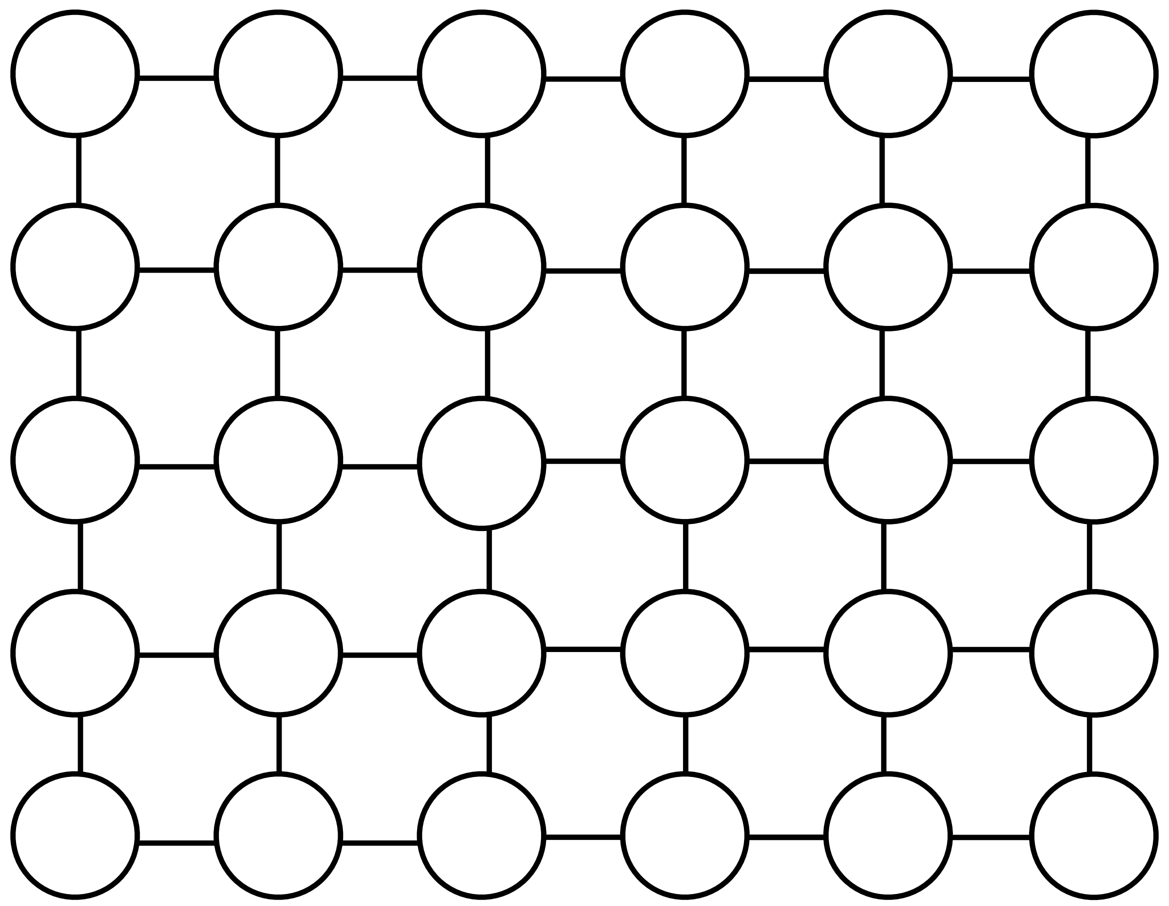
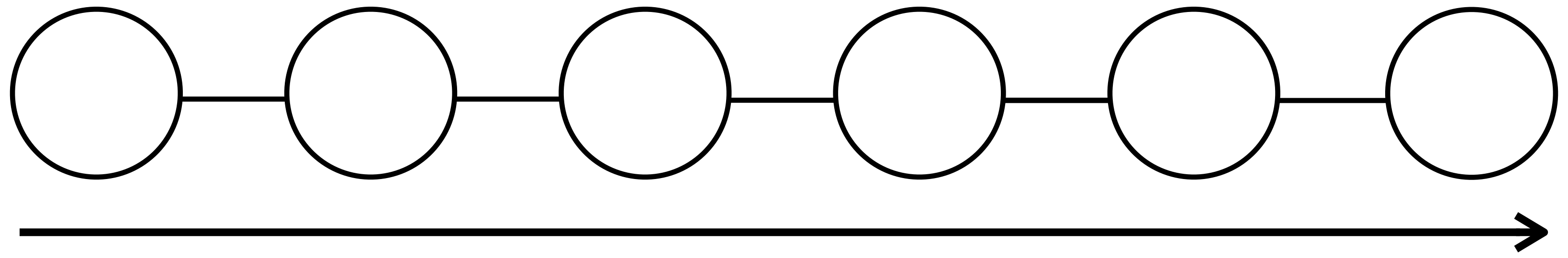
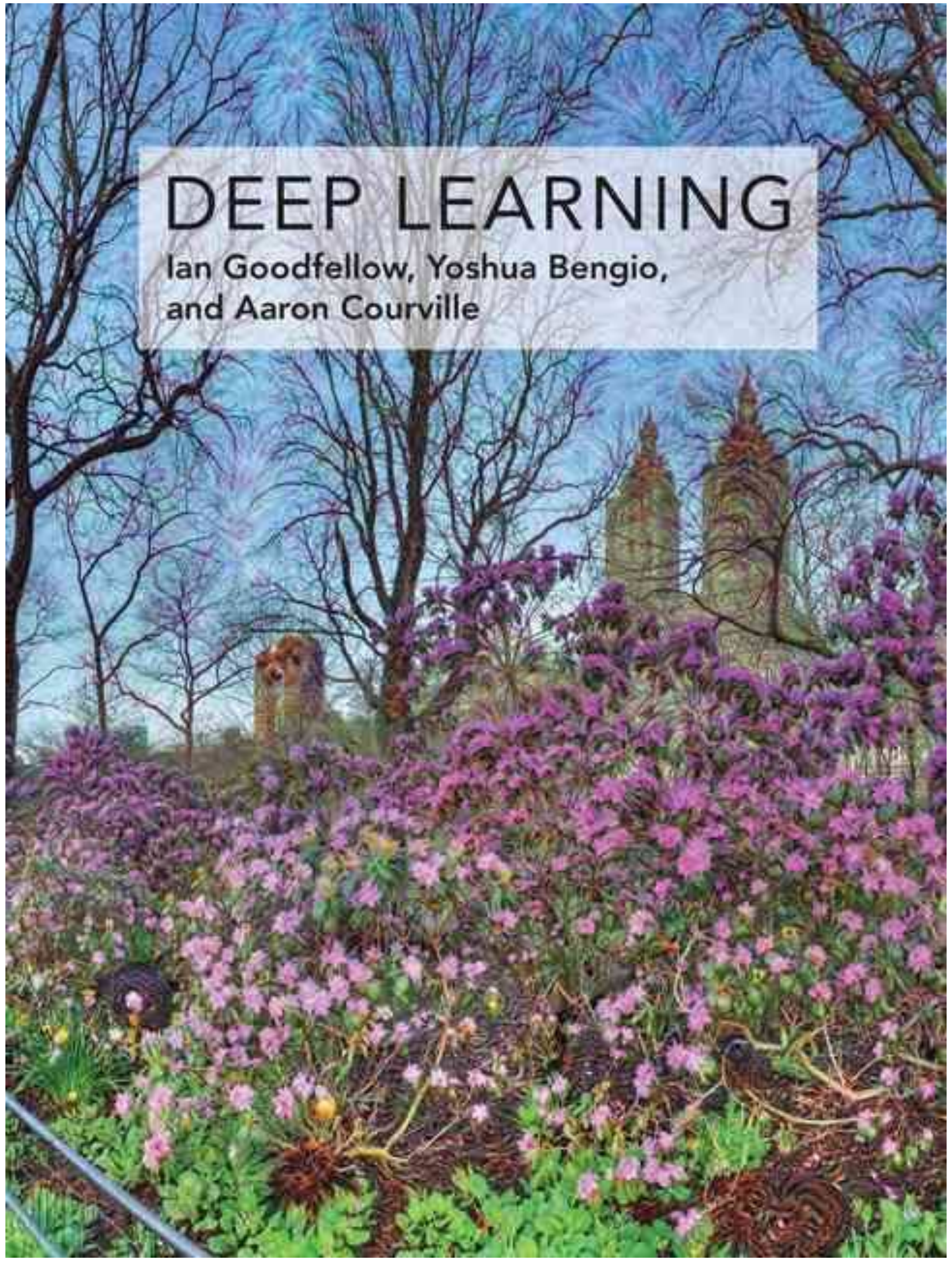


Modern CNNS, RNNs and Sequential Processing



Bill Freeman, Antonio Torralba, Phillip Isola

6.819 / 6.869



http://www.deeplearningbook.org/

By Ian Goodfellow, Yoshua Bengio and Aaron Courville

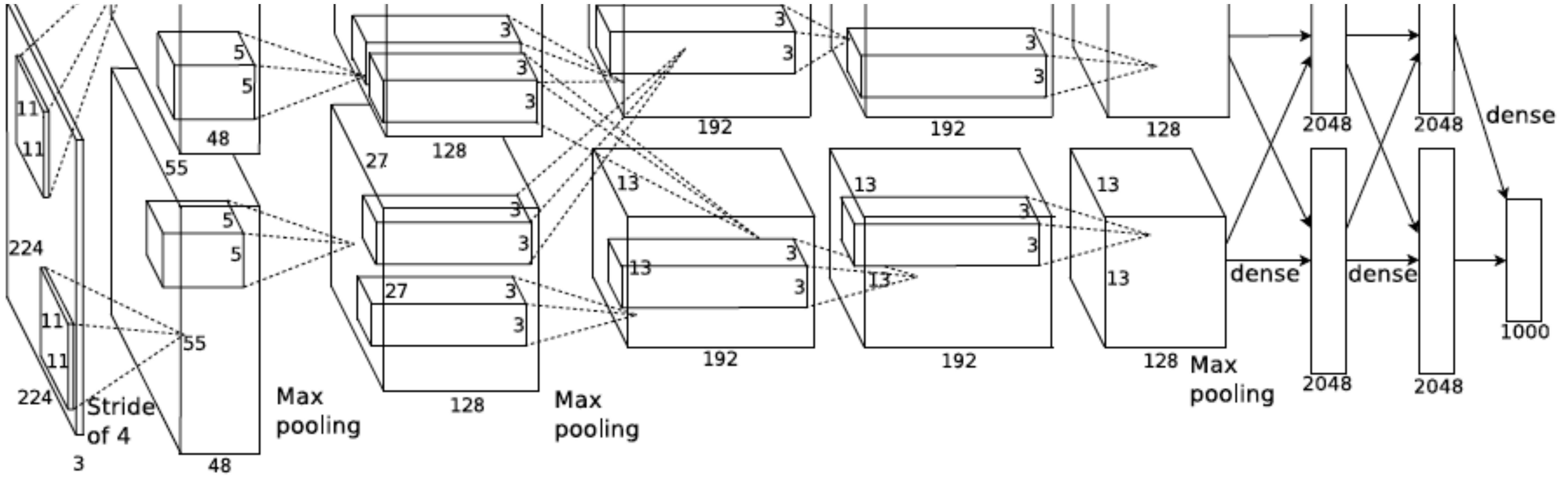
November 2016

Today: parts of chapter 10

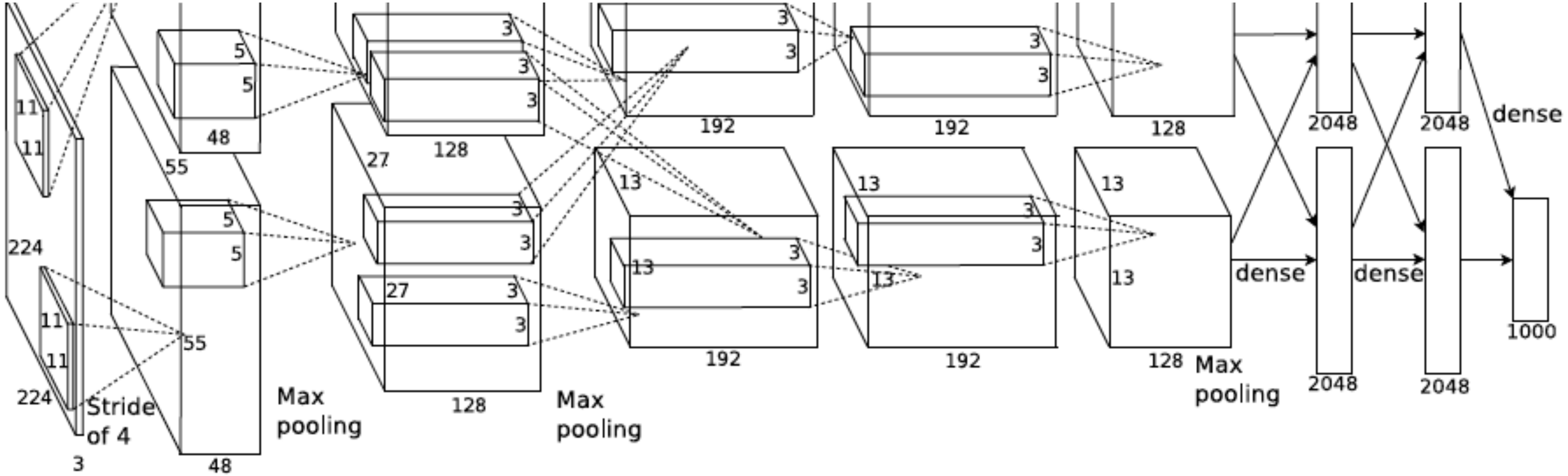
Review lecture 7

Other good resources for RNNs:
http://karpathy.github.io/2015/05/21/rnn-effectiveness/
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Alexnet — [Krizhevsky et al. NIPS 2012]



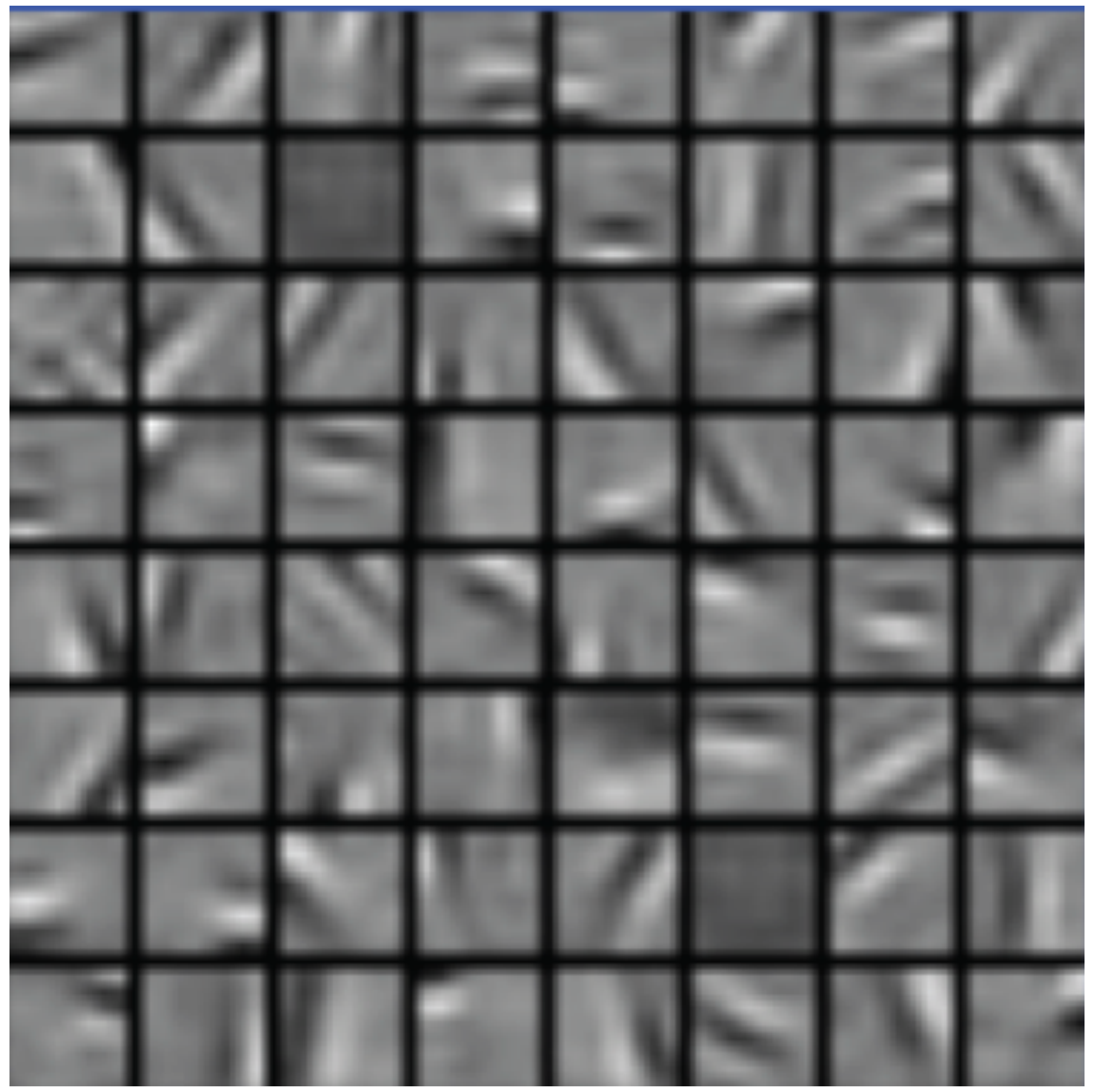
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384]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



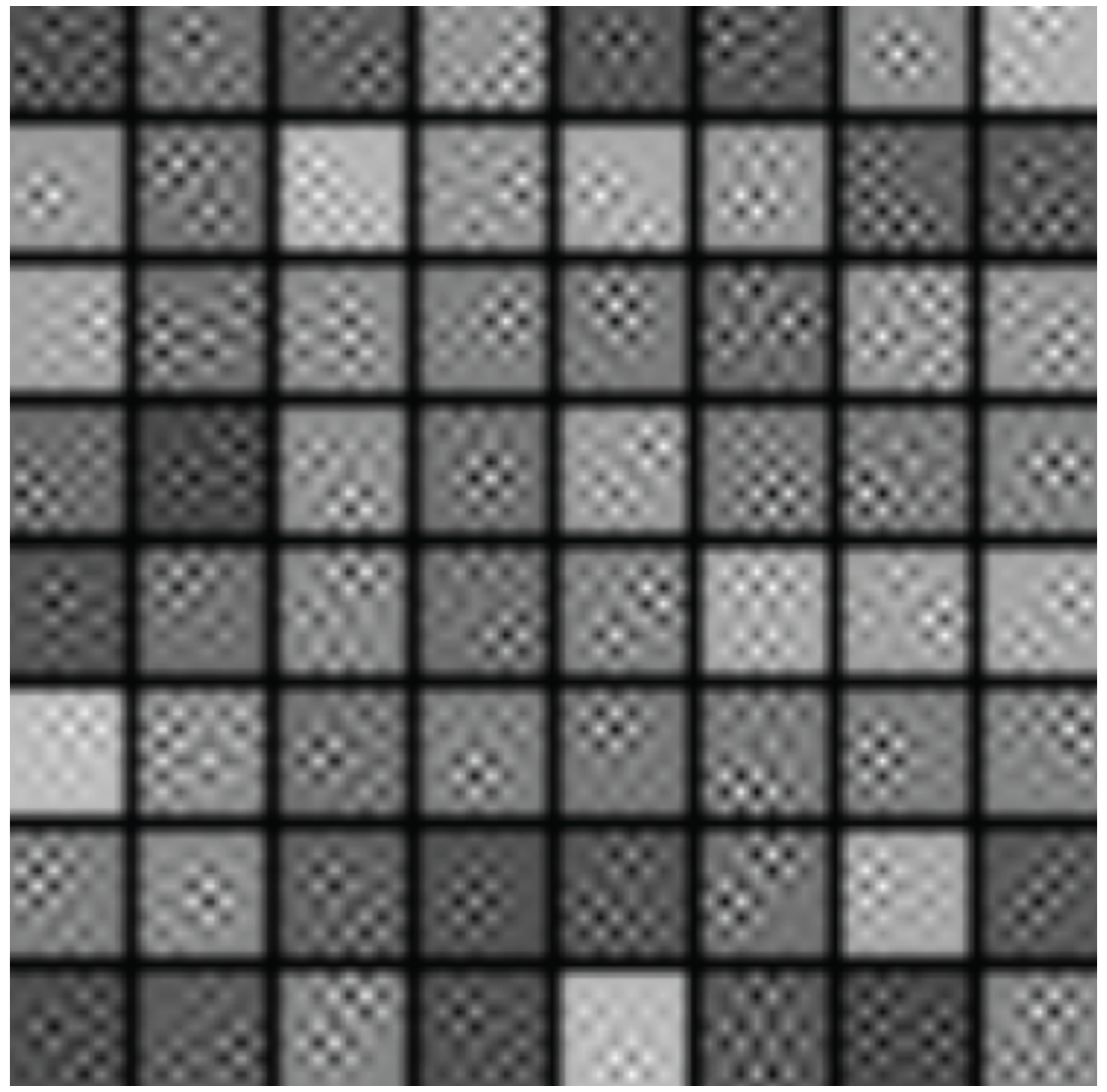
What filters are learned?

What filters are learned?

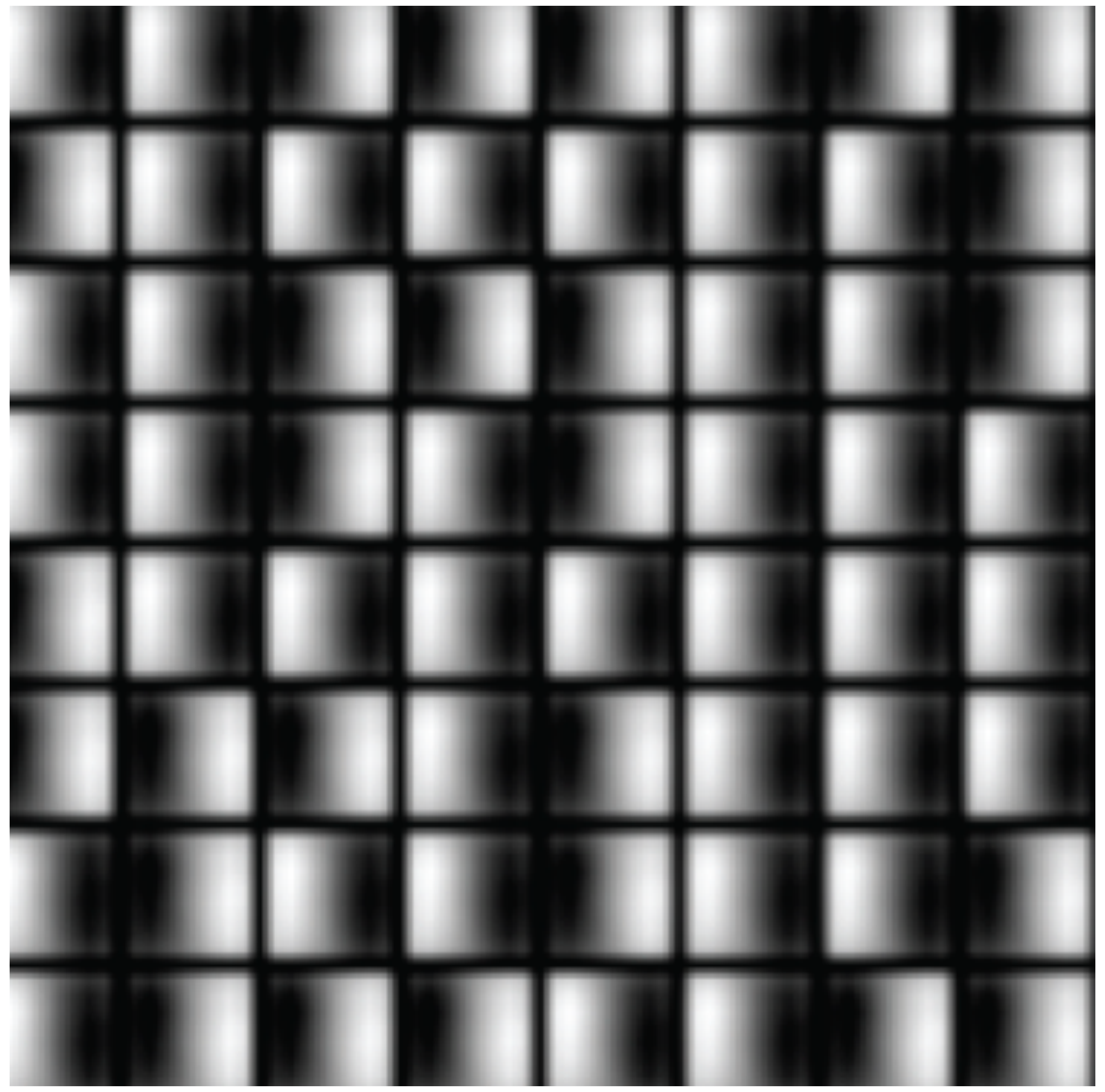
A



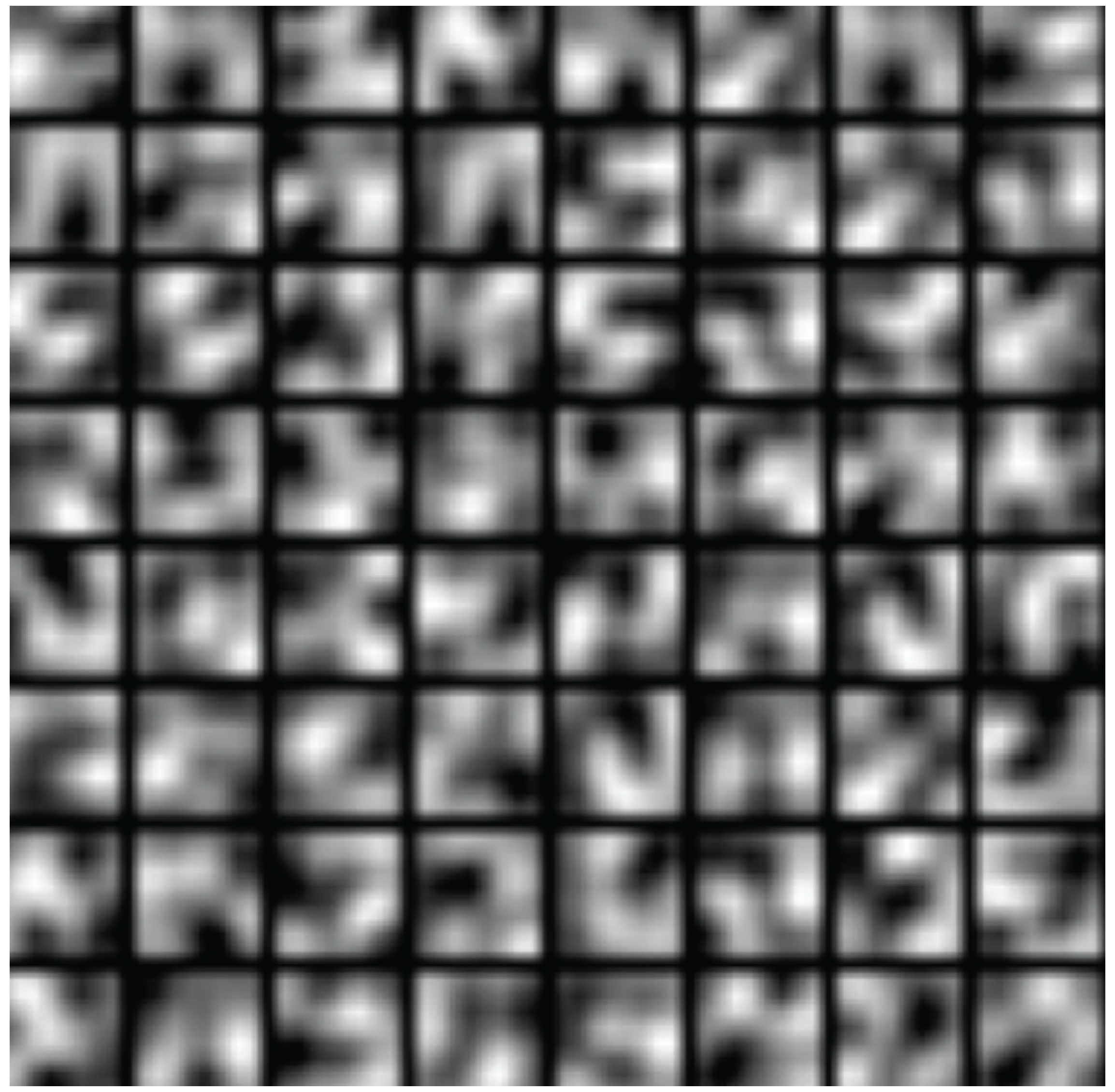
B



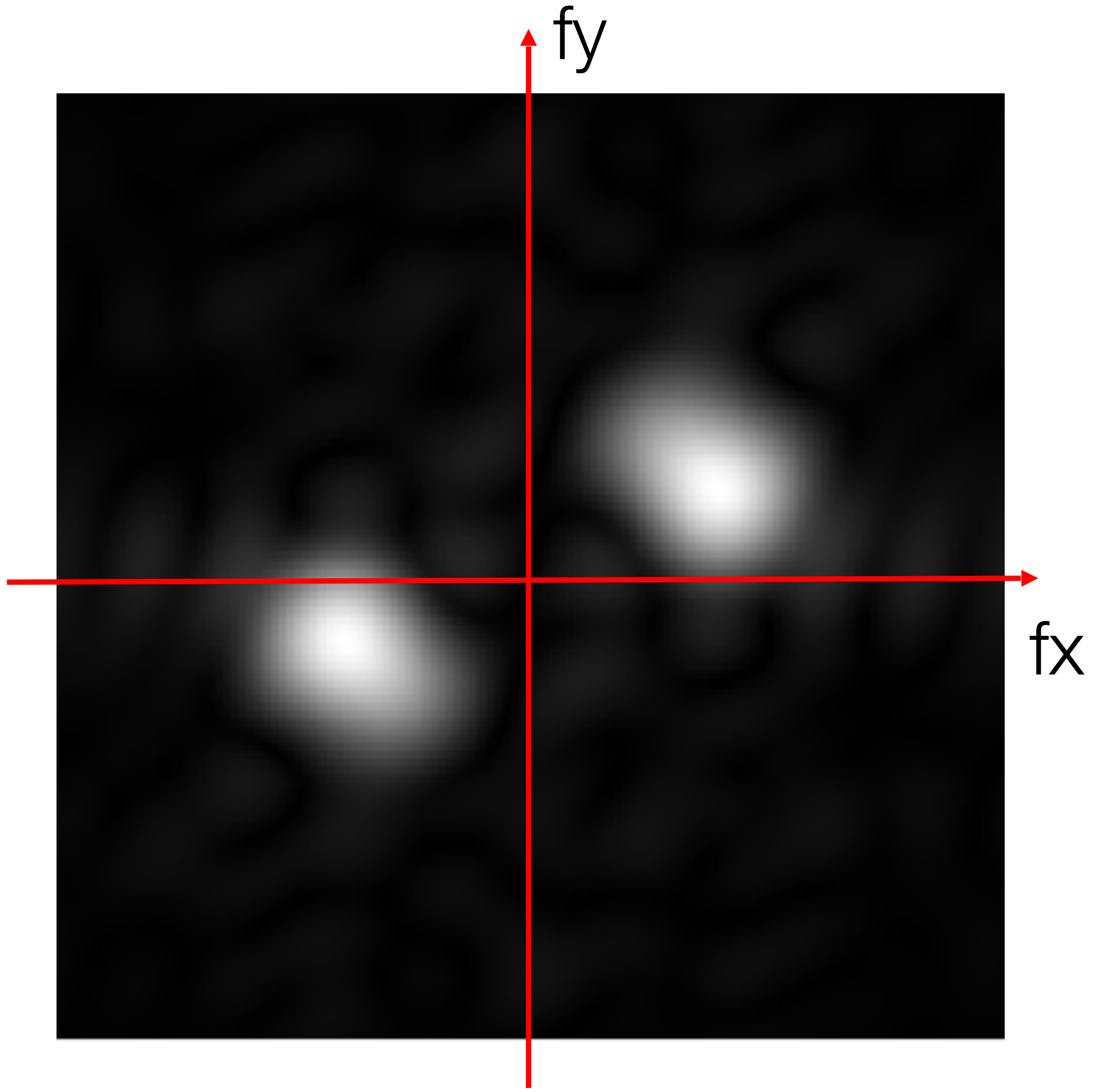
C



D

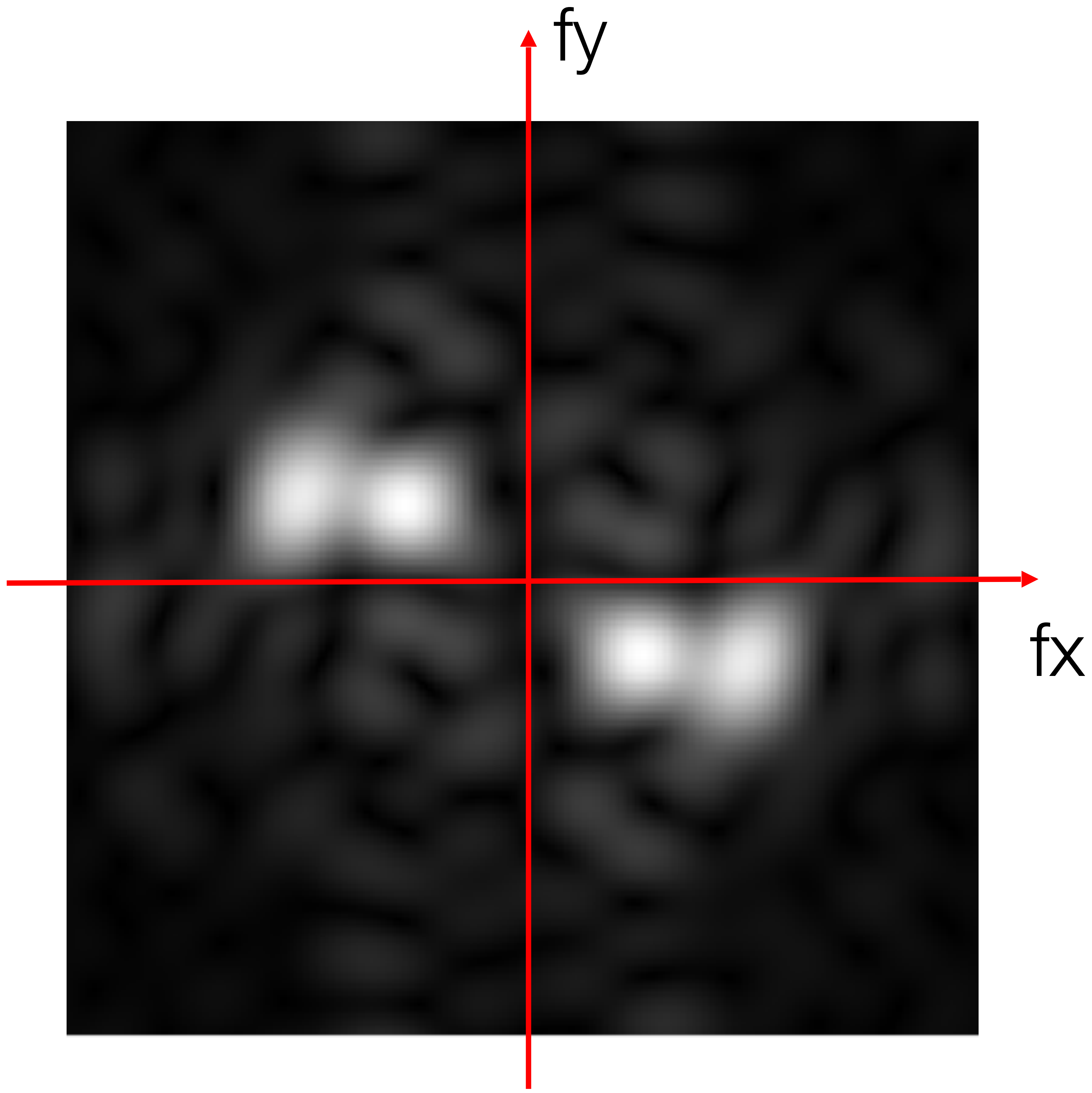


Get to know your units

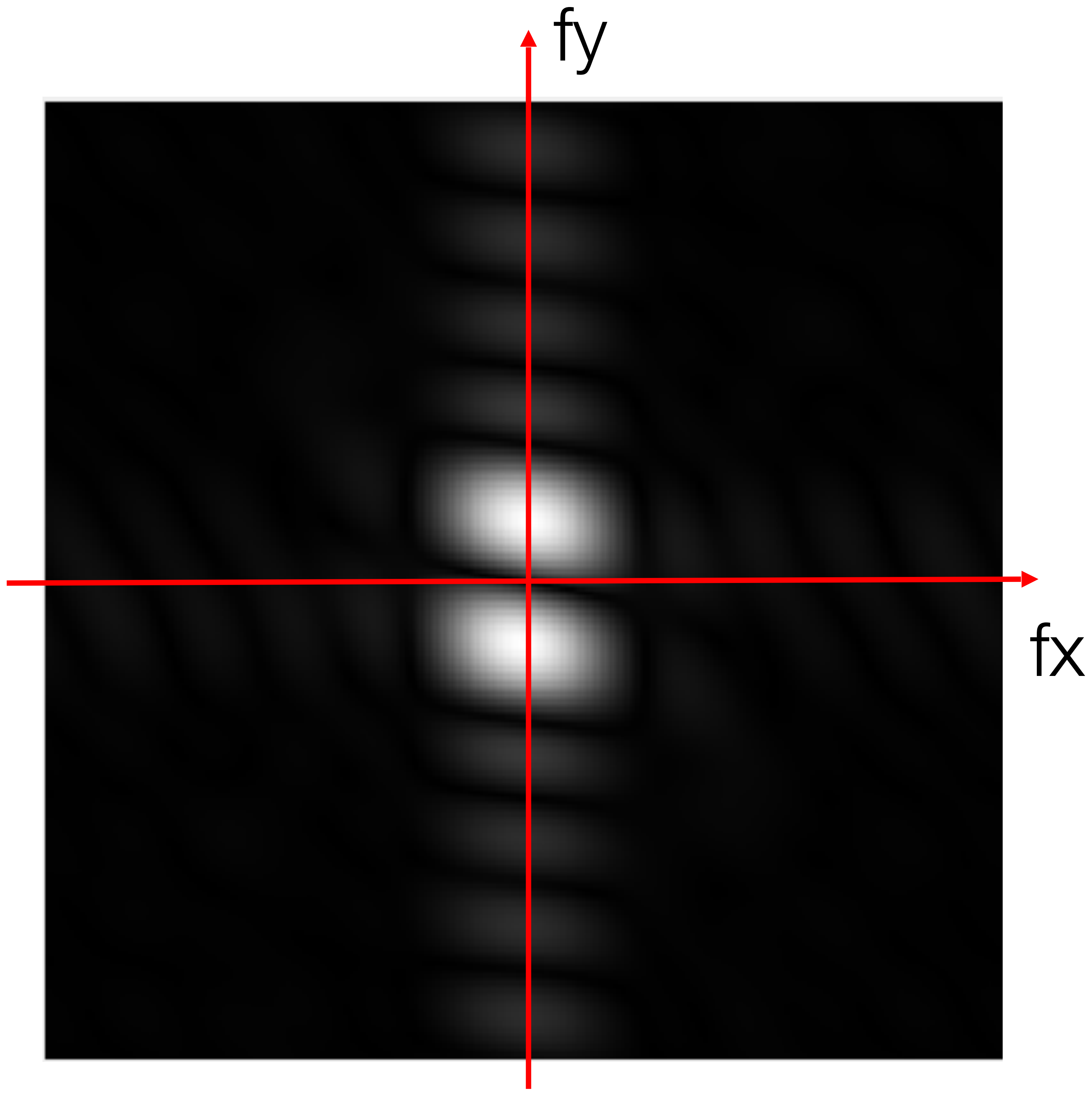


11x11 convolution kernel
(3 color channels)

