Neural Networks for Visual Recognition

Gül Varol

IMAGINE team, École des Ponts ParisTech

gul.varol@enpc.fr

http://imagine.enpc.fr/~varolg/

@RecVis, 31.10.2023

École des Ponts ParisTech

With many slides from: J. Sivic, R. Fergus, A. Vedaldi, A. Zisserman, S. Lazebnik, A. Karpathy, L. Fei Fei, M. Aubry, N. Murray, J. Johnson, K. Derpanis, S. Yeung, A. Amini, F. Fleuret

Announcements

Assignment 2 due Tuesday Nov 14

http://imagine.enpc.fr/~varolg/teaching/recvis23/

Neural Networks

Last week: Introduction to neural networks

(A. Joulin)

This week: Neural networks for visual recognition

(G. Varol)

Next week: Beyond classification: Object detection, Segmentation, Human pose estimation

(G. Varol)

First words that come to your mind when hearing "neural networks for visual recognition"?

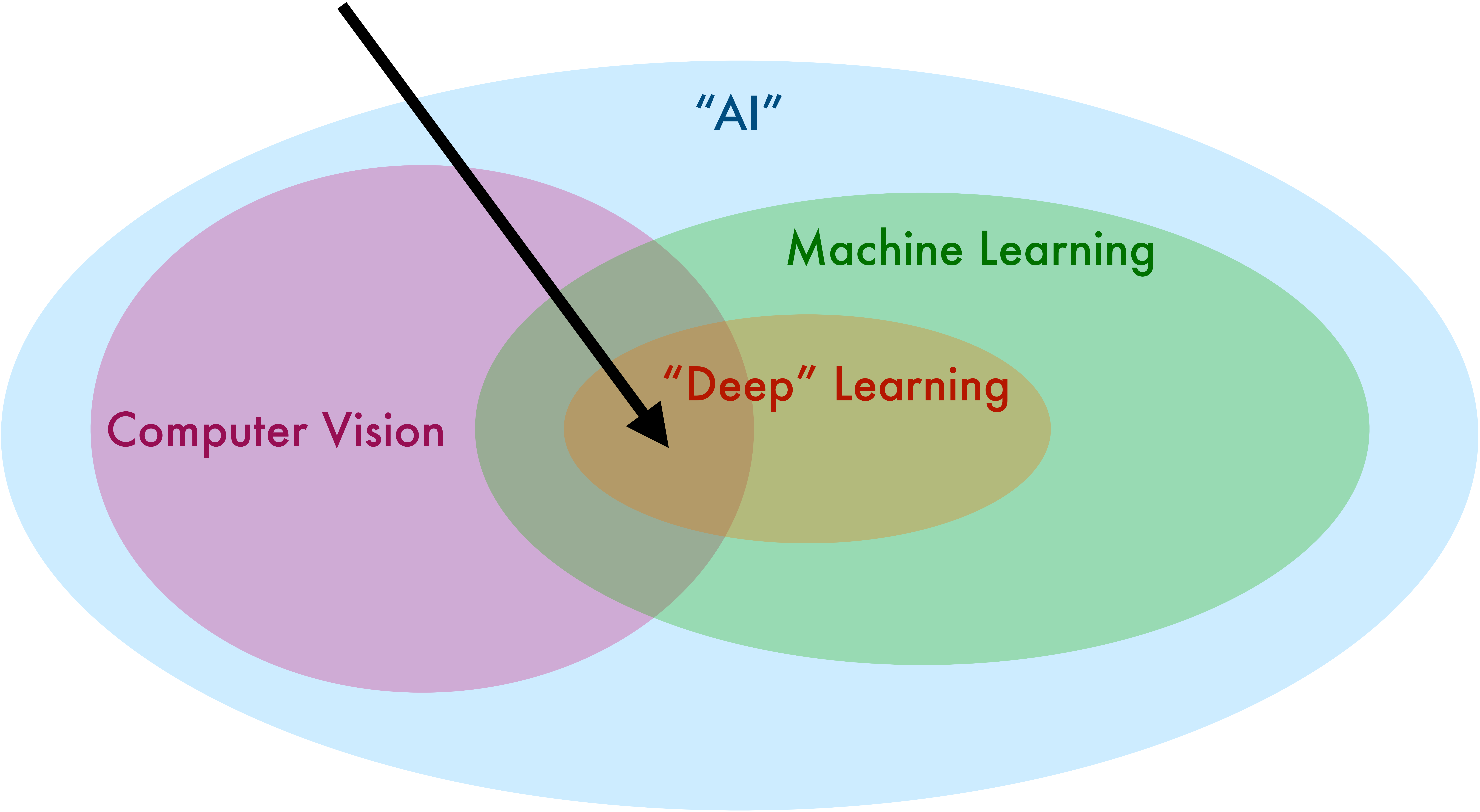
slido.com

#2557 457

Disclaimer: Terminology

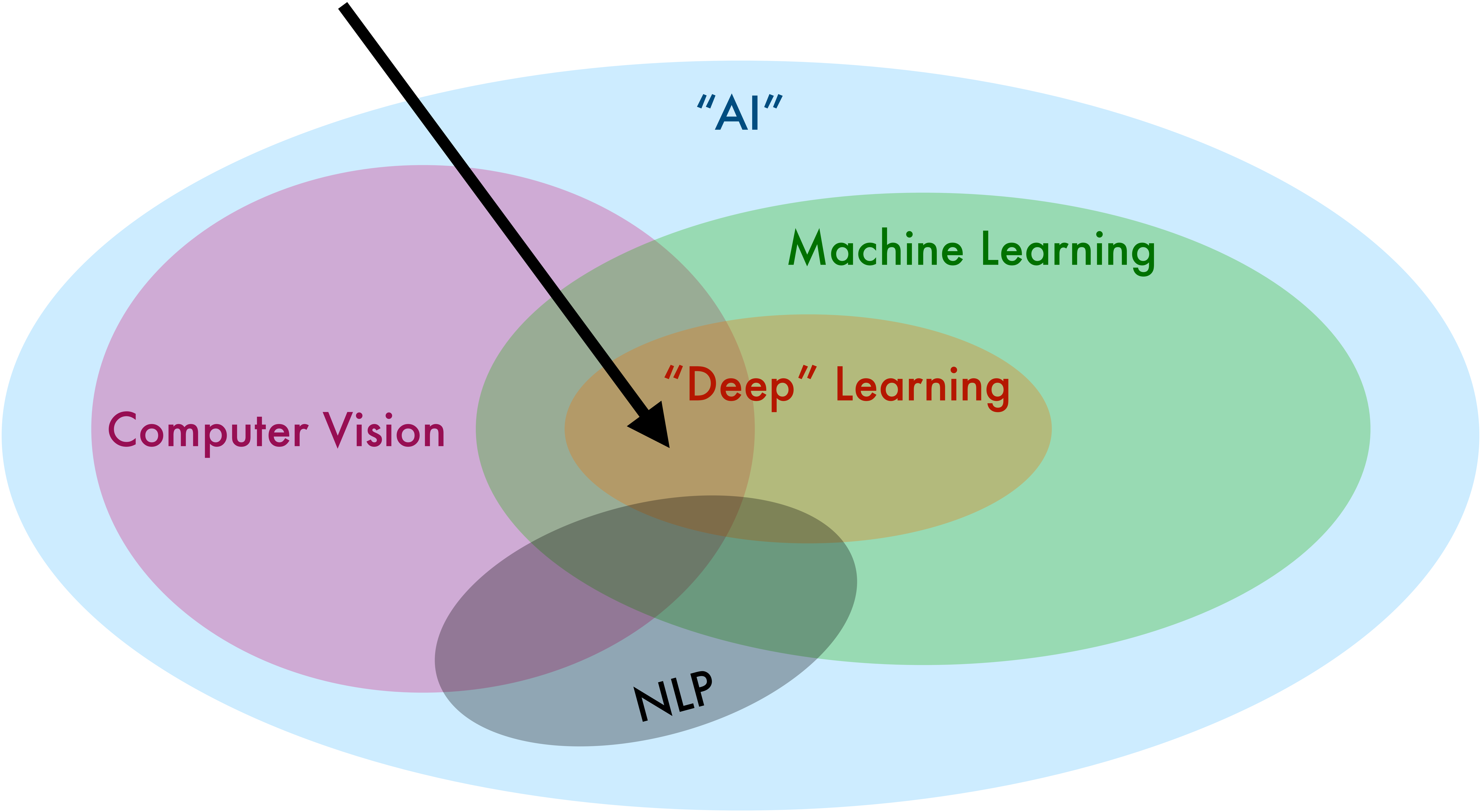
- ~~Deep learning~~
  - Neural networks?
  - Artificial neural networks?
  - Multilayer neural networks?
  - ...

This lecture



Slide credit: Justin Johnson

This lecture



Slide credit: Justin Johnson

Definitions

"AI"	Any technique that enables **computers** to **mimic human** behavior
Machine Learning	Ability to learn **without** explicitly being **programmed**
"Deep" Learning	Extract patterns from data using **neural networks**
Computer Vision	Extracting meaning from **visual** signals
NLP	Extracting meaning from **textual** signals

Slide credit: Alexander Amini

Agenda

- **1. Recap: Bag of Visual Words, Analogy with NNs**

- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs

- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures

- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers

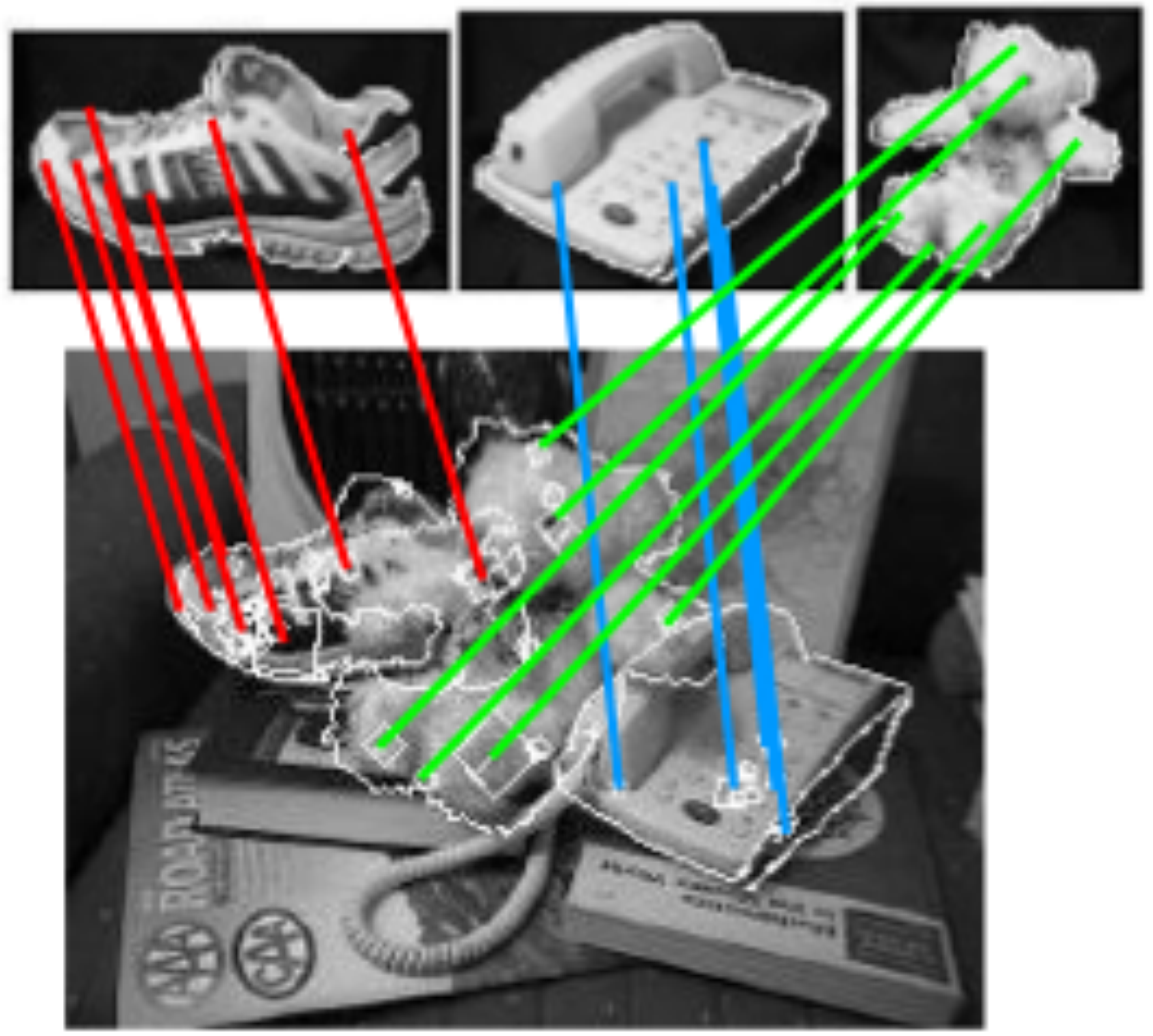
- **5. Beyond classification**

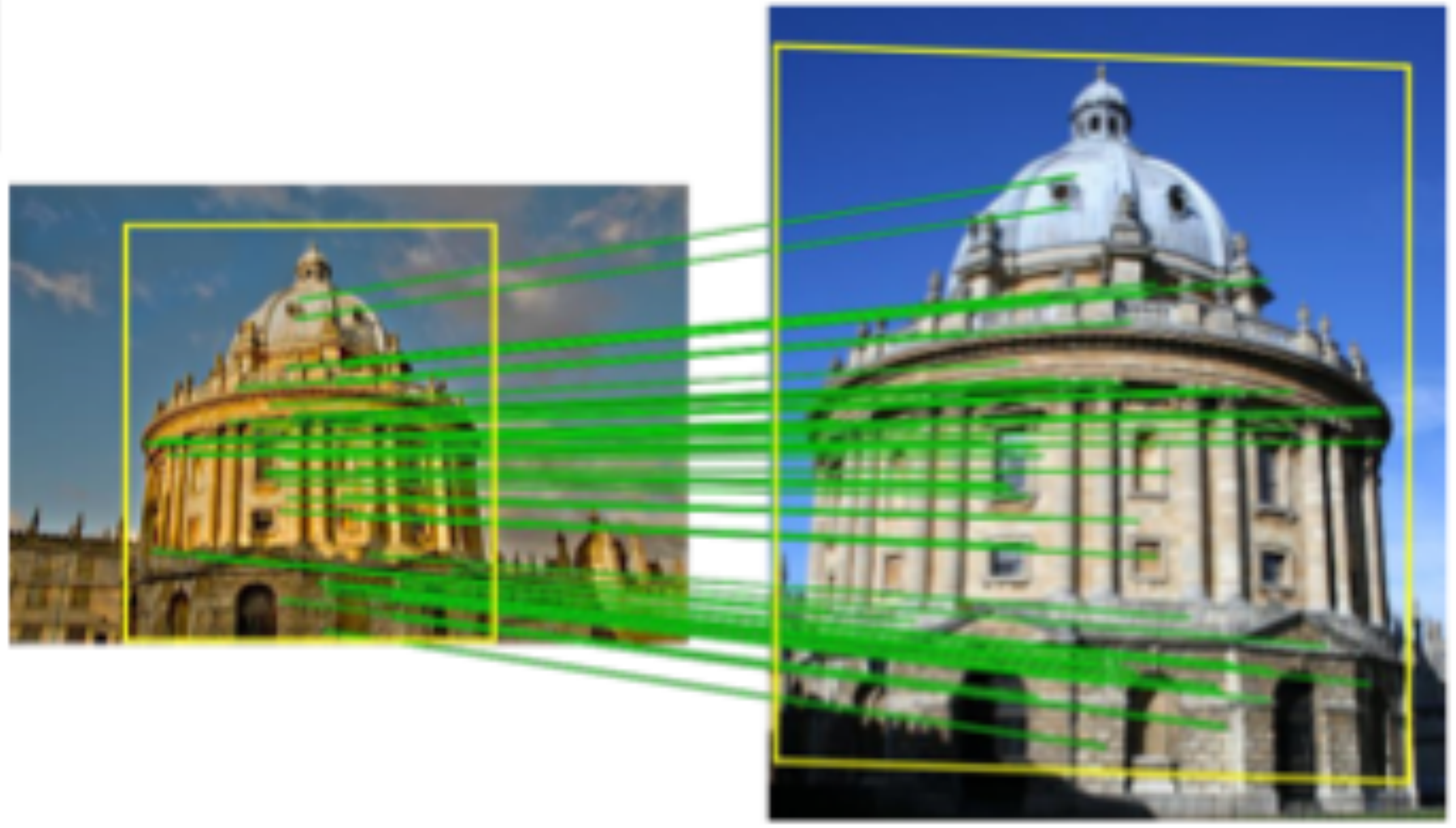
Agenda

- **1. Recap: Bag of Visual Words, Analogy with NNs**
- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs
- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures
- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers
- **5. Beyond classification**

Recap: Image recognition so far

Instance-level
recognition





Category Recognition

- Image classification: assigning a class label to the image

Car: present
Cow: present
Bike: not present
Horse: not present
...

Category Recognition

- Image classification: assigning a class label to the image

Car: present
Cow: present
Bike: not present
Horse: not present
…

- Object localization: define the location and the category

Location

Category

Slide: C. Schmid

Difficulties: within-class variations

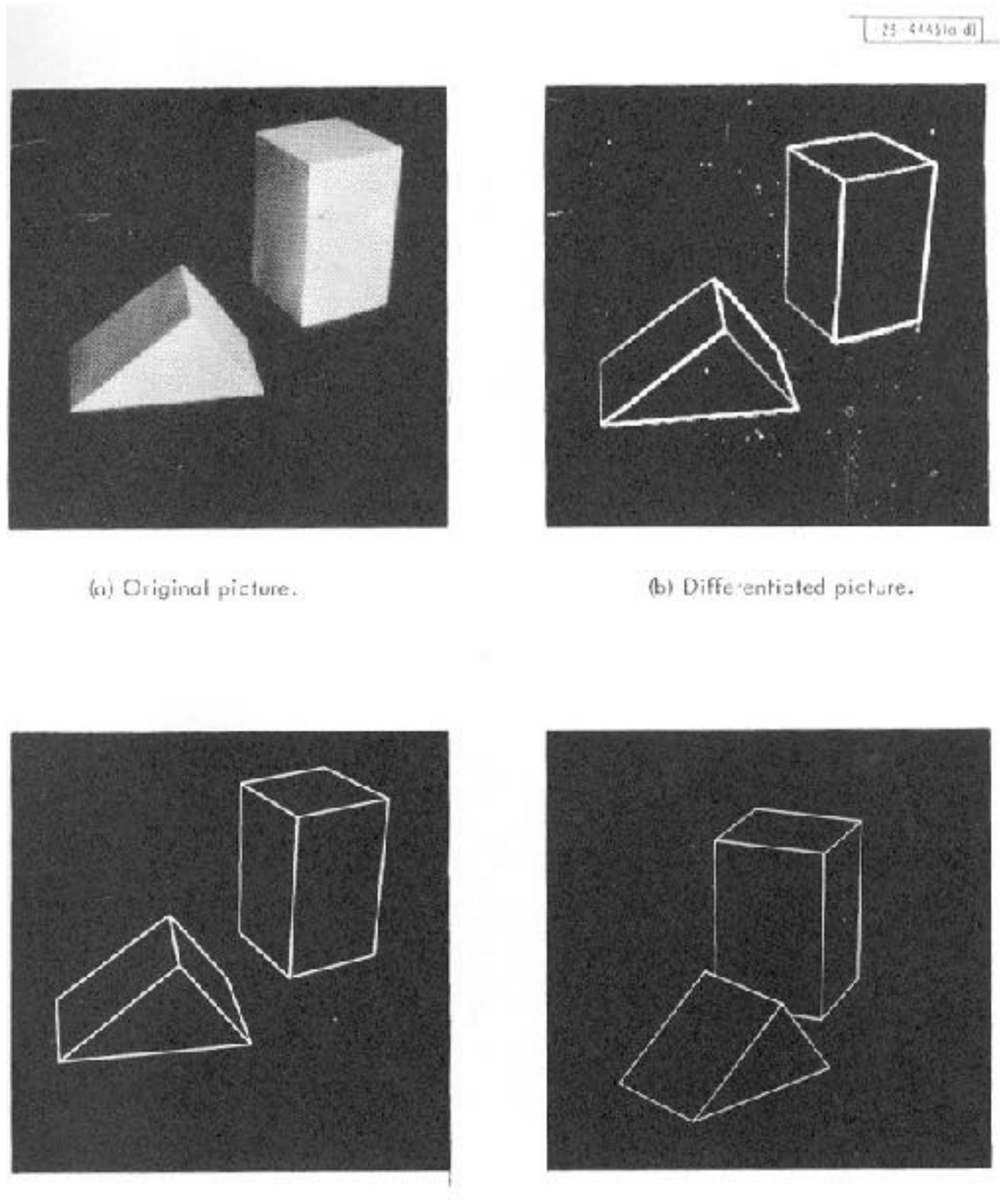





Slide: C. Schmid

Why machine learning?

- Early approaches: simple features + handcrafted models
- Can handle only few images, simple tasks



(a) Original picture. (b) Differentiated picture. (c) Line drawing. (d) Rotated view.

L. G. Roberts, *Machine Perception of Three Dimensional Solids,* Ph.D. thesis, MIT Department of Electrical Engineering, 1963.

ABSTRACT

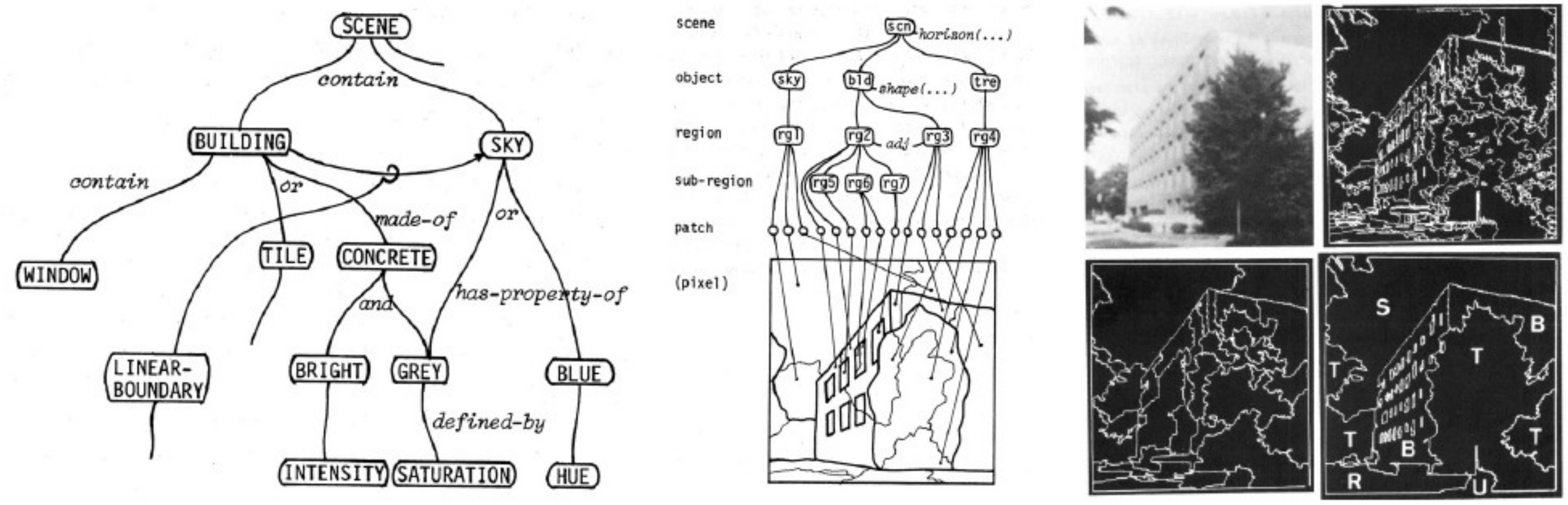
In order to make it possible for a computer to construct and display a three-dimensional array of solid objects from a single two-dimensional photograph, the rules and assumptions of depth perception have been carefully analyzed and mechanized. It is assumed that a photograph is a perspective projection of a set of objects which can be constructed from transformations of known three-dimensional models, and that the objects are supported by other visible objects or by a ground plane. These assumptions enable a computer to obtain a reasonable, three-dimensional description from the edge information in a photograph by means of a topological, mathematical process.

A computer program has been written which can process a photograph into a line drawing, transform the line drawing into a three-dimensional representation, and finally, display the three-dimensional structure with all the hidden lines removed, from any point of view. The 2-D to 3-D construction and 3-D to 2-D display processes are sufficiently general to handle most collections of planar-surfaced objects and provide a valuable starting point for future investigation of computer-aided three-dimensional systems.

Slide: C. Schmid

Why machine learning?

- Early approaches: manual programming of rules
- Tedious, limited, and does not take data into account



(a) Bottom-up process (b) Top-down process (c) Result

Figure 3. A system developed in 1978 by Ohta, Kanade and Sakai [33, 32] for knowledge-based interpretation of outdoor natural scenes. The system is able to label an image (c) into semantic classes: S-sky, T-tree, R-road, B-building, U-unknown.

Y. Ohta, T. Kanade, and T. Sakai, "An Analysis System for Scenes Containing objects with Substructures," *International Joint Conference on Pattern Recognition*, 1978.

Slide: C. Schmid

Why machine learning?

- Today lots of data, complex tasks

Internet images,
personal photo albums

Movies, news, sports

- Instead of trying to encode rules directly, learn them from examples of inputs and desired outputs

Slide: C. Schmid

Texture Classification

- Profound observation: Grass and sea pictures don't look the same!
- Basic idea: Model the distribution of "texture" over the image (or over a region) and classify in different classes based on the texture models learned from training examples.

Grass

Sea

Image categorization

- Profound observation: Cows and buildings don't look the same!
- Basic idea: Model the distribution of "texture" over the image (or over a region) and classify in different classes based on the texture models learned from training examples.

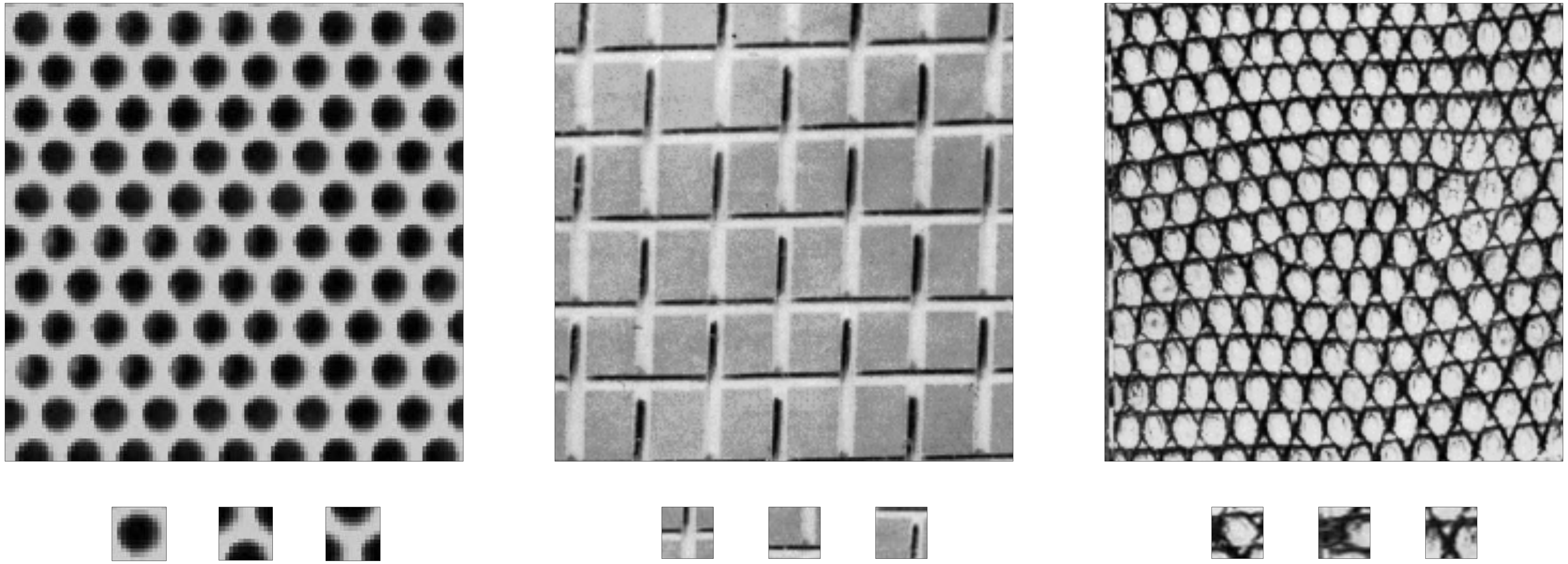
Cow

Building

Bag-of-features for image classification

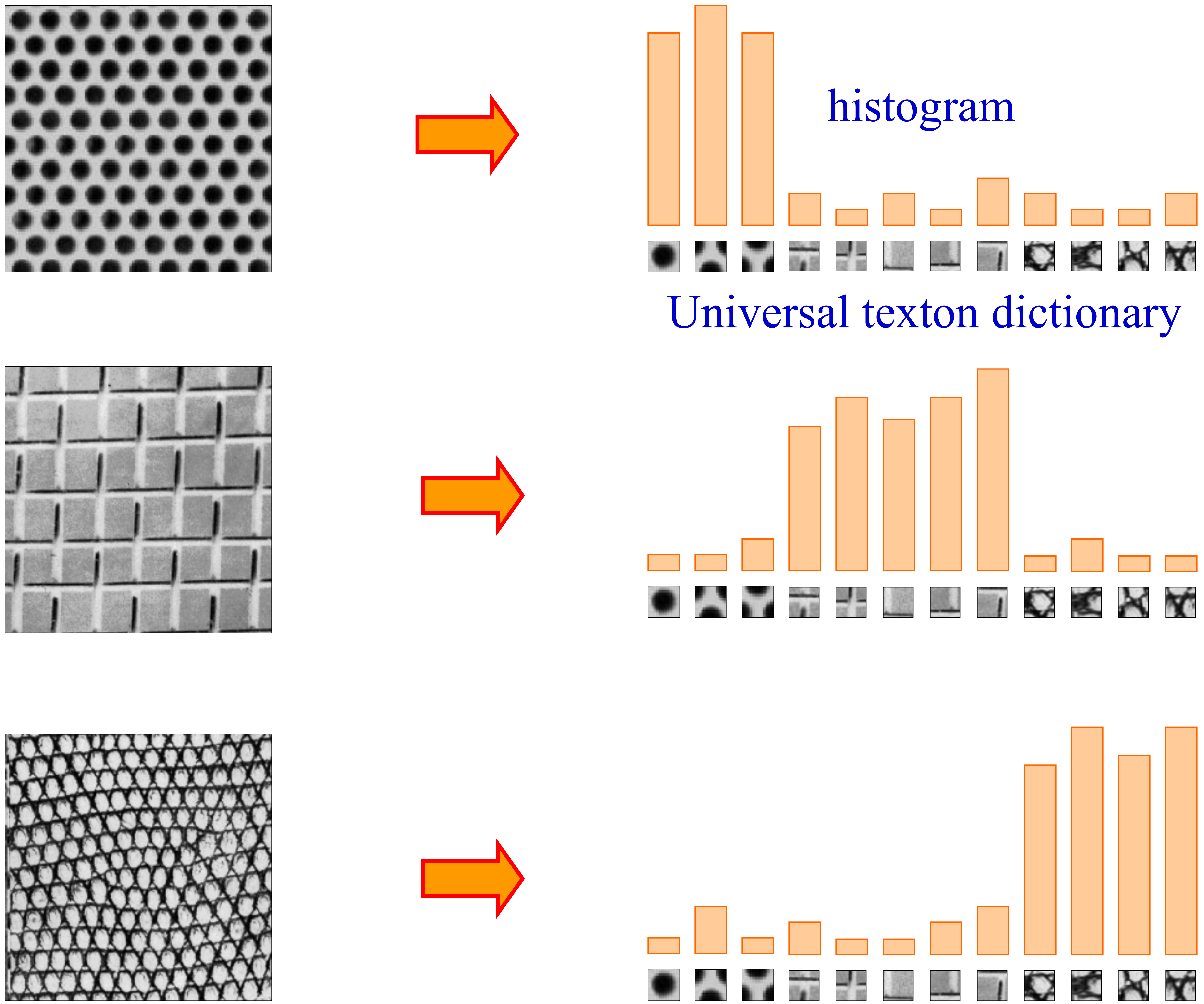
Origin: texture recognition

- Texture is characterized by the repetition of basic elements or *textons*



Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

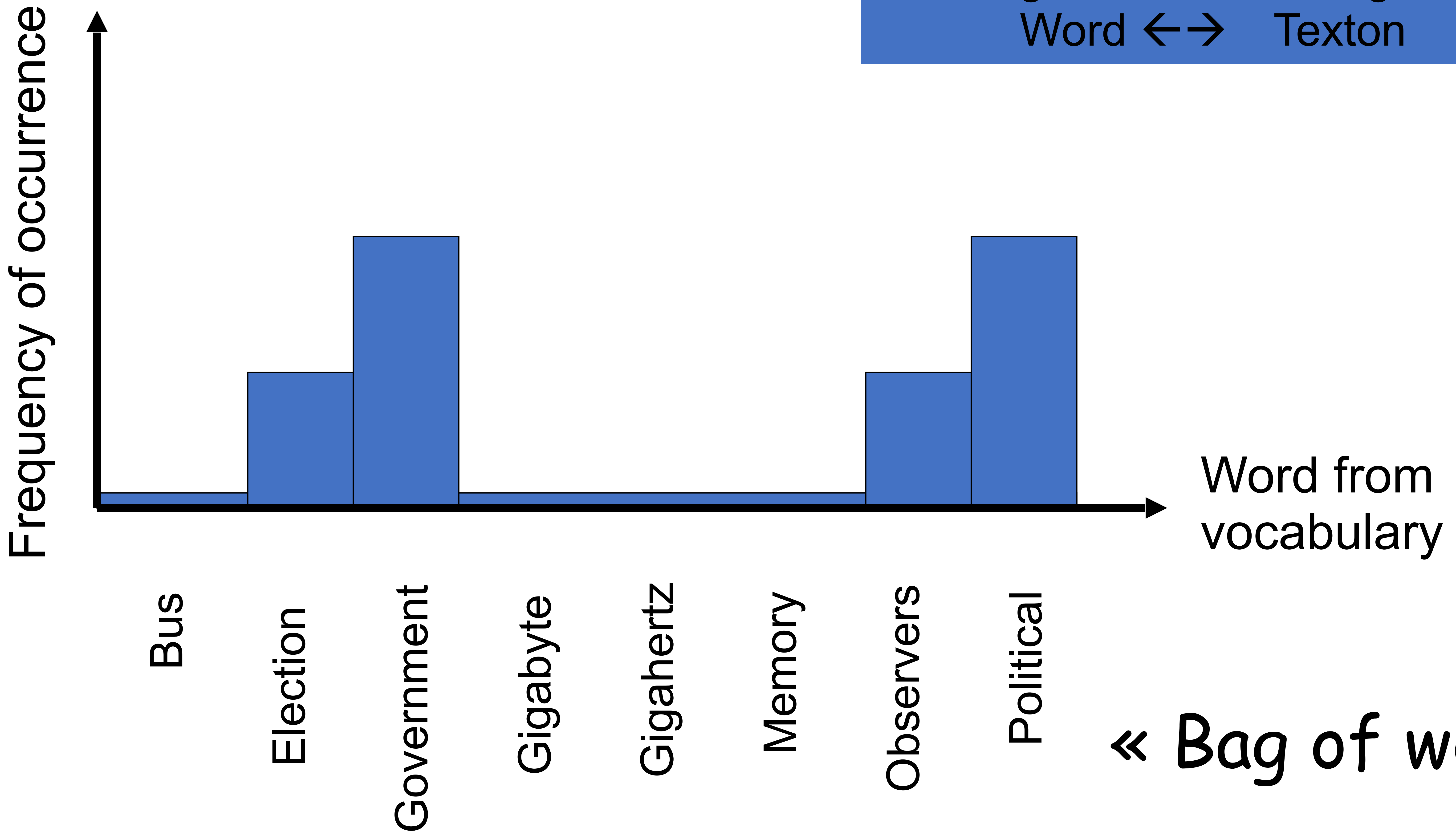
Bag-of-features for image classification



Analogy with Text Analysis

Political observers say that the government of Zorgia does not control the political situation. The government will not hold elections …

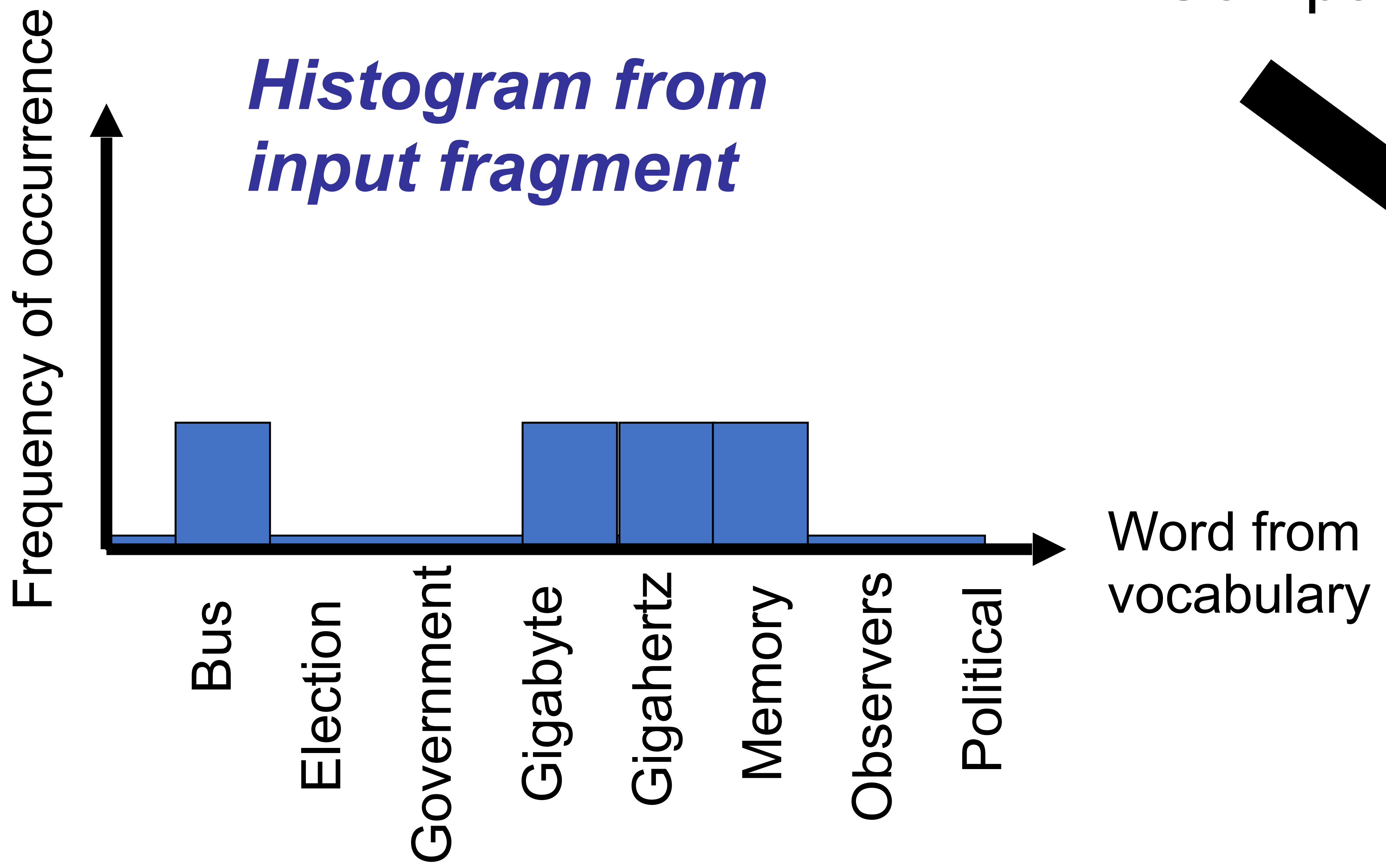
Analogy:
Text fragment ←→ Image region
Word ←→ Texton



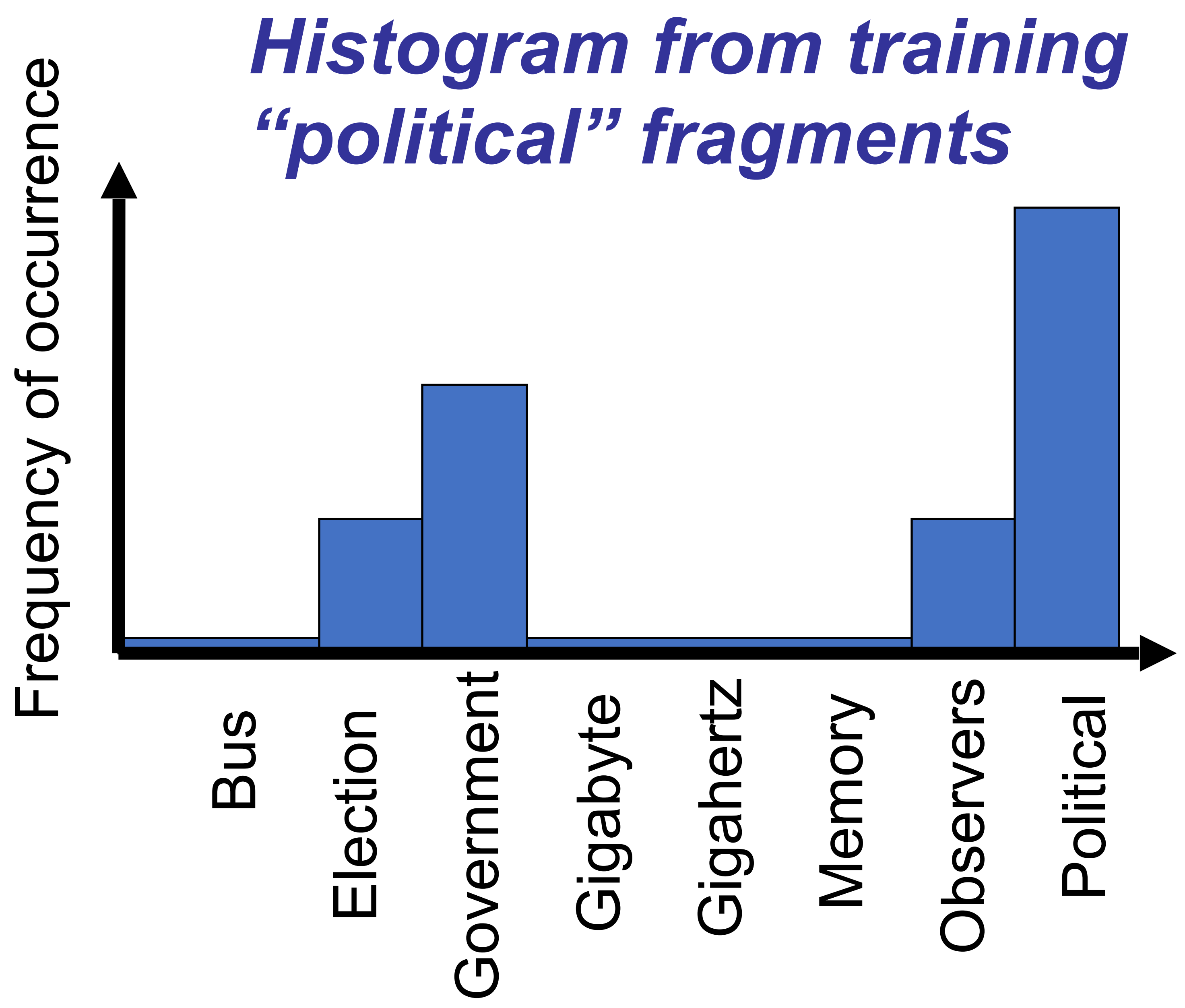
« Bag of words »

