LECTURE 3: ATTENTION IN MACHINE LEARNING

CSC5991: Introduction to LLMs

Slides taken from Stanford CS231n, CMU 16-385, and UWaterloo CS 886.

OUTLINE

- What is attention?
- RNN + Attention in machine translation (NLP)
- CNN architecture
  - Attention in CNNs
- RNN + Attention in image captioning (CV)

WHAT IS ATTENTION?

- The notion of **exploiting context** is not new
  - **RNN** – context from **temporal locality** (useful for sequences/time-series data)
  - **CNN** – context from **spatial locality** (useful for images)
  - Embedding priors into models forces them to pay "attention" to relevant features for a given problem

- What we now call "attention" in DL
  - The idea of paying "attention" to the most relevant parts of I/P
  - Very useful in **sequence-to-sequence** modelling
  - Ideally, we'd like to **learn** this!

WHAT IS A LEARNED ATTENTION MECHANISM?

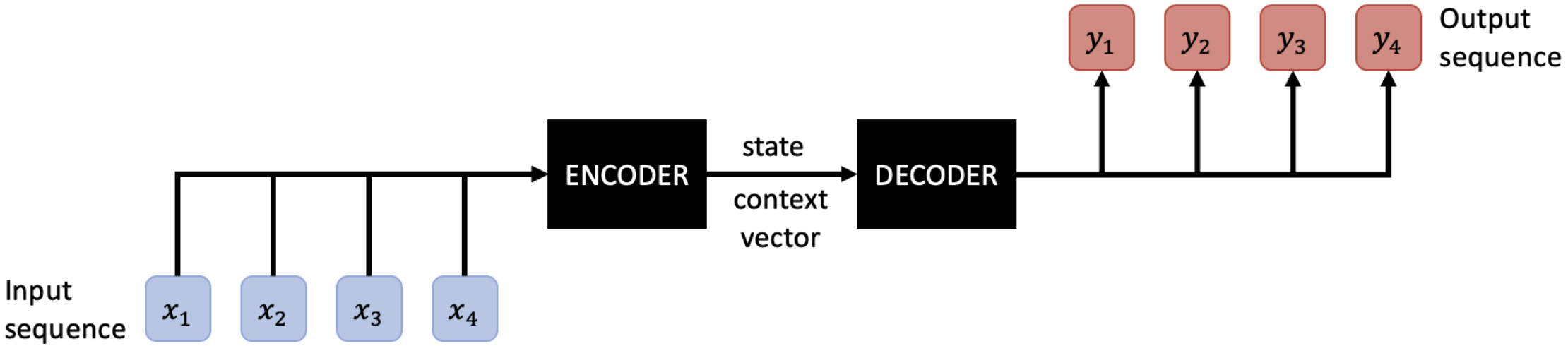
- An attention mechanism typically refers to **function** that allows a model to attend to different content

- There are many forms of attention mechanisms
  - **Additive**
  - **Dot-product**

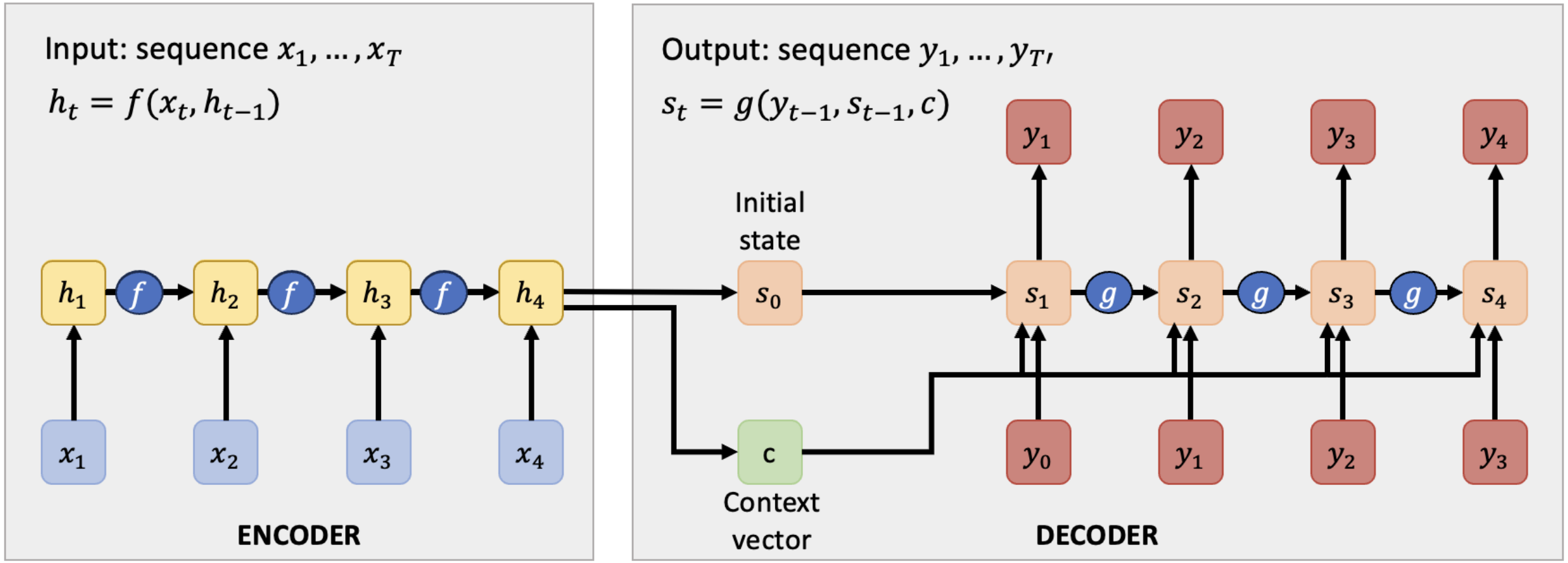
- We have names to distinguish attention based on what is attended to
  - **Self-attention** (intra attention)
  - **Cross-attention** (encoder-decoder attention/inter attention)

SEQ2SEQ: RNNS + **ATTENTION**

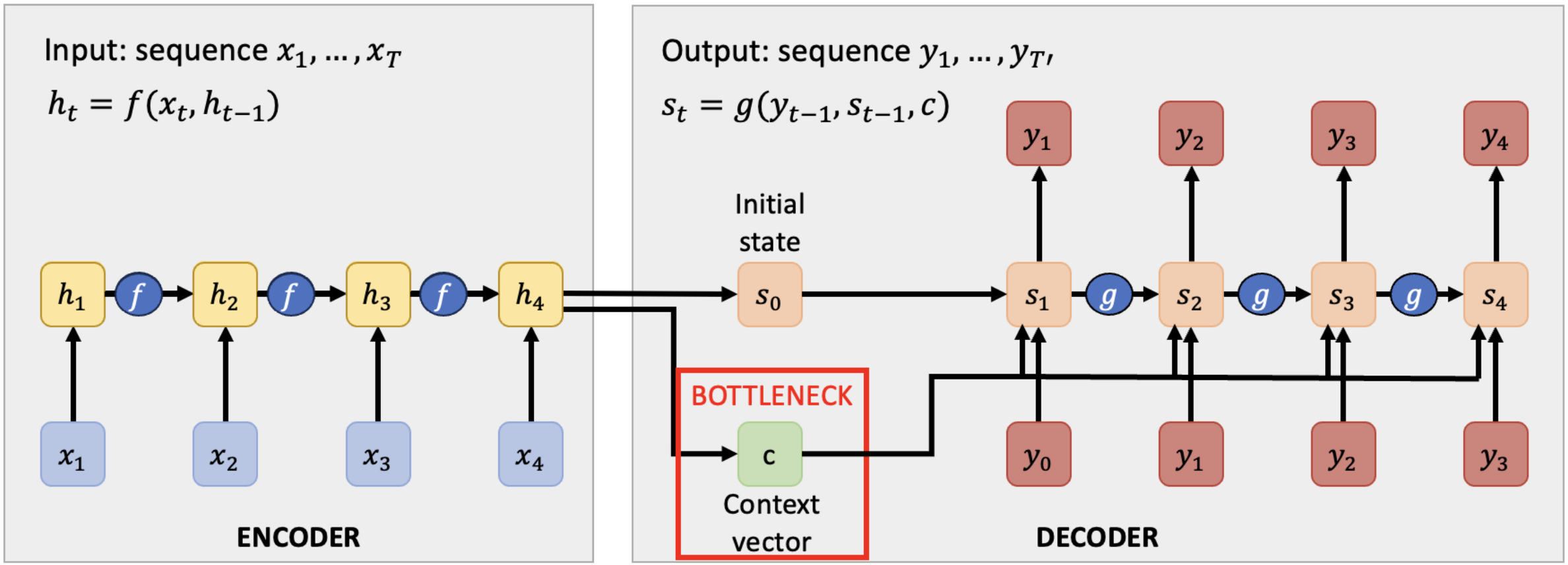
- **Example Scenarios**
  - Text → Text (e.g. Q/A, translation, text summarization)
  - Image → Text (e.g. image captioning)
- **How?** Usually Encoder-Decoder models
  - e.g. RNNs, transformers



FIXED CONTEXT



BOTTLENECK


Input: sequence $x_1, \dots, x_T$
$h_t = f(x_t, h_{t-1})$
h_1 f h_2 f h_3 f h_4
x_1 x_2 x_3 x_4
ENCODER
Output: sequence $y_1, \dots, y_{T'}$
$s_t = g(y_{t-1}, s_{t-1}, c)$
Initial state
s_0
BOTTLENECK
c
Context vector
y_1 y_2 y_3 y_4
s_1 g s_2 g s_3 g s_4
y_0 y_1 y_2 y_3
DECODER

"you can't cram the meaning of a whole %&@#&ing sentence into a single $*(&@ing vector!"

— Ray Mooney (NLP professor at UT Austin)

DESIGN CONTEXT USING ATTENTION

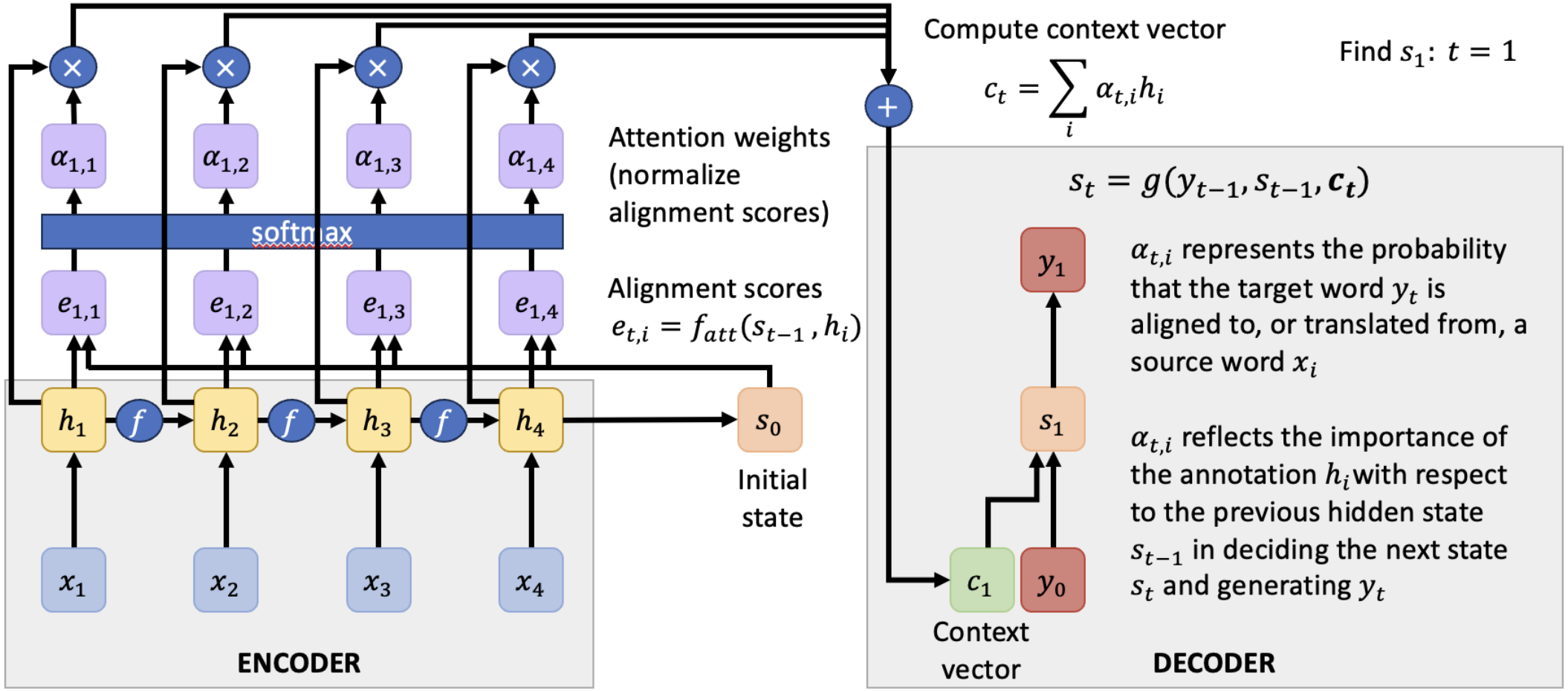
- **Idea!** Use a different context vector for each timestep in the decoder

$$s_t = g(y_{t-1}, s_{t-1}, \boldsymbol{c_t})$$

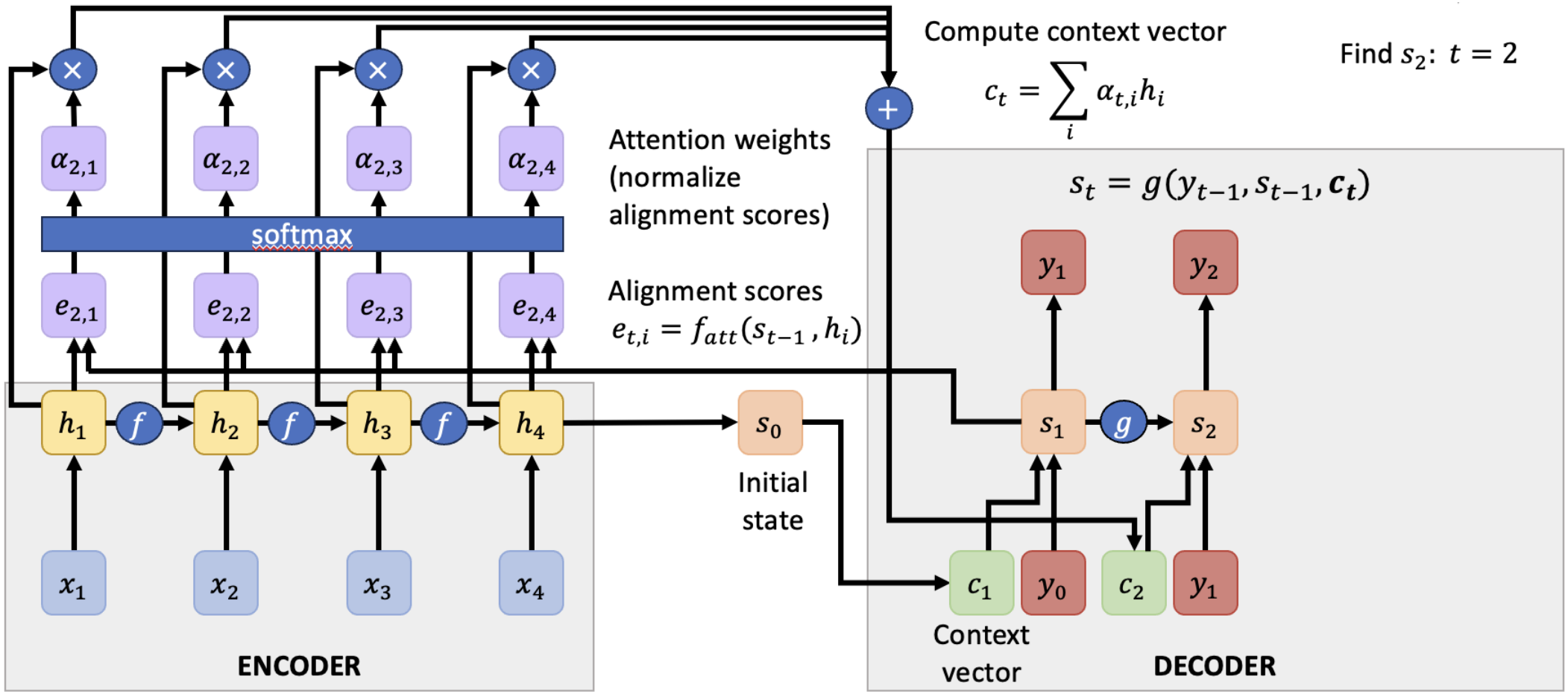
- No more bottleneck through a single vector

- Craft the context vector so that it "looks at" different parts of the input sequence for each decoder timestep

