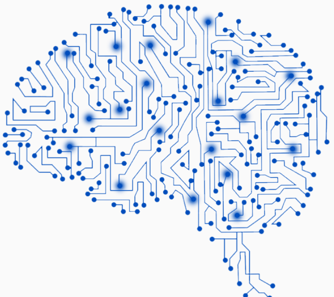
Master 2 Statistique & Data Science, Ingénierie Mathématique

Réseaux de neurones profonds pour l'apprentissage
Deep neural networks for machine learning

Course III – Introduction to Artificial Neural Networks: Deep neural networks and Convolutional Neural Networks (CNN)

Bruno Galerne

2024-2025



Credits

Most of the slides from **Charles Deledalle's** course "UCSD ECE285 Machine learning for image processing" (30 $\times$ 50 minutes course)

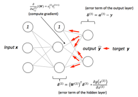
`www.charles-deledalle.fr/`

`https://www.charles-deledalle.fr/pages/teaching.php#learning`

Machine learning – Timeline

Timeline of (deep) learning

1958 Perceptron



(compute gradient)
(error term of the output layer)
Input x
output ŷ
target y
(error term of the hidden layer)

1974 Backpropagation

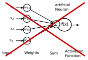
Convolution Neural Networks for Handwritten Recognition
1998

Google Brain Project on 16k Cores
2012

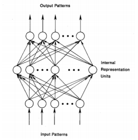
awkward silence (AI winter)

1969
Perceptrons book

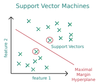
Perceptron criticized



~1980
Multilayer network



1995
SVM reigns



2006
Restricted Boltzmann Machine

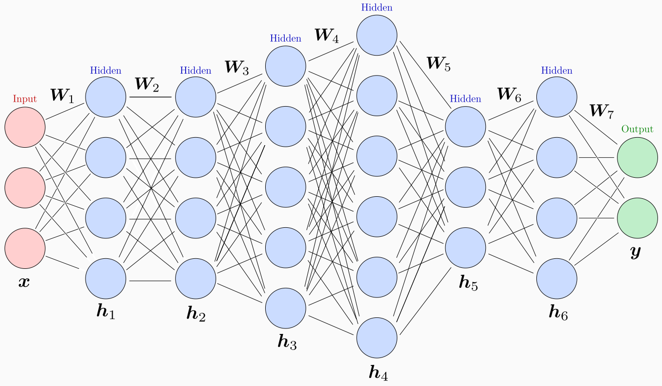
2012
AlexNet wins ImageNet
IMAGENET

(Source: Lucas Masuch & Vincent Lepetit)

Deep learning

(Source: Dan Page)

ANNs recap

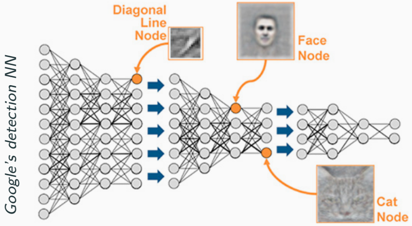


- Interconnections of neurons (summations and activations),
- Feedforward architecture: input $\rightarrow$ hidden layer(s) $\rightarrow$ output,
- Parameterized by a collection of weights $\boldsymbol{W}_k$ (and biases),
- Backprop: parameters learned from a collection of labeled data.

Deep learning

What is deep learning?

- Representation learning using artificial neural networks
  - → Learning good features automatically from raw data.
  - → Exceptionally effective at learning patterns.
- Learning representations of data with multiple levels of abstraction
  - → hierarchy of layers that mimic the neural networks of our brain,
  - → cascade of non-linear transforms.

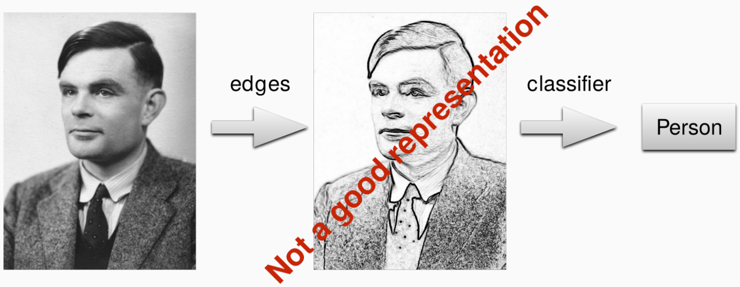


- If you provide the system with tons of information, it begins to understand it and responds in useful ways.

(Source: Caner Hazırbaş & Lucas Masuch)

Deep learning

How to teach a machine?

