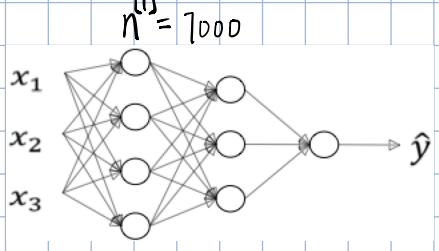
Computer Vision Problems

- Image Classification
- Object Detection
- Neural Style Transfer and more ...

Deep Learning on Large Images

Low Resolution Image

$64 \times 64 \times 3 = 12288$, $w^{[1]} = (1000, 12288)$

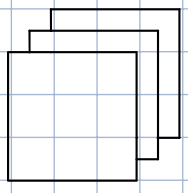


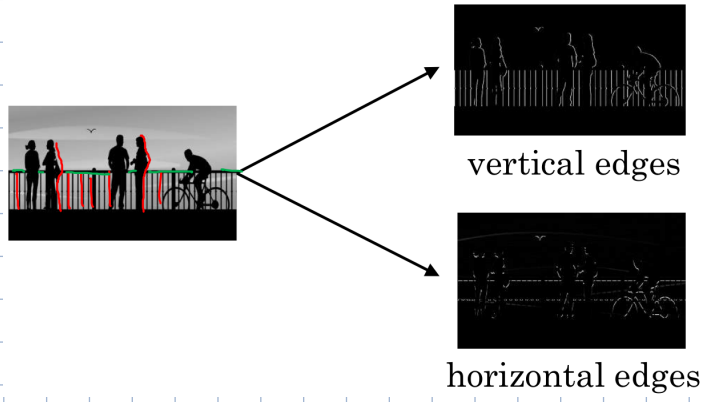
High Resolution Image

$1000 \times 1000 \times 3 = 3 \times 10^6$, $w^{[1]} = (1000, 3 \cdot 10^6)$

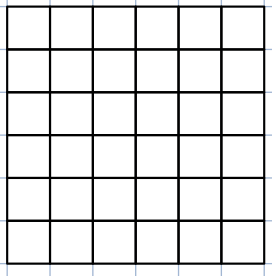
- With 3 billions parameters, it's hard to get large amount of data to prevent overfitting.
- The memory consumption is so high that it's unrealistic to train it.

Edge Detection Example





Vertical Edge Detection

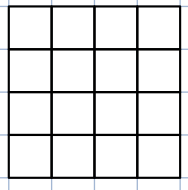


Convolution *

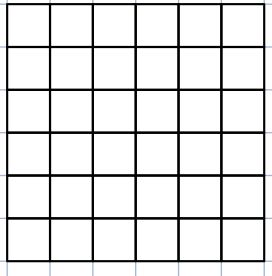
1	0	-1
1	0	-1
1	0	-1

Filter Kernel

=



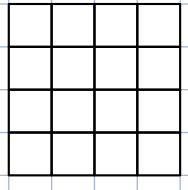
Horizontal Edge Detection

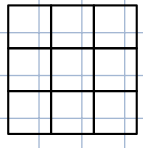


*

1	1	1
0	0	0
-1	-1	-1

=





And other hand-crafted filters:

1	0	-1
2	0	-2
1	0	-1

Sobel

3	0	-3
10	0	-10
3	0	-3

Scharr

In convolutional neural network, it doesn't have hand-crafted filters.

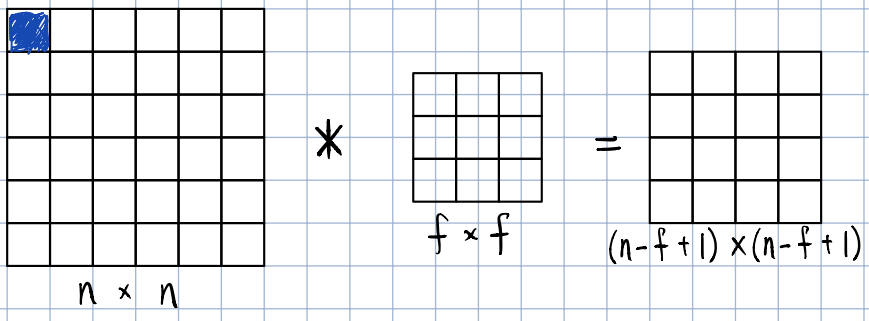
Instead, it learns the nine parameters

w_1	w_4	w_7
w_2	w_5	w_8
w_3	w_6	w_9

. AMAZING!!!

