

# Lecture 9: Convolutional networks

# Announcements

- PyTorch tutorial tonight!
  - 5:30 - 7:00 in EECS 1200
  - Totally optional! Recording, notes will be available.
- PS4 out. Similar in difficulty to PS3.
- PS1 grades and solution will be out next week
  - 1 week for regrade requests
- Reading: Deep Learning book by Goodfellow et al.

|  |
|--|
| EECS 504 PyTorch tutorial                  |
| Introduction                               |
| PyTorch                                    |
| Tensor Basics                              |
| Creating and Accessing tensors             |
| Tensor constructors                        |
| Datatypes                                  |
| Tensor indexing                            |
| Slice indexing                             |
| Integer tensor indexing                    |
| Boolean tensor indexing                    |
| Reshaping operations                       |
| View                                       |
| Swapping axes                              |
| Contiguous tensors                         |
| Your turn                                  |
| <b>Tensor operations</b>                   |
| Elementwise operations                     |
| Reduction operations                       |
| Matrix operations                          |
| Broadcasting                               |
| Running on GPU                             |
| Two-Layer Network by Using PyTorch Tensors |
| Autograd                                   |
| nn   |
| Optim                                      |

# Image classification

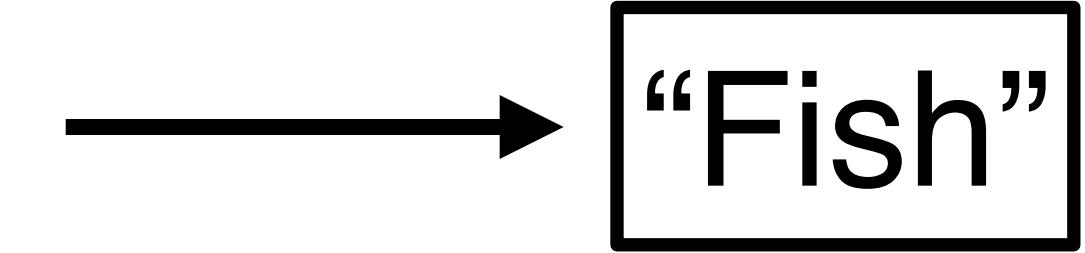
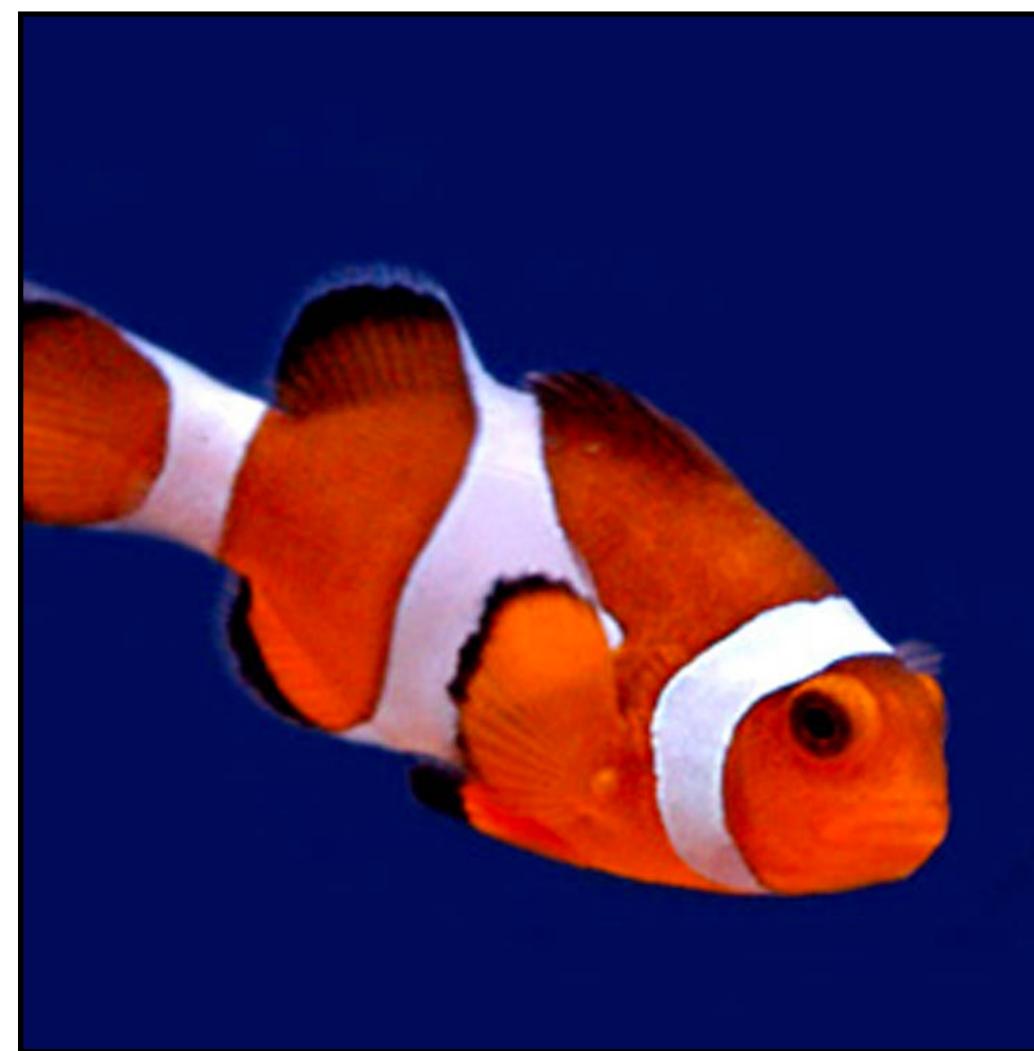


image  $x$

label  $y$

# Image classification

What should these be?

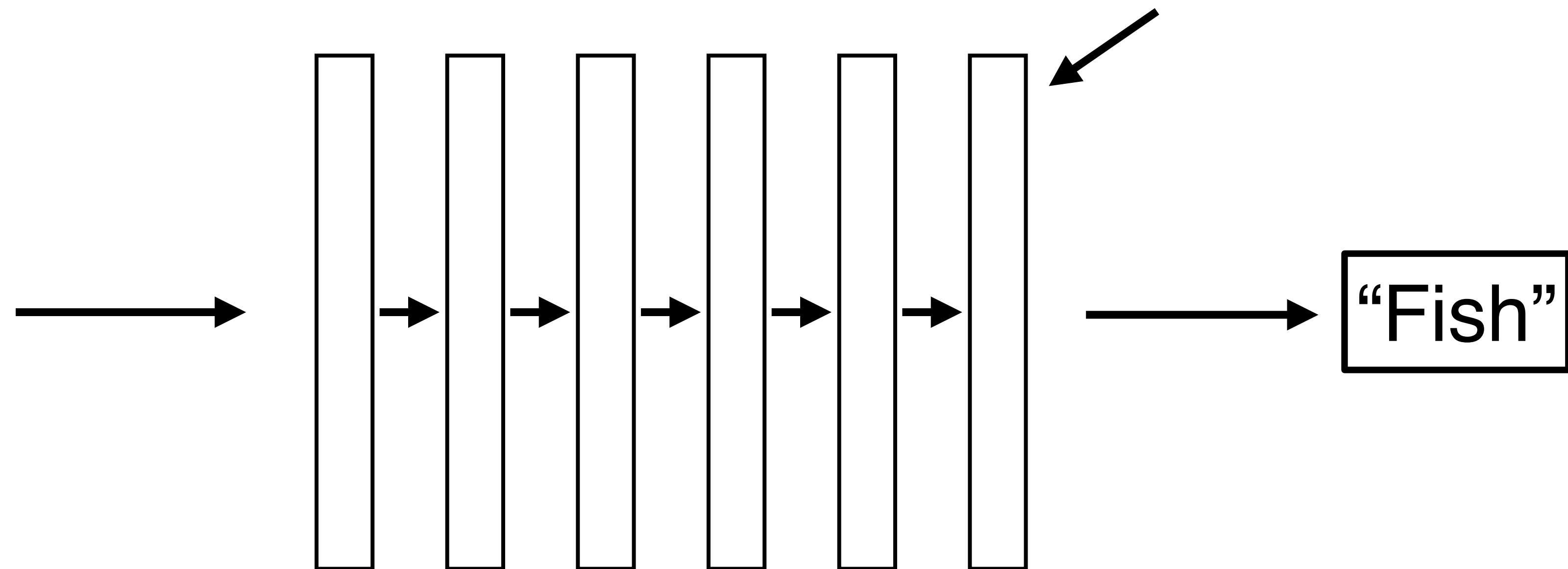
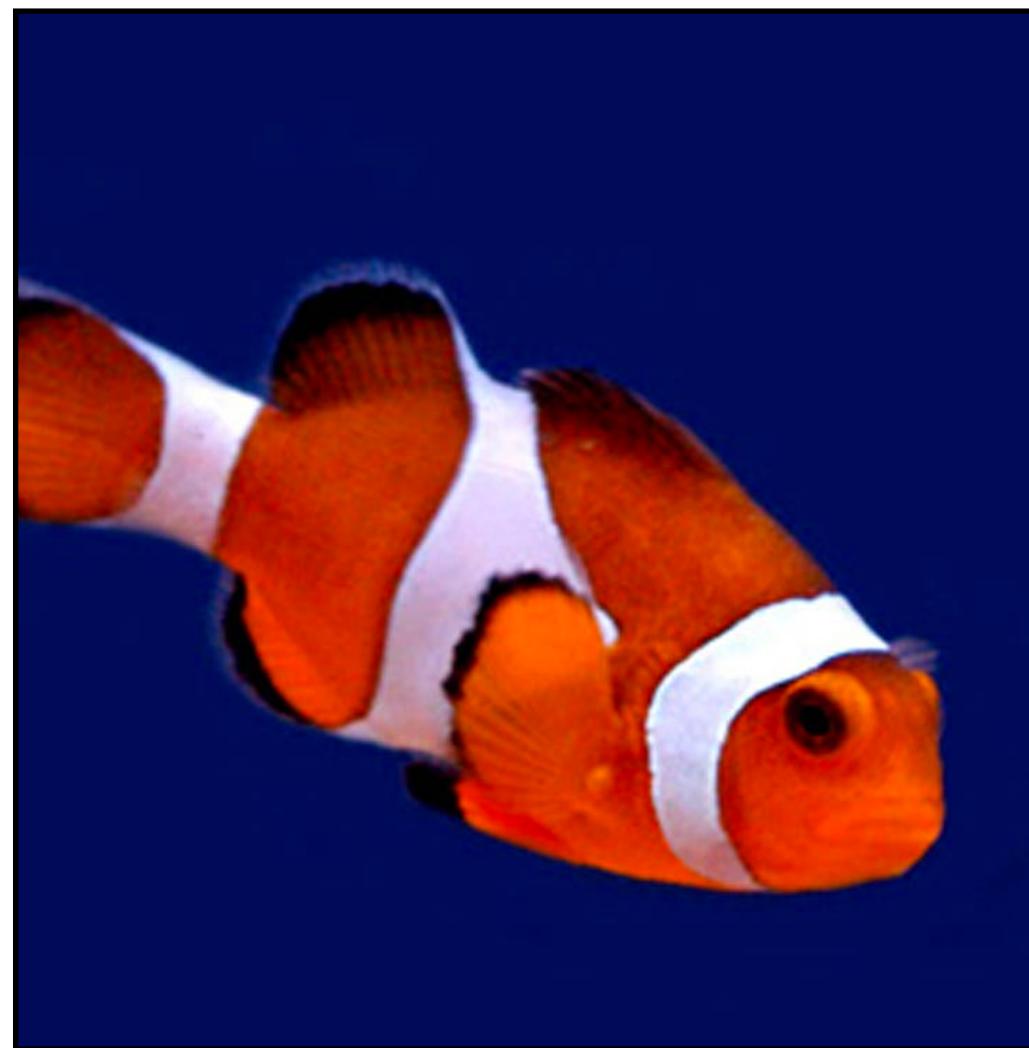
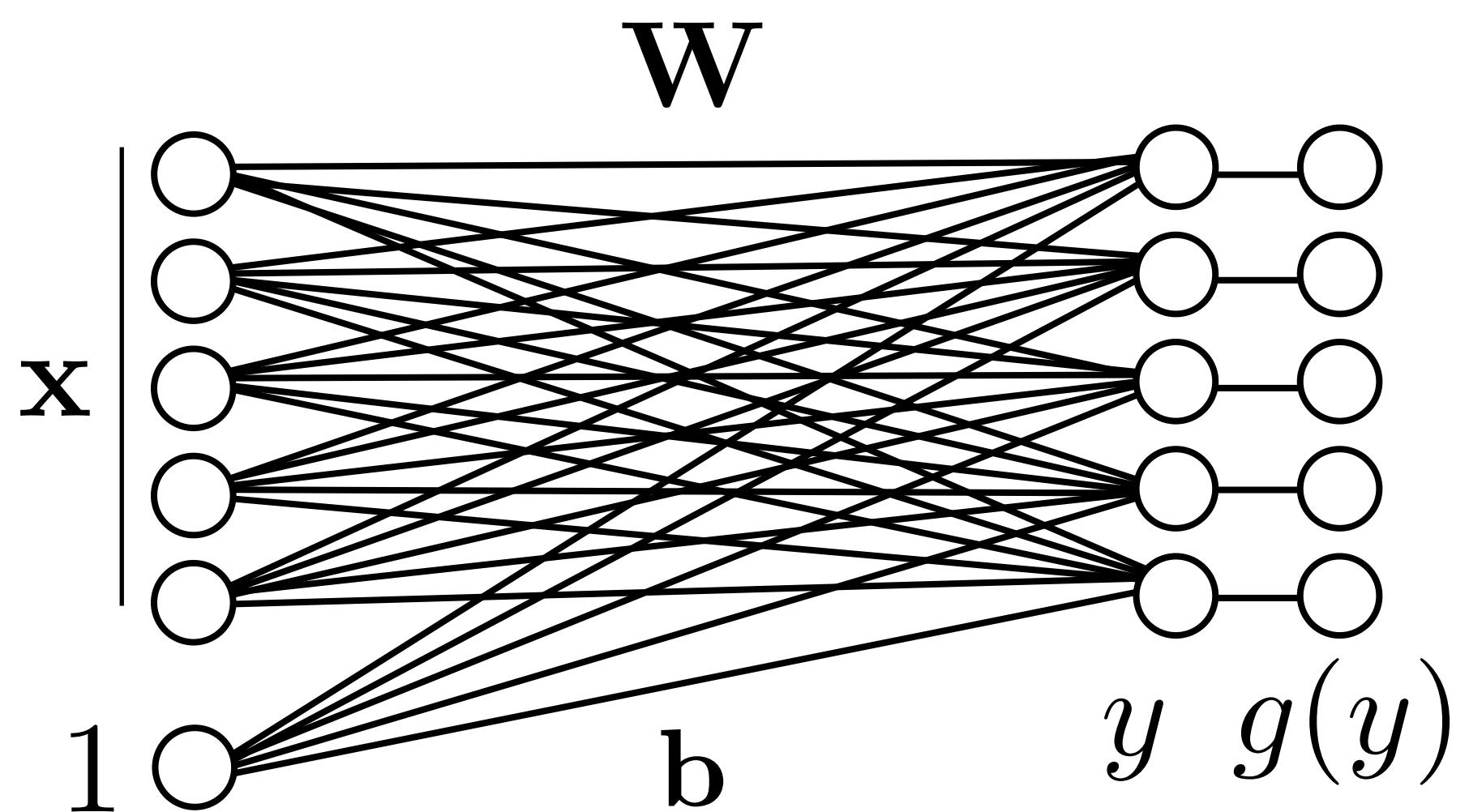


image x

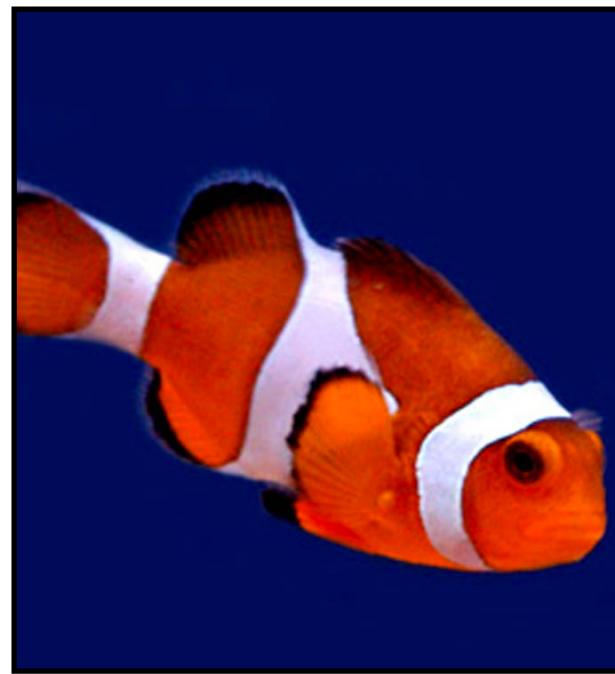
label y

# Fully-connected network

## Fully-connected (fc) layer

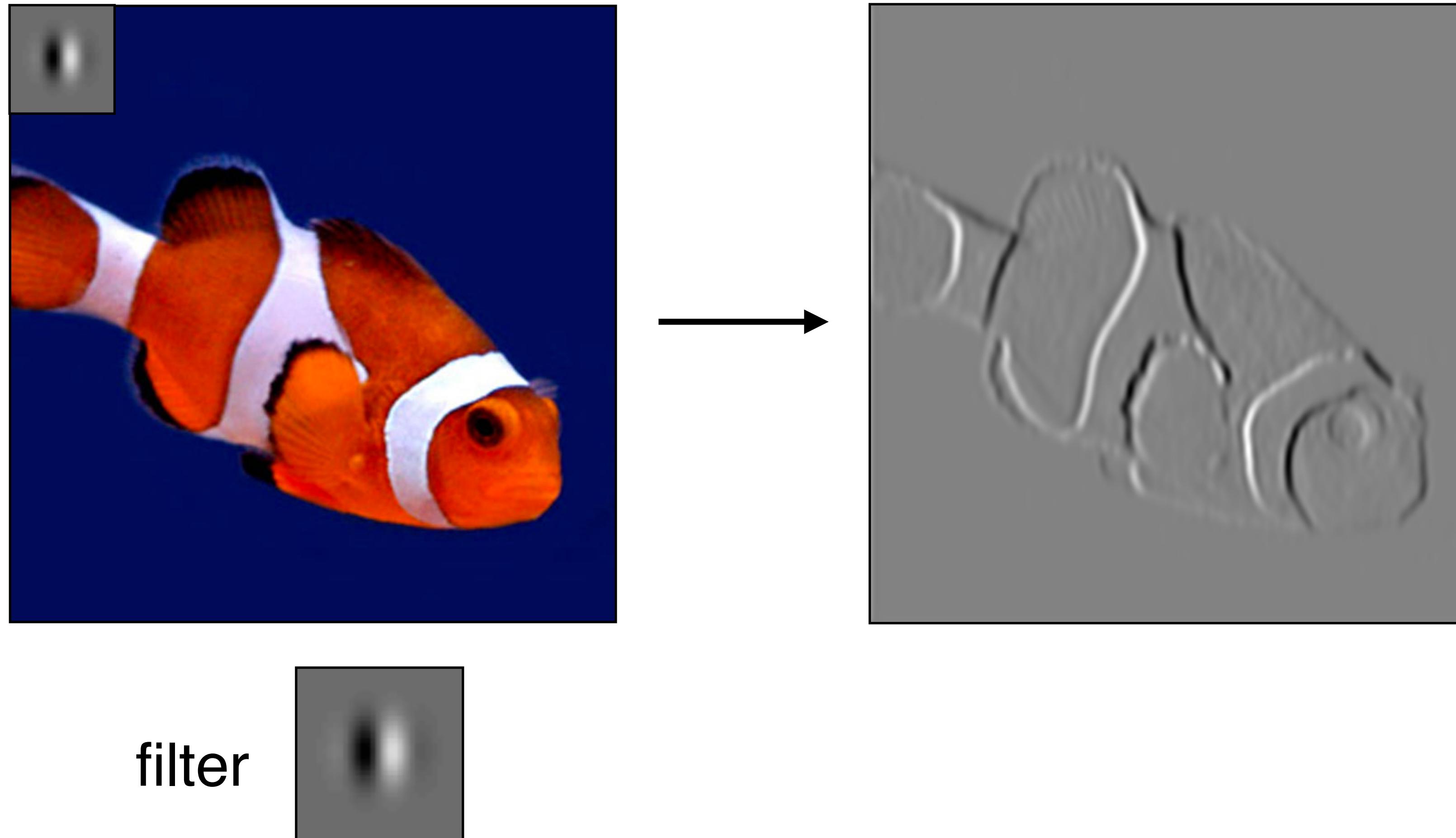


**x** =

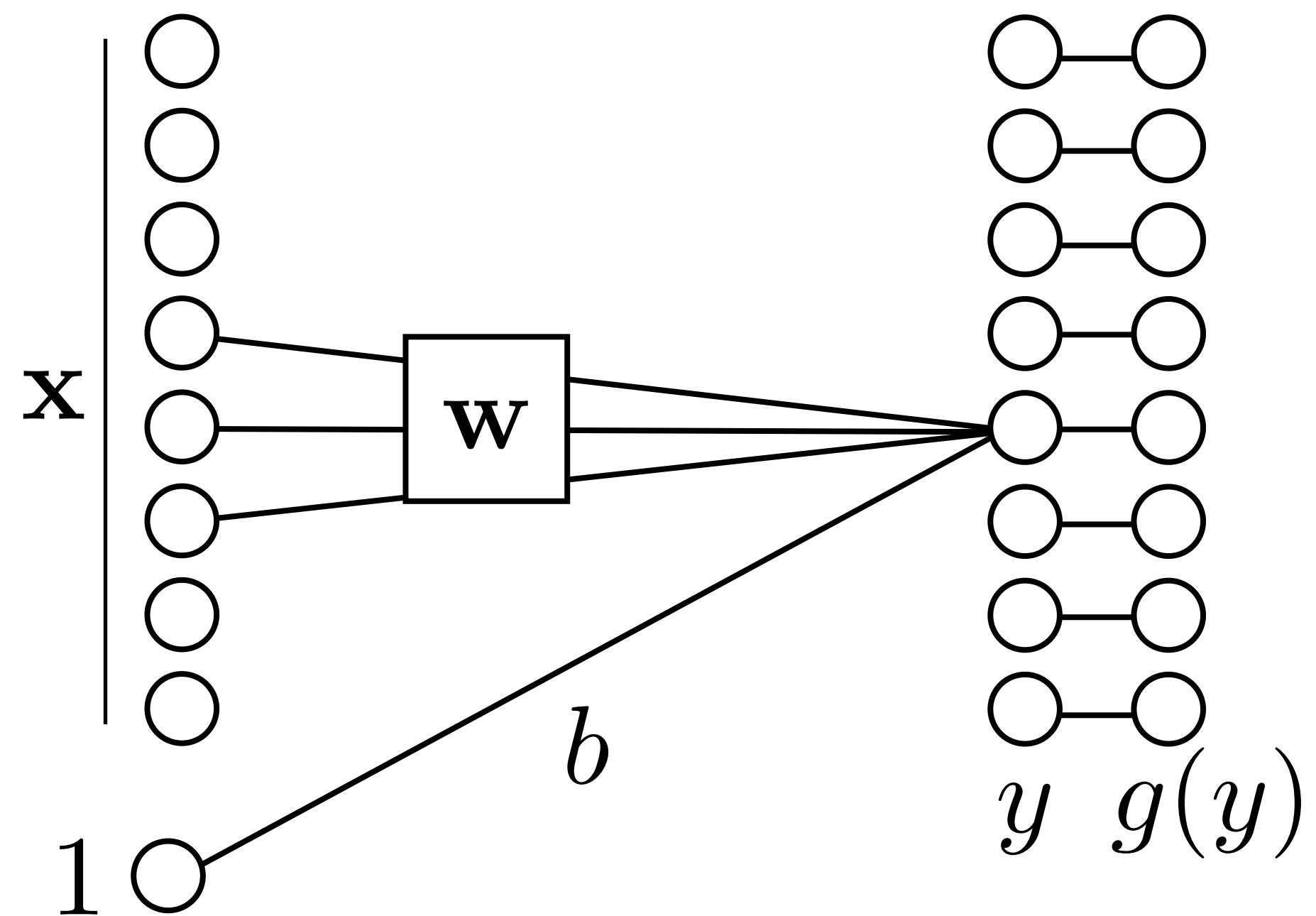


- **x** is really big! E.g.  $256 \times 256$
- If a feature is useful in one location, it should be useful in others, too

# Can we use convolution in a neural network?



# Locally connected network

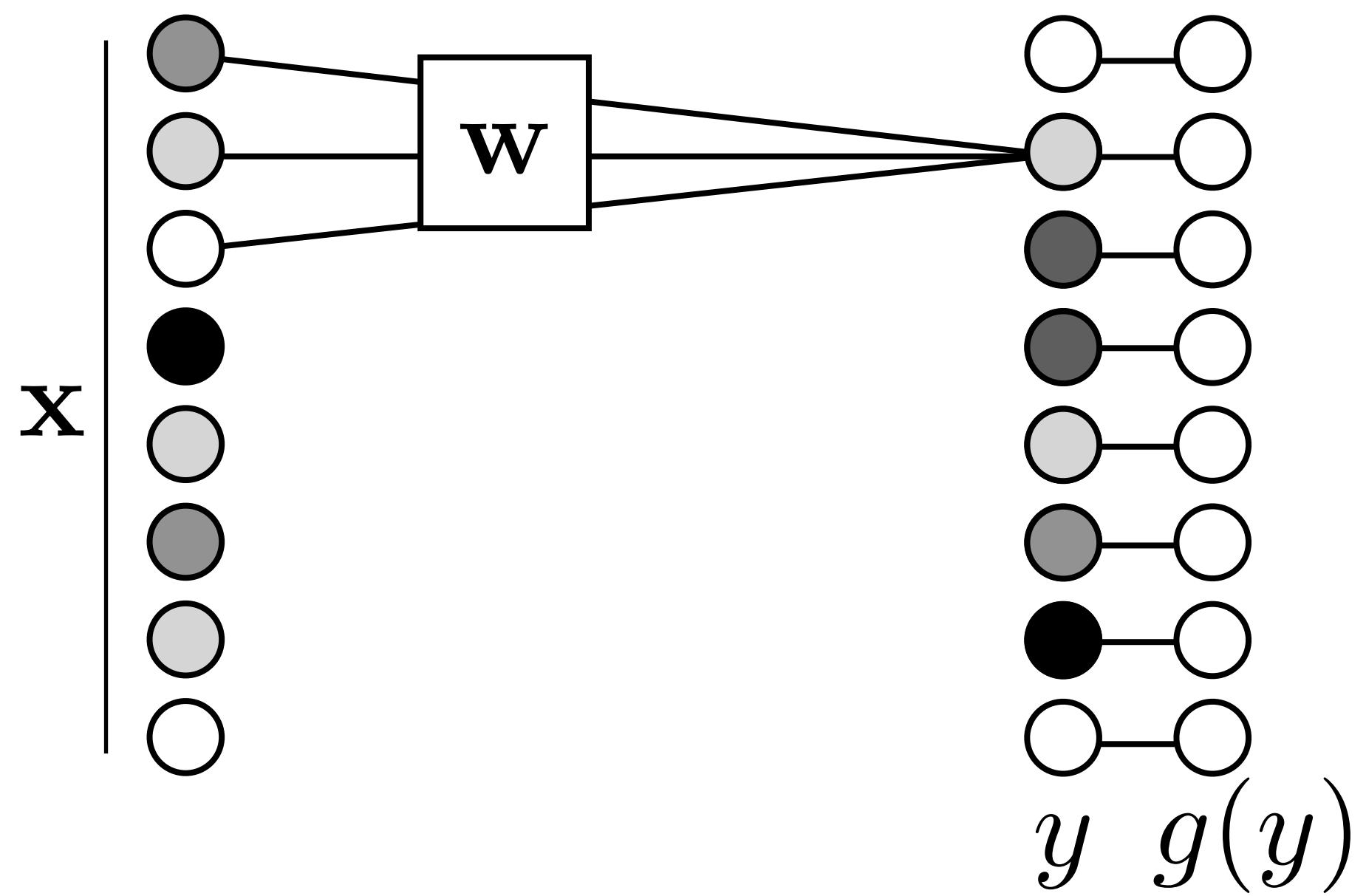


Often, we assume output is a **local** function of input.

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

# Convolutional neural network

## Conv layer

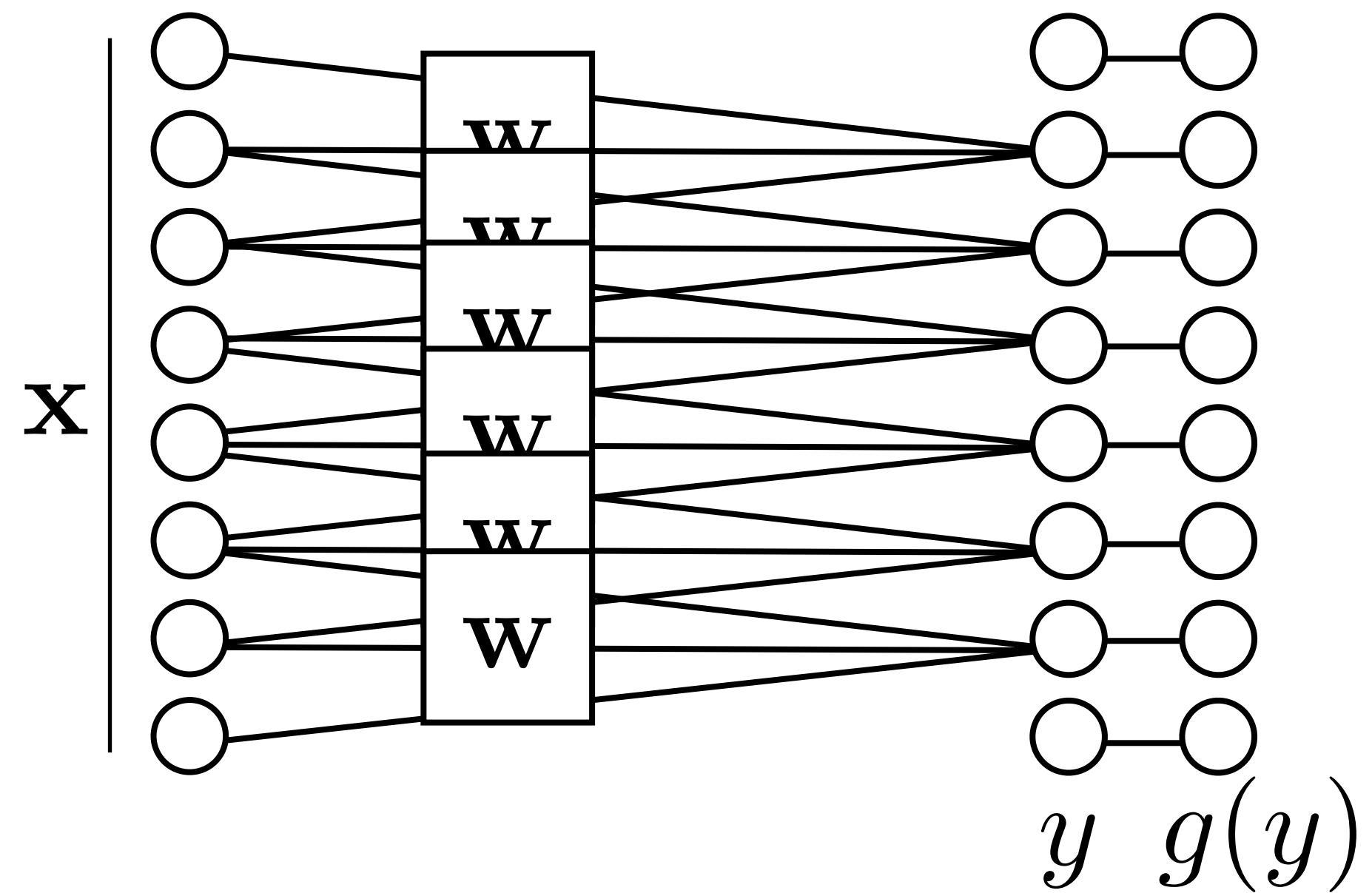


Each output unit is computed from an image patch.

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

# Weight sharing

## Conv layer



Each output unit is computed from an image patch.

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

# Recall: convolution is a linear function

**Toeplitz matrix**

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

= 

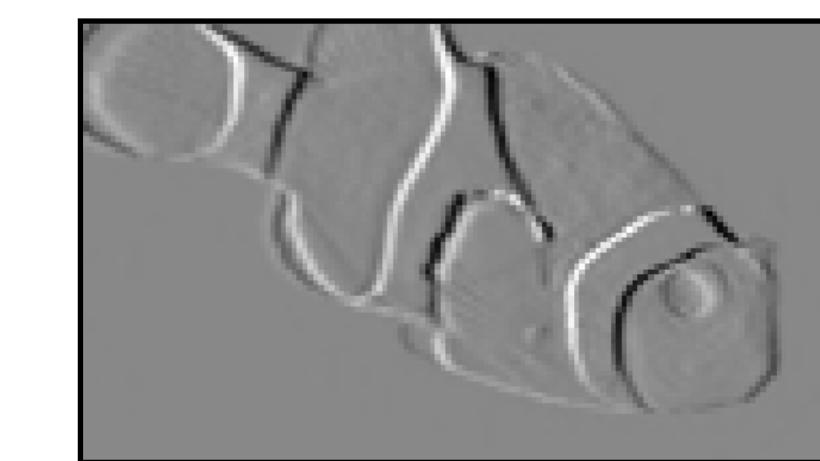
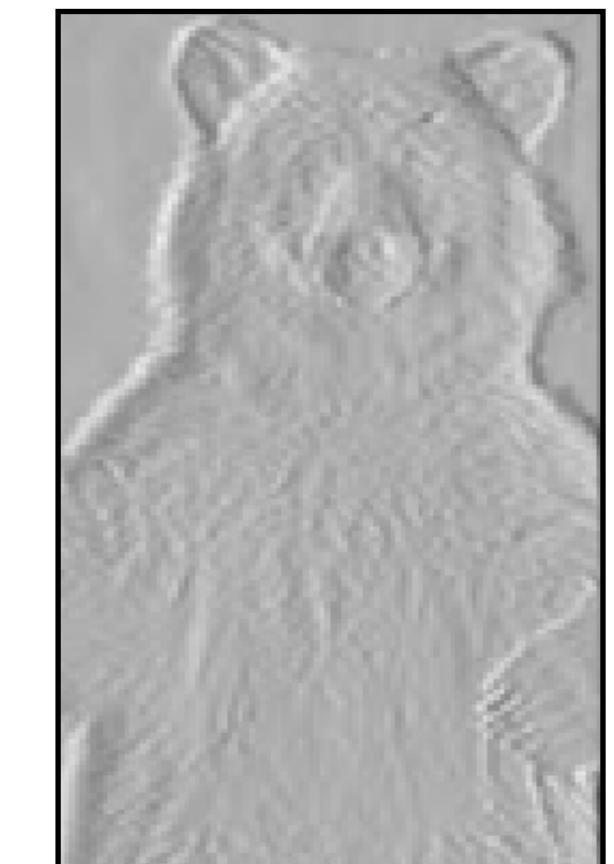
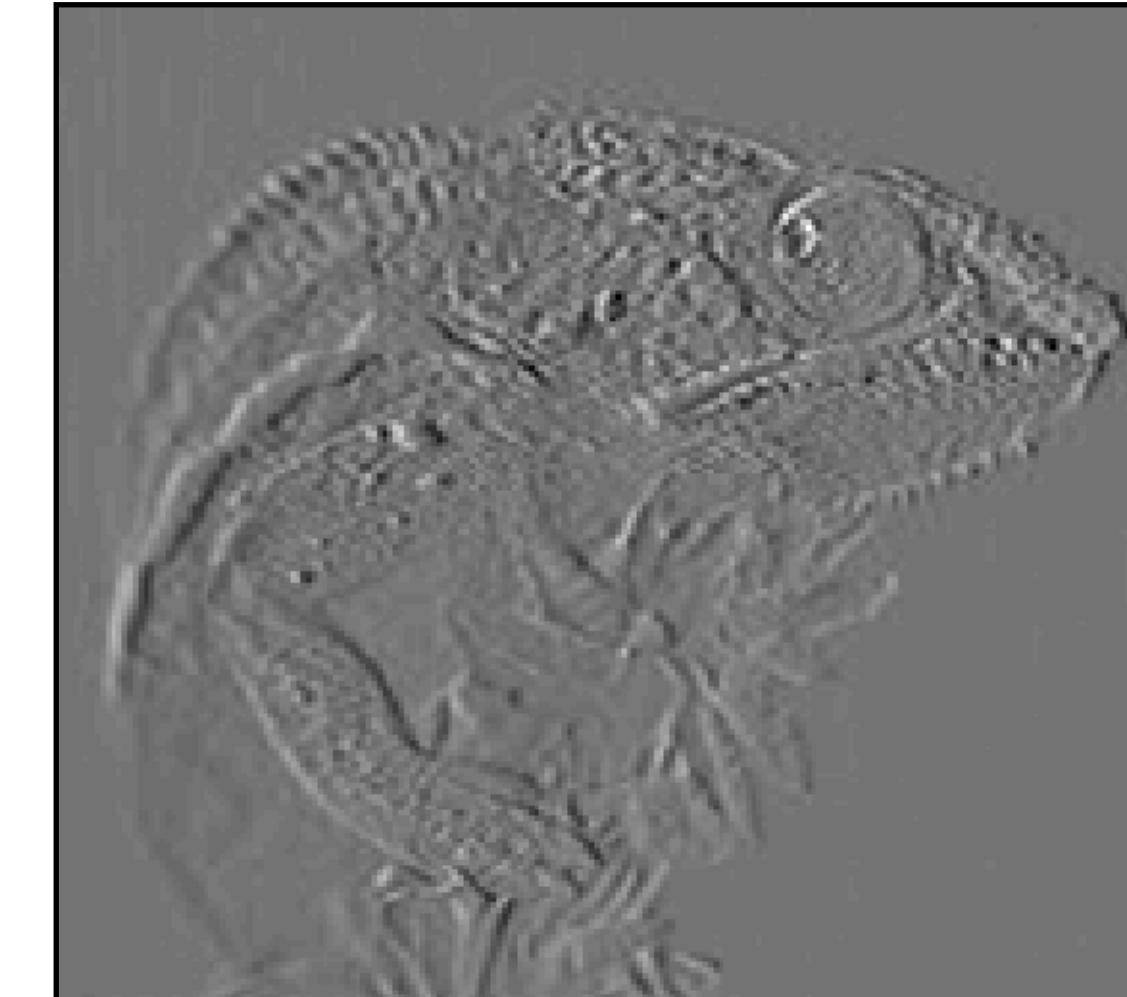
$\mathbf{x}^{(l+1)}$

$*$

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

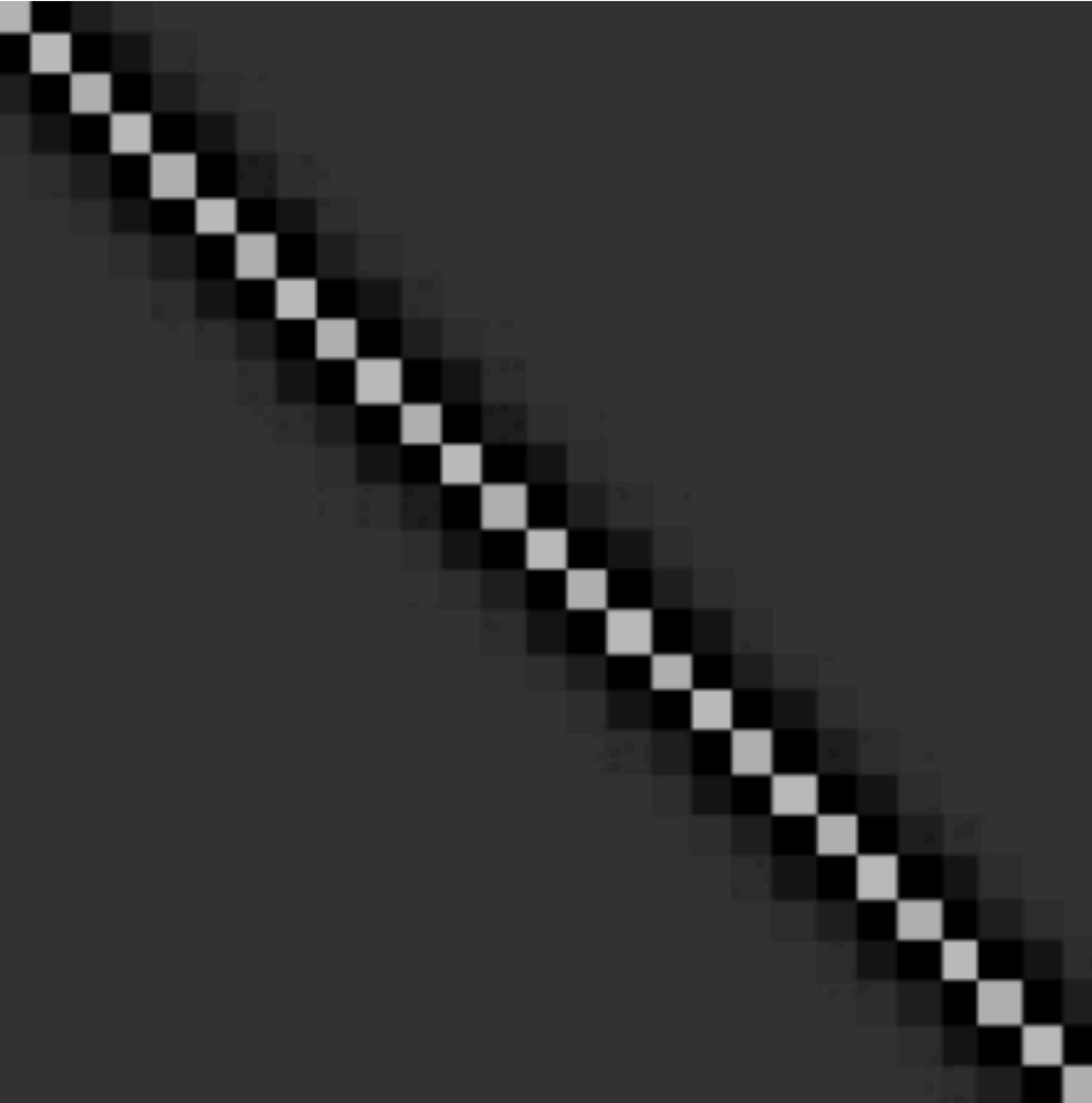
- Constrained linear layer
- Fewer parameters: easier to learn, less overfitting

e.g., pixel image



Convolution layers can be applied to arbitrarily-sized inputs

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

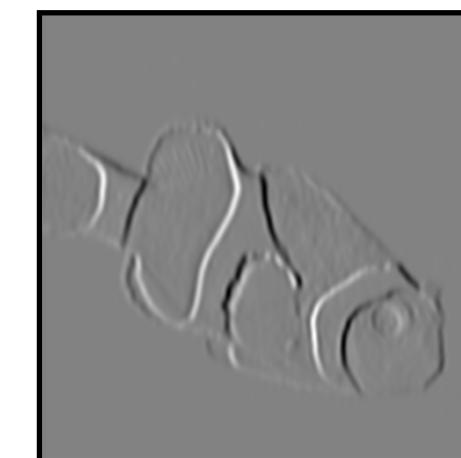
$\mathbf{x}^{(l+1)}$  $=$  $\mathbf{x}^{(l)}$

# Different interpretations of convolutional layers

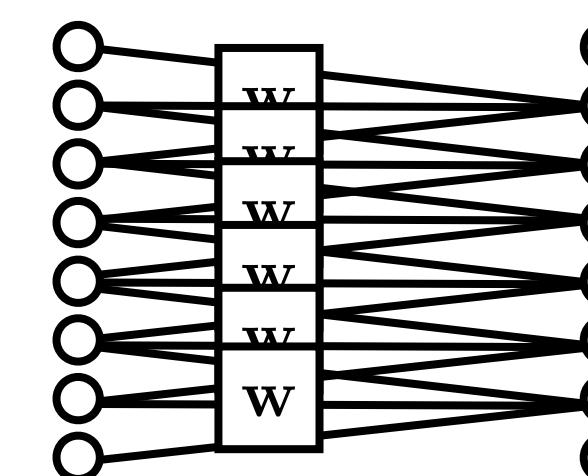
1. Equivariant with translation:  $f(\text{translate}(x)) = \text{translate}(f(x))$

2. Patch processing (Markov assumption)

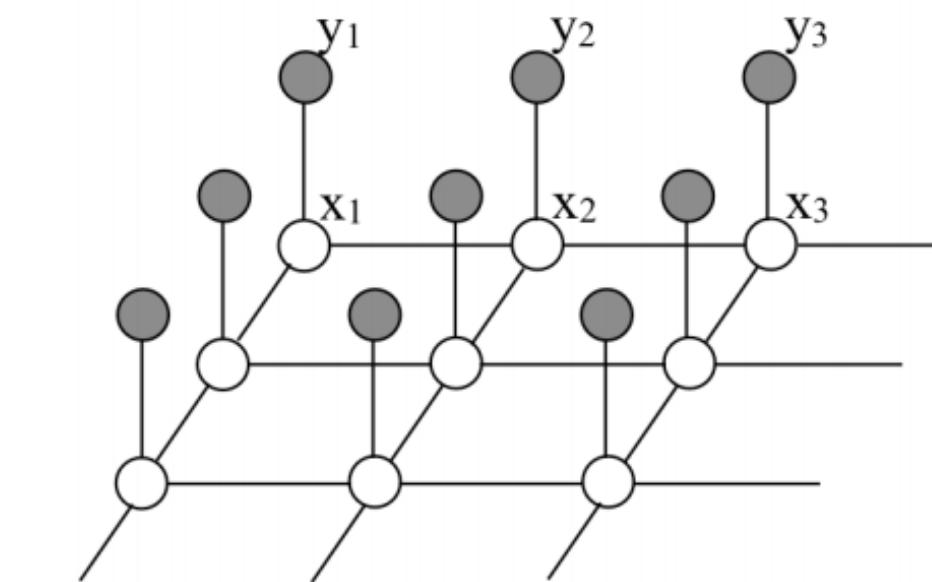
3. Image filter



4. Parameter sharing



5. A way to process variable-sized tensors

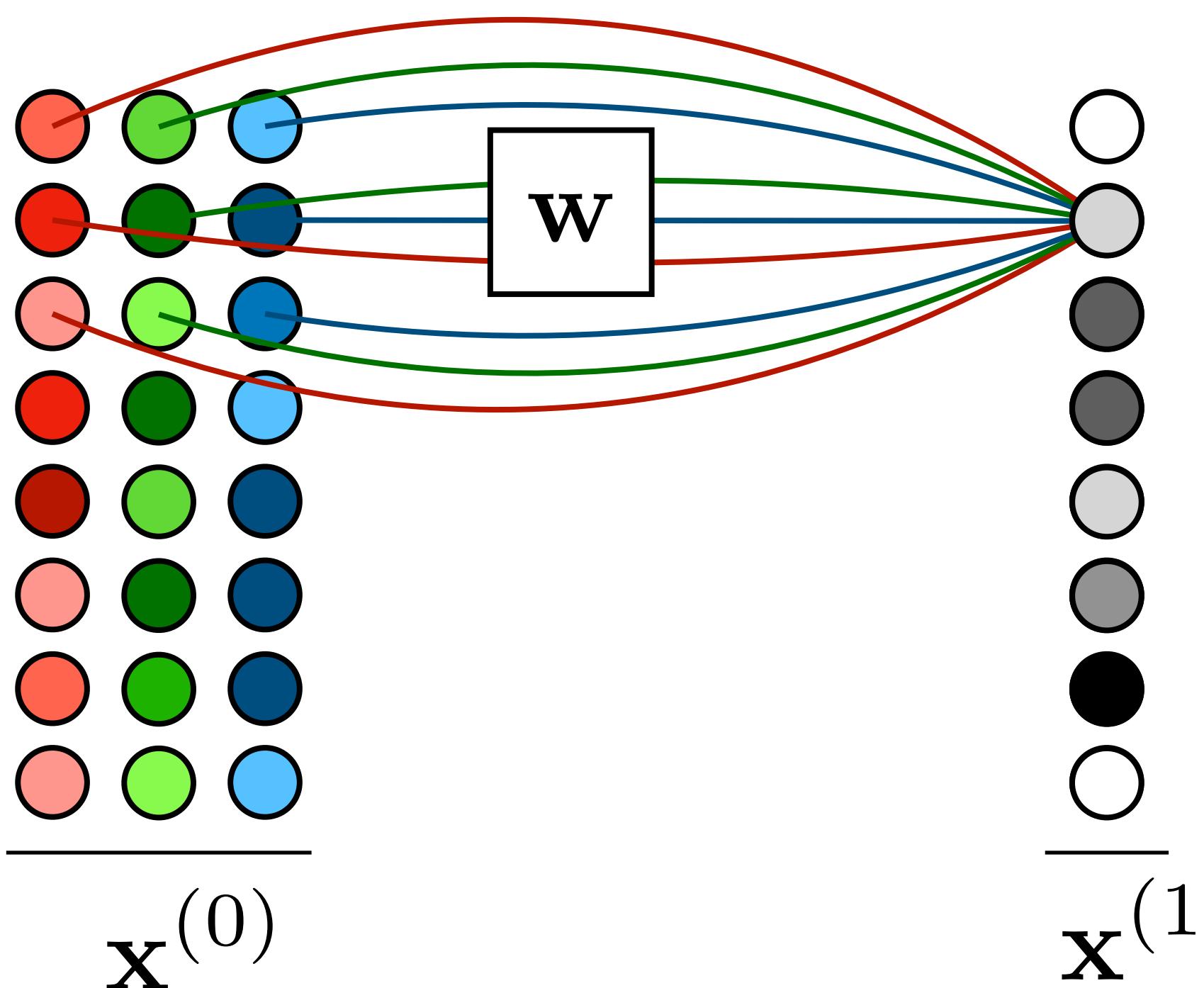


# What if we have color?

(And multiple input channels in general)

# Multiple channels

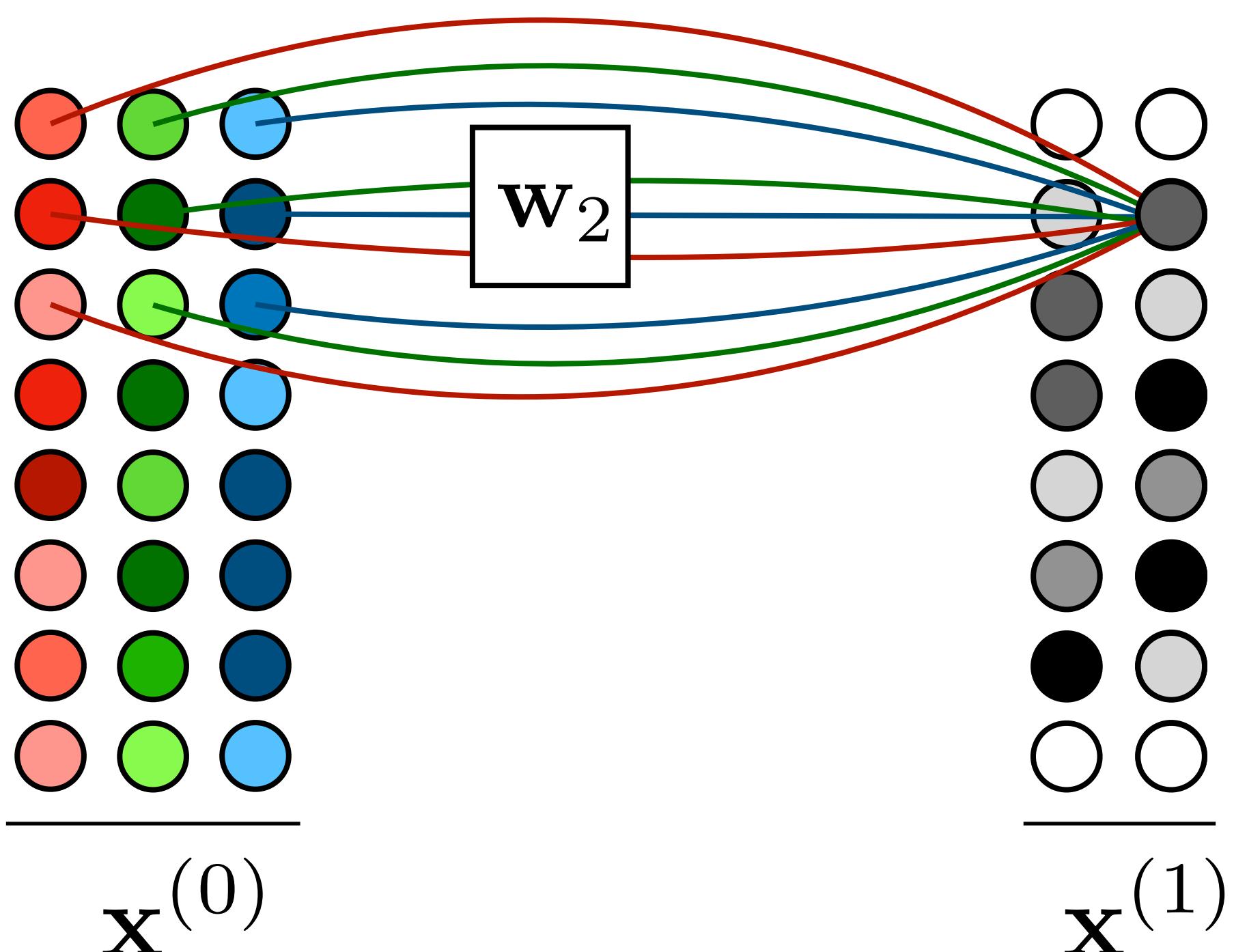
## Conv layer



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

# Multiple channels

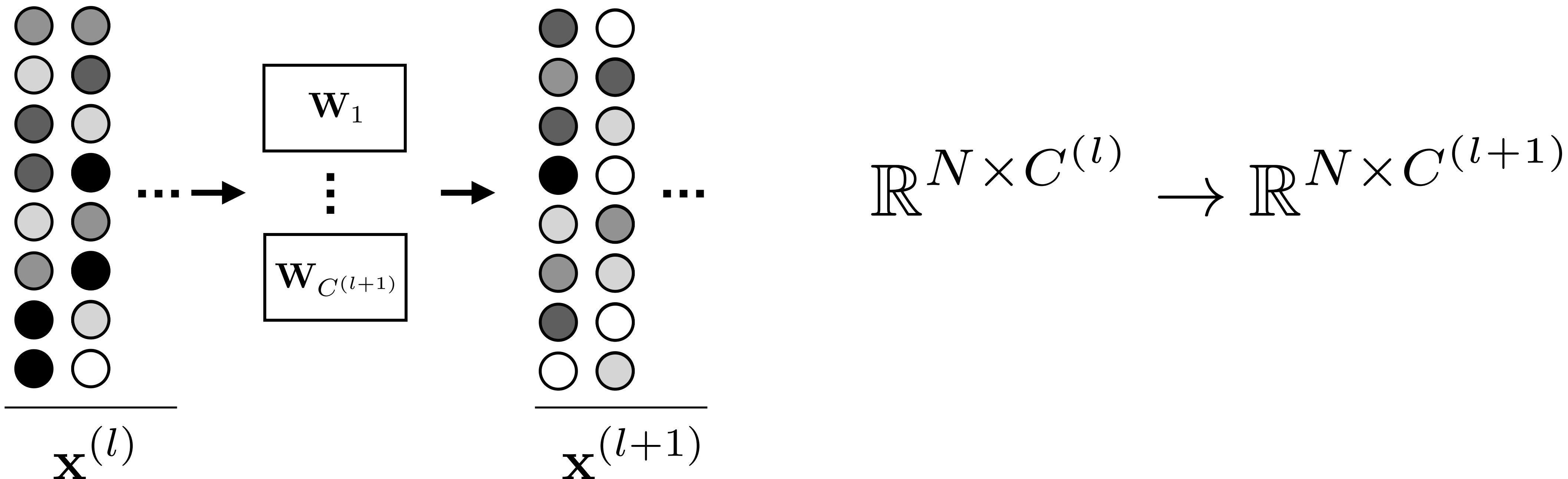
## Conv layer



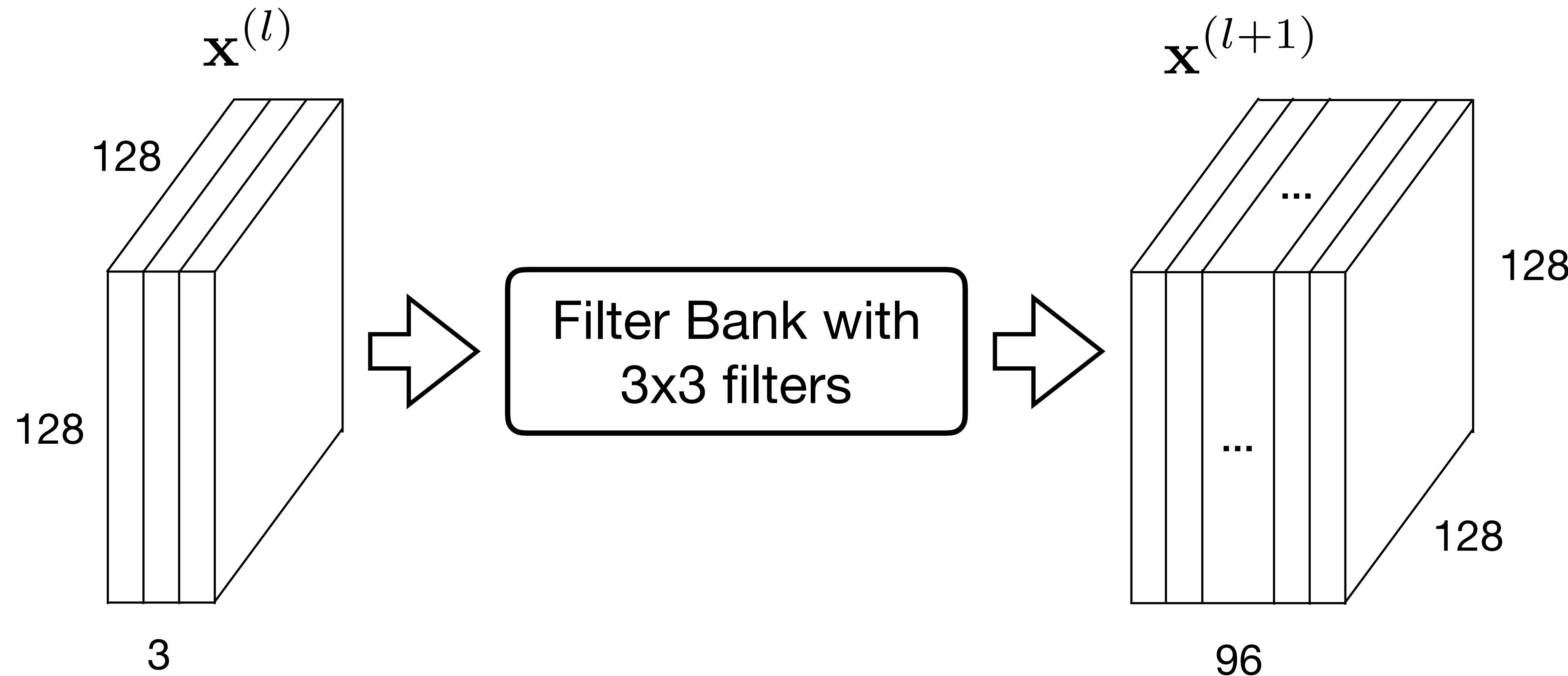
$$\mathbb{R}^{N \times C^{(0)}} \rightarrow \mathbb{R}^{N \times C^{(1)}}$$

# Multiple channels

## Conv layer



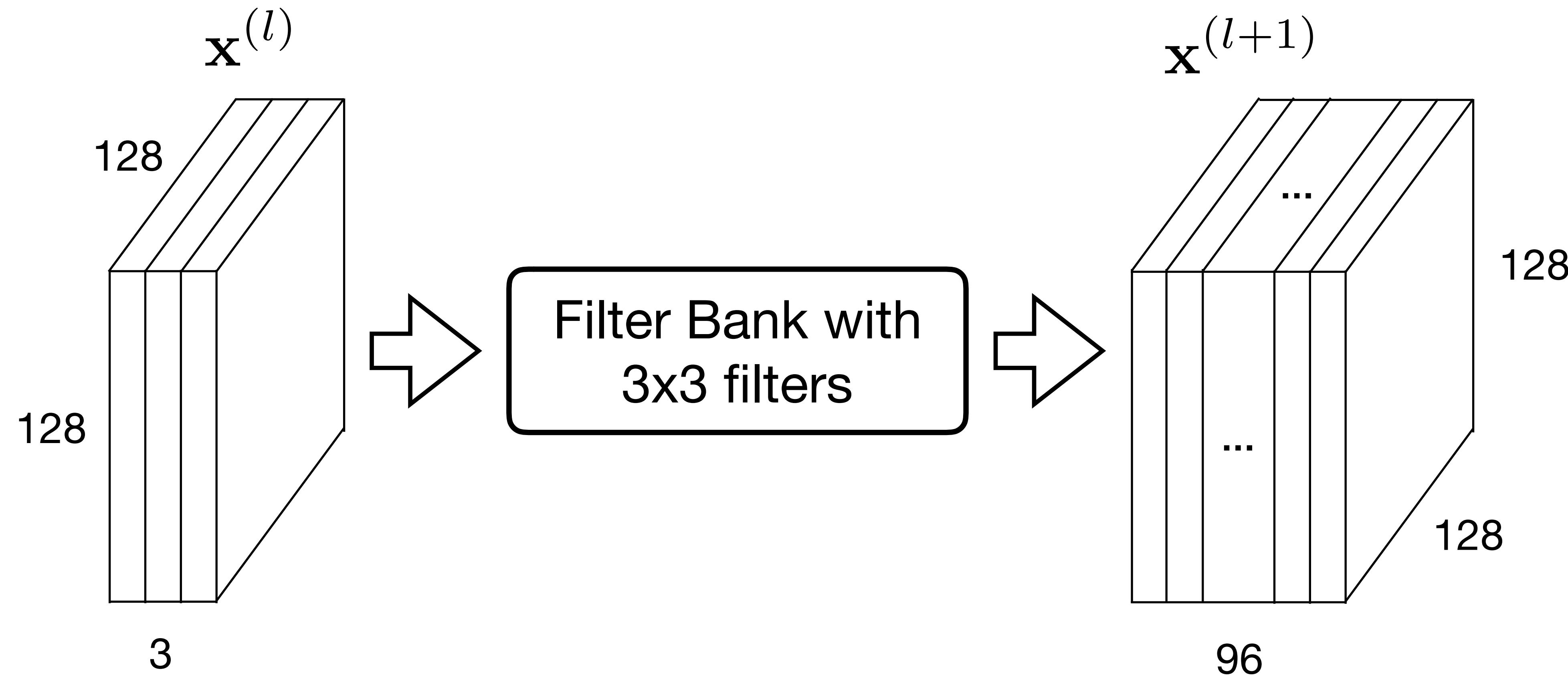
# Multiple channels: Example



How many parameters does each *filter* have?

- (a) 9
- (b) 27
- (c) 96
- (d) 2592

# Multiple channels: Example



How many parameters *total* does this layer have?

- (a) 9
- (b) 27
- (c) 96
- (d) 2592

# Filter sizes

When mapping from

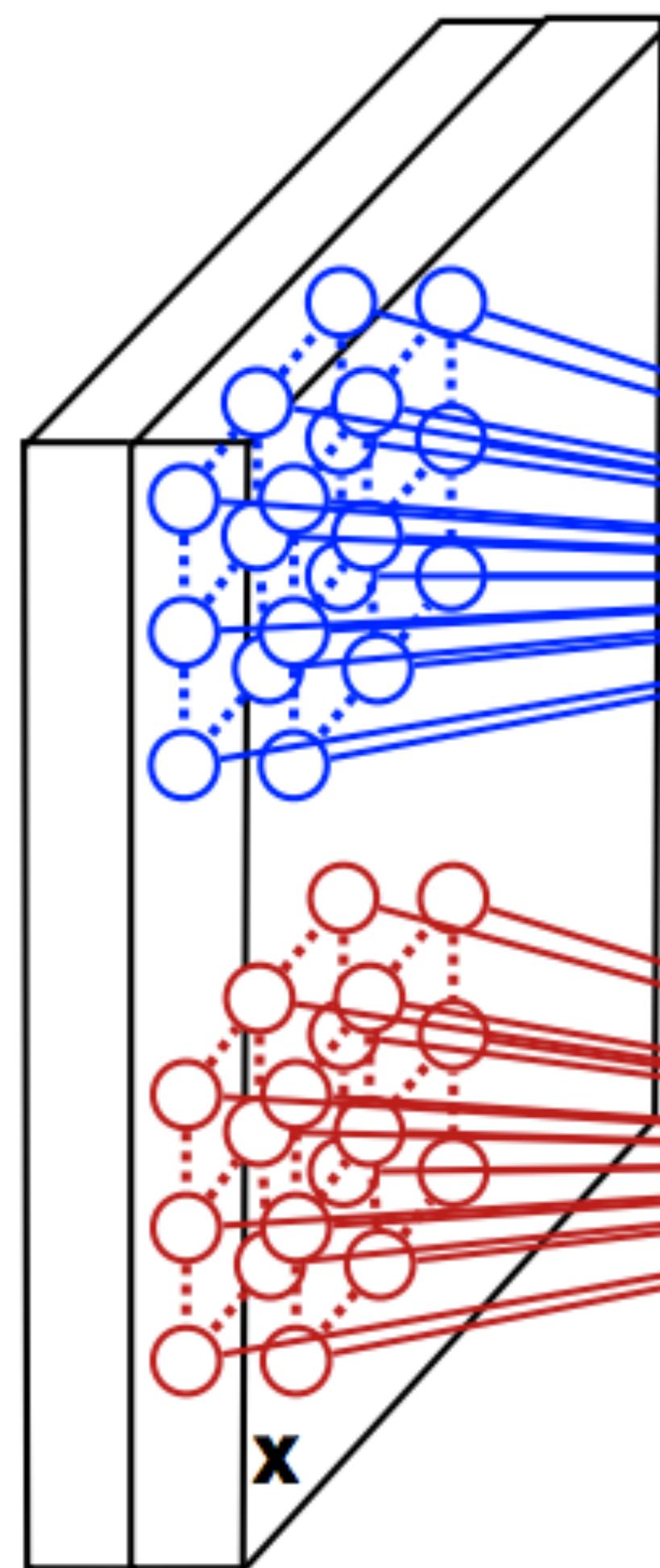
$$\mathbf{x}^{(l)} \in \mathbb{R}^{H \times W \times C^{(l)}} \rightarrow \mathbf{x}^{(l+1)} \in \mathbb{R}^{H \times W \times C^{(l+1)}}$$

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

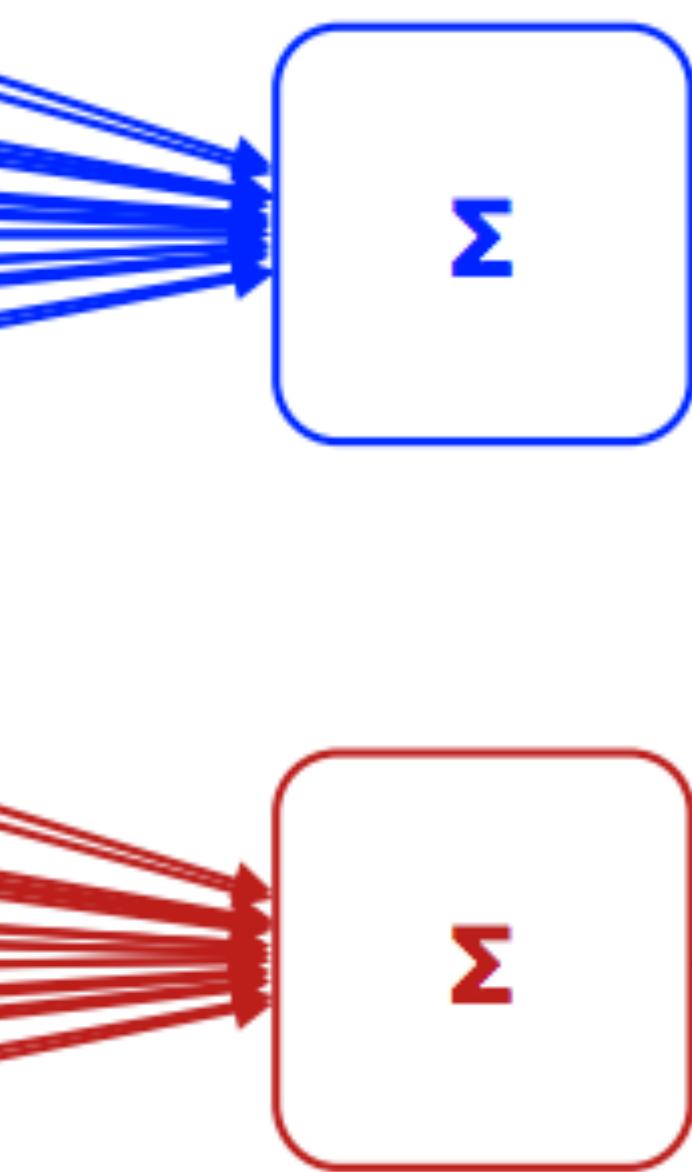
Number of parameters per filter:  $M \times N \times C^{(l)}$

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

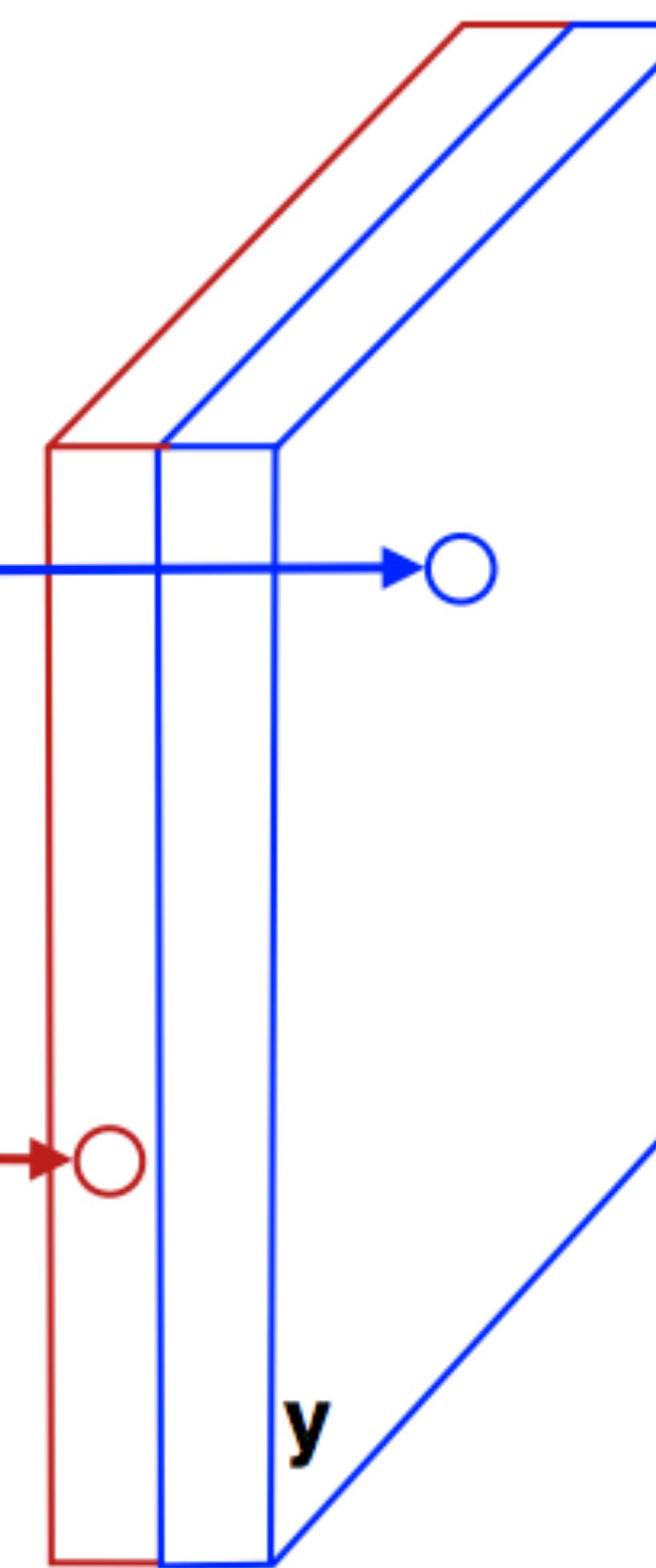
Input features



A bank of 2 filters

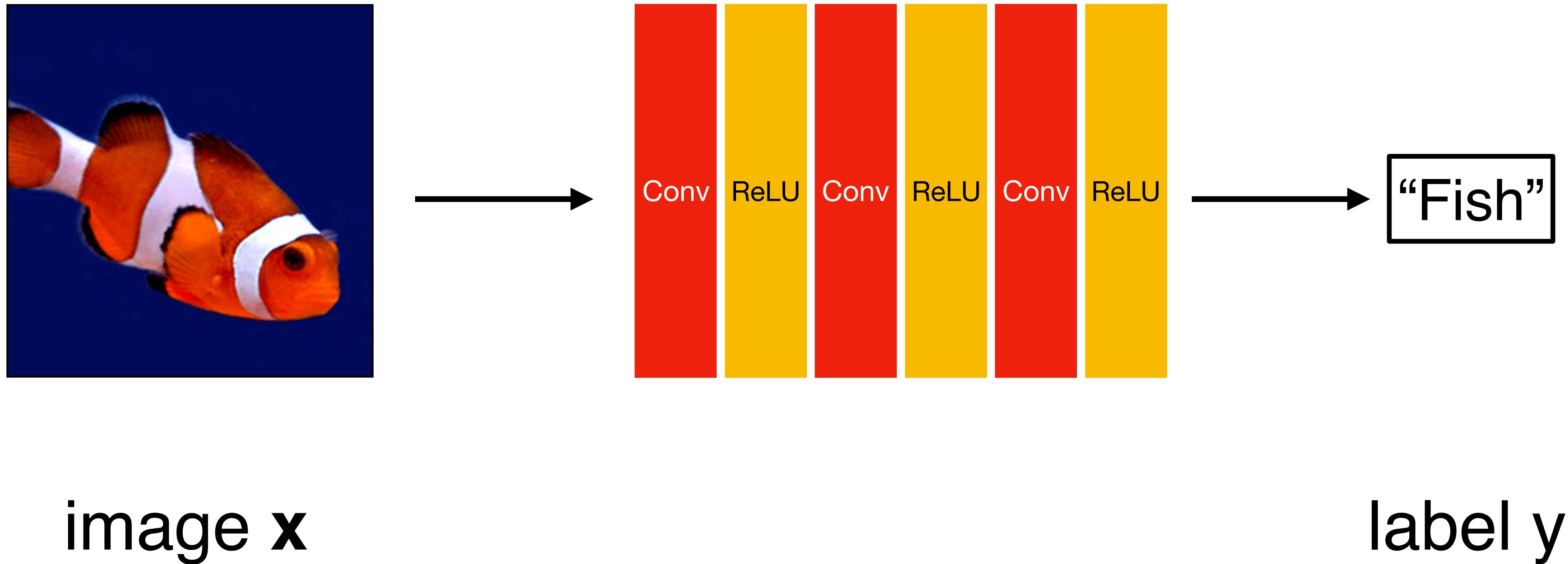


2-dimensional output features

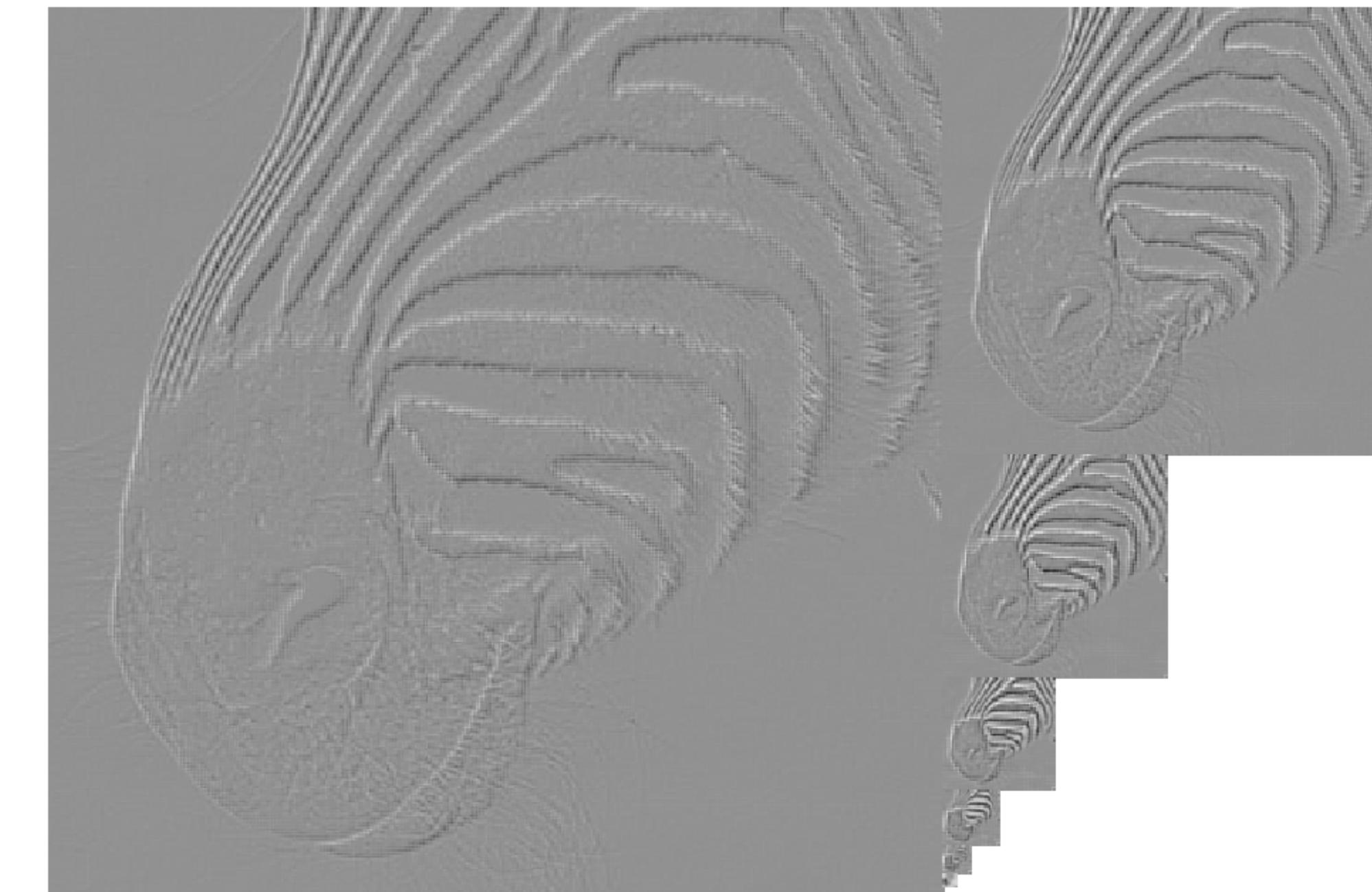
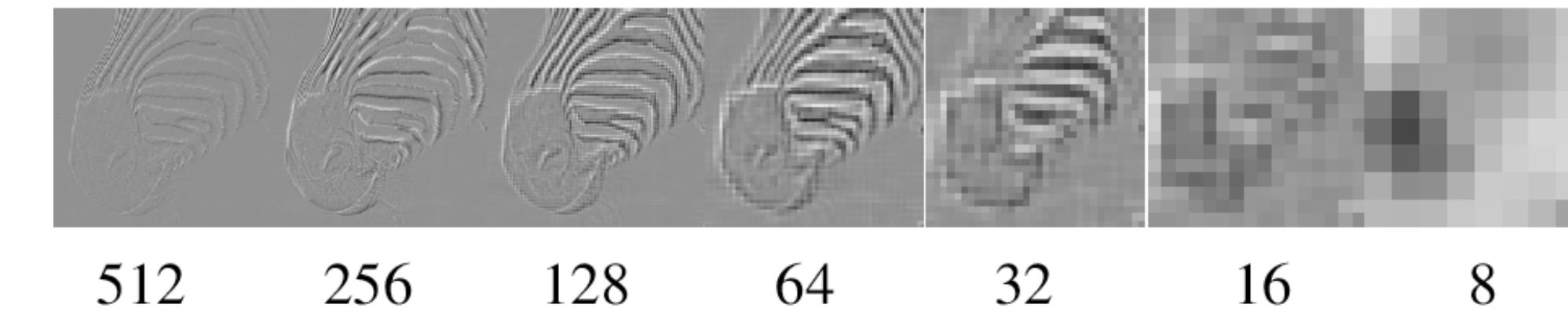
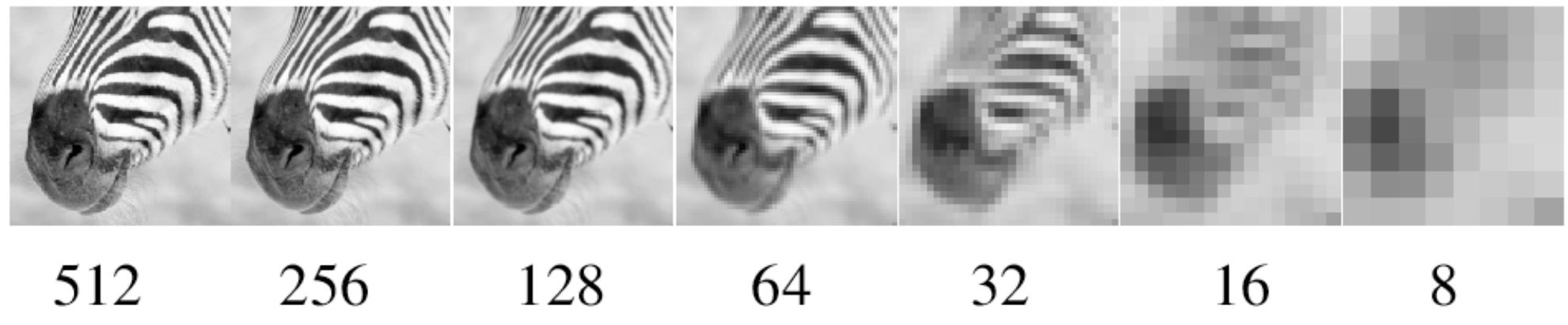


$$\mathbb{R}^{H \times W \times C^{(l)}} \rightarrow \mathbb{R}^{H \times W \times C^{(l+1)}}$$

# Image classification



# Recall: pyramid representations



Gaussian Pyramid

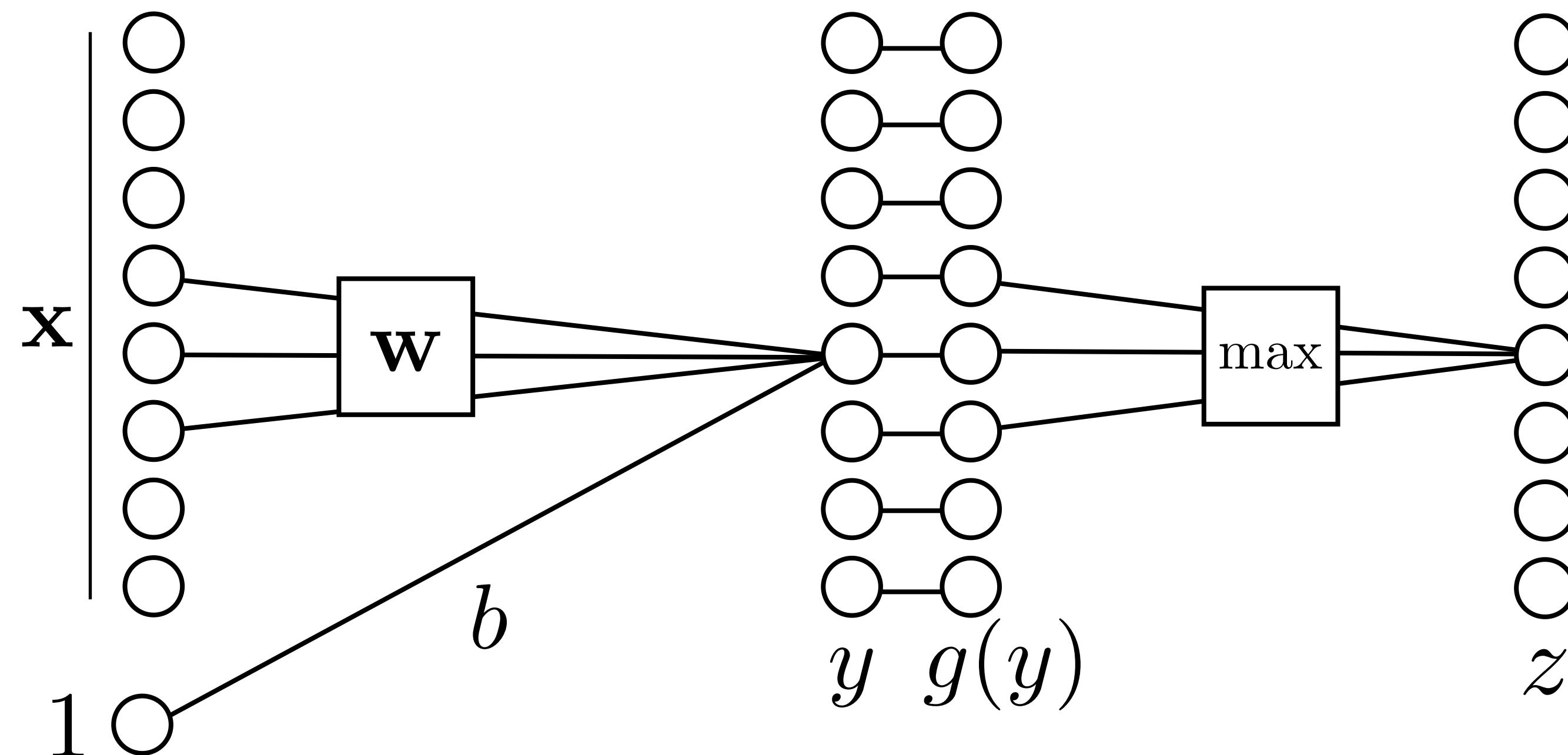
Laplacian Pyramid

**How can we use multi-scale modeling in CNNs?**

# Pooling

*Filter*

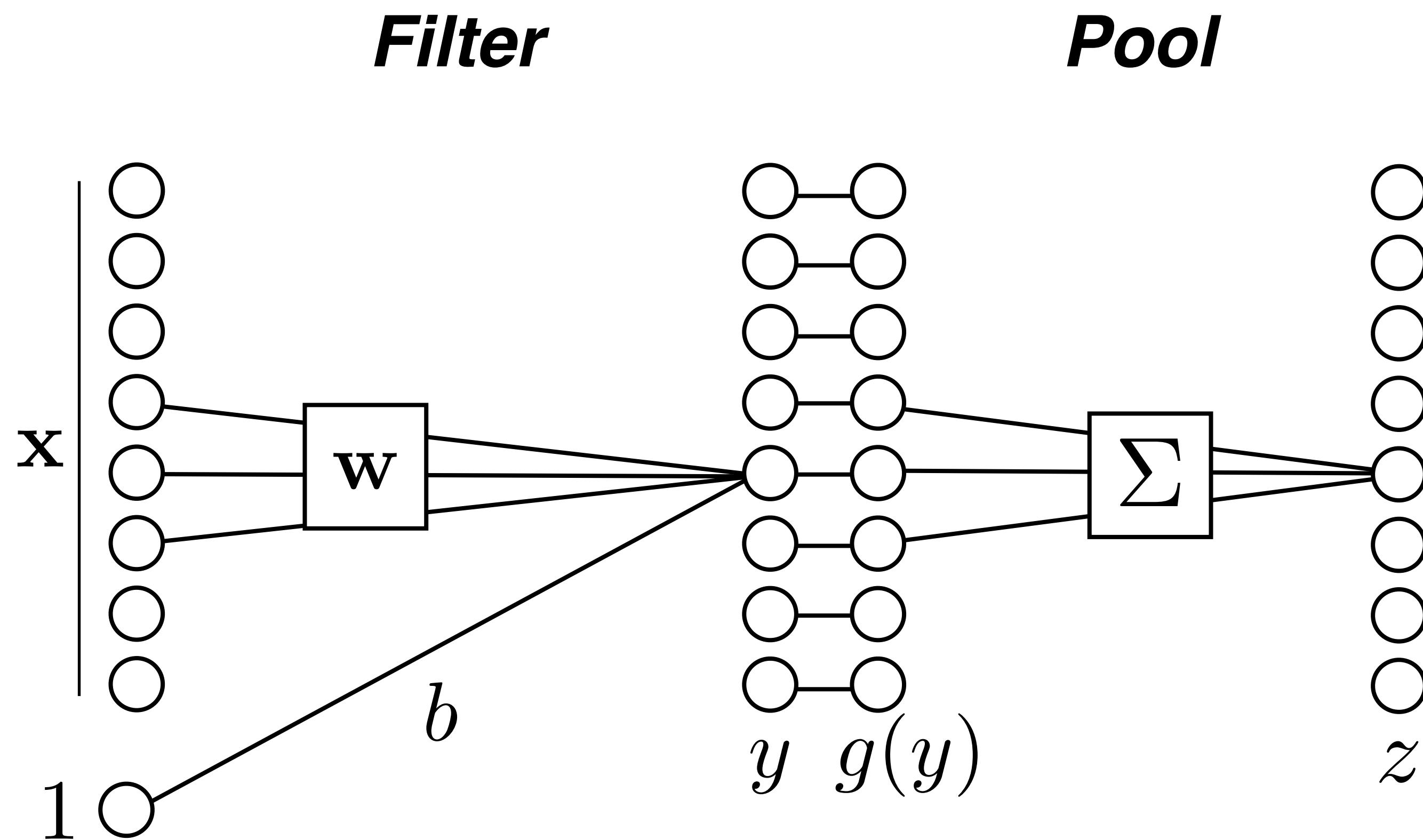
*Pool*



**Max pooling**

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

# Pooling



**Max pooling**

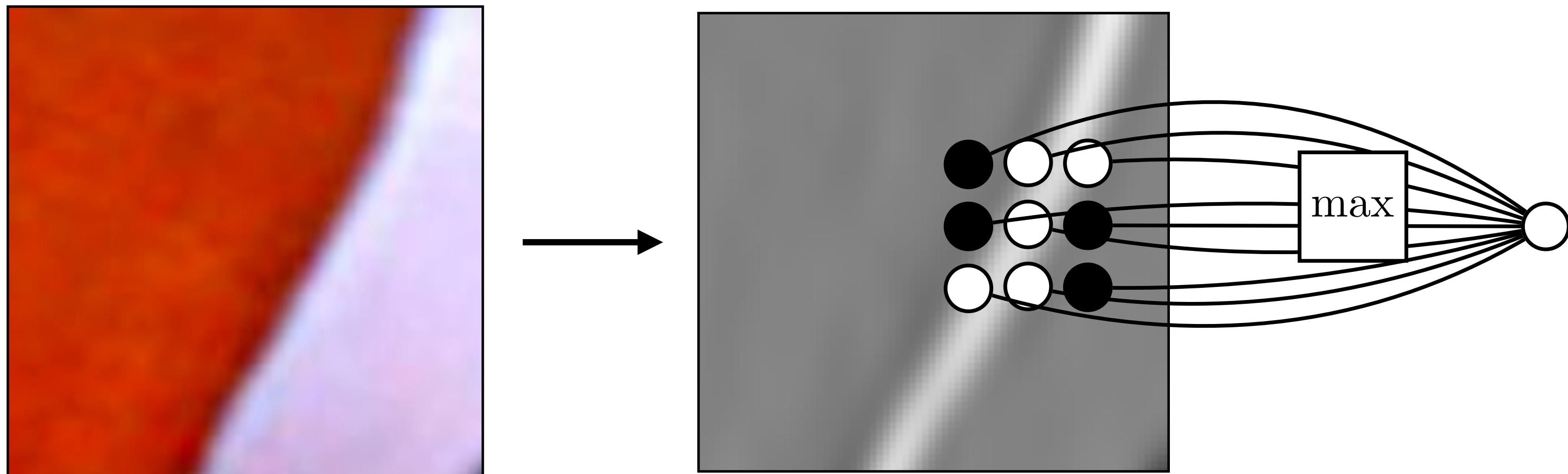
$$z_k = \max_{j \in \mathcal{N}(j)} g(y_j)$$

**Mean pooling**

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

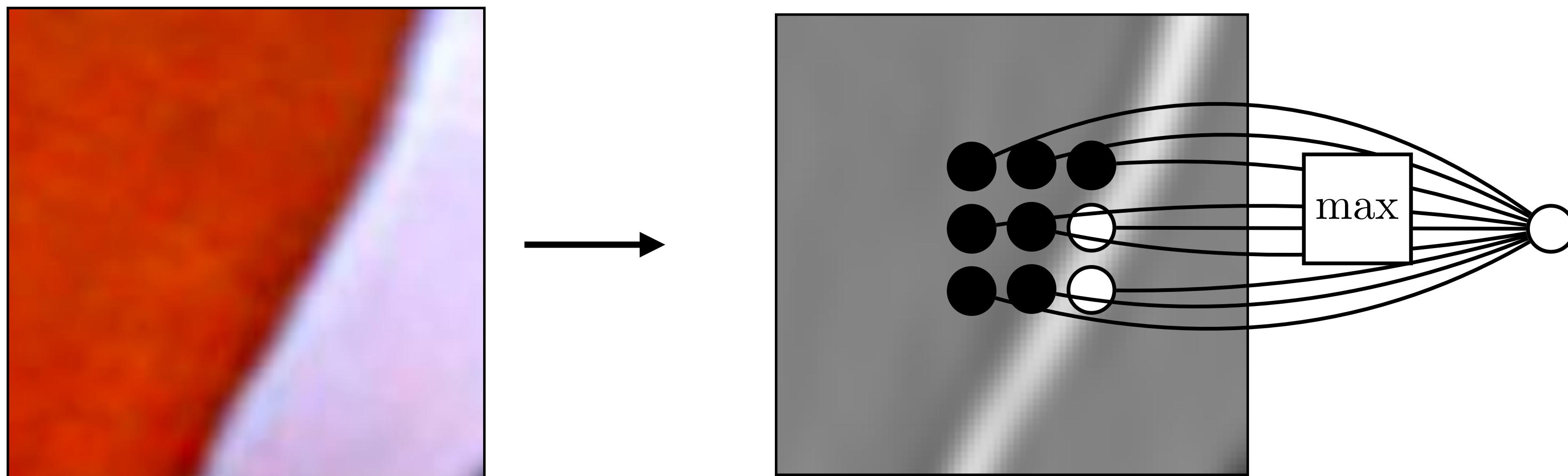
# Pooling – Why?

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



# Pooling – Why?

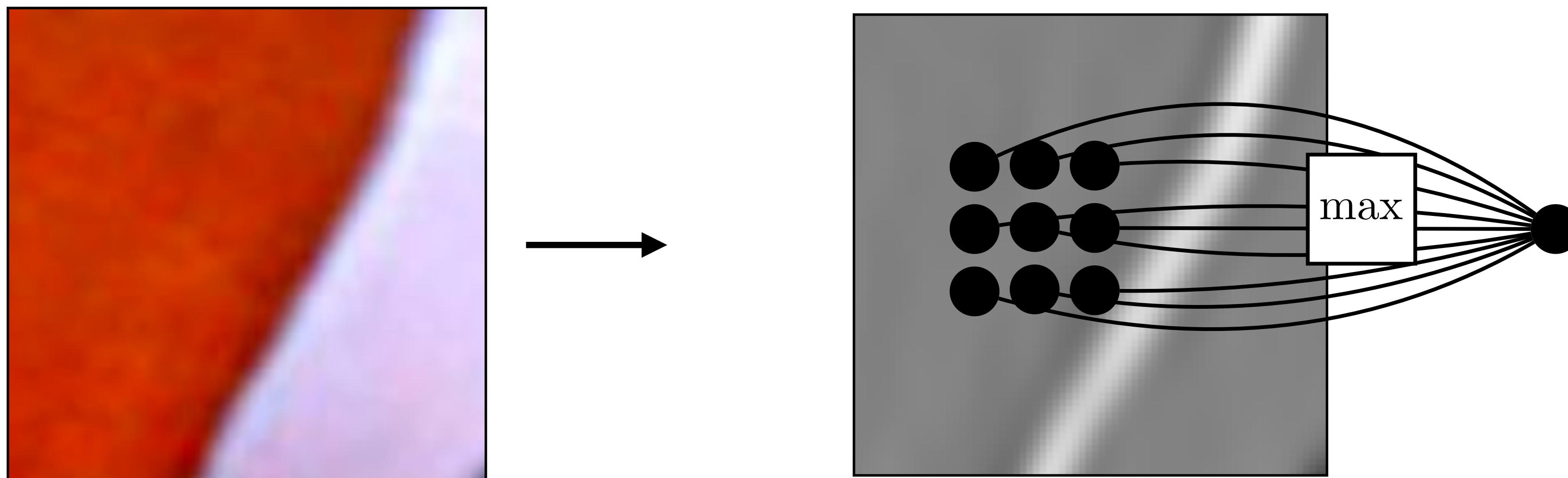
Pooling across spatial locations achieves stability w.r.t. small translations:



large response  
regardless of exact  
position of edge

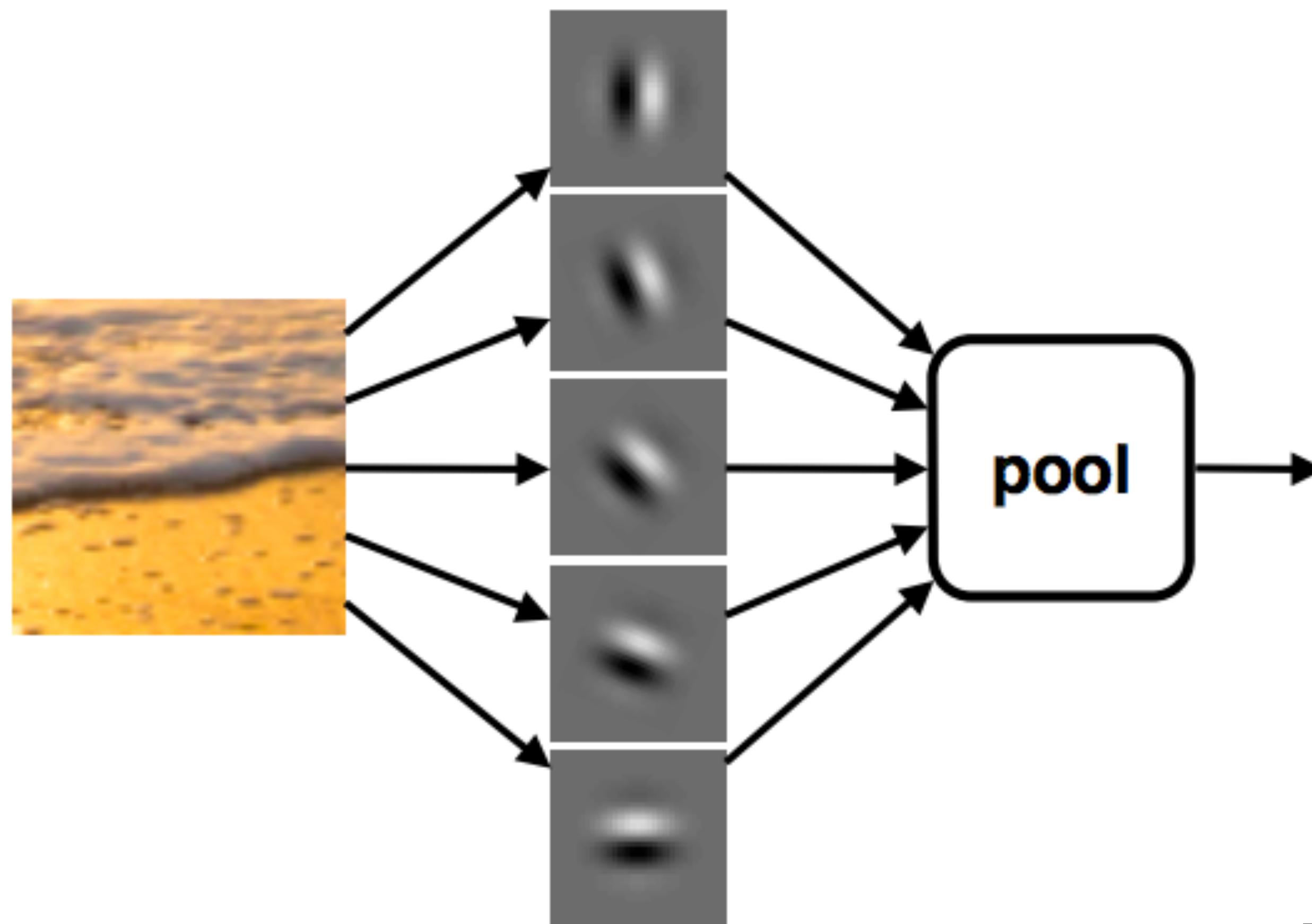
# Pooling – Why?

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



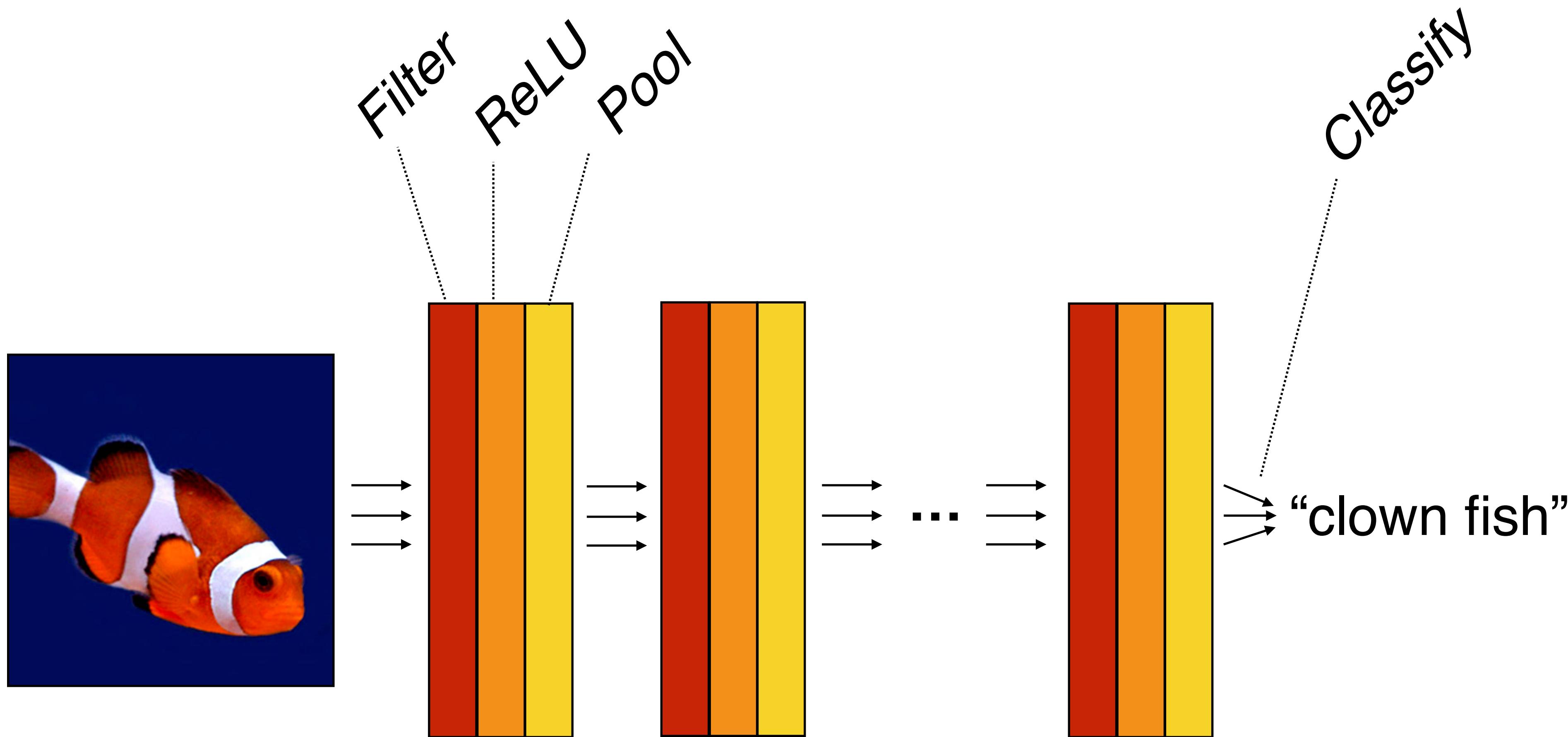
# Pooling – Why?

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



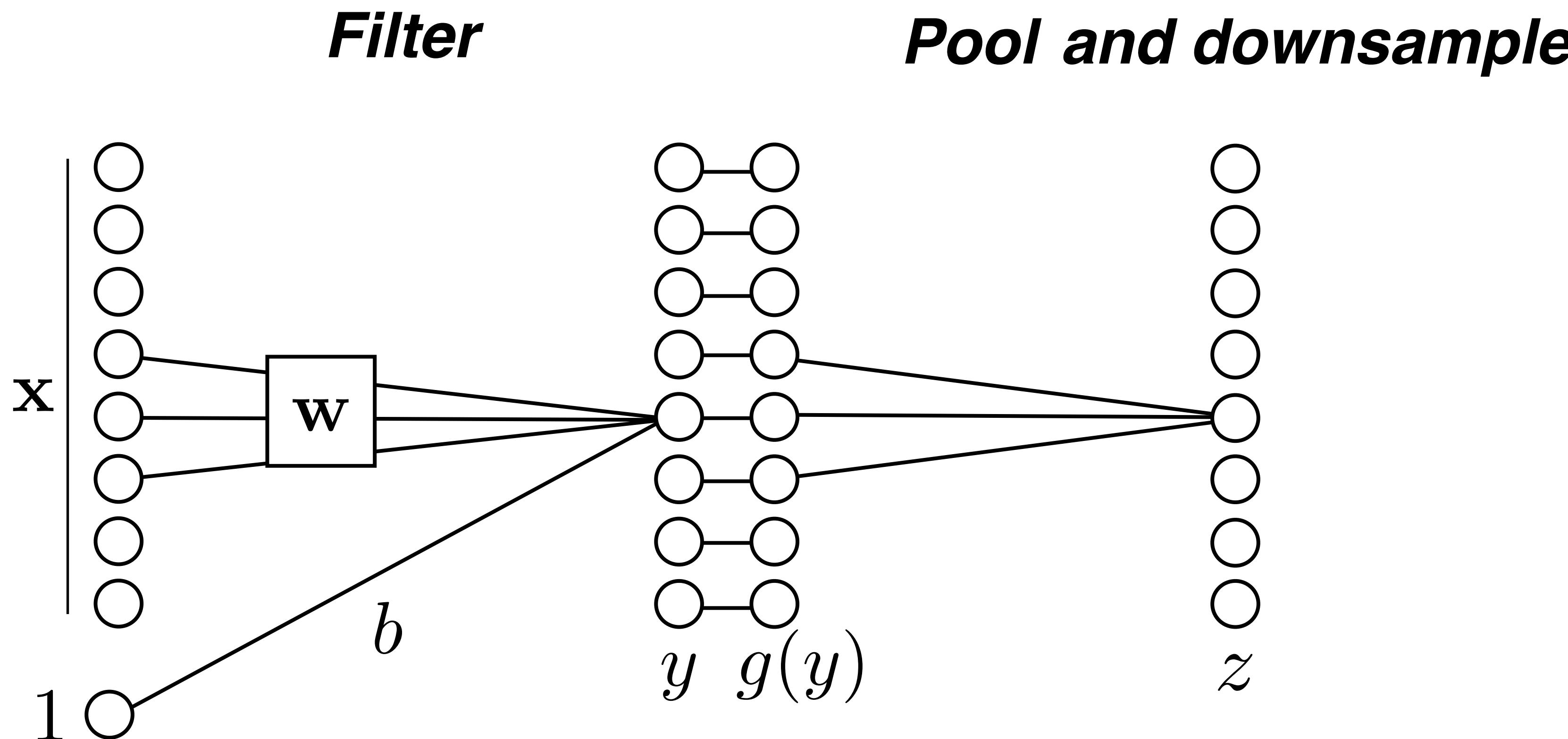
[Derived from slide by Andrea Vedaldi]

# Computation in a neural net

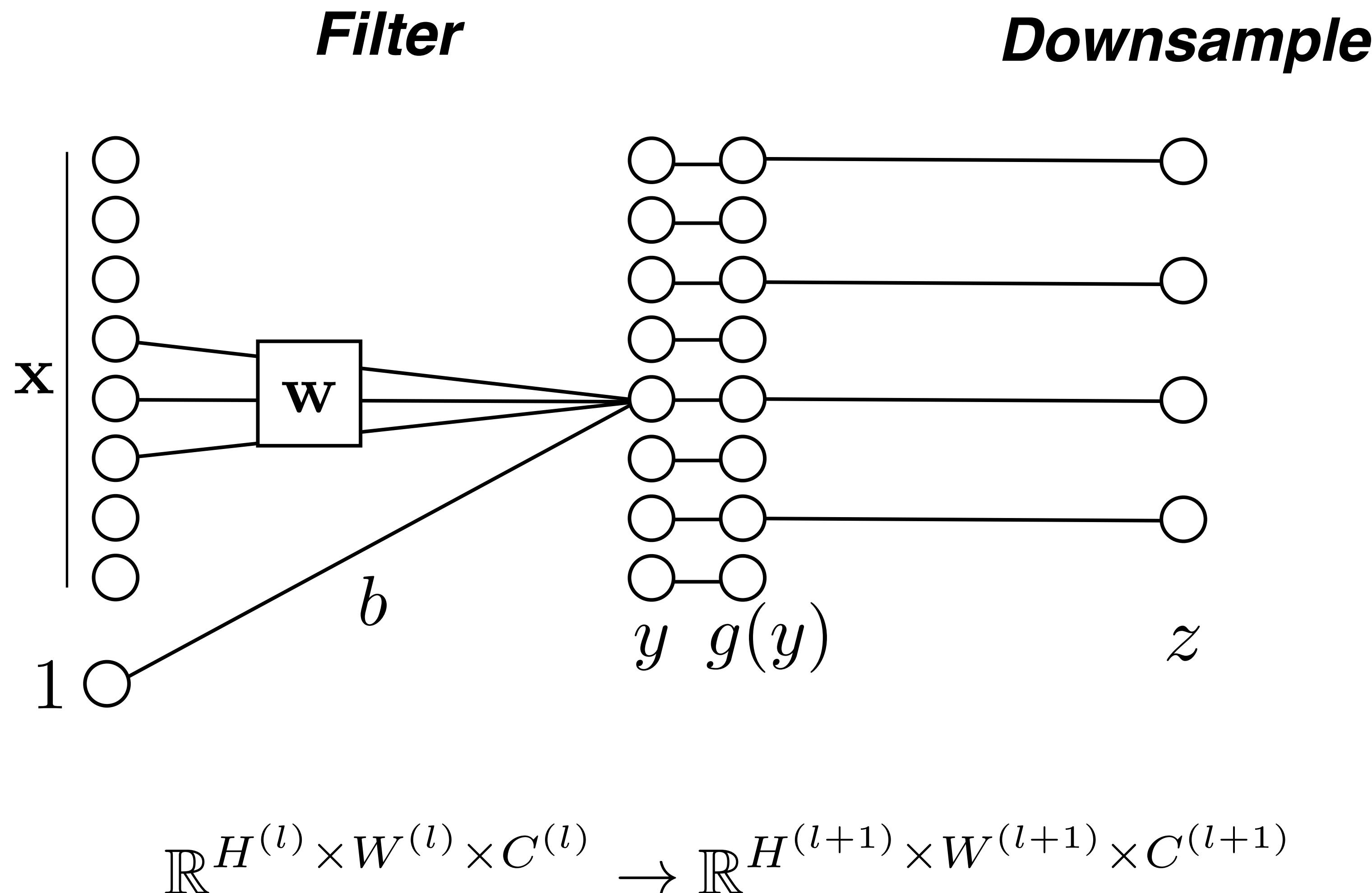


$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

# Downsampling

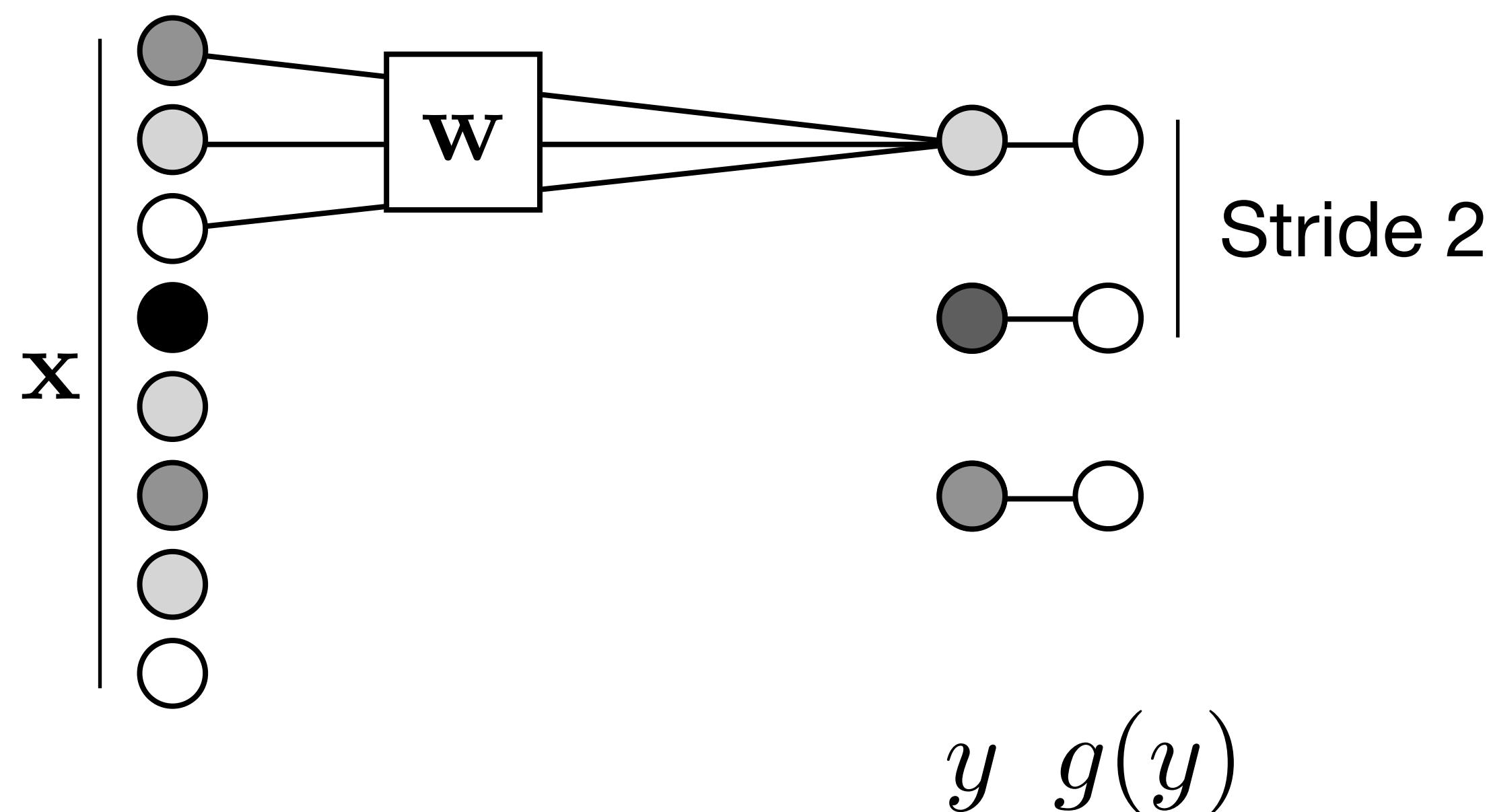


# Downsampling



# Strided operations

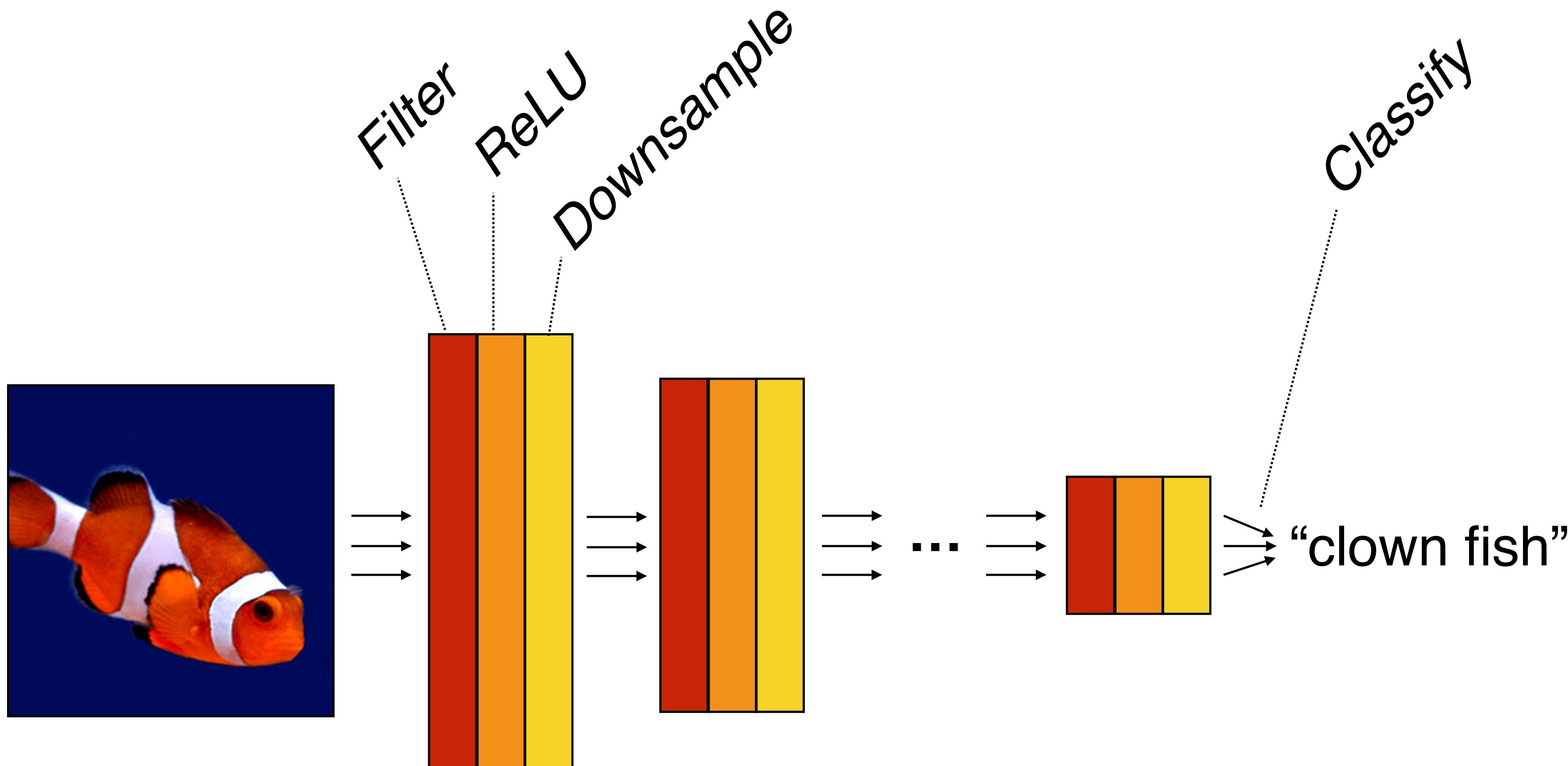
## Conv layer



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

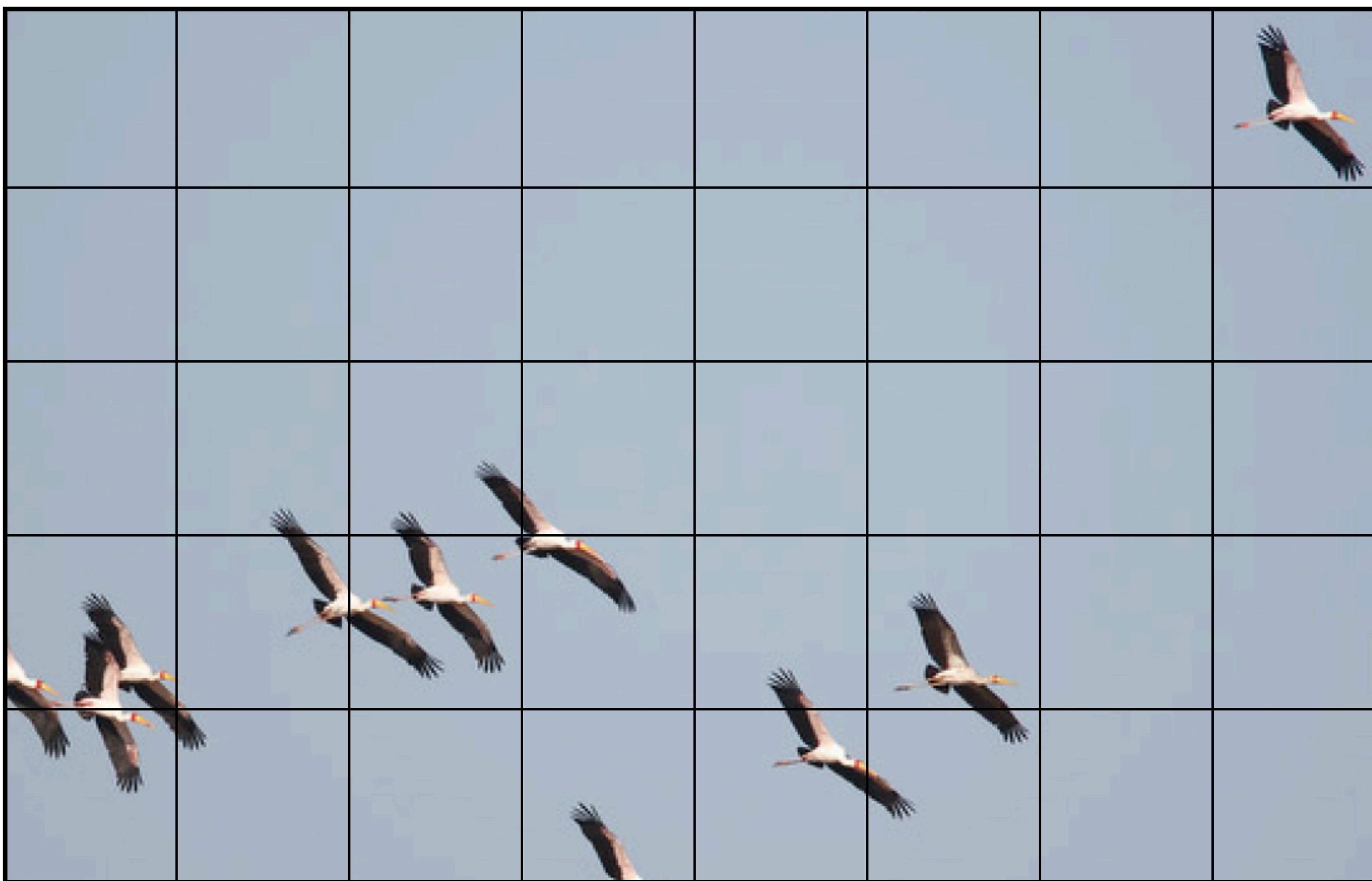
**Strided convolution is an alternative to pooling layers:** just do a strided convolution!

# Computation in a neural net

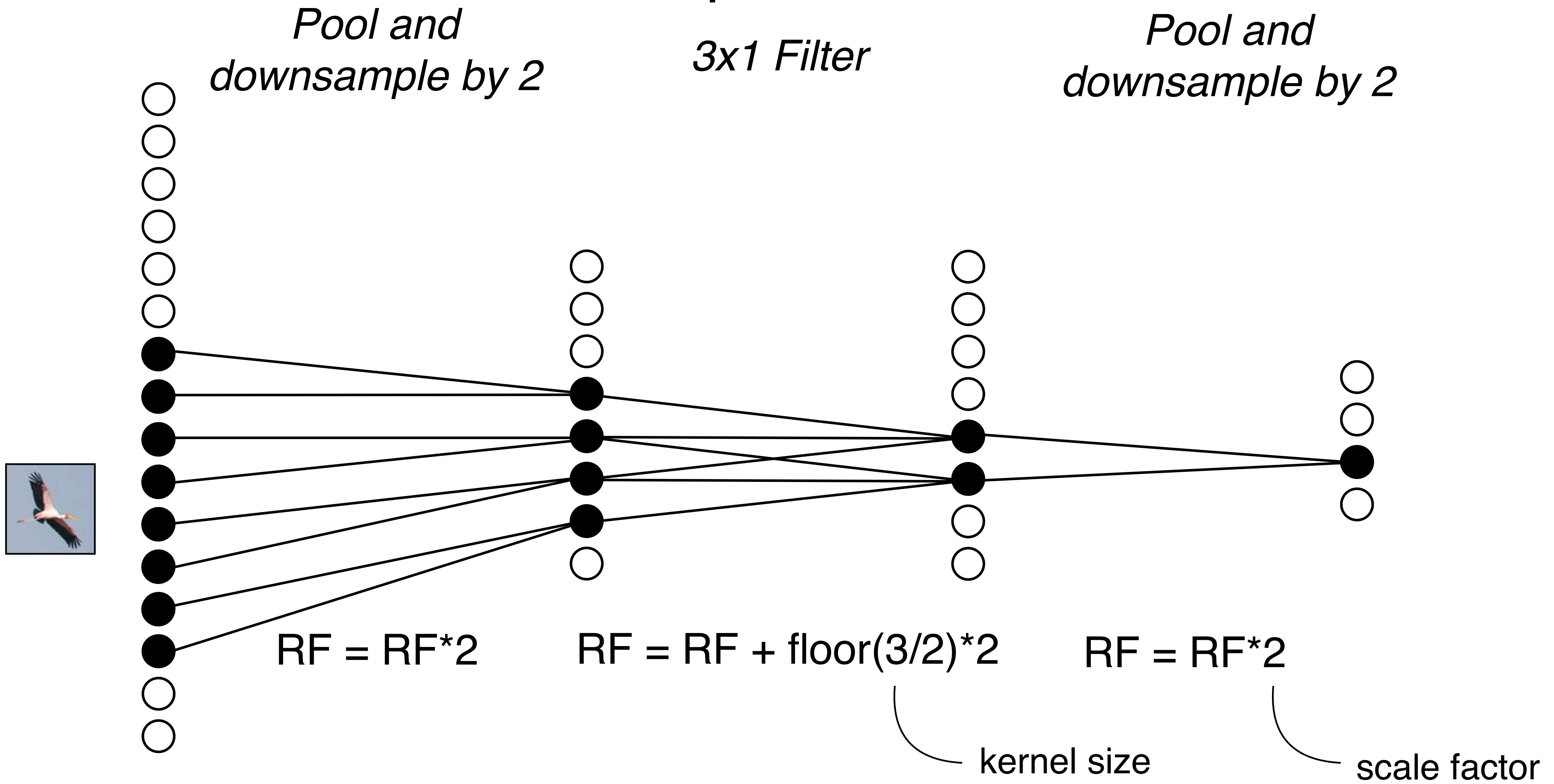


$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

# Receptive fields



# Receptive fields



# Effective Receptive Field

Contributing input units to a convolutional filter.

@jimmfleming // fomoro.com



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

# Gradients for backprop

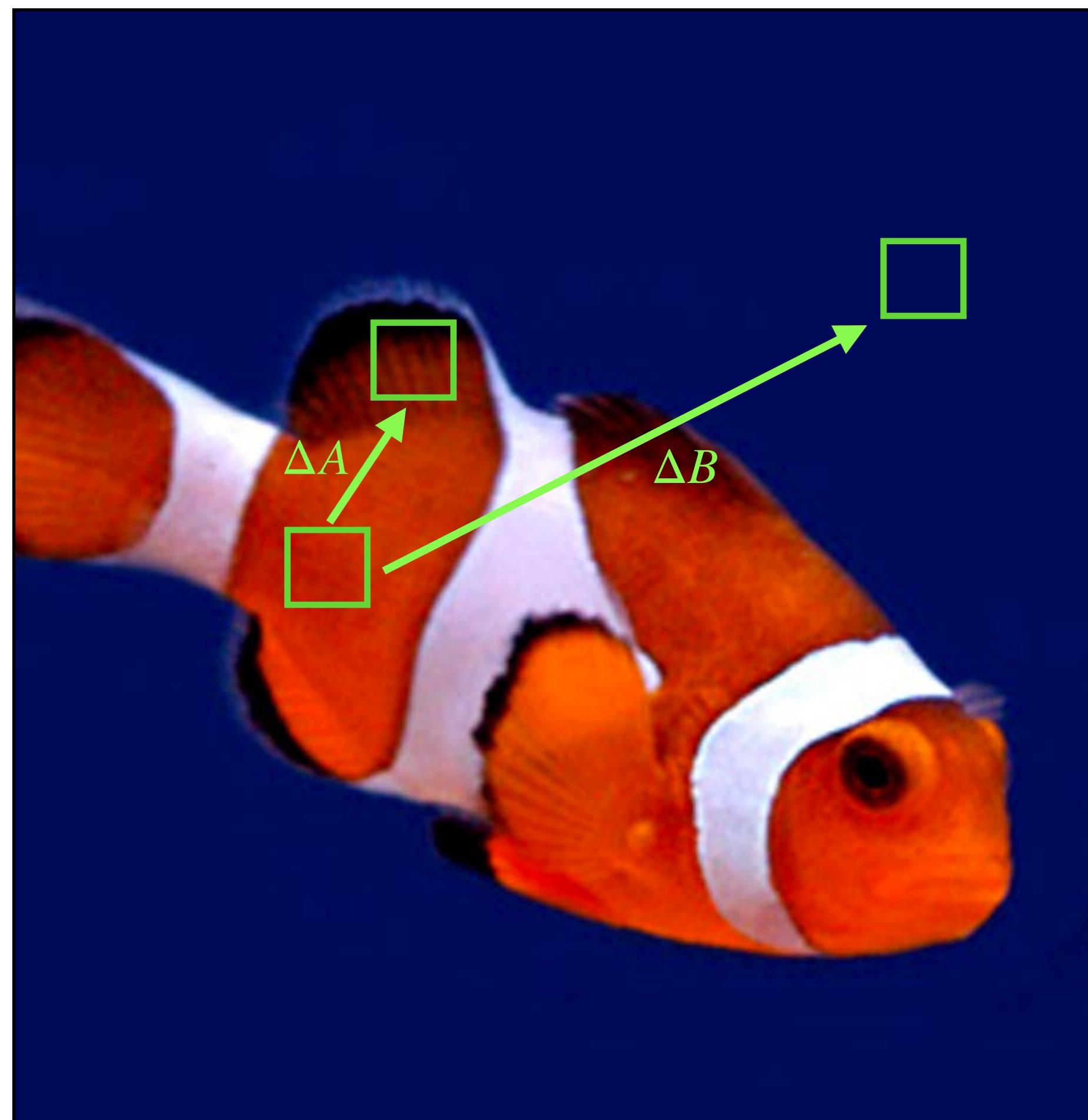
## Convolutional layers:

**Option 1:** Just make the giant convolution matrix and use the fully backprop equations for fully connected layers!  
Early deep learning frameworks did this.

**Option 2:** Use a more optimized implementation. Result looks similar to convolution.

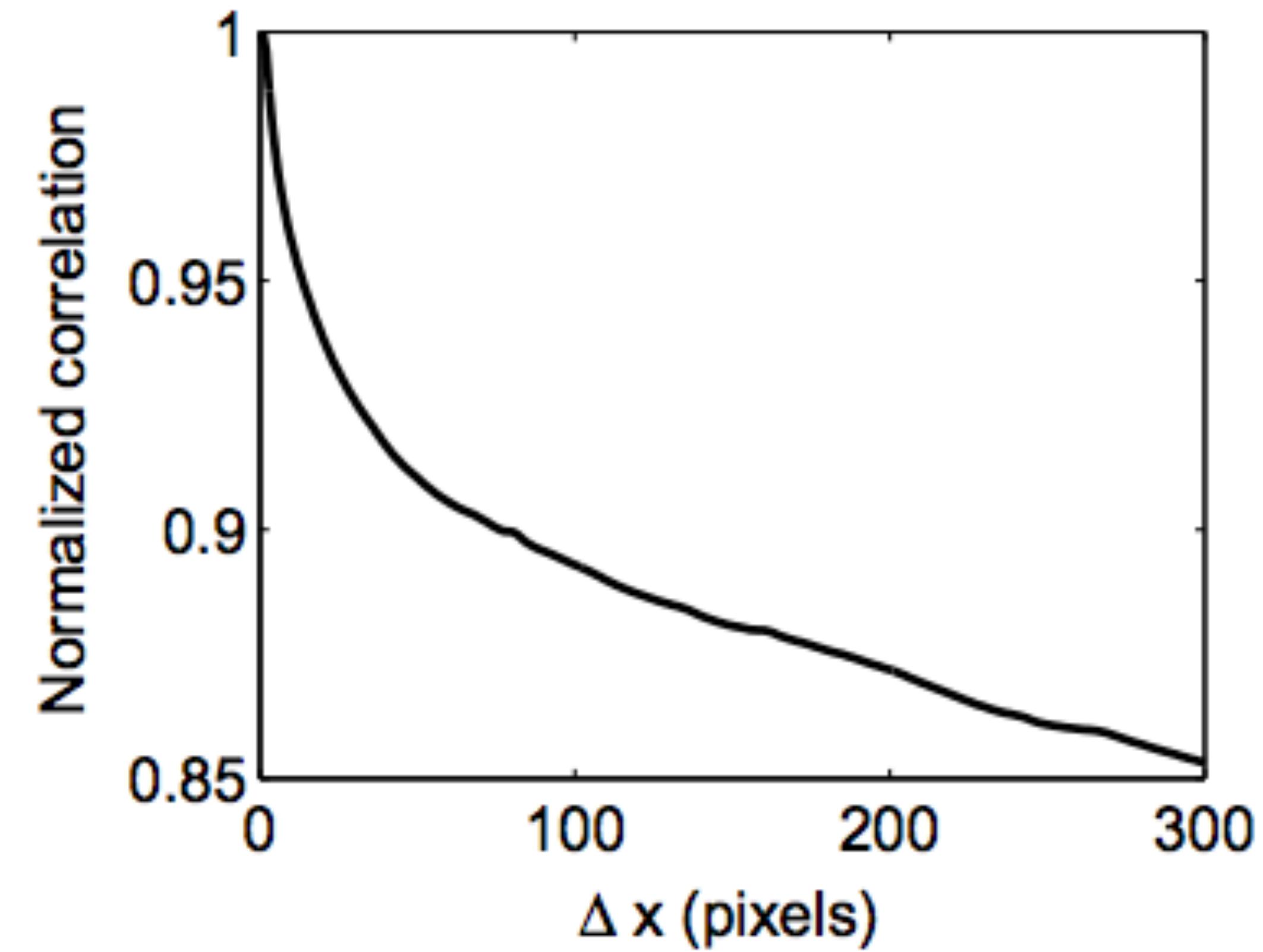
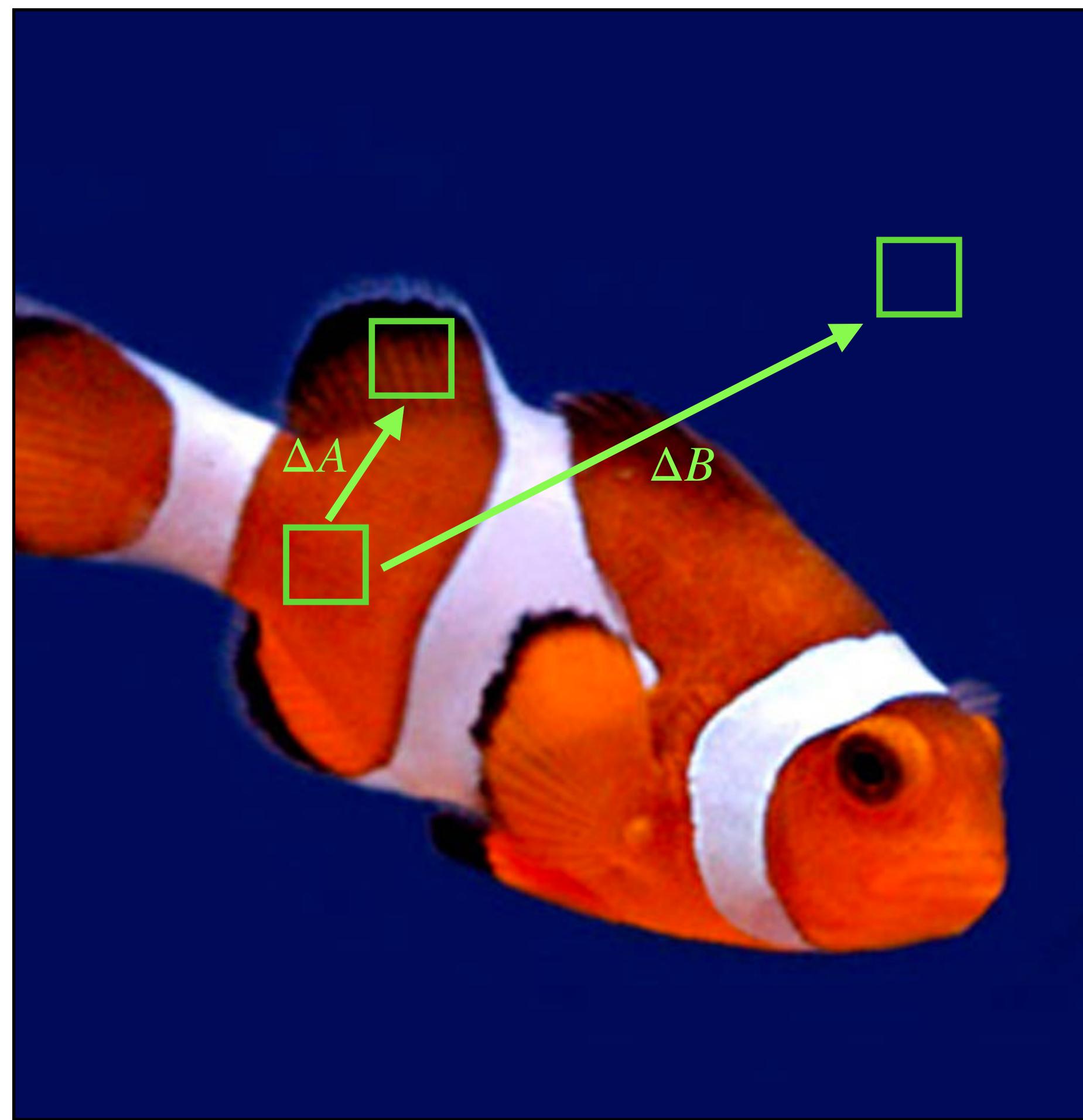
**Max pooling:** similar to ReLU gradient.

# Local vs. global processing



- Across all images, which is higher:
- (1) correlation between points with distance  $\Delta A$
  - (2) correlation between points with distance  $\Delta B$
  - (3) can't say

# Local vs. global processing



# CNNs – Why?

Statistical dependences between pixels decay as a power law of distance between the pixels.

It is therefore often sufficient to model local dependences only. —> **Convolution**

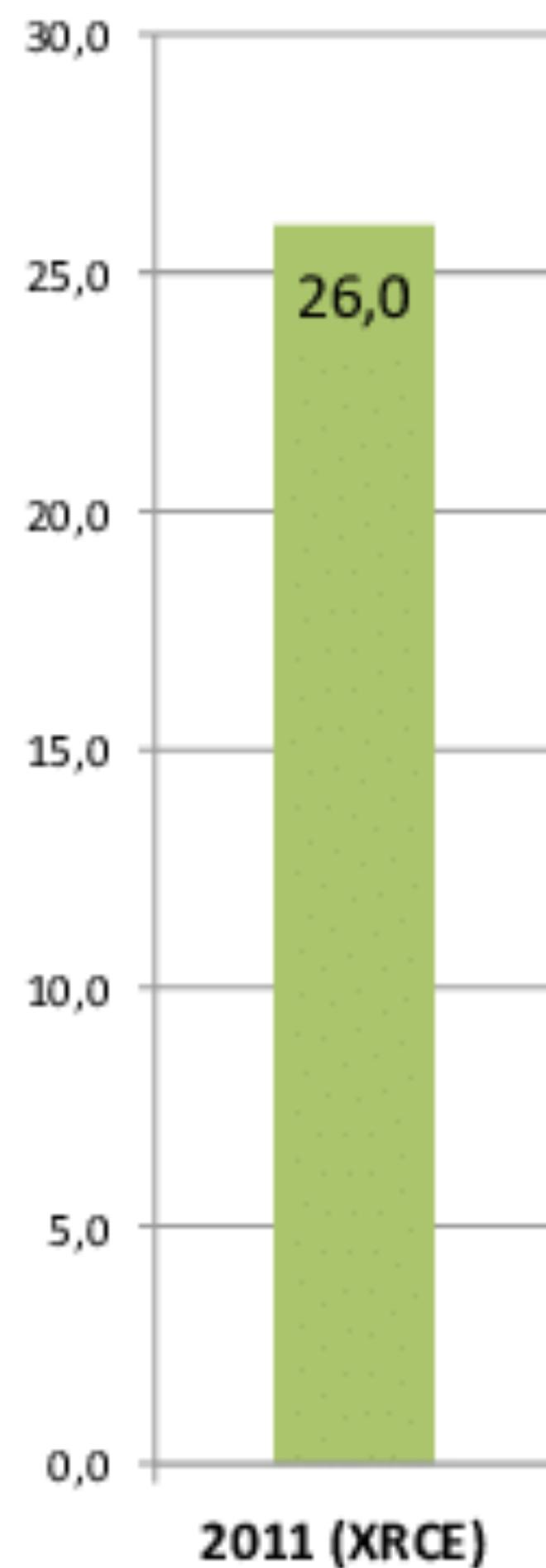
More generally, we should allocate parameters that model dependencies in proportion to the strength of those dependences. —> **Multiscale, hierarchical representations**

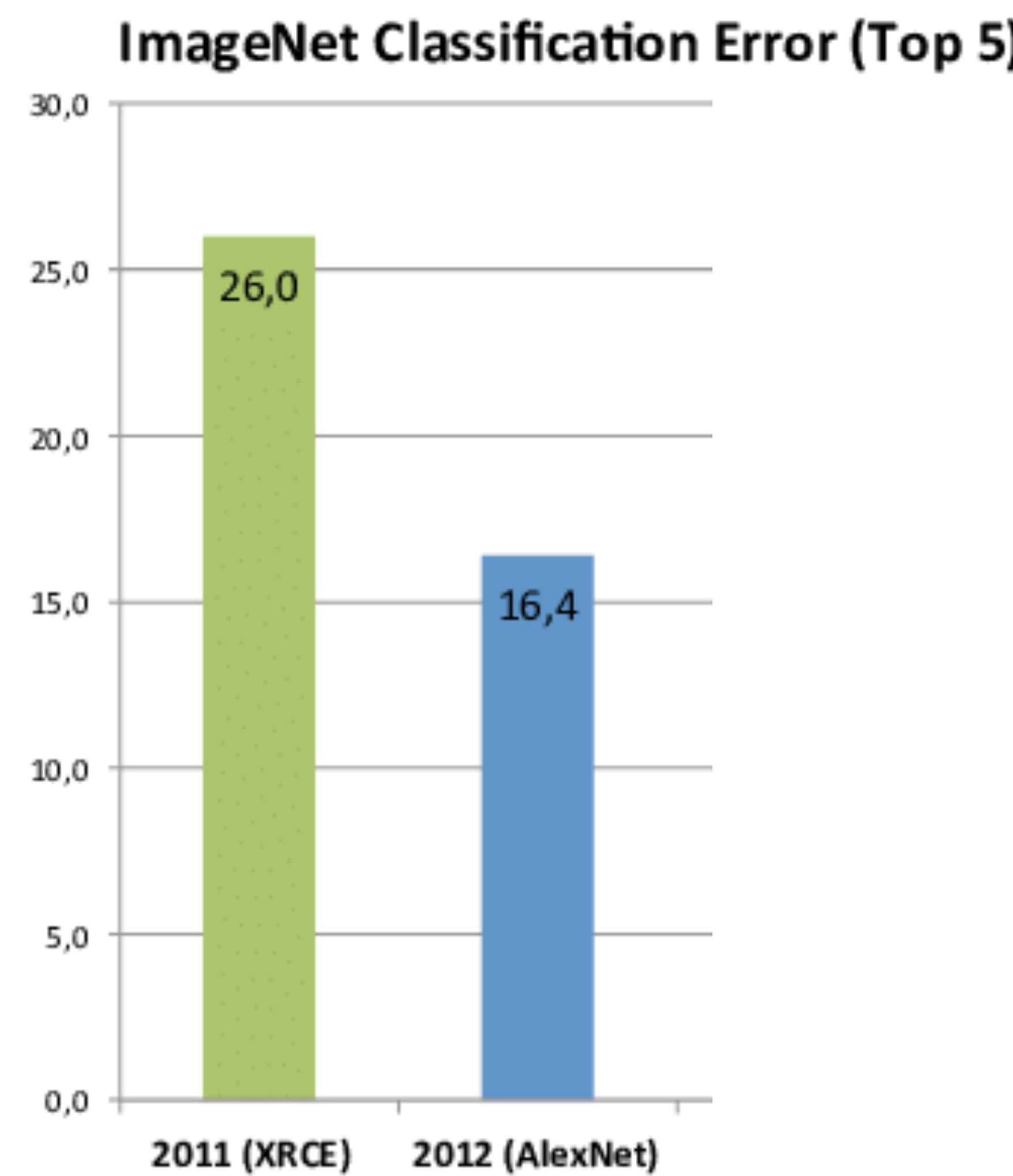
[For more discussion, see “Why does Deep and Cheap Learning Work So Well?”, Lin et al. 2017]

# Some networks

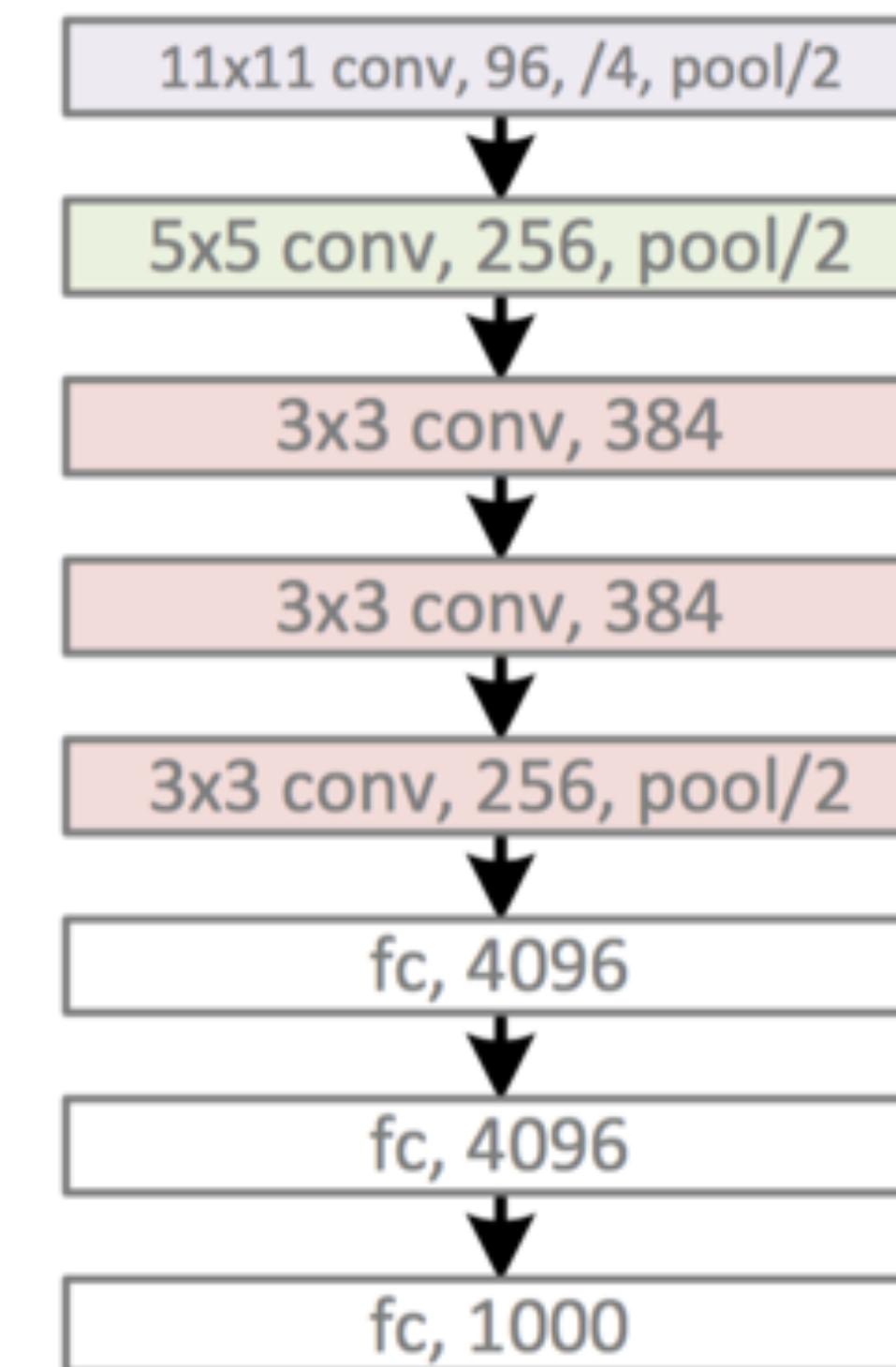
... and what makes them work

## ImageNet Classification Error (Top 5)





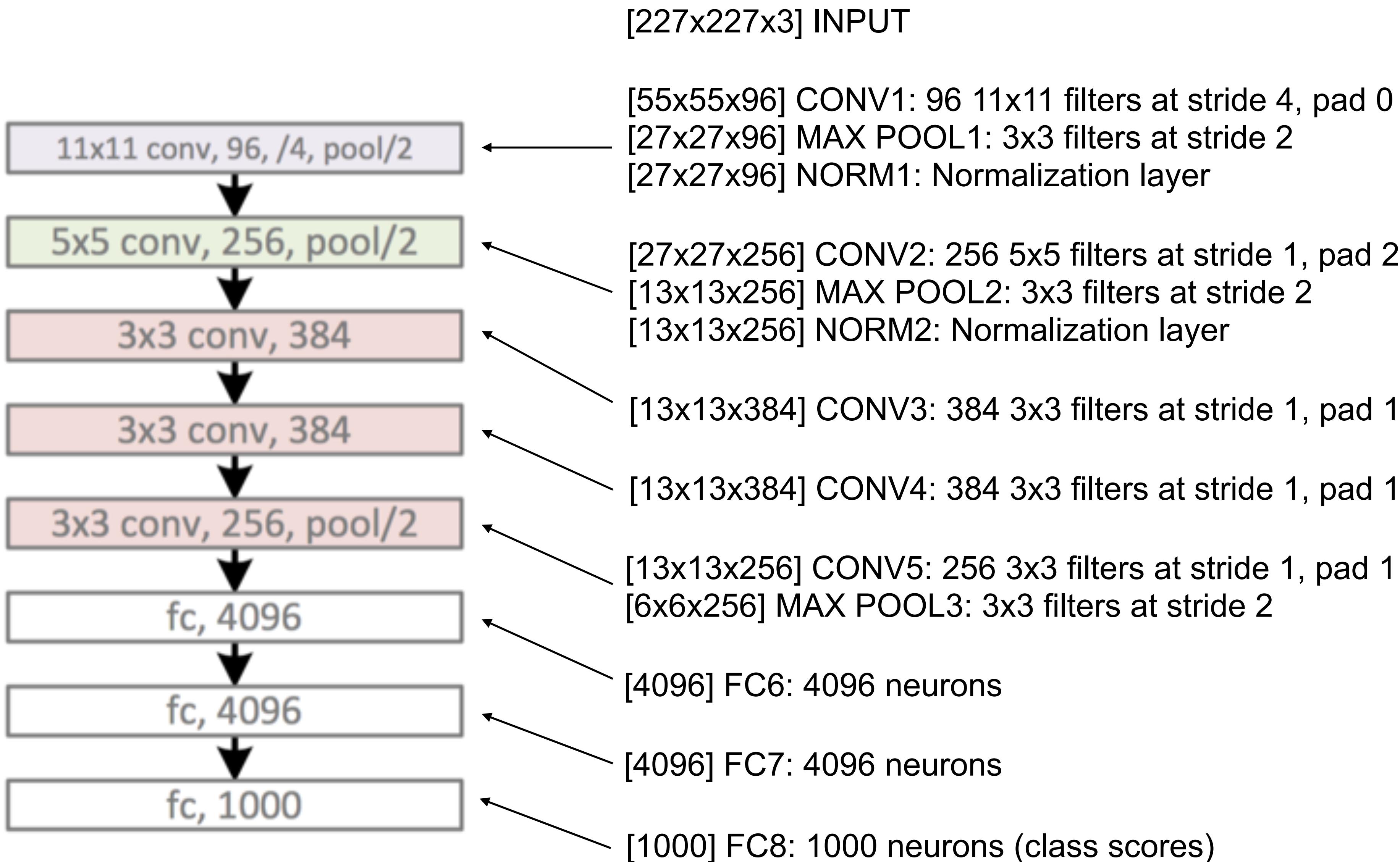
2012: AlexNet  
5 conv. layers

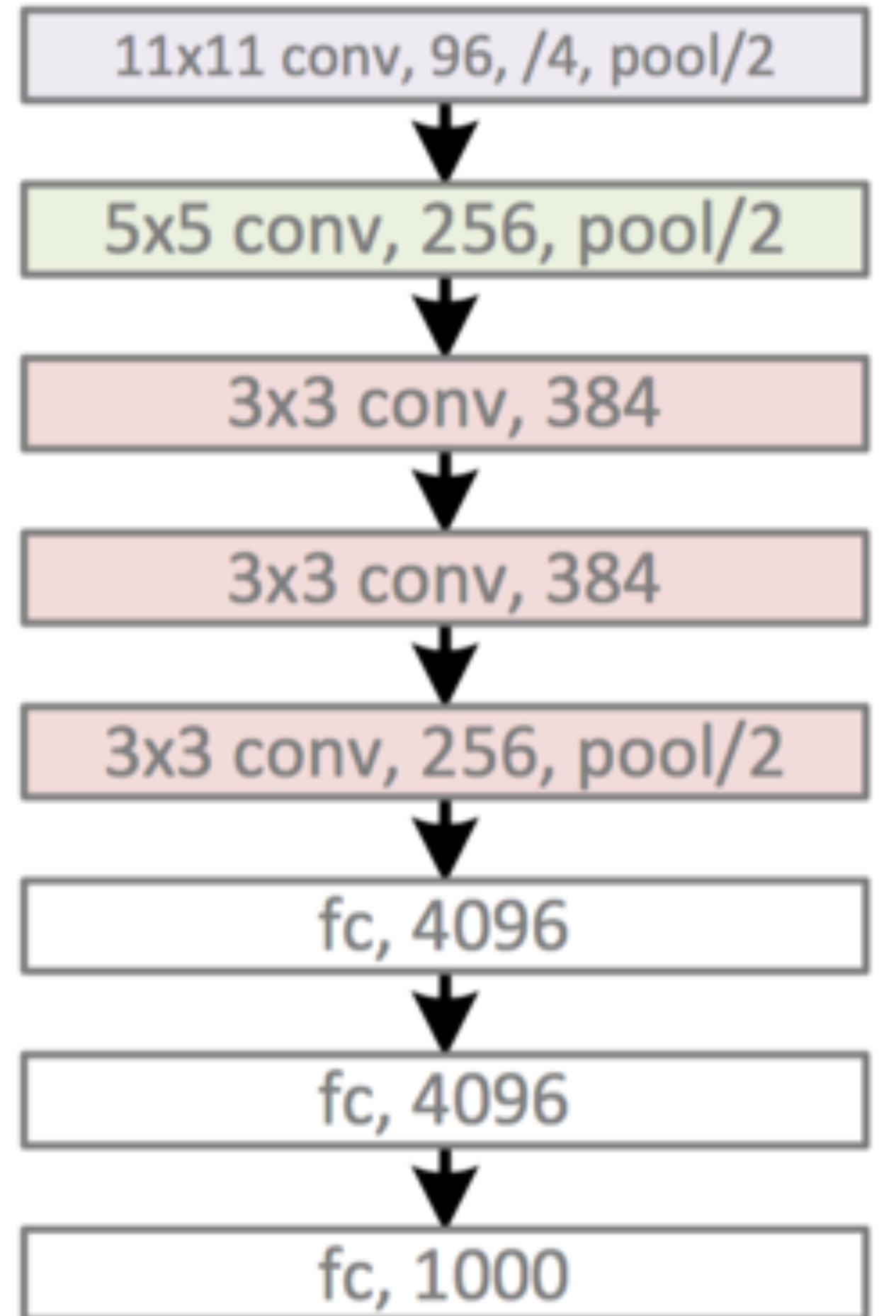


Error: 16.4%

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

# Alexnet – [Krizhevsky et al. NIPS 2012]

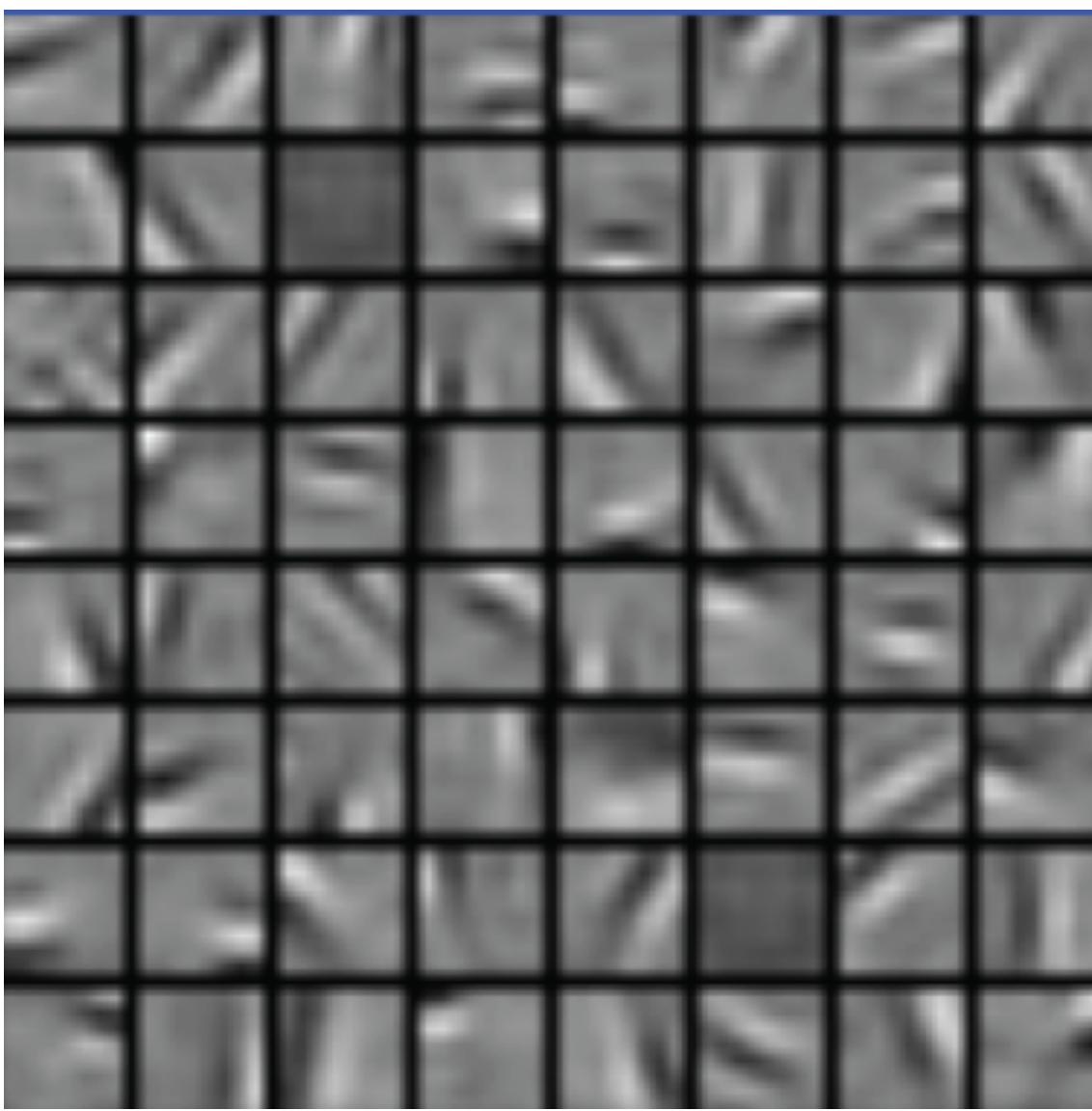




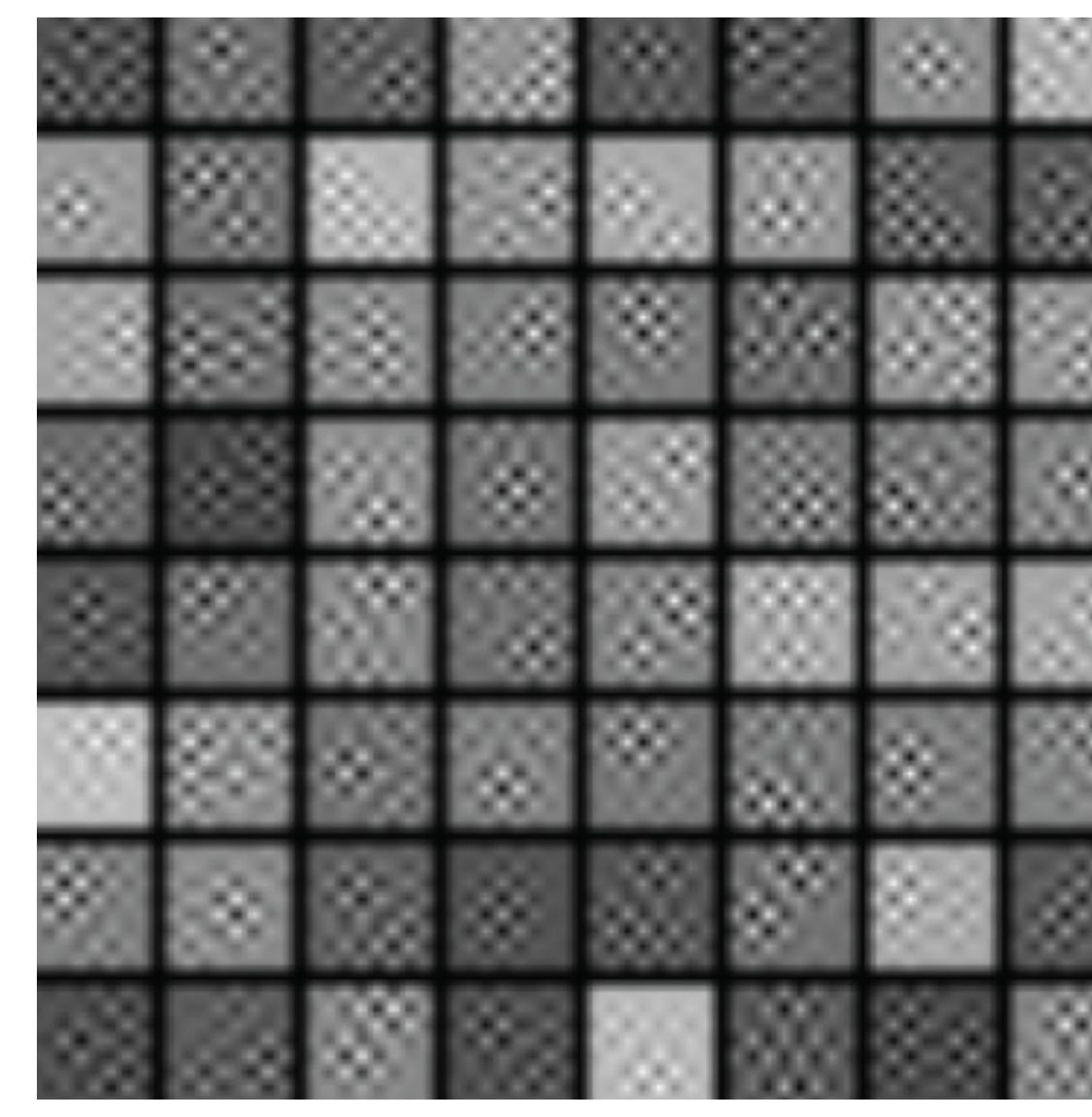
What filters are learned?

# What filters are learned?

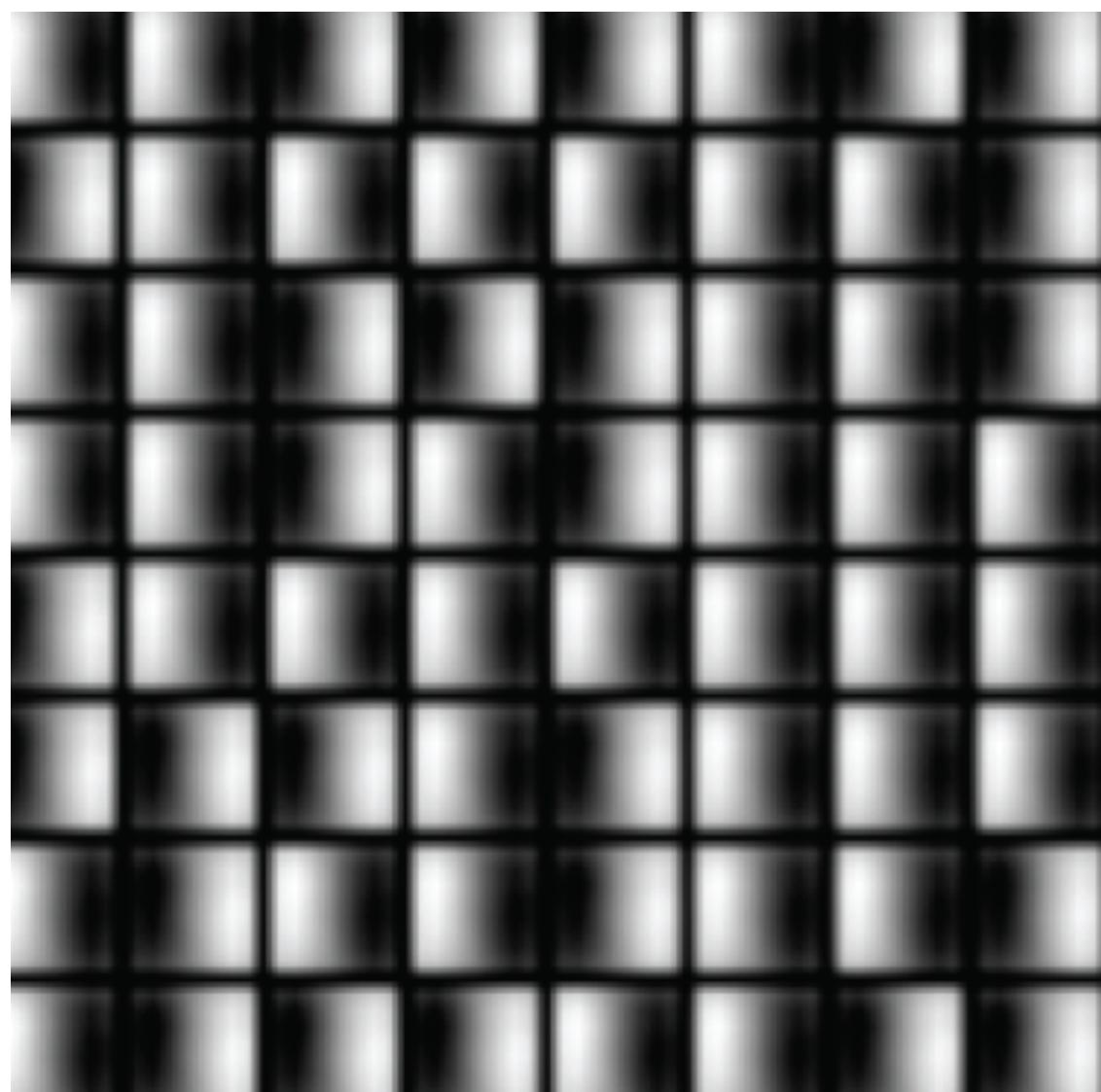
A



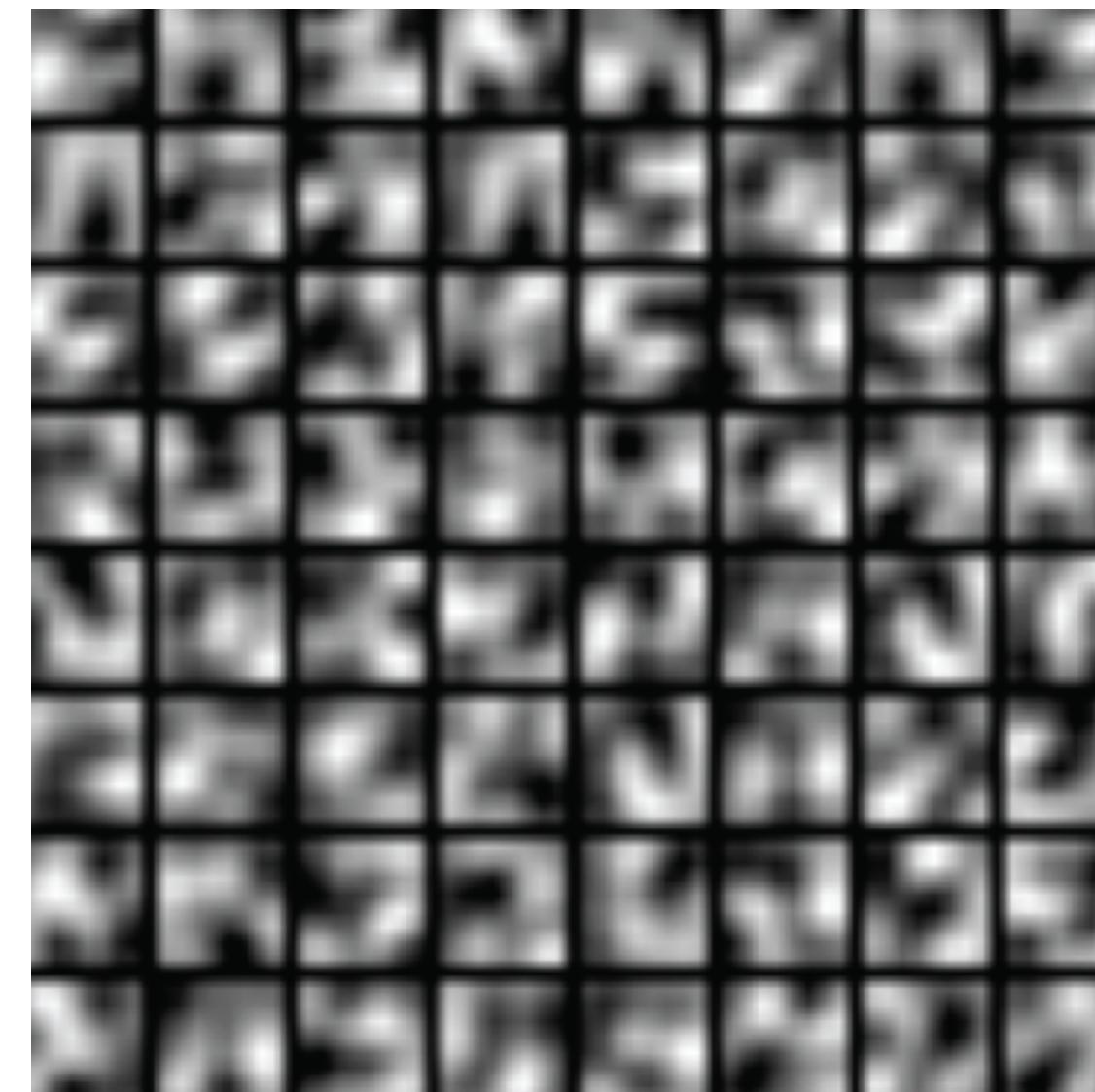
B



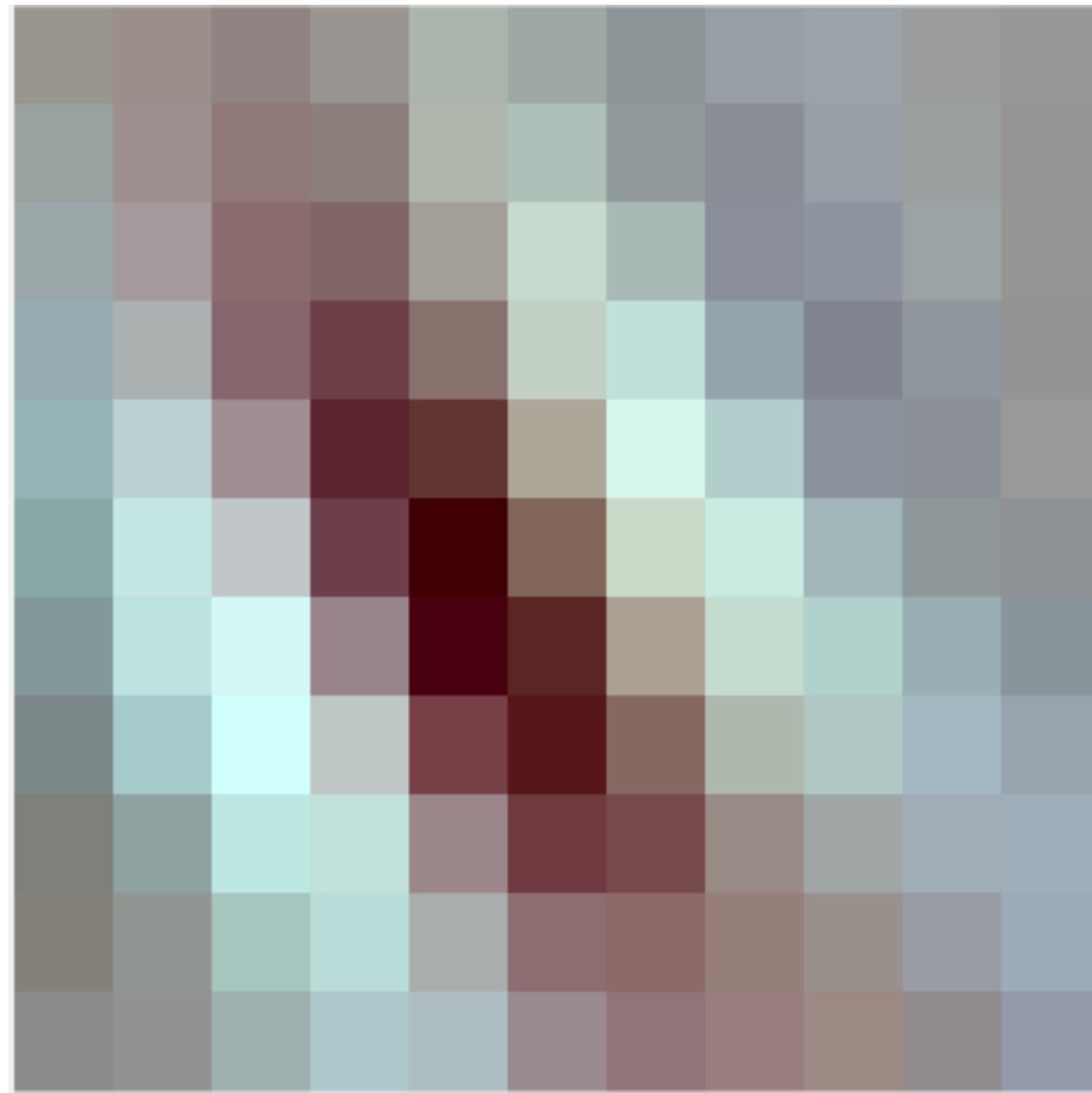
C



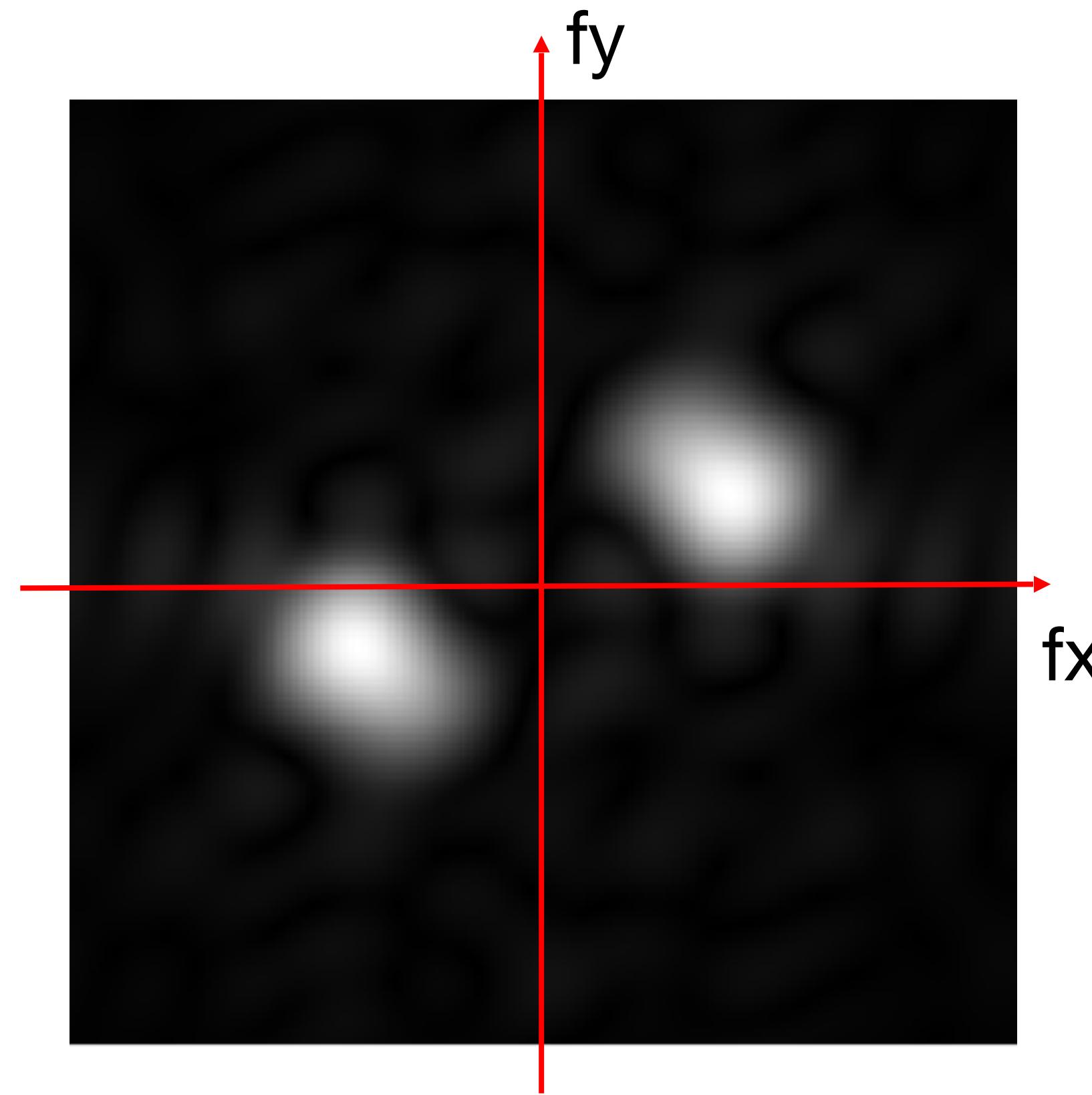
D



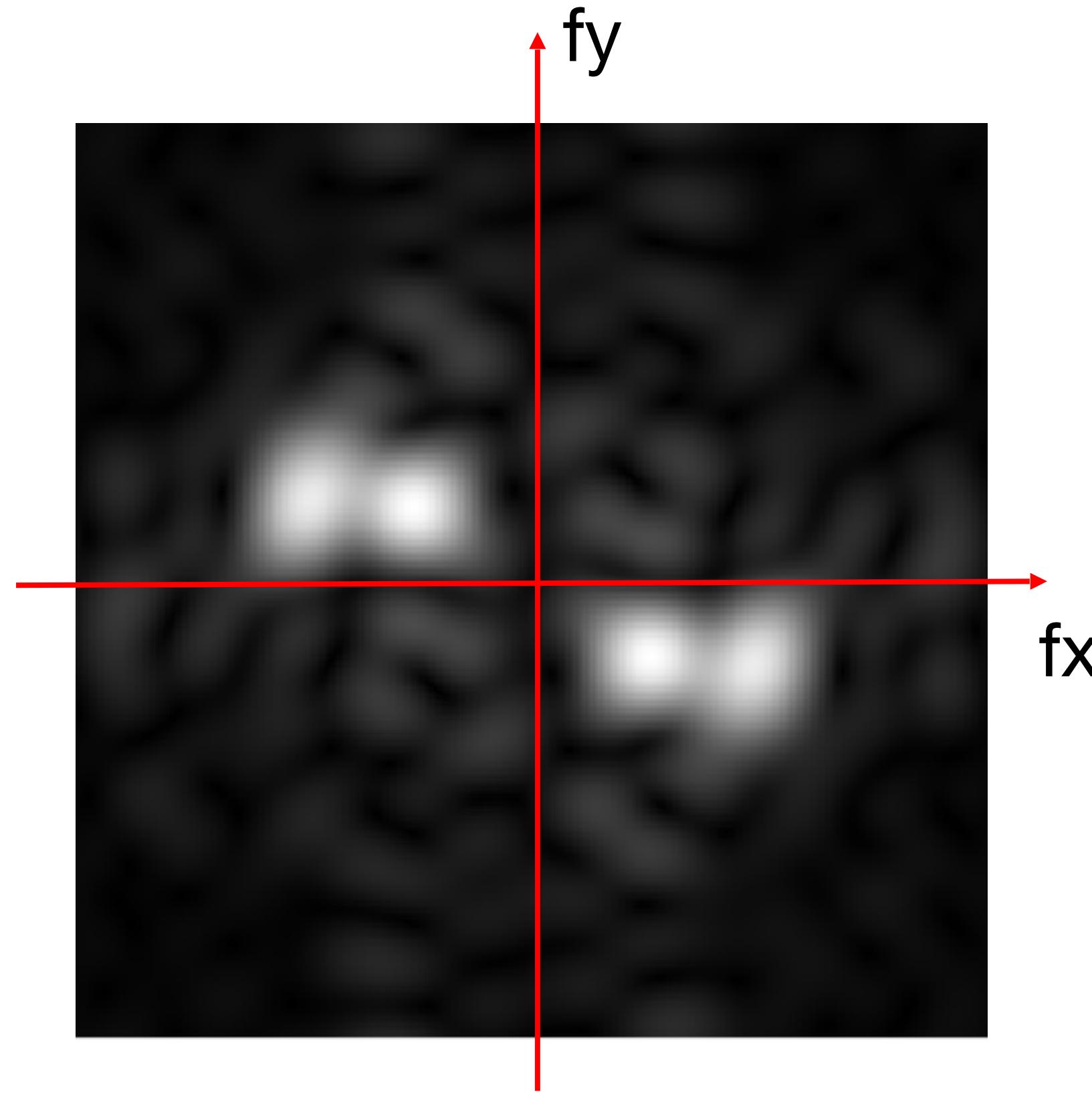
# Get to know your units



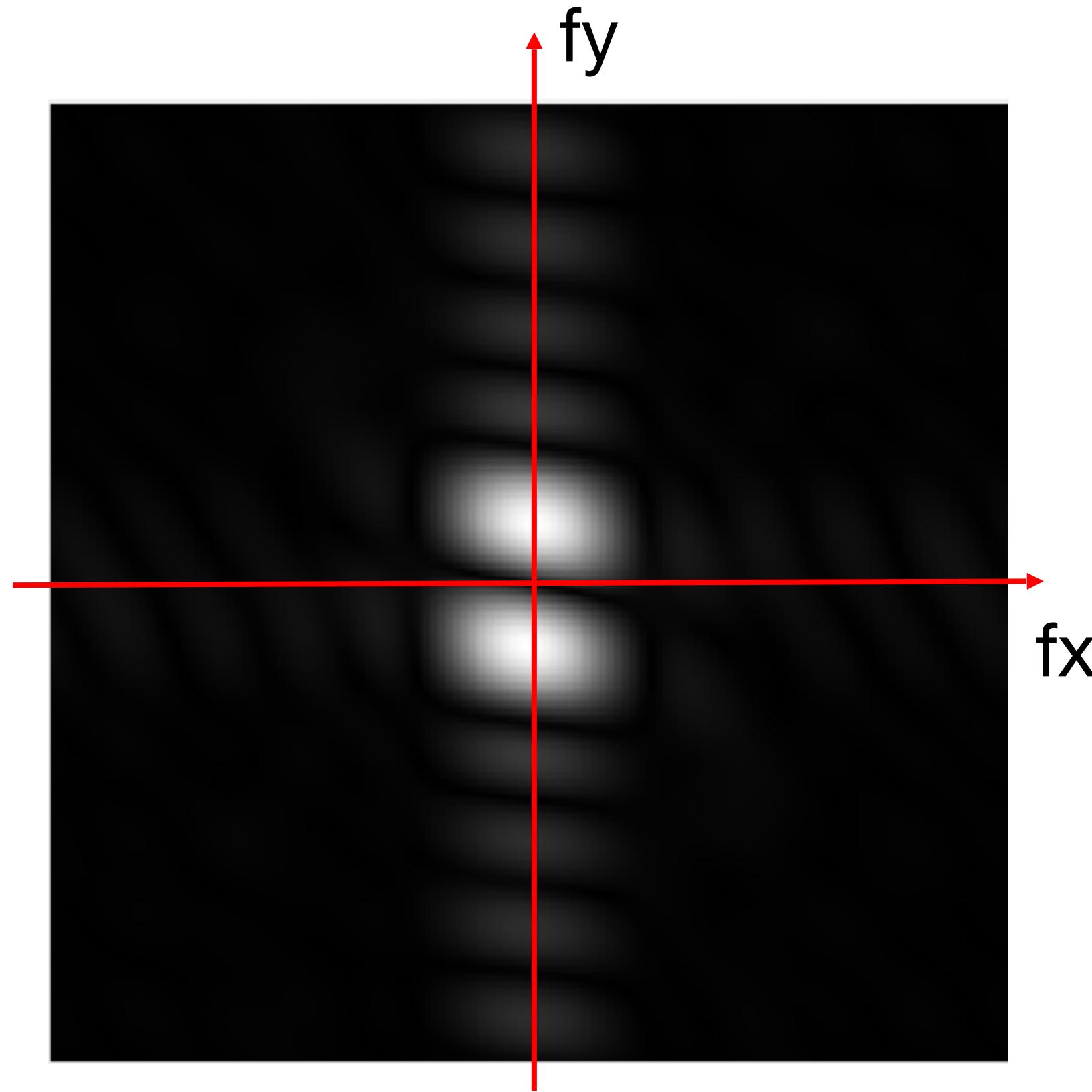
11x11 convolution kernel  
(3 color channels)



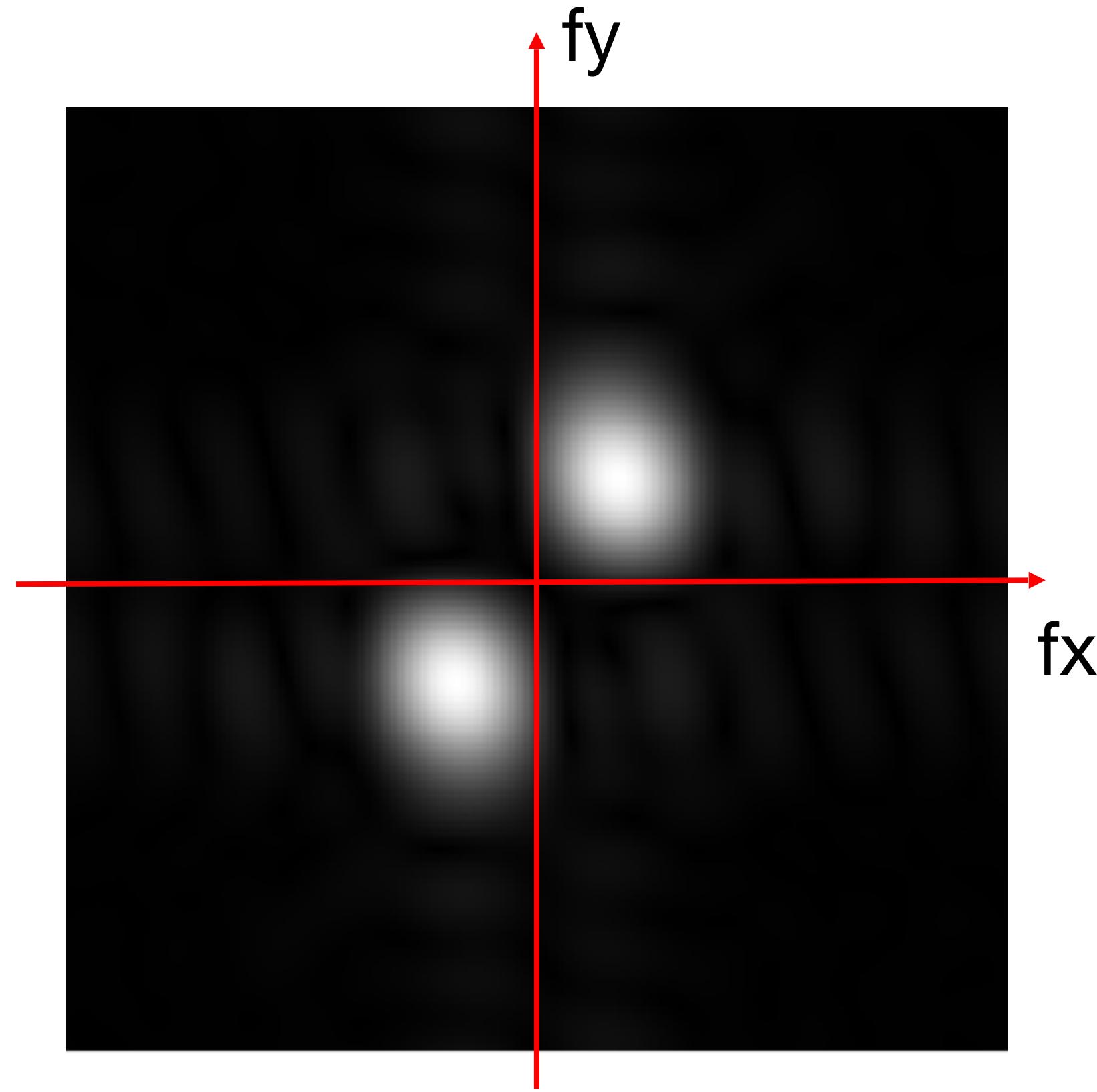
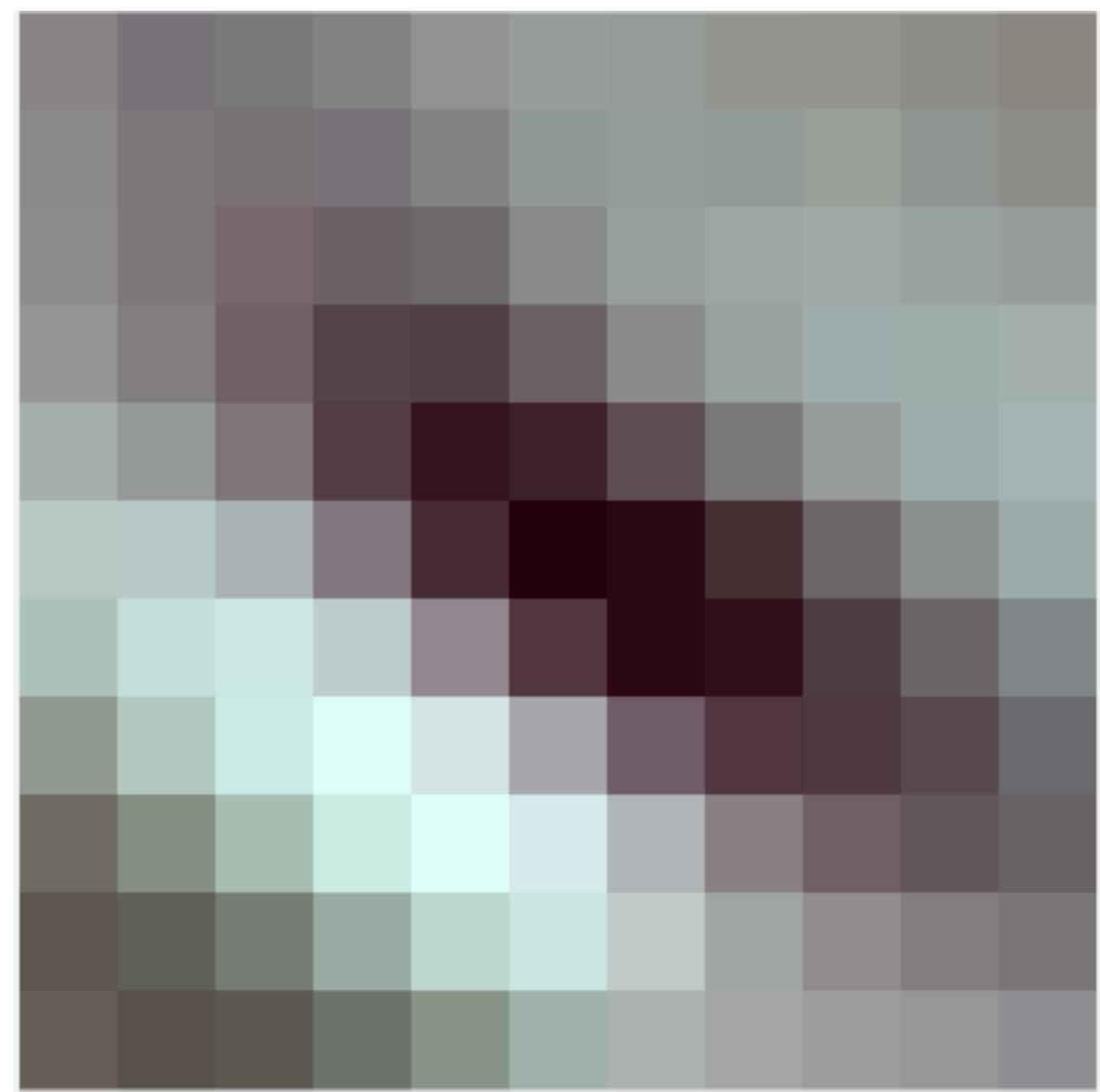
# Get to know your units



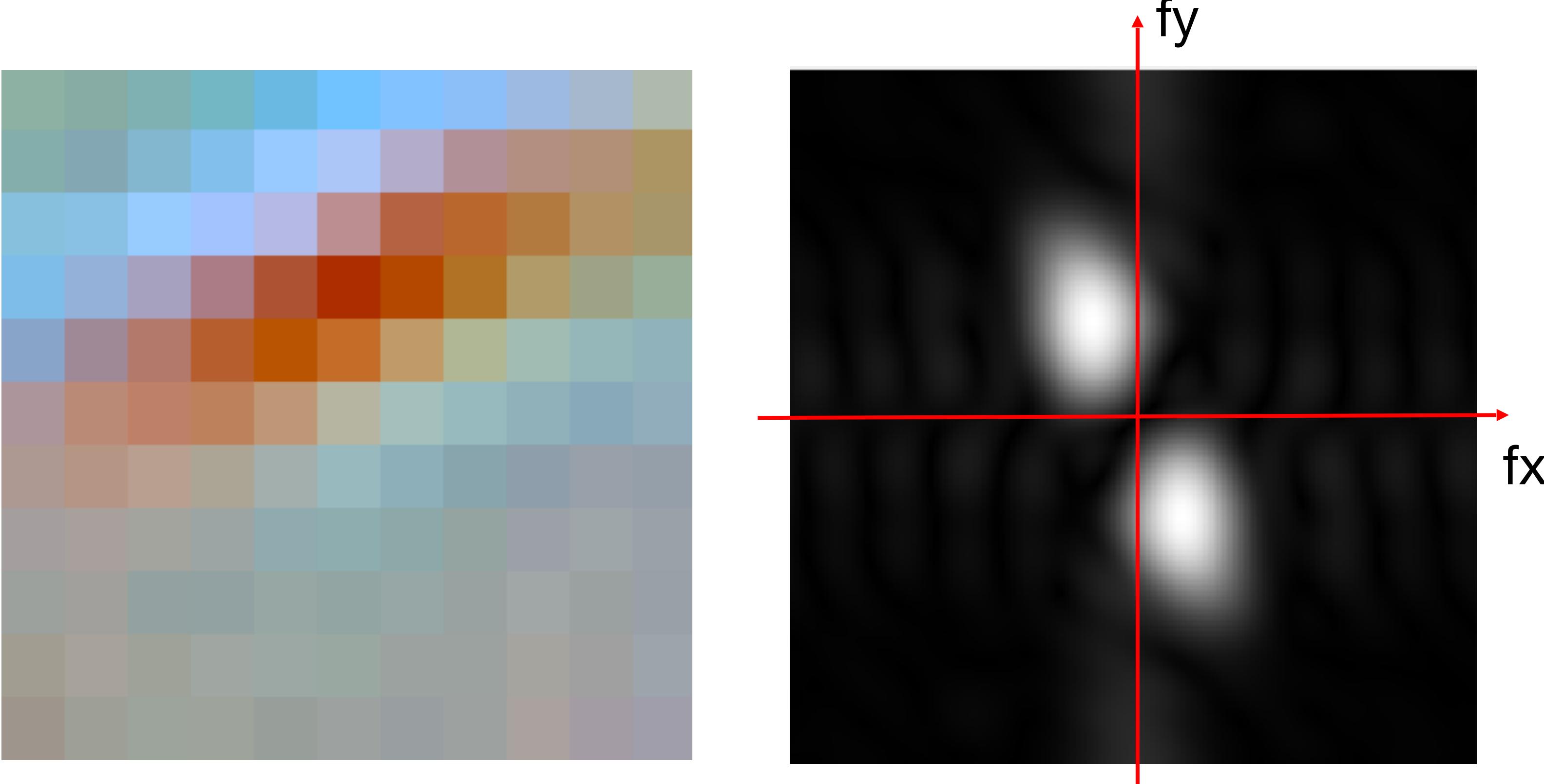
# Get to know your units



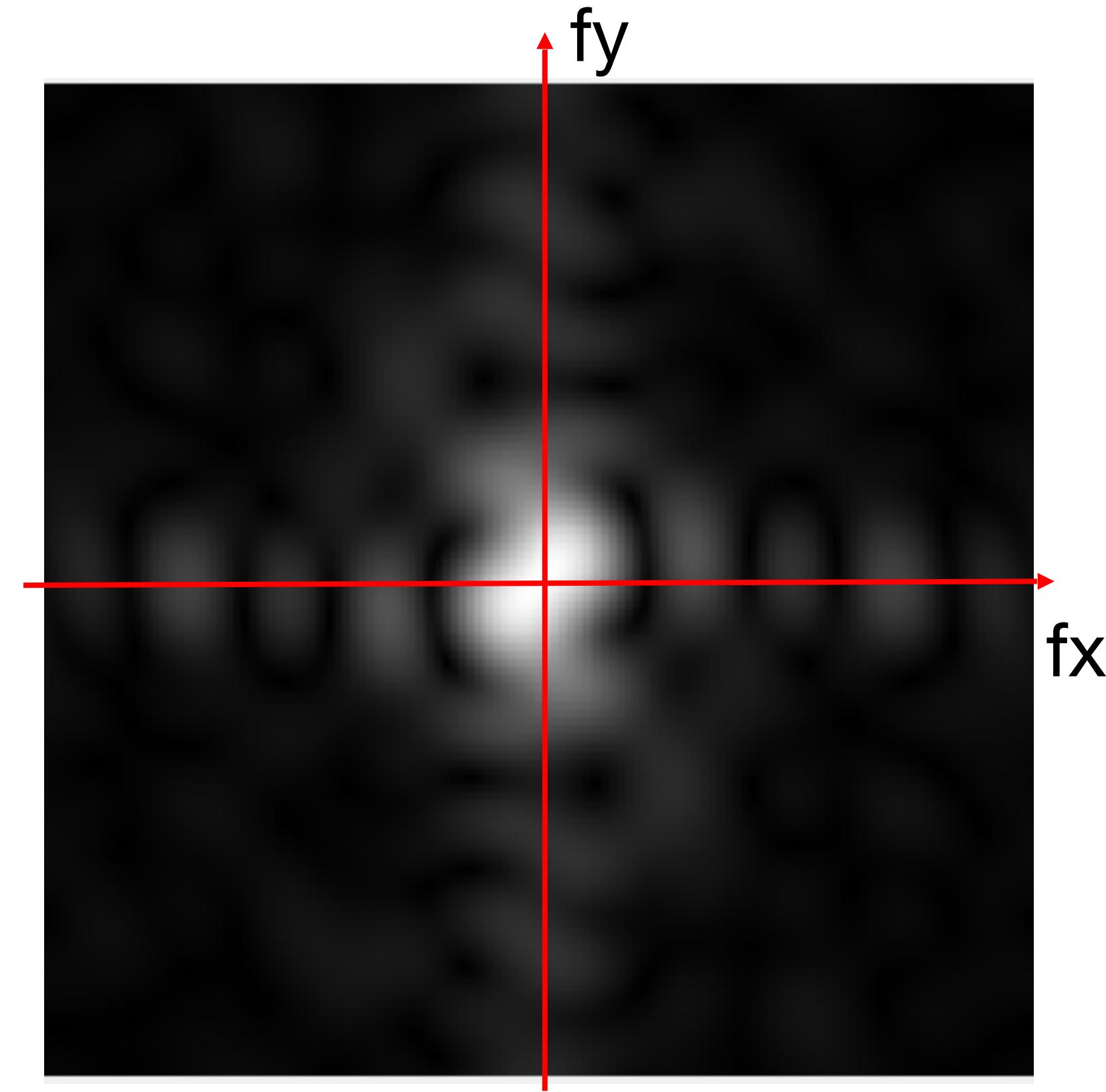
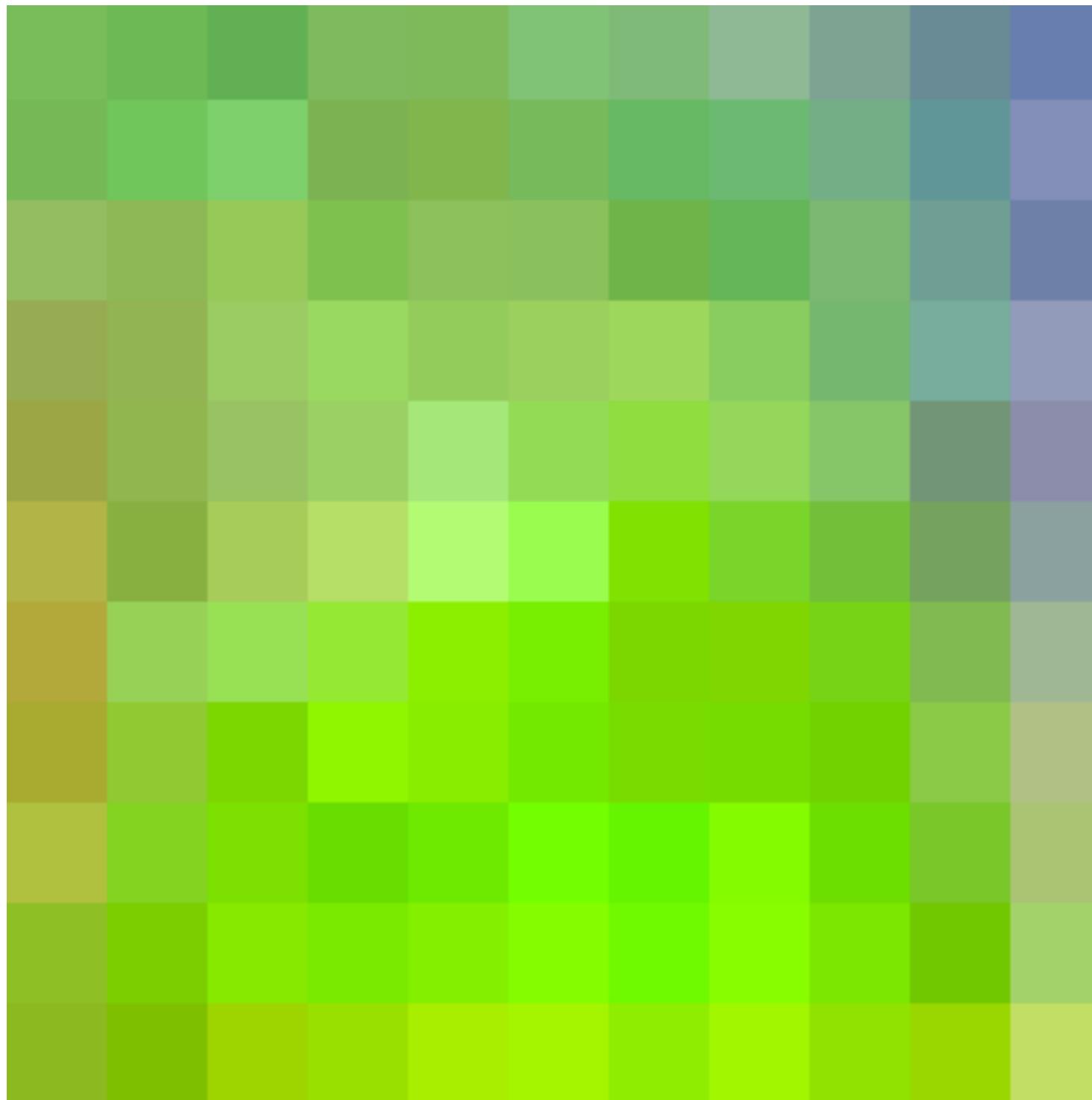
# Get to know your units



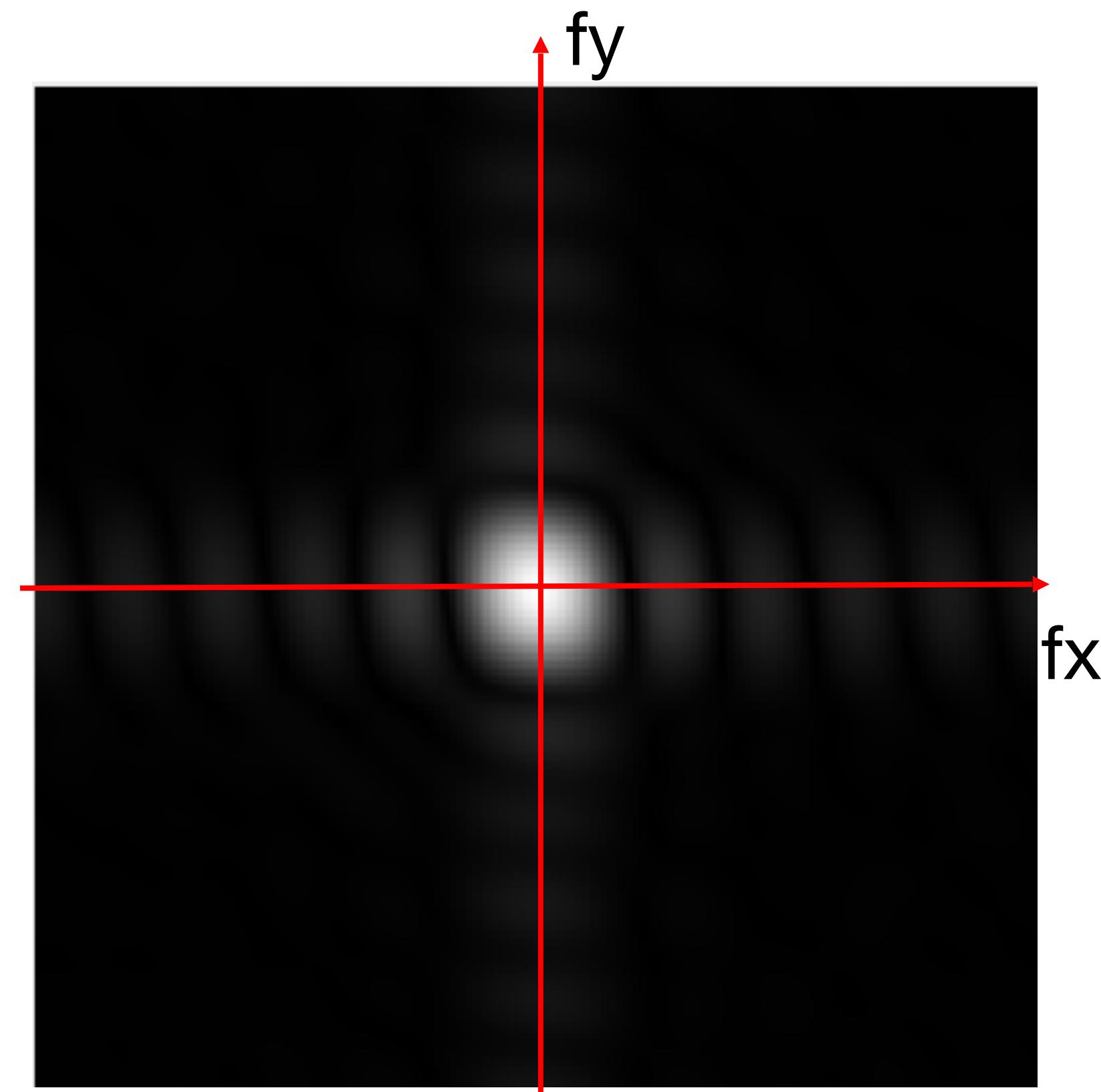
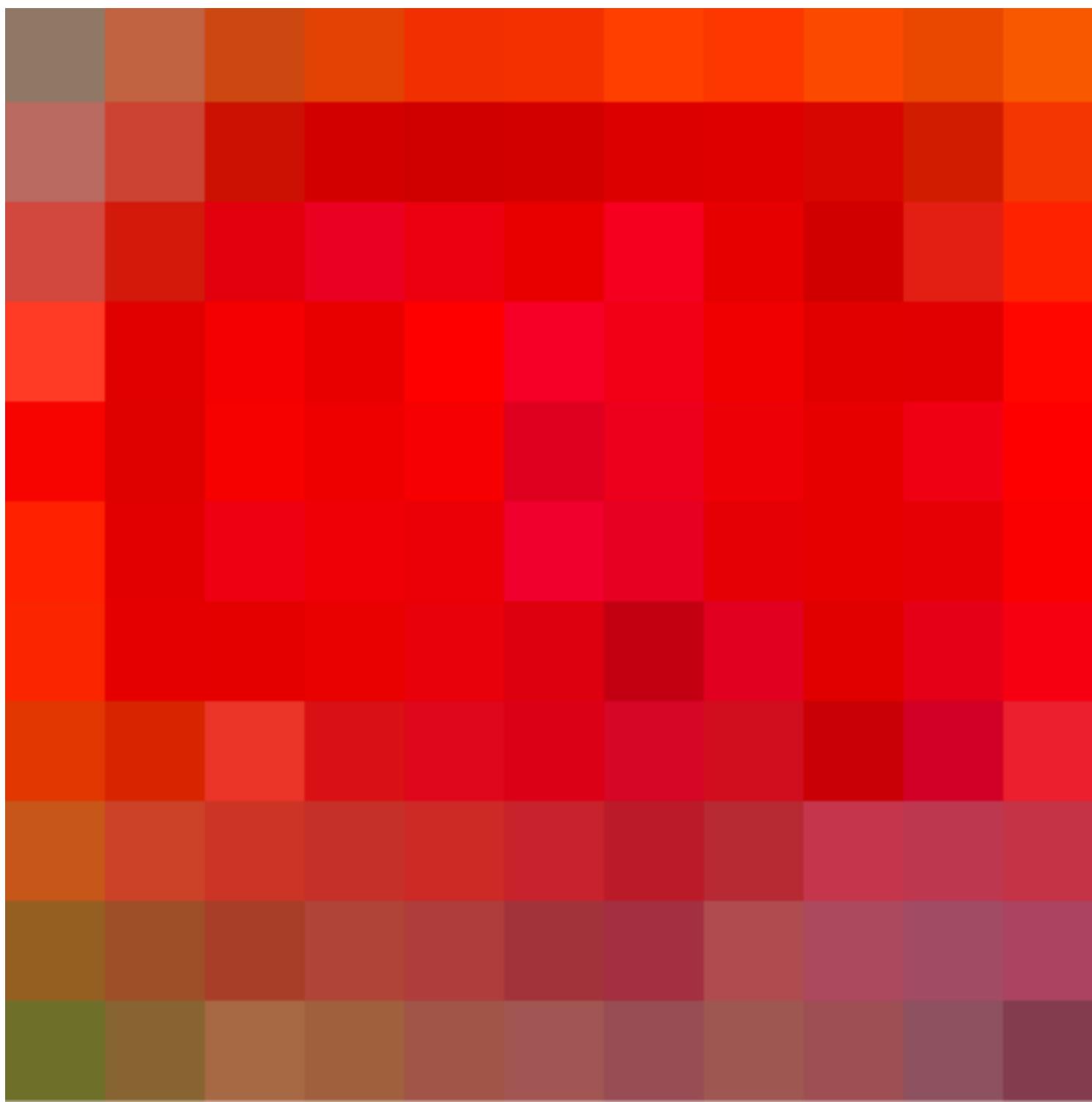
# Get to know your units



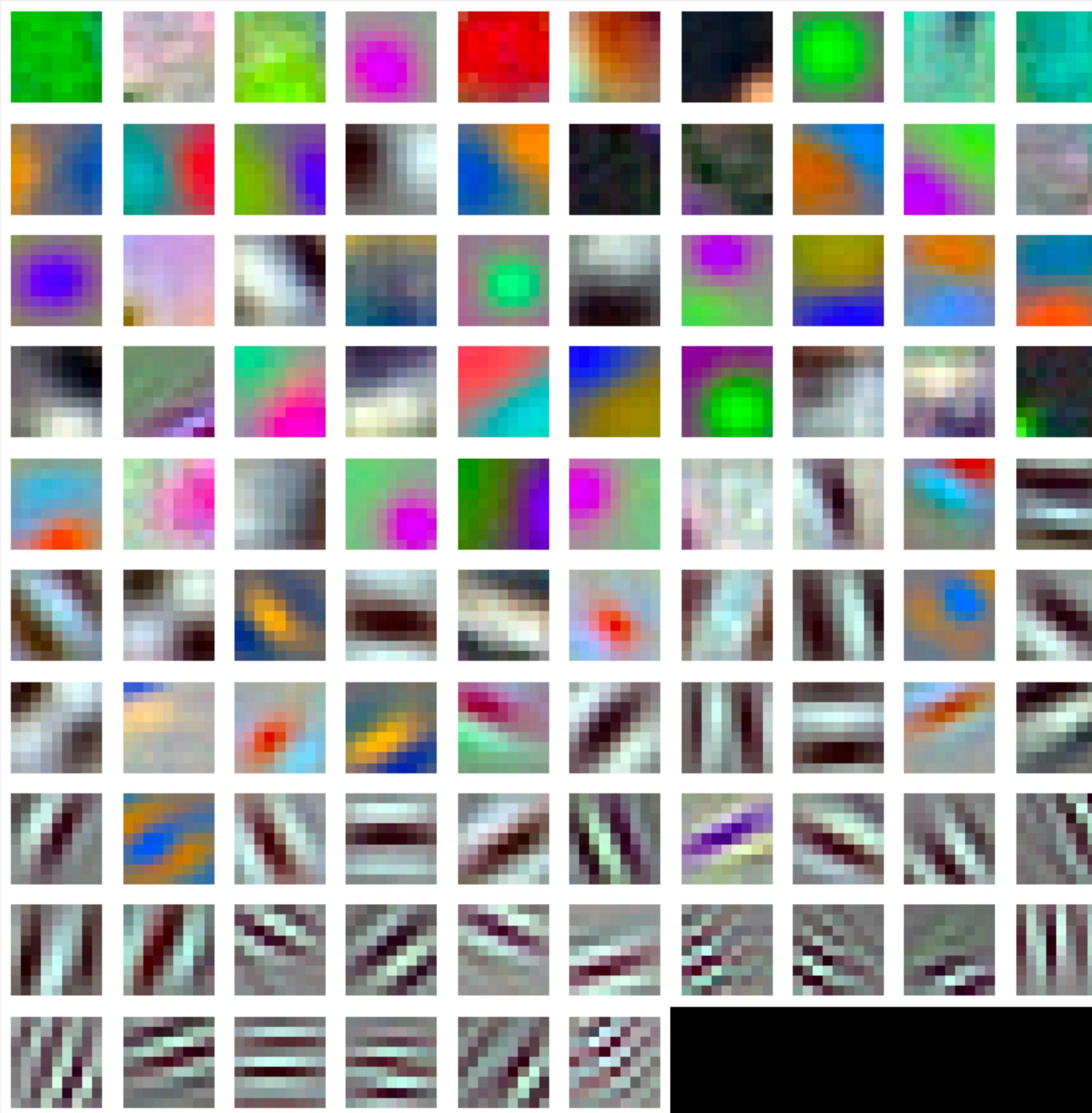
# Get to know your units



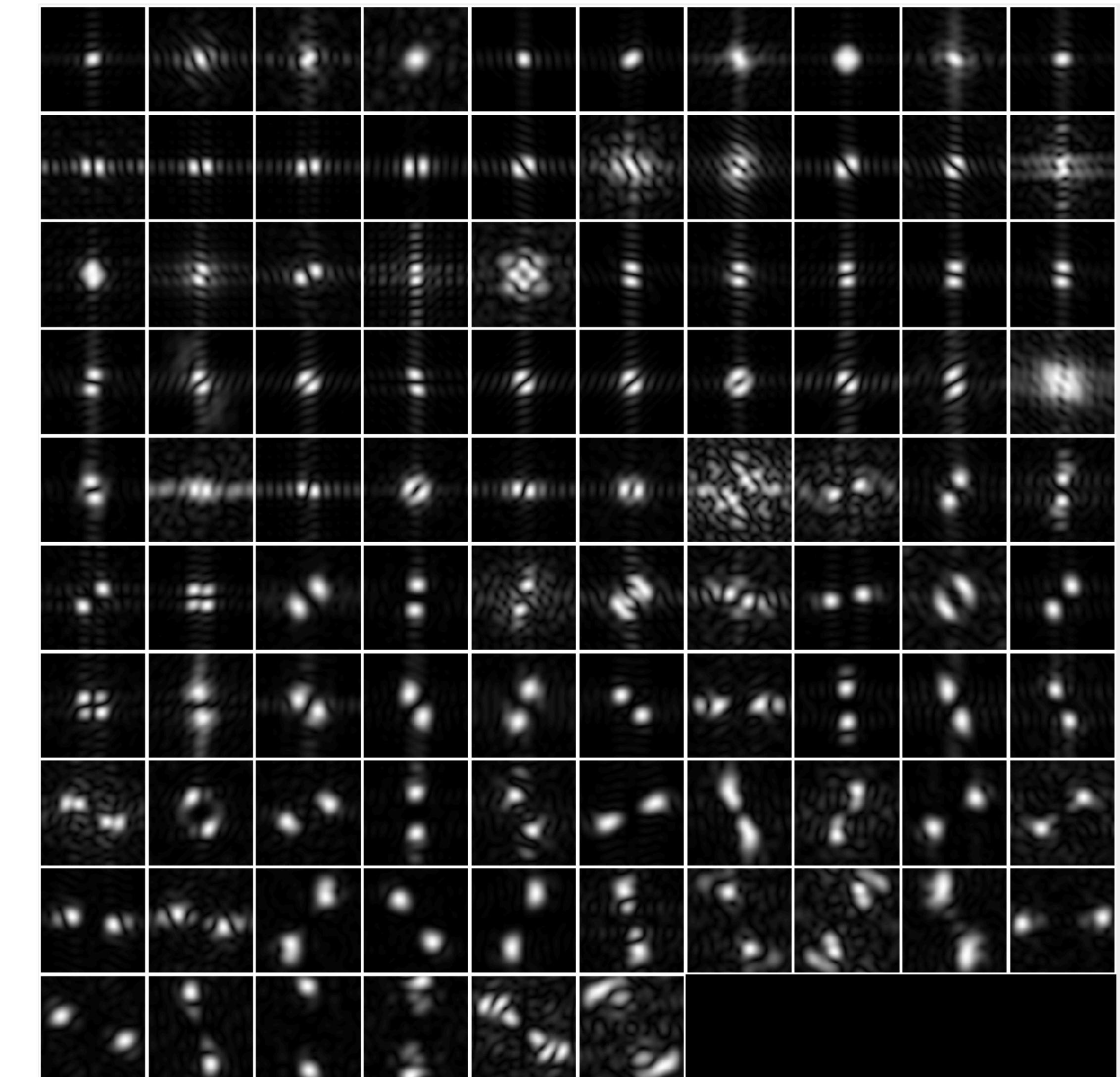
# Get to know your units



# Get to know your units



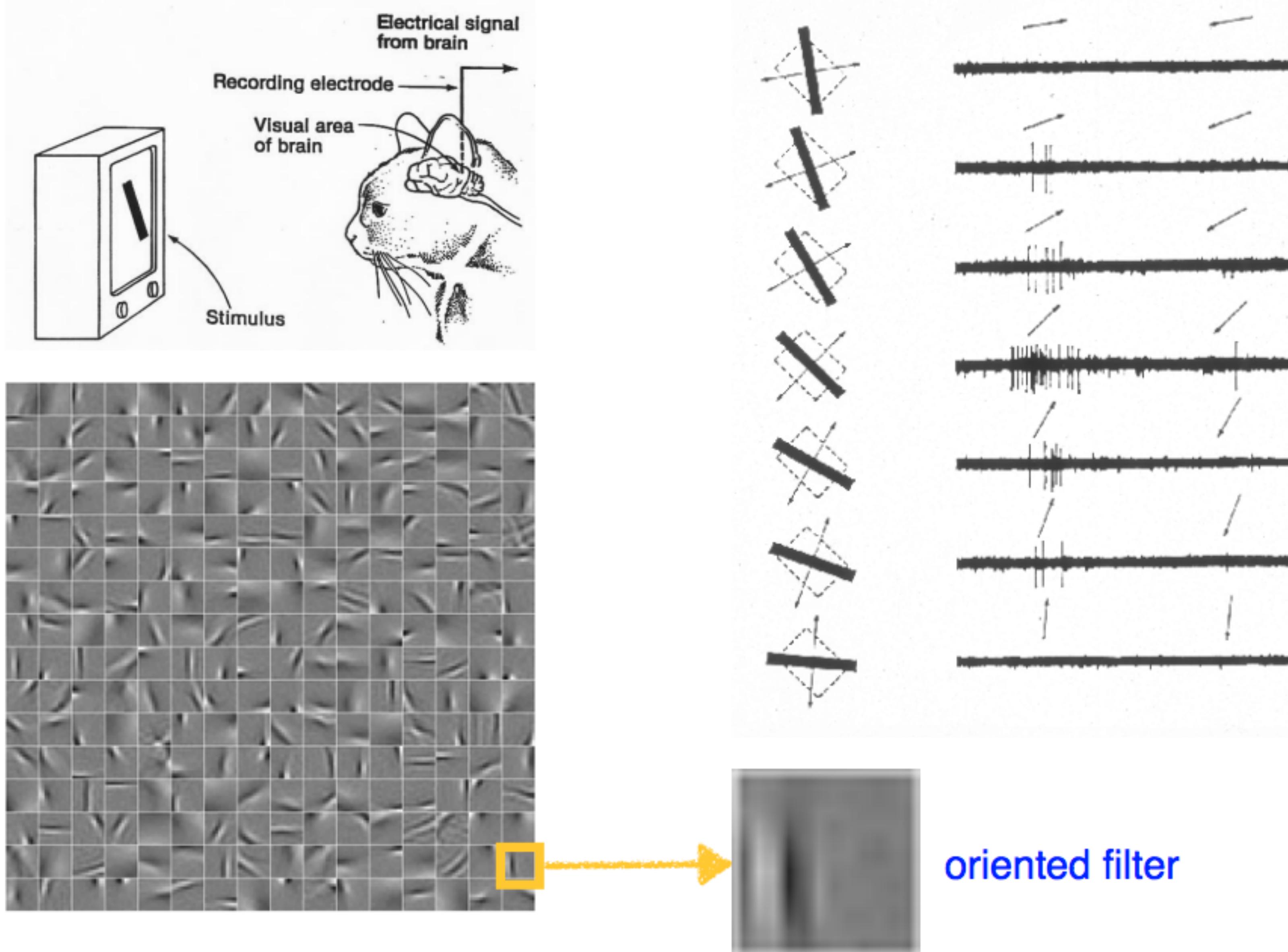
96 Units in conv1



56

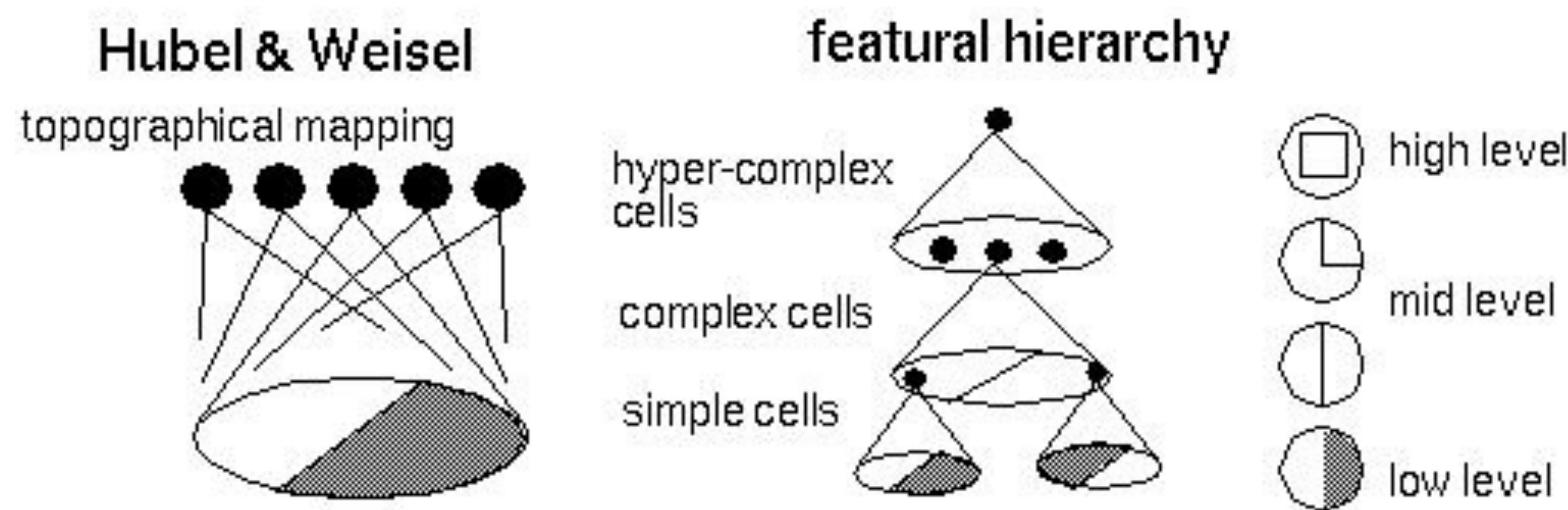
Source: Isola, Torralba, Freeman

## [Hubel and Wiesel 59]

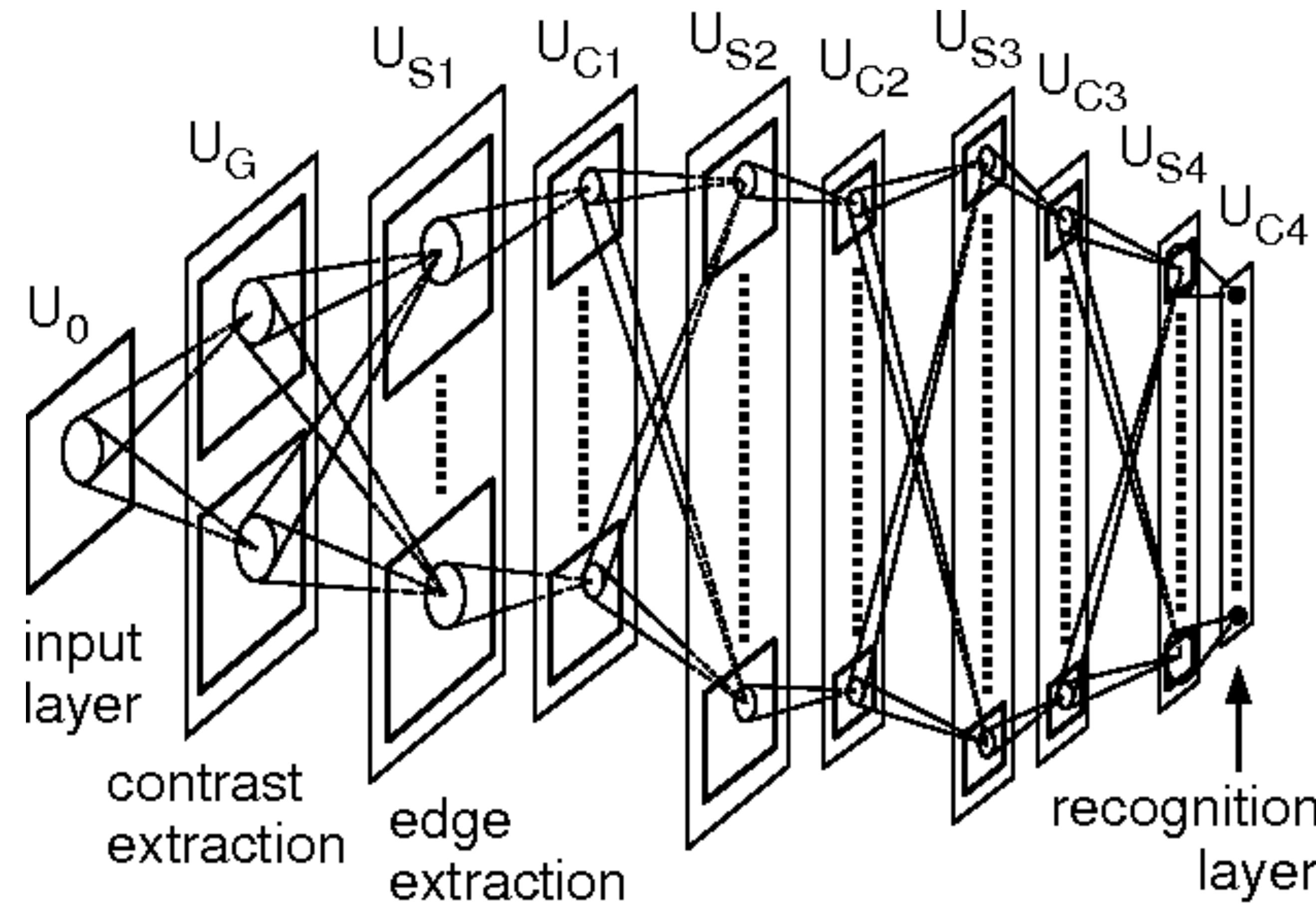


# Biological visual system

- D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)
  - Visual cortex consists of a hierarchy of *simple*, *complex*, and *hyper-complex* cells



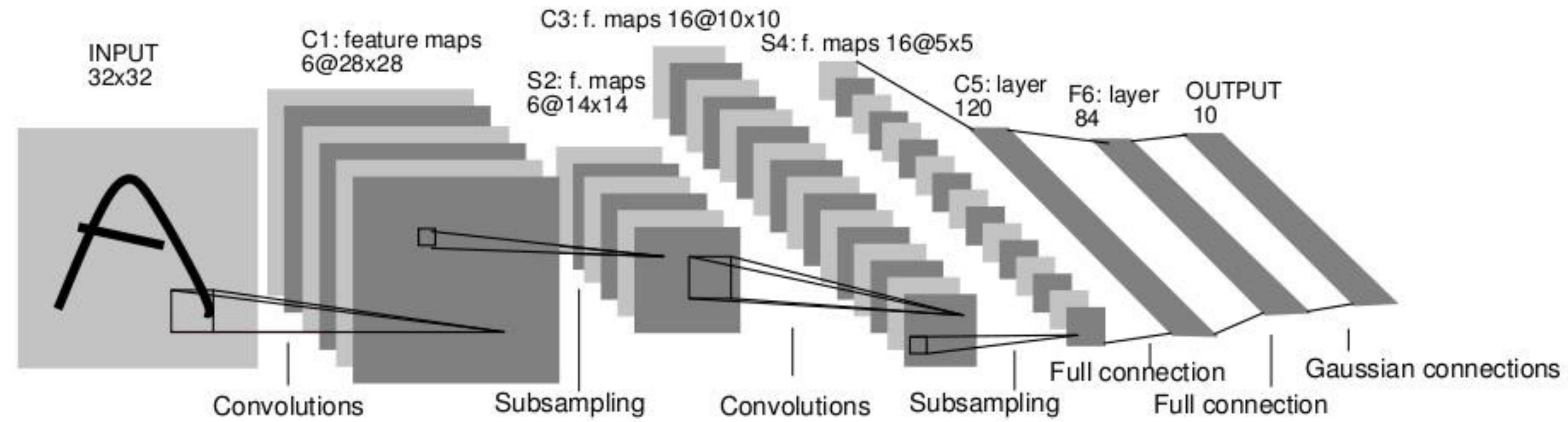
# History: Neocognitron



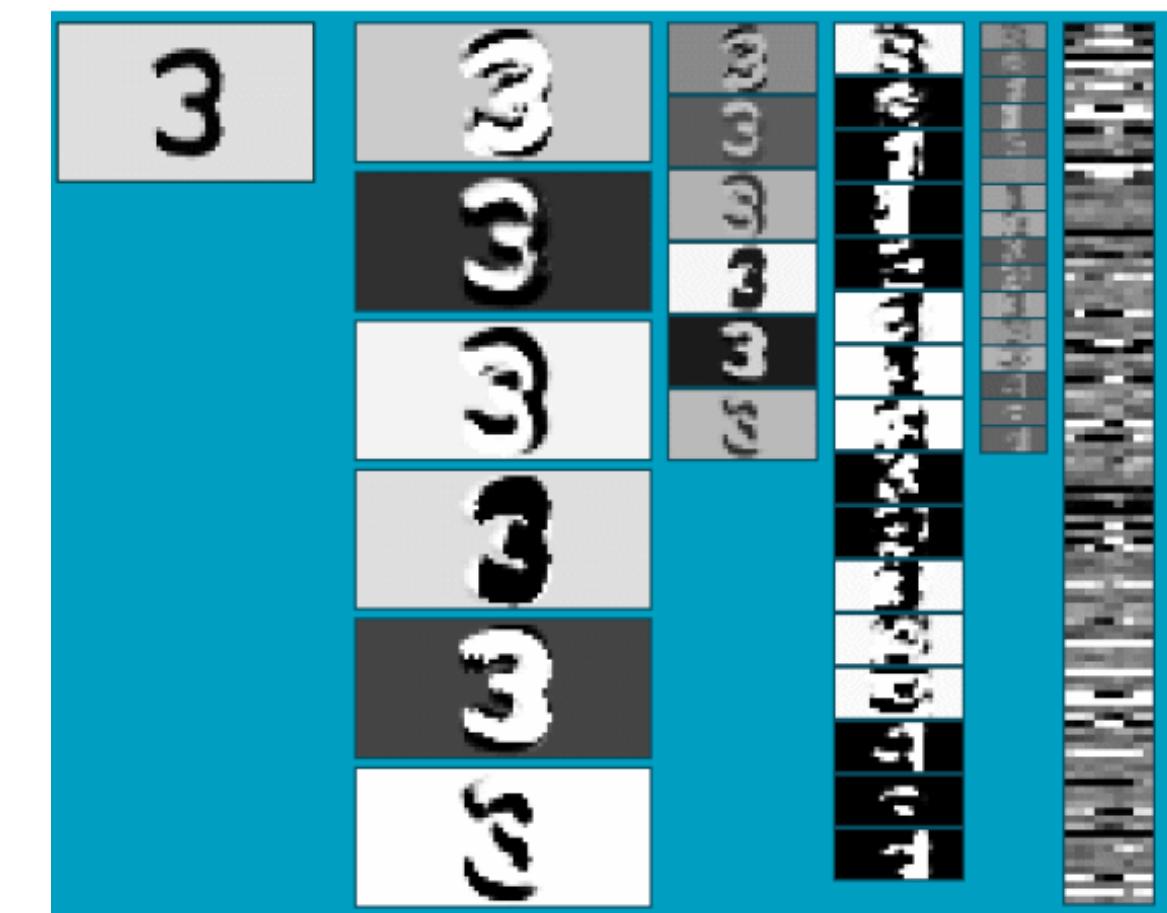
K. Fukushima, 1980s

<https://en.wikipedia.org/wiki/Neocognitron>

# History: LeNet-5



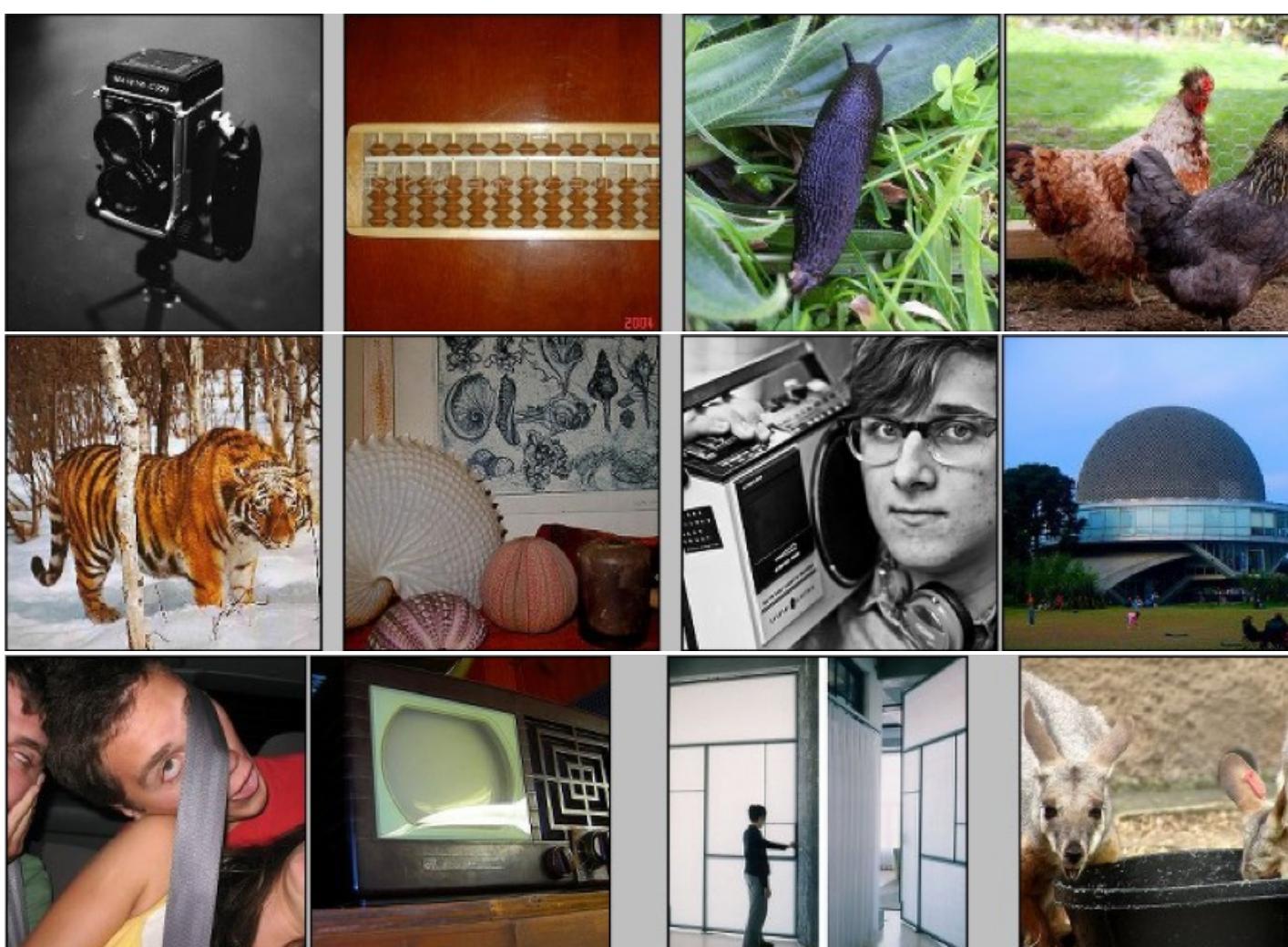
- Neocognitron + backpropagation
- Average pooling
- Sigmoid or tanh nonlinearity
- Fully connected layers at the end
- Trained on MNIST digit dataset with 60K training examples



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, [Gradient-based learning applied to document recognition](#), Proc. IEEE 86(11): 2278–2324, 1998.

Source: S. Lazebnik

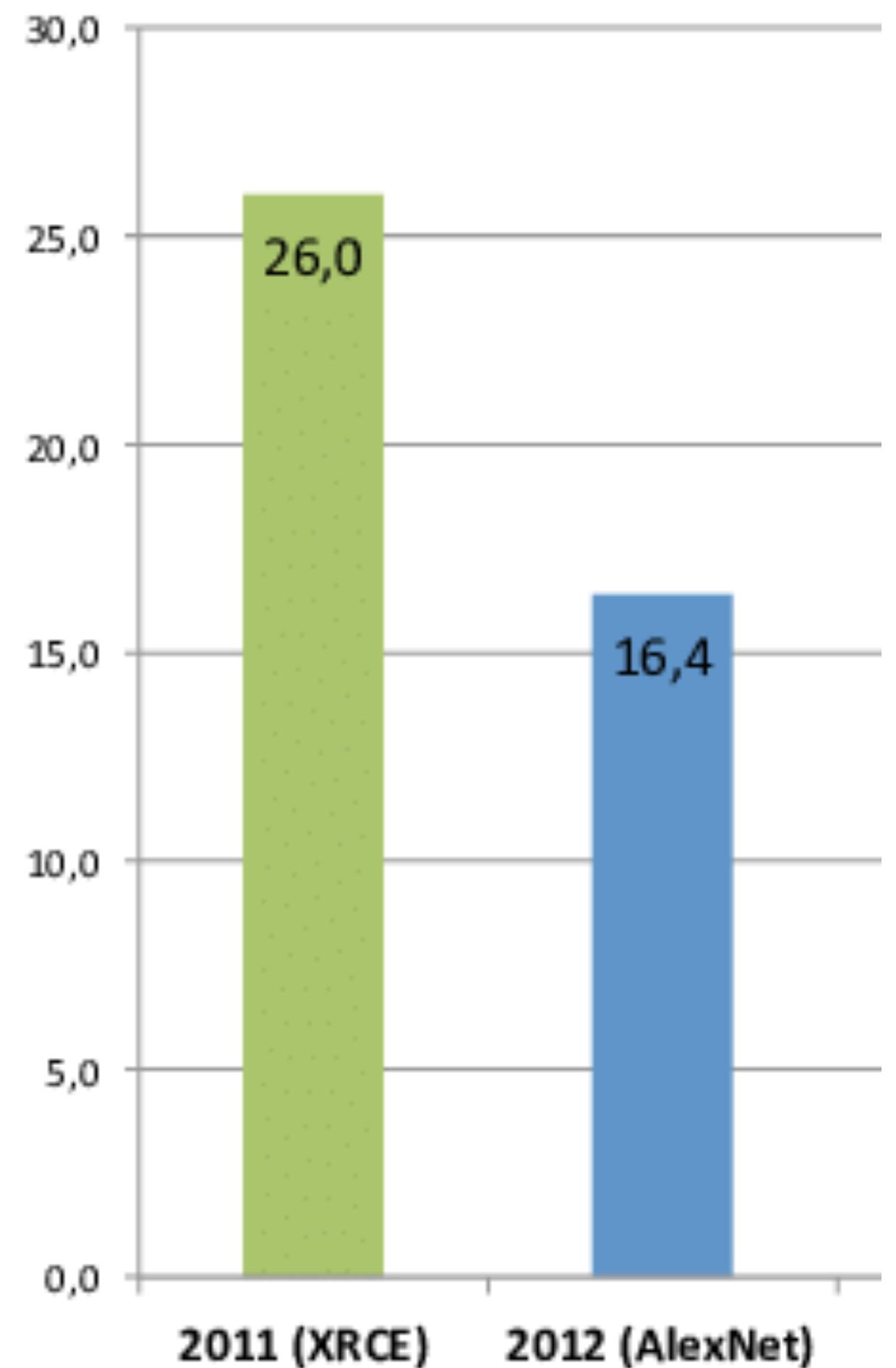
# ImageNet Challenge



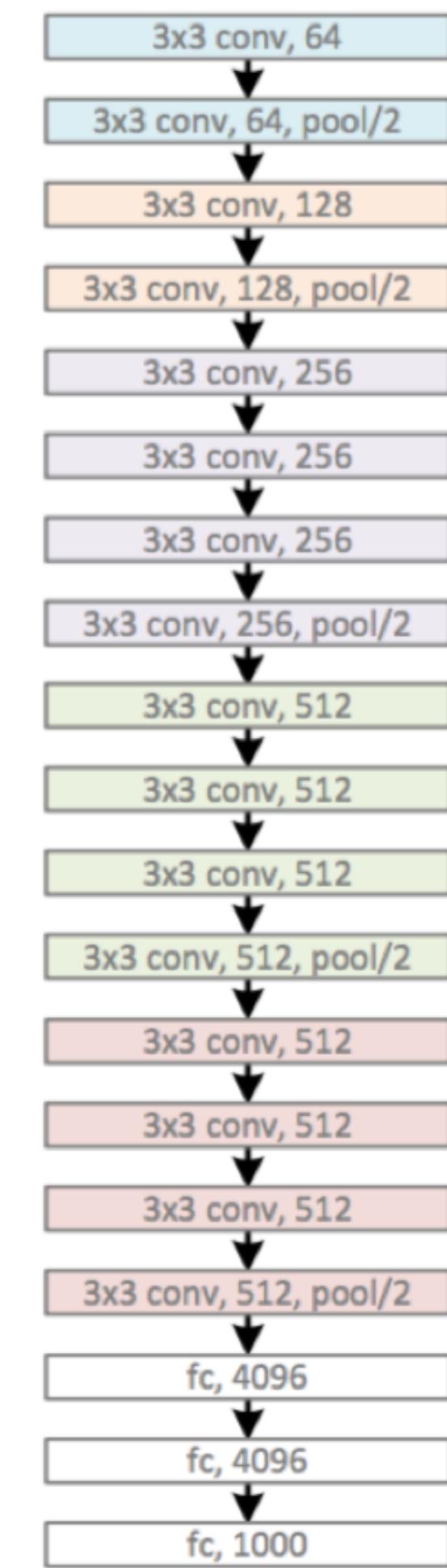
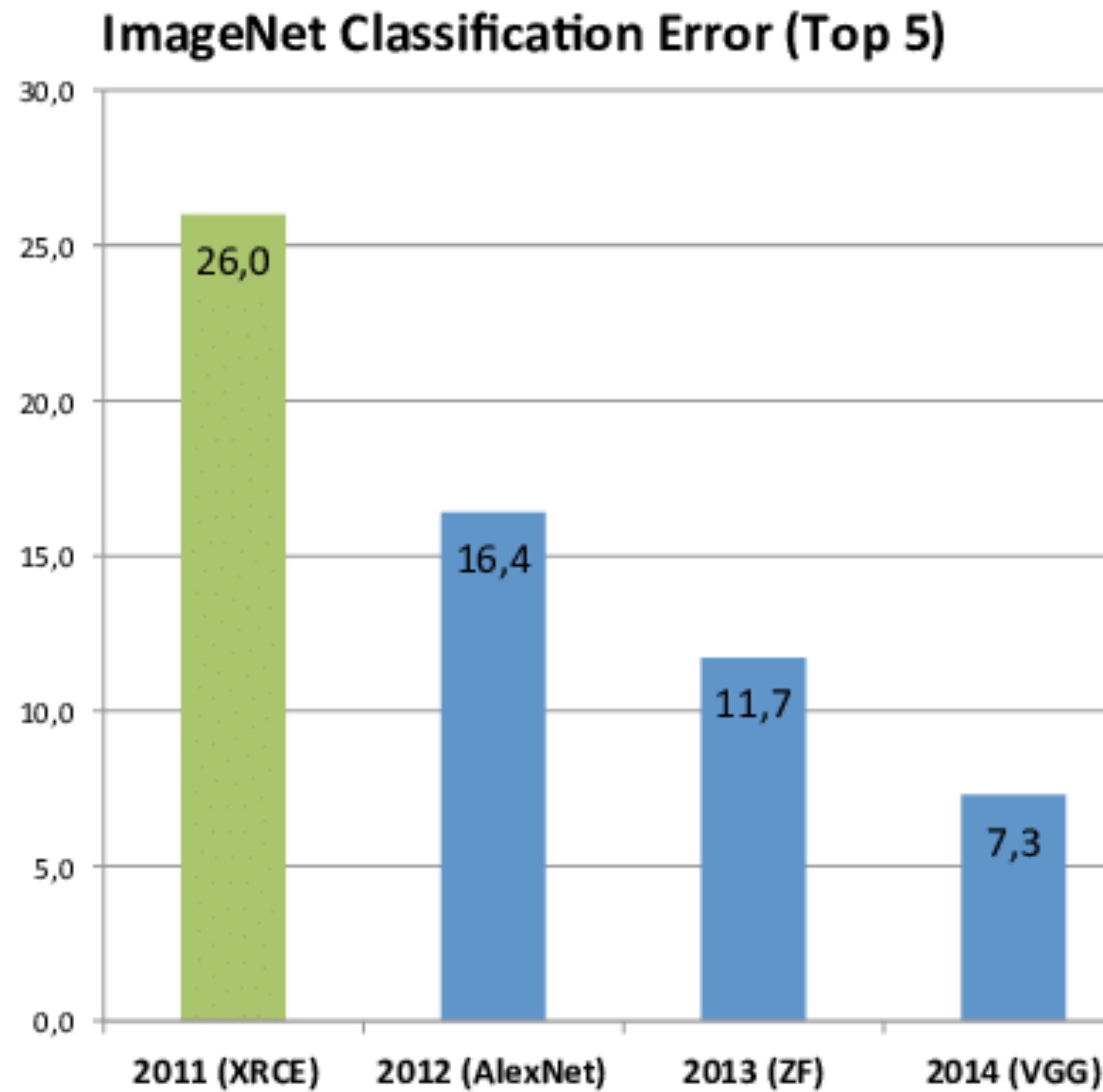
- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC):  
1.2 million training images, 1000 classes

[www.image-net.org/challenges/LSVRC/](http://www.image-net.org/challenges/LSVRC/)

## ImageNet Classification Error (Top 5)



2014: VGG  
16 conv. layers

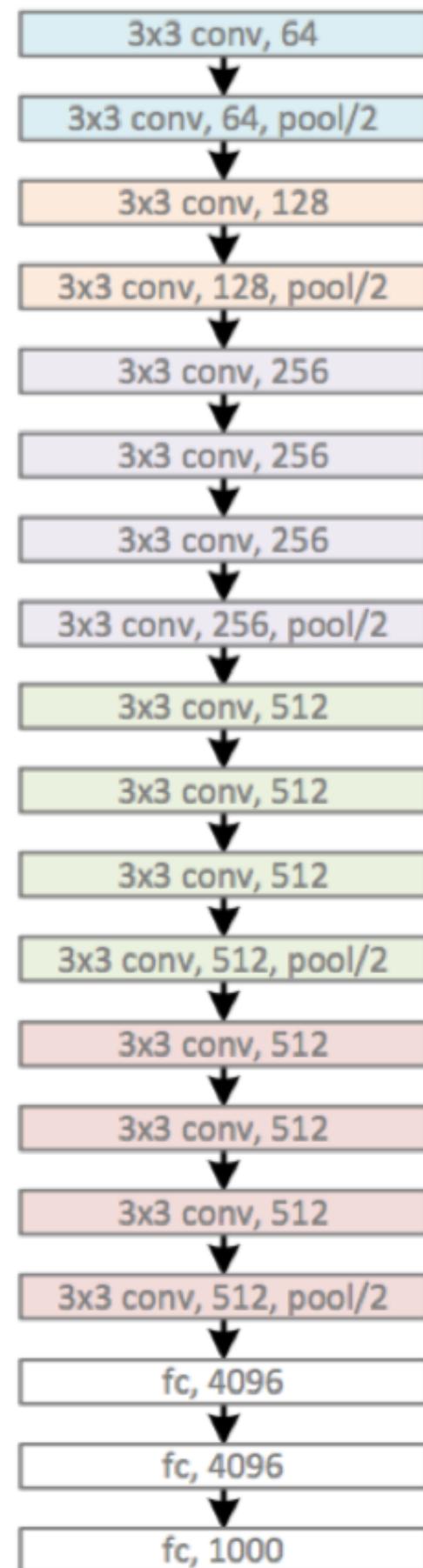


Error: 7.3%

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

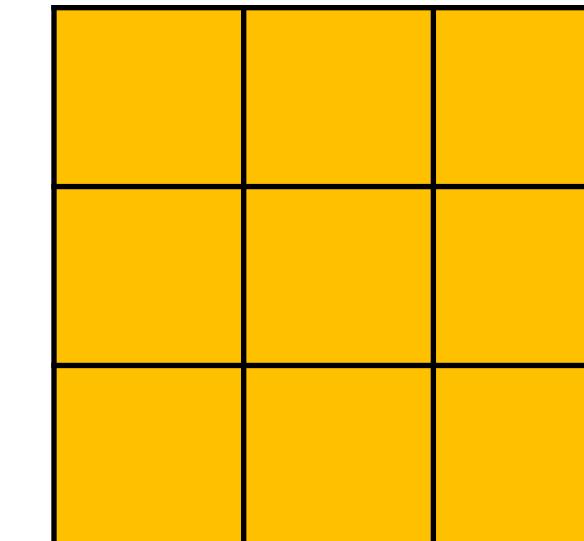
# VGG-Net [Simonyan & Zisserman, 2015]

2014: VGG  
16 conv. layers



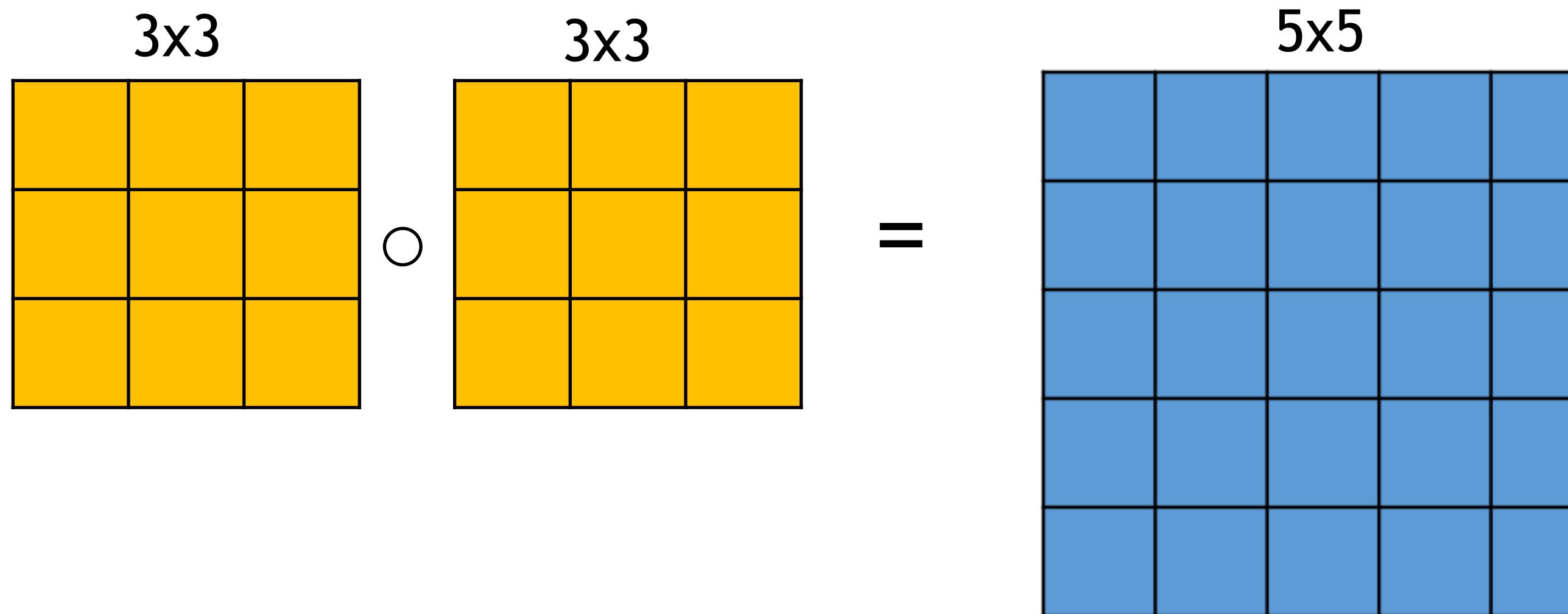
## Main developments

- Small convolutional kernels: only 3x3
- Increased depth (5 -> 16/19 layers)

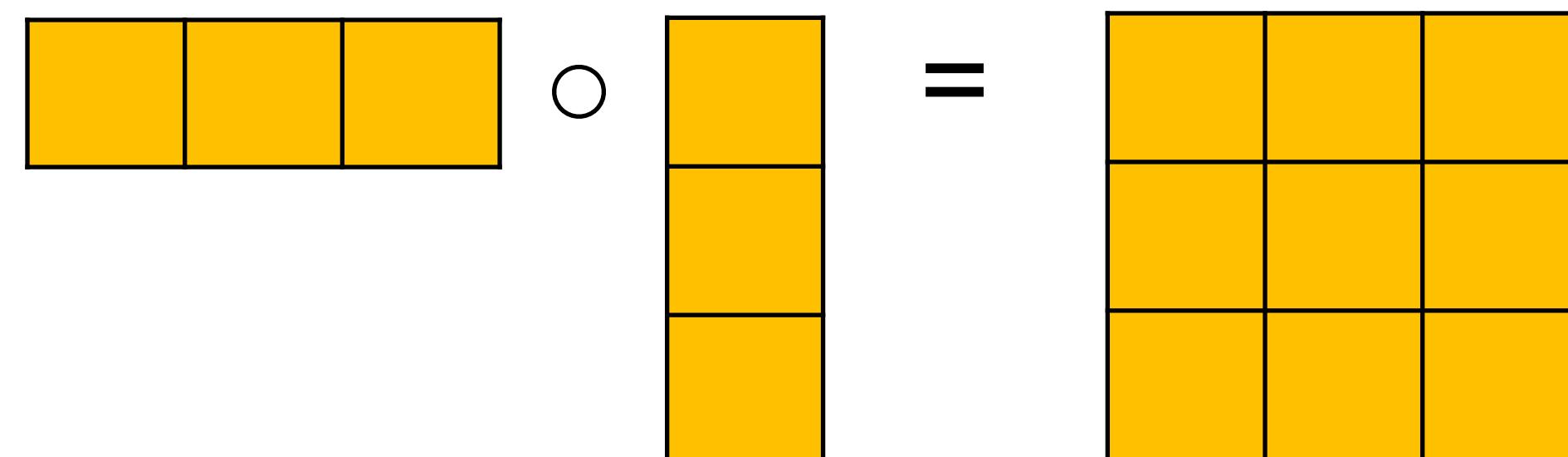


Error: 7.3%

# Chaining convolutions

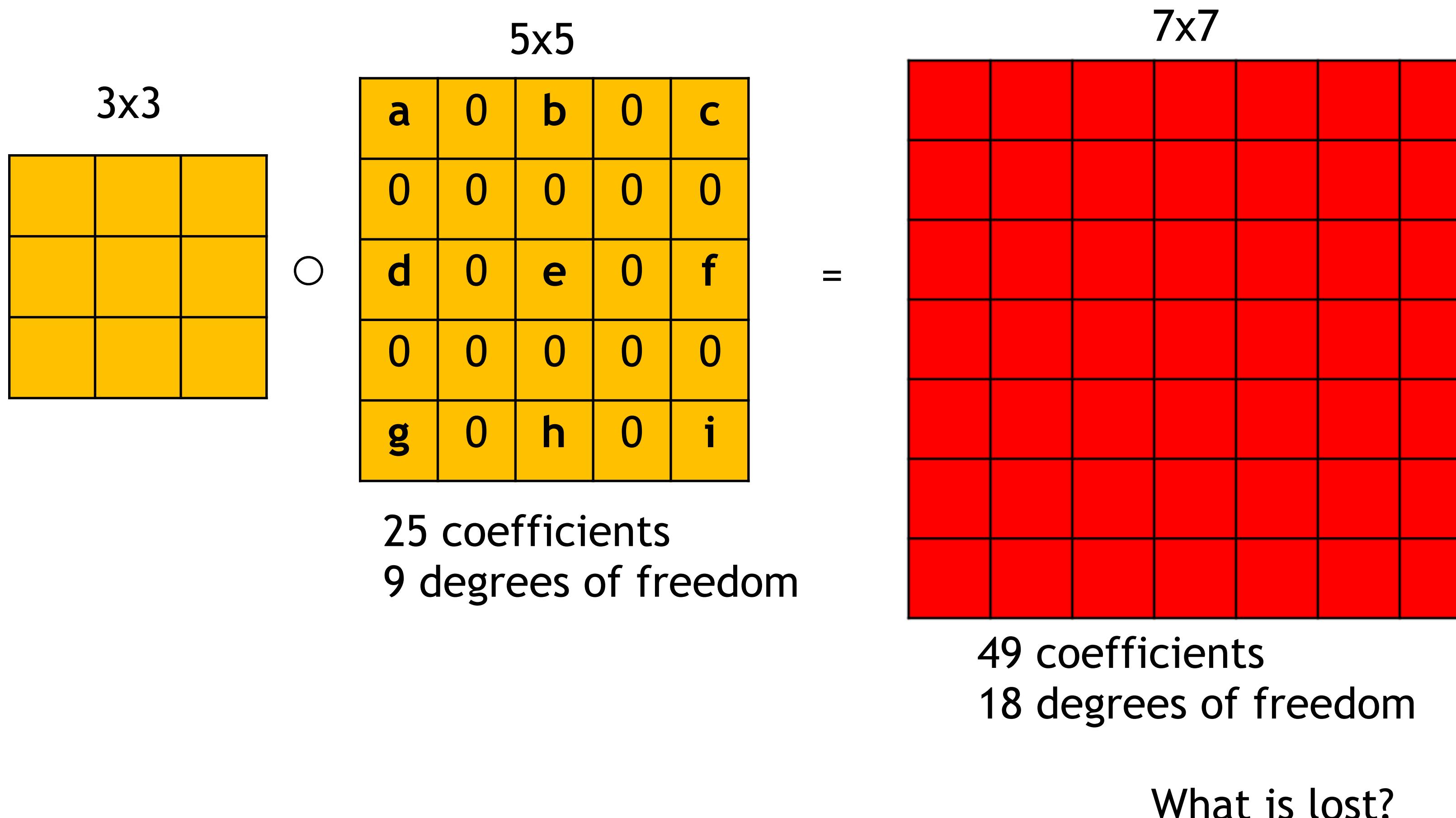


25 coefficients, but only  
18 degrees of freedom

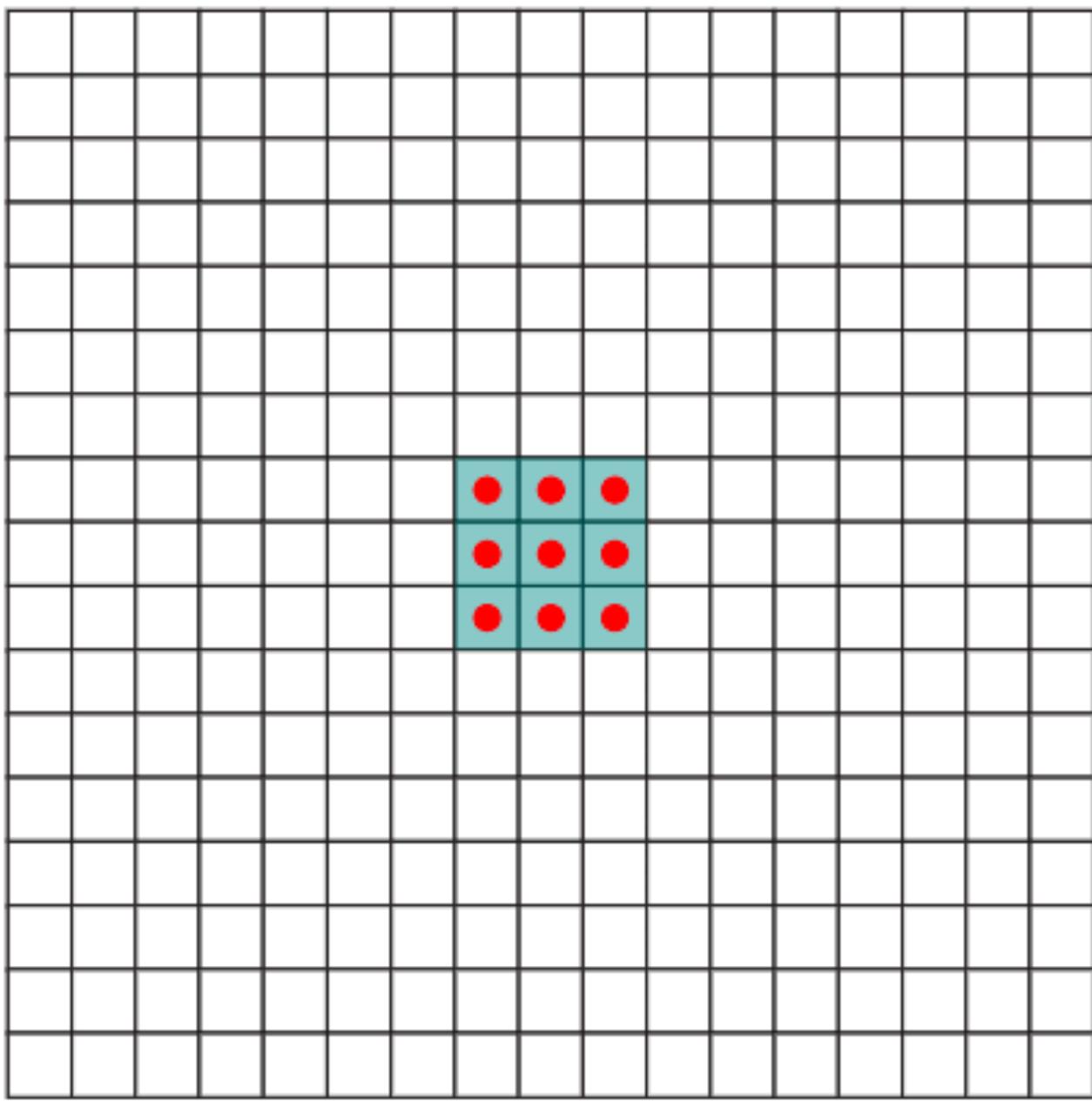


9 coefficients, but only  
6 degrees of freedom.  
Only separable filters... would this be enough?

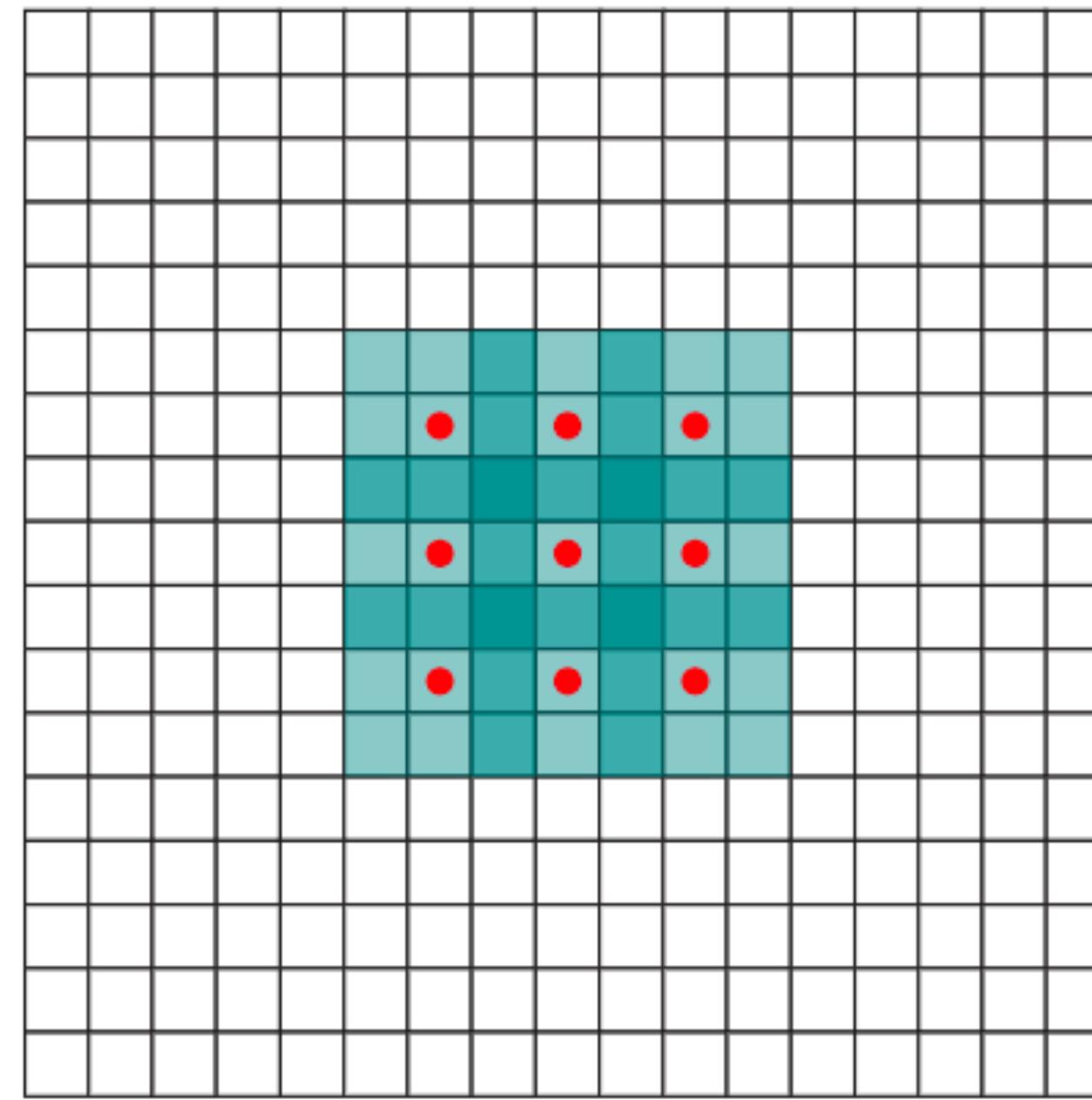
# Dilated convolutions



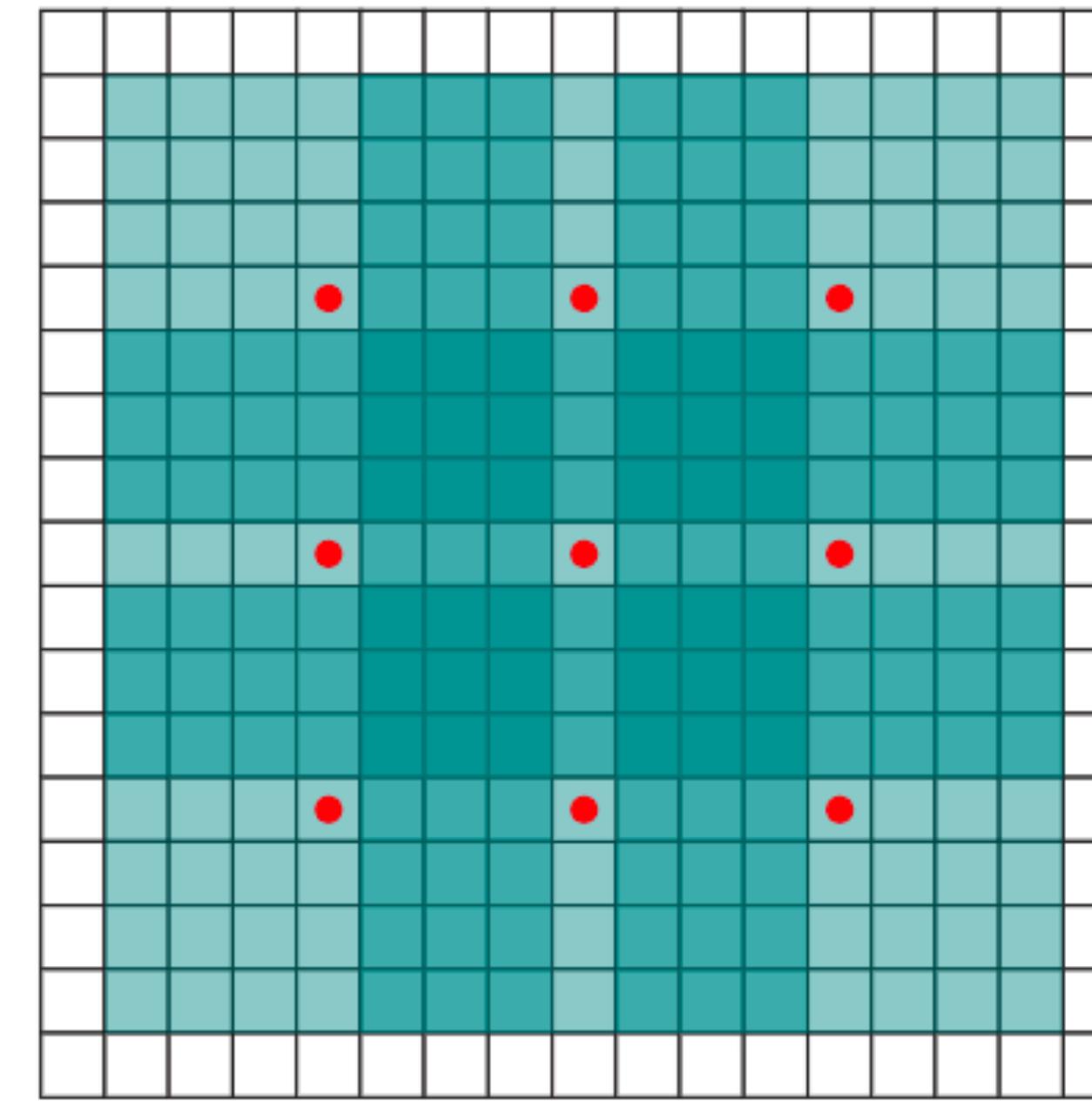
[<https://arxiv.org/pdf/1511.07122.pdf>]



(a)



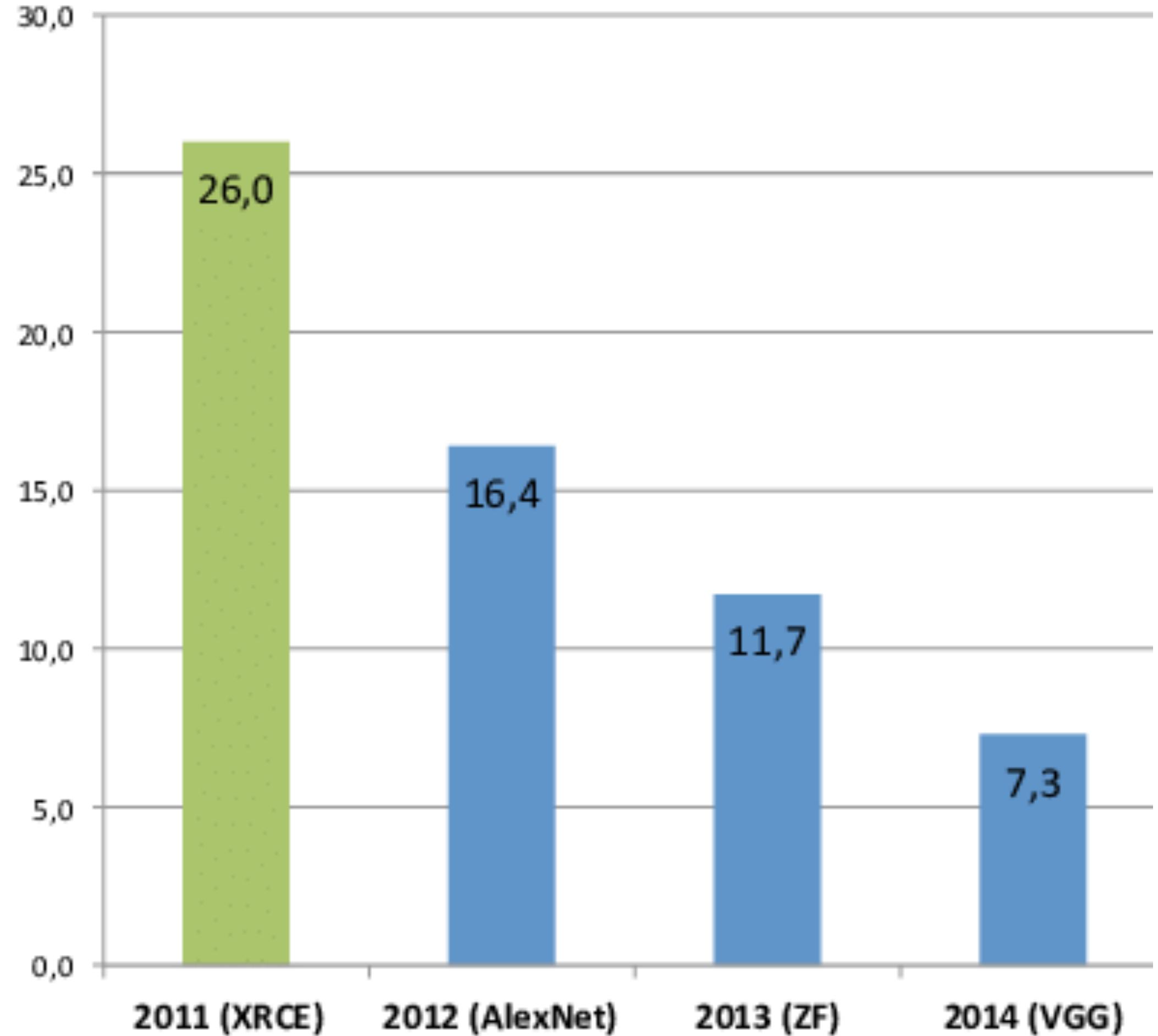
(b)



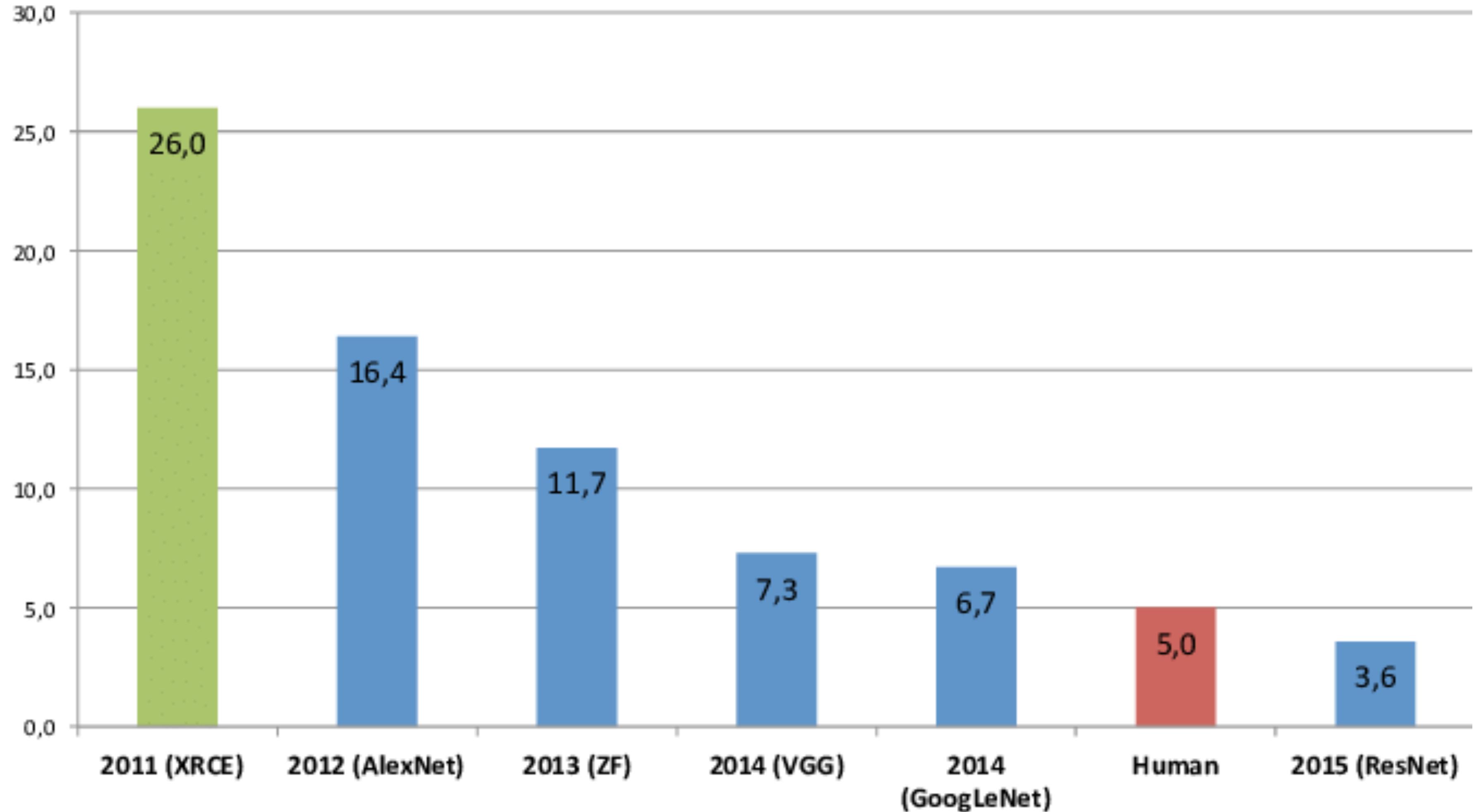
(c)

Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a)  $F_1$  is produced from  $F_0$  by a 1-dilated convolution; each element in  $F_1$  has a receptive field of  $3 \times 3$ . (b)  $F_2$  is produced from  $F_1$  by a 2-dilated convolution; each element in  $F_2$  has a receptive field of  $7 \times 7$ . (c)  $F_3$  is produced from  $F_2$  by a 4-dilated convolution; each element in  $F_3$  has a receptive field of  $15 \times 15$ . The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

## ImageNet Classification Error (Top 5)



## ImageNet Classification Error (Top 5)



2016: ResNet  
>100 conv. layers

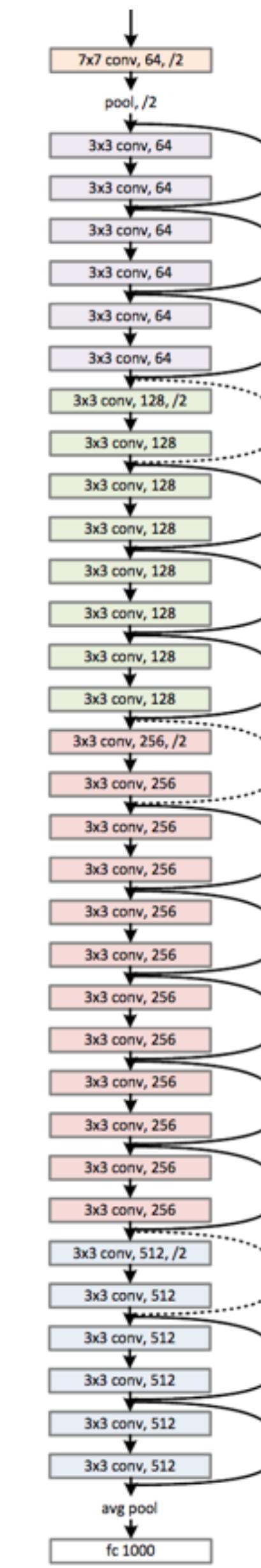
Error: 3.6%

[He et al: Deep Residual Learning for Image Recognition, CVPR 2016]  
69

2016: ResNet  
>100 conv. layers



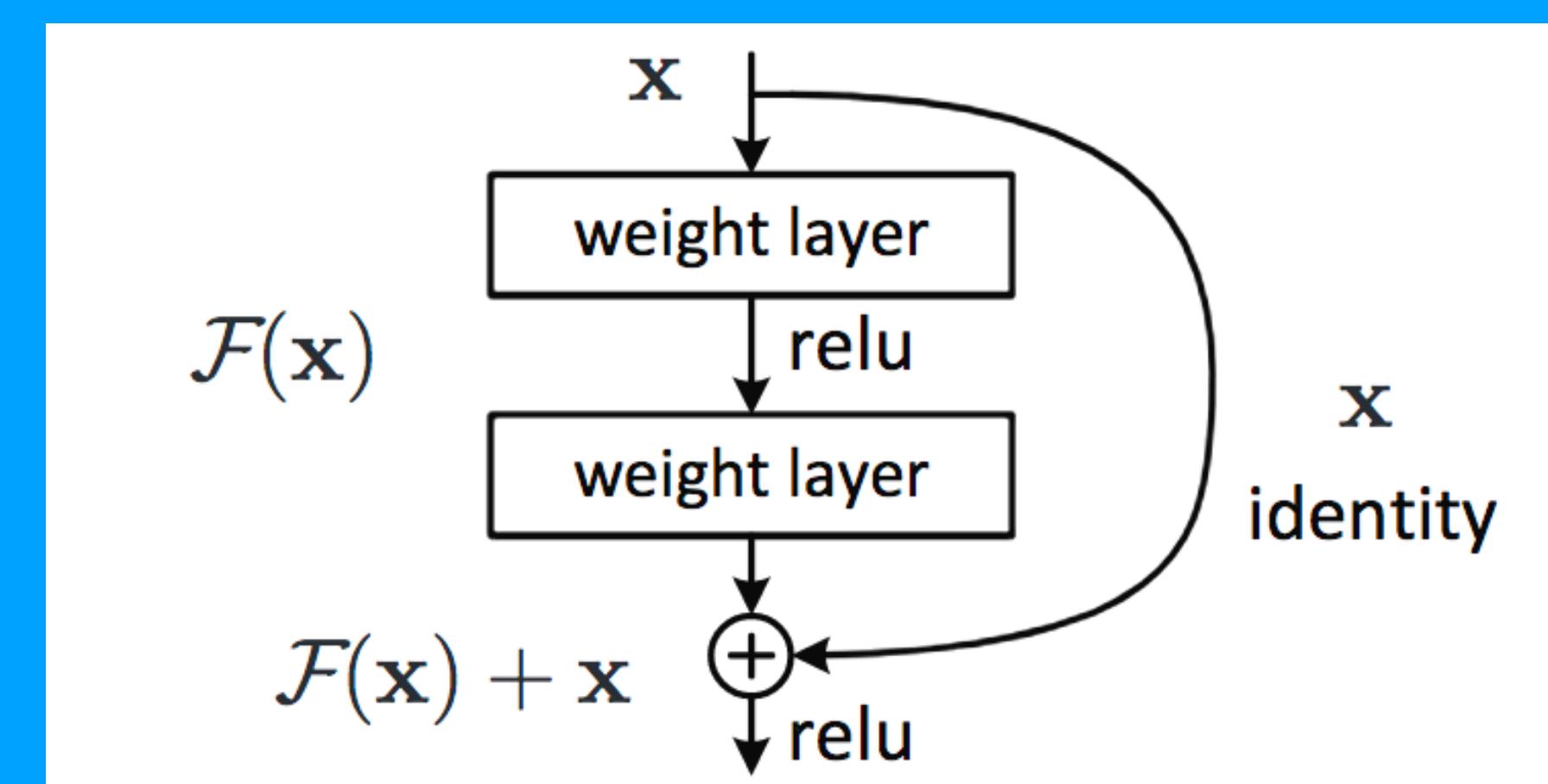
Error: 3.6%



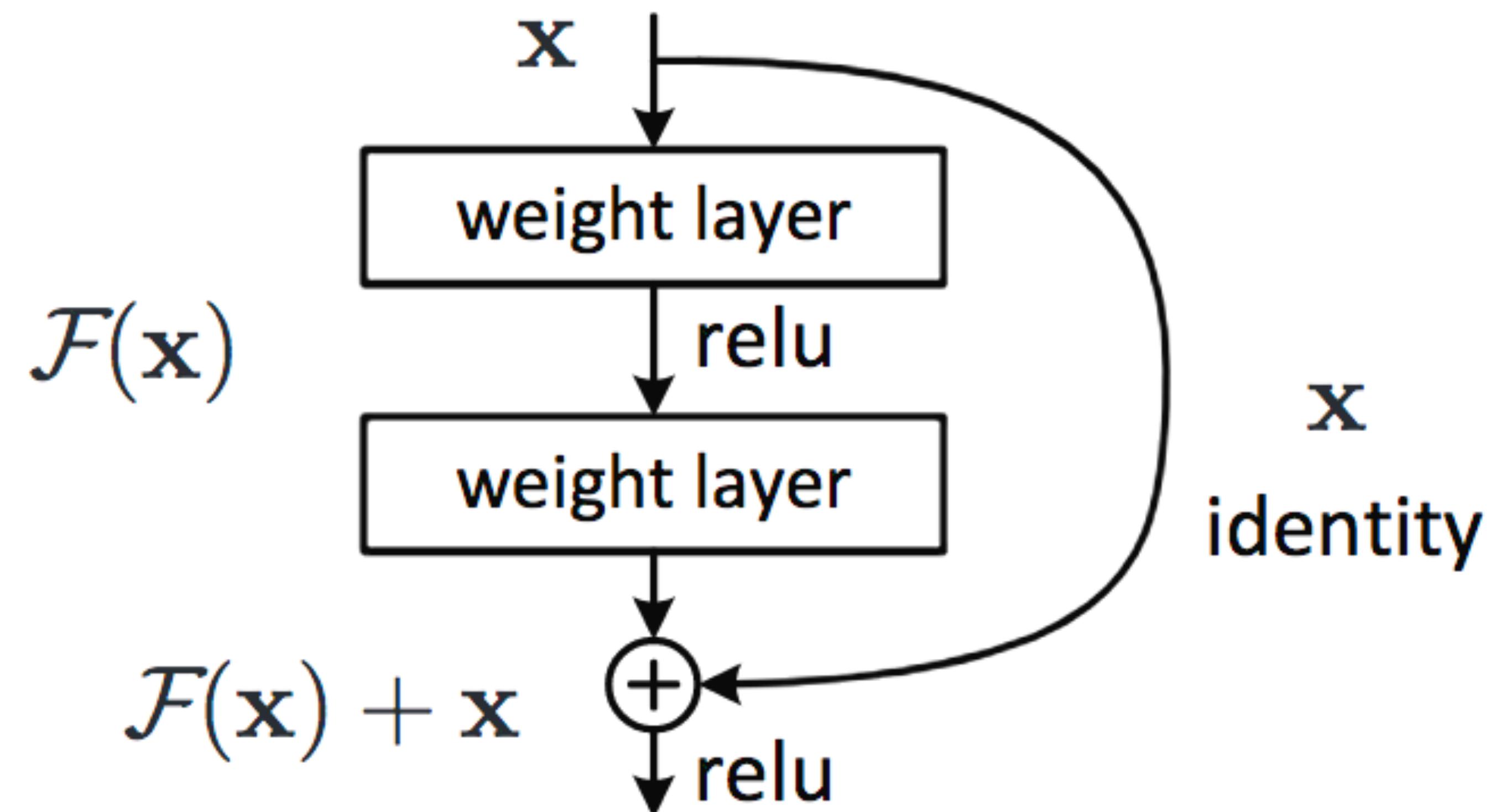
# ResNet [He et al, 2016]

## Main developments

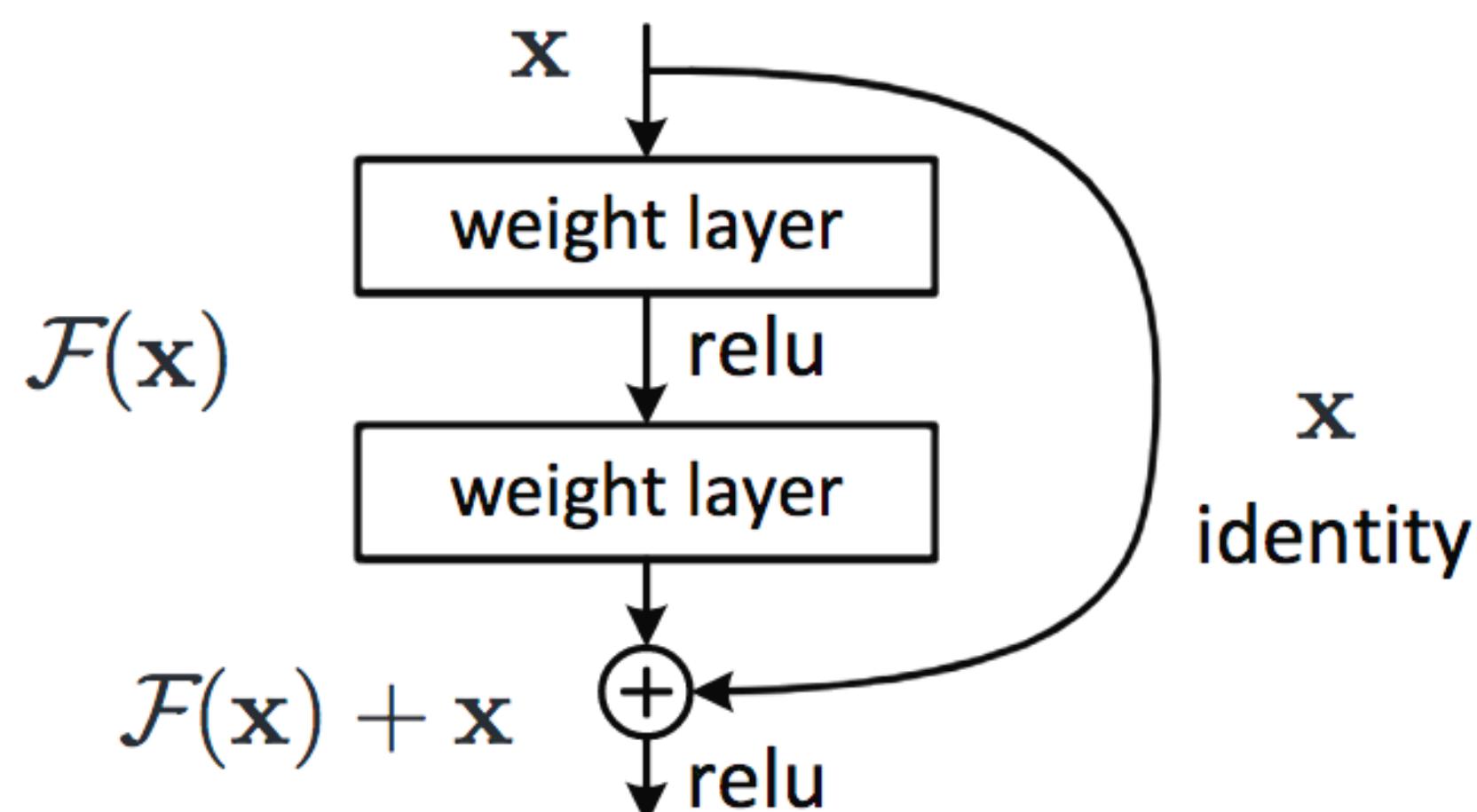
- Increased depth possible through residual blocks



# Residual Blocks



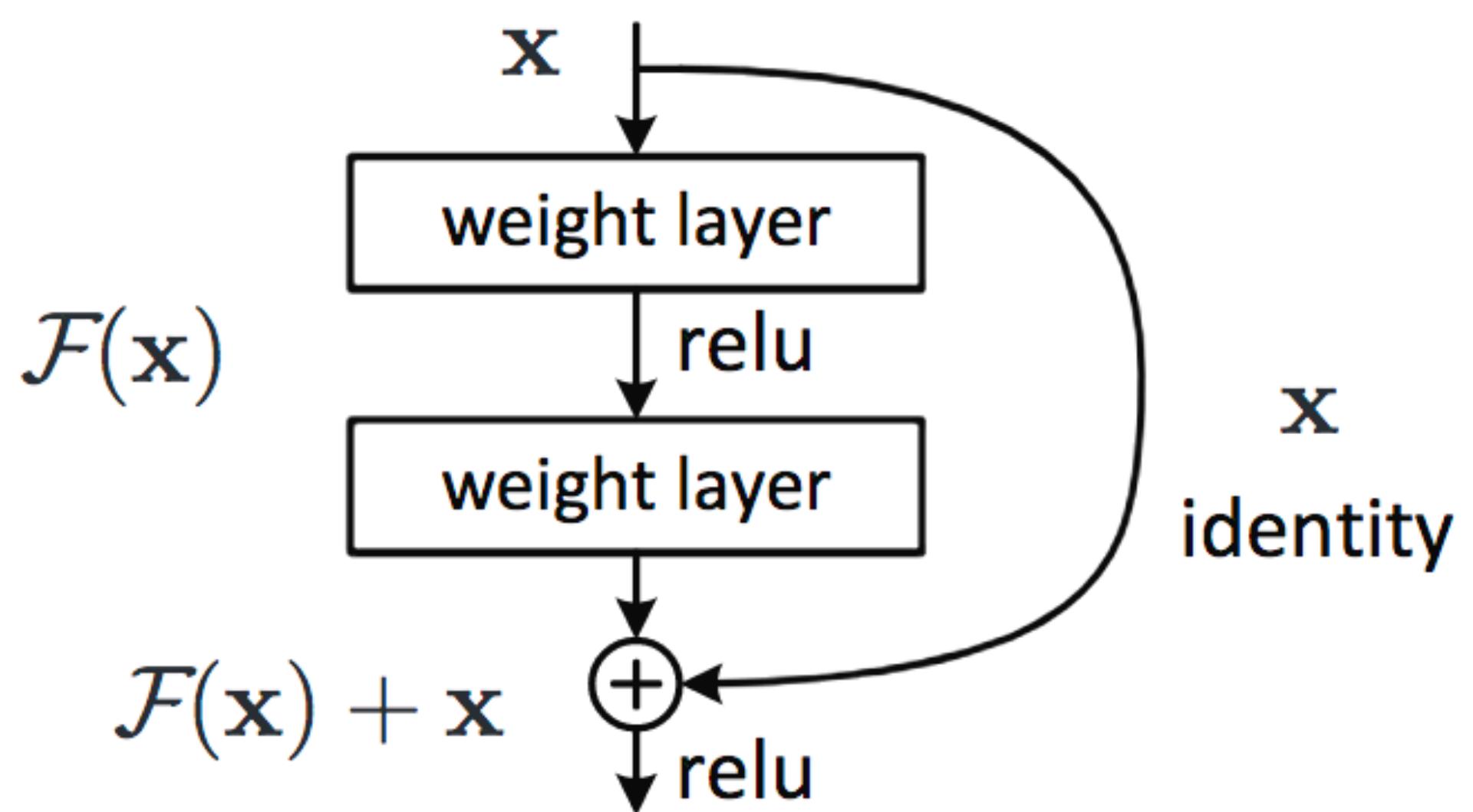
# Residual Blocks



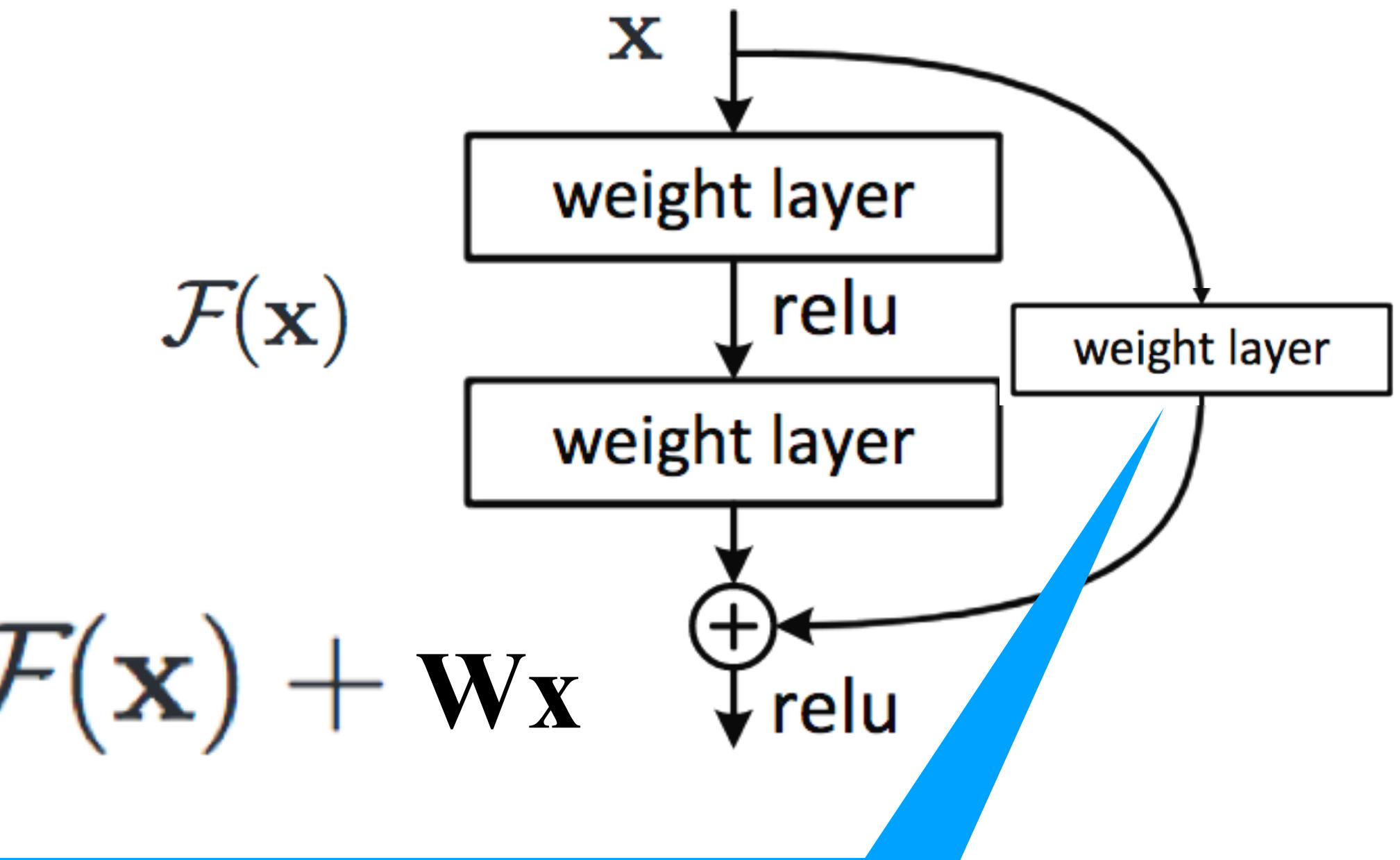
## Why do they work?

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

If output has same size as input:

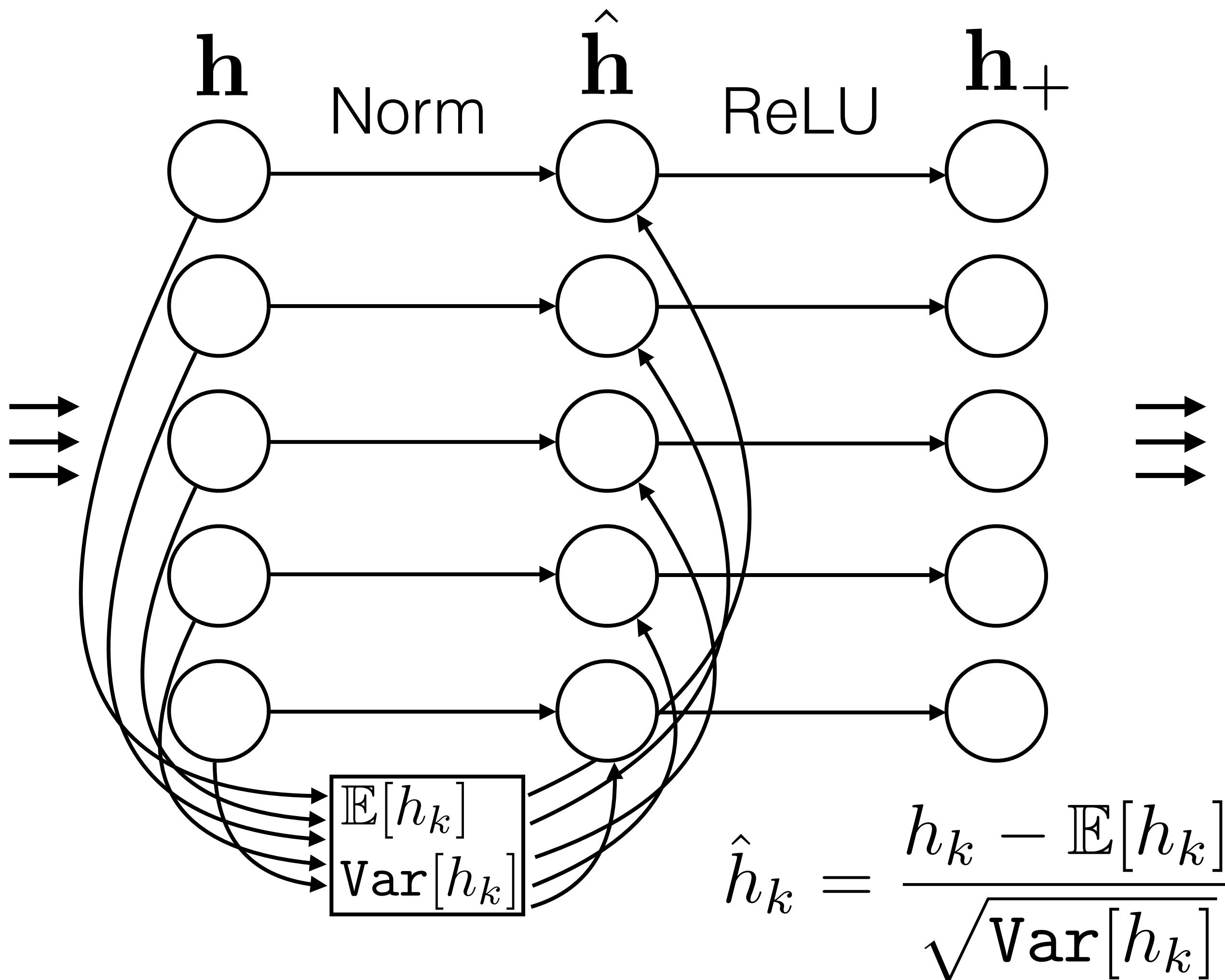


If output has a different size:



Projects into the right  
dimensionality:  
 $\dim(\mathcal{F}(x)) = \dim(Wx)$

# Normalization layers



# Normalization layers

Keep track of mean and variance of a unit (or a population of units) over time.

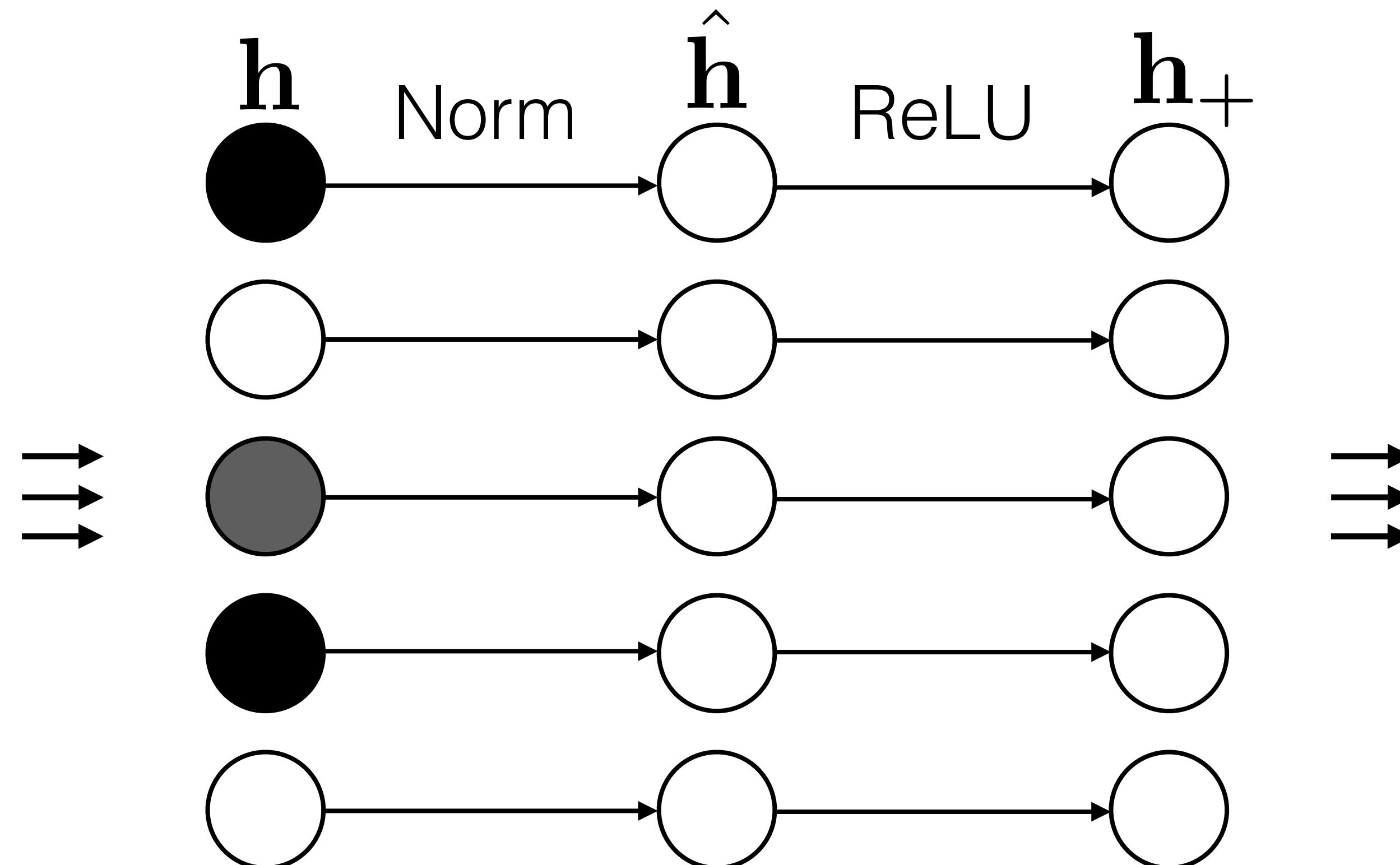
Standardize unit activations by subtracting mean and dividing by variance.

Squashes units into a **standard range**, avoiding overflow.

Also achieves **invariance** to mean and variance of the training signal.

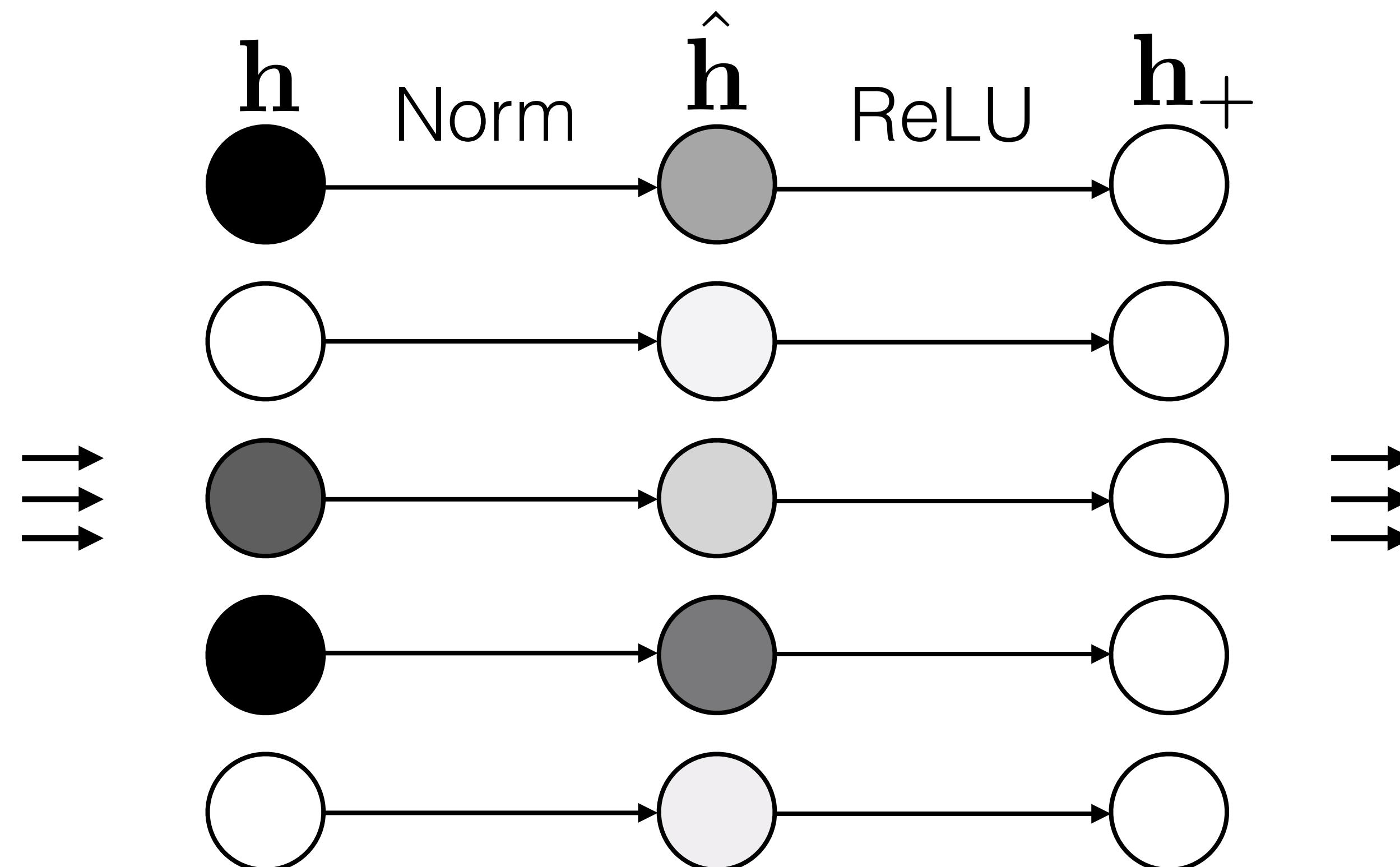
Both these properties reduce the effective capacity of the model, i.e. regularize the model.

# Normalization layers



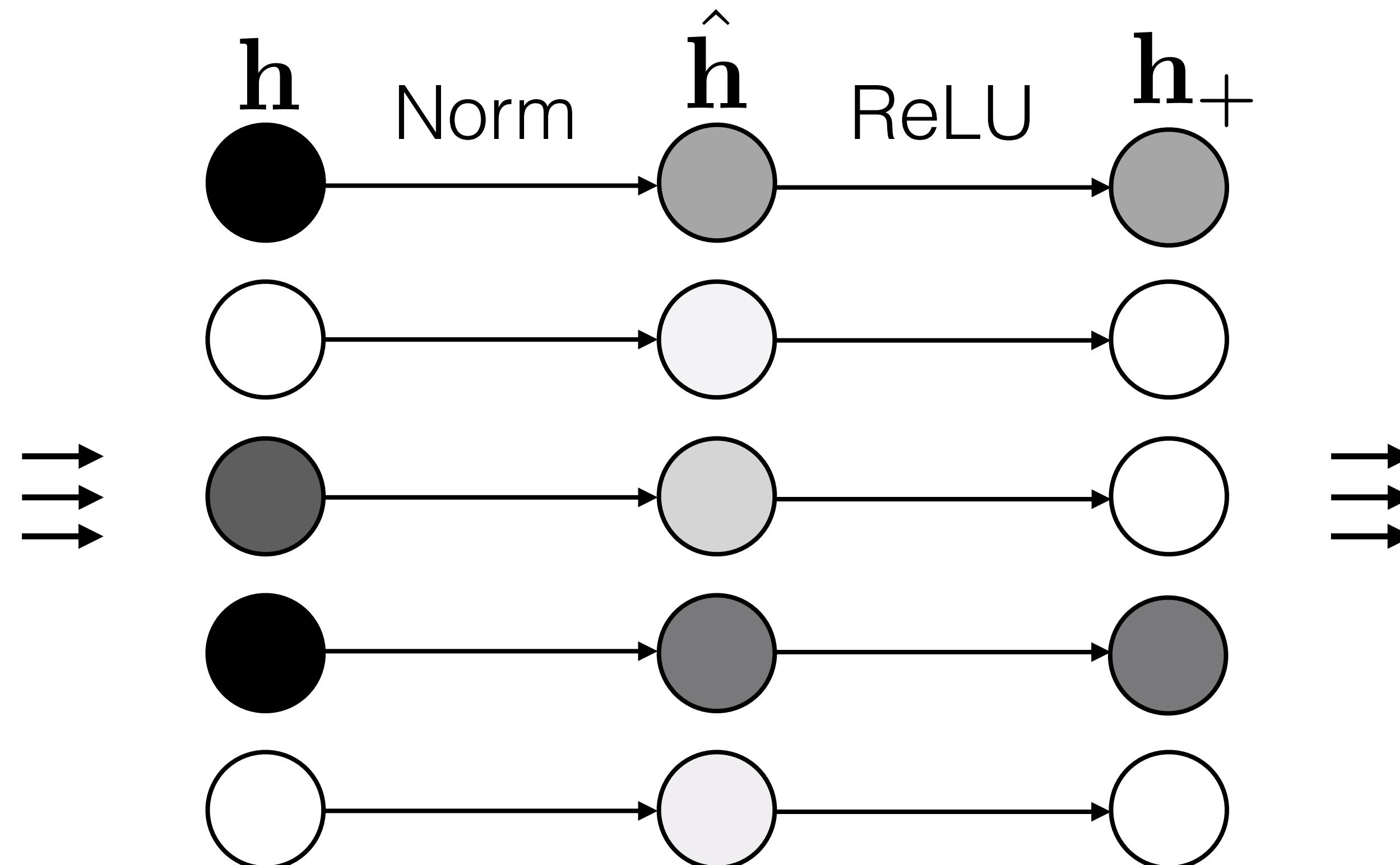
$$\hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\text{Var}[h_k]}}$$

# Normalization layers



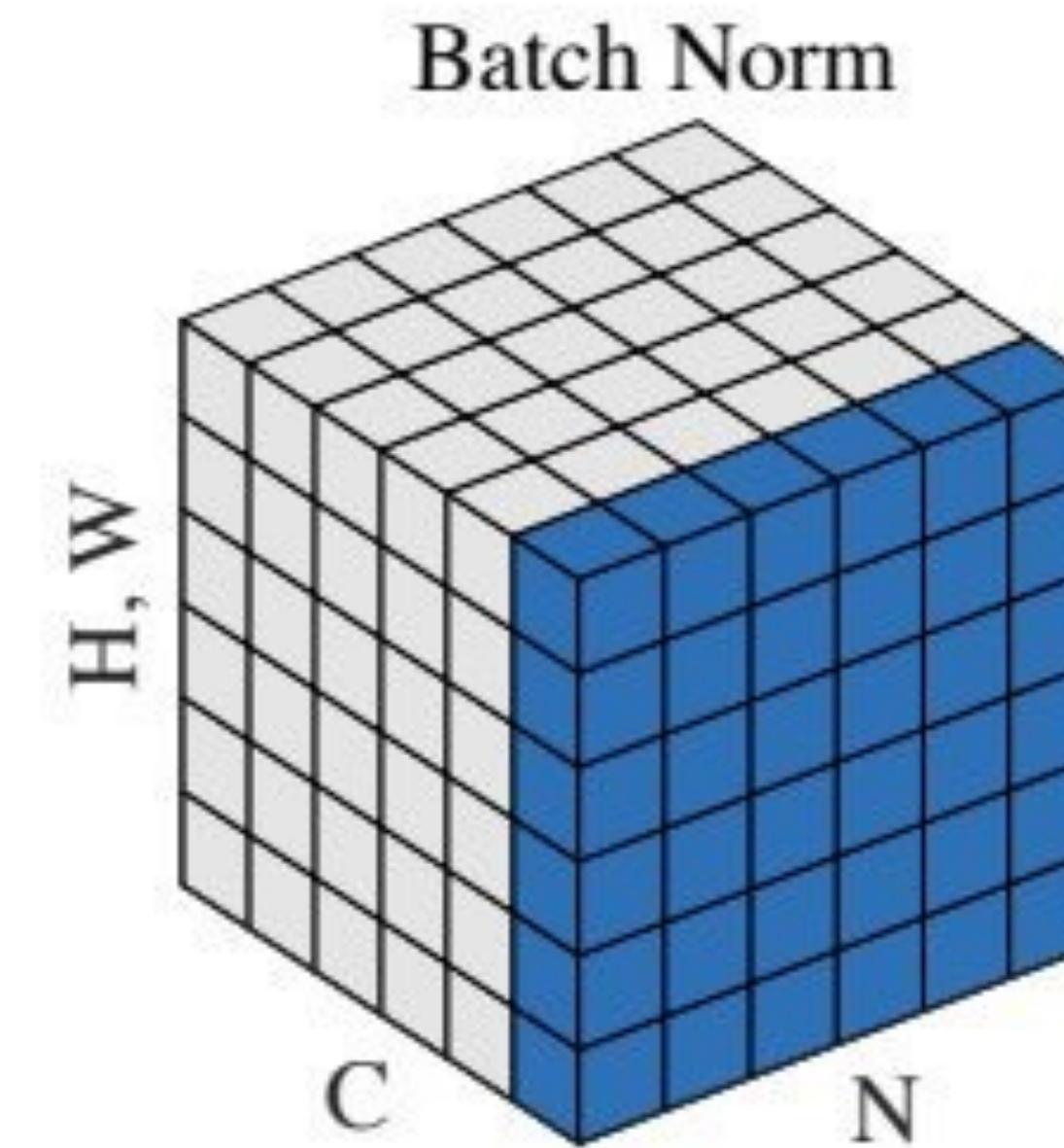
$$\hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\text{Var}[h_k]}}$$

# Normalization layers



$$\hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\text{Var}[h_k]}}$$

# Batch normalization



Filter value in a CNN layer

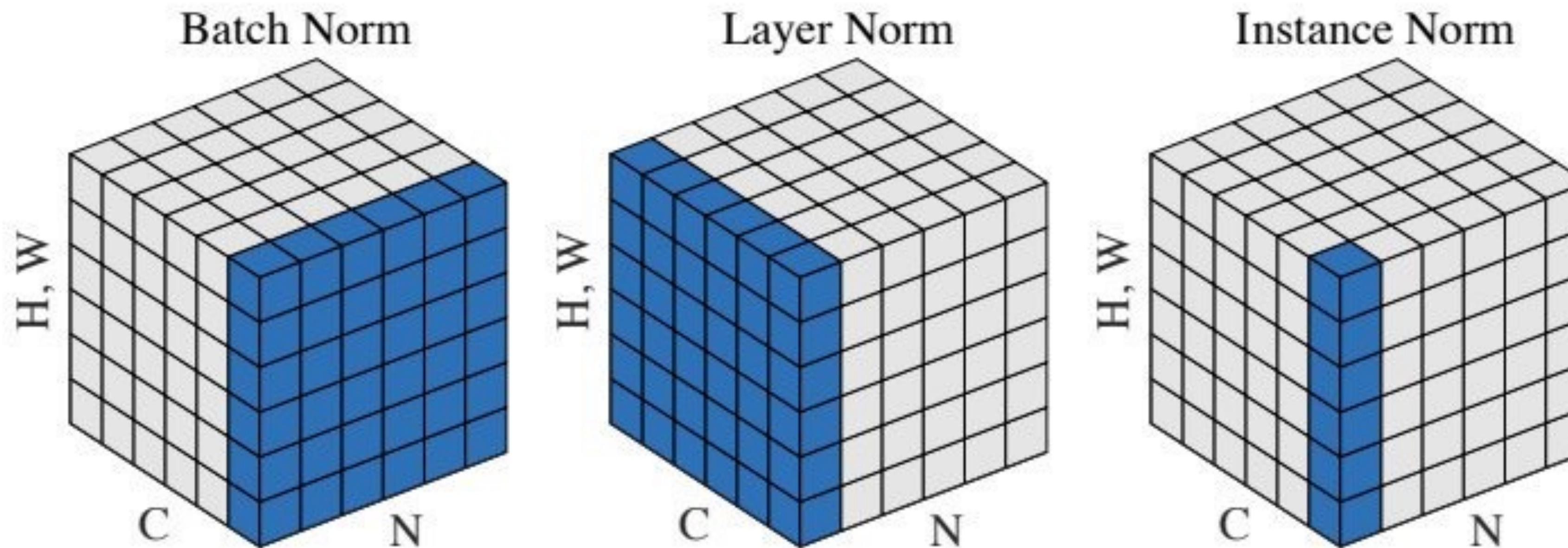
$$\hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\text{Var}[h_k]}}$$

Average and standard deviation  
each filter response, estimated  
over whole batch.

Normalize w.r.t. a single hidden unit's pattern of activation over training examples  
(a batch of examples).

[Figure from Wu & He, arXiv 2018] [Ioffe & Szegedy, 2015]

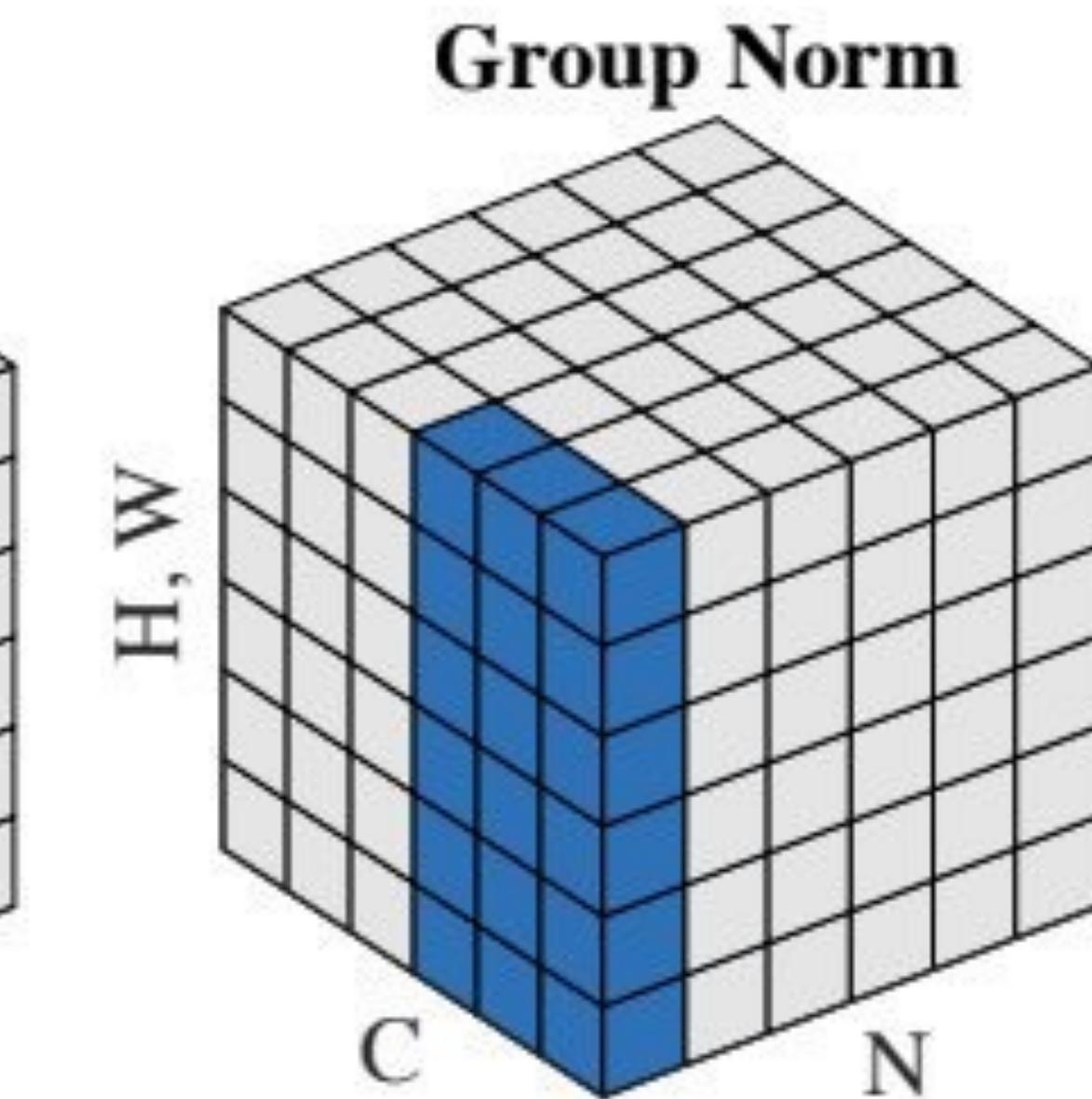
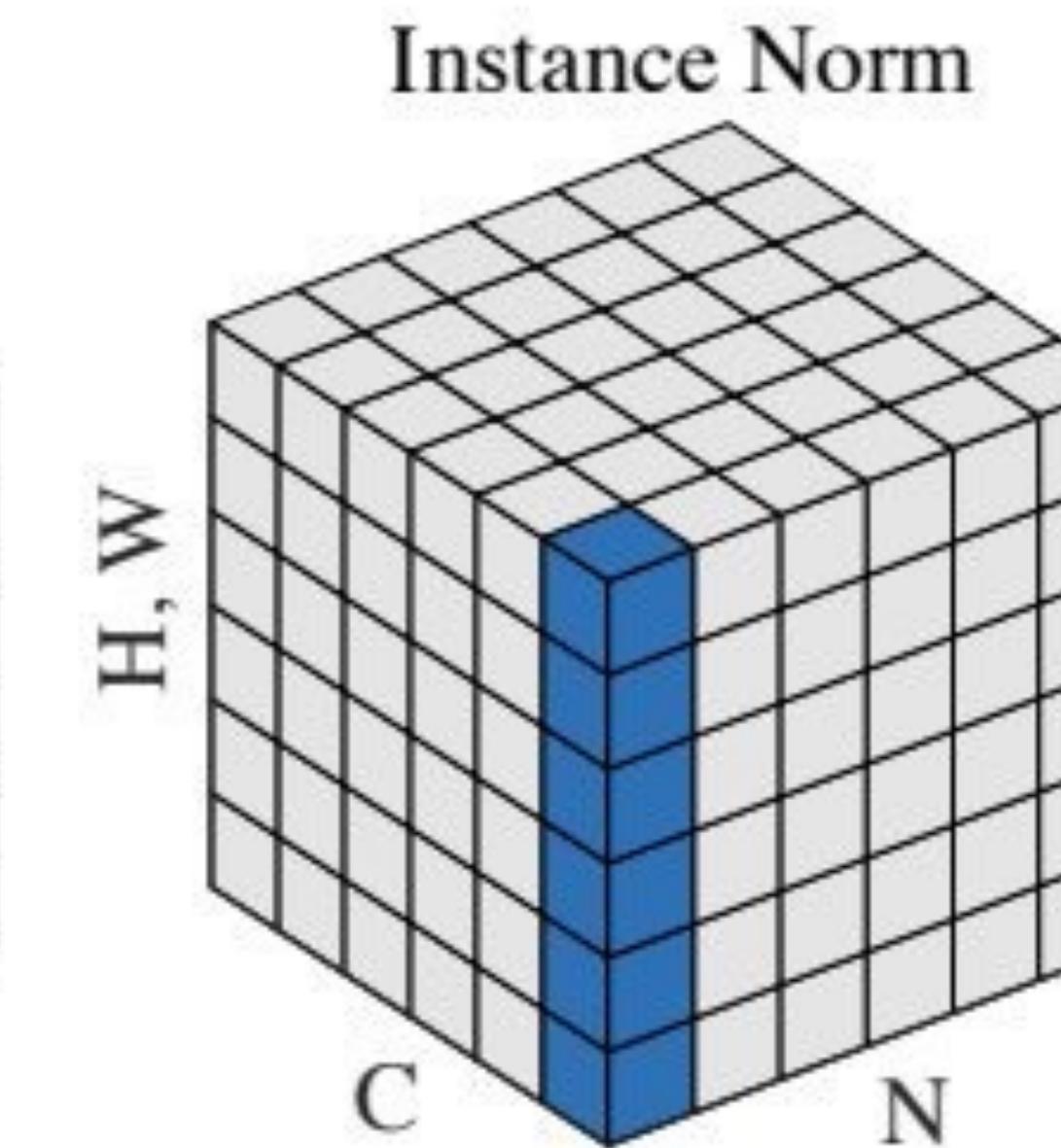
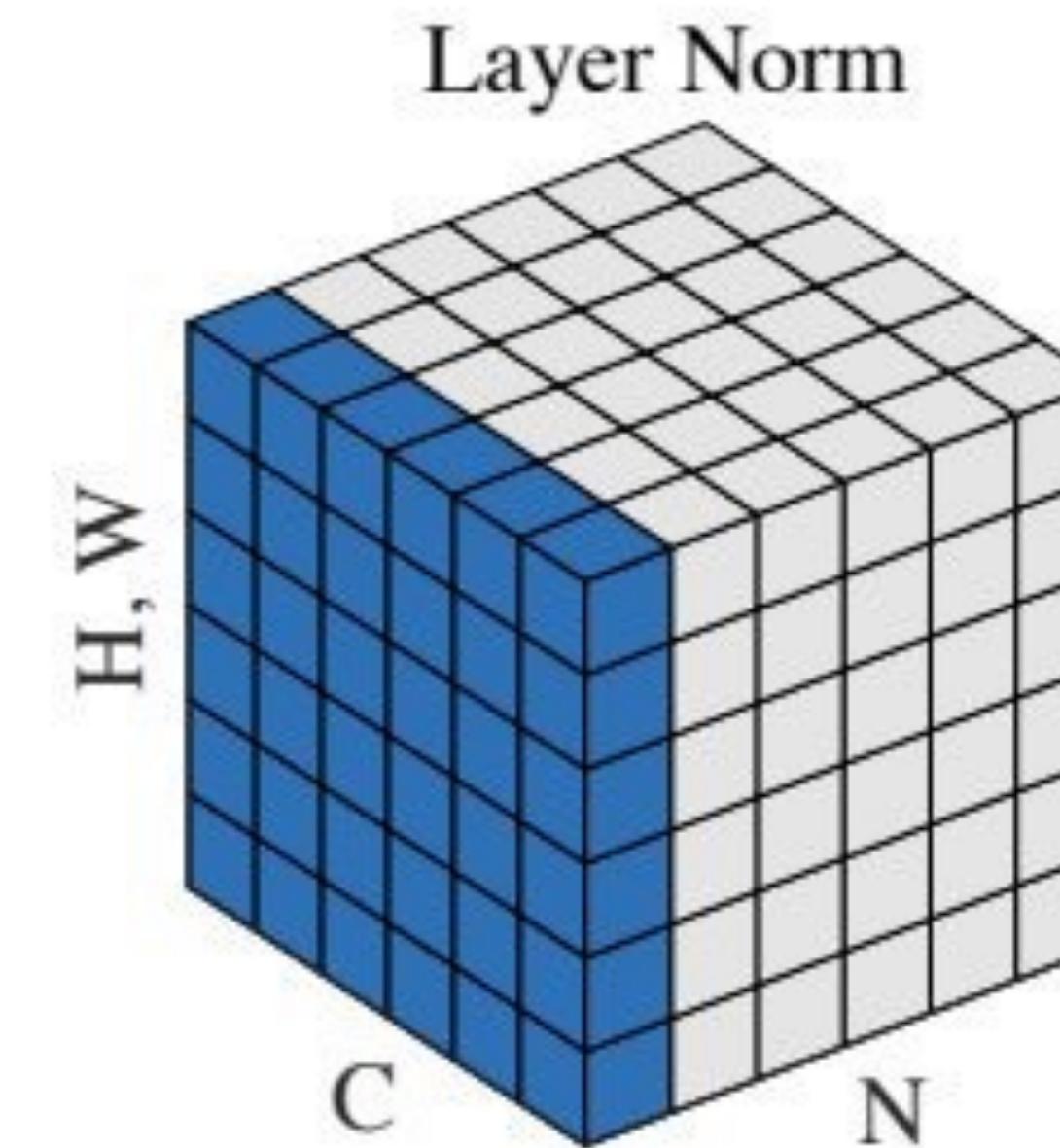
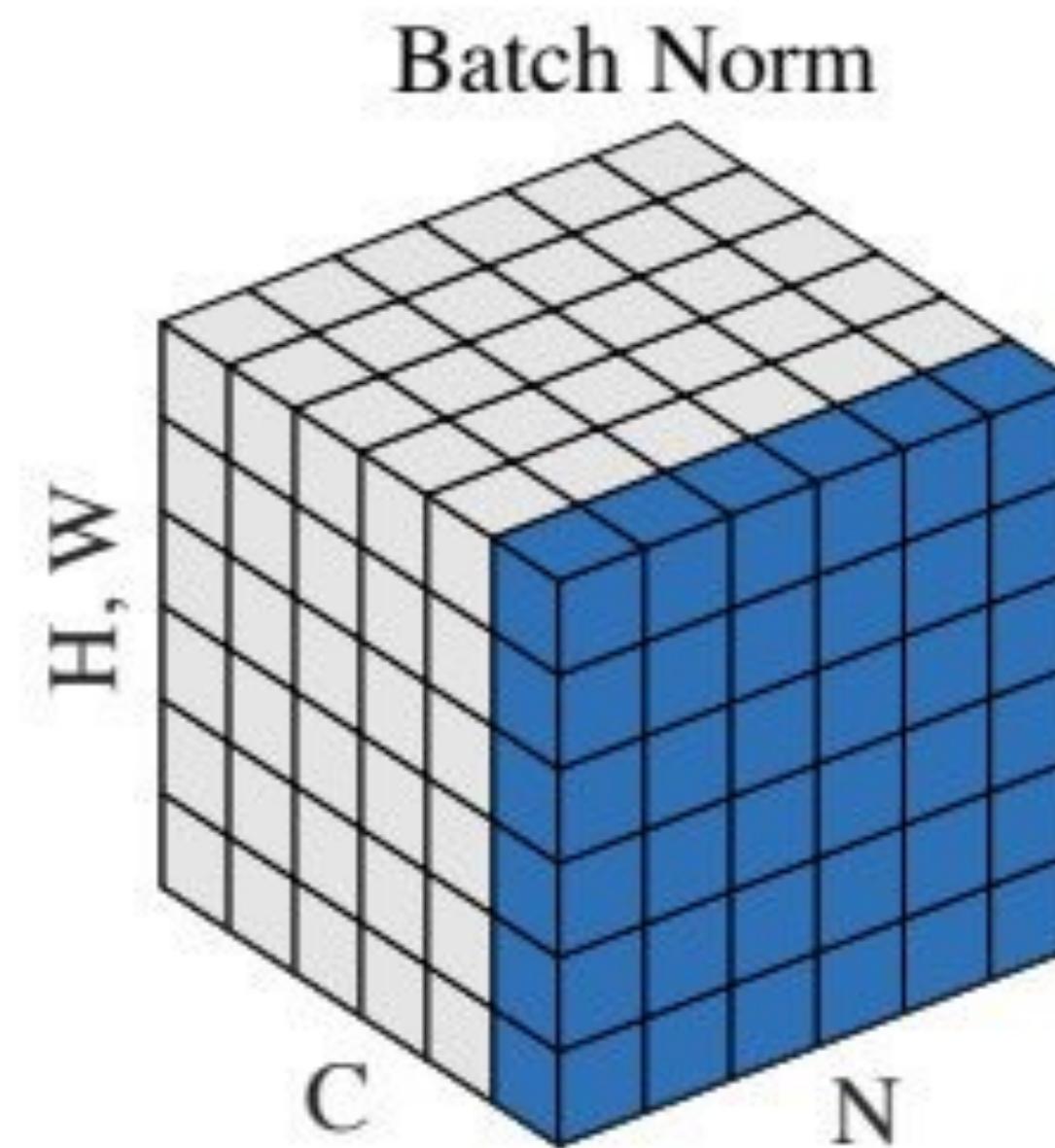
# Normalization layers



Normalize w.r.t. the mean and variance of the activations of all the hidden units (neurons) on this layer ( $c$ ) that process this particular location  $(h,w)$  in the image.

[Figure from Wu & He, ECCV 2018]

# Normalization layers



All used in CNNs, especially when batch size is small.

[Figure from Wu & He, ECCV 2018]