Deep Learning and Applications

DSA 5204 • Lecture 4

Dr Low Yi Rui (Aaron)
Department of Mathematics

National University of Singapore

Last Time

We discussed deep fully connected neural networks

$$\boldsymbol{h}^{(i+1)} = \mathrm{ReLU}\left(W^{(i+1)}\boldsymbol{h}^{(i)} + \boldsymbol{b}^{(i+1)}\right), \qquad \boldsymbol{h}^{(0)} = \boldsymbol{x}$$
$$\hat{y} = \boldsymbol{w}^T\boldsymbol{h}^{(\ell)} + c$$

They do not generally perform better than other ML methods, even with abundant data

Today, we will look at deep convolutional neural networks, which is arguably what put deep learning ahead in modern ML.

Demo: Permutation Invariance of Fully-Connected NNs

Permutations

A **permutation** of n objects is a one-to-one transformation on these objects.

In other words, it is a **bijection** on $\{1,2,\dots,n\}$

$$\{1,2,3,4,5,6,7,8,9\}$$

$$p \Big\downarrow \qquad p(1) = 3, p(2) = 9 \dots$$

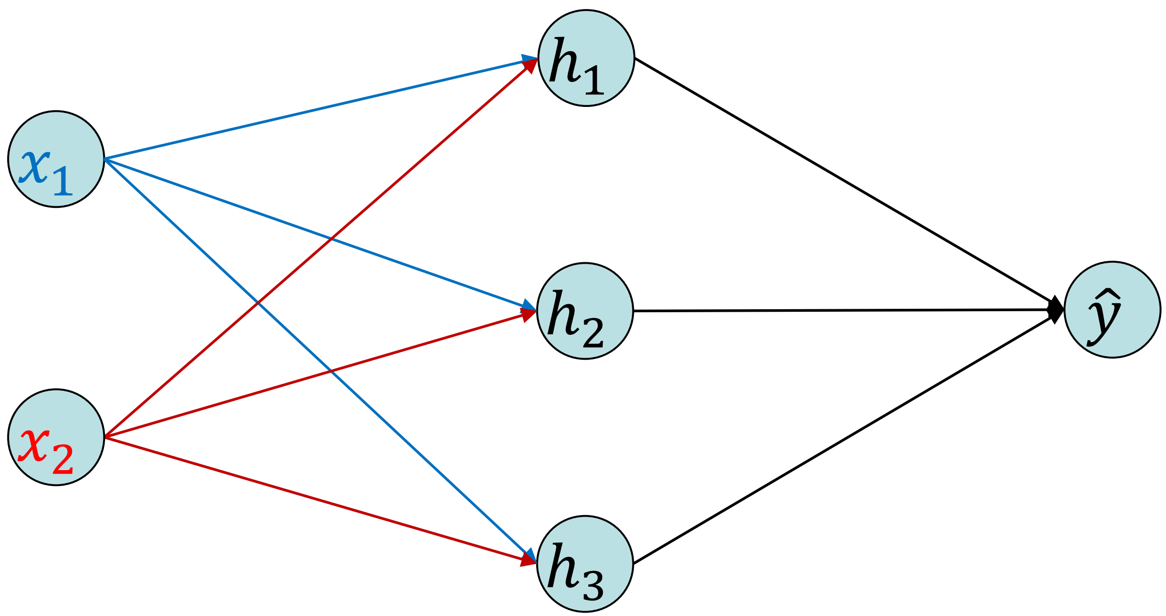
$$\{3,9,1,5,4,6,8,7,2\}$$

Permutation Invariance of Hypothesis Spaces

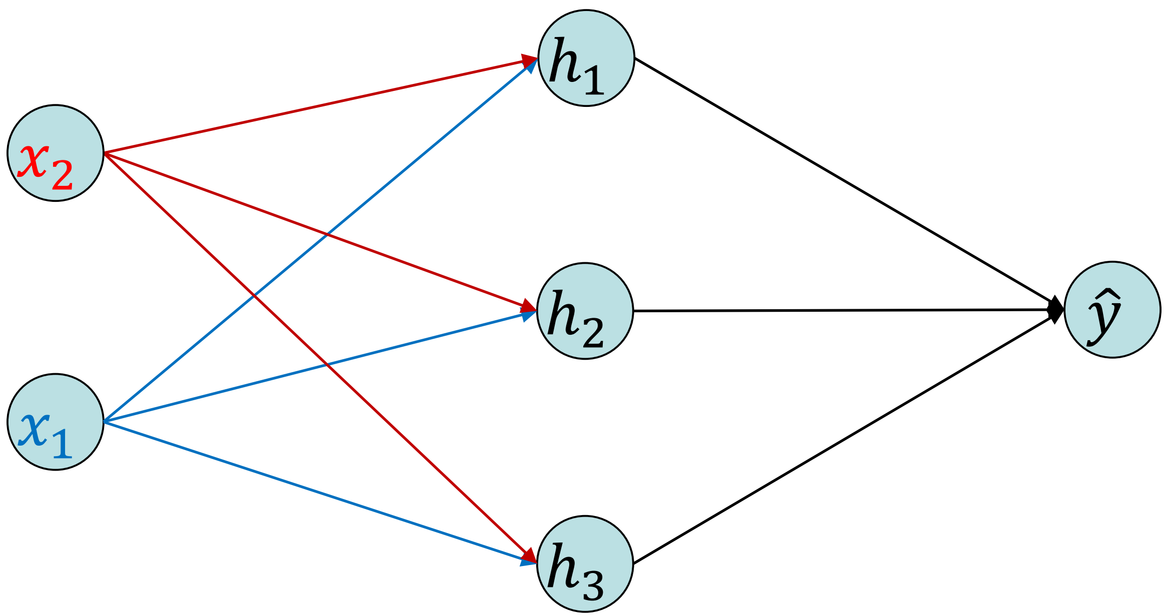
Observe that the FCNN hypothesis space $\mathcal{H}$ has the following invariance property:

Suppose $f \in \mathcal{H}$*, then for any permutation* p *on the indices, define* $f_p(x_1, \dots, x_d) \equiv f\left(x_{p(1)}, \dots, x_{p(d)}\right)$*, then the function* $f_p \in \mathcal{H}$ *as well*

Permutation Invariance of (Fully Connected) Neural Networks



Permutation Invariance of (Fully Connected) Neural Networks



FCNN and Permutation Invariance

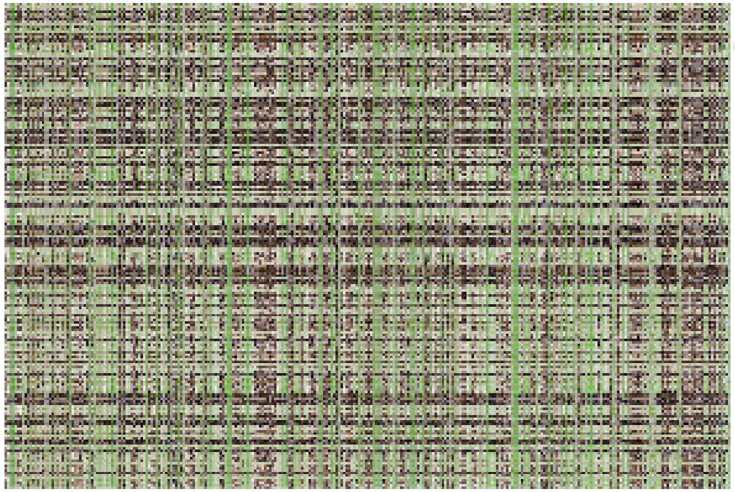
FCNN does not care about permuting the signal’s components since if we can fit one permutation, we can fit any permutation!

Is this sensible?

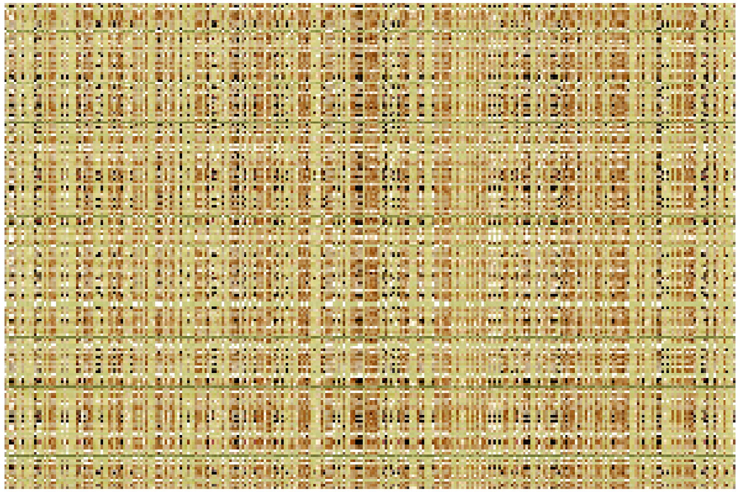
	Cement (component 1)(kg in a m^3 mixture)	Blast Furnace Slag (component 2)(kg in a m^3 mixture)	Fly Ash (component 3)(kg in a m^3 mixture)	Water (component 4)(kg in a m^3 mixture)	Superplasticizer (component 5)(kg in a m^3 mixture)	Coarse Aggregate (component 6)(kg in a m^3 mixture)	Fine Aggregate (component 7)(kg in a m^3 mixture)	Age (day)
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360

