Lecture 13: Video and temporal models

Announcements

- PS6 due next Weds.
- Project proposal information out
  - Rolling deadline, due Nov. 13
- Discussion section: project office hours + GANs

Project proposal

- Due November 13th
- Rolling deadline
- What we want: 1 page summary of what you’d like to do.
- Not worth much of total grade. Graded pass/fail. We just want to know whether it’s an acceptable project.

Project proposal

2 Project ideas

To help you think of projects, we've provided a few ideas below. Please note that these projects only cover a very small portion of the possible things you can do — most involve reimplementing and extending a paper. We encourage you to propose your own, creative project ideas, and to use these as a starting point! We may also add new project ideas to this list in the coming weeks.

Applications of vision. Apply computer vision to a task that's important to you! We highly encourage this option if it appeals to you, since it's often the most fun option. For example, students in previous EECS 442 classes have applied computer vision to Settlers of Catan, measured the volume of liquid that could be held by a teacup, and analyzed the coffee coming out of an espresso machine. Often these projects will involve applying a few different computer vision models to a task, and analyzing the results.

Image synthesis. Implement a (small) version of a generative image model, such as VQ-GAN [1] or a diffusion model [2].

Extend an existing image synthesis model. Extend an existing image synthesis model, such as Stable Diffusion [3] in an interesting way (see here [4] for architecture details).

Video magnification. Implement a motion magnification algorithm, such as the method of Wadhwa et al. [5]. Try running it on your own videos, too.

Stereo. Implement a system that can estimate depth from a collection of photos using stereo. An easy-to-implement reference point is Goesele et al. [6].

Source: A. Torralba

Source: A. Torralba

Source: A. Torralba

“What will the girl do next?”

Source: A. Torralba

Simple video task: action recognition

Making latte art

Jaywalking

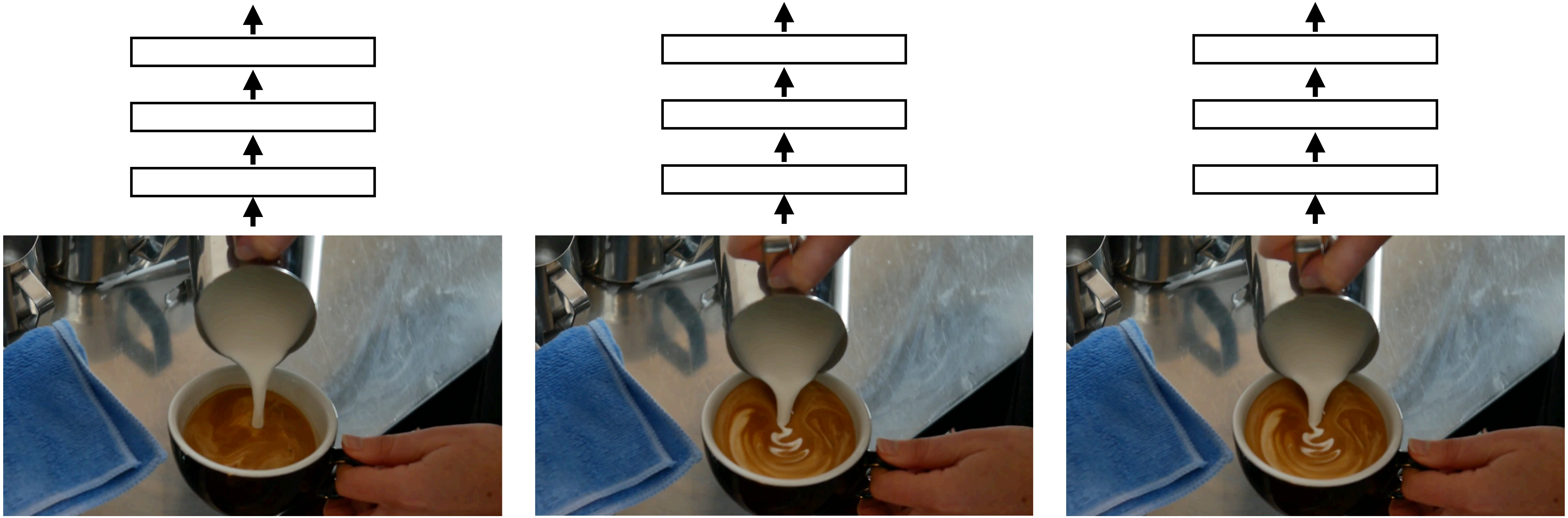
Grooming dog

Analogous to image recognition. Useful testbed for designing video models.

Examples from the Kinetics dataset [Carreira et al. 2017 - 2019]
700 human activity classes, 650K 10-sec videos

Simple model: averaging

$\frac{1}{3}$ p(making latte art | I_1) + $\frac{1}{3}$ p(making latte art | I_2) + $\frac{1}{3}$ p(making latte art | I_3)



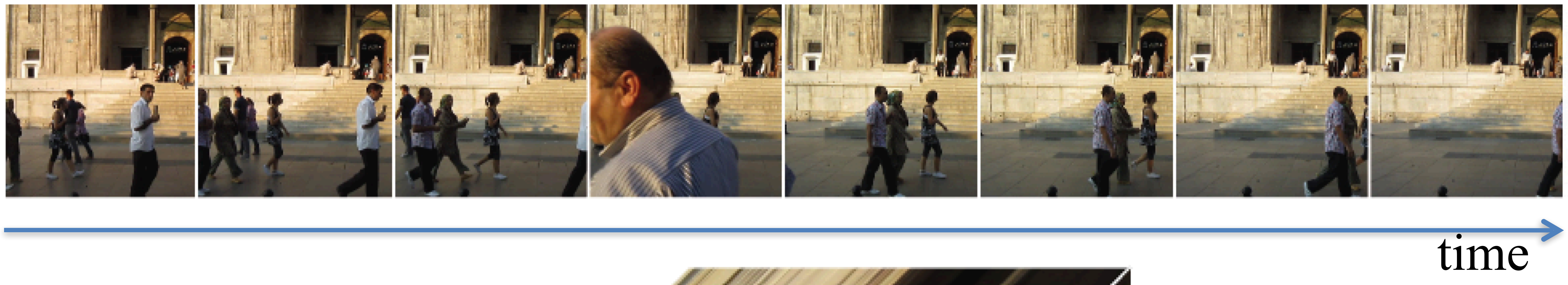
Can't analyze motion.

