Deep Learning & Applied AI

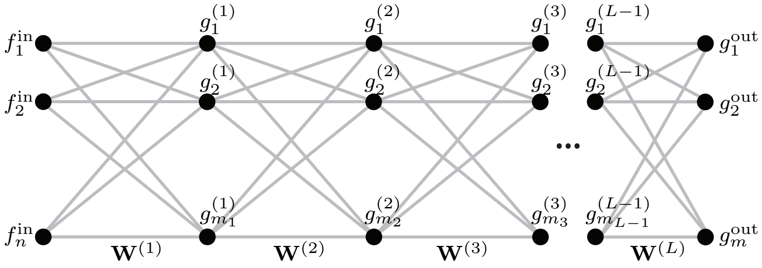
Convolutional neural networks

Emanuele Rodolà
rodola@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

2nd semester a.y. 2022/2023 · April 5, 2023

Neural network (NN)



Deep neural network consisting of L layers

Net output $\mathbf{g}^{\text{out}} = \sigma\left(\ldots \mathbf{W}^{(2)}\sigma\left(\mathbf{W}^{(1)}\mathbf{f}^{\text{in}}\right)\right)$

Activation, e.g. $\sigma(x) = \max\{x, 0\}$ rectified linear unit (ReLU)

Parameters weights of all layers $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}$ (including biases)

The need for priors

Deep feed-forward networks are provably universal.

However:

The need for priors

Deep feed-forward networks are provably universal.

However:

- We can make them arbitrarily complex.

The need for priors

Deep feed-forward networks are provably universal.

However:

- We can make them arbitrarily complex.
- The number of parameters can be huge.

The need for priors

Deep feed-forward networks are provably universal.

However:

- We can make them arbitrarily complex.
- The number of parameters can be huge.
- Very difficult to optimize.

The need for priors

Deep feed-forward networks are provably universal.

However:

- We can make them arbitrarily complex.
- The number of parameters can be huge.
- Very difficult to optimize.
- Very difficult to achieve generalization.

The need for priors

Deep feed-forward networks are provably universal.

However:

- We can make them arbitrarily complex.
- The number of parameters can be huge.
- Very difficult to optimize.
- Very difficult to achieve generalization.

We need additional priors as a (partial) remedy to the above.

The need for priors

Deep feed-forward networks are provably universal.

However:

- We can make them arbitrarily complex.
- The number of parameters can be huge.
- Very difficult to optimize.
- Very difficult to achieve generalization.

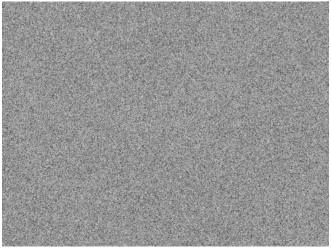
We need additional priors as a (partial) remedy to the above.

Look for "universal" priors that are task-independent to some extent.

Task-independent priors must come with the data.

Structure as a strong prior

Key insight: Data often carries structural priors in terms of repeating patterns, compositionality, locality, ...

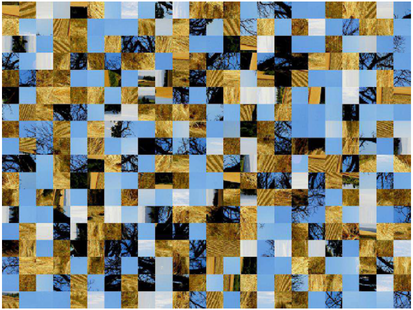


Structure as a strong prior

Key insight: Data often carries structural priors in terms of repeating patterns, compositionality, locality, ...

Structure as a strong prior

Key insight: Data often carries structural priors in terms of repeating patterns, compositionality, locality, ...



Questa immagine è stata riordinata. Nel paper qui sotto hanno sviluppato una rete neurale in grado di ricostruire l'immagine originale (senza conoscerla) tramite il riordinamento dei pezzi. Questo viene fatto in base a quanto un pezzo si "sposa" bene con quelli adiacenti.
Quindi questa "smoothness" è il prior.
Il risultato...

Son et al, "Solving Small-piece Jigsaw Puzzles by Growing Consensus", CVPR 2016

Structure as a strong prior

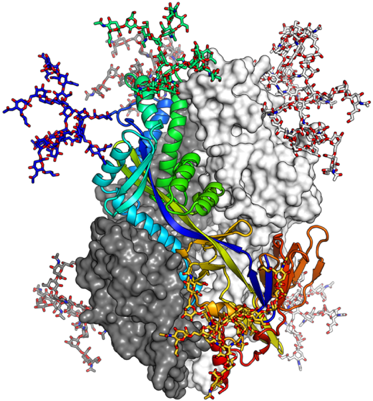
Key insight: Data often carries structural priors in terms of repeating patterns, compositionality, locality, ...

... è perfetto

Son et al, "Solving Small-piece Jigsaw Puzzles by Growing Consensus", CVPR 2016

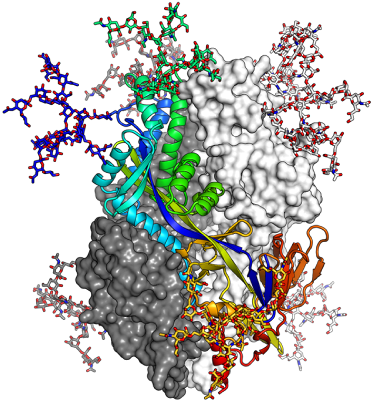
Structure as a strong prior

Key insight: Data often carries structural priors in terms of repeating patterns, compositionality, locality, ...



