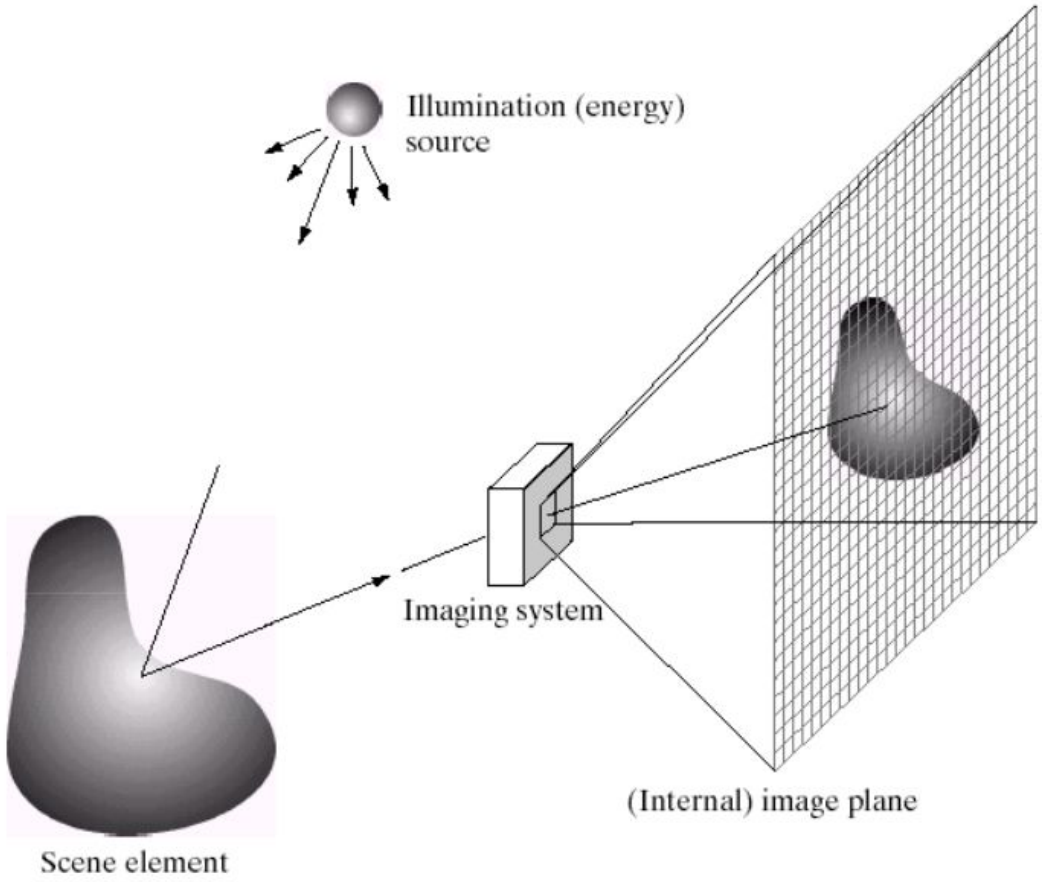
Visión por computadoras

...

#1. Intro a imágenes. Correlación y convolución.
Redes Convolucionales

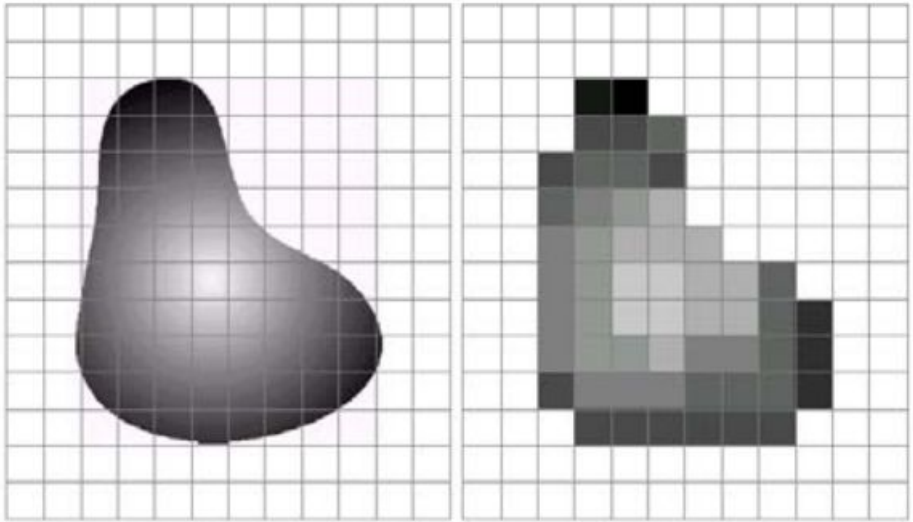
Intro imágenes. Correlación y convolución

Image Formation



slides: D. Hoeim

Digital images

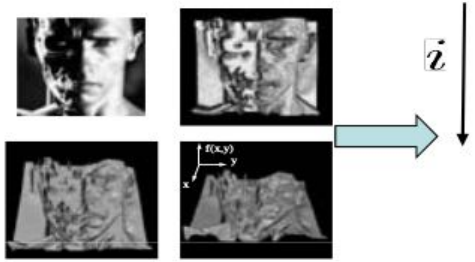


a b

FIGURE 2.17 (a) Continuos image projected onto a sensor array. (b) Result of image sampling and quantization.

Digital images

- **Sample** the 2D space on a regular grid
- **Quantize** each sample (round to nearest integer)
- Image thus represented as a matrix of integer values.

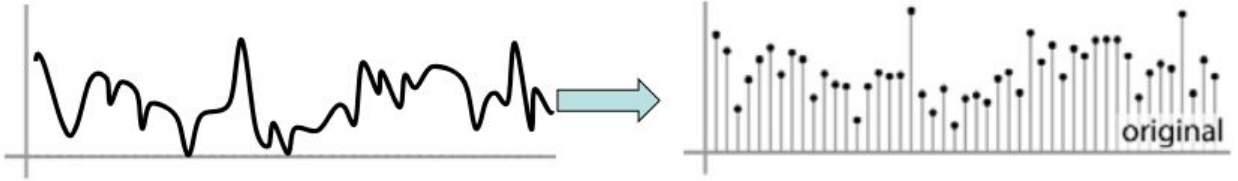


j

i

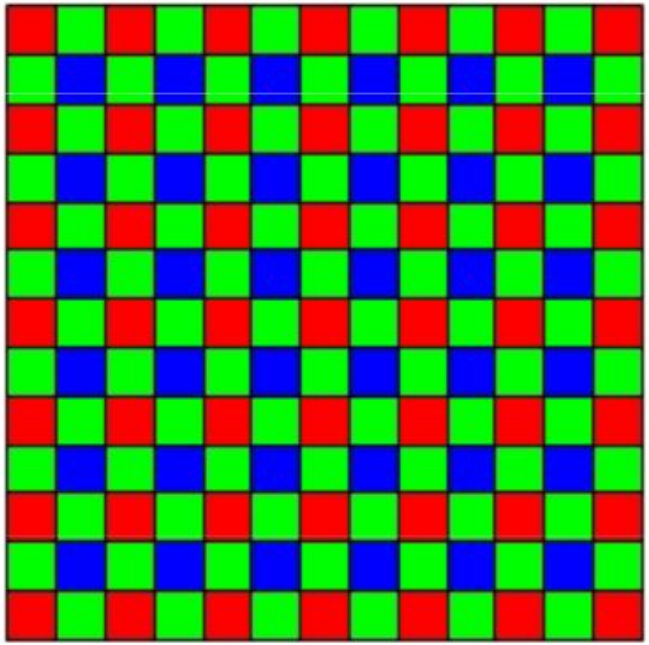
62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

2D



1D

Digital color images



Bayer filter

Digital color images

Color images, RGB color space

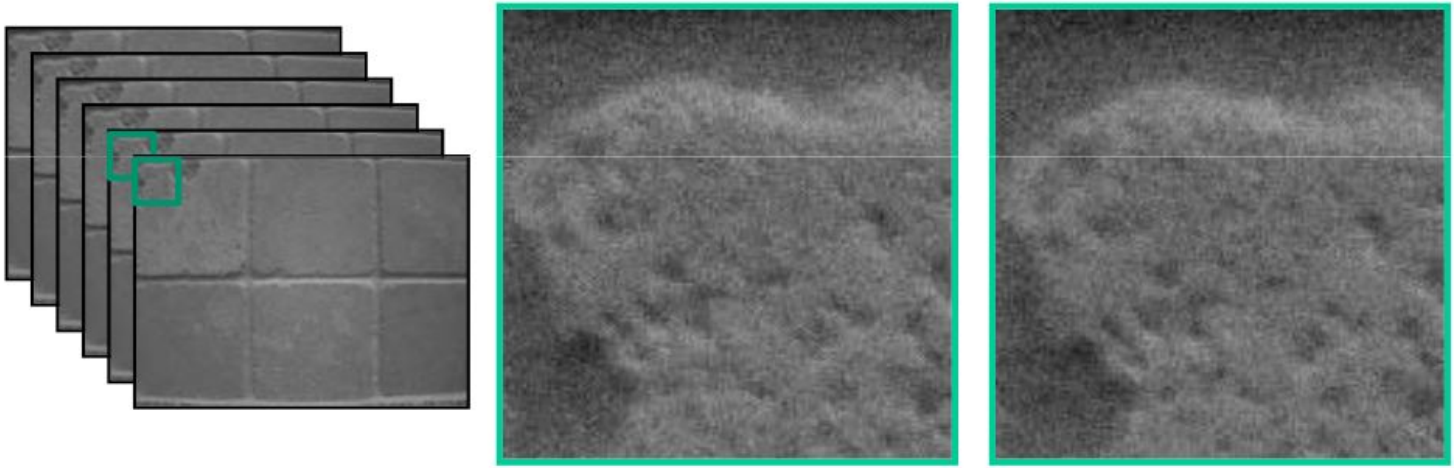
R

G

B

slides: K. Grauman

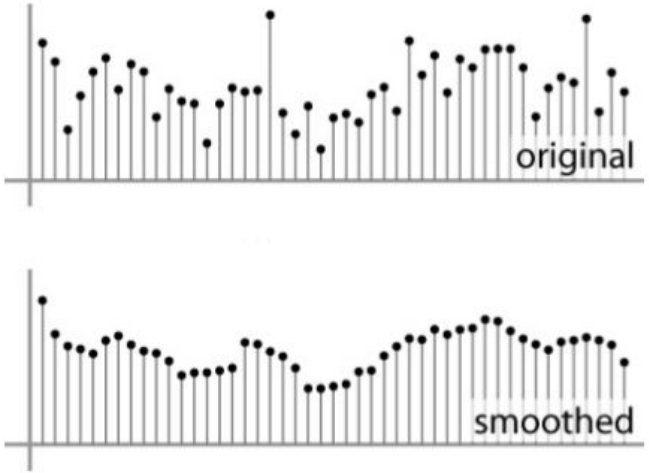
Motivation: noise reduction



- Even multiple images of the same static scene will not be identical.
- How could we reduce the noise, i.e., give an estimate of the true intensities?
- **What if there's only one image?**

First attempt at a solution

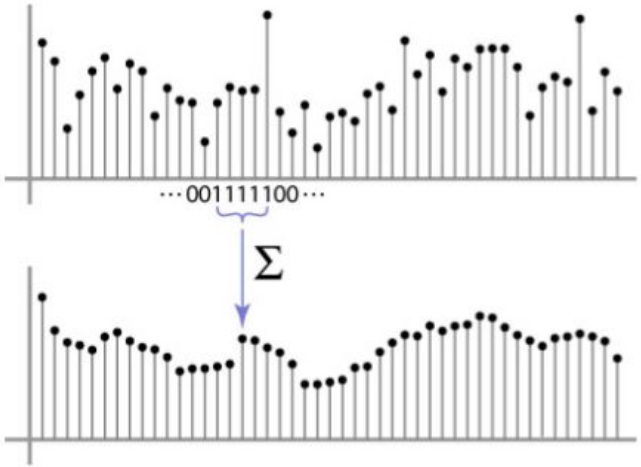
- Let's replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:



Weighted Moving Average

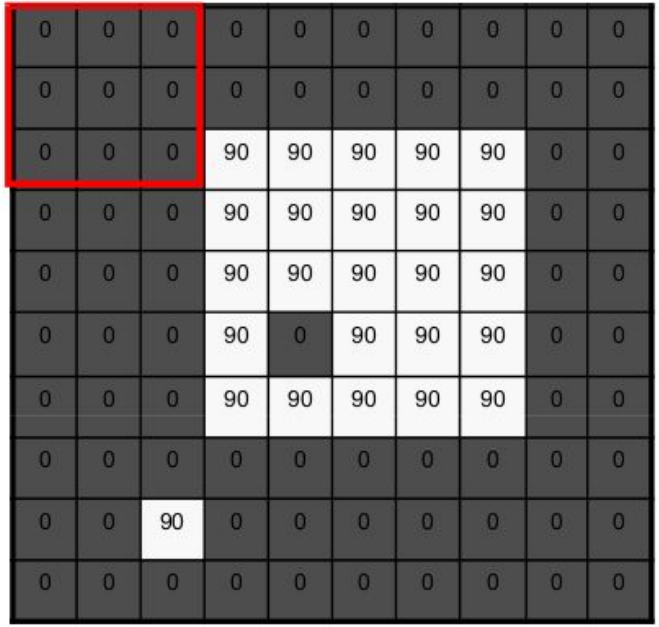
Can add weights to our moving average

Weights [1, 1, 1, 1, 1] / 5

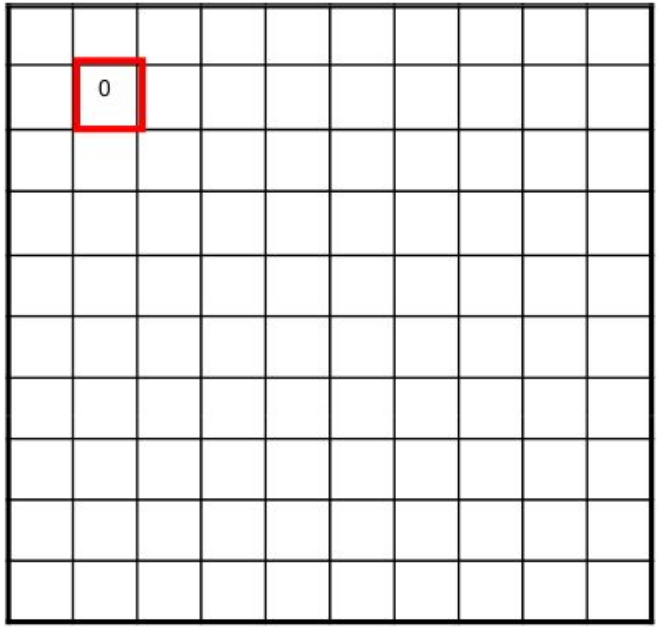


Moving Average In 2D

$F[x, y]$

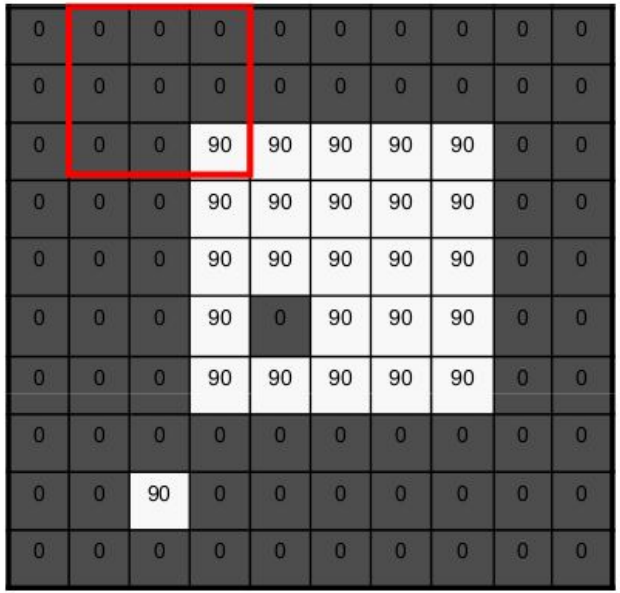


$G[x, y]$

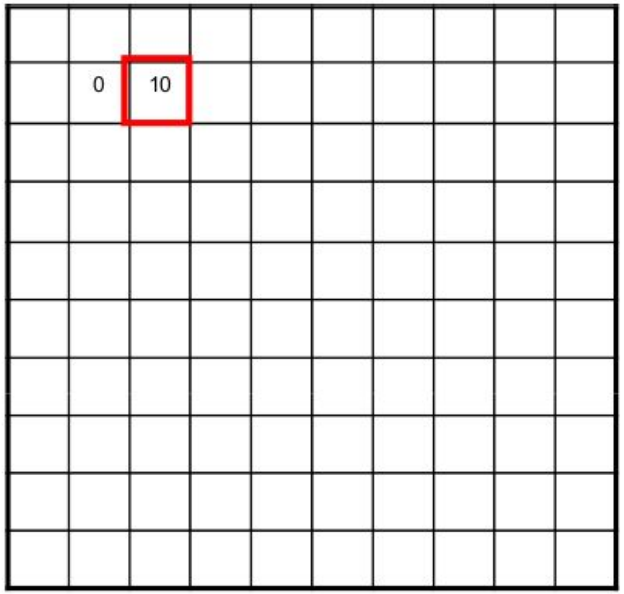


Moving Average In 2D

$F[x, y]$

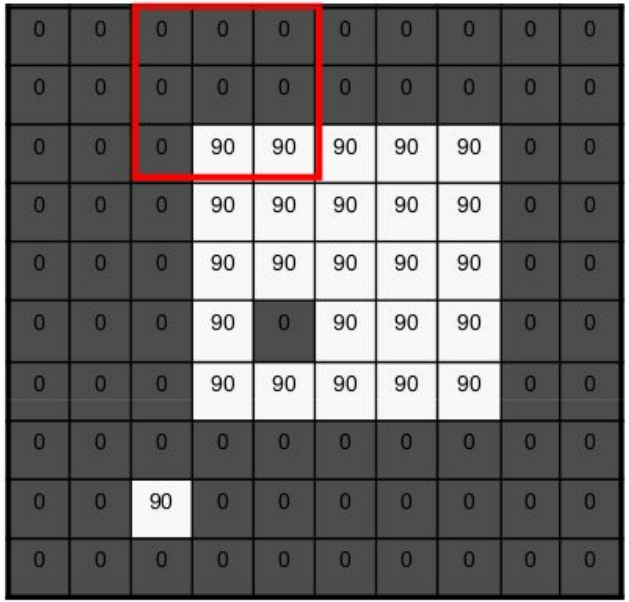


$G[x, y]$

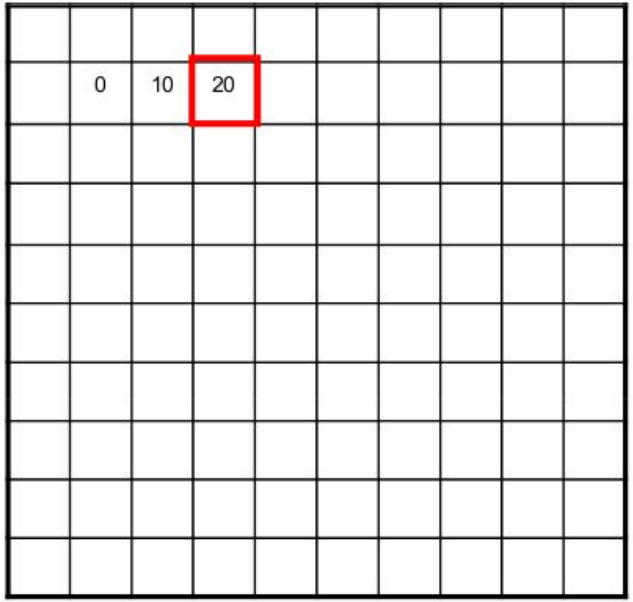


Moving Average In 2D

$F[x, y]$

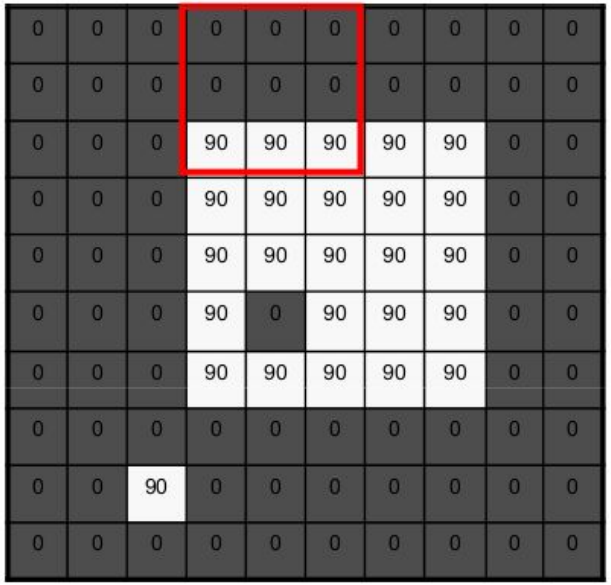


$G[x, y]$

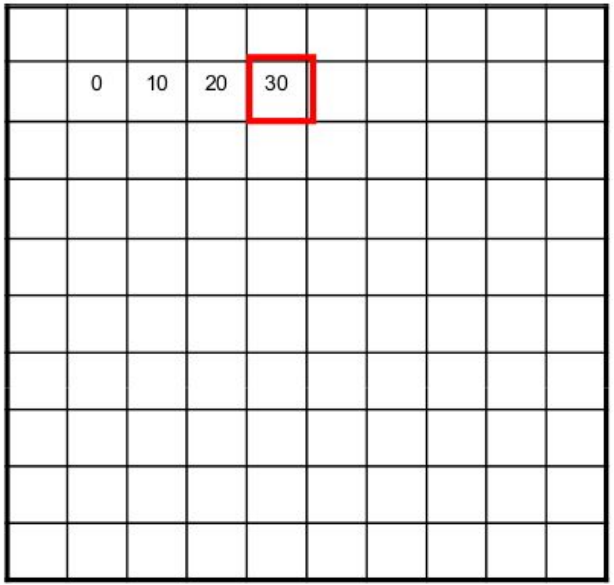


Moving Average In 2D

$F[x, y]$

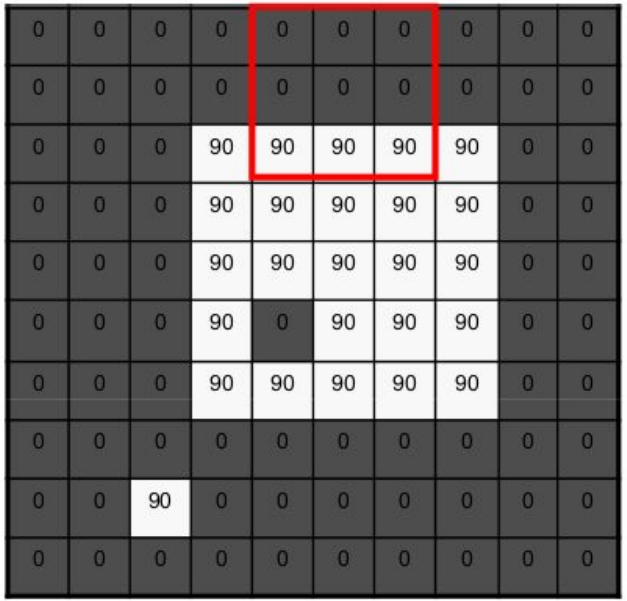


$G[x, y]$

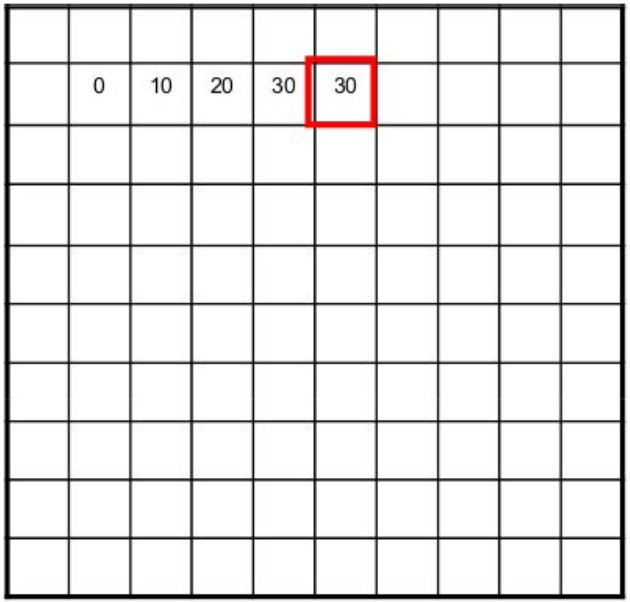


Moving Average In 2D

$F[x, y]$



$G[x, y]$



Moving Average In 2D

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Correlation filtering

Say the averaging window size is 2k+1 x 2k+1:

$$G[i,j] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i+u, j+v]}_{\text{Loop over all pixels in neighborhood around image pixel F[i,j]}}$$

Now generalize to allow **different weights** depending on neighboring pixel's relative position:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\text{Non-uniform weights}} F[i+u, j+v]$$

Correlation filtering

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

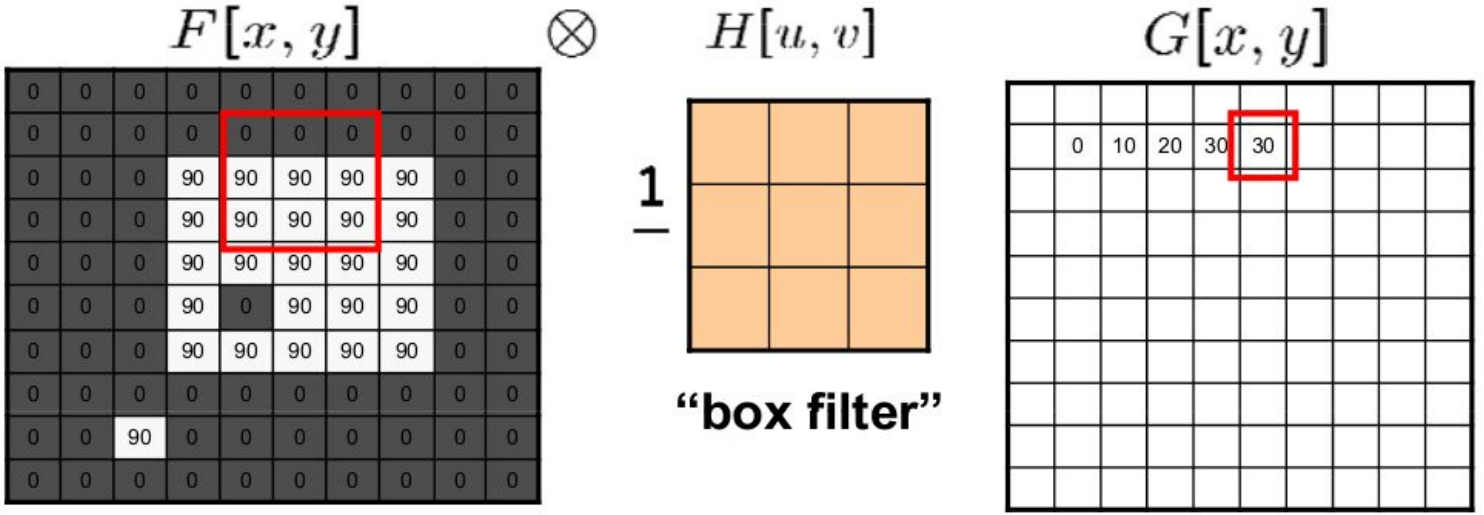
This is called **cross-correlation**, denoted $G = H \otimes F$

Filtering an image: replace each pixel with a linear combination of its neighbors.

The filter "**kernel**" or "**mask**" *H*[*u,v*] is the prescription for the weights in the linear combination.

Averaging filter

- What values belong in the kernel *H* for the moving average example?

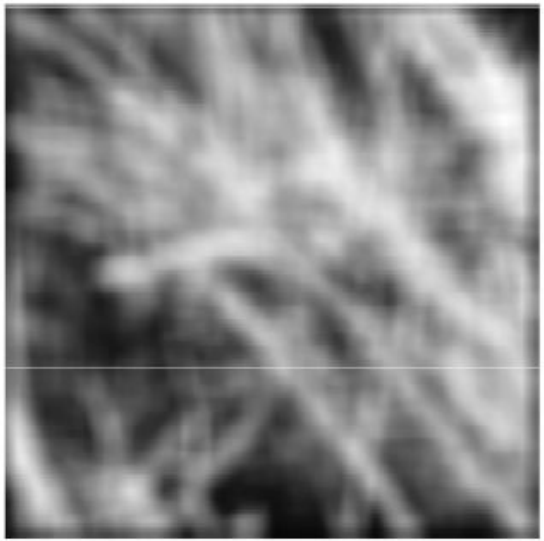


$$G = H \otimes F$$

Smoothing by averaging

depicts box filter:
white = high value, black = low value

original



filtered

What if the filter size was 5 x 5 instead of 3 x 3?