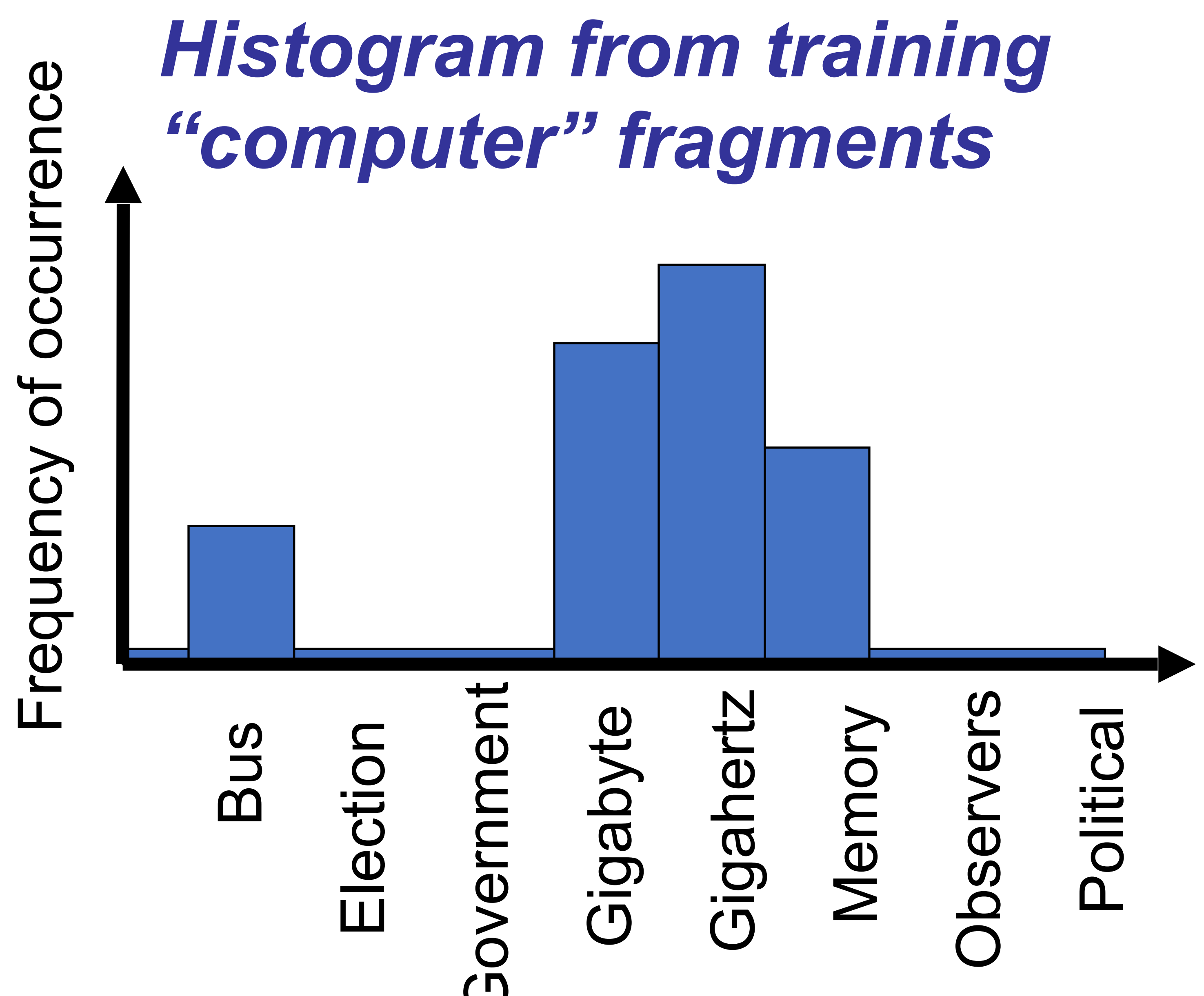
Analogy with Text Analysis

The ZH-20 unit is a 200Gigahertz processor with 2Gigabyte memory. Its strength is its bus and high-speed memory……

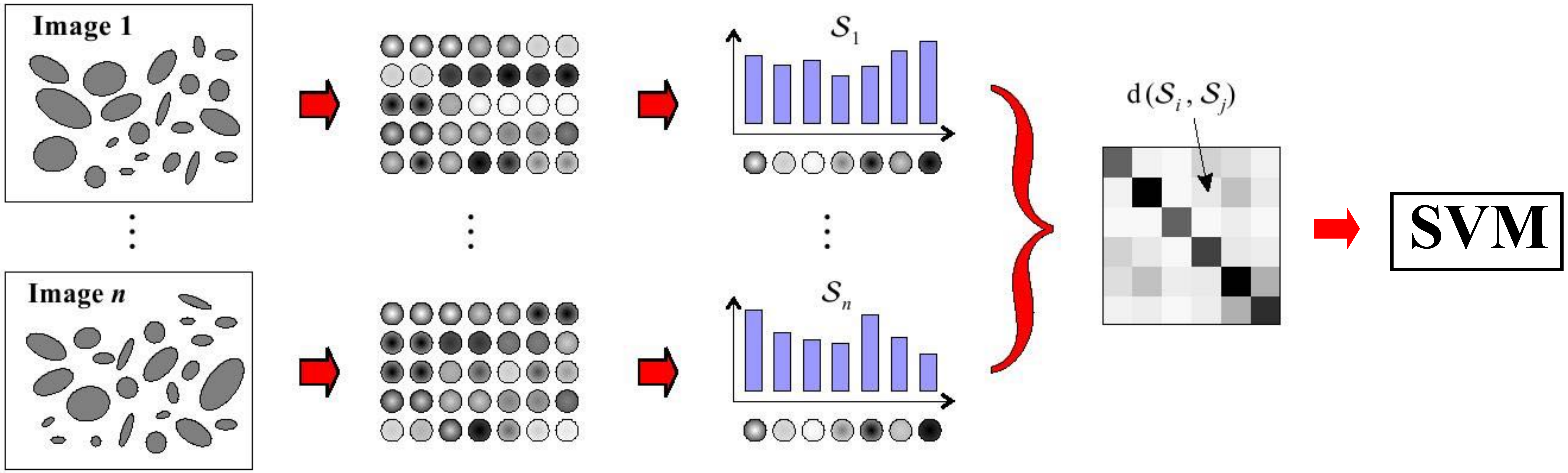


Compare





Bag-of-features for image classification



Extract regions

Compute descriptors

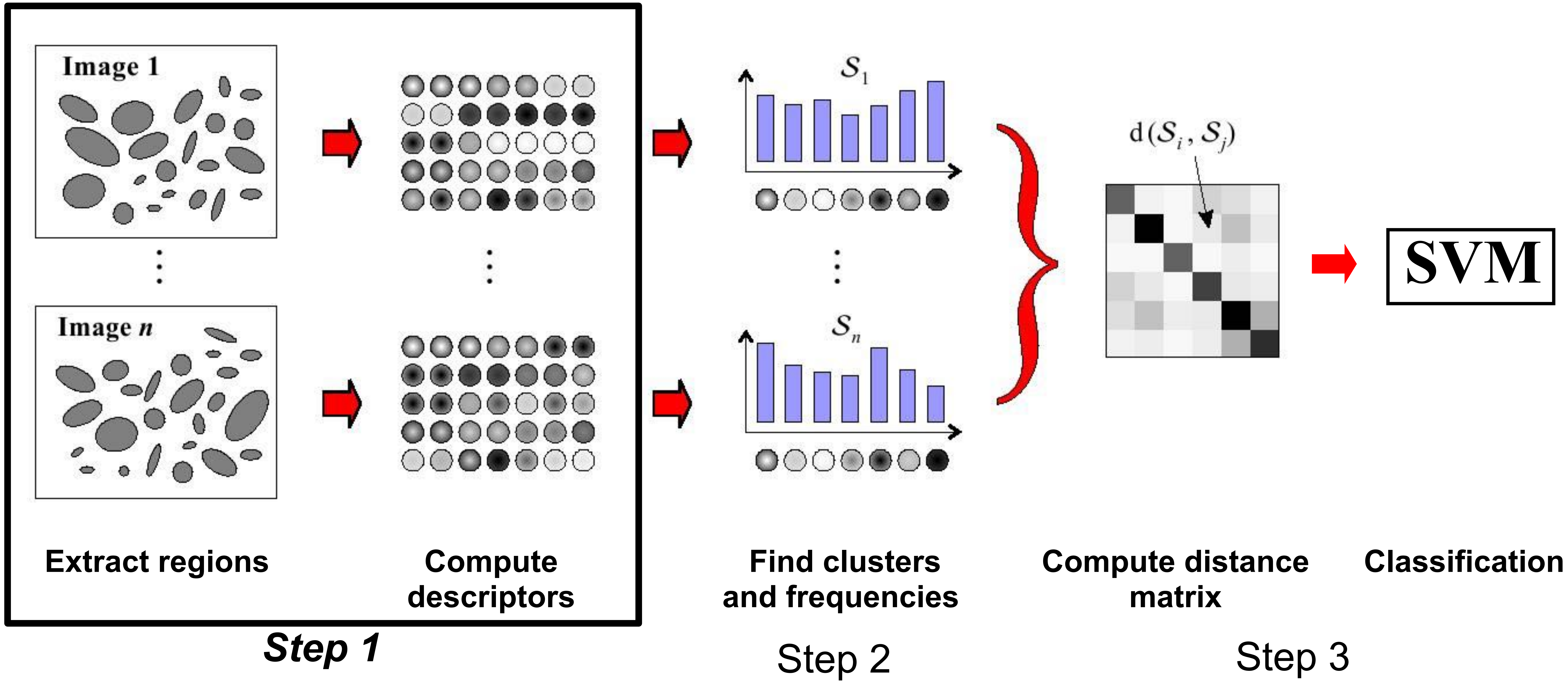
Find clusters and frequencies

Compute distance matrix

Classification

[Csurka et al. WS'2004], [Nowak et al. ECCV'06], [Zhang et al. IJCV'07]

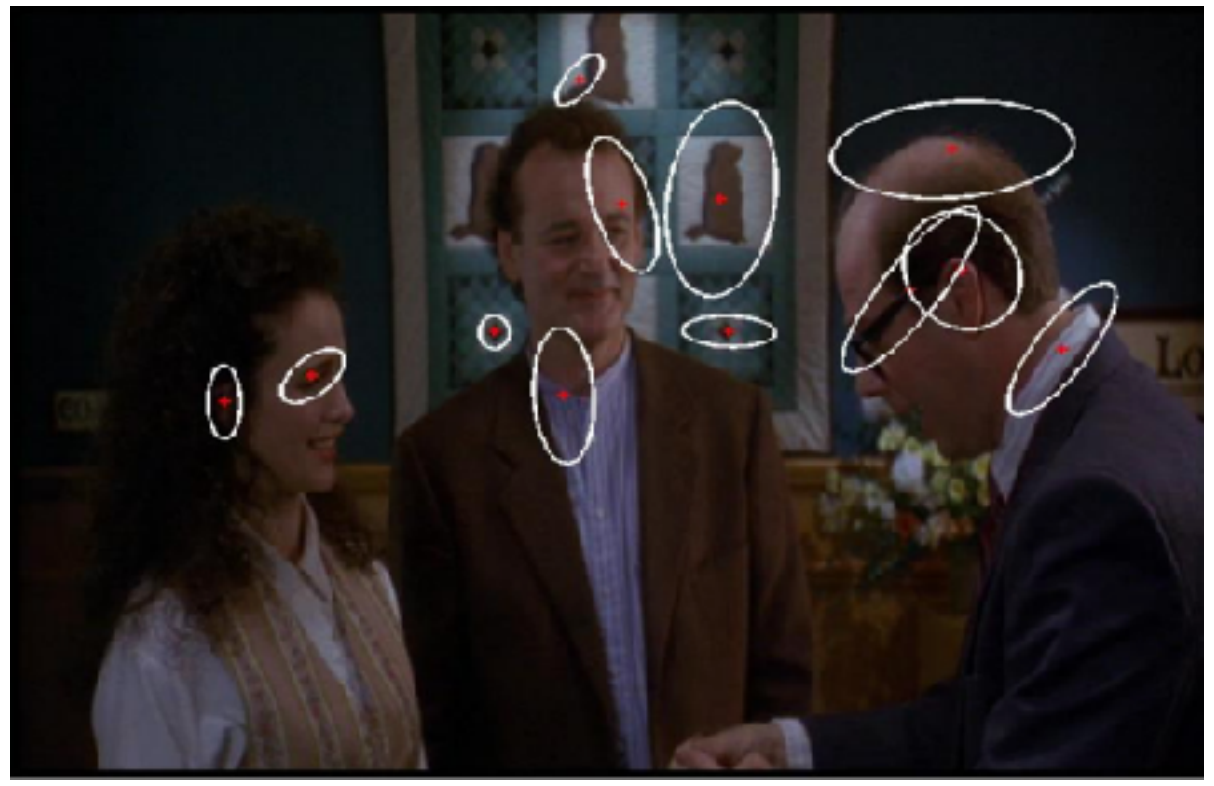
Bag-of-features for image classification



Step 1: feature extraction

Sparse sampling

- SIFT as interest point detector



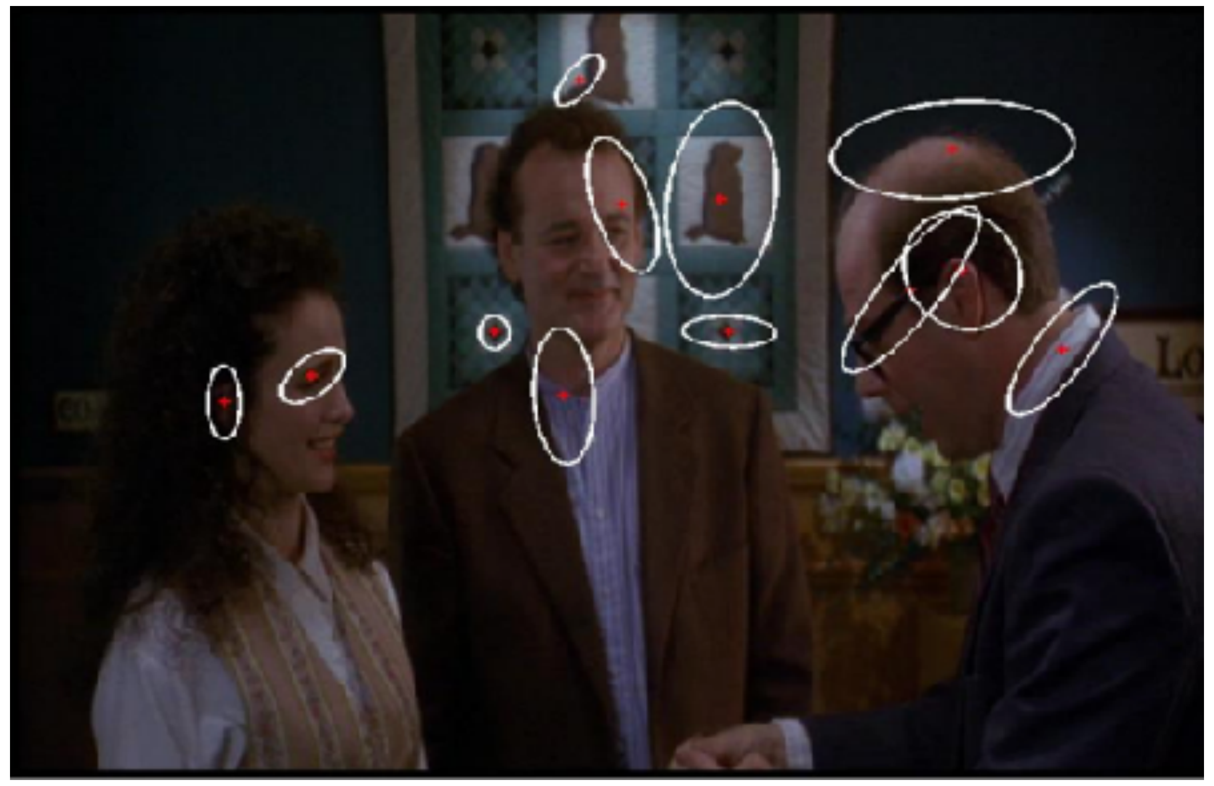
Dense sampling

- Interest points do not necessarily capture “all” features

Step 1: feature extraction

Sparse sampling

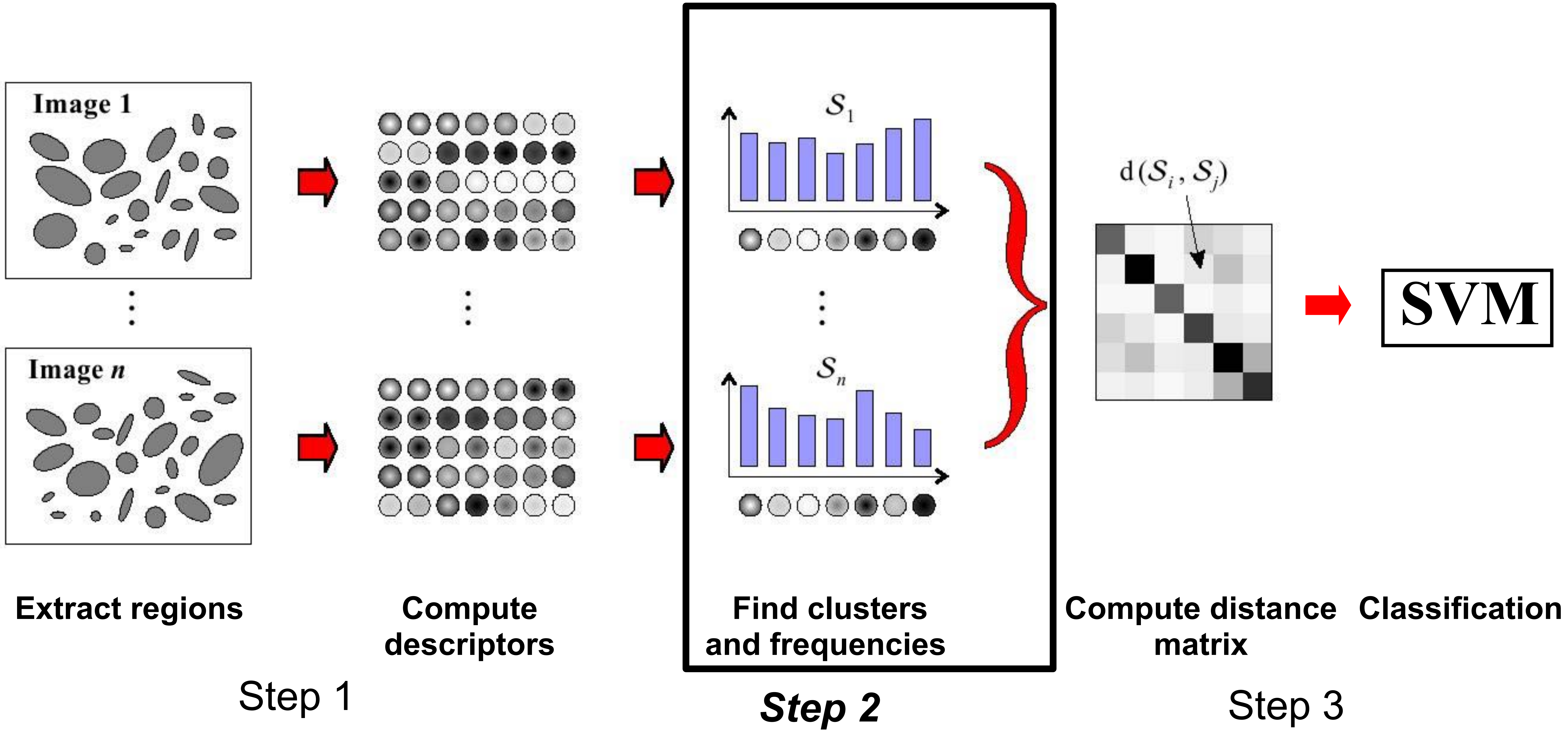
- SIFT as interest point detector



Dense sampling

- Interest points do not necessarily capture "all" features
- Spatial pyramid (Lazebnik, Schmid & Ponce, CVPR 2006)

Bag-of-features for image classification



Step 2: Quantization

Cluster descriptors

- K-means
- Gaussian mixture model

Assign each visual word to a cluster

- Hard or soft assignment

Build frequency histogram

Examples for visual words

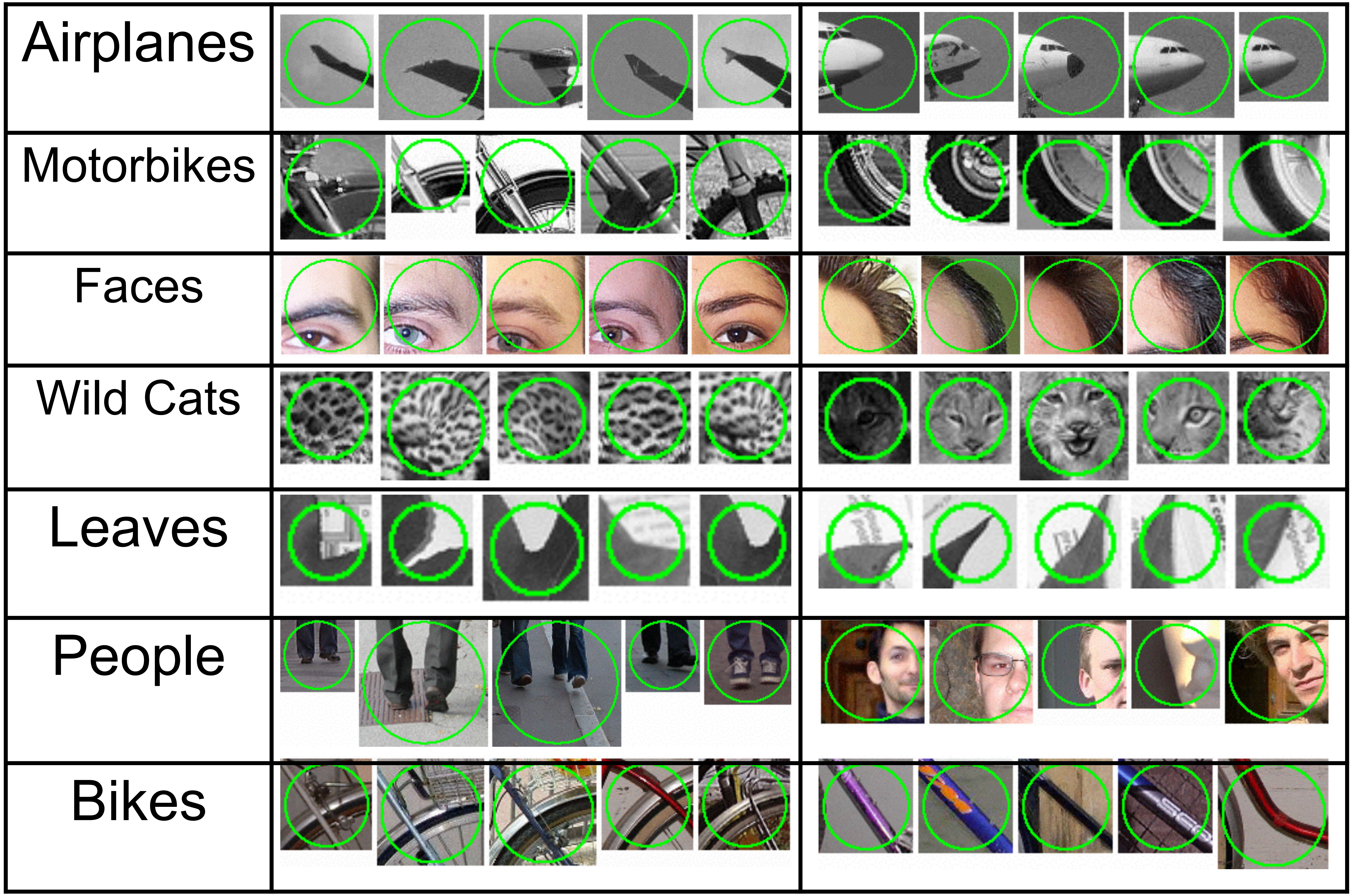
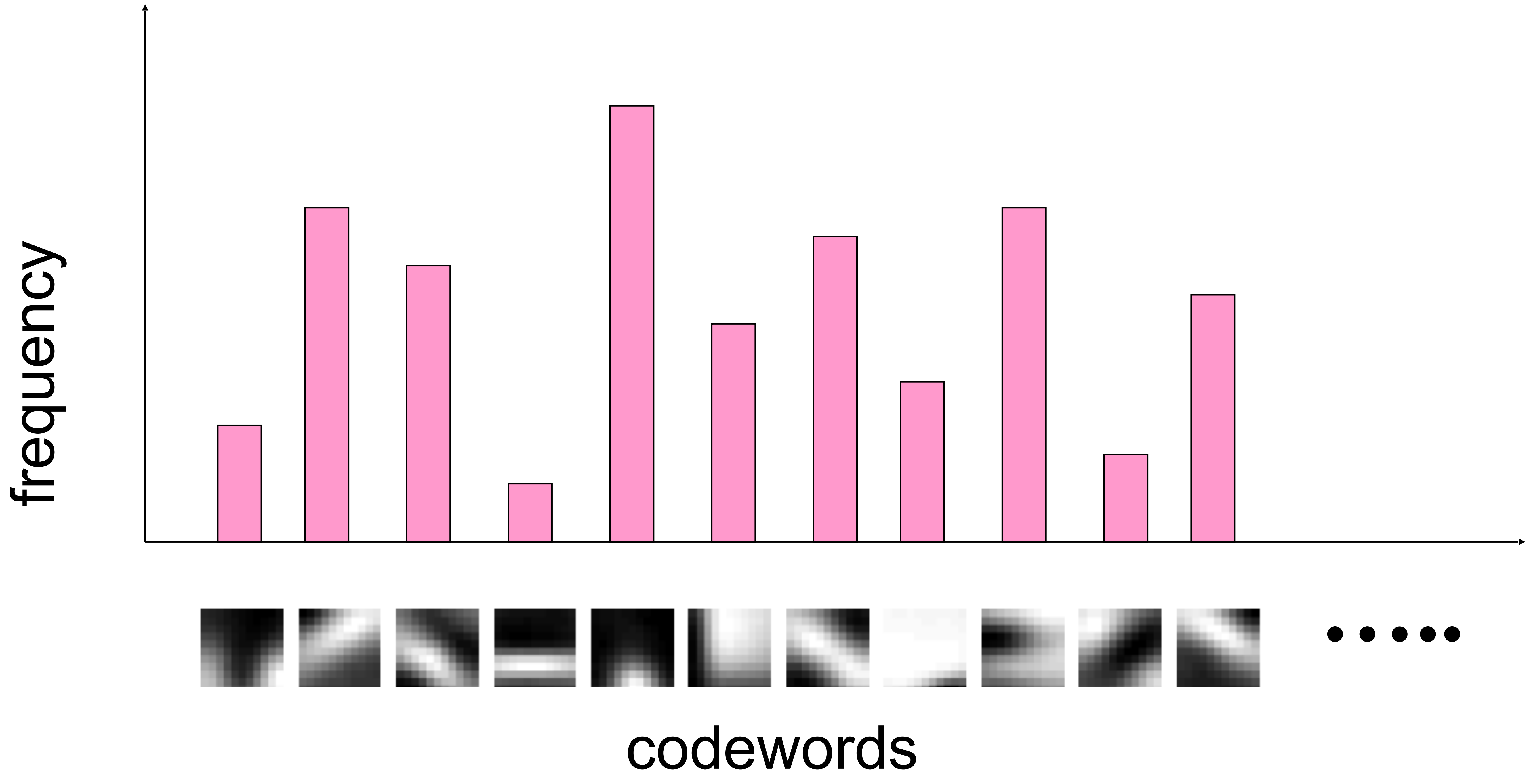
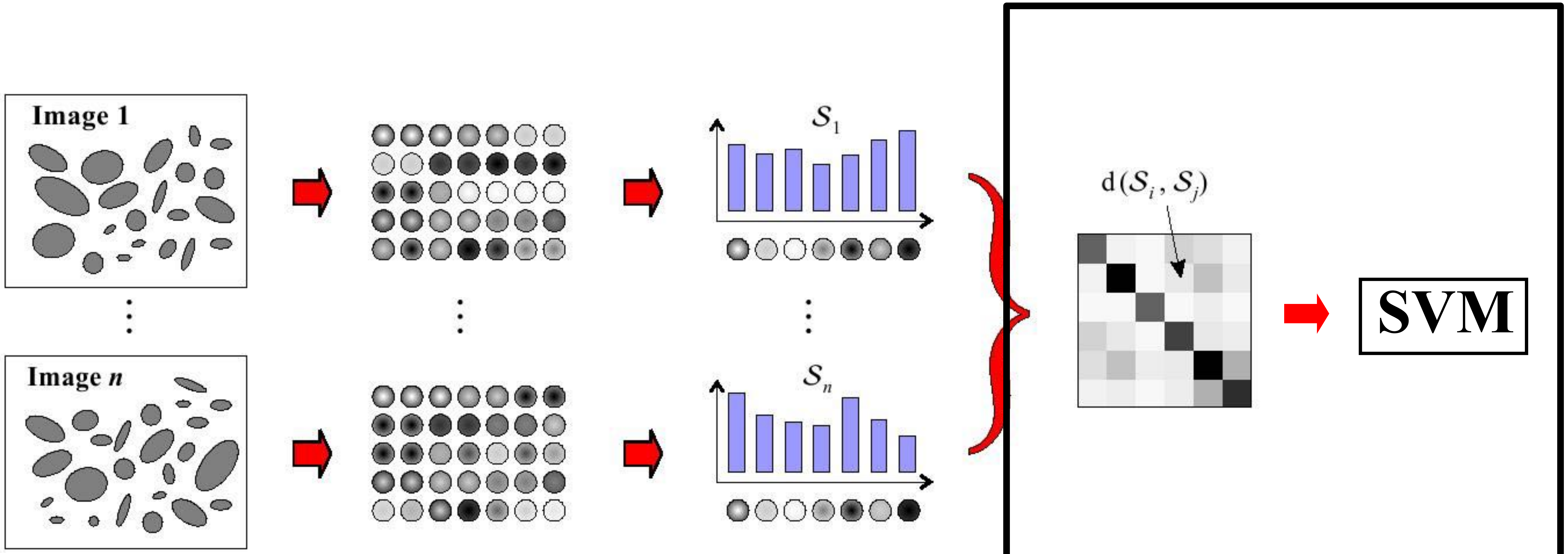


Image representation



- Each image is represented by an aggregated histogram vector, typically 1000-4000 dimensional
- Normalized with L2 norm
- Fisher Vectors [Perronnin et al. ECCV’10]: improvements over Bag of Features

Bag-of-features for image classification

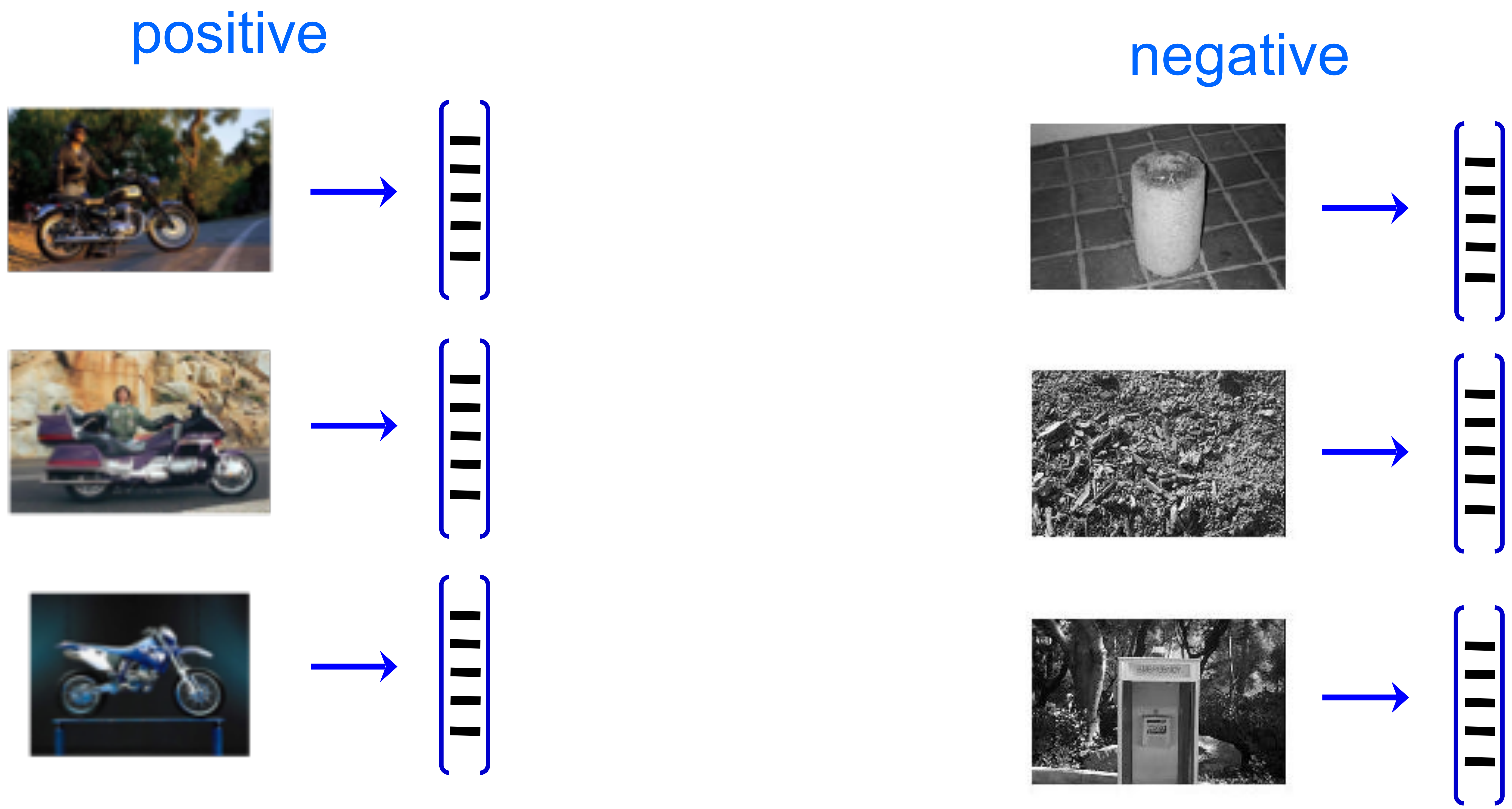


Extract regions | Compute descriptors | Find clusters and frequencies | Compute distance matrix | Classification

Step 1 | Step 2 | ***Step 3***

Step 3: Classification

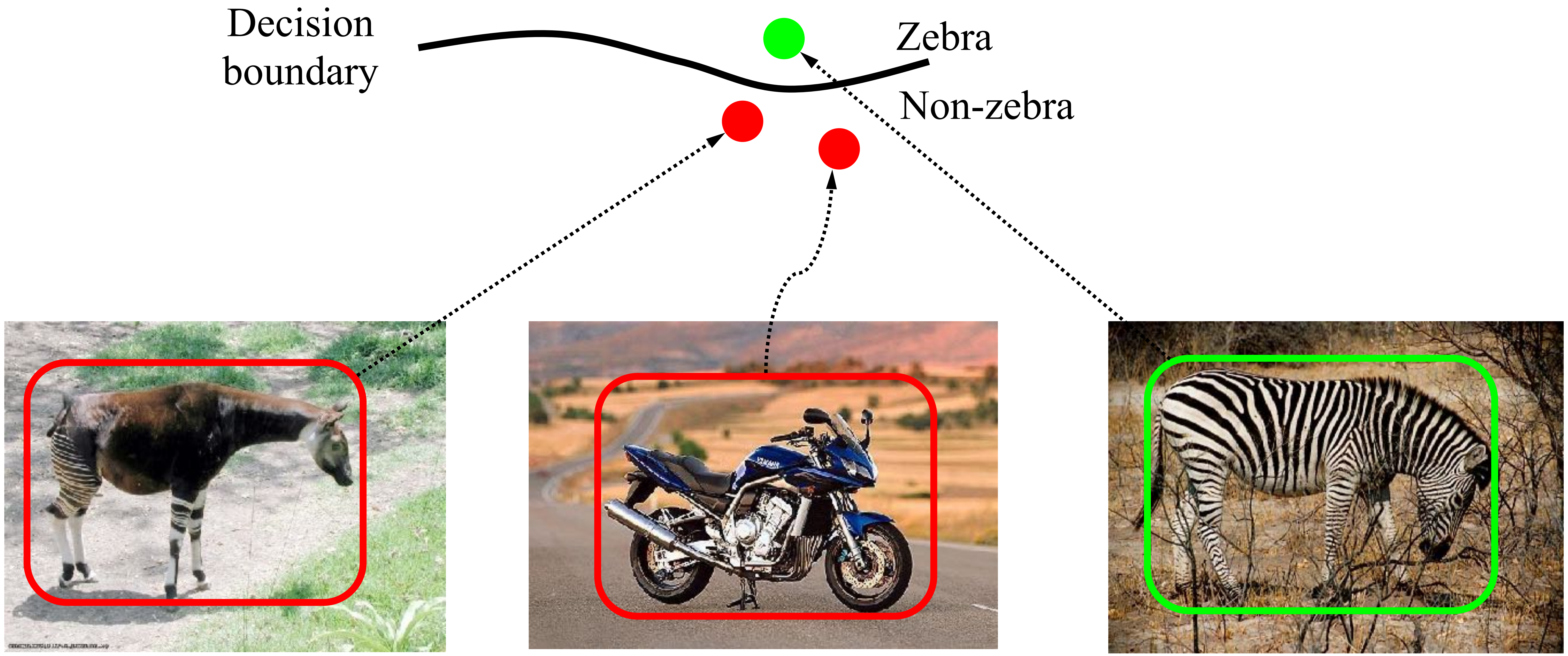
Training data: Vectors are histograms, one from each image



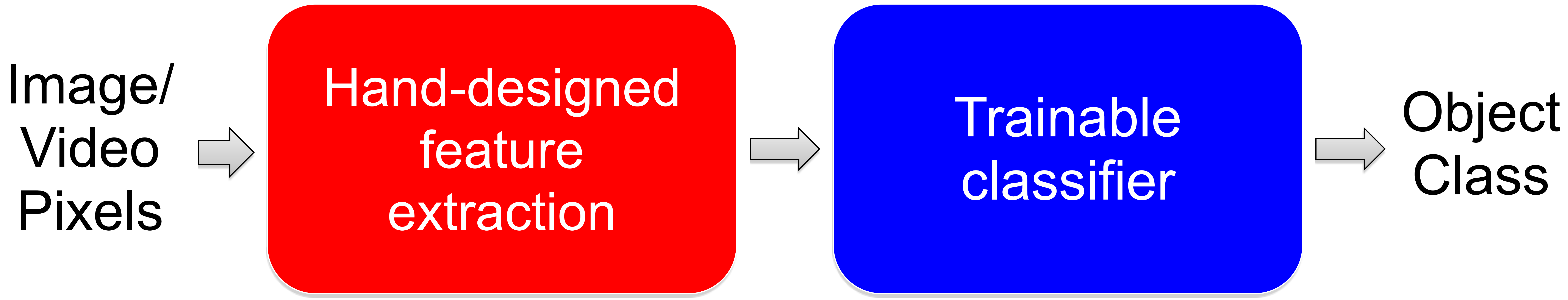
Train classifier,e.g. SVM

Step 3: Classification

Learn a decision rule (classifier) assigning bag-of-features representations of images to different classes

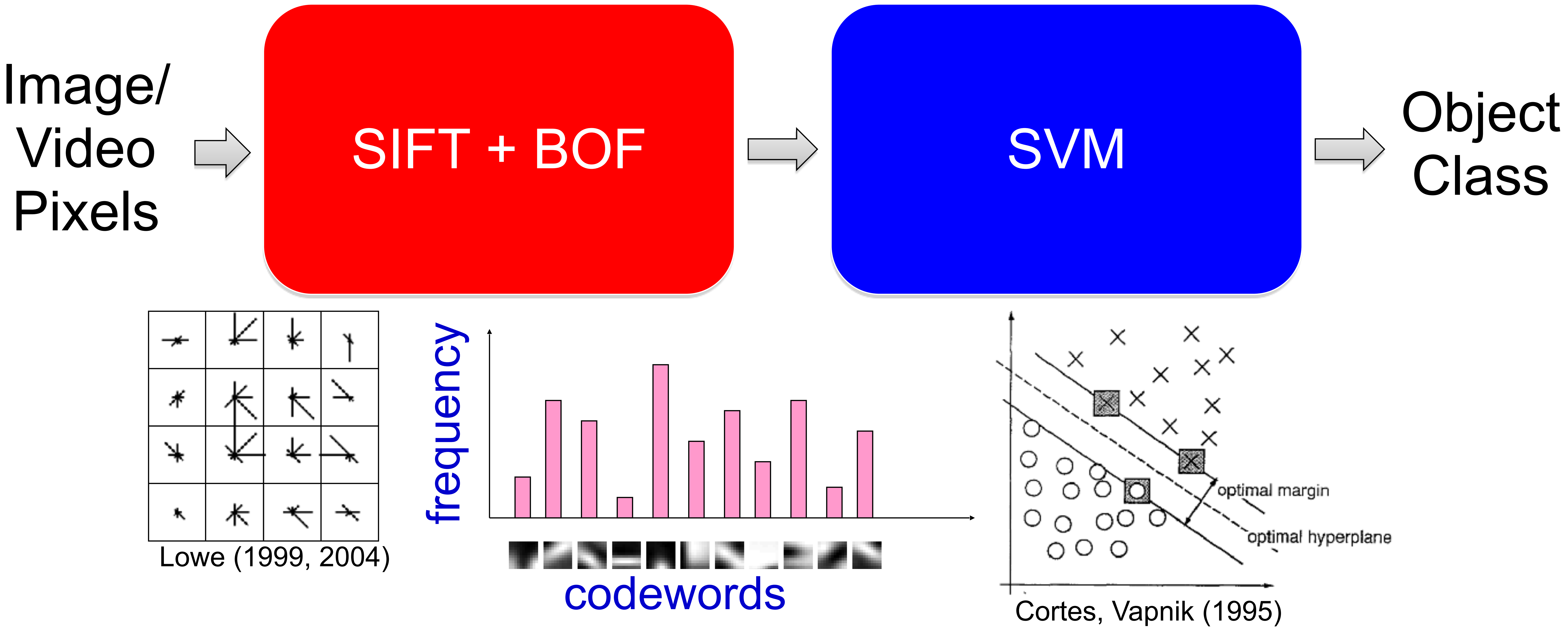


Traditional Recognition Approach



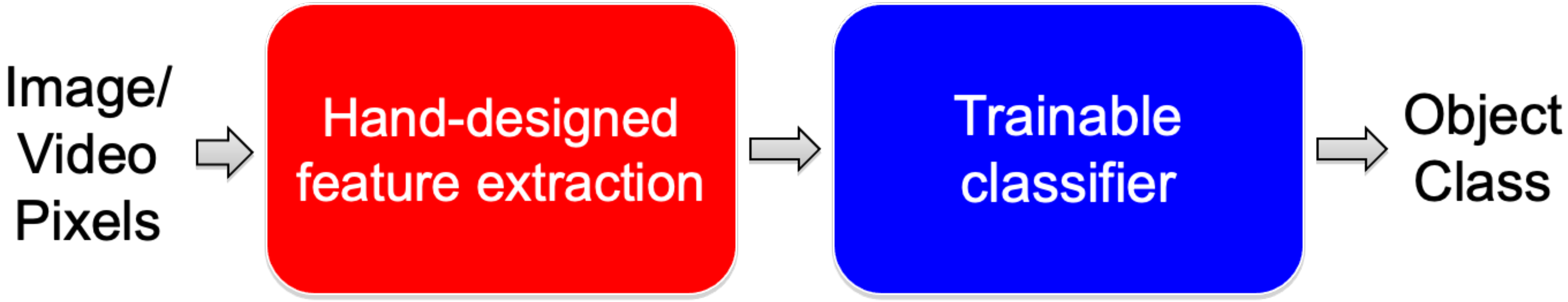
Slide: R. Fergus / S. Lazebnik

Traditional Recognition Example



- SIFT features
- BOF: Bag of Features / Visual Words (inspired by Bag of Words in NLP)
- SVM: Support Vector Machines for classification

Analogy to the traditional visual recognition pipeline



- Features are not learned (e.g., HOG, SIFT, Bag of Features)
- Trainable classifier is often generic (e.g., SVM, Random Forest)

Slide: R. Fergus / S. Lazebnik

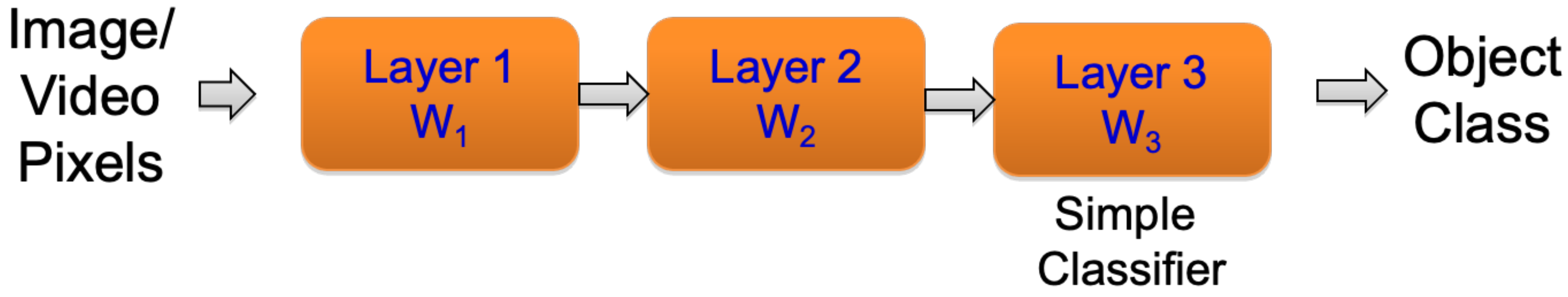
Analogy to the traditional visual recognition pipeline

What about learning the features?