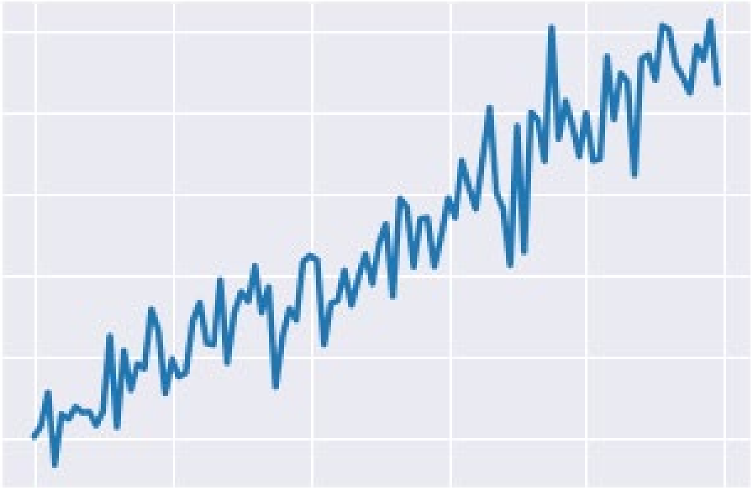
Random Permutation

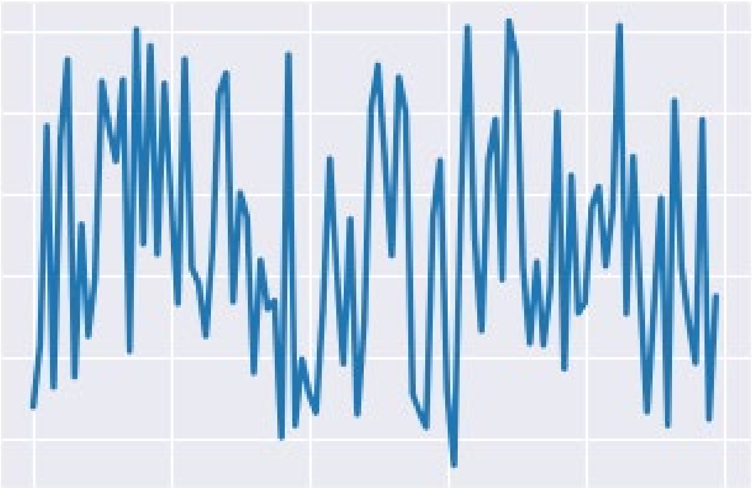
	Fly Ash (component 3)(kg in a m^3 mixture)	Cement (component 1)(kg in a m^3 mixture)	Fine Aggregate (component 7)(kg in a m^3 mixture)	Superplasticizer (component 5)(kg in a m^3 mixture)	Age (day)	Coarse Aggregate (component 6)(kg in a m^3 mixture)	Water (component 4) (kg in a m^3 mixture)	Blast Furnace Slag (component 2)(kg in a m^3 mixture)
0	0.0	540.0	676.0	2.5	28	1040.0	162.0	0.0
1	0.0	540.0	676.0	2.5	28	1055.0	162.0	0.0
2	0.0	332.5	594.0	0.0	270	932.0	228.0	142.5
3	0.0	332.5	594.0	0.0	365	932.0	228.0	142.5
4	0.0	198.6	825.5	0.0	360	978.4	192.0	132.4



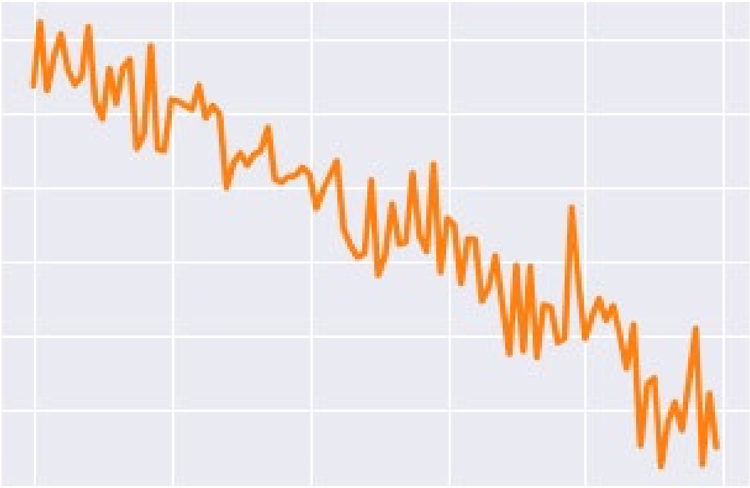
Random Permutation

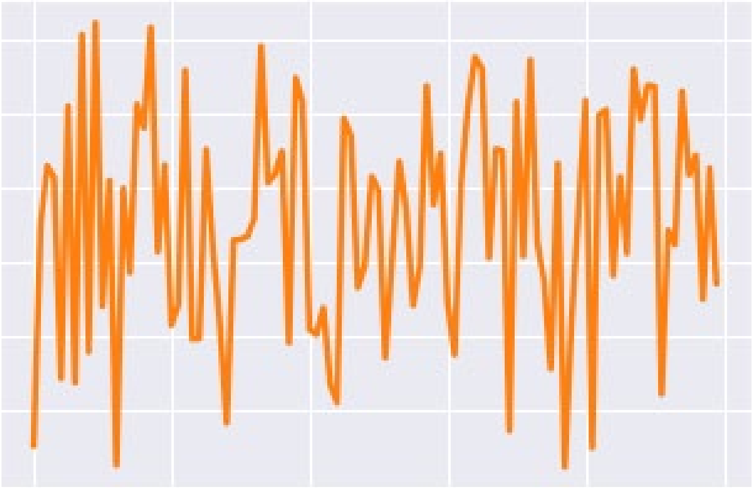






Random Permutation





Limitation of FCNN: The permutation invariance property loses spatial/temporal structure in the data!

The Convolution Operation

Tracking a Spaceship

Suppose we are tracking the location of a spaceship with a laser sensor

The sensor provides a single output

$$x(t) \approx \text{Position of spaceship at time } t$$

The sensor is noisy!

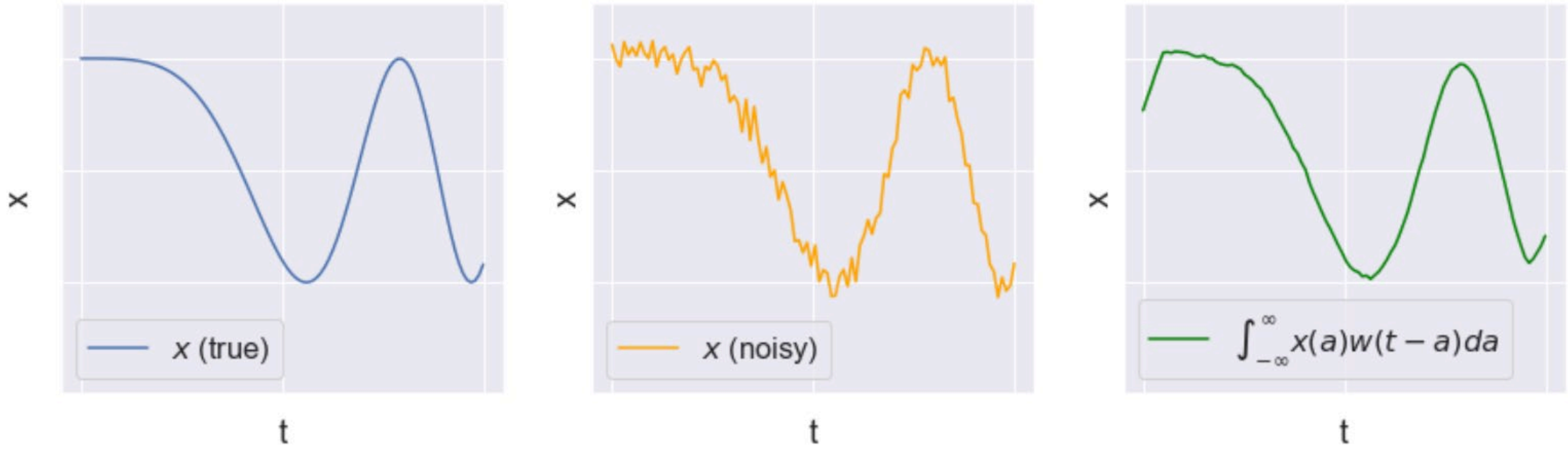
Averaged Measurement

To obtain a less noisy measurement, we average measurements received previously:

$$s(t) = \int_{-\infty}^{+\infty} x(a)w(t-a)da$$

For example, we can take

$$w(t-a) = \begin{cases} e^{-(t-a)} & t \geq a \\ 0 & t < a \end{cases}$$



Convolution of Functions

The previous weighted average is in fact a special case of convolutions.

Given two real-valued functions $x(t)$ and $w(t)$, their convolution is denoted by

$$s(t) = x * w(t)$$

$$s(t) = \int_{-\infty}^{+\infty} x(a)w(t-a)da$$

The convolution operation is symmetric (commutative)

$$x * w = w * x$$

But, we usually distinguishes them

- $x(t)$ is called the **signal** or **input**
- $w(t)$ is called the **kernel** or **filter**
- The output $s = x * w$ is sometimes called the **feature map**

Discrete Convolutions

In reality, measurements can only be carried out at discrete time steps

$$t = \ \dots, 0, 1, 2, \dots$$

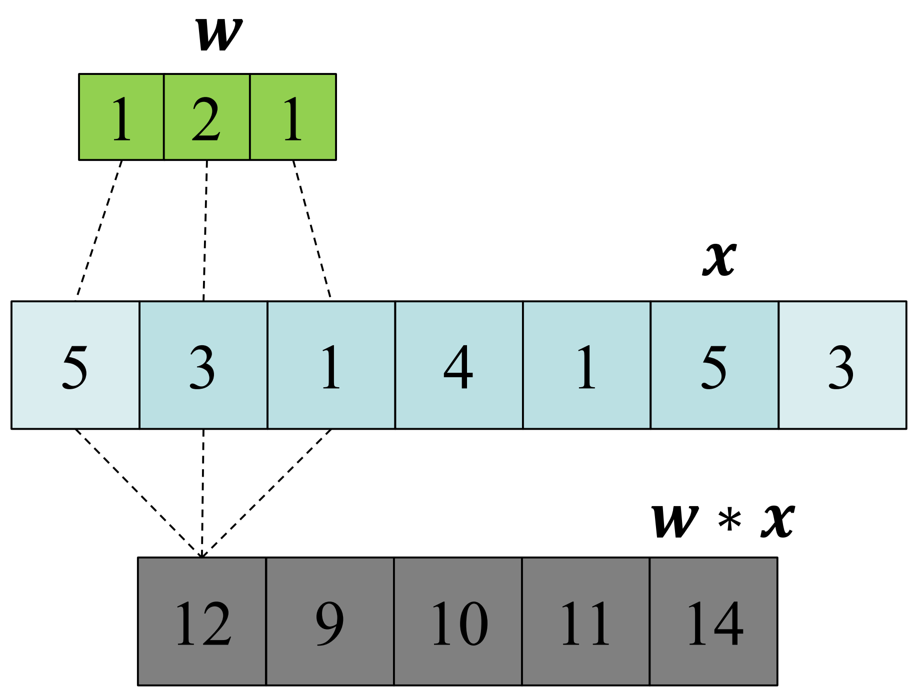
In this case, it is useful to define the discrete convolution

$$\boldsymbol{s}(t) = (\boldsymbol{x} * \boldsymbol{w})(t) = \sum_{-\infty}^{+\infty} \boldsymbol{x}(a)\boldsymbol{w}(t-a)$$

