On Convolutional Neural Networks

network architecture plays a crucial role in Deep Learning

It's "strange" to use networks w/ fully-connected layers on image classification task

or more genearally, work with images

one reason is that such a network architecture does not take into account the

spatial structure of the images.

⇒ (Deep) convolutional neural networks

feat. { a special architecture suits for image tasks
fast to train (has less parmeters).

a remark on "convolutional neural network" (the name)

LeCun: "The (biological) neural inspiration in models like convolutional nets is very tenuous

That's why I call them 'convolutional nets' not 'convolutional neural nets'

and why we call the nodes 'units' and not 'neurons'"

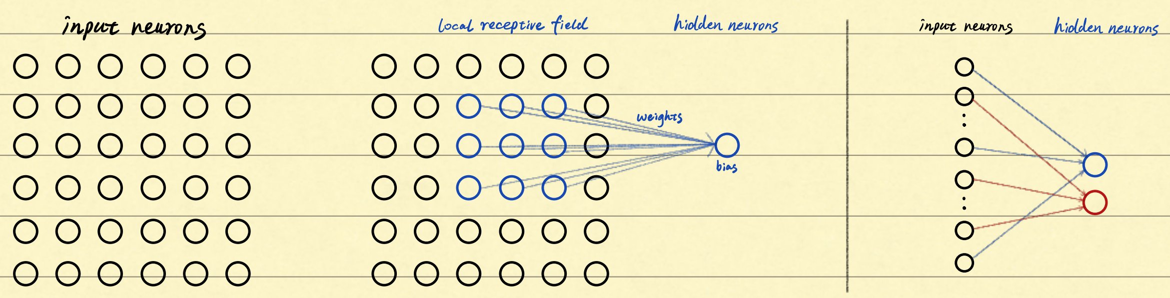
Despite this remark, convolutional nets use many of the same ideas as the neural networks:

backpropagation, gradient descent, regularization, non-linear activation functions ...

So, in common practice, 'convolutional nets' and 'convolutional neural nets' are interchangeable ✓

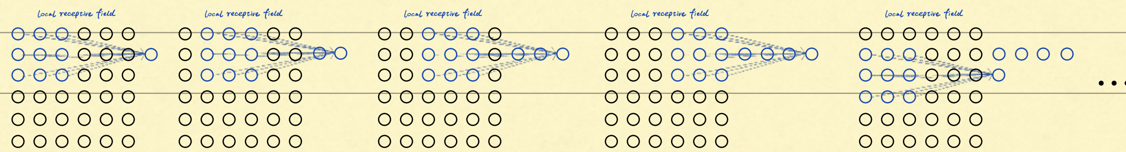
3 basic ideas in CNNs: local receptive fields, shared weights, and pooling.

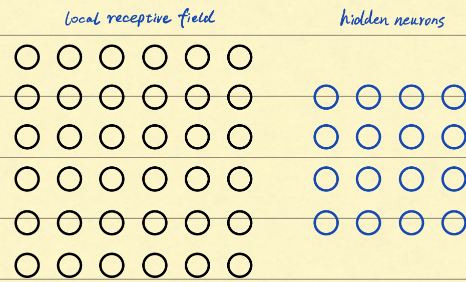
1. local receptive fields: connect the input pixels to a layer of hidden neurons
(局部感受视野)
but with selected group of neurons



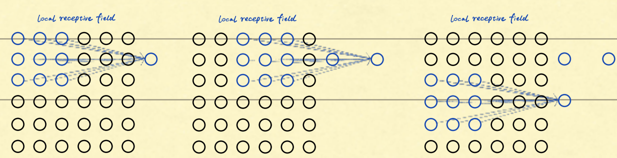
Stride (hyperparameter)