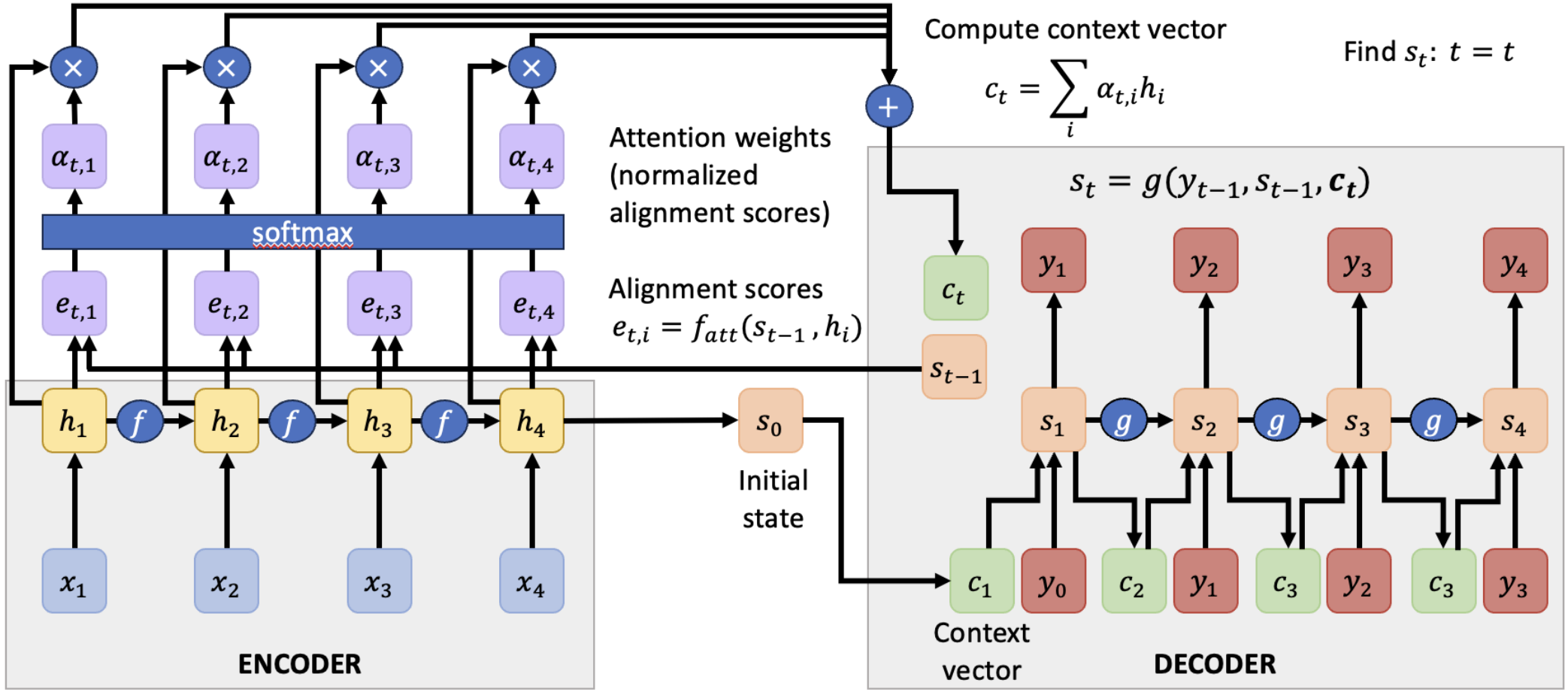
SEQ2SEQ: RNNS + **ATTENTION**



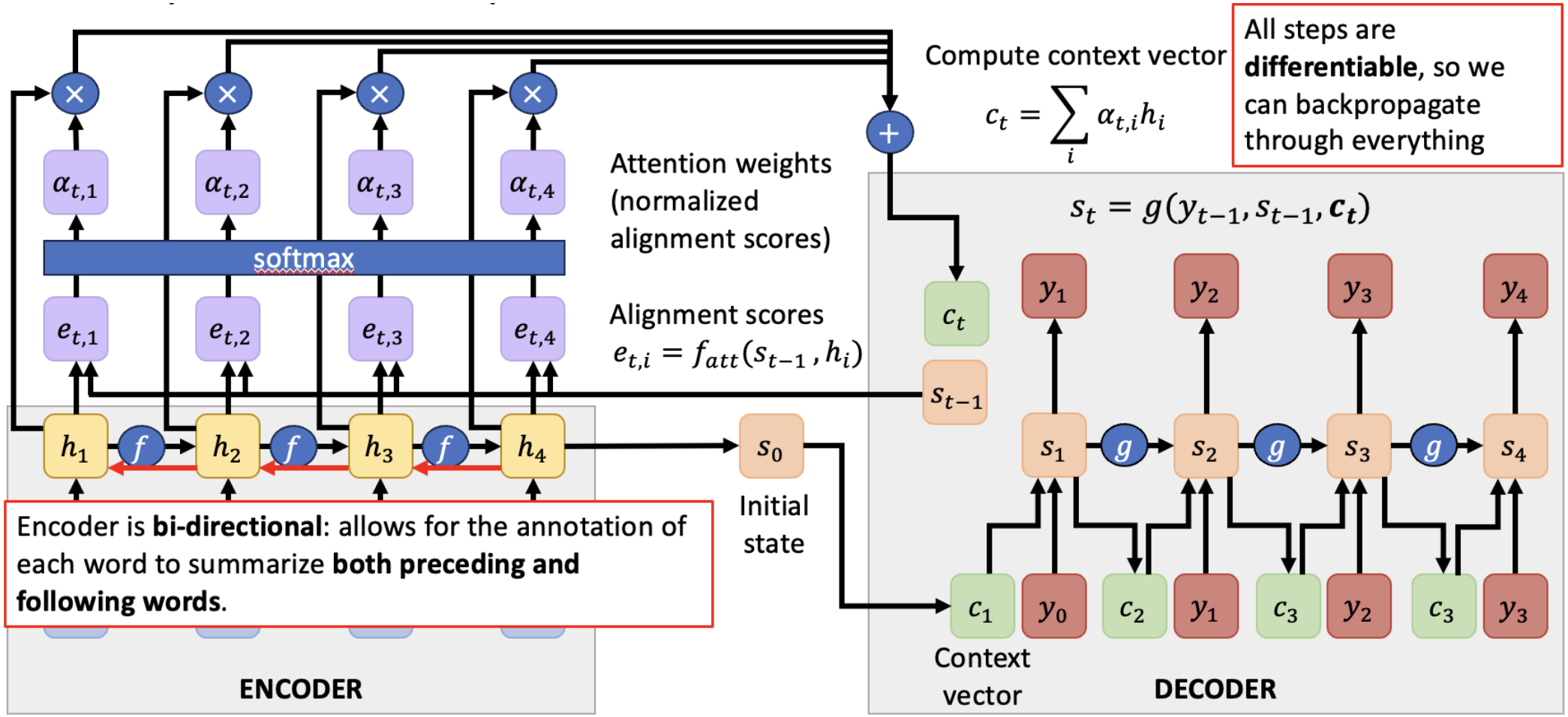
SEQ2SEQ: RNNS + **ATTENTION**



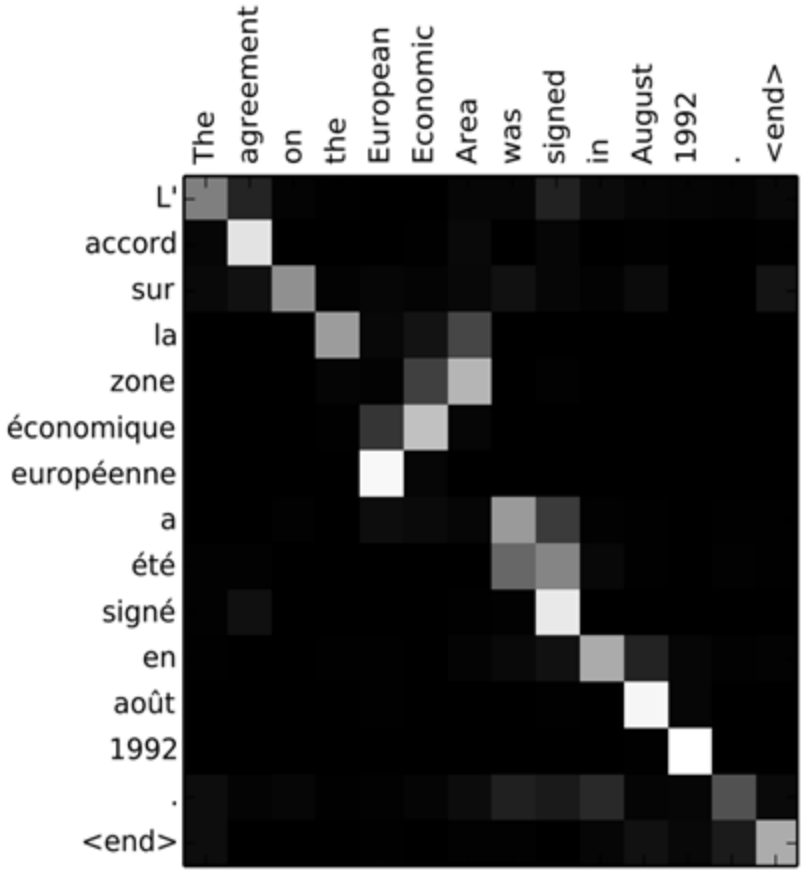
SEQ2SEQ: RNNS + **ATTENTION**

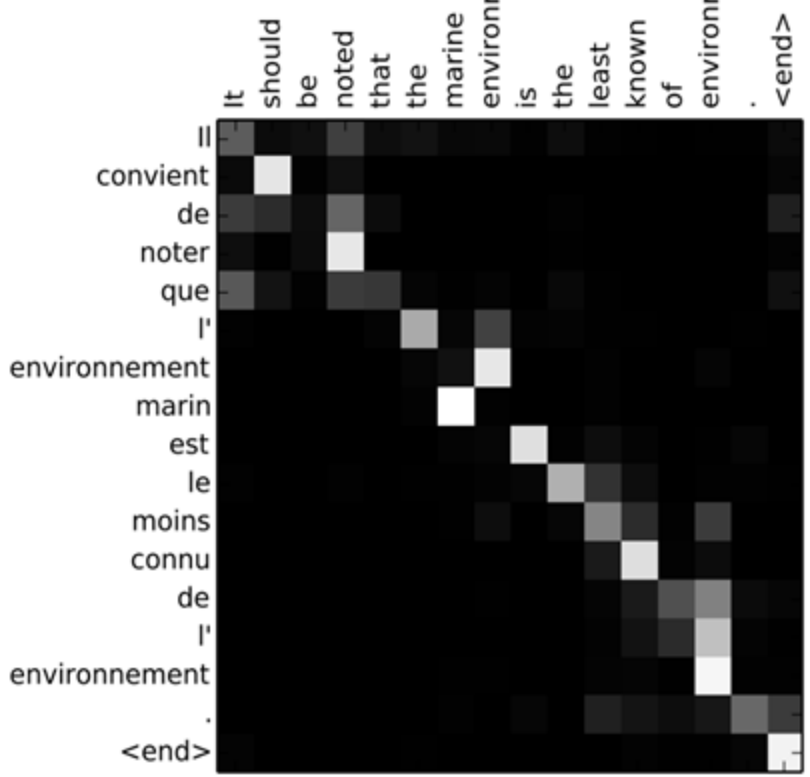


SEQ2SEQ: RNNS + **ATTENTION**



ATTENTION MAP





CONVOLUTIONAL NEURAL NETWORKS (CNNS)

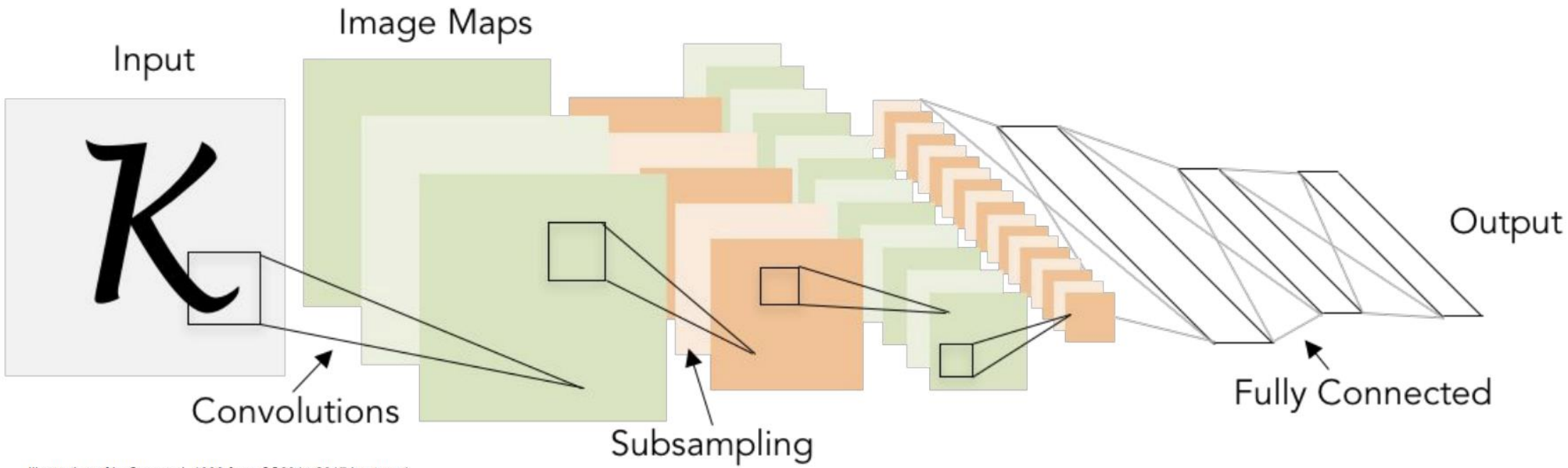
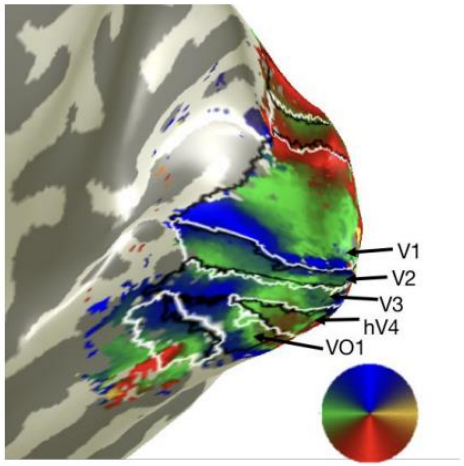
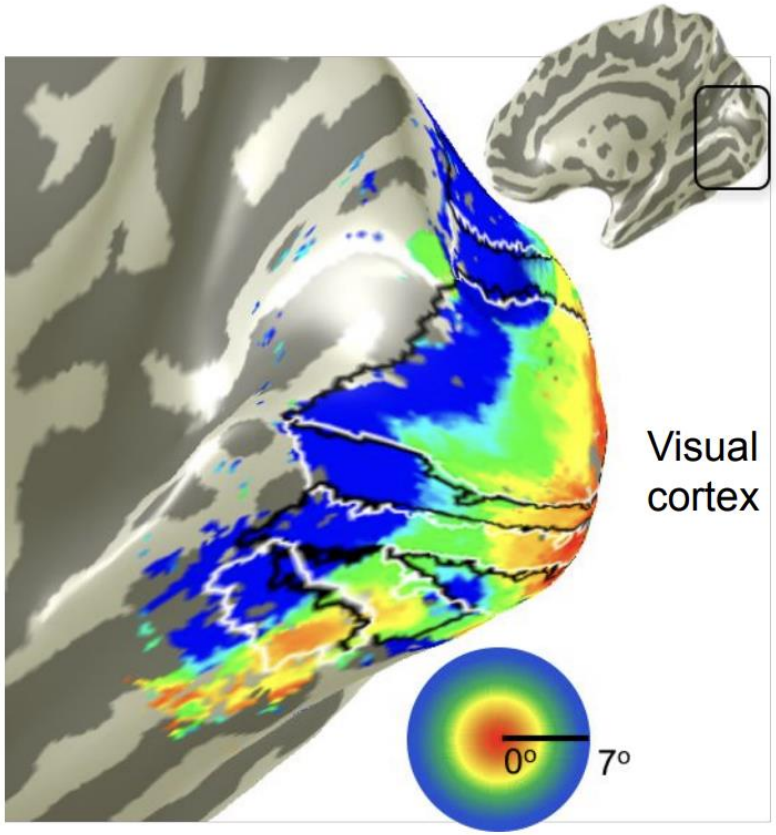


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

A bit of history

Topographical mapping in the cortex:
nearby cells in cortex represent
nearby regions in the visual field





Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

Hierarchical organization

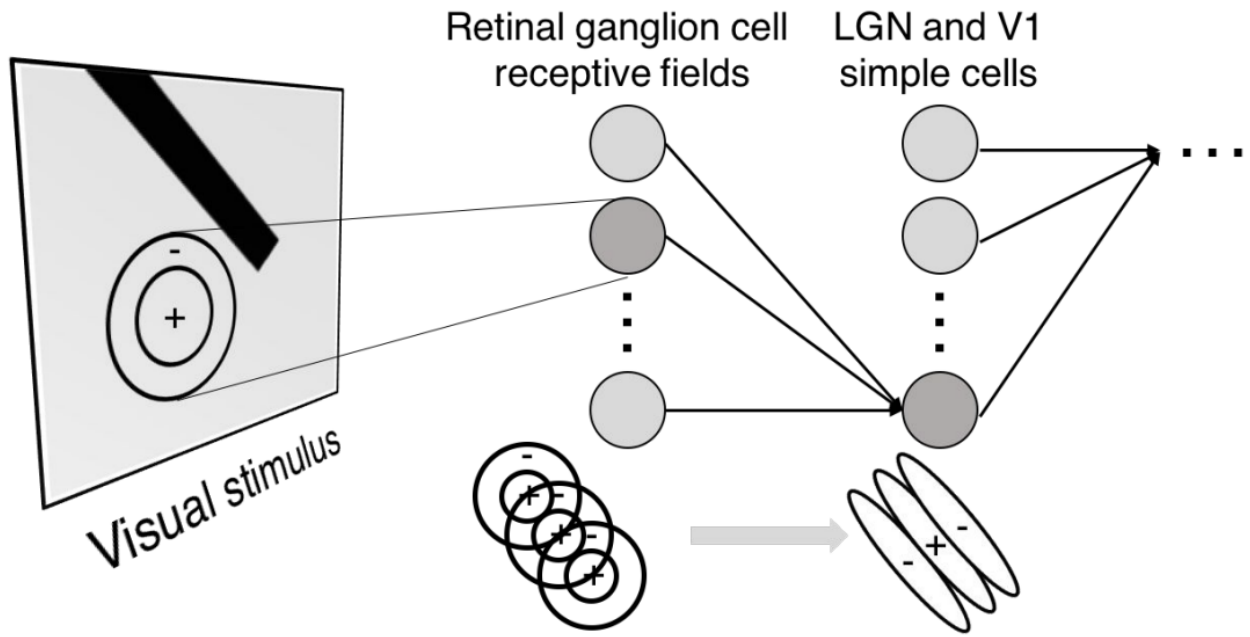
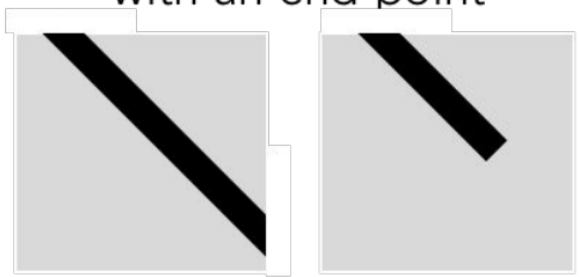


