Convolutional Networks

卷积神经网络是神经网络的一种架构，在图像、语言、语音等领域很常见。

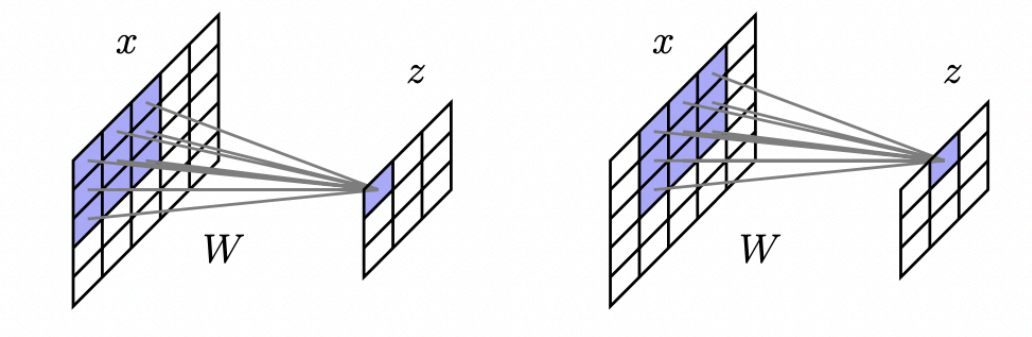
FCN 的问题在于将输入整体看作一个向量，在输入较大图像时，输入维度大，所需要的参数量随之增大，计算量和内存量无法被满足。

同时，每次采样都考虑整个图像，无法得到直观上的不变性。（例如，将图像向右移动一个像素点会导致下一层数据非常不同）

Convolutional Operator

卷积主要有两个特点：

1. 层与层之间的 activation 采用局部视野，并将 hidden layer 视为空间图像。
2. 在生成 hidden layer 时共享权重参数。



优点：

- 极大地减少参数数量
- 能捕捉到直观上的不变性（例如，向右移动一个像素后生成的 hidden layer 也相应地移动一个像素）

卷积实际上就是在图像上移动一个 $k \times k$ 的权重矩阵来生成新图像，写作 $y = z * w$

权重矩阵也称为过滤器 (filter), 其中 k 称为 kernel size.

z_{11}	z_{12}	z_{13}	z_{14}	z_{15}
z_{21}	z_{22}	z_{23}	z_{24}	z_{25}
z_{31}	z_{32}	z_{33}	z_{34}	z_{35}
z_{41}	z_{42}	z_{43}	z_{44}	z_{45}
z_{51}	z_{52}	z_{53}	z_{54}	z_{55}

$*$

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

$=$

y_{11}	y_{12}	y_{13}
y_{21}	y_{22}	y_{23}
y_{31}	y_{32}	33

$$y_{23} = z_{23}w_{11} + z_{24}w_{12} + z_{25}w_{13} + z_{33}w_{21} + \dots$$

在神经网络中，卷积都是多通道 (multi-channel) 的，将多通道的输入映射到多通道的 hidden unit.

- 输入 $x \in R^{h \times w \times c_{in}}$ 代表含有 c_{in} 个通道，大小为 $h \times w$ 的图像

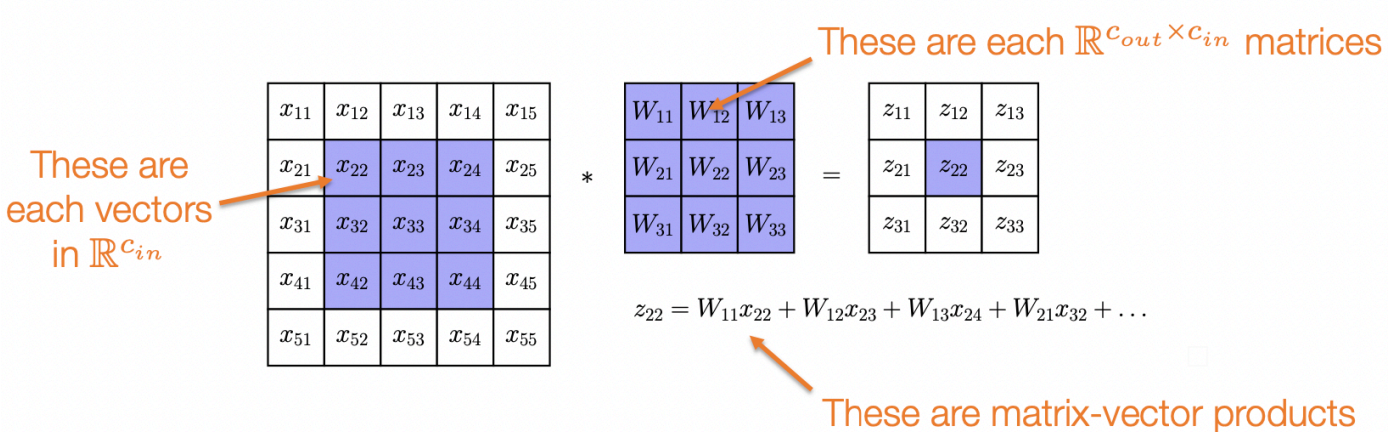
- 输出 $z \in R^{h \times w \times c_{out}}$ 代表含有 c_{out} 个通道，大小为 $h \times w$ 的图像
- 卷积核 $W \in R^{c_{in} \times c_{out} \times k \times k}$