- Features are learned "end-to-end" (i.e., pixels are input)
- "Feature hierarchy" all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly

Slide: R. Fergus / S. Lazebnik

Analogy to the traditional visual recognition pipeline

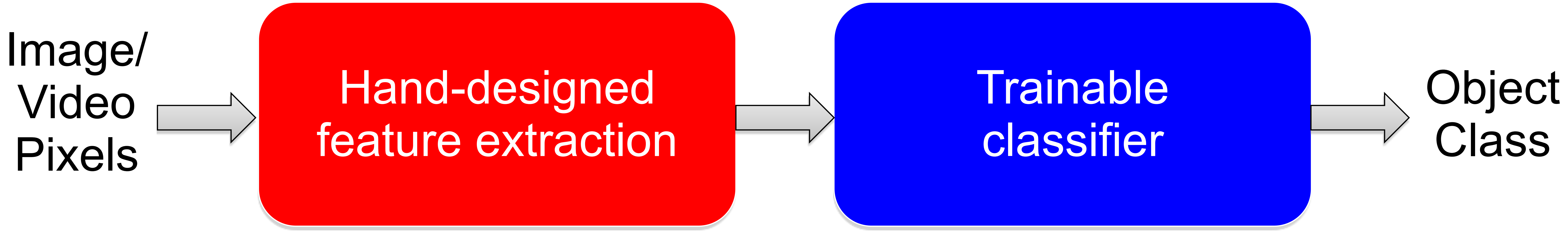


- Features are learned "end-to-end" (i.e., pixels are input)
- "Feature hierarchy" all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly

Slide: R. Fergus / S. Lazebnik

“Shallow” vs. “deep” models

Traditional recognition: “Shallow” architecture



Deep learning: “Deep” architecture

Slide: R. Fergus / S. Lazebnik

Agenda

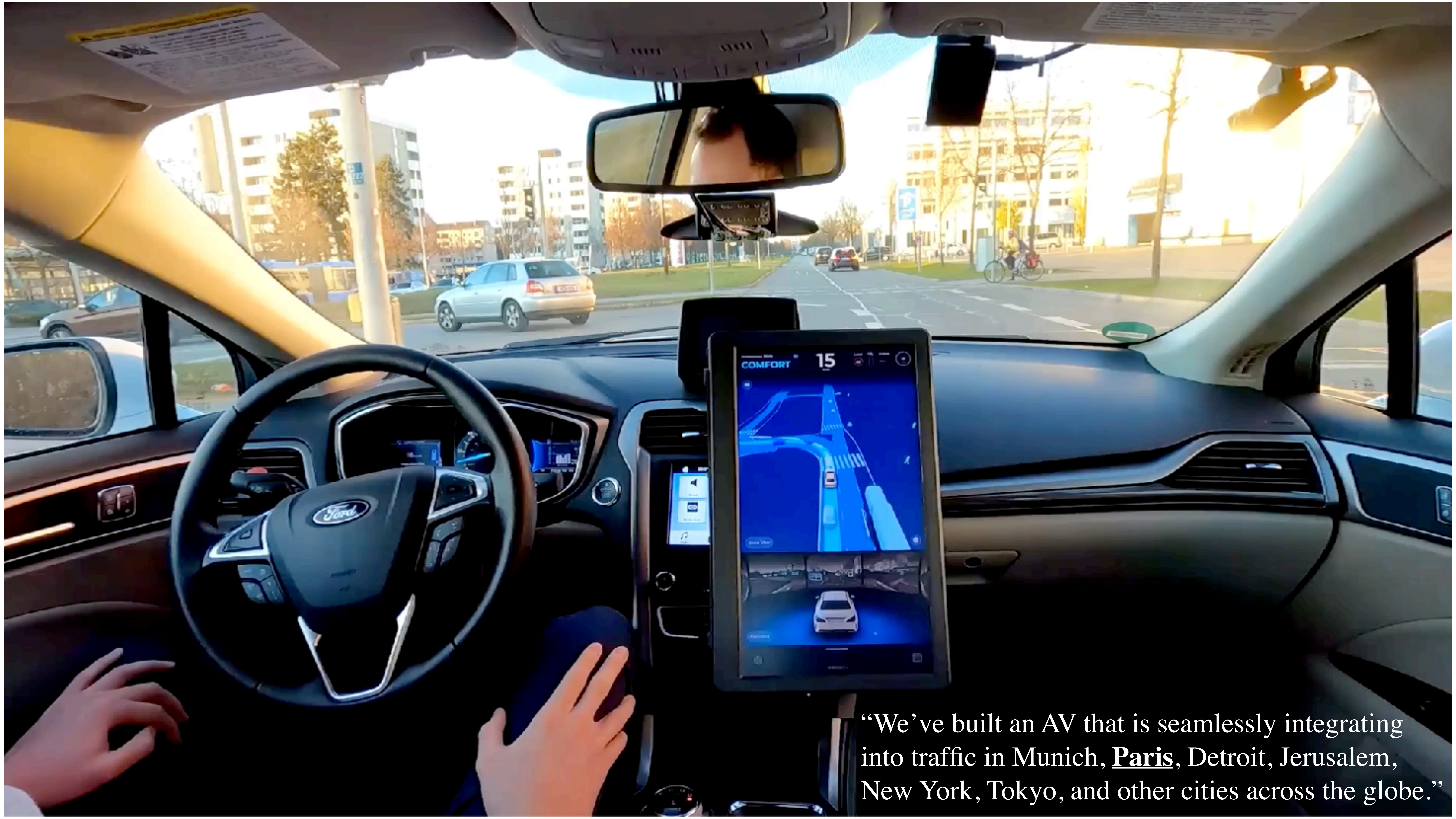
- **1. Recap: Bag of Visual Words, Analogy with NNs**
- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs
- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures
- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers
- **5. Beyond classification**

Neural Networks in Production

Face detection

Slide credit: Kosta Derpanis

Self-driving cars / Autonomous vehicles

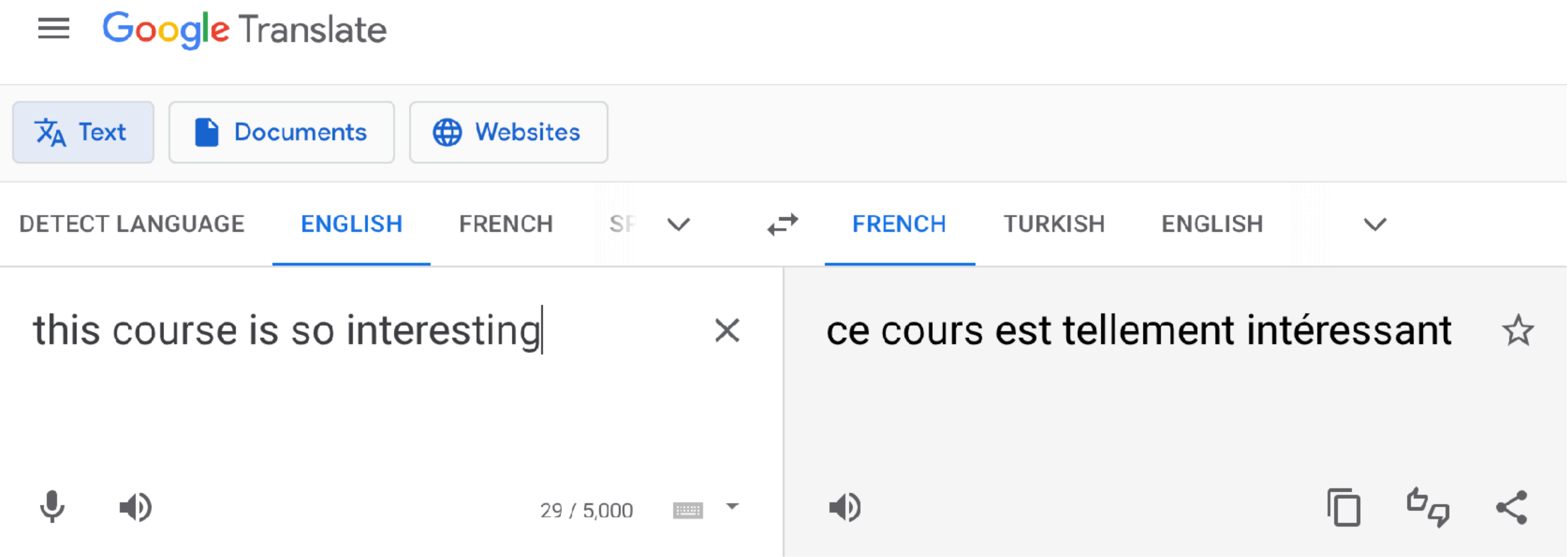


mobileye™

Shopping

Slide credit: Kosta Derpanis

Google Translate



What is “Deep” Learning?

Recap: Basics of supervised learning

- n training data pairs $(x_1, y_1), ..., (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$
- Learn a predictor/decision function $\hat{f} : \mathcal{X} \to \mathcal{A}$
- By minimizing $$\sum_{i=1}^{n} l(f(x_i), y_i)$$

Recap: Basics of supervised learning

- n training data pairs

$$(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$$

- Learn a predictor/decision function

$$\hat{f} : \mathcal{X} \to \mathcal{A}$$

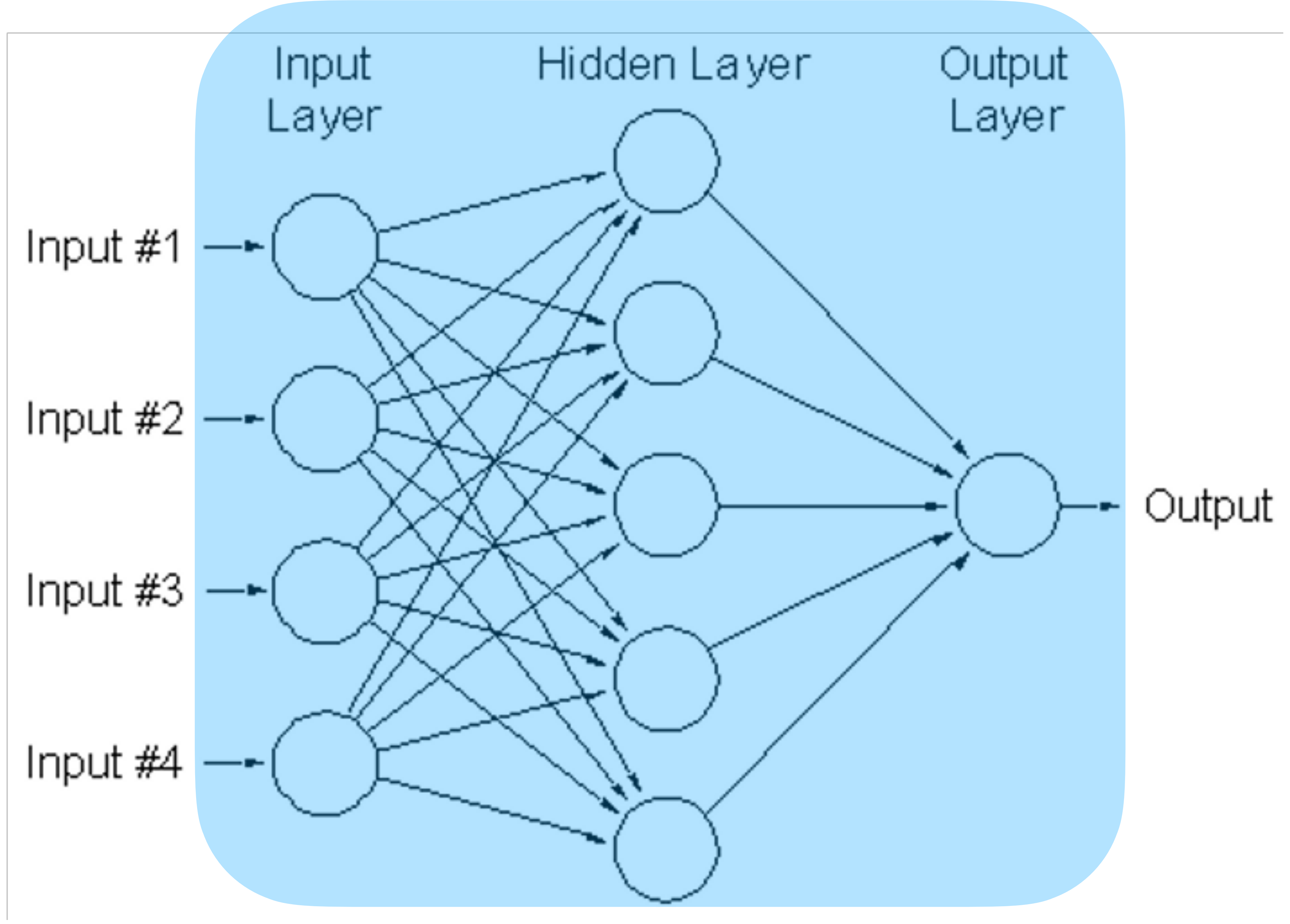
- By minimizing

$$\sum_{i=1}^{n} \underbrace{l}_{\text{Loss}}(\underbrace{f}_{\text{Model}}(\underbrace{x_i}_{\text{Input}}), \underbrace{y_i}_{\text{Label}})$$

Deep learning

$$\sum_{i=1}^{n} l(f(x_i), y_i)$$

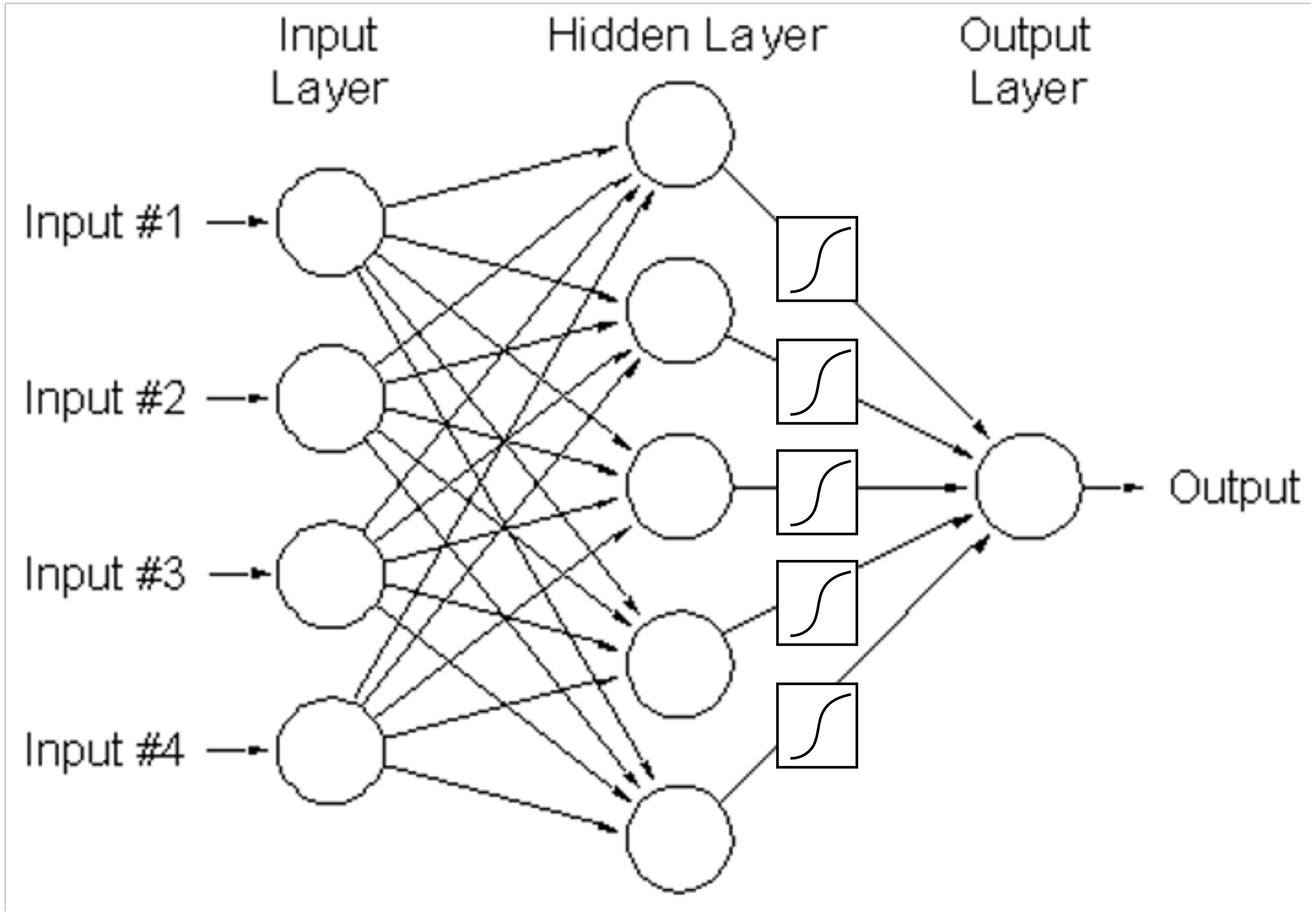
Loss Model Input Label



Deep learning:
Model = neural network

What is a “deep” neural network?

Stacking more than one **layer**



W_1
(Layer 1)

W_2
(Layer 2)

What is a layer?

Typically matrix multiplication! (But the function can take many forms*)

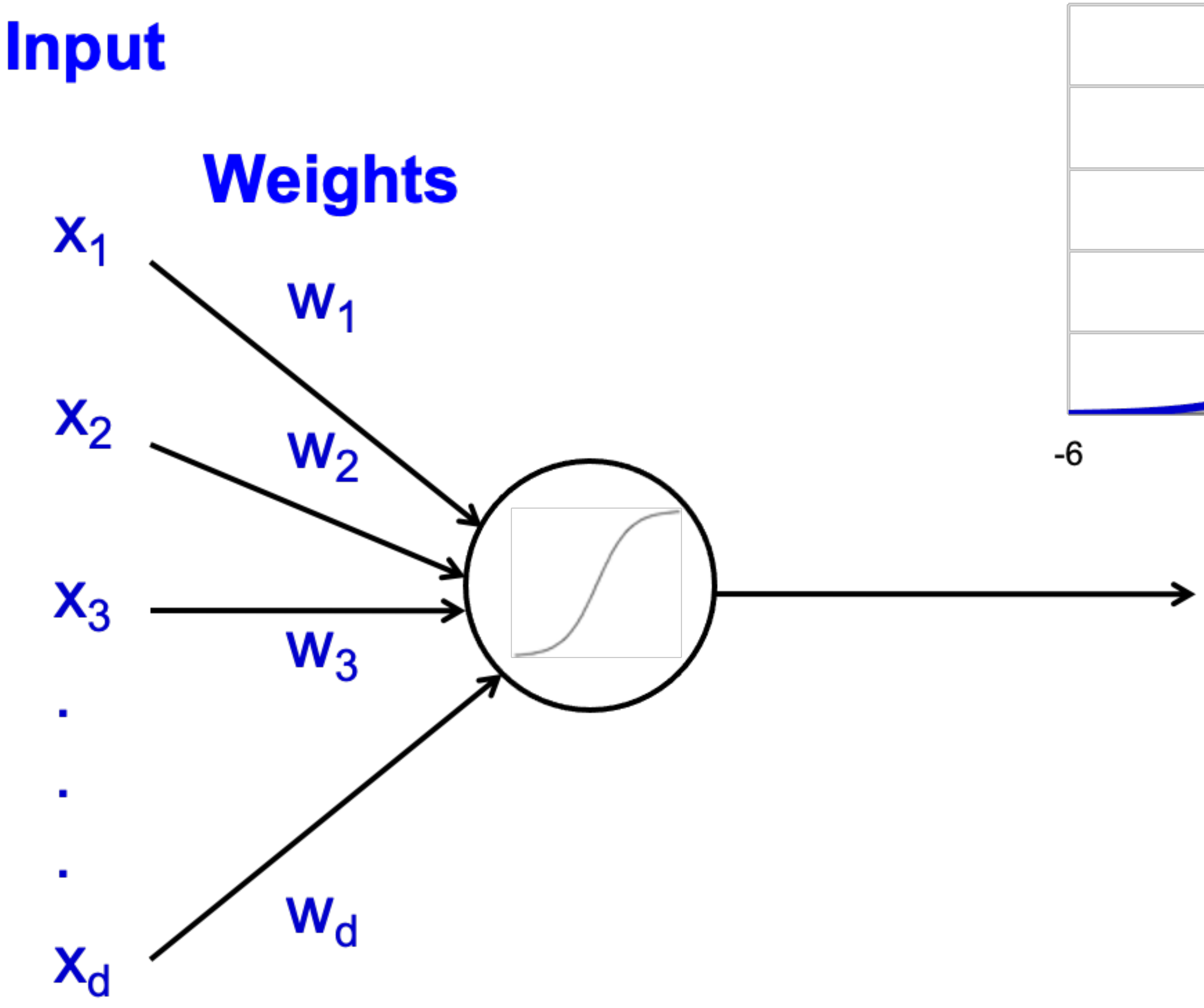
- **Fully-connected** layer
- **Convolution** layer
- **Pooling** layer (e.g., Max-pooling)
- **Non-linearity** layer (e.g., ReLU)
- **Attention** layer
- …

*requirement to be differentiable if optimized with gradient descent algorithm variants

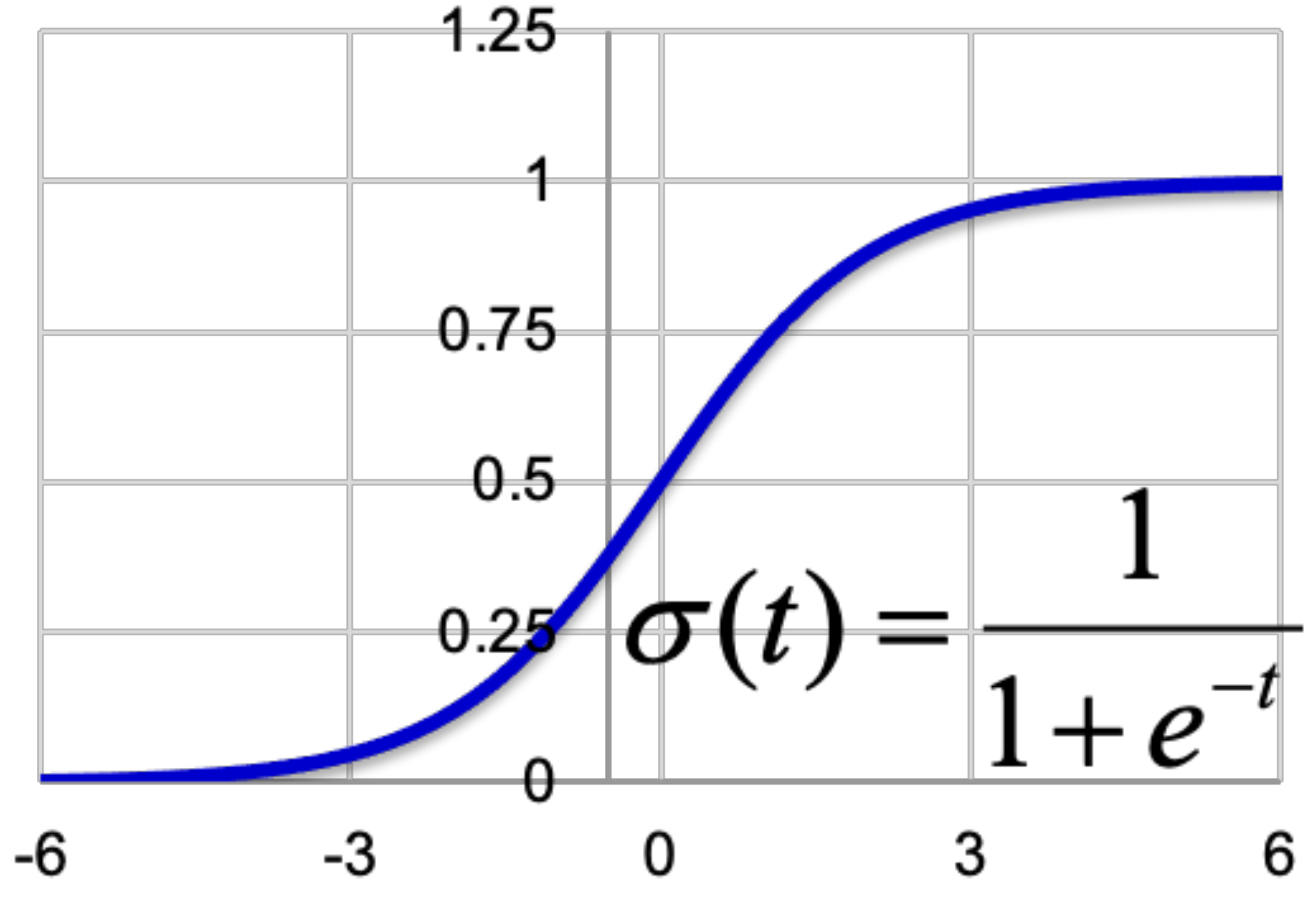
Recap: Perceptrons

[Rosenblatt, 1957]

Most basic form of a neural network



Sigmoid function:



Output: $\sigma(\mathbf{w} \cdot \mathbf{x} + b)$

Non-linearity

Linear combination of inputs

Bias

1958 New York Times...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

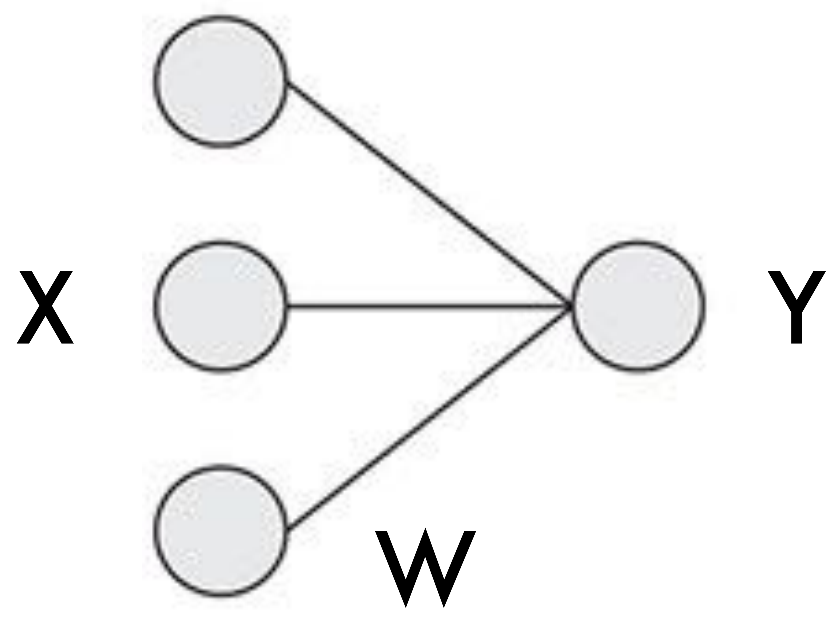
Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

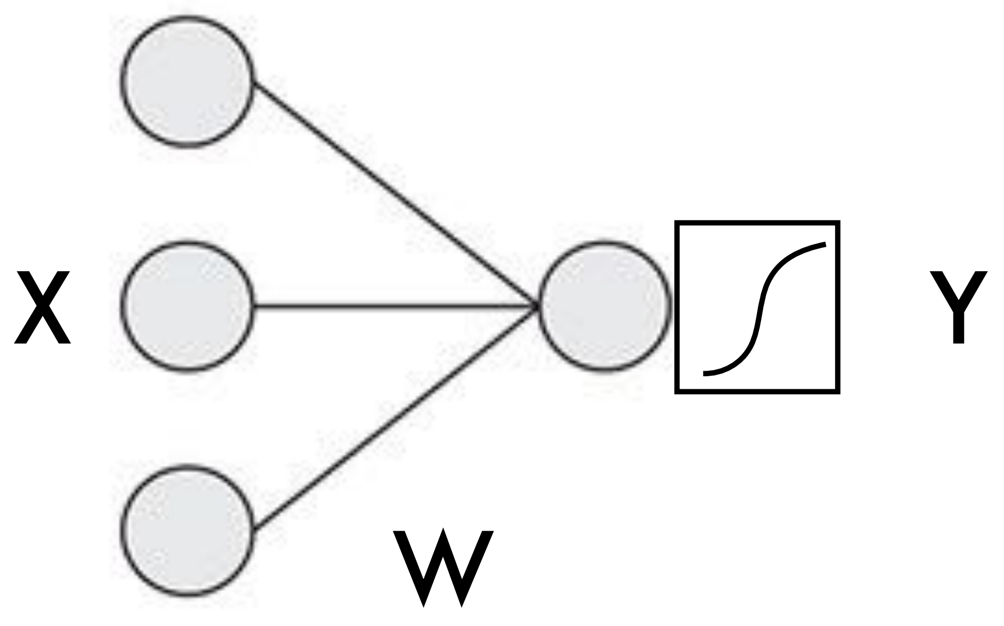
Slide credit: Lana Lazebnik

Recap: Multi-Layer Perceptron (MLP)

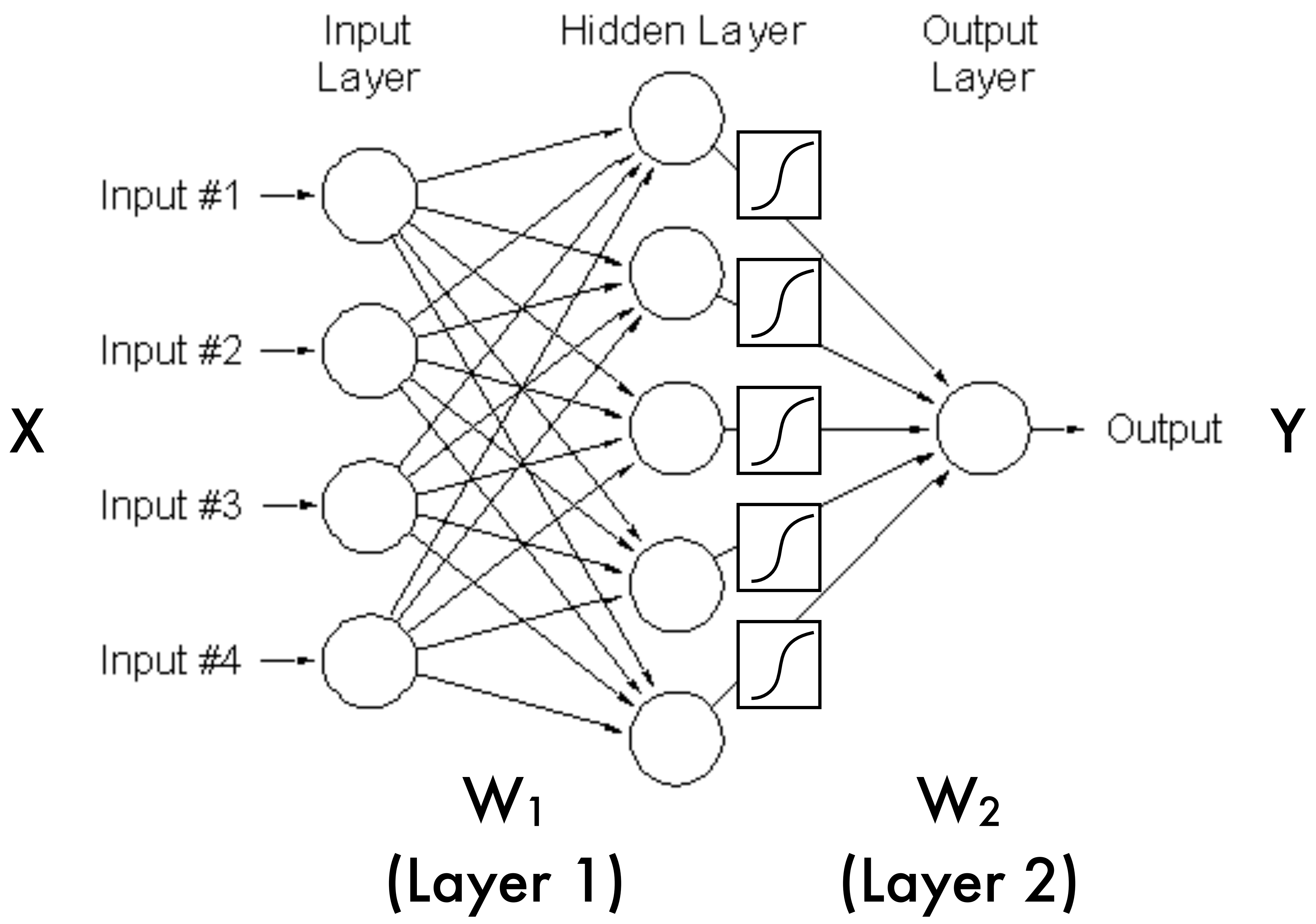
Linear regression:



Perceptron:

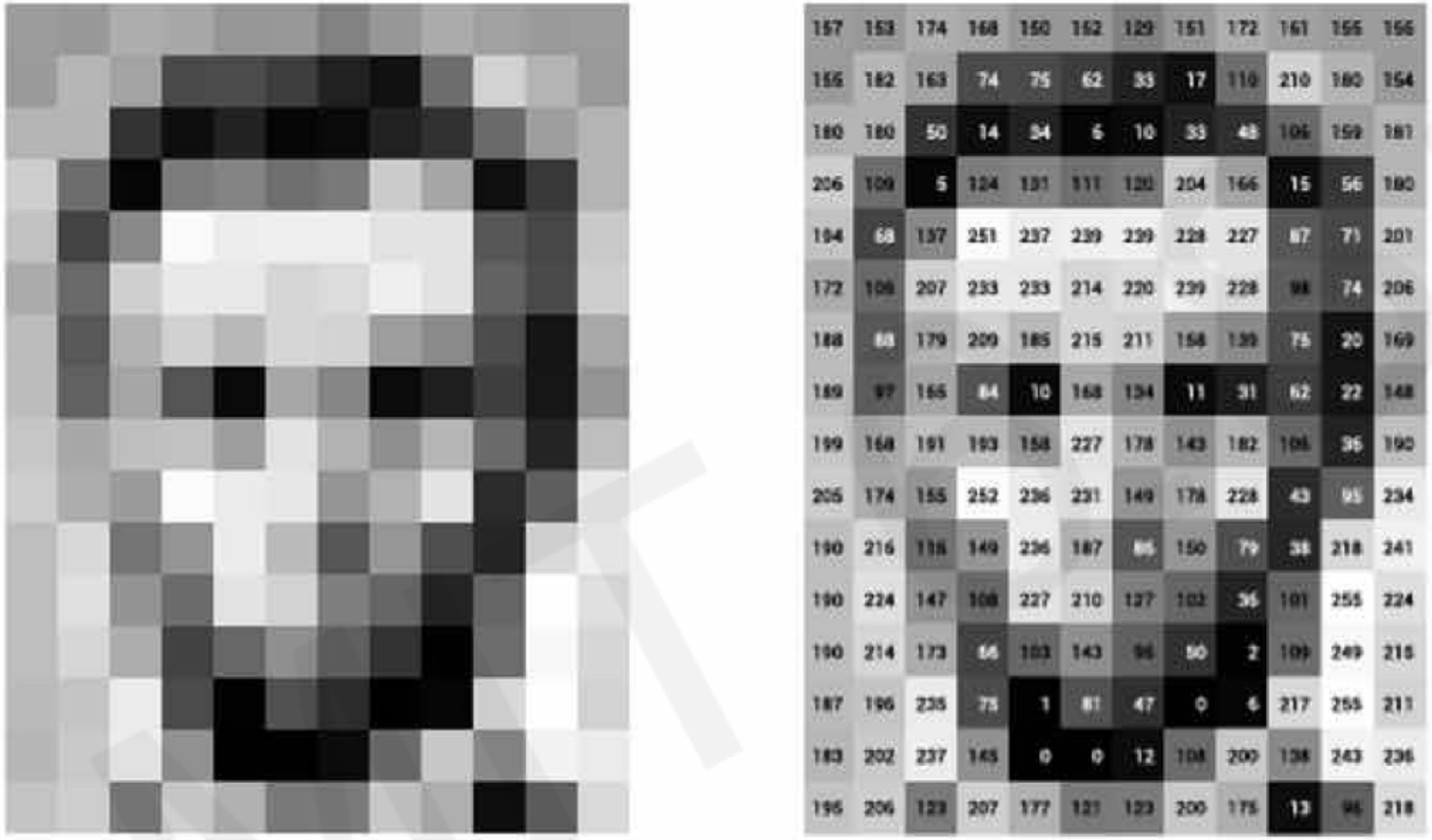


MLP:



Slide: R. Fergus / S. Lazebnik

Images are numbers



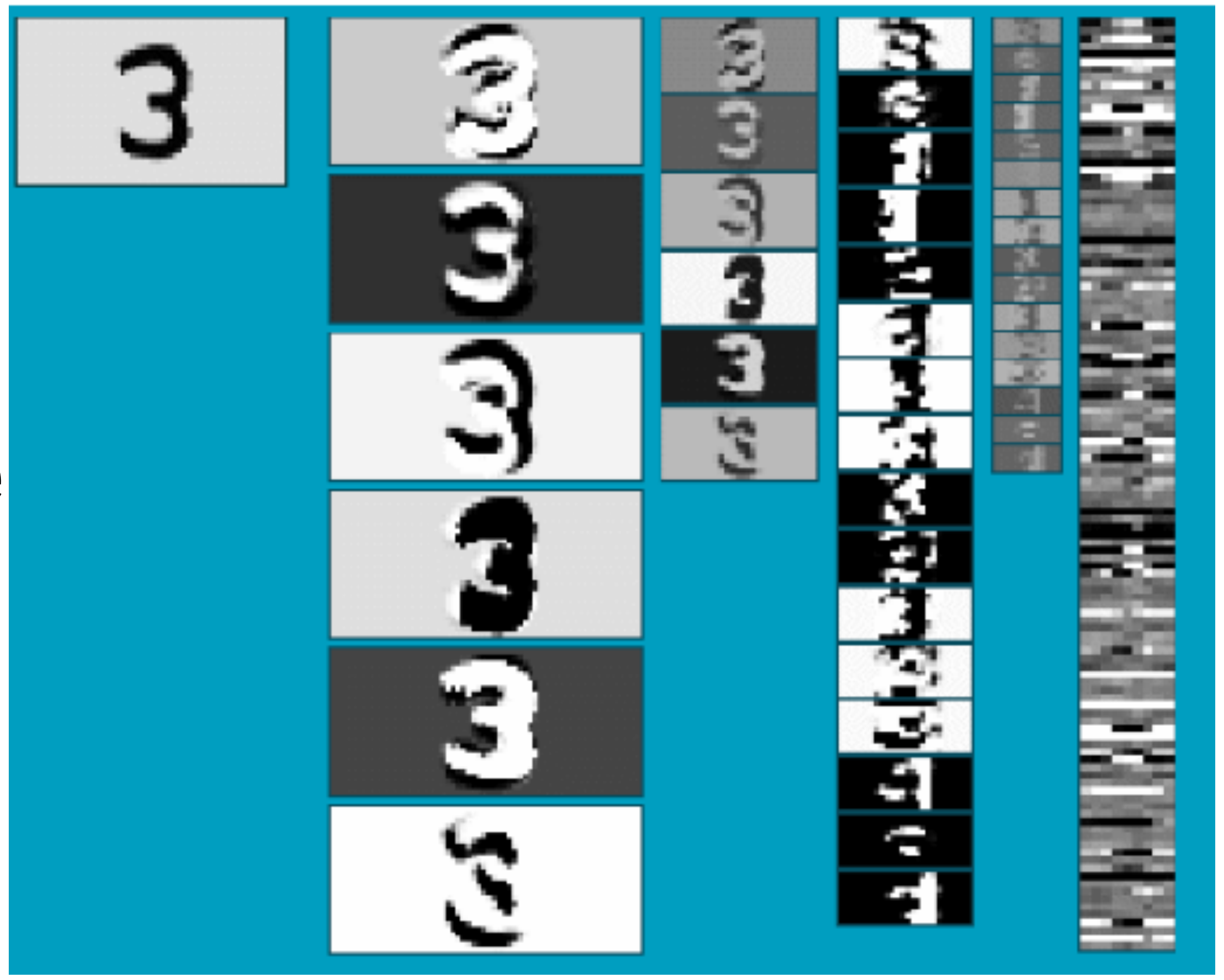
What the computer sees

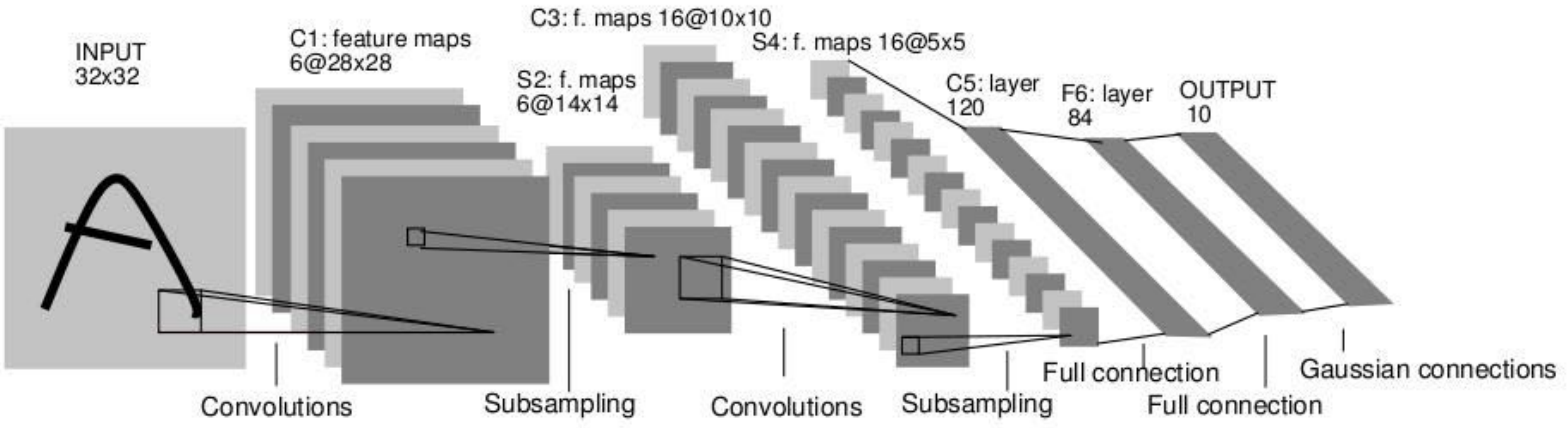
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

Review: Convolutional Neural Networks (CNN)

- Neural network with specialized **connectivity** structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant feature
- Classification layer at the end

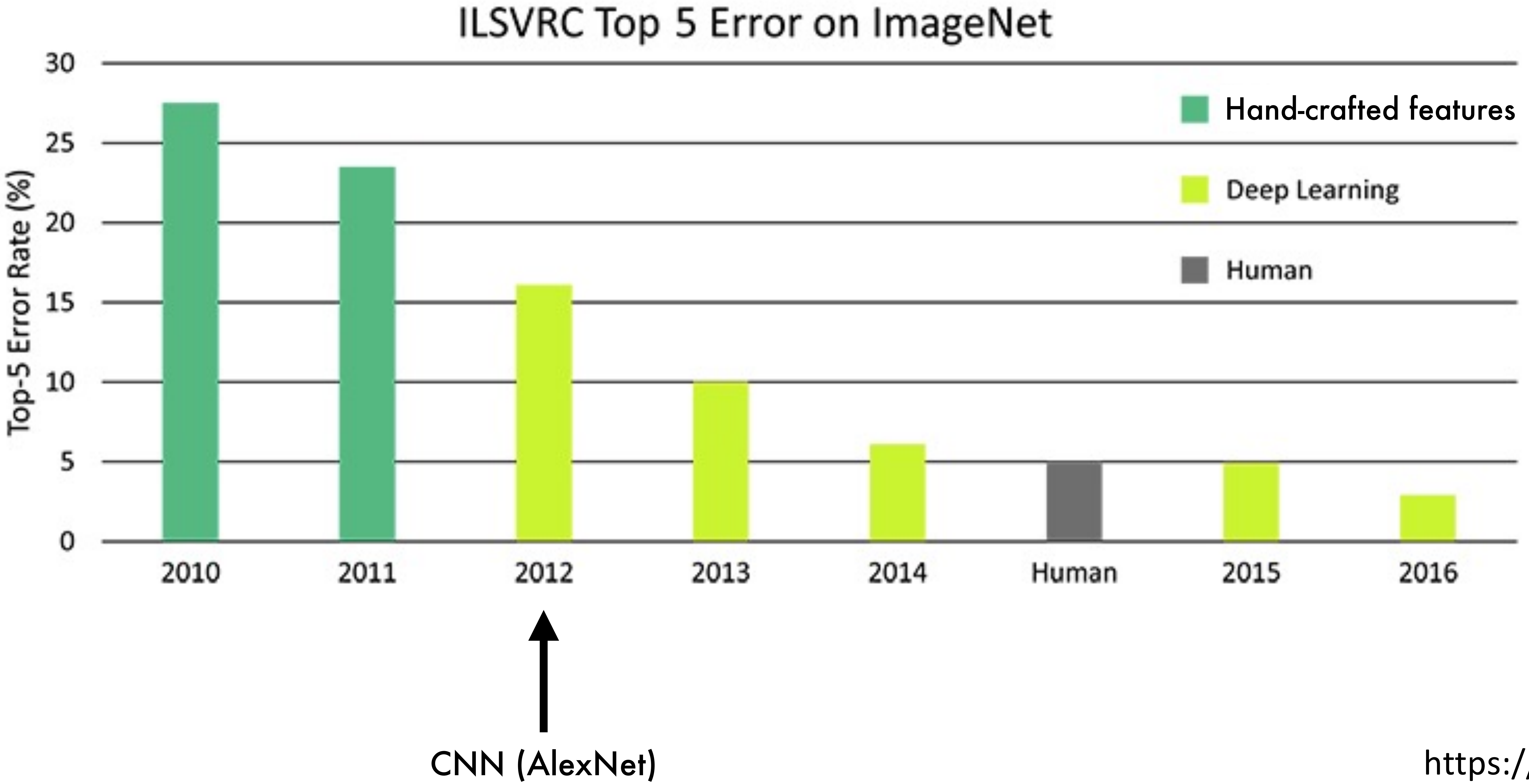




Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86(11): 2278–2324, 1998.