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point



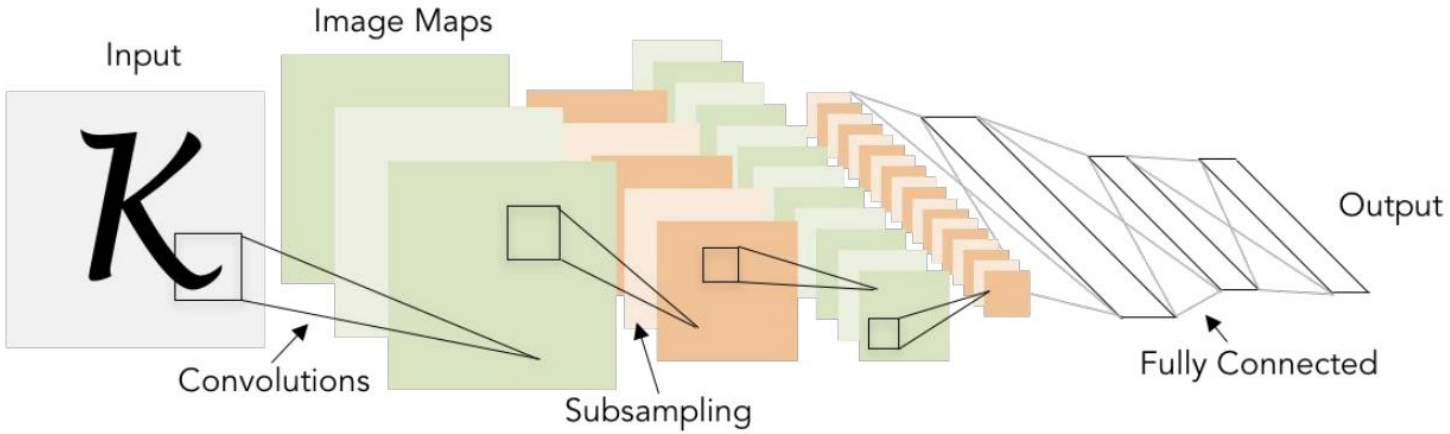
No response

Response (end point)

A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

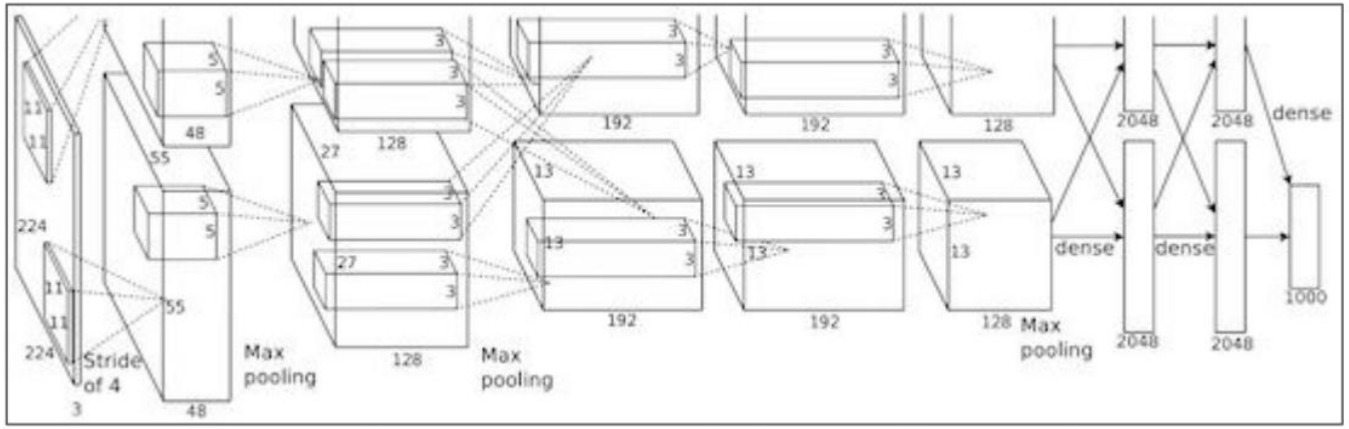


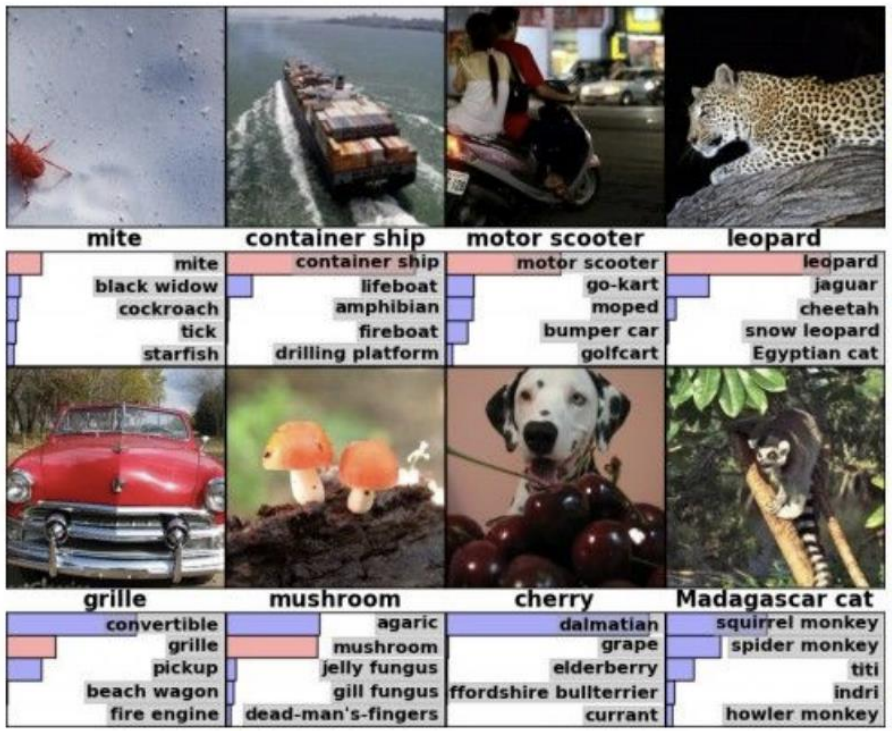
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

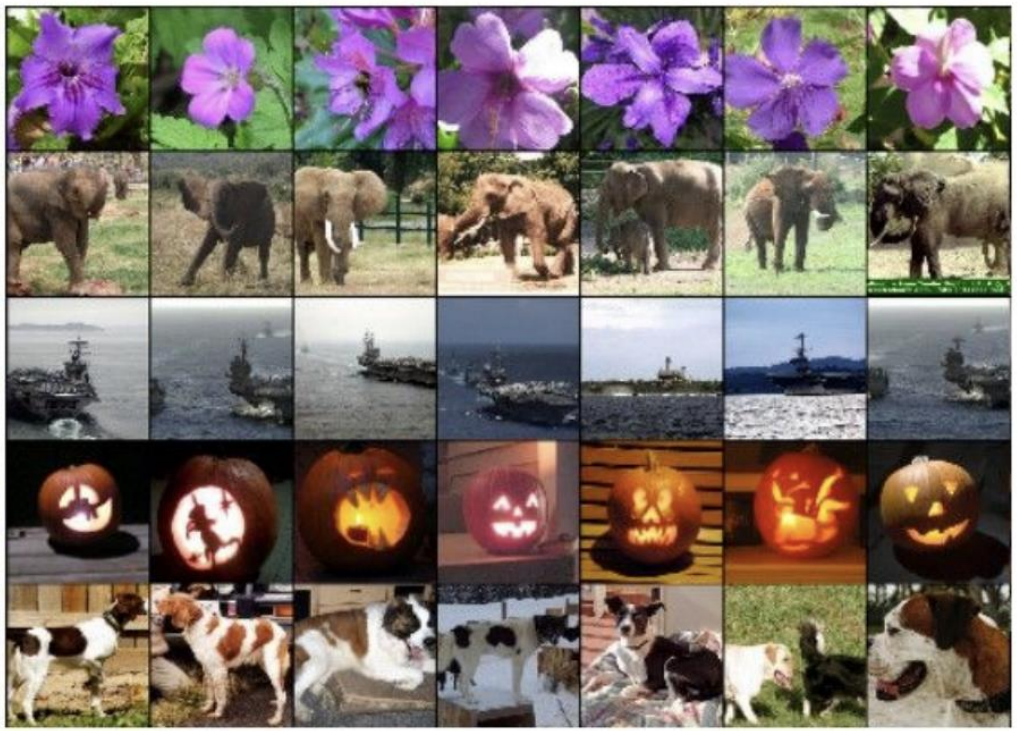
"AlexNet"

Fast-forward to today: ConvNets are everywhere

Classification

Retrieval





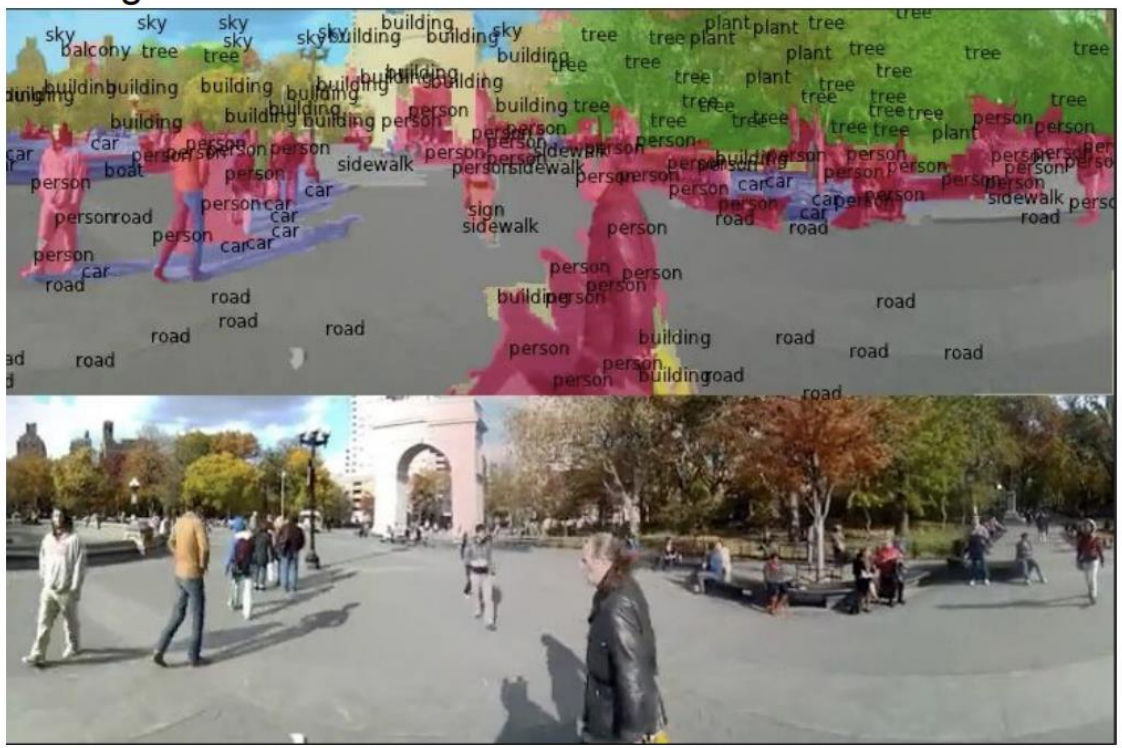
Fast-forward to today: ConvNets are everywhere

Detection

Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

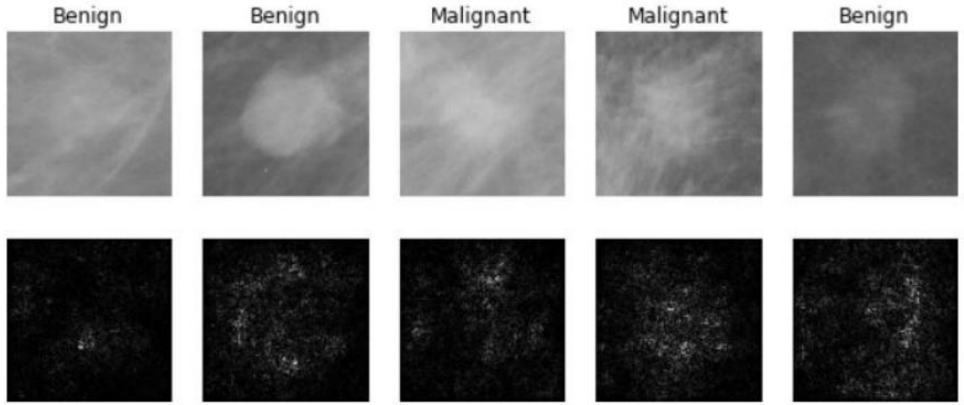
Segmentation



Figures copyright Clement Farabet, 2012.
Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere



[Levy et al. 2016]

[Dieleman et al. 2014]

[Sermanet et al. 2011]
[Ciresan et al.]

Fast-forward to today: ConvNets are everywhere

No errors

A white teddy bear sitting in the grass

Minor errors

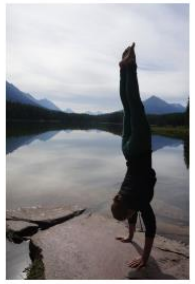
A man in a baseball uniform throwing a ball

Somewhat related

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor



A woman standing on a beach holding a surfboard

Image Captioning

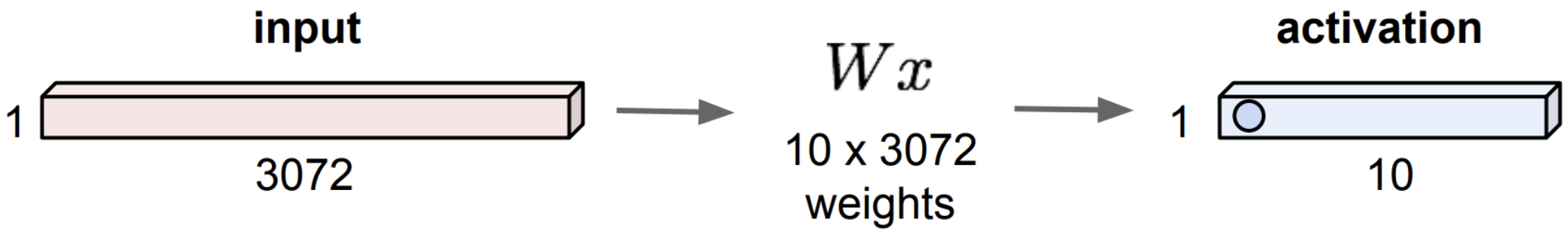
[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-player-shortstop-infield-1045263/