Finite Convolutions and Boundary Conditions

Practical signals are finite, so need to truncate

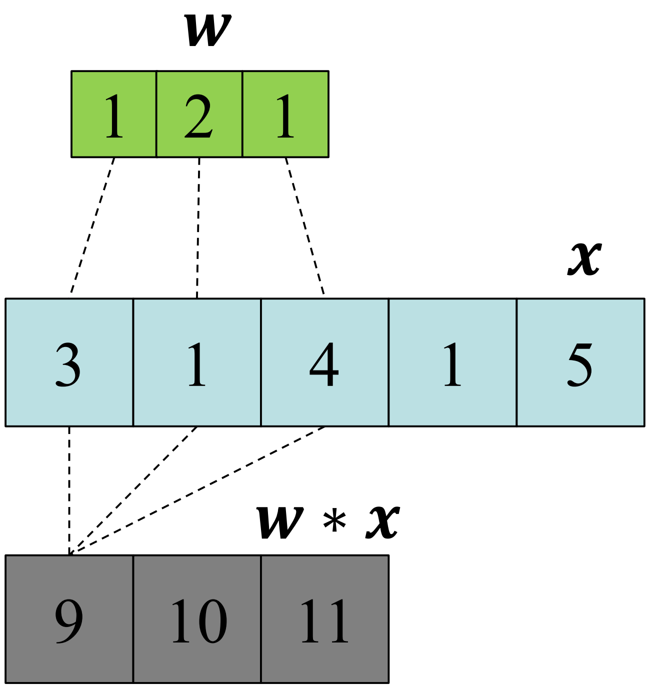
- Circular convolutions



Finite Convolutions and Boundary Conditions

Practical signals are finite, so need to truncate

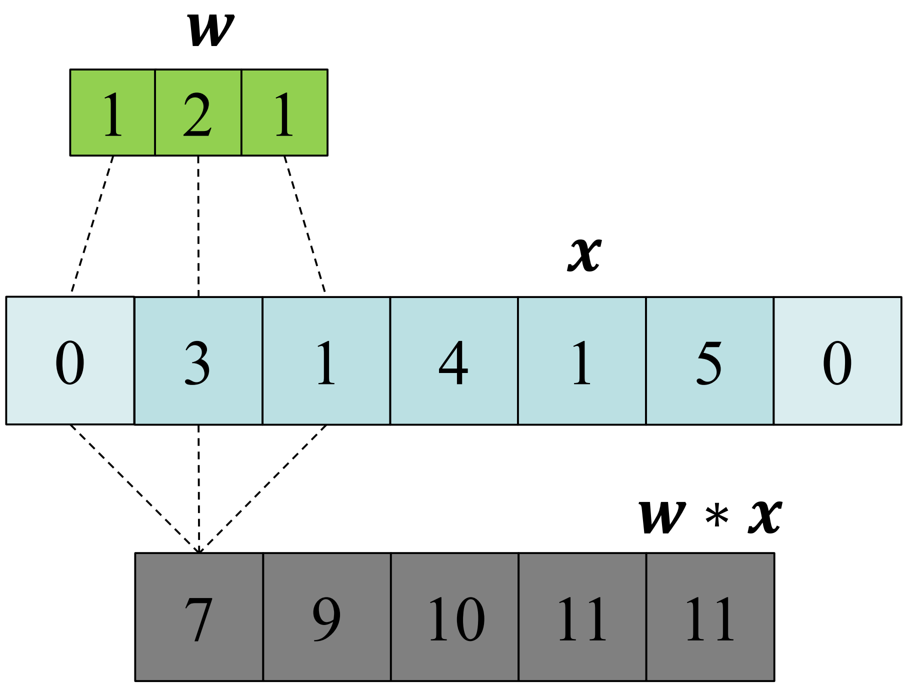
- Valid convolutions



Finite Convolutions and Boundary Conditions

Practical signals are finite, so need to truncate

- Zero-Padded convolutions with "same" padding



2D Discrete (Finite) Convolutions

For image/vision applications, signals come in the form of a 2D matrix. In this case, we define the 2D convolution operation given an input image I and a kernel K

$$S(i,j) = (I * K)(i,j)$$
$$= \sum_{m} \sum_{n} I(m,n)K\,(i-m, j-n)$$

As before, we have $I * K = K * I$

Cross-correlation vs Convolution

A similar concept to convolution is the cross-correlation:

- Convolution

$$S(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

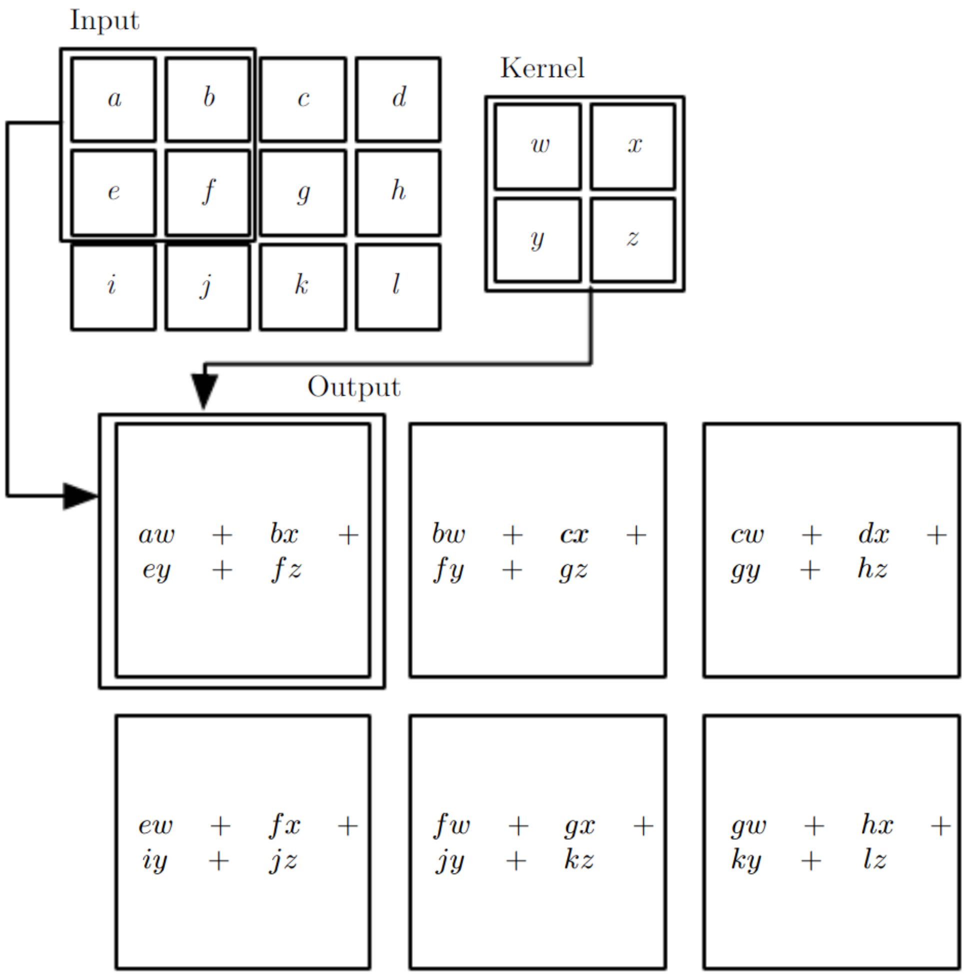
- Cross-correlation

$$S(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

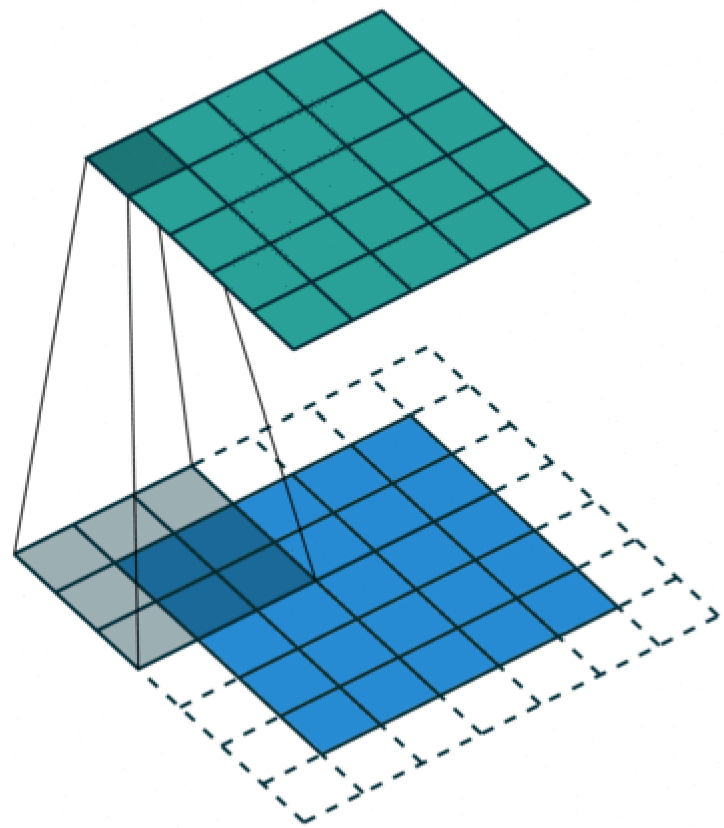
Value is the same, but cross-correlation loses commutativity.