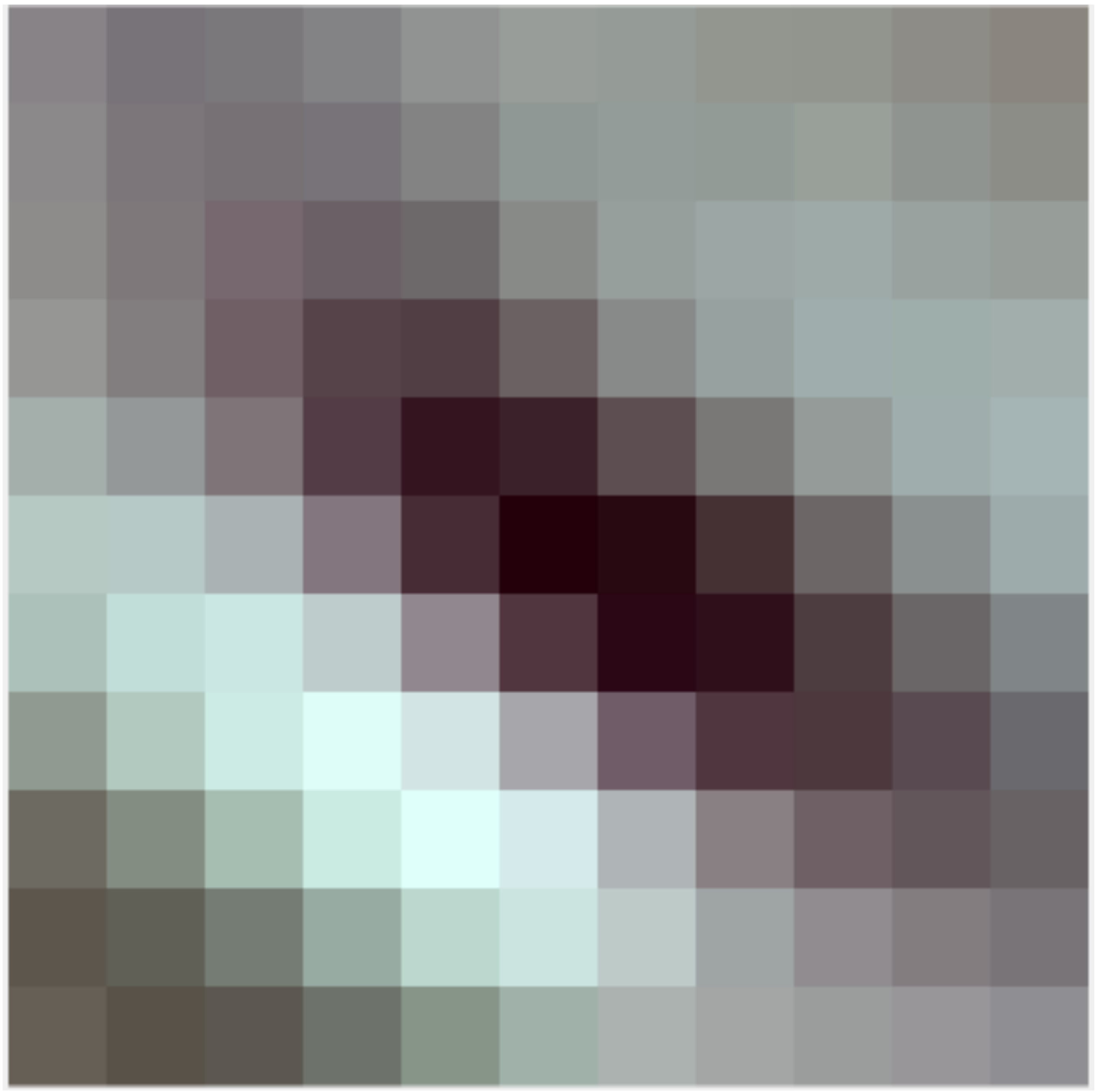
Get to know your units

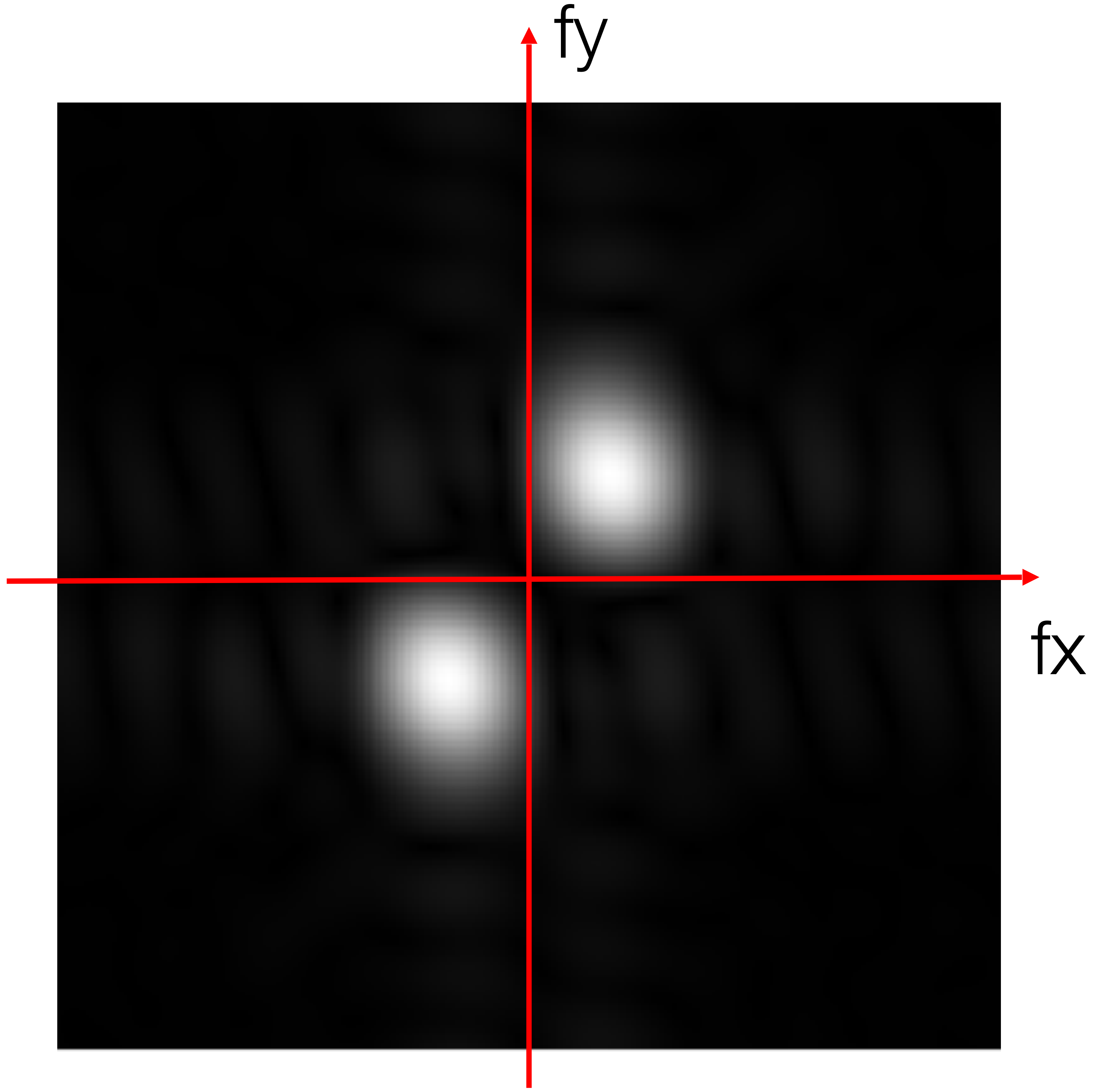


Get to know your units

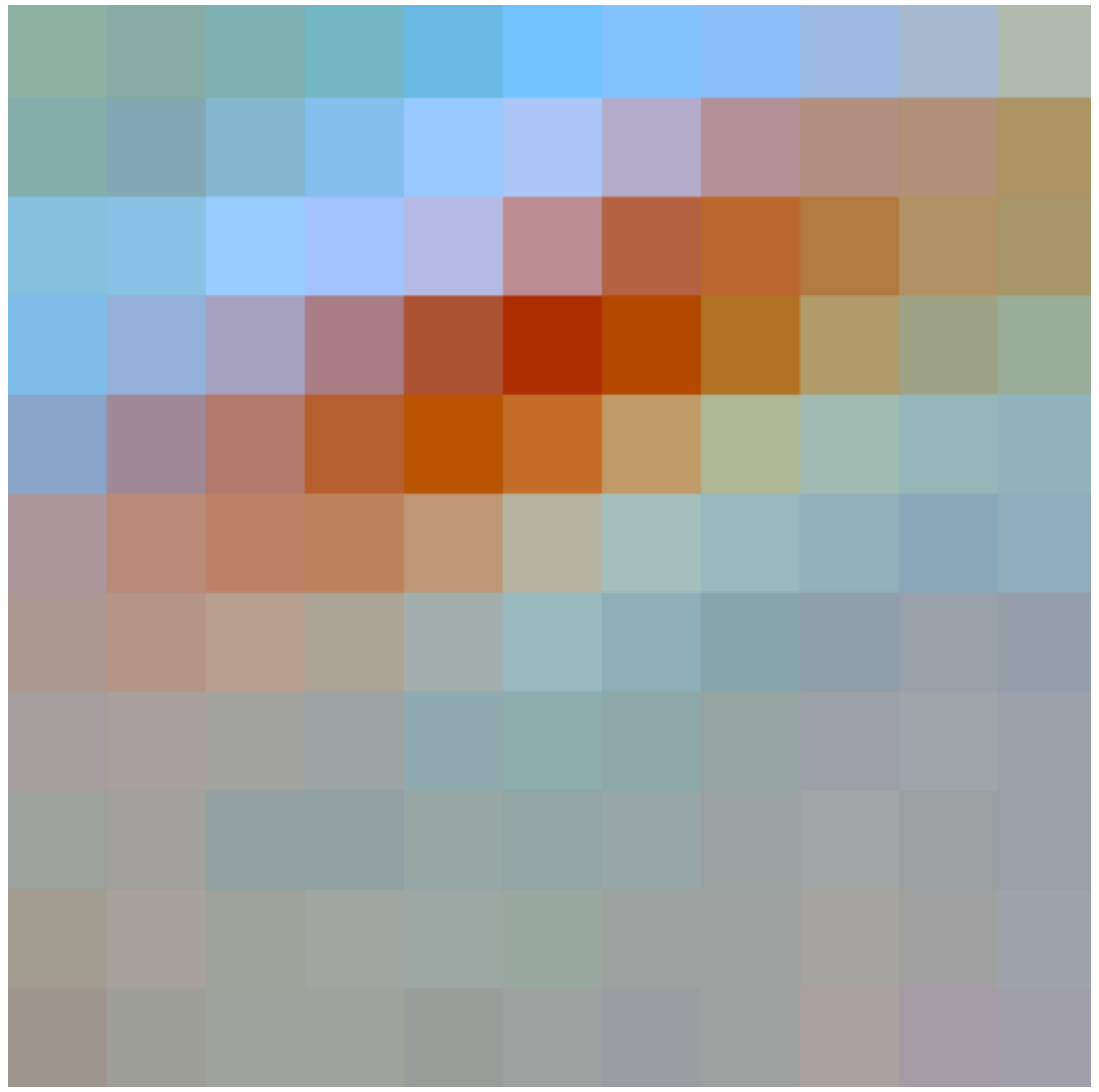


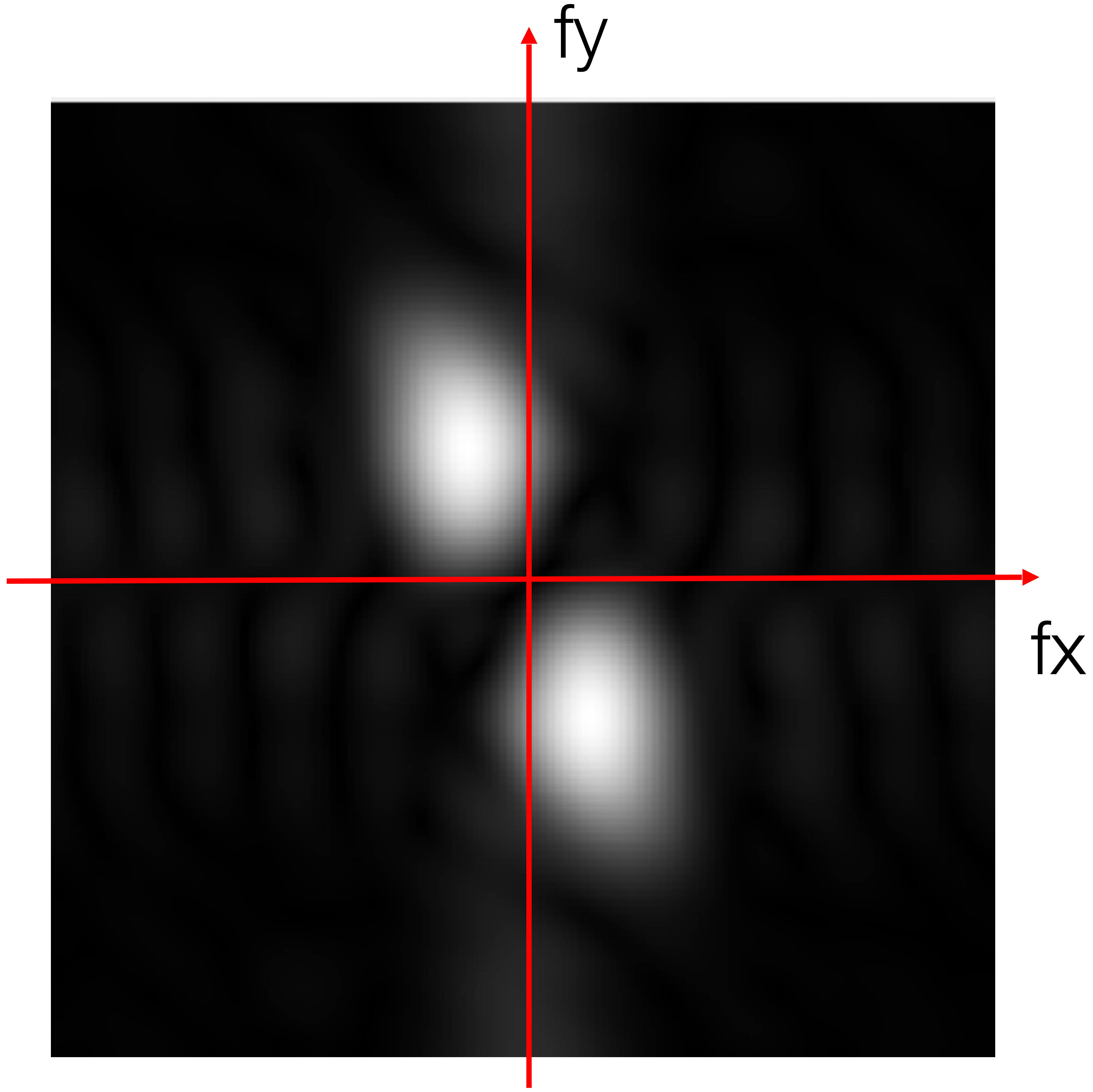
Get to know your units



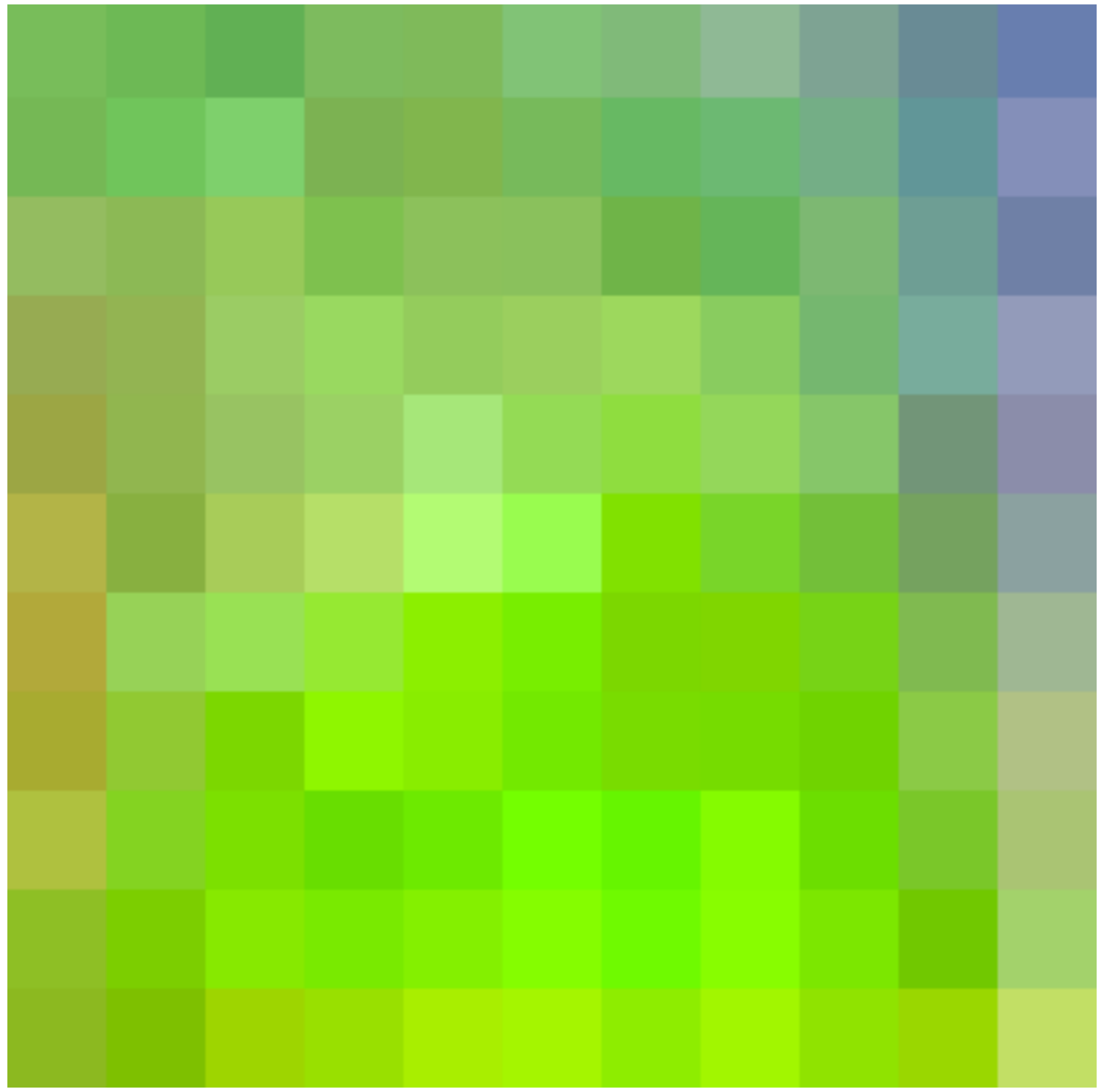


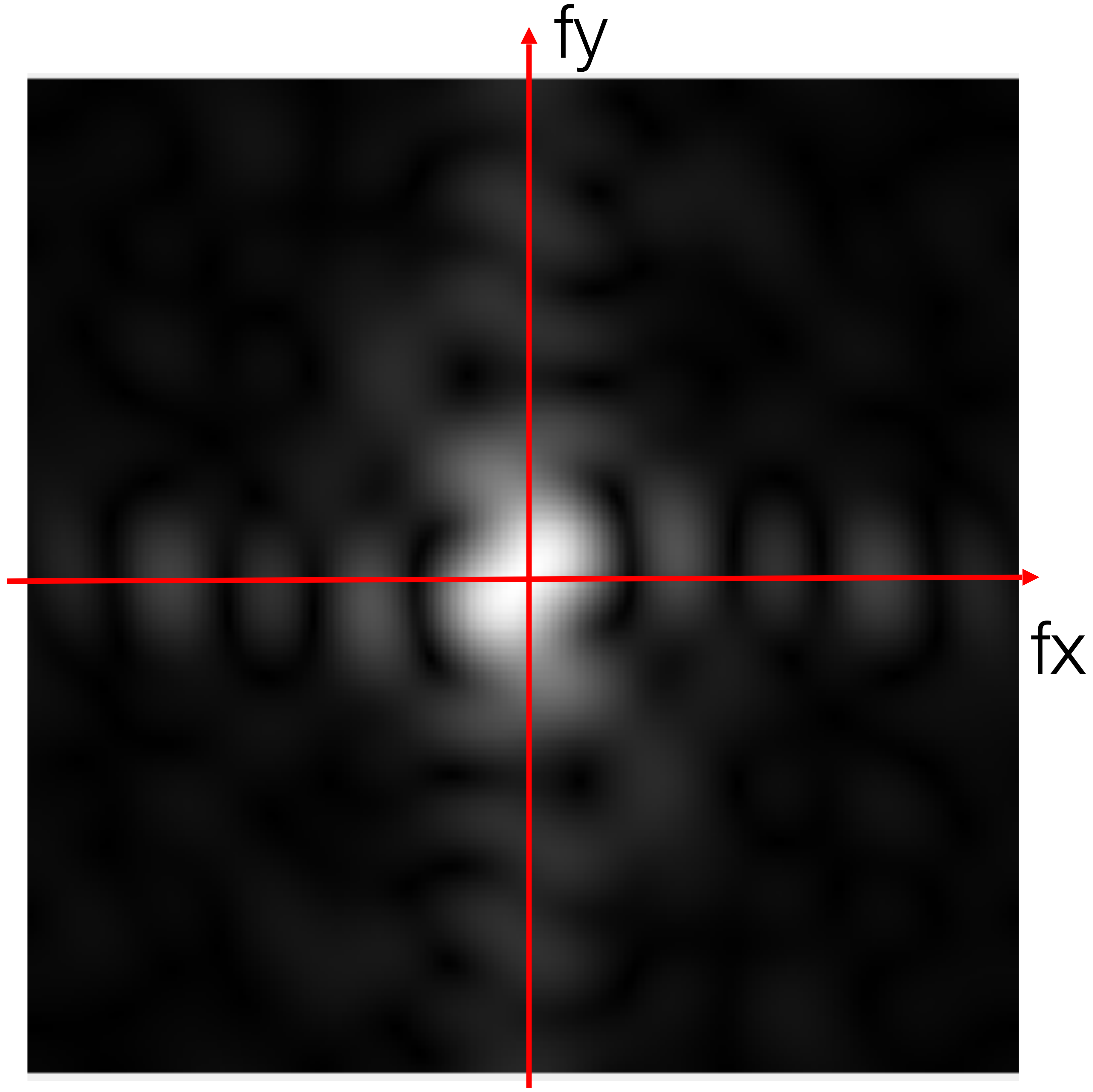
Get to know your units



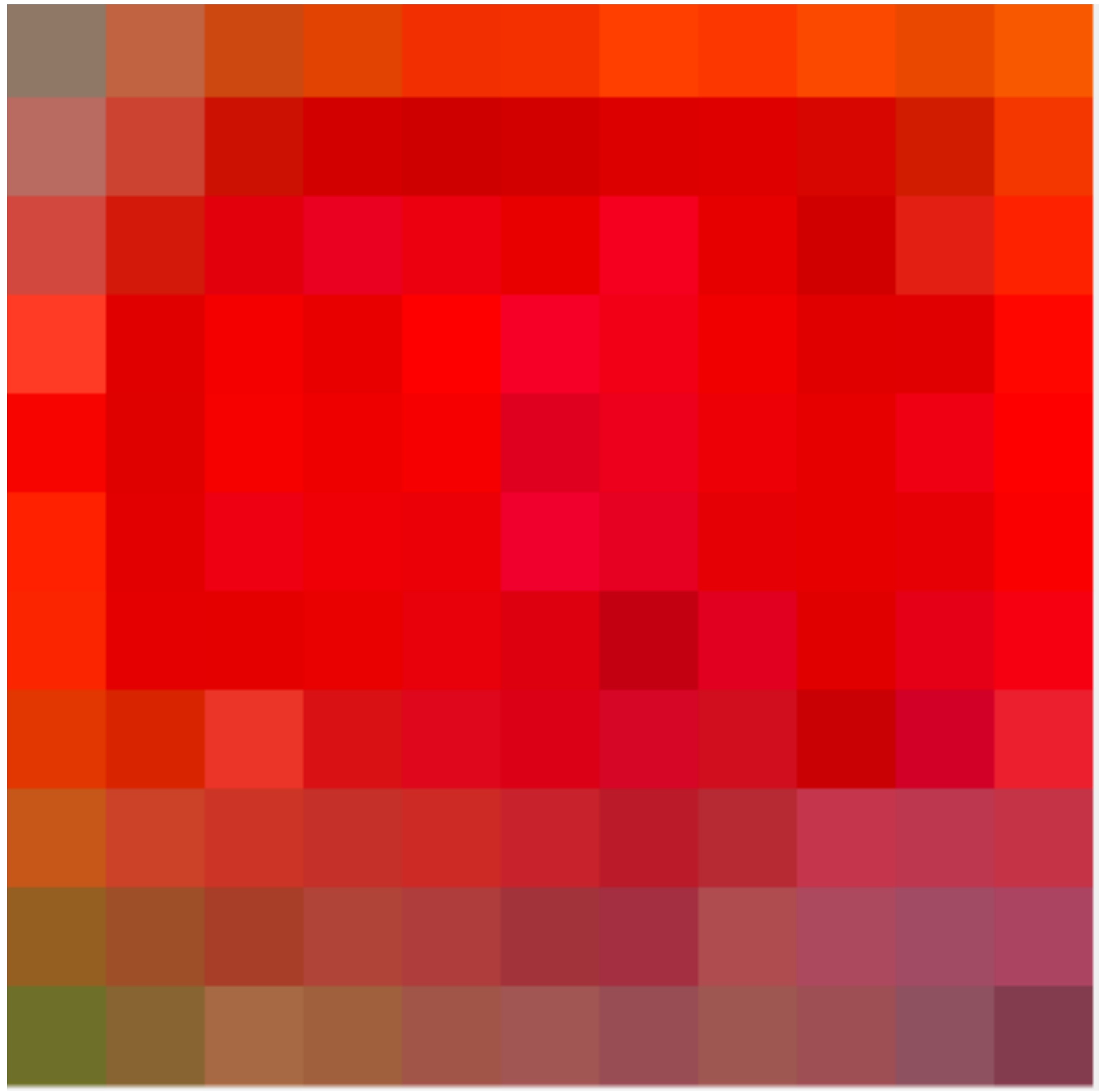


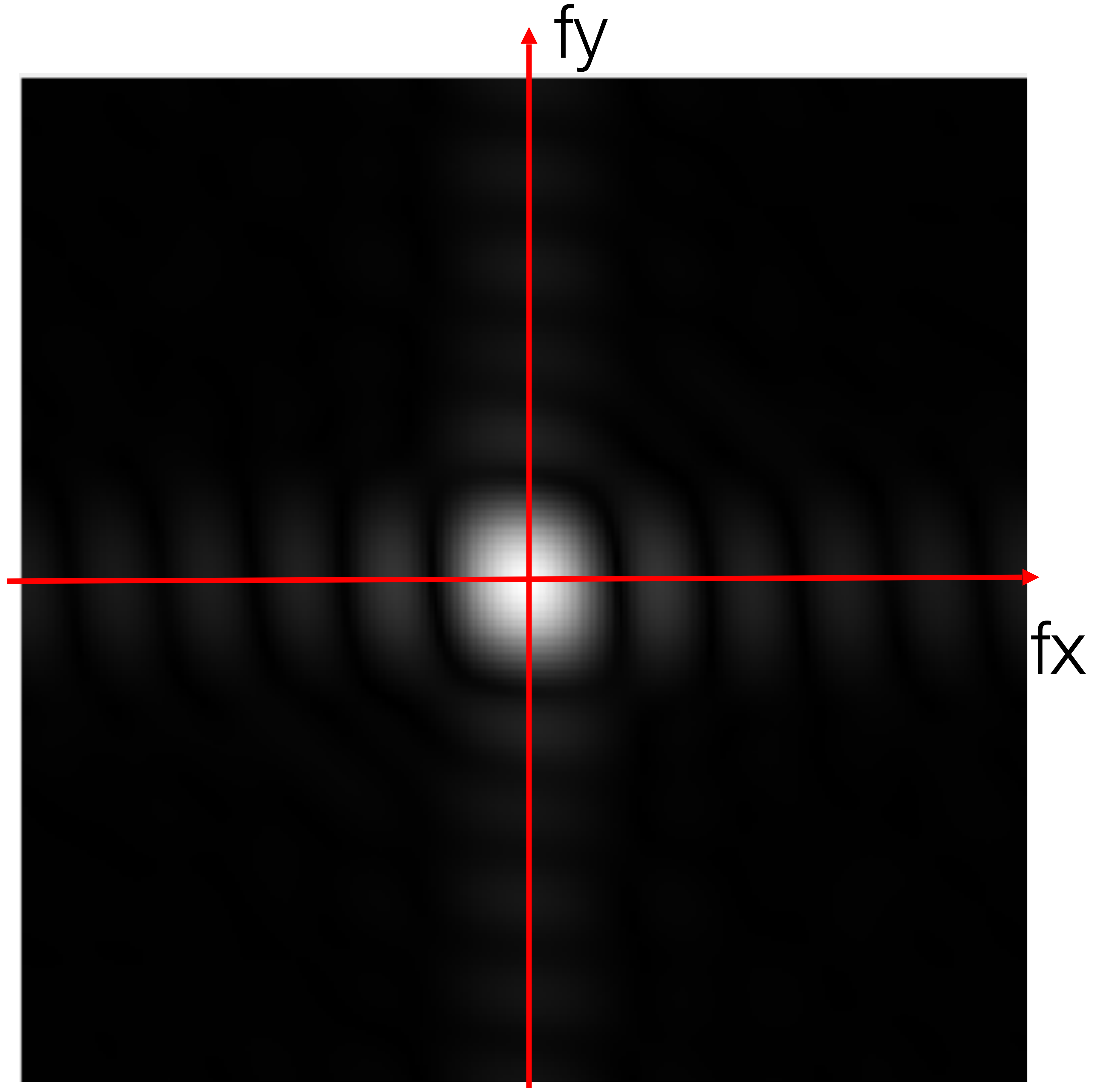
Get to know your units



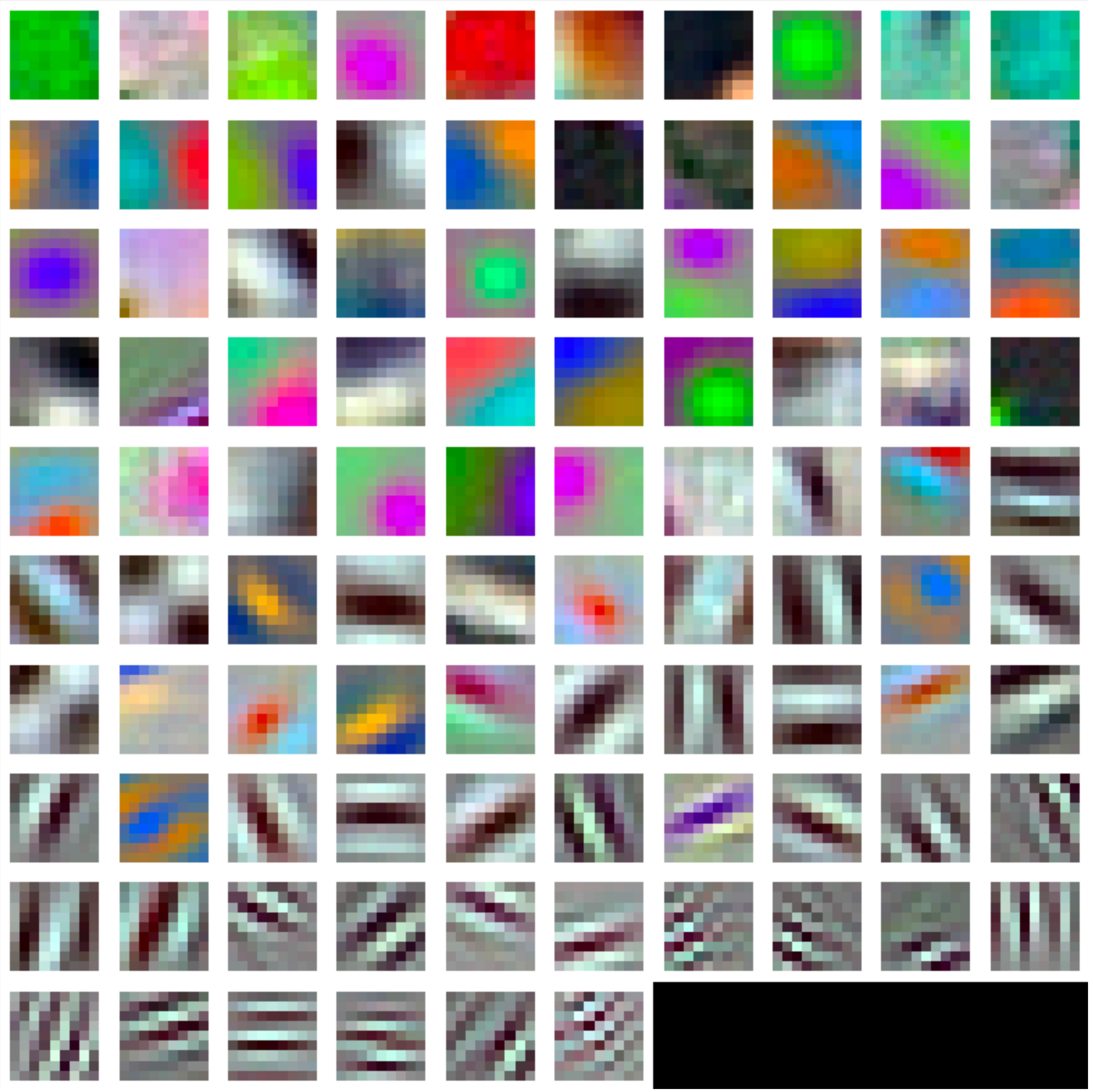


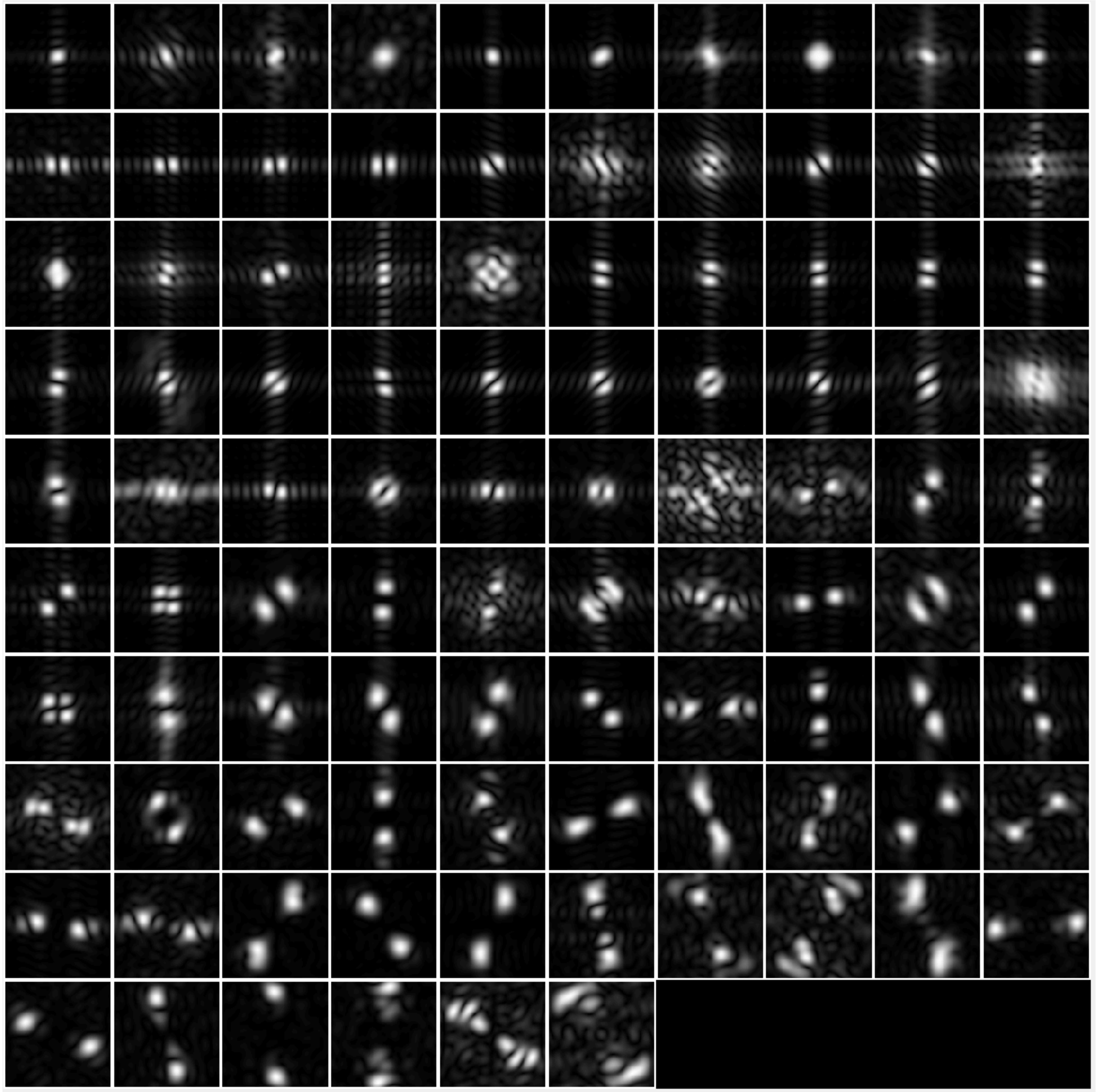
Get to know your units





Get to know your units

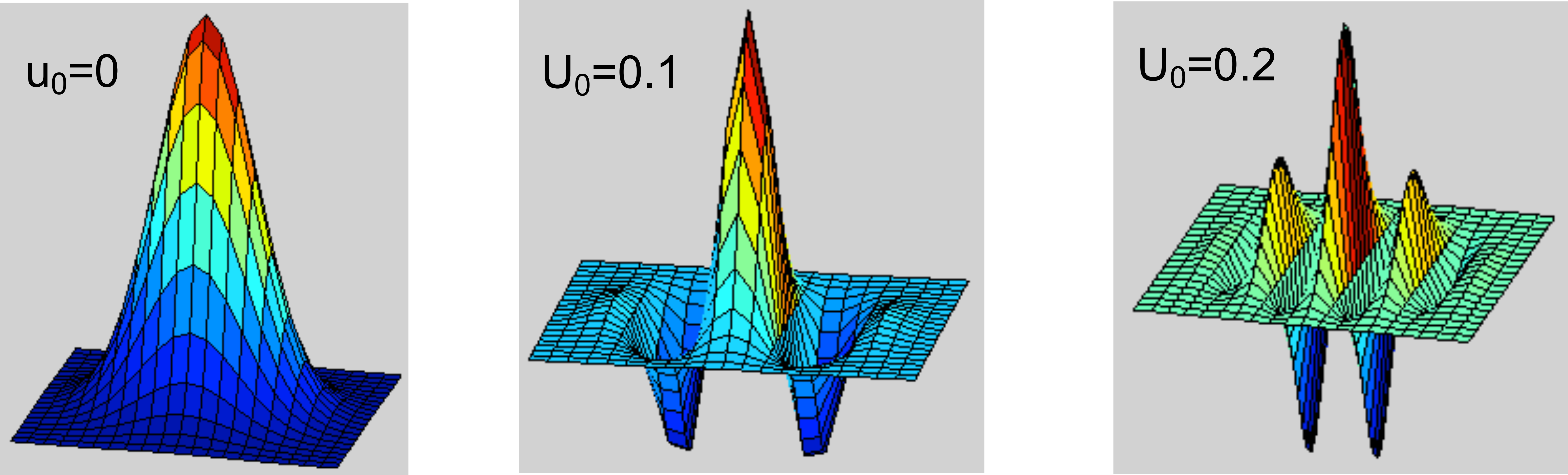




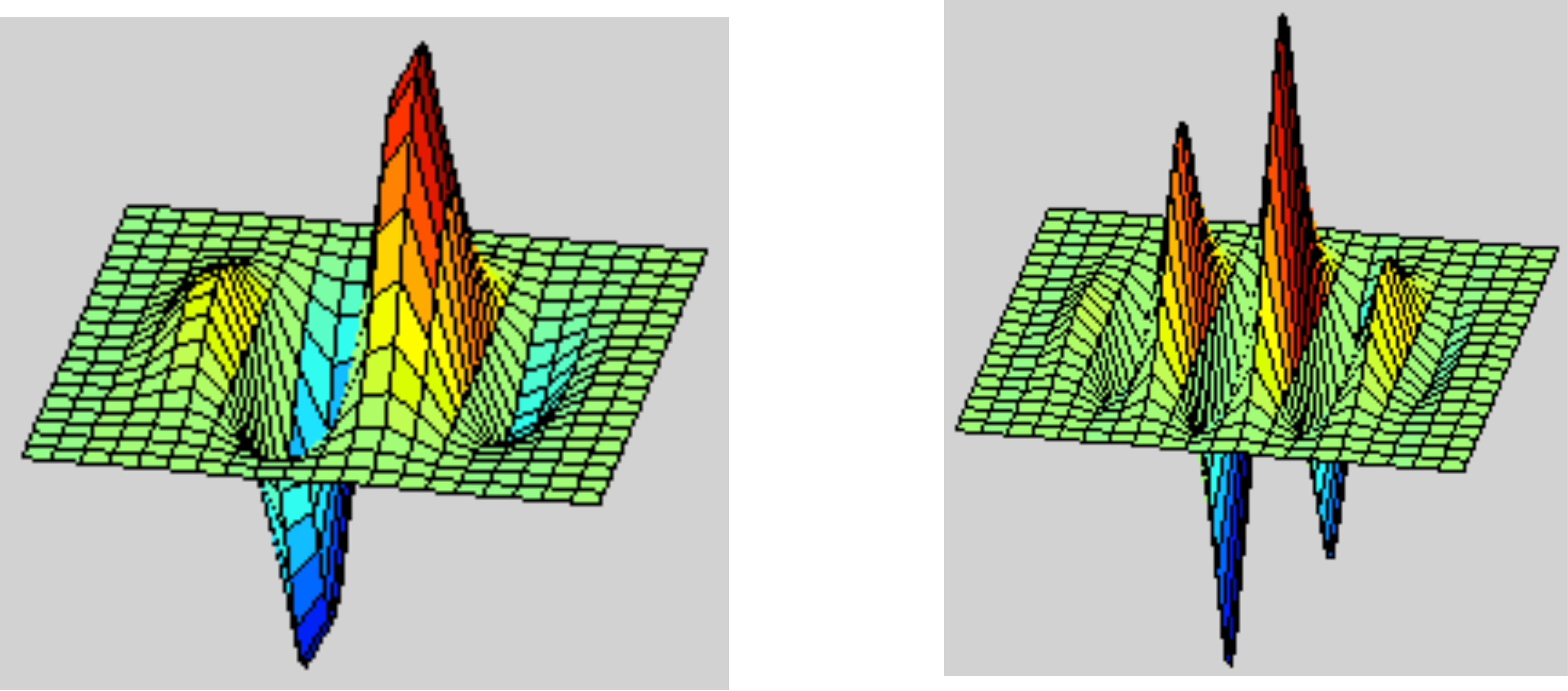
96 Units in conv1

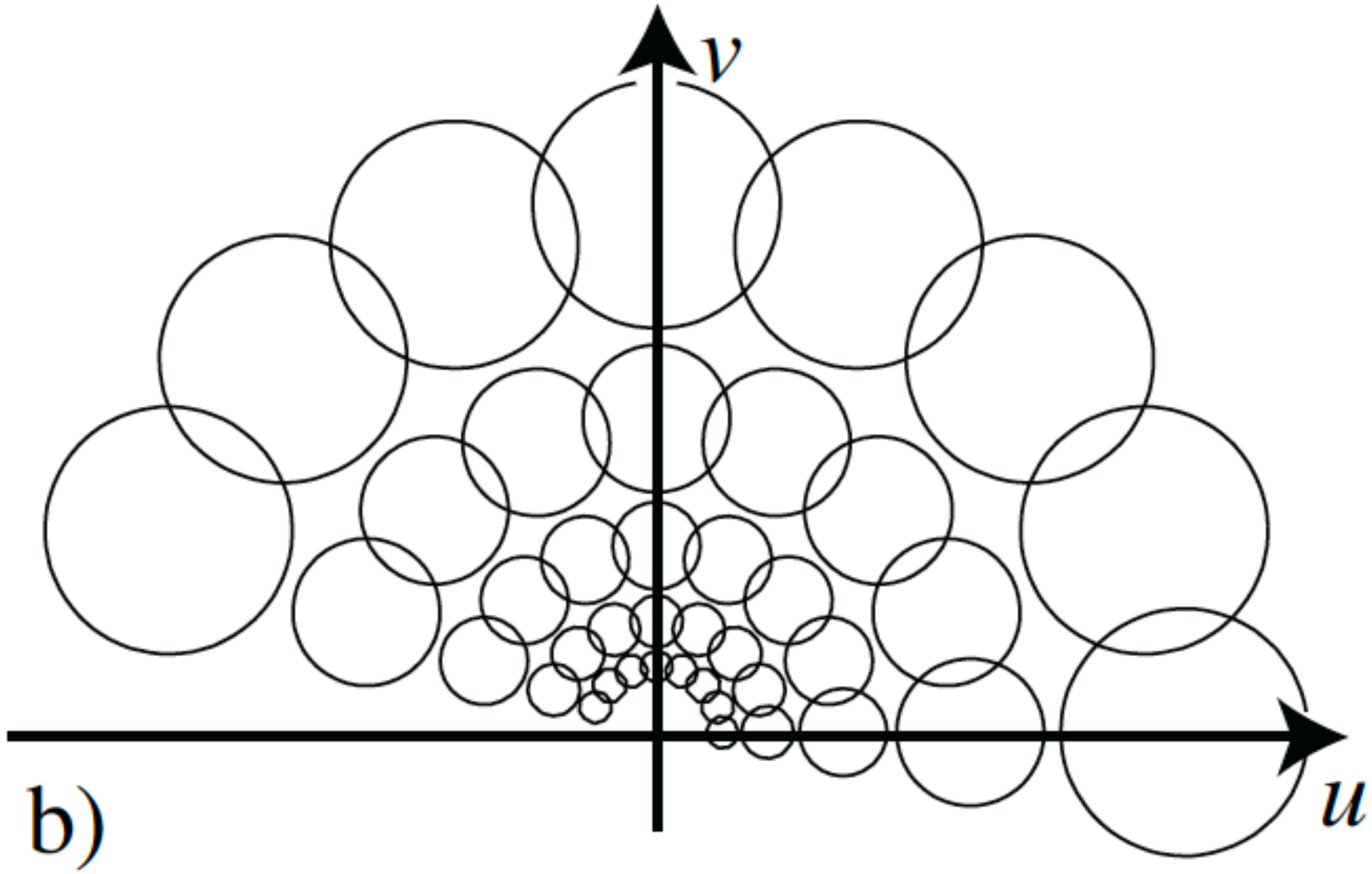
Gabor wavelets

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

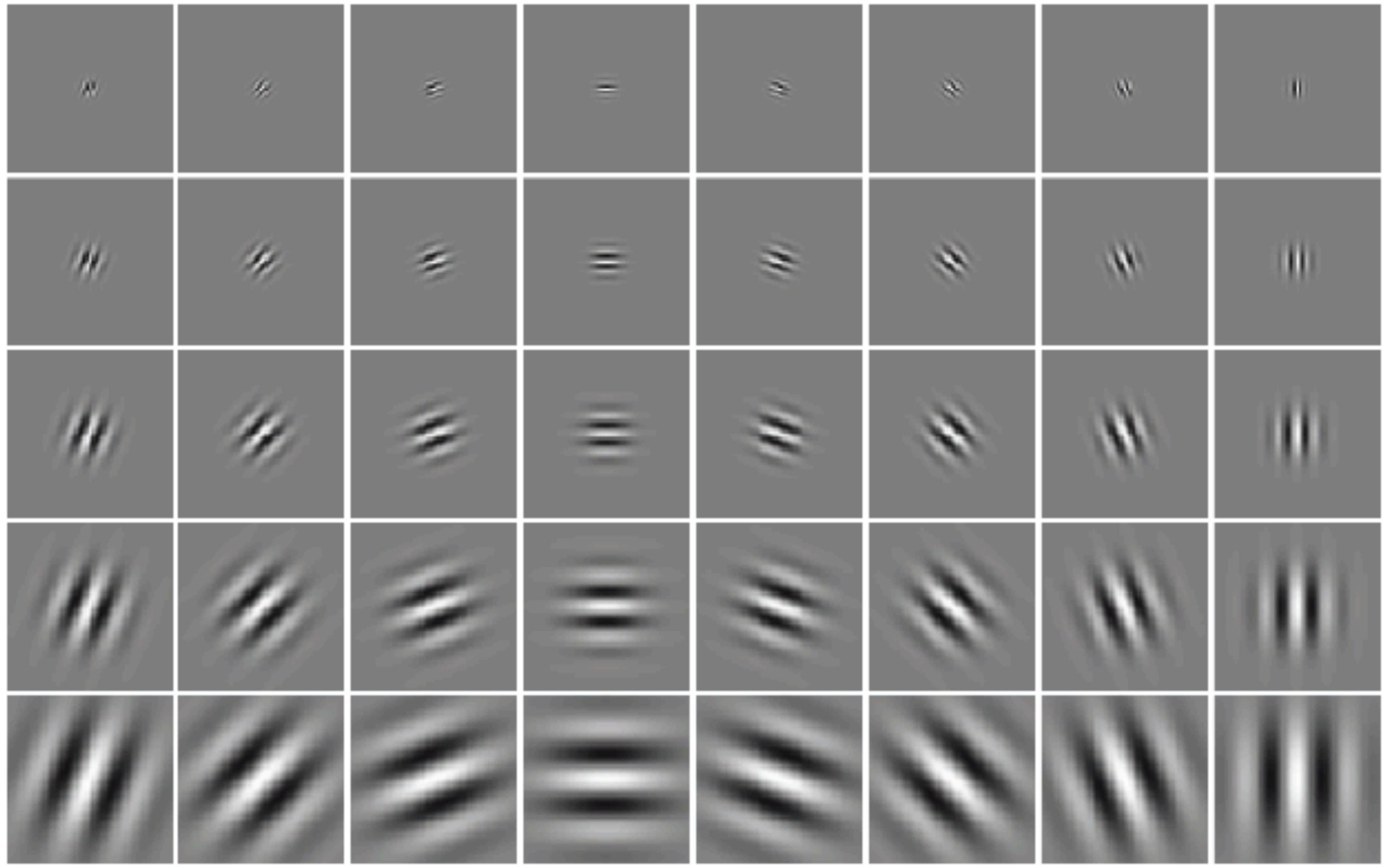


$$\psi_s(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \sin(2\pi u_0 x)$$





b)



Comparing Human and Machine Perception

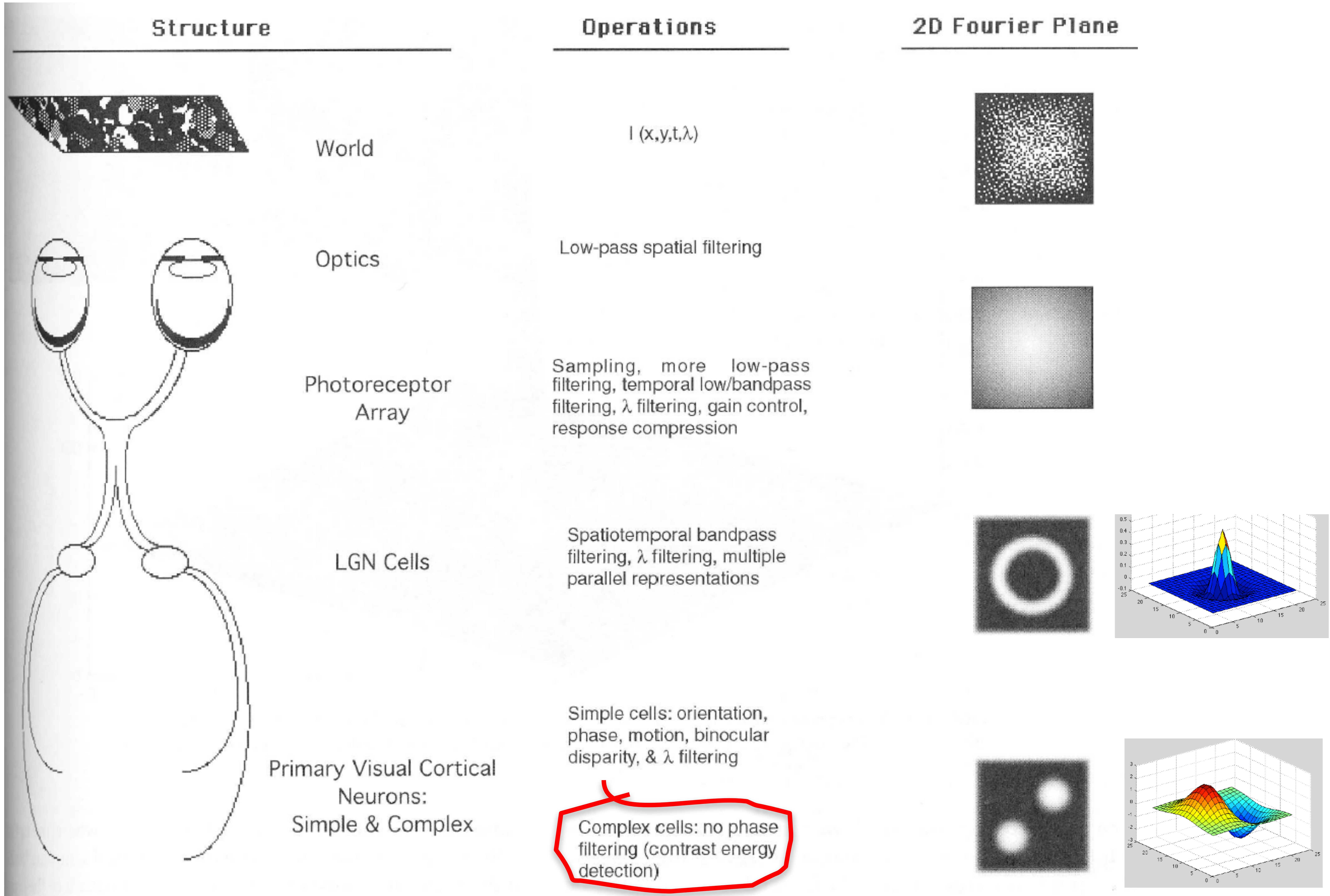


FIGURE 1 Schematic overview of the processing done by the early visual system. On the left, are some of the major structures to be discussed; in the middle, are some of the major operations done at the associated structure; in the right, are the 2-D Fourier representations of the world, retinal image,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

John Daugman, 1988

IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, VOL. 36, NO. 7, JULY 1988

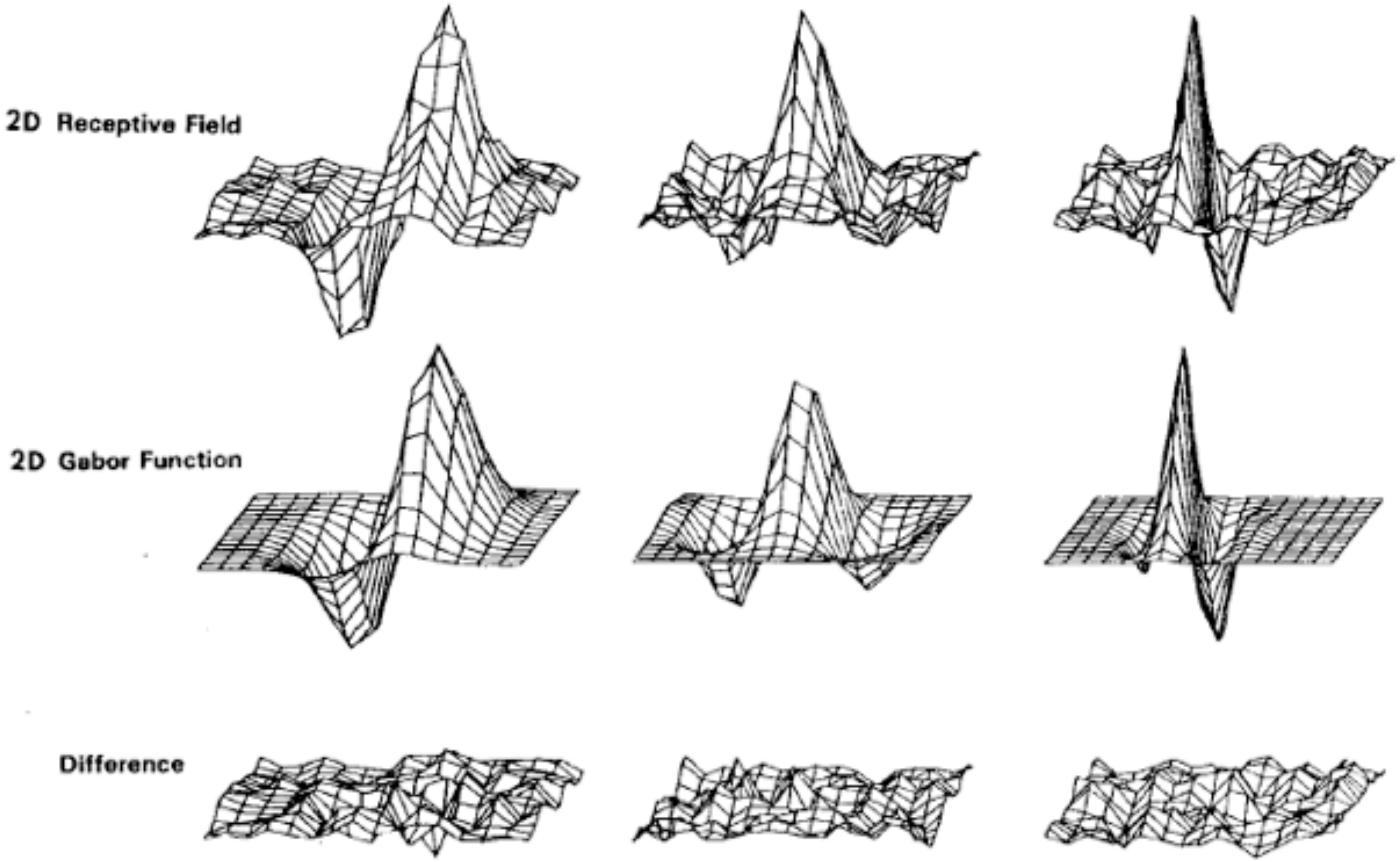
