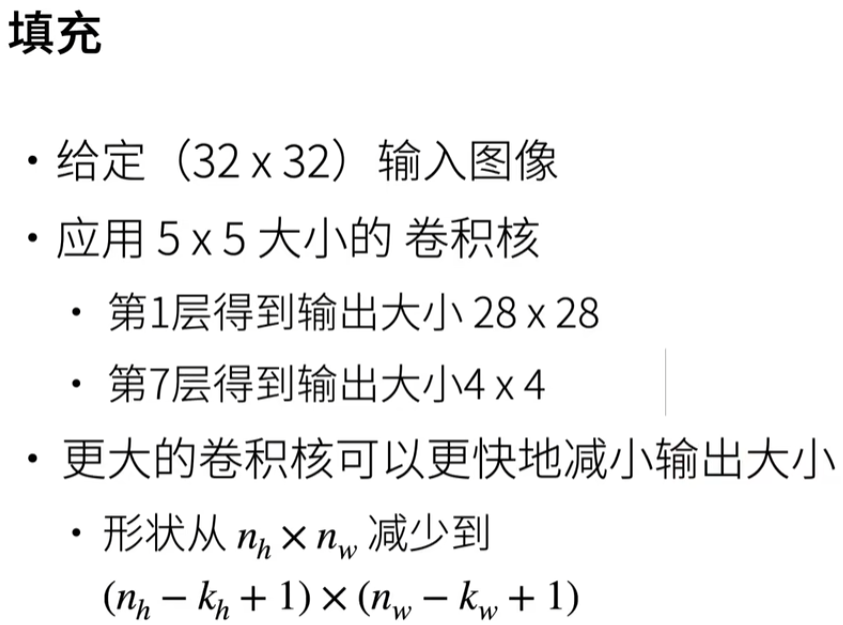
二、填充和步幅

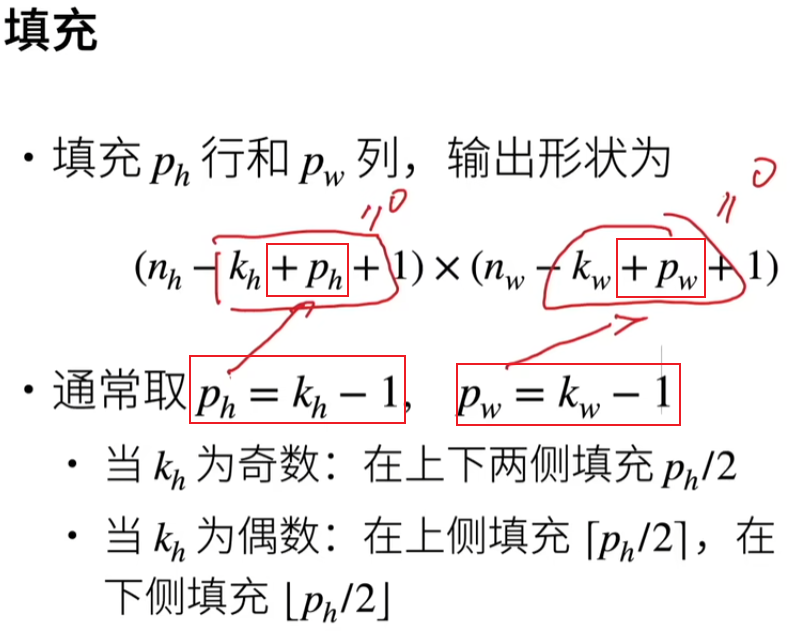
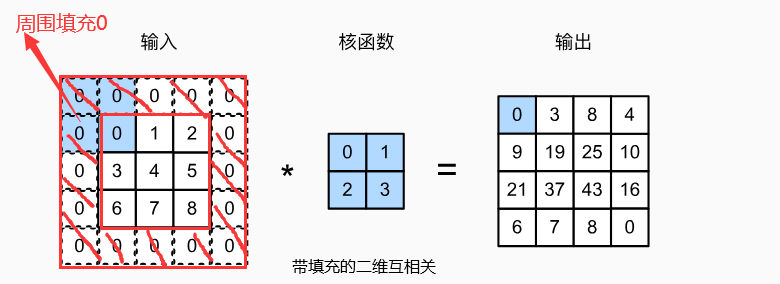
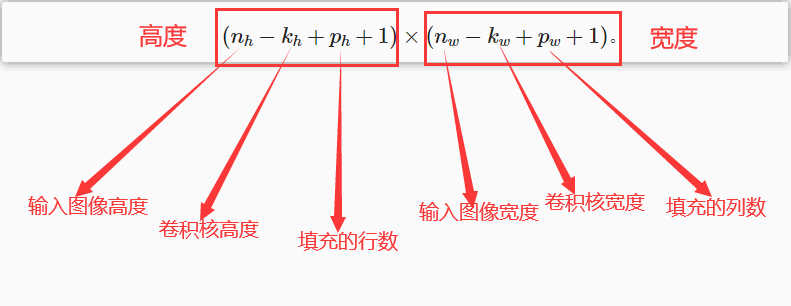
**两个控制输出大小的超参数**

