Title: PSI 2018/2019 - Machine Learning - Lecture 6

Speakers: Lauren Hayward Sierens

Collection: PSI 2018/2019 - Machine Learning (Hayward Sierens)

Date: April 01, 2019 - 9:00 AM

URL: http://pirsa.org/19040001

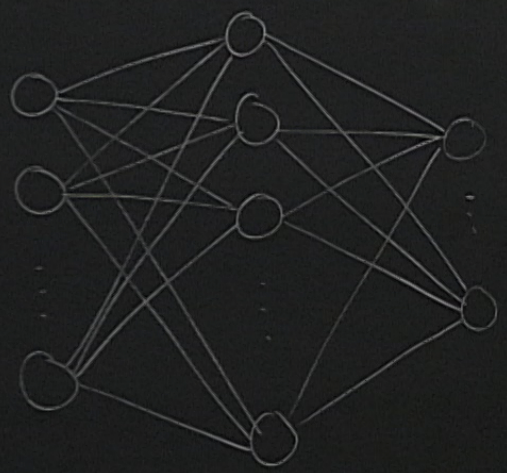
Today: Convolutional Neural Networks (CNNs)

↳ local receptive fields

↳ weight and bias sharing

↳ coarse-graining through "pooling"

Last week: fully-connected feedforward NNs



every neuron in layer ℓ is connected to every neuron in layer $\ell+1$

cted feedforward NNs

every neuron in layer l is connected to every neuron in layer $l+1$

Issue: these NNs do not exploit properties such as locality and translational invariace

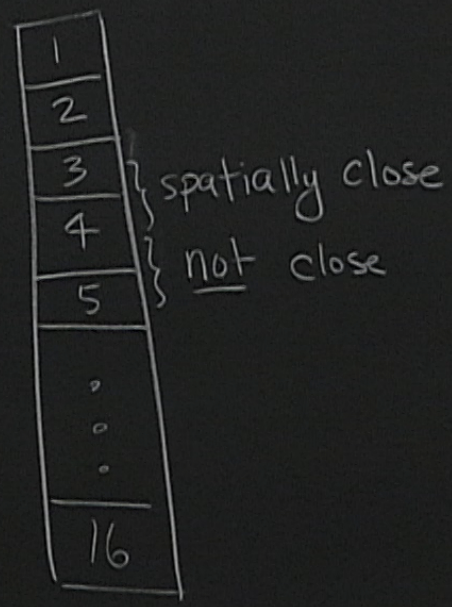
$\Rightarrow$ spatial information is lost or must be learned

Consider a 4x4 image (picture of a dog, Ising spin config., etc.)

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

f a dog, Ising

Flatten when taken as input to a FFNN:



locality is easily lost!