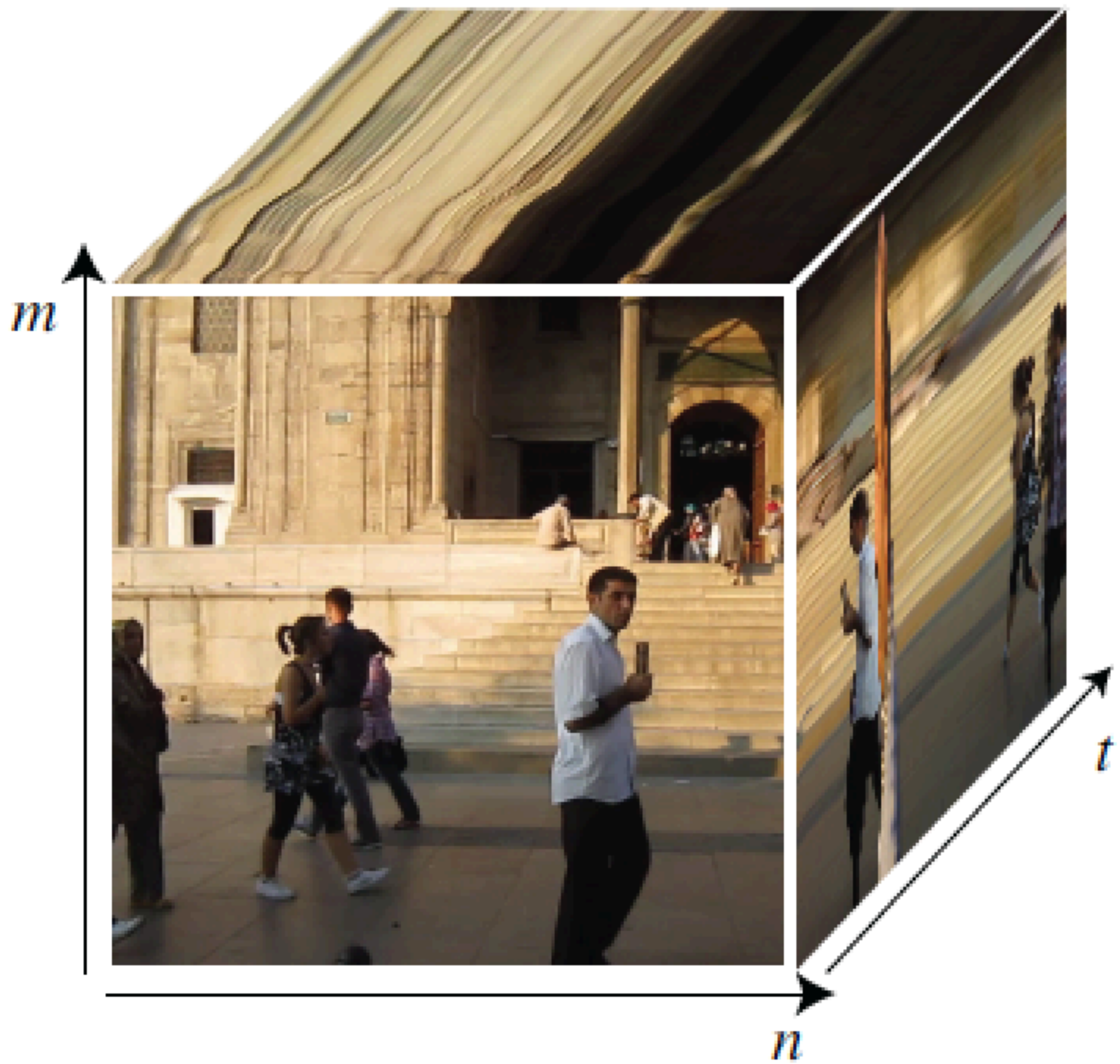
Temporal filtering

Temporal filtering

Source: Freeman, Torralba, Isola

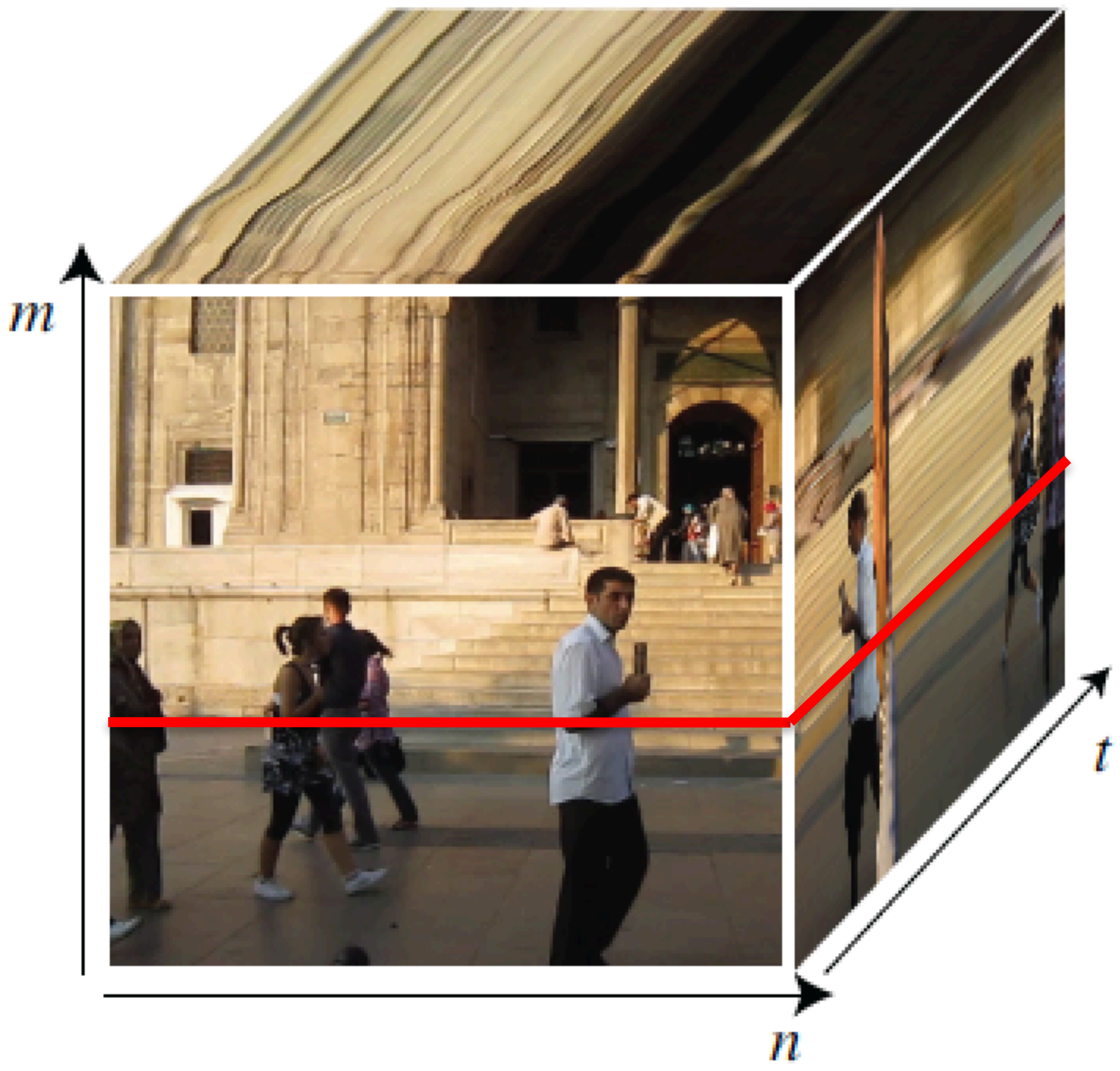
Videos



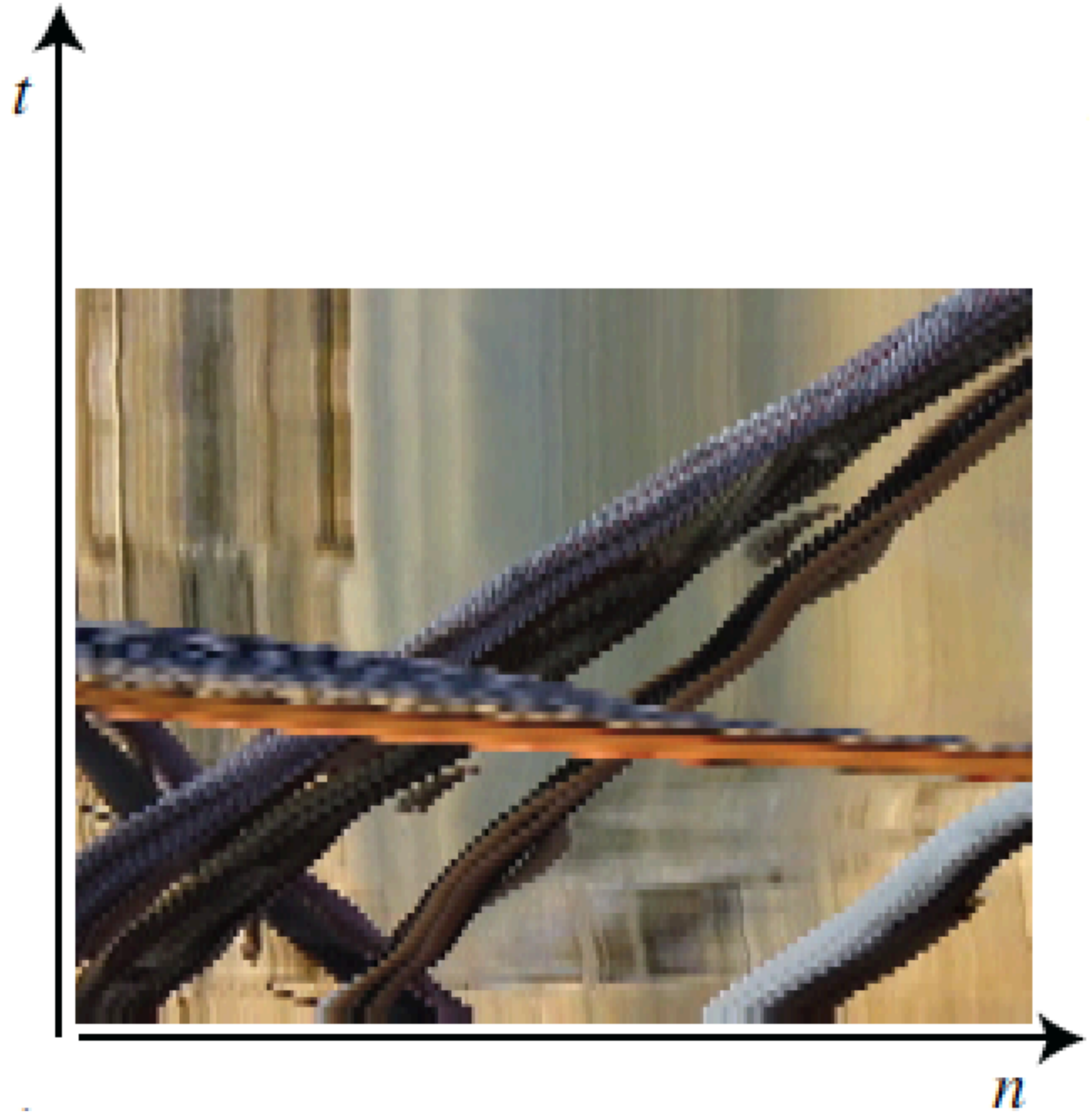


Source: Freeman, Torralba, Isola

Videos

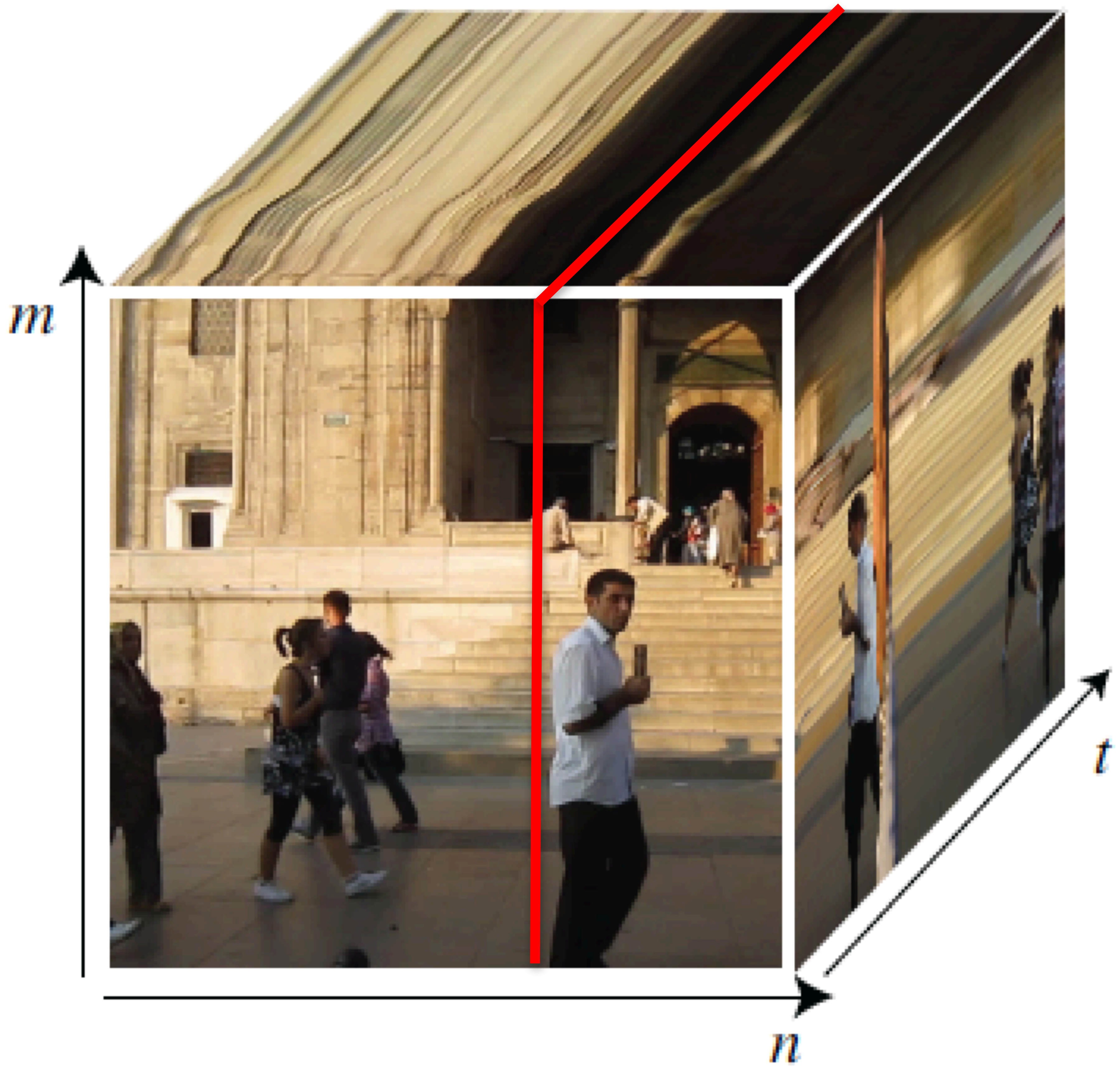


Cube size = 128x128x90

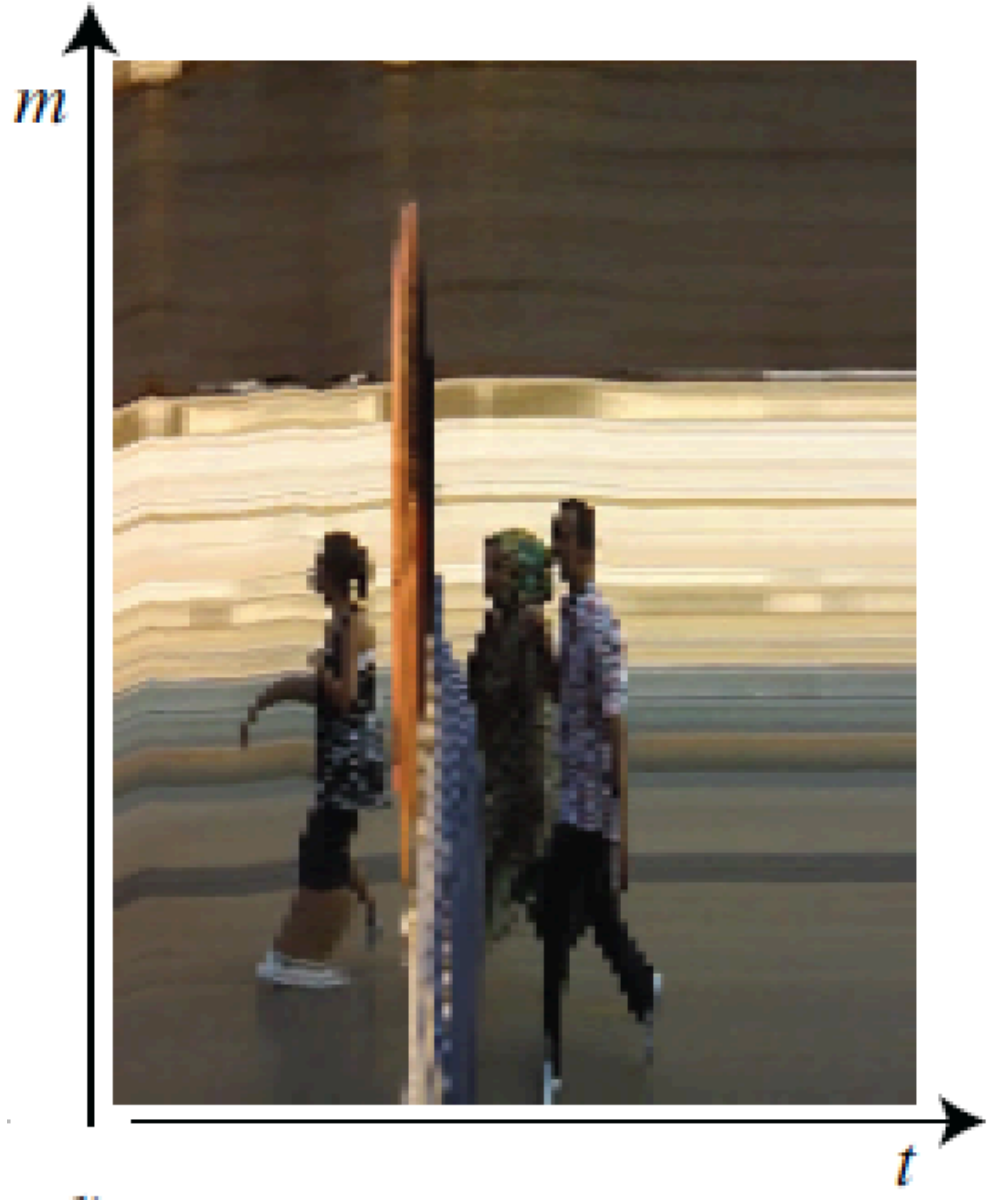


Source: Freeman, Torralba, Isola

Videos



Cube size = 128x128x90



Source: Freeman, Torralba, Isola

Examples of temporal filtering

Temporal median filtering

Source: Alexei Efros

Background subtraction

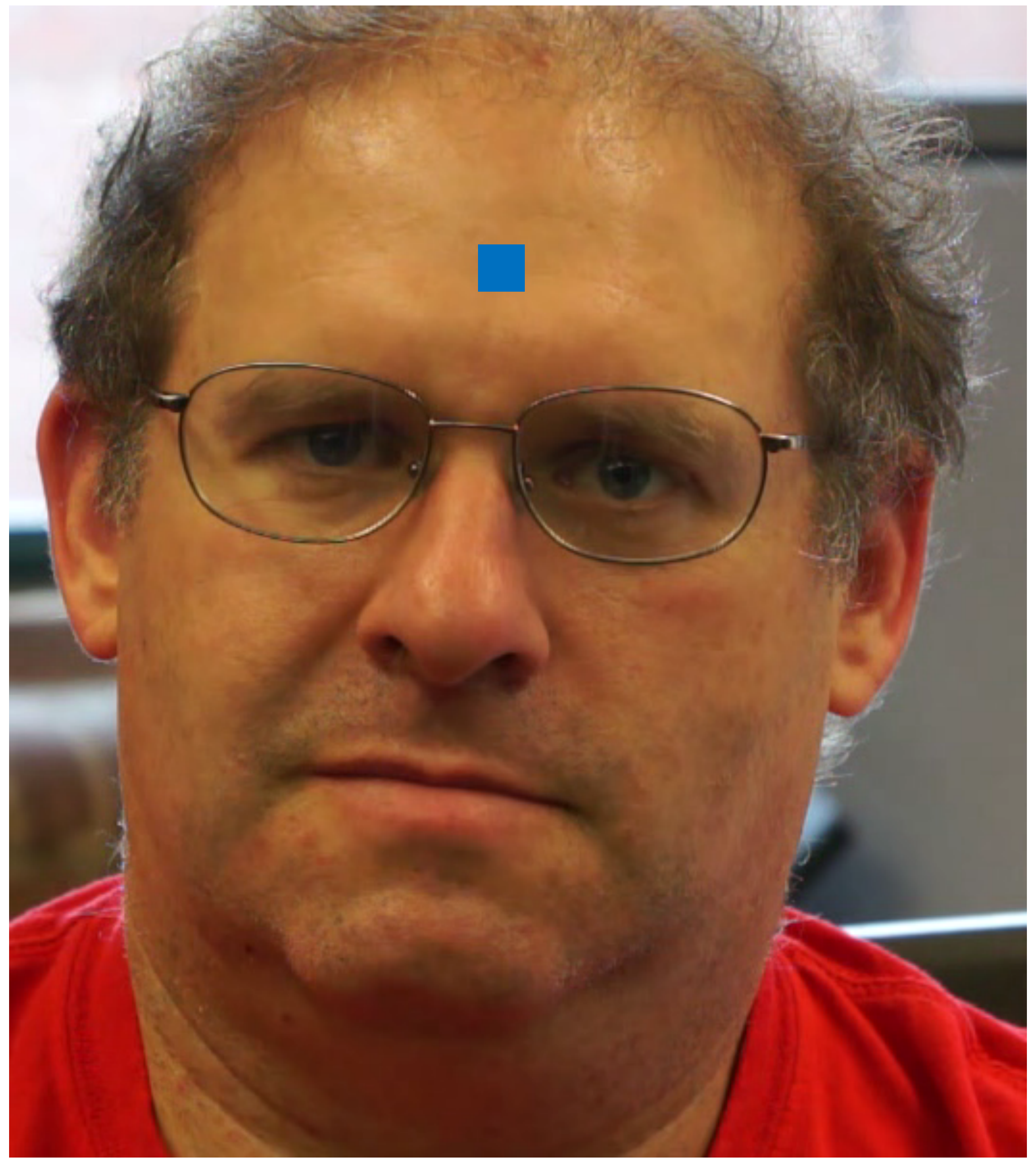
-

=

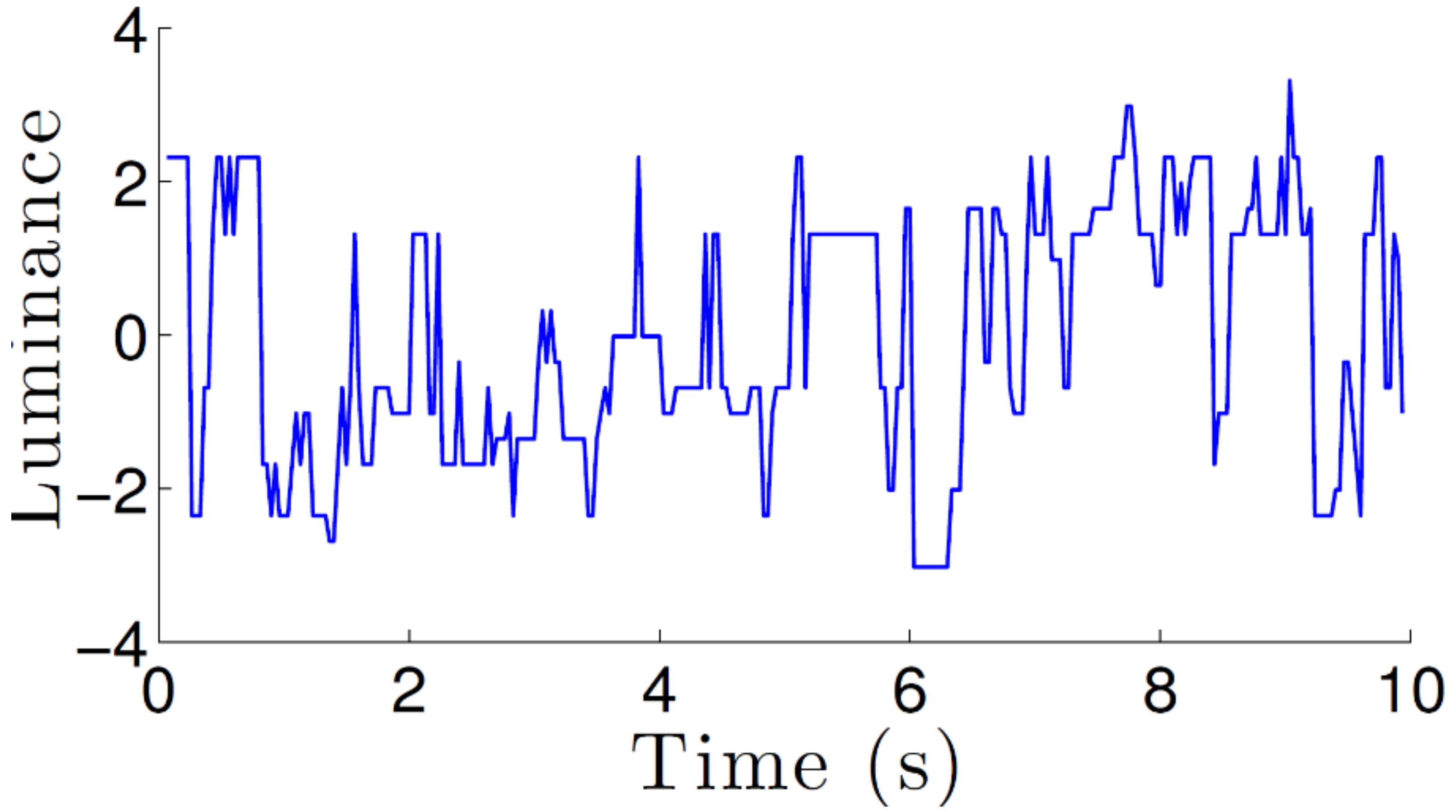
Source: Alexei Efros

Finding subtle color variations

- The face gets slightly redder when blood flows



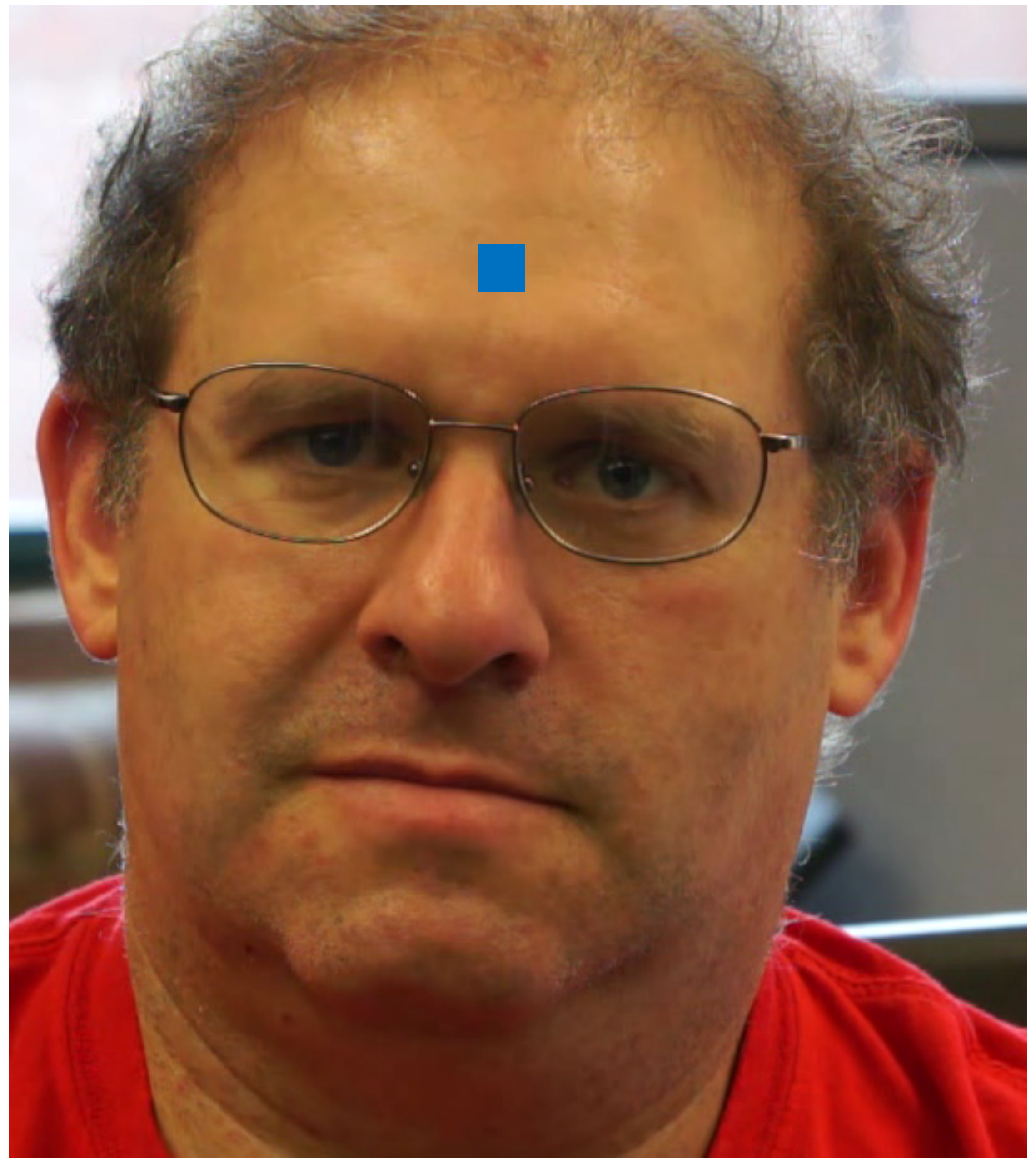
Input frame

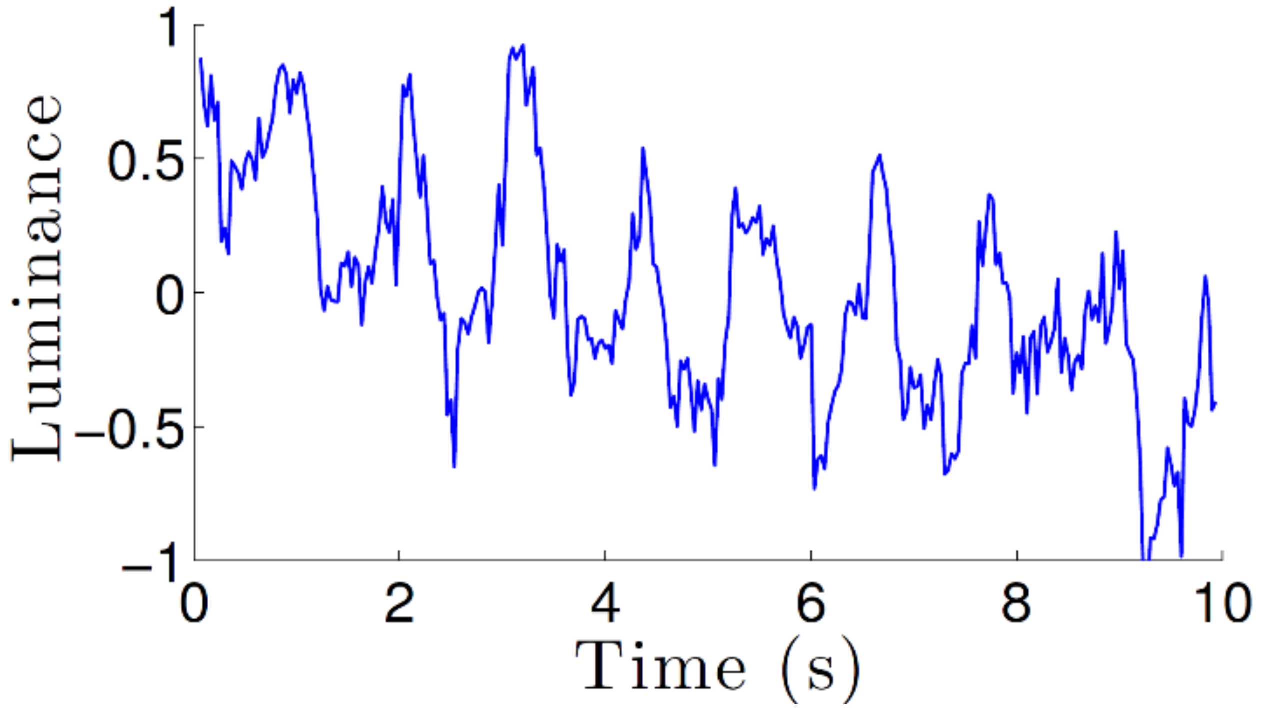


Luminance trace (zero mean)

Source: D. Hoiem

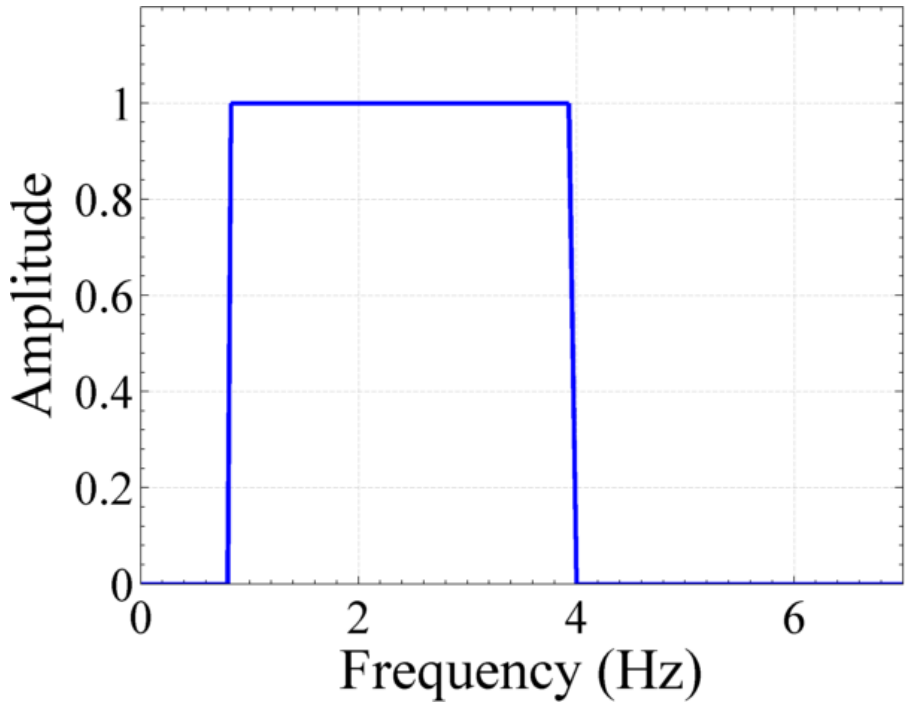
Amplifying Subtle Color Variations





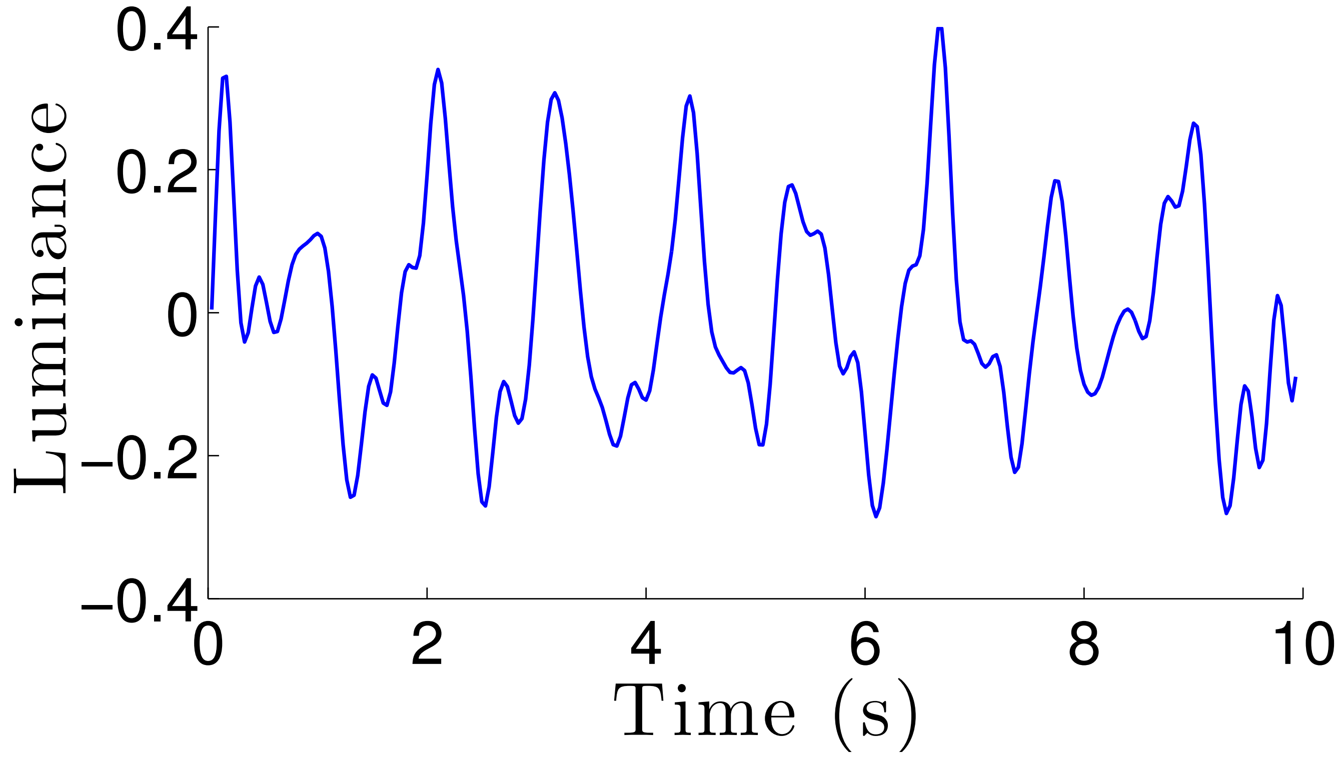
Spatially averaged luminance

$\otimes$



Temporal filter

=

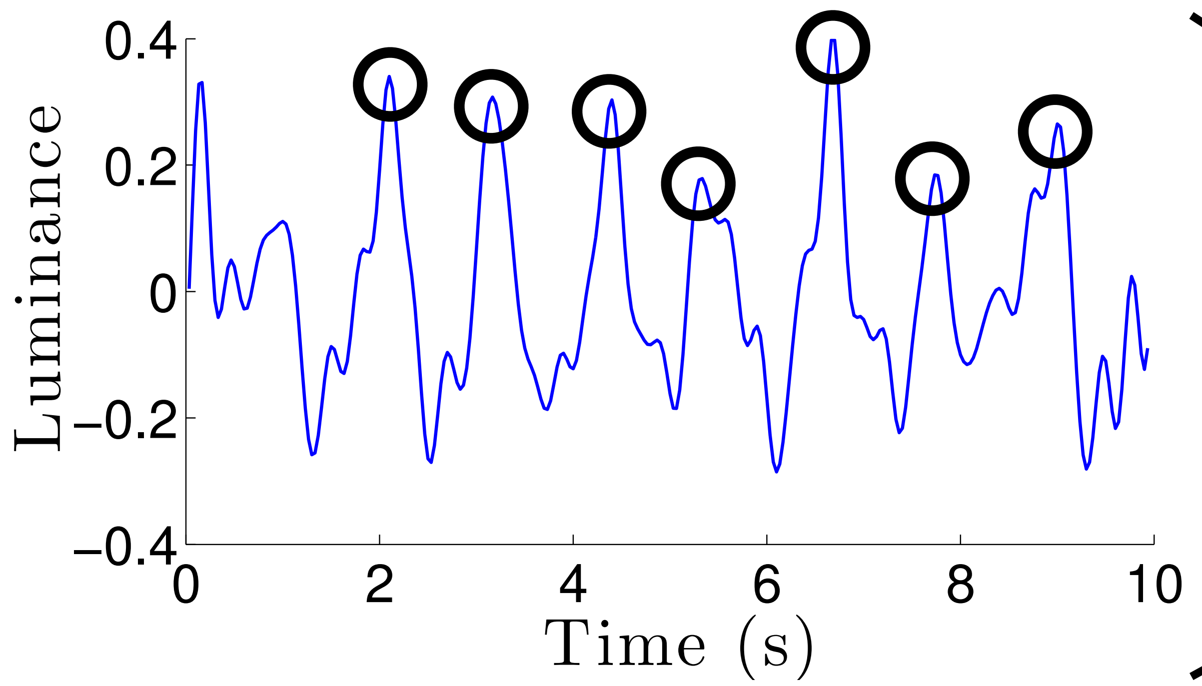


Result

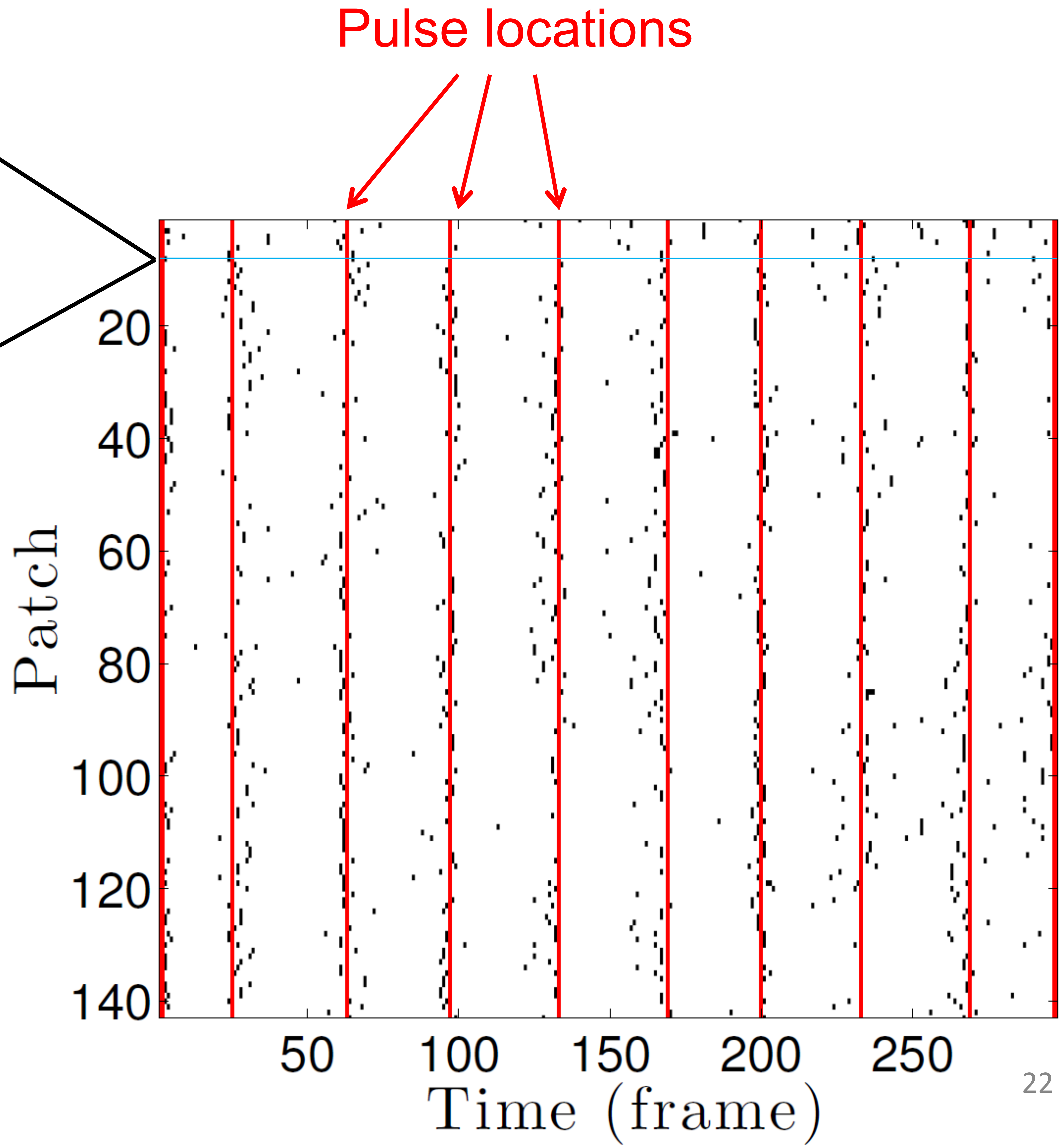
Source: D. Hoiem

Heart Rate Extraction

Peak detection

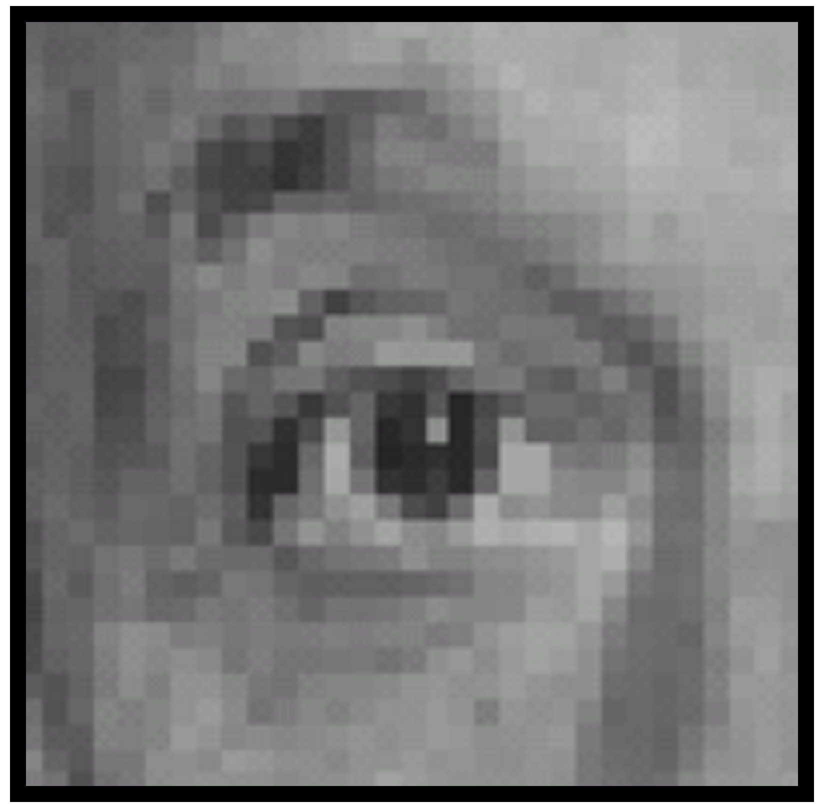


Temporally bandpassed trace
(one pixel)