问题一：应用了连续卷积，最终得到的输出远小于输入大小，使得原始图像的边界丢失了许多有用信息，我们希望输入大小和输出大小相同？

解决：**填充**（padding）：在输入图像的边界填充元素（通常填充元素是0）

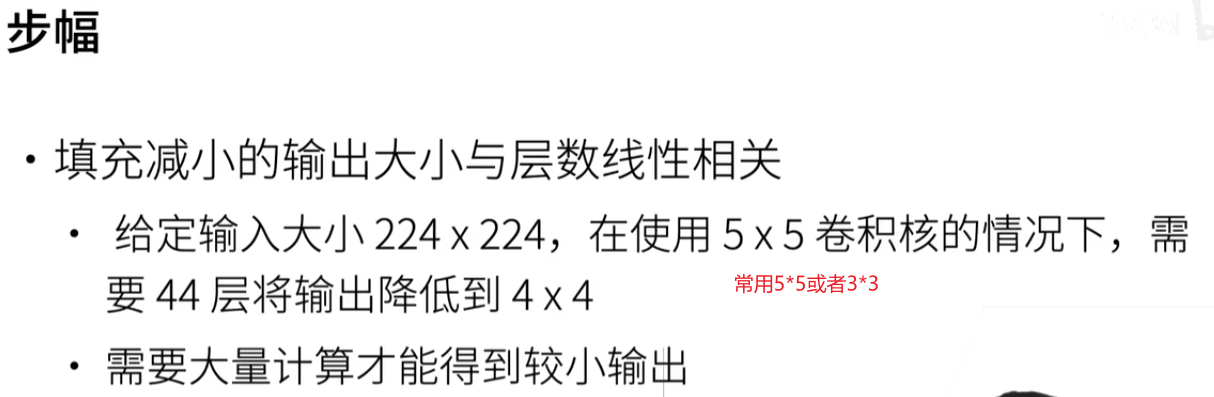
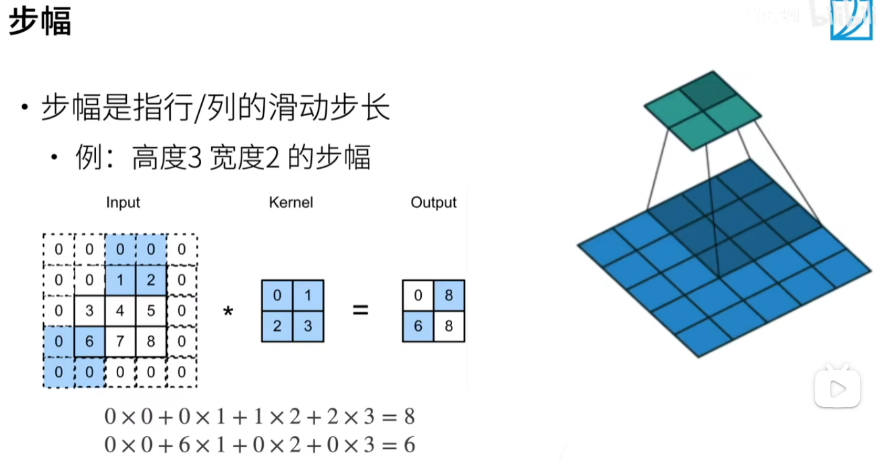
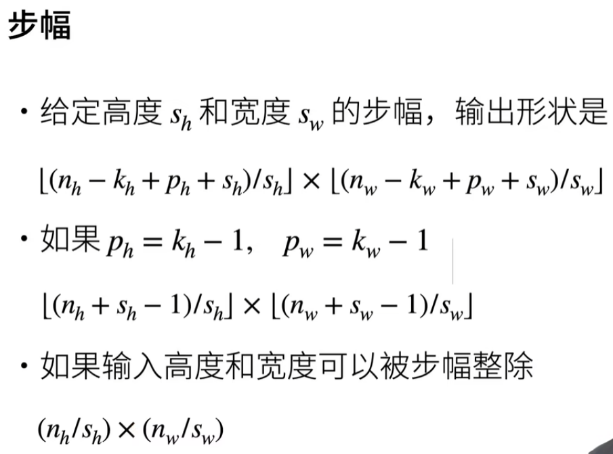
（当不想模型模型改变输入的大小，或者想要训练深层网络时使用。）