Slide: R. Fergus / S. Lazebnik

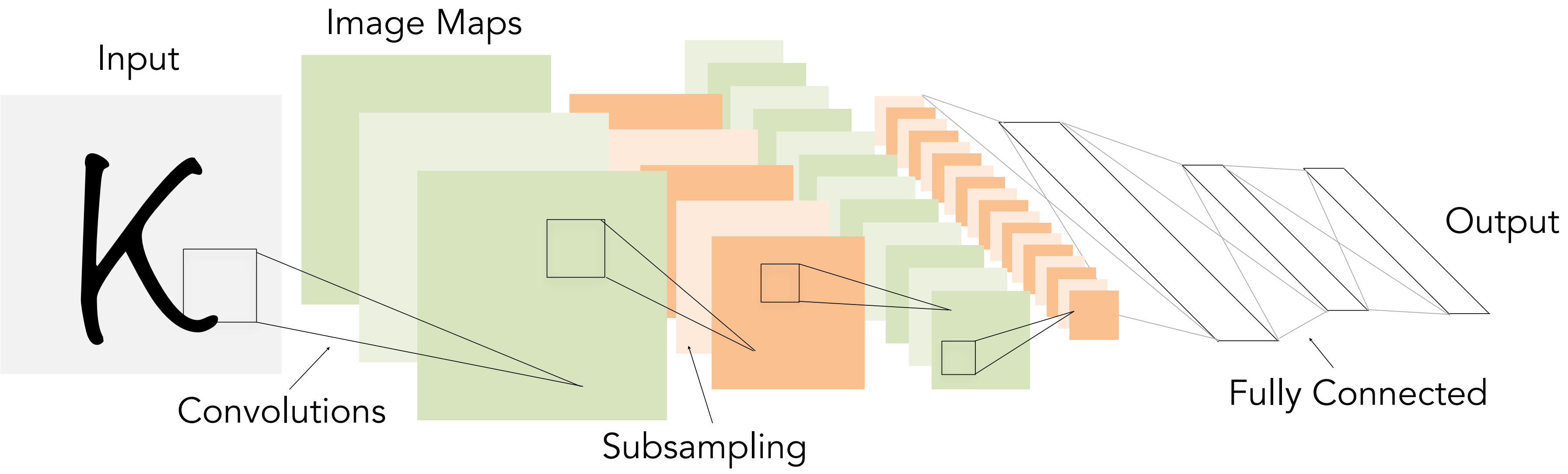
Progress on ImageNet



https://www.dsiac.org

CNNs were not invented overnight

1998
LeCun et al.



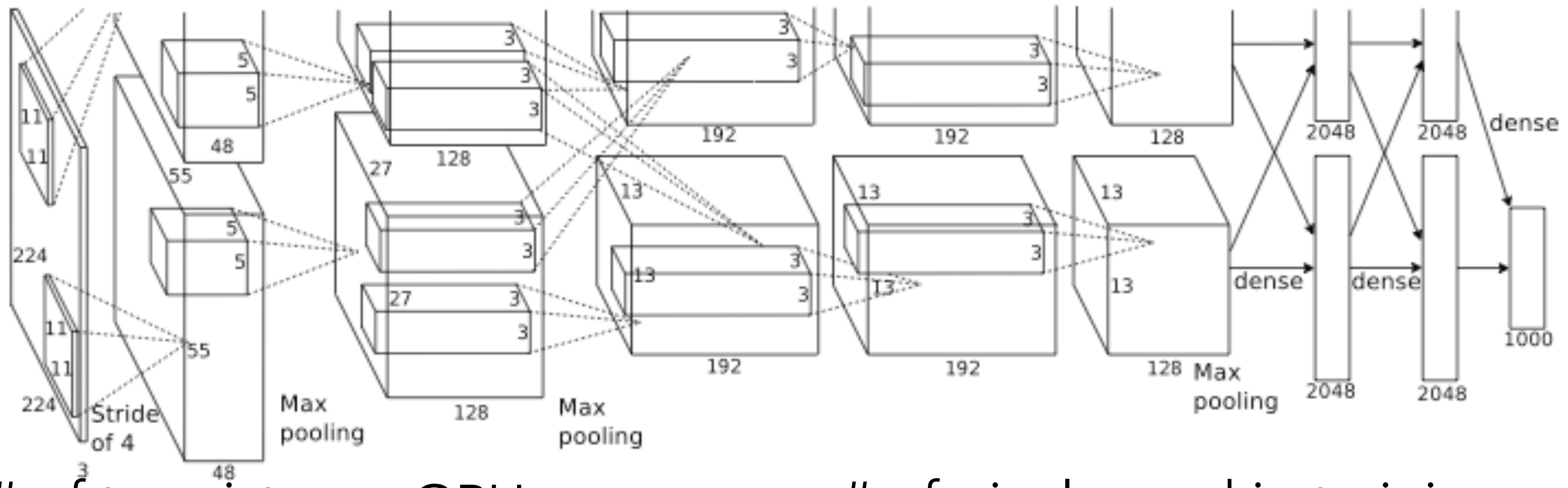
\# of transistors

10^6

\# of pixels used in training

10^7 NIST

2012
Krizhevsky et al.



\# of transistors GPUs

10^9

\# of pixels used in training

10^{14} IMAGENET

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

IMAGENET Large Scale Visual Recognition Challenge

Year 2010

NEC-UIUC

Dense descriptor grid: HOG, LBP

Coding: local coordinate, super-vector

Pooling, SPM

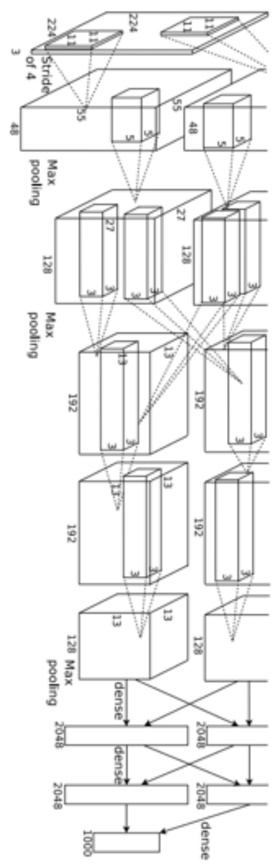
Linear SVM

[Lin CVPR 2011]

Lion image by Swissfrog is licensed under CC BY 3.0

Year 2012

SuperVision

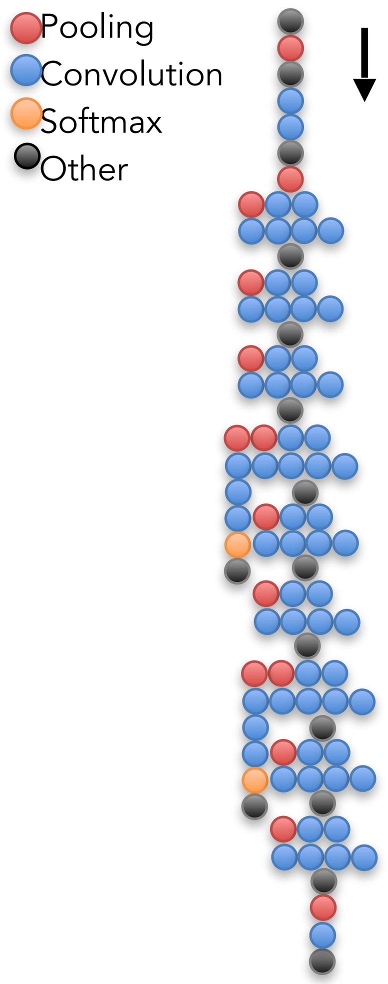


[Krizhevsky NIPS 2012]

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

Year 2014

GoogLeNet



[Szegedy arxiv 2014]

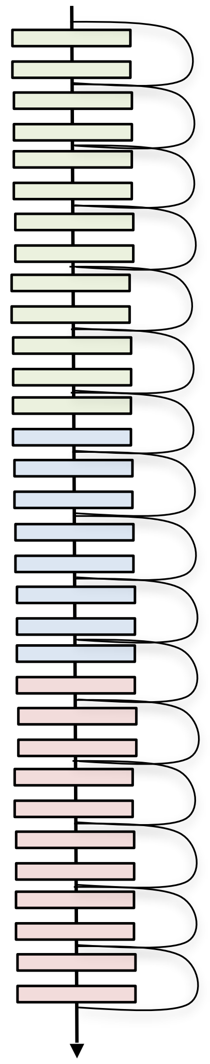
VGG

Image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
fc-4096
fc-4096
fc-1000
softmax

[Simonyan arxiv 2014]

Year 2015

MSRA



[He ICCV 2015]

Slide credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Why now?

Neural Networks date back decades.

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

2012 AlexNet

⋮

1. Big Data

- Larger datasets
- Easier collection & storage

LAION-400-MILLION OPEN DATASET

...

2. Hardware

- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software

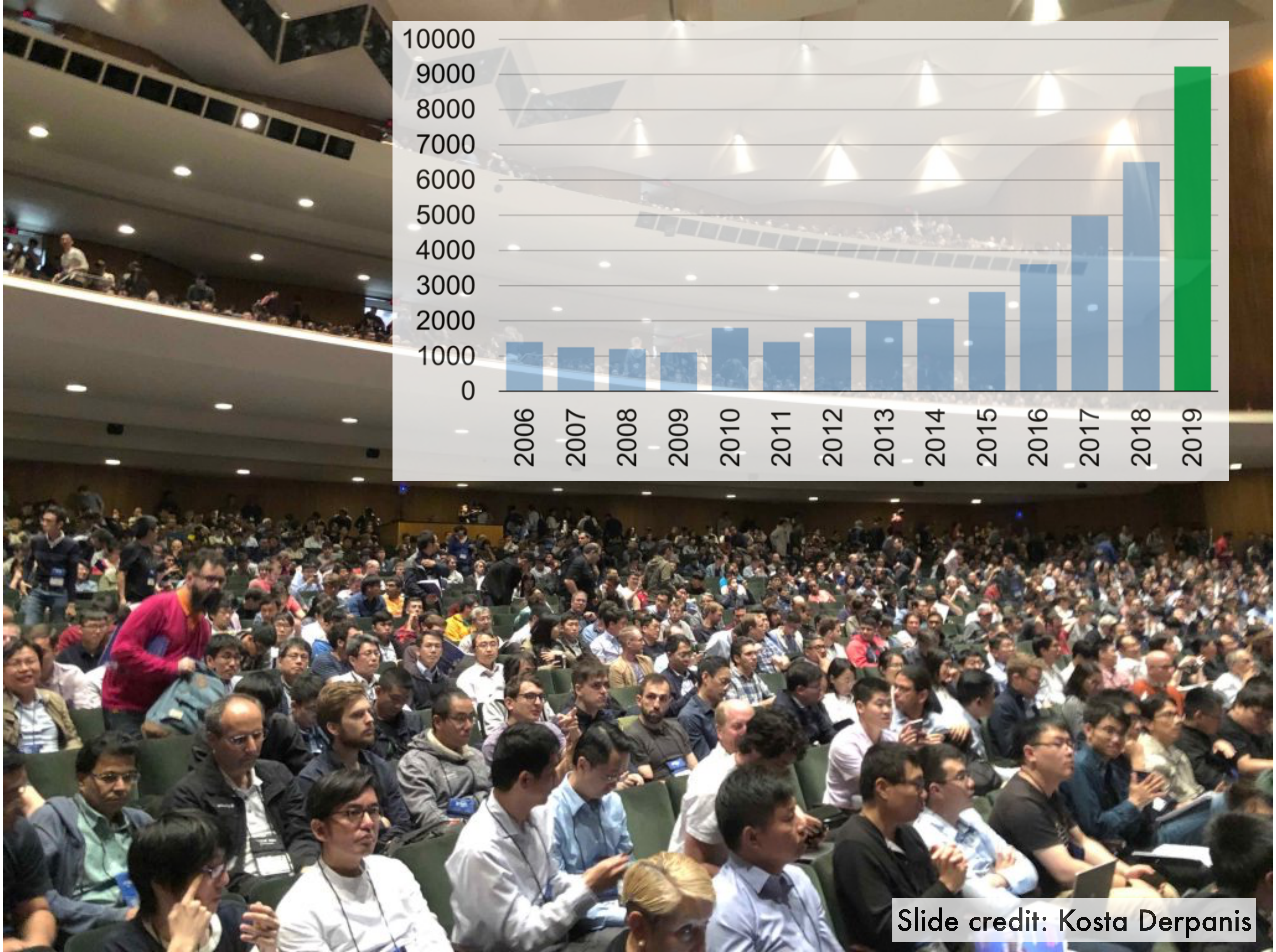
- Improved Techniques
- New Models
- Toolboxes

PyTorch

TensorFlow

...

Slide credit: Alexander Amini



CVPR:
(**C**omputer **V**ision **P**attern **R**ecognition Conference)

Agenda

- **1. Recap: Bag of Visual Words, Analogy with NNs**
- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs
- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures
- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers
- **5. Beyond classification**

Agenda

- **1. Recap: Bag of Visual Words, Analogy with NNs**

- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs

- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures

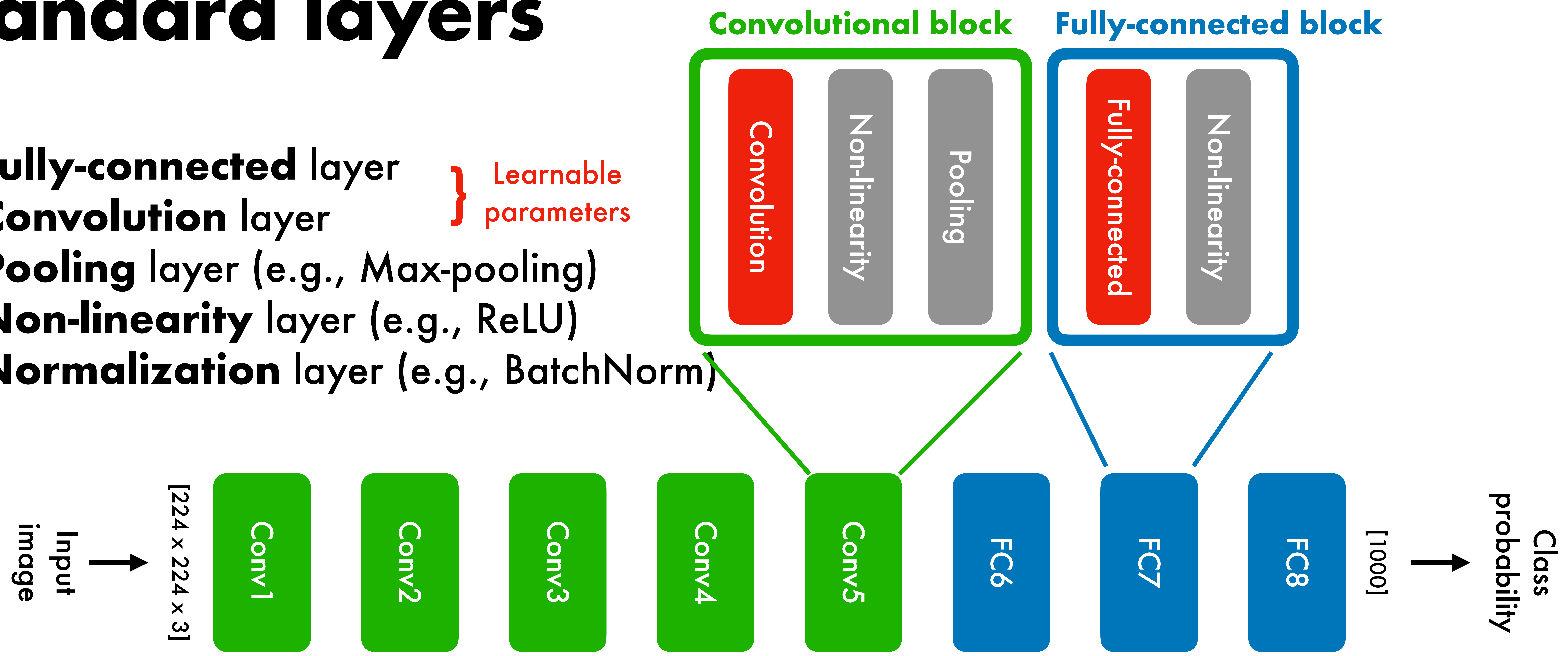
- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers

- **5. Beyond classification**

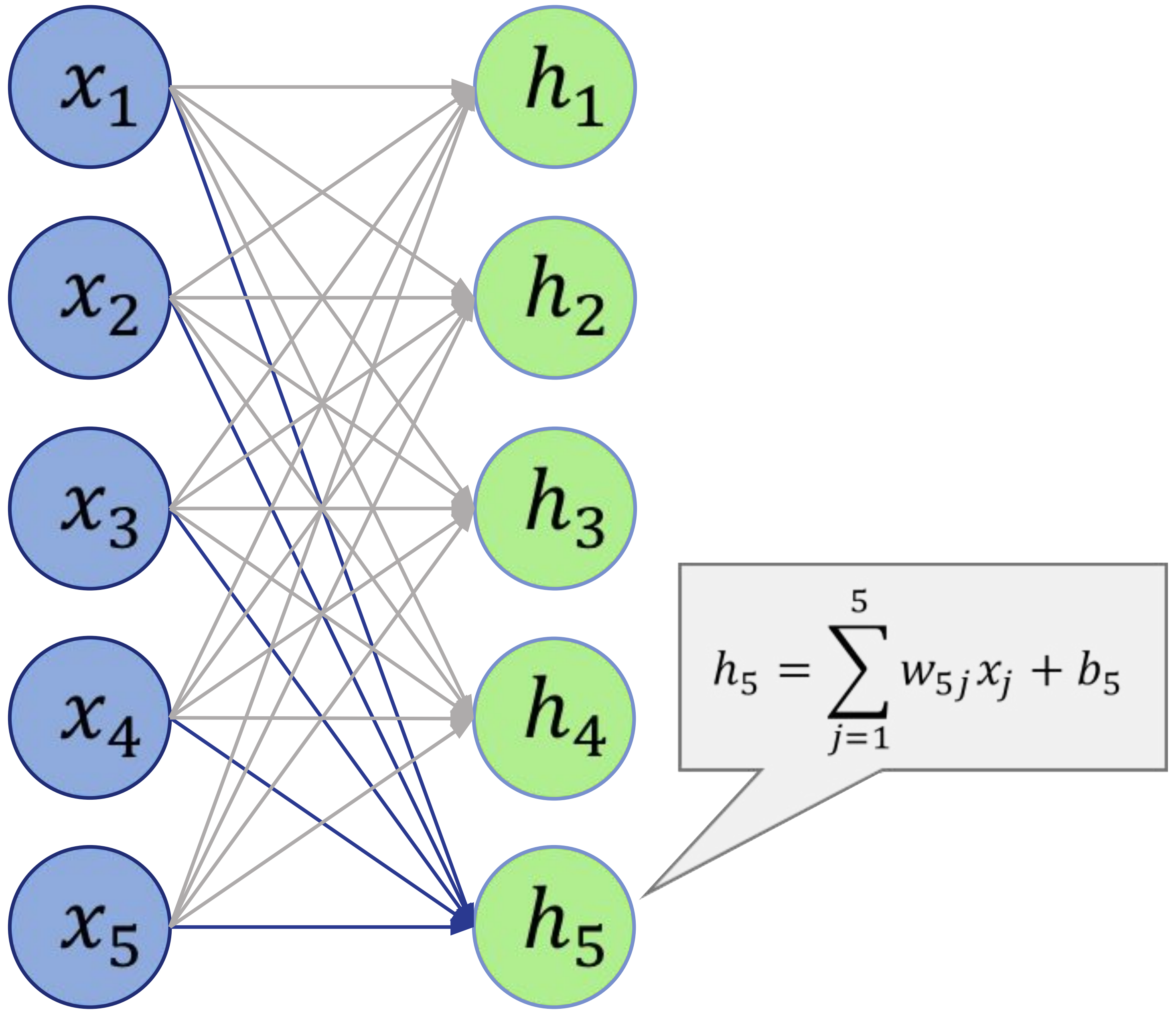
Standard layers

- 1. **Fully-connected** layer
- 2. **Convolution** layer
- 3. **Pooling** layer (e.g., Max-pooling)
- 4. **Non-linearity** layer (e.g., ReLU)
- 5. **Normalization** layer (e.g., BatchNorm)
- ...

} Learnable parameters



1. Fully-connected layer

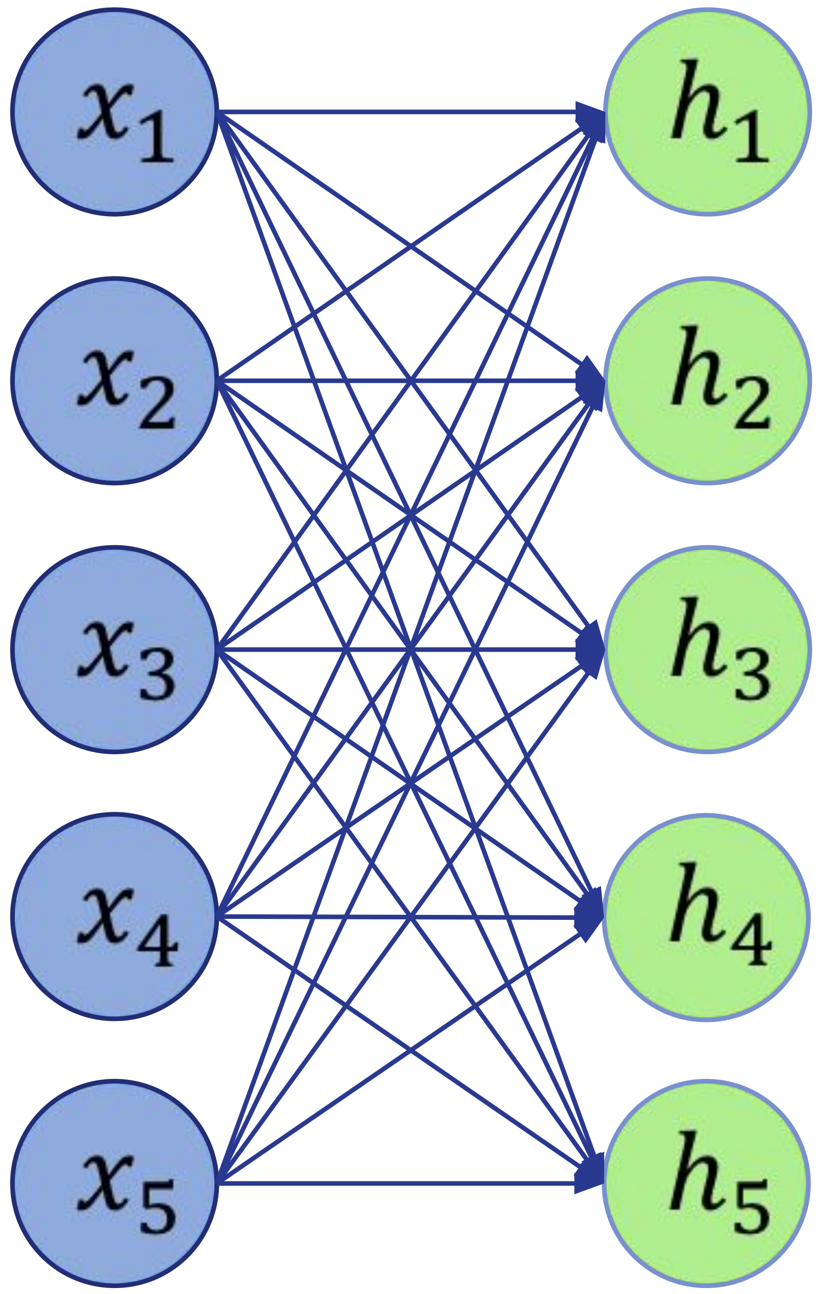


$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

Slide credit: Naila Murray

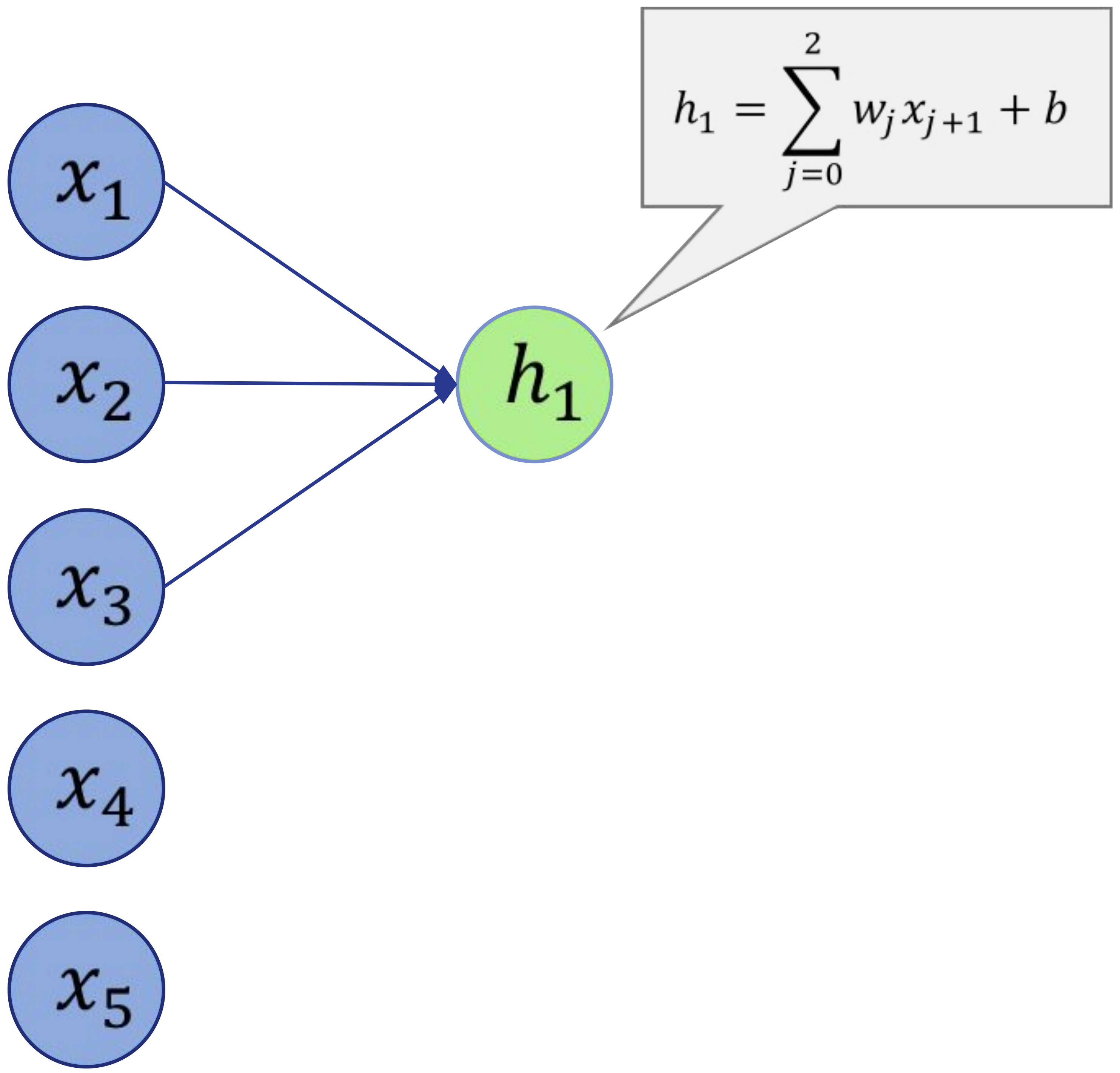
2. Convolution layer

Fully-connected



$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

1D Convolutional



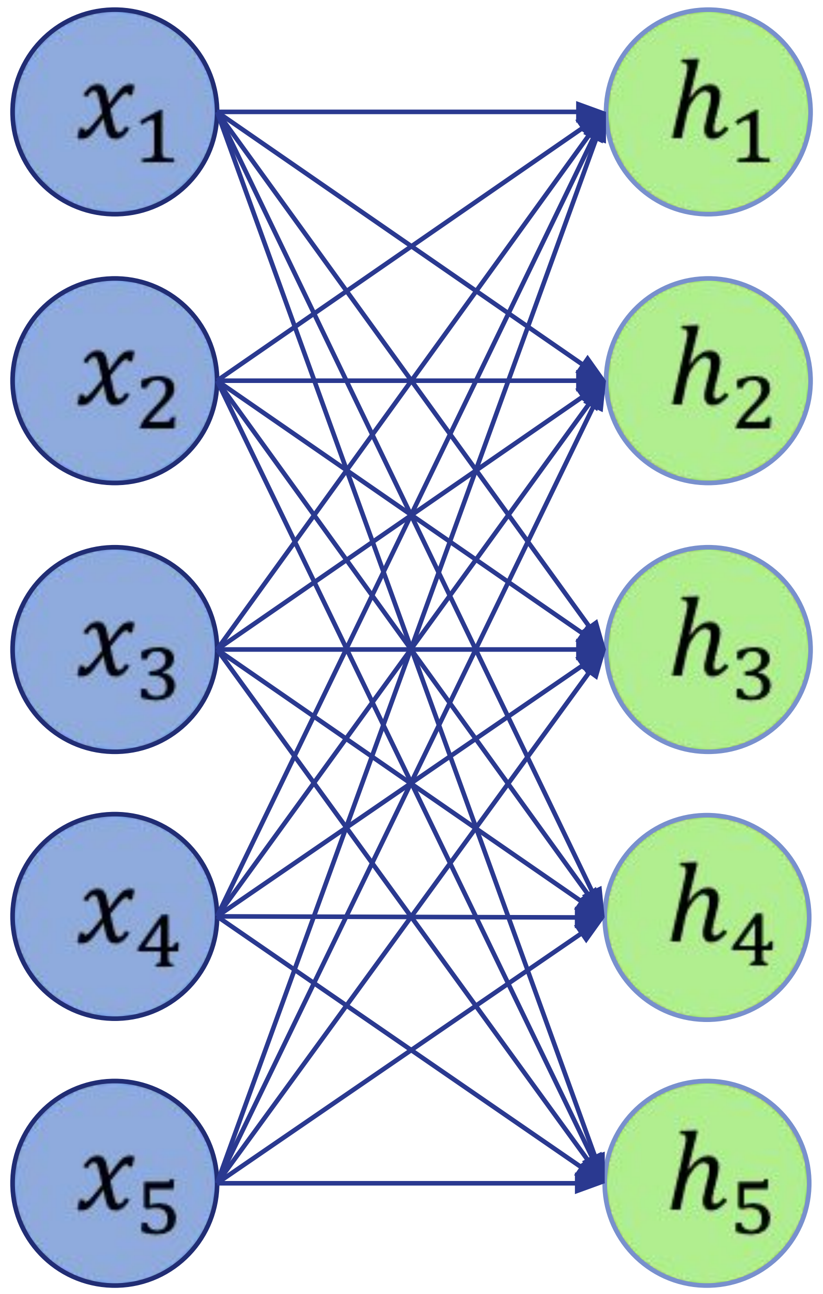
$$h_i = \sum_j w_j x_{j+i} + b$$

- Layer with a special connectivity structure
- Dependencies are local
- Translation invariance

Slide credit: Naila Murray

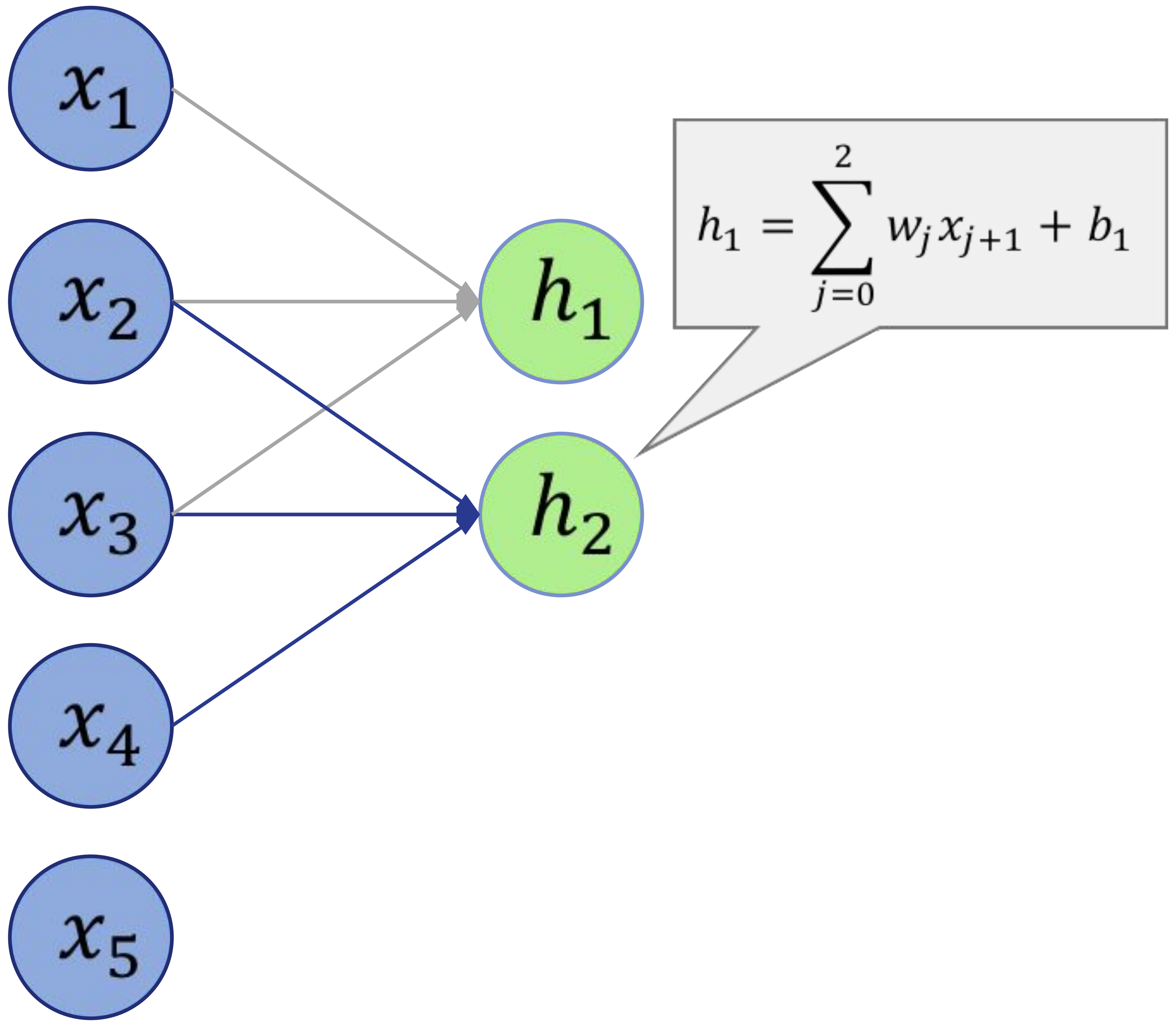
2. Convolution layer

Fully-connected



$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

1D Convolutional

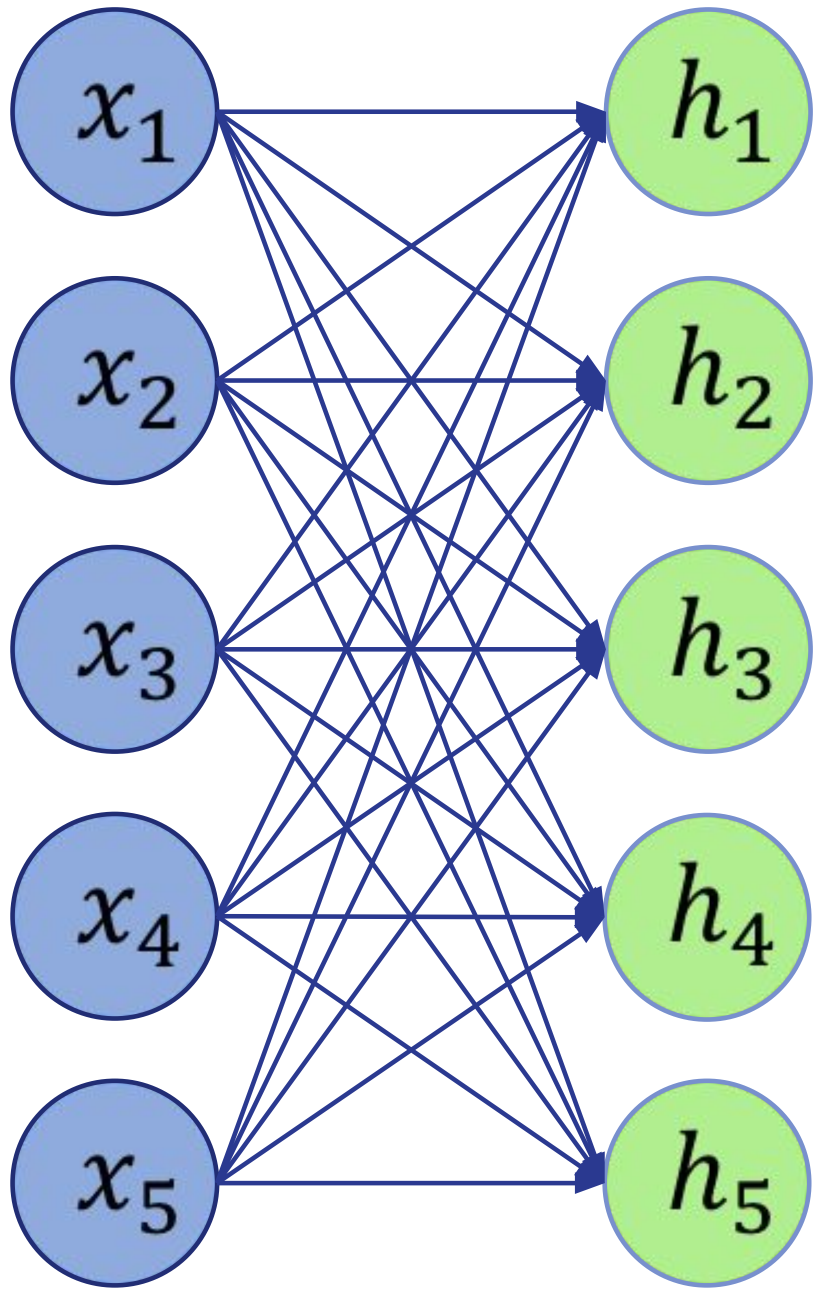


$$h_i = \sum_j w_j x_{j+i} + b$$

Slide credit: Naila Murray

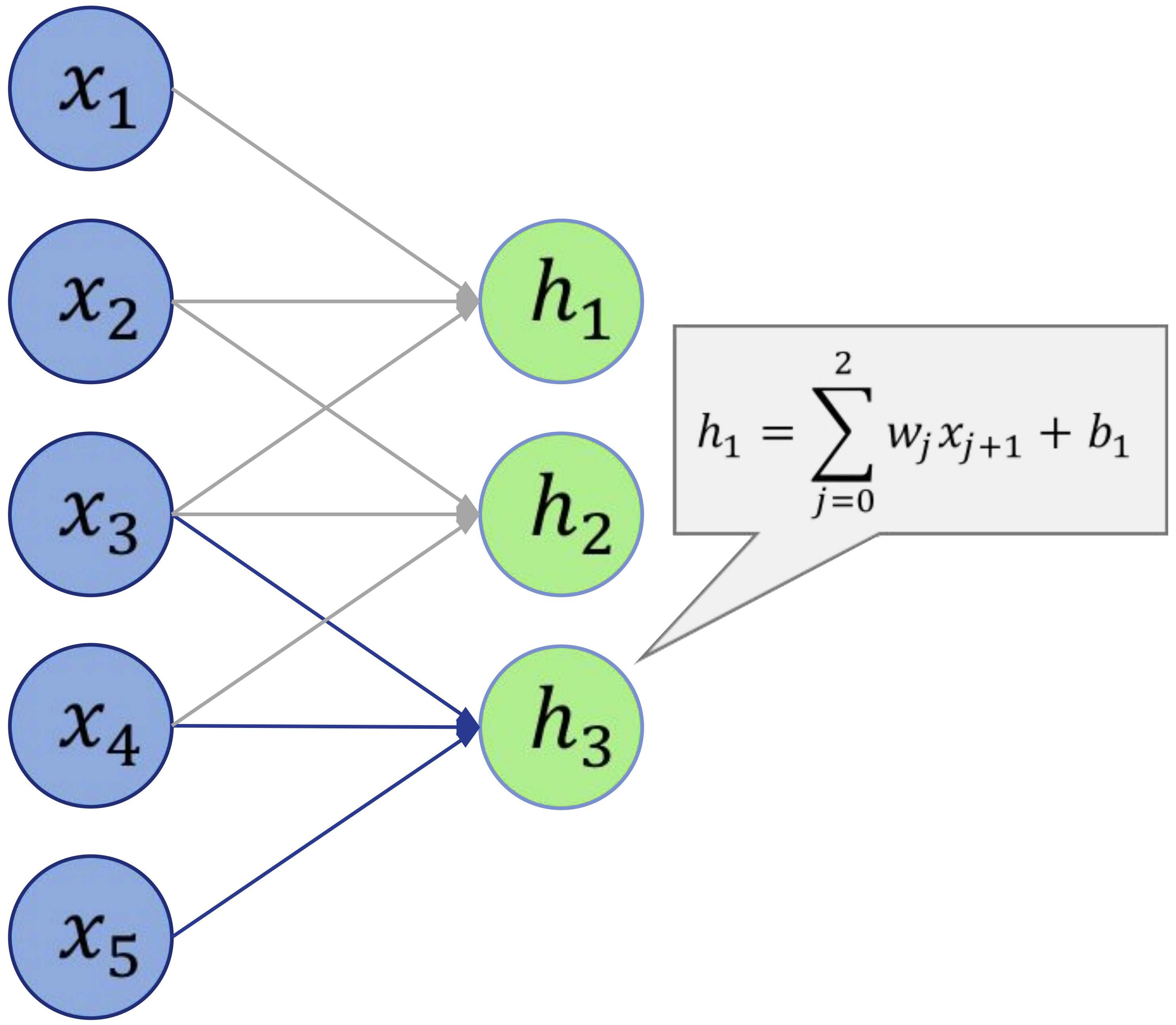
2. Convolution layer

Fully-connected



$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

1D Convolutional



$$h_i = \sum_j w_j x_{j+i} + b$$

Slide credit: Naila Murray

2. Convolution layer

2D Convolutions

$$I = \begin{bmatrix} 1 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 1 & 2 & 1 & 1 \\ 1 & 1 & 3 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 \end{bmatrix} \star K = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

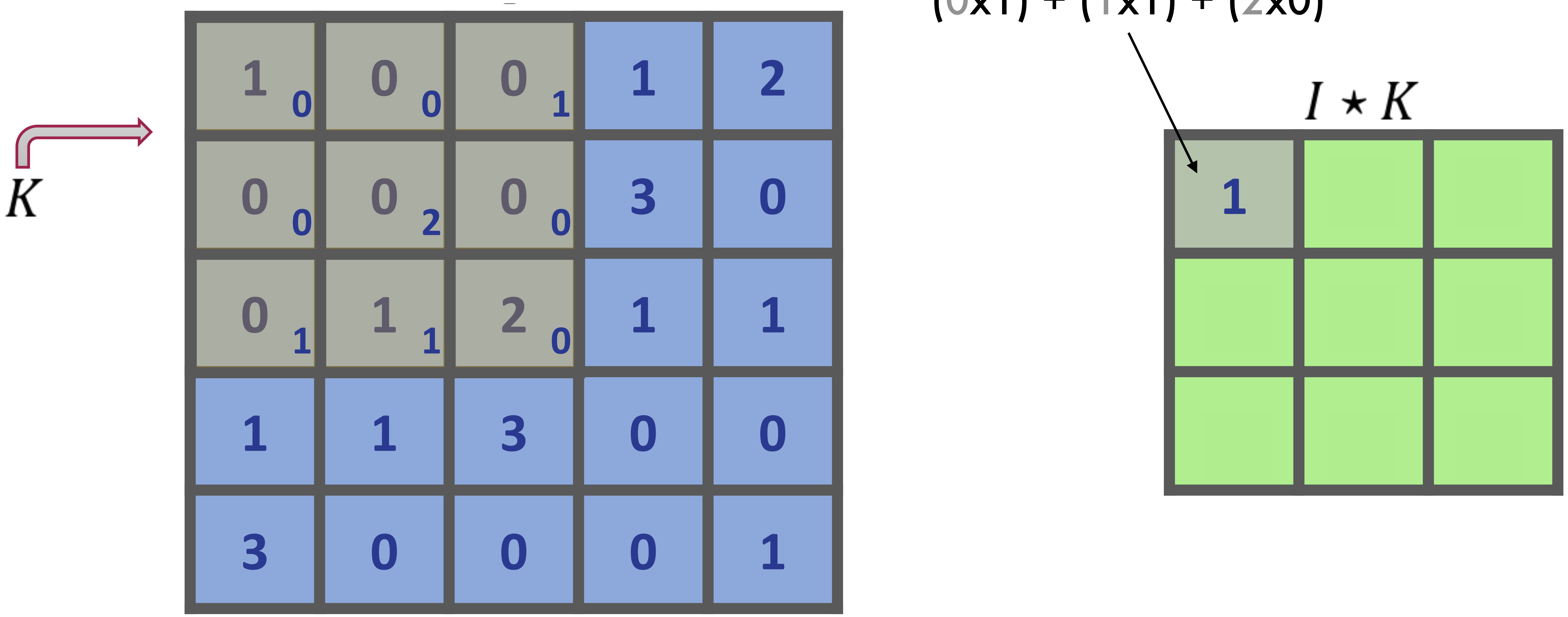
$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m, j+n)$$

Slide credit: Naila Murray

2. Convolution layer

2D Convolutions

(1x0) + (0x0) + (0x1) +
(0x0) + (0x2) + (0x0) +
(0x1) + (1x1) + (2x0)