Gaussian filter

- What if we want nearest neighboring pixels to have the most influence on the output?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$

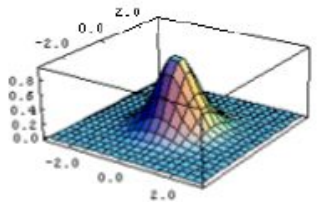
$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

$H[u, v]$

This kernel is an approximation of a 2d Gaussian function:

$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



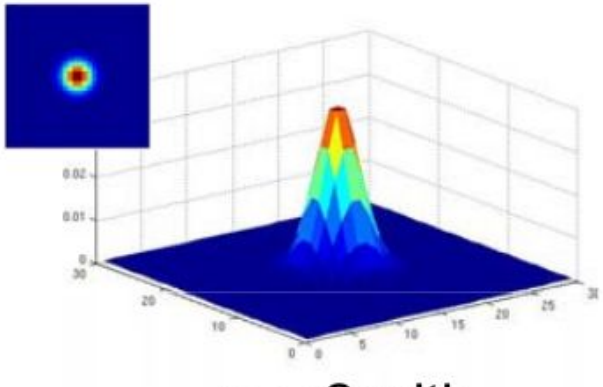
- Removes high-frequency components from the image ("low-pass filter").

slide: S. Seiz

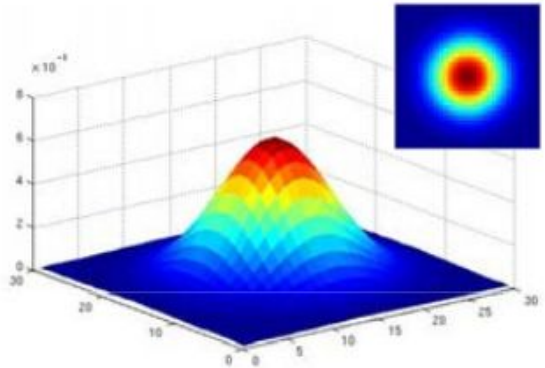
Smoothing with a Gaussian

Gaussian filters

- What parameters matter here?
- **Variance** of Gaussian: determines extent of smoothing

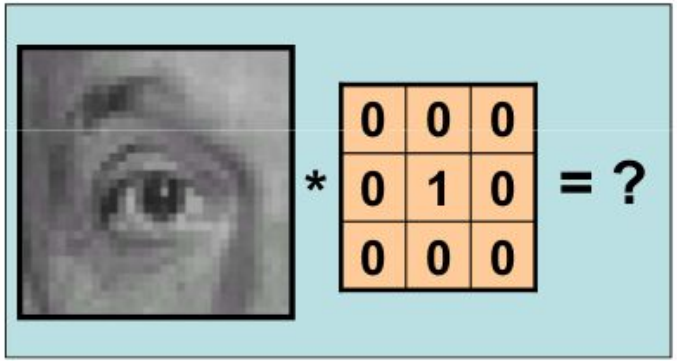


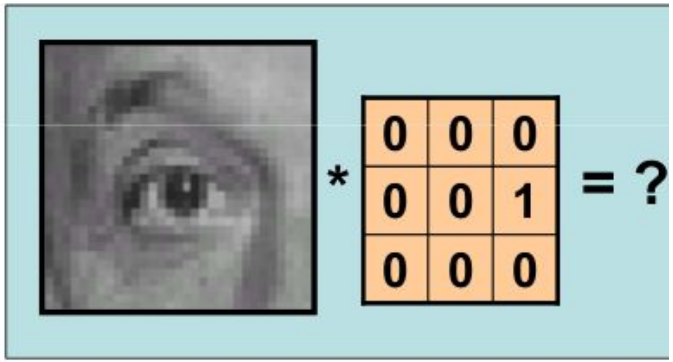
σ = 2 with
30 x 30
kernel

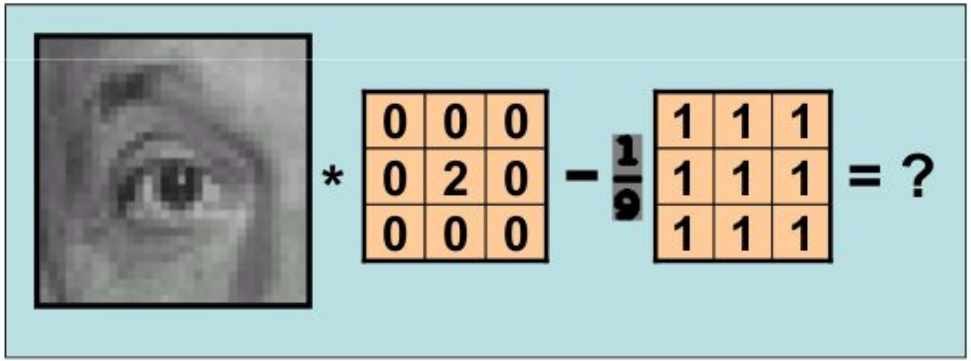


σ = 5 with
30 x 30
kernel

Predict the outputs using correlation filtering







slides: D. Lowe

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Practice with linear filters

Original

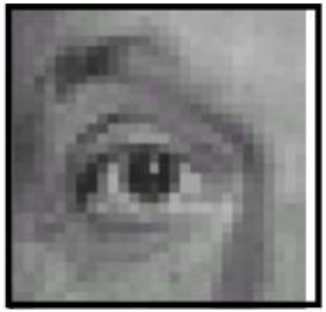
0	0	0
0	0	1
0	0	0

?

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0



Shifted left by 1 pixel with correlation

Practice with linear filters

Original

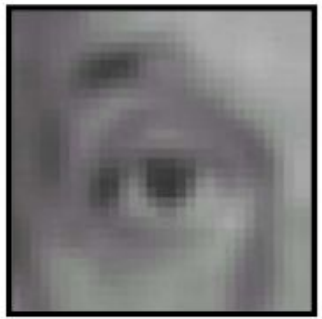
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

?

Practice with linear filters

Original

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Blur (with a box filter)

Practice with linear filters

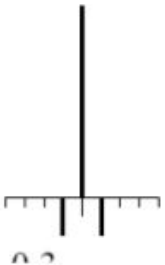
Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = ?$$

Practice with linear filters

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Sharpening filter:
accentuates differences with local average

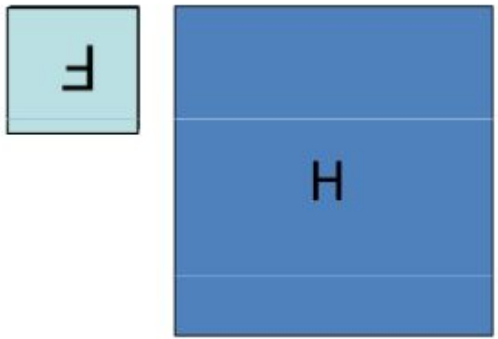
Convolution

- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

$$G = H \star F$$

Notation for convolution operator



Convolution vs. correlation

Convolution

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

$$G = H \star F$$

Cross-correlation

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

$$G = H \otimes F$$

For a Gaussian or box filter, how will the outputs differ?

If the input is an impulse signal, how will the outputs differ?

Shift invariant linear system

- **Shift invariant:**
  - Operator behaves the same everywhere, i.e. the value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood.
- **Linear:**
  - Superposition: h * (f1 + f2) = (h * f1) + (h * f2)
  - Scaling: h * (k f) = k (h * f)

Properties of convolution

- Linear & shift invariant
- Commutative:

  $f * g = g * f$
- Associative

  $(f * g) * h = f * (g * h)$
- Identity:

  unit impulse e = [..., 0, 0, 1, 0, 0, ...]. $f * e = f$
- Differentiation:

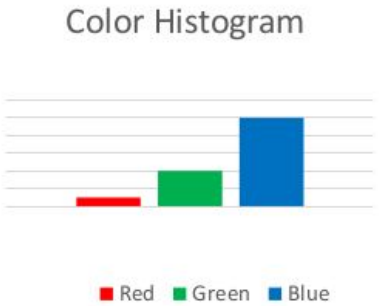
$$\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$$

Redes convolucionales

Features

x

Extract features



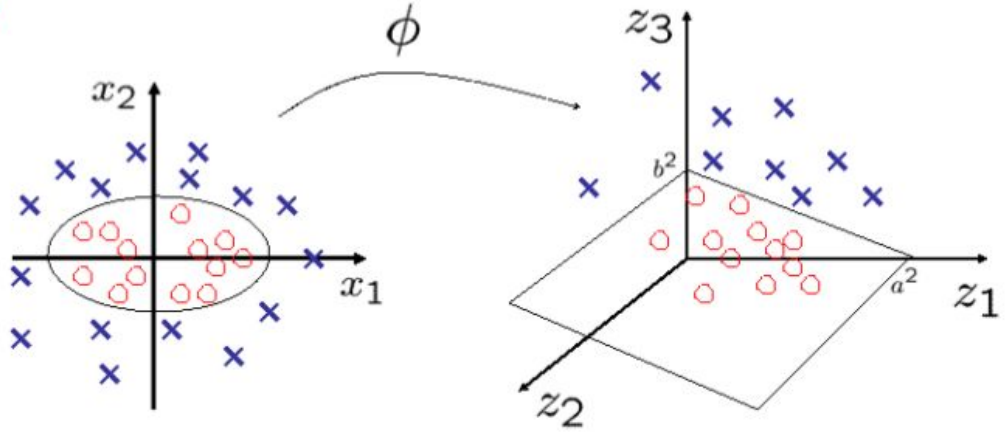
build hypothesis

$$y = w^T \phi(x)$$

slides: Y. Liang

Features: part of the model

Nonlinear model



build hypothesis

$$y = w^T \phi(x)$$

Linear model

Motivation: representation learning

- Why don't we also learn $\phi(x)$?

x

Learn $\phi(x)$

$\phi(x)$

Learn w

$y = w^T \phi(x)$

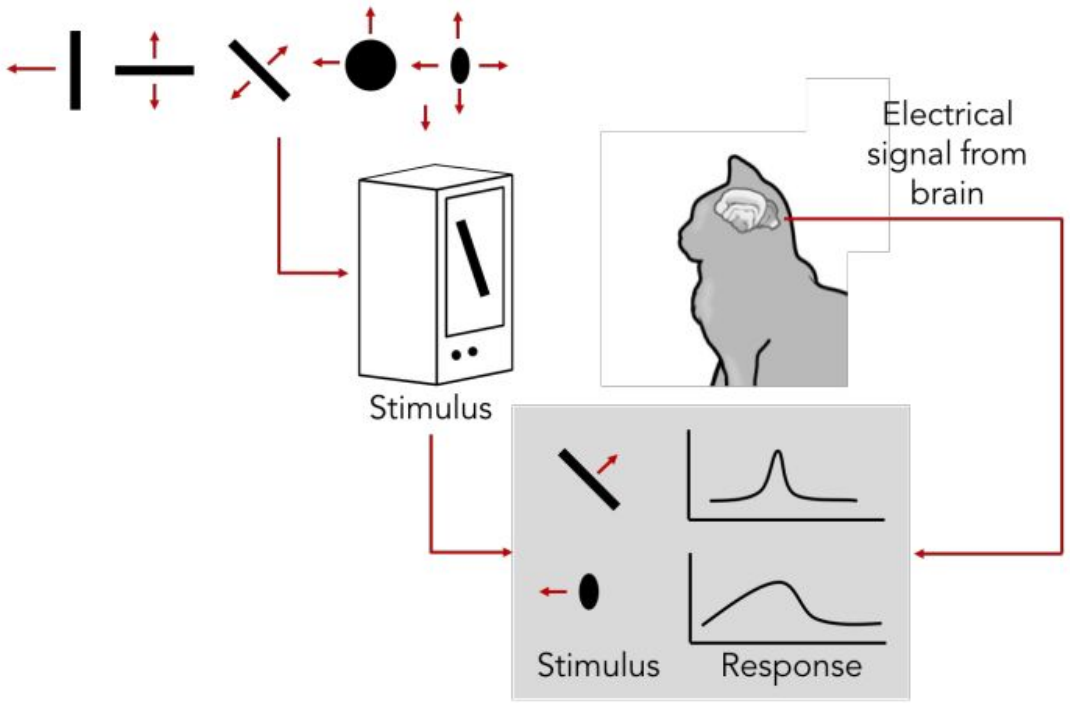
A bit of history:

Hubel & Wiesel, 1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX



Hierarchical organization

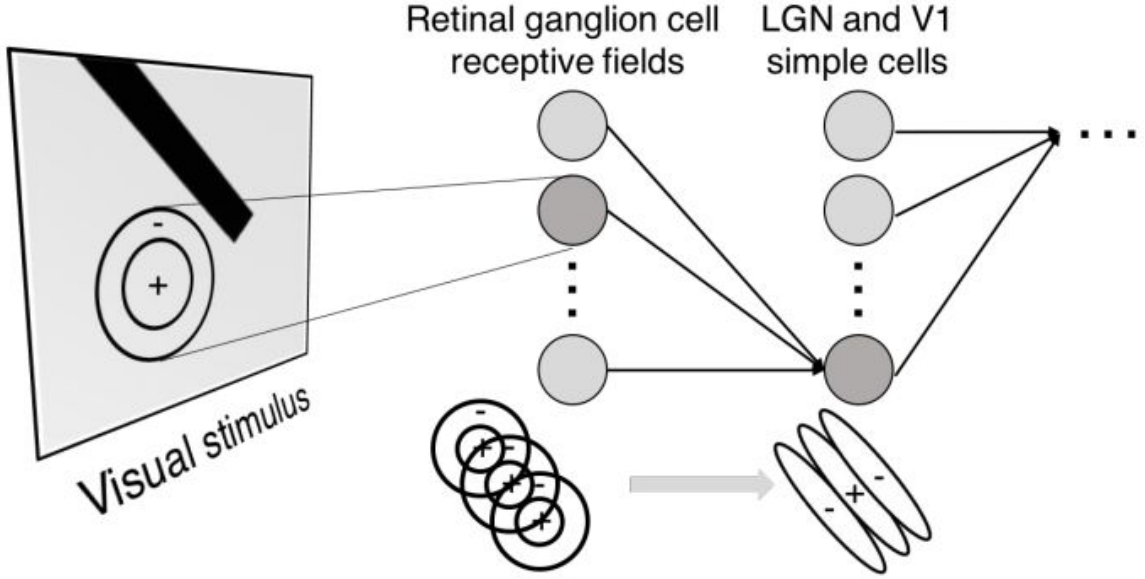
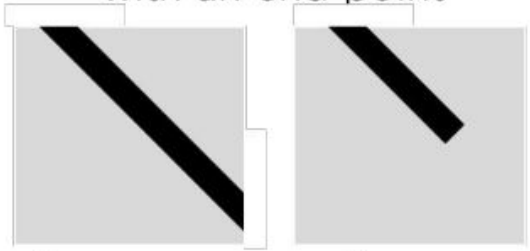


