Deep Learning & Applied AI

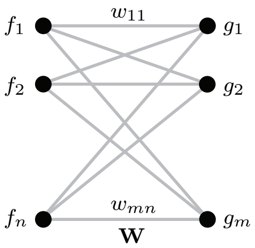
Convolutional neural networks

Emanuele Rodolà
rodola@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

2nd semester a.y. 2019/2020 · April 06, 2020

Neural network (NN)



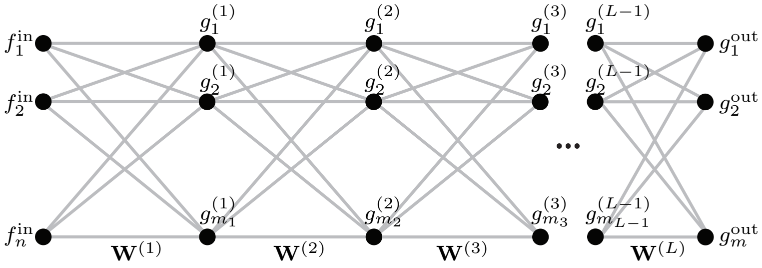
Single linear layer

Linear layer $$g_\ell = \sigma\left(\sum_{\ell'=1}^{n} f_{\ell'} w_{\ell,\ell'}\right) \qquad \begin{array}{l}\ell = 1,\ldots,m\\ \ell' = 1,\ldots,n\end{array}$$

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

Parameters layer weights $\mathbf{W}$ (including bias)

Neural network (NN)



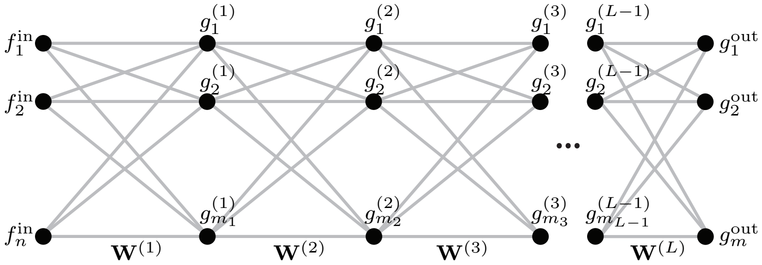
Deep neural network consisting of L layers

Linear layer $$g_\ell^{(k)} = \sigma\left(\sum_{\ell'=1}^{m_{k-1}} g_{\ell'}^{(k-1)} w_{\ell,\ell'}^{(k)}\right) \qquad \begin{aligned}\ell &= 1,\ldots,m_k \\ \ell' &= 1,\ldots,m_{k-1}\end{aligned}$$

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)

Neural network (NN)



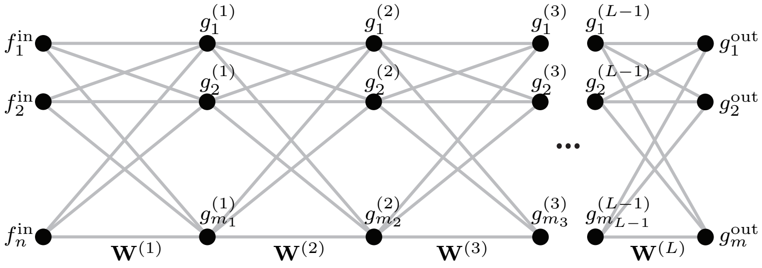
Deep neural network consisting of L layers

Linear layer $\quad \mathbf{g}^{(k)} = \sigma\left(\mathbf{W}^{(k)}\mathbf{g}^{(k-1)}\right)$

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

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

Neural network (NN)



Deep neural network consisting of L layers

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

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

Parameters weights of all layers $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}$ (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.

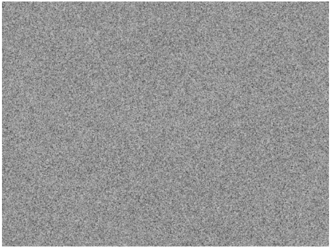
We need additional priors as a (partial) remedy to the above.

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, ...

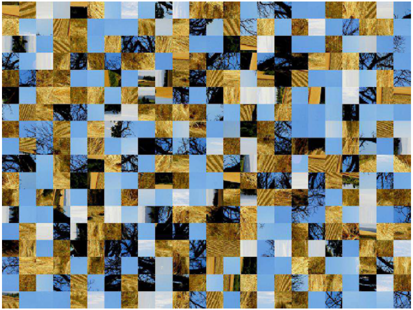


Structure as a strong prior

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, ...



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

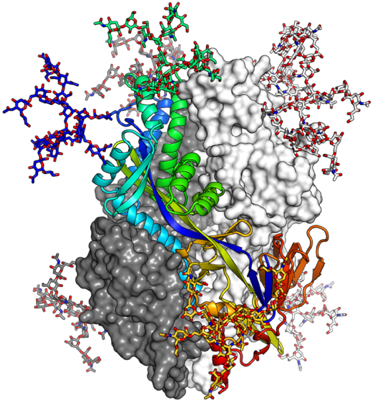
Structure as a strong prior

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

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

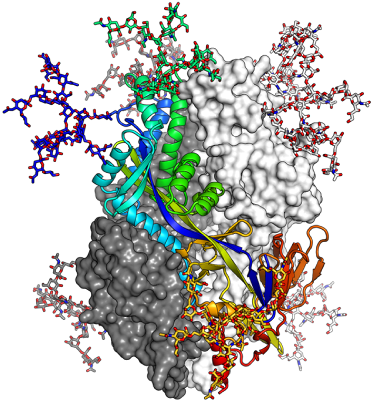
Structure as a strong prior

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



Structure as a strong prior

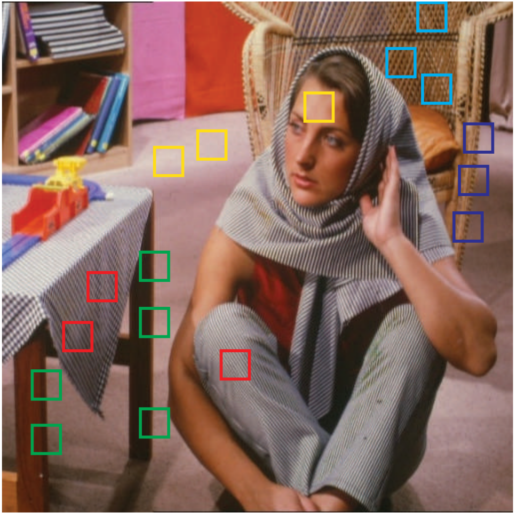
Key insight: Data often carries structural priors in terms of repeating patterns, compositionality, locality, ...



Take advantage of the structure of the data.

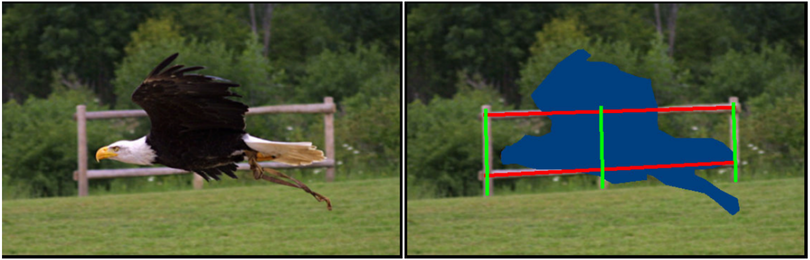
Self-similarity

Data tends to be self-similar across the domain:



Self-similarity

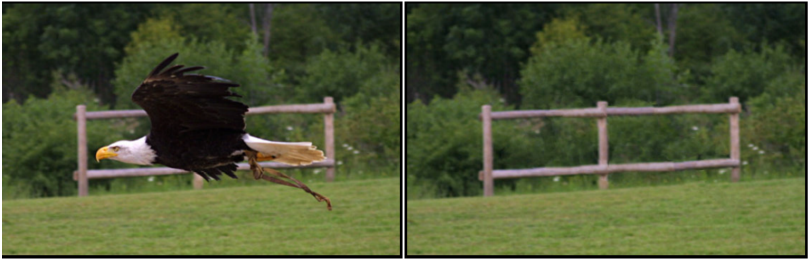
Data tends to be self-similar across the domain:



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:

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

Therefore, it is desirable to enforce translation invariance:

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

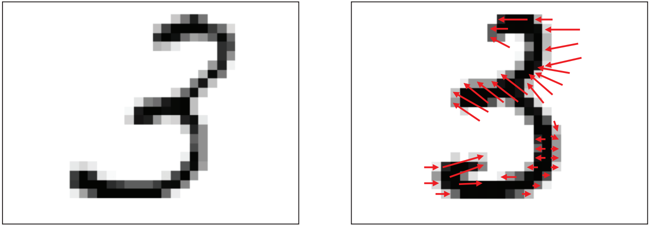
where y is a classification functional.

Deformation invariance

Other types of invariance are possible.

For example, consider the warping operator $\mathcal{L}$ along a deformation field τ:

$$\mathcal{L}_\tau f(x) = f(x - \tau(x))$$

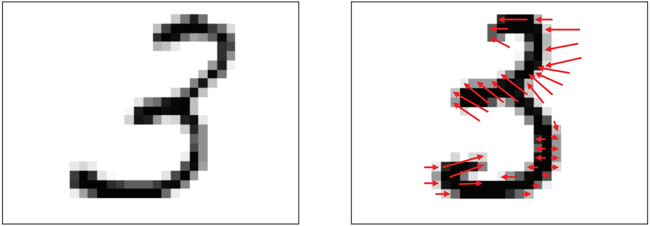


Deformation invariance

Other types of invariance are possible.

For example, consider the warping operator $\mathcal{L}$ along a deformation field τ:

$$\mathcal{L}_\tau f(x) = f(x - \tau(x))$$



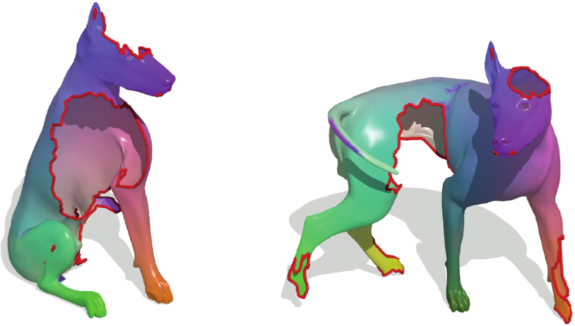
Here, the desirable invariance would be:

$$|y(\mathcal{L}_\tau f) - y(f)| \approx \|\nabla \tau\| \quad \forall f, \tau$$

Deformation invariance

Other types of invariance are possible.

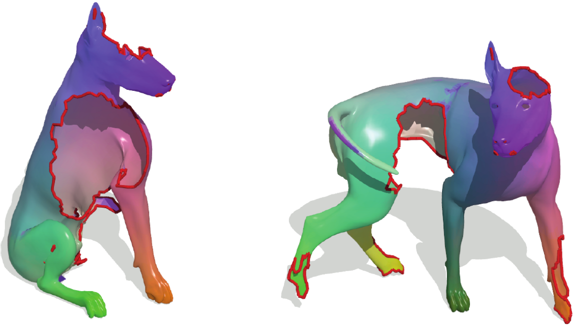
Invariance to partiality and isometric deformations:



Deformation invariance

Other types of invariance are possible.

Invariance to partiality and isometric deformations:

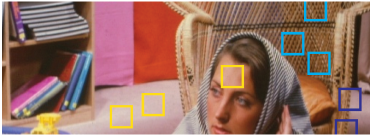


In many cases, invariance can be directly injected into the network.

Today we concentrate on translation invariance.

Hierarchy and compositionality

Translation invariance is desirable across multiple scales:



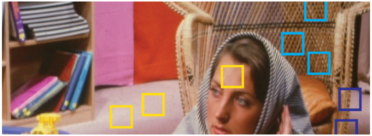
We expect local features to be invariant to their location in the image:

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

where p are image patches of variable size.

Hierarchy and compositionality

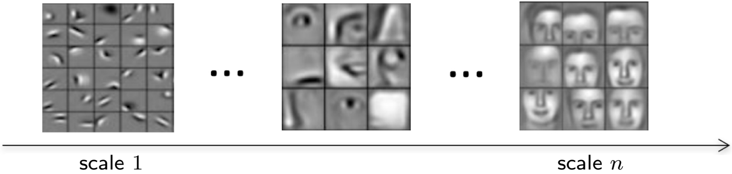
Translation invariance is desirable across multiple scales:



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.



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

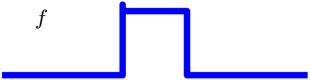
Data is often composed of hierarchical, local, shift-invariant patterns.

CNNs directly exploit this fact as a prior.

Convolution

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

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

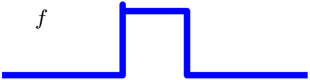


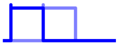
d'Alembert 1754

Convolution

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

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



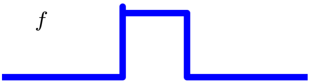


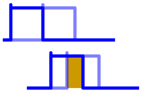
d'Alembert 1754

Convolution

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

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



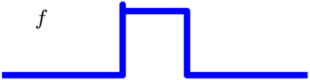


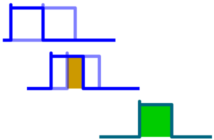
d'Alembert 1754

Convolution

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

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



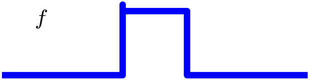


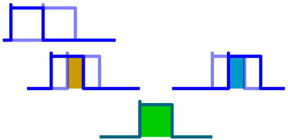
d'Alembert 1754

Convolution

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

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



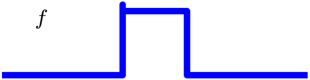


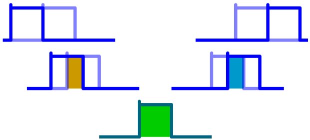
d'Alembert 1754

Convolution

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

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



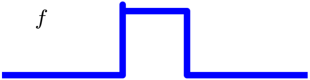


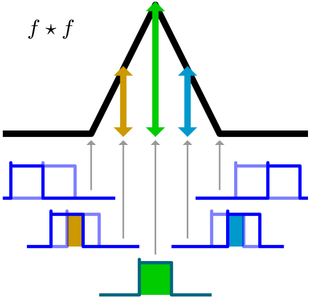
d'Alembert 1754

Convolution

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

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



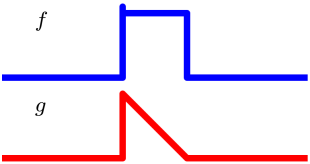


d'Alembert 1754

Convolution

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

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

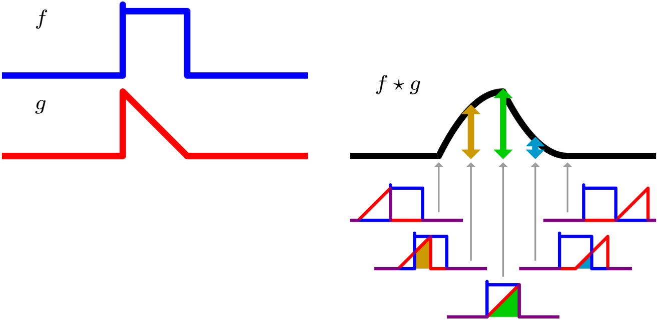


d'Alembert 1754

Convolution

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

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

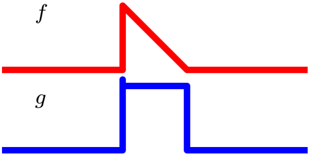


d'Alembert 1754

Convolution

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

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

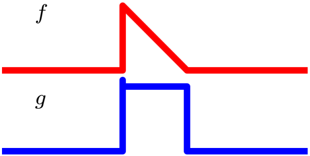


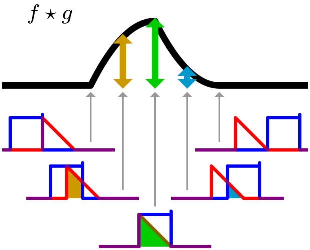
d'Alembert 1754

Convolution

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

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



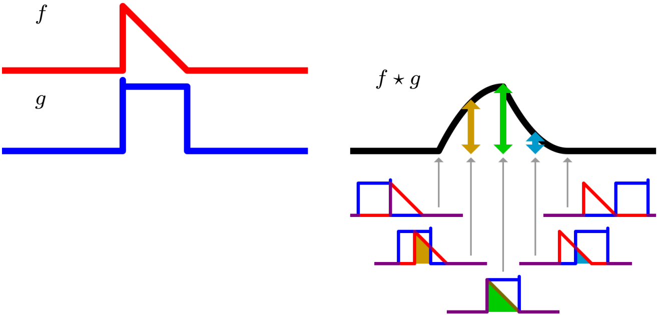


d'Alembert 1754

Convolution

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

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



d'Alembert 1754

Convolution: Commutativity

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

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

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:

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

Convolution is commutative:

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

Convolution: Commutativity

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

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

Convolution is commutative:

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

Convolution: Commutativity

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

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

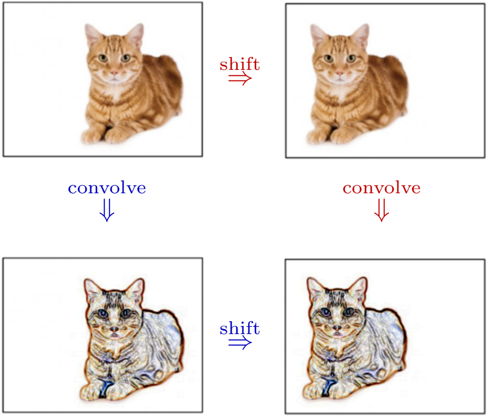
Convolution is commutative:

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

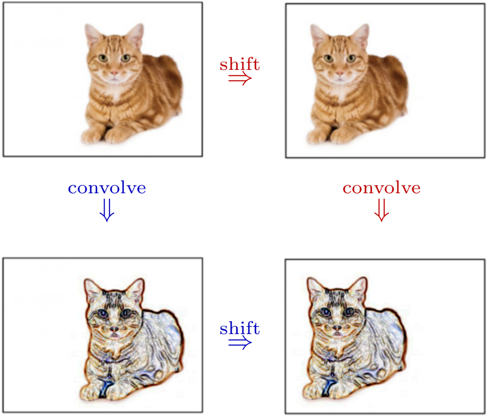
Further, convolution is shift-equivariant:

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

Convolution: Shift-equivariance



Convolution: Shift-equivariance



In fact, equivariance is a defining property of convolutions.

Convolution: Linearity

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

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

Convolution: Linearity

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

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

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 $\mathcal{G}$:

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

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 $\mathcal{G}$:

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

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 $\mathcal{G}$:

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

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 $\mathcal{G}$:

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

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 $\mathcal{G}$:

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

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 $\mathcal{G}$:

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

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

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

Convolution: Linearity

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

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

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

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

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

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:

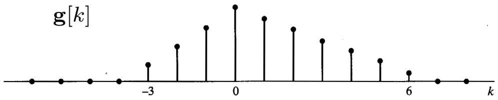
$$(\mathbf{f} \star \mathbf{g})[n] = \sum_{k=-\infty}^{\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$

Discrete convolution

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

We define the convolution sum:

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

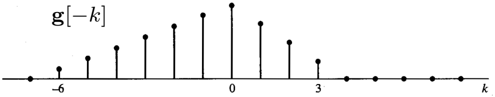


Discrete convolution

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

We define the convolution sum:

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

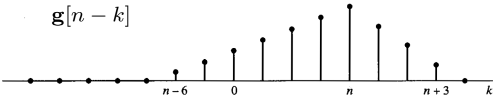


Discrete convolution

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

We define the convolution sum:

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

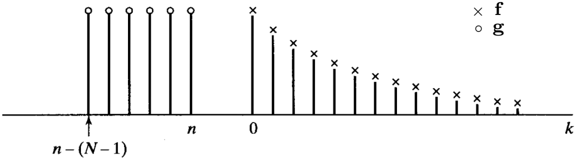


Discrete convolution

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

We define the convolution sum:

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

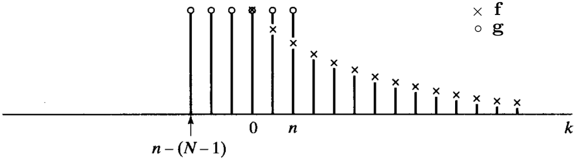


Discrete convolution

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

We define the convolution sum:

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

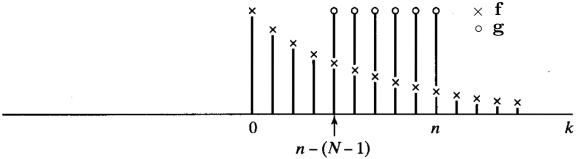


Discrete convolution

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

We define the convolution sum:

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

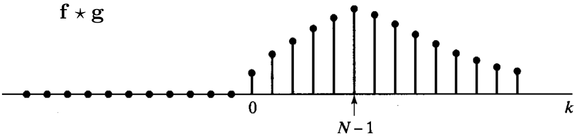


Discrete convolution

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

We define the convolution sum:

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

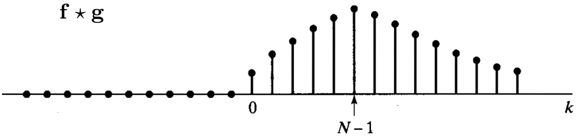


Discrete convolution

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

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

We define the convolution sum:

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

Assuming cyclic boundary conditions, the convolution operator can be encoded as a Toeplitz matrix:

$$\mathbf{f} \star \mathbf{g} = \begin{pmatrix} g_1 & g_2 & \dots & \dots & g_n \\ g_n & g_1 & g_2 & \dots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \dots & g_1 & g_2 \\ g_2 & g_3 & \dots & \dots & g_1 \end{pmatrix} \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}$$

Discrete convolution

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

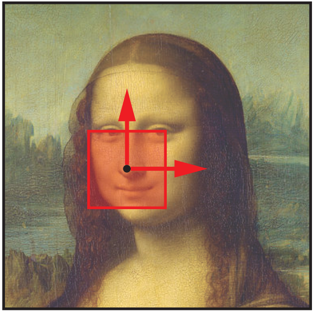
$$(\mathbf{f} \star \mathbf{g})[m, n] = \sum_k \sum_\ell \mathbf{f}[k, \ell]\, \mathbf{g}[m - k, n - \ell]$$

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:

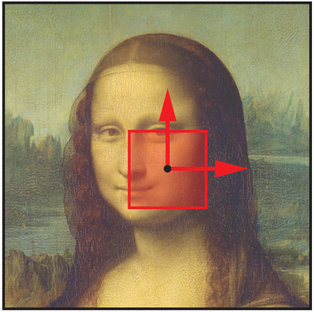


Discrete convolution

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

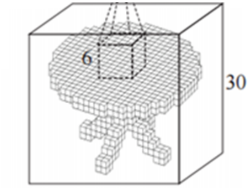
$$(\mathbf{f} \star \mathbf{g})[m, n] = \sum_k \sum_\ell \mathbf{f}[k, \ell]\, \mathbf{g}[m - k, n - \ell]$$

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



Discrete convolution

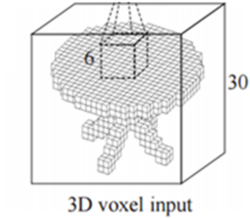
Similarly for 3D domains:



3D voxel input

Discrete convolution

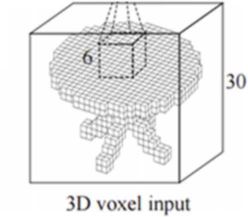
Similarly for 3D domains:



In general, for functions $f : \mathbb{R}^k \rightarrow \mathbb{R}^m$ defined on Euclidean domains, convolution is well-defined up to appropriate boundary conditions.

Discrete convolution

Similarly for 3D domains:

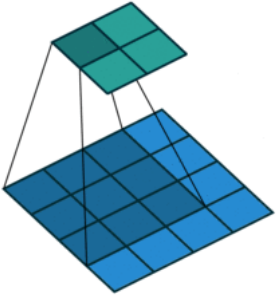


In general, for functions $f : \mathbb{R}^k \to \mathbb{R}^m$ defined on Euclidean domains, convolution is well-defined up to appropriate boundary conditions.

In practice, convolution is often replaced by other operations with similar properties (locality, compositionality, etc.). We'll see this on graphs.

Boundary conditions and stride

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

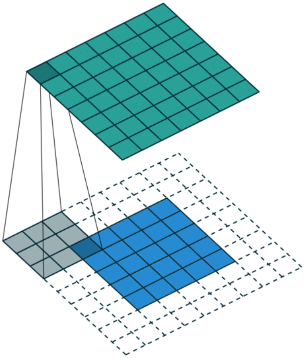


The result of the convolution is a smaller image.

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.

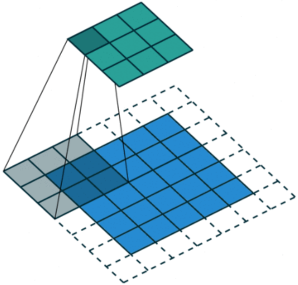


The result of the convolution is a larger image.

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.

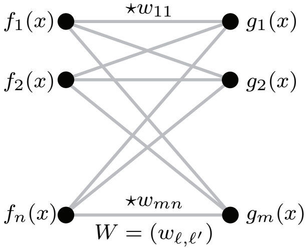


The result is the same as no stride followed by downsampling.

Animations: Vincent Dumoulin

Convolutional neural network (CNN)

Main idea: Compose equivariant layers implemented via convolution.



Single convolutional layer

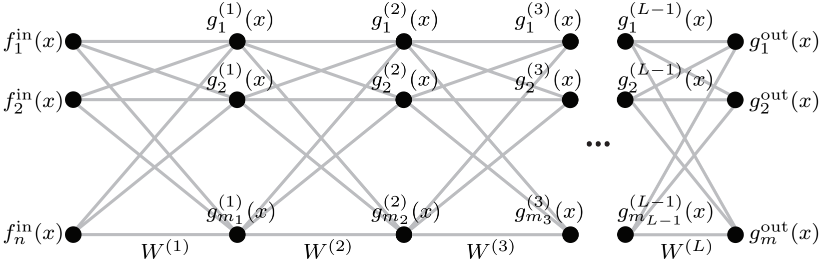
Conv. layer $\quad g_\ell(x) = \sigma\left(\sum_{\ell'=1}^{n} (f_{\ell'} \star w_{\ell,\ell'})(x)\right) \quad \begin{array}{l} \ell = 1, \ldots, m \\ \ell' = 1, \ldots, n \end{array}$

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

Parameters $\quad$ filters W

Convolutional neural network (CNN)

Main idea: Compose equivariant layers implemented via convolution.



CNN consisting of L convolutional layers

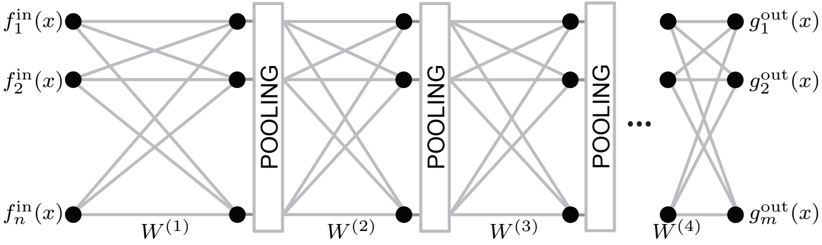
Conv. layer $$g_\ell^{(k)}(x) = \sigma\left(\sum_{\ell'=1}^{m_{k-1}} (g_{\ell'}^{(k-1)} \star w_{\ell,\ell'}^{(k)})(x)\right) \qquad \begin{matrix} \ell = 1, \dots, m_k \\ \ell' = 1, \dots, m_{k-1} \end{matrix}$$

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

Parameters filters of all layers $W^{(1)}, \dots, W^{(L)}$

Convolutional neural network (CNN)

Main idea: Compose equivariant layers implemented via convolution.

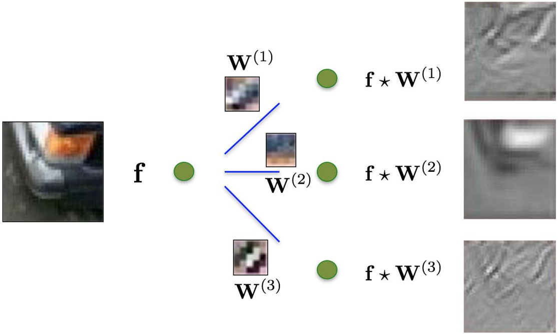


CNN consisting of L convolutional layers interleaved with pooling

Conv. layer	$g_\ell^{(k)}(x) = \sigma\left(\sum_{\ell'=1}^{m_{k-1}} (g_{\ell'}^{(k-1)} \star w_{\ell,\ell'}^{(k)})(x)\right)$	$\ell = 1, \ldots, m_k$ $\ell' = 1, \ldots, m_{k-1}$
Activation, e.g.	$\sigma(x) = \max\{x, 0\}$	rectified linear unit (ReLU)
Parameters	filters of all layers $W^{(1)}, \ldots, W^{(L)}$	
Pooling	$g_\ell^{(k)}(x) = \lVert g_\ell^{(k-1)}(x') : x' \in \mathcal{N}(x)\rVert_n$	$n = 1, 2,$ or ∞

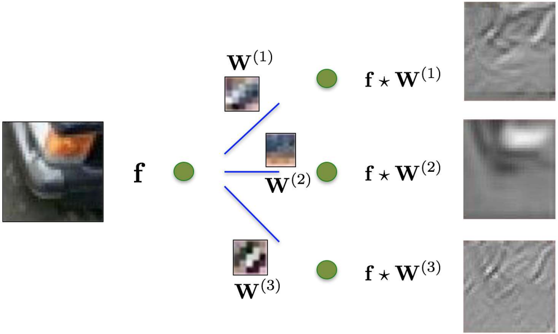
Local filters

Shift-invariance is implemented via convolutional operators.



Local filters

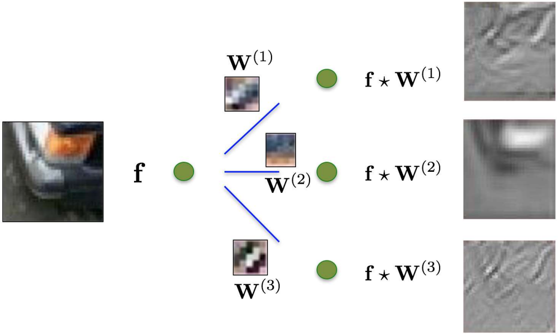
Shift-invariance is implemented via convolutional operators.



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

Local filters

Shift-invariance is implemented via convolutional operators.



- $O(1)$ parameters per filter; huge gain compared to the MLP.
- Filter weights are applied across the entire image $\Rightarrow$ weight sharing.

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

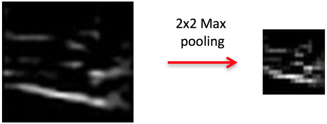
$=$

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



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

Sparse interactions

Fully-connected layer:

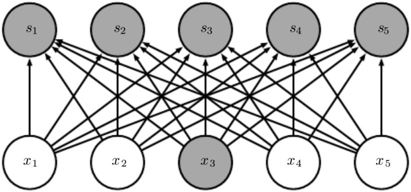


Image: Goodfellow et al, 2016

Sparse interactions

Fully-connected layer:

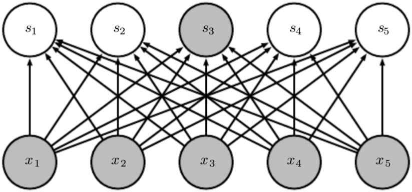
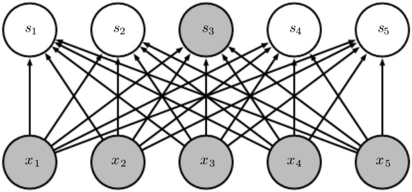


Image: Goodfellow et al, 2016

Sparse interactions

Fully-connected layer:

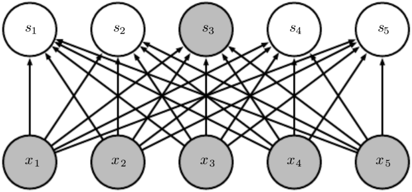


each edge is a different weight

Image: Goodfellow et al, 2016

Sparse interactions

Fully-connected layer:



each edge is a different weight

Convolutional layer:

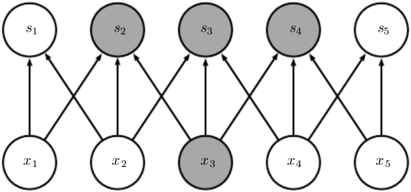
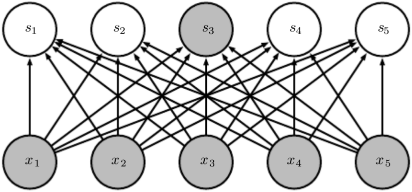


Image: Goodfellow et al, 2016

Sparse interactions

Fully-connected layer:



each edge is a different weight

Convolutional layer:

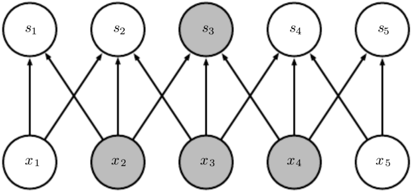
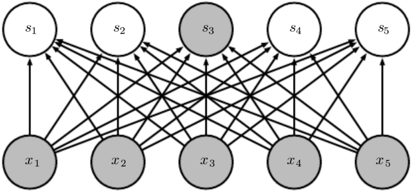


Image: Goodfellow et al, 2016

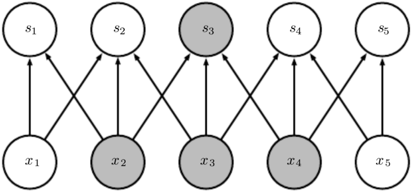
Sparse interactions

Fully-connected layer:



each edge is a different weight

Convolutional layer:

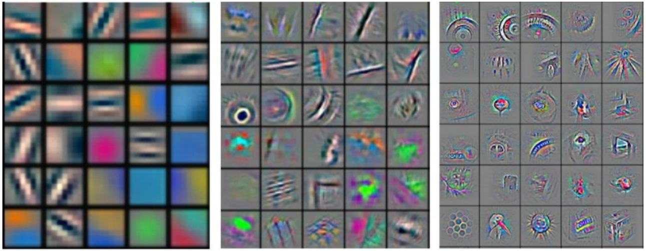


the outgoing edges have the same weights
for each input variable (weight sharing)

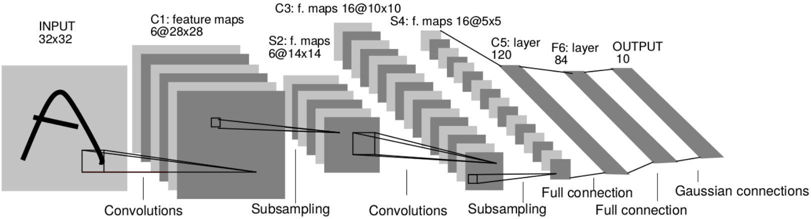
Image: Goodfellow et al, 2016

Learned features

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

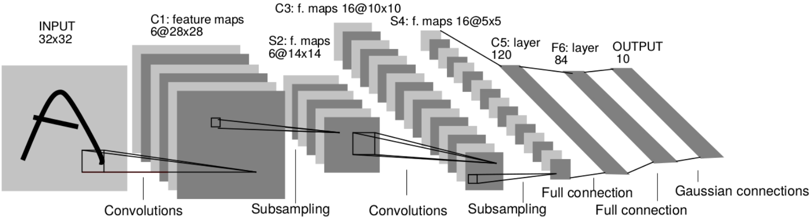


Key properties of CNNs



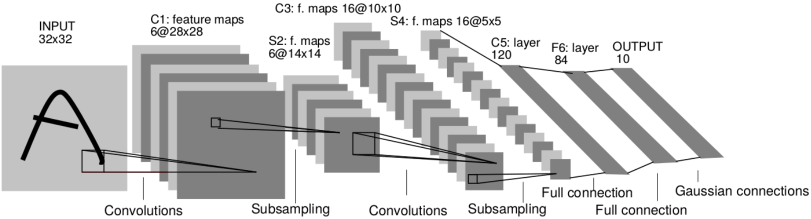
- Convolutional filters **(Translation equivariance)**

Key properties of CNNs



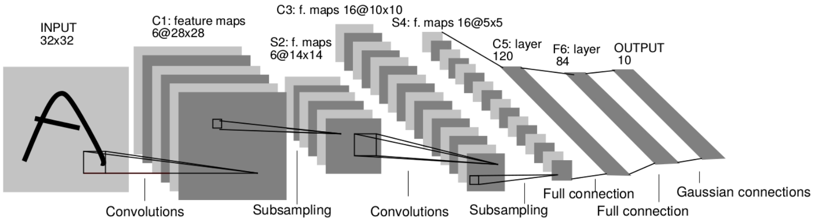
- Convolutional filters **(Translation equivariance)**
- Multiple layers **(Compositionality)**

Key properties of CNNs



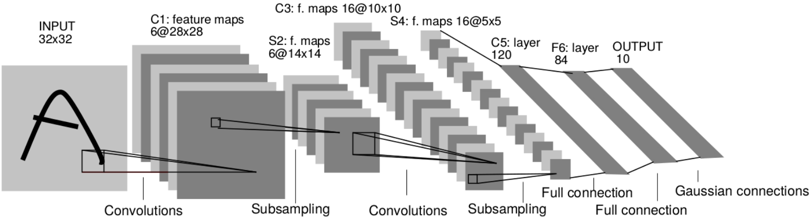
- Convolutional filters **(Translation equivariance)**
- Multiple layers **(Compositionality)**
- Filters localized in space **(Locality)**

Key properties of CNNs



- Convolutional filters **(Translation equivariance)**
- Multiple layers **(Compositionality)**
- Filters localized in space **(Locality)**
- Weight sharing **(Self-similarity)**

Key properties of CNNs



- Convolutional filters **(Translation equivariance)**
- Multiple layers **(Compositionality)**
- Filters localized in space **(Locality)**
- Weight sharing **(Self-similarity)**
- $\mathcal{O}(1)$ parameters per filter (independent of input image size n)

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