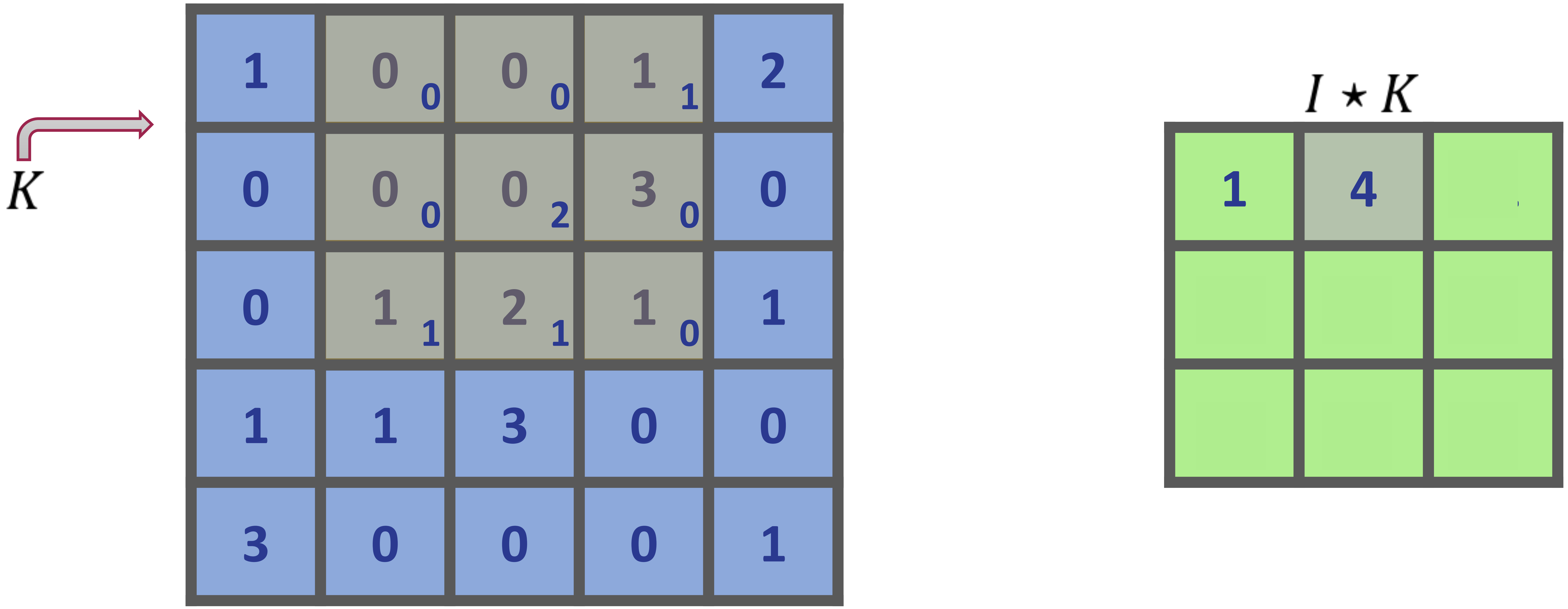


$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m, j+n)$$

Slide credit: Naila Murray

2. Convolution layer

2D Convolutions

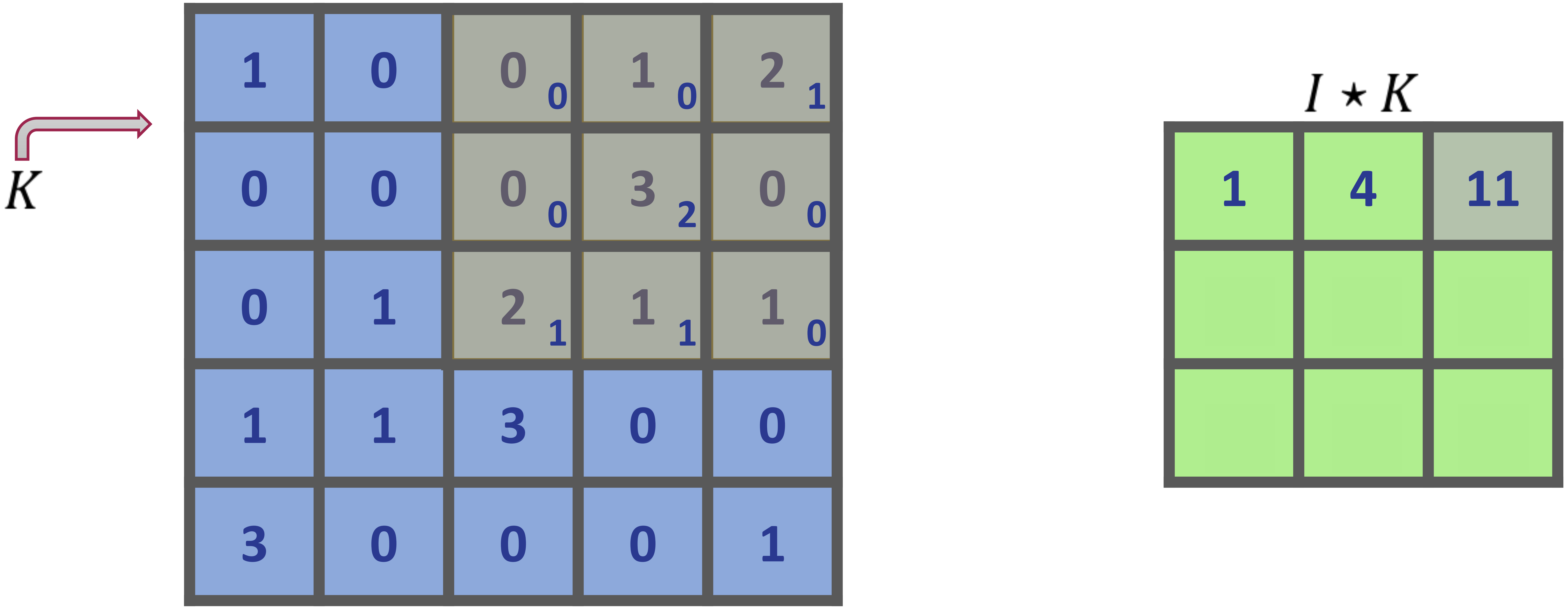


$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m, j+n)$$

Slide credit: Naila Murray

2. Convolution layer

2D Convolutions

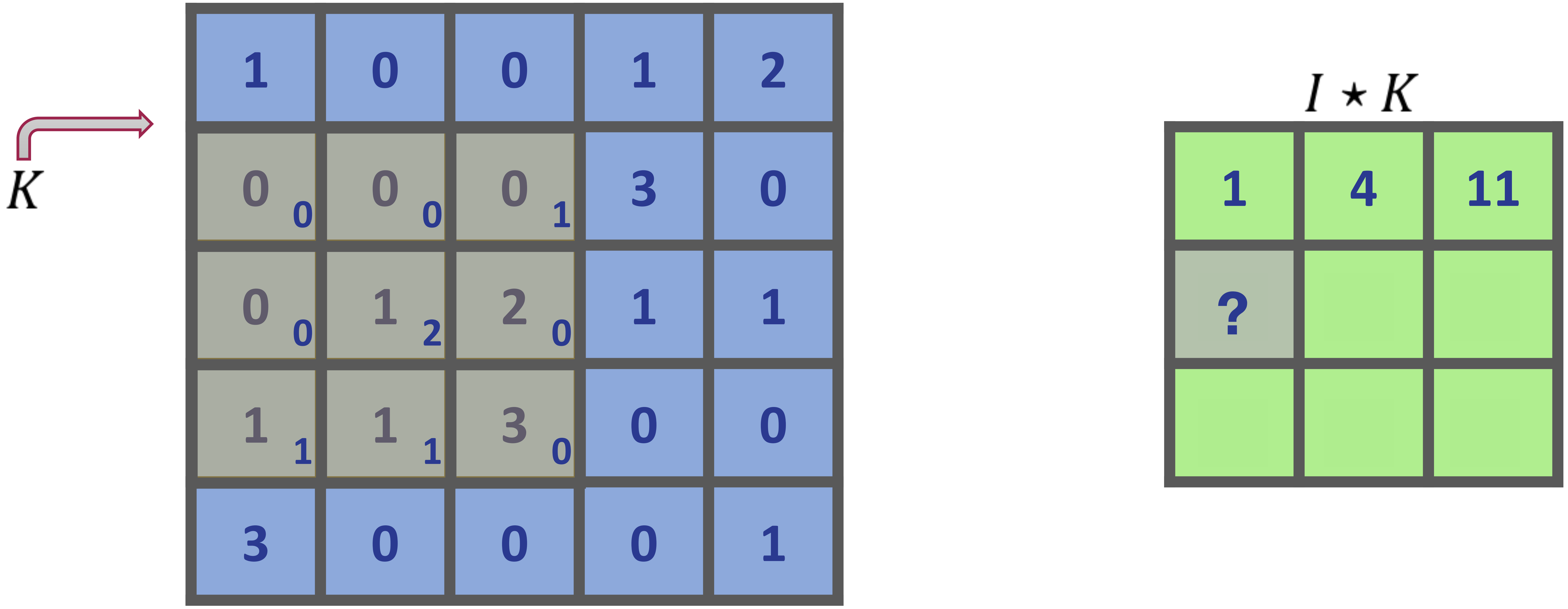


$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m, j+n)$$

Slide credit: Naila Murray

2. Convolution layer

2D Convolutions



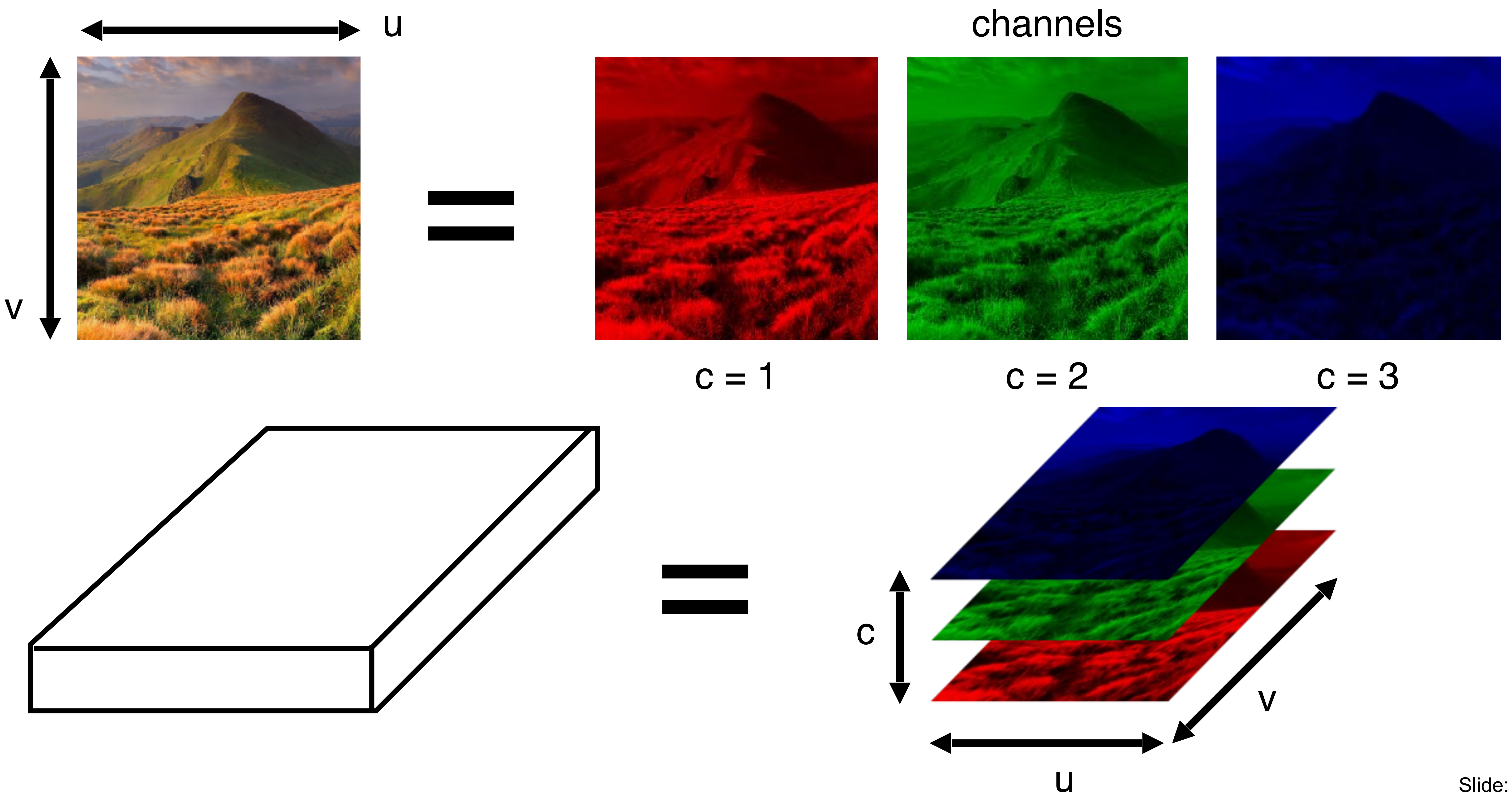
$$(I \star K)(i,j) == \sum_m \sum_n I(m,n)K(i+m,j+n)$$

Slide credit: Naila Murray

2. Convolution layer

The data manipulated by a CNN has the form of 3D tensors. These are interpreted as **discrete vector fields x**, assigning a feature vector (x_{uv1}, …, x_{uvC}) at each spatial location (v,u).

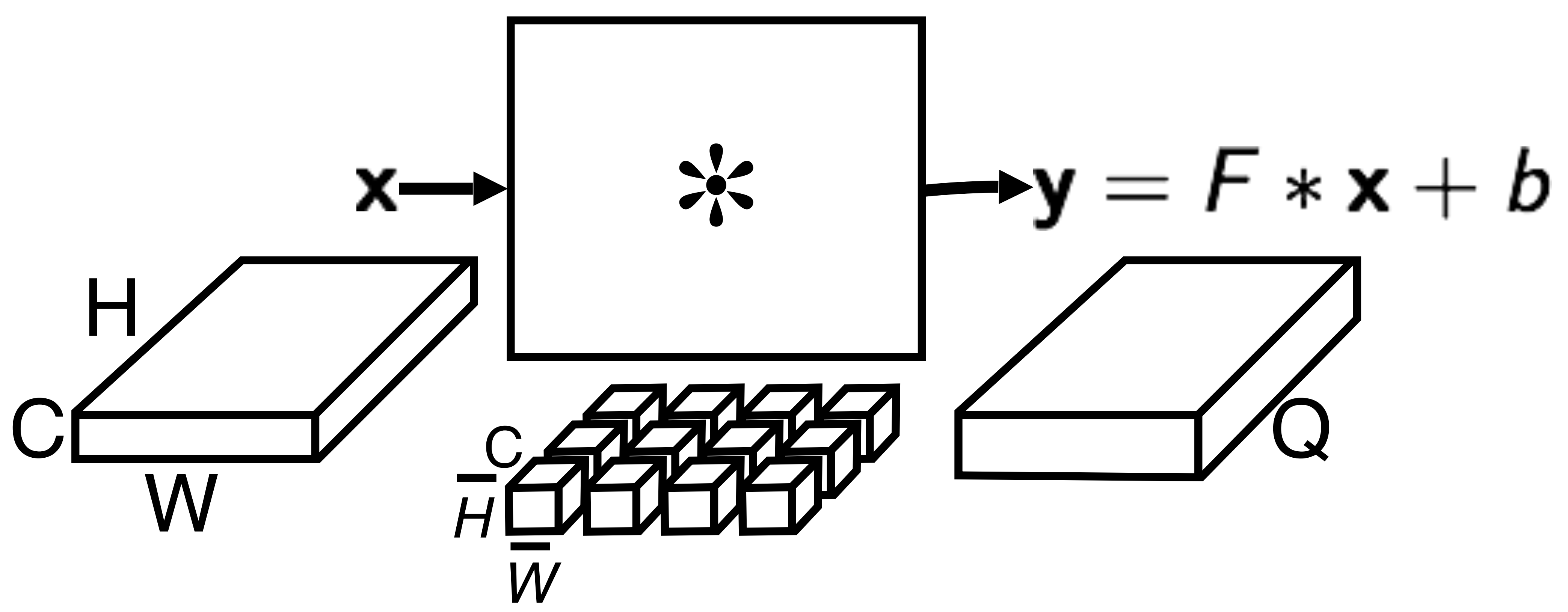
A colour image is a simple example of a vector field with 3D features (RGB):



Slide: A. Vedaldi

2. Convolution layer

With a bank of 3D filters



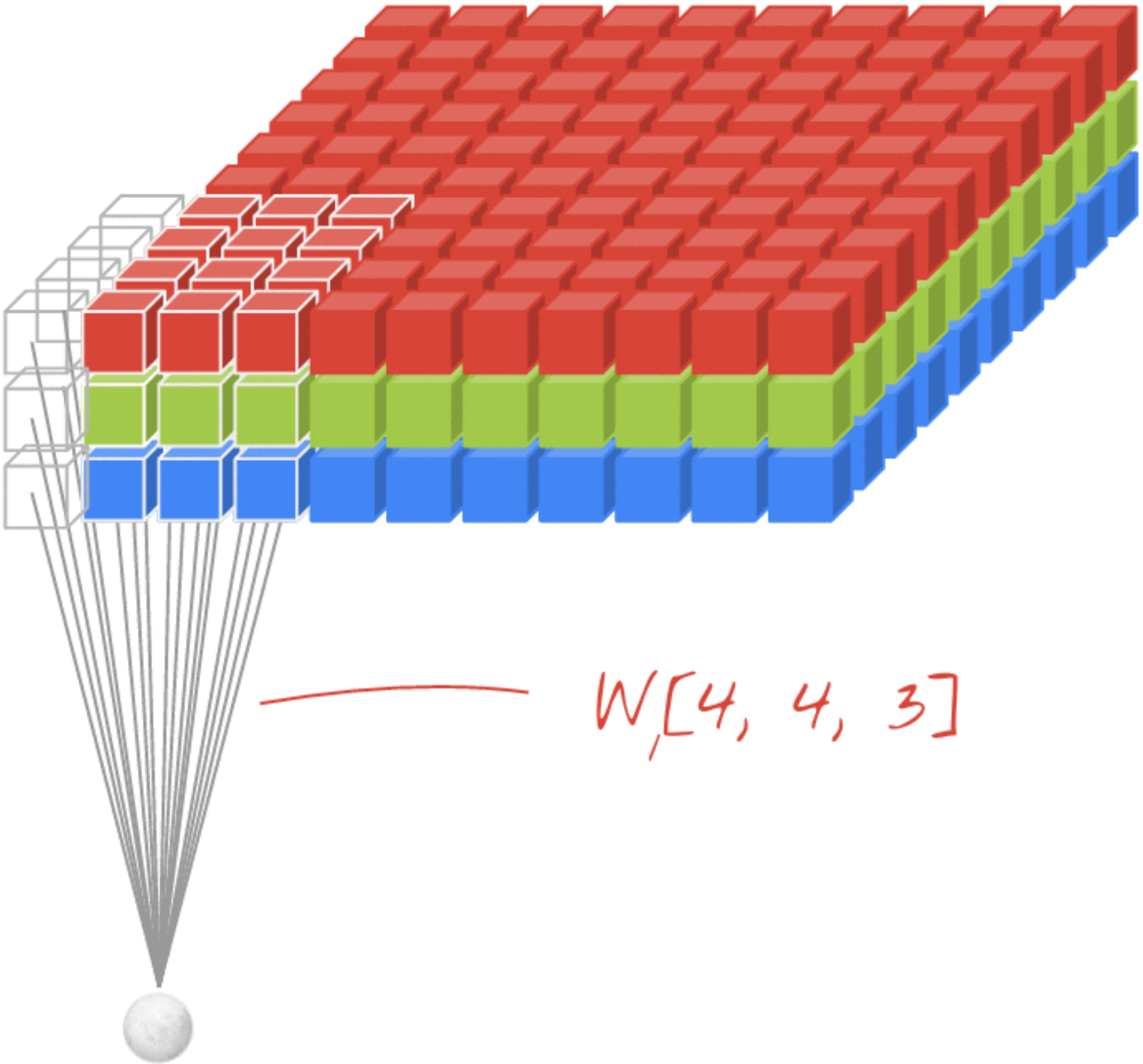
$$y_{v'u'q'} = b_{q'} + \sum_{\bar{v}=1}^{\bar{H}} \sum_{\bar{u}=1}^{\bar{W}} \sum_{c=1}^{C} x_{v'+\bar{v}-1,u'+\bar{u}-1,c} f_{\bar{v}\bar{u}cq'}$$

Linear convolution applies a bank of linear filters *F* to the input tensor **x**.

- Input tensor **x** = H × W × C array
- Filter bank *F* = $\bar{\text{H}}$ × $\bar{\text{W}}$ × C × Q array
- Output tensor **y** = (H - $\bar{\text{H}}$ + 1) × (W - $\bar{\text{W}}$ + 1) × Q array

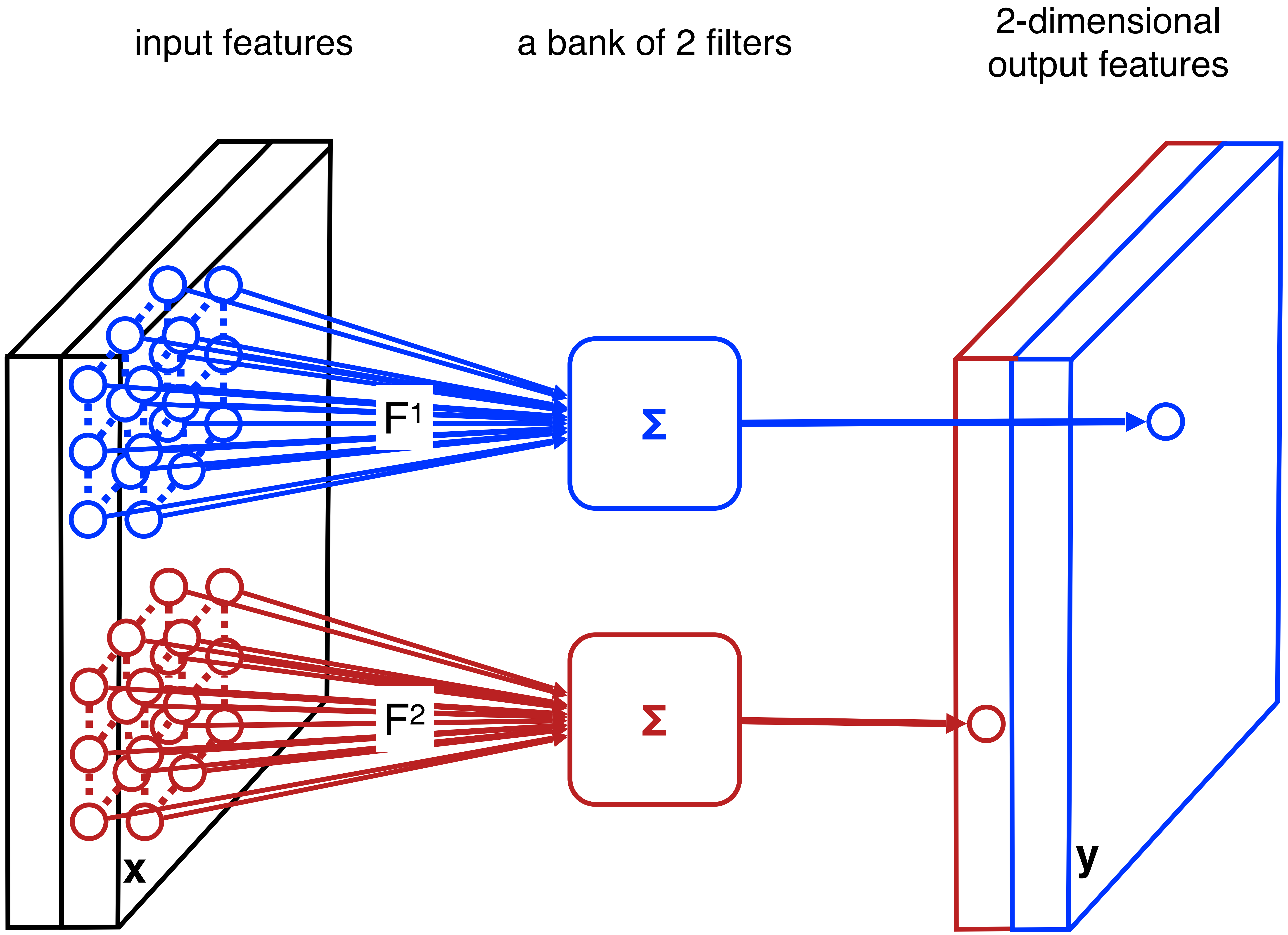
Slide: A. Vedaldi

2. Convolution layer



2. Convolution layer

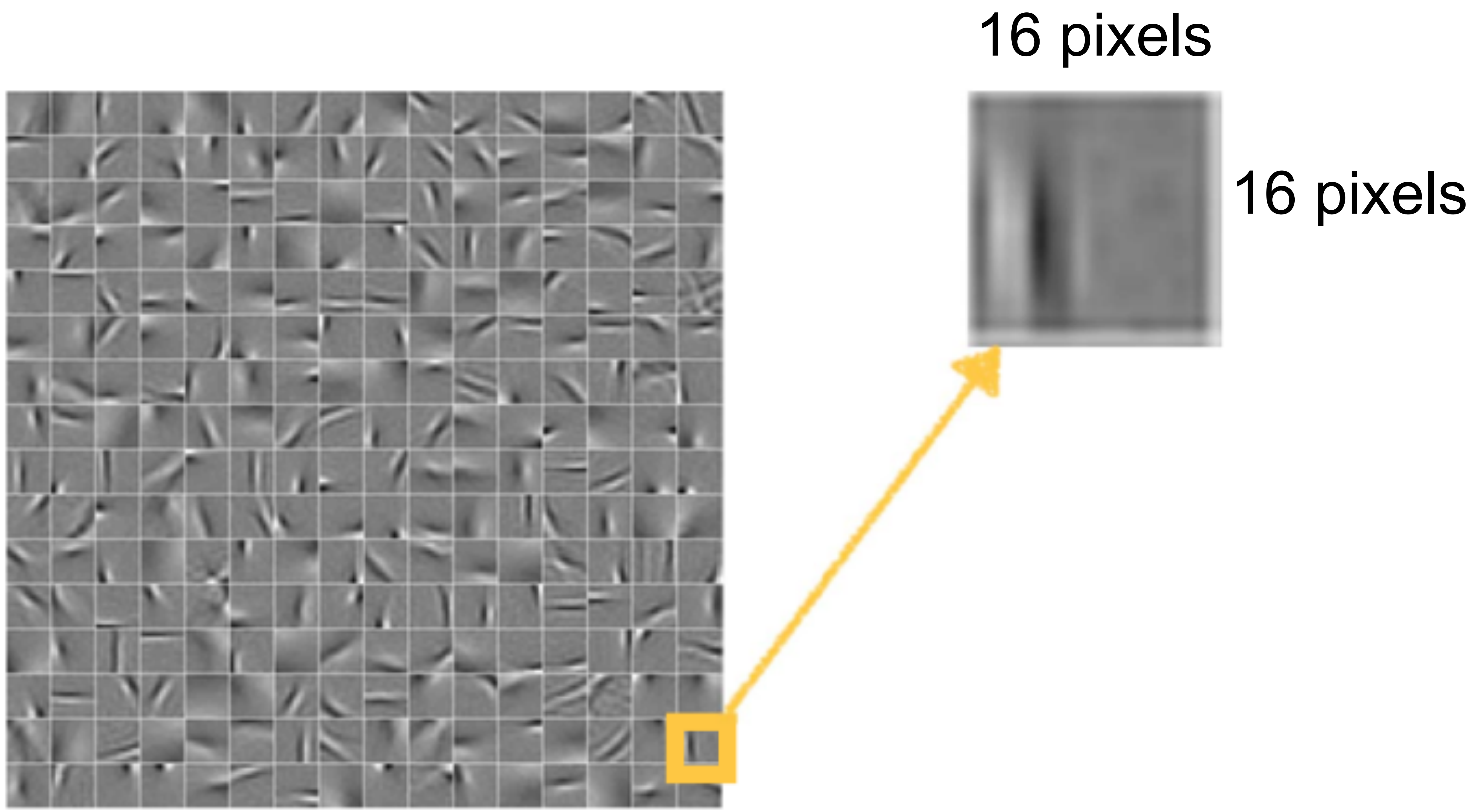
As a neural network



Slide: A. Vedaldi

Filter bank example

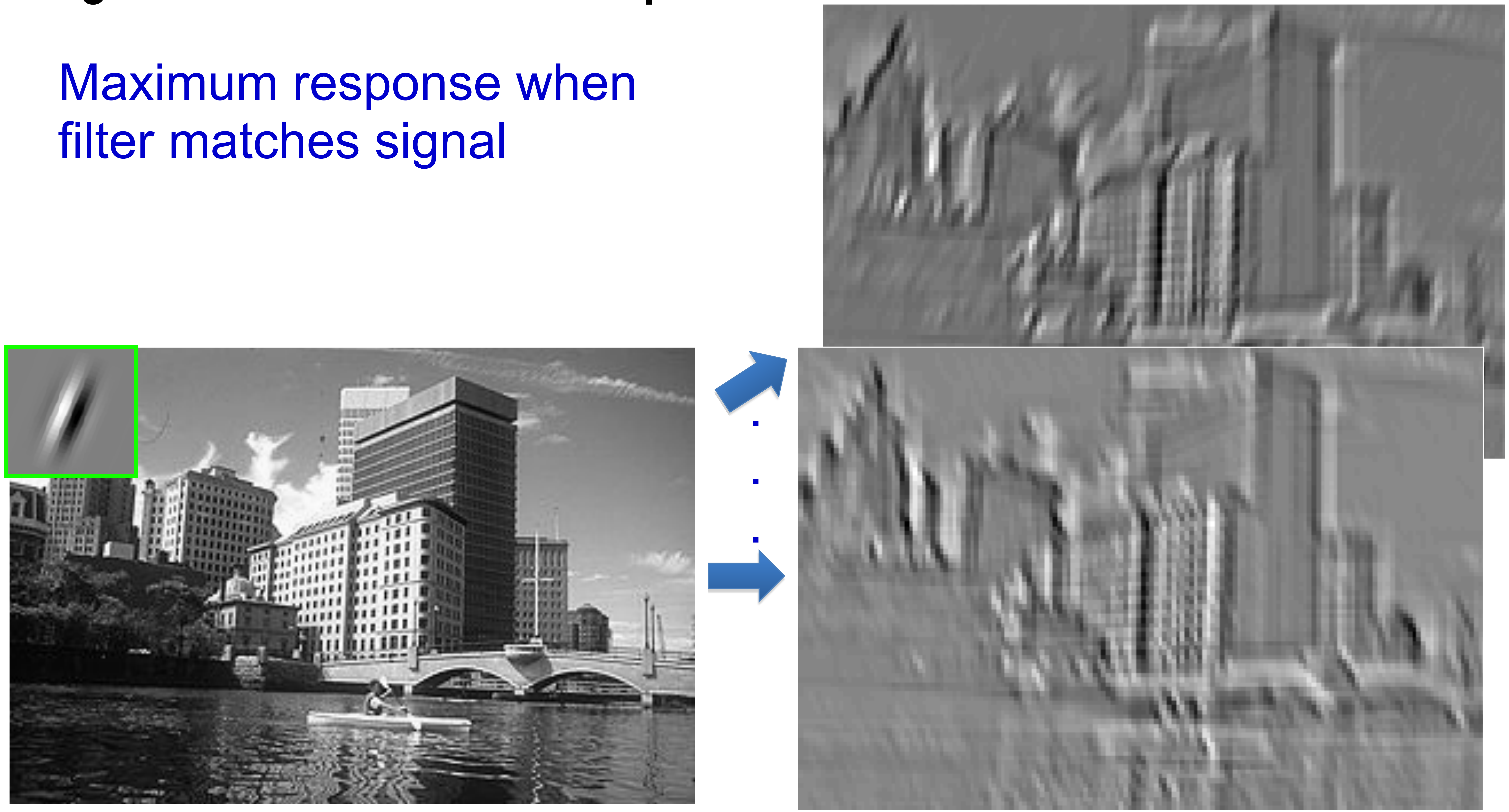
- A bank of 256 filters (learned from data)
- Each filter has 1 channel (it applies to a grayscale image)
- Each filter is 16x16 pixels



Slide: A. Zisserman

Filtering

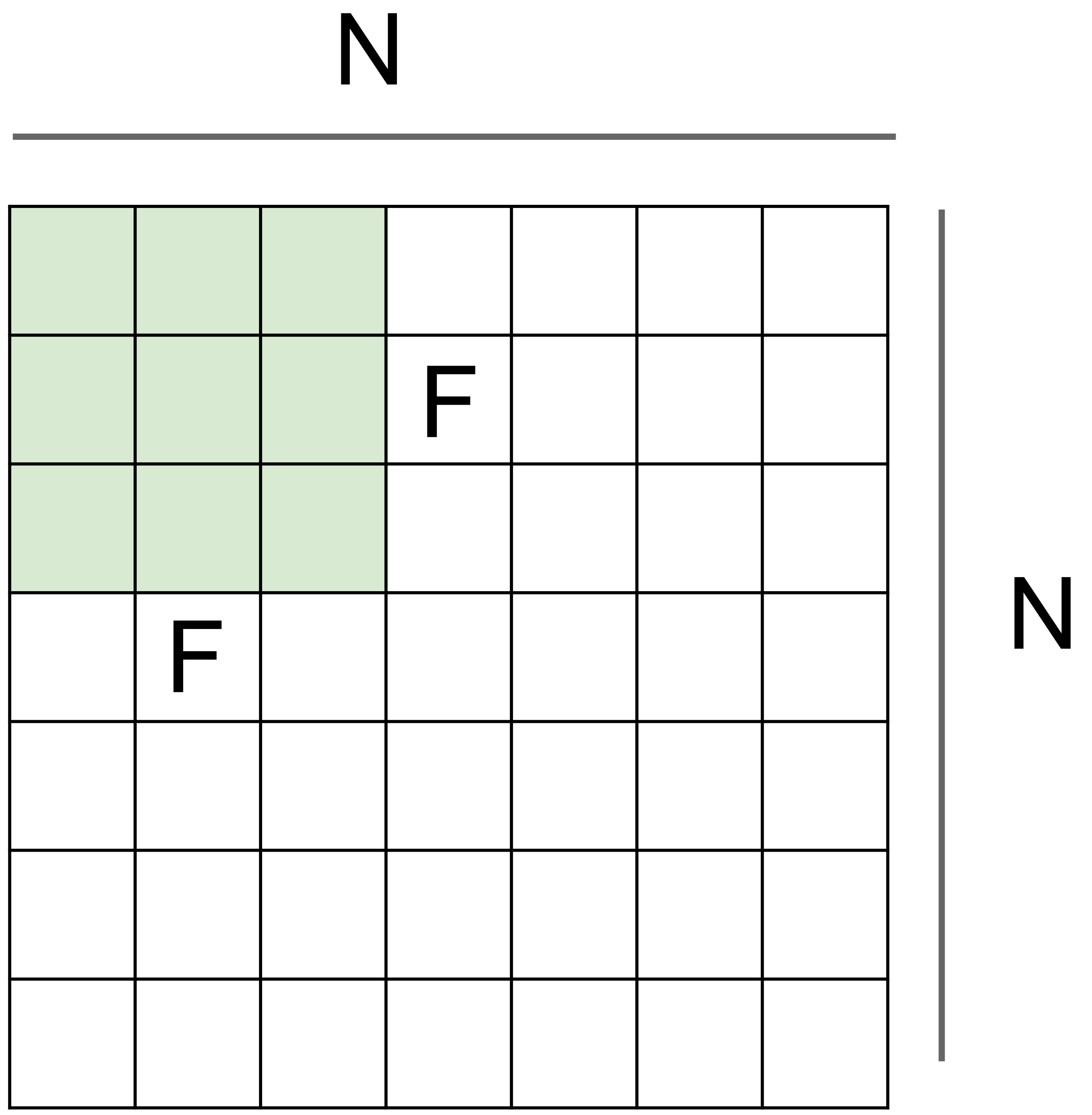
Each filter generates a "feature map"



Slide: R. Fergus / S. Lazebnik

Convolution details

What is the output size?

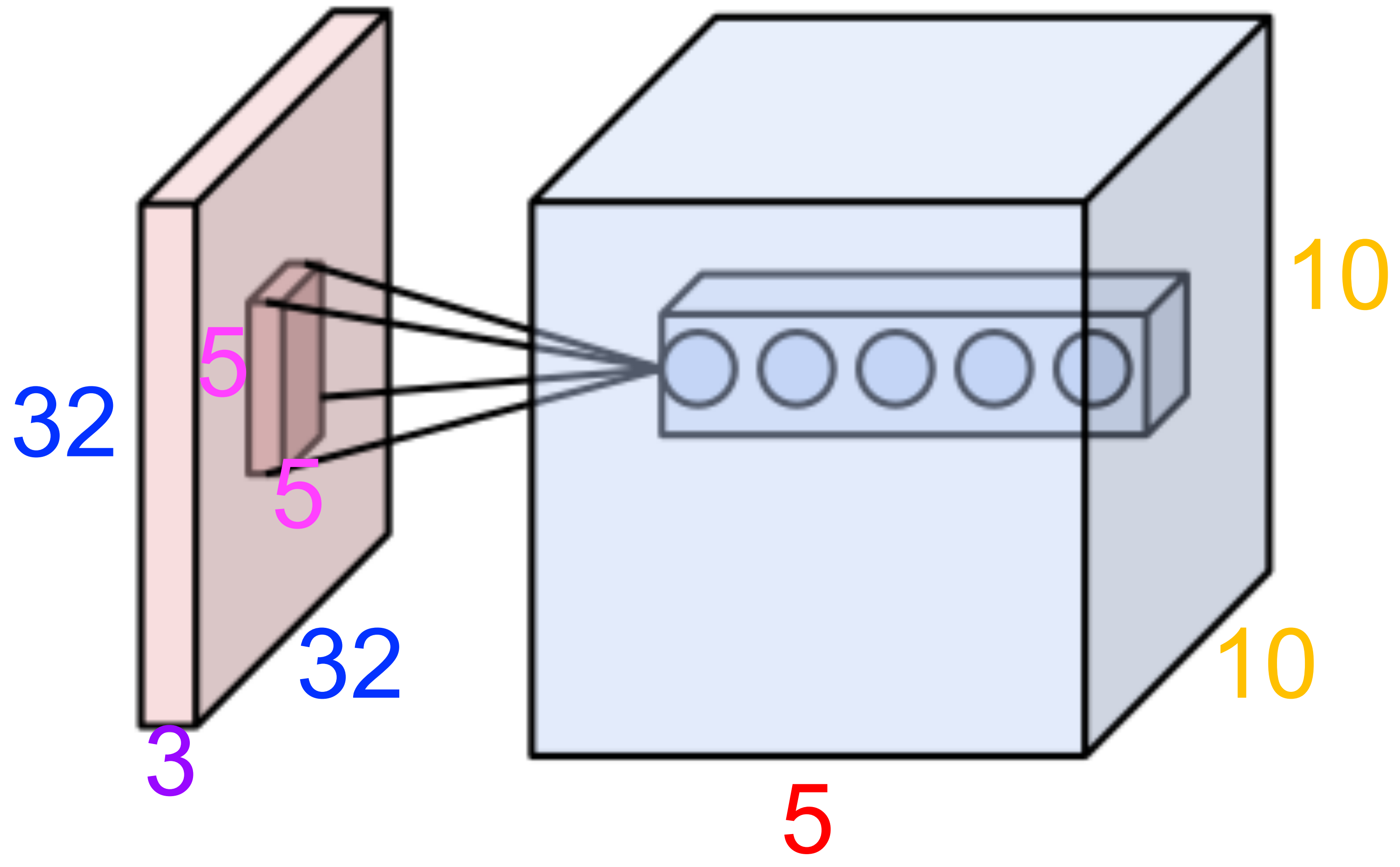


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = … :\

Slide: A. Karpathy / L. Fei Fei

Example: What is the output volume?

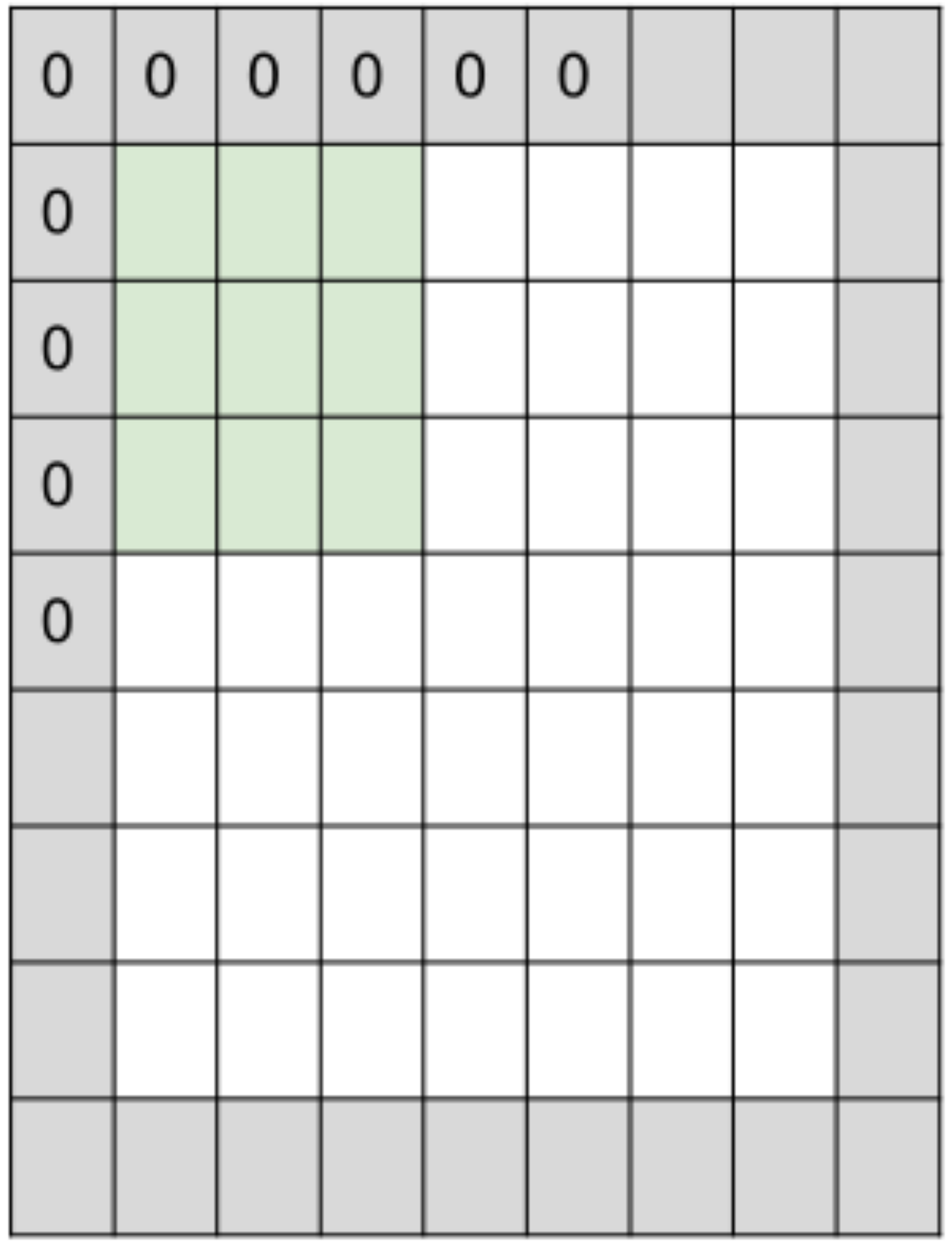


Input volume: 32x32x3
Receptive fields: 5x5, stride 3
Number of neurons: 5

Output volume: (32 - 5) / 3 + 1 = 10, so: 10x10x5

Slide: A. Karpathy / L. Fei Fei

Zero padding (in each channel)



e.g. input 7x7
neuron with receptive field 3x3, stride 1
pad with 1 pixel border => what is the output?

7x7 => preserved size!

in general, common to see stride 1, size F, and zero-padding with (F-1)/2.
(Will preserve input size spatially)

Slide: A. Karpathy / L. Fei Fei

What is the number of parameters?

- Consider an input gray-scale image of 1000x1000 pixels.
- What is the number of parameters of a filter bank of 100 7x7 filters?
- How does it compare to a fully connected layer that considers the entire input image?

Convolution:
100x 7x7
= 4900 parameters

vs.

Fully connected layer:
1000x1000
x
1000x1000
= 1B parameters.