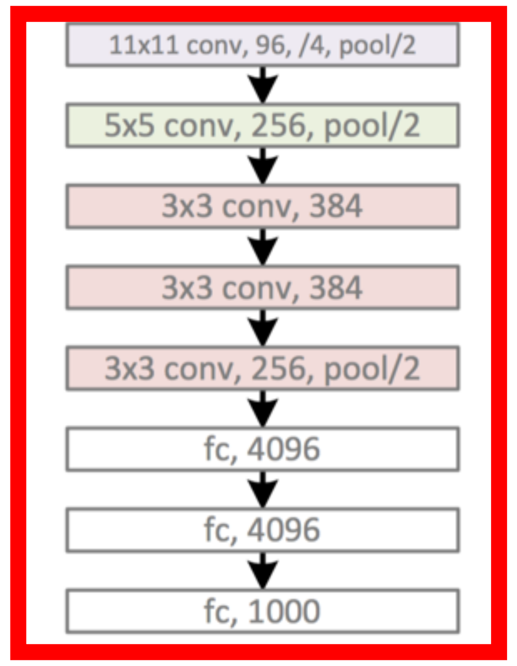


Fig. 5. Top row: illustrations of empirical 2-D receptive field profiles measured by J. P. Jones and L. A. Palmer (personal communication) in simple cells of the cat visual cortex. Middle row: best-fitting 2-D Gabor elementary function for each neuron, described by (10). Bottom row: residual error of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

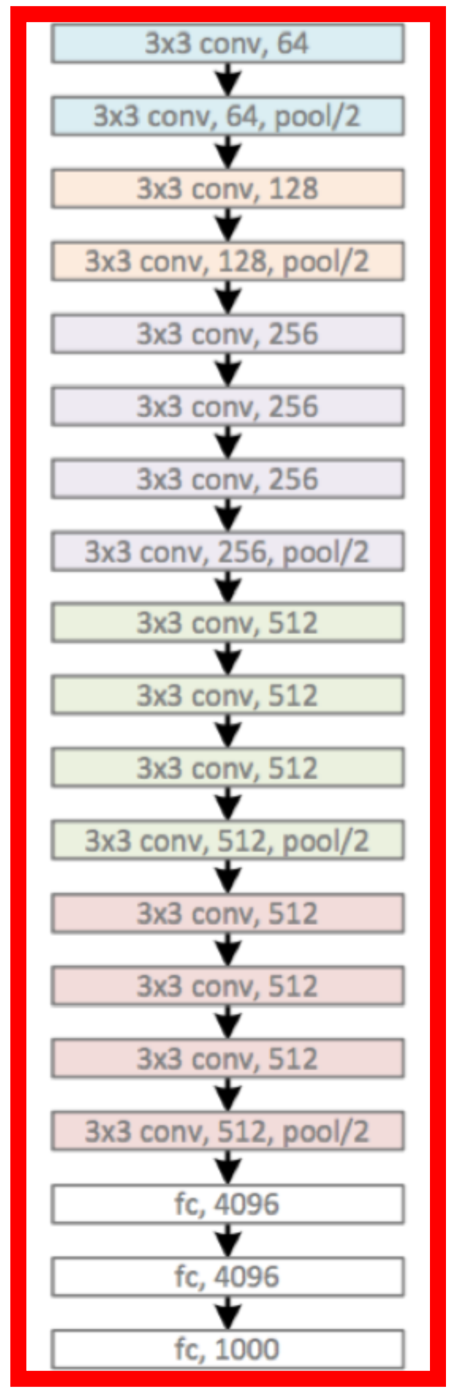
Deep Neural Networks for Visual Recognition

2012: AlexNet
5 conv. layers



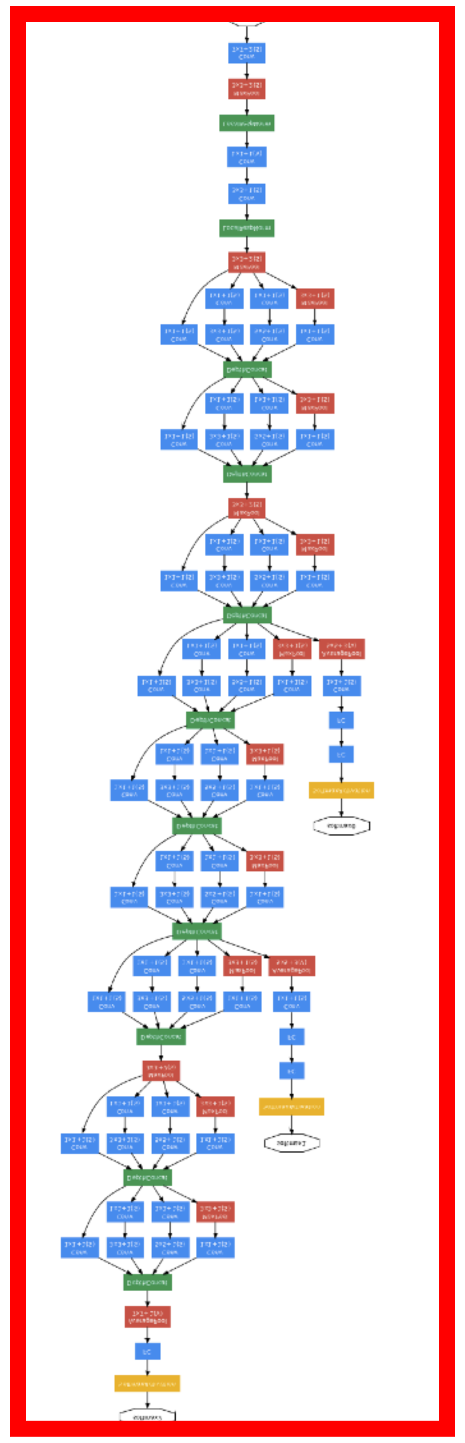
Error: 15.3%

2014: VGG
16 conv. layers

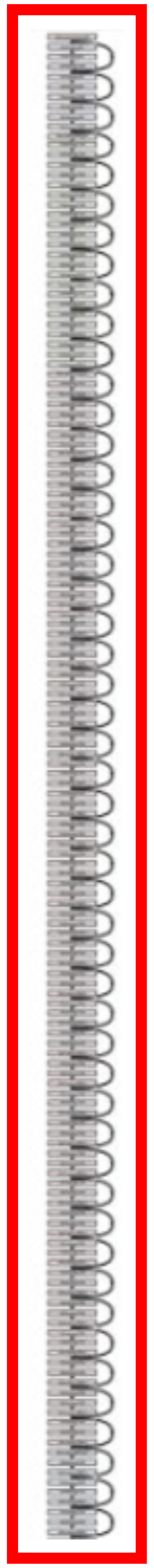


Error: 8.5%

2015: GoogLeNet
22 conv. layers



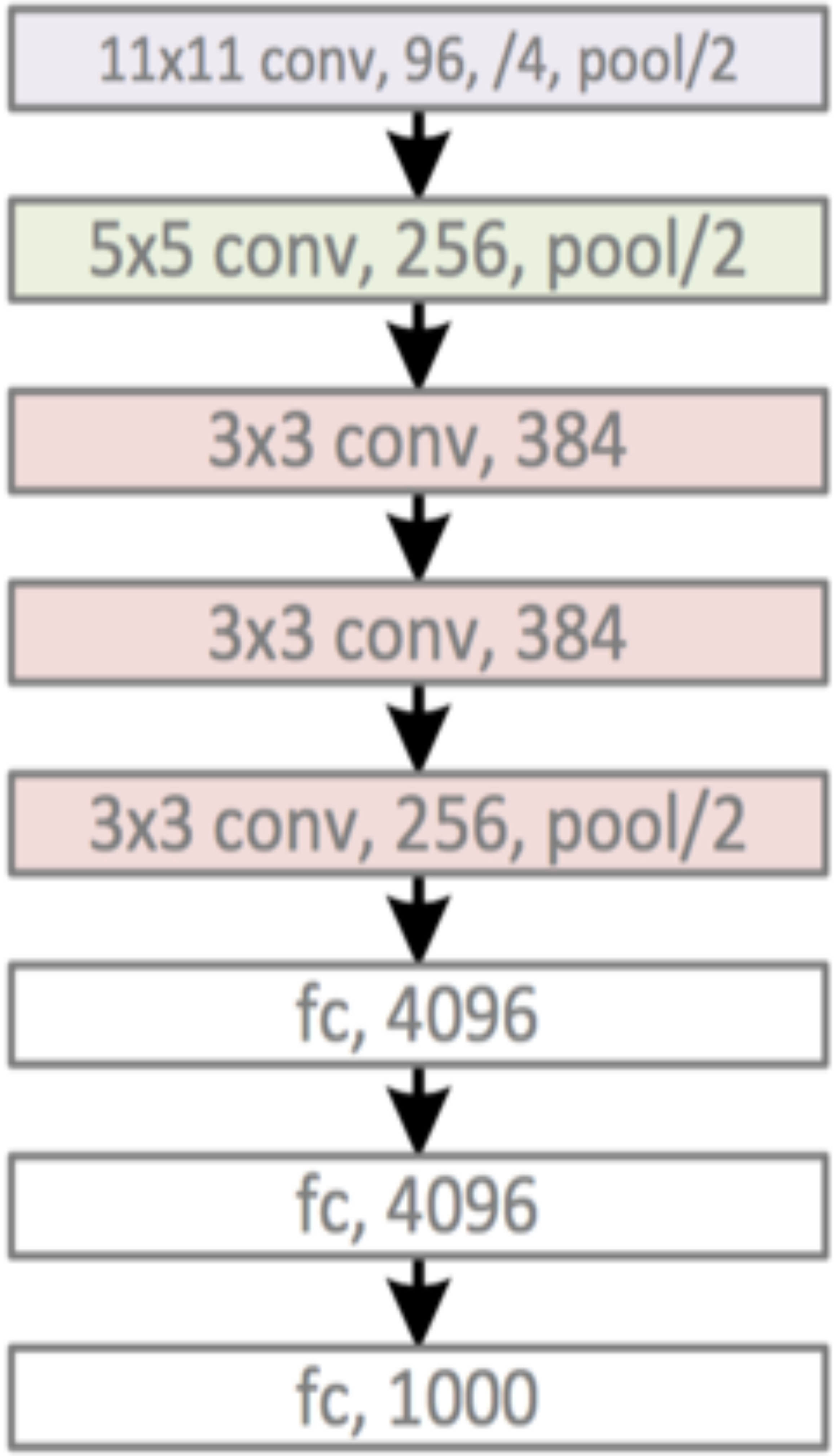
Error: 7.8%



2016: ResNet
>100 conv. layers

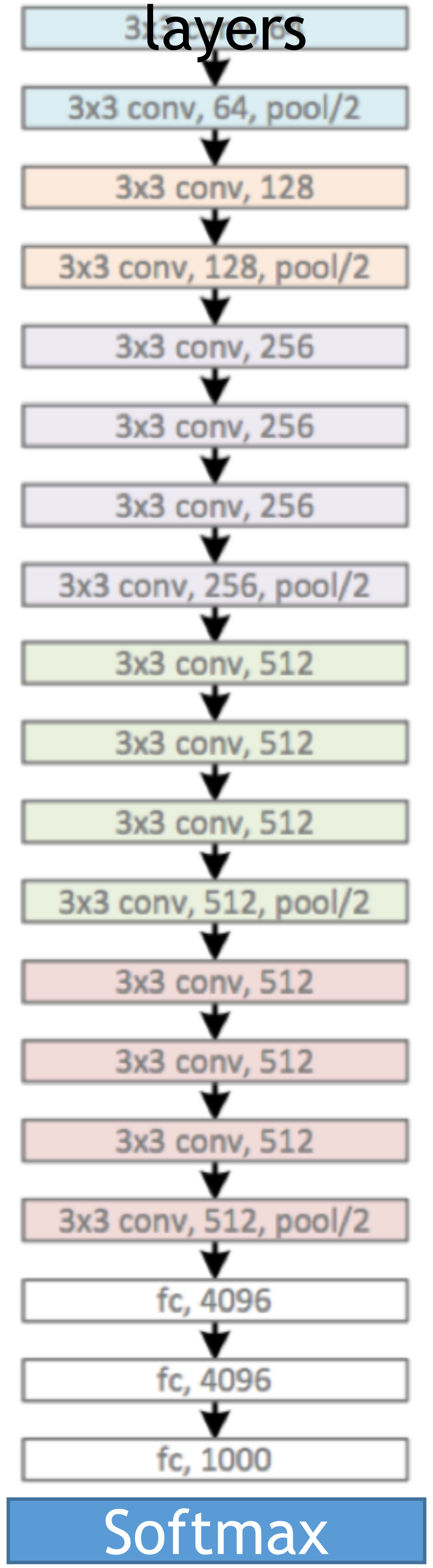
Error: 4.4%

2012: AlexNet
5 conv. layers



Error: 15.3%

2014: VGG
16 conv. layers

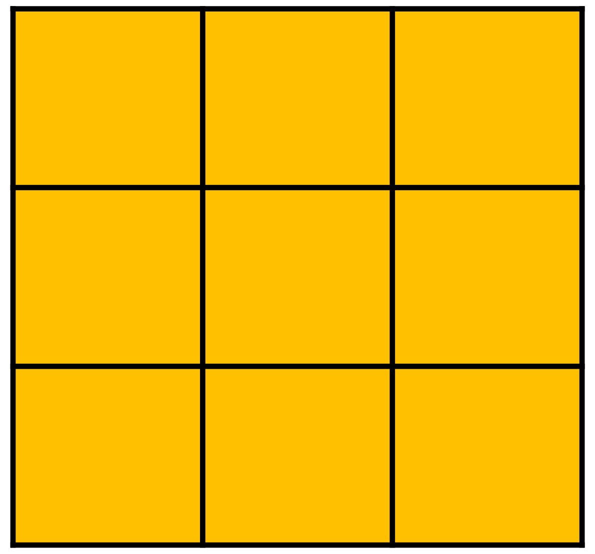


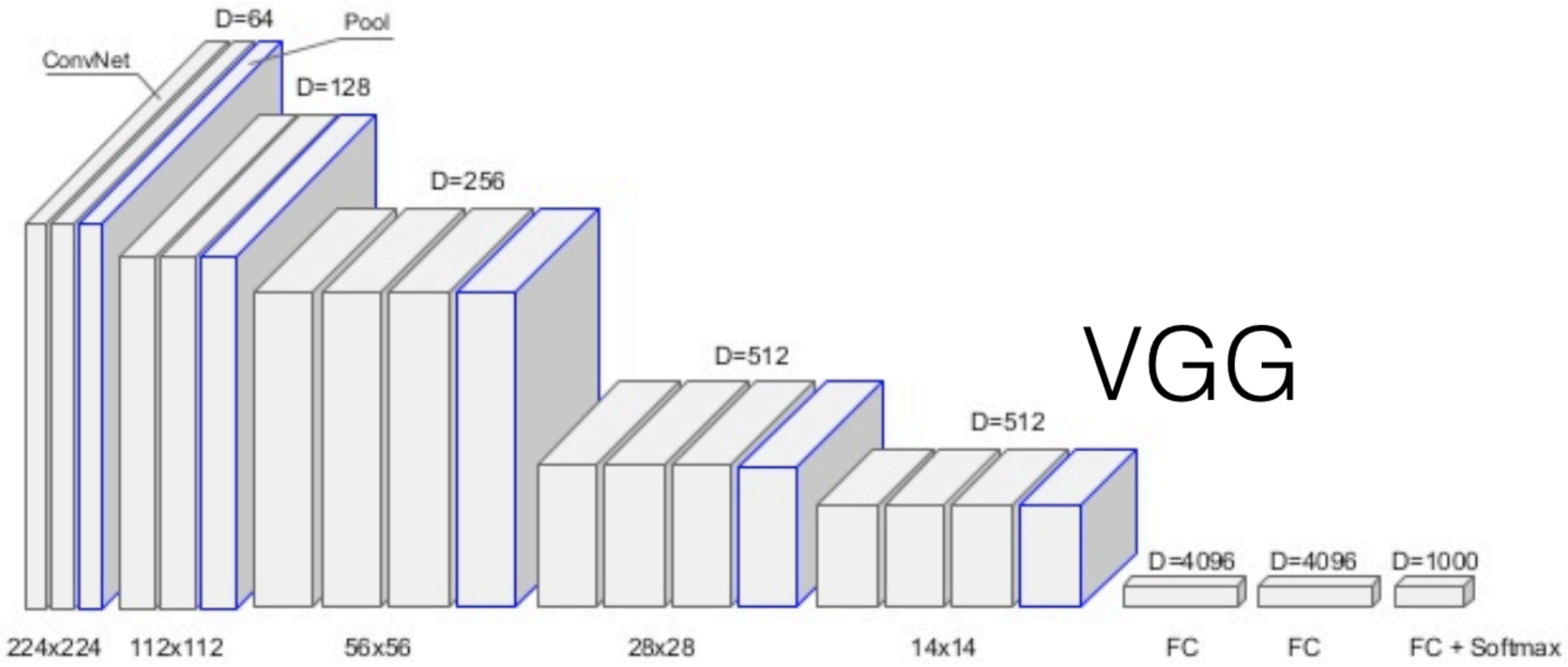
Error: 8.5%

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

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

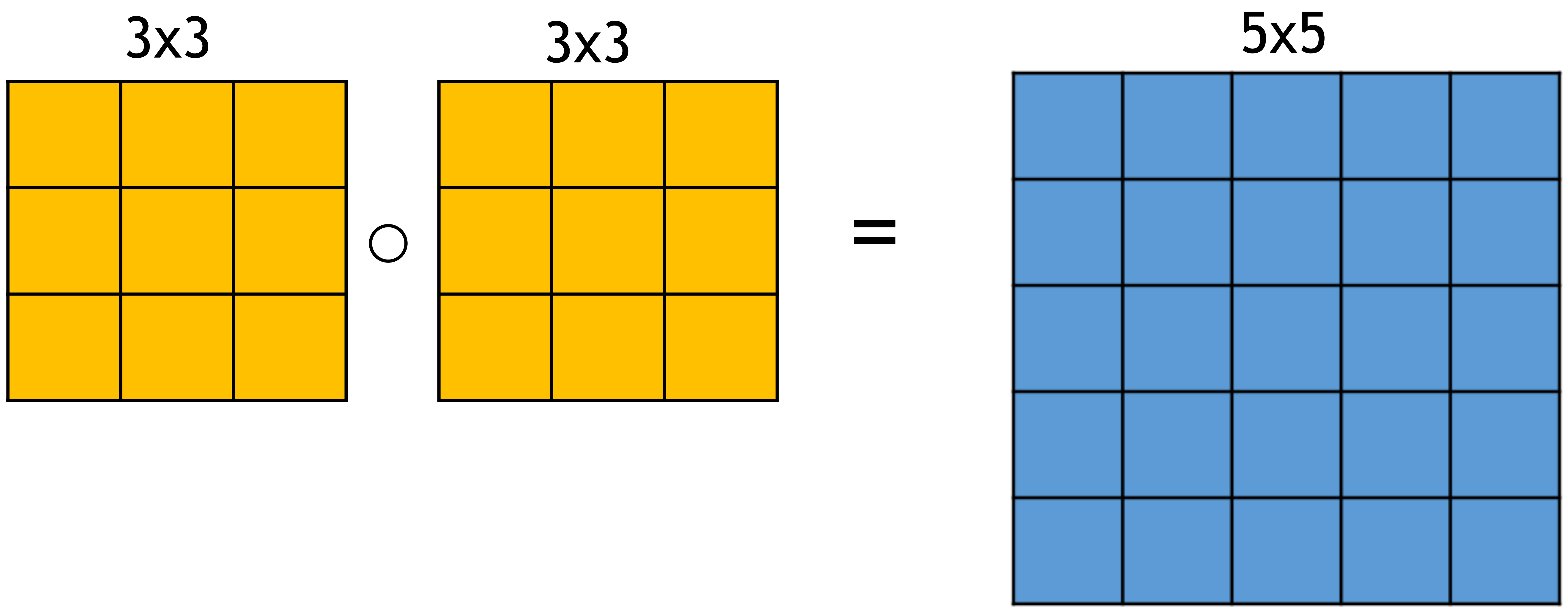
Small convolutional kernels: 3x3
ReLu non-linearities
>100 million parameters.



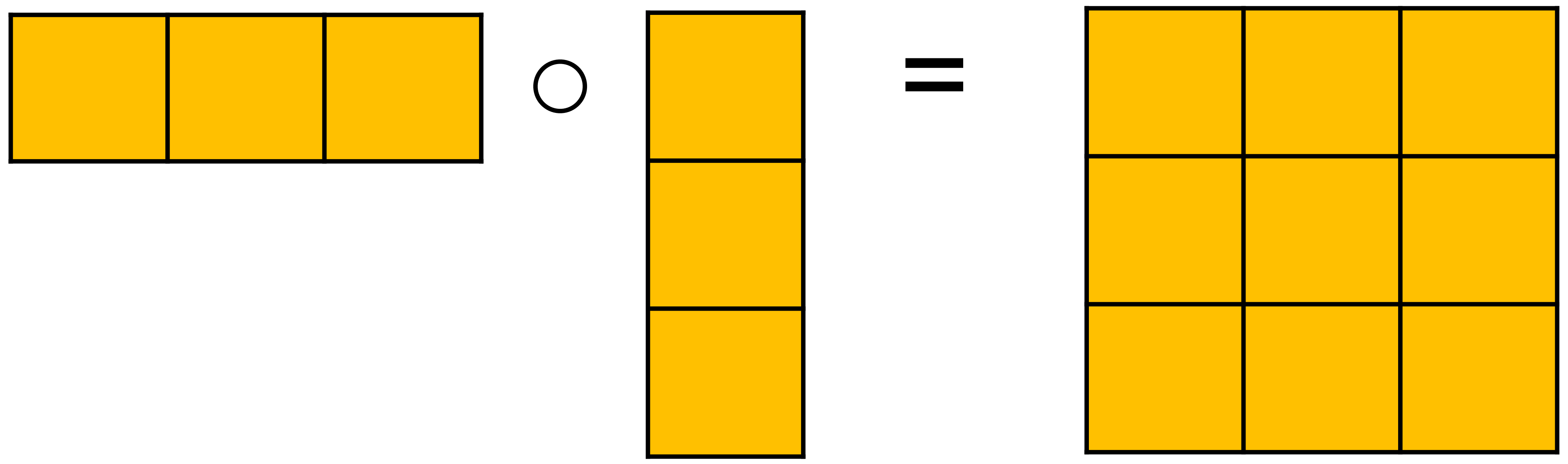


VGG

Chaining convolutions

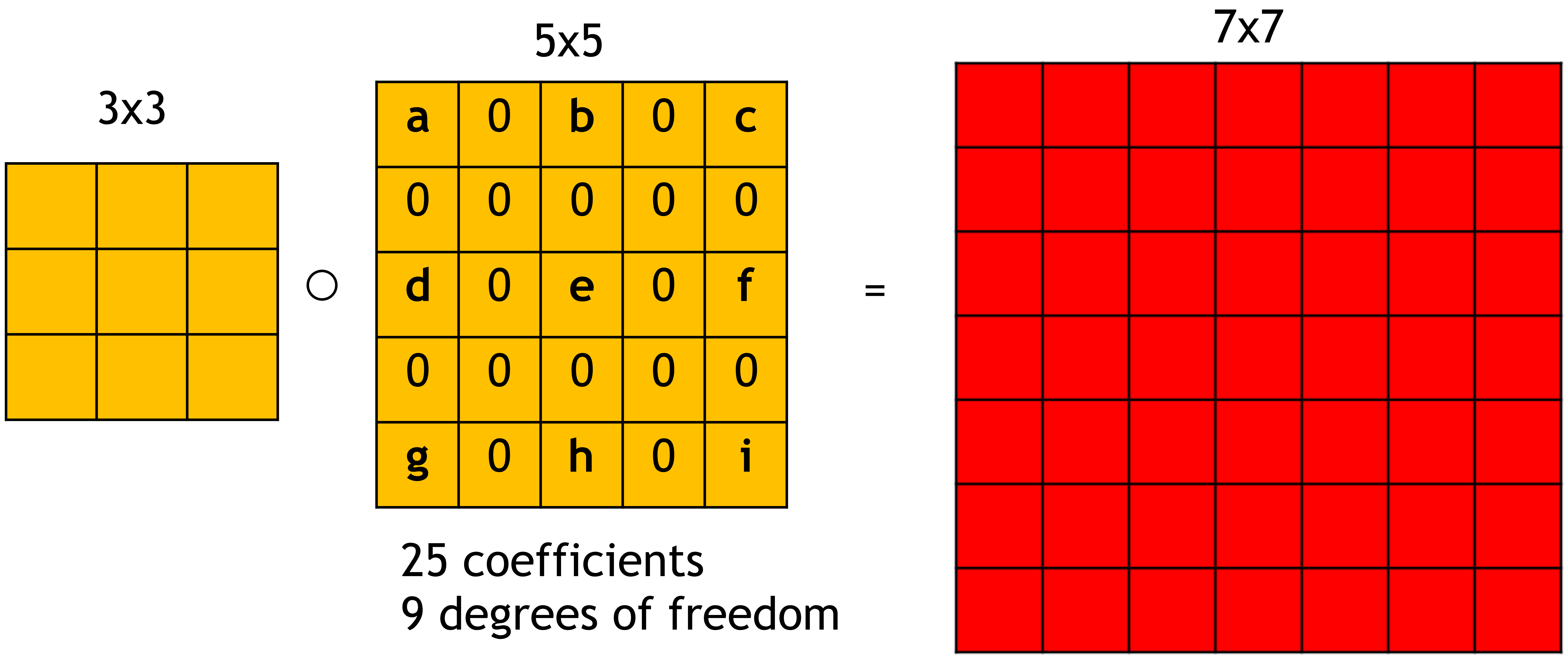


25 coefficients, but only
18 degrees of freedom



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

Dilated convolutions



25 coefficients
9 degrees of freedom

49 coefficients
18 degrees of freedom

What is lost?

