

Contents

1	1. Convolutional Neural Networks	1
1.1	1.1 Computer Vision	1
1.2	1.2 Edge Detection Example	1
1.3	1.3 More Edge Detection	2
1.4	1.4 Padding	5
1.5	1.5 Strided Convolutions	5
1.6	1.6 Convolutions Over Volume	6
1.7	1.7 One Layer of Convolutional Network	8
1.8	1.8 Simple Neural Network Example	10
1.9	1.9 Pooling Layers	10
1.10	1.10 CNN Example	10
1.11	1.11 Why Convolutions?	12

contents

- 1. Convolutional Neural Networks
- 1.1 Computer Vision
- 1.2 Edge Detection Example
- 1.3 More Edge Detection
- 1.4 Padding
- 1.5 Strided Convolutions
- 1.6 Convolutions Over Volume
- 1.7 One Layer of Convolutional Network
- 1.8 Simple Neural Network Example
- 1.9 Pooling Layers
- 1.10 CNN Example
- 1.11 Why Convolutions?

1 1. Convolutional Neural Networks

1.1 1.1 Computer Vision

dl 对 cv 的影响:

- 能够催生 cv 领域中很多新产品和应用
- cv 的很多思想对其他领域也很有借鉴作用, 例如语音识别等

cv problems:

- image classification
- object detection
- neural style transfer

deep learning on large images:

- 如果输入是 64×64 的图像, 那么由于有 3 个通道, 所以是 $64 \times 64 \times 3 = 12288$ 维
- 如果输入是 1000×1000 的图像, 就有 $300w$ 维, 如果第一层有 1000 个隐层单元, 那么如果是全连接, $W^{[1]}$ 就是一个 $(1000, 300w)$, 所以有 $3000m = 3\text{billion}$ 的参数, 一方面难以获得足够的数据来防止过拟合, 另一方面对计算量和内存的需求是很大的。因此, 需要卷积。

1.2 1.2 Edge Detection Example

例如, 我们希望有两个 detector, 一个可以检测垂直方向的边缘, 一个可以检测水平方向的边缘。

vertical edge detection:

- 输入一张 6×6 的灰度图 (所以没有 3 通道, 只有 1 通道), 数字越小, 越明亮 (白)
- 可以定义一个 3×3 的 filter (又称 kernel),
- 然后两者之前进行卷积运算, 得到的结果是一个 4×4 的矩阵。计算过程如下:
 - 4×4 的左上角的元素是将 filter『粘贴』到原始图片的左上角, 然后对应元素相乘, 再把所有相乘的结果相加起来
 - 其他元素就是将 filter 不断地 shift, 然后进行卷积, 得到对应的结果, 以此类推。

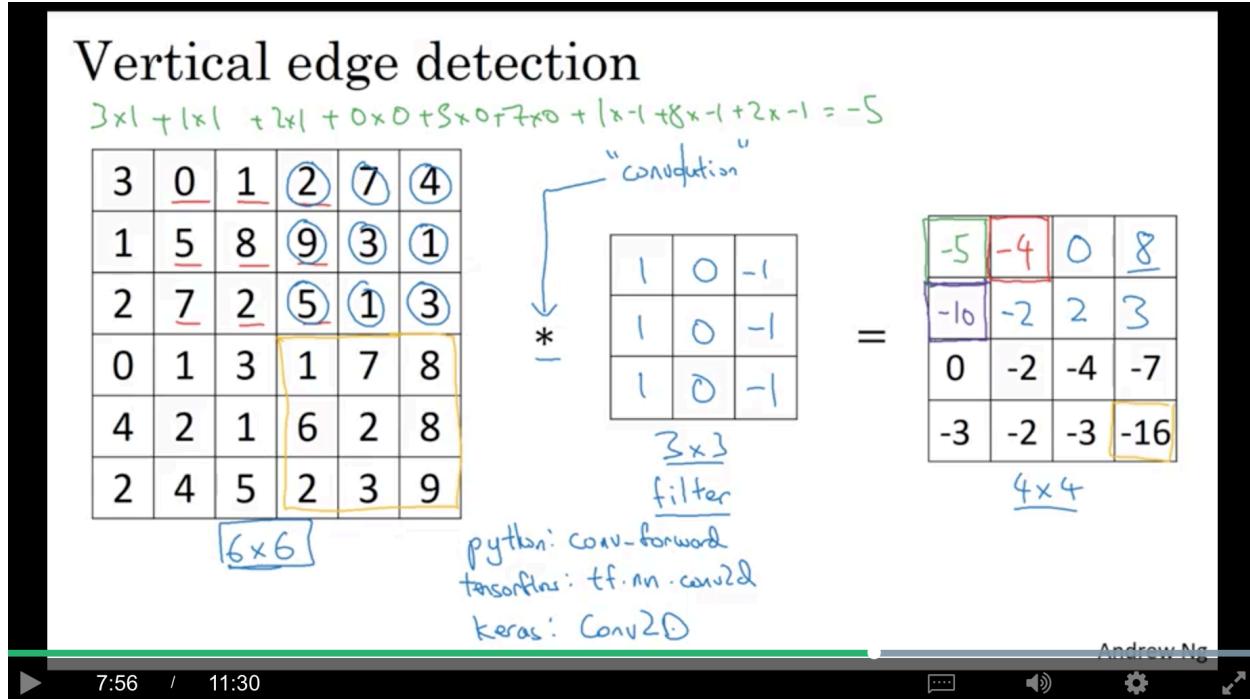


Figure 1: vertical-edge-detection.png

卷积的实现:

- python: conv_forward
- tensorflow: tf.nn.conv2d
- keras: Conv2D

从下图可以看到, 这个 filter 可以学习到原始图像中明暗分界的那条垂直竖线 (对中间那个垂直区域的『激活』值比较高)。可见, 这个 filter (左边亮, 右边暗) 可以检测从明到暗的 transition。

1.3 1.3 More Edge Detection

如果输入图像左右颠倒, 使用同一个 filter, 可以发现卷积结果仍然可以检测这条明暗交界线, 只是这次变成了从暗到明的 transition, 而结果的中间垂直区域变成黑色的。

类似地, 如果要检测水平边缘, 可以用如下 filter, 第一行亮, 第三行暗。

- sobel filter: 就是把垂直 filter 的第二行乘了 2, 会稍微 robust 一点
- scharf filter: 把垂直 filter 第一行和第三行分别乘 3, 第二行乘 10