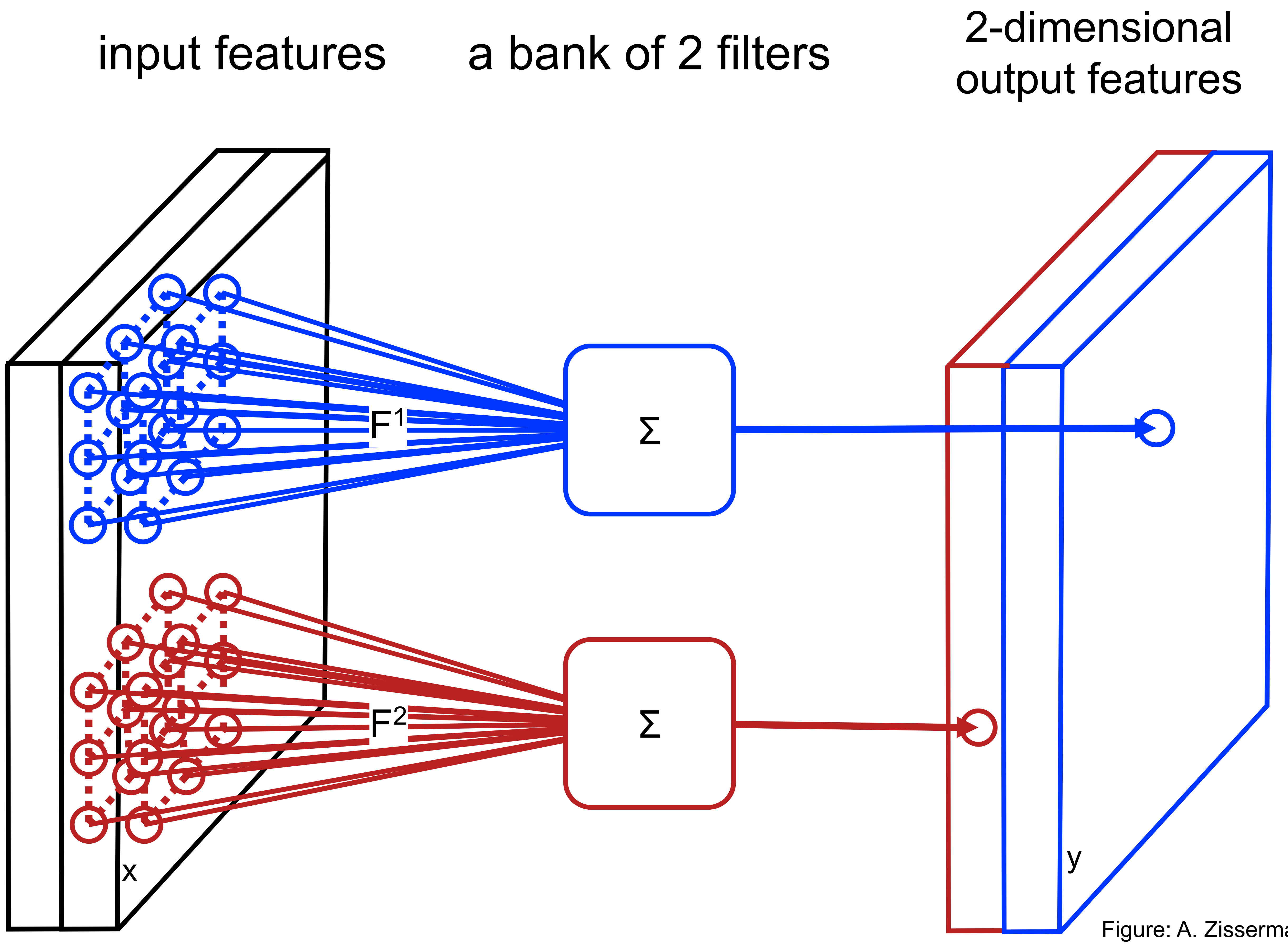
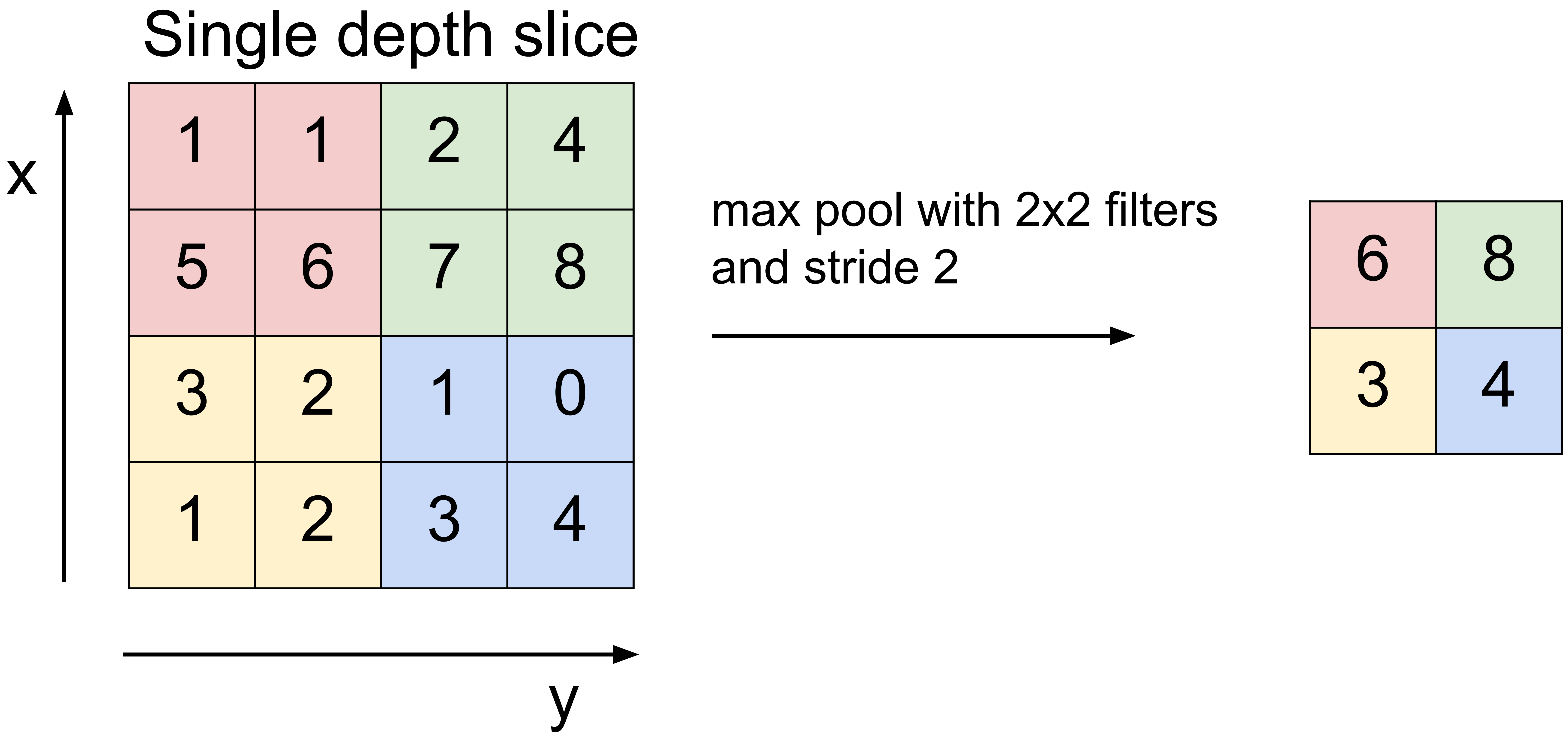


Figure: A. Zisserman

3. Spatial Max Pooling



Slide credit: Andrej Karpathy & Fei-Fei Li

3. Spatial Max Pooling

Dimensions of pooling outputs

Input volume of size [W1 x H1 x D1]
Pooling unit receptive fields F x F and applying them at strides of S gives

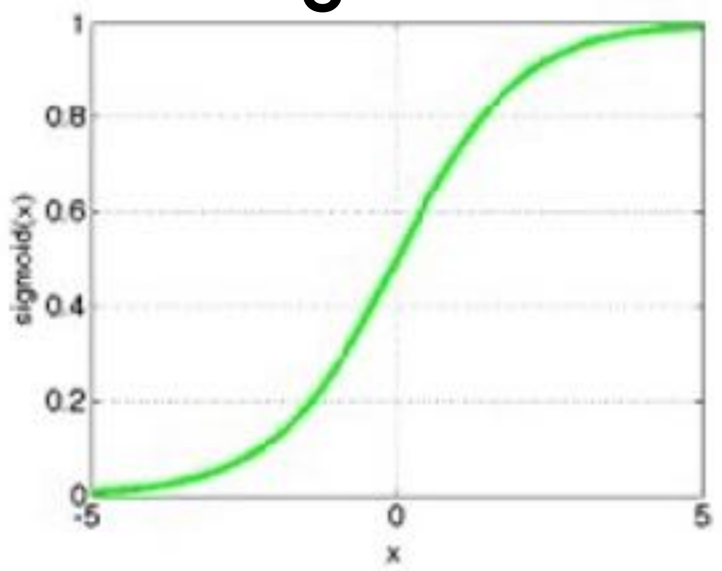
Output volume: [W2, H2, D1]
W2 = (W1-F)/S+1, H2 = (H1-F)/S+1

Note: pooling happens independently in each channel/slice

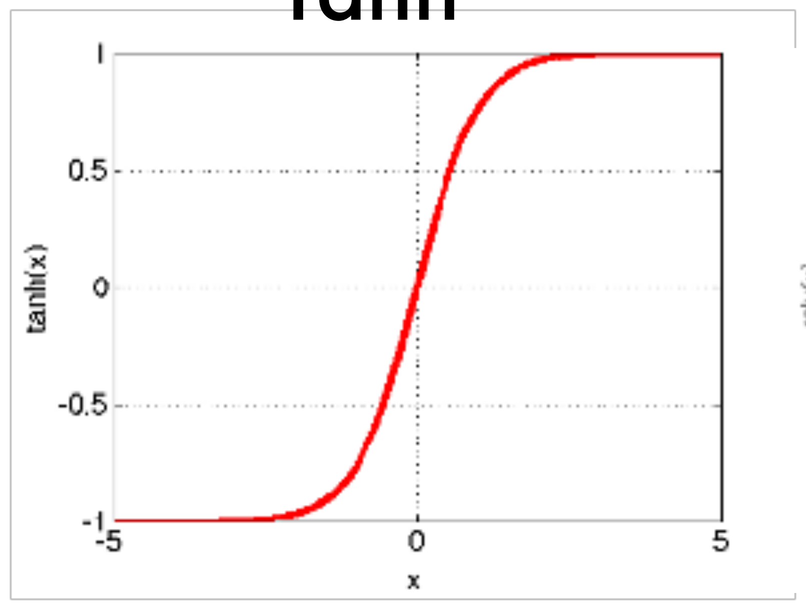
Slide: A. Karpathy / L. Fei Fei

4. Non-linearity

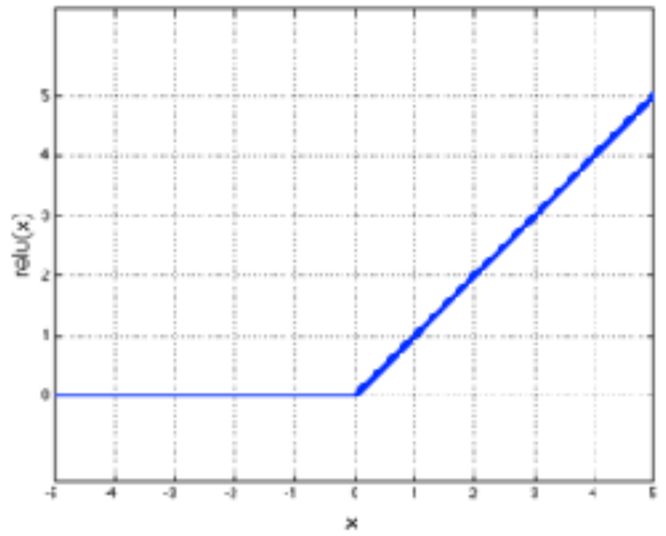
Sigmoid



Tanh



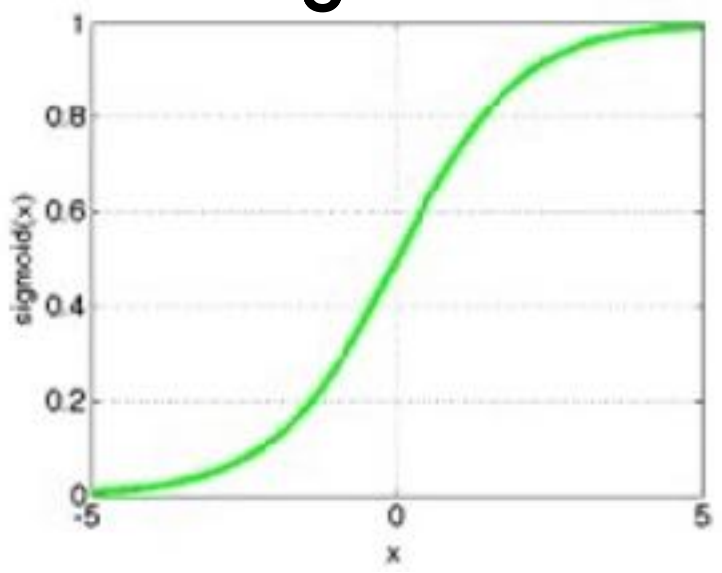
ReLU



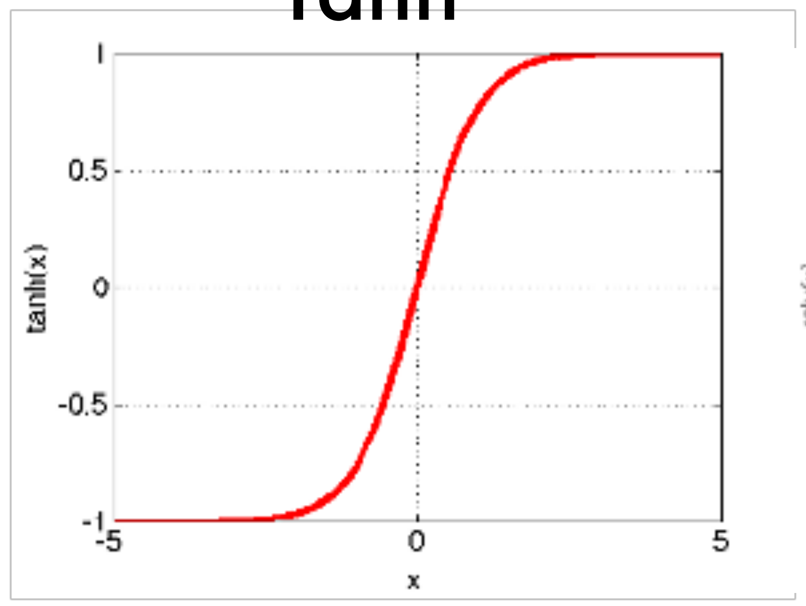
- The non-linear activation functions are essential. Why?

4. Non-linearity
Why?

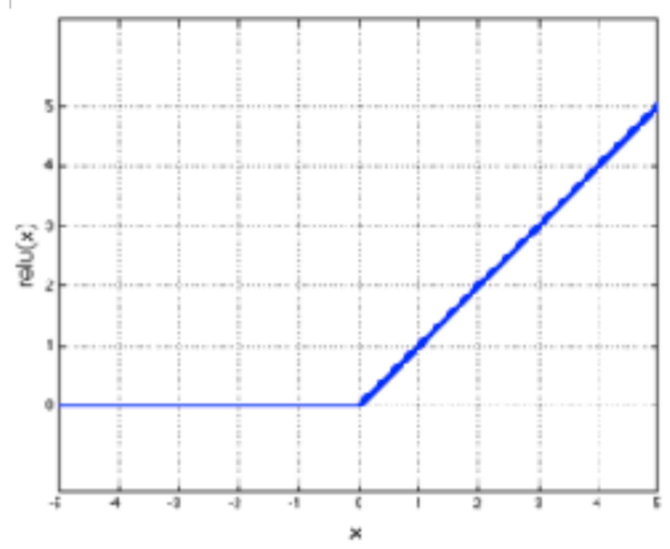
Sigmoid



Tanh



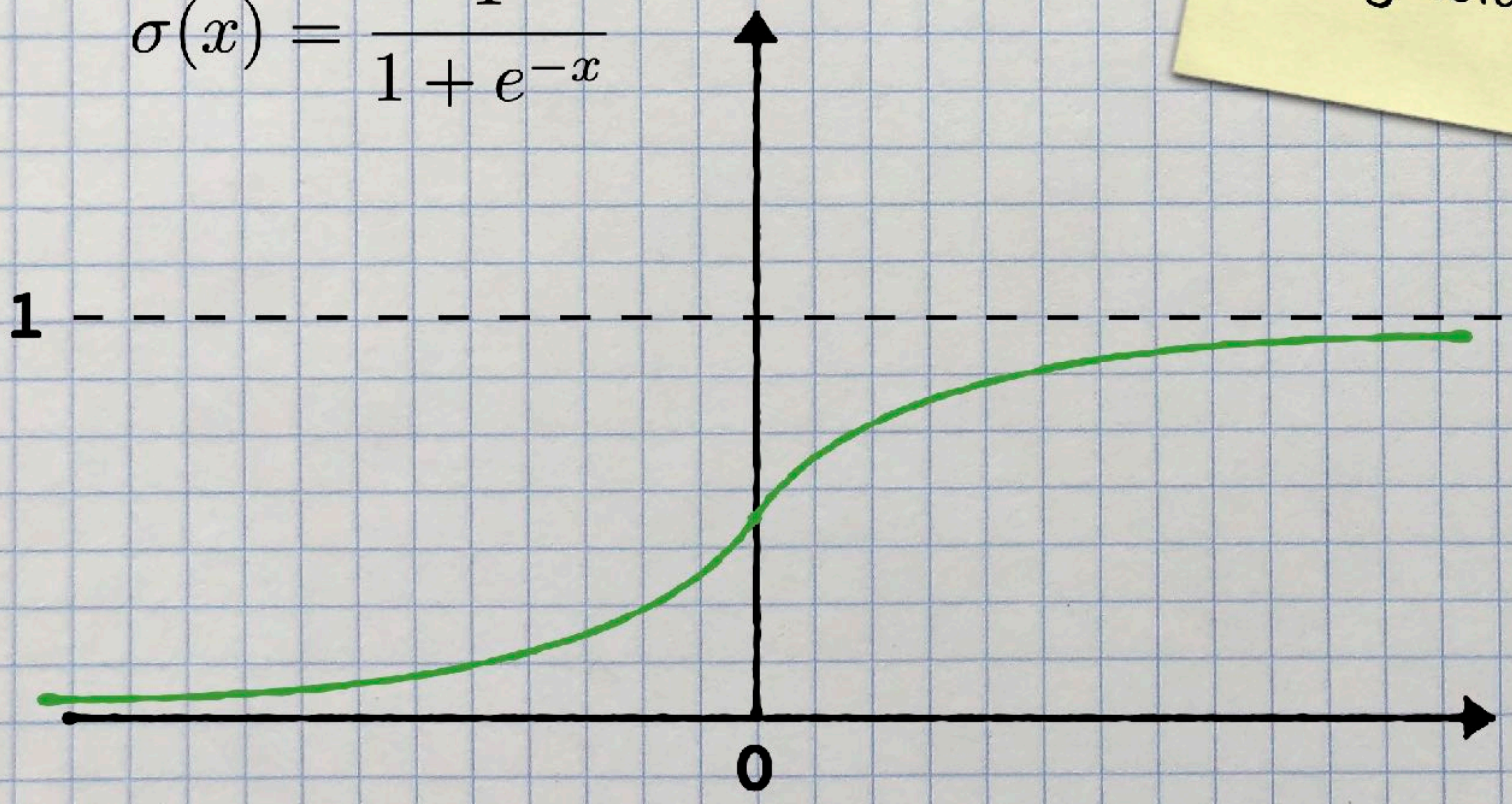
ReLU

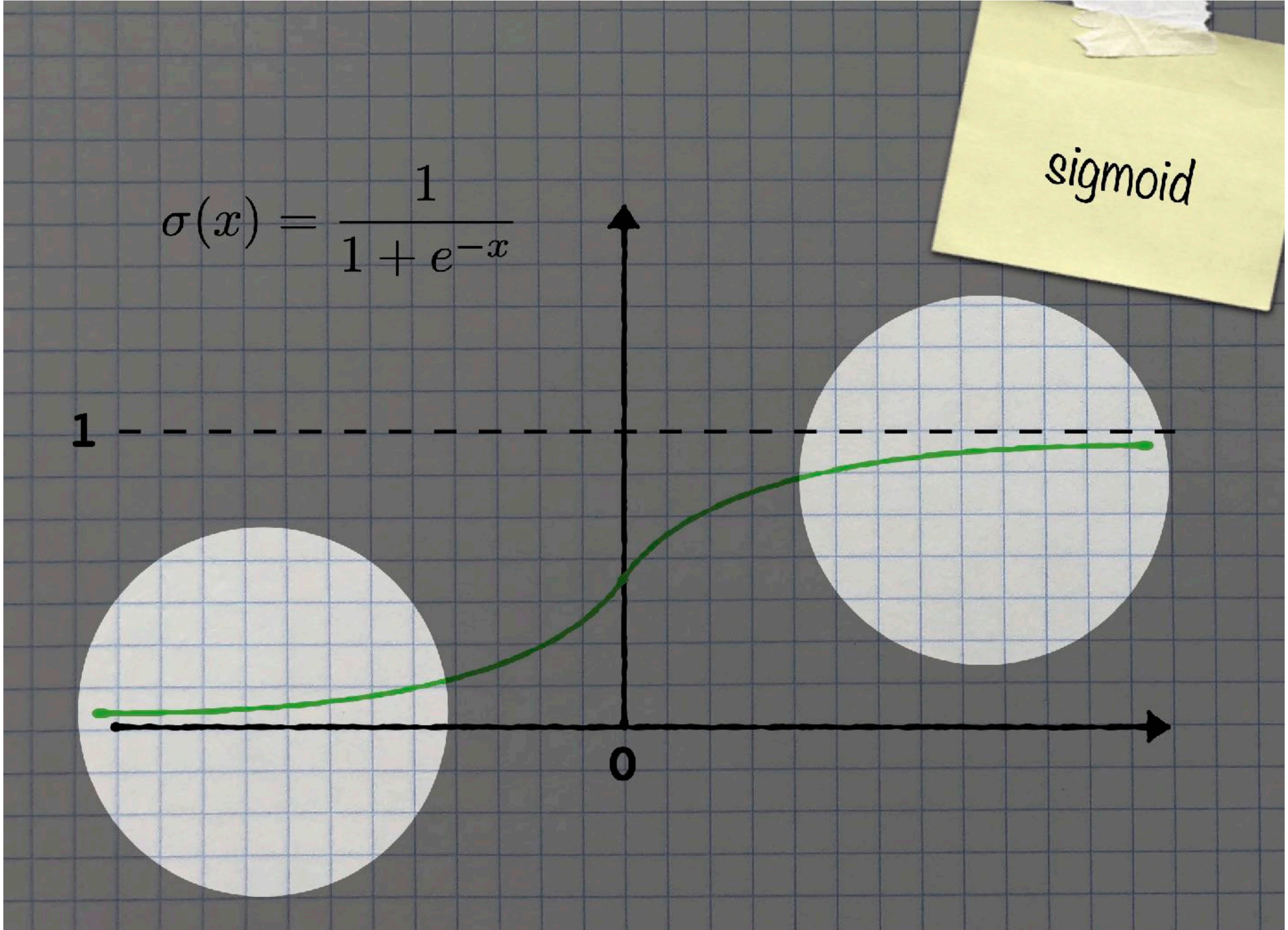


- Non-linearities allow us to approximate arbitrarily **complex functions**.
- **Universal approximation theorem**: A two-layer multilayer perceptron (MLP) with increasing continuous and bounded non-linearity can approximate any continuous function on a compact given <u>enough</u> hidden neurons. [Cybenko 1989]
- Linear activation functions produce **linear decisions no matter what the model size**, i.e., stacking multiple linear functions can be expressed with a single linear function.

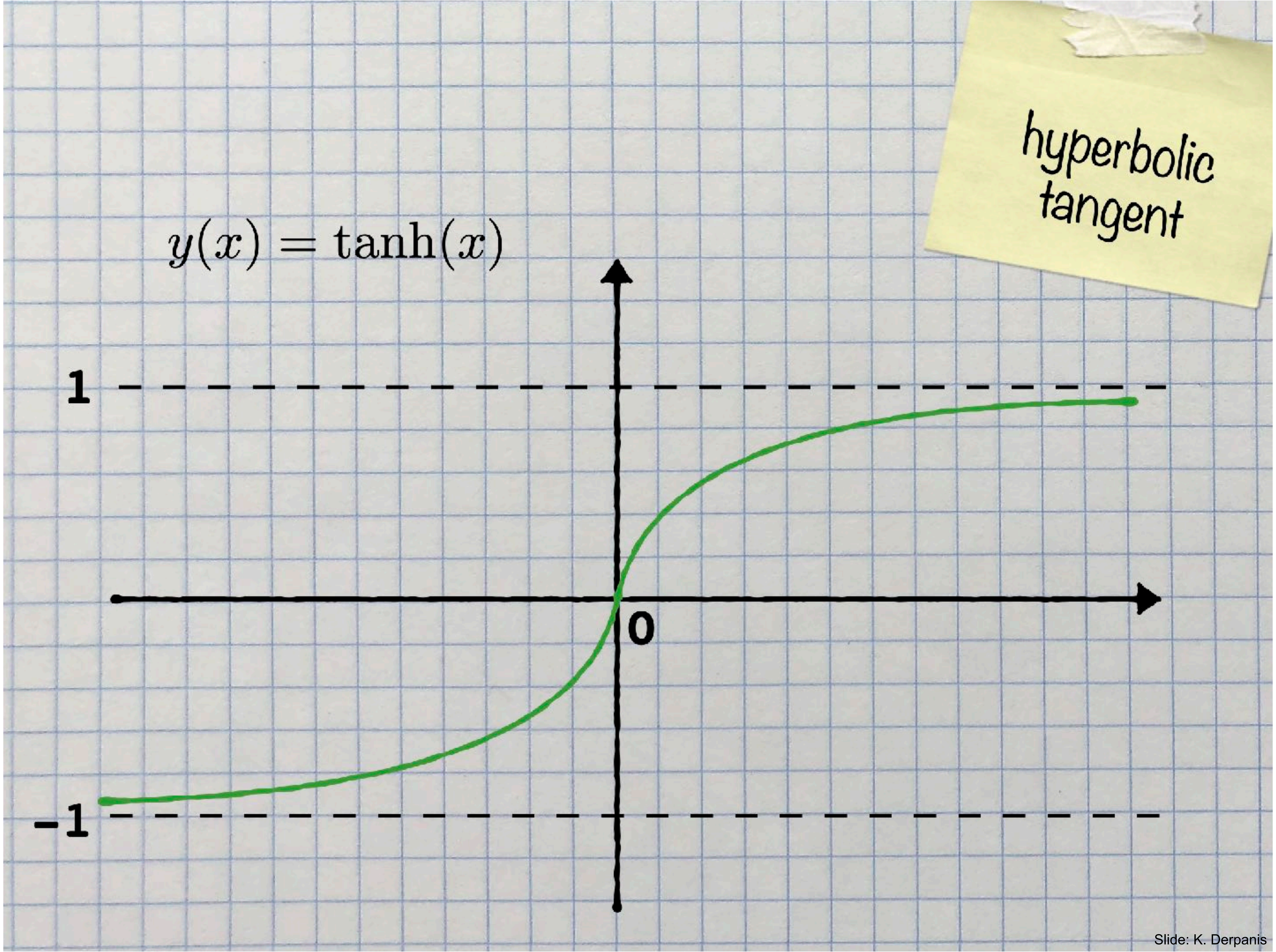
sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

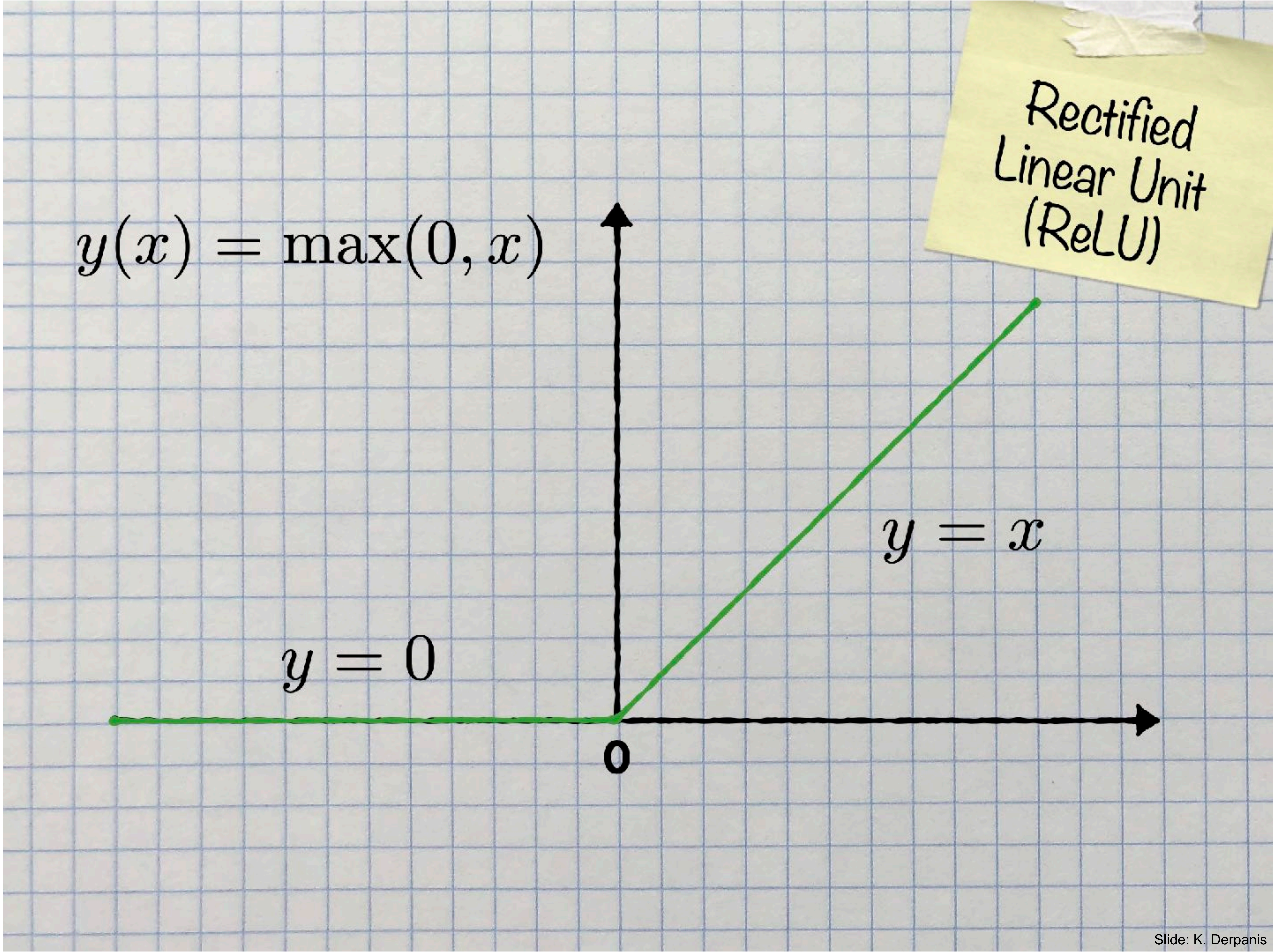




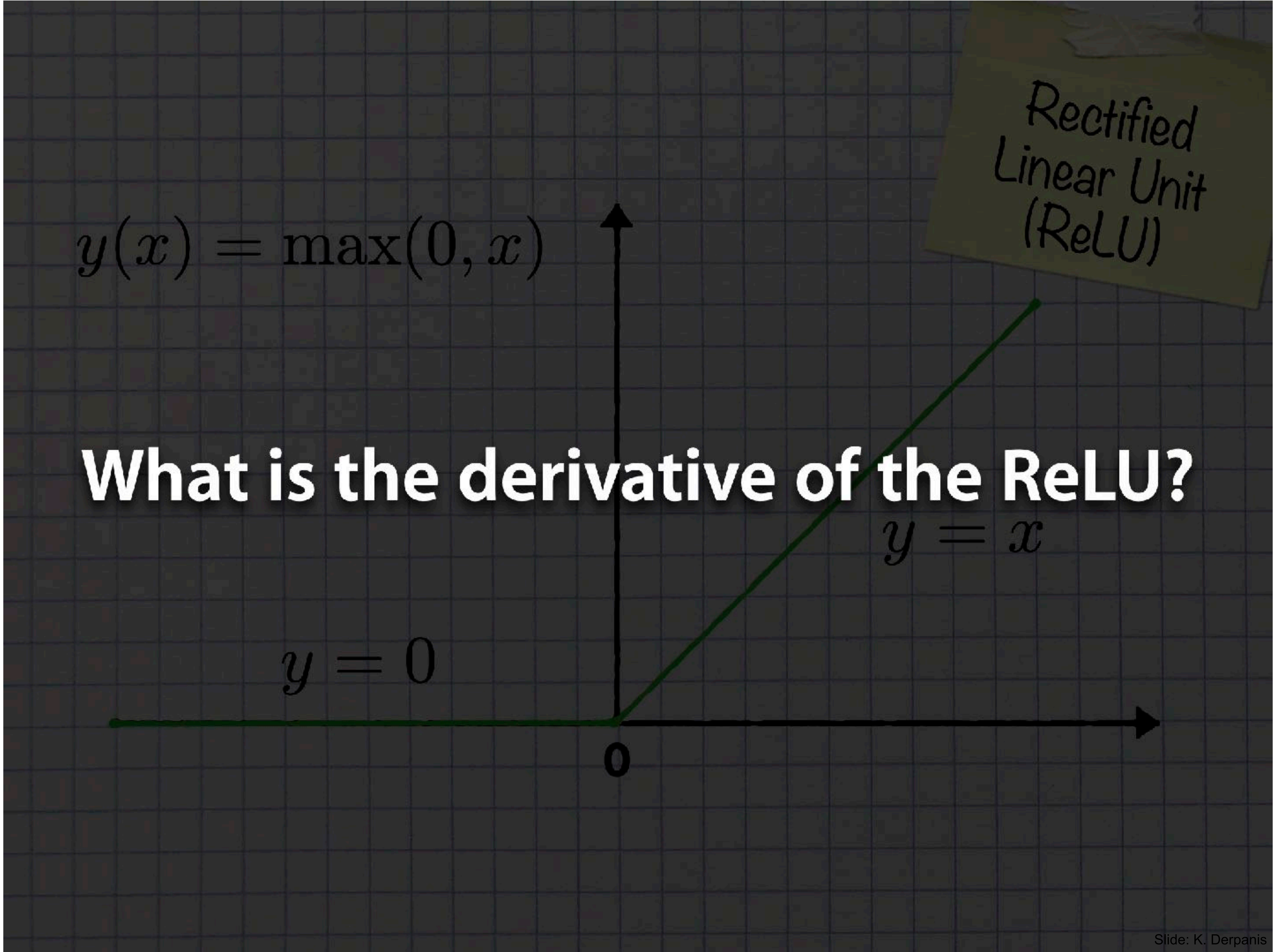
Slide: K. Derpanis

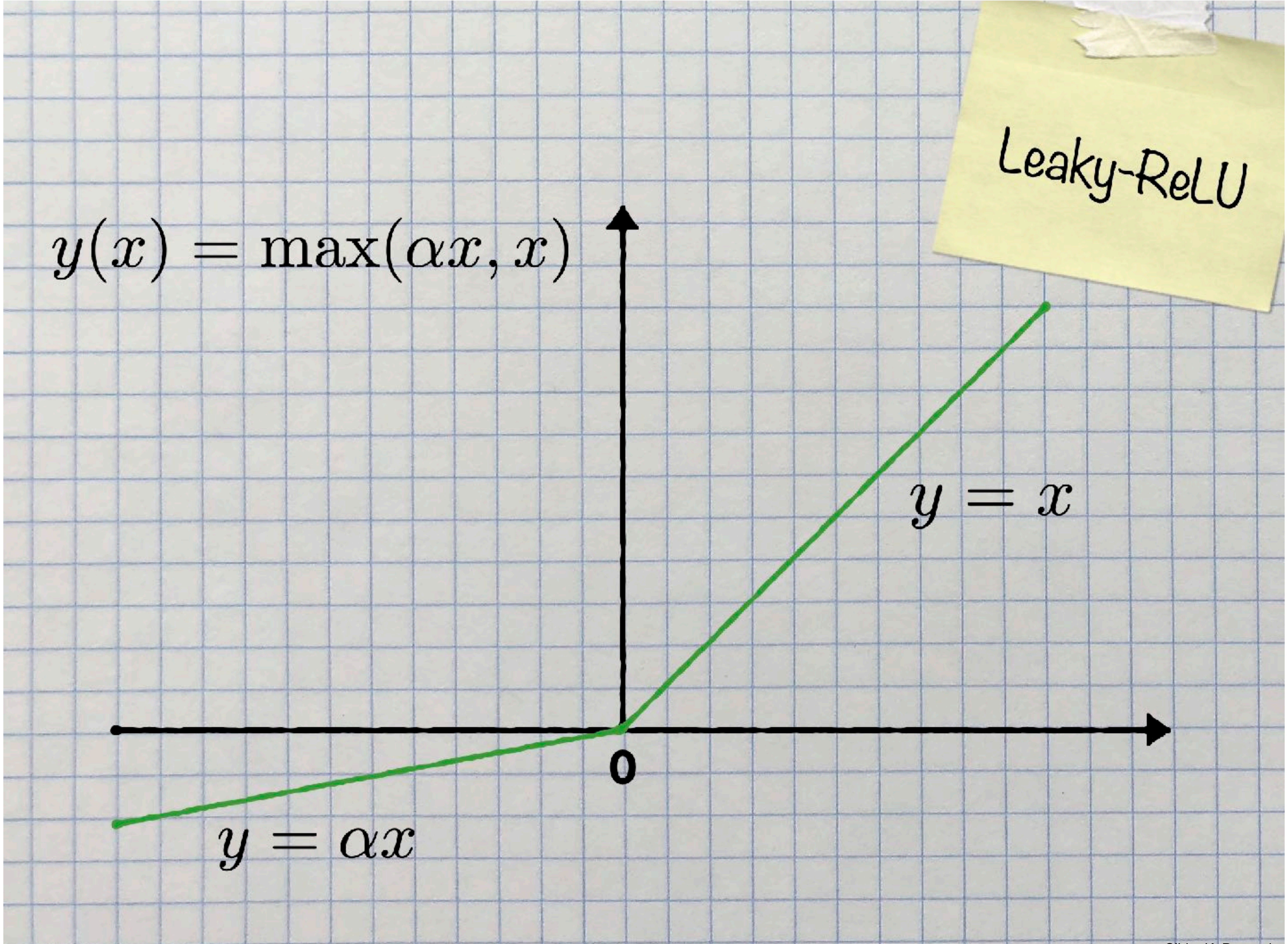


Slide: K. Derpanis



Slide: K. Derpanis


Rectified
Linear Unit
(ReLU)
$y(x) = \max(0, x)$
What is the derivative of the ReLU?
$y = x$
$y = 0$
0
Slide: K. Derpanis

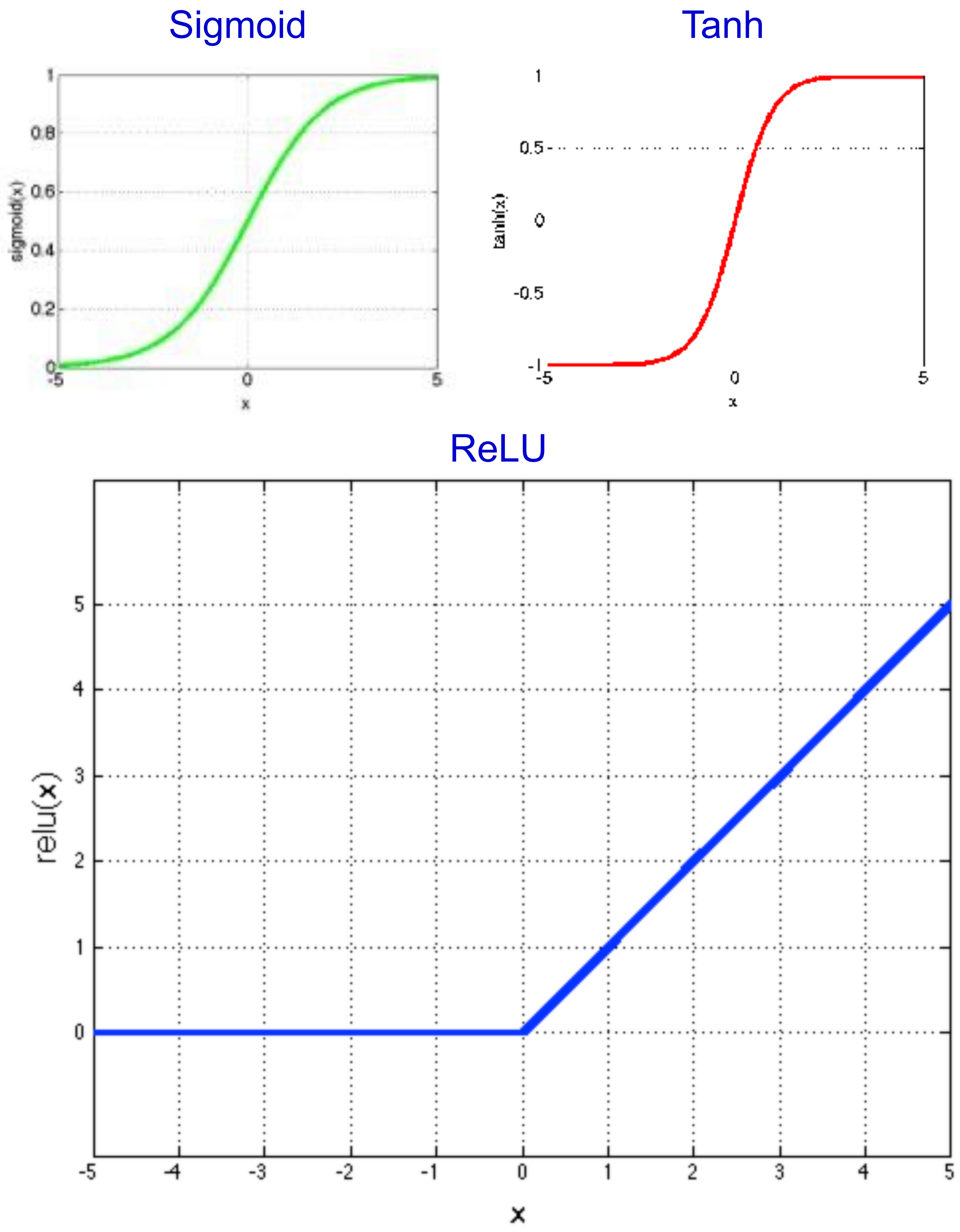


Slide: K. Derpanis

4. Non-Linearity

- Per-element (independent)

- Options:
  - Sigmoid: 1/(1+exp(-x))
  - Tanh
  - Rectified linear unit (ReLU)
    - Simplifies backpropagation
    - Makes learning faster
    - Avoids saturation issues
  - Variants of ReLU, e.g. Leaky ReLU



Slide: R. Fergus / S. Lazebnik

5. Normalization

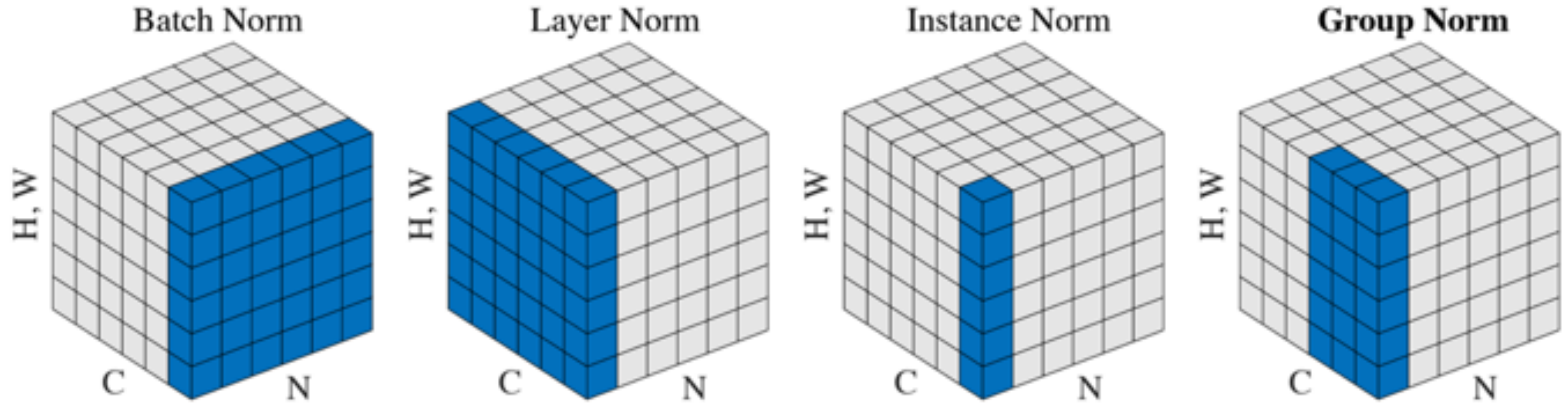


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

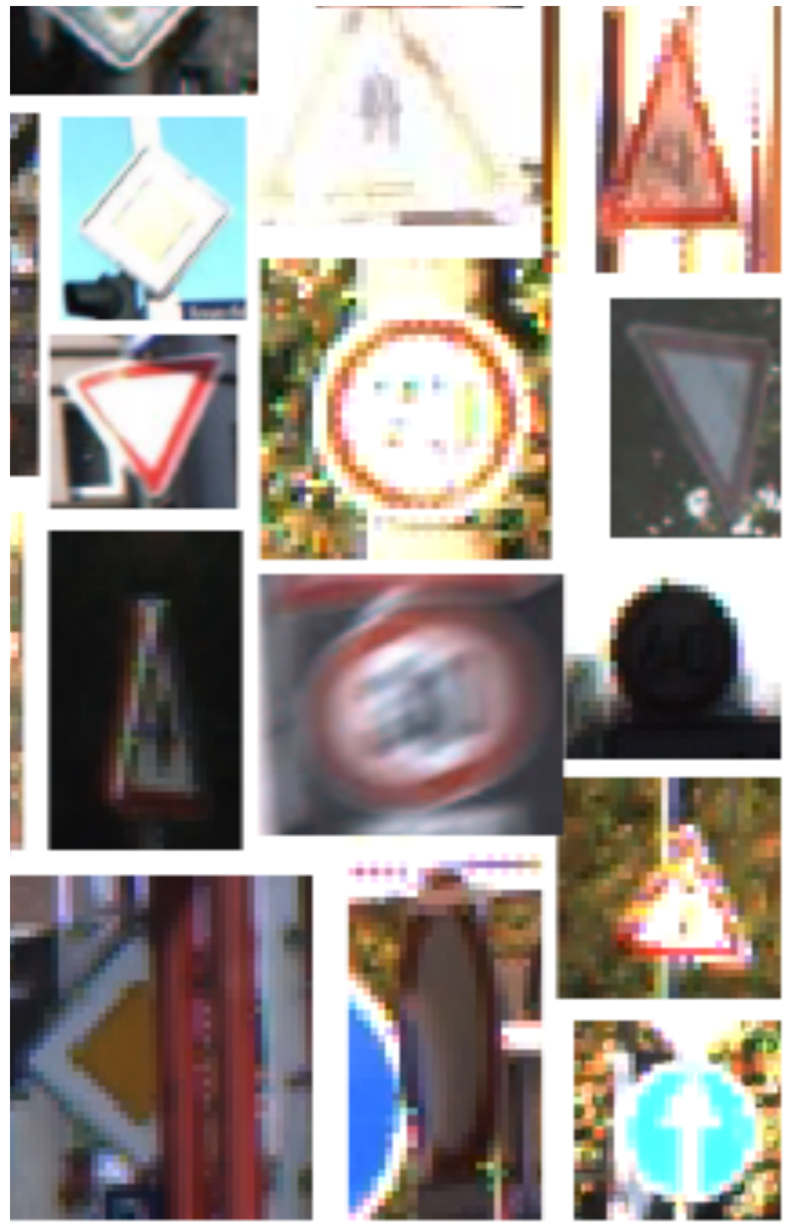
[Wu & He, "Group normalization", ECCV 2018]