Source: D. Hoiem

Recall: sharpening filter

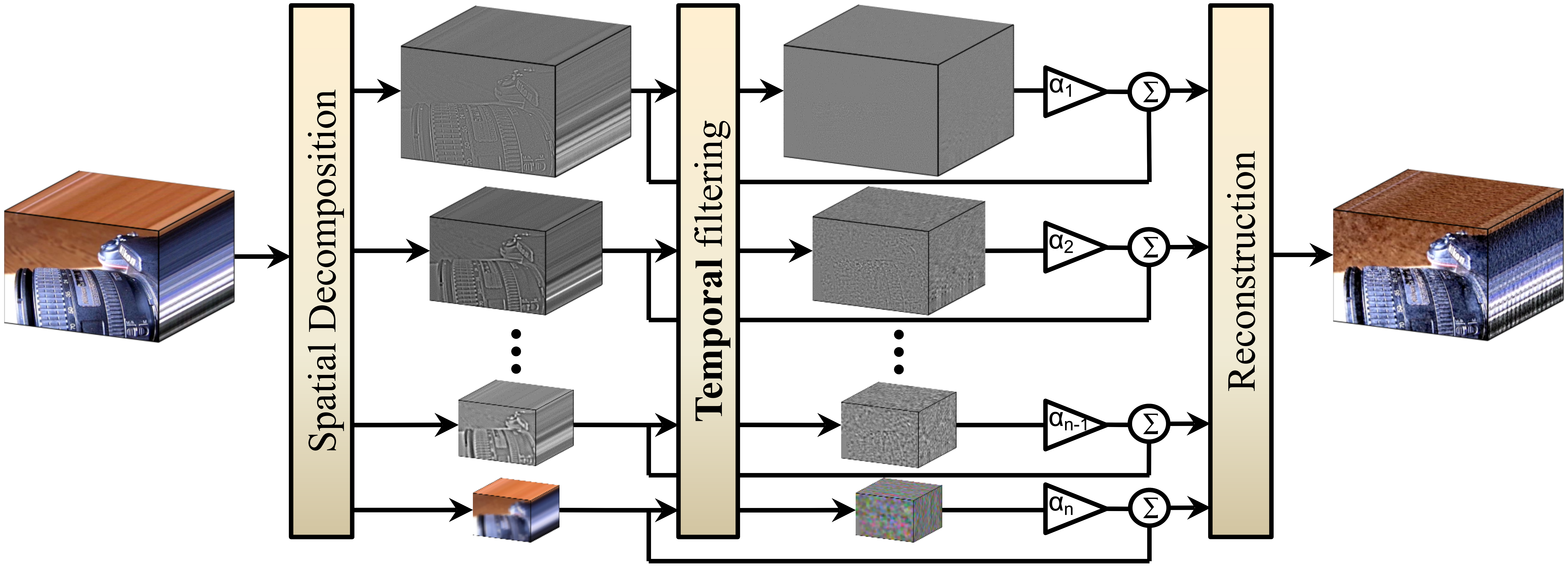


Original

$$* \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right)$$

Source: D. Lowe

Magnifying tiny motions



Laplacian Pyramid

Bandpass filter intensity at each pixel over time

Amplify bandpassed signal and add back to original

Source: D. Hoiem

Color amplification

Source: [Wadhwa et al., Phase-based Video Motion Processing. SIGGRAPH 2013, Wu et al. SIGGRAPH 2012]

Motion magnification

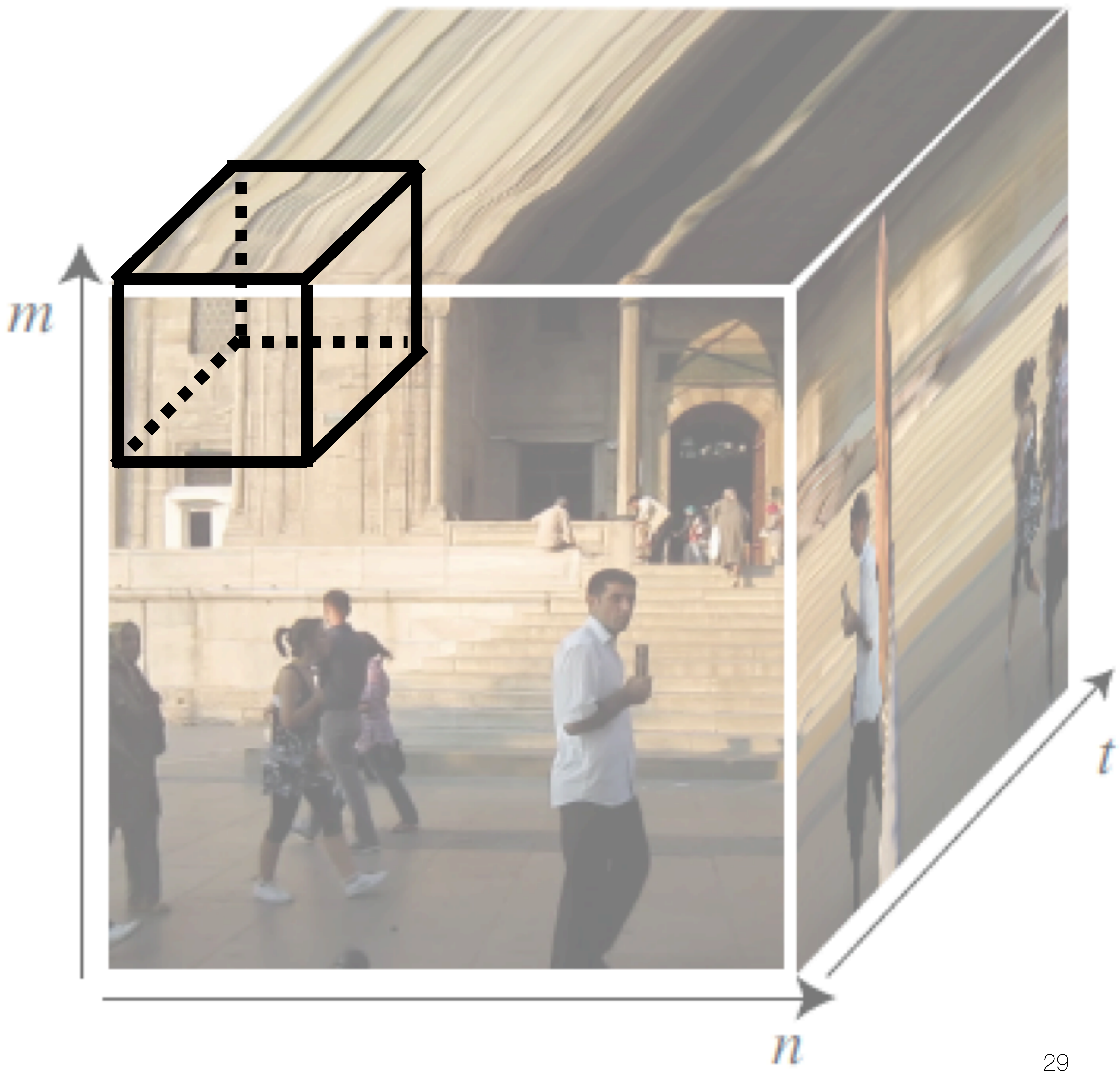
Original

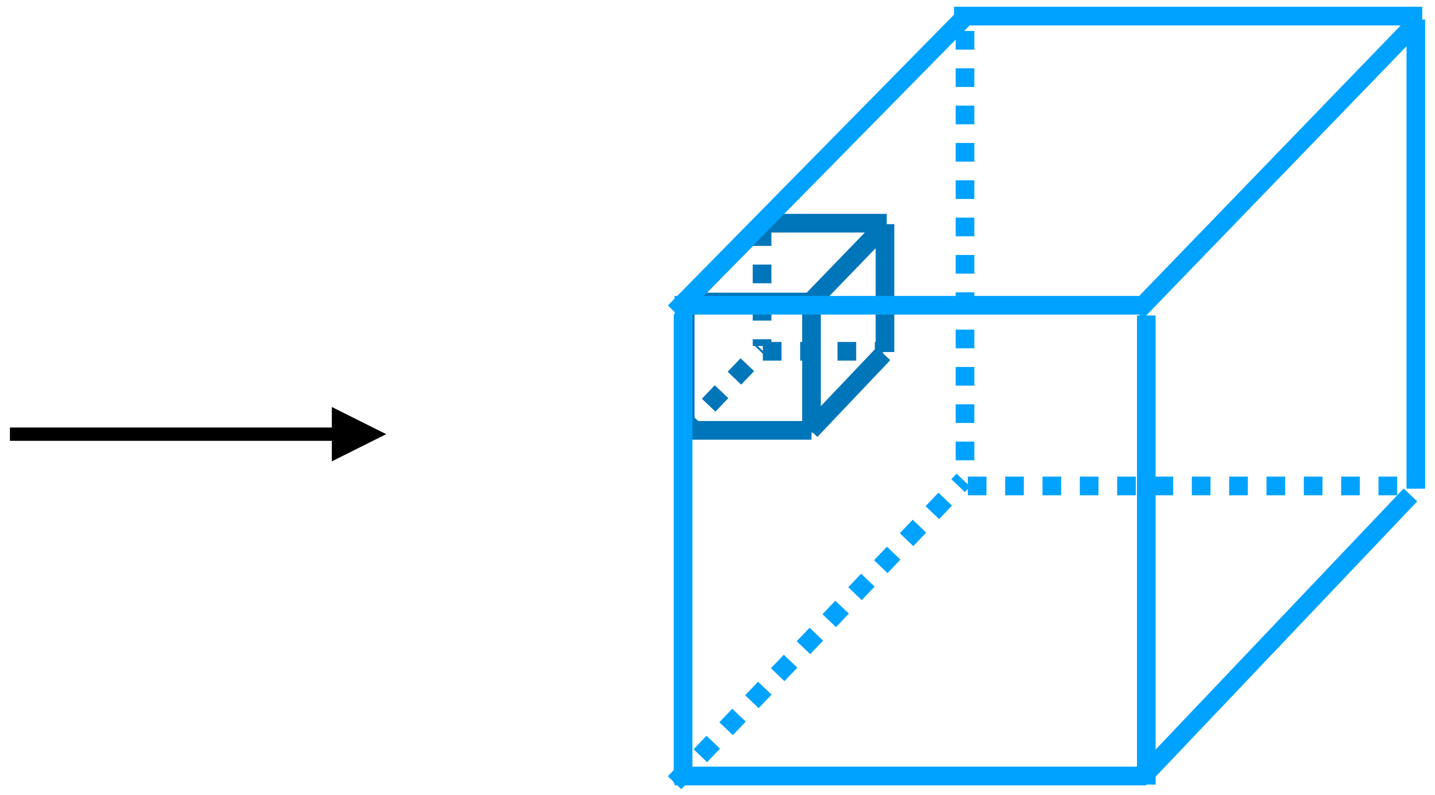
[Wu et al., SIGGRAPH 2012]

[Wadhwa et al., SIGGRAPH 2013]

Space-time convolutions

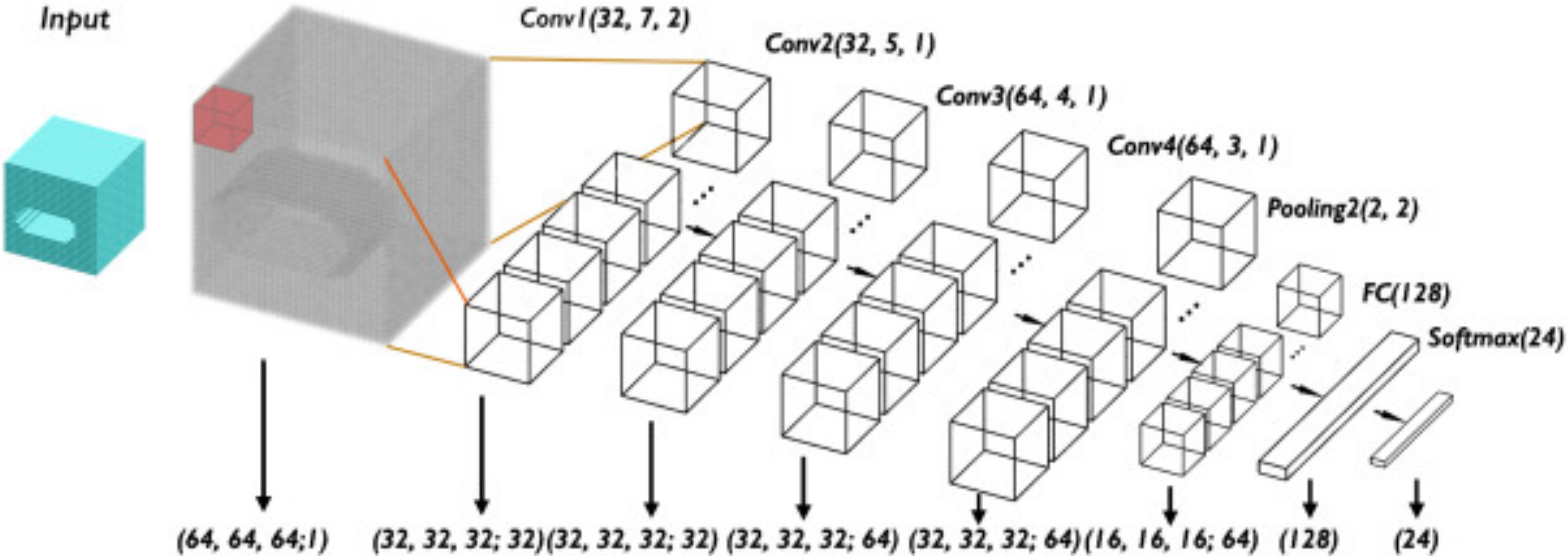
3D space-time convolution





Source: Torralba, Freeman, Isola

3D space-time CNN

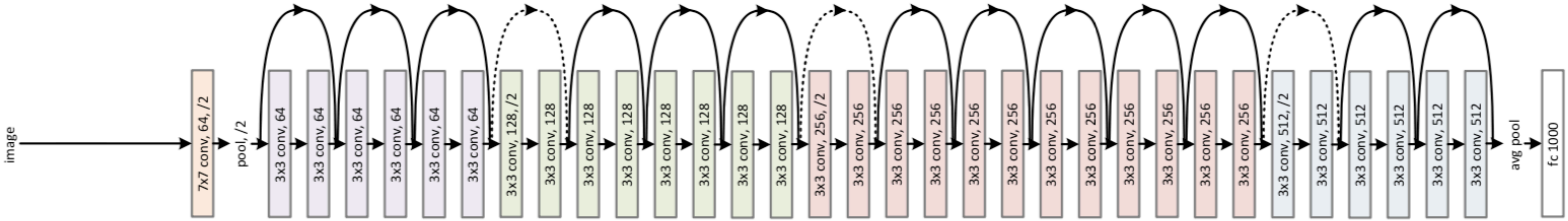


[Source: FeatureNet: Machining feature recognition based on 3D Convolution Neural Network]

Source: Torralba, Freeman, Isola

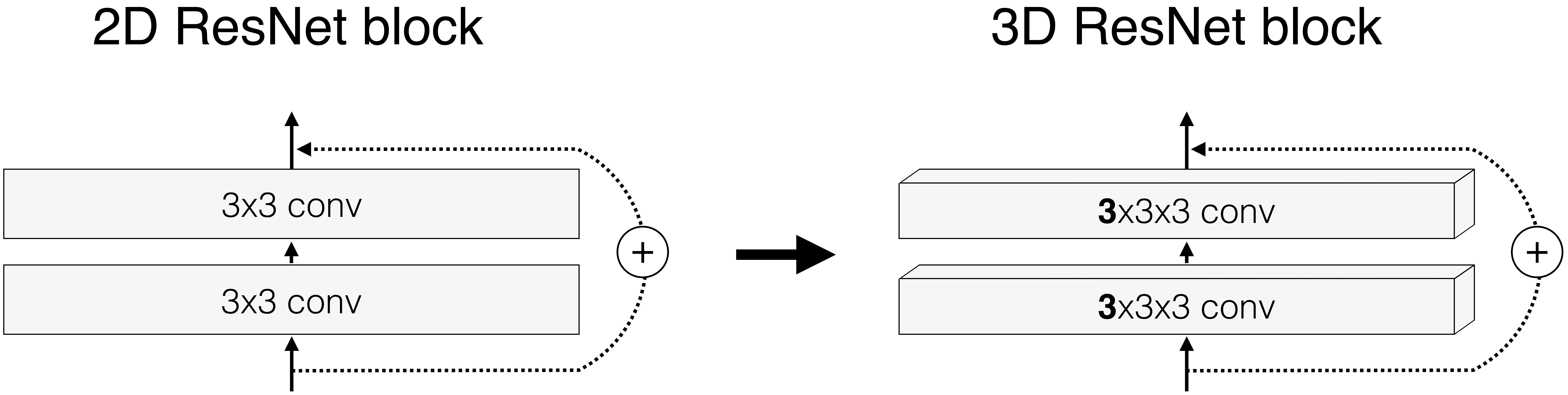
Designing a 3D CNN architecture

Starting point: 2D image CNNs



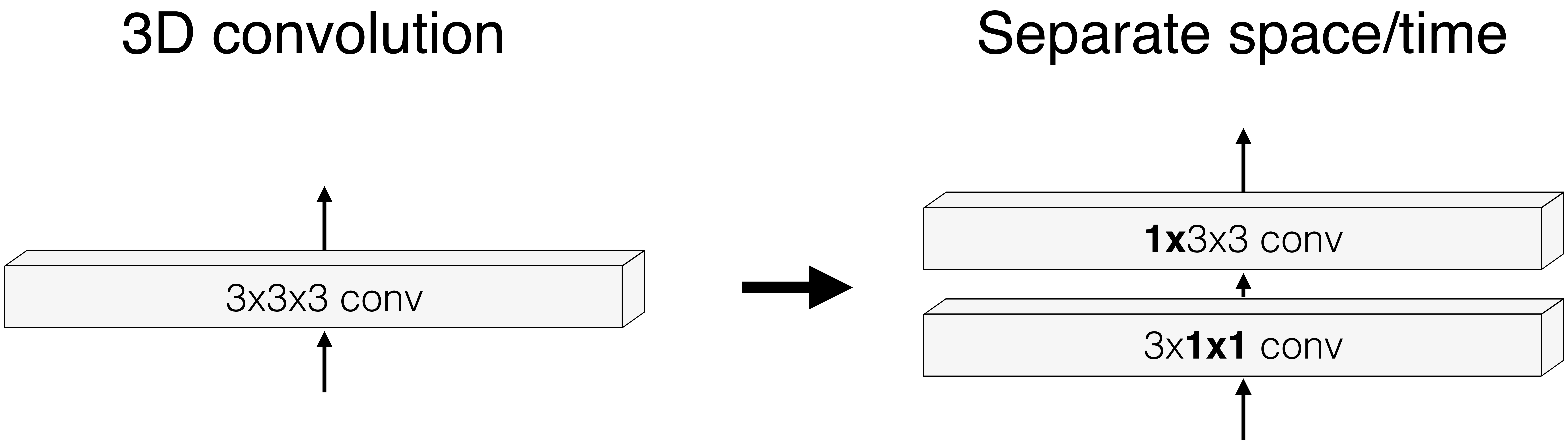
ResNet [Kaiming He et al. 2016]

Inflated convolutions



- Can reuse 2D architectures. [Carreira et al. 2017]
- Pretrain with 2D nets (“inflating” 2D filter to 3D)

Separable convolutions



Often works well. Faster and fewer parameters.

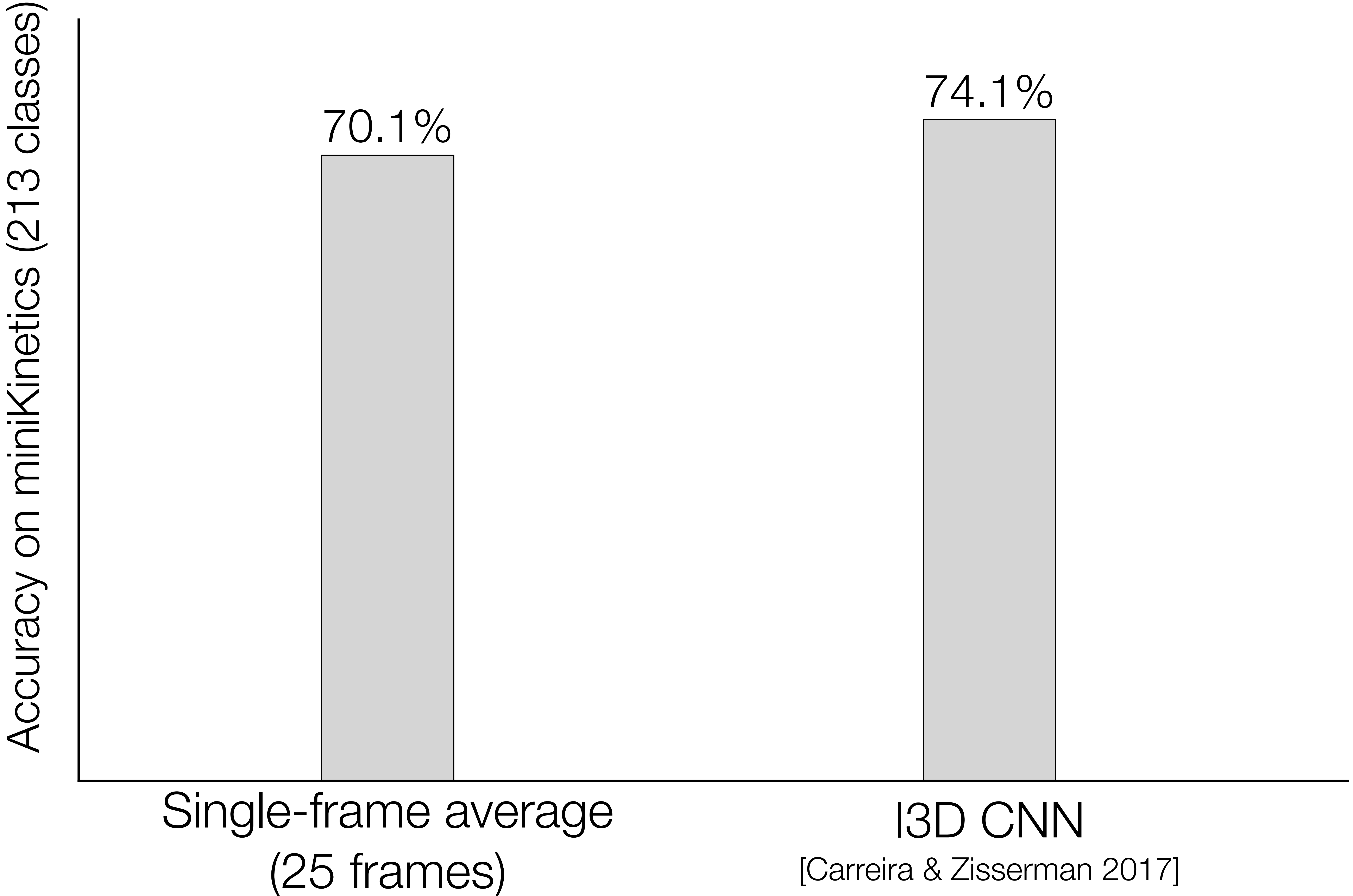
$$3 \times 3 \times 3 = 27 \qquad \text{vs.} \qquad 3 \times 3 + 3 = 12$$

[Tran et al., 2018], [Xie et al., 2018]

Learned space-time filters

$(7 \times 7 \times 7)$ I3D conv1 filters, [Carreira & Zisserman 2017]

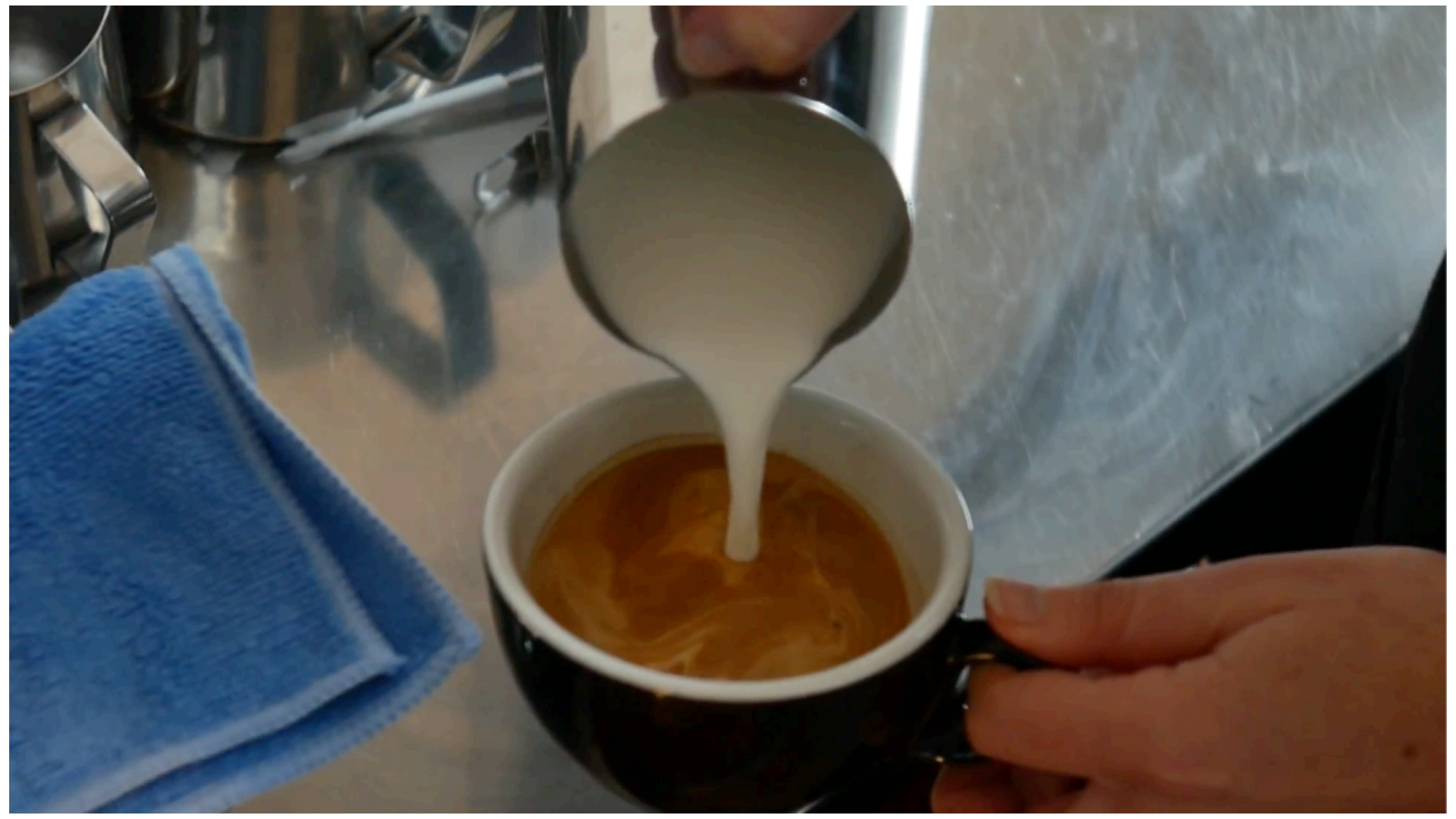
When do we actually need motion?



Adapted from David Fouhey

When do we actually need motion?

Let’s look at these again:



Making latte art

Jaywalking

Grooming dog

When do we actually need motion?

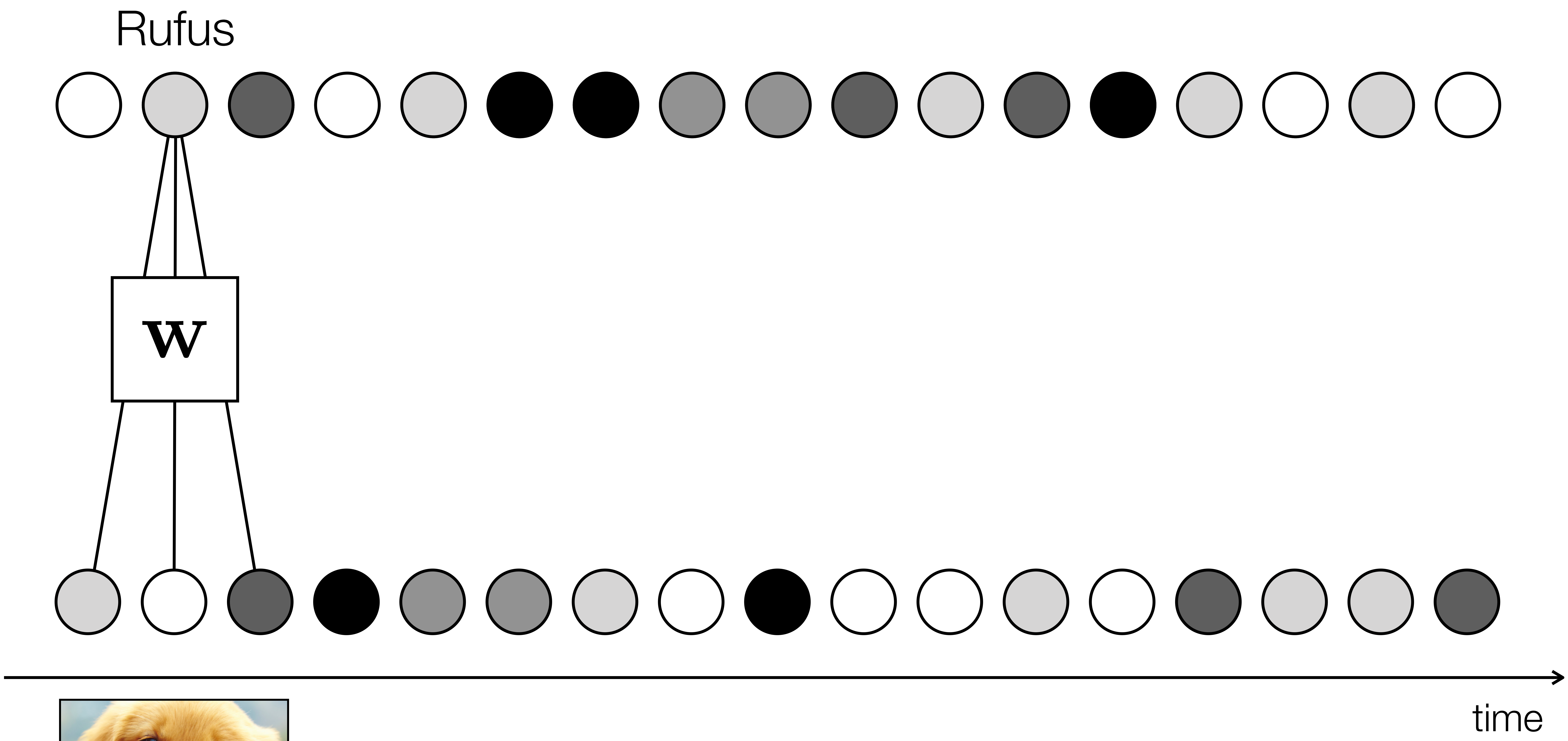
Photo by H. Edgerton

- Single-frame models usually don't work well.
- But single-image model + temporal pooling often is surprisingly competitive.
- In many tasks that use video, time often provides **extra samples**, rather than motion.
- Later in the course, we'll see tasks where motion is essential, such as 3D reconstruction.

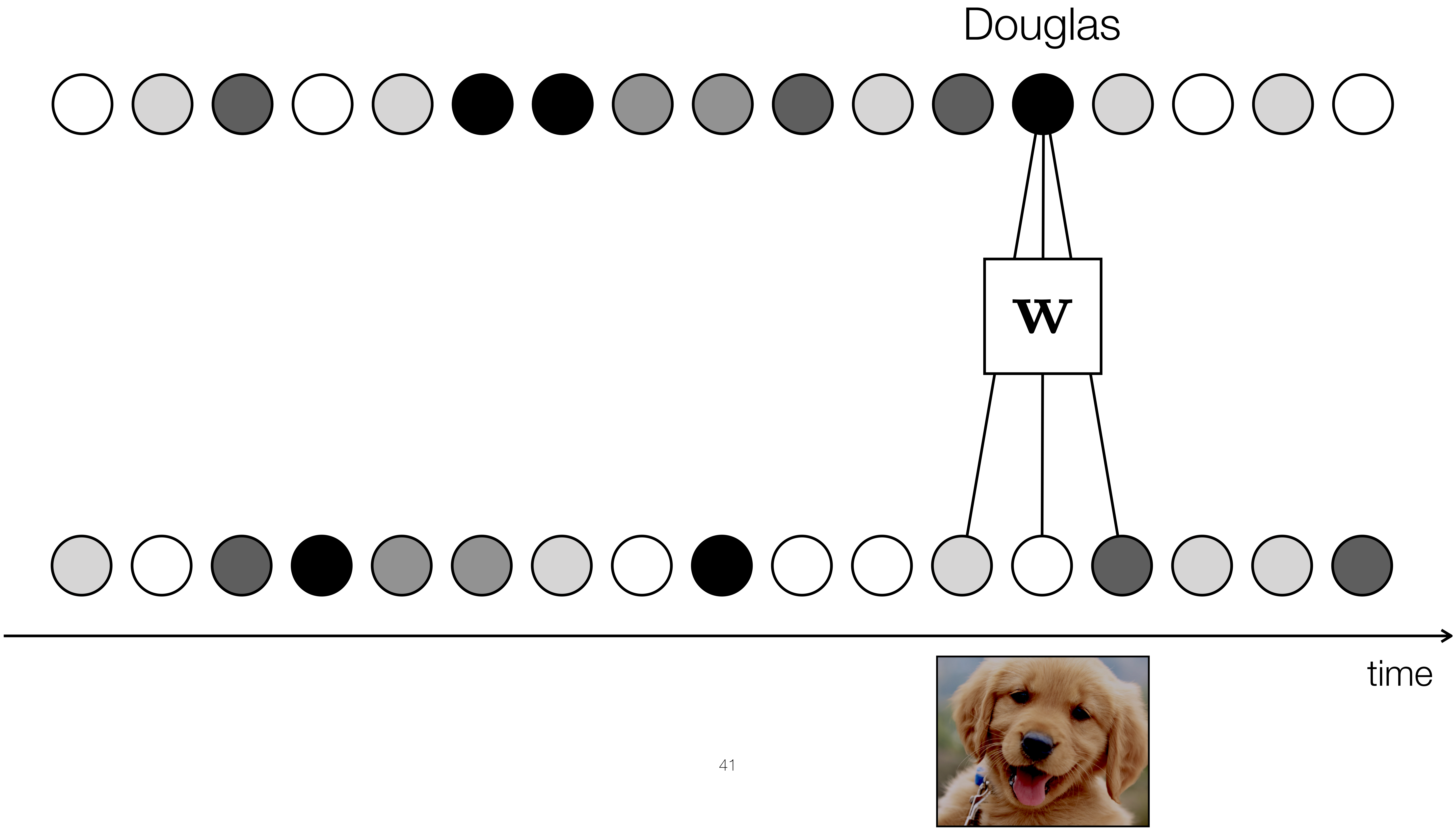
Recurrent networks

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

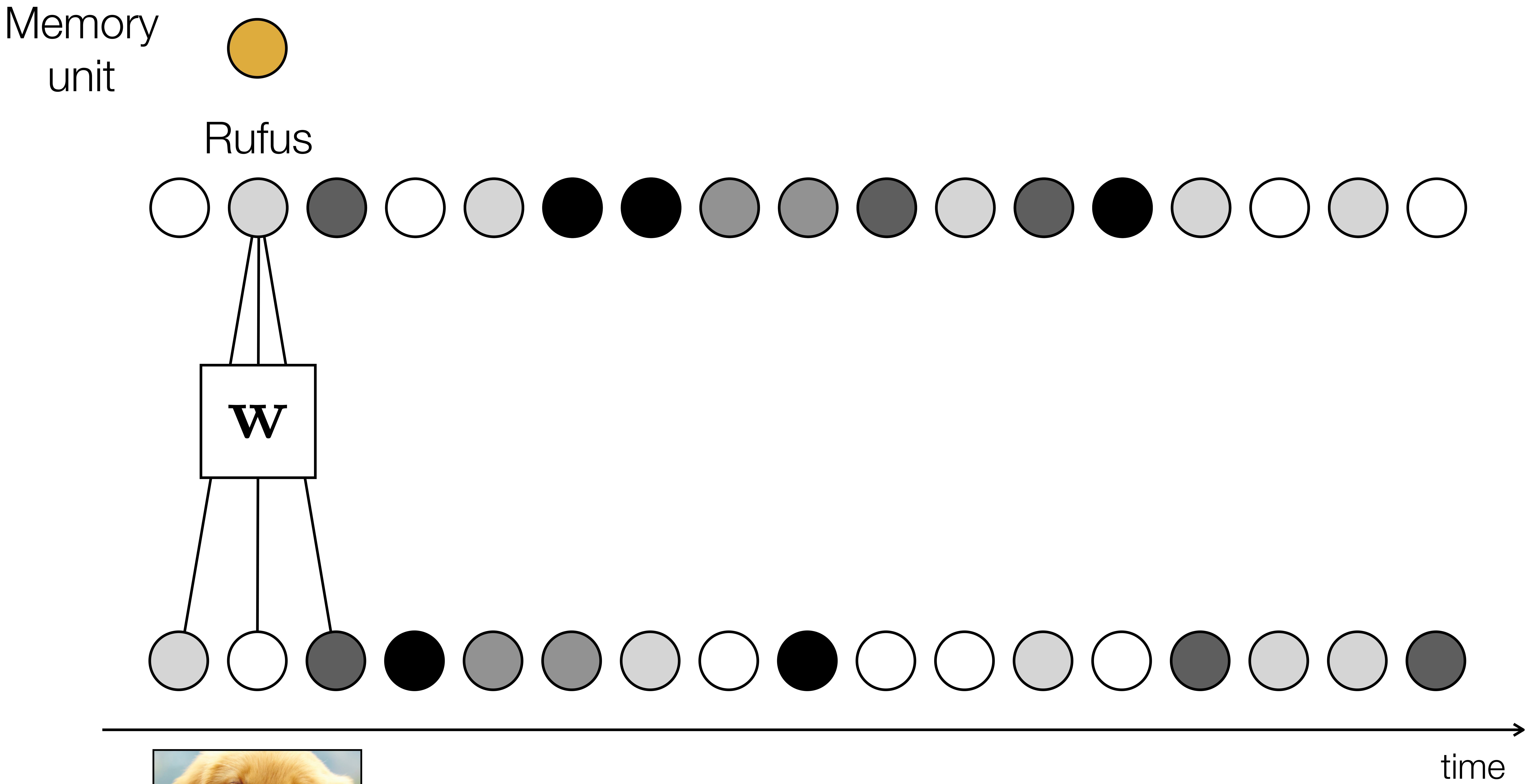
Source: Torralba, Freeman, Isola



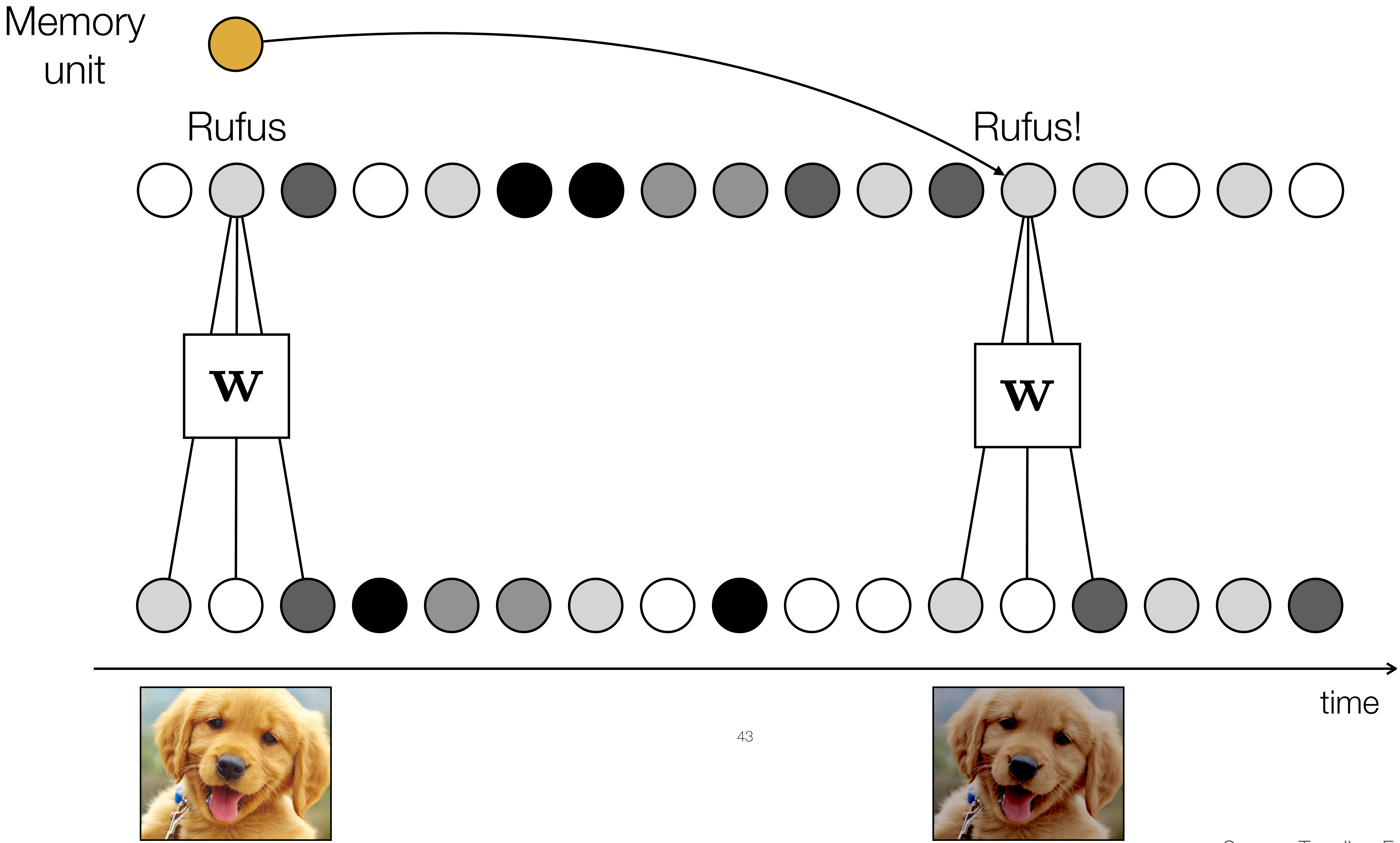
Source: Torralba, Freeman, Isola



Source: Torralba, Freeman, Isola

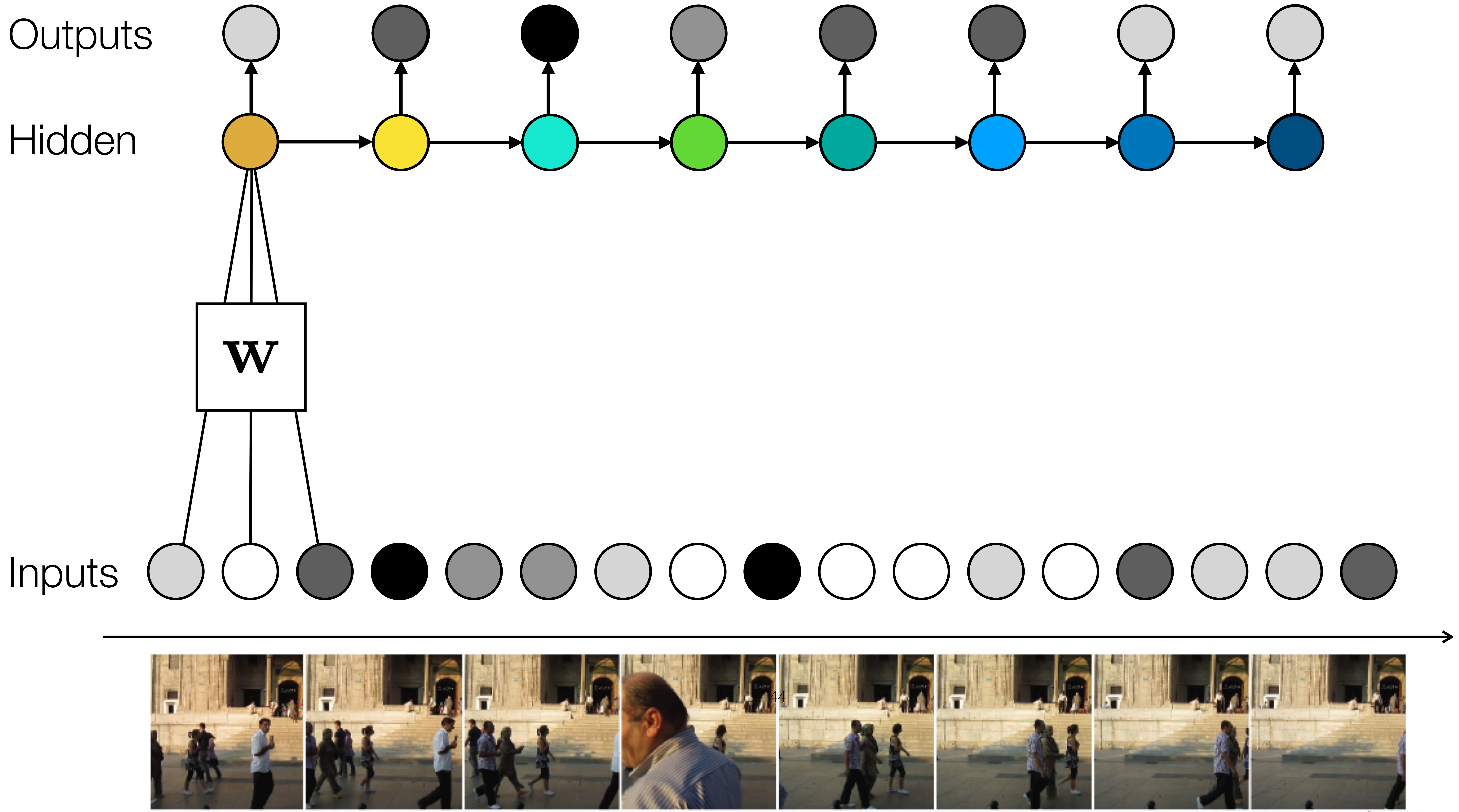


Source: Torralba, Freeman, Isola



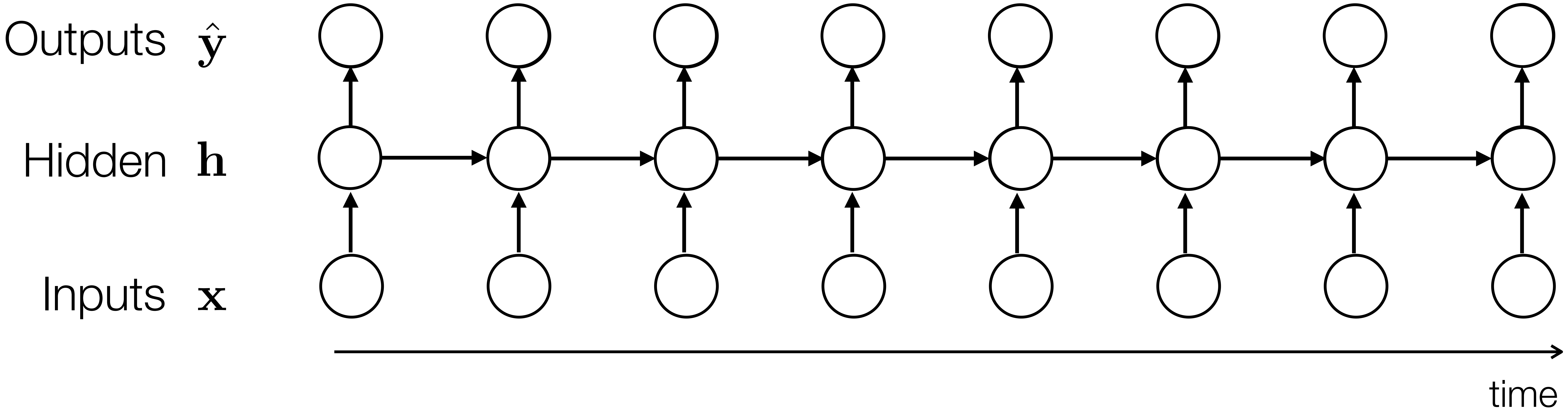
Source: Torralba, Freeman, Isola

Recurrent Neural Networks (RNNs)



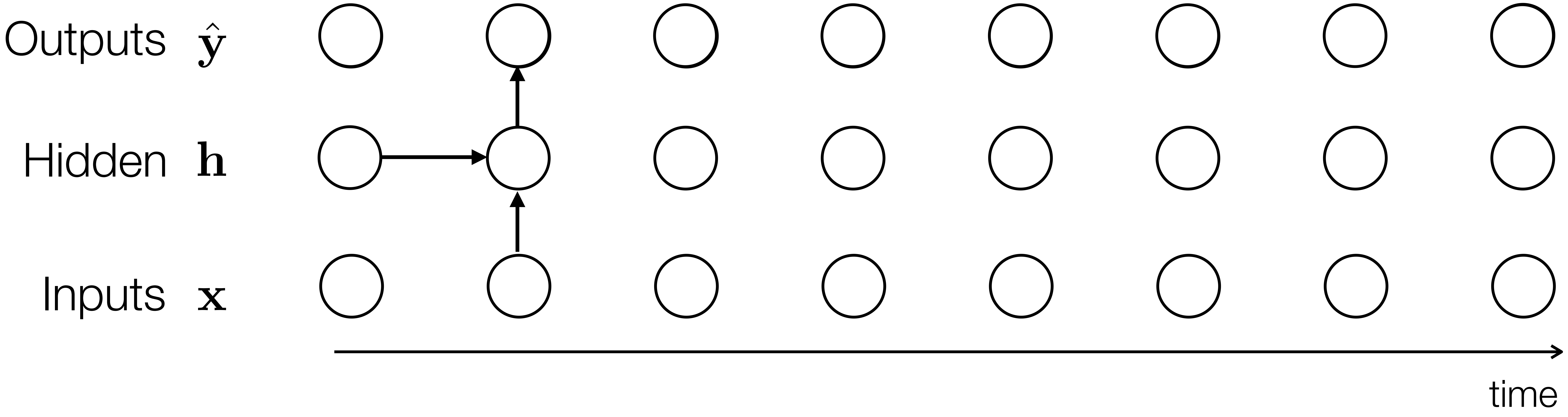
Source: Torralba, Freeman, Isola

Recurrent Neural Networks (RNNs)



Source: Torralba, Freeman, Isola

Recurrent Neural Networks (RNNs)

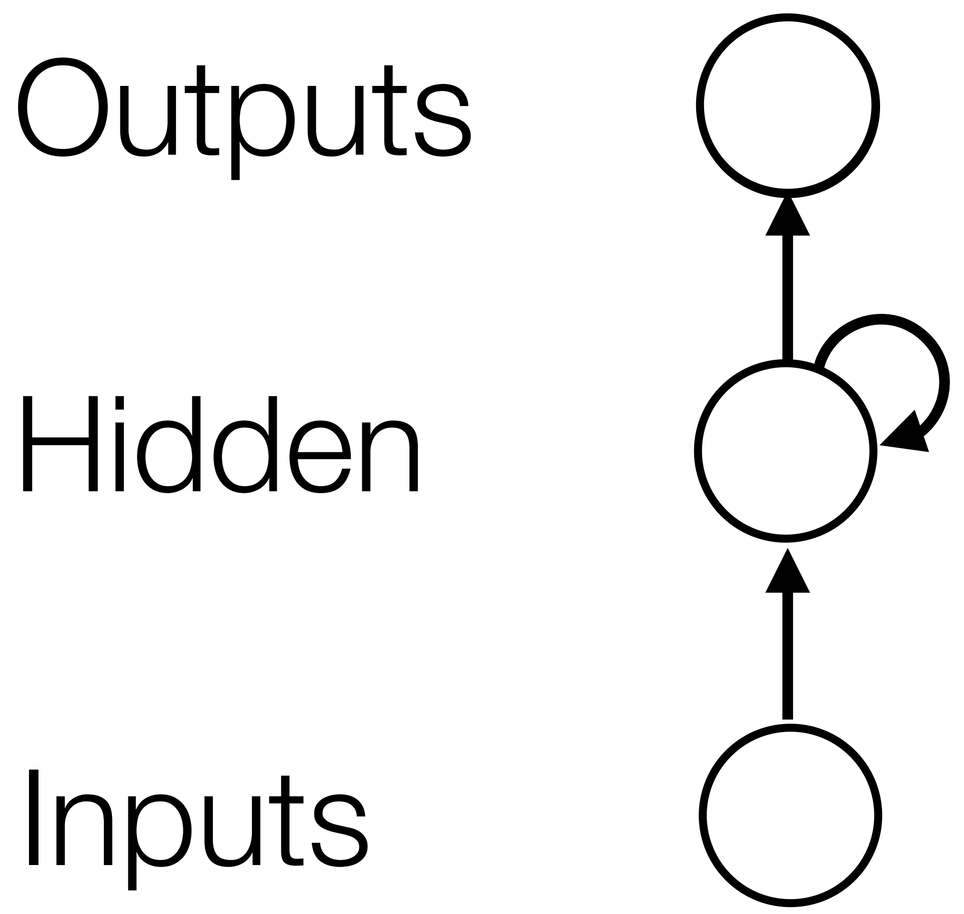


$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)})$$

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

Source: Torralba, Freeman, Isola

Recurrent Neural Networks (RNNs)



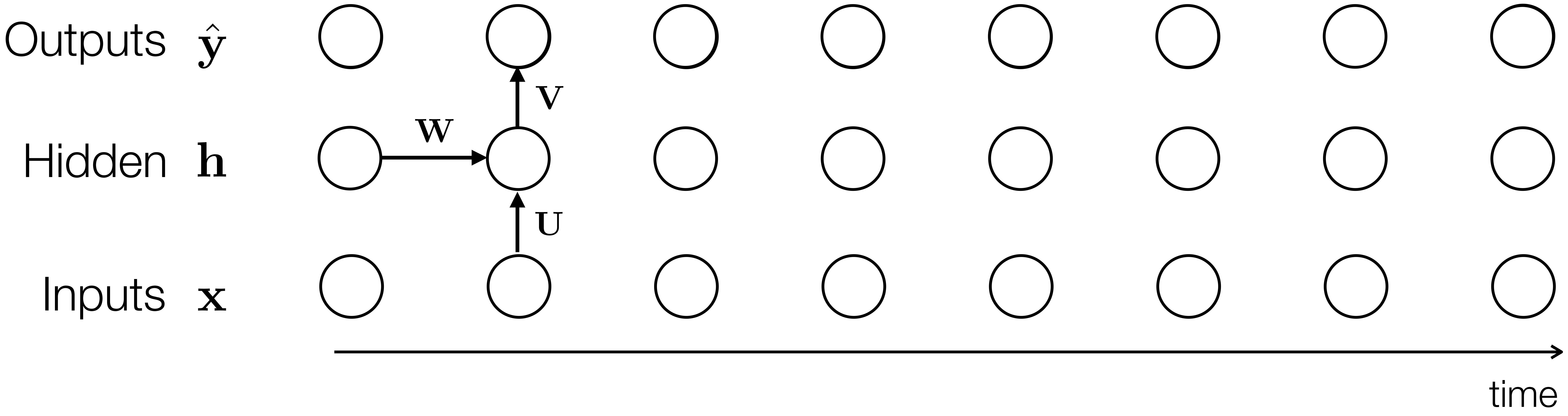
Recurrent!

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

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

Source: Torralba, Freeman, Isola

Recurrent Neural Networks (RNNs)



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

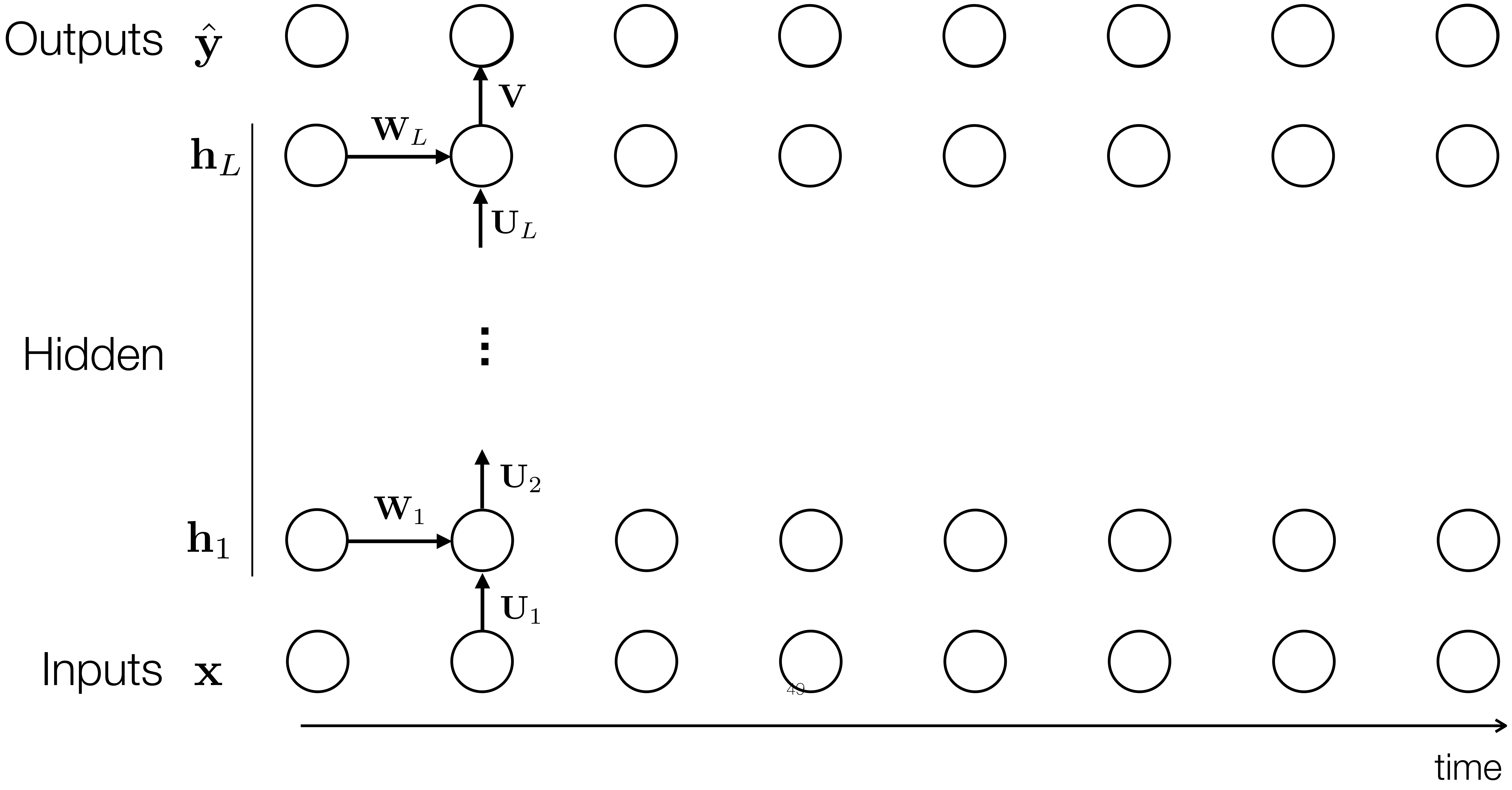
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$

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

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

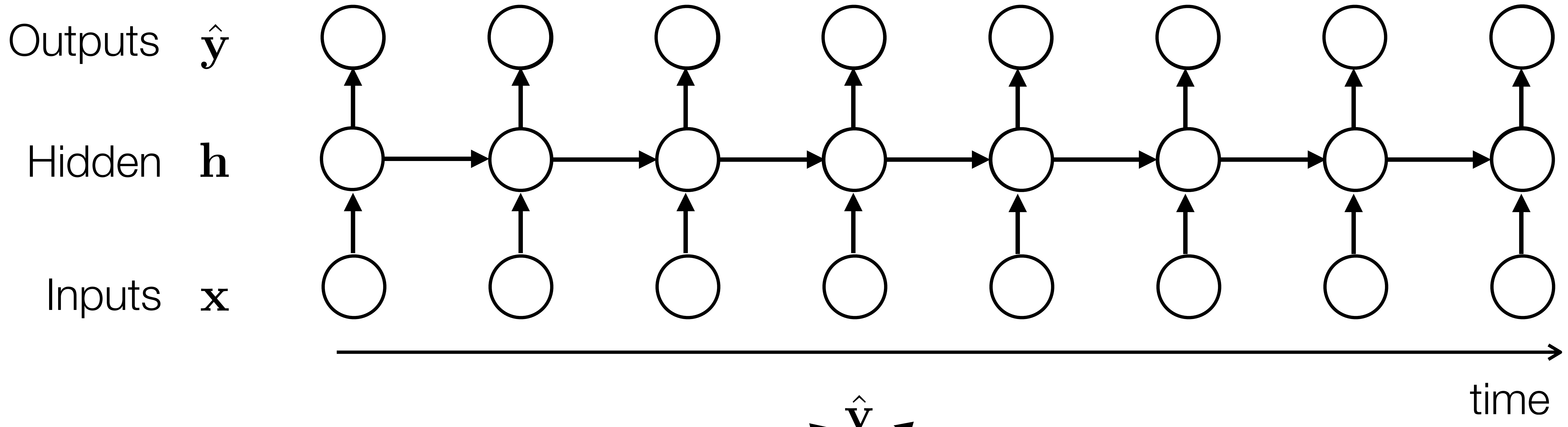
Source: Torralba, Freeman, Isola

Deep Recurrent Neural Networks (RNNs)

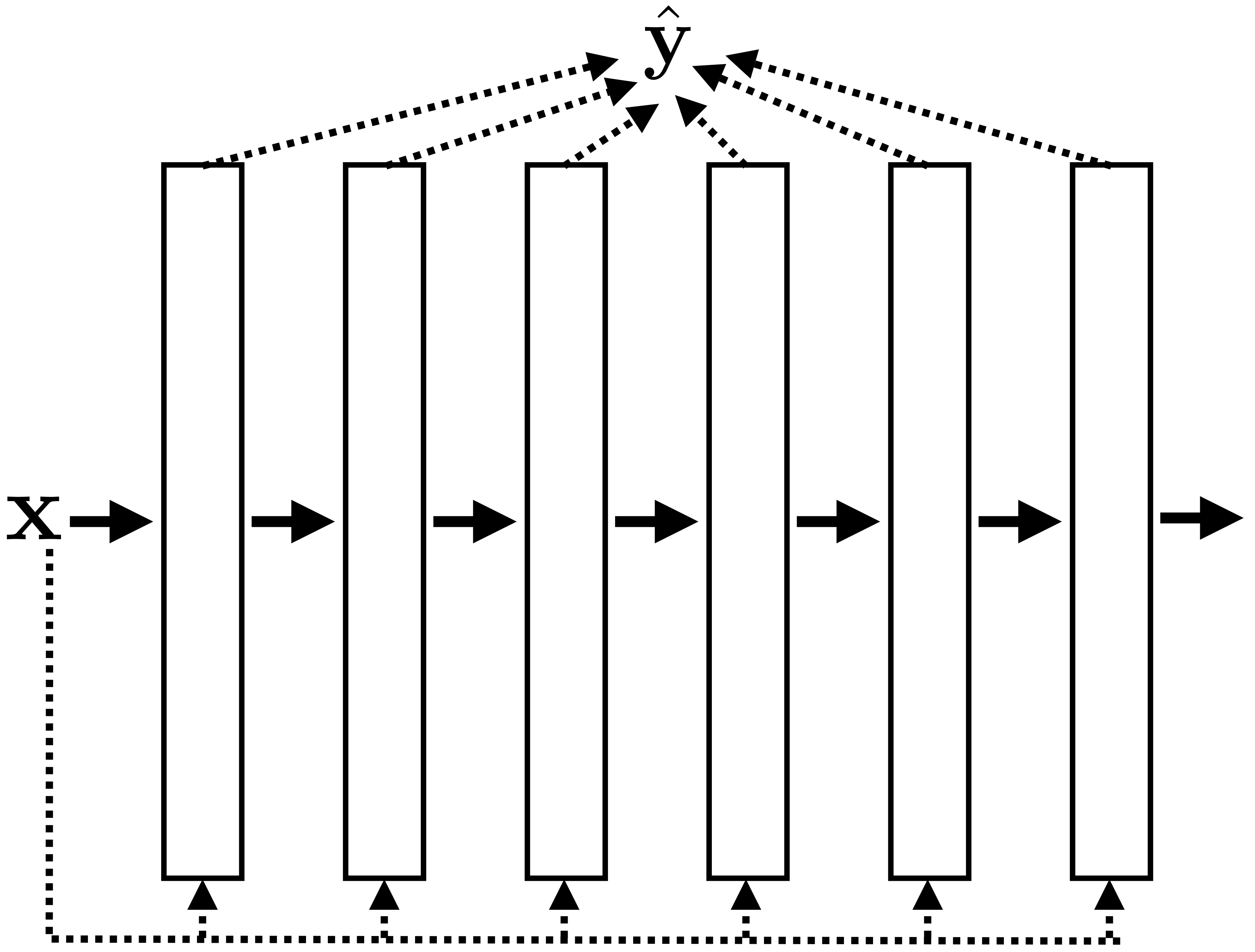


Source: Torralba, Freeman, Isola

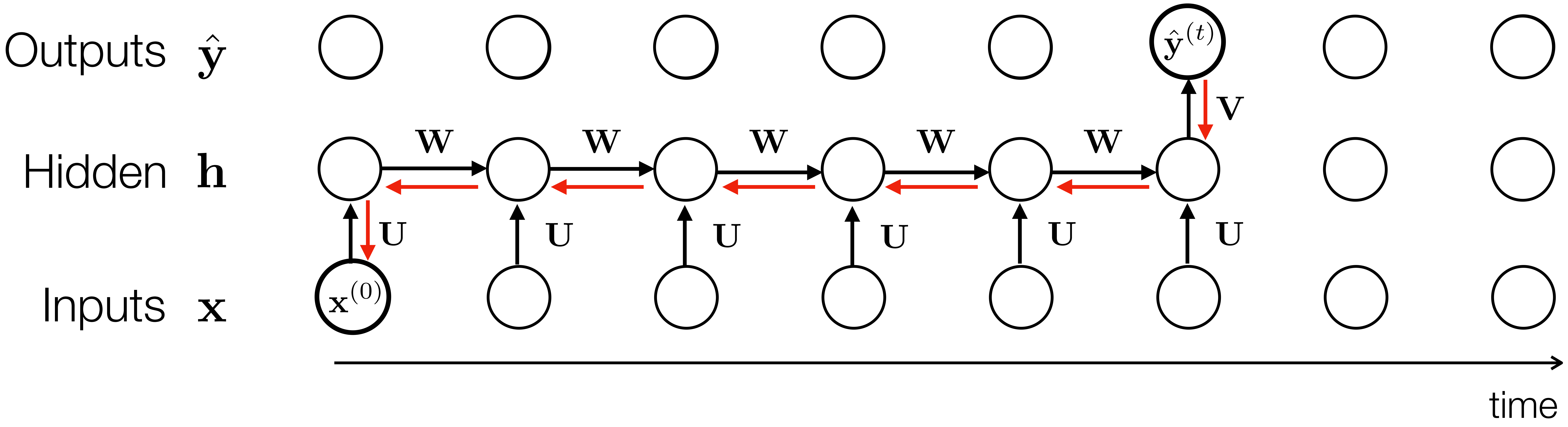
Unrolling an RNN



Equivalent to
"unrolled" network
with shared weights



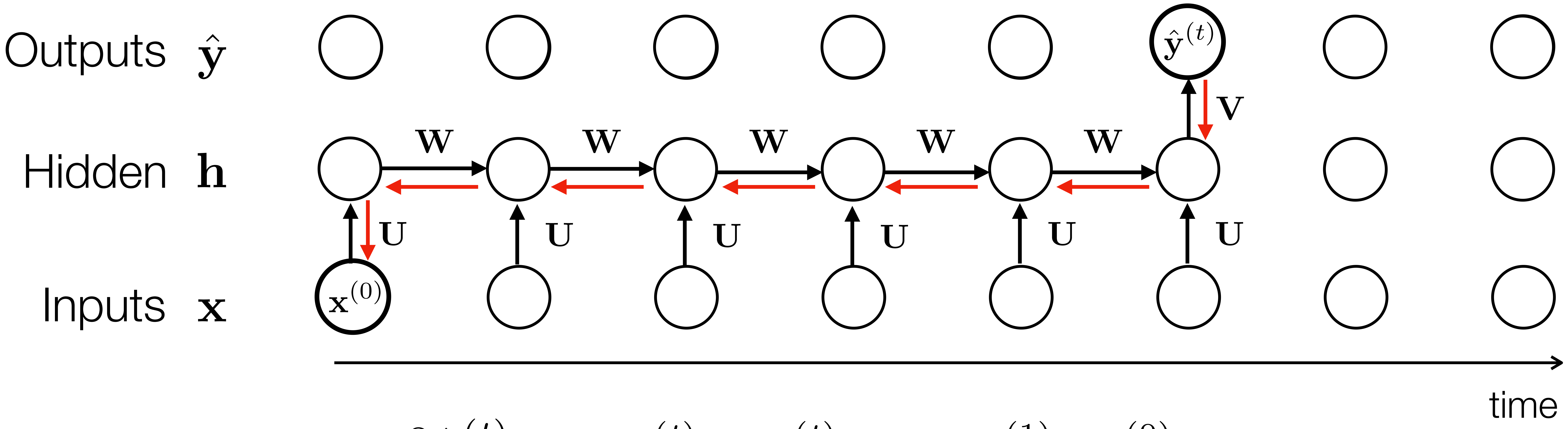
Backprop in RNNs



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

Source: Torralba, Freeman, Isola

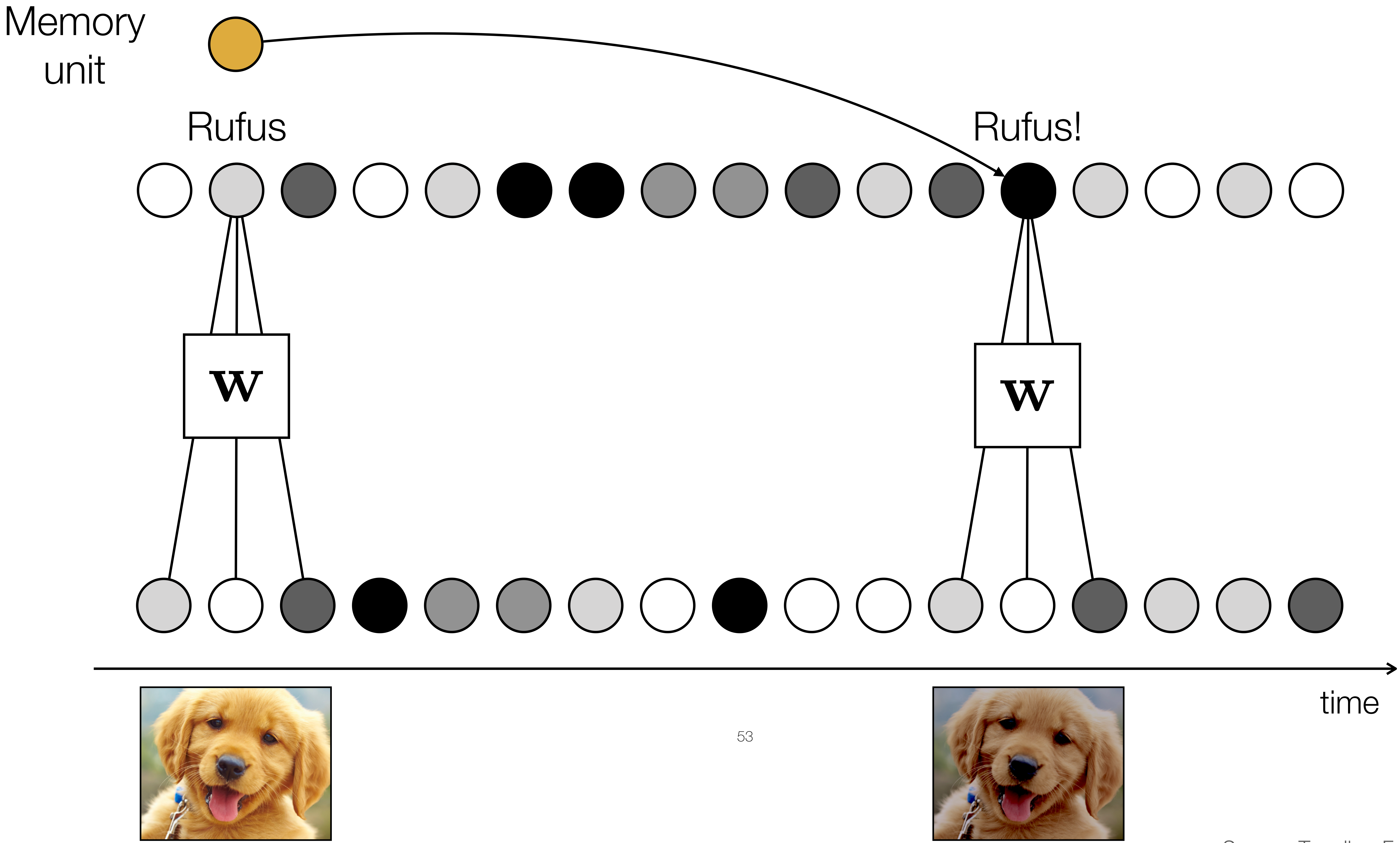
The problem of long-range dependences



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

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

Source: Torralba, Freeman, Isola



Source: Torralba, Freeman, Isola

LSTMs

Long Short Term Memory

- A special kind of RNN designed to avoid forgetting [Hochreiter & Schmidhuber 1995].

- Related to ResNets' 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*.

- Bit of a complex design. Works well but simpler methods like Gated Recurrent Unit (GRU) are competitive [Jozefowicz et al. 2015].

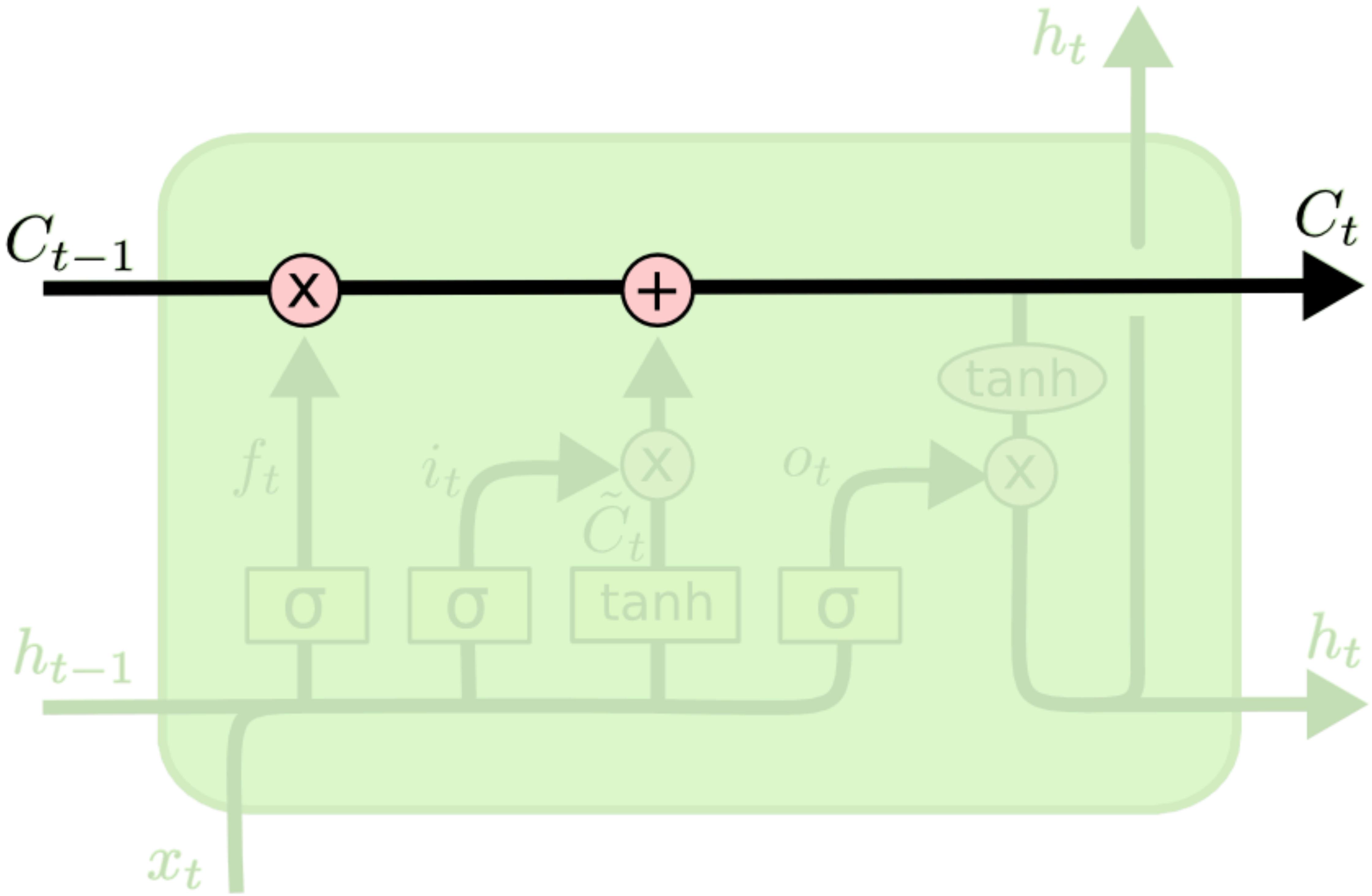
Source: Torralba, Freeman, Isola

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

Source: Torralba, Freeman, Isola

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

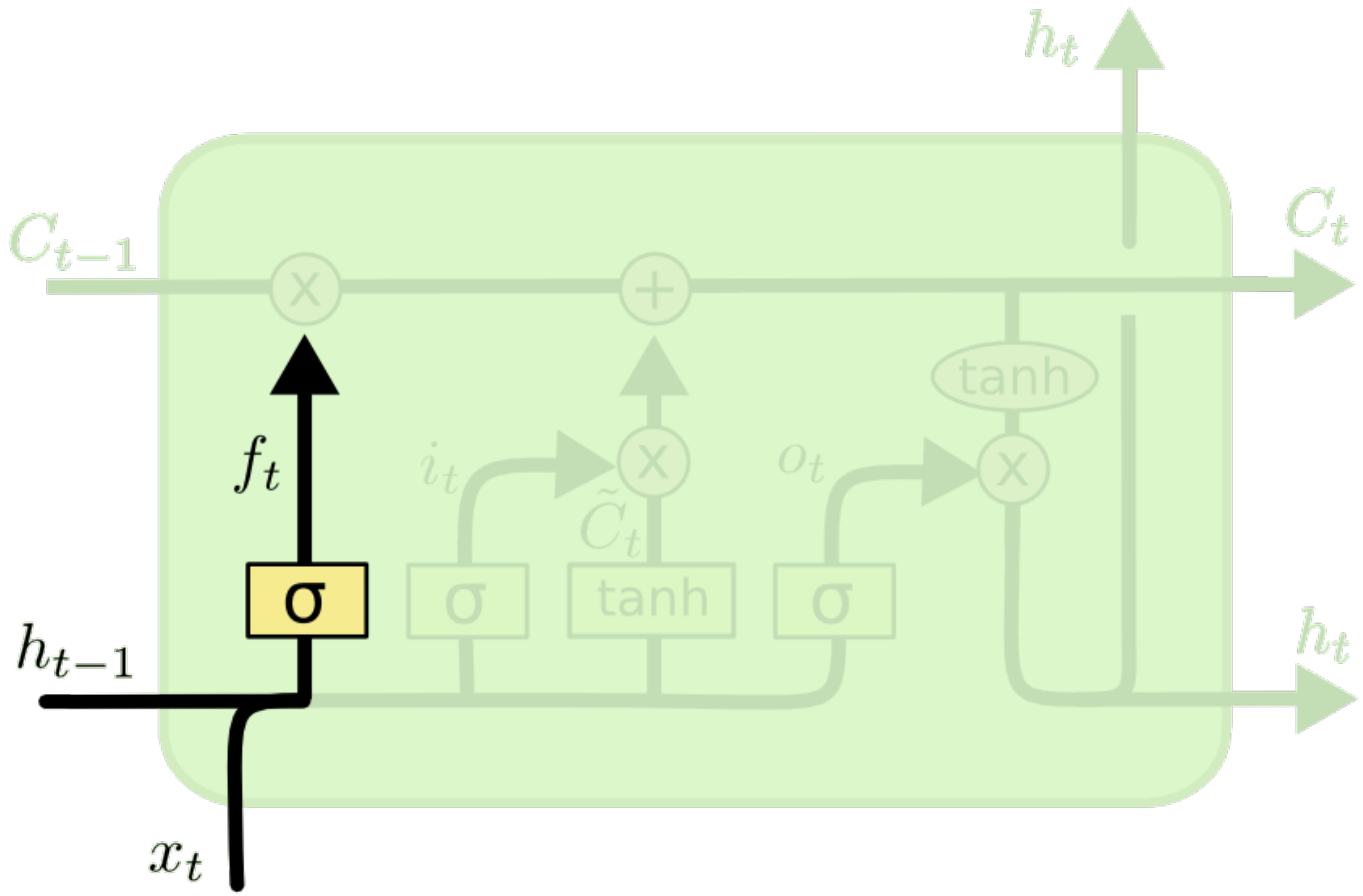
Source: Torralba, Freeman, Isola

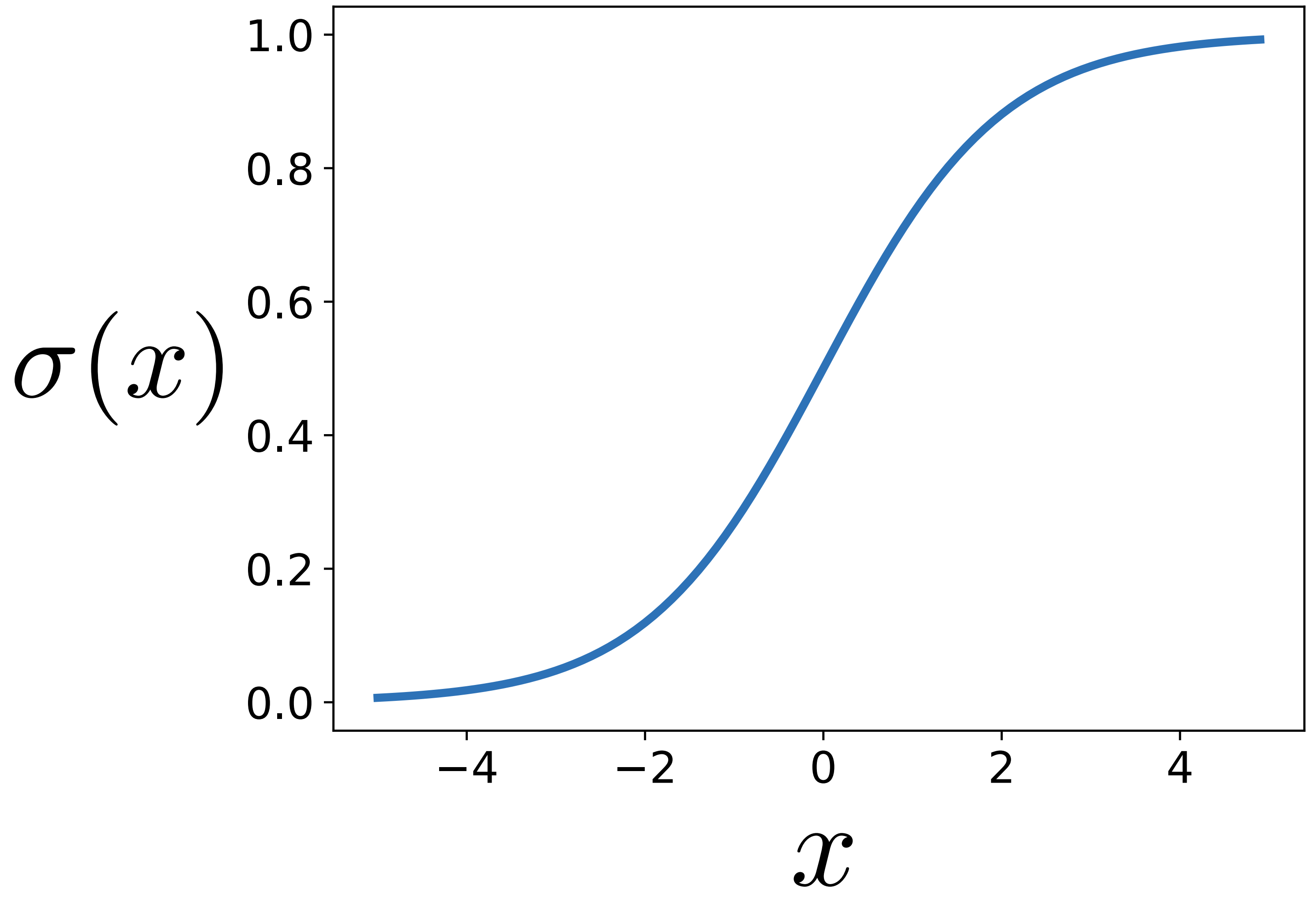


C_t = Cell state

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

Source: Torralba, Freeman, Isola





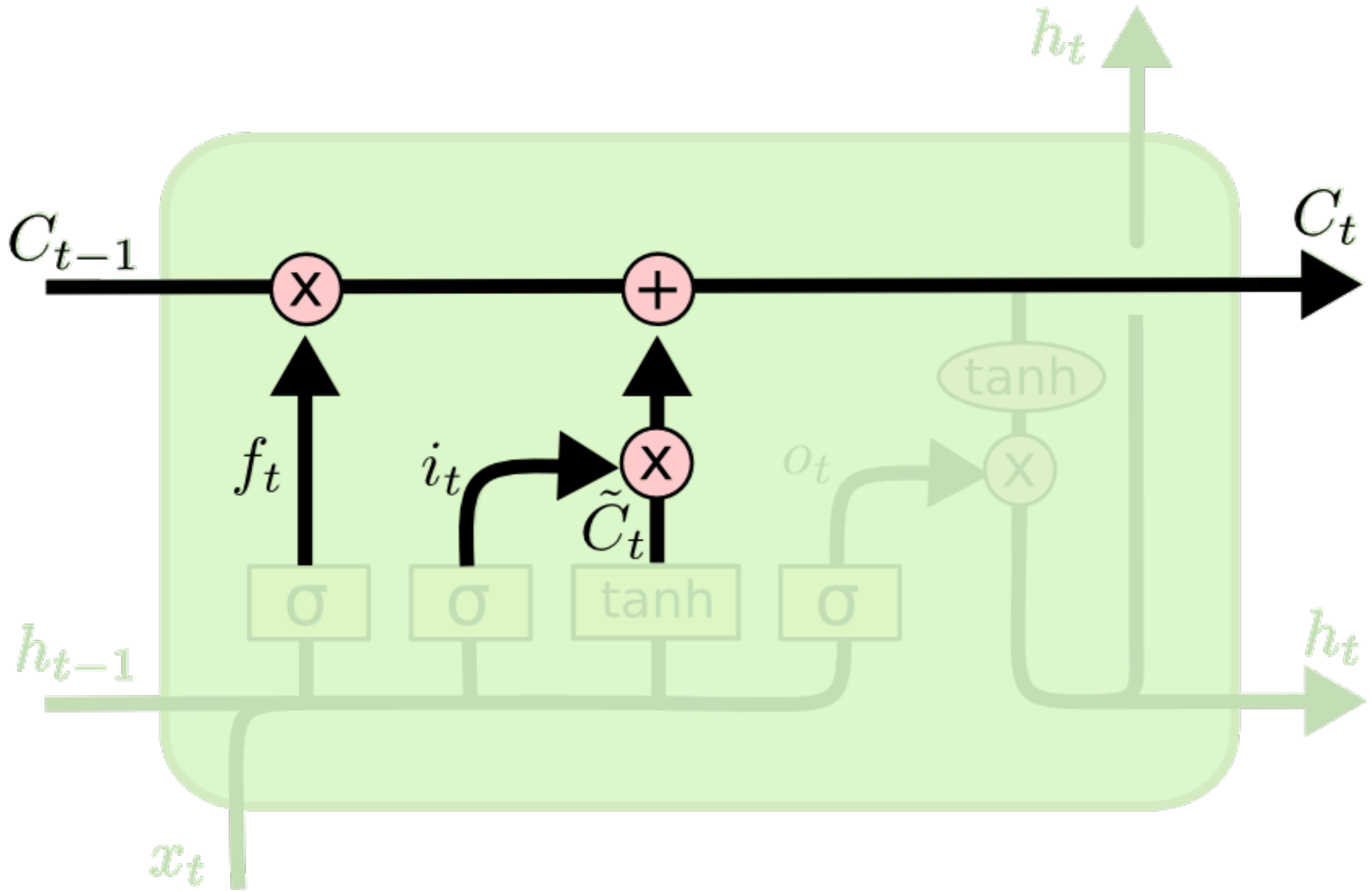
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Decide what information to throw away from the cell state.

Forget gate: 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/]

Source: Torralba, Freeman, Isola



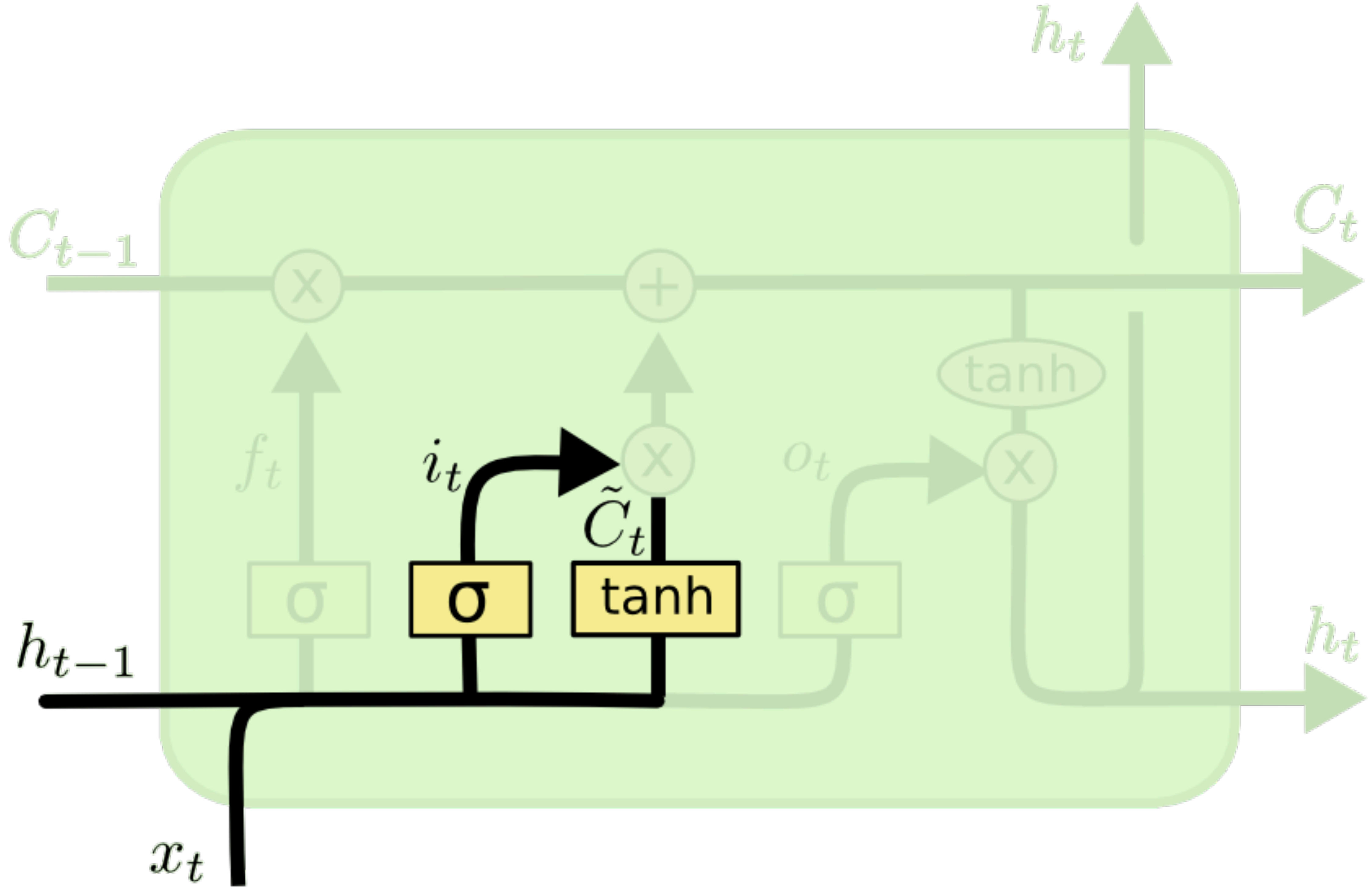
Forget   New values to write

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

Forget selected old information, write selected new information.

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

Source: Torralba, Freeman, Isola



which components to write to

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

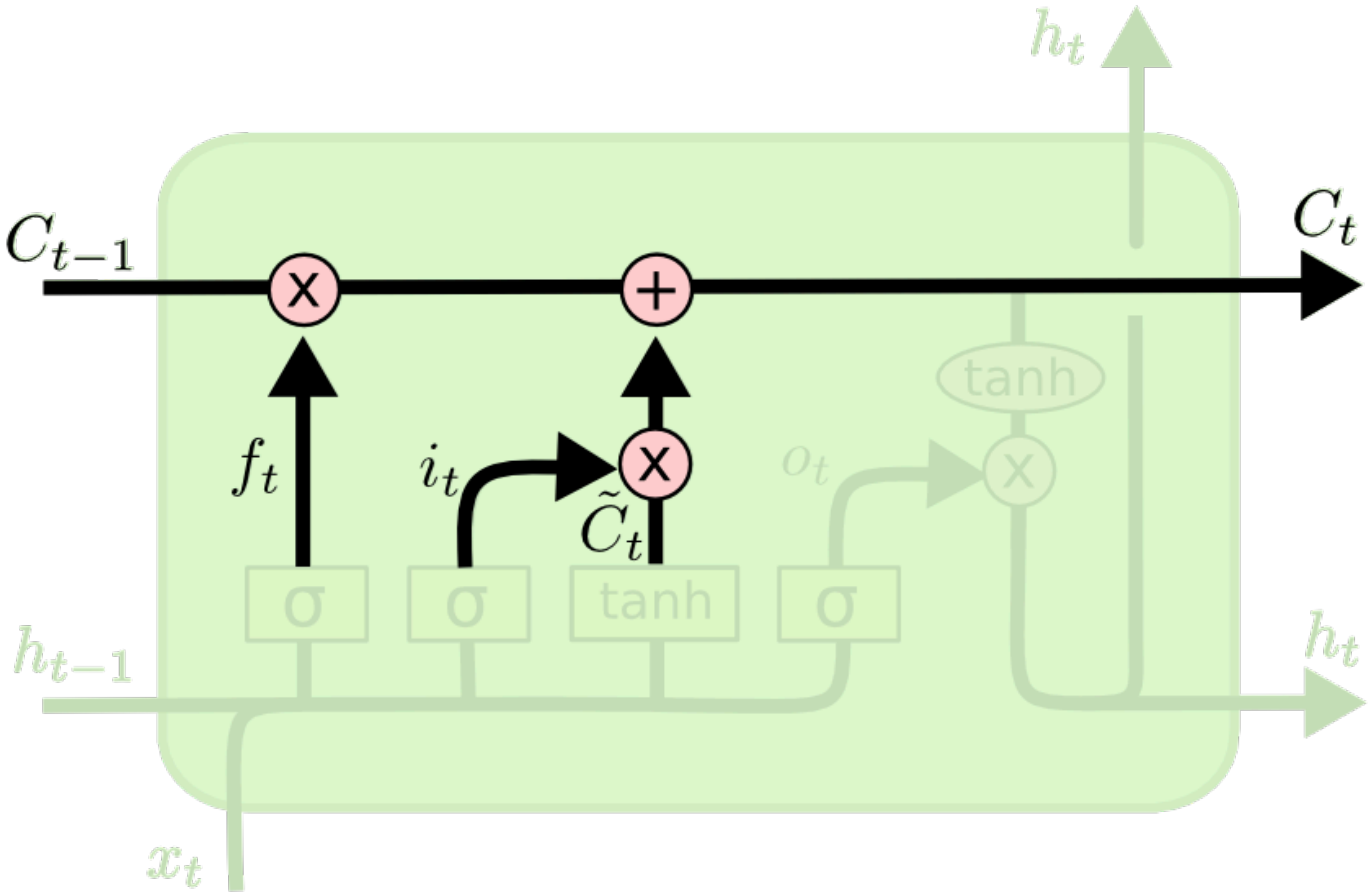
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

what to write into those components

Decide what new information to add to the cell state.

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

Source: Torralba, Freeman, Isola



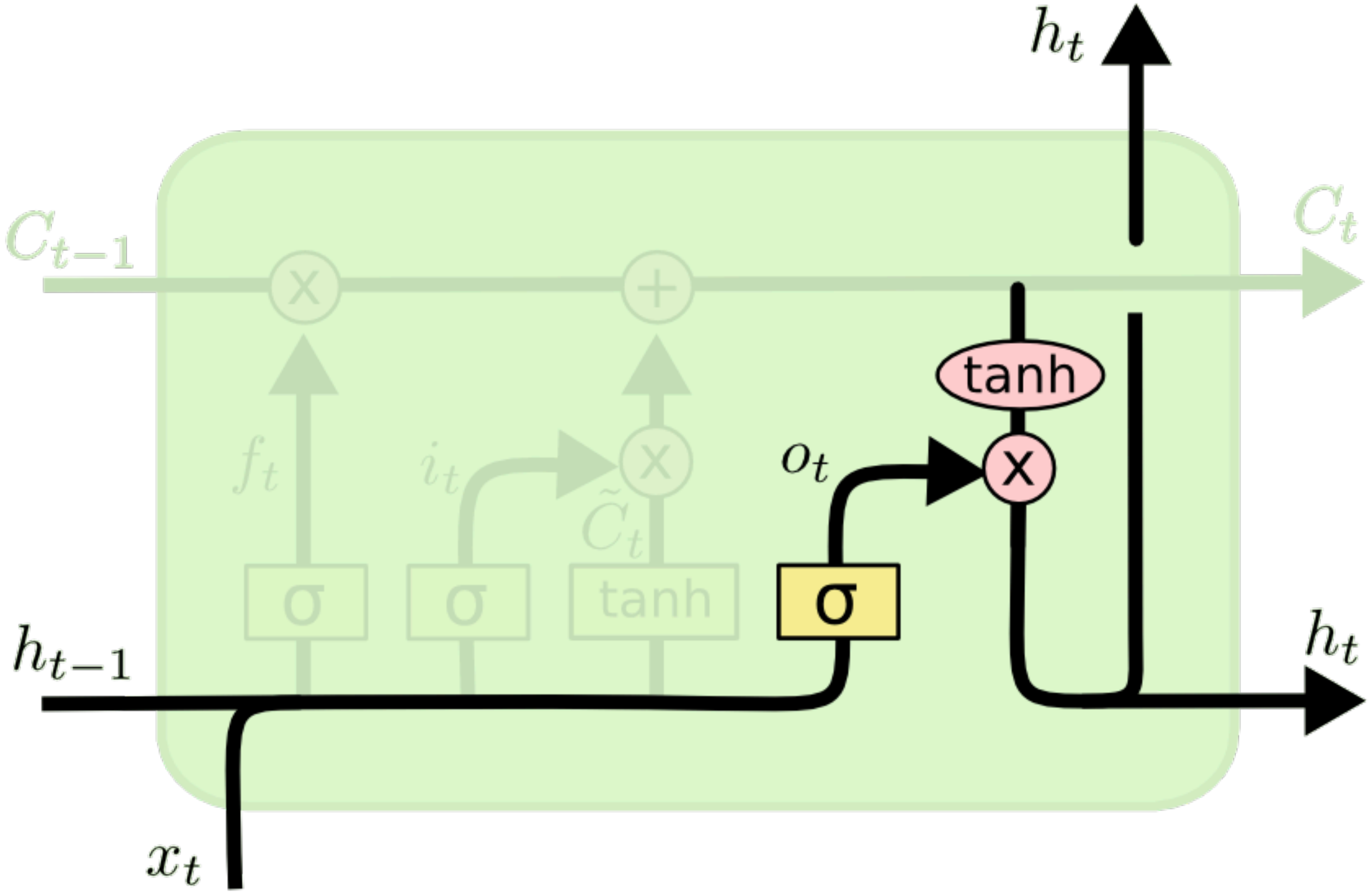
Forget   Write new values

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

Forget selected old information, write selected new information.

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

Source: Torralba, Freeman, Isola



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

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

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/]

Source: Torralba, Freeman, Isola

Some uses for RNNs

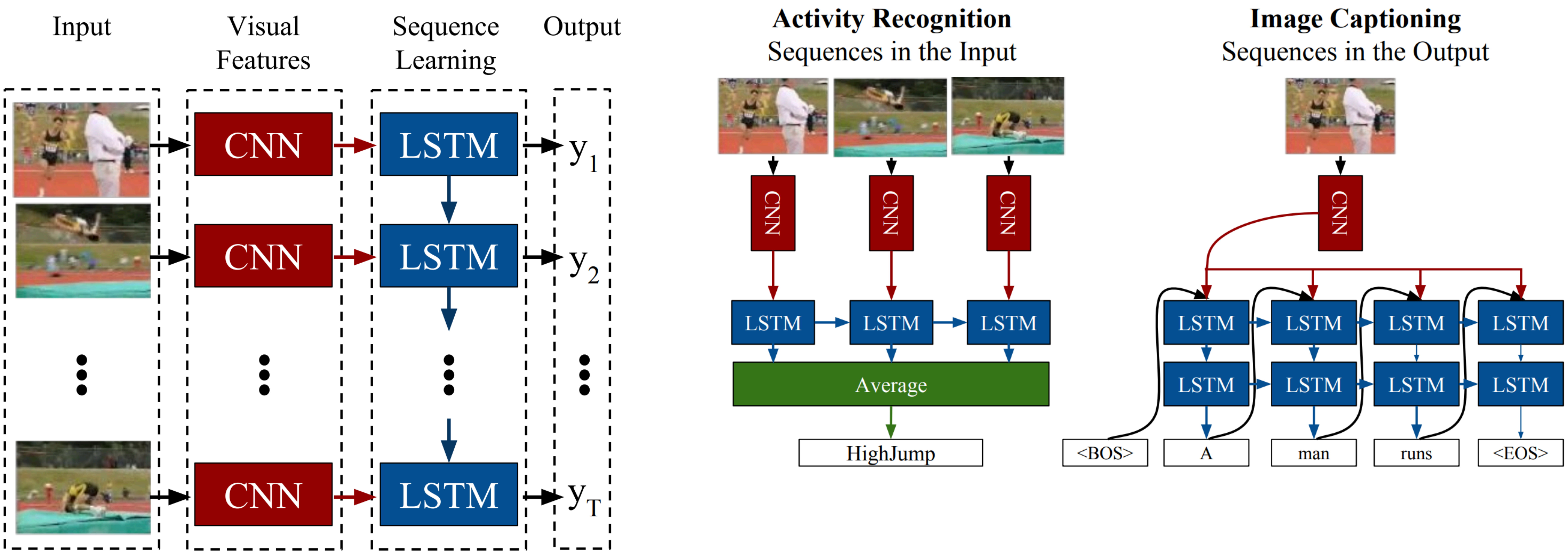


Figure from [Donahue et al. 2016]

Problems with RNNs

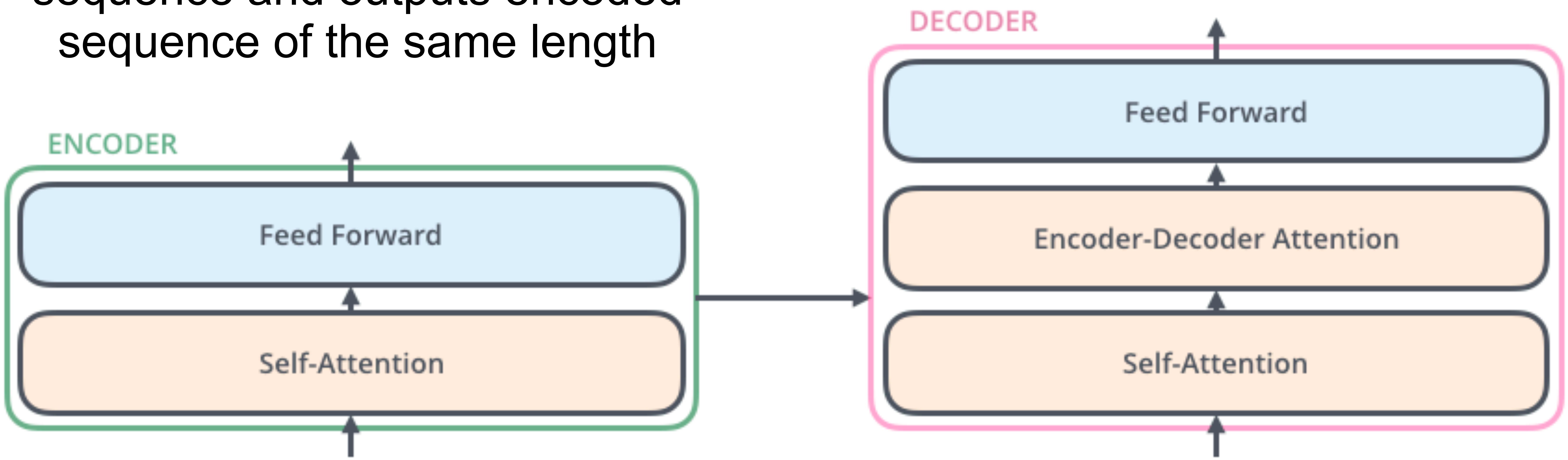
- Hard to obtain motion information.
- Results in very deep networks (often slow, hard to train)
- Doesn't parallelize well
- Depth of network = length of the sequence

Basic transformer model

- Do we really need these sequence models?
- Sequence-to-sequence architecture using only point-wise processing and attention (no recurrent units or convolutions)

Decoder: predicts next token conditioned on encoder output and previously predicted tokens

Encoder: receives entire input sequence and outputs encoded sequence of the same length



A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, NeurIPS 2017

Image source

Self-attention

- Used to capture context within the sequence

As we are encoding “it”, we should focus on “the animal”

As we are encoding “it”, we should focus on “the street”

Source: S. Lazebnik

Image source

Self-attention layer

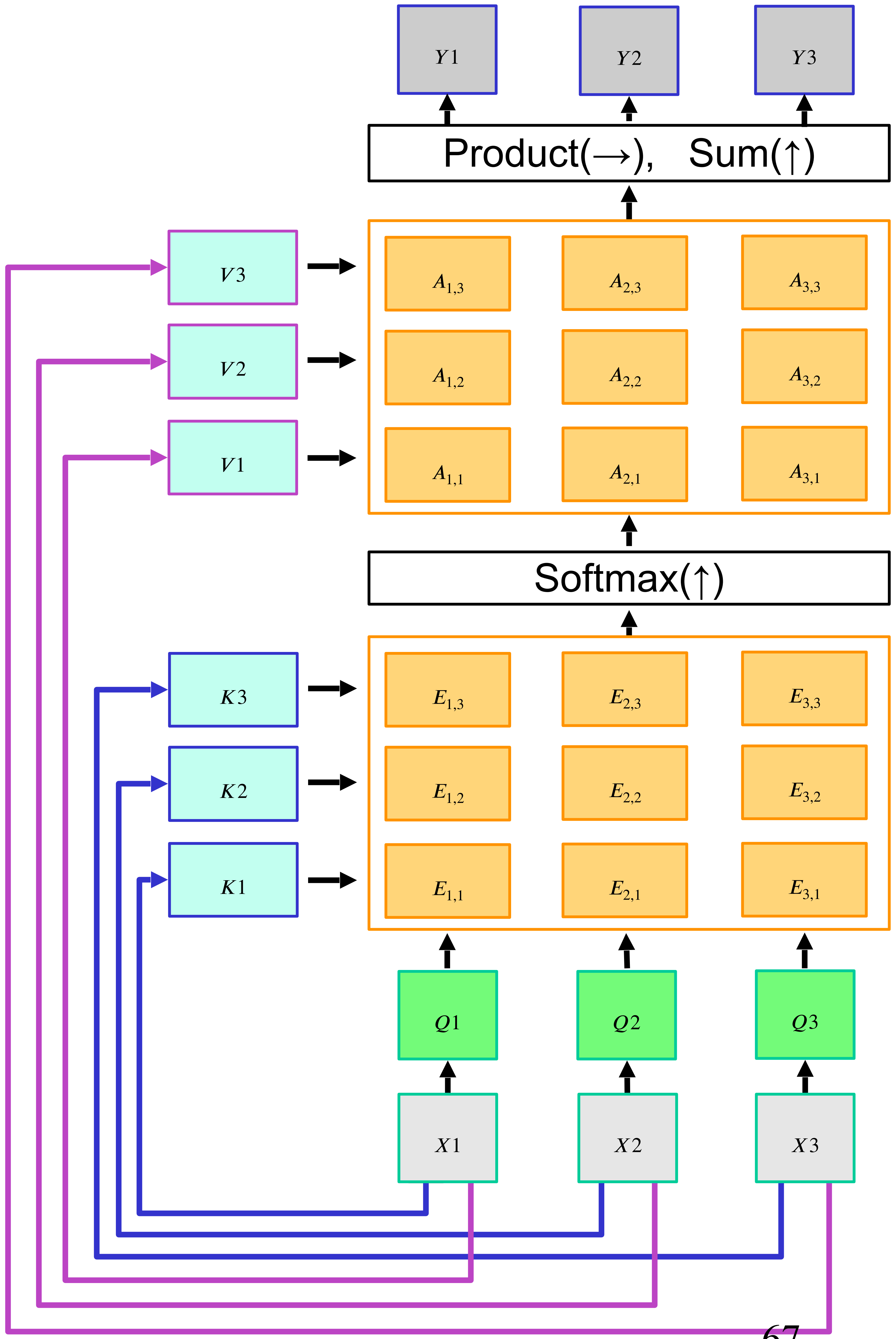
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Similarities: *scaled dot-product attention*

$$E_{i,j} = \frac{(Q_i \cdot Kj)}{\sqrt{D}} \quad \text{or} \quad E = QK^T / \sqrt{D}$$

(D is the dimensionality of the keys)

- Attn. weights: $A = \text{softmax}(E, \text{dim} = 1)$
- Output vectors:

$$Y_i = \sum_j A_{i,j} V_j \quad \text{or} \quad Y = AV$$

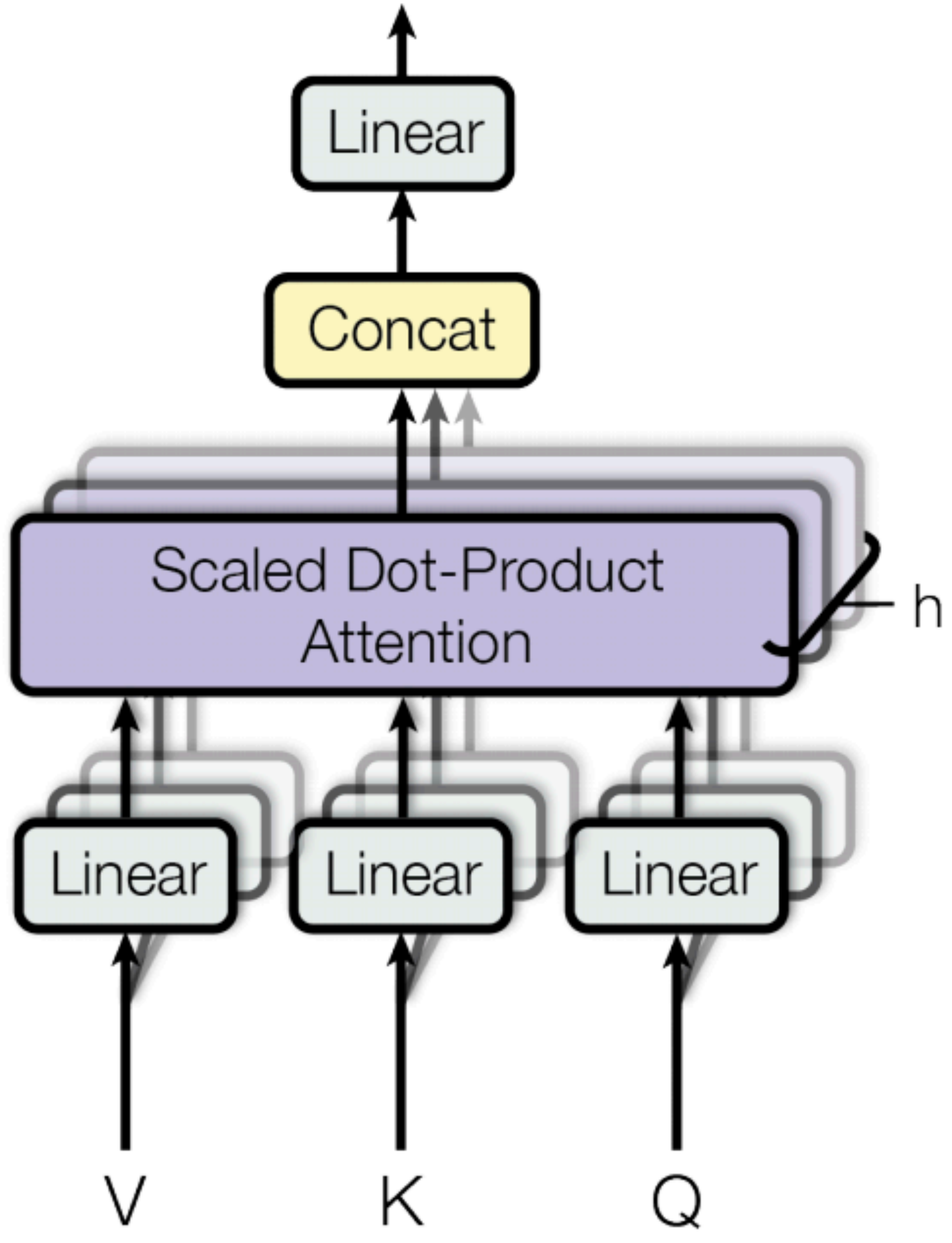


One query per input vector

Adapted from J. Johnson and S. Lazebnik.

Multi-head attention

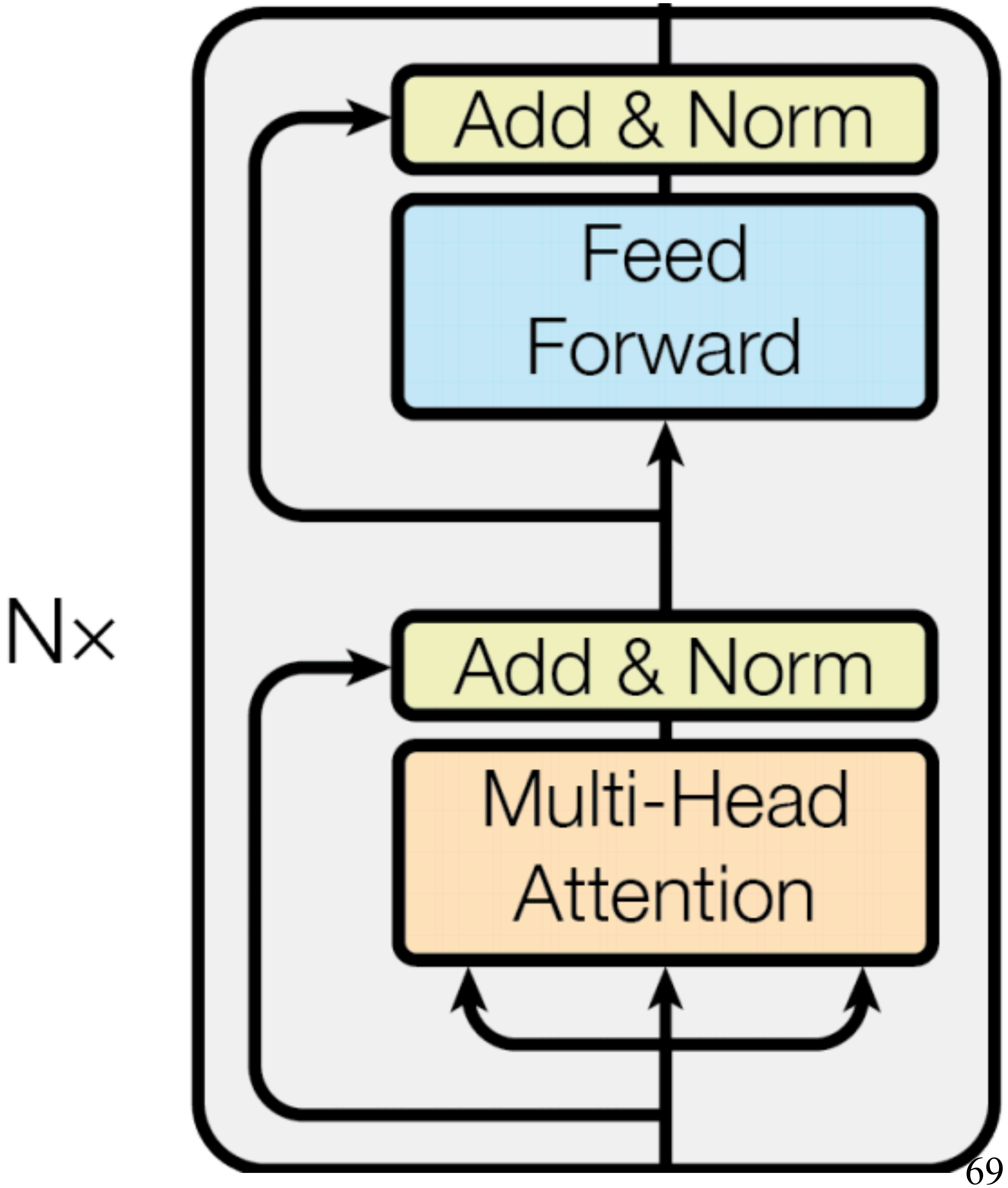
- Run h attention models in parallel on top of different linearly projected versions of Q, K, V; concatenate and linearly project the results
- Intuition: enables model to attend to different kinds of information at different positions



Source: S. Lazebnik

Transformer blocks

- A **Transformer** is a sequence of transformer blocks
  - Vaswani et al.: N=12 blocks, embedding dimension = 512, 6 attention heads
  - **Add & Norm:** residual connection followed by layer normalization
  - **Feedforward:** two linear layers with ReLUs in between, applied independently to each vector
- Attention is the only interaction between inputs!



Source: S. Lazebnik

Positional encoding

- To give transformer information about ordering of tokens, add function of position (based on sines and cosines) to every input

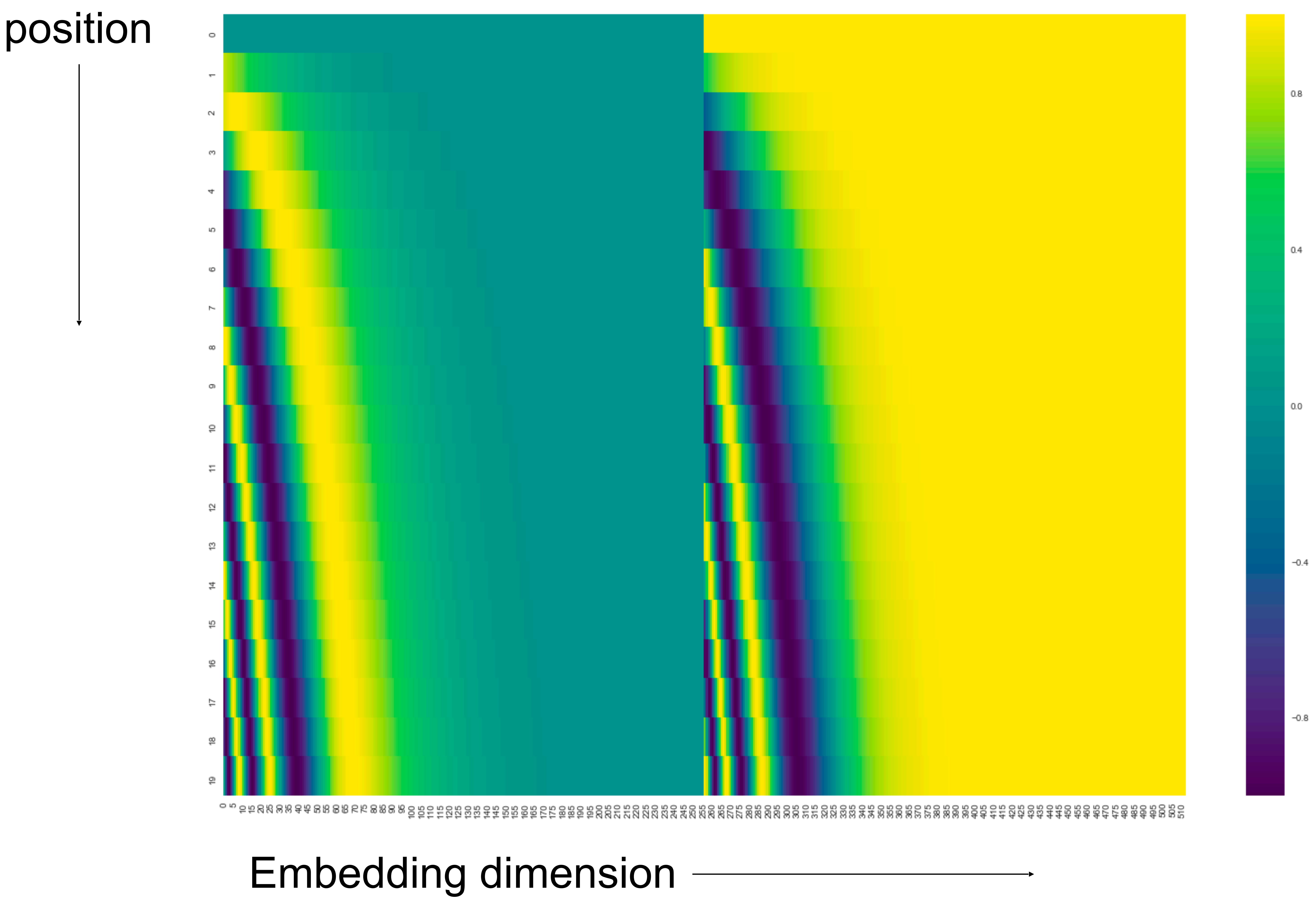
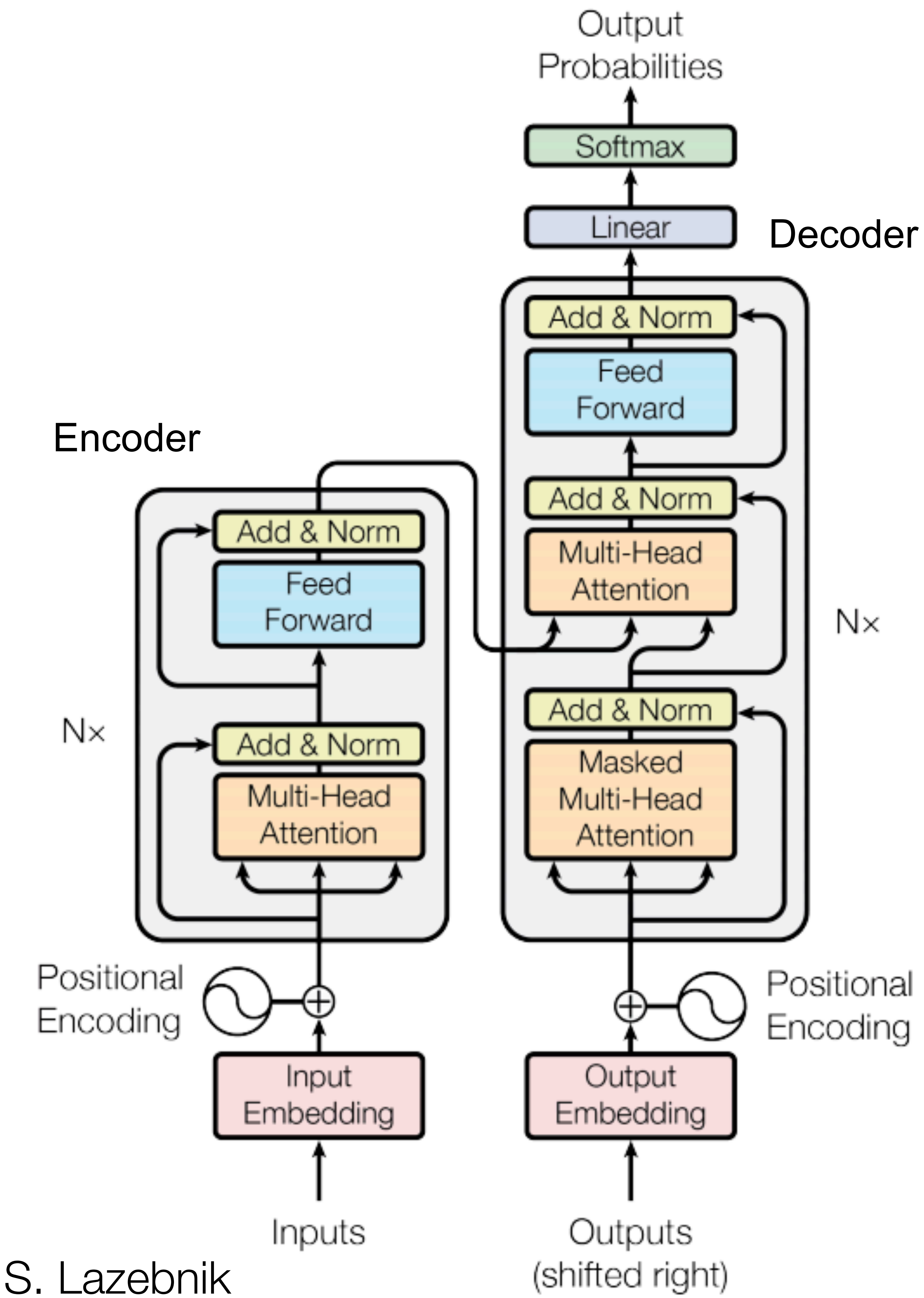


Image source

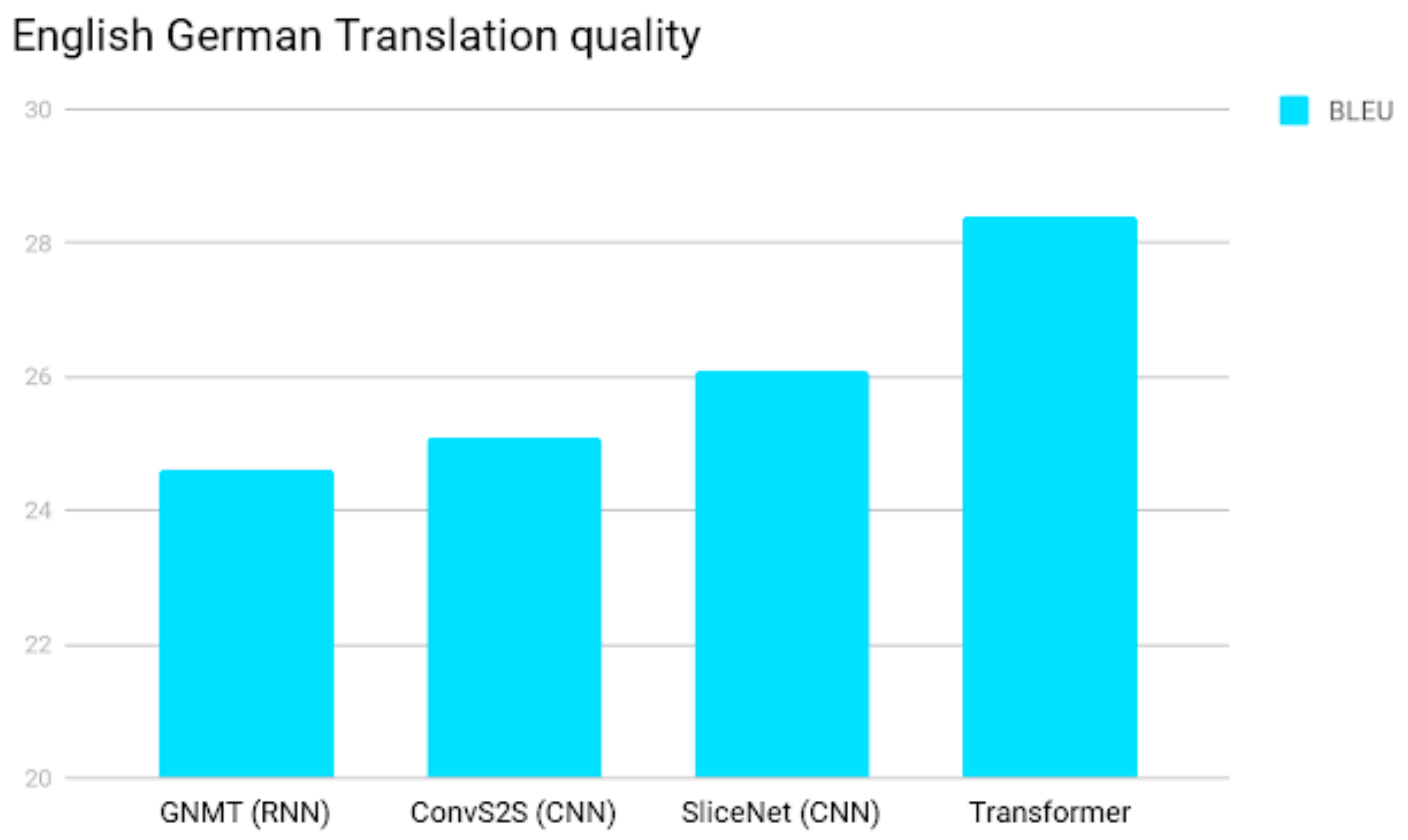
Transformer architecture: Zooming back out

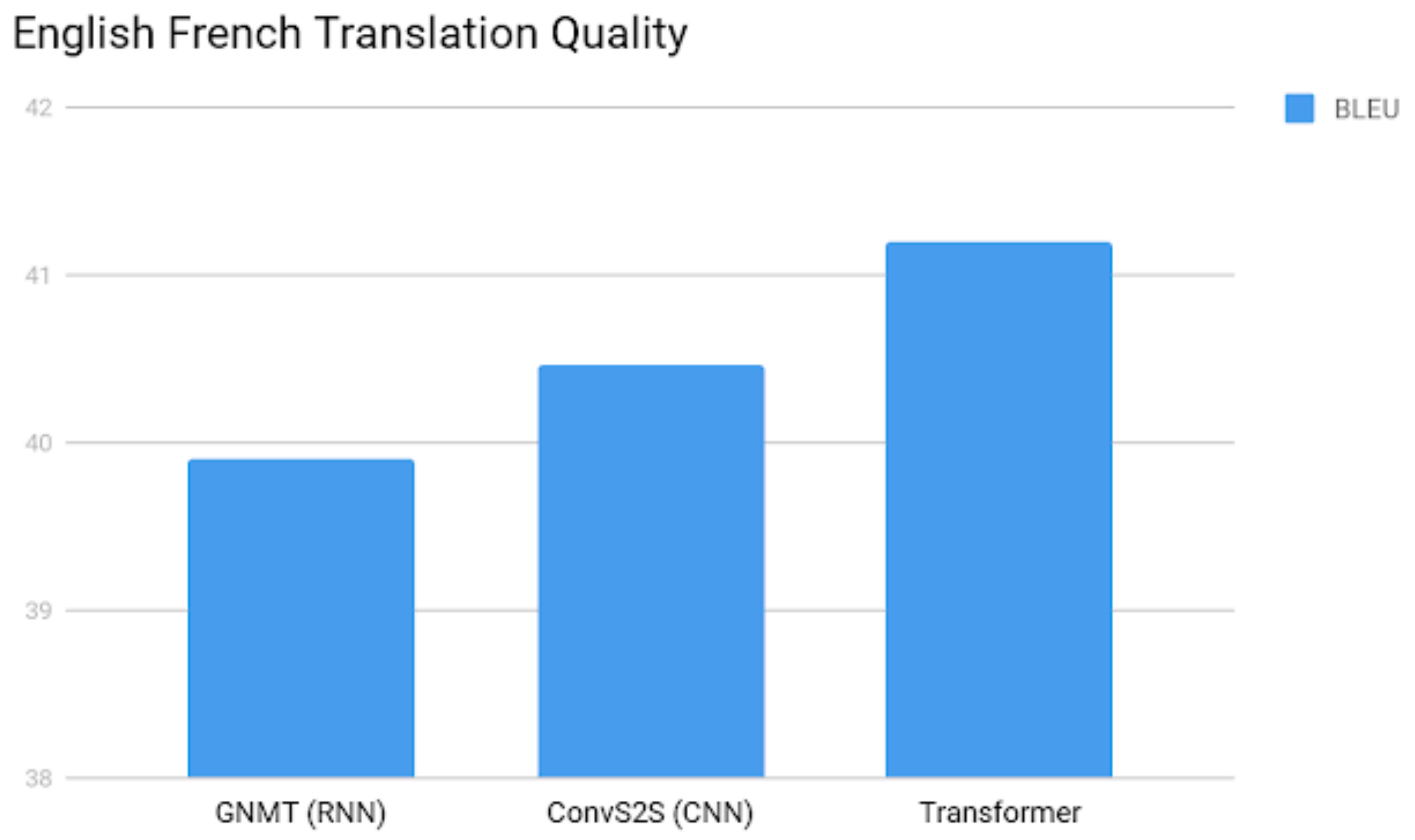


Source: S. Lazebnik

A. Vaswani et al., [Attention is all you need](), NeurIPS 2017

Language translation results



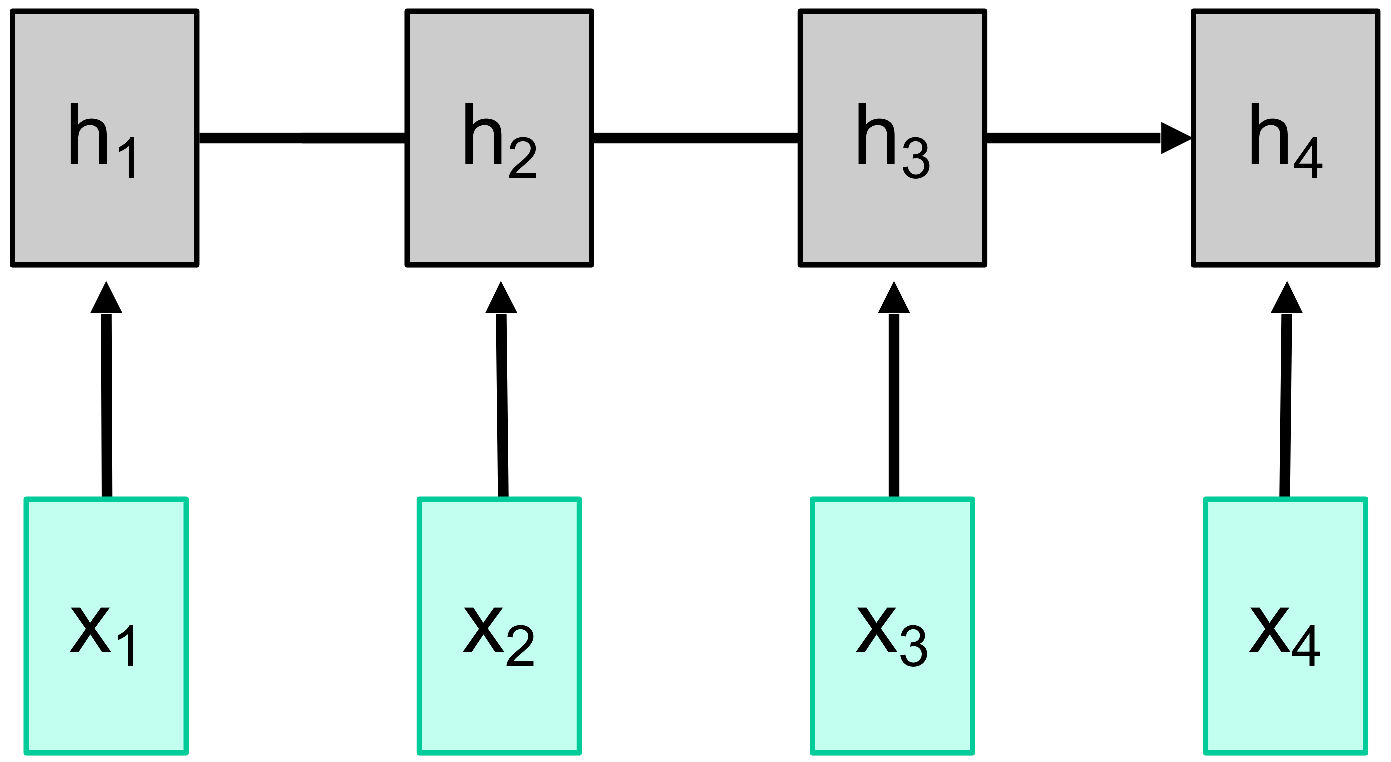


https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Source: S. Lazebnik

Different ways of processing sequences

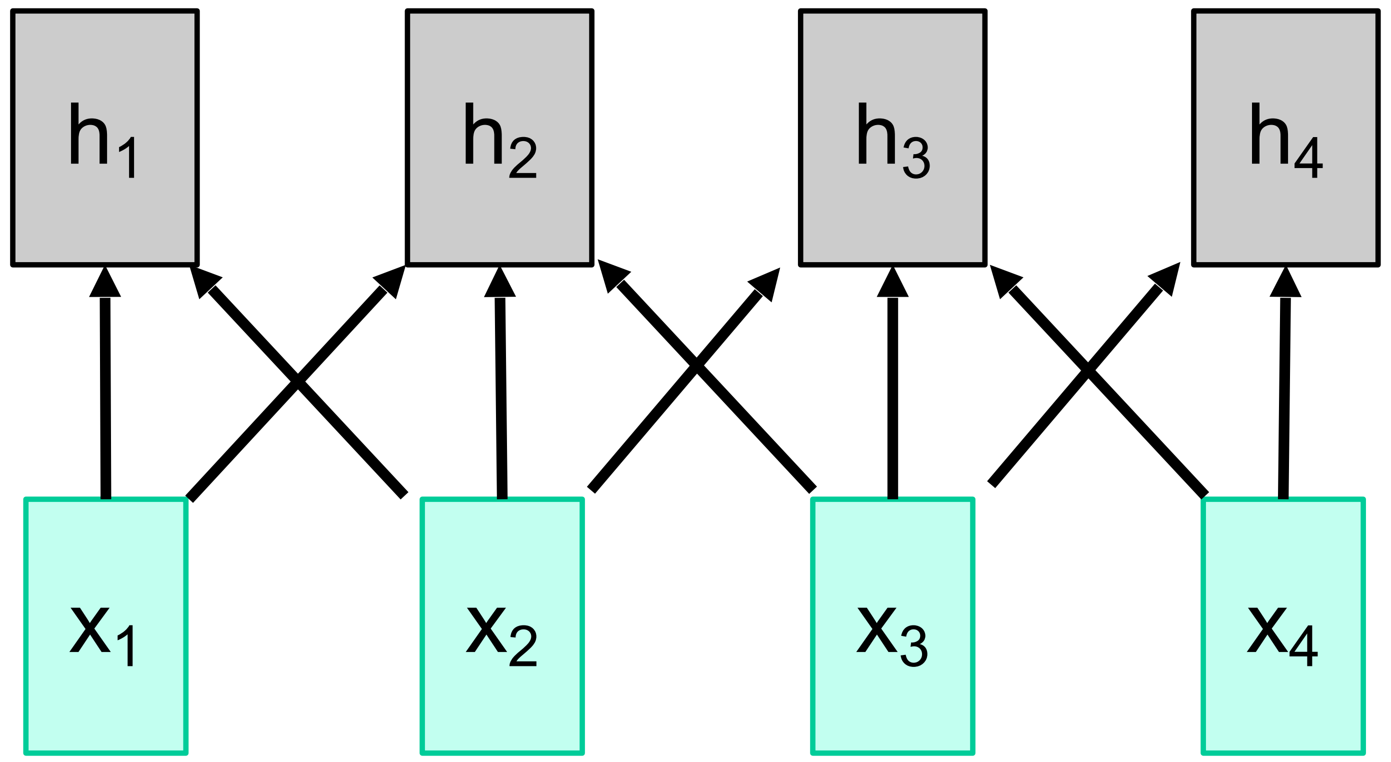
RNN



Works on **ordered sequences**

- Pros: Good for long sequences: After one RNN layer, h_T "sees" the whole sequence
- Con: Not parallelizable: need to compute hidden states sequentially. Very deep.
- Con: Hidden states have limited expressive capacity

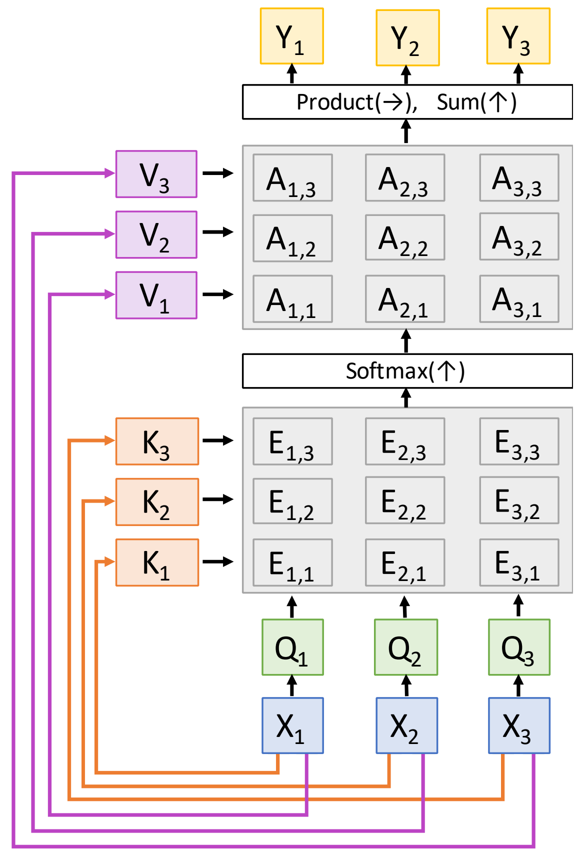
Convolutional network



Works on **multidimensional grids**

- Pro: Each output can be computed in parallel (at training time)
- Con: Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

Transformer

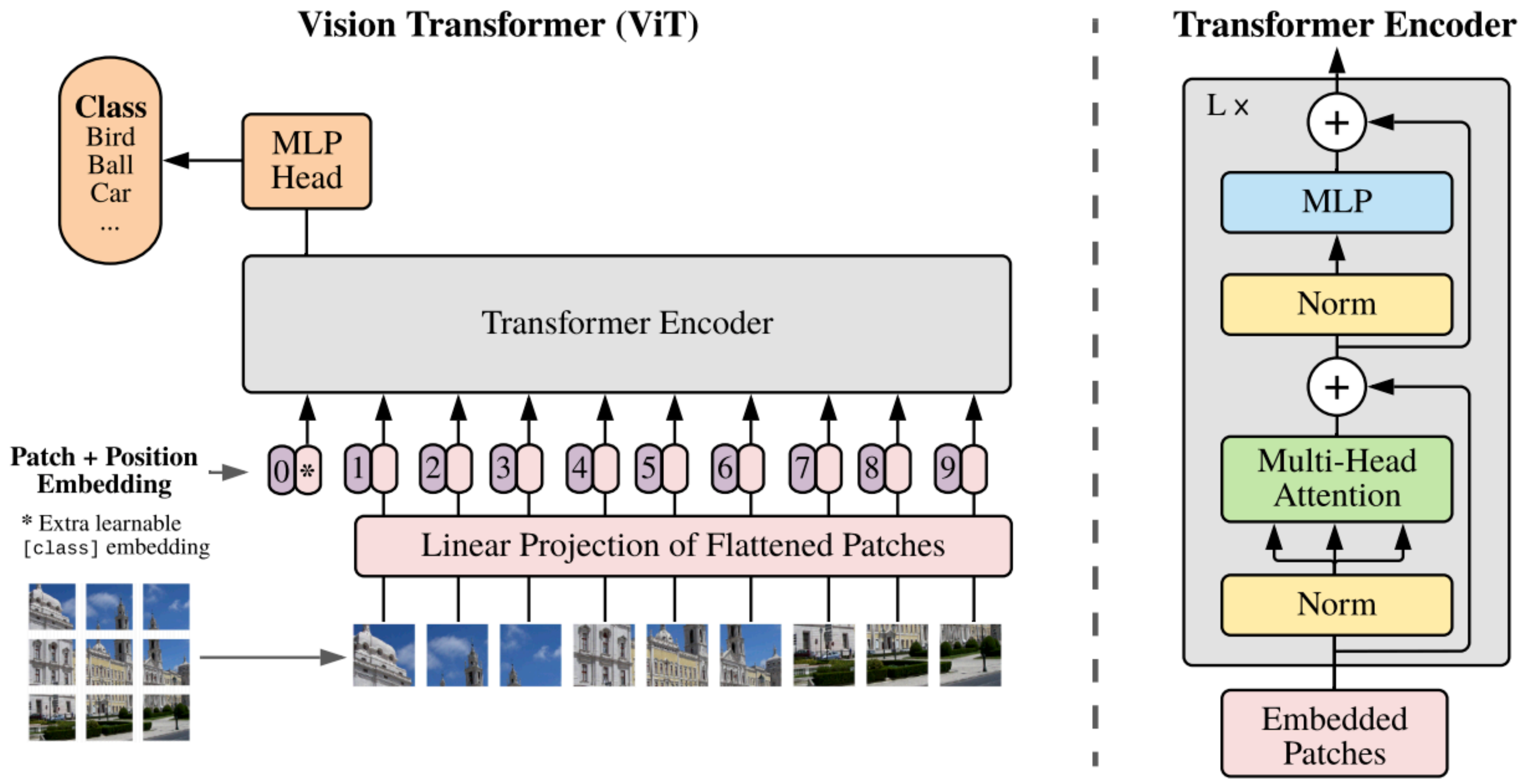


- Works on **sets of vectors**
- Pro: Good at long sequences: after one self-attention layer, each output "sees" all inputs!
- Pro: Each output can be computed in parallel (at training time)
- Con: Memory-intensive. O(N^2) without modifications.

Source: S. Lazebnik

Preview for later in the course: Vision transformer (ViT)

- Split an image into patches, feed linearly projected patches into standard transformer encoder
  - With patches of 14x14 pixels, you need 16x16=256 patches to represent 224x224 images



A. Dosovitskiy et al. [An image is worth 16x16 words: Transformers for image recognition at scale](). ICLR 2021

Source: S. Lazebnik

Next class: representation learning