在 dl 中, 这个 3×3 的 filter 的 9 个元素, 不需要手动设置, 当做模型的参数, 通过 bp 来学习, 这样, 这些 filter 就可以用来检测各种角度的边缘。

Vertical edge detection

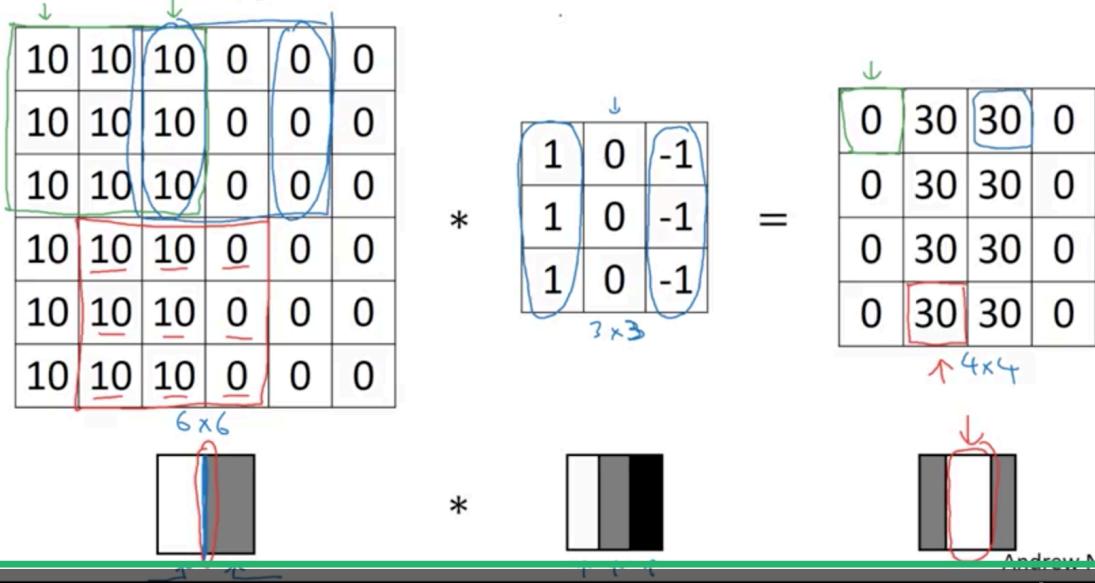


Figure 2: vertical-edge-detection-2.png

Vertical edge detection examples

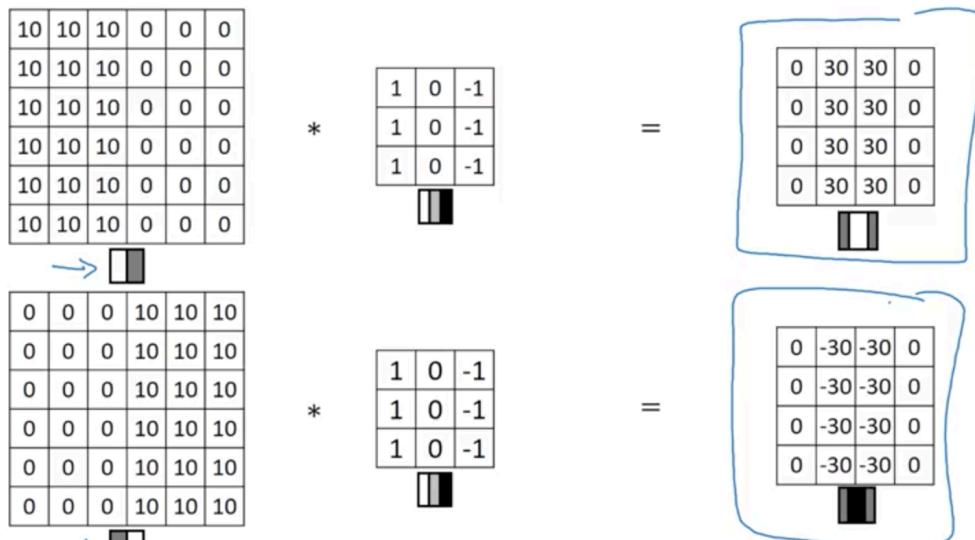


Figure 3: vertical-edge-detection-3.png

Vertical and Horizontal Edge Detection

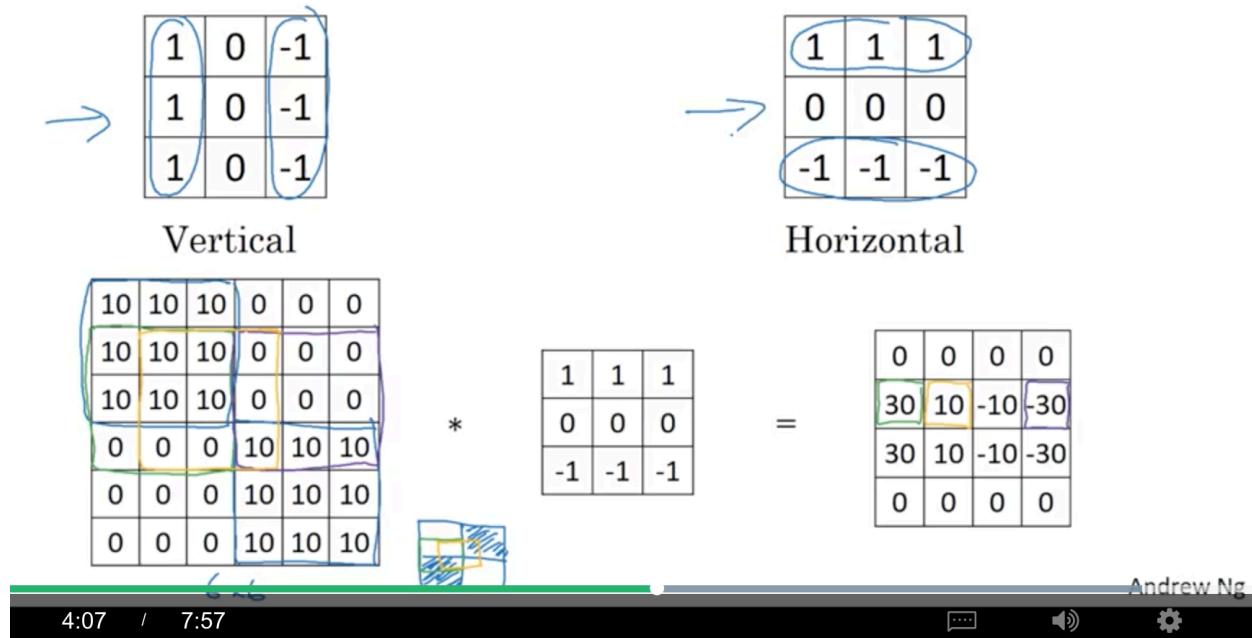


Figure 4: vertical-edge-detection-4.png

Learning to detect edges

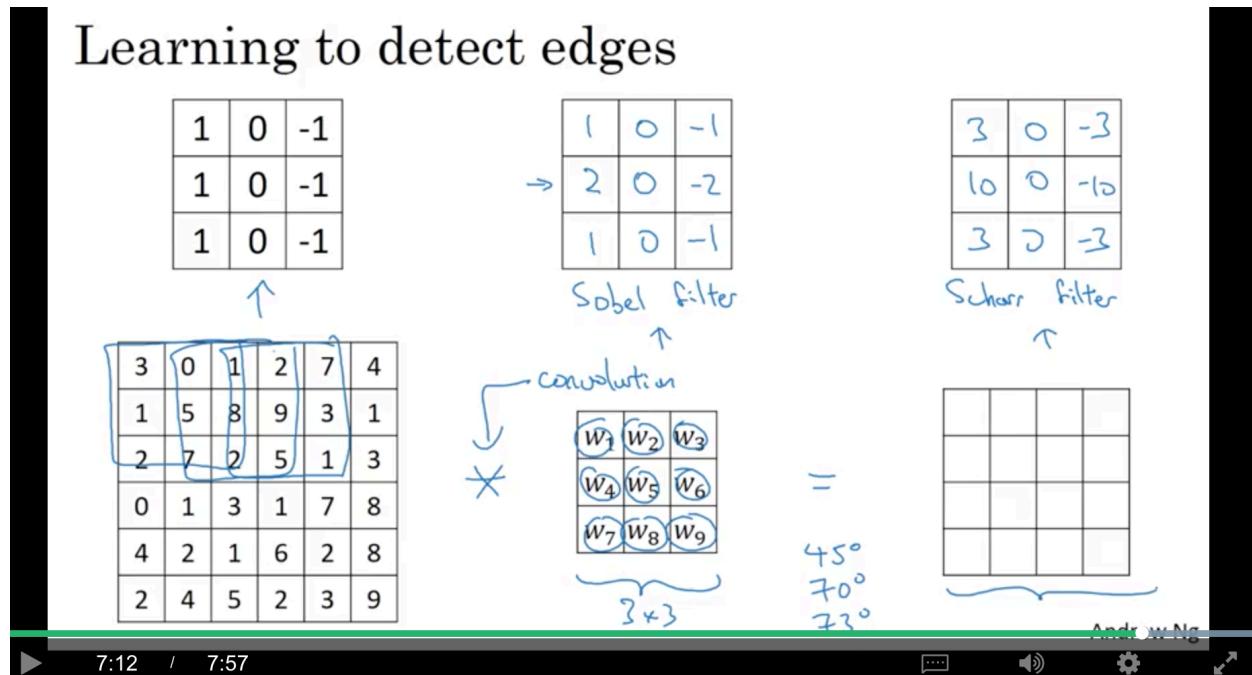


Figure 5: learning-to-detect-edges.png

1.4 1.4 Padding

正常的卷积操作，输入图片是 $n \times n$, filter 是 $f \times f$, 得到的卷积结果是 $(n - f + 1) \times (n - f + 1)$, 有以下两个缺点:

- **shrinking output**: 卷积的结果比原图像小，所以只能做比较少次的卷积，不然图片会越变越小，最后变成 1×1
- **throwing away info from edges**: 看左上角的元素，其实只被使用了一次；而中间的元素则会被使用很多次。所以角落或者边缘附近的元素被使用的次数比中间元素少得多，因此，图片边缘的很多信息其实没有被充分利用。

padding 就是在整幅图像的边缘，加上一个宽度为 p (这个例子中 $p = 1$) 的 border(可以都填充上 0)，例如， 6×6 就变成了 8×8 ，即 $(n + 2) \times (n + 2)$ ，那么与一个 3×3 的 filter 做卷积，就得到一个 6×6 ，即 $(n + 2 - 3 + 1) \times (n + 2 - 3 + 1)$ 的结果。

