Machine Learning for Image Processing

Chapter IV – Image classification and CNNs

Charles Deledalle

July 9, 2019

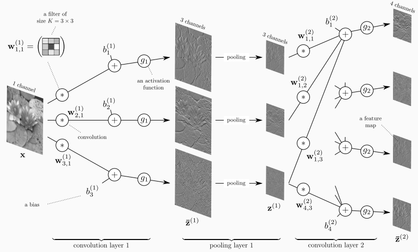
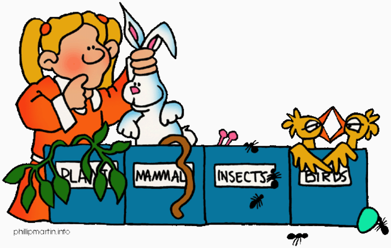


Image classification

Reminder of classification

Classification: predict class d from observation $\boldsymbol{x}$.



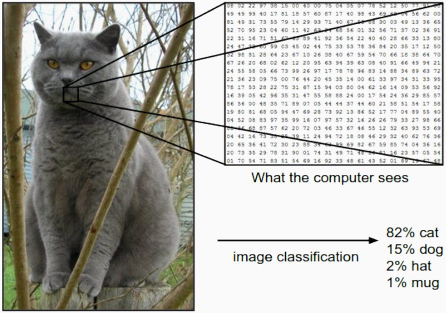
(Source: Philip Martin)

Image classification:

- Observation $\boldsymbol{x}$ is an image (usually a tensor with 3 RGB channels),
- Class d is anything: $\{\texttt{'boat'}, \texttt{'leopard'}, \texttt{'mushroom'}, \ldots\}$,
- We assume one class (object) per image
  ($\neq$ multi-label classification or multi-object detection).

Reminder of image representation

- Represented as 3d arrays (tensors),
- Integers between $[0, 255]$,
- Example: $300 \times 100 \times 3$
  (3 for 3 color channels RGB)



```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

What the computer sees

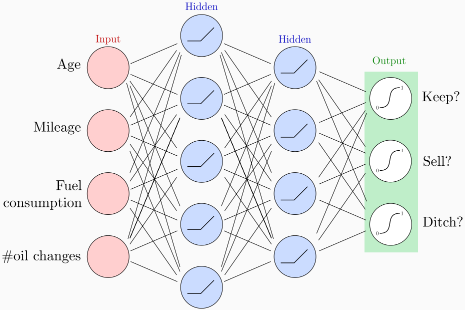
image classification

82% cat
15% dog
2% hat
1% mug

Difficulty: semantic gap between representation and its content.

Reminder of neural networks for classification

- **Typical architecture:**



- Hidden layer:

$$\text{ReLU}(a) = \max(a, 0)$$

- Output layer:

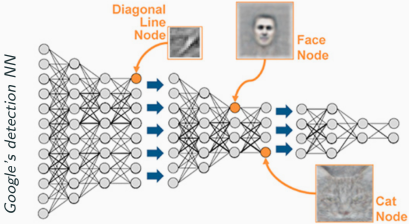
$$\text{softmax}(\boldsymbol{a})_k = \frac{\exp(a_k)}{\sum_{l=1}^{K} \exp(a_l)}$$

$\rightarrow$ 1-of-K code with confidence levels $(0, 1)$

- **What about image classification?**
  - What should be the input?
  - How many layers? How many units per layer?
  - How to organize the connections between each layer?

Reminder of Deep Learning concepts

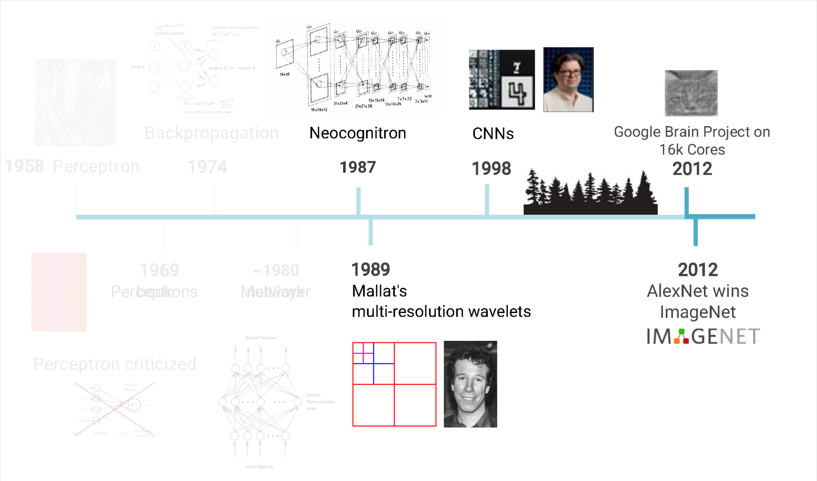
- Input should be the pixel values/tensors (no hand-crafted features).
- Design your network with many hidden layers.
- Learn a hierarchy of features from low- to high-level concepts.



- Use: SGD, dropout, batch-normalization, data-augmentation, . . .
- Regularize / simplify / adapt your network to fit your prior knowledge.

Timeline

Timeline of CNNs



1958 Perceptron

Backpropagation 1974

Neocognitron 1987

CNNs 1998

Google Brain Project on 16k Cores 2012

1969 Perceptrons book

~1980 Multilayer Network

Perceptron criticized

1989 Mallat's multi-resolution wavelets

2012 AlexNet wins ImageNet

(Source: Lucas Masuch & Vincent Lepetit)

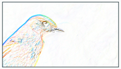
Convolutions and filter banks

Regularization of NNs for images

- **Reminder:** when designing a network, one should inject prior knowledge about intrinsic regularities of the data.

- One particularity of images is that location is not of main importance,
- They are defined on a grid whose origin was chosen arbitrarily,
- A same feature/object can be localized anywhere in the image,
- It should be detected regardless of its position.

Regularization of NNs for images

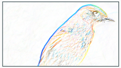
- Let $\boldsymbol{x} =$ and $\boldsymbol{h} =$  ,

  input image

  1st hidden response

- If we shift the input $\boldsymbol{x}$ by (δ_i, δ_j), the output $\boldsymbol{h}$ should be also shifted

  $\boldsymbol{x}' =$

  $\boldsymbol{h}' =$ 

  $$x'_{i+\delta_i, j+\delta_j} = x_{i,j} \quad \Rightarrow \quad h'_{i+\delta_i, j+\delta_j} = h_{i,j}$$

  $$\text{where} \quad \begin{cases} \boldsymbol{h} & = g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \\ \boldsymbol{h}' & = g(\boldsymbol{W}\boldsymbol{x}' + \boldsymbol{b}) \end{cases}$$

  where (i, j) is any pixel location and g is the activation function.

- The output $\boldsymbol{h}$ is called a `feature map`,

- In signal processing, this property is called translation invariance, though it should be called translation equivariance.

Regularization of NNs for images

- In practice g is chosen such that it is either
    - element-wise $g(\boldsymbol{a})_i = g(a_i)$ (`tanh`, `ReLU`, ...), or
    - equivariant through permutations (`maxout`, `softmax`)

$$g(\boldsymbol{P}\boldsymbol{a}) = \boldsymbol{P}g(\boldsymbol{a}) \quad \text{where} \quad \boldsymbol{P} \text{ is a permutation matrix.}$$

- Asking for translation invariance of $\boldsymbol{h}$ or $\boldsymbol{a}$ is then equivalent

$$x'_{i+\delta_i,j+\delta_j} = x_{i,j} \quad \Rightarrow \quad a'_{i+\delta_i,j+\delta_j} = a_{i,j}$$
$$\text{where} \quad \begin{cases} \boldsymbol{a} & = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \\ \boldsymbol{a}' & = \boldsymbol{W}\boldsymbol{x}' + \boldsymbol{b} \end{cases}$$

- As a result, the bias should be constant: $\boldsymbol{b} = b\mathbf{1}$, $b \in \mathbb{R}$,
- And $\psi : \boldsymbol{x} \to \boldsymbol{W}\boldsymbol{x}$ should be linear translation invariant (LTI).

Linear translation invariant filters

Linear translation invariant filters

Let ψ be satisfying

1. **Linearity** $\psi(a\boldsymbol{x} + b\boldsymbol{y}) = a\psi(\boldsymbol{x}) + b\psi(\boldsymbol{y})$
2. **Translation-invariance** $\psi(\boldsymbol{x}^\delta) = \psi(\boldsymbol{x})^\delta$ where $x^\delta_{i,j} = x_{i+\delta_i, j+\delta_j}$

Then, there exist coefficients $\kappa_{k,l}$ such that

$$\psi(\boldsymbol{x})_{i,j} = \sum_{(k,l)\in\mathbb{Z}^2} \kappa_{k,l} x_{i+k,j+l}$$

The reciprocal holds true

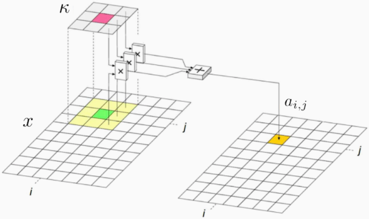
Convolutions and filter banks

Linear translation invariant filters

- LTI filters implement the (discrete) cross-correlation

$$\boldsymbol{a} = \boldsymbol{\kappa} \star \boldsymbol{x} \quad \Leftrightarrow \quad a_{i,j} = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} \kappa_{k,l} x_{i+k,j+l}$$

- $\boldsymbol{\kappa}$ is often called filter.



Here κ has a $S = 3 \times 3$ support

$$\Rightarrow \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} \equiv \sum_{k=-1}^{+1} \sum_{l=-1}^{+1}$$

S called the filter size.

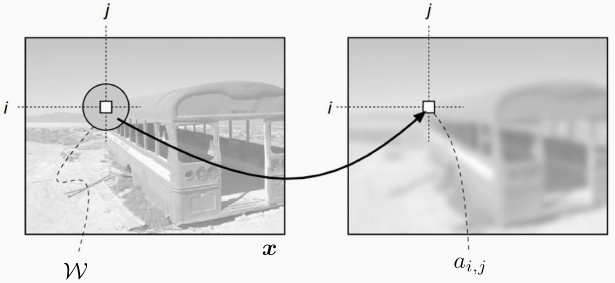
Direct computation requires $O(NS)$.

$\Rightarrow S \ll N$ (small filters)

Local neighboring filters

Local neighboring filters

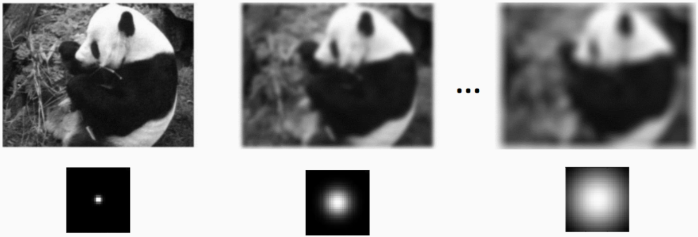
- Using a small finite support leads to local neighboring filters,
- Output depends only on a few values of x located in a small window $\mathcal{W}$,
- Each neighborhood is processed in the exact same way (invariance),



Convolutions and filter banks

Example: simulated blur

(Truncated) Gaussian filter: $\kappa_{i,j} = \frac{1}{Z} \exp\left(-\frac{i^2 + j^2}{2\tau^2}\right)$



Small τ Medium τ Large τ

- $i^2 + j^2$: square distance to the central pixel,
- τ: controls the influence of neighbor pixels, *i.e.*, the strength of the blur.
- Z: normalization such that $\sum \kappa_{i,j} = 1$.

Cross-correlation vs Convolution product

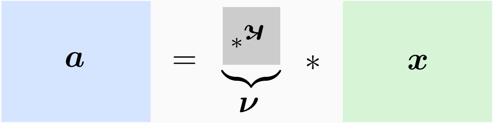
- If κ is complex then the cross-correlation becomes

$$\boldsymbol{a} = \boldsymbol{\kappa} \star \boldsymbol{x} \quad \Leftrightarrow \quad a_{i,j} = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} \kappa^*_{k,l} x_{i+k,j+l}.$$

- Complex conjugate: $(a + ib)^* = a - ib$.
- $\boldsymbol{a} = \boldsymbol{\kappa} \star \boldsymbol{x}$ can be re-written as the (discrete) convolution product

$$\boldsymbol{a} = \boldsymbol{\nu} * \boldsymbol{x} \quad \Leftrightarrow \quad a_{i,j} = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} \nu_{-k,-l} x_{i+k,j+l} \quad \text{with} \quad \nu_{k,l} = \kappa^*_{-k,-l}$$

- $\boldsymbol{\nu}$ called convolution kernel = flip in both directions of the filter $\boldsymbol{\kappa}^*$.



Why convolution instead of cross-correlation?