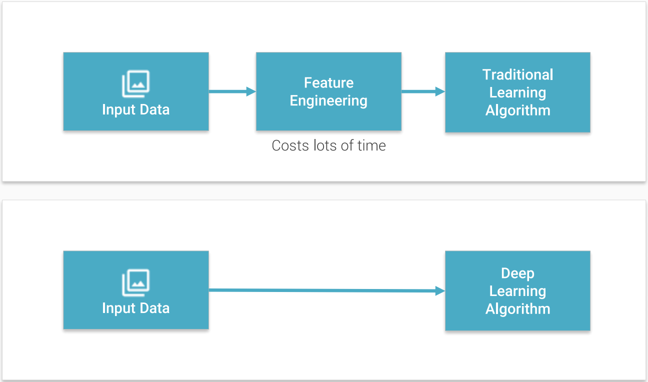


(or any other **hand-crafted** features)

Good representations are often very complex to define.

(Source: Caner Hazırbaş)

Deep learning: No more feature engineering



Learn the latent factors/features behind the data generation

(Source: Lucas Masuch)

Deep learning

Inspired by the Brain

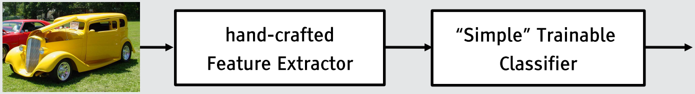
- The first hierarchy of neurons that receives information in the visual cortex are sensitive to specific edges while brain regions further down the visual pipeline are sensitive to more complex structures such as faces.
- Our brain has lots of neurons connected together and the strength of the connections between neurons represents long term knowledge.
- One learning algorithm hypothesis: all significant mental algorithms are learned except for the learning and reward machinery itself.

(Source: Lucas Masuch)

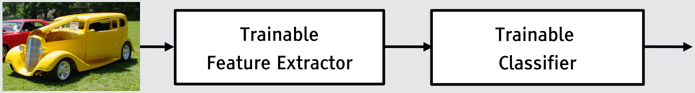
Deep learning

Deep learning: No more feature engineering

- The traditional model of pattern recognition (since the late 50's)
  - Fixed/engineered features (or fixed kernel) + trainable classifier



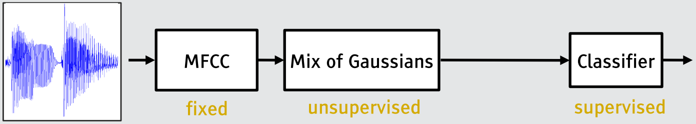
- End-to-end learning / Feature learning / Deep learning
  - Trainable features (or kernel) + trainable classifier



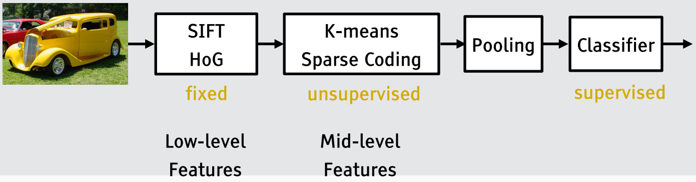
(Source: Yann LeCun & Marc'Aurelio Ranzato)

Deep learning: No more feature engineering

- Non-deep learning architecture for pattern recognition
  - Speech recognition: early 90's – 2011

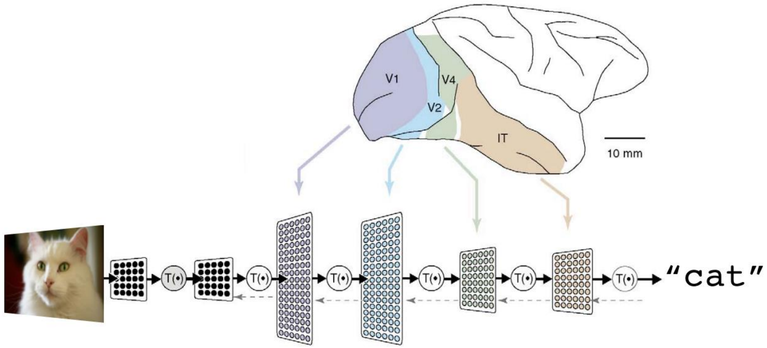


  - Object Recognition: 2006 – 2012



(Source: Yann LeCun & Marc'Aurelio Ranzato)

Deep learning – Basic architecture



A deep neural network consists of a hierarchy of layers, whereby each layer transforms the input data into more abstract representations (e.g. edge -> nose -> face). The output layer combines those features to make predictions.

(Source: Lucas Masuch)

Trainable feature hierarchy

- Hierarchy of representations with increasing levels of abstraction.
- Each stage is a kind of trainable feature transform.

Image recognition

- Pixel $\rightarrow$ edge $\rightarrow$ texton $\rightarrow$ motif $\rightarrow$ part $\rightarrow$ object

Text

- Character $\rightarrow$ word $\rightarrow$ word group $\rightarrow$ clause $\rightarrow$ sentence $\rightarrow$ story

Speech

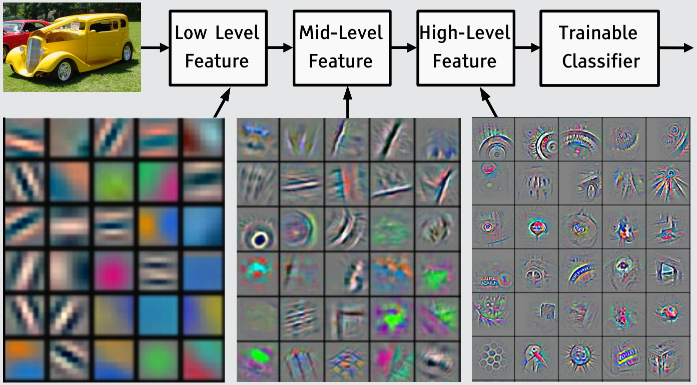
- Sample $\rightarrow$ spectral band $\rightarrow$ sound $\rightarrow$ ... $\rightarrow$ phone $\rightarrow$ phoneme $\rightarrow$ word

Deep Learning addresses the problem of learning hierarchical representations.

(Source: Yann LeCun & Marc'Aurelio Ranzato)

Deep learning – Feature hierarchy

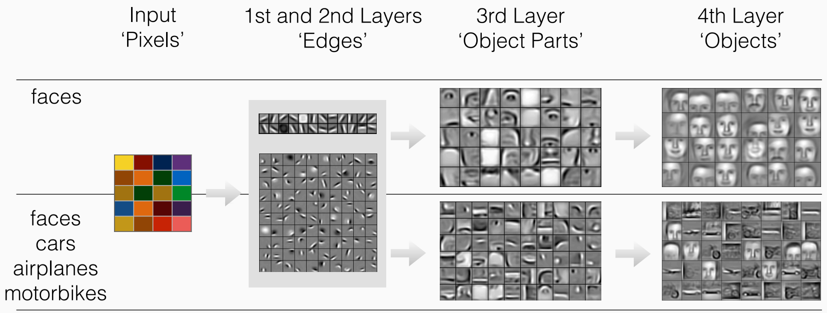
- It's deep if it has more than one stage of non-linear feature transformation



Feature visualization of convolutional net trained on ImageNet from (Zeiler & Fergus, 2013)

(Source: Yann LeCun & Marc'Aurelio Ranzato)

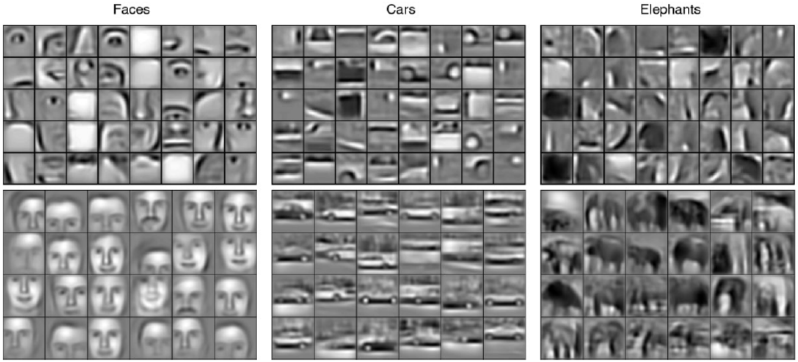
Deep learning – Feature hierarchy



Each layer progressively extracts higher level features of the input until the final layer essentially makes a decision about what the input shows. The more layers the network has, the higher level features it will learn.

(Source: Andrew Ng & Lucas Masuch & Caner Hazırbaş)

Deep learning – Feature hierarchy



- Mid- to high-level features are specific to the given task.
- Low-level representations contain less specific features.
  (can be shared for many different applications)

Deep learning – Training

Today's trend: make it deeper and deeper

• 2012: 8 layers	(AlexNet – Krizhevsky *et al.*, 2012)
• 2014: 19 layers	(VGG Net – Simonyan & Zisserman, 2014)
• 2014: 22 layers	(GoogLeNet – Szegedy *et al.*, 2014)
• 2015: 152 layers	(ResNet – He *et al.*, 2015)
• 2016: 201 layers	(DenseNet – Huang *et al.*, 2017)

But remember, with back-propagation:

- We got stuck at local optima or saddle points
- The learning time does not scale well
    - it is very slow for deep networks and can be unstable.

How did networks get so deep? First, why does backprop fail?

Deep learning – Gradient vanishing problems

Back-propagation and gradient vanishing problems

Update: $\boldsymbol{W}_k = \boldsymbol{W}_k - \gamma \nabla_{\boldsymbol{W}_k} E$ with $\nabla_{\boldsymbol{W}_k} E = \boldsymbol{\delta}_k \boldsymbol{h}_{k-1}^T$

where $\boldsymbol{\delta}_k = \nabla_{\boldsymbol{a}_k} E = \nabla_{\boldsymbol{h}_k} E \odot g_k'(\boldsymbol{a}_k)$.

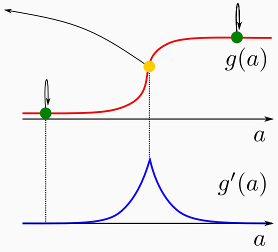
- With deep networks, the gradient vanishes quickly.
- Unfortunately, this arises even though we are far from a solution.
- The updates become insignificant, which leads to slow training rates.
- This strongly depends on the shape of $g'(a)$.

- The gradient may also explode leading to instabilities:
  $\rightarrow$ gradient exploding problem.

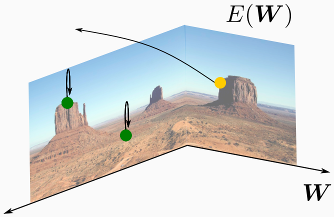
Deep learning – Gradient vanishing problem

As the network gets deeper, the landscape of E becomes:

- very hilly $\rightarrow$ lots of stationary points,
- with large plateaus $\rightarrow$ gradient vanishing problem,
- and delimited by cliffs $\rightarrow$ gradient exploding problem.



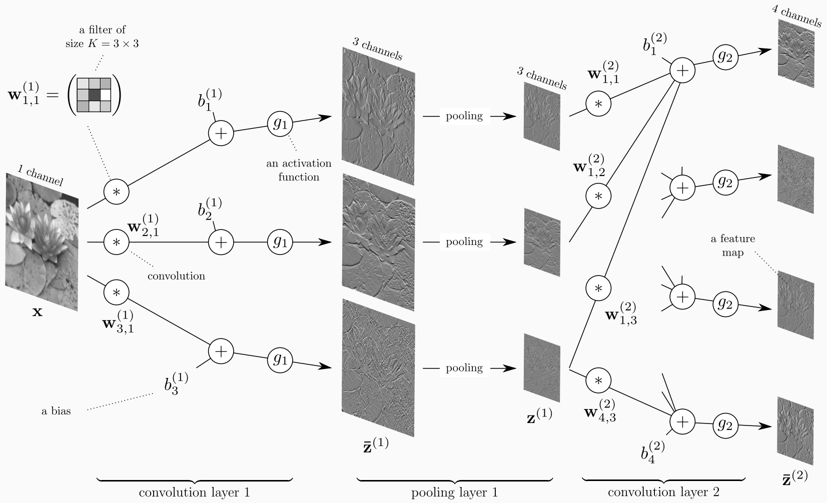
Activation function



Cost landscape

So, what has changed? (see later for recipes...)

CNN for image processing



Convolution: One chanel

For an image $x = x(i,j)$ having **one chanel**, the convolution with a kernel κ of size $[-s,s]\times[-s,s]$

- the **convolution** is:

$$x * \kappa(i,j) = \sum_{(k,\ell)\in[-s,s]\times[-s,s]} \kappa(k,\ell)x(i-k,j-\ell)$$

  $x * \kappa =$ local average of x with the weight of κ in **opposite position**

- the **cross-correlation** is:

$$x \otimes \kappa(i,j) = \sum_{(k,\ell)\in[-s,s]\times[-s,s]} \kappa(k,\ell)x(i+k,j+\ell).$$

  $x \otimes \kappa =$ local average of x with the weight of κ in **same position**

- Need to deal with boundary issues for pixels at the border: zero-padding (0 if outside border), valid positions only (do not compute at border, smaller output images), mirror symmetry at border...

Convolution

- Images have generally several chanels (eg RGB).
- By computing several convolutions we can stack the result as a single multi-channel output.

In machine learning

"convolution"

means

"cross-correlation + bias"

- Recall that "linear" layers are affine map $x \mapsto Wx + b$, so convolution layers are a specific case where $x \mapsto Wx$ is a cross-correlation.

Convolution layer

Convolution layer with c_{in} input chanels and c_{out} output chanels:

- Input image $\boldsymbol{x}$ with c_{in} **chanels**: values $\boldsymbol{x}(i,j) \in \mathbb{R}^{c_{\text{in}}}$
- Output image $\boldsymbol{y}$ with c_{out} **chanels**.
- Kernel: $\boldsymbol{\kappa}$ such that for all $(k,\ell) \in [-s,s] \times [-s,s]$

$$\boldsymbol{\kappa}(k,\ell) \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}}}, \quad \text{is a } c_{\text{out}} \times c_{\text{in}} \text{ matrix}$$

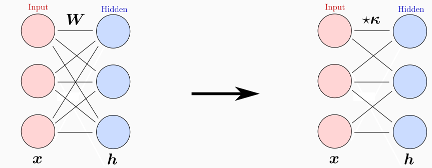
- Bias: $\boldsymbol{b} \in \mathbb{R}^{c_{\text{out}}}$

$$\begin{aligned} \boldsymbol{y}(i,j) &= \mathrm{Conv}(\boldsymbol{x}; \boldsymbol{\kappa}, \boldsymbol{b})(i,j) \\ &= \left[\sum_{(k,\ell) \in [-s,s] \times [-s,s]} \boldsymbol{\kappa}(k,\ell) \boldsymbol{x}(i+k, j+\ell) \right] + \boldsymbol{b} \in \mathbb{R}^{c_{\text{out}}} \end{aligned}$$

- Number of parameters: $(2s+1)^2 \times c_{\text{in}} \times c_{\text{out}}$ for $\boldsymbol{\kappa}$ and c_{out} for $\boldsymbol{b}$

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

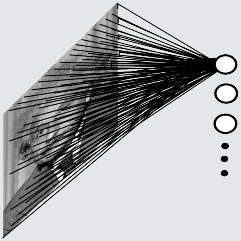
4. Redundancy: more units in a hidden layer than inputs,
5. Sparsity: units should not all fire for the same stimulus.

All take inspiration from the visual cortex.

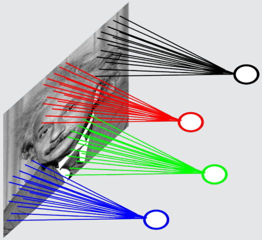
Convolutional neural networks

Local receptive fields $\rightarrow$ 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

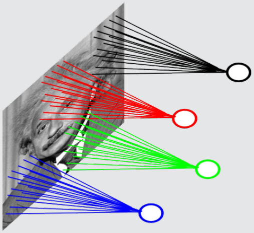
For a 200×200 image and 40,000 hidden units

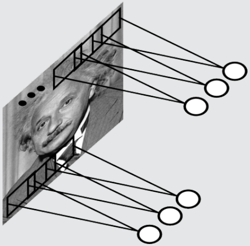
- Fully connected: 1.6 billion parameters,
- Locally connected (10×10 fields): 4 million parameters.

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

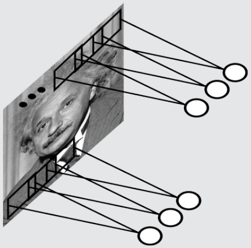
For a 200×200 image and 40,000 hidden units

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

Convolutional neural networks

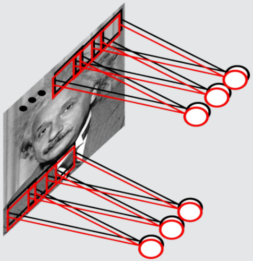
Specialized cells → Filter bank

- 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

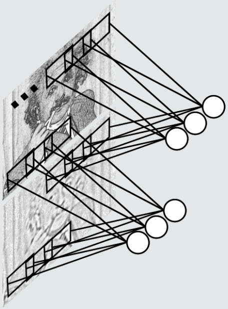
(1 input → 2 feature maps)

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

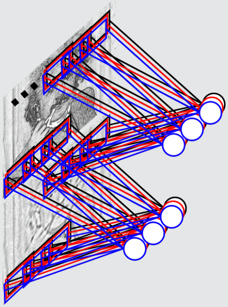
Convolutional neural networks

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)



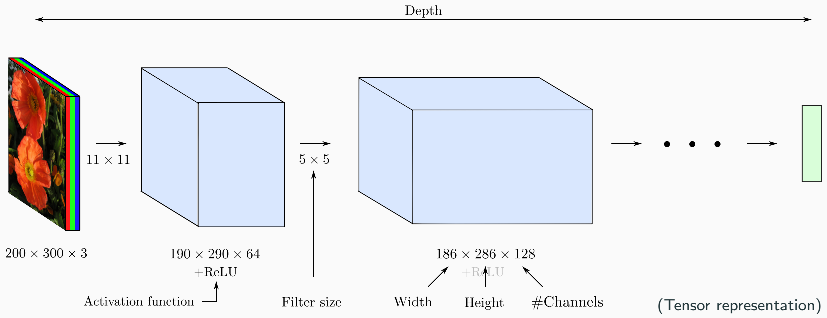
Multi-input filter bank
(2 inputs → 3 feature maps)

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

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

Convolutional neural networks

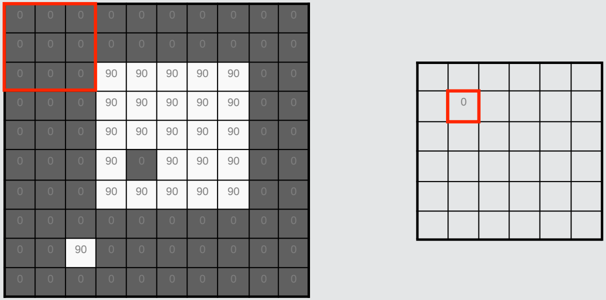
Overcomplete $\rightarrow$ increase the number of channels



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

Controlling the number of simple cells $\rightarrow$ Stride

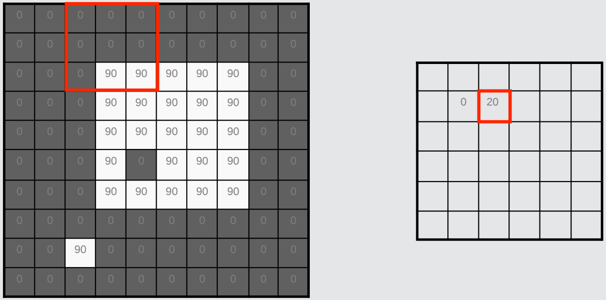
3×3 boxcar strided convolution with stride $s = 2$



- 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* $\rightarrow$ 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

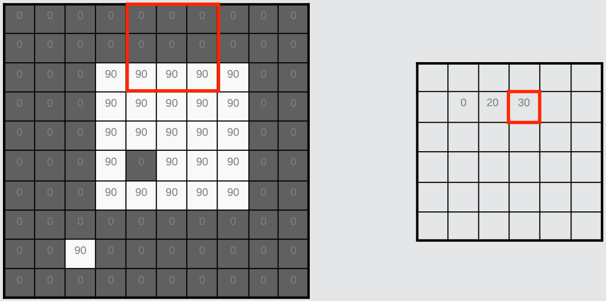
3×3 boxcar strided convolution with stride $s = 2$



- 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 $\rightarrow$ Stride

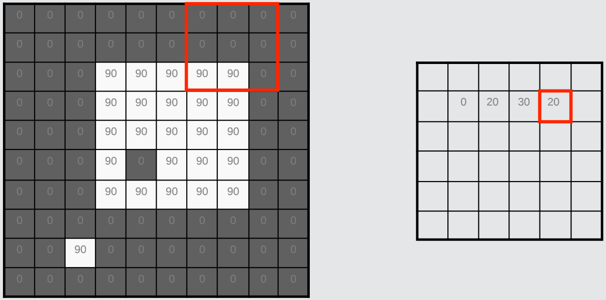
3×3 boxcar strided convolution with stride $s = 2$



- 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* $\rightarrow$ 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 $\rightarrow$ Stride

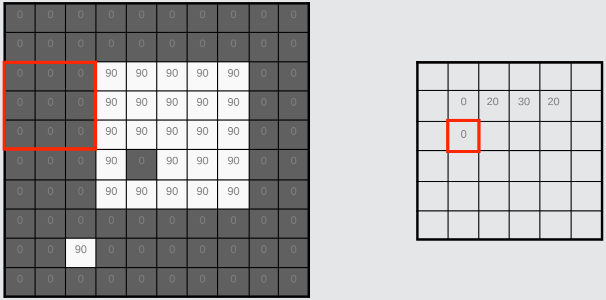
3×3 boxcar strided convolution with stride $s = 2$



- 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* $\rightarrow$ 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 $\rightarrow$ Stride

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



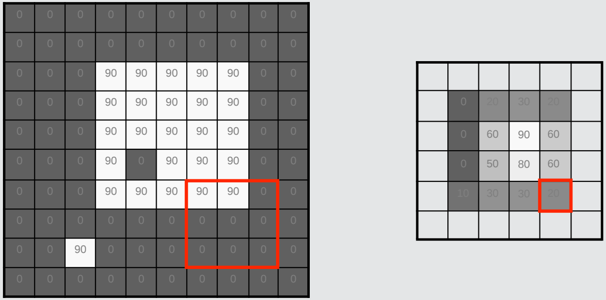
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	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* $\rightarrow$ 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 $\rightarrow$ Stride

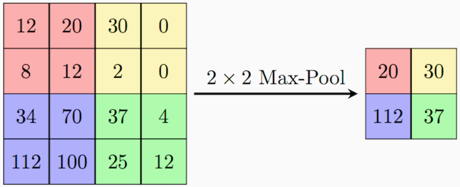
3×3 boxcar strided convolution with stride $s = 2$



- 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* $\rightarrow$ 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.

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

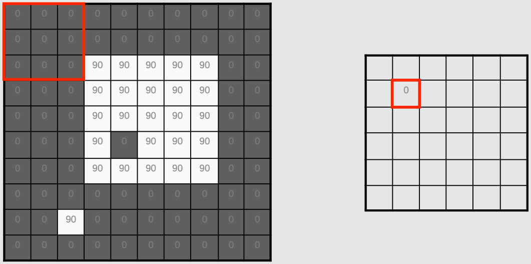


  - average-pooling: take the average

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 $s = 2$:

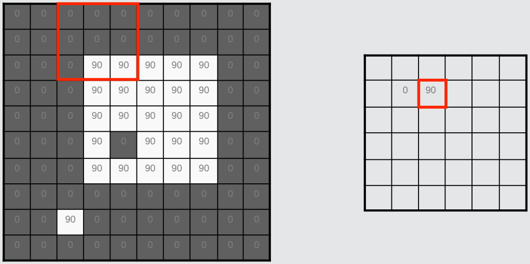


- When $z = s$: standard pooling (non-overlapping),

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 $s = 2$:

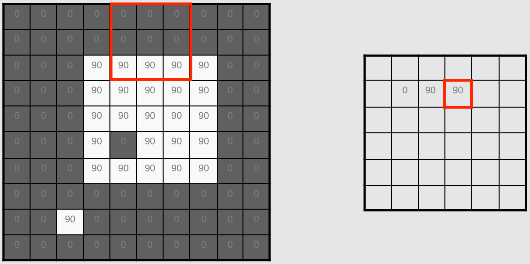


- When $z = s$: standard pooling (non-overlapping),

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 $s = 2$:

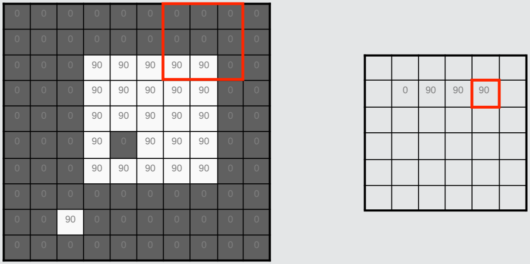


- When $z = s$: standard pooling (non-overlapping),

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 $s = 2$:

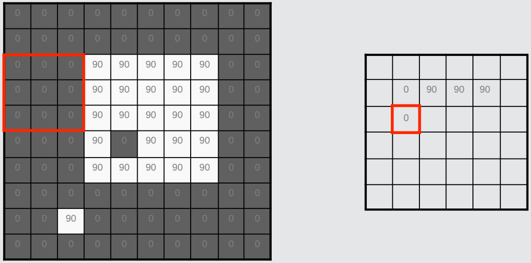


- When $z = s$: standard pooling (non-overlapping),

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 $s = 2$:

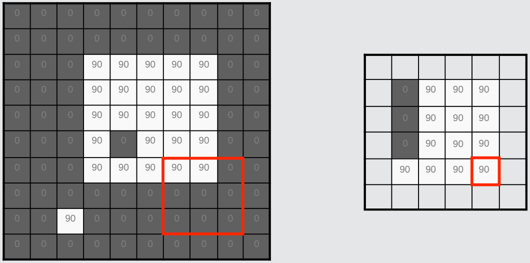


- When $z = s$: standard pooling (non-overlapping),

Pooling layer

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

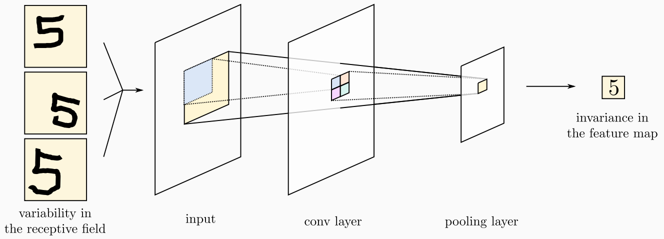
- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:



- When $z = s$: standard pooling (non-overlapping),

Convolutional neural networks

Pooling layer



variability in the receptive field
input
conv layer
pooling layer
5
invariance in the feature map

- 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

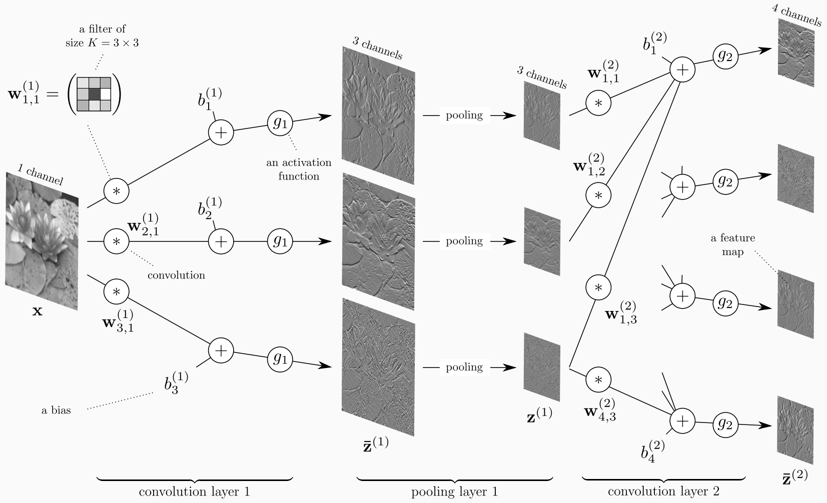
The filter weights $\boldsymbol{\kappa}$ and the bias b are learned by backprop.

Setting up a pooling layer requires choosing

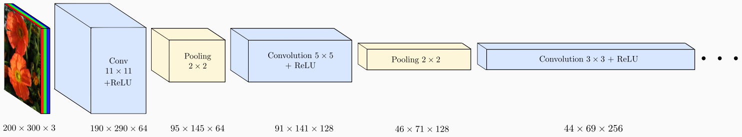
• Pooling size:	$z \times z$
• Aggregation rule:	max-pooling, average-pooling, . . .
• Stride:	s
• Padding:	p

No free parameters to be learned here.

All concepts together

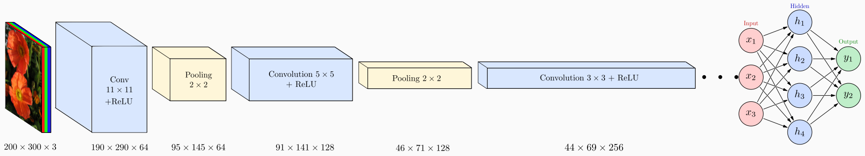


All concepts together with tensor representation



CNN: Alternate:
Conv + ReLU + pooling

All concepts together with tensor representation


Conv
11×11
+ReLU
Pooling
2×2
Convolution 5×5
+ ReLU
Pooling 2×2
Convolution 3×3 + ReLU
Input
Hidden
Output
x_1
x_2
x_3
h_1
h_2
h_3
h_4
y_1
y_2
$200 \times 300 \times 3$
$190 \times 290 \times 64$
$95 \times 145 \times 64$
$91 \times 141 \times 128$
$46 \times 71 \times 128$
$44 \times 69 \times 256$

CNN: Alternate:

Conv + ReLU + pooling

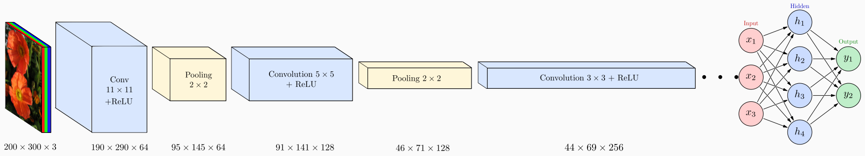
End of network:

Plug a standard neural network:

Fully connected hidden layers

(linear) + ReLU

All concepts together with tensor representation


Conv
11×11
+ReLU
Pooling
2×2
Convolution 5×5
+ ReLU
Pooling 2×2
Convolution 3×3 + ReLU
Input
Hidden
Output
x_1
x_2
x_3
h_1
h_2
h_3
h_4
y_1
y_2
$200 \times 300 \times 3$
$190 \times 290 \times 64$
$95 \times 145 \times 64$
$91 \times 141 \times 128$
$46 \times 71 \times 128$
$44 \times 69 \times 256$

CNN: Alternate:
Conv + ReLU + pooling

End of network:
Plug a standard neural network:
Fully connected hidden layers (linear) + ReLU

Full network:

- **CNN:** Extract features specific to spatial data
- **Fully connected part:** Use CNN features for specific regression/classification task
- **Training:** Learn regression/classification and feature extraction **jointly**

Questions?

Slides from Charles Deledalle

Sources, images courtesy and acknowledgment

K. Chatfield	A. Horodniceanu	L. Masuch
P. Gallinari,	Y. LeCun	A. Ng
C. Hazırbaș	V. Lepetit	M. Ranzato

Practical session – Convolutional neural networks

Go through the PyTorch tutorial:

“Deep Learning with PyTorch: A 60 Minute Blitz”

`https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`

Each part is a notebook with a “Run in Google Colab” button.

After that:

1. Reproduce the results of the previous session using PyTorch: Define and train a neural network for multiclass logistic regression (linear+cross-entropy loss) on the toy 2D datasets.
2. Add a first hidden layer with ReLU and train this new model.
3. Observe that the performances are better than with a random hidden layer.