假设 padding size 是 p ，那么，得到的卷积结果是 $(n + 2p - f + 1) \times (n + 2p - f + 1)$

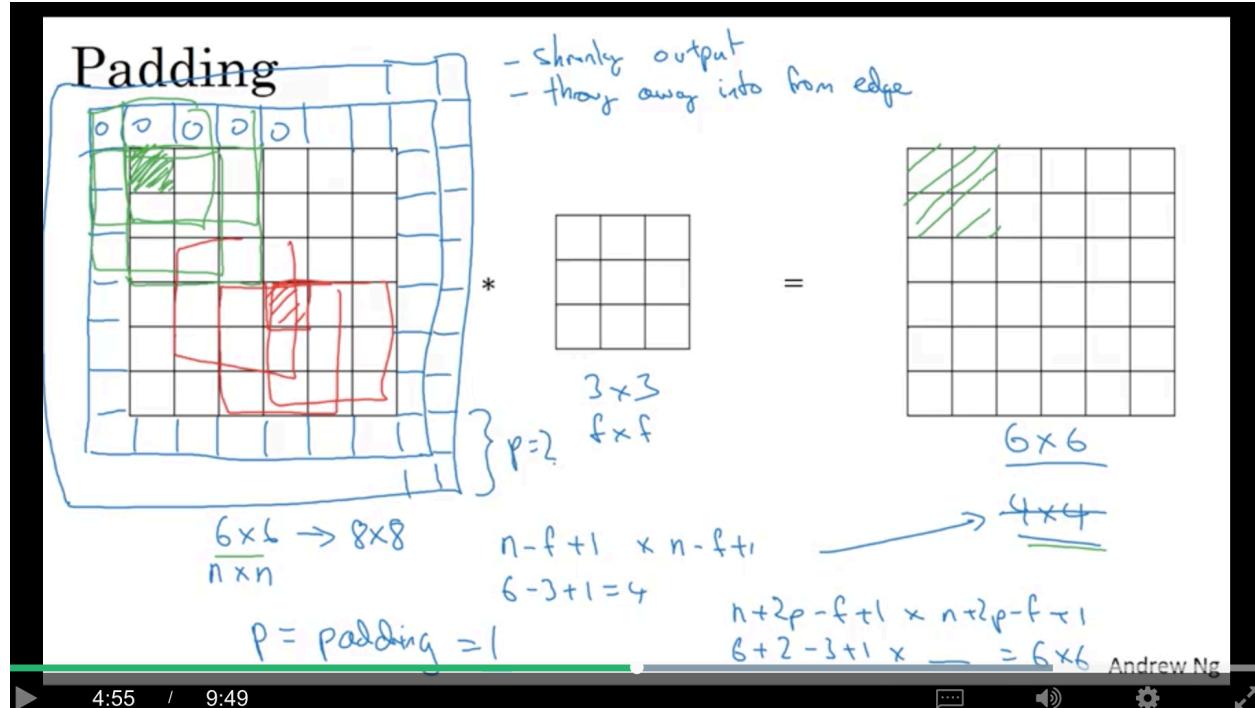


Figure 6: padding.png

总结一下，两种卷积：

- valid convolutions: no padding
- same convolutions: padding 使得 output 和 input 的 size 一样

$$n + 2p - f + 1 = n \Rightarrow p = \frac{f - 1}{2}$$

在 cv 的传统中， f 一般都是奇数，原因如下：

- 如果是偶数，就会出现不对称的 padding
- 使用奇数 f 的 filter 话，这样的 filter 会有一个中心点，在 cv 中，有一个特殊点是很好的，可以描述一个 filter 的位置

1.5 1.5 Strided Convolutions

stride=2 意味着，算完一个后，要算下一个的时候，往右不只 shift 一个像素，而是 shift stride 个像素。纵向 shift 的时候也一样，一次 shift stride 个像素。

Valid and Same convolutions

$\nearrow n \rightarrow \text{padding}$

“Valid”: $n \times n \times f \times f \rightarrow n-f+1 \times n-f+1$
 $6 \times 6 \times 3 \times 3 \rightarrow 4 \times 4$

“Same”: Pad so that output size is the same as the input size.

$$n+2p-f+1 \times n+2p-f+1$$

$$n+2p-f+1 = n \Rightarrow p = \frac{f-1}{2}$$

$$3 \times 3 \quad p = \frac{3-1}{2} = 1 \quad | \quad 5 \times 5 \quad p=2$$

f is usually odd
 1×1
 3×3
 5×5
 7×7

9:12 / 9:49

Andrew Ng

Figure 7: valid-same-conv.png

所以，对于 padding 是 p ，而 stride 是 s 的卷积，得到的 output 大小是 $(\frac{n+2p-f}{s} + 1)(\frac{n+2p-f}{s} + 1)$ ，如果这个结果不是一个整数，那么就取 floor(向下取整)。【因为要保证参与计算卷积的都是图像的一部分，或者是 padding 的一部分，如图右上角超出的部分就不应该参与计算，所以这里要取 floor】

在标准的信号处理/数学课本中，在进行卷积的 element-wise 乘法之前，需要将 filter 沿水平和垂直方向旋转（如图）。这样，可以使卷积操作满足结合律，即 $(A * B) * C = A * (B * C)$ ，这对信号处理很有用，但对深度学习其实并不重要。

所以，严格意义上说，前面讲到的『卷积』，应该叫做『交叉相关 (cross-correlation)』。而在大多数 ml 相关的文献中，都叫做卷积。

1.6 1.6 Convolutions Over Volume

对于 rgb 图像而言，有 3 个 channel，所以，相应的，filter 也堆叠了 3 层。

注意：

- 输入图片的 shape 是 (height, width, num of channels)，
- filter 的 shape 是 (height_f, width_f, num of channels)，filter 和输入的第三维必须相等。
- 输出的图片是 (height_o, width_o)，注意，并没有第三维！！

卷积操作就是，将这个三维的 filter (例如 $3 \times 3 \times 3$) 与输入图像的对应位置相乘，再将这 27 个数相加，得到的结果就是 output 的一个元素。

如果只想检测某一个 channel 的边缘，可以把另外几个 channel 的 filter 全部变成 0；如果想检测多个 channel，就正常搞。

如果有多个 filter，那 output 就可以有第三维了，例如，有两个 $3 \times 3 \times 3$ 的 filter，那么可以把这两个 output 堆叠起来，这样 output 的第三维就是 2 啦。

总结：

输入是 $n \times n \times n_C$ ，有 n'_C 个 $f \times f \times n_C$ 的 filter，那么得到的结果就是 $(n - f + 1) \times (n - f + 1) \times n'_C$

