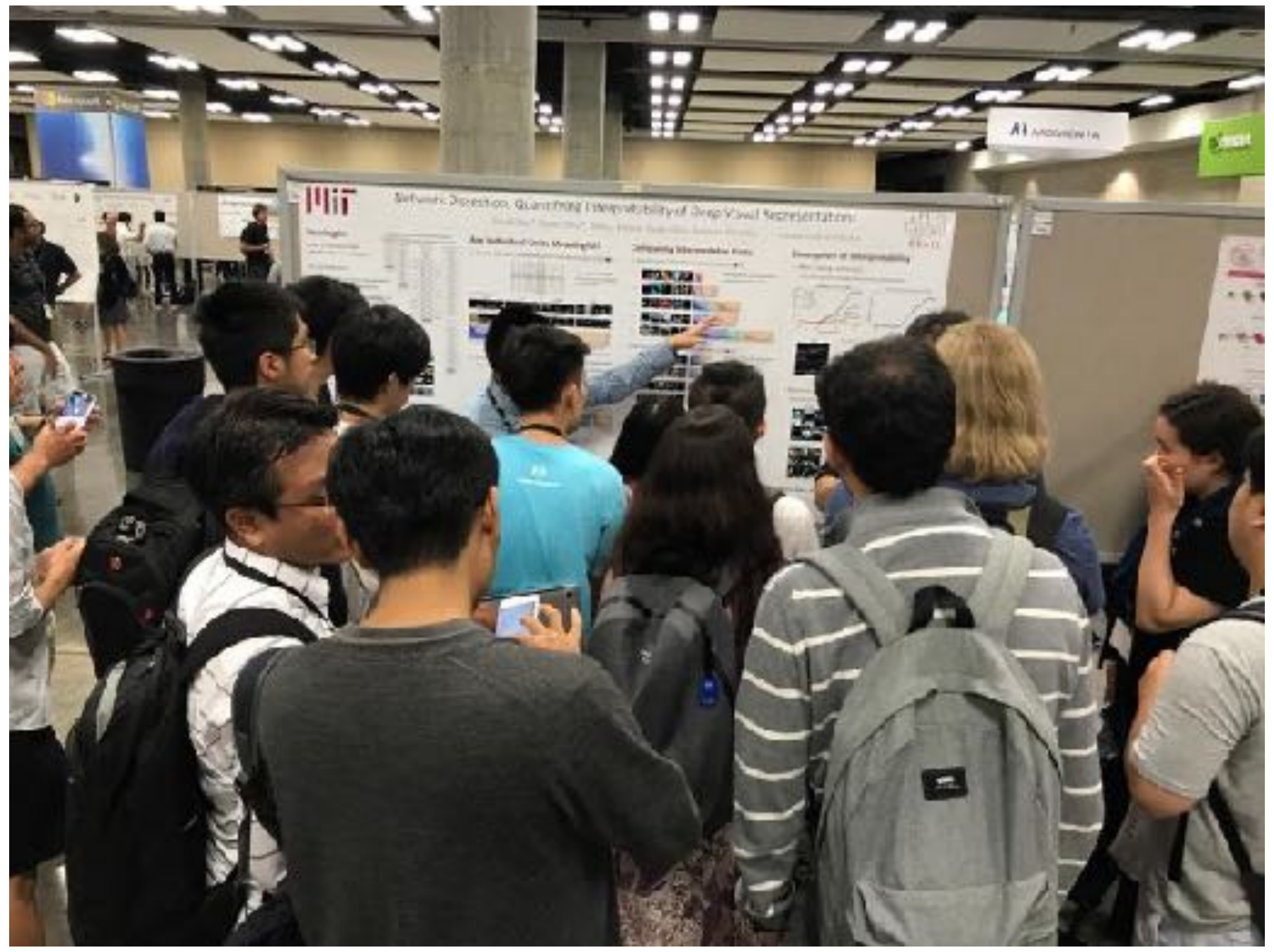
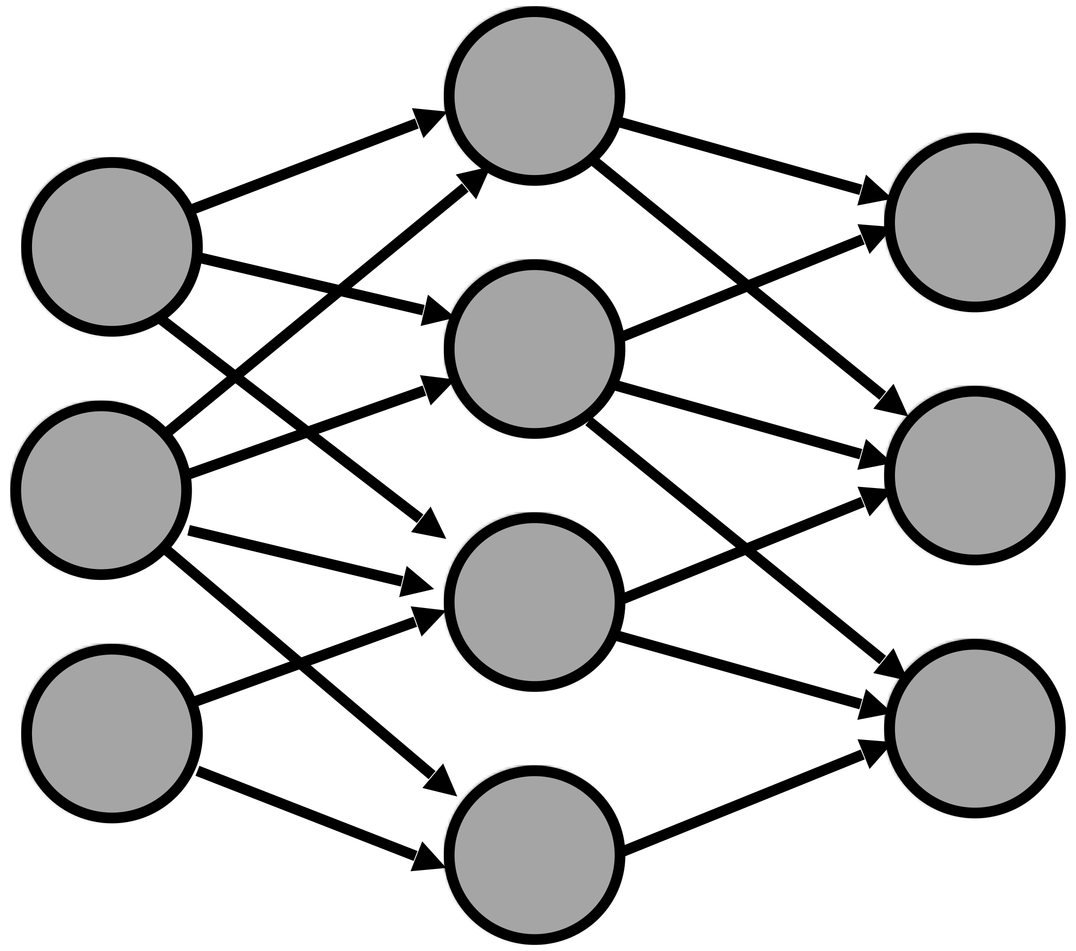




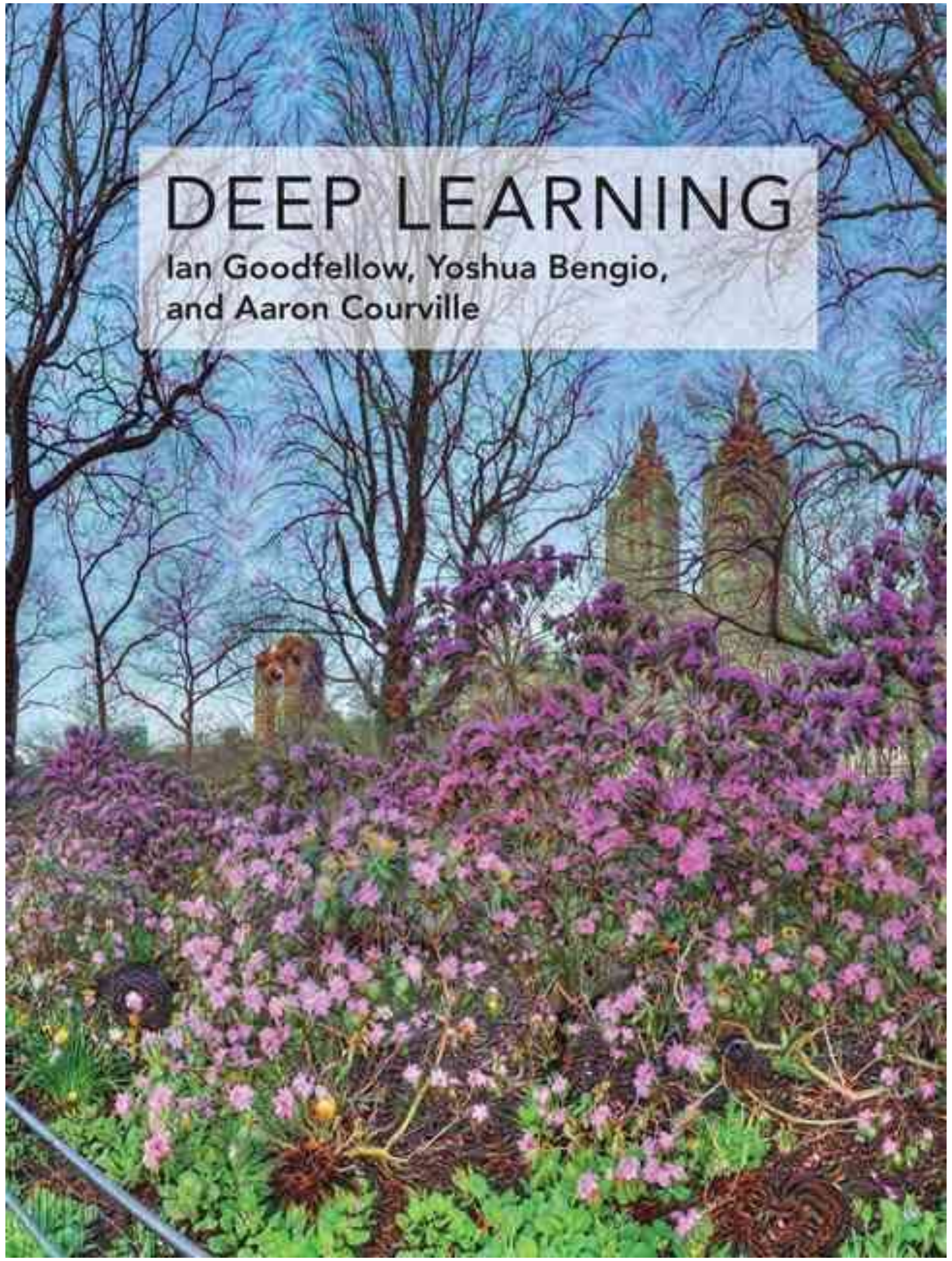
72% Indoor Booth

Introduction to Deep Learning

Slides are adapted from Bill Freeman, Antonio Torralba, Phillip Isola. 6.819 / 6.869

Announcement for the final exam

Final exam date: 07 May 2021 (Friday) 09:30 - 11:30 am:
Online final exam: self-arranged invigilation



http://www.deeplearningbook.org/

By Ian Goodfellow, Yoshua Bengio and Aaron Courville

November 2016

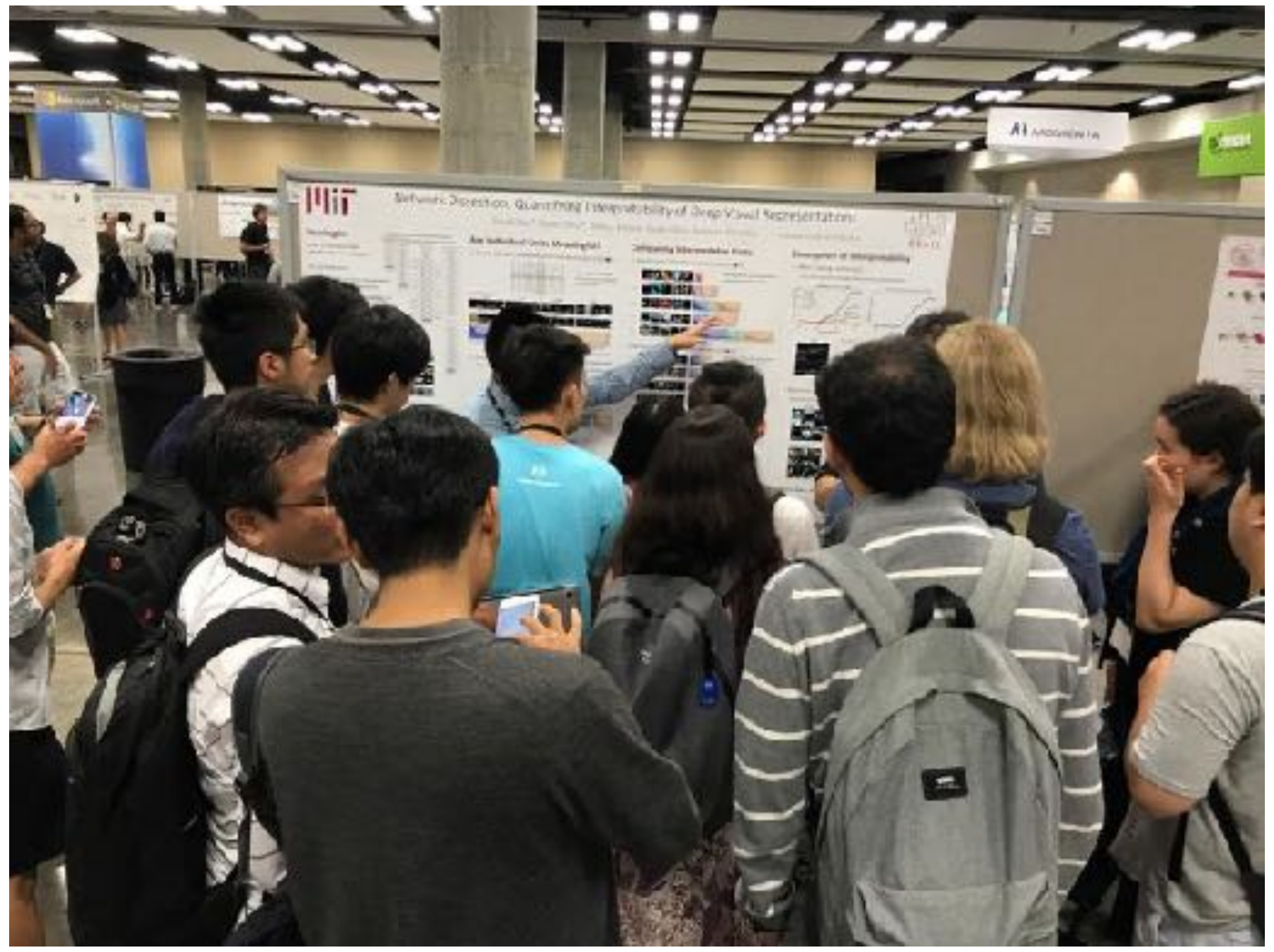
Parts of chapter 9

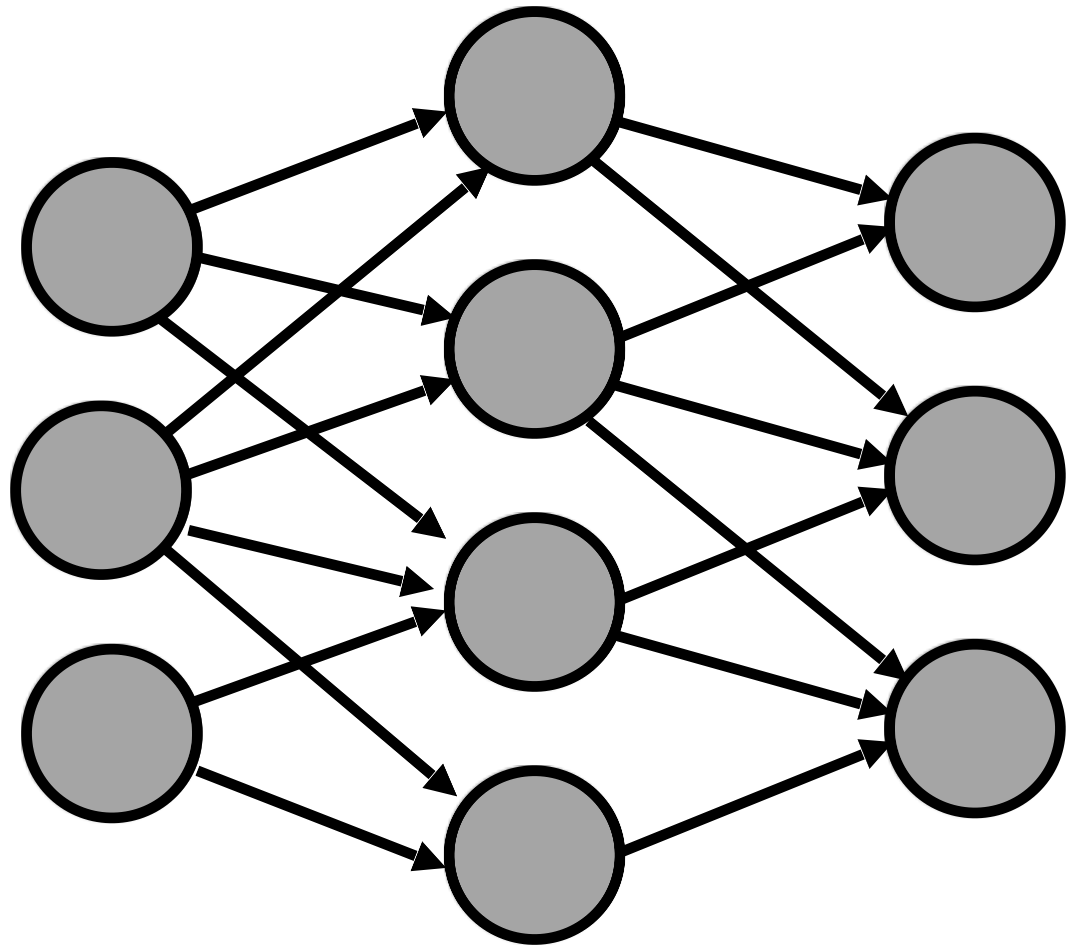
Review background on signal processing, convolution, — this is the technology that underlies convnets!

This Week: Into Deep Learning

- Scene classification: A live demo
- A brief overview of Convolutional Neural Networks (CNNs)
- Standard building blocks of CNNs
- Some important networks & their tricks
- **You are highly suggested to attend this week's TA Tutorial on learning PyTorch and how to use deep networks**

Image Classification





72% Indoor Booth

Try this!

http://places2.csail.mit.edu/demo.html

LeCun conv nets, 1998

PROC. OF THE IEEE, NOVEMBER 1998 7

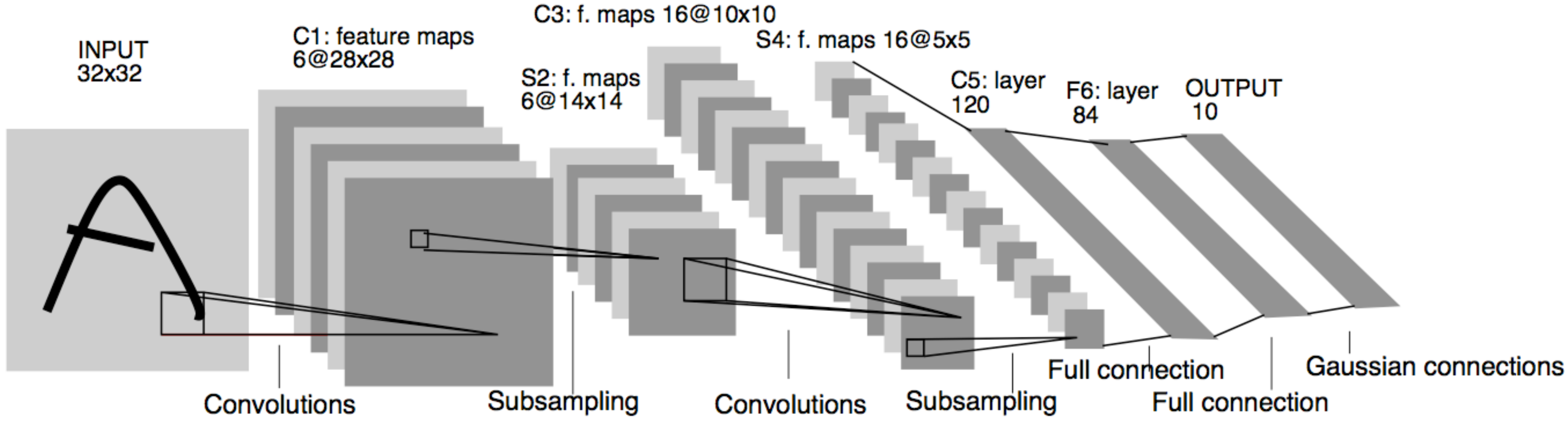


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Demos:
http://yann.lecun.com/exdb/lenet/index.html

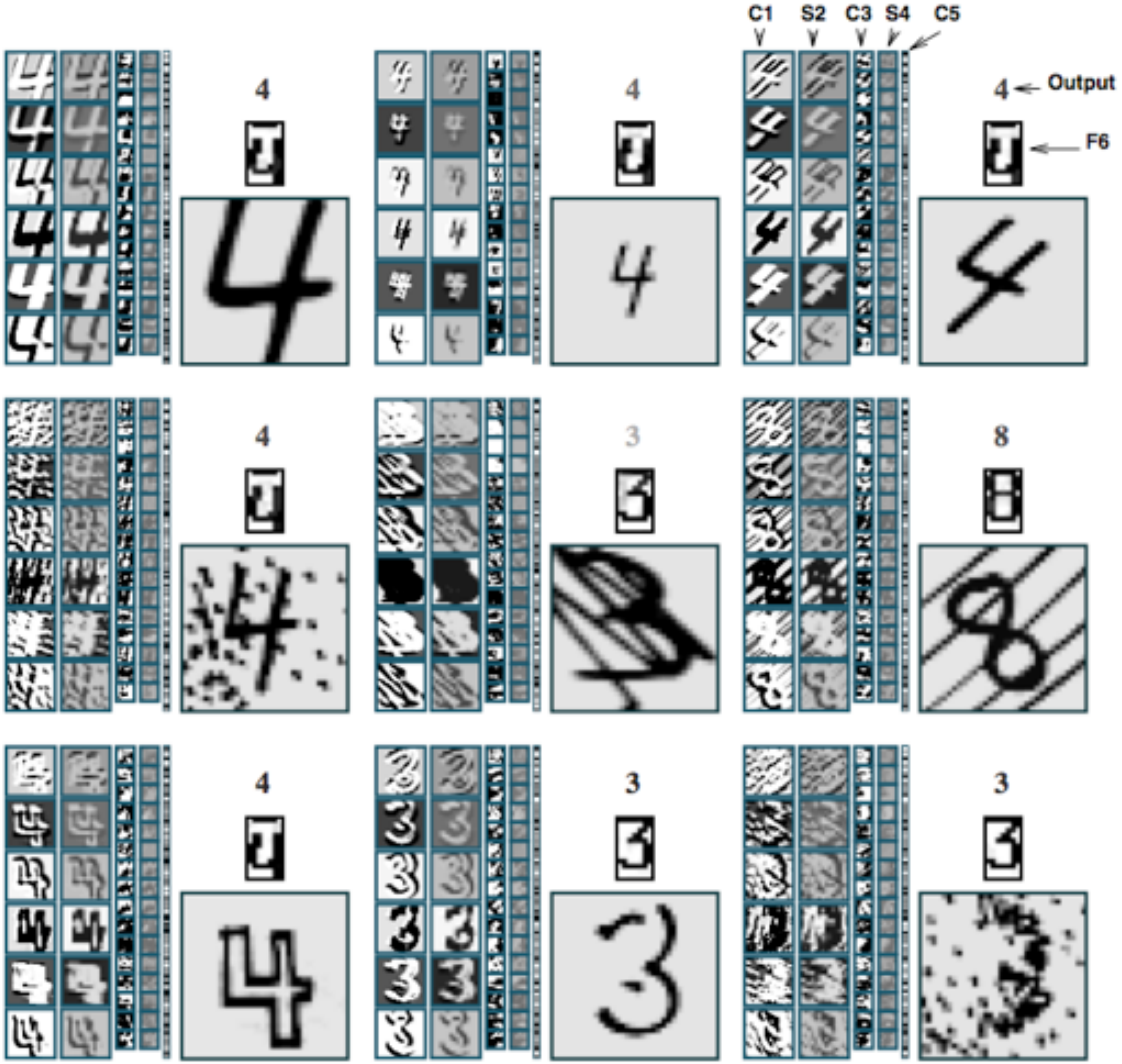
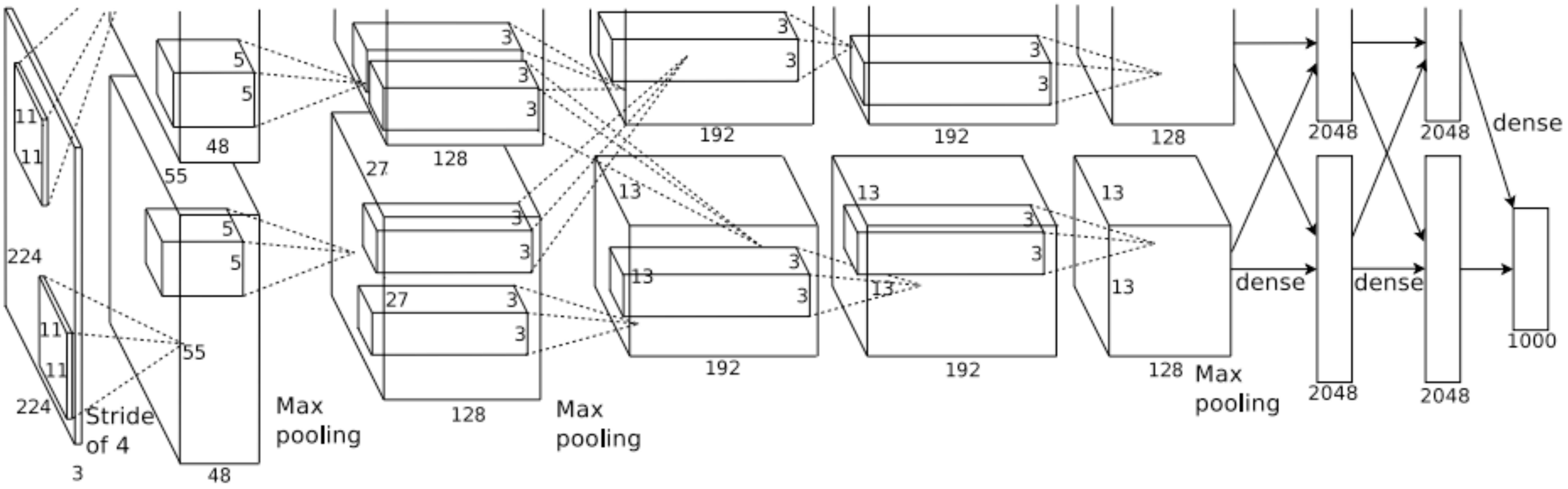


Fig. 13. Examples of unusual, distorted, and noisy characters correctly recognized by LeNet-5. The grey-level of the output label represents the penalty (lighter for higher penalties).

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

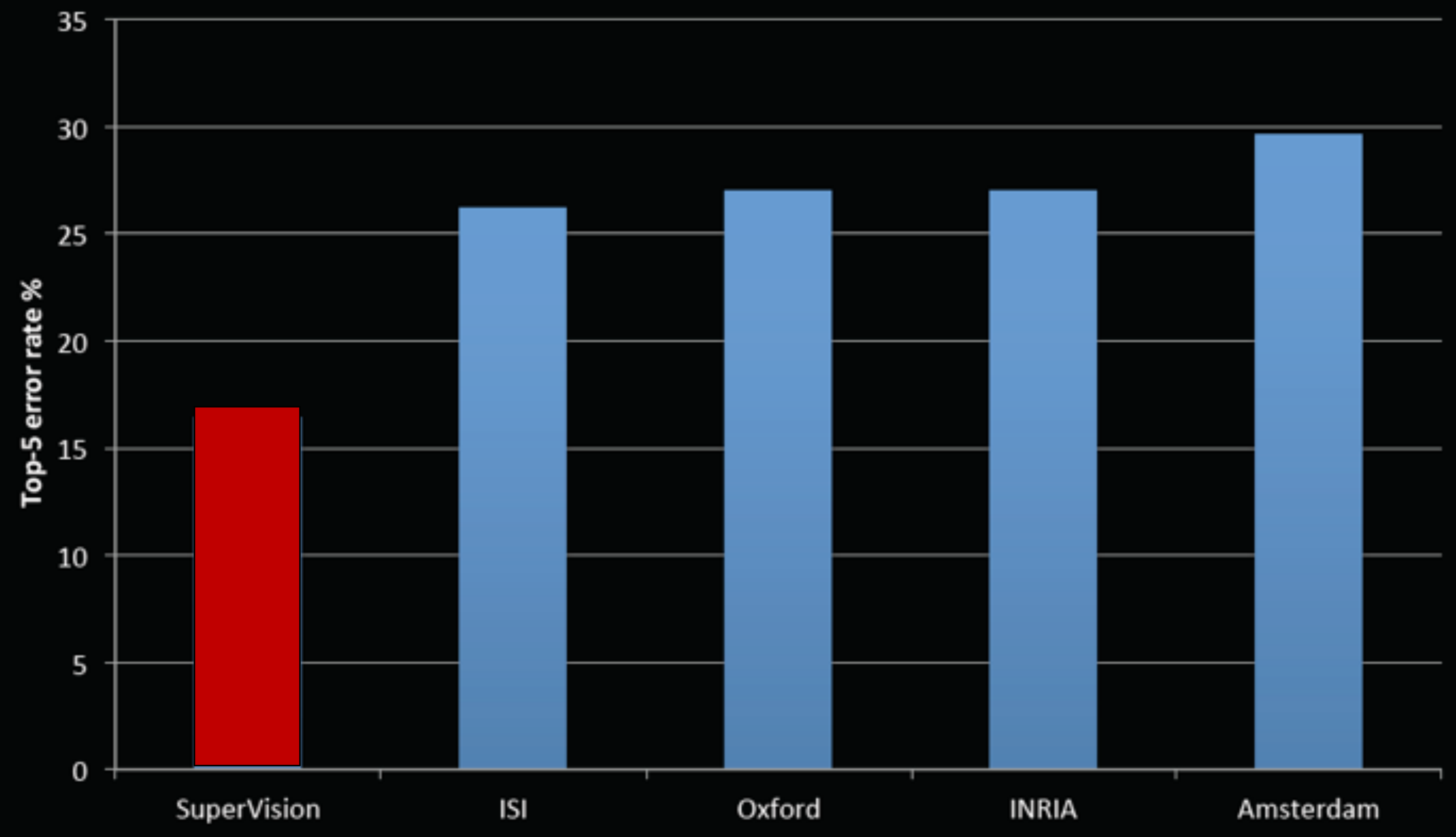
“Alexnet”



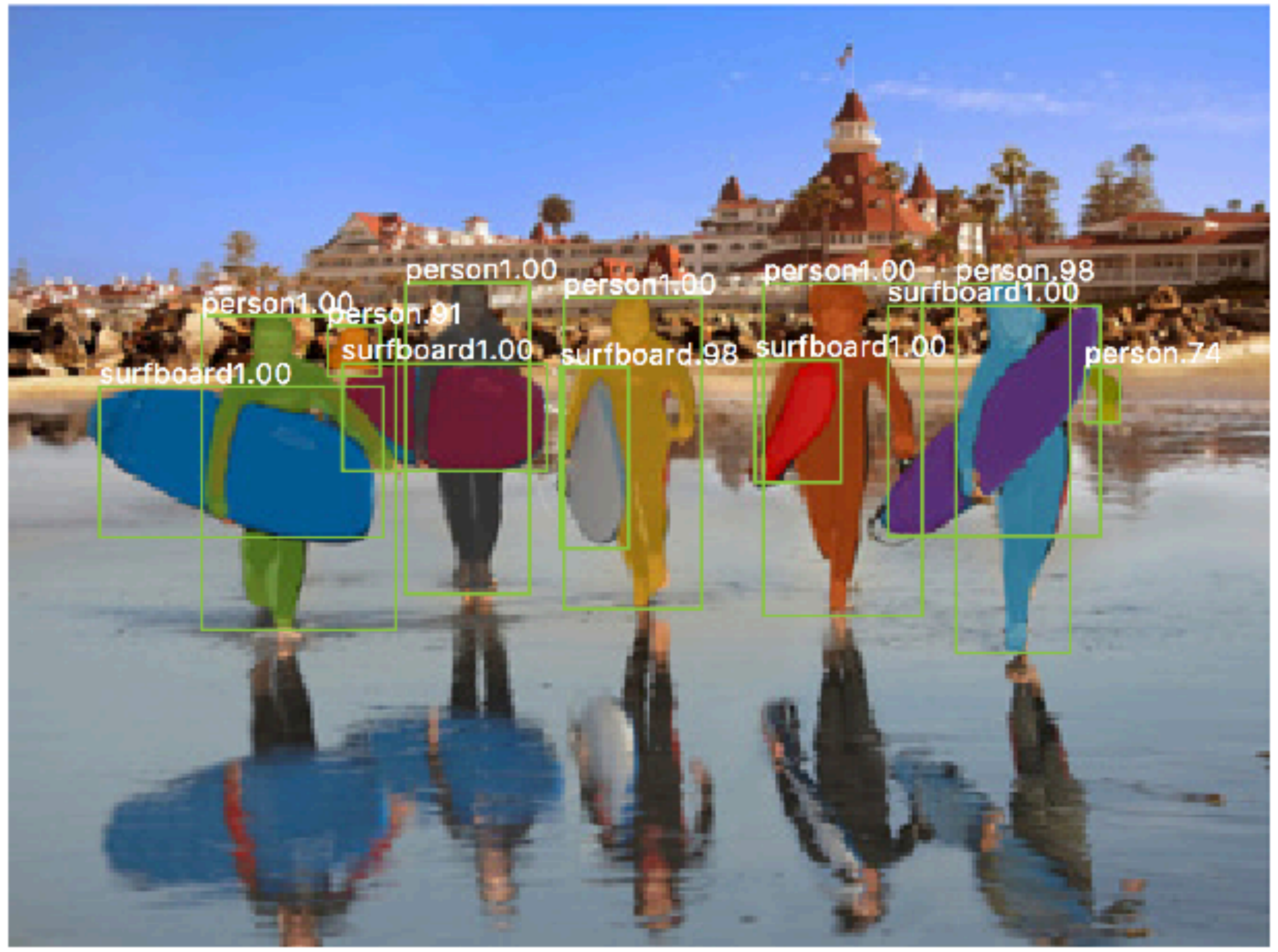
Slide from Rob Fergus, NYU

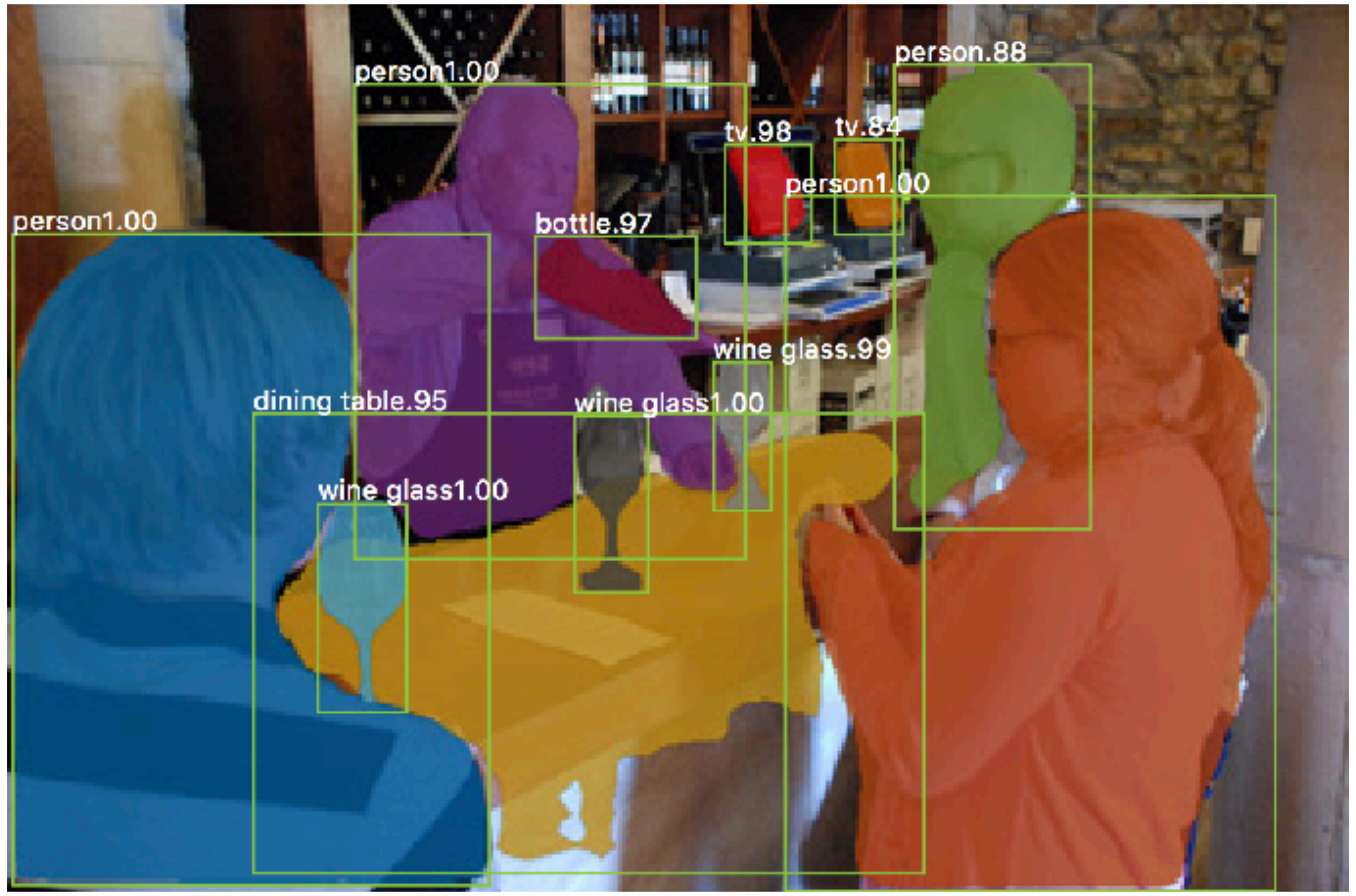
ImageNet Classification 2012

- Krizhevsky et al. -- 16.4% error (top-5)
- Next best (non-convnet) – 26.2% error



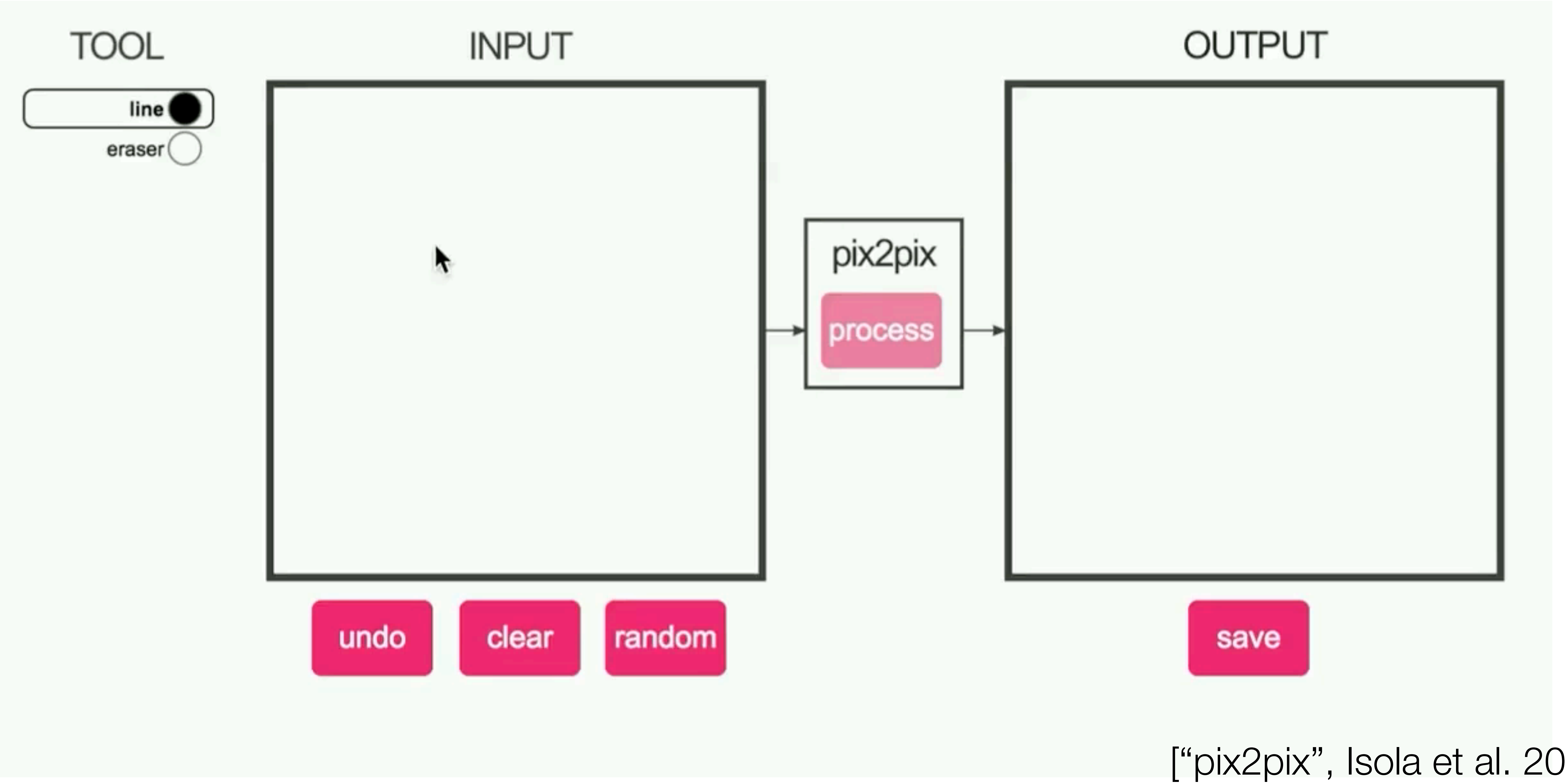
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012



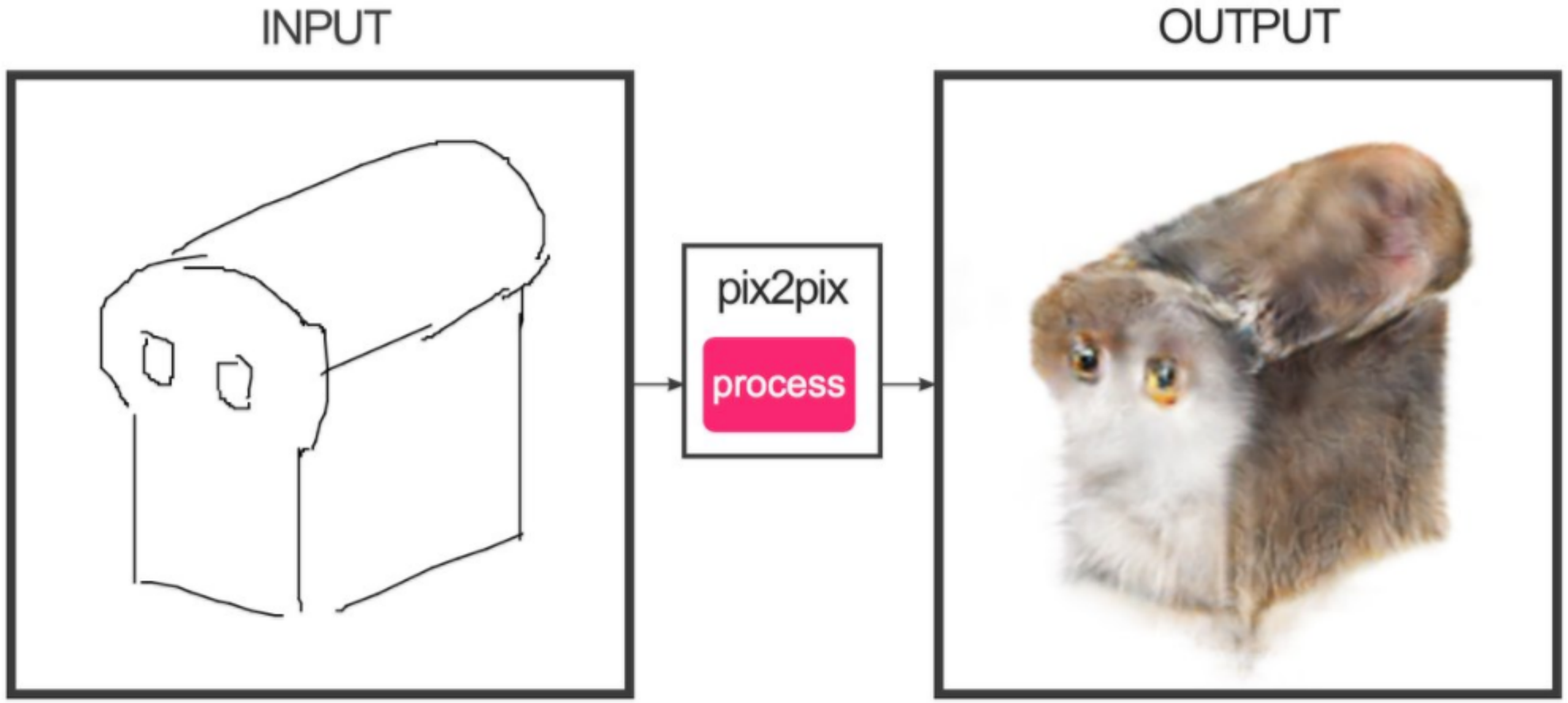


[“Mask RCNN”, He et al. 2017]

#edges2cats [Chris Hesse]



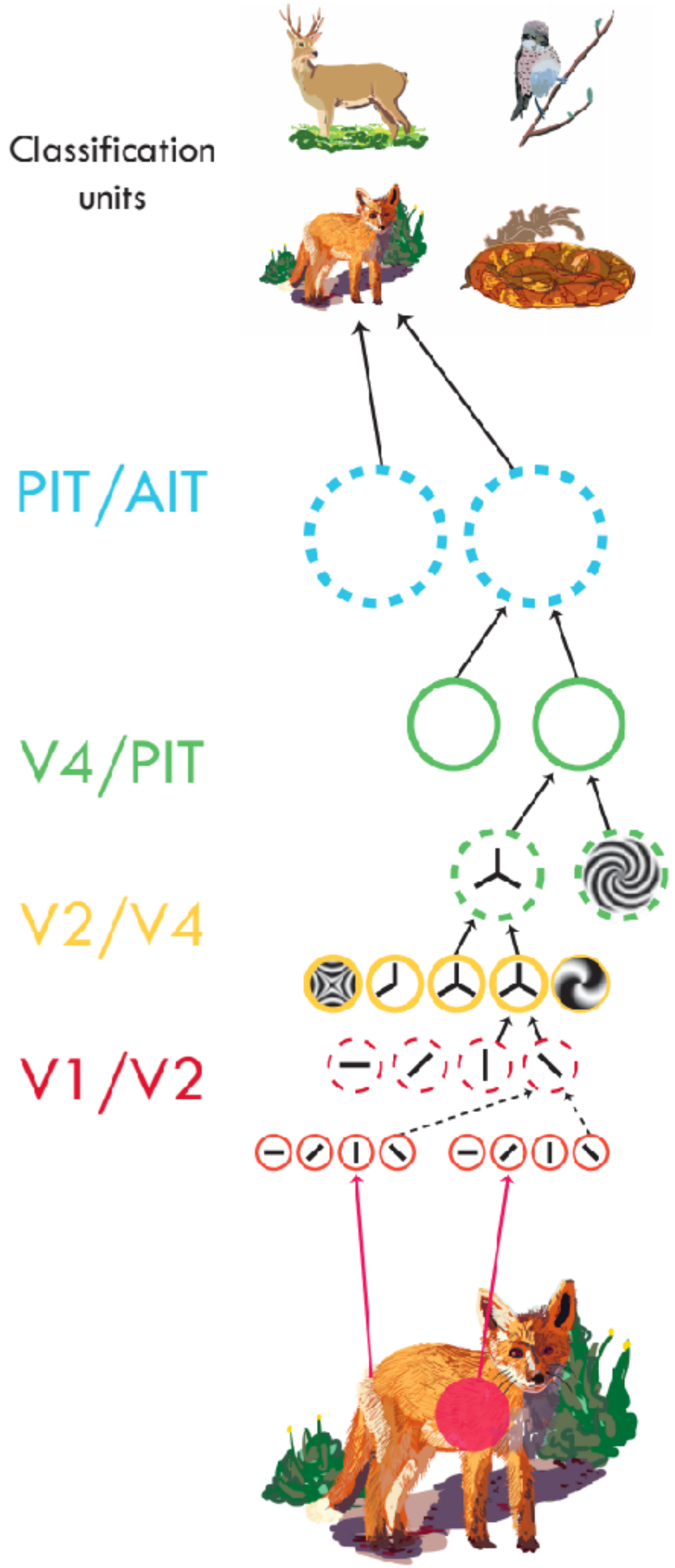
["pix2pix", Isola et al. 2017]



Ivy Tasi @ivymyt

Vitaly Vidmirov @vvid

["pix2pix", Isola et al. 2017]



Serre, 2014

Image recognition

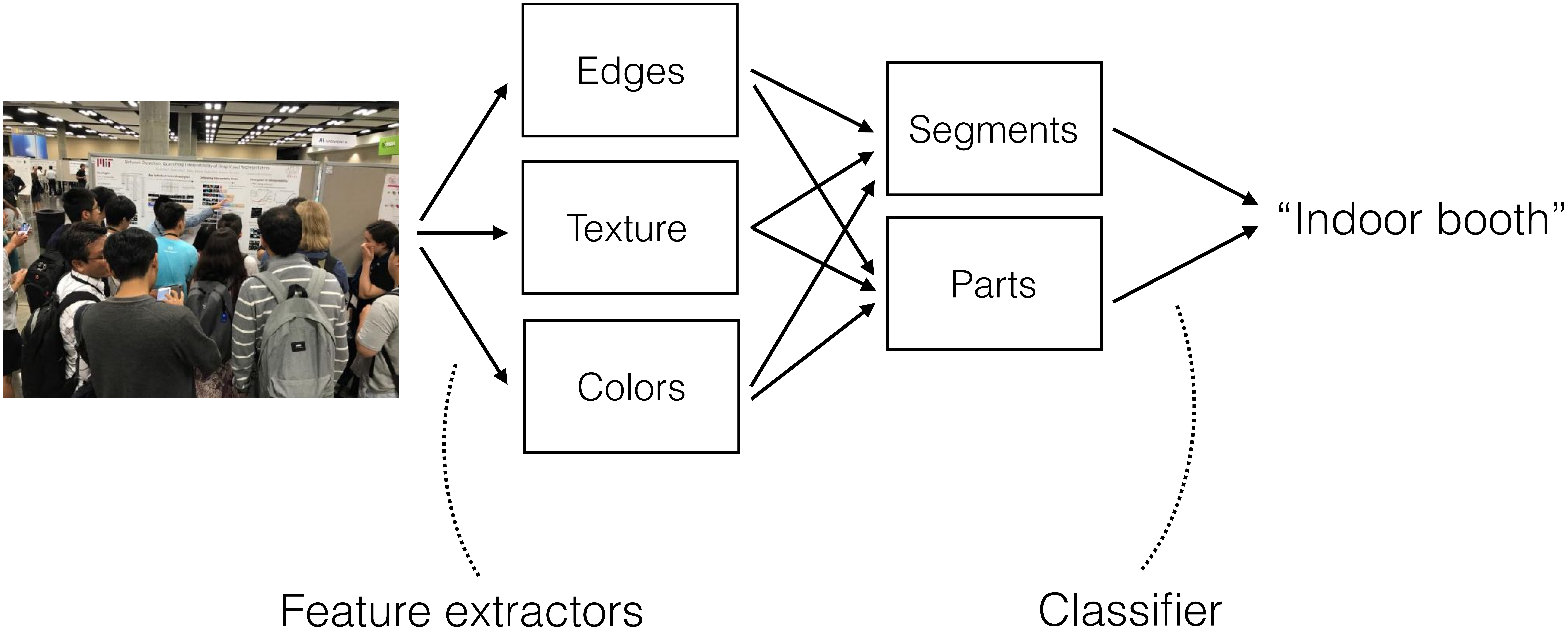
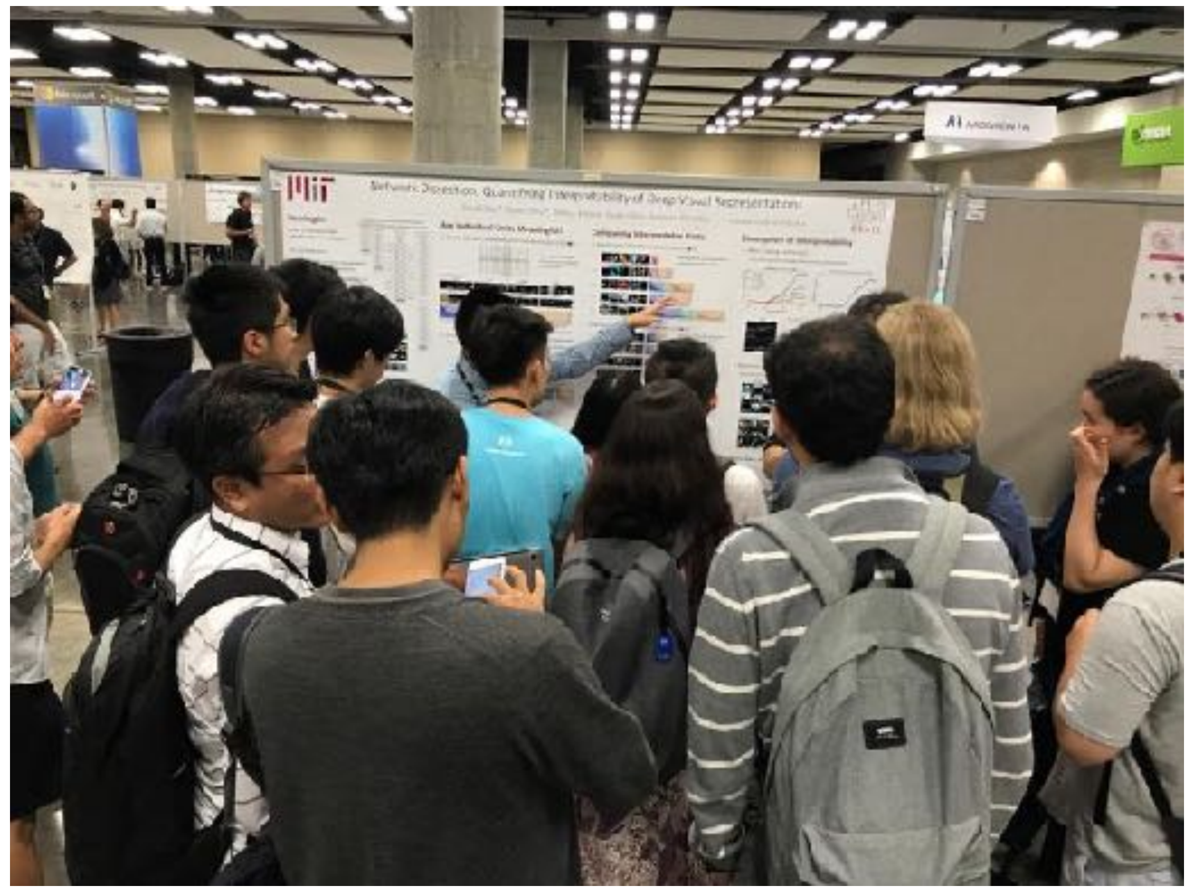


Image recognition



Edges

Texture

Colors

Segments

Parts

Learned

“Indoor booth”

$\phi_k(x)$

$$f_\theta(x) = \sum_{k=1}^{K} \theta_k \phi_k(x)$$

Feature extractors

Classifier

Image recognition

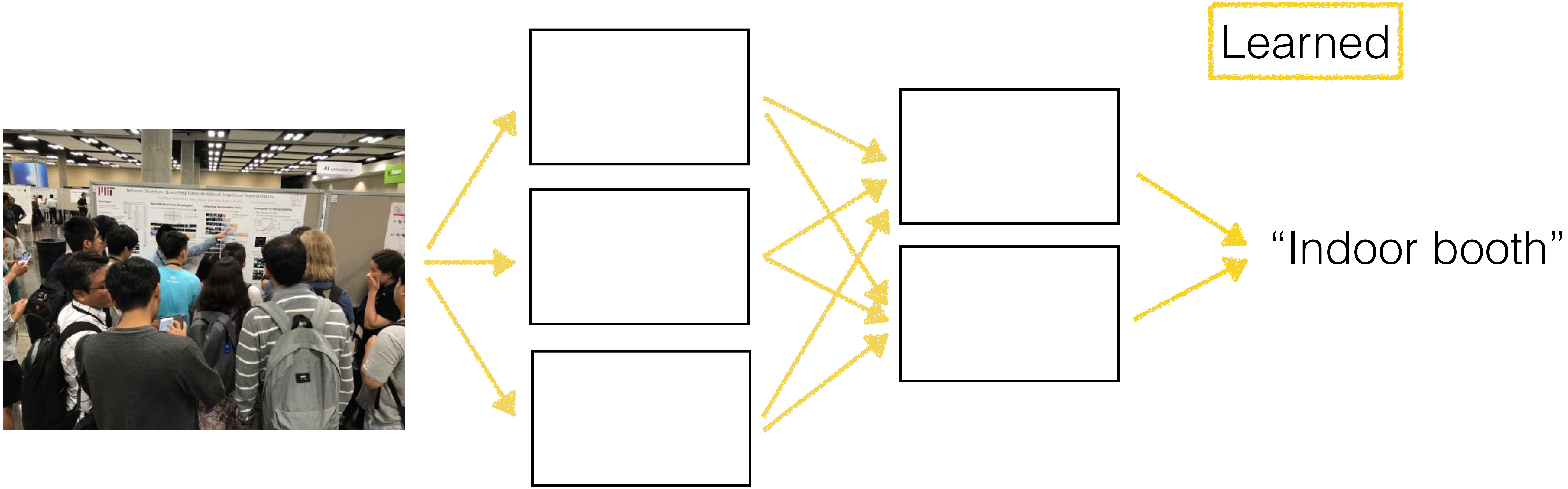
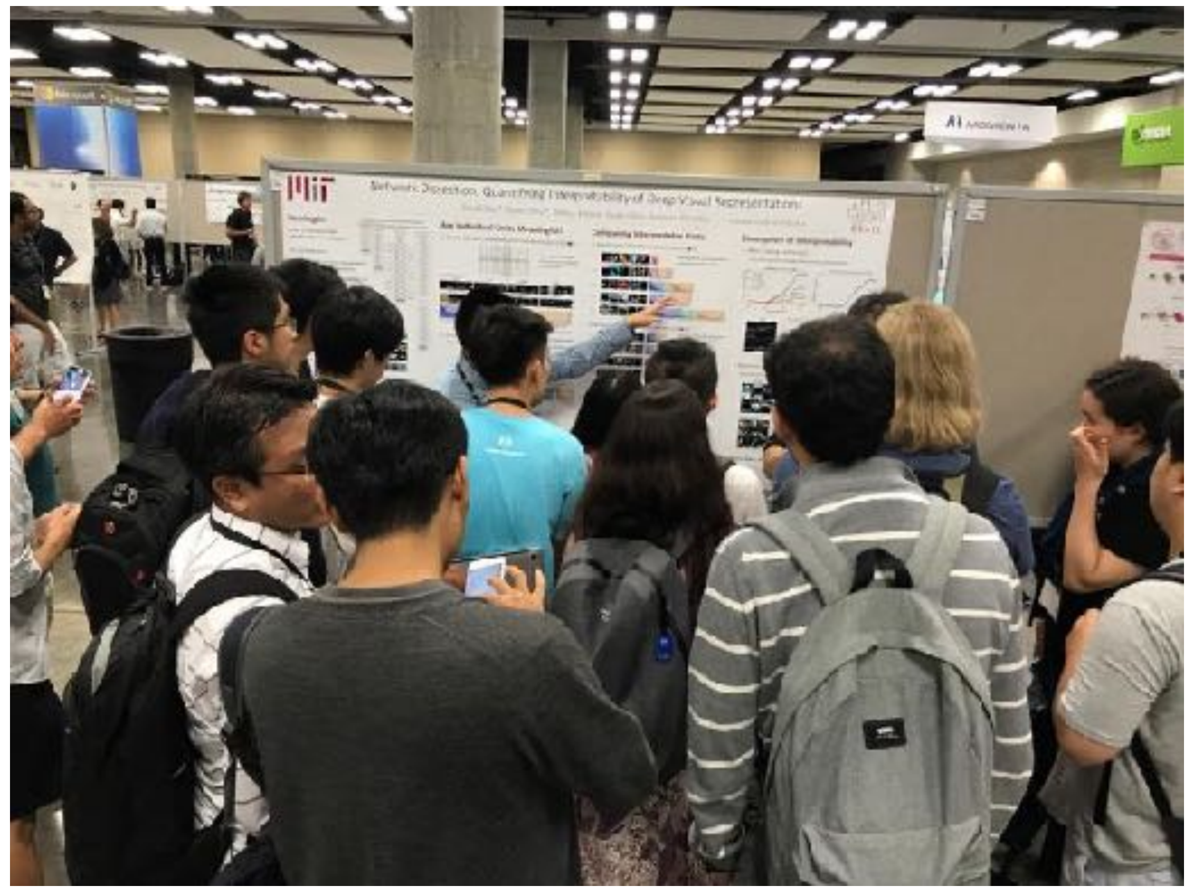


Image recognition

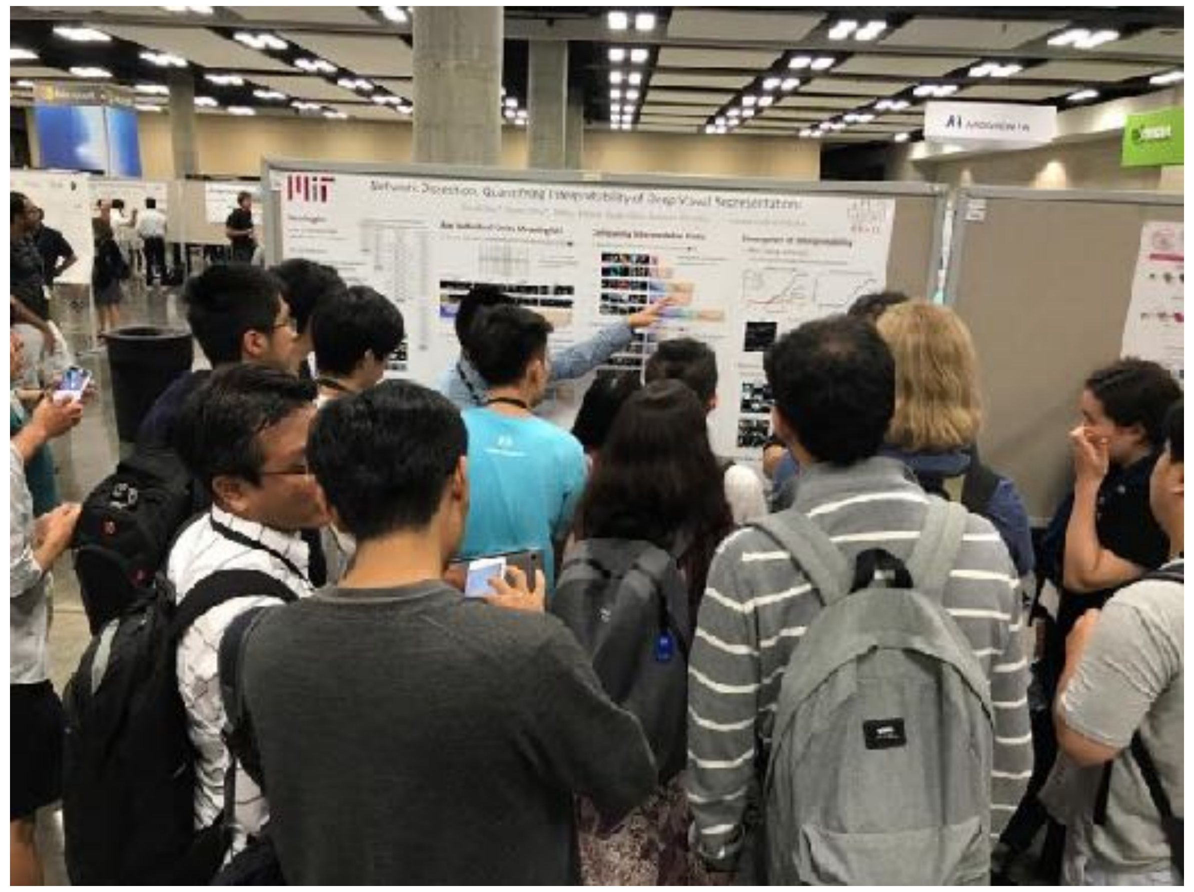
Learned

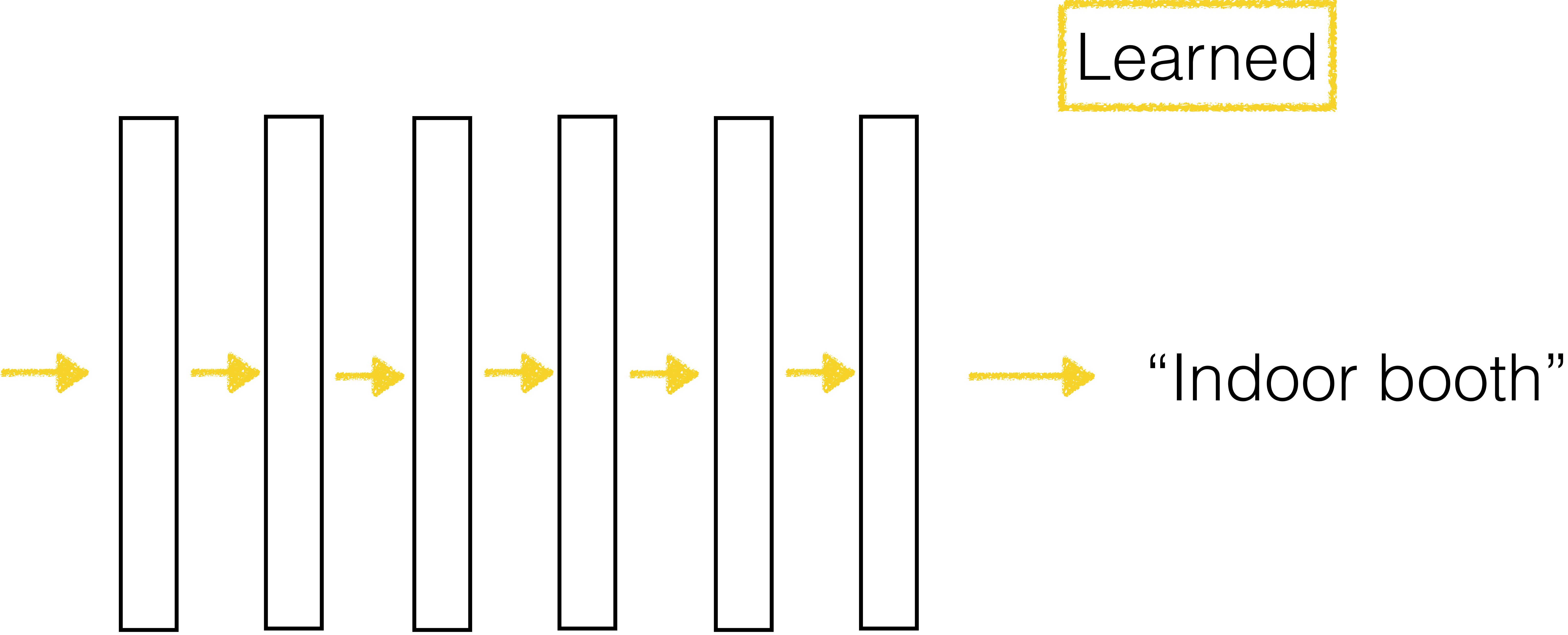


"Indoor booth"

Neural net

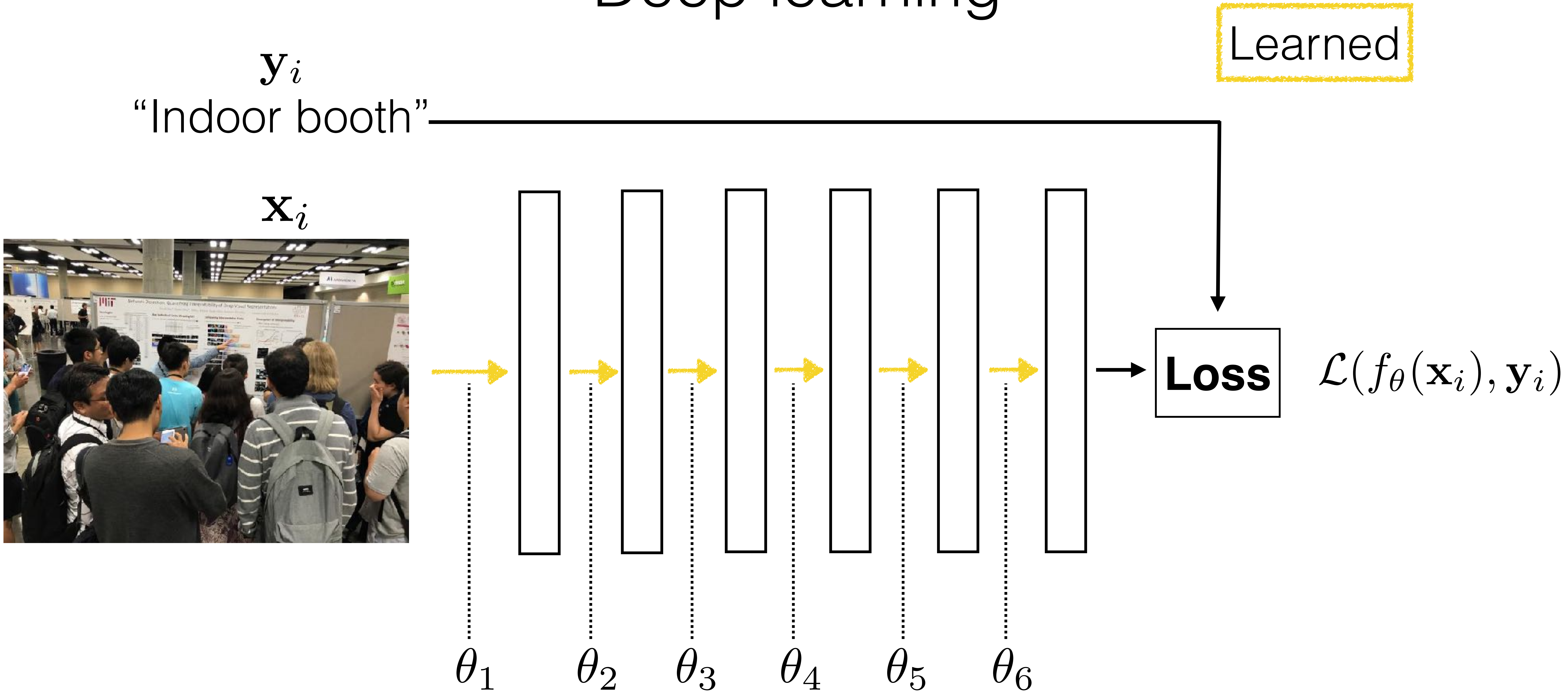
Image recognition





Deep neural net

Deep learning

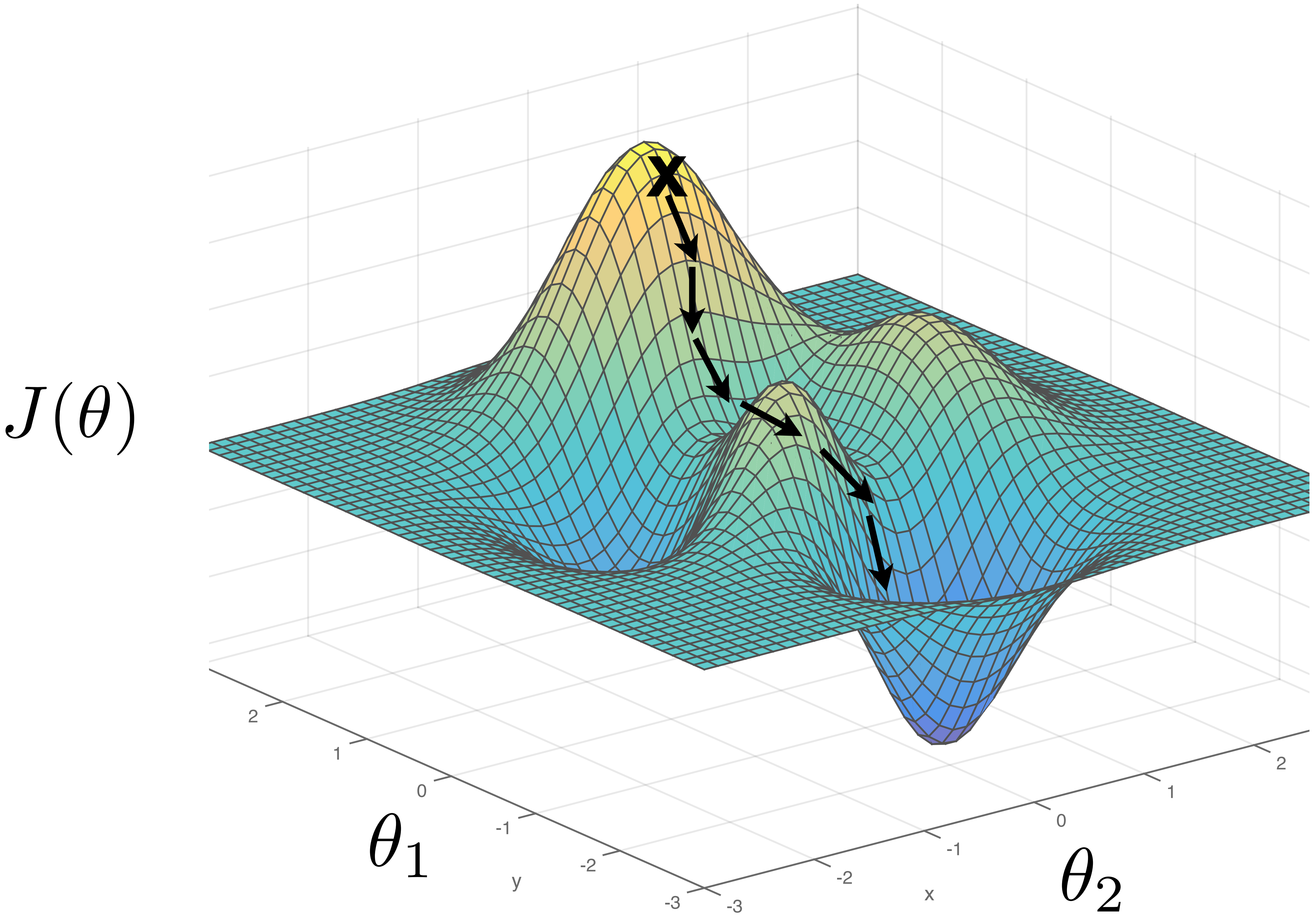


$$\boxed{\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)}$$

Gradient descent

$$\boxed{\theta^* = \arg\min_{\theta} \underbrace{\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)}_{J(\theta)}}$$

Gradient descent



$$\boxed{\theta^* = \arg\min_\theta J(\theta)}$$

Gradient descent

$$\boxed{\theta^* = \arg\min_\theta \underbrace{\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)}_{J(\theta)}}$$

One iteration of gradient descent:

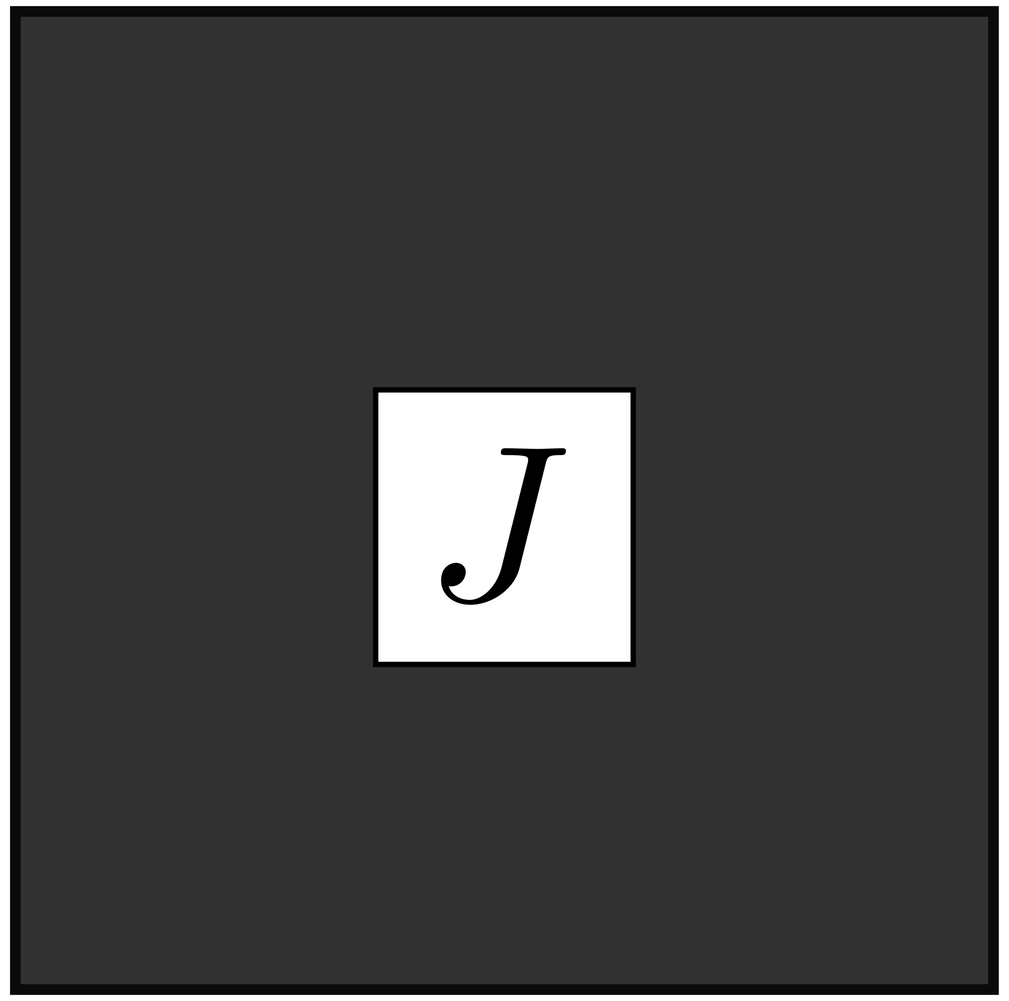
$$\theta^{t+1} = \theta^t - \eta_t \left. \frac{\partial J(\theta)}{\partial \theta} \right|_{\theta=\theta^t}$$

learning rate

Optimization

Params

$\theta \longrightarrow$



$\longrightarrow J(\theta)$

$\nabla_\theta J(\theta)$

$H_\theta(J(\theta))$

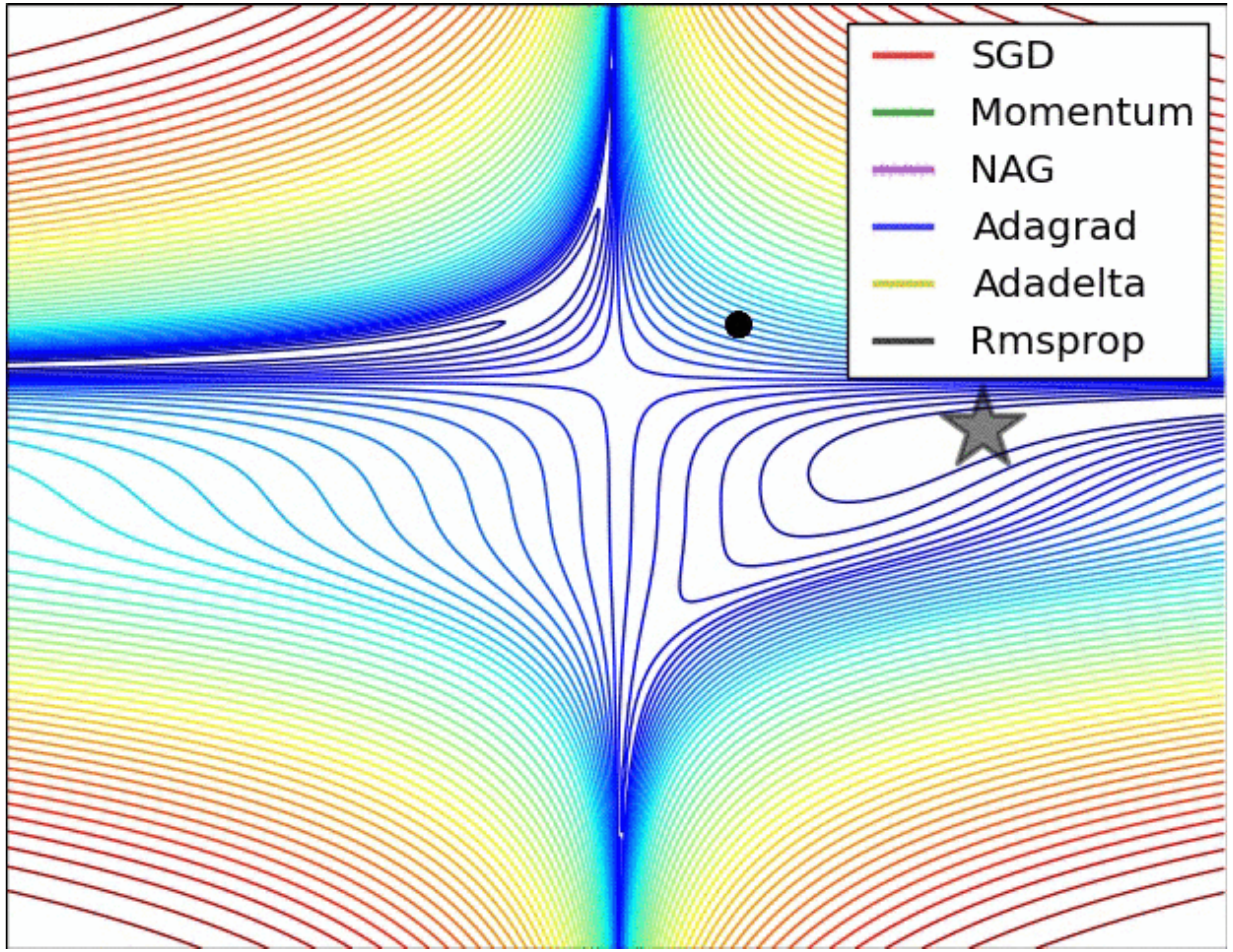
$$\boxed{\theta^* = \arg\min_\theta J(\theta)}$$

- What's the knowledge we have about J?
  - We can evaluate $J(\theta)$ $\longleftarrow$ Black box optimization
  - We can evaluate $J(\theta)$ and $\nabla_\theta J(\theta)$ $\longleftarrow$ First order optimization

    Gradient
  - We can evaluate $J(\theta)$, $\nabla_\theta J(\theta)$, and $H_\theta(J(\theta))$ $\longleftarrow$ Second order optimization

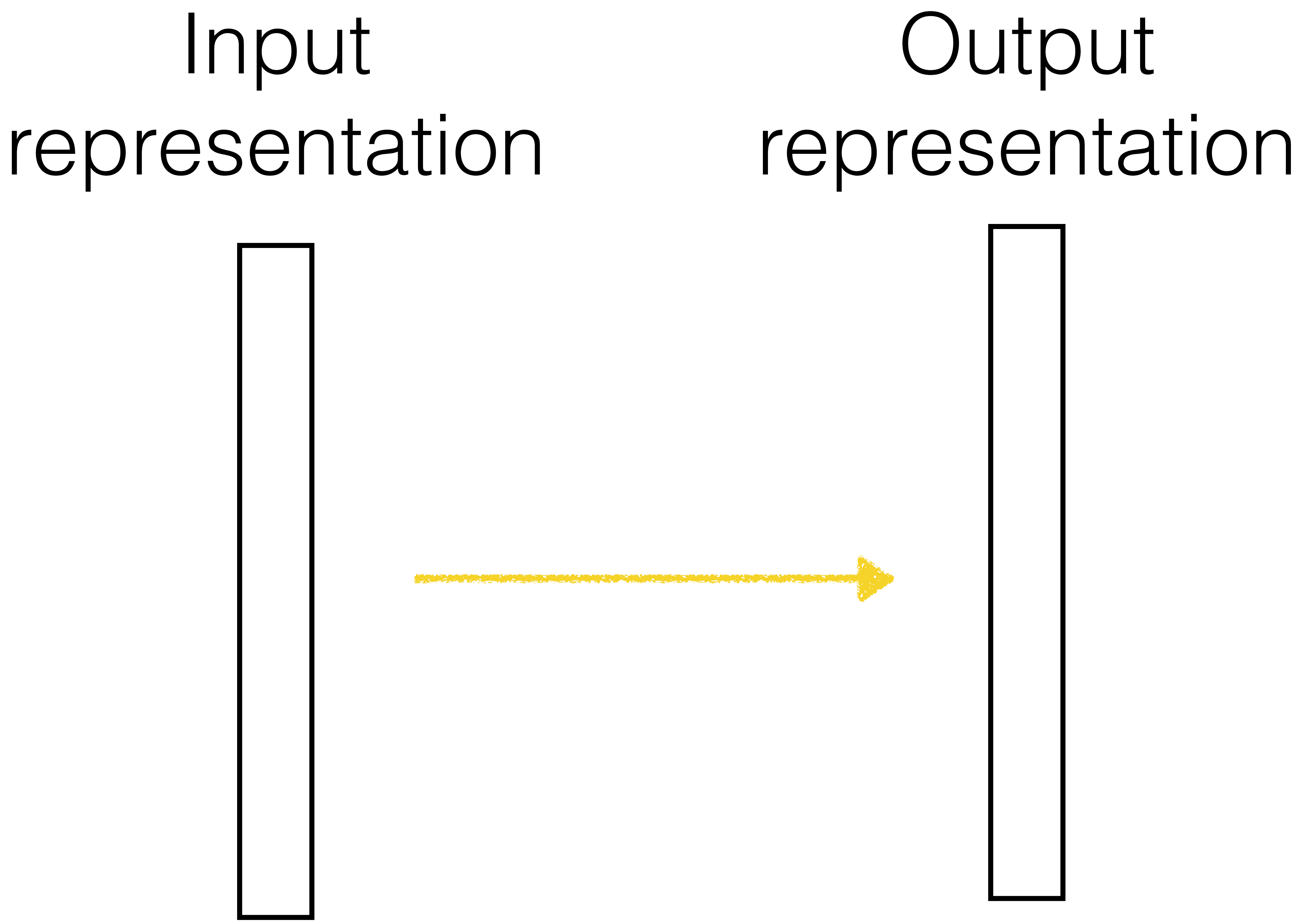
    Hessian

Comparison of gradient descent variants



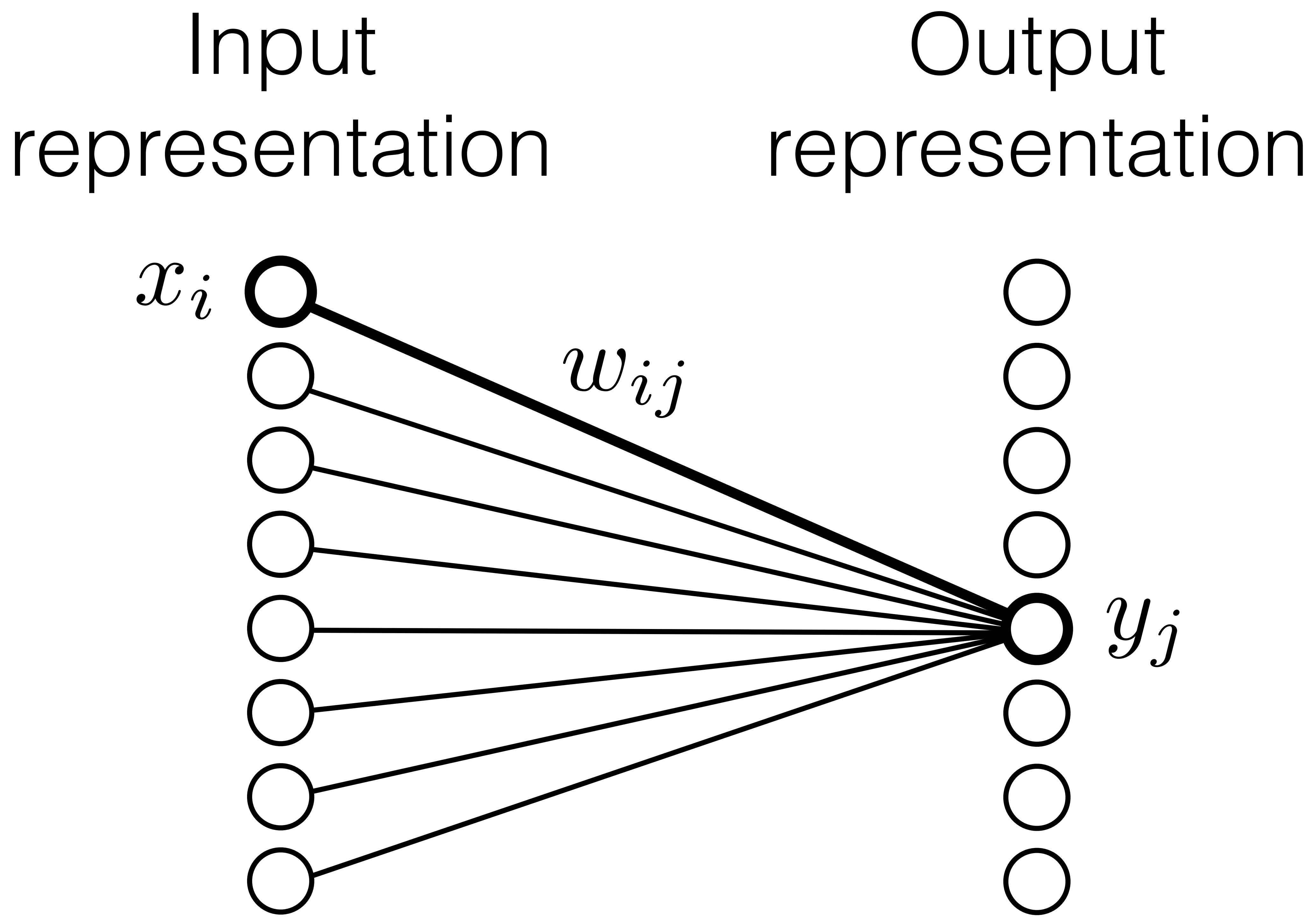
[http://ruder.io/optimizing-gradient-descent/]

Computation in a neural net



Computation in a neural net

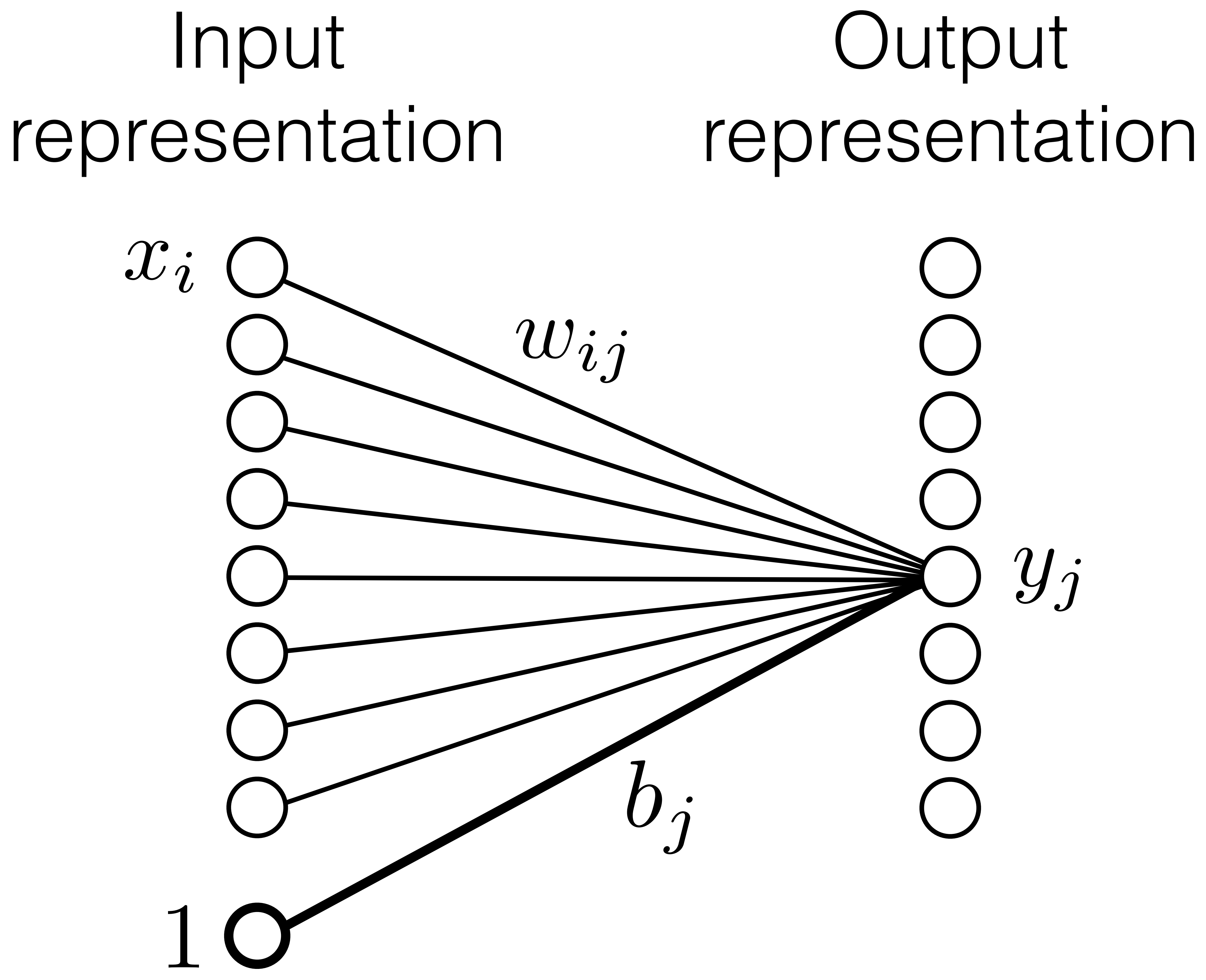
Linear layer



$$y_j = \sum_i w_{ij} x_i$$

Computation in a neural net

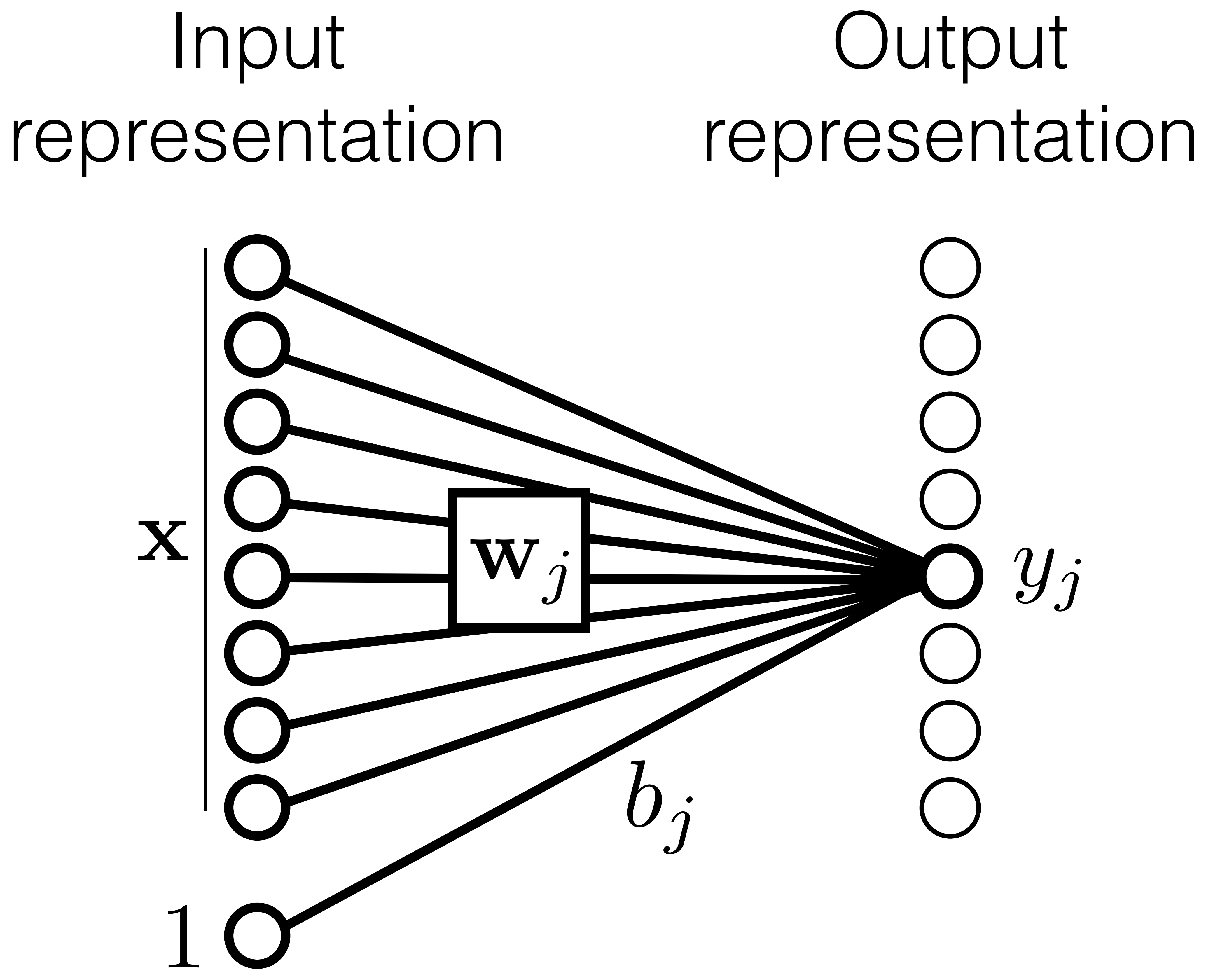
Linear layer



$$y_j = \sum_i \underbrace{w_{ij}}_{\text{weights}} x_i + \underbrace{b_j}_{\text{bias}}$$

Computation in a neural net

Linear layer



weights

$$y_j = \mathbf{x}^T \mathbf{w}_j + b_j$$

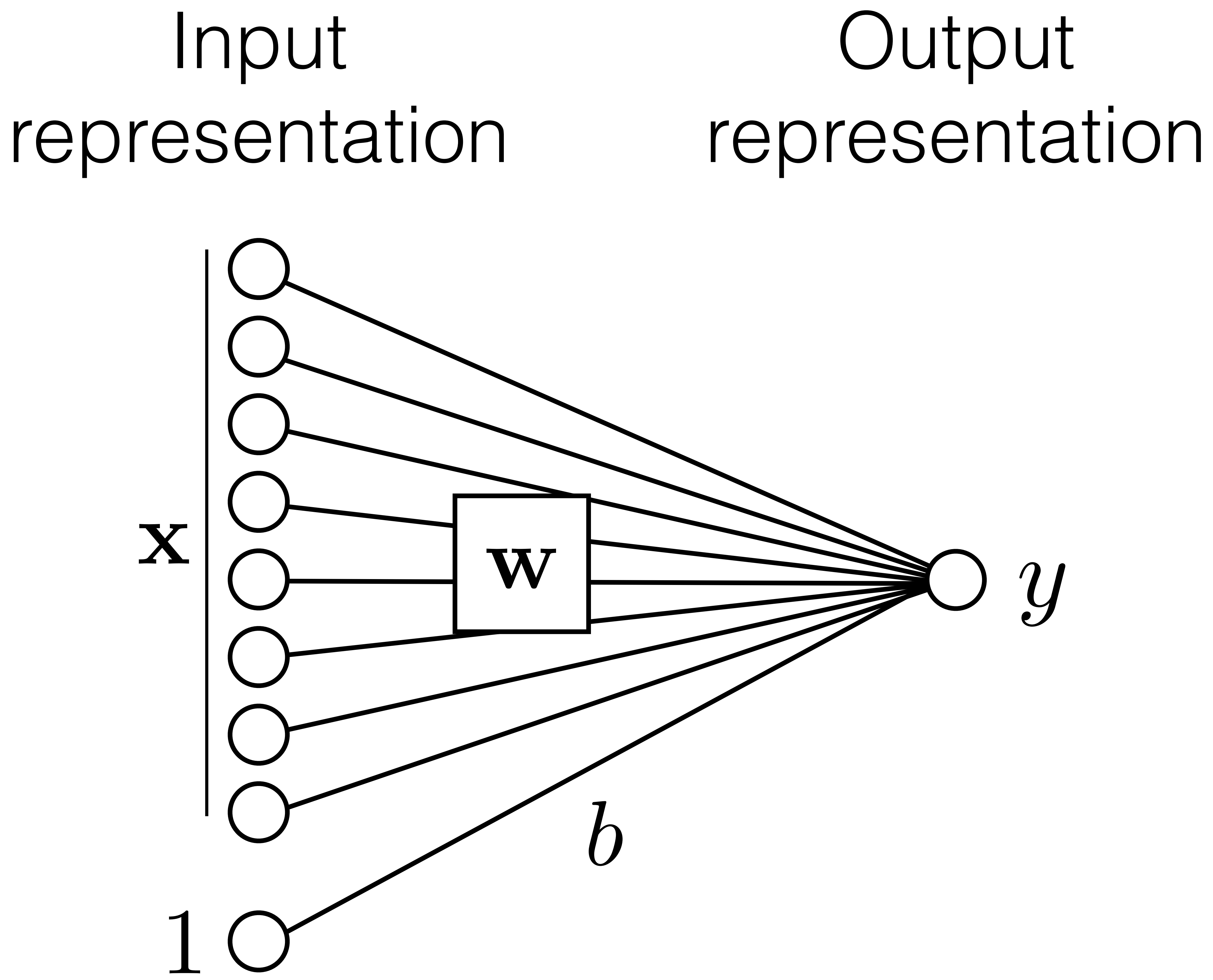
bias

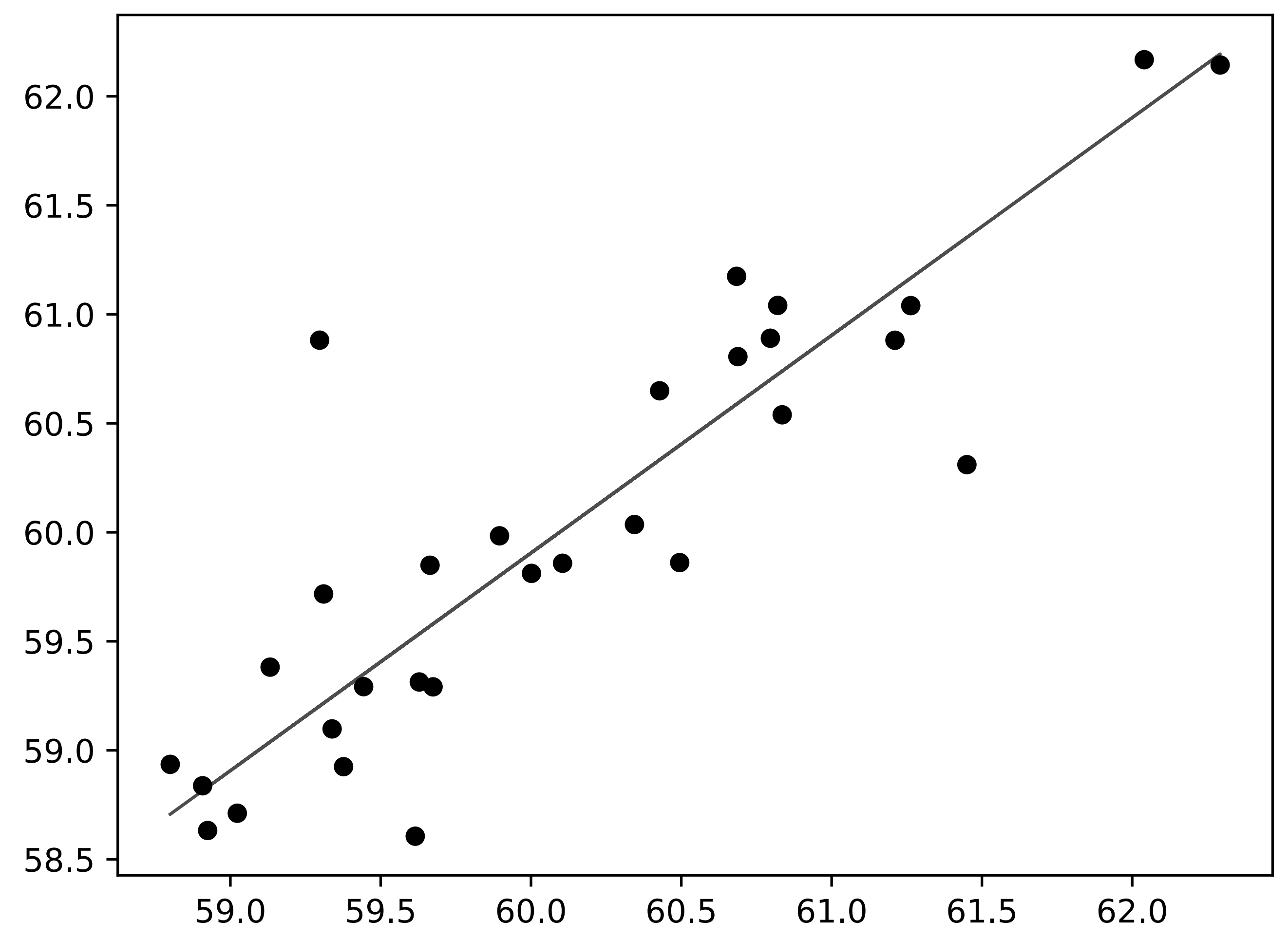
$$\theta = \{\mathbf{W}, \mathbf{b}\}$$

parameters of the model

Example: linear regression with a neural net

Linear layer

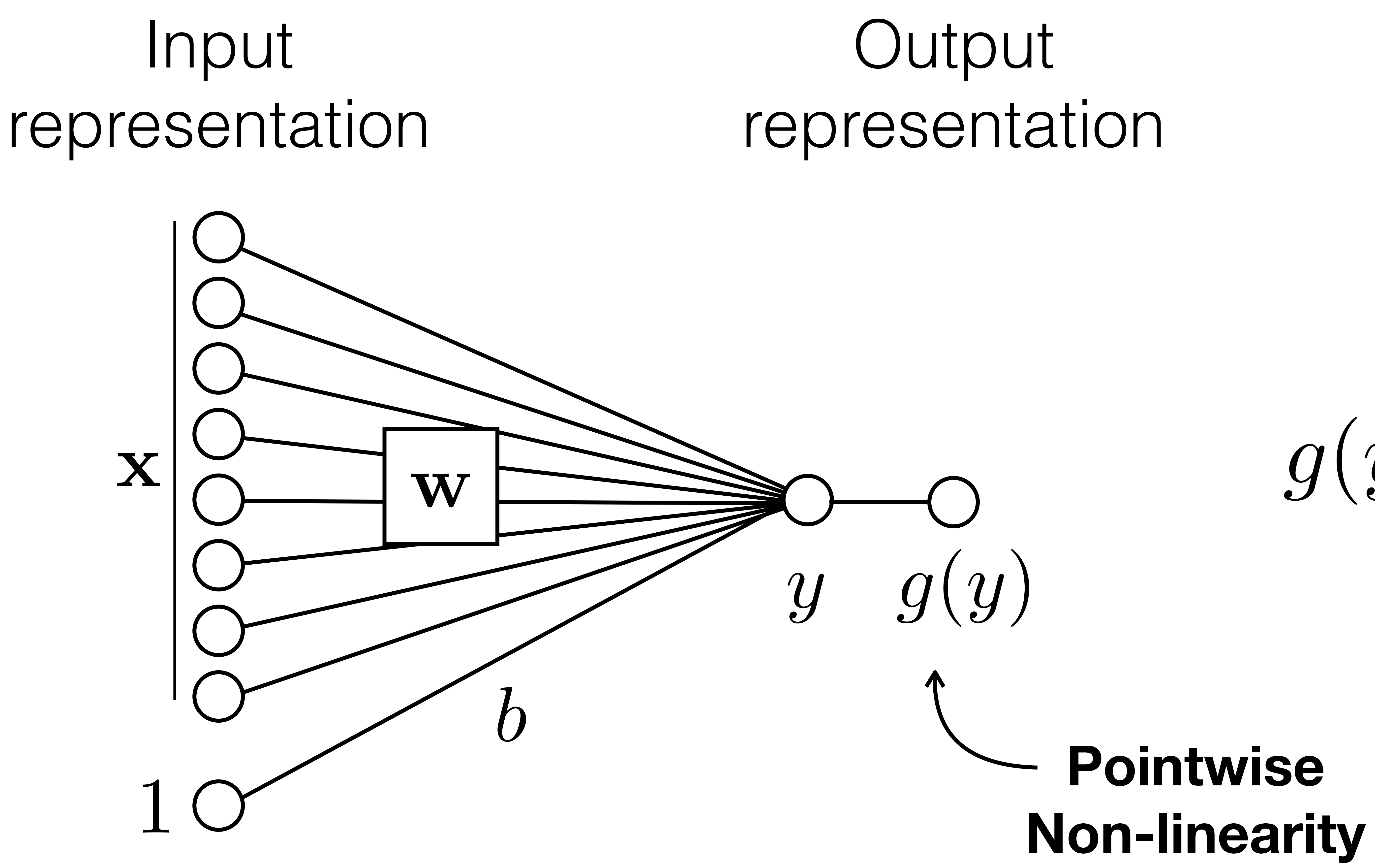




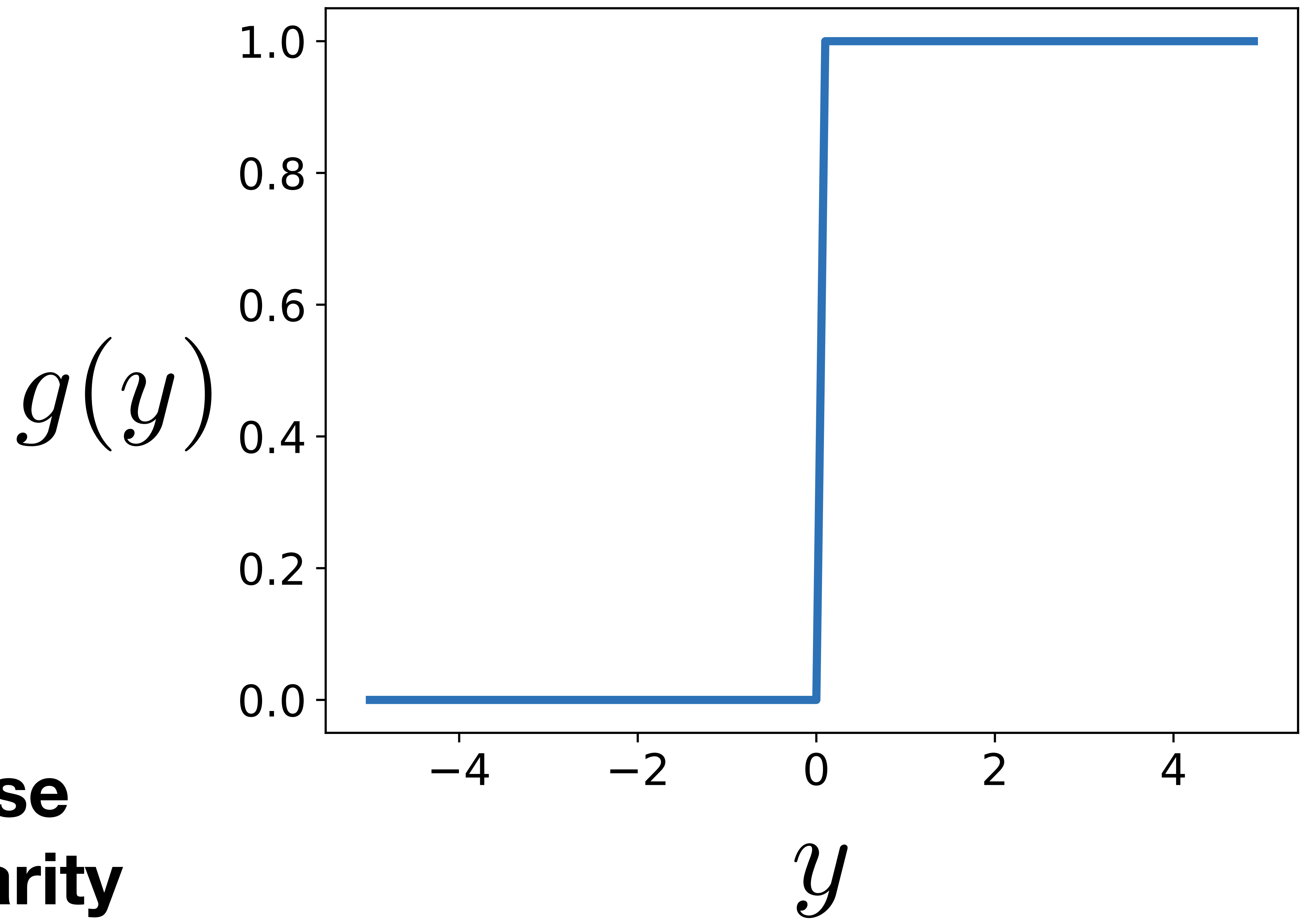
$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b$$

Computation in a neural net

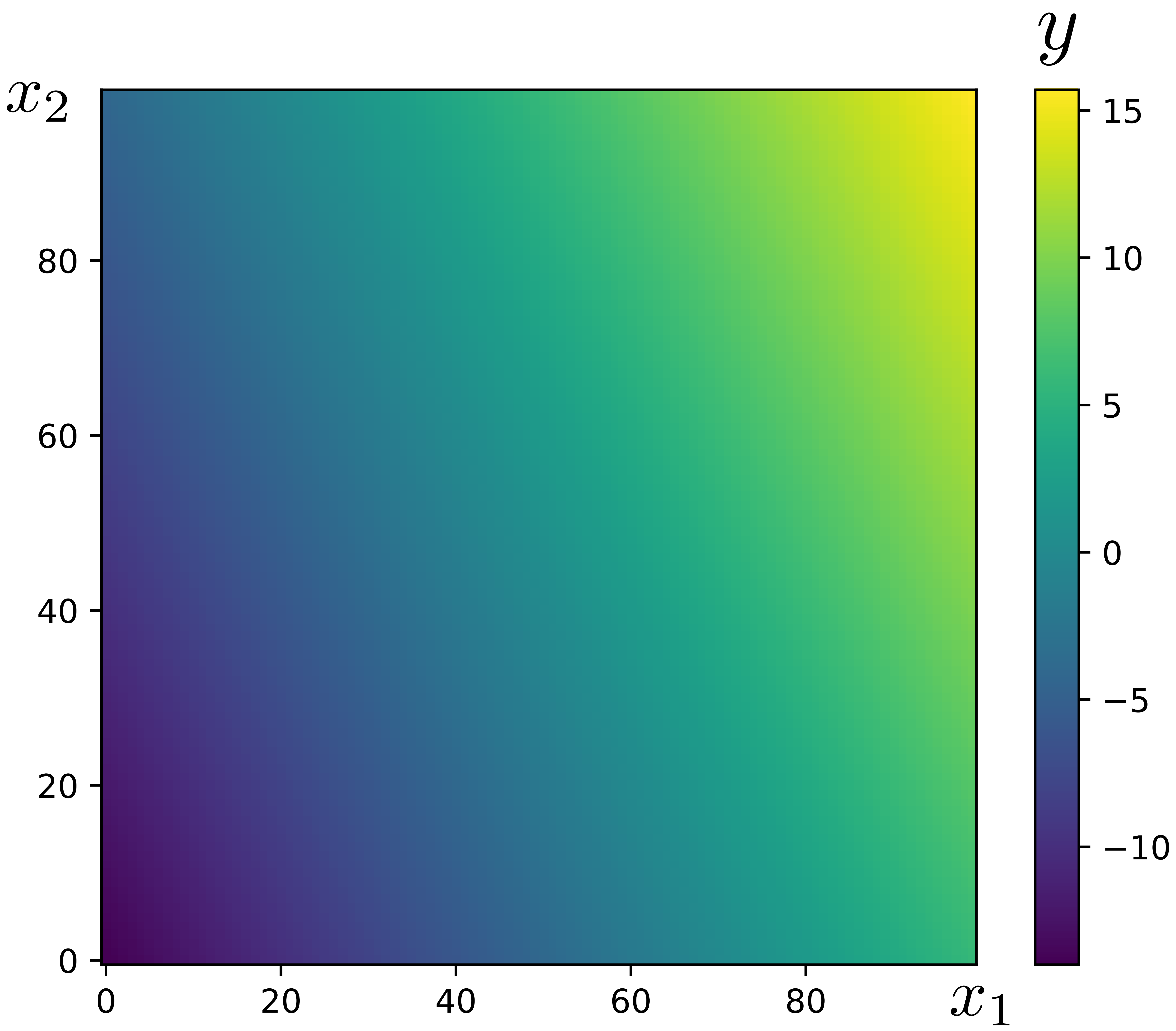
"Perceptron"



$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

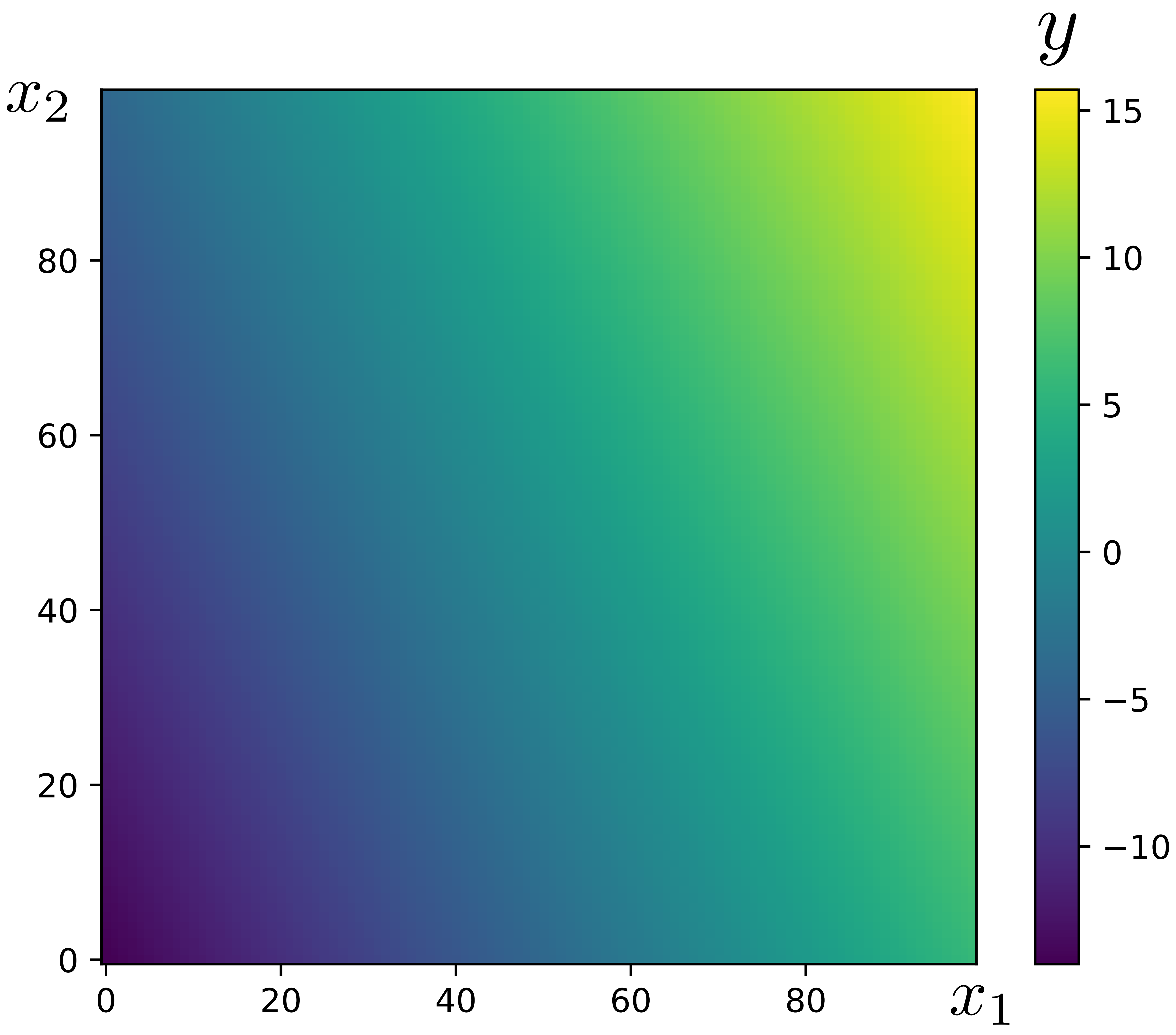


Example: linear classification with a perceptron



$$y = \mathbf{x}^T \mathbf{w} + b$$

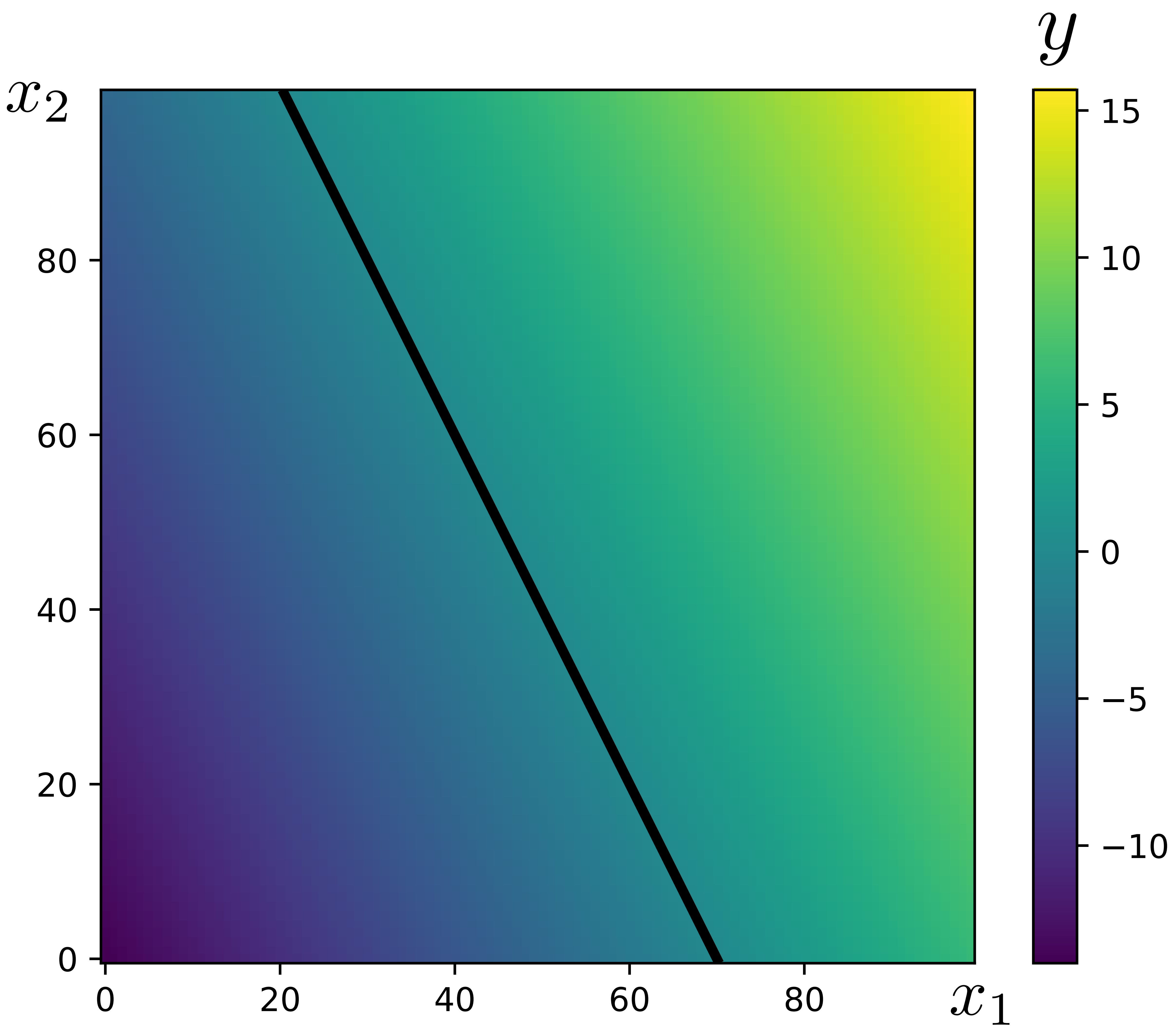
Example: linear classification with a perceptron



$$y = \mathbf{x}^T \mathbf{w} + b$$

$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

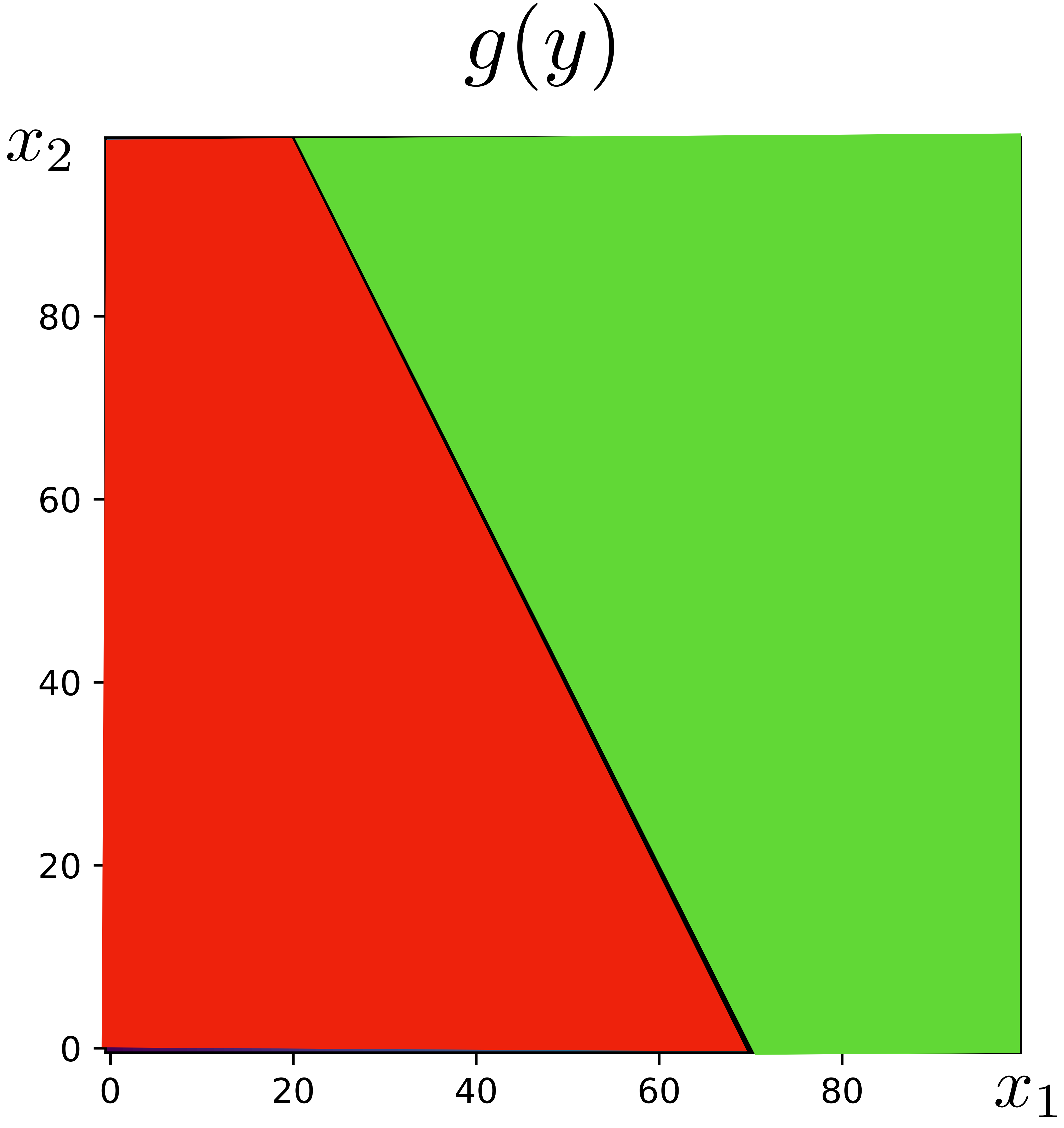
Example: linear classification with a perceptron



$$y = \mathbf{x}^T \mathbf{w} + b$$

$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

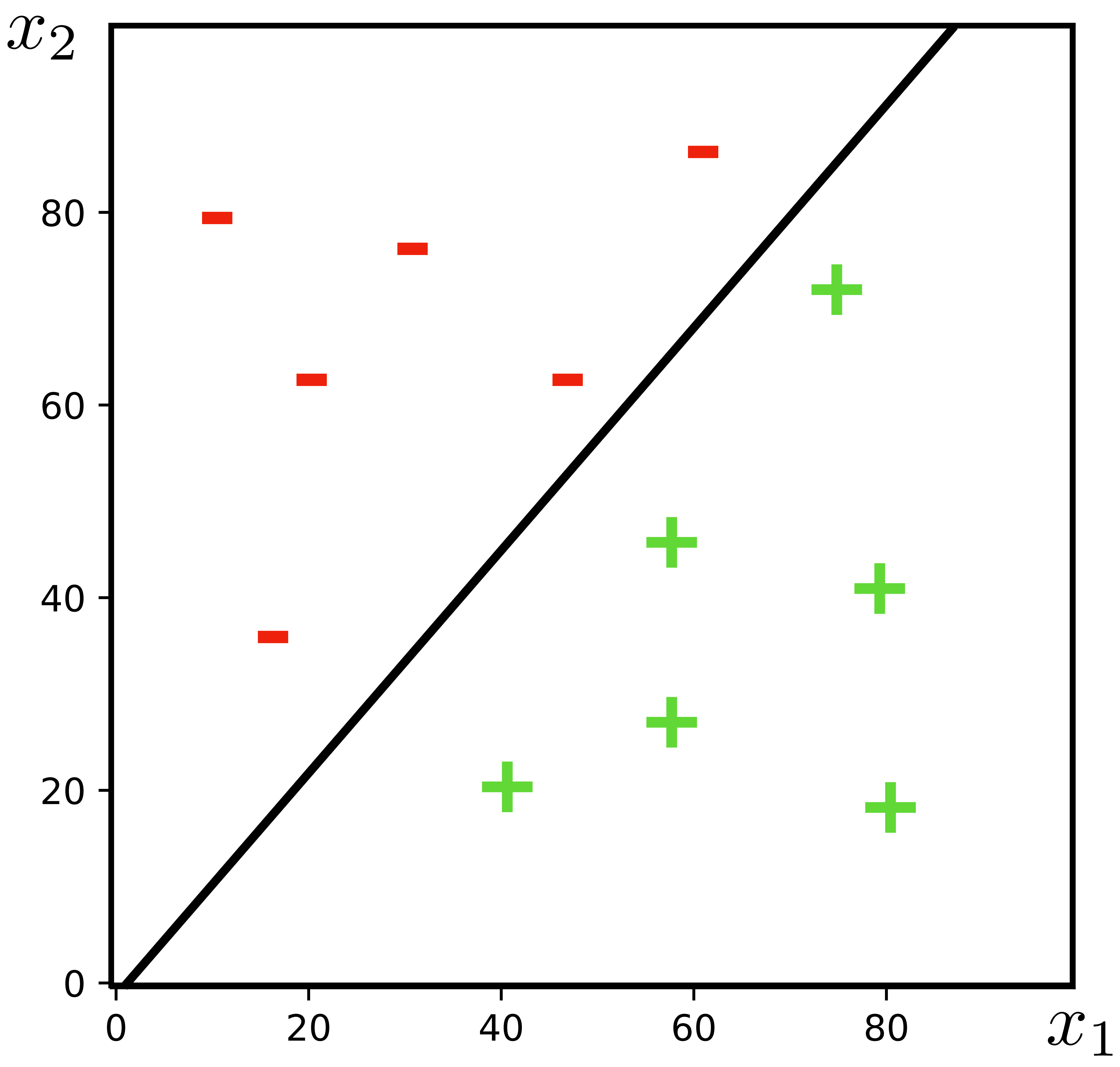
Example: linear classification with a perceptron



$$y = \mathbf{x}^T \mathbf{w} + b$$

$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

Example: linear classification with a perceptron

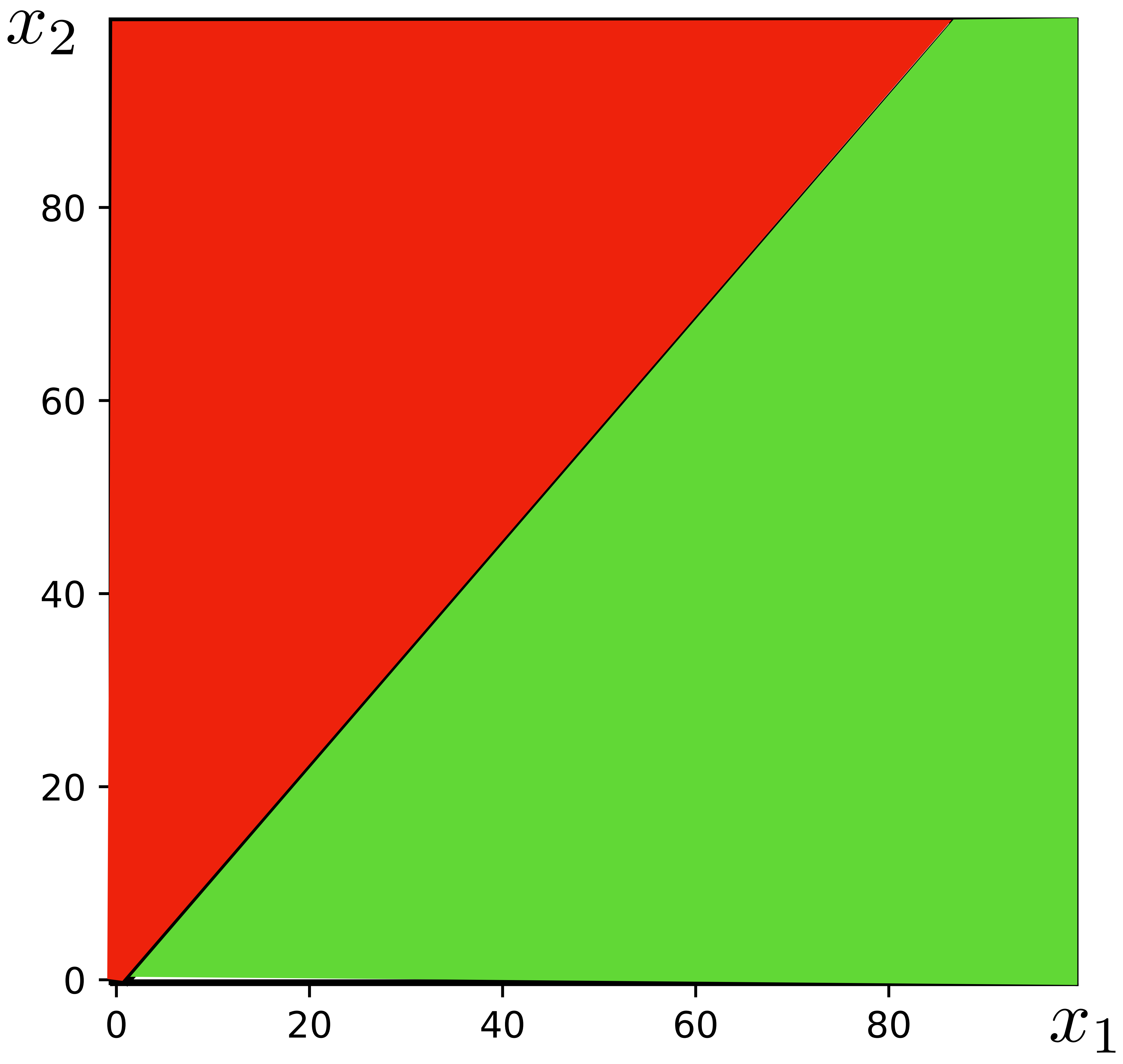


$$\hat{y} = \mathbf{x}^T \mathbf{w} + b$$

$$g(\hat{y}) = \begin{cases} 1, & \text{if} \quad \hat{y} > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\boxed{\mathbf{w}^*, b^* = \arg\min_{\mathbf{w}, b} \mathcal{L}(g(\hat{y}), y_i)}$$

Example: linear classification with a perceptron

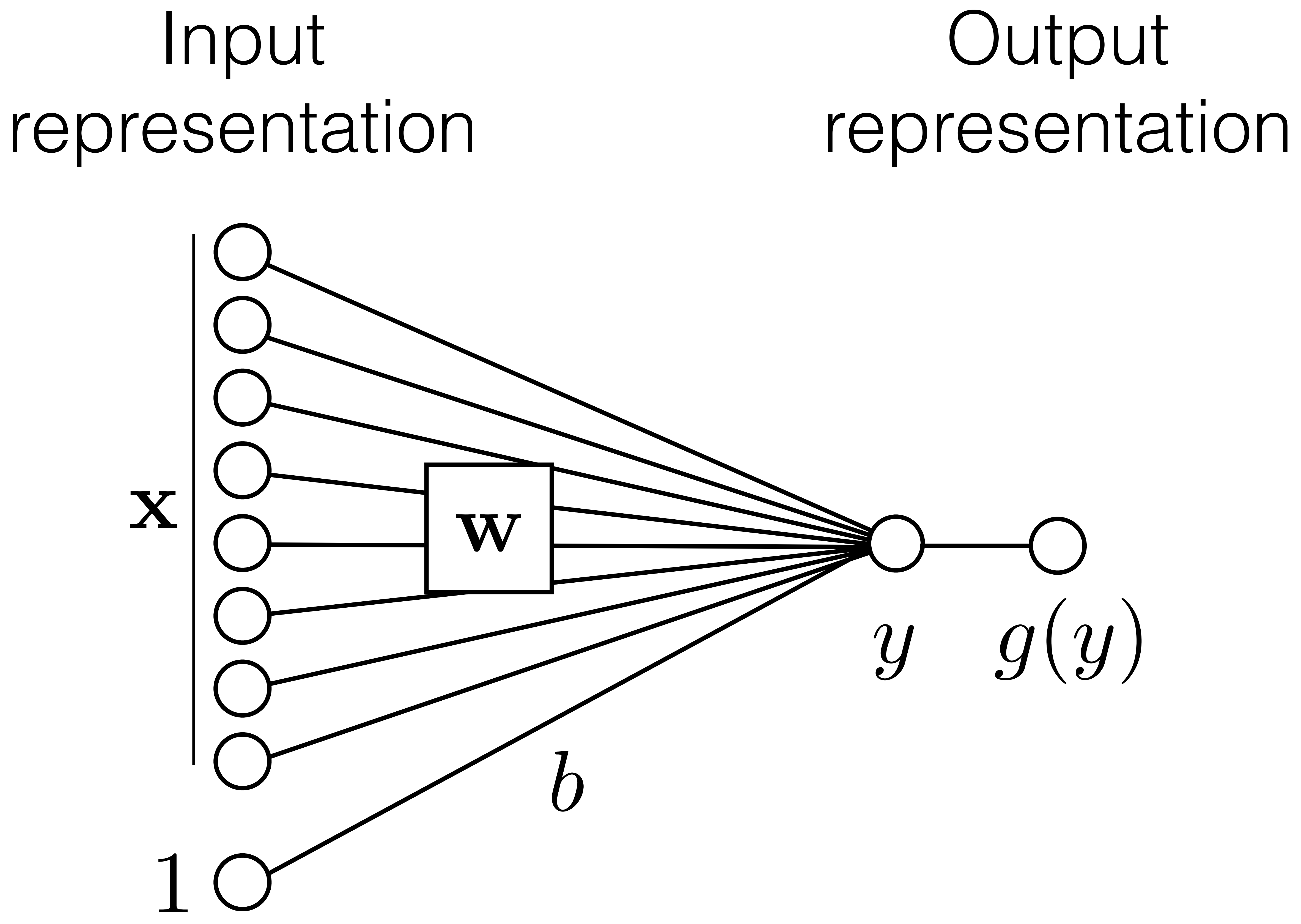


$$\hat{y} = \mathbf{x}^T \mathbf{w} + b$$

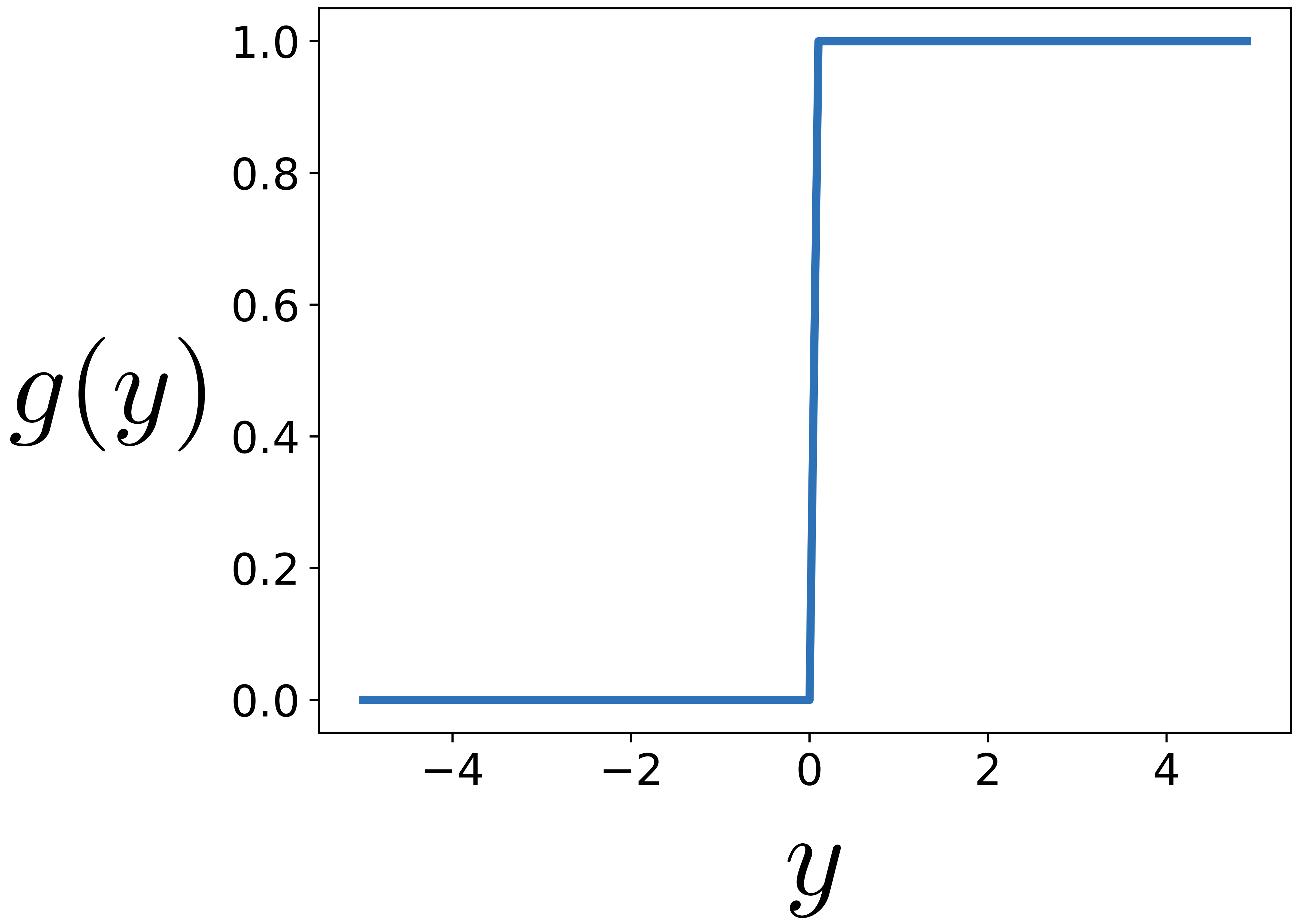
$$g(\hat{y}) = \begin{cases} 1, & \text{if} \quad \hat{y} > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\boxed{\mathbf{w}^*, b^* = \arg\min_{\mathbf{w}, b} \mathcal{L}(g(\hat{y}), y_i)}$$

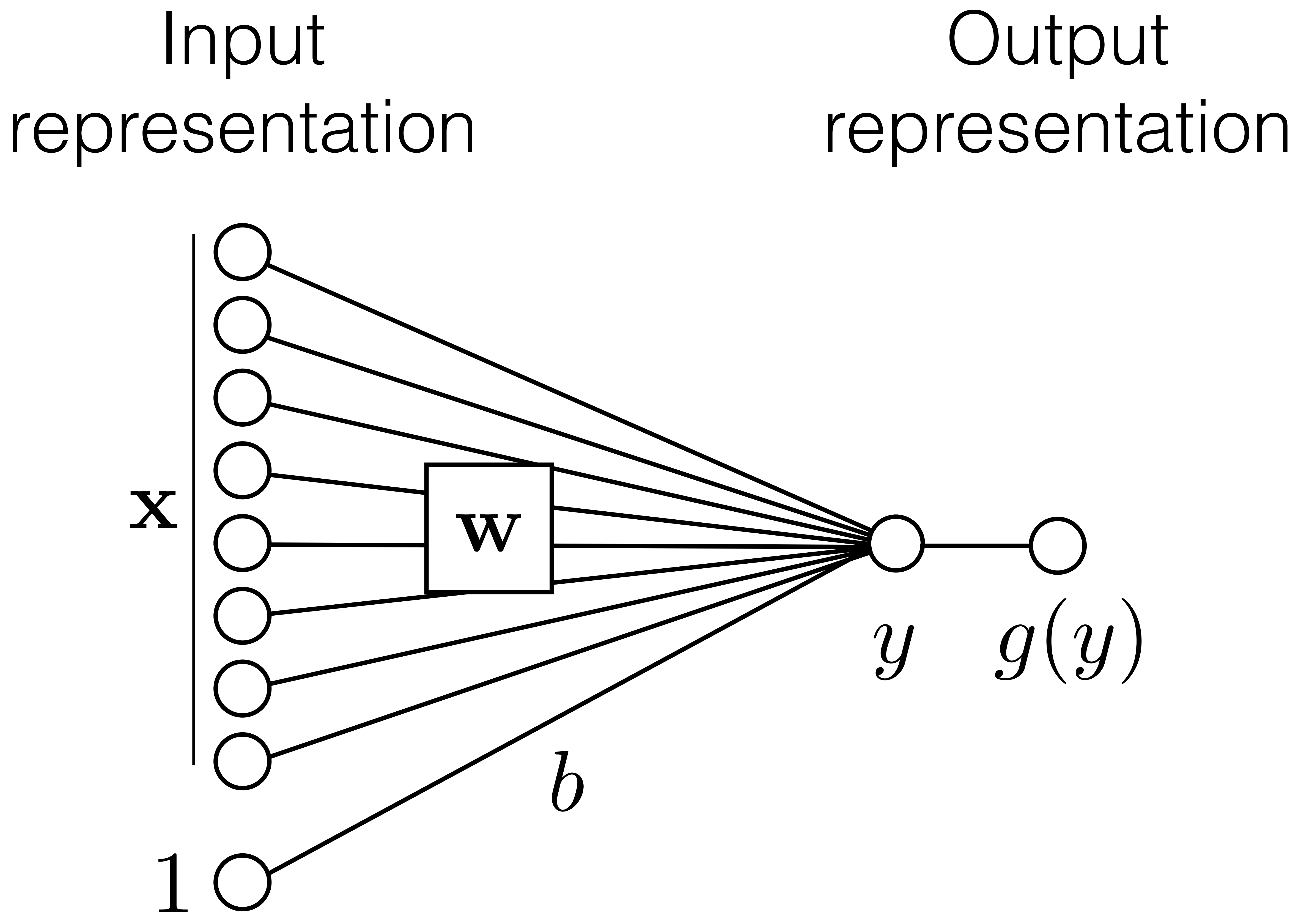
Computation in a neural net



$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

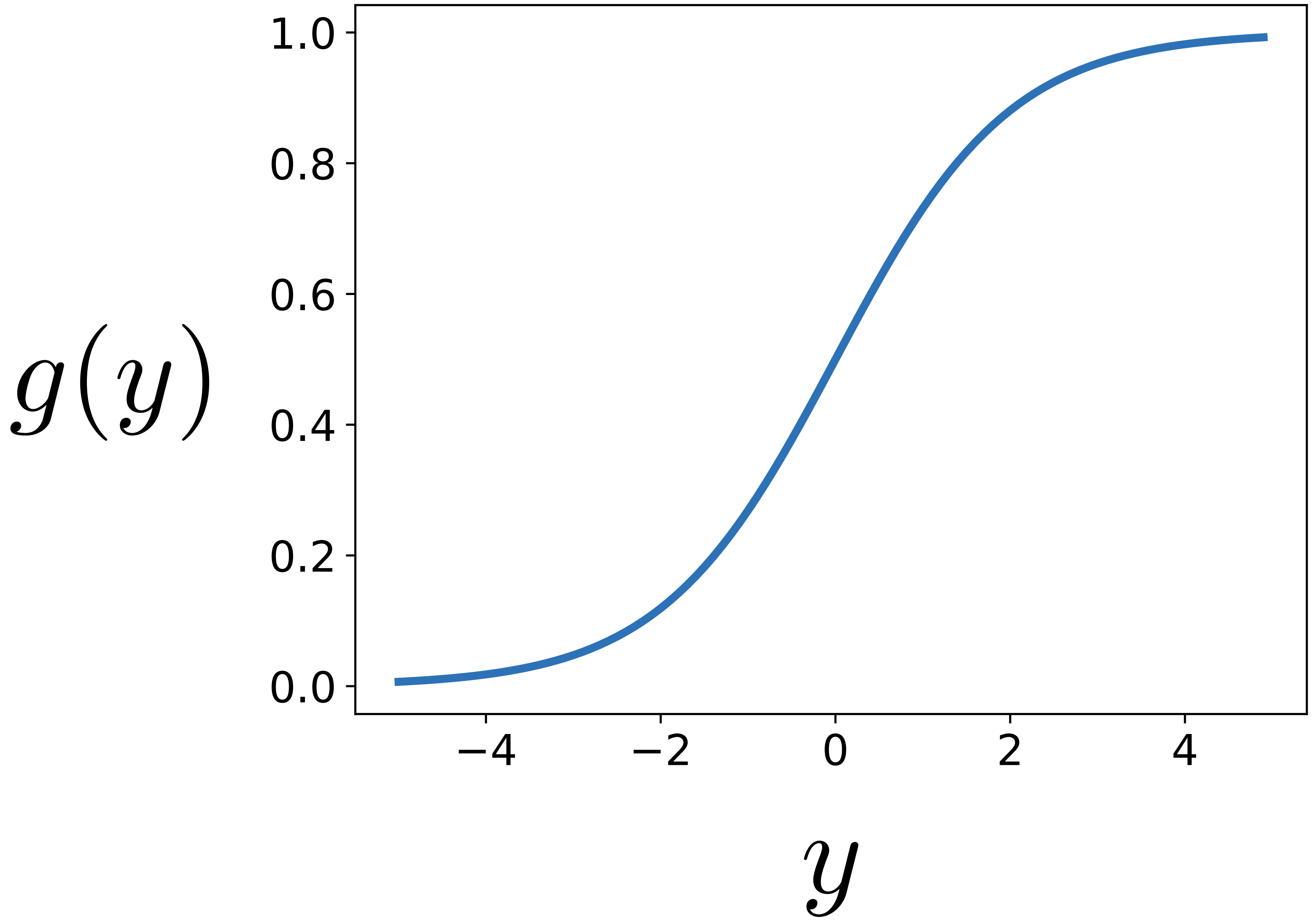


Computation in a neural net — nonlinearity



Sigmoid

$$g(y) = \frac{1}{1 + e^{-y}}$$

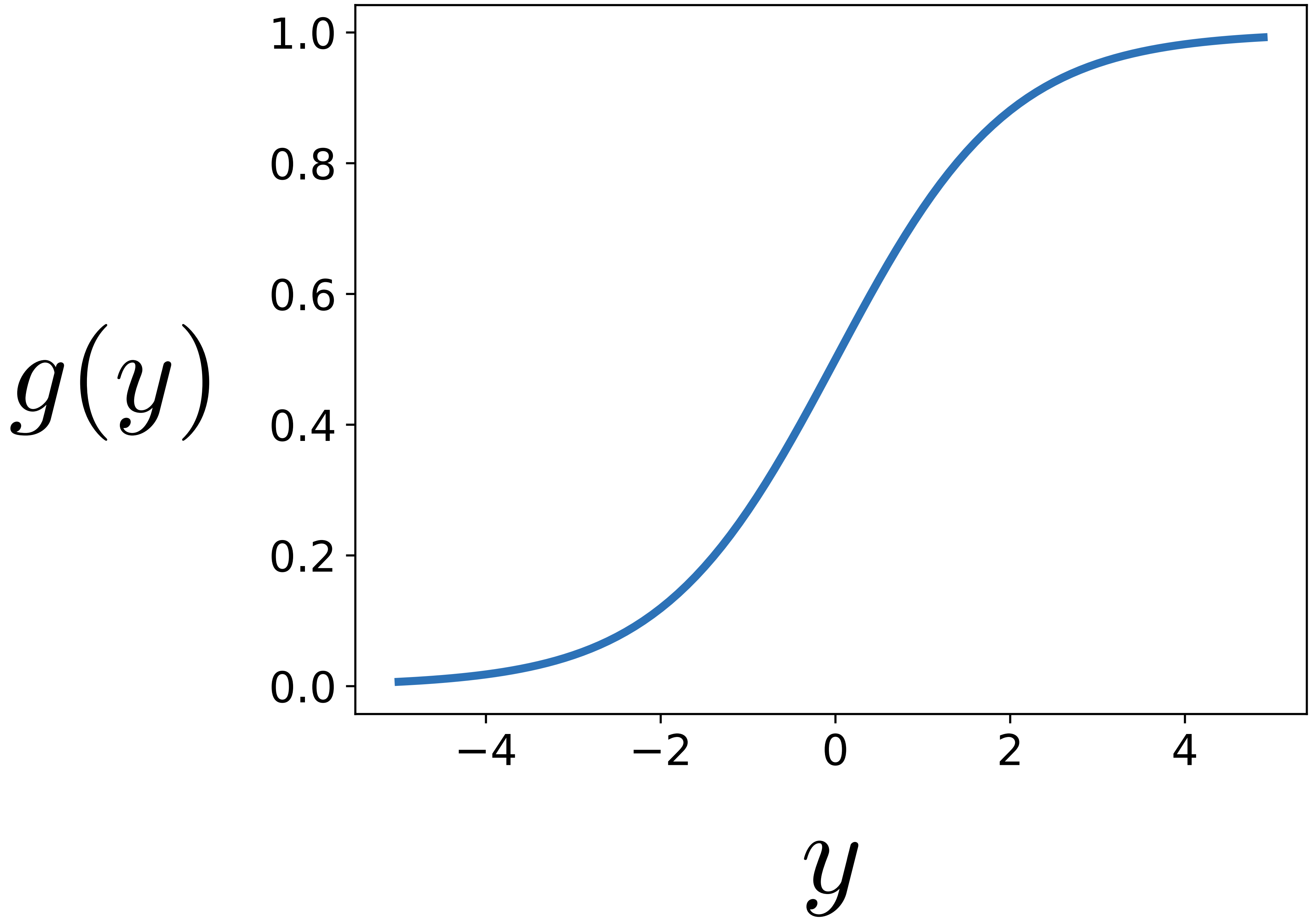


Computation in a neural net — nonlinearity

- Interpretation as firing rate of neuron
- Bounded between [0,1]
- Saturation for large +/- inputs
- Gradients go to zero
- Outputs centered at 0.5
  (poor conditioning)
- Not used in practice

Sigmoid

$$g(y) = \frac{1}{1 + e^{-y}}$$



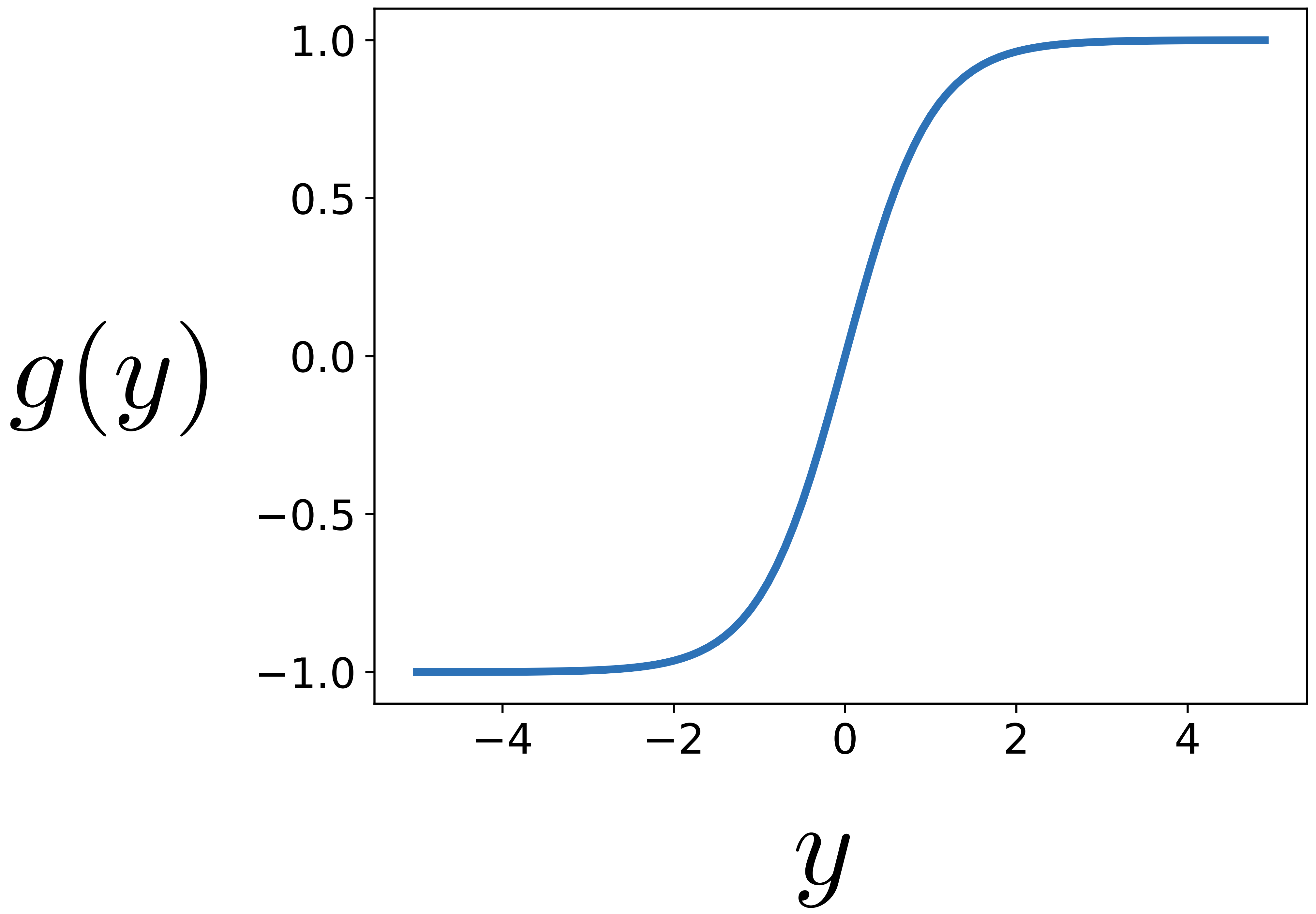
Computation in a neural net — nonlinearity

- Bounded between [-1,+1]
- Saturation for large +/- inputs
- Gradients go to zero
- Outputs centered at 0
- Preferable to sigmoid

tanh(x) = 2 sigmoid(2x) –1

Tanh

$$g(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}}$$

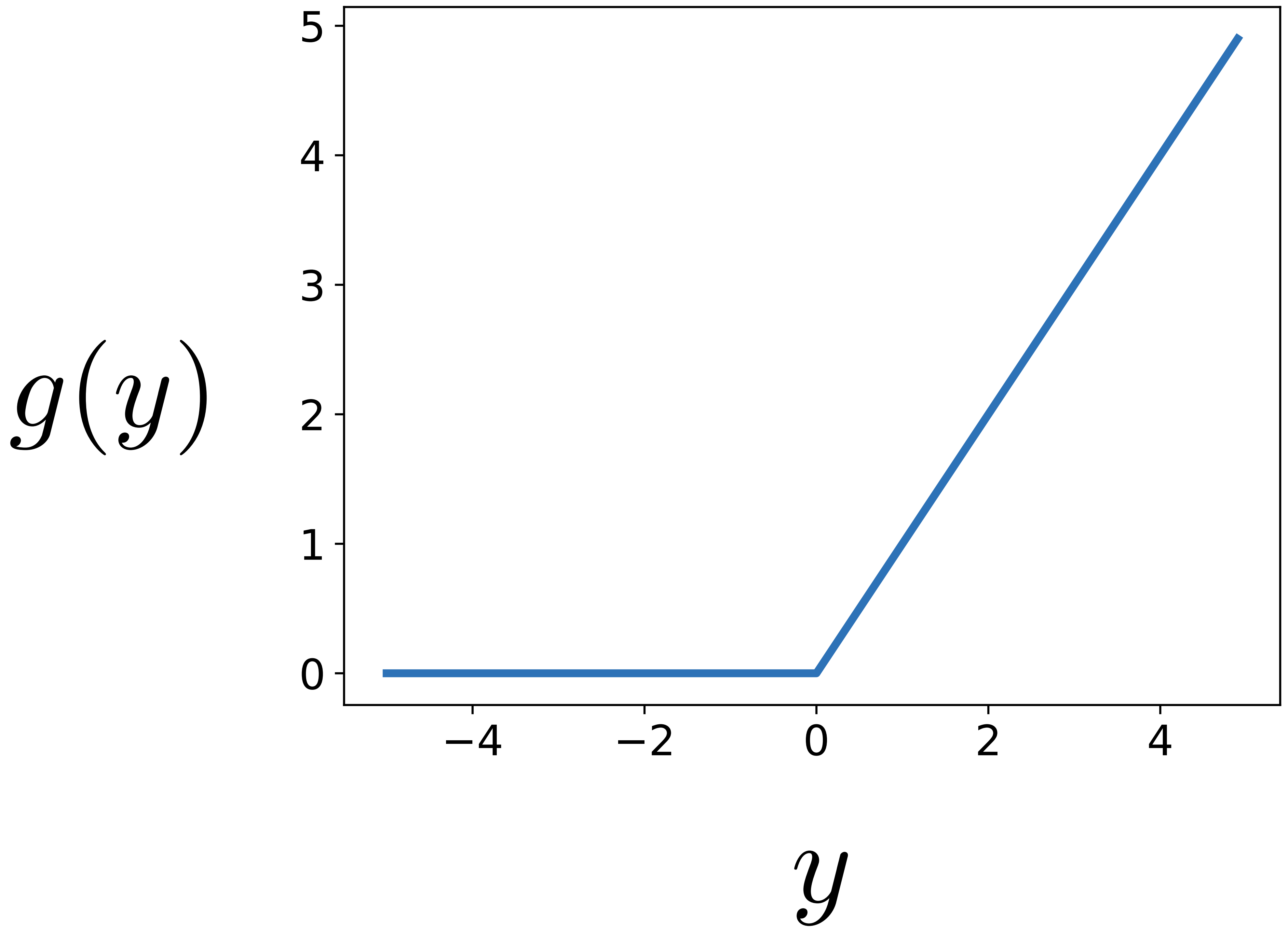


Computation in a neural net — nonlinearity

- Unbounded output (on positive side)
- Efficient to implement: $\frac{\partial g}{\partial y} = \begin{cases} 0, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$
- Also seems to help convergence (see 6x speedup vs tanh in [Krizhevsky et al.])
- Drawback: if strongly in negative region, unit is dead forever (no gradient).
- Default choice: widely used in current models.

Rectified linear unit (ReLU)

$$g(y) = \max(0, y)$$

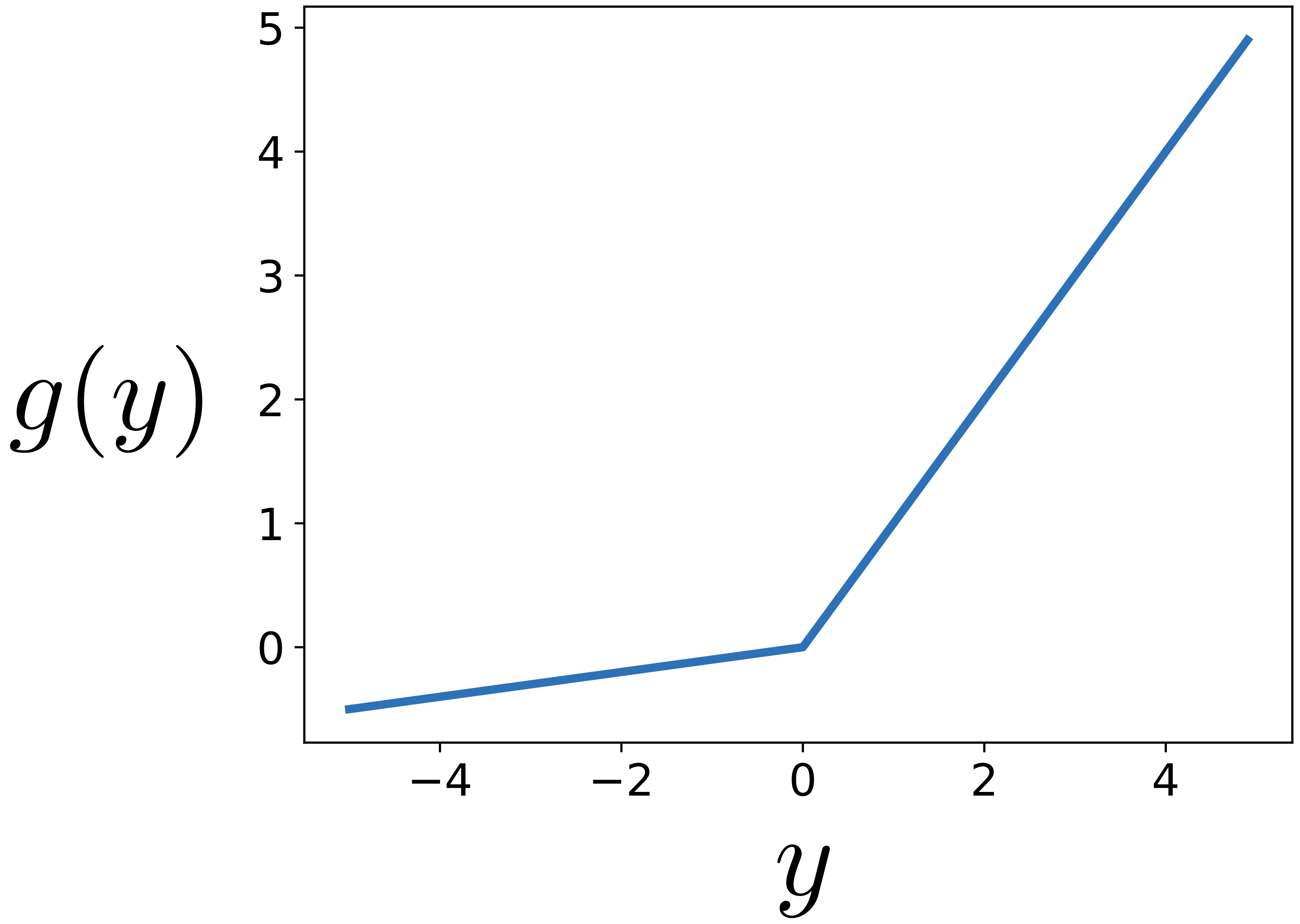


Computation in a neural net — nonlinearity

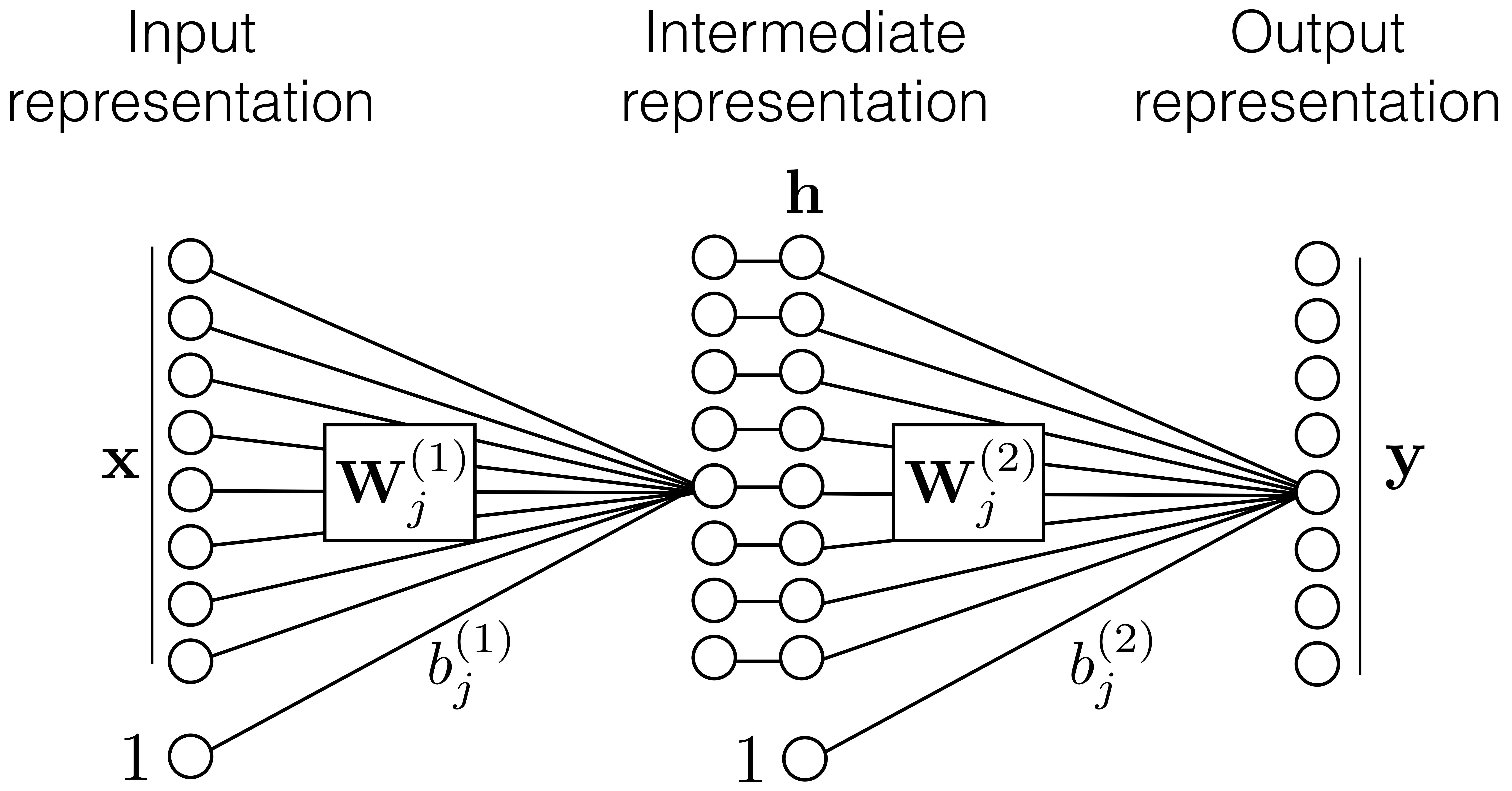
- where α is small (e.g. 0.02)
- Efficient to implement: $\frac{\partial g}{\partial y} = \begin{cases} -a, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$
- Also known as probabilistic ReLU (PReLU)
- Has non-zero gradients everywhere (unlike ReLU)
- α can also be learned (see Kaiming He et al. 2015).

Leaky ReLU

$$g(y) = \begin{cases} \max(0, y), & \text{if} \quad y \geq 0 \\ a \min(0, y), & \text{if} \quad y < 0 \end{cases}$$

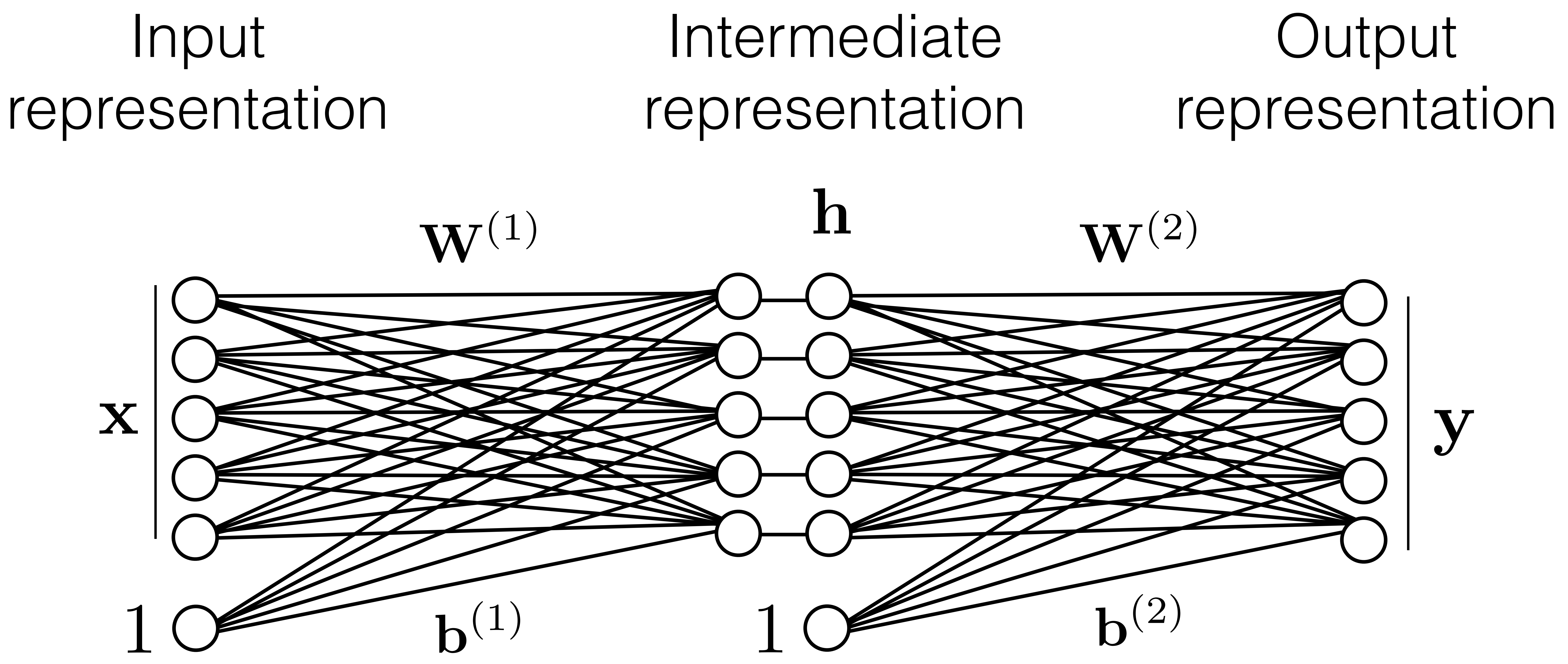


Stacking layers



$\mathbf{h}$ = **“hidden units”**

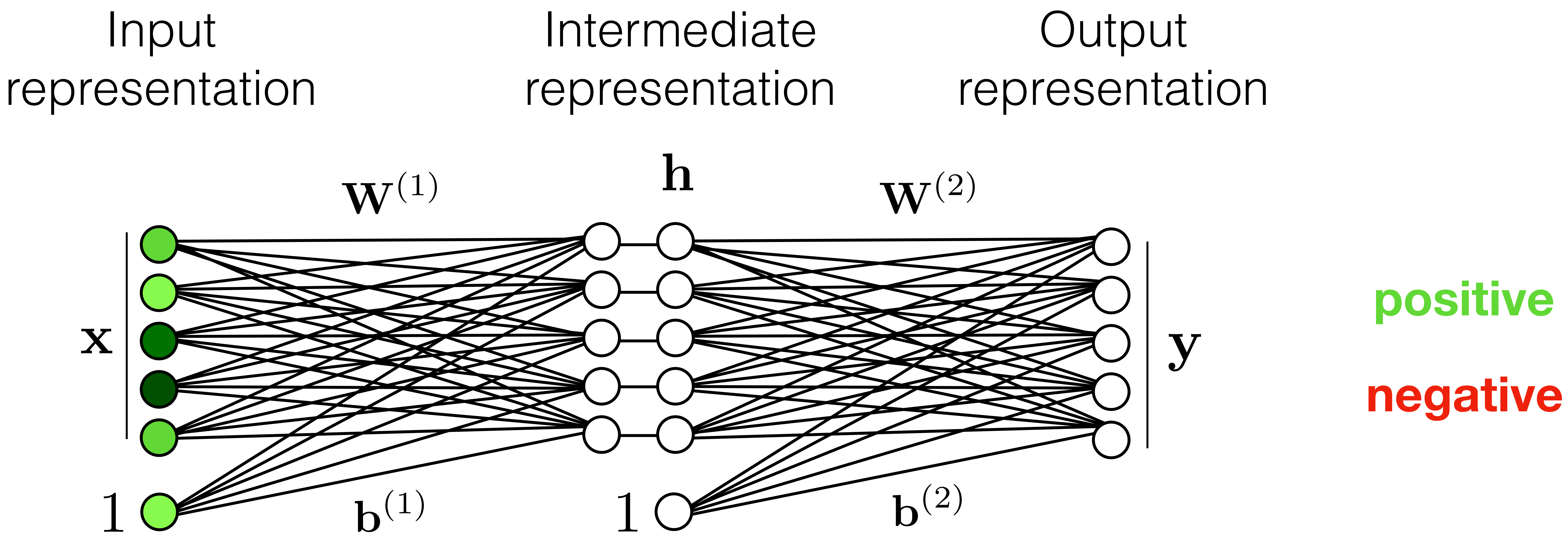
Stacking layers



$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

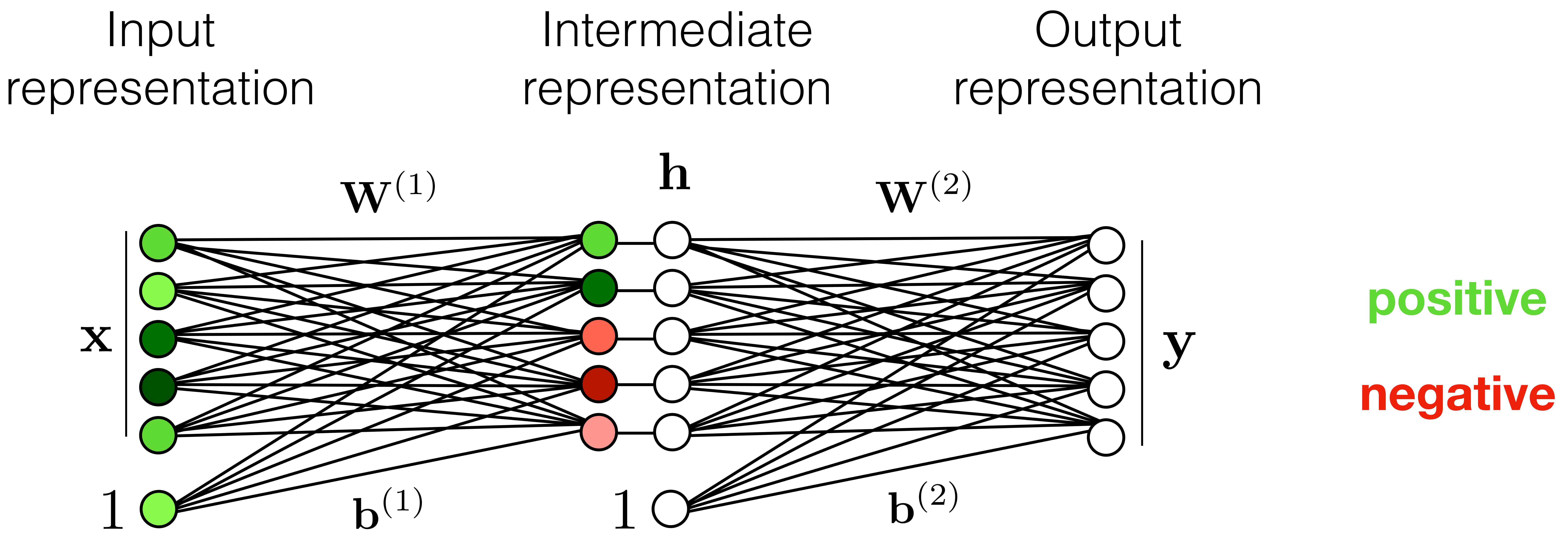
Stacking layers



$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

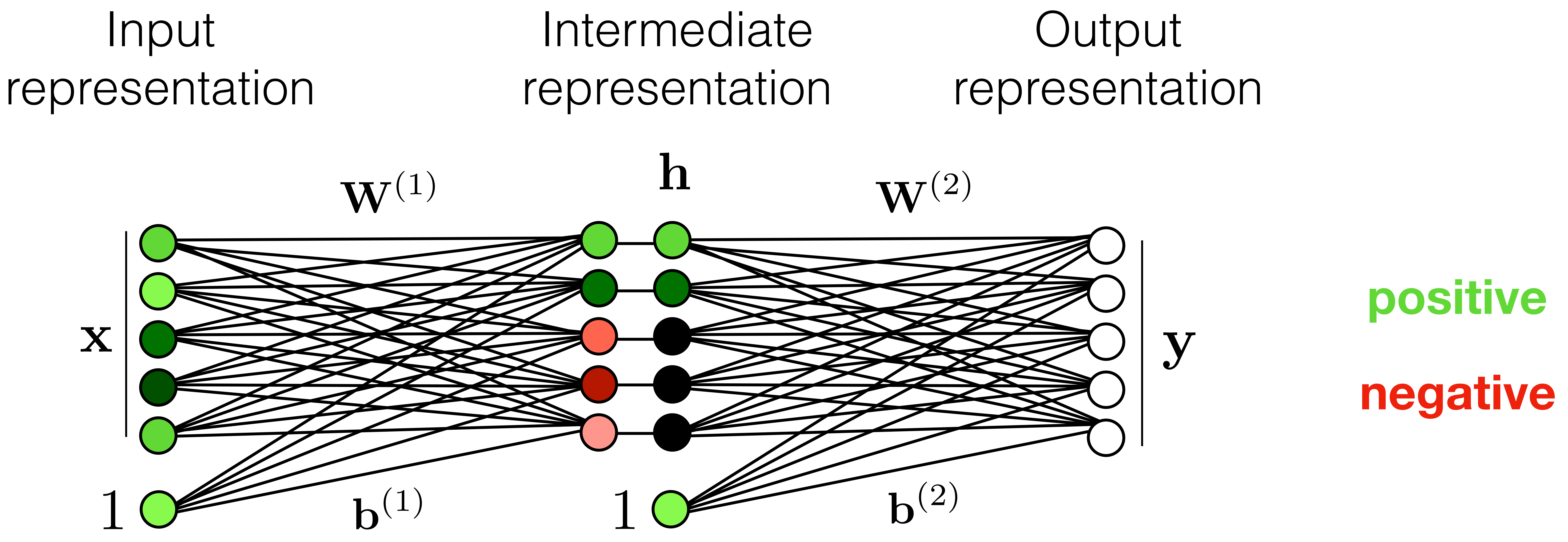
Stacking layers



$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

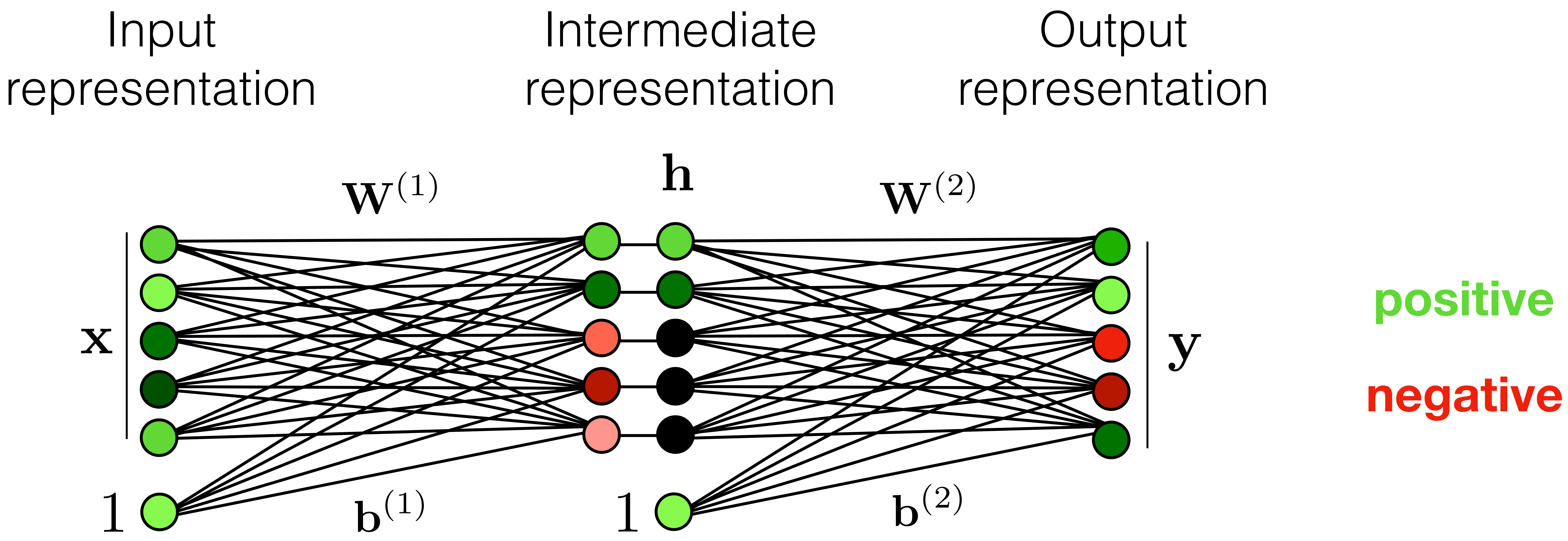
Stacking layers



$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

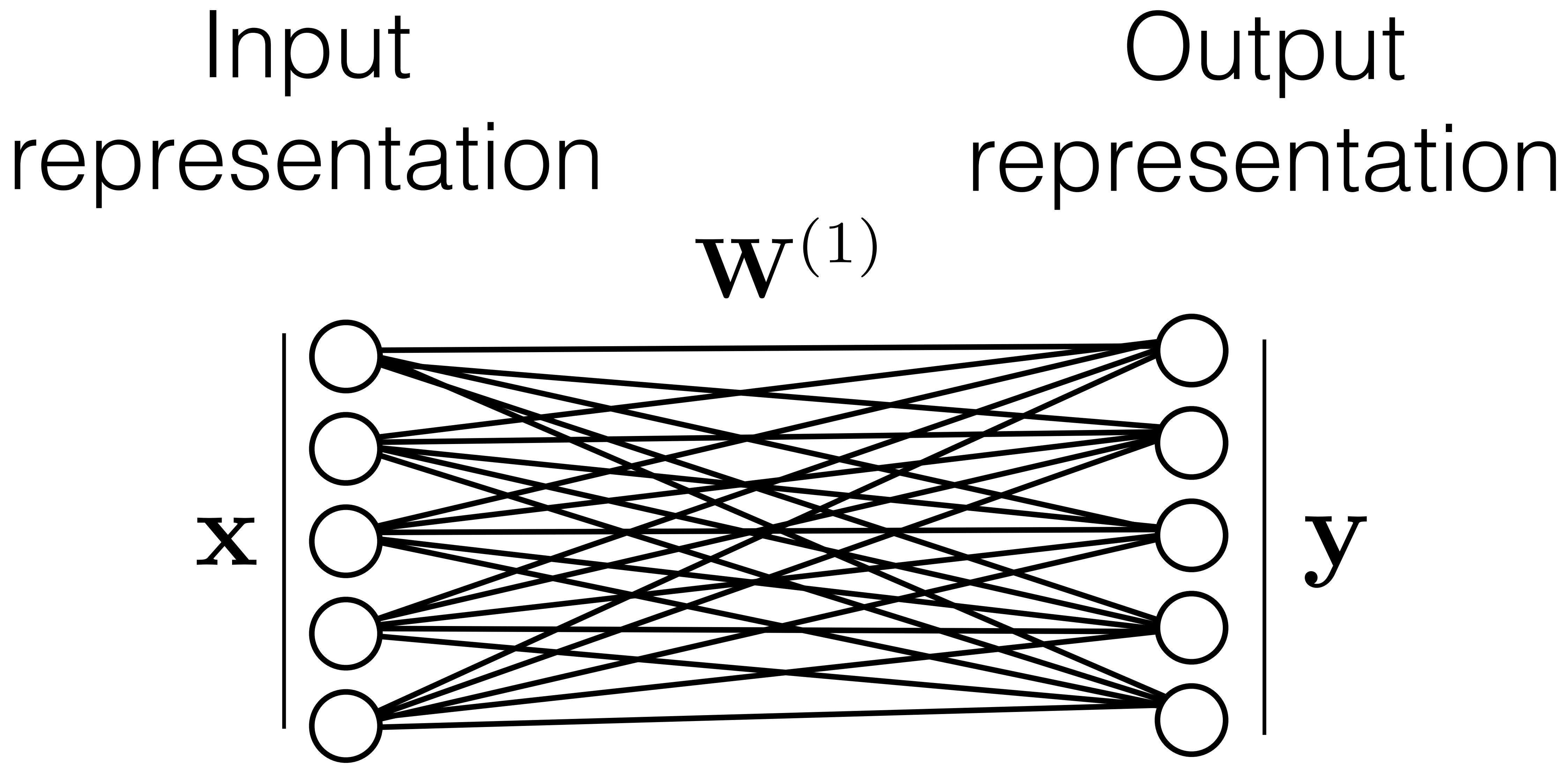
Stacking layers



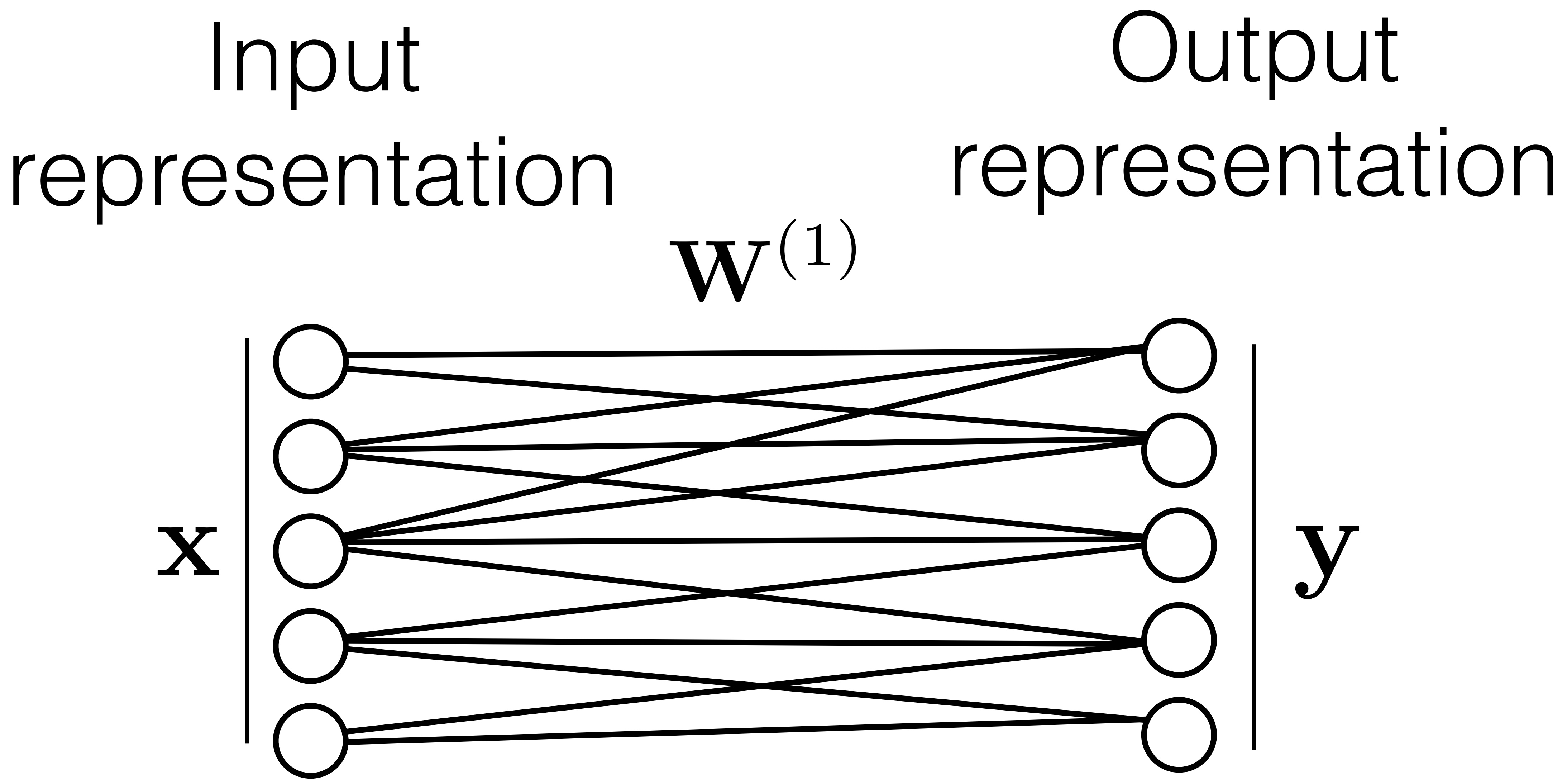
$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \qquad \mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}$$

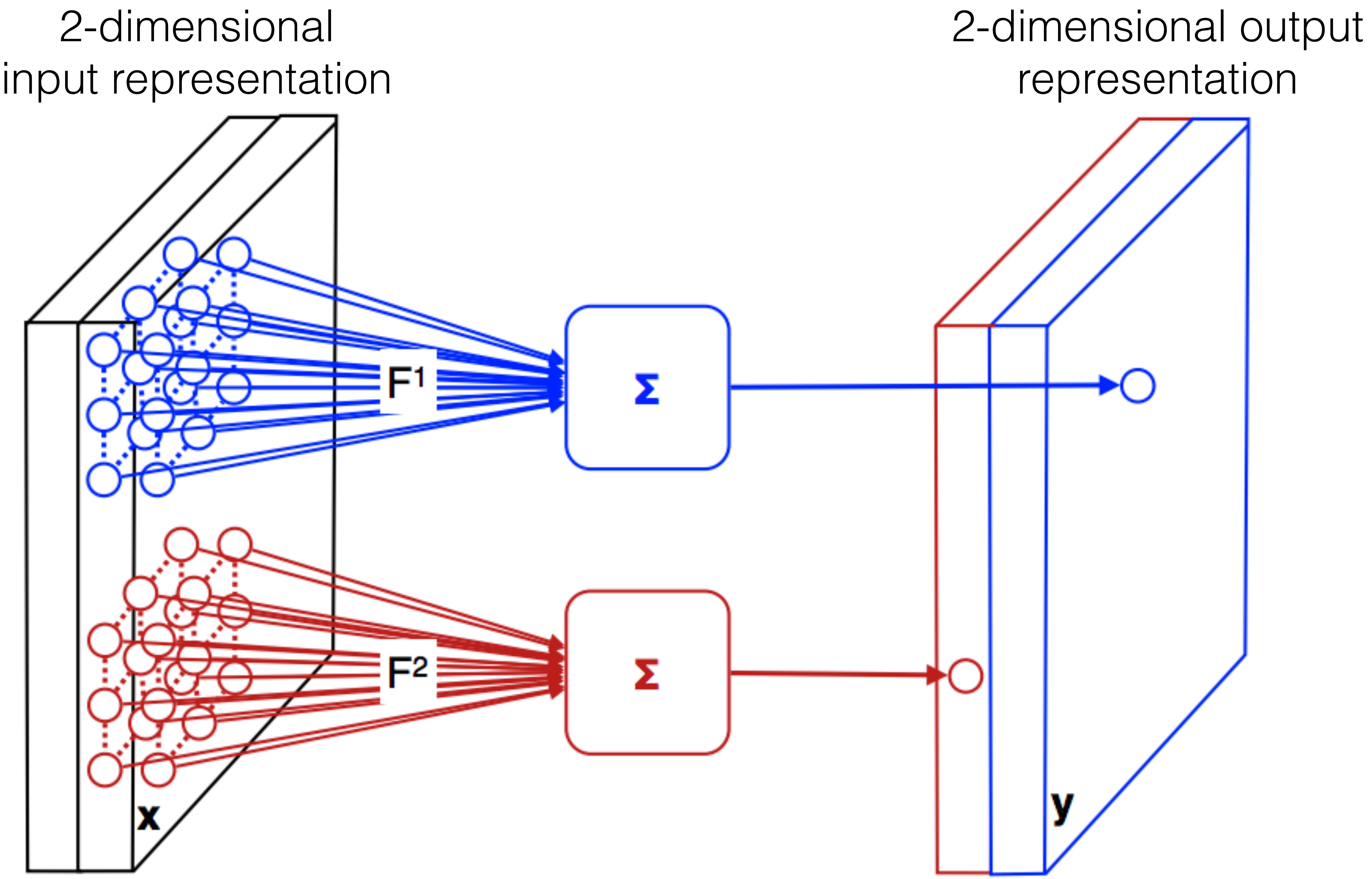
Connectivity patterns



Fully connected layer



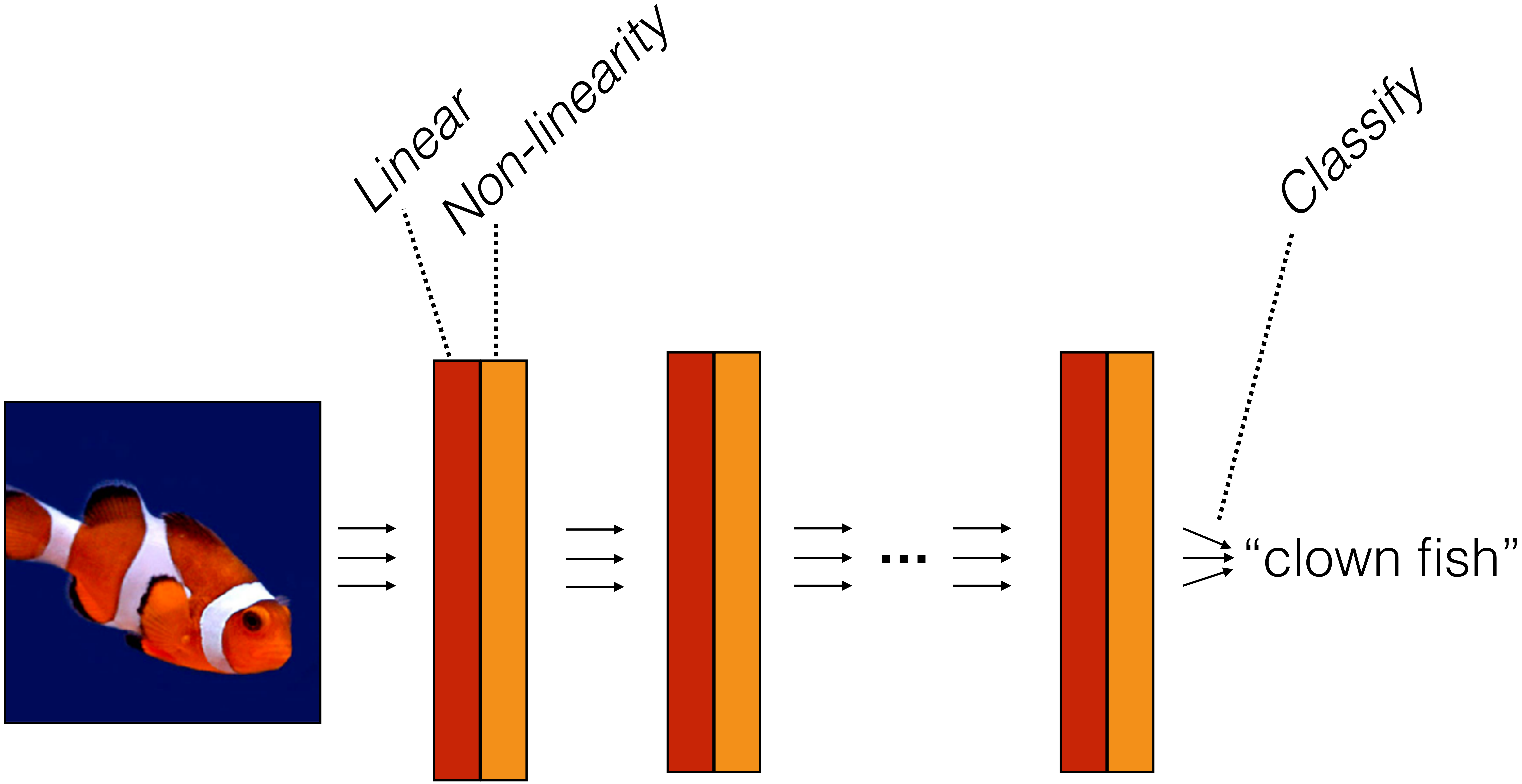
Locally connected layer
(Sparse W)



$$\mathbb{R}^{H \times W \times C^{(l)}} \rightarrow \mathbb{R}^{H \times W \times C^{(l+1)}}$$

[Figure from Andrea Vedaldi]

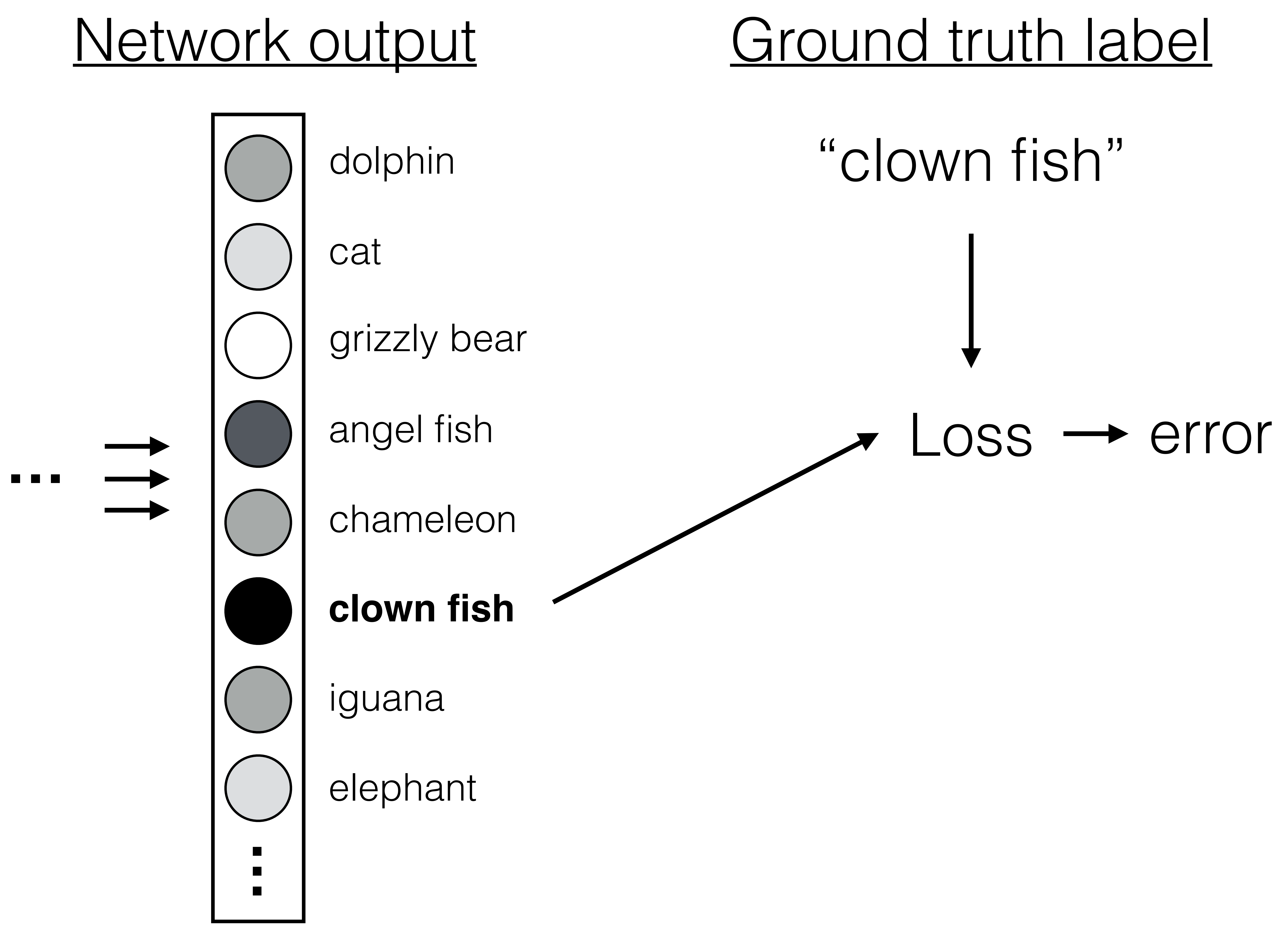
Deep Neural Networks



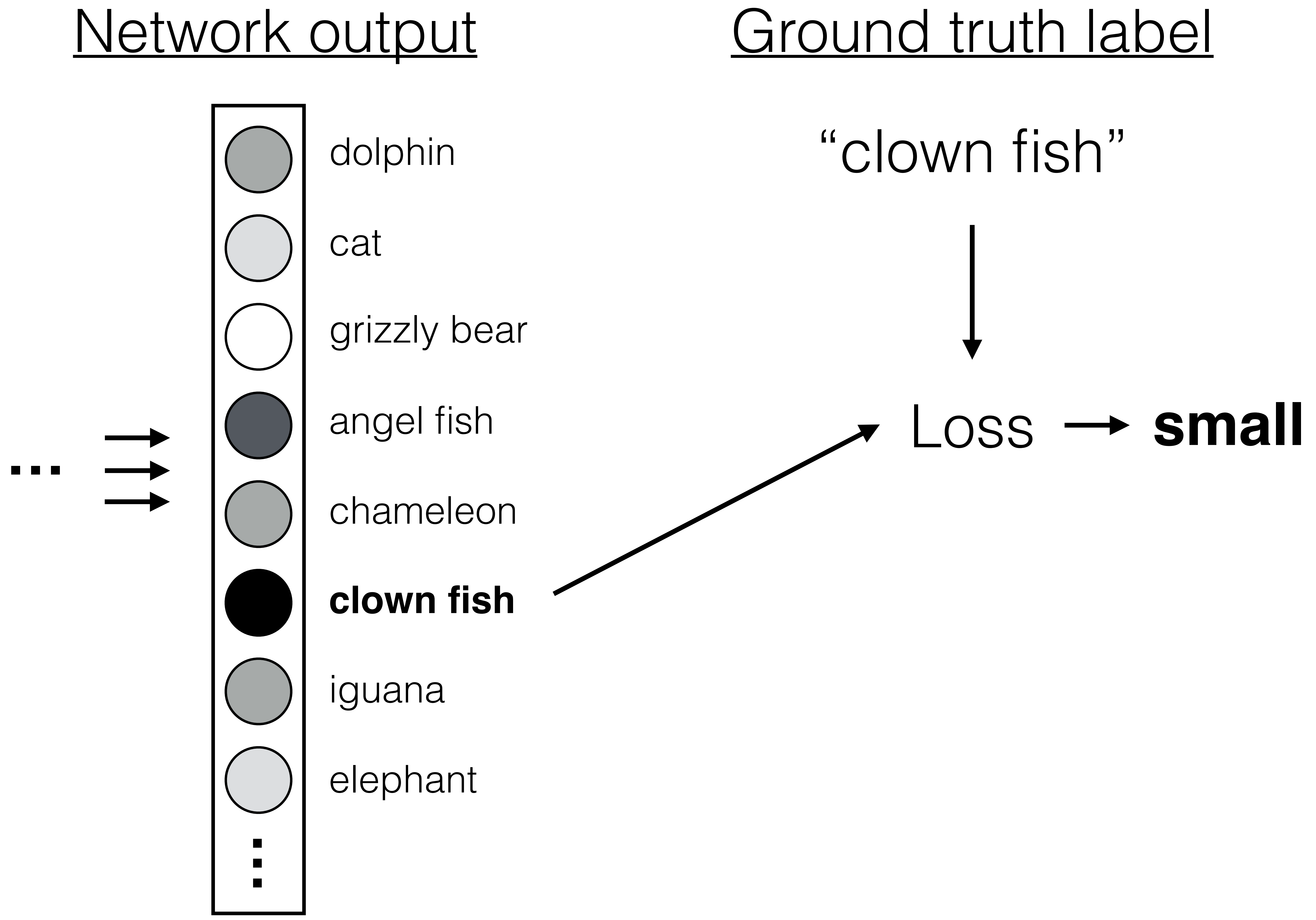
$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x})))$$

Classifier layer

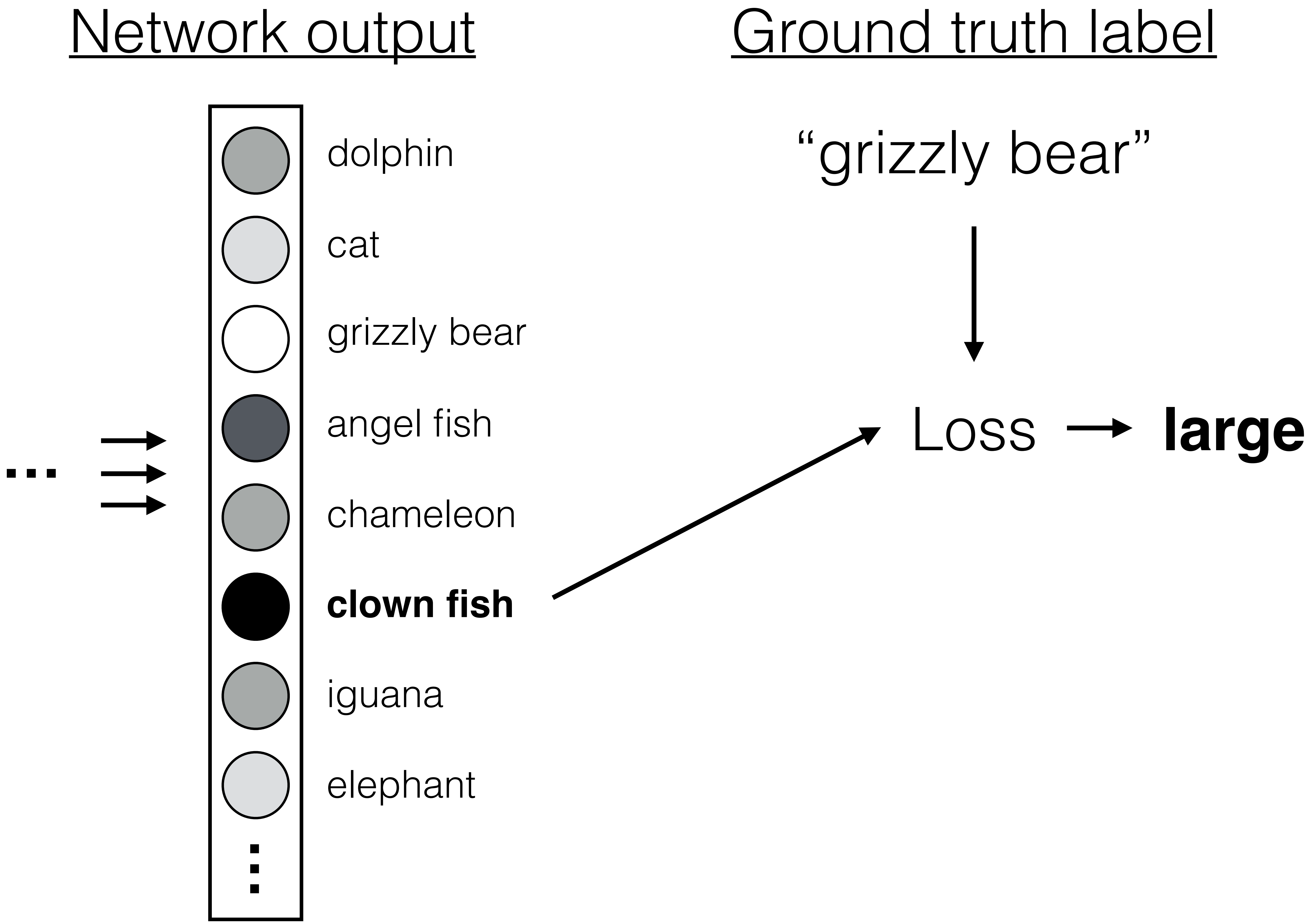
Loss function

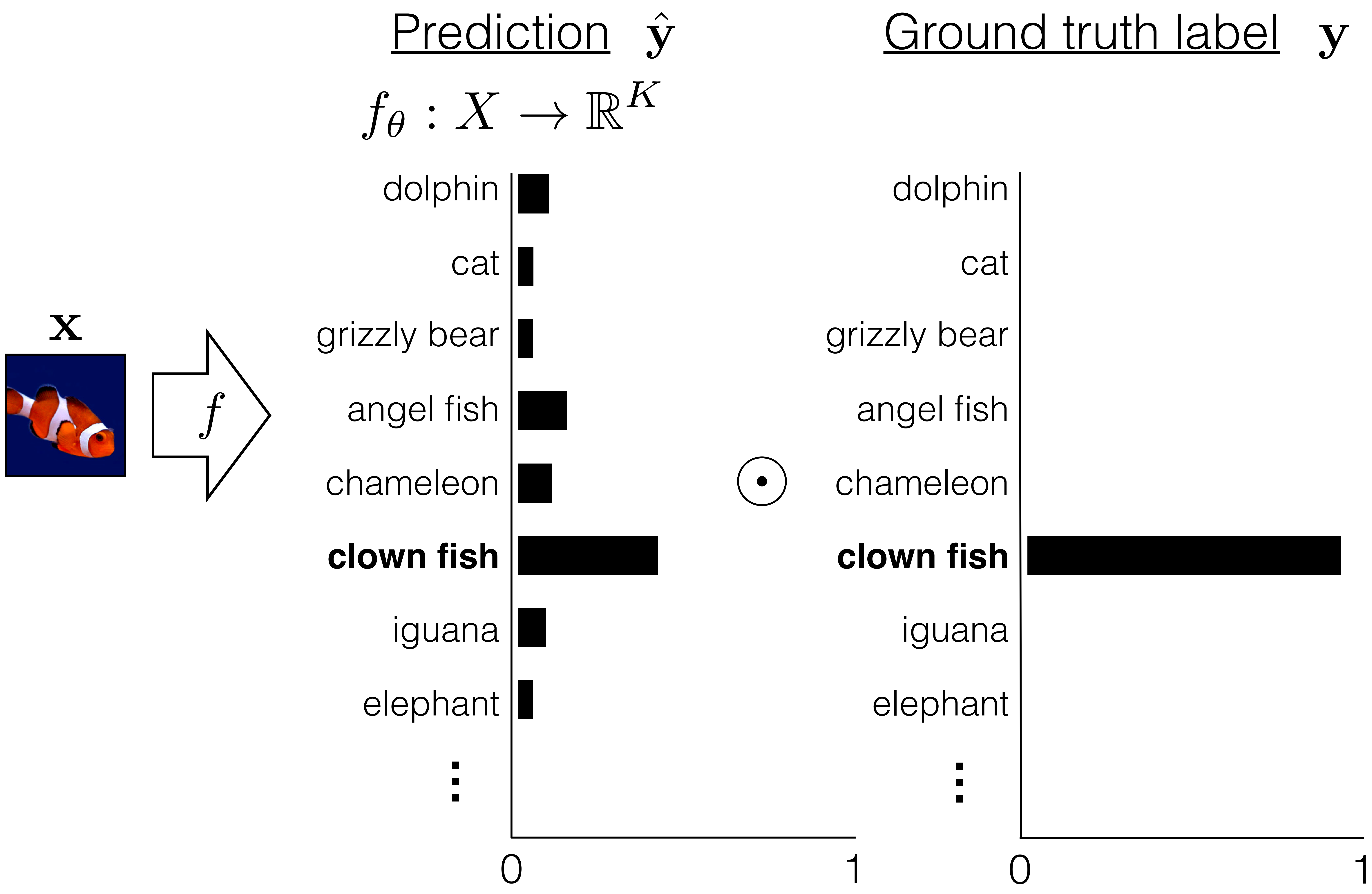


Loss function

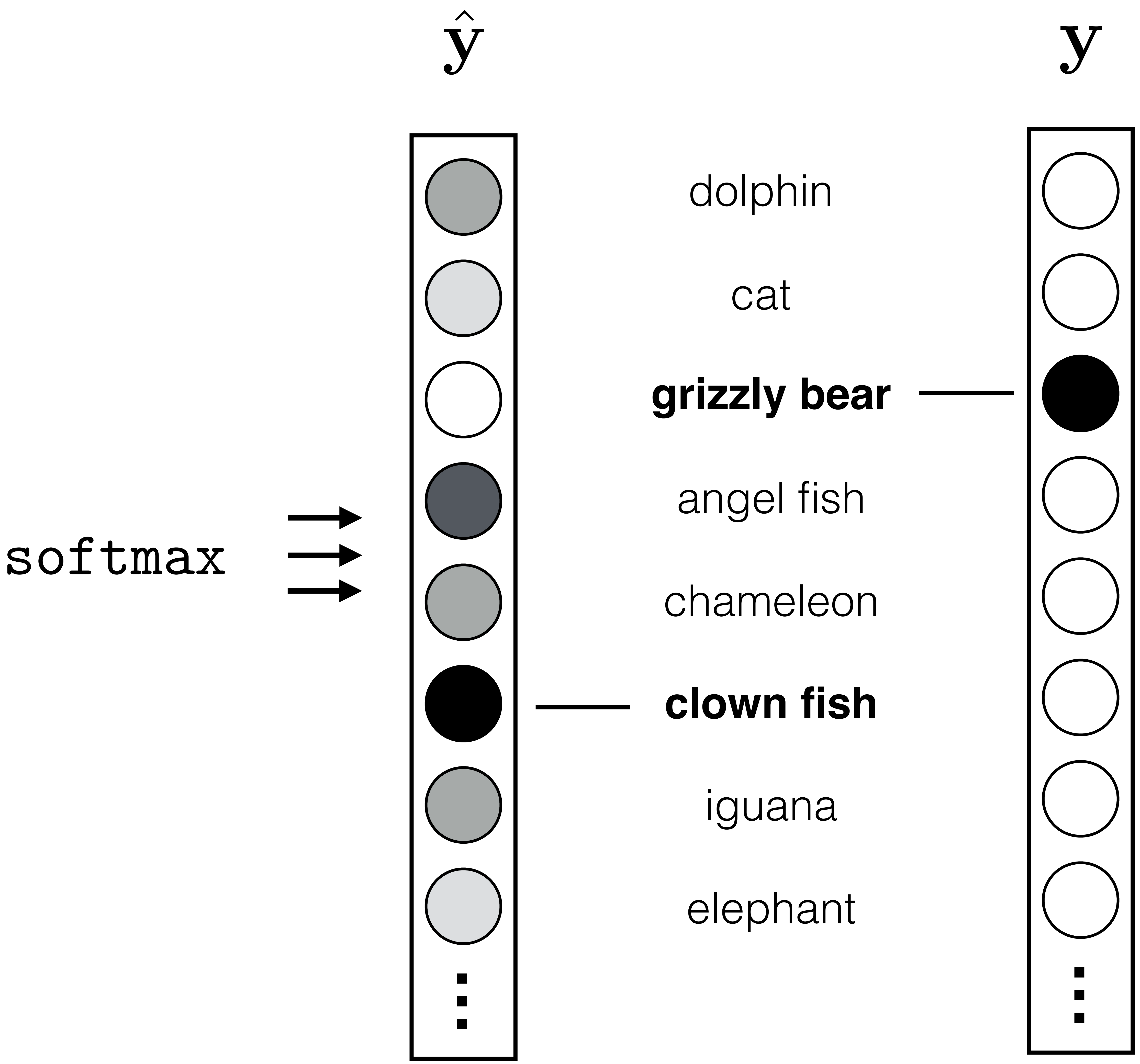


Loss function




Prediction $\hat{\mathbf{y}}$
Ground truth label $\mathbf{y}$
$f_\theta : X \to \mathbb{R}^K$
$\mathbf{x}$
f
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
0
1
$\odot$
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
0
1

Network output Ground truth label



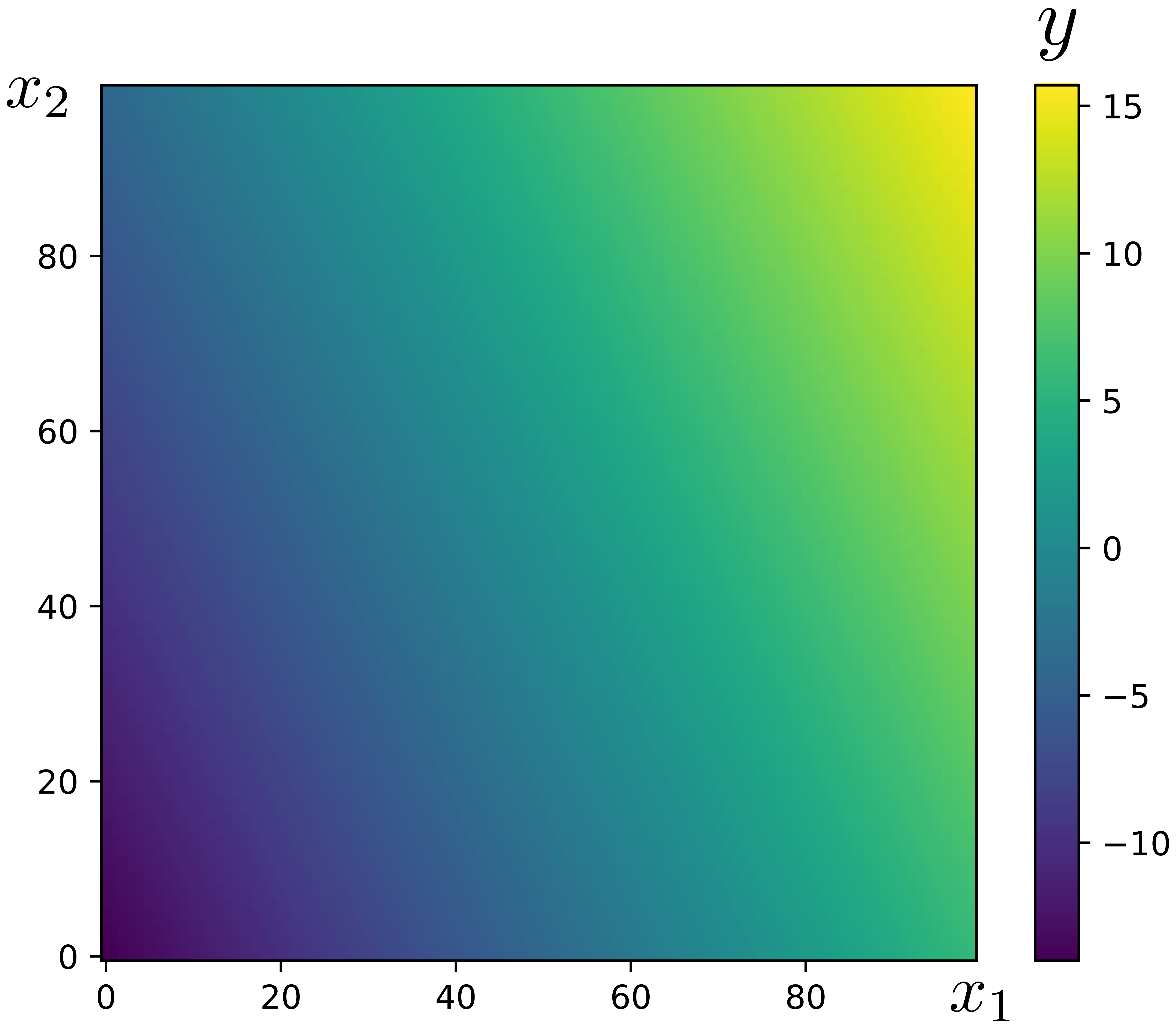
Probability of the observed data under the model

$$H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

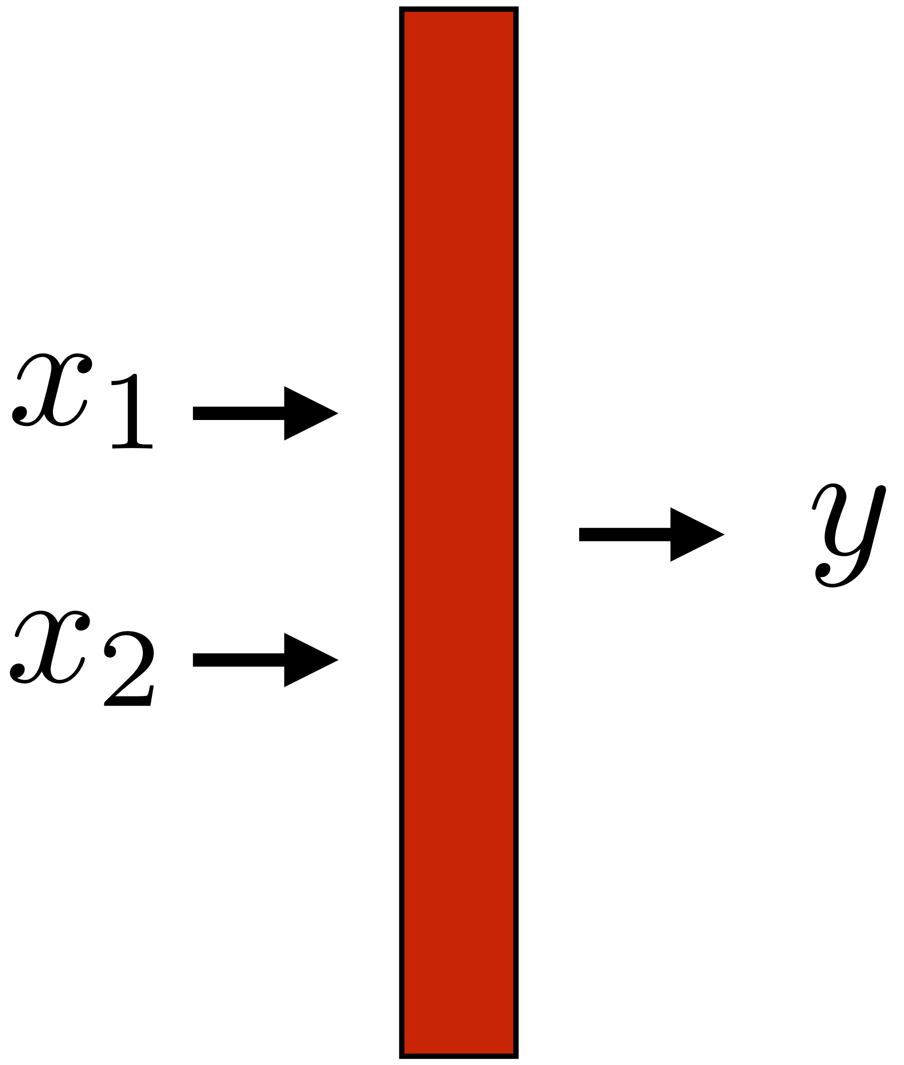
Representational power

- 1 layer? Linear decision surface.
- 2+ layers? In theory, can represent any function. Assuming non-trivial non-linearity.
  - Bengio 2009, http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf
  - Bengio, Courville, Goodfellow book http://www.deeplearningbook.org/contents/mlp.html
  - Simple proof by M. Neilsen http://neuralnetworksanddeeplearning.com/chap4.html
  - D. Mackay book http://www.inference.phy.cam.ac.uk/mackay/itprnn/ps/482.491.pdf
- But issue is efficiency: very wide two layers vs narrow deep model? In practice, more layers helps.

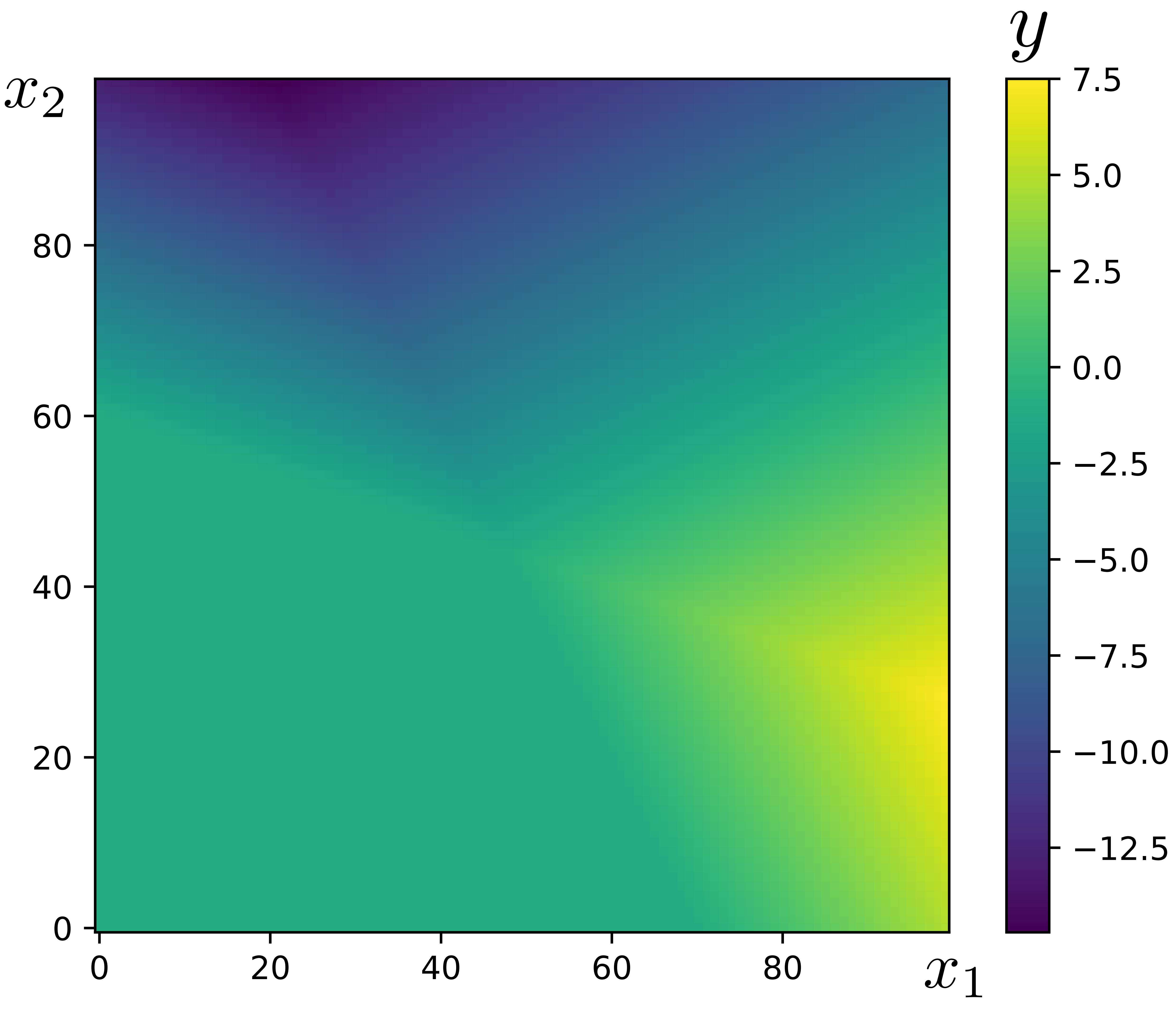
Example: linear classification with a perceptron



$$y = \mathbf{x}^T \mathbf{w} + b$$

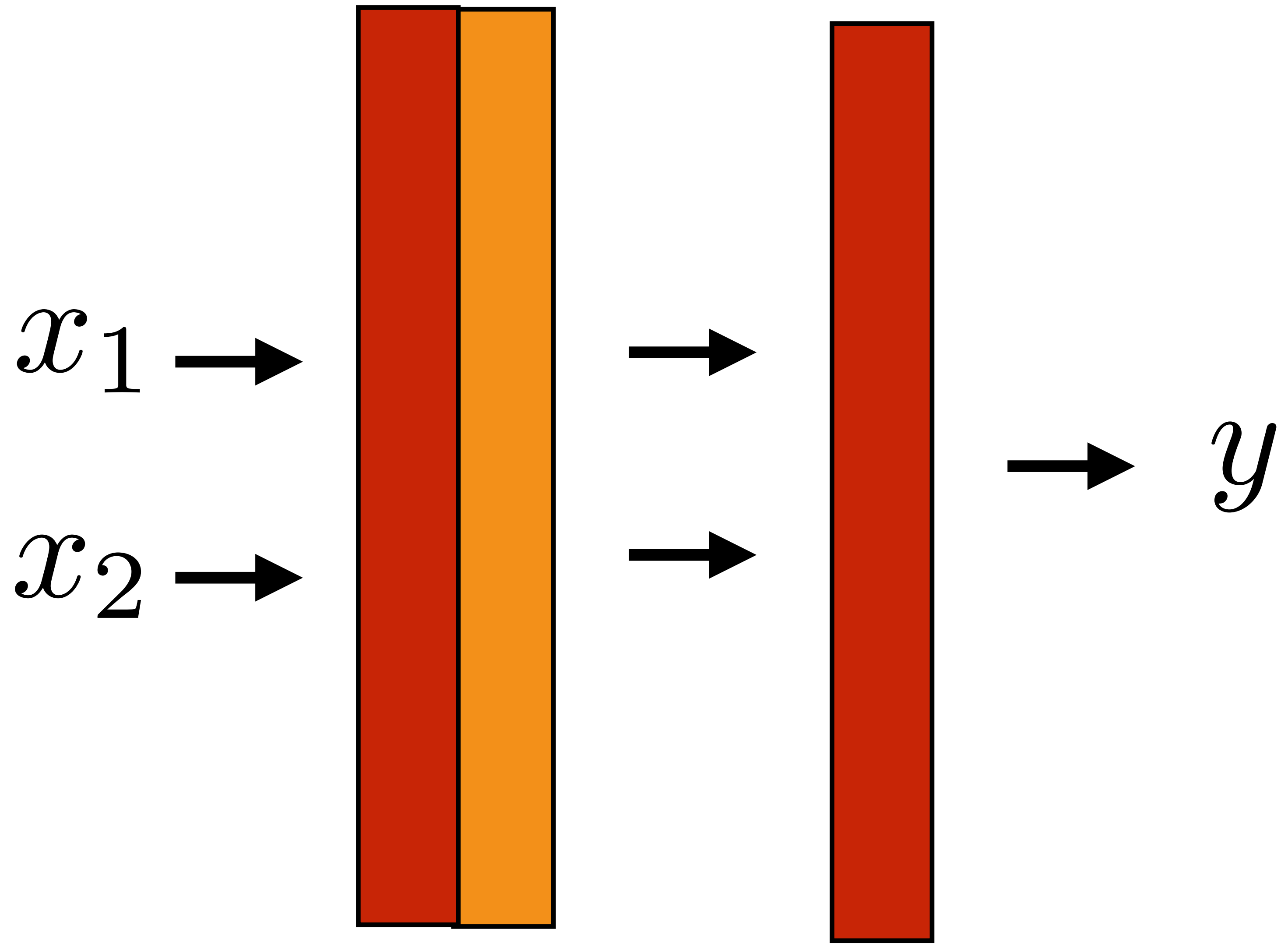


Example: nonlinear classification with a deep net net



$$\mathbf{h} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

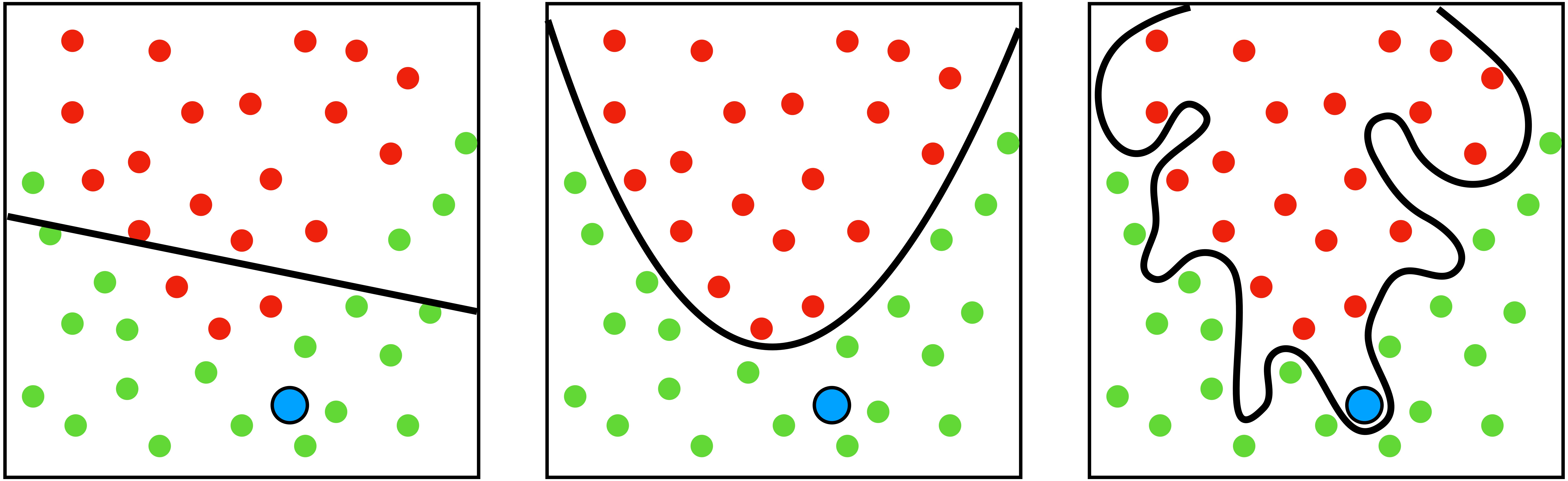
$$y = \mathbf{W}^{(2)}\mathbf{h} + b^{(2)}$$



[http://playground.tensorflow.org]

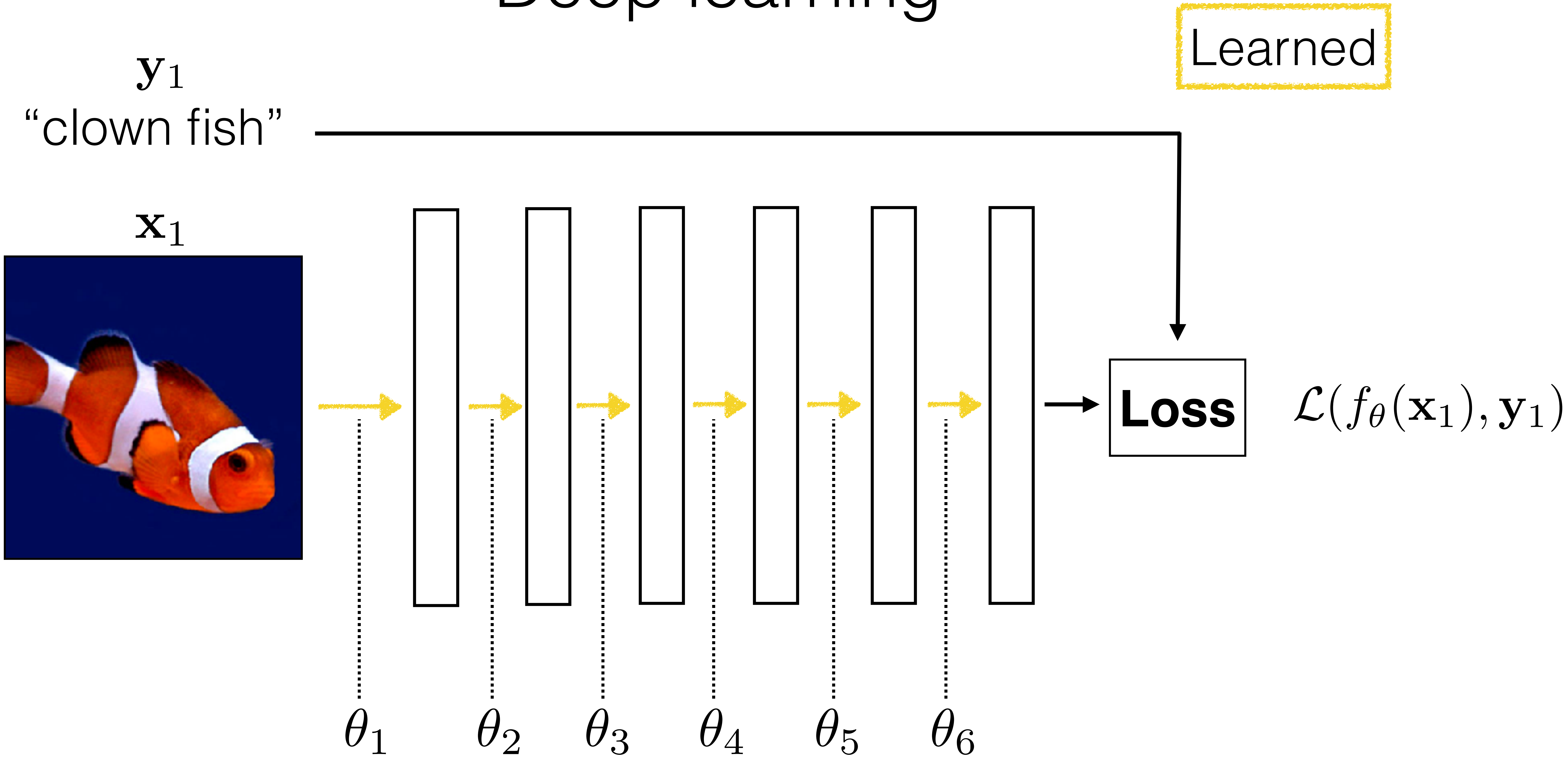
Example: nonlinear classification with a deep net

What class is ?



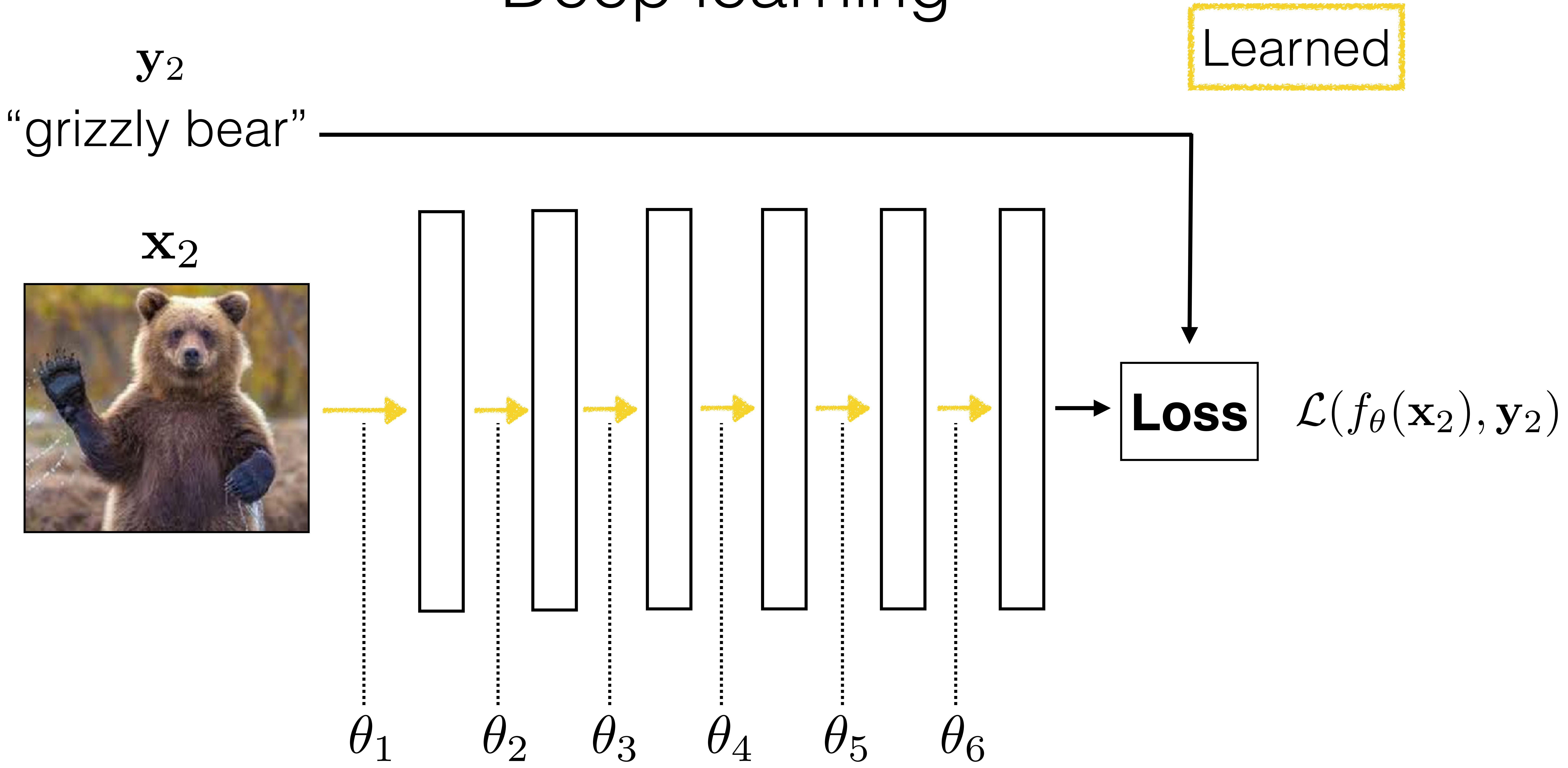
Answer:	Answer:	Answer:
Underfitting	Appropriate model	Overfitting

Deep learning



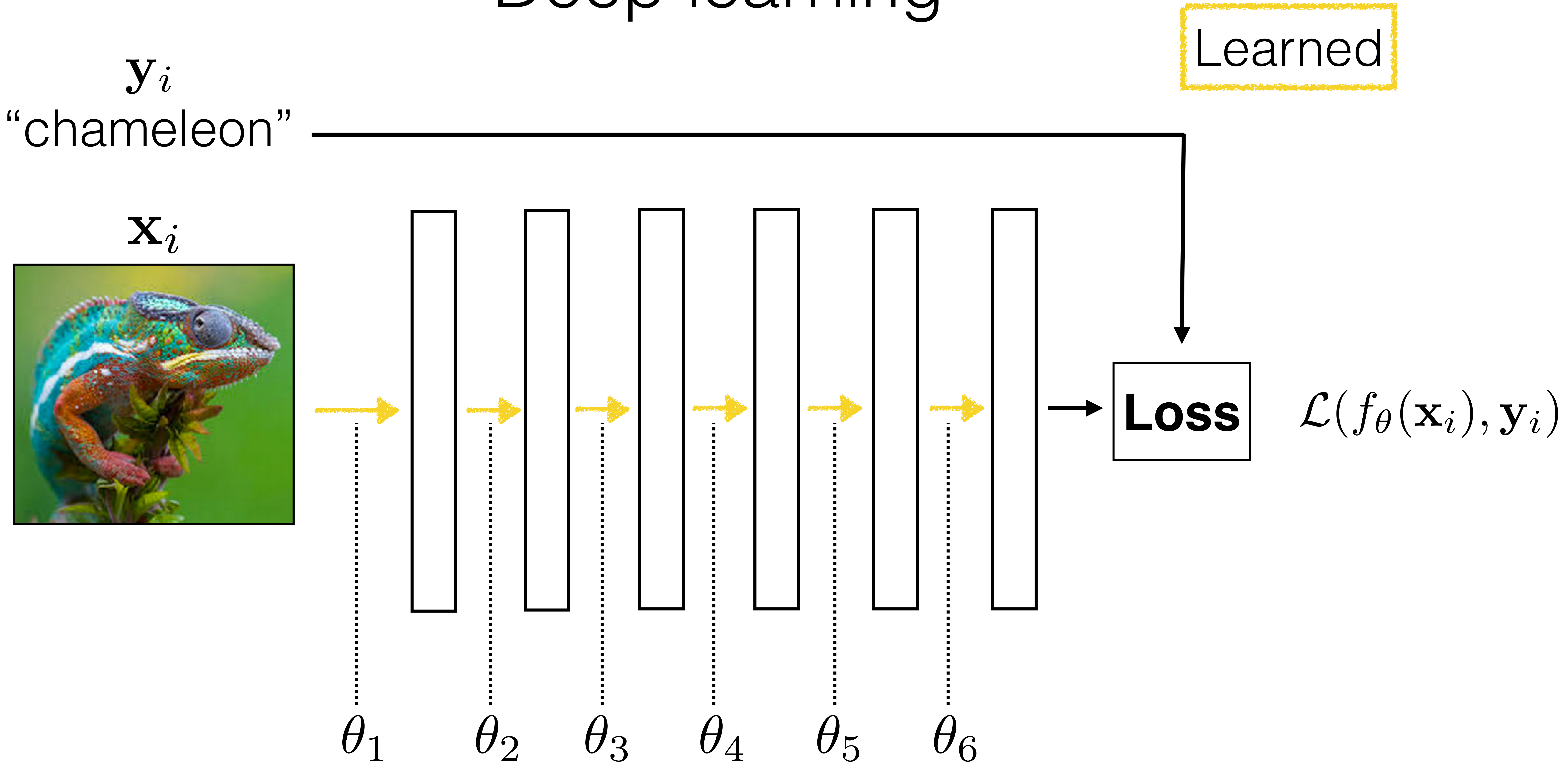
$$\boxed{\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)}$$

Deep learning



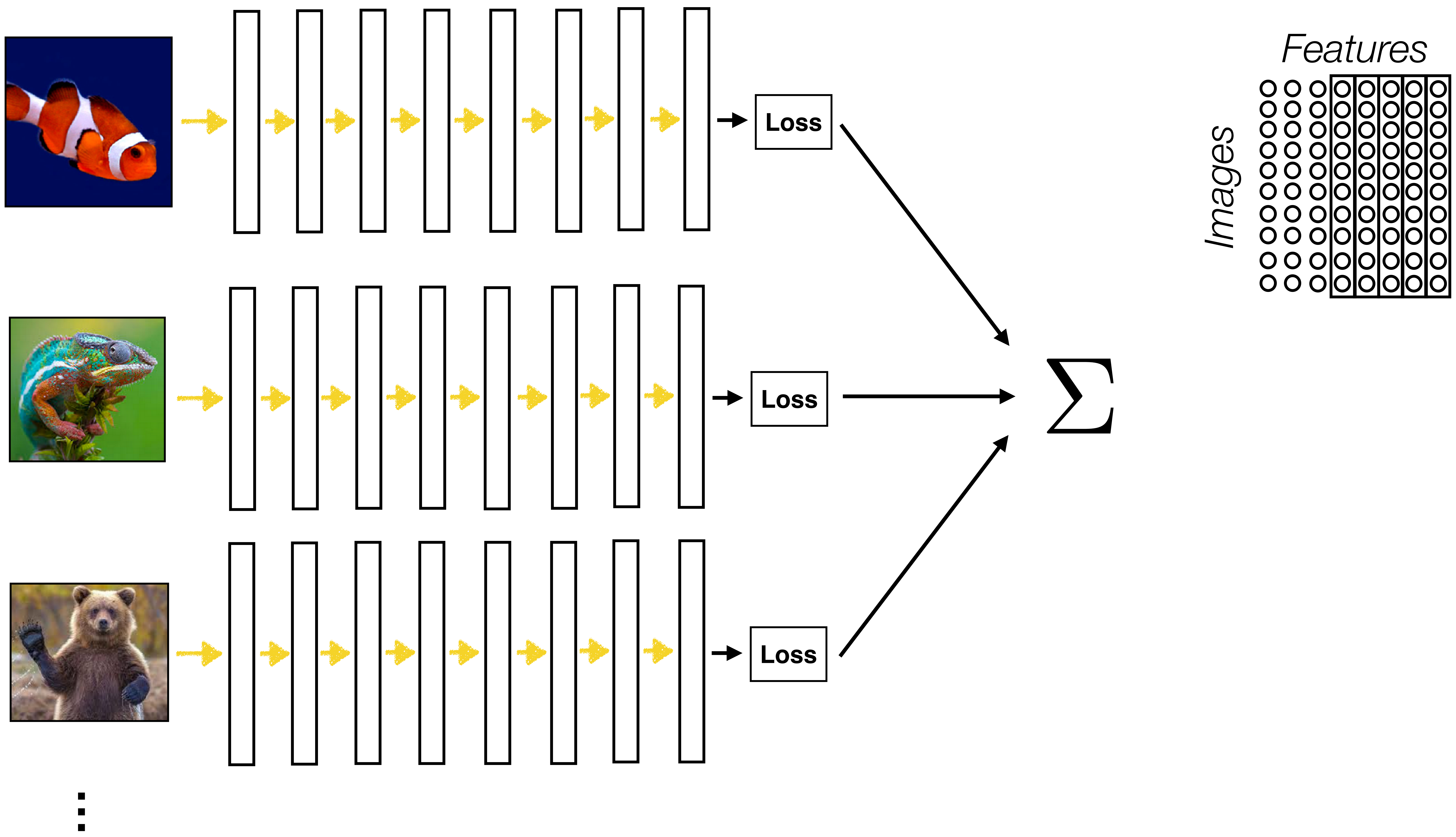
$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$

Deep learning



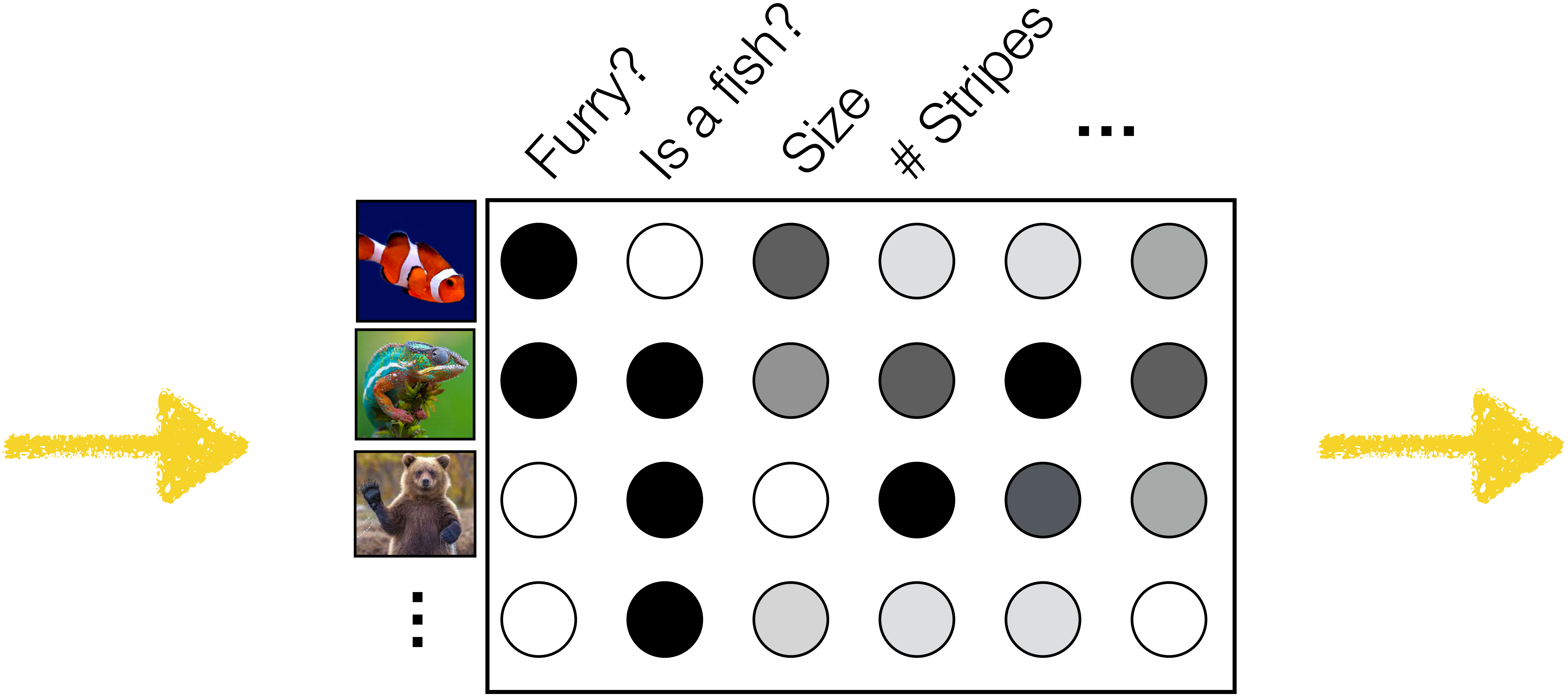
$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$

Batch (parallel) processing



Tensors

(multi-dimensional arrays)



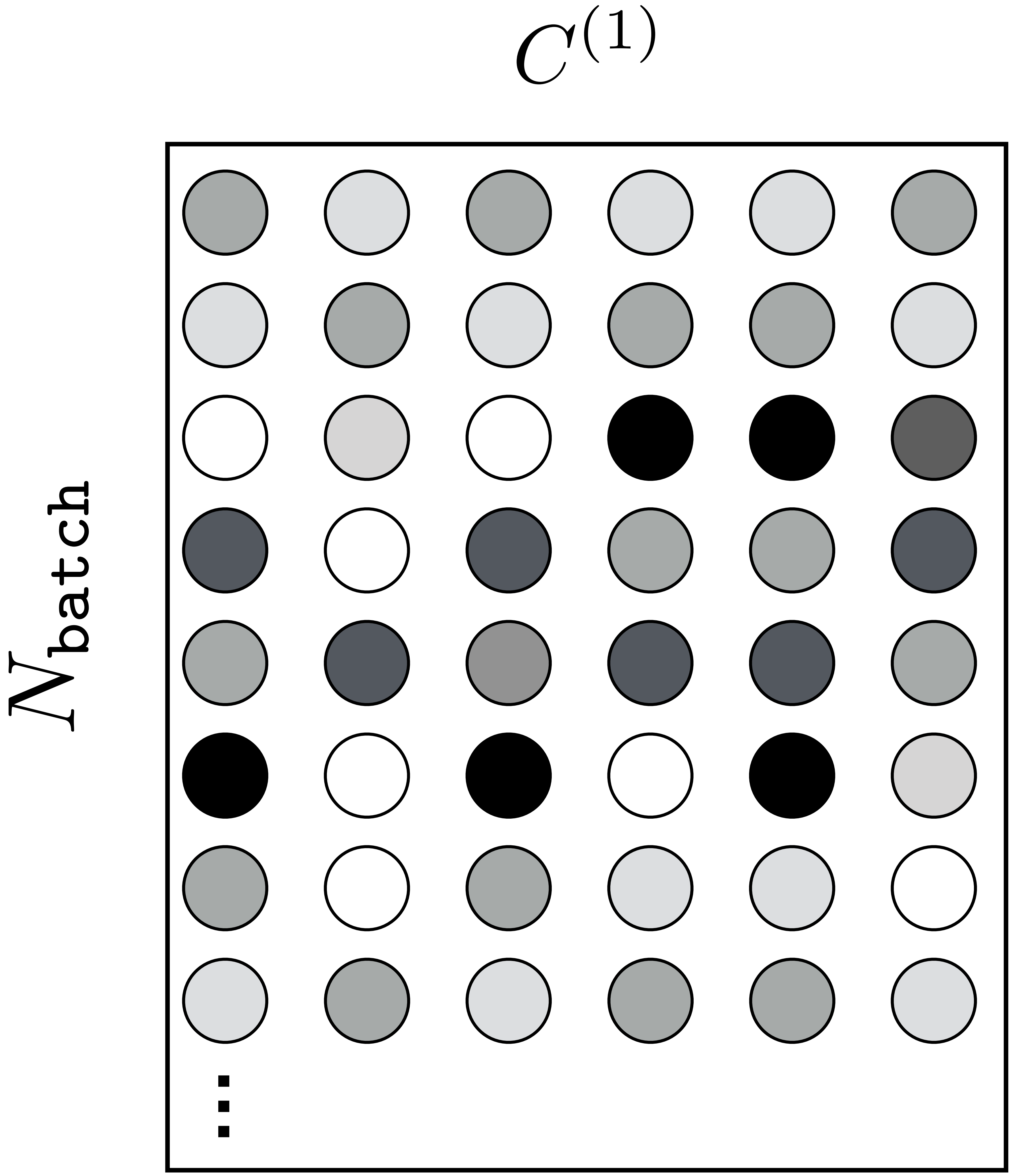
Each layer is a representation of the data

Tensors

(multi-dimensional arrays)

$$\mathbf{h}^{(1)} \in \mathbb{R}^{N_{\texttt{batch}} \times C^{(1)}}$$

neurons
features
units
“channels”



Tensors

(multi-dimensional arrays)

$$\mathbf{h}^{(1)} \in \mathbb{R}^{N_{\texttt{batch}} \times C^{(1)}}$$

\# neurons
\# features
\# units
\# "channels"

$C^{(1)}$

$N_{\texttt{batch}}$

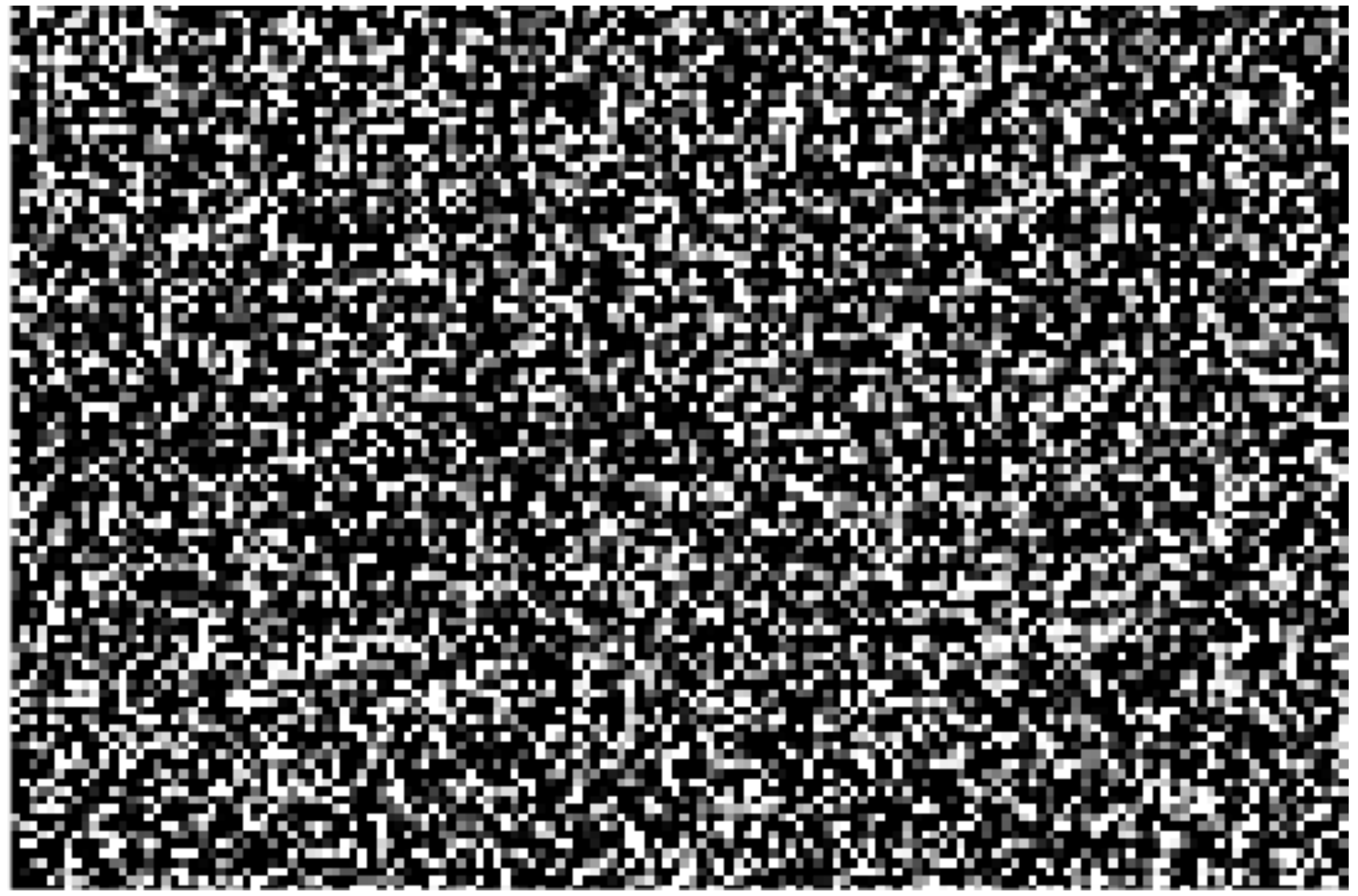
"Tensor flow"

$\mathbf{h}^{(1)} \in \mathbb{R}^{N_{\texttt{batch}} \times C^{(1)}}$

$\mathbf{h}^{(2)} \in \mathbb{R}^{N_{\texttt{batch}} \times C^{(2)}}$

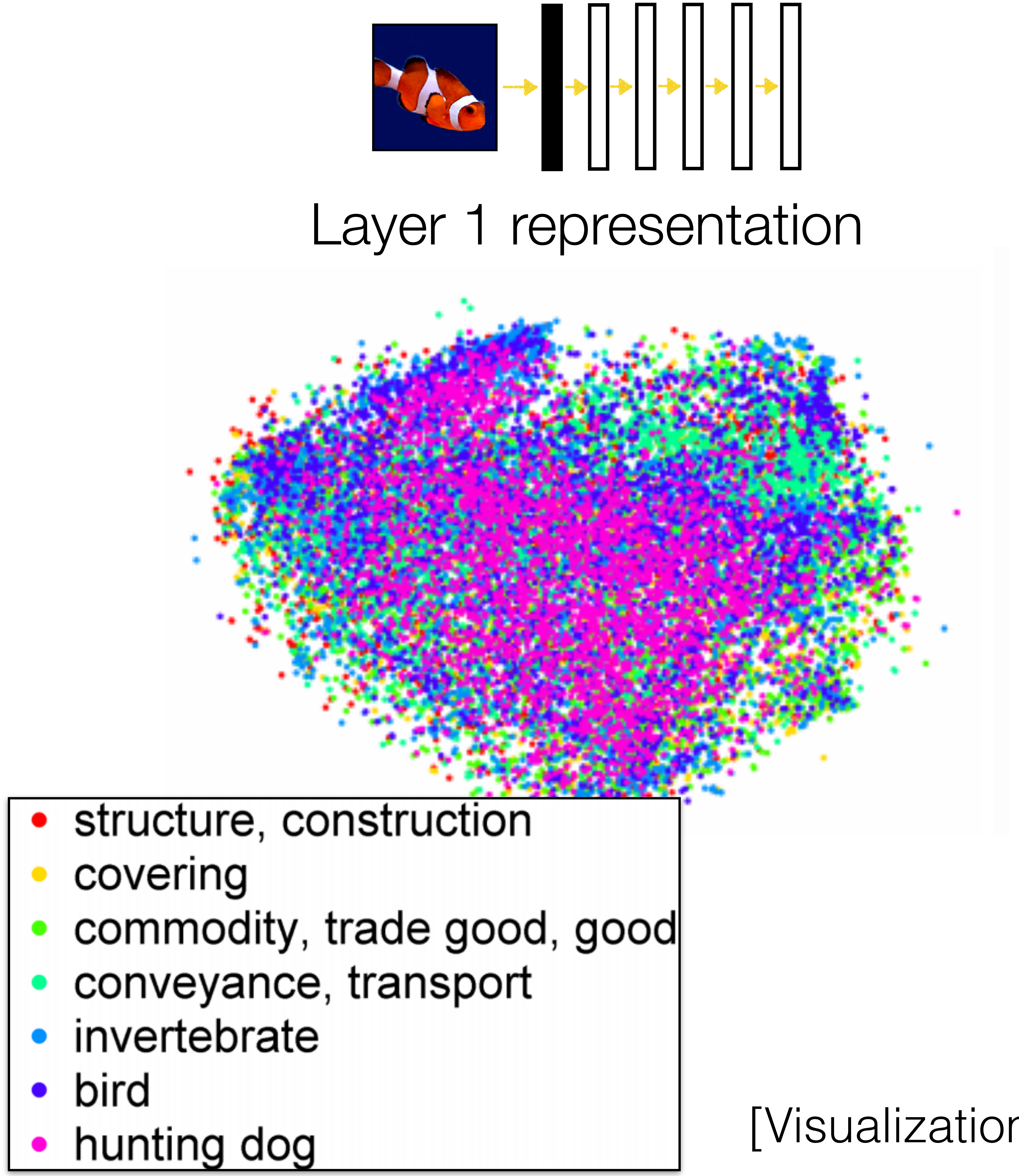
$C^{(1)}$

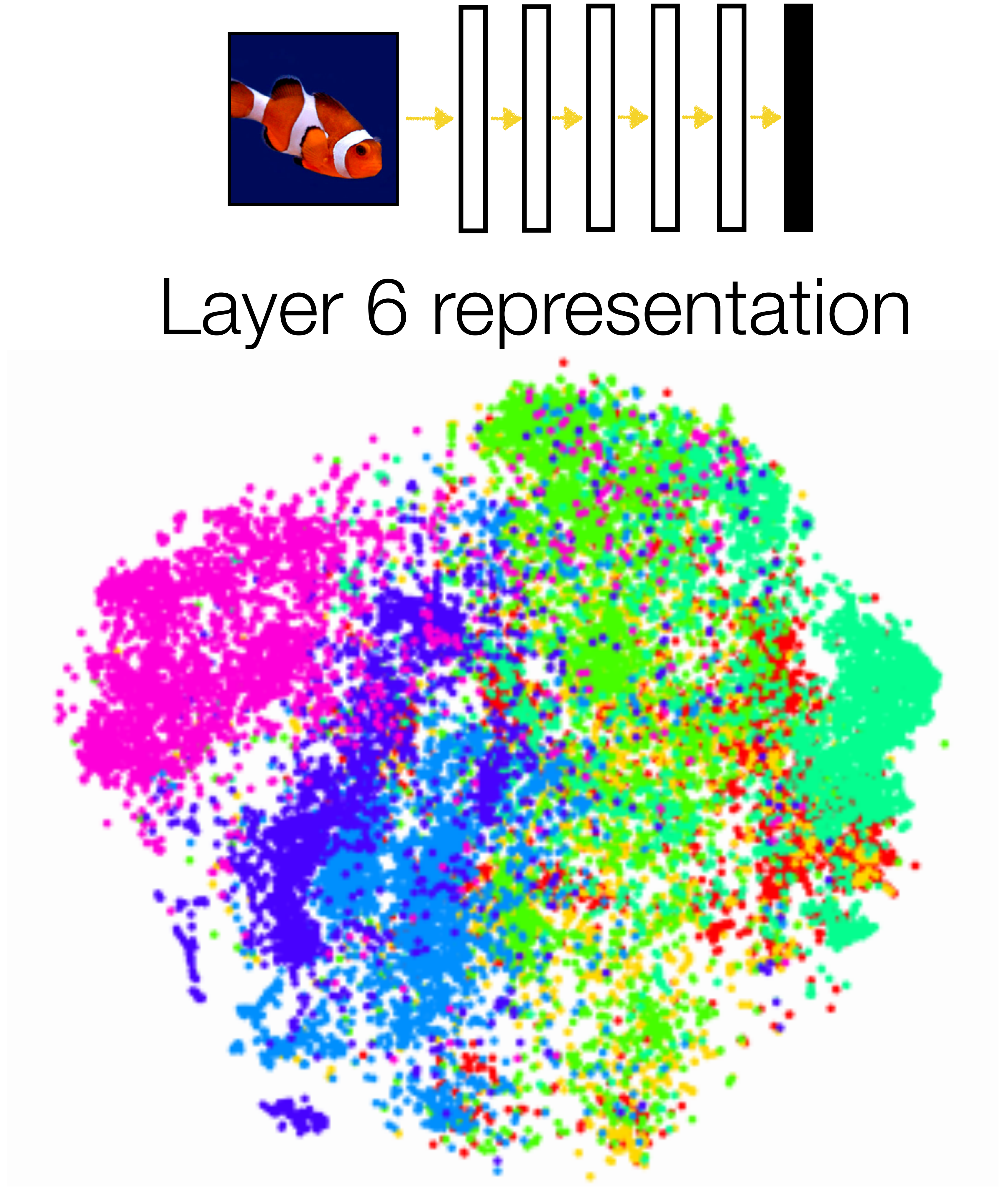
$N_{\texttt{batch}}$



$C^{(2)}$

$N_{\texttt{batch}}$



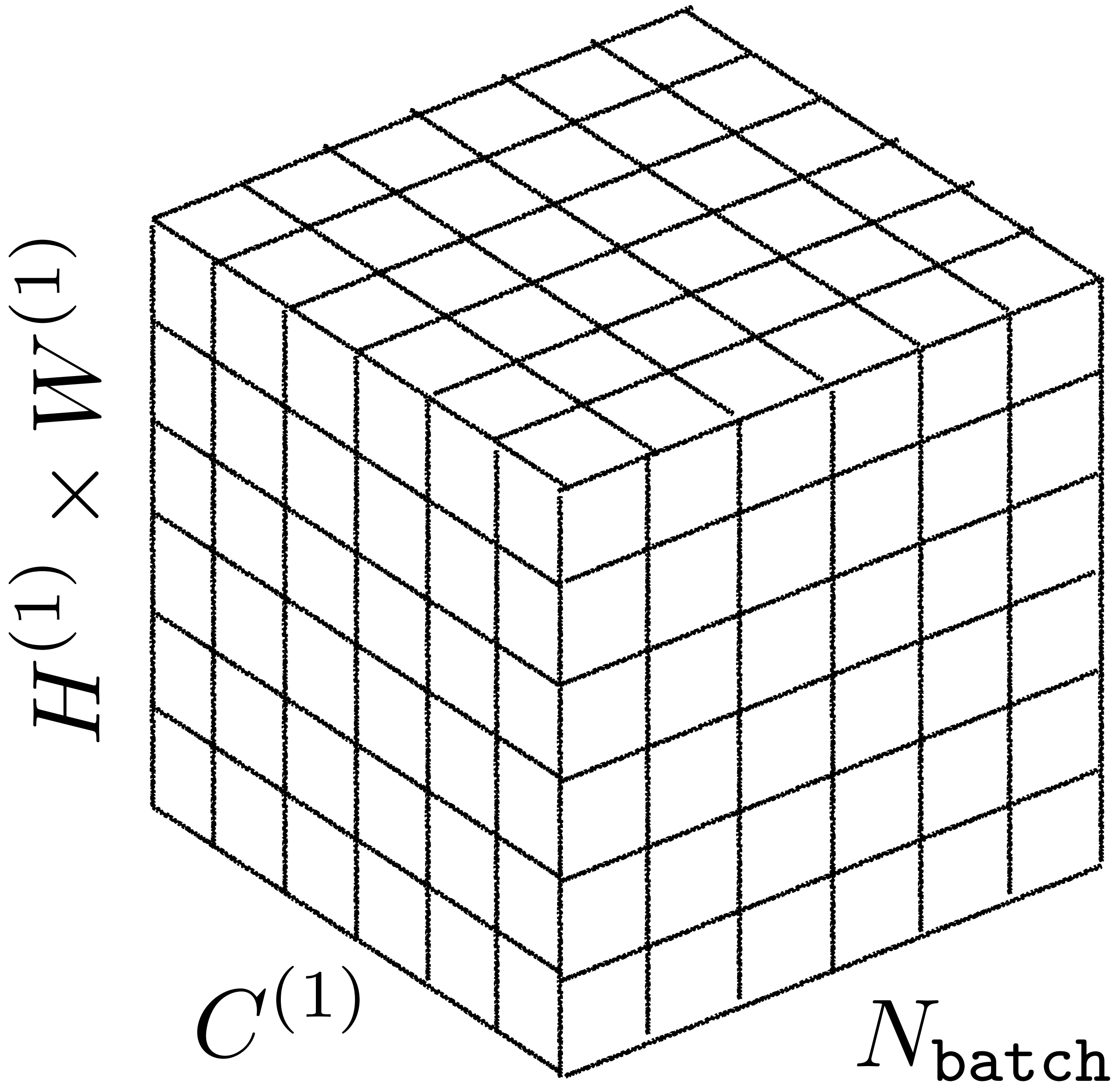


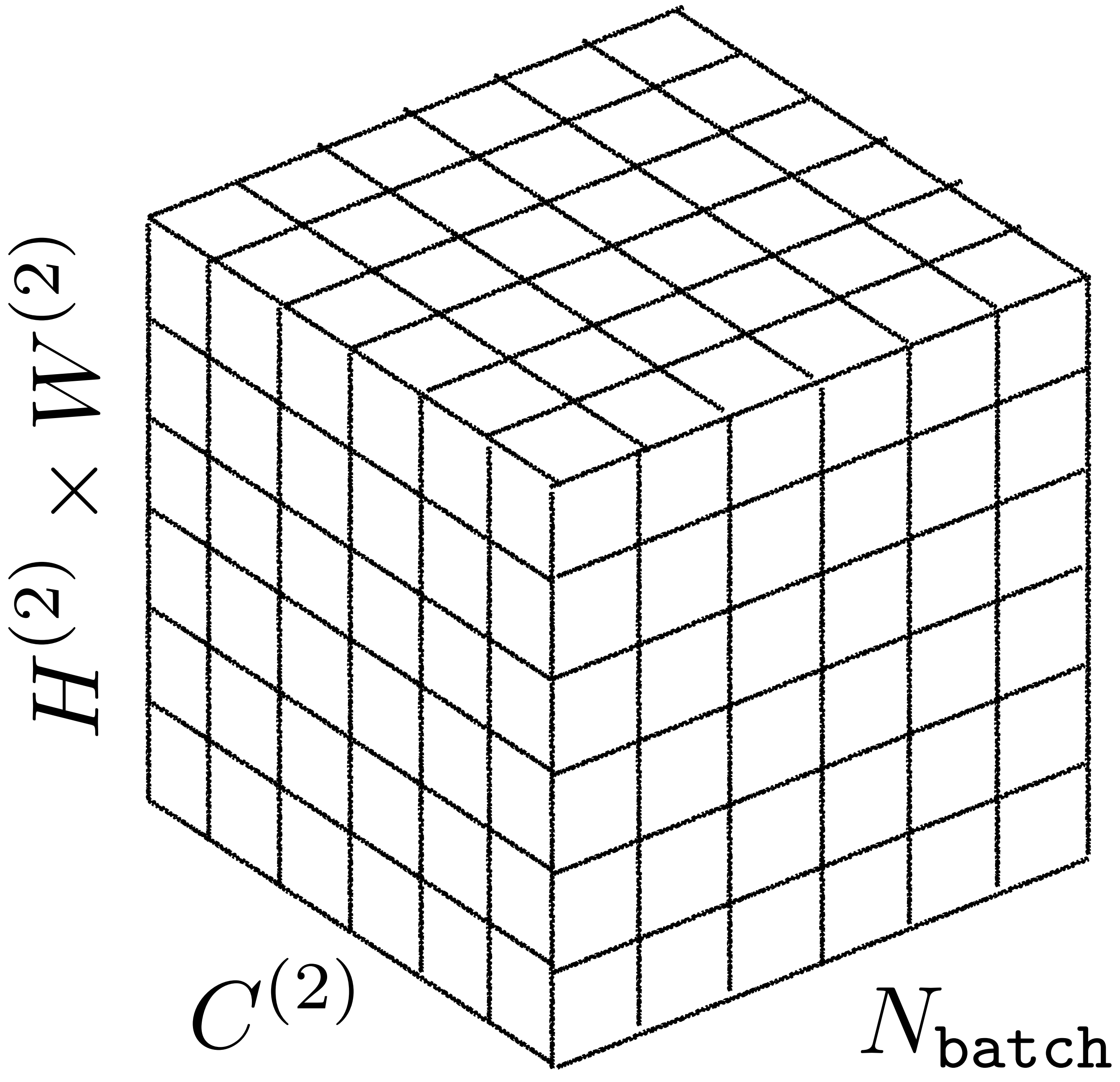
[DeCAF, Donahue, Jia, et al. 2013]

[Visualization technique : t-sne, van der Maaten & Hinton, 2008]

"Tensor flow"

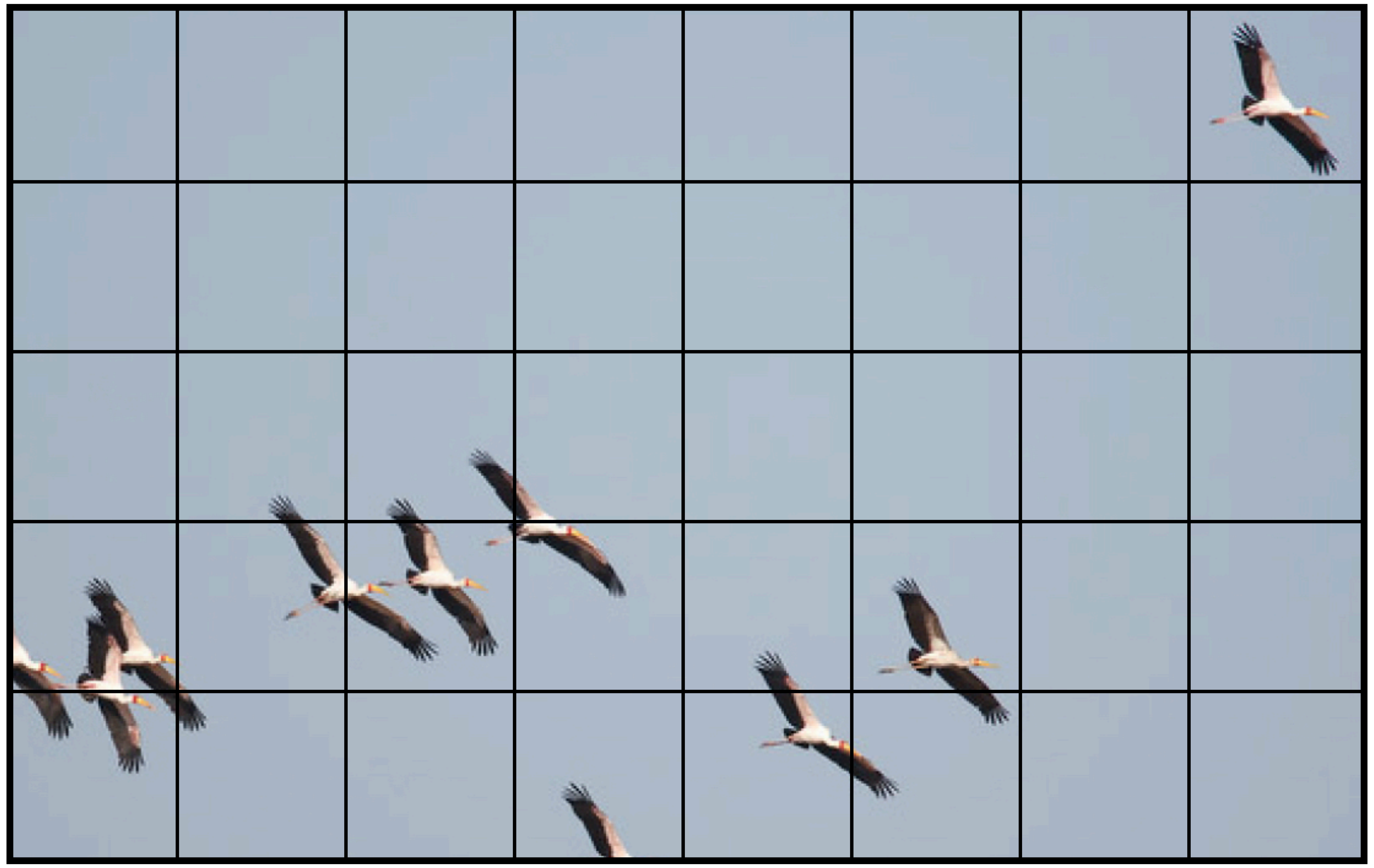
$\mathbf{h}^{(1)} \in \mathbb{R}^{N_{\texttt{batch}} \times H^{(1)} \times W^{(1)} \times C^{(1)}}$ $\mathbf{h}^{(2)} \in \mathbb{R}^{N_{\texttt{batch}} \times H^{(2)} \times W^{(2)} \times C^{(2)}}$

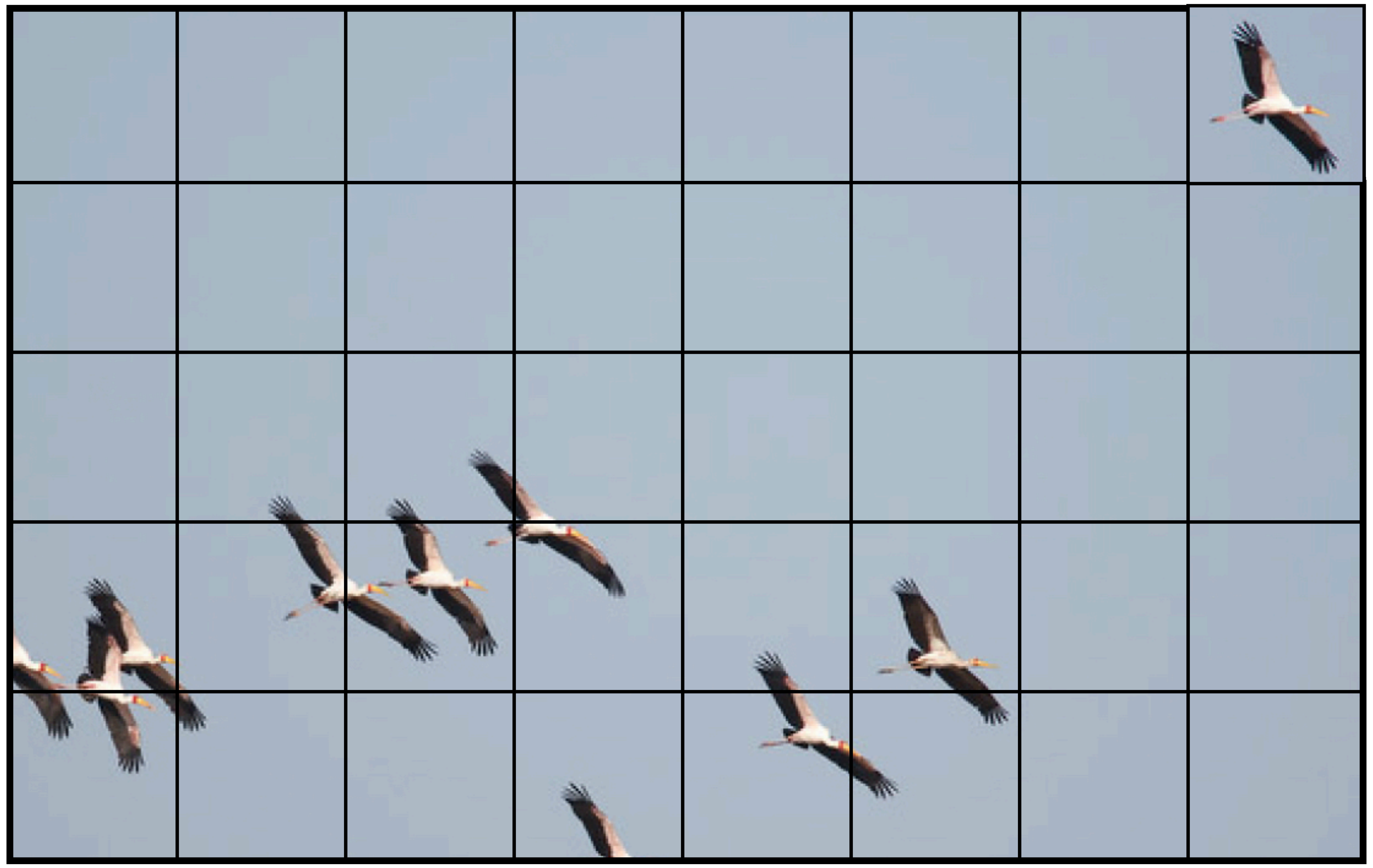




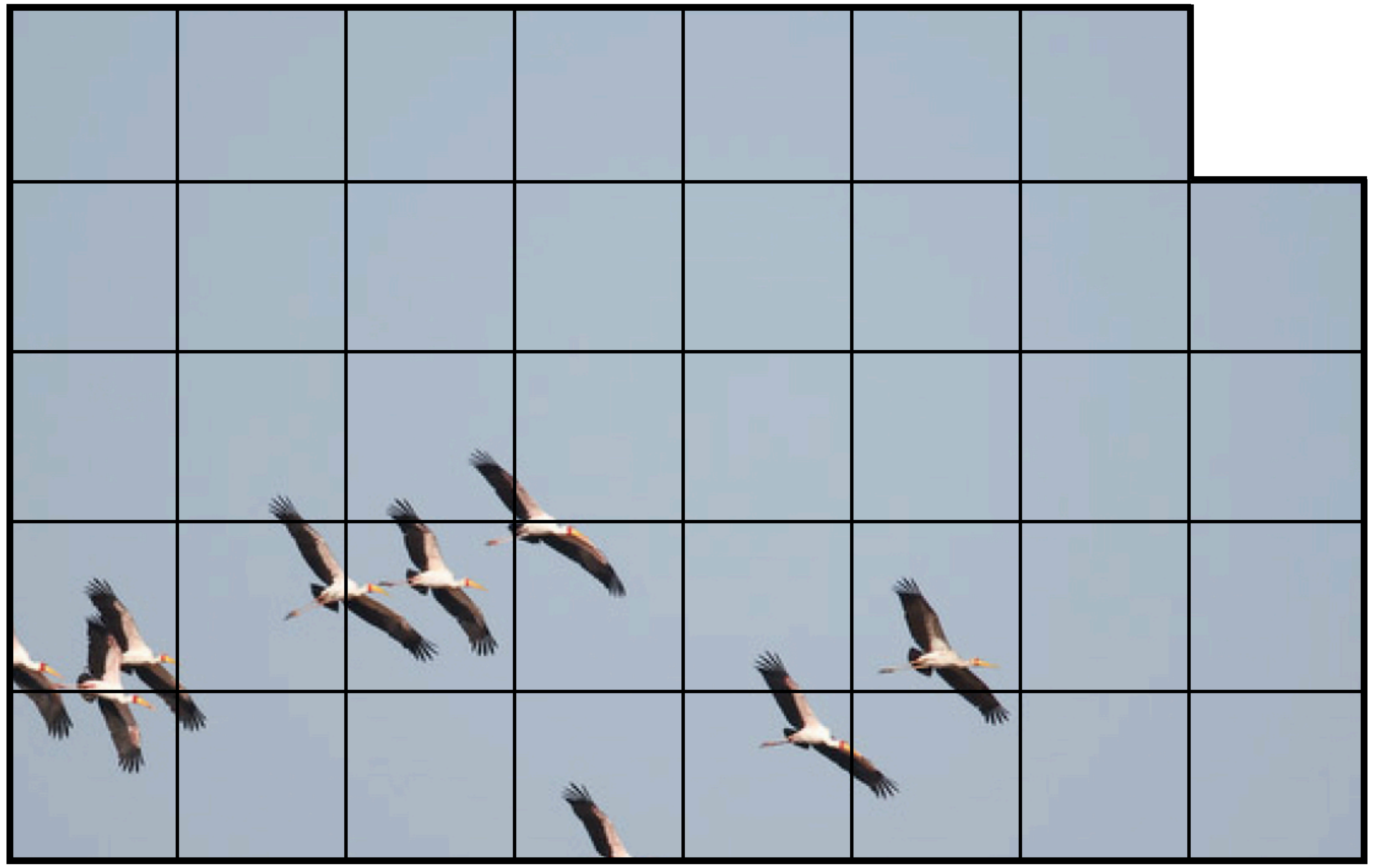
Convolutional Neural Networks

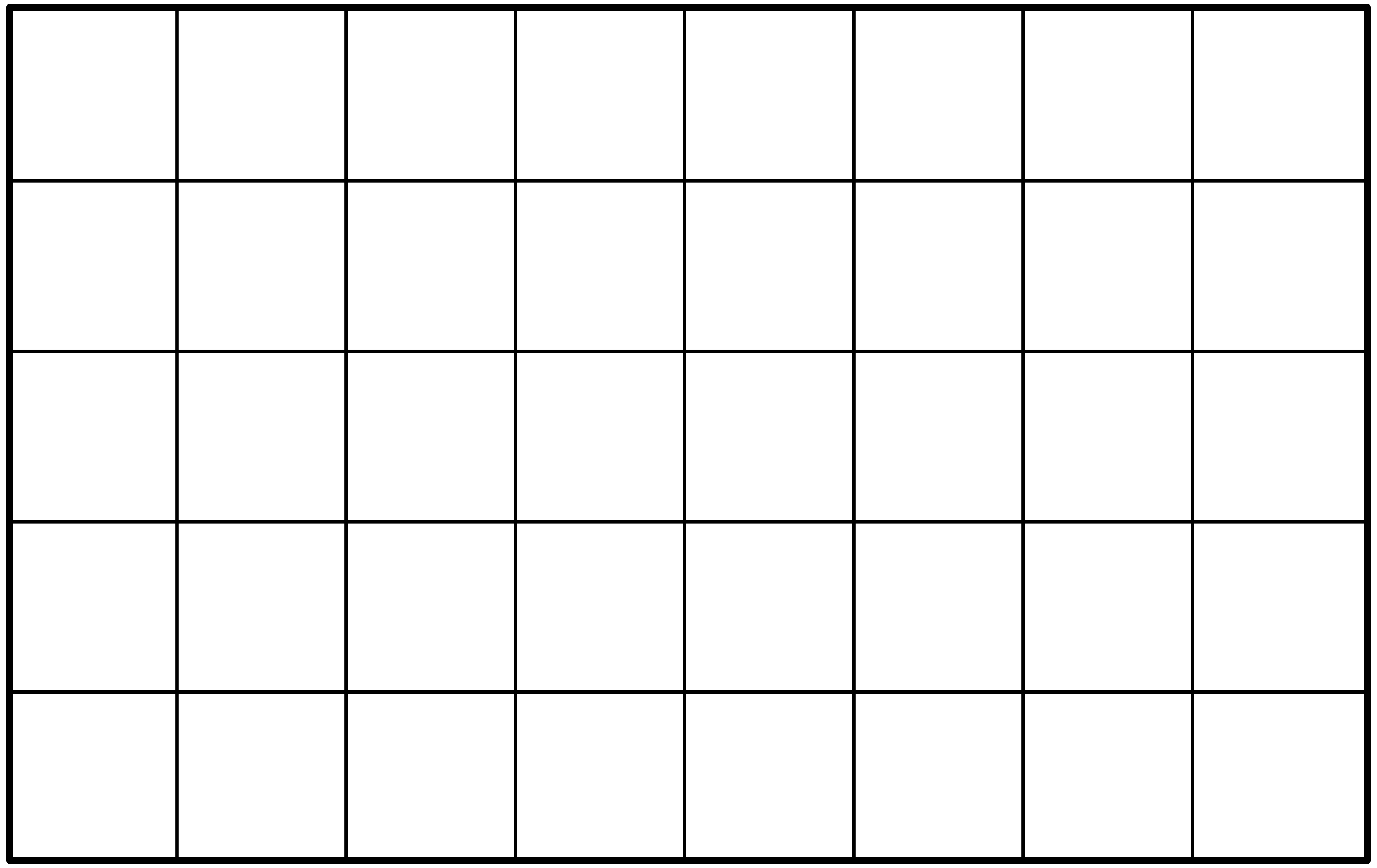
Photo credit: Fredo Durand



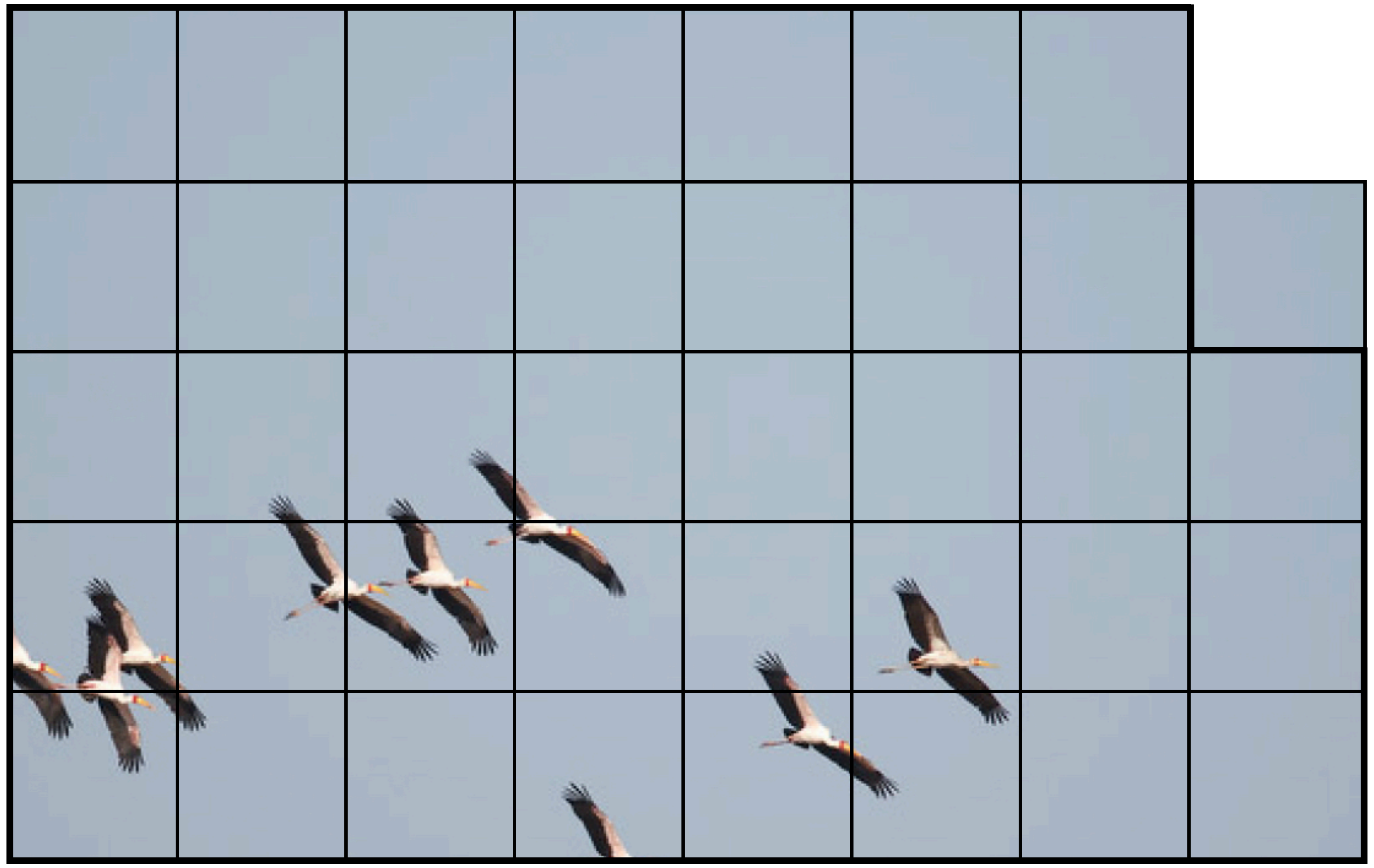


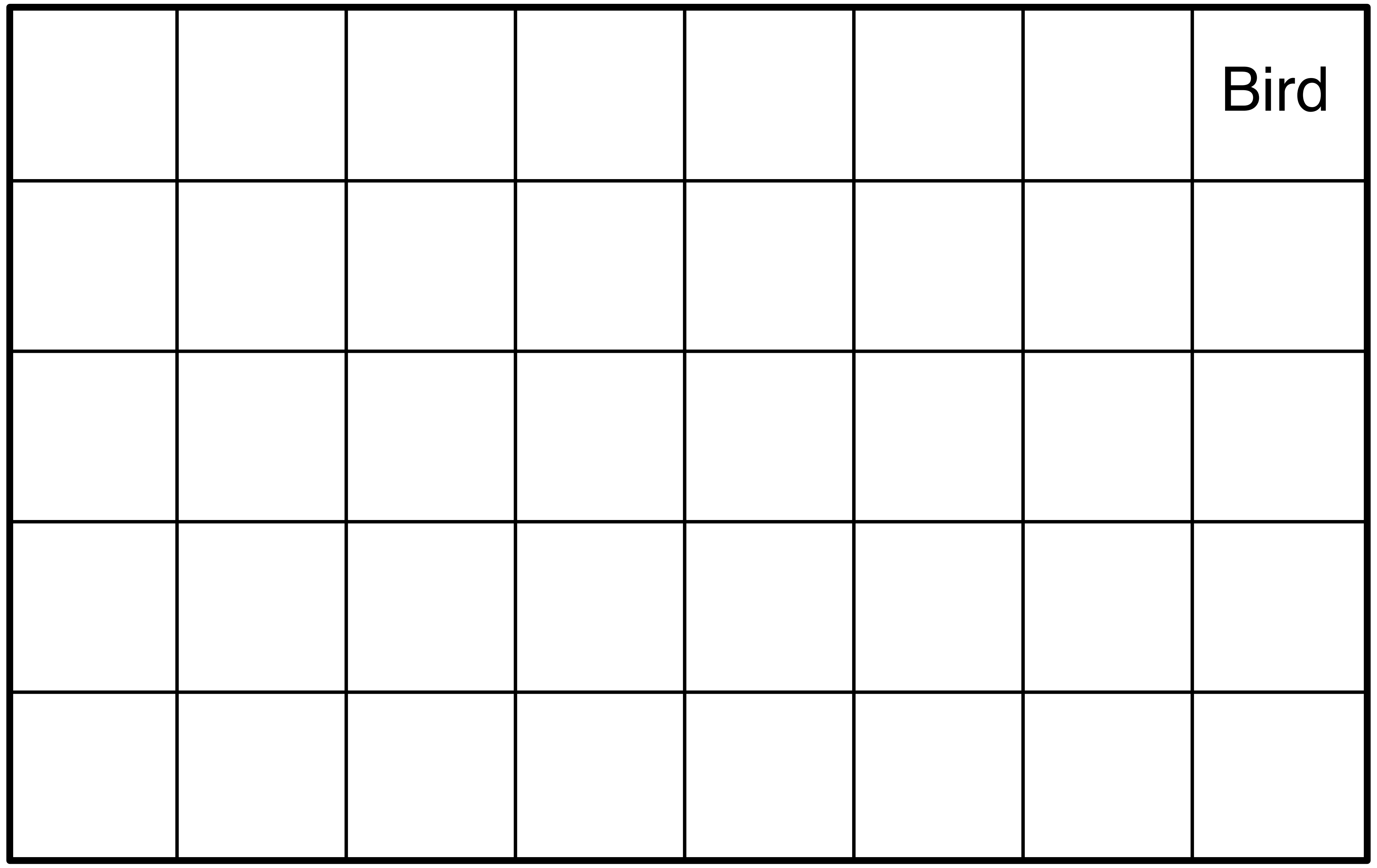

Classifier
"Bird"

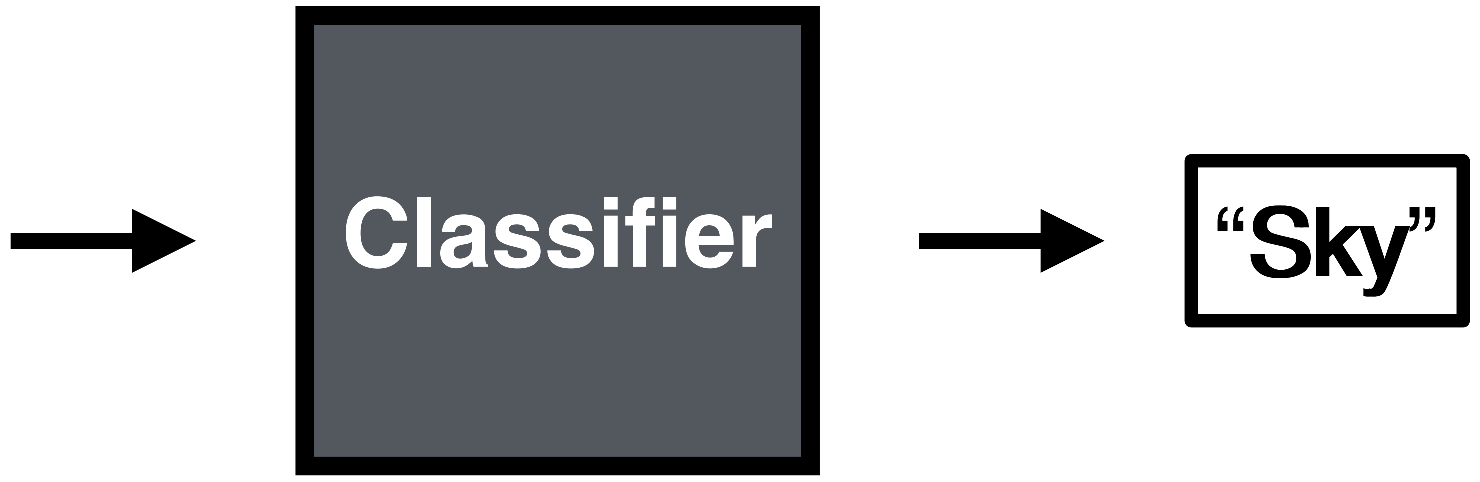


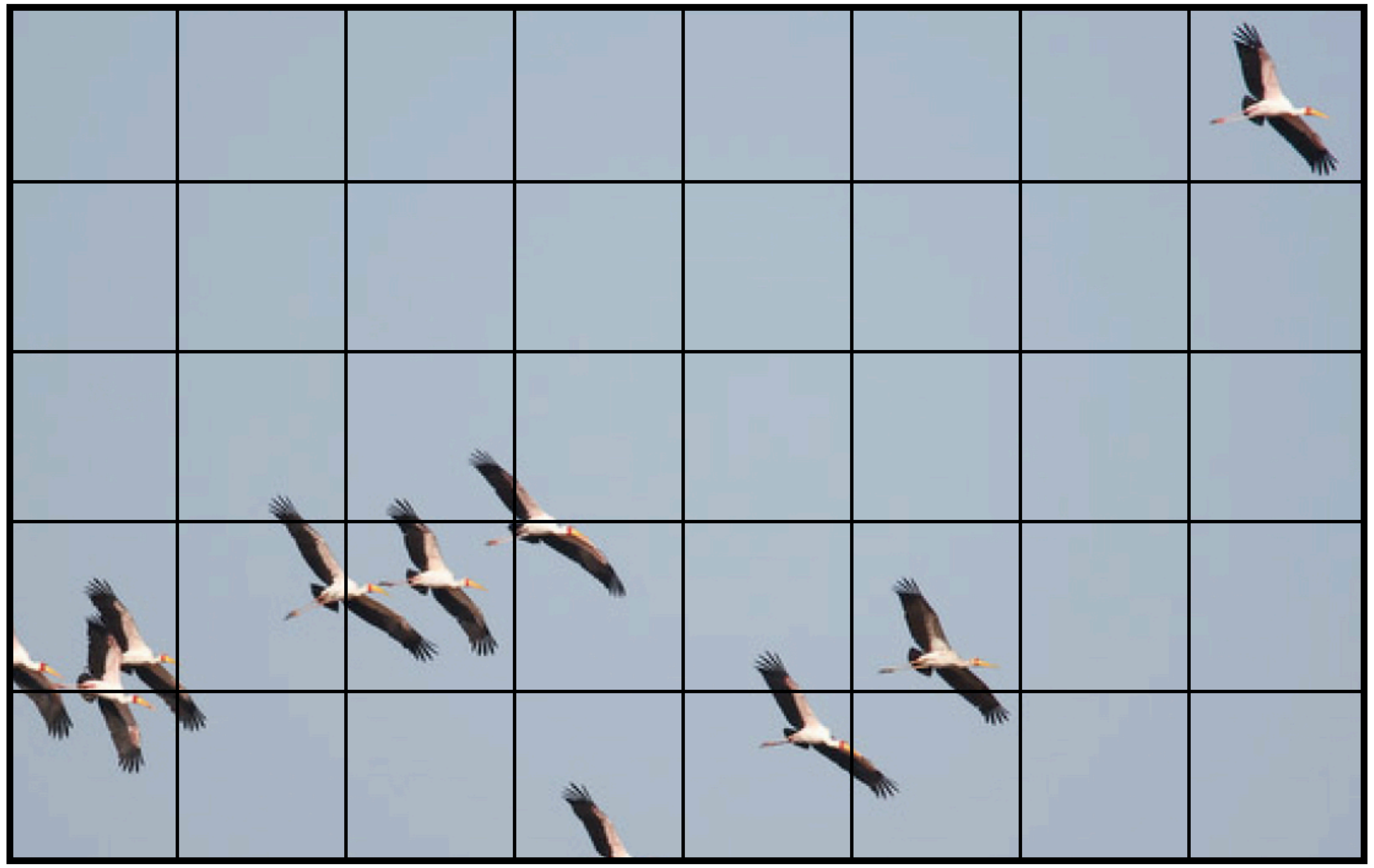



Classifier
“Bird”

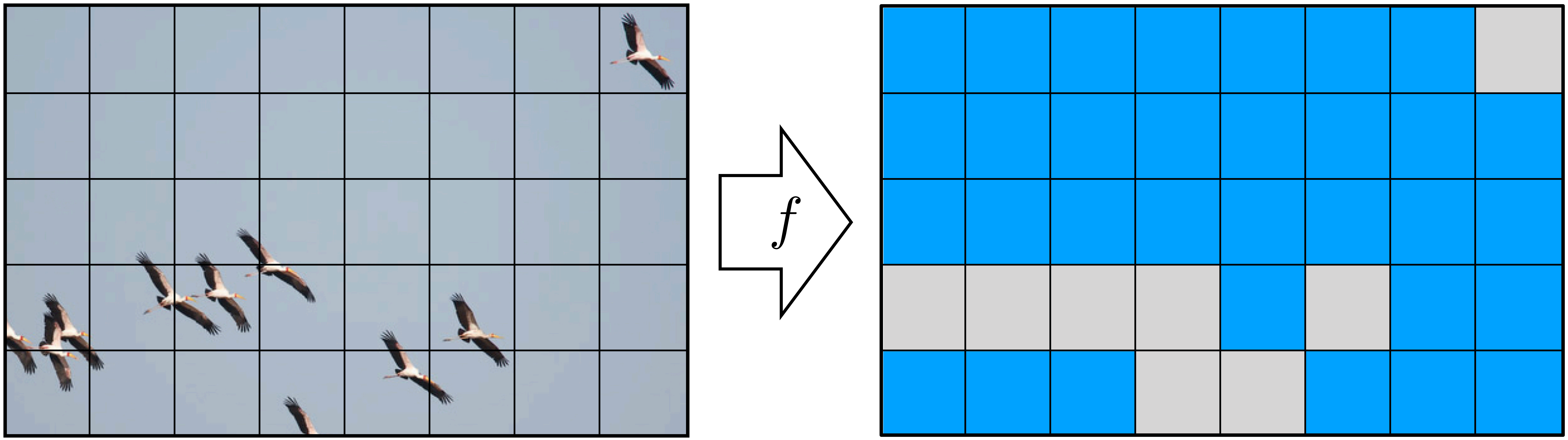



Bird


Classifier
"Sky"



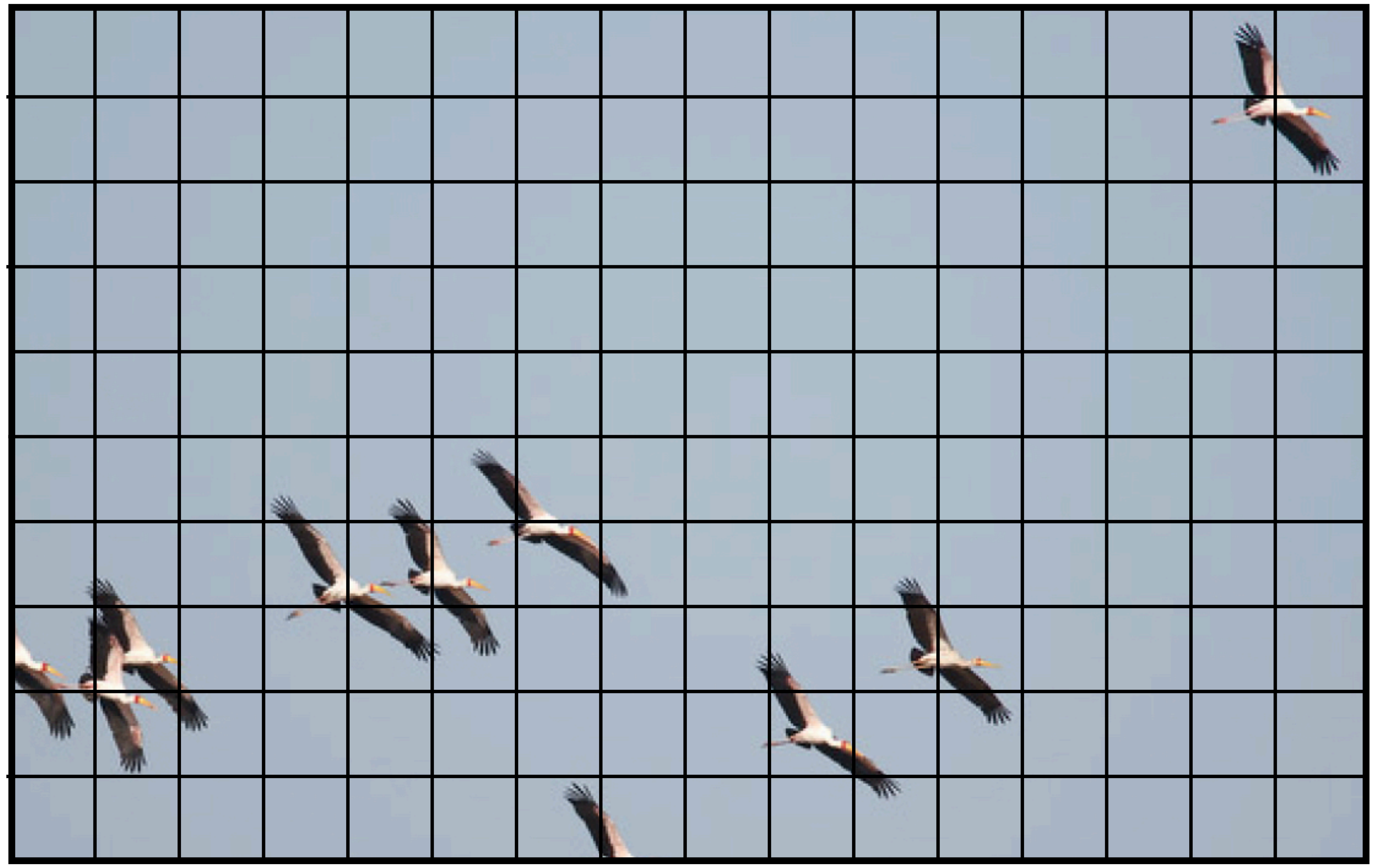
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Bird
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Bird	Bird	Bird	Sky	Bird	Sky	Sky	Sky
Sky	Sky	Sky	Bird	Sky	Sky	Sky	Sky



Problem:

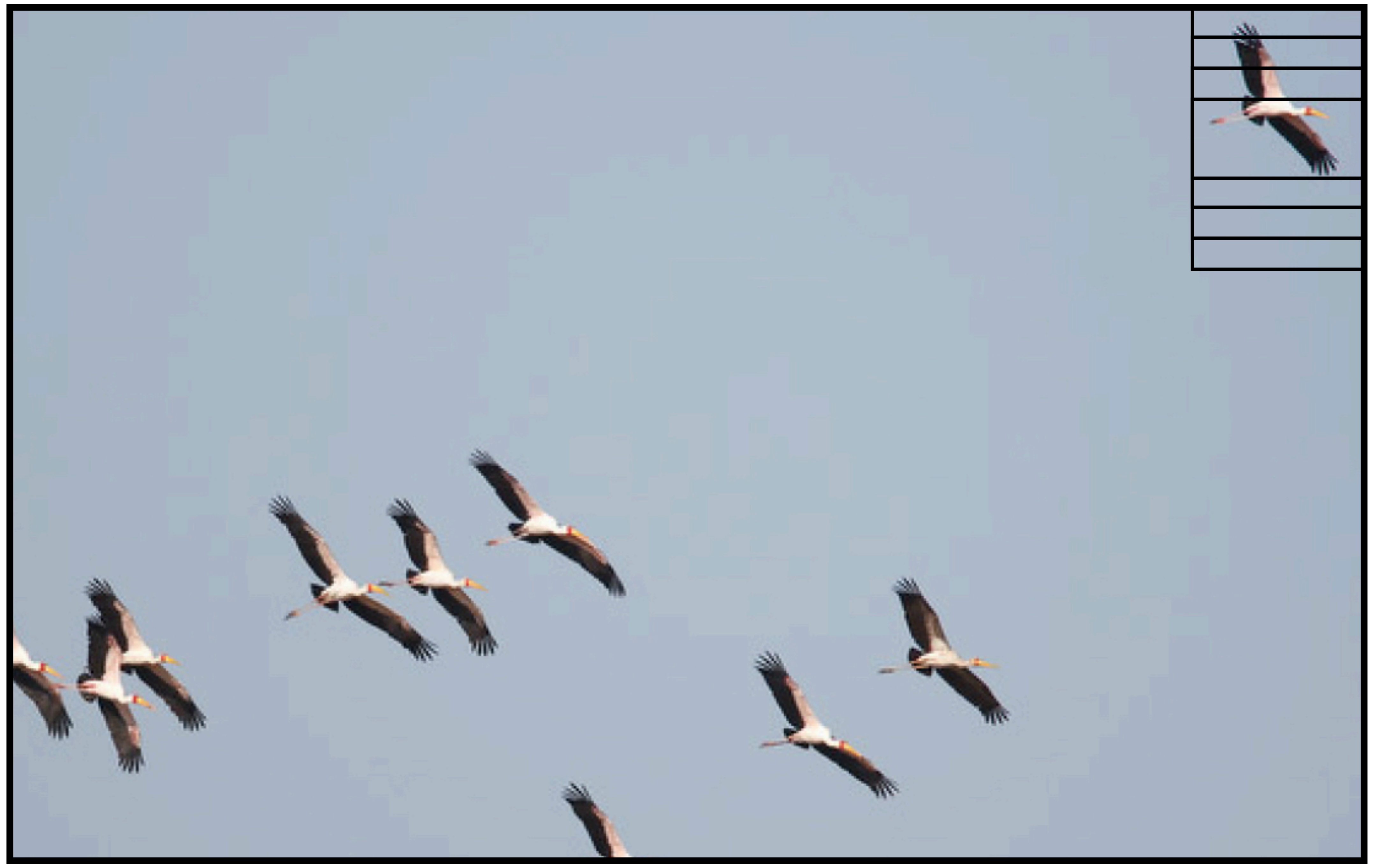
What happens to objects that are bigger?

What if an object crosses multiple cells?

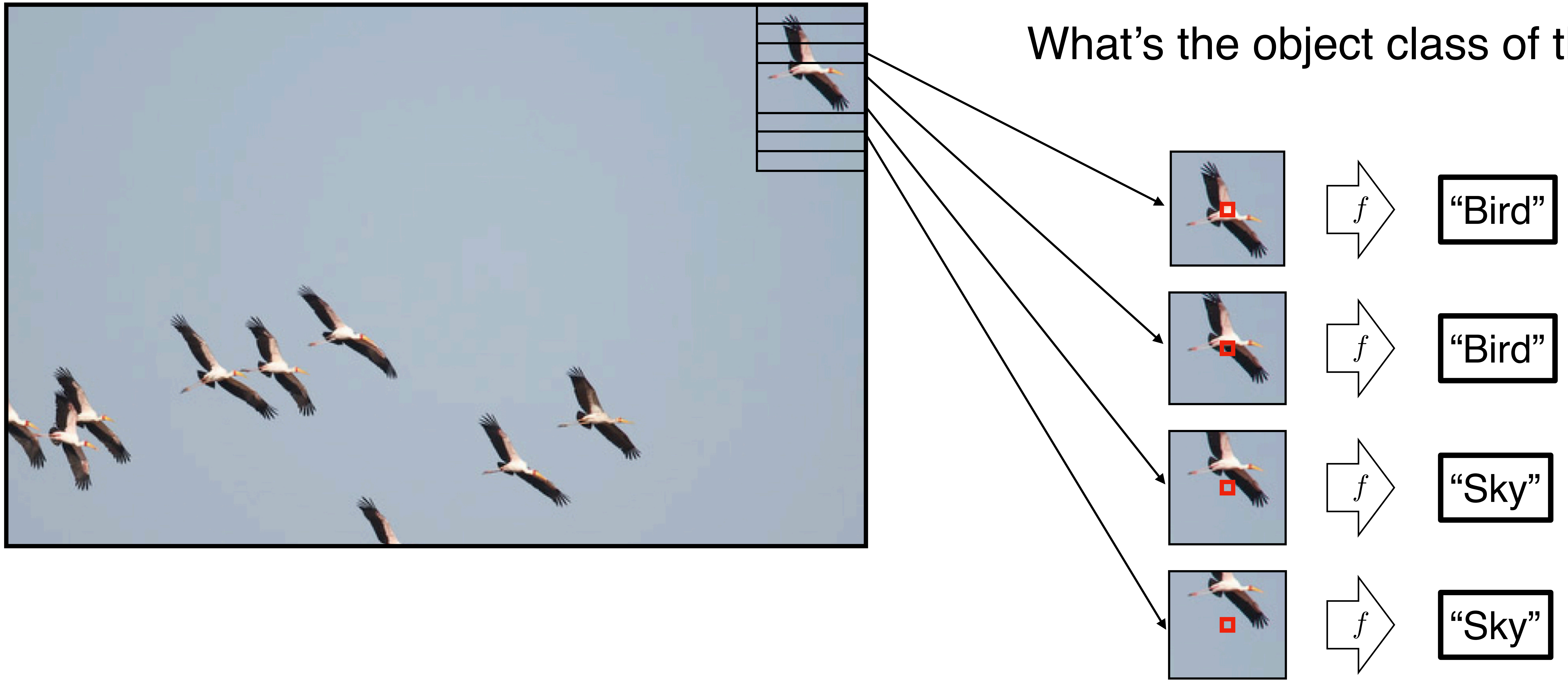


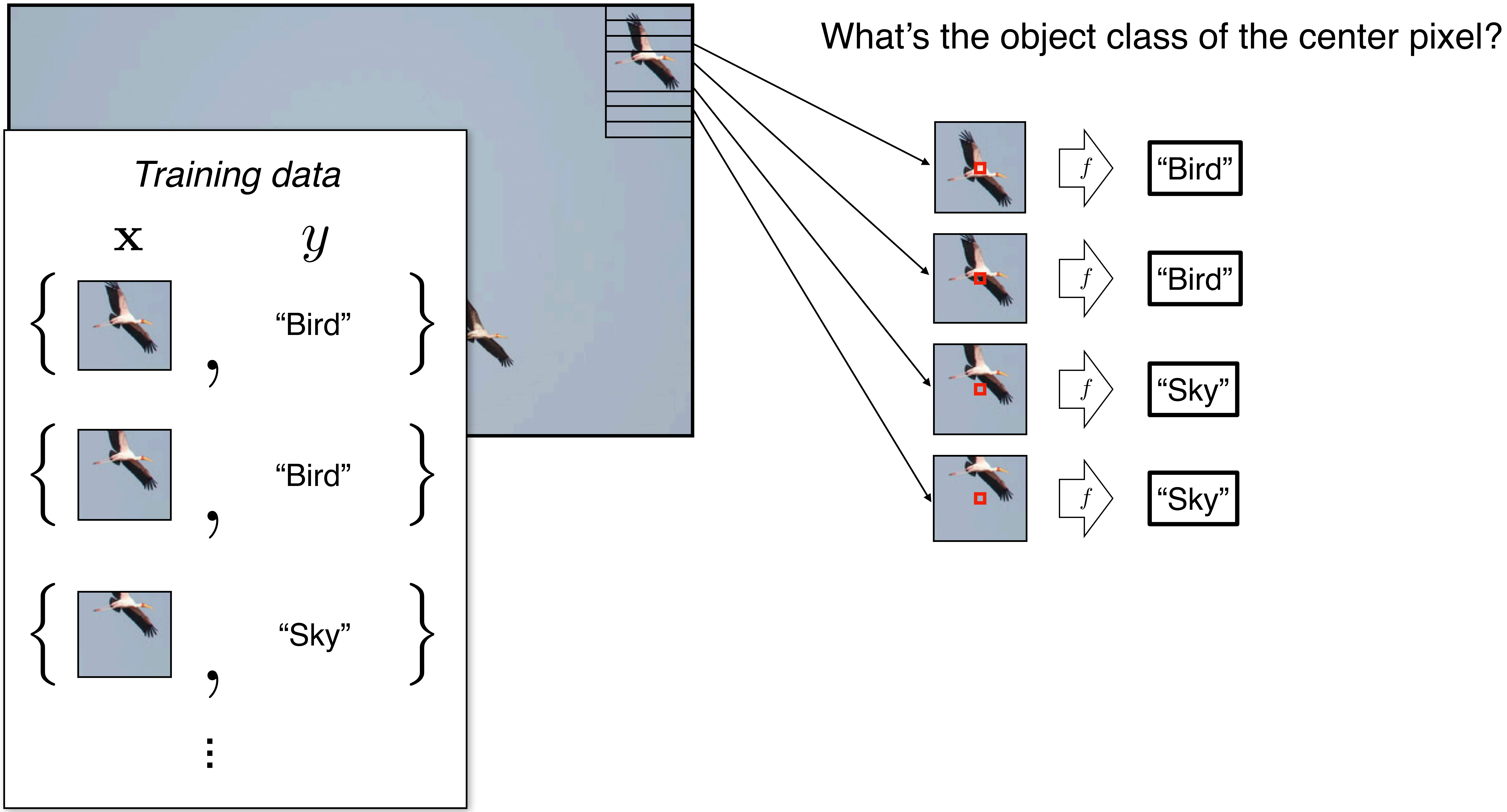
“Cell”-based approach is limited.

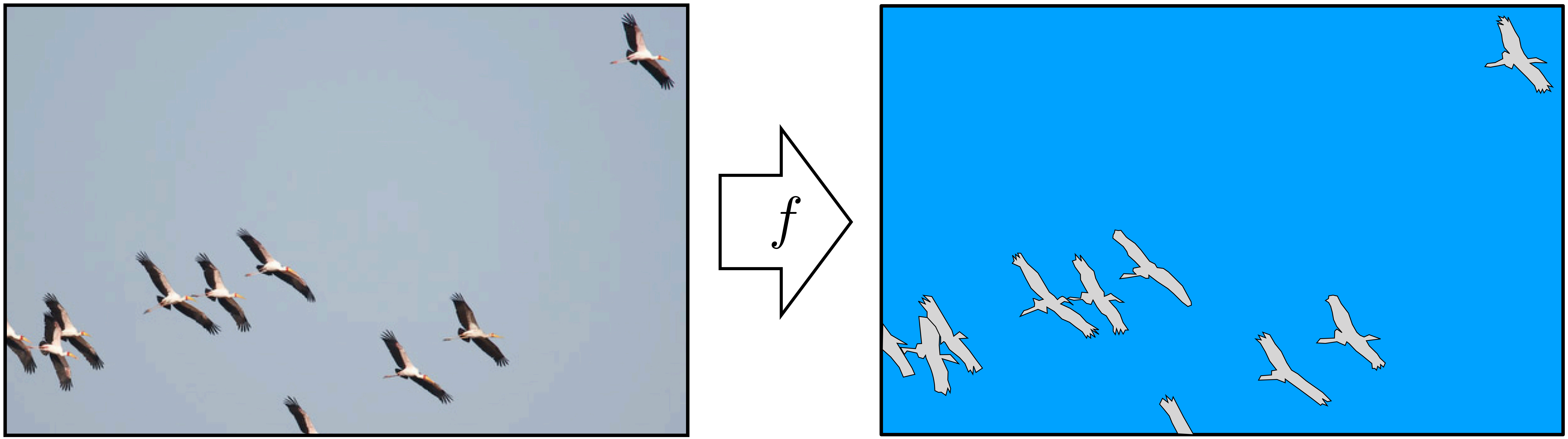
What can we do instead?



What’s the object class of the center pixel?



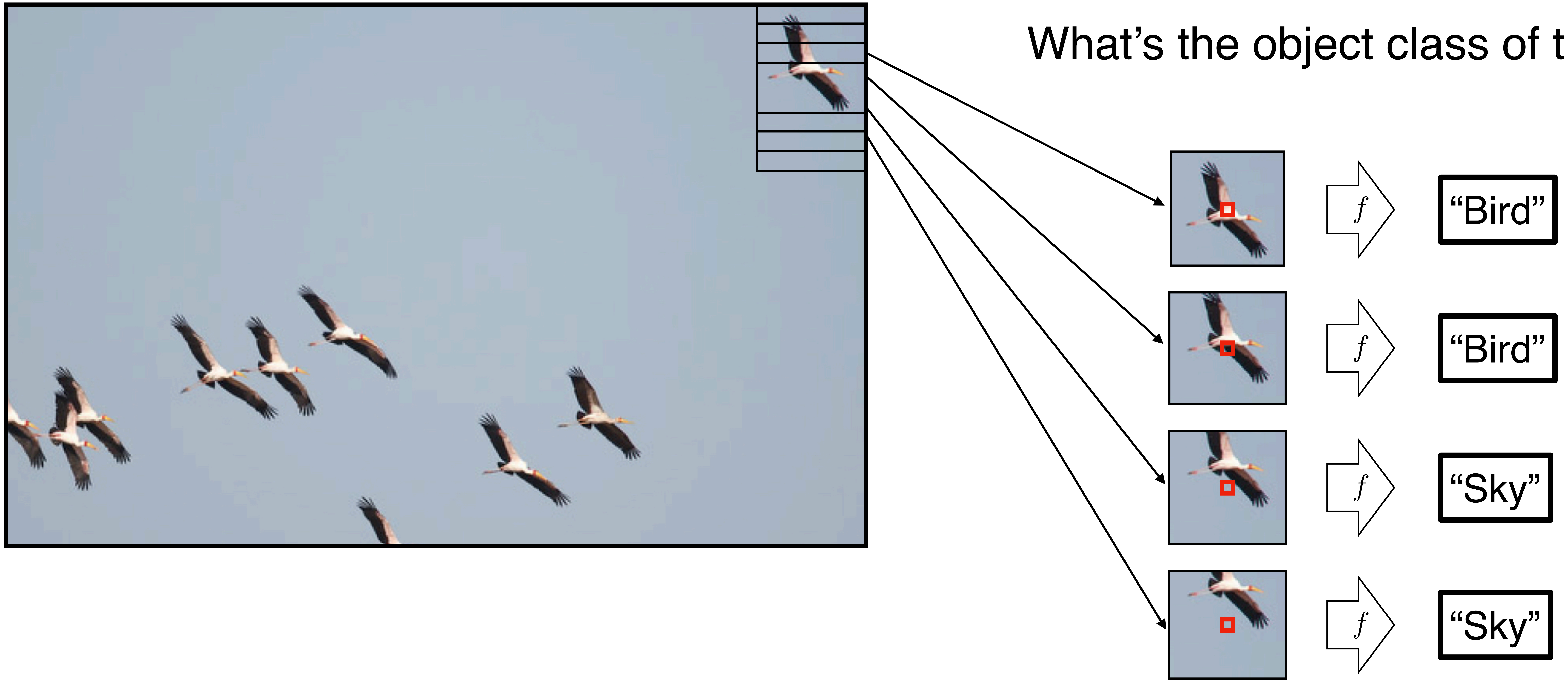

What's the object class of the center pixel?
Training data
x
y
"Bird"
"Bird"
"Sky"
f
"Bird"
f
"Bird"
f
"Sky"
f
"Sky"



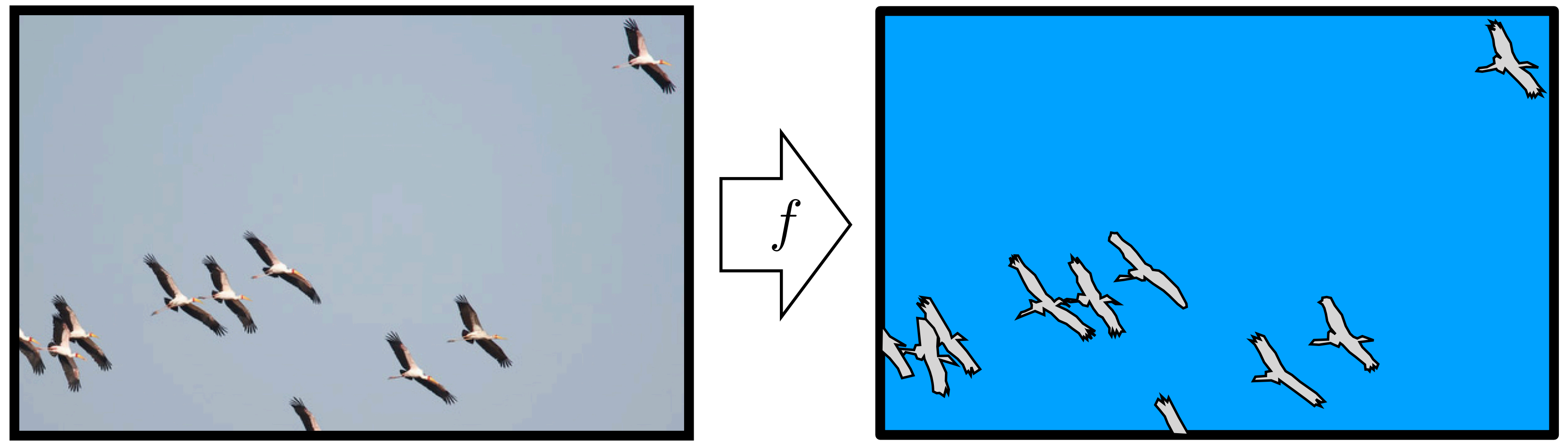
(Colors represent one-hot codes)

This problem is called **semantic segmentation**

What’s the object class of the center pixel?



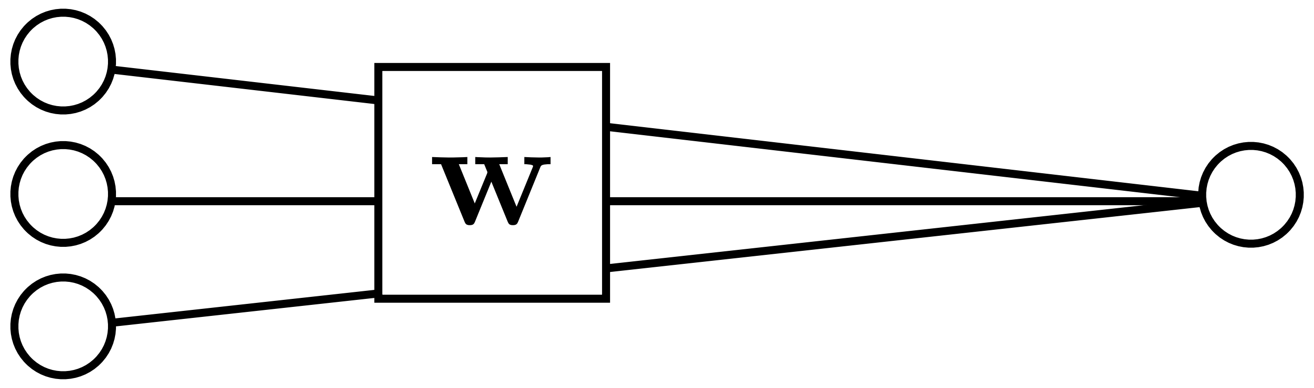
Translation invariance: process each patch in the same way.

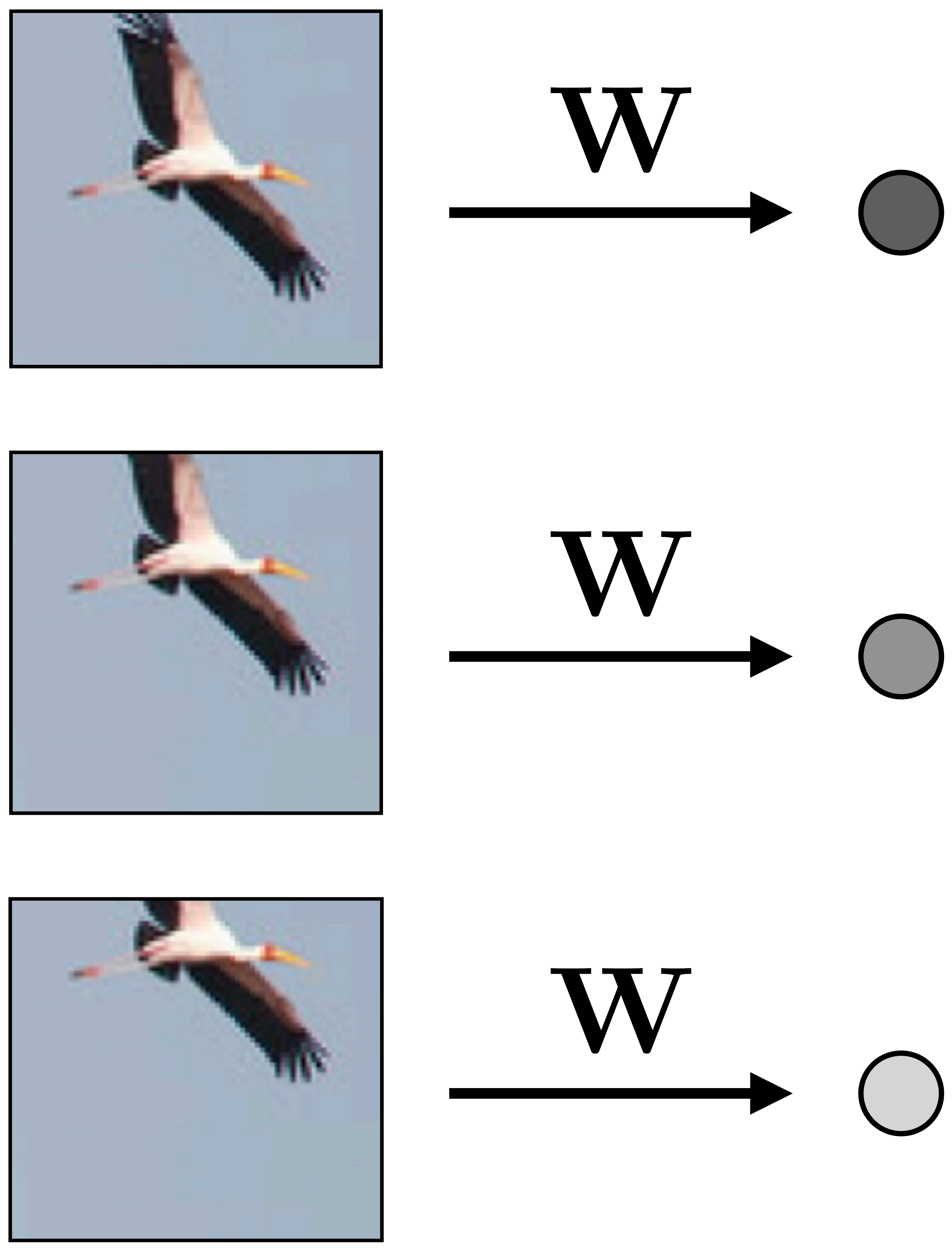


An *equivariant* mapping:

$$f(\texttt{translate}(x)) = \texttt{translate}(f(x))$$

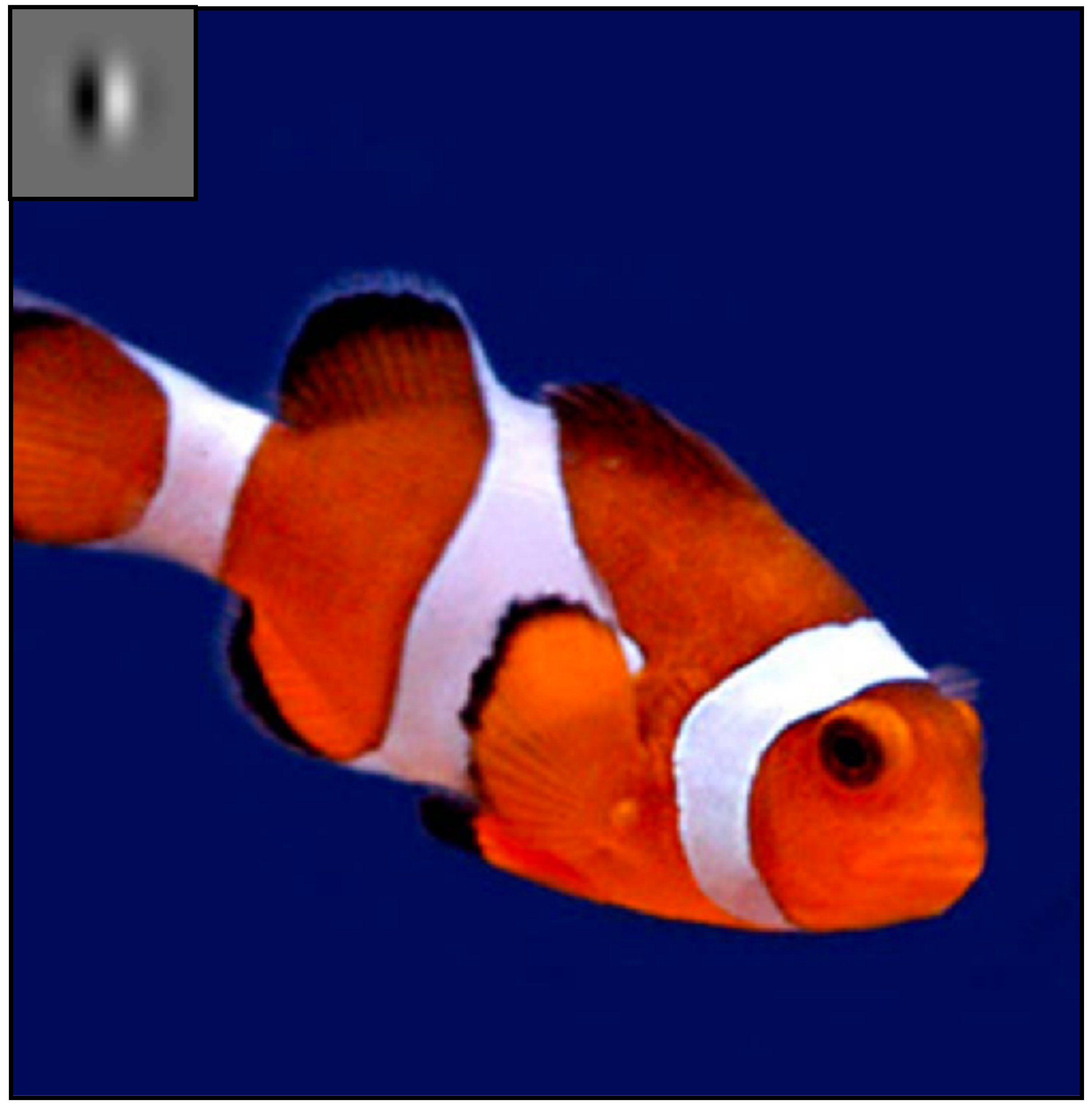
W computes a weighted sum of all pixels in the patch

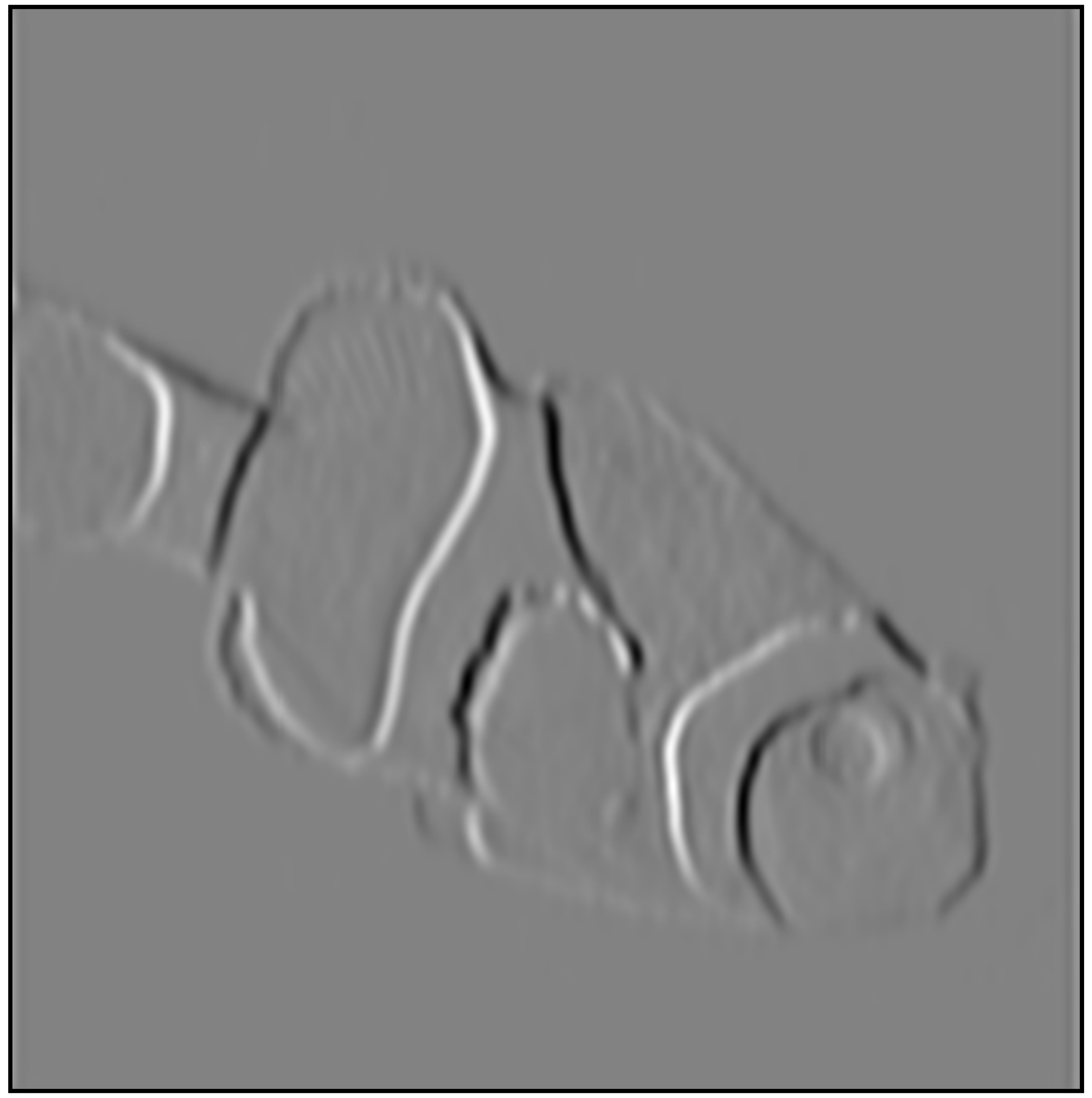




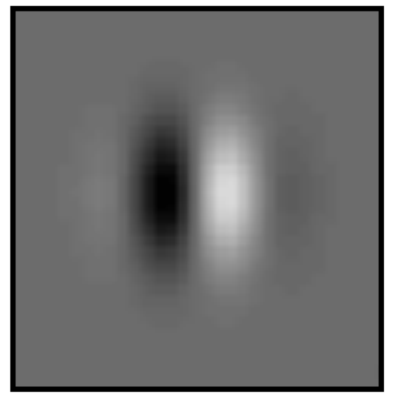
W is a convolutional kernel applied to the full image!

Convolution



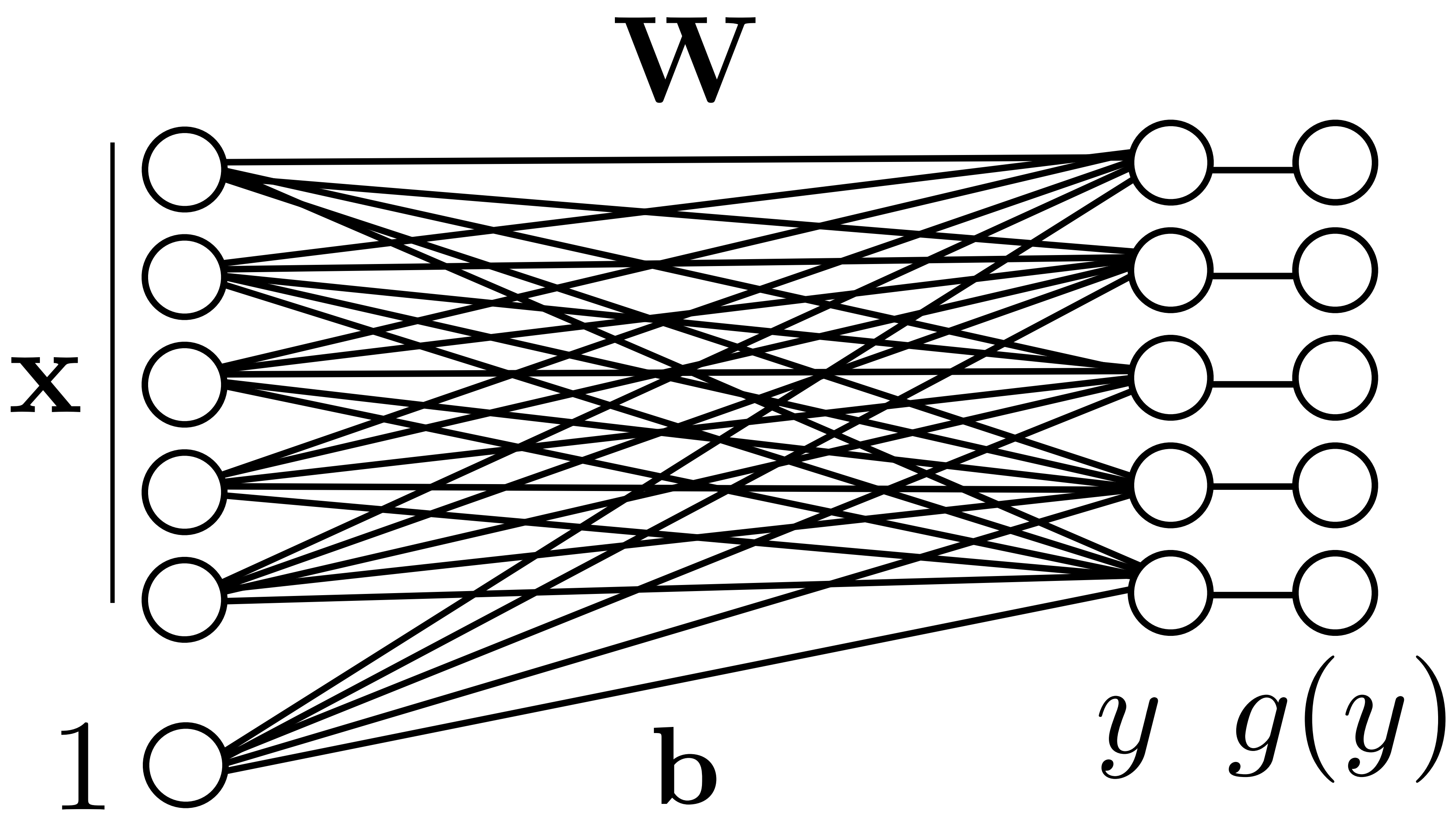


filter

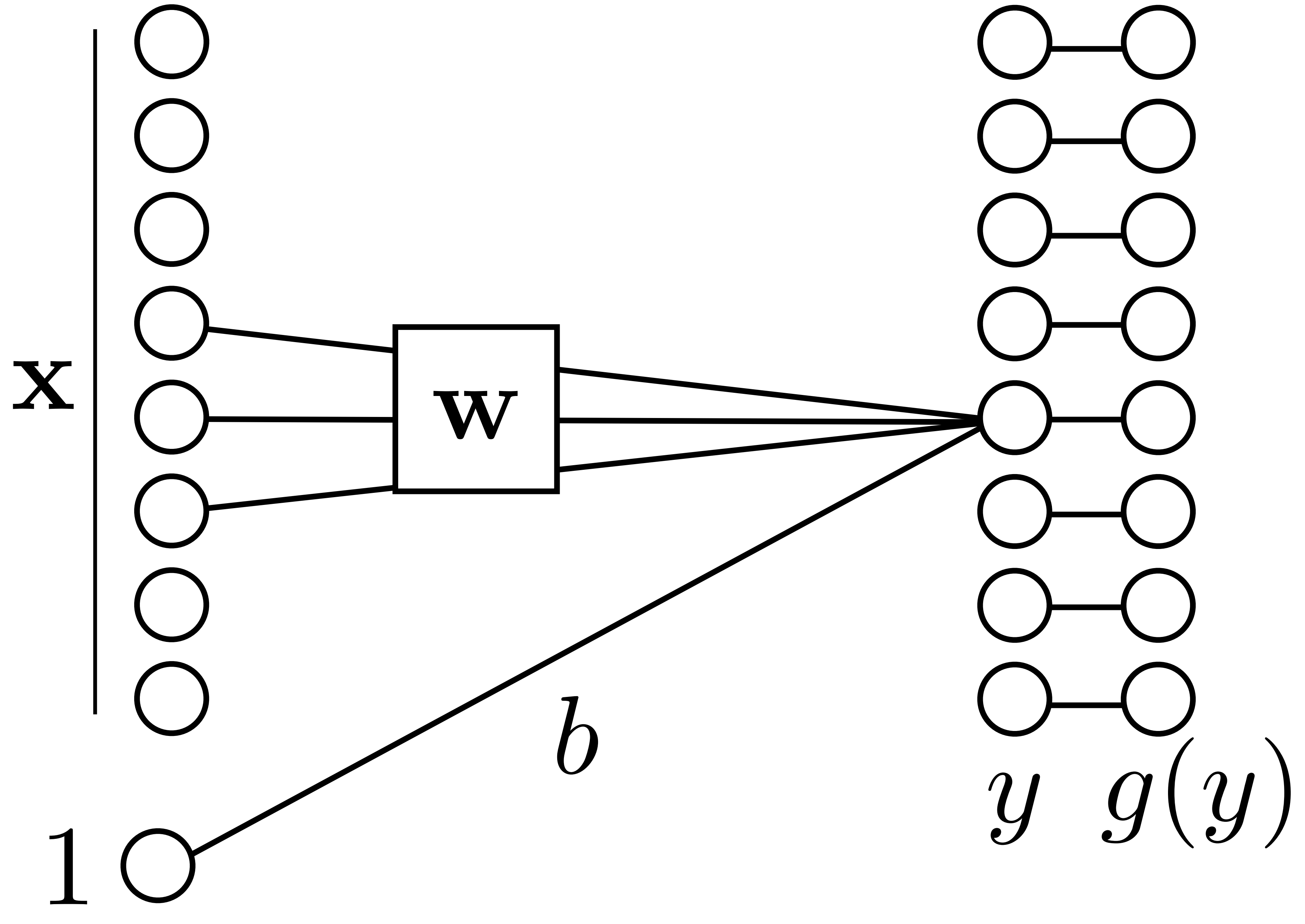


Fully-connected network

Fully-connected (fc) layer



Locally connected network

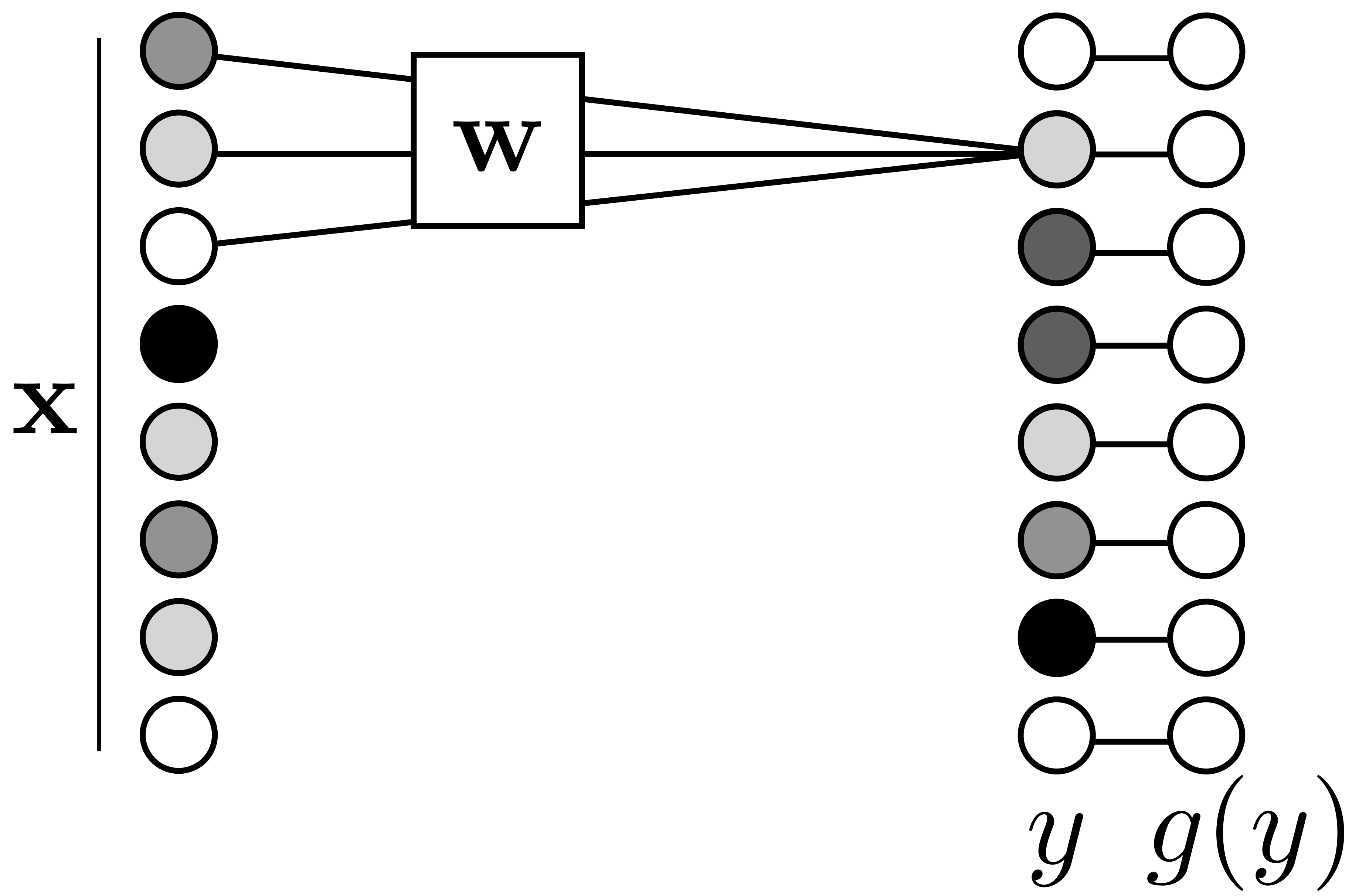


Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

Convolutional neural network

Conv layer

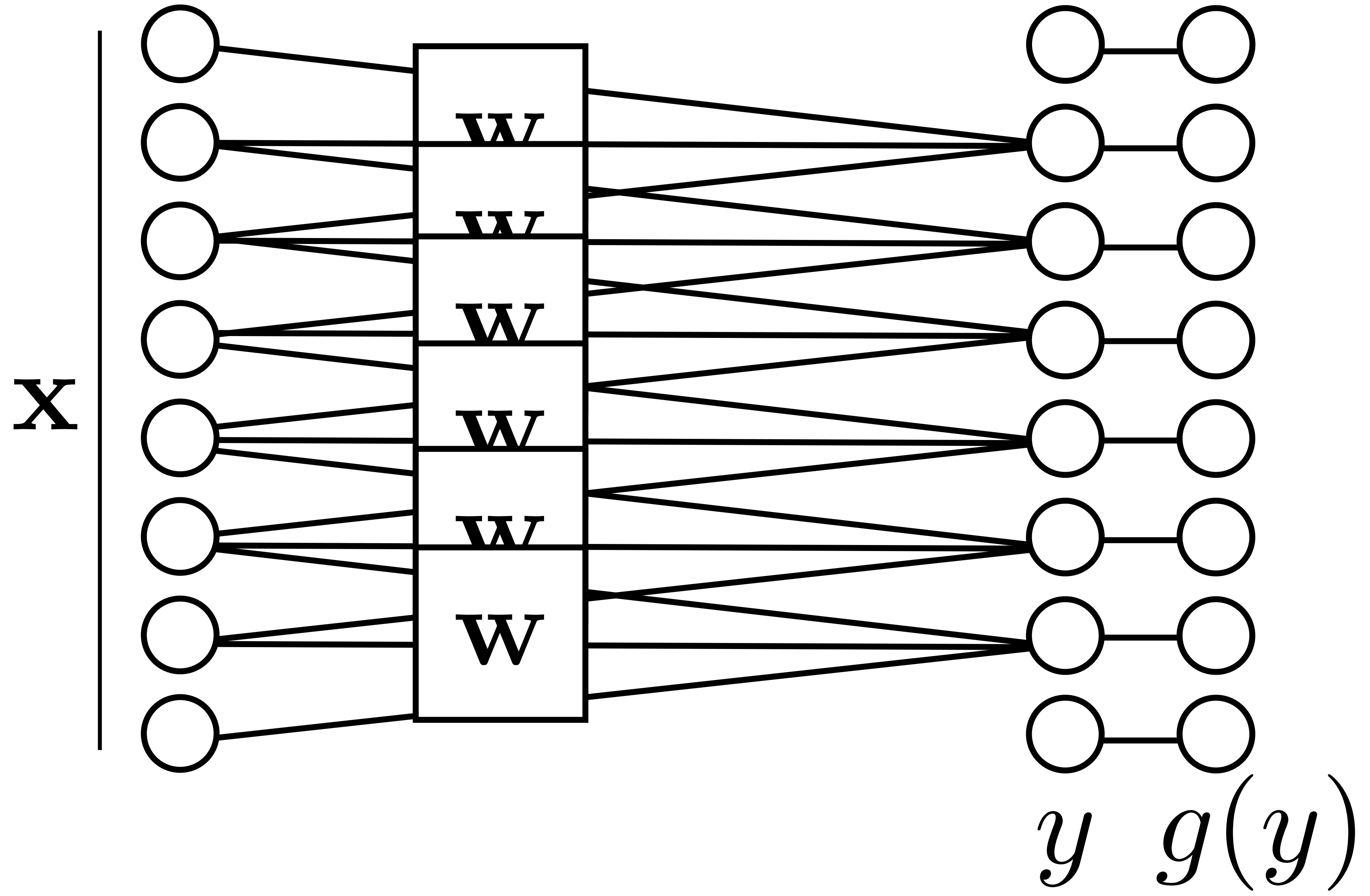


Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

Weight sharing

Conv layer



Often, we assume output is a **local** function of input.

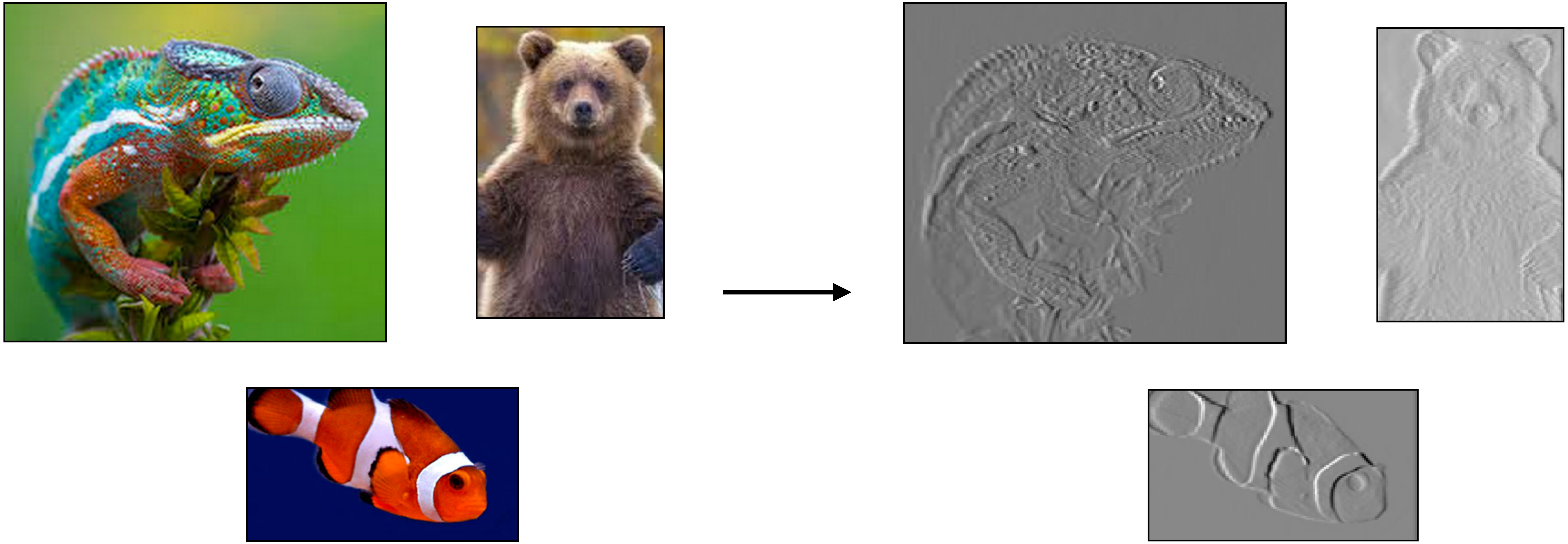
If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

Toeplitz matrix

$$\begin{pmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{pmatrix}$$

e.g., pixel image

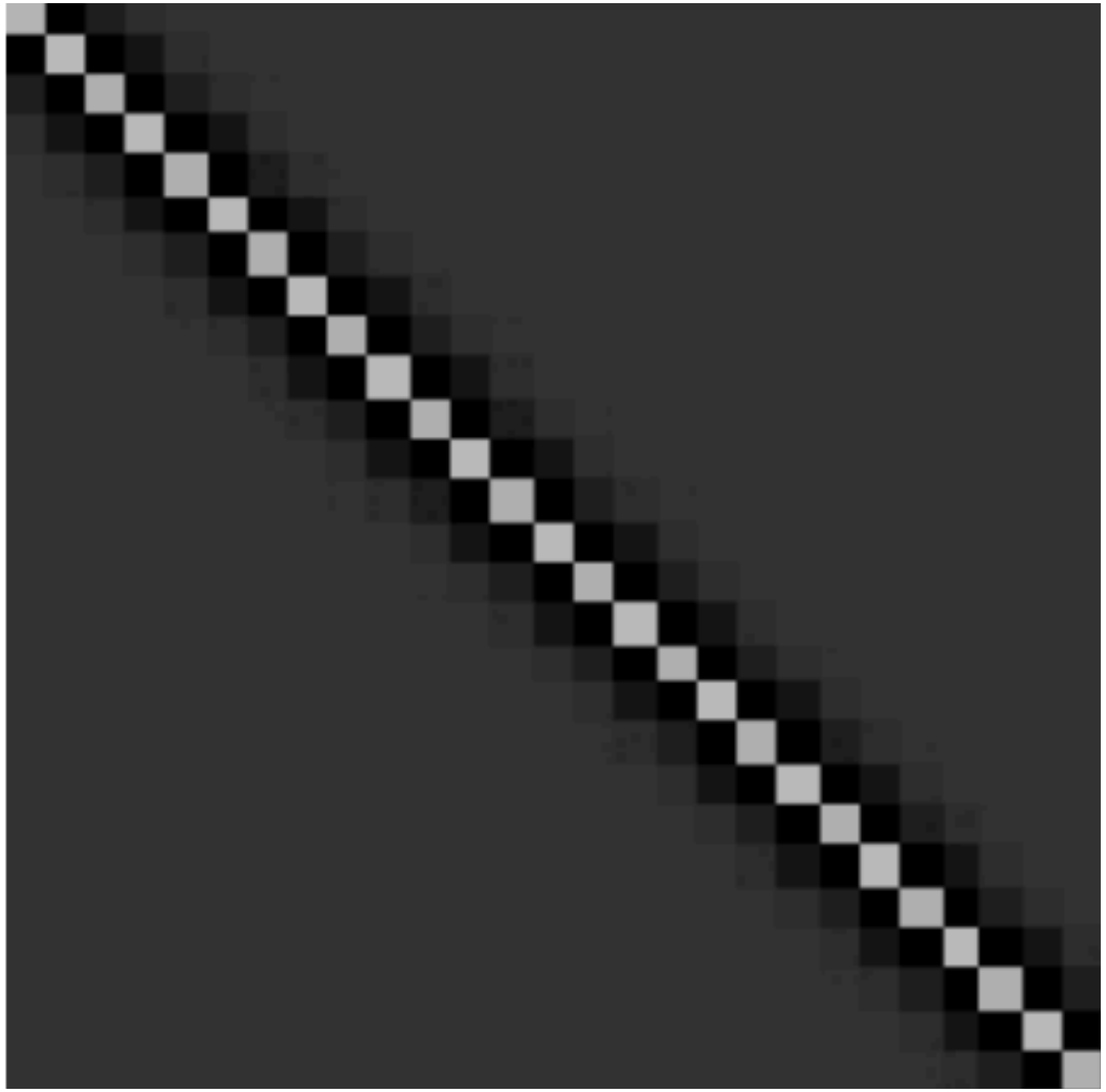
- Constrained linear layer
- Fewer parameters —> easier to learn, less overfitting



Conv layers can be applied to arbitrarily-sized inputs


$\mathbf{x}^{(l+1)}$
$=$
$*$
$\mathbf{x}^{(l)}$

$\mathbf{x}^{(l+1)} =$

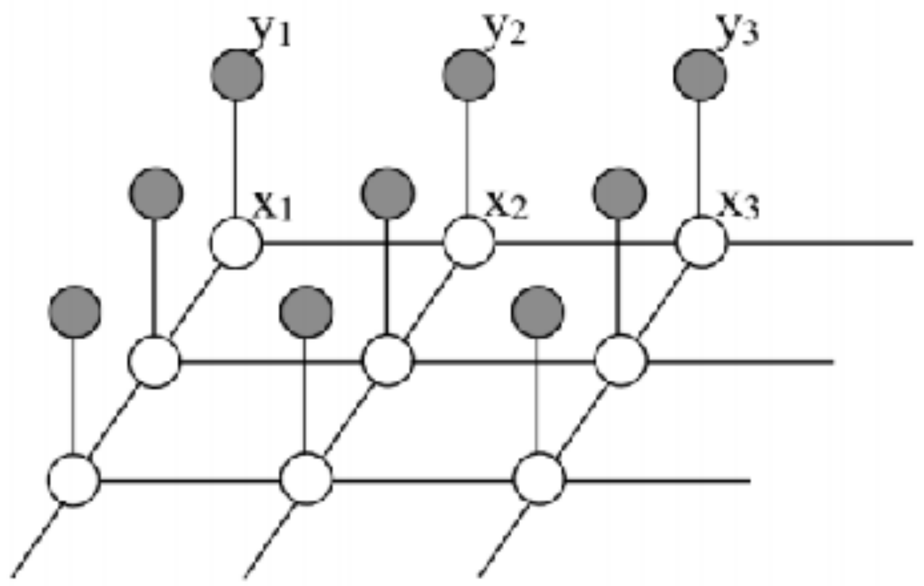


$* \; \mathbf{x}^{(l)}$

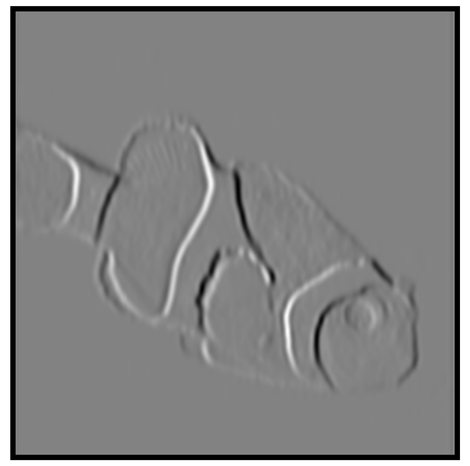
Five views on convolutional layers

1. Equivariant with translation (stationarity) $f(\texttt{translate}(x)) = \texttt{translate}(f(x))$

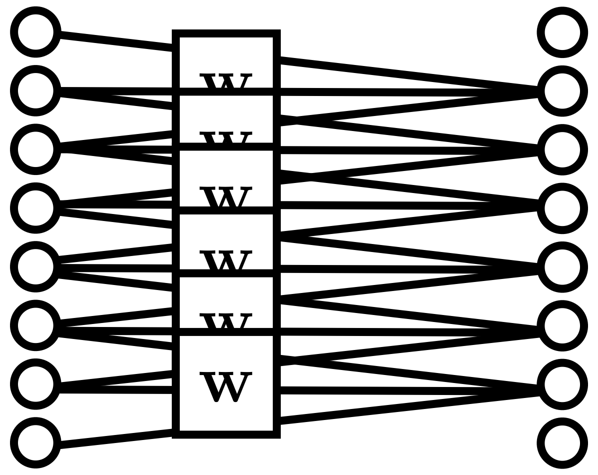
2. Patch processing (Markov assumption)



3. Image filter



4. Parameter sharing

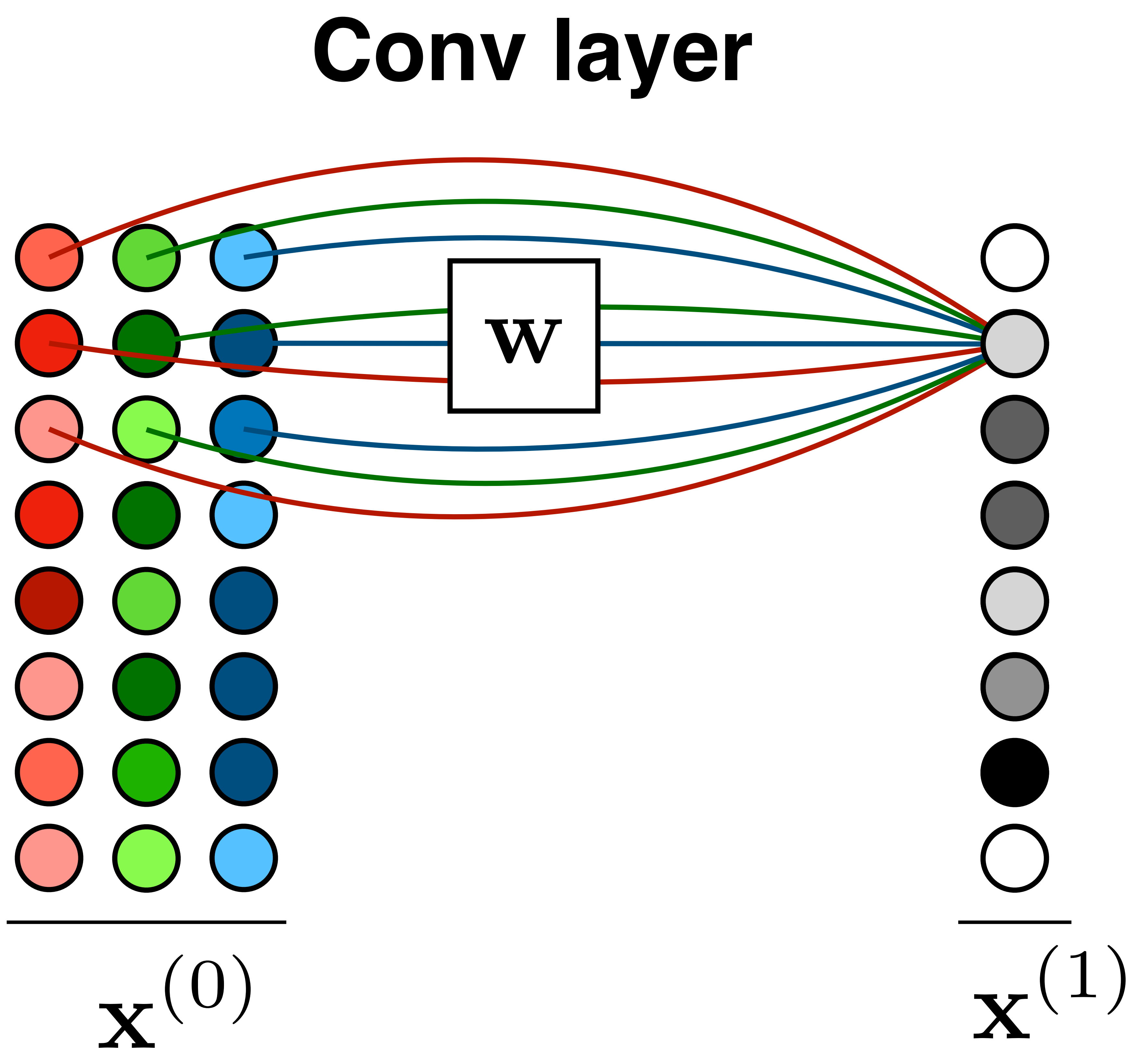


5. A way to process variable-sized tensors

What if we have color?

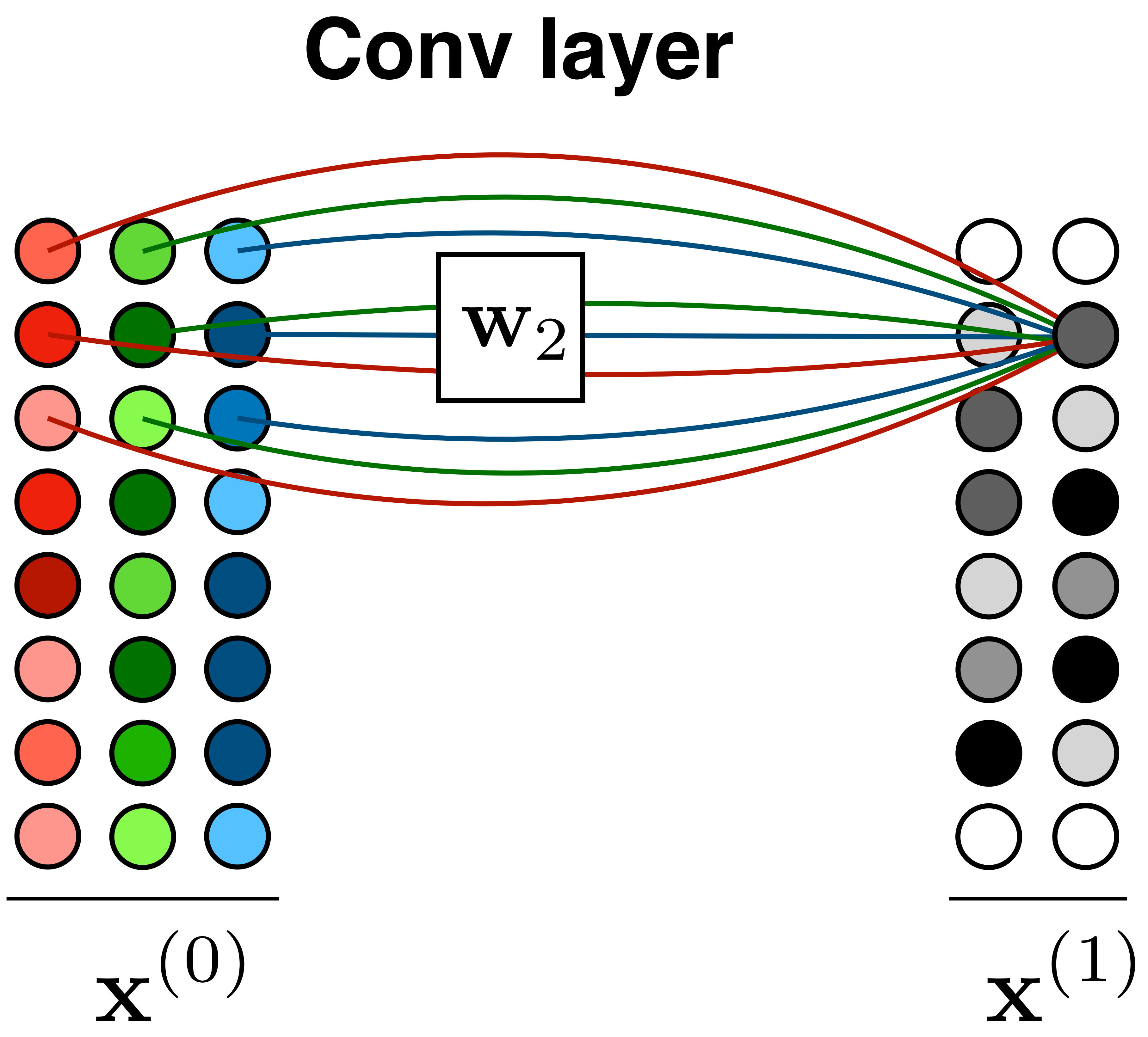
(aka multiple input channels?)

Multiple channels



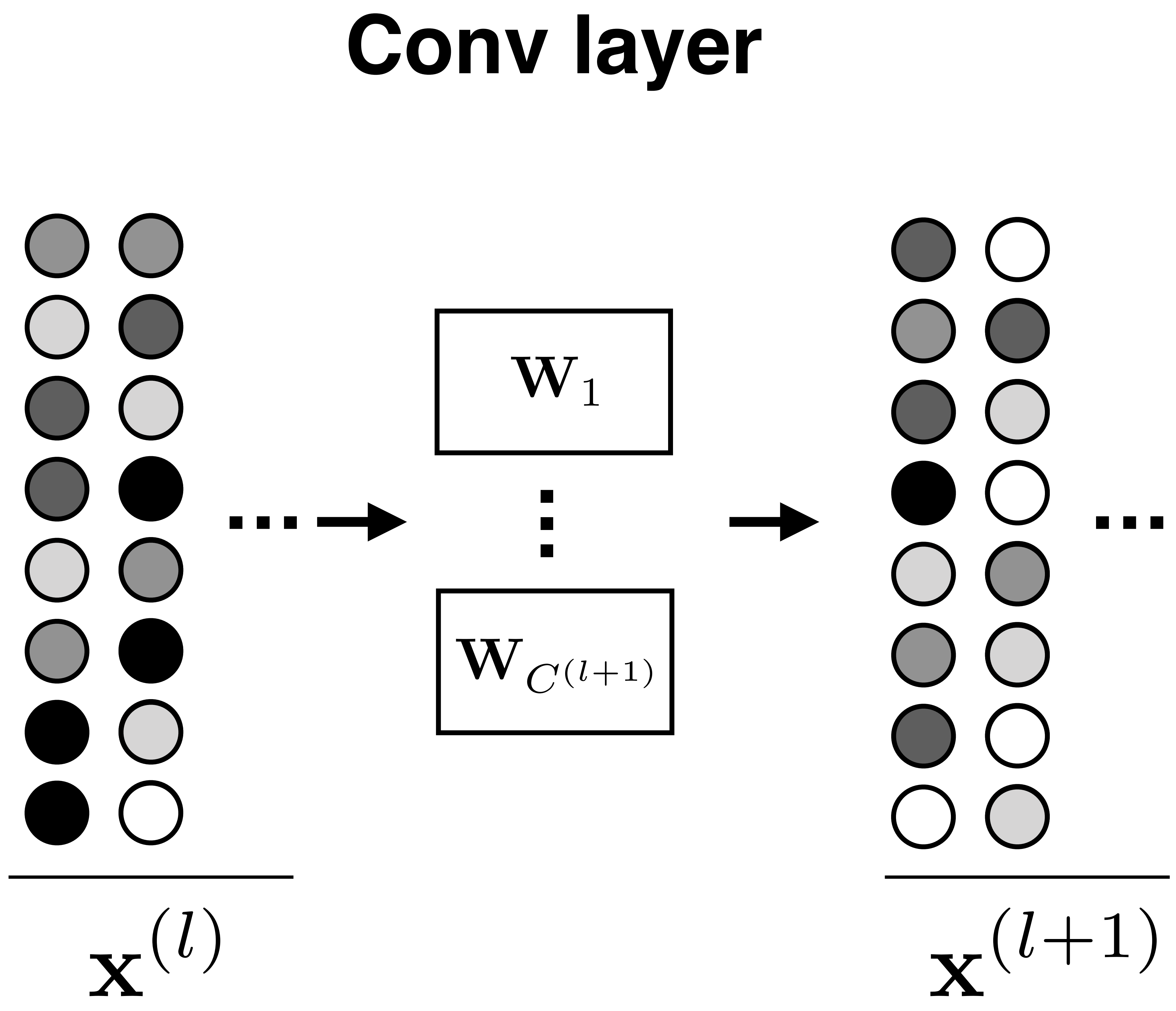
$$\mathbb{R}^{N \times C} \to \mathbb{R}^{N \times 1}$$

Multiple channels



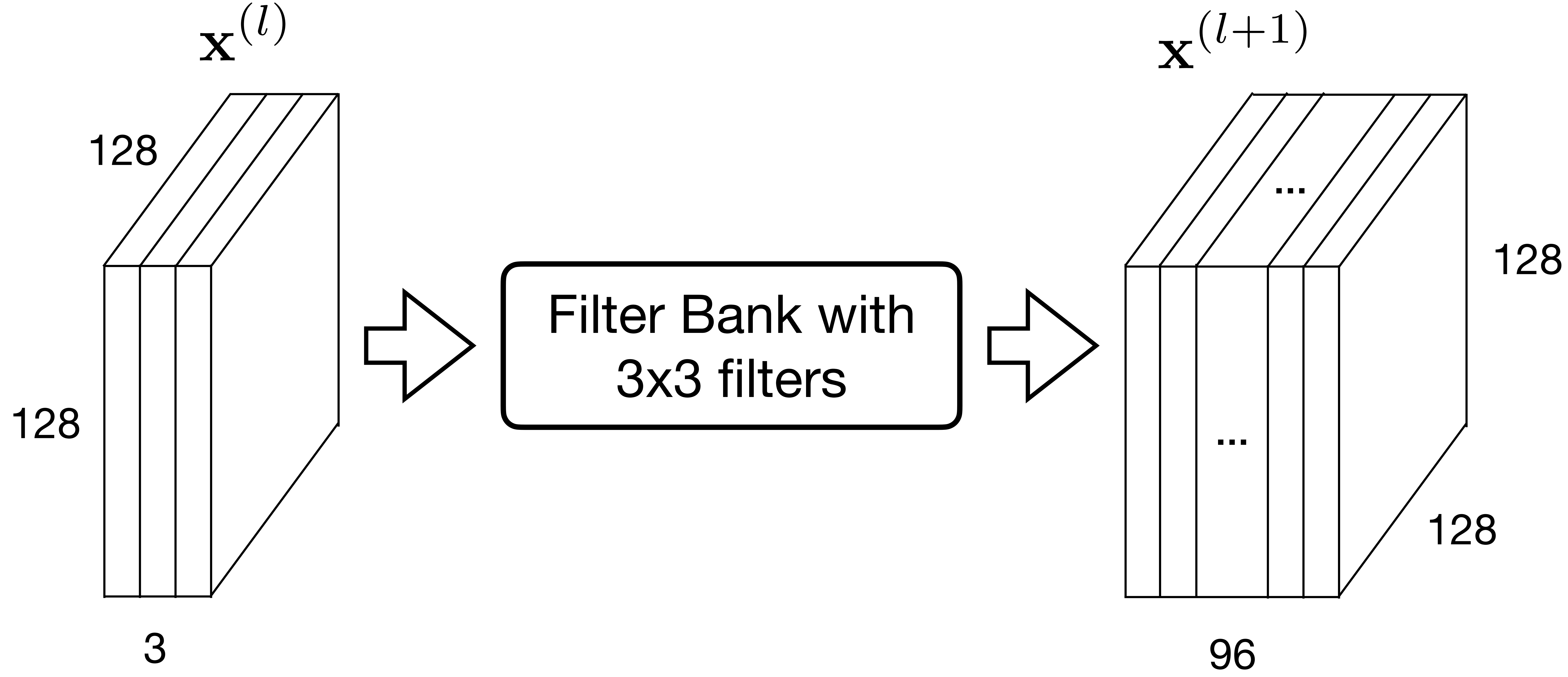
$$\mathbb{R}^{N\times C^{(0)}} \to \mathbb{R}^{N\times C^{(1)}}$$

Multiple channels



$$\mathbb{R}^{N \times C^{(l)}} \to \mathbb{R}^{N \times C^{(l+1)}}$$

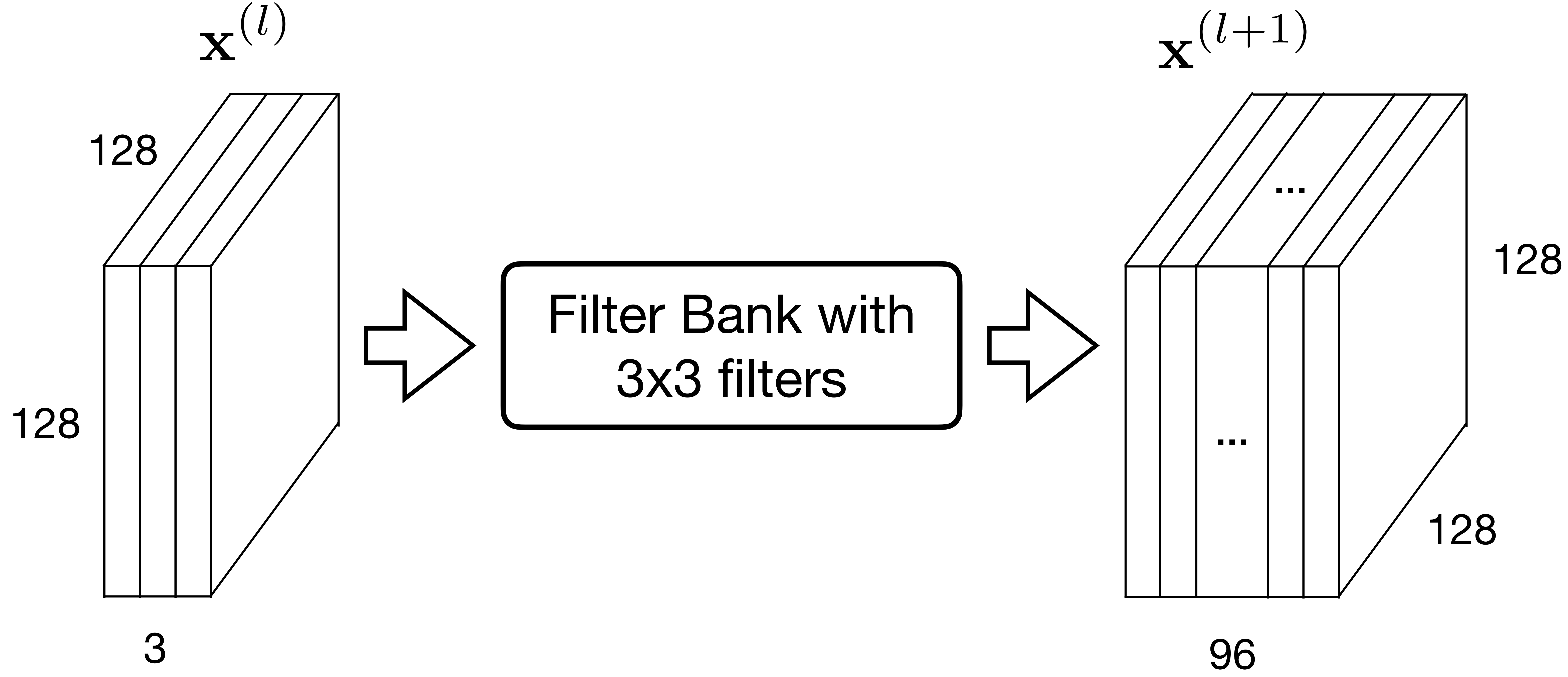
Multiple channels: Example



How many parameters does *each filter* have?

(a) 9 (b) 27 (c) 96 (d) 864

Multiple channels: Example



How many filters are in the bank?

(a) 3 (b) 27 (c) 96 (d) can't say

Filter sizes

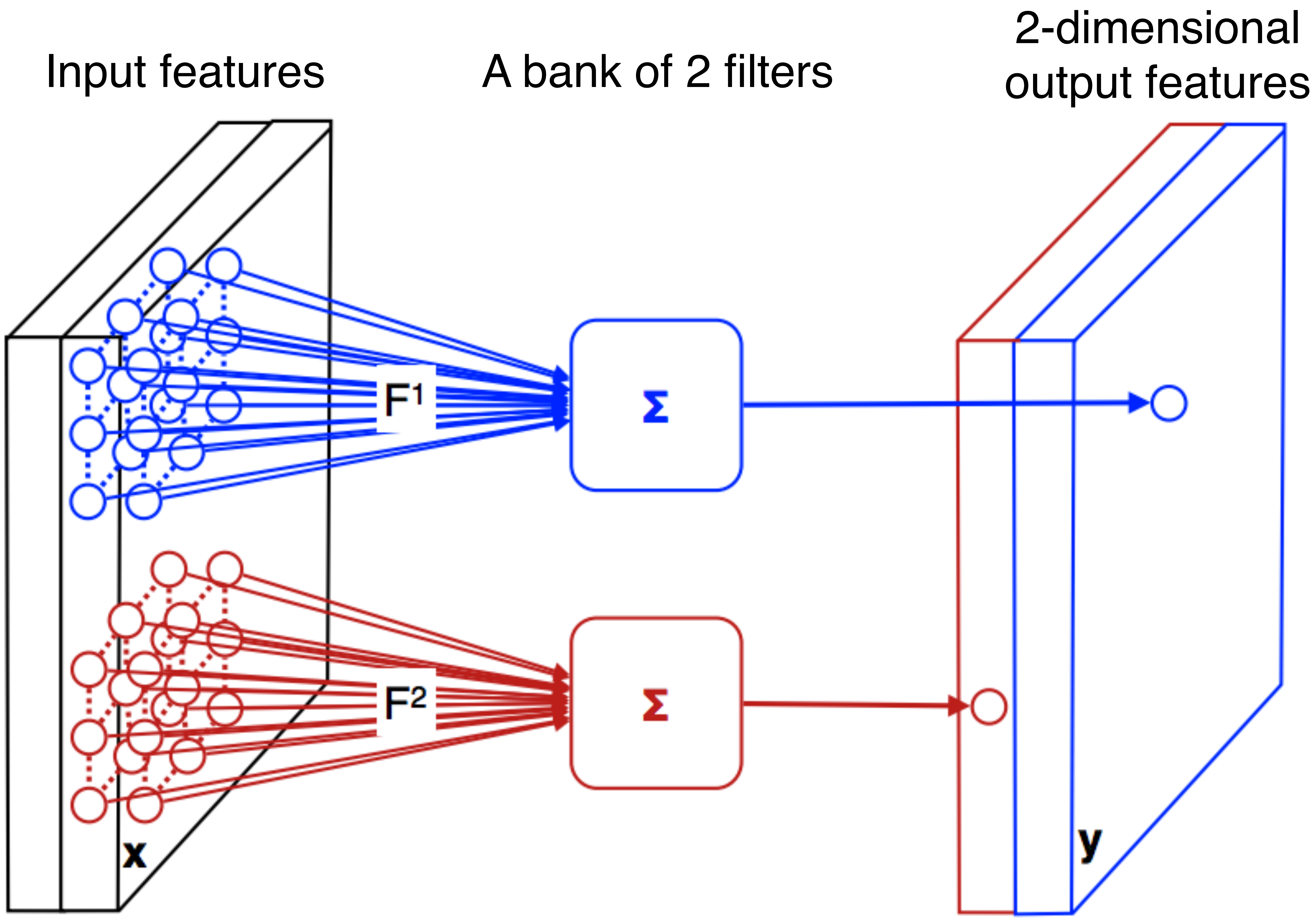
When mapping from

$$\mathbf{x}^{(l)} \in \mathbb{R}^{H \times W \times C^{(l)}} \rightarrow \mathbf{x}^{(l+1)} \in \mathbb{R}^{H \times W \times C^{(l+1)}}$$

using an filter of spatial extent $M \times N$

Number of parameters per filter: $M \times N \times C^{(l)}$

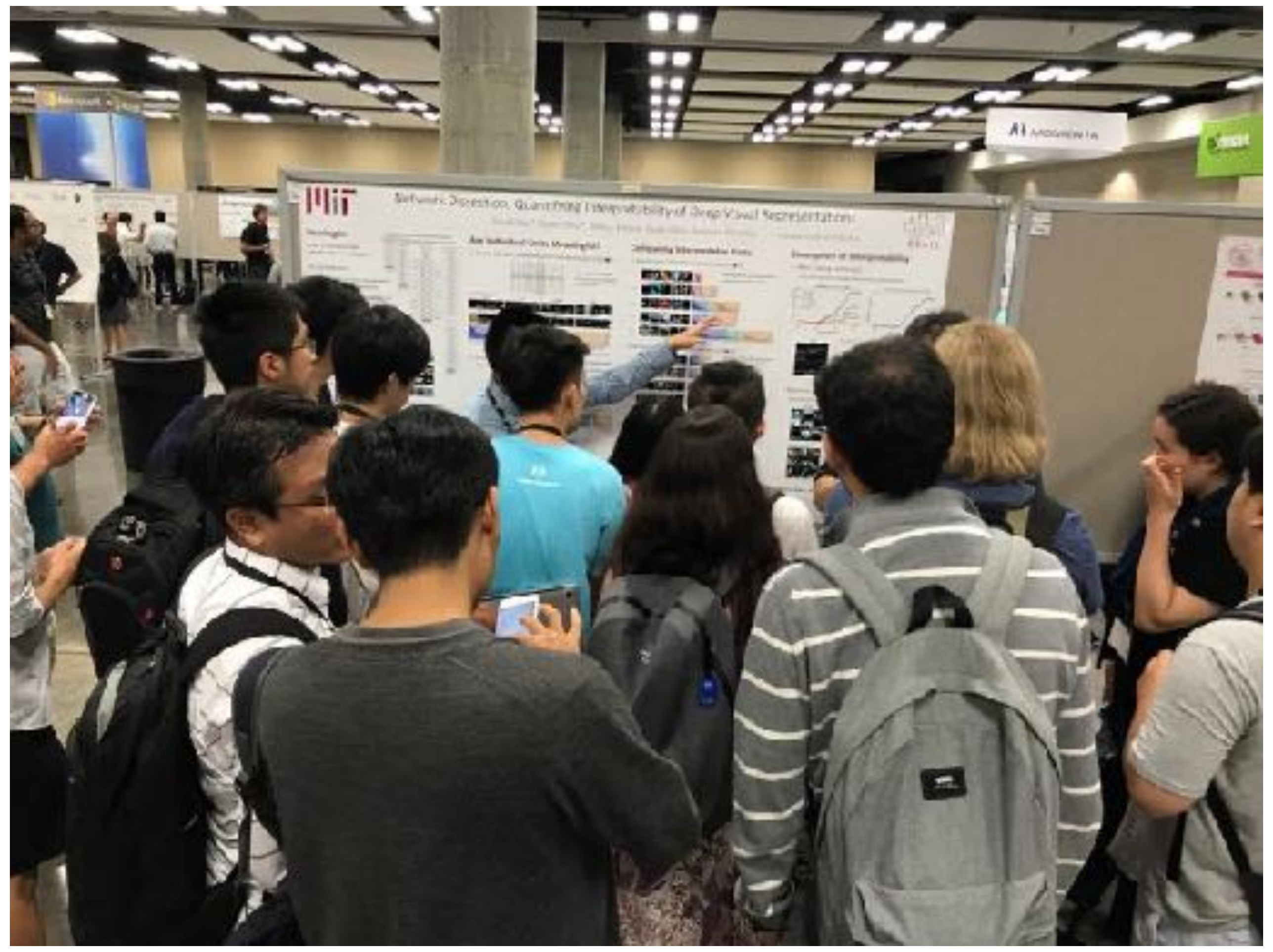
Number of filters: $C^{(l+1)}$



$$\mathbb{R}^{H \times W \times C^{(l)}} \rightarrow \mathbb{R}^{H \times W \times C^{(l+1)}}$$

[Figure from Andrea Vedaldi]

Image classification

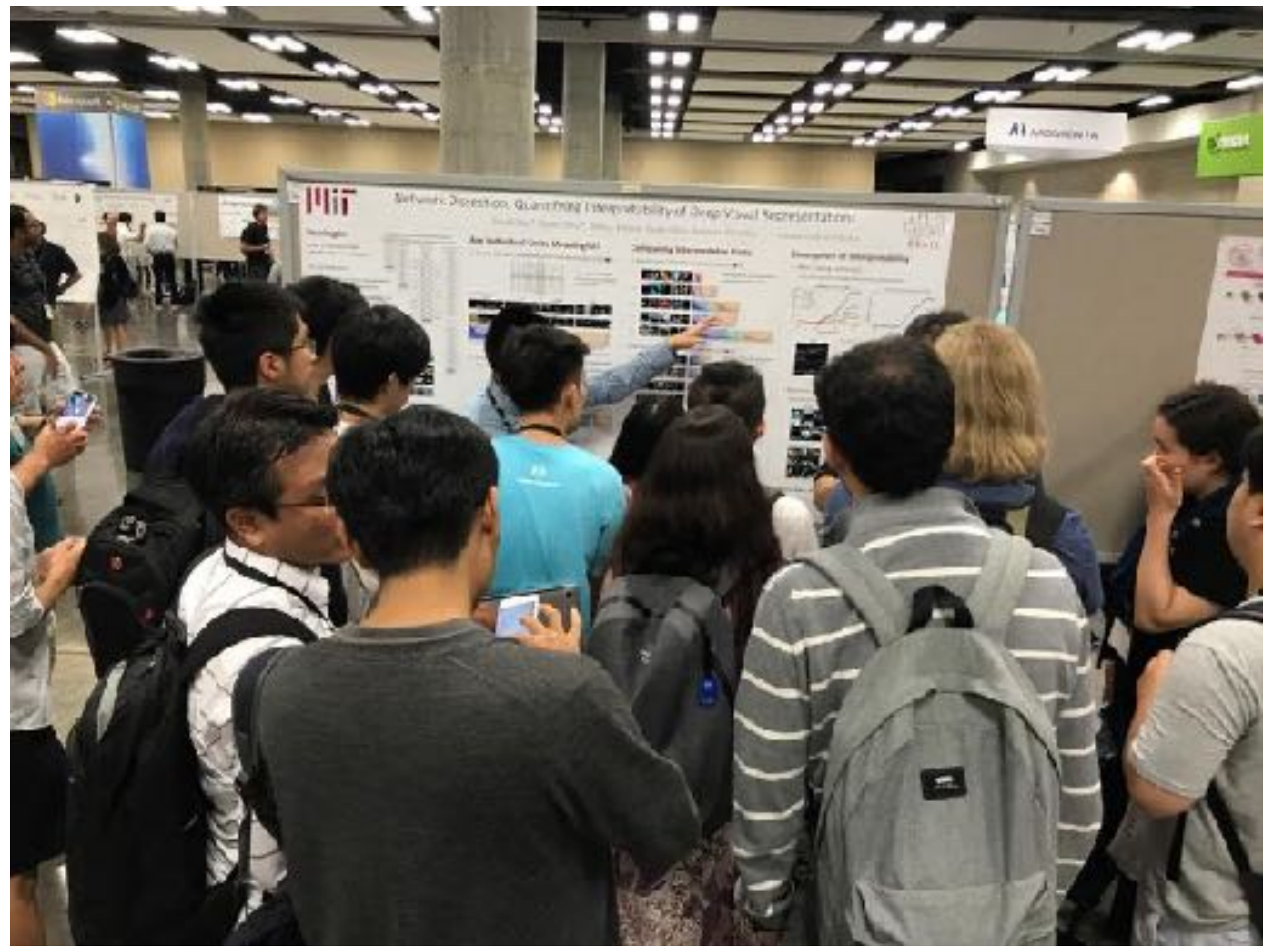


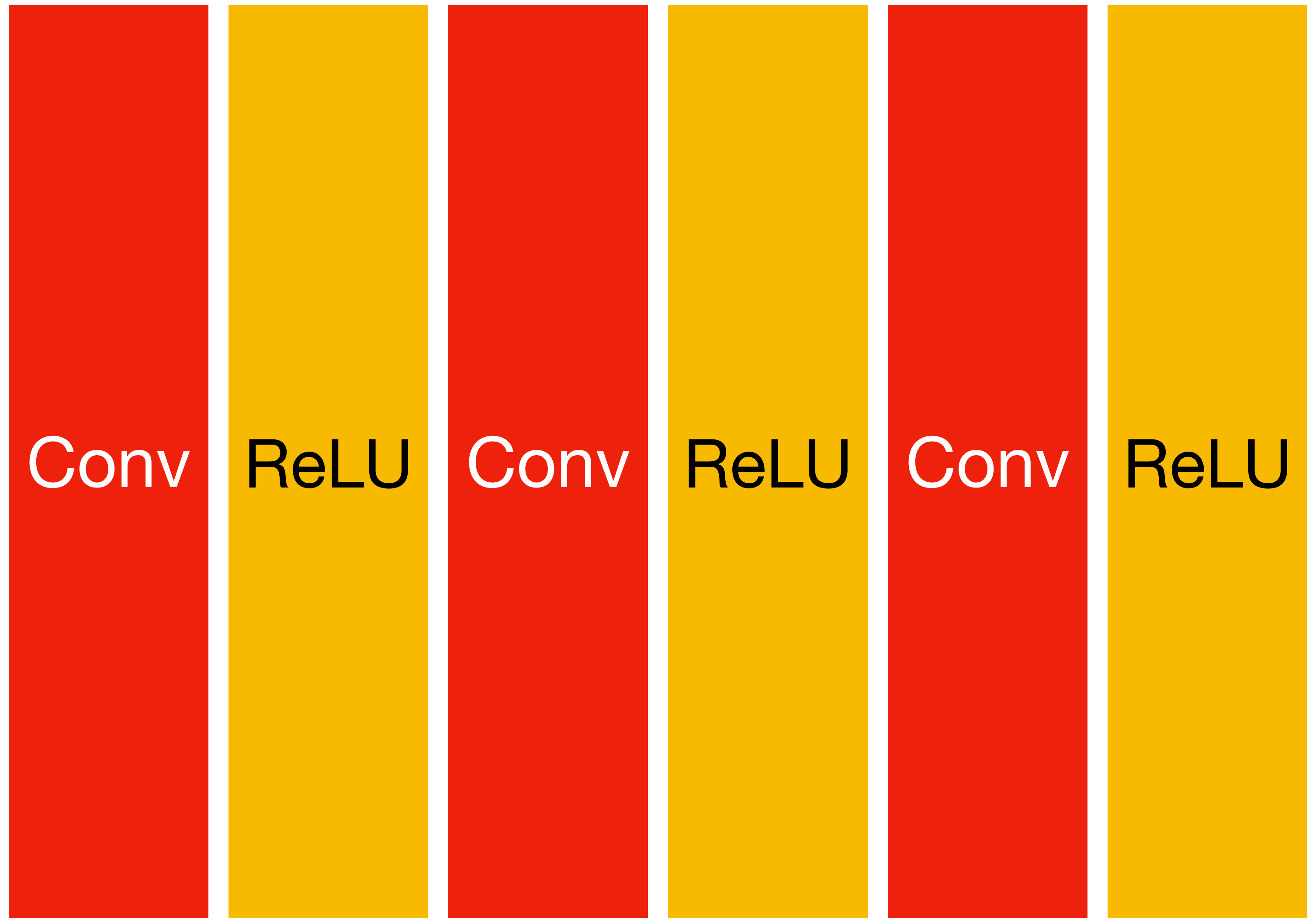
→ **Classifier** → “Indoor booth”

image **x**

label y

Image classification



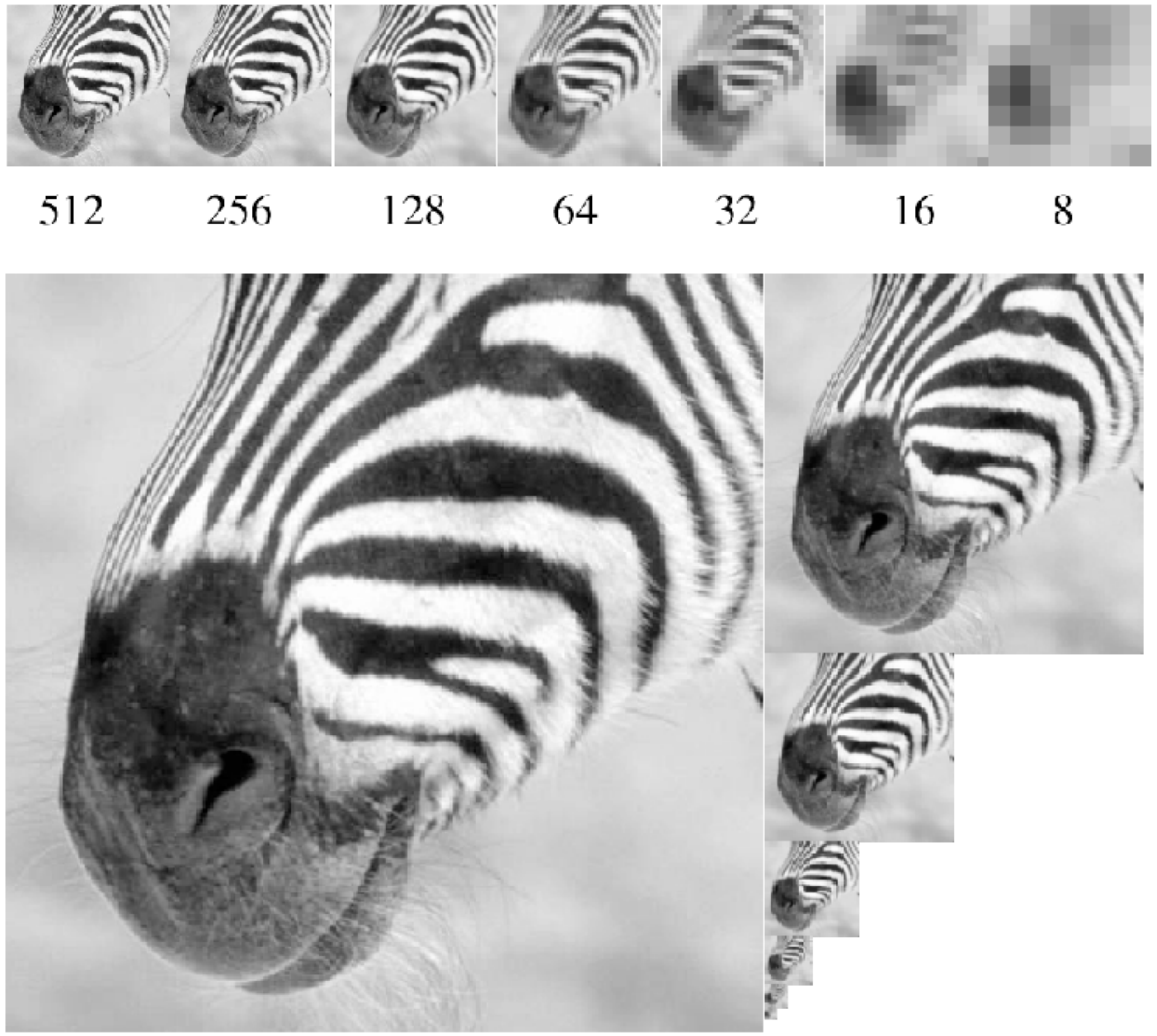


“Indoor booth”

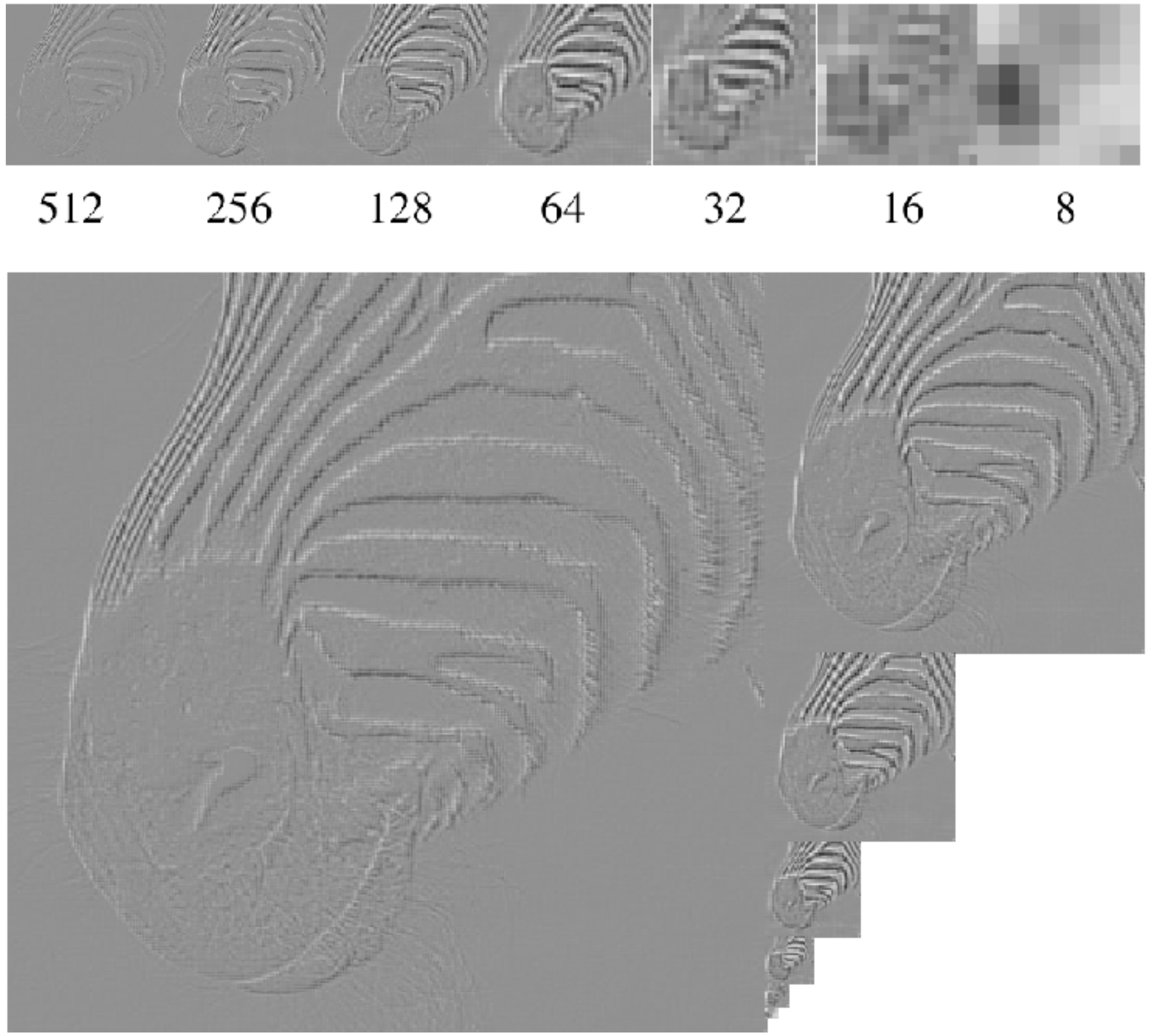
image **x**

label y

Multiscale representations are great!



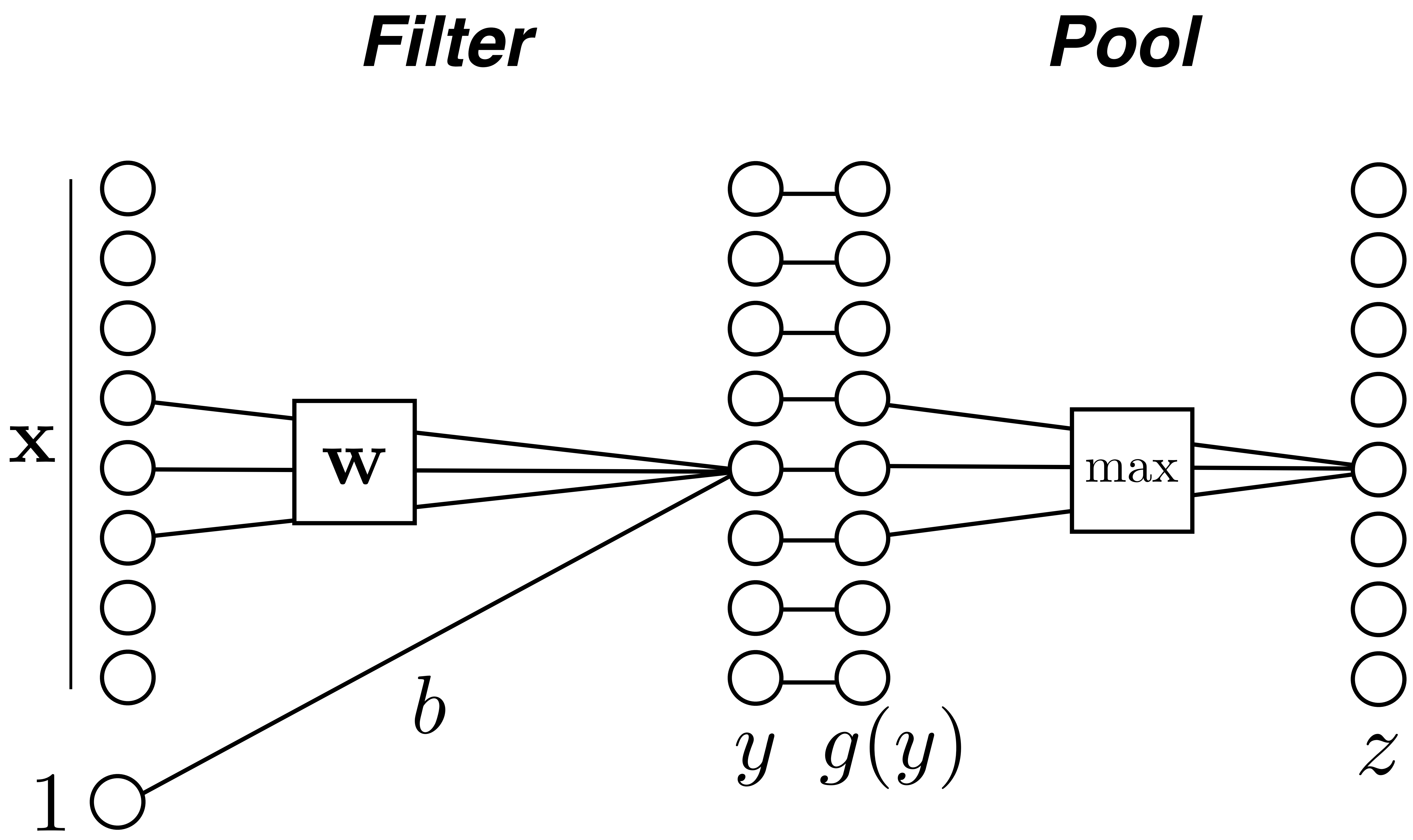
Gaussian Pyr



Laplacian Pyr

How can we use multi-scale modeling in Convnets?

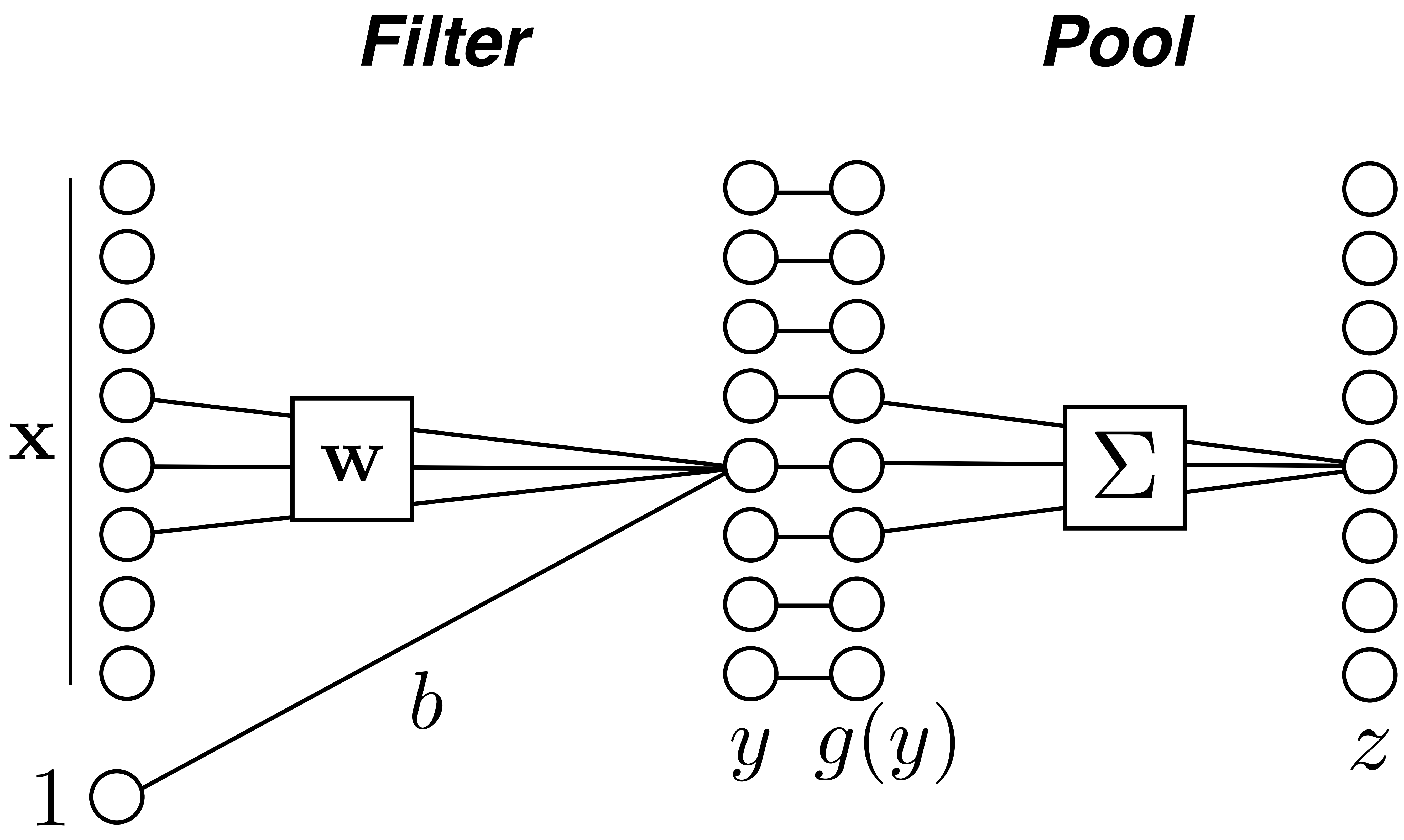
Pooling



Max pooling

$$z_k = \max_{j \in \mathcal{N}(j)} g(y_j)$$

Pooling



Max pooling

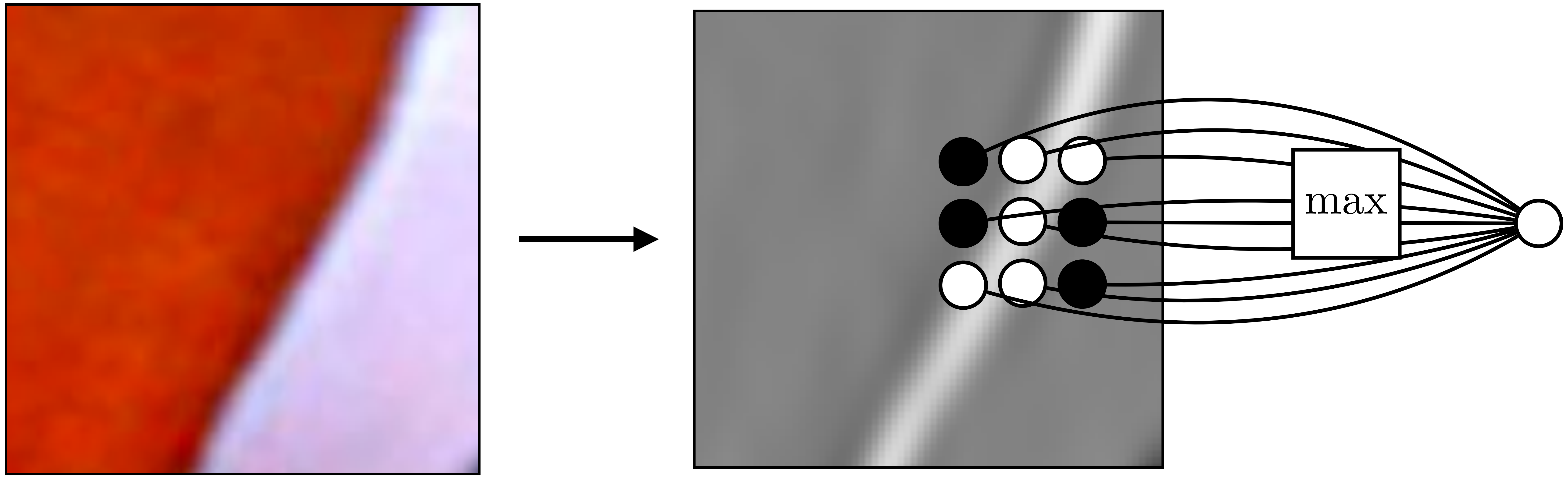
$$z_k = \max_{j \in \mathcal{N}(j)} g(y_j)$$

Mean pooling

$$z_k = \frac{1}{|\mathcal{N}|} \sum_{j \in \mathcal{N}(j)} g(y_j)$$

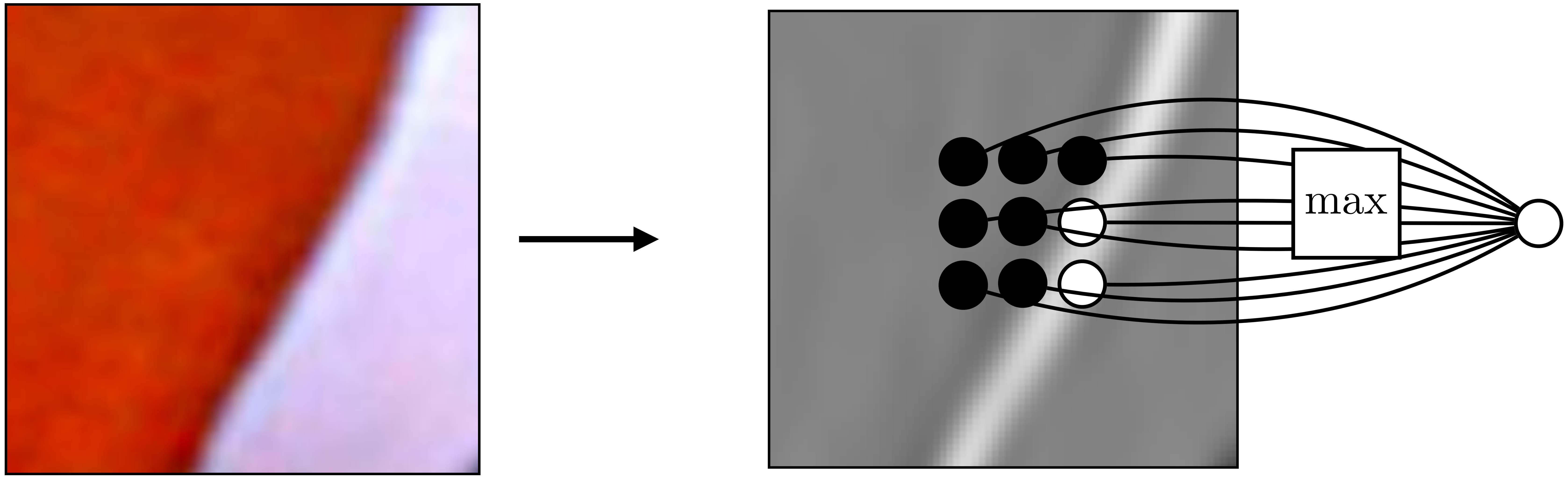
Pooling — Why?

Pooling across spatial locations achieves stability w.r.t. small translations:



Pooling — Why?

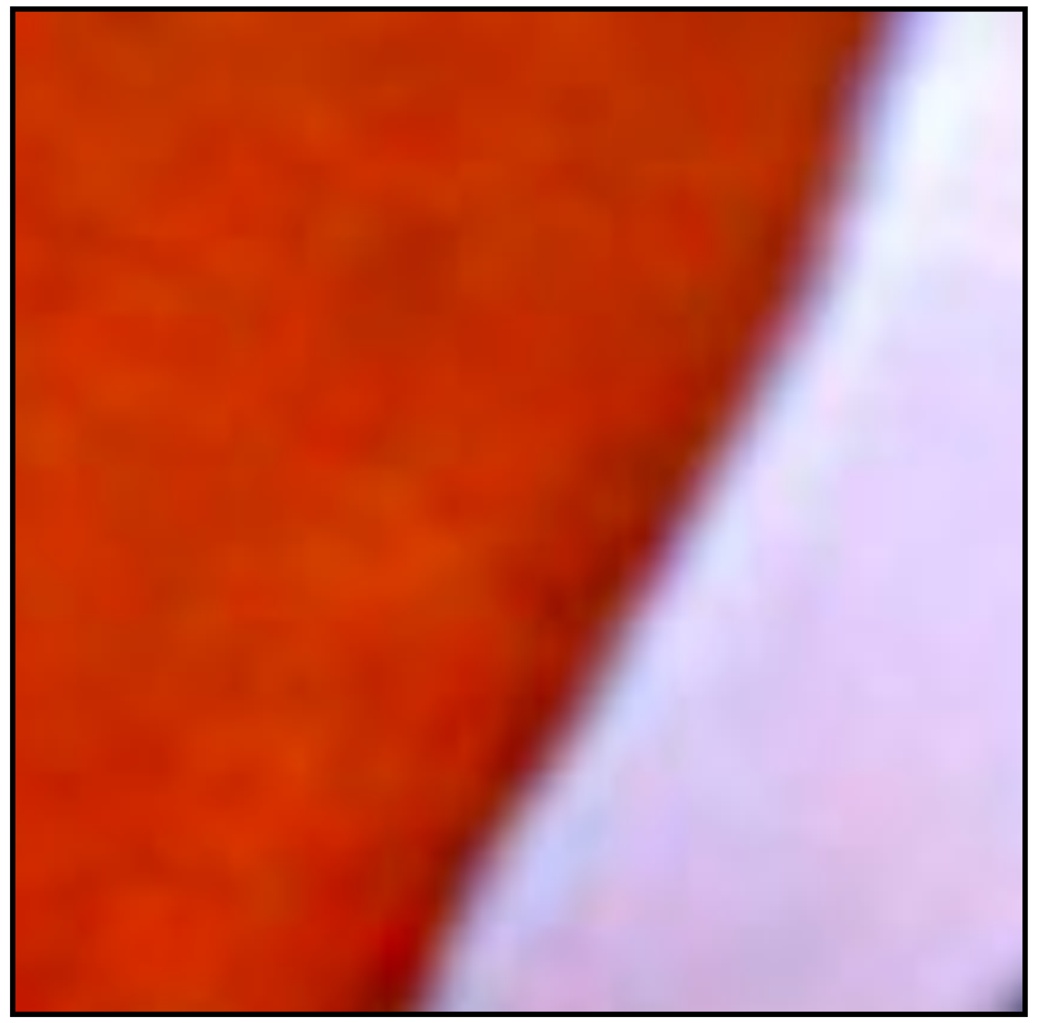
Pooling across spatial locations achieves stability w.r.t. small translations:

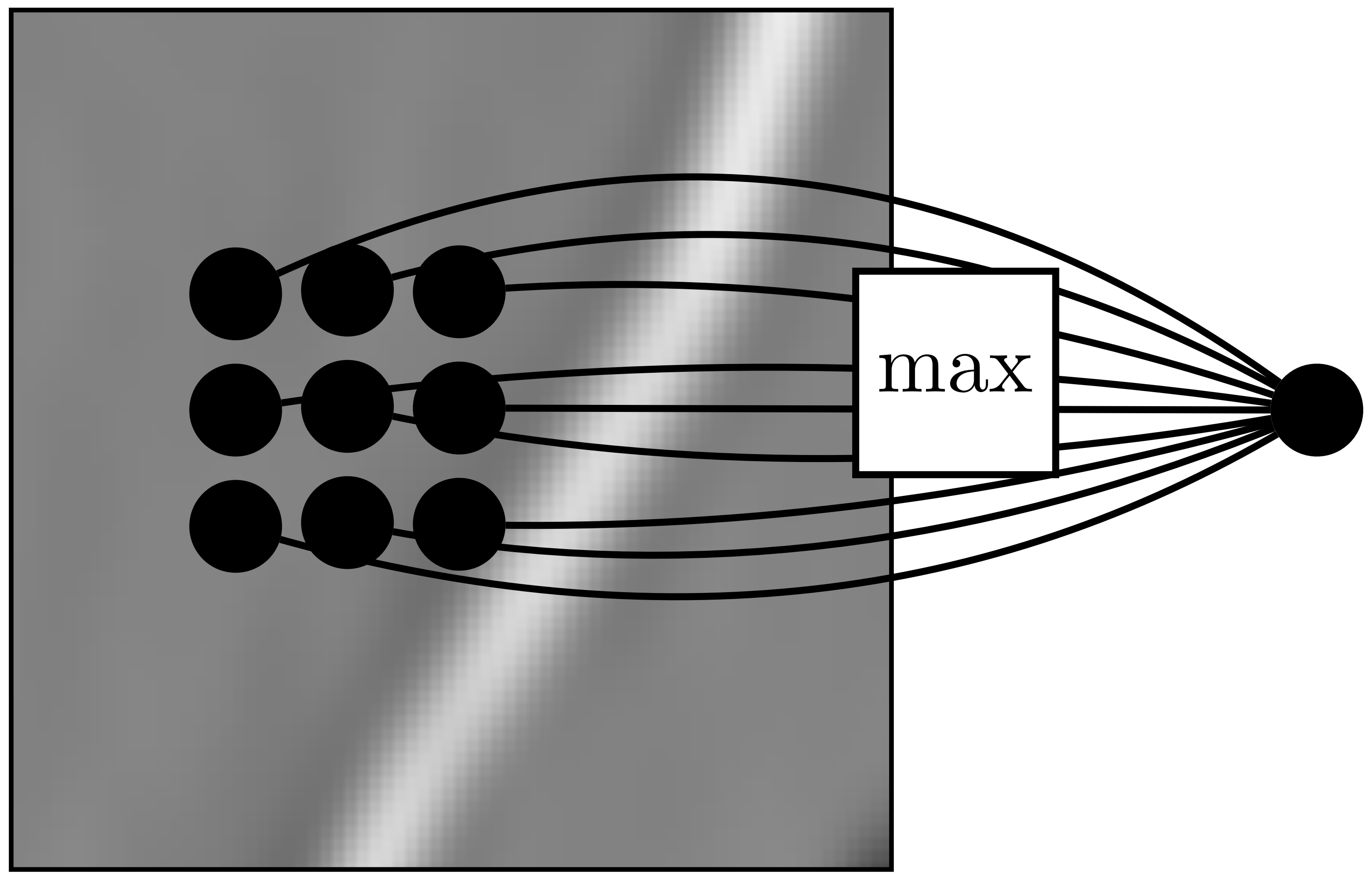


large response regardless of exact position of edge

Pooling — Why?

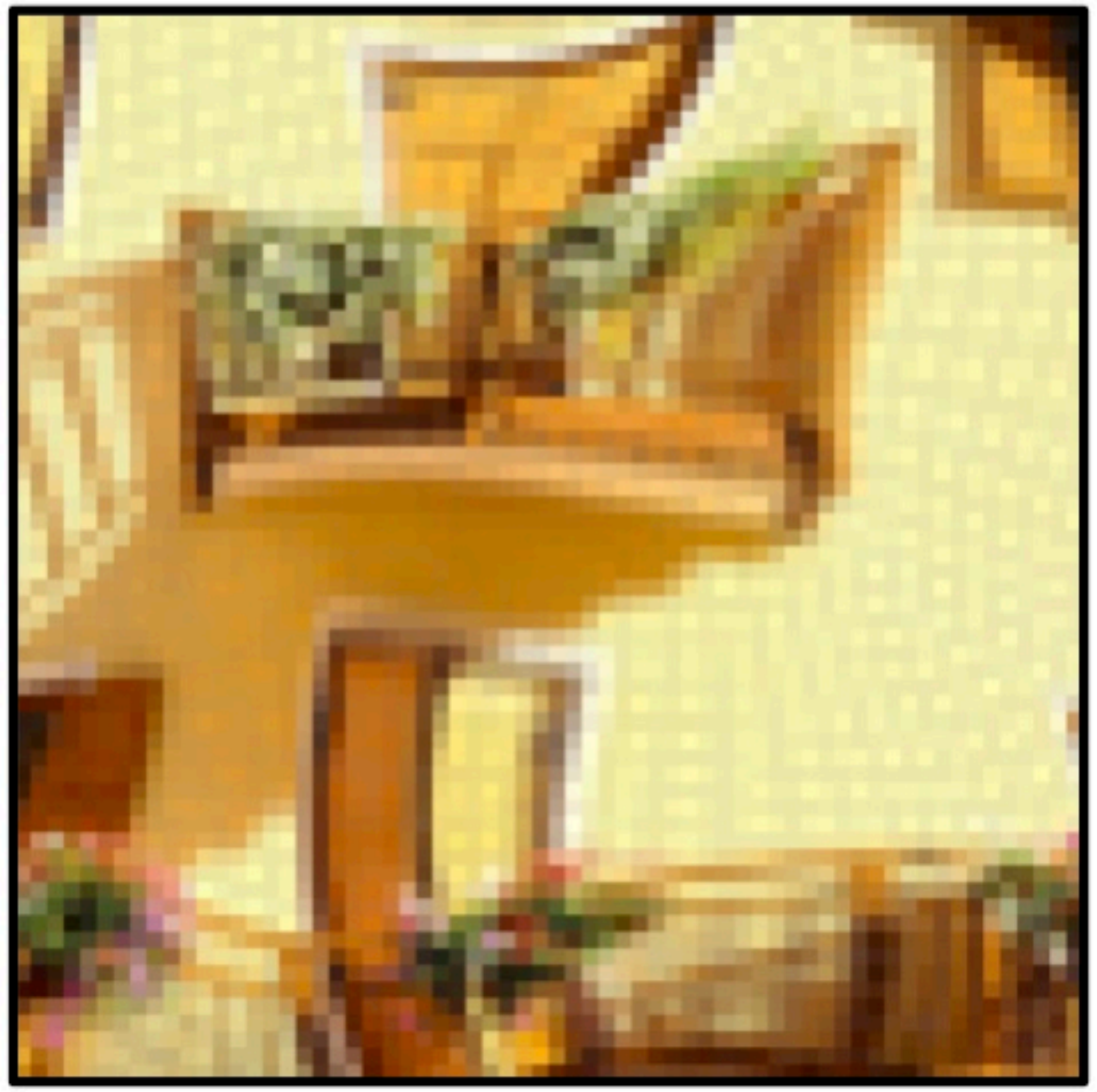
Pooling across spatial locations achieves stability w.r.t. small translations:





CNNs are stable w.r.t. diffeomorphisms

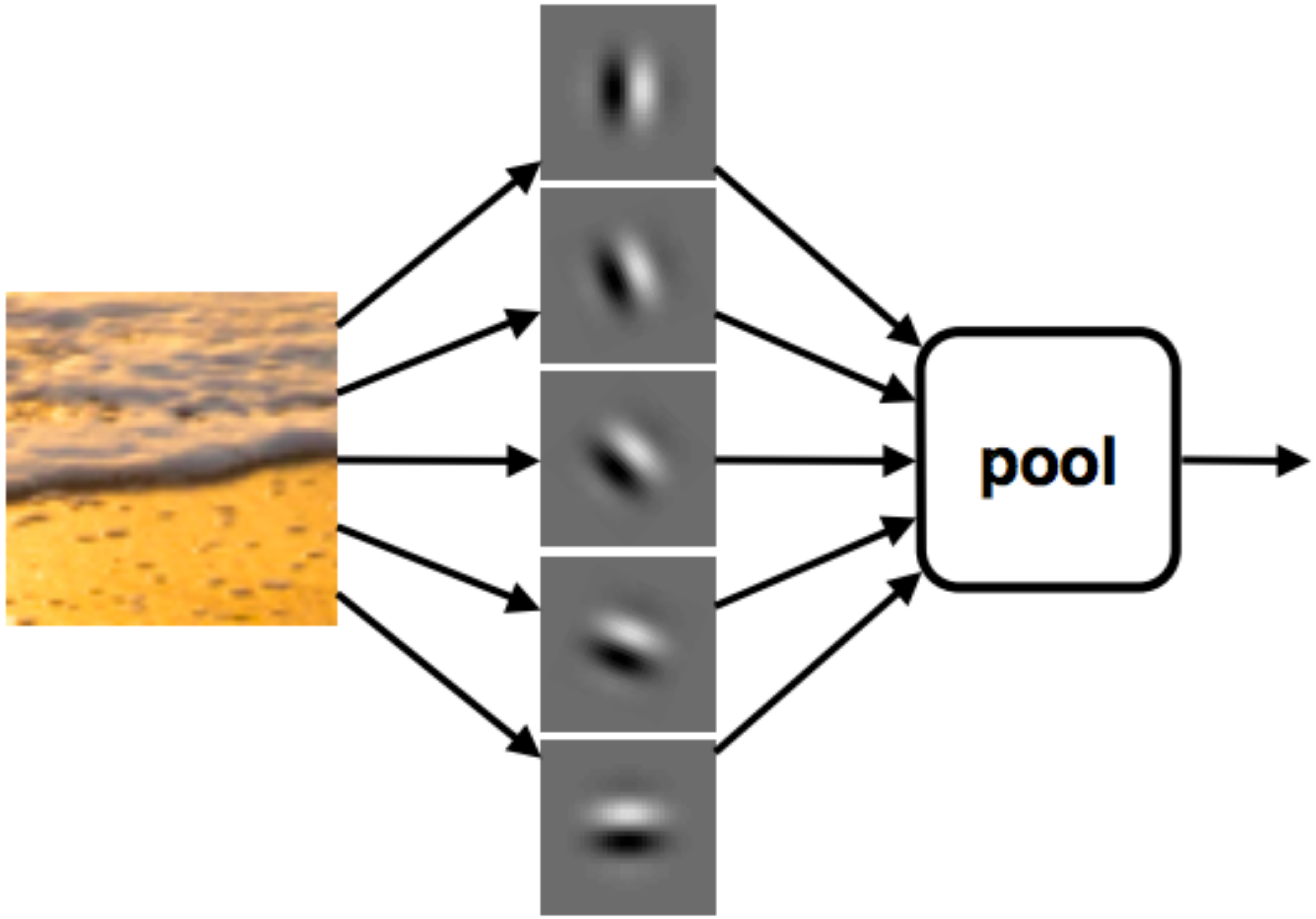
$$\approx$$



[“Unreasonable effectiveness of Deep Features as a Perceptual Metric”, Zhang et al. 2018]

Pooling — Why?

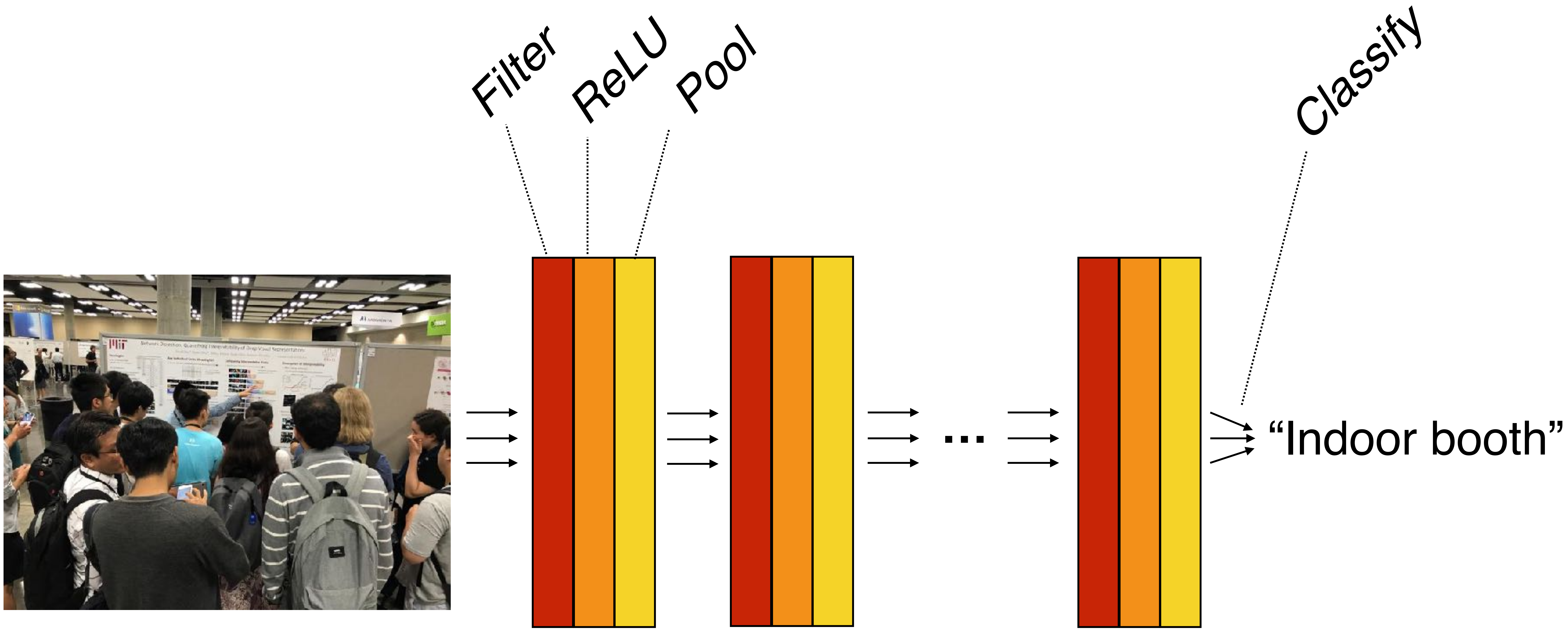
Pooling across feature channels (filter outputs) can achieve other kinds of invariances:



large
response for
any edge,
regardless of
its orientation

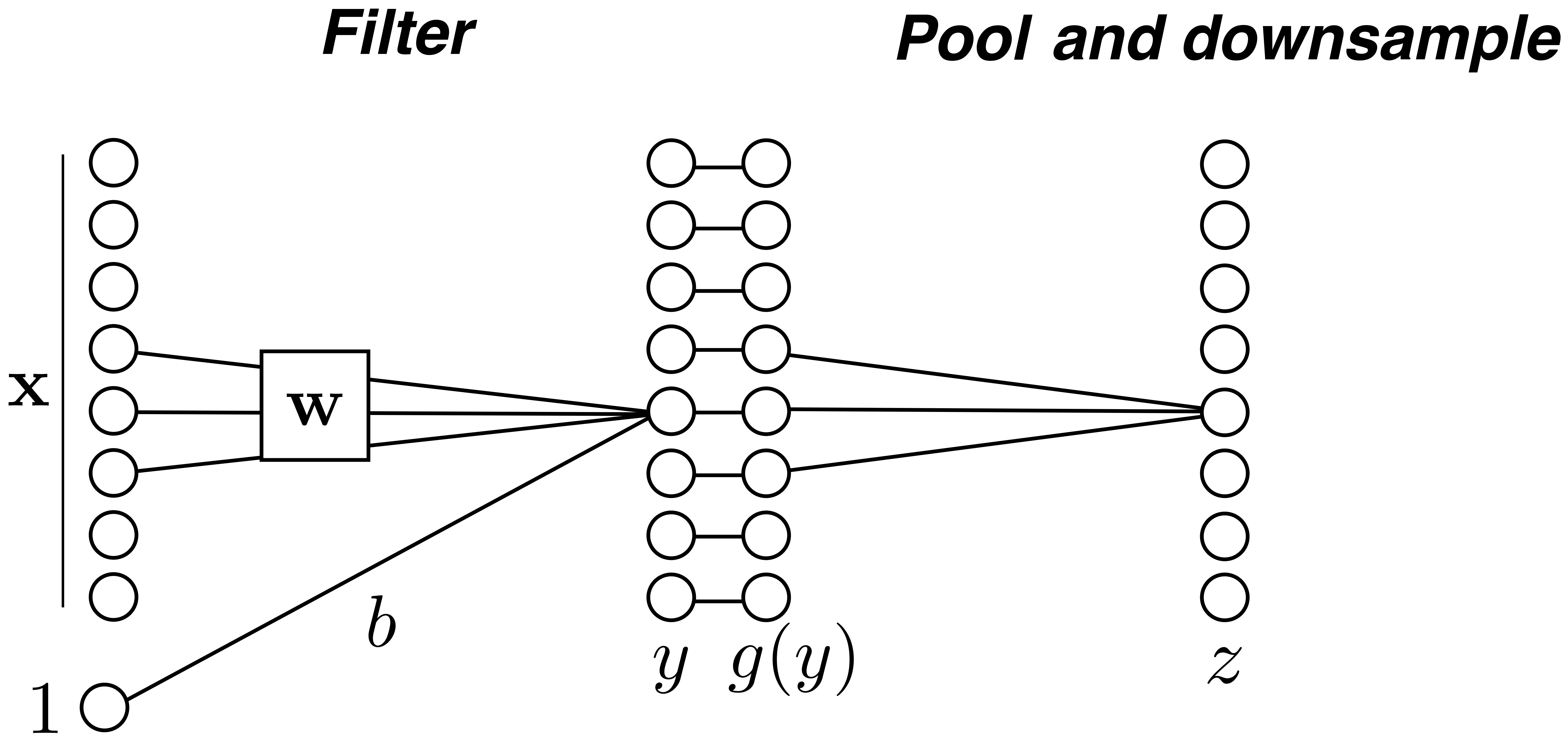
[Derived from slide by Andrea Vedaldi]

Computation in a neural net

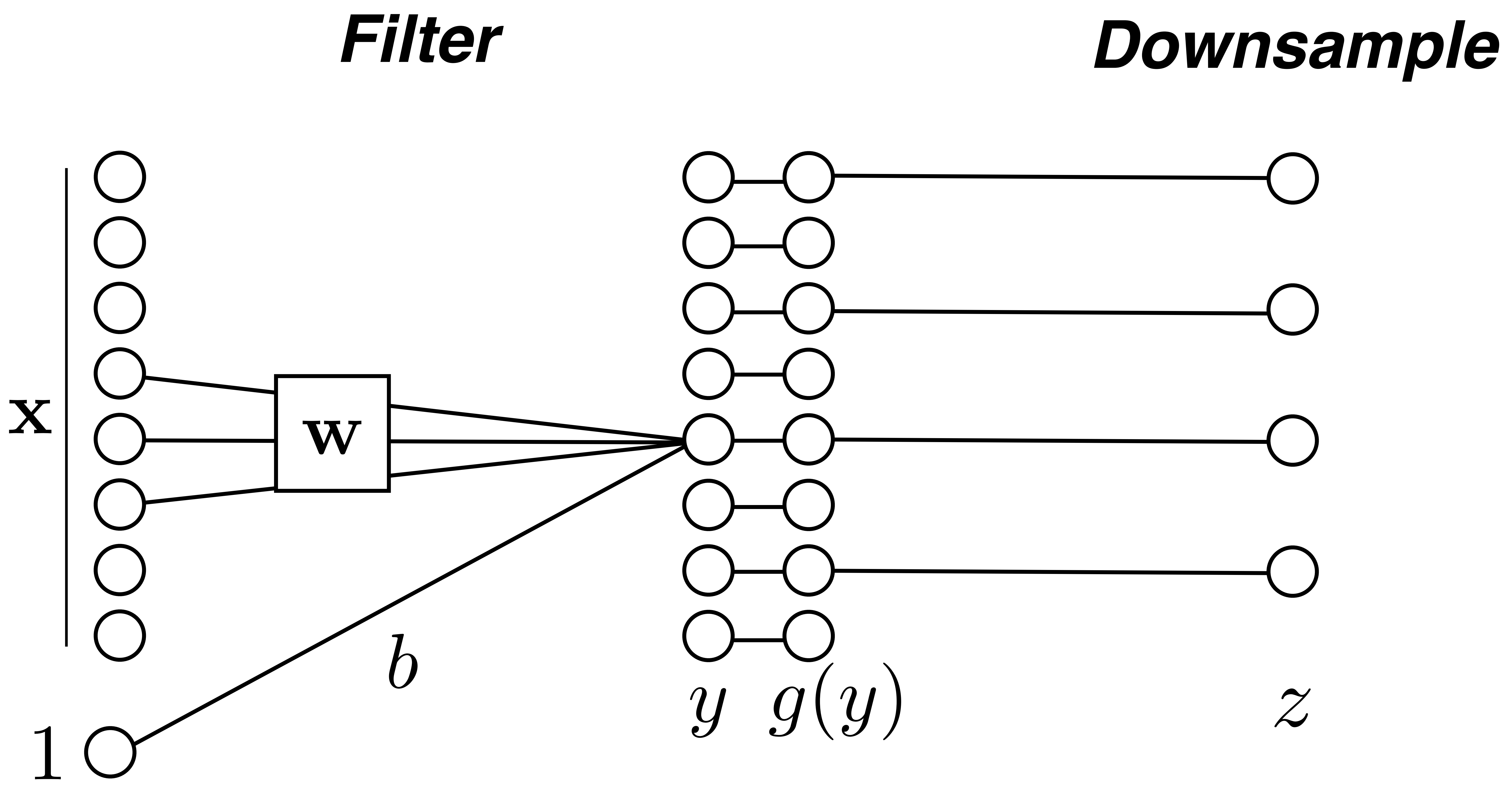


$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

Downsampling

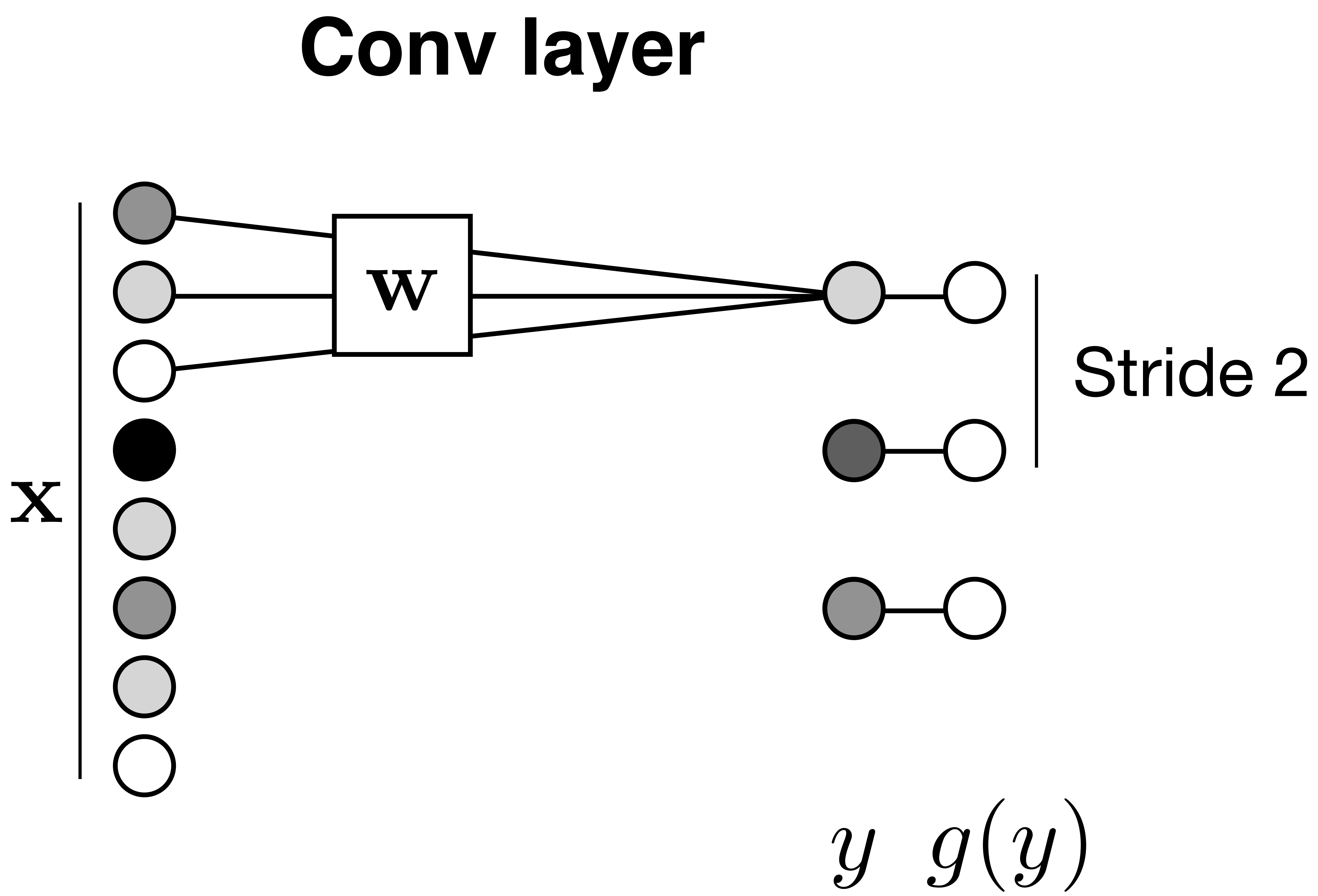


Downsampling



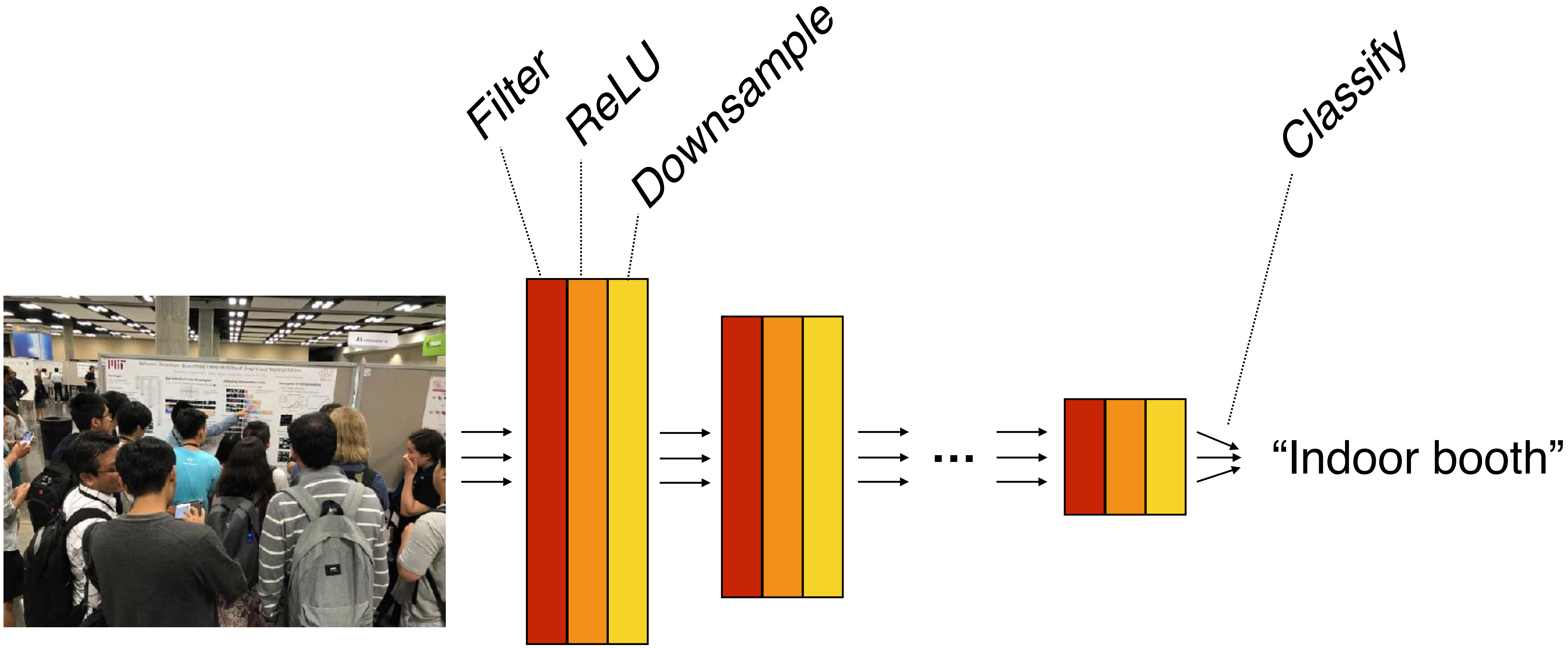
$$\mathbb{R}^{H^{(l)} \times W^{(l)} \times C^{(l)}} \rightarrow \mathbb{R}^{H^{(l+1)} \times W^{(l+1)} \times C^{(l+1)}}$$

Strided operations



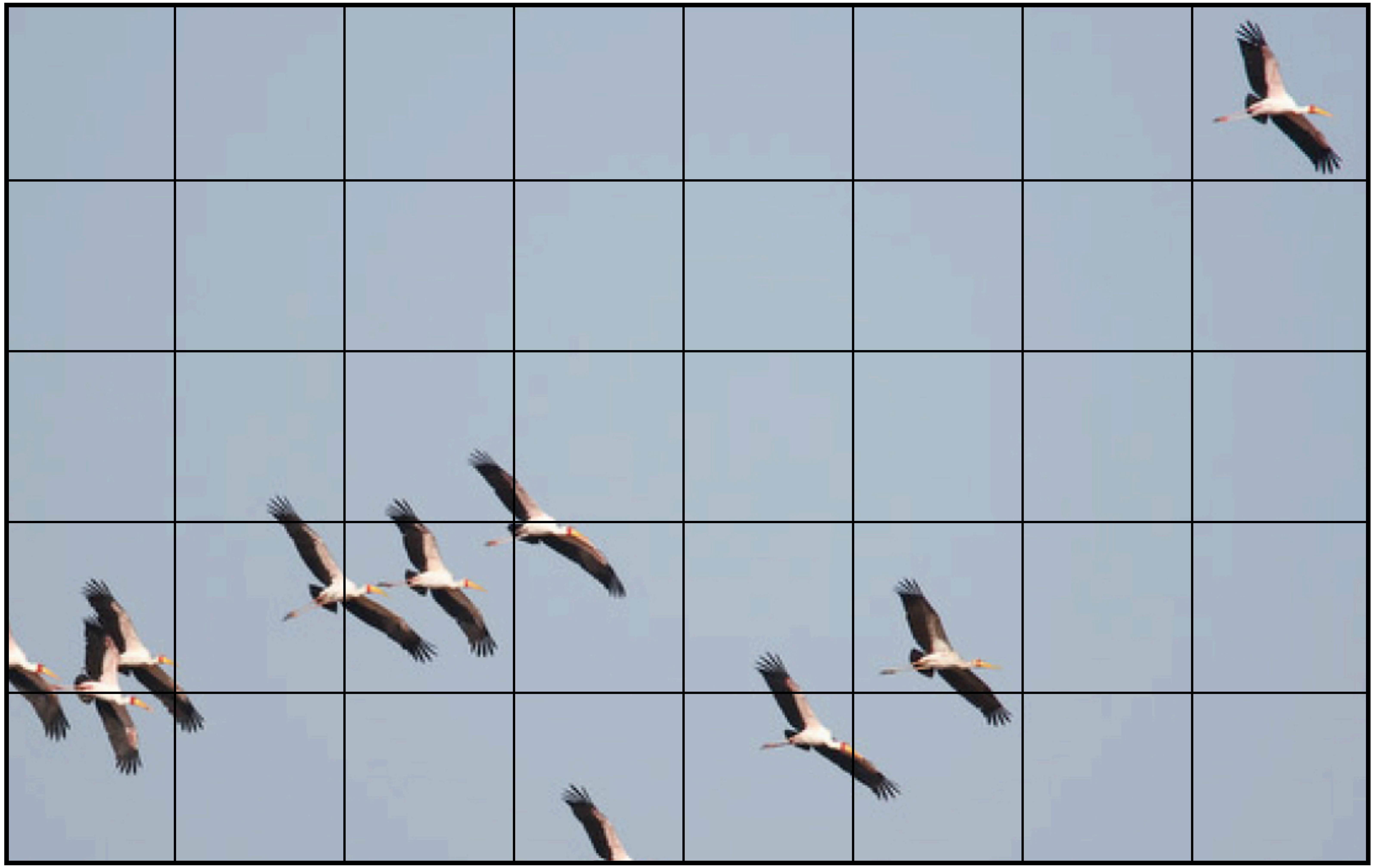
Strided operations combine a given operation (convolution or pooling) and downsampling into a single operation.

Computation in a neural net

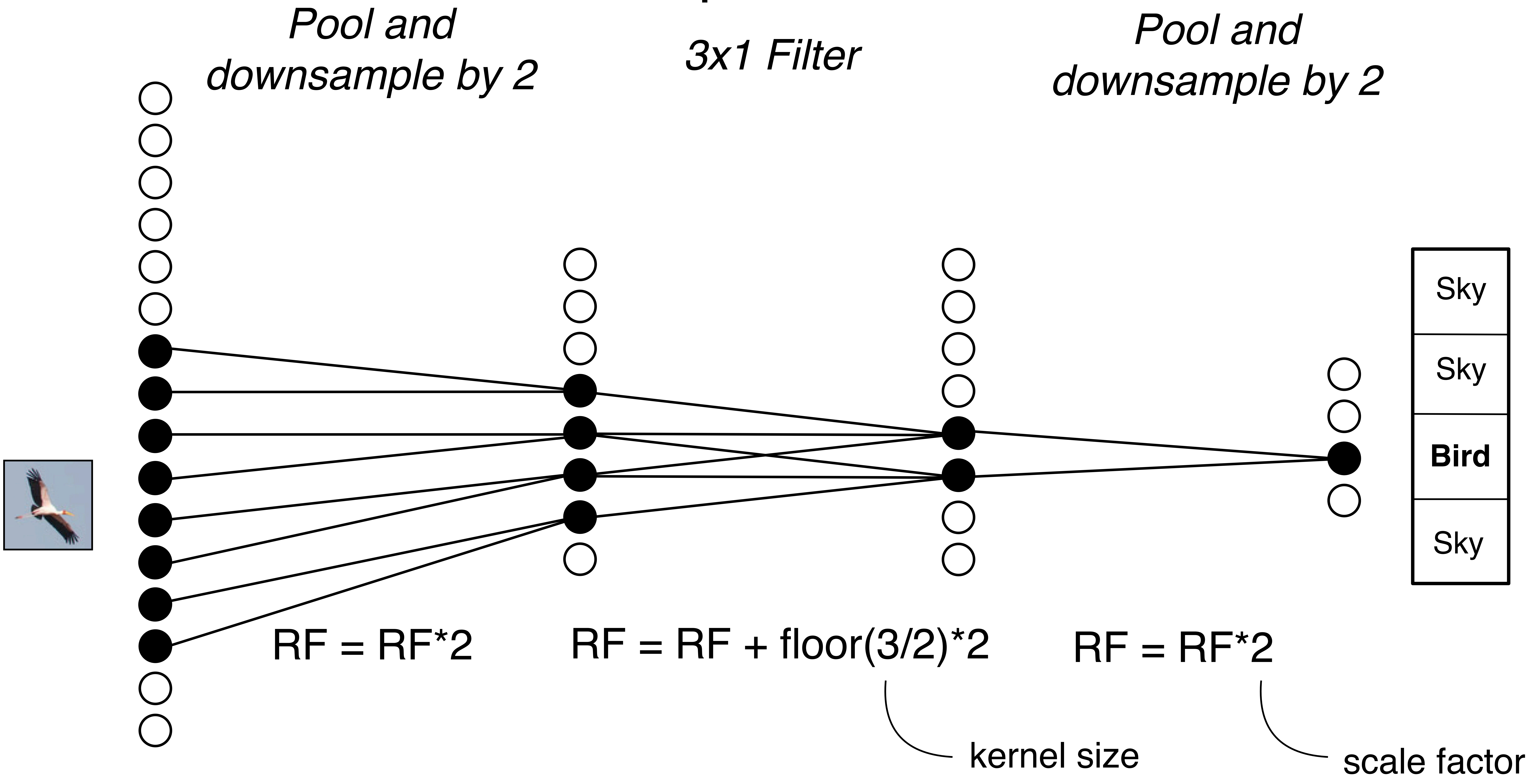


$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

Receptive fields



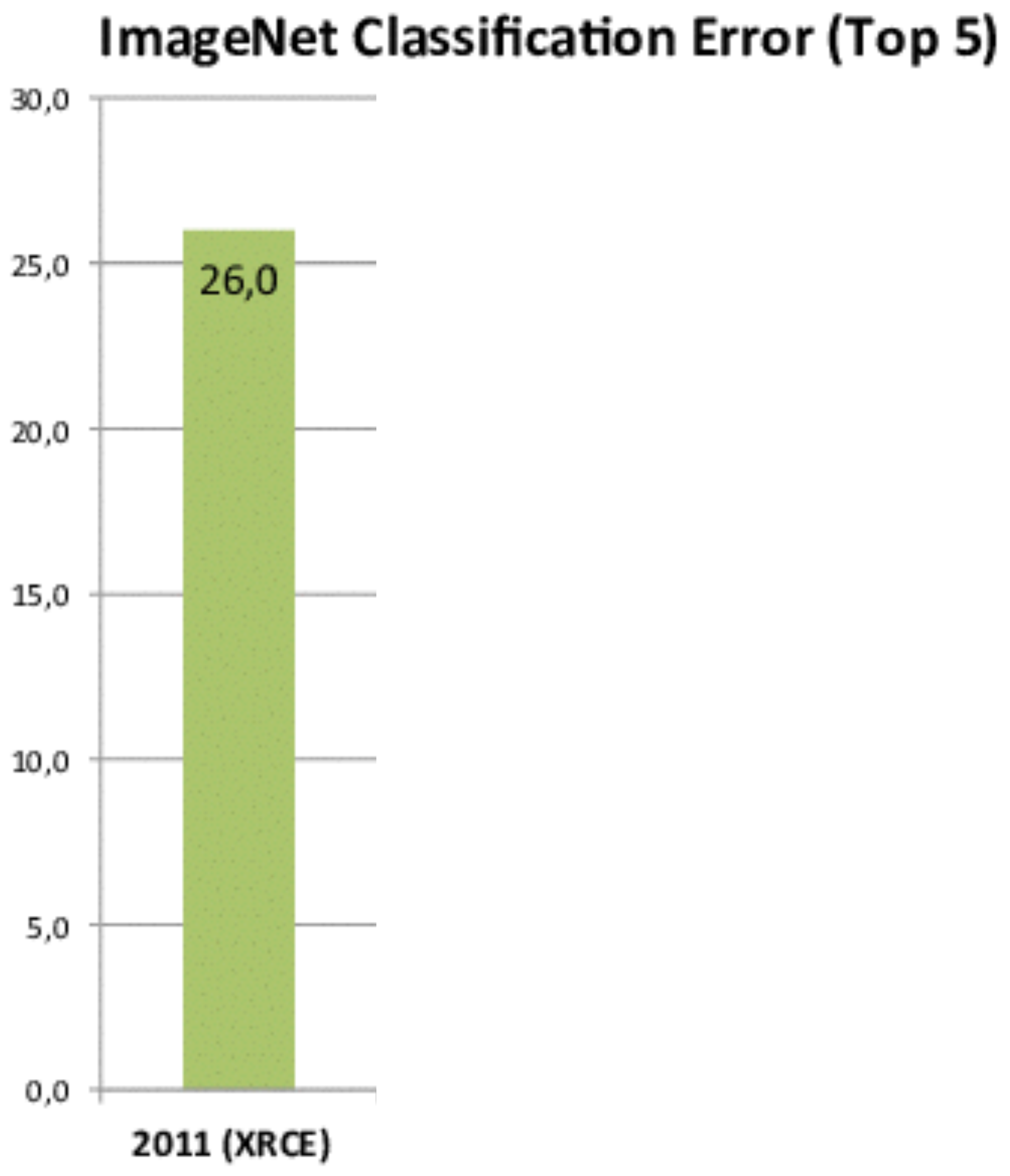
Receptive fields

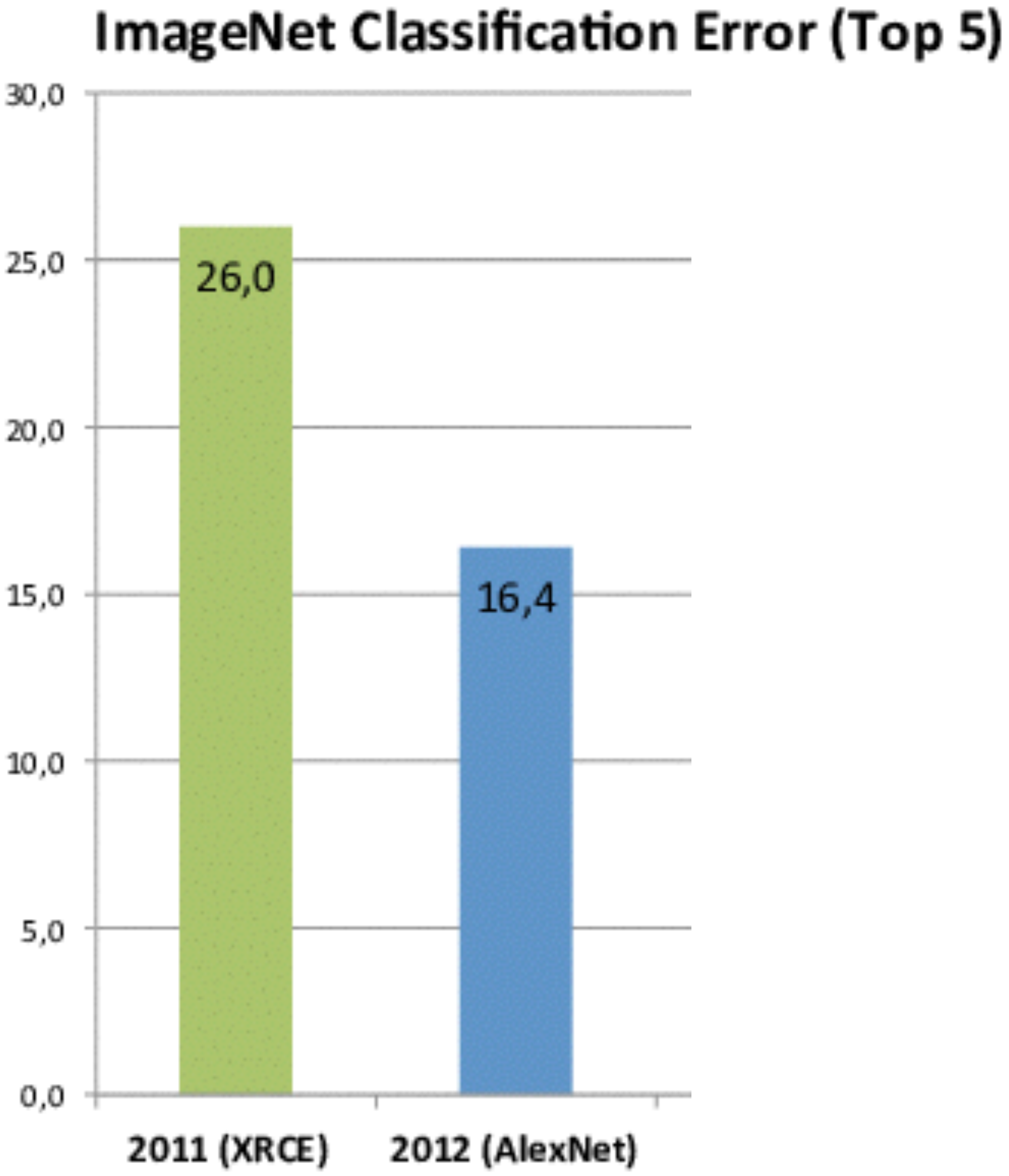


[http://fomoro.com/tools/receptive-fields/index.html]

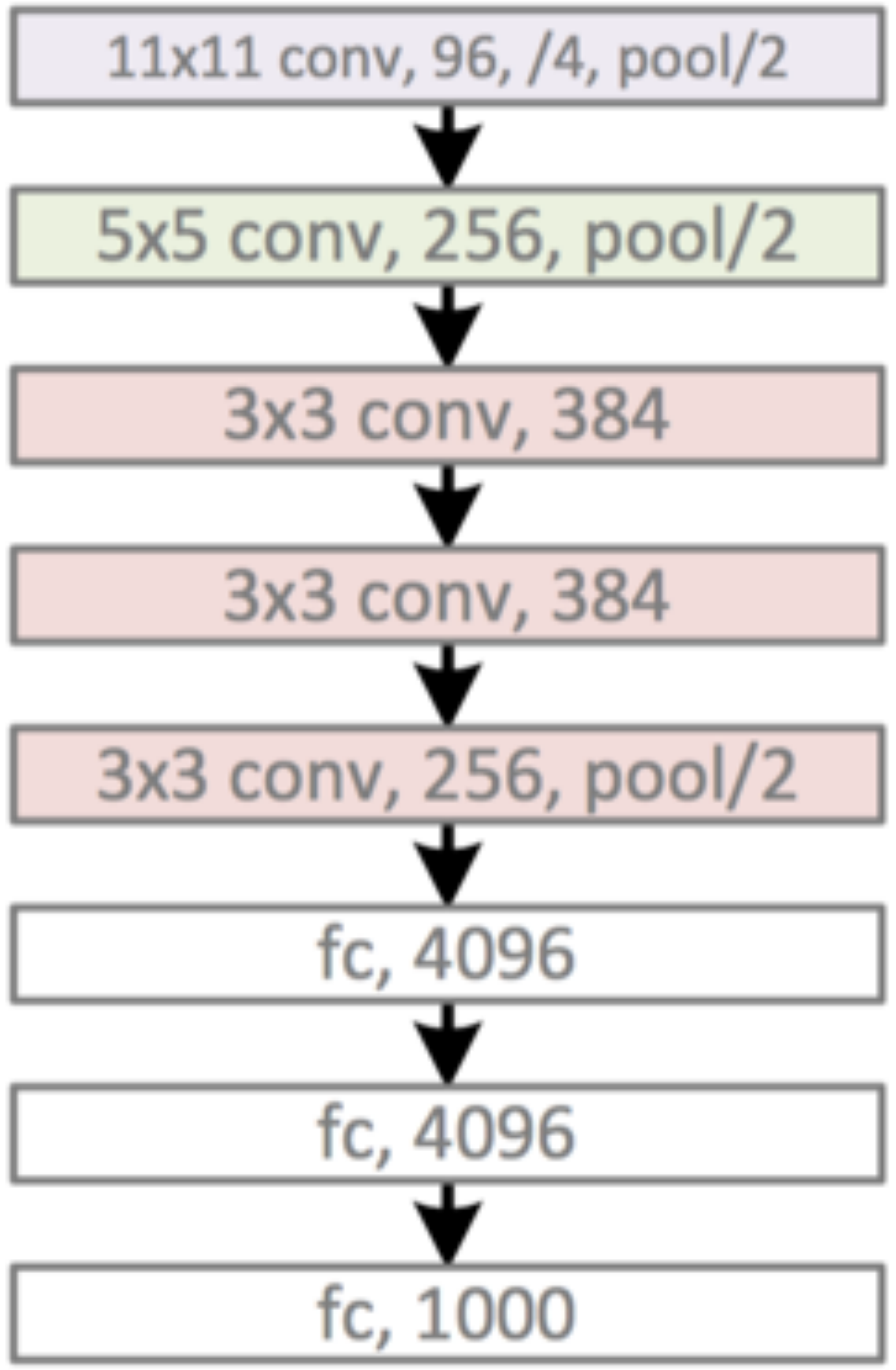
Some networks

… and what makes them work


ImageNet Classification Error (Top 5)
30,0
25,0
20,0
15,0
10,0
5,0
0,0
26,0
2011 (XRCE)



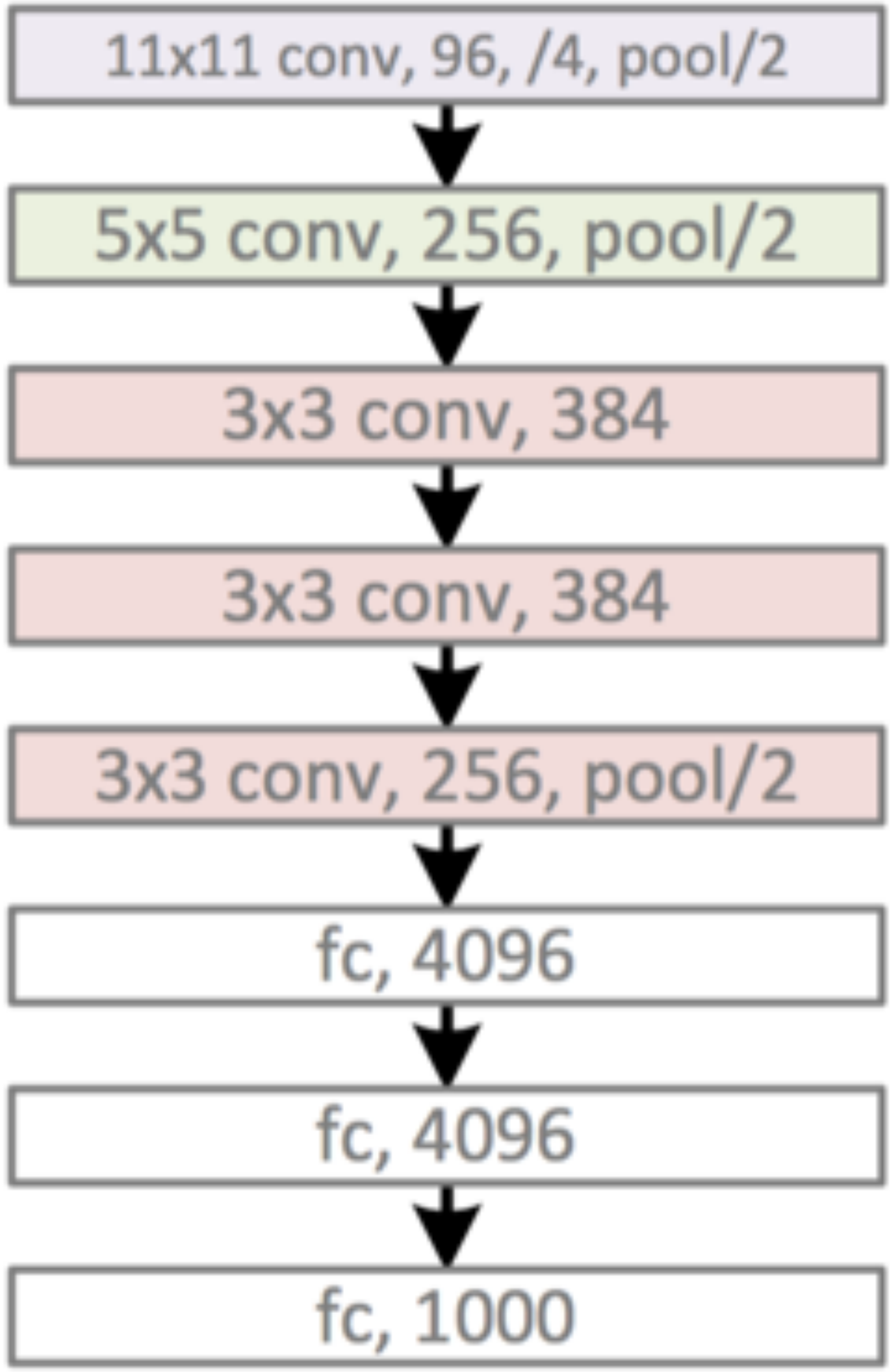
2012: AlexNet
5 conv. layers



Error: 16.4%

[Krizhevsky et al: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012]

Alexnet — [Krizhevsky et al. NIPS 2012]



[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

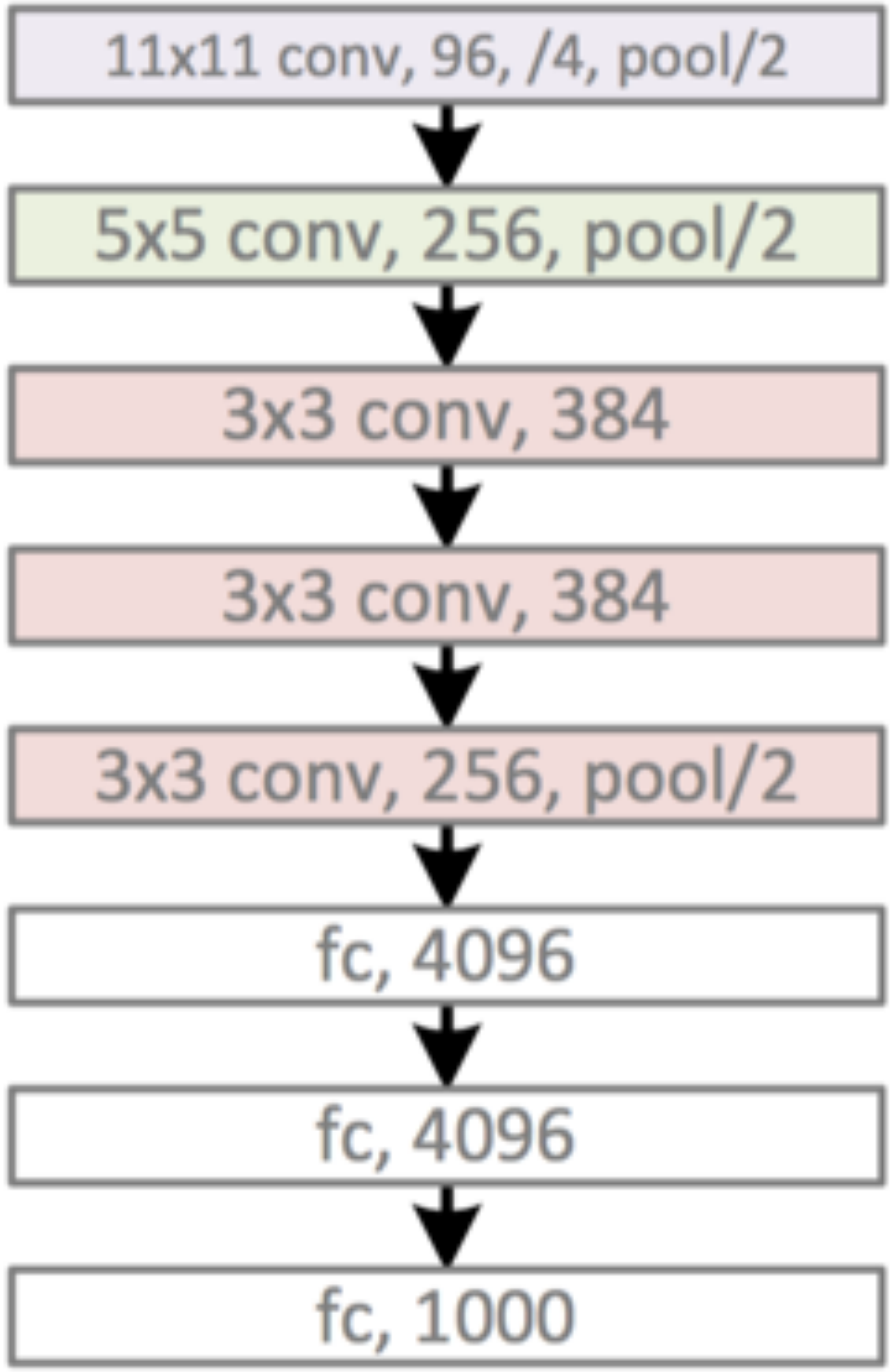
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

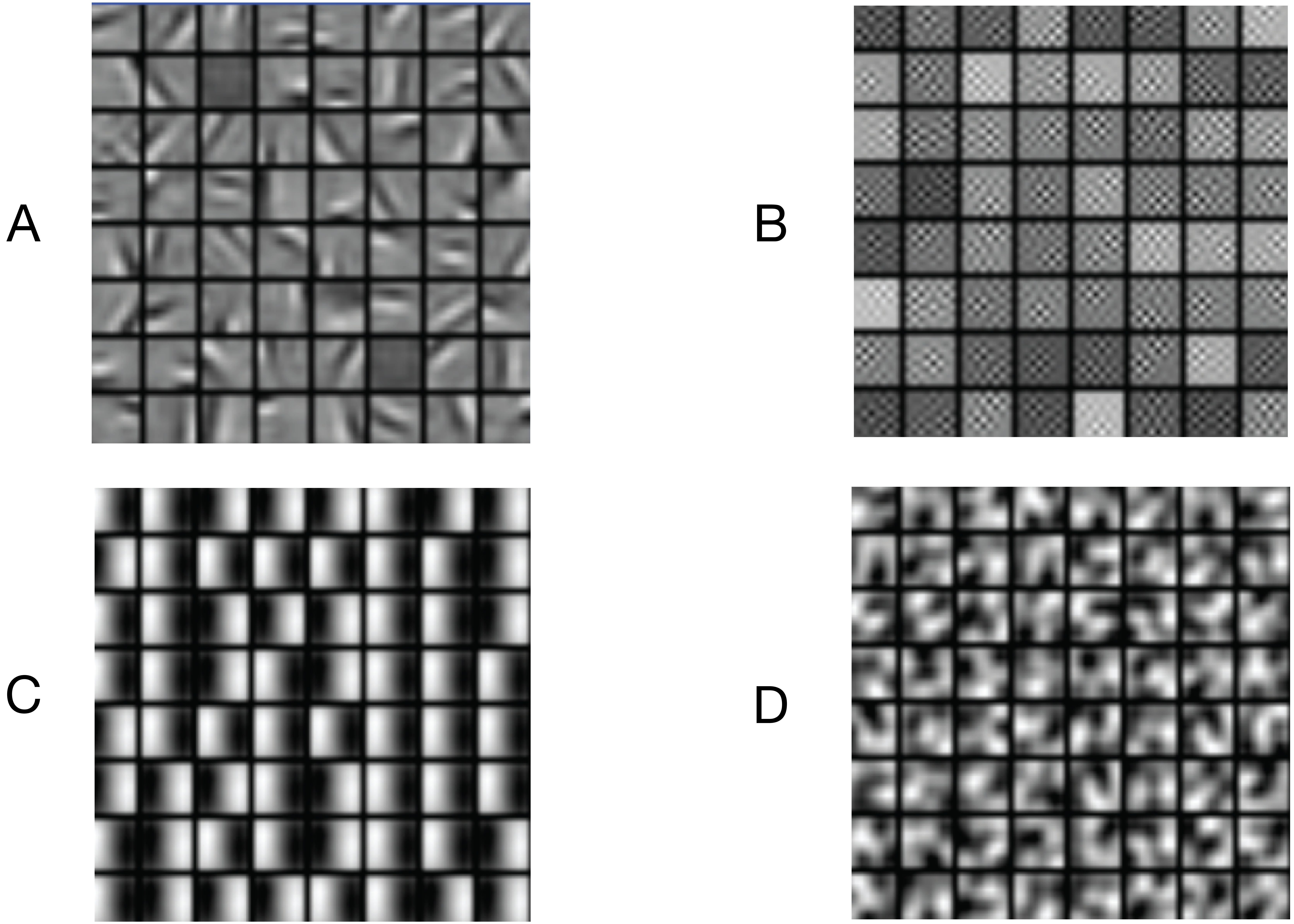
[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)



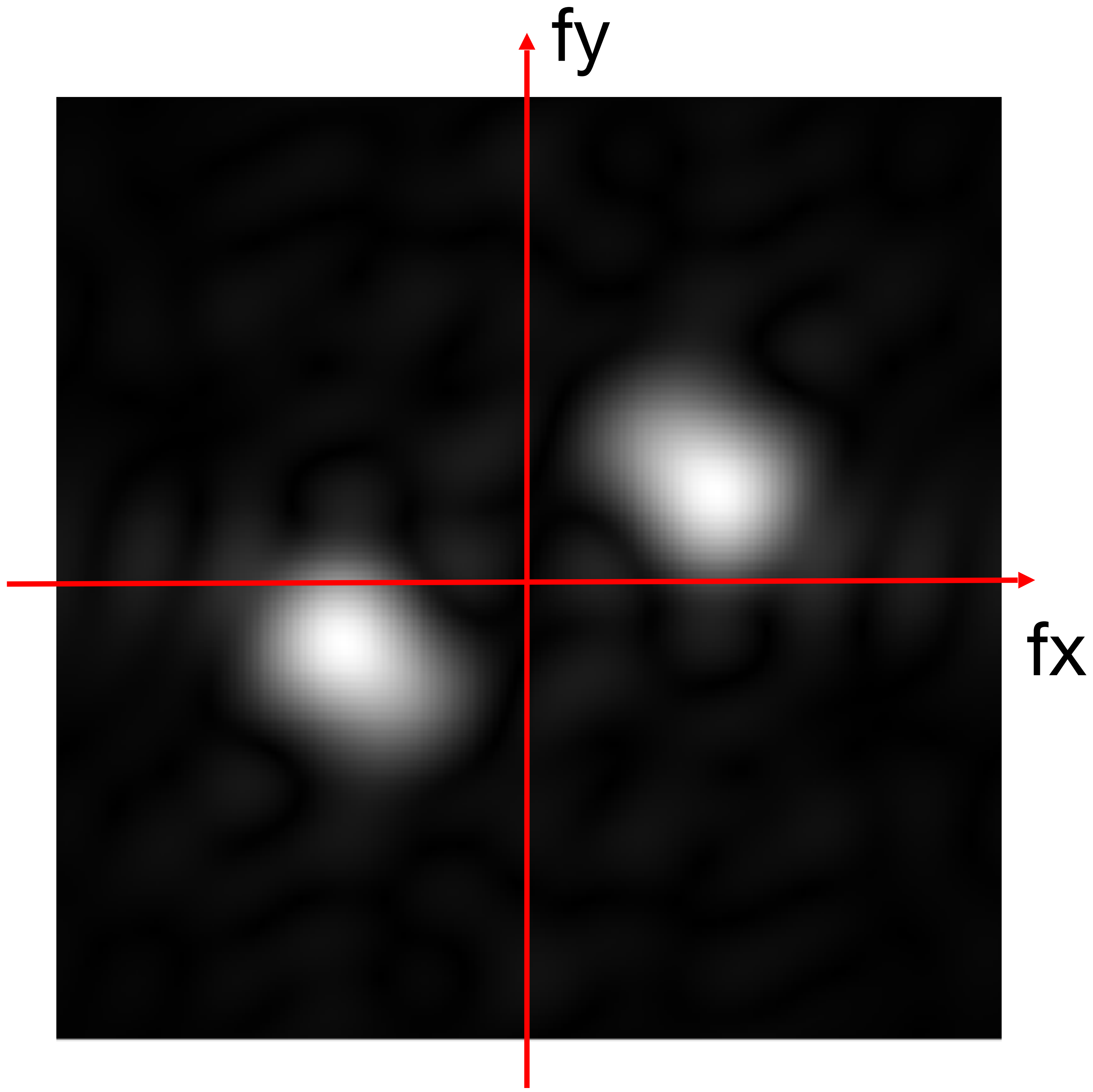
What filters are learned?

What filters are learned?

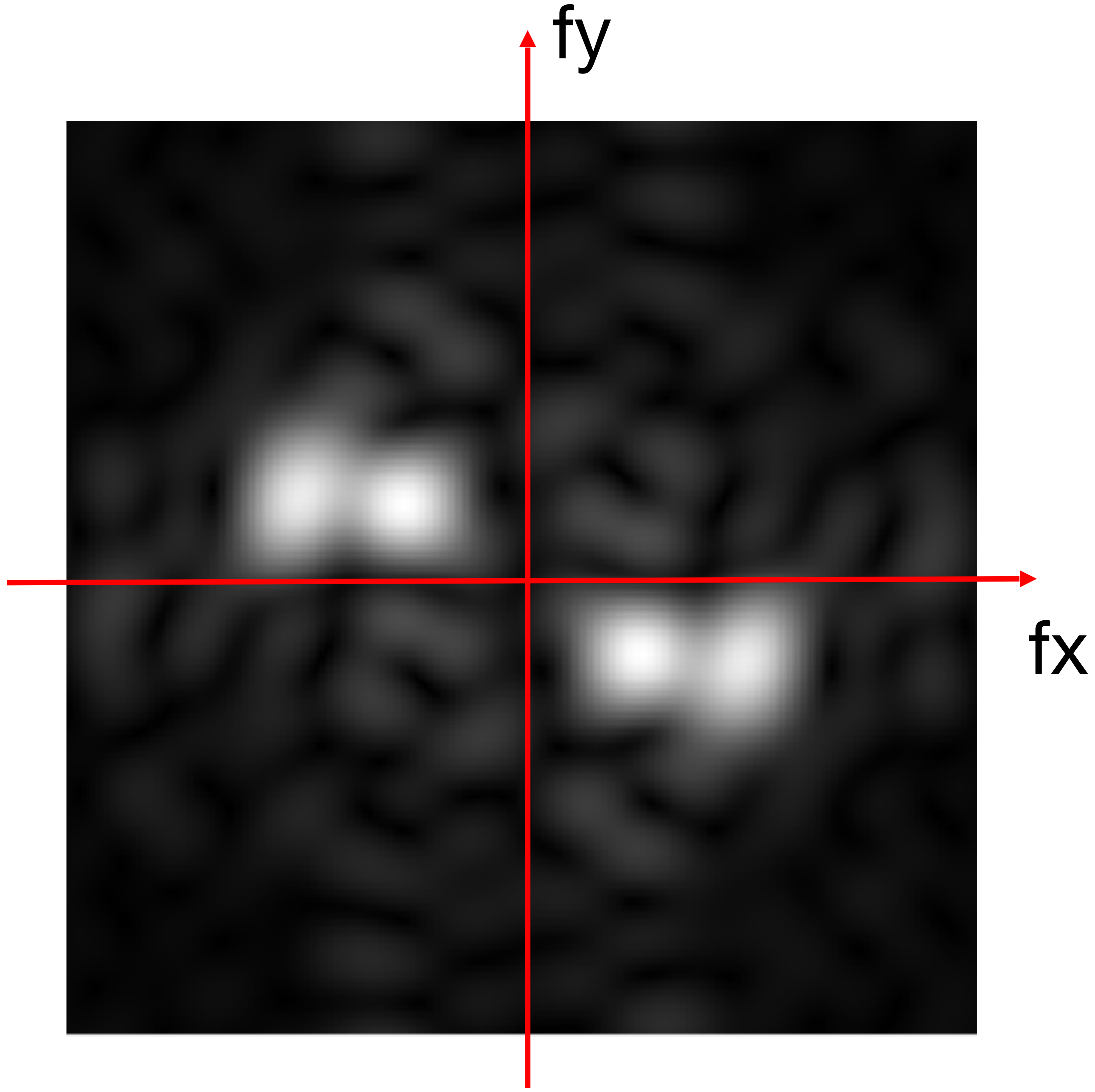


Get to know your units

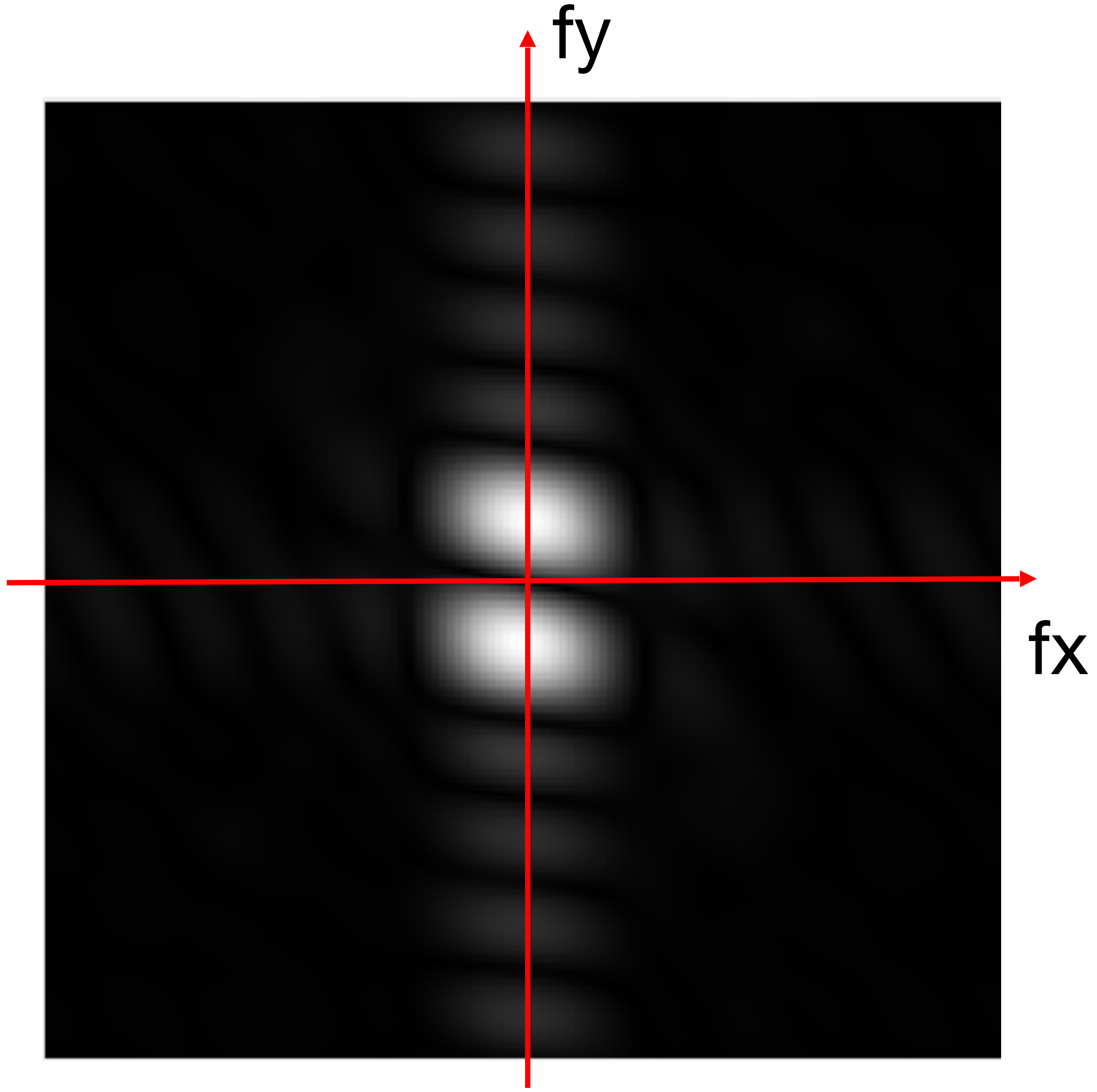
11x11 convolution kernel
(3 color channels)



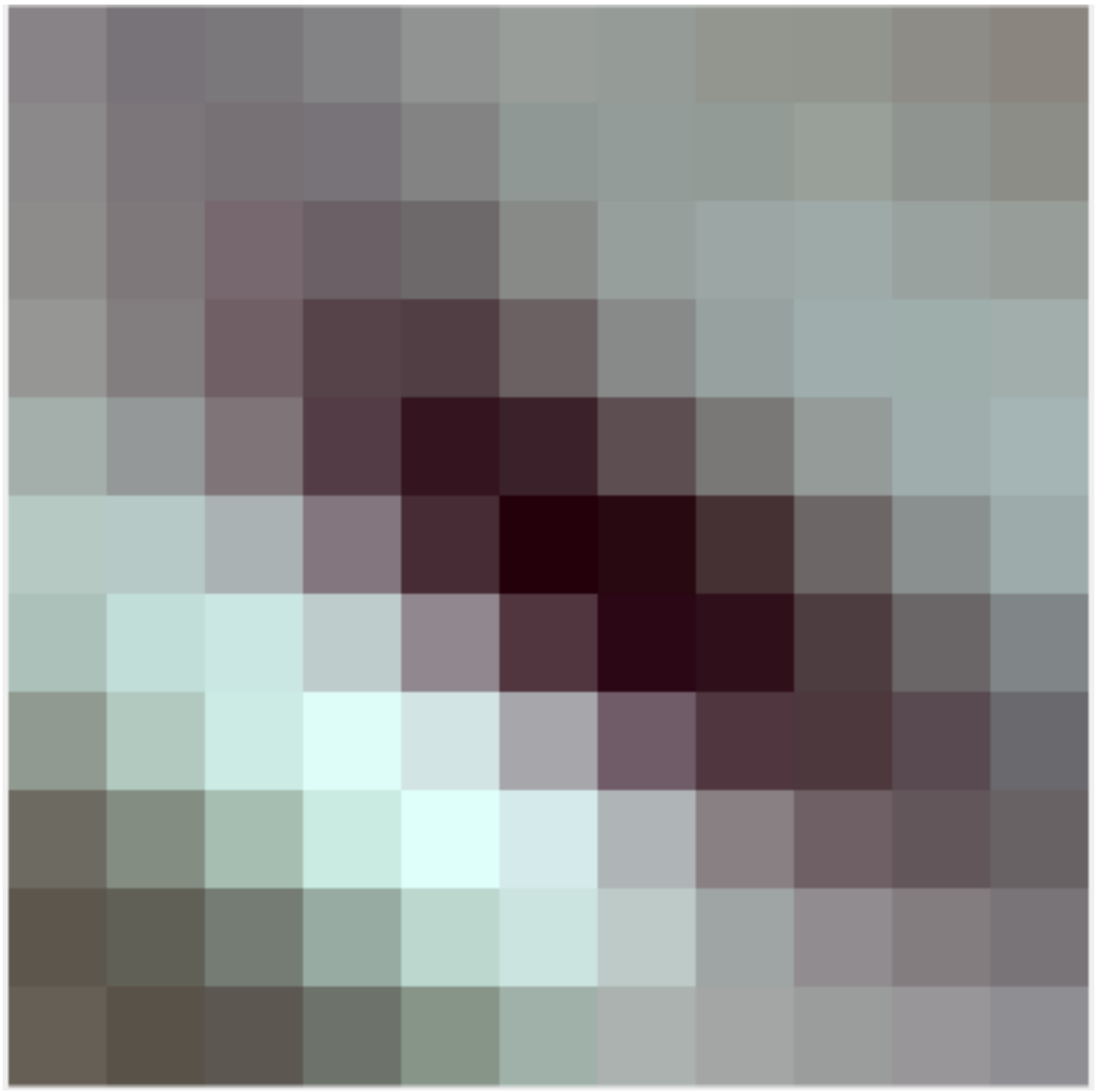
Get to know your units

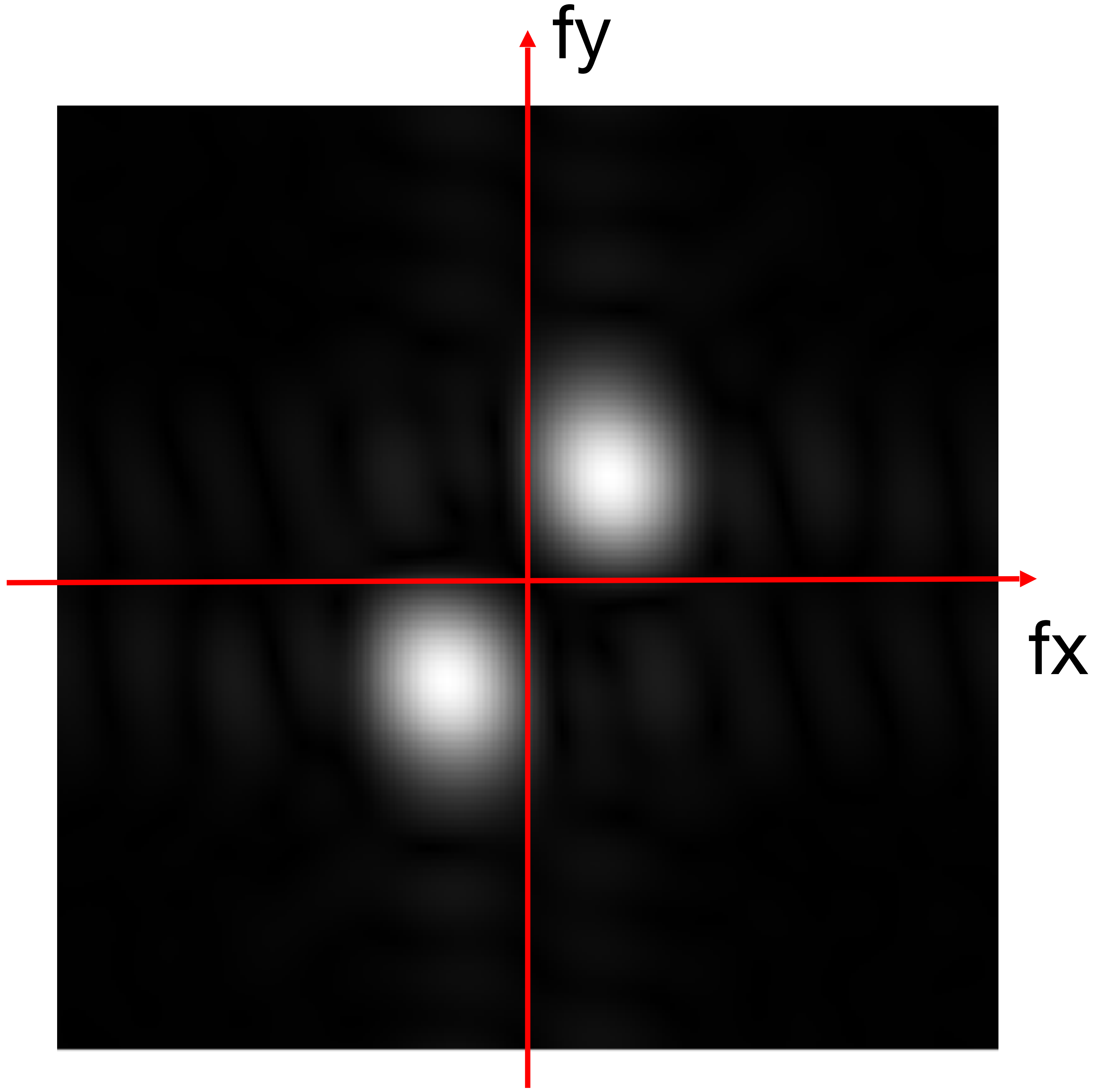


Get to know your units

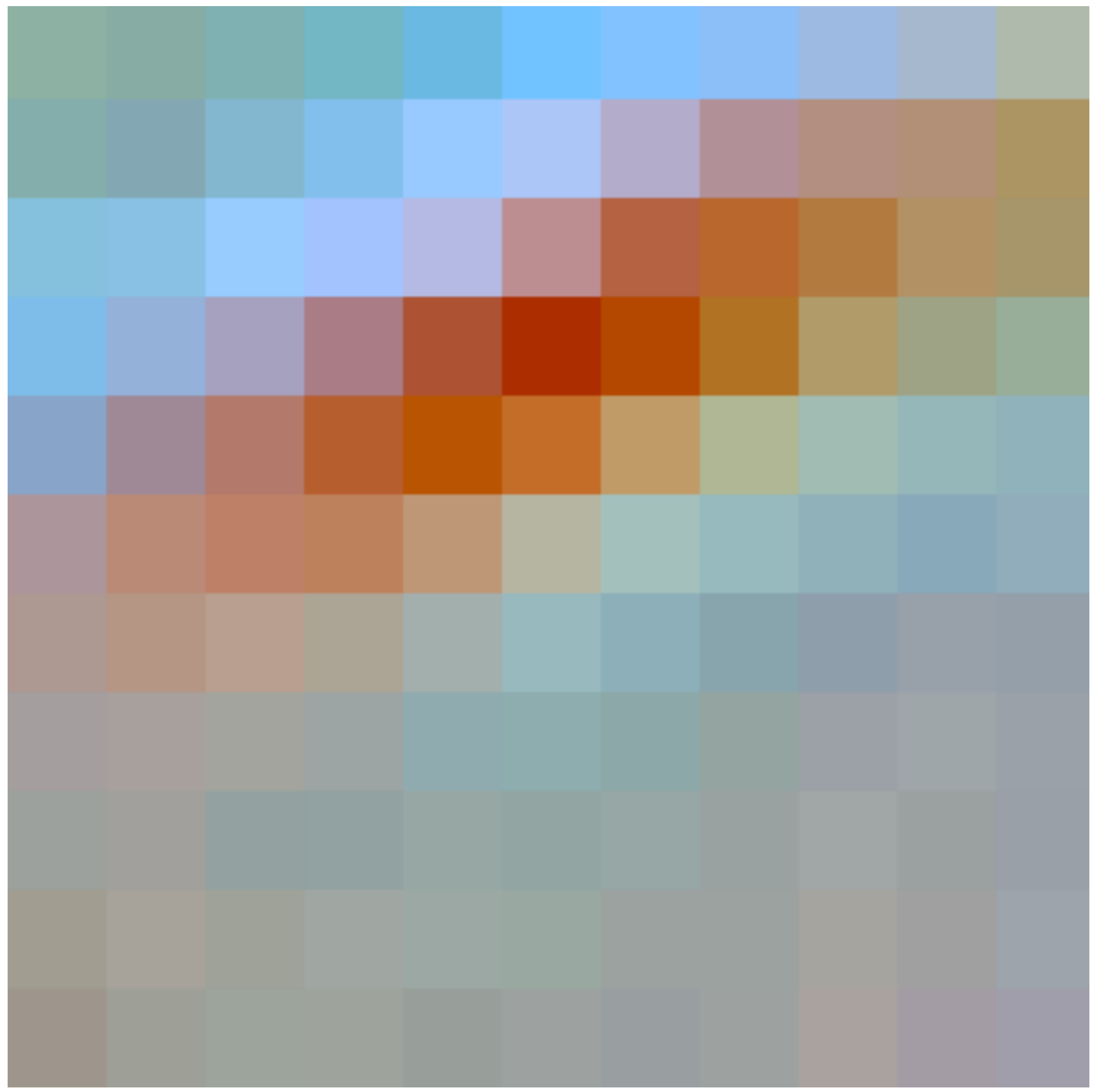


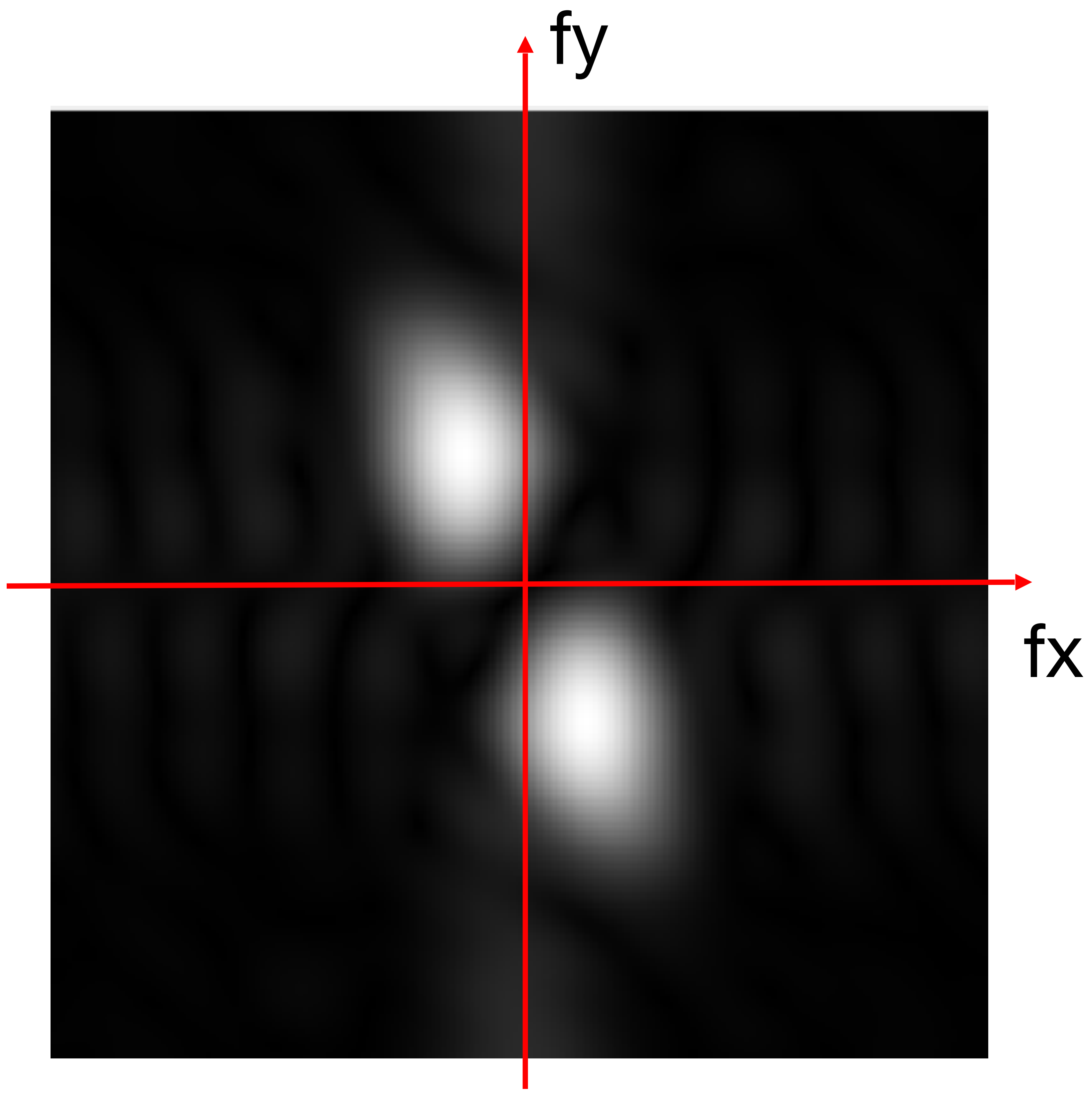
Get to know your units



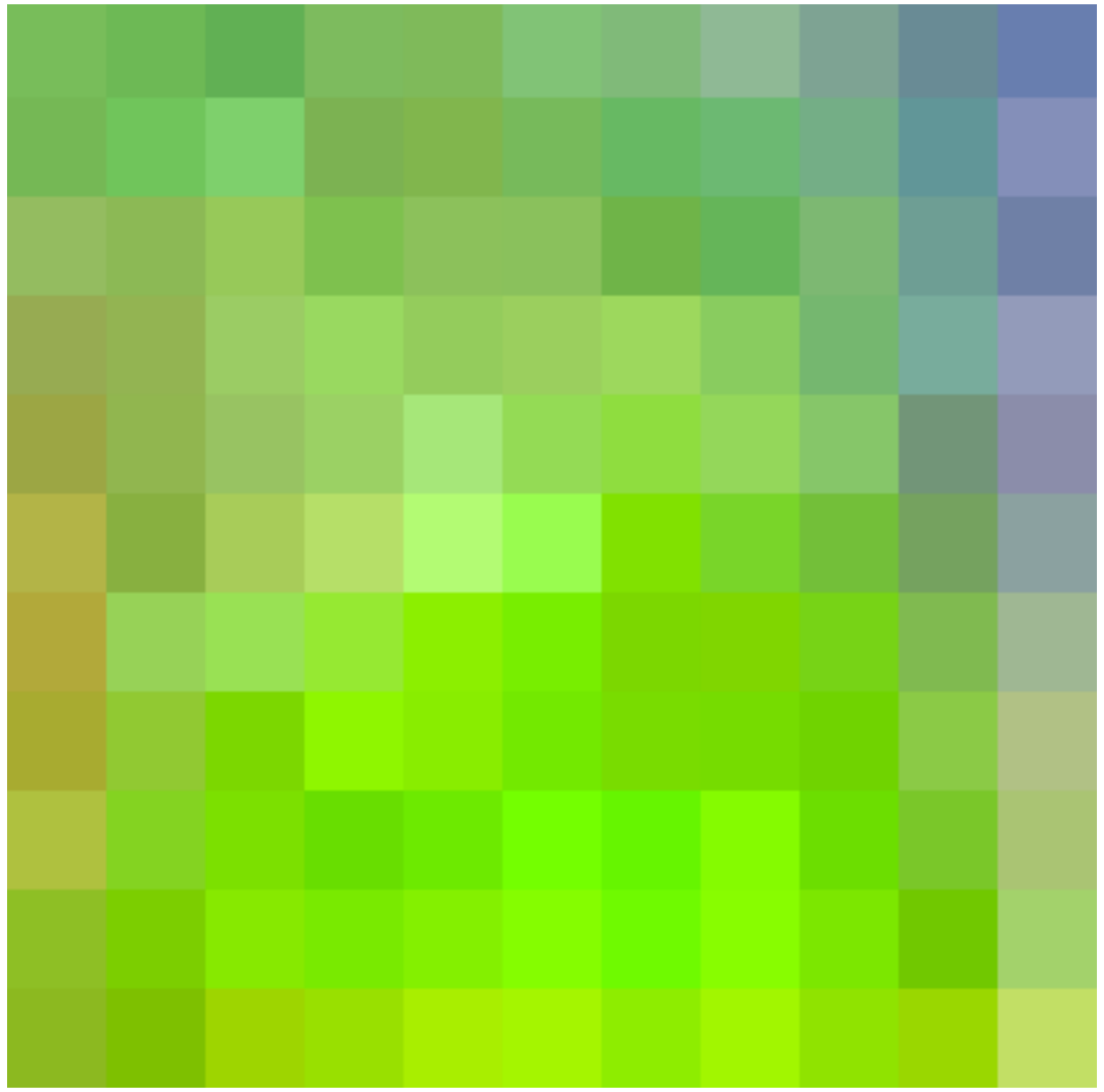


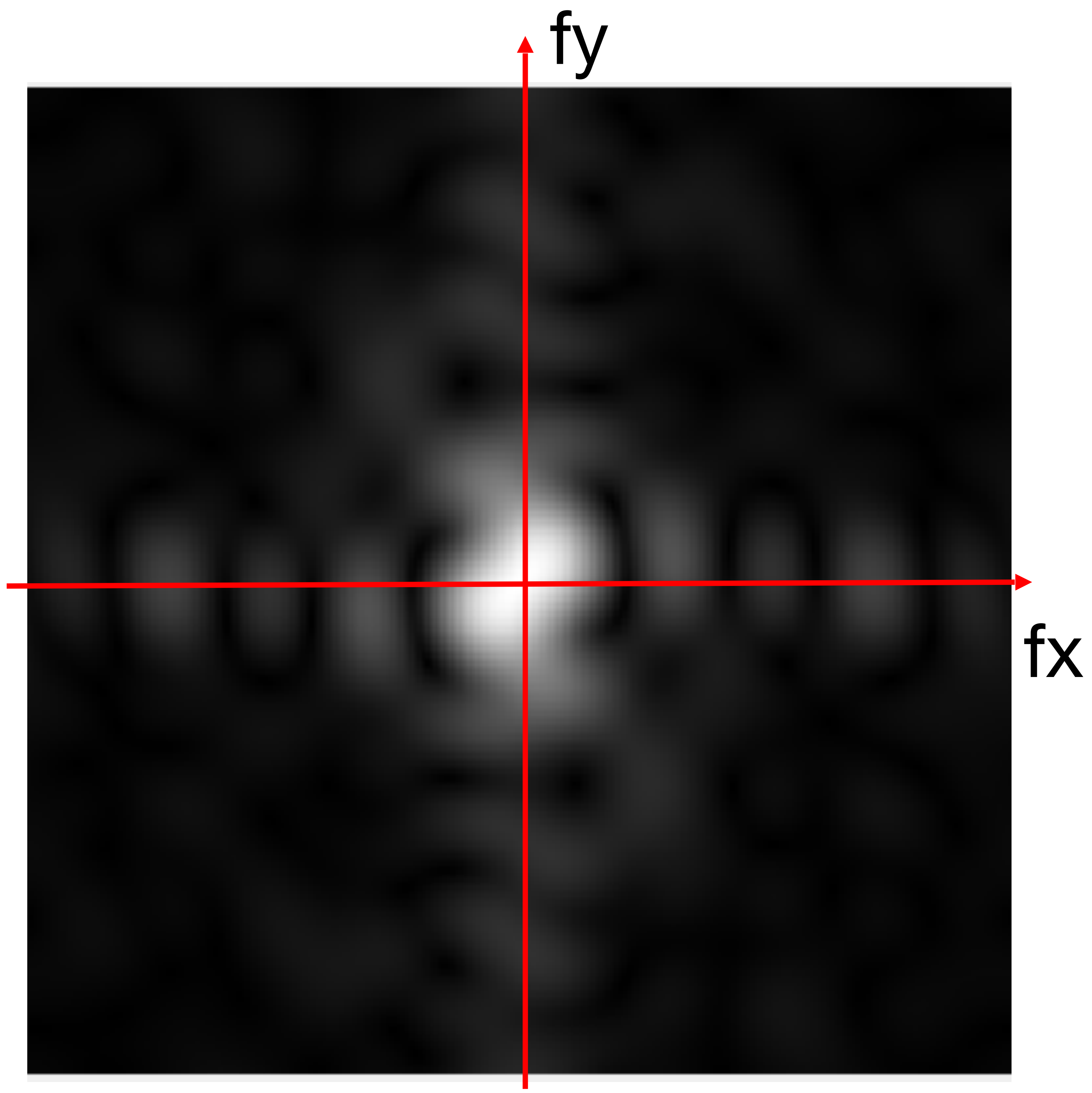
Get to know your units



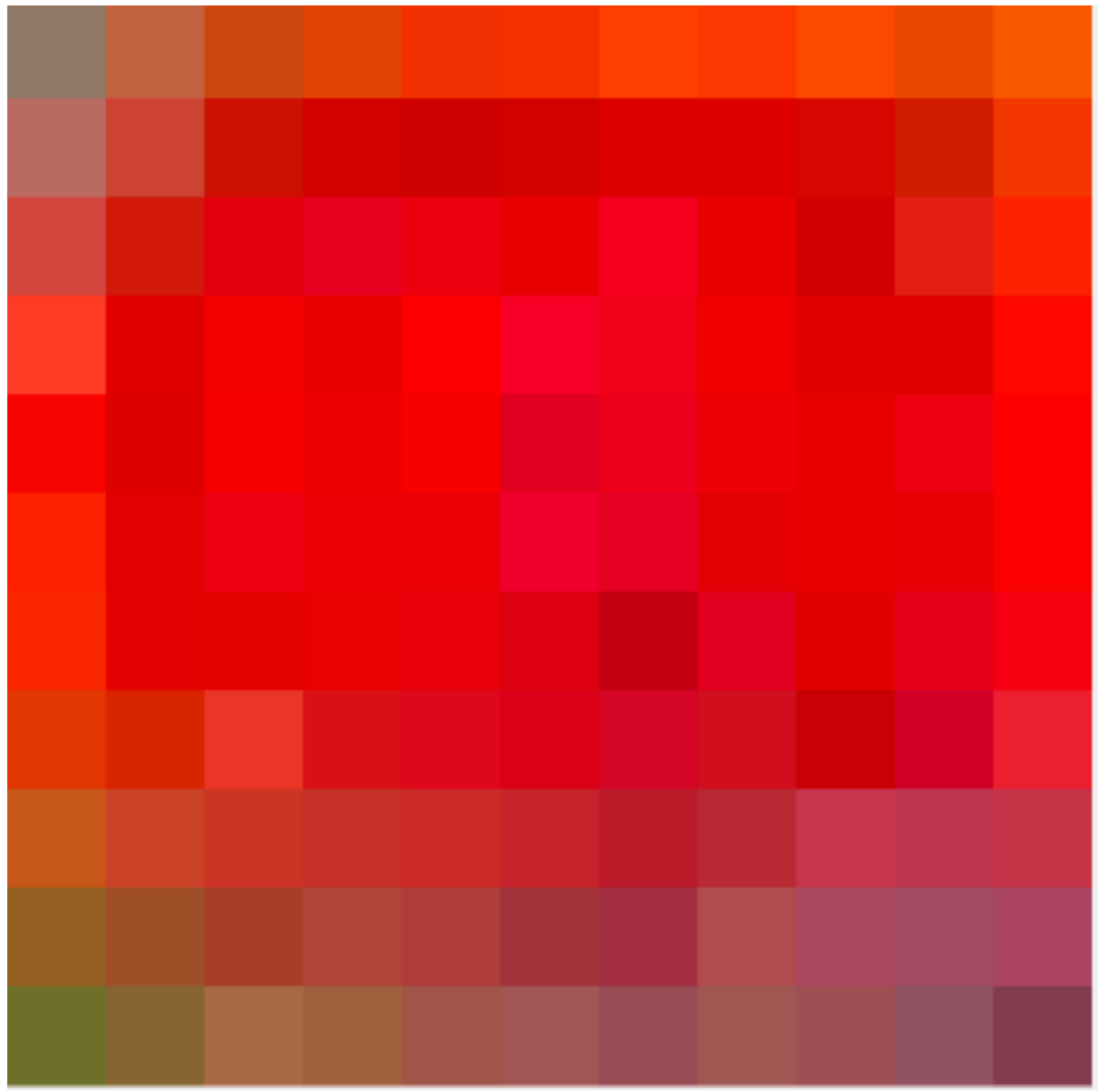


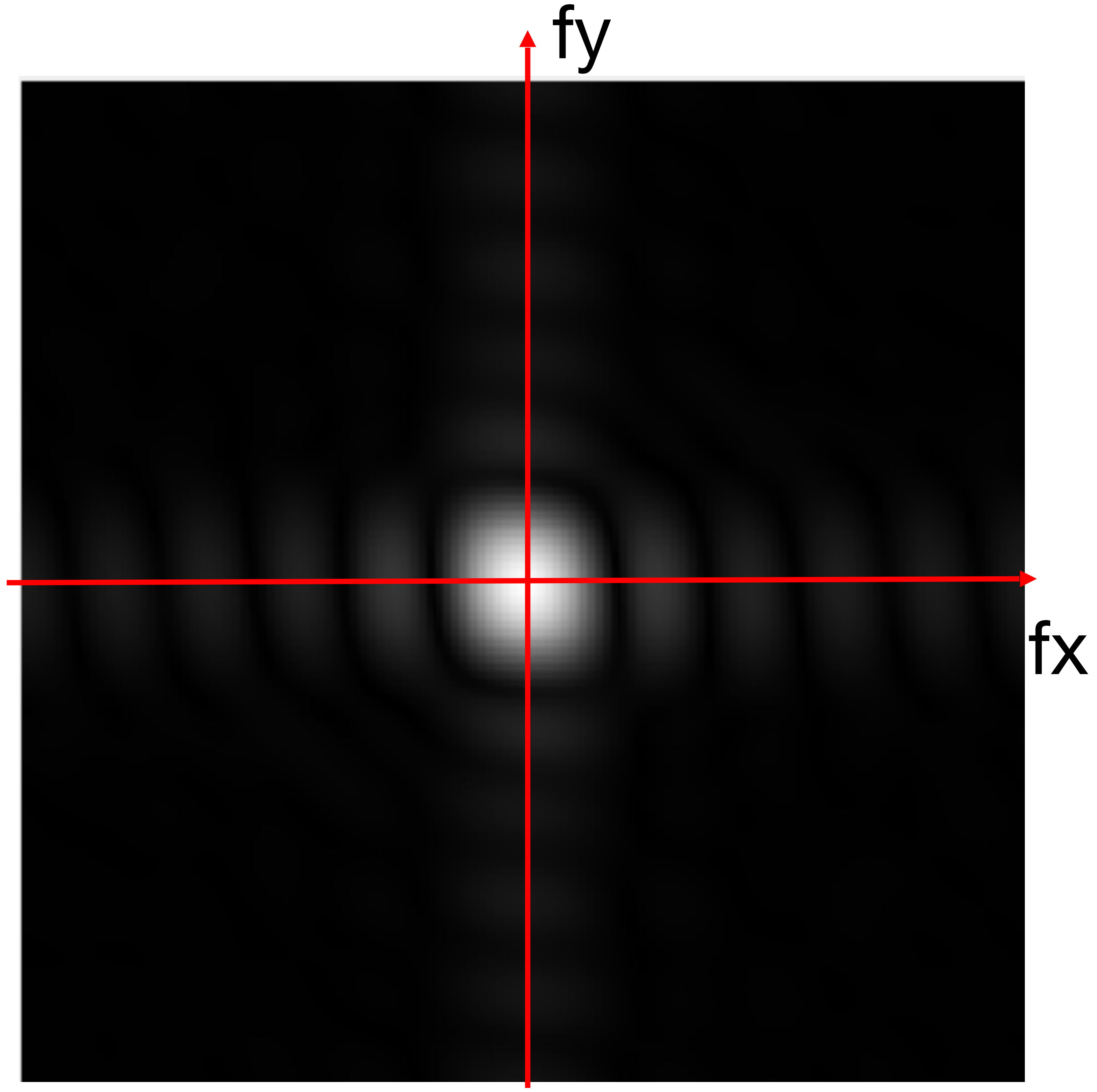
Get to know your units



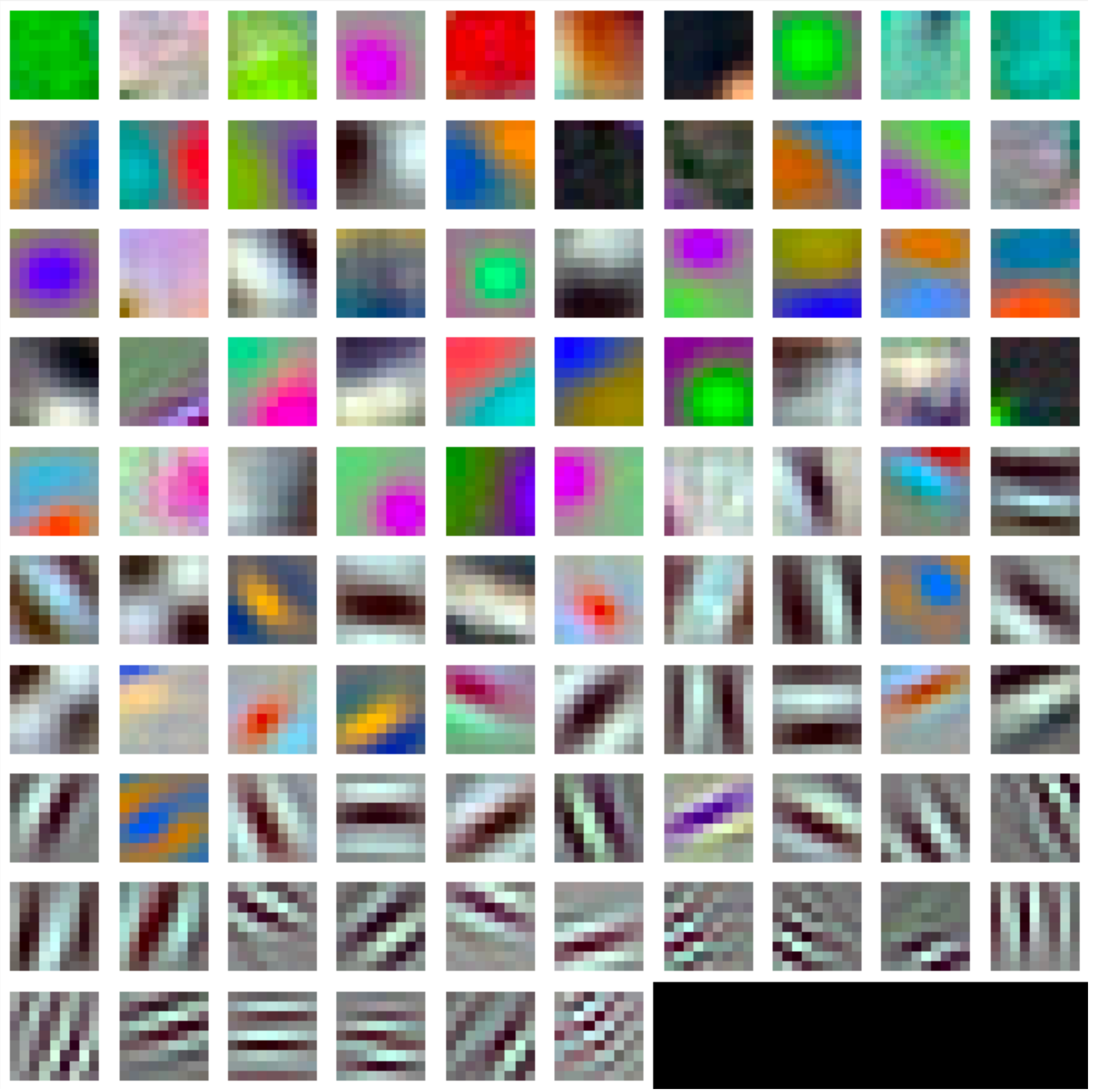


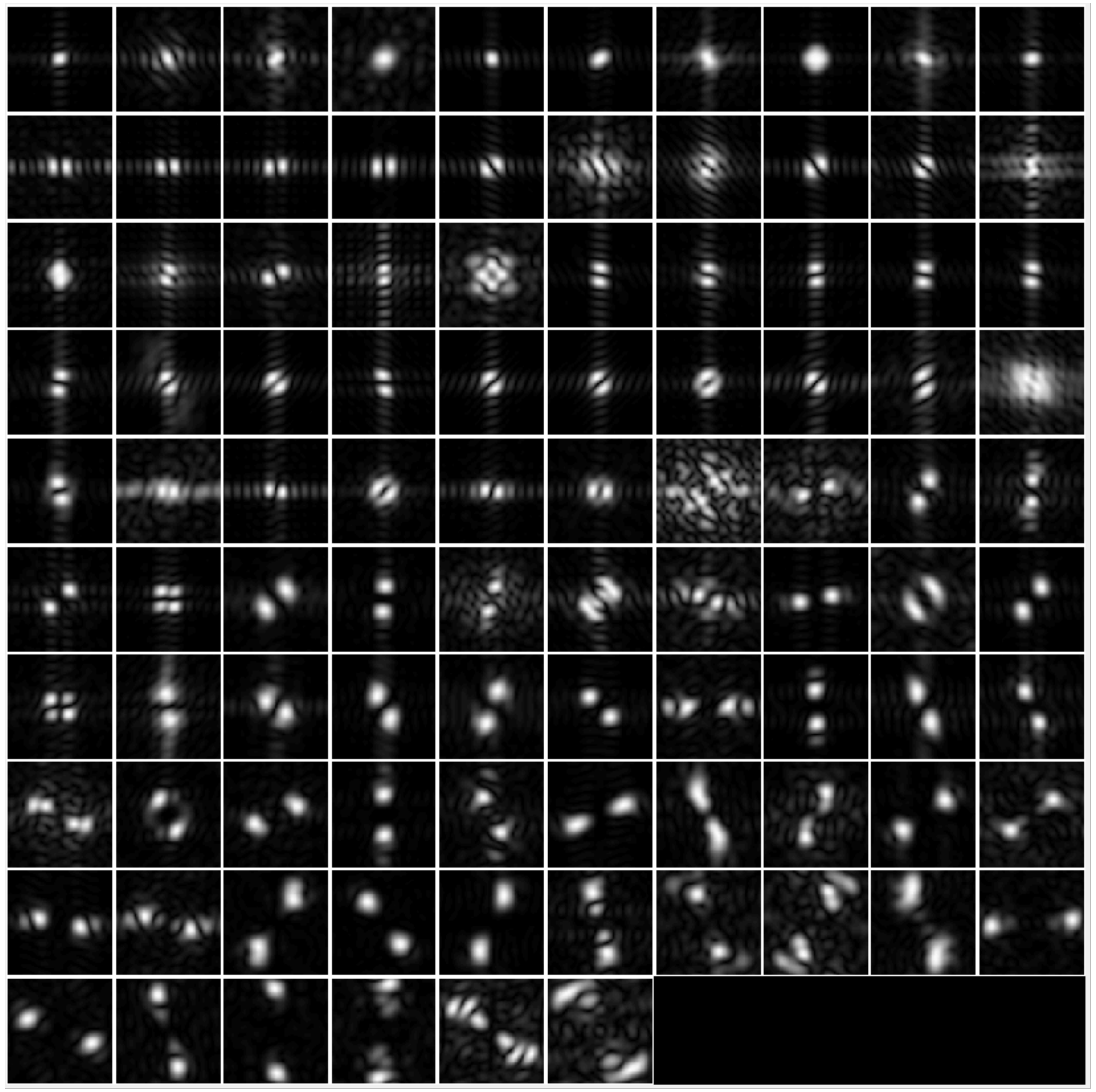
Get to know your units





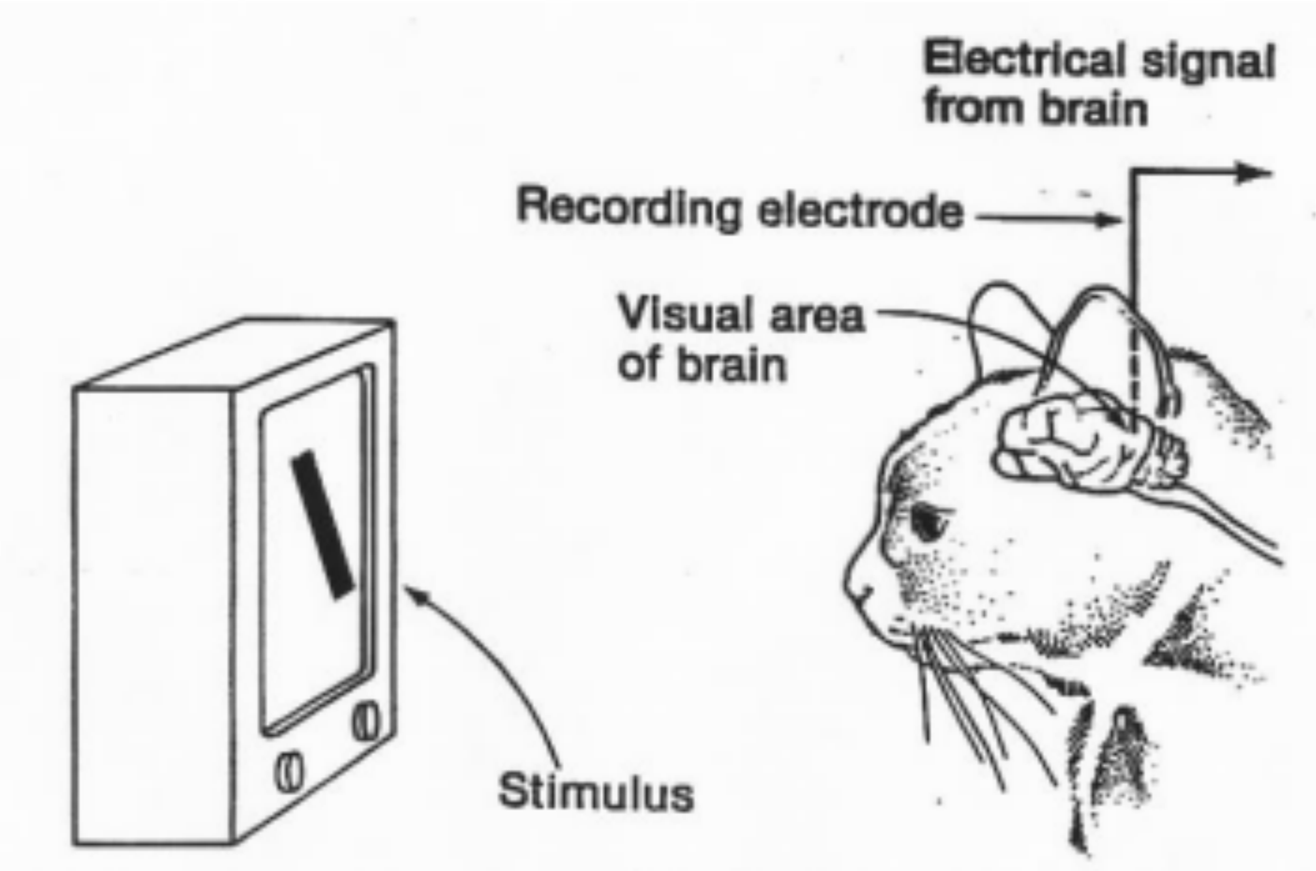
Get to know your units

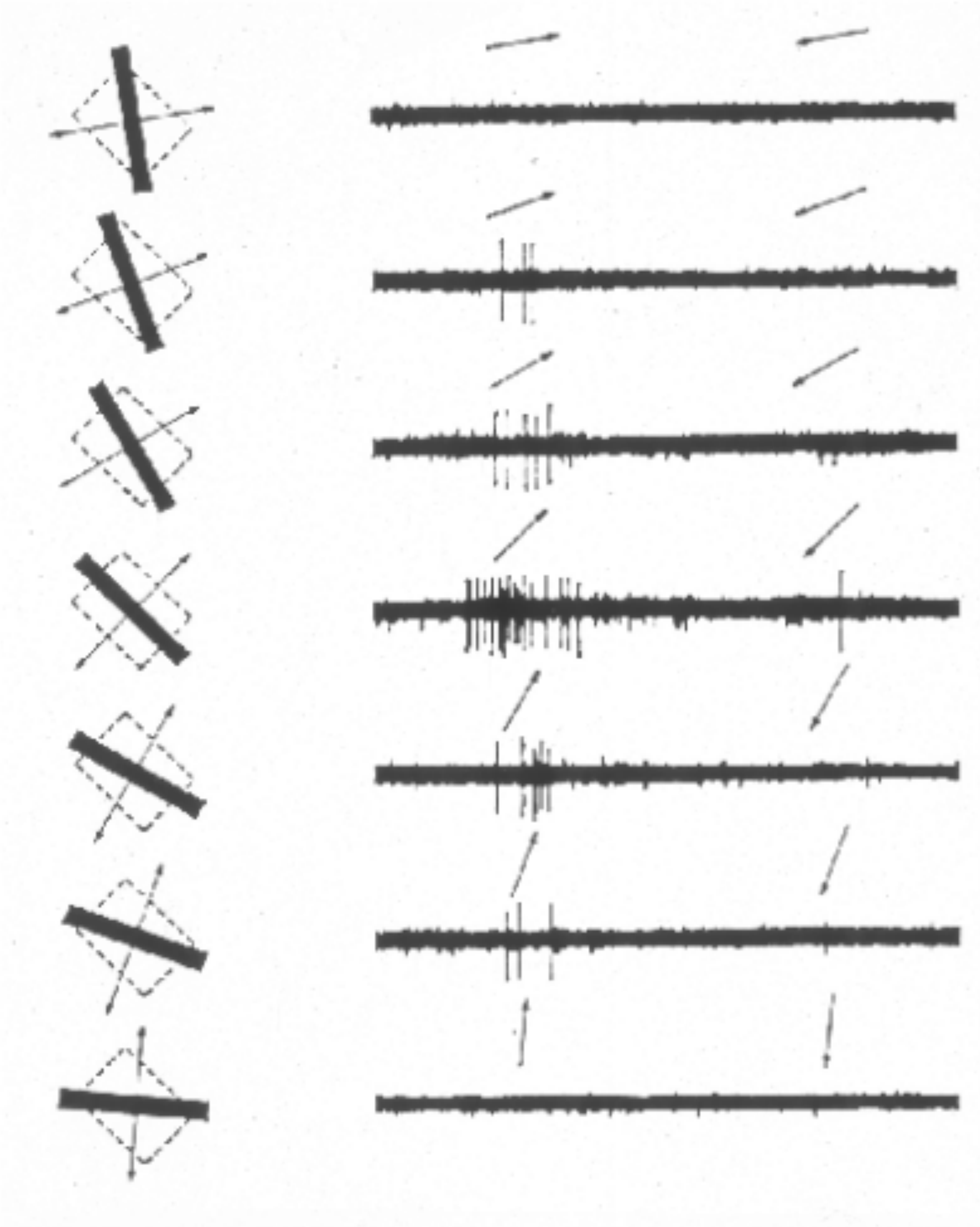


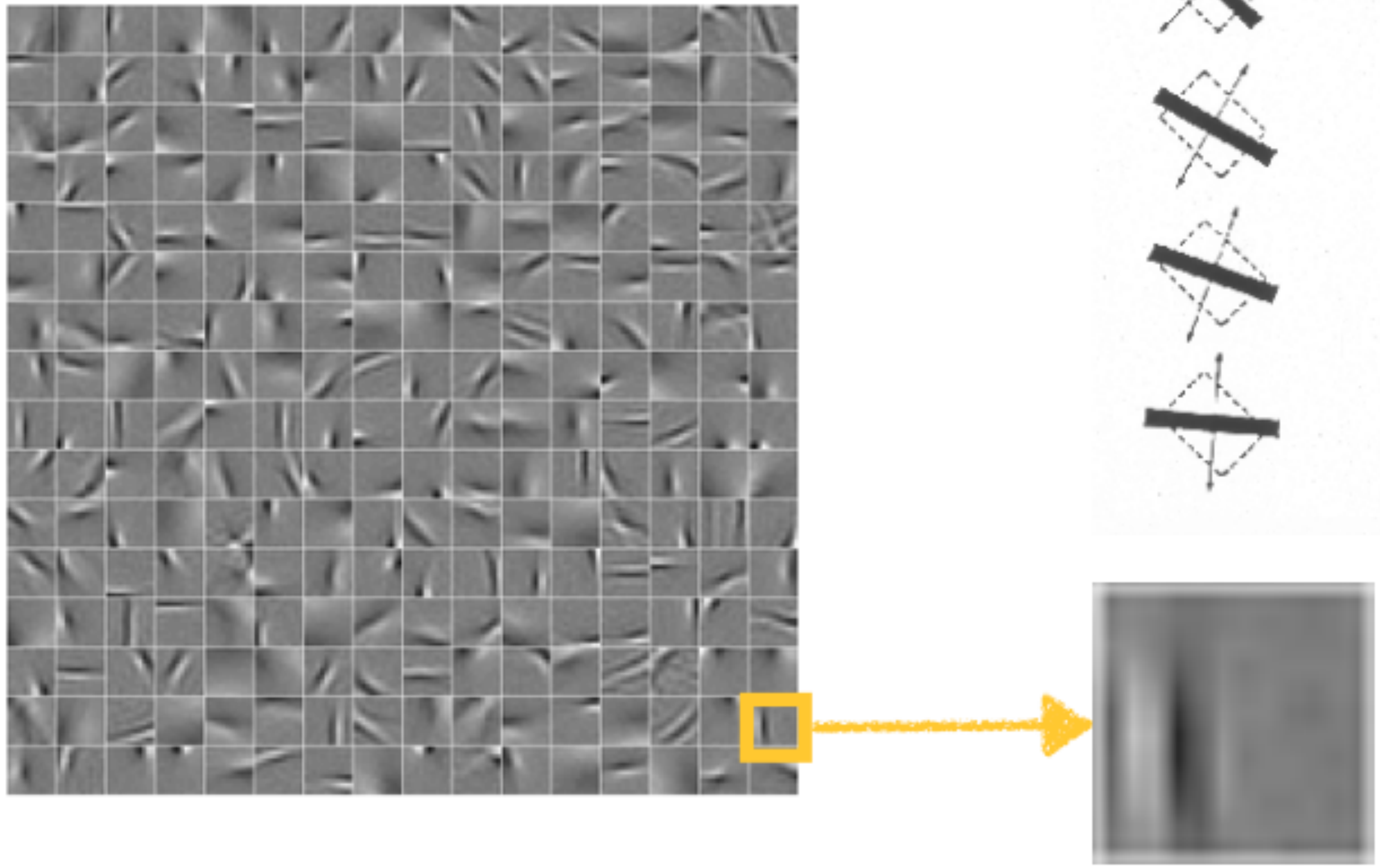


96 Units in conv1

[Hubel and Wiesel 59]



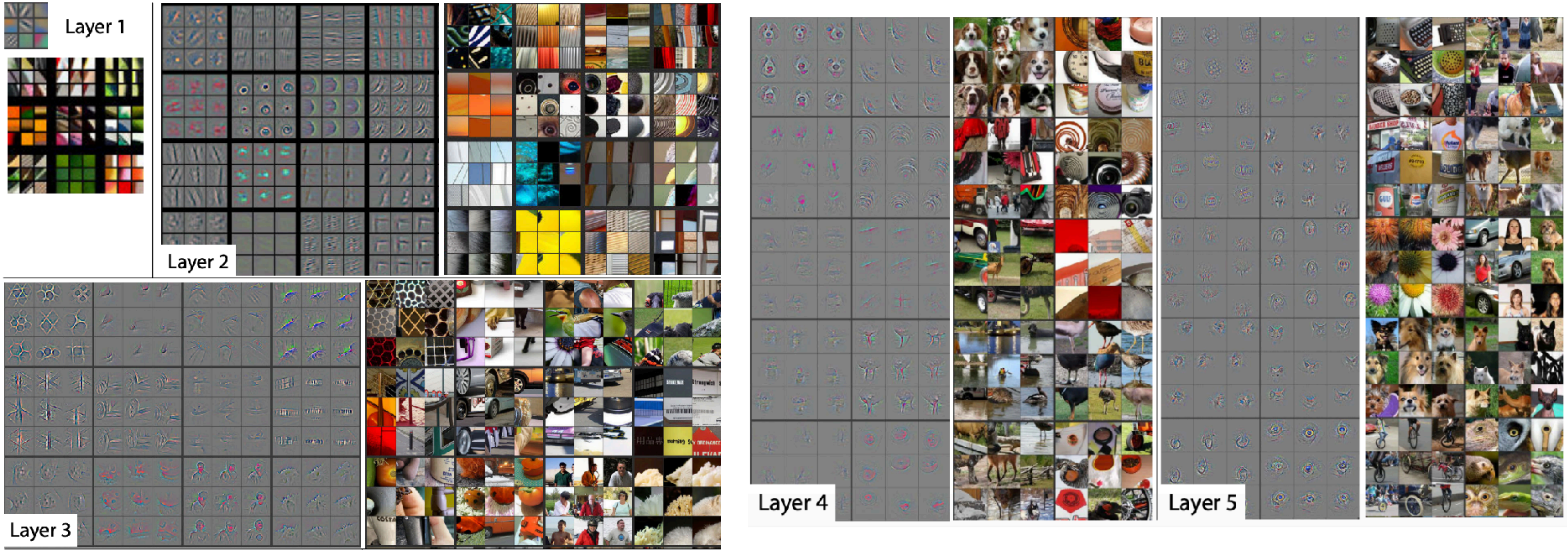




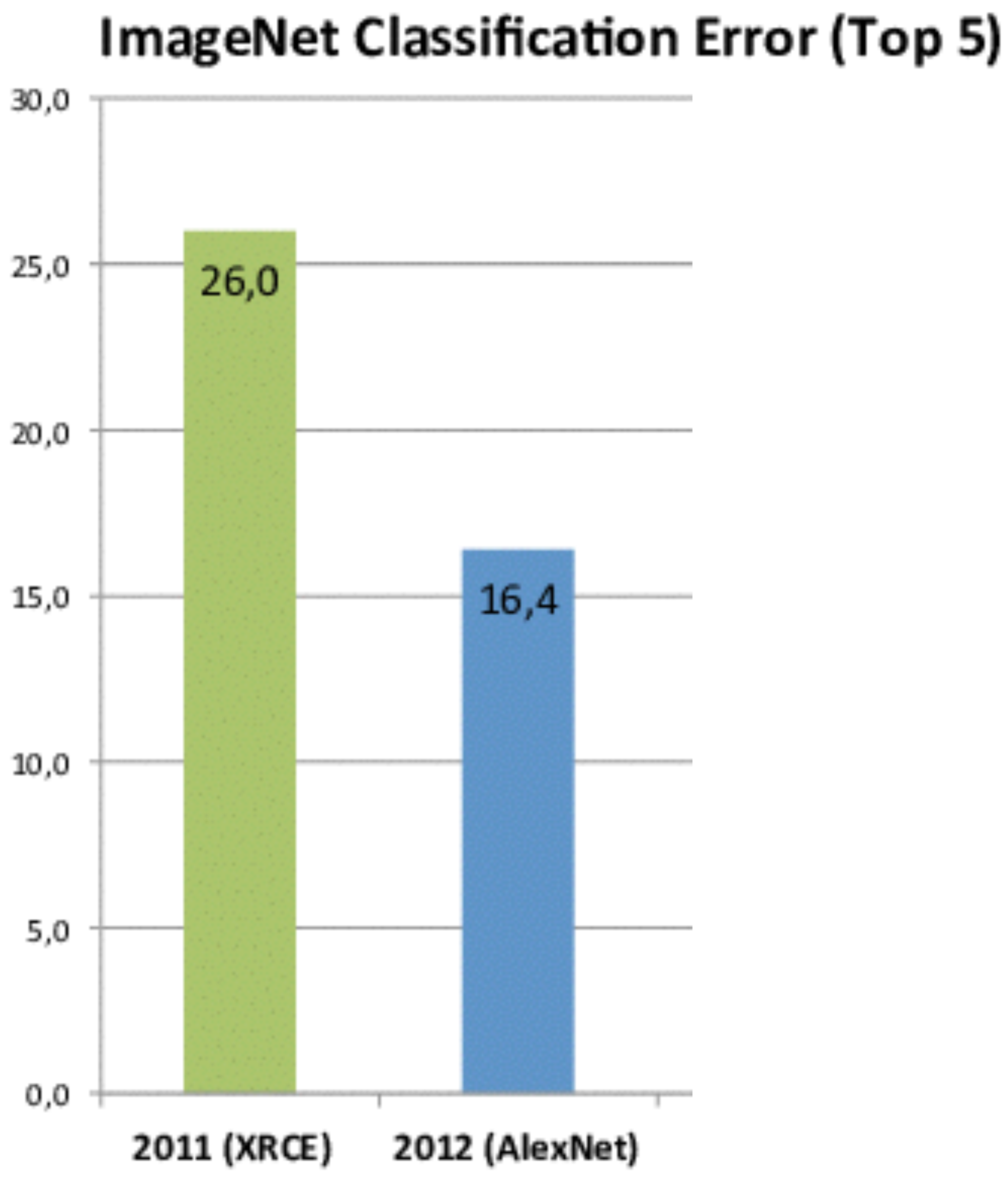
oriented filter

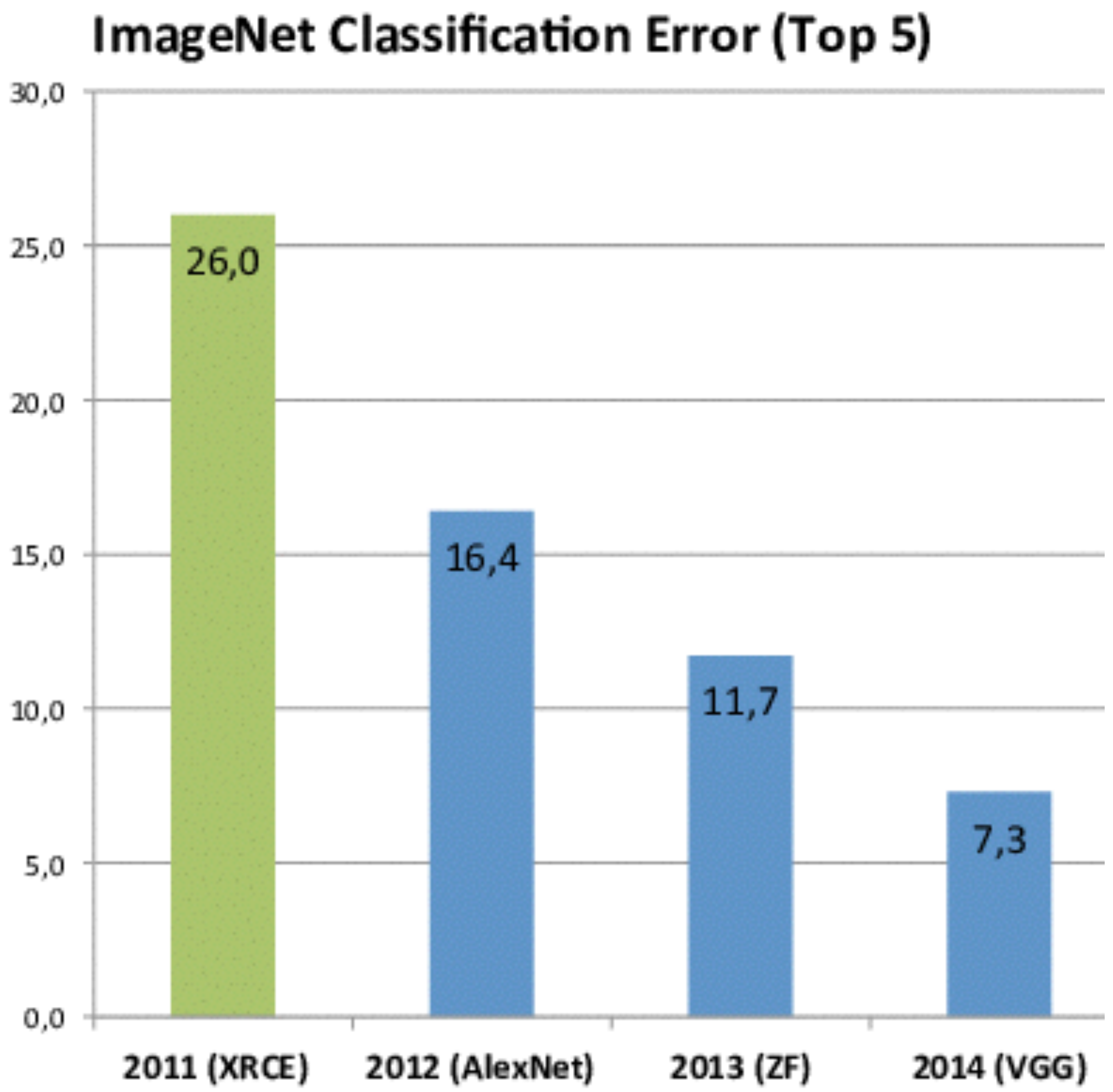
[Slide from Andrea Vedaldi]

Units at Higher Layers

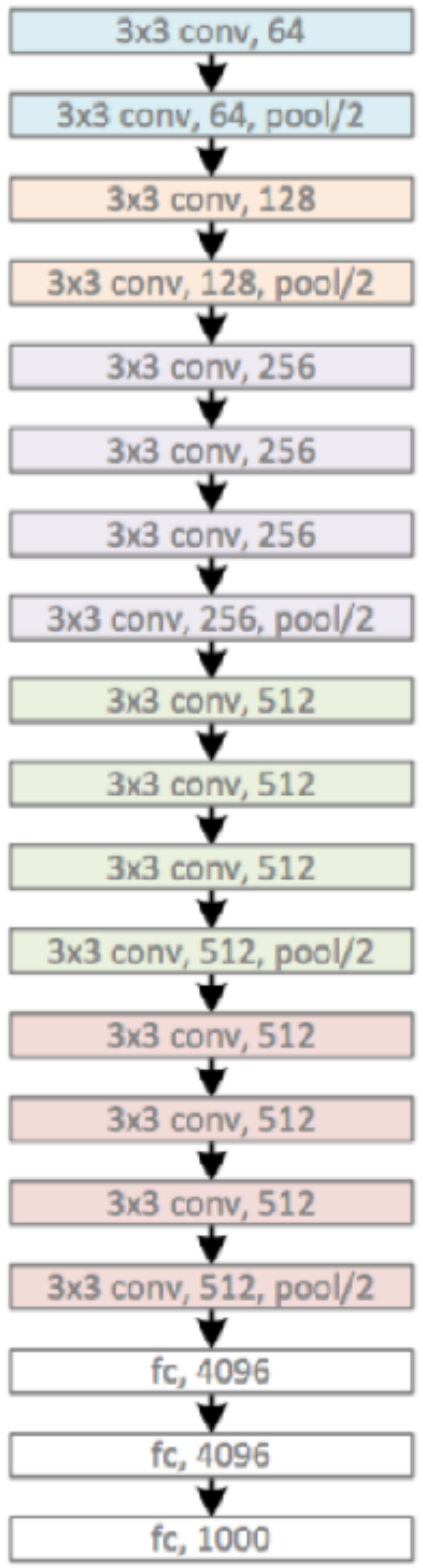


Zeiler and Fergus, ECCV'14


ImageNet Classification Error (Top 5)
30,0
25,0
20,0
15,0
10,0
5,0
0,0
26,0
16,4
2011 (XRCE)
2012 (AlexNet)



2014: VGG
16 conv. layers

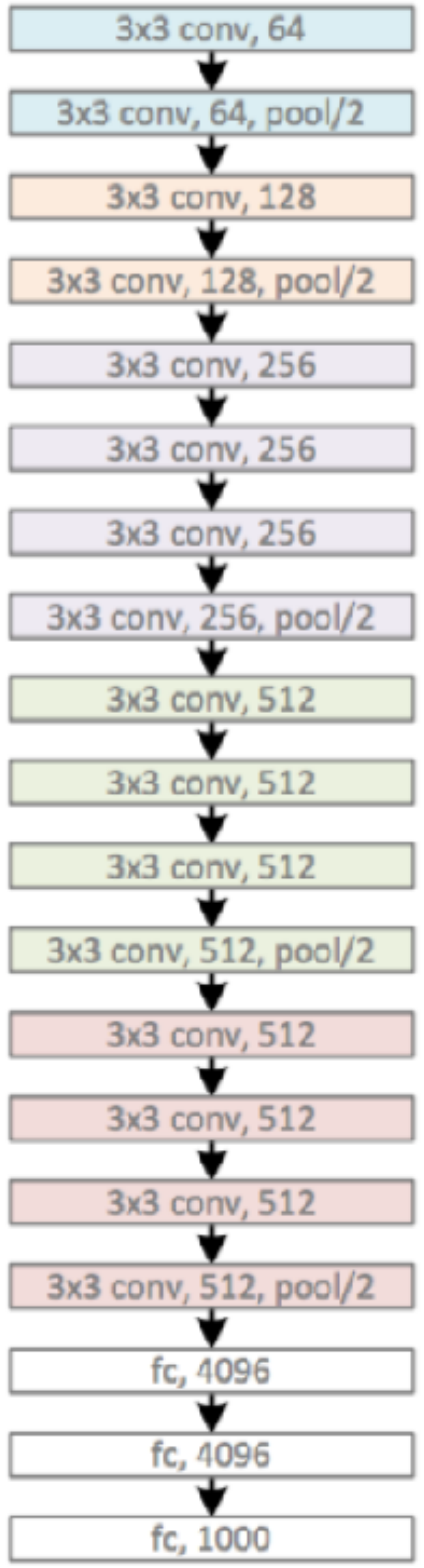


Error: 7.3%

[Simonyan & Zisserman: Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015]

VGG-Net [Simonyan & Zisserman, 2015]

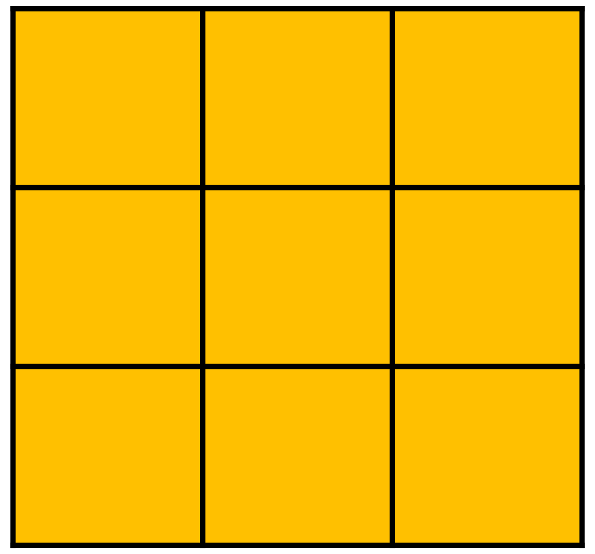
2014: VGG
16 conv. layers



Error: 7.3%

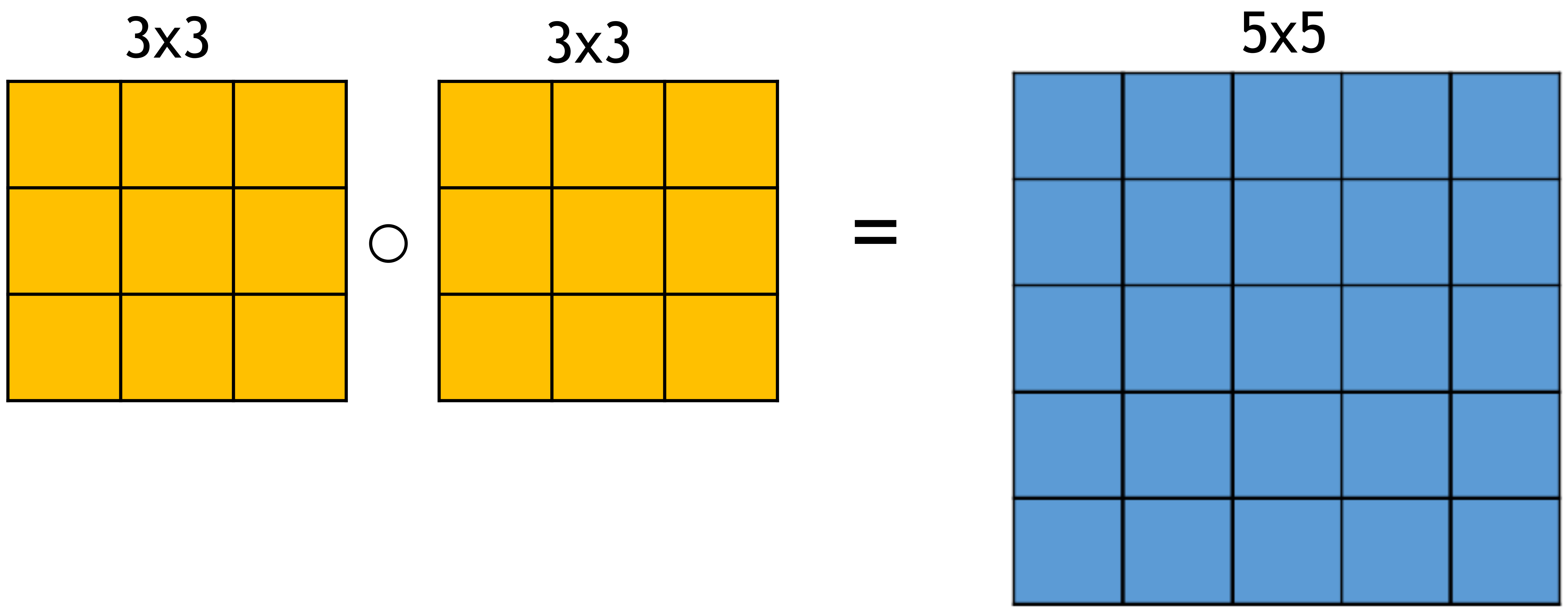
Main developments

- Small convolutional kernels: only 3x3

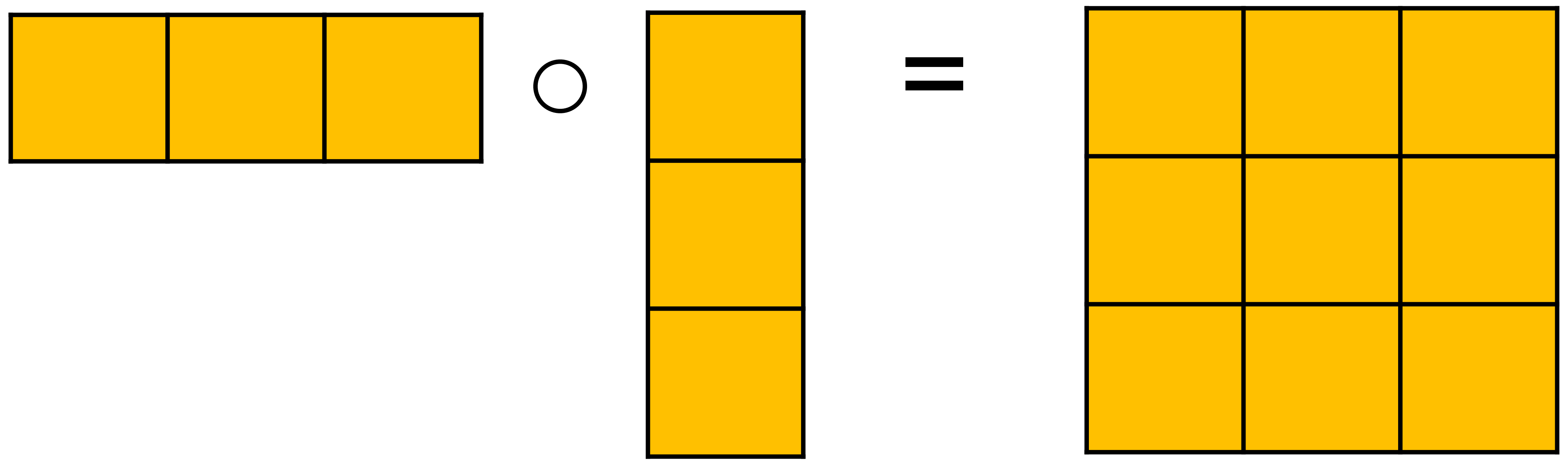


- Increased depth (5 -> 16/19 layers)

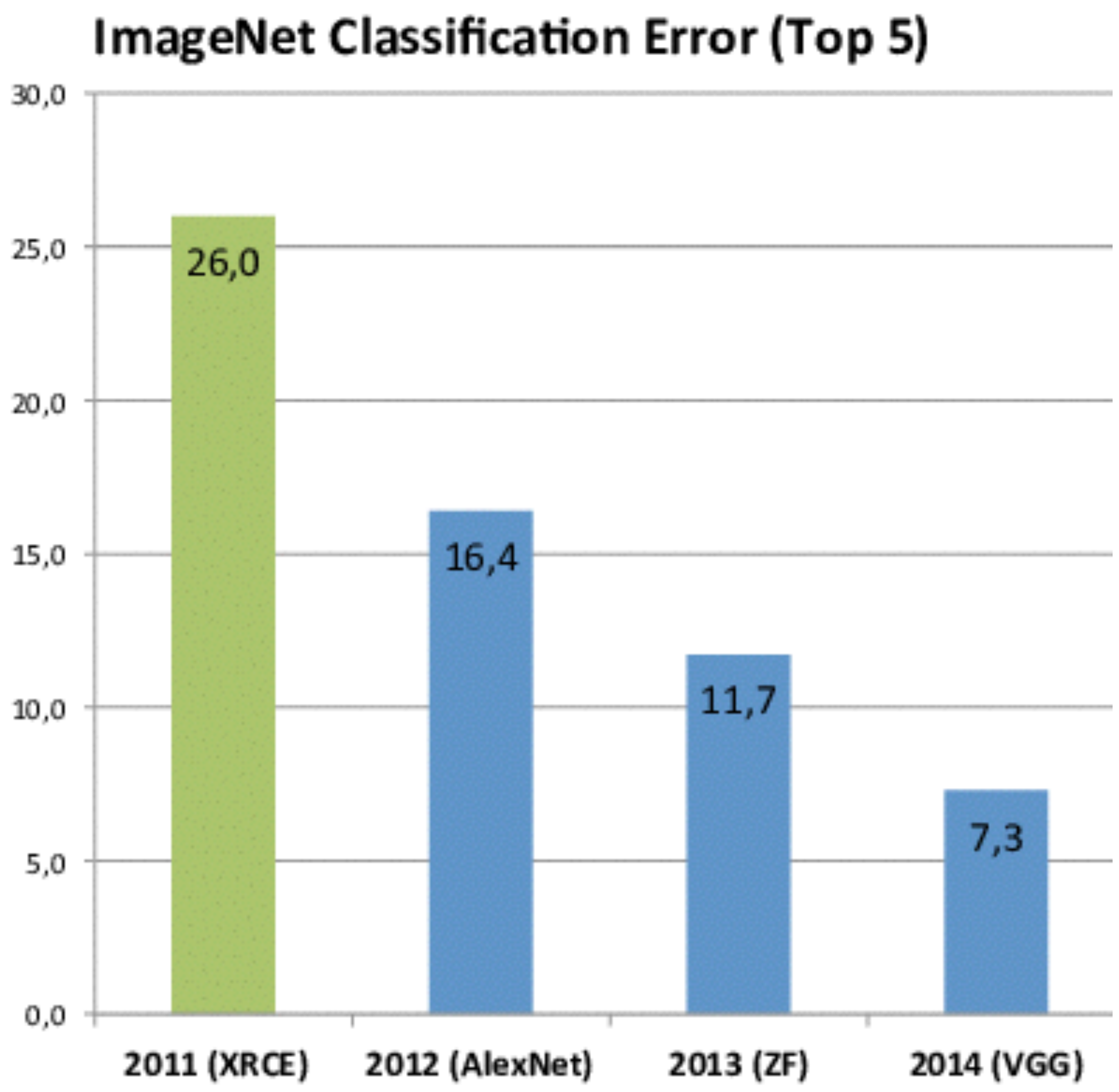
Chaining convolutions

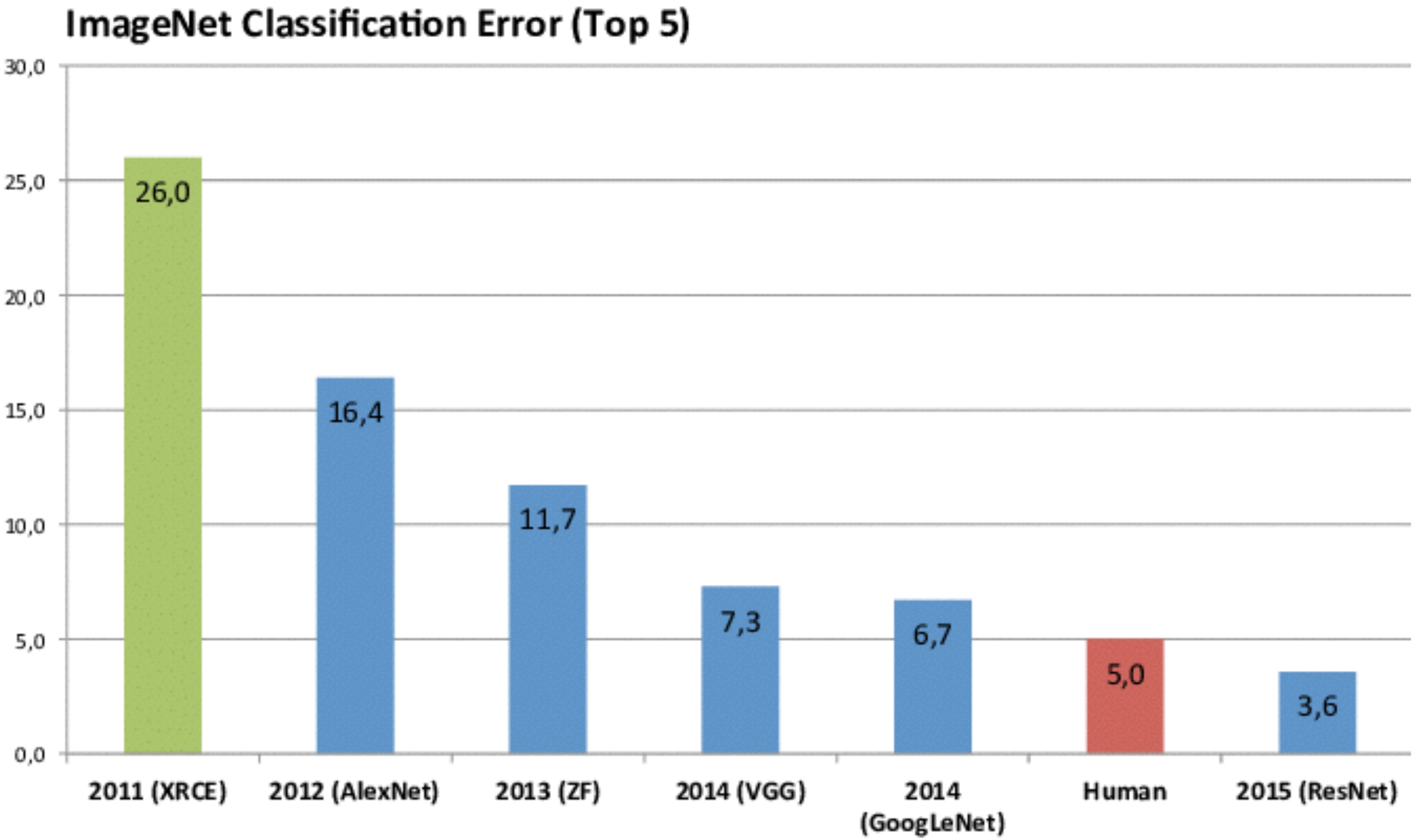


25 coefficients, but only
18 degrees of freedom



9 coefficients, but only
6 degrees of freedom.
Only separable filters… would this be enough?


ImageNet Classification Error (Top 5)
30,0
25,0
20,0
15,0
10,0
5,0
0,0
26,0
16,4
11,7
7,3
2011 (XRCE)
2012 (AlexNet)
2013 (ZF)
2014 (VGG)

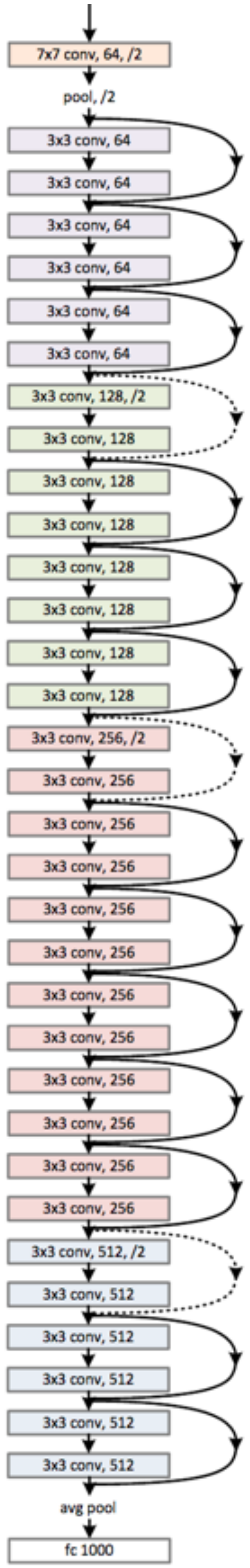


2016: ResNet
>100 conv. layers

Error: 3.6%

[He et al: Deep Residual Learning for Image Recognition, CVPR 2016]

2016: ResNet
>100 conv. layers

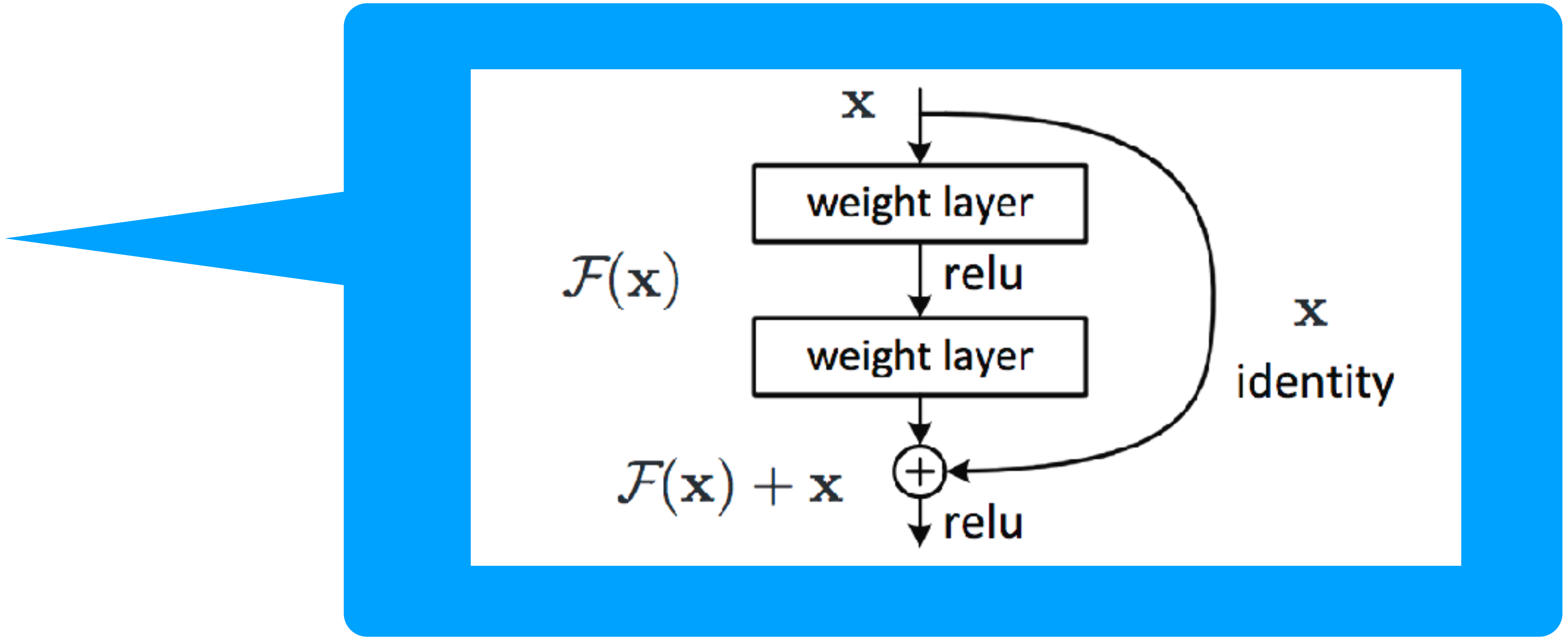


Error: 3.6%

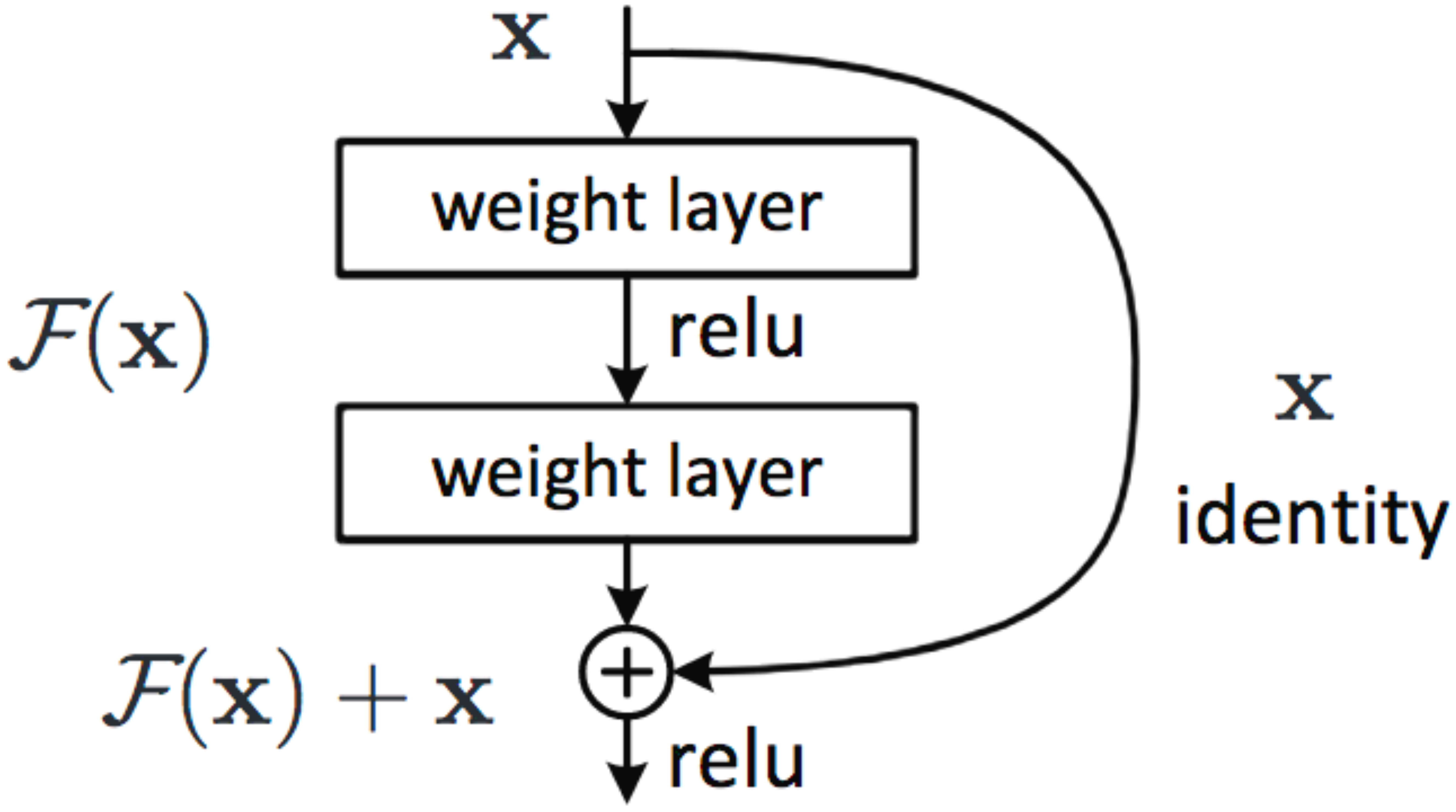
ResNet [He et al, 2016]

Main developments

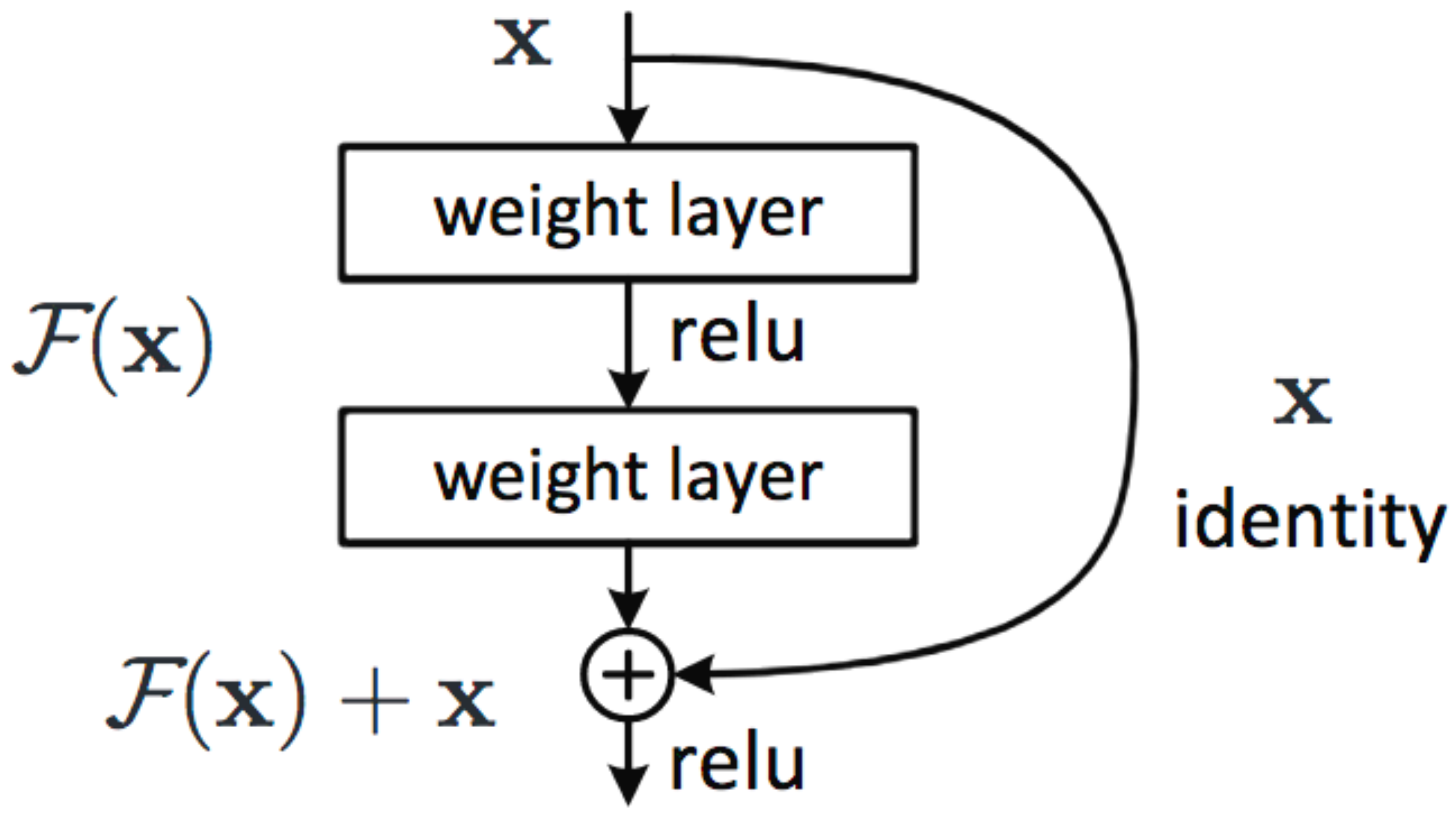
- Increased depth possible through residual blocks



Residual Blocks



Residual Blocks



Why do they work?

- Gradients can propagate faster (via the identity mapping)
- Within each block, only small residuals have to be learned