每个输入-输出对都有相应的卷积核，单个输出通道对应所有输入通道上的卷积之和。

$$z[:,:,s] = \sum_{r=1}^{c_{in}} x[:,:,r] * W[r,s,:,:]$$

Multi-channel Convolutions in Matrix-Vector Form

抛开标量角度，将多通道卷积看作矩阵-向量乘更合适



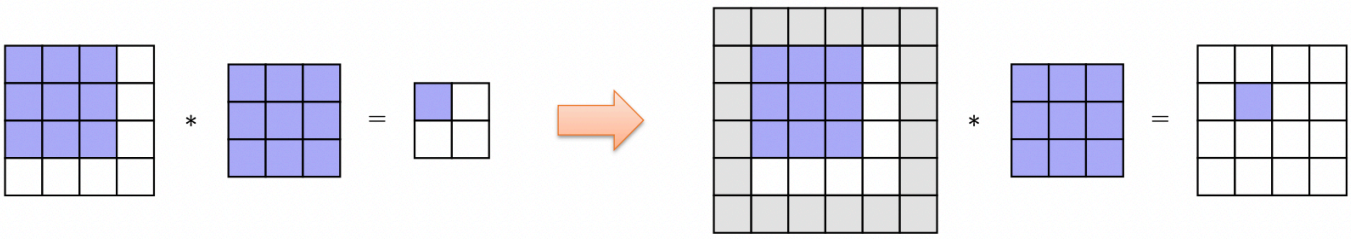
Elements of Practical Convolutions

Padding

Challenge: 卷积输出的图像比原输入图像小，计算不便

核心思想：对于 kernel size k 为奇数的情况，在输入图像的每一边填上 $(k-1)/2$ 个 0, 使输出图像与原输入图像大小相同

Variants: circular padding, padding with mean values, etc

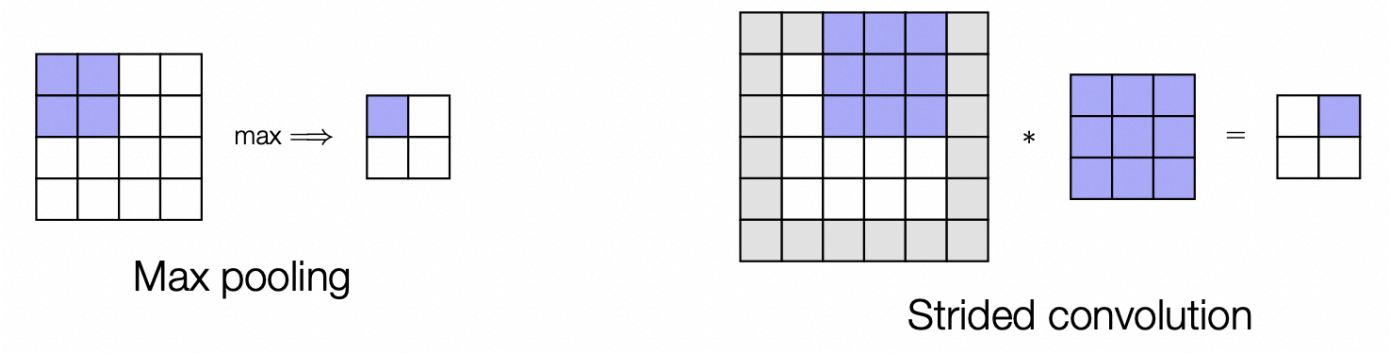


Strided Convolutions / Pooling

Challenge: 卷积在每一层对输入的分辨率相同，无法表示不同的分辨率，内存开销大。

Pooling: 对图像划分后的每个小区域做 max (max pooling) 或 average (average pooling) 操作，提炼信息

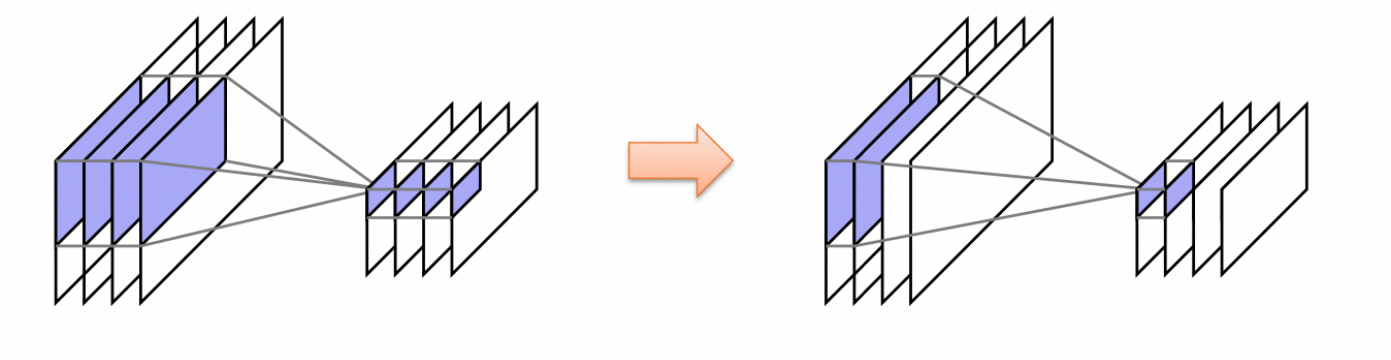
Strided Convolutions: 增大卷积核遍历图像的步长



Grouped Convolutions

Challenge: 对于输入/输出通道数量大的情况，卷积核的参数数量仍然很大，容易导致过拟合以及减慢计算速度的问题。

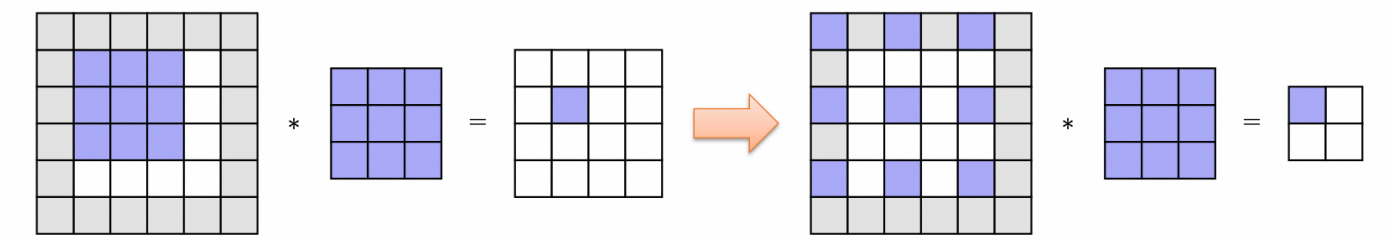
核心思想：对通道进行分组，减少卷积核数量，输入通道与输出通道之间为组与组的对应关系。



Dilations

Challenge: 卷积的感受野 (receptive field) 太小，需要扩大视野

核心思想：改变与卷积核相乘的位置，让卷积核覆盖图像中更广的范围。但同时为了让输出与输入大小相同，需要更多的 padding.



Differentiating Convolutions

卷积运算实际上是矩阵-向量乘法，但在计算图中直接实现的话重复计算以及中间计算结果多，内存开销大。因此要将卷积运算实现为原子操作，即实现为一个算符而非模块。

将卷积操作定义为

$$z = conv(x, W) \tag{2}$$

在自微分中需要计算

$$\overline{v}\frac{\partial conv(x,W)}{\partial W},\ \overline{v}\frac{\partial conv(x,W)}{\partial x} \tag{3}$$