另外，这里的 channel 数，也就是 n_C ，通常也会被称为 depth。

Strided convolution

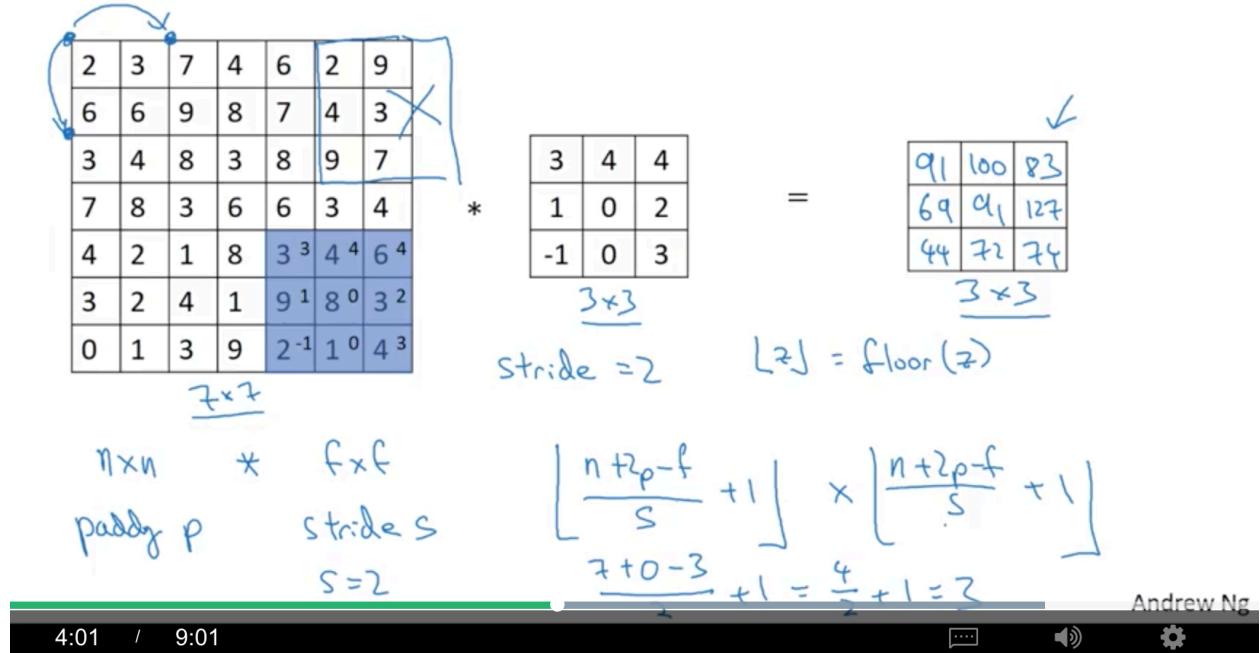


Figure 8: strided-conv.png

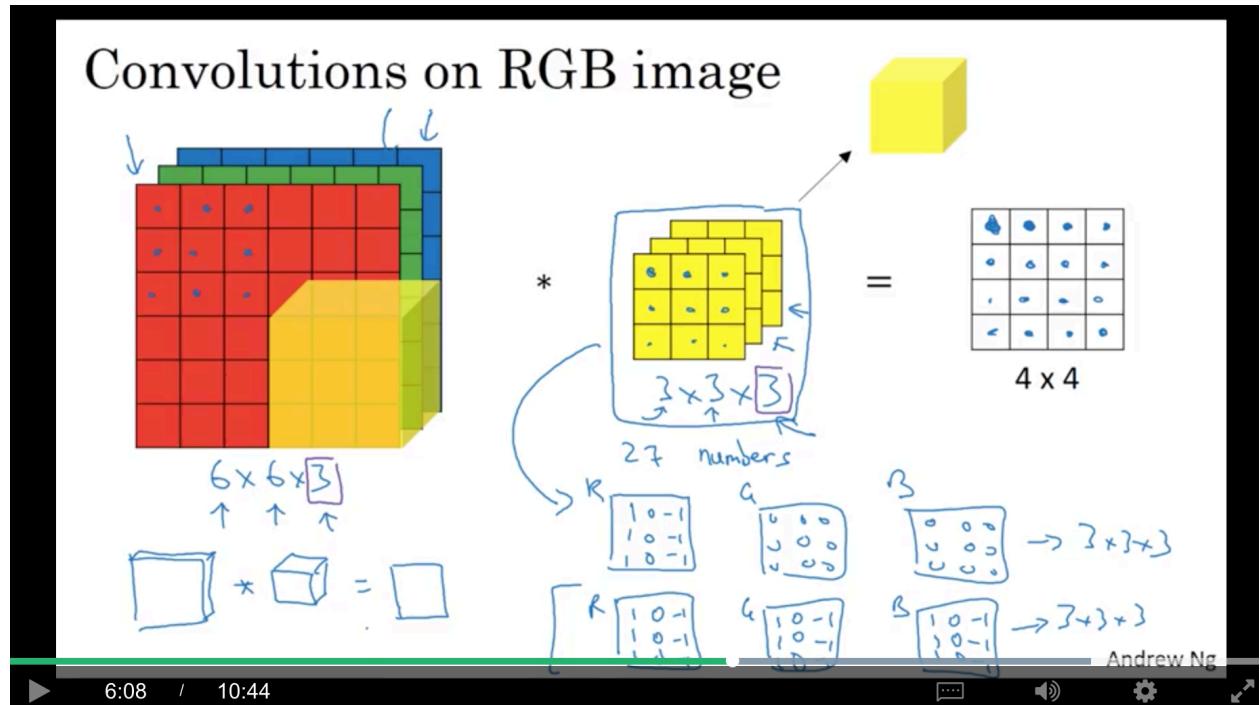


Figure 9: rgb-images-convs.png

Multiple filters

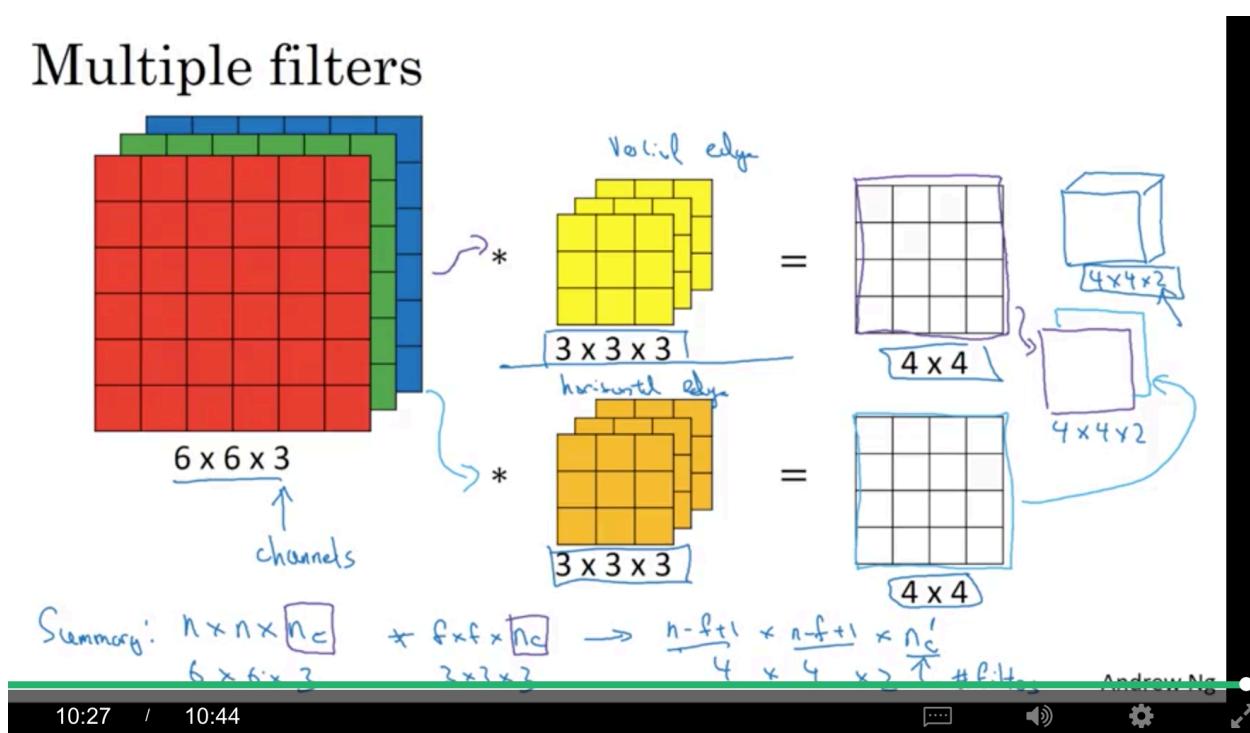


Figure 10: multiple-filters.png

1.7 One Layer of Convolutional Network

对于每一个 filter 的输出，加上一个 bias(输出矩阵的每一维都加上这个 bias)，然后再经过非线性激活（如 relu），得到最终的输出。

可以把原始输入看成 $a^{[0]}$ ，而 filter 可以看成是 $W^{[1]}$

参数个数：假设有 10 个 $3 \times 3 \times 3$ 的 filter，因为每个 filter 有一个 bias，所以有 $(3 \times 3 \times 3 + 1) \times 10 = 280$ 个参数。所以，cnn 的参数比全连接的 nn 少了很多，所以更不容易过拟合。

对于第 l 层，

- filter 的高/宽是 $f^{[l]}$
- padding 是 $p^{[l]}$
- stride 是 $s^{[l]}$
- filter 的个数是 $n_C^{[l]}$
- 每个 filter 的 shape 是 $f^{[l]} \times f^{[l]} \times n_C^{[l-1]}$
- input 的 shape 是 $n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}$
- output 的 shape 是 $n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$
- $n^{[l]} = \text{floor}(\frac{n^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1)$ ，对 n_H 和 n_W 是类似的
- activations $^{[l]}$ 的 shape 是 $n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$
- 所以，在做 mini-batch 梯度下降的时候，有 m 个 examples 的矩阵 $A^{[l]}$ 的 shape 就是 $m \times n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$
- 因为有 $n_C^{[l]}$ 个 filter，而每个 filter 的 shape 是 $f^{[l]} \times f^{[l]} \times n_C^{[l-1]}$ ，所以，权重矩阵的 shape 就是 $f^{[l]} \times f^{[l]} \times n_C^{[l-1]} \times n_C^{[l]}$
- 而 bias 的 shape 就是 $n_C^{[l]}$ ，为了统一，会计为 $1 \times 1 \times 1 \times n_C^{[l]}$

在有些开源的代码里，顺序可能不一样，例如 $m \times n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$ 可能变成 $m \times n_C^{[l]} \times n_H^{[l]} \times n_W^{[l]}$ ，而在大部分开源的框架里，其实都有一个参数，可以让用户决定把 $n_C^{[l]}$ 放在最前面或者最后面的。

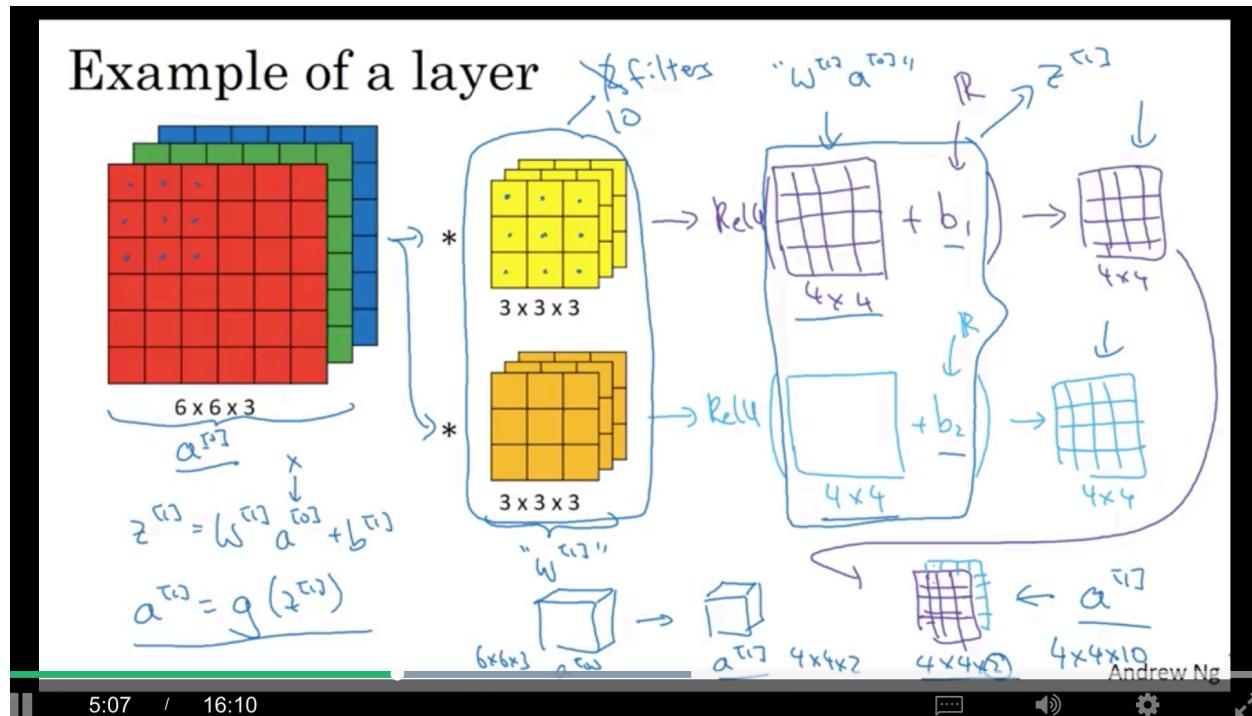


Figure 11: one-conv-layer.png

Summary of notation

If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$n_c^{[l]}$ = number of filters

→ Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l]}$

Activations: $a^{[l-1]} \rightarrow n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

bias: $n_c^{[l]} - (1, 1, 1, n_c^{[l]})$ #filters in layer l .