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

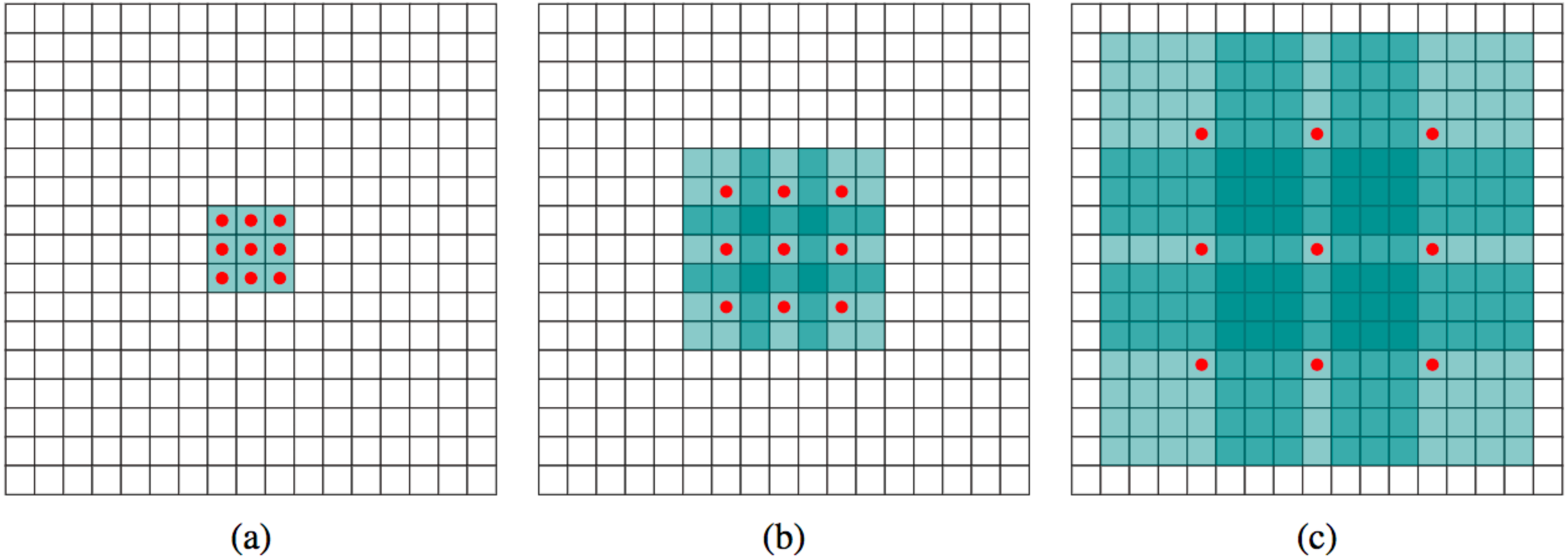


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element 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.

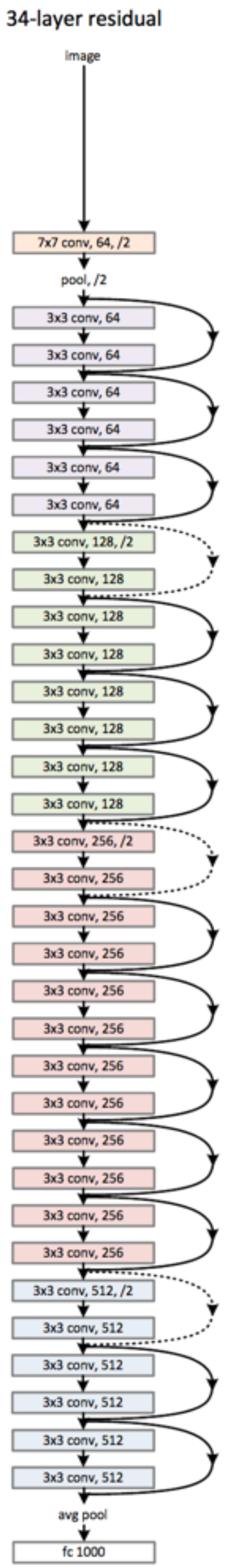
[https://arxiv.org/pdf/1511.07122.pdf]

2016: ResNet
>100 conv. layers

Deep Residual Learning for Image Recognition

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

Error: 4.4%



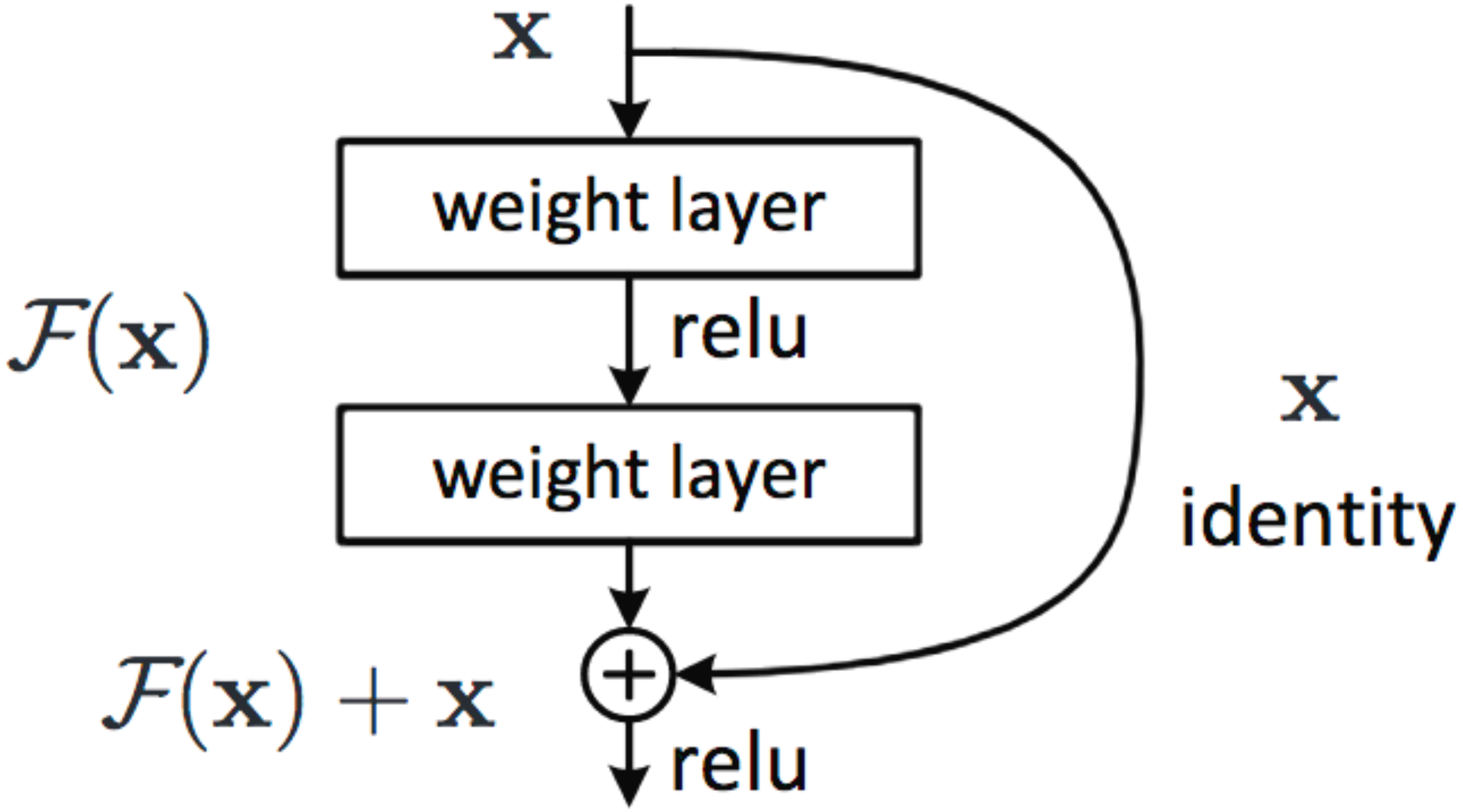
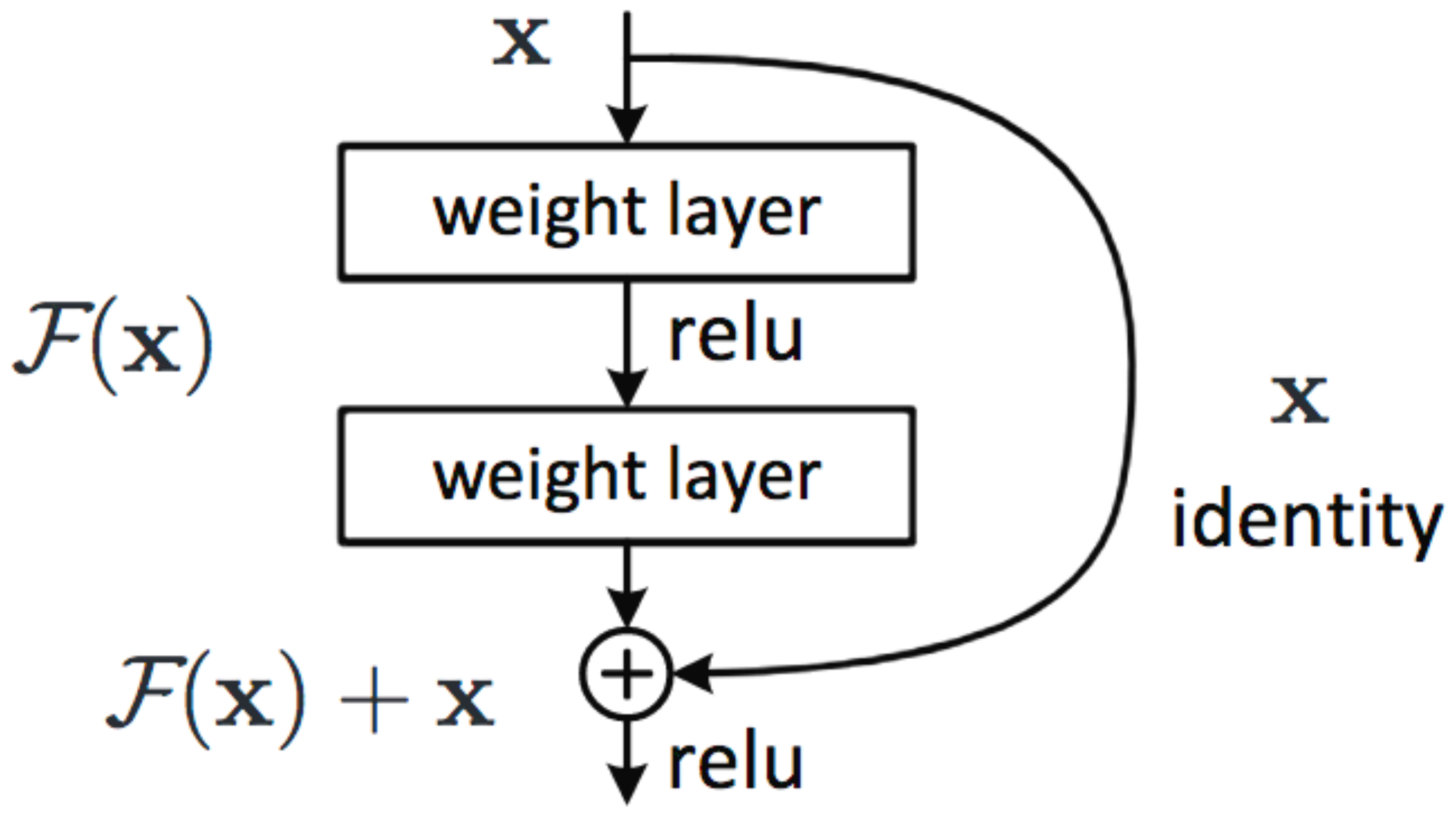
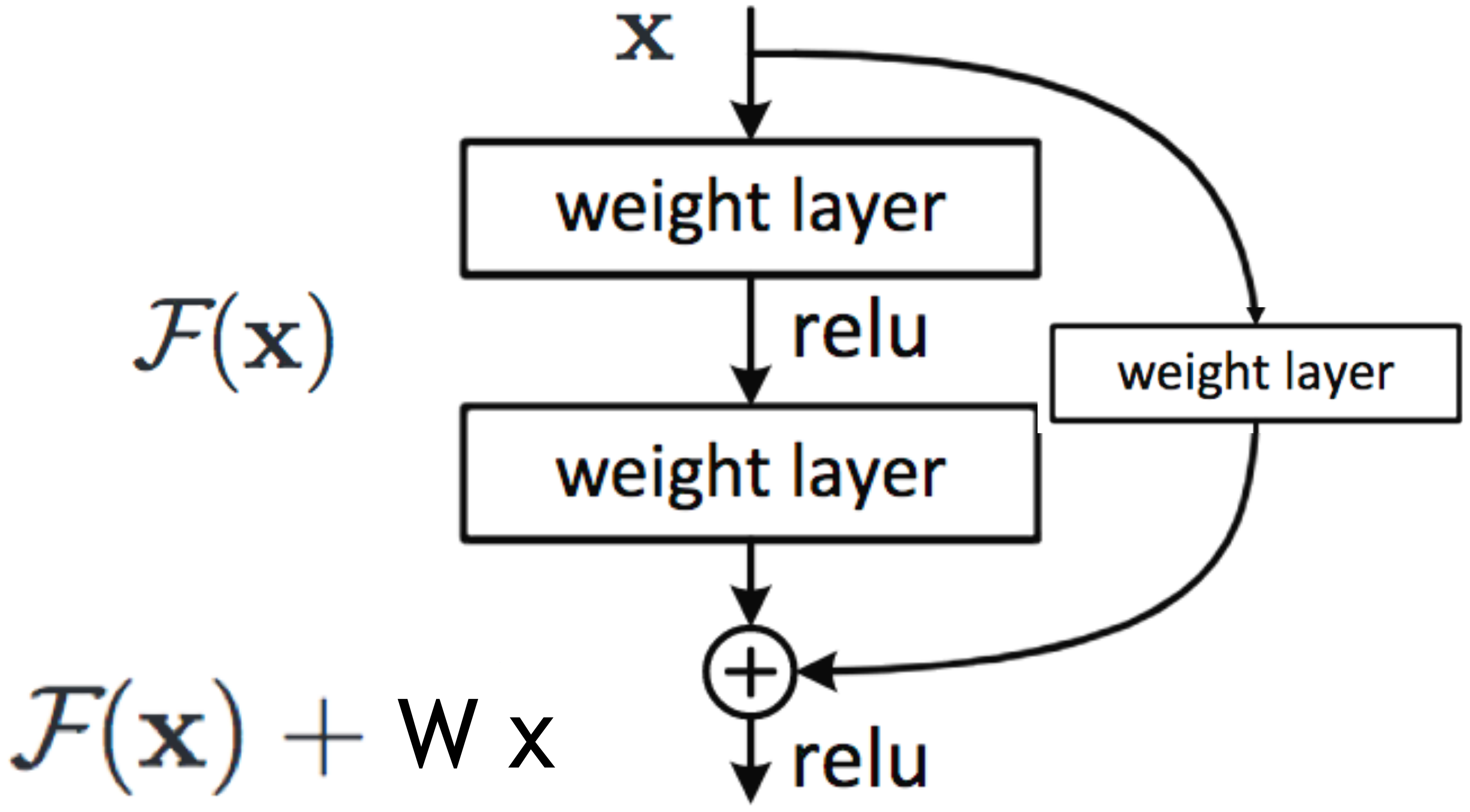


Figure 2. Residual learning: a building block.

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

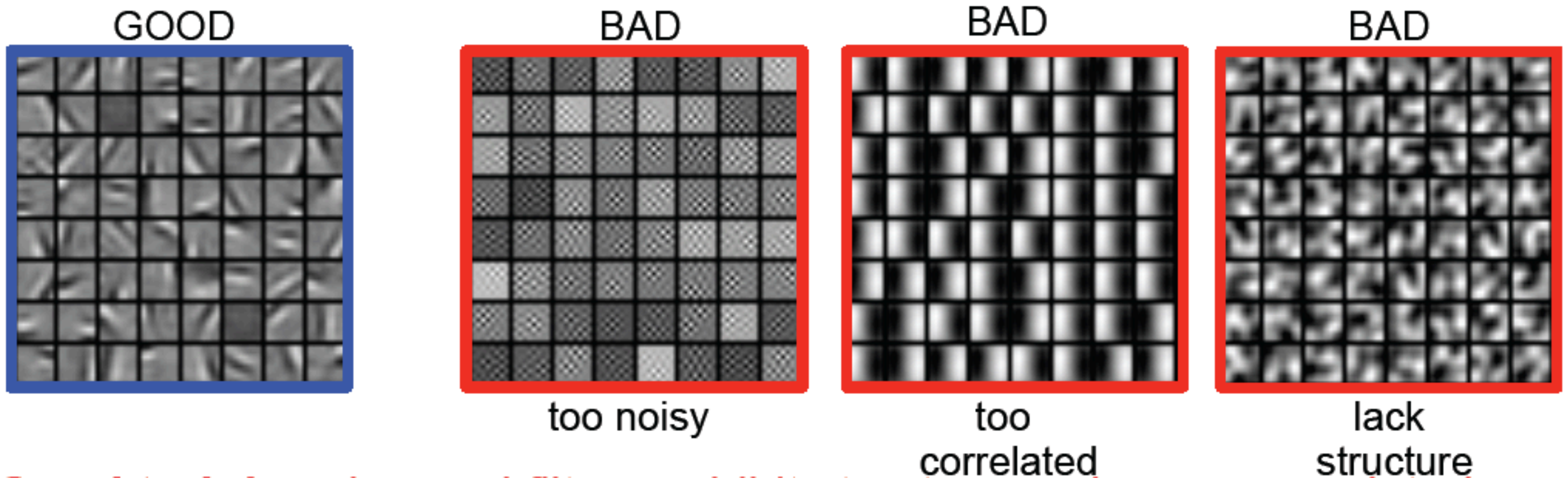
- Check gradients numerically by finite differences
- Visualize hidden activations — should be uncorrelated and high variance

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

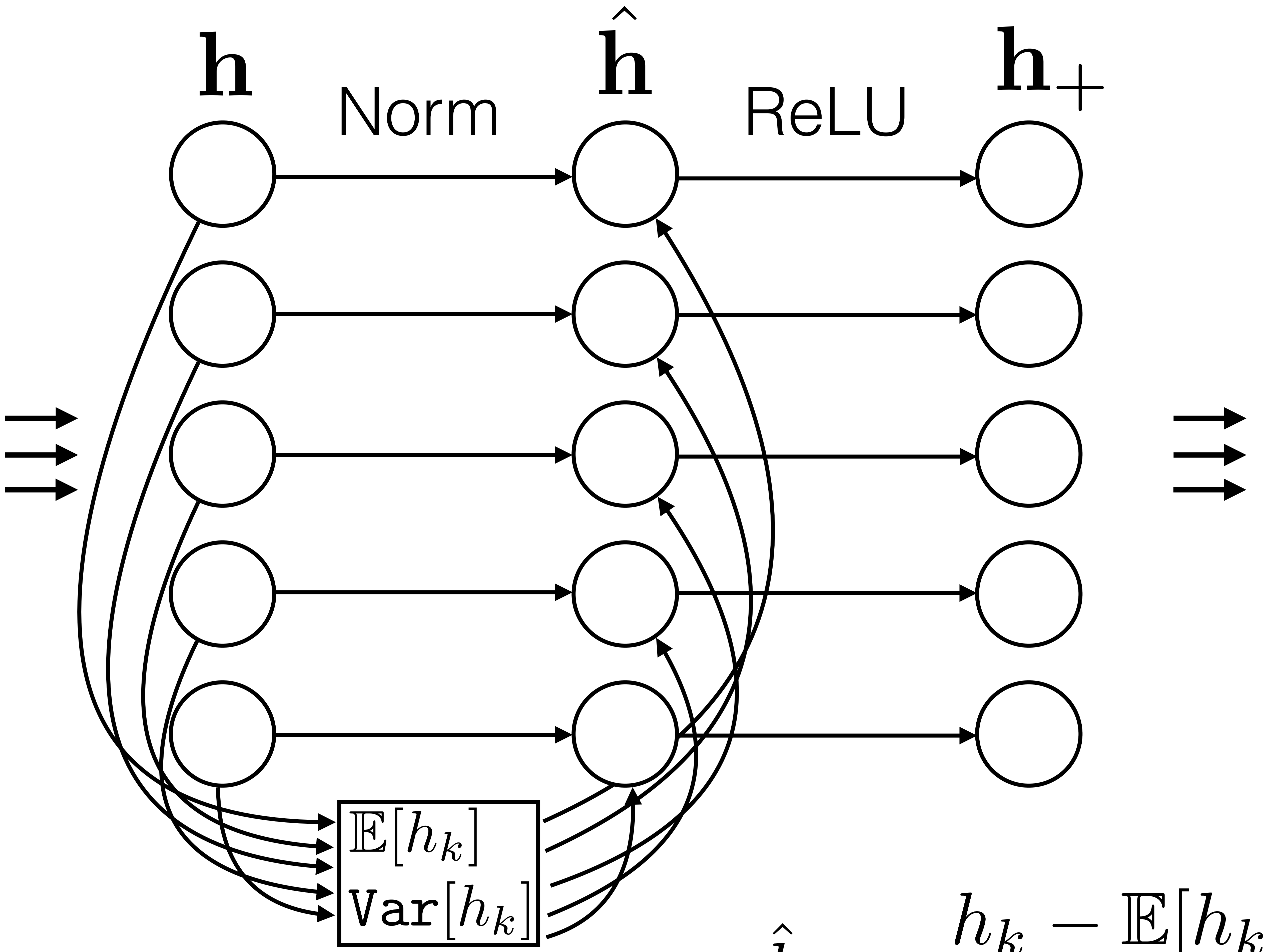
- Check gradients numerically by finite differences
- Visualize hidden activations — should be uncorrelated and high variance
- Visualize filters



Good training: learned filters exhibit structure and are uncorrelated.

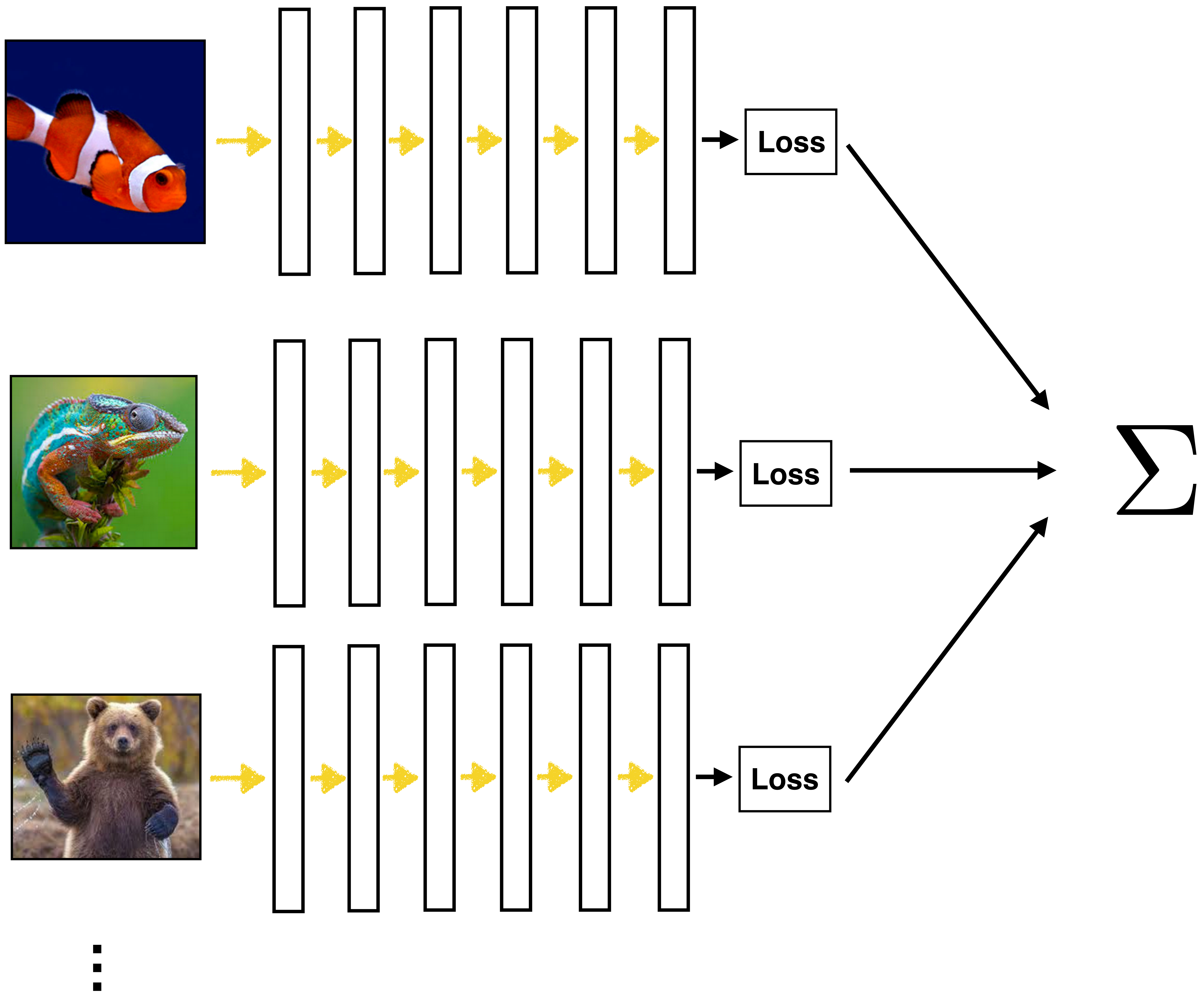
[Derived from slide by Marc'Aurelio Ranzato]

Normalization layers



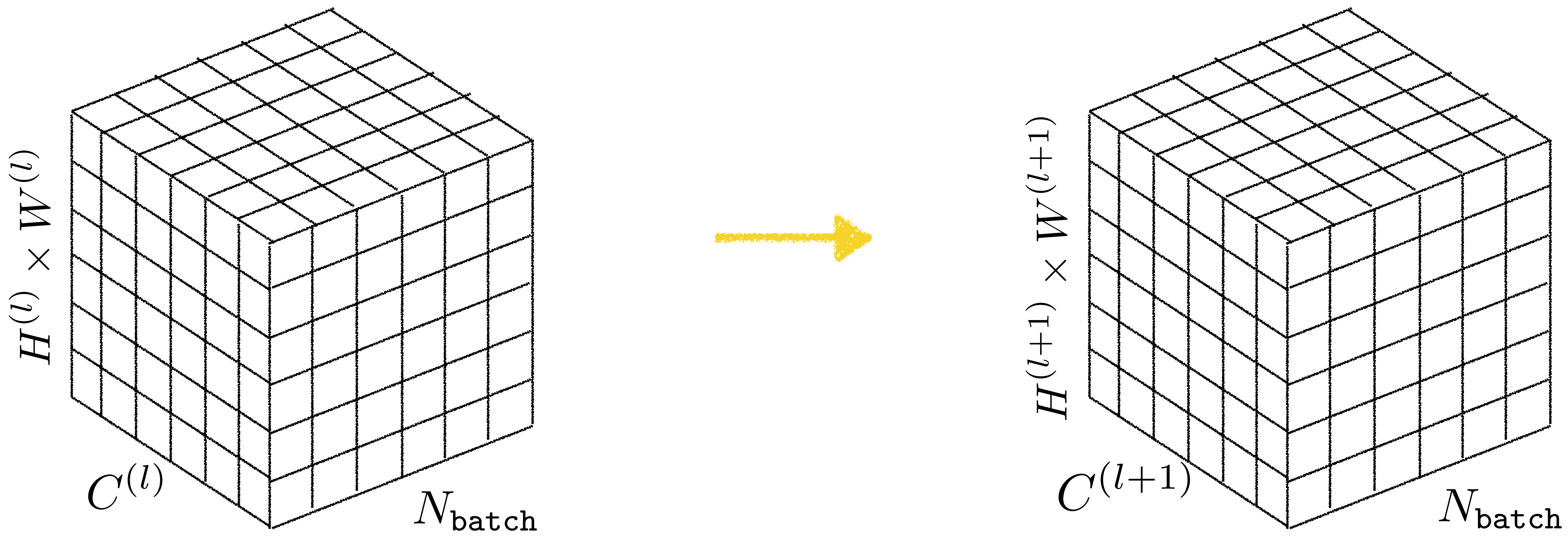
$$\hat{h}_k = \frac{h_k - \mathbb{E}[h_k]}{\sqrt{\texttt{Var}[h_k]}}$$

Batch processing

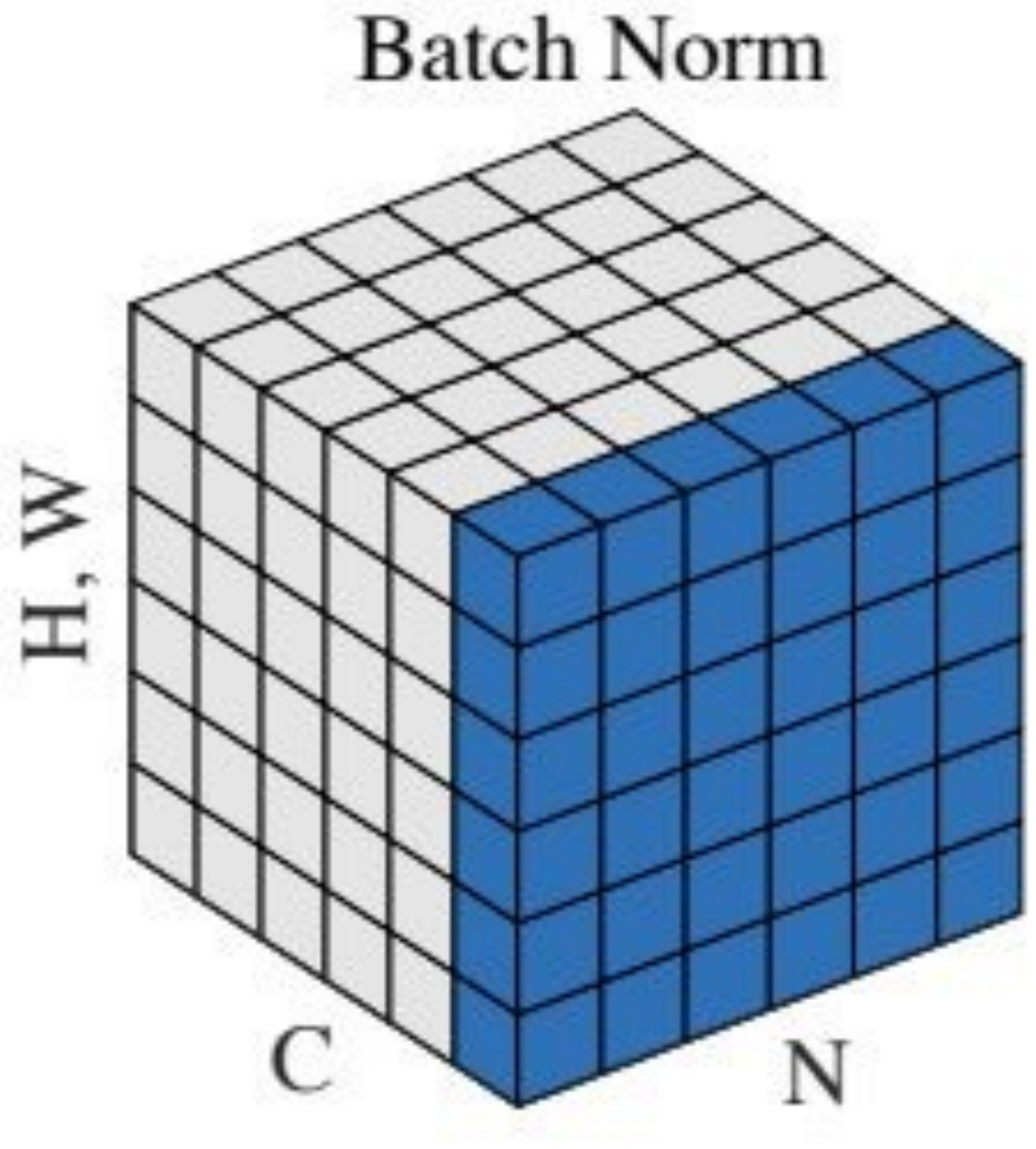


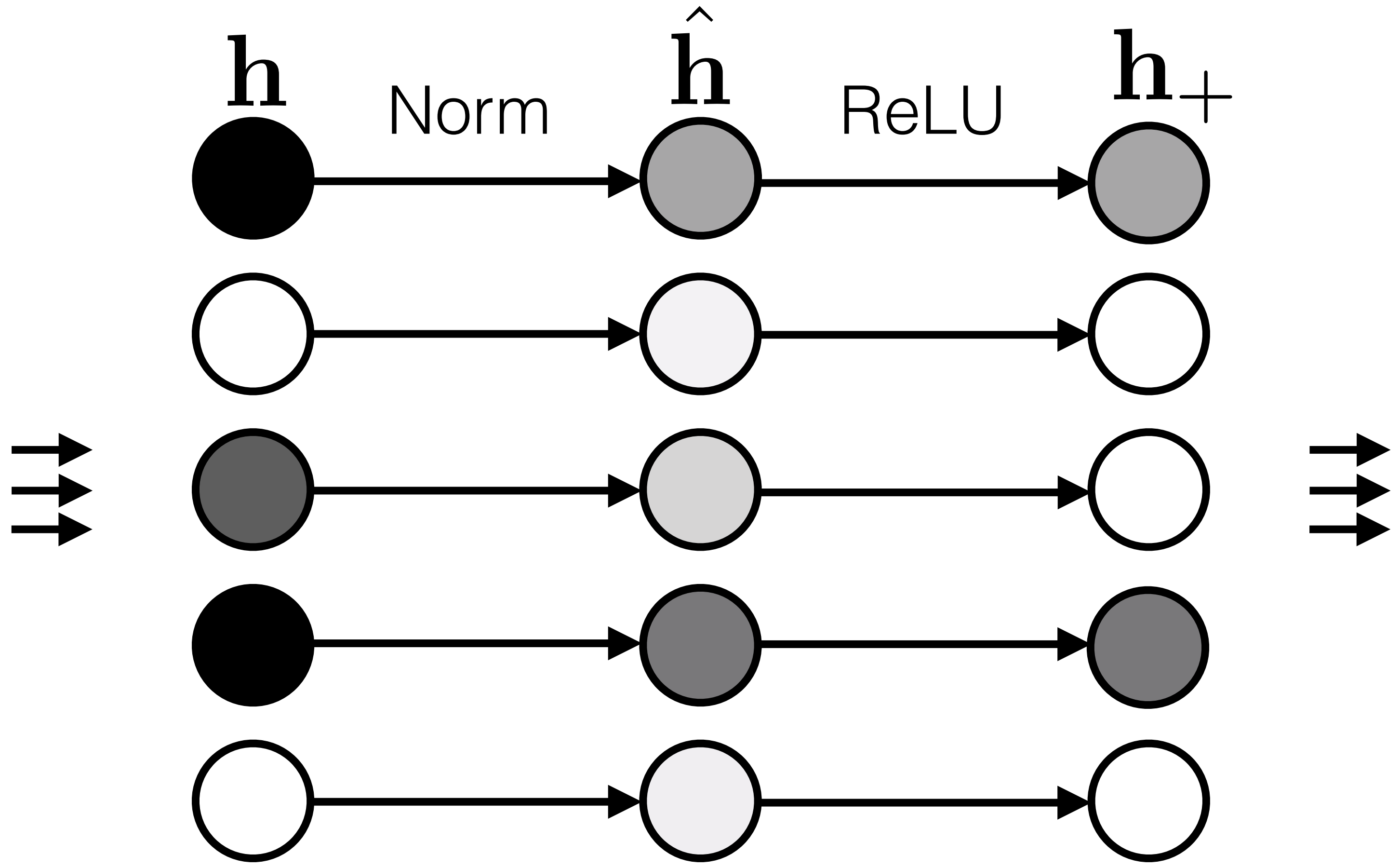
“Tensor flow”

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



Normalization layers

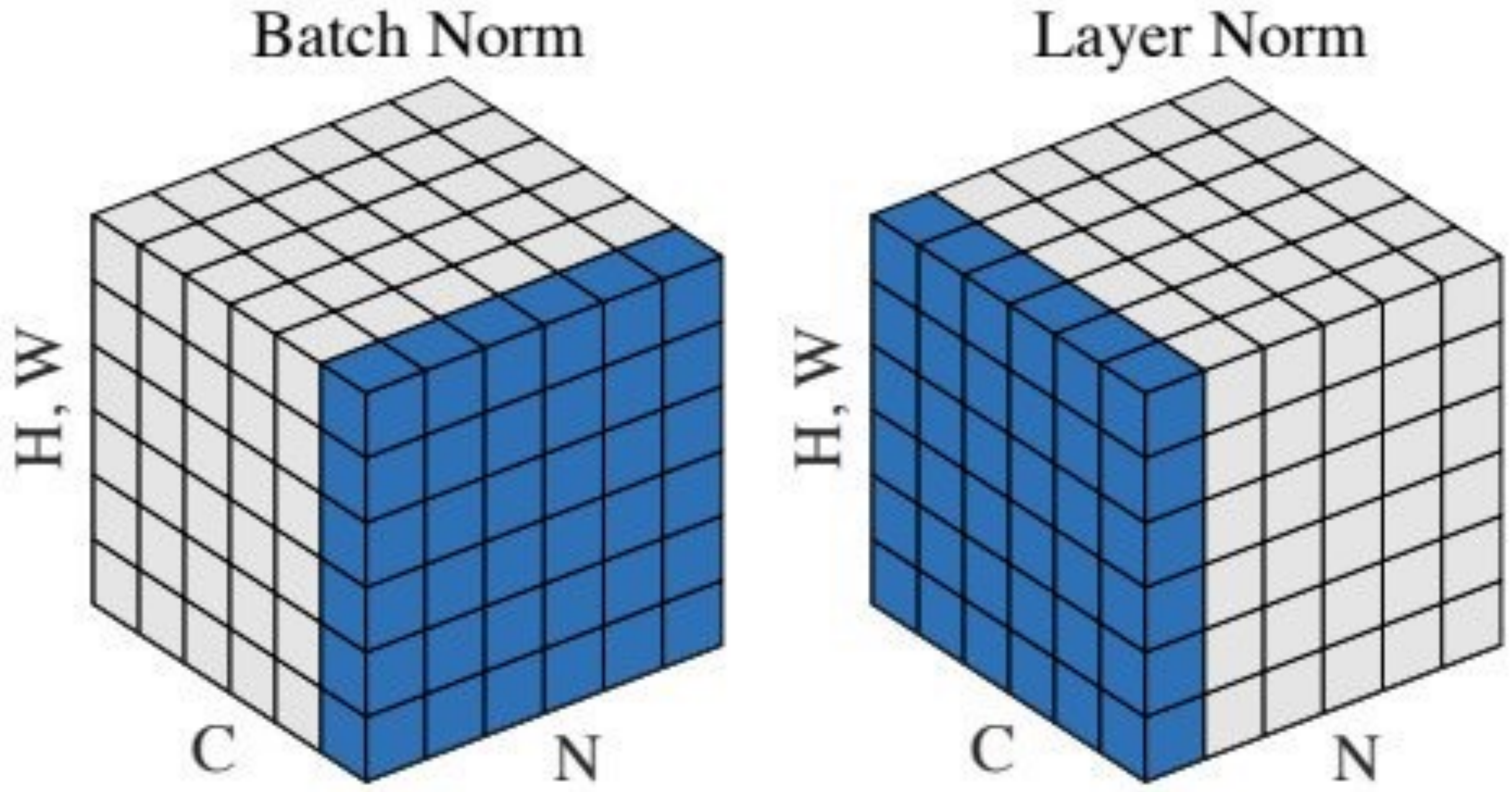




Normalize w.r.t. a single hidden unit's pattern of activation over training examples (a batch of examples).