Captions generated by Justin Johnson using Neuraltalk2

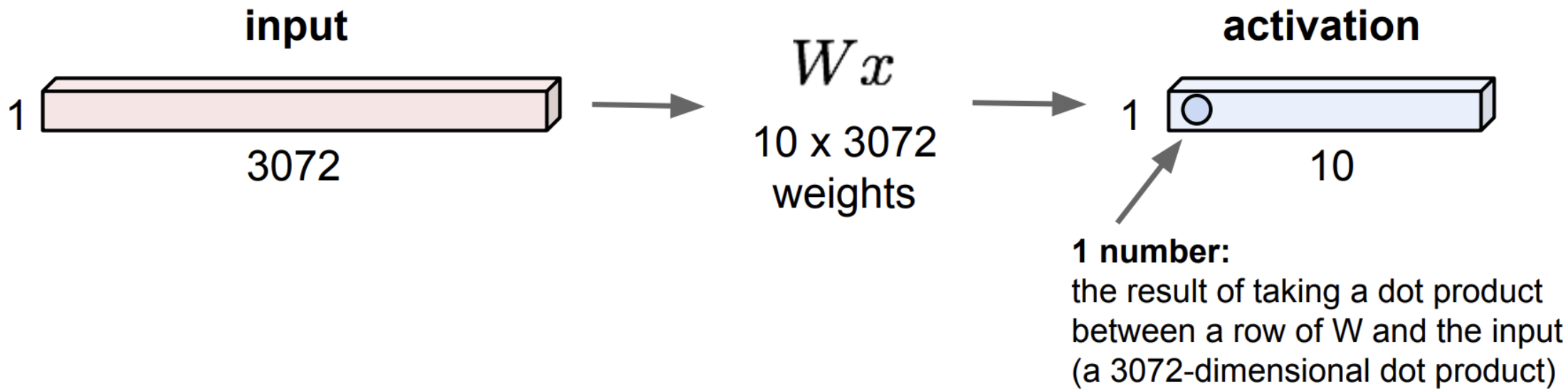
FULLY CONNECTED LAYER

32x32x3 image -> stretch to 3072 x 1



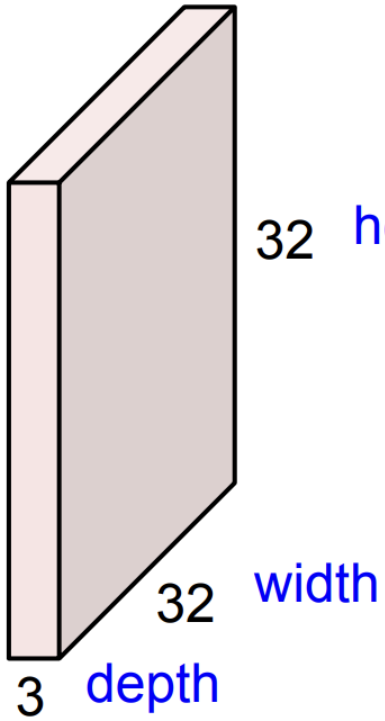
FULLY CONNECTED LAYER

32x32x3 image -> stretch to 3072 x 1



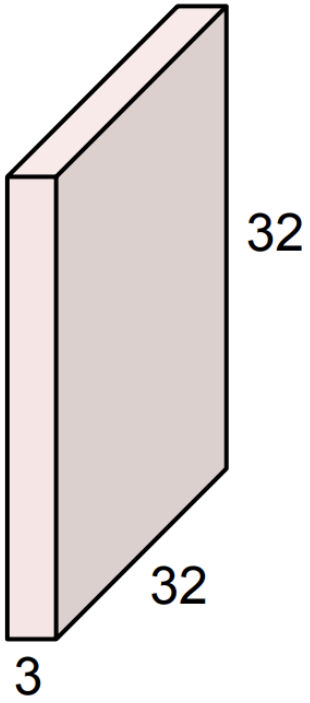
CONVOLUTION LAYER

32x32x3 image -> preserve spatial structure

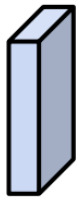


CONVOLUTION LAYER

32x32x3 image



5x5x3 filter

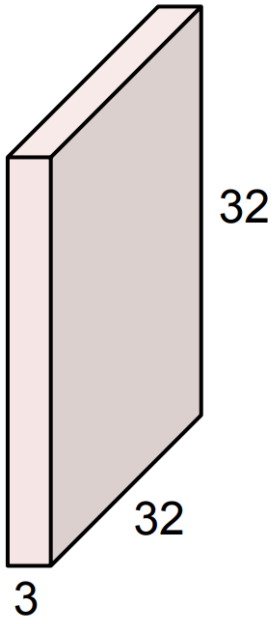


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

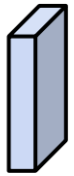
CONVOLUTION LAYER

Filters always extend the full depth of the input volume

32x32x3 image

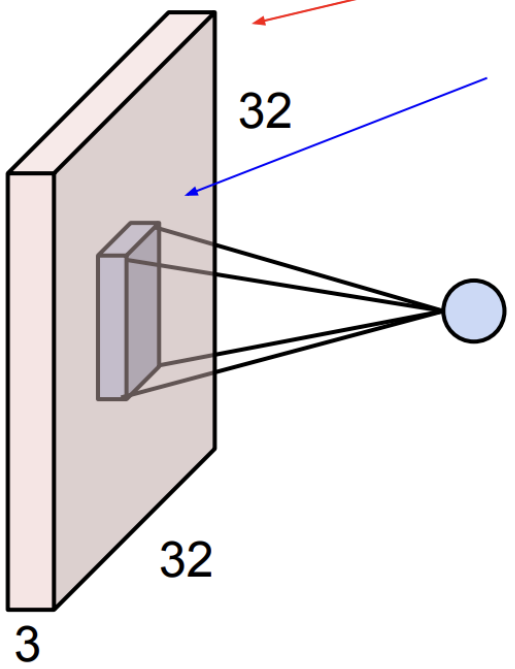


5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

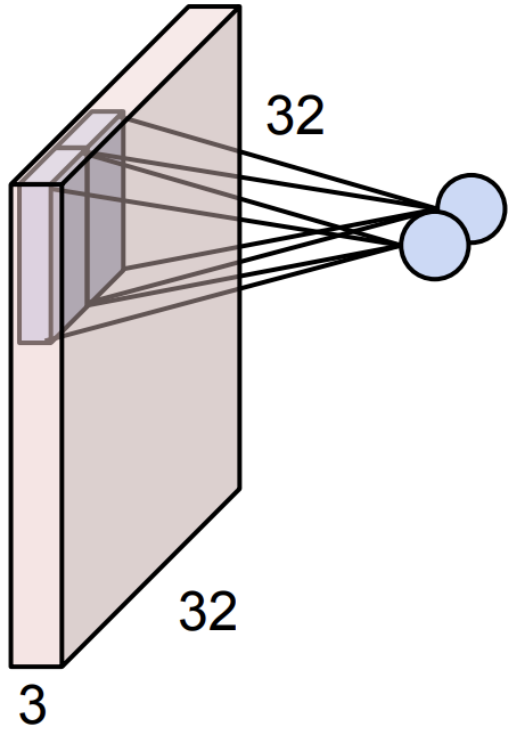
CONVOLUTION LAYER



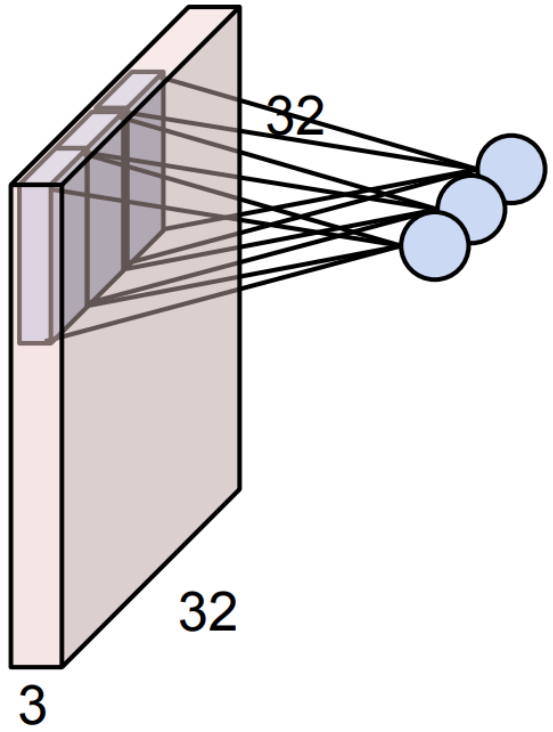
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