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$$n_{HW}^{[l]} = \left\lfloor \frac{n_H^{[l-1]} \times n_W^{[l-1]} + 2p - f}{s^{[l]}} \right\rfloor + 1$$

$$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$$

$$n_c \times n_H \times n_W$$



Figure 12: notations-conv.png

1.8 1.8 Simple Neural Network Example

如图，在 $7 \times 7 \times 40$ 之后，将这层直接 flatten，变成一个 1960 维的向量，然后再接一个 logistic 或者 softmax，再进行分类

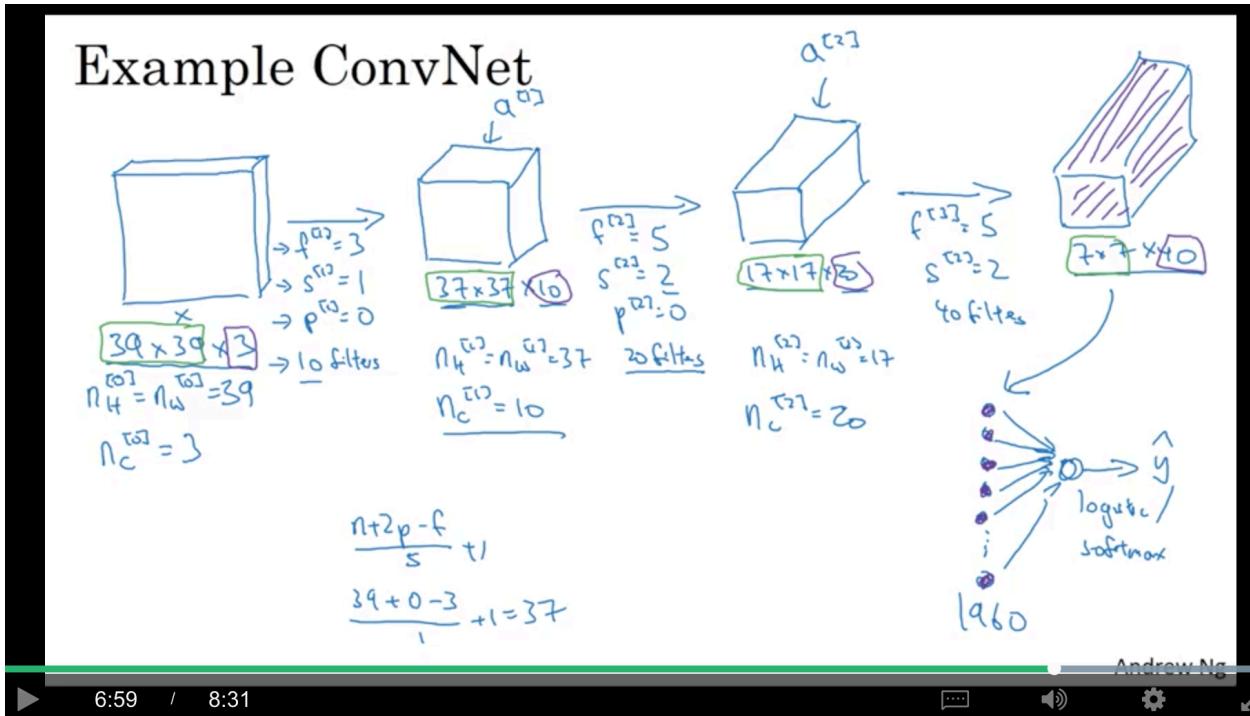


Figure 13: conv-nets.png

1.9 1.9 Pooling Layers

pooling 的基本思想，与卷积的运算基本一样，区别在于，filter 和原图像做的不是卷积操作，而是对原图像的 filter 大小的区域做 max/min 之类的操作，而且 filter 并没有第三维，输入和输出的第三维是一样的。

同样有 f /stride 这些参数，而且 $\text{floor}(\frac{n+2p-f}{s} + 1)$ 同样适用。这些参数是超参，针对 pooling，并没有需要学习的参数。

average pooling 相对于 max pooling 来说，应用得少很多。有个例外，在非常深的网络里，例如将 $7 \times 7 \times 1000$ 的层 pooling 成 $1 \times 1 \times 1000$ 的层时，一般用 average pooling，具体原因后面会讲，不急不急。。

超参的设置，一般 $s=2$, $f=2$ 或者 $s=3$, $f=2$ ，当然，还有 padding 可以设置，但非常少用，有一个例外，下周再说

1.10 1.10 CNN Example

以 LeNet-5 为例，

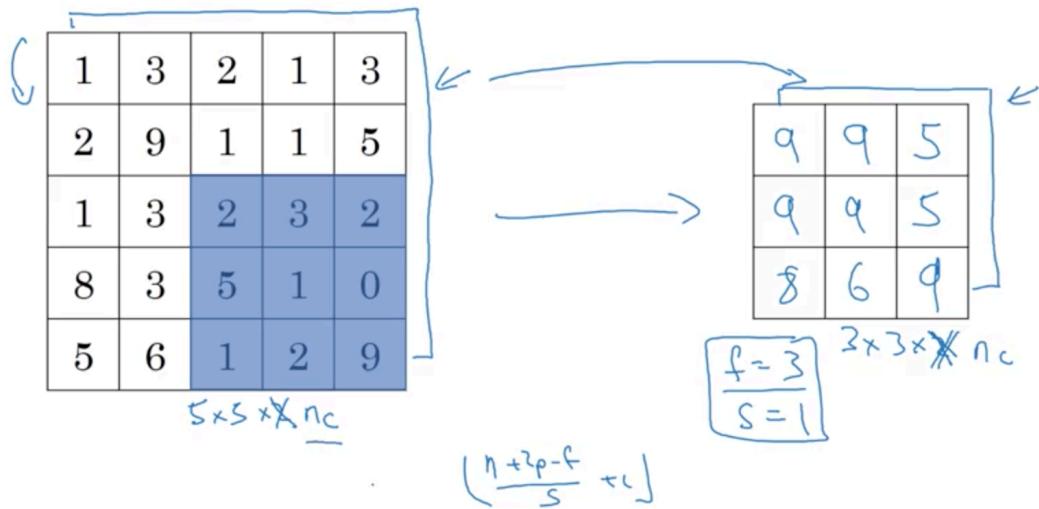
conv-1 的输入有 6 个 filter，有两种叫法：

- 一个 conv 加上一个 pooling，称为一个 layer。
- conv 算一个 layer，pooling 也算一个 layer。

但因为 conv 有参数可以学习，pooling 只有超参，所以本课程中会使用叫法一。

layer2 的输出展开成一个 400 维的向量，然后接 FC3 (权重矩阵 $W^{[3]}$ 的 shape 是 $(120, 400)$)，变成一个 120 维的向量，然后接一个 FC4 变成 84 维，最后接一个 softmax。