Padding

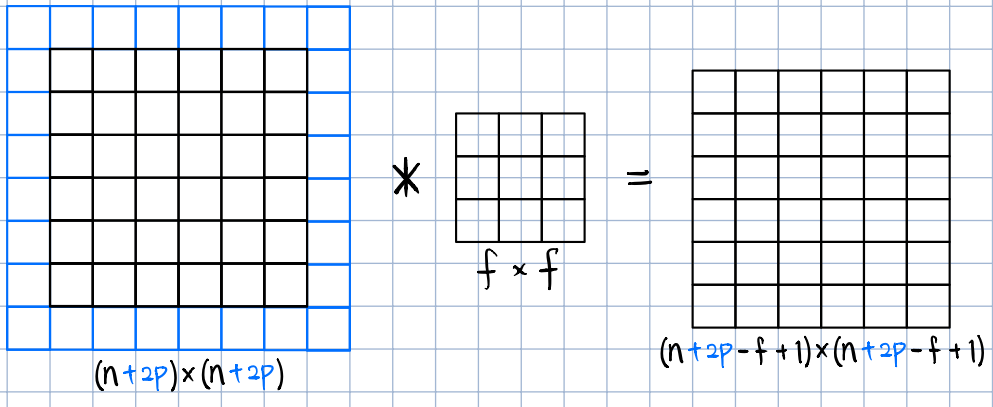
Without Padding



First issue: the output size shrinks.

Second issue: the top-left corner pixel is used only once.

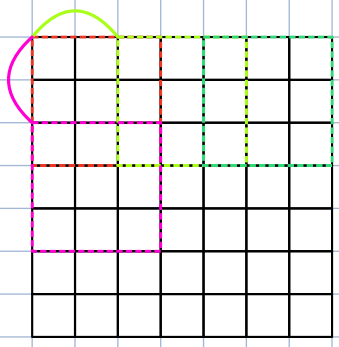
With Padding, $p = 1$



Convolution in Valid mode: No padding

Convolution in Same mode: Pad so that output size is the Same as the input size.

Strided Convolution

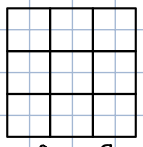


$n \times n$

with padding p

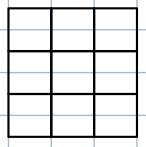
stride s

$*$



$f \times f$

$=$



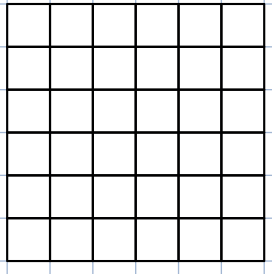
$$\left\lfloor \left(\frac{n+2p-f}{s} + 1 \right) \right\rfloor \times \left\lfloor \left(\frac{n+2p-f}{s} + 1 \right) \right\rfloor$$

→ Take the floor

Cross-correlation v.s. Convolution

In theory, all convolution operation described above is actually cross-correlation.

The mathematical definition of convolution is the following:



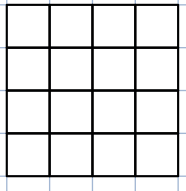
Convolution $*$

3	4	5
1	0	2
-1	9	7

The kernel needs to be first flipped vertically and horizontally. And then does element-wise multiplication.