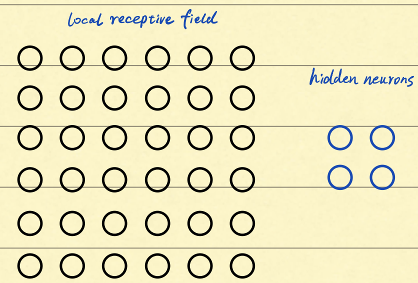
Stride = 1





Stride = 2





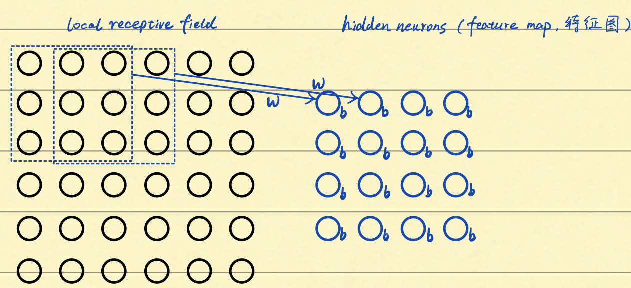
2. Shared weights and bias

We're going to use the same weights and bias for each of the hidden neurons

output for j-th, k-th hidden neuron:

$$\sigma\left(b+\sum_{l=0}^{2}\sum_{m=0}^{2} w_{l,m}\, a_{j+l,k+m}\right)$$

this operation is sometime known as a convolution



shared w&b also means that all the neurons in the first hidden layer detect exactly the same feature.
(just at different locations)

why this makes sense? —— weights as feature detector —— hidden layer : a feature map

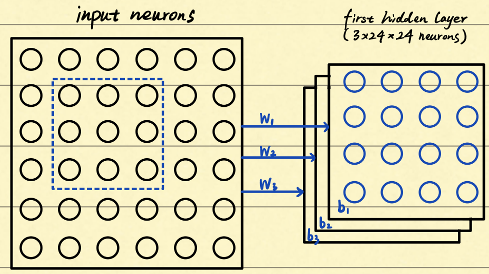
for the weights & bias defining the feature map (or the weights & bias in a feature detector),

we often call them a kernel or filter

so far, one feature detector / kernel / filter can detect one kind of localized feature.

it's natrul to extend this to more features

— we'll need more than one feature (map) for image recognition.



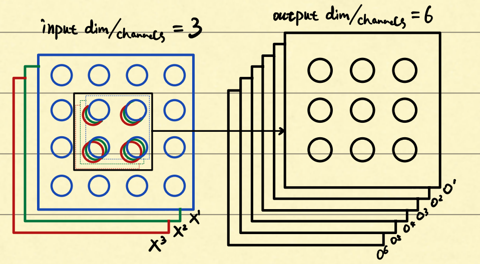
每个特征图 ~ 一个 Kernel ~ 输入(特征图)的一类特征

* a huge advantage of sharing weights & biases : it greatly reduces the number of parameters

e.g. : $\underbrace{(3\times3+1)\times 3 = 30}_{\text{conv layer}}$ vs. $\underbrace{(6\times6)\times 5 = 180}_{\text{fully-conn layer}}$ intuition : less params to get same performance

* input channel > 1 : (won't effect outputs!)

卷积操作自然扩展到每个输入通道



for t-th feature map O^t

$$O^t_{(i,j)} = \sigma\left(\sum_{c=1}^{C_{in}}\sum_{m=1}^{K_h}\sum_{n=1}^{K_w} X_{i+m,\,j+n,\,c}\cdot K^t_{m,n,c} + b^t\right)$$

C_{in} : number of input channels

K^t : Kenel for t-th feature map.

$K^t \in \mathbb{R}^{k_h\times k_w\times C_{in}}$

Note that the Kernel Shape here is $[\underbrace{k_h}_{\text{kernel height}}, \underbrace{k_w}_{\text{kernel width}}, \underbrace{C_{in}}_{\text{input channels}}]$

thus the number of weights is $k_h\cdot k_w\cdot C_{in}$

* 1×1 Convolutional Kernels (1×1 卷积核有什么用) $O^t_{(i,j)} = \sigma\left(\sum_{c=1}^{C_{in}} X_{i,j,c}\, K^t_c + b^t\right)$

输入在通道维度上的线性组合.

