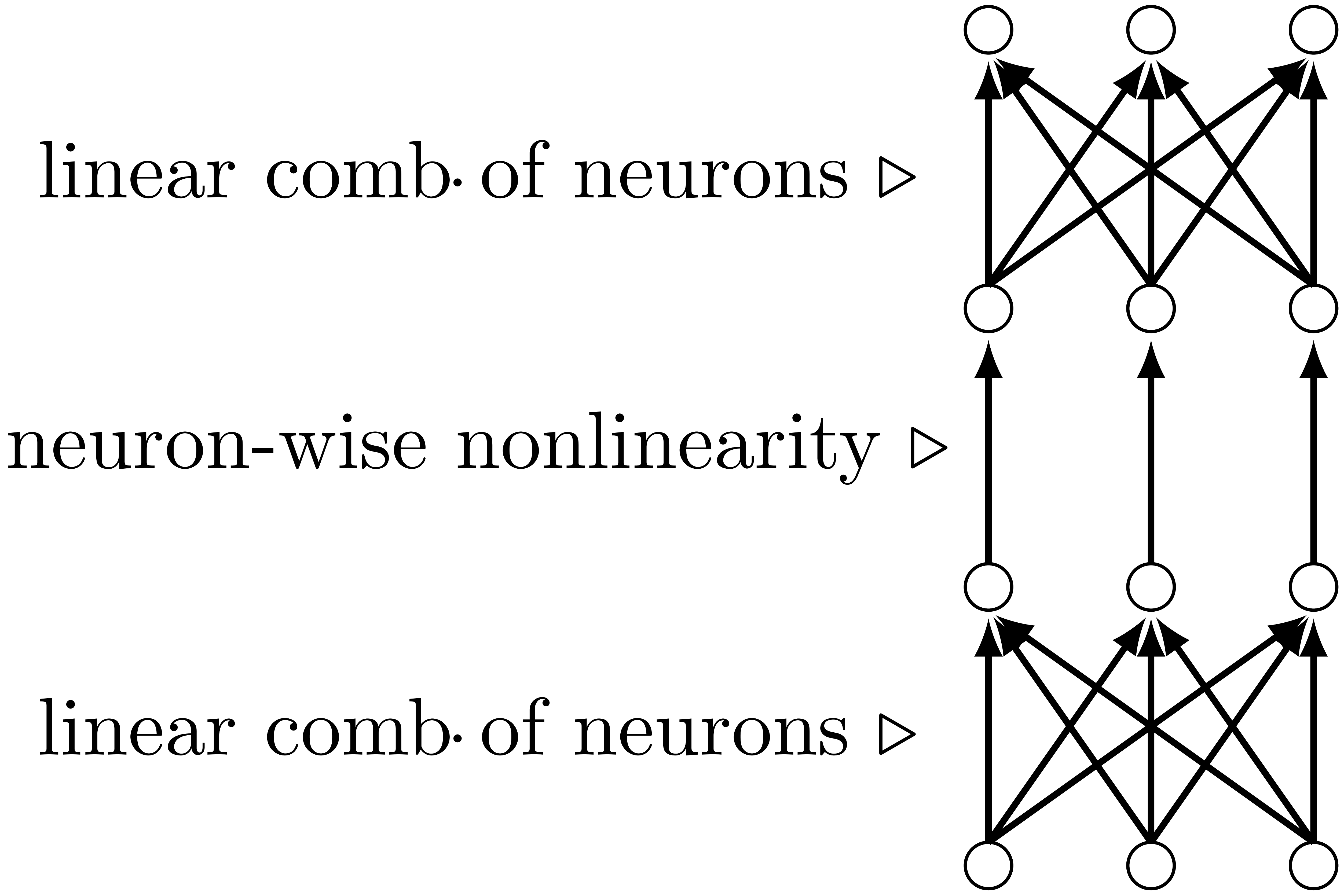
Lecture 4: Architectures for Grids

Speaker: Sara Beery

4. Architectures for Grids

- Why build better architectures?
- Convolutional layers
- Pyramids
- Architecture zoo
- Neural fields and positional encodings

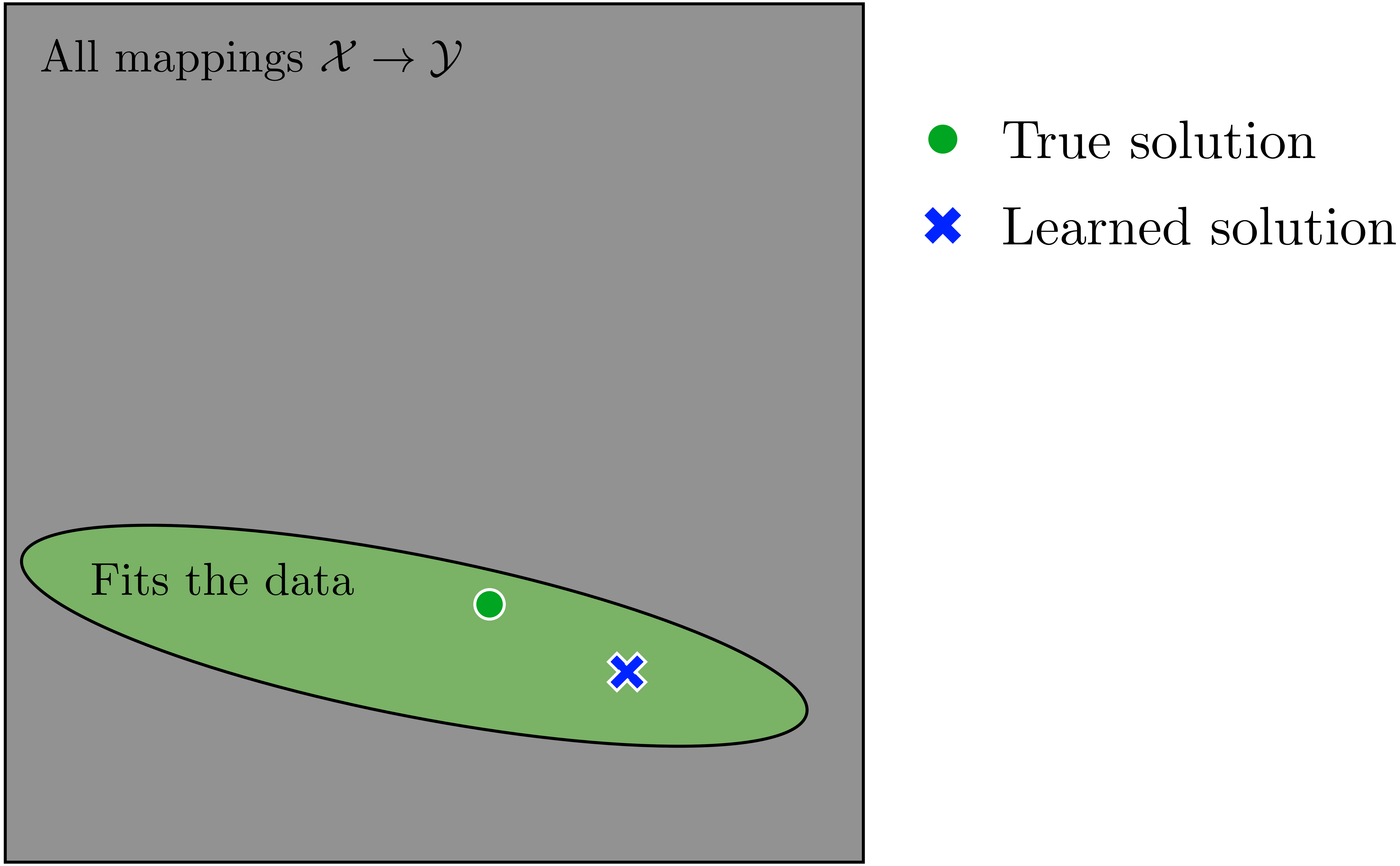
Multilayer Perceptron



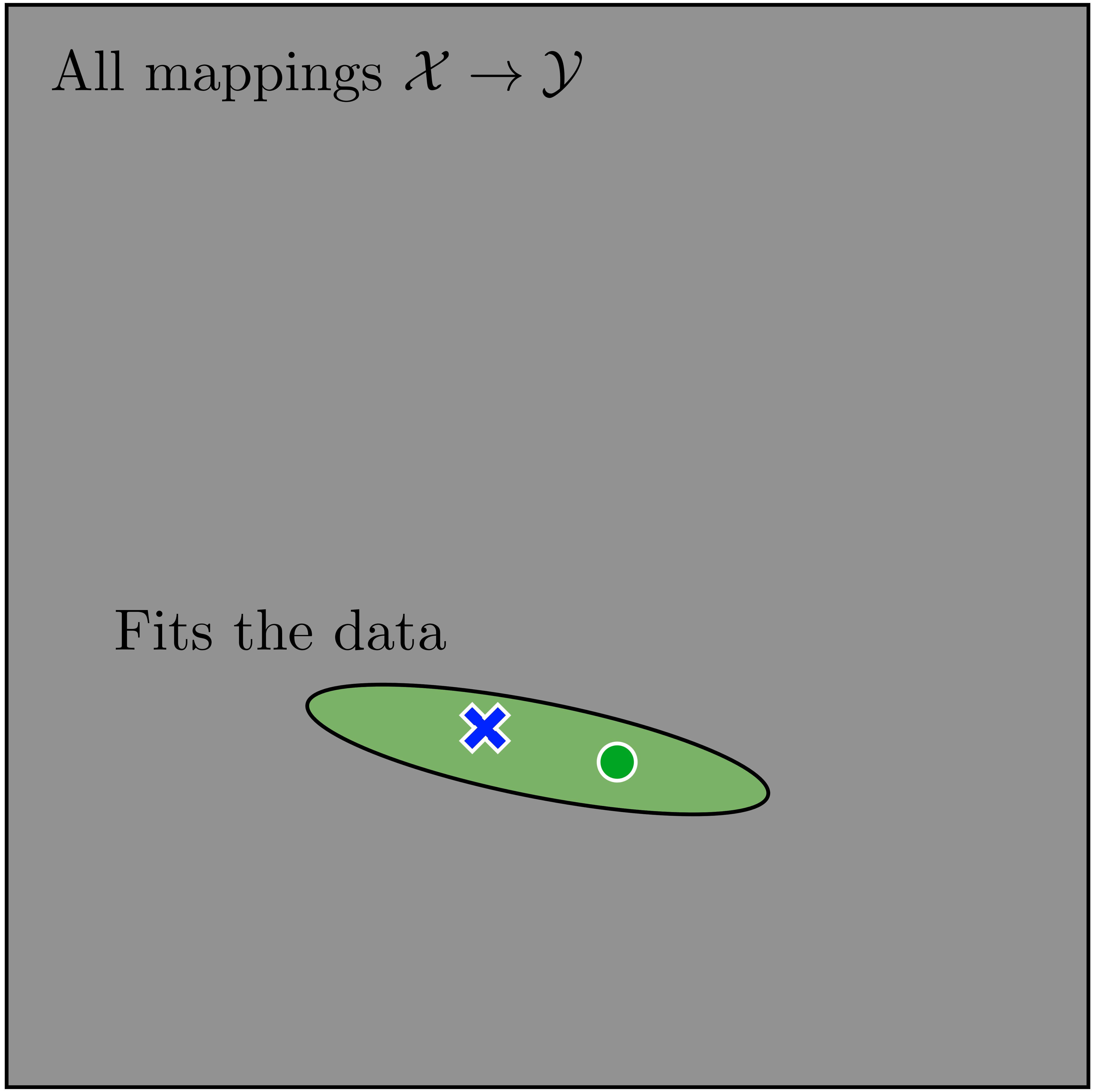
+ Universal
+ Simple (elegant theory)
+ Embarassingly parallel

- Weak inductive biases
- Sample inefficient / data hungry
- Dense (fully-connected) linear layers take a lot of compute

Why use other architectures?



Why use other architectures?

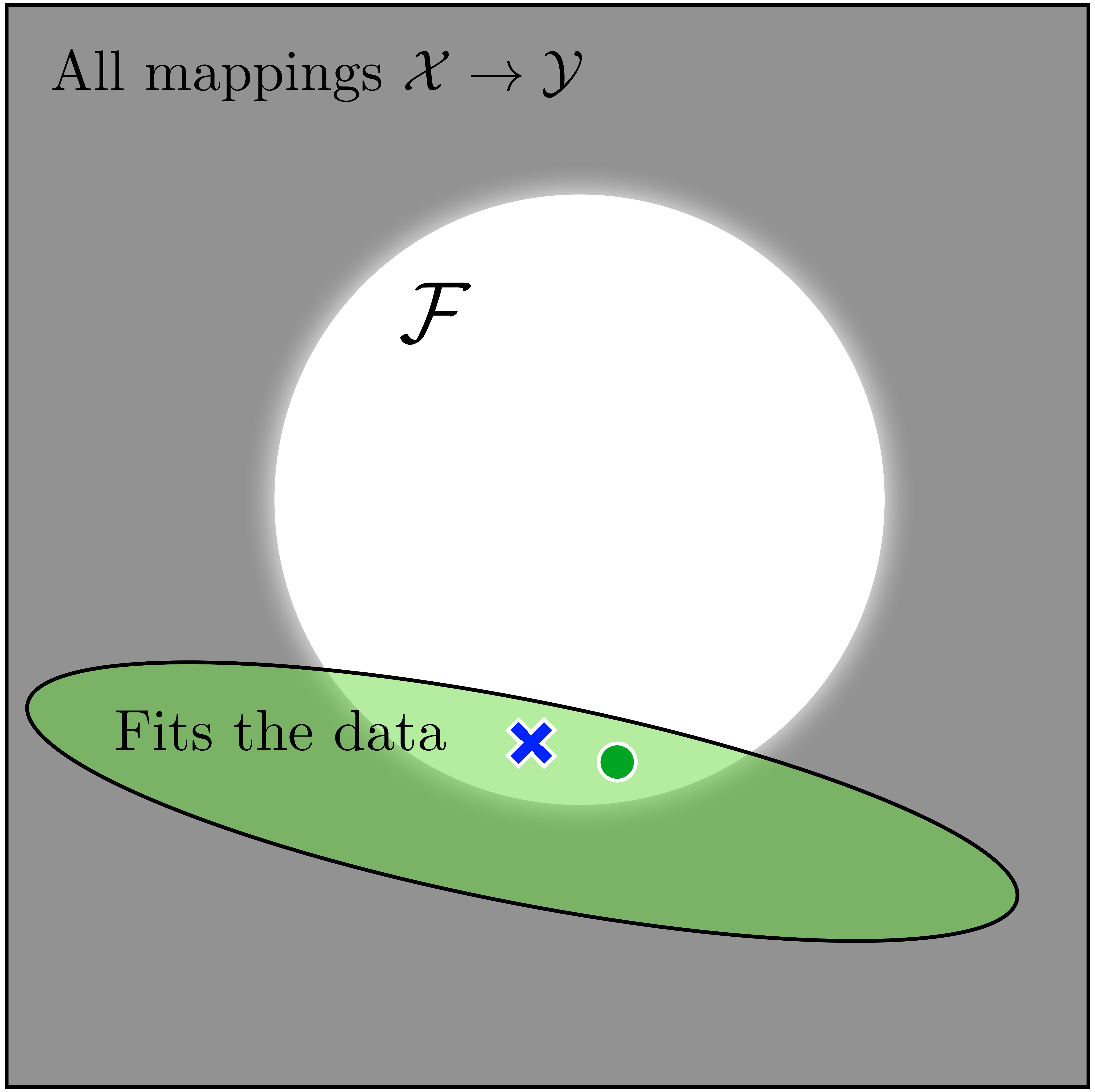


● True solution

✖ Learned solution

Effect of adding more data.

Why use other architectures?



$\mathcal{F}$ – Hypothesis space

● True solution

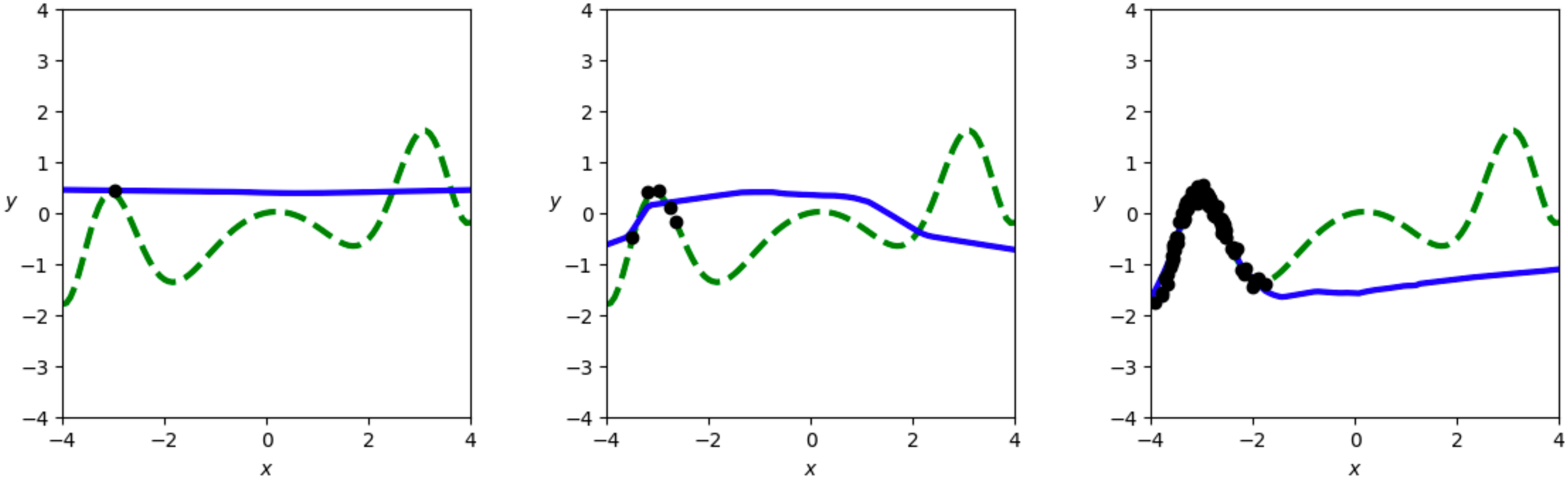
✖ Learned solution

Less data, better architecture.

We can pin down truth *either* by adding more data, or by using a more constrained architecture.

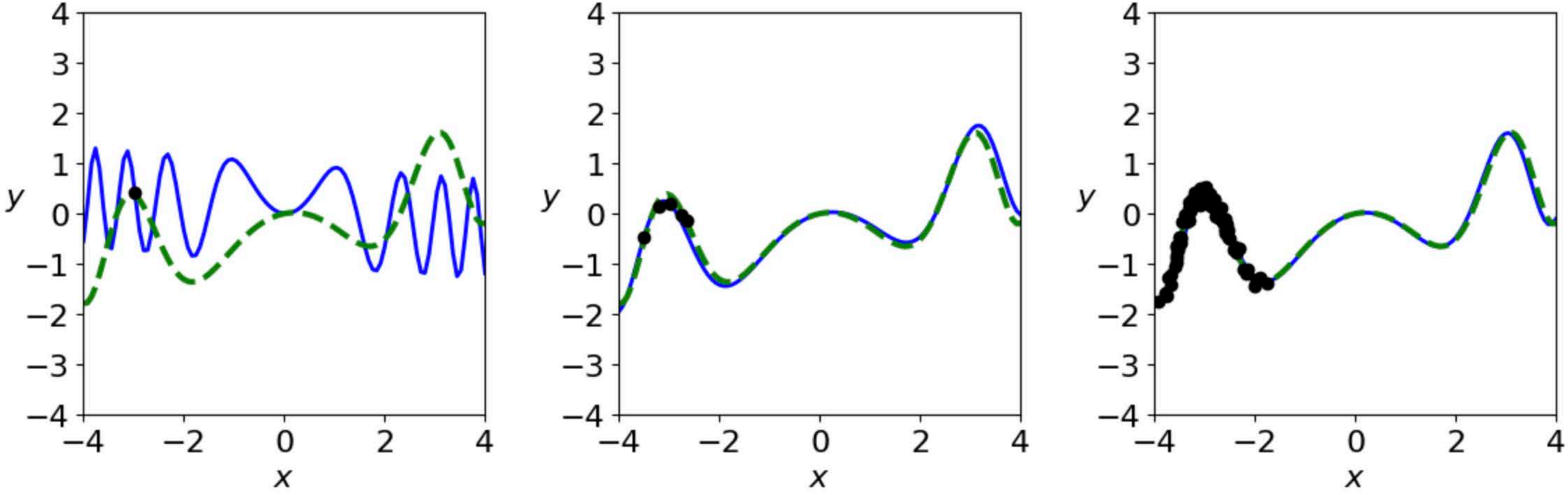
$$y = f(x)$$

f: 5 layer ReLU-net



- True solution
- Learned solution
- Training data

$$y = ax + \sin(bx^2)$$



- True solution
- Learned solution
- Training data

Architectures enable us to generalize *outside the training distribution.*

A good architecture is one that can represent the true function and is otherwise minimal (and is also easy to search over via gradient-based learning, easy to parallelize, fast on GPU, etc).

Preview: better architectures can approximate important function classes more efficiently.

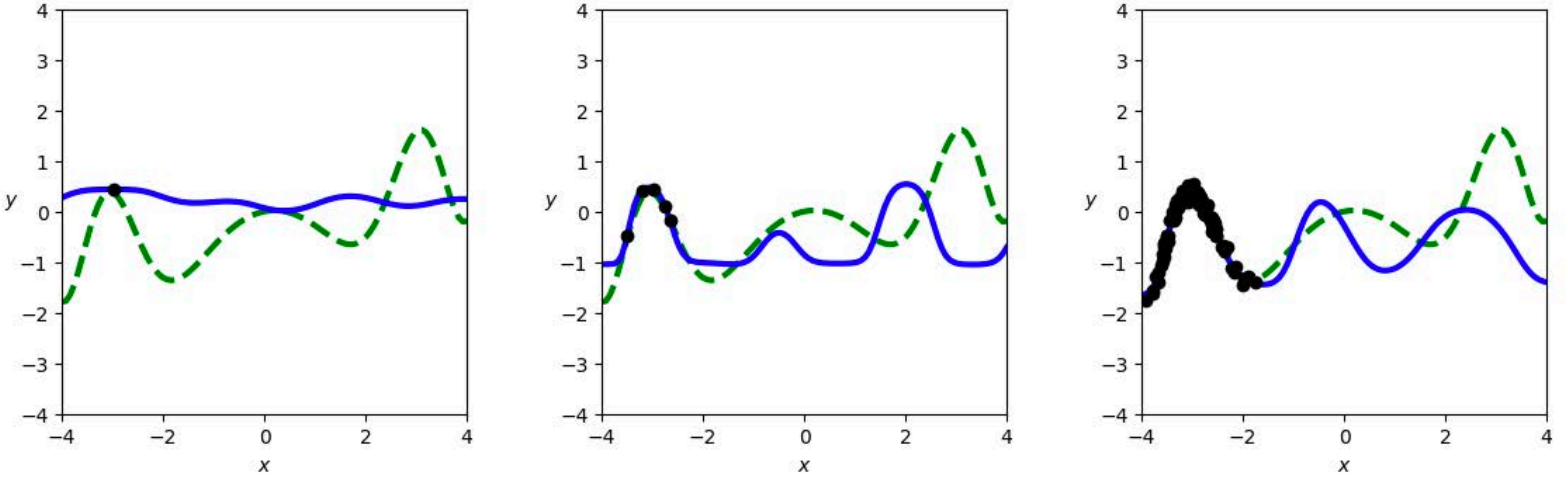
Goal: Fit this image (a function x,y —> I) | **ReLU-net** | **TanH-net** | **ReLU P.E.** | **RBF ReLU** | **SIREN**

(Note: this result may be due to improved approximation ability but it might also be due to improved optimization ability; these two effects are typically coupled in experiments)

[Sitzmann*, Martel*, Bergman, Lindell, Wetzstein, NeurIPS 2020]

$y = f(x)$

f: 5 layer sin-net (SIREN)

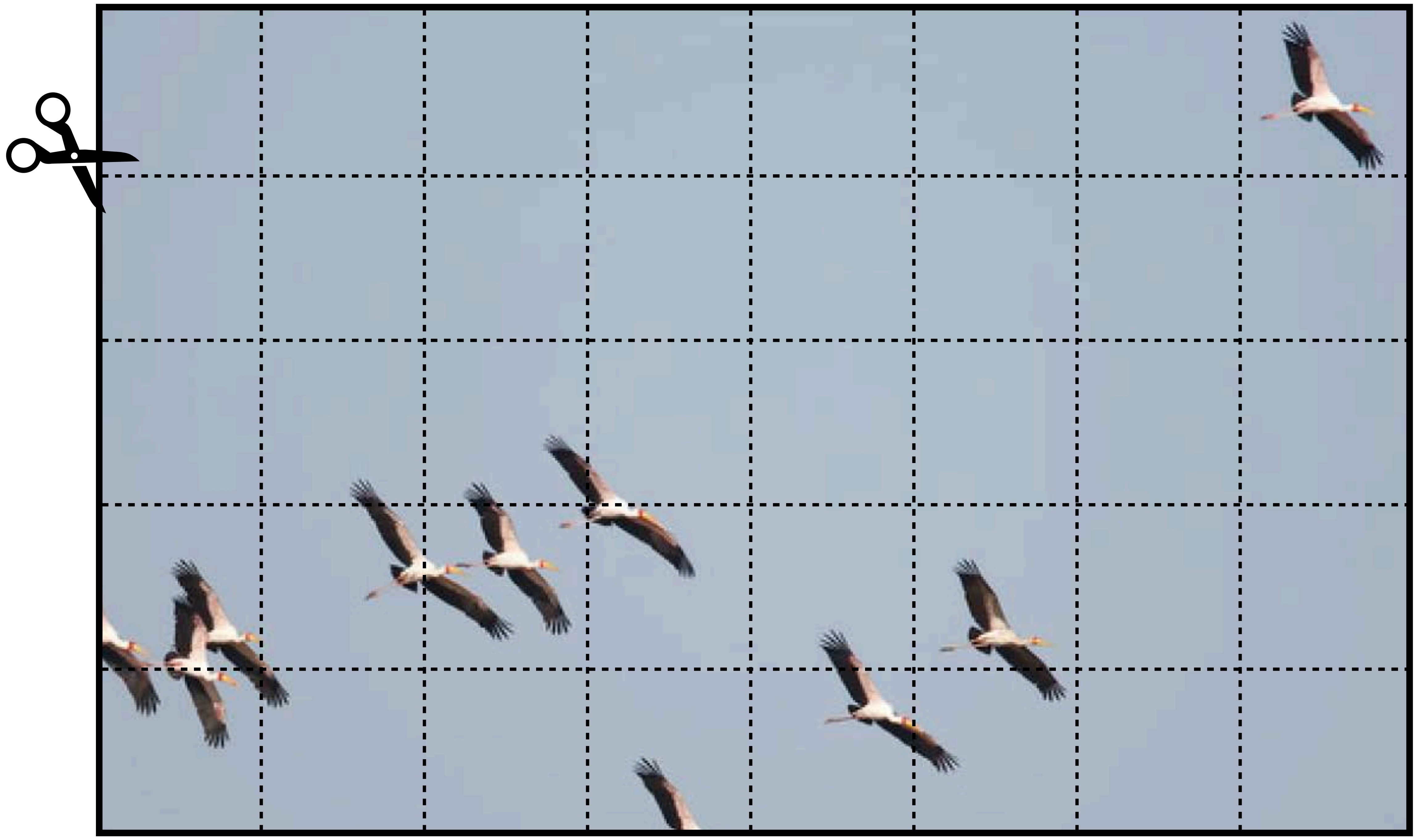


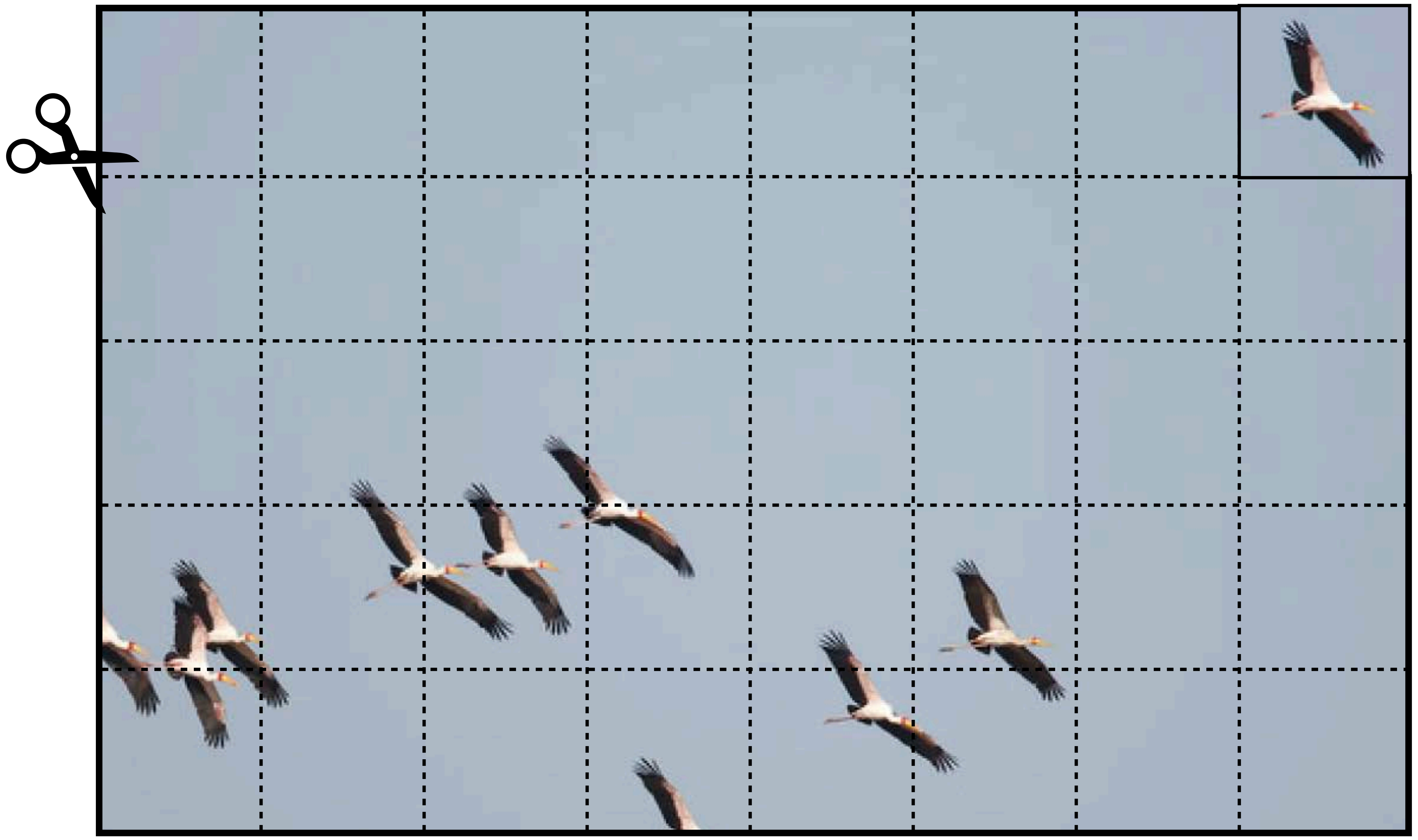
- True solution
- Learned solution
- Training data

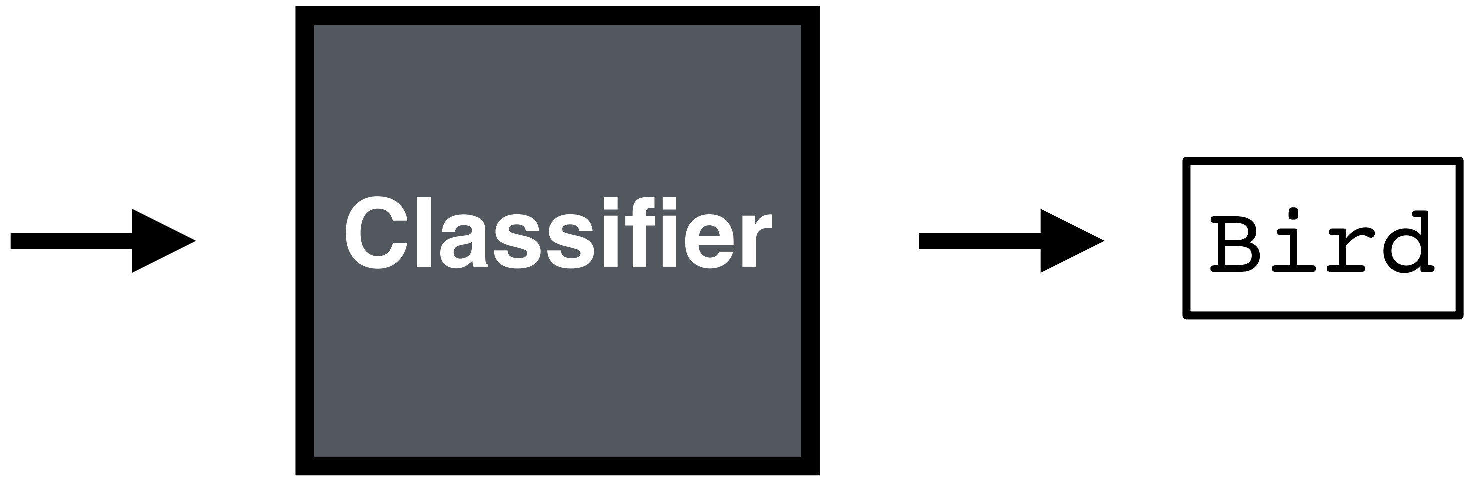
[Sitzmann*, Martel*, Bergman, Lindell, Wetzstein, NeurIPS 2020]

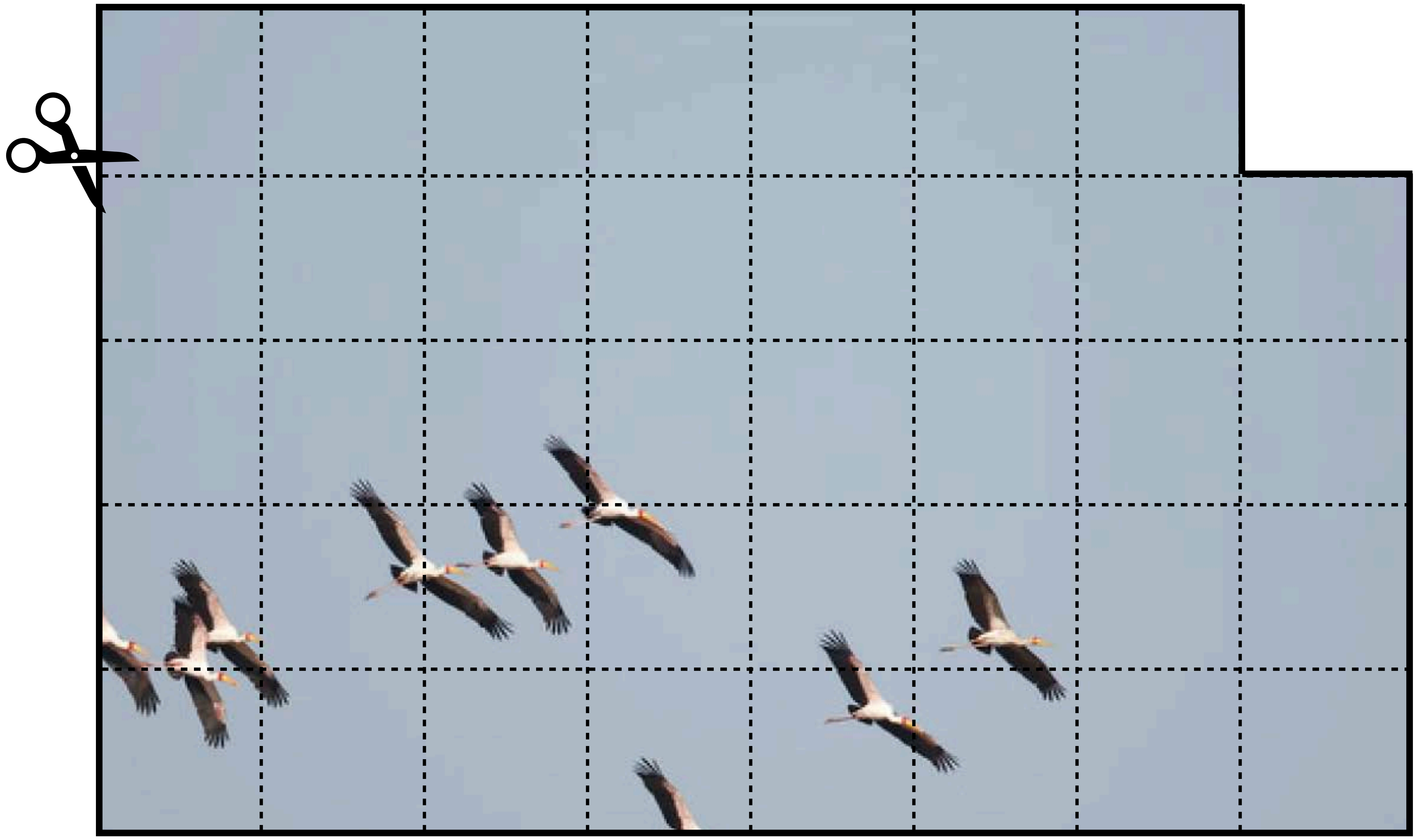
Convolutional Neural Networks

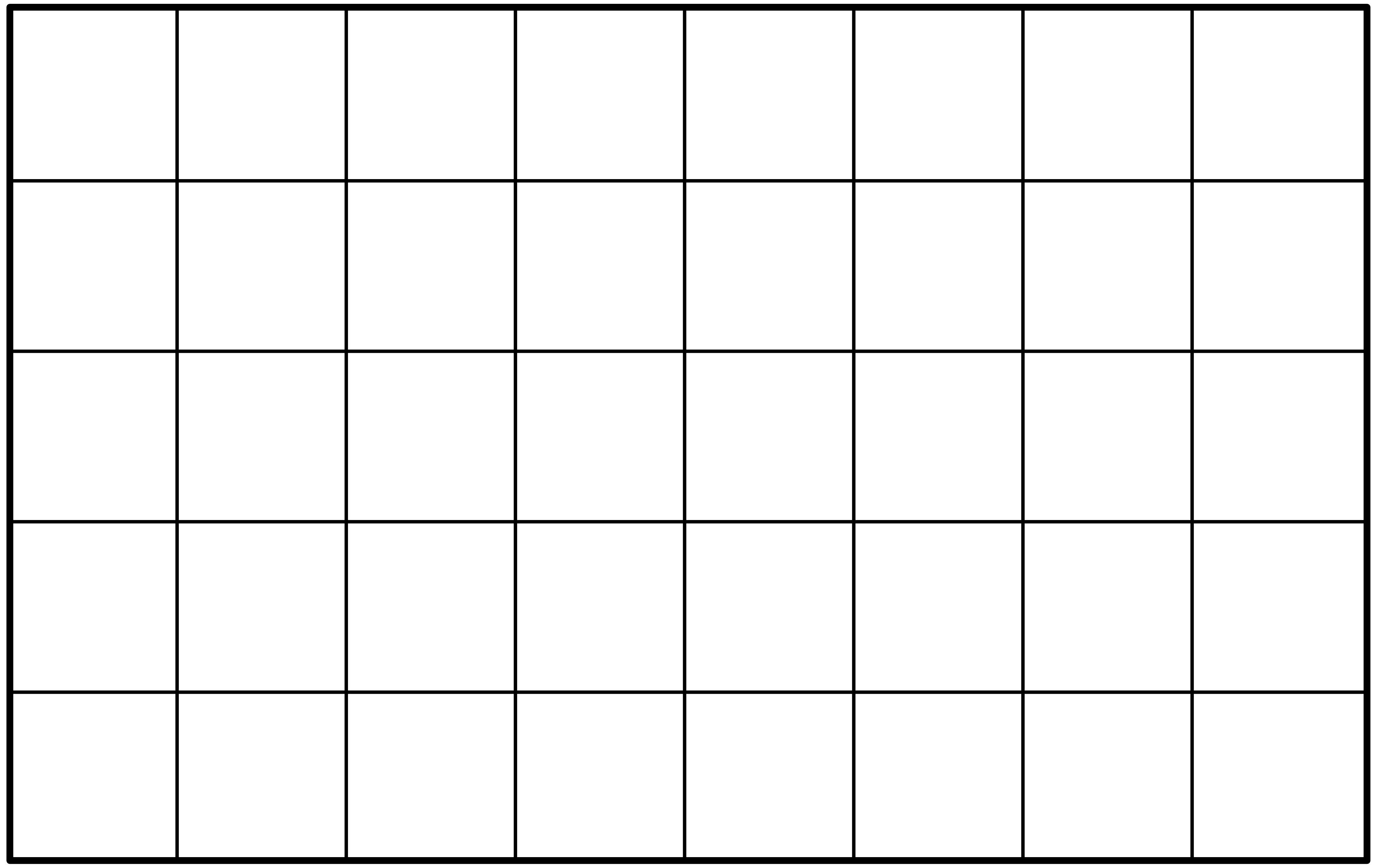
Photo credit: Fredo Durand



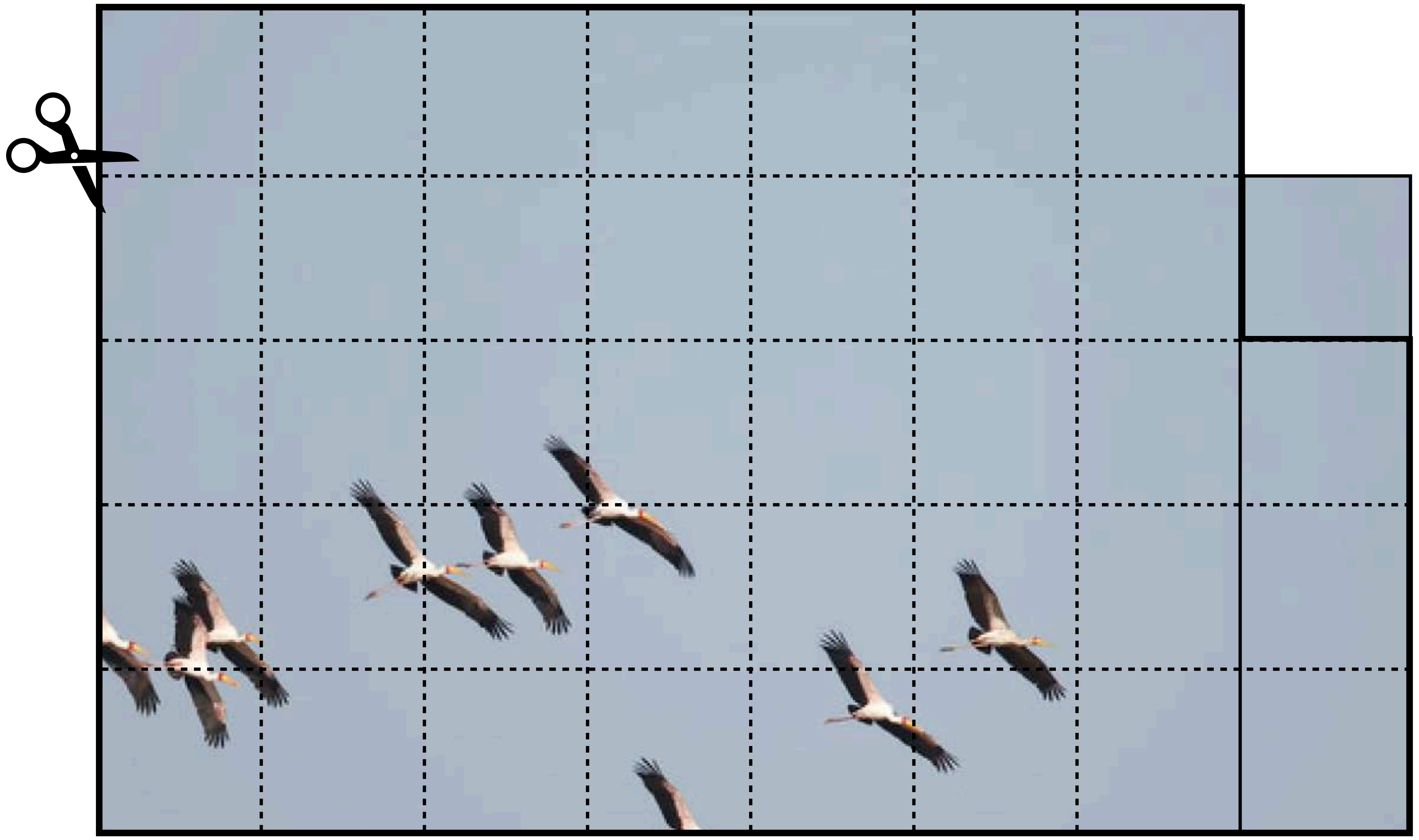


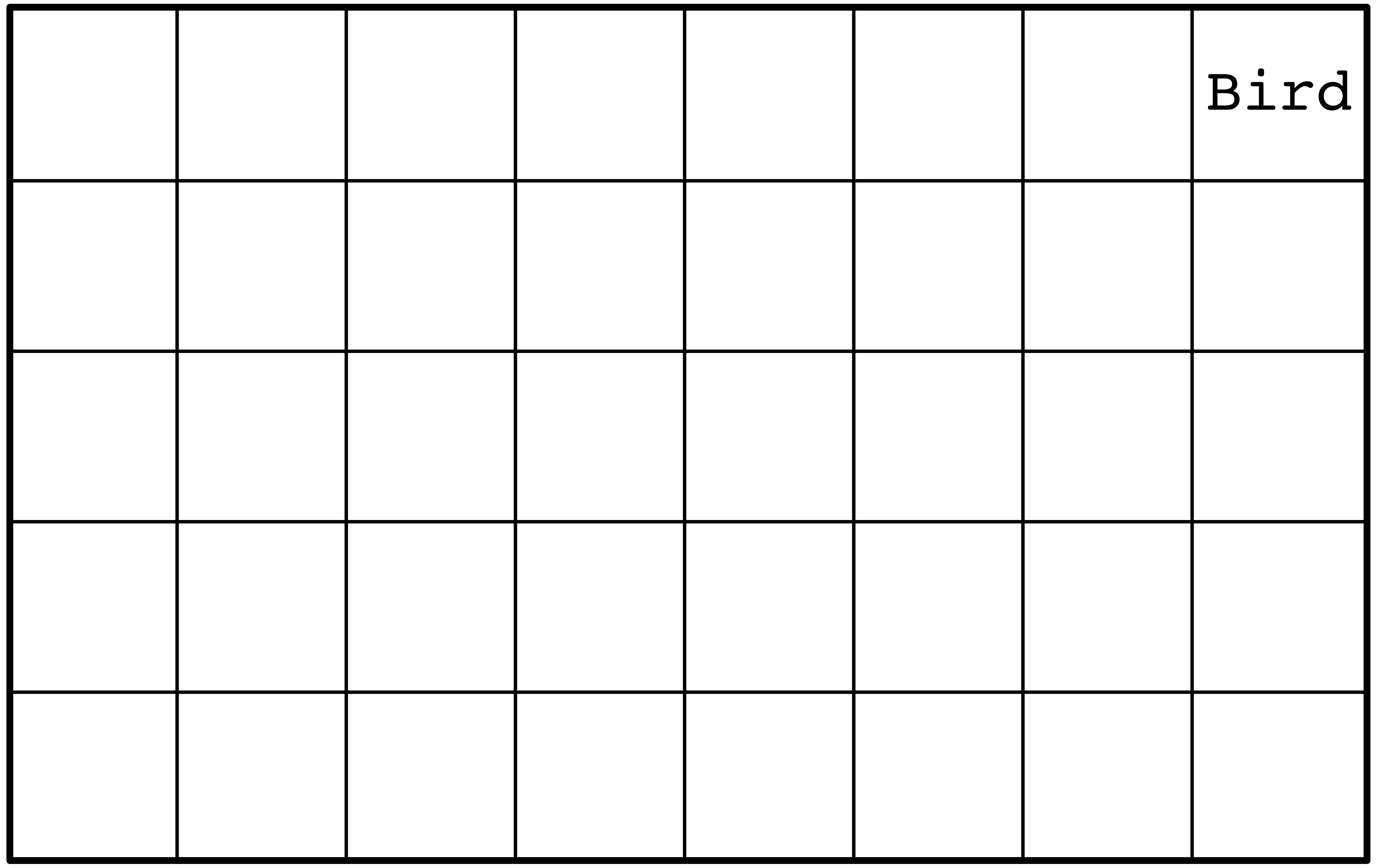

Classifier
Bird

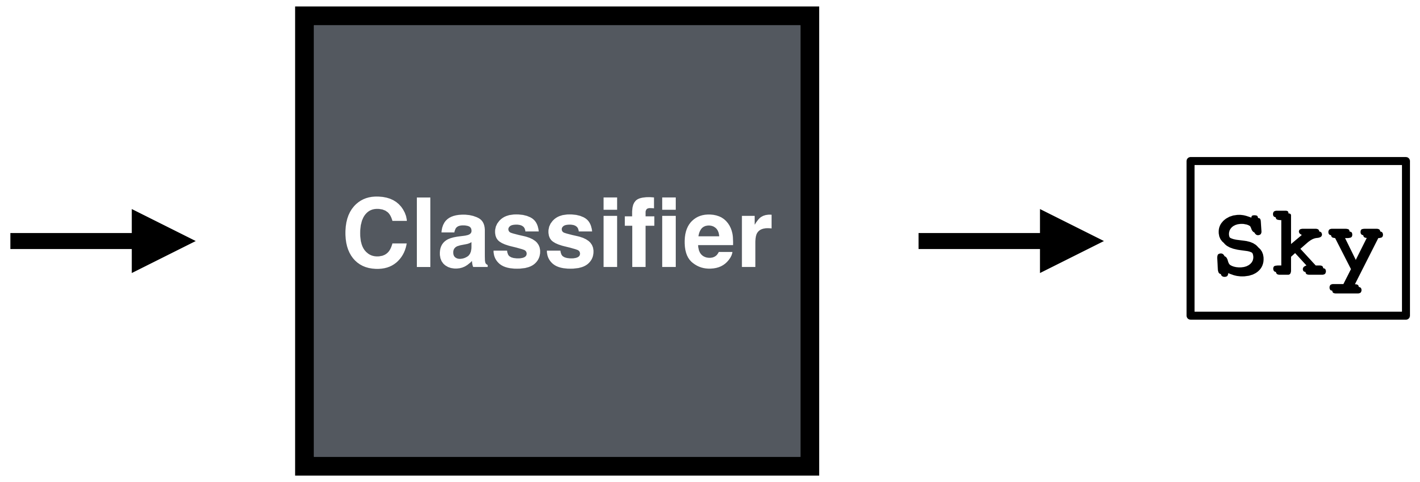


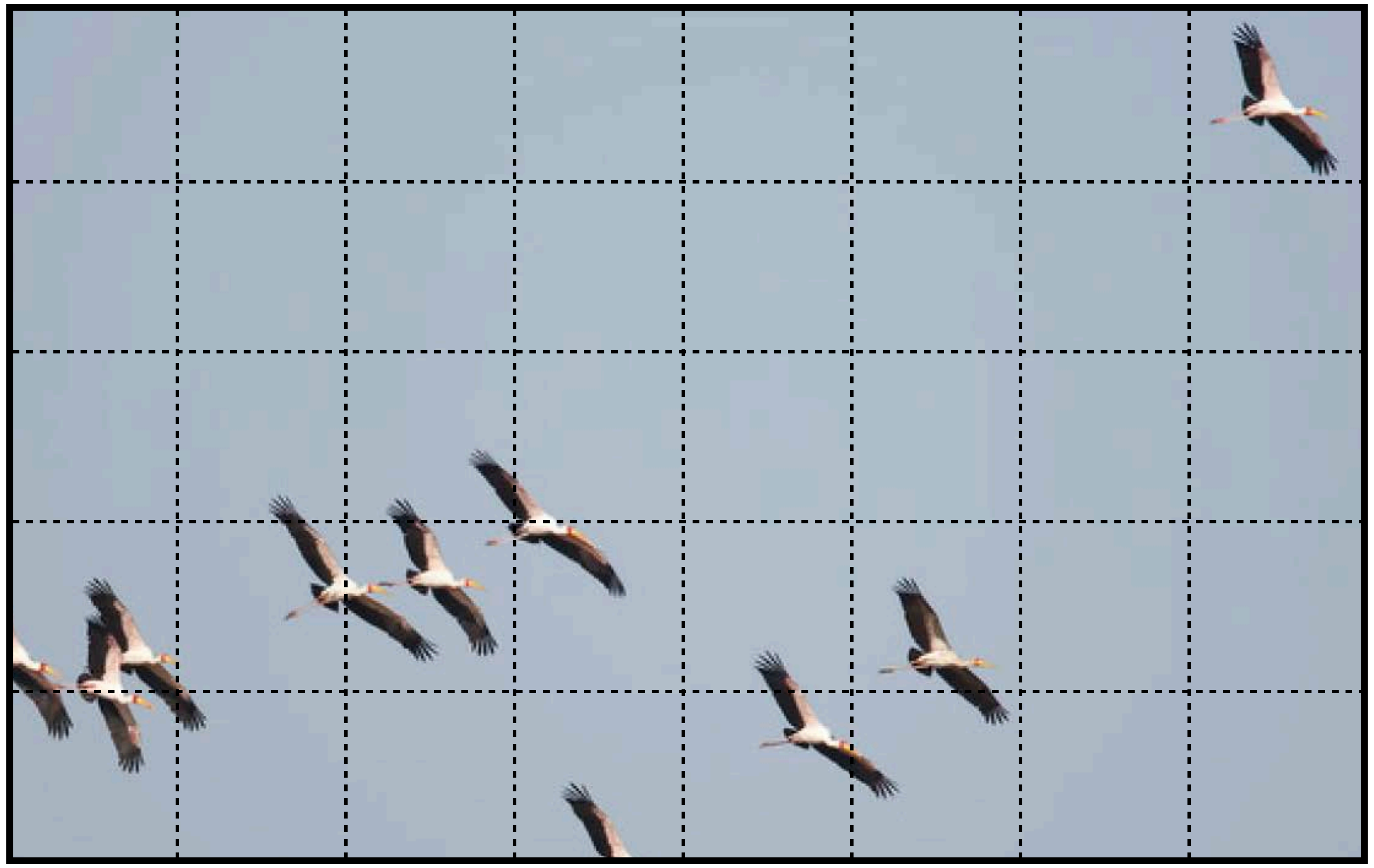



Classifier
Bird

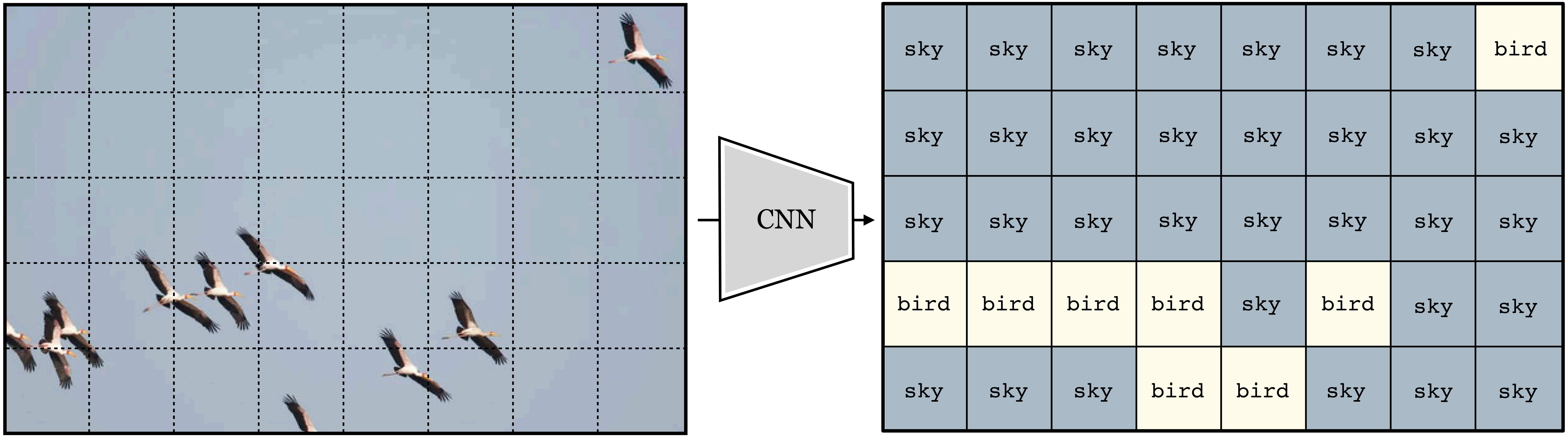



Bird


Classifier
Sky



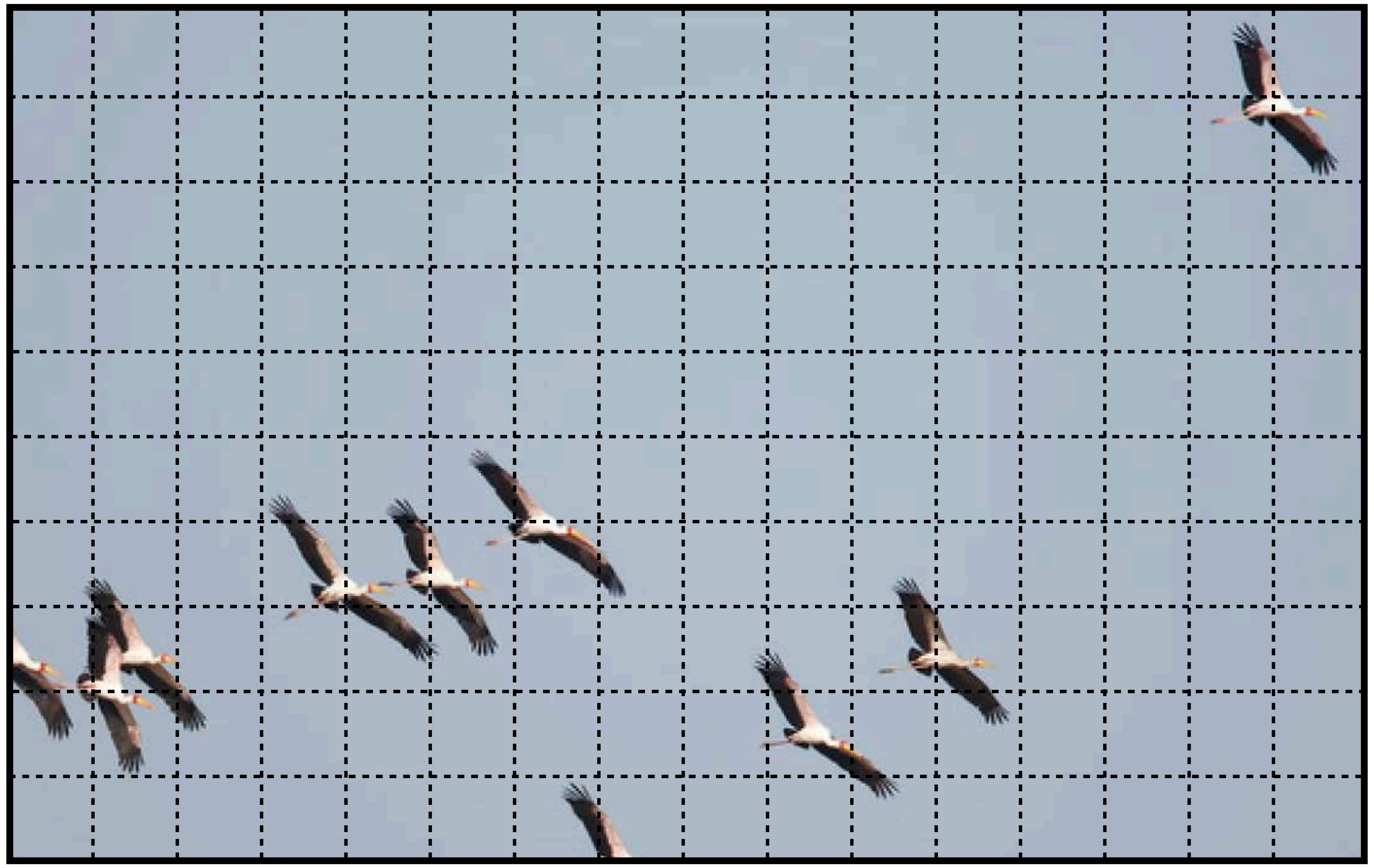
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Bird
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Bird	Bird	Bird	Sky	Bird	Sky	Sky	Sky
Sky	Sky	Sky	Bird	Sky	Sky	Sky	Sky



Problem:

What if objects don't fit neatly into these patches?

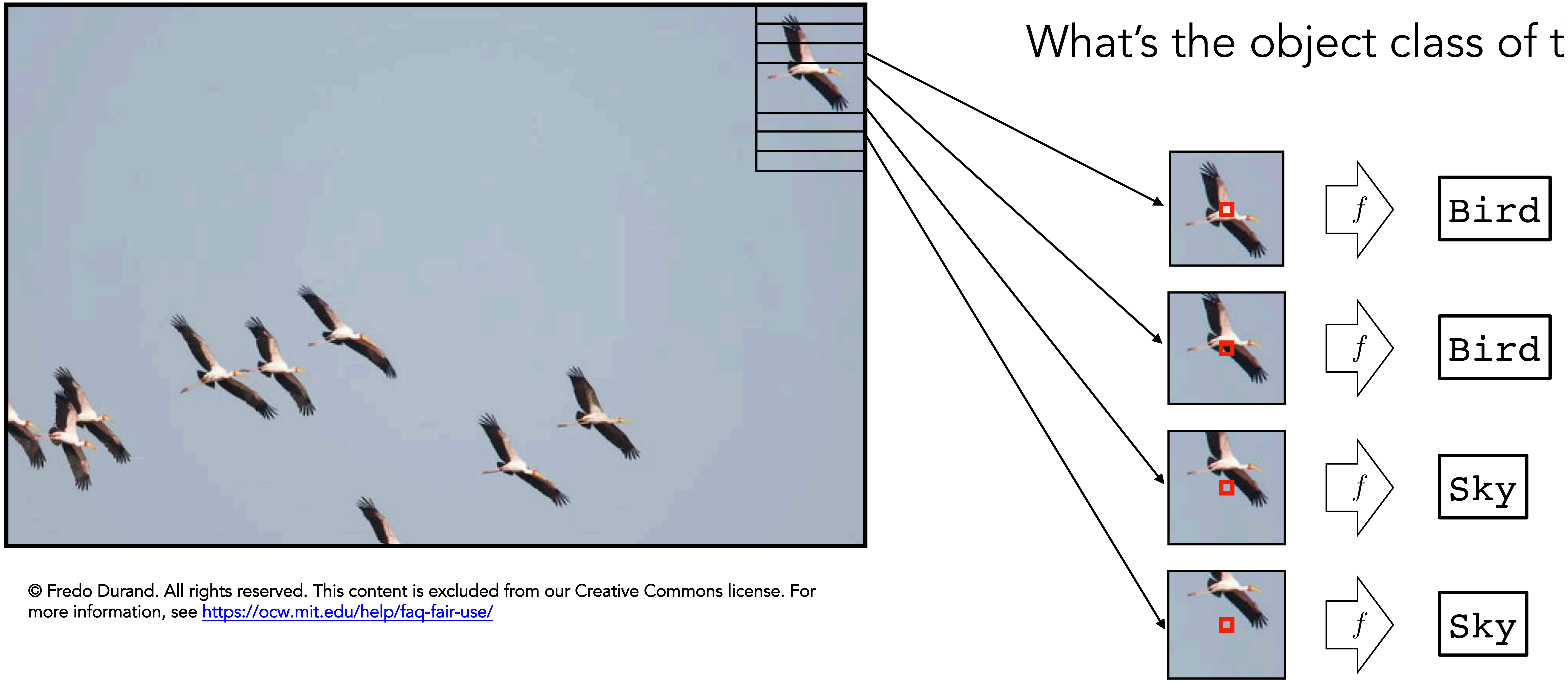
How to increase the resolution of the output map?



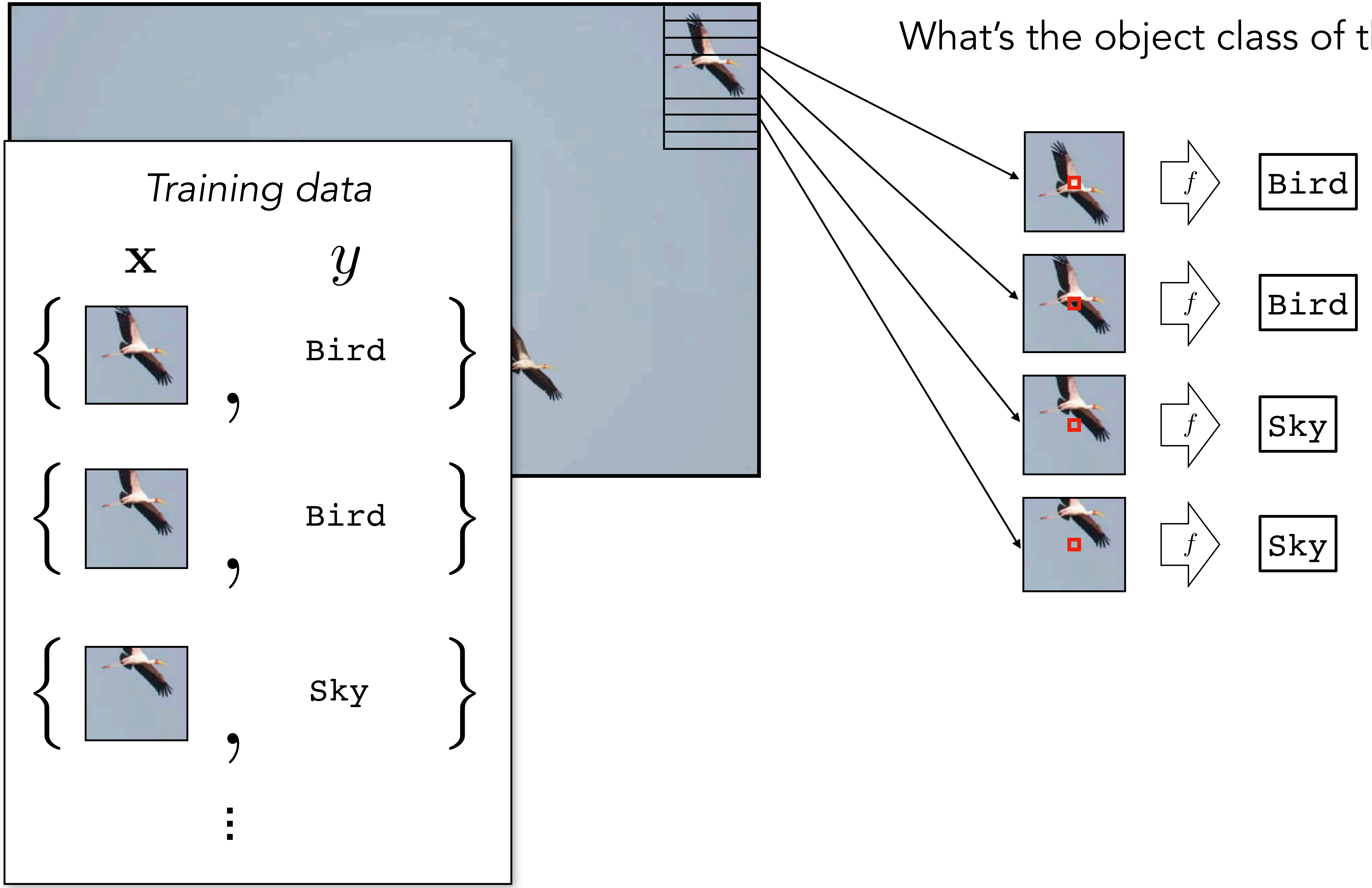
Smaller patches increase resolution, but not easy to recognize content in small each patch

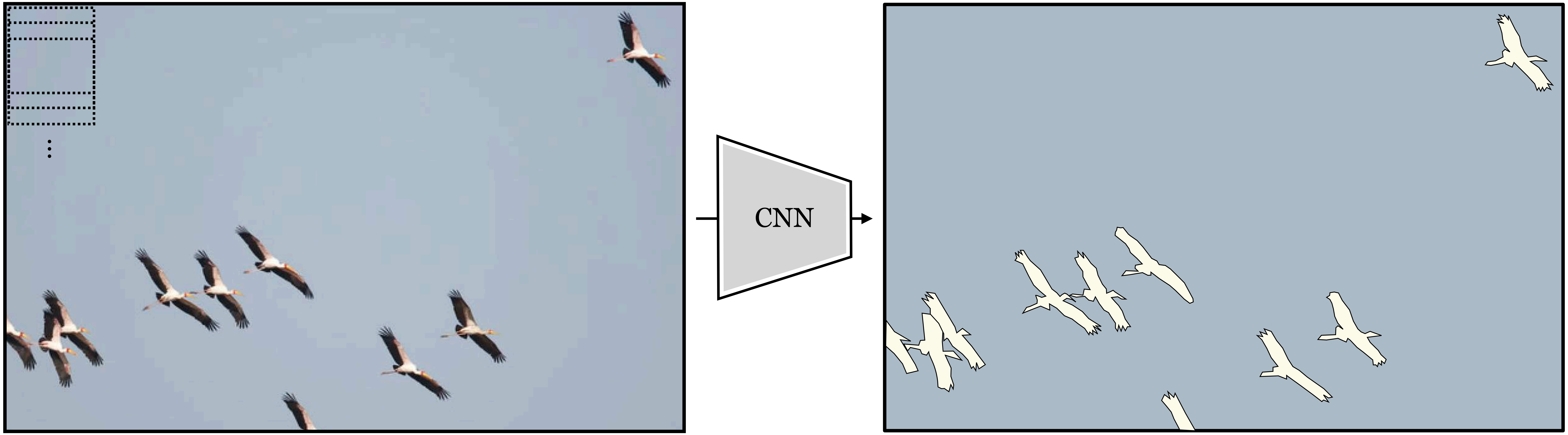
Instead: we will use large but *overlapping* patches

What's the object class of the center pixel?



What’s the object class of the center pixel?

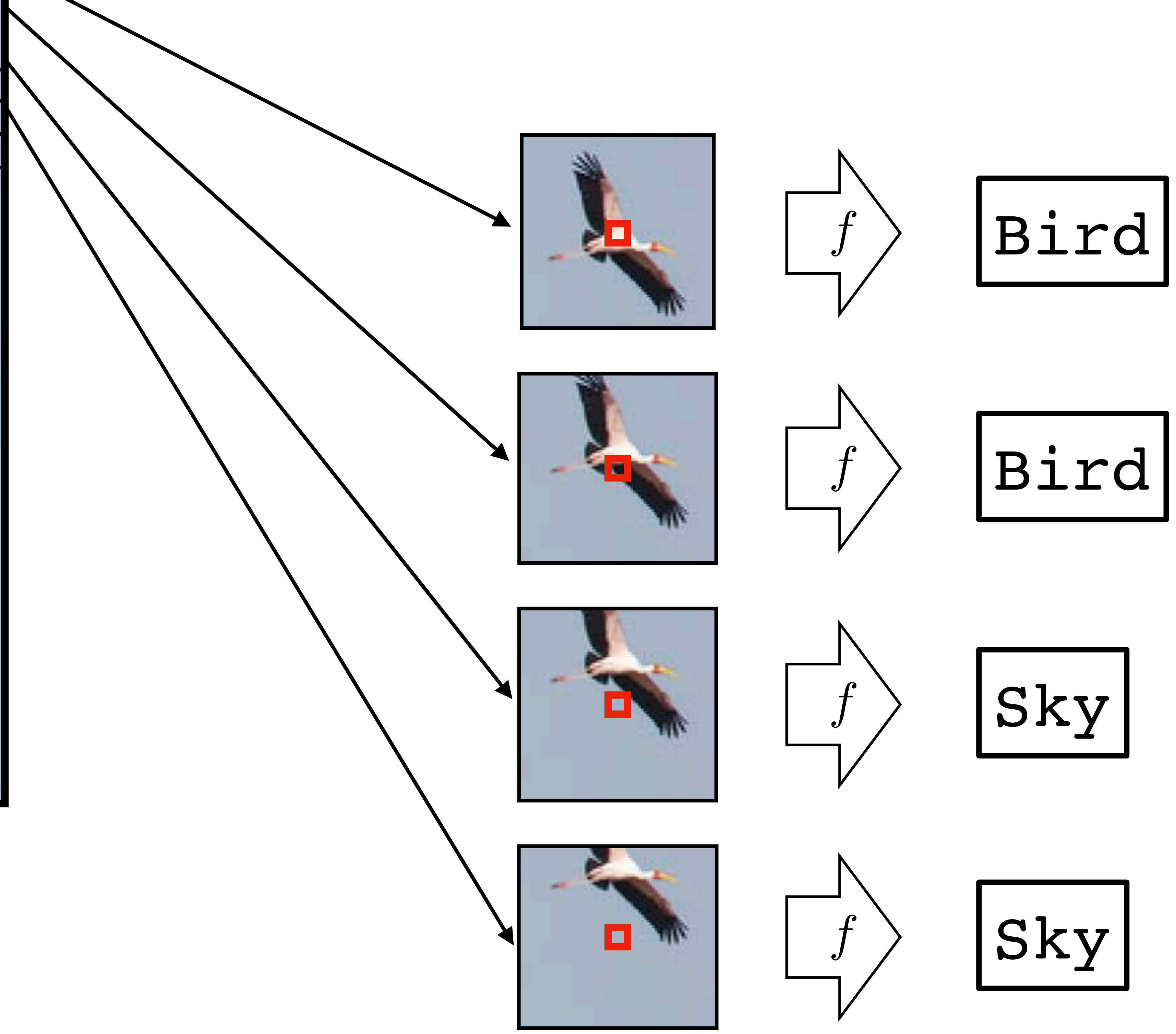


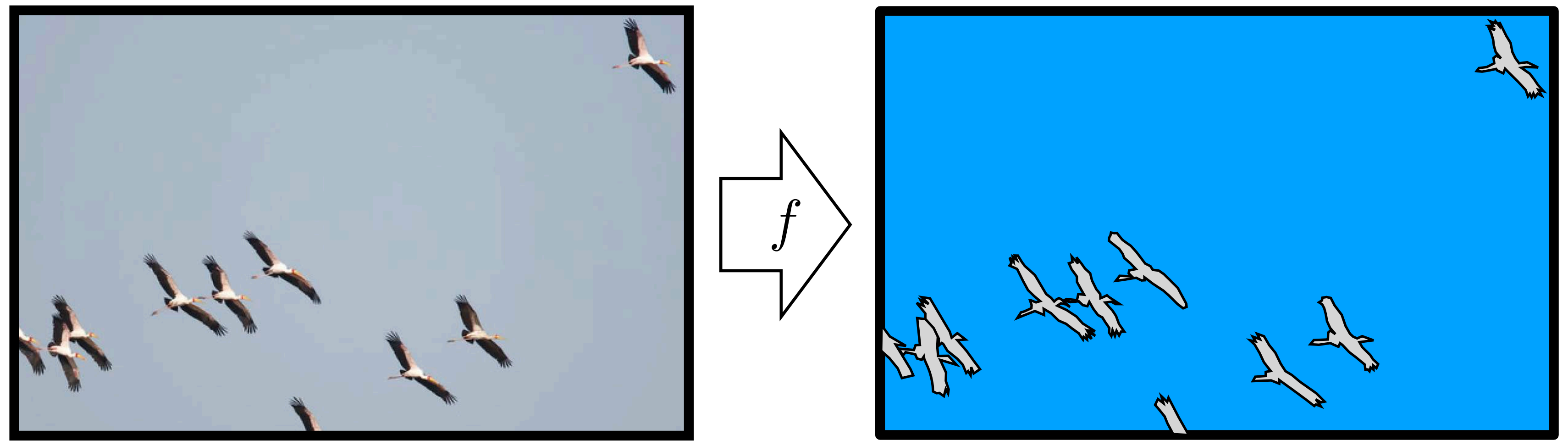


(Colors represent one-hot codes)

This problem is called **semantic segmentation**

What's the object class of the center pixel?



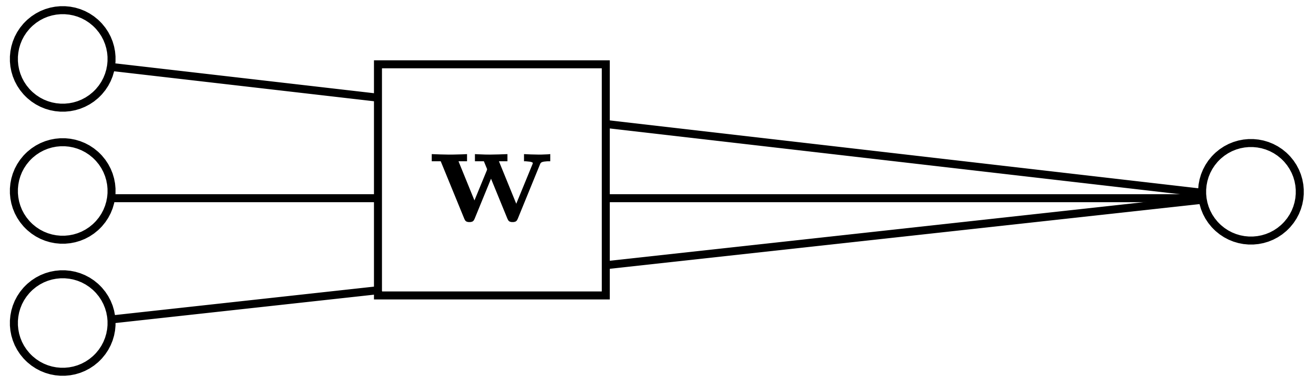


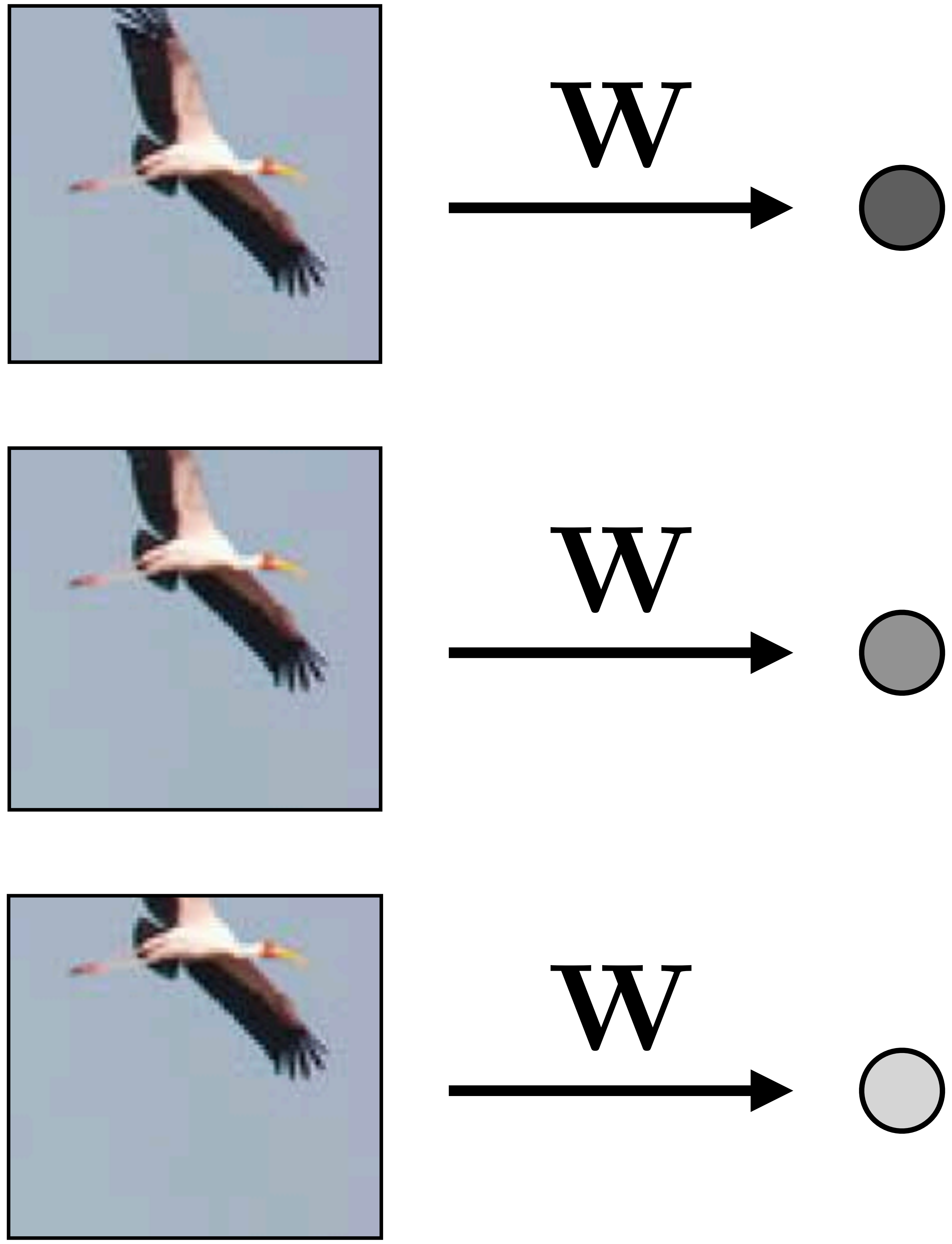
Translation invariance: process each patch in the same way.

An *equivariant* mapping:

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

W computes a weighted sum of all pixels in the patch

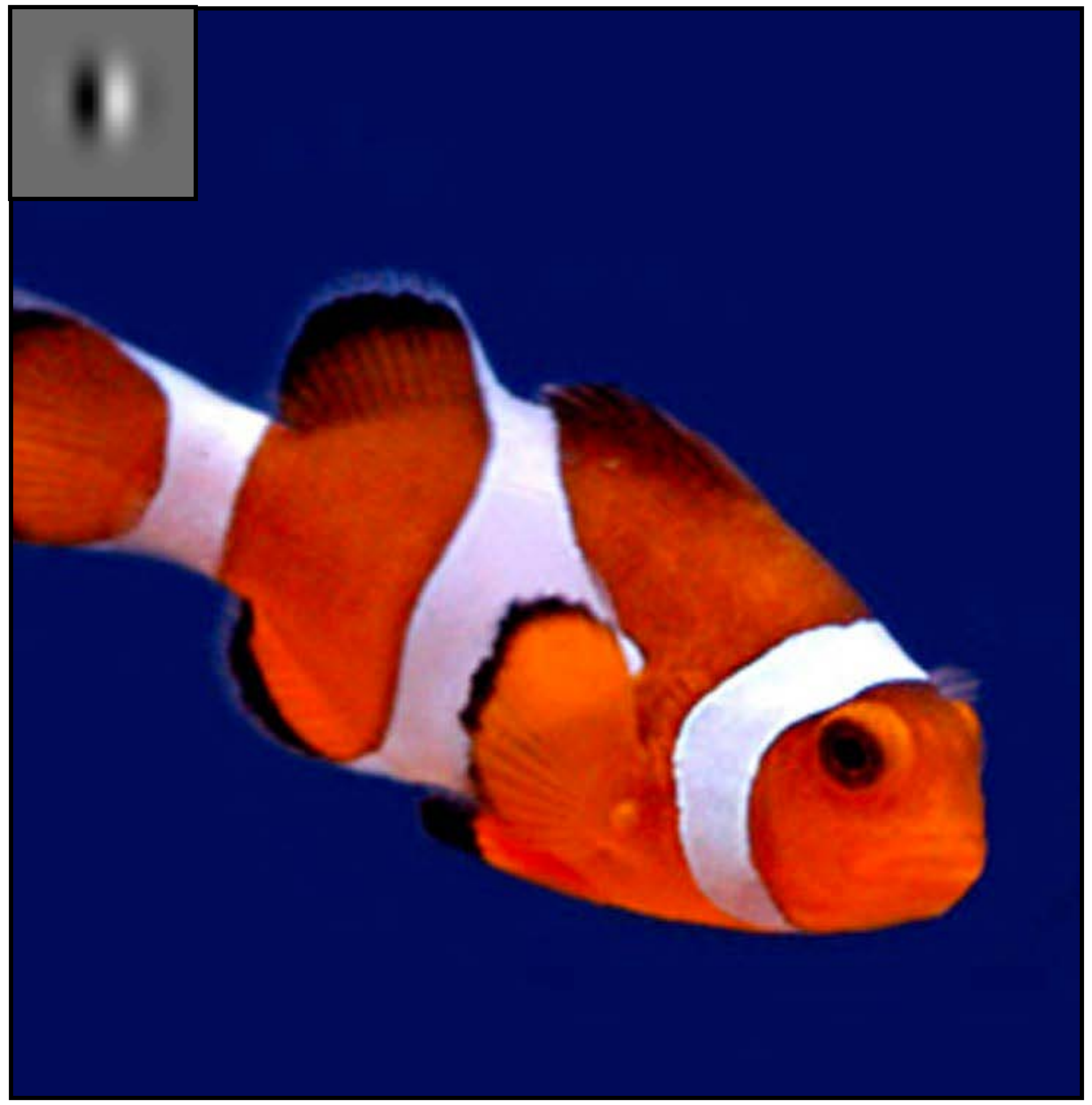


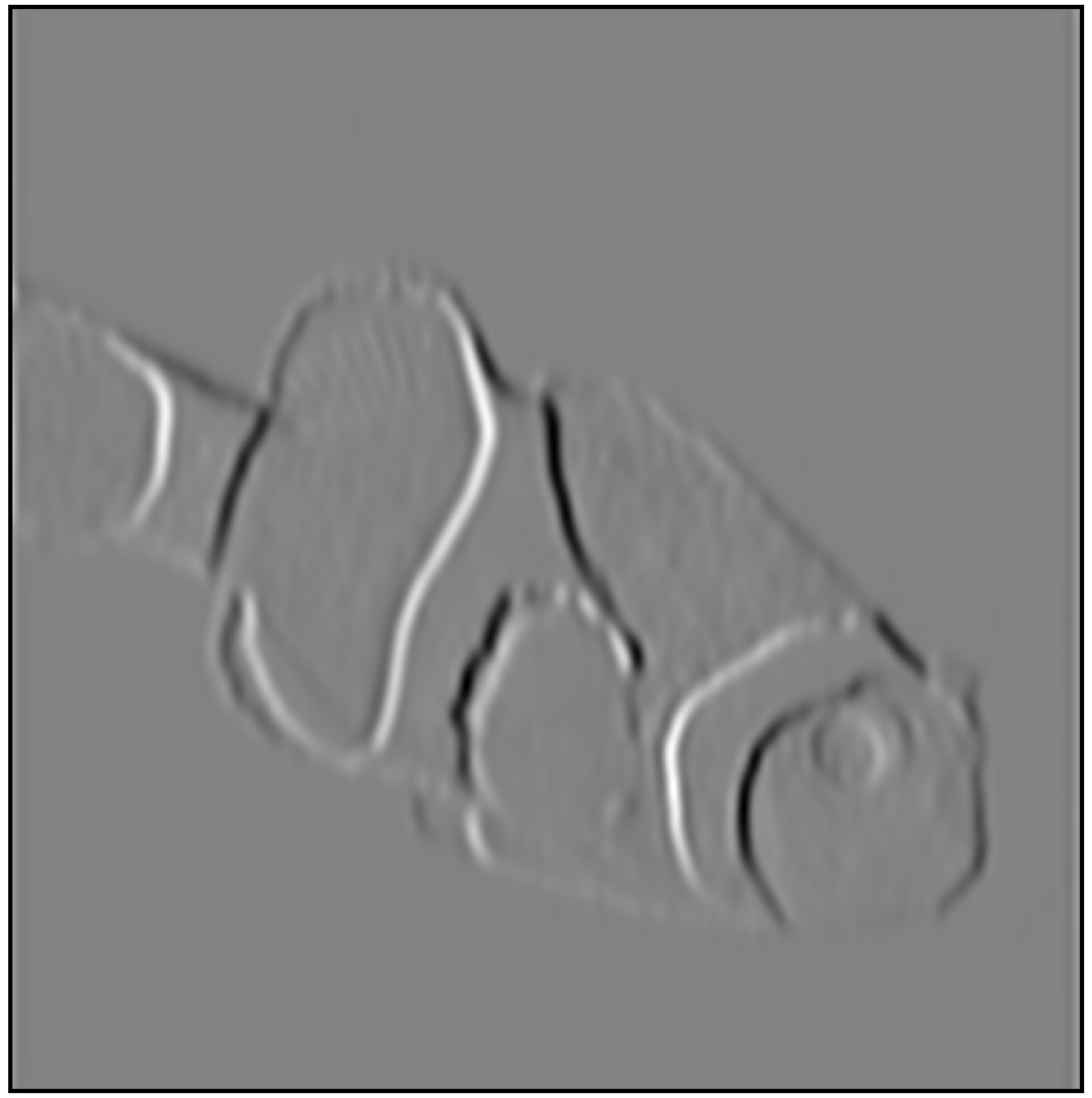


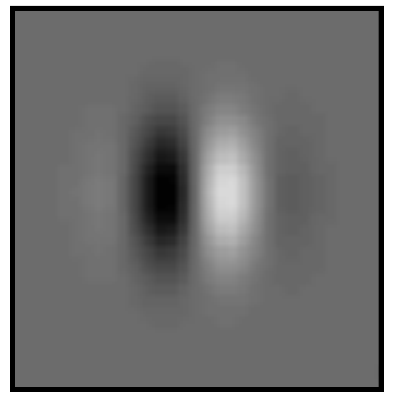
W is a **convolutional kernel** applied to the full image!

Convolution

Linear, shift-invariant transformation



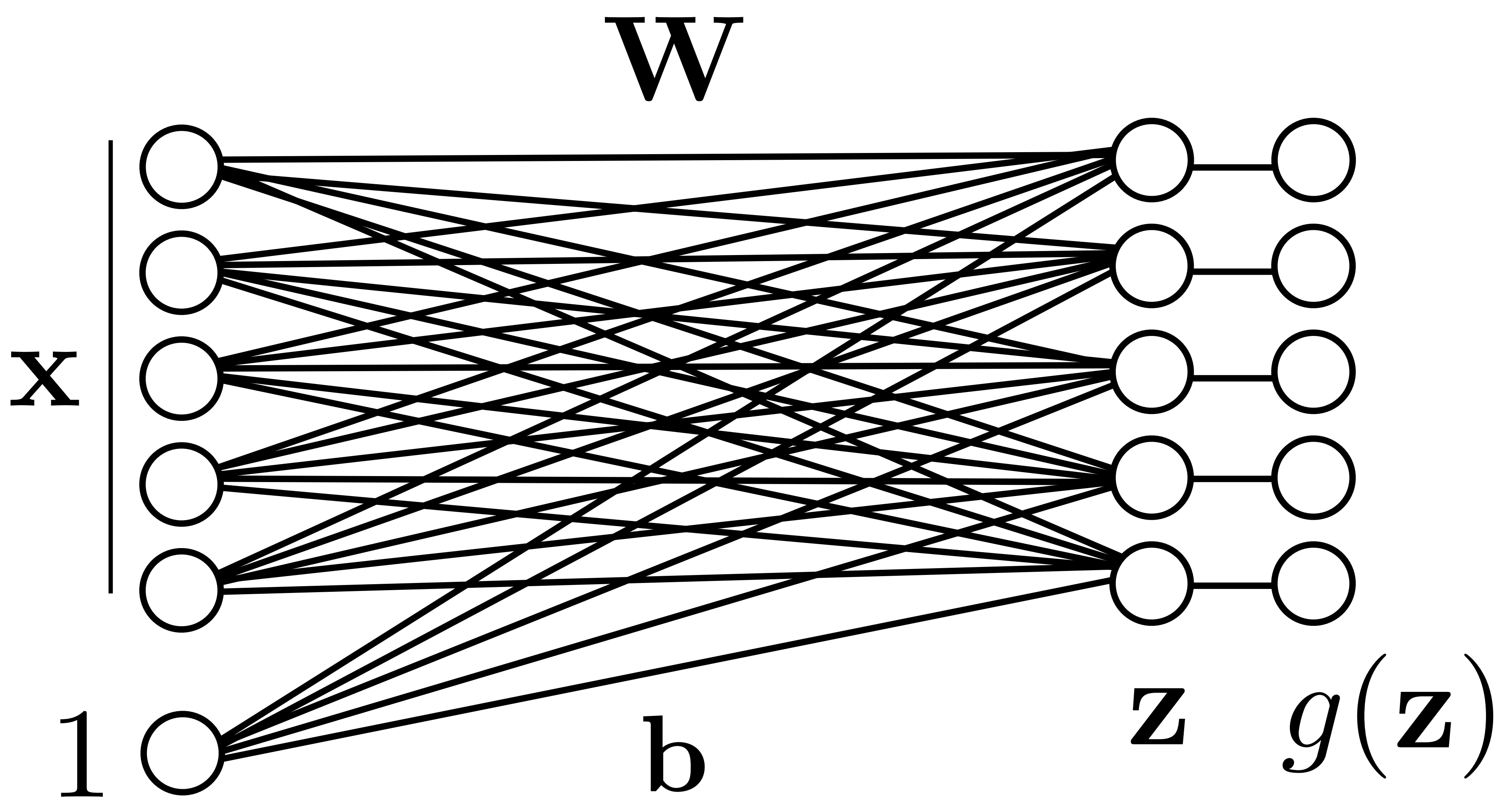


filter 

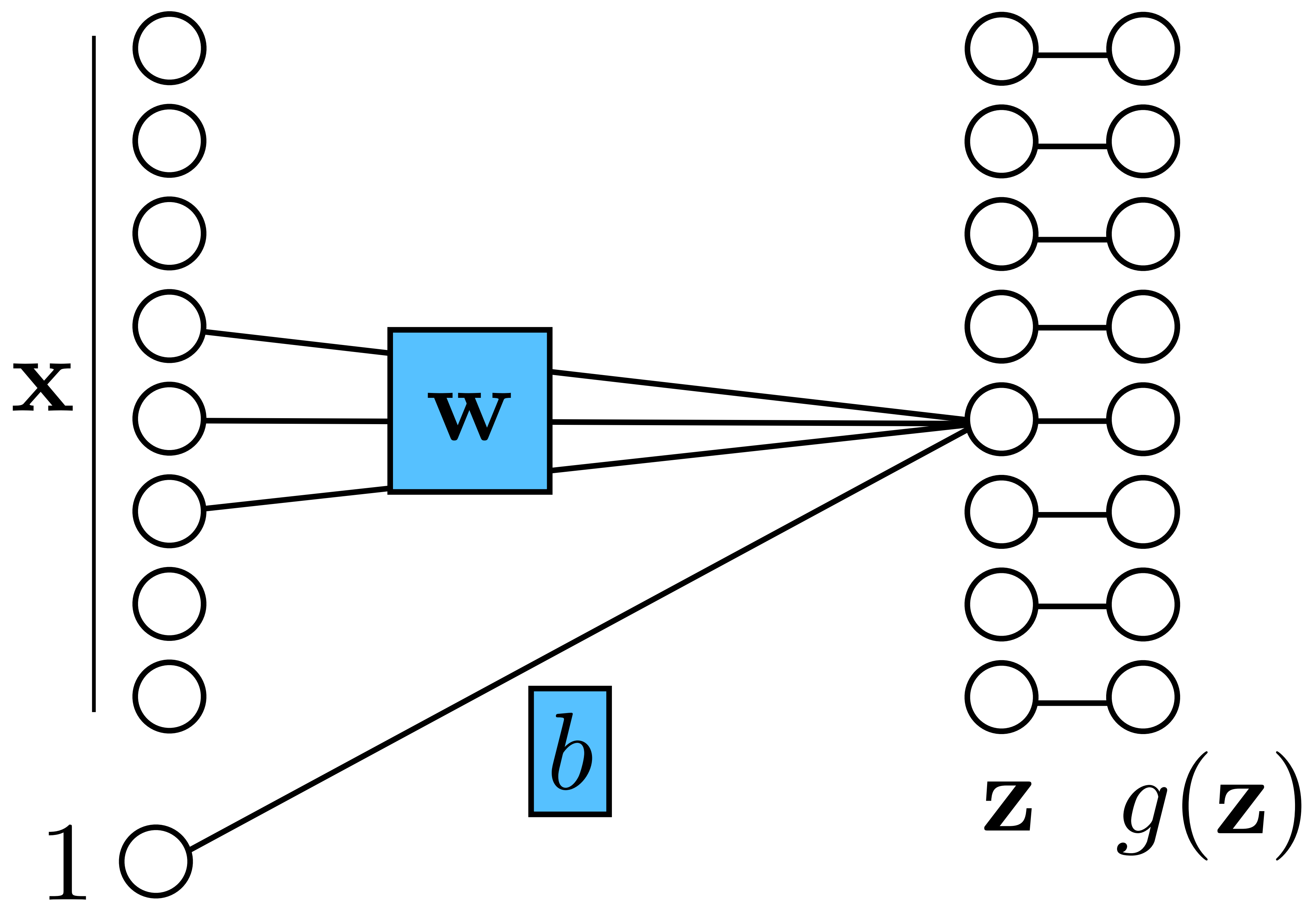
$$x_{\mathtt{out}}[n,m] = b + \sum_{k_1,k_2=-K}^{K} w[k_1,k_2]\,x_{\mathtt{in}}[n+k_1, m+k_2]$$

Fully-connected network

Fully-connected (fc) layer



Locally connected network

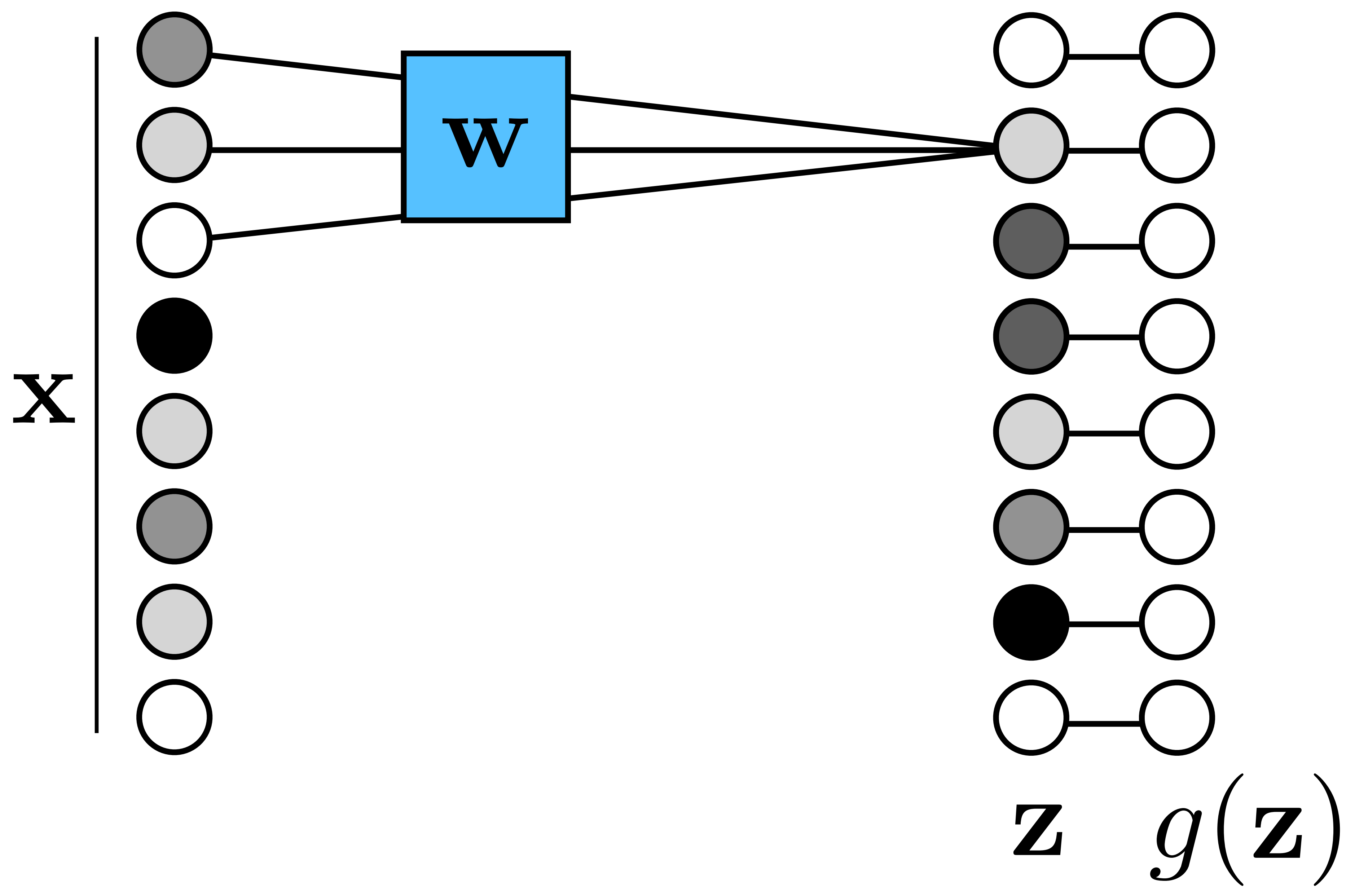


Often, we assume output is a **local** function of input.

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

Convolutional neural network

Conv layer



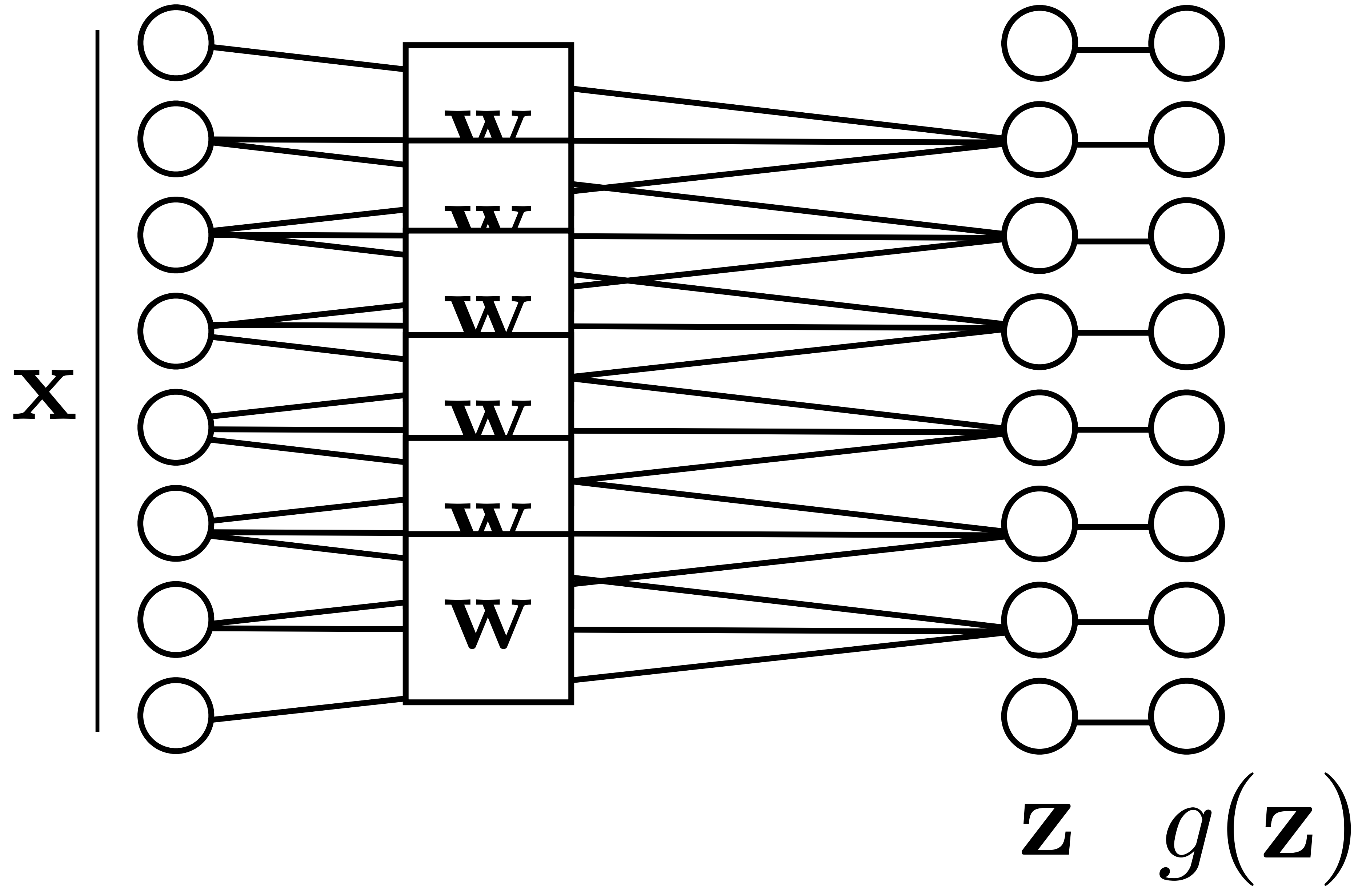
$$\mathbf{z} = \mathbf{w} \star \mathbf{x} + b$$

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



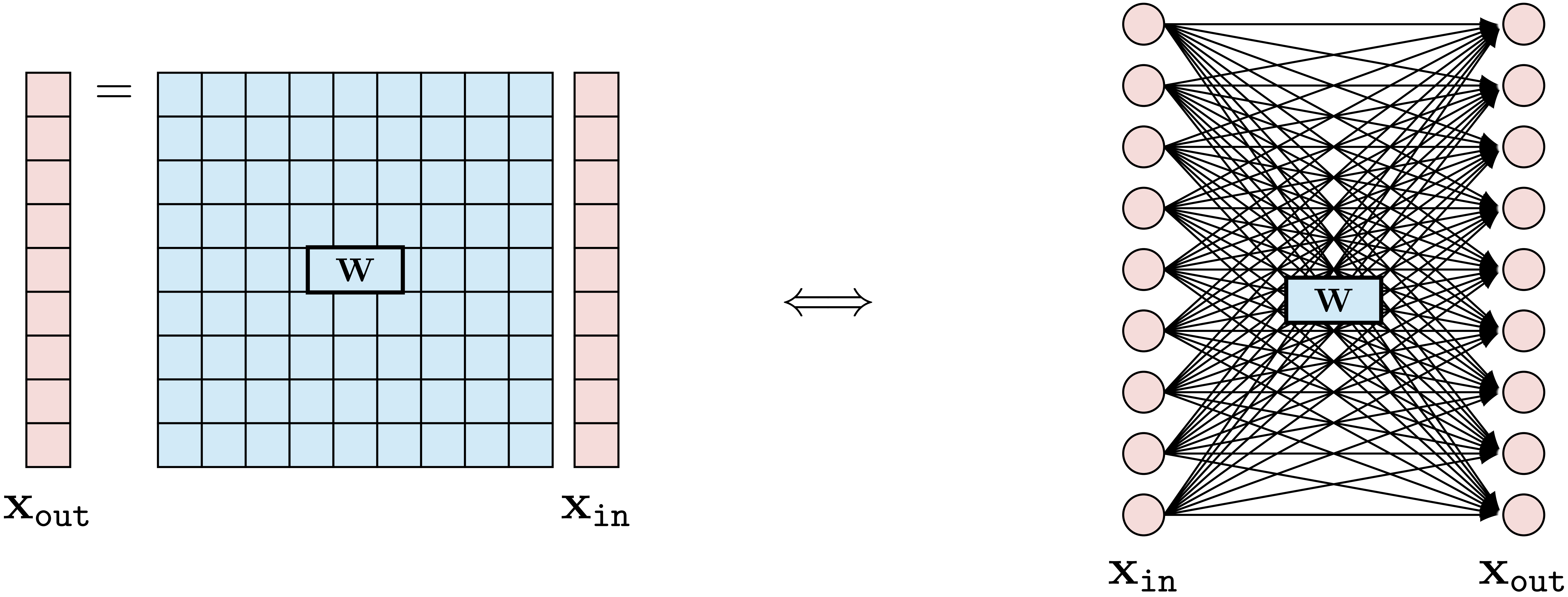
$$\mathbf{z} = \mathbf{w} \star \mathbf{x} + b$$

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.

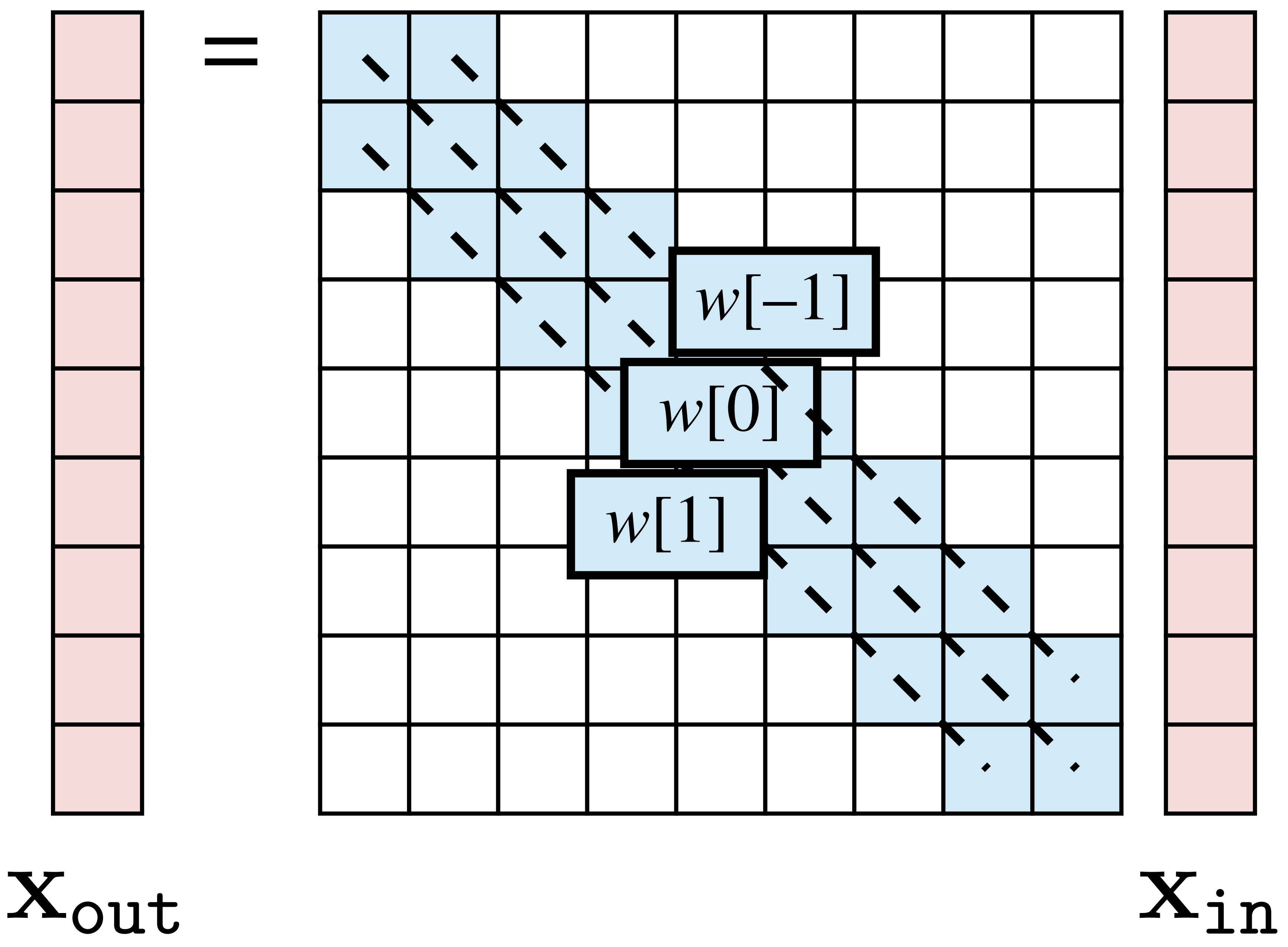
(Fully-connected) linear layer

$$\mathbf{x}_{\mathtt{out}} = \mathbf{W}\mathbf{x}_{\mathtt{in}} + \mathbf{b}$$

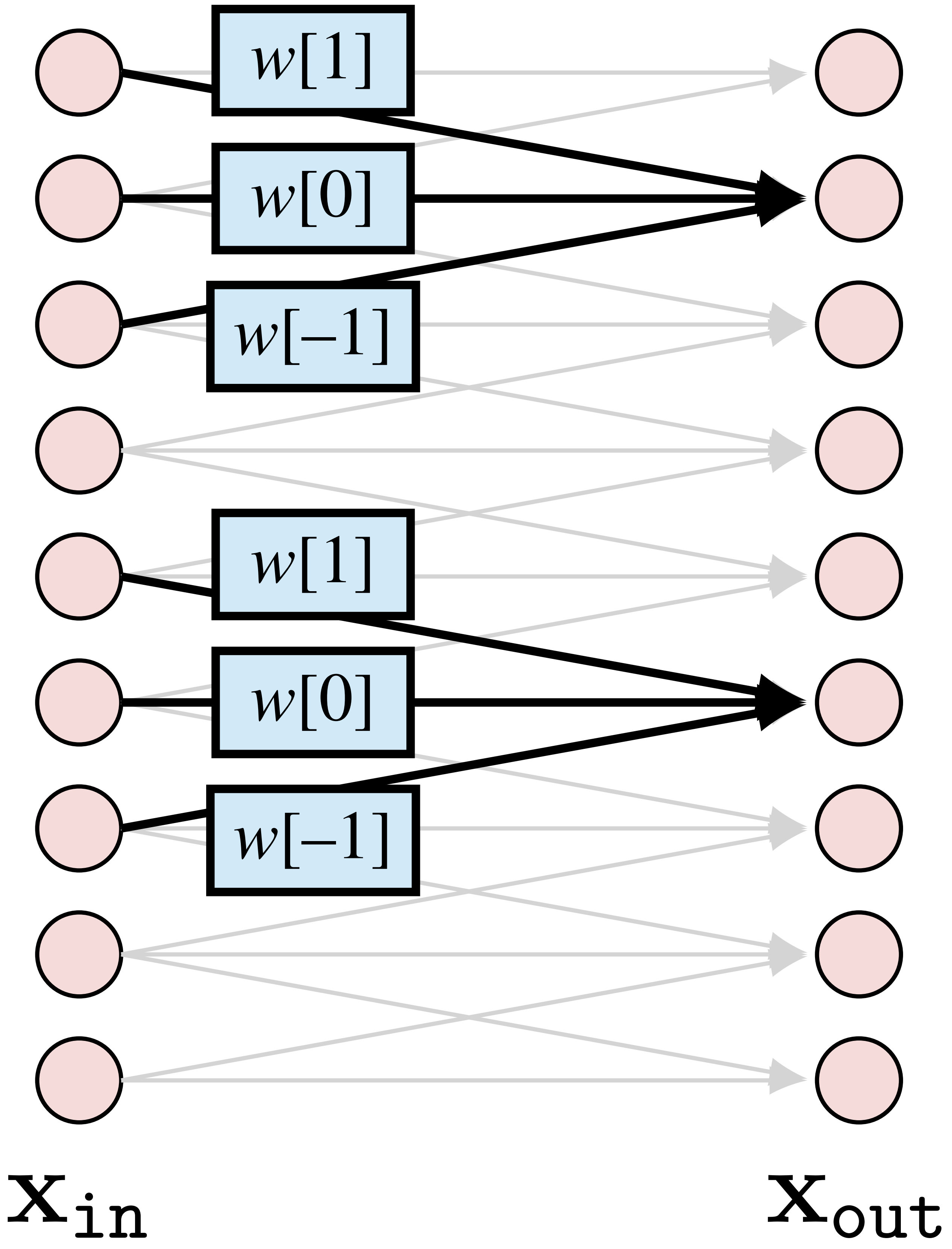


Convolutional layer

$$\mathbf{x}_{\texttt{out}} = \mathbf{w} \star \mathbf{x}_{\texttt{in}} + b$$

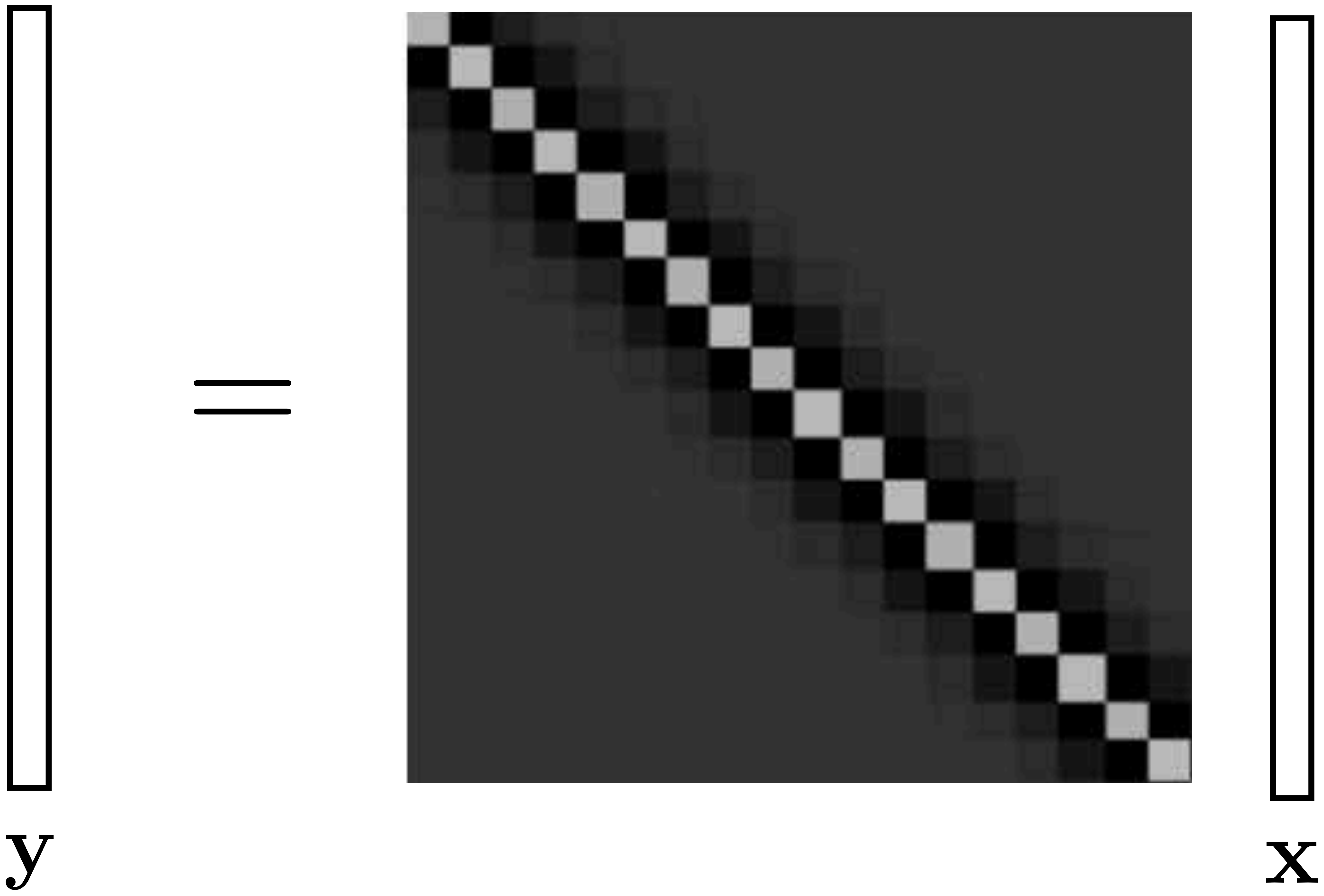


$\iff$



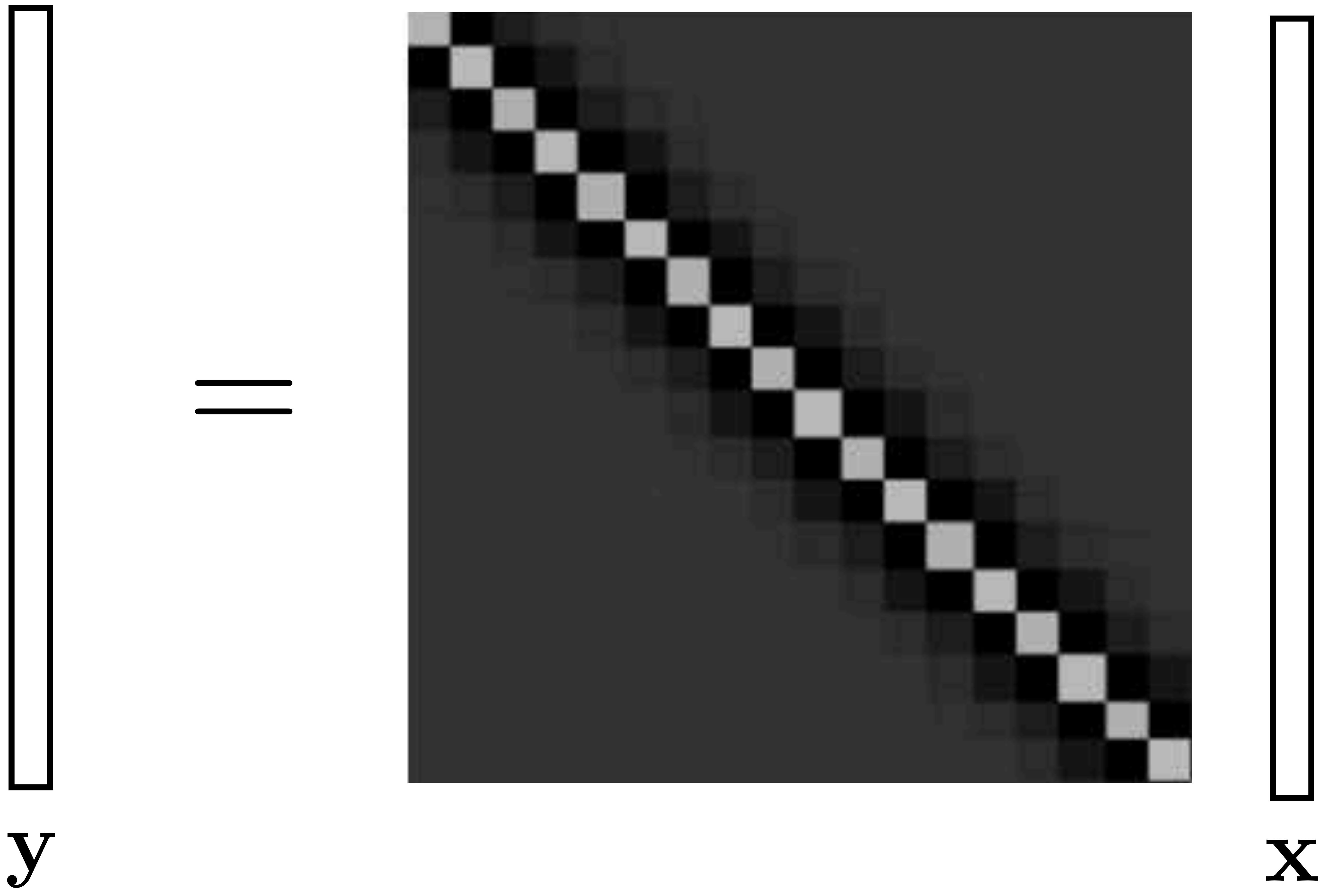
Toeplitz matrix

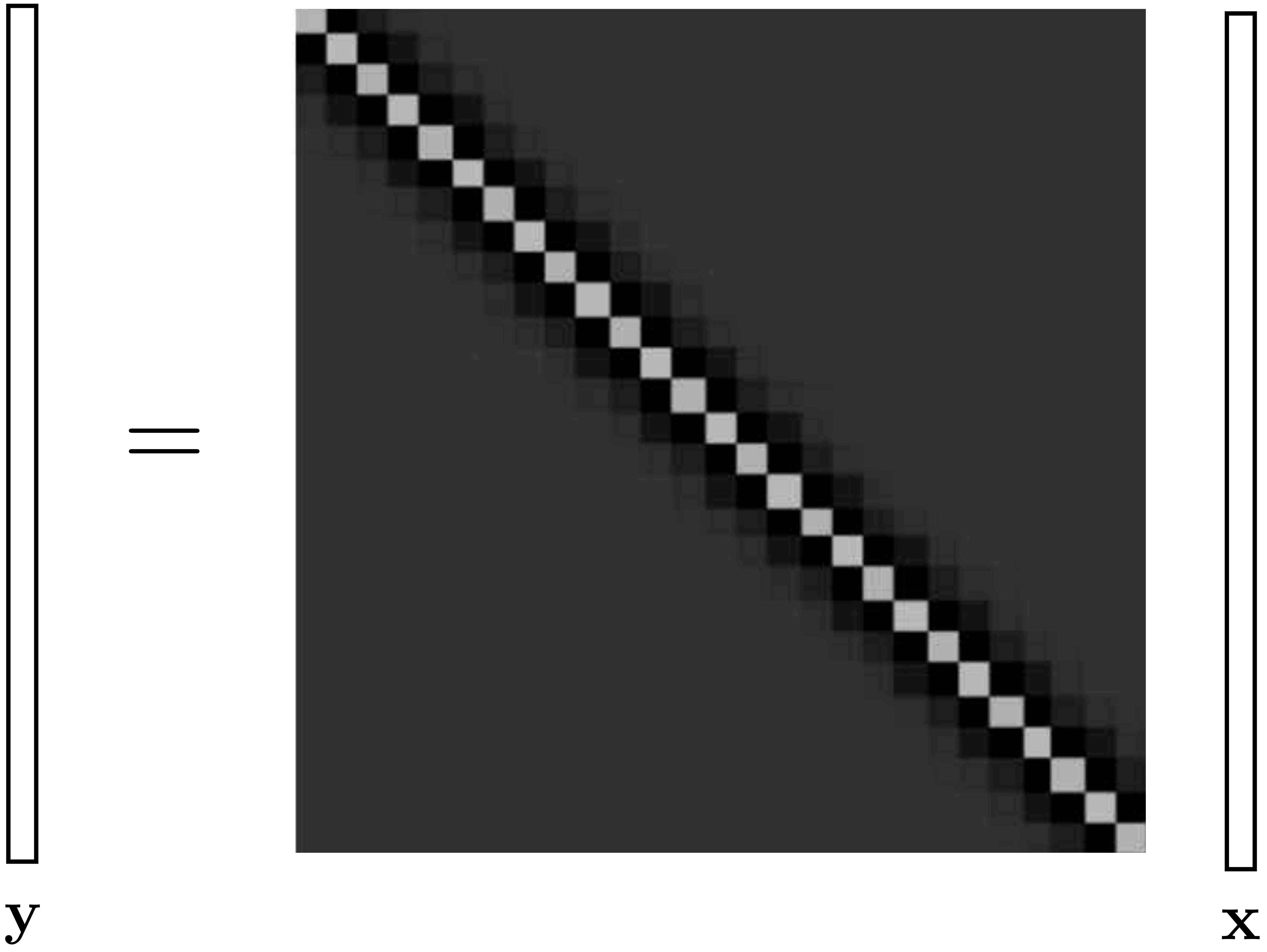
$$\begin{pmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{pmatrix}$$



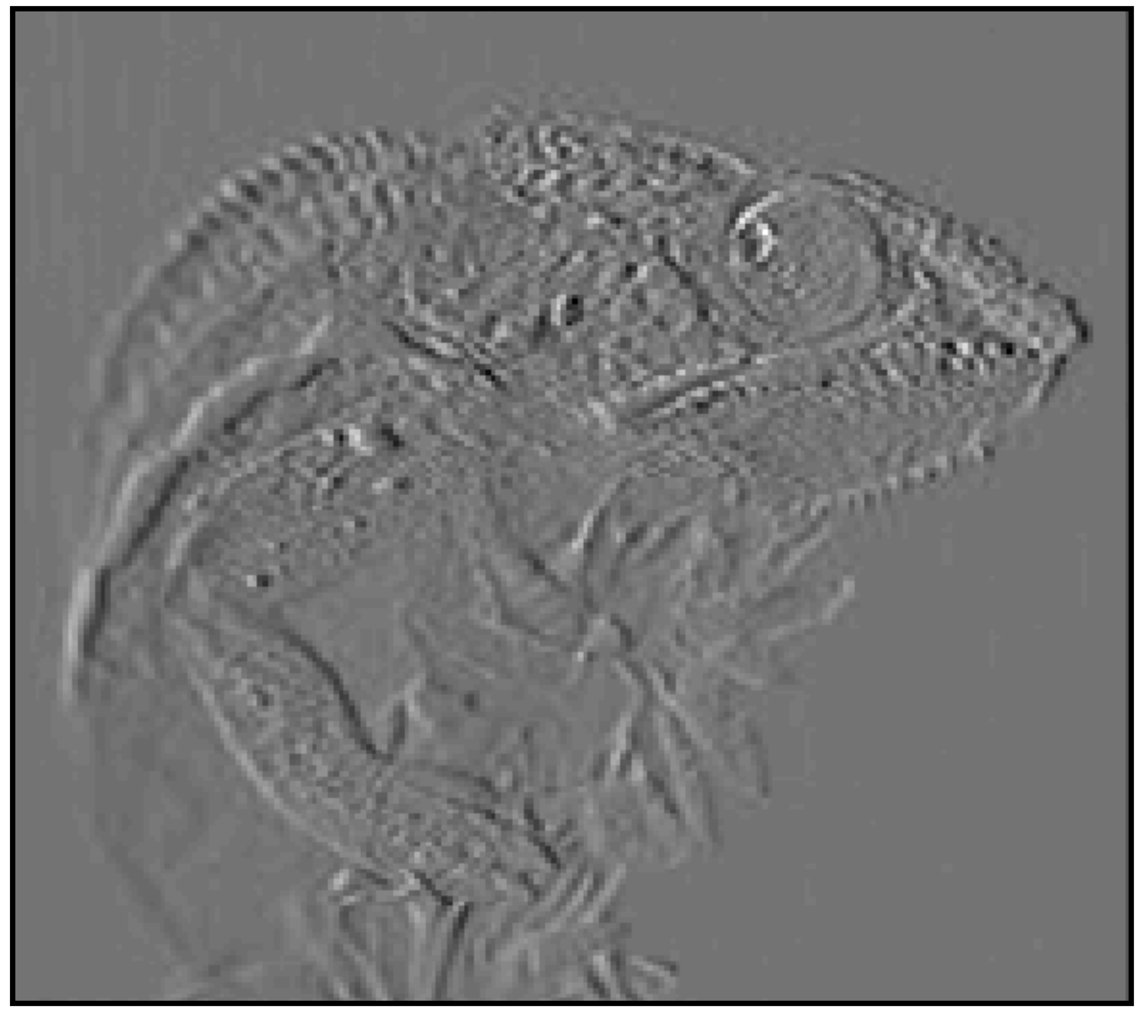
e.g., pixel image

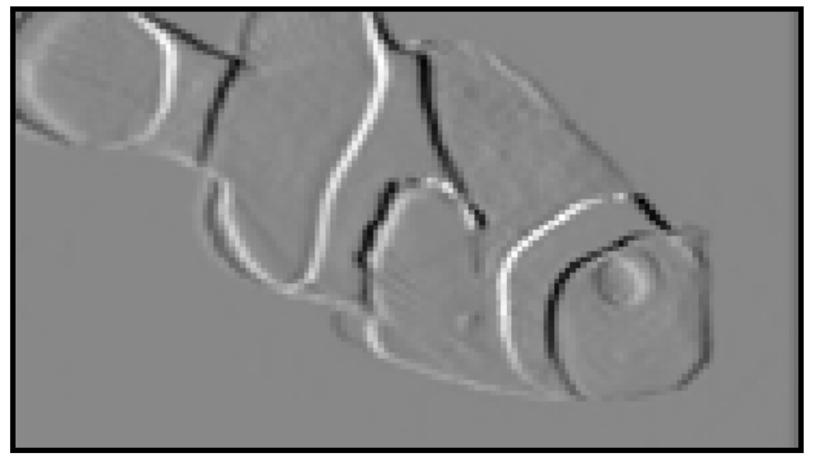
- Constrained linear layer
- Fewer parameters —> easier to learn, less overfitting


y
x



Conv layers can be applied to arbitrarily-sized inputs
(generalizes beyond the training data due to an architectural structure!)

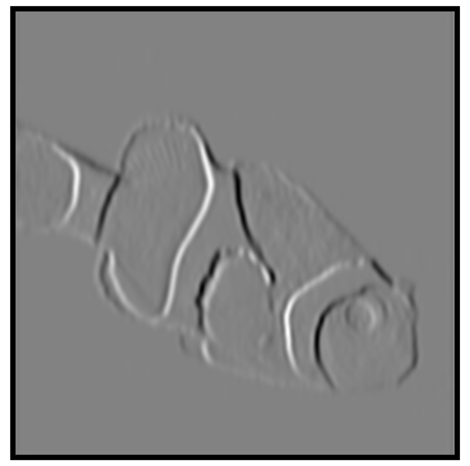


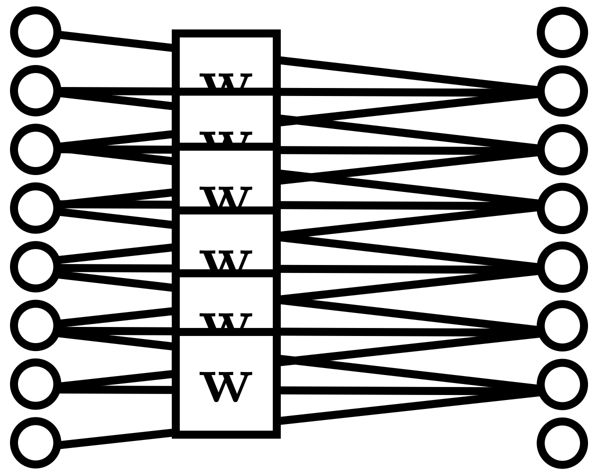


Five views on convolutional layers

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

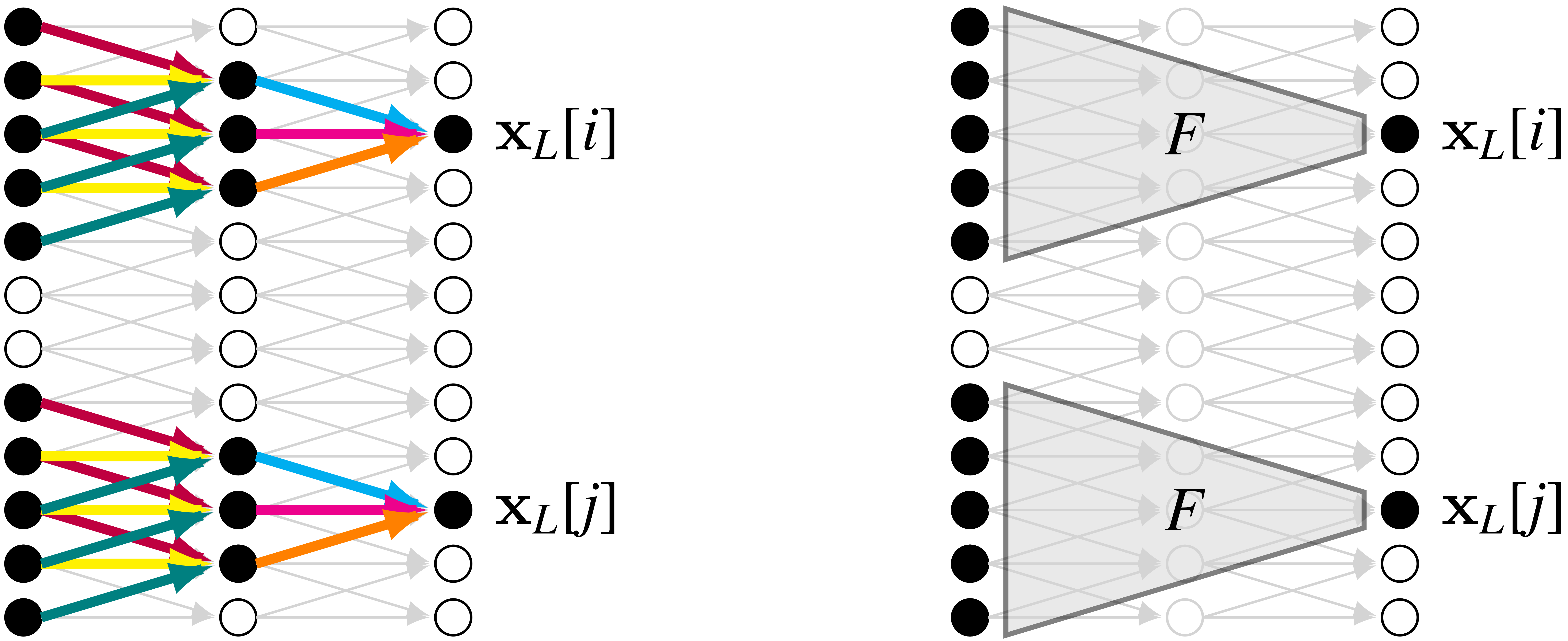
2. Patch processing

3. Image filter 

4. Parameter sharing 

5. A way to process variable-sized tensors

What happens when you stack convolutional layers?

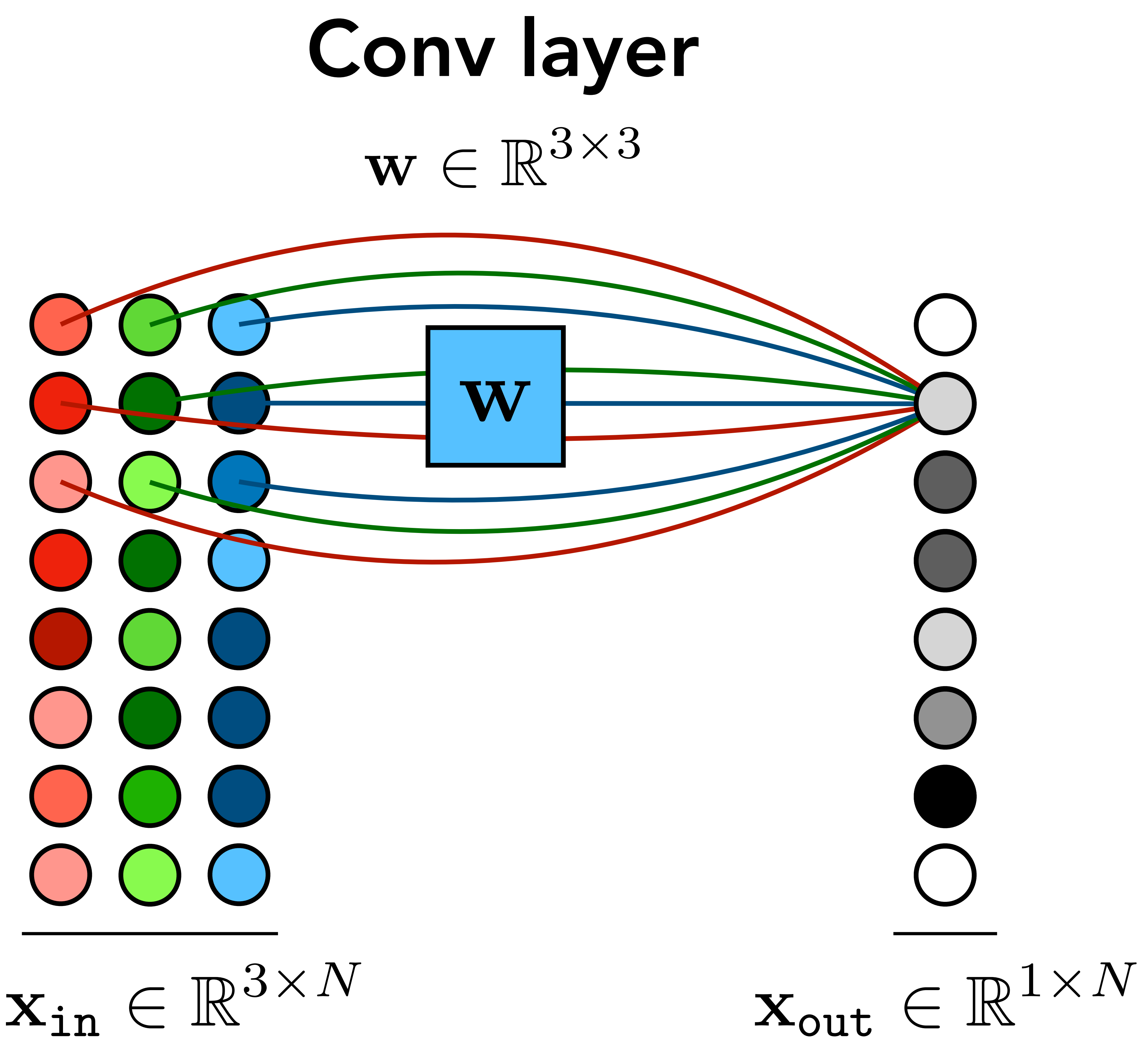


The whole CNN acts like a (nonlinear) convolutional filter!

What if we have color?

(aka multiple input channels?)

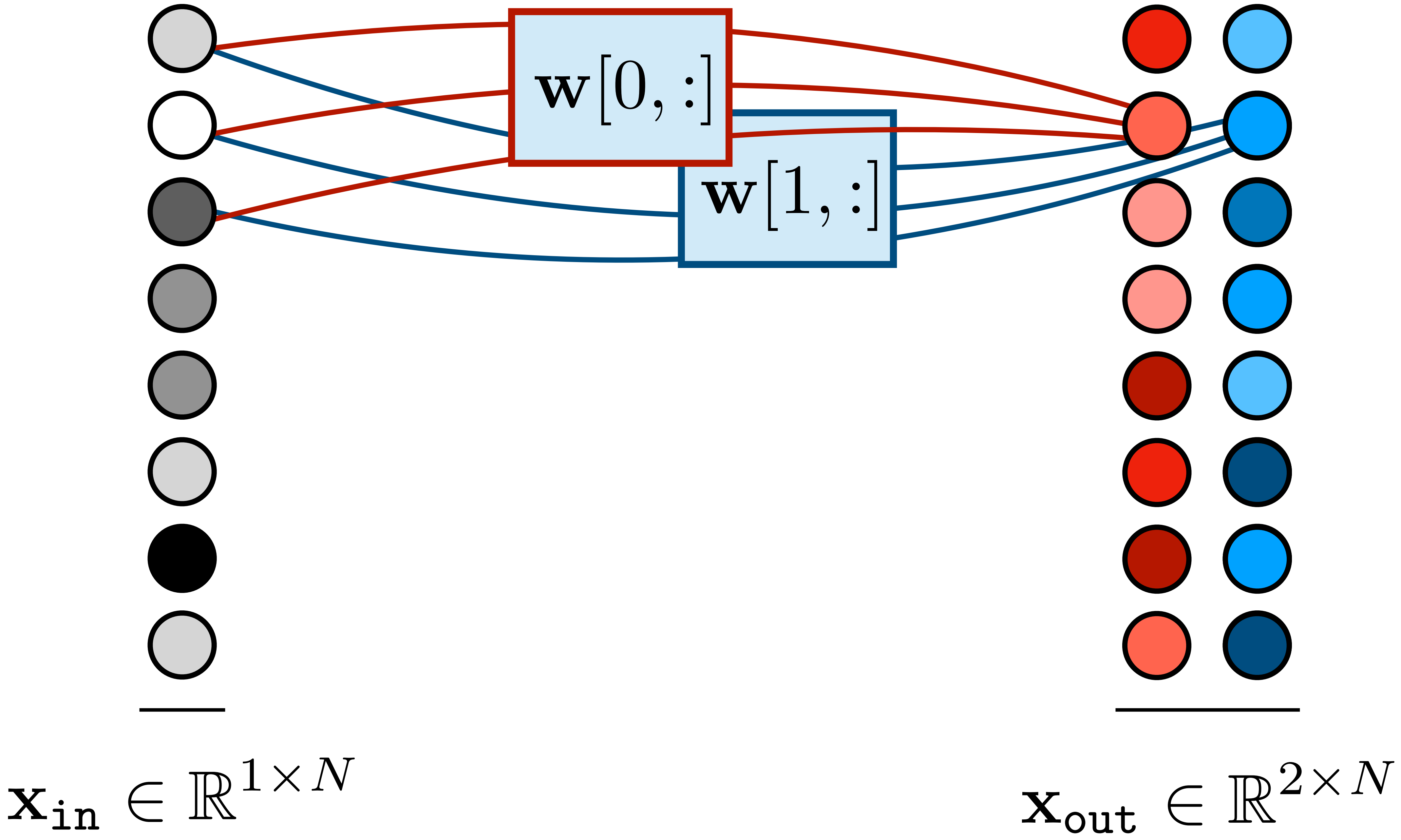
Multichannel inputs



$$\mathbf{x}_{\texttt{out}} = \sum_c \mathbf{w}[c,:] \star \mathbf{x}_{\texttt{in}}[c,:] + b[c]$$

Multichannel *outputs*

Conv layer



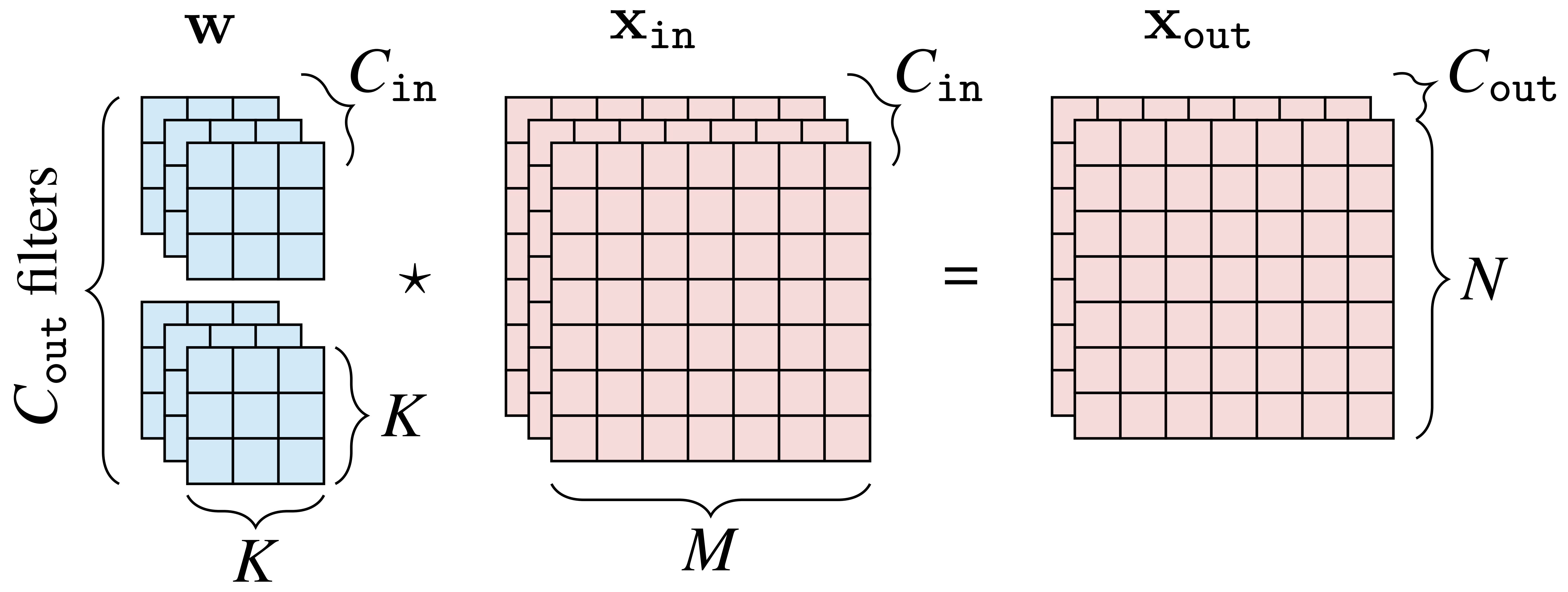
Filter bank of C filters

$$\mathbf{x}_{\mathtt{out}}[0,:] = \mathbf{w}[0,:] \star \mathbf{x}_{\mathtt{in}} + b[0]$$

$$\vdots$$

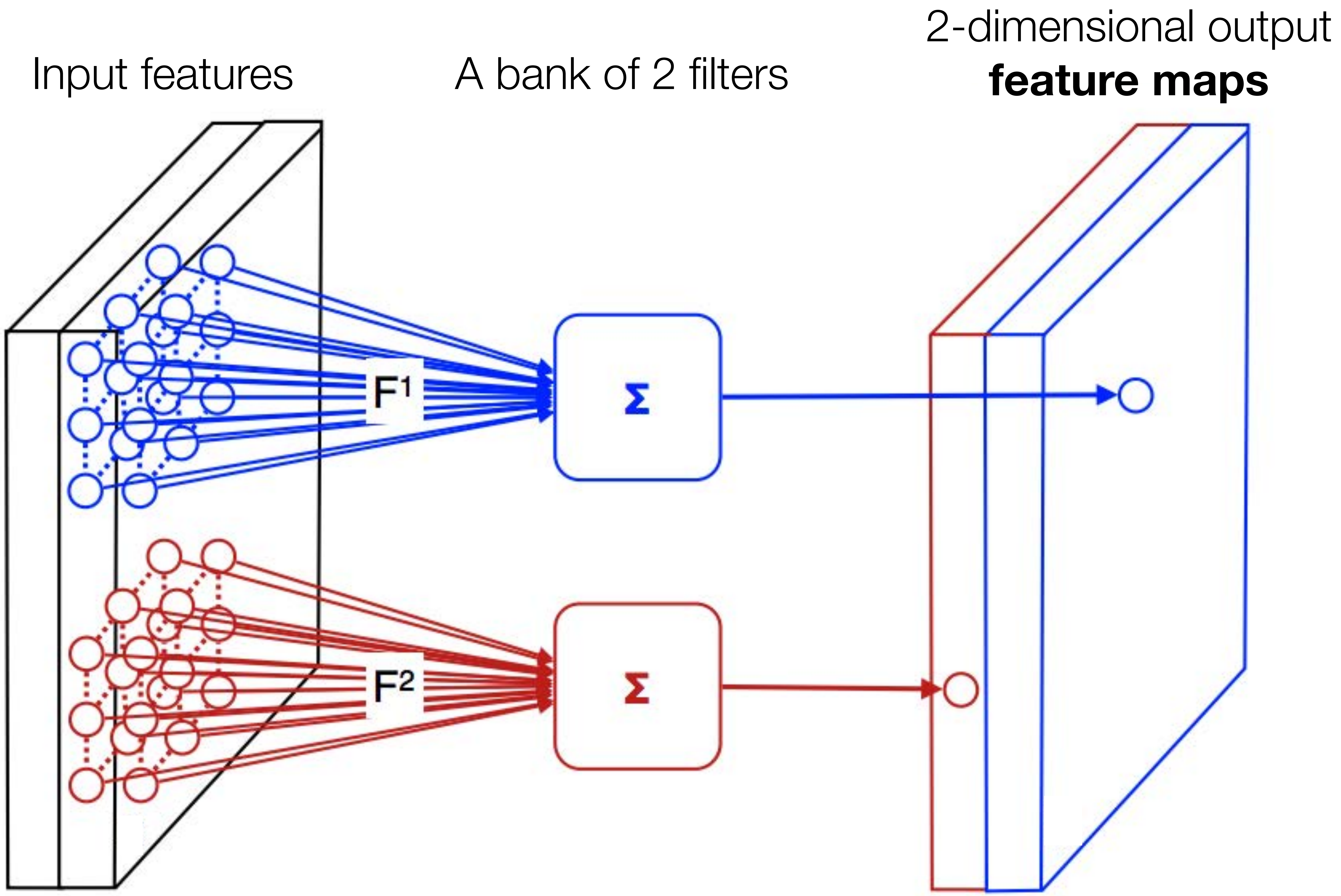
$$\mathbf{x}_{\mathtt{out}}[C,:] = \mathbf{w}[C-1,:] \star \mathbf{x}_{\mathtt{in}} + b[C-1]$$

General Convolutional Layer Form: Multi-Input, Multi-Output



$$\mathbf{x}_{\texttt{out}}[c_2,:,:] = \sum_{c_1=1}^{C_{\texttt{in}}} \mathbf{w}[c_1, c_2, :, :] \star \mathbf{x}_{\texttt{in}}[c_1, :, :] + b[c_2]$$

Input channel (c_1) — Output channel (c_2)



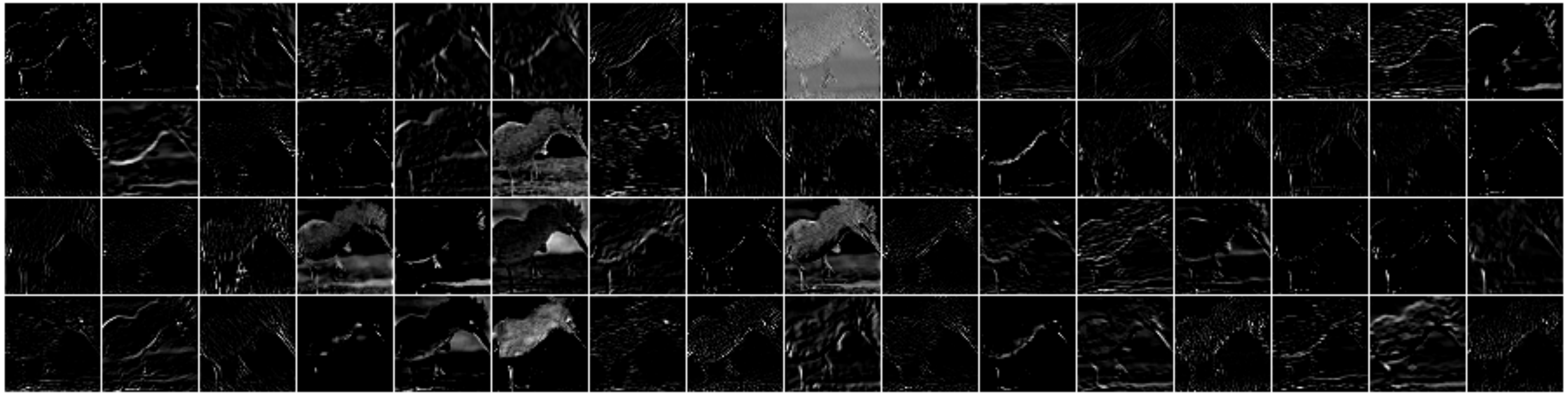
$$\mathbf{x}_{\texttt{in}} \in \mathbb{R}^{C_{\texttt{in}} \times H \times W} \rightarrow \mathbf{x}_{\texttt{out}} \in \mathbb{R}^{C_{\texttt{out}} \times H \times W}$$

[Figure modified from Andrea Vedaldi]

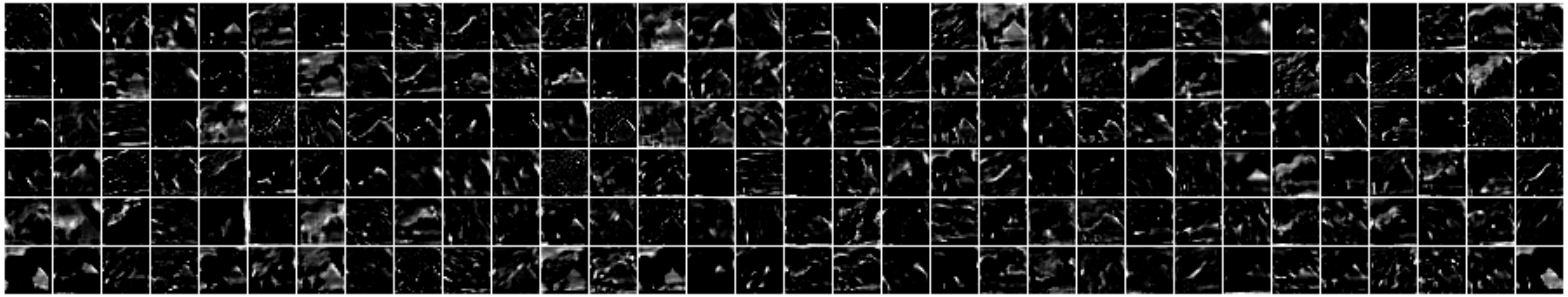
Feature maps

Input

conv1 (after first conv layer)

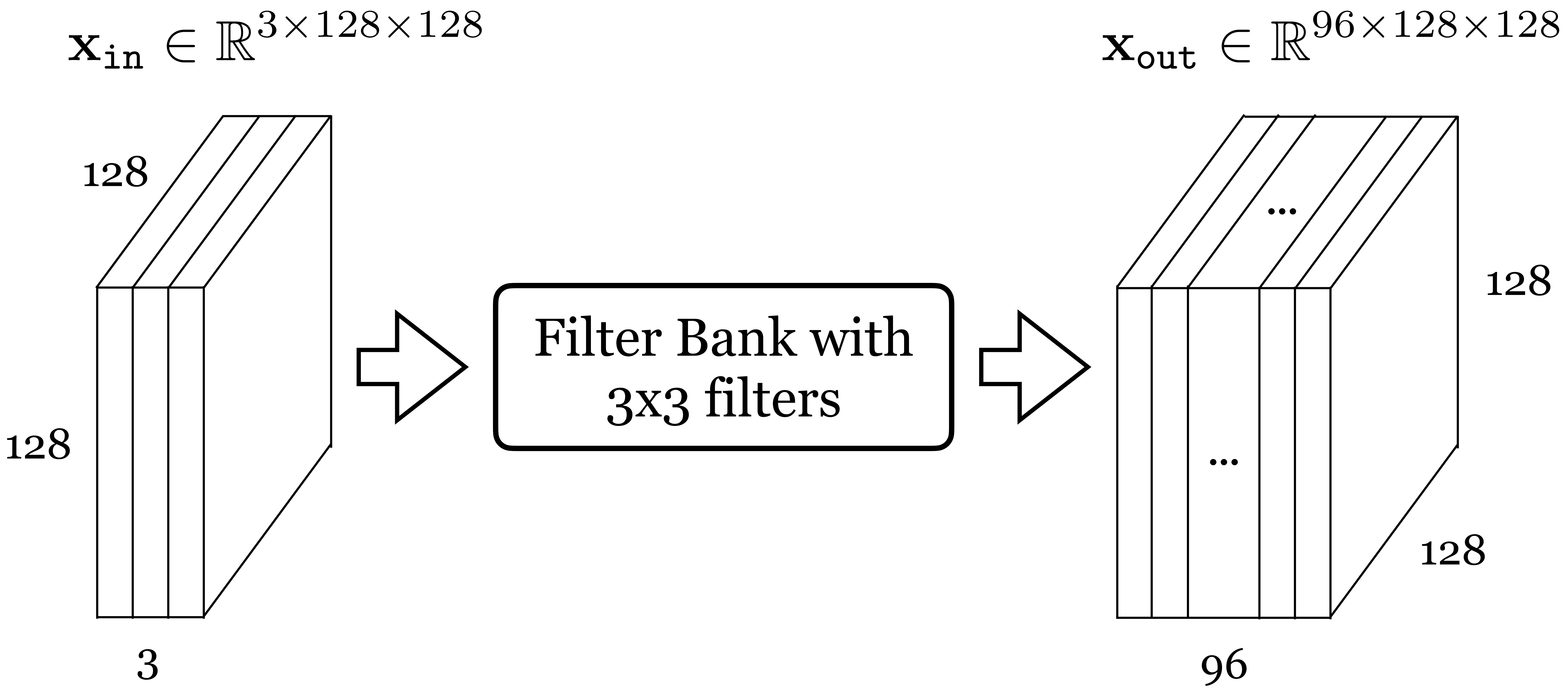


conv2 (after second conv layer)



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

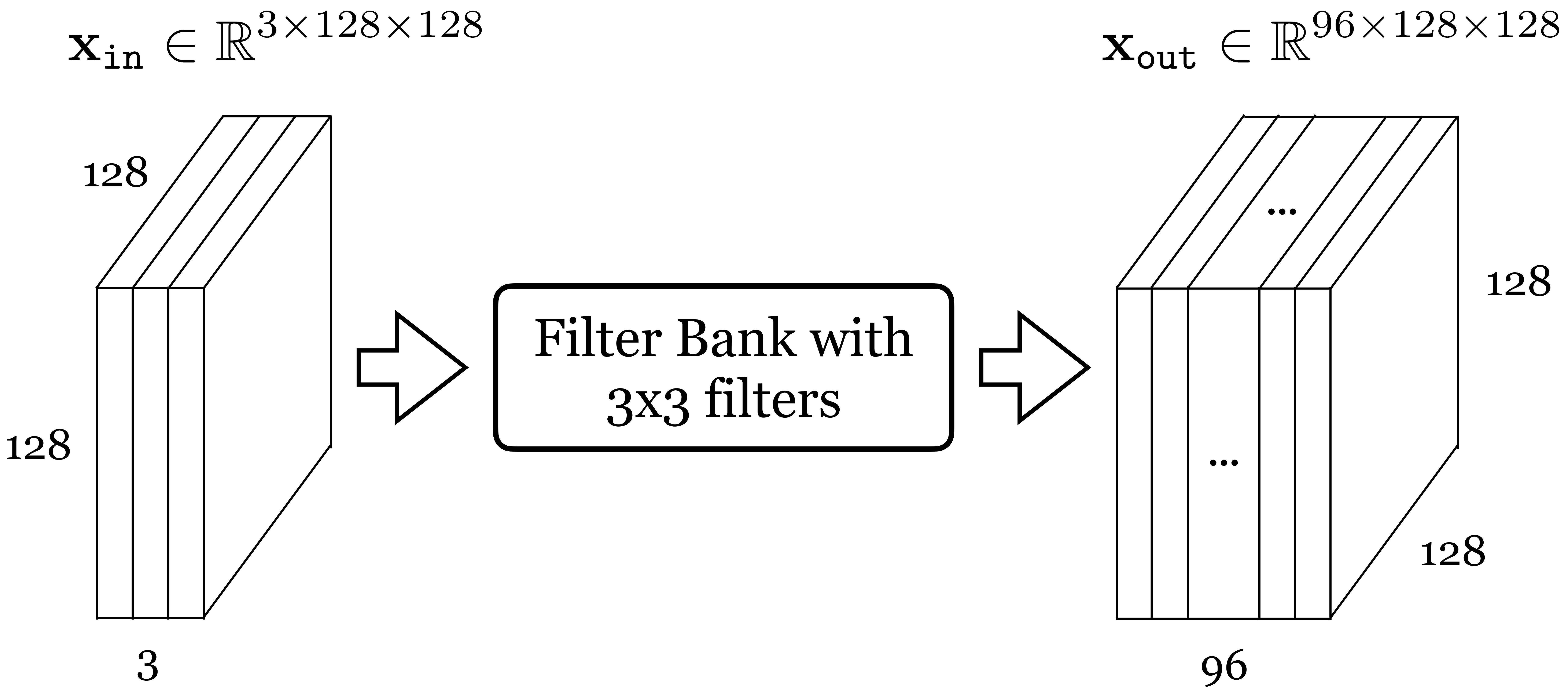
Multiple channels: Example



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

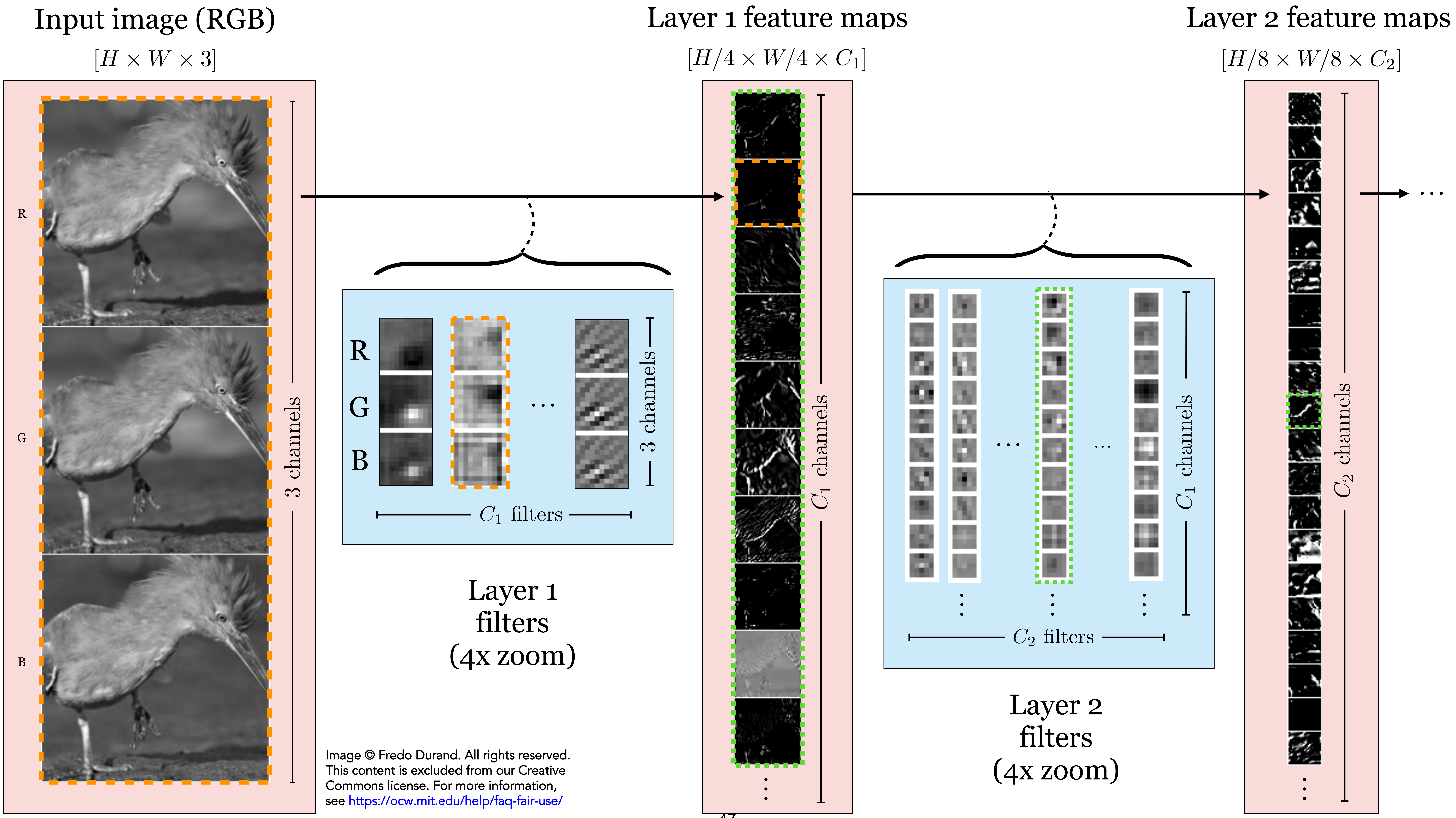
When mapping from

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

using an filter of spatial extent $K_1 \times K_2$

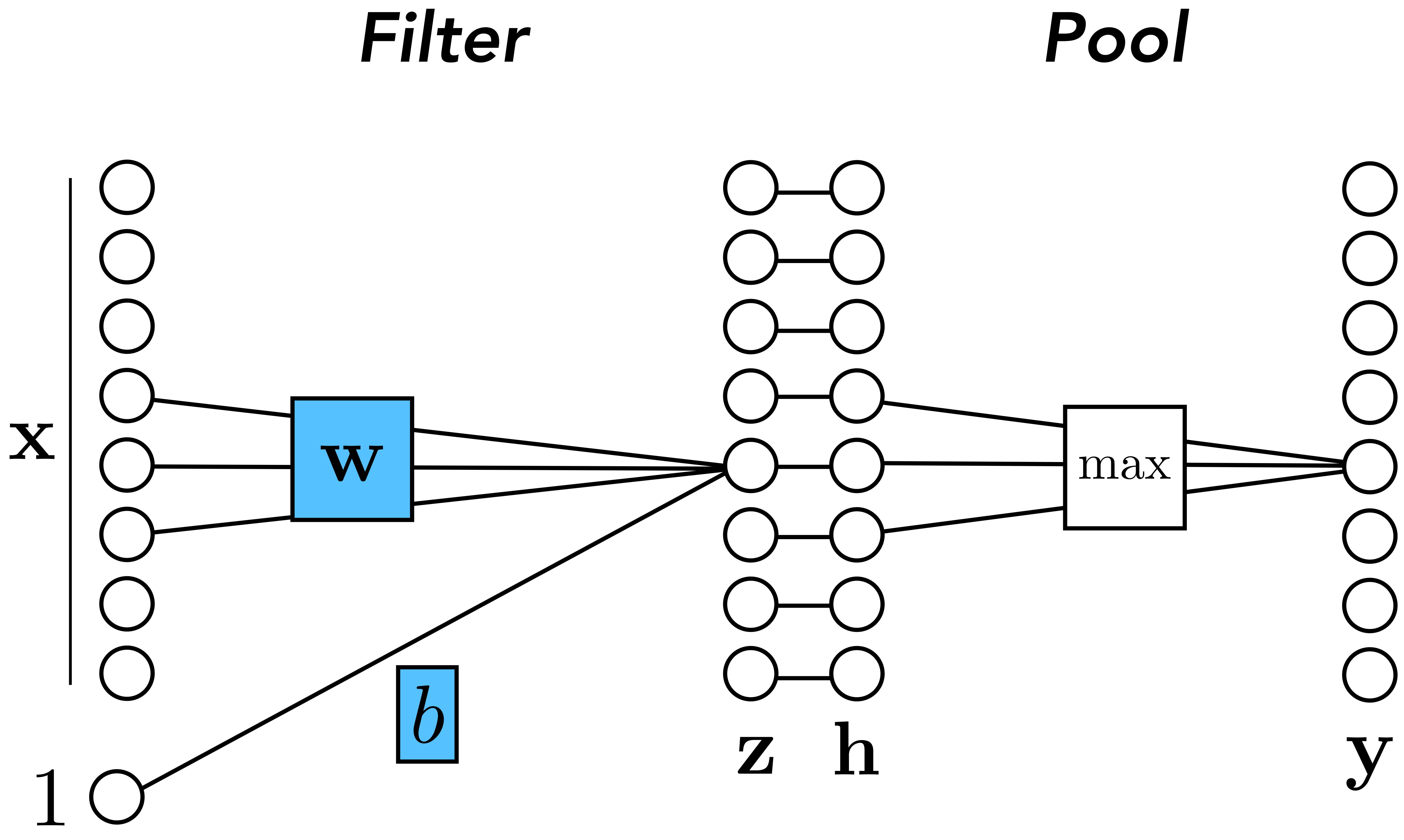
Number of parameters per filter: $K_1 \times K_2 \times C_l$

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



Input image (RGB)
$[H \times W \times 3]$
R
G
B
3 channels
Layer 1 feature maps
$[H/4 \times W/4 \times C_1]$
C_1 channels
Layer 2 feature maps
$[H/8 \times W/8 \times C_2]$
C_2 channels
R
G
B
3 channels
C_1 filters
Layer 1 filters (4x zoom)
C_1 channels
C_2 filters
Layer 2 filters (4x zoom)

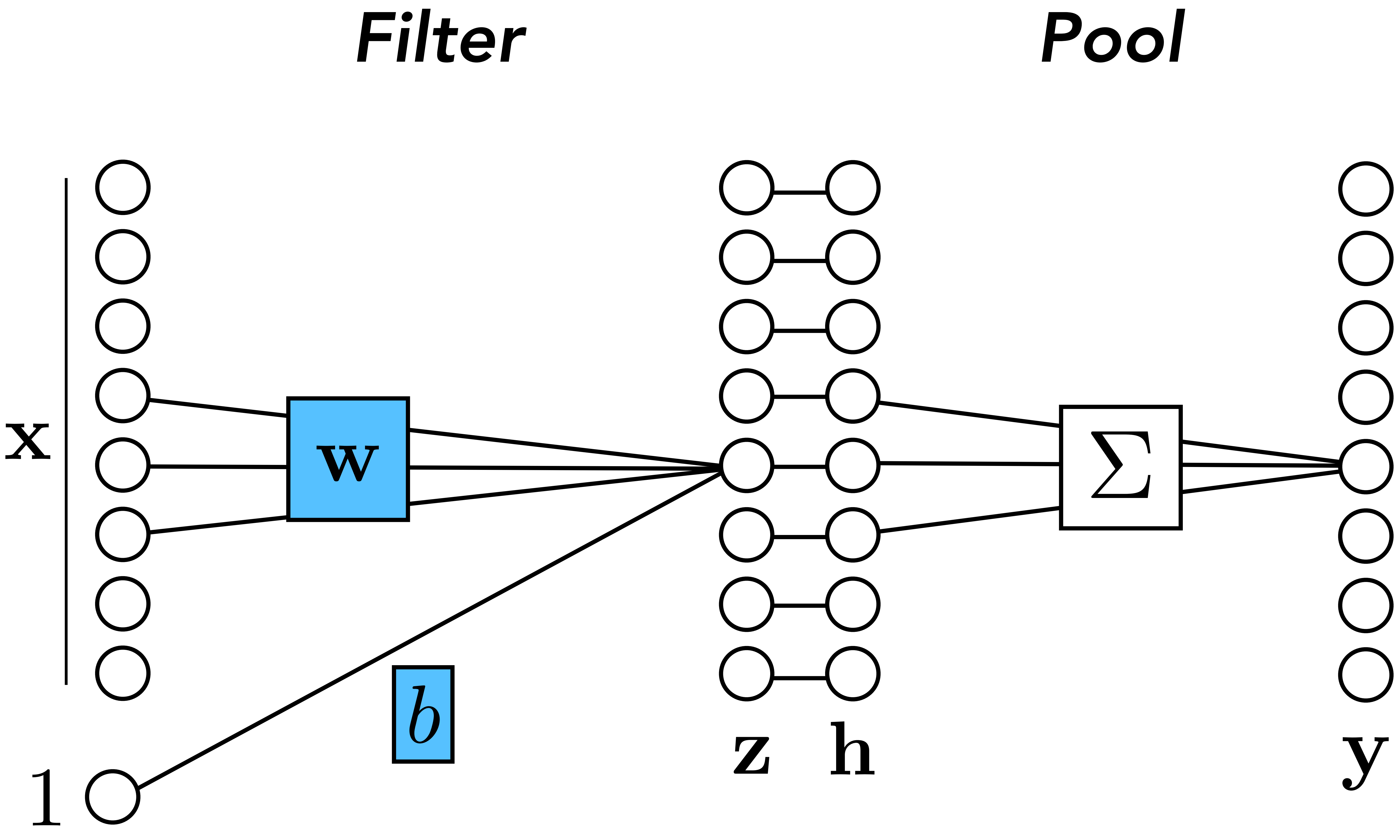
Pooling



Max pooling

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

Pooling



Max pooling

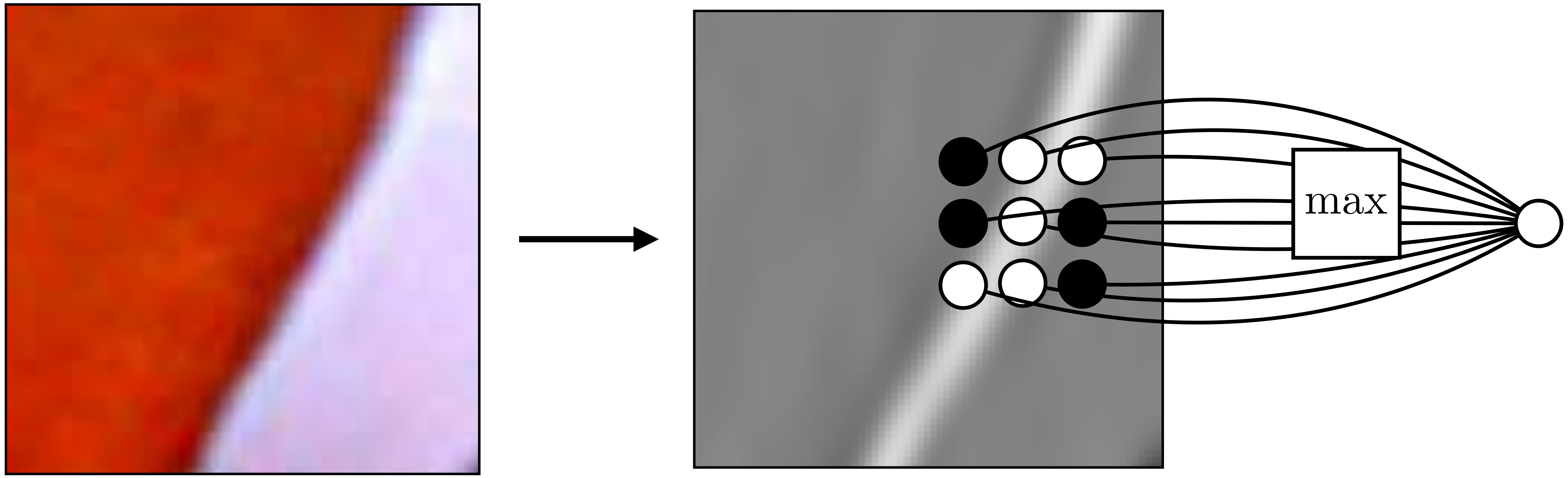
$$y_j = \max_{j \in \mathcal{N}(j)} h_j$$

Mean pooling

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

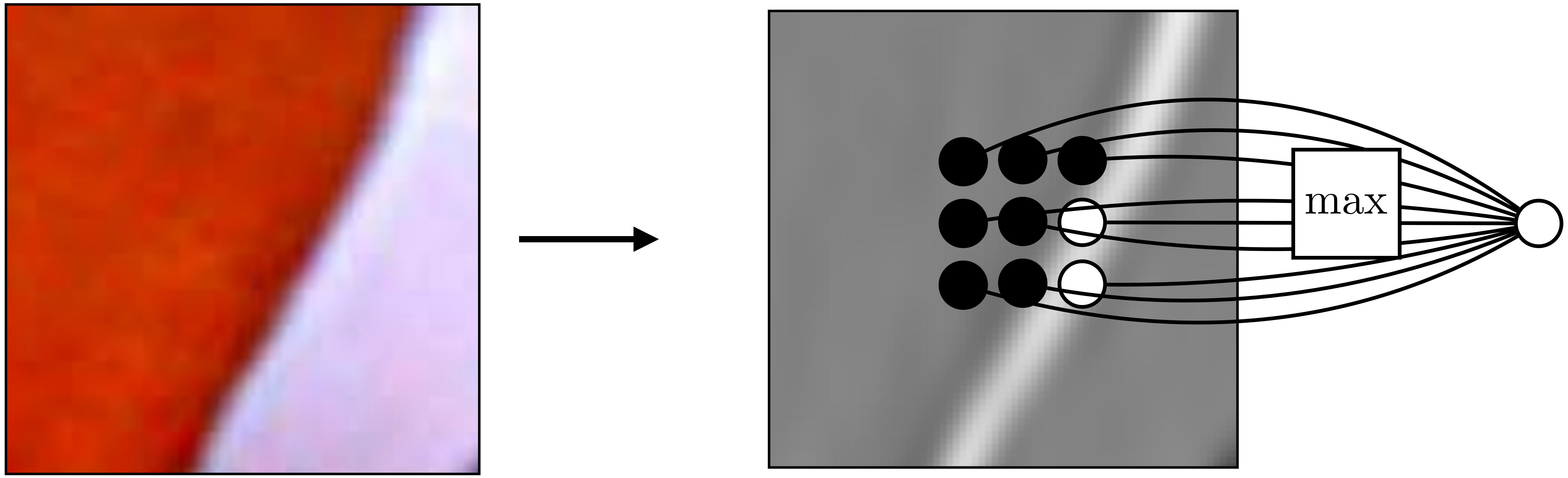
Pooling — Why?

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



Pooling — Why?

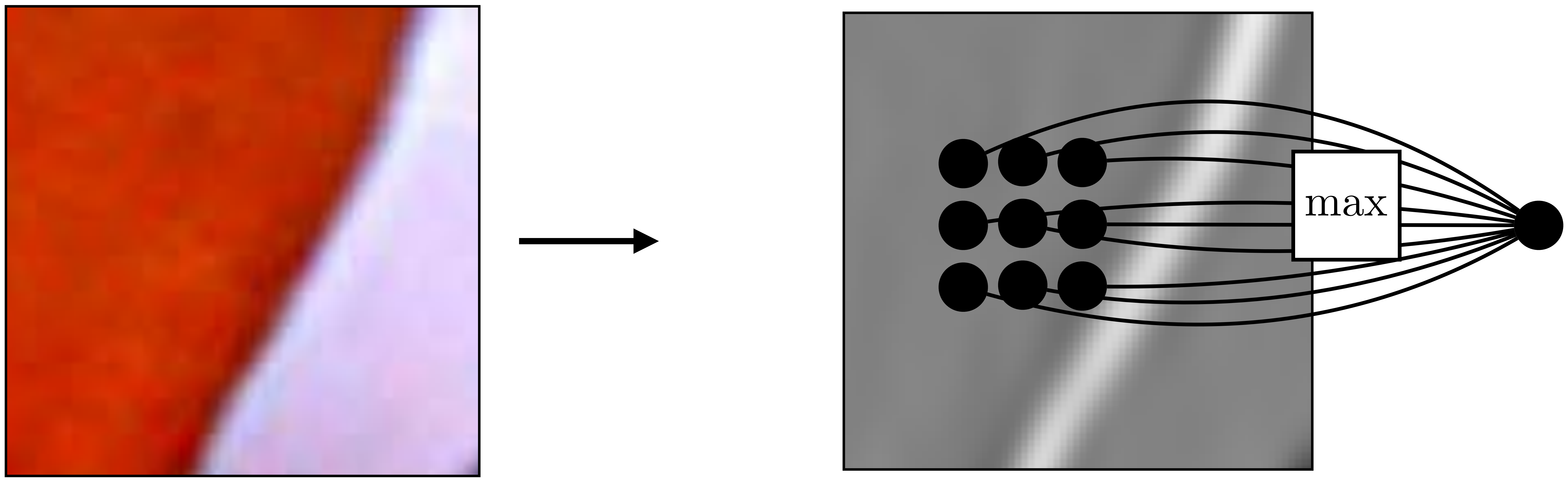
Pooling across spatial locations achieves stability w.r.t. small translations:



large response regardless of exact position of edge

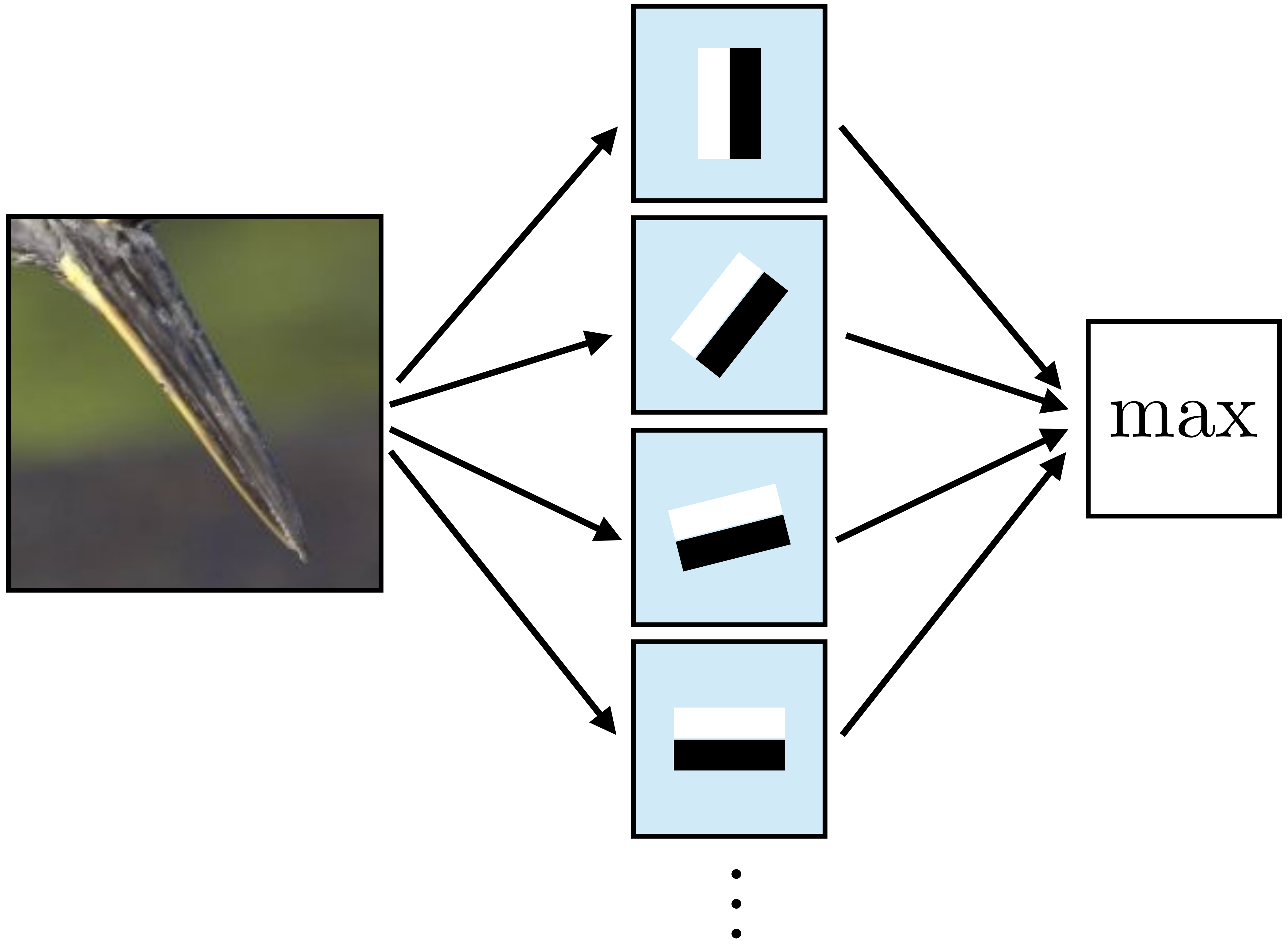
Pooling — Why?

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



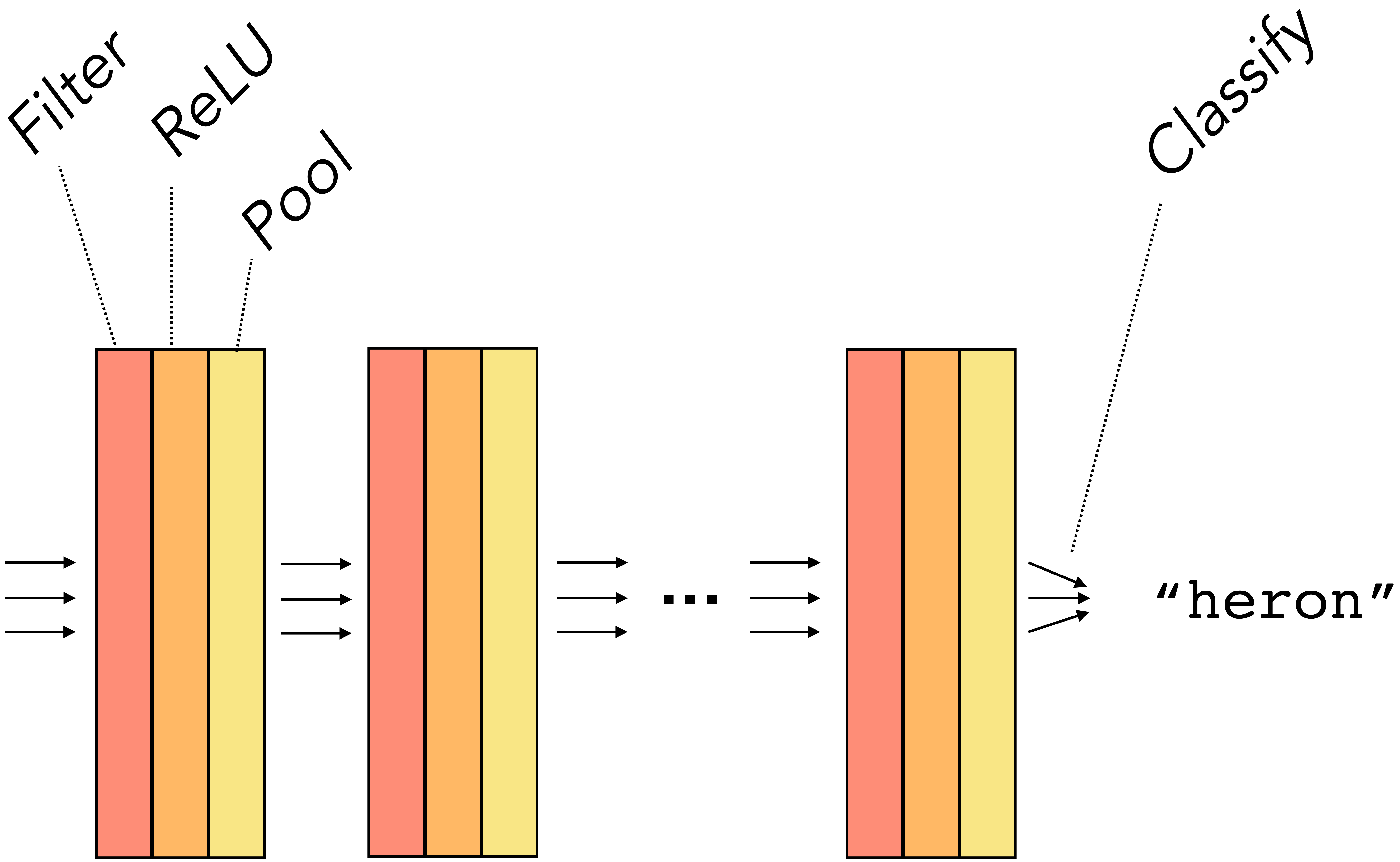
Pooling *across channels* — Why?

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



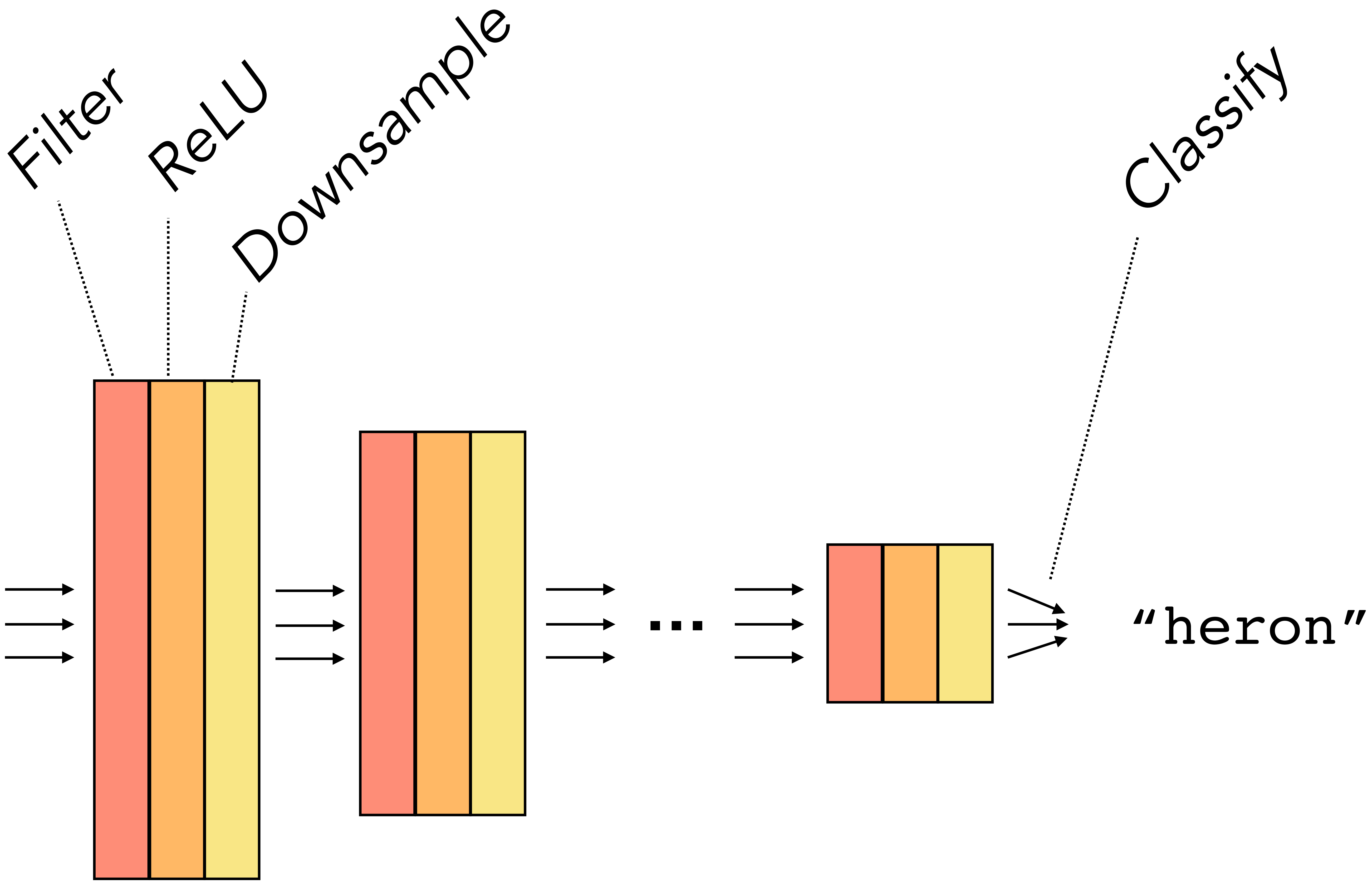
large response for any edge, regardless of its orientation

Computation in a neural net



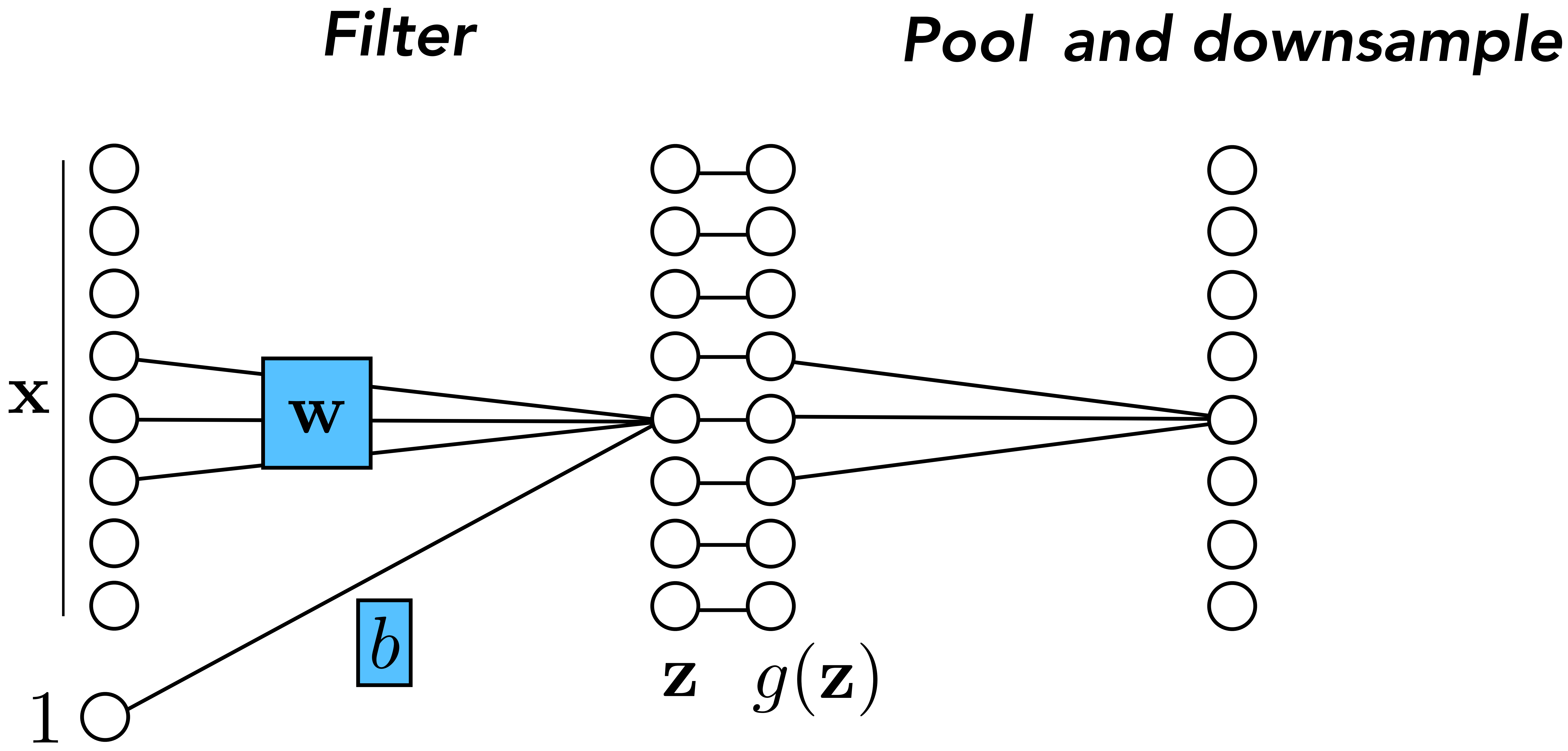
$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

Computation in a neural net

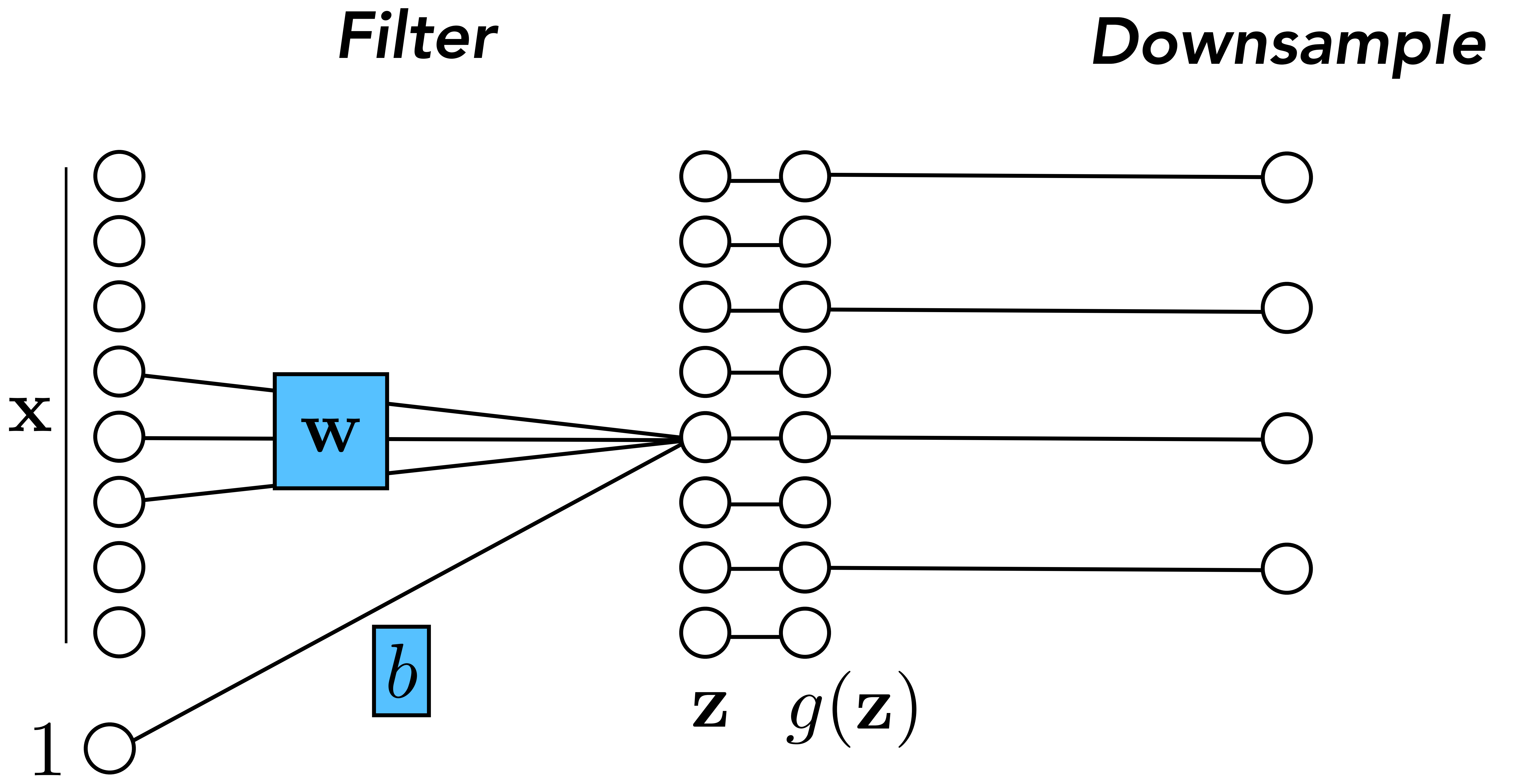


$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x})))$$

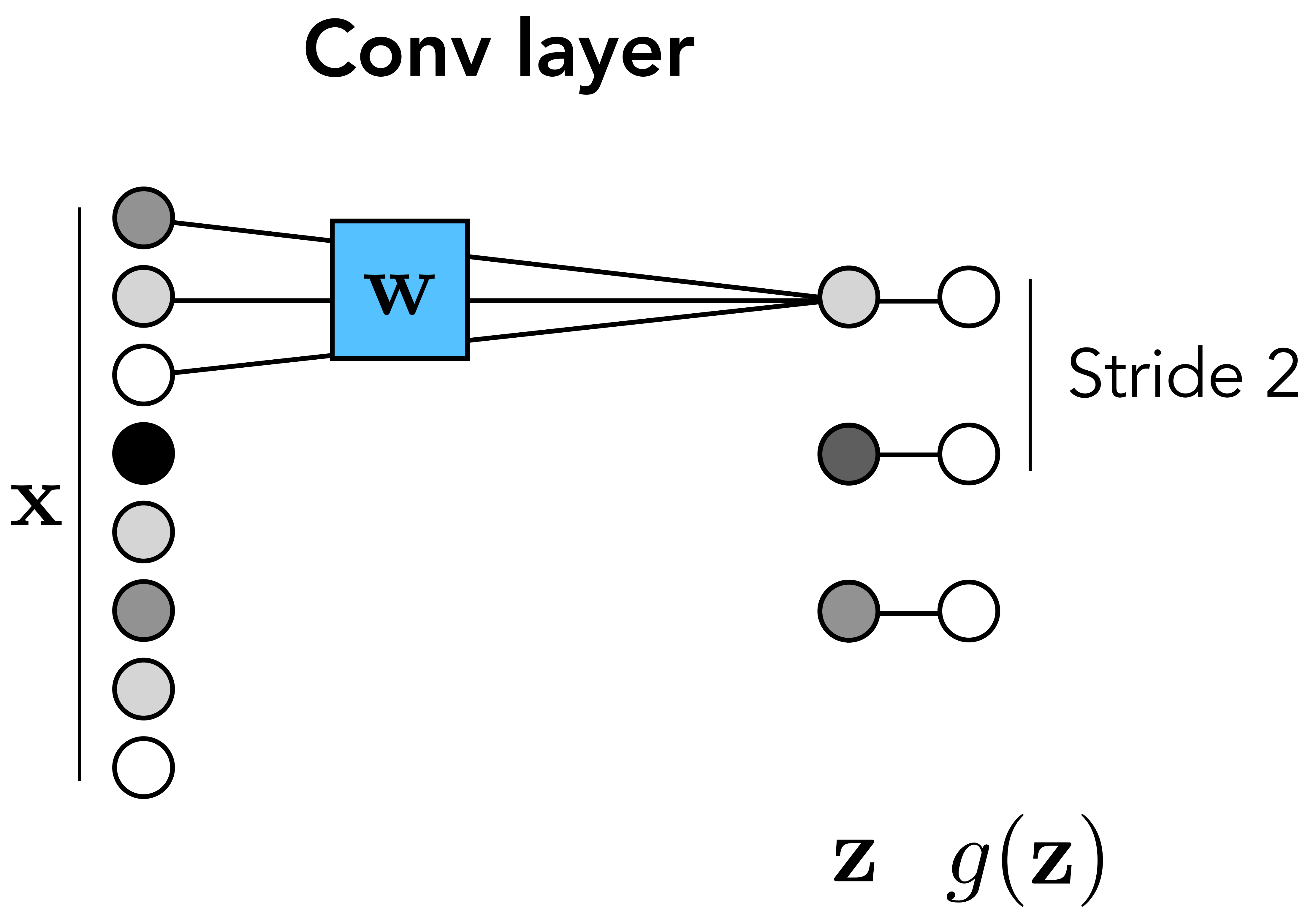
Downsampling



Downsampling

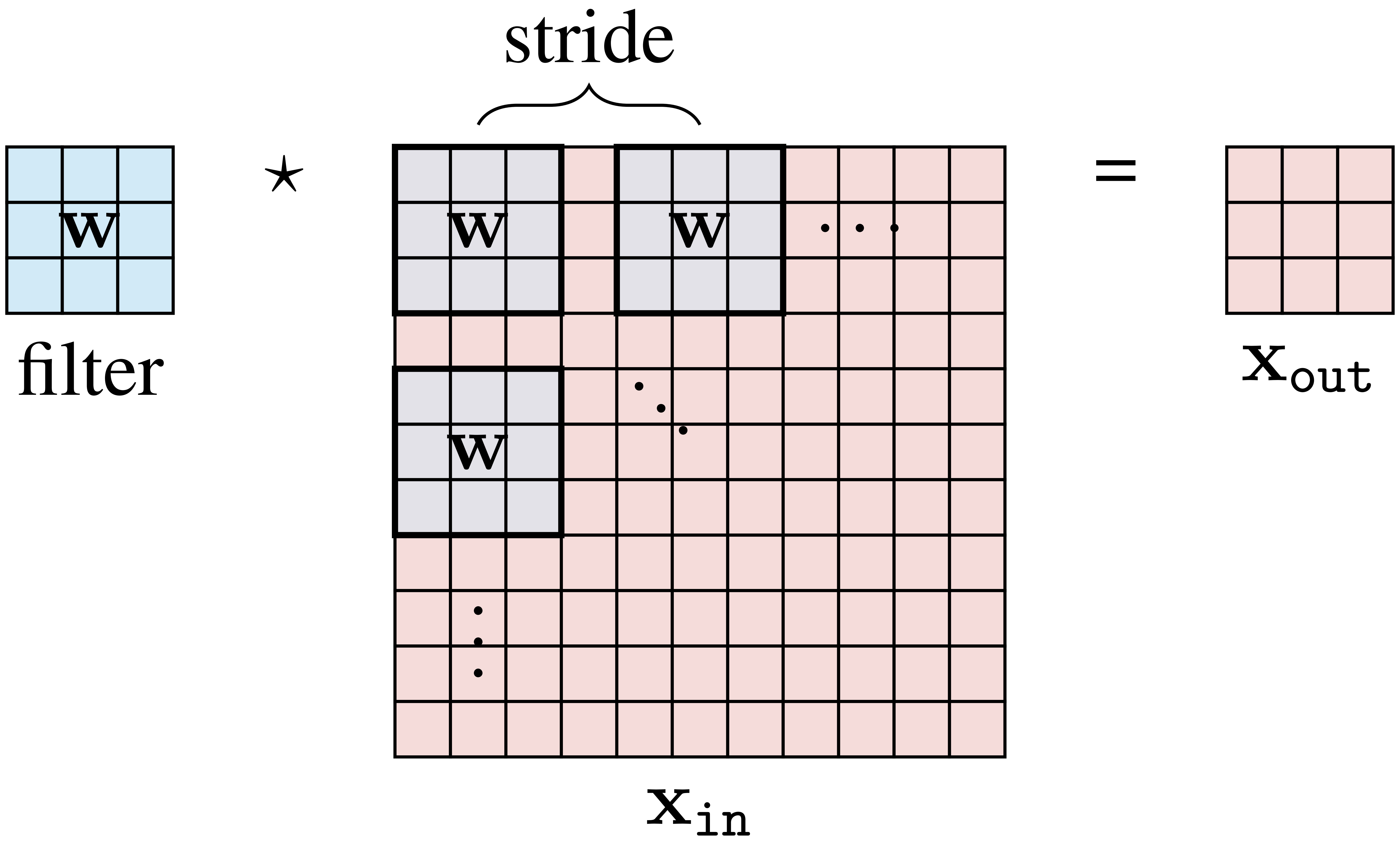


Strided operations

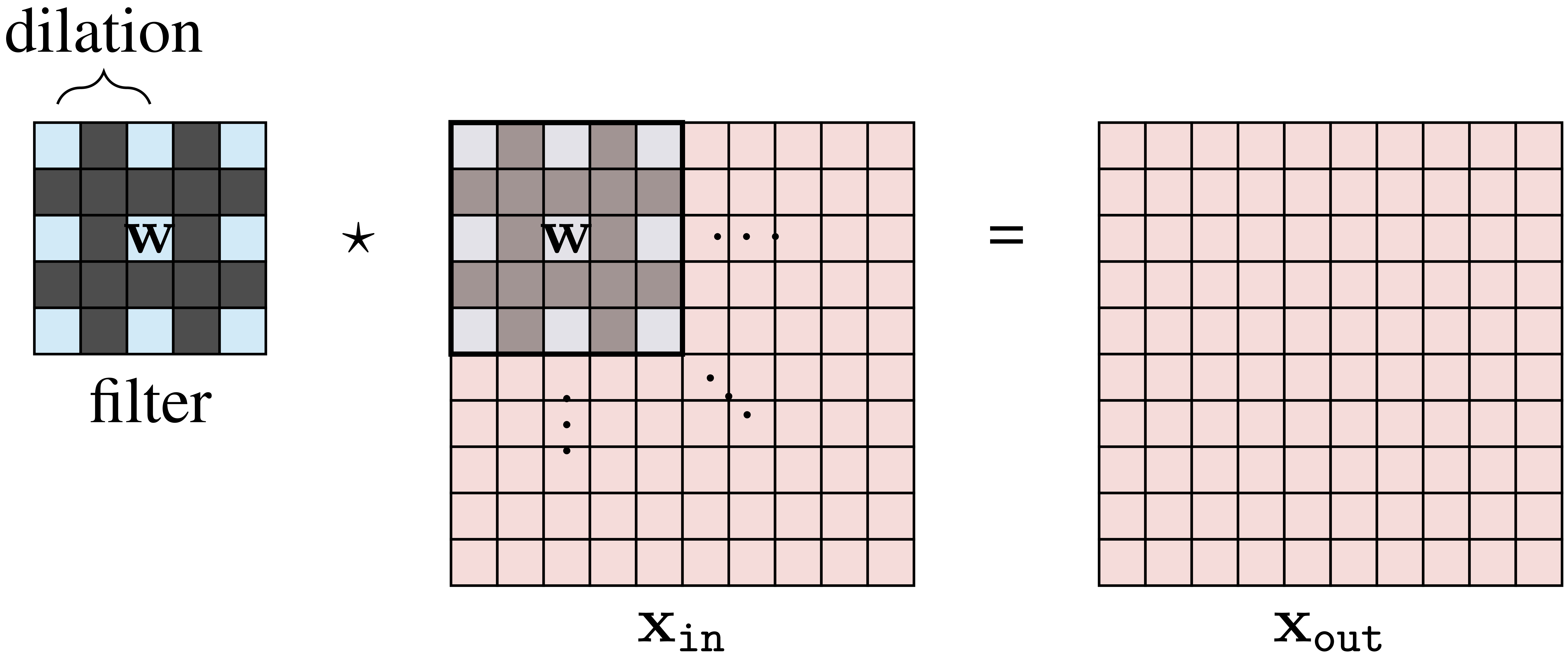


Strided operations combine a given operation (convolution or pooling) and downsampling into a single operation.

Strided operations (2D)

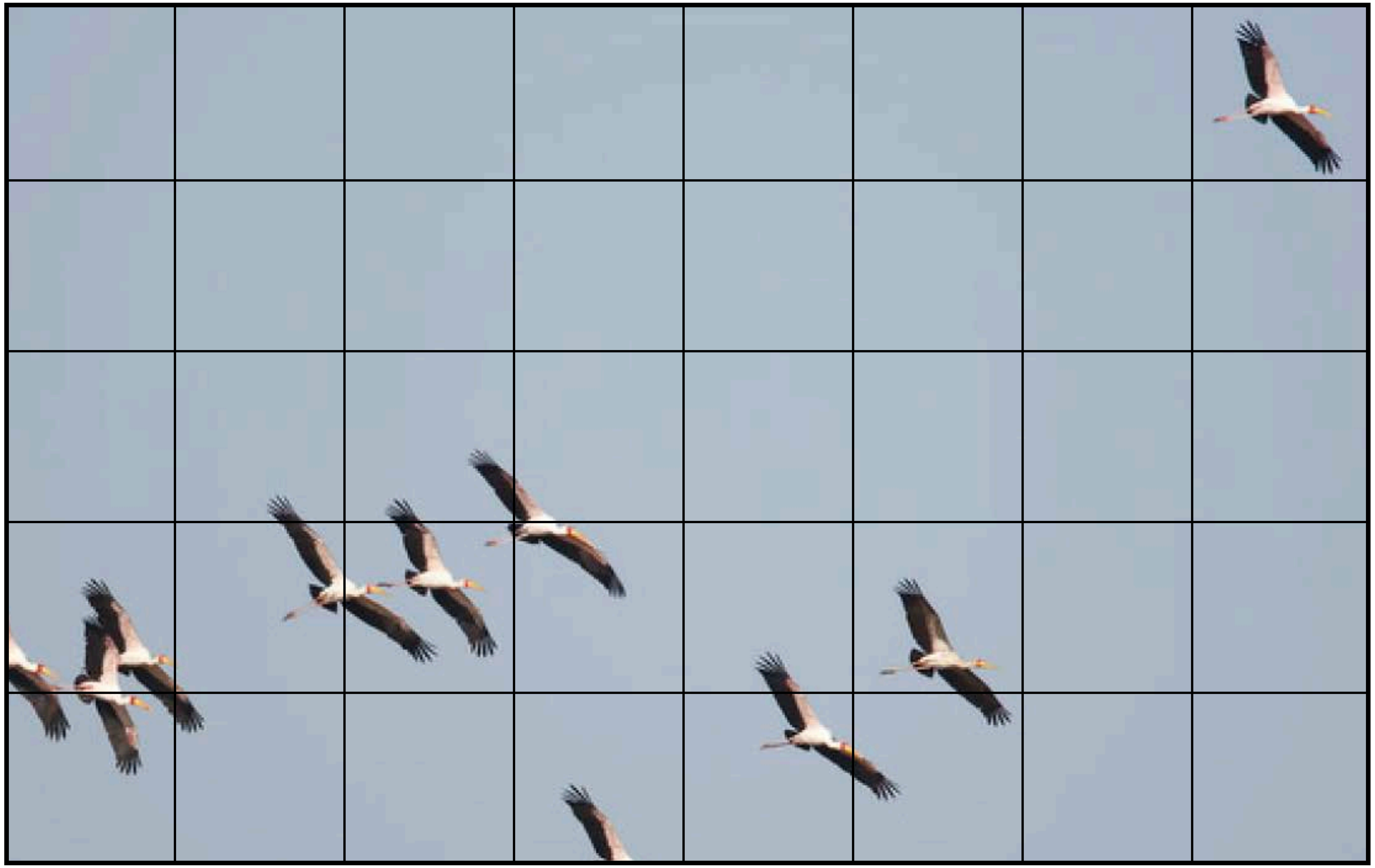


Dilated filter

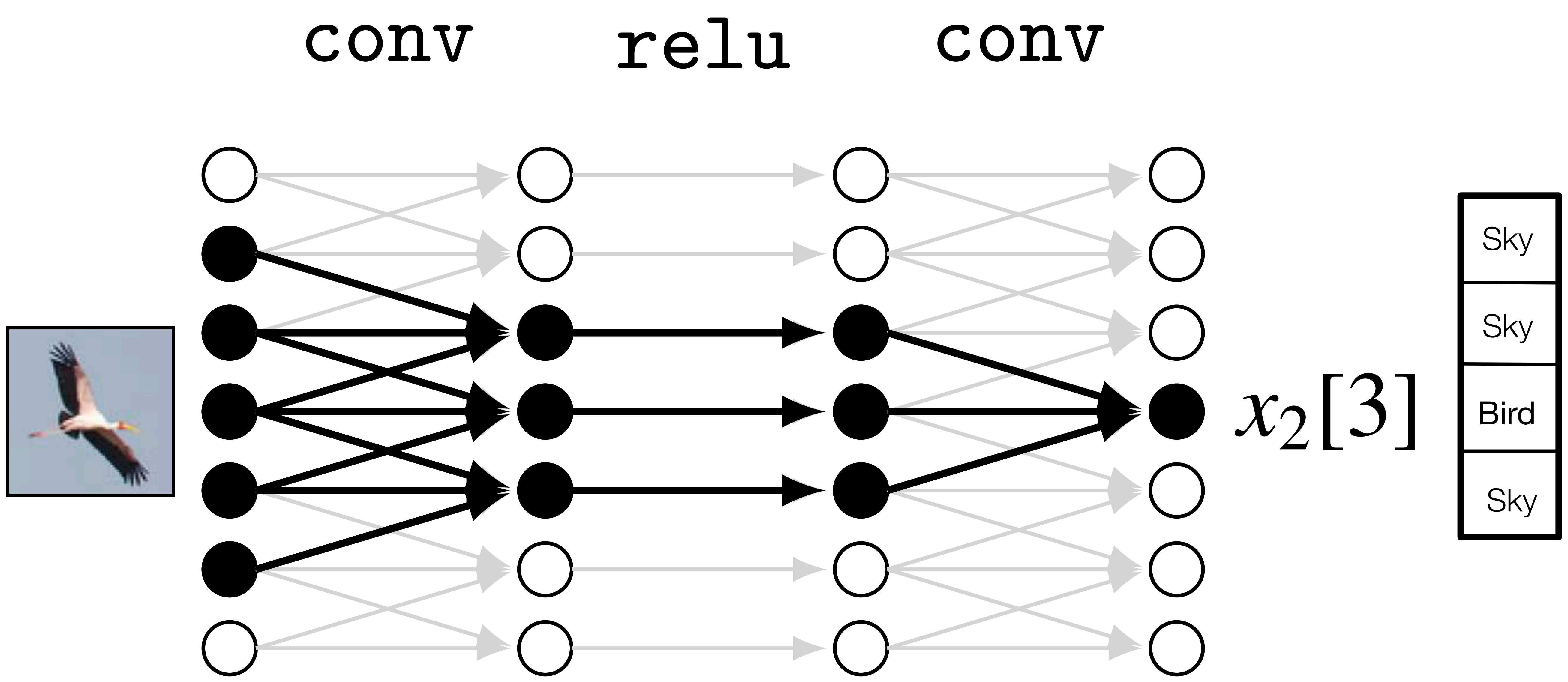


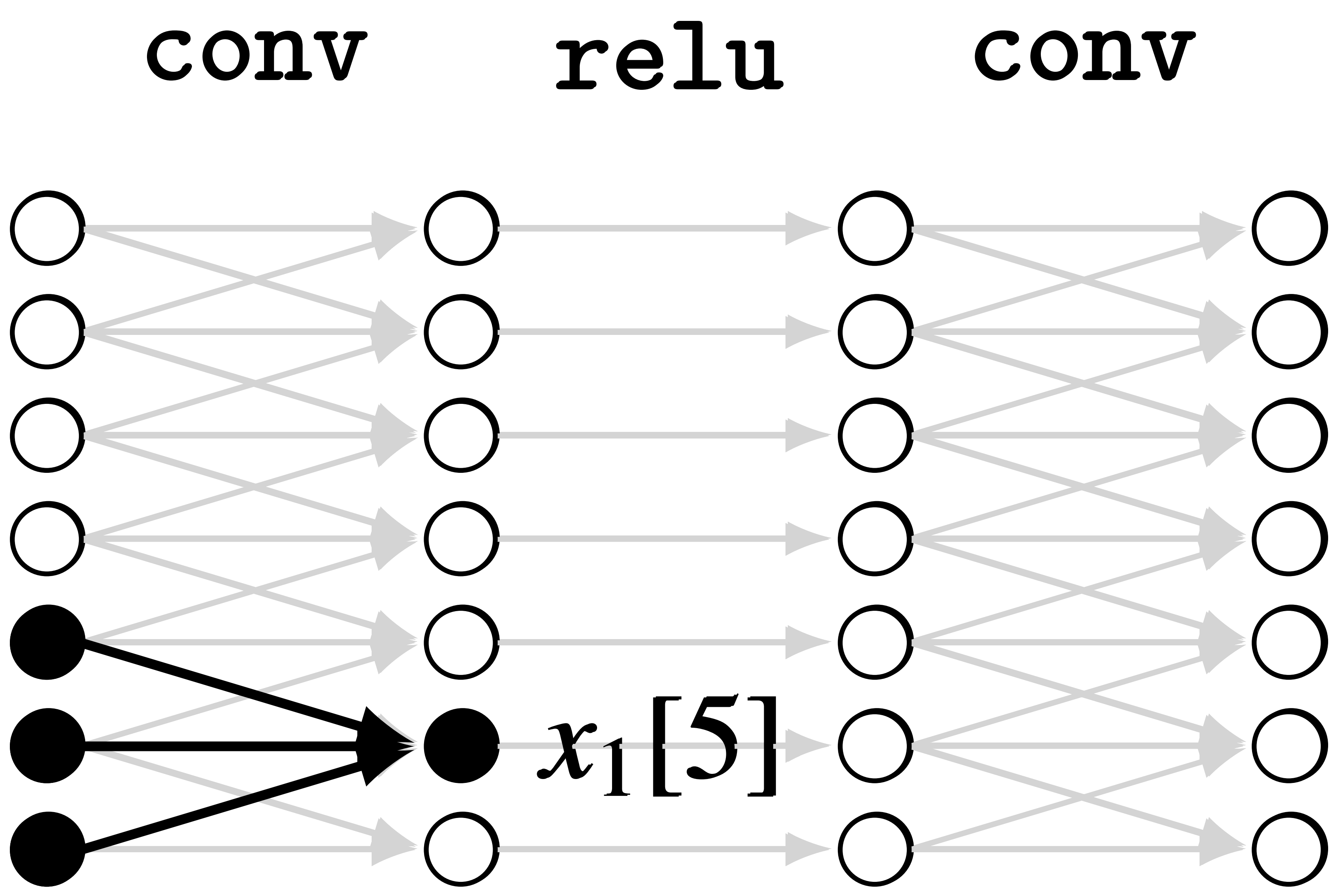
Covers a large receptive field with fewer parameters.

Receptive fields



Receptive fields






input image
conv1
conv2
conv3
conv4
conv5
alexnet
input image
conv1
conv4
conv7
conv10
conv13
vgg16
input image
conv1
block2
block4
block6
block8
resnet18

Implementing conv

Basic implementation:

1. `im2col`: NxMxC —> Npatches x (K x K x C)
2. `bmm`: batch `matmul` with kernel
3. `col2im`: Npatches x (K x K x C) —> NxMxC

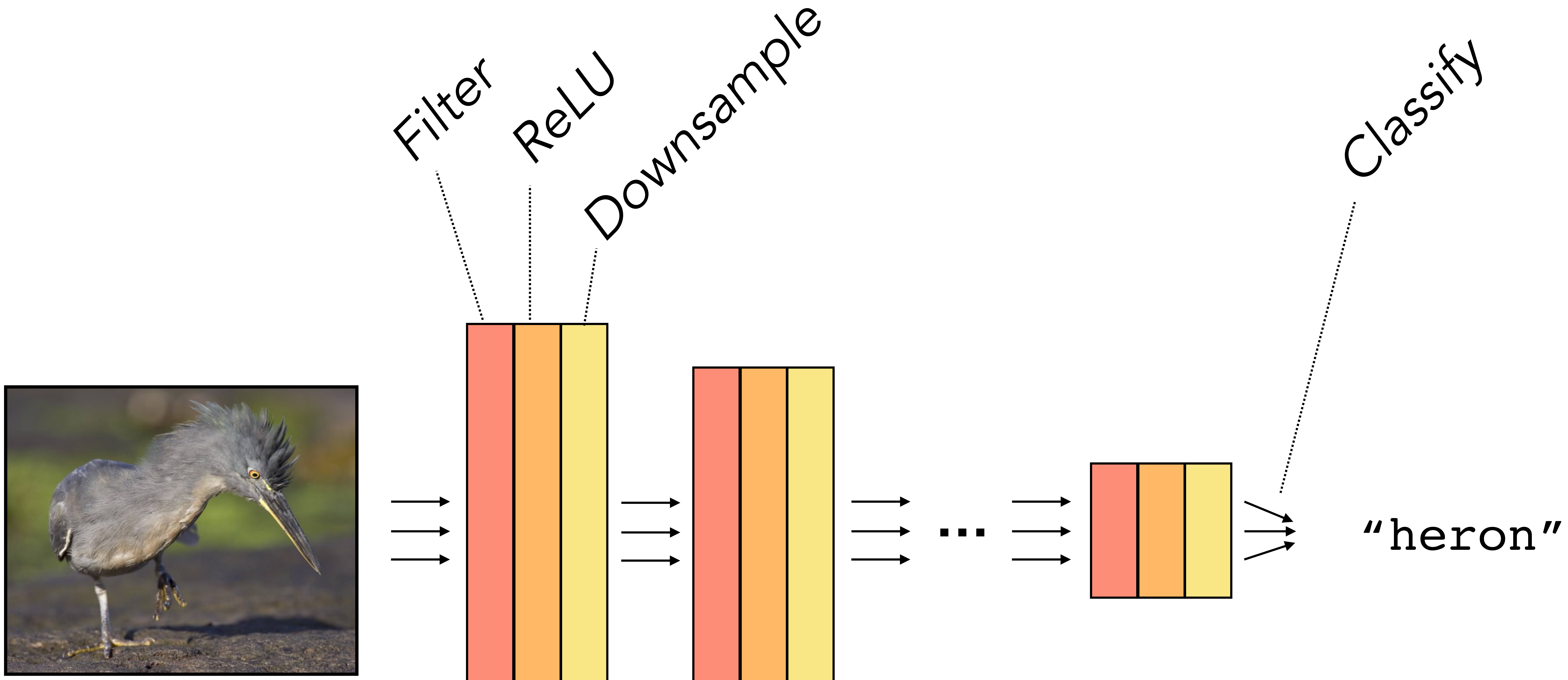
or: fft signal processing stuff…

Useful library:

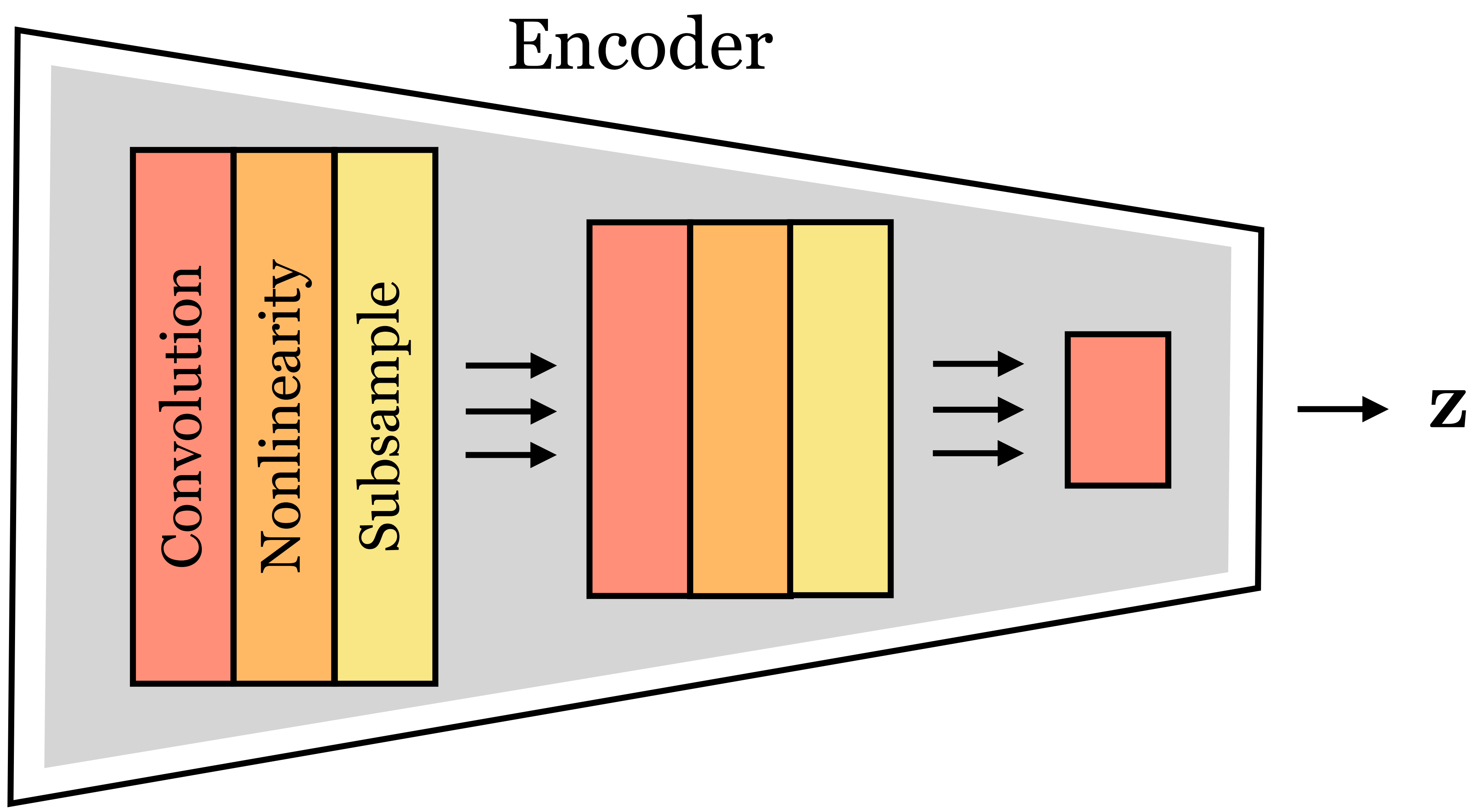
timm: `https://github.com/rwightman/pytorch-image-models`

Popular CNN Architectures

Computation in a neural net



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


Encoder
Convolution
Nonlinearity
Subsample
z

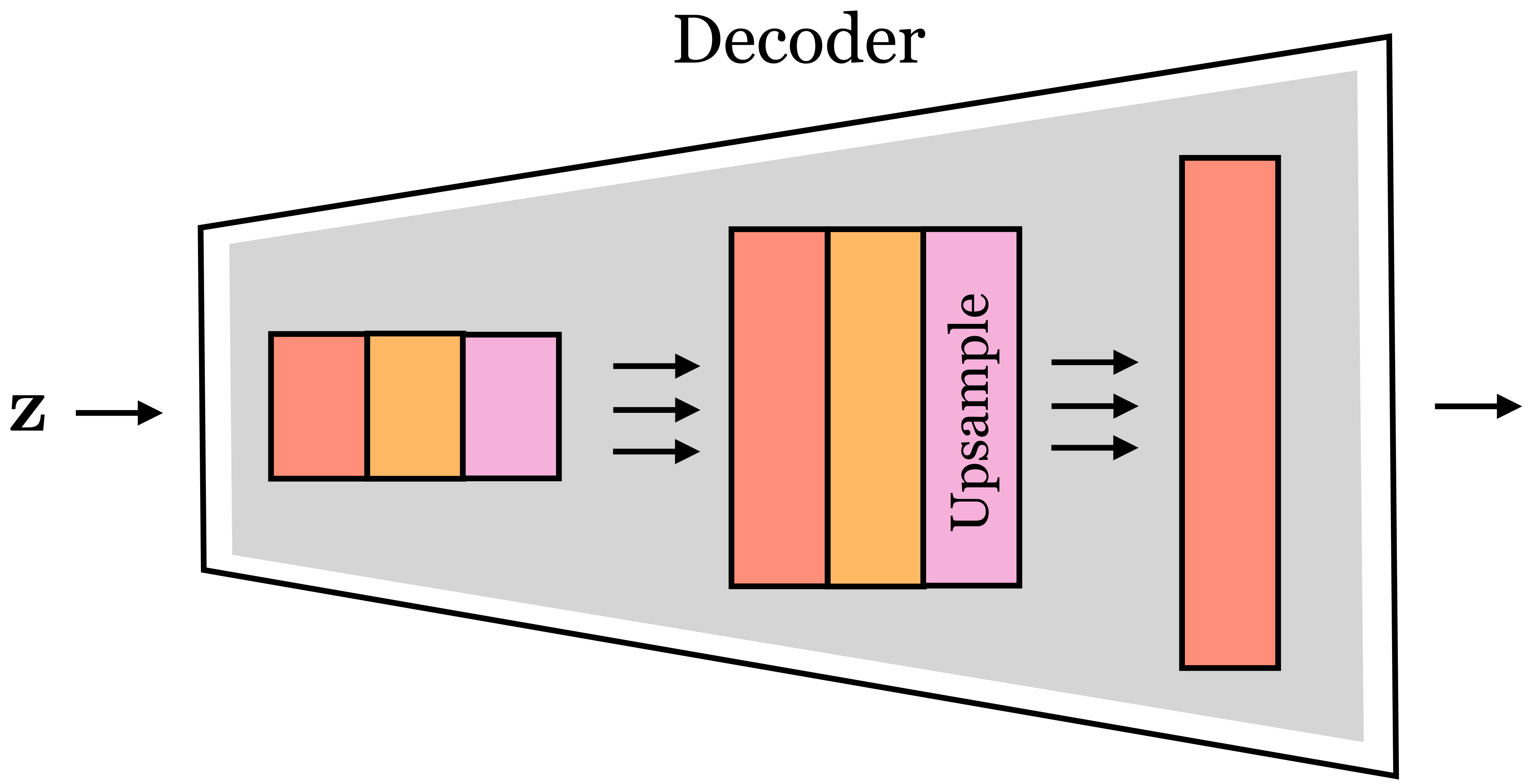

Decoder
z
Upsample

Image-to-image

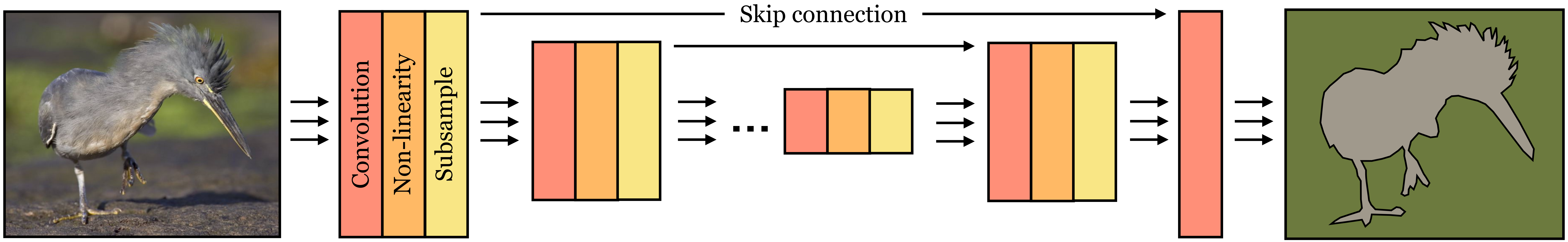
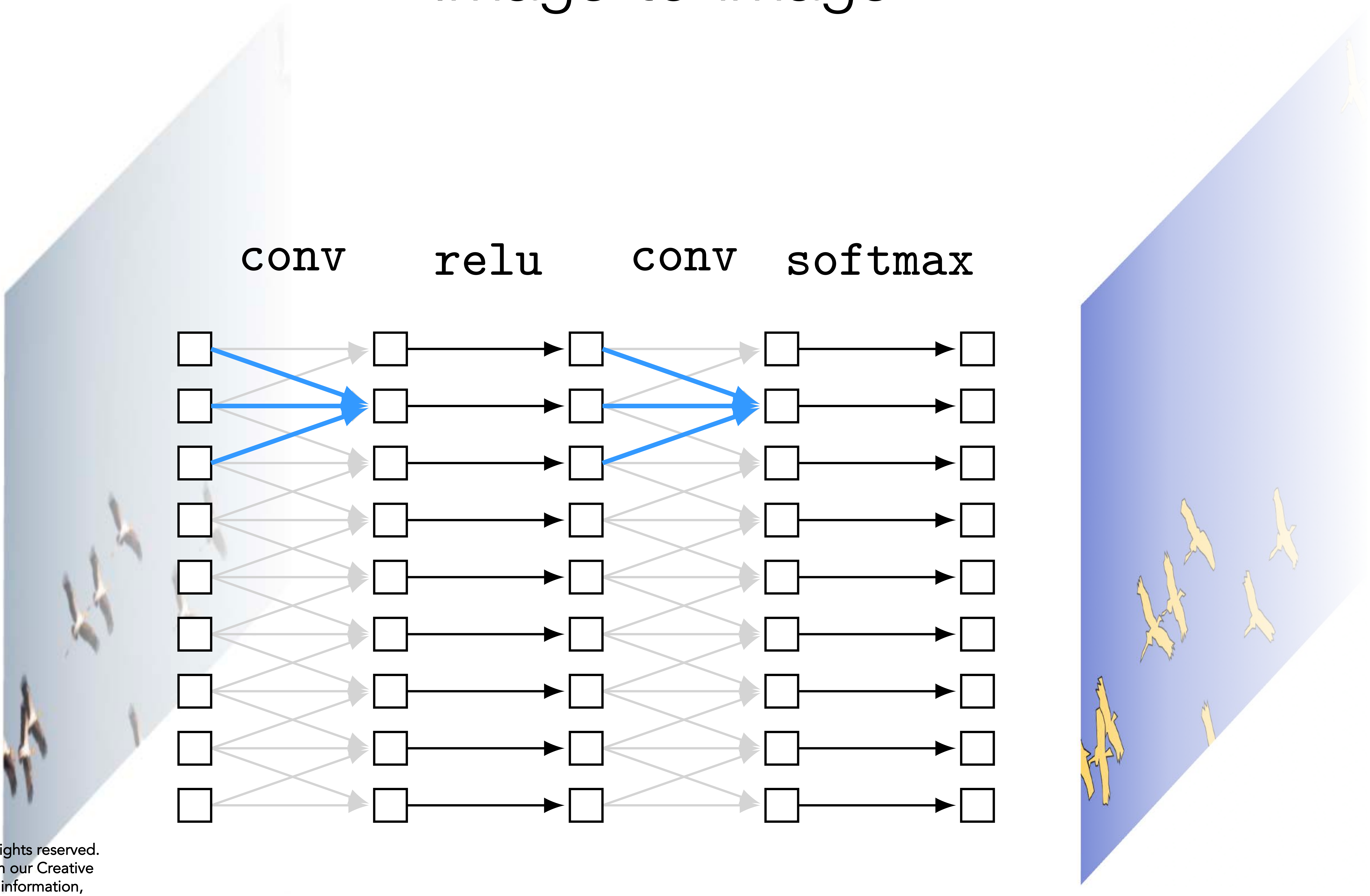
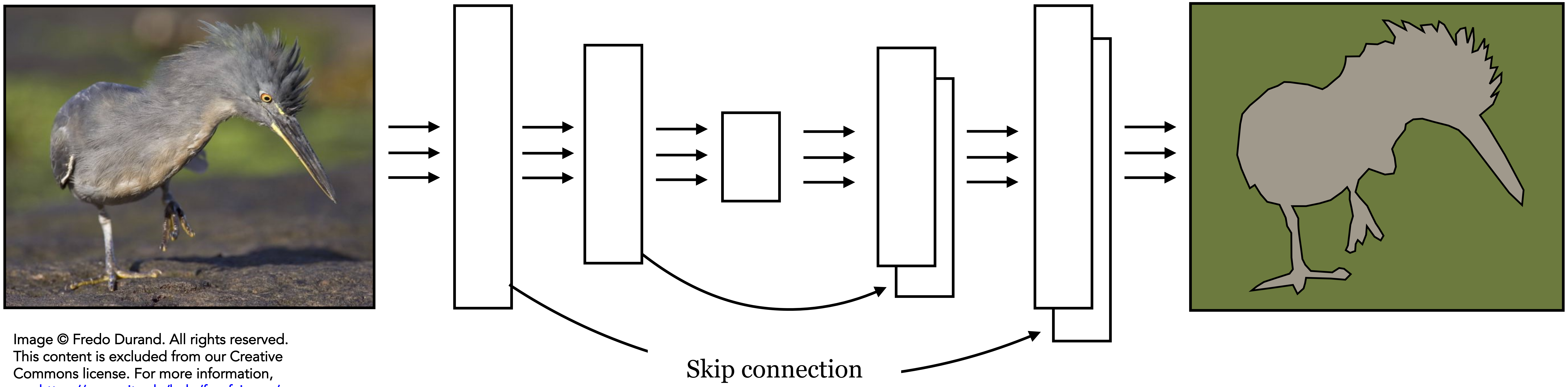


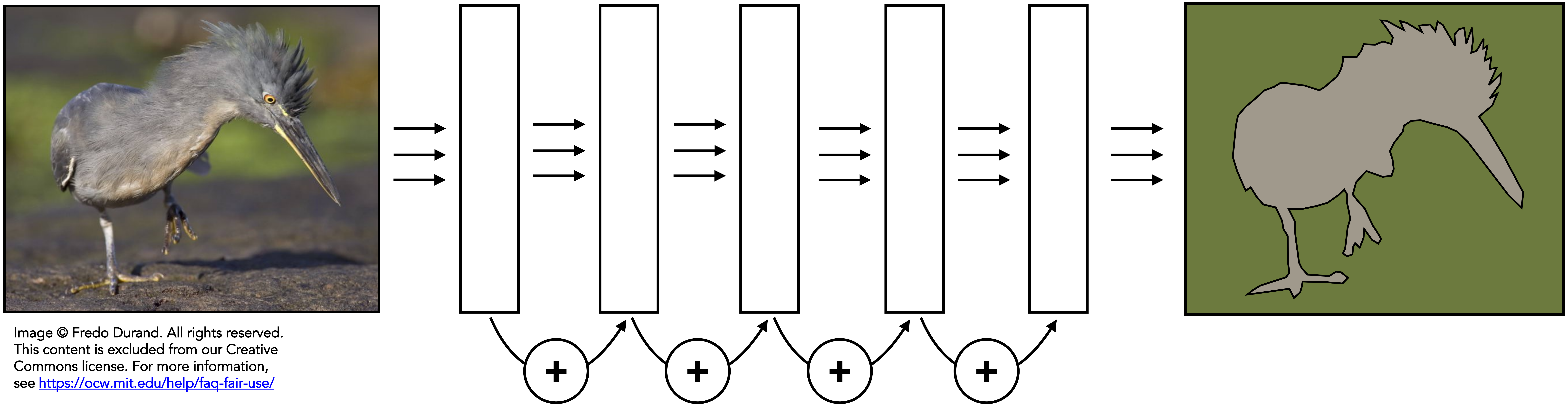
Image-to-image



U-net



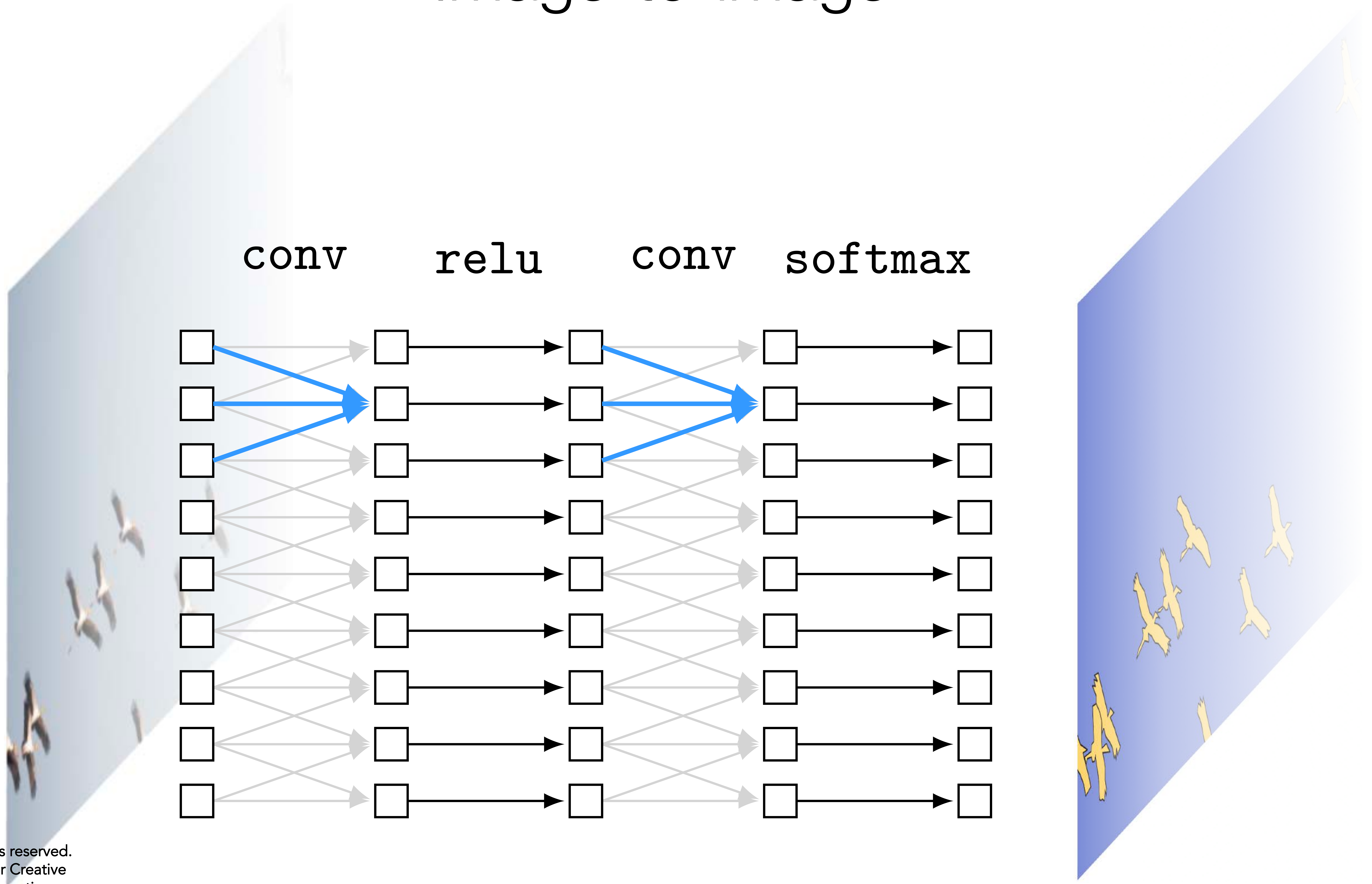
ResNet



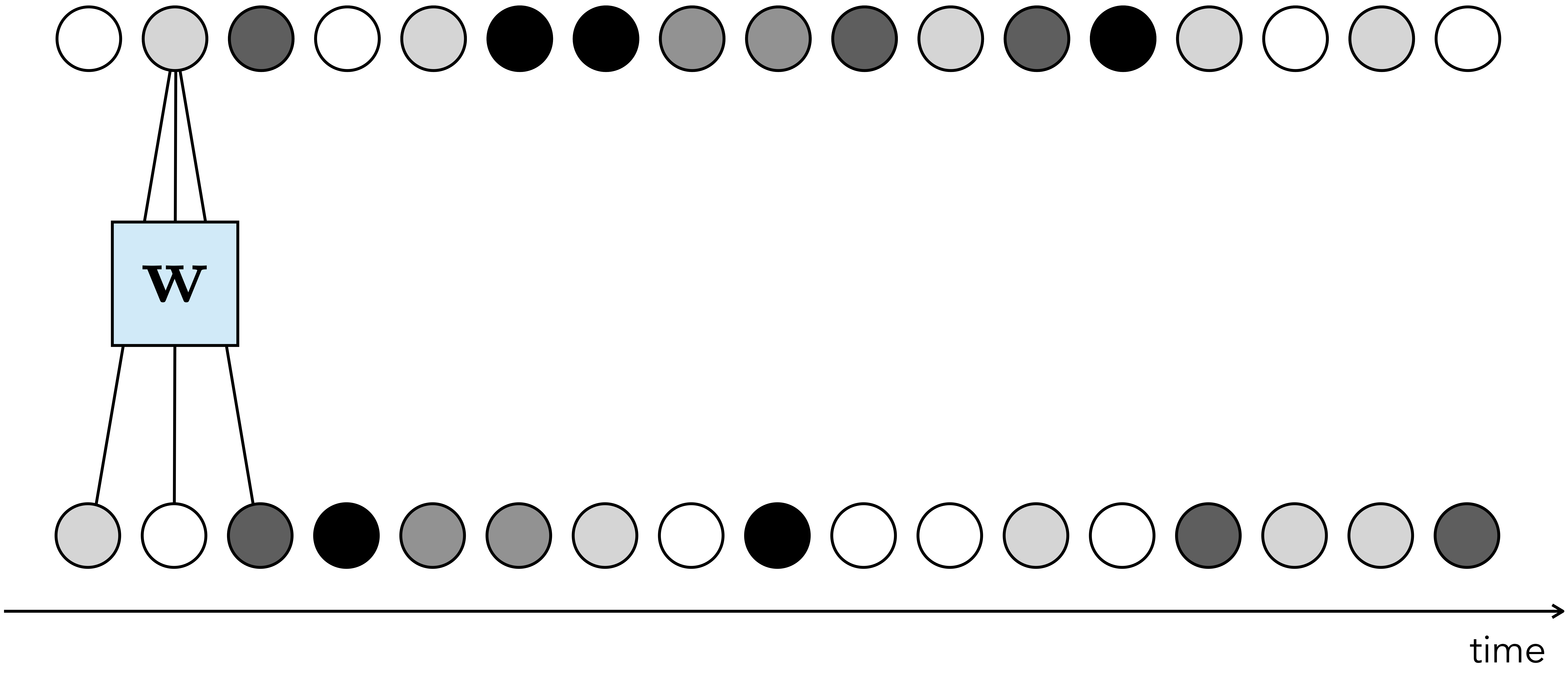
Residual connection: $\mathbf{x}_{\texttt{out}} = F(\mathbf{x}_{\texttt{in}}) + \mathbf{x}_{\texttt{in}}$

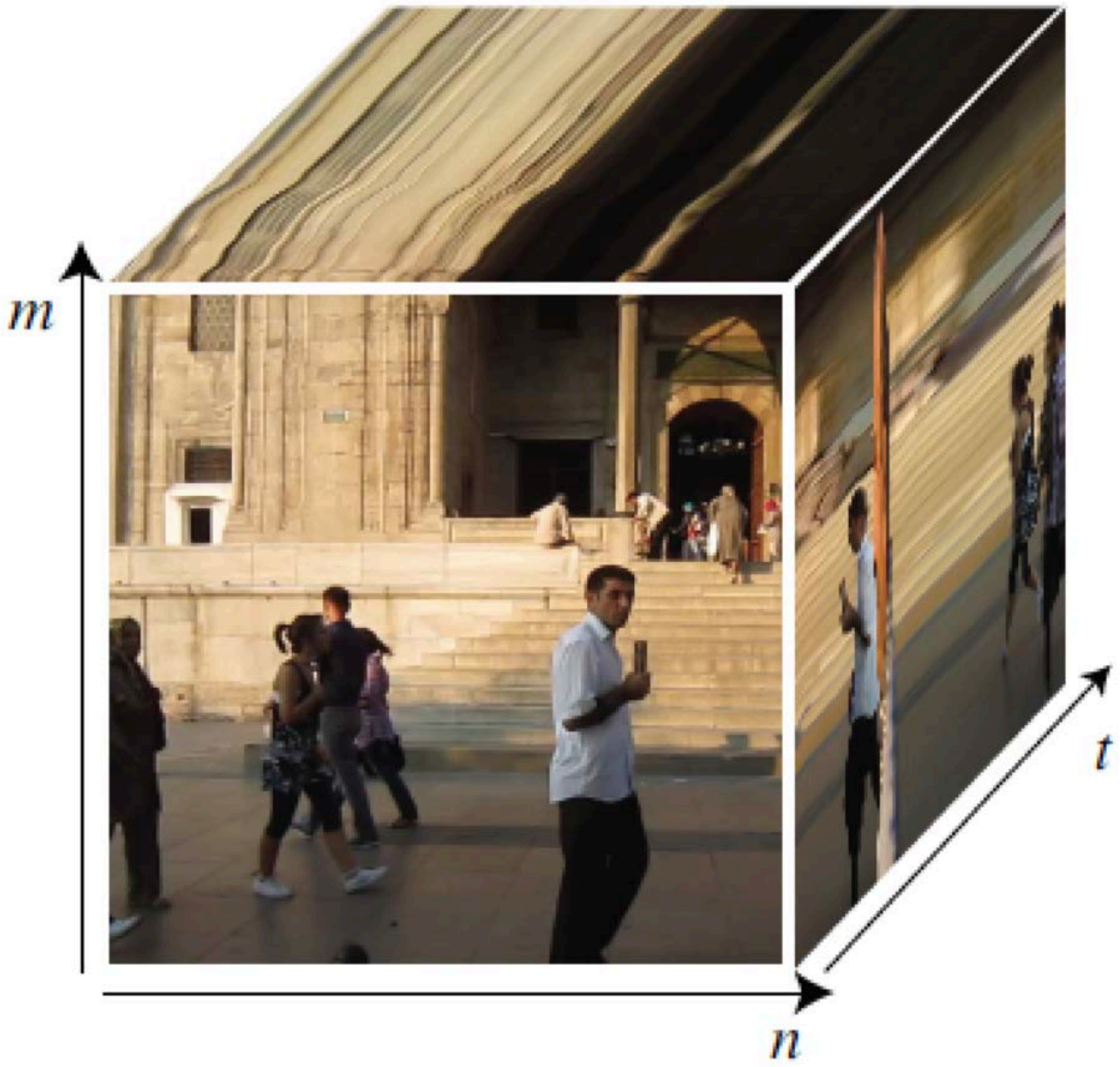
Or, if you want to change dimensionality: $\mathbf{x}_{\texttt{out}} = F(\mathbf{x}_{\texttt{in}}) + \mathbf{W}\mathbf{x}_{\texttt{in}}$

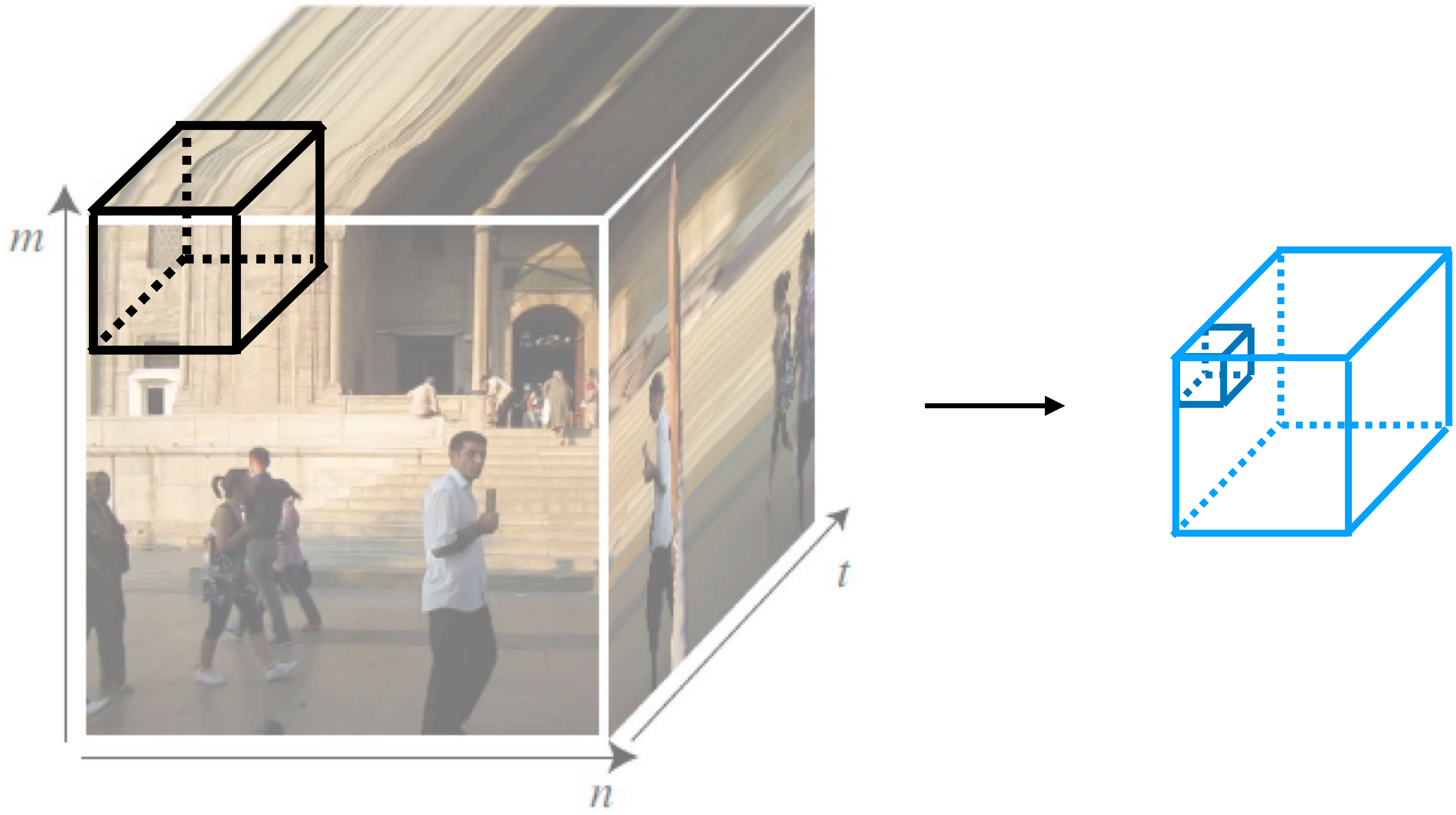
Image-to-image



Convolutions in time




m
n
t



m
n
t

What if you *don't* want to be shift invariant?

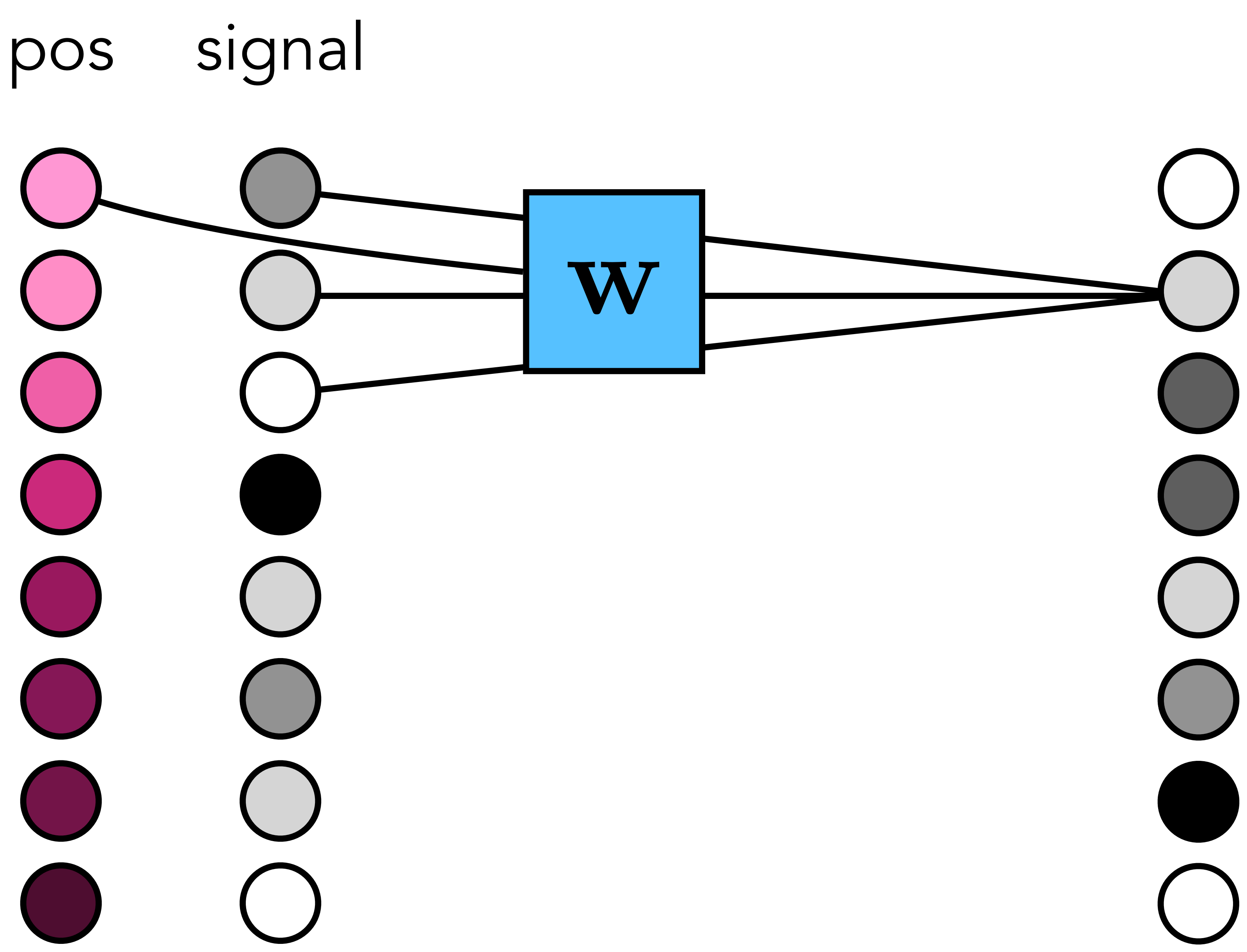
1. Use an architecture that is not shift invariant (e.g., MLP)

2. Add location information to the *input* to the convolutional filters — this is called **positional encoding**

What if you *don't* want to be shift invariant?

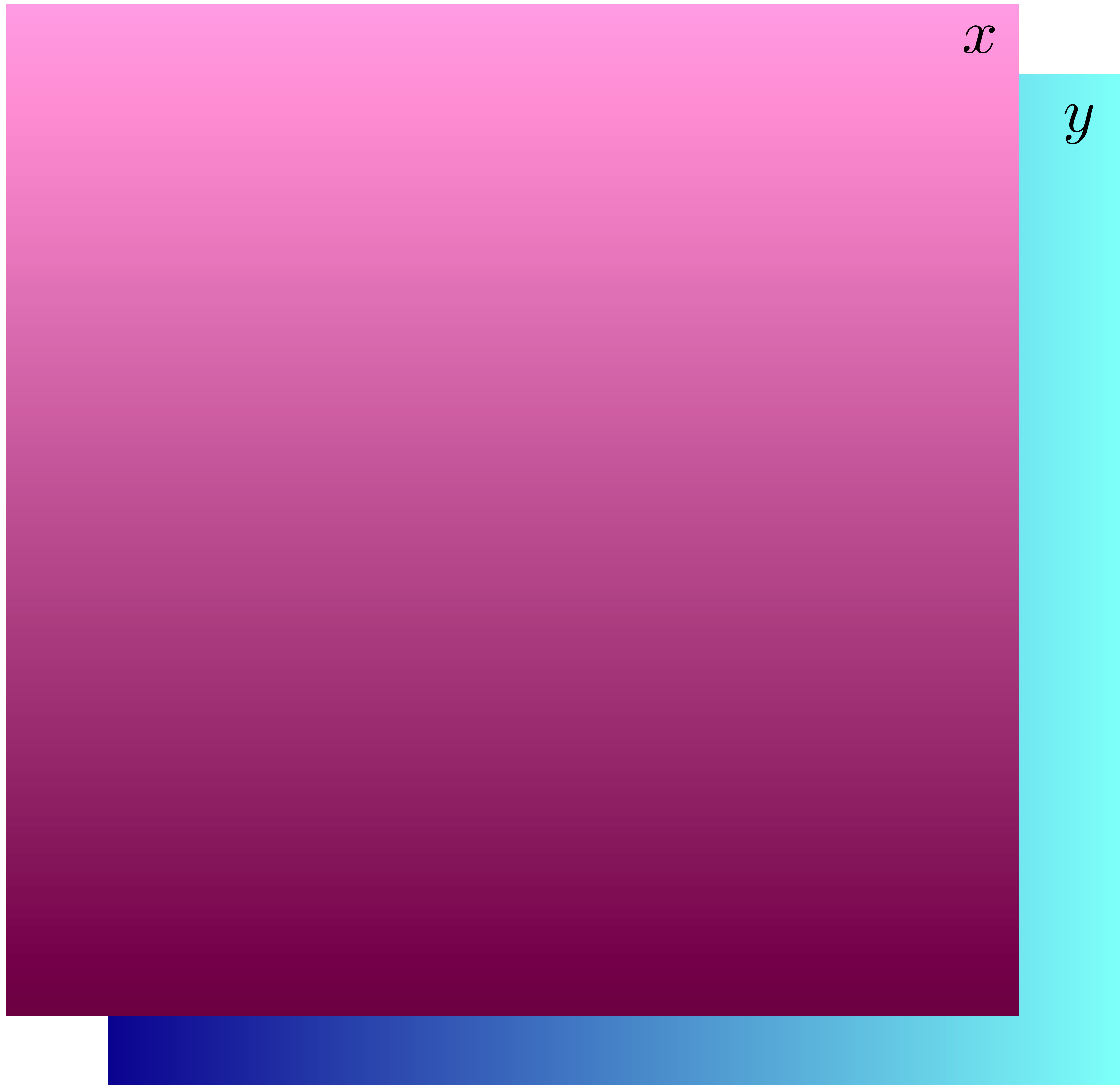
1. Use an architecture that is not shift invariant (e.g., MLP)

2. Add location information to the *input* to the convolutional filters — this is called **positional encoding**



Neural Fields

Coordinates



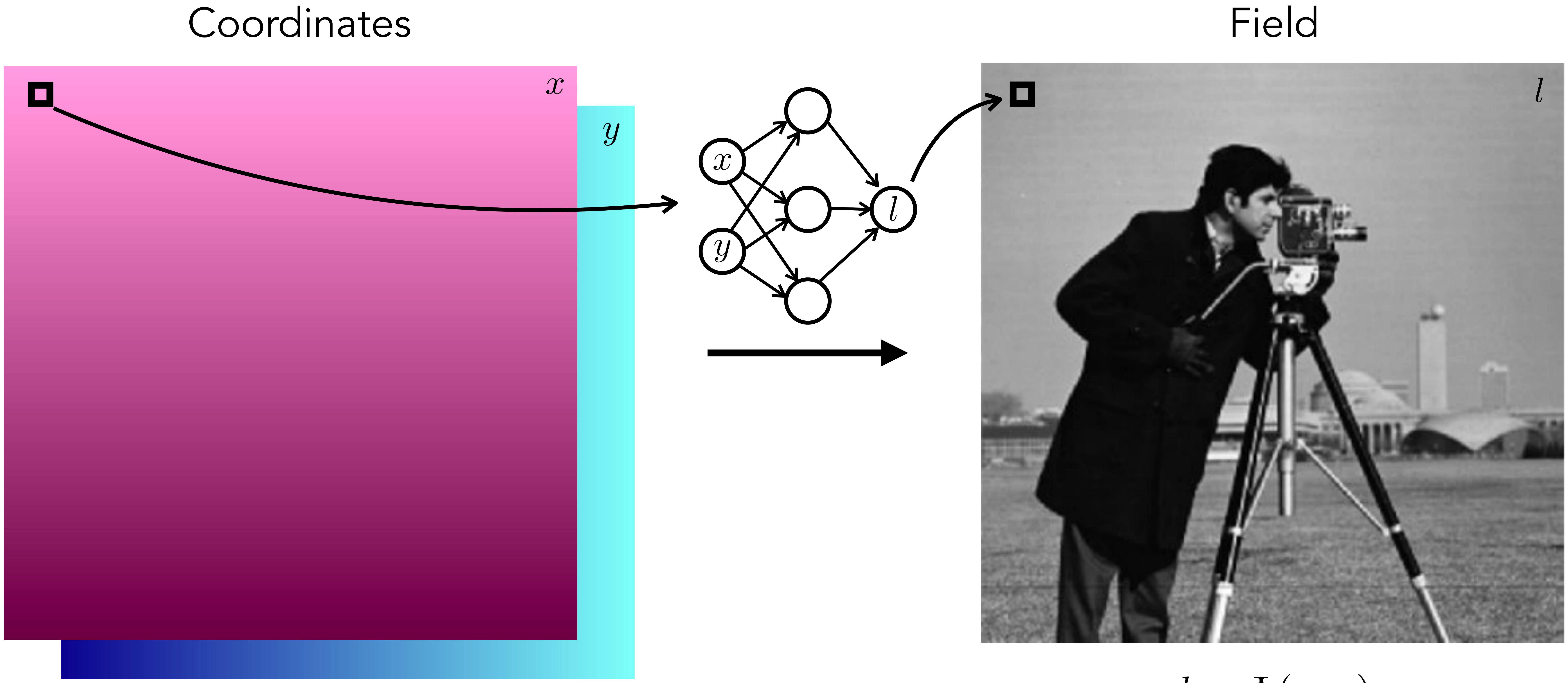
$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}$

Field

$$l = \Phi(x, y)$$

Neural Fields — SIREN

CNN applied *per-pixel* to map from a coordinate grid to a color



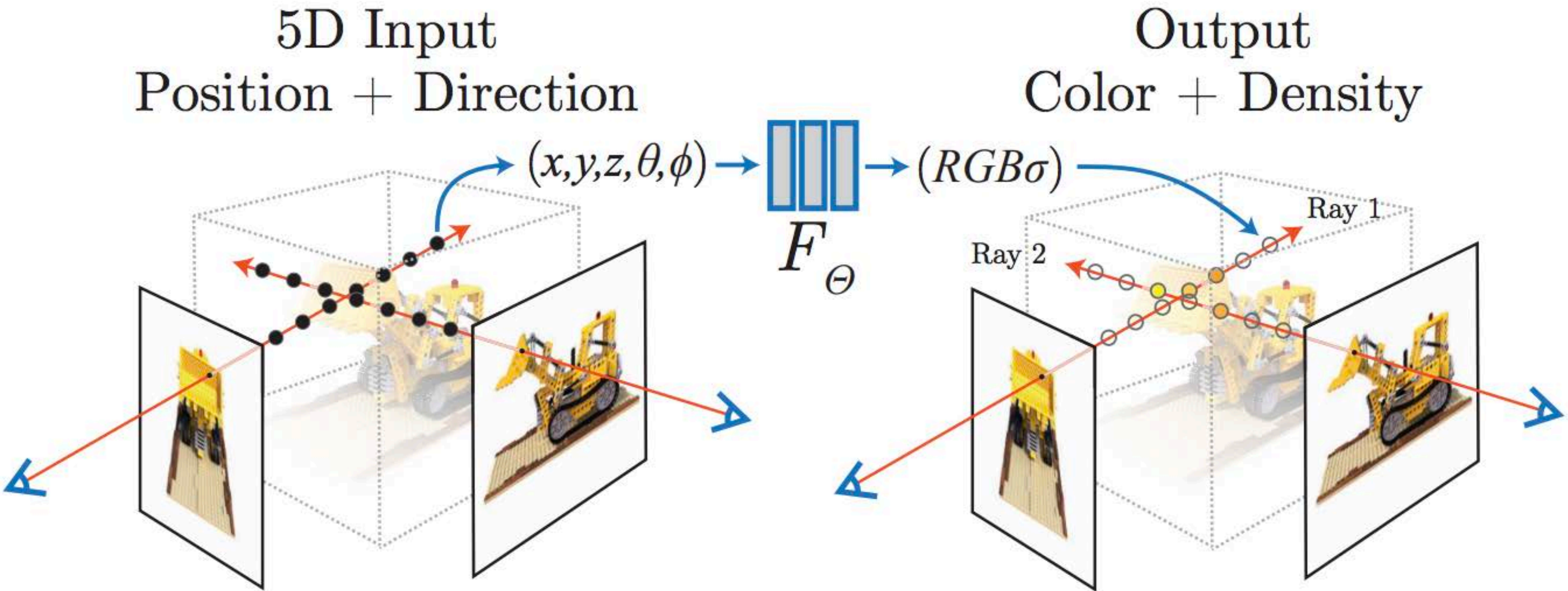
$$l = \Phi(x, y)$$

Can take continuous coordinates as input!

["SIREN", Sitzmann, Martel et al. 2020]

Neural Fields — NeRF

Conv net applied to map from 5-D coordinate grid to a color + volumetric density



["NeRF", Mildenhall, Srinivasan, Tancik et al., ECCV 2020]

made by Yen-Chen Lin

Concluding remarks

Convolution is a fundamental operation for image processing.

It just means: chop up the image into patches and apply the same function to each patch.

This concept appears in almost all modern architectures, such as CNNs, transformers, NeRFs, and more.

4. Architectures for Grids

- Why build better architectures?
- Convolutional layers
- Pyramids
- Architecture zoo
- Neural fields and positional encodings