7	9	-1
2	0	1
5	4	3

$=$

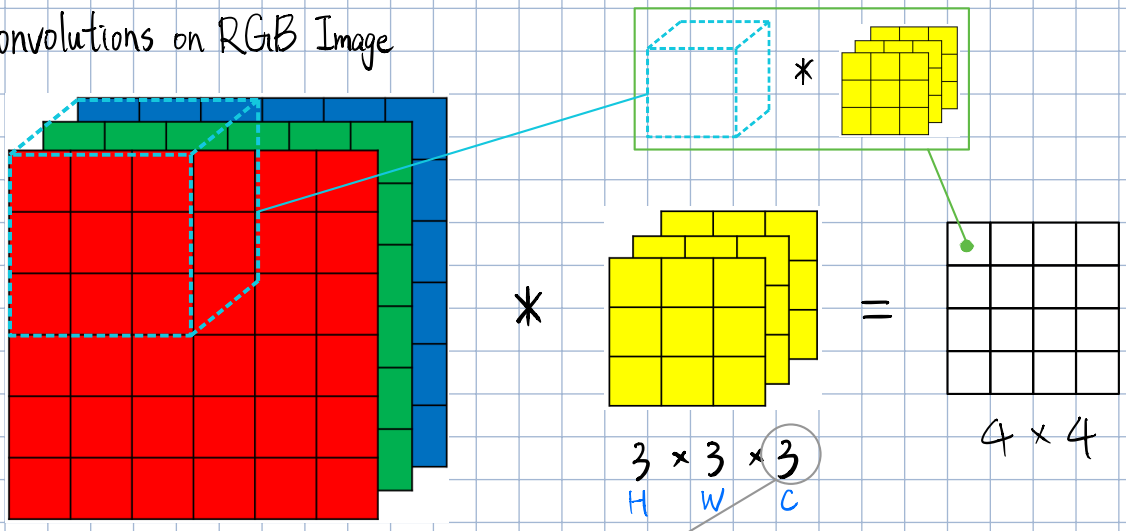


However, many papers and Andrew don't bother call cross-correlation convolution.

Notice, in the signal processing world, convolution SHALL not be confused with cross-correlation.

Convolutions Over Volume

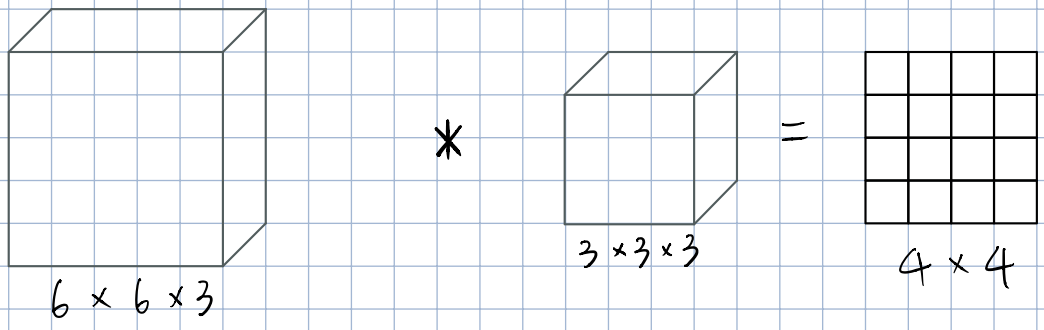
Convolutions on RGB Image



6 × 6 × 3

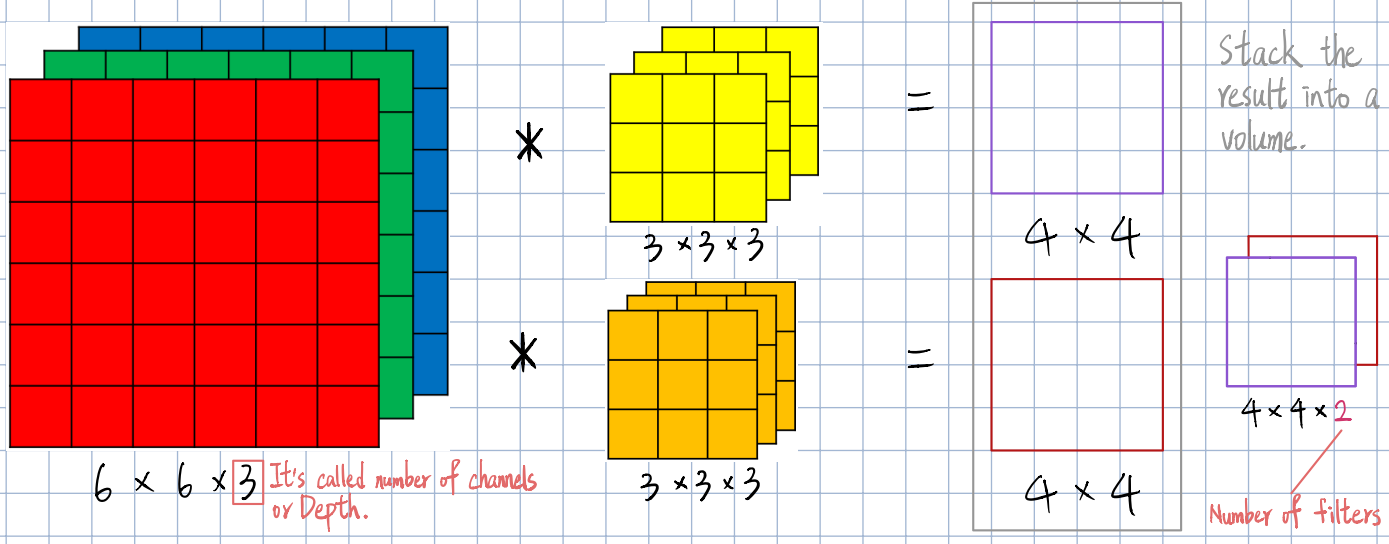
Height Width # Channels

These two numbers have to be equal. (Channel-wise convolution)
The 3×3×3 filter has 27 parameters to train.



Using Multiple Filters

Use two filters

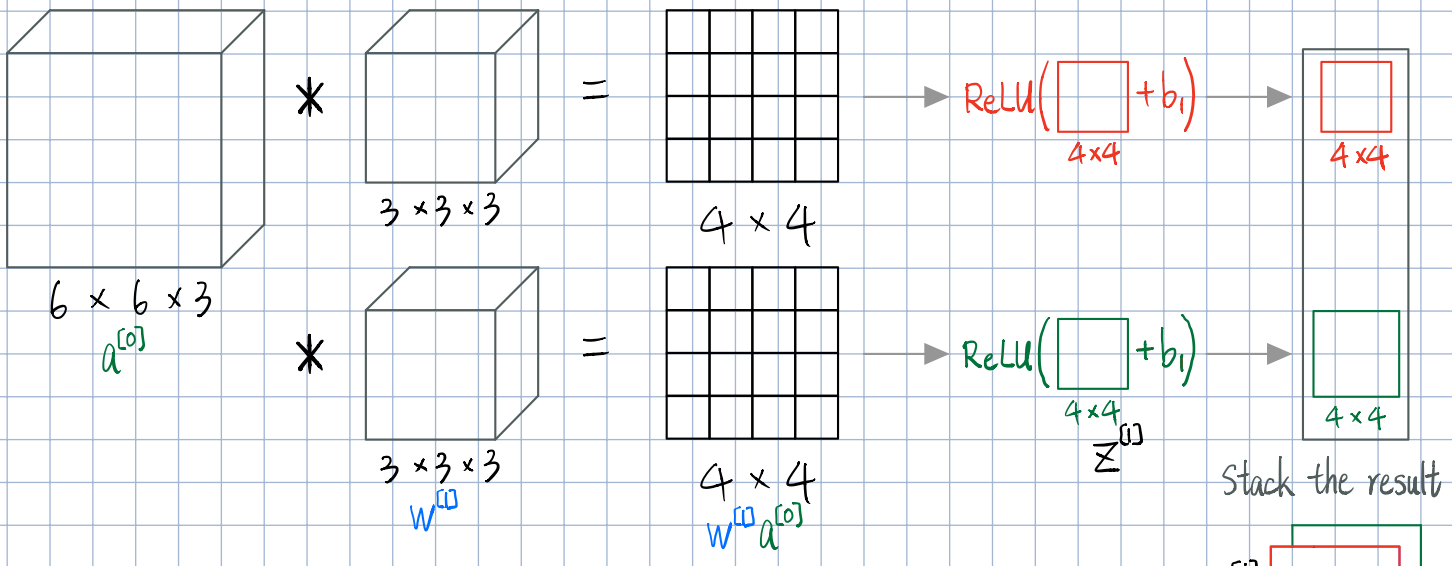


6 × 6 × 3 It's called number of channels or Depth.

Stack the result into a volume.

Number of filters

One Layer of a Convolutional Network



Recall $z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = g(z^{[1]})$

Notice $a^{[1]}$ is $4 \times 4 \times 2$ due to the fact that two filters are used. If ten filters are used, then $a^{[1]}$ would be $4 \times 4 \times 10$.

Quick Quiz: If you have 10 filters that are $3 \times 3 \times 3$ in one layer of a neural network, how many parameters does that layer have?

Answer: $(3 \times 3 \times 3 + 1 \text{ bias}) \times 10 = 280$ parameters ※

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

Activations $a^{[l]}$: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ Batch $A^{[l]}$: $m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ (m: number of samples)

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

bias: $n_c^{[l]}$, shape $(1, 1, 1, n_c^{[l]})$

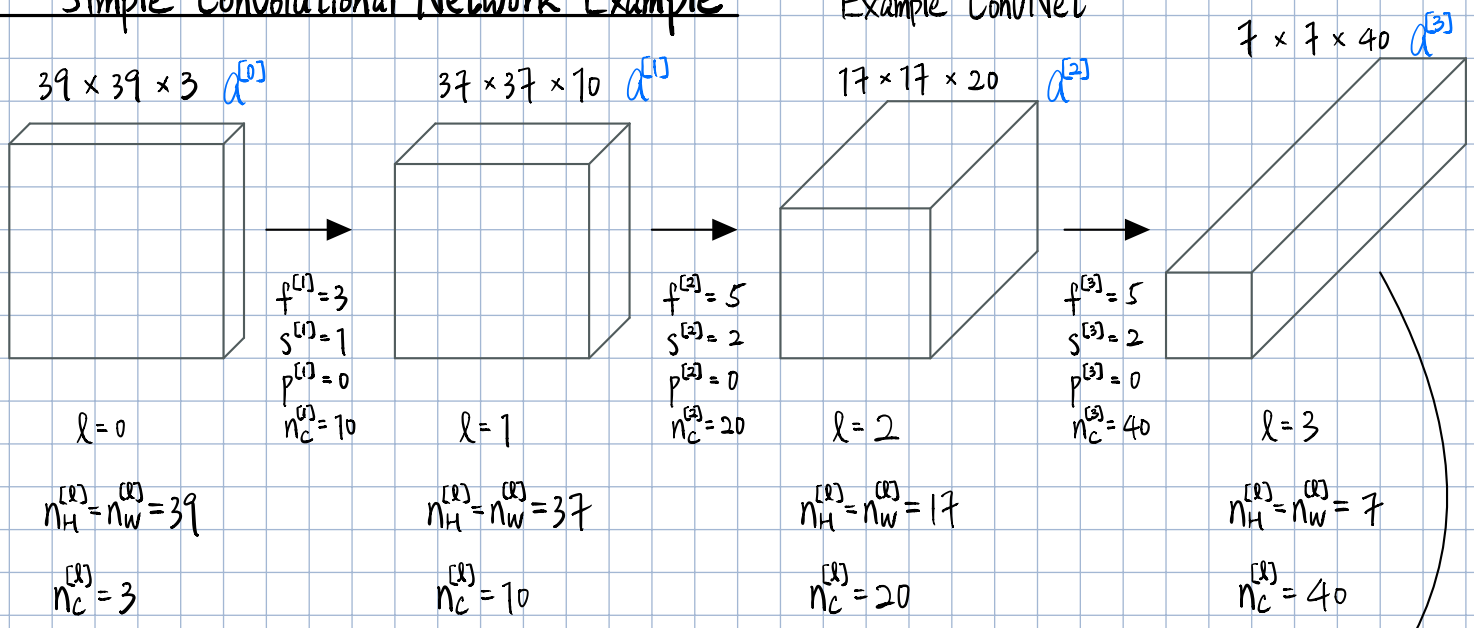
Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$ # channels in the previous $l-1$ layer

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

$$n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \rightarrow \text{floor}$$