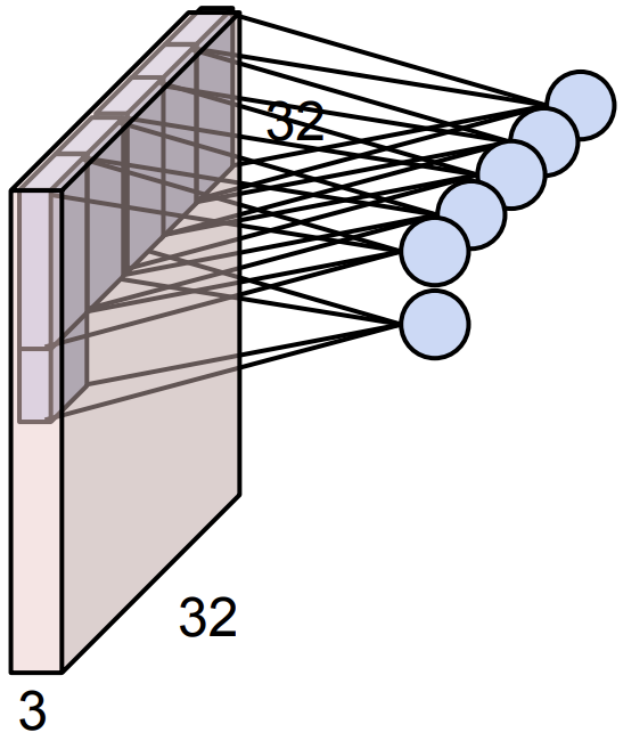
CONVOLUTION LAYER



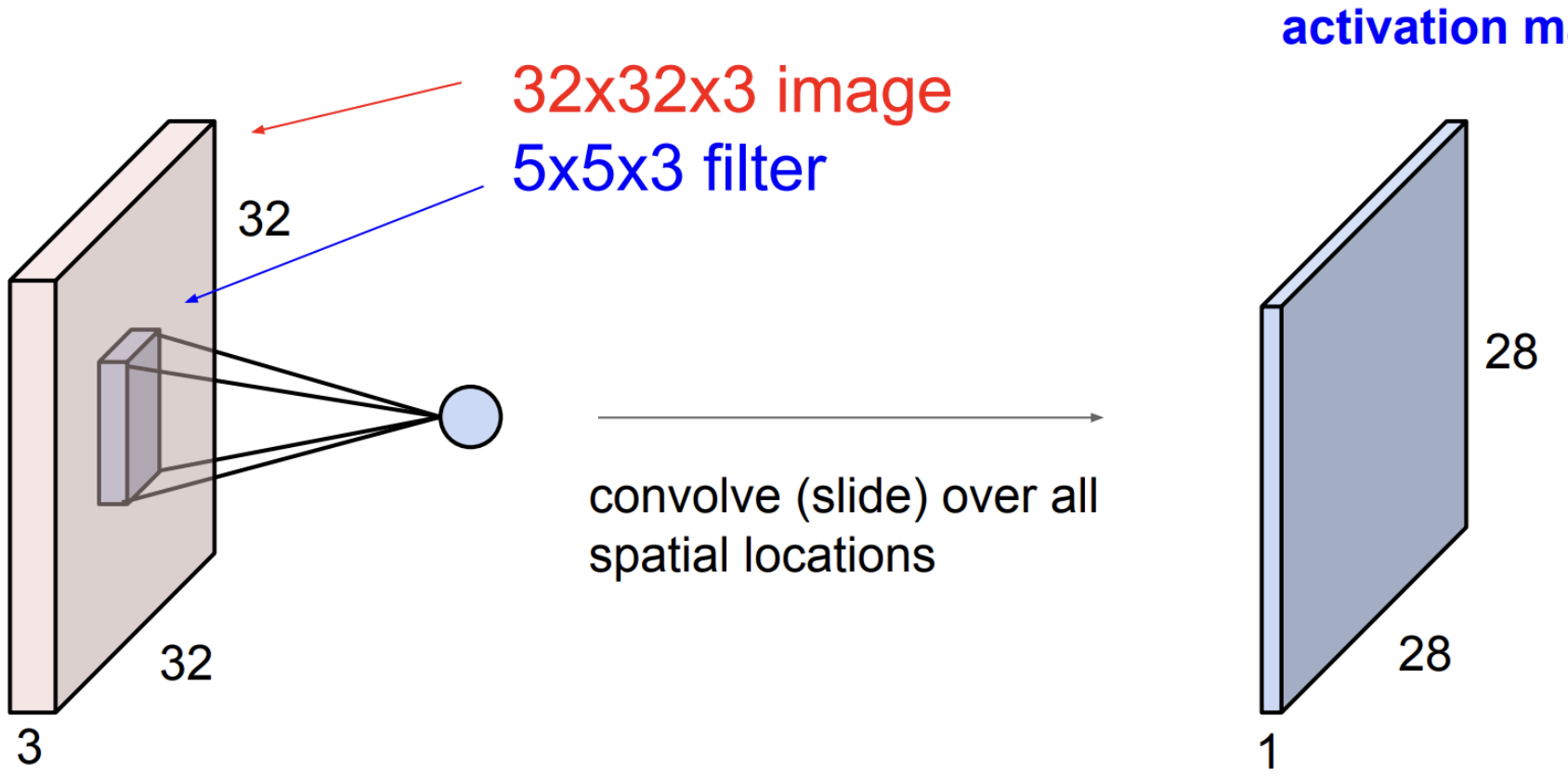
CONVOLUTION LAYER



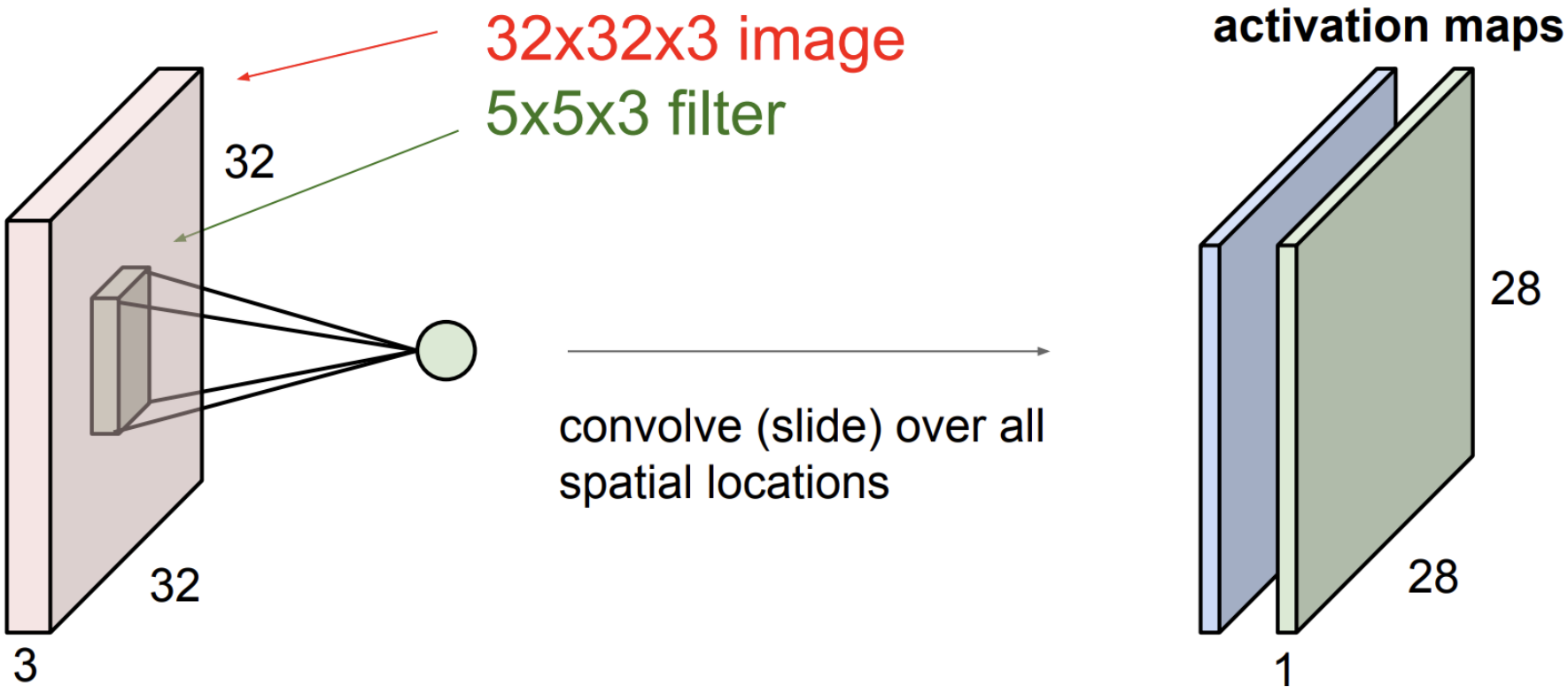
CONVOLUTION LAYER



CONVOLUTION LAYER

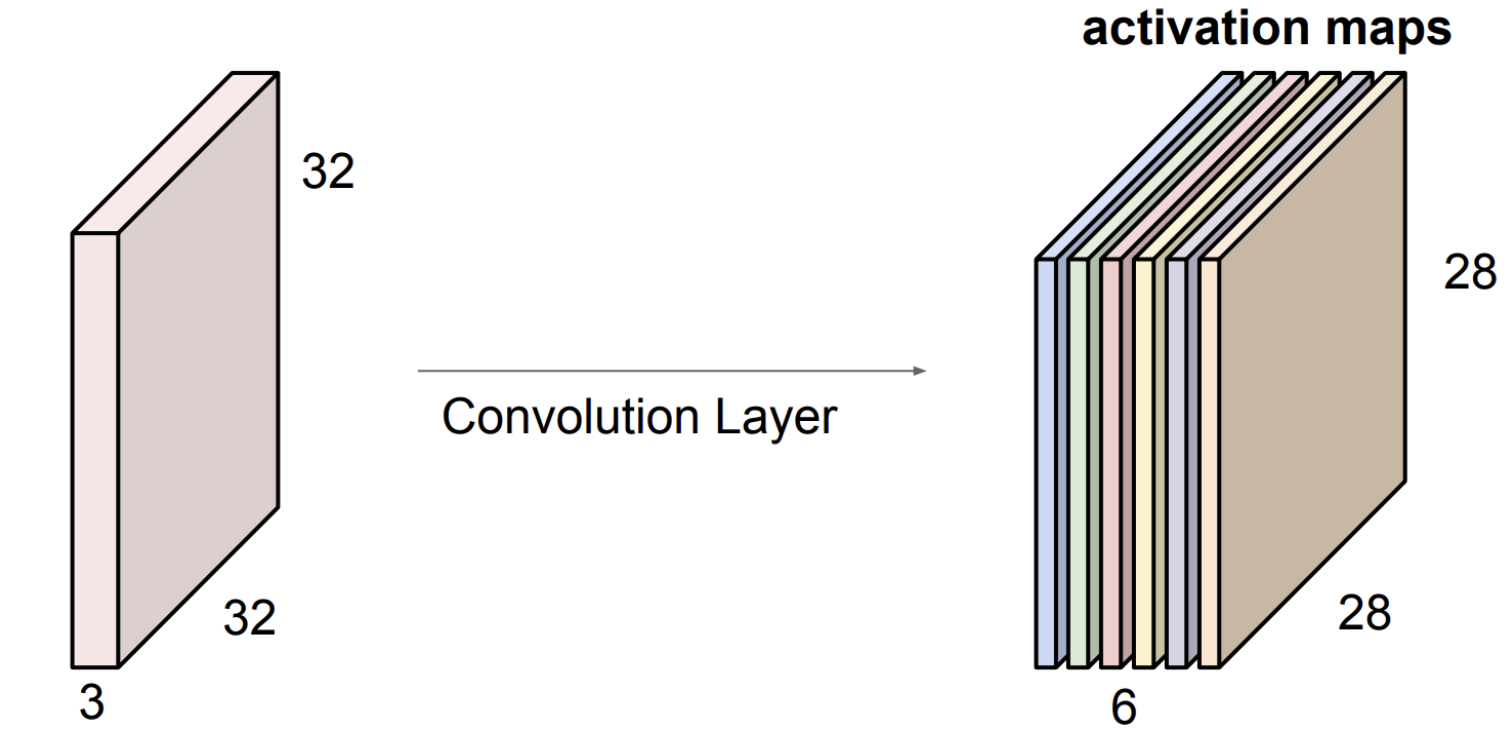


CONVOLUTION LAYER



CONVOLUTION LAYER

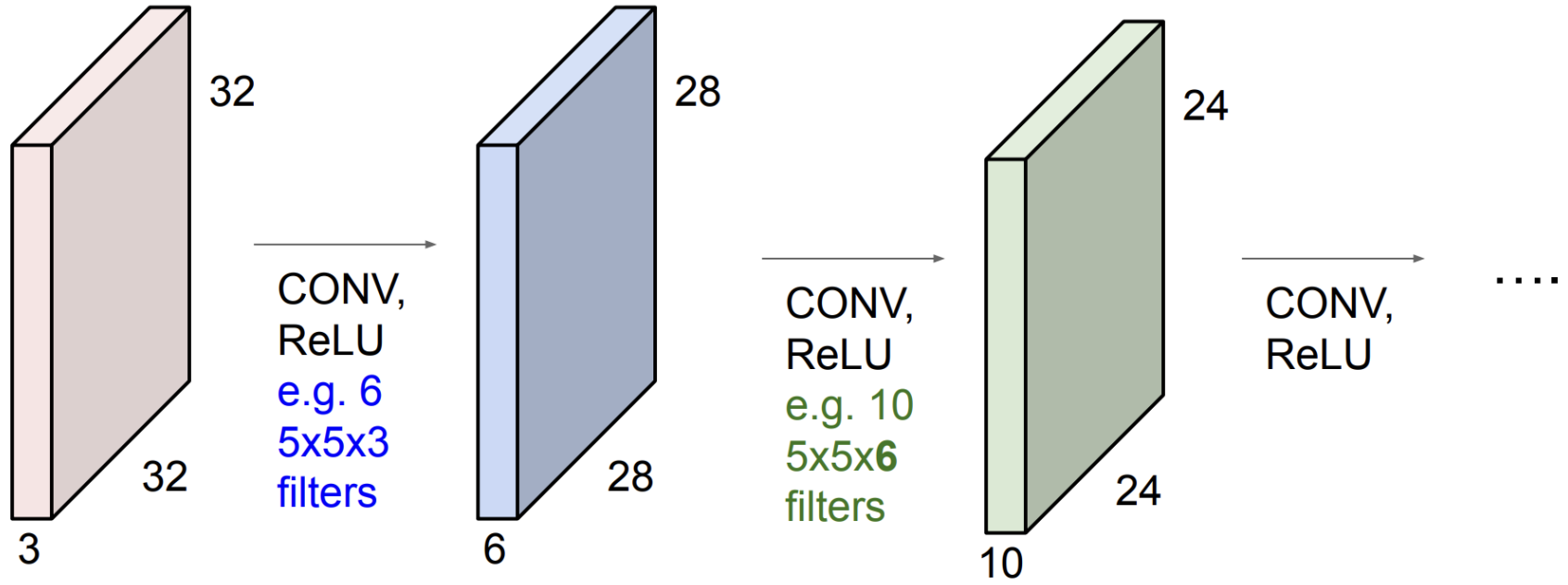
- For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



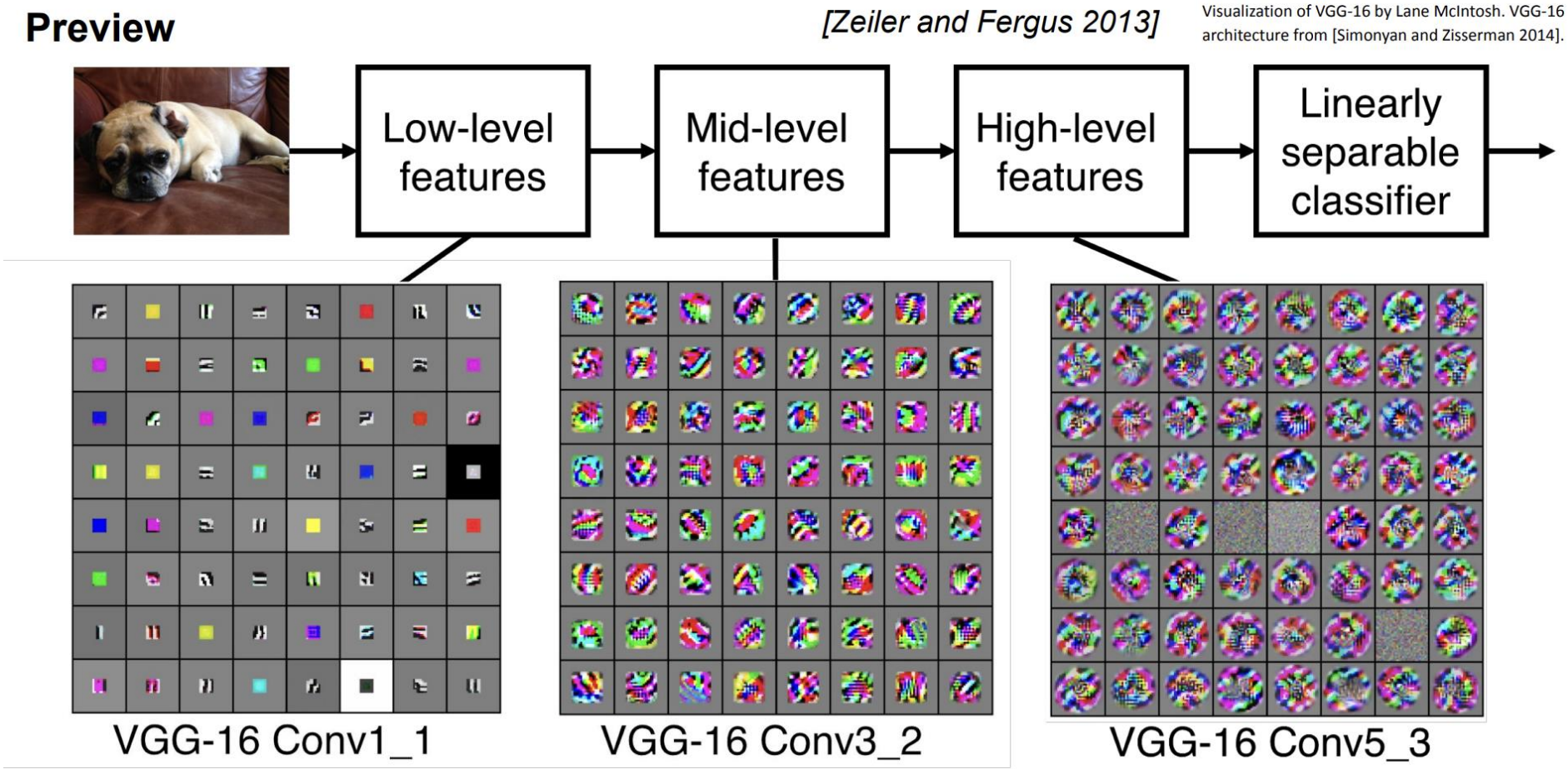
We stack these up to get a "new image" of size 28x28x6!

CNN: SEQUENCE OF CONVOLUTION LAYERS

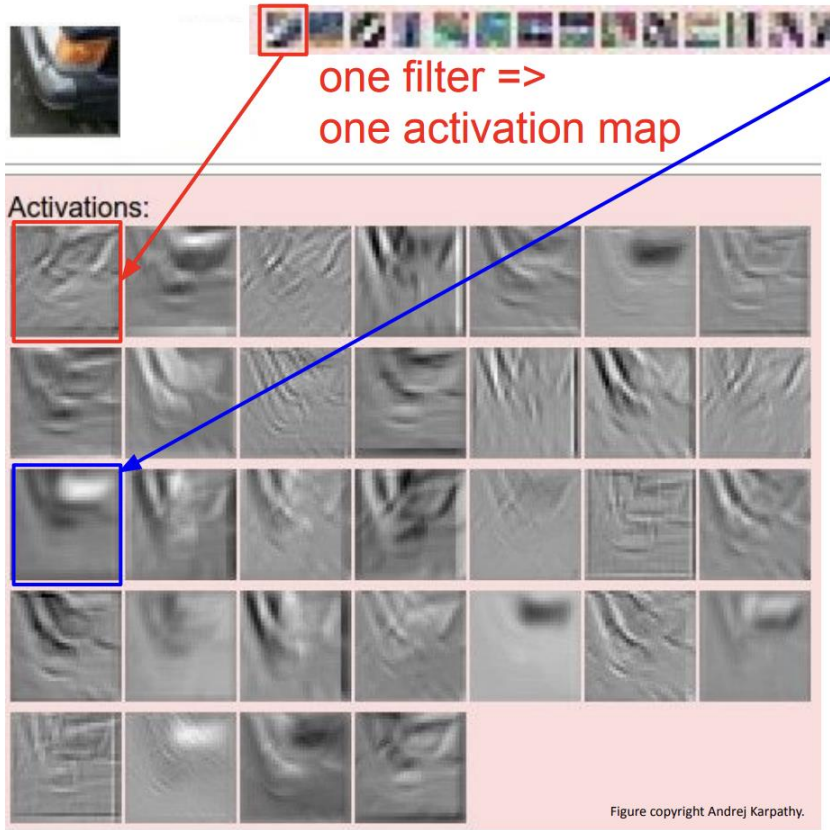
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



HIERARCHICAL FEATURES



EXAMPLE



example 5x5 filters
(32 total)

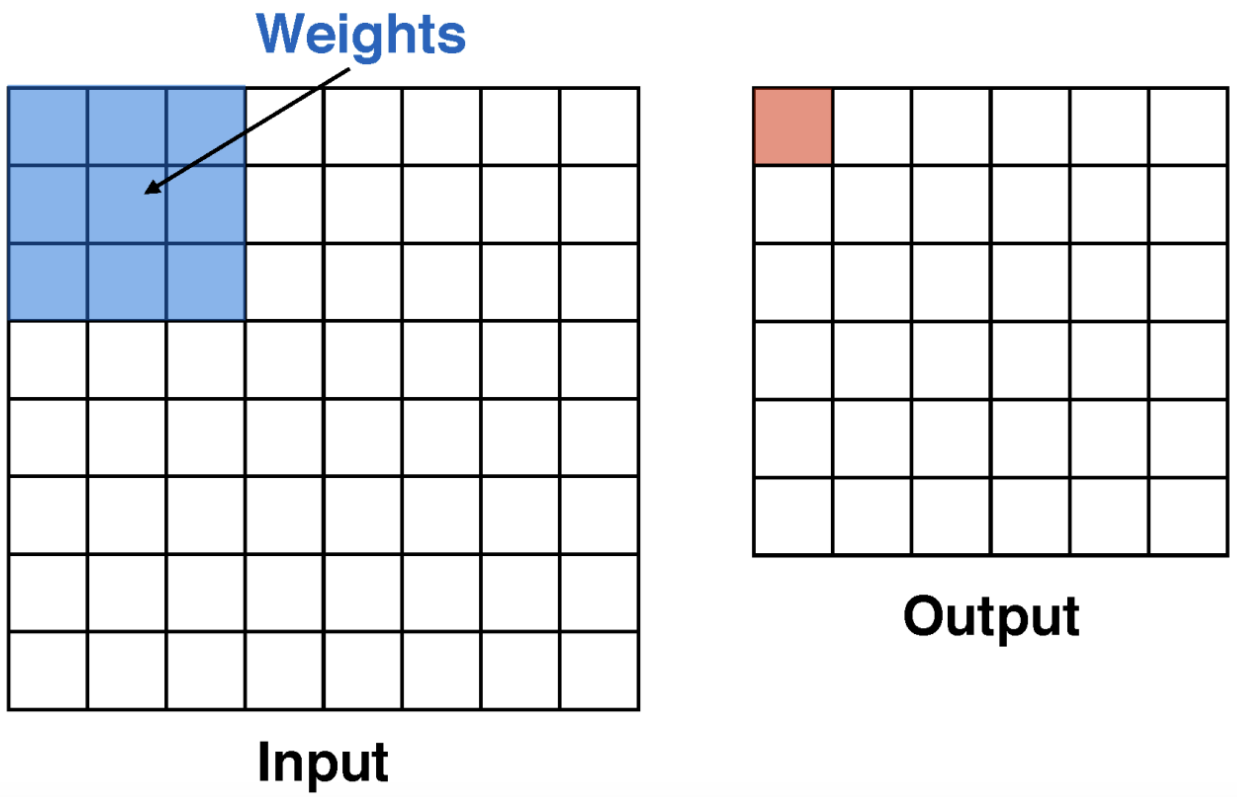
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

