Lecture 9

CNNs and Spatial Processing

6.8300/1 Advances in Computer Vision

Spring 2023

Bill Freeman, Vincent Sitzmann, Mina Konaković Luković

9. CNNs and Spatial Processing

- How to use deep nets for images
- New layer types: convolutional, pooling
- Feature maps and multichannel representations
- Popular architectures: Alexnet, VGG, Resnets
- Getting to know learned filters
- Unit visualization

Image classification

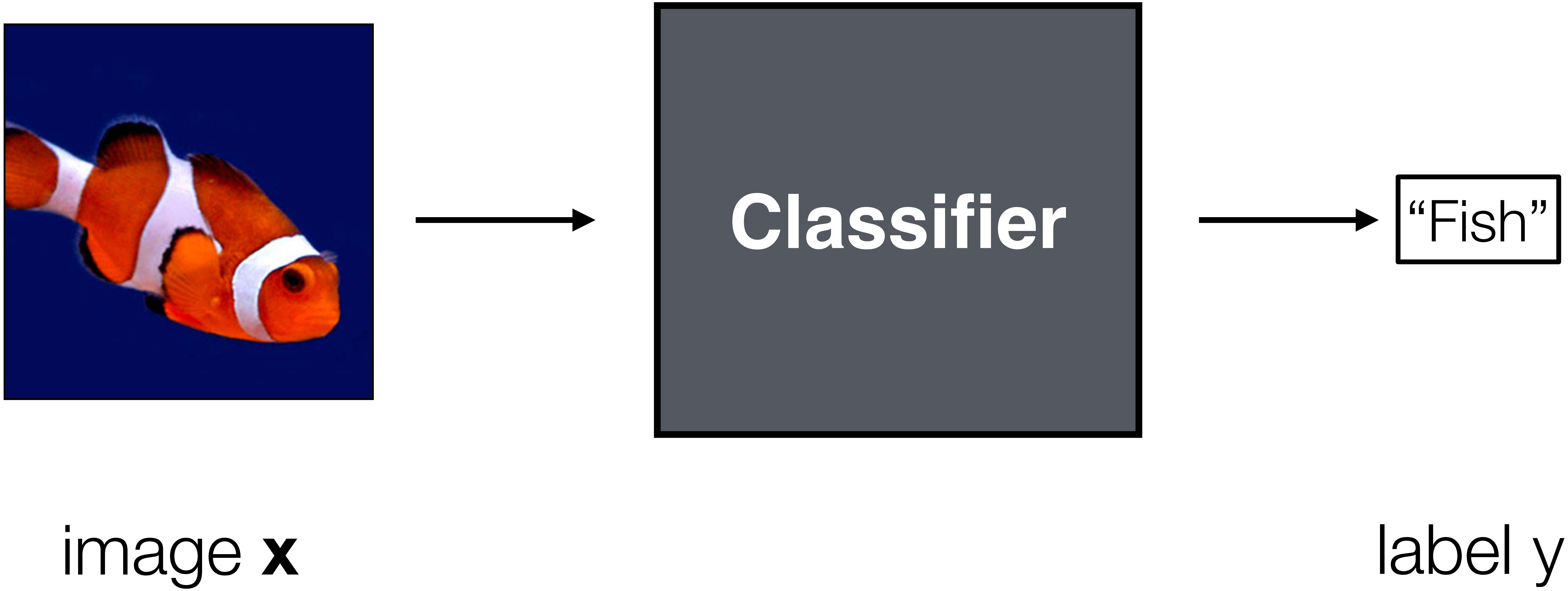
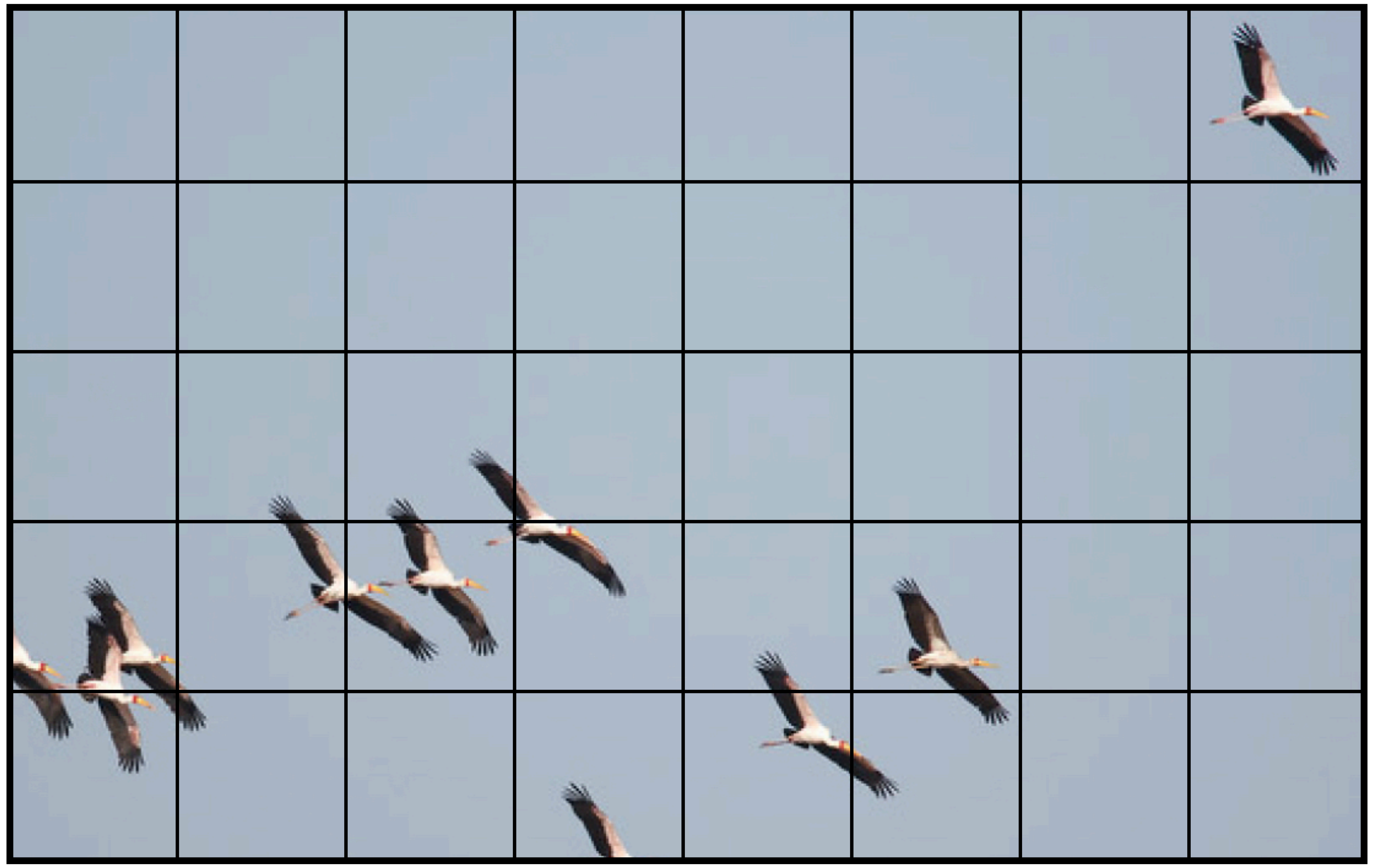
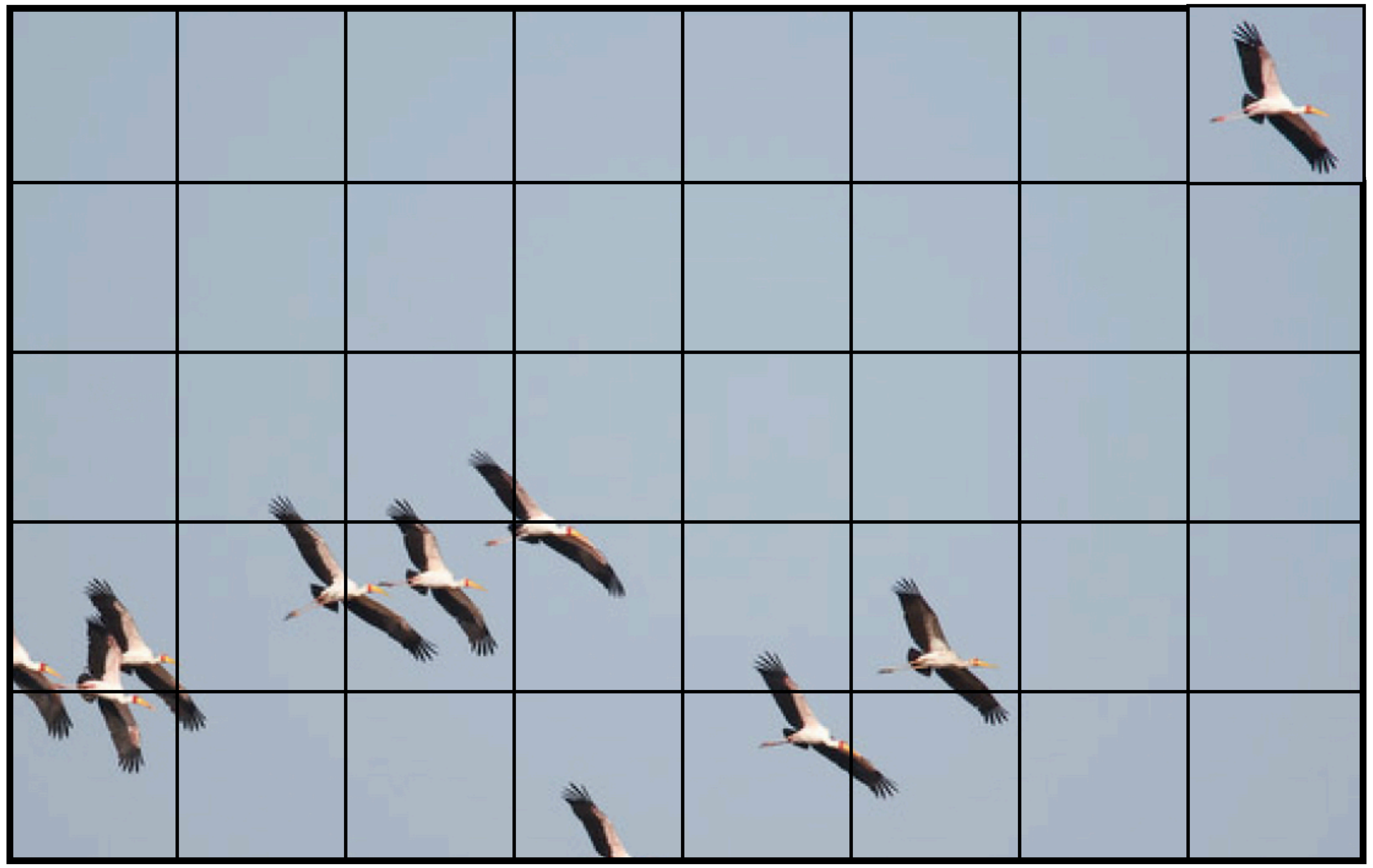
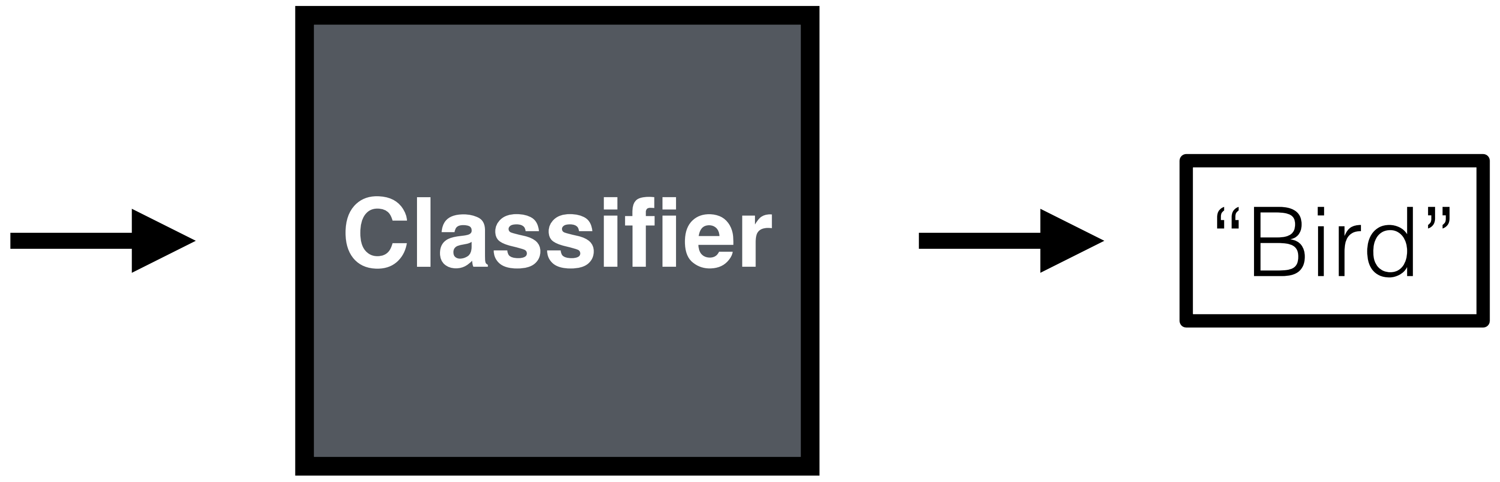


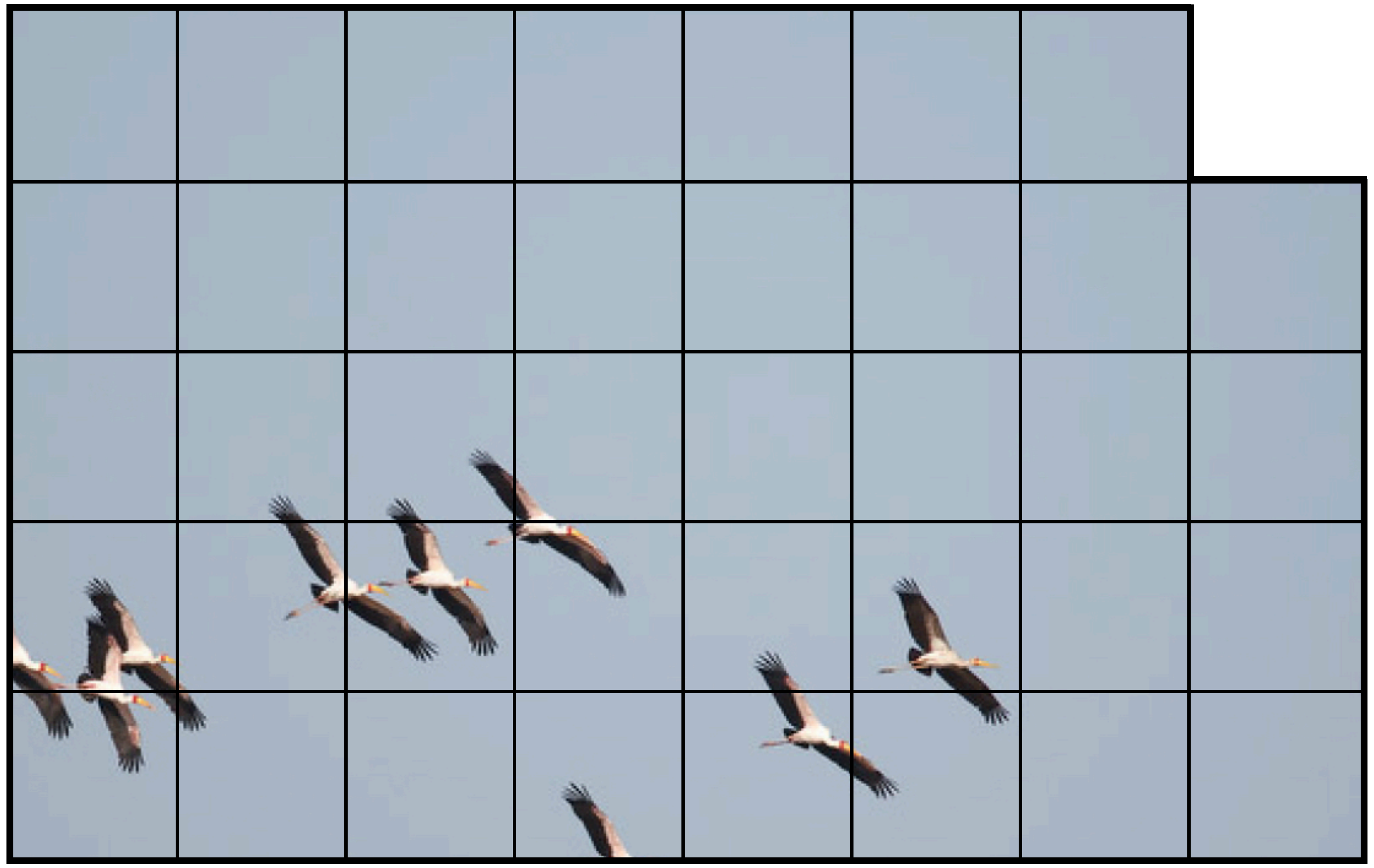

Photo credit: Fredo Durand

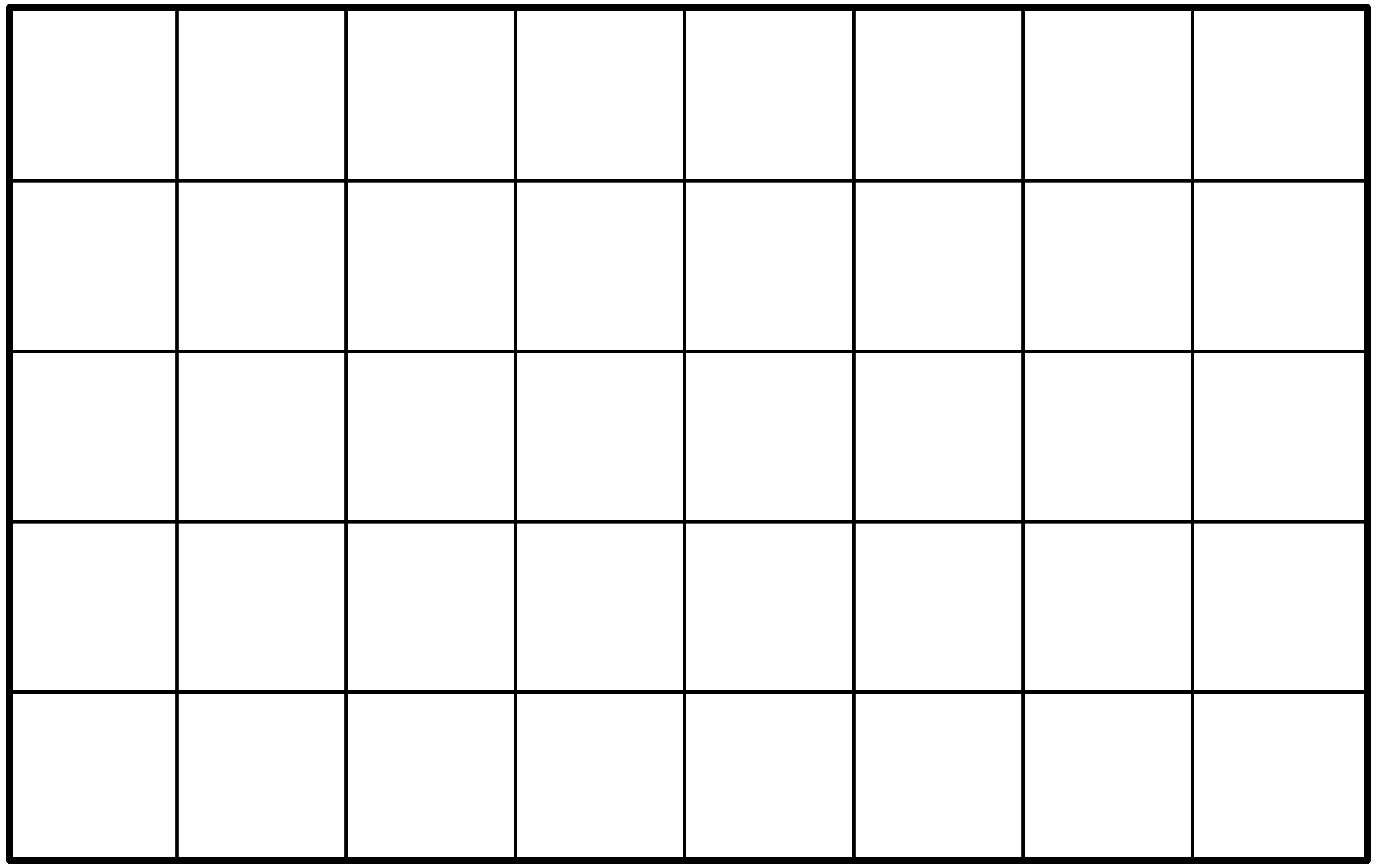




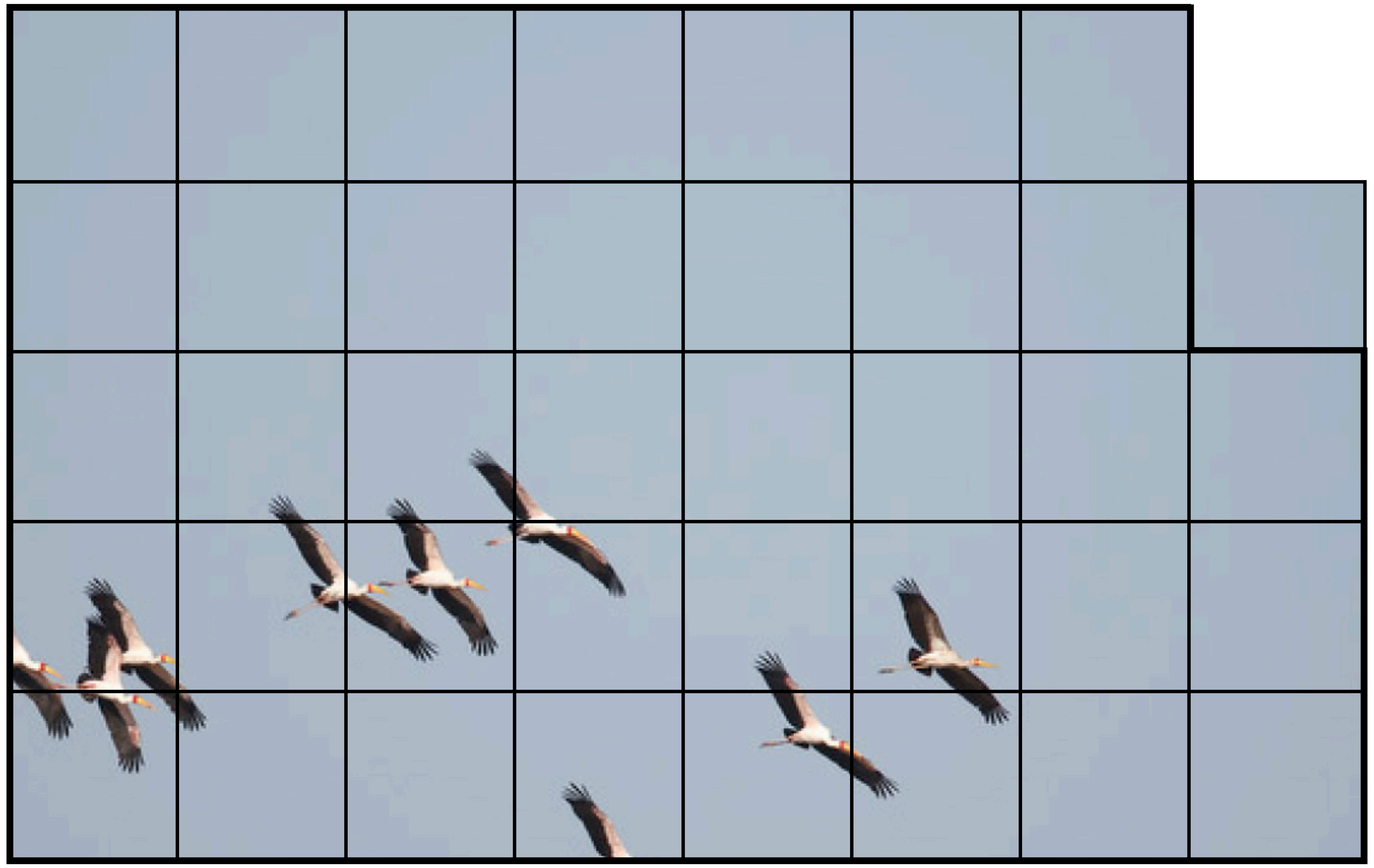


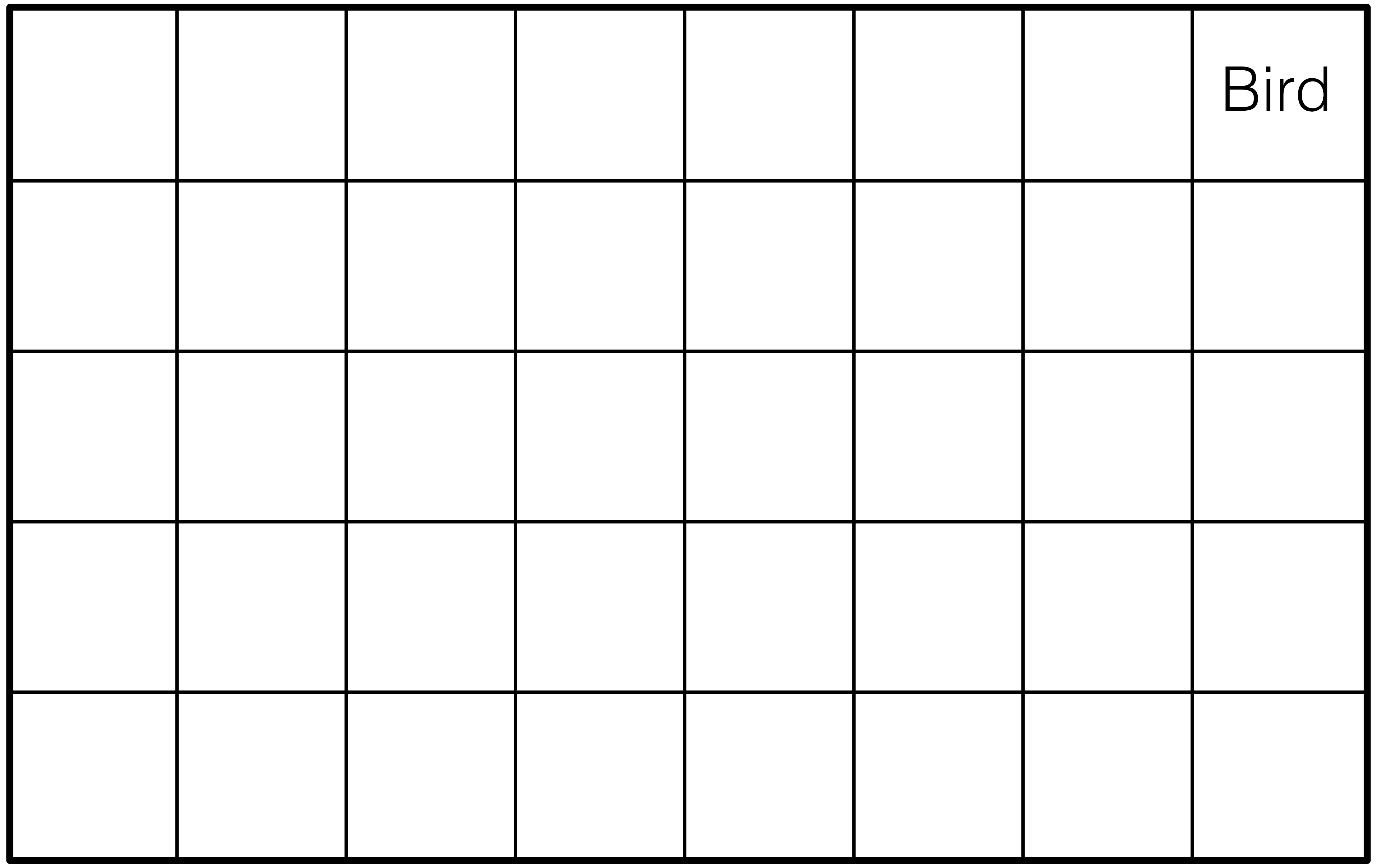
Classifier
"Bird"



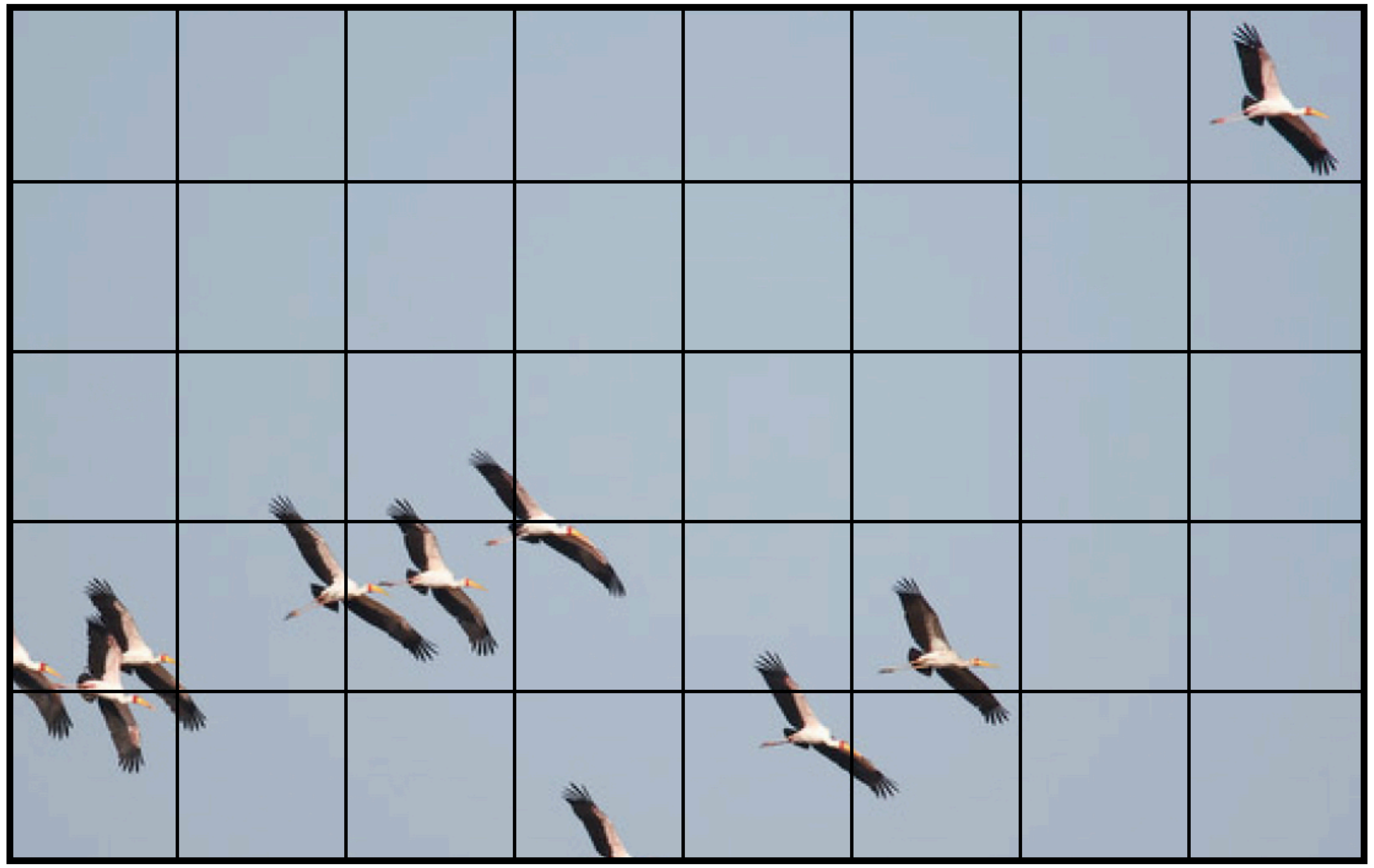


Classifier
“Bird”

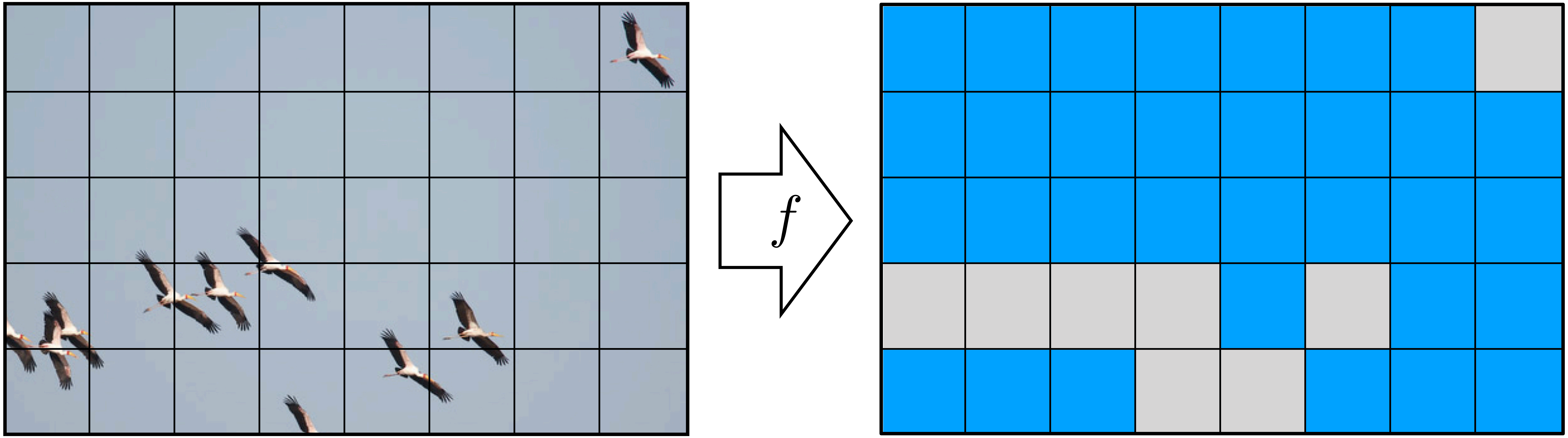



Bird


Classifier
"Sky"



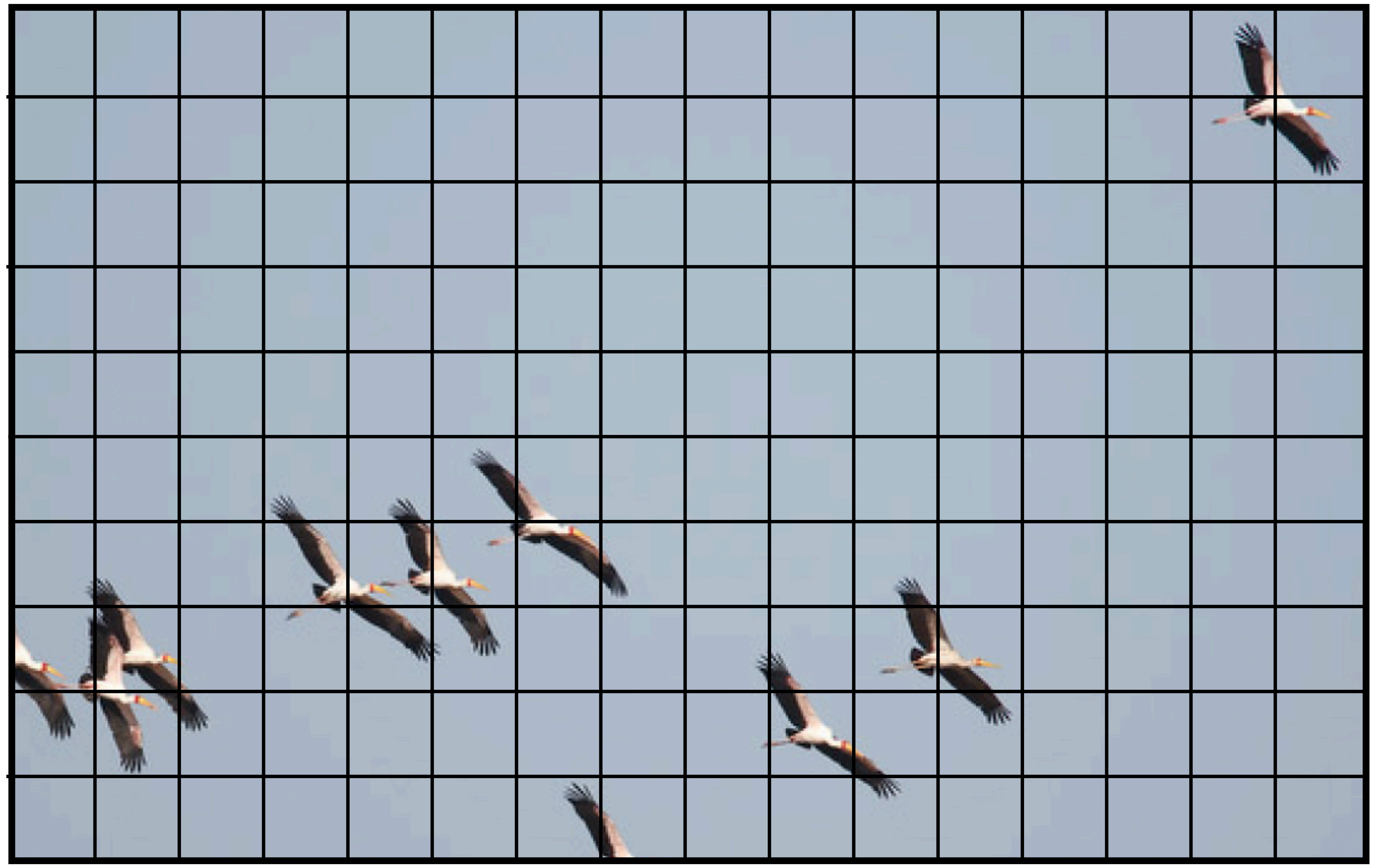
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Bird
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Bird	Bird	Bird	Sky	Bird	Sky	Sky	Sky
Sky	Sky	Sky	Bird	Sky	Sky	Sky	Sky



Problem:

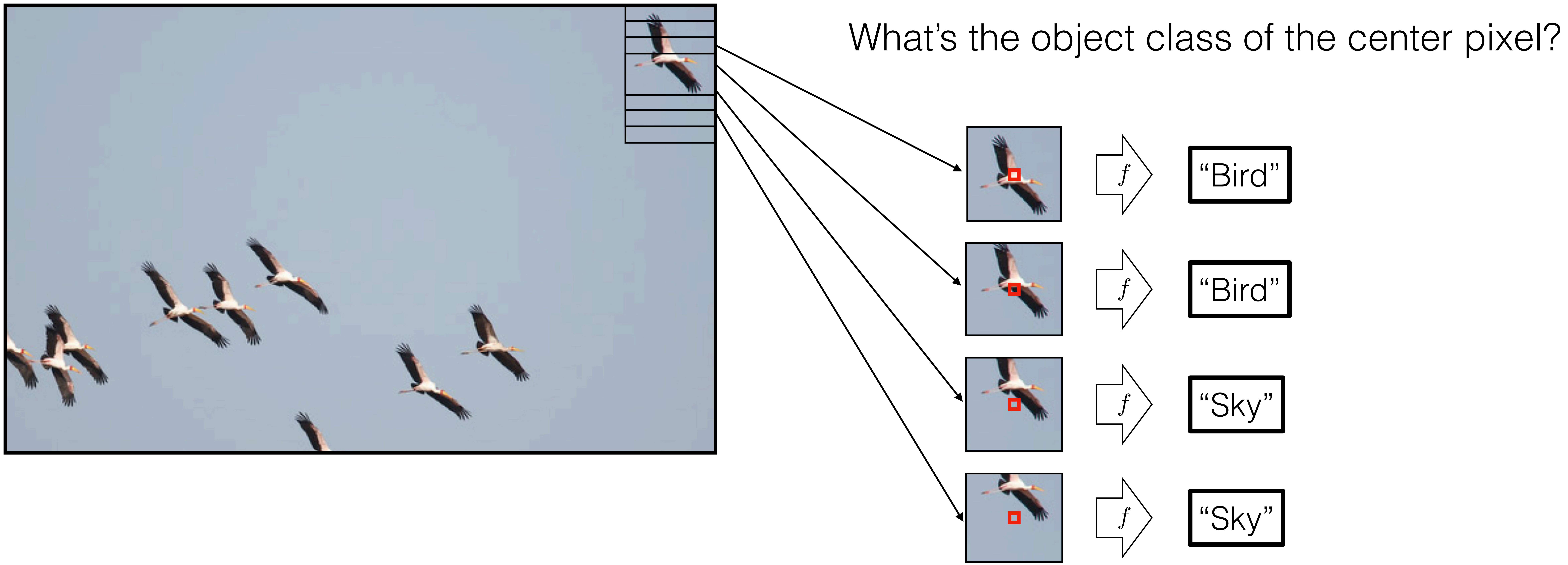
What happens to objects that are bigger?

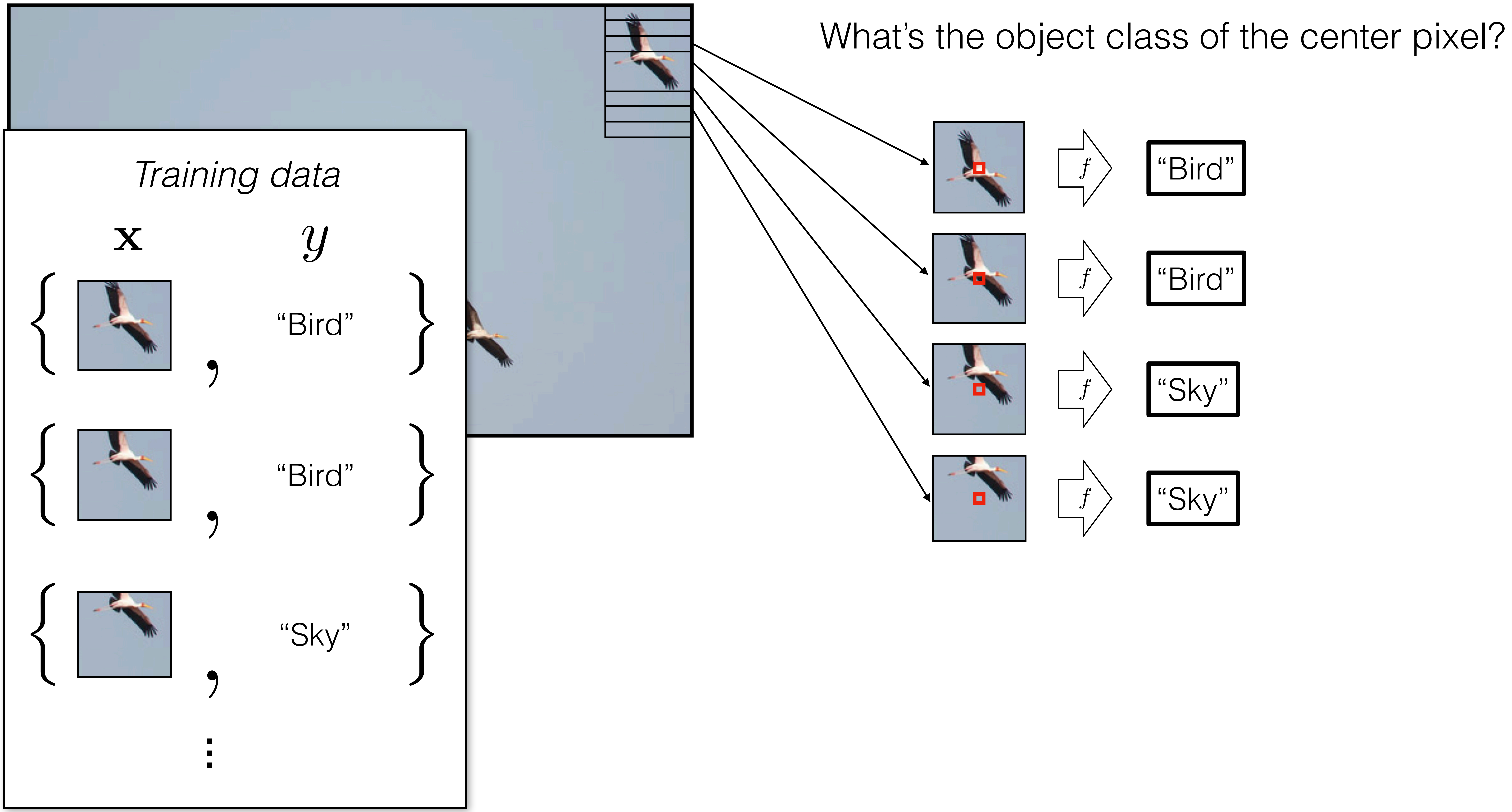
What if an object crosses multiple cells?

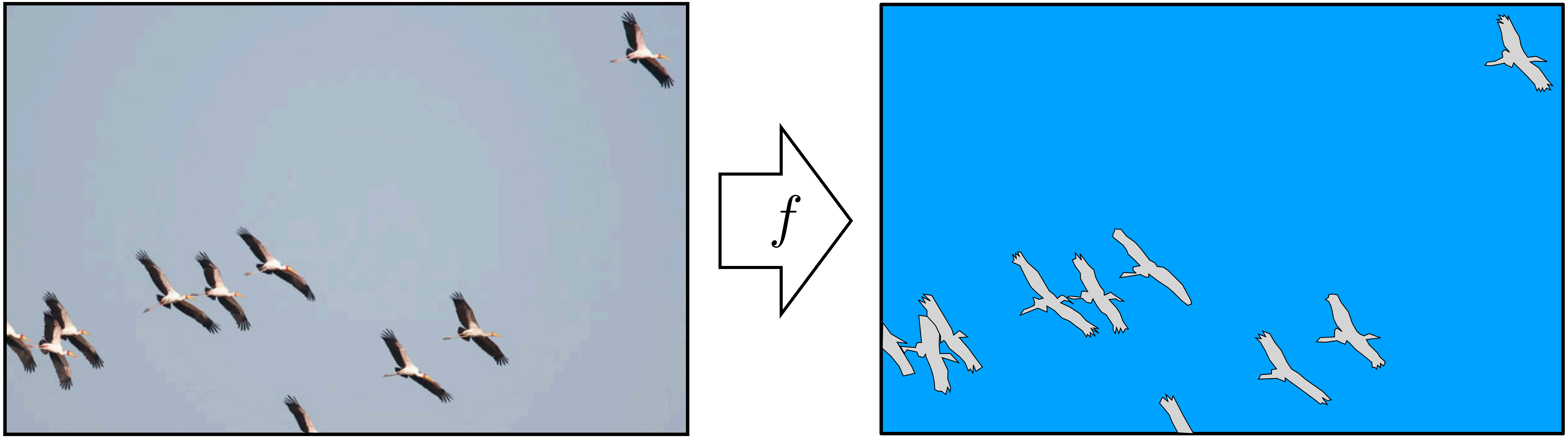


“Cell”-based approach is limited.

What can we do instead?


What's the object class of the center pixel?
f
"Bird"
f
"Bird"
f
"Sky"
f
"Sky"

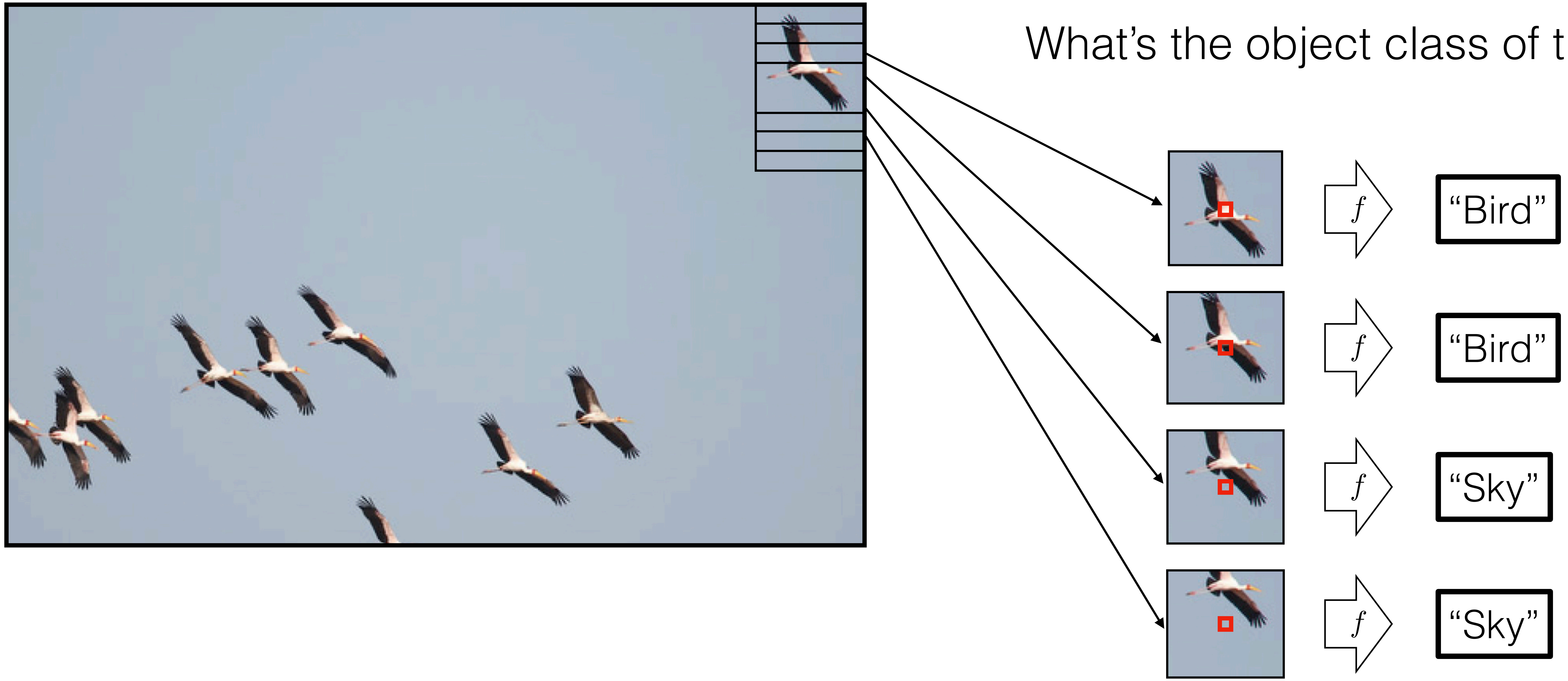

What's the object class of the center pixel?
Training data
x
y
"Bird"
"Bird"
"Sky"
f
"Bird"
"Bird"
"Sky"
"Sky"



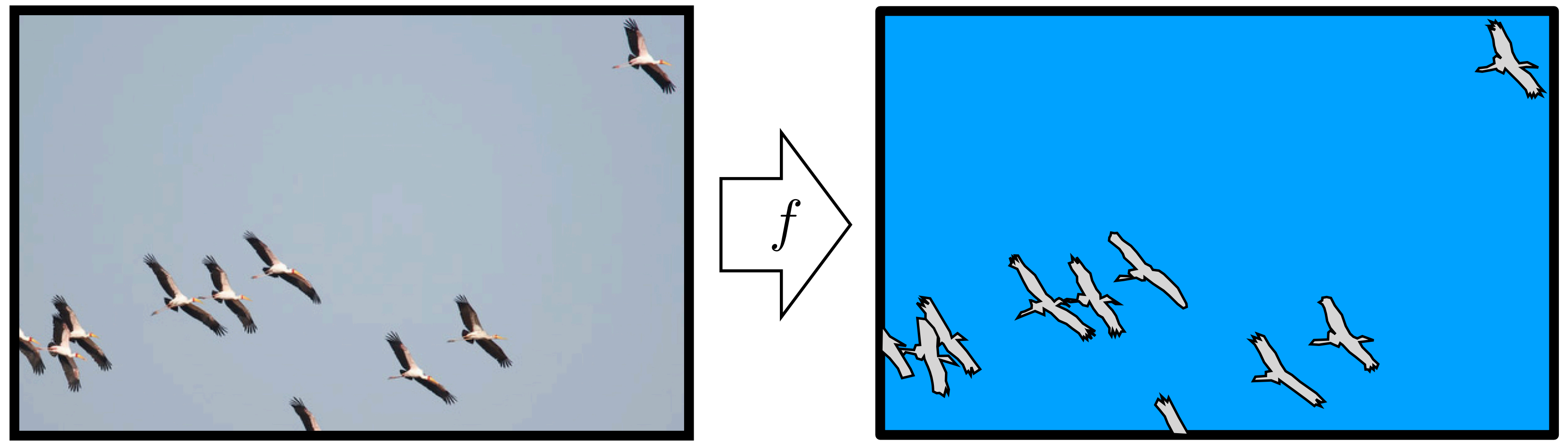
(Colors represent one-hot codes)

This problem is called **semantic segmentation**

What's the object class of the center pixel?



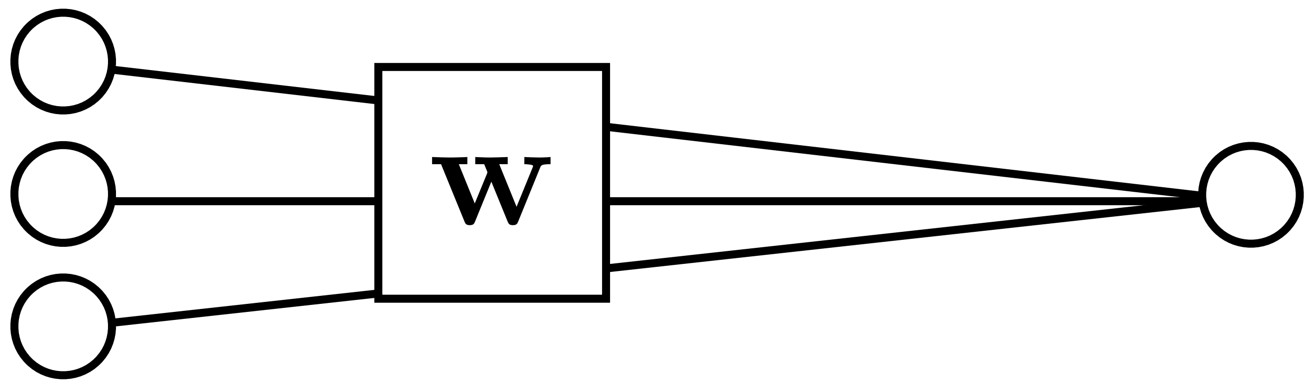
Translation invariance: process each patch in the same way.

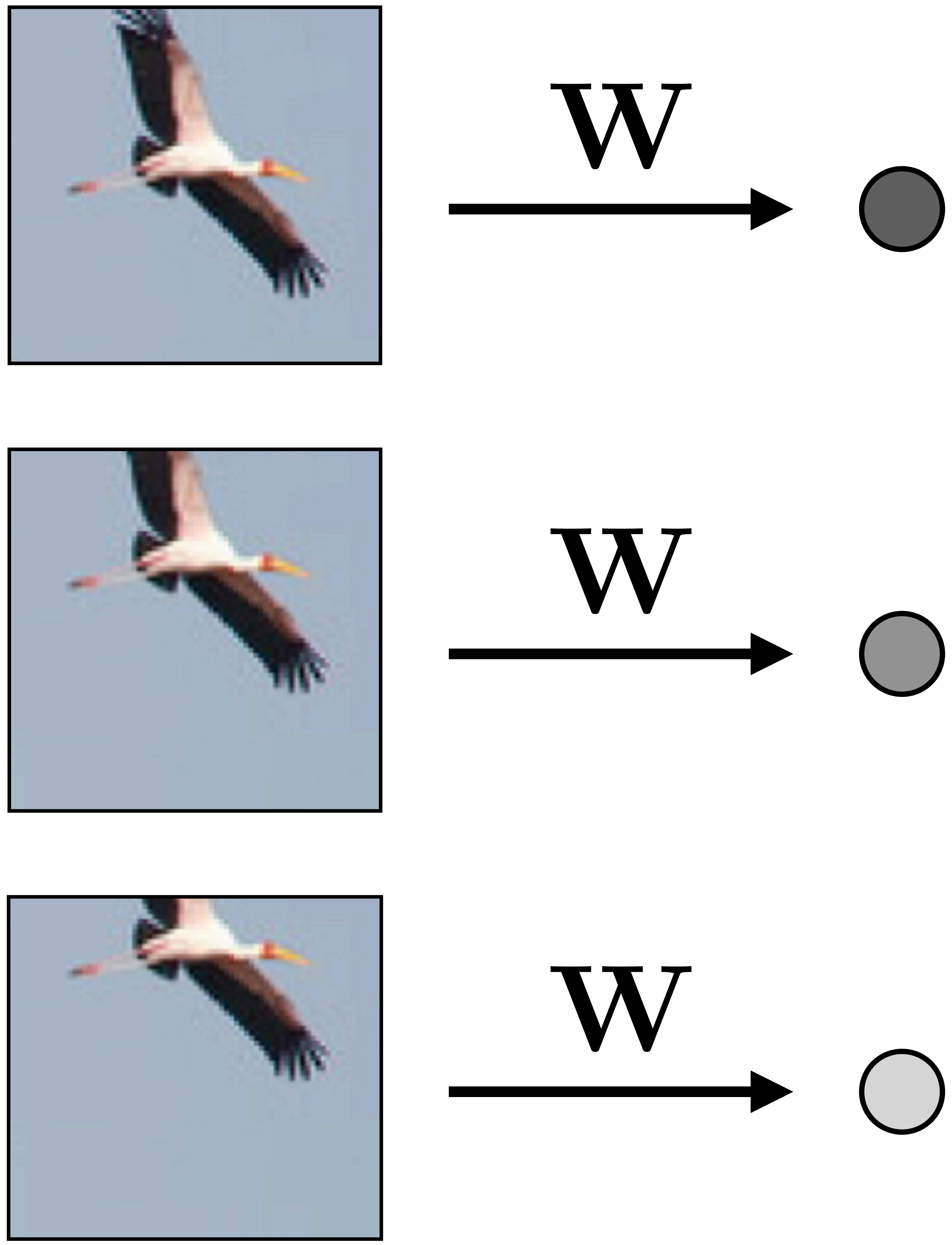


An *equivariant* mapping:

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

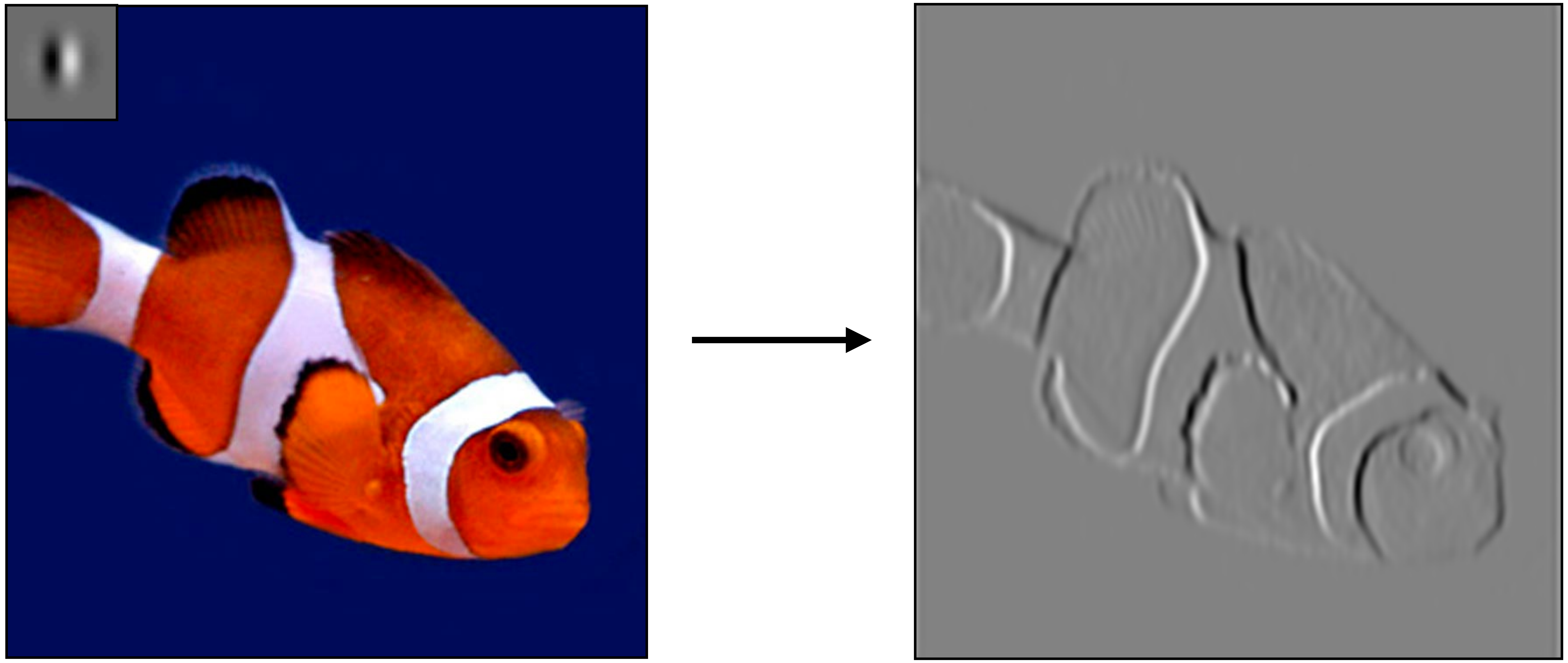
W computes a weighted sum of all pixels in the patch



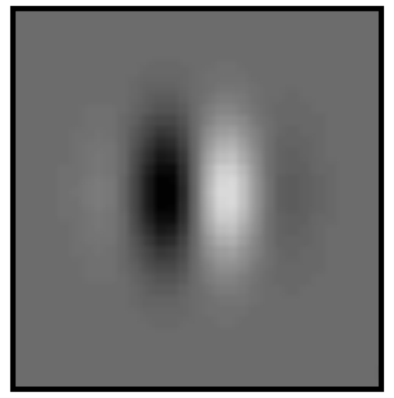


W is a convolutional kernel applied to the full image!

Convolution

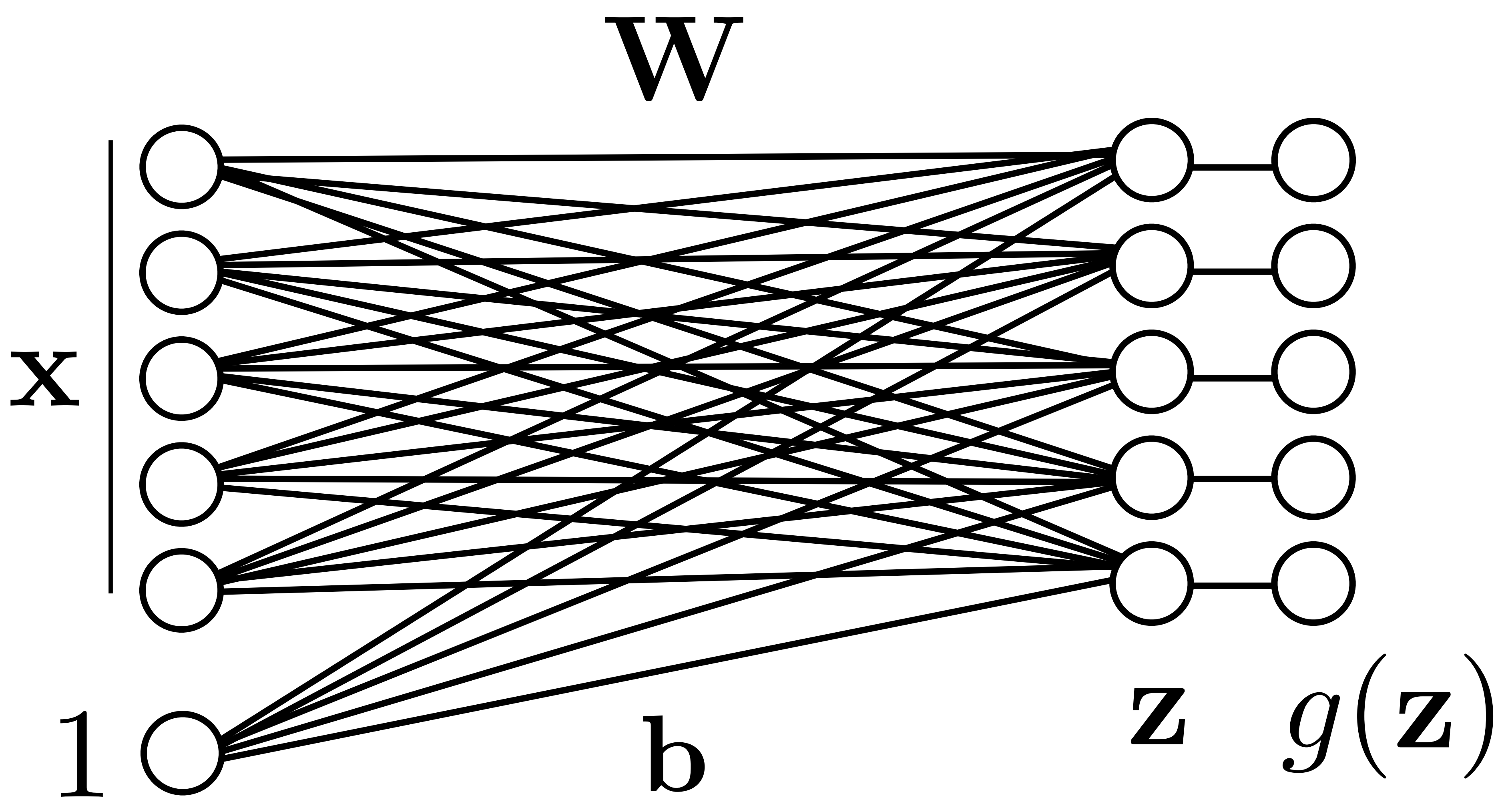


filter

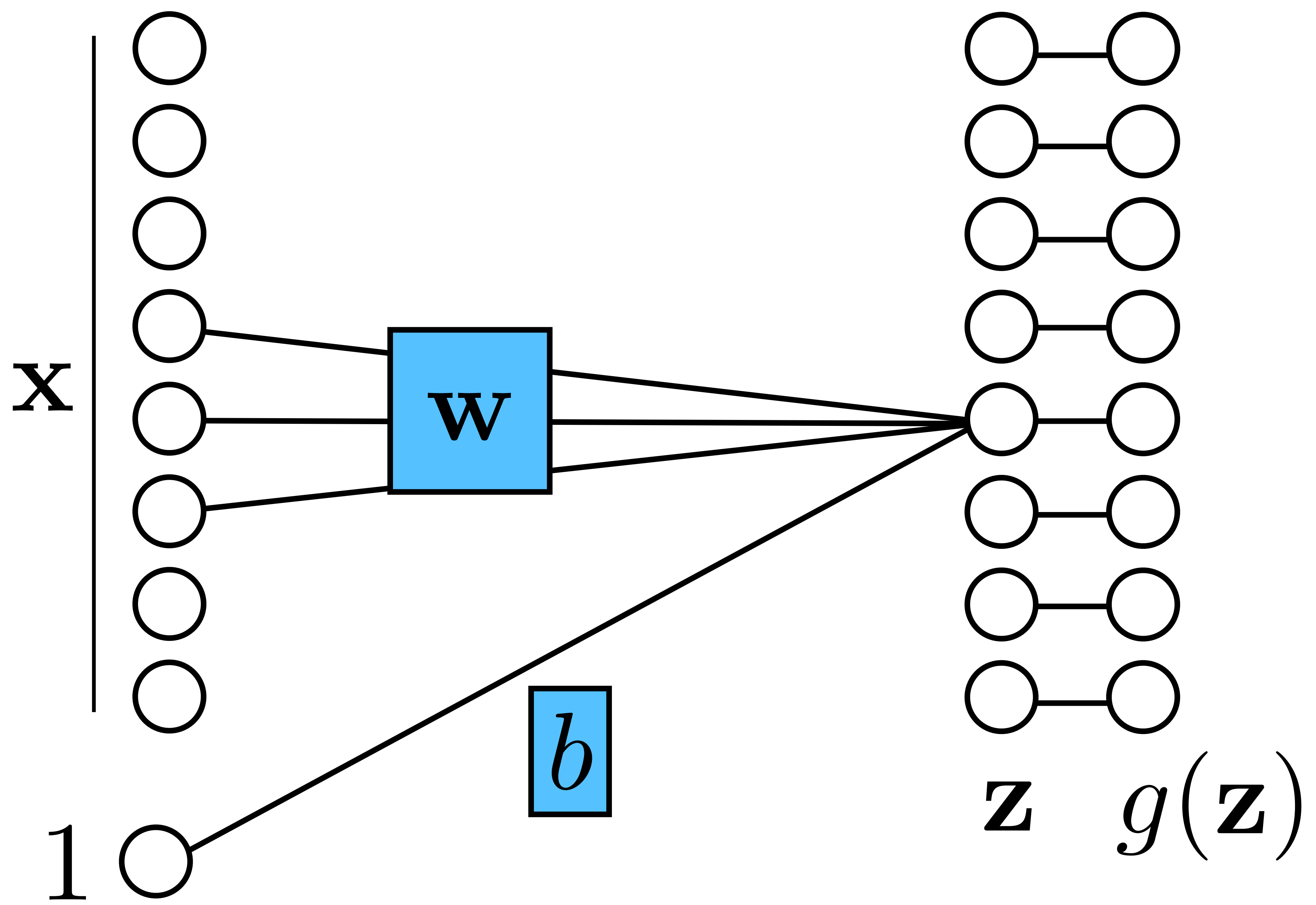


Fully-connected network

Fully-connected (fc) layer



Locally connected network

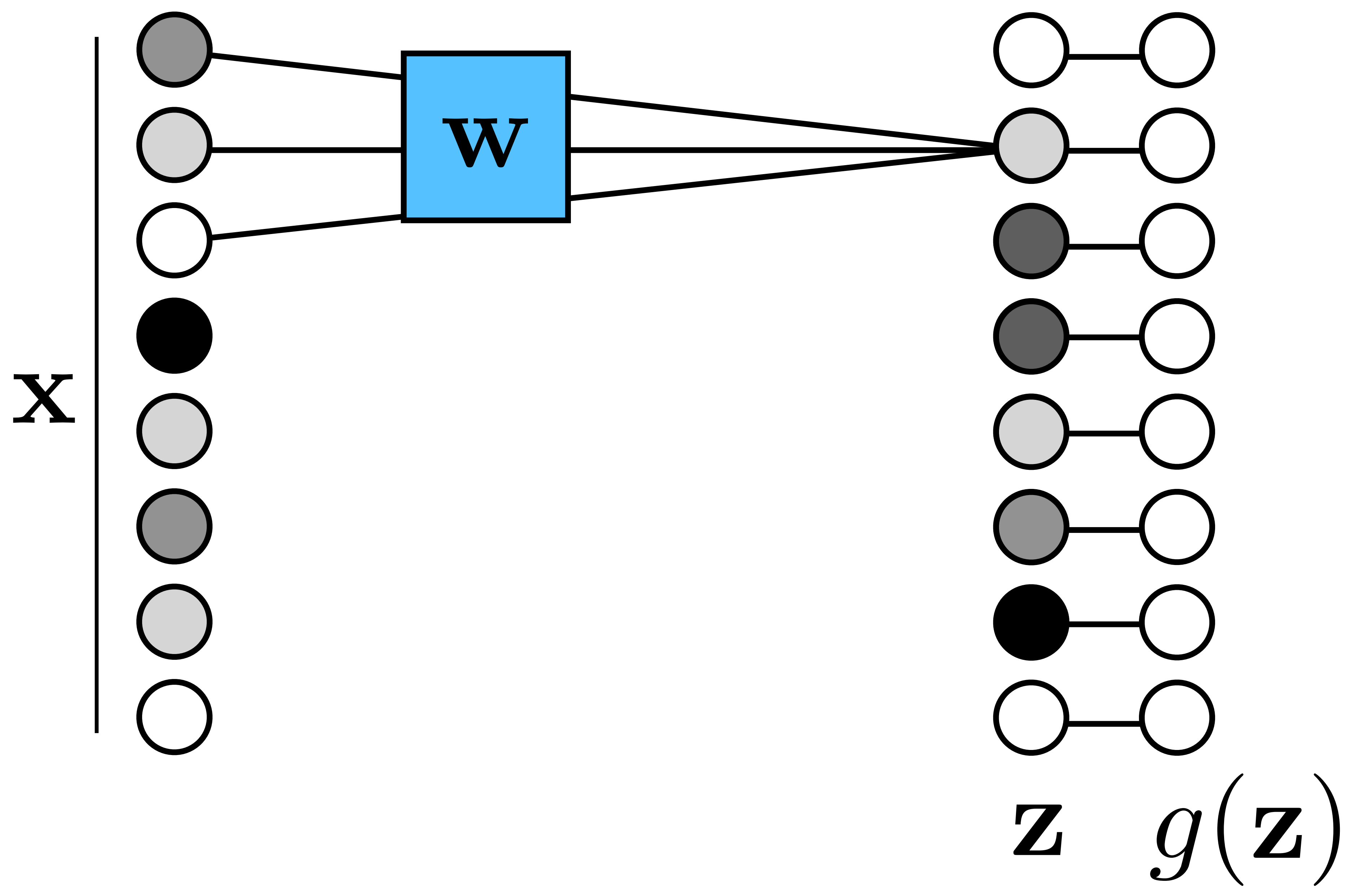


Often, we assume output is a **local** function of input.

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

Convolutional neural network

Conv layer



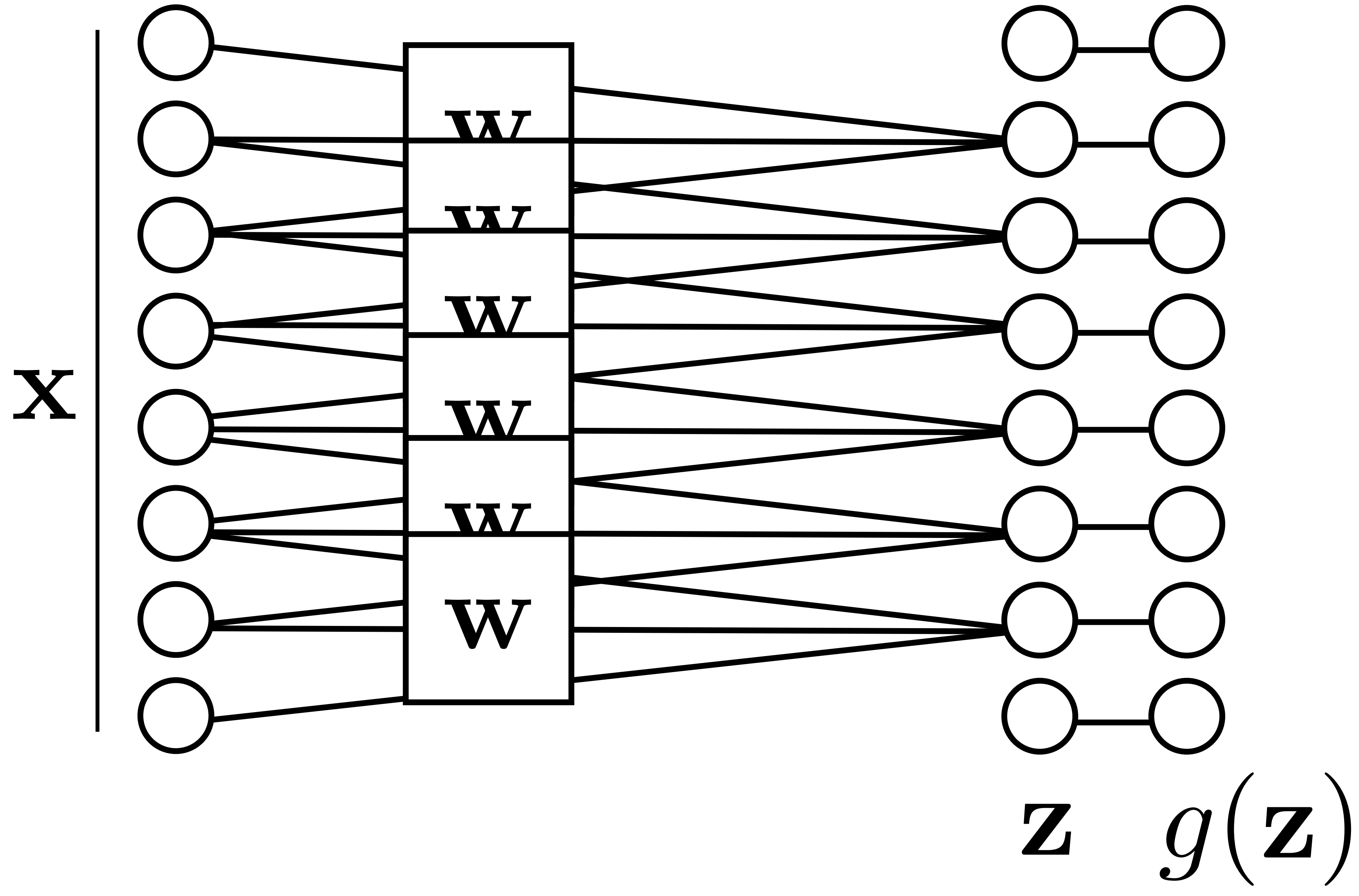
$$\mathbf{z} = \mathbf{w} \circ \mathbf{x} + \mathbf{b}$$

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



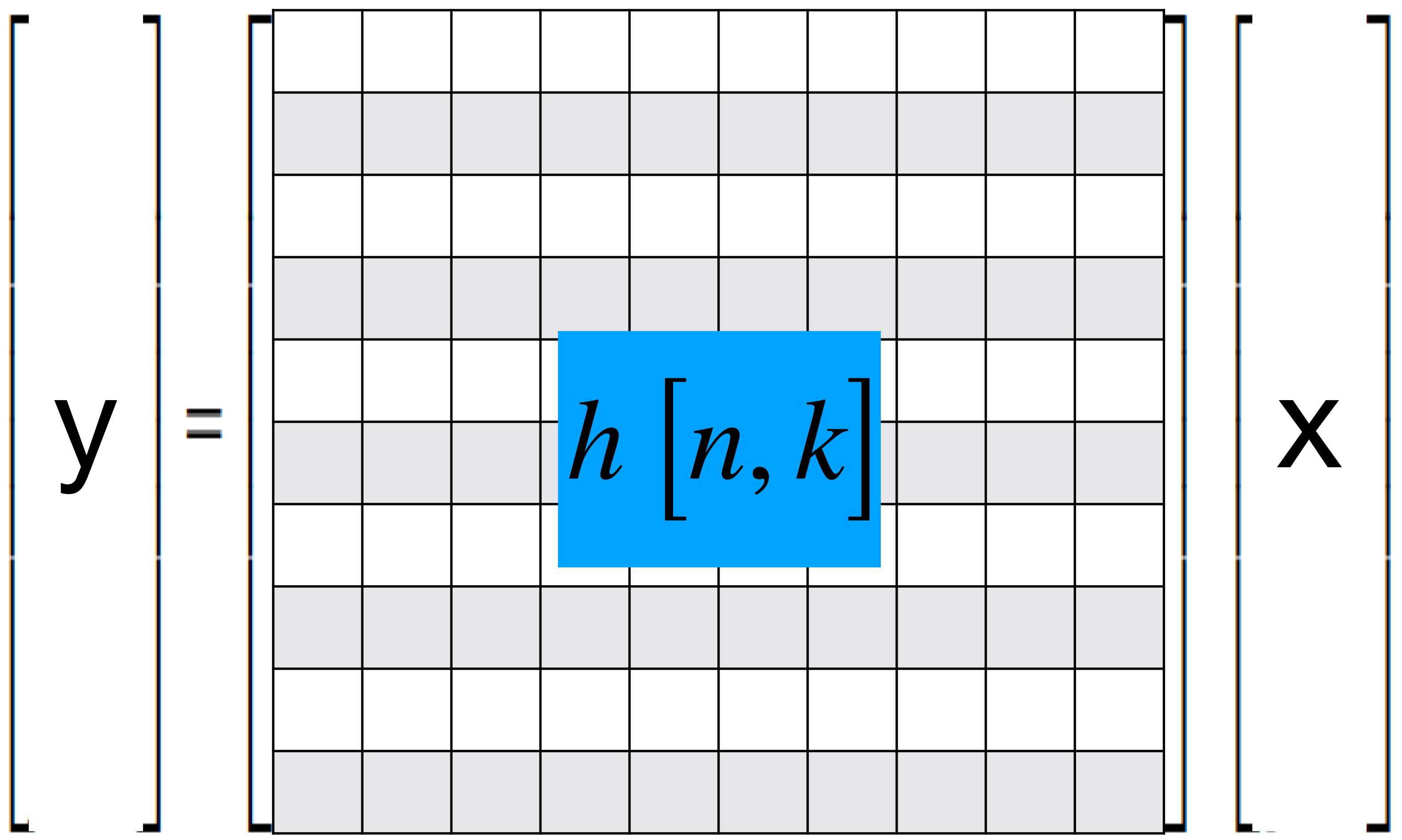
$$\mathbf{z} = \mathbf{w} \circ \mathbf{x} + \mathbf{b}$$

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.

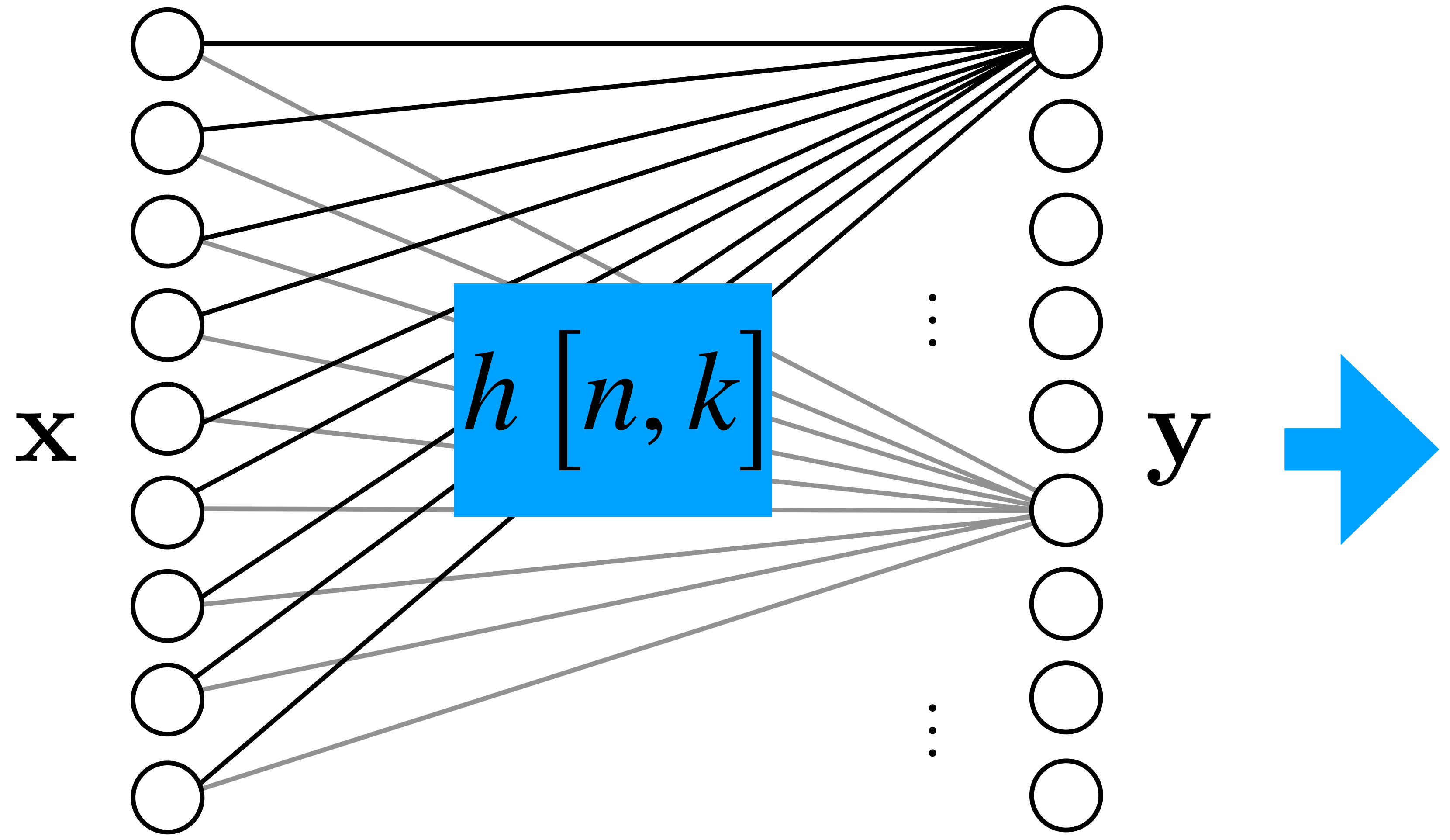
Linear system: $\mathbf{y} = f(\mathbf{x})$

A linear function f can be written as a matrix multiplication:



n indexes rows,
k indexes columns

It can also be represented as a fully connected linear neural network

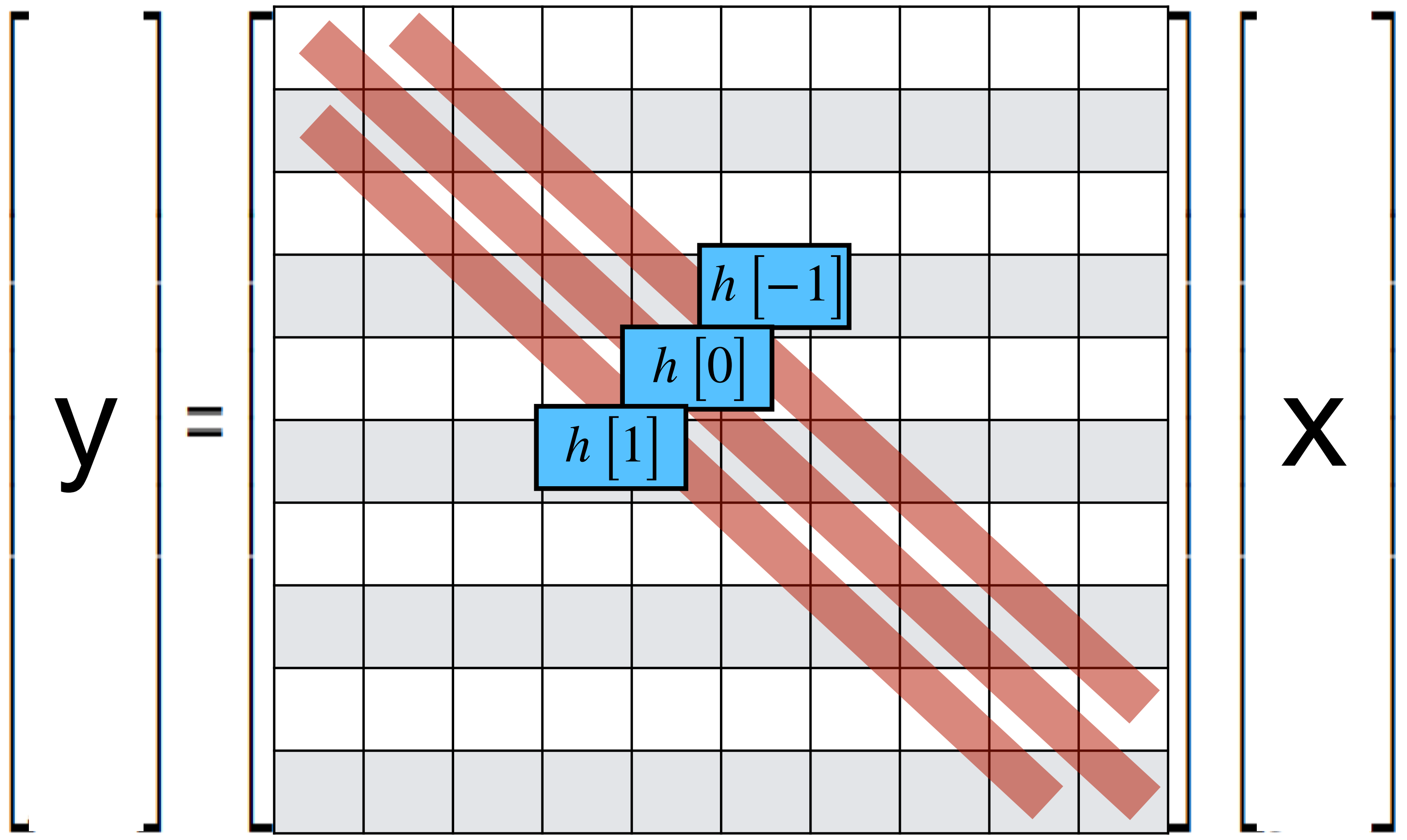


$$y[n] = \sum_{k=0}^{N-1} h[n,k]\, x[k]$$

$h[n,k]$ Is the strength of the connection between x[k] and y[n]

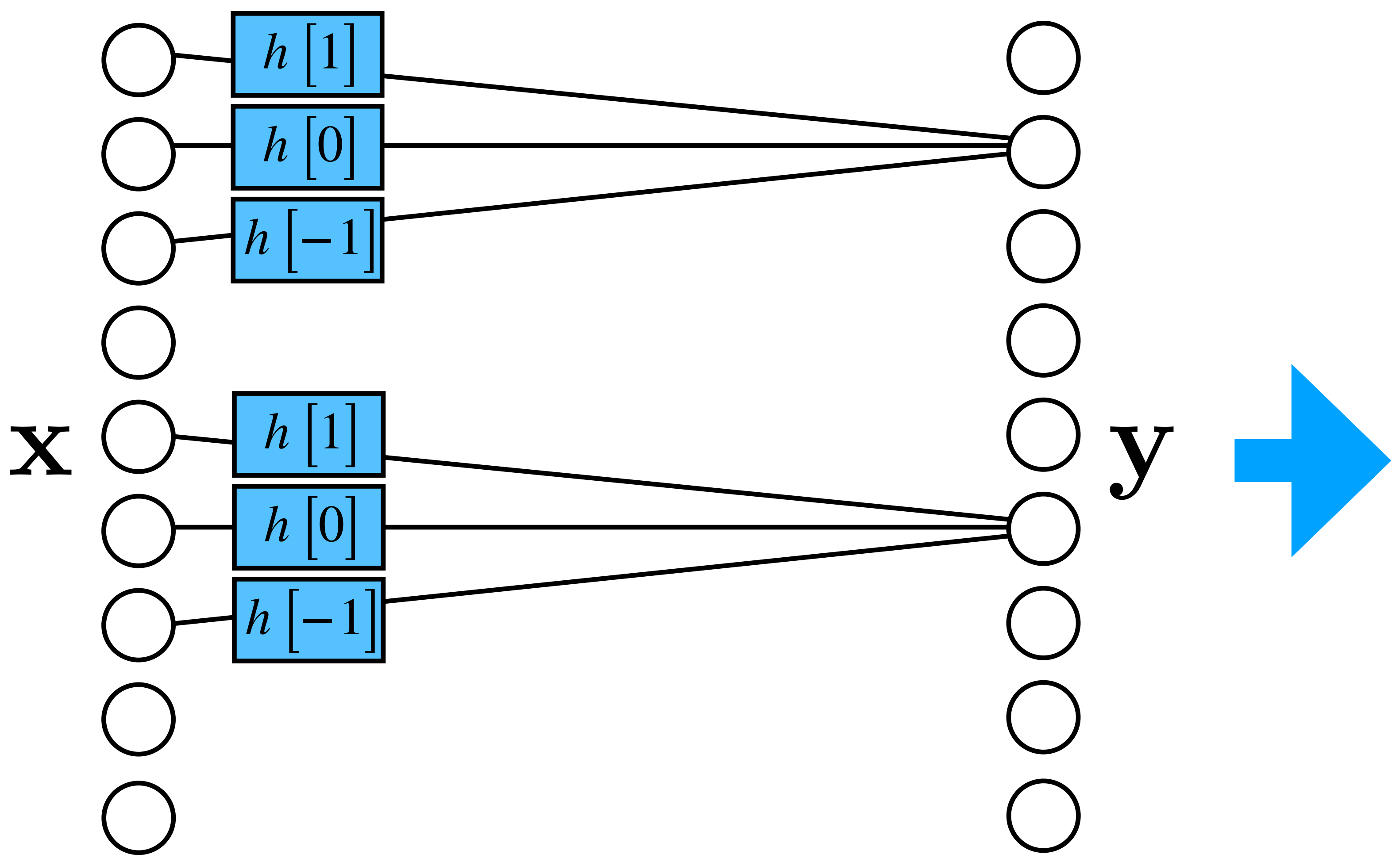
Convolution

A linear shift invariant (LSI) function f can be written as a matrix multiplication:



$h\left[n-k\right]$ n indexes rows, k indexes columns

It can also be represented as a convolutional layer of neural net:

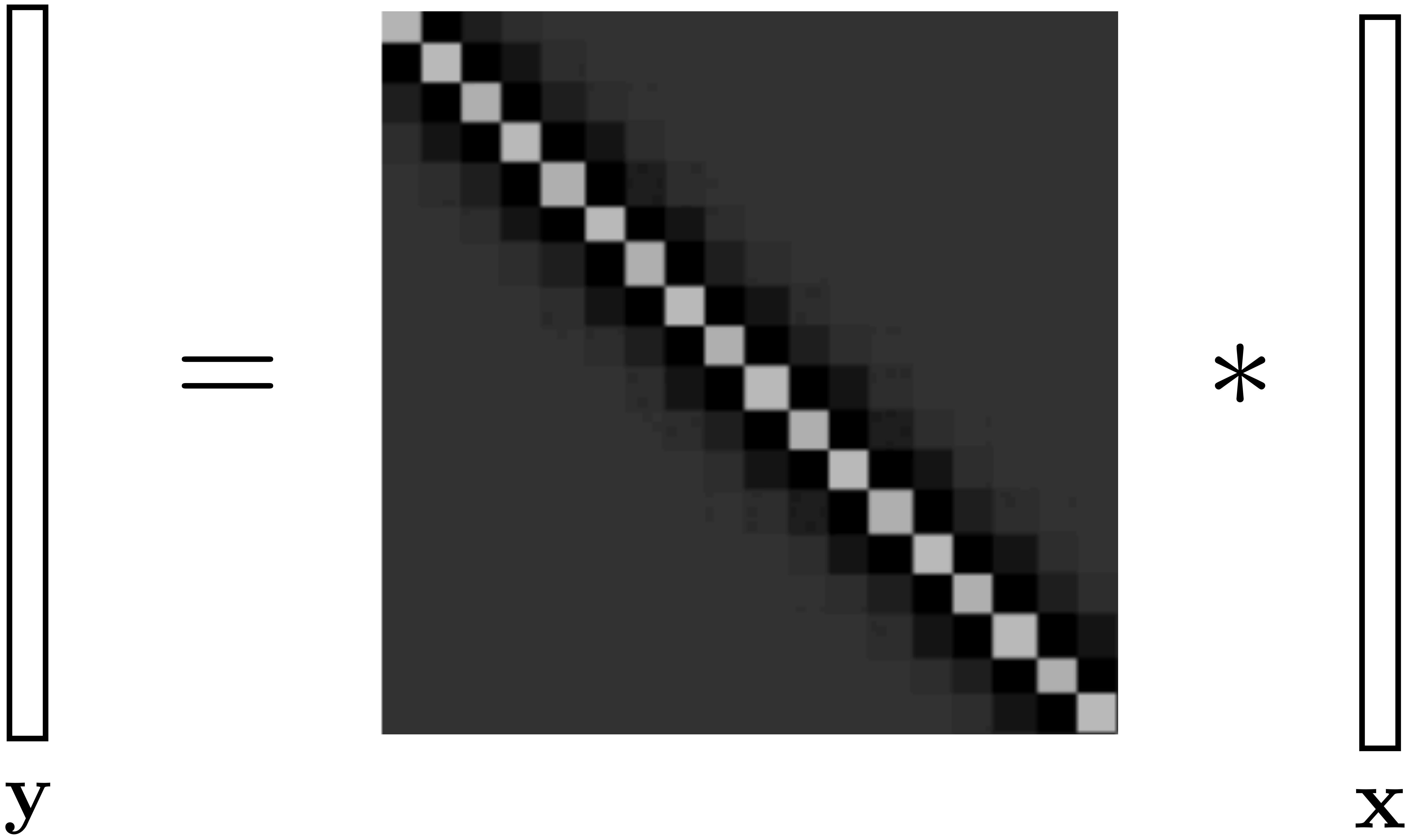


$$y\left[n\right] = \sum_{k=-1}^{1} h\left[k\right] x\left[n-k\right]$$

$h\left[n-k\right]$ Is the strength of the connection between x[k] and y[n]

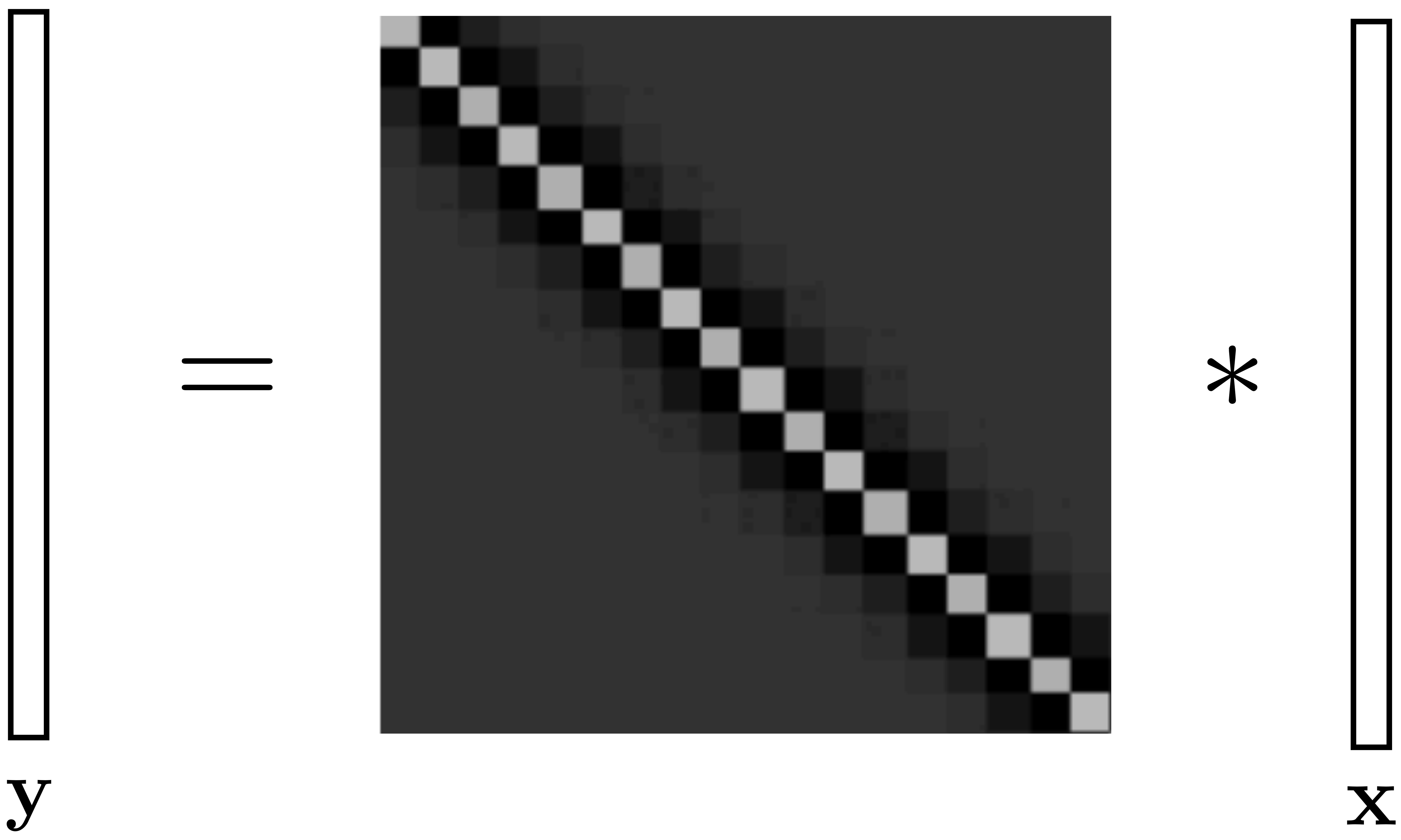
Toeplitz matrix

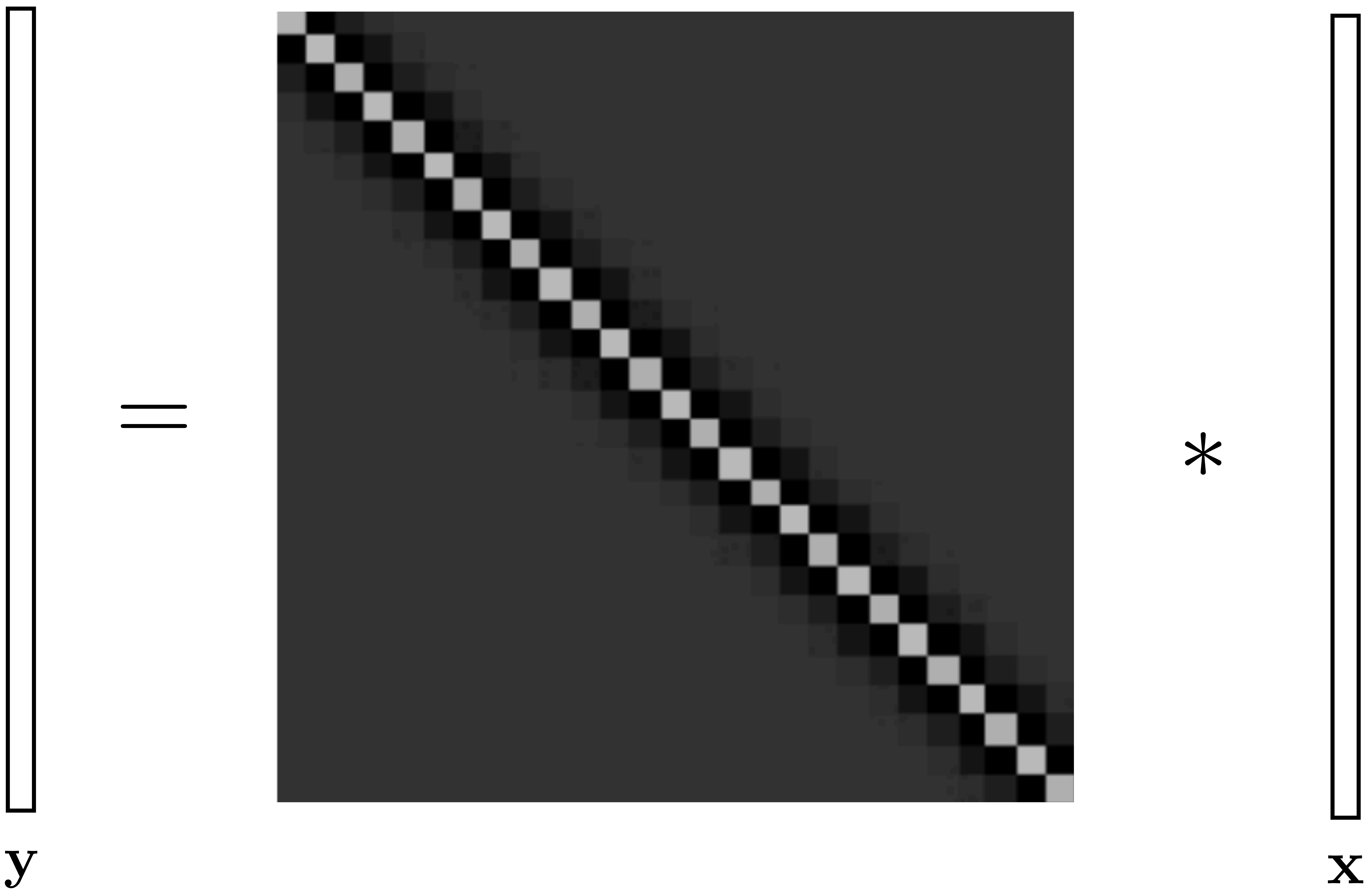
$$\begin{pmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{pmatrix}$$

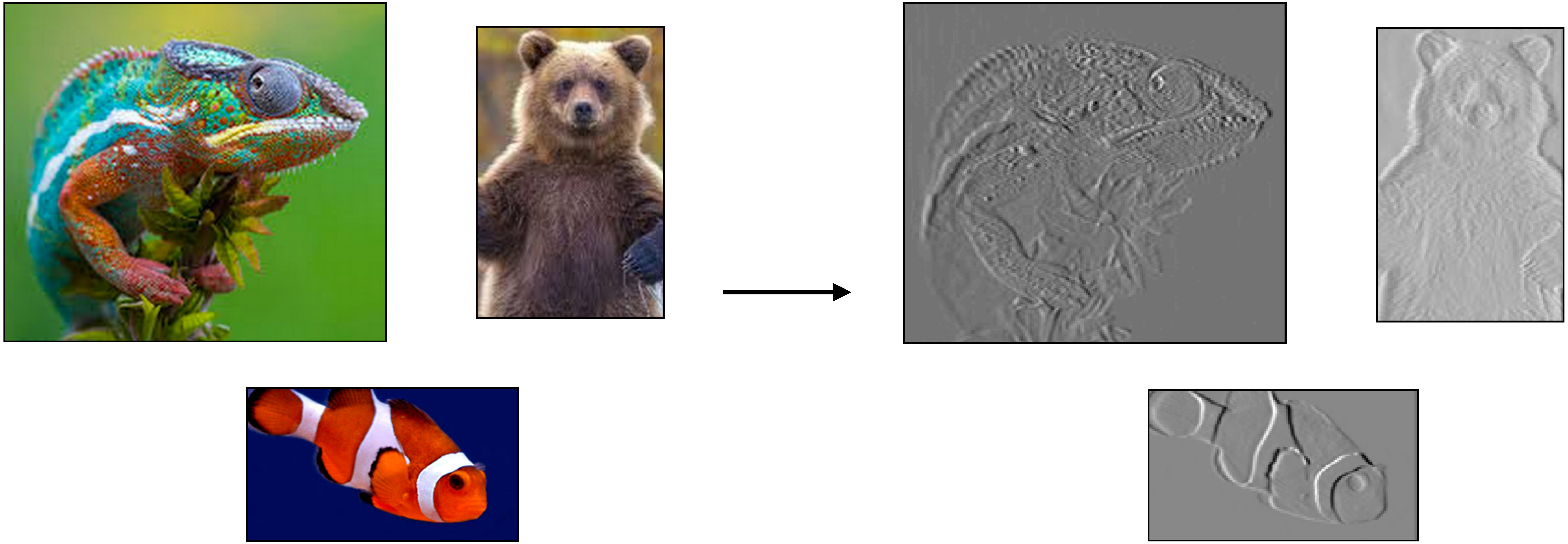


e.g., pixel image

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


=
*
y
x


y
=
*
x

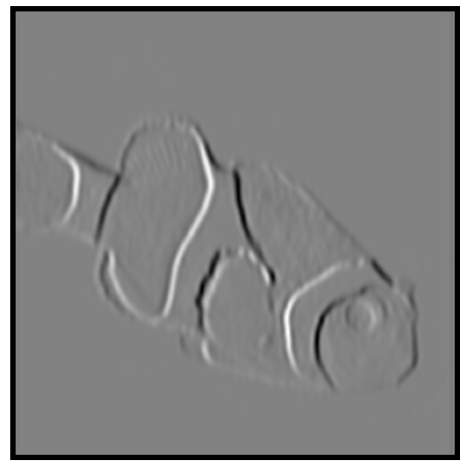


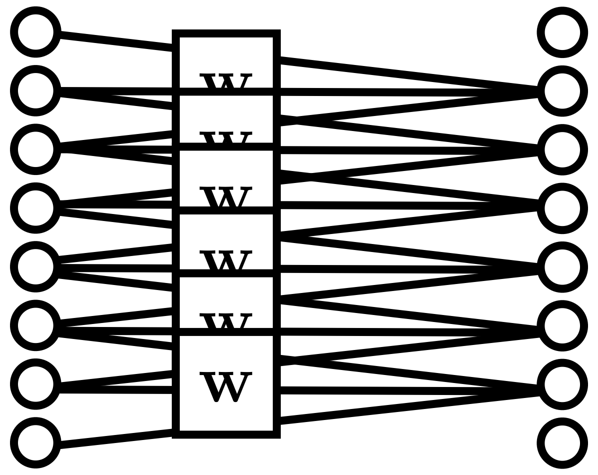
Conv layers can be applied to arbitrarily-sized inputs

Five views on convolutional layers

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

2. Patch processing

3. Image filter 

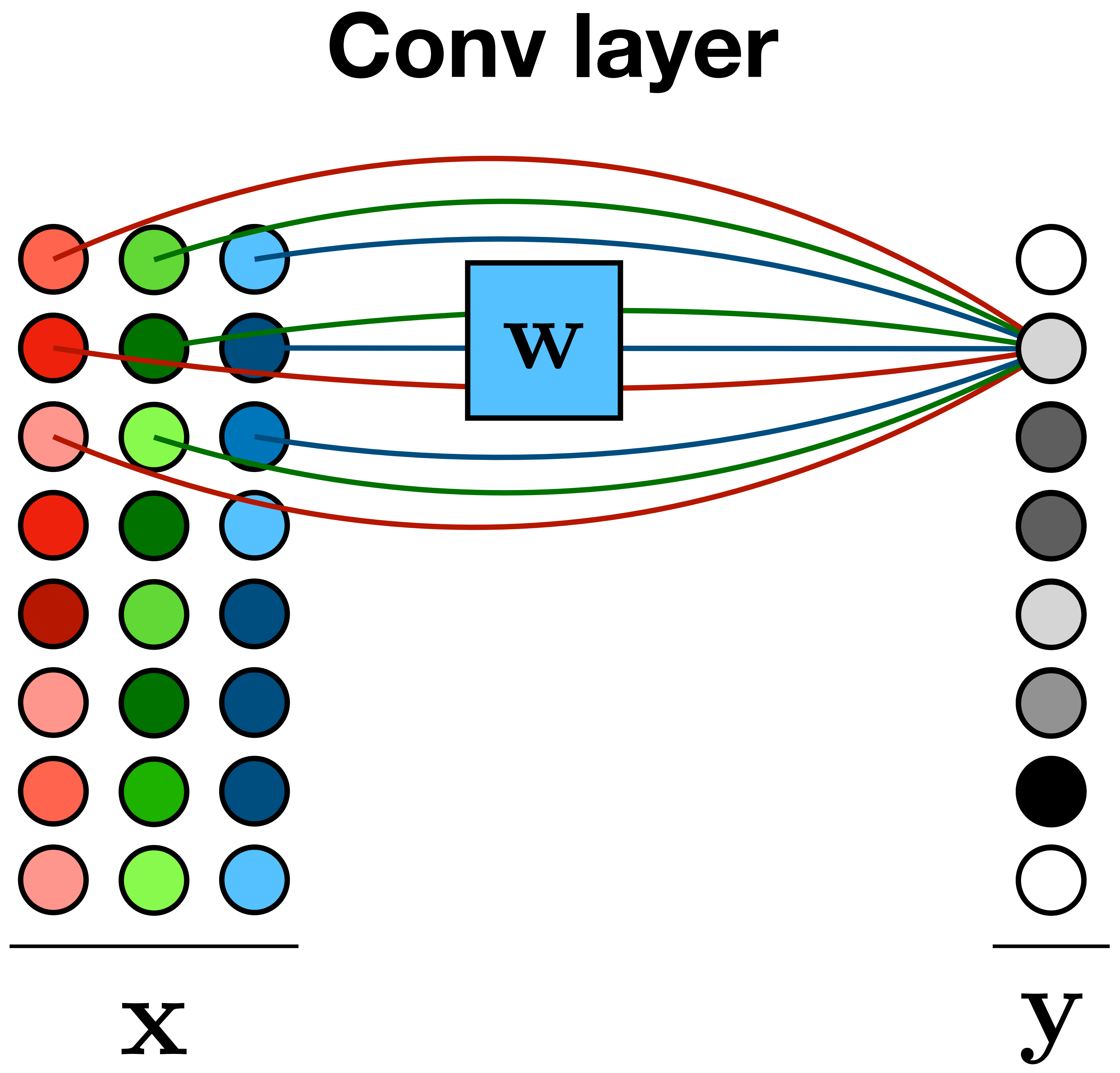
4. Parameter sharing 

5. A way to process variable-sized tensors

What if we have color?

(aka multiple input channels?)

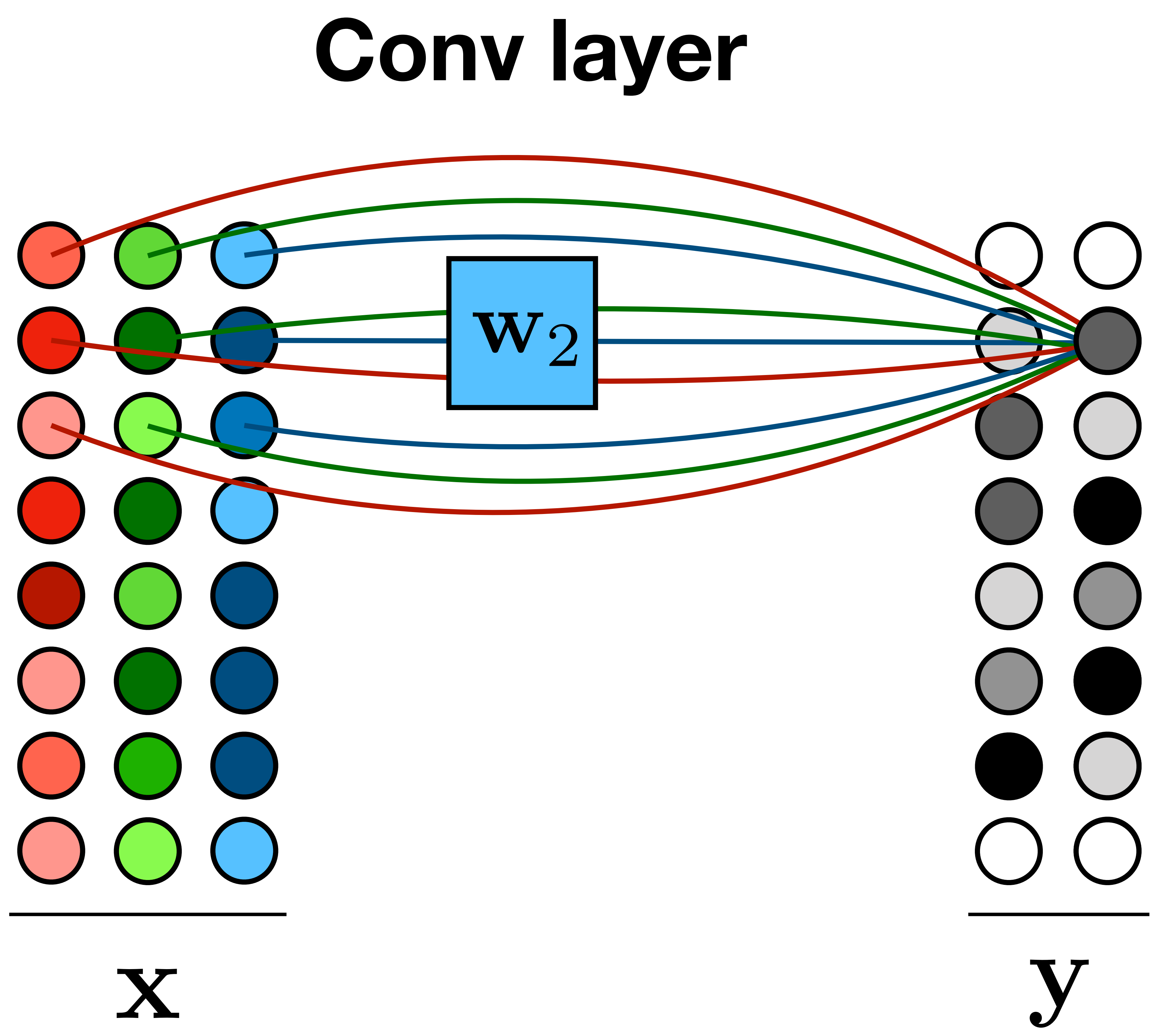
Multiple channel inputs



$$\mathbf{y} = \sum_c \mathbf{w}_c \circ \mathbf{x}_c$$

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

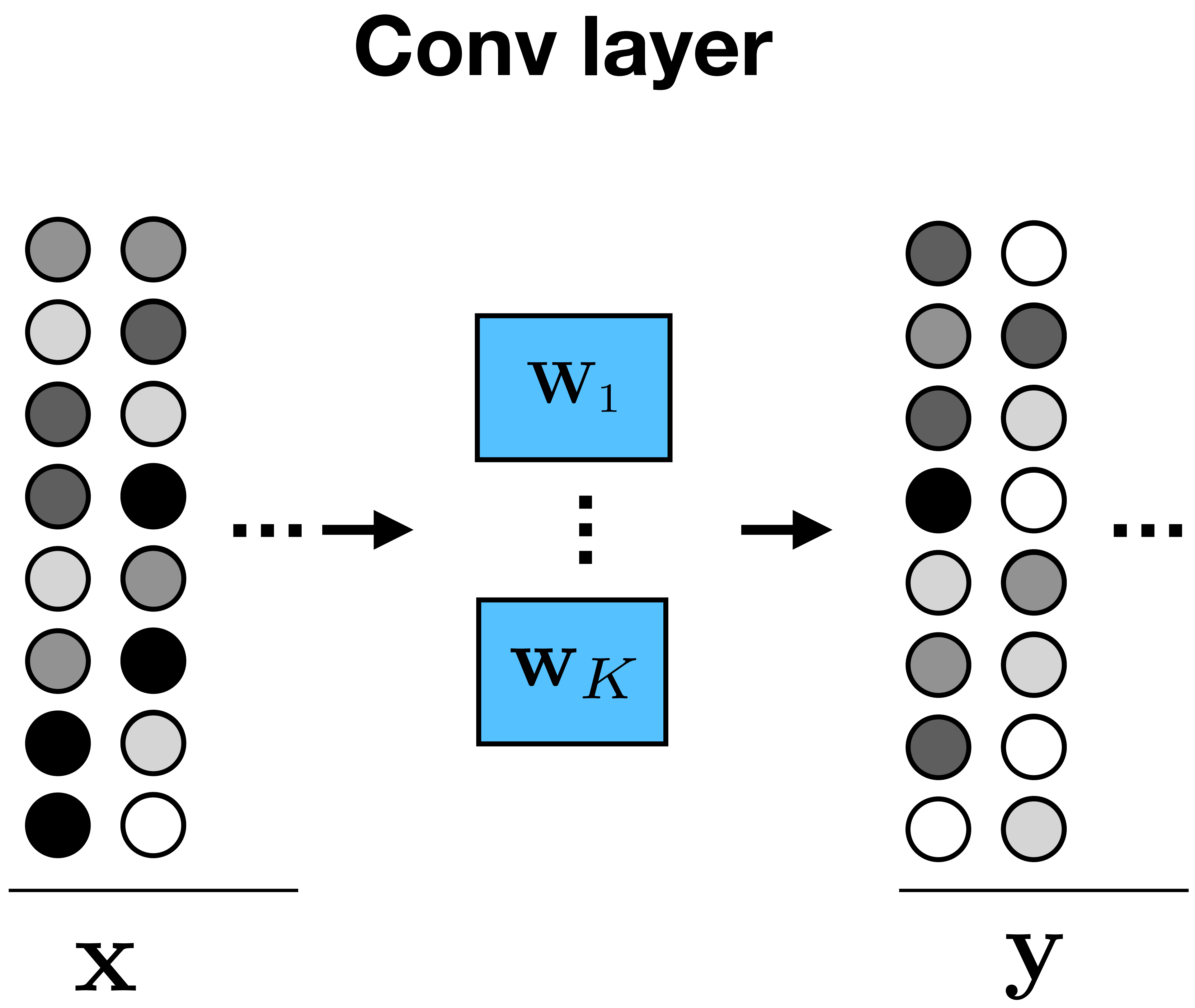
Multiple channel *outputs*



$$\mathbf{y}_k = \sum_c \mathbf{w}_{k_c} \circ \mathbf{x}_c$$

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

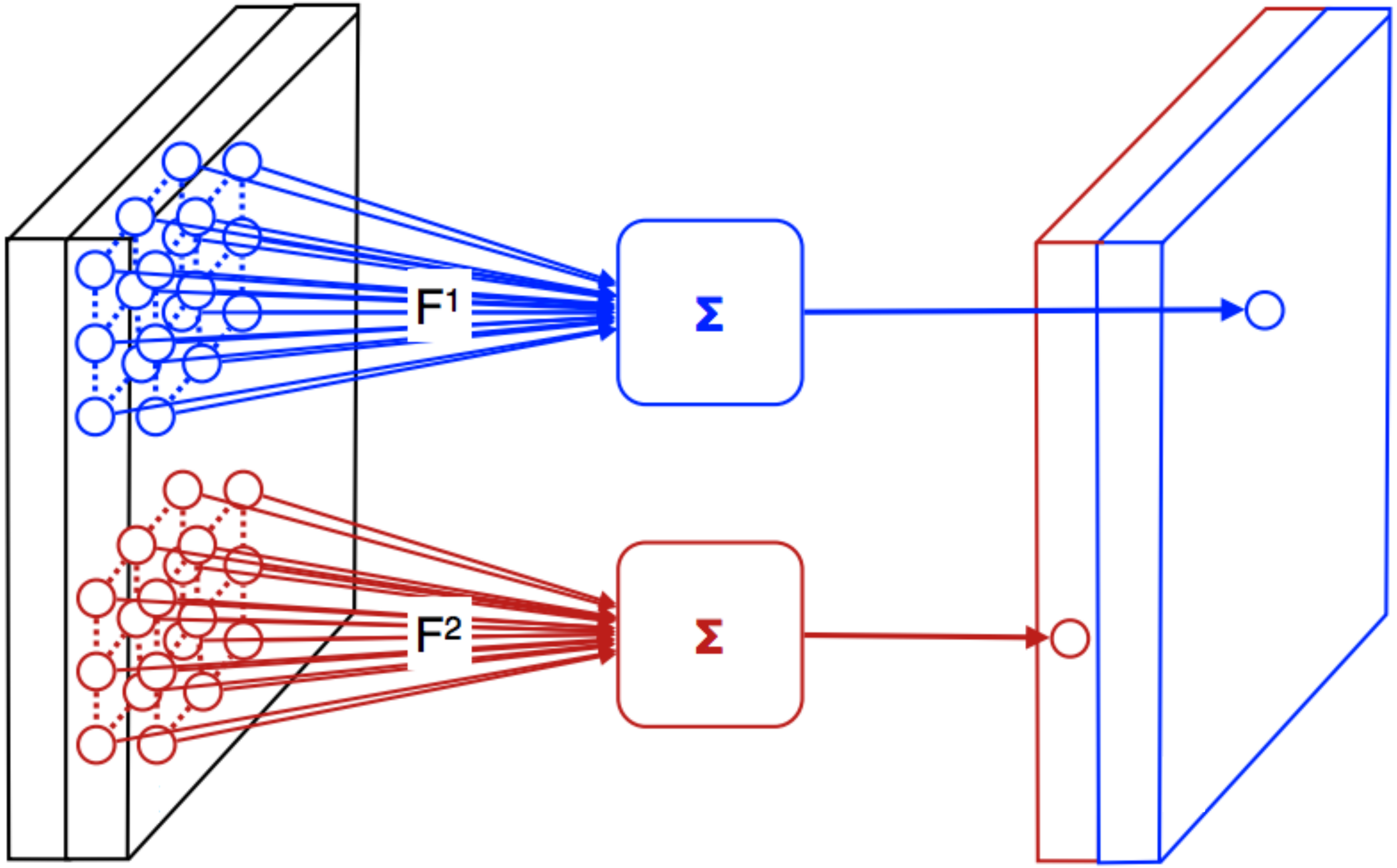
Multiple channels



$$\mathbf{y}_k = \sum_c \mathbf{w}_{k_c} \circ \mathbf{x}_c$$

$$\mathbb{R}^{N \times C} \rightarrow \mathbb{R}^{N \times K}$$

Input features A bank of 2 filters 2-dimensional output **feature maps**



$$\mathbf{x}_l \in \mathbb{R}^{H \times W \times C_l} \quad \longrightarrow \quad \mathbf{x}_{(l+1)} \in \mathbb{R}^{H \times W \times C_{(l+1)}}$$

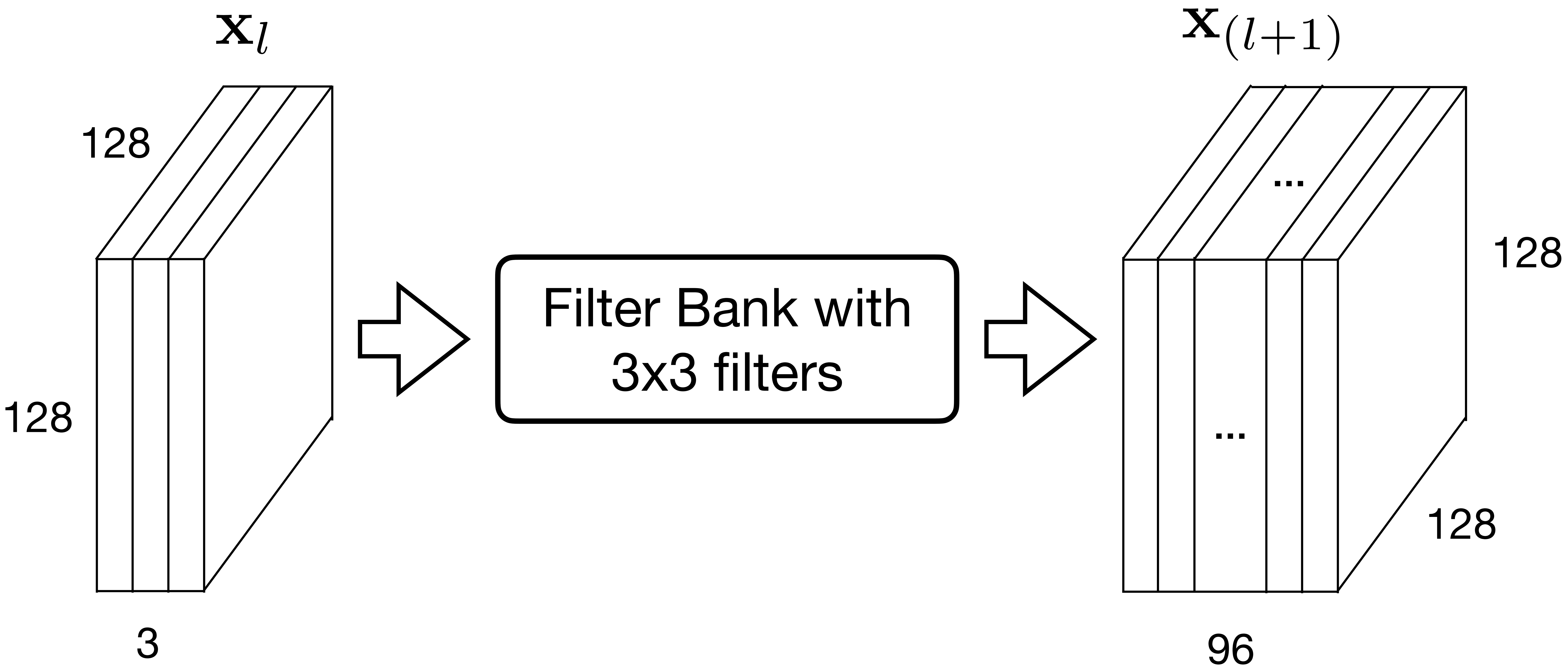
[Figure modified from Andrea Vedaldi]

Feature maps

- Each layer can be thought of as a set of C **feature maps** aka **channels**
- Each feature map is an NxM image

Image source: https://stackoverflow.com/questions/45678473/convolution-neural-networks-all-feature-maps-are-blackpixel-value-is-0

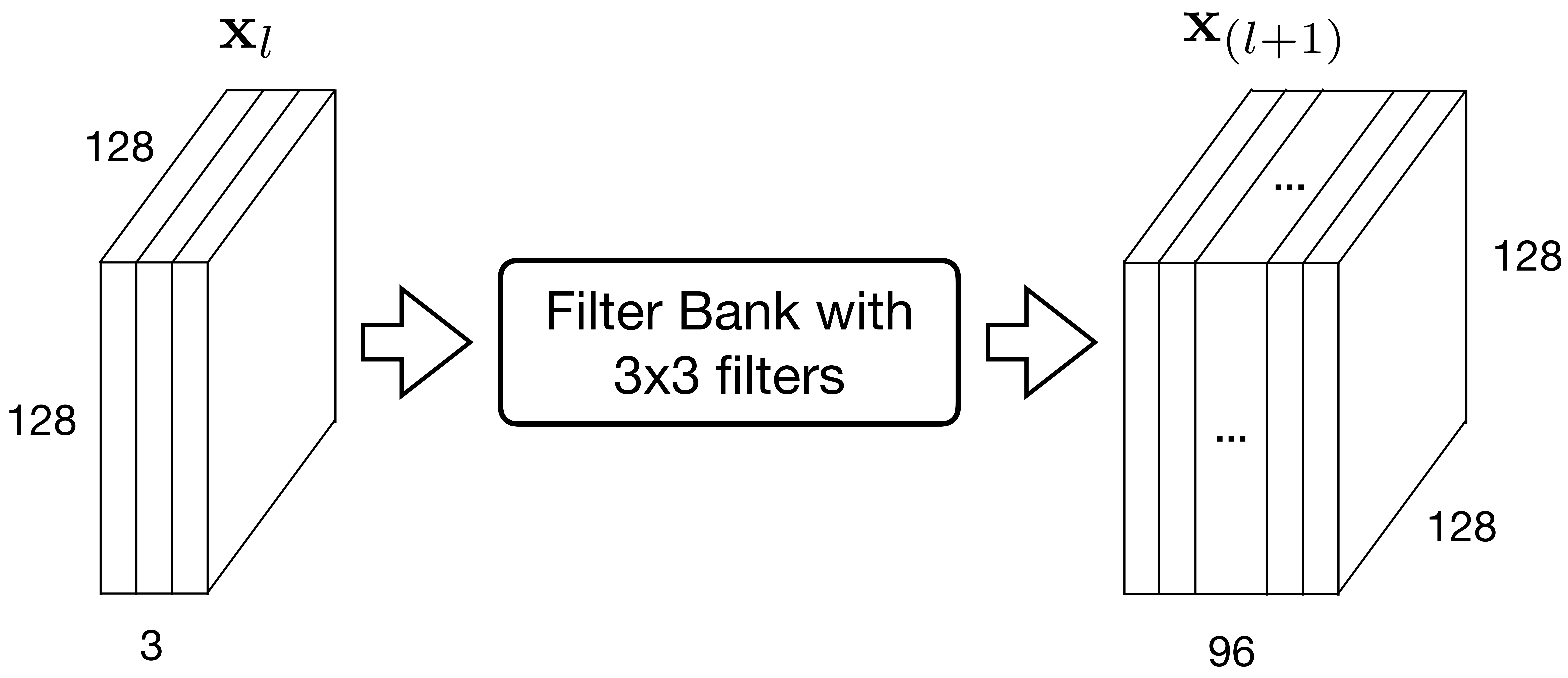
Multiple channels: Example



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

$$\mathbf{x}_l \in \mathbb{R}^{H \times W \times C_l} \quad \rightarrow \quad \mathbf{x}_{(l+1)} \in \mathbb{R}^{H \times W \times C_{(l+1)}}$$

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

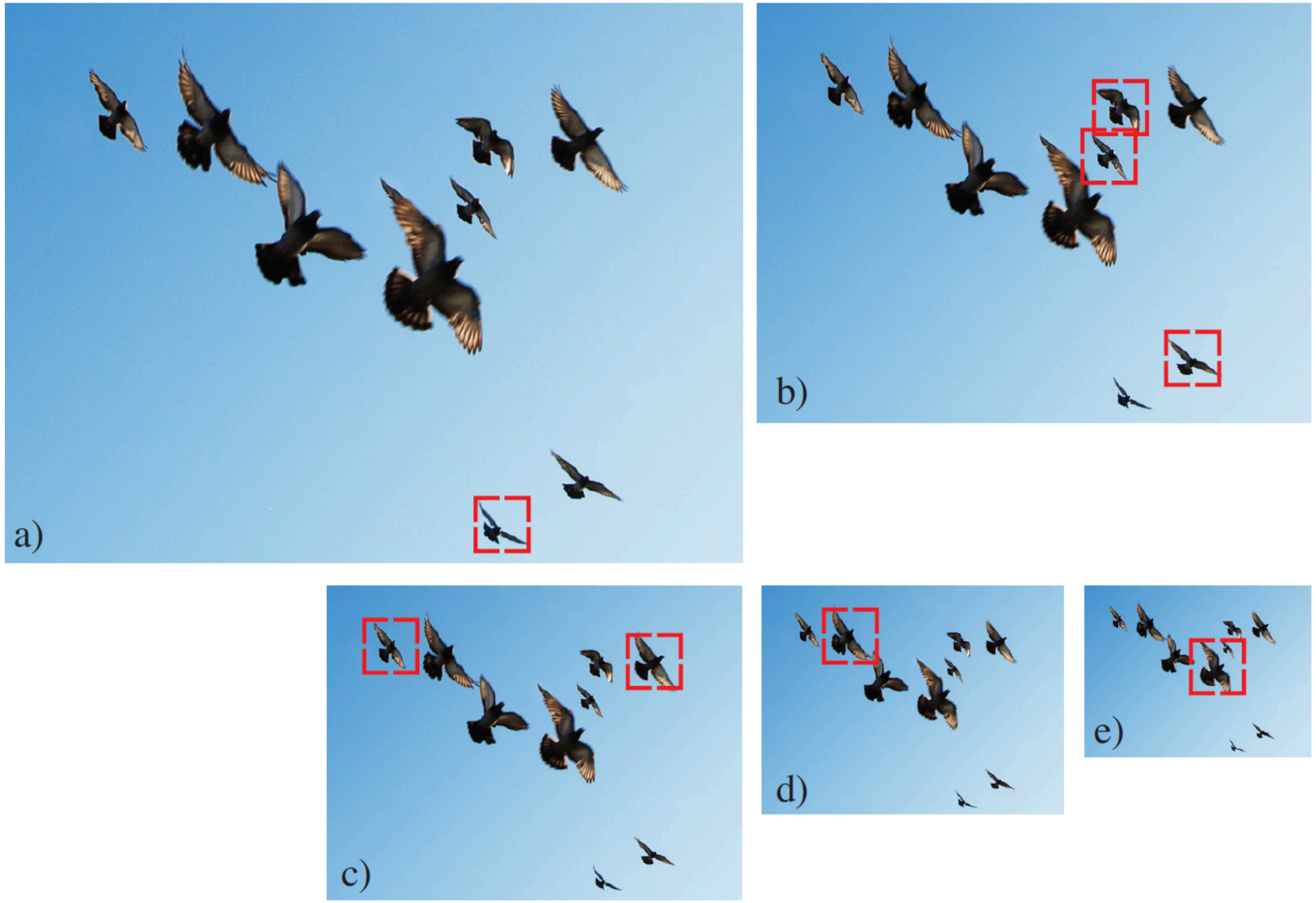
Number of parameters per filter: $M \times N \times C_l$

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

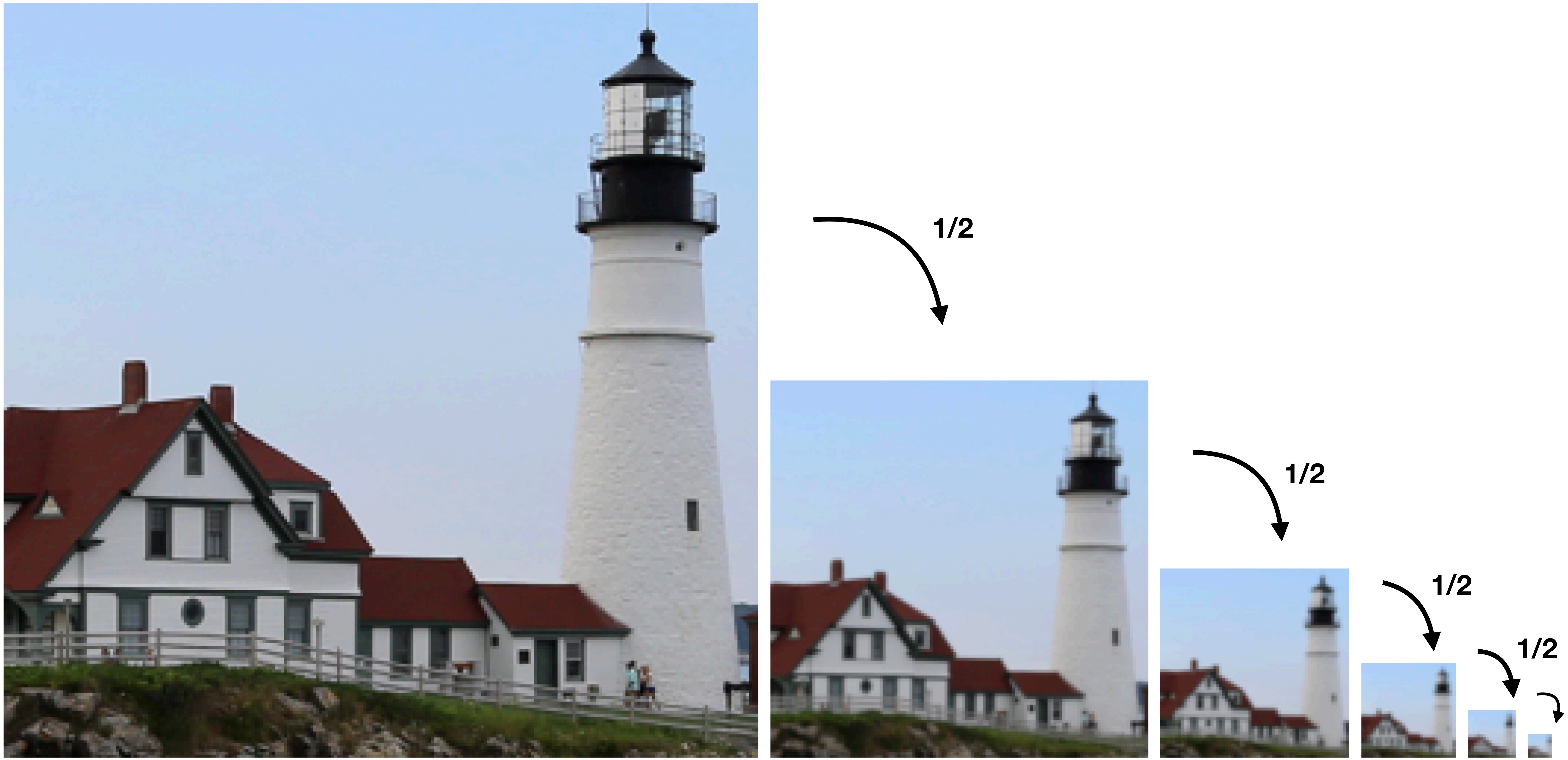
Pooling and downsampling

We need translation and **scale** invariance

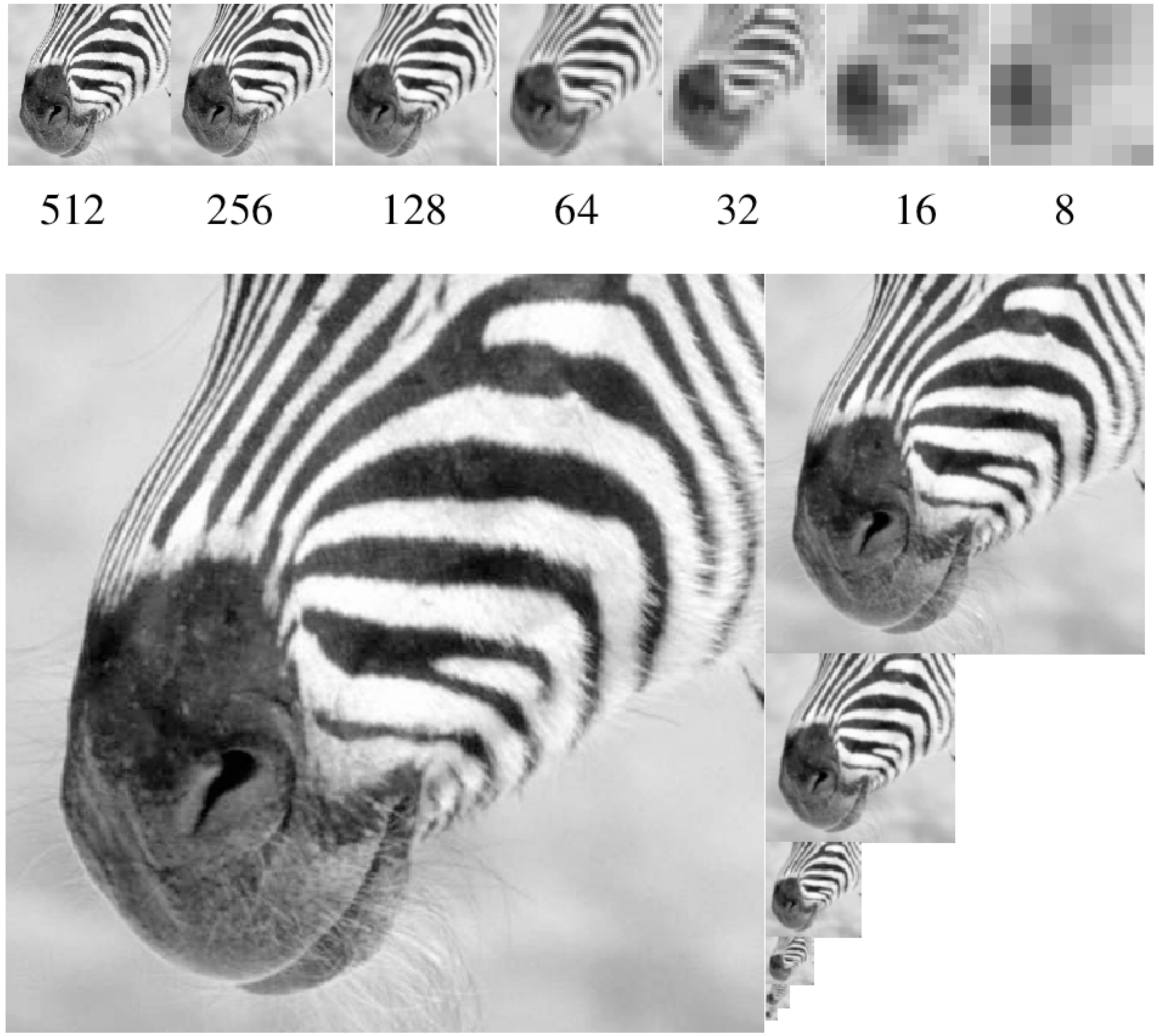
Image pyramids



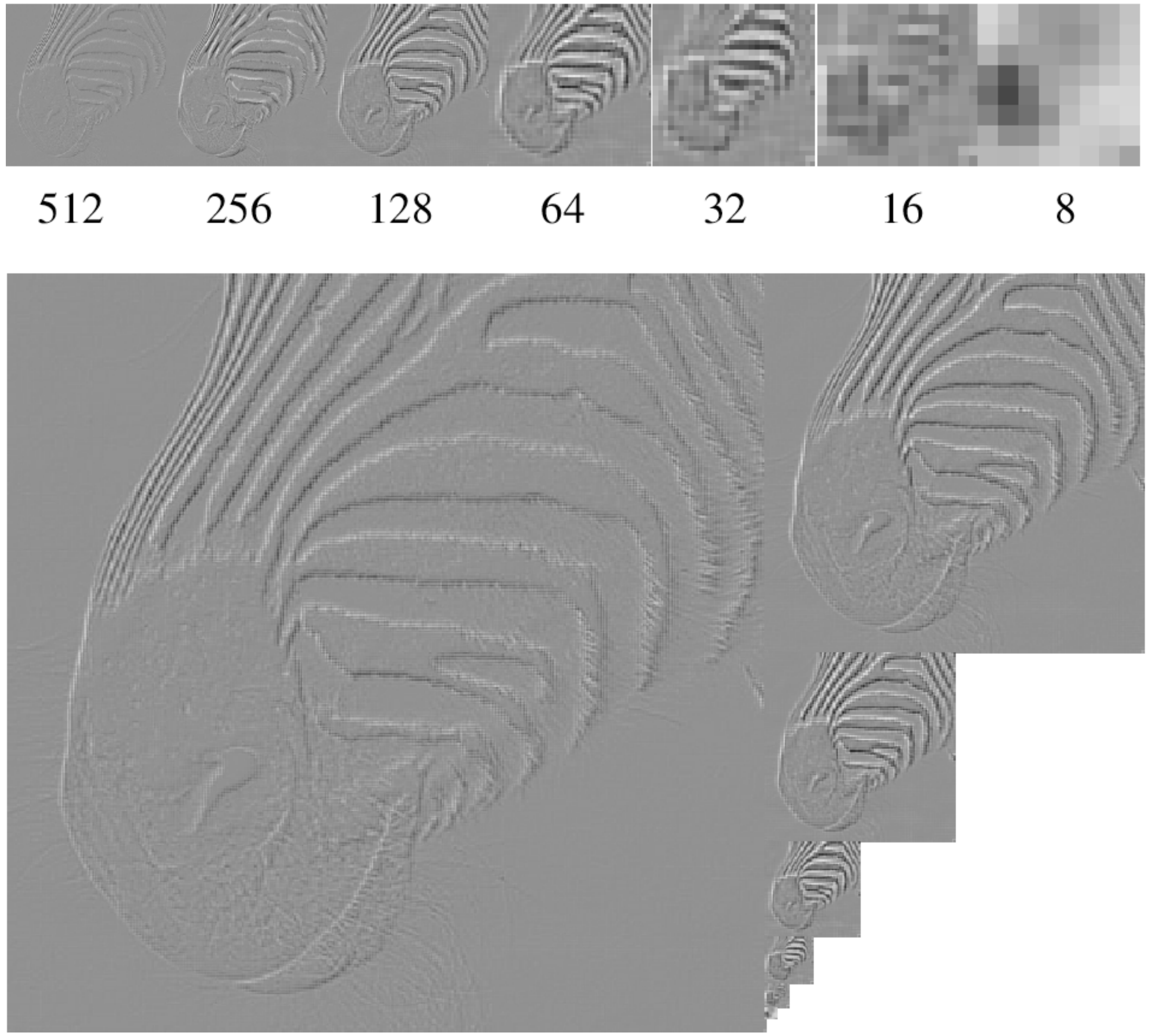
Gaussian Pyramid



Multiscale representations are great!



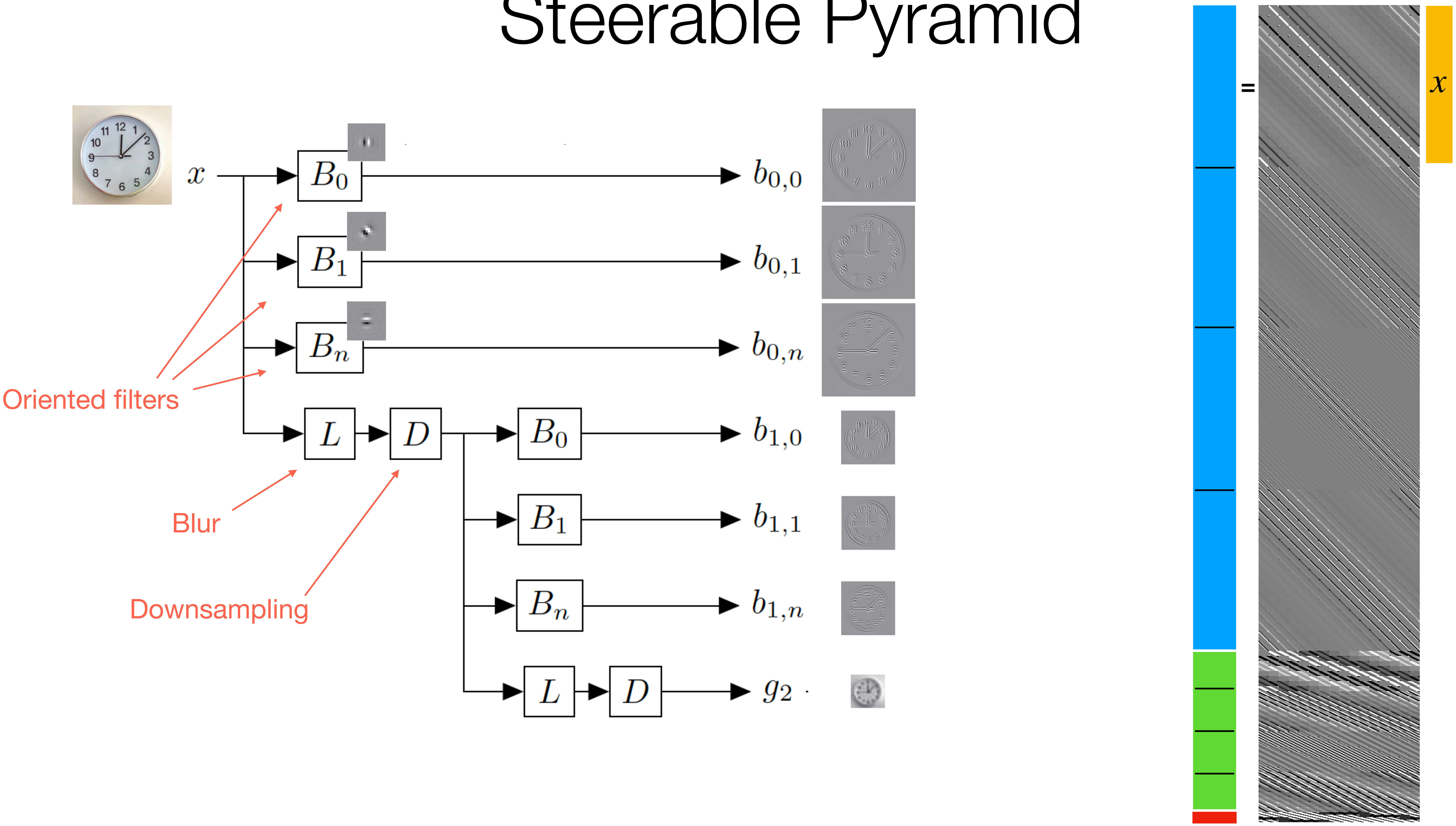
Gaussian Pyr



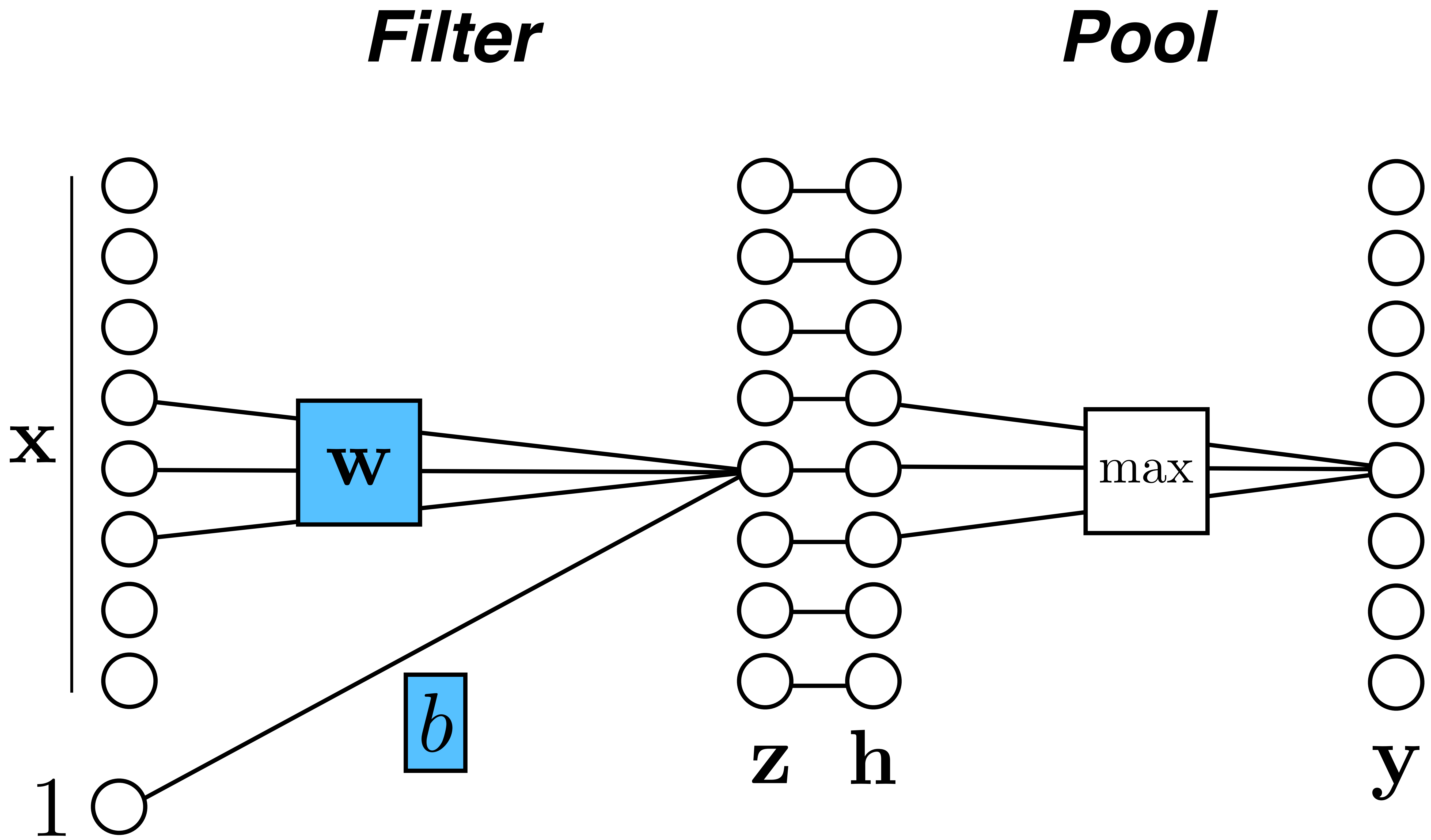
Laplacian Pyr

How can we use multi-scale modeling in Convnets?

Steerable Pyramid



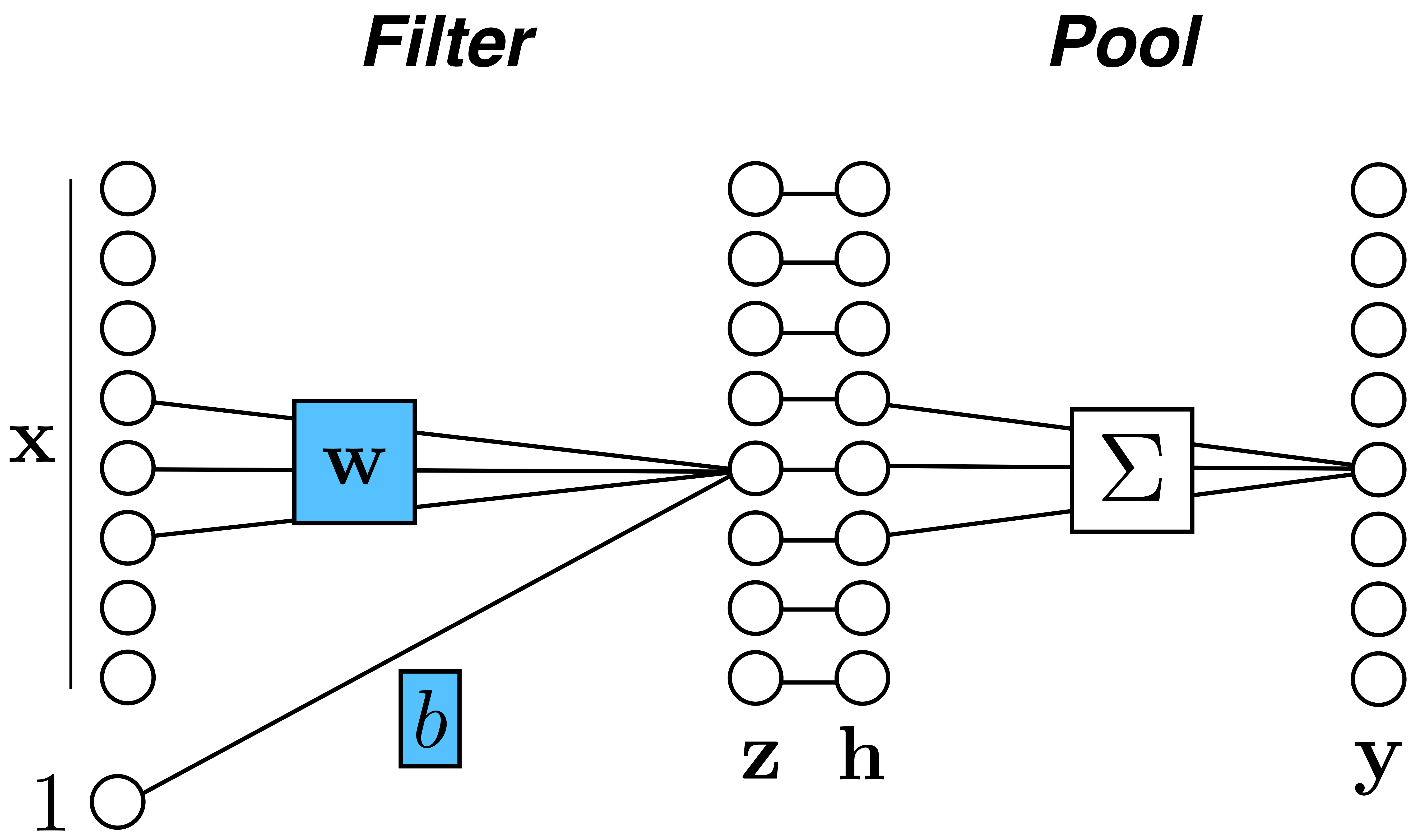
Pooling



Max pooling

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

Pooling



Max pooling

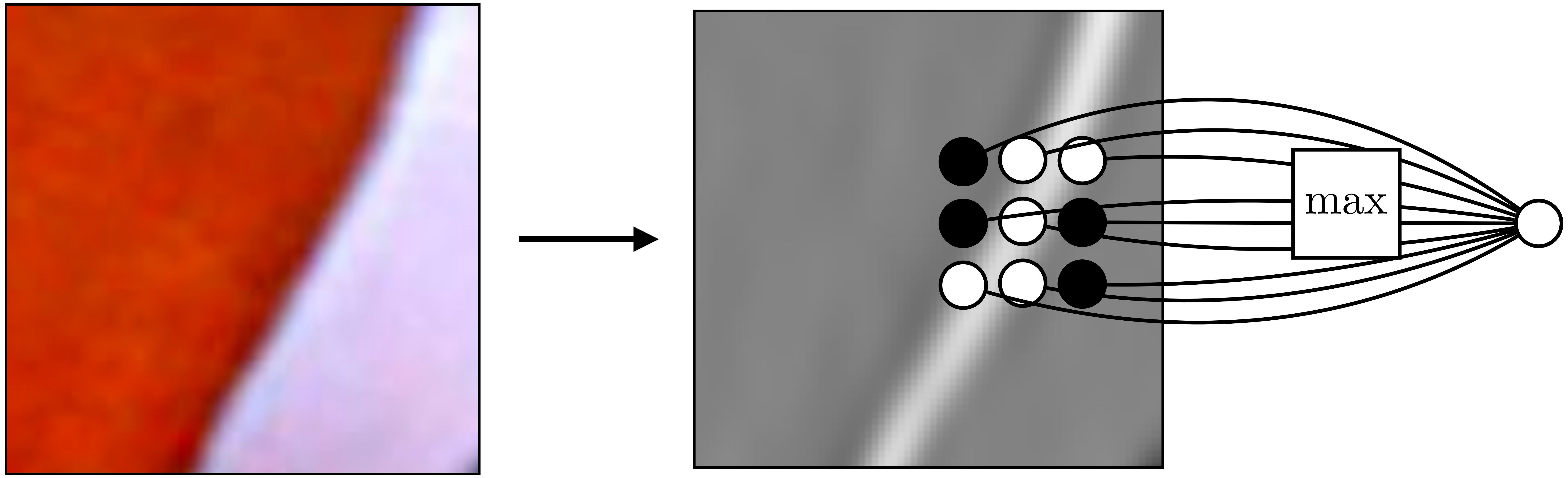
$$y_j = \max_{j \in \mathcal{N}(j)} h_j$$

Mean pooling

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

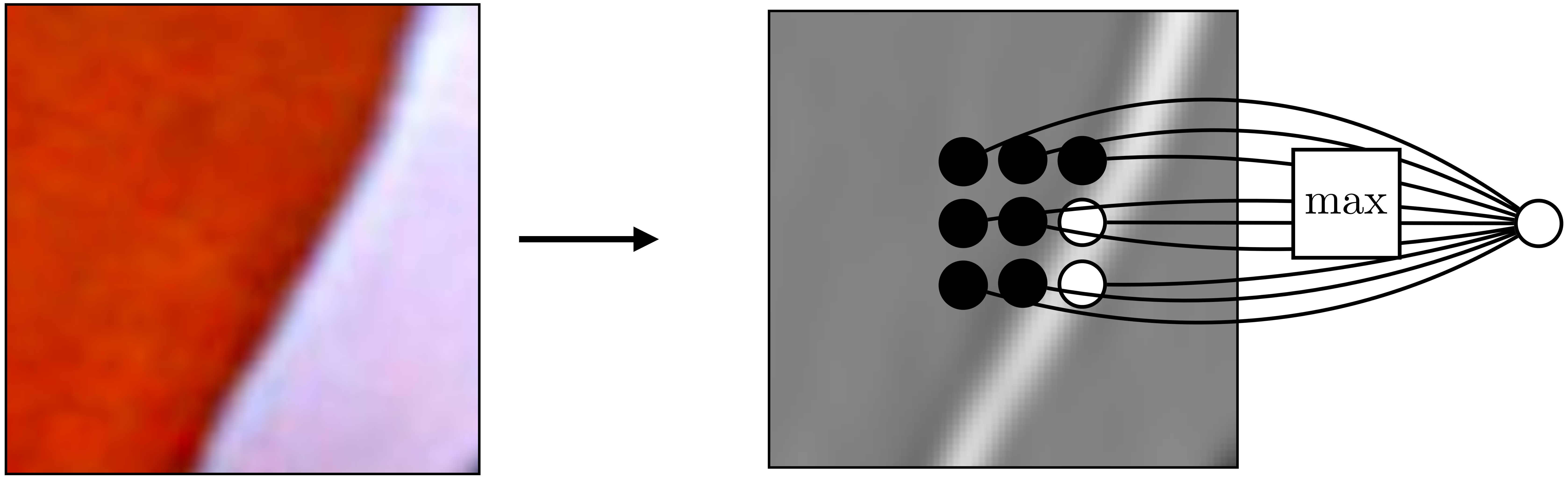
Pooling — Why?

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



Pooling — Why?

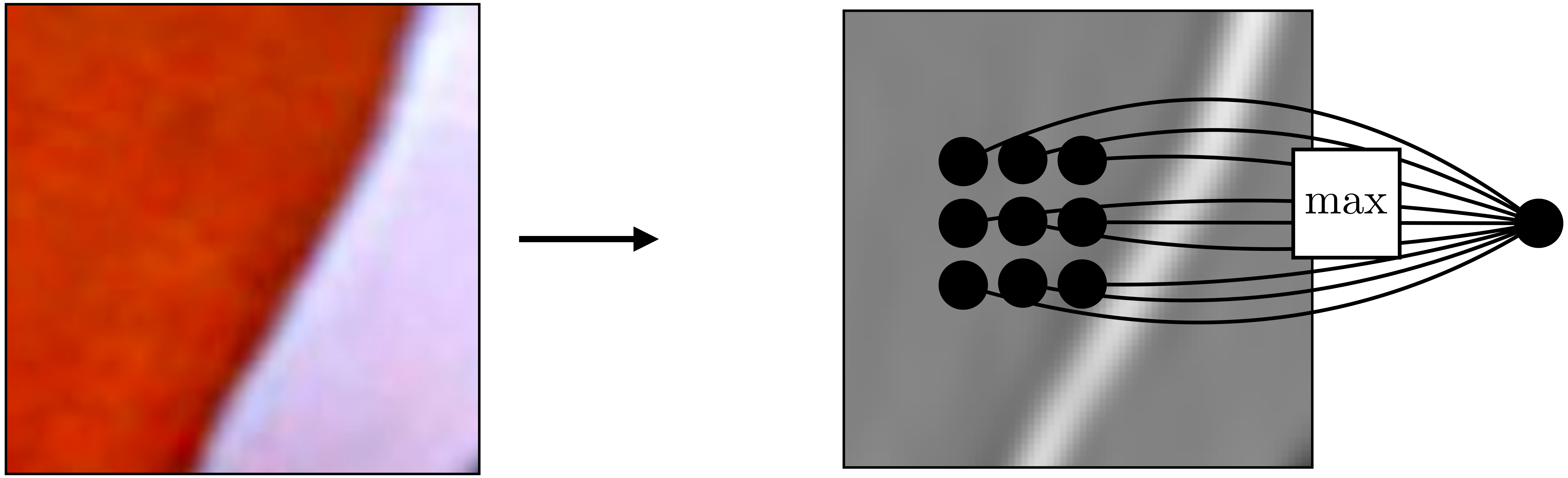
Pooling across spatial locations achieves stability w.r.t. small translations:



large response regardless of exact position of edge

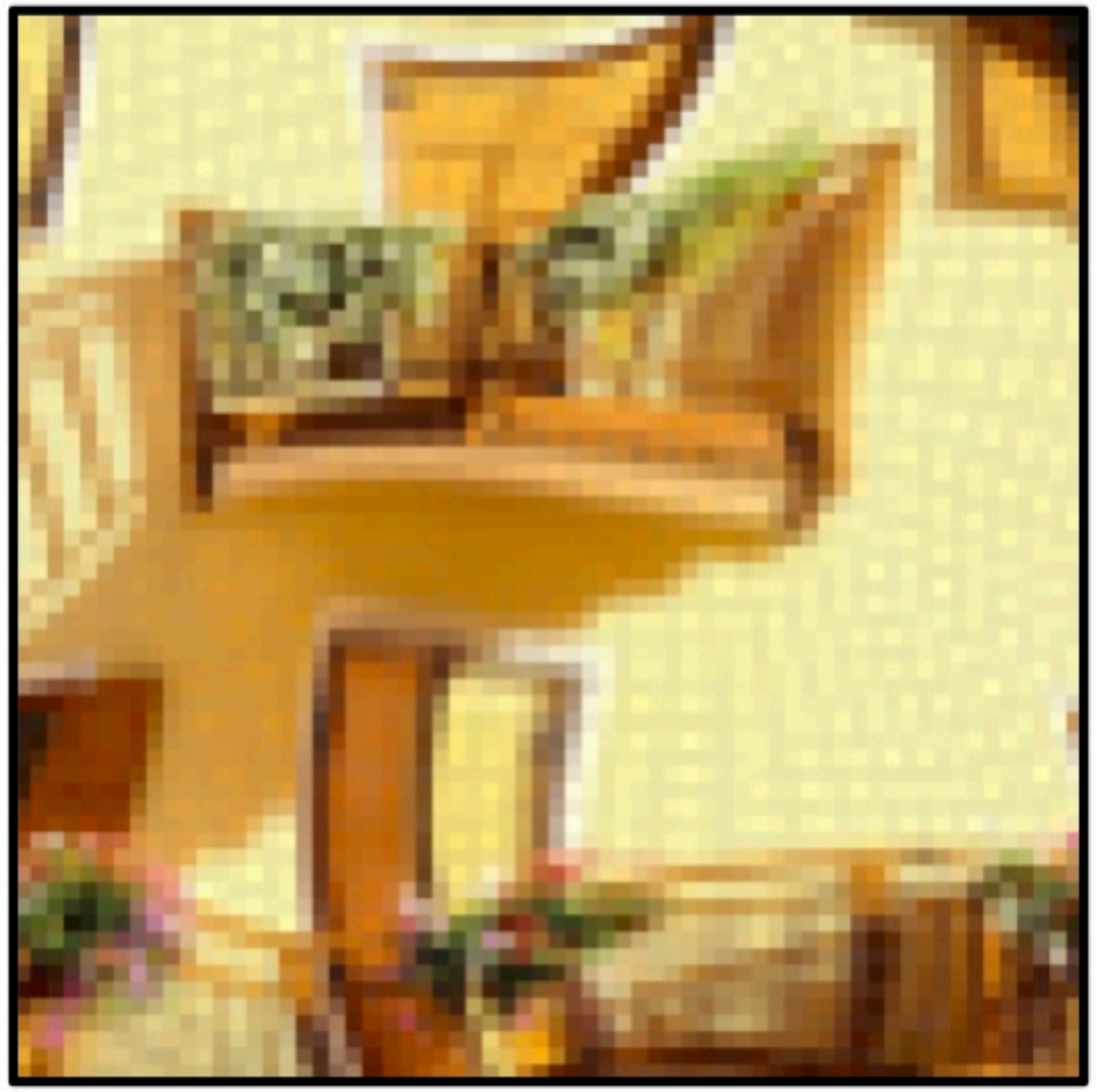
Pooling — Why?

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



CNNs are stable w.r.t. diffeomorphisms

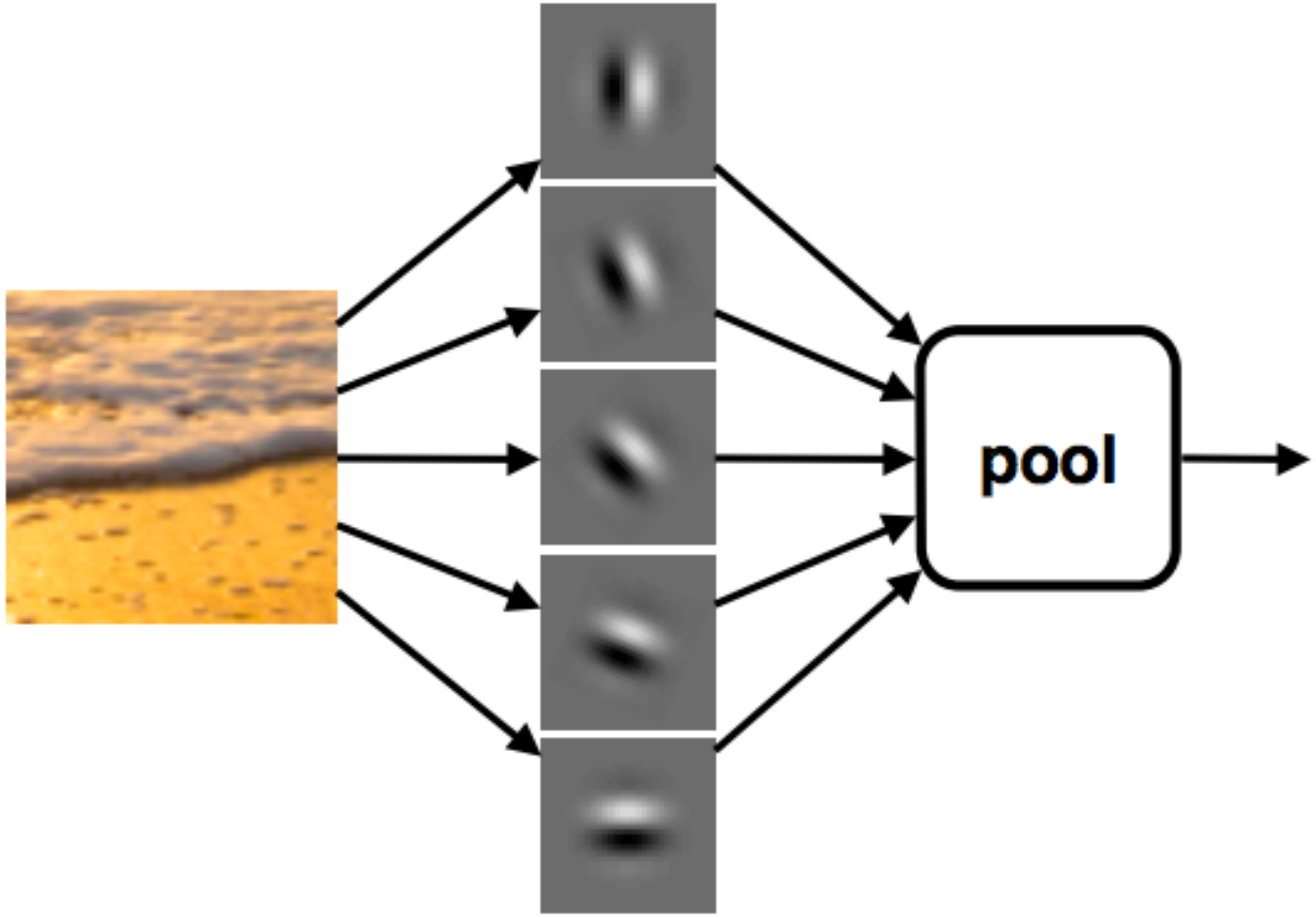
$\approx$



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

Pooling *across channels* — Why?

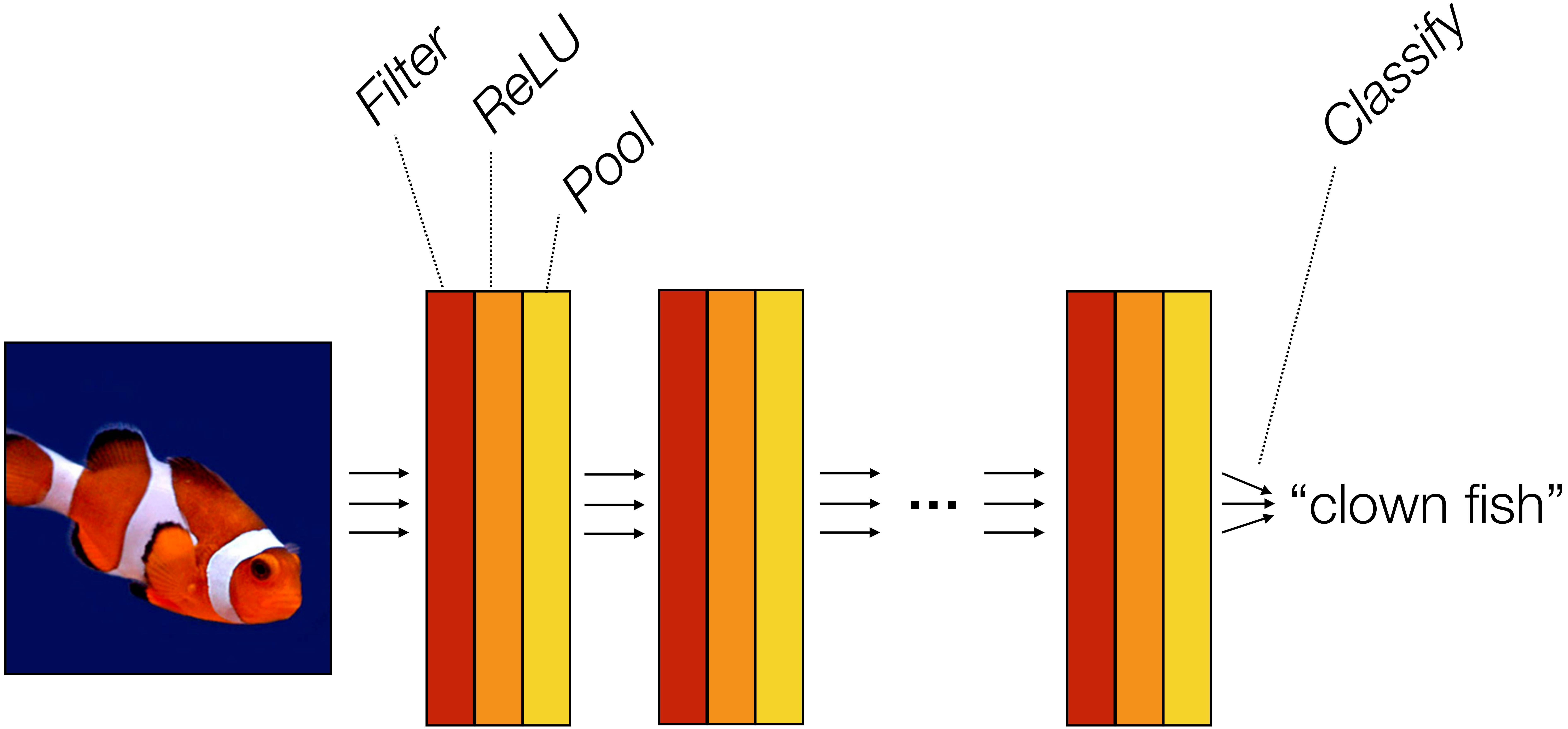
Pooling across feature channels (filter outputs) can achieve other kinds of invariances:



large response for any edge, regardless of its orientation

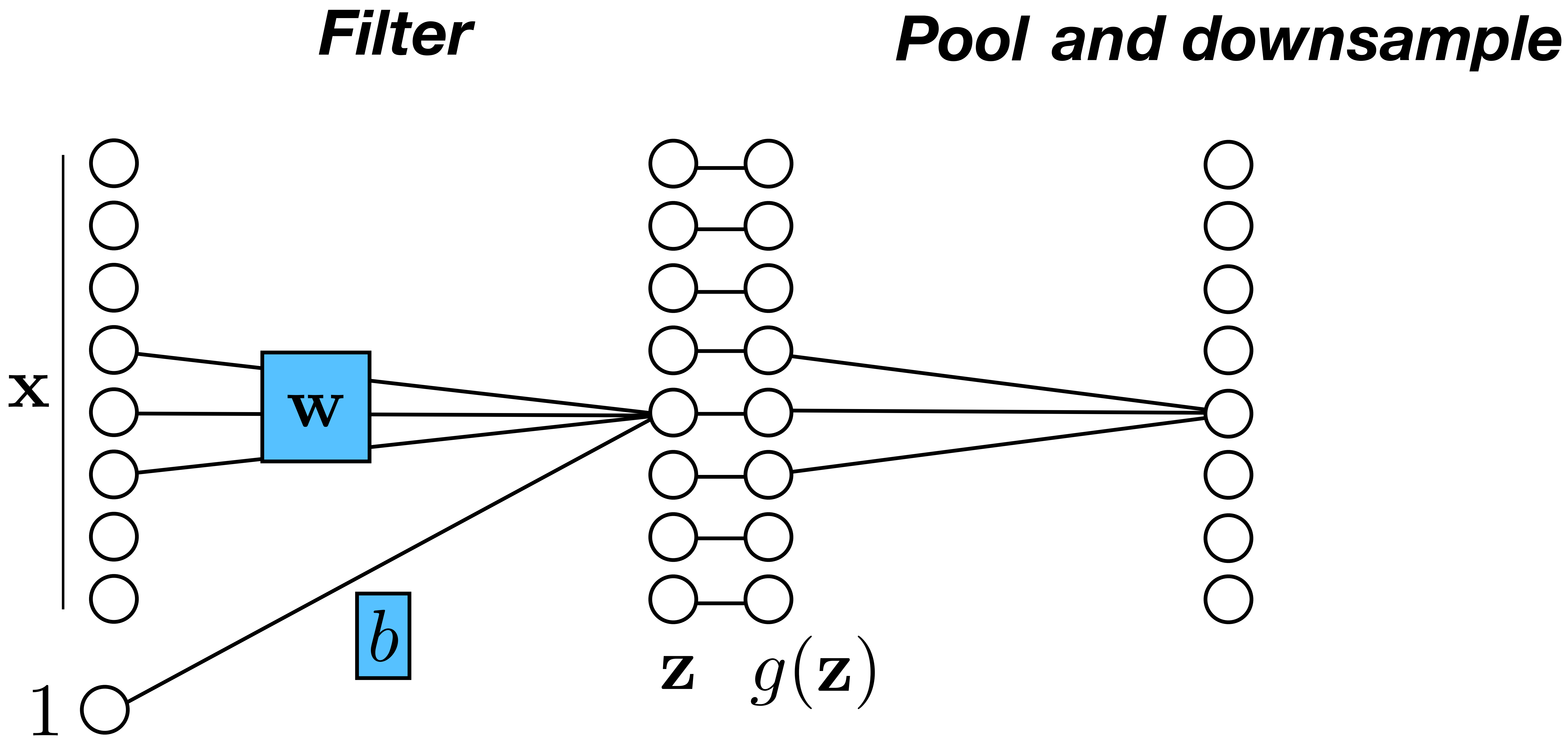
[Derived from slide by Andrea Vedaldi]

Computation in a neural net

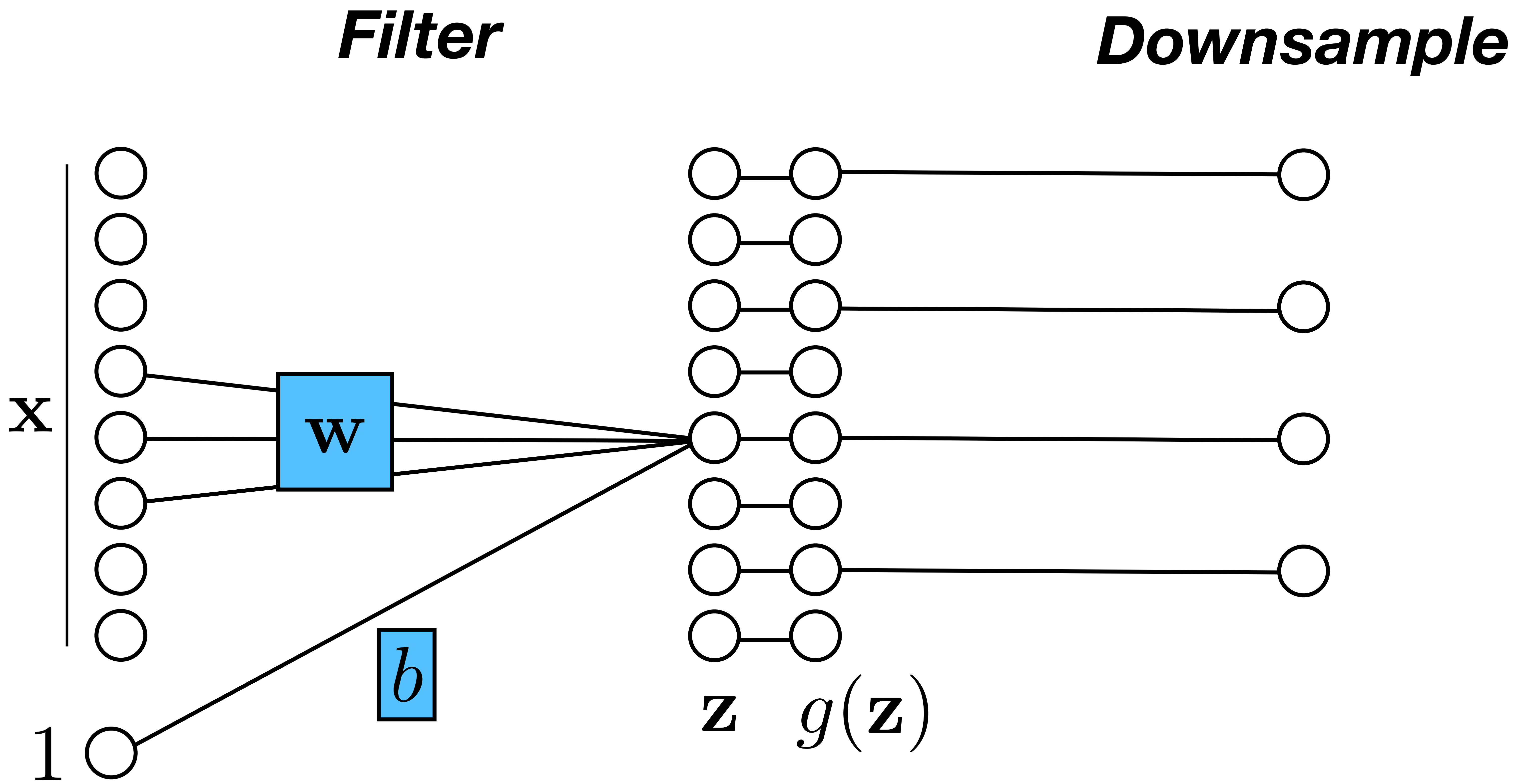


$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

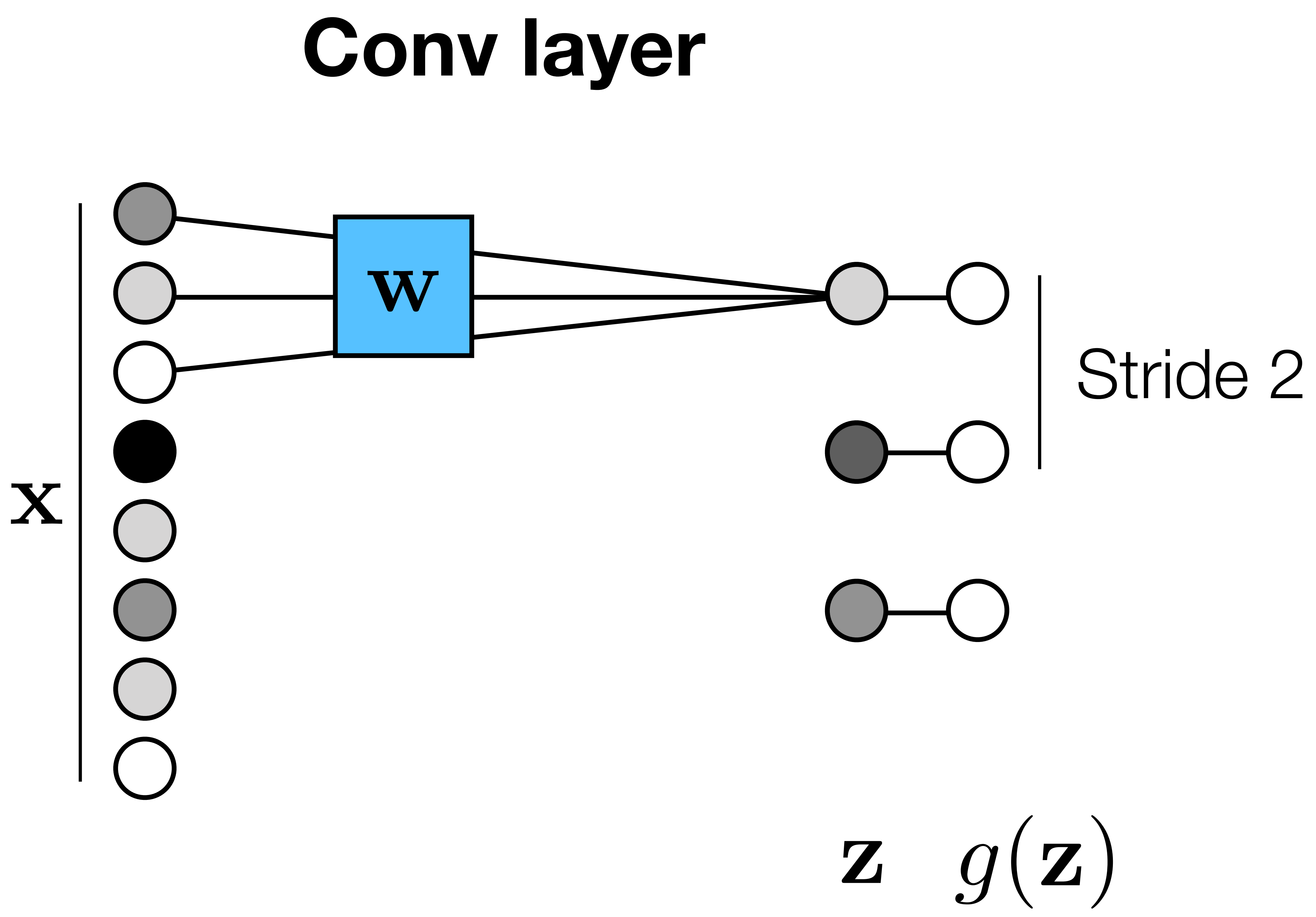
Downsampling



Downsampling

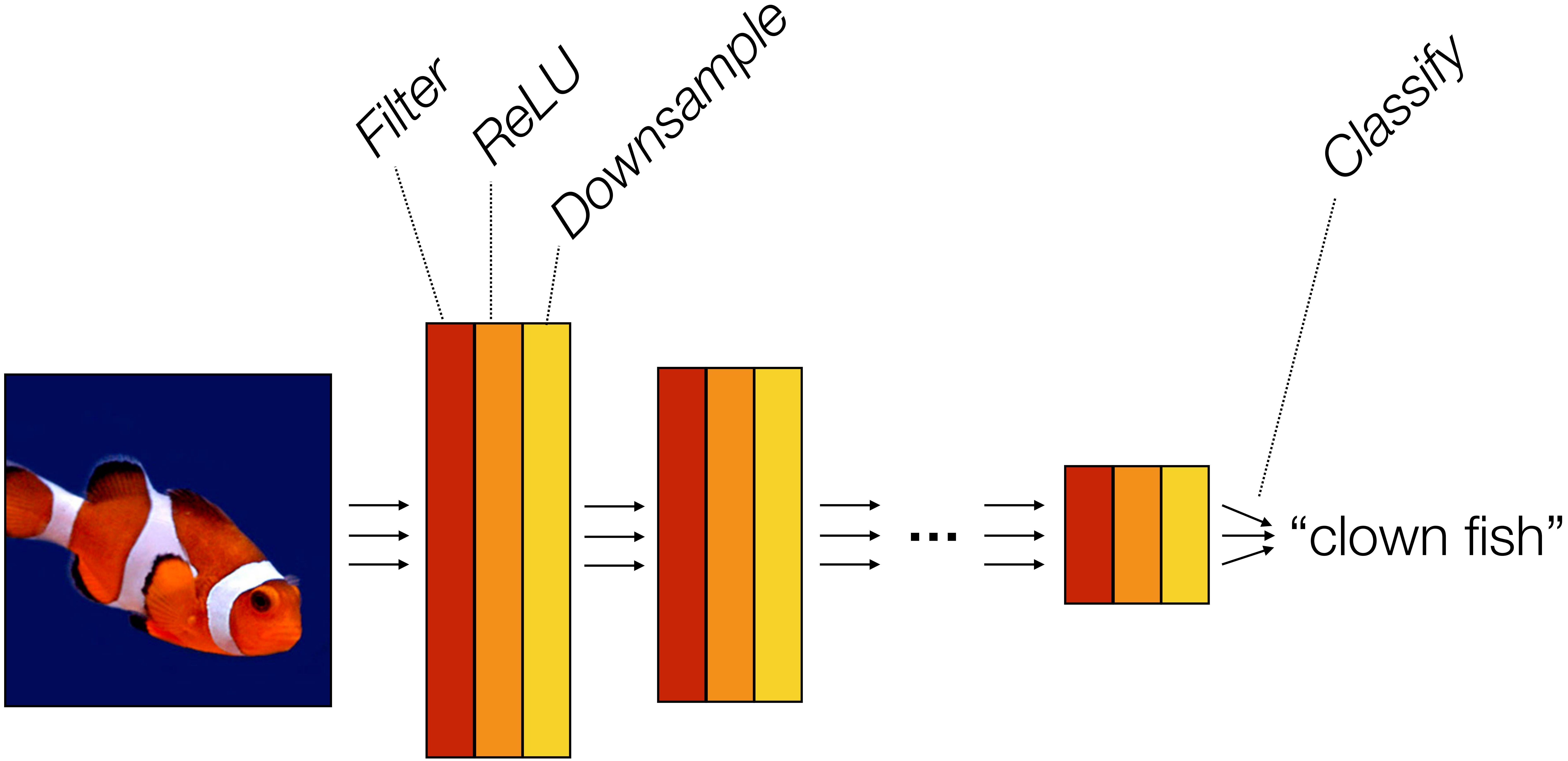


Strided operations



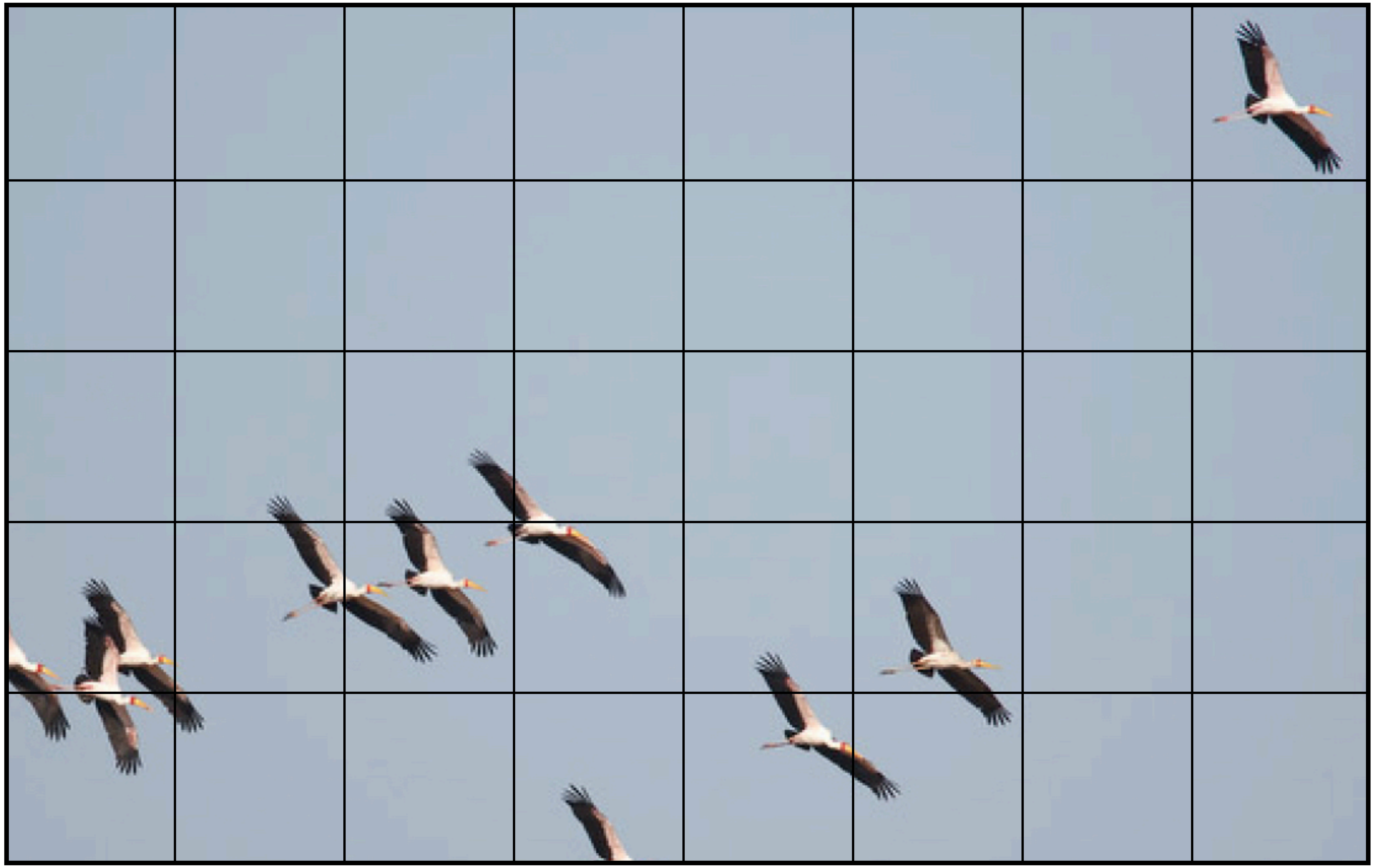
Strided operations combine a given operation (convolution or pooling) and downsampling into a single operation.

Computation in a neural net

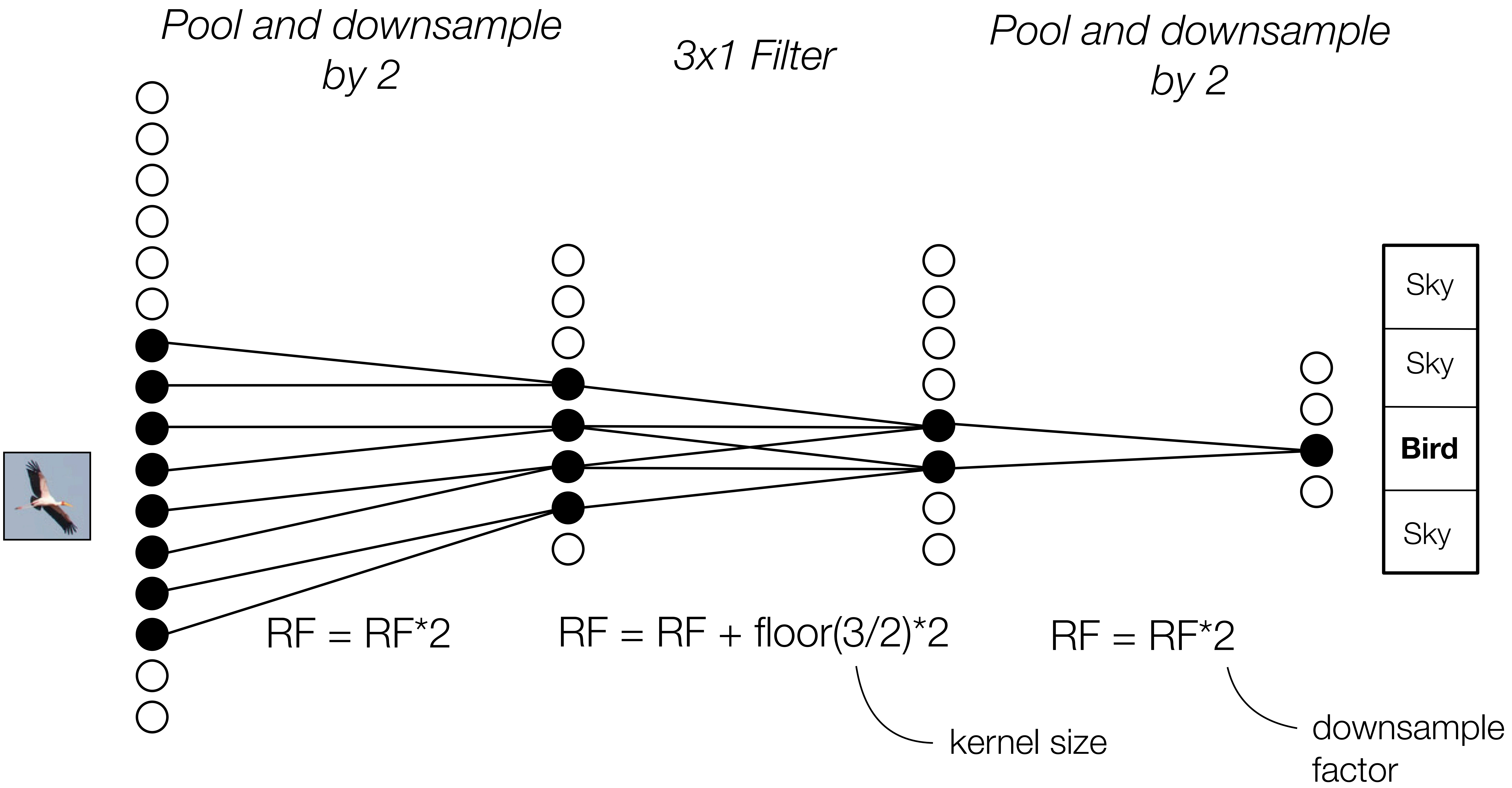


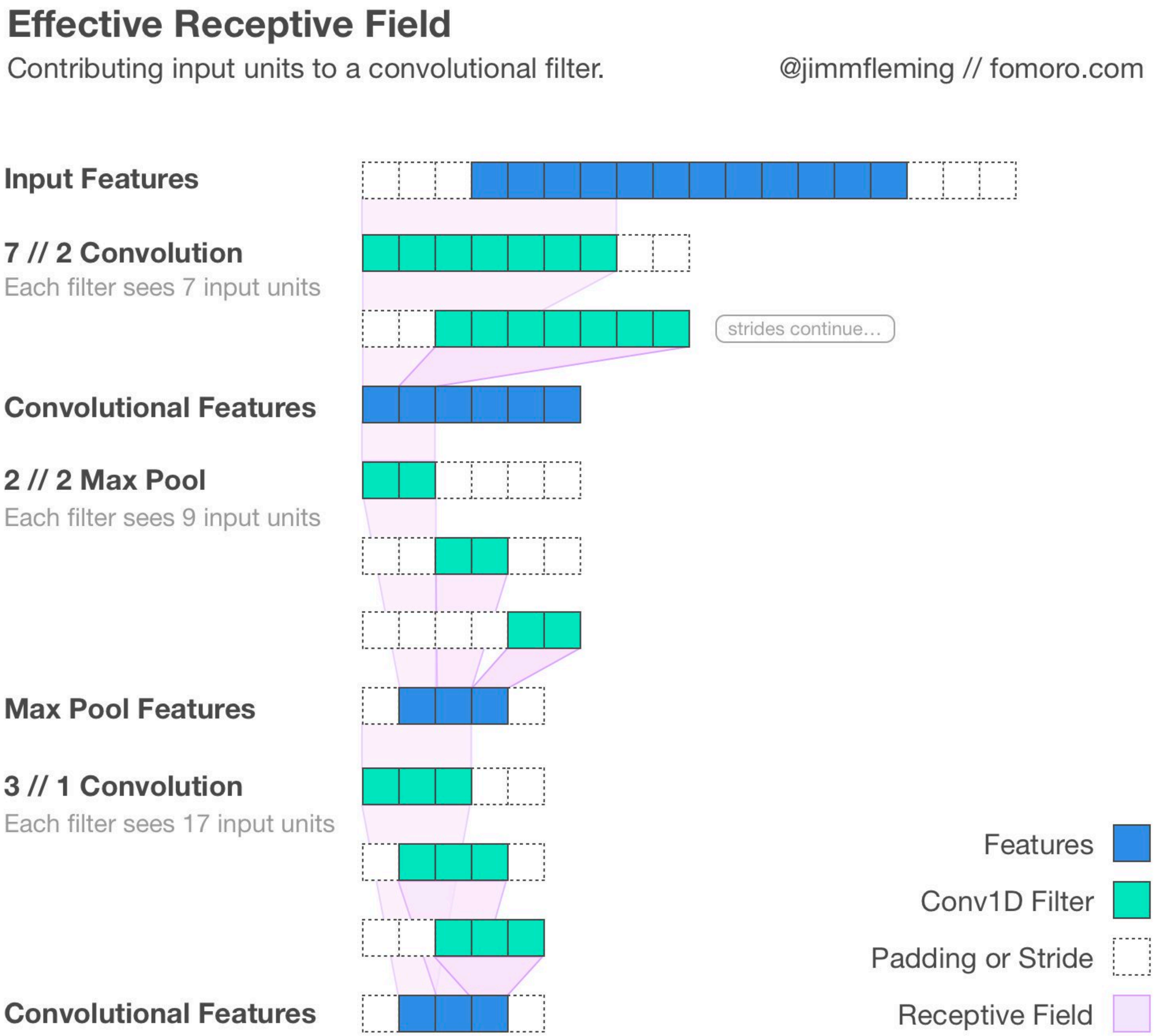
$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

Receptive fields



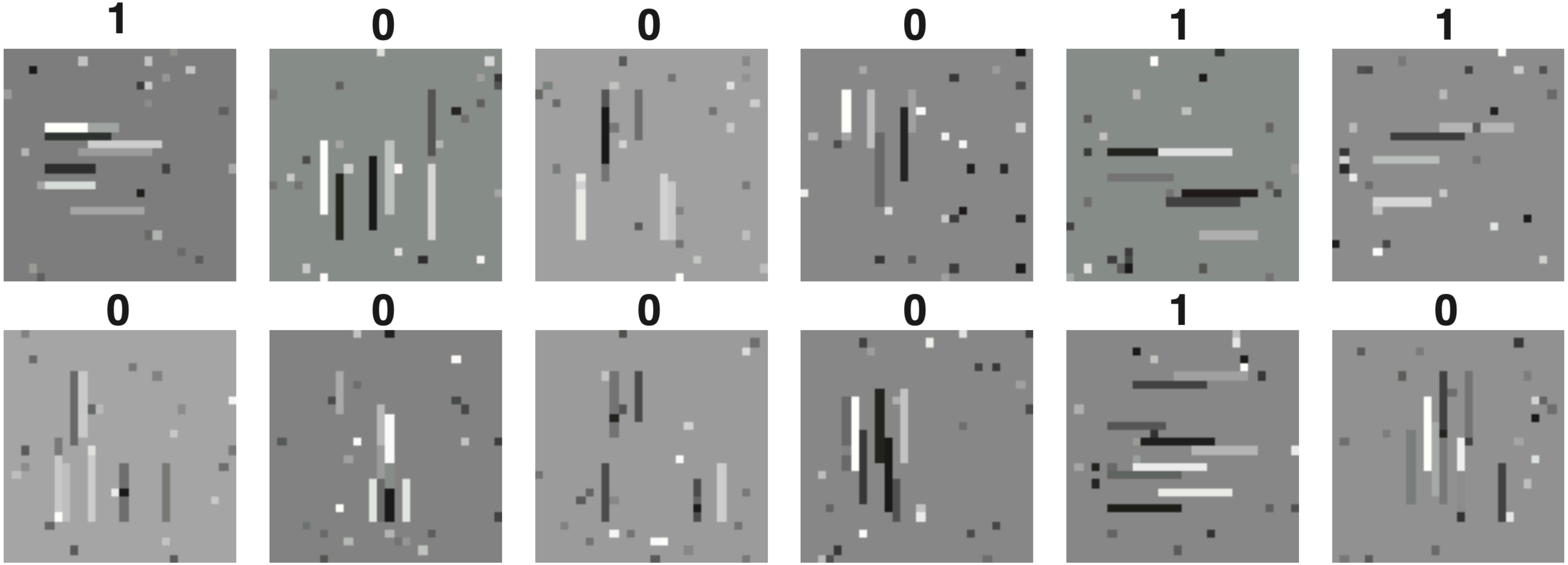
Receptive fields





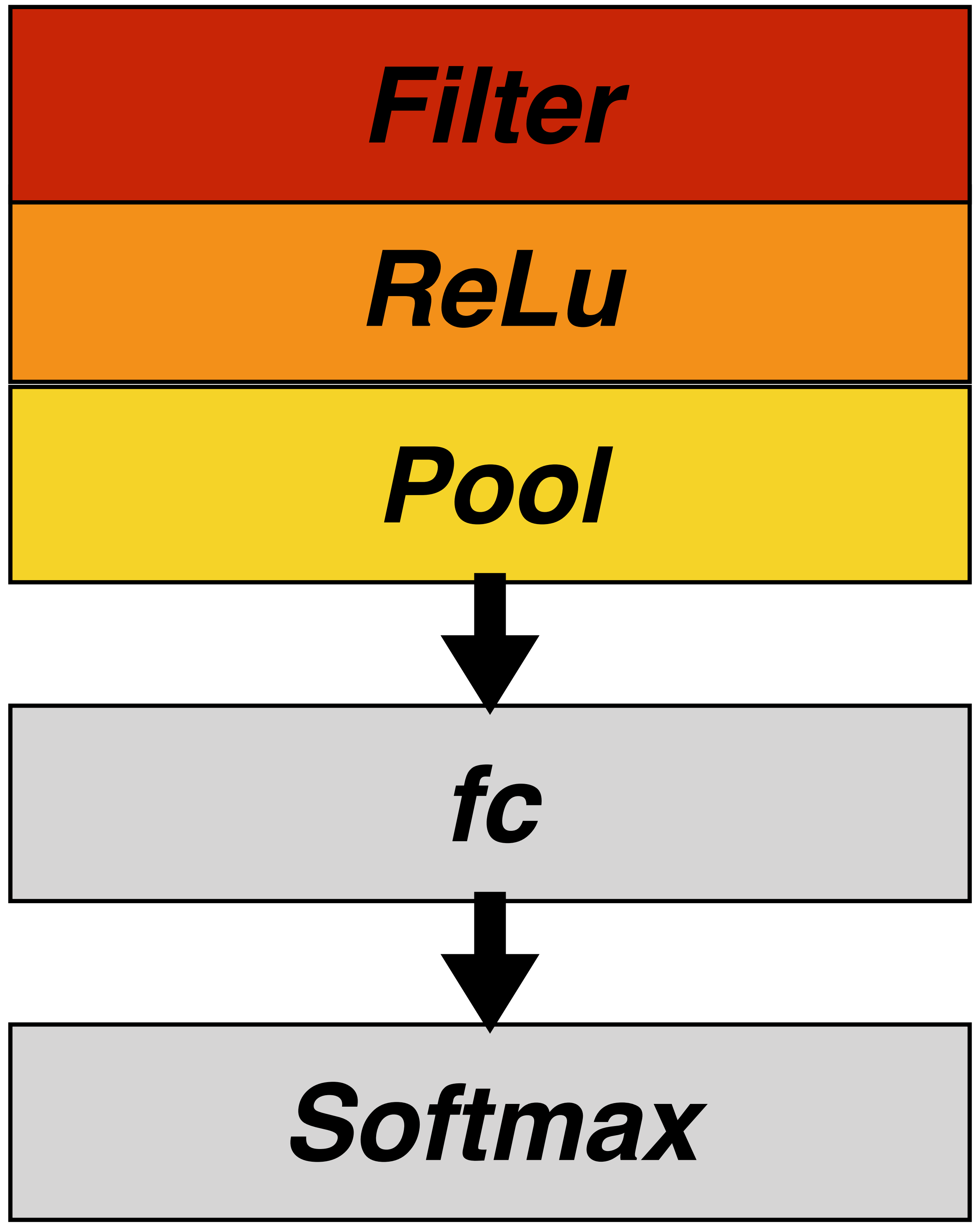
[http://fomoro.com/tools/receptive-fields/index.html]

Example: simple line classification



Class 0: vertical lines
Class 1: horizontal lines

Example: simple line classification



$$\mathbf{z}_{1_i} = \mathbf{w}_i \circ \mathbf{x} + \mathbf{b}_i$$

with 2 learned kernels $\mathbf{w}_1, \mathbf{w}_2$

$$\mathbf{h}_i = \max(\mathbf{z}_{1_i}, 0)$$

$$\mathbf{z}_{2_i} = \frac{1}{NM} \sum_{n,m} \mathbf{h}_i[n, m]$$

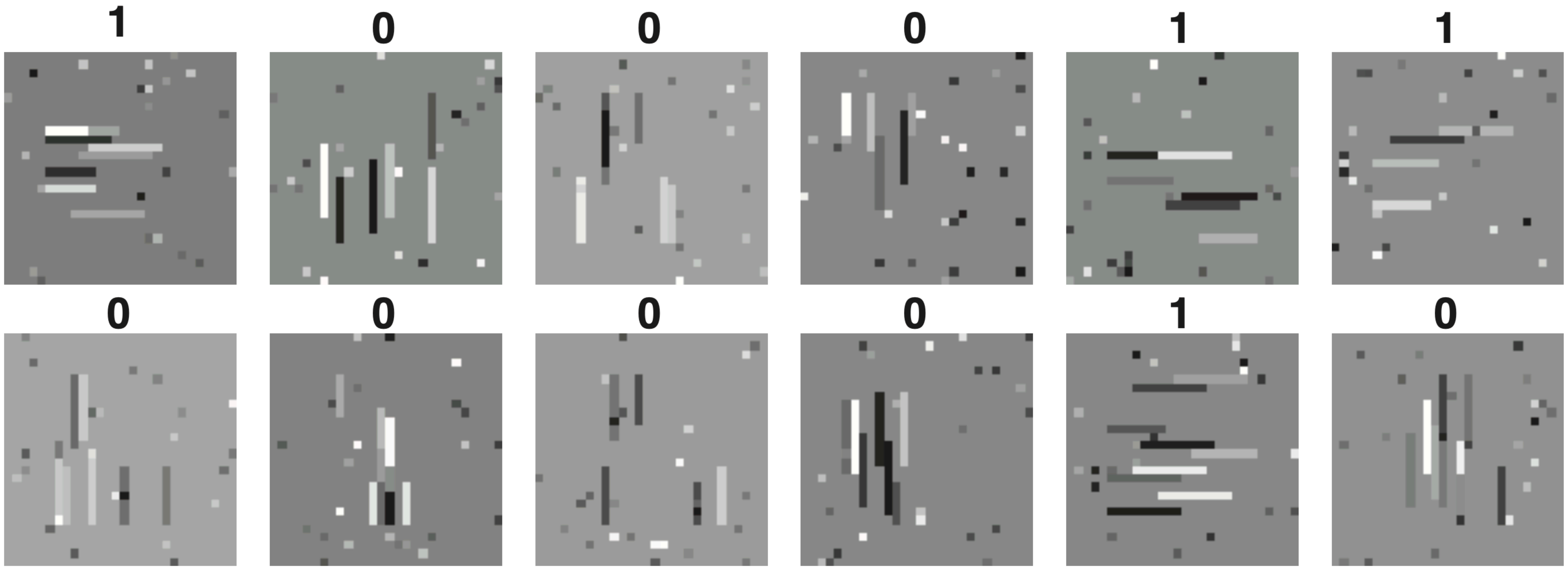
$$\mathbf{z}_3 = \mathbf{W}\mathbf{z}_2 + \mathbf{c}$$

$$\mathbf{y}_i = \frac{e^{-\tau \mathbf{z}_{3_i}}}{\sum_{k=1}^{K} e^{-\tau \mathbf{z}_{3_k}}}$$

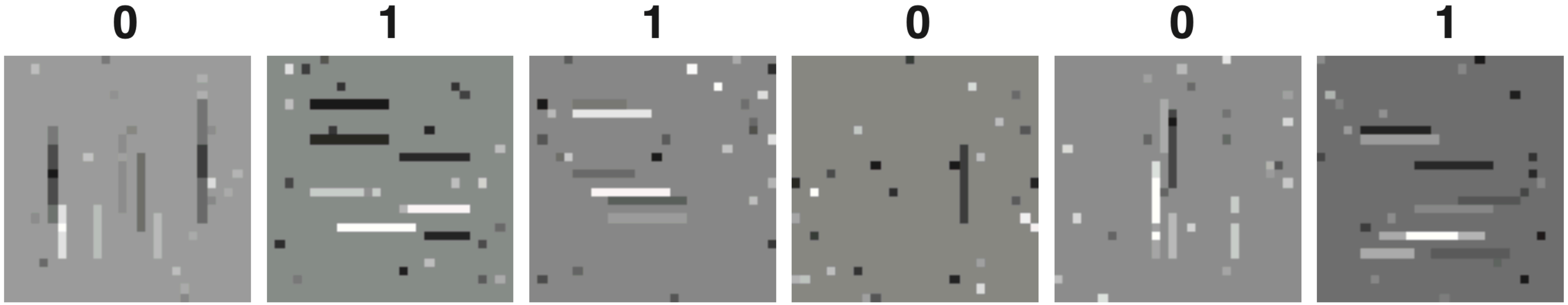
parameters $\mathbf{w}_1, \mathbf{w}_2, \mathbf{b}_1, \mathbf{b}_2, \mathbf{W}, \mathbf{c}$

Network training and evaluation

Training

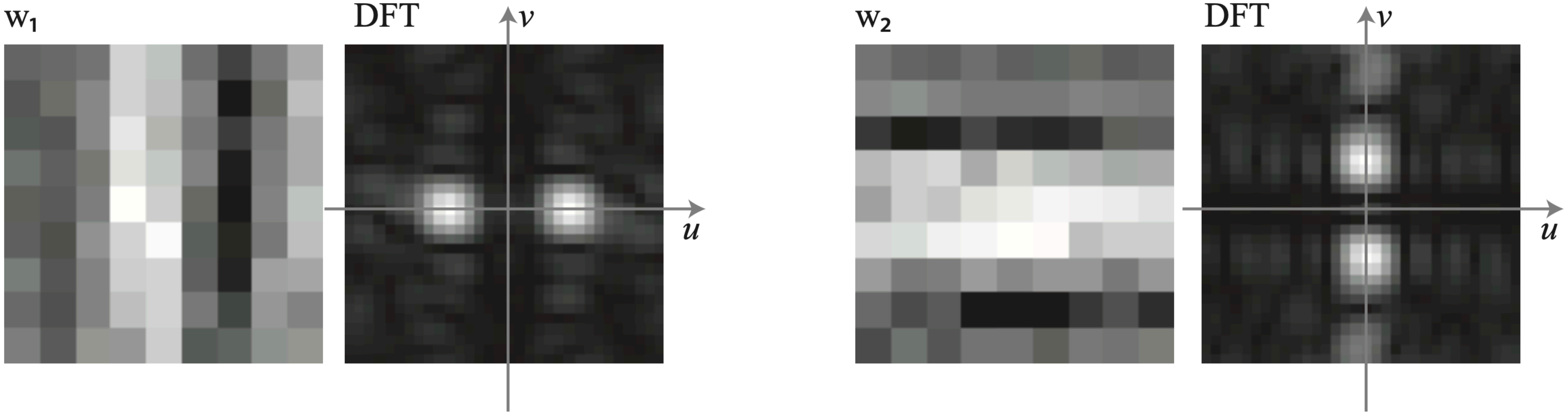


Testing
3/10000 misclassified



Network visualization

9x9 learned kernels:

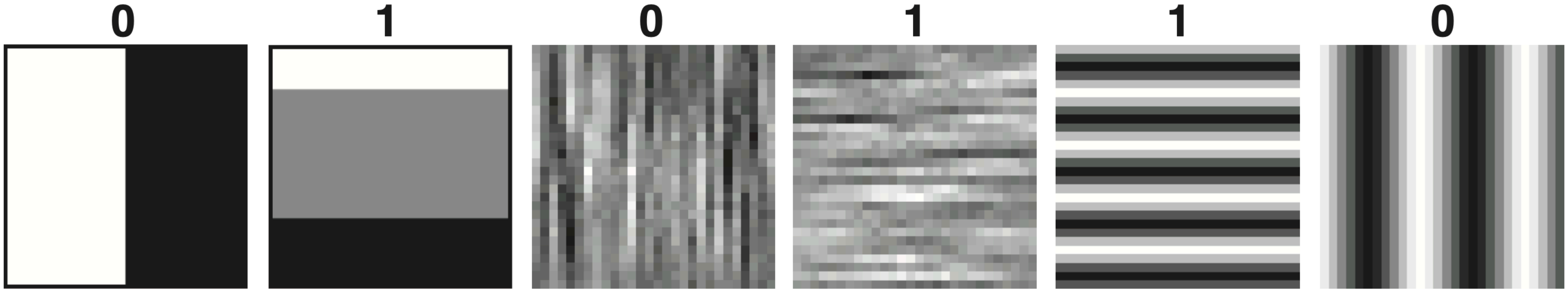


fc layer learned weight:

$$\mathbf{W} = \begin{bmatrix} 2.83 & -2.36 \\ -0.60 & 1.14 \end{bmatrix}$$

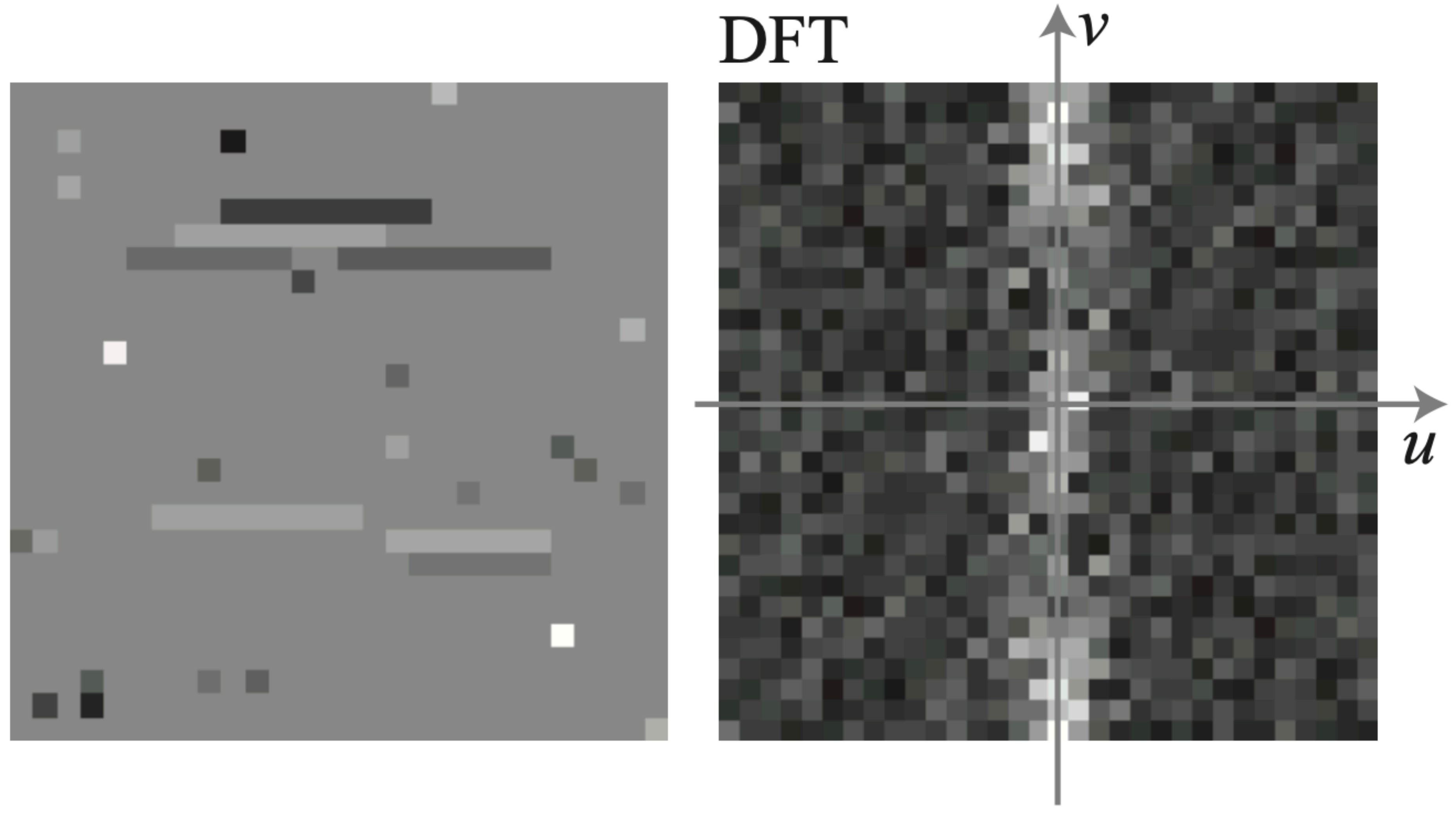
Out of domain generalization

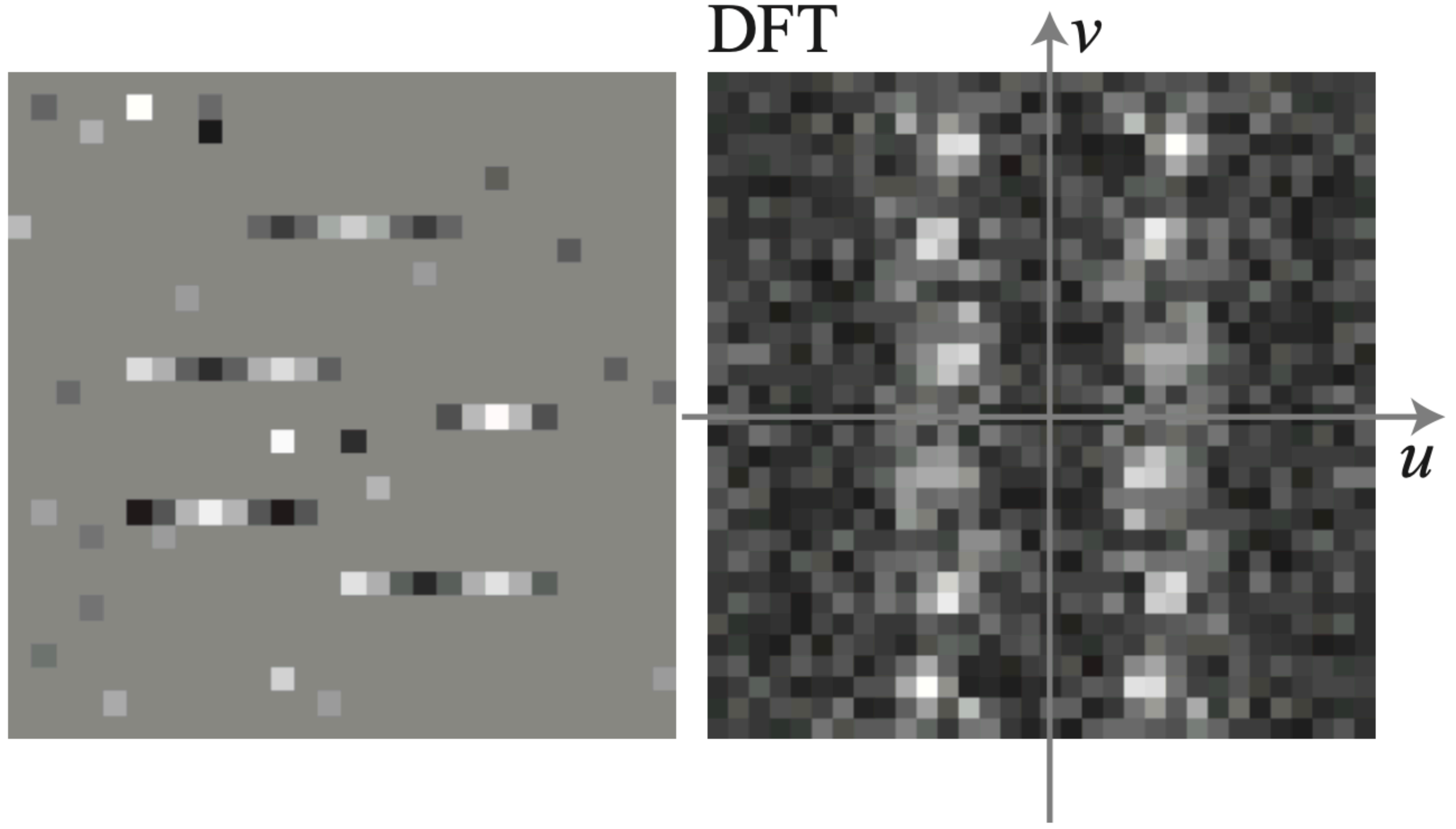
Out of domain test samples are classified correctly



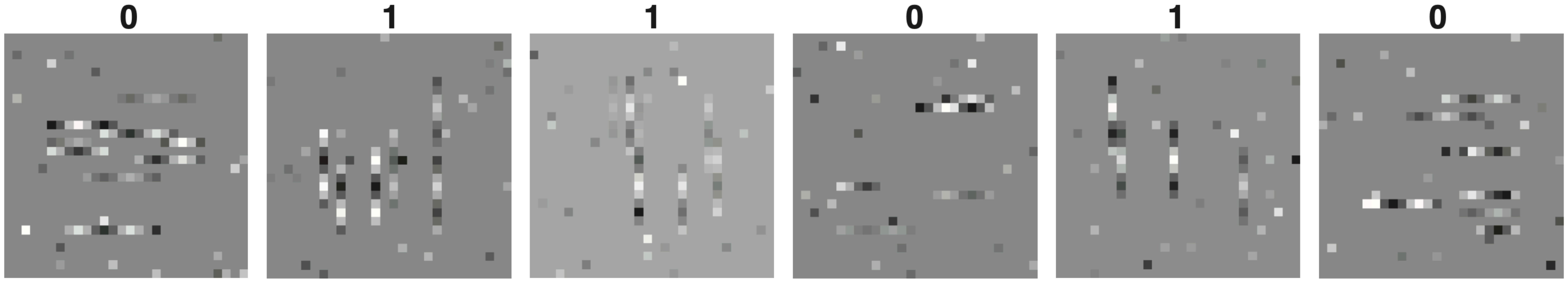
Identifying vulnerability

- Modulation: multiplying image with sinusoidal wave moves spectral content horizontally





- All lines with sinusoidal texture are misclassified

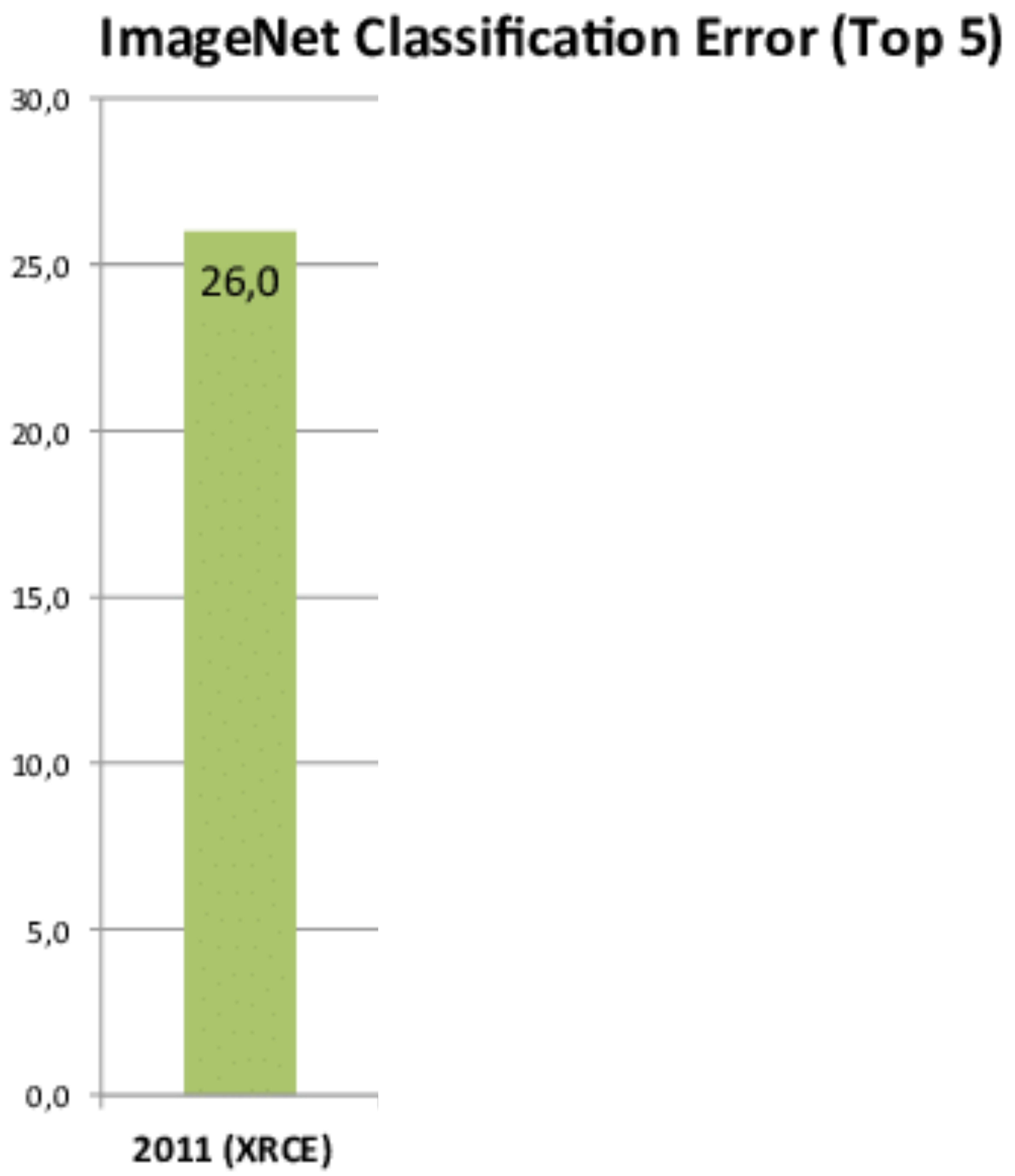


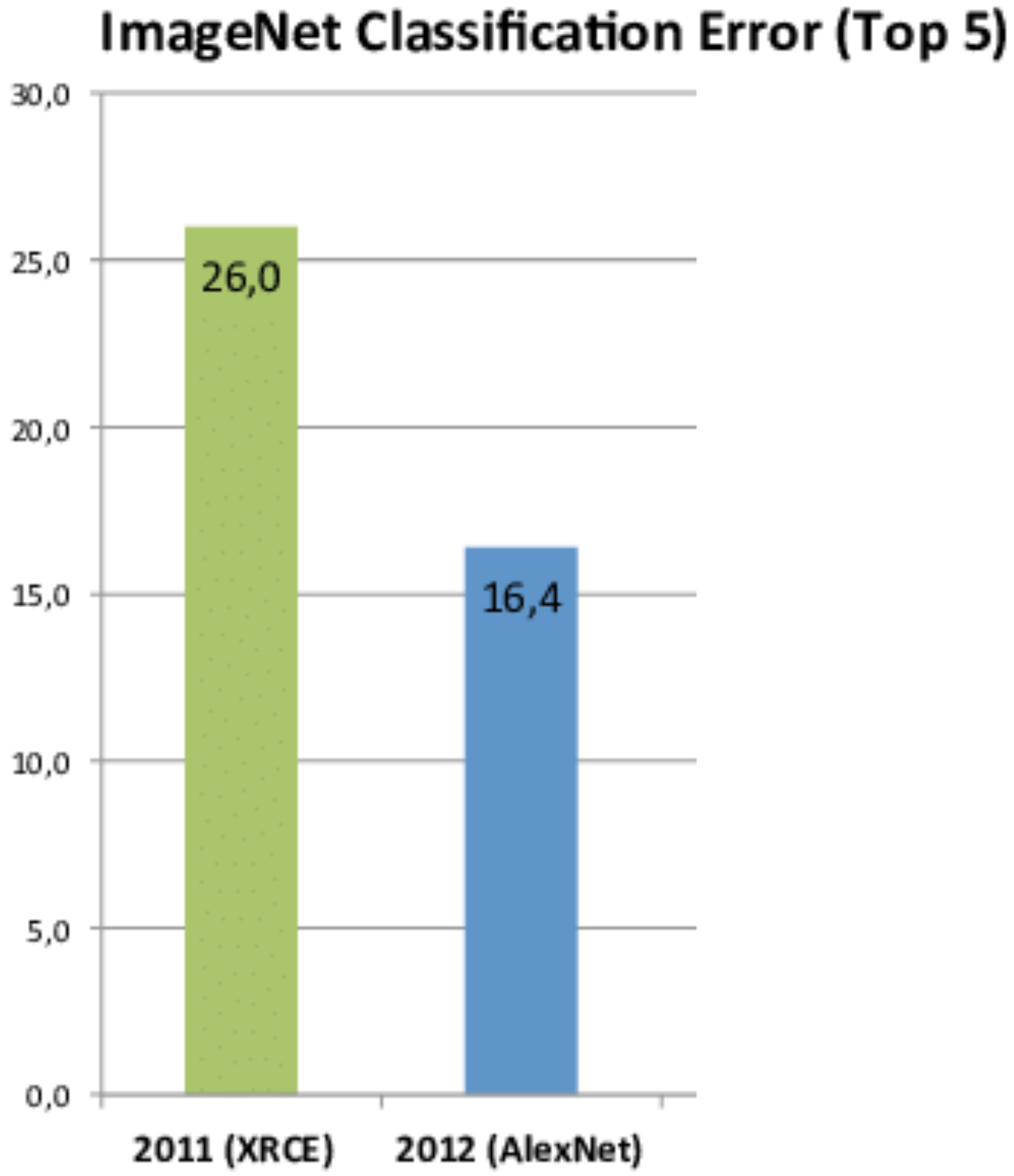
Analysis to run when training a large system

- Training and evaluation
- Visualize and understand the network
- Out of domain generalization
- Identifying vulnerability

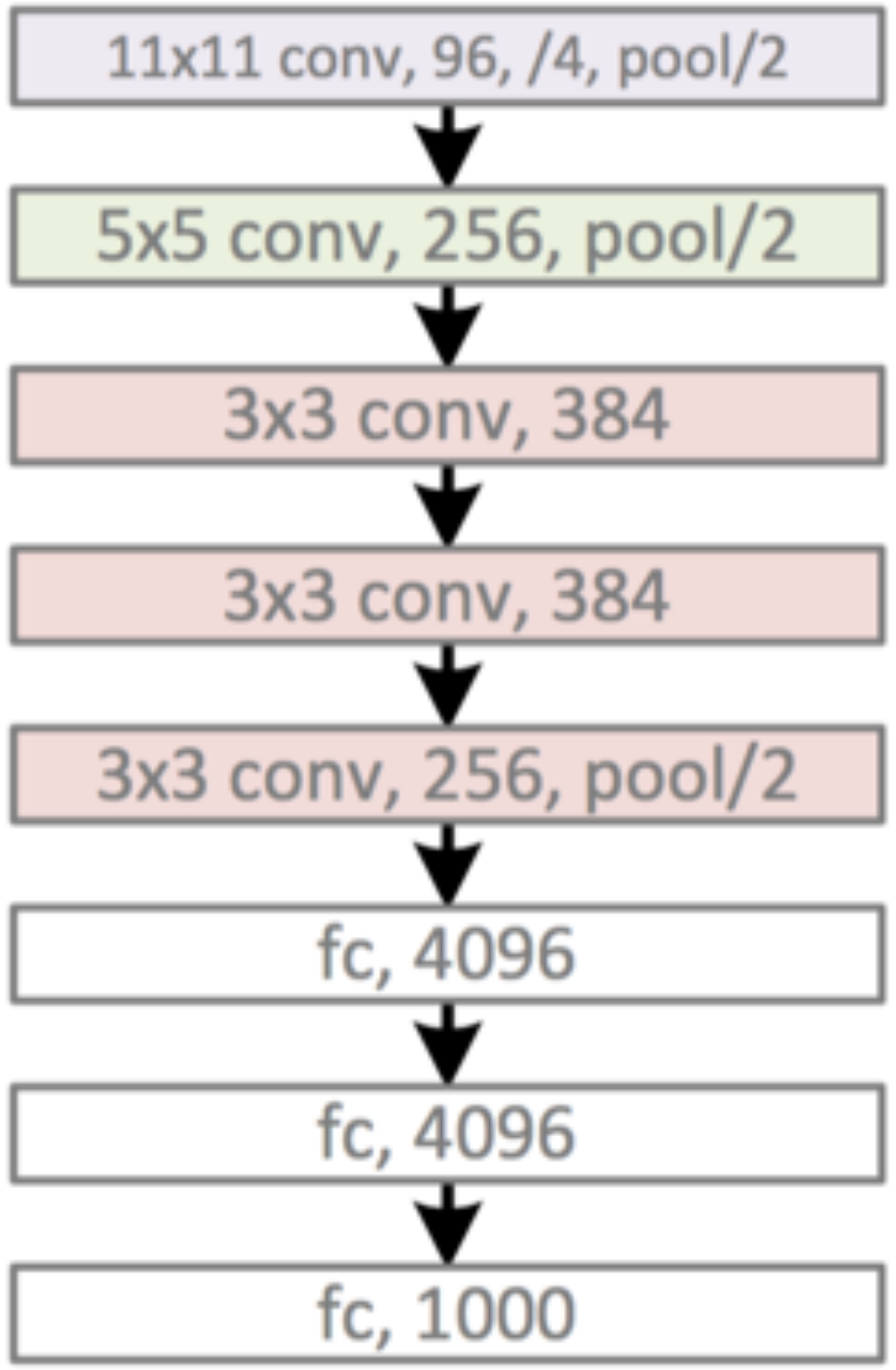
Some networks

… 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)



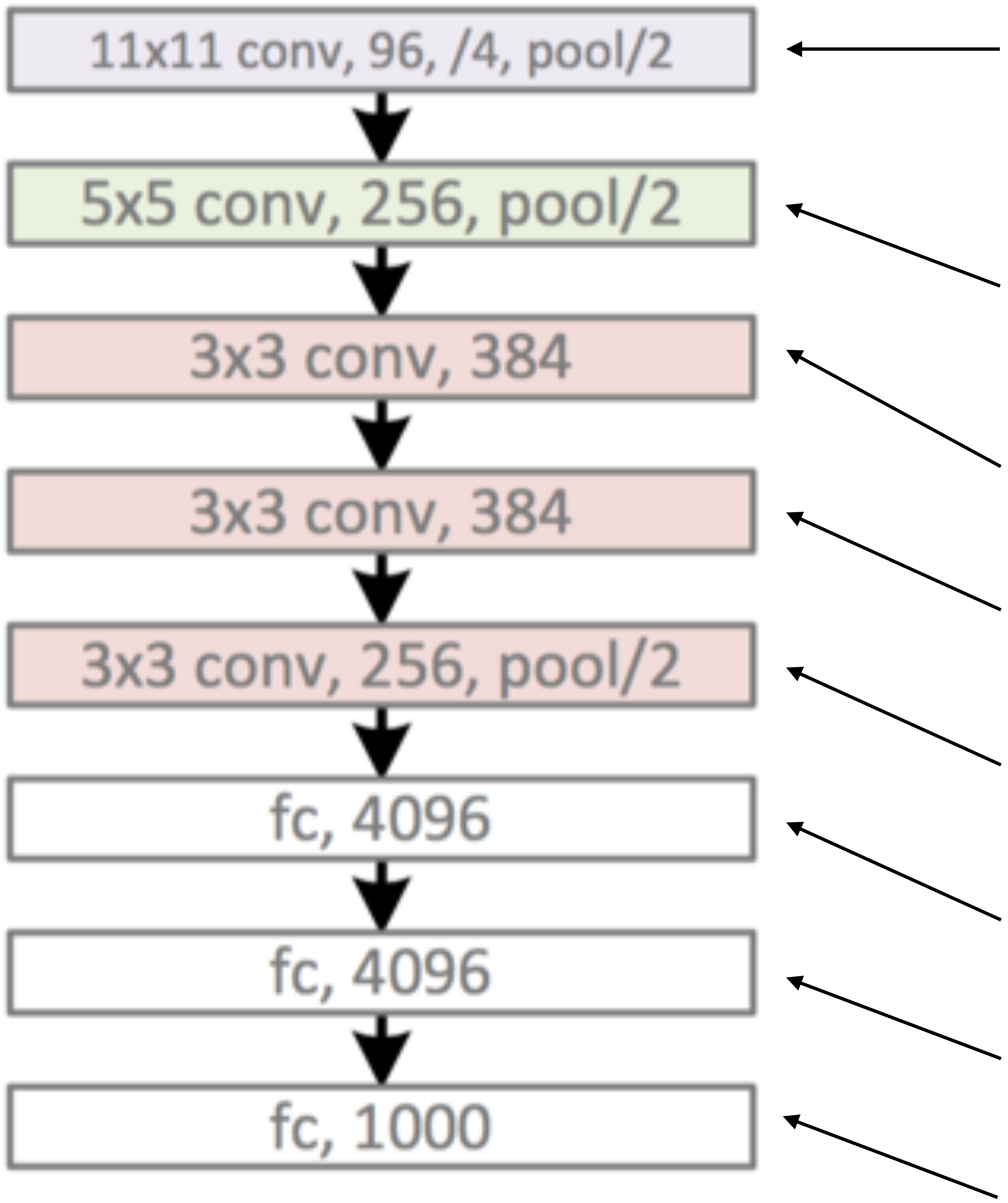
2012: AlexNet
5 conv. layers



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

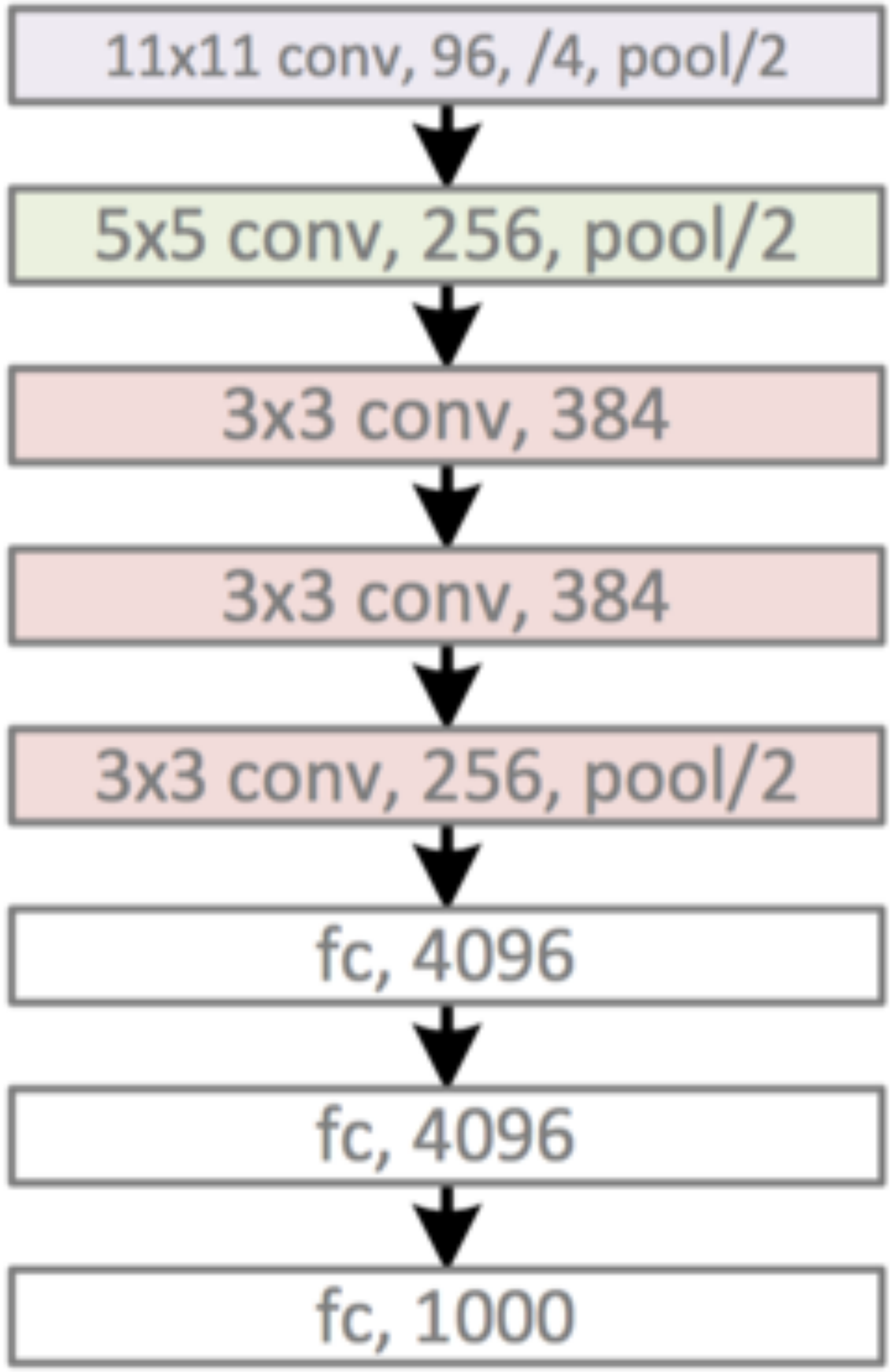
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[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

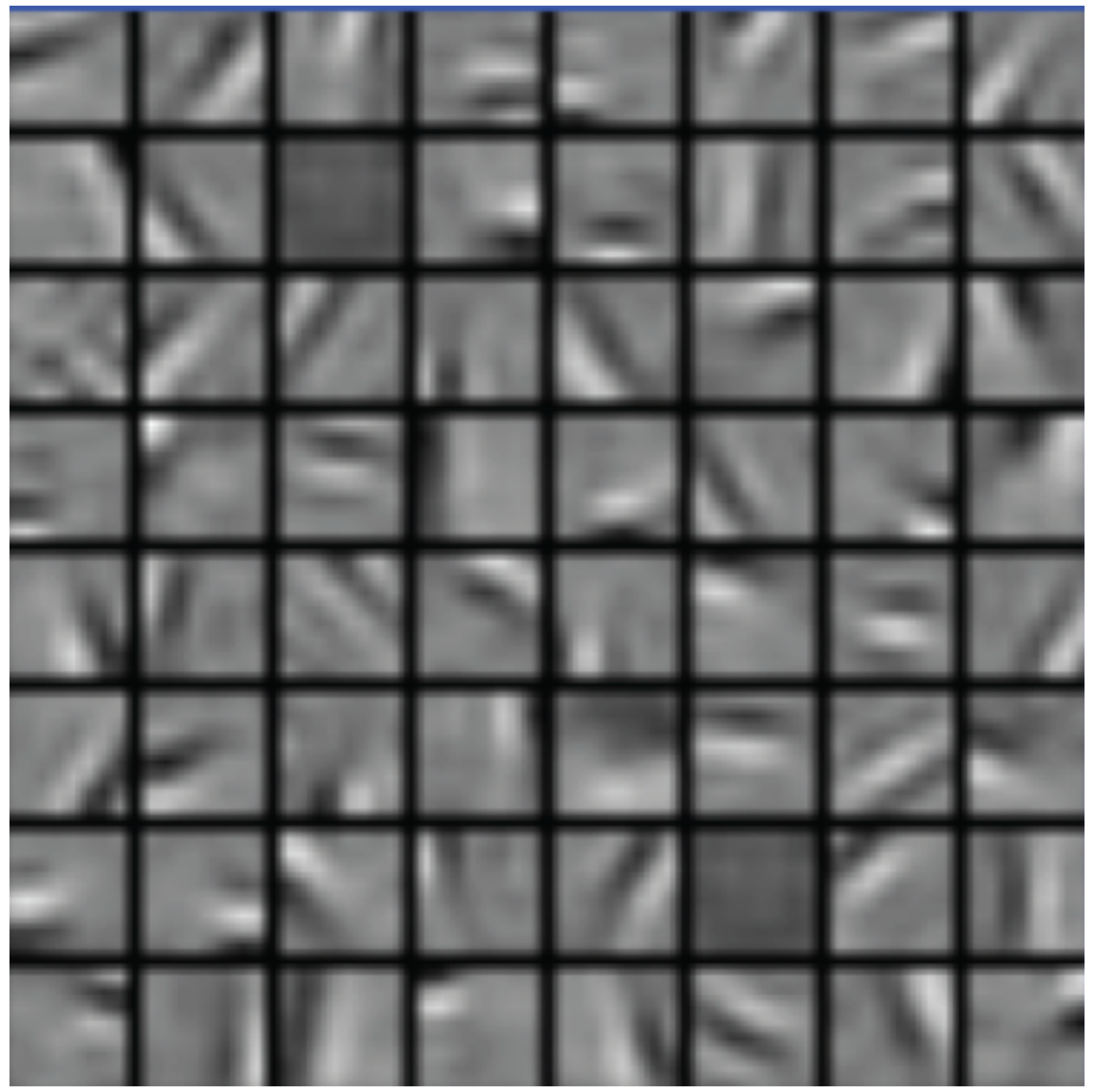
[1000] FC8: 1000 neurons (class scores)



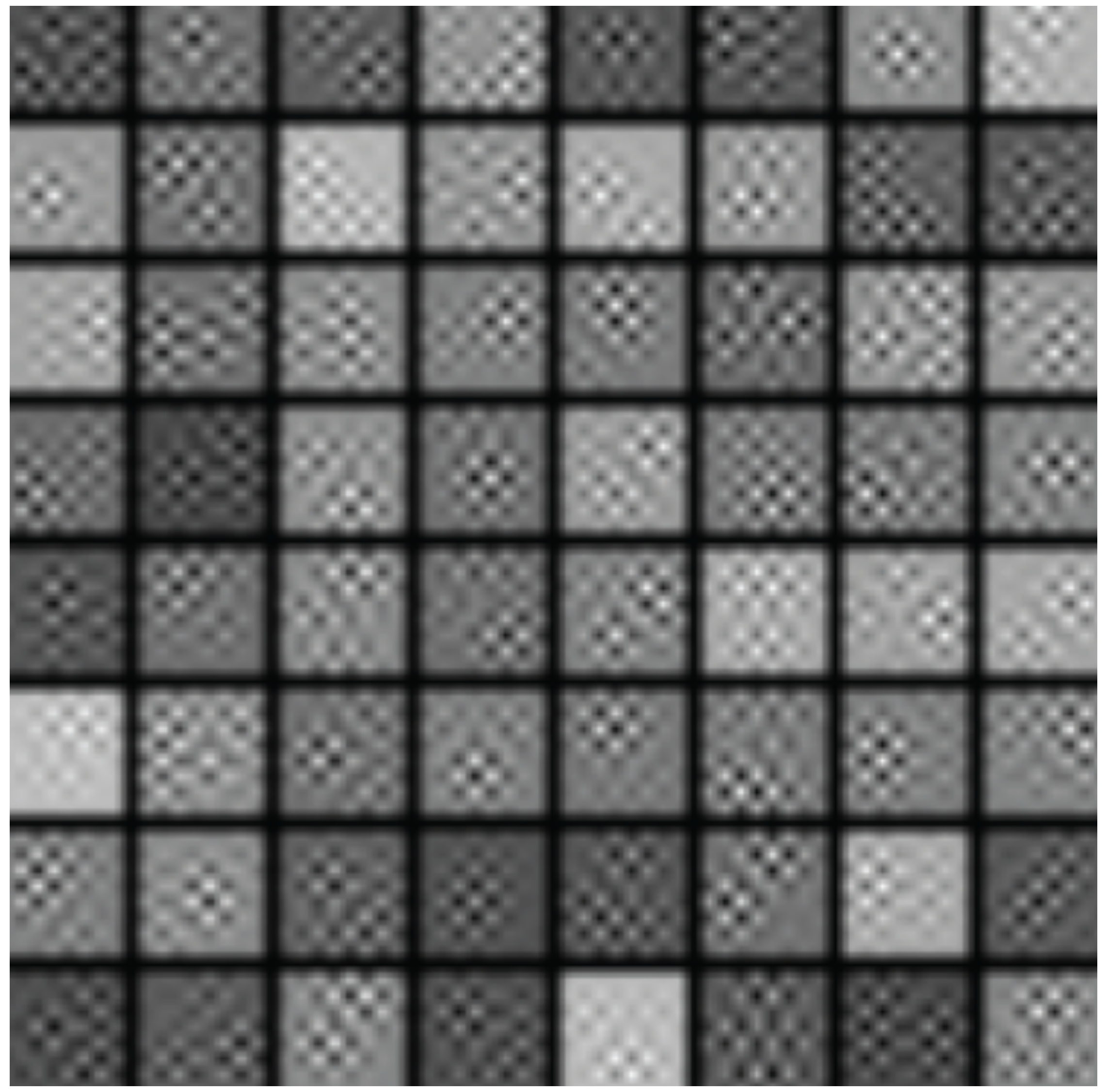
What filters are learned?

What filters are learned?

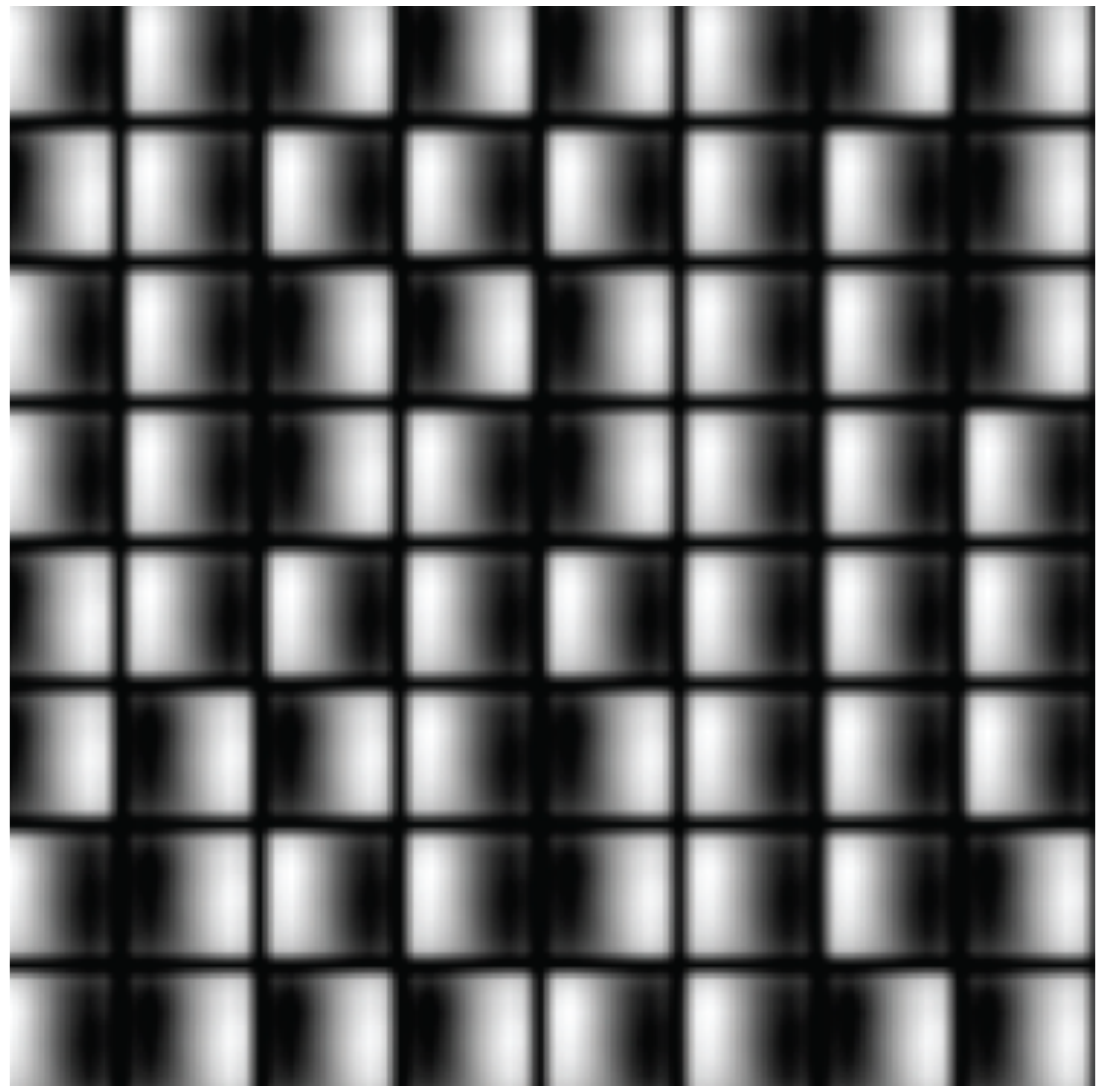
A



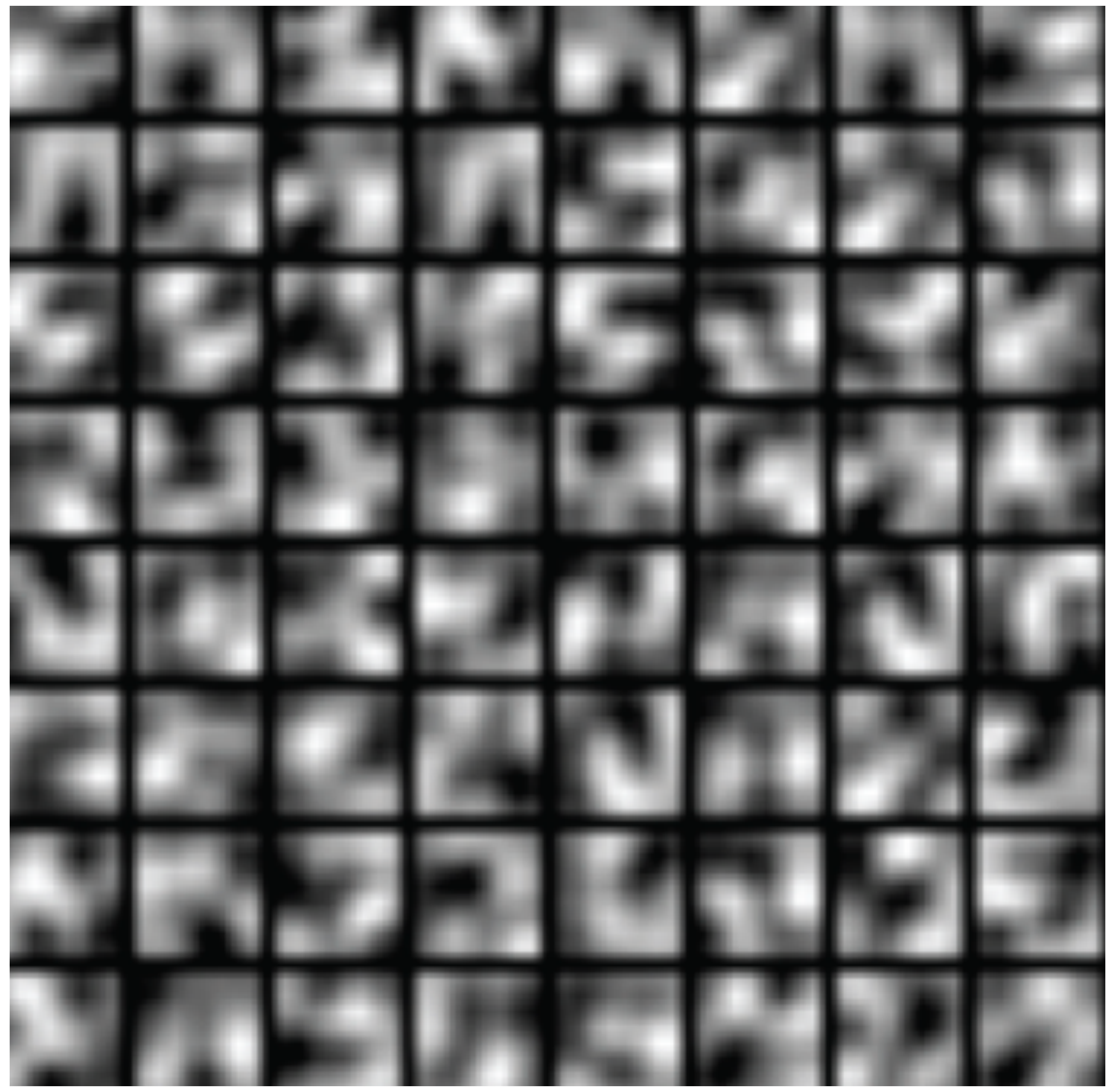
B



C

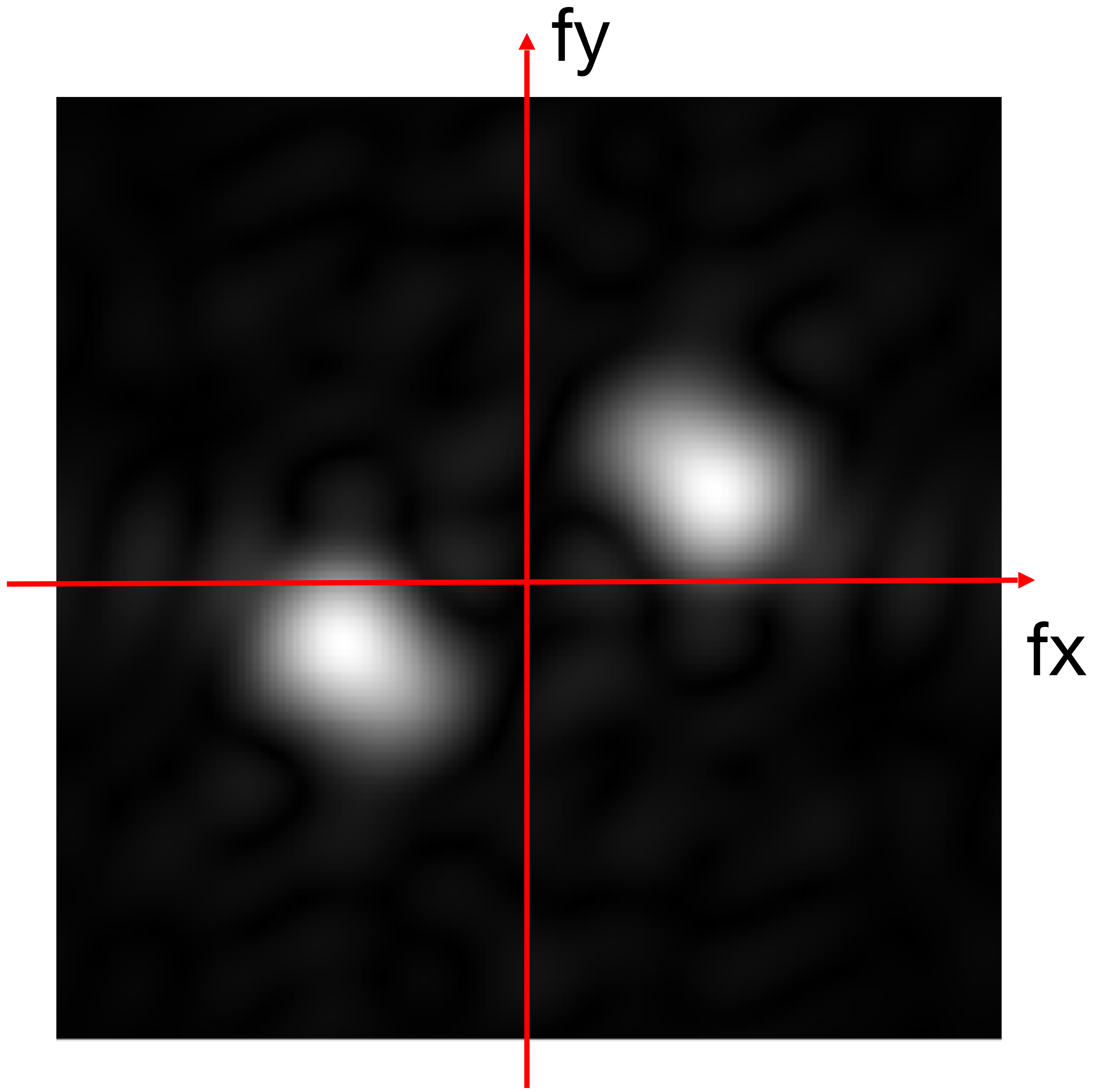


D

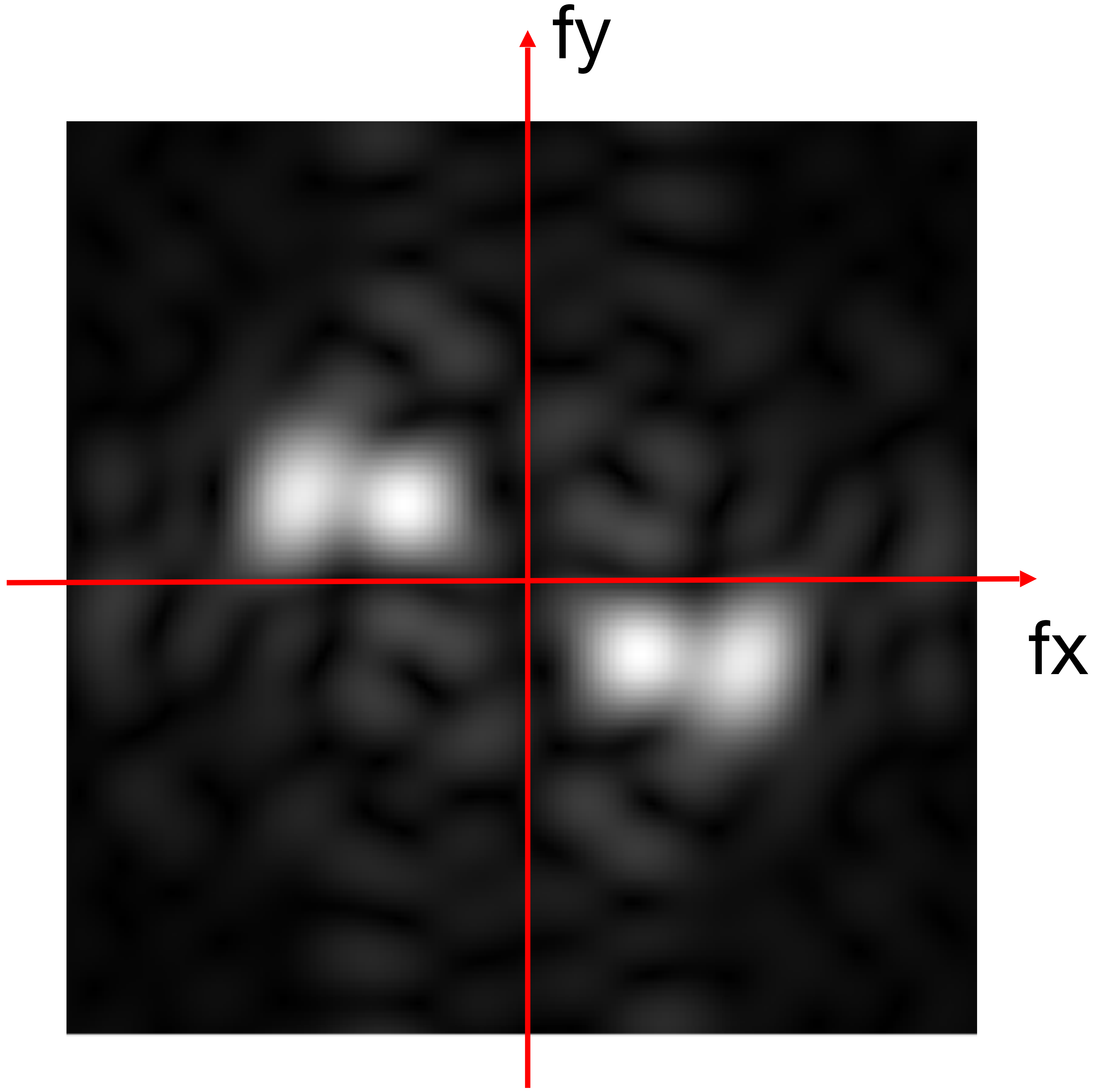


Get to know your units

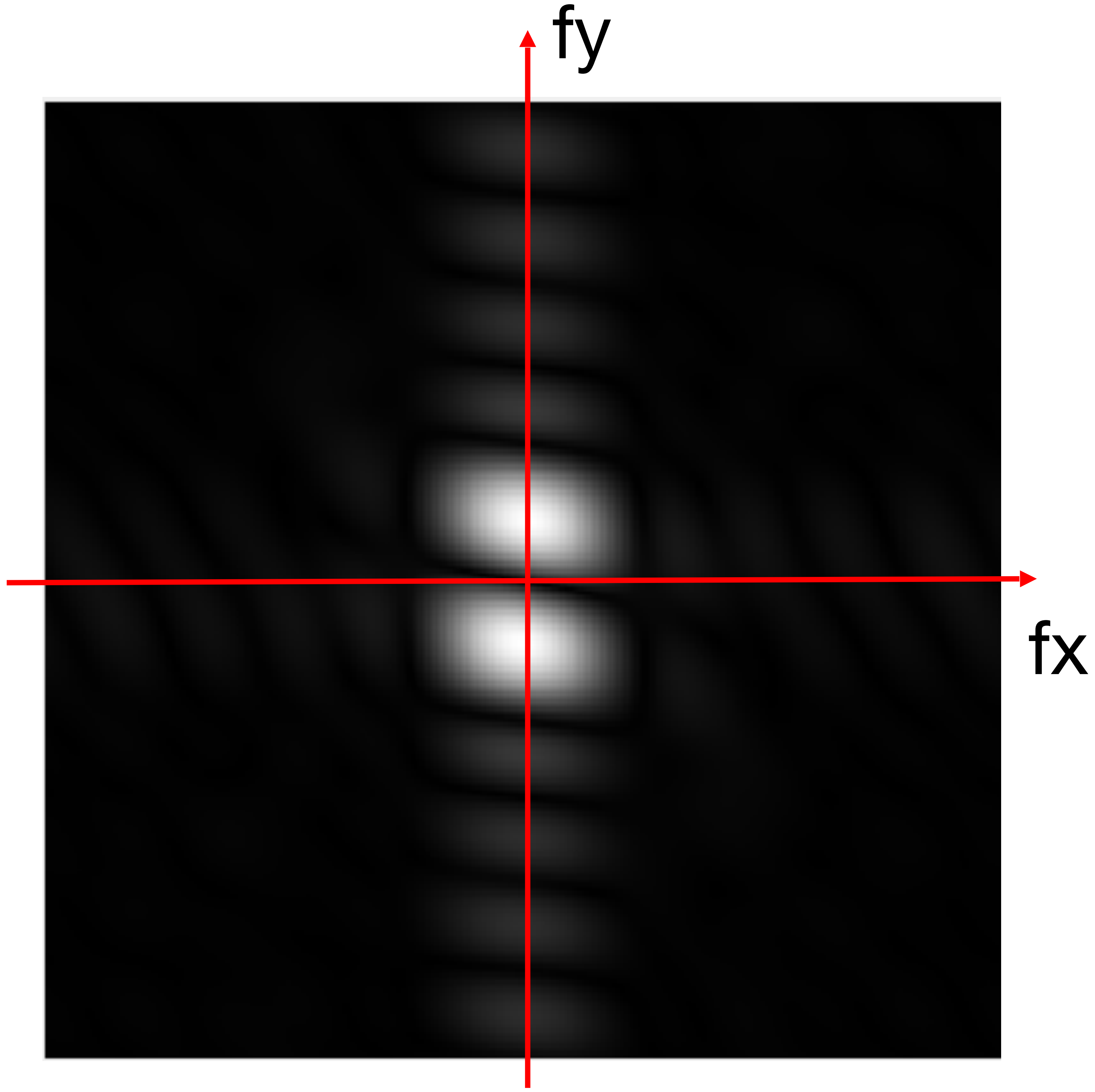
11x11 convolution kernel
(3 color channels)



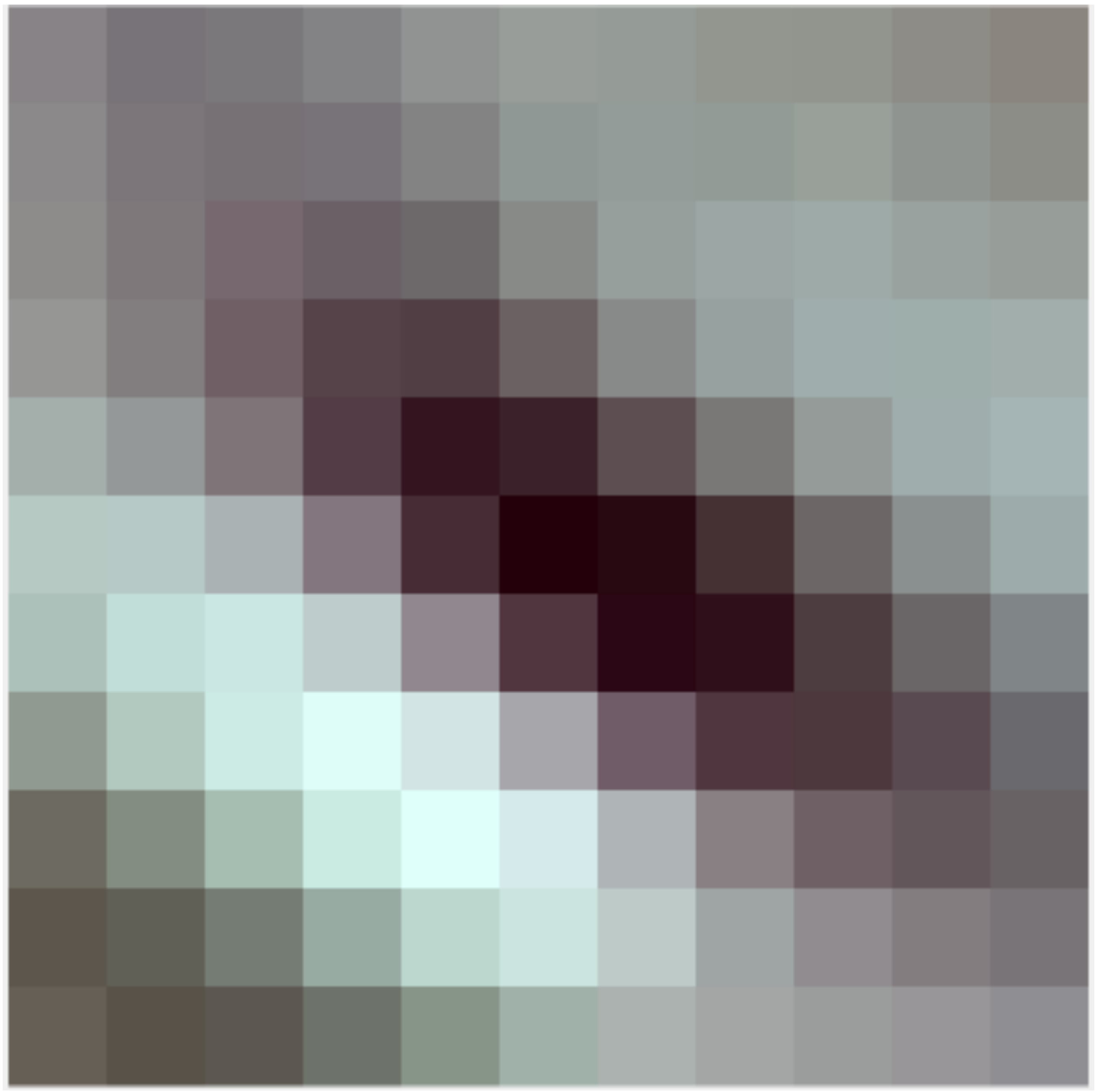
Get to know your units

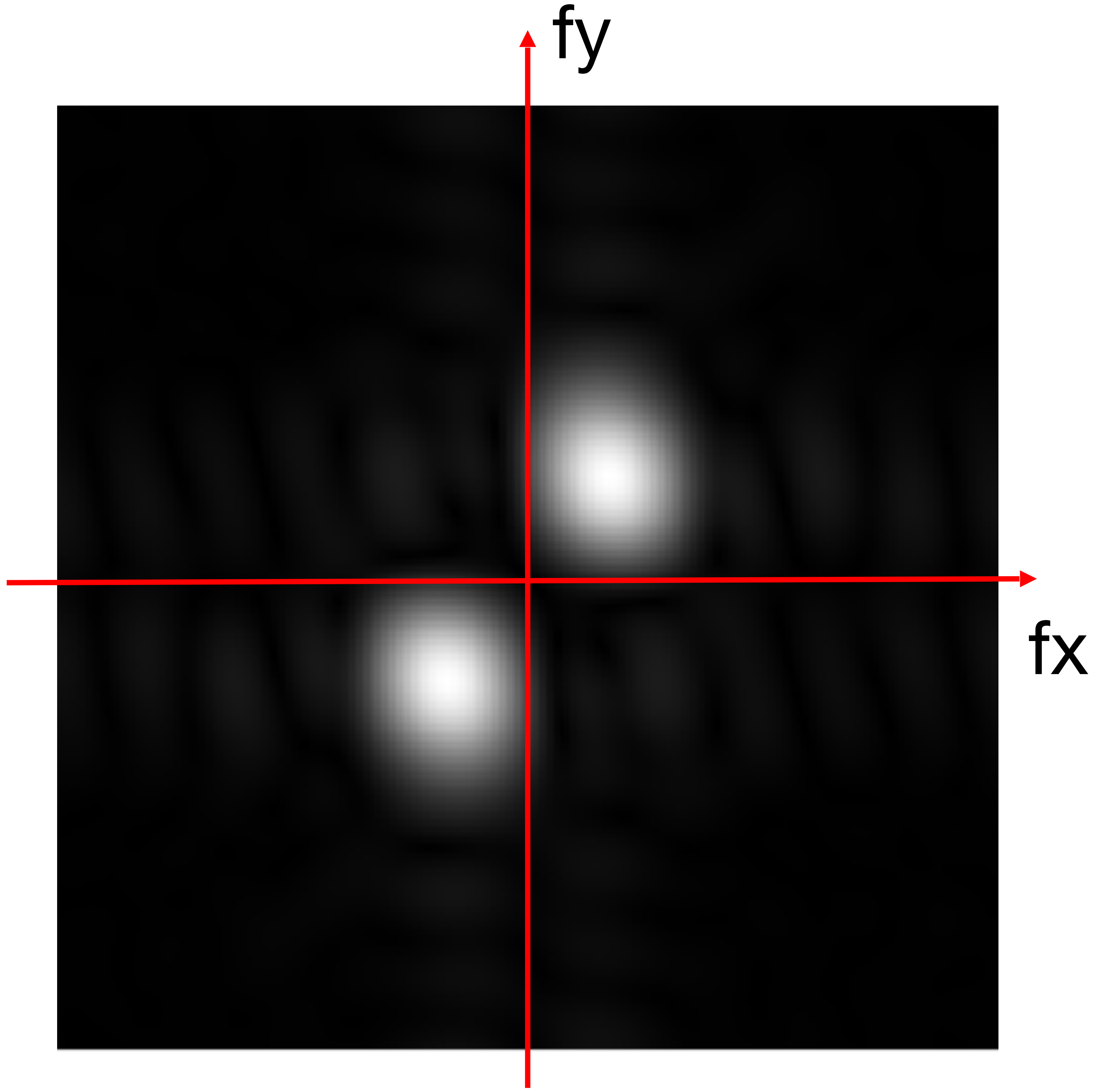


Get to know your units

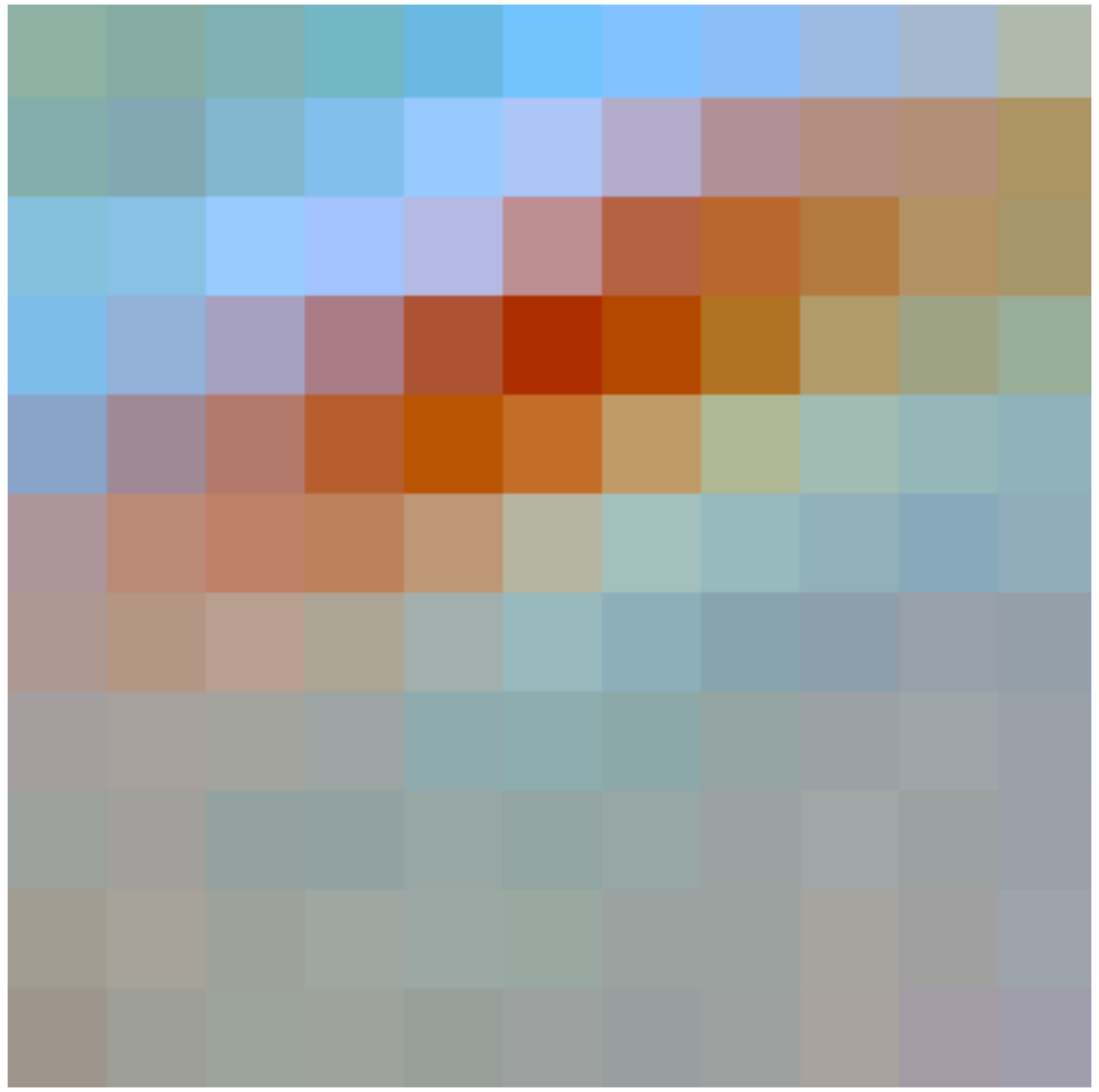


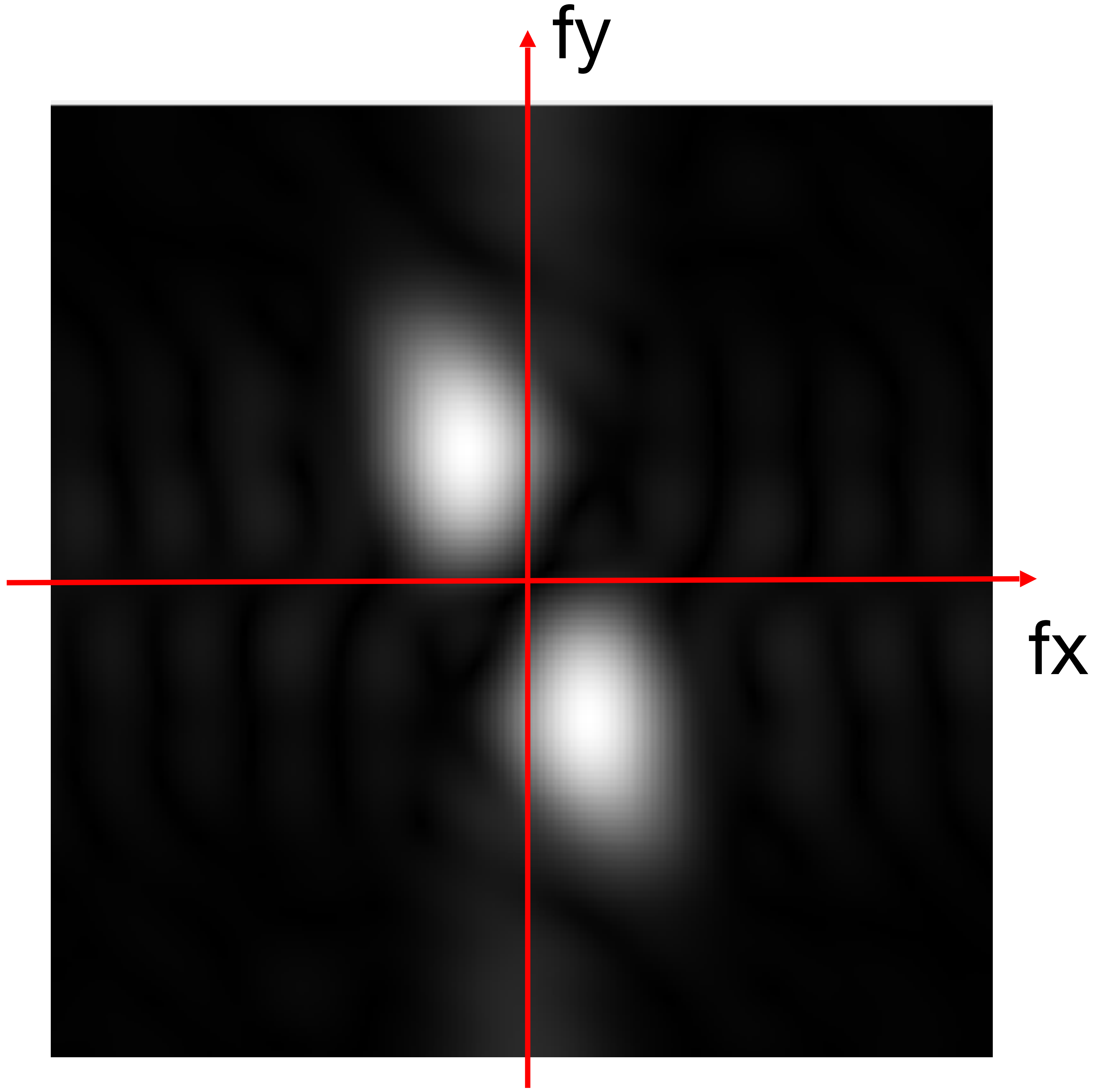
Get to know your units



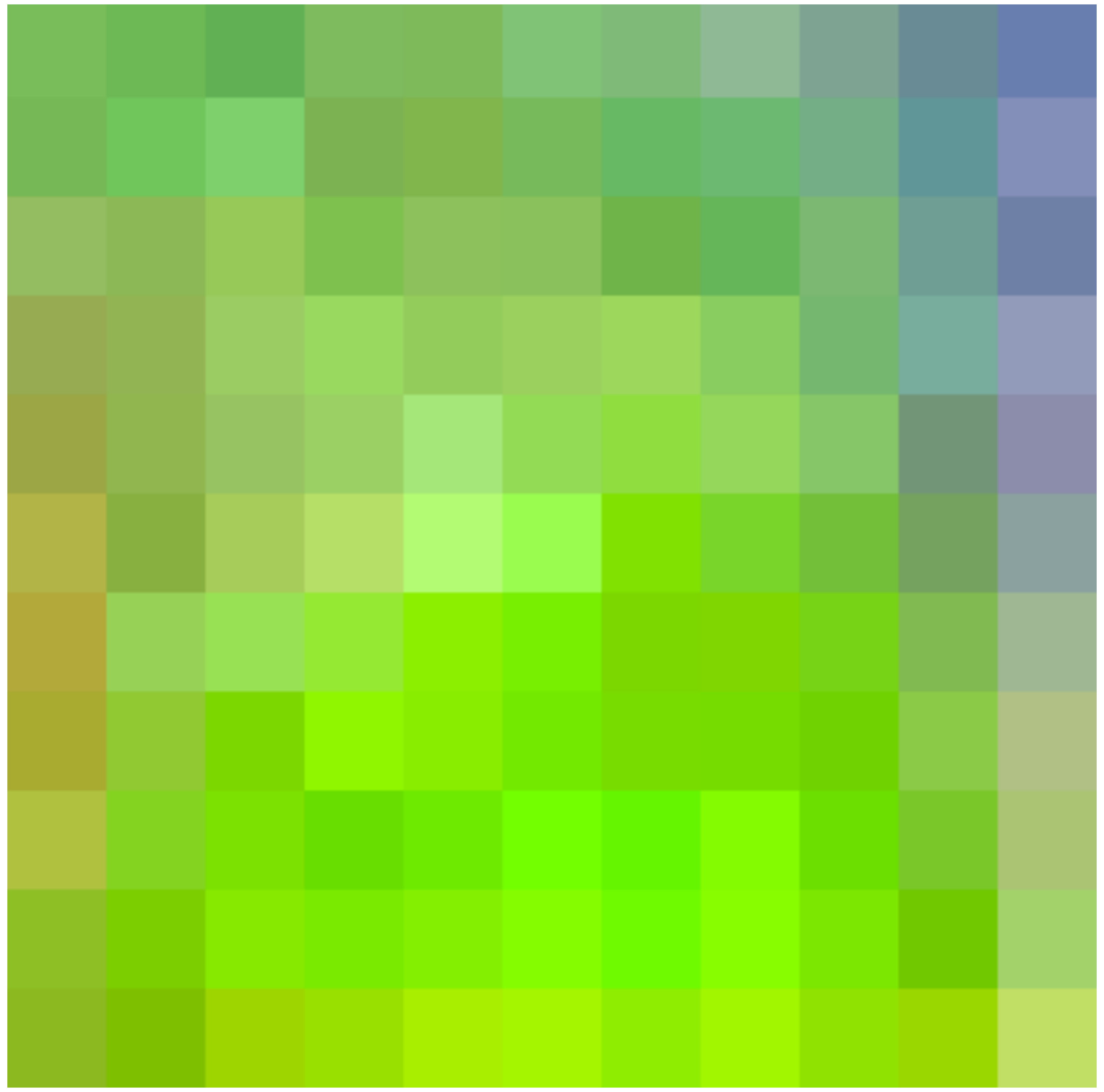


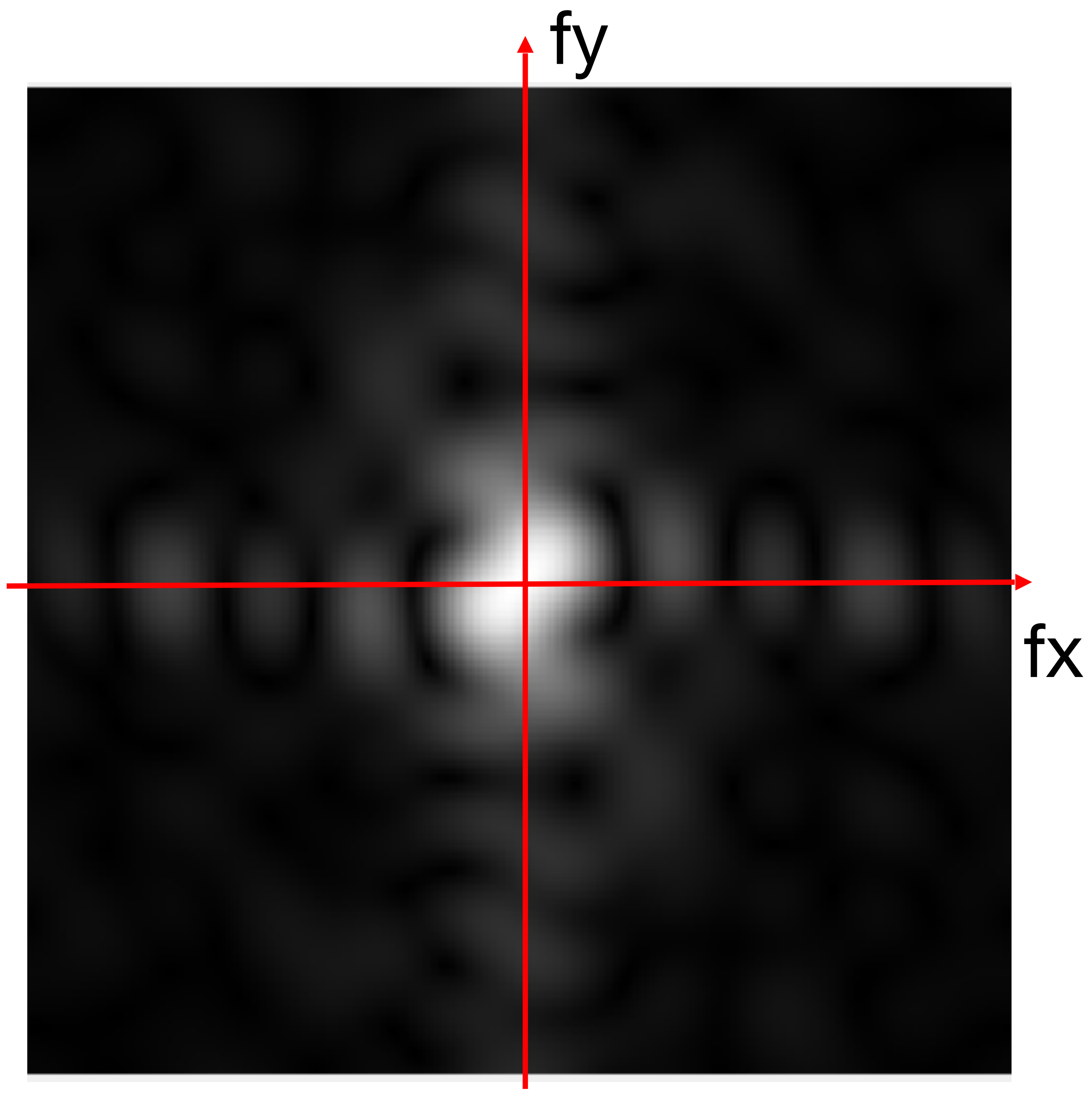
Get to know your units



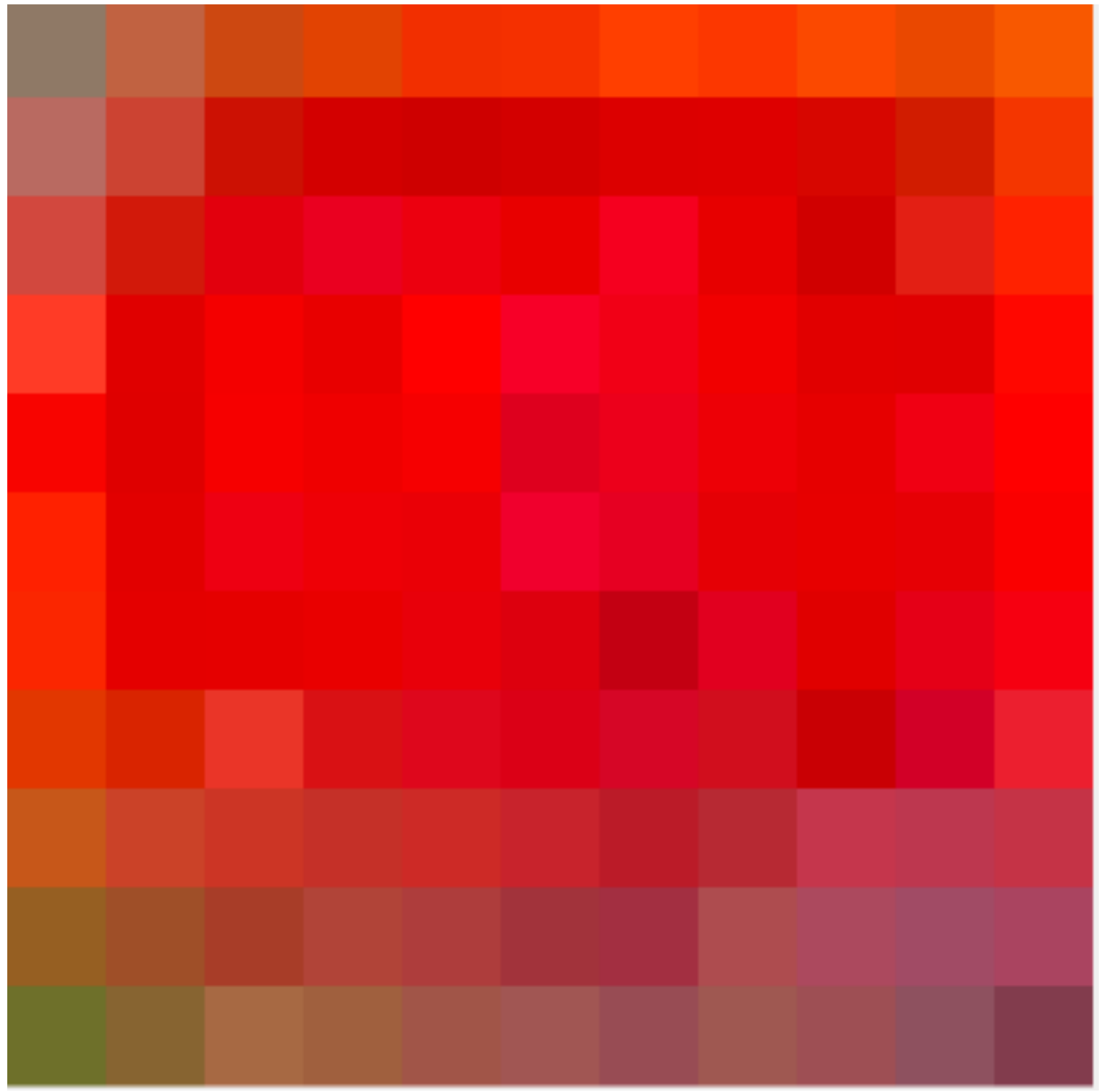


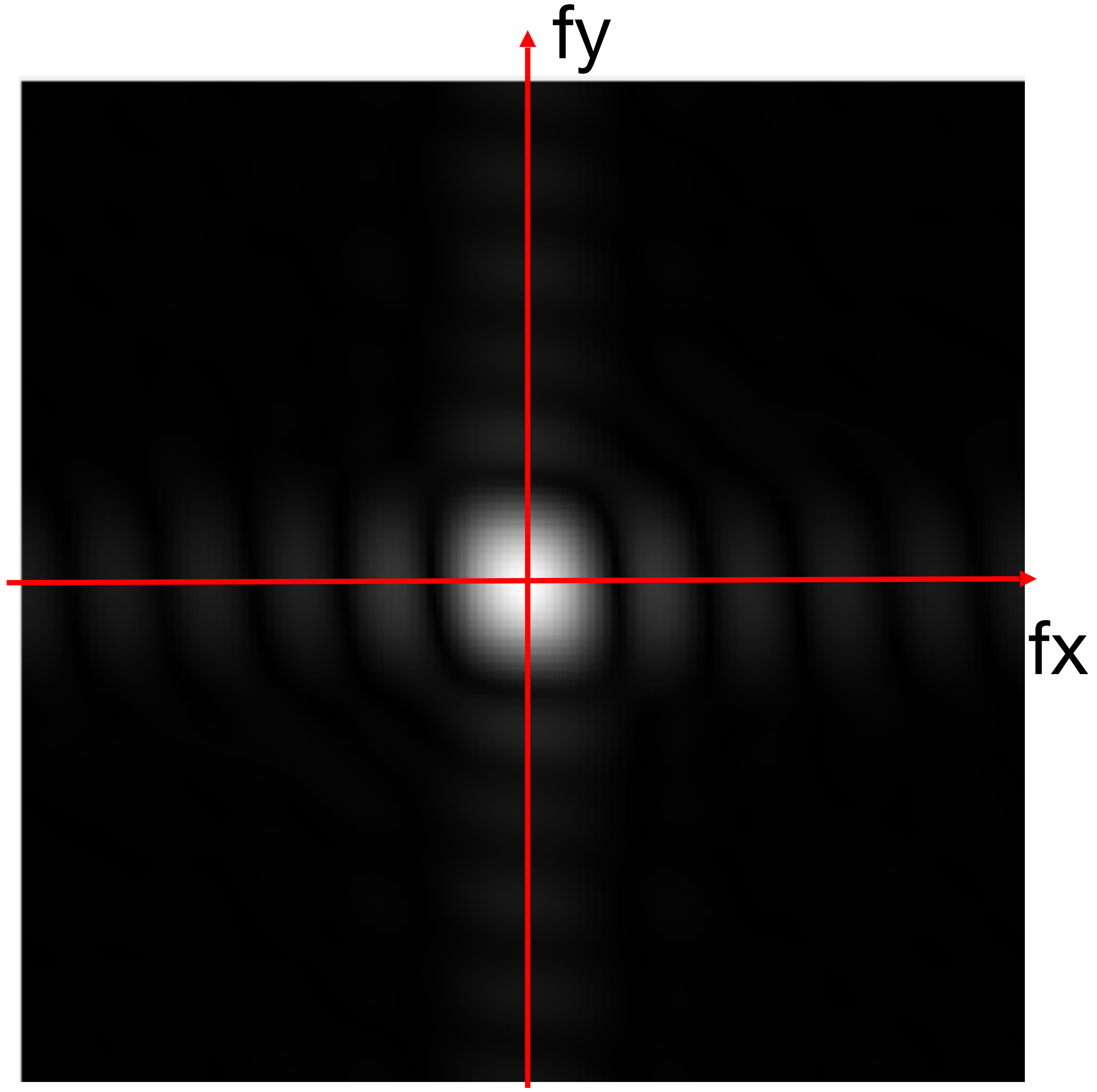
Get to know your units



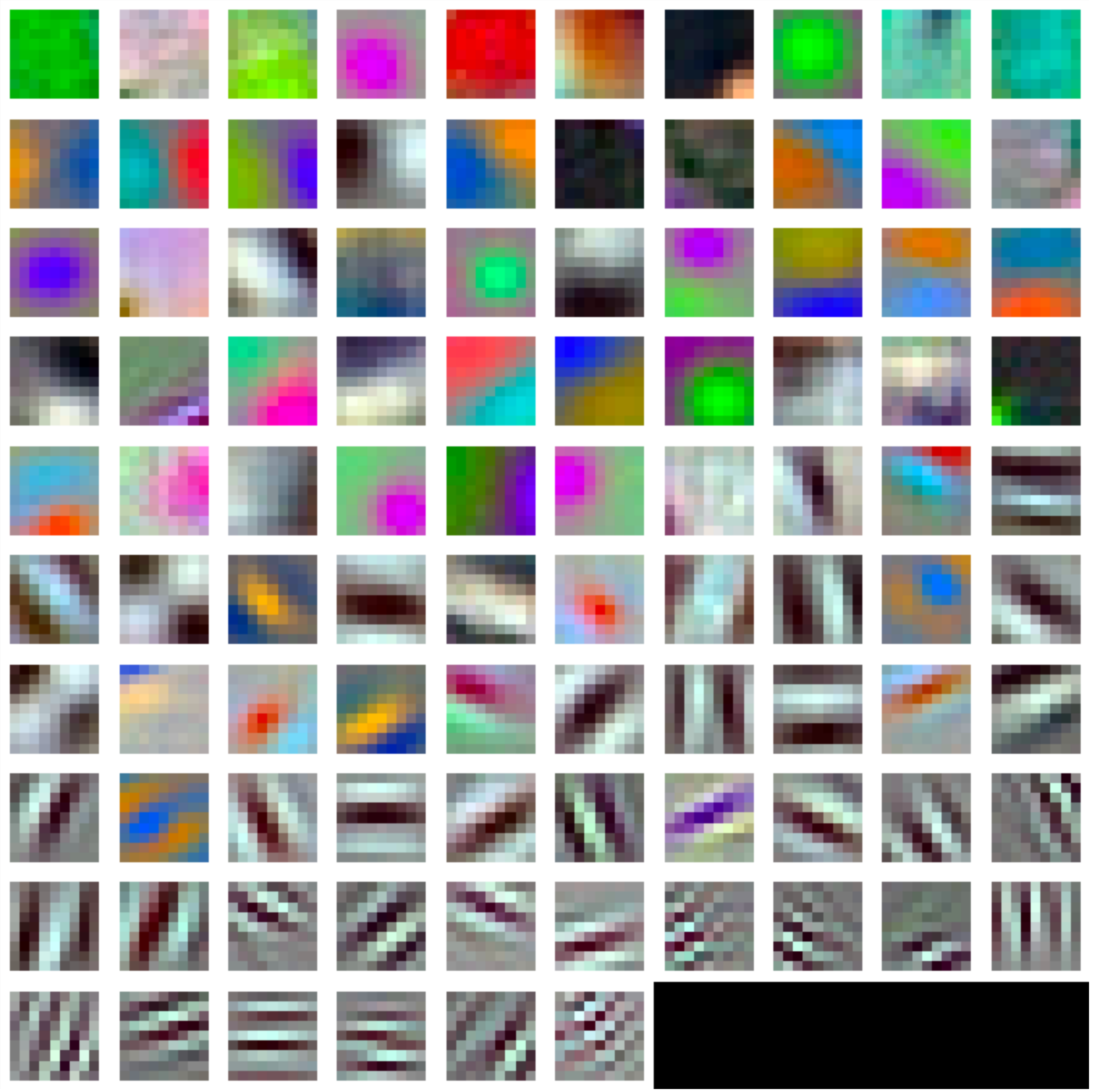


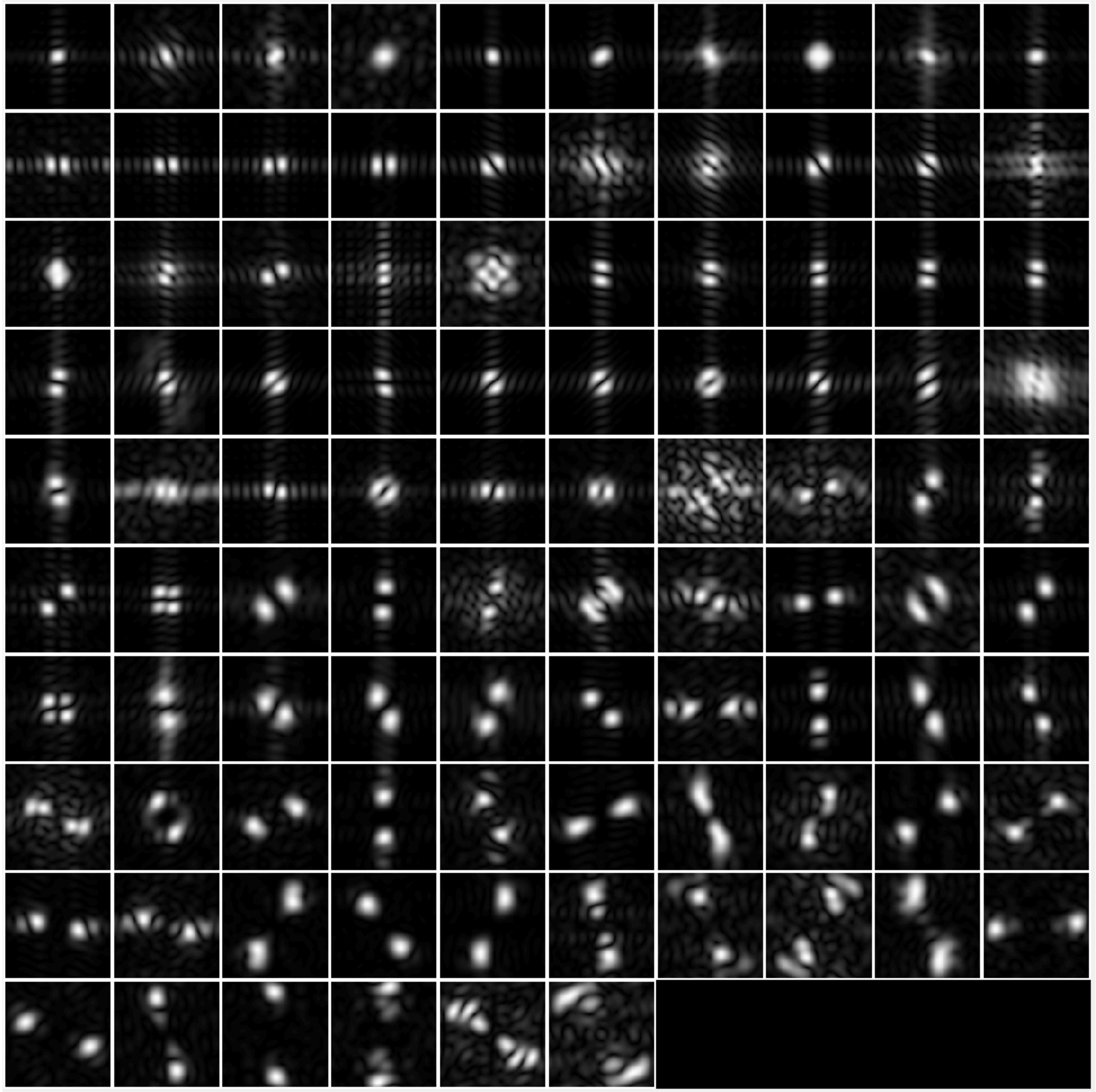
Get to know your units





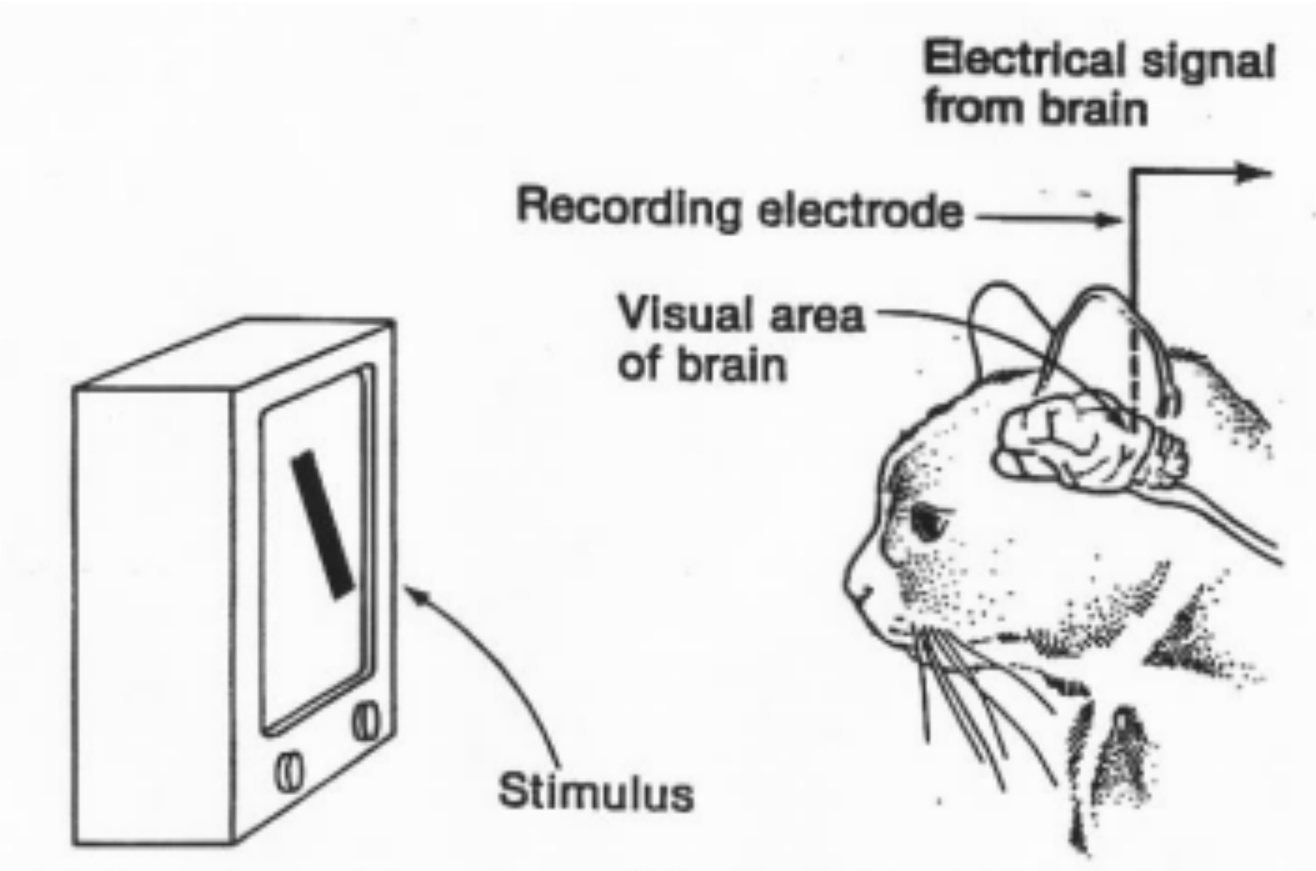
Get to know your units

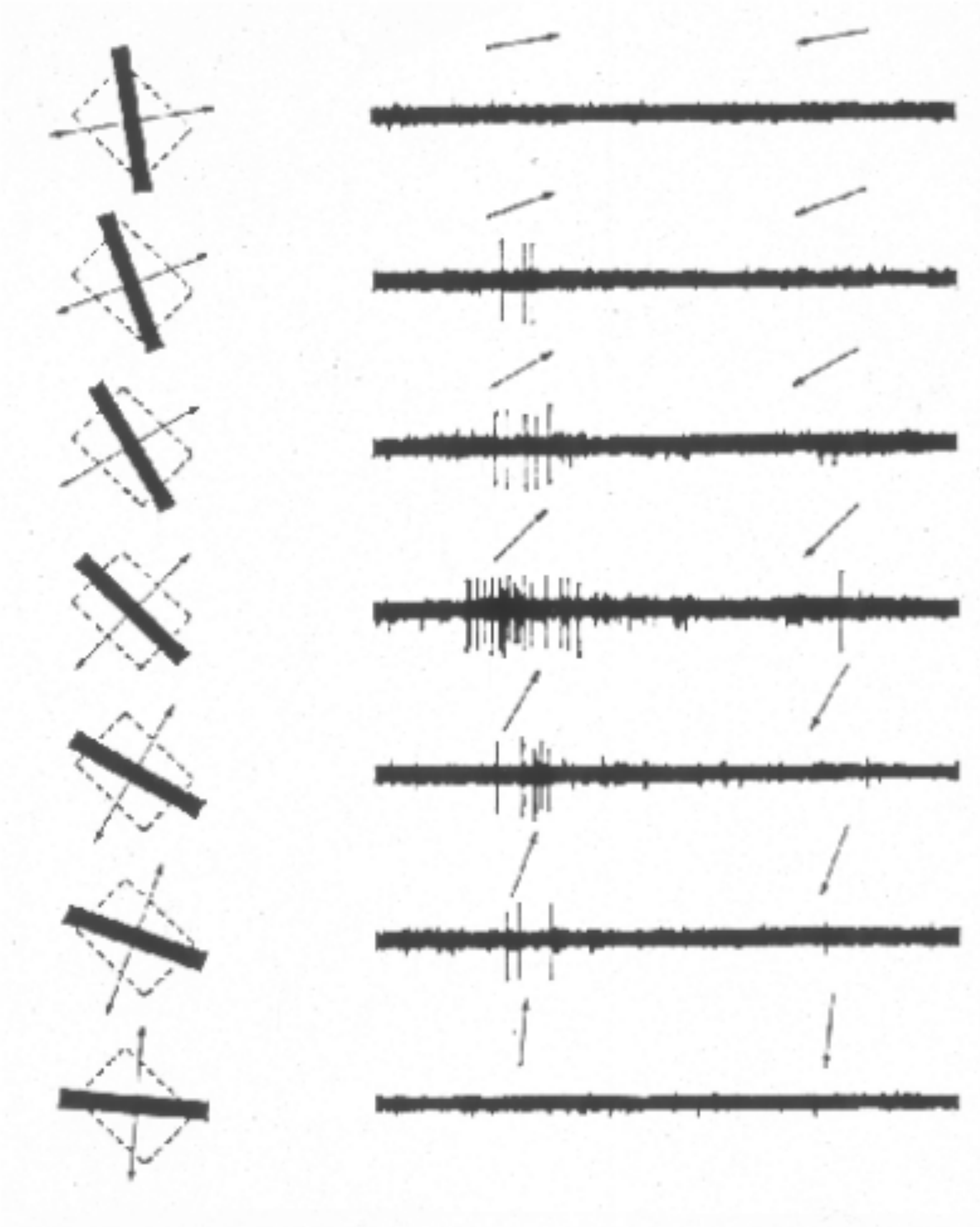


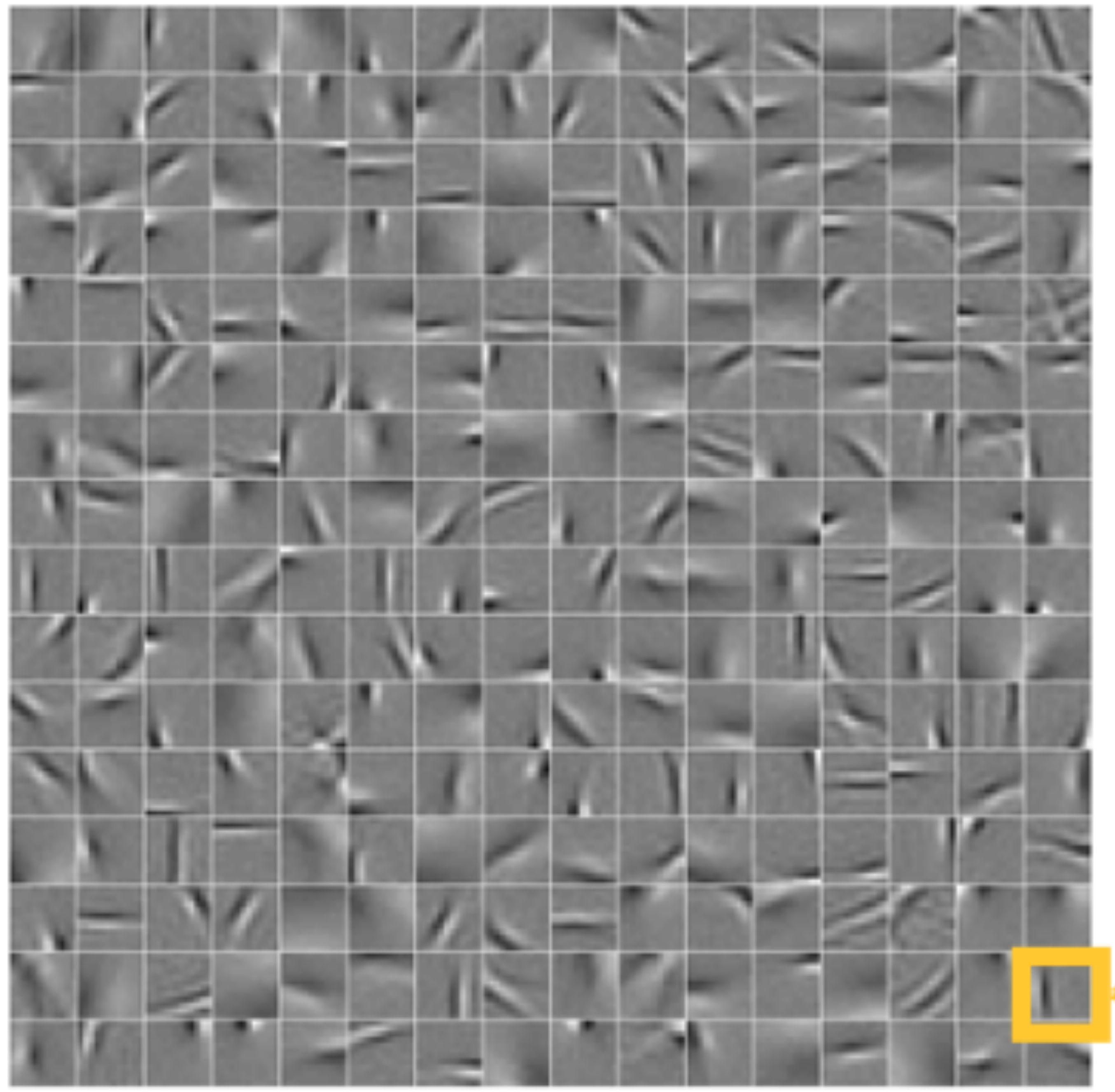


96 Units in conv1

[Hubel and Wiesel 59]

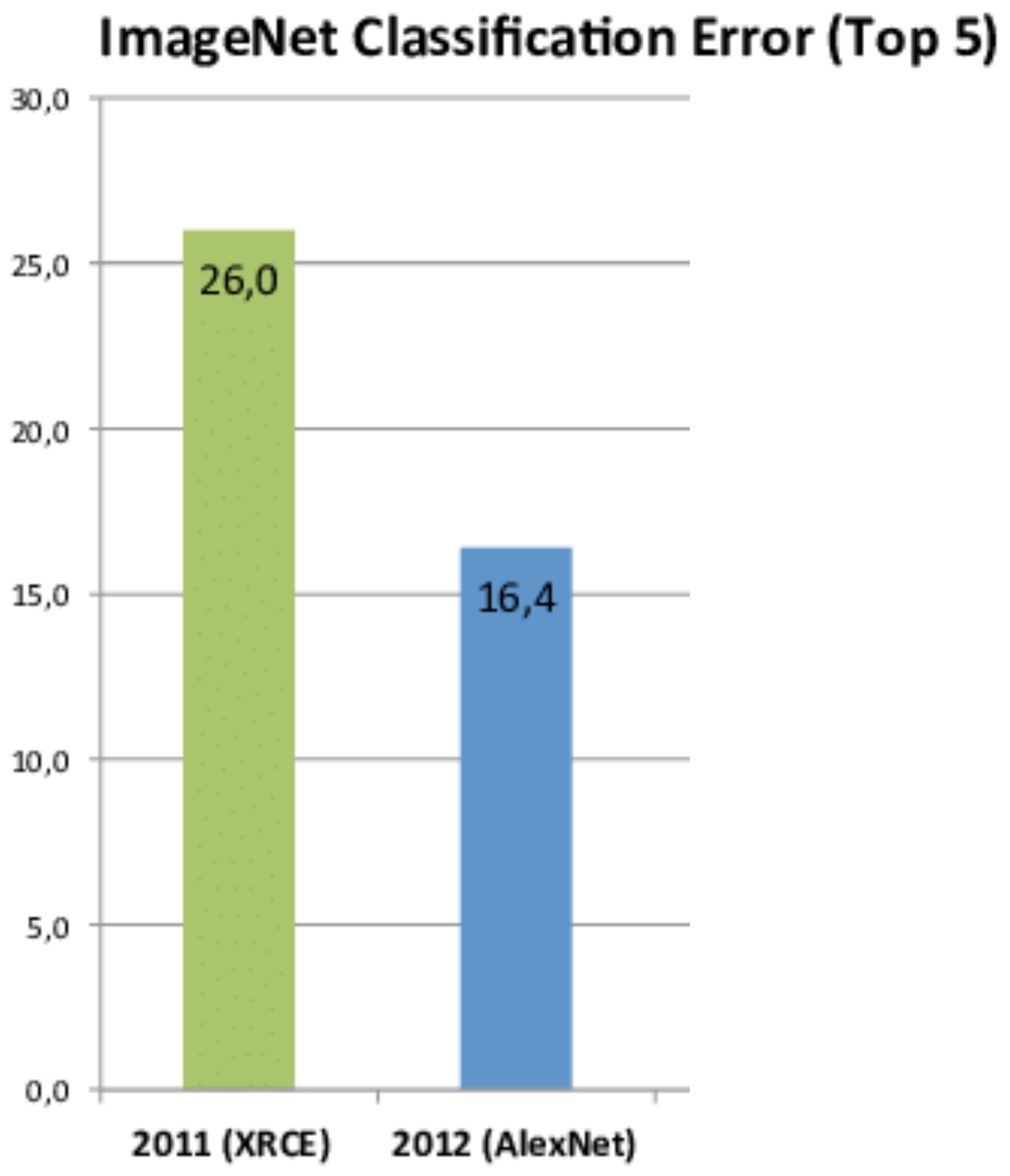


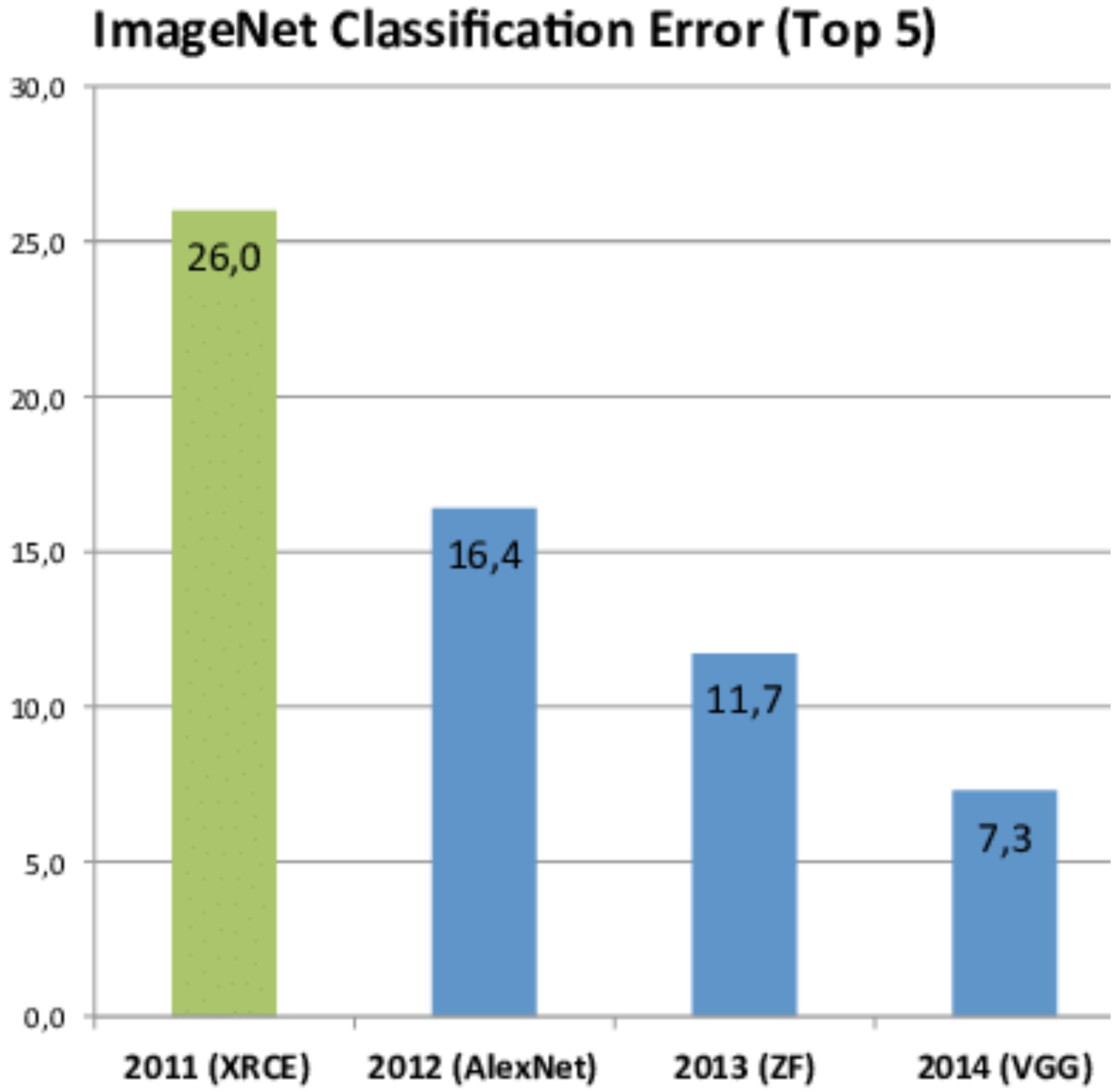




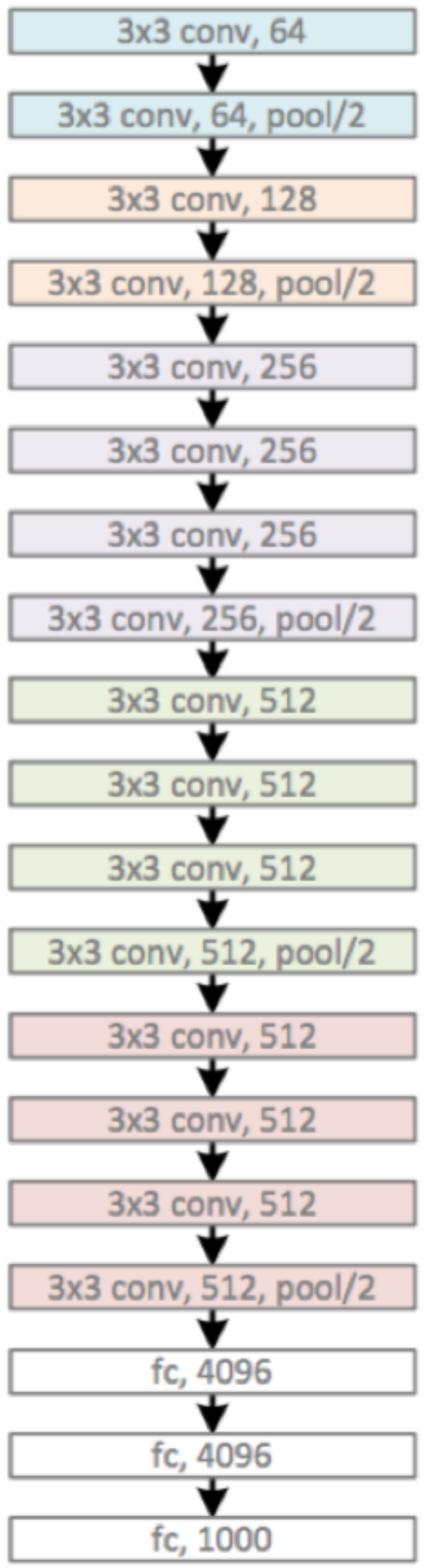
oriented filter

[Slide from Andrea Vedaldi]


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

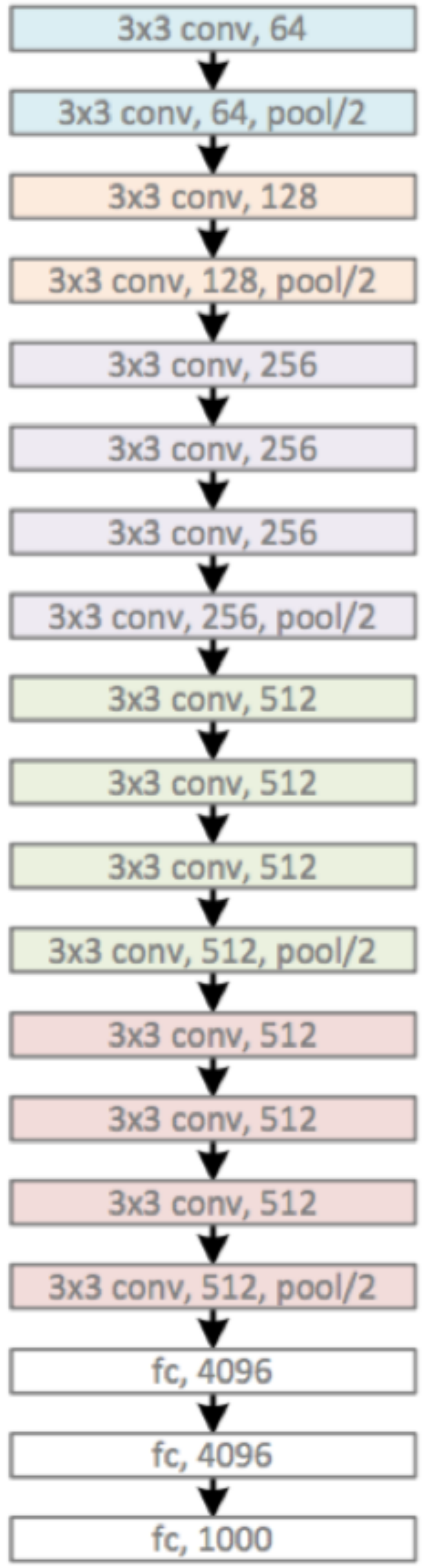


Error: 7.3%

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

VGG-Net [Simonyan & Zisserman, 2015]

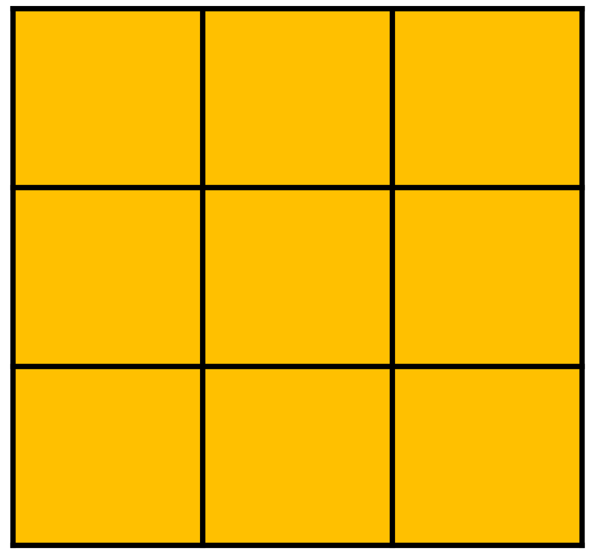
2014: VGG
16 conv. layers



Error: 7.3%

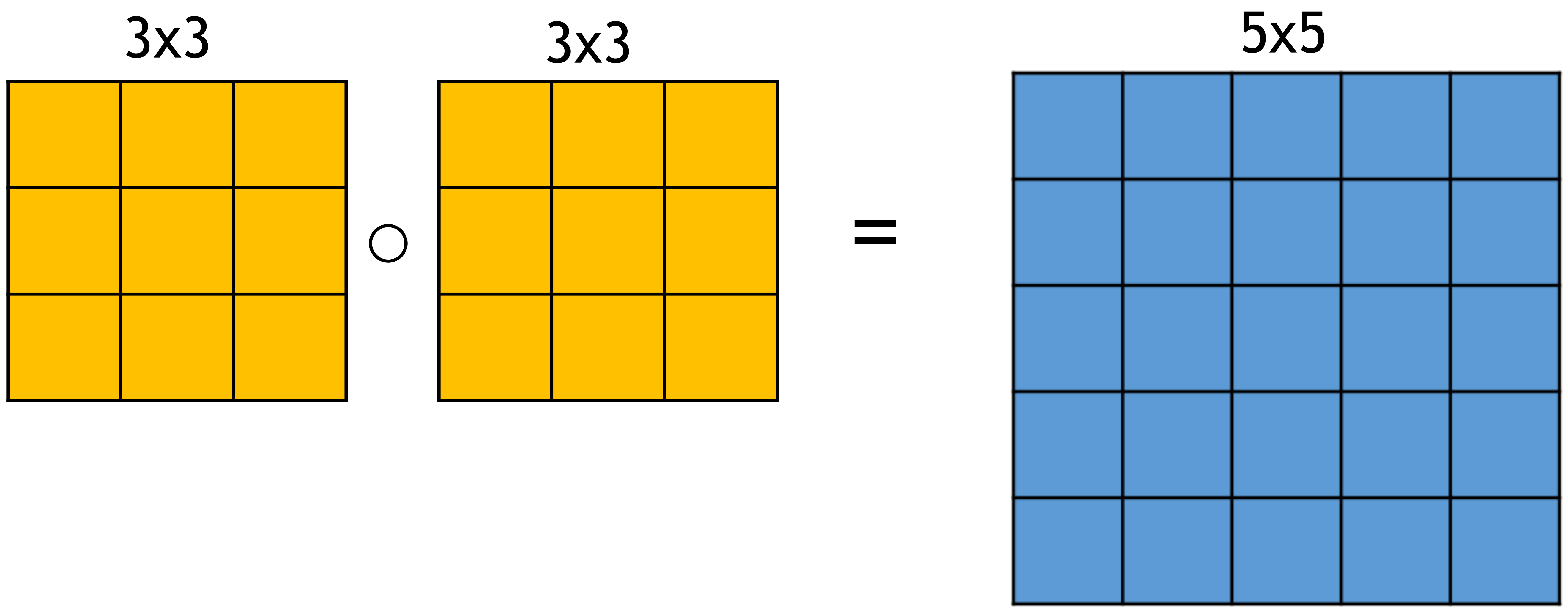
Main developments

- Small convolutional kernels: only 3x3

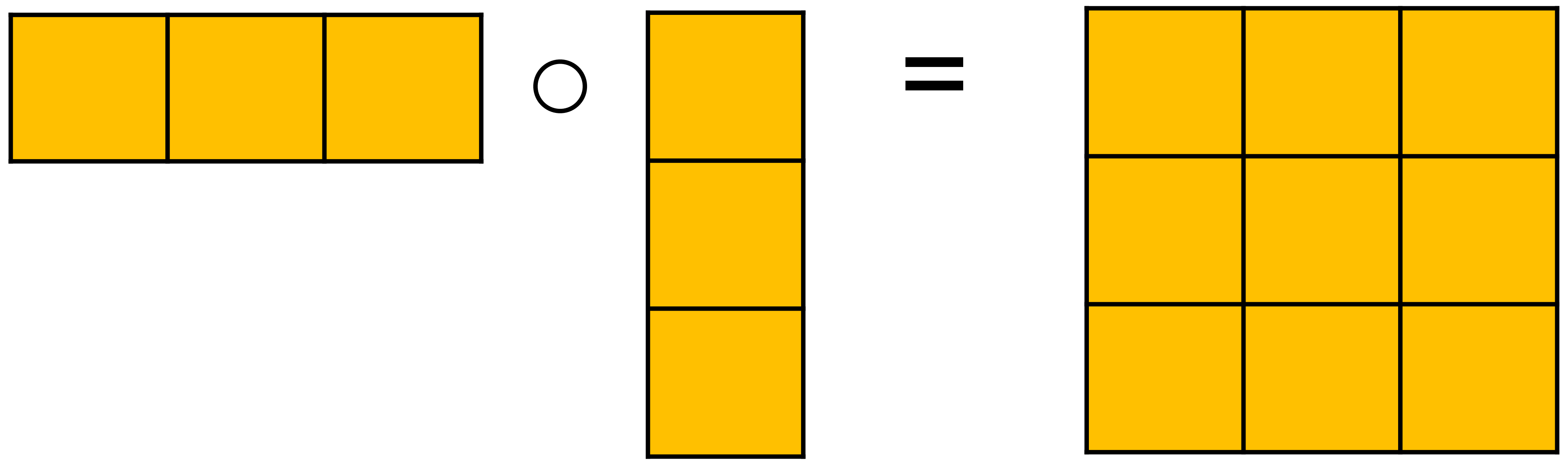


- Increased depth (5 -> 16/19 layers)

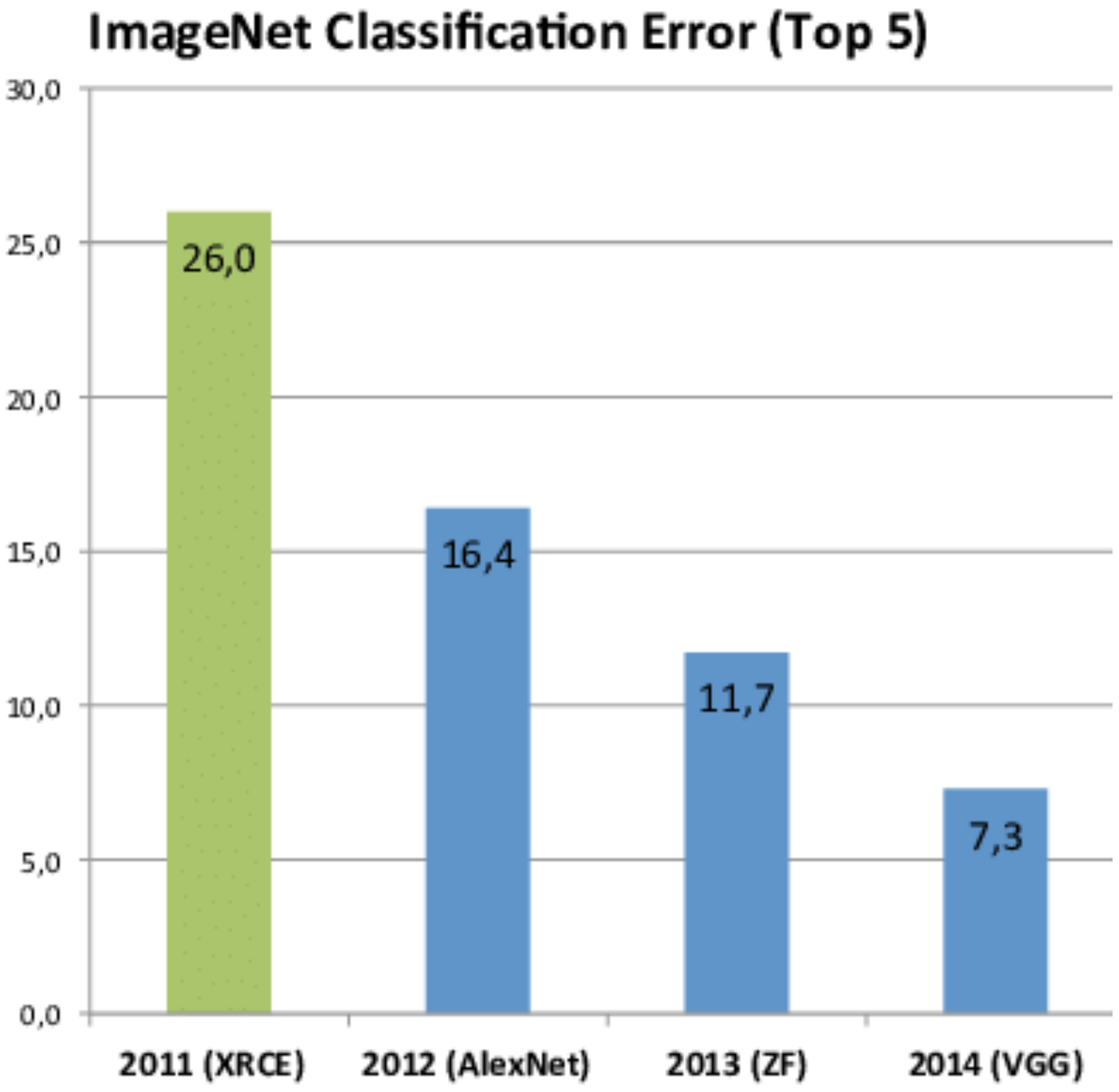
Chaining convolutions

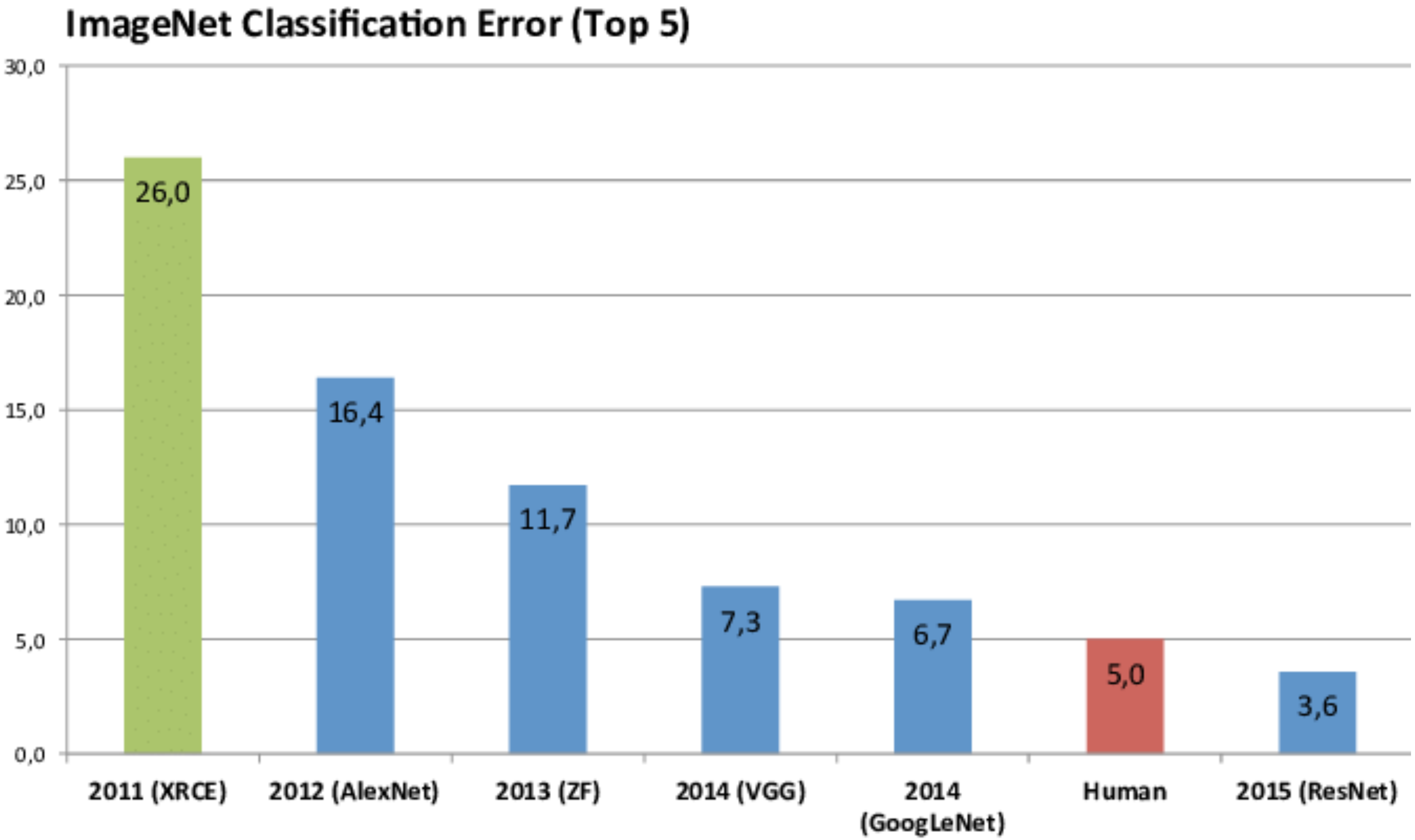


25 coefficients, but only
18 degrees of freedom



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)



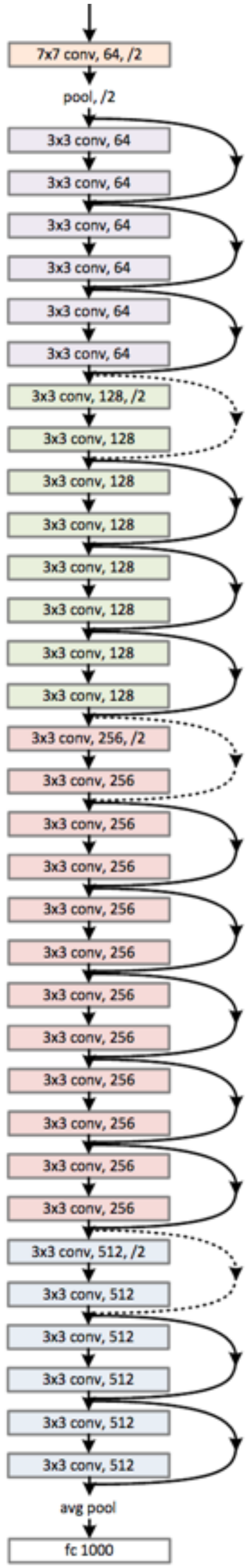
2016: ResNet
>100 conv. layers

Error: 3.6%

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

2016: ResNet
>100 conv. layers

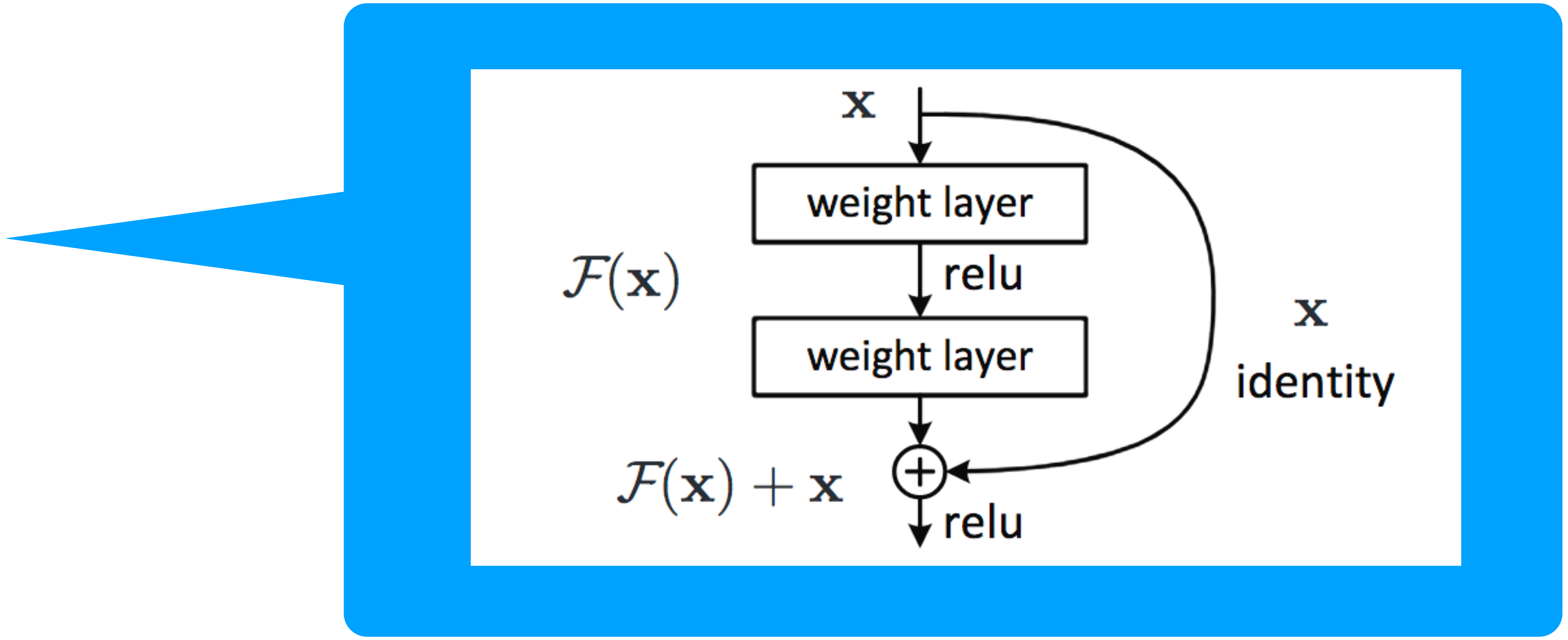
Error: 3.6%



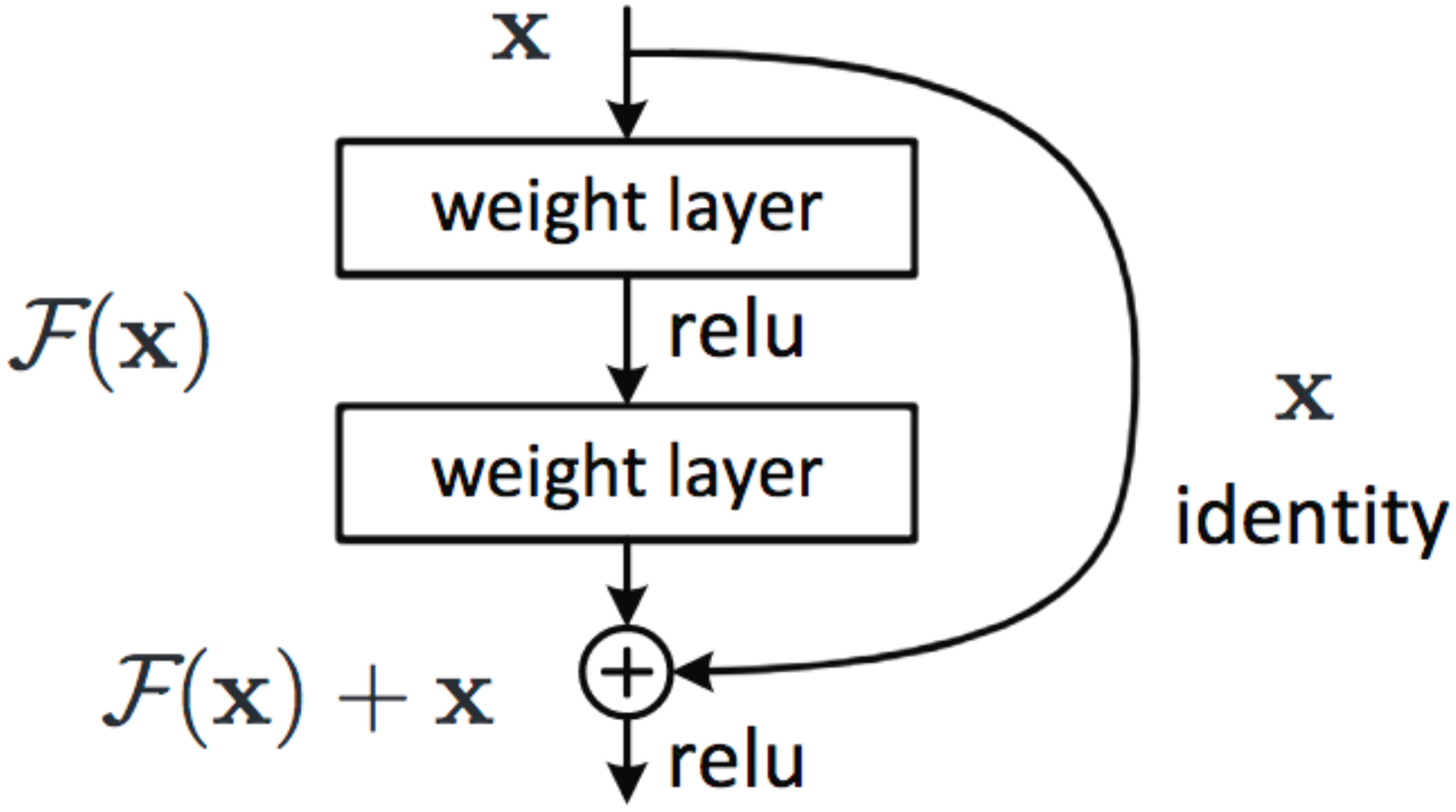
ResNet [He et al, 2016]

Main developments

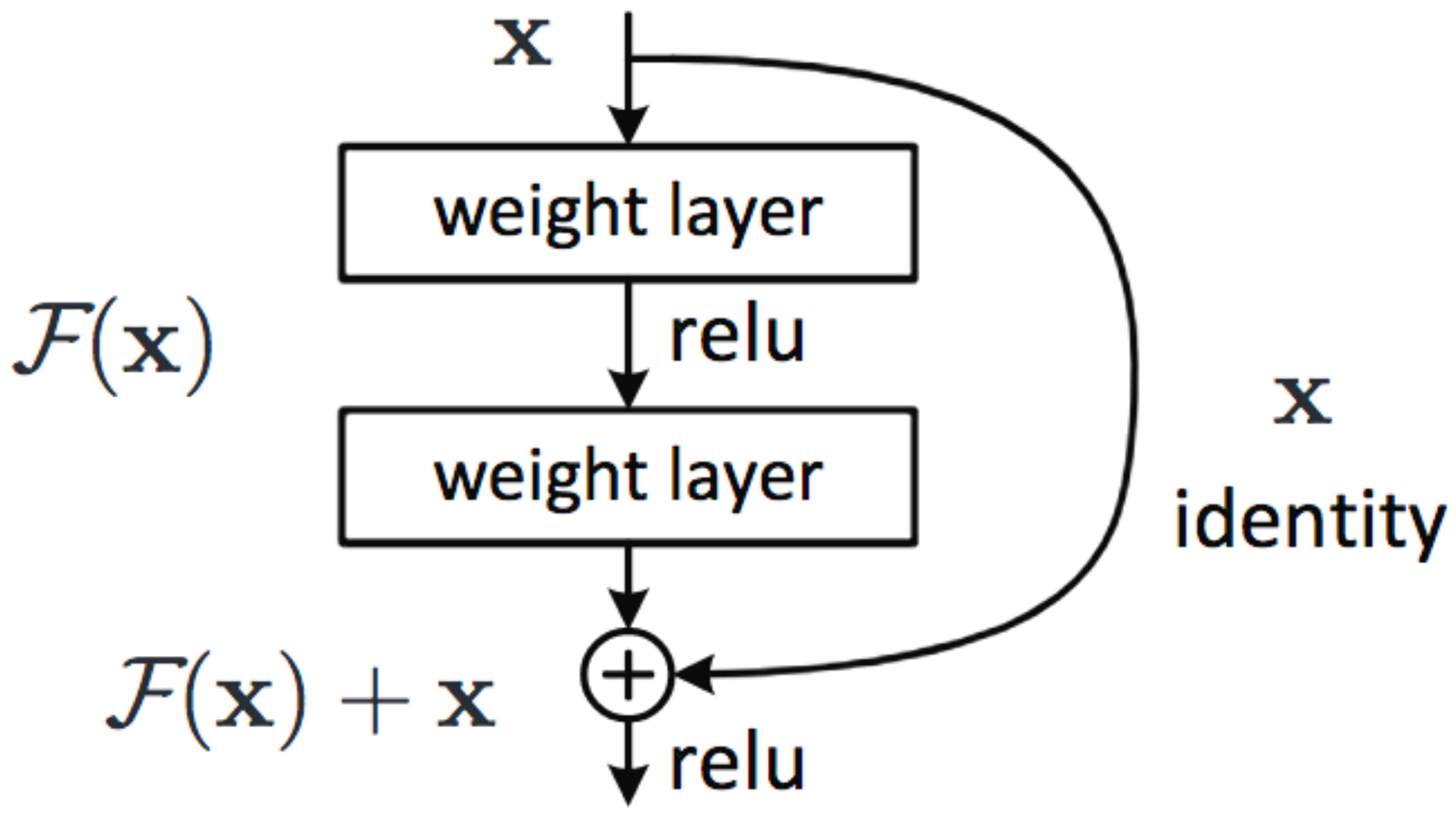
- Increased depth possible through residual blocks



Residual Blocks



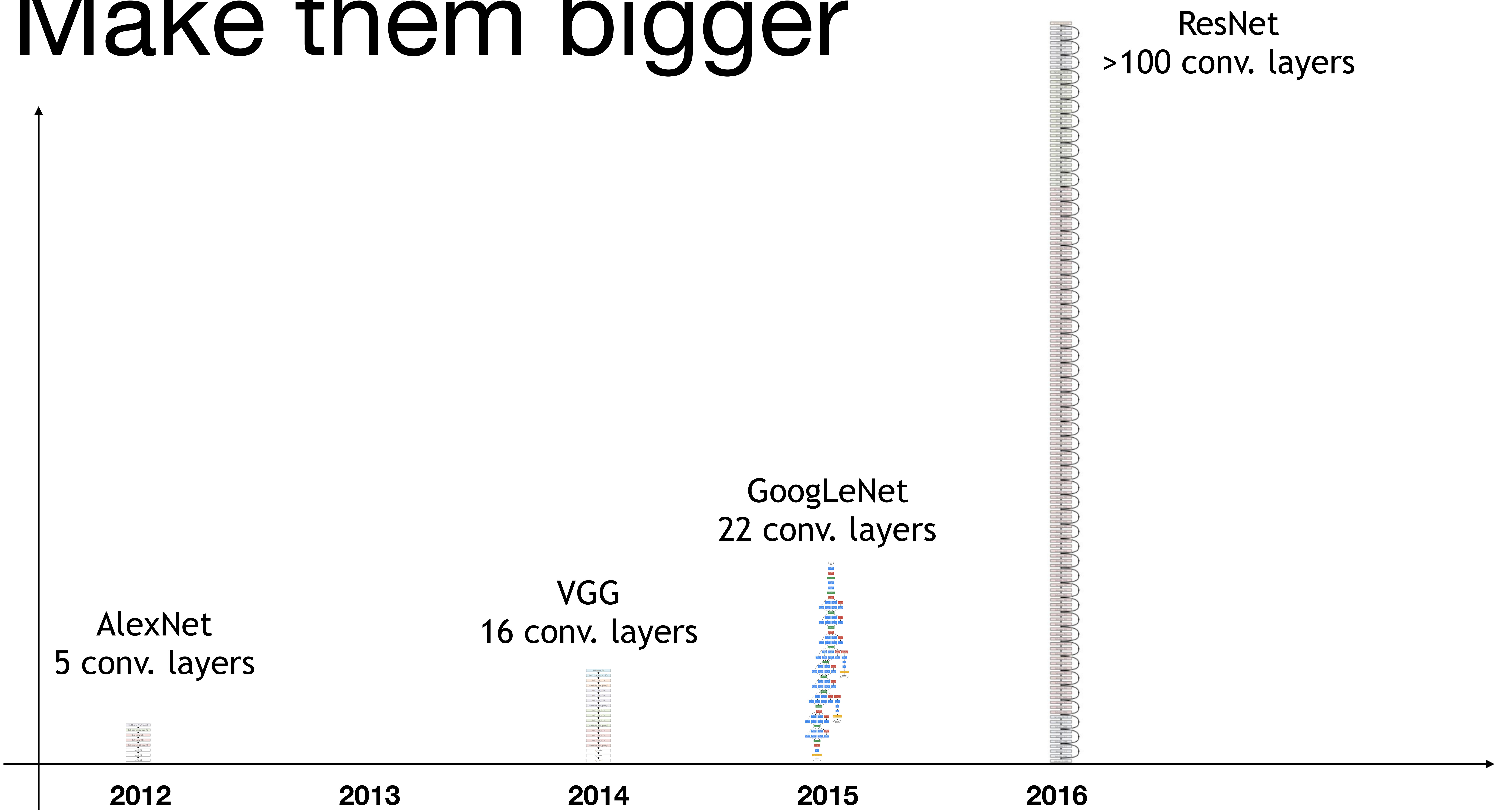
Residual Blocks



Why do they work?

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

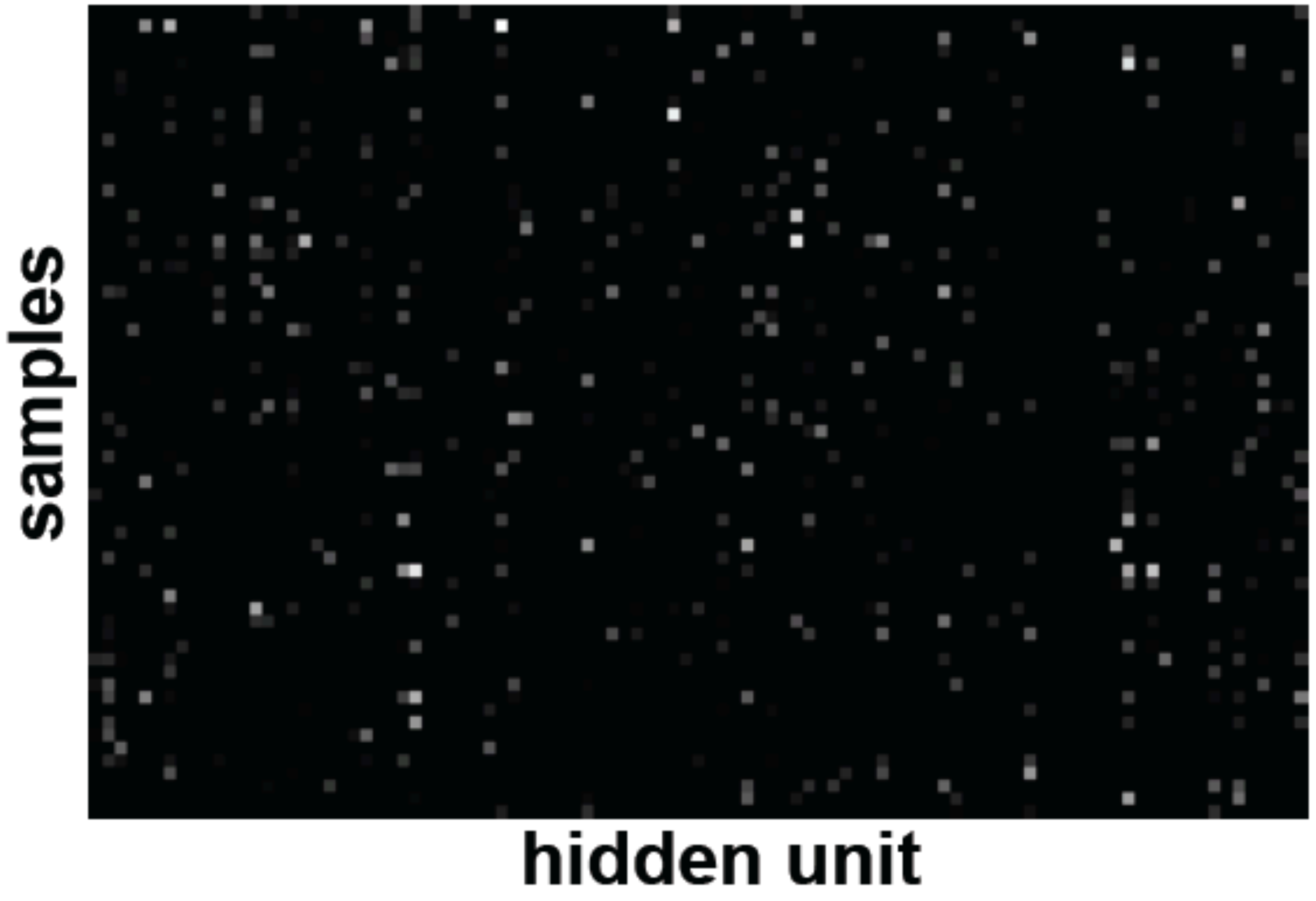
Make them bigger



Some debugging advice

Other good things to know

- Check gradients numerically by finite differences
- Visualize hidden activations — should be uncorrelated and high variance



Good training: hidden units are sparse across samples and across features.

[Derived from slide by Marc'Aurelio Ranzato]

Other good things to know

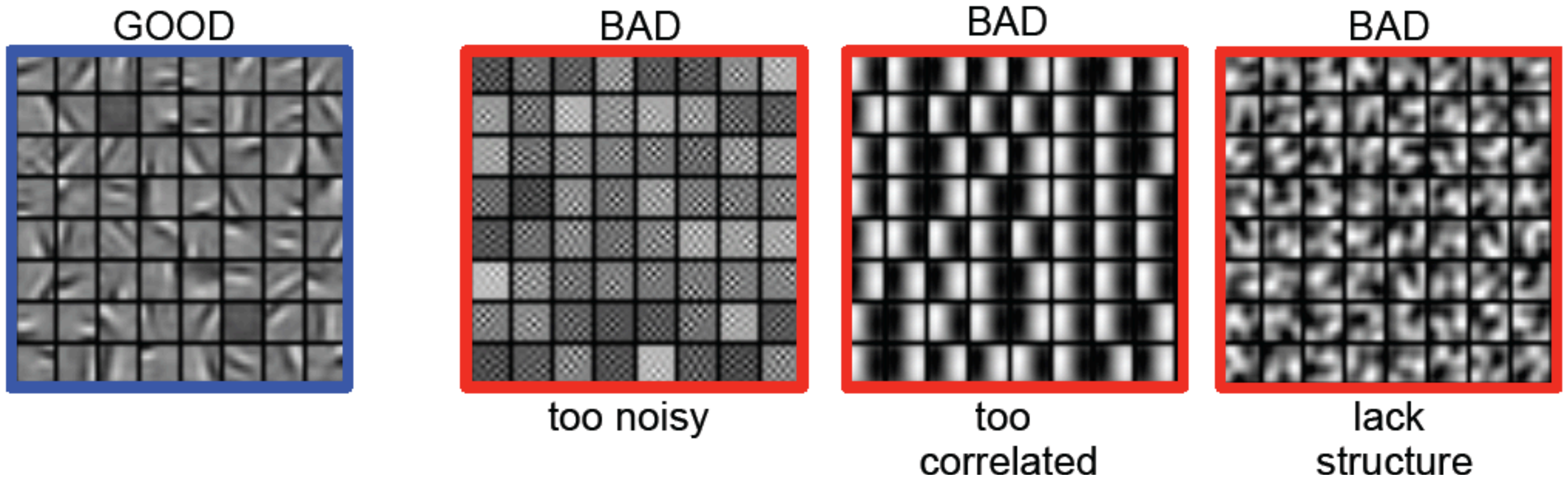
- Check gradients numerically by finite differences
- Visualize hidden activations — should be uncorrelated and high variance

Bad training: many hidden units ignore the input and/or exhibit strong correlations.

[Derived from slide by Marc'Aurelio Ranzato]

Other good things to know

- Check gradients numerically by finite differences
- Visualize hidden activations — should be uncorrelated and high variance
- Visualize filters

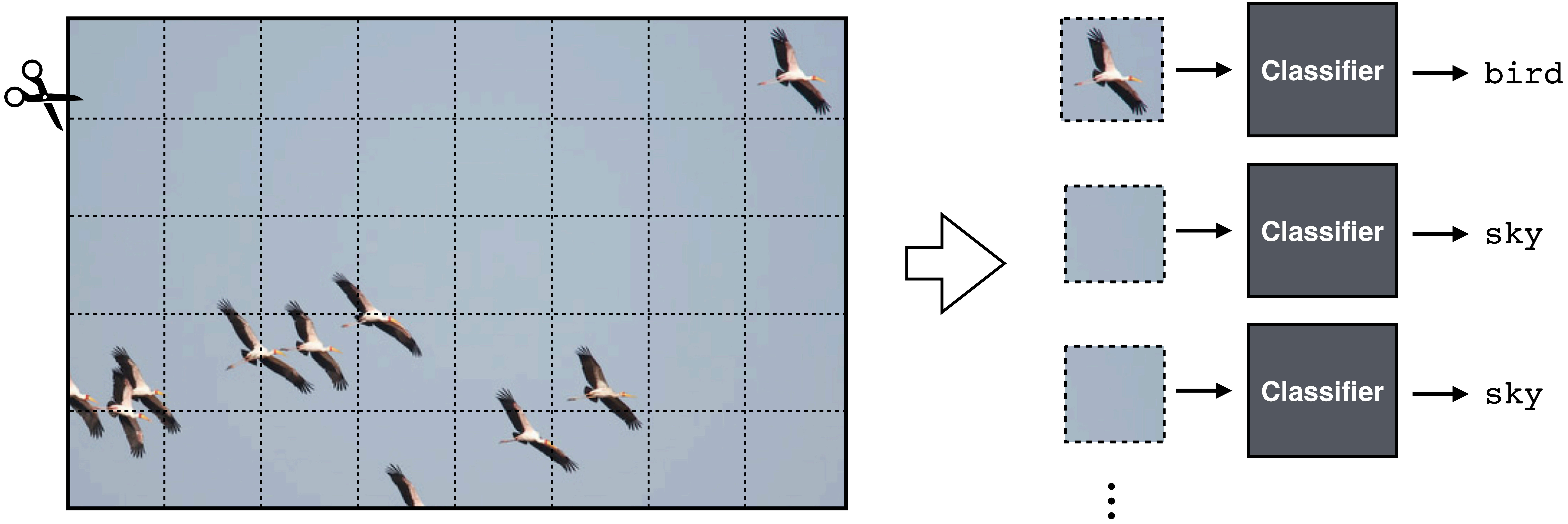


Good training: learned filters exhibit structure and are uncorrelated.

[Derived from slide by Marc'Aurelio Ranzato]

Transformers

Convnets in Disguise



Enduring principles:

1. Chop up signal into patches (divide and conquer)
2. Process each patch identically (and in parallel)

9. CNNs and Spatial Processing

- How to use deep nets for images
- New layer types: convolutional, pooling
- Feature maps and multichannel representations
- Popular architectures: Alexnet, VGG, Resnets
- Getting to know learned filters
- Unit visualization