维度缩减/扩展（Dimensionality Reduction/Expansion）：在不影响空间维度的情况下改变通道数量，降低计算成本

跨通道信息混合/特征融合：在每个空间位置组合跨通道的信息。本质上是对每个像素独立使用全连接层

增加非线性性/非线性变换：通过激活函数（activation functions）添加非线性，同时保持空间结构。

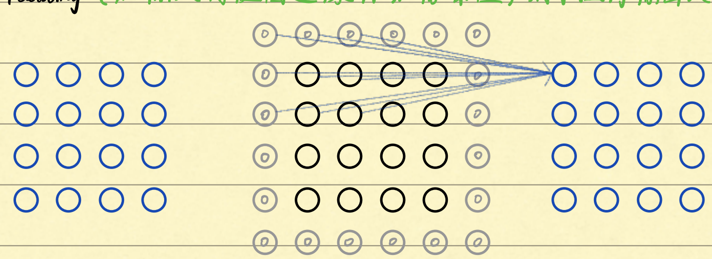
*瓶颈架构（Bottleneck Architectures）：在ResNet和Inception网络中使用，在保持表示能力的同时减少参数数量。

事实上，如果考虑 $\vec{O}_{(i,j)}$ (表示输出特征图在通道维度上形成的向量)

$$\vec{O}_{(i,j)} = \sigma\left(\underbrace{\begin{bmatrix} K^1_1 & \cdots & K^1_{C_{in}} \\ \vdots & & \vdots \\ K^{C_{out}}_1 & \cdots & K^{C_{out}}_{C_{in}} \end{bmatrix}}_{C_{out}\times C_{in}} \cdot \underbrace{\begin{bmatrix} x_{i,j,1} \\ \vdots \\ x_{i,j,C_{in}} \end{bmatrix}}_{C_{in}\times 1} + \underbrace{\begin{bmatrix} b^1 \\ \vdots \\ b^{C_{out}} \end{bmatrix}}_{C_{out}\times 1}\right) = \sigma\left(K\vec{x}_{(i,j)} + \vec{b}\right)$$

那么 1×1 的卷积核 ⟺ $\mathbb{R}^{C_{in}} \to \mathbb{R}^{C_{out}}$ 上的一个全连接层

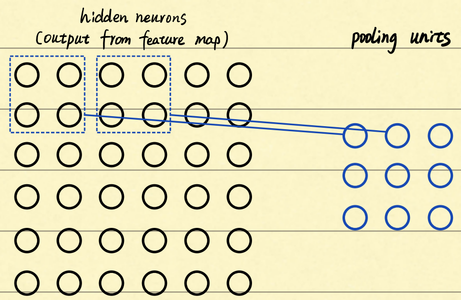
* Padding (在输入特征图边缘添加像素值，用于控制输出大小)



- Valid Padding (padding = 0) 输出尺寸小于输入尺寸
- Same Padding (padding = $\frac{\text{Kernelsize} - 1}{2}$) 保持输出尺寸 = 输入尺寸 (stride = 1 时)
- Full Padding (padding = Kernelsize − 1) 卷积核在任何与输入有交集的地方都有计算.

3. Pooling layers.

- usually used immediatly after convolutional layers
- it simplify the information in the output from the convolutional layer (reduce dimension → reduce number of parameters)



a pooling layer takes each feature map output from the convolutional layer and prepares a condensed feature map

- max-pooling : outputs the maximum activation in the input region
- mean-pooling : ---- mean ----
- L2-pooling : $\sqrt{\sum_j o_j^2}$

※ Output size:

输入高度 padding 高度 kerel 高度

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding_h - H_k}{stride_h} \right\rfloor + 1$$

步长高度

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding_w - W_k}{stride_w} \right\rfloor + 1$$