Structure as a strong prior

Key insight: Data often carries structural priors in terms of repeating patterns, compositionality, locality, . . .

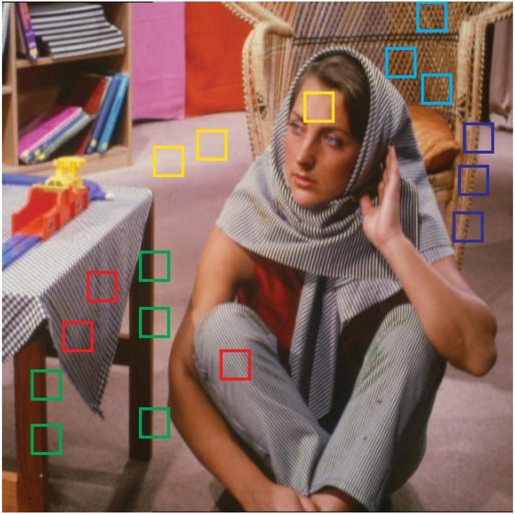


Take advantage of the structure of the data.

Self-similarity

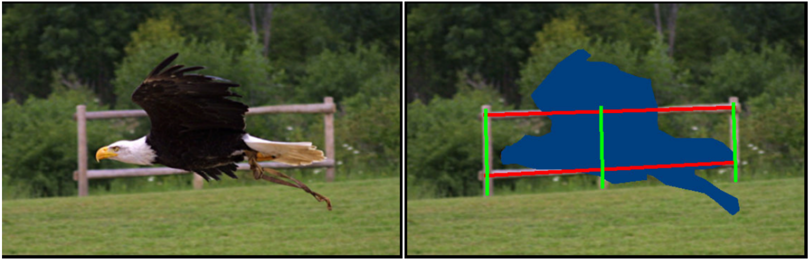
le feature sono locali (si possono trovare analizzando piccole parti dell'immagine)

Data tends to be self-similar across the domain:



Self-similarity

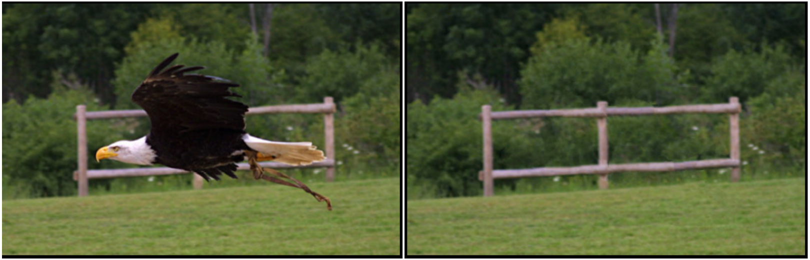
Data tends to be self-similar across the domain:



Barnes et al, "PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing", TOG 2009

Self-similarity

Data tends to be self-similar across the domain:



Barnes et al, "PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing", TOG 2009

Translation invariance

Translations do not change the image content.

Translation invariance

Translations do not change the image content.

Define the (linear!) translation operator $\mathcal{T}$ along vector $v \in \mathbb{R}^2$ as:

$$\mathcal{T}_v f(x) = f(x - v)$$

Translation invariance

Translations do not change the image content.

Define the (linear!) translation operator $\mathcal{T}$ along vector $v \in \mathbb{R}^2$ as:

T si applica a tutti i dati nel campo (in questo caso i pixel dell'immagine, ovvero le x)

$$\mathcal{T}_v f(x) = f(x - v)$$

in pratica, i pixel non cambiano di posizione, ma cambiano i loro valori (il colore)

Therefore, it is desirable to enforce translation invariance:

$$\mathcal{C}(\mathcal{T}_v f) = \mathcal{C}(f) \quad \forall f, \mathcal{T}_v$$

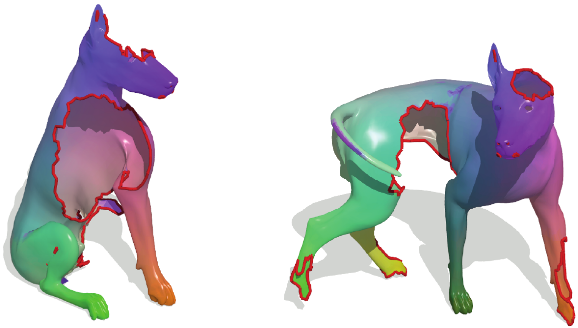
where $\mathcal{C}$ is a classification functional.

Deformation invariance

per altri tipi di invarianze ed equivarianze, guarda l'allegato

Other types of invariance are possible.

Invariance to partiality and isometric deformations:

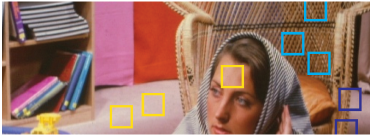


In many cases, invariance can be directly injected into the network.