Properties of the convolution product

- **Linear** $\boldsymbol{f} * (\alpha \boldsymbol{g} + \beta \boldsymbol{h}) = \alpha(\boldsymbol{f} * \boldsymbol{g}) + \beta(\boldsymbol{f} * \boldsymbol{h})$
- **Commutative** $\boldsymbol{f} * \boldsymbol{g} = \boldsymbol{g} * \boldsymbol{f}$
- **Associative** $\boldsymbol{f} * (\boldsymbol{g} * \boldsymbol{h}) = (\boldsymbol{f} * \boldsymbol{g}) * \boldsymbol{h}$
- **Separable** $\boldsymbol{h} = \boldsymbol{h}_1 * \boldsymbol{h}_2 * \ldots * \boldsymbol{h}_p$

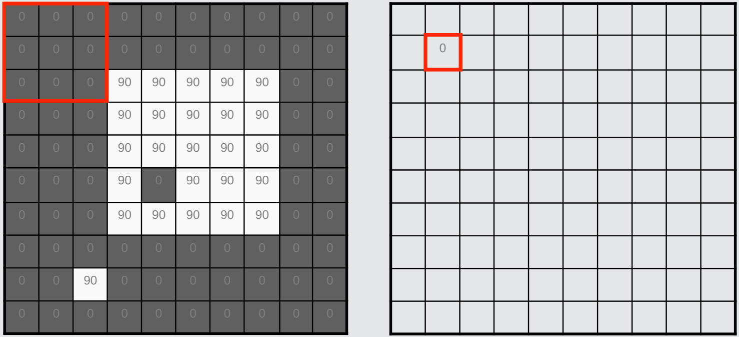
$$\Rightarrow \quad \boldsymbol{f} * \boldsymbol{h} = (((\boldsymbol{f} * \boldsymbol{h}_1) * \boldsymbol{h}_2) \ldots * \boldsymbol{h}_p)$$

For cross-correlation, only true if the signal is Hermitian, i.e., if $f_{k,l} = f^*_{-k,-l}$.

Many machine learning libraries implement cross-correlation but often call it convolution!

Convolution example

3×3 boxcar convolution



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

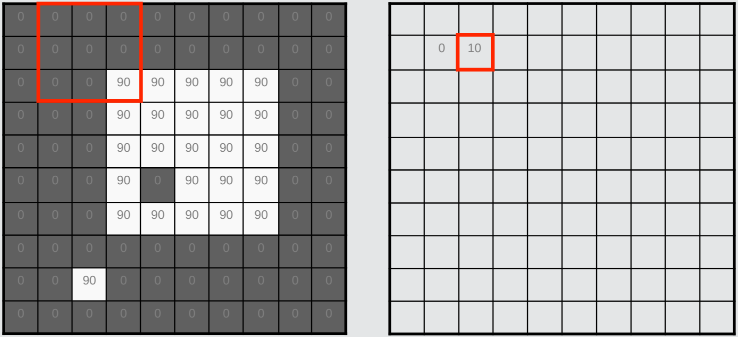
	0								

(Source: Steven Seitz)

Boxcar filter: $\kappa_{i,j} = \begin{cases} \frac{1}{9} & \text{if} \quad \max(|i|, |j|) \leqslant 1 \\ 0 & \text{otherwise} \end{cases}$

Convolution example

3×3 boxcar convolution



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

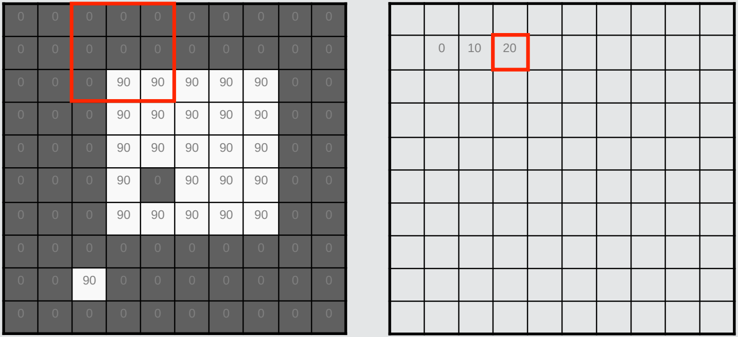
	0	10							

(Source: Steven Seitz)

Boxcar filter: $\kappa_{i,j} = \begin{cases} \frac{1}{9} & \text{if} \quad \max(|i|,|j|) \leqslant 1 \\ 0 & \text{otherwise} \end{cases}$

Convolution example

3×3 boxcar convolution



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

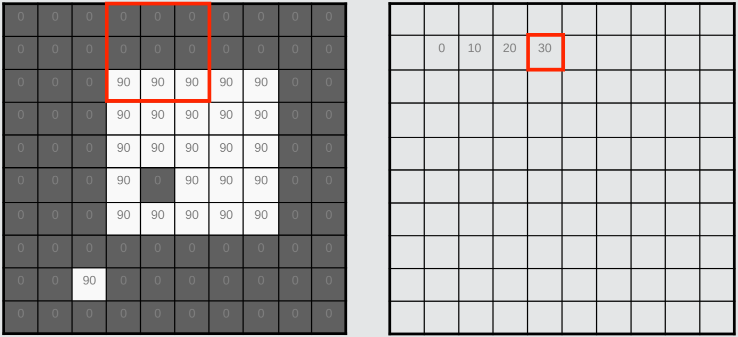
	0	10	20						

(Source: Steven Seitz)

Boxcar filter: $\kappa_{i,j} = \begin{cases} \frac{1}{9} & \text{if} \quad \max(|i|, |j|) \leqslant 1 \\ 0 & \text{otherwise} \end{cases}$

Convolution example

3×3 boxcar convolution



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

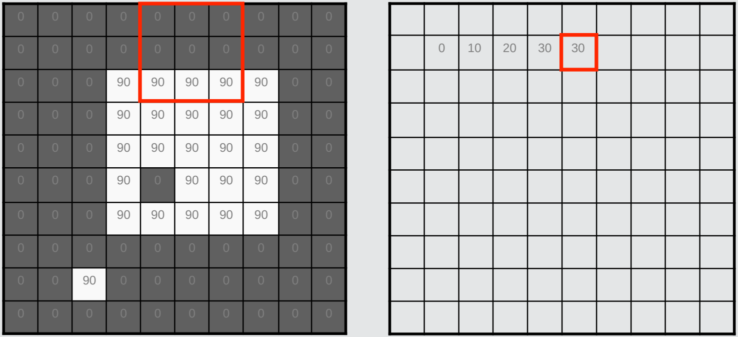
	0	10	20	30					

(Source: Steven Seitz)

Boxcar filter: $$\kappa_{i,j} = \begin{cases} \frac{1}{9} & \text{if} \quad \max(|i|,|j|) \leqslant 1 \\ 0 & \text{otherwise} \end{cases}$$

Convolution example

3×3 boxcar convolution



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

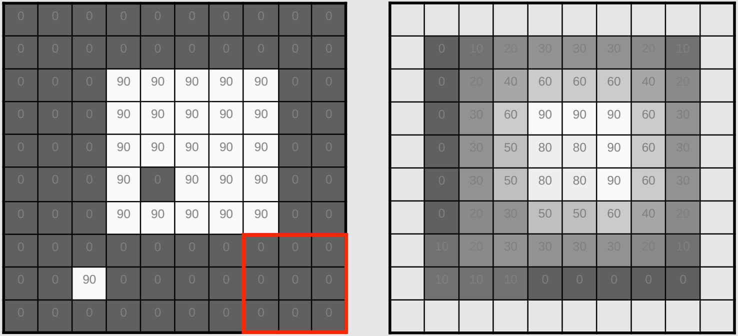
	0	10	20	30	30				

(Source: Steven Seitz)

Boxcar filter: $\kappa_{i,j} = \begin{cases} \frac{1}{9} & \text{if} \quad \max(|i|, |j|) \leqslant 1 \\ 0 & \text{otherwise} \end{cases}$

Convolution example

3×3 boxcar convolution



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

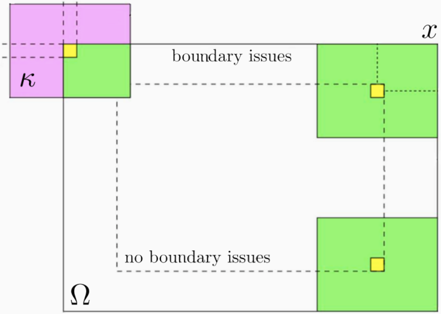
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

(Source: Steven Seitz)

Boxcar filter: $\kappa_{i,j} = \begin{cases} \frac{1}{9} & \text{if} \quad \max(|i|,|j|) \leqslant 1 \\ 0 & \text{otherwise} \end{cases}$

Boundary conditions

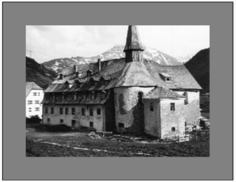
How to deal when the kernel window overlaps outside the image domain?



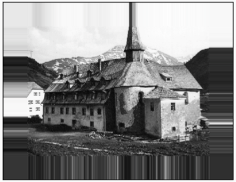
i.e., how to evaluate $a_{i,j} = \sum_{k,l} \kappa_{k,l} x_{i+k,j+l}$ when $(i+k, j+l) \notin \Omega$?

Boundary conditions

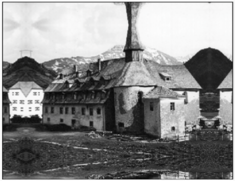
Standard techniques virtually enlarge x with either:



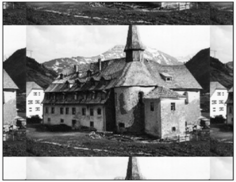
zero-padding



extension



mirror

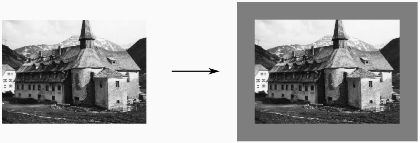


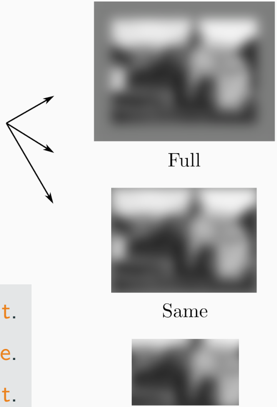
periodical

Most machine learning libraries use zero-padding.

Boundary conditions

Then, apply the formula: $a_{i,j} = \sum_{(k,l)\in\mathcal{W}} \kappa_{(k,l)} x_{i+k,j+l}$





for all (i,j) such that:

- **Full:** $\exists(i+k, j+l) \in \Omega$ → larger output.
- **Same:** $(i,j) \in \Omega$ → same size.
- **Valid:** $\forall(i+k, j+l) \in \Omega$ → smaller output.

Most machine learning libraries implement the three of them.

Boundary conditions – Padding parameter

Alternatively, machine learning toolboxes introduce a padding parameter p:

- First, add p rows and columns of 0 on each boundary (zero-padding),
- Next, apply convolution with valid boundary conditions.

For a $n \times n$ filter (n odd):

$$p = 0 \equiv \text{valid}$$
$$p = \tfrac{n-1}{2} \equiv \text{same}$$
$$p = n - 1 \equiv \text{full}$$

$p = 0$ **is the most common option for CNNs (*i.e.*, valid boundary condition)**

Convolutions and filter banks

What is the link between κ and the weights W?

One dimensional case

- Small-support + Zero-padding + Same → $\boldsymbol{W}$ is a band Toeplitz matrix:

$$\boldsymbol{W} = \begin{pmatrix} \kappa_0 & \kappa_1 & & & \dots & 0 \\ \kappa_{-1} & \kappa_0 & \kappa_1 & & & \vdots \\ & \kappa_{-1} & \kappa_0 & \kappa_1 & & \\ & & \ddots & \ddots & \ddots & \\ \vdots & & & \ddots & \ddots & \kappa_1 \\ 0 & \dots & & & \kappa_{-1} & \kappa_0 \end{pmatrix}$$

- Periodical + Same → $\boldsymbol{W}$ is a circulant matrix:
  (*Connection with the Fourier transform, see, convolution theorem*).

Quiz: how many degrees of freedom for a general $N \times N$ matrix?
a Toeplitz matrix? a convolution?

What is the link between κ and the weights W?

Two-dimensional case

- Encode images as vectors: row1, row2, ...
- Small-support + Z.-P. + Same → $\boldsymbol{W}$ is doubly block band Toeplitz:

$$\boldsymbol{W} = \begin{pmatrix} \boldsymbol{K}_0 & \boldsymbol{K}_1 & & & & \boldsymbol{0} \\ \boldsymbol{K}_{-1} & \boldsymbol{K}_0 & \boldsymbol{K}_1 & & & \\ & \boldsymbol{K}_{-1} & \boldsymbol{K}_0 & \boldsymbol{K}_1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \boldsymbol{K}_1 \\ \boldsymbol{0} & & & & \boldsymbol{K}_{-1} & \boldsymbol{K}_0 \end{pmatrix}$$

- Each block $\boldsymbol{K}_i$ is a band Toeplitz matrix.

PS: don't use matrix operations, apply cross-correlation: $\underbrace{\boldsymbol{W}\boldsymbol{x}}_{O(N^2)} = \underbrace{\kappa \star \boldsymbol{x}}_{O(NS)}$

Two main types of filters

- **Averaging filters:** $\sum_{i,j} \kappa_{i,j} = 1$ → blurs the image
    - Low-pass filter: remove high frequencies of the signal,
    - Applications: remove imperfection / denoise / zoom out,
    - Example: boxcar, Gaussian filter.
- **Derivative filters:** $\sum_{i,j} \kappa_{i,j} = 0$ → measures local correlations

$$a_{i,j} = \sum_{(k,l)\in\mathcal{W}} \kappa_{k,l} x_{i+k,j+l} = \langle \boldsymbol{\kappa},\ \boldsymbol{x}|_{\mathcal{W}_{i,j}} \rangle = \mathsf{cov}(\boldsymbol{\kappa},\ \boldsymbol{x}|_{\mathcal{W}_{i,j}})$$

- High- or band-pass filter: select frequencies of the signal,
- Applications: detect specific features / structures / patterns.

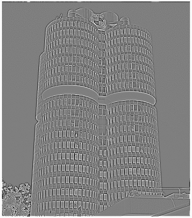
Convolutions and filter banks

Example: Laplacian filter

Extract 2nd order slopes (approximate second derivative):

$$\boldsymbol{k}_{\triangle} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

(a) $\boldsymbol{x}$



(b) $\boldsymbol{k}_{\triangle} \star \boldsymbol{x}$

Quiz: averaging or derivative? what does it detect?

Convolutions and filter banks

Filter banks

- Of course, what matters is not to detect one single feature, but many.
- Performs several convolutions each of them specialized in one particular type of feature

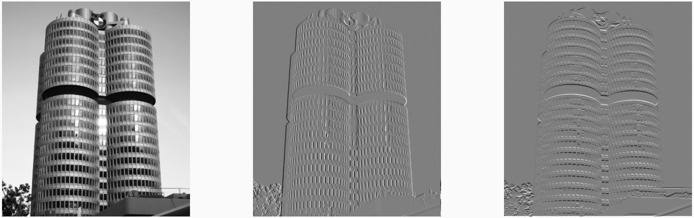
$$\boldsymbol{x} \longrightarrow \begin{cases} \kappa^1 * \boldsymbol{x} \\ \kappa^2 * \boldsymbol{x} \\ \vdots \\ \kappa^M * \boldsymbol{x} \end{cases}$$

- Make predictions by examining all responses,
- Combine the feature maps in a non-linear way.

Example 1: Prewitt filters (1970)

Extract vertical and horizontal features:

$$\boldsymbol{k}_{\text{vert}} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{k}_{\text{hor}} = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$



(a) $\boldsymbol{x}$

(b) $\boldsymbol{k}_{\text{vert}} \star \boldsymbol{x}$

(c) $\boldsymbol{k}_{\text{hor}} \star \boldsymbol{x}$

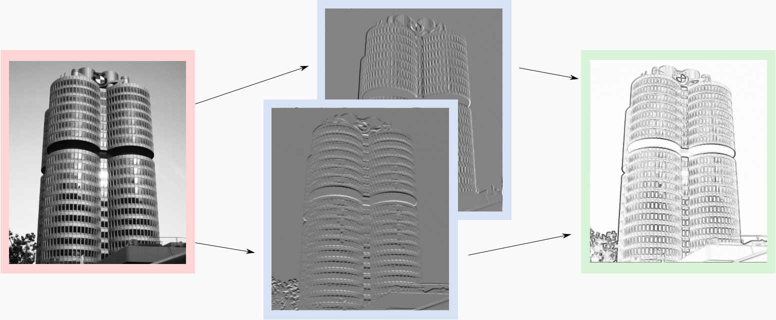
Variants: Gradient, Sobel (1968), ...

Convolutions and filter banks

Example 1: Prewitt filters (1970)

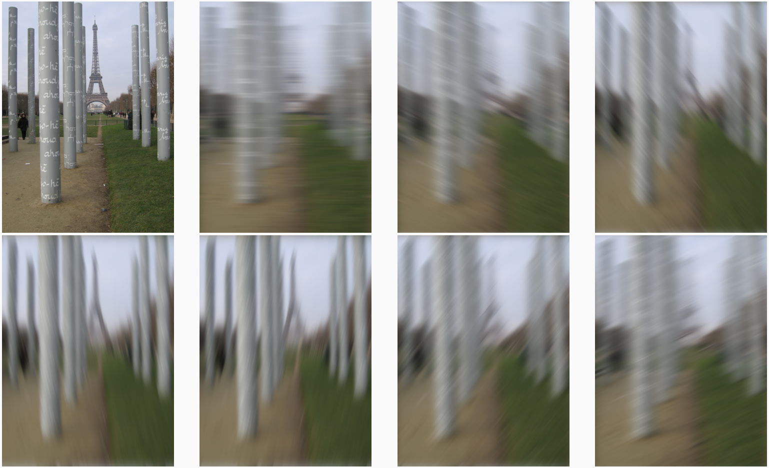
Detect edges? Pool the answers:

$$\max(|\boldsymbol{k}_{\text{vert}} * \boldsymbol{x}|, |\boldsymbol{k}_{\text{hor}} * \boldsymbol{x}|) \quad \text{or} \quad \sqrt{|\boldsymbol{k}_{\text{vert}} * \boldsymbol{x}|^2 + |\boldsymbol{k}_{\text{hor}} * \boldsymbol{x}|^2}$$



Convolutions and filter banks

Example 2: Oriented Gaussian filters



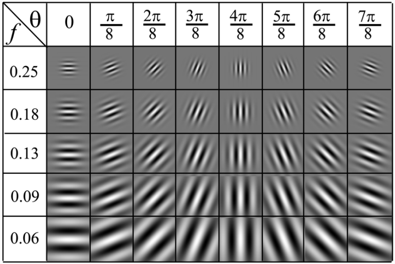
Remove high-frequencies in a given orientation θ.

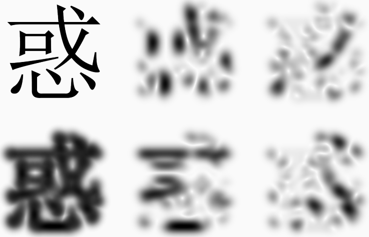
Example 3: Gabor filters

Extract oscillating features with orientation θ and frequency f

$$k_{i,j} = \exp\left(-\frac{i^2 + j^2}{2\tau^2}\right)\cos(2\pi f(i\cos\theta + j\sin\theta))$$

and τ acts on the support of the neighborhood being analyzed.

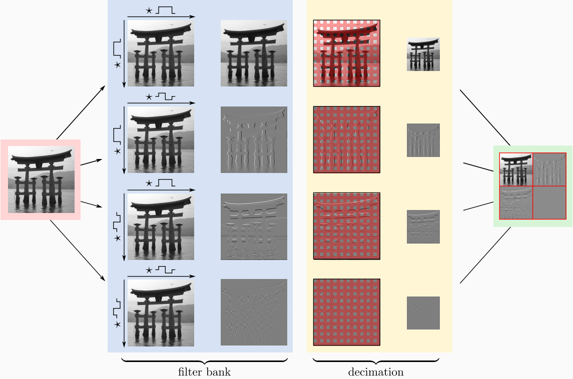




Was used for OCR, Iris and fingerprint recognition (prior to deep learning).

Example 4: Wavelets (Haar, 1901)

Wavelet representations are obtained by applying a (specific) filter bank first,



Wavelets:
bank of orthogonal filters obtained by composition of averaging and derivative filters in vertical and horizontal directions.

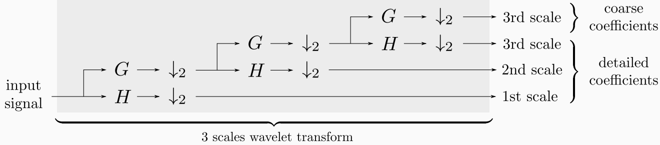
and a decimation step ⇒ invertible transform (as many coefficients as pixels).

Application: denoising by non-linear shrinkage of the coeffs (thresholding).

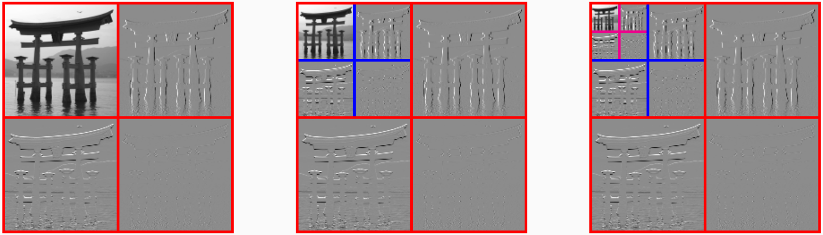
Convolutions and filter banks

Example 5: Multi-scale analysis (Mallat, 1989)

Wavelets can be used recursively (in cascade),



it creates a sparse multi-scale representation of the data (hierarchy).



(a) Level 1

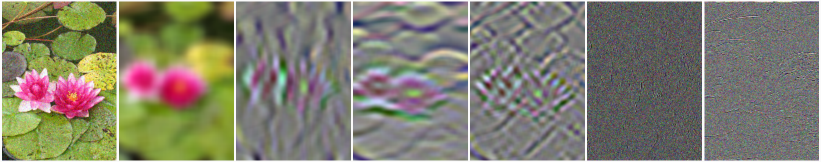
(b) Level 2

(c) Level 3

Convolutions and filter banks

Example 6: **Frame analysis** (Holschneider *et al.*, 1989)

Similar to wavelet representation but redundant (more coefficients than pixels).



More efficient than using a wavelet basis in several applications of pattern recognition and image restoration (prior to deep learning).

Deep learning point of view

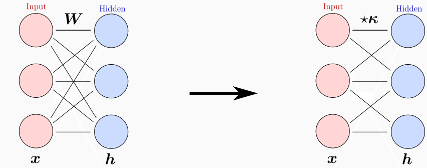
- Don't use hand-crafted filters, learn them!
- Add some non-linearities between layers to get more relevant stuff.

Convolutional neural networks

Convolutional neural networks

What are CNNs?

- Essentially neural networks that use convolution in place of general matrix multiplications at least for the first layers.



Fully connected (FC) layer

Convolutional layer

- CNNs are designed to process the data in the form of multidimensional arrays/tensors (*e.g.*, 2D images, 3D video/volumetric images).
- Composed of series of stages: convolutional layers and pooling layers.
- Units connected to local regions in the feature maps of the previous layer.
- Do not only mimic the brain connectivity but also the visual cortex.

Convolutional neural networks

CNNs are composed of three main ingredients:

1. Local receptive fields
   - hidden units connected only to a small region of their input,
2. Shared weights
   - same weights and biases for all units of a hidden layer,
3. Pooling
   - condensing hidden layers.

but also

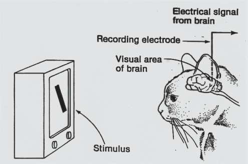
4. Redundancy: more units in a hidden layer than inputs,
5. Sparsity: units should not all fire for the same stimulus.

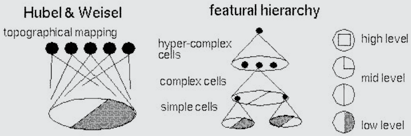
All take inspiration from the visual cortex.

Convolutional neural networks

Hubel & Wiesel (1959, 62, 68, . . . , Nobel prize in 1981)

Show different stimuli to a cat and observe its brain activity.





- **Simple cells:** some cells fire only in the presence of a vertical edge at a particular location in the visual field, while other nearby cells respond to edges of other orientations in that same region of the visual field (∼50ms).
- **Complex cells:** some others behave similarly but respond to these edges in a larger region of the visual field (∼70ms).
- **Hyper-complex cells:** respond to more complex combinations of the simple features (∼90ms).

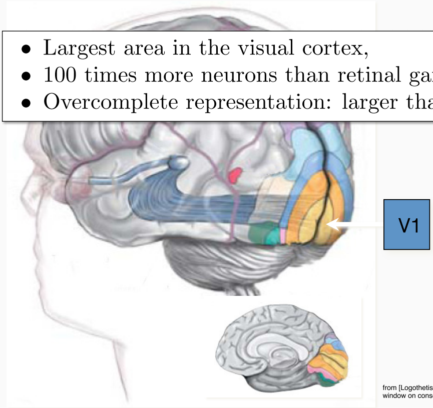
⇒ **local and self-similar receptive fields & low- to high-level features.**

Convolutional neural networks

Olshausen et al. (1996, 2004)

Theoretical and experimental works on mammalian primary visual cortex (V1).

- Largest area in the visual cortex,
- 100 times more neurons than retinal ganglion cells,
- Overcomplete representation: larger than input.



from [Logothetis, N., November 1999. Vision: A window on consciousness. Scientific American]

Retina ganglion cells: input units receiving information from photoreceptors.

Olshausen et al. (1996, 2004)

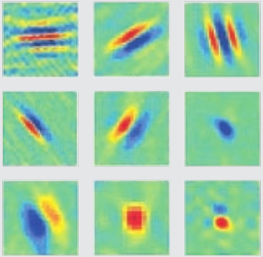
The receptive fields of cells are

1. spatially localized,
2. oriented,
3. bandpass.

Neural responses are:

- sparse thanks to interactions with other areas (redundancy).

receptive fields estimated by reverse correlation:



(Ringach, 2002)

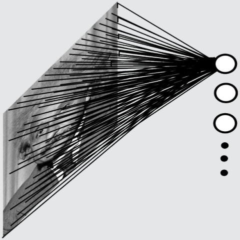
This sparse coding confers several advantages

- eases read out at subsequent levels,
- increases storage capacity in associative memories,
- saves energy.

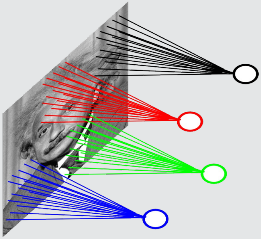
Convolutional neural networks

Local receptive fields → Locally connected layer

- Each unit in a hidden layer can see only a small neighborhood of its input,
- Captures the concept of spatiality.



Fully connected



Locally connected

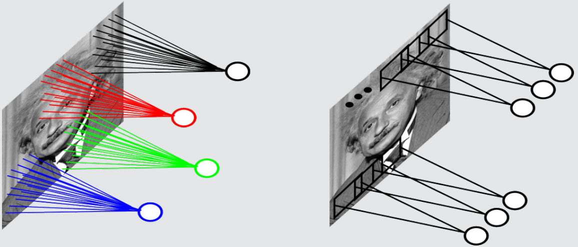
For a 200×200 image and 40,000 hidden units

- Fully connected: 1.6 billion parameters,
- Locally connected $(10 \times 10$ fields): 4 million parameters.

Convolutional neural networks

Self-similar receptive fields → Shared weights

- Detect features regardless of position (translation invariance),
- Use convolutions to learn simple input patterns.



Locally connected

Shared weights

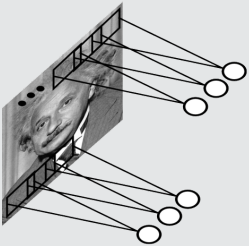
For a 200×200 image and 40,000 hidden units

- Locally connected (10×10 fields): 4 million parameters,
- & Shared weights: 100 parameters (independent of image size).

Convolutional neural networks

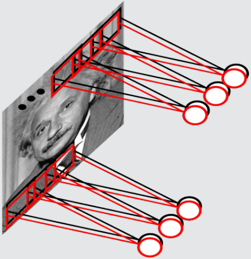
Specialized cells → Filter bank

- Use a filter bank to detect multiple patterns at each location,
- Multiple convolutions with different kernels,
- Result is a 3d array, where each slice is a feature map.



Shared weights

(1 input → 1 feature map)



Filter bank

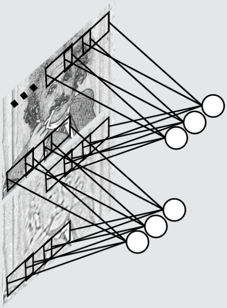
(1 input → 2 feature maps)

- 10×10 fields & 10 output features: 1,000 parameters.

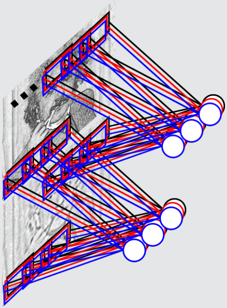
Convolutional neural networks

Hierarchy → inputs of deep layers are themselves 3d arrays

- Learn to filter each channel such that their sum detects a relevant feature,
- Repeat as many times as the desired number of output features should be.



Multi-input filter
(2 inputs → 1 feature map)



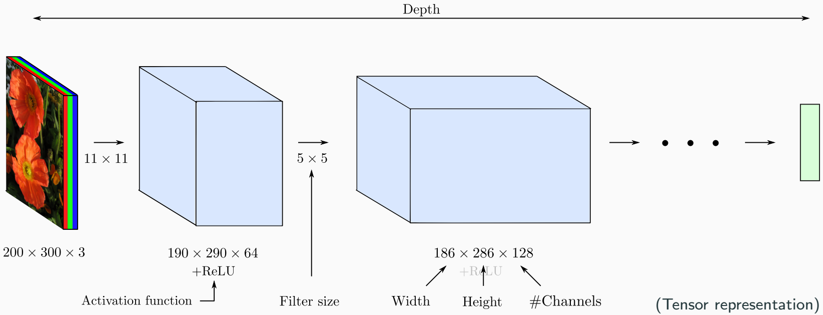
Multi-input filter bank
(2 inputs → 3 feature maps)

- **Remark:** these are not 3d convolutions, but sums of 2d convolutions.

- 10×10 fields & 10 inputs & 10 outputs: 10,000 parameters.

Convolutional neural networks

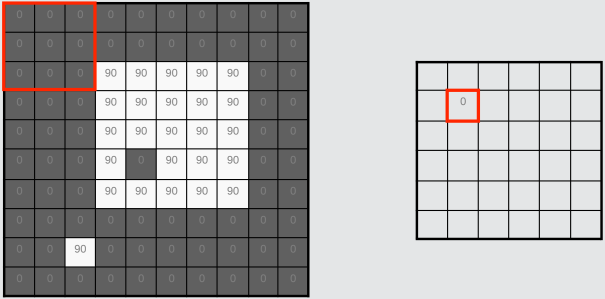
Overcomplete → increase the number of channels



- **Redundancy**: increase the number of channels between layers.
- **Padding**: $n \times n$ conv + *valid* → width and height decrease by $n - 1$.
- Can we control even more the number of simple cells?

Controlling the number of simple cells → Stride

3×3 boxcar strided convolution with stride $s = 2$



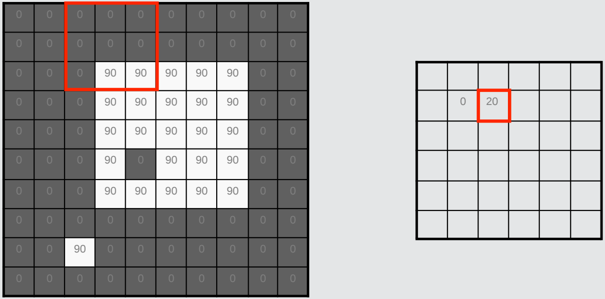
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0				

- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* → width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Controlling the number of simple cells → Stride

3×3 boxcar strided convolution with stride $s = 2$



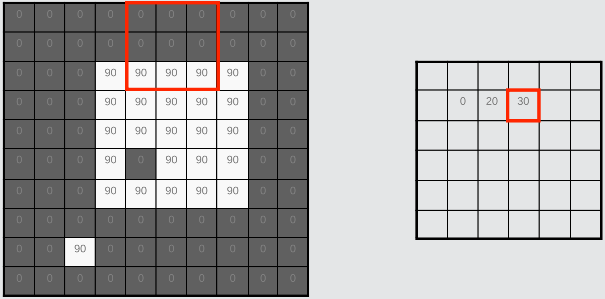
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	20			

- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* → width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Controlling the number of simple cells → Stride

3×3 boxcar strided convolution with stride $s = 2$



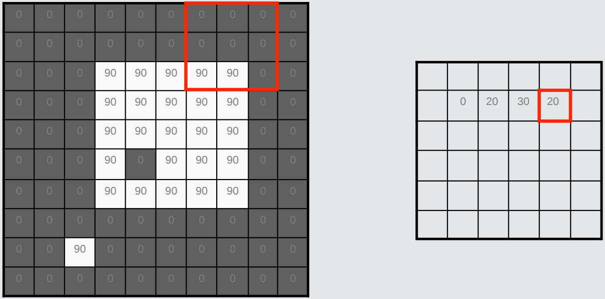
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	20	30		

- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* → width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Controlling the number of simple cells → Stride

3×3 boxcar strided convolution with stride $s = 2$



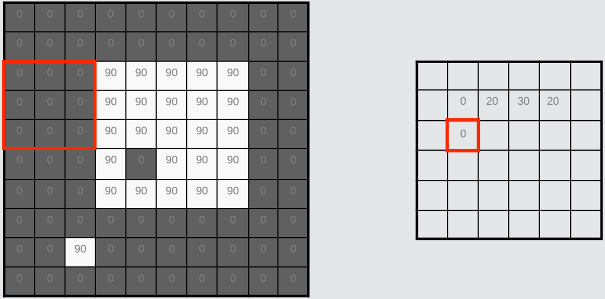
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	20	30	20	

- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* → width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Controlling the number of simple cells → Stride

3×3 **boxcar strided convolution with stride** $s = 2$



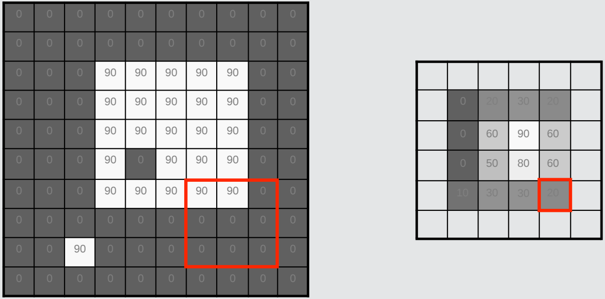
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	20	30	20	
	0				

- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* → width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Controlling the number of simple cells → Stride

3×3 boxcar strided convolution with stride $s = 2$



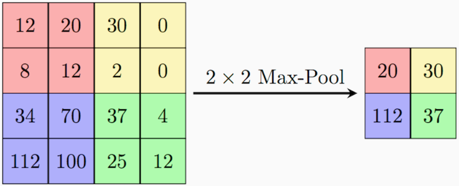
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

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

- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* → width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Complex cells → Pooling layer

- Used after each convolution layer to mimic complex cells,
- Unlike striding, reduce the size by aggregating inputs:
  - Partition the image in a grid of $z \times z$ windows (usually $z = 2$),
  - max-pooling: take the $\max$ in the window,



12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

$\xrightarrow{2 \times 2 \text{ Max-Pool}}$

20	30
112	37

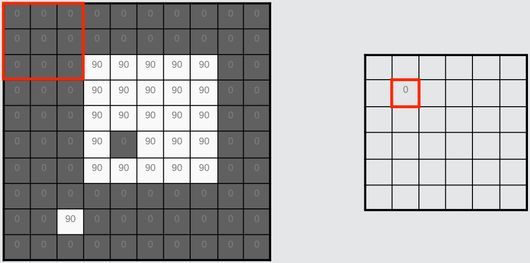
  - average-pooling: take the average,
  - ℓ_p-pooling: take the Hölder mean (generalization):

$$h_{i,j} = \left(\frac{1}{z^2} \sum_{k,l} x_{i+k,j+l}^p \right)^{1/p} \longrightarrow \begin{cases} \text{Average: } p = 1 \\ \text{Max: } p \to \infty \end{cases}$$

Complex cells → Pooling layer

Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:



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

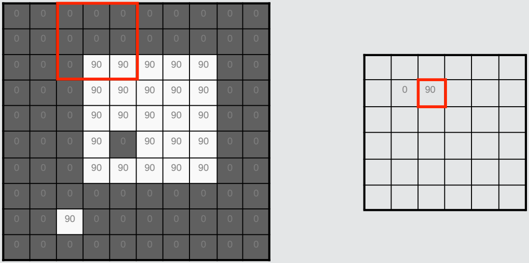
	0				

- When $z = s$: standard pooling (non-overlapping),
- When $z = 1$: the operation is known as dilation.

Complex cells → Pooling layer

Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:



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

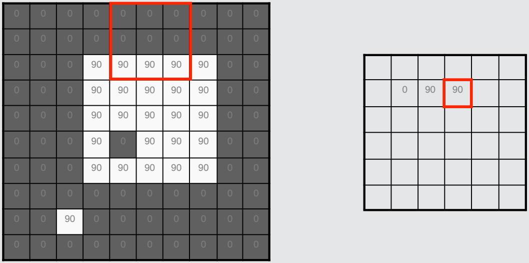
	0	90			

- When $z = s$: standard pooling (non-overlapping),
- When $z = 1$: the operation is known as dilation.

Complex cells → Pooling layer

Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:



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

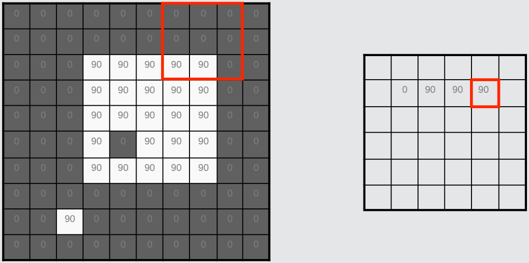
	0	90	90		

- When $z = s$: standard pooling (non-overlapping),
- When $z = 1$: the operation is known as dilation.

Complex cells → Pooling layer

Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:



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

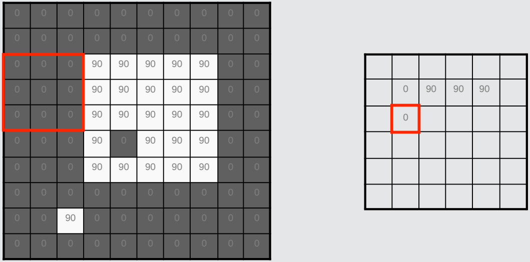
	0	90	90	90	

- When $z = s$: standard pooling (non-overlapping),
- When $z = 1$: the operation is known as dilation.

Complex cells → Pooling layer

Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:



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

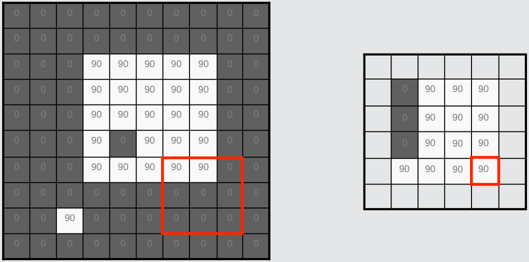
	0	90	90	90	
	0				

- When $z = s$: standard pooling (non-overlapping),
- When $z = 1$: the operation is known as dilation.

Complex cells → Pooling layer

Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:

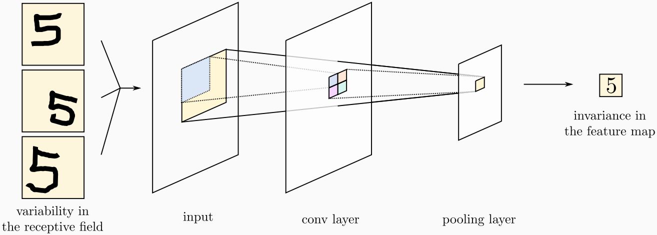


0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

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

- When $z = s$: standard pooling (non-overlapping),
- When $z = 1$: the operation is known as dilation.

Complex cells → Pooling layer



- Makes the output unchanged even if the input is a little bit changed,
- Allows some invariance/robustness with respect to the exact position,
- Simplifies/Condenses/Summarizes the output from hidden layers,
- Increases the effective receptive fields (with respect to the first layer.)

CNNs parameterization

Setting up a convolution layer requires choosing

- Filter size: $n \times n$
- #output channels: C
- Stride: s
- Padding: p

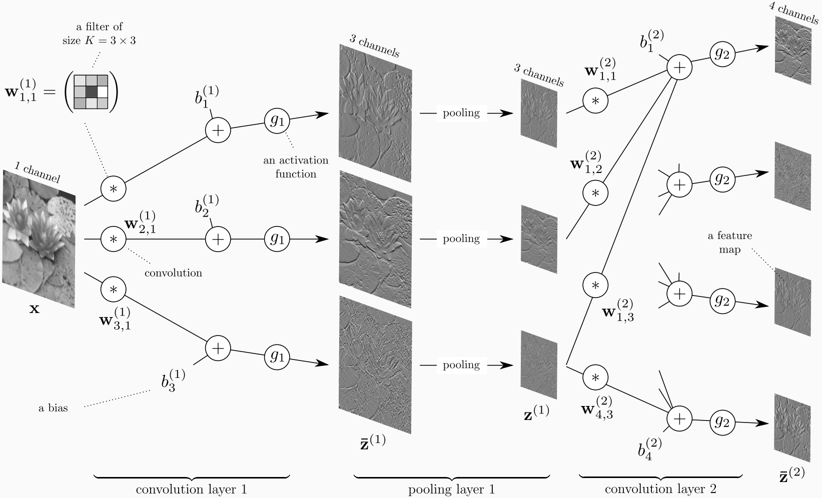
The filter weights $\boldsymbol{\kappa}$ and the bias b are learned by backprop.

Setting up a pooling layer requires choosing

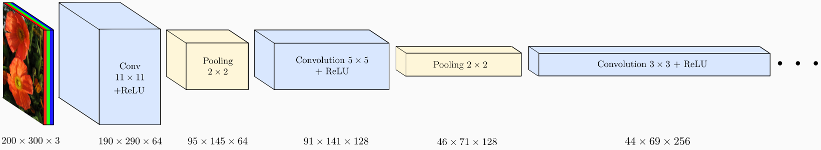
- Pooling size: $z \times z$
- Aggregation rule: max-pooling, average-pooling, . . .
- Stride: s
- Padding: p

No free parameters to be learned here.

All concepts together



All concepts together with tensor representation



General scheme:

- Alternate: Conv + ReLU + pooling,
- Variant: Conv + pooling + ReLU.

Why deep? Allows for a hierarchy of abstractions and invariance:

characters → *words* → *sentence* → *paragraph* → *concepts*

Why redundant/overcomplete? Allows for sparsity, expressivity, efficiency:

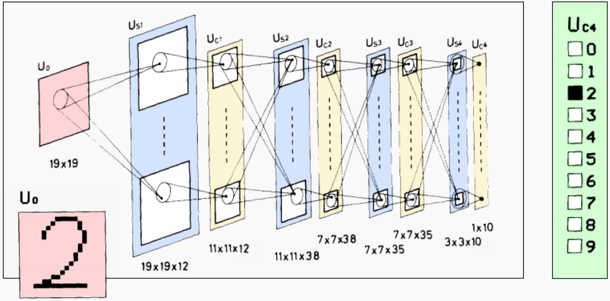
the larger the alphabet, the shorter the words,
the more vocabulary, the fewer words you need to use.

If you repeat enough, you can capture tons of features that are flat, *i.e.*, no longer related to the original representation (space/pixels).

Example: **Neocognitron** (Fukushima, 1979, 1980, 1987)

Though CNNs are attributed to Yann LeCun (1989), they date back from the Neocognitron designed to recognize handwritten digits from 0 to 9.

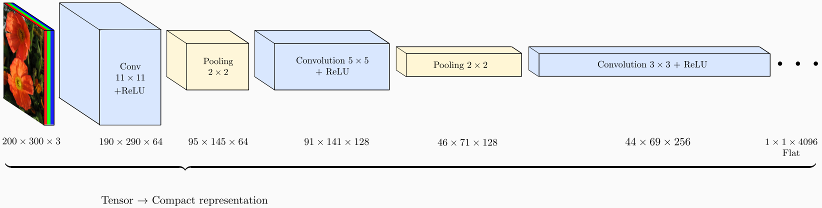
retina simple cells & complex cells output

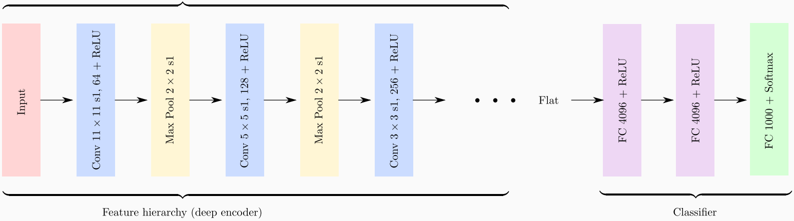


Alternate simple and complex cells, with different numbers of channels, until the final layer has 10 nodes: 10 channels and the space domains is 1×1 (flat) (used average-pooling, did not use backprop or ReLU).

Convolutional neural networks

Instead, modern CNNs make it flat and plug a Fully-Connected (FC) classifier

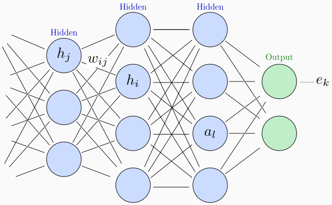



Input
Conv 11 × 11 s1, 64 + ReLU
Max Pool 2 × 2 s1
Conv 5 × 5 s1, 128 + ReLU
Max Pool 2 × 2 s1
Conv 3 × 3 s1, 256 + ReLU
• • •
Flat
FC 4096 + ReLU
FC 4096 + ReLU
FC 1000 + Softmax
Feature hierarchy (deep encoder)
Classifier

Example of a typical modern architecture

CNNs and backprop

Reminder of backprop with fully connected layer



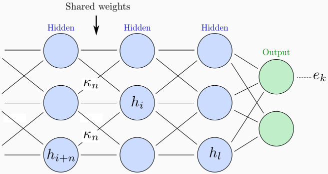
- j: neuron in the previous hidden layer
- h_j: response of hidden neuron j
- $w_{i,j}$: synaptic weights between j and i
- h_i: response of hidden neuron i

$$h_i = g\left(a_i\right) \quad \text{with} \quad a_i = \sum_j w_{i,j} h_j$$

Apply chain rule:

$$\begin{aligned}\frac{\partial E(\boldsymbol{W})}{\partial w_{i,j}} &= \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_k \frac{\partial e_k}{\partial w_{i,j}} = \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_k \frac{\partial e_k}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_{i,j}} \\ &= \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_k \left(\sum_l \frac{\partial e_k}{\partial a_l} \frac{\partial a_l}{\partial h_i} \right) \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_{i,j}} \\ &= \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_l \underbrace{\left(\sum_k \frac{\partial e_k}{\partial a_l} \right)}_{\delta_l} \frac{\partial a_l}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial w_{i,j}}\end{aligned}$$

Backprop with convolutional layer



- h_j: response of hidden neuron j
- κ_n: weight between $j = i + n$ and i
- h_i: response of hidden neuron i

$$h_i = g\left(a_i\right) \quad \text{with} \quad a_i = \sum_n \kappa_n h_{i+n}$$

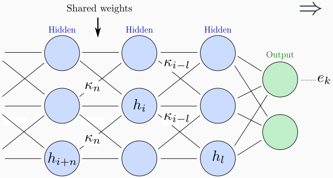
$$\begin{aligned}
\frac{\partial E(\boldsymbol{W})}{\partial \kappa_n} &= \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_k \frac{\partial e_k}{\partial \kappa_n} = \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_k \sum_i \frac{\partial e_k}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial \kappa_n} \\
&= \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_k \sum_i \left(\sum_l \frac{\partial e_k}{\partial a_l} \frac{\partial a_l}{\partial h_i} \right) \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial \kappa_n} \\
&= \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_l \underbrace{\left(\sum_k \frac{\partial e_k}{\partial a_l} \right)}_{\delta_l} \sum_i \frac{\partial a_l}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial \kappa_n}
\end{aligned}$$

Backprop with convolutional layer

$$a_l = \sum_{n'} \kappa_{n'} h_{l+n'} \quad \Rightarrow \quad \frac{\partial a_l}{\partial h_i} = \kappa_{i-l}$$

$$h_i = g(a_i) \quad \Rightarrow \quad \frac{\partial h_i}{\partial a_i} = g'(a_i),$$

$$a_i = \sum_{n'} \kappa_{n'} h_{i+n'} \quad \Rightarrow \quad \frac{\partial a_i}{\partial \kappa_n} = h_{i+n}$$



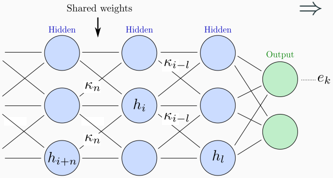
$$\Rightarrow \quad \frac{\partial E(\boldsymbol{W})}{\partial \kappa_n} = \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_l \delta_l \sum_i \frac{\partial a_l}{\partial h_i}\frac{\partial h_i}{\partial a_i}\frac{\partial a_i}{\partial \kappa_n}$$

$$= \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_i \underbrace{\left(\sum_l \kappa_{i-l}\delta_l\right) g'(a_i)}_{\delta_i} h_{i+n}$$

$$= \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_i \delta_i h_{i+n}$$

Backprop with convolutional layer

$$
\begin{aligned}
a_l &= \sum_{n'} \kappa_{n'} h_{l+n'} &\Rightarrow\quad \frac{\partial a_l}{\partial h_i} &= \kappa_{i-l} \\
h_i &= g(a_i) &\Rightarrow\quad \frac{\partial h_i}{\partial a_i} &= g'(a_i), \\
a_i &= \sum_{n'} \kappa_{n'} h_{i+n'} &\Rightarrow\quad \frac{\partial a_i}{\partial \kappa_n} &= h_{i+n}
\end{aligned}
$$



$$
\begin{aligned}
\Rightarrow\quad \frac{\partial E(\boldsymbol{W})}{\partial \kappa_n} &= \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_l \delta_l \sum_i \frac{\partial a_l}{\partial h_i} \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial \kappa_n} \\
&= \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \sum_i \underbrace{(\boldsymbol{\kappa}^* \star \boldsymbol{\delta})_i \quad g'(a_i)}_{\delta_i} \, h_{i+n} \\
&= \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} (\boldsymbol{\delta} \star \boldsymbol{h})_n
\end{aligned}
$$

Backpropagation with both FC and convolutional layers

Multidimensional/Tensor form

With **matrix-vector and convolution notations** for each layer:
(k denotes the layer)

If k is a FC layer: $(\boldsymbol{h}_0 = \boldsymbol{x})$

$$\nabla_{\boldsymbol{W}_k} E(\boldsymbol{W}) = \boldsymbol{\delta}_k \boldsymbol{h}_{k-1}^T$$

If k is a convolutional layer:

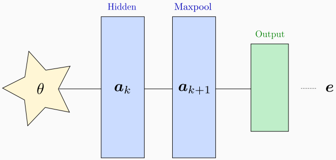
$$\nabla_{\boldsymbol{\kappa}_k} E(\boldsymbol{W}) = \boldsymbol{\delta}_k \star \boldsymbol{h}_{k-1} \quad \text{(cropped to the size of the filter)}$$

$$\text{where} \quad \boldsymbol{\delta}_k = g'(\boldsymbol{a}_k) \times \begin{cases} \boldsymbol{y}_k - \boldsymbol{d}_k & \text{for } k \text{ a SSE output layer} \\ \boldsymbol{W}_{k+1}^T \boldsymbol{\delta}_{k+1} & \text{for } k+1 \text{ a FC layer} \\ \boldsymbol{\kappa}_{k+1}^* \star \boldsymbol{\delta}_{k+1} & \text{for } k+1 \text{ a convolutional layer} \end{cases}$$

But what about pooling?

Backprop with pooling layer

$$\boldsymbol{a}_{k+1} = \mathrm{maxpool}(\boldsymbol{a}_k)$$



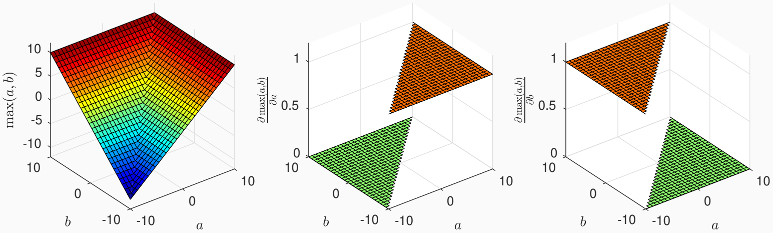
$$\Rightarrow \quad \frac{\partial E(\boldsymbol{W})}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{a}_{k+1}} \frac{\partial \boldsymbol{a}_{k+1}}{\partial \boldsymbol{a}_k} \frac{\partial \boldsymbol{a}_k}{\partial \boldsymbol{\theta}} = \underbrace{\boldsymbol{\delta}_{k+1} \frac{\partial \mathrm{maxpool}(\boldsymbol{a}_k)}{\partial \boldsymbol{a}_k}}_{\boldsymbol{\delta}_k} \frac{\partial \boldsymbol{a}_k}{\partial \boldsymbol{\theta}}$$

How to compute: $\boldsymbol{\delta}_k = \boldsymbol{\delta}_{k+1} \frac{\partial \mathrm{maxpool}(\boldsymbol{a}_k)}{\partial \boldsymbol{a}_k}$?

Backprop with max-pooling layer

- As ReLU, the max operation is continuous and almost everywhere differentiable, such that we can use a sub-gradient

$$\frac{\partial \max(a,b)}{\partial a} = \begin{cases} 1 & \text{if} \quad a \geqslant b \\ 0 & \text{if} \quad a < b \end{cases}$$



Backprop with max-pooling layer

- The Jacobian matrix of $\max$ is then given by

$$\frac{\partial \max(\boldsymbol{h})}{\partial \boldsymbol{h}} = \begin{pmatrix} \frac{\partial \max(\boldsymbol{h})}{\partial h_1} & \dots & \frac{\partial \max(\boldsymbol{h})}{\partial h_n} \end{pmatrix}$$

$$\frac{\partial \max(\boldsymbol{h})}{\partial \boldsymbol{h}} = \begin{pmatrix} 0 & \dots & 0 & \underset{\substack{\uparrow \\ \text{position of the max}}}{1} & 0 & \dots & 0 \end{pmatrix}$$

- Example:

$$\boldsymbol{h} = \begin{pmatrix} 3 & 5 & 6 & 2 & 2 & 9 & 4 \end{pmatrix}^T$$

$$\frac{\partial \max(\boldsymbol{h})}{\partial \boldsymbol{h}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Backprop with max-pooling layer

- Consider the 1d maxpool in a sliding window of size 3:

$$\boldsymbol{h} = \begin{pmatrix} 3 & 5 & 6 & 2 & 2 & 9 & 4 \end{pmatrix}^T$$

$$\mathrm{maxpool}(\boldsymbol{h}) = \begin{pmatrix} 5 & 6 & 6 & 6 & 9 & 9 & 9 \end{pmatrix}^T$$

- The Jacobian matrix of $\mathrm{maxpool}$ is then

$$\frac{\partial \mathrm{maxpool}(\boldsymbol{h})}{\partial \boldsymbol{h}} = \begin{pmatrix} 0 & 1 & & & & & \\ 0 & 0 & 1 & & & & \\ & 0 & 1 & 0 & & & \\ & & 1 & 0 & 0 & & \\ & & & 0 & 0 & 1 & \\ & & & & 0 & 1 & 0 \\ & & & & & 1 & 0 \end{pmatrix}$$

- This matrix is huge. Never store/manipulate such a matrix!
- Instead, implement its action when involved in matrix/vector products.

CNNs and backprop

Backprop with max-pooling layer

- Given $$\boldsymbol{\delta}_{k+1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix}$$

- The update becomes

$$\boldsymbol{\delta} = \boldsymbol{\delta}_{k+1} \frac{\partial \text{maxpool}(\boldsymbol{h})}{\partial \boldsymbol{h}} = \boldsymbol{\delta}_{k+1} \begin{pmatrix} 0 & 1 & & & & & \\ 0 & 0 & 1 & & & & \\ & 0 & 1 & 0 & & & \\ & & 1 & 0 & 0 & & \\ & & & 0 & 0 & 1 & \\ & & & & 0 & 1 & 0 \\ & & & & & 1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 1 & 9 & 0 & 0 & 18 & 0 \end{pmatrix}$$

- **Forward step:** take the max and memorize its position in the window,
- **Backward step:** distribute the errors according to these original positions.

But what about striding?

Backprop with striding

- Consider the 1d striding with step $s = 2$:

$$\begin{aligned} \boldsymbol{h} &= \begin{pmatrix} 3 & 5 & 6 & 2 & 2 & 9 & 4 & 5 \end{pmatrix}^T \\ \mathrm{striding}(\boldsymbol{h}) &= \begin{pmatrix} 3 & 6 & 2 & 4 \end{pmatrix}^T \end{aligned}$$

- The Jacobian matrix of $\mathrm{striding}$ is then

$$\frac{\partial \mathrm{striding}(\boldsymbol{h})}{\partial \boldsymbol{h}} = \begin{pmatrix} 1 & 0 & & & & & & \\ & & 1 & 0 & & & & \\ & & & & 1 & 0 & & \\ & & & & & & 1 & 0 \end{pmatrix}$$

- Again, just implement its action when involved in matrix/vector products.

Backprop with striding

- Given $$\boldsymbol{\delta}_{k+1} = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}$$
- The update becomes

$$\begin{aligned}\boldsymbol{\delta} = \boldsymbol{\delta}_{k+1} \frac{\partial \text{striding}(\boldsymbol{h})}{\partial \boldsymbol{h}} &= \boldsymbol{\delta}_{k+1} \begin{pmatrix} 1 & 0 & & & & & & \\ & & 1 & 0 & & & & \\ & & & & 1 & 0 & & \\ & & & & & & 1 & 0 \end{pmatrix} \\ &= \quad \begin{pmatrix} 1 & 0 & 2 & 0 & 3 & 0 & 4 & 0 \end{pmatrix}\end{aligned}$$

- **Forward step:** take one every s elements,
- **Backward step:** inject $s - 1$ zeros after each element.

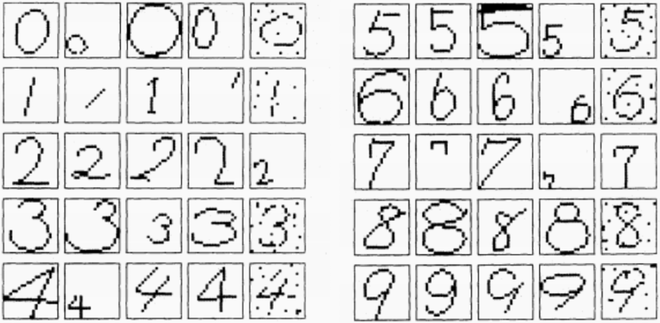
Successful CNNs architectures

Successful CNNs architectures

Neocognitron (Fukushima, 1979, 1980, 1987)

In the history of Optical Character Recognition (OCR), first system working

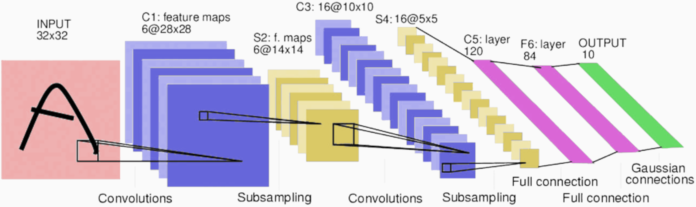
- regardless of where the digits were placed in the field of view,
- high degree of tolerance to distortion of the character,
- fairly insensitive to the size of the character.



Some digits correctly recognized

Successful CNNs architectures

LeNet-5 (LeCun, Bottou, Bengio and P. Haffner, 1998)



- Classifier for 32×32 hand-written digits,
- Was used for reading $\approx$ 10% of the checks in North America,
- Demo: `http://yann.lecun.com/exdb/lenet/`,
- Used hyperbolic tangent connection after convolutions,
- Plugged a linear classifier using Gaussian connections (RBF kernels):

$y_j = \sum_i (h_i - w_{ij})^2$ instead of classical dot-products $y_j = \sum_i w_{ij} h_i$.

First deep architecture (5 layers) that was successfully trained.

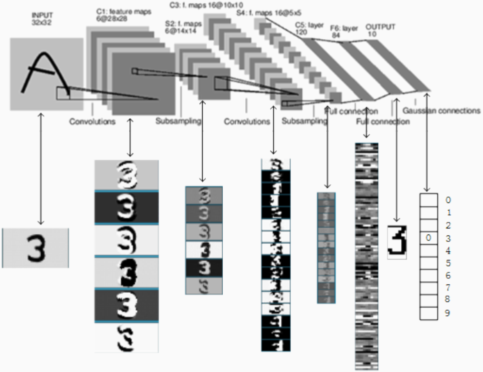
Successful CNNs architectures

LeNet-5 (LeCun, Bottou, Bengio and P. Haffner, 1998)

- Trained on MNIST:
  - 28×28 images,
  - 60,000 for training,
  - 10,000 for testing.
- Used data augmentation:
  - generated random distortions,
  - extended to 32×32px.
  - 600,000 training samples,
- Backprop with SGD:
  - only 20 epochs,
  - no mini-batches,
  - 60,000 parameters,
  - about 3 days (at that time).

Successful CNNs architectures

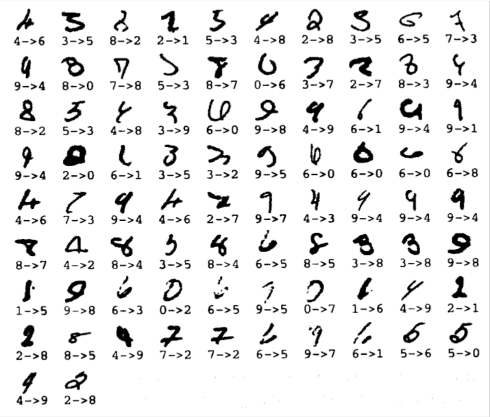
LeNet-5 (LeCun, Bottou, Bengio and P. Haffner, 1998)



What did it learn? Look at the evolution of the features in the network.

Successful CNNs architectures

LeNet-5 (LeCun, Bottou, Bengio and P. Haffner, 1998)

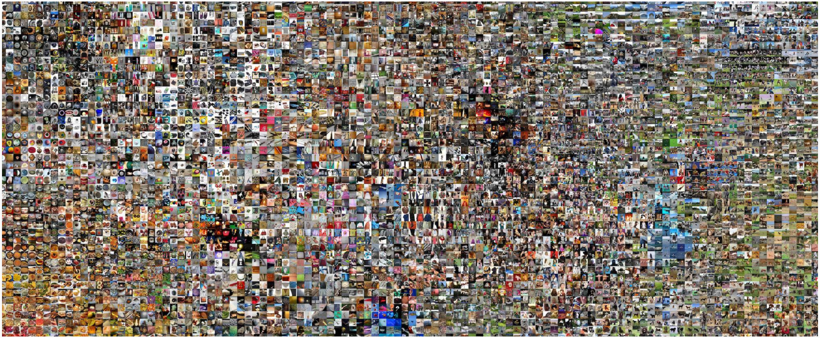


Made only 82 errors among the 10,0000 samples of the testing dataset.
Nowadays, the best results on MNIST are of 23 errors (Cireșan *et. al*, 2012).

But with the proliferation of multimedia, MNIST challenge became outdated.

Successful CNNs architectures

ImageNet



ImageNet:

- 12 million labeled images,
- 22,000 classes,
- Labeled by crowd-sourcing (Amazon Mechanical Turk).

ImageNet challenge (ILSVRC)

ILSVRC:

- Annual challenges since 2010,
- Limited to 1,000 classes,
- 1.2 million of 256×256 images for training,
- 50,000 images for validation and 100,000 for testing.

Successful CNNs architectures

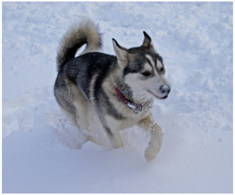
ImageNet classification challenge (ILSVRC)

Ranking: top-5 error

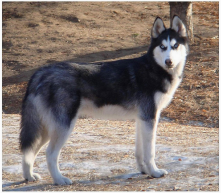
Make five guesses – Correct if the desired label is within these guesses.

top-5 error rate: percentage of incorrect such answers.

why? desired labels can be ambiguous



(a) Siberian husky

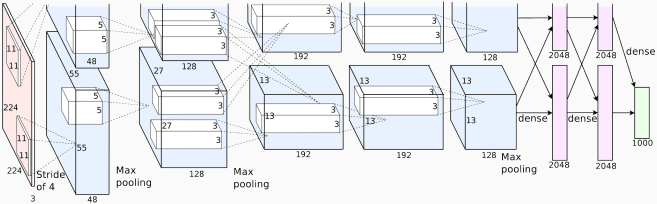


(b) Eskimo dog

And the 2012 winner was...

Successful CNNs architectures

AlexNet (Krizhevsky, Sutskever, Hinton, 2012)



- Similar to LeNet-5, a bit deeper (8 layers), but much larger.
- Use ReLU after convolutions, use softmax for the output layer.

Introduced concepts:

- **Specific GPU architecture:** split the channels in two stages in order to be trained simultaneously on two Nvidia Geforce GTX 580 GPUs.
- **Local Response Normalization:** after applying the first 2 filter banks, they normalize each feature map with respect to their adjacent ones.
- **Overlapping pooling:** use striding also after pooling with $z = 3$ and $s = 2$.

Successful CNNs architectures

AlexNet (Krizhevsky, Sutskever, Hinton, 2012)

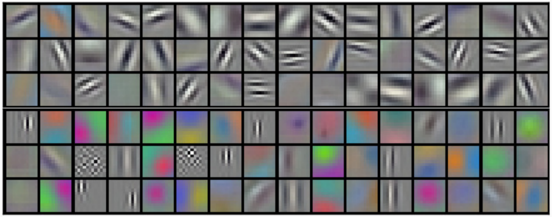
Other settings:

- **Number of parameters**: 60 million,
- **Data augmentation:** horizontal reflection, 224×224 subimages from 256×256 images, altering RGB channels. At test time, average the prediction obtained on 10 subimages extracted from the original one.
- **Dropout:** used for hidden neurons with probability $.5$.
- **Optimization:** SGD with mini-batch size 128 + weight decay + momentum → Took 6 days for 90 epochs.
- **Combination:** Trained 7 such networks and average their predictions (some of them were pretrained on other datasets and fine-tuned.)

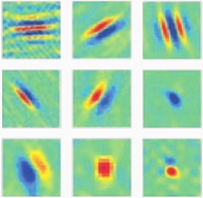
Successful CNNs architectures

AlexNet (Krizhevsky, Sutskever, Hinton, 2012)

What did it learn? Look at the filters that have been learned.



receptive fields estimated by reverse correlation:

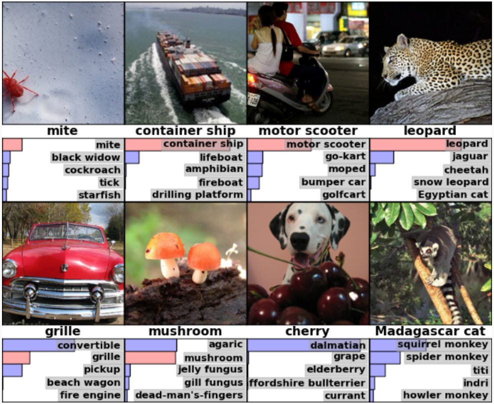


(left) 96 filters learned in the first convolution layer in AlexNet.

Many filters turn out to be edge detectors, similar to Gabor filters, common to human visual systems (right).

Successful CNNs architectures

AlexNet (Krizhevsky, Sutskever, Hinton, 2012)

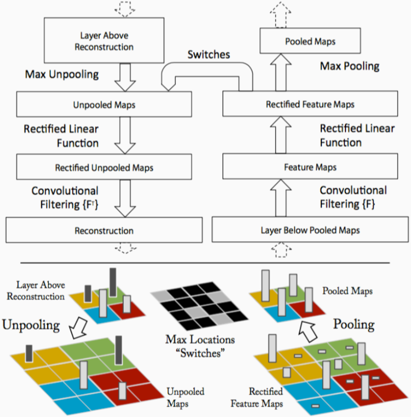


- Won ILSVRC 2012 by reducing the top-5 error to 16.4%.
- The second place, which was not a CNN, was around 26.2%.

Successful CNNs architectures

ZFNet (Zeiler & Fergus, 2013)

Visualizing the evolution of features, or the learned filters is not informative enough to understand what has been learned.



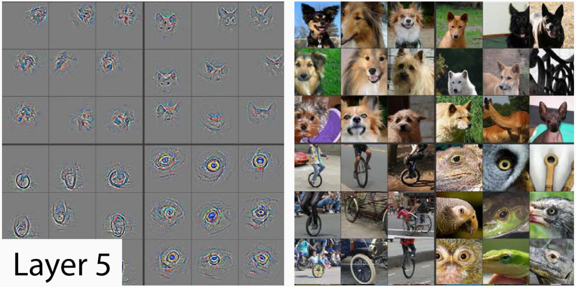
Introduced concept

Visualization with DeconvNet: Offered a way to project an intermediate feature back to the image: starting from this feature, browse the network backward, replace pooling by unpooling, and convolutions by transposed convolutions (*i.e.*, flip the filters, aka, deconvolution layer).

We will go back to this later.

Successful CNNs architectures

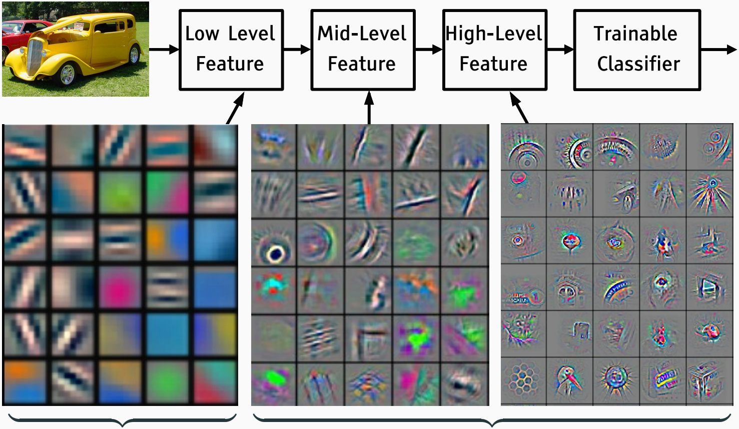
ZFNet (Zeiler & Fergus, 2013)



- Look at the intermediate features of AlexNet at each layer,
- Inspect the features and make some changes accordingly:
  - Layer 1: Conv 11×11 s4 → Conv 7×7 s2,
  - Layer 2: Conv 5×5 s1 → Conv 5×5 s2,
  - Use also Local Response Normalization,
  - Do not use a split architecture in two stages for GPU.

Successful CNNs architectures

ZFNet (Zeiler & Fergus, 2013)



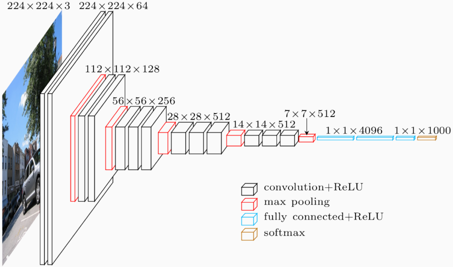
That's how we can produce such beautiful illustrations!

ZFNet (Zeiler & Fergus, 2013)

- Other settings:
  - **Number of parameters**: < 60 million,
  - **Data-augmentation**: crops and flips,
  - **Dropout**: used for FC layers with probability $.5$,
  - **Filter normalization**: when their norm exceed a threshold,
  - **Optimization**: SGD with mini-batch size 128 + momentum
    → Took 12 days for 70 epochs on a single Nvidia GTX 580 GPU.
  - **Combination:** Average the prediction of 5 such CNNs and another one with a slightly different architecture:
    - Layer 3,4,5: #Channels 384, 384, 256 → 512, 1024, 512.
- Won ILSVRC 2013 with 11.7% top-5 error (AlexNet: 16.4%).

Successful CNNs architectures

VGG (Simonyan & Zisserman, 2014)



Introduced concept

Deep and simple:

- 16 conv filters, 3×3 s1,
- 5 max pool, 2×2 s2,
- 3 FC layers,
- No need of local response normalization.

Why does it work?

- Two first 3×3 conv layers: effective receptive field is 5×5,
- Three first 3×3 conv layers: effective receptive field is 7×7,
- Why is it better than ZFNet which uses 7×7?
  - More discriminant: 3 ReLUs instead of 1 ReLU,
  - Less parameters: $3 \times (3 \times 3) = 27$ vs $1 \times (7 \times 7) = 49$.
- Next, apply max-pooling and the effective receptive field double!

Successful CNNs architectures

VGG (Simonyan & Zisserman, 2014)

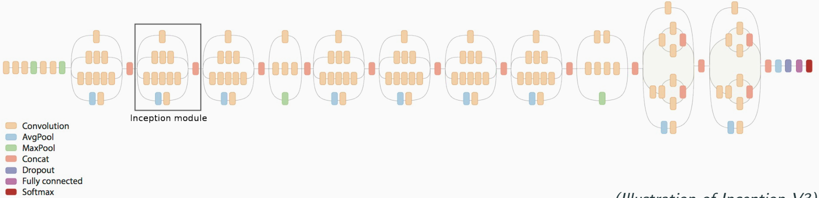
Other settings:

- **Number of parameters**: 140 million,
- **Data-augmentation**: crops, flips, RGB color shift and scaling jittering.
- **Dropout**: used for the first two FC layers with probability $.5$,
- **Optim**: SGD with mini-batch size 256 + momentum + weight decay
  → Took about 3 weeks for 74 epochs on 4 Nvidia Titan Black GPUs.
- **Combination:** Combined 7 such CNNs.

Runner-up of ILSVRC 2014 with 7.3% top-5 error (ZFNet: 11.7%).
Not winner but very influential: "Keep it deep. Keep it simple".

Successful CNNs architectures

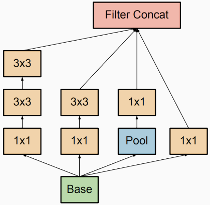
Inception/GoogLeNet (Szegedy *et al.*, 2014), V2 (2015), V3 (2016), V4 (2017)



(Illustration of Inception V3)

Introduced concept:

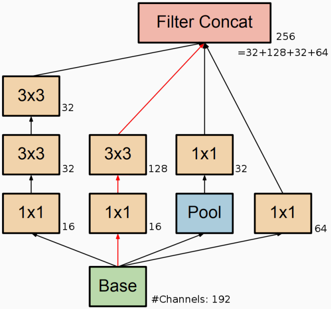
- **Inception module:** apply filters of different sizes in parallel and concatenate their answers. Each module captures in parallel details at different scales and has thus different effective receptive fields.
- **Deeper with fewer parameters:** introduced "1×1" convolutions for dimensionality reduction. The first version had 22 layers and 4 million parameters! (AlexNet has 8 layers and 60 million parameters).



(An inception module)

Successful CNNs architectures

Inception/GoogLeNet (Szegedy *et al.*, 2014), V2 (2015), V3 (2016), V4 (2017)



What are 1×1 convolutions?

- 1×1 (multi) convolutions: outputs are simply element-wise weighted sums of the inputs. Producing less outputs than inputs allows for dimensionality reduction.

Why do they perform dimensionality reduction?

- They can take 192 feature maps and reduce the information to 16 feature maps (with same width and height).

Why do they allow for reducing the number of parameters?

- Consider the branch in red:
  - Without: $192 \times 3 \times 3 \times 128 \approx 221{,}000$ parameters,
  - With: $192 \times 16 + 16 \times 3 \times 3 \times 128 \approx 21{,}500$ parameters.

They also decrease by the same factor the computation cost!

Successful CNNs architectures

Inception/GoogLeNet (Szegedy *et al.*, 2014), V2 (2015), V3 (2016), V4 (2017)

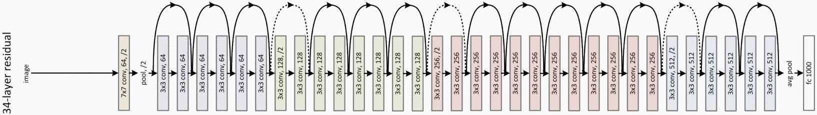
Other settings:

- **Number of parameters**: 6.8 million (VGG: 140M),
- **Data-augmentation**: crops of different sizes, flips, random interpolations.
- **Dropout**: used for one FC layer with probability $.4$,
- **Optimization**: SGD + momentum + learning rate schedule
  $\rightarrow$ Authors suggest it may take about a week with few GPUs.
- **Combination:** Average the prediction of 7 CNNs over 144 crops per images (shifted, rescaled and flipped).

Won ILSVRC 2014 with 6.7% top-5 error (VGG: 7.3%).