[Figure from Wu & He, arXiv 2018]

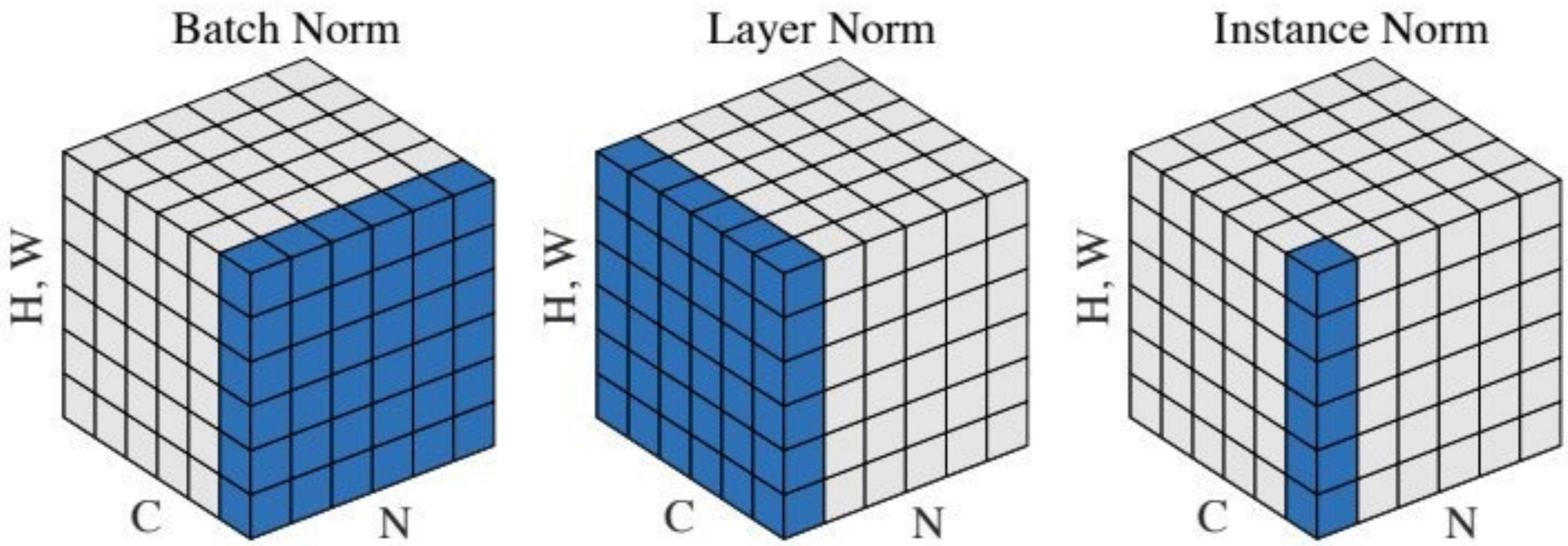
Normalization layers



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

[Figure from Wu & He, arXiv 2018]

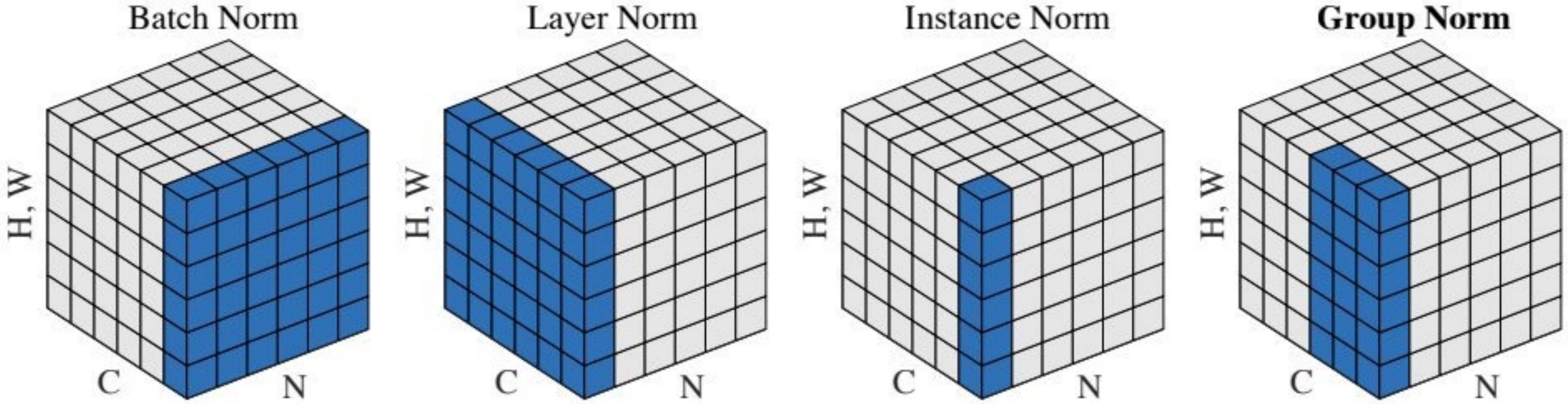
Normalization layers



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

[Figure from Wu & He, arXiv 2018]

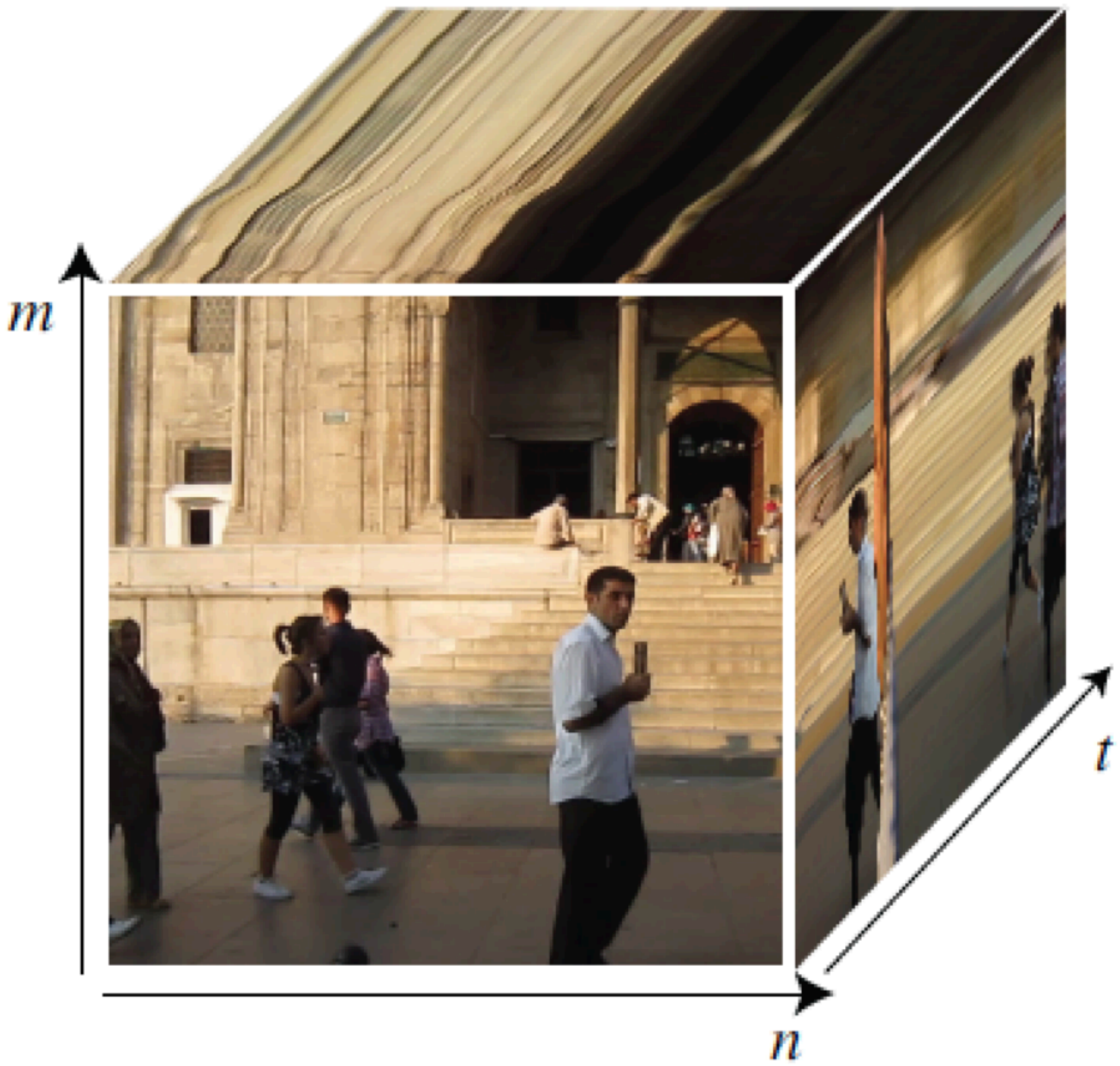
Normalization layers

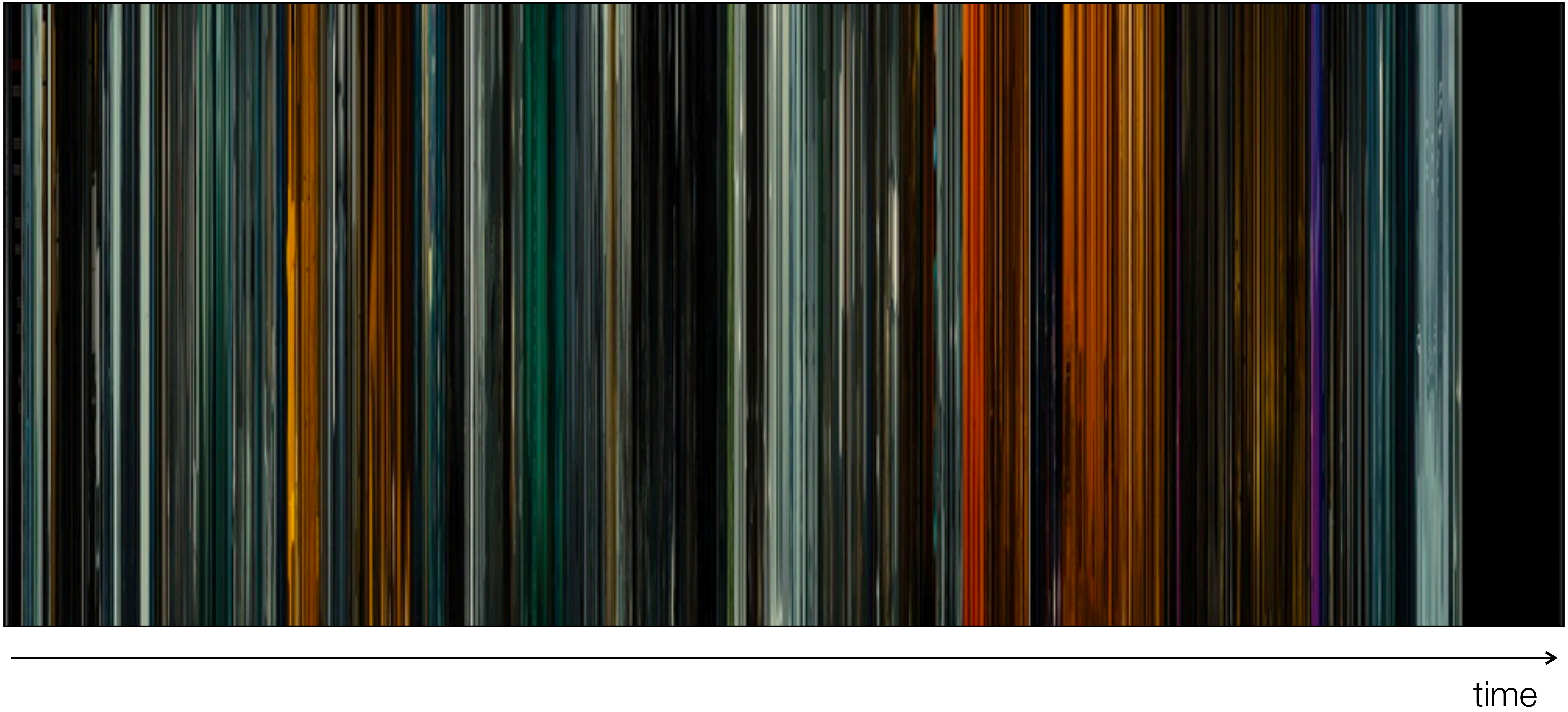


Might as well…

[Figure from Wu & He, arXiv 2018]

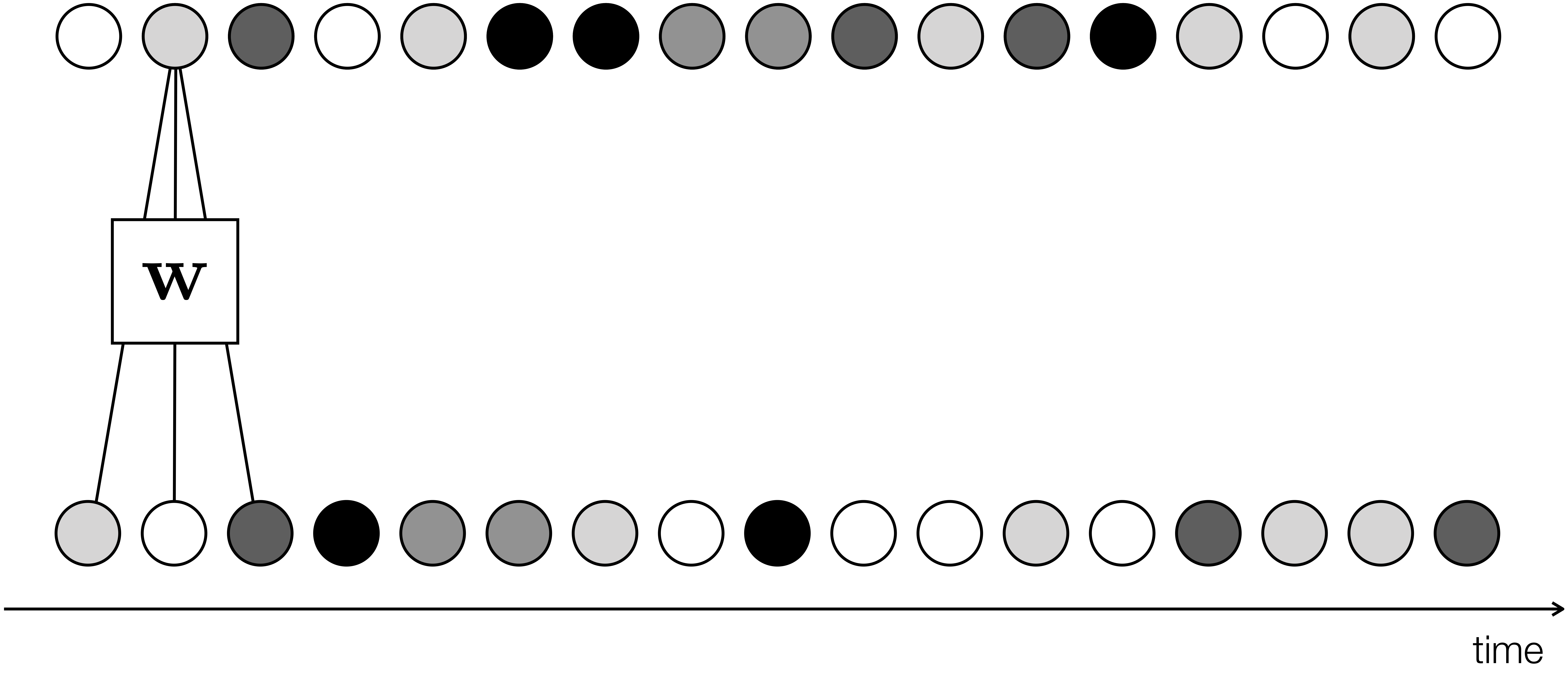
Sequences

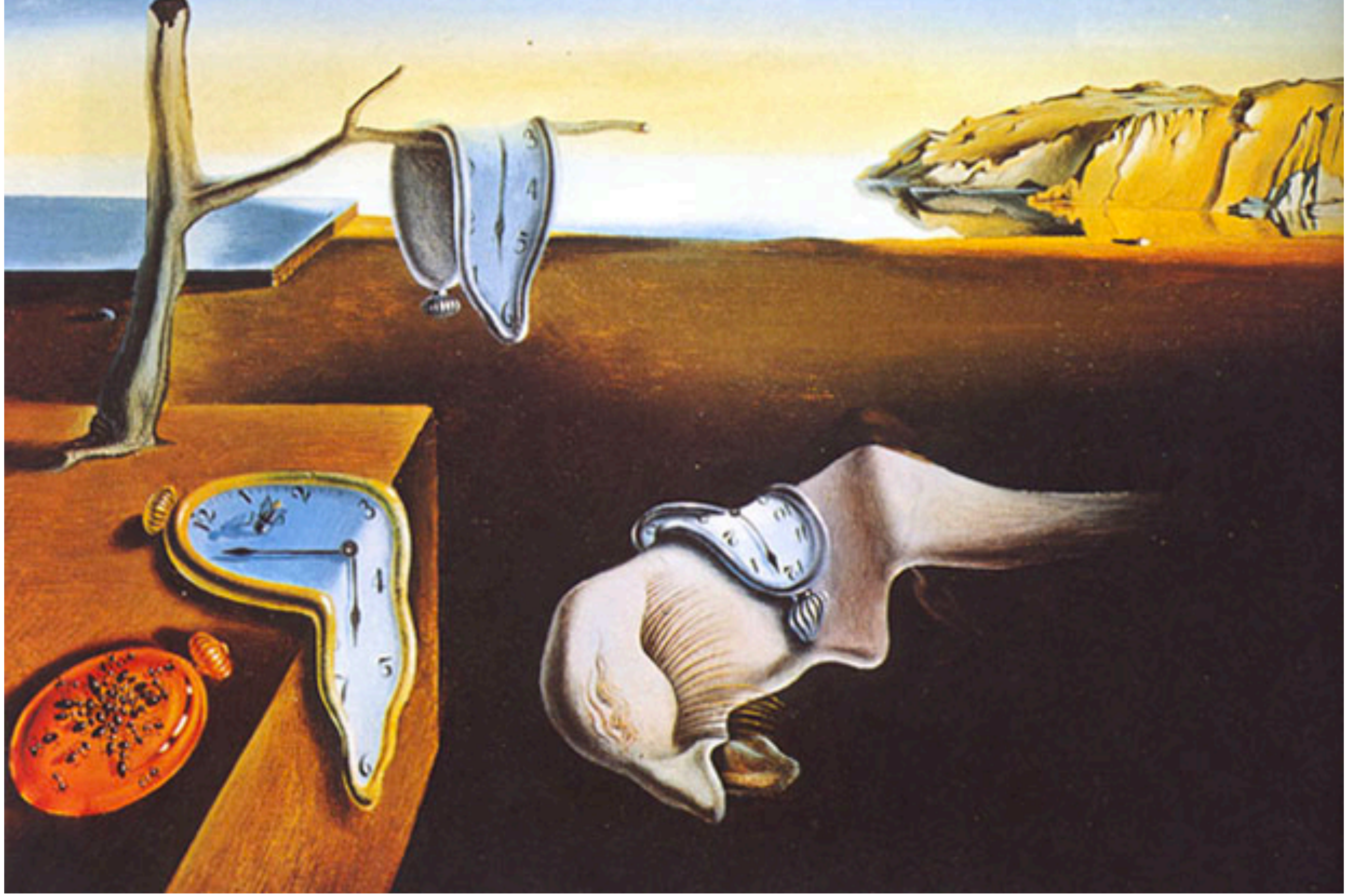




[http://moviebarcode.tumblr.com/]

Convolutions in time

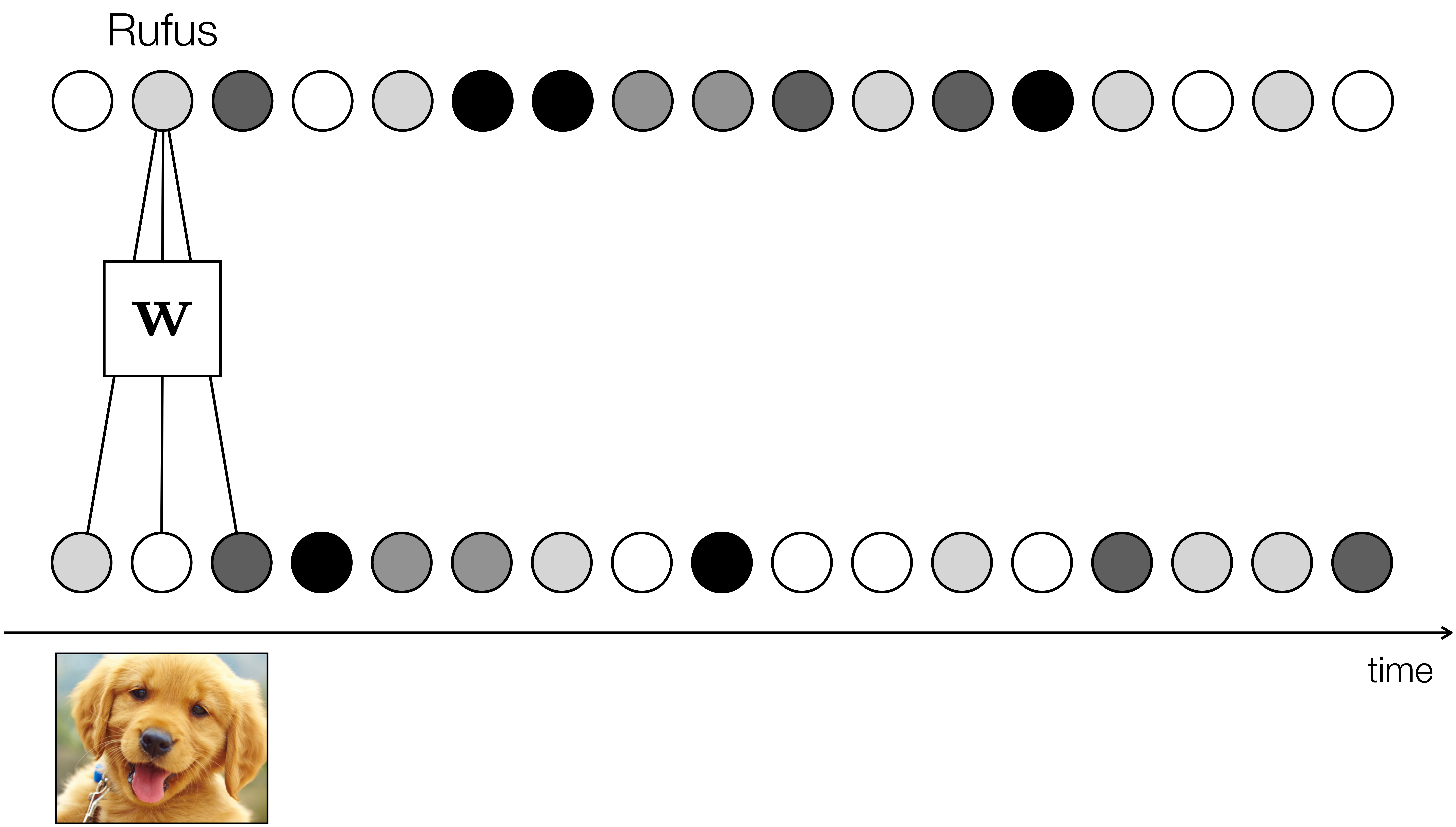




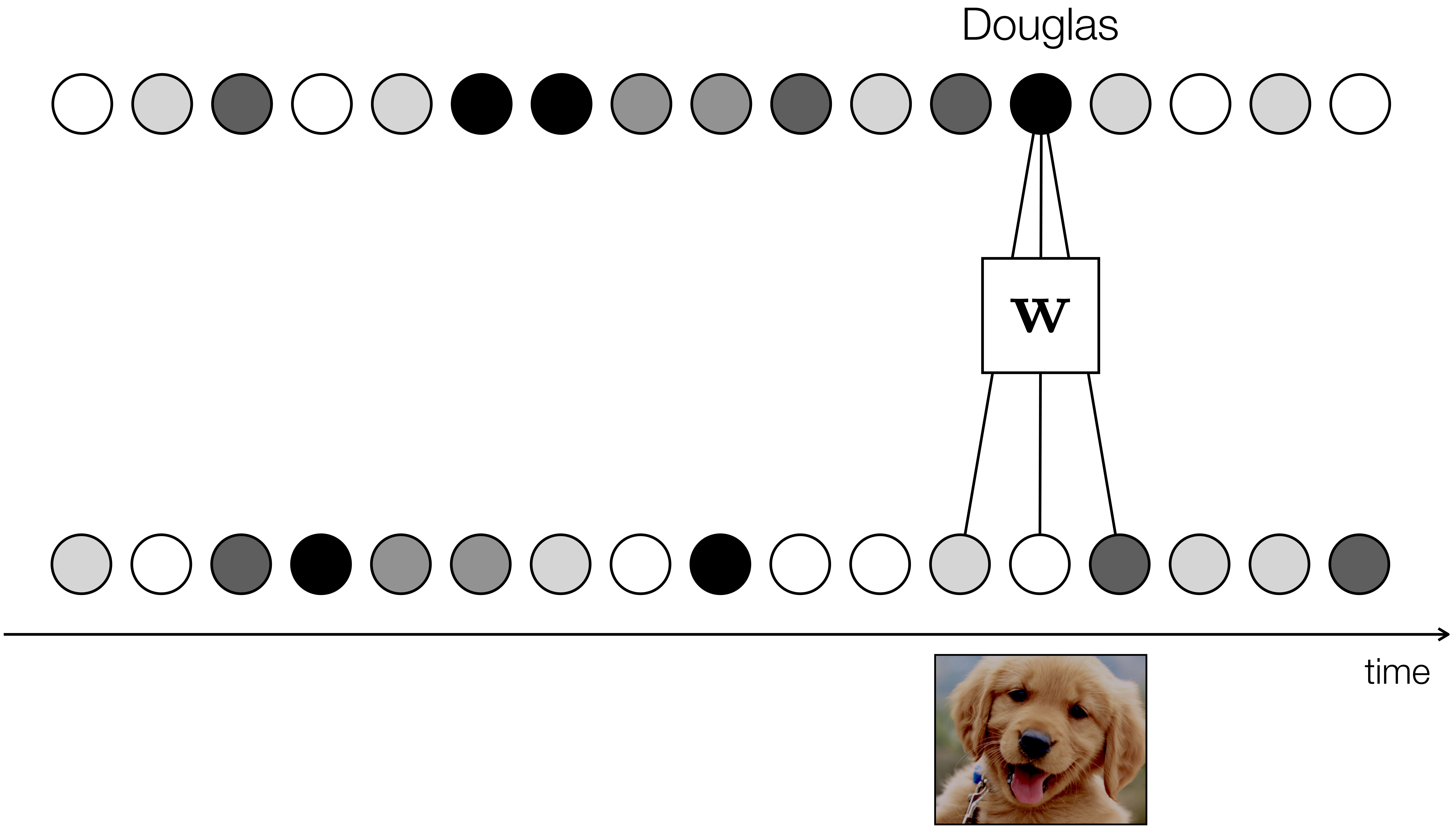
“The Persistence of Memory”,
Dali 1931

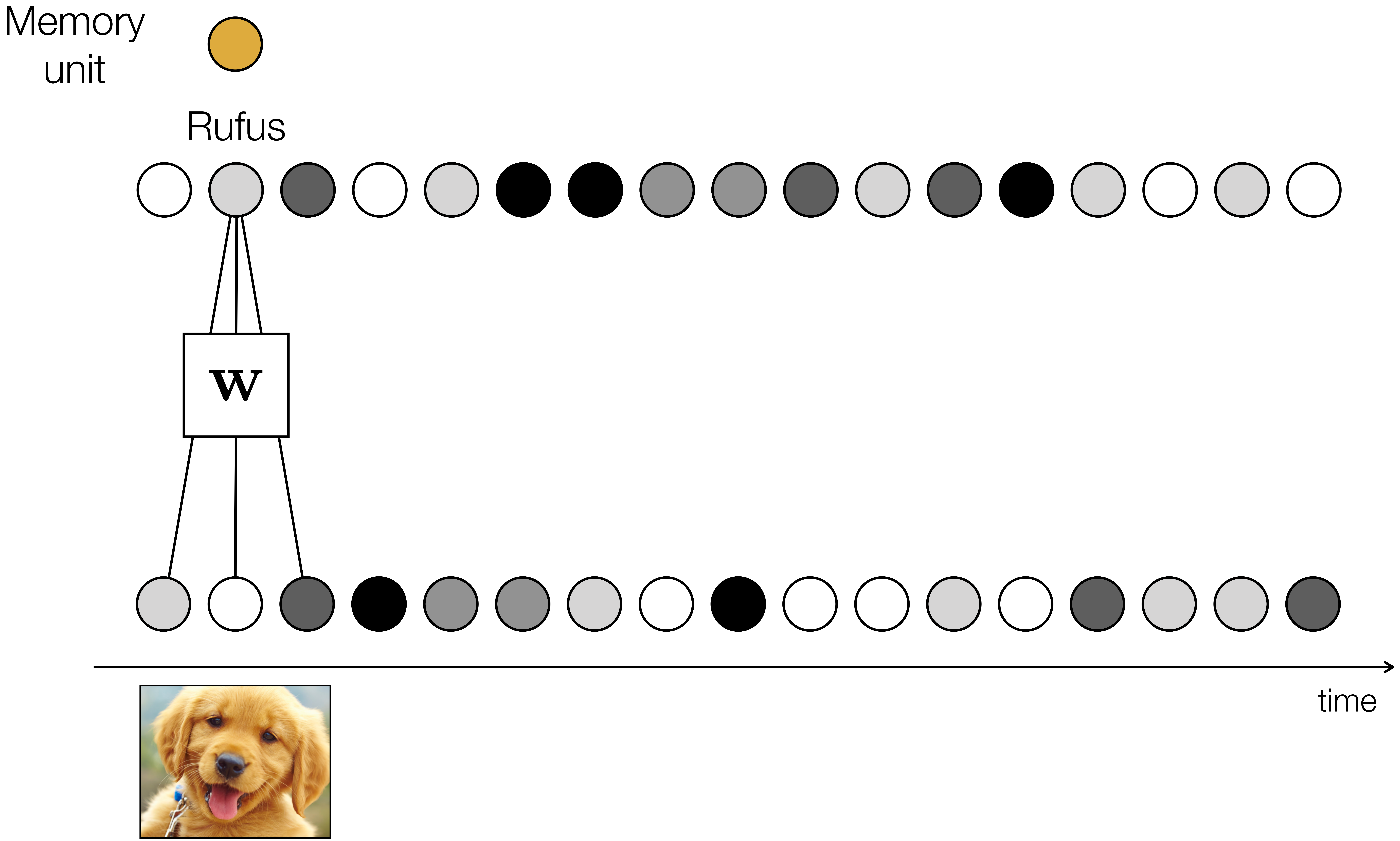
It bothered him that the dog at three fourteen (seen from the side) should have the same name as the dog at three fifteen (seen from the front).
— “Funes the Memorius”, Borges 1962

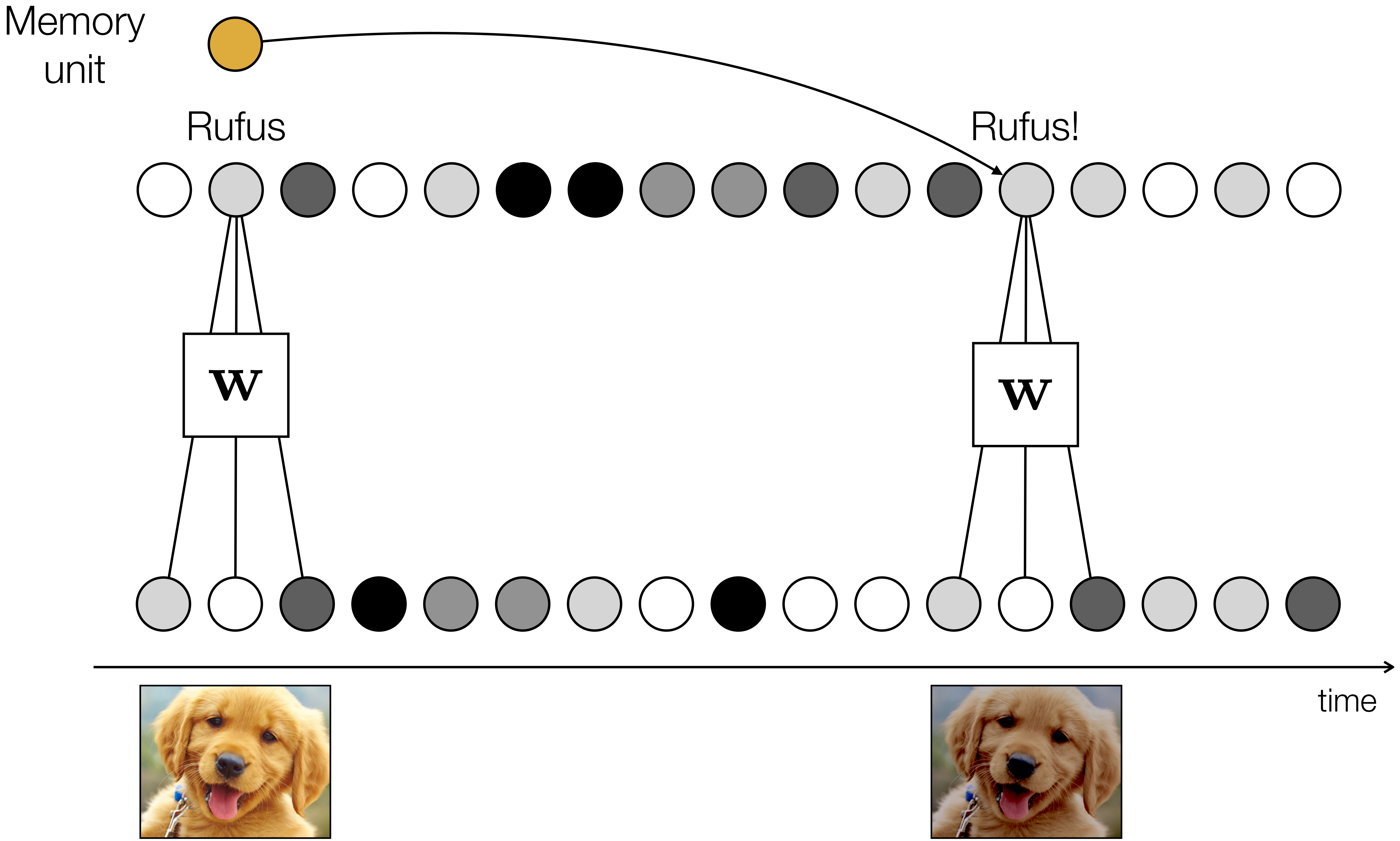
[https://www.youtube.com/watch?v=wxfGT-kKxiM]



Rufus
$\mathbf{w}$
time

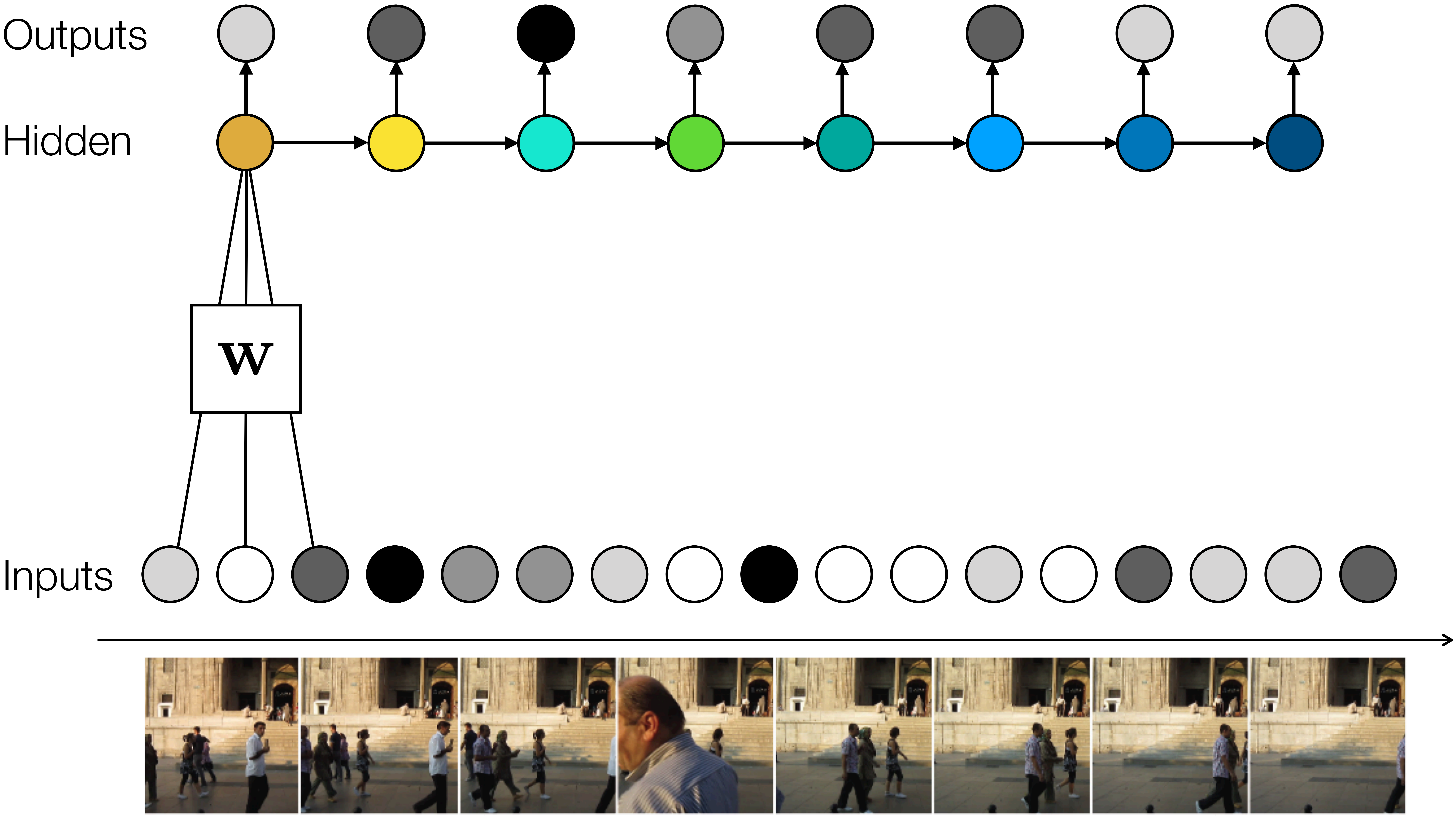

Douglas
$\mathbf{w}$
time


Memory unit
Rufus
w
time

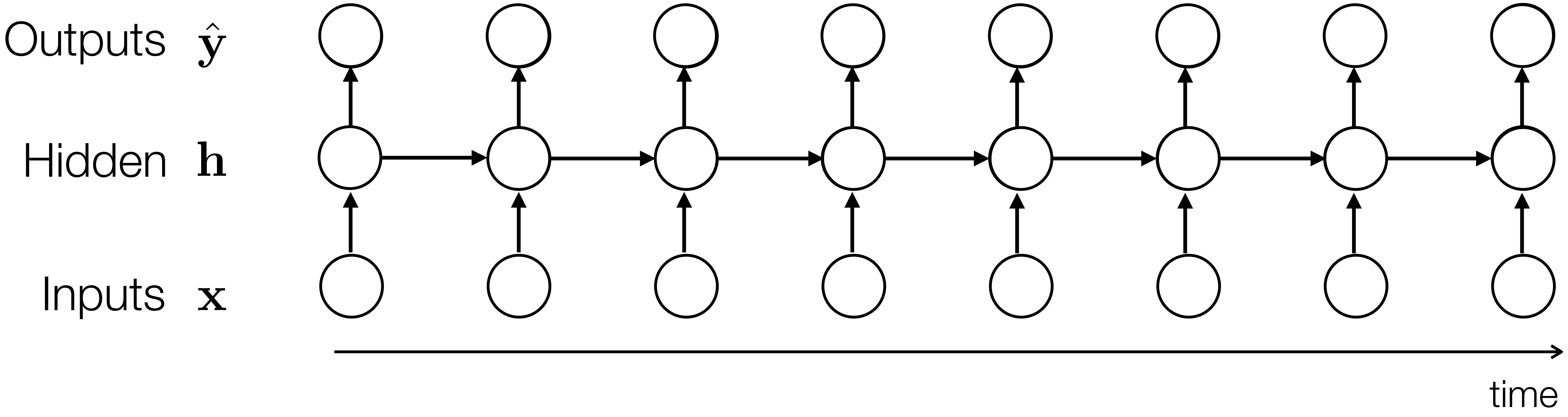


Memory unit
Rufus
Rufus!
W
W
time

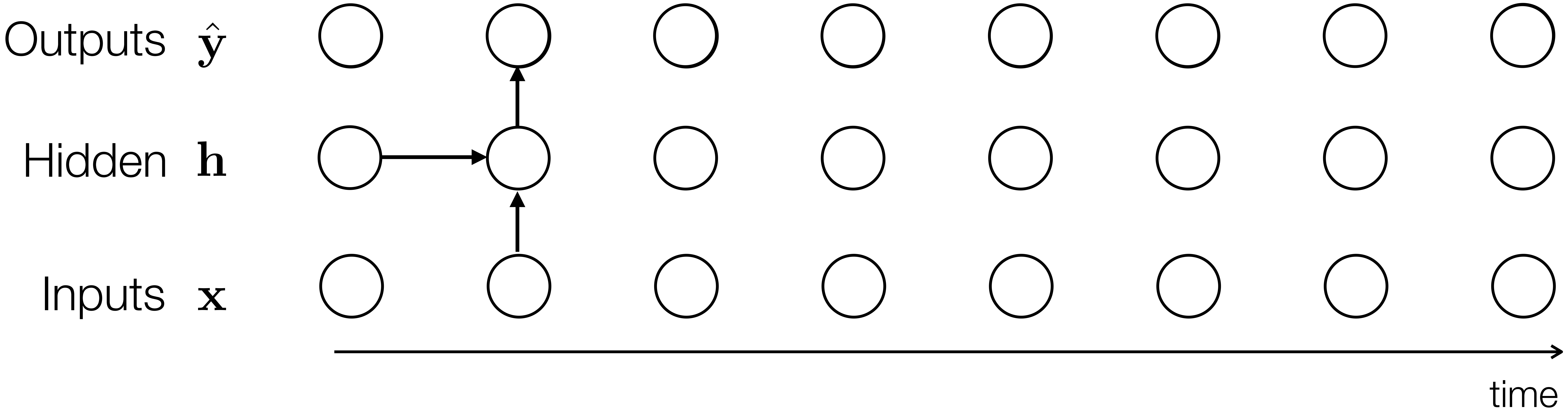
Recurrent Neural Networks (RNNs)



Recurrent Neural Networks (RNNs)



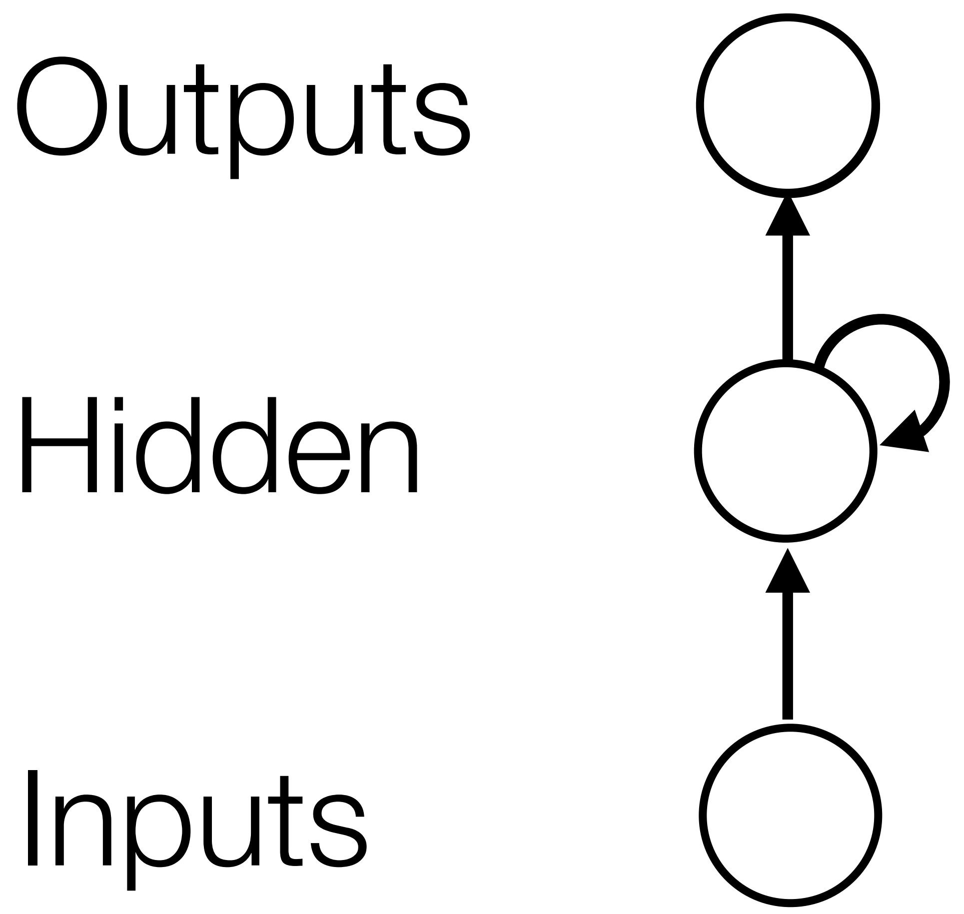
Recurrent Neural Networks (RNNs)



$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)})$$

$$\mathbf{y}^{(t)} = g(\mathbf{h}^{(t)})$$

Recurrent Neural Networks (RNNs)

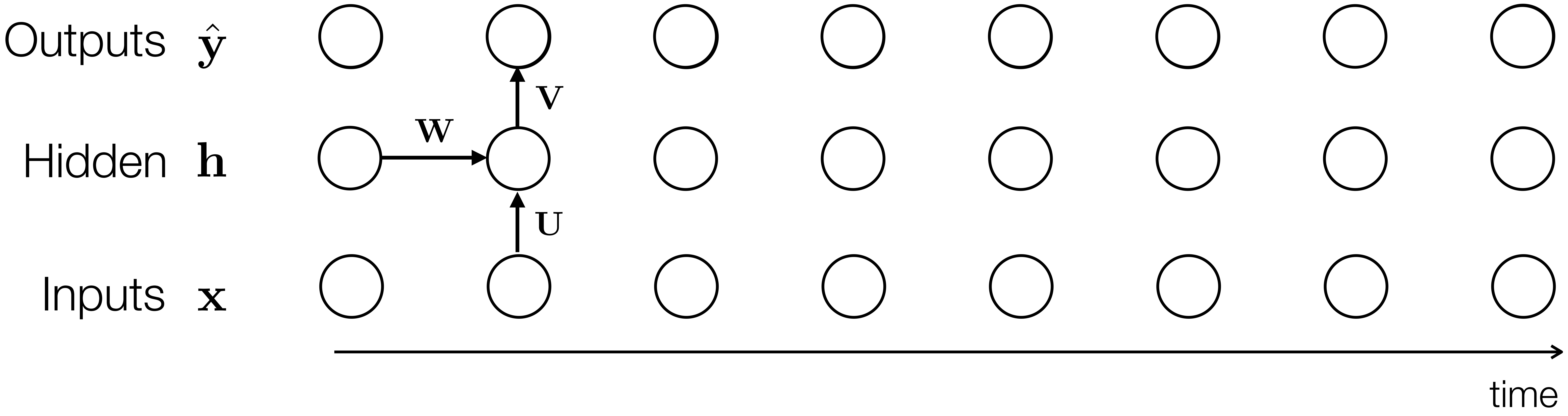


Recurrent!