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

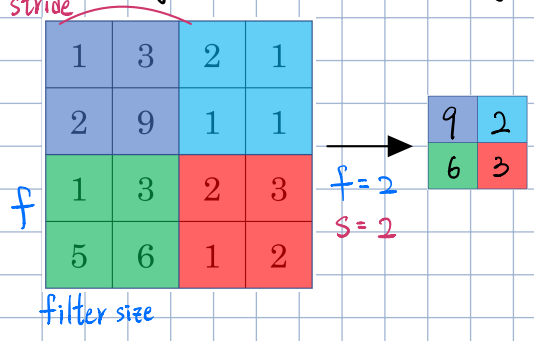
Simple Convolutional Network Example

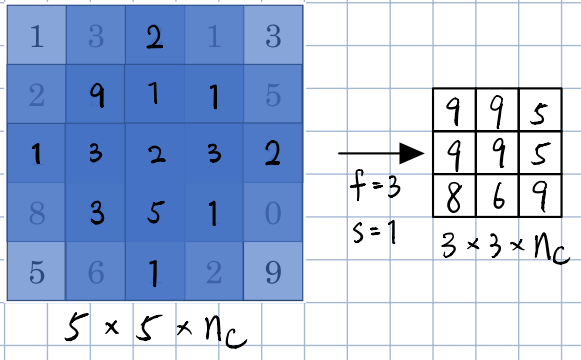


Types of layer in a convolutional network:

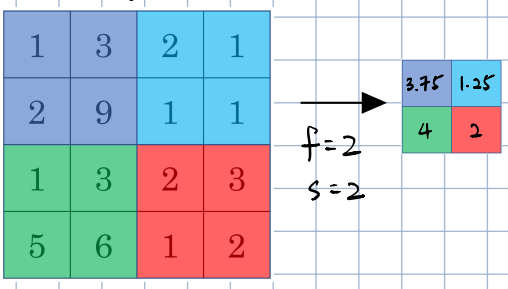
- Convolution (Conv)
- Pooling (Pool)
- Fully Connected (FC)

Pooling Layer : Max Pooling





Pooling Layer : Average Pooling



Doing pooling on 3D data is very similar to 2D.

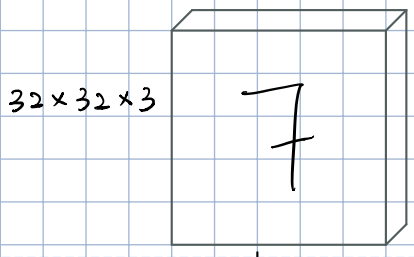
We just need to do pooling on each channel separately.

There are no parameters to learn in Pooling Layer.

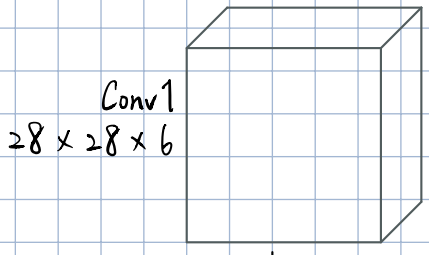
Convolutional Neural Network (CNN) Example

Inspired by LeNet-5

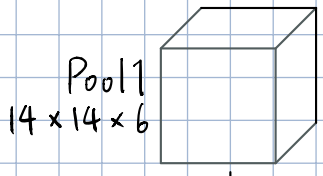
Goal: Build a neural network that can recognize 10 digits from 0 to 9.



f = 5
s = 1



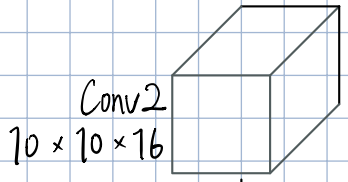
MaxPooling f = 2 s = 2



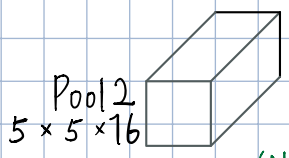
It's considered as the layer 1. The pooling layer doesn't have weights to train. Thus, the convolutional layer and the pooling layer are considered as one layer.

f = 5
s = 1

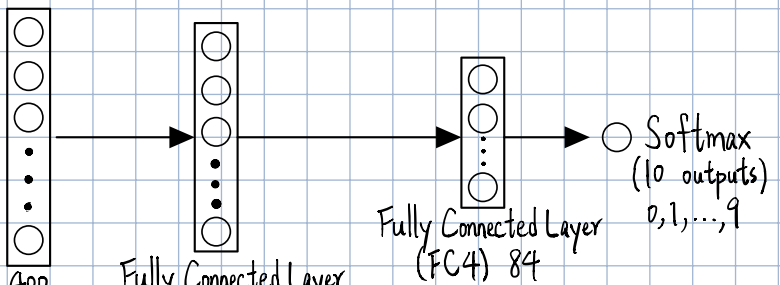
Layer 2



MaxPooling f = 2 s = 2



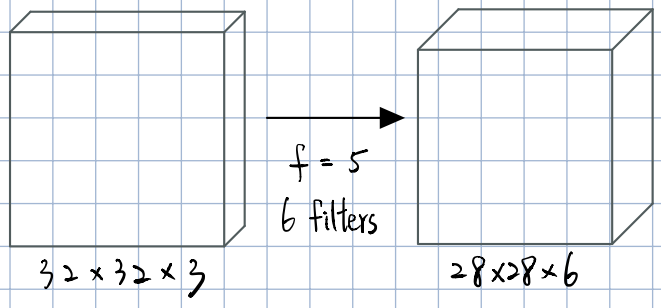
Flattening
(No parameters here)



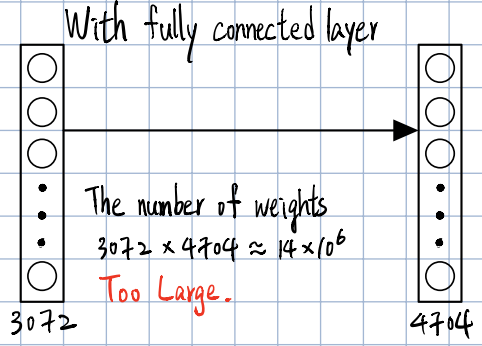
$W^{[3]}$: (120, 400)
$b^{[3]}$: (120,)

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 $a^{[0]}$	0
CONV1 (f=5, s=1)	(28,28,8)	~~6,272~~ 4704	~~208~~ 456 (5×5×3×6+6)
POOL1	(14,14,8)	~~1,568~~ 1176	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	~~416~~ 2416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	~~48,001~~ 48120 (120×400+120)
FC4	(84,1)	84	~~10,081~~ 10164 (84×120+84)
Softmax	(10,1)	10	~~841~~ 850

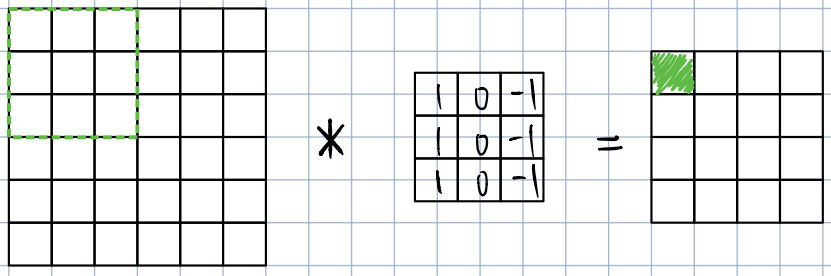
Why Convolutions?



With filters, there will only be $(5 \times 5 \times 3 + 1) \times 6$ weights to learn.



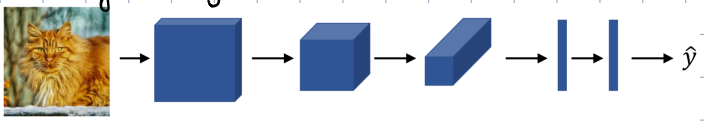
Quiz: The sparsity of connections and weight sharing are mechanisms that allow us to use fewer parameters in a CNN making it possible to train a network with smaller training set. Hmmm...



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. Only one filter/kernel is needed to run through the whole image to find features.

Sparsity of connections: In each layer, each output value depends only on a small number of inputs. The output pixel on the top-left corner is only depend on the 9 (3×3) out of 36 (6×6) pixels. The rest 27 (36-9) pixels do not have any effects on the output top-left corner pixel.

Putting it together



Training set: $(x^{(1)}, y^{(1)}) \ldots (x^{(m)}, y^{(m)})$

Cost $J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$. Use gradient descent to optimize parameters to reduce J.