Successful CNNs architectures

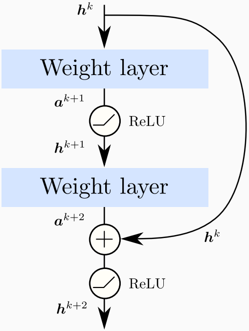
ResNet (Microsoft – He *et al.*, 2016)



(short version only 34 layers here!)

Introduced concept:

- **Be ultra deep:** 152 layers! First layer uses 7×7 convolutions and everything else is 3×3 with only two pooling layers.
- **Residual layer:** learn how to change the input instead of learning what the output should be (using shortcut connections). Refer to the previous class.



Successful CNNs architectures

ResNet (Microsoft – He *et al.*, 2015)

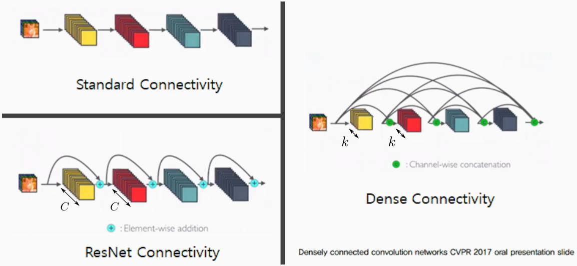
Other settings:

- **Number of parameters**: 60 million,
- **Data-augmentation**: resized, cropped, flipped.
- **Batch-normalization**: between convolutions and ReLU (no dropout).
- **Optimization**: SGD with mini-batch size 256 + momentum
  + weight decay + learning rate schedule
  + He initialization (He *et al.*, 2015. Same guy)
  → Trained for about two weeks using 8 GPUs.
- **Combination:** average predictions of 6 CNNs (only two with 152 layers)
  over ten crops and multiple scales.

Won ILSVRC 2015 with 3.57% top-5 error (GoogLeNet, 6.7%).
First time a system beats human level prediction on ImageNet (about 5.1%).

Successful CNNs architectures

DenseNet (Huang *et al.*, 2017)



Densely connected convolution networks CVPR 2017 oral presentation slide

Introduced concept:

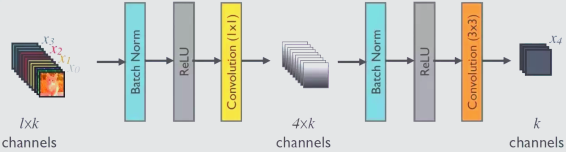
- **Dense connectivity:** Each layer at level l received the concatenation of all previous feature maps of size k as input: $l \times k$ channels . Advantages:
  1. aggregate information from different levels of abstraction,
  2. layers can be slim (k small), and the network can be deep (l large),
  3. reduce even more the number of parameters to learn:

$$\text{ResNet: } C \times C \quad \text{vs} \quad \text{DenseNet: } l \times k \times k \quad \text{with} \quad k \ll C$$

Successful CNNs architectures

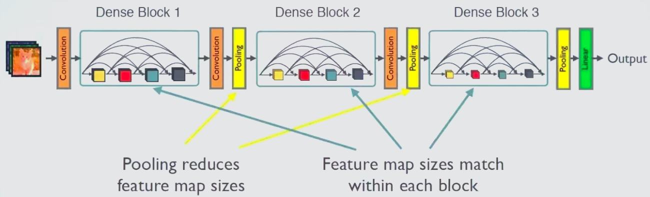
DenseNet (Huang *et al.*, 2017)

Since l can be large, use 1×1 convolution to reduce dimension



Since each layer needs to have the same spatial dimension.

→ Use a succession of blocks with dense connectivity separated by pooling:



Pooling reduces feature map sizes

Feature map sizes match within each block

Successful CNNs architectures

DenseNet (Huang *et al.*, 2017)

Other settings:

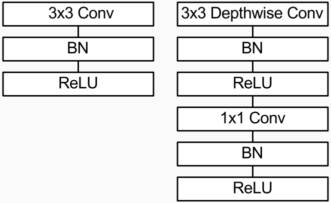
- **Depth:** 201 layers,
- **Number of parameters**: 20 million,
- **Data-augmentation**: crops and flips.
- **Batch-normalization**: BN, next ReLU and next Conv.
  Note that ReLU is used before Conv → pre-activation (He et al., 2016).
- **Optim**: SGD with mini-batch size 256 and 90 epochs + Nesterov + weight decay + learning rate schedule.

Same top-1 error rates as ResNet but with 3 times less parameters.
Won the CVPR 2017 best paper award.

Successful CNNs architectures

MobileNet (Google – Howard *et al.*, 2016-17)

Goal: to develop small and fast CNNs for mobile vision applications.



3x3 Conv

BN

ReLU

3x3 Depthwise Conv

BN

ReLU

1x1 Conv

BN

ReLU

Introduced concept: to factorize $n \times n$ conv. with M inputs and N outputs by:

1. **Depthwise convolution:** one filter per channel $n \times n$ $(\times M)$.
2. **Pointwise convolution:** 1×1 conv. to generate N new features from M.

Separable convolution

Successful CNNs architectures

MobileNet (Google – Howard *et al.*, 2016-17)

Other settings:

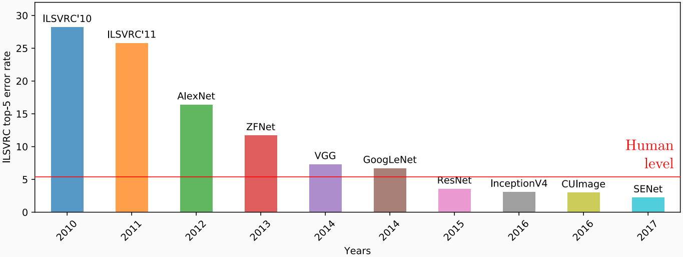
- **Depth:** 28 layers (depthwise + pointwise counted as 2)
- **Number of parameters**: 4.2 million (GoogleNet: 6.8M),
- **Data-augmentation**: less than other models,
- **Batch-normalization**: after each convolution,
- **Optimization**: SGD with no weight-decay.

Similar performance than GoogleNet but 2/3 less parameters.

Offers the possibility to reduce the width and resolution to trade accuracy and latency for specific devices.

Latest CNNs

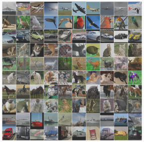
- **Inception v4** (Szegedy et al, 2016) – top-5 error: 3.08%
- **CUImage** (Zeng *et al.*, 2016) – top-5 error: 2.99% (winner ILSVRC 2016)
- **Xception** (Google – Chollet, 2016) – Similar to MobileNet
- **FractalNet** (Larson *et al.*, 2016) – Ultra-deep alternative to ResNet
- **SqueezeNet** (Iandola, *et al*, 2016) – as AlexNet but 50× fewer parameters
- **SENet** (Hu et al, 2017) – top-5 error: 2.23% (winner ILSVRC 2017)
- and many others. . .



Other ways to compare them?

Other standard datasets

CIFAR-10



STL-10

SVHN

- CIFAR-10 datasets
  - 60000 32x32 color images,
  - 10 classes, with 6000 images per class,
  - 50000 training images and 10000 testing images.
- CIFAR-100 datasets
  - Same but with 100 classes,
  - Each image as a fine and coarse label.
- STL-10
  - Similar as CIFAR-10 but images are 96x96.
- SVHN (Street View House Numbers)
  - Similar to MNIST but with 600,000 color digit images.

Evaluation on these datasets usually comes with other metrics.

Classification evaluation metrics

Other evaluation metrics

Vocabulary from detection theory

	Present	Not present
Detected	True positive	False positive
Not detected	False negative	True negative

$$\text{Precision} = \frac{\#TP}{\#TP + \#FP} = P(\text{present}|\text{detected})$$

$$\text{Recall} = \frac{\#TP}{\#TP + \#FN} = P(\text{detected}|\text{present})$$

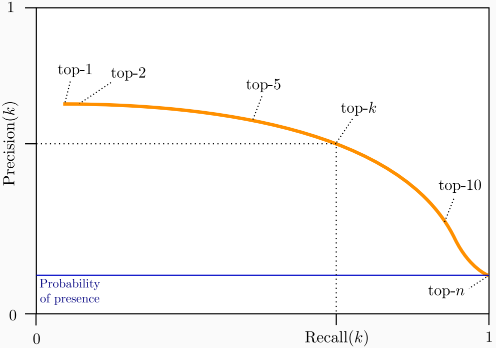
$$\text{Specificity} = \frac{\#TN}{\#TN + \#FP} = P(\text{not detected}|\text{not present})$$

$$\text{Accuracy} = \frac{\#TP + \#TN}{\text{Everything}} = P(\text{being correct})$$

From detection to classification

Make use of detection metrics for classification into n classes

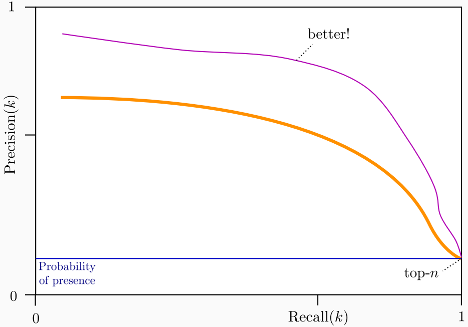
- Each class = one detection problem: is this type of object present or not?
- Use top-1 or top-5 ranking for recording error types: TP, FP, TN, FN,
- Or, look at the (mean) precision-recall curve for all top-k:



Precision-recall curve

Precision-recall curve:

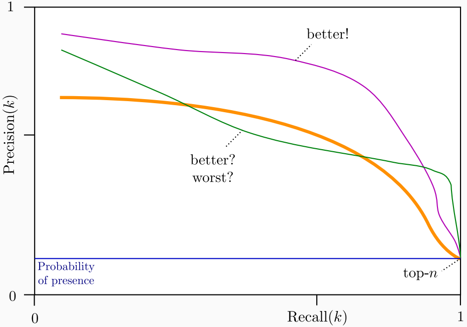
- A classifier is better if it's precision is higher for all recall.



Precision-recall curve

Precision-recall curve:

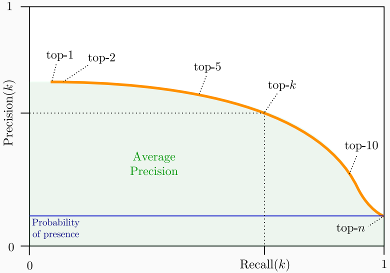
- A classifier is better if it's precision is higher for all recall.



- But what if they cross?

Mean Average Precision

- **Average precision** (AP) = Area under the curve (AuC)

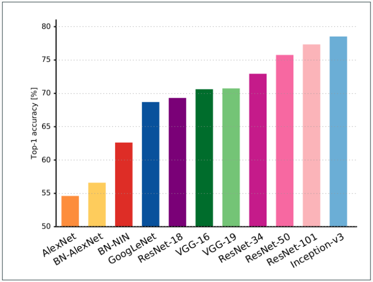


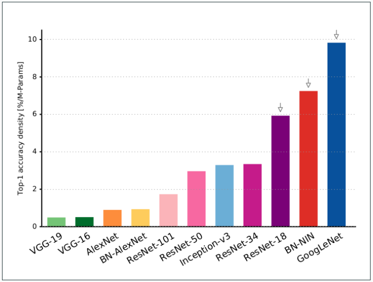
$$\text{AP} = \underbrace{\int_0^1 \text{Precision-at-recall}(r)\, \text{d}r}_{\text{area under the curve}} \approx \sum_{k=1}^{n} \text{Precision}(k)(\text{Recall}(k) - \text{Recall}(k-1))$$

- **mean Average Precision (mAP):** mean of AP over all classes.

Other evaluation metrics

- Average accuracy, or
- Average accuracy normalized per million of parameters:

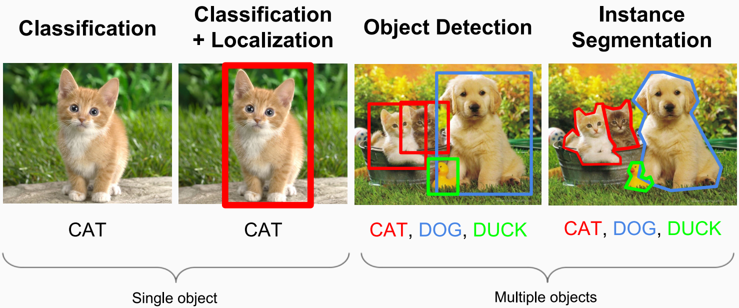




(Canziani et al., 2016)

Image classification and CNNs

What's next?



Classification

Classification + Localization

Object Detection

Instance Segmentation

CAT

CAT

CAT, DOG, DUCK

CAT, DOG, DUCK

Single object

Multiple objects

Questions?

Next class: Localization, detection and segmentation

Sources, images courtesy and acknowledgment

A. Horodniceanu

J. Johnson

A. Karpathy

Y. LeCun

V. Lepetit

F.-F. Li

L. Masuch

S. Seitz

V. Tong Ta

+ all referenced articles