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point

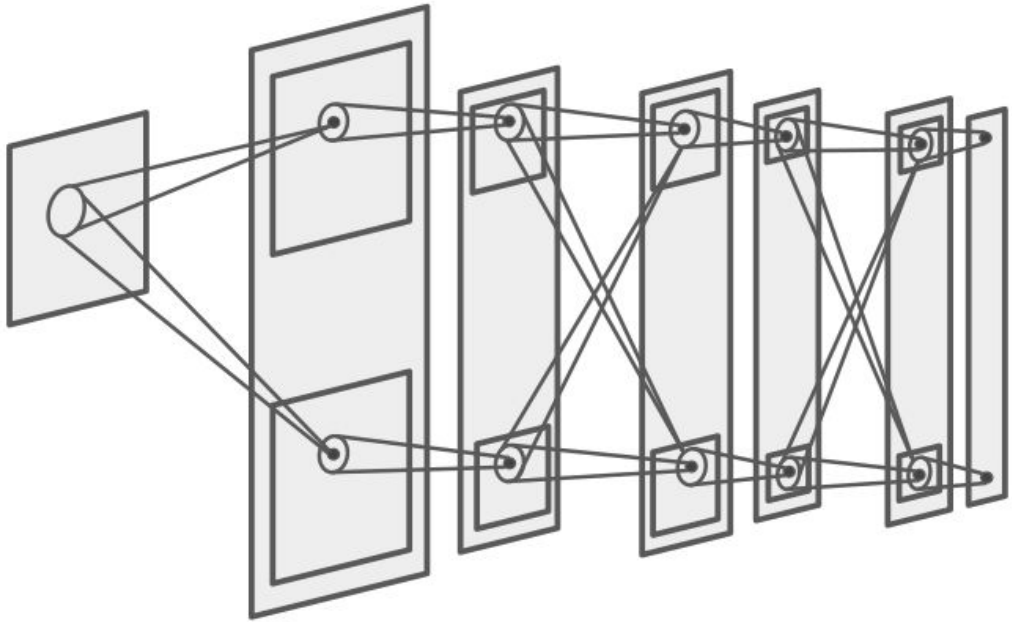


No response

Response (end point)

A bit of history:

Neocognitron
[Fukushima 1980]

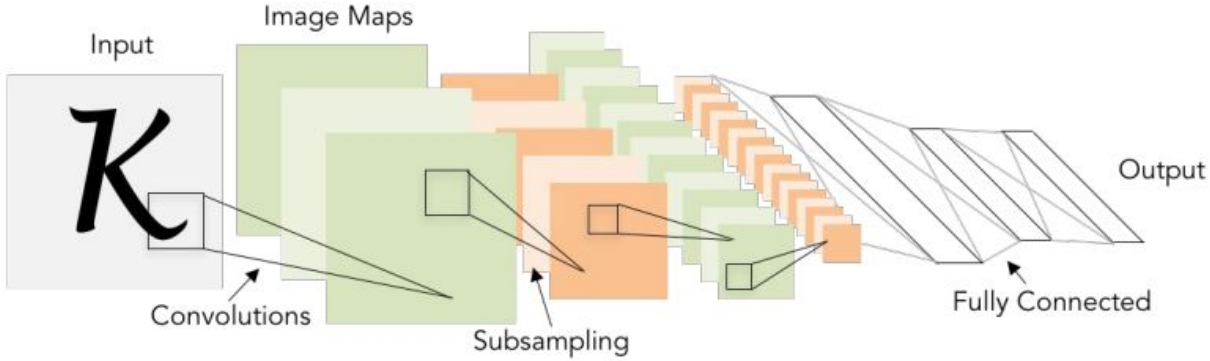


"sandwich" architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

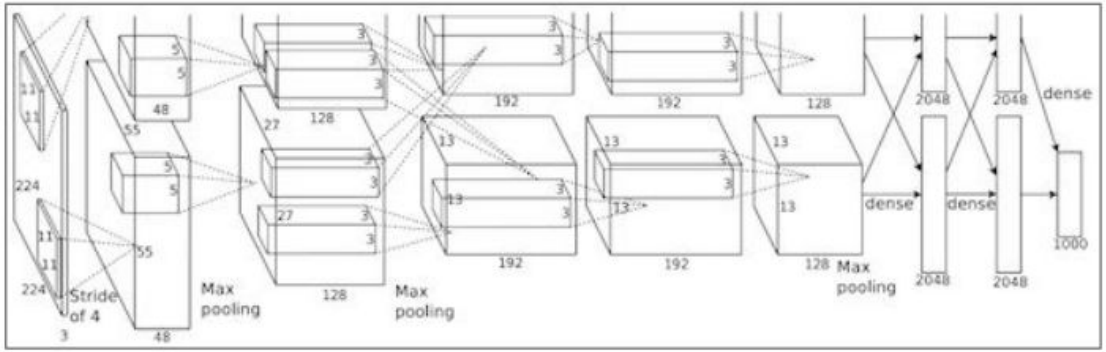


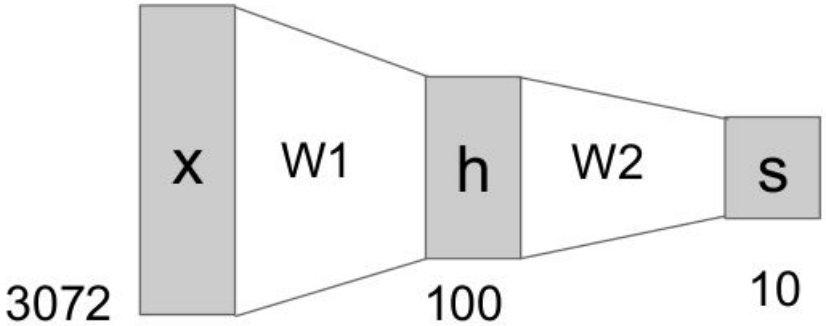
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

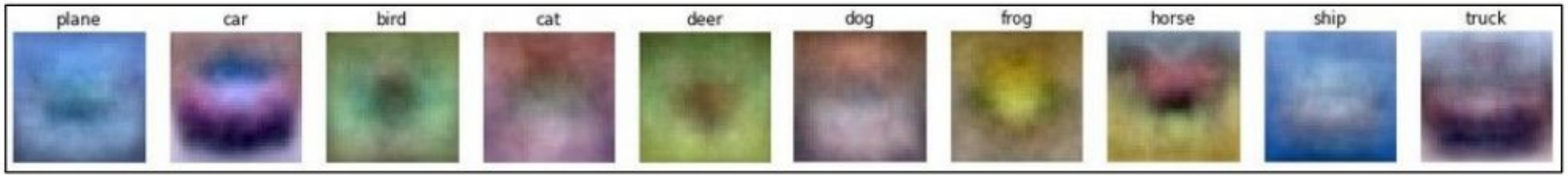
"AlexNet"

Last time: Neural Networks

Linear score function: $f = Wx$

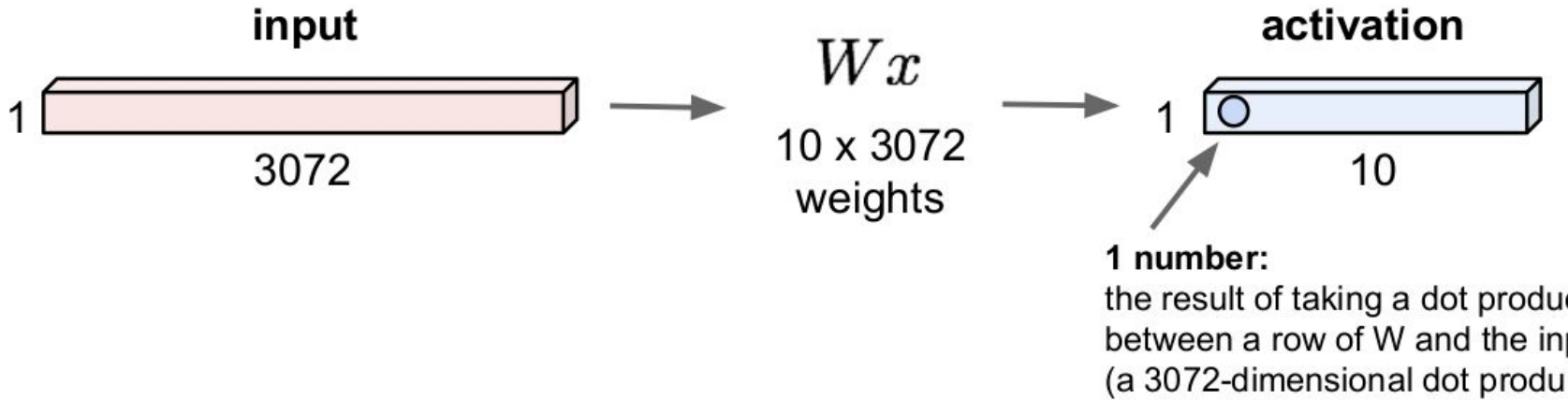
2-layer Neural Network $f = W_2 \max(0, W_1 x)$





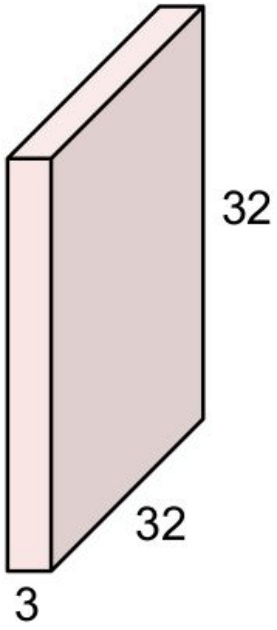
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

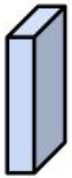


Convolution Layer

32x32x3 image



5x5x3 filter

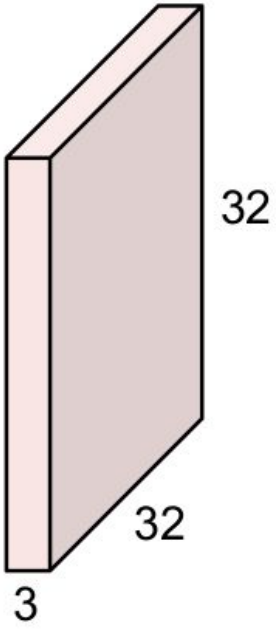


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

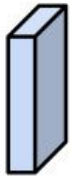
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