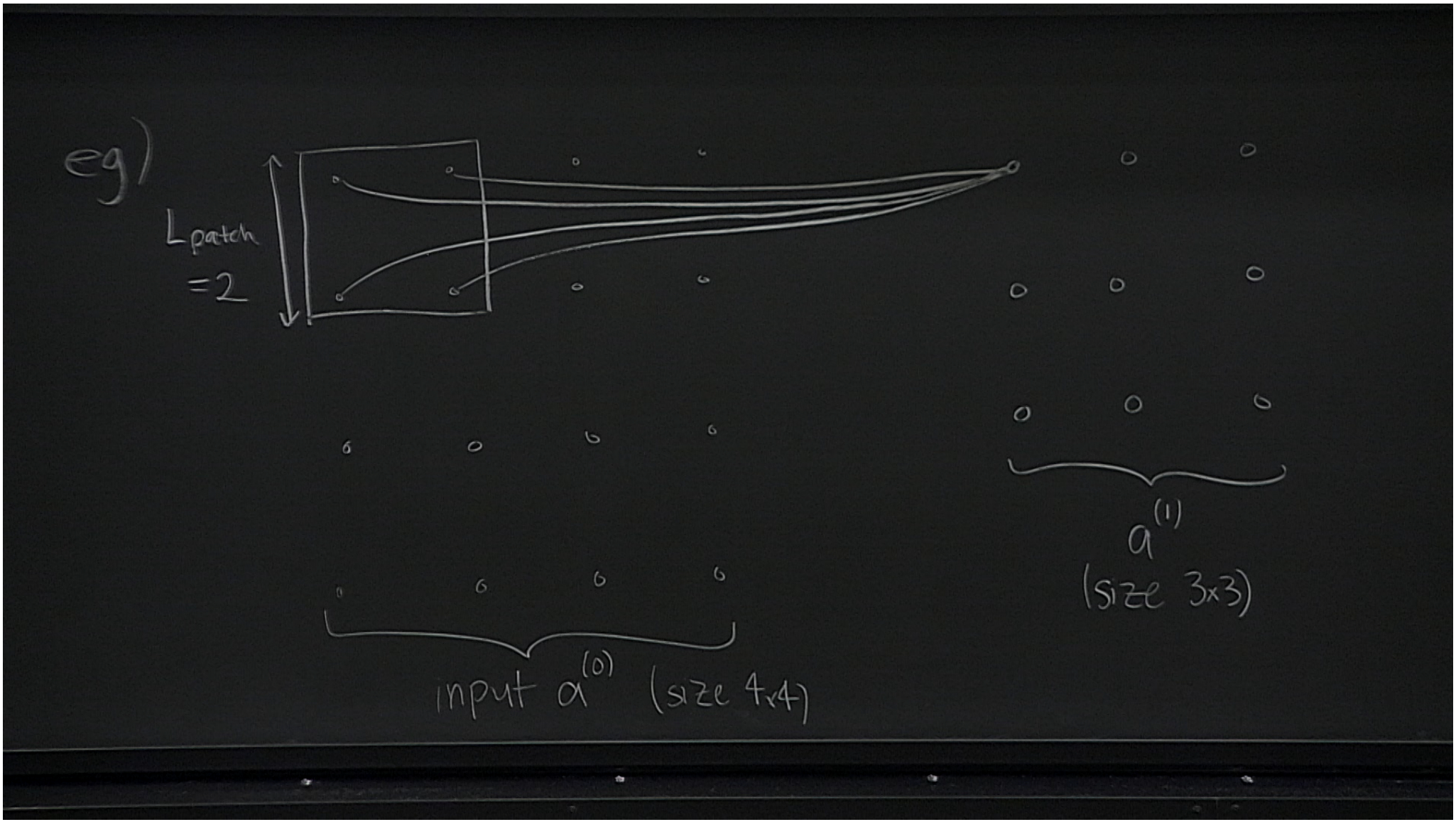
is

lost!

Solution:

Local Receptive Fields (LRFs)

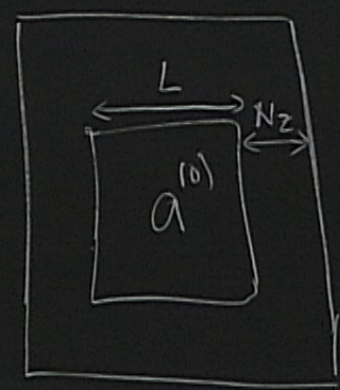
Idea: process the data in local "patches" that preserve the inherent dimensional structure


eg)
$L_{patch} = 2$
$a^{(1)}$
(size 3x3)
input $a^{(0)}$ (size 4x4)

Generalizations:

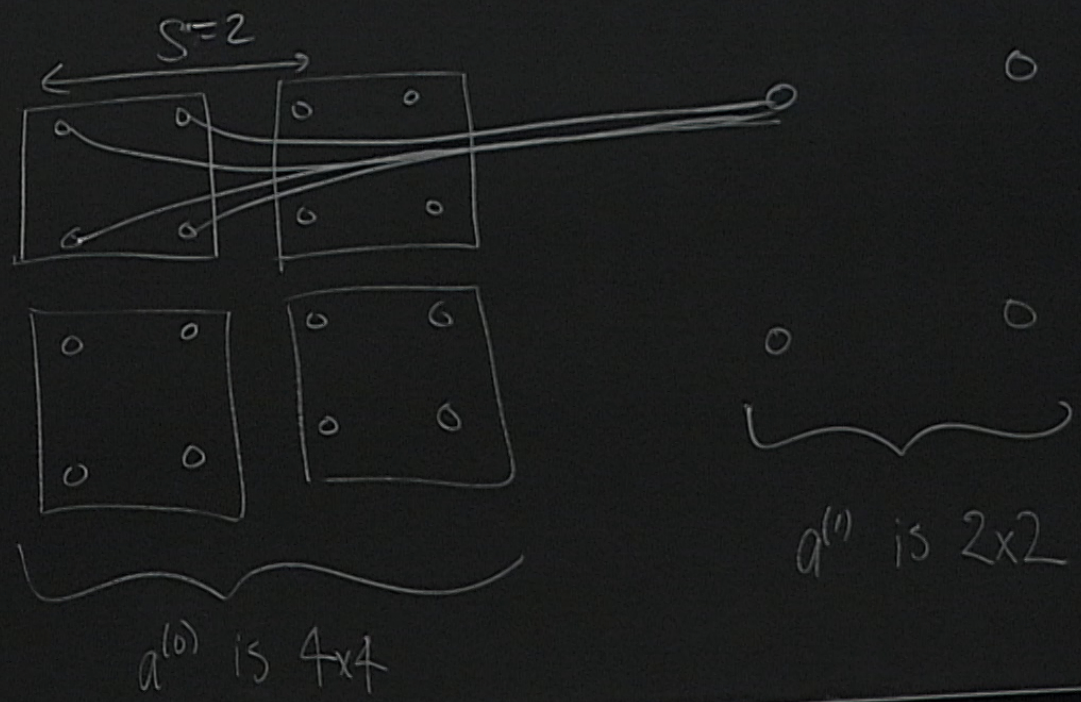
- **patch size**: can be a square of size $L_{patch} \times L_{patch}$ or a more general shape

- **padding**: we can pad the input with N_z zeros around each edge



- <u>stride</u>: we can take a step of size S before applying the next patch

eg)



If $a^{(0)}$ has $n_0 \times n_0 = L \times L$ neurons, then

$a^{(1)}$ has $n_1 \times n_1 = L_1 \times L_1$ neurons, where $L_1 = \frac{L - L_{patch} + 2N_z}{S} + 1$

If we apply periodic boundary conditions instead of padding: $L_1 = \frac{L}{S}$

For images, in addition to the spatial coordinates we can have different "channels"

$$\text{input}: \quad a^{(0)}_{\underbrace{x,y}_{\text{spatial indices}},\, c}$$

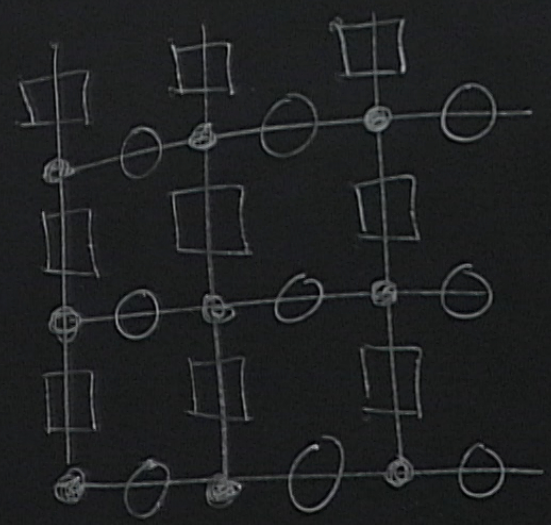
$c \to$ channel index

- for images: c can label R, G, B channels
- for lattice spin configs: c can label the sublattices

ordinates

each patch (filter) is then of size $L_{patch} \times L_{patch}$

eg) For the $\mathbb{Z}_2$ gauge theory



○ : sublattice 1

□ : sublattice 2

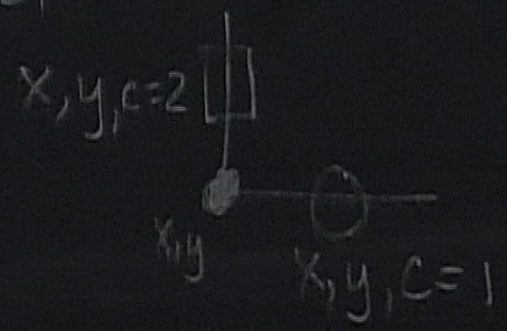
For
Valu

# of channels

For each x, y pair we consider two values of c

1

2



Shared weights and biases

Idea: the LRF moves across the input data (along x and y) and the weights & biases associated w/ the filters do not change with x and y.

$\Rightarrow$ the convolutional layer applies a translationally inv. transformation

Compare with the Ising model $H = -J \sum_{\langle ij \rangle} S_i S_j$

same coupling for all i, j

At layer ℓ, we can apply N_f different filters of size $L_{patch} \times L_{patch} \times N_c$ to the output $a^{(\ell-1)}$ from layer $\ell-1$

Such that

$$a^{(\ell)}_{x,y,k} = \underbrace{g_\ell}_{\text{activation func.}}\left(\underbrace{\sum_{m=1}^{L_{patch}} \sum_{n=1}^{L_{patch}} \sum_{c=1}^{N_c} a^{(\ell-1)}_{x+m-1,\, y+n-1,\, c}\, W^{(\ell)}_{mnsk}}_{\text{convolution}} + \underbrace{b^{(\ell)}_k}_{\text{bias}} \right)$$

different filters of size

utput $a^{(\ell-1)}$ from layer $\ell-1$

$$\underbrace{\sum_{n=1}^{L_{patch}} \sum_{c=1}^{N_c} a^{(\ell-1)}_{x+m-1,\, y+n-1,\, c}\, W^{(\ell)}_{mnck}}_{\text{convolution}} + \underbrace{b_k^{(\ell)}}_{\text{bias}} \Bigg)$$

where

where the weight indices are:

$$W_{m\,n\,c\,k}$$

- m → sliding along x
- n → sliding along y
- c → channel/sublattice index
- k → labels the different filters $(1 \leq k \leq N_f)$

where the weight indices are:

$$W_{m\,n\,c\,k} \qquad (L_{patch} \times L_{patch} \times N_c \times N_f)$$

- m: sliding along x
- n: sliding along y
- c: channel/sublattice index
- k: labels the different filters $(1 \leq k \leq N_f)$

At layer ℓ, we can apply N_f different filters of size $L_{patch} \times L_{patch} \times N_c$ to the output $a^{(\ell-1)}$ from layer

such that

$$a^{(\ell)}_{x,y,k} = \underbrace{g_\ell}_{\text{activation func.}}\Bigg(\underbrace{\sum_{m=1}^{L_{patch}} \sum_{n=1}^{L_{patch}} \sum_{c=1}^{N_c} a^{(\ell-1)}_{x+m-1,\, y+n-1,\, c}}_{\text{convolution}} W$$

where the weight indices are:

$$W_{m\,n\,c\,k} \qquad (L_{patch} \times L_{patch} \times N_c \times N_f)$$

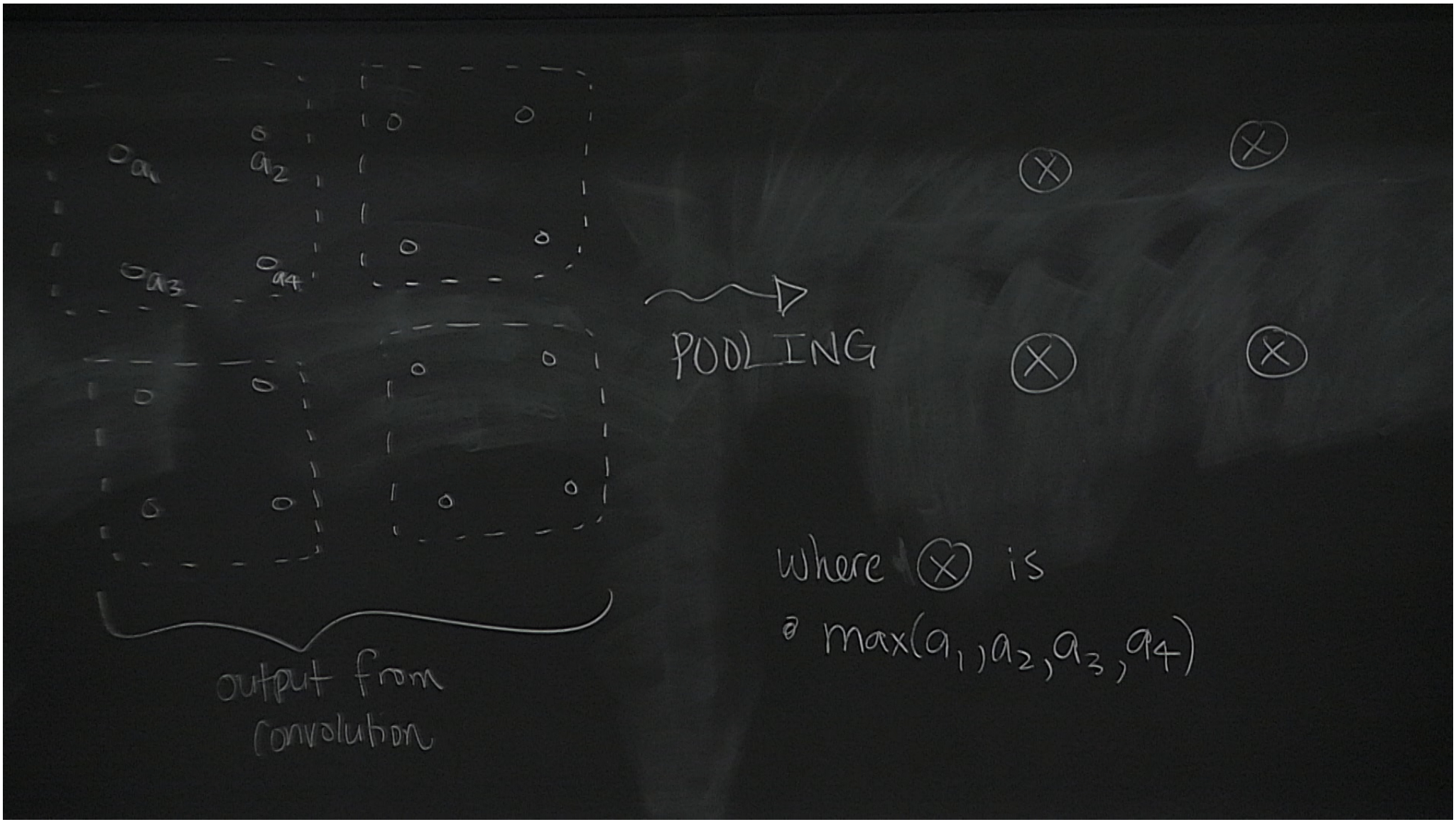
- m → sliding along x
- n → sliding along y
- c → channel/sublattice index
- k → labels the different filters $(1 \leq k \leq N_f)$

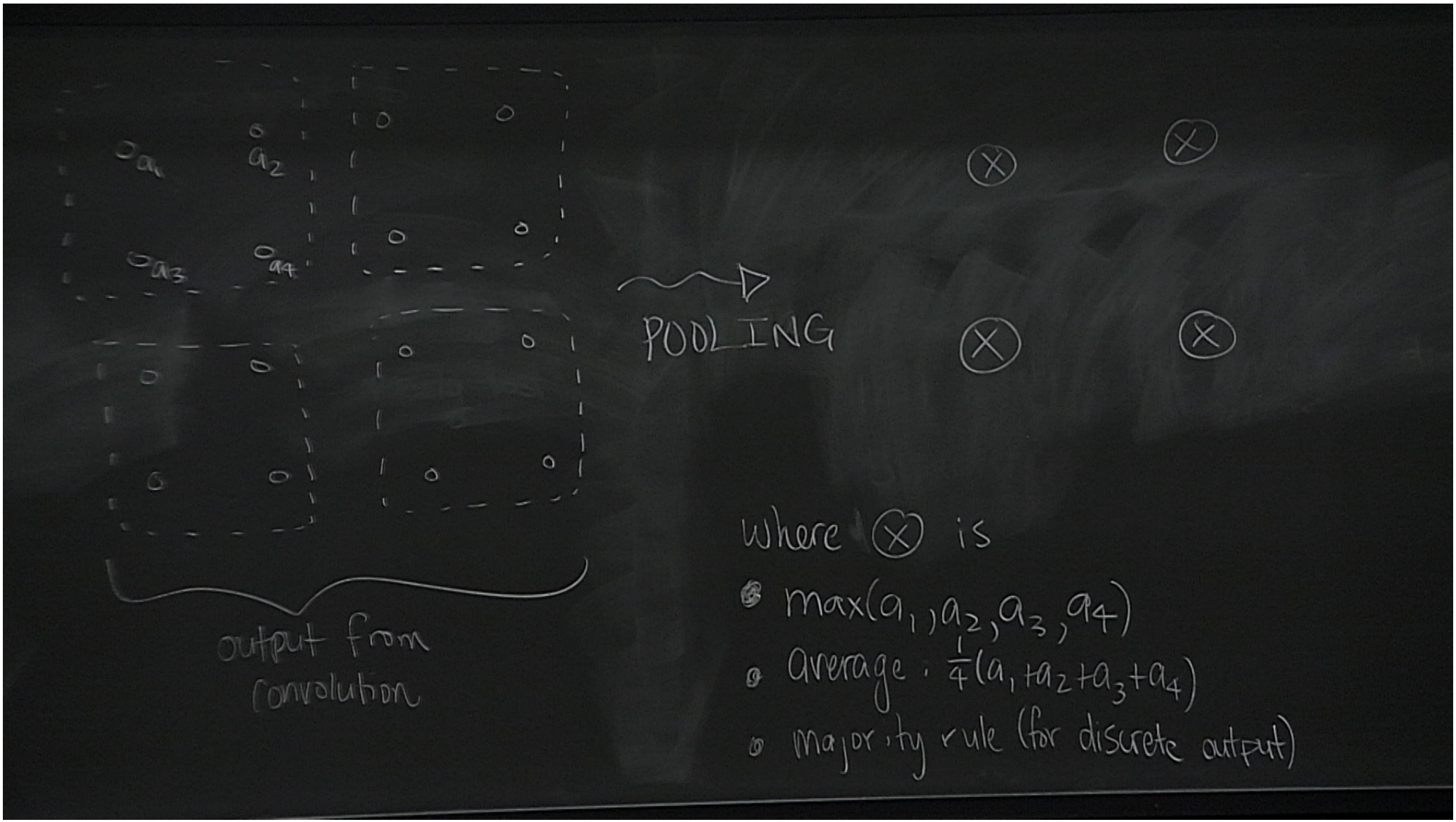
CNNs tend to have far fewer free params. to learn during training and can train well for larger inputs and deeper layers than FFNNs (assuming input is trans. inv. and local)

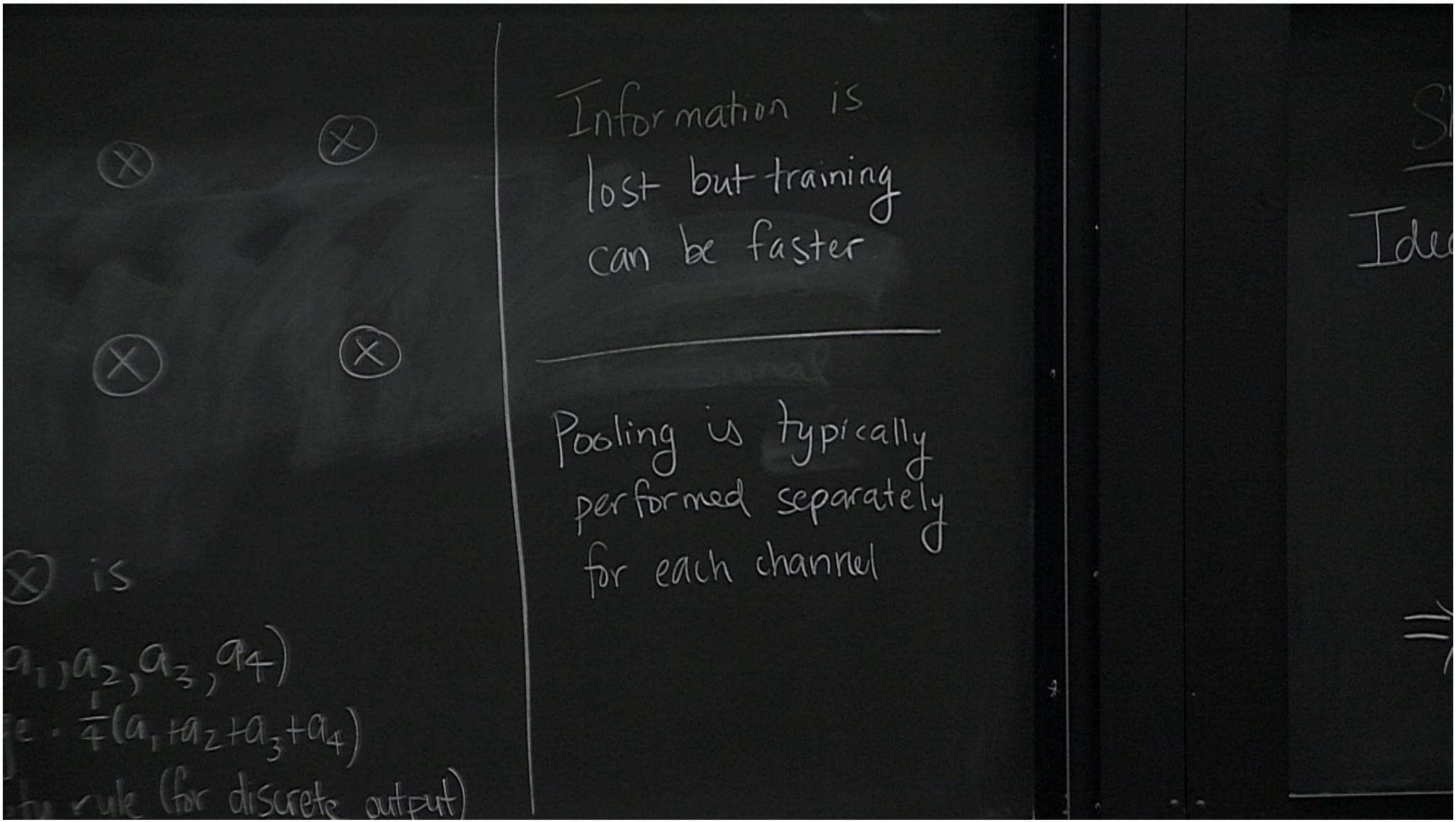
Pooling

Idea: perform a "coarse-graining" after the convolution while still preserving locality

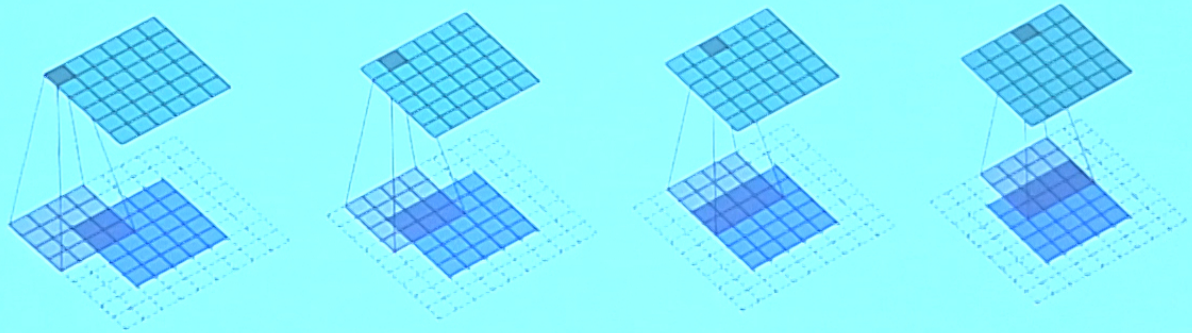
(especially useful for data with high correlations (redundant info.))


a_1
a_2
a_3
a_4
POOLING
output from convolution
where ⊗ is
$\max(a_1, a_2, a_3, a_4)$


a_1
a_2
a_3
a_4
POOLING
output from convolution
where ⊗ is
• $\max(a_1, a_2, a_3, a_4)$
• average : $\frac{1}{4}(a_1 + a_2 + a_3 + a_4)$
• majority rule (for discrete output)


Information is lost but training can be faster
Pooling is typically performed separately for each channel
is
$a_1, a_2, a_3, a_4)$
$\frac{1}{4}(a_1+a_2+a_3+a_4)$
rule (for discrete output)

Convolutional neural networks

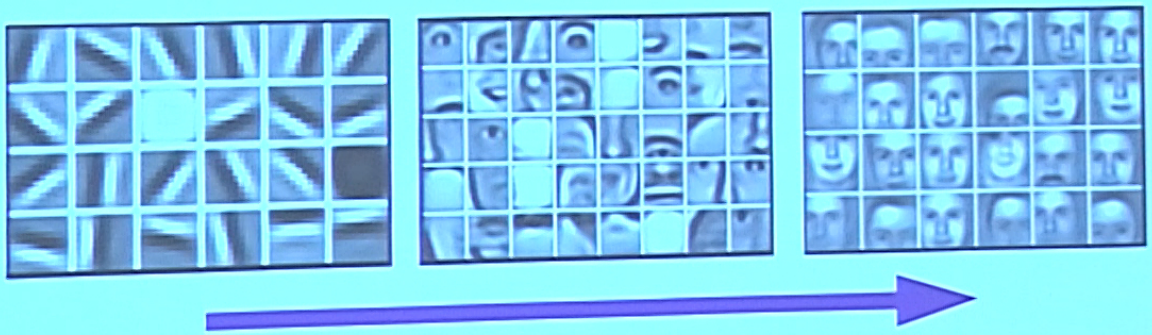


Dumoulin and Visin, arXiv:1603.07285

PERIMETER SCHOLARS INTERNATIONAL

Convolutional neural networks

A CNN tends to detect more and more complex features as we move through the layers from left to right:



https://killianlevacher.github.io/blog/posts/post-2016-03-01/img/layeredRepresentation.jpg

PERIMETER SCHOLARS INTERNATIONAL