Deep learning libraries usually implement cross-correlation but calls it convolution

Example: 2D “Valid” Convolution



Example: 2D “Same” Convolution



https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Convolution as a Linear Operation

Consider 1D convolution (zero-padded, same)

$\boldsymbol{w} = (w_1, w_2, w_3)$ **and** $\boldsymbol{x} = (x_1, \dots, x_5)$

$$\boldsymbol{w} * \boldsymbol{x} = \underbrace{\begin{pmatrix} w_2 & w_3 & 0 & 0 & 0 \\ w_1 & w_2 & w_3 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 \\ 0 & 0 & 0 & w_1 & w_2 \end{pmatrix}}_{A(\boldsymbol{w})} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$$

Convolutions are still linear operations: $\boldsymbol{w} * \boldsymbol{x} = A(\boldsymbol{w})\boldsymbol{x}$

Convolution Neural Networks

The basic idea of convolutional neural networks (CNN) is to replace the linear operation in FCNN by convolutions:

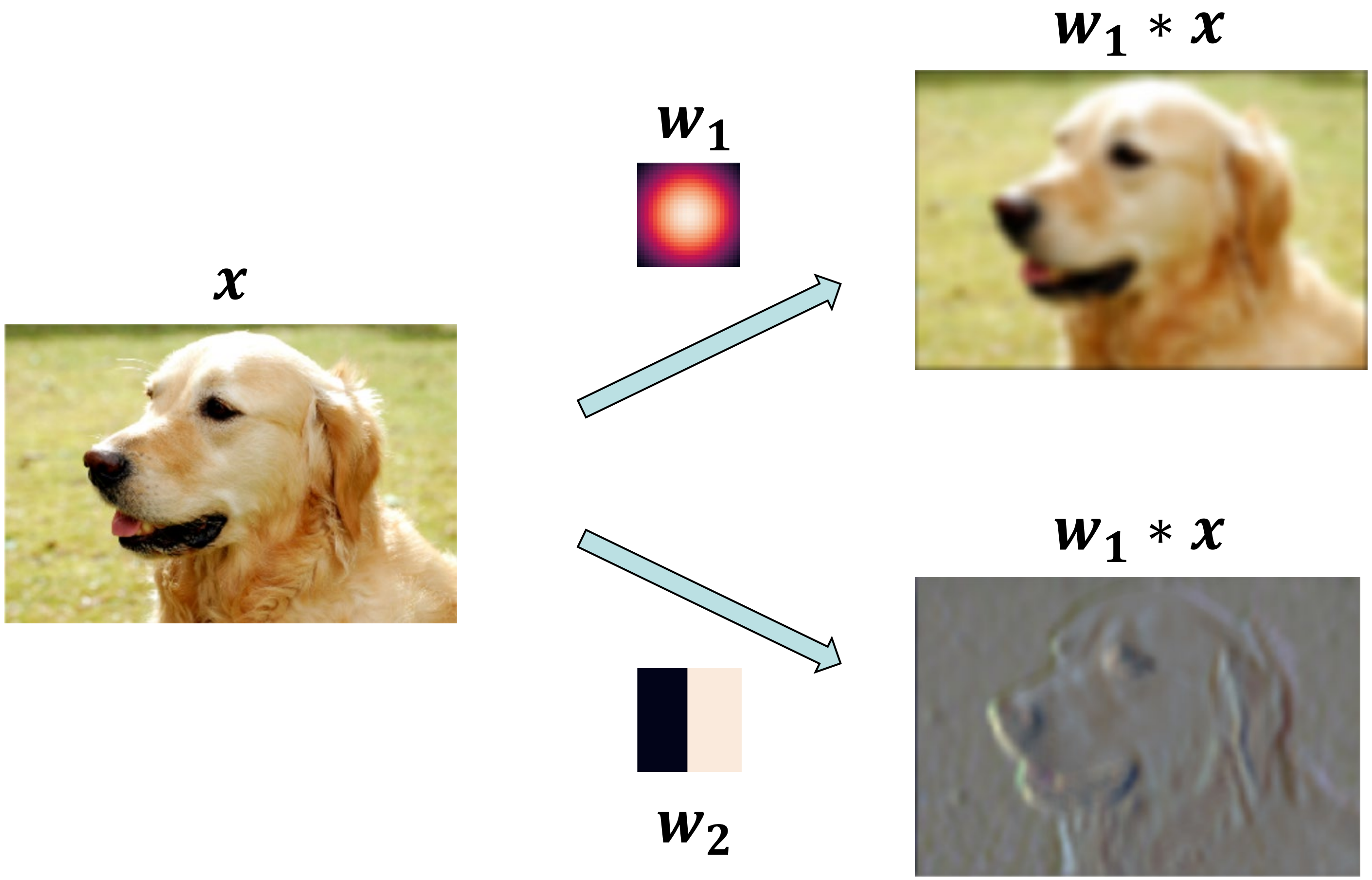
$$\boldsymbol{h}^{(i+1)} = g\left(W^{(i+1)}\boldsymbol{h}^{(i)} + \boldsymbol{b}^{(i+1)}\right)$$

$$\Downarrow$$

$$\begin{aligned}\boldsymbol{h}^{(i+1)} &= g\left(\boldsymbol{w}^{(i+1)} * \boldsymbol{h}^{(i)} + \boldsymbol{b}^{(i+1)}\right)\\ &= g\left(A\left(\boldsymbol{w}^{(i+1)}\right)\boldsymbol{h}^{(i)} + \boldsymbol{b}^{(i+1)}\right)\end{aligned}$$

Why Convolutions?

Motivation 0: Convolutions are effective feature extractors



Motivation 1: Sparse Interactions

Due to the spatial locality of convolutions, not every node interact with every node!

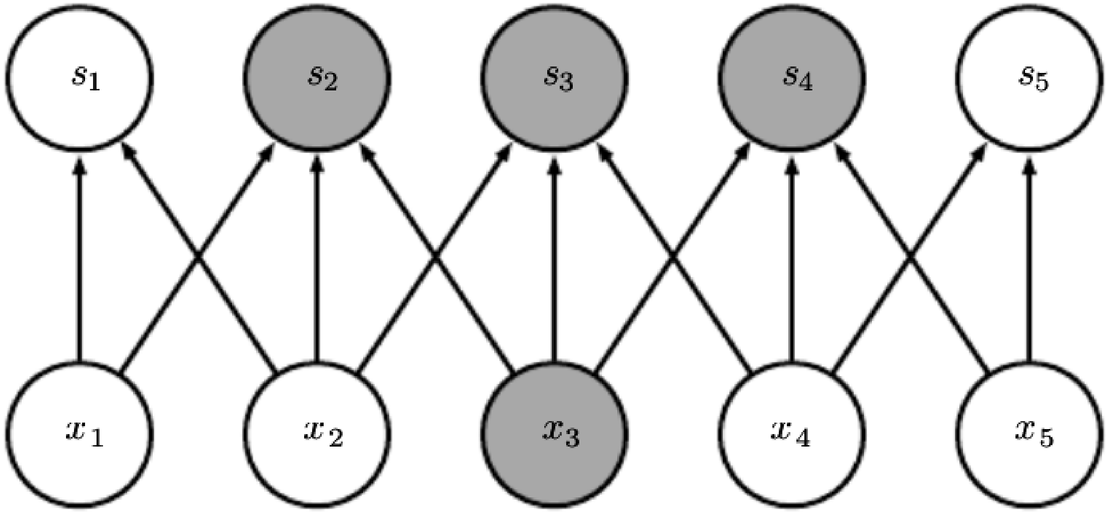
Convolution

$$A(\boldsymbol{w}) = \begin{pmatrix} w_2 & w_3 & 0 & 0 & 0 \\ w_1 & w_2 & w_3 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 \\ 0 & 0 & 0 & w_1 & w_2 \end{pmatrix}$$

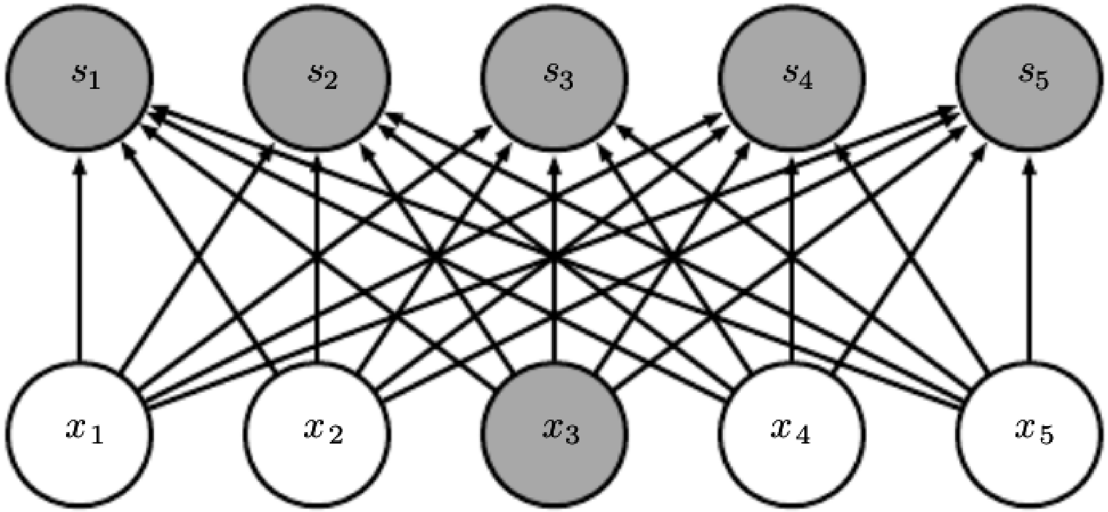
Fully Connected

$$W = \begin{pmatrix} W_{11} & W_{12} & W_{13} & W_{14} & W_{15} \\ W_{21} & W_{22} & W_{23} & W_{24} & W_{25} \\ W_{31} & W_{32} & W_{33} & W_{34} & W_{35} \\ W_{41} & W_{42} & W_{43} & W_{44} & W_{45} \\ W_{51} & W_{52} & W_{53} & W_{54} & W_{55} \end{pmatrix}$$