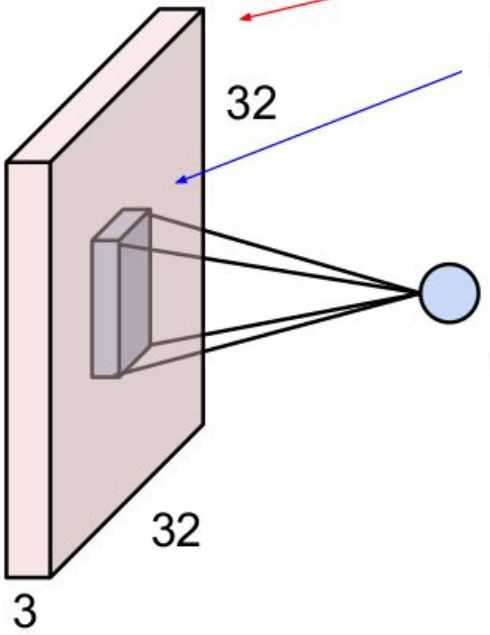


5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer



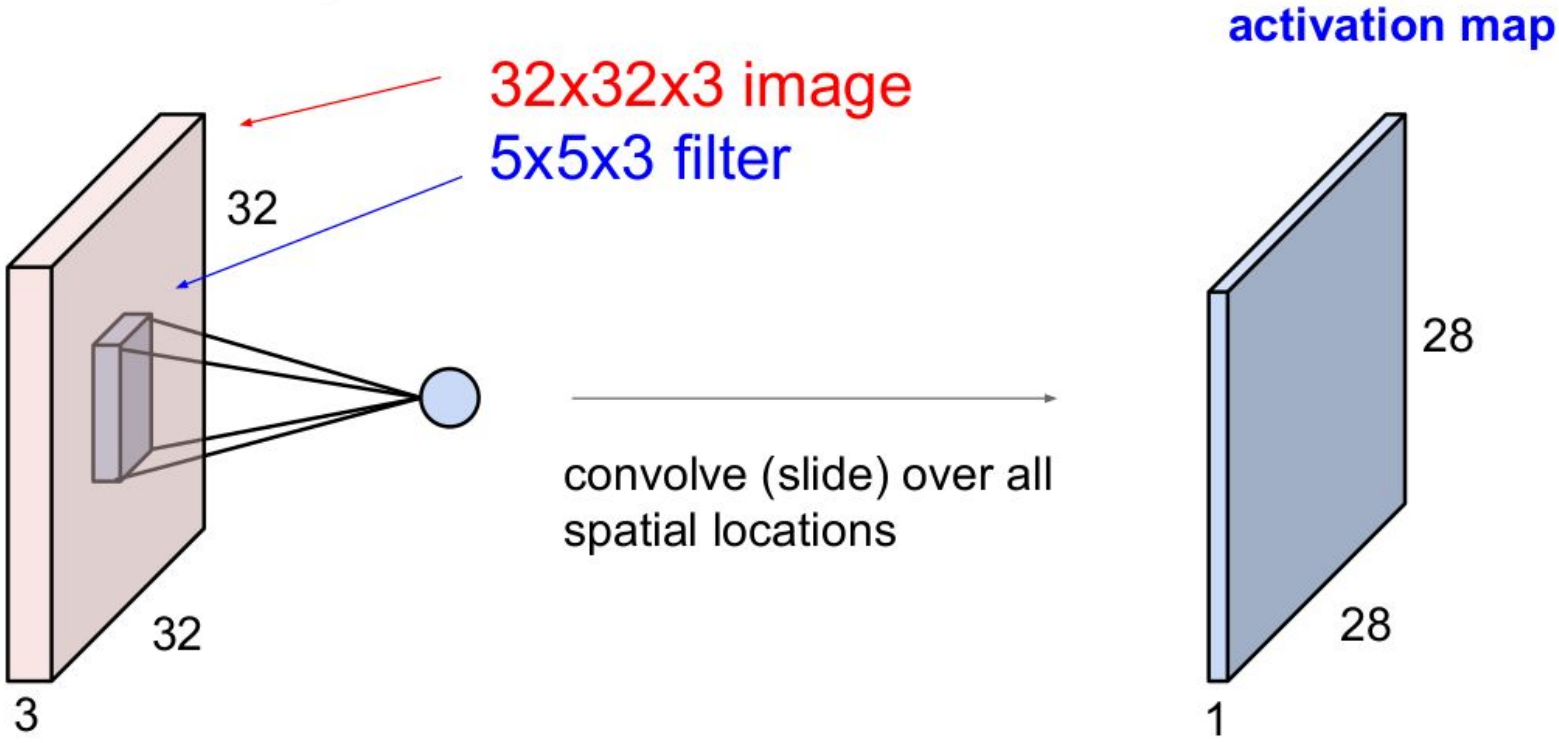
32x32x3 image

5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

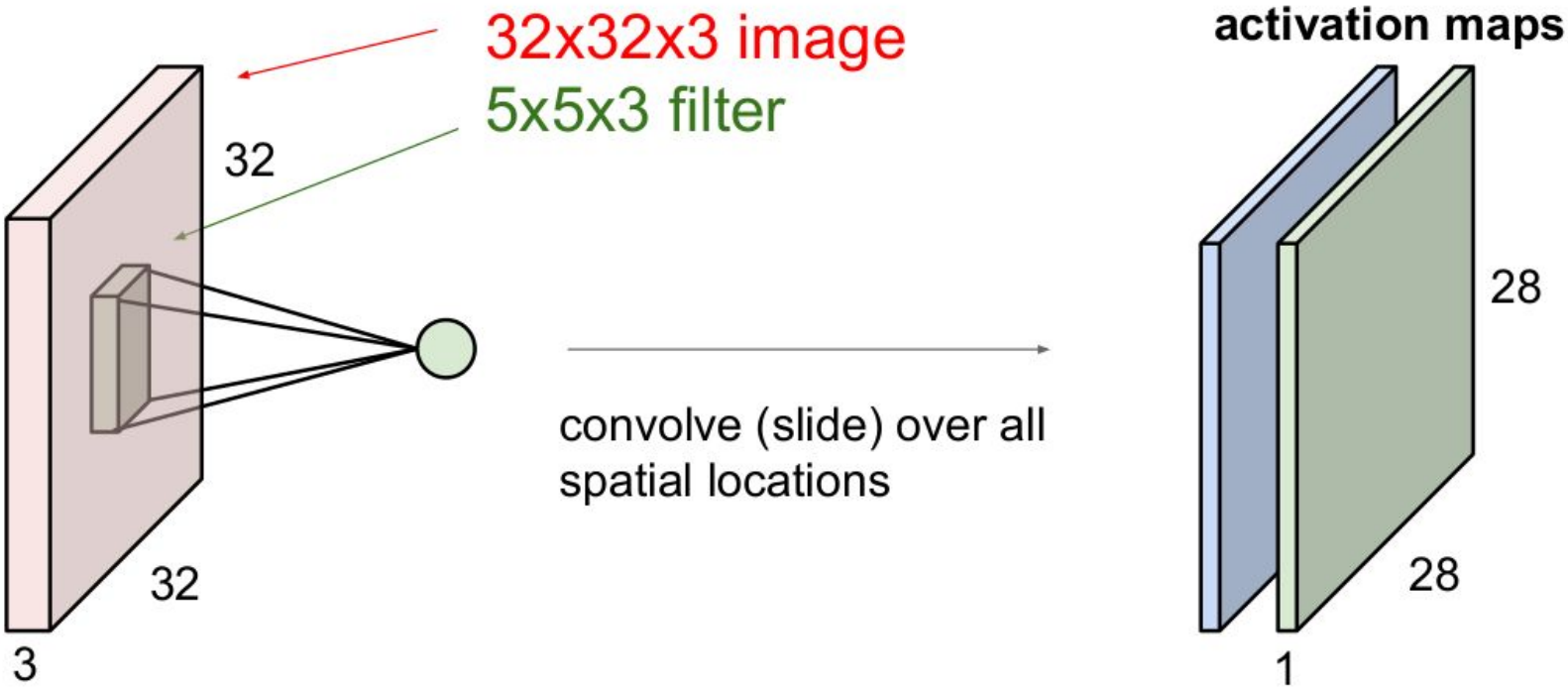
$$w^T x + b$$

Convolution Layer

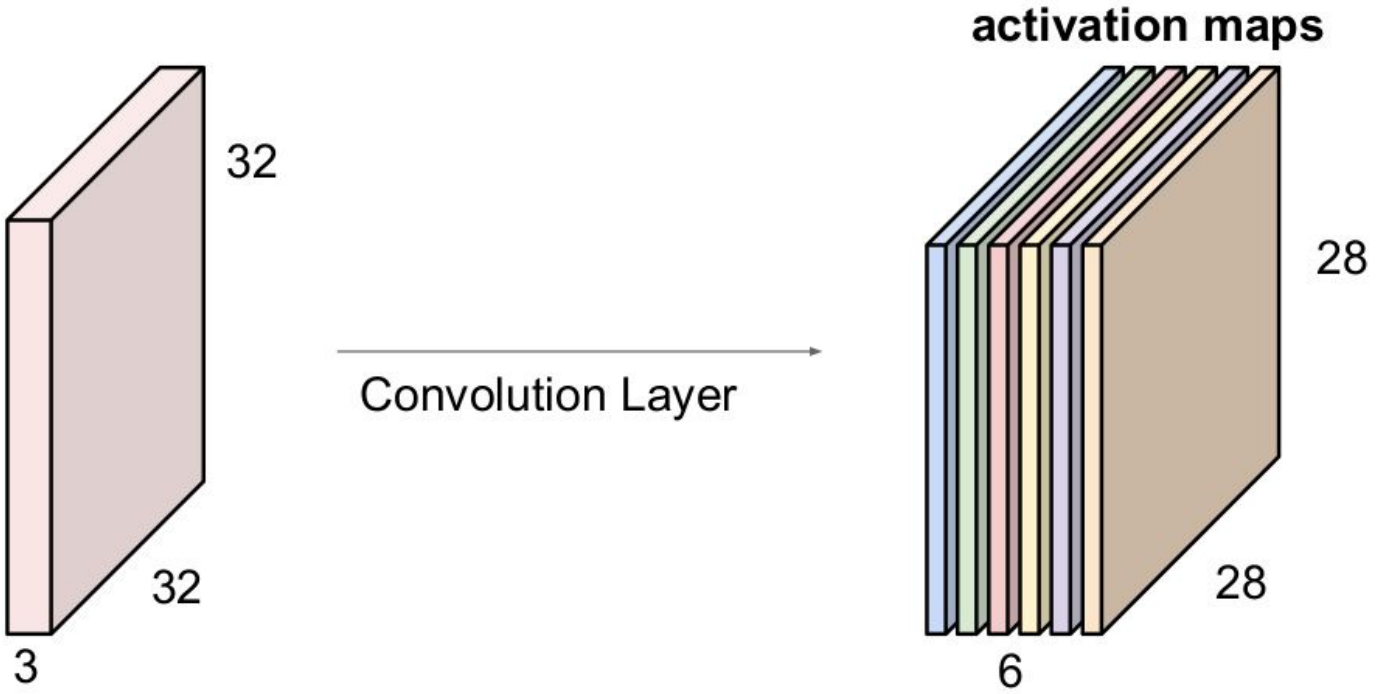


Convolution Layer

consider a second, green filter

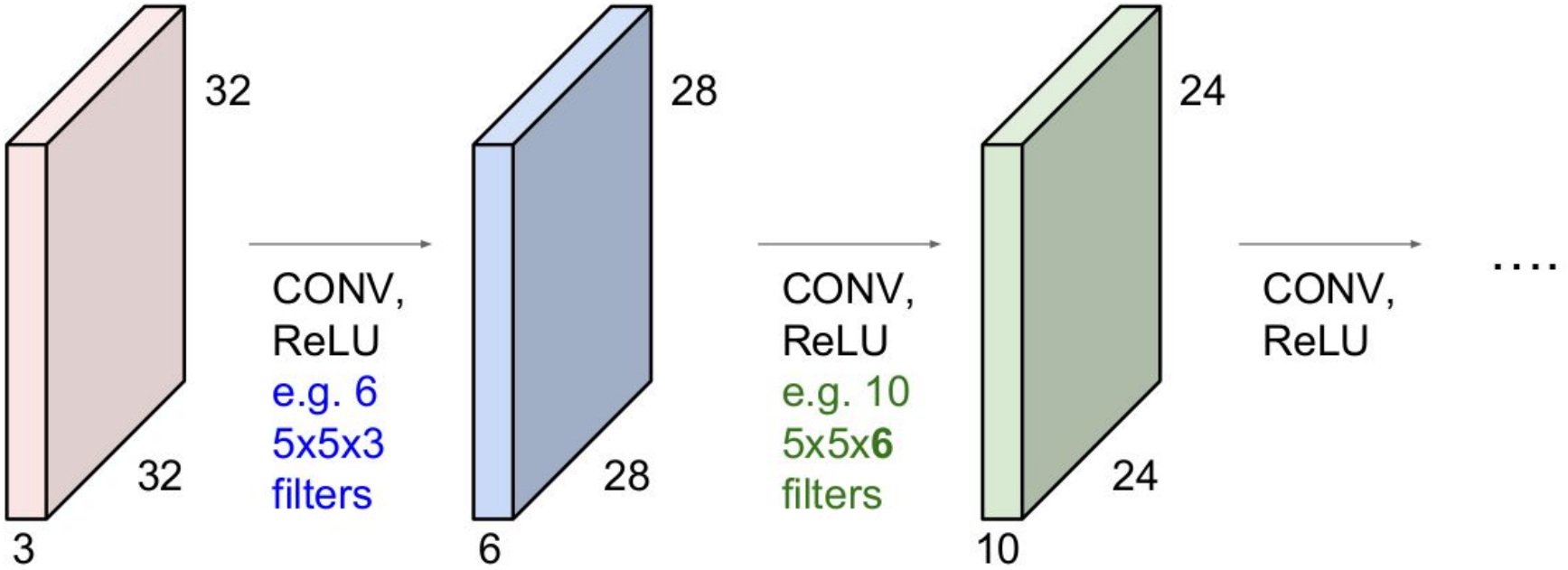


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

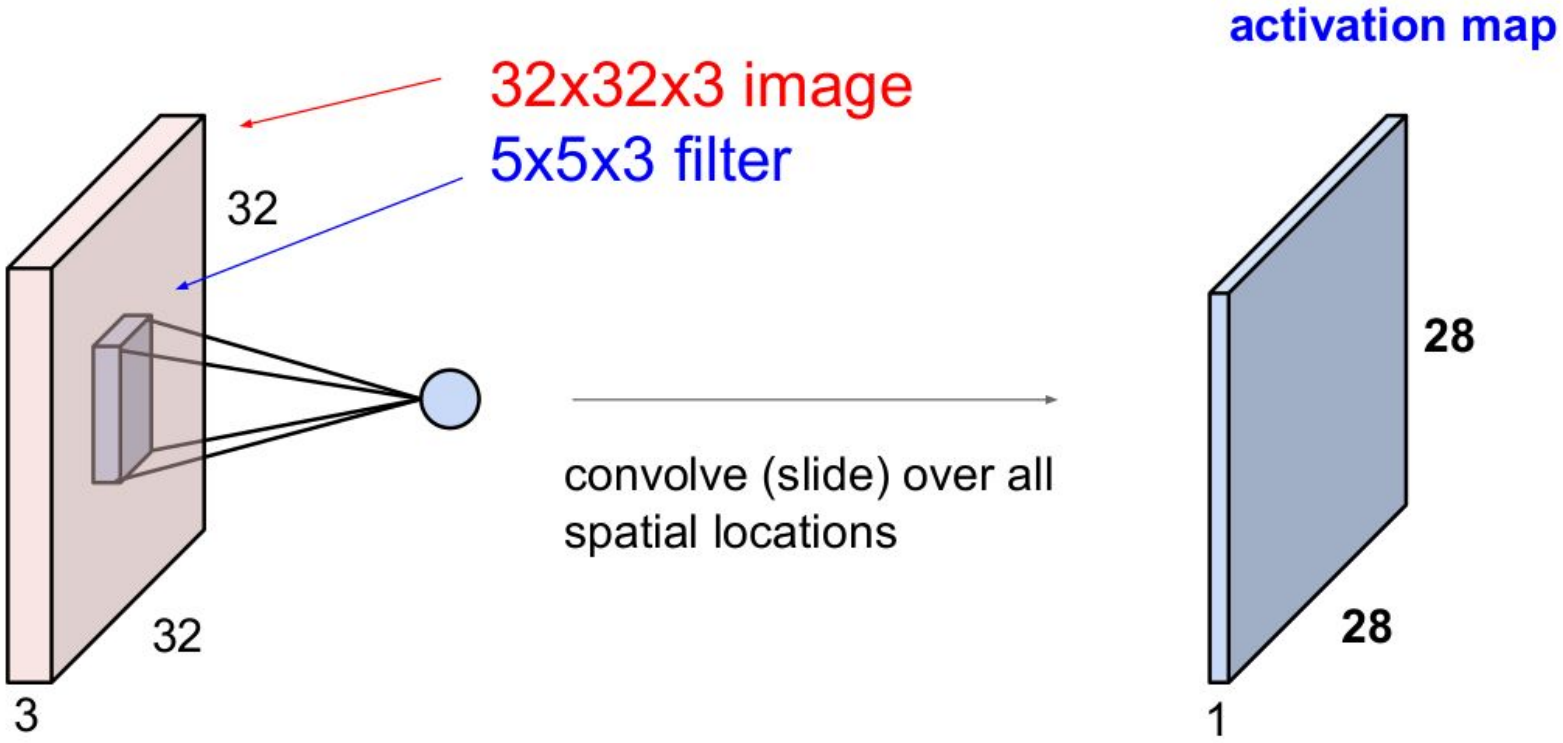


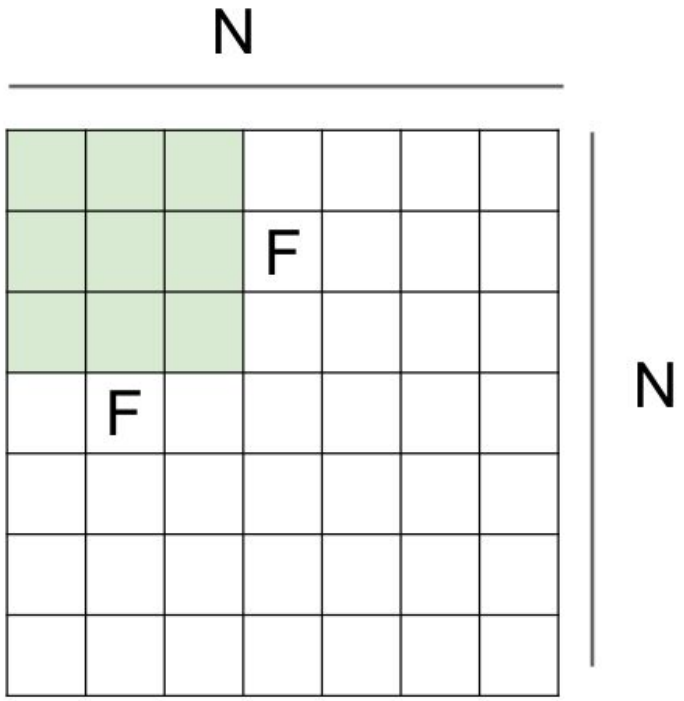
We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



A closer look at spatial dimensions:

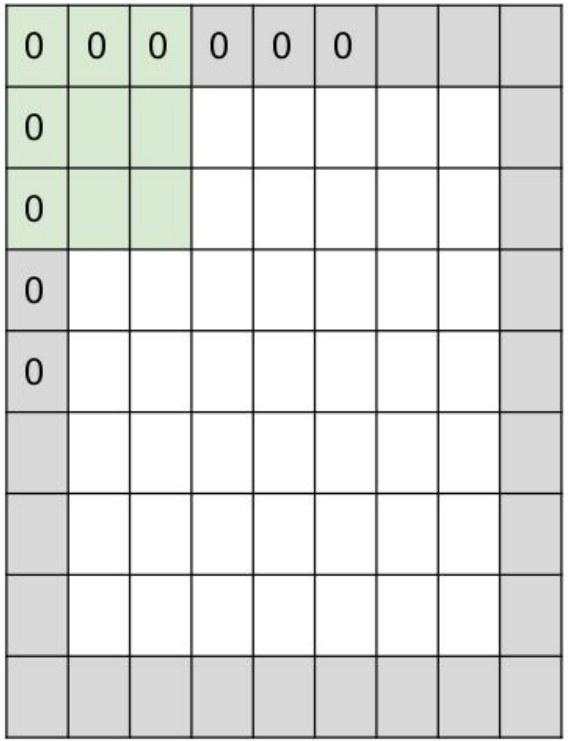




Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

In practice: Common to zero pad the border

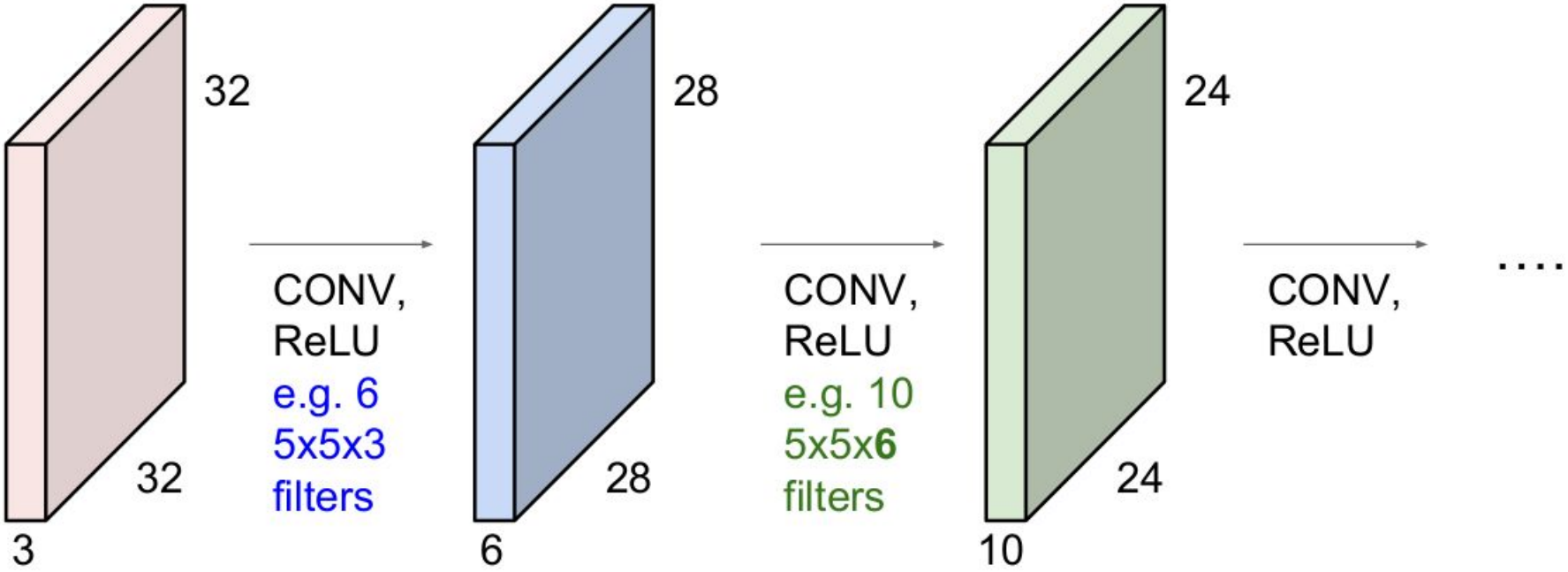


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

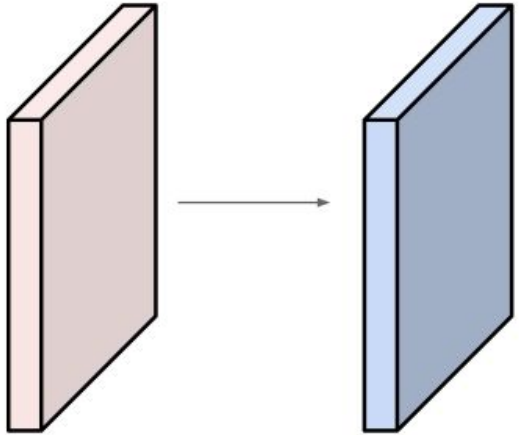
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



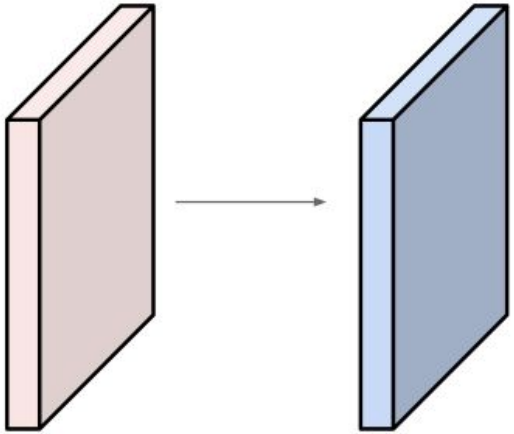
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Summary. To summarize, the Conv Layer:

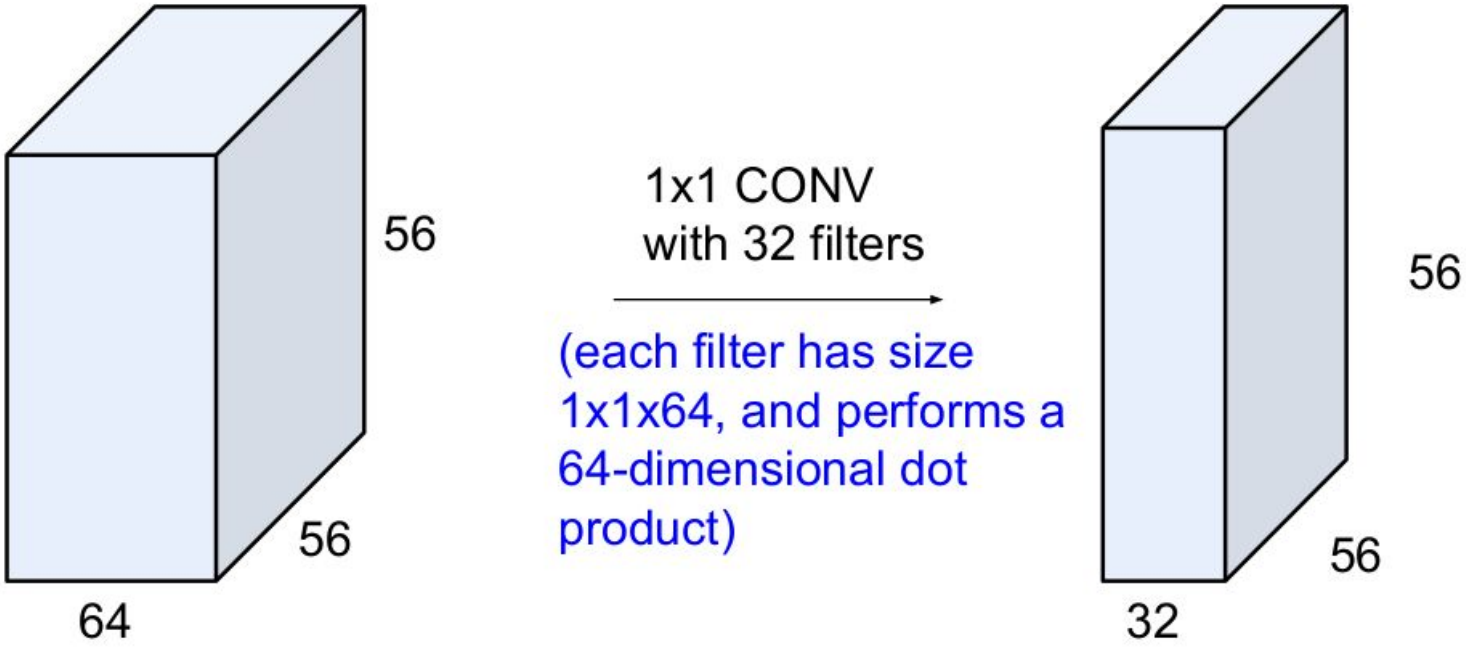
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

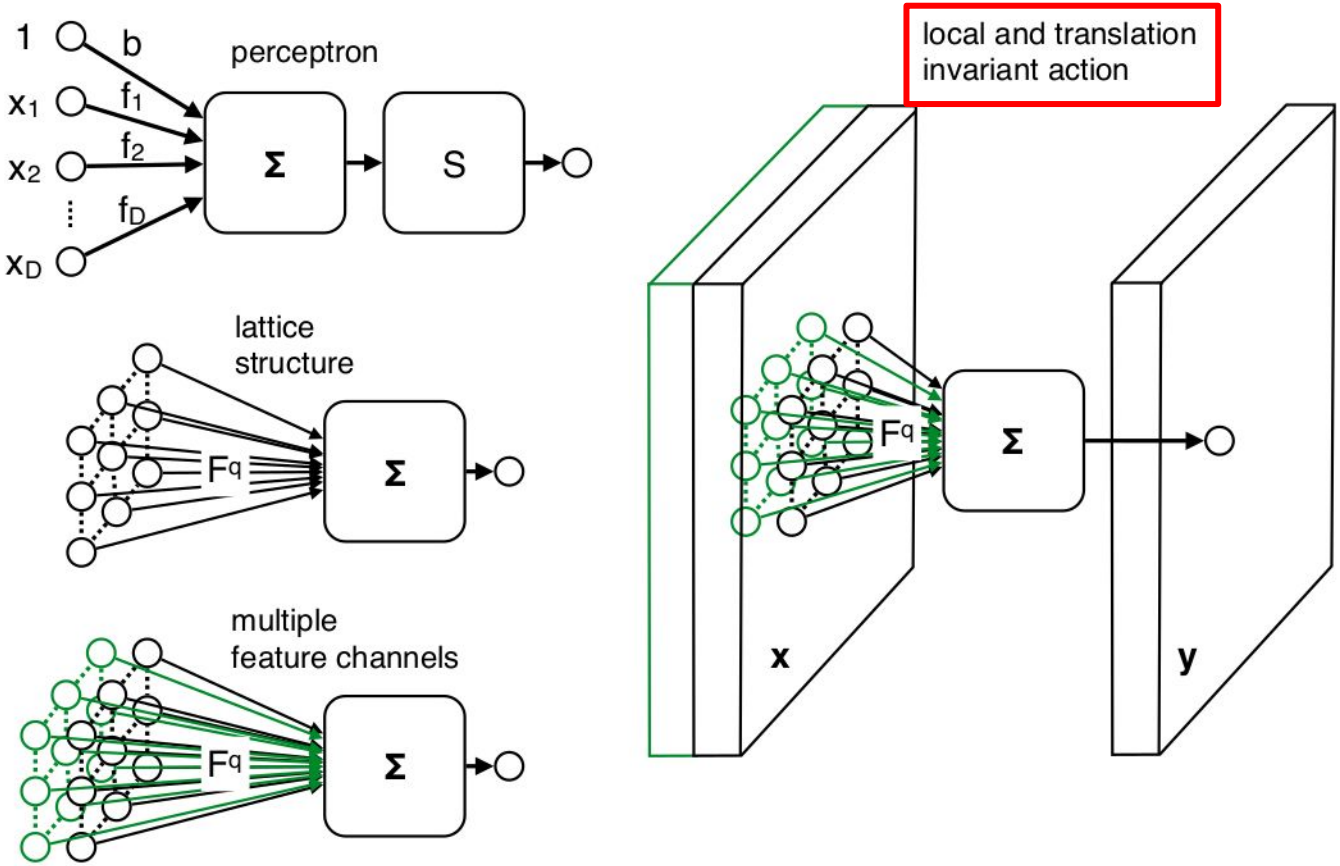
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

(btw, 1x1 convolution layers make perfect sense)