CNN Successes

- Handwritten text/digits
  - MNIST (0.17% error [Ciresan et al. 2011])
  - Arabic & Chinese [Ciresan et al. 2012]

- Simpler recognition benchmarks
  - CIFAR-10 (9.3% error [Wan et al. 2013])
  - Traffic sign recognition (0.56% error vs 1.16% for humans [Ciresan et al. 2011])

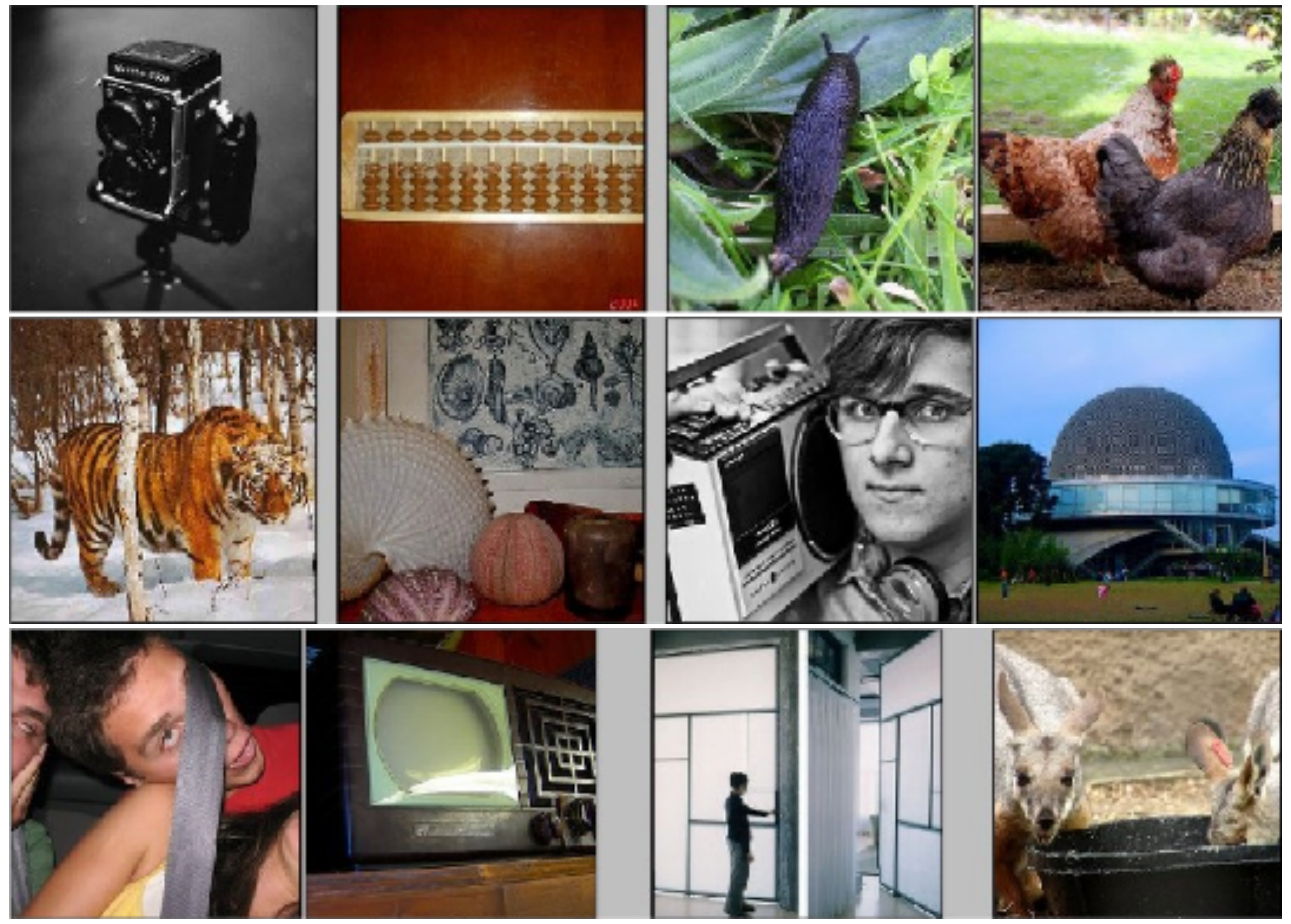
- But until recently, less good at more complex datasets
  - Caltech-101/256 (few training examples)



Slide: R. Fergus / S. Lazebnik

ImageNet Dataset

IMAGENET



[Deng et al. CVPR 2009]

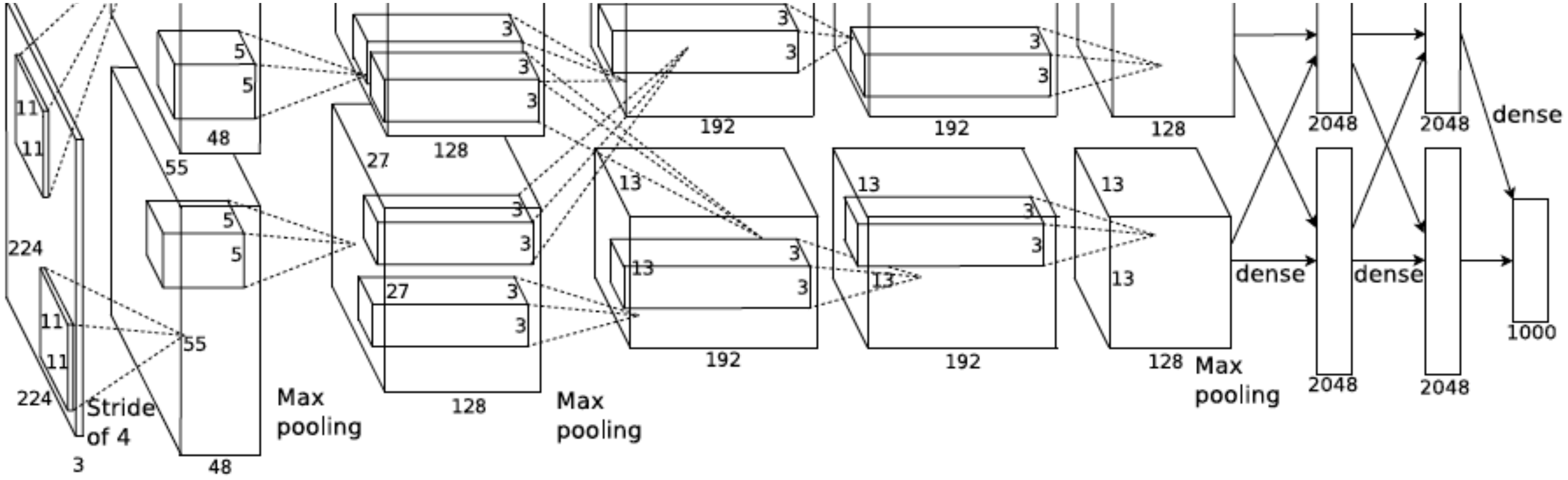
- ~14 million labeled images, 20k classes
- Challenge: 1.2 million training images, 1000 classes
- Images gathered from Internet
- Human labels via Amazon Mechanical Turk

Slide: R. Fergus / S. Lazebnik

ImageNet Challenge 2012 (ILSVRC)

- Similar framework to LeCun'98 but:
  - **Bigger model** (7 hidden layers, 60,000,000 params)
  - **More data** (10^6 vs. 10^3 images)
  - **GPU** implementation (50x speedup over CPU)
    - Trained on two GPUs for a week
  - Better regularization for training (DropOut)

AlexNet



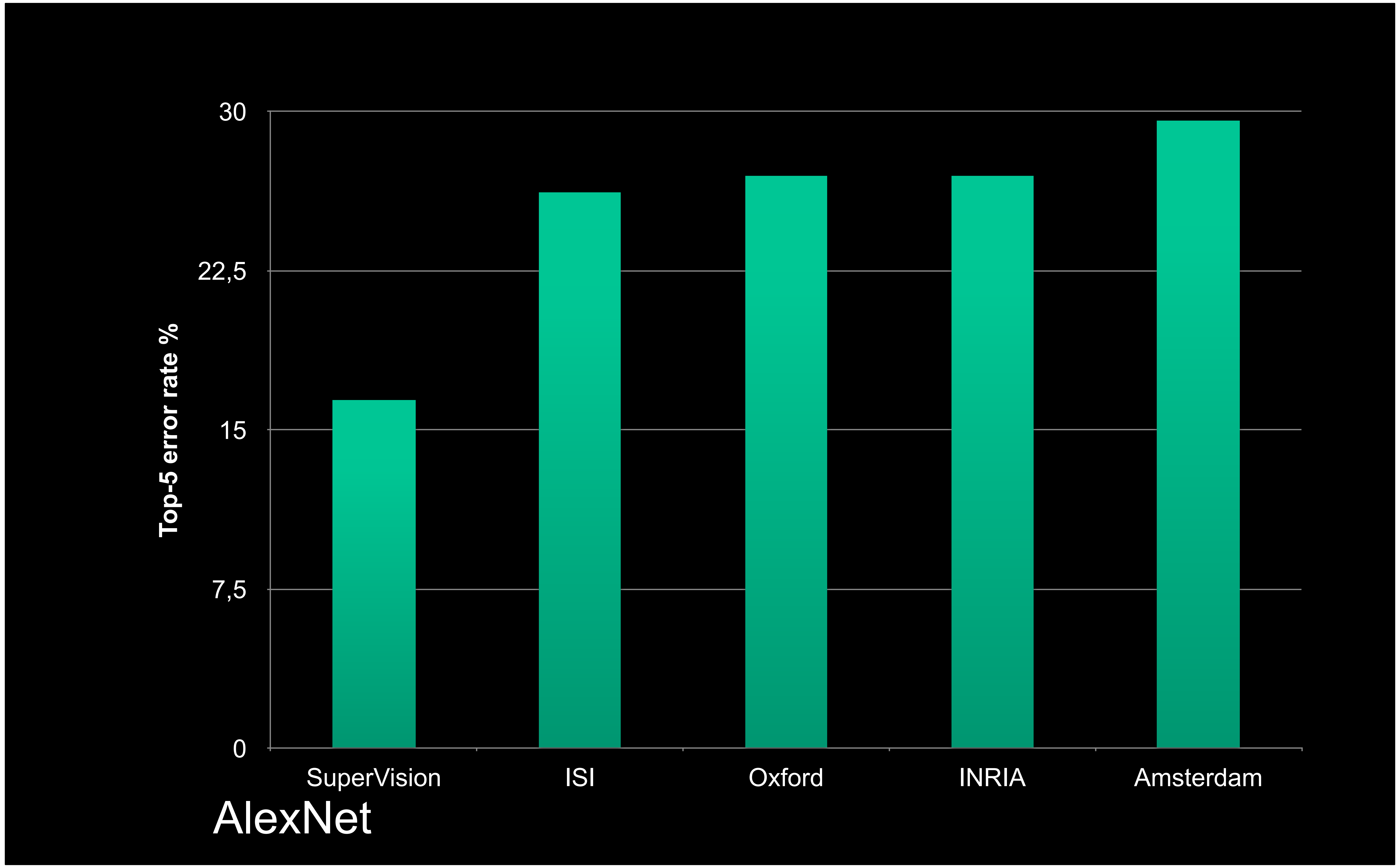
Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Slide: R. Fergus / S. Lazebnik

ImageNet Challenge 2012 (ILSVRC)

AlexNet – **16.4% error** (top-5)

Next best (non-convnet) – **26.2% error**



Slide: R. Fergus / S. Lazebnik

Agenda

- **1. Recap: Bag of Visual Words, Analogy with NNs**
- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs
- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures
- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers
- **5. Beyond classification**

Gradient descent

- The objective function is an average over all N training data points:

$$L(\theta) = \frac{1}{N}\sum_i l(\theta,\ X_i,\ Y_i)$$

- Performing a gradient descent is iterating.

$$\theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N}\sum_i \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta}$$

- Need to choose the learning rate policy α_t
- If the function is not convex, get stuck in a local minimum
- Each step can be expensive to compute if the dataset is large

Convex case

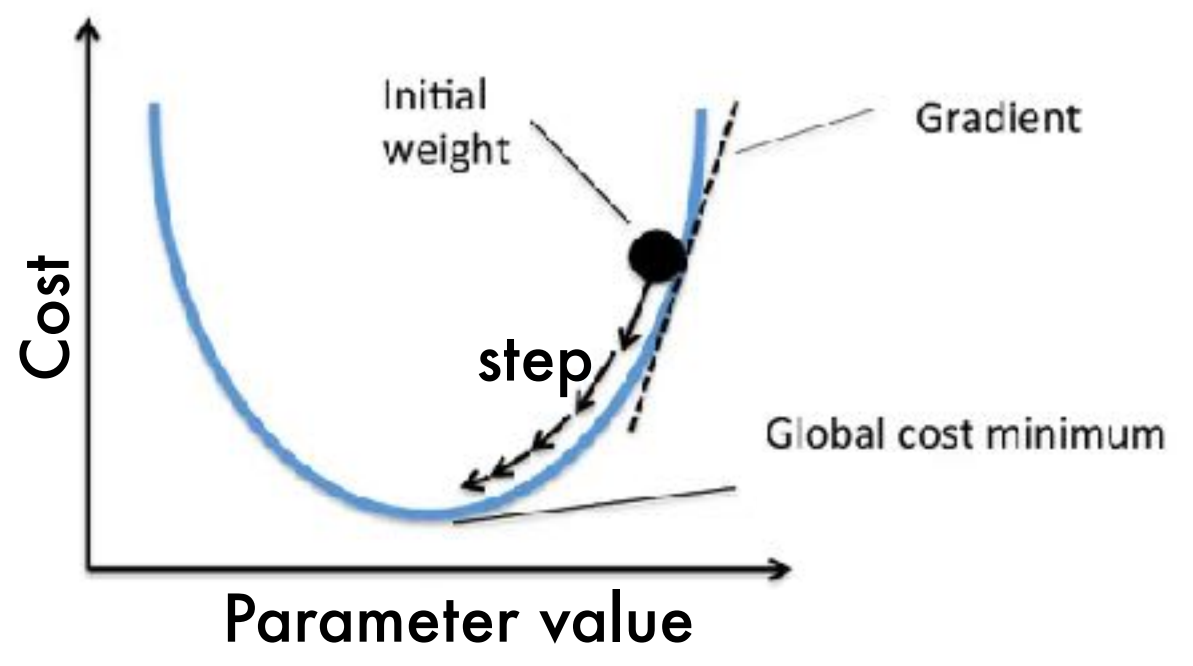


Image source

Non-convex

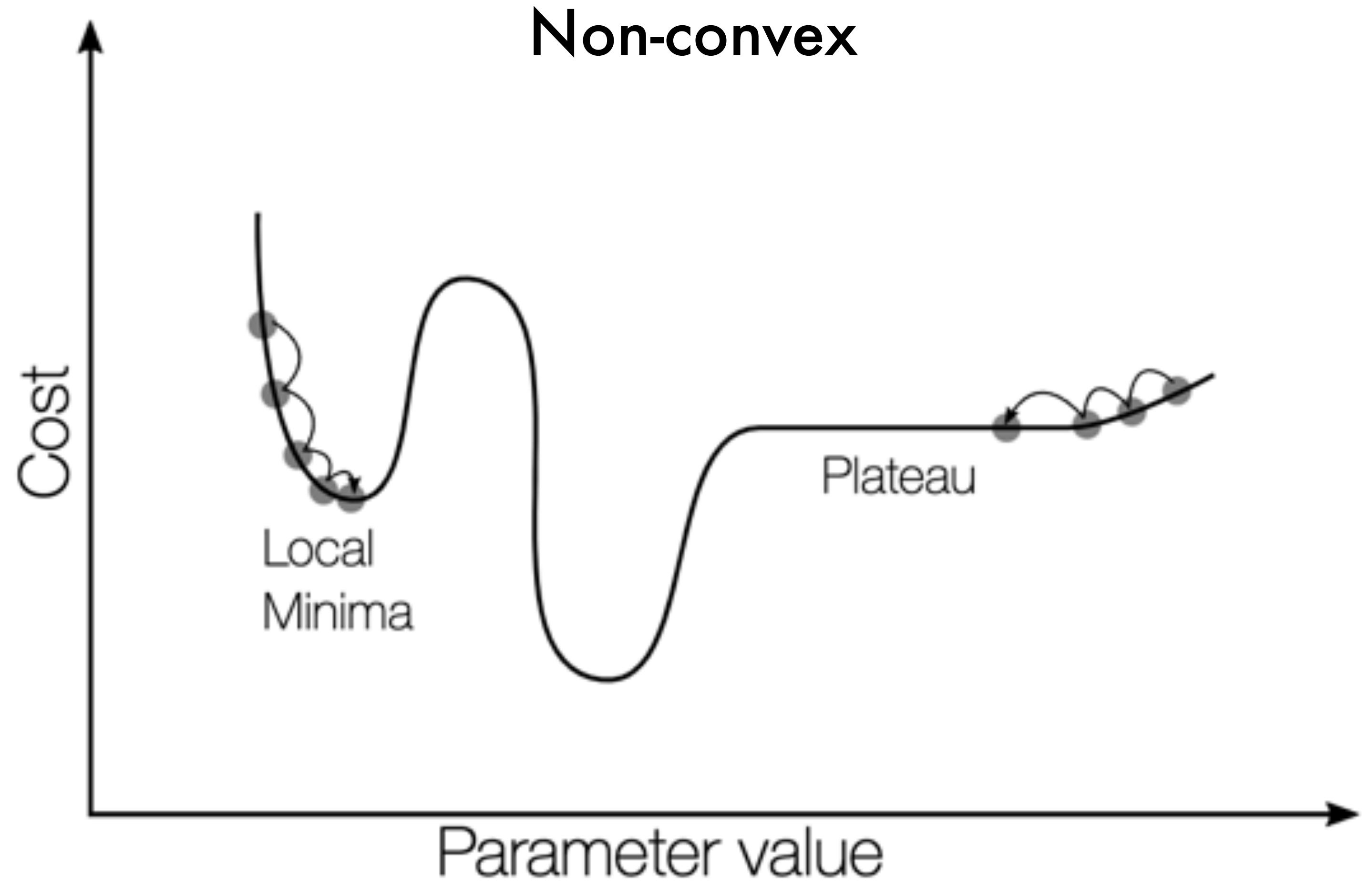


Image source

Stochastic gradient descent

- Instead of computing the gradient, compute an approximation:

$$\theta_{t+1} \to \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta}$$

$$\Downarrow$$

$$\theta_{t+1} \to \theta_t - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta}$$

- Can take advantage of large datasets, in particular infinite* datasets!

- Introduce stochasticity, which might be good to get out of local minima in the non-convex case

Stochastic gradient descent with minibatch

- Some variance is good, too much can be bad

$$\theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta}$$

$$\Downarrow$$

$$\theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{K} \sum_{i=1}^{K} \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta}$$

(with $K << N$)

- It's faster to compute several gradients in parallel

- In practice, using batches as large as possible so that the network fits in the GPU memory (e.g., between 1 and 256, depending on the task and network)

Summary: Stochastic Gradient Descent (SGD)

The objective function is an average over all *N* training data points:

$$\theta_{t+1} \to \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta}$$

(gradient descent)

Key idea: approximate the gradient with 1 random datapoint:

$$\theta_{t+1} \to \theta_t - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta}$$

(stochastic gradient descent)

Pick K random points instead of picking 1 (with $K << N$):

$$\theta_{t+1} \to \theta_t - \frac{\alpha_t}{K} \sum_{i=1}^{K} \frac{\partial \ell(\theta, X_i, Y_i)}{\partial \theta}$$

(stochastic gradient descent with mini-batches)

=> commonly used

Slide credit: Andrea Vedaldi

Quiz: 5 minutes

Let's consider a training dataset of N samples. How many iterations (i.e., parameter updates) are there in one training epoch?

a. Gradient descent: ___
b. Stochastic gradient descent: ___
c. Stochastic gradient descent with minibatch of size K: ___

a.

slido.com
#2871 232

b.

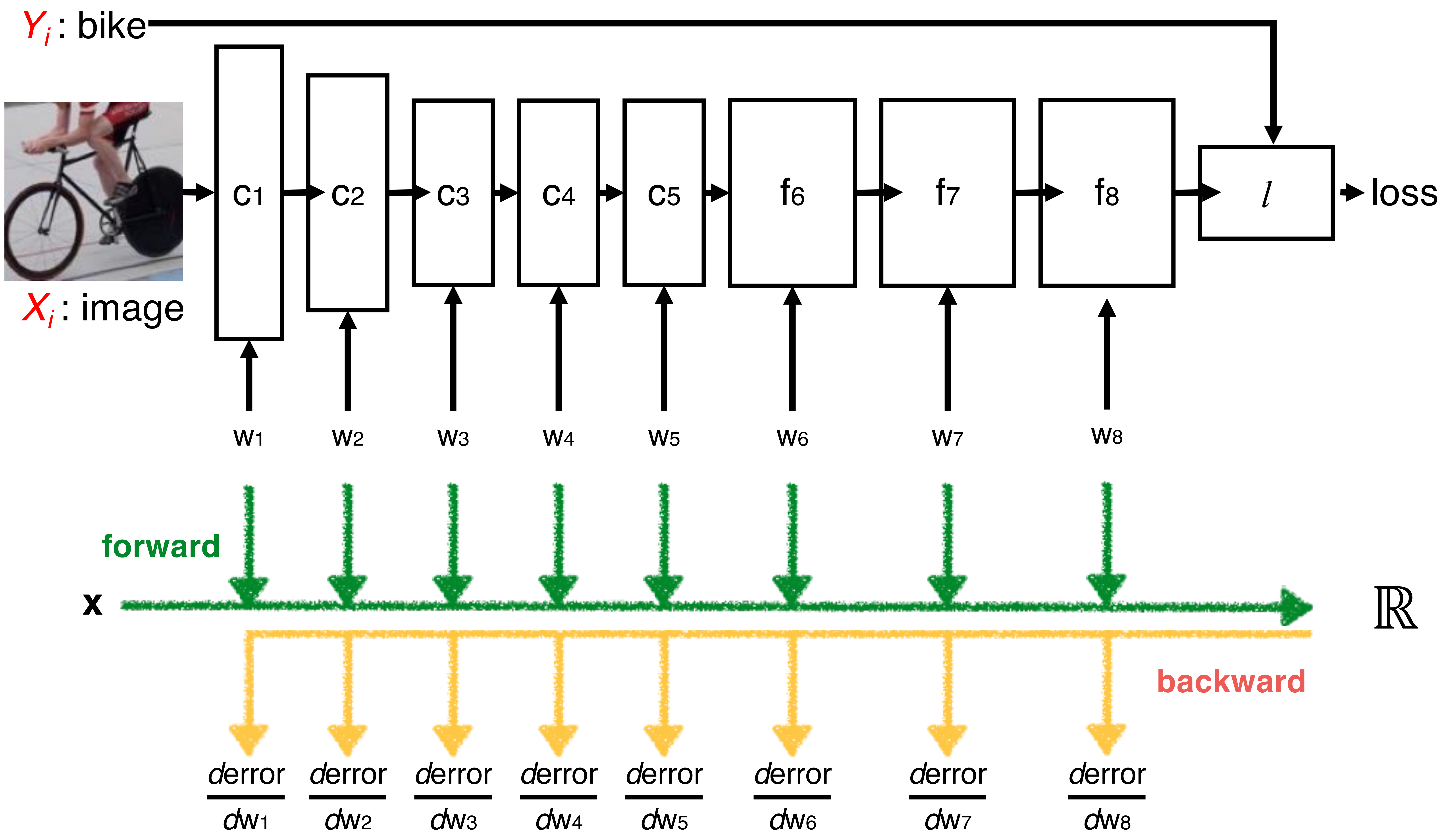
slido.com
#2871 233

c.

slido.com
#2549 484

Backpropagation

Computing the gradients: While in theory, we just have the gradients of composite functions and for that apply **chain rule**, there is an **efficient** way to do it, called **backpropagation**.



Training a neural network

Given a (X, Y) pair:

- **Forward pass**: apply network to X to produce an output $\hat{Y}$
- **Evaluation**: Compute loss function, i.e., $\ell(\hat{Y}, Y)$
- **Backward pass**: compute the gradient with backprogation
- **Update:** Take a step in the direction of the gradient

Slide: A. Joulin

Loss Function

- Regression:
  - L1 (absolute error) / L2 (squared error)
- Classification:
  - Cross-entropy loss

Loss Function: Regression

Estimating a continuous value

- L1 (absolute error)

$$L = \left| f(X_i, \theta) - Y_i \right|$$

- L2 (squared error)

$$L = \Big(\underbrace{f(X_i, \theta)}_{\text{Prediction}} - Y_i \Big)^2$$

Prediction:
output of
the network f
with parameters θ
given input X_i

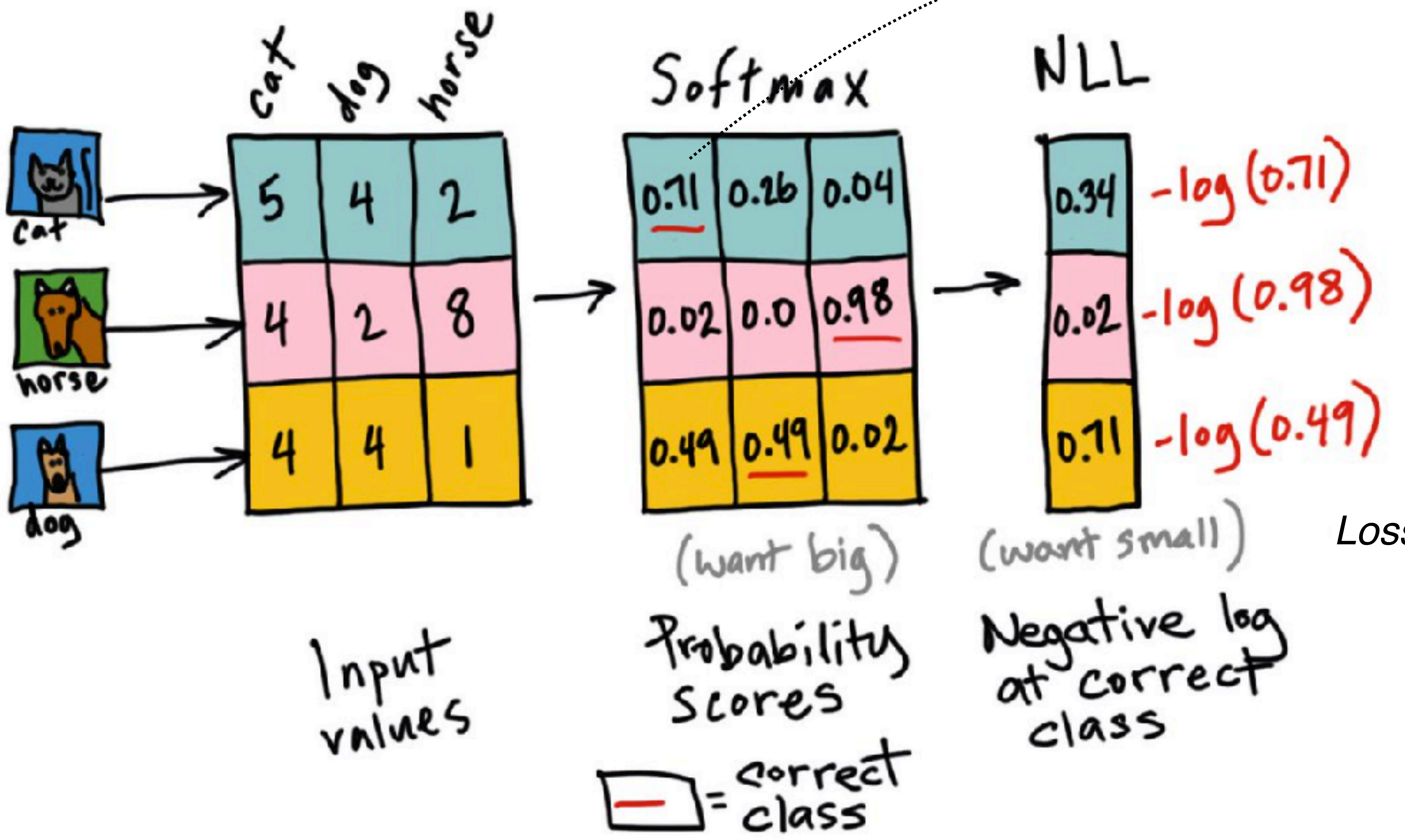
Ground truth:
(label, annotation)

Loss Function: Classification

- Cross-entropy loss = softmax + negative log-likelihood

$$\text{loss}(x, \text{class}) = -\log\left(\frac{\exp(x[\text{class}])}{\sum_j \exp(x[j])}\right)$$

$$\frac{e^5}{e^5 + e^4 + e^2}$$



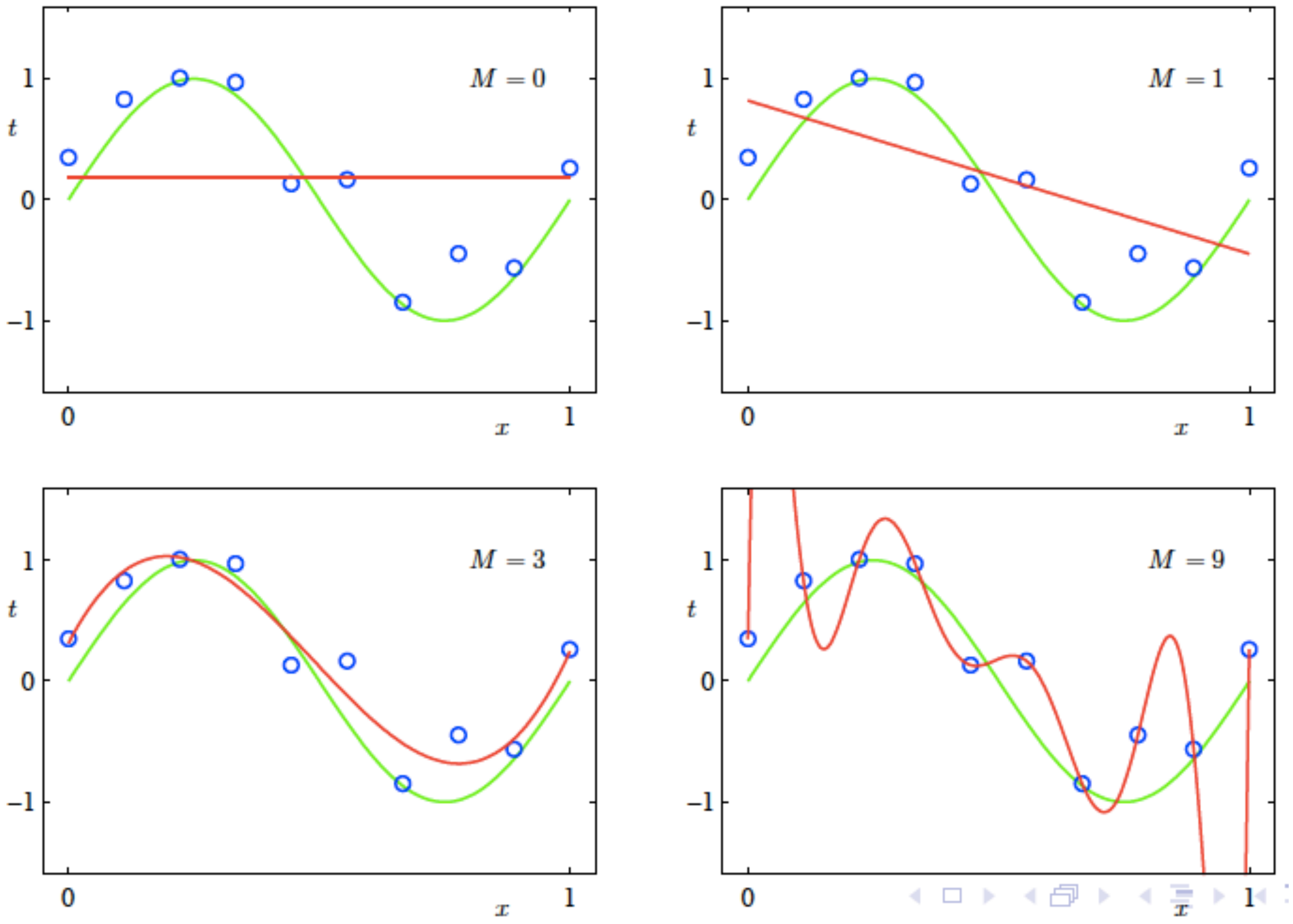
Loss = 0.34+0.02+0.71 = 1.07

Fig: Micheleen Harris

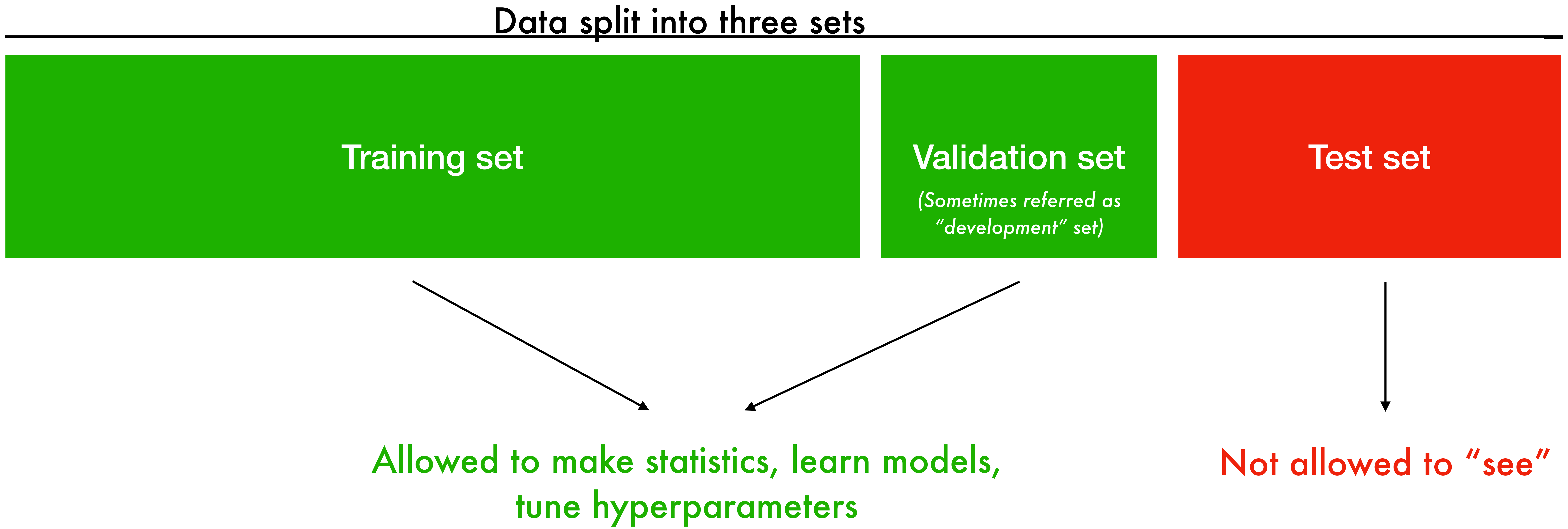
“Problems” with training

- Making poor predictions on the training data (underfitting)
- Not generalizing to unseen data (overfitting)

Example: polynomial regression of degree M

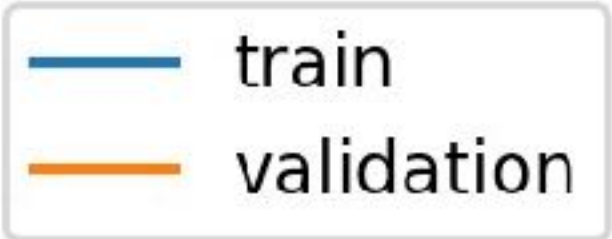


“Typical” machine learning setup

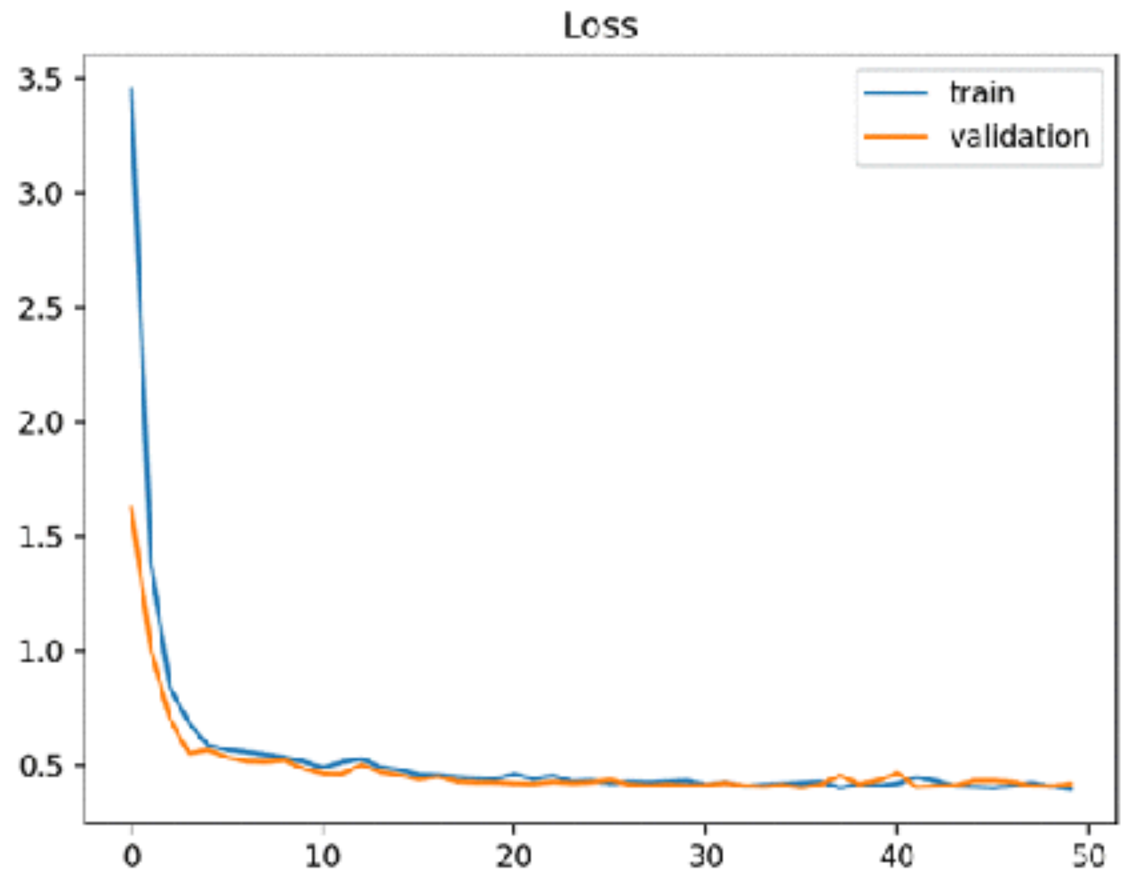


- Learn models on the **training** set
- Evaluate on the **validation** set many times (run experiments to find good hyperparameters, e.g., number of epochs, learning rate, batch size...)
- (Optional: Learn the final model on the combination of **training and validation** sets)
- Evaluate on the **test** set “once”

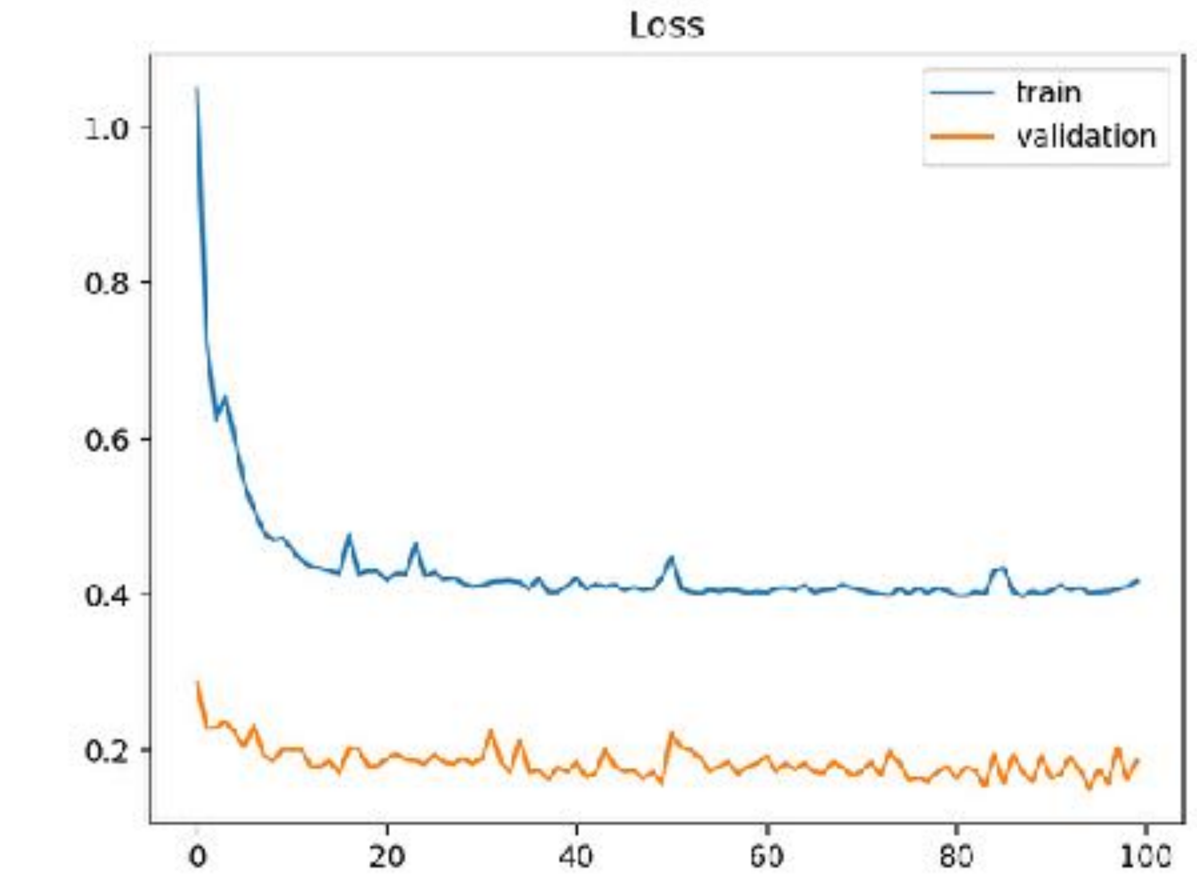
A few possible scenarios for learning curves



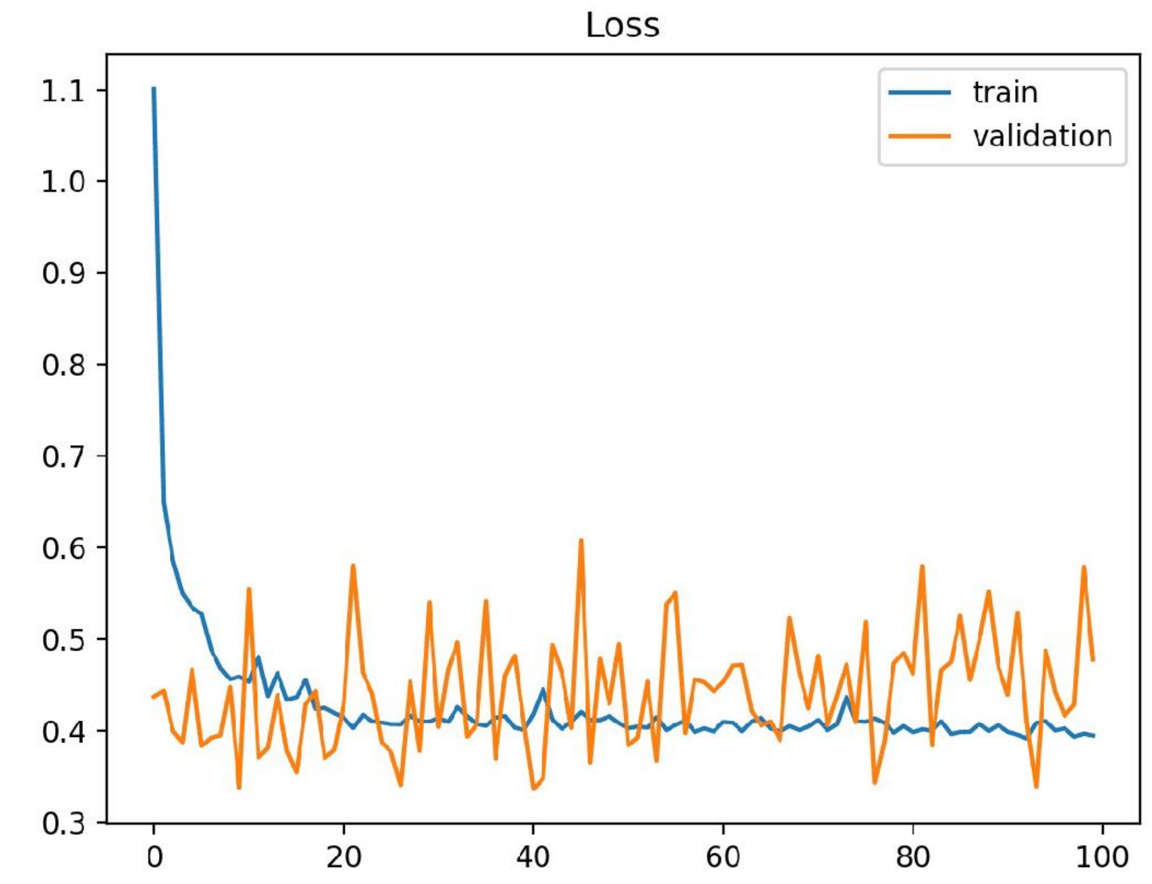
Good fit: both decreasing, converging, minimal gap