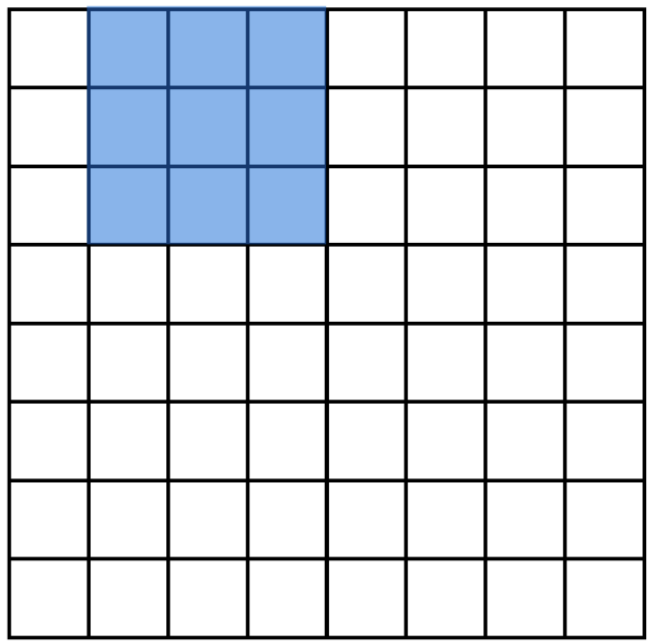
Convolution: Stride

During convolution, the weights “slide” along the input to generate each output

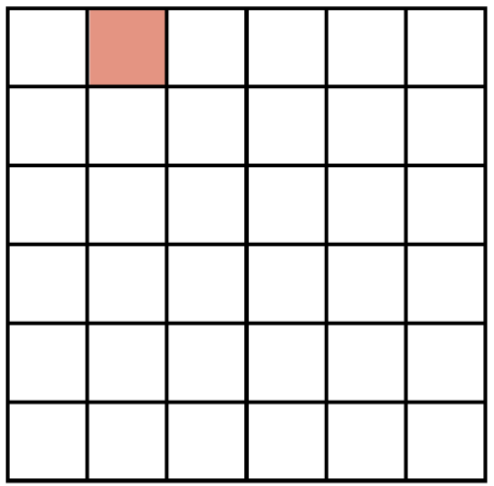


Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



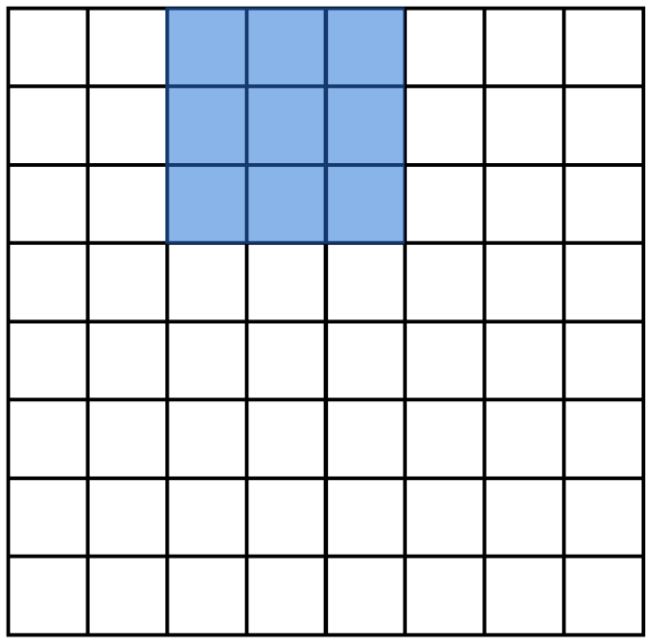
Input



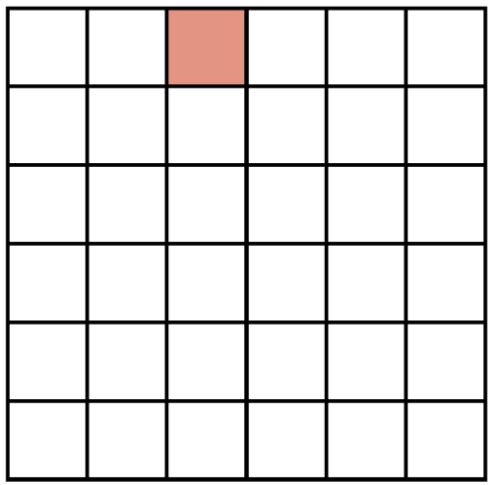
Output

Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



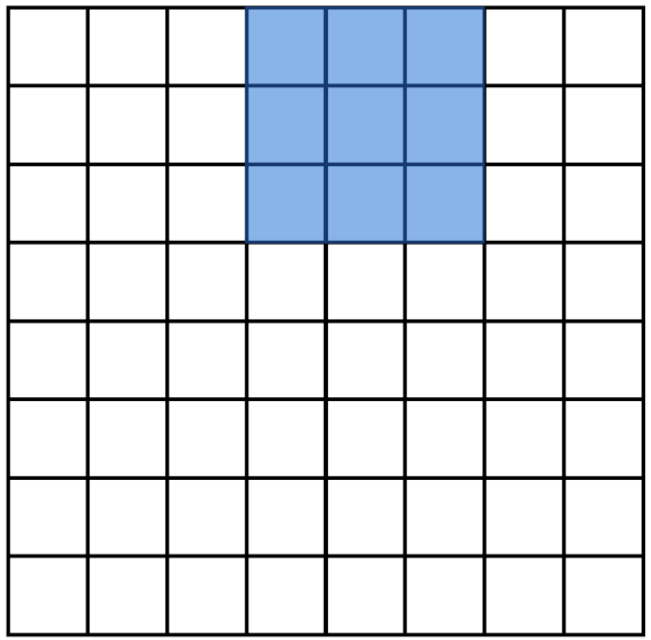
Input



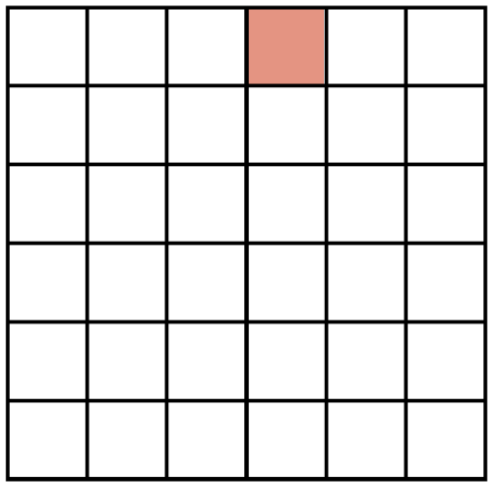
Output

Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



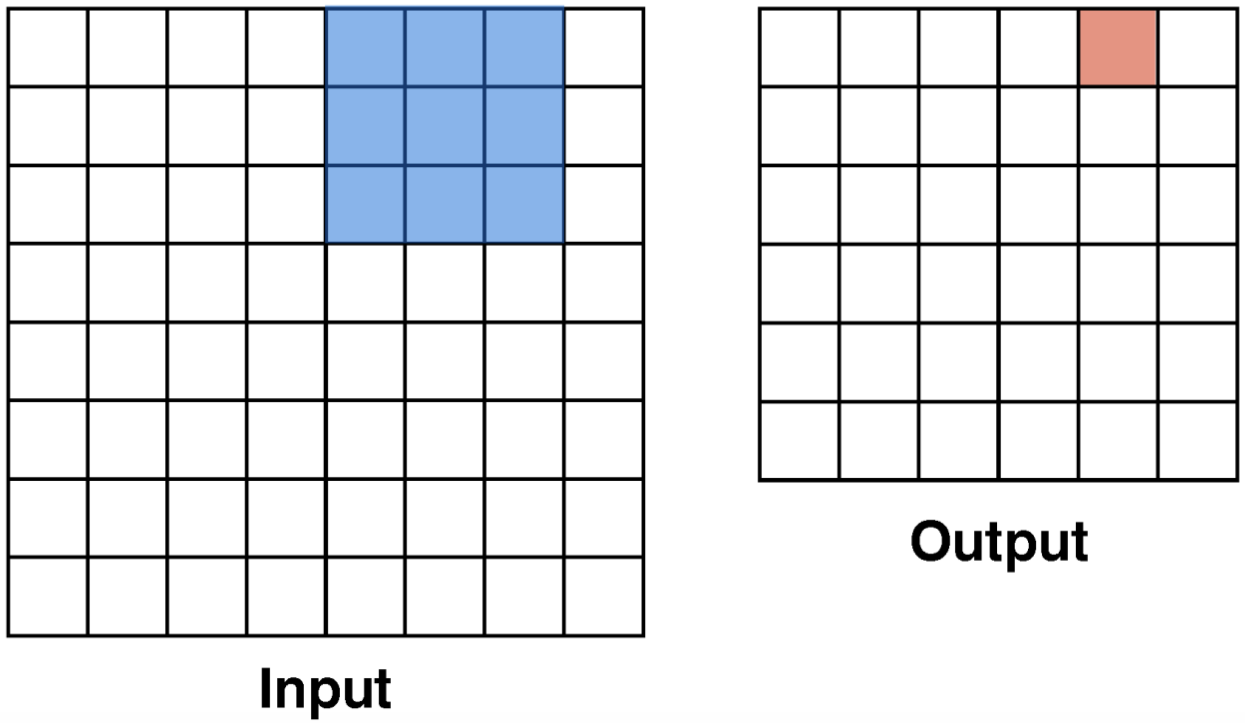
Input



Output

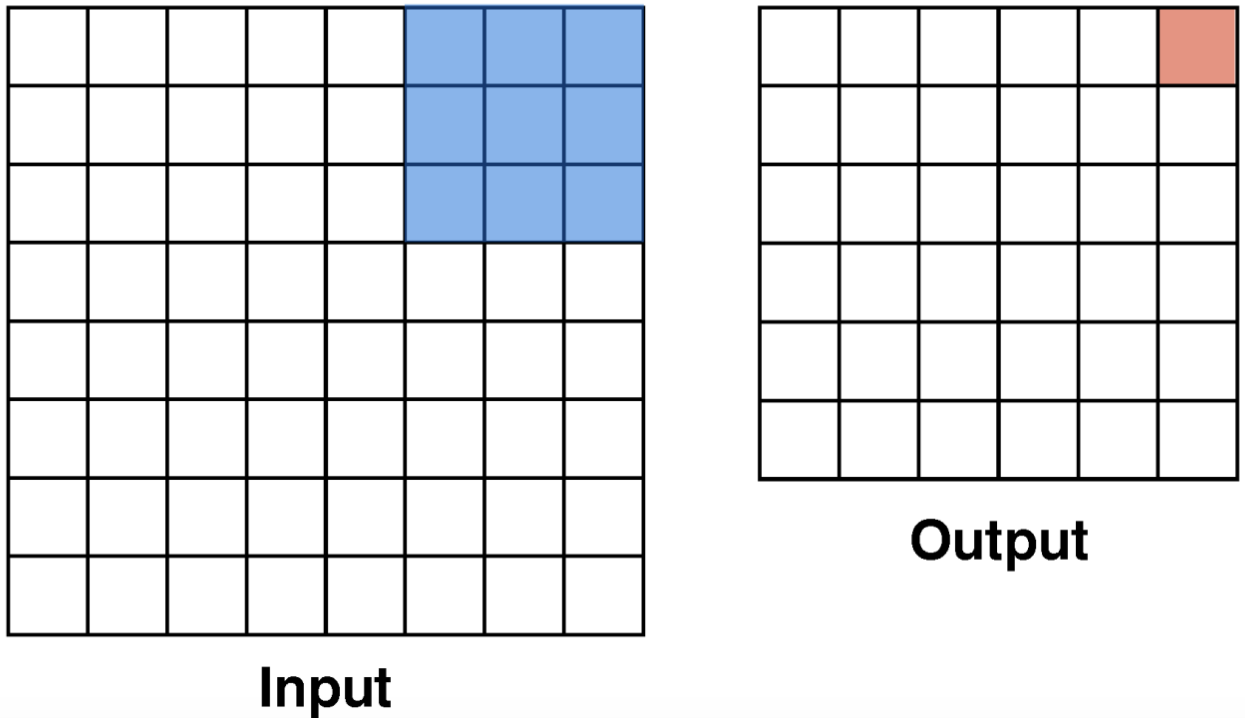
Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



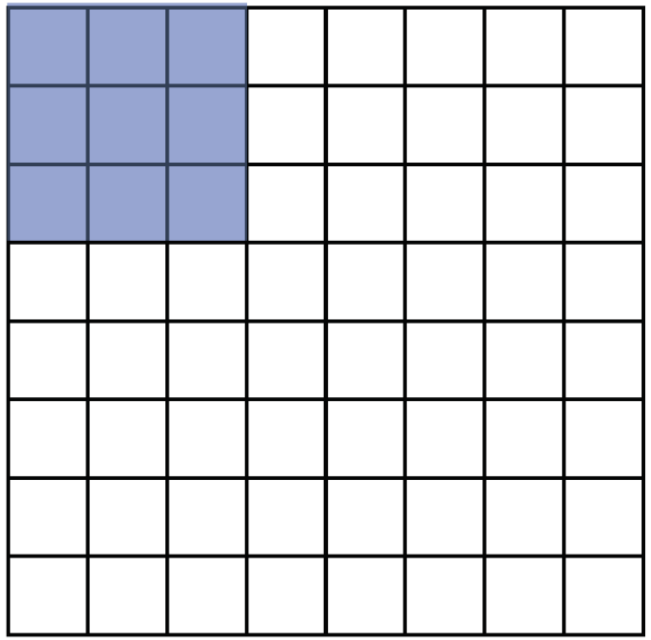
Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



Input

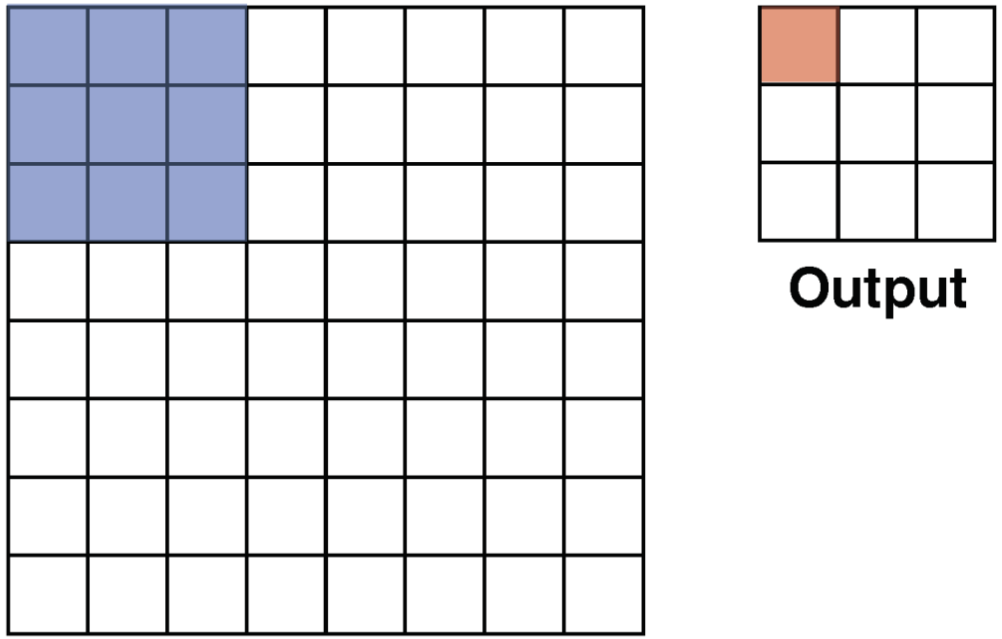
Recall that at each positior we are doing a **3D** sum:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

(channel, row, column)

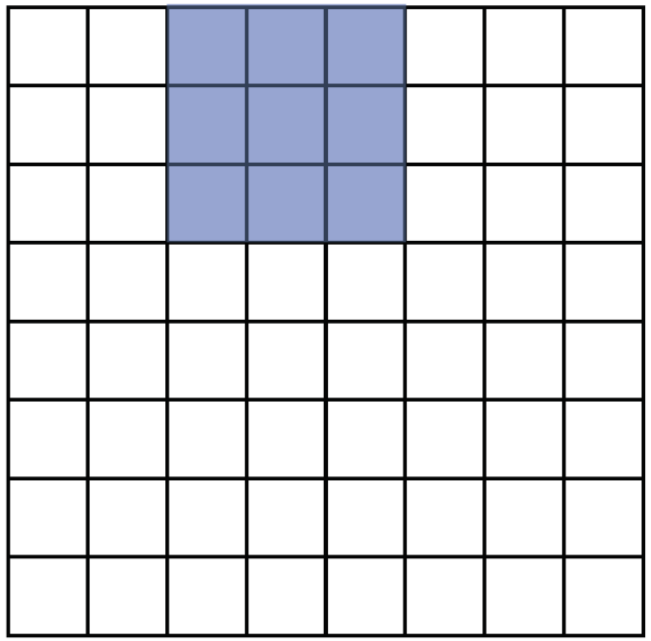
Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2

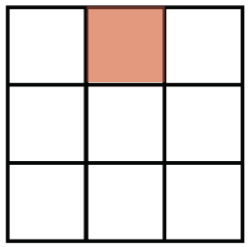


Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



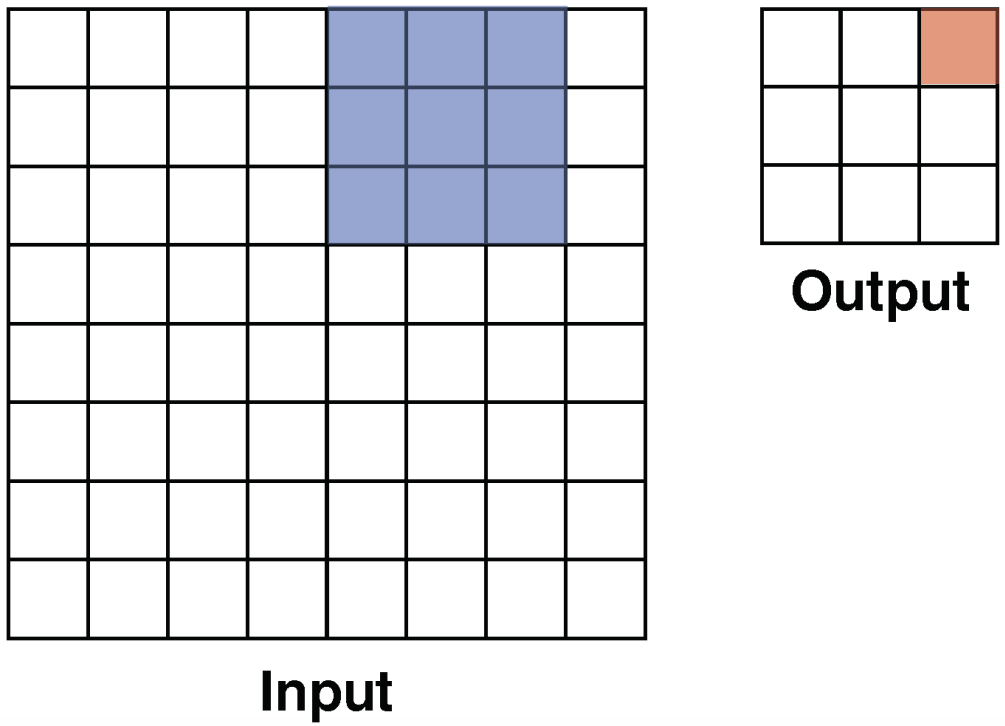
Input



Output

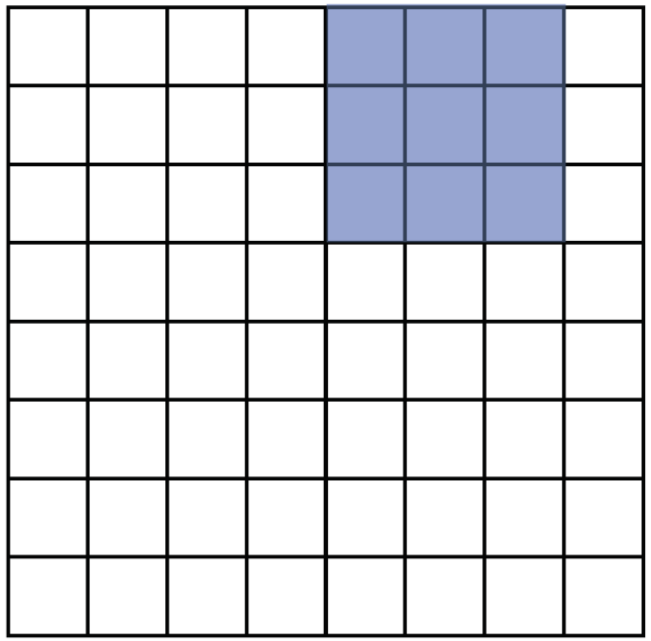
Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2

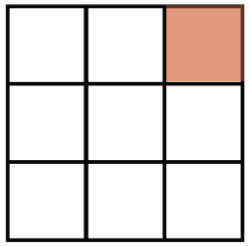


Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



Input



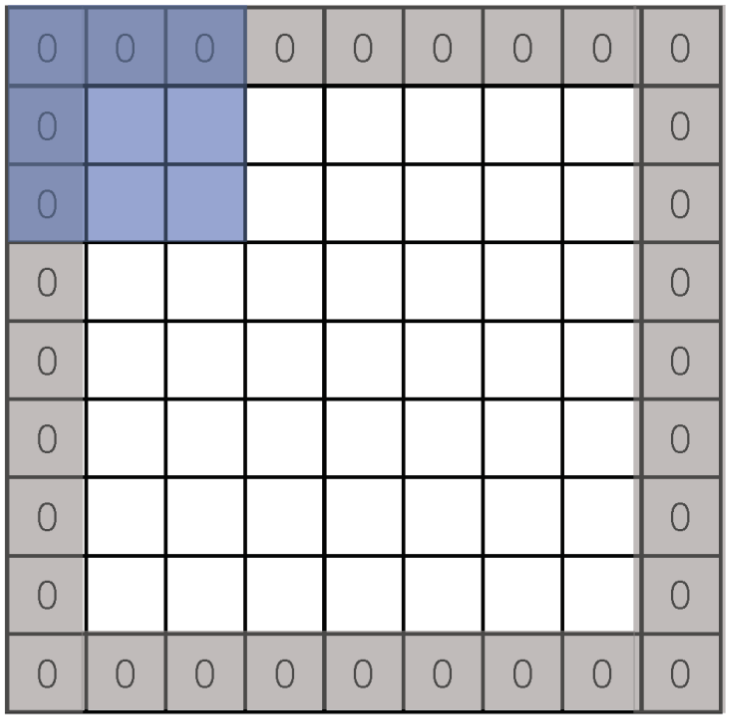
Output

- Notice that with certain strides, we may not be able t cover all of the input

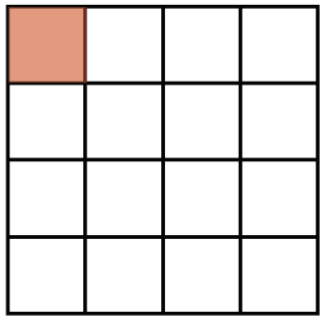
- The output is also half the size of the input