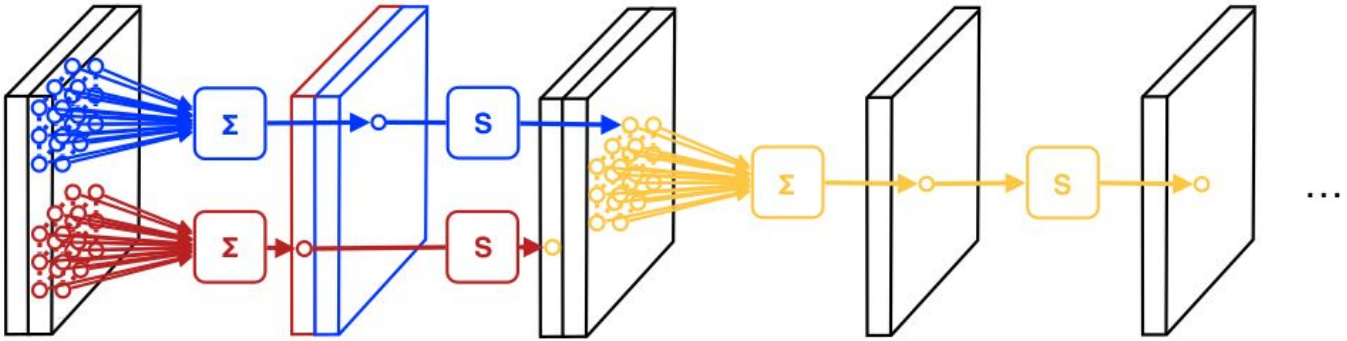
Linear convolution

As a neural network



Multiple layers

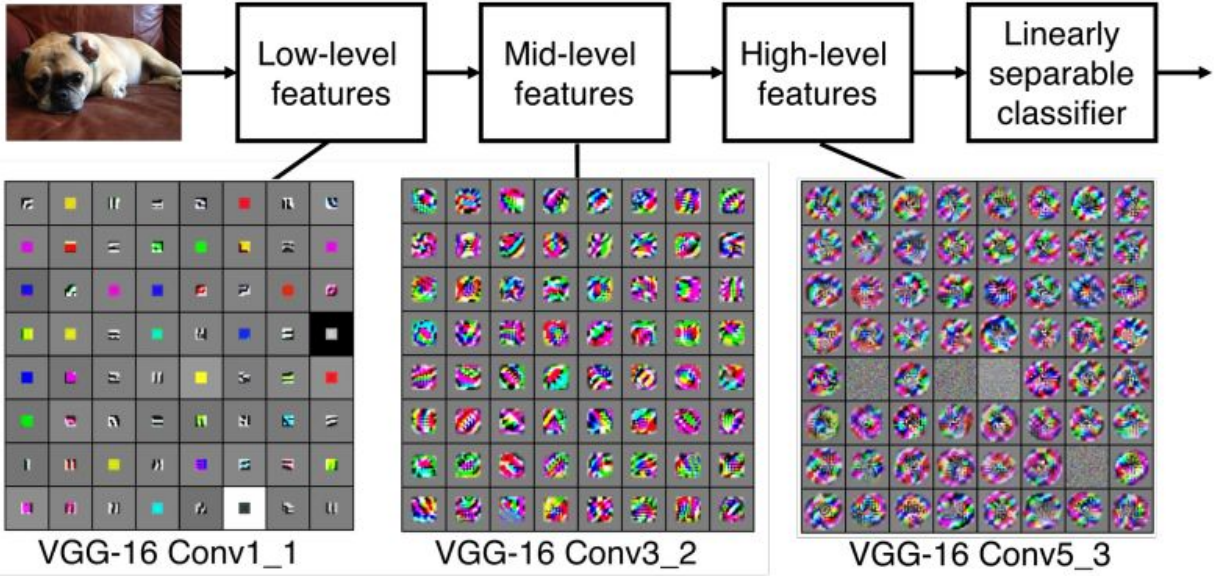
Convolution, activation, convolution, activation, …

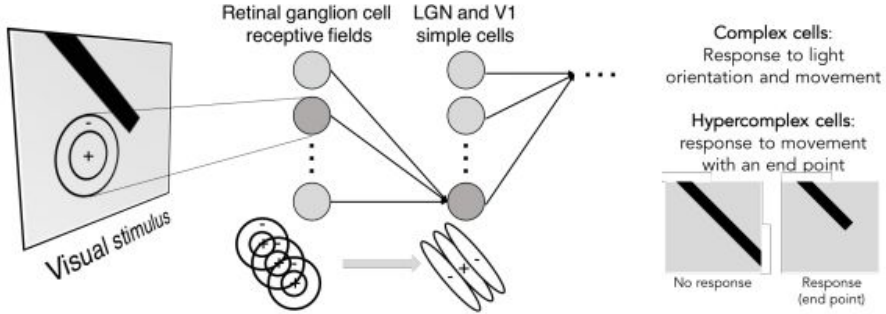


A deep convolutional neural networks chains several filtering & non-linear activation function sequences.

The non-linear activation functions are essential. Why?

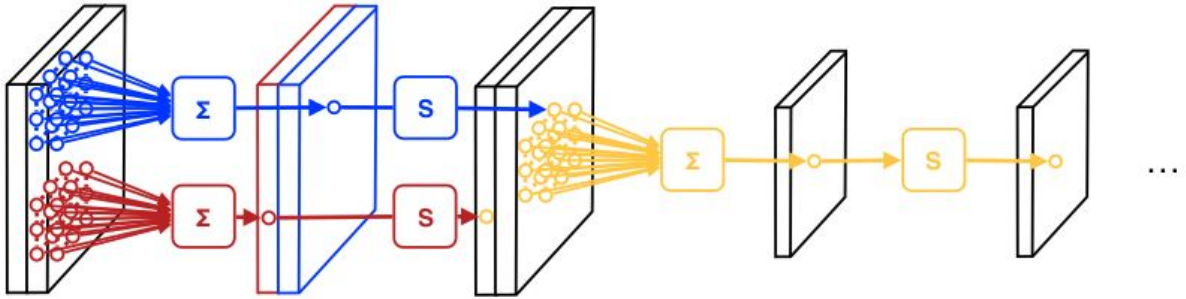
Preview


Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3


Retinal ganglion cell receptive fields
LGN and V1 simple cells
Visual stimulus
Complex cells:
Response to light orientation and movement
Hypercomplex cells:
response to movement with an end point
No response
Response (end point)

Multiple layers

Downsampling

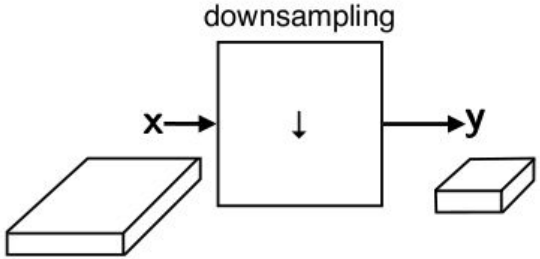


Downsampling by a factor S amounts to keeping only one every S pixels, discarding the others.

Filter banks ofter incorporate, or are followed by, 2× output downsampling.

Downsampling is often paired with an increase in the number of feature channels.

Overall, as depth increases the *volume* of the tensors decreases, but slowly.

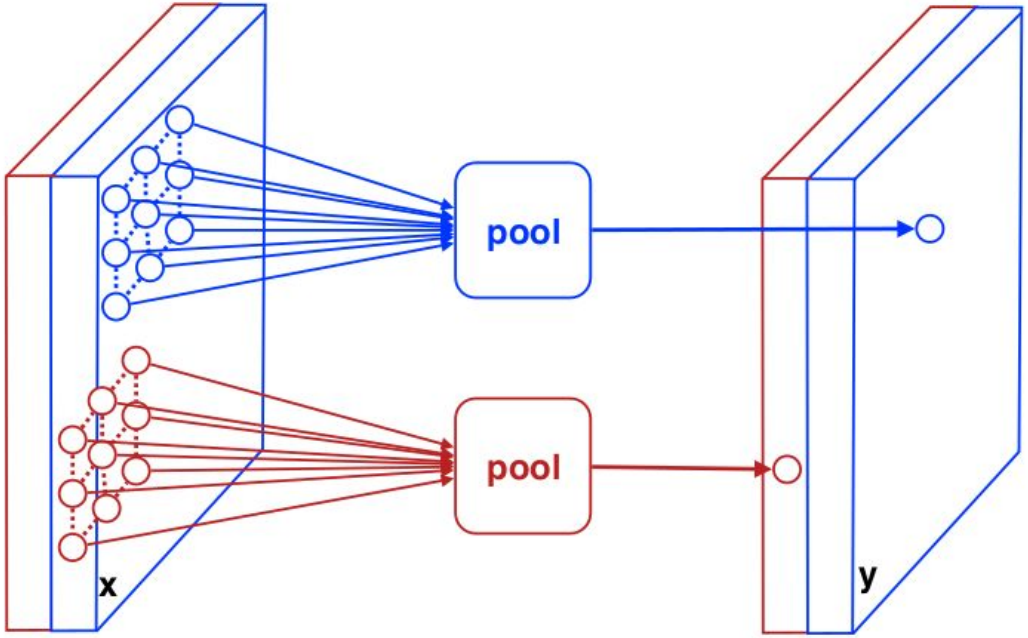


Spatial pooling

Local translation invariance

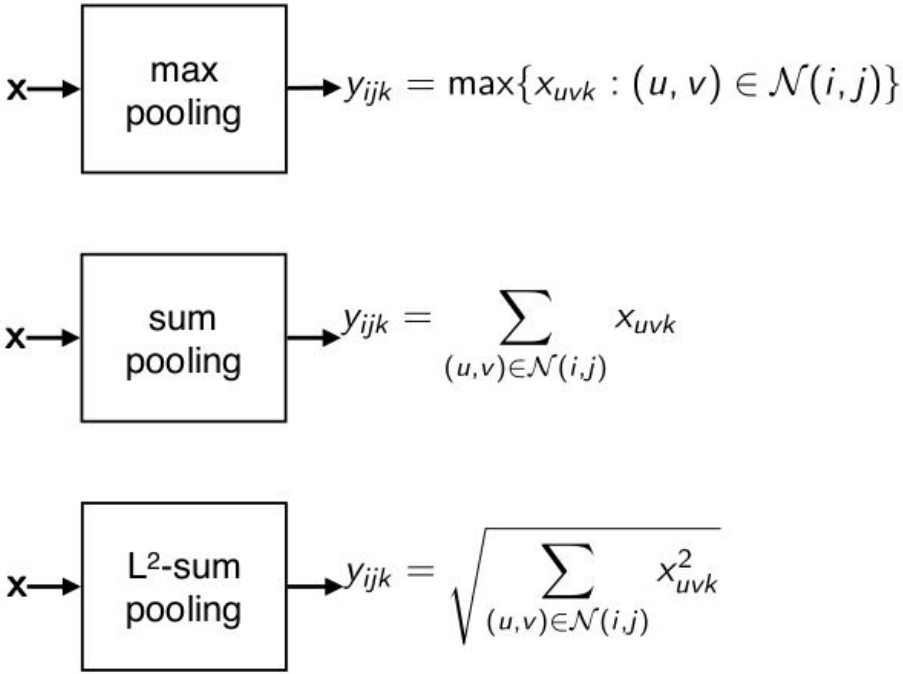
Pooling computes the average or max of a feature response in a small image spatial neighbourhood.

It is applied to each feature channel independently and in parallel.



Spatial pooling

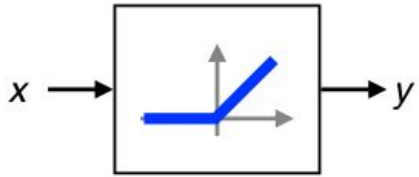
Variants

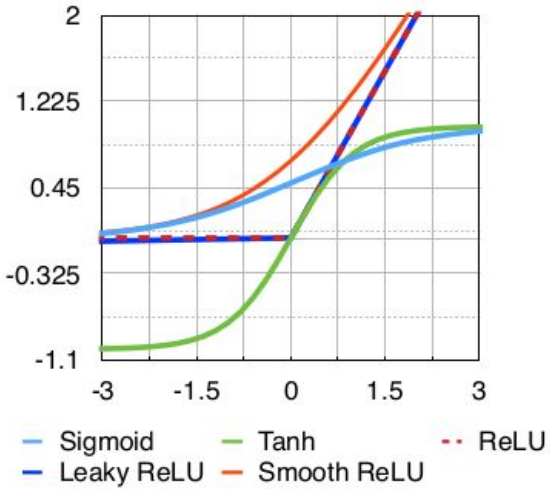


By far, the most common variant is **max pooling**.