Pooling layer: Max pooling



6:44 / 10:25

Andrew Ng

Summary of pooling

Hyperparameters:

f : filter size	$f=2, s=2$
s : stride	$f=3, s=2$
<u>Max or average</u> pooling	
→ <u>p: padding</u>	

No parameters to learn!

$$\begin{aligned}
 & n_H \times n_w \times n_c \\
 & \downarrow \\
 & \left\lfloor \frac{n_H-f+1}{s} \right\rfloor \times \left\lfloor \frac{n_w-f}{s} + 1 \right\rfloor \\
 & \times n_c
 \end{aligned}$$

10:06 / 10:25

Andrew Ng

Figure 14: max-pooling.png

对于超参的设置，一般不要自己发明新的，直接参考业界取得比较好结果的设置。

一般来说，随着层数的增加， n_H 和 n_W 会减小，而 n_C 一般会增加。

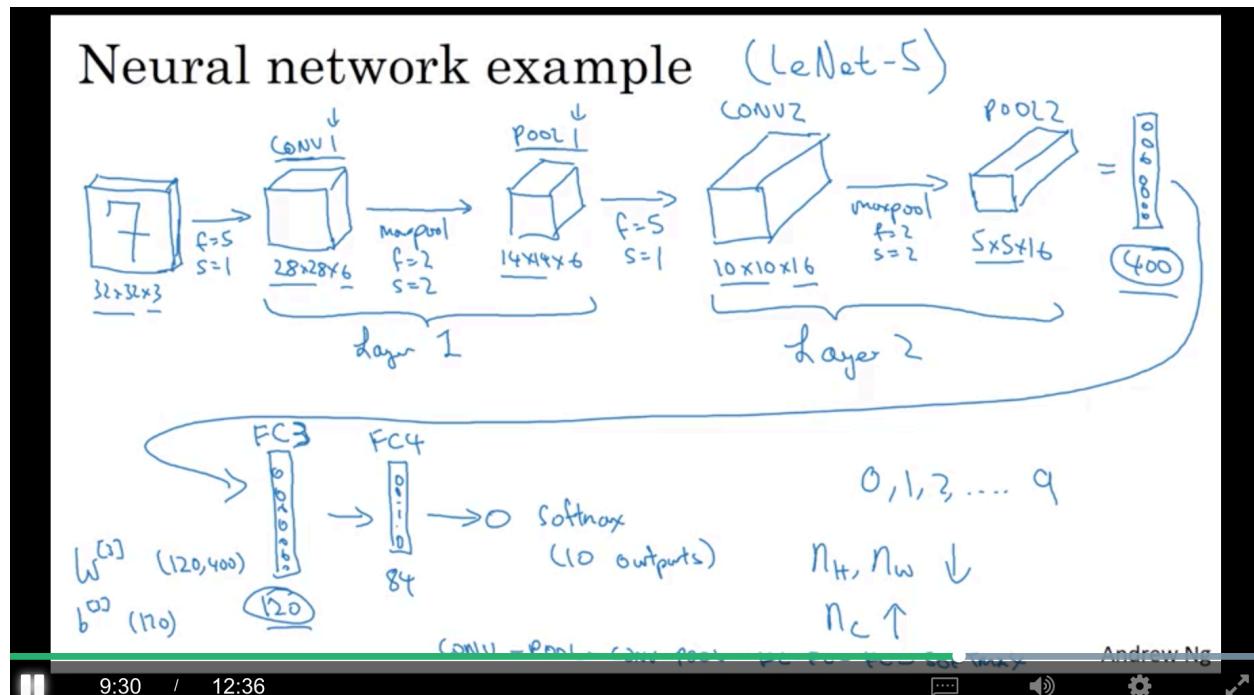


Figure 16: lenet-5.png

各层的 shape 和 size，以及参数数目如下表，可见：

- pool 层没有参数
- conv 层的参数比 fc 的少，例如 CONV1 层，可见是 filter 的大小是 5×5 (本来应该是 $5 \times 5 \times 3$ 的，可能是只要 5×5 ，然后 3 个 channel 都用同一个 5×5 去搞吧 ~)，个数为 8，所以参数有 $(5 \times 5 + 1) \times 8 = 208$
- 随着层数的增加，activation size 逐渐变小，没有『骤降』

1.11 Why Convolutions?

cnn 的两个好处：

- 参数共享：一个 feature detector 对图片的一部分有效时，很可能对图片的其他部分也有效
- 稀疏连接：输出的每一个元素只与输入的一部分有关系

因此，cnn 可以有更少的参数，从而可以对更小的数据集，更不容易过拟合。

另外，cnn 在捕捉 translation invariance 方面也很擅长，例如，一张猫的图像，往右平移了一些距离，很多 feature 还是类似的，cnn 也能感知到。

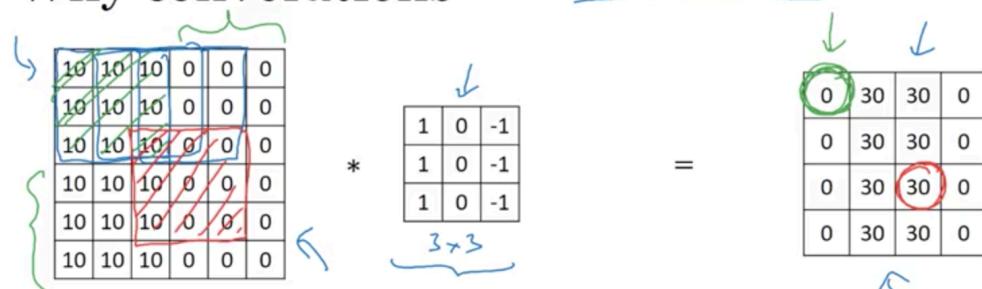
Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 $a^{(0)}$	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208 ←
POOL1	(14,14,8)	1,568	0 ←
CONV2 (f=5, s=1)	(10,10,16)	1,600	416 ←
POOL2	(5,5,16)	400	0 ←
FC3	(120,1)	120	48,001 ←
FC4	(84,1)	84	10,081 ←
Softmax	(10,1)	10	841



Figure 17: lenet-5-params.png

Why convolutions



Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

→ **Sparsity of connections:** In each layer, each output value depends only on a small number of inputs.



Figure 18: why-cnns.png