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)})$$

$$\mathbf{y}^{(t)} = g(\mathbf{h}^{(t)})$$

Recurrent Neural Networks (RNNs)



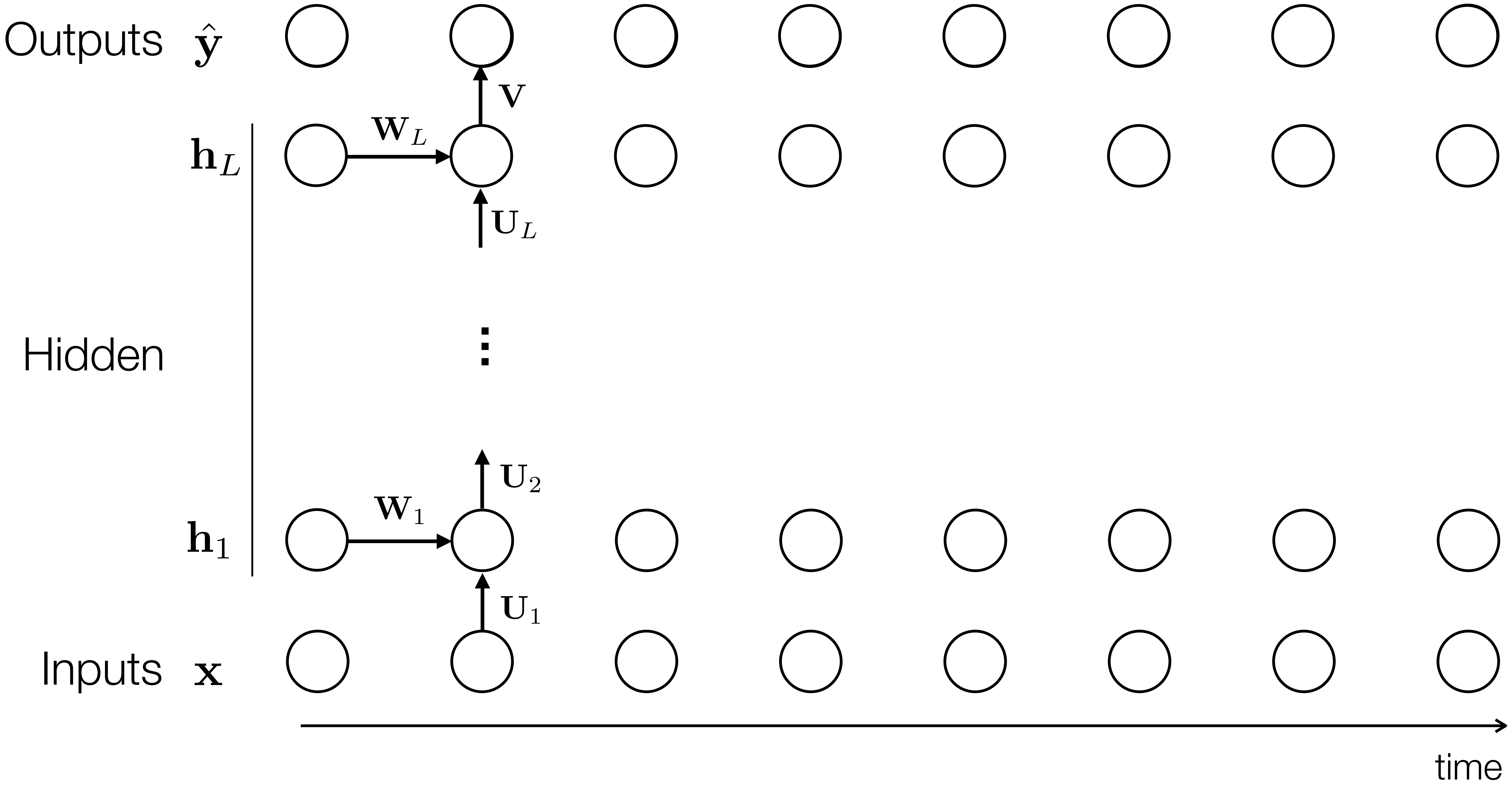
$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$

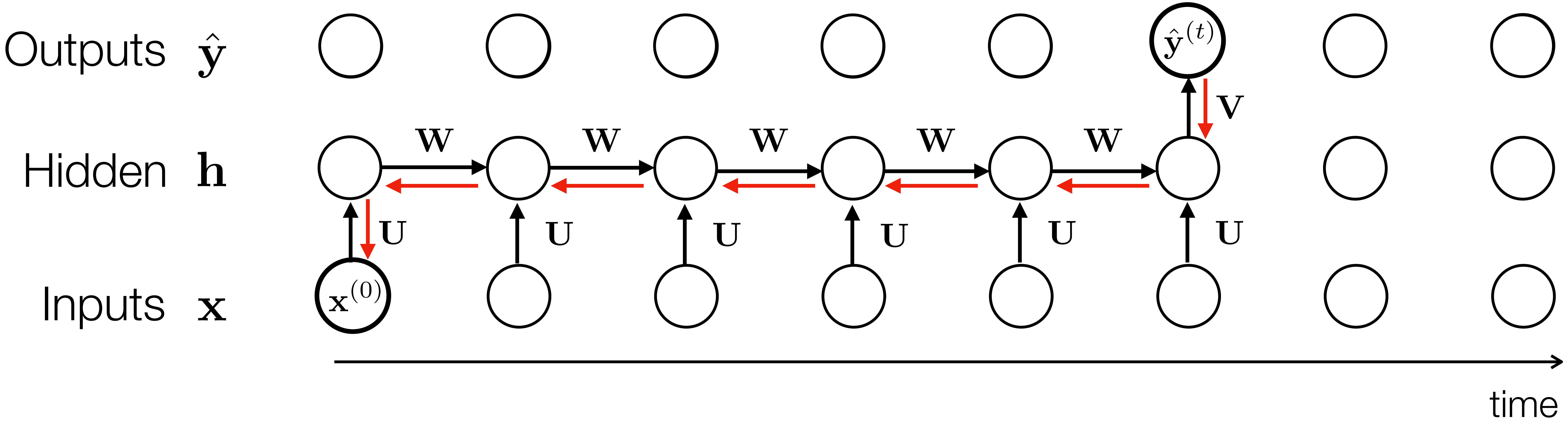
$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)} + \mathbf{c}$$

$$\hat{\mathbf{y}}^{(t)} = \texttt{softmax}(\mathbf{o}^{(t)})$$

Deep Recurrent Neural Networks (RNNs)

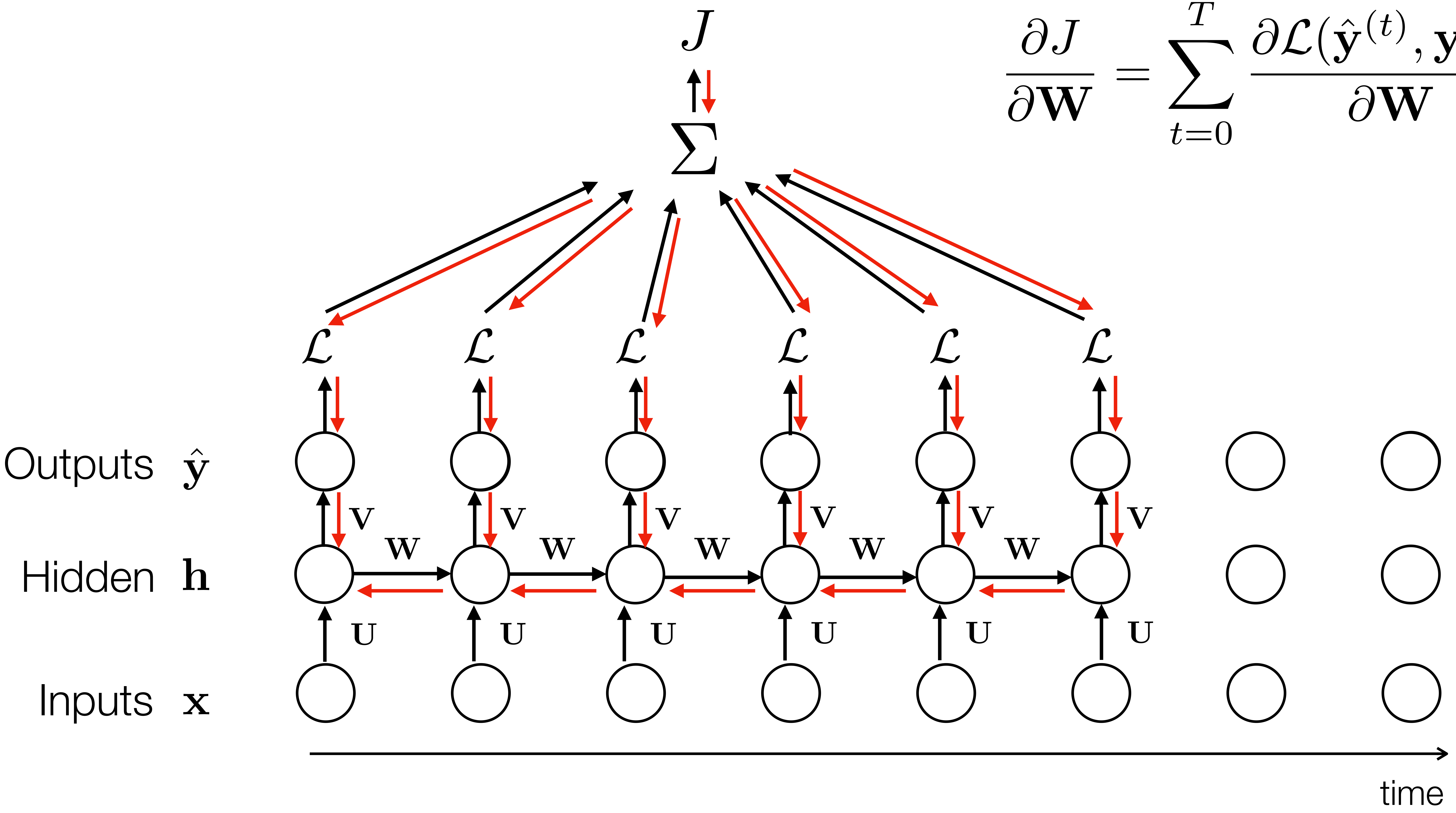


Backprop through time



$$\frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{x}^{(0)}} = \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \cdots \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{h}^{(0)}} \frac{\partial \mathbf{h}^{(0)}}{\partial \mathbf{x}^{(0)}}$$

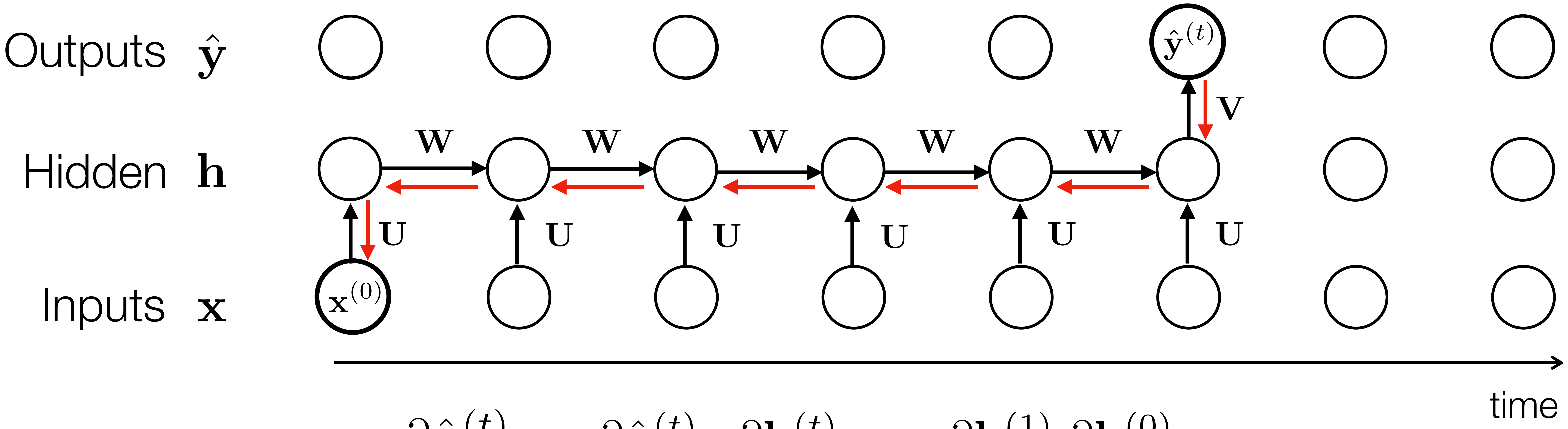
$$\frac{\partial J}{\partial \mathbf{W}} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}(\hat{\mathbf{y}}^{(t)}, \mathbf{y}^{(t)})}{\partial \mathbf{W}}$$



Recurrent linear layer

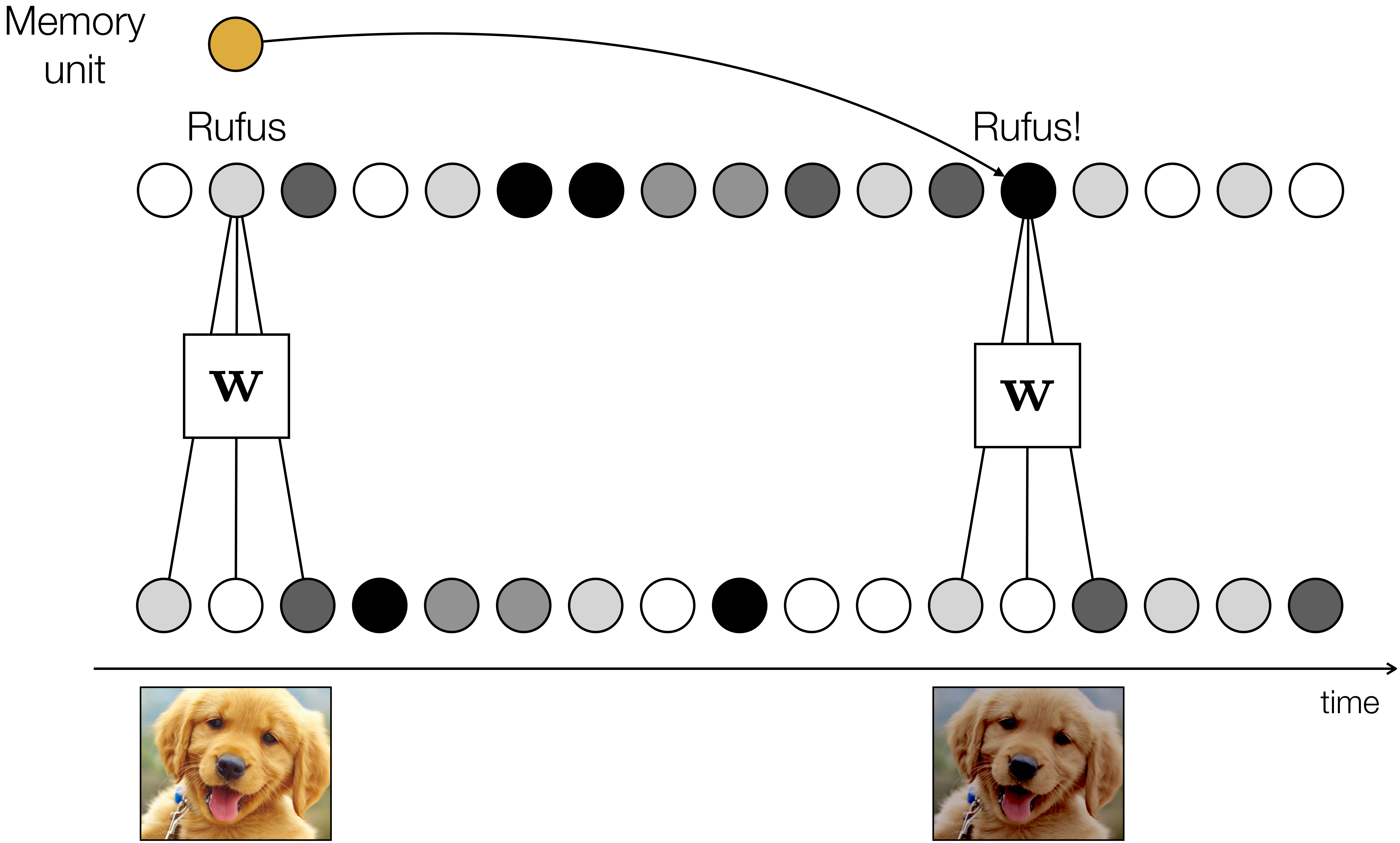
$$\frac{\partial J}{\partial \mathbf{W}} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}(\hat{\mathbf{y}}^{(t)}, \mathbf{y}^{(t)})}{\partial \mathbf{W}}$$

The problem of long-range dependences

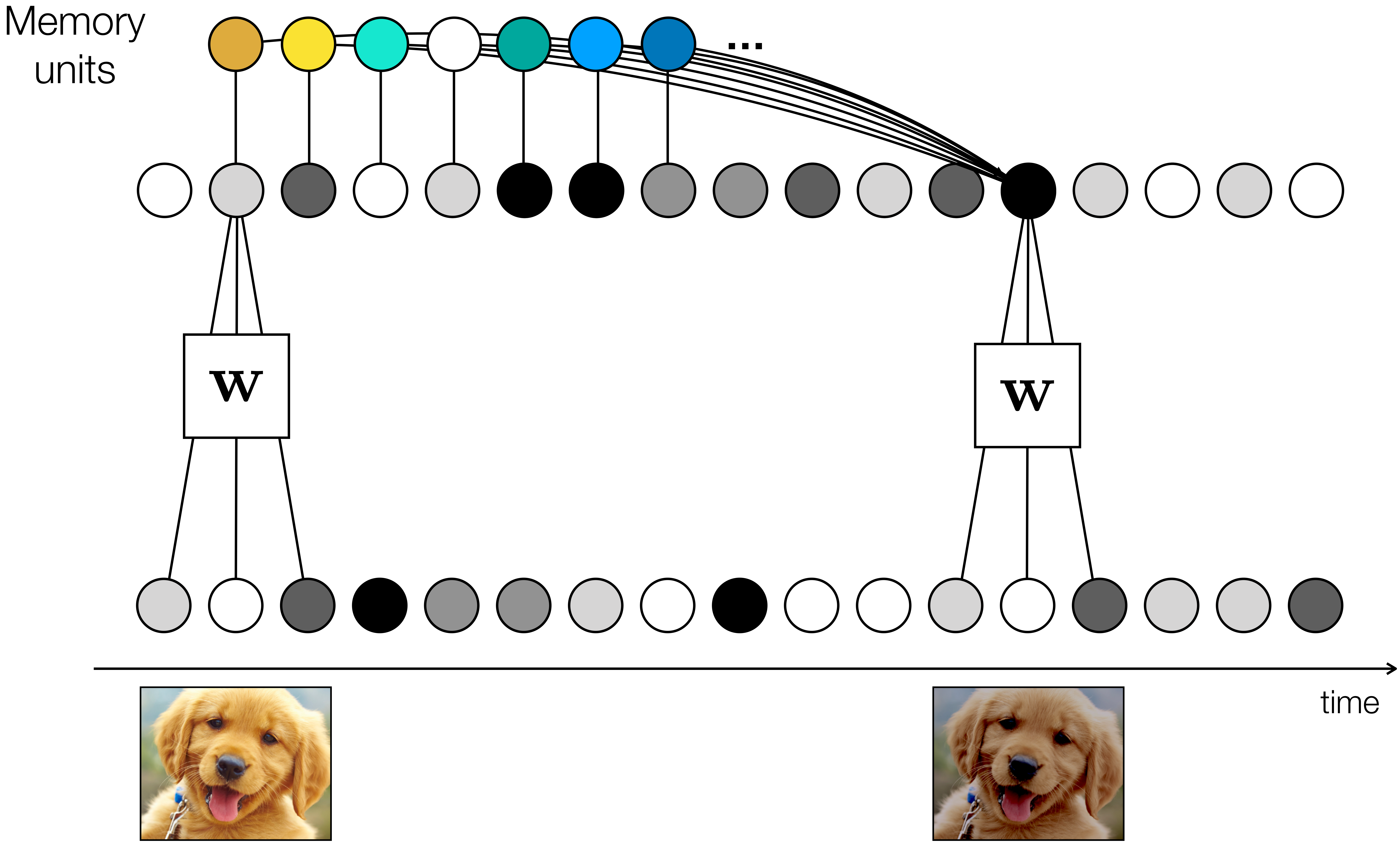


$$\frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{x}^{(0)}} = \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \cdots \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{h}^{(0)}} \frac{\partial \mathbf{h}^{(0)}}{\partial \mathbf{x}^{(0)}}$$

- Capturing long-range dependences requires propagating information through a long chain of dependences.
- Old observations are forgotten
- Stochastic gradients become high variance (noisy), and gradients may **vanish** or **explode**



Memory unit
Rufus
Rufus!
w
w
time


Memory
units
...
w
w
time

The problem of long-range dependences

Why not remember everything?

- Memory size grows with t
- This kind of memory is **nonparametric**: there is no finite set of parameters we can use to model it
- RNNs make a Markov assumption — the future hidden state only depends on the immediately preceding hidden state
- By putting the right info in to the hidden state, RNNs can model depedences that are arbitrarily far apart

The problem of long-range dependences

Other methods exist that do directly link old "memories" (observations or hidden states) to future predictions:

- Temporal convolutions
- Attention (see https://arxiv.org/abs/1706.03762)
- Memory networks (see https://arxiv.org/abs/1410.3916)

LSTMs

Long Short Term Memory

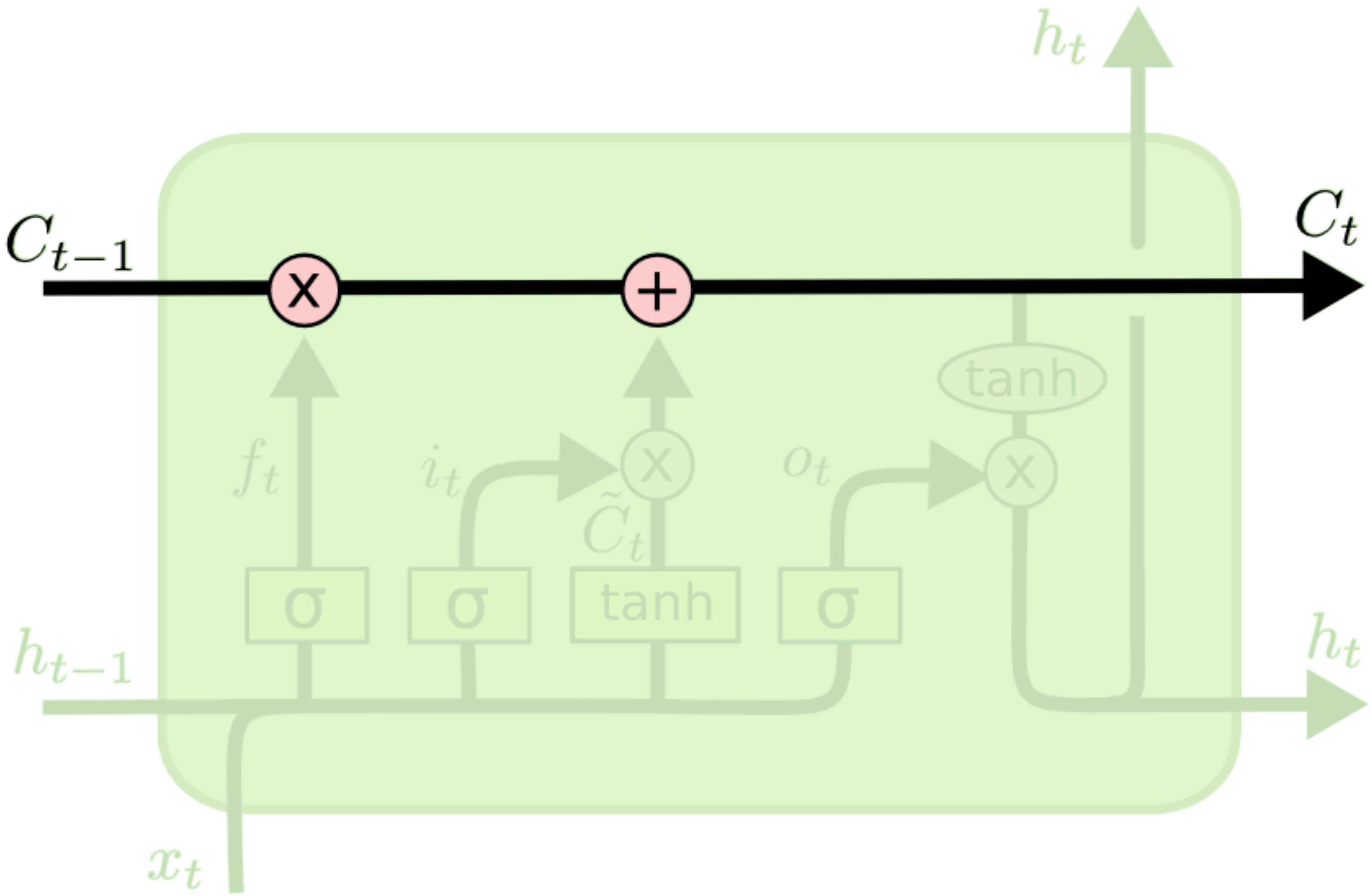
A special kind of RNN designed to avoid forgetting.

Related to resnets: inductive bias is that state transition is an identity function.

This way the default behavior is not to forget an old state. Instead of forgetting by default, the network has to *learn to forget*.

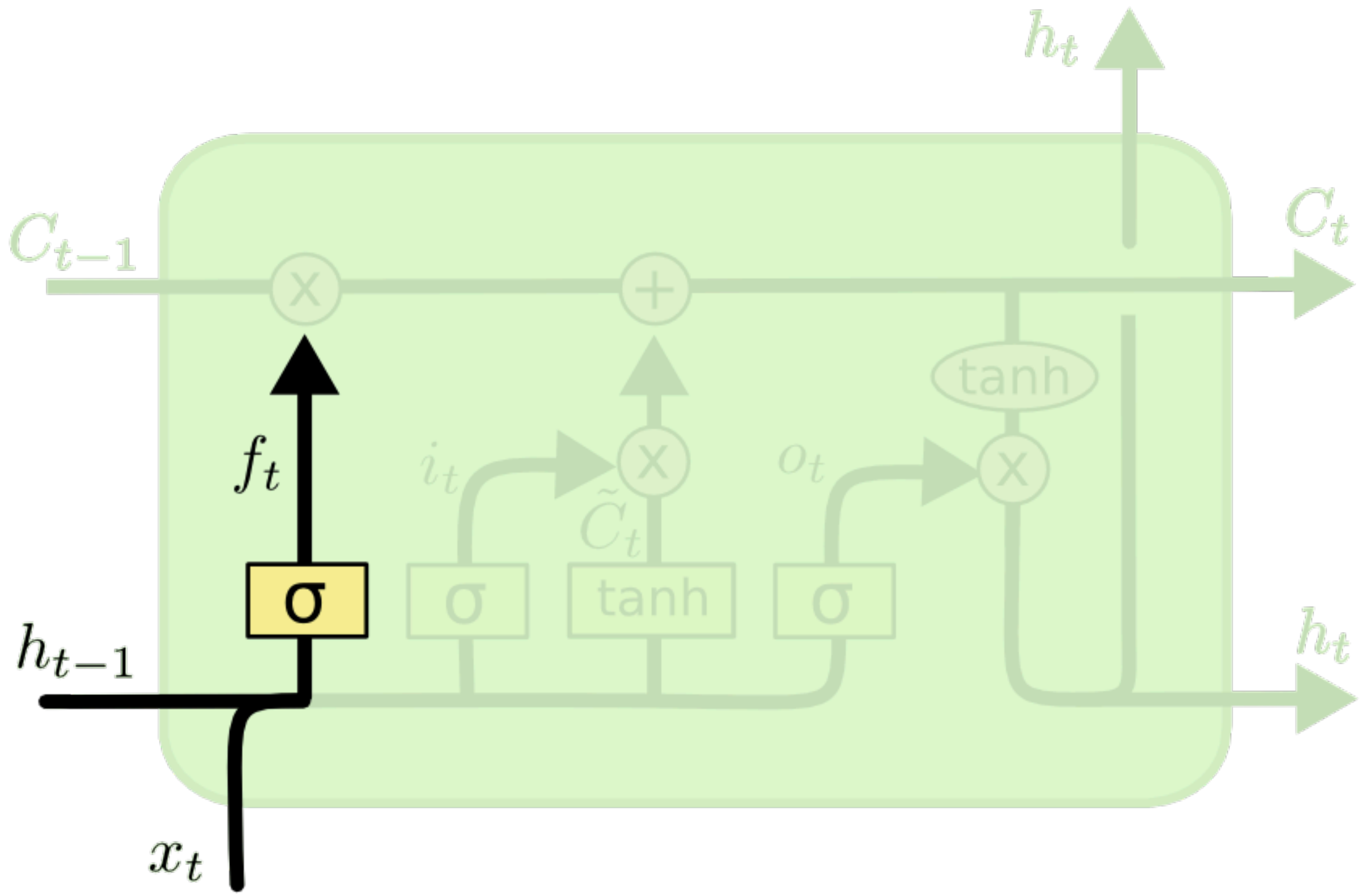
[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]

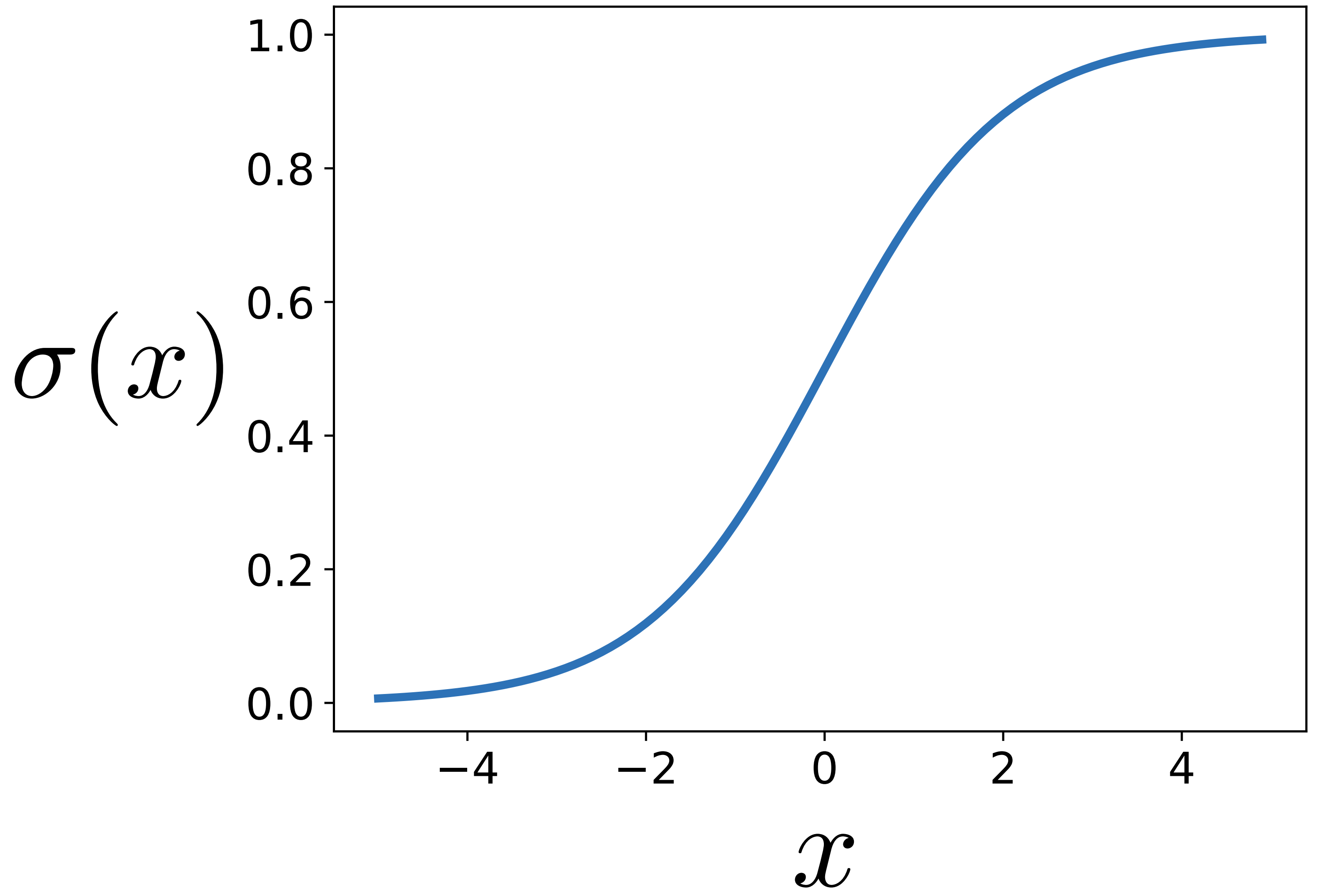
[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]



C_t = Cell state

[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]



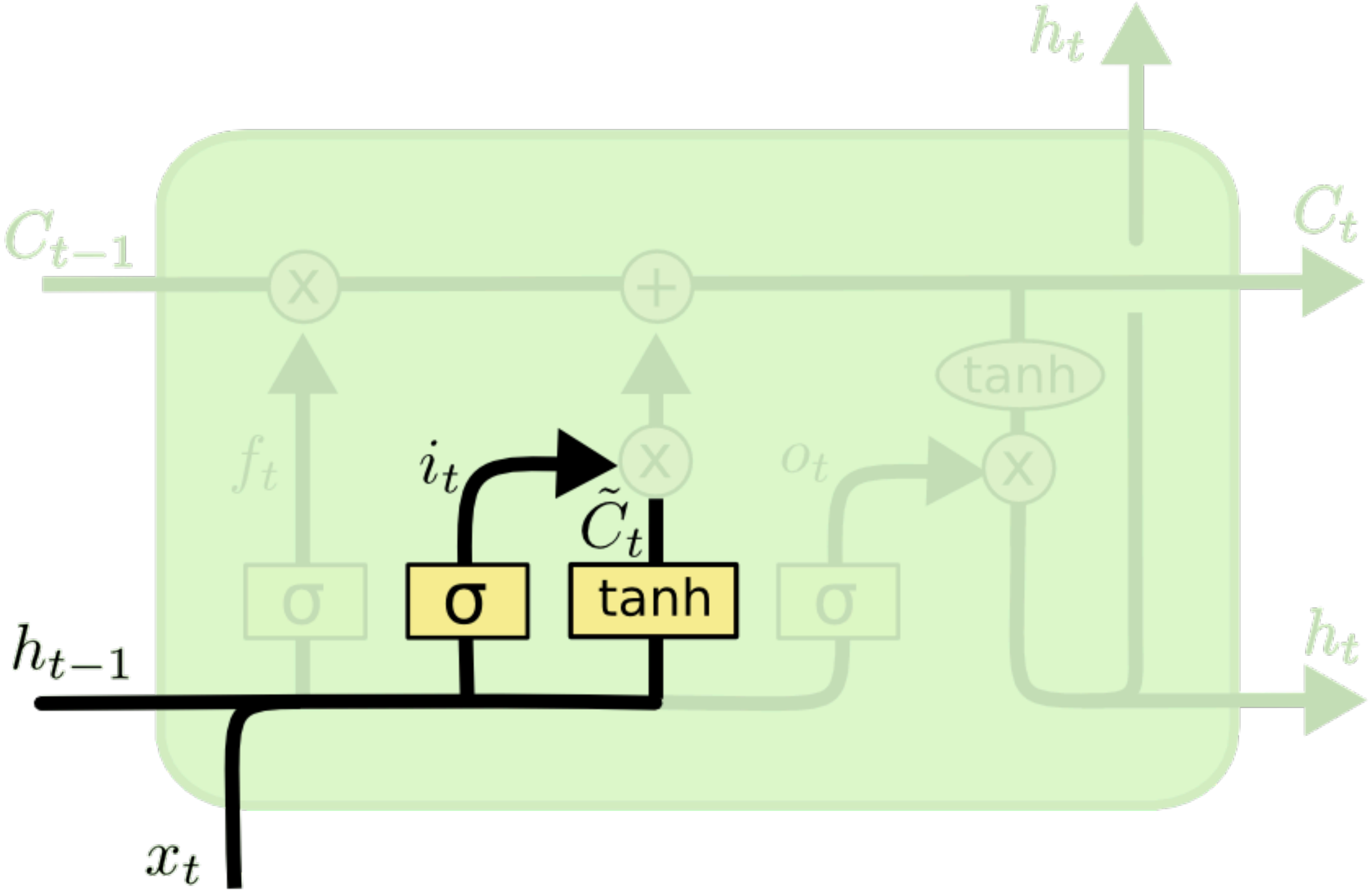


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Decide what information to throw away from the cell state.

Each element of cell state is multiplied by ~1 (remember) or ~0 (forget).

[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]



which indices to write to

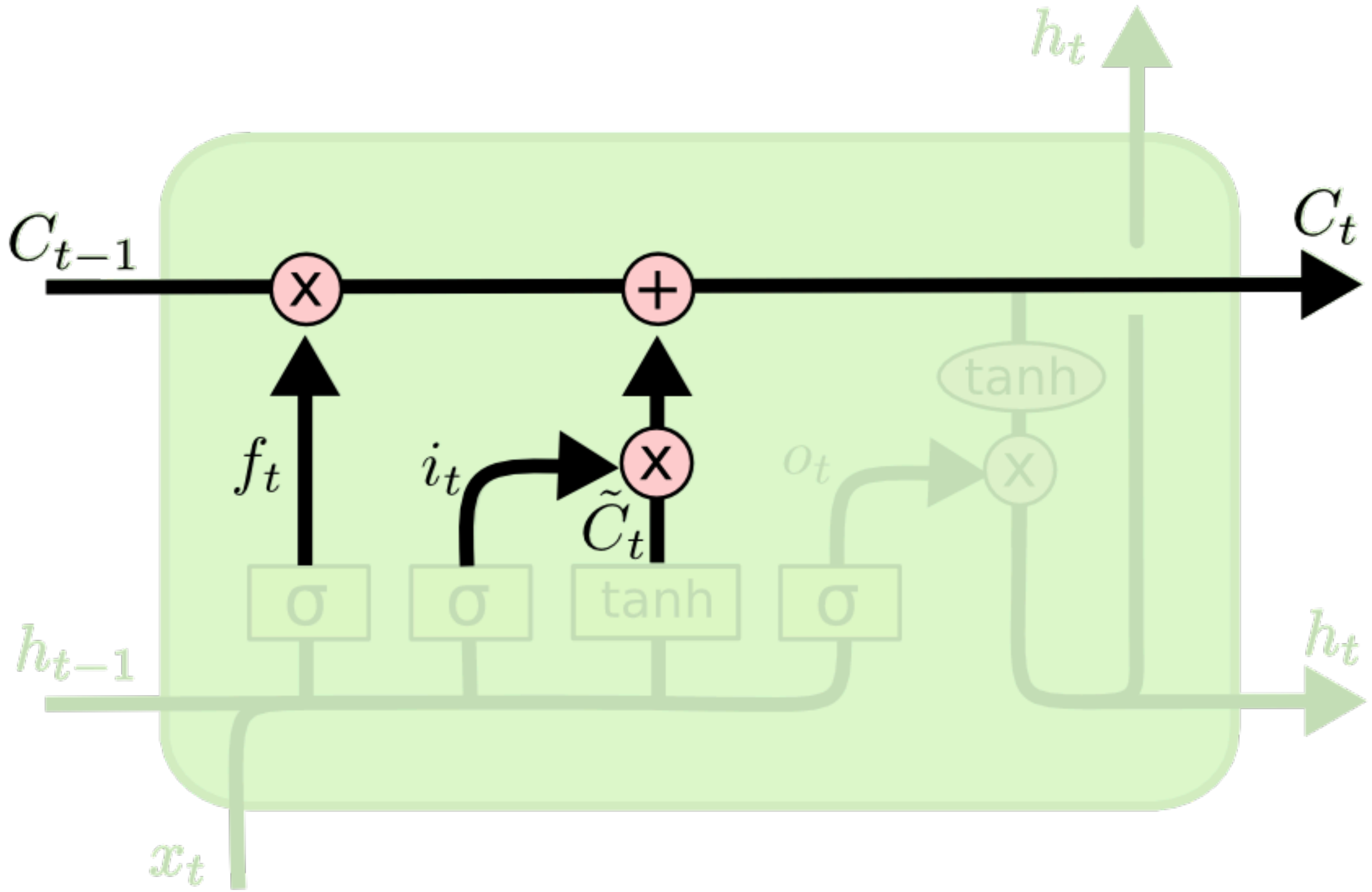
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

what to write to those indices

Decide what new information to add to the cell state.

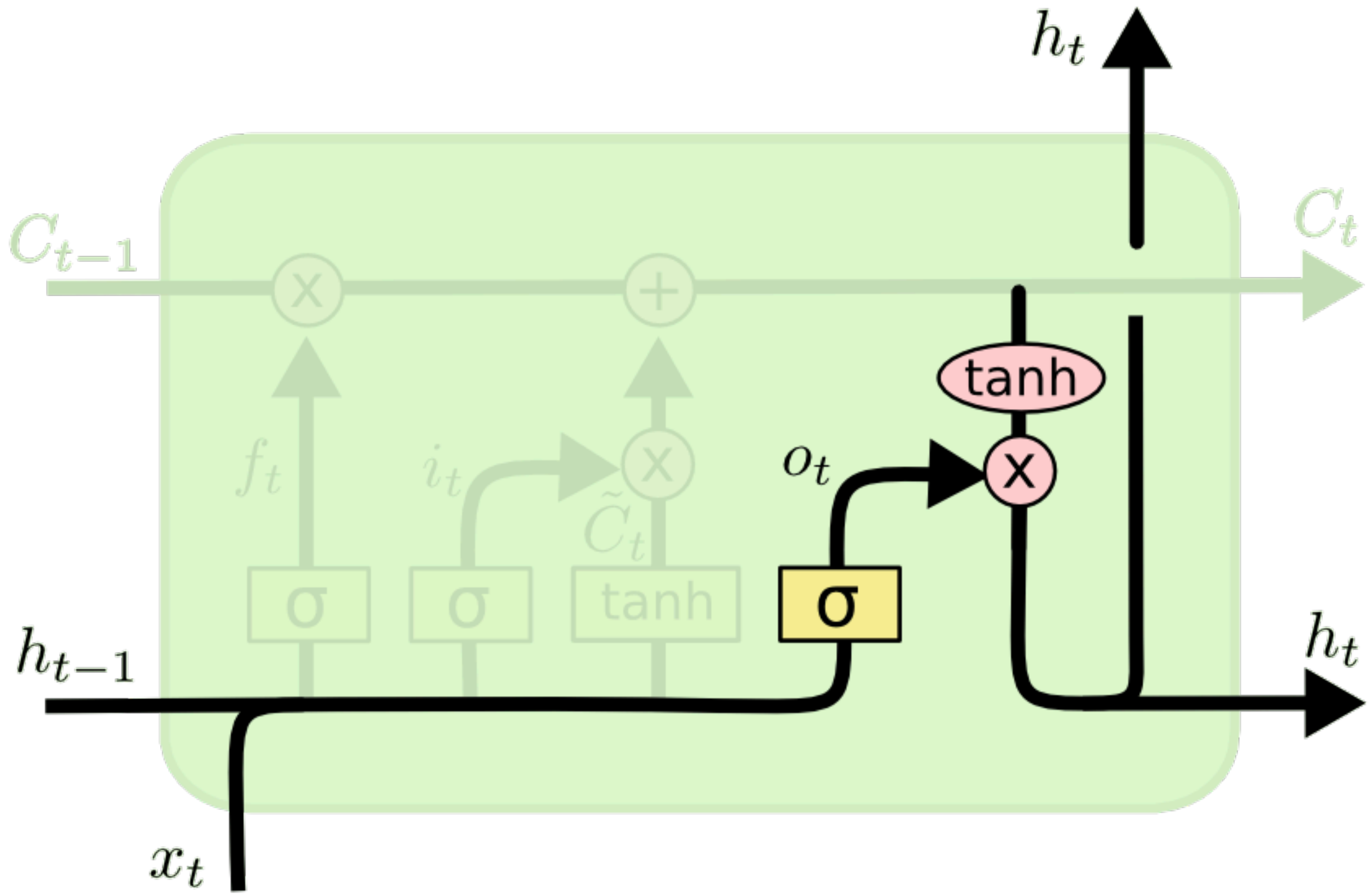
[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Forget old selected old information, write selected new information.

[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]



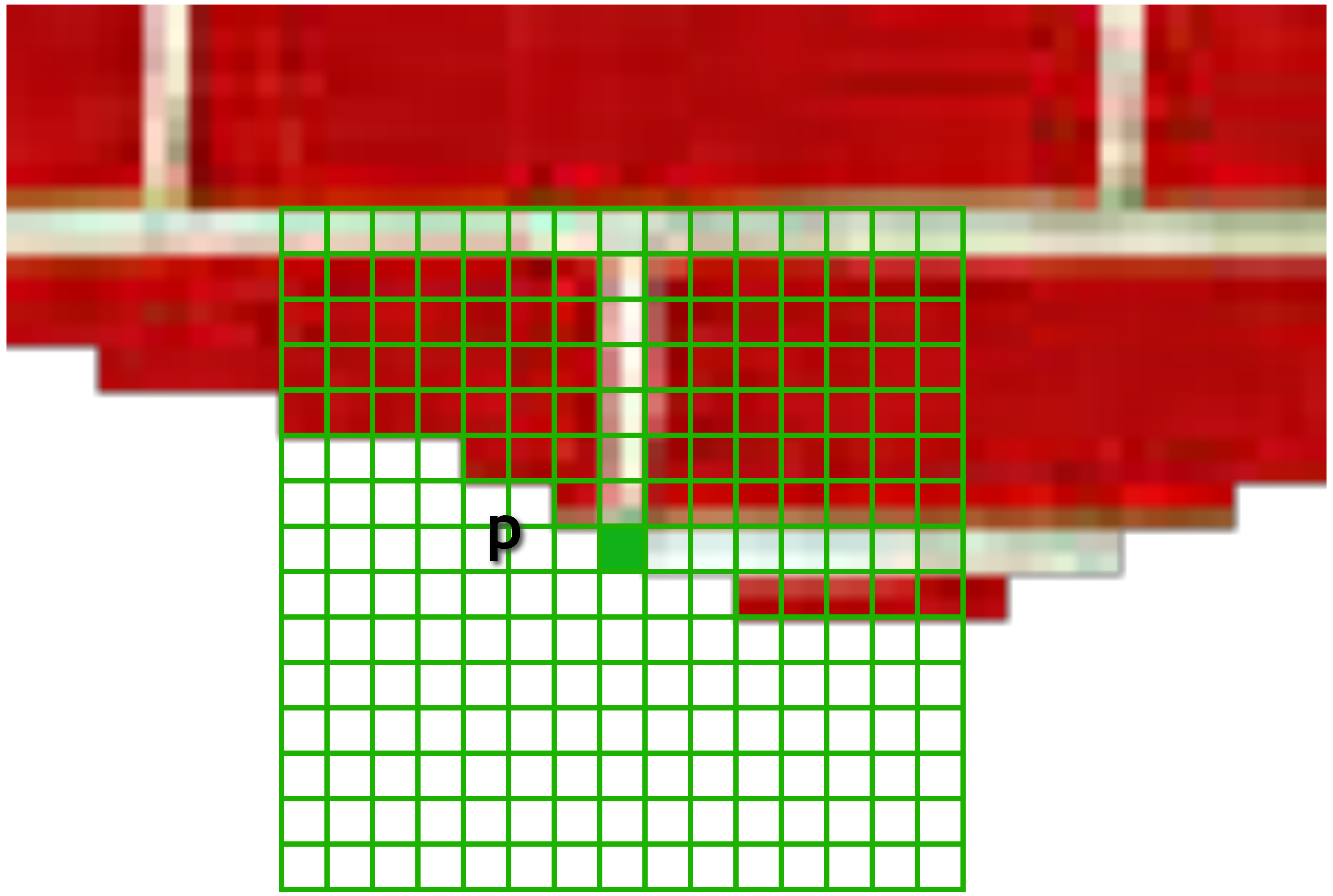
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

After having updated the cell state's information, decide what to output.

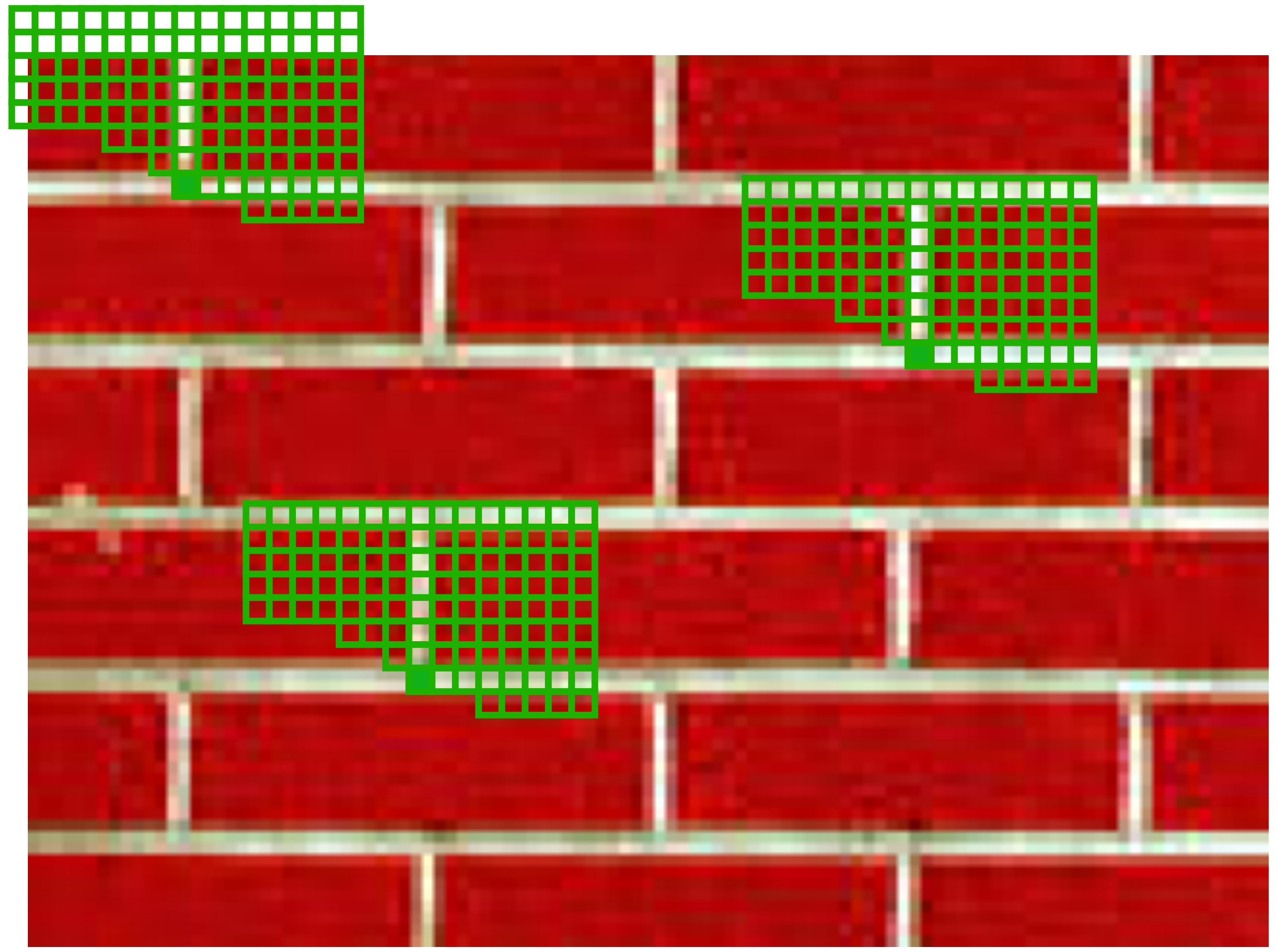
[Slide derived from Chris Olah: http://colah.github.io/posts/2015-08-Understanding-LSTMs/]

Texture synthesis by non-parametric sampling



Synthesizing a pixel

non-parametric
sampling

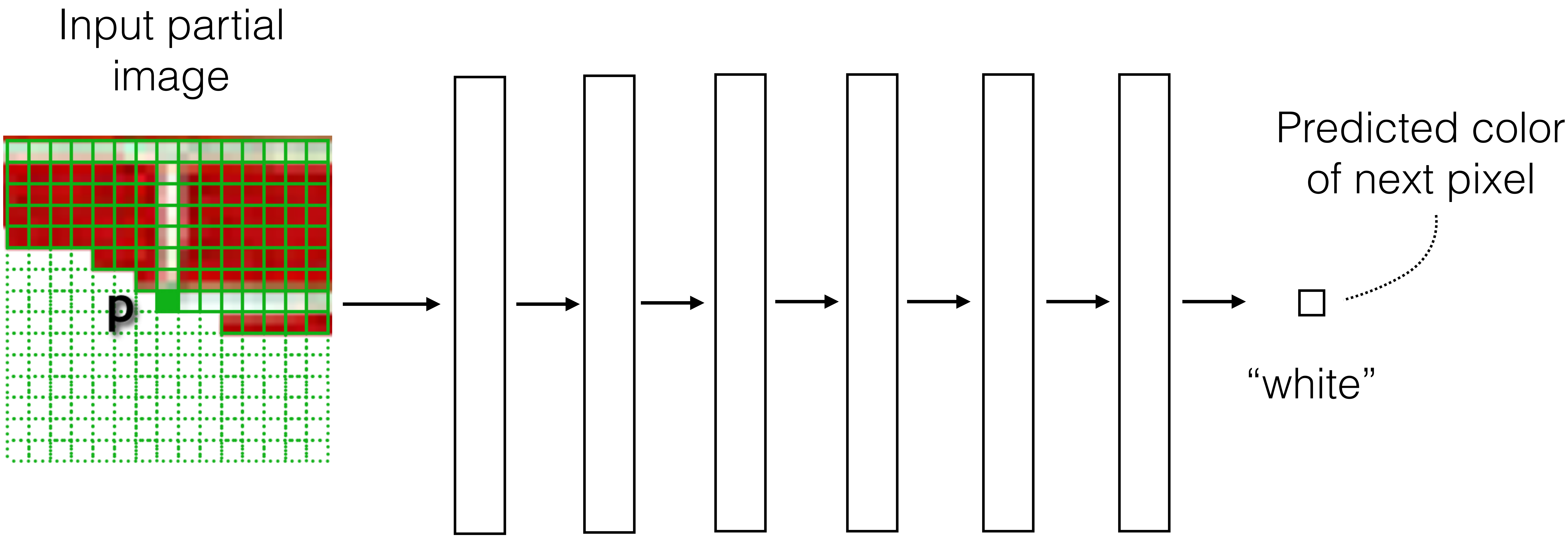


Input image

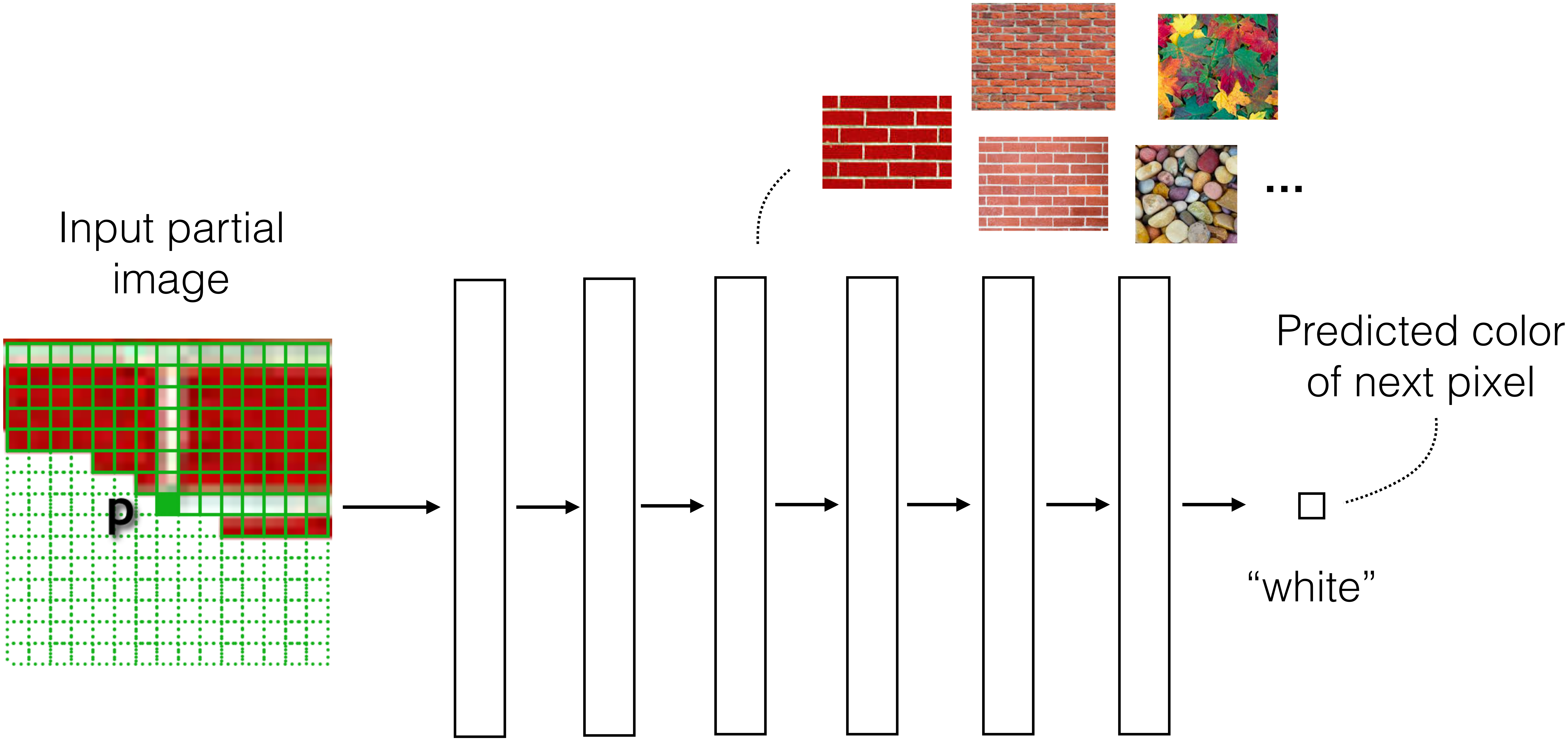
Models $P(p|N(p))$

[Efros & Leung 1999]

Texture synthesis with a deep net



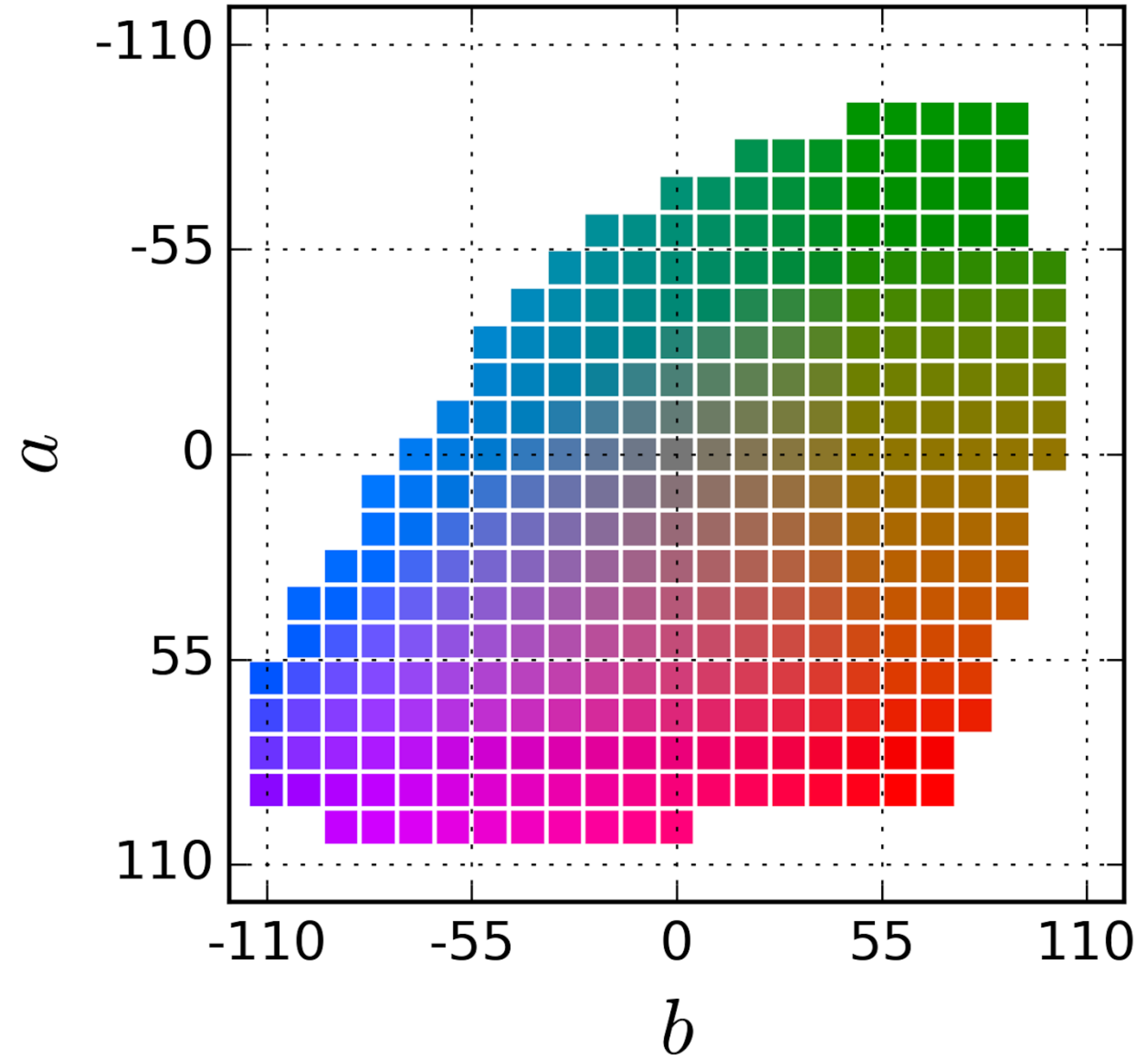
[PixelRNN, PixelCNN, van der Oord et al. 2016]



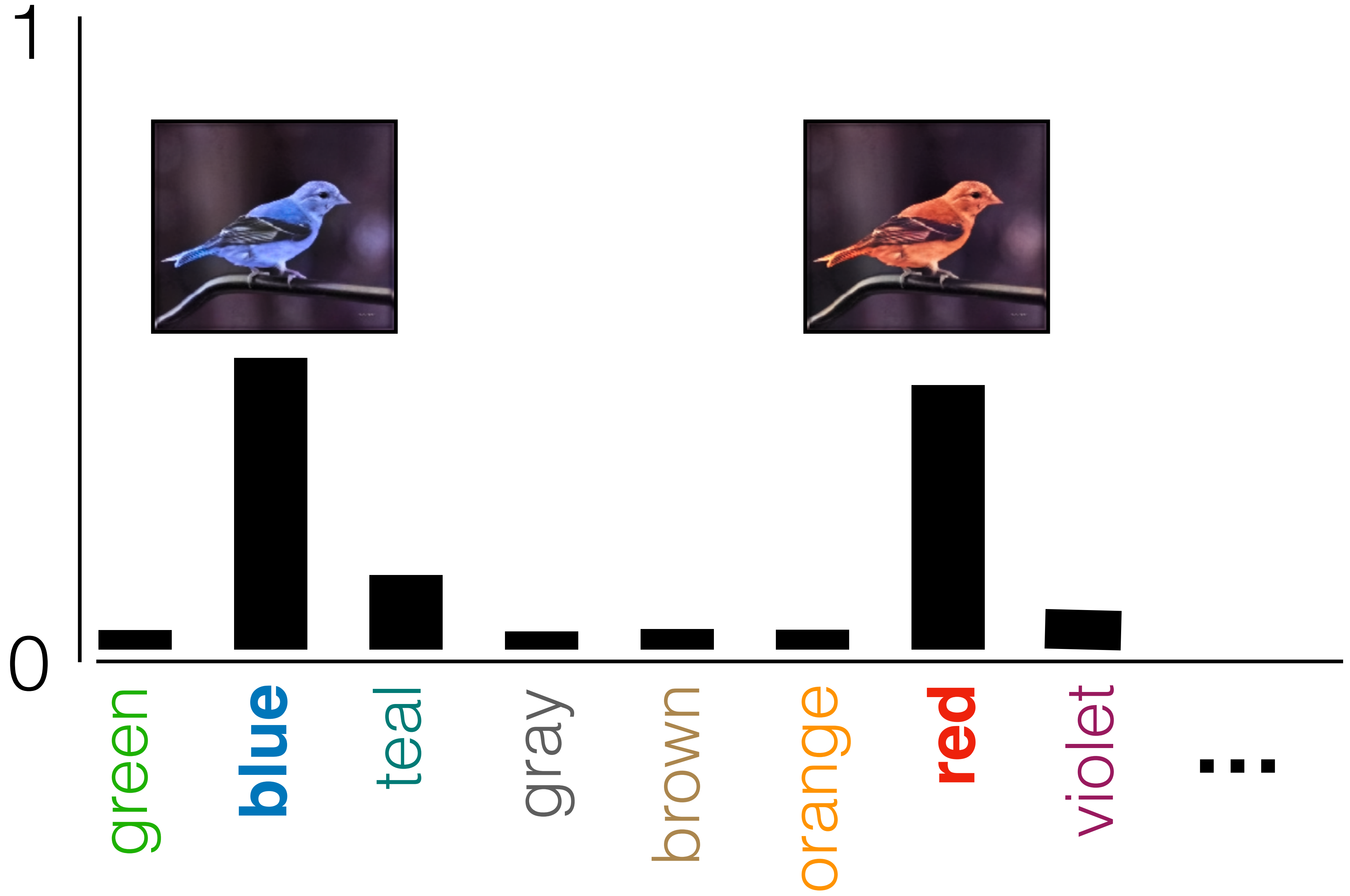
[PixelRNN, PixelCNN, van der Oord et al. 2016]

Recall from lecture 12: we can represent colors as discrete classes

$$\mathbf{y} \in \mathbb{R}^{H \times W \times K}$$



Prediction for a single pixel i,j



$$\mathcal{L}(\mathbf{y}, f_\theta(\mathbf{x})) = H(\mathbf{y}, \texttt{softmax}(f_\theta(\mathbf{x})))$$

And we can interpret the learner as modeling P(next pixel | previous pixels):

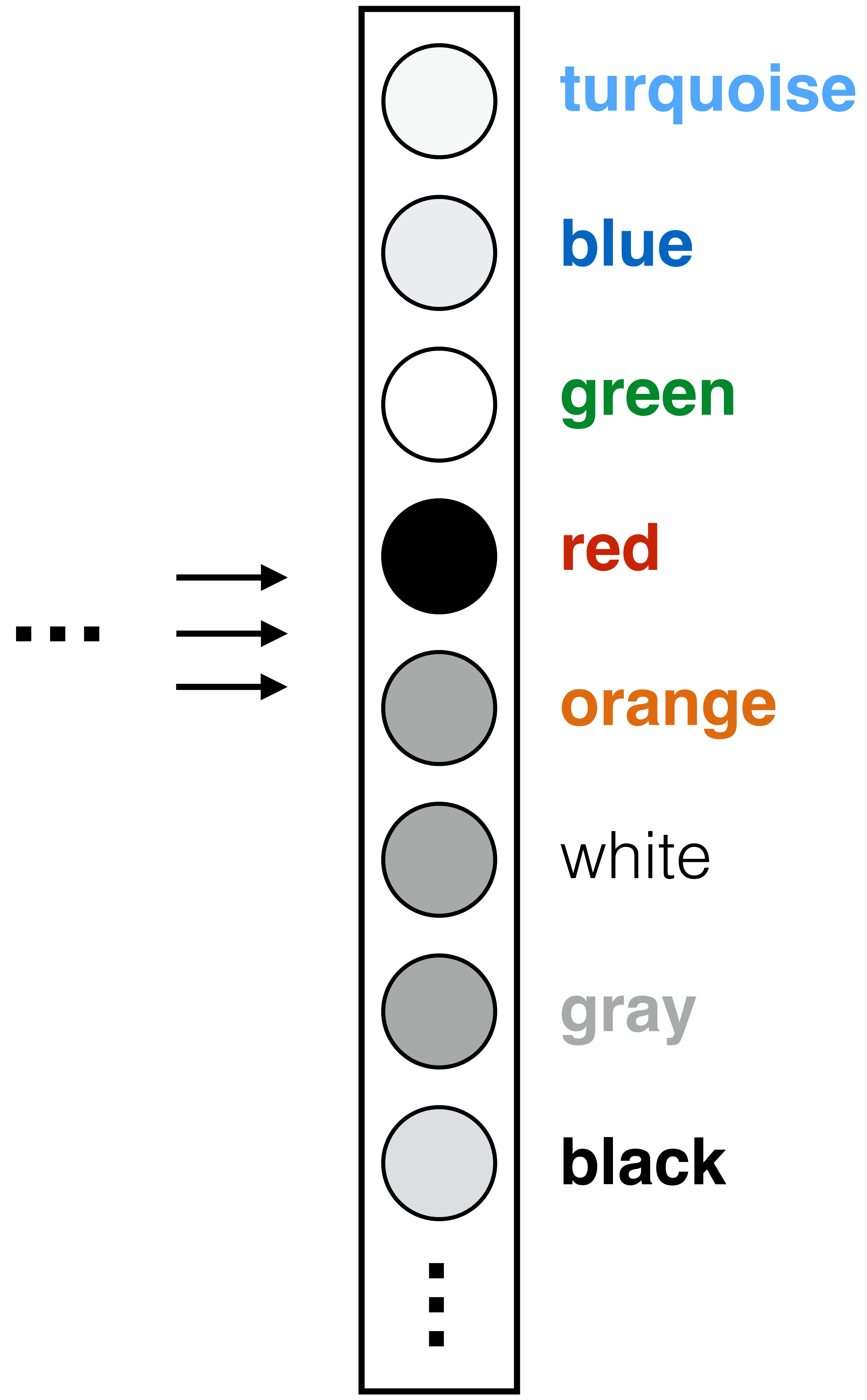
Softmax regression (a.k.a. multinomial logistic regression)

$$\hat{\mathbf{y}} \equiv [P_\theta(Y = 1|X = \mathbf{x}), \dots, P_\theta(Y = K|X = \mathbf{x})]$$ ⟵ predicted probability of each class given input **x**

$$H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$ ⟵ picks out the -log likelihood of the ground truth class $\mathbf{y}$ under the model prediction $\hat{\mathbf{y}}$

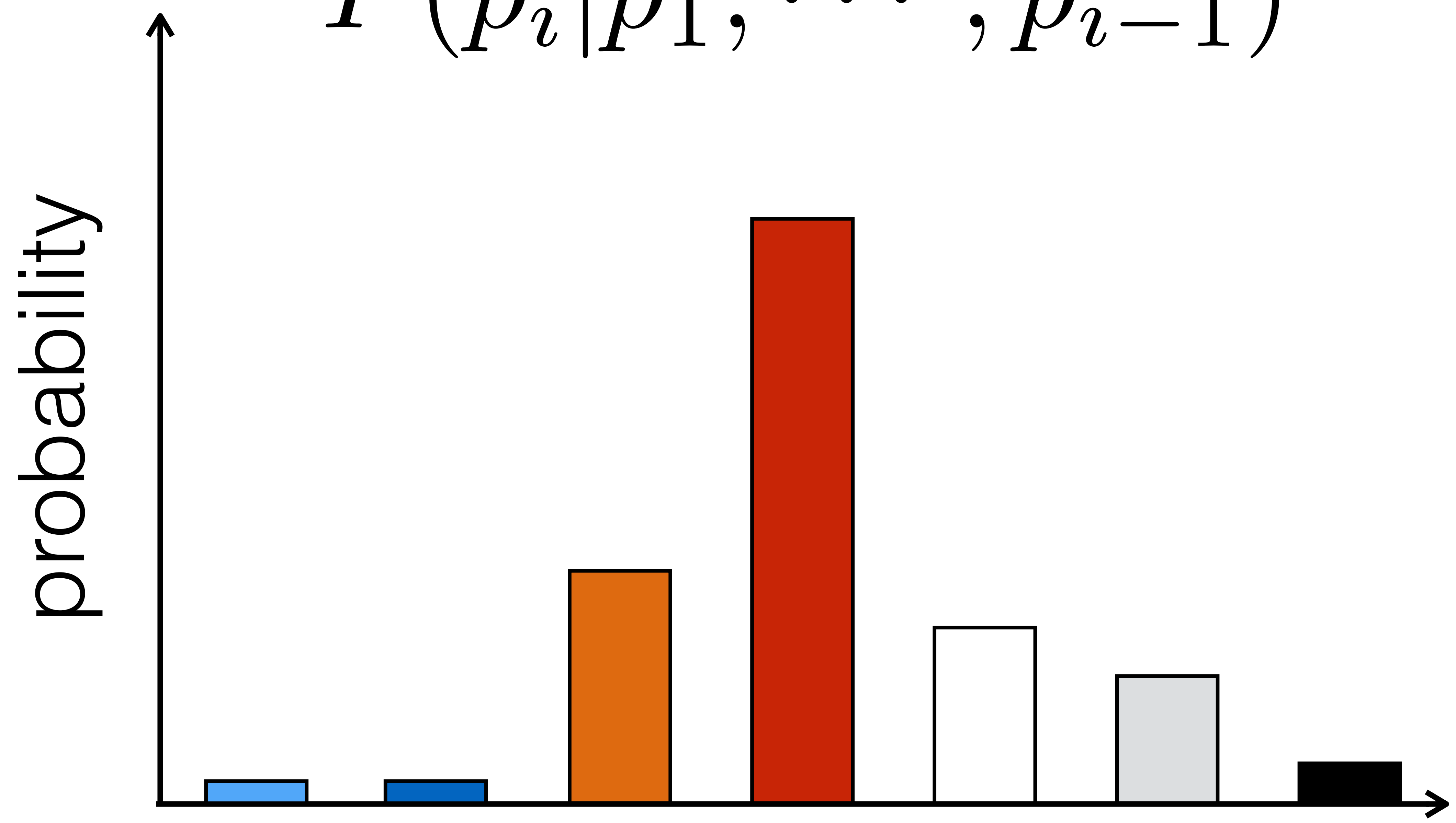
$$f^* = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} H(\mathbf{y}_i, \hat{\mathbf{y}}_i)$$ ⟵ max likelihood learner!