Activation functions

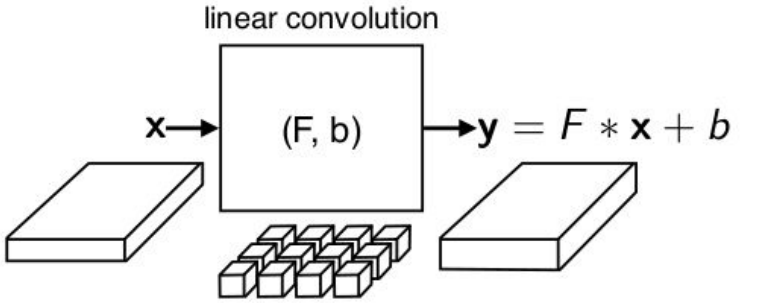
Non-linear functions applied to each element of a tensor

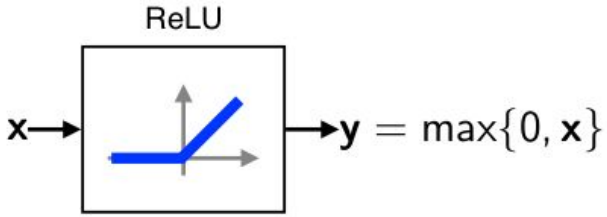


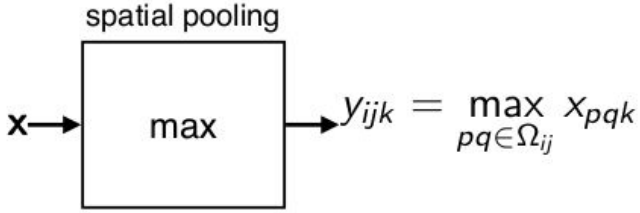


sigmoid	$y = \frac{1}{1+e^{-x}}$
hyperb. tan	$y = \tanh(x)$
ReLU	$y = \max\{0, x\}$
Soft ReLU	$y = \log(1+e^{x})$
Leaky ReLU	$y = \epsilon x + (1-\epsilon)\max\{0, x\}$

CNN layers summary







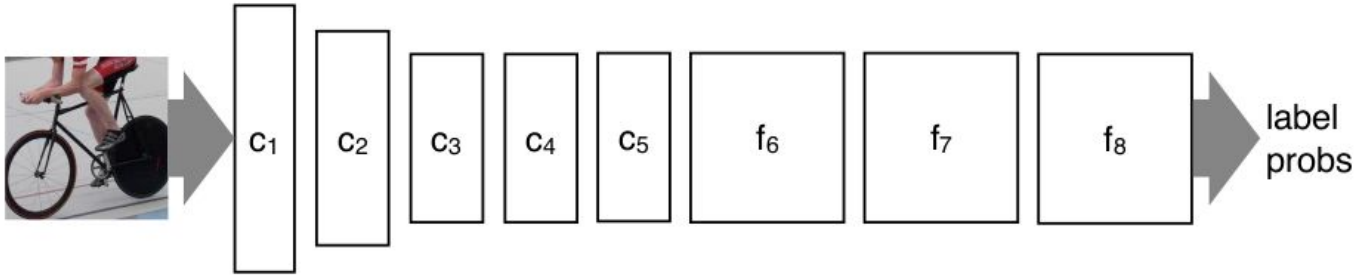
Many common CNN architectures can be obtained by just three layer types:

- Linear convolution
- ReLU
- Max pooling

Other common layers include:

- Cross-feature normalisation such as LRN.
- Within-feature normalisation such as contrast and batch normalisation.
- Parametric and pyramid spatial pooling (see next lecture).
- Sum and stacking (for branching network topologies).

A typical CNN design



From left to right

- decreasing spatial resolution
- increasing feature dimensionality

"Fully-connected" layers

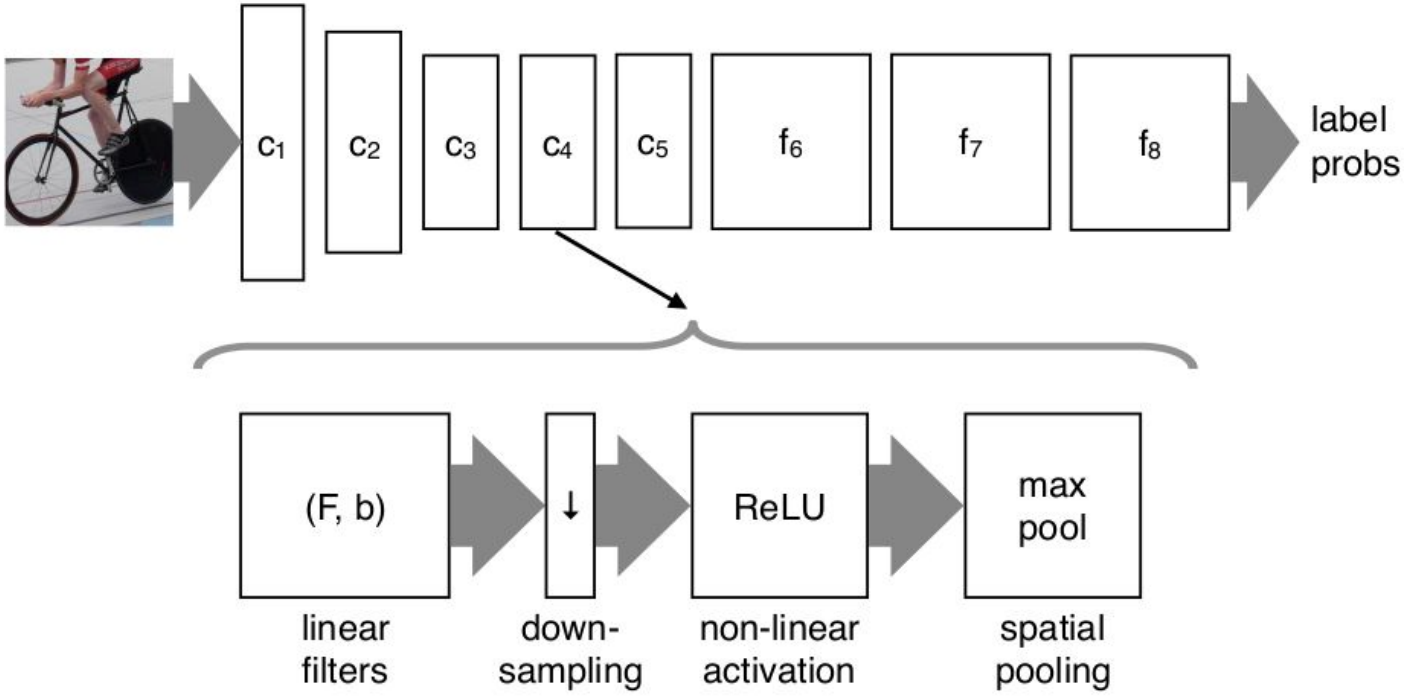
- Here the spatial resolution is 1×1 (tensors are mere vectors), but there can be thousands of channels

Label probabilities and softmax

- The last feature vector has one entry per class label which is converted to a vector of label probabilities by the softmax operator:

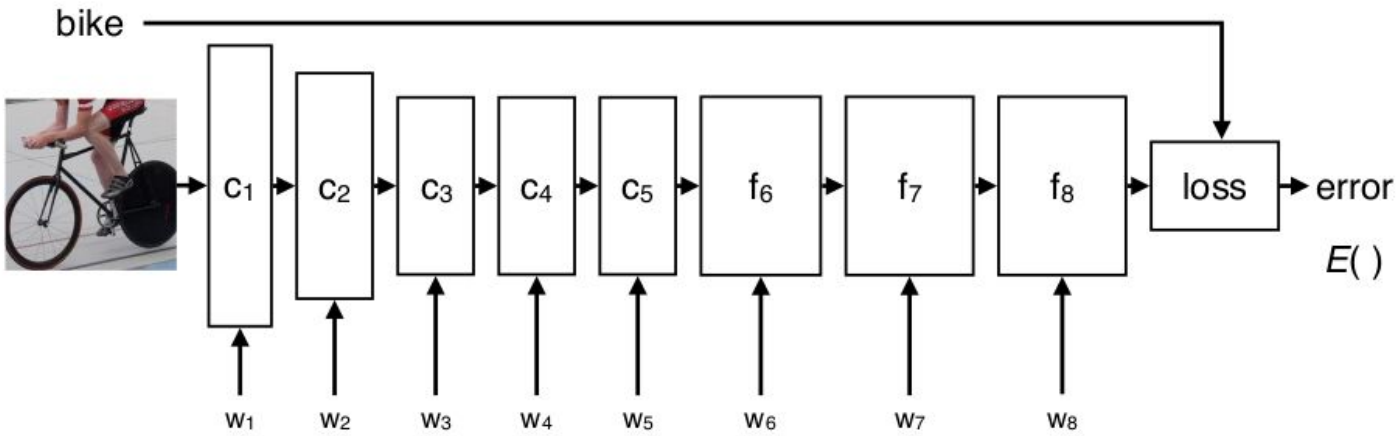
$$[\mathrm{sftmax}(\mathbf{x})]_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Convolutional layers



Each block c_1, c_2, …, f_8 incorporates convolution, ReLU and, possibly, downsampling and max pooling.

Learning a CNN



$$\text{argmin } E(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_8)$$

Stochastic gradient descent
(with momentum, dropout, …)

Learning CNNs classifiers

Challenge

- many parameters, prone to overfitting

Key ingredients

- very large annotated data
- heavy regularisation (dropout)
- stochastic gradient descent
- GPU(s)

Training time

- ~ 90 epochs
- days—weeks of training
- requires processing ~150 images/sec

IMAGENET

- 1K classes
- ~ 1K training images per class
- ~ 1M training images

Stochastic gradient descent

The objective function is an average over many data points:

$$E(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} E_i(\mathbf{w})$$

Key idea: approximate the gradient sampling a point at a time:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla E_i(\mathbf{w}_t), \qquad i \sim U(\{1, 2, \ldots, N\})$$

uniform distribution

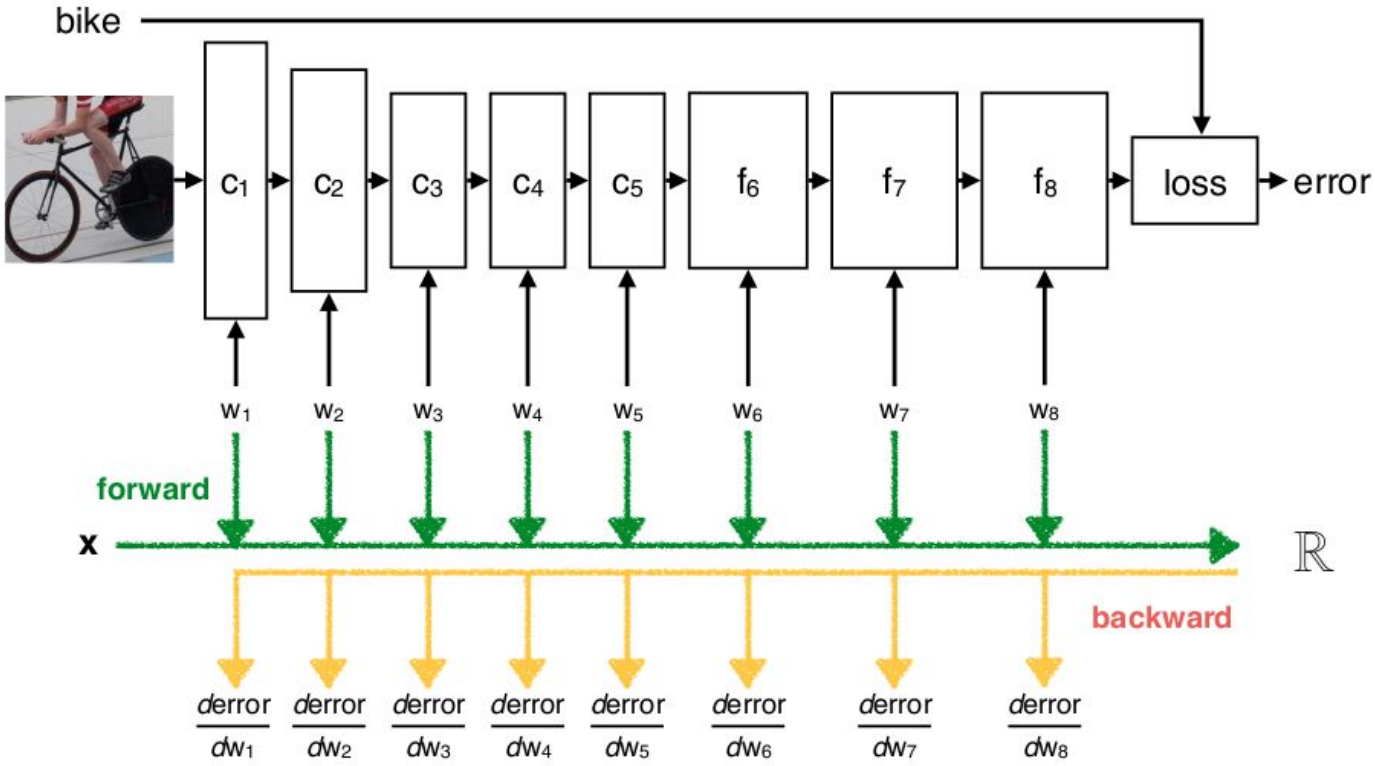
Details:

- **Epochs**: all points are visited sequentially, but in random order
- **Validation**: evaluate E($\mathbf{w}_t$) on an held-out validation set to diagnose objective decrease
- **Learning rate** η_t: is decreased tenfold once the objective E($\mathbf{w}_t$) stops decreasing.
- **Momentum**: the gradient estimate is smoothed by using a moving average:

$$\mathbf{m}_{t+1} = 0.9\,\mathbf{m}_t + \eta_t \nabla E_i(\mathbf{w}_t), \qquad \mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{m}_{t+1}$$

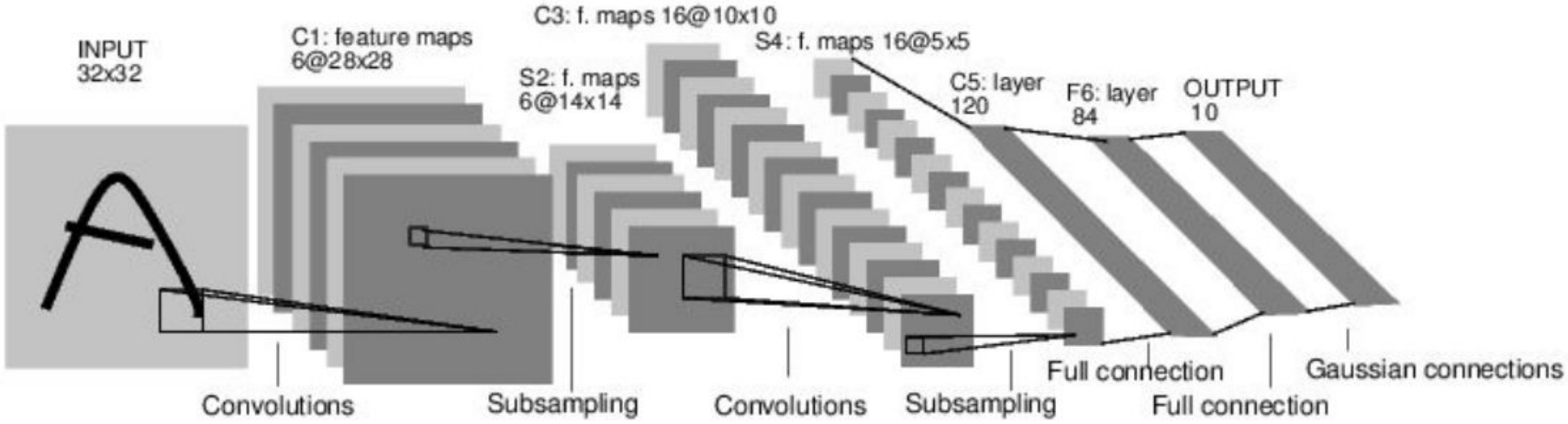
Backpropagation

Computing derivatives using the chain rule



Case Study: LeNet-5

[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]