

Convolutional Neural Networks

Convolutional neural networks (CNNs) are used when the data $x \in \mathbb{R}^d$ has an underlying Euclidean geometric structure. The most prominent example is when x is an $N \times N$ image so that x can be written as:

$$x(n_1, n_2) \in [0, 1], \quad 0 \leq n_i < N, \quad i=1, 2 \quad (*)$$

In this case $x \in [0, 1]^d$ where $d = N^2$, but x has additional structure given by (*). An example we have already seen is MNIST:

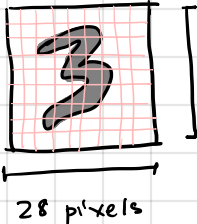
$x =$  28 pixels , $x \in [0, 1]^{28^2} = [0, 1]^{784}$

Image processing is quite common in machine learning, e.g. computer vision, and countless other examples exist. Some popular image databases include:

- MNIST : 70,000 ; 28×28 ; grayscale ; handwritten digits
- CIFAR-10 : 60,000 ; 32×32 ; color images with 10 classes
- CIFAR-100 : 60,000 ; 32×32 ; color images with 100 classes
- ImageNet : Over 14 million color images with over 20,000 classes

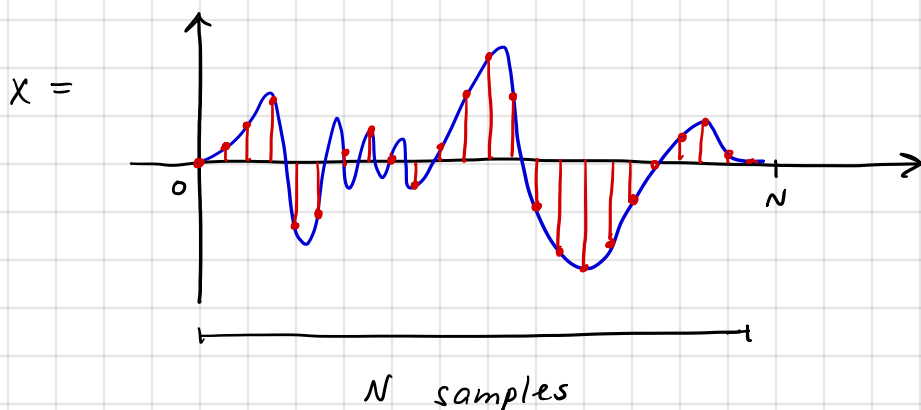
CNNs are also used for data with 1D and 3D geometries. 1D signals consist of, for example:

- audio recordings, as in speech, music, etc.
- medical device recordings, as in EEG, etc.
- more generally time series, although if one wants to make predictions of the future values of a single time series, one should use a recurrent neural network.

In this case $d = N$ and x is of the form:

$$x(n) \in \mathbb{R}, \quad 0 \leq n < N$$

which is the same vector structure from before, but now there is the implication that the order $x(0), x(1), x(2), \dots, x(N-1)$ matters.



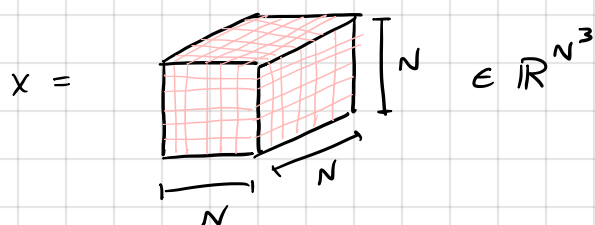
3D signals may consist of, e.g.,

- volumetric medical data
- volumetric data from physics, e.g., many particle physics and fluid mechanics
- self-driving car data
- LiDAR data

The idea is similar to 1D & 2D, in this case:

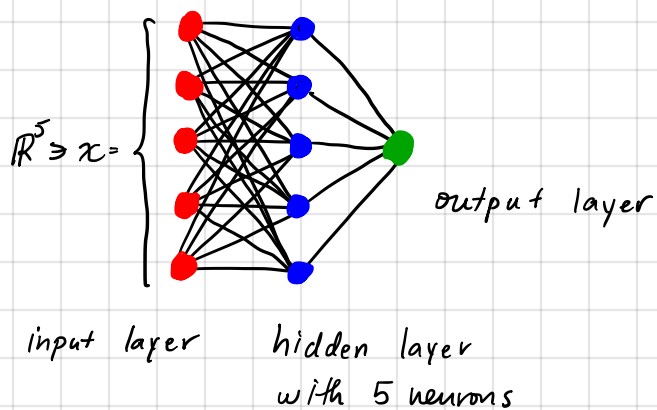
$$x(n_1, n_2, n_3) \in \mathbb{R}, \quad 0 \leq n_i < N, \quad i=1,2,3$$

and so $x \in \mathbb{R}^d$, $d = N^3$



CNNs as special cases of ANNs

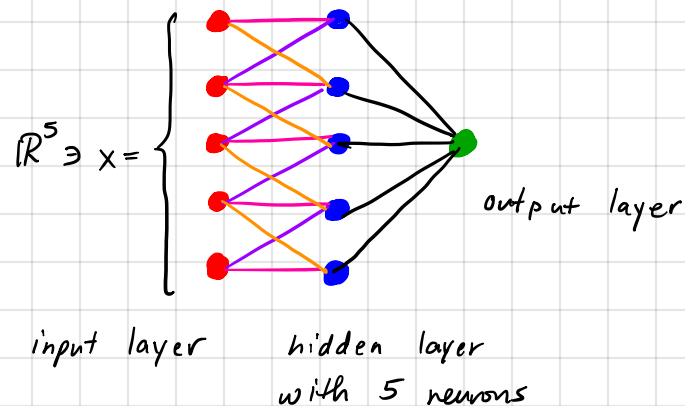
CNNs can be viewed as ANNs with sparse, shared weights. Let us first look at a diagram, supposing that $x \in \mathbb{R}^d$, $d=N$, has a 1D geometric structure.



All edges indicate different weights

Fully connected ANN w/ one hidden layer.

$$f(x; \theta) = \langle \alpha, \sigma(Wx + \beta) \rangle$$



All pink/purple/orange edges have the same weight

Black edges may have different weights

CNN = sparsely connected ANN with shared weights.

$$\tilde{f}(x; \theta) = \langle \alpha, \sigma(\tilde{W}x + \beta) \rangle$$

Let us examine the CNN diagram more closely:

We have five neurons:

$$\left. \begin{aligned} w_0 &= (b, c, 0, 0, 0) \\ w_1 &= (a, b, c, 0, 0) \\ w_2 &= (0, a, b, c, 0) \\ w_3 &= (0, 0, a, b, c) \\ w_4 &= (0, 0, 0, a, b) \end{aligned} \right\} \tilde{W} = \begin{pmatrix} b & c & 0 & 0 & 0 \\ a & b & c & 0 & 0 \\ 0 & a & b & c & 0 \\ 0 & 0 & a & b & c \\ 0 & 0 & 0 & a & b \end{pmatrix} = \text{Toeplitz type weight matrix}$$

(let us ignore the biases.)

Notice then that:

$$\langle x, w_0 \rangle = b x(0) + c x(1)$$

$$\langle x, w_n \rangle = a x(n-1) + b x(n) + c x(n+1), \quad 1 \leq n \leq 3$$

$$\langle x, w_4 \rangle = a x(3) + b x(4)$$

Let $x, y \in \mathbb{R}^N$ and define their correlation as:

$$(x \star y)(n) = \sum_{m=0}^{N-1} x(m) y(n+m), \quad y(n) = 0 \quad \forall n \notin [0, N)$$

Note $(x \star y)(n)$ takes, in general, nonzero values for: $-N < n < N$.

Therefore we can think of $x \star y \in \mathbb{R}^{2N-1}$

A subset of these values correspond to $\langle x, w_n \rangle$. In our example above:

Define $w = w_2 = (0, a, b, c, 0) \in \mathbb{R}^5$. Then:

$$\langle x, w_n \rangle = (x \star w)(n-2), \quad 0 \leq n \leq 4$$

The vector w is called a filter. *Finished class here*

The operation of convolution is closely related to correlation. It is defined as:

$$(x \star y)(n) = \sum_{m=0}^{N-1} x(m) y(n-m)$$

Set $\bar{y}(n) = y(-n)$. Then:

$$(x \star \bar{y})(-n) = \sum_{m=0}^{N-1} x(m) \bar{y}(-n-m) = \sum_{m=0}^{N-1} x(m) y(n+m) = (x \star y)(n) \quad (*)$$

Notice in the CNN network, we only need to learn 3 parameters, a, b, c , in the hidden layer, versus $5^2 = 25$ parameters in the fully connected network.

In 2D, correlation is defined as:

$$(x \star y)(n_1, n_2) = \sum_{m_1=0}^{N-1} \sum_{m_2=0}^{N-1} x(m_1, m_2) y(n_1+m_1, n_2+m_2)$$

In practice, CNNs implement correlation, not convolution. But equation (*) shows they are equivalent, so we will use both definitions and will often use convolution.

In practice these filters are often small, e.g., if $x \in \mathbb{R}^{N^2}$ is an $N \times N$ image, the filters may only be $(2s+1) \times (2s+1)$ in which the filter is indexed as:

$$w(m_1, m_2) \in \mathbb{R}, \quad m_i \in [-s, s] \cap \mathbb{Z}, \quad i=1,2$$

We still index x as:

$$x(n_1, n_2) \in \mathbb{R}, \quad n_i \in [0, N) \cap \mathbb{Z}, \quad i=1,2$$

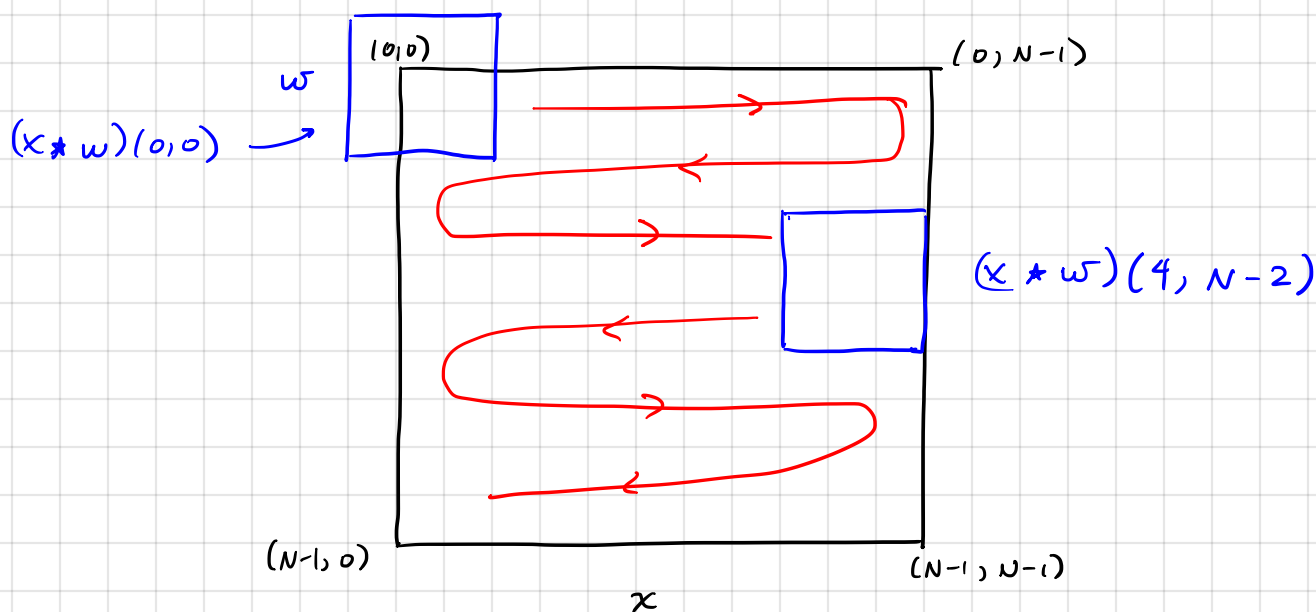
Then:

$$(x * w)(n_1, n_2) = \sum_{m_1=-s}^s \sum_{m_2=-s}^s w(m_1, m_2) x(n_1 + m_1, n_2 + m_2)$$

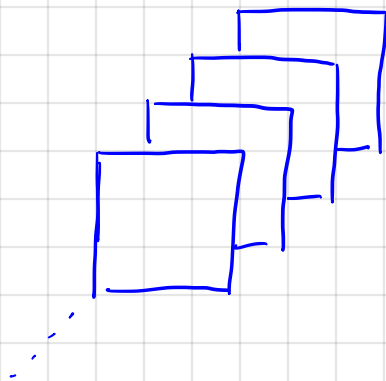
$$n_i \in [0, N) \cap \mathbb{Z}, \quad i=1,2$$

Here s may be, e.g., $s=1, 2$, or 3

Pictorially:



Also in practice there will be more than one filter in each layer. We'll come back to this point later.



A stack of filters for one layer

CNNs work under two related priors, both related to the underlying geometry of the signal:

(i) Neighboring dimensions, e.g. pixels, are related and their relation is relevant to the task. Note this relation may be "long range," that is, "goes over large portions of the image, but it is still governed by the underlying geometry of each data point."

⇒ weights only between nearby dimensions
 • long range relations obtained through depth and new nonlinearities (more on this later)

(ii) Translation equivariance / invariance: (small) translations of the signal, e.g., $x_t(n) = x(n-t)$, do not change the class of x .

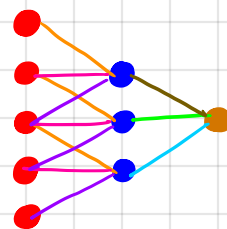
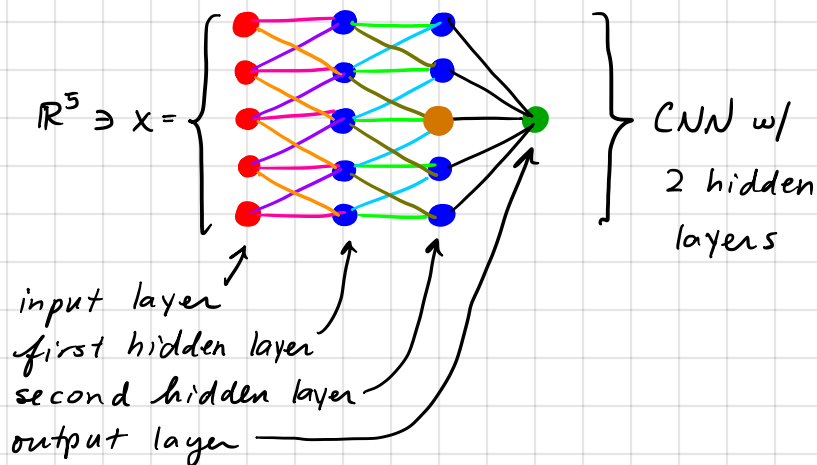
⇒ weight sharing and correlation/convolution operators.
 Indeed,

$$\begin{aligned} (x_t * y)(n) &= (x * y)(n-t) \\ (x_t * y)(n) &= (x * y)(n-t) \end{aligned} \quad \left[\begin{array}{l} \text{assuming} \\ \text{zero padding} \end{array} \right]$$

* Translating the input merely translates the output by the same amount. * ← EQUIVARIANCE

and (ii)

Let us examine (i) a bit more closely. In the previous example we used filter with a receptive field (that is support) of 3 dimensions. But what if we believe there are relations between all 5 dimensions? One option is to use a wider filter. Another is to use a deeper network, e.g.



Closer look at the orange neuron. Notice the orange neuron collects information from all 5 original dims. Thus, even though

$|supp(w_1)| = |supp(w_2)| = 3$,
 the effective support of w_2 relative to the input x , i.e. its receptive field, is 5.

$$f(x; \theta) = \sum_n \alpha(n) \tau(\tau(x * w_1) * w_2)(n)$$

Notice in particular if you have L filters w_l , $1 \leq l \leq L$, each with $|\text{supp}(w_l)| = s$

and you compute an L layer network of the form:

$$\sigma(\dots \sigma(\sigma(x * w_1) * w_2) * \dots * w_L)$$

then the effective receptive field of w_l relative to x is:

$$\text{effective receptive field of layer } l = (s-1)l + 1 \quad \left[\text{This is for 1D} \right]$$

This means the receptive field grows linearly with depth. While this is good, for high dimensional data (e.g., time series with very large numbers of samples such as $N \geq 2^{13}$ or high resolution images) this may not be fast enough. CNNs thus incorporate an additional (nonlinear) **pooling operation**. This pooling operator works by (nonlinearly) downsampling the output of a given layer. Let $x \in \mathbb{R}^N$, a factor 2 downsampling operator applied to x returns a new vector x_d with half the length:

$$x_d(n) = x(2n), \quad 0 \leq n < N/2$$

This type of downsampling is standard in signal processing.

In 2D for signals $x \in \mathbb{R}^{N^2}$, $x_d \in \mathbb{R}^{(N/2)^2} = \mathbb{R}^{N^2/4}$ with:

$$x_d(n_1, n_2) = x(2n_1, 2n_2), \quad 0 \leq n_1, n_2 < N/2$$

CNNs incorporate other types. Two common ones are:

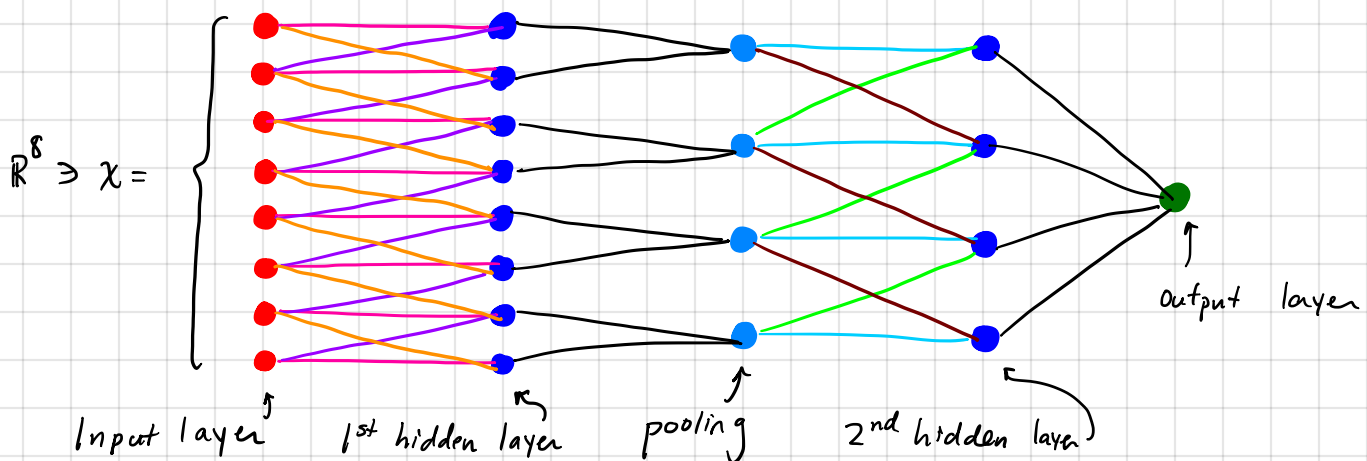
(i) Average pooling:

$$x_d(n) = \frac{1}{2} (x(2n) + x(2n+1)), \quad 0 \leq n < N/2$$

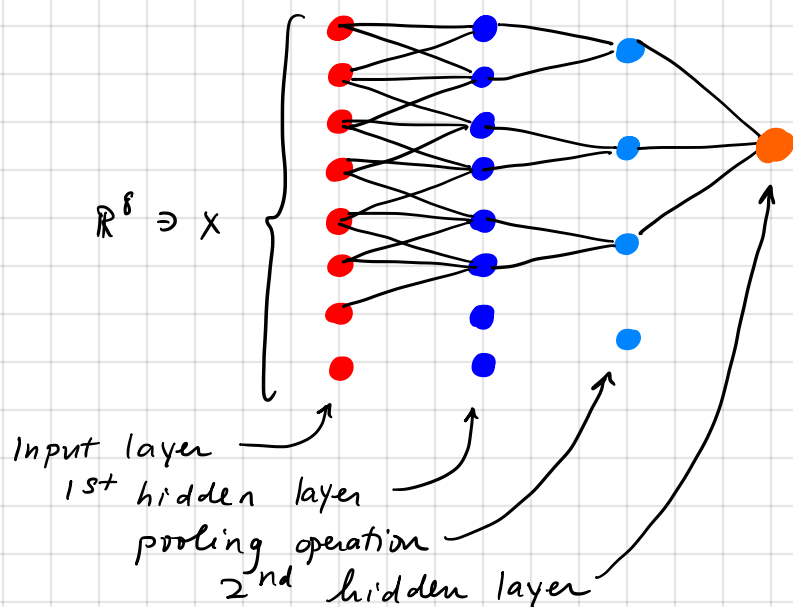
(ii) Max pooling, which is nonlinear:

$$x_d(n) = \max(x(2n), x(2n+1))$$

Pooling operations serve multiple roles. One is to expand the receptive field of deeper filters. They also encode invariance into the representation of x . Let us first consider the receptive field:



Notice how the pooling operation expands the receptive field of the 2nd hidden layer. In the example without pooling, the receptive field of the 2nd hidden layer was 5. Now with the pooling operation we have:



Thus the receptive field of the orange neuron in the 2nd hidden layer is 7 with the pooling operation, but the number of learned parameters has not increased. To get the same receptive field without pooling, we would need 3 hidden layers which would increase the number of trained parameters.

Pooling operations in 1D effectively double the support of the neurons coming after them. In 2D a pooling is a function of 4 pixels:

$$x_d(n_1, n_2) = \text{pool} \left[x(2n_1, 2n_2), x(2n_1+1, 2n_2), x(2n_1, 2n_2+1), x(2n_1+1, 2n_2+1) \right]$$

So the support is quadrupled

Equivariance vs. Invariance

The other important role of convolutional operators and pooling operators is their role in developing equivariant and invariant representations.

Let $x_t(n) = x(n-t)$ be the translation of x by t .

Let $\Phi(x) \in \mathbb{R}^p$ be a representation of x , e.g., $\Phi(x)$ could be the last hidden layer of a neural network.

We say $\Phi(x)$ is translation equivariant if

$$\Phi(x_t)(n) = \Phi(x)(n-t)$$

The representation $\Phi(x)$ is translation invariant if

$$\Phi(x_t) = \Phi(x)$$

The representation $\Phi(x)$ is translation invariant up to scale 2^J if:

$$\|\Phi(x) - \Phi(x_t)\|_2 \leq C |t| 2^{-J} \|x\|_2$$