Today we concentrate on translation invariance.

Hierarchy and compositionality

Translation invariance is desirable across multiple scales:



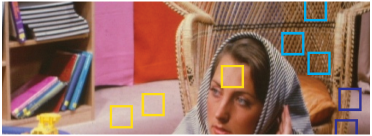
We expect local features to be invariant to their location in the image:

$$z(\mathcal{T}_v p) = z(p) \quad \forall p, \mathcal{T}_v$$

where p are image patches of variable size.

Hierarchy and compositionality

Translation invariance is desirable across multiple scales:



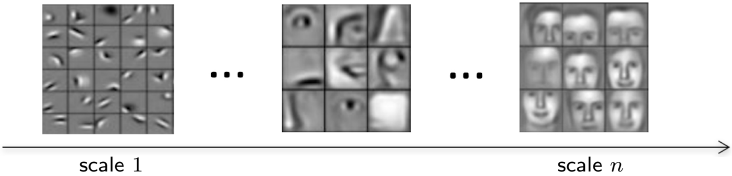
vogliamo poter identificare un pattern singolo indipendentemente dalla sua posizione

We expect local features to be invariant to their location in the image:

z è una feature intermedia nella CNN

$$z(\mathcal{T}_v p) = z(p) \quad \forall p, \mathcal{T}_v$$

where p are image patches of variable size.



Data is often composed of hierarchical, local, shift-invariant patterns.

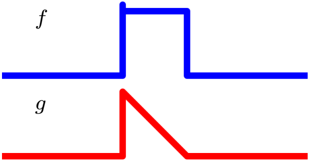
Data is often composed of hierarchical, local, shift-invariant patterns.

CNNs directly exploit this fact as a prior.

Convolution

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$(f \star g)(x) = \int_{-\pi}^{\pi} f(t)g(x-t)dt$$



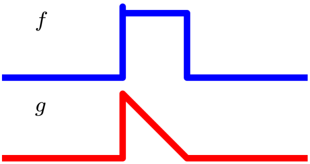
d'Alembert 1754

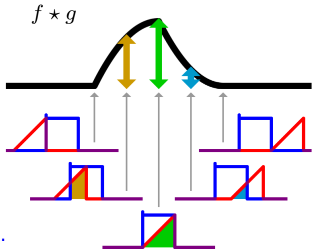
Convolution

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$(f \star g)(x) = \int_{-\pi}^{\pi} f(t)g(x-t)dt$$

la convoluzione continua
è quindi l'integrale del prodotto
di una funzione shiftata
(per un certo *t*) e un'altra funzione





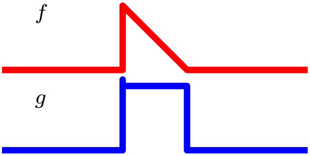
La convoluzione di due funzioni valuta quanto
una funzione si sovrappone all'altra man mano che
una delle funzioni viene traslata lungo l'asse delle x.
Il risultato della convoluzione continua è la funzione espressa
dalla linea nera che ha il valore massimo quando le due funzioni
sono perfettamente sovrapposte

d'Alembert 1754

Convolution

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$(f \star g)(x) = \int_{-\pi}^{\pi} f(t)g(x-t)dt$$

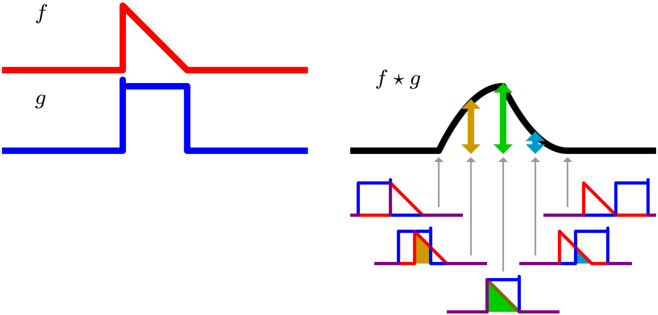


d'Alembert 1754

Convolution

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$(f \star g)(x) = \int_{-\pi}^{\pi} f(t) g(x - t) dt$$

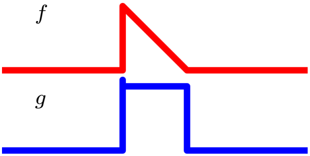


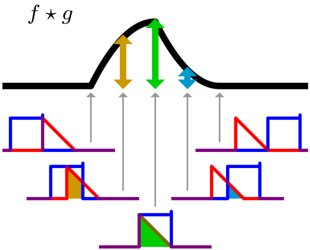
d'Alembert 1754

Convolution

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$\underbrace{(f \star g)(x)}_{\text{feature map}} = \int_{-\pi}^{\pi} f(t)\, \underbrace{g(x-t)}_{\text{filter kernel}}\, dt$$





cambiando l'ordine delle funzioni,
il risultato non cambia. La convoluzione
è quindi una operazione commutativa

d'Alembert 1754

Convolution: Commutativity

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$\underbrace{(f \star g)(x)}_{\text{feature map}} = \int_{-\pi}^{\pi} f(t) \underbrace{g(x - t)}_{\text{filter}} dt$$

Convolution is commutative:

$$(f \star g)(x) = \int_{-\pi}^{\pi} f(t)g(x - t)dt$$

Convolution: Commutativity

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$\underbrace{(f \star g)(x)}_{\text{feature map}} = \int_{-\pi}^{\pi} f(t)\, \underbrace{g(x-t)}_{\text{filter}}\, dt$$

Convolution is commutative:

$$(f \star g)(x) = \int_{-\pi}^{\pi} f(t) g(x-t) dt \overset{\overset{z:=x-t}{\downarrow}}{=} \int_{-\pi}^{\pi} f(x-z) g(z) dz$$

Convolution: Commutativity

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$\underbrace{(f \star g)(x)}_{\text{feature map}} = \int_{-\pi}^{\pi} f(t) \underbrace{g(x-t)}_{\text{filter}} dt$$

Convolution is commutative:

$$(f \star g)(x) = \int_{-\pi}^{\pi} f(t) g(x-t) dt \overset{\substack{z:=x-t \\ \downarrow}}{=} \int_{-\pi}^{\pi} f(x-z) g(z) dz = (g \star f)(x)$$

Convolution: Commutativity

Given two functions $f, g : [-\pi, \pi] \to \mathbb{R}$ their convolution is a function:

$$\underbrace{(f \star g)(x)}_{\text{feature map}} = \int_{-\pi}^{\pi} f(t) \underbrace{g(x-t)}_{\text{filter}} dt$$

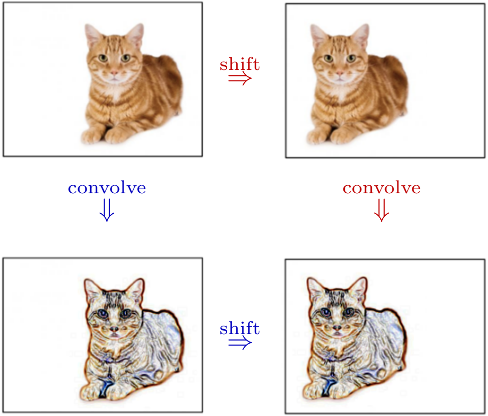
Convolution is commutative:

$$(f \star g)(x) = \int_{-\pi}^{\pi} f(t) g(x-t) dt \overset{\overset{z:=x-t}{\downarrow}}{=} \int_{-\pi}^{\pi} f(x-z) g(z) dz = (g \star f)(x)$$

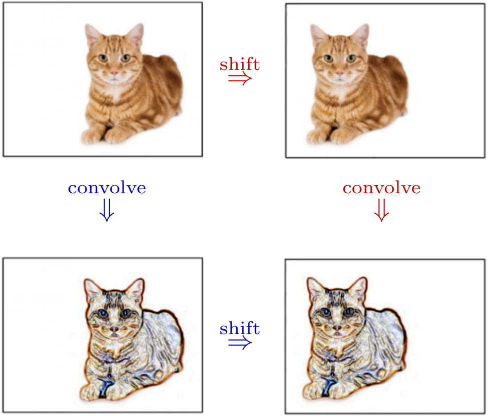
Further, convolution is shift-equivariant (or translation-equivariant):

$$f(x - x_0) \star g(x) = (f \star g)(x - x_0)$$

Convolution: Shift-equivariance



Convolution: Shift-equivariance



In fact, equivariance is a defining property of convolutions.

(any linear operator that is shift-equivariant, is a convolution)

Convolution: Linearity

We can see convolution as the application of a linear operator $\mathcal{G}$:

$$\mathcal{G}f(x) = (f \star g)(x) = \int_{-\pi}^{\pi} f(t)\, \underbrace{g(x-t)}_{\text{filter}}\, dt$$

Convolution: Linearity

We can see convolution as the application of a linear operator $\mathcal{G}$:

$$\mathcal{G}f(x) = (f \star g)(x) = \int_{-\pi}^{\pi} f(t)\, \underbrace{g(x-t)}_{\text{filter}}\, dt$$

It is easy to show that $\mathcal{G}$ is linear:

$$\mathcal{G}(\alpha f(x)) = \int_{-\pi}^{\pi} \alpha f(t) g(x-t) dt$$

Convolution: Linearity

We can see convolution as the application of a linear operator $\mathcal{G}$:

$$\mathcal{G}f(x) = (f \star g)(x) = \int_{-\pi}^{\pi} f(t) \underbrace{g(x-t)}_{\text{filter}} dt$$

It is easy to show that $\mathcal{G}$ is linear:

$$\mathcal{G}(\alpha f(x)) = \alpha \int_{-\pi}^{\pi} f(t) g(x-t) dt$$

Convolution: Linearity

We can see convolution as the application of a linear operator $\mathcal{G}$:

$$\mathcal{G}f(x) = (f \star g)(x) = \int_{-\pi}^{\pi} f(t) \underbrace{g(x-t)}_{\text{filter}} dt$$

It is easy to show that $\mathcal{G}$ is linear:

$$\mathcal{G}(\alpha f(x)) = \alpha \int_{-\pi}^{\pi} f(t)g(x-t)dt = \alpha\, \mathcal{G}f(x)$$

Convolution: Linearity

We can see convolution as the application of a linear operator $\mathcal{G}$:

$$\mathcal{G}f(x) = (f \star g)(x) = \int_{-\pi}^{\pi} f(t)\,\underbrace{g(x-t)}_{\text{filter}}\,dt$$

It is easy to show that $\mathcal{G}$ is linear:

$$\mathcal{G}(\alpha f(x)) = \alpha \int_{-\pi}^{\pi} f(t)g(x-t)dt = \alpha\,\mathcal{G}f(x)$$

$$\mathcal{G}(f+h)(x) = \int_{-\pi}^{\pi} (f+h)(t)g(x-t)dt$$

Convolution: Linearity

We can see convolution as the application of a linear operator $\mathcal{G}$:

$$\mathcal{G}f(x) = (f \star g)(x) = \int_{-\pi}^{\pi} f(t)\,\underbrace{g(x-t)}_{\text{filter}}\,dt$$

It is easy to show that $\mathcal{G}$ is linear:

$$\mathcal{G}(\alpha f(x)) = \alpha \int_{-\pi}^{\pi} f(t)g(x-t)dt = \alpha\,\mathcal{G}f(x)$$

$$\mathcal{G}(f+h)(x) = \int_{-\pi}^{\pi} (f(t)+h(t))g(x-t)dt$$

Convolution: Linearity

We can see convolution as the application of a linear operator $\mathcal{G}$:

$$\mathcal{G}f(x) = (f \star g)(x) = \int_{-\pi}^{\pi} f(t)\,\underbrace{g(x-t)}_{\text{filter}}\,dt$$

It is easy to show that $\mathcal{G}$ is linear:

$$\mathcal{G}(\alpha f(x)) = \alpha \int_{-\pi}^{\pi} f(t)g(x-t)dt = \alpha\,\mathcal{G}f(x)$$

$$\mathcal{G}(f+h)(x) = \int_{-\pi}^{\pi} f(t)g(x-t)dt + \int_{-\pi}^{\pi} h(t)g(x-t)dt$$

Convolution: Linearity

We can see convolution as the application of a linear operator $\mathcal{G}$:

$$\mathcal{G}f(x) = (f \star g)(x) = \int_{-\pi}^{\pi} f(t)\,\underbrace{g(x-t)}_{\text{filter}}\,dt$$

It is easy to show that $\mathcal{G}$ is linear:

$$\mathcal{G}(\alpha f(x)) = \alpha \int_{-\pi}^{\pi} f(t)g(x-t)dt = \alpha\,\mathcal{G}f(x)$$

$$\begin{aligned}\mathcal{G}(f+h)(x) &= \int_{-\pi}^{\pi} f(t)g(x-t)dt + \int_{-\pi}^{\pi} h(t)g(x-t)dt \\ &= \mathcal{G}f(x) + \mathcal{G}h(x)\end{aligned}$$

Convolution: Linearity

We can see convolution as the application of a linear operator $\mathcal{G}$:

$$\mathcal{G}f(x) = (f \star g)(x) = \int_{-\pi}^{\pi} f(t) \underbrace{g(x-t)}_{\text{filter}} dt$$

It is easy to show that $\mathcal{G}$ is linear:

$$\begin{aligned}
\mathcal{G}(\alpha f(x)) &= \alpha \int_{-\pi}^{\pi} f(t) g(x-t) dt = \alpha\, \mathcal{G}f(x) \\
\mathcal{G}(f+h)(x) &= \int_{-\pi}^{\pi} f(t) g(x-t) dt + \int_{-\pi}^{\pi} h(t) g(x-t) dt \\
&= \mathcal{G}f(x) + \mathcal{G}h(x)
\end{aligned}$$

Translation equivariance can then be phrased as:

$$\mathcal{G}(\mathcal{T}f) = \mathcal{T}(\mathcal{G}f)$$

i.e., the convolution and translation operators commute.

Discrete convolution

In the discrete setting, we deal with vectors $\mathbf{f}, \mathbf{g}$.

We define the convolution sum:

$$(\mathbf{f} \star \mathbf{g})[n] = \sum_{k=-\infty}^{\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$

Discrete convolution

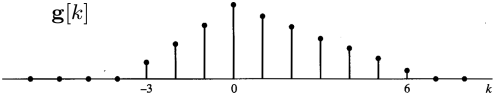
In the discrete setting, we deal with vectors $\mathbf{f}, \mathbf{g}$.

We define the convolution sum:

la convoluzione discreta combina due sequenze discrete (array di numeri) per produrre una terza sequenza discreta. Questo è un esempio di convoluzione 1D (perché i vettori sono delle sequenze di reali)

$$(\mathbf{f} \star \mathbf{g})[n] = \sum_{k=-\infty}^{\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$

usa la somma e non l'integrazione



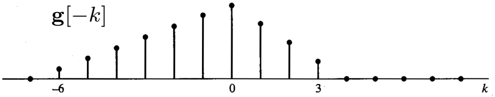
Quando calcoliamo la convoluzione, stiamo valutando ($f \star g$) in un punto specifico n. Ogni valore di n ci dà un singolo punto del segnale convoluto risultante.
L'indice n scorre lungo l'intero dominio dei vettori f e g. Per ogni n, l'indice k scorre attraverso tutti i possibili valori per calcolare la somma dei prodotti.

Discrete convolution

In the discrete setting, we deal with vectors $\mathbf{f}, \mathbf{g}$.

We define the convolution sum:

$$(\mathbf{f} \star \mathbf{g})[n] = \sum_{k=-\infty}^{\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$

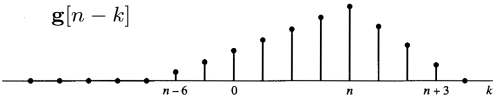


Discrete convolution

In the discrete setting, we deal with vectors $\mathbf{f}, \mathbf{g}$.

We define the convolution sum:

$$(\mathbf{f} \star \mathbf{g})[n] = \sum_{k=-\infty}^{\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$

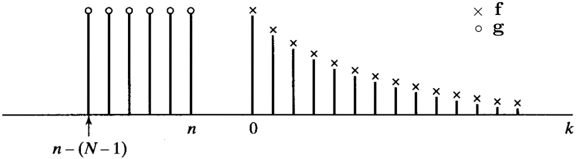


Discrete convolution

In the discrete setting, we deal with vectors $\mathbf{f}, \mathbf{g}$.

We define the convolution sum:

$$(\mathbf{f} \star \mathbf{g})[n] = \sum_{k=-\infty}^{\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$

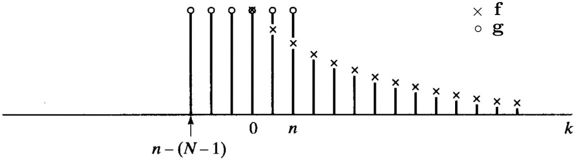


Discrete convolution

In the discrete setting, we deal with vectors $\mathbf{f}, \mathbf{g}$.

We define the convolution sum:

$$(\mathbf{f} \star \mathbf{g})[n] = \sum_{k=-\infty}^{\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$

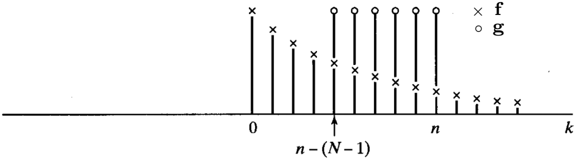


Discrete convolution

In the discrete setting, we deal with vectors $\mathbf{f}, \mathbf{g}$.

We define the convolution sum:

$$(\mathbf{f} \star \mathbf{g})[n] = \sum_{k=-\infty}^{\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$

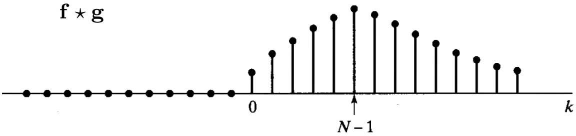


Discrete convolution

In the discrete setting, we deal with vectors $\mathbf{f}, \mathbf{g}$.

We define the convolution sum:

$$(\mathbf{f} \star \mathbf{g})[n] = \sum_{k=-\infty}^{\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$

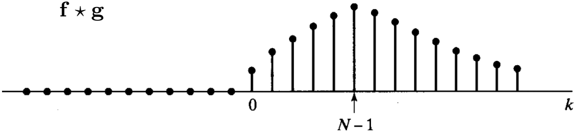


Discrete convolution

In the discrete setting, we deal with vectors $\mathbf{f}, \mathbf{g}$.

We define the convolution sum:

$$(\mathbf{f} \star \mathbf{g})[n] = \sum_{k=-\infty}^{\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$



The specific discretization depends on the boundary conditions.

In the example above, $\mathbf{f}$ was zero-padded in order for the products to be well defined for all shifts.

Discrete convolution

On 2D domains (e.g. RGB images $f : \mathbb{R}^2 \to \mathbb{R}^3$), for each channel:

$$(\mathbf{f} \star \mathbf{g})[m, n] = \sum_k \sum_\ell \mathbf{f}[k, \ell]\, \mathbf{g}[m - k, n - \ell]$$

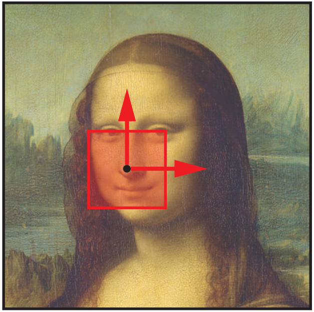
Discrete convolution

On 2D domains (e.g. RGB images $f : \mathbb{R}^2 \to \mathbb{R}^3$), for each channel:

$$(\mathbf{f} \star \mathbf{g})[m,n] = \sum_k \sum_\ell \mathbf{f}[k,\ell]\, \mathbf{g}[m-k, n-\ell]$$

We get the classical interpretation in terms of a moving window:

questo è un esempio di
convoluzione 2D, perché
i vettori *f* e *g* sono in realtà delle
matrici (perché prendono due
parametri in input).
f è l'immagine e *g* è il kernel

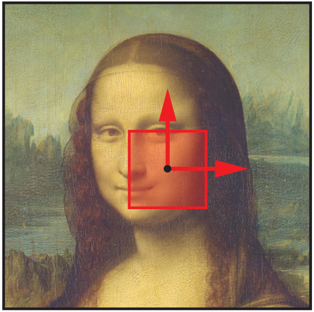


Discrete convolution

On 2D domains (e.g. RGB images $f : \mathbb{R}^2 \to \mathbb{R}^3$), for each channel:

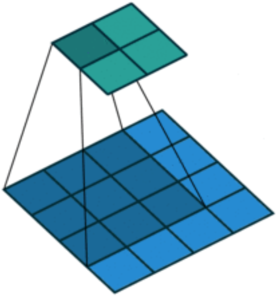
$$(\mathbf{f} \star \mathbf{g})[m,n] = \sum_k \sum_\ell \mathbf{f}[k,\ell]\, \mathbf{g}[m-k, n-\ell]$$

We get the classical interpretation in terms of a moving window:



Boundary conditions and stride

No padding: The convolution kernel is directly applied within the boundaries of the underlying function (an image in this example).

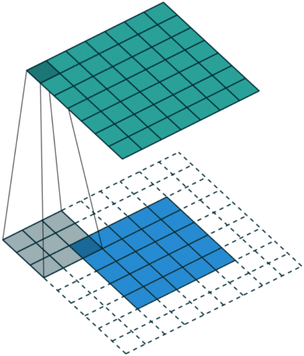


The result of the convolution is a smaller image.

Animations: Vincent Dumoulin

Boundary conditions and stride

Full zero-padding: The domain is enlarged and padded with zeroes. The convolution kernel is applied within the (now larger) boundaries.

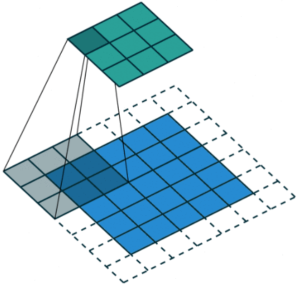


The result of the convolution is a larger image.

Animations: Vincent Dumoulin

Boundary conditions and stride

Arbitrary zero-padding, with stride: The domain is enlarged and padded with zeroes, but not enough to capture the boundary pixels. Further, each discrete step skips one pixel.



The result is the same as no stride followed by downsampling.

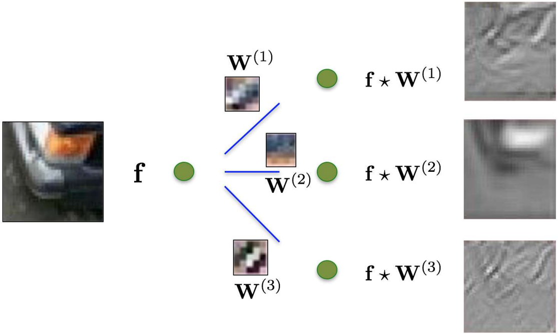
Animations: Vincent Dumoulin

CNN vs. MLP

We are replacing the large matrices of MLPs with small local filters.

CNN vs. MLP

We are replacing the large matrices of MLPs with small local filters.



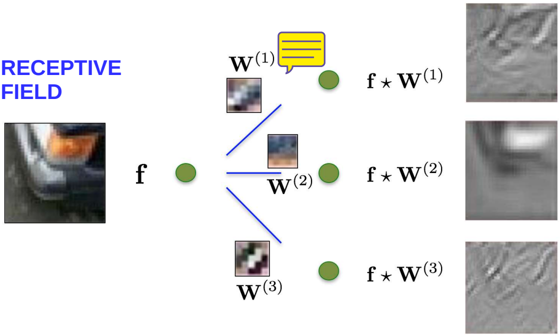
- $O(1)$ parameters per filter; huge gain compared to the MLP.

CNN vs. MLP

We are replacing the large matrices of MLPs with small local filters.

RECEPTIVE FIELD

NEURONE PER CNN



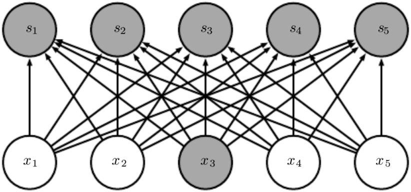
ogni filtro estrae informazioni diverse

- $O(1)$ ameters per filter; huge gain compared to the MLP.
- Filter weights are applied across the entire image $\Rightarrow$ weight sharing, which implements the notion of self-similarity and shift-equivariance.

quindi i pesi della rete sono i valori del kernel e c'è weight sharing perché il kernel ha una dimensione prefissata

Sparse interactions

Fully-connected layer:

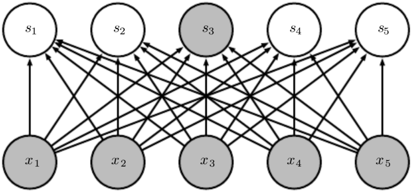


i pesi (le frecce) di ogni dimensione dell'input (le x) sono condivisi con tutti i nodi dell'hidden layer...

Image: Goodfellow et al, 2016

Sparse interactions

Fully-connected layer:

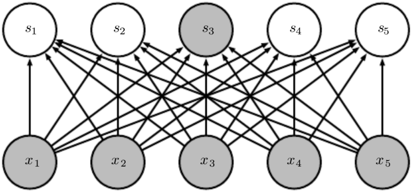


...e quindi ogni nodo dell'hidden layer calcola le informazioni riguardo tutte le dimensioni dell'input

Image: Goodfellow et al, 2016

Sparse interactions

Fully-connected layer:

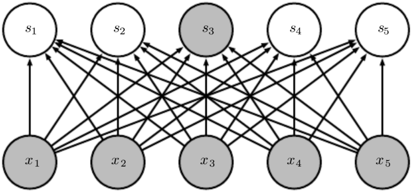


each edge is a different weight

Image: Goodfellow et al, 2016

Sparse interactions

Fully-connected layer:



each edge is a different weight

Convolutional layer:

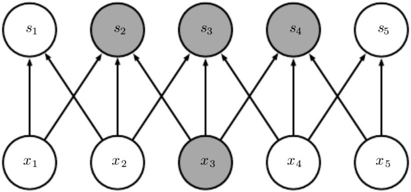
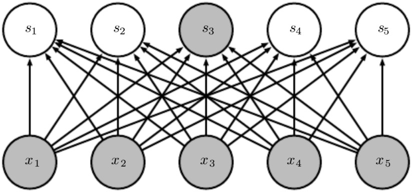


Image: Goodfellow et al, 2016

Sparse interactions

Fully-connected layer:



each edge is a different weight

Convolutional layer:

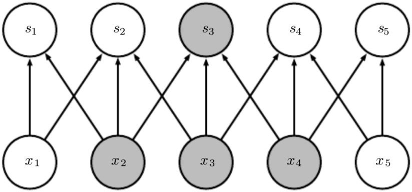
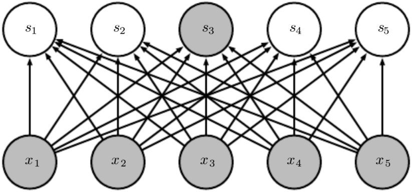


Image: Goodfellow et al, 2016

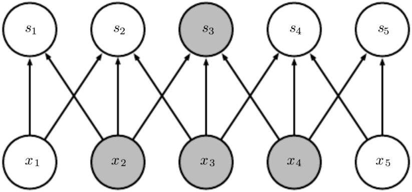
Sparse interactions

Fully-connected layer:



each edge is a different weight

Convolutional layer:



Ad ogni movimento del kernel, solo i nodi input x in grigio influiscono sulla dimensione s3 dell'output (**receptive field**)

the outgoing edges have the same weights for each input variable (weight sharing)

Image: Goodfellow et al, 2016

Pooling

At deep layers, filters interact with larger portions of the input.

3	3	2	1	0	0
3	3	2	1	0	0
3	3	2	1	0	0
3	3	3	2	0	0
3	3	2	1	0	0
3	2	1	1	0	0

Input data

$*$

1	0	1
0	1	0
1	0	1

Filter

$=$

6	8	6	3	1	0
9	13	10	5	2	0
9	14	11	6	3	0
9	13	11	6	2	0
8	13	10	5	3	0
6	7	5	3	1	0

Feature map

Pooling

At deep layers, filters interact with larger portions of the input.

3	3	2	1	0	0
3	3	2	1	0	0
3	3	2	1	0	0
3	3	3	2	0	0
3	3	2	1	0	0
3	2	1	1	0	0

Input data

*

1	0	1
0	1	0
1	0	1

Filter

=

6	8	6	3	1	0
9	13	10	5	2	0
9	14	11	6	3	0
9	13	11	6	2	0
8	13	10	5	3	0
6	7	5	3	1	0

Feature map

13	10	2
14	11	3
13	10	3

Max pooling

Pooling

At deep layers, filters interact with larger portions of the input.

3	3	2	1	0	0
3	3	2	1	0	0
3	3	2	1	0	0
3	3	3	2	0	0
3	3	2	1	0	0
3	2	1	1	0	0

Input data

$*$

1	0	1
0	1	0
1	0	1

Filter

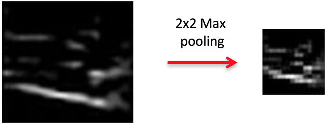
$=$

6	8	6	3	1	0
9	13	10	5	2	0
9	14	11	6	3	0
9	13	11	6	2	0
8	13	10	5	3	0
6	7	5	3	1	0

Feature map

13	10	2
14	11	3
13	10	3

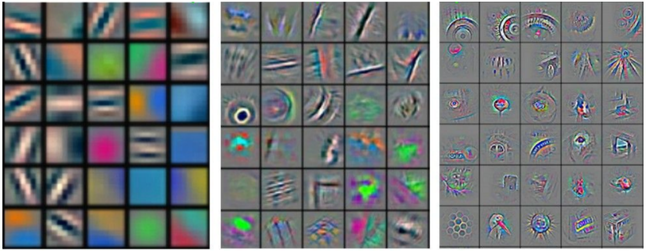
Max pooling
dato da una finestra 2x2



This allows to capture complicated non-local interactions via simple building blocks that only describe sparse interactions.

Learned features

 → low-level features → mid-level features → hi-level features → “car”



Learned features

https://openai.com/blog/microscope/

Suggested reading

Convolution animations, including variants:
https://github.com/vdumoulin/conv_arithmetic

Seminal paper on CNN, seen as a set of feature detectors:
http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf