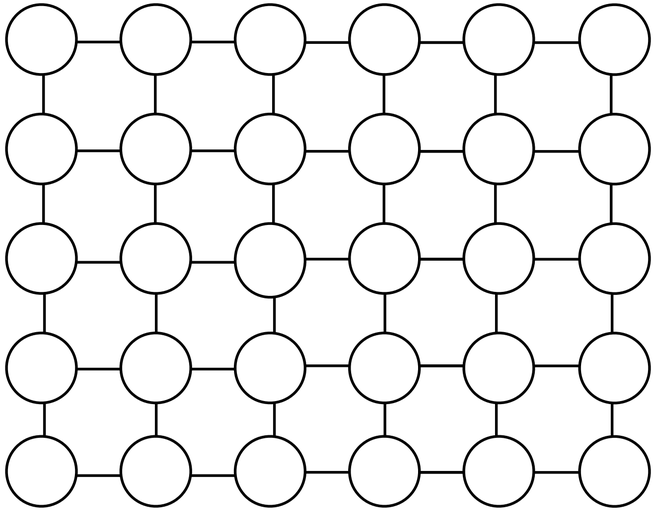
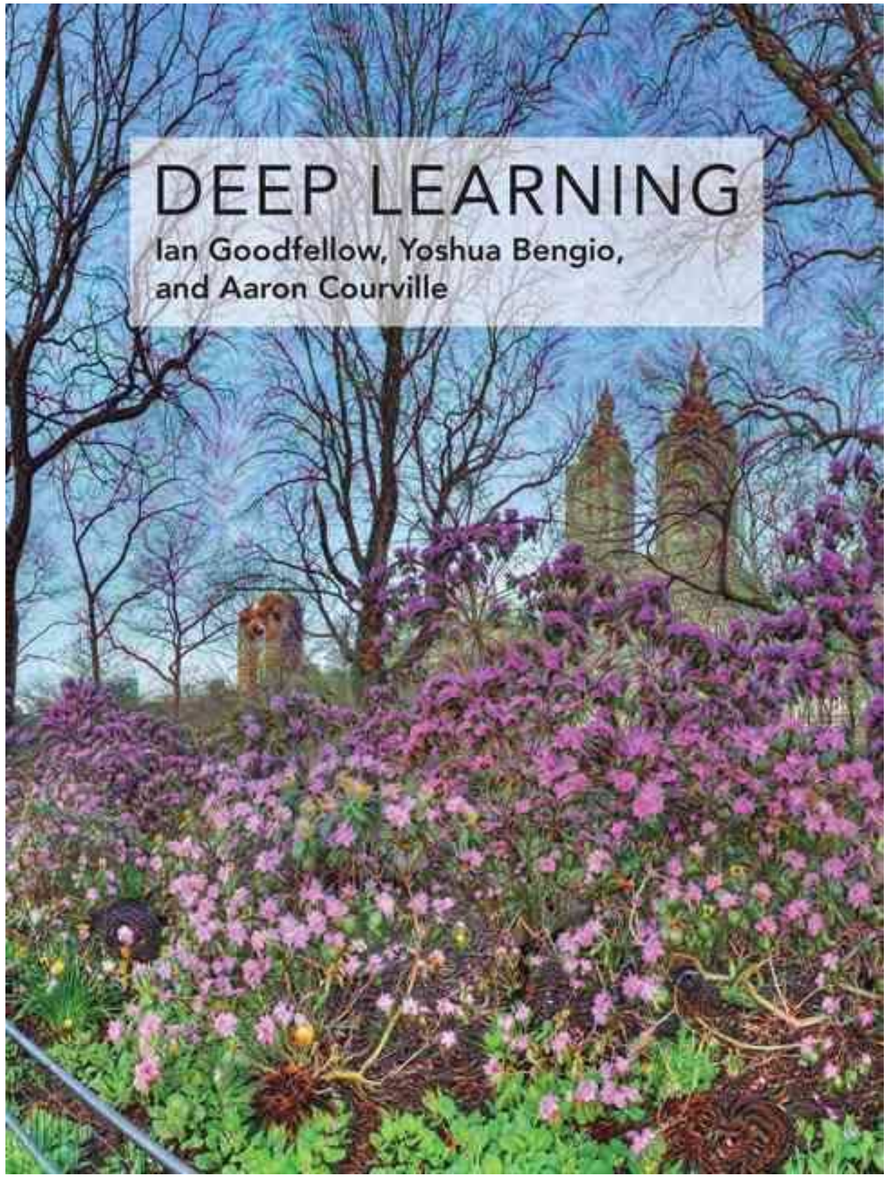


CNNs and Spatial Processing

Bill Freeman, Antonio Torralba, Phillip Isola

6.819 / 6.869



http://www.deeplearningbook.org/

By Ian Goodfellow, Yoshua Bengio and Aaron Courville

November 2016

Today: parts of chapter 9

Review lectures 5 through 8 for background on signal processing, convolution, and multiscale image processing — this is the technology that underlies convnets!

Backpropagation Summary

- Forward pass: for each training example, compute the outputs for all layers:

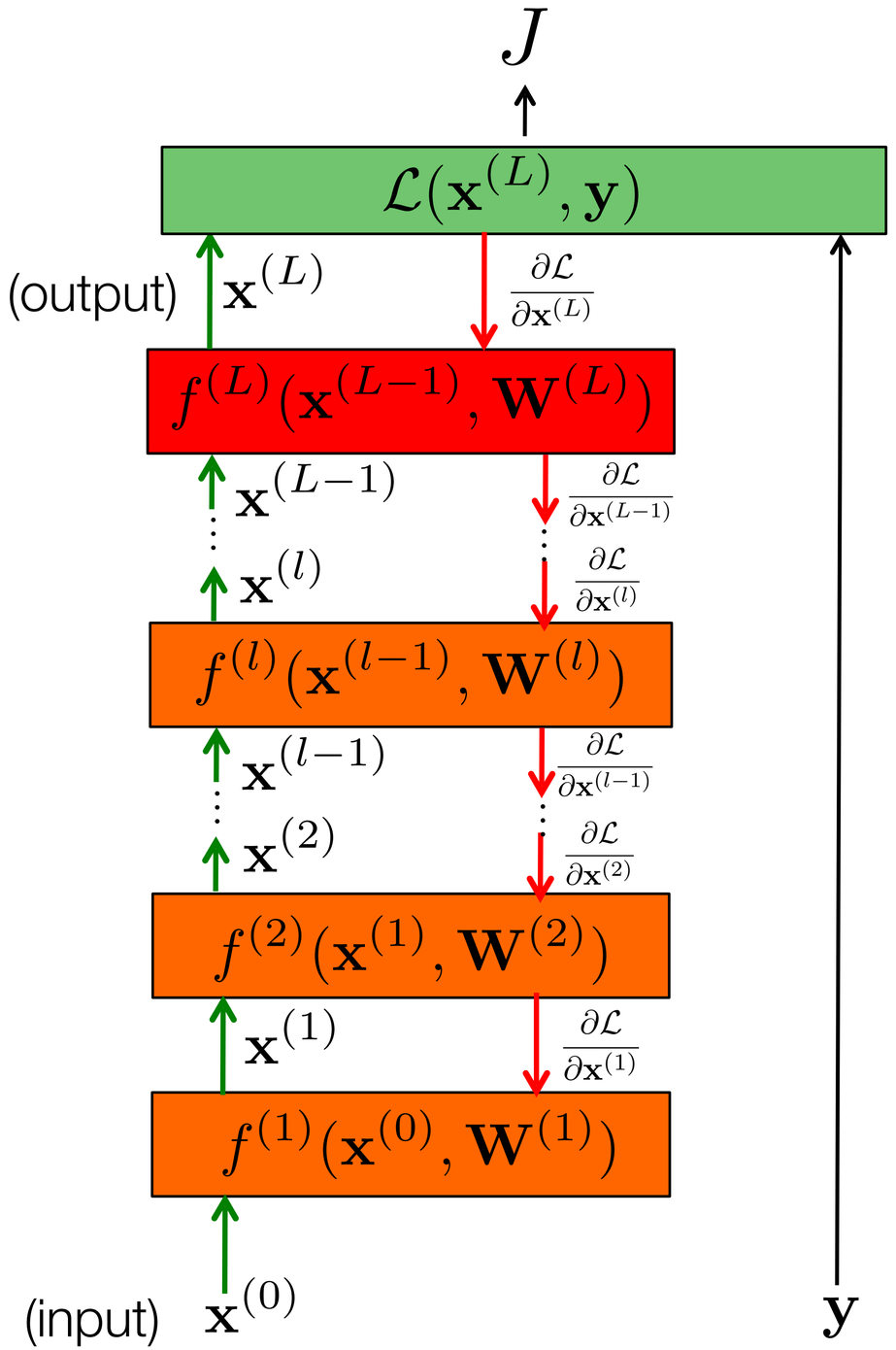
$$\mathbf{x}^{(l)} = f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})$$

- Backwards pass: compute loss derivatives iteratively from top to bottom:

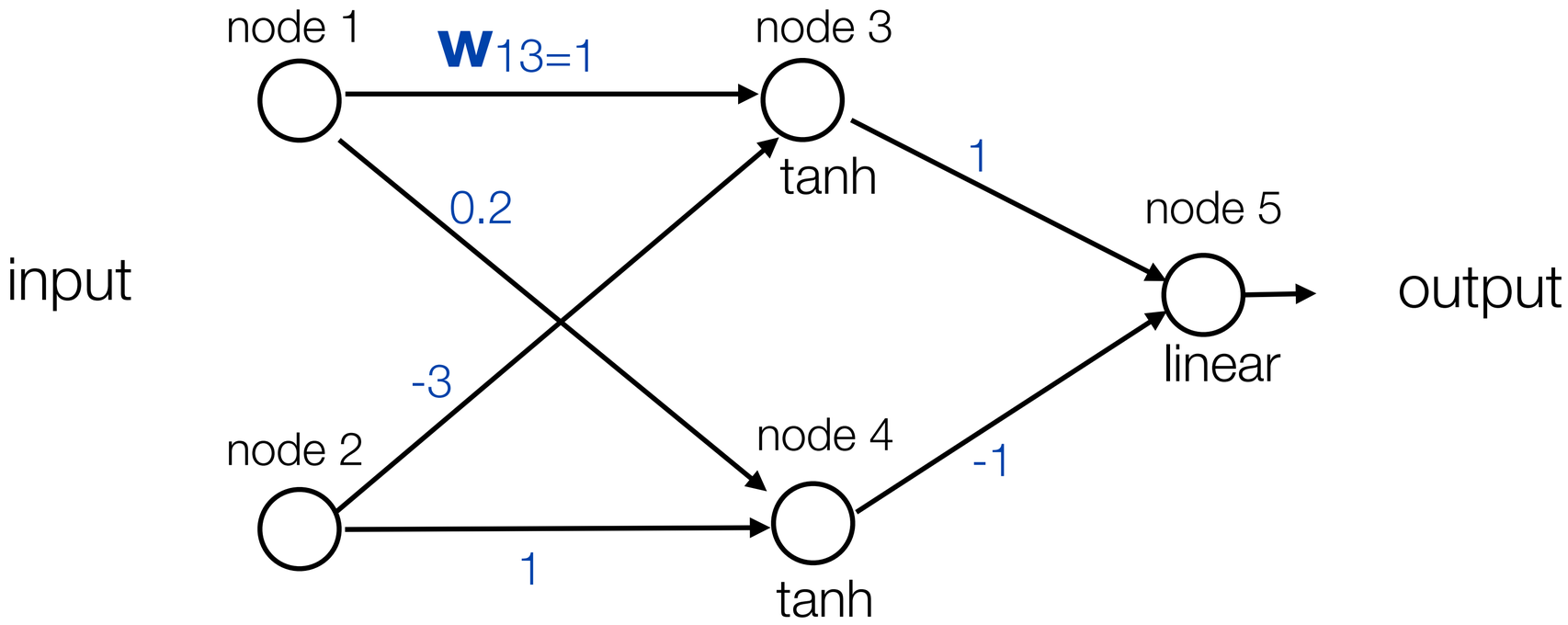
$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \cdot \frac{\partial f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})}{\partial \mathbf{x}^{(l-1)}}$$

- Compute gradients w.r.t. weights, and update weights:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \cdot \frac{\partial f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})}{\partial \mathbf{W}^{(l)}}$$



Backpropagation example



Learning rate η = -0.2 (because we used positive increments)

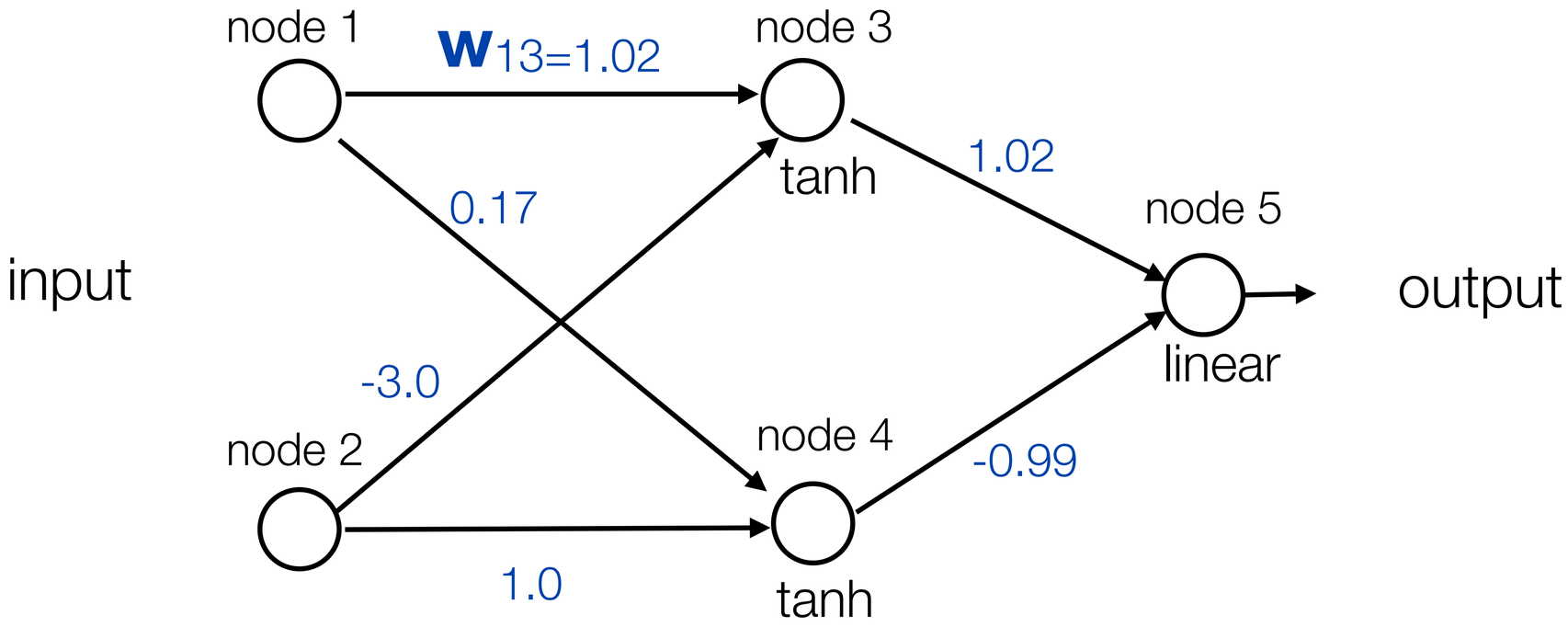
Euclidean loss

Training data: input desired output

node 1	node 2	node 5
1.0	0.1	0.5

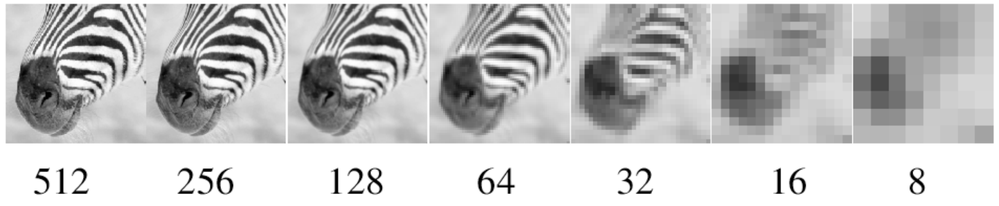
Exercise: run one iteration of back propagation

Backpropagation example

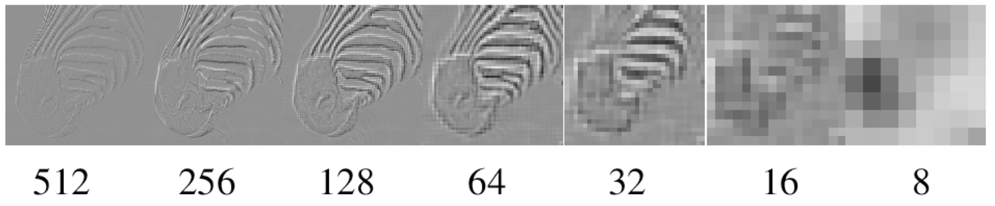


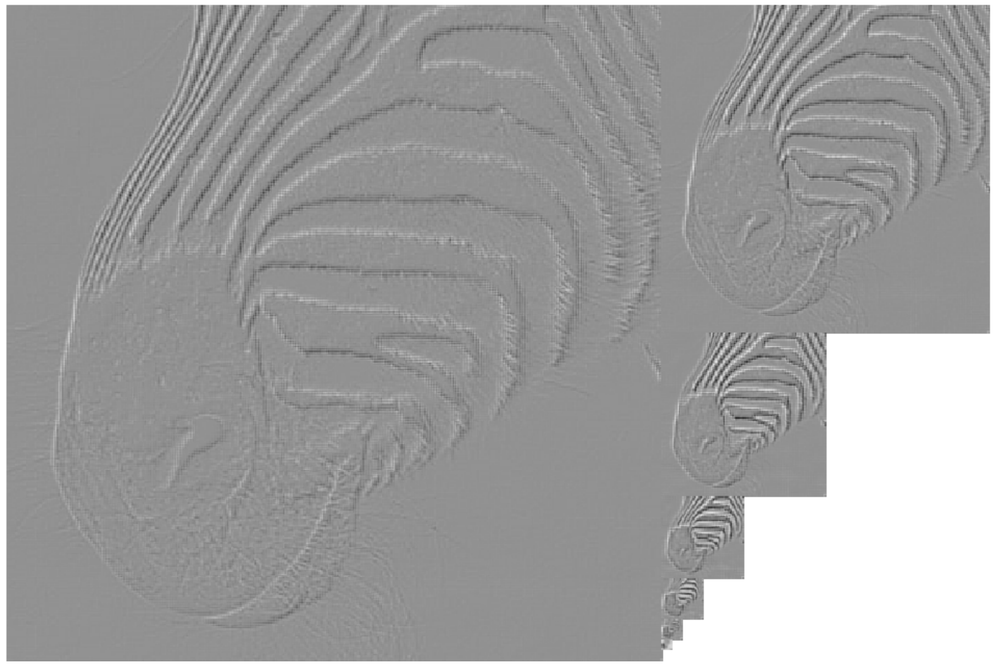
After one iteration (rounding to two digits)

Lots of image pyramids…



Gaussian Pyr





Laplacian Pyr

And many more: QMF, steerable, … **Convnets!**

Convolutional Neural networks

LeCun et al. 1989

Neural network with specialized connectivity

Tailored to processing natural signals with a grid topology (e.g., images).

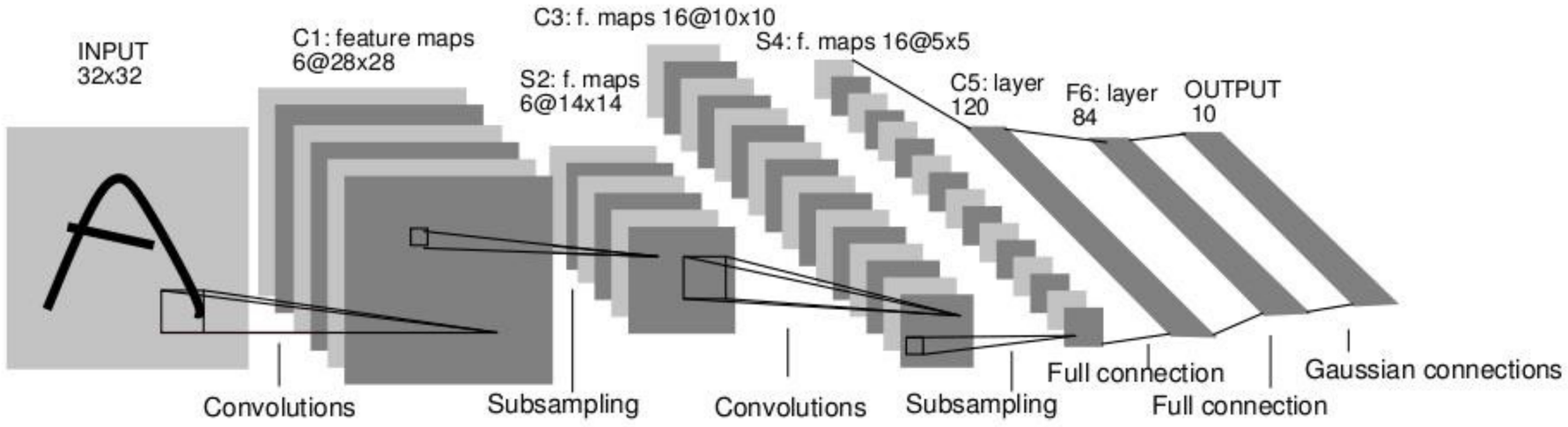
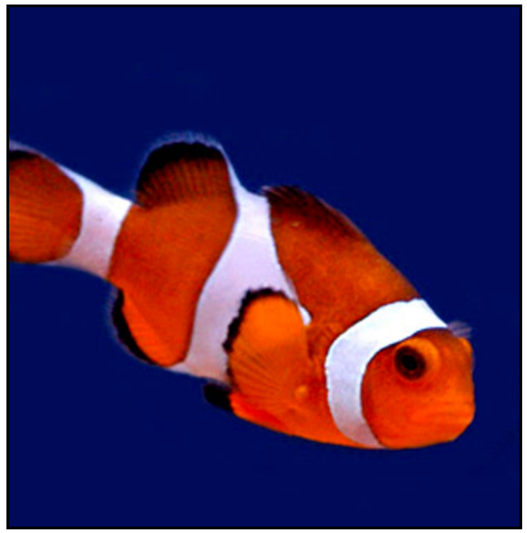


Image classification



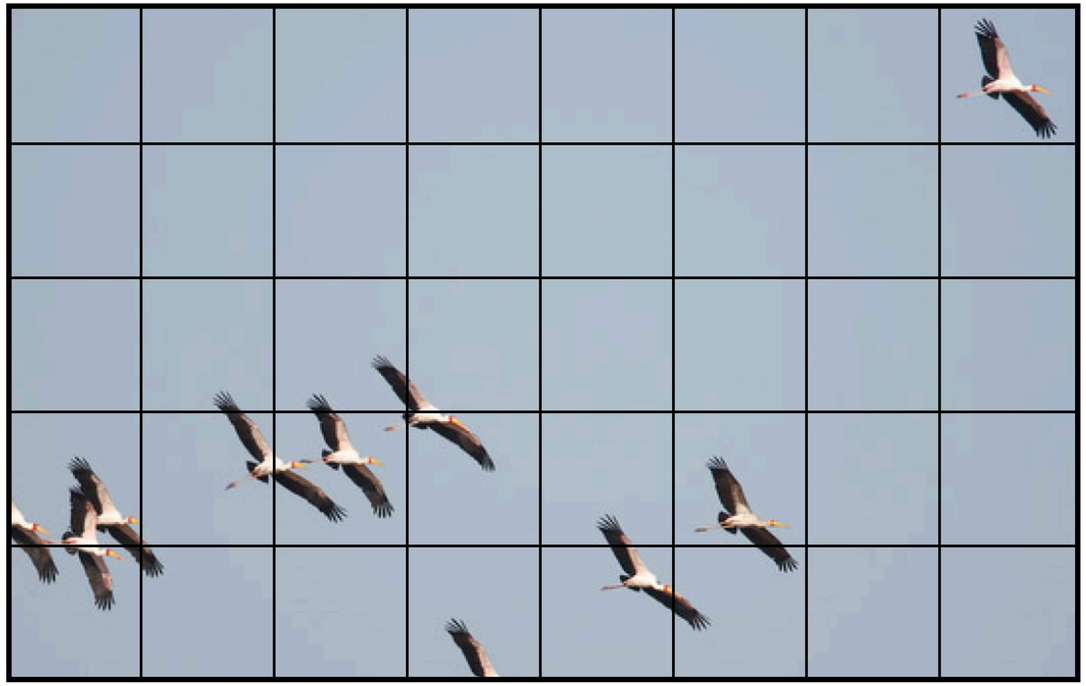
Classifier

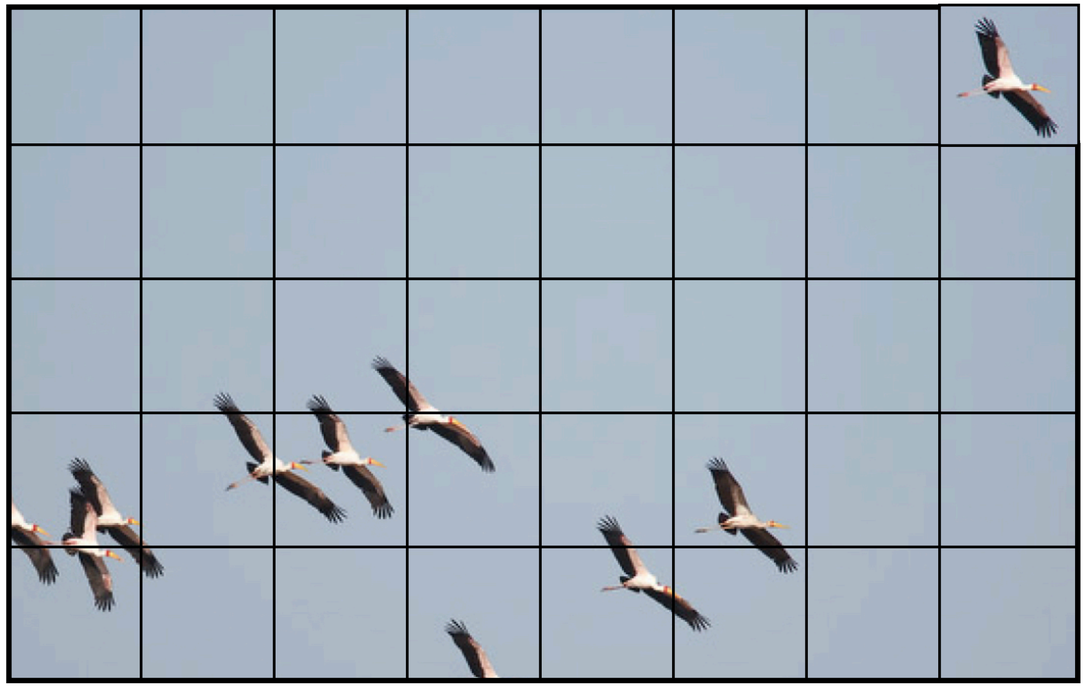
“Fish”

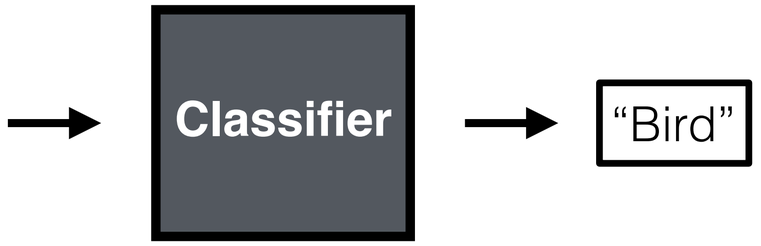
image **x**

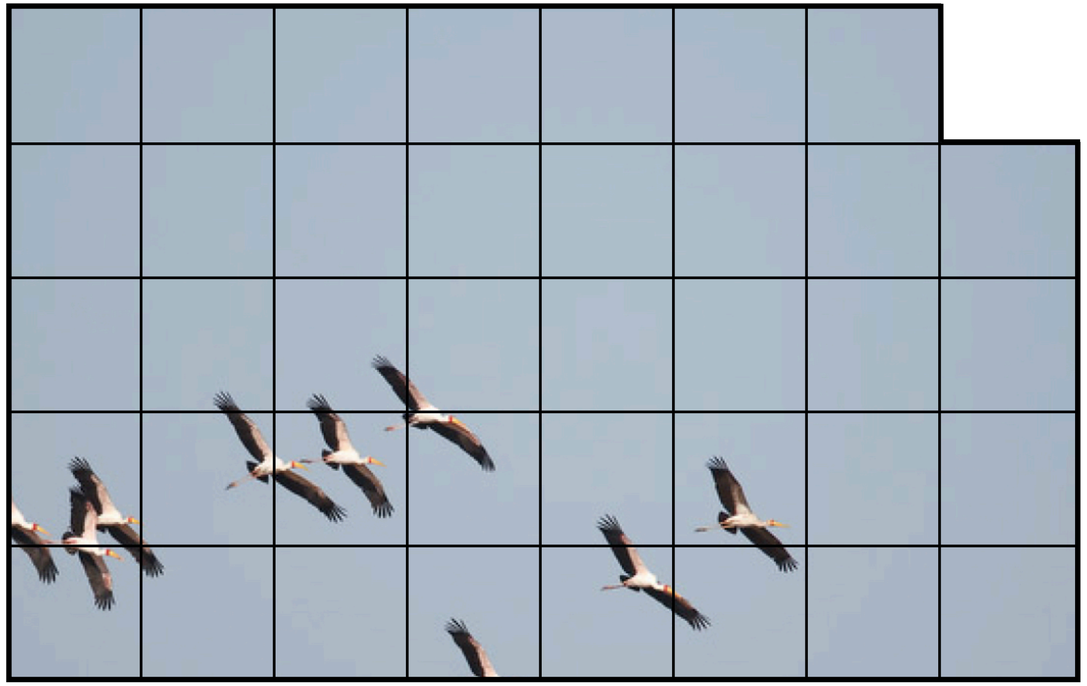
label y

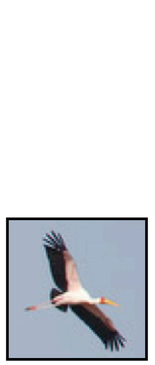
Photo credit: Fredo Durand





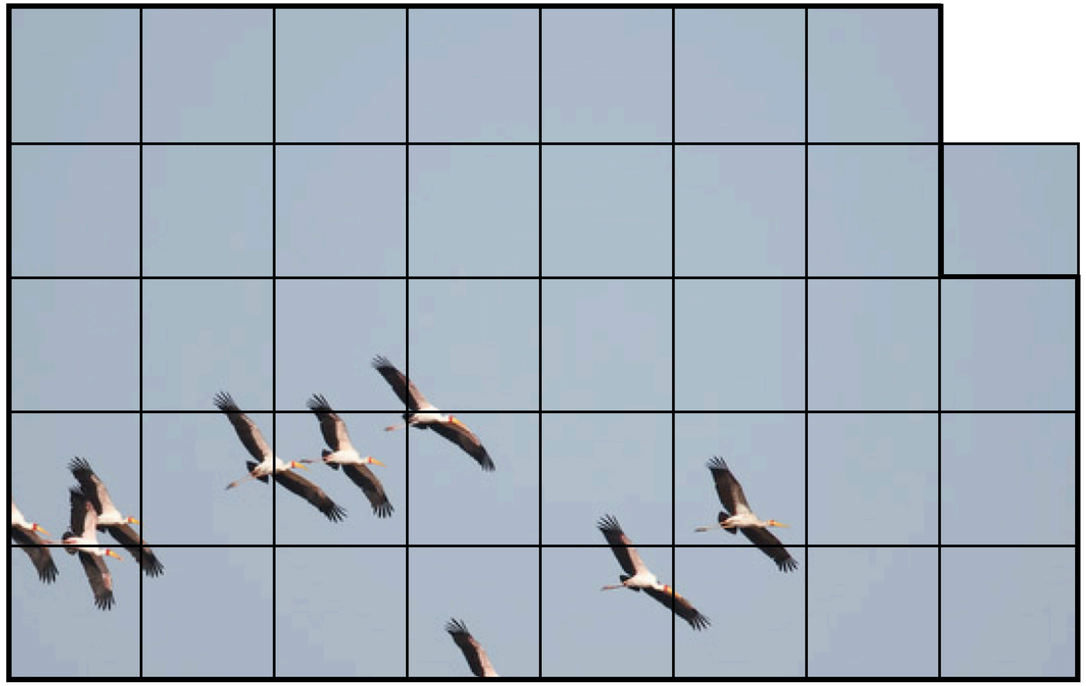

Classifier
“Bird”