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



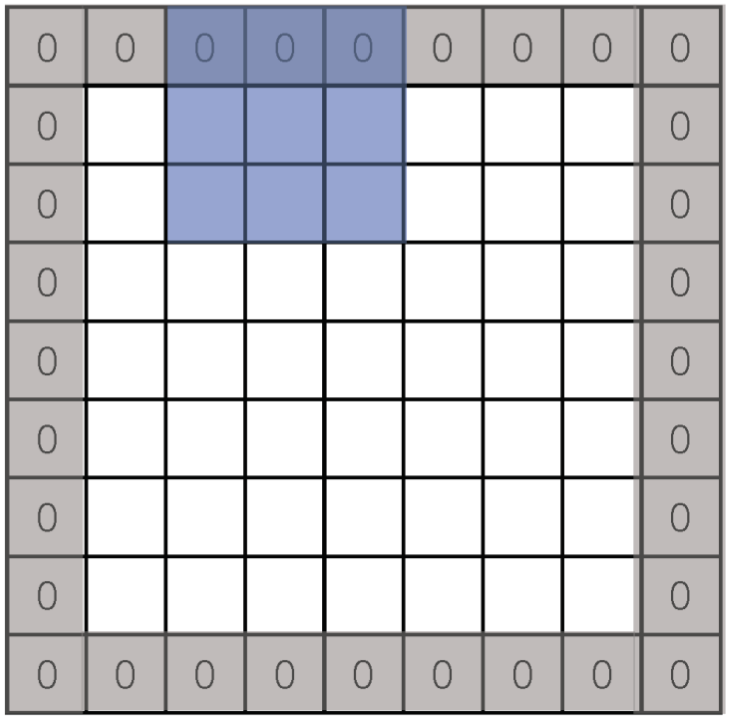
Input



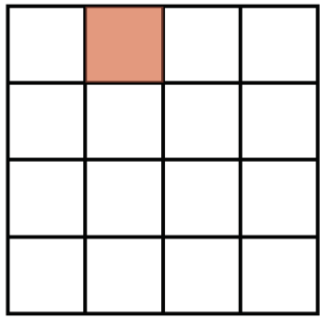
Output

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



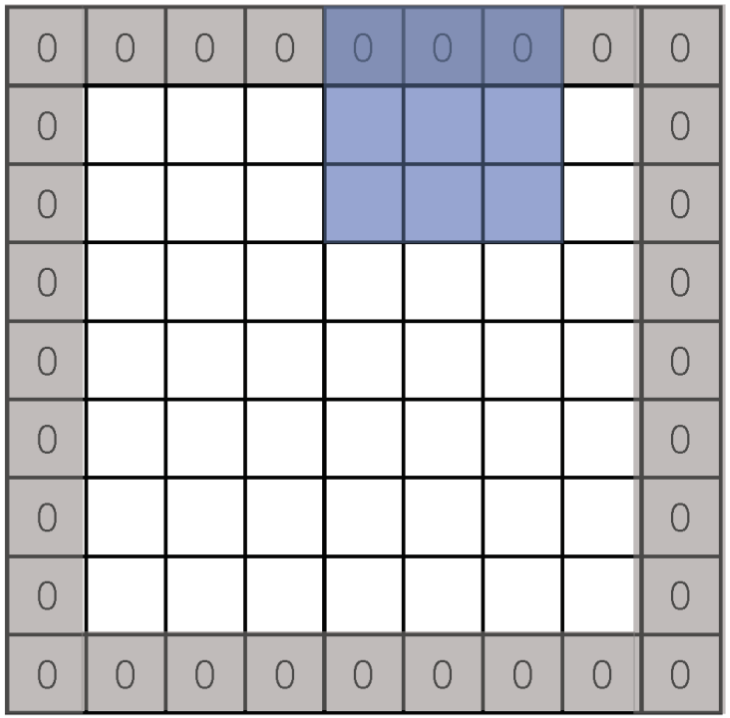
Input



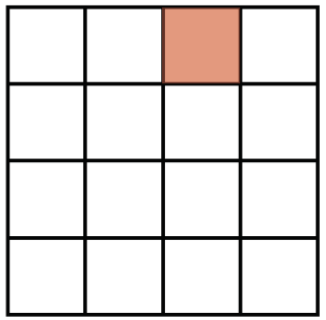
Output

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



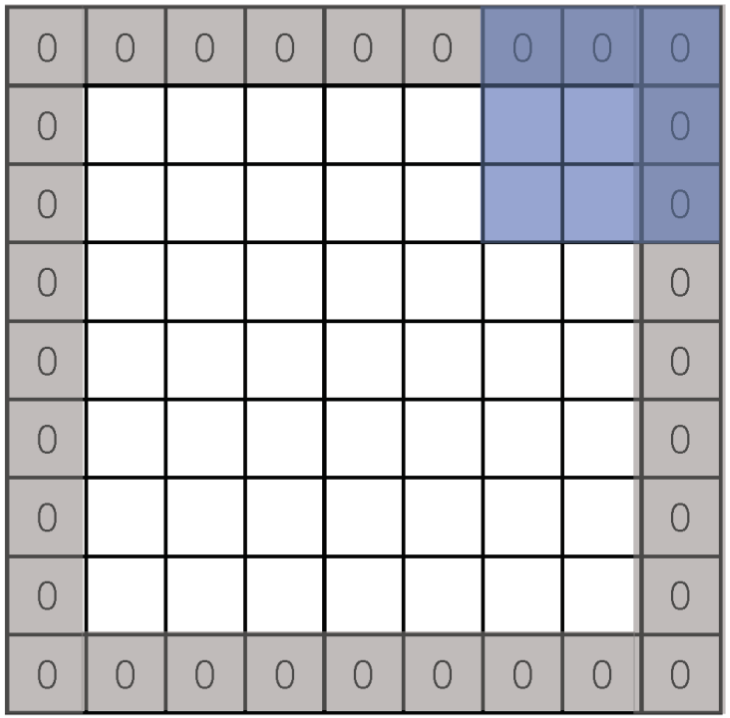
Input



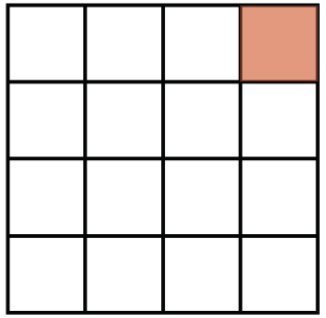
Output

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



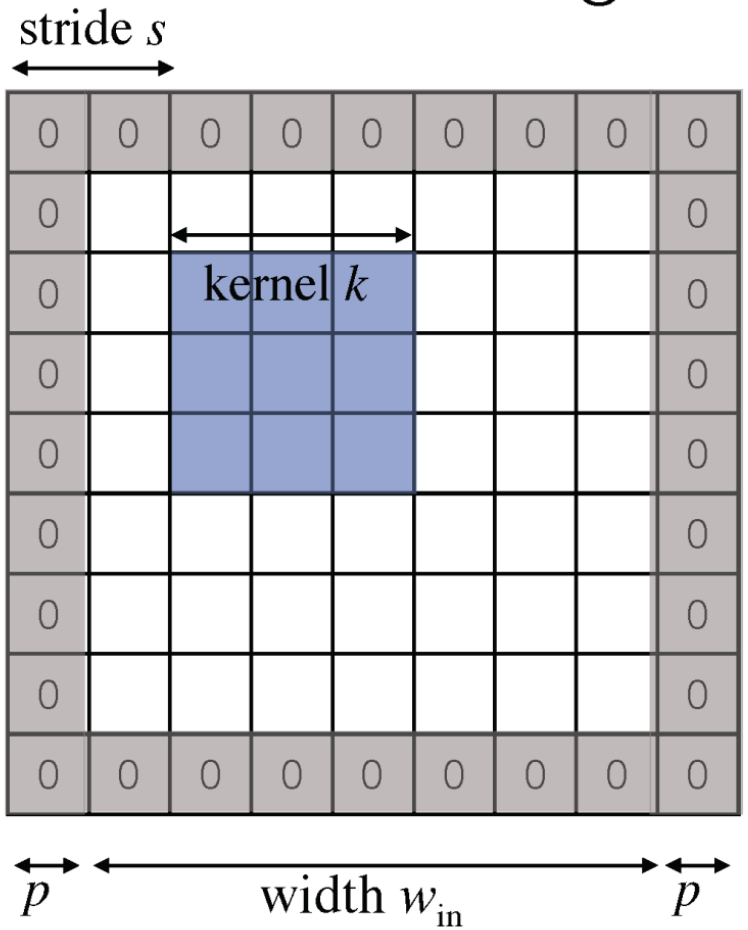
Input



Output

Convolution:

How big is the output?

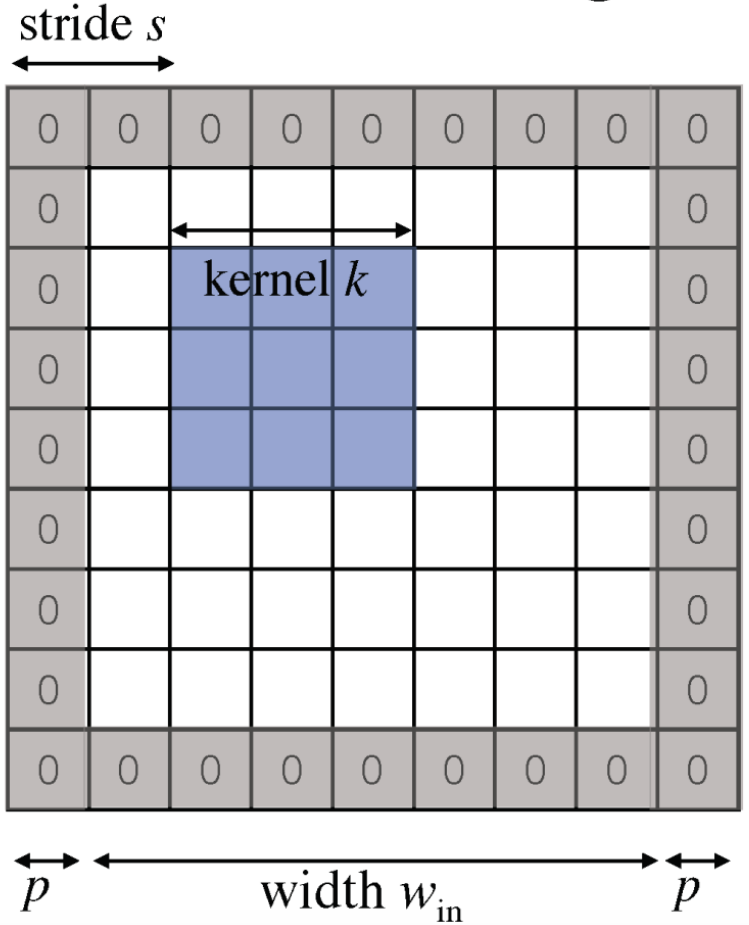


In general, the output has size:

$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$

Convolution:

How big is the output?



Example: k=3, s=1, p=1

$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$

$$= \left\lfloor \frac{w_{in} + 2 - 3}{1} \right\rfloor + 1$$

$$= w_{in}$$

VGGNet [Simonyan 2014] uses filters of this shape

Pooling

For most ConvNets, **convolution** is often followed by **pooling**:

- Creates a smaller representation while retaining the most important information
- The “max” operation is the most common
- Why might “avg” be a poor choice?

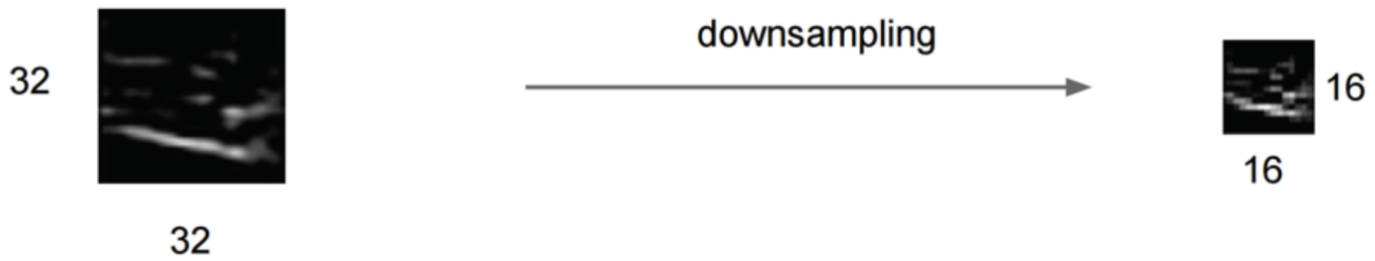
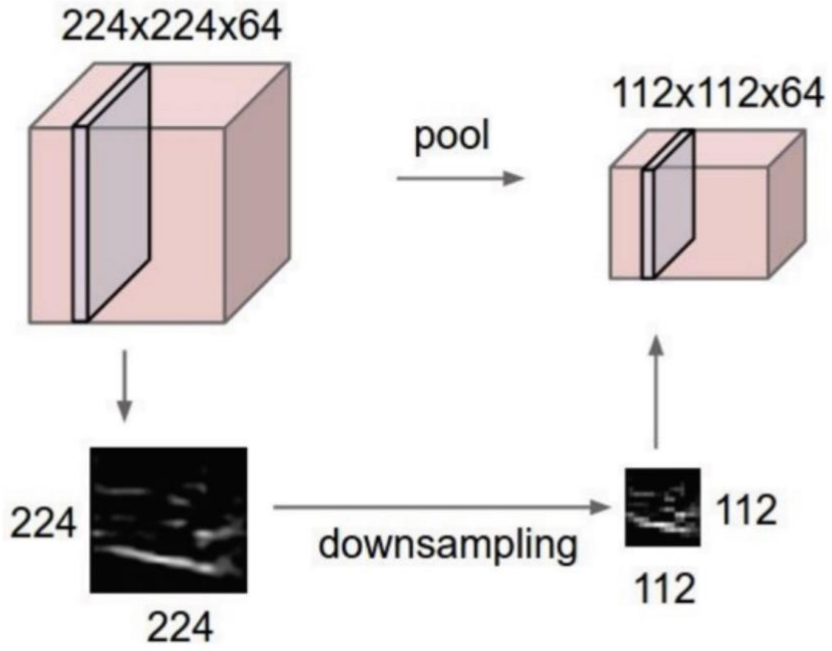


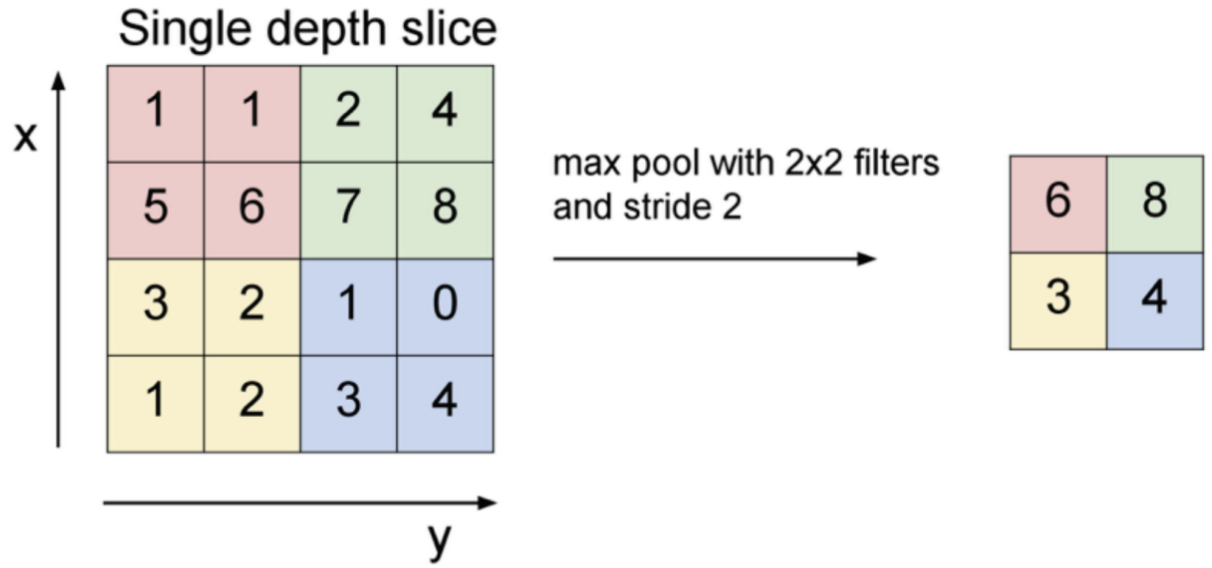
Figure: Andrej Karpathy

Pooling

- makes the representations smaller and more manageable
- operates over each activation map independently:



Max Pooling



What's the backprop rule for max pooling?

- In the forward pass, store the index that took the max
- The backprop gradient is the input gradient at that index

Figure: Andrej Karpathy

Example ConvNet

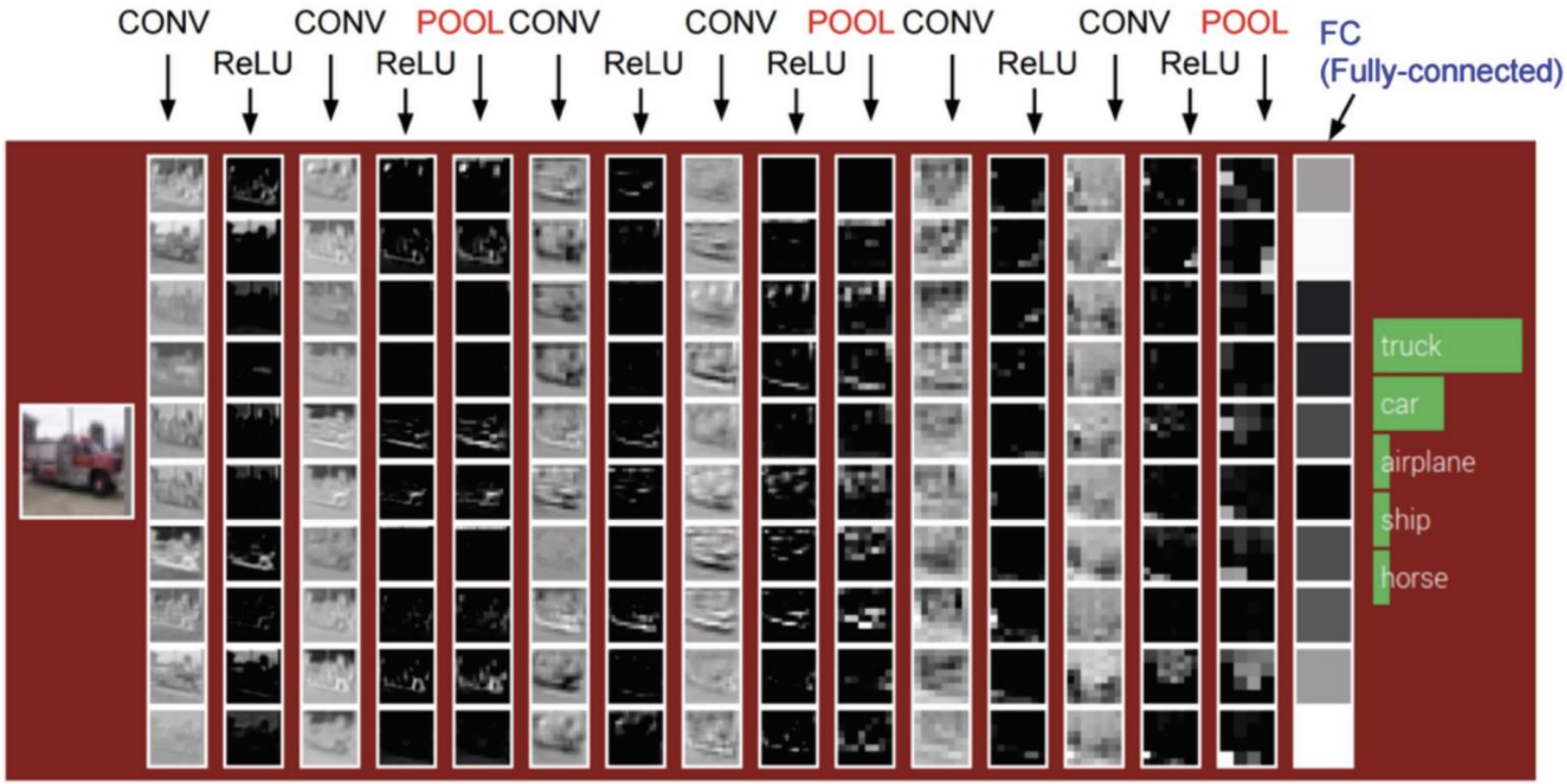


Figure: Andrej Karpathy

RESOURCES

- **CIFAR-10**
- https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

- **MNIST**
- https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

IMAGE CAPTIONING WITH VISUAL ATTENTION

- We can similarly use attention for image captioning (image → text)
- Builds directly on previous work

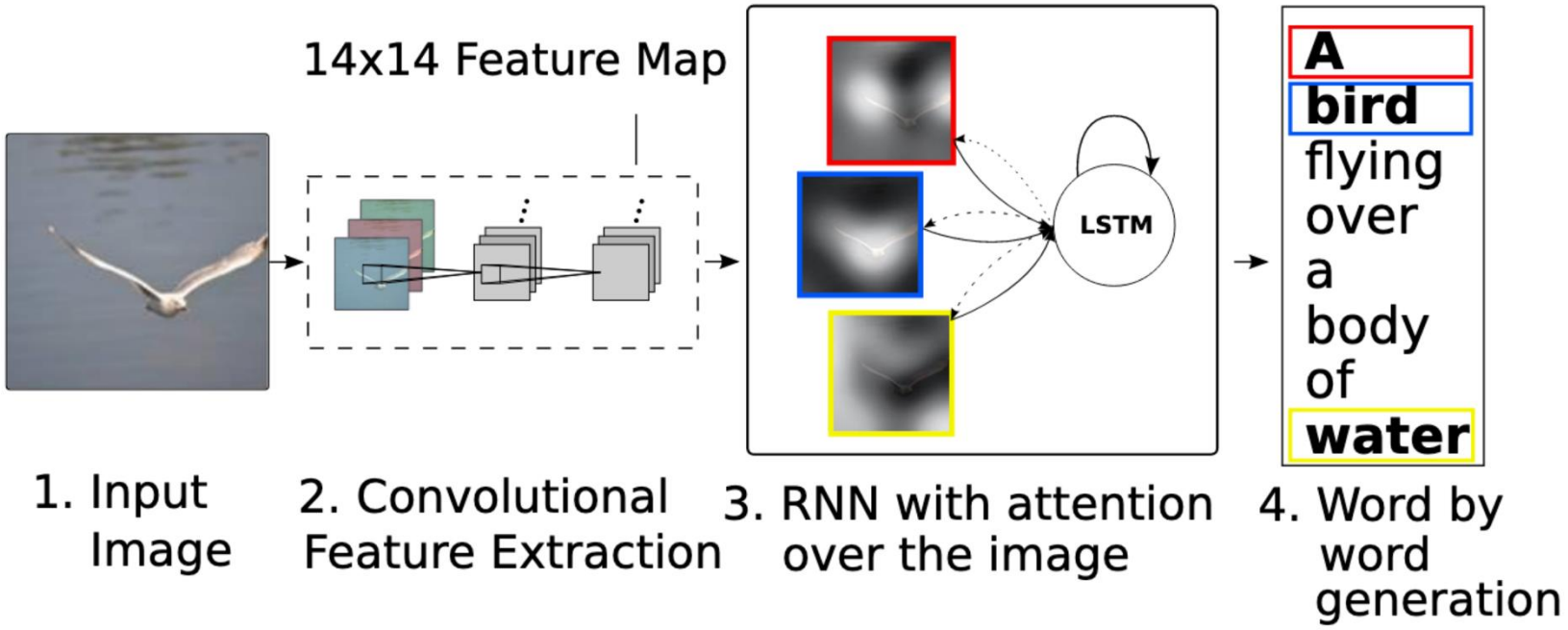
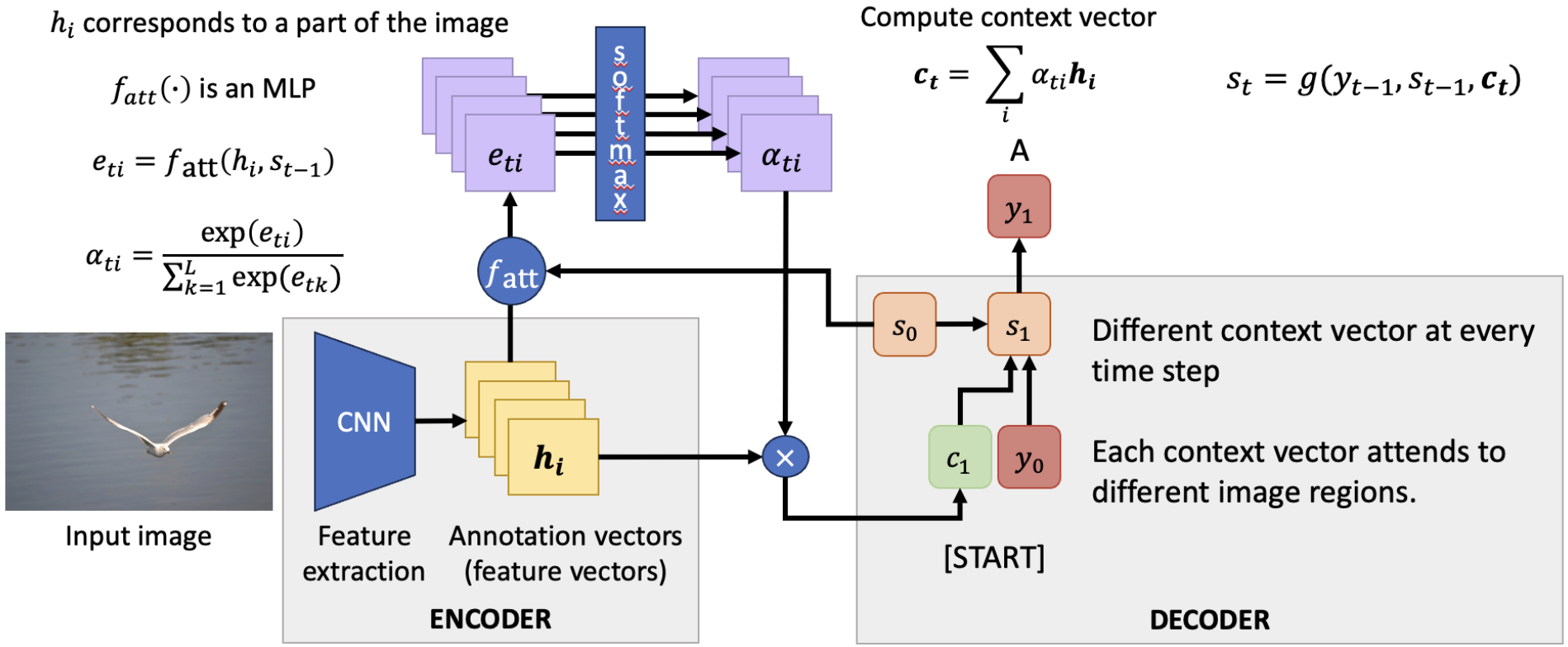
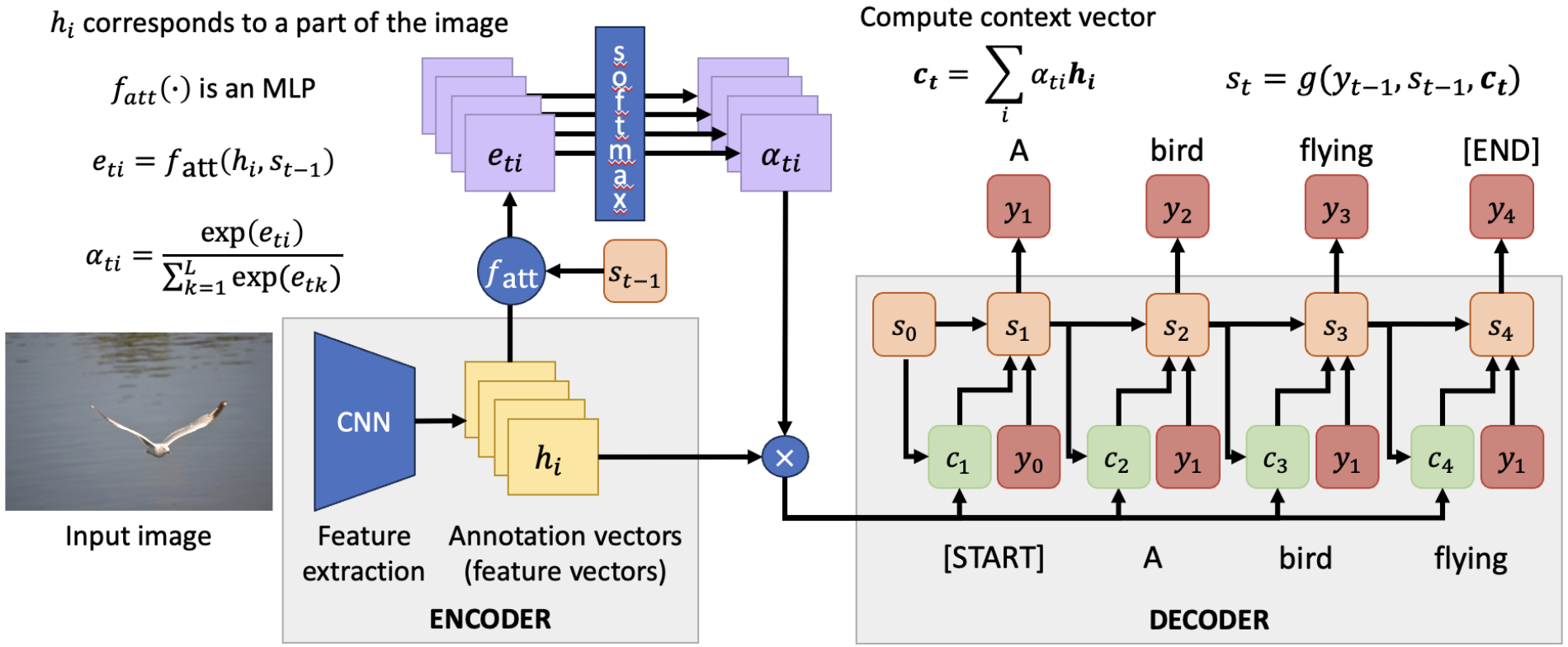


IMAGE CAPTIONING WITH VISUAL ATTENTION



K. Xu *et al.*, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," in *PMLR*, 2015, pp. 2048–2057.

IMAGE CAPTIONING WITH VISUAL ATTENTION

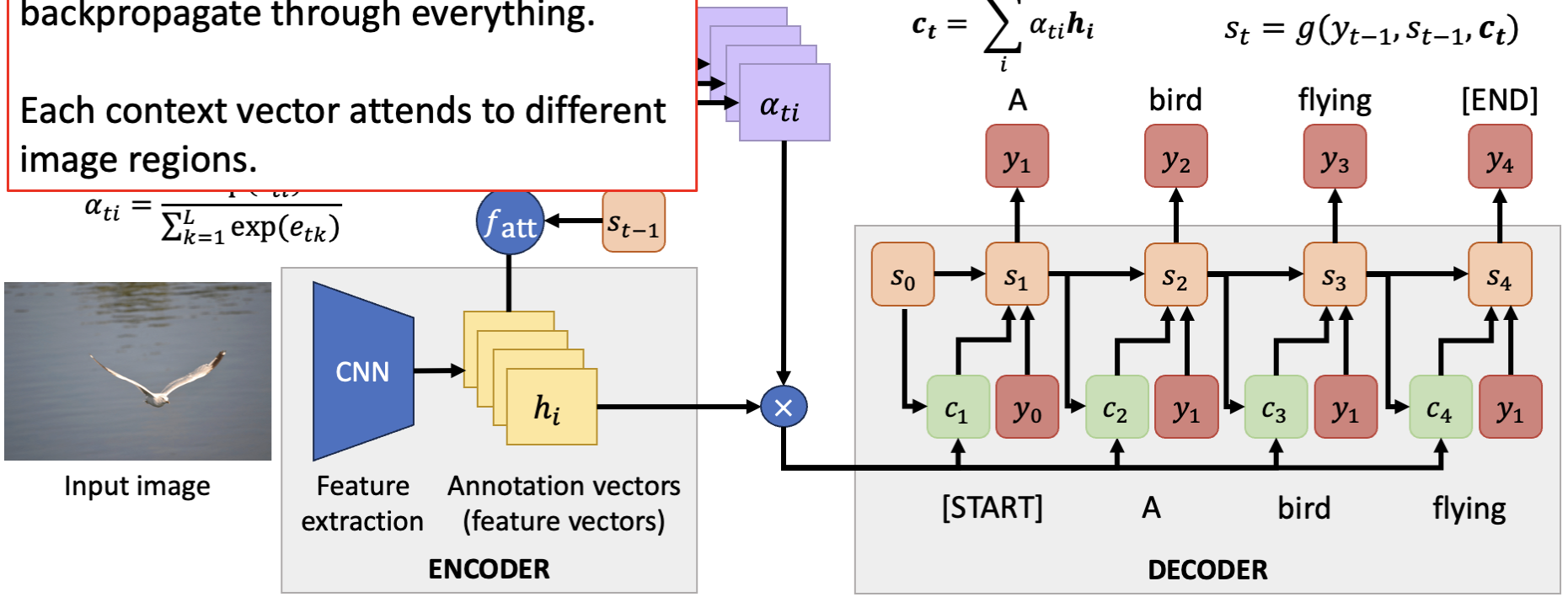


K. Xu *et al.*, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," in *PMLR*, 2015, pp. 2048–2057.

IMAGE CAPTIONING WITH VISUAL ATTENTION

All steps are **differentiable**, so we can backpropagate through everything.

Each context vector attends to different image regions.



K. Xu *et al.*, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," in *PMLR*, 2015, pp. 2048–2057.

IMAGE CAPTIONING WITH VISUAL ATTENTION

- Visualization of the attention for each generated word
  - Gives insight to “where” and “what” the attention focused on when generating each word

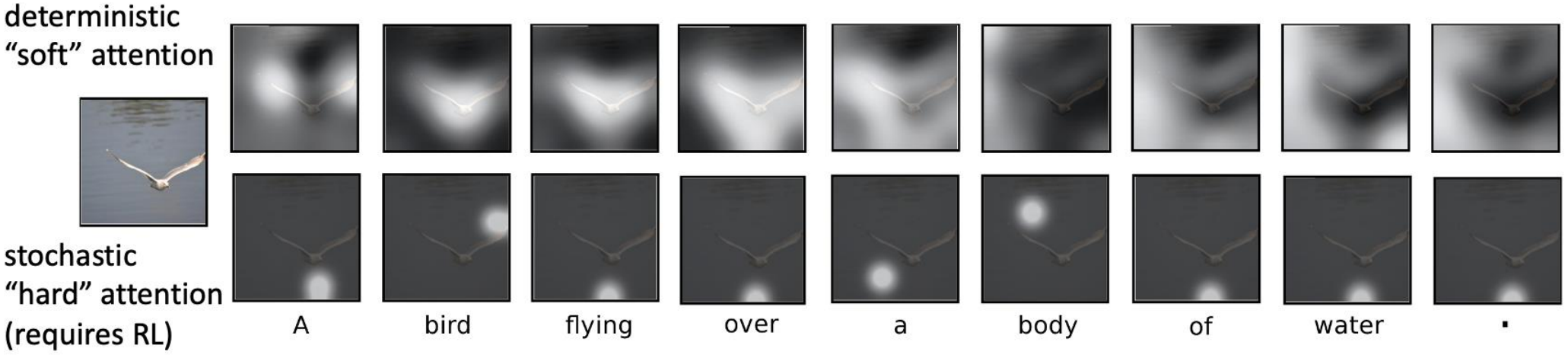


IMAGE CAPTIONING WITH VISUAL ATTENTION

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

K. Xu *et al.*, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," in *PMLR*, 2015, pp. 2048–2057.

RESOURCES

- https://medium.com/@deepeshrishu09/automatic-image-captioning-with-pytorch-cf576c98d319
- https://www.kaggle.com/code/mdteach/image-captioning-with-attention-pytorch