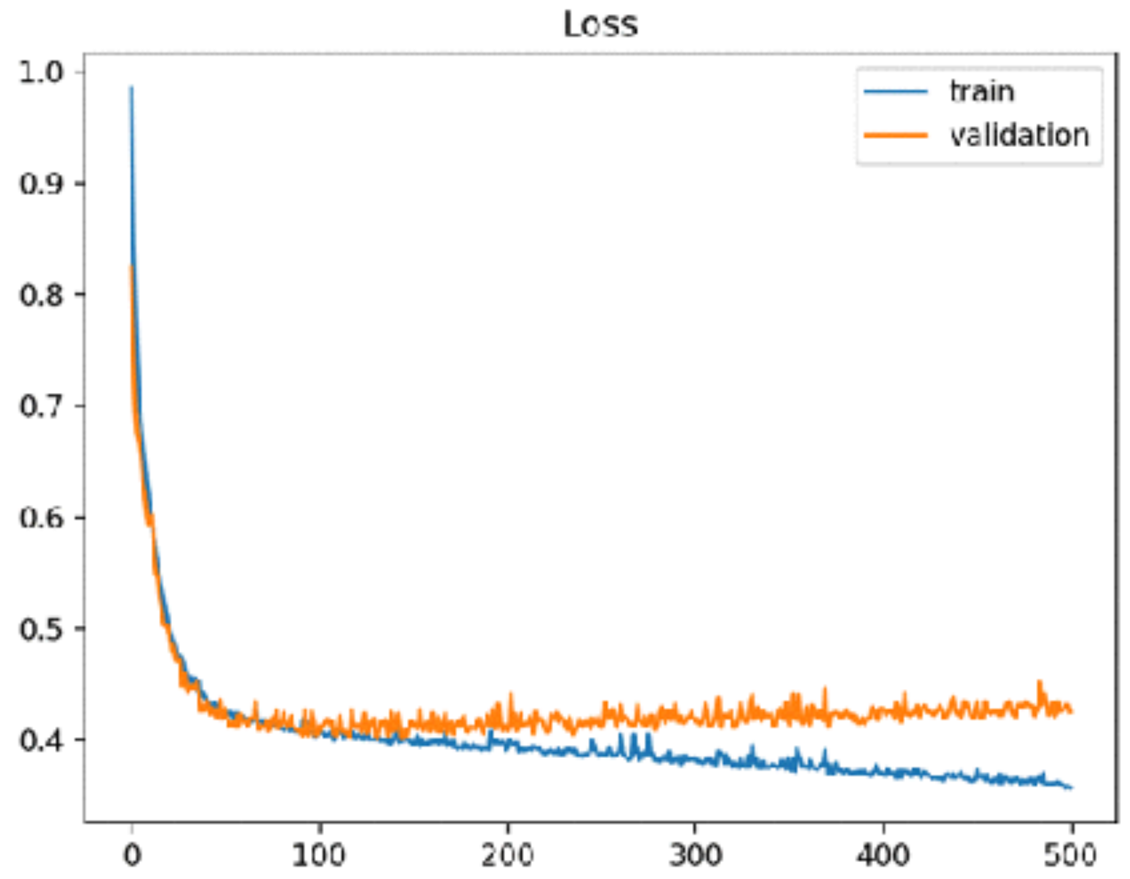
Unrepresentative validation set: easier than training set



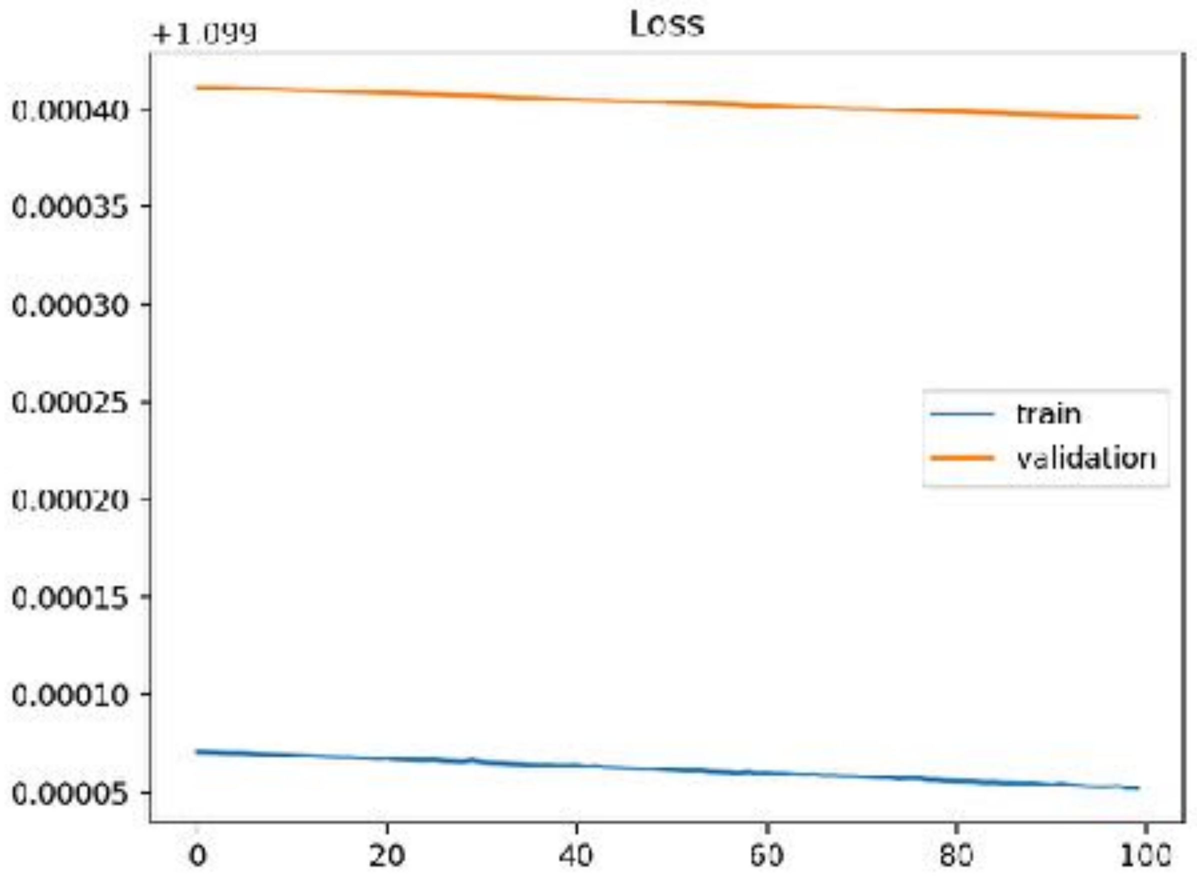
Unrepresentative validation set: too few examples



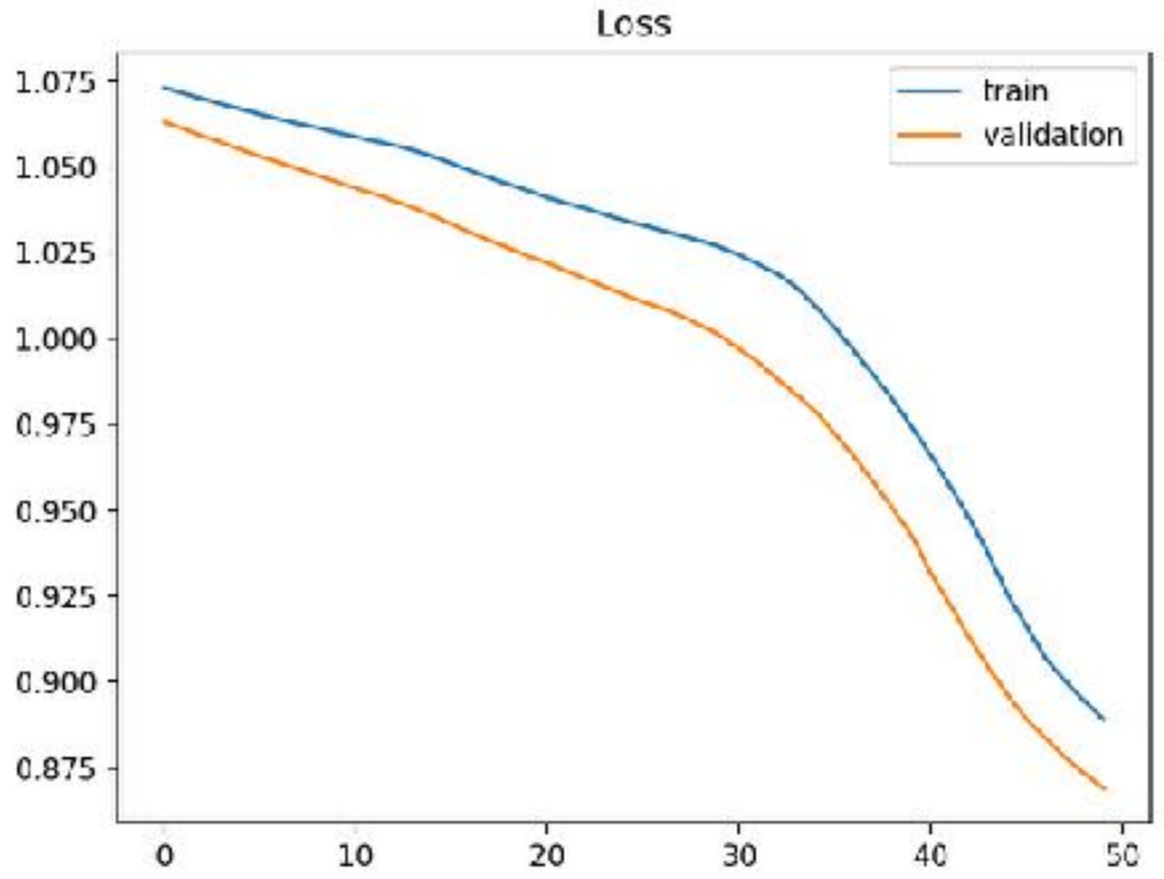
Overfit: validation increasing



Underfit: training loss not decreasing



Underfit: training halted prematurely



loss

number of iterations/epochs

Credit: **Jason Brownlee**

Image sources

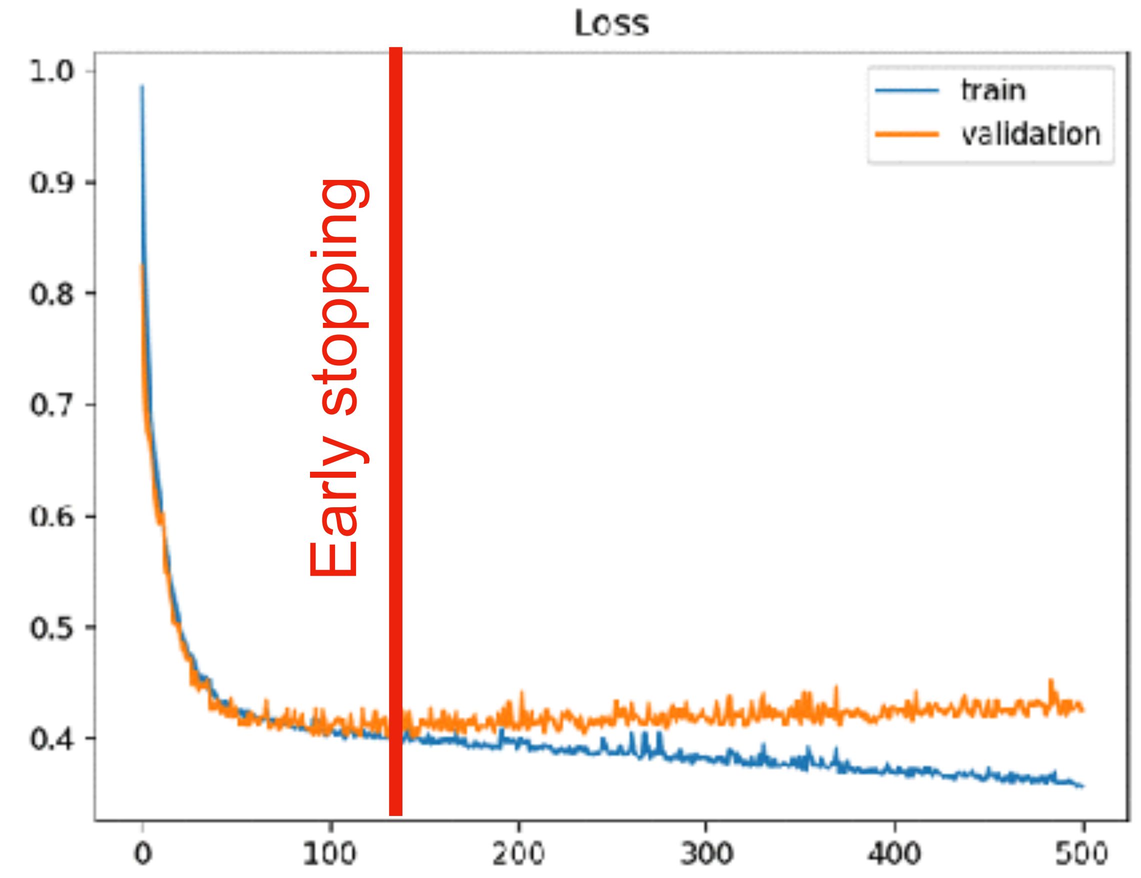
How to avoid overfitting?

Deep networks have many parameters.
Some regularization techniques:

- Smaller network, i.e., less parameters
- Data augmentation
- Suboptimize, i.e., "early stopping"
- Force redundancy in hidden units, i.e., "dropout"
- Penalize parameter norms, i.e., "weight decay"

Data augmentation

Image source



L2 penalty:
encourages the norm of
the parameters to be low

$$L(\theta) + \underbrace{\frac{\lambda}{2} ||\theta||_2^2}$$

$$\theta_{t+1} \rightarrow \theta_t - \alpha \frac{\partial L}{\partial \theta_t} - \underbrace{\alpha \lambda \theta_t}_{\text{gradient}}$$

Look at your results

- When you train a network, you should try to really understand what is happening:
  - Train/val/test sets are important
  - **<u>Look at loss</u>** and performance on train/val sets during training
  - Choose LR, compare networks, try different initialization (random seeds)

- **Very important: <u>Look</u>** at your data <u>and</u> results (e.g., visualize predictions) on training <u>and</u> testing data.

Slide: M. Aubry

Practical problems

- **Data loading:**
  - Loading "on the fly": needed for big datasets, use efficient database structure, fast disk access, e.g., SSD
  - Loading to RAM: possible for smaller datasets, or pre-computed features
- **Speed:** use GPUs, parallel data loading
- **Network size:** get lots of memory on your GPU or/and use several GPUs

Good news: you don't have to do all of it!

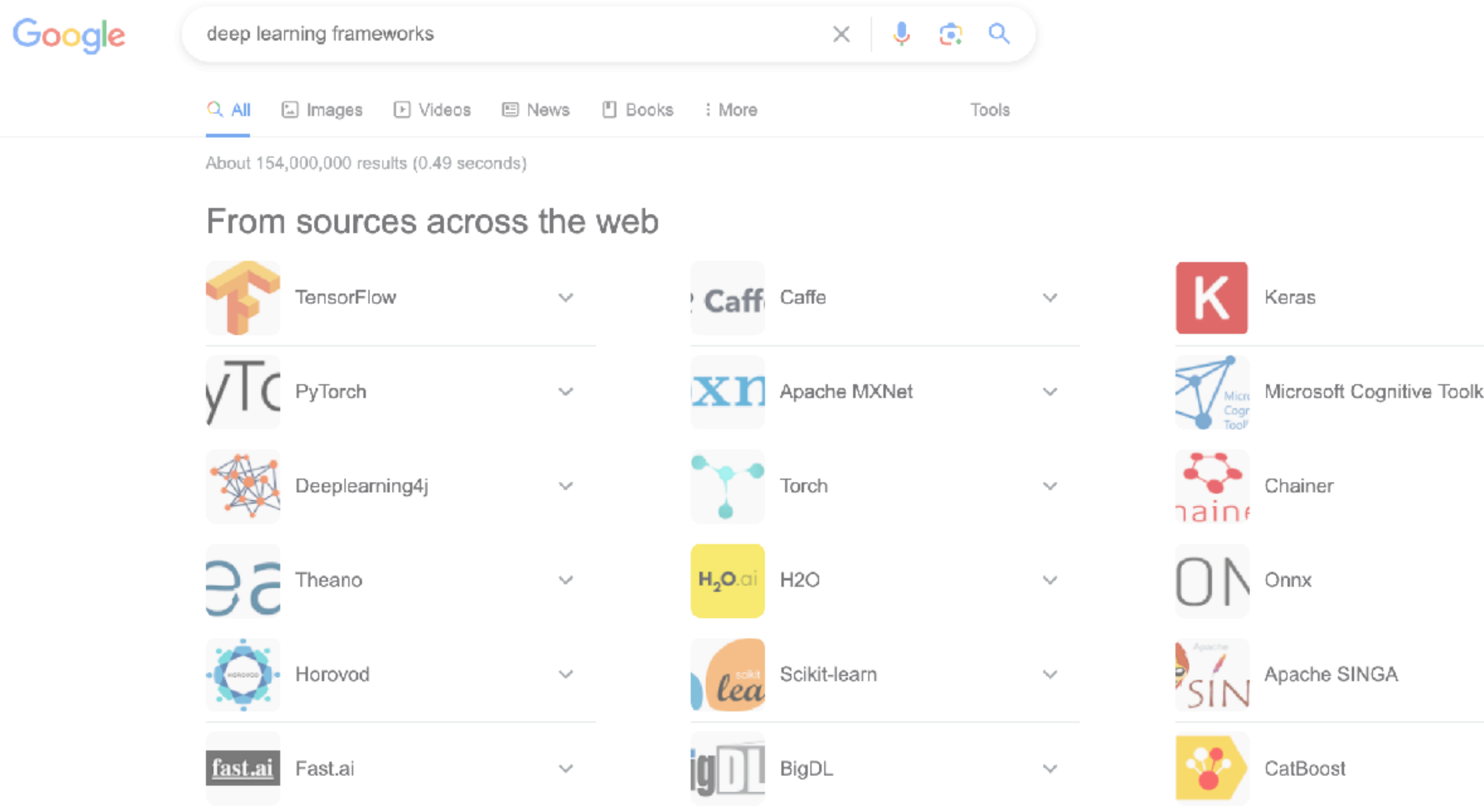
Many ready-to-use and efficient frameworks are available (e.g., Pytorch)

Slide: M. Aubry

NN packages

- PyTorch (Python)
  - http://pytorch.org/
- TensorFlow (Python) - Google
  - https://www.tensorflow.org/
- Lua Torch
  - http://torch.ch/
- Caffe (C++, pycaffe, matcaffe)
  - http://caffe.berkeleyvision.org/
- MatConvNet (Matlab)
  - http://www.vlfeat.org/matconvnet/

…



Let’s look at some code

(more in Assignment 2)

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

model = Net()
if args.cuda:
    model.cuda()

optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum)

def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if args.cuda:
            data, target = data.cuda(), target.cuda()

        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

for epoch in range(1, args.epochs + 1):
    train(epoch)
```

- Key part of pytorch code for CNN learning

- The key objects are
  - model,
  - optimizer,
  - dataloader,
  - loss.

Slide: M. Aubry

Agenda

- **1. Recap: Bag of Visual Words, Analogy with NNs**

- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs

- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures

- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers

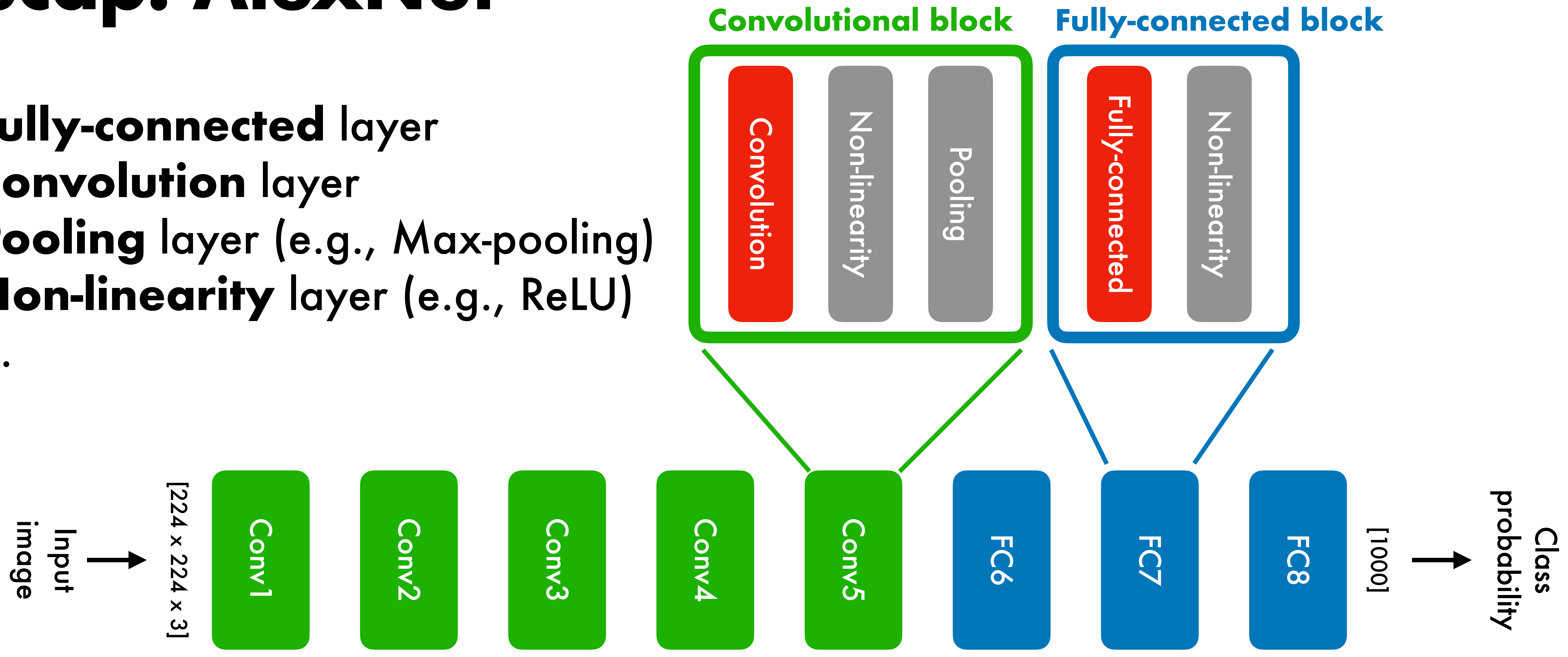
- **5. Beyond classification**

Visualizing CNNs

What does CNN learn once it is trained?

Recap: AlexNet

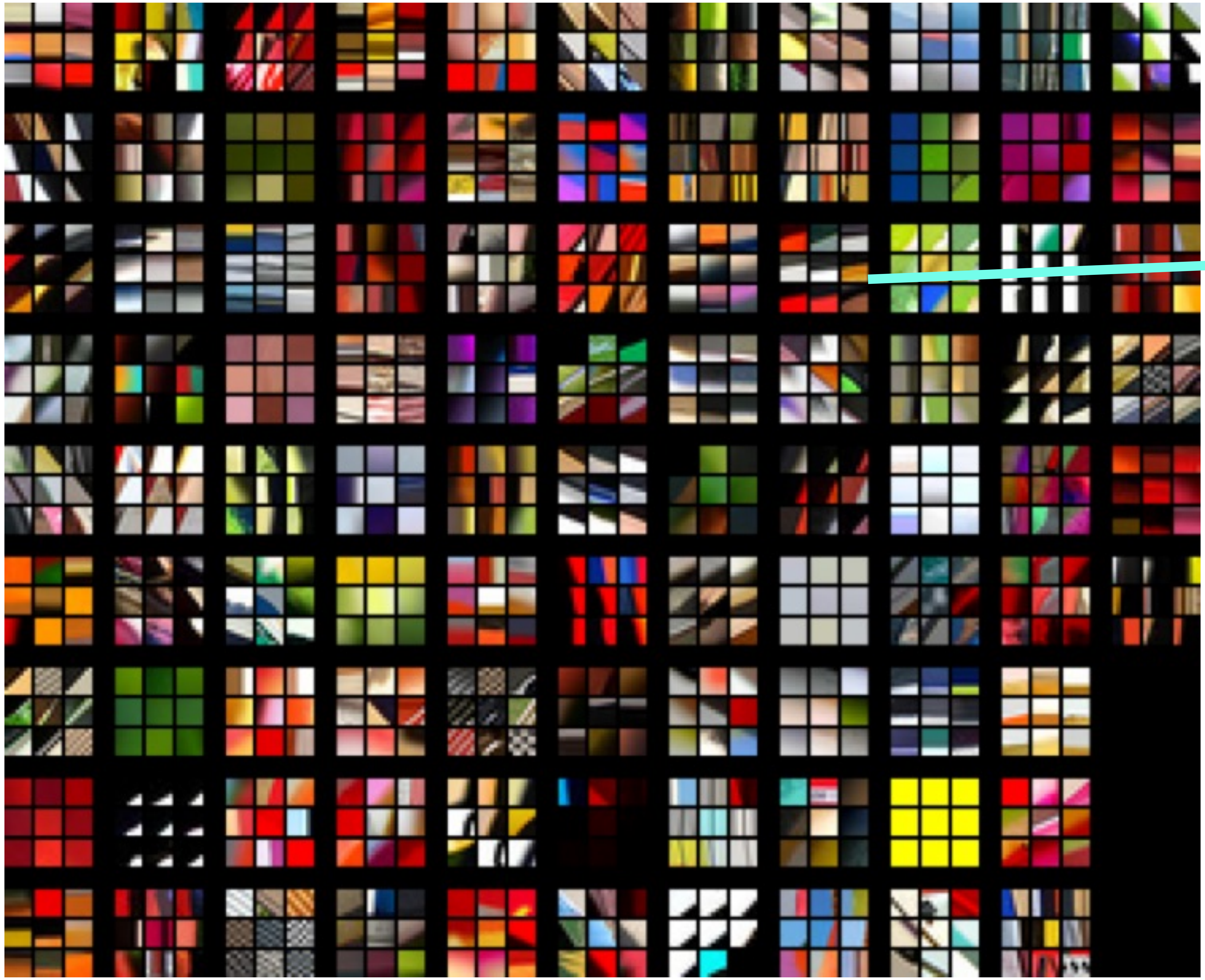
- **Fully-connected** layer
- **Convolution** layer
- **Pooling** layer (e.g., Max-pooling)
- **Non-linearity** layer (e.g., ReLU)
- …



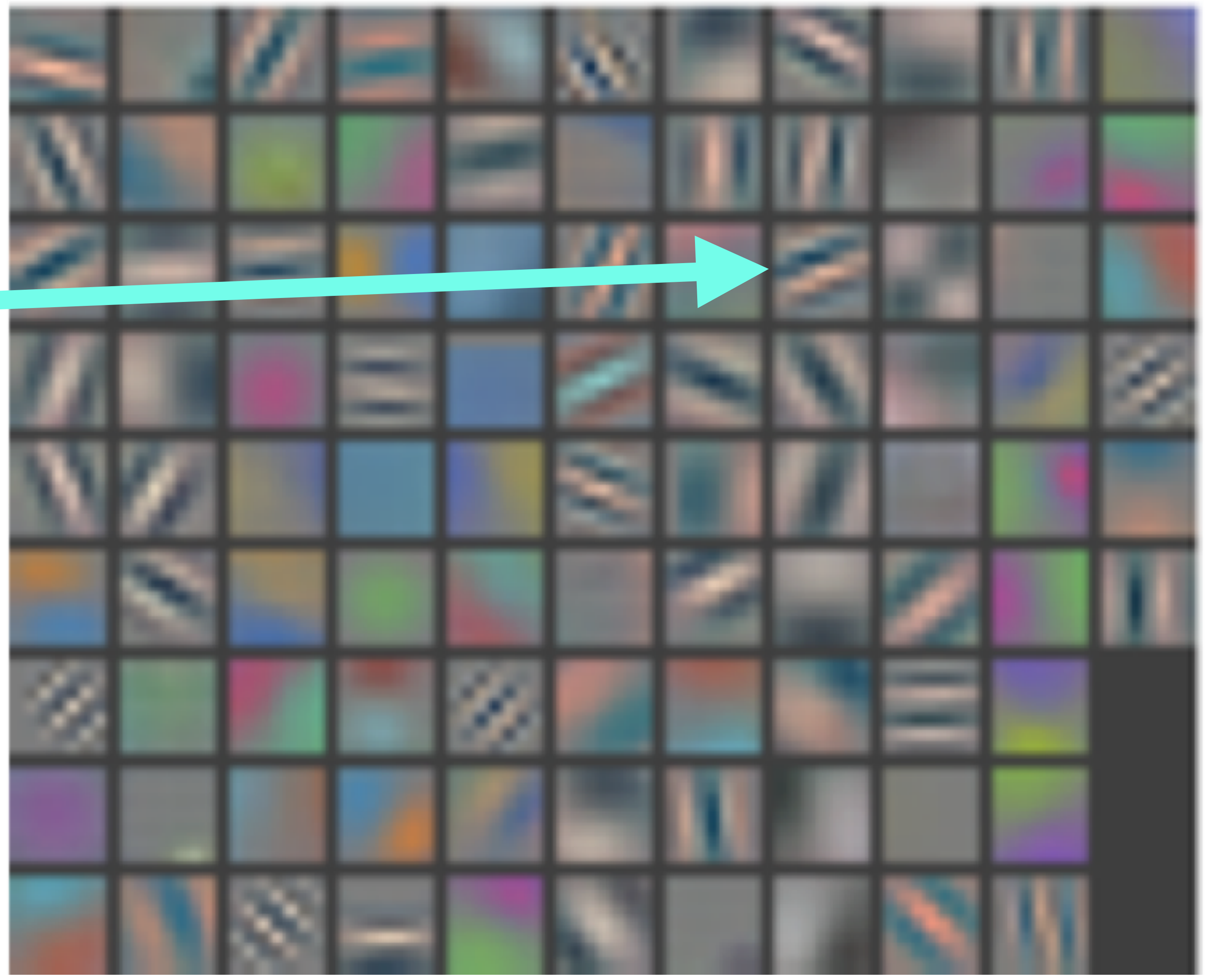
AlexNet: [Krizhevsky 2012]

Layer 1: Top-9 patches

Patches from validation images that give maximal activation of a given feature map



Learned filters



[Zeiler and Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014]

Layer 2: Top-9 patches

Layer 3: Top-9 patches

Layer 4: Top-9 patches

Layer 5: Top-9 patches

References: Visualizing and understanding NNs

Analysis tools

Visualizing higher-layer features of a deep network
Ethan et al. 2009
[intermediate features]

Deep inside convolutional networks
Simonyan et al. 2014
[deepest features, aka “deep dreams”]

DeConvNets
Zeiler et al. In ECCV, 2014
[intermediate features]

Understanding neural networks through deep visualisation
Yosinksi et al. 2015
[intermediate features]

Artistic tools

Google’s “inceptionism”
Mordvintsev et al. 2015

Style synthesis and transfer

Gatys et al. 2015

Slide: A. Vedaldi

Agenda

- **1. Recap: Bag of Visual Words, Analogy with NNs**

- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs

- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures

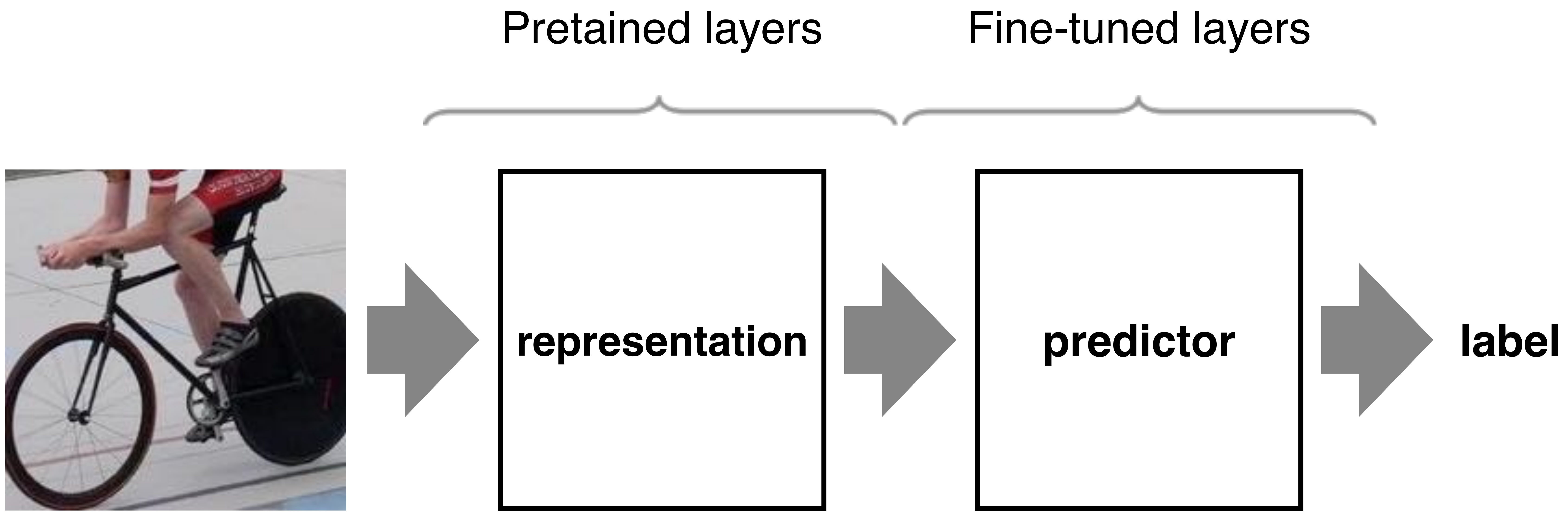
- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers

- **5. Beyond classification**

Transferring learnt representations

"pretraining"

“Pre-training” and transfer learning



CNN as universal representations

- First several layers in most CNNs are generic
- They can be reused when training data is comparatively scarce

Application

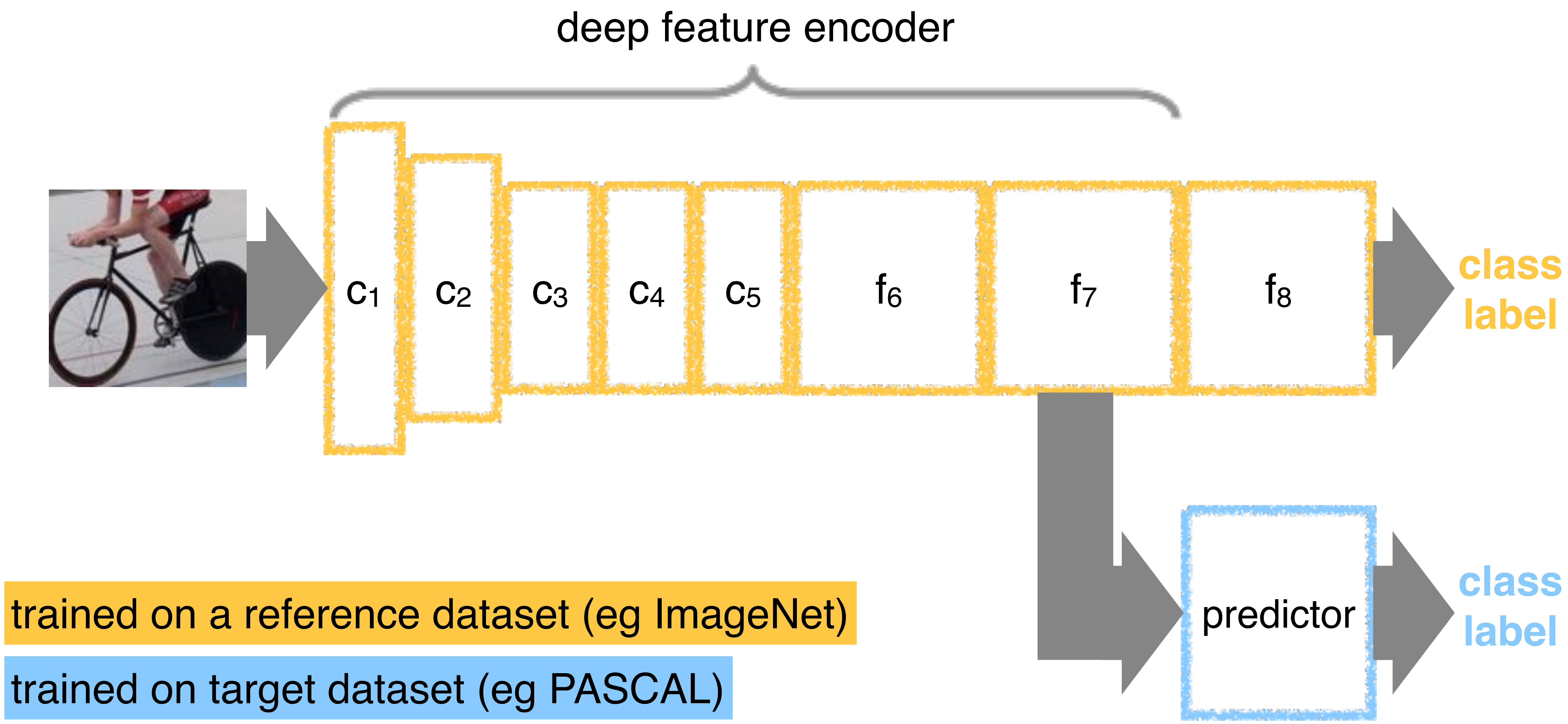
- Pre-train on ImageNet classification 1M images
- Cut at some deep conv or FC layer to get features

[Evaluations in A. S. Razavian, 2014, Chatfield et al., 2014]

Slide: A. Vedaldi

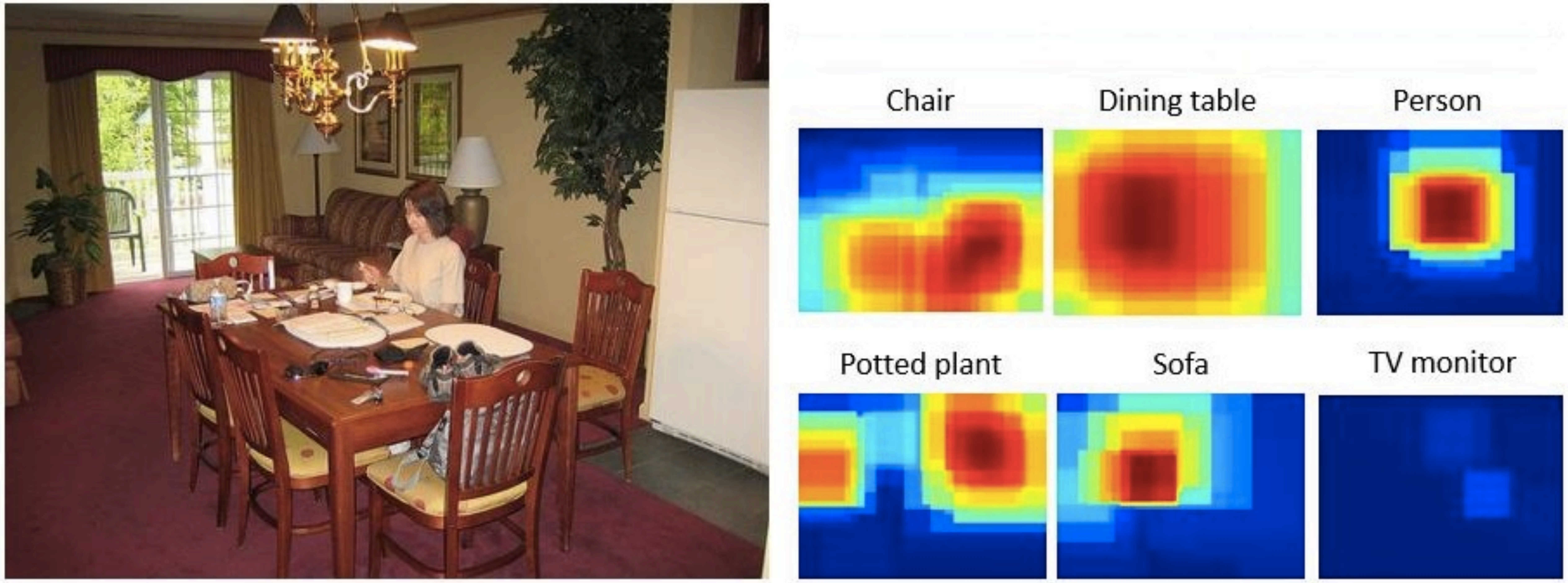
“Pre-training” and transfer learning

Deep representations are generic



A general purpose deep encoder is obtained by chopping off the last layers of a CNN trained on a large dataset.

Example



Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks
M. Oquab, L. Bottou, I. Laptev, J. Sivic
In CVPR 2014
http://www.di.ens.fr/willow/research/cnn/

Slide: A. Vedaldi

ImageNet classification challenge

ImageNet classification challenge

Object centric

1000 classes

1.2M images

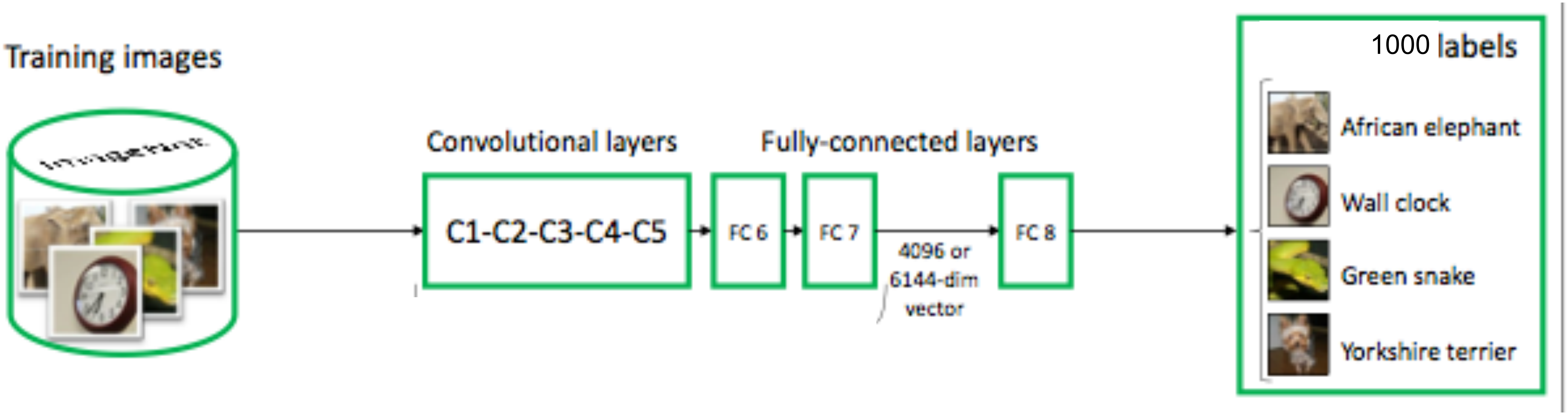
What about other recognition tasks and datasets?

ImageNet classification challenge

Object centric
1000 classes
1.2M images

Complex scenes
20 classes
10k images

Background – Convolutional neural network of [Krizhevsky et al. 2012]



Input: ~1M **labelled** images (1000 images / 1000 classes)

Number of parameters: ~60 million --- image representation

Training time: ~1 week on one GPU

Learn parameters using stochastic gradient descent on cross-entropy error function.

Can we transfer learnt parameters to other tasks with limited training data?

Challenge

The dataset statistics between the source task (ImageNet) and the target task (Pascal VOC) can be very different.

- Type of objects and labels
- Object size, object location, scene clutter
- Object viewpoints, imaging conditions

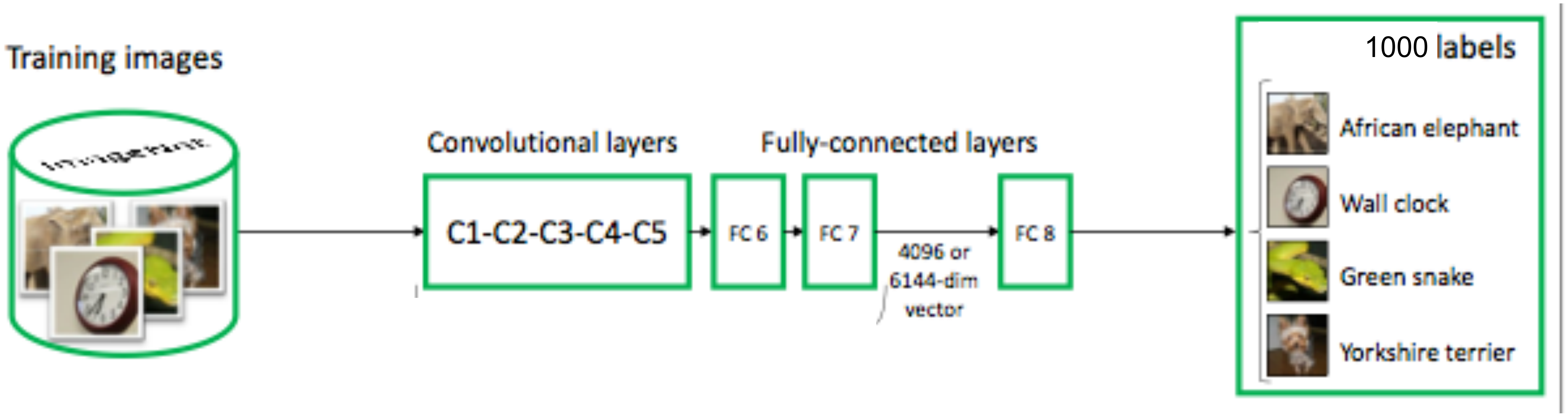
ImageNet

Maltese terrier

Pascal VOC