Network output

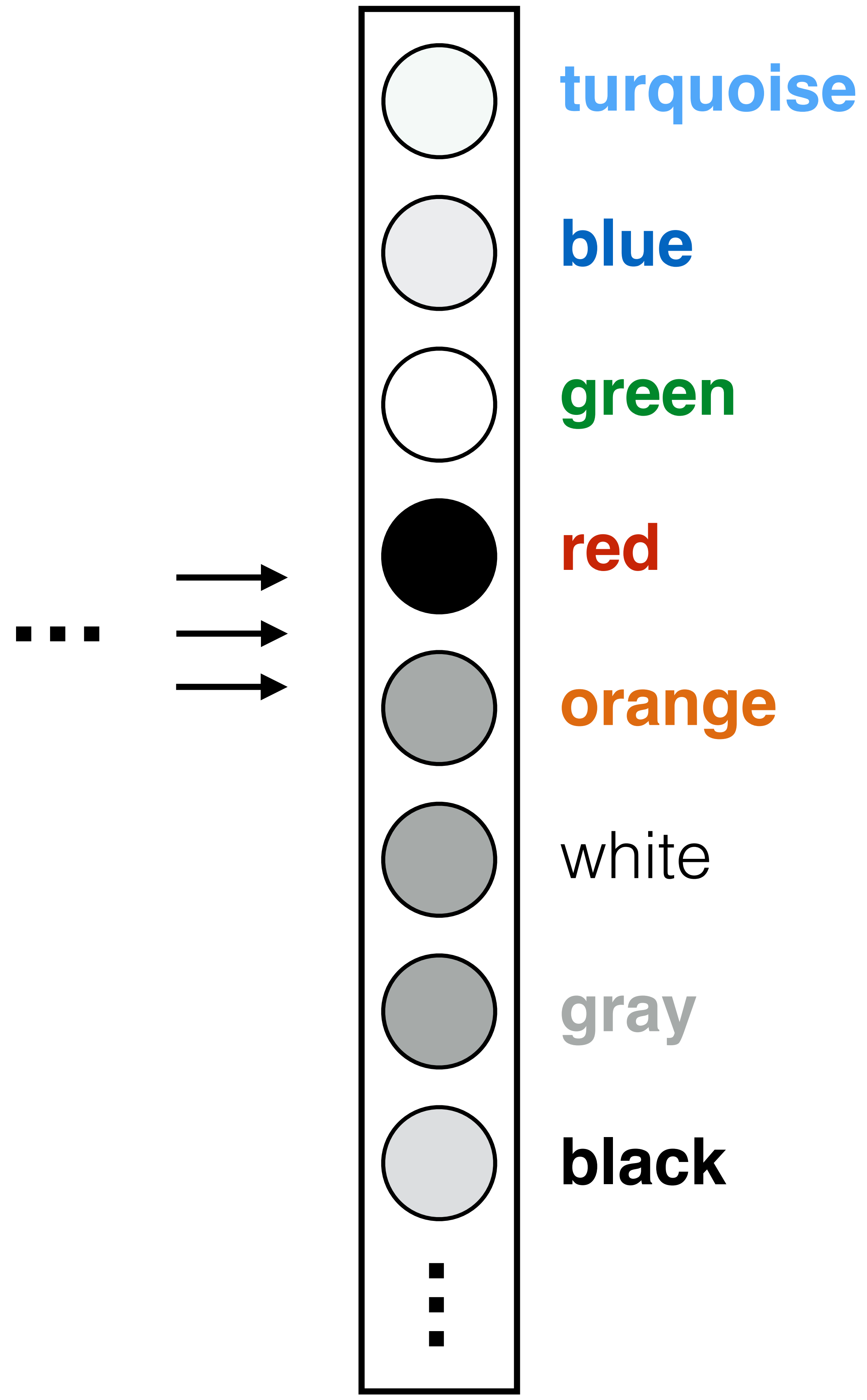


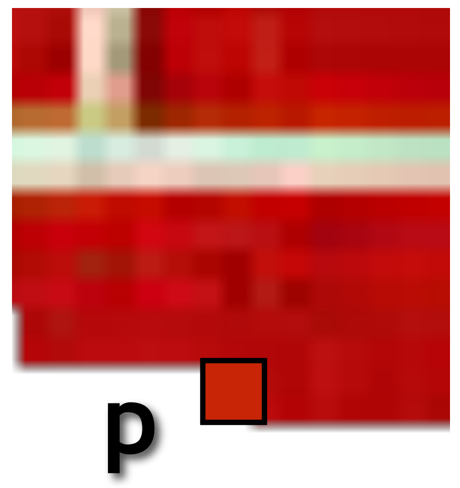
P(next pixel | previous pixels)

$$P(p_i|p_1, \cdots, p_{i-1})$$

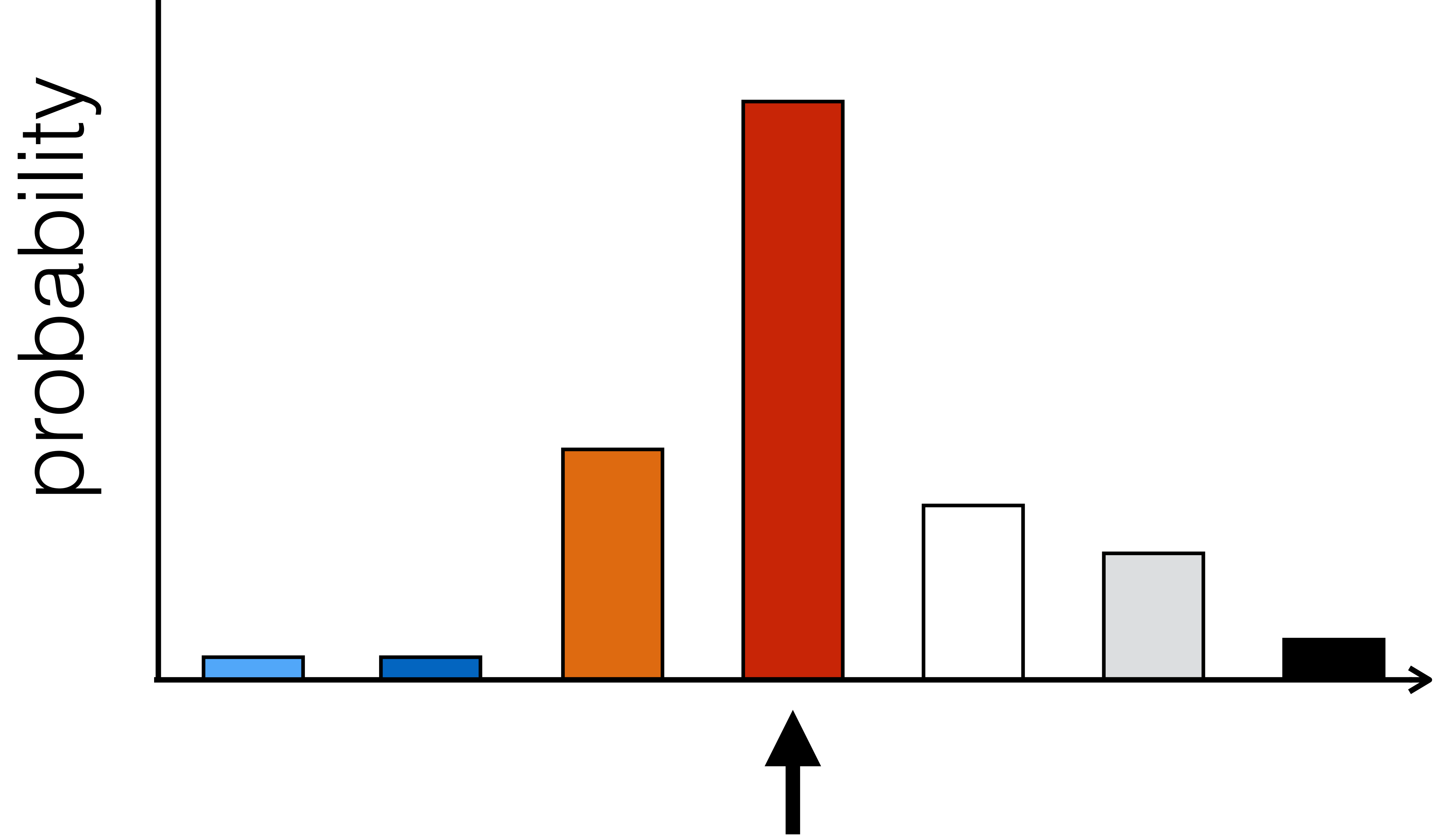


Network output

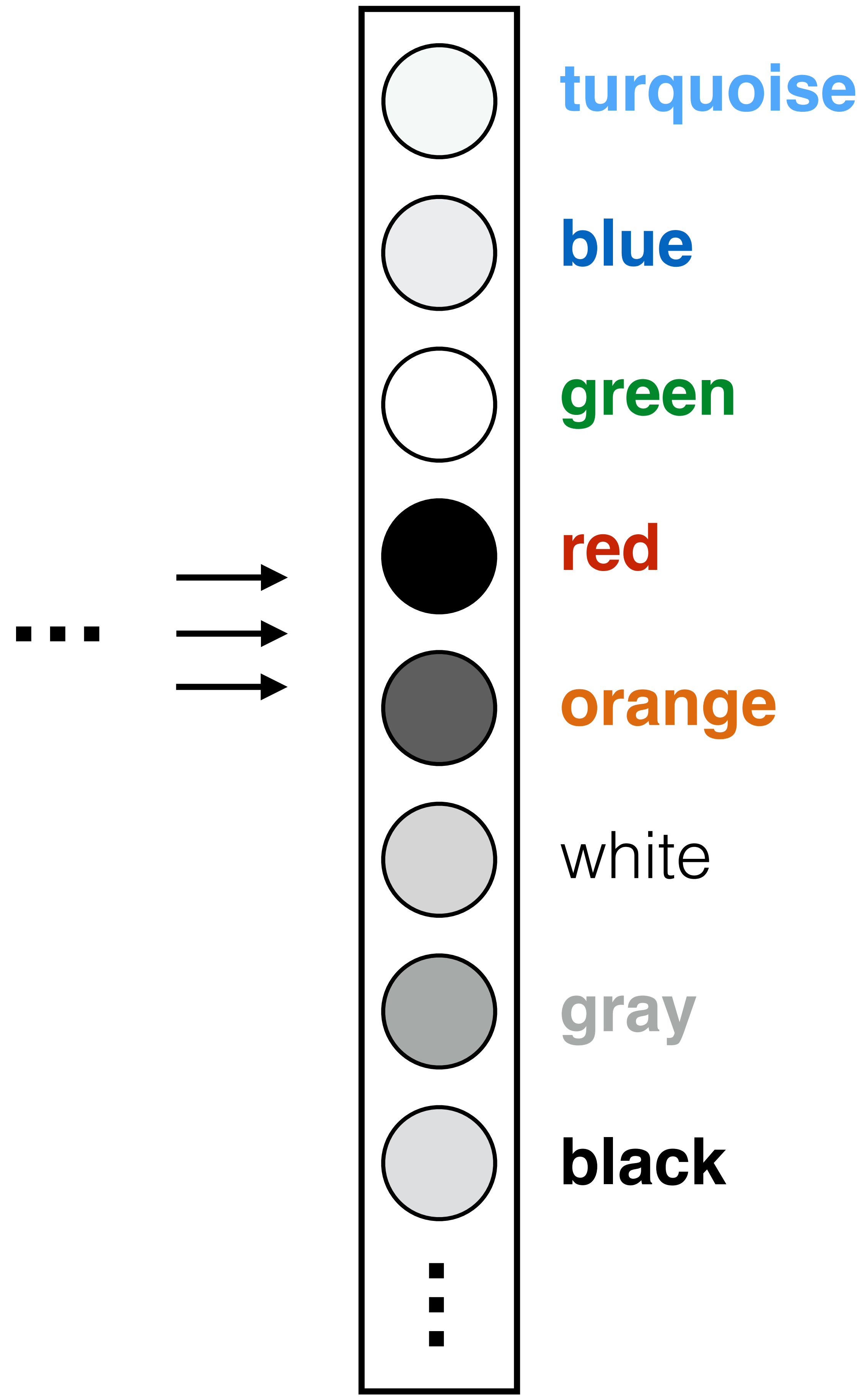


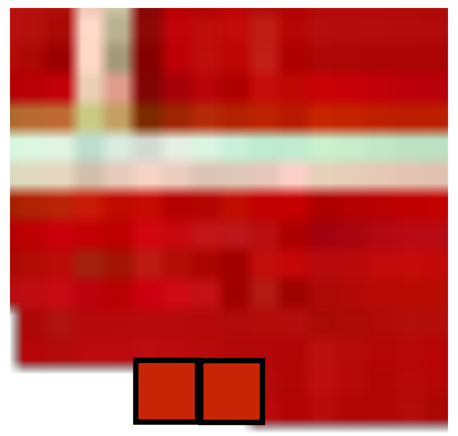


$$p_i \sim P(p_i | p_1, \cdots, p_{i-1})$$

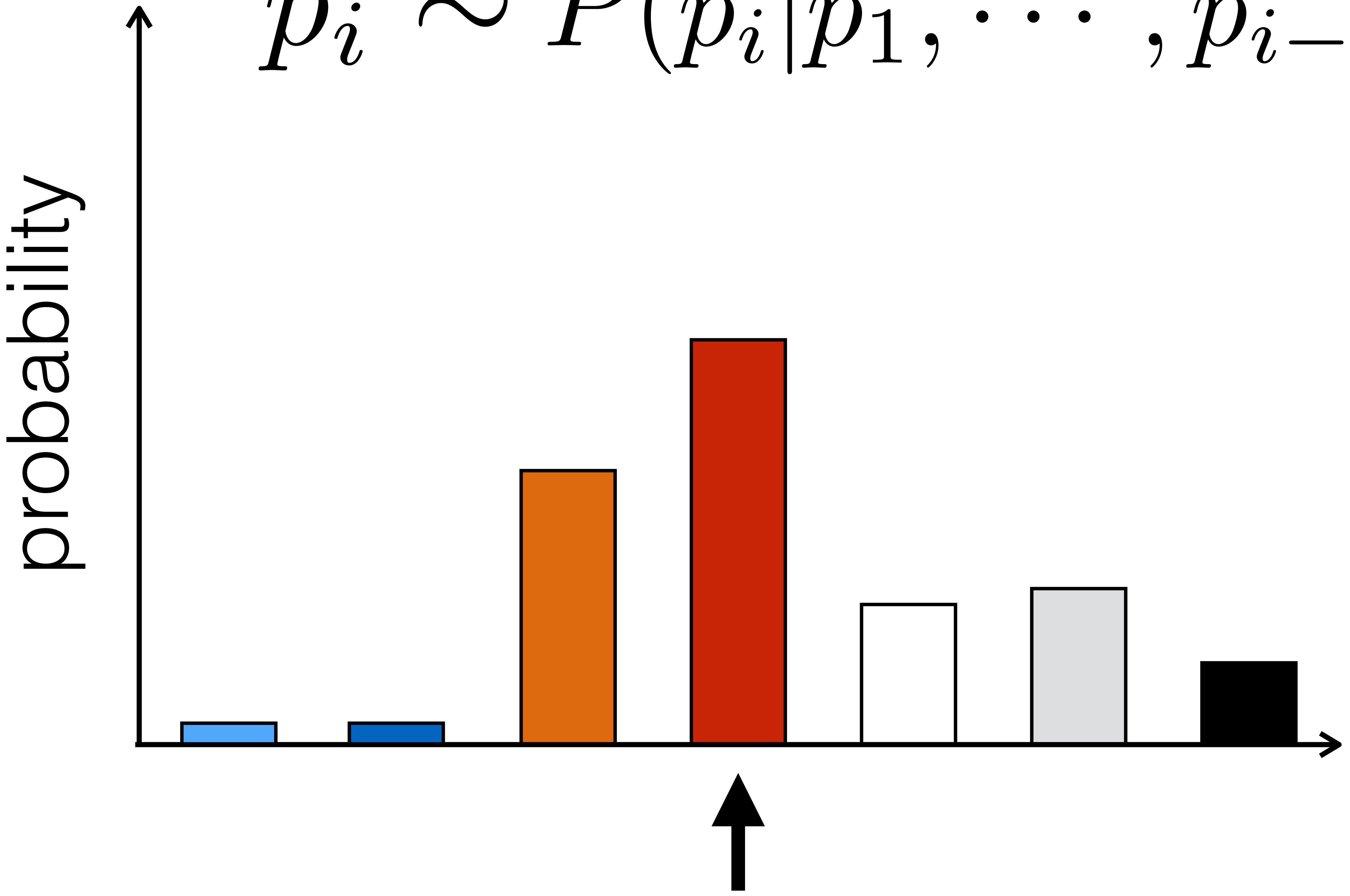


Network output

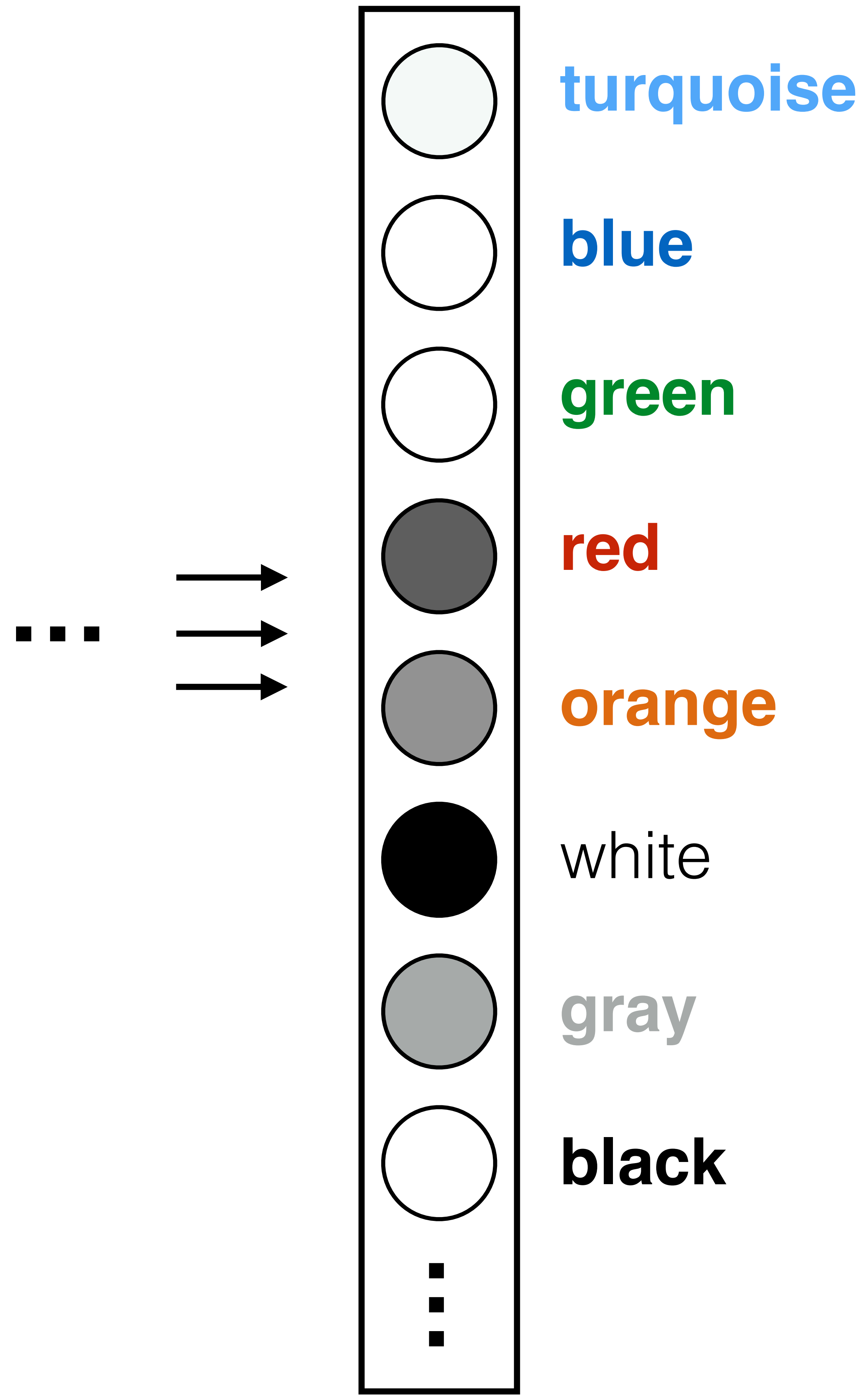


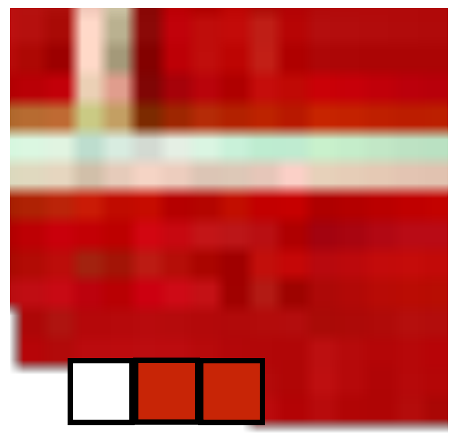


$$p_i \sim P(p_i | p_1, \cdots, p_{i-1})$$

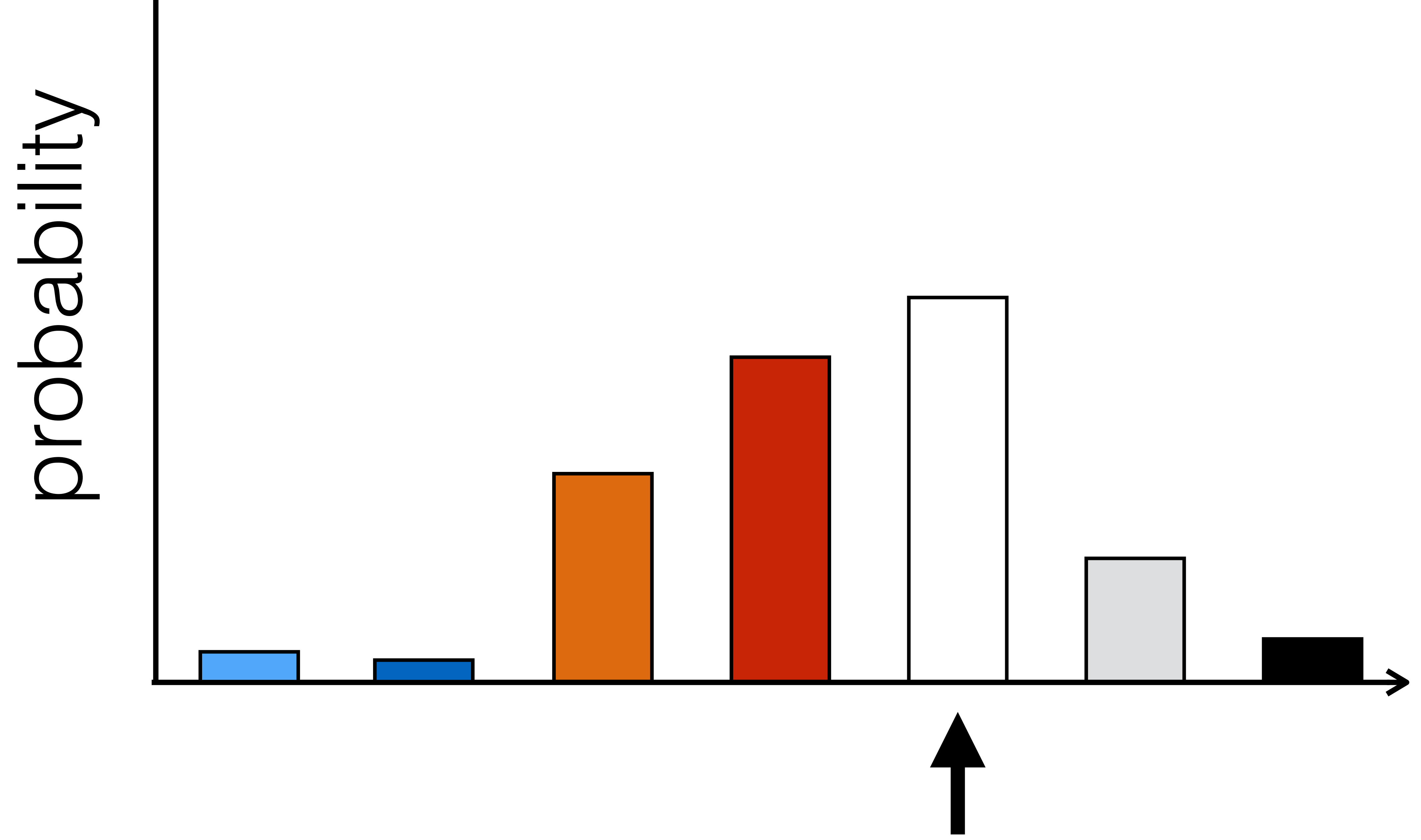


Network output

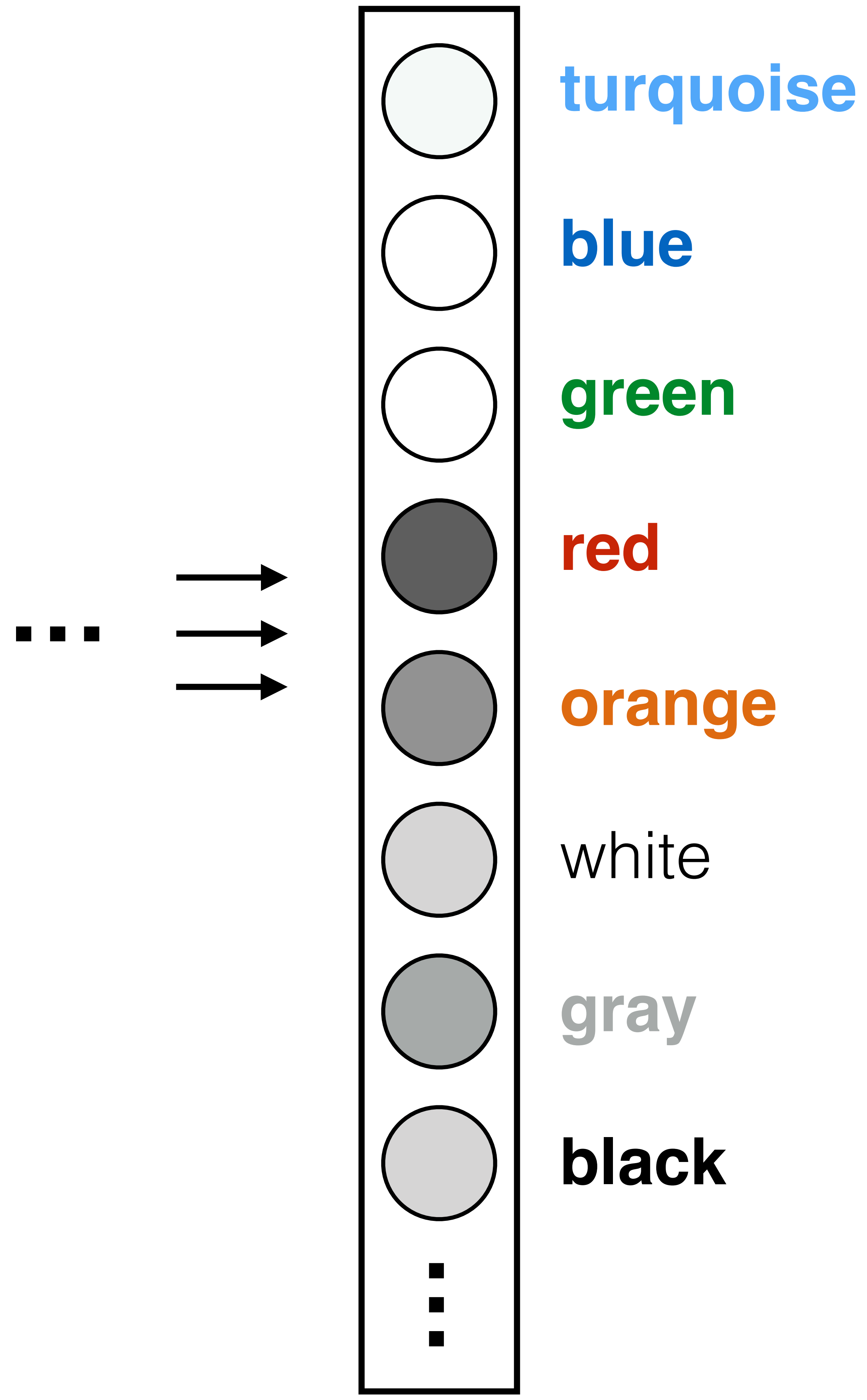




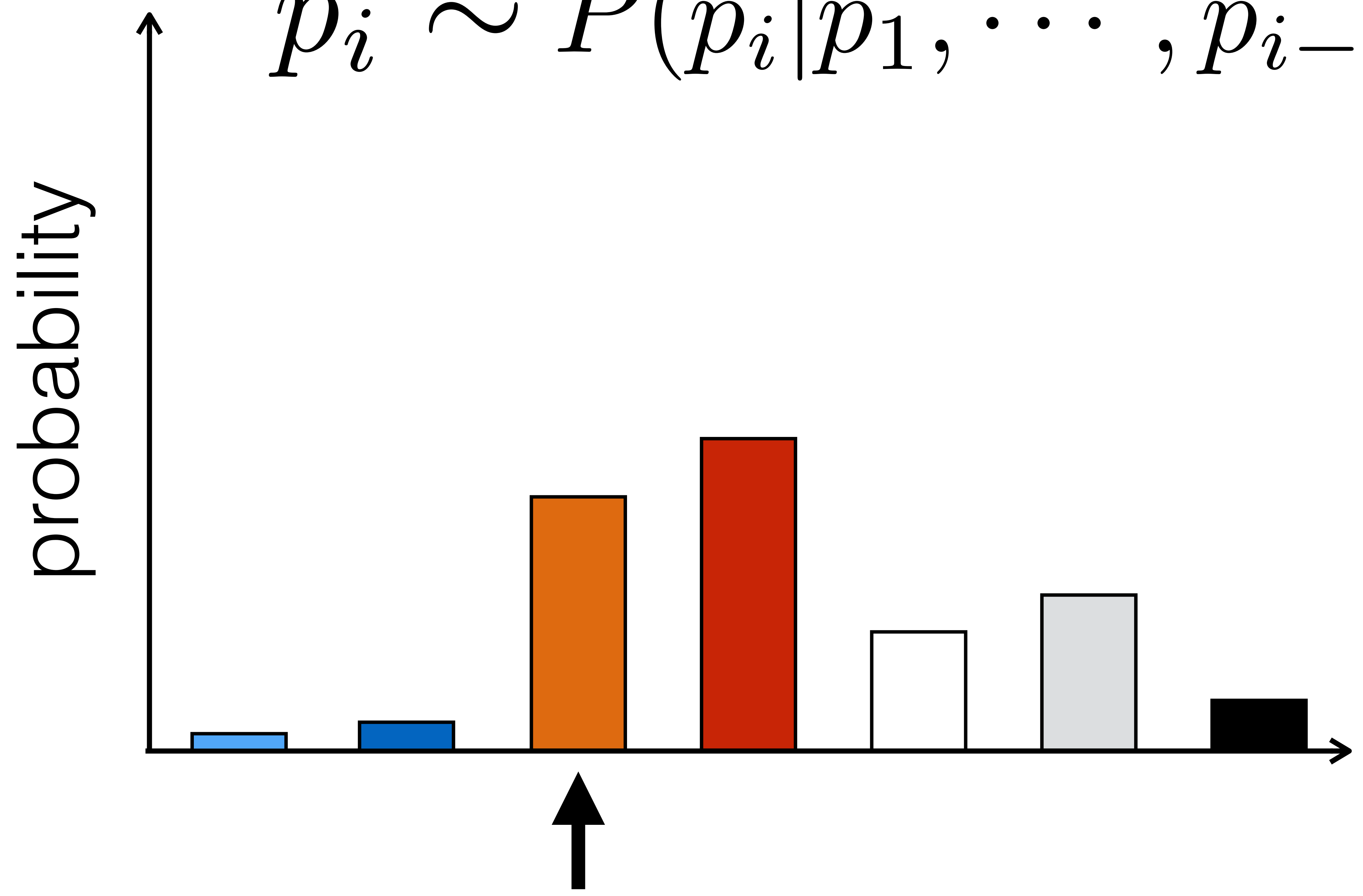
$$p_i \sim P(p_i | p_1, \cdots, p_{i-1})$$

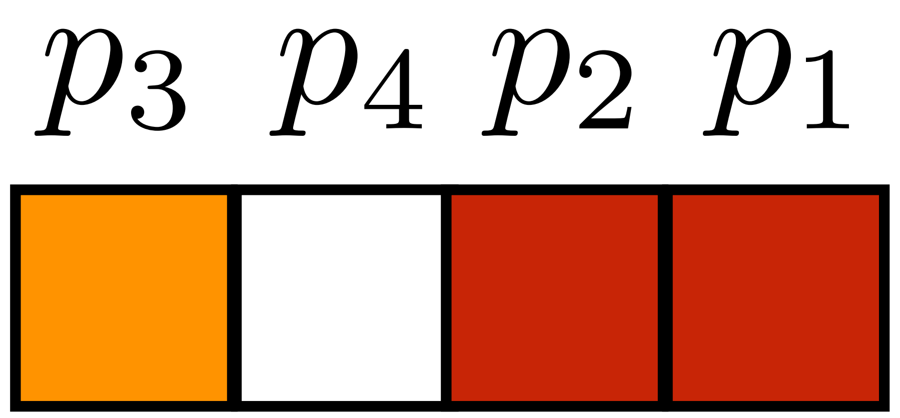


Network output



$$p_i \sim P(p_i | p_1, \cdots, p_{i-1})$$





$$p_1 \sim P(p_1)$$

$$p_2 \sim P(p_2|p_1)$$

$$p_3 \sim P(p_3|p_1, p_2)$$

$$p_4 \sim P(p_4|p_1, p_2, p_3)$$

$$\{p_1, p_2, p_3, p_4\} \sim P(p_4|p_1, p_2, p_3)P(p_3|p_1, p_2)P(p_2|p_1)P(p_1)$$

$$p_i \sim P(p_i|p_1, \ldots, p_{i-1})$$

$$\boxed{\mathbf{p} \sim \prod_{i=1}^{N} P(p_i|p_1, \ldots, p_{i-1})}$$

Autoregressive probability model

$$\mathbf{p} \sim \prod_{i=1}^{N} P(p_i | p_1, \ldots, p_{i-1})$$

$$P(\mathbf{p}) = \prod_{i=1}^{N} P(p_i | p_1, \ldots, p_{i-1}) \longleftarrow \textbf{General product rule}$$

The sampling procedure we defined above takes exact samples from the learned probability distribution (pmf).

Multiplying all conditionals evaluates the probability of a full joint configuration of pixels.

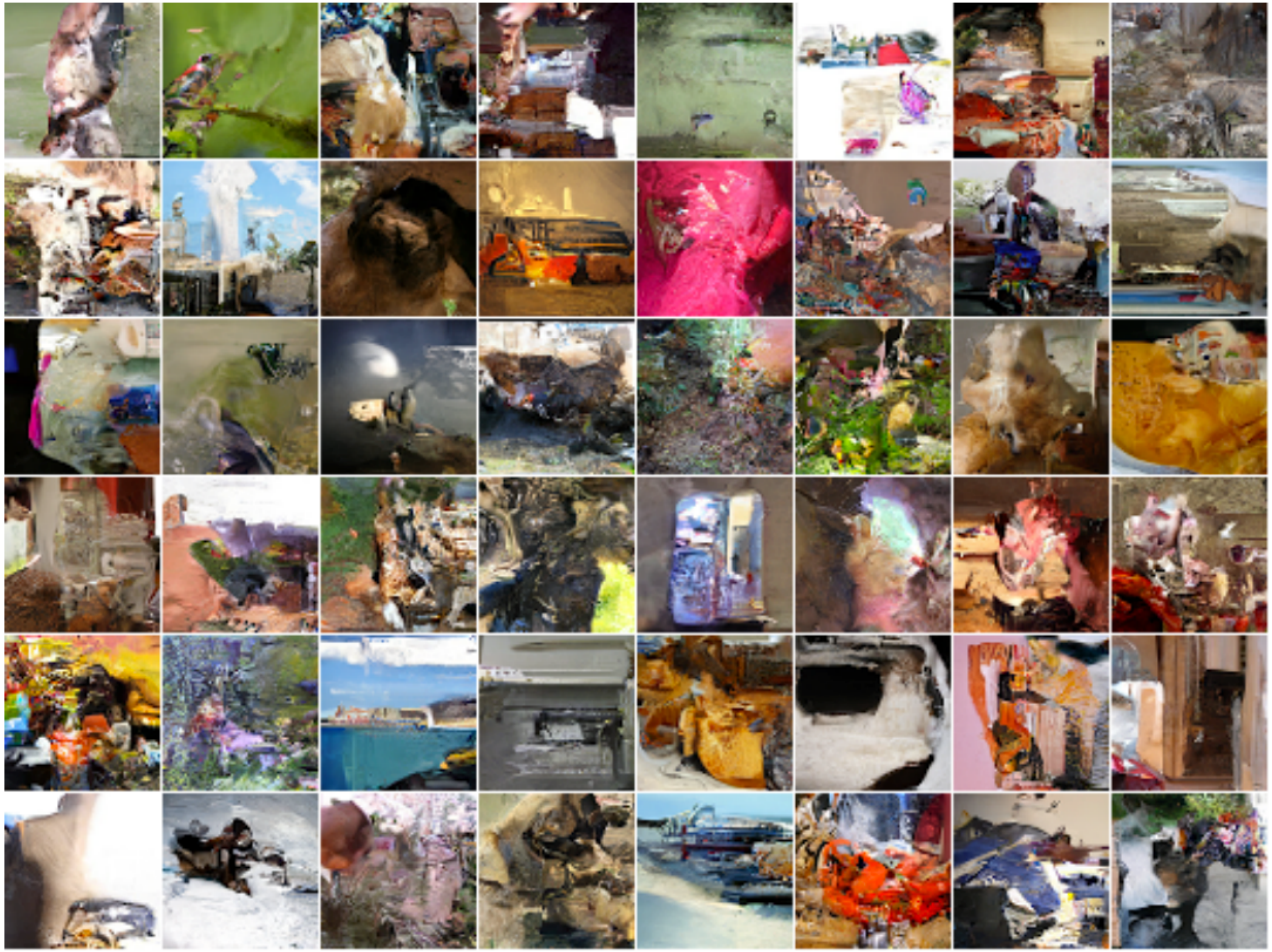
Autoregressive probability model

$$\mathbf{p} \sim P(\mathbf{p})$$

Models that allow us to sample, i.e. *generate*, images from scratch are called **generative models**.

We will see more examples in a future lecture.

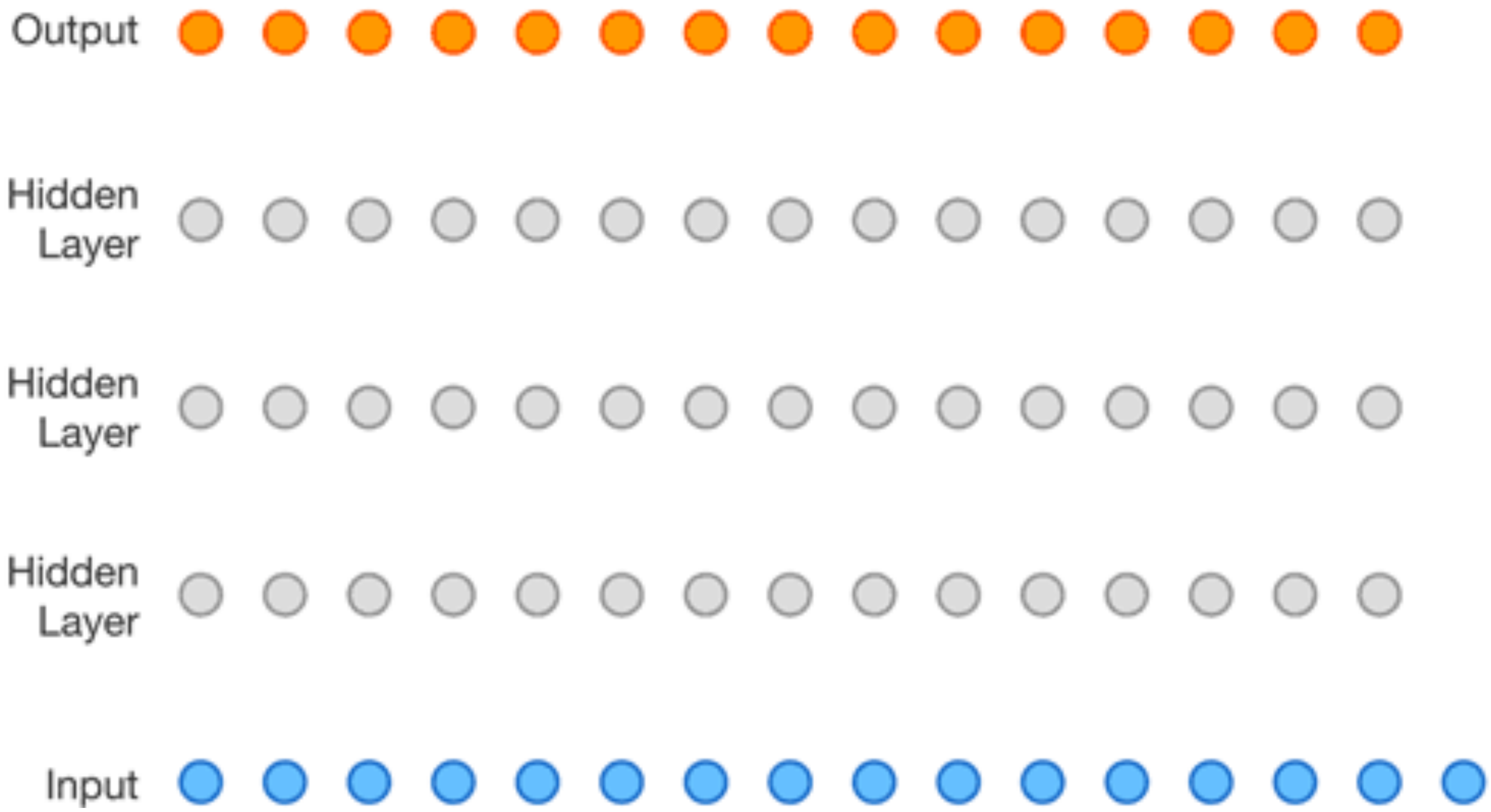
Samples from PixelRNN



[PixelRNN, van der Oord et al. 2016]

Image completions (conditional samples) from PixelRNN

[PixelRNN, van der Oord et al. 2016]



[**Wavenet**, https://deepmind.com/blog/wavenet-generative-model-raw-audio/]