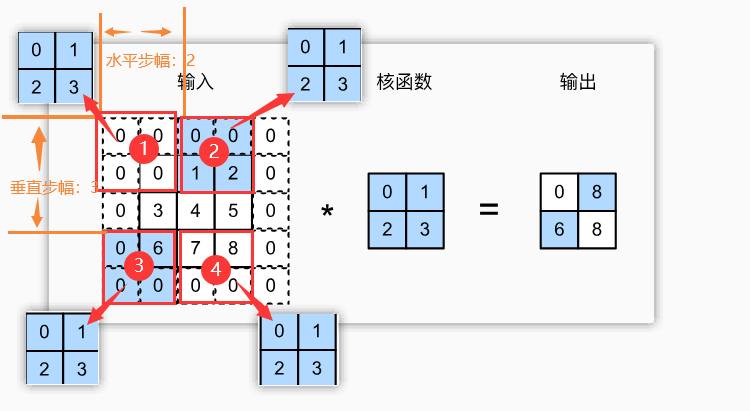




问题二：有时原始的输入分辨率十分冗余，我们可能希望大幅降低图像的宽度和高度？

解决：**步幅**（stride）：每次滑动元素的数量，设置卷积核滑动的步幅来减少采样次数。控制滑动窗口挪动行列的步长，成倍减少输入的形状。当输入shape过大，可以用步幅成倍缩小输出shape，减少计算量。

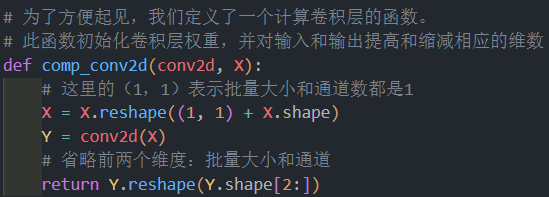




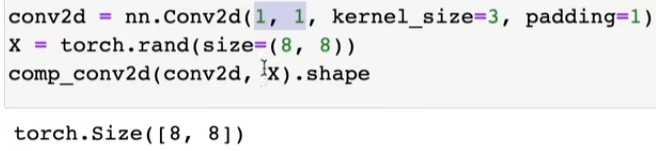
问题三：卷积核为什么一般选择**奇数**？

解决：保持空间维度的同时，我们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。

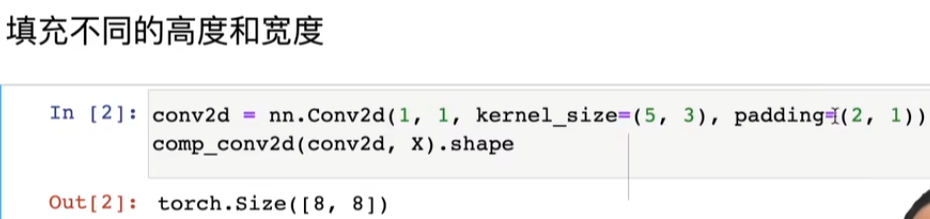
代码

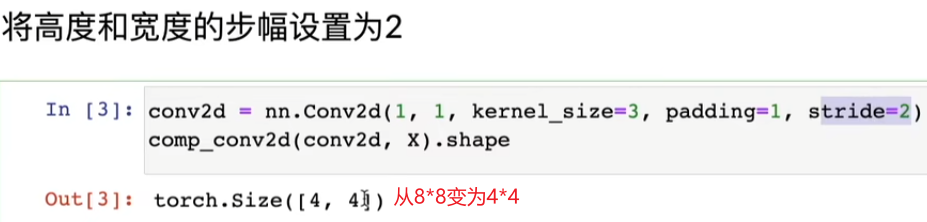


X首先被重塑为(1, 1) + X.shape，其中(1, 1)表示批量大小和通道数都是1。方便后续的卷积操作。conv2d(X)执行卷积操作，得到输出张量Y。只看后面两个维度，重点体现输出矩阵的大小



这里每边都填充了1行或1列，因此总共添加了2行或2列



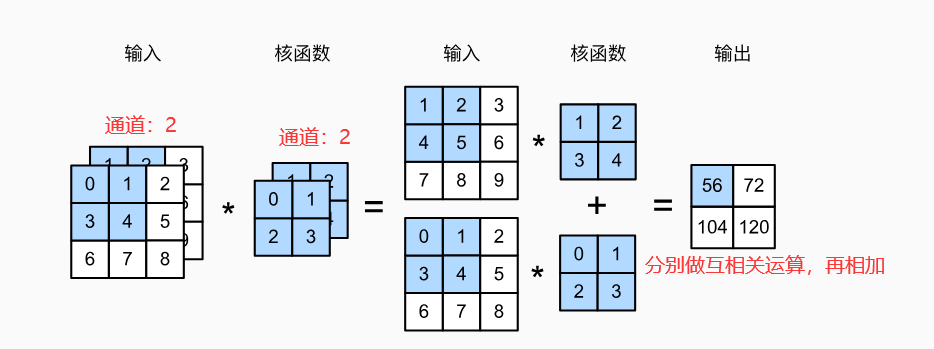


将高度和宽度的步幅设置为2，从而将输入的高度和宽度减半：(8 + 2 - 3) / 2 = 4

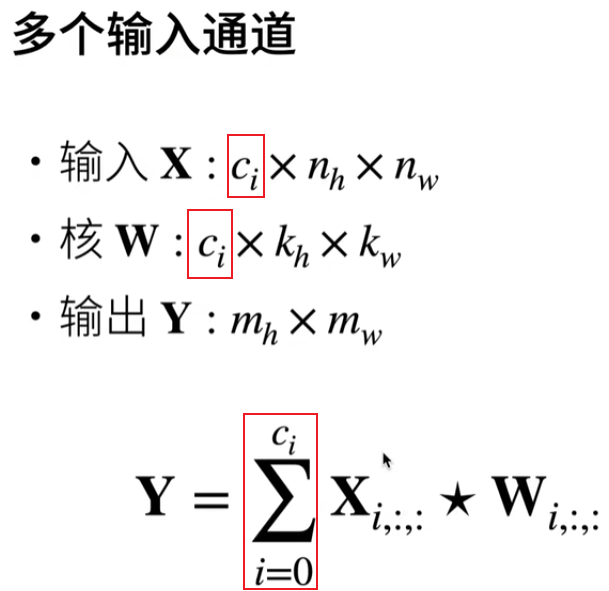
三、多输入多输出通道

1.多输入

当输入包含多个通道时，需要构造一个与输入数据具有**相同输入通道数**的卷积核，以便与输入数据进行互相关运算。

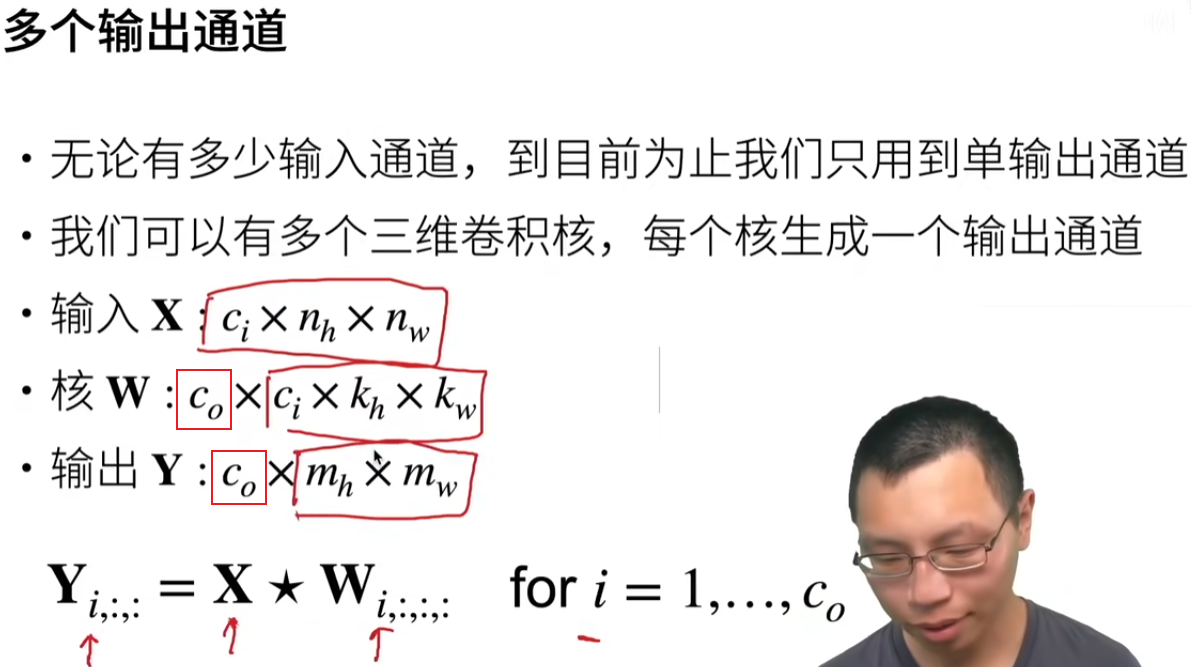






2.多输出

1. 在最流行的神经网络架构中，随着神经网络层数的加深，我们常会增加输出通道的维数，通过减少空间分辨率以获得更大的通道深度。
2. 将每个通道看作对不同特征的响应。

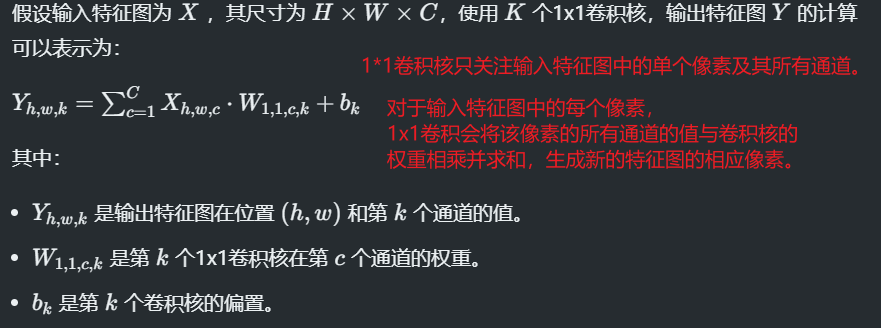
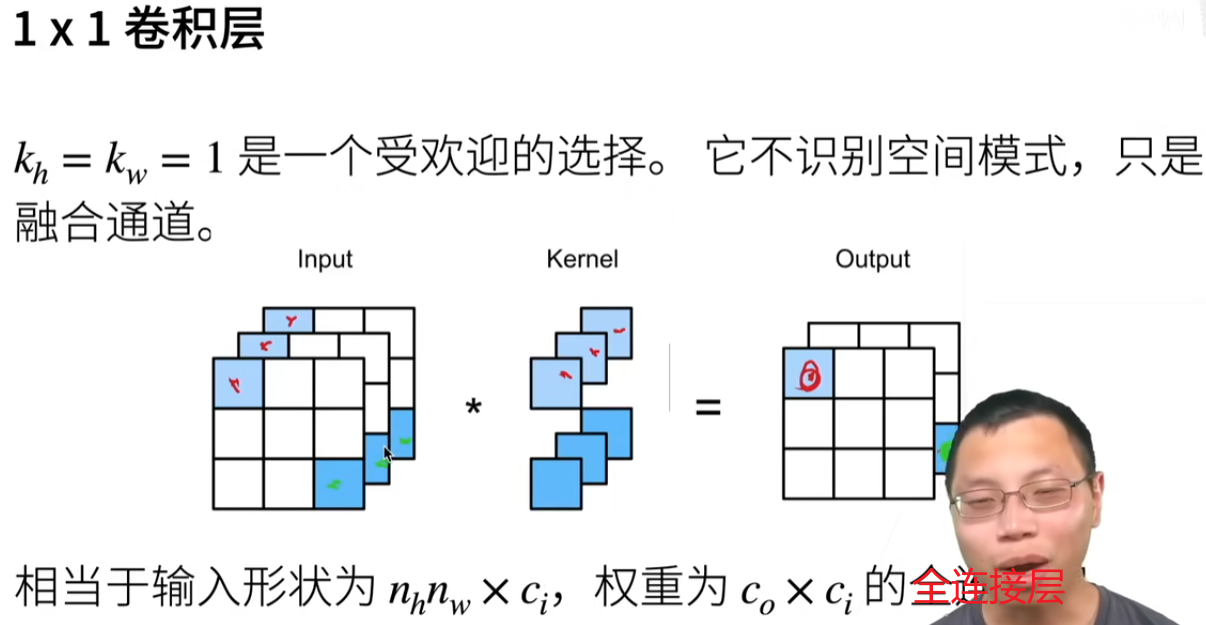


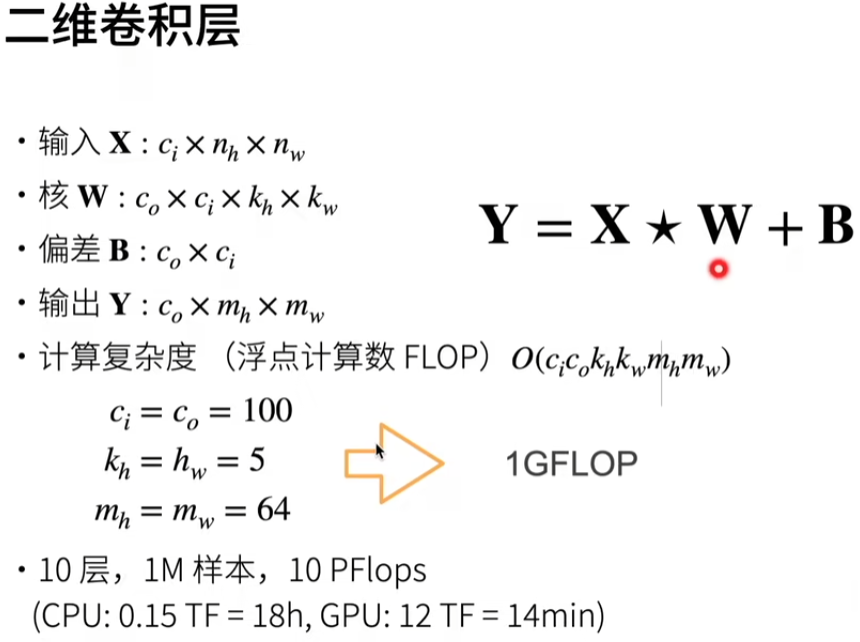


多输出通道，每个通道输出不同的模式（纹理），这个输出通道丢给下一层做多输入通道，每个纹理做按权重相加，组合的模式识别。上层组合局部纹理，所有纹理组合起来–所有层加起来，最后能识别出一个猫。所以这就是为什么要有多输入多输出通道。

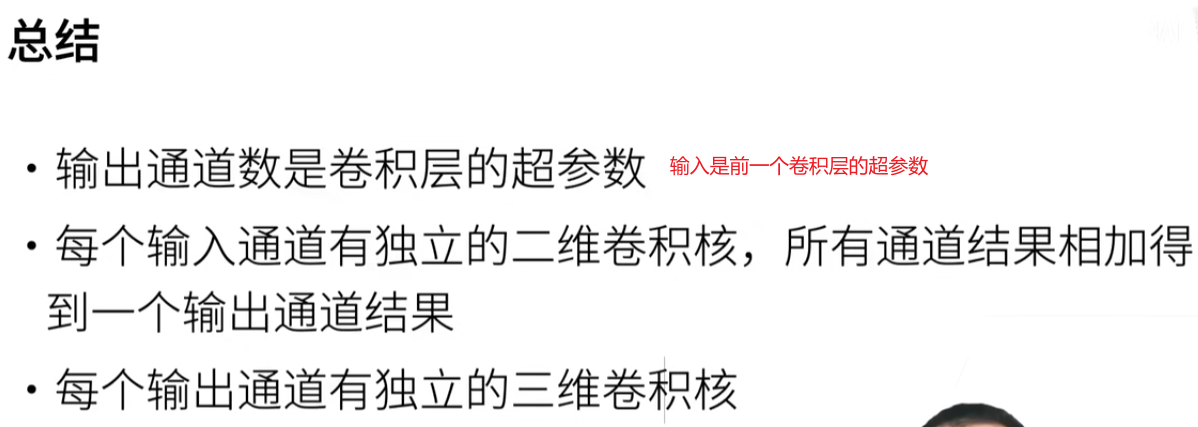
3.1×1卷积核

1×1卷积核被经常用来改变通道，相当于全连接层。可以对输入和输出的形状进行调整





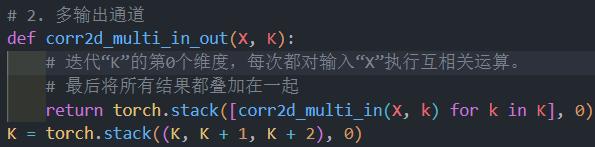
1Gflop 10亿个浮点数运算



代码

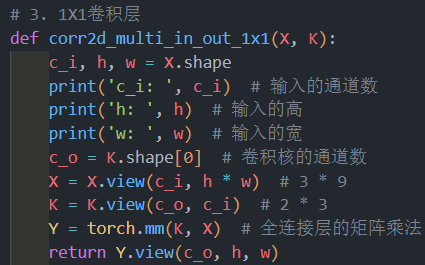


X是一个四维张量，表示多个输入通道的图像，K是一个三维张量，表示多个卷积核。函数通过遍历X和K的第0个维度（通道维度），对每个通道执行互相关运算，并将结果相加。



X是一个四维张量，表示多个输入通道的图像，K是一个四维张量，表示多个卷积核。函数通过遍历K的第0个维度，对每个卷积核执行互相关运算，并将结果堆叠在一起。

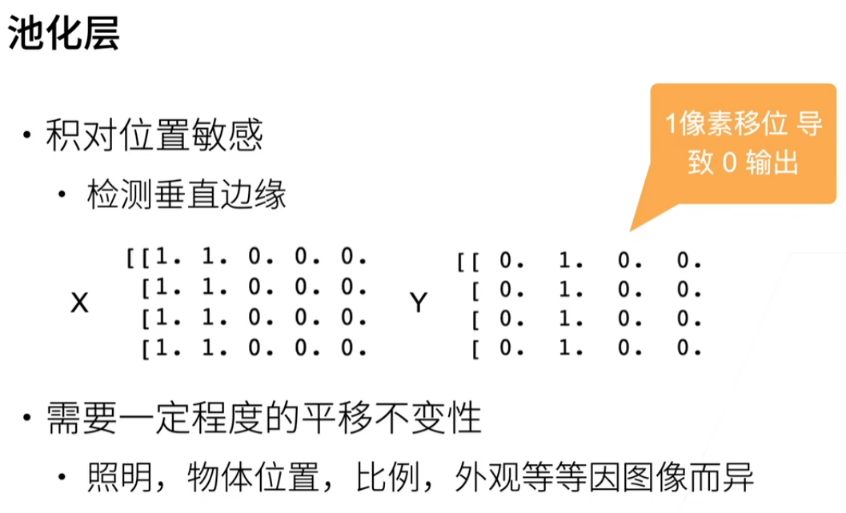
torch.stack()是PyTorch中的一个函数，用于沿新的维度连接一系列张量。这个操作与torch.cat()类似，但torch.cat()是在已有维度上进行连接，而torch.stack()是在新的维度上进行堆叠。可以通过指定dim参数来控制新的维度的位置。通过调整dim参数，你可以灵活地控制新维度的插入位置，从而改变结果张量的形状。



corr2d\_multi\_in\_out\_1x1实现了1x1卷积层。它接受两个参数：X和K，其中X是一个四维张量，表示多个输入通道的图像，K是一个四维张量，表示多个卷积核。函数首先将X和K的形状调整为二维张量，然后执行矩阵乘法，最后将结果调整回原来的形状。

1x1卷积层是一种特殊的卷积操作，它可以在不改变图像空间维度的情况下，通过改变通道数来提取特征。

四. 池化层

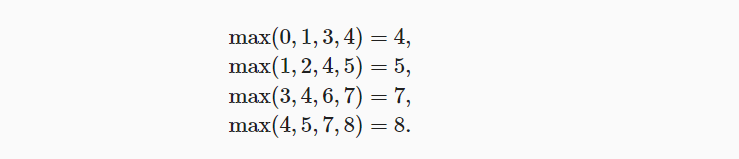
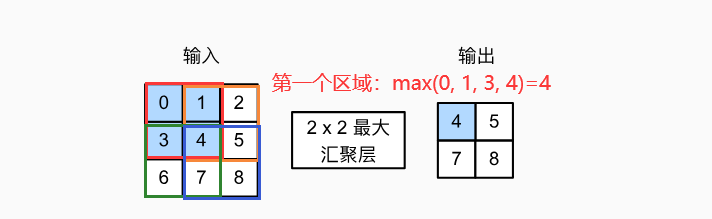


池化的原理在于通过固定大小的窗口对特征图进行下采样，聚合窗口内的值，从而实现降维、特征抽象和增强模型的鲁棒性。

池化层（pooling）（也叫做汇聚层）：

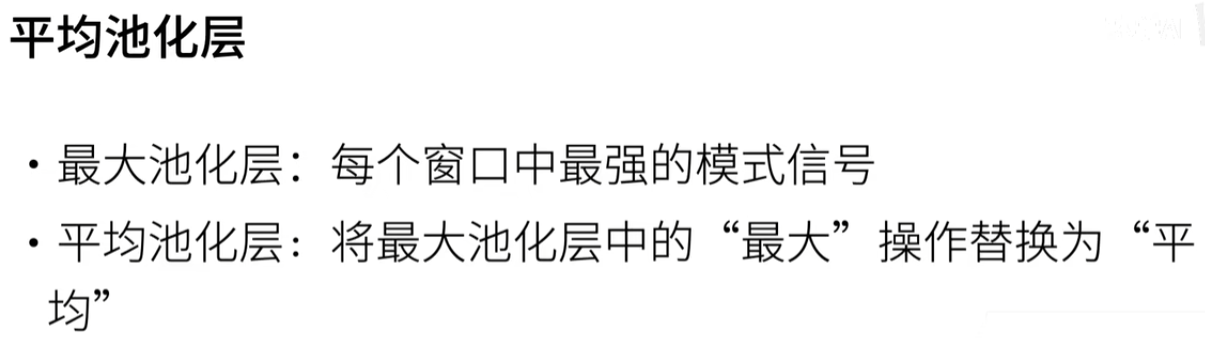
1. 降低卷积层对位置的敏感性
2. 降低对空间降采样表示的敏感性

**二维最大池化**

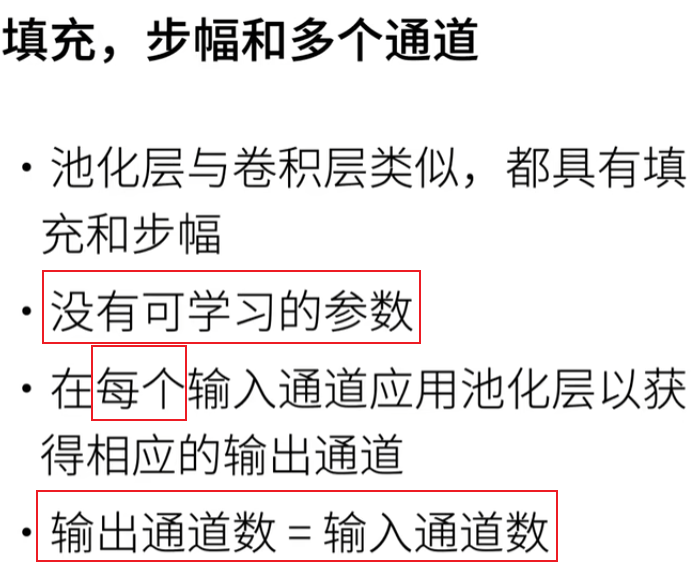


最大池化层和平均池化层

池化层与卷积层的原理大体相似，只不过把互相关运算换成求最大值或者求平均值

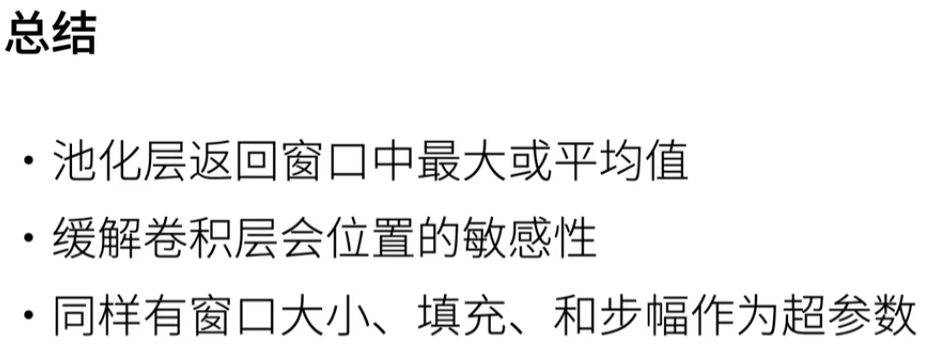


填充、步幅核和多个通道

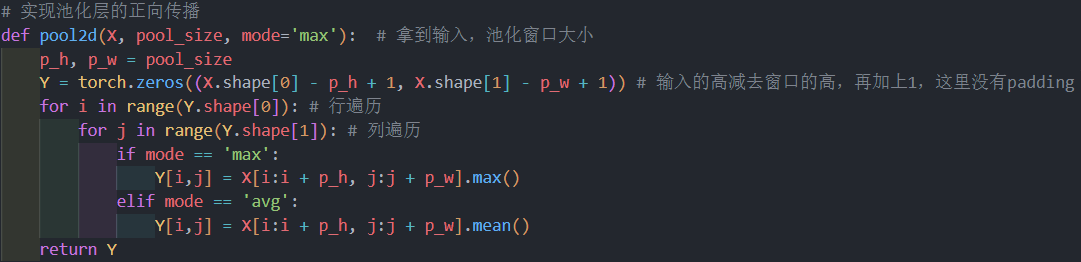


与卷积层一样，池化层也可以改变输出形状

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



代码



实现一个二维池化层，支持最大池化和平均池化。（与之前的卷积层差不多）

**X**：输入张量，形状为(H, W)。

**pool\_size**：池化窗口大小，形状为(p\_h, p\_w)。

**mode**：池化模式，max表示最大池化，avg表示平均池化。

通过遍历输入张量，在每个窗口内计算最大值或平均值，生成输出张量。