考虑矩阵-向量乘 $z = Wx$, 有 $\frac{\partial z}{\partial x} = W$

因此 adjoint 为 $v^T W \Leftrightarrow W^T v$

换而言之，对于计算矩阵向量乘法 Wx, backward pass 需要 W^T

对于卷积计算而言，考虑一维卷积操作 $z = x * w$:

0	x_1	x_2	x_3	x_4	x_5	0	$*$	w_1	w_2	w_3	$=$	z_1	z_2	z_3	z_4	z_5

可将该卷积操作写成矩阵-向量乘 $\hat{W}x$ 的形式

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} = x * w = \underbrace{\begin{bmatrix} w_2 & w_3 & 0 & 0 & 0 \\ w_1 & w_2 & w_3 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 \\ 0 & 0 & 0 & w_1 & w_2 \end{bmatrix}}_{\hat{W}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

而 $\hat{W}^T$ 实际上就是把卷积核翻转后做卷积对应的矩阵

因此 adjoint 操作实际上为

$$\overline{v}\frac{\partial conv(x,W)}{\partial x} = \overline{v}^T\hat{W} \Leftrightarrow \hat{W}^T\overline{v} = \overline{v} * flipped(W) = conv(\overline{v}, W) \tag{4}$$

同理，对于 $\overline{v}\frac{\partial conv(x,W)}{\partial W}$ 而言，可将 W 视作向量，构造矩阵-向量乘的形式

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} = x * w = \begin{bmatrix} 0 & x_1 & x_2 \\ x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \\ x_3 & x_4 & x_5 \\ x_4 & x_5 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

此处基于 x 的 $\hat{X}$ 矩阵实际上对应 im2col 操作。

在实际场景中，通常会先得到该矩阵，再进行计算。虽然内存开销增加，但计算开销会减小很多。