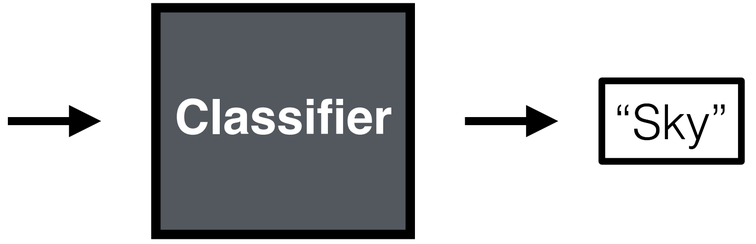


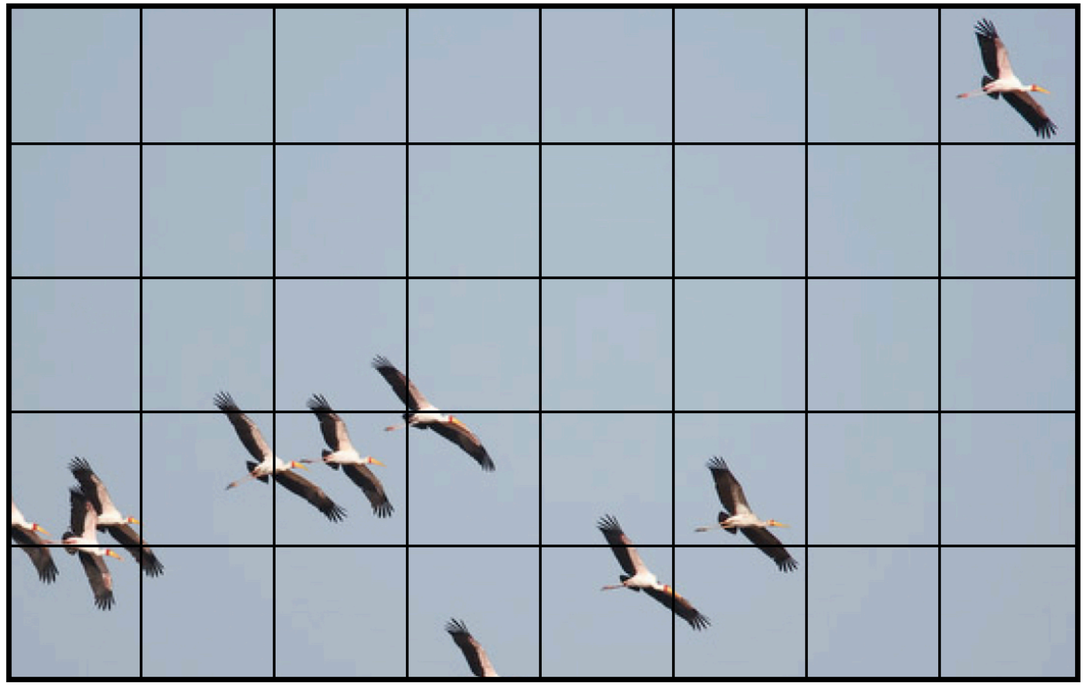
Classifier

“Bird”

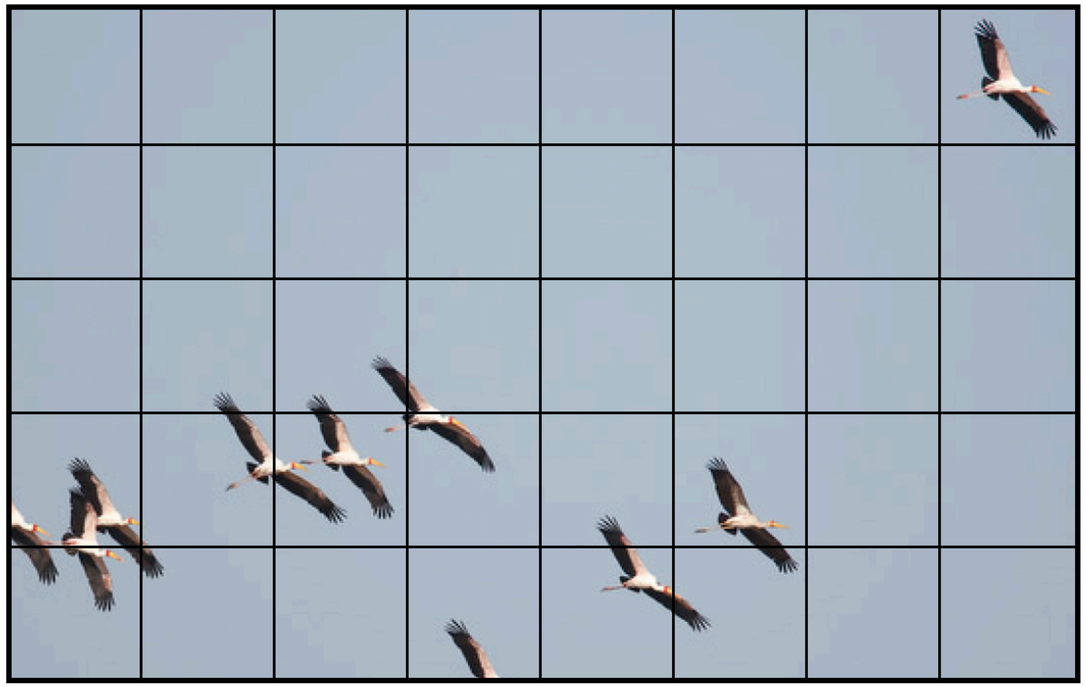


							Bird

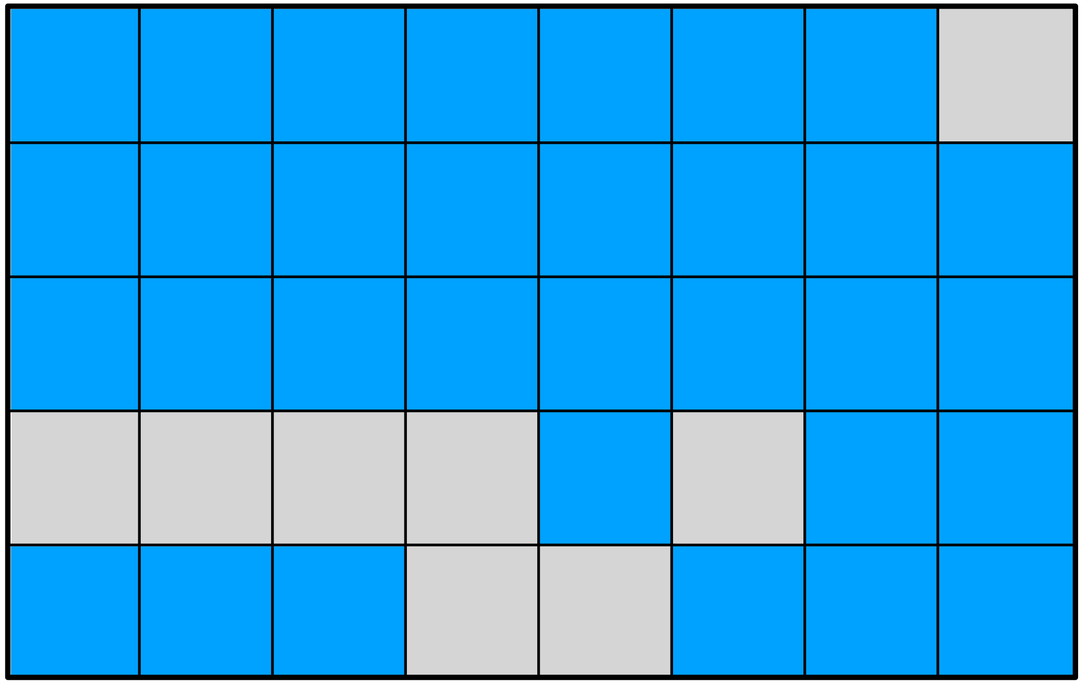


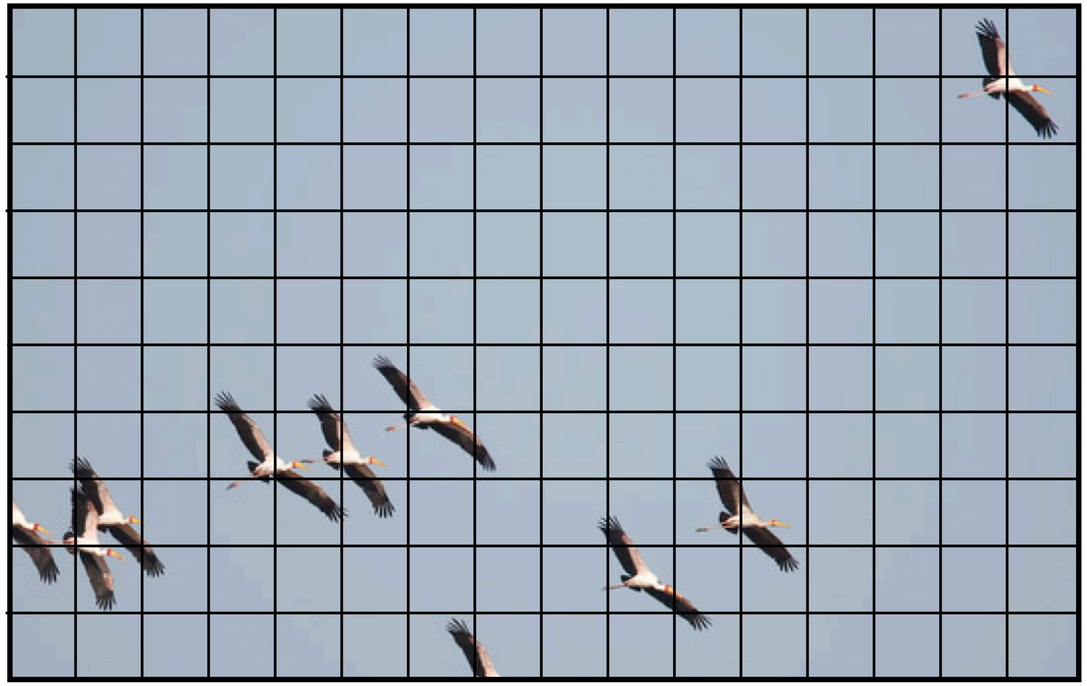


Sky	Sky	Sky	Sky	Sky	Sky	Sky	Bird
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Bird	Bird	Bird	Sky	Bird	Sky	Sky	Sky
Sky	Sky	Sky	Bird	Sky	Sky	Sky	Sky

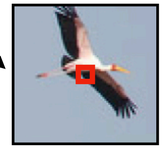
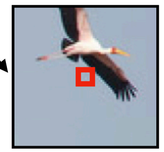
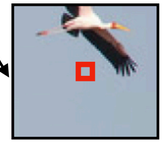


f





What's the object class of the center pixel?

f

"Bird"

f

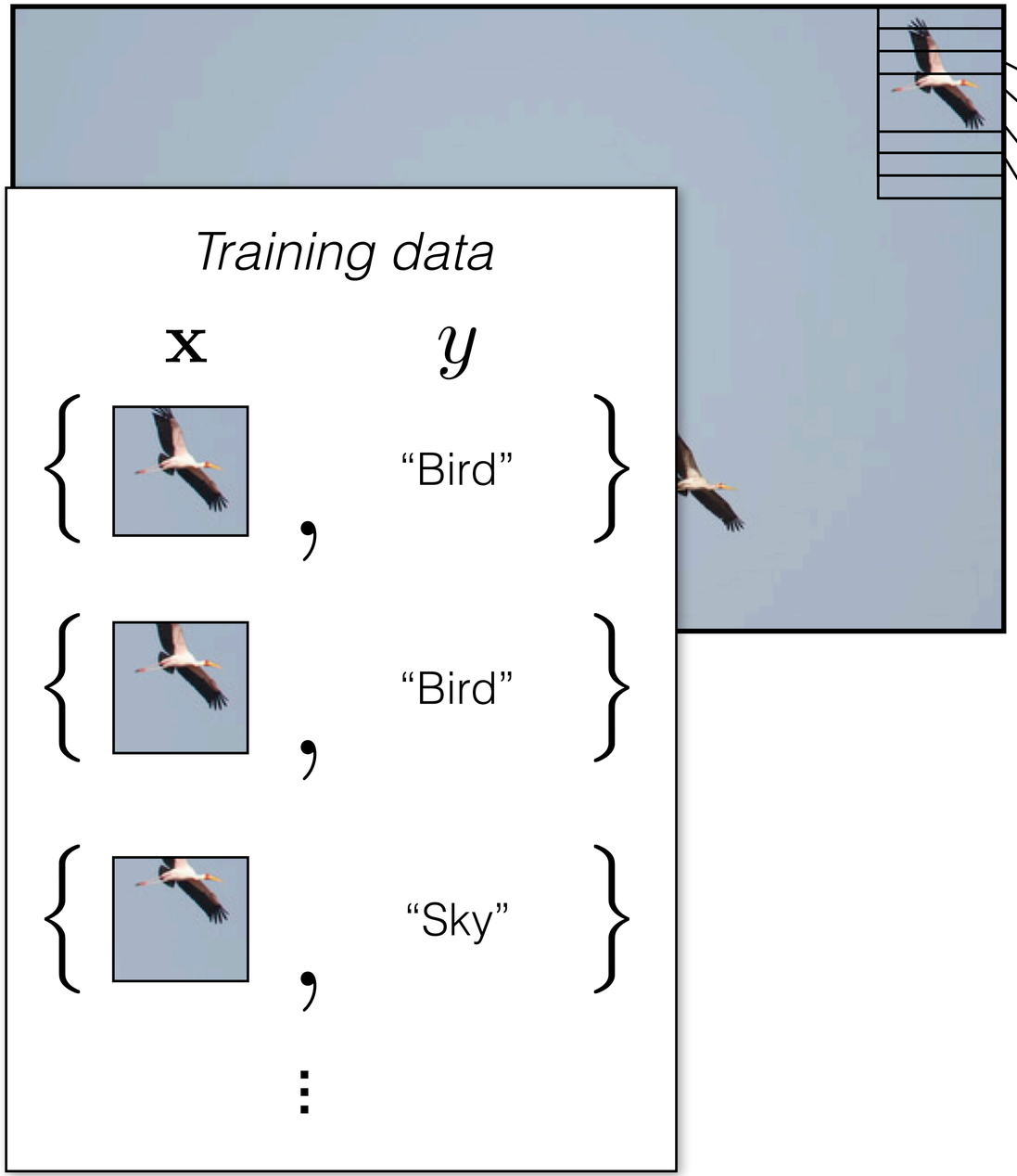
"Bird"

f

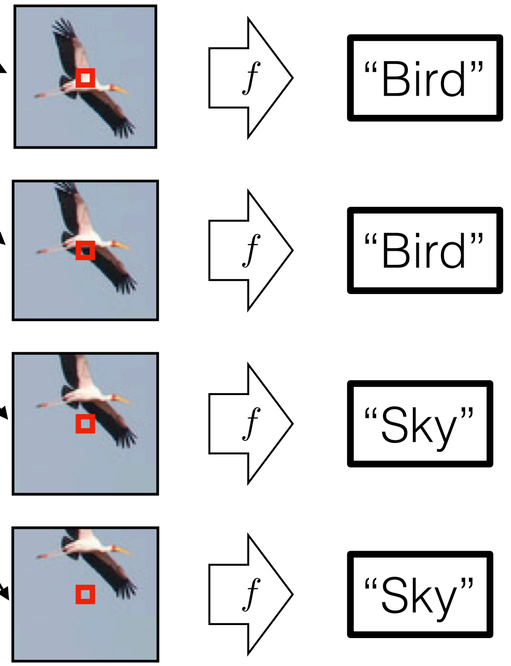
"Sky"

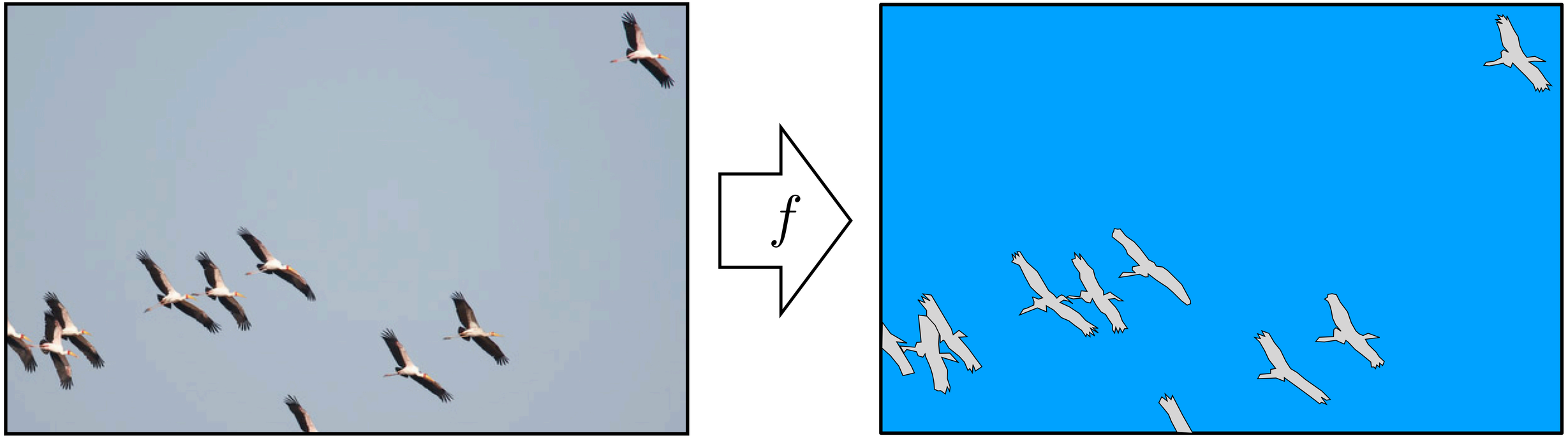
f

"Sky"



What’s the object class of the center pixel?

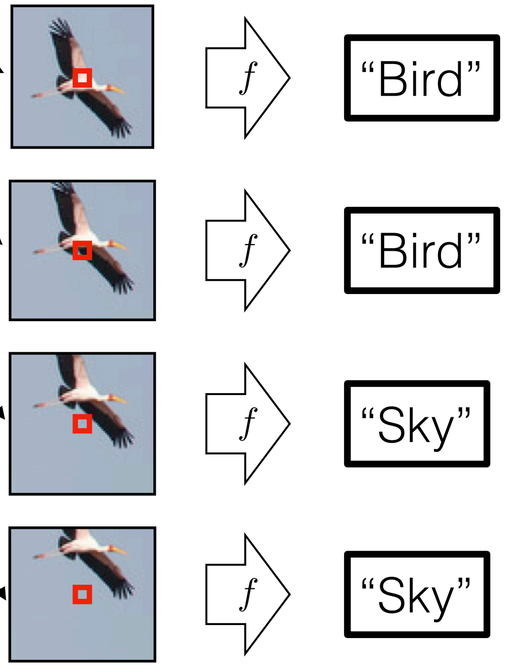




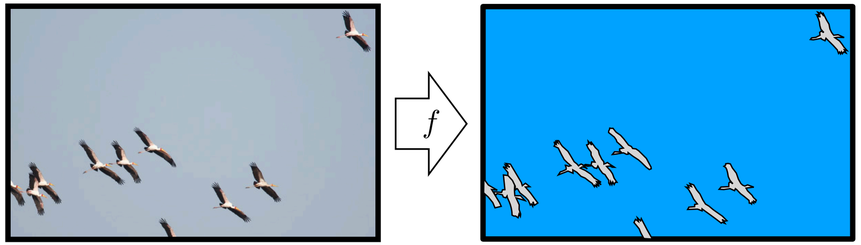
(Colors represent one-hot codes)

This problem is called **semantic segmentation**

What's the object class of the center pixel?



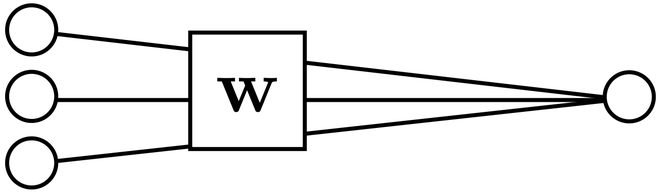
Translation invariance: process each patch in the same way.

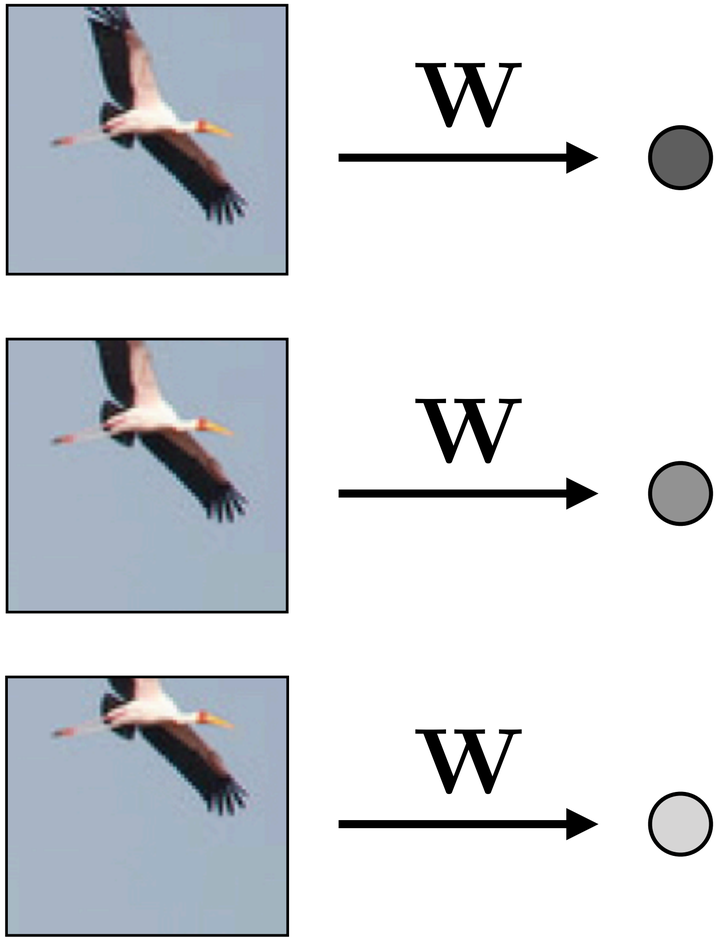


An *equivariant* mapping:

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

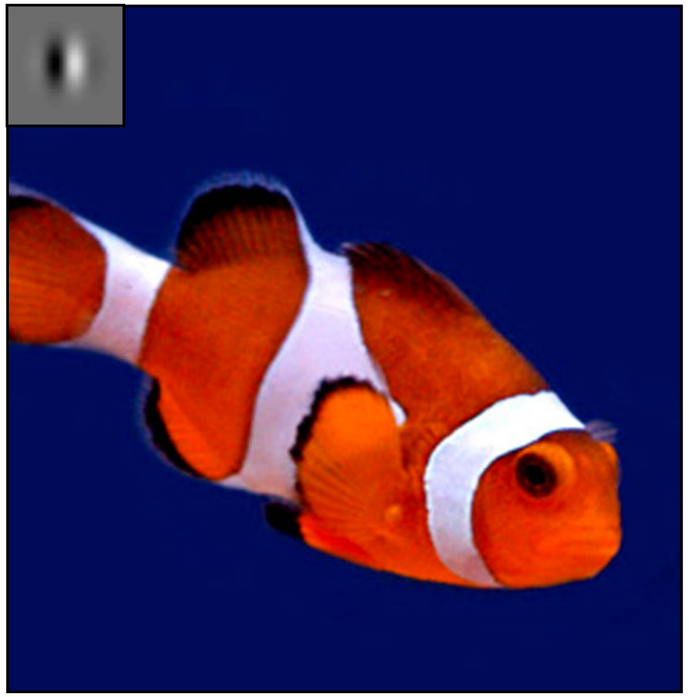
W computes a weighted sum of all pixels in the patch

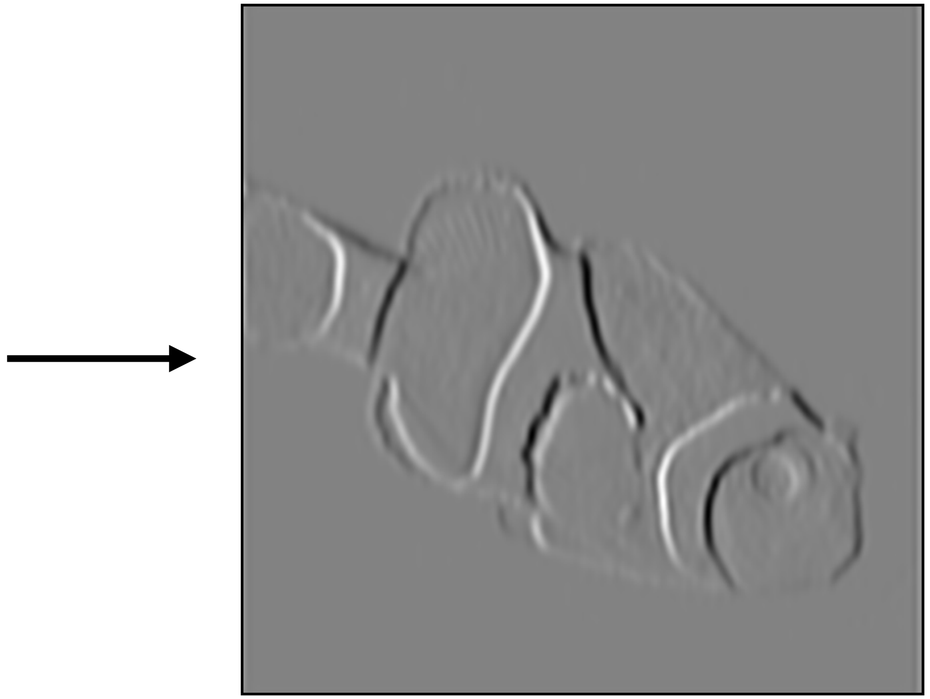




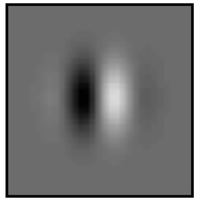
W is a convolutional kernel applied to the full image!

Convolution



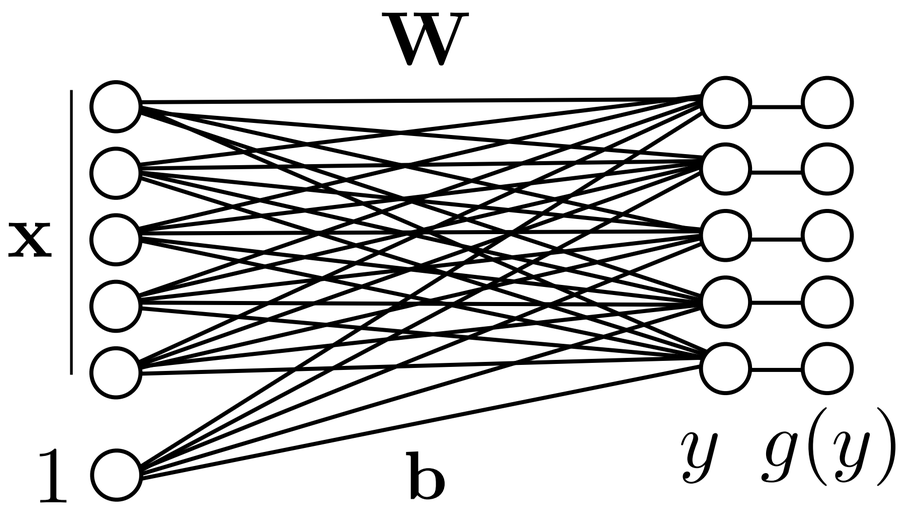


filter

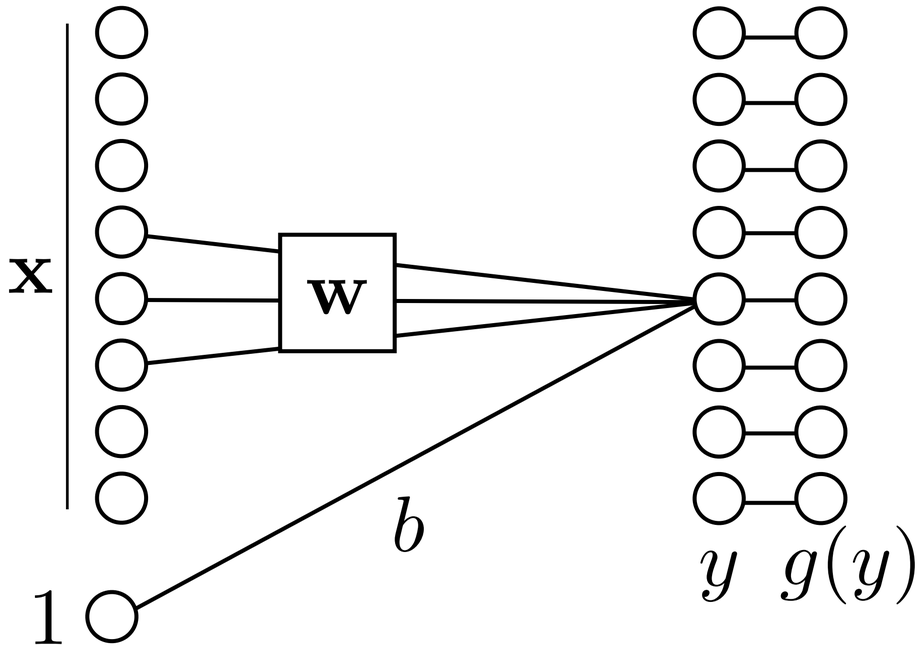


Fully-connected network

Fully-connected (fc) layer



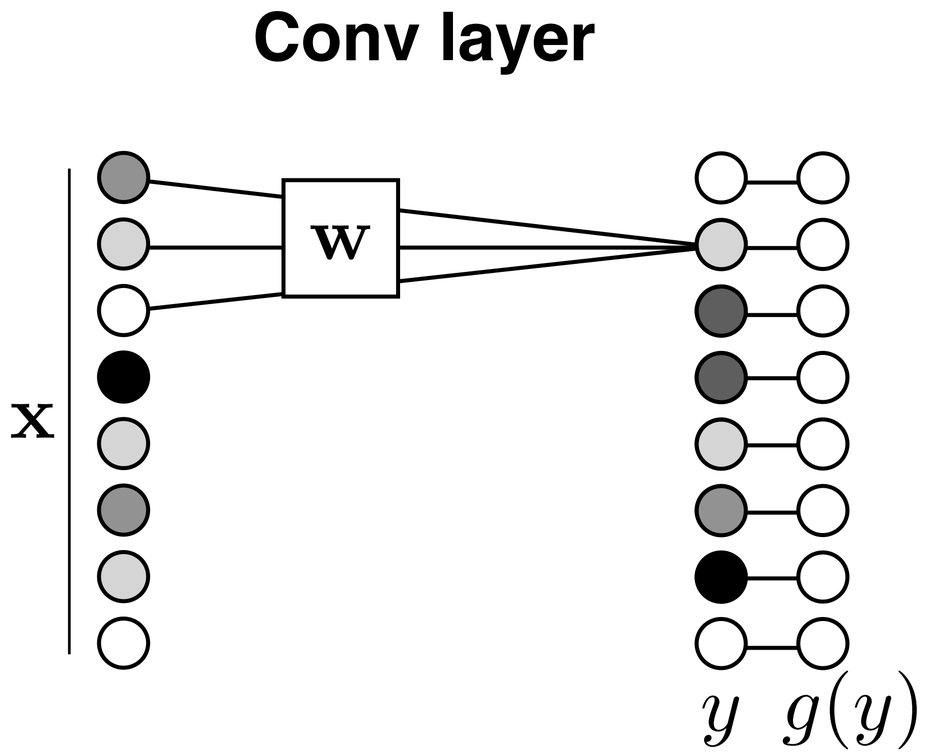
Locally connected network



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.

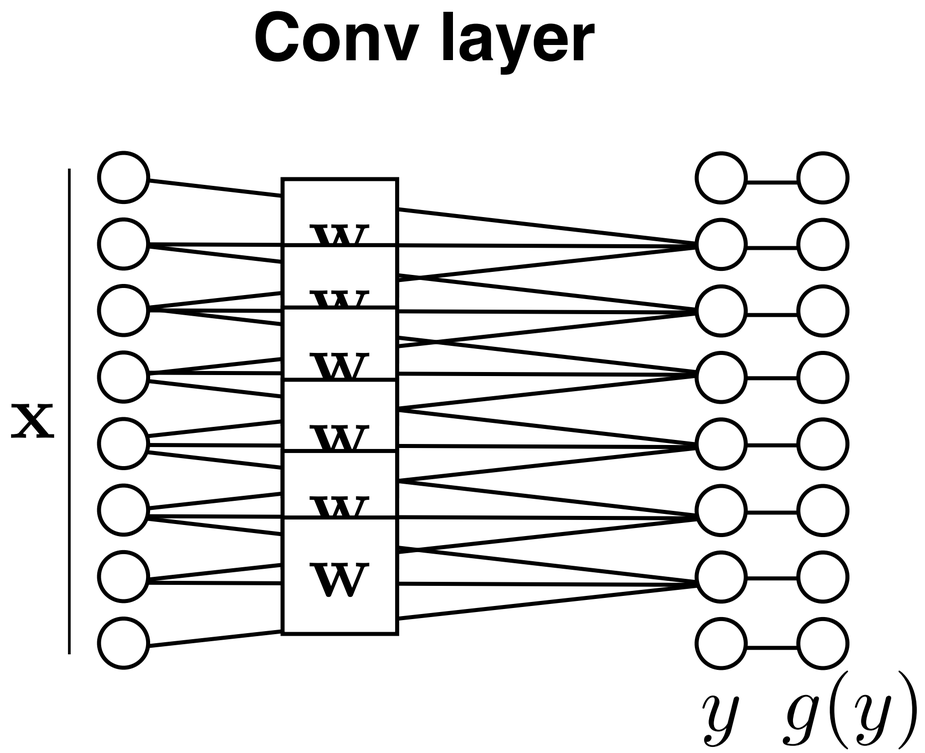
Convolutional neural network



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

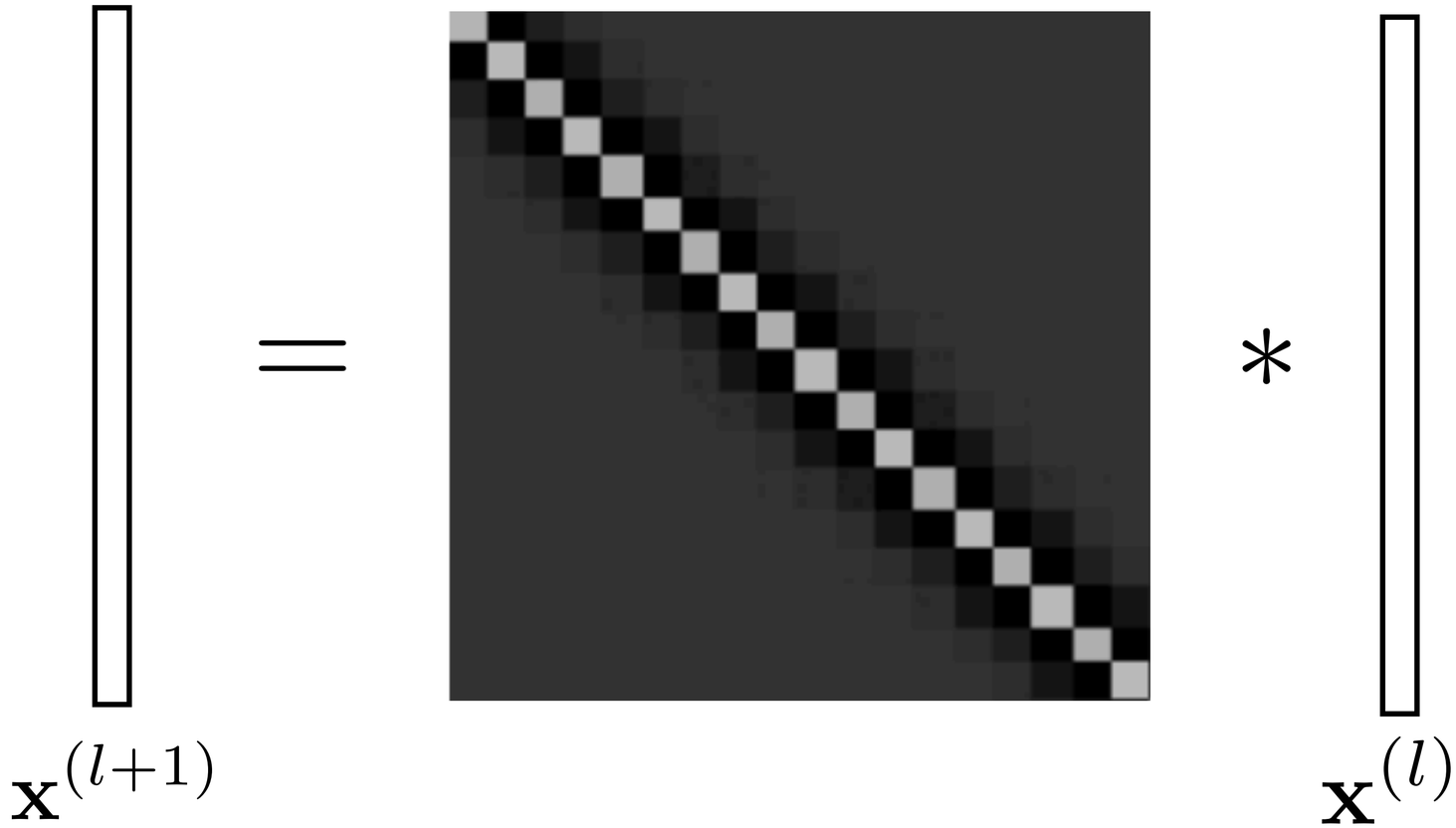


Often, we assume output is a **local** function of input.

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

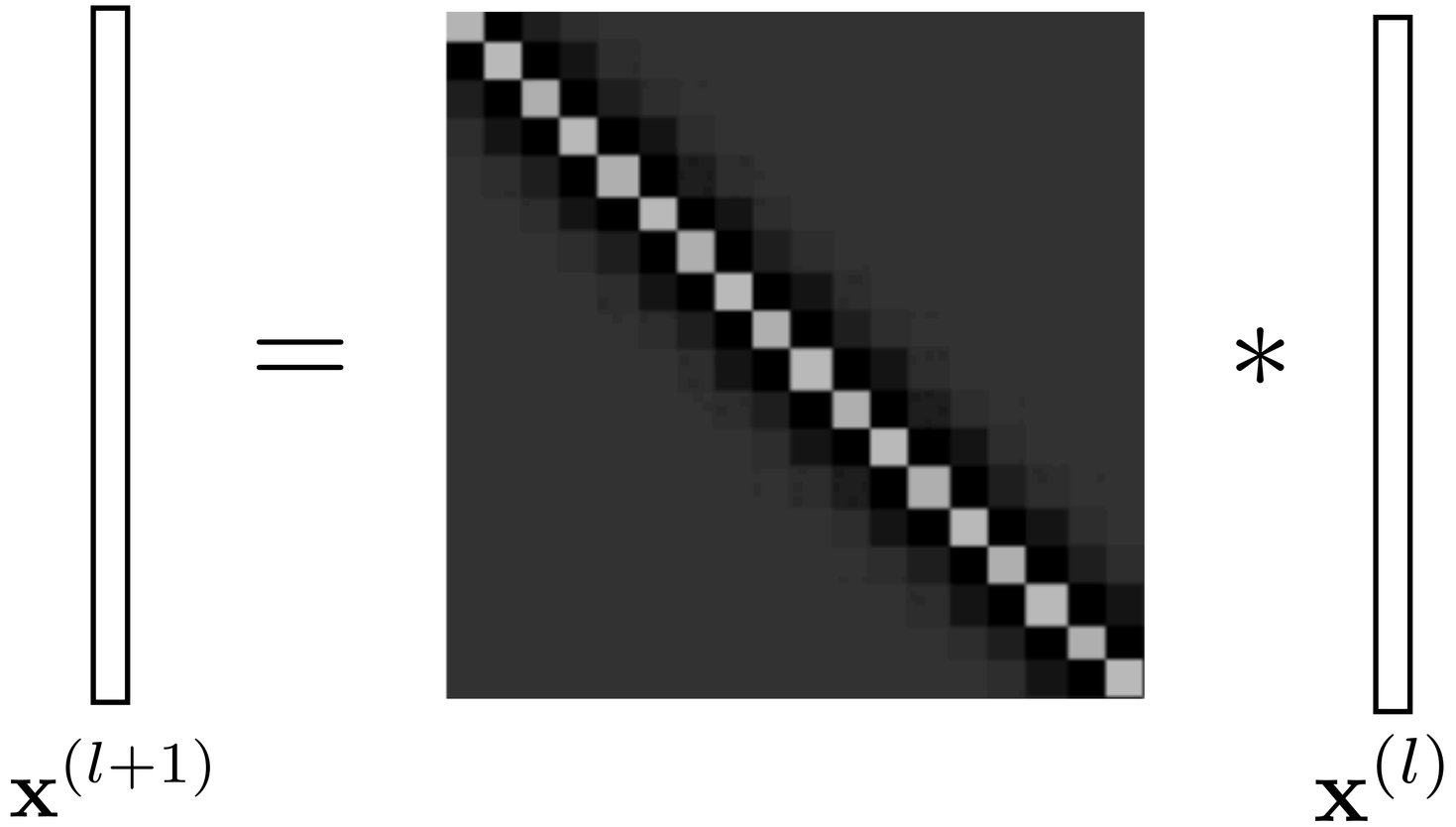
Toeplitz matrix

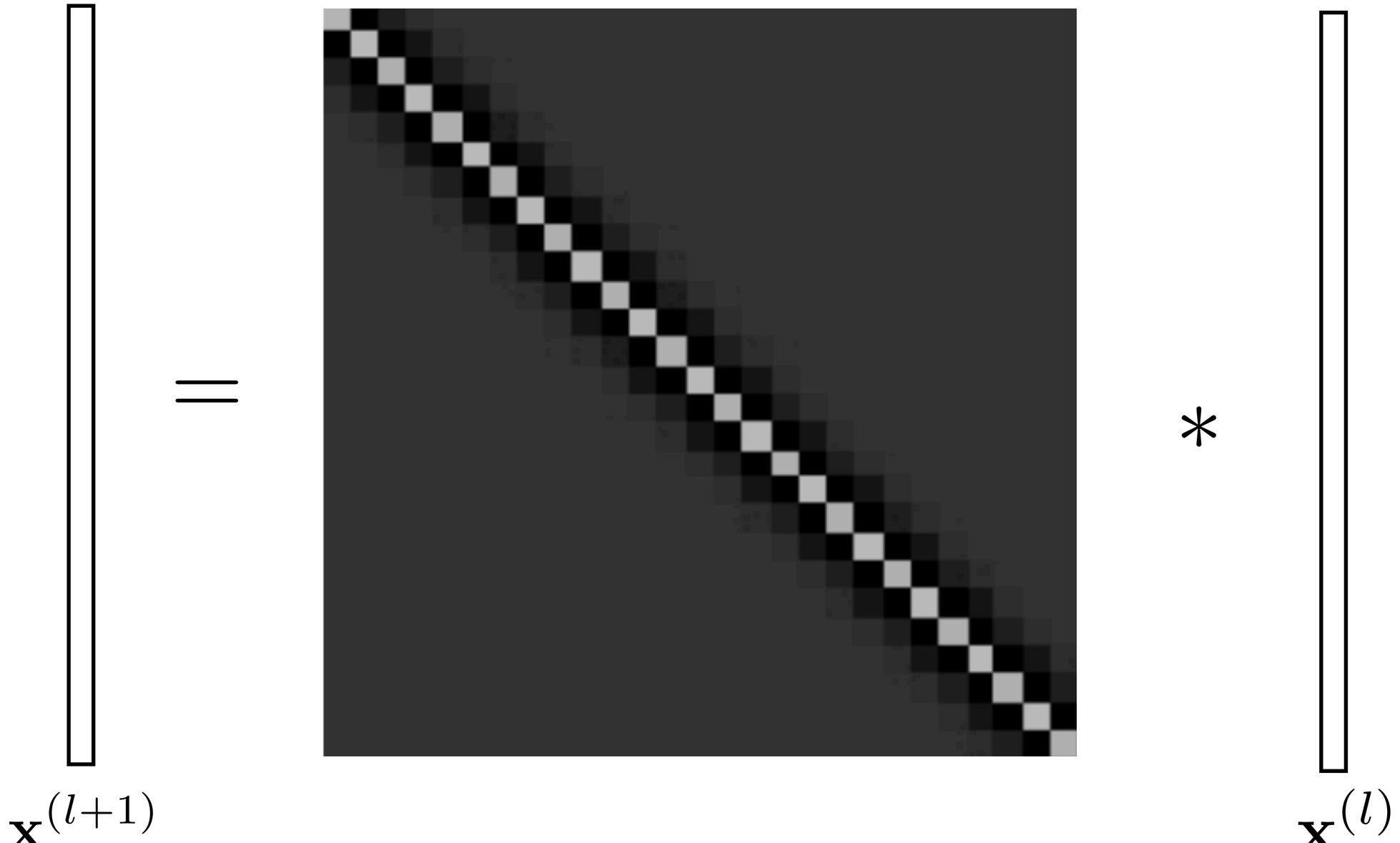
$$\begin{pmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{pmatrix}$$

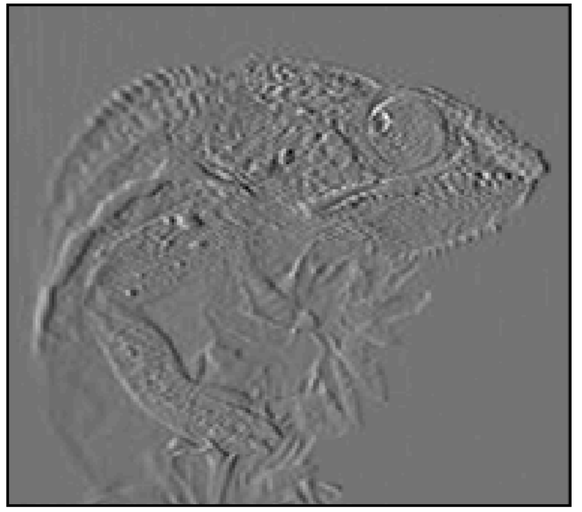


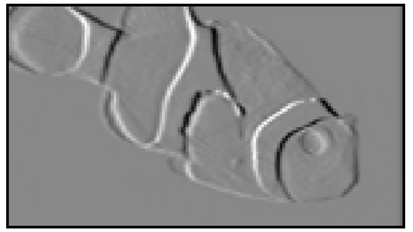
e.g., pixel image

- Constrained linear layer (infinitely strong regularization)
- Fewer parameters —> easier to learn, less overfitting


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


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

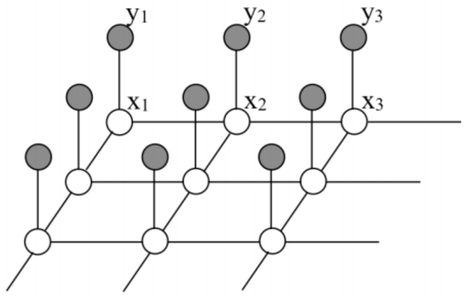




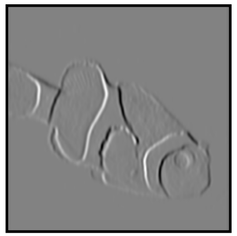
Conv layers can be applied to arbitrarily-sized inputs

Five views on convolutional layers

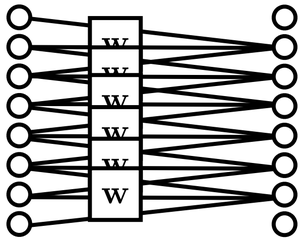
1. Equivariant with translation (stationarity) $f(\texttt{translate}(x)) = \texttt{translate}(f(x))$
2. Patch processing (Markov assumption)



3. Image filter

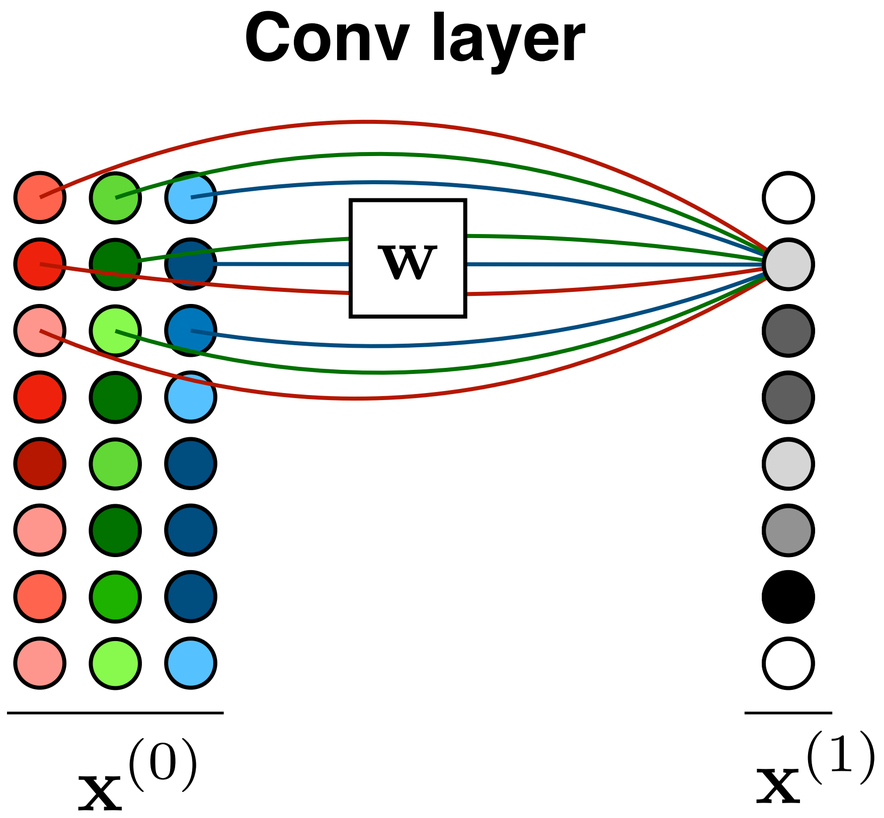


4. Parameter sharing



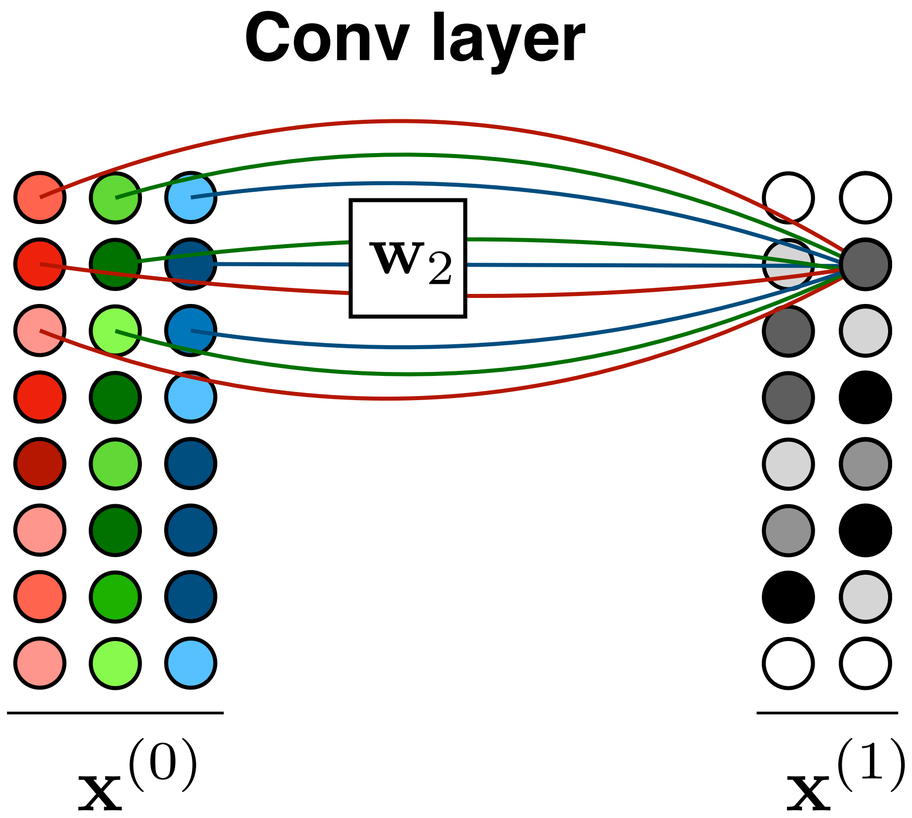
5. A way to process variable-sized tensors

Multiple channels



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

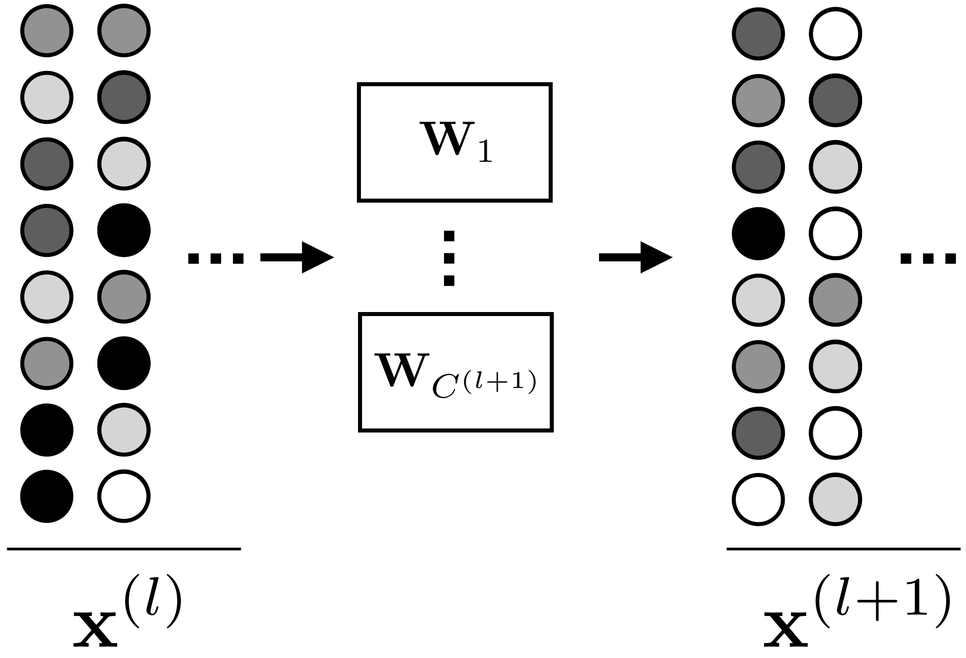
Multiple channels



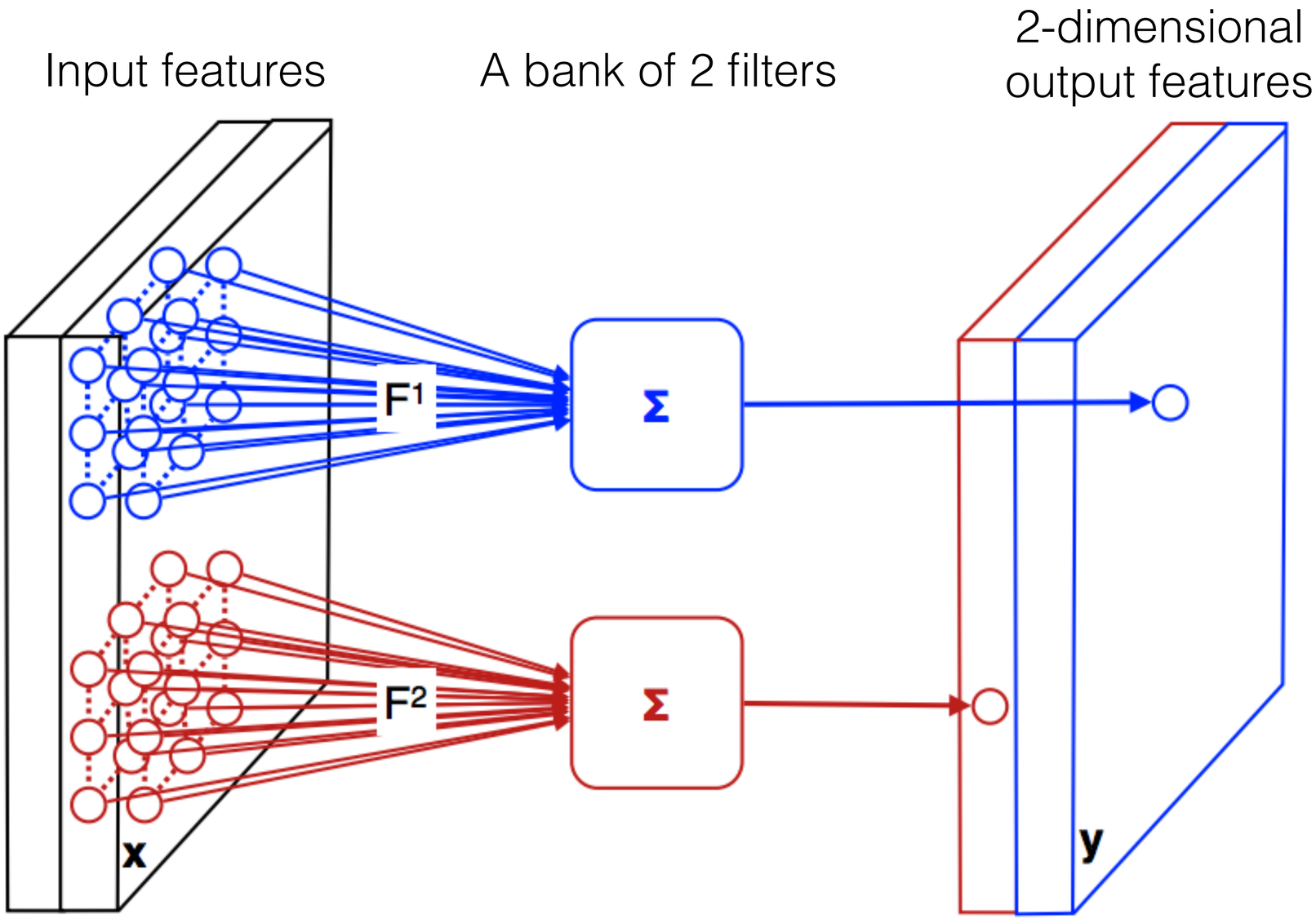
$$\mathbb{R}^{N \times C^{(0)}} \rightarrow \mathbb{R}^{N \times C^{(1)}}$$

Multiple channels

Conv layer



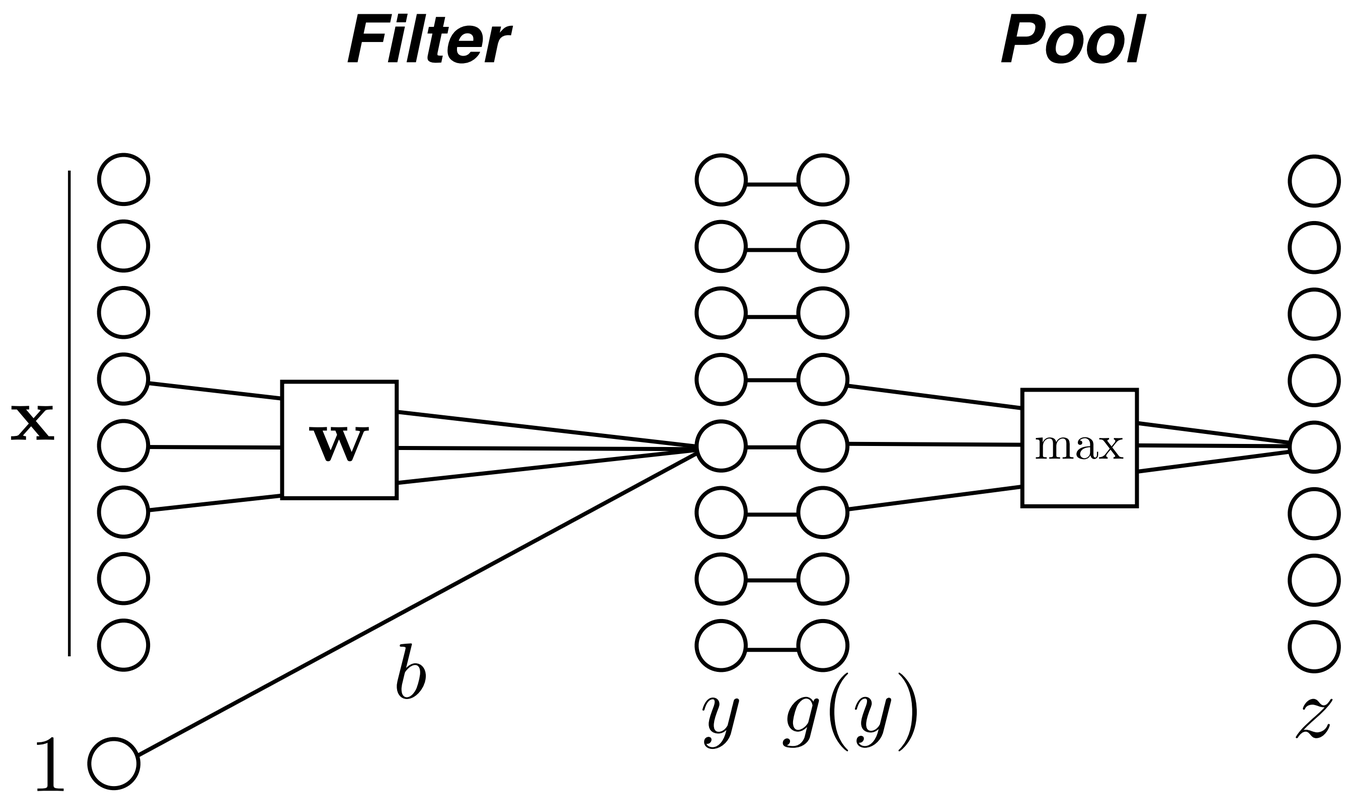
$$\mathbb{R}^{N \times C^{(l)}} \rightarrow \mathbb{R}^{N \times C^{(l+1)}}$$



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

[Figure from Andrea Vedaldi]

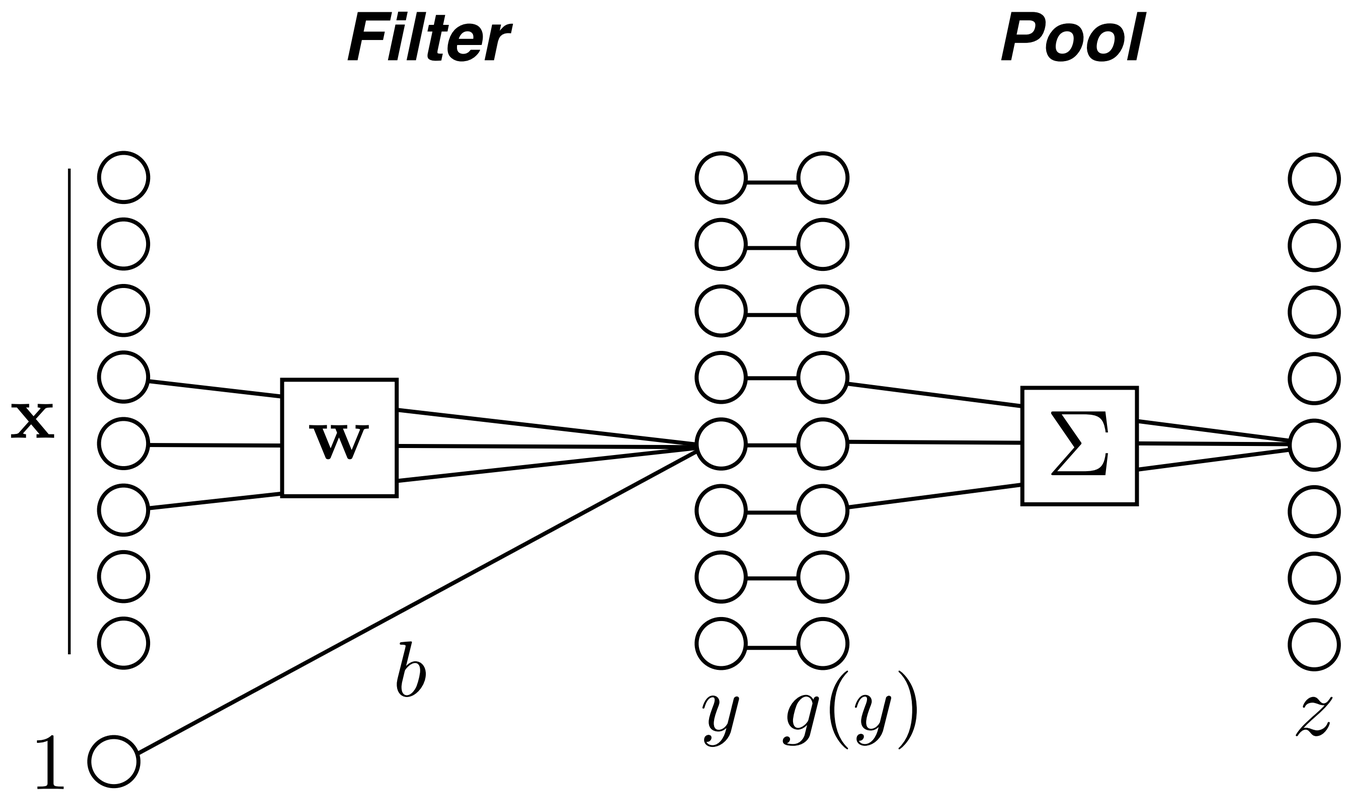
Pooling



Max pooling

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

Pooling



Max pooling

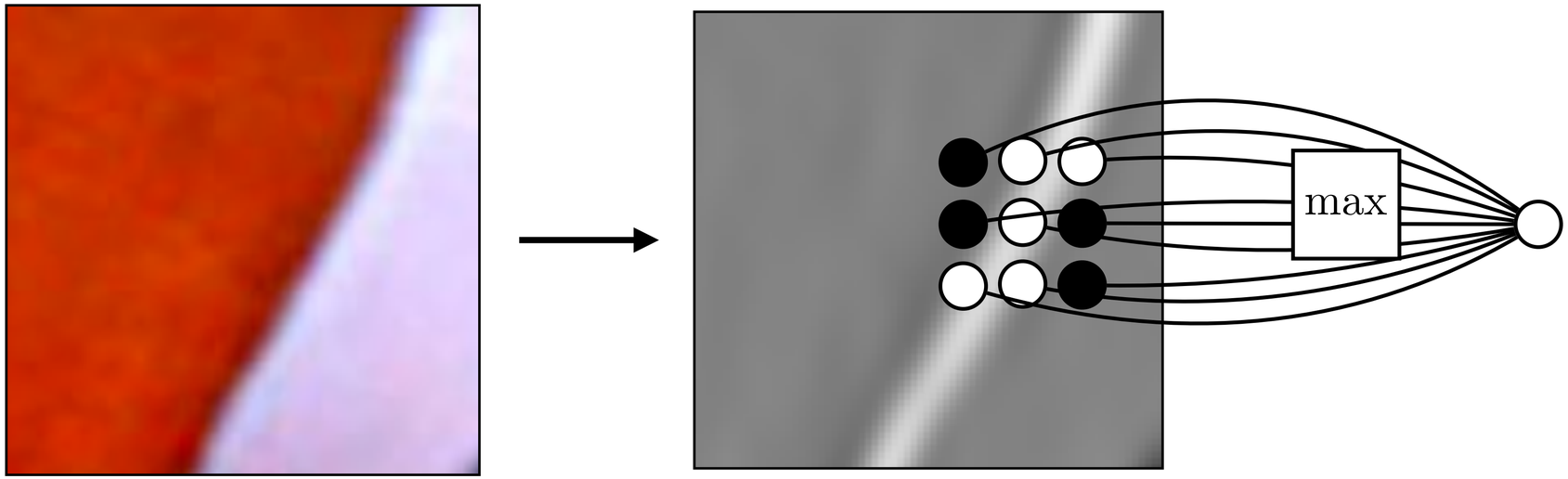
$$z_k = \max_{j \in \mathcal{N}(j)} g(y_j)$$

Mean pooling

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

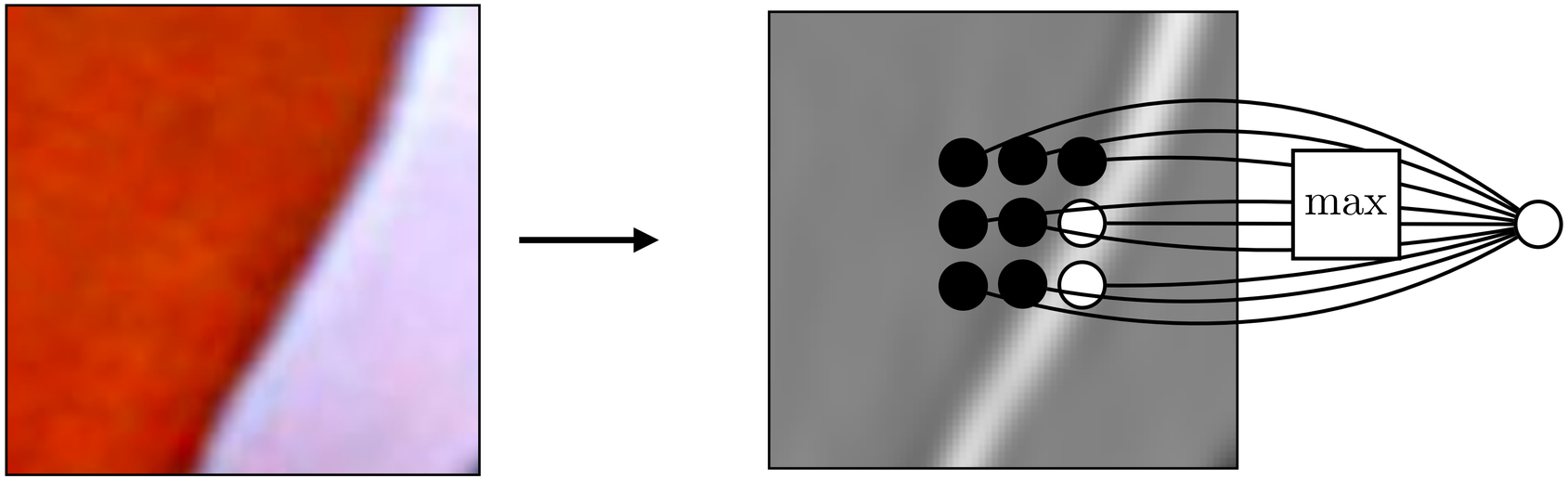
Pooling — Why?

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



Pooling — Why?

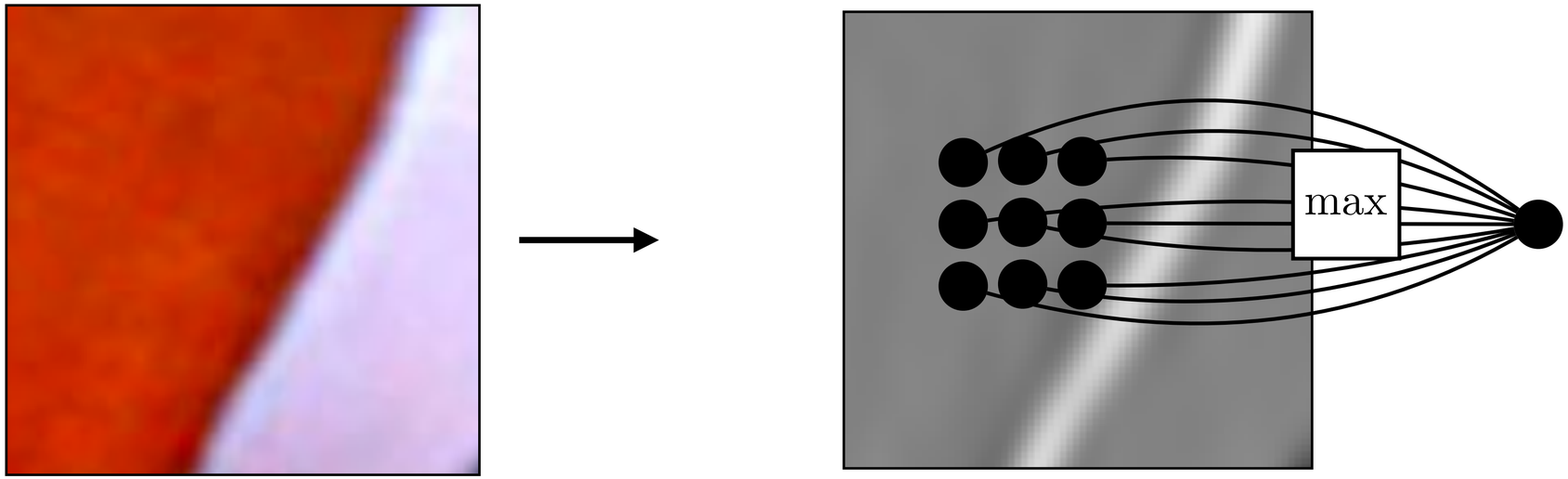
Pooling across spatial locations achieves stability w.r.t. small translations:



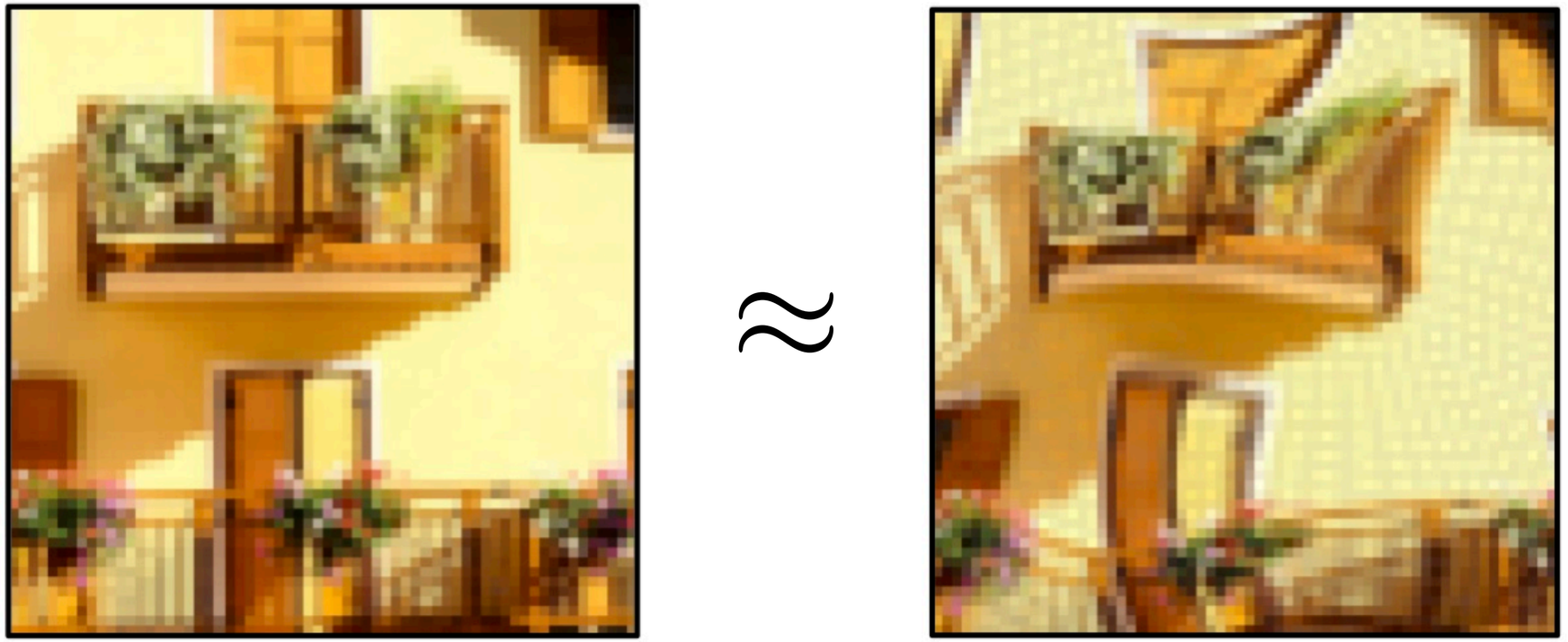
large response regardless of exact position of edge

Pooling — Why?

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



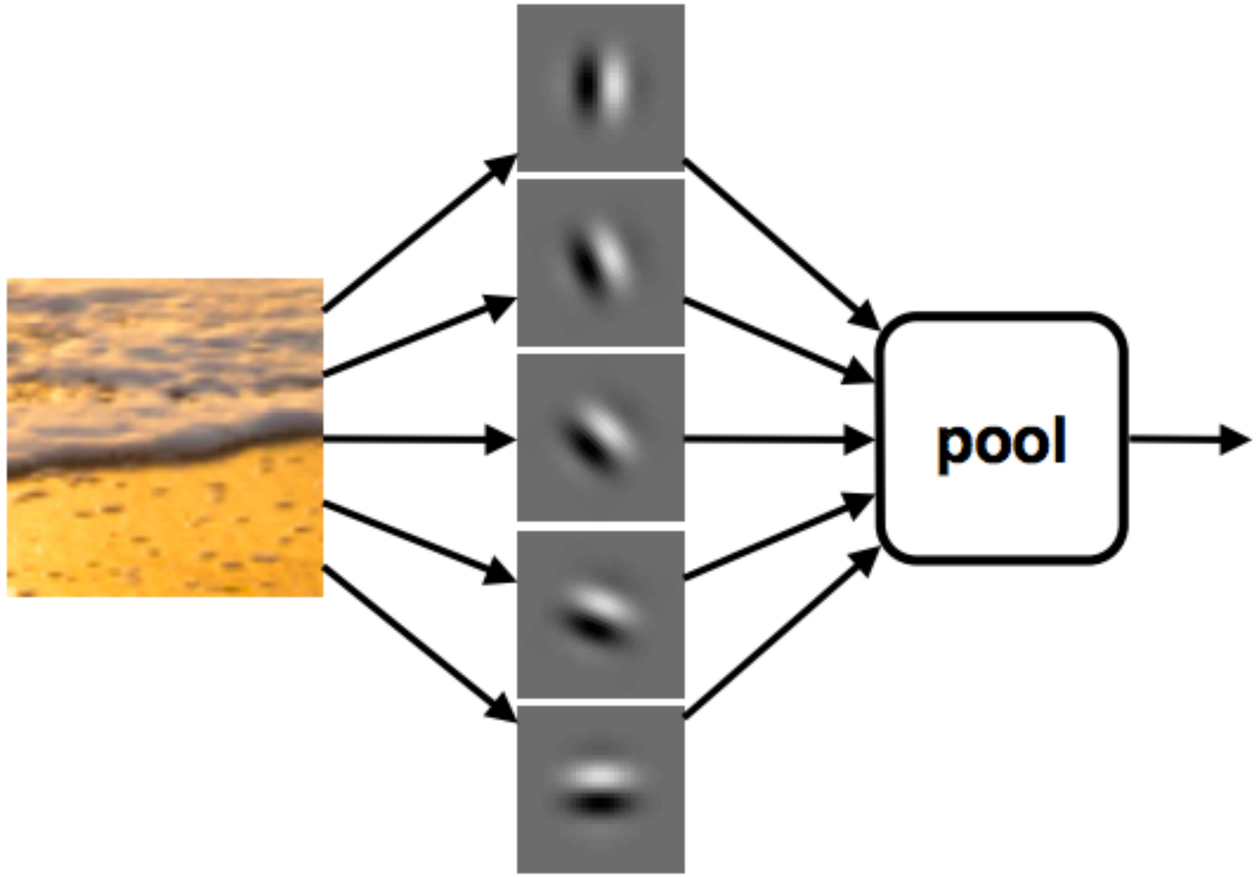
CNNs are stable w.r.t. diffeomorphisms



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

Pooling — Why?

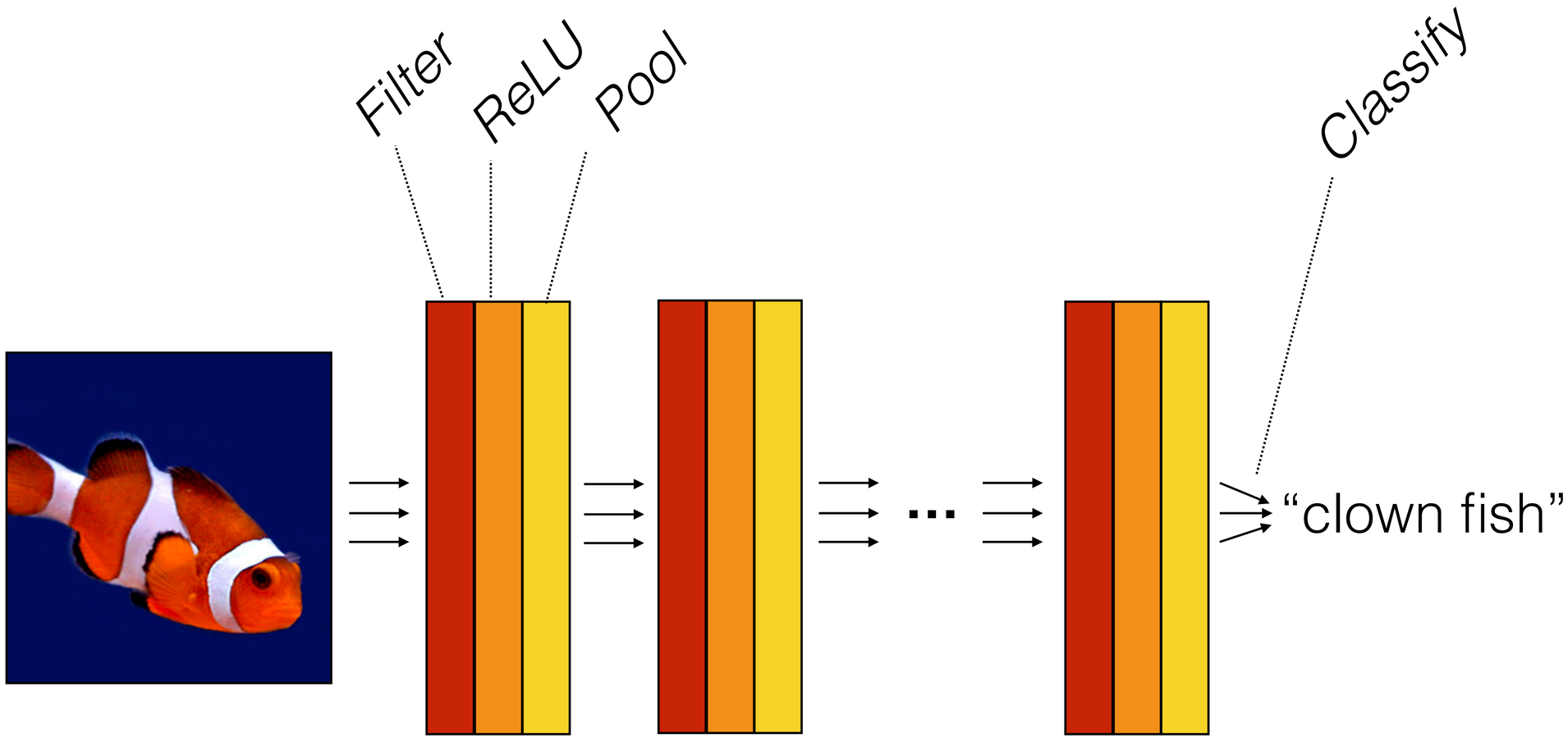
Pooling across feature channels (filter outputs) can achieve other kinds of invariances:



large response for any edge, regardless of its orientation

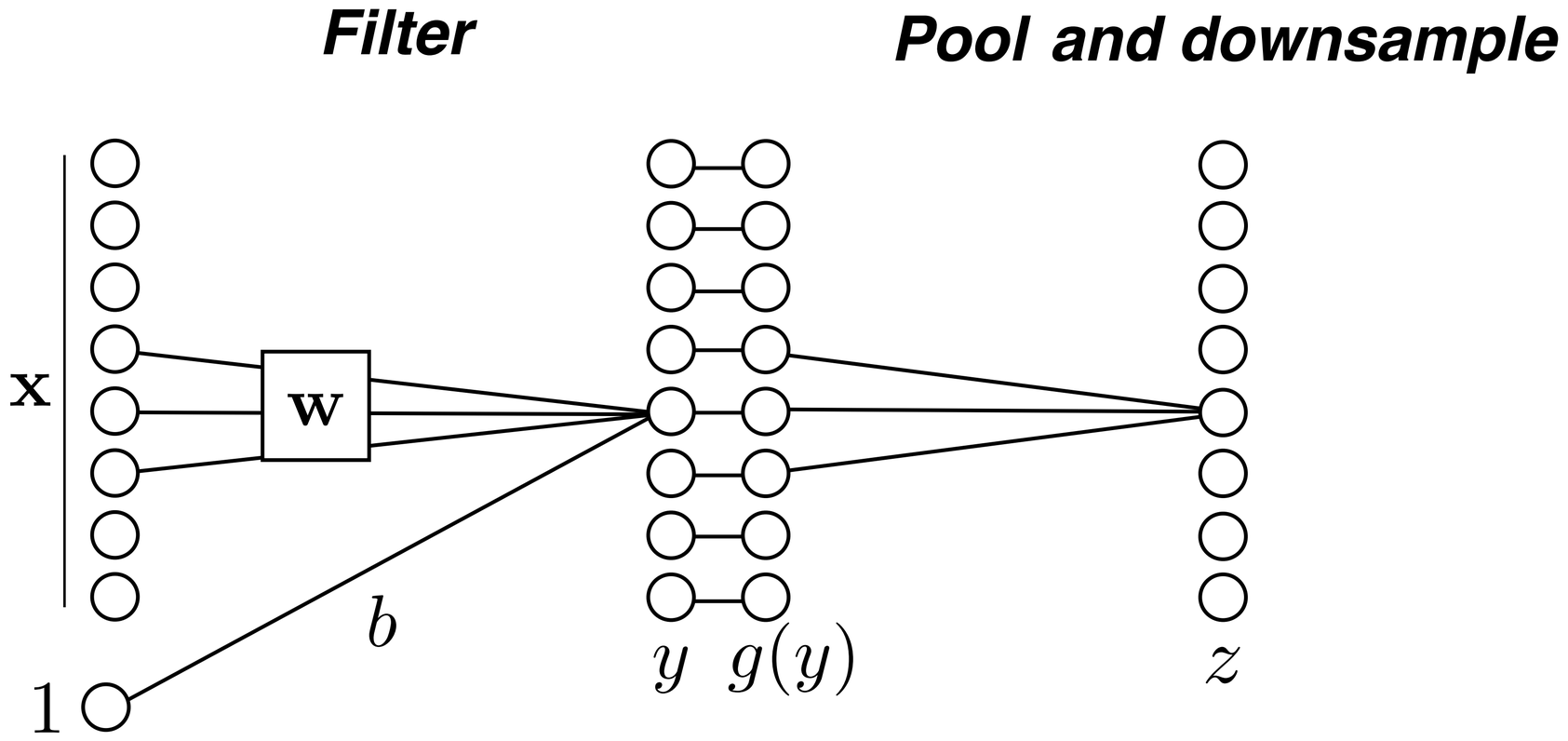
[Derived from slide by Andrea Vedaldi]

Computation in a neural net

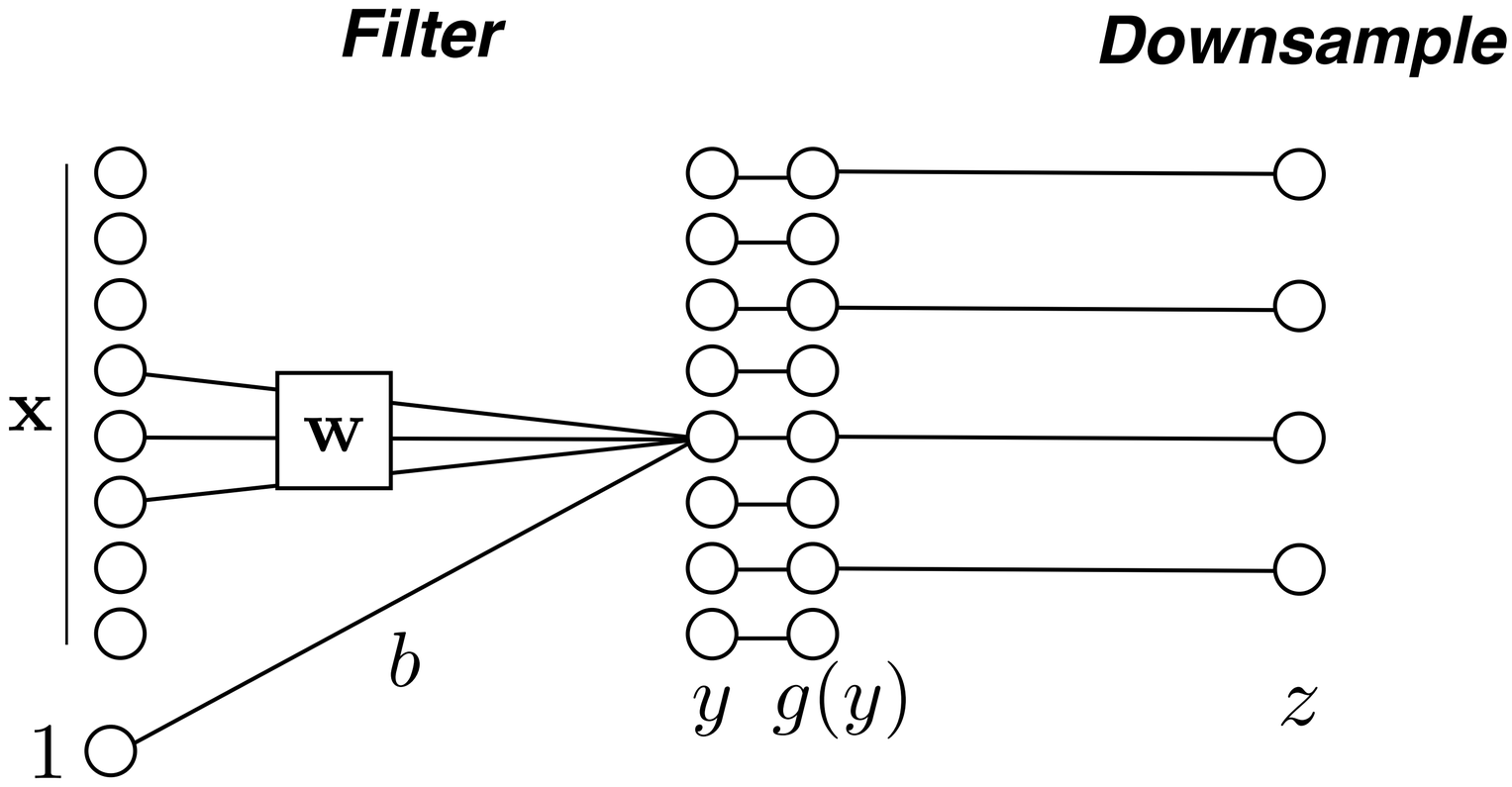


$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

Downsampling

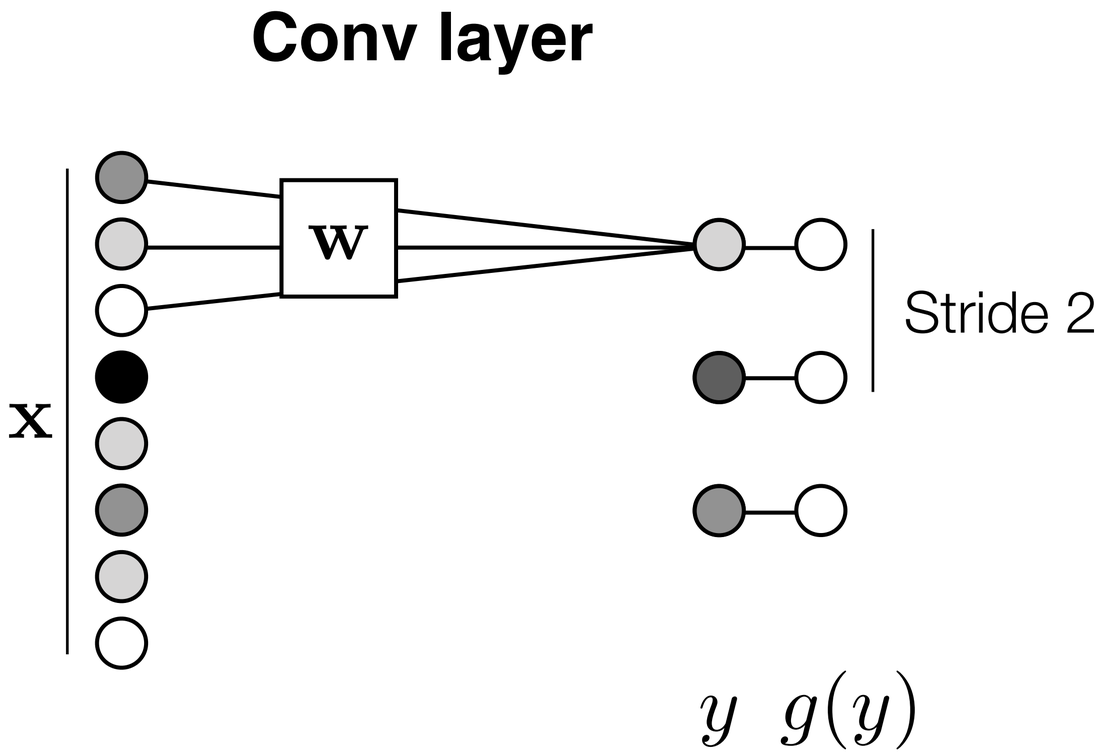


Downsampling



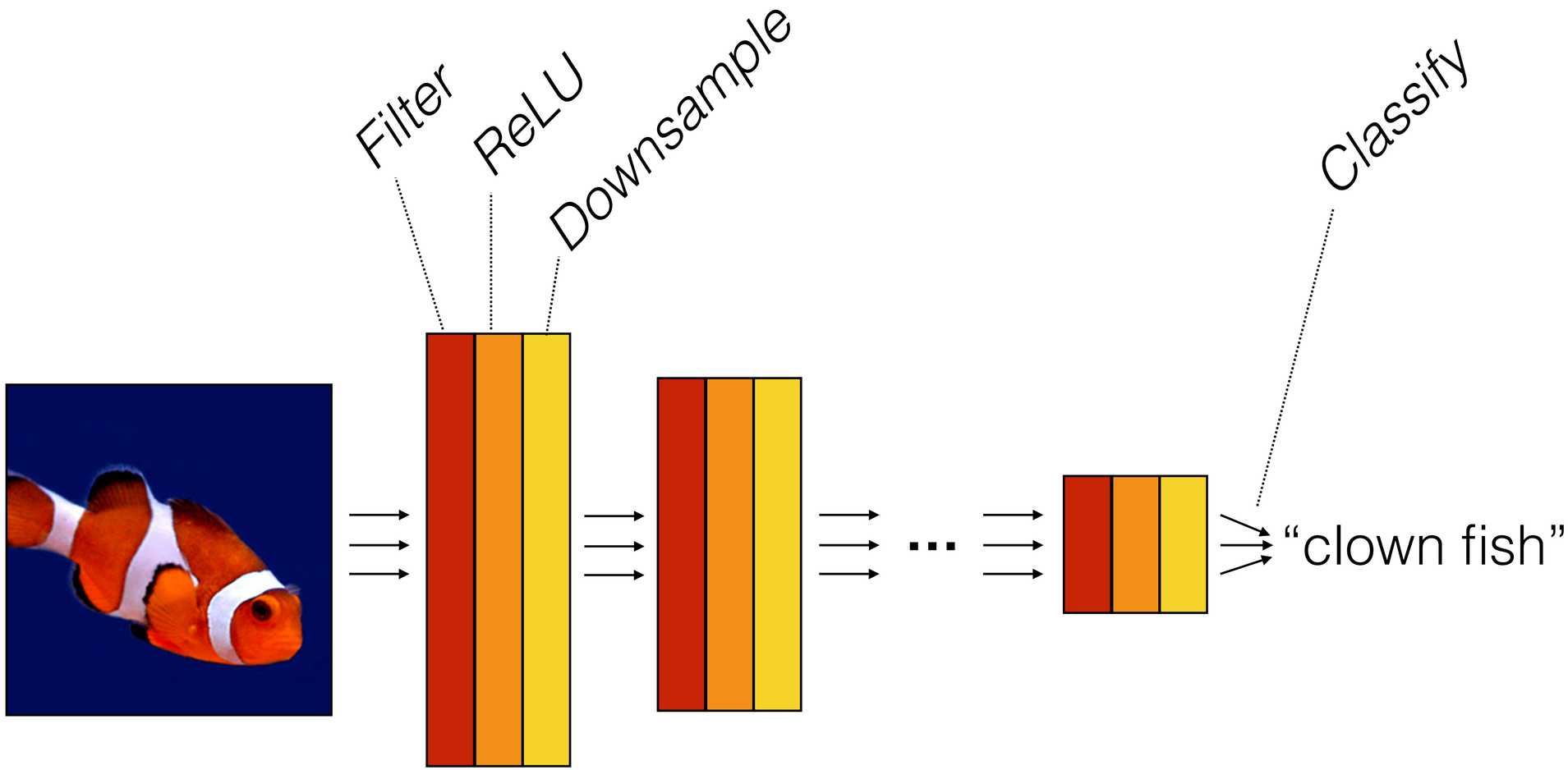
$$\mathbb{R}^{H^{(l)} \times W^{(l)} \times C^{(l)}} \rightarrow \mathbb{R}^{H^{(l+1)} \times W^{(l+1)} \times C^{(l+1)}}$$

Strided convolution



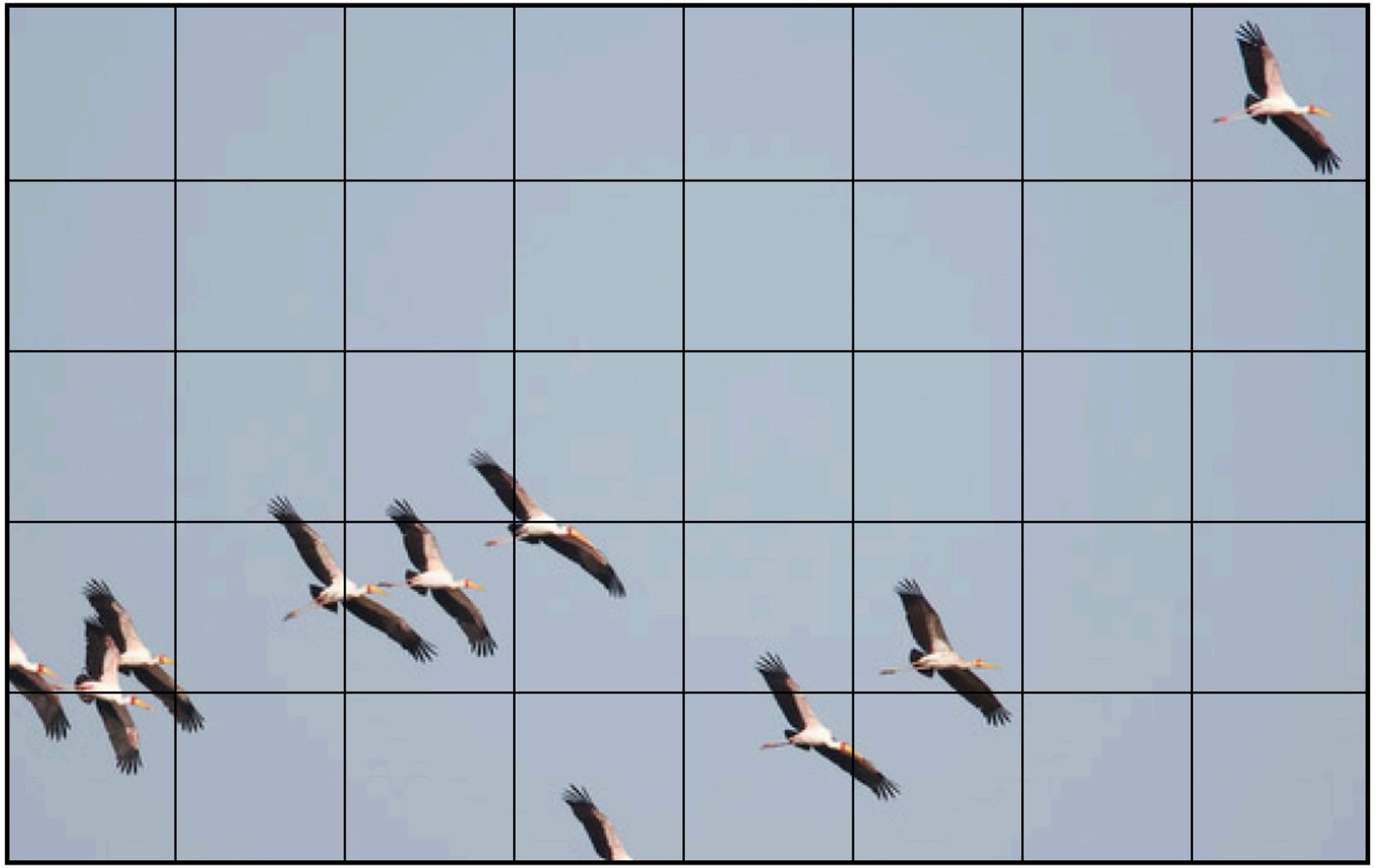
Strided convolutions combine convolution and downsampling into a single operation.

Computation in a neural net



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

Receptive fields



Receptive fields

Pool and downsample by 2

3x1 Filter

Pool and downsample by 2

Sky

Sky

Bird

Sky

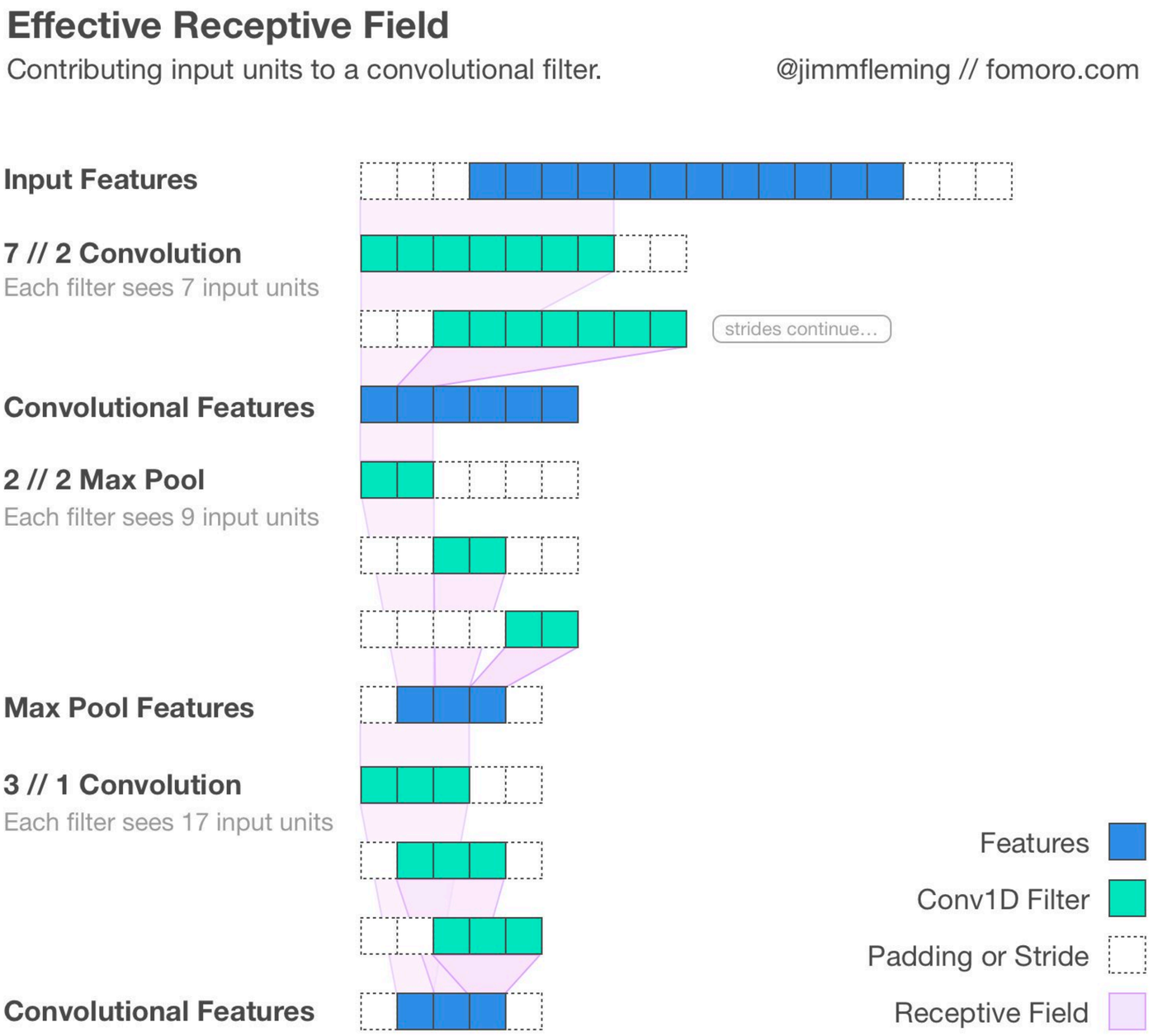
RF = RF*2

RF = RF + floor(3/2)*2

RF = RF*2

kernel size

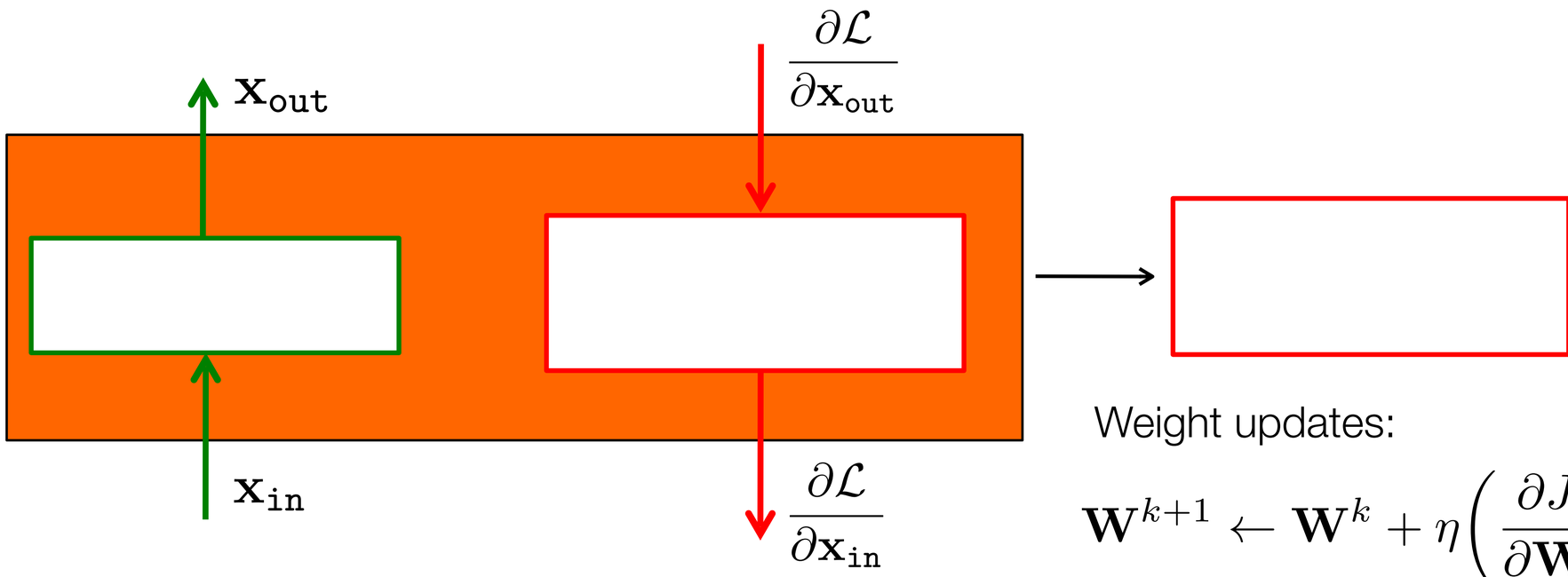
scale factor



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

Miniplaces part 1: Convolution Module

Assume the input x_{in} and output x_{out} are 1D signals of the same length N.
The convolution kernel is w_i, and has length M < N



Weight updates:

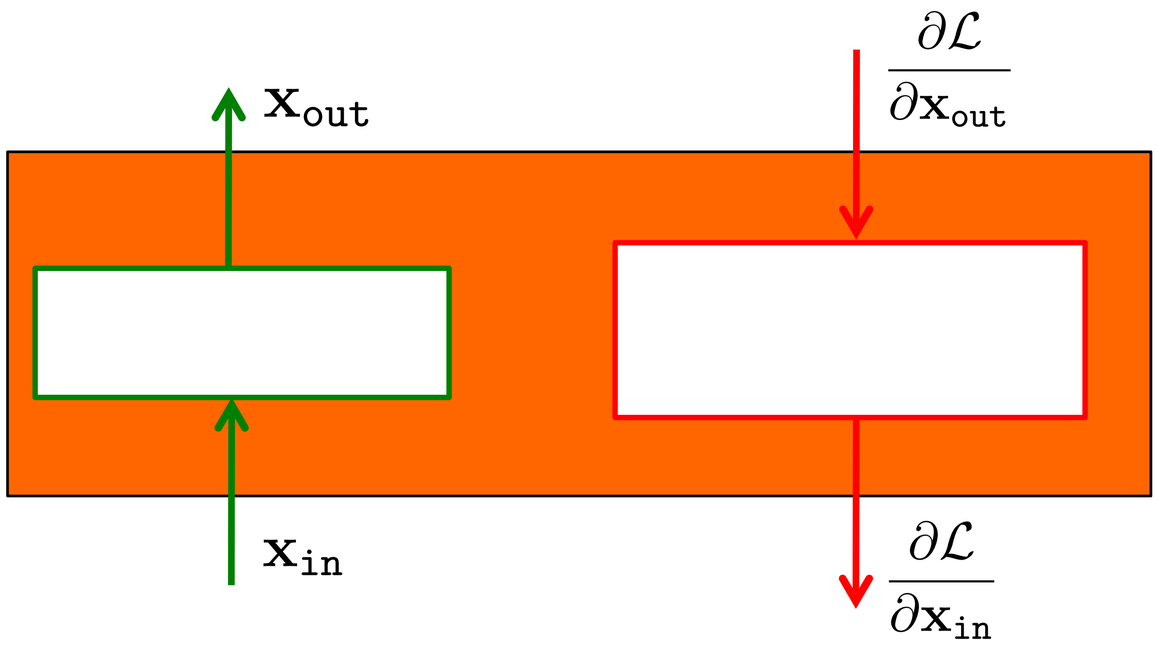
$$\mathbf{W}^{k+1} \leftarrow \mathbf{W}^{k} + \eta \left(\frac{\partial J}{\partial \mathbf{W}} \right)^{T}$$

Derive the equations that go inside each box.
Discuss how you handle the boundaries.

Miniplaces challenge part 1: max pooling module

(grad course, optional for undergrads)

Assume the input x_{in} and output x_{out} are 1D signals of different lengths.



Weight updates:

$$\mathbf{W}^{k+1} \leftarrow \mathbf{W}^{k} + \eta \left(\frac{\partial J}{\partial \mathbf{W}} \right)^{T}$$

Derive the equations that go inside each box.
Discuss how you handle the boundaries.

CNNs — Why?

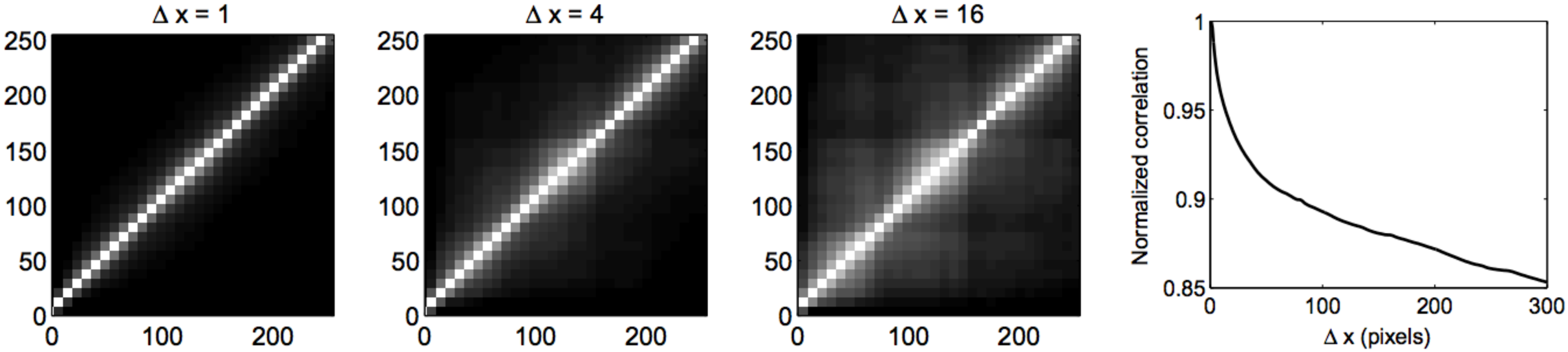
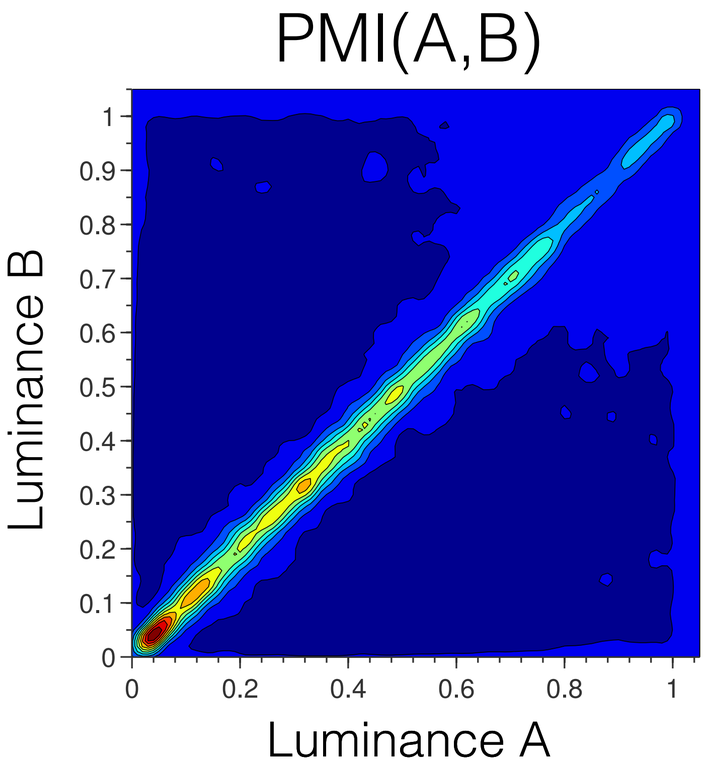


Fig. 1. (a) Scatterplots of pairs of pixels at three different spatial displacements, averaged over five examples images. (b) Autocorrelation function. Photographs are of New York City street scenes, taken with a Canon 10D digital camera, and processed in RAW linear sensor mode (producing pixel intensities are in roughly proportional to light intensity). Correlations were computed on the logs of these sensor intensity values [41].

[http://6.869.csail.mit.edu/fa18/notes/simoncelli2005.pdf]



Pointwise mutual information (PMI)

$$\mathrm{PMI}(A, B) = \log \frac{P(A, B)}{P(A)P(B)}$$

["Crisp boundary detection using pointwise mutual information", Isola et al. 2014]

CNNs — Why?

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

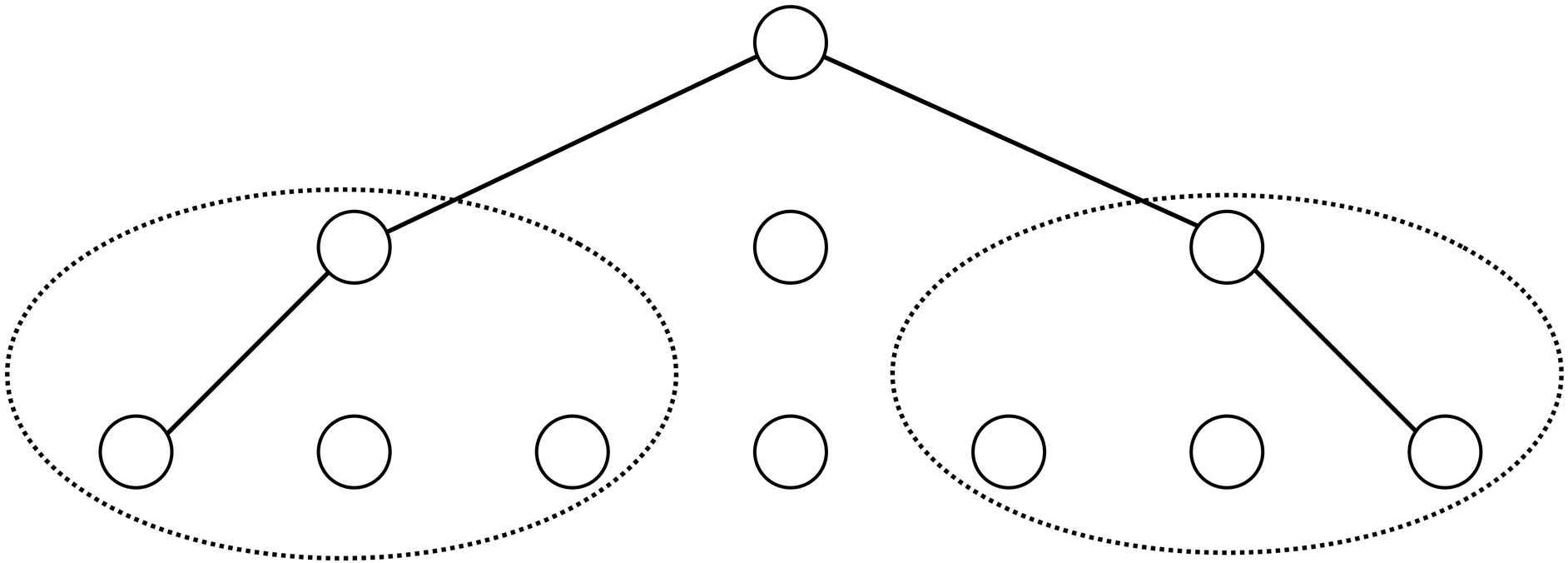
It is therefore often sufficient to model local dependences only. —> **Convolution**

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

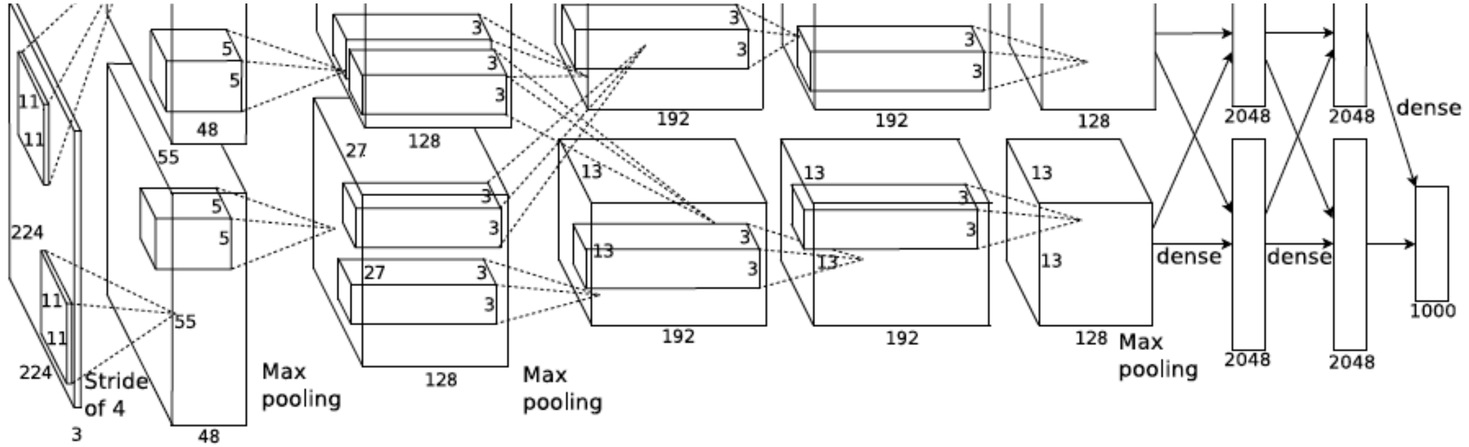
[For more discussion, see “Why does Deep and Cheap Learning Work So Well?”, Lin et al. 2017]

CNNs — Why?

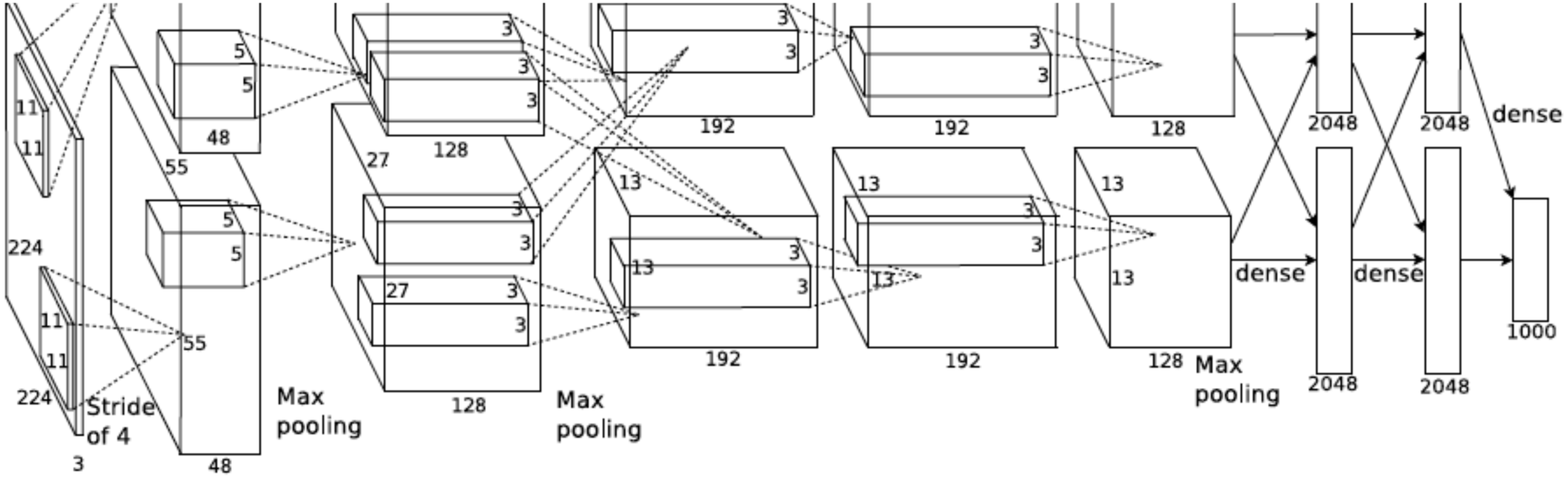
Capturing long-range dependences:



Alexnet — [Krizhevsky et al. NIPS 2012]



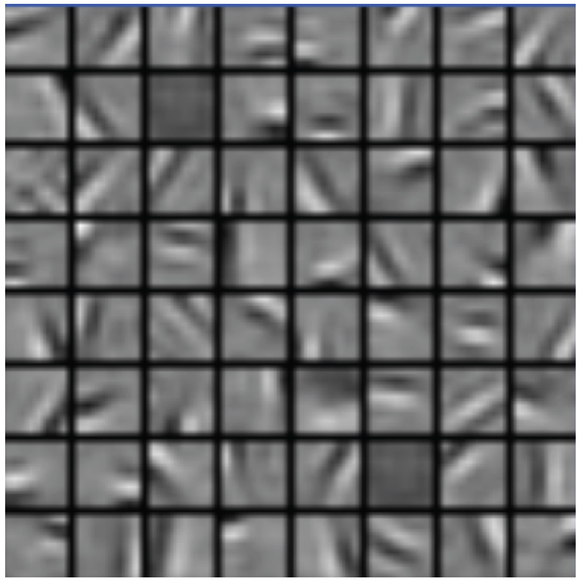
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] 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)



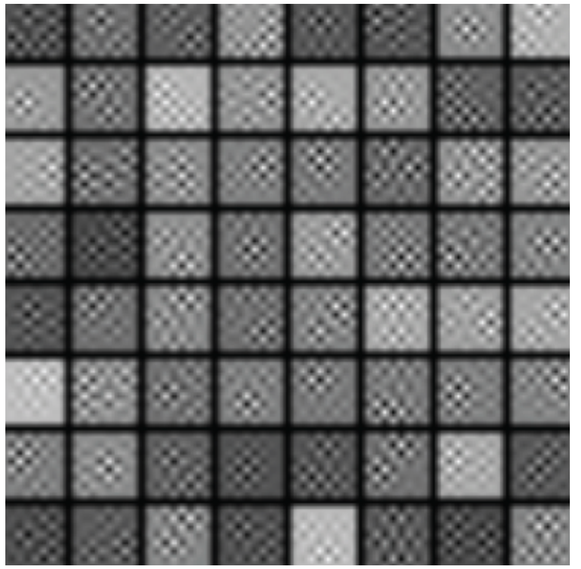
What filters are learned?

What filters are learned?

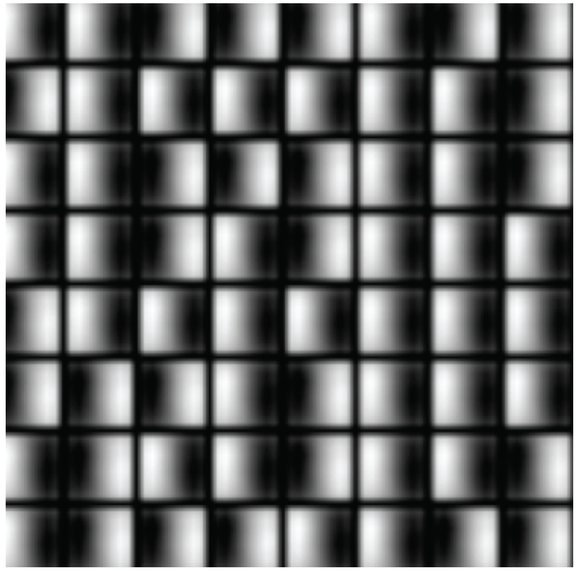
A



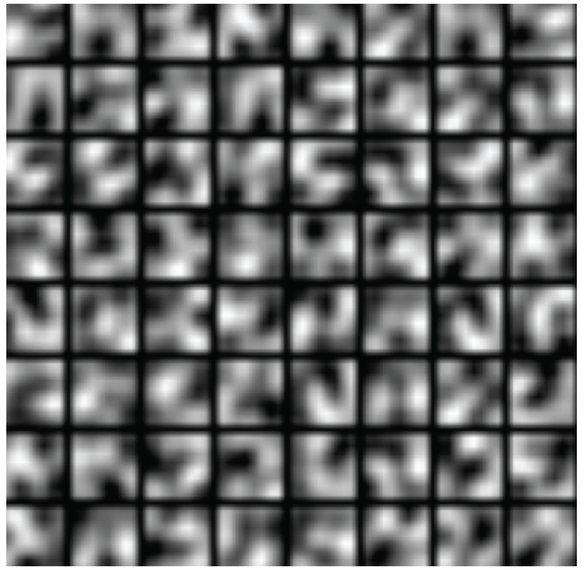
B



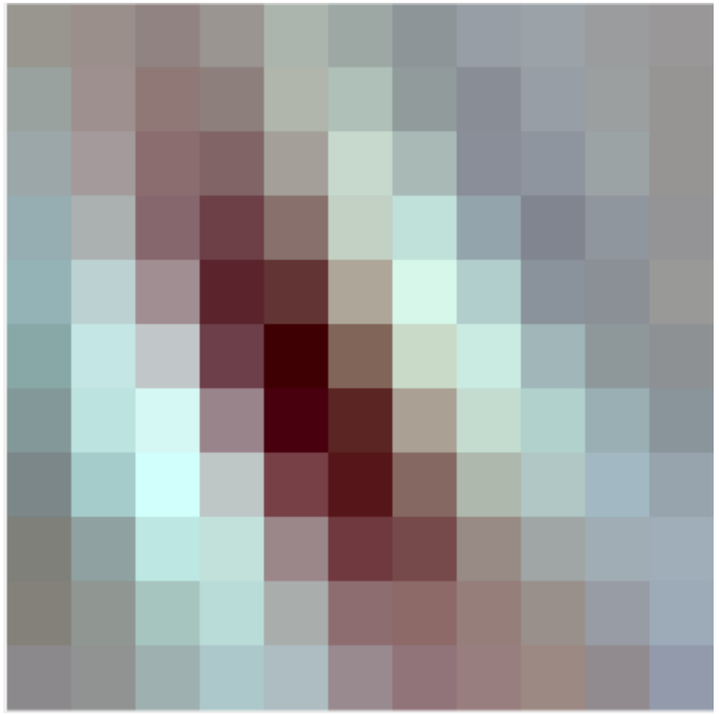
C



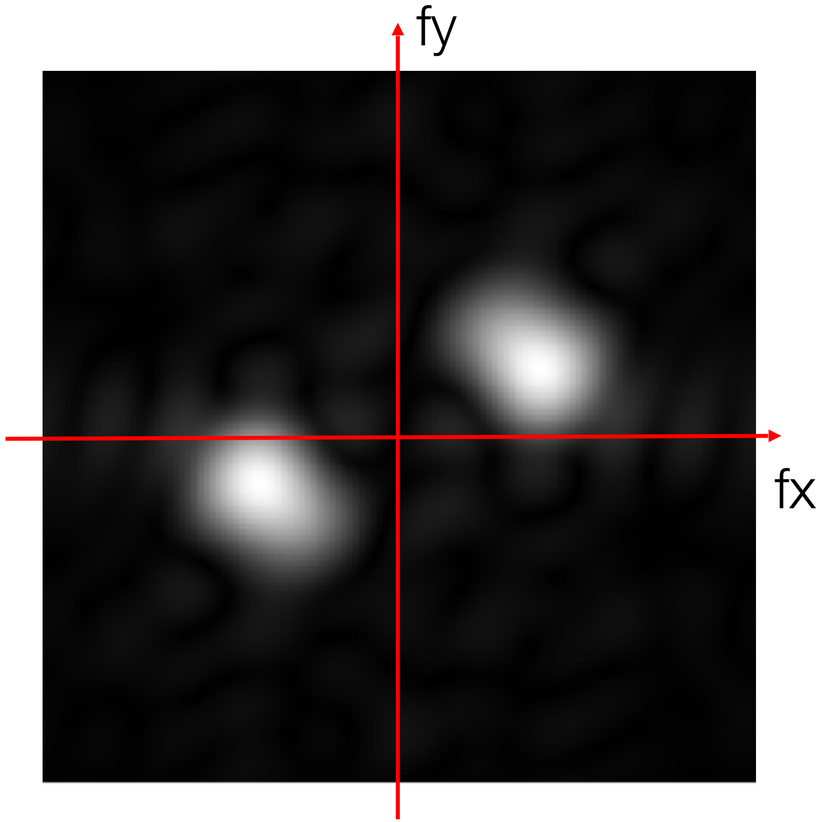
D



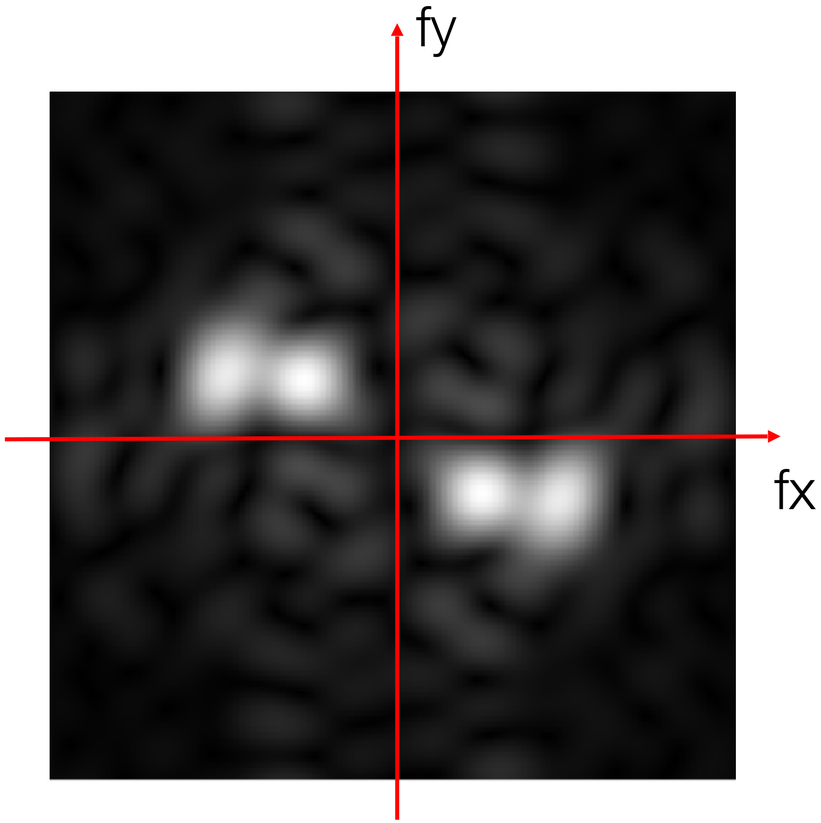
Get to know your units



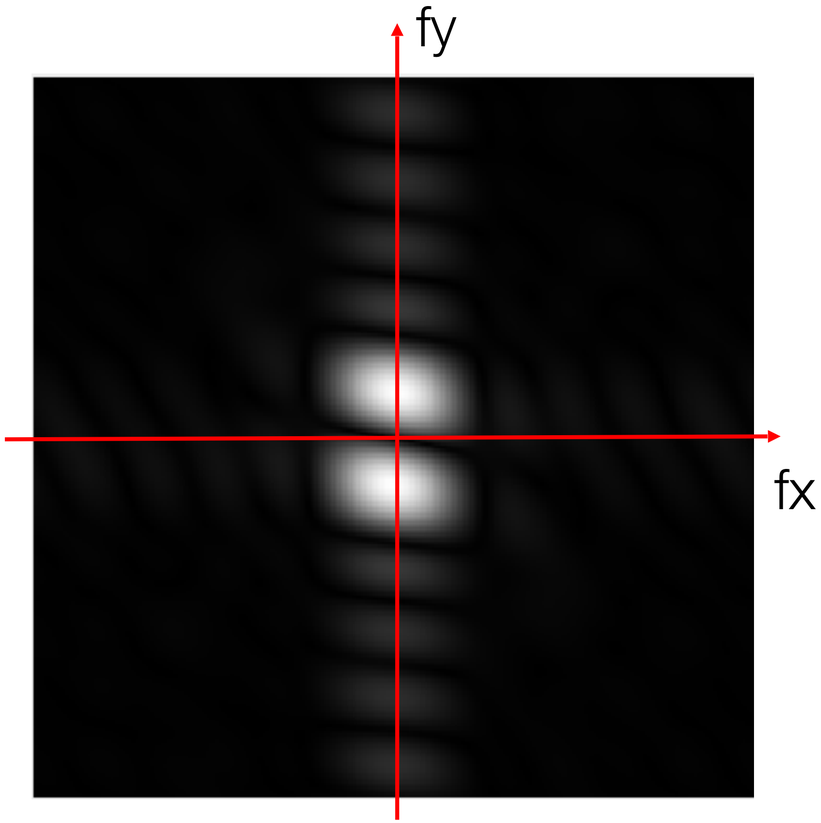
11x11 convolution kernel
(3 color channels)



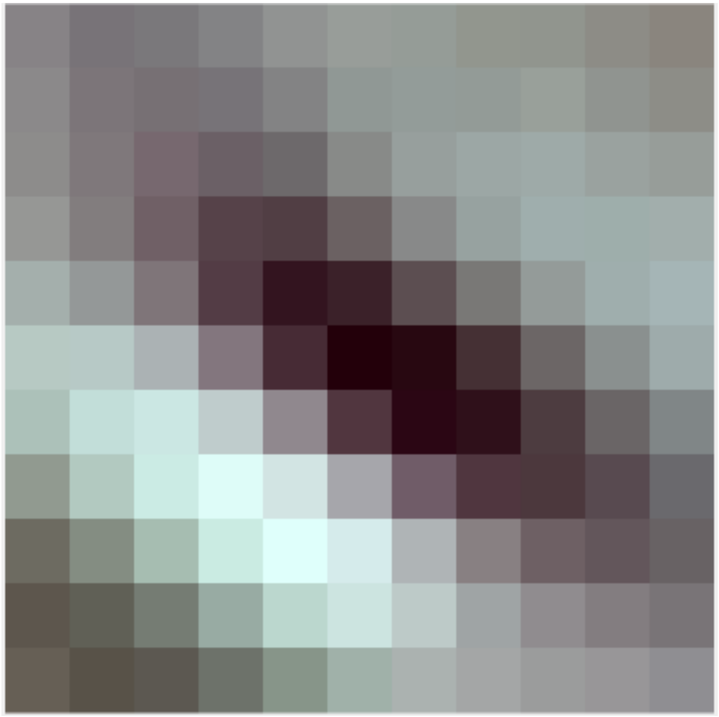
Get to know your units

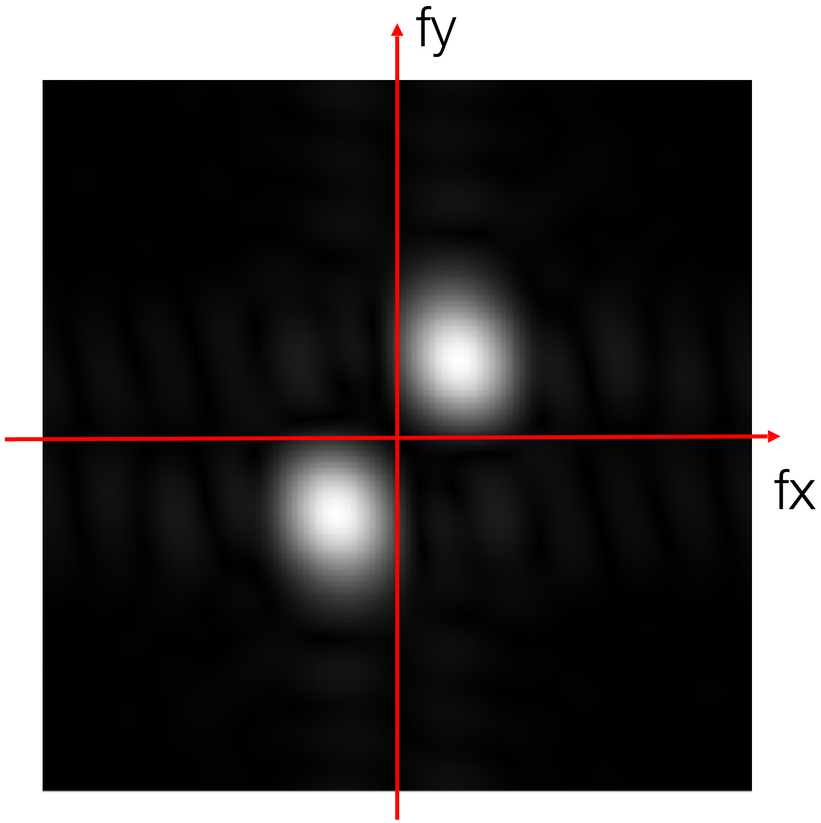


Get to know your units

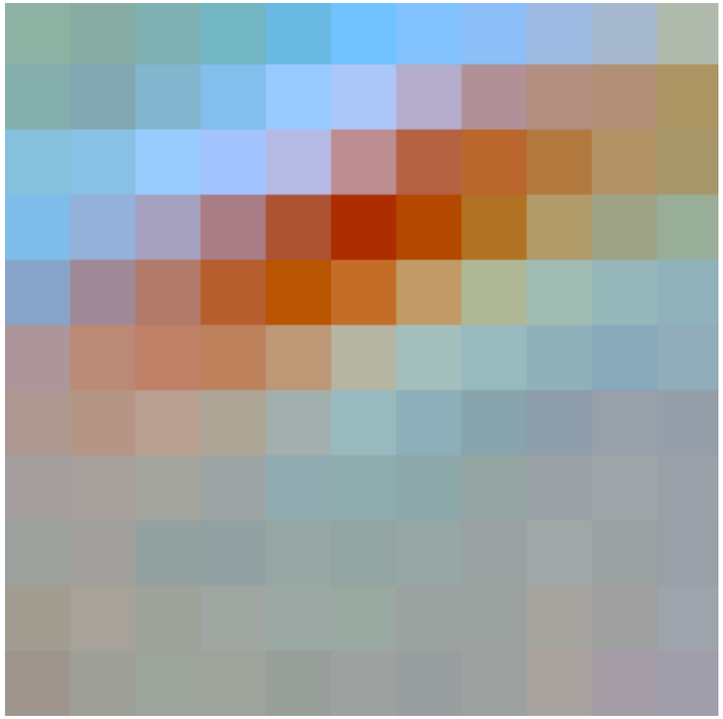


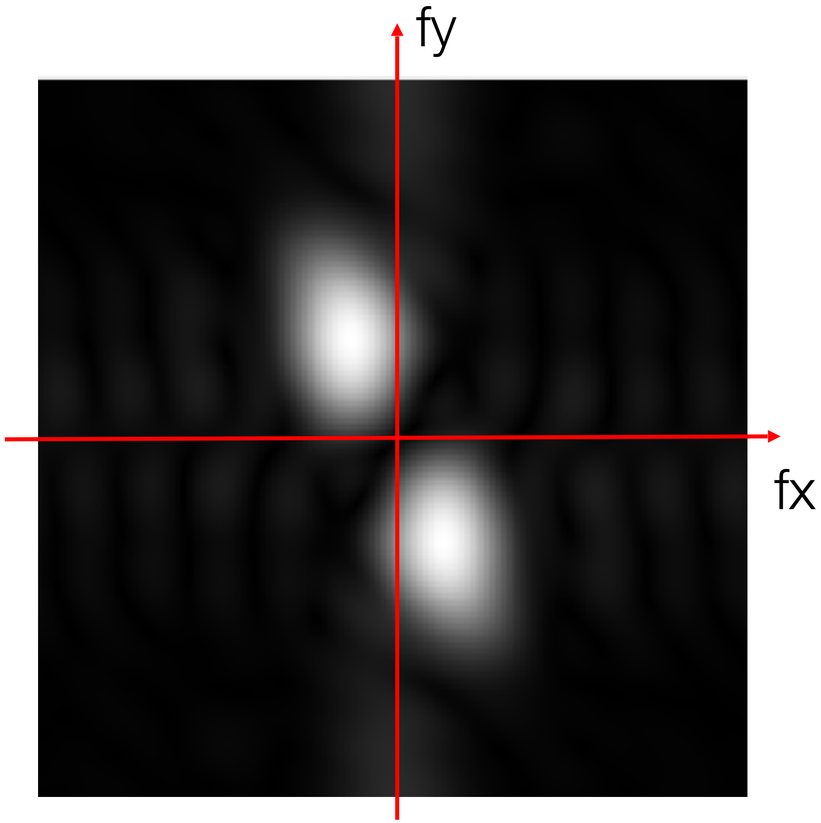
Get to know your units



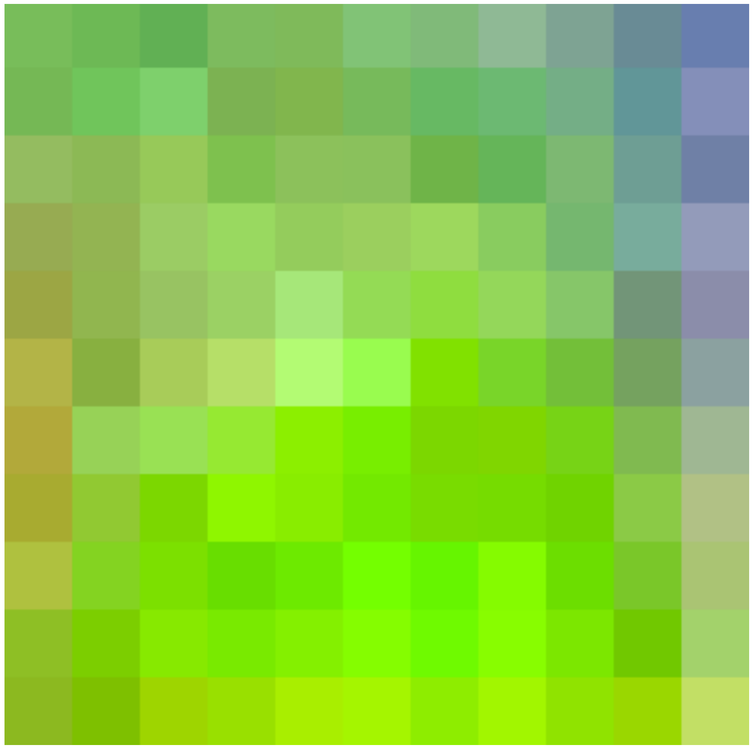


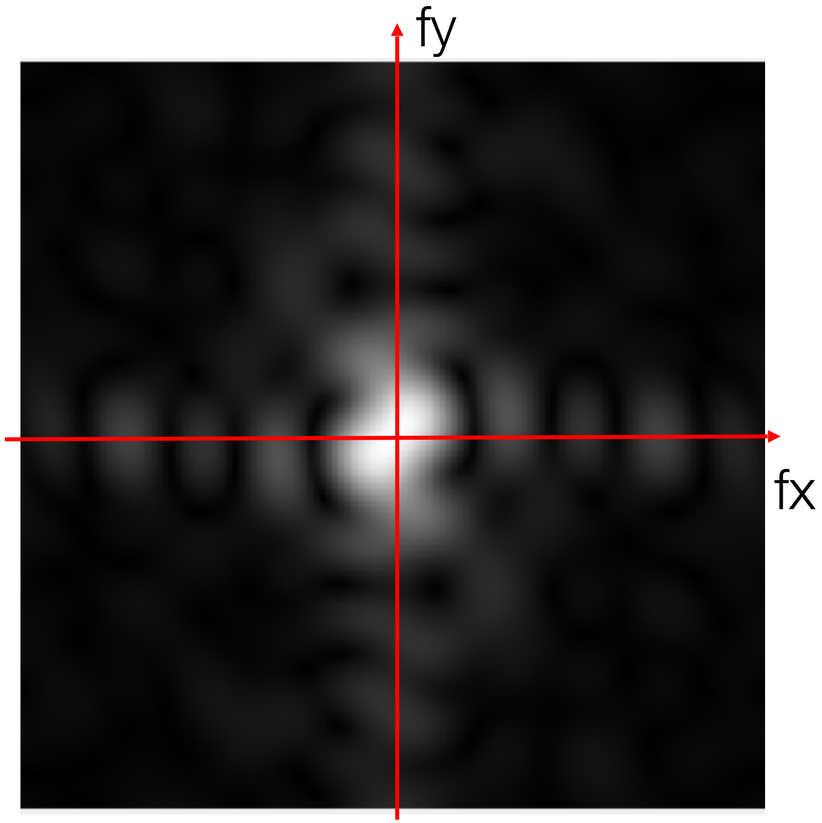
Get to know your units



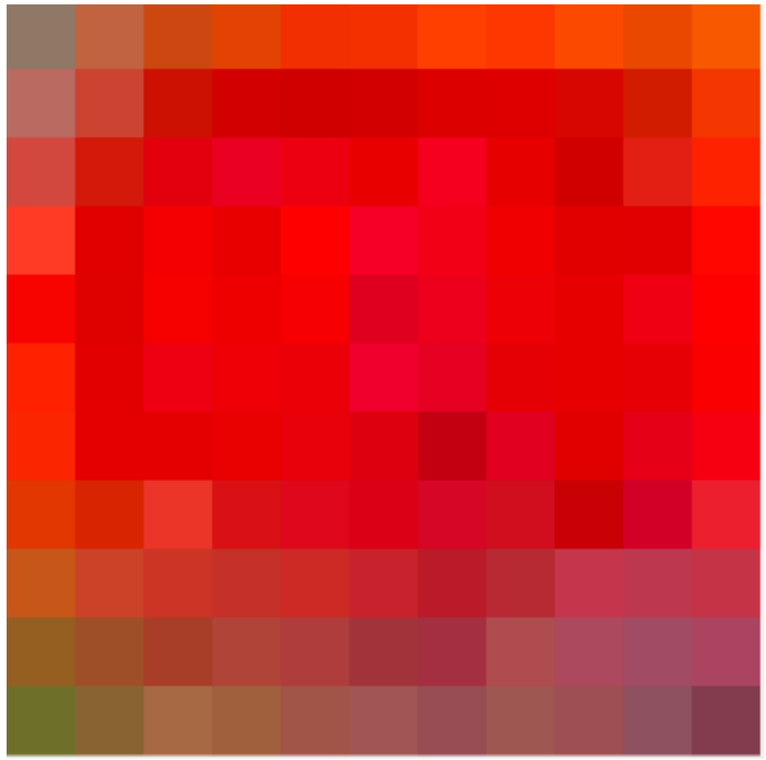


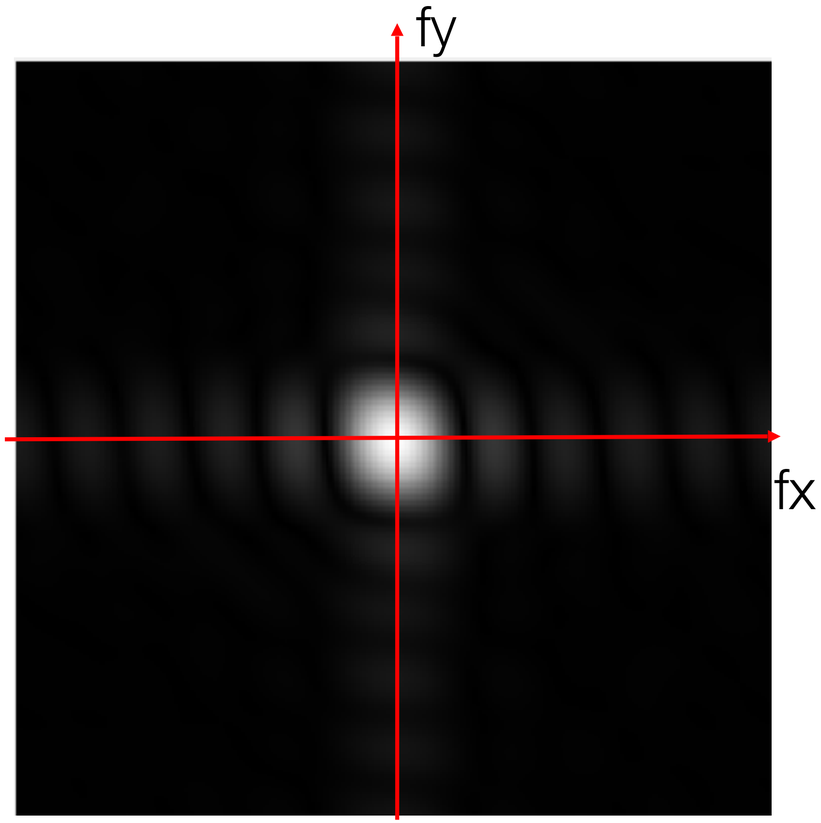
Get to know your units



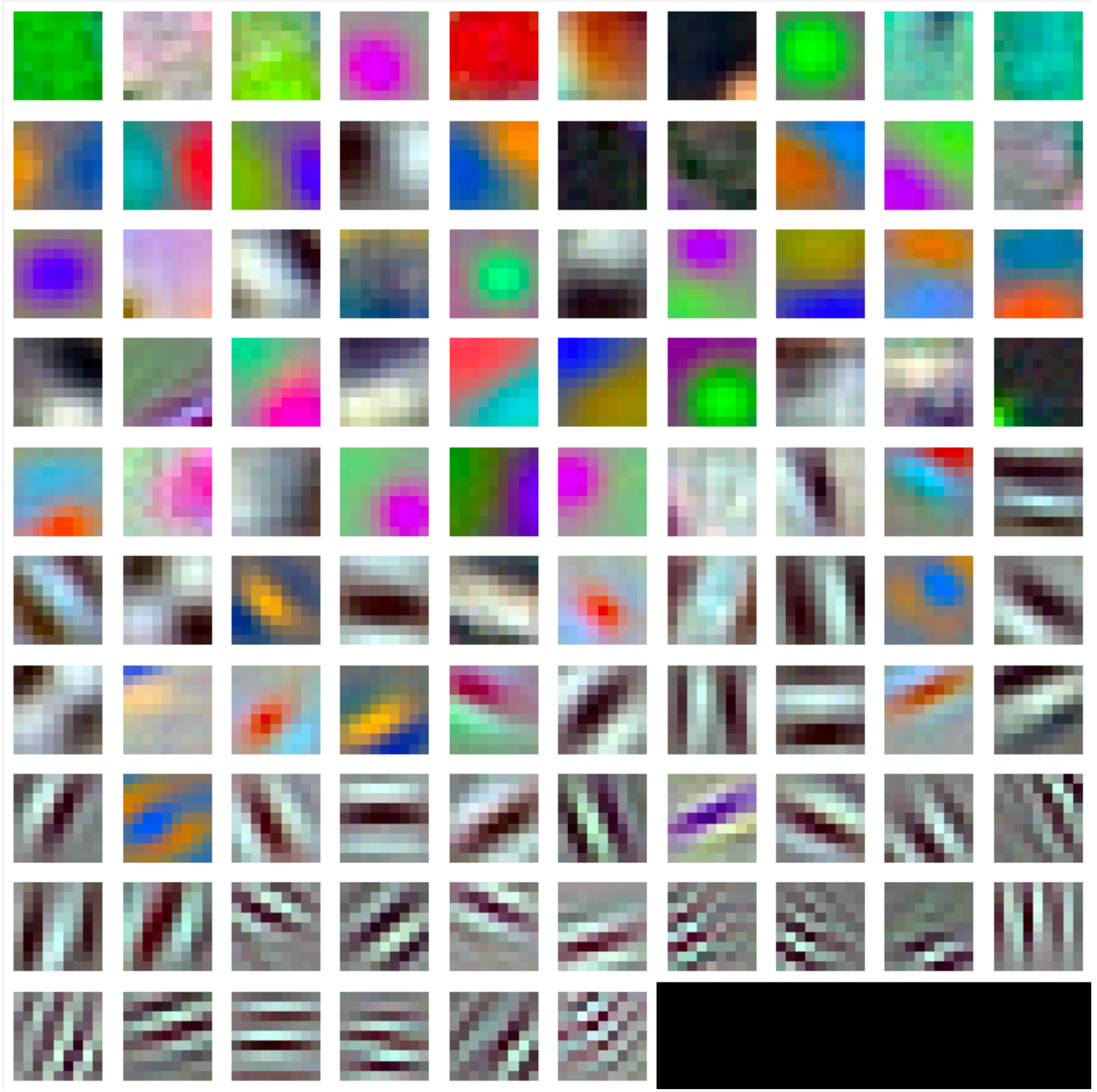


Get to know your units

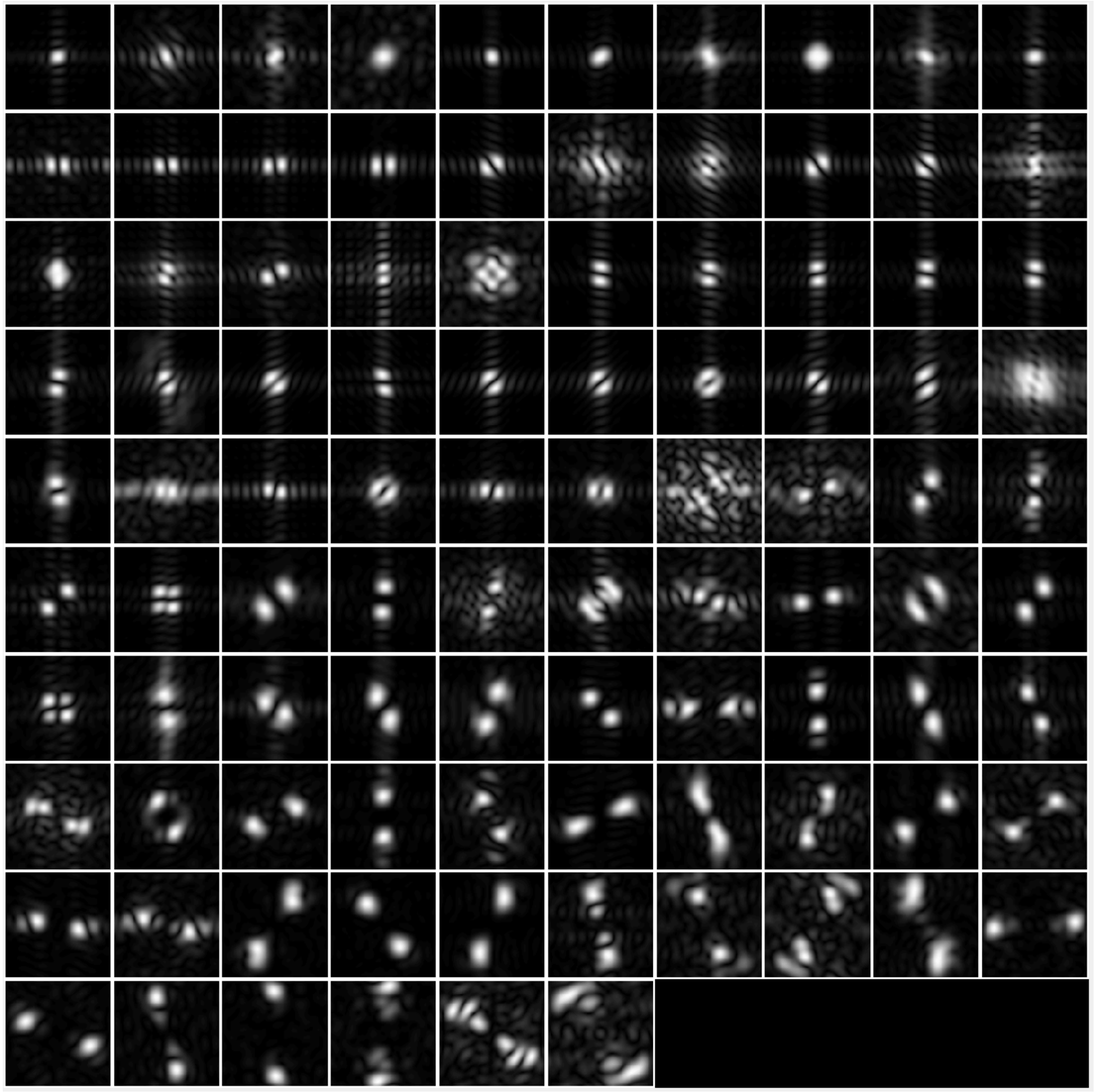




Get to know your units

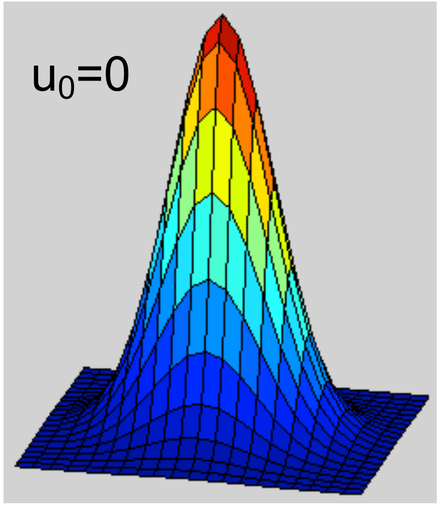


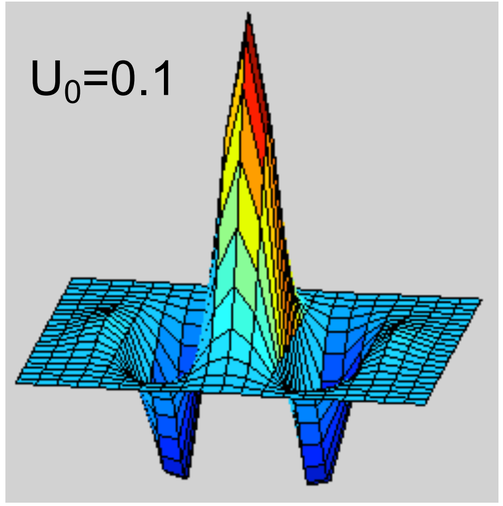
96 Units in conv1

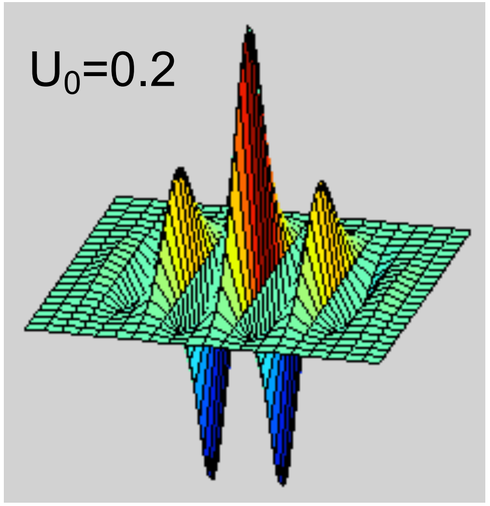


Gabor wavelets

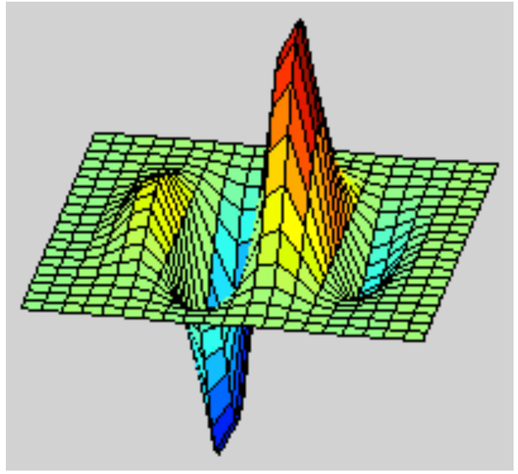
$$\psi_c(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \cos(2\pi u_0 x)$$

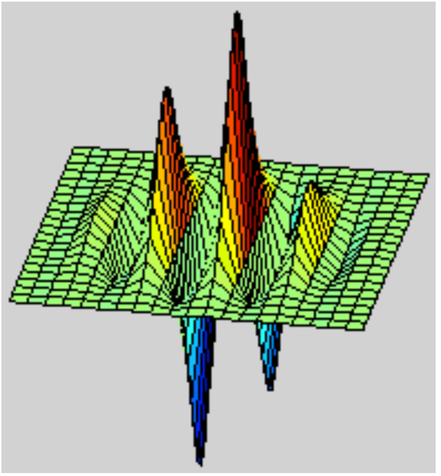


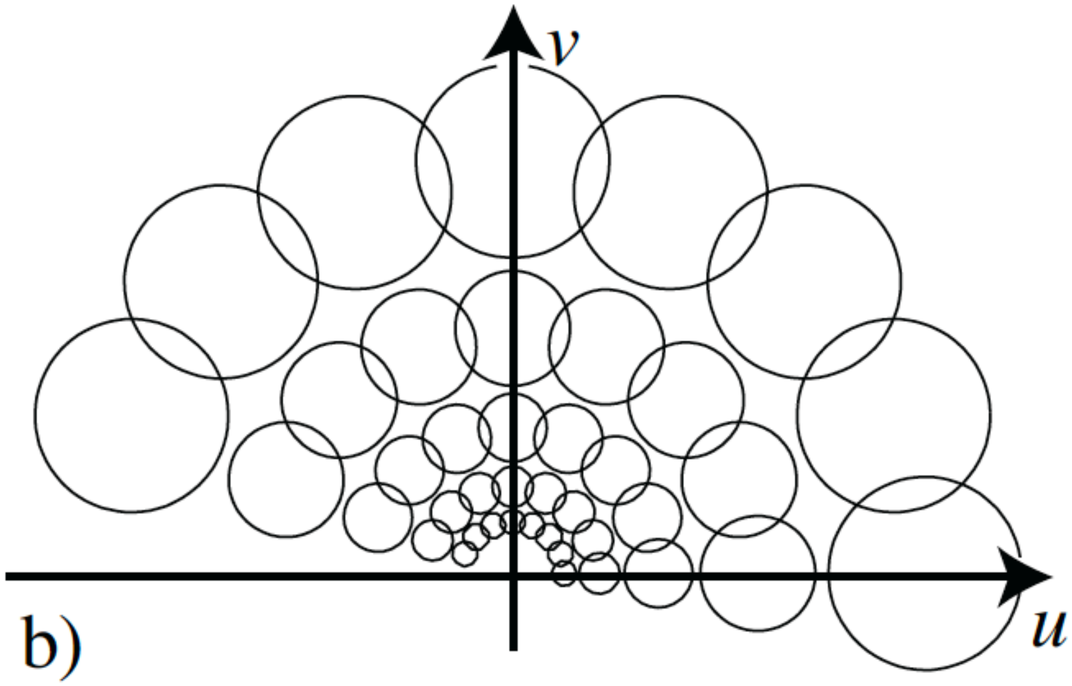




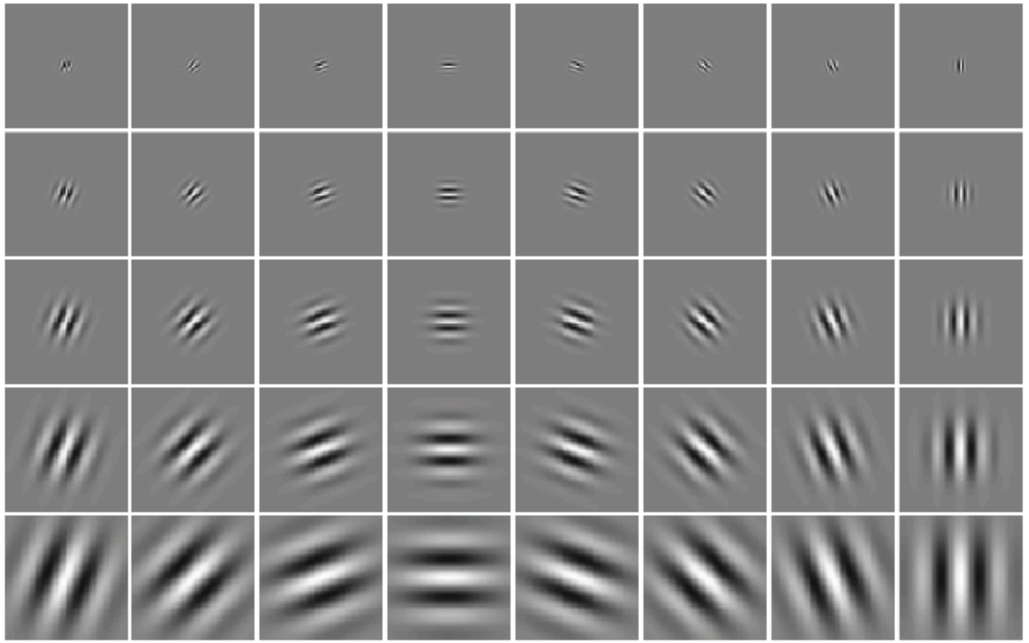
$$\psi_s(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \sin(2\pi u_0 x)$$







b)



Comparing Human and Machine Perception

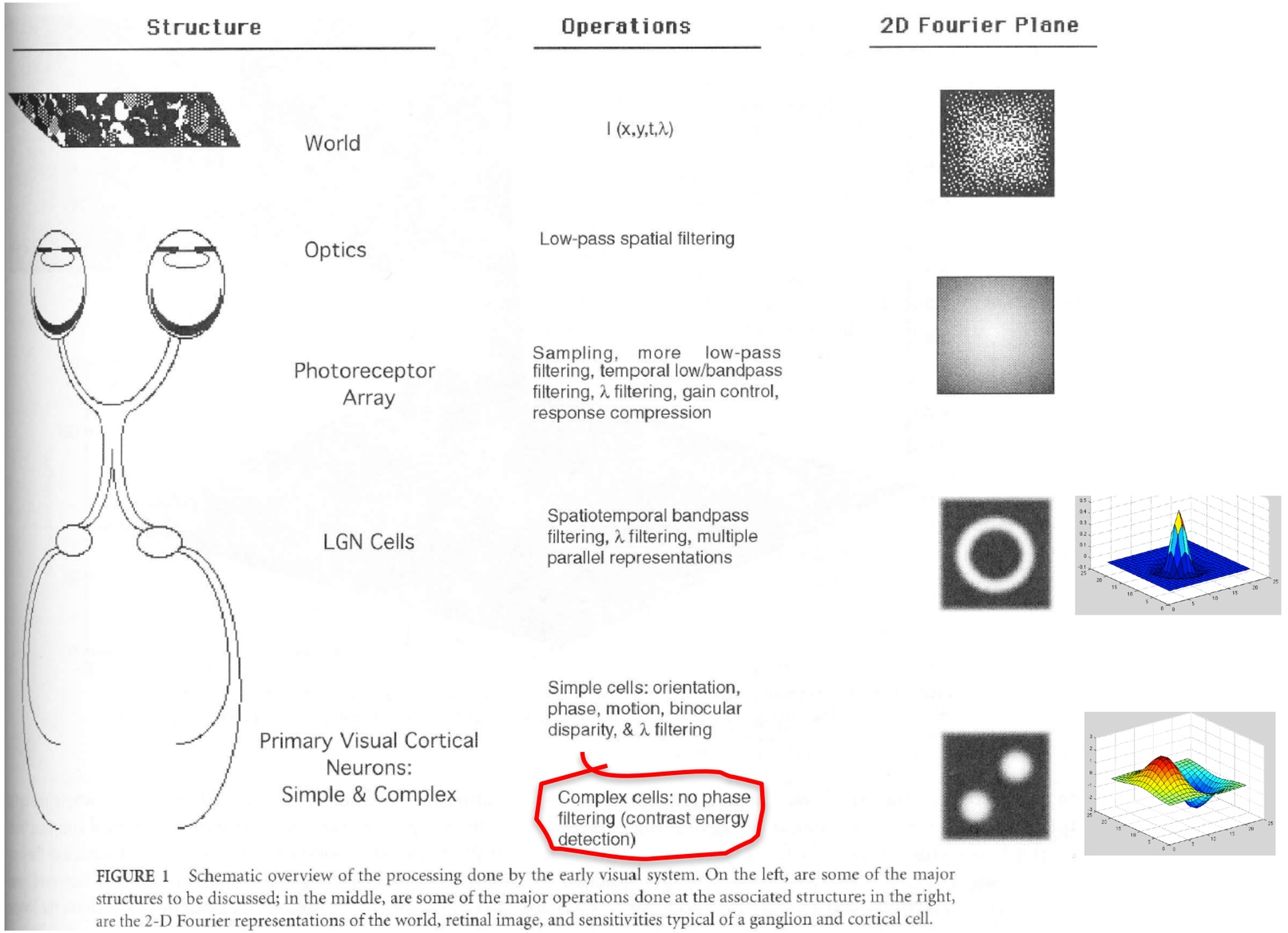


FIGURE 1 Schematic overview of the processing done by the early visual system. On the left, are some of the major structures to be discussed; in the middle, are some of the major operations done at the associated structure; in the right, are the 2-D Fourier representations of the world, retinal image, and sensitivities typical of a ganglion and cortical cell.

John Daugman, 1988

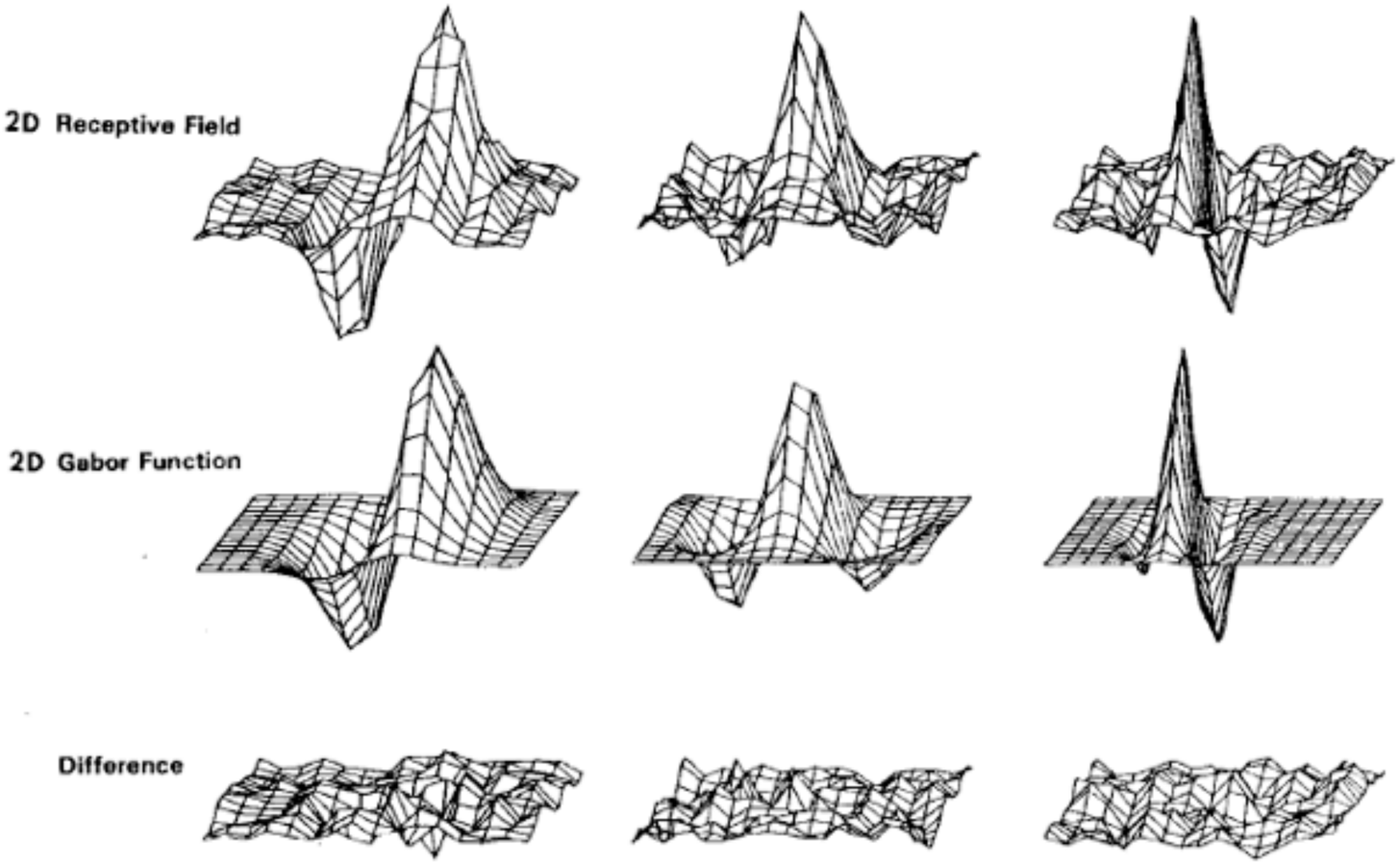
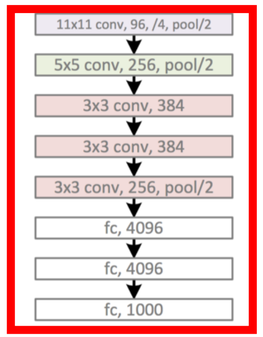


Fig. 5. Top row: illustrations of empirical 2-D receptive field profiles measured by J. P. Jones and L. A. Palmer (personal communication) in simple cells of the cat visual cortex. Middle row: best-fitting 2-D Gabor elementary function for each neuron, described by (10). Bottom row: residual error of the fit, indistinguishable from random error in the Chi-squared sense for 97 percent of the cells studied.

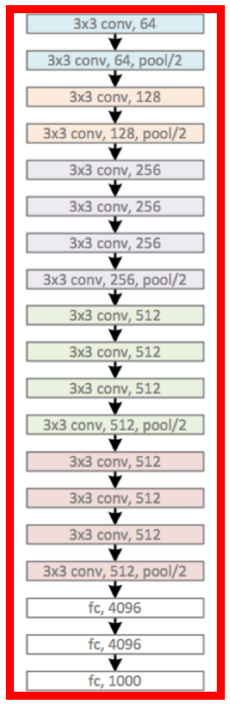
Deep Neural Networks for Visual Recognition

2012: AlexNet
5 conv. layers



Error: 15.3%

2014: VGG
16 conv. layers



Error: 8.5%

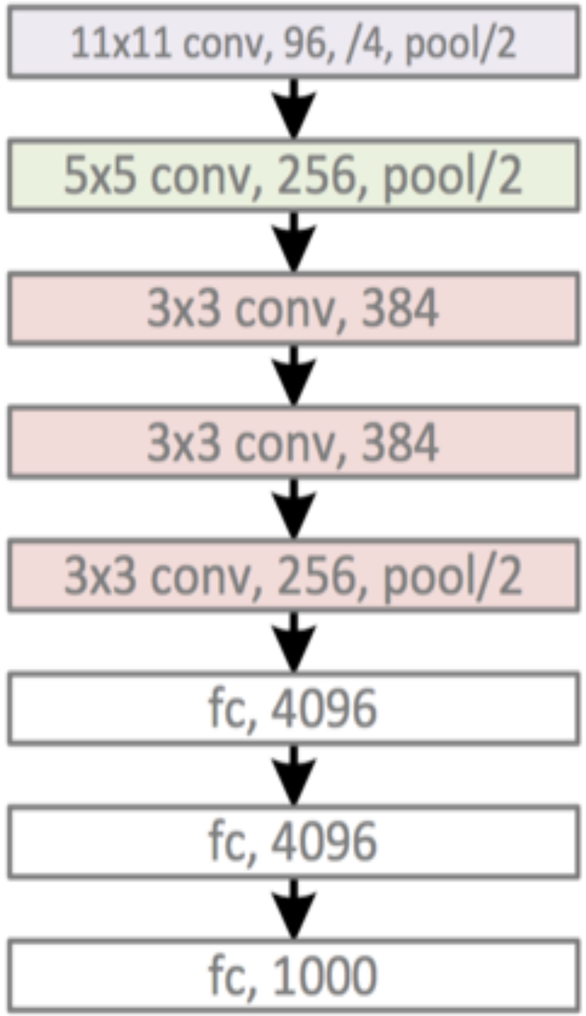
2015: GoogLeNet
22 conv. layers

Error: 7.8%

2016: ResNet
>100 conv. layers

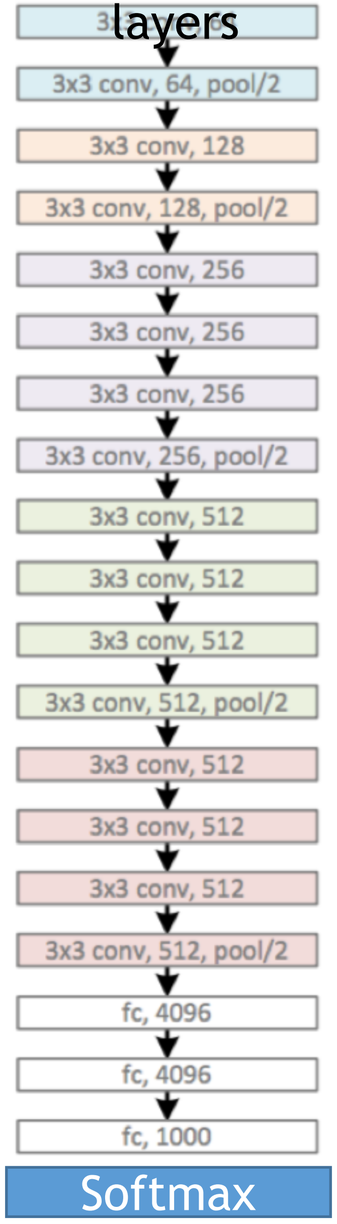
Error: 4.4%

2012: AlexNet
5 conv. layers



Error: 15.3%

2014: VGG
16 conv.
layers

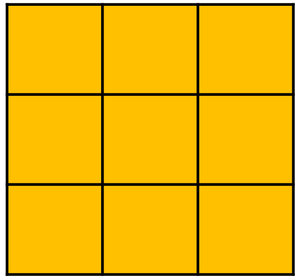


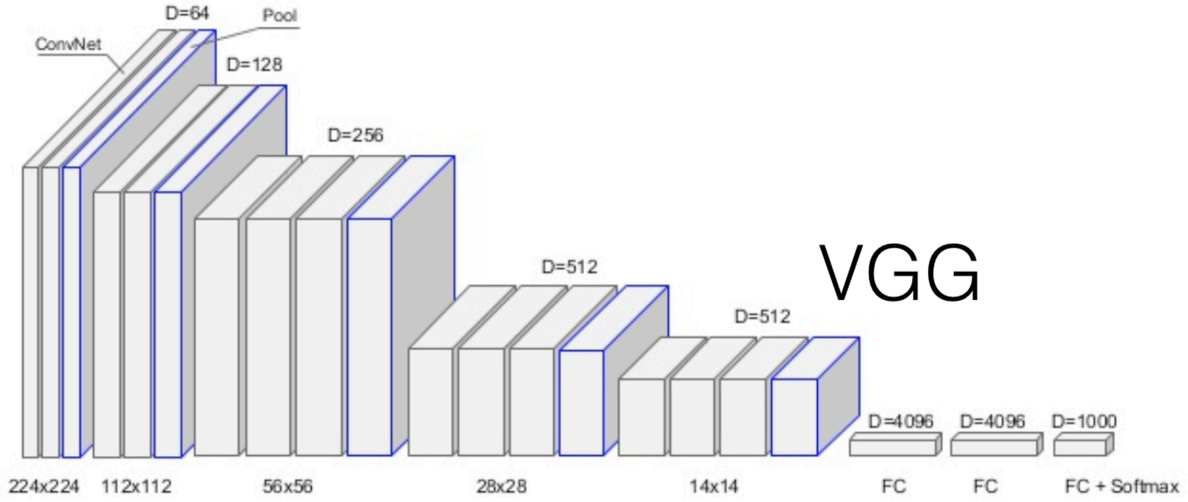
Error: 8.5%

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

https://arxiv.org/pdf/1409.1556.pdf

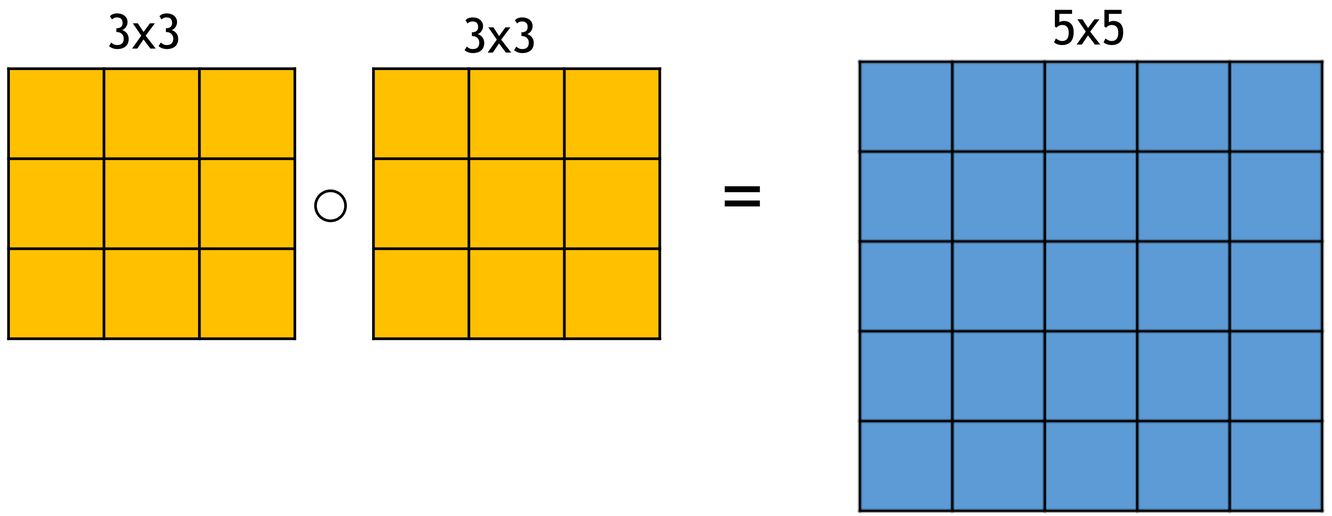
Small convolutional kernels: 3x3
ReLu non-linearities
>100 million parameters.



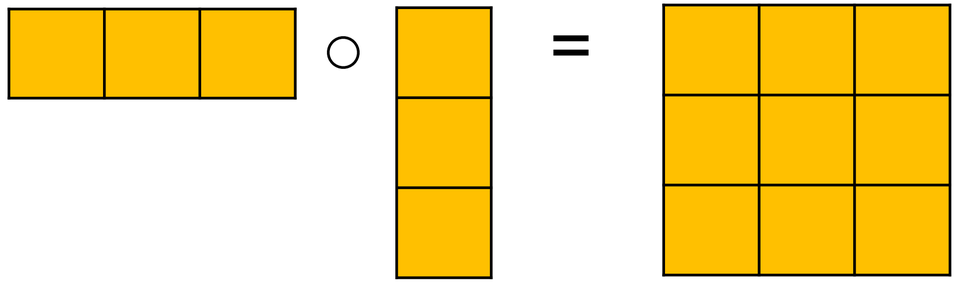


VGG

Chaining convolutions

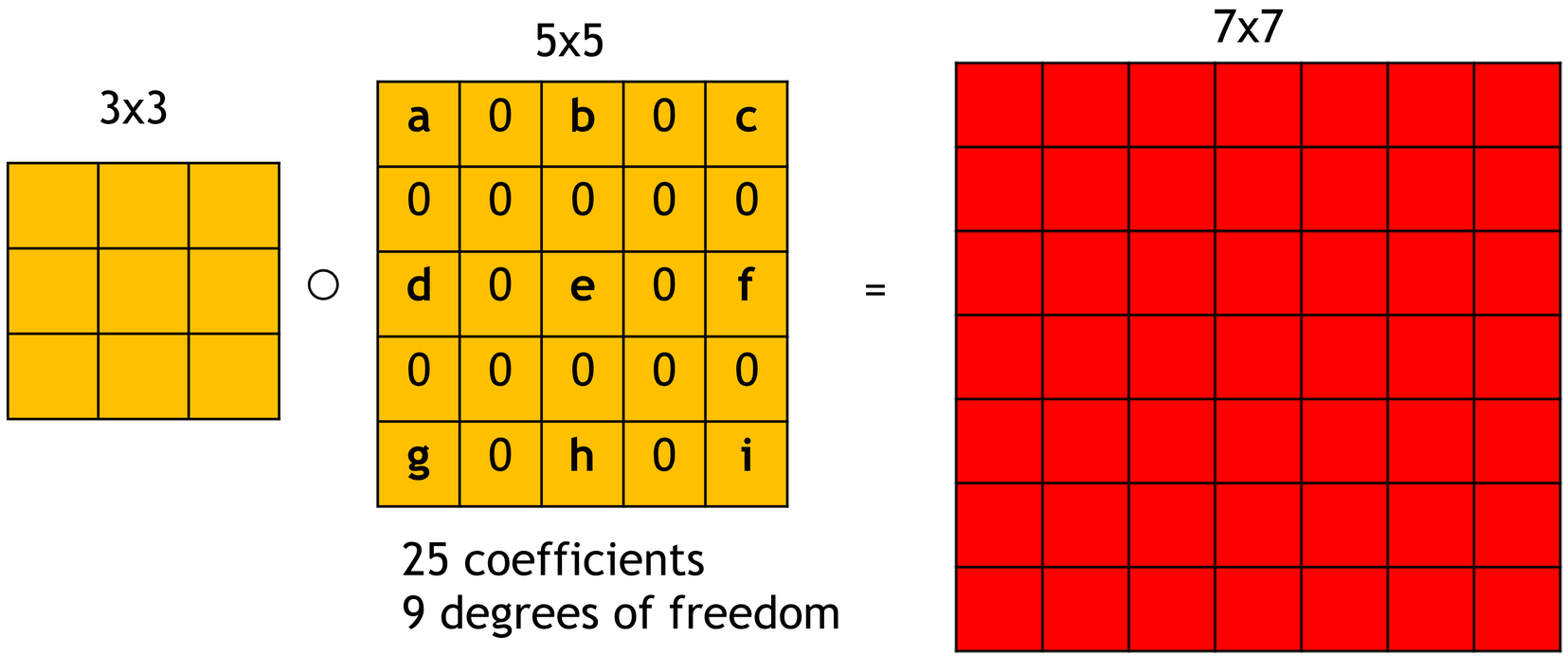


25 coefficients, but only
18 degrees of freedom



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

Dilated convolutions



What is lost?

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

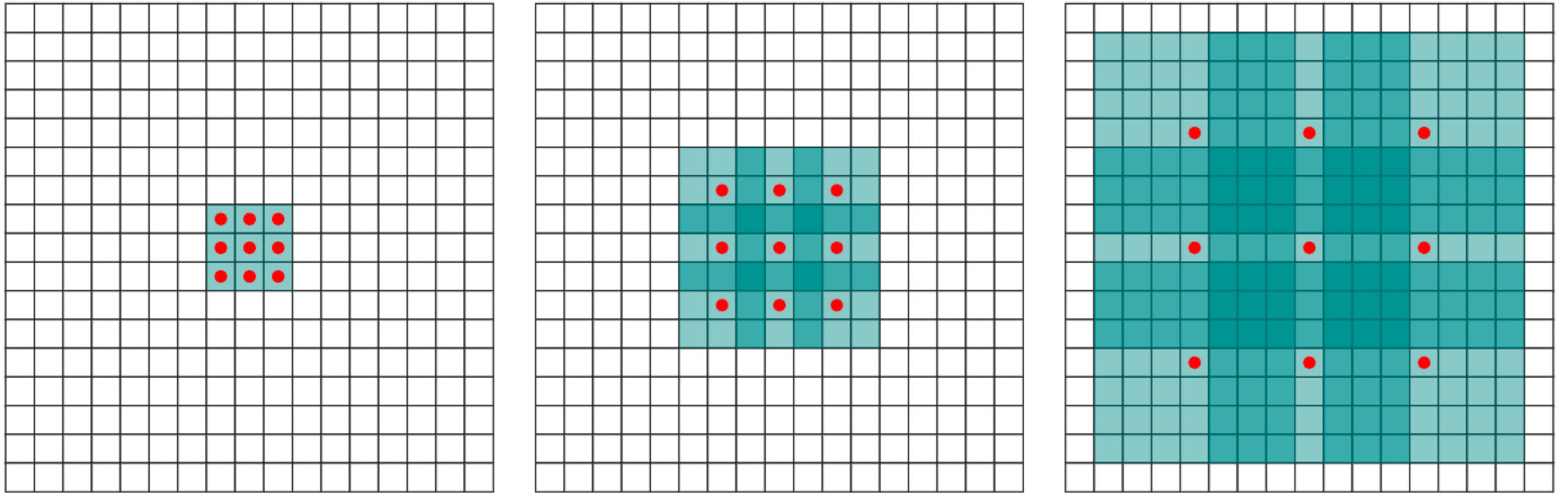
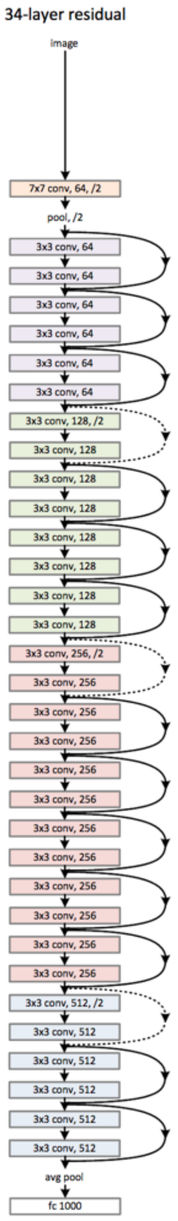


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element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

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

2016: ResNet
>100 conv. layers

Error: 4.4%



Deep Residual Learning for Image Recognition

https://arxiv.org/pdf/1512.03385.pdf

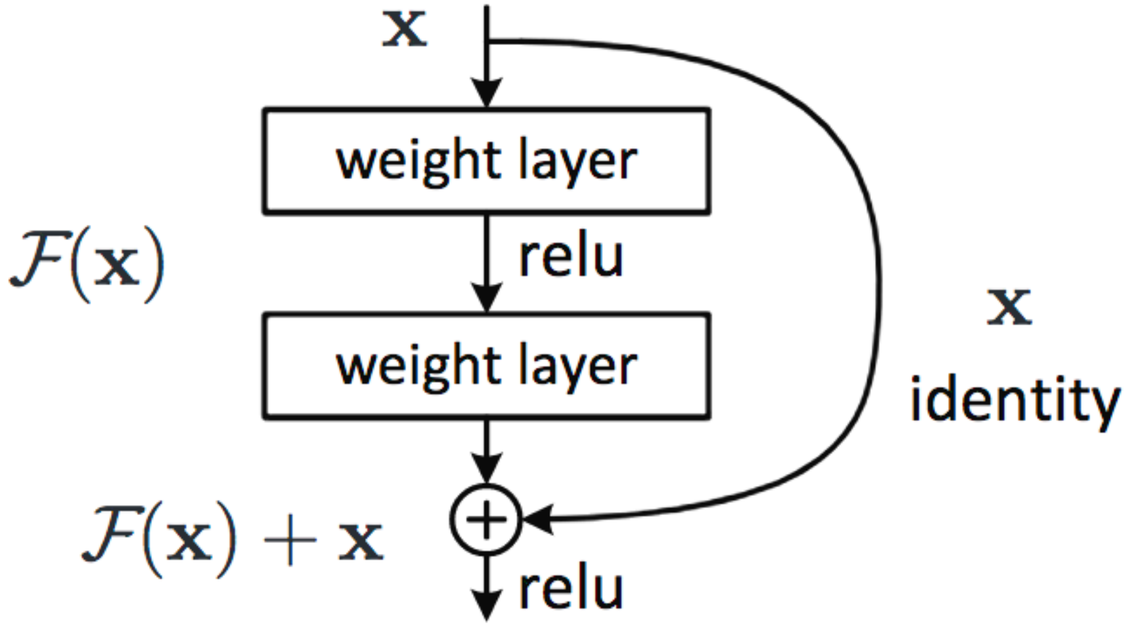
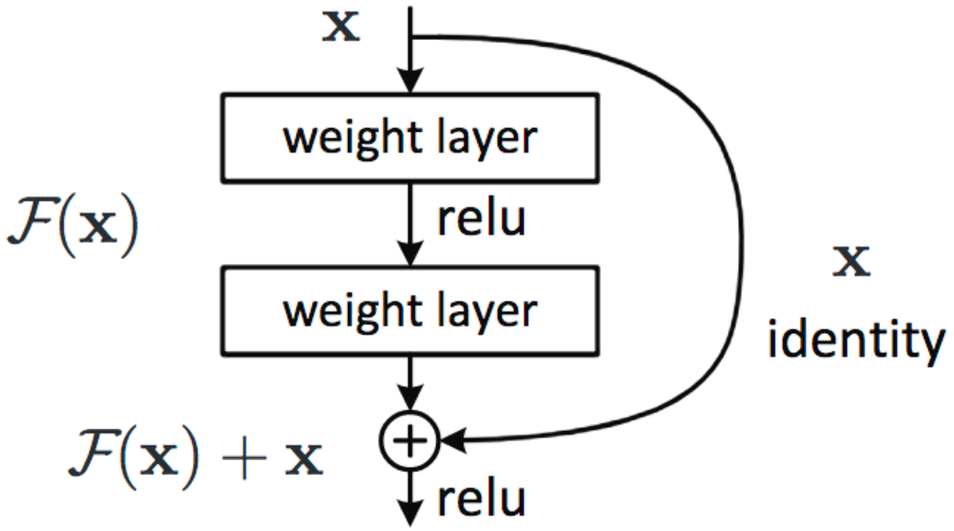
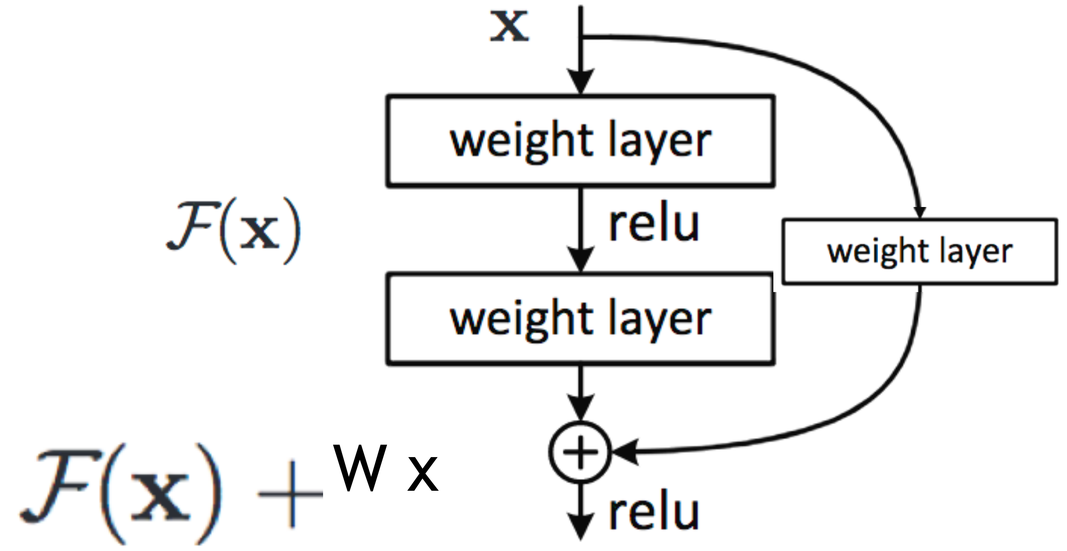


Figure 2. Residual learning: a building block.

If output has same size as input:

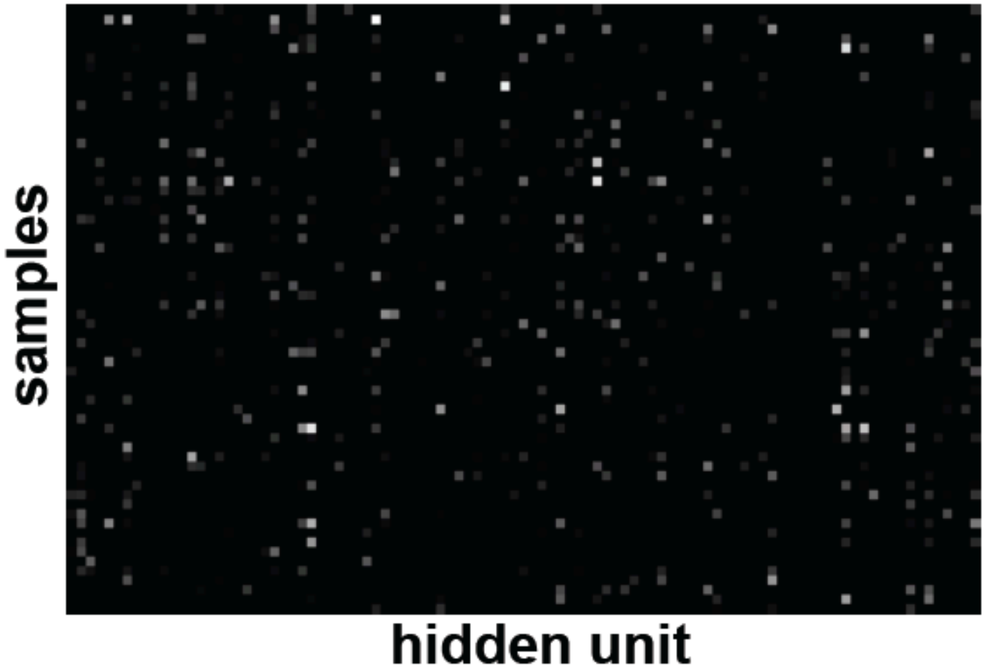


If output has a different size:



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

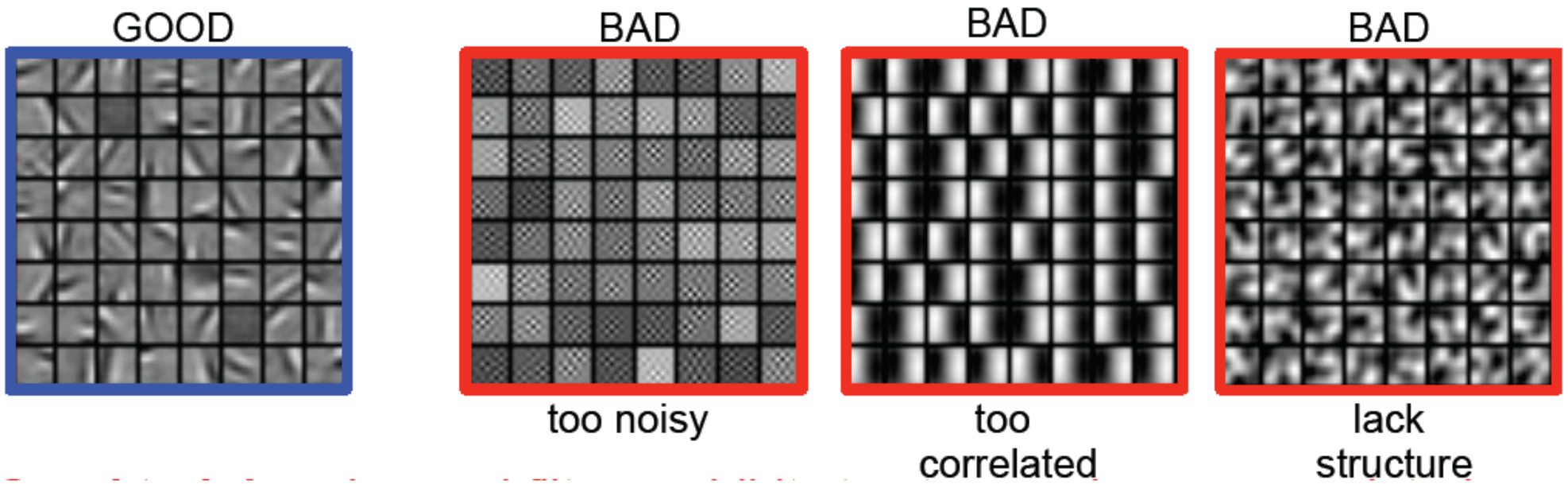
- Check gradients numerically by finite differences
- Visualize hidden activations — should be uncorrelated and high variance

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

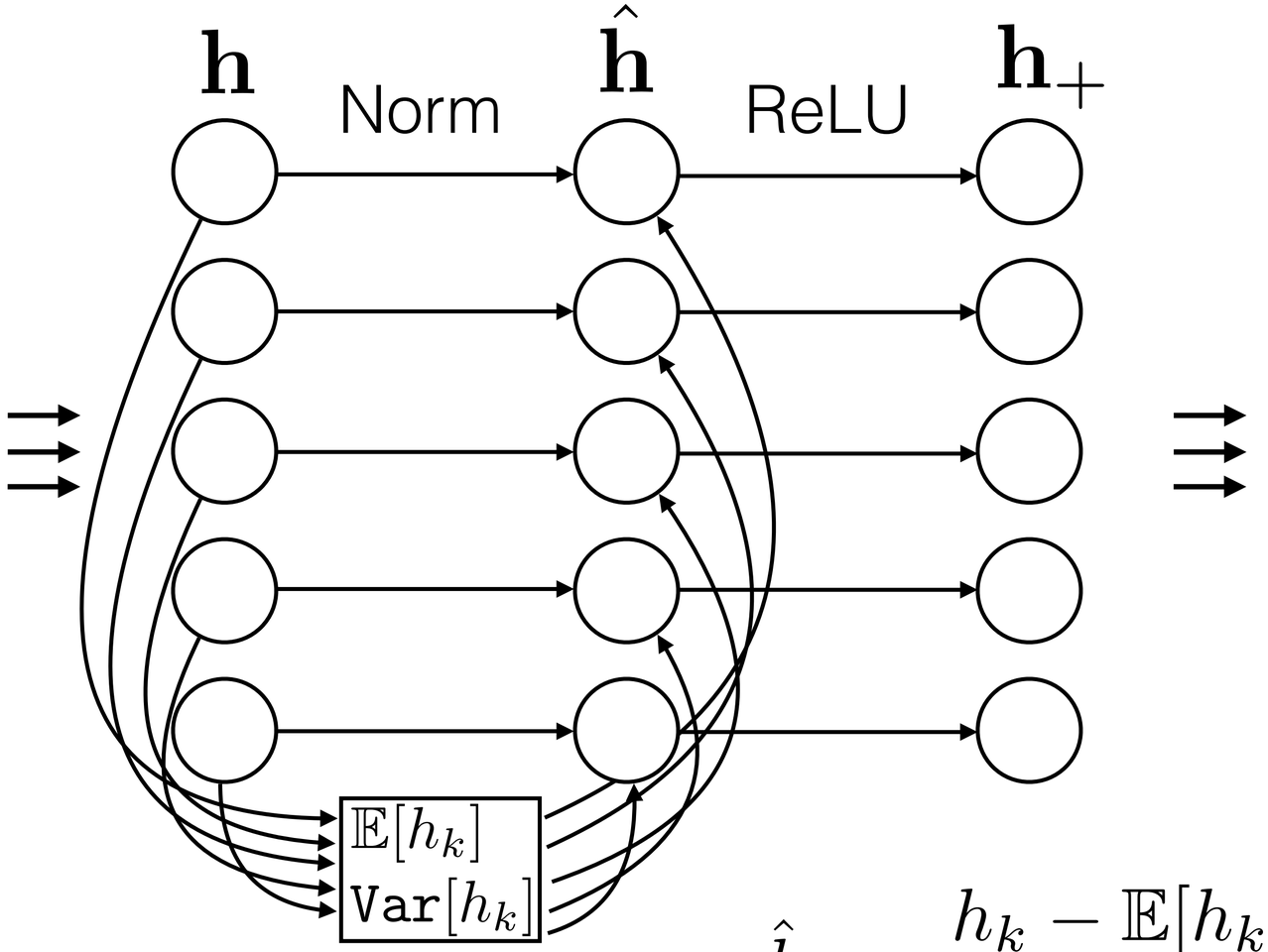
- Check gradients numerically by finite differences
- Visualize hidden activations — should be uncorrelated and high variance
- Visualize filters



Good training: learned filters exhibit structure and are uncorrelated.

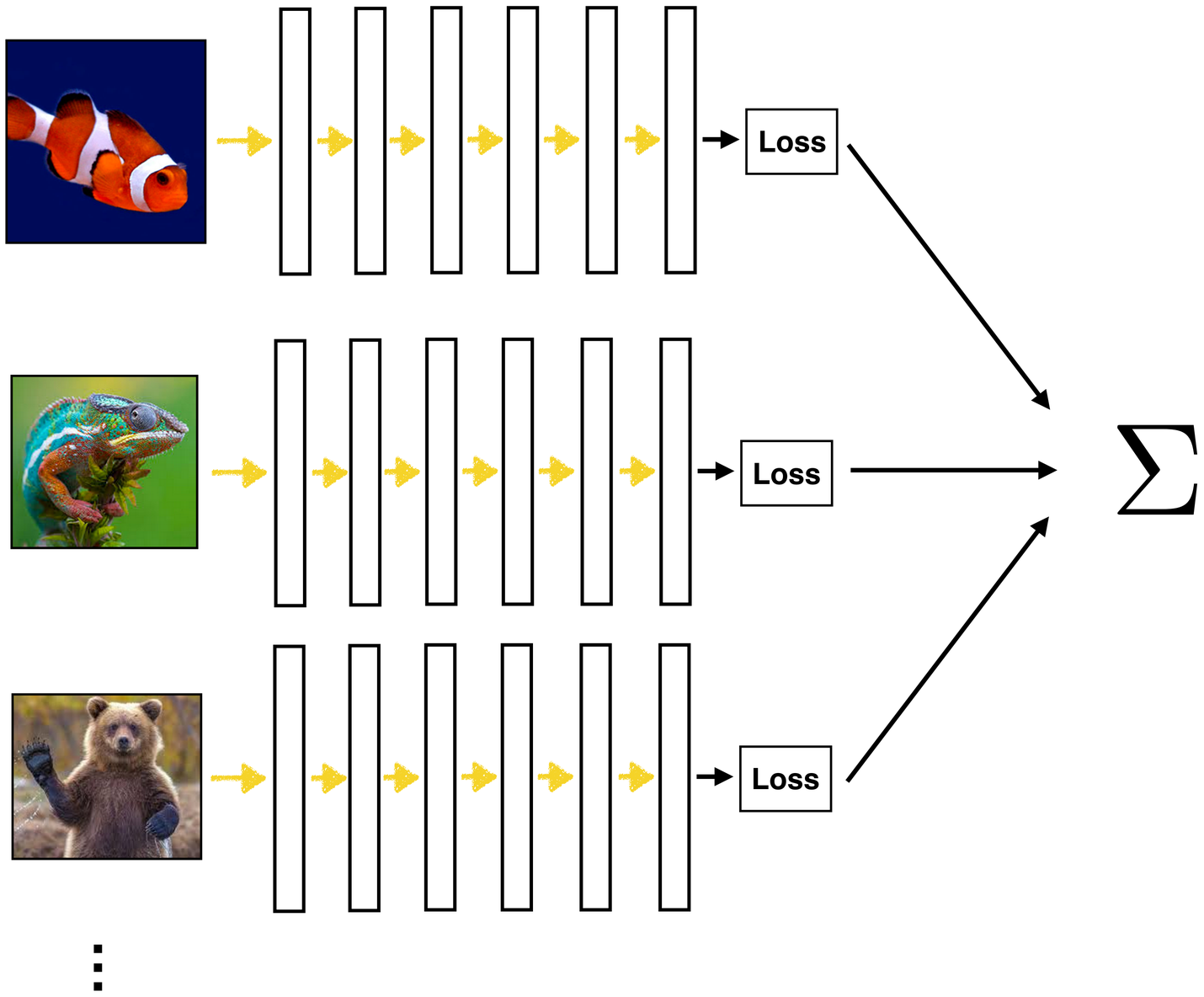
[Derived from slide by Marc'Aurelio Ranzato]

Normalization layers



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

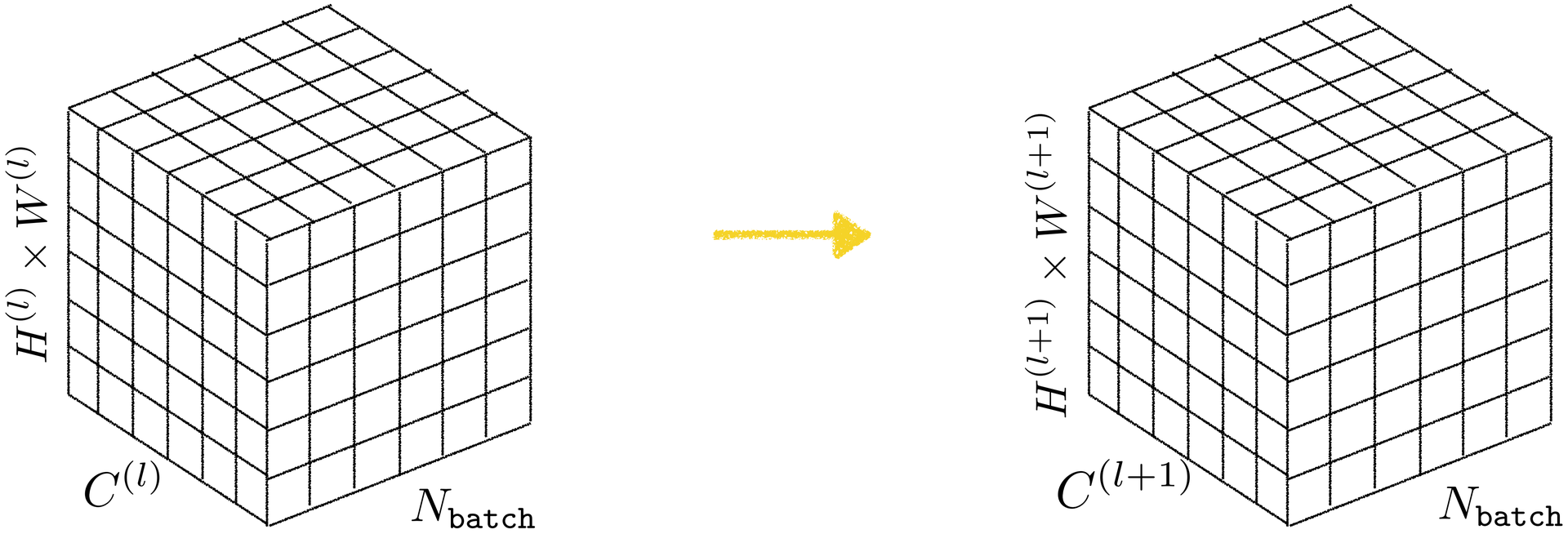
Batch processing



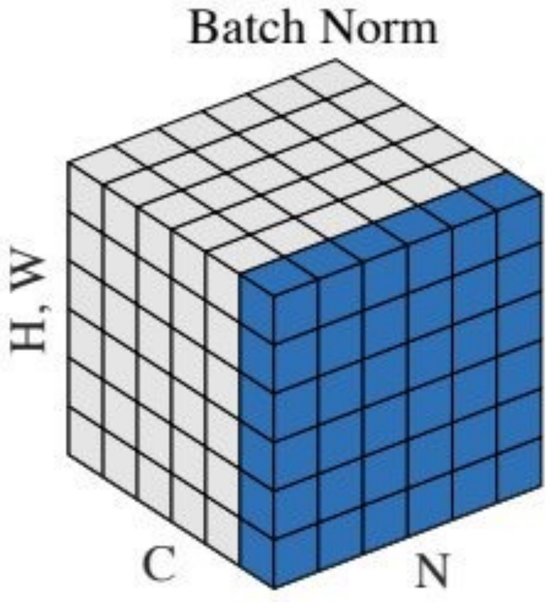
“Tensor flow”

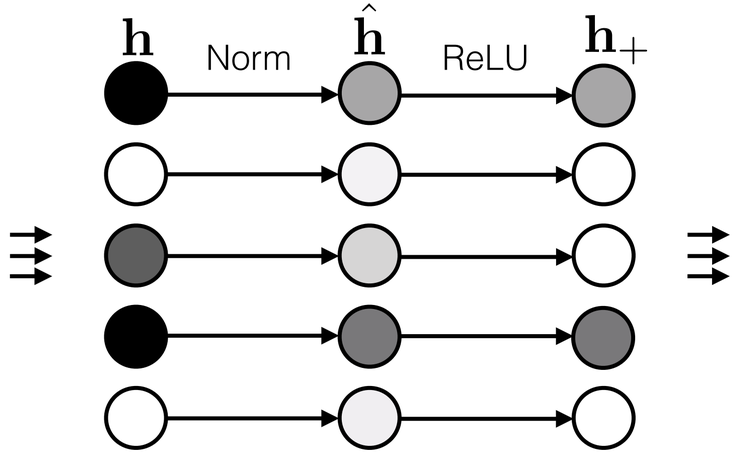
$$\mathbf{x}^{(l)} \in \mathbb{R}^{N_{\texttt{batch}} \times H^{(l)} \times W^{(l)} \times C^{(l)}}$$

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



Normalization layers

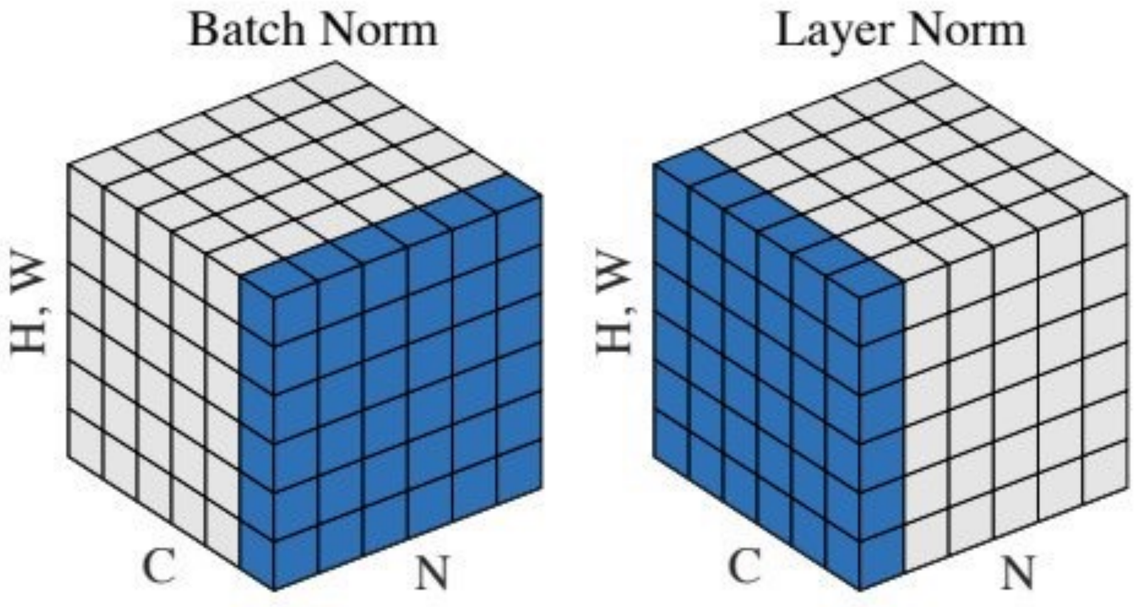




Normalize w.r.t. a single hidden unit's pattern of activation over training examples (a batch of examples).

[Figure from Wu & He, arXiv 2018]

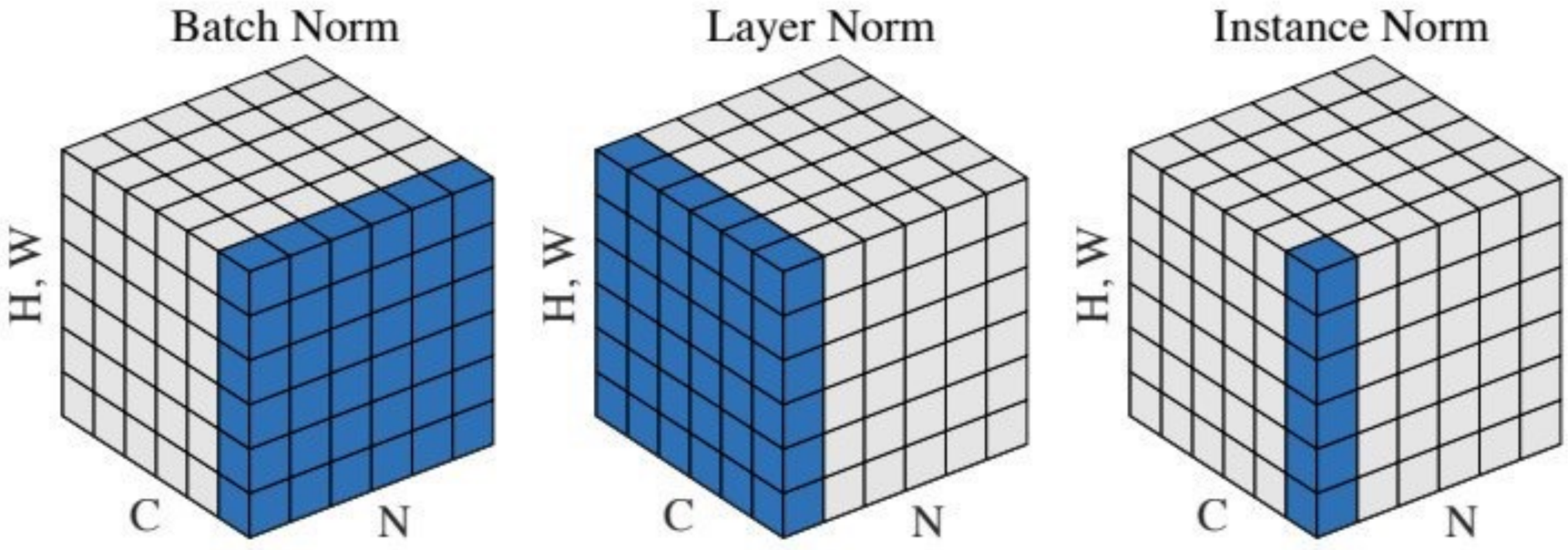
Normalization layers



Normalize w.r.t. the mean and variance of the activations of all the hidden units (neurons) on this layer (c).

[Figure from Wu & He, arXiv 2018]

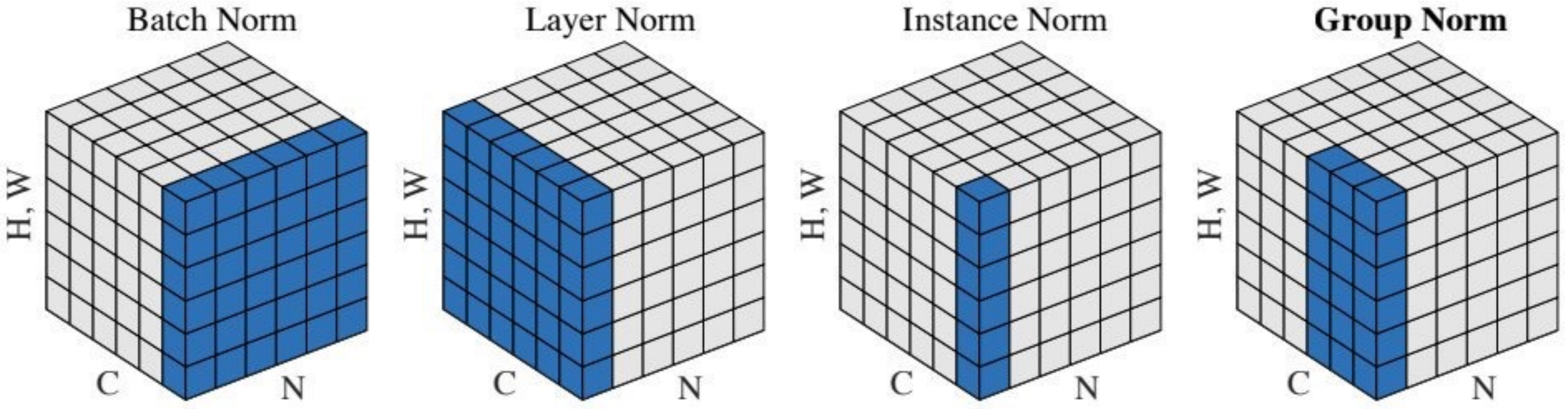
Normalization layers



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, arXiv 2018]

Normalization layers



Might as well…

[Figure from Wu & He, arXiv 2018]

Next up:

1. RNNs and sequential processing