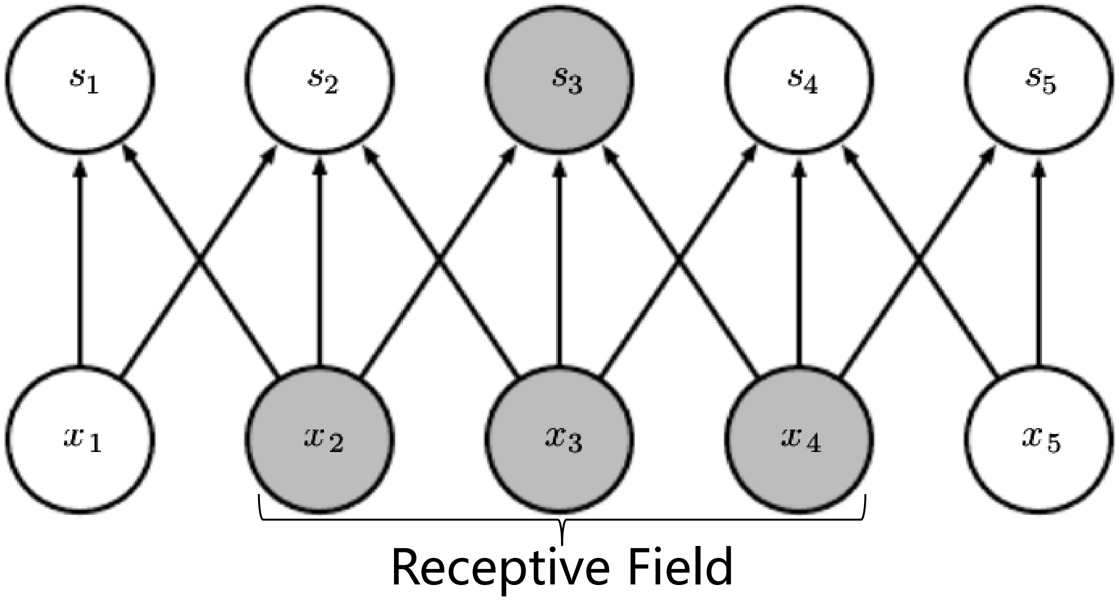
Convolution



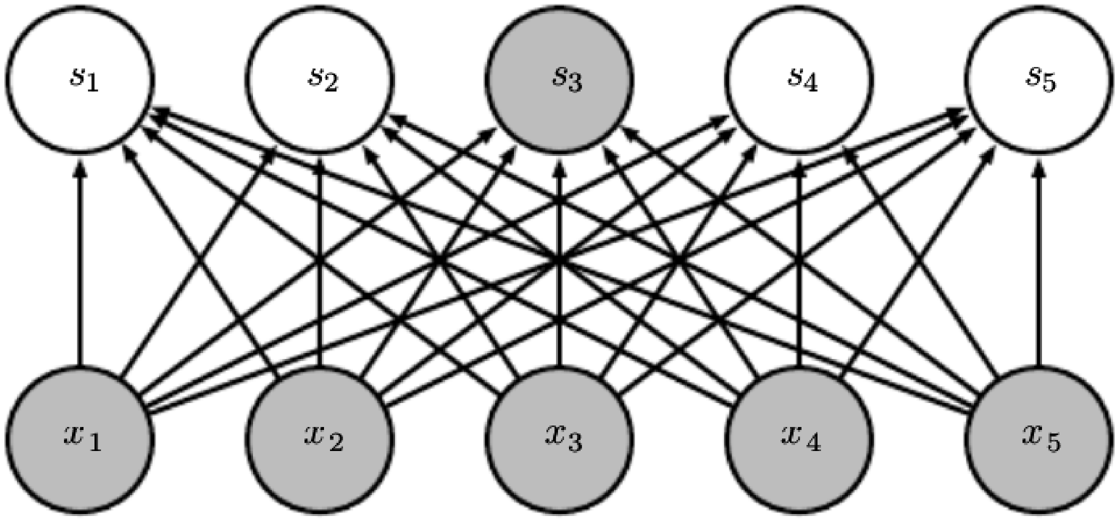
Fully Connected



Convolution



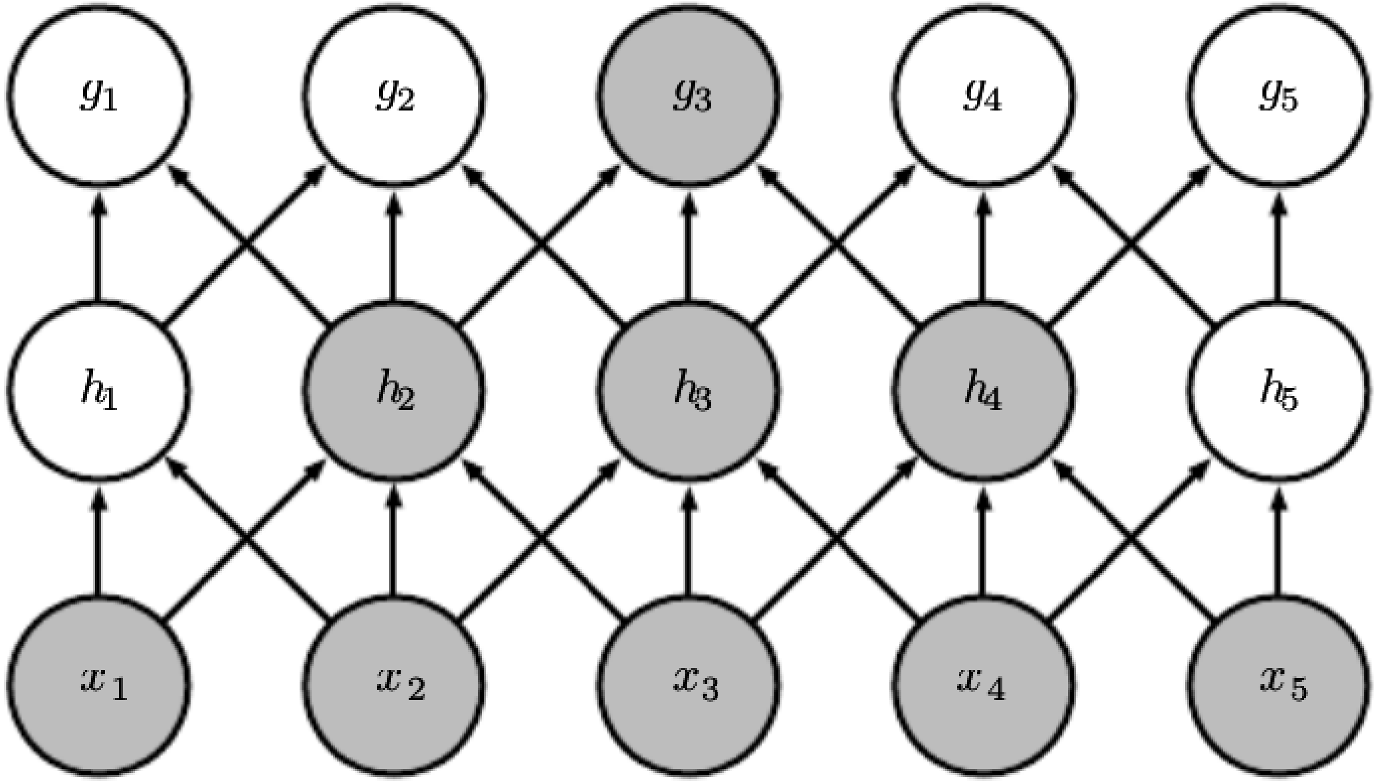
Fully Connected



Implicit Connectivity

Although for each layer, the dependence is local.

For multi-layer CNNs, signals far apart are connected implicitly



Computational Efficiency

Image size: $m \times n$ **Kernel size:** $k \times l$

Computational cost for one matrix multiplication:

- CNN: $\mathcal{O}(k \times l \times m \times n)$
- FCNN: $\mathcal{O}(m^2 \times n^2)$

This represents savings if $k, l \ll m, n$, which is often the case!

Motivation 2: Parameter Sharing

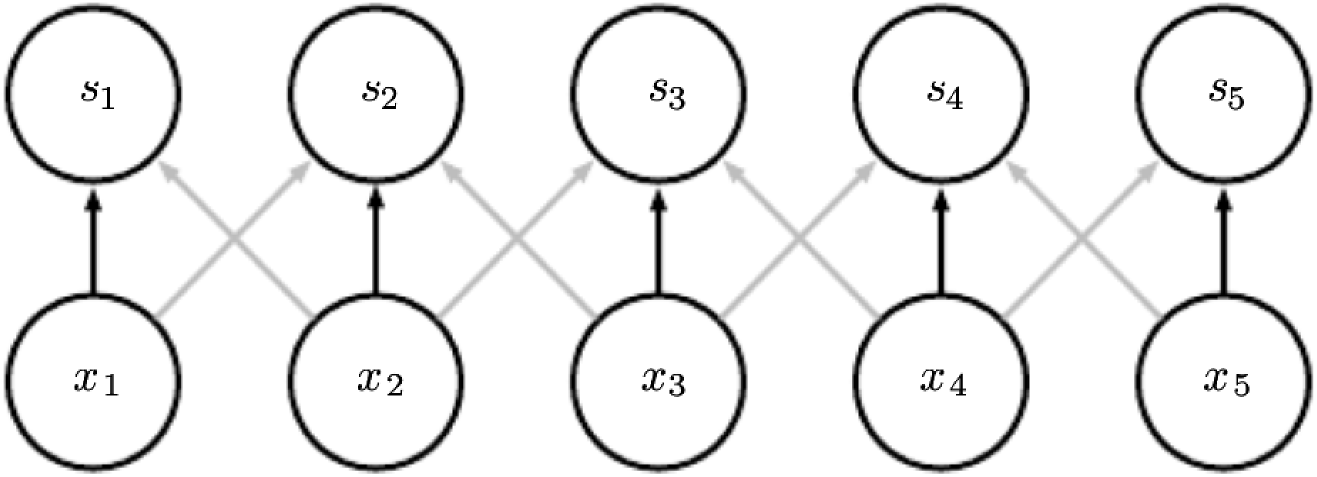
Parameter sharing refers to using the same trainable parameters for more than one place in a ML model

An alternative name is tied weights

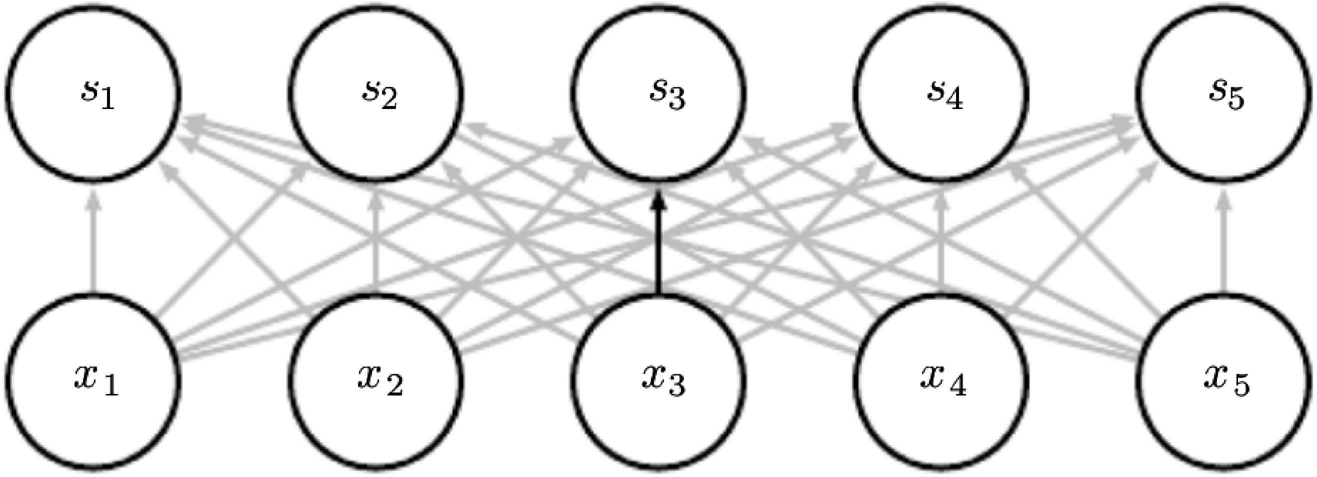
$$A(\boldsymbol{w}) = \begin{pmatrix} w_2 & w_3 & 0 & 0 & 0 \\ w_1 & w_2 & w_3 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 \\ 0 & 0 & 0 & w_1 & w_2 \end{pmatrix}$$

(Only 3 degrees of freedom)

Convolution



Fully Connected



Memory Efficiency

Image size: $m \times n$ **Kernel size:** $k \times l$

Storage size for trained parameters:

$$\mathcal{O}(k \times l)$$

Without parameter sharing, need:

$$\mathcal{O}(m \times n \times k \times l)$$

Motivation 3: Equivariance and Invariance

Let f, g be functions. We say that f is equivariant with respect to g if

$$f\big(g(x)\big) = g\big(f(x)\big)$$

We say that f is invariant with respect to g if

$$f\big(g(x)\big) = f(x)$$

Example: Equivariance

Let

- $g(x) = \lambda x$ be a positive scaling function, where $\lambda \geq 0$
- $f(x) = \mathrm{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$

Then, f is equivariant with respect to g

Example: Invariance

Let

- $g(\boldsymbol{x}) = G\boldsymbol{x}$ where G is an orthogonal matrix ($G^{-1} = G^T$)
- $f(\boldsymbol{x}) = ||\boldsymbol{x}||^2$ be the norm function

Then, f is invariant with respect to g

Let

- $g(\boldsymbol{x})_j = x_{j-\tau}$ be a translation by τ
- $f(\boldsymbol{x}) = \mathbf{1}^T\boldsymbol{x} + c$ be an affine transformation

Then f is invariant with respect to g

General Group Equivariance and Invariance*

More generally, let $f: \mathbb{R}^d \to \mathbb{R}^{d'}$ be a function and $\mathcal{G}$ be a group (of symmetries).

For any $g \in \mathcal{G}$ we denote by ϕ_g and ψ_g some group actions on $\mathbb{R}^d$ and $\mathbb{R}^{d'}$ respectively

- We say that f is **equivariant** under $\mathcal{G}$ if for any $g \in \mathcal{G}$ we have
$$f \circ \phi_g = \psi_g \circ f$$
- Invariance is a special case of equivariance if $\psi_g = \mathrm{id}$ (identity map)

Convolutions and Translations

Let T_τ be a translation of a signal x in time by τ

$$T_\tau(x)(t) = x(t - \tau)$$

Then, for any kernel w we have

$$w * T_\tau(x) = T_\tau(w * x)$$

Because

$$\begin{aligned}[w * T_\tau(x)](t) &= \int w(s)\big(T_\tau(x)\big)(t - s)ds \\ &= \int w(s)x(t - s - \tau)ds \\ &= \int w(s)x[(t - \tau) - s]ds = T_\tau(w * x)\end{aligned}$$

In other words, convolutions are equivariant with respect to translations.

Example: Discrete 1D Convolutions

Let

$$\boldsymbol{x} = (1, 2, 3, 4, 5)$$
$$\boldsymbol{w} = (1, -2, 1)$$

Define translation $T(\boldsymbol{x}) = (x_3, x_4, x_5, x_1, x_2)$, then equivariance holds under circular convolution.

What about other paddings?

Example: Element-wise Nonlinearities

Instead of convolutions, we consider element-wise nonlinear activations.

$$\boldsymbol{x} = (1, 2, 3, 4, 5)$$
$$\sigma(\boldsymbol{x}) = \big(\sigma(1), \dots, \sigma(5)\big)$$

Then $\sigma\big(T(\boldsymbol{x})\big) = T\big(\sigma(\boldsymbol{x})\big)$

Compositions of Equivariant Transformations

Let f_1, f_2 be equivariant with respect g, then so is $f_2 \circ f_1$:

$$f_2 \circ f_1\big(g(x)\big) =$$
$$= f_2\Big(f_1\big(g(x)\big)\Big) = f_2\Big(g\big(f_1(x)\big)\Big)$$
$$= g\Big(f_2\big(f_1(x)\big)\Big) = g\big(f_2 \circ f_1(x)\big)$$

This means that each convolution layer

$$\boldsymbol{h}^{(i+1)} = \sigma\big(\boldsymbol{w} * \boldsymbol{h}^{(i)} + \boldsymbol{b}\big)$$

is equivariant with respect to translations

Building an Invariant Mapping

Let $f_1, \ldots, f_\ell$ be a collection of functions equivariant with respect to g

Let F be another function that is invariant with respect to g

Then, observe that $F \circ f_\ell \circ \cdots \circ f_1$ is invariant with respect to g

In other words, a deep CNN

$$\boldsymbol{h}^{(i+1)} = \sigma\left(\boldsymbol{W} * \boldsymbol{h}^{(i)} + \boldsymbol{b}\right) \qquad \boldsymbol{h}^{(0)} = \boldsymbol{x}$$

$$\hat{y} = \mathbf{1}^T \boldsymbol{h}^{(\ell)} + c$$

is invariant with respect to translations!

Importance of Translation Invariance

Let $f^*\colon \text{Images} \to \{\text{dog}, \text{cat}\}$

These three images are translations of the same image, but they all give the label "dog".

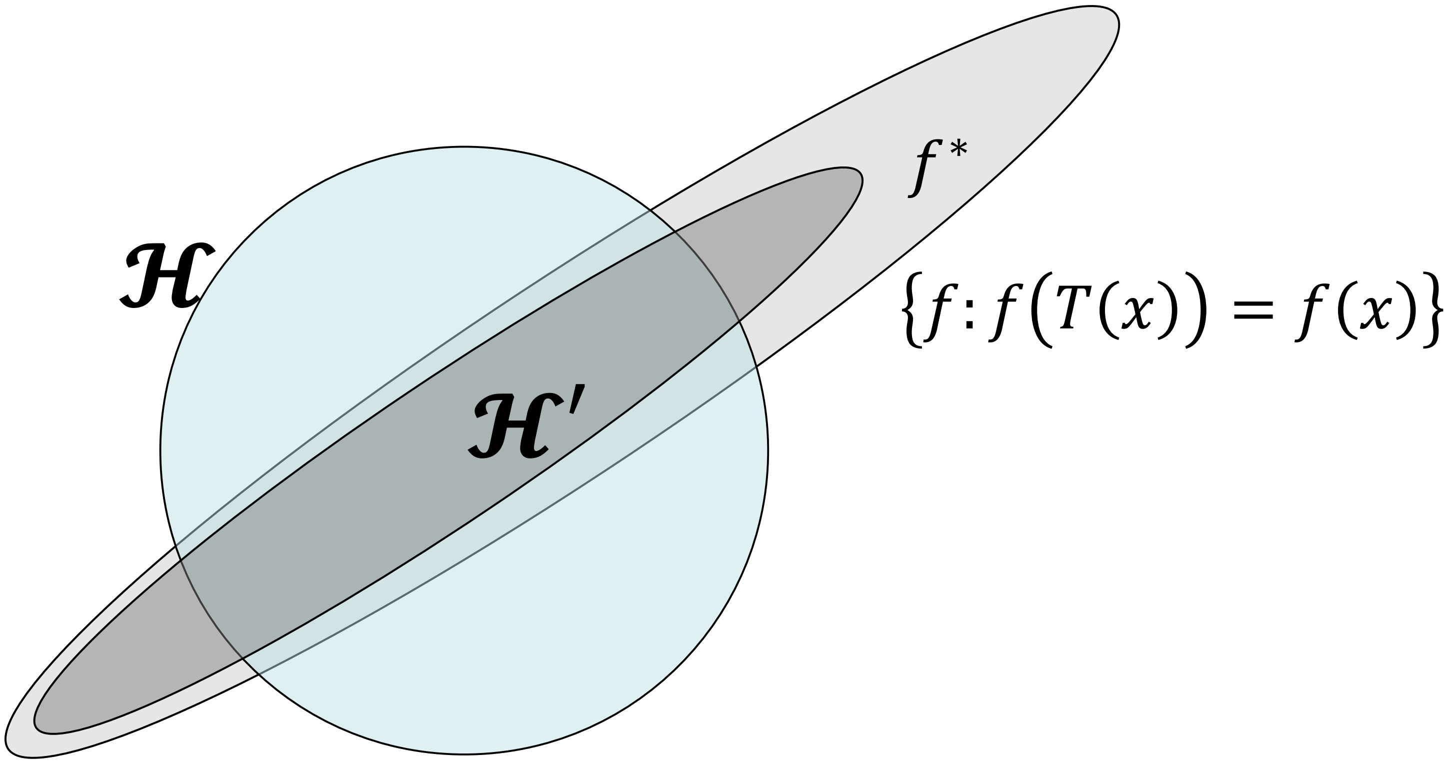
In other words, we must have

$$f^*\big(T(I)\big) = f^*(I)$$

for this classification problem!

Using a CNN to solve this problem places precisely this **prior**

Infinitely Strong Priors



Pooling Layers

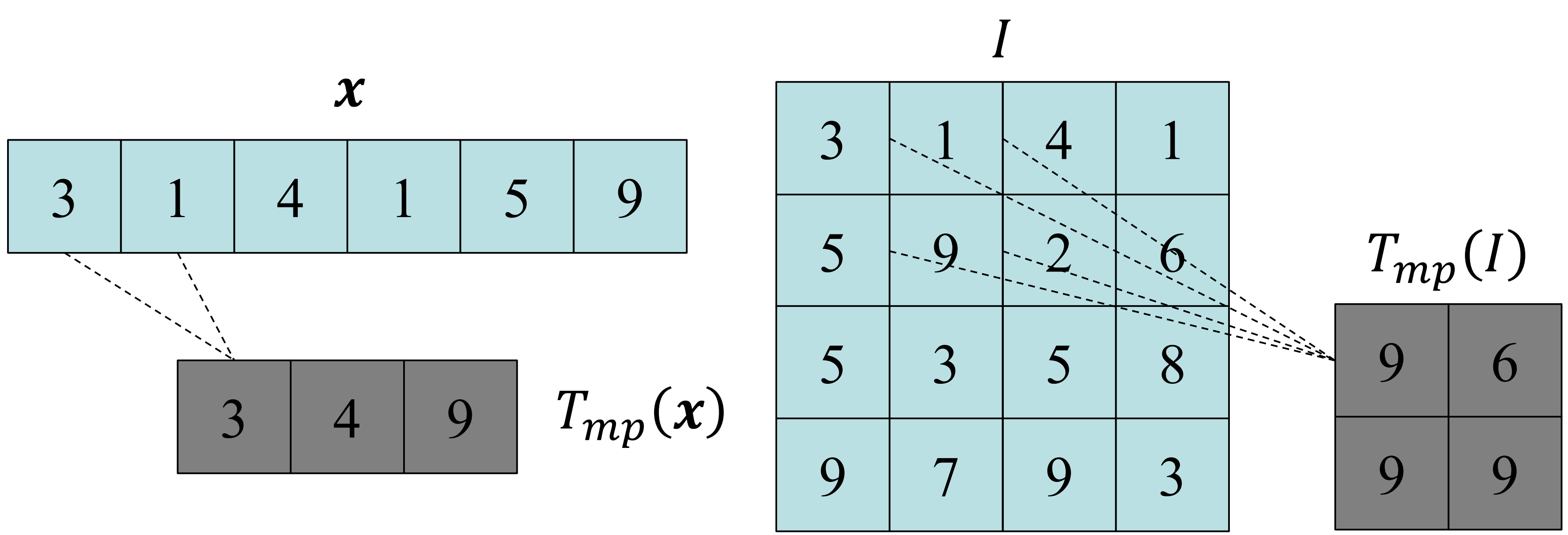
Pooling is another type of operation that builds approximate invariance to small translations/deformations

Simplest form is max pooling, but there are also other types, e.g. average pooling

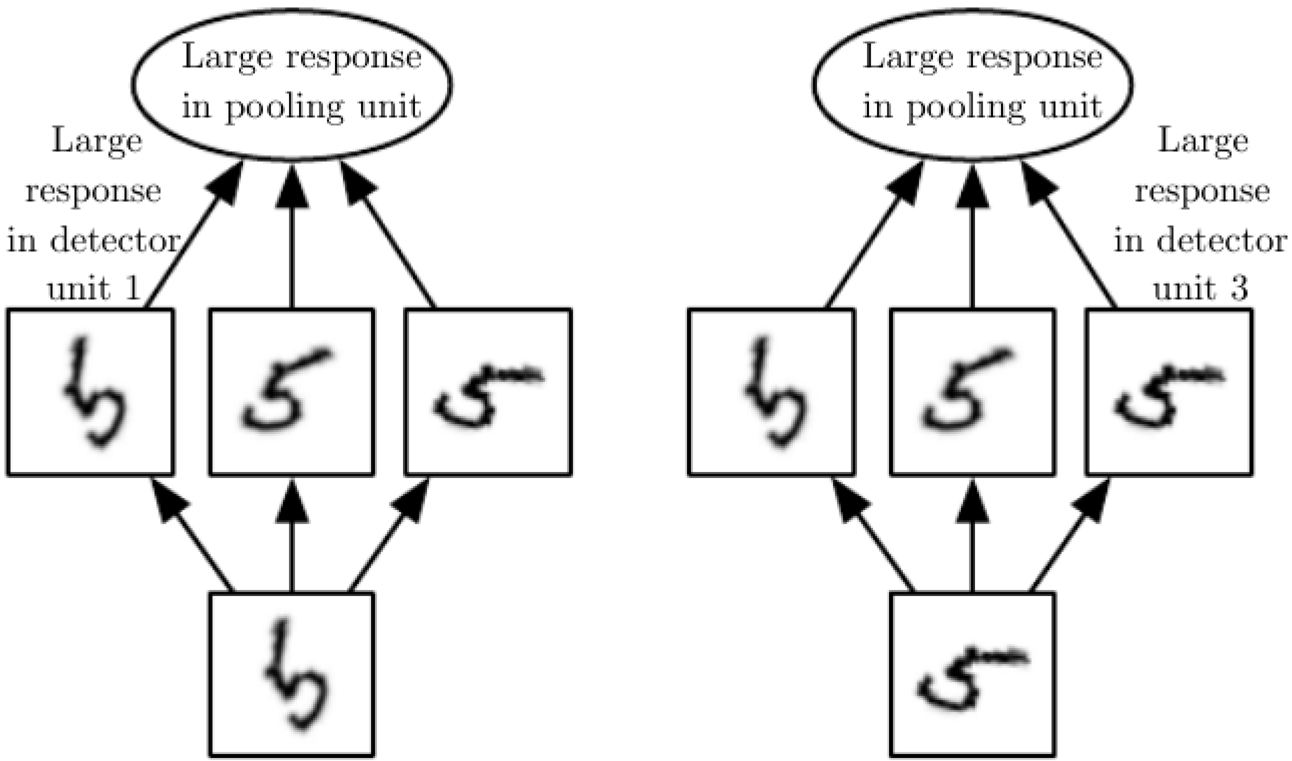
Example: Max Pooling Layers

Max pooling In 1D with stride p:

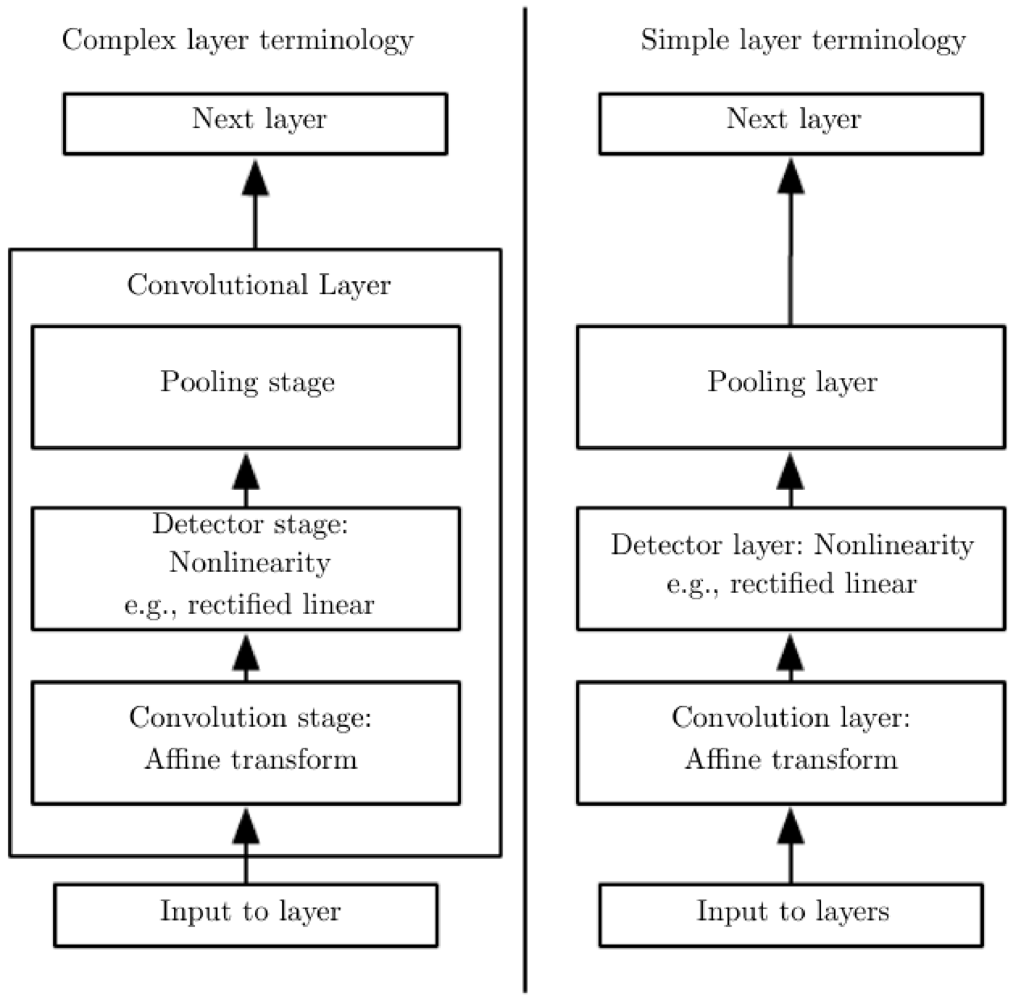
$$T_{mp}(\boldsymbol{x})_k = \max_{i=kp,\ldots,(k+1)p} x_i$$



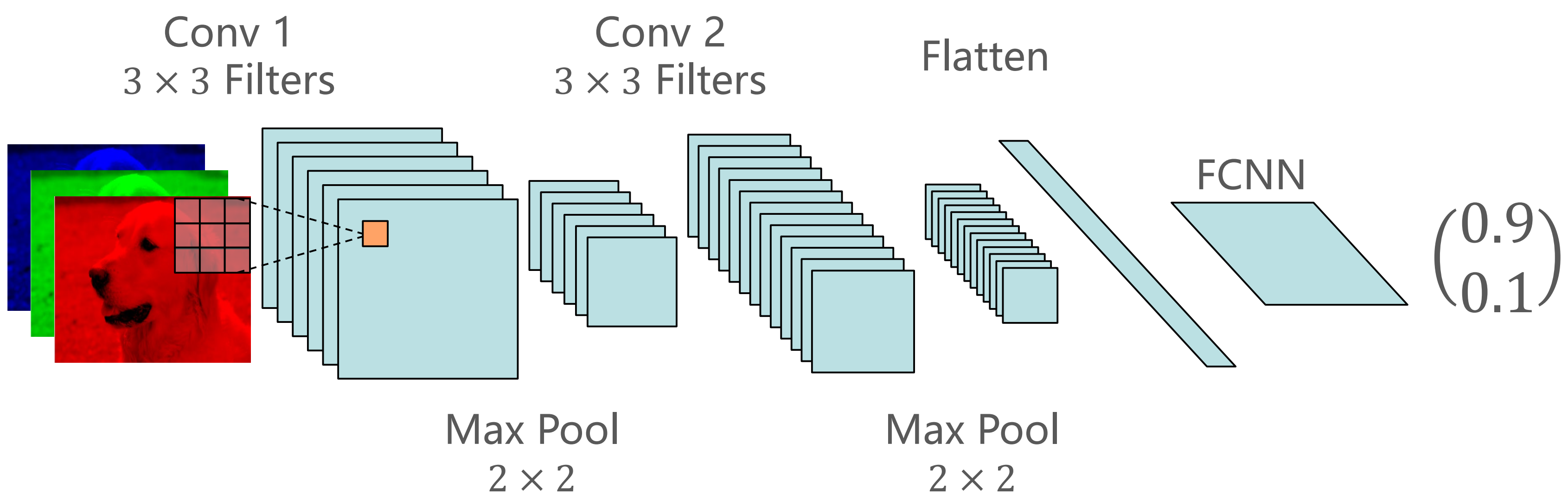
Learning Invariances with Pooling



The Basic Structure of CNNs



The Basic Structure of CNNs



Variants of Convolutions

There are many variants to the basic conv and pooling layers introduced so far, including:

- Strided convolutions
- Up and down-sampling operations
- Unshared convolutions
- Channel-wise/separable convolutions

Refer to Chapter 9 in the deep learning book for more details

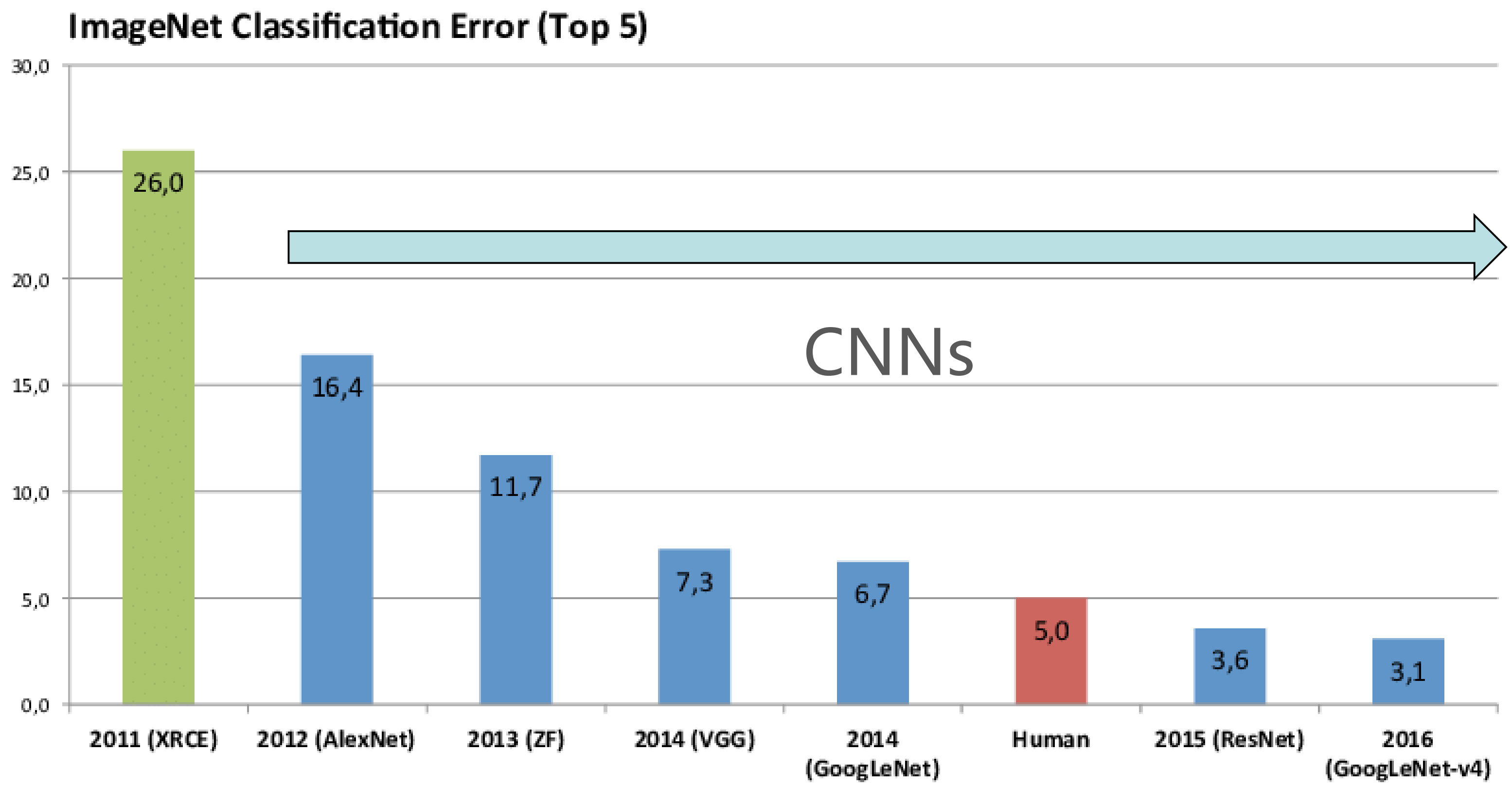
Historical notes on CNN

The ImageNet Challenge

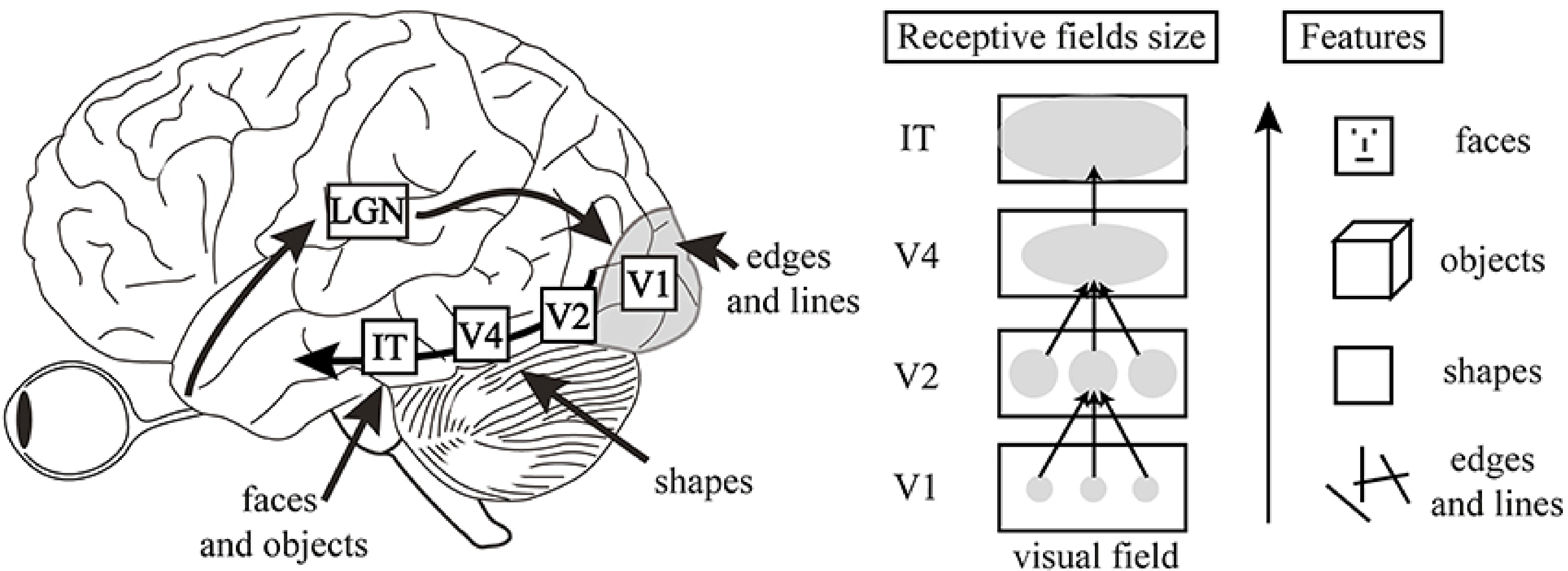
A large benchmark problem consisting of ~1M images, 20k categories

http://www.image-net.org/

History of CNNs



Neuroscience Basis of CNNs (Primary Visual Cortex)



https://neurdiness.wordpress.com/2018/05/17/deep-convolutional-neural-networks-as-models-of-the-visual-system-qa/

Demo: CNNs for MNIST and Pneumonia Classification

Summary

In this lecture, we introduced

- Limitations of FCNNs
- Convolution as a replacement for general linear transformations
- Motivations of using convolutions for image processing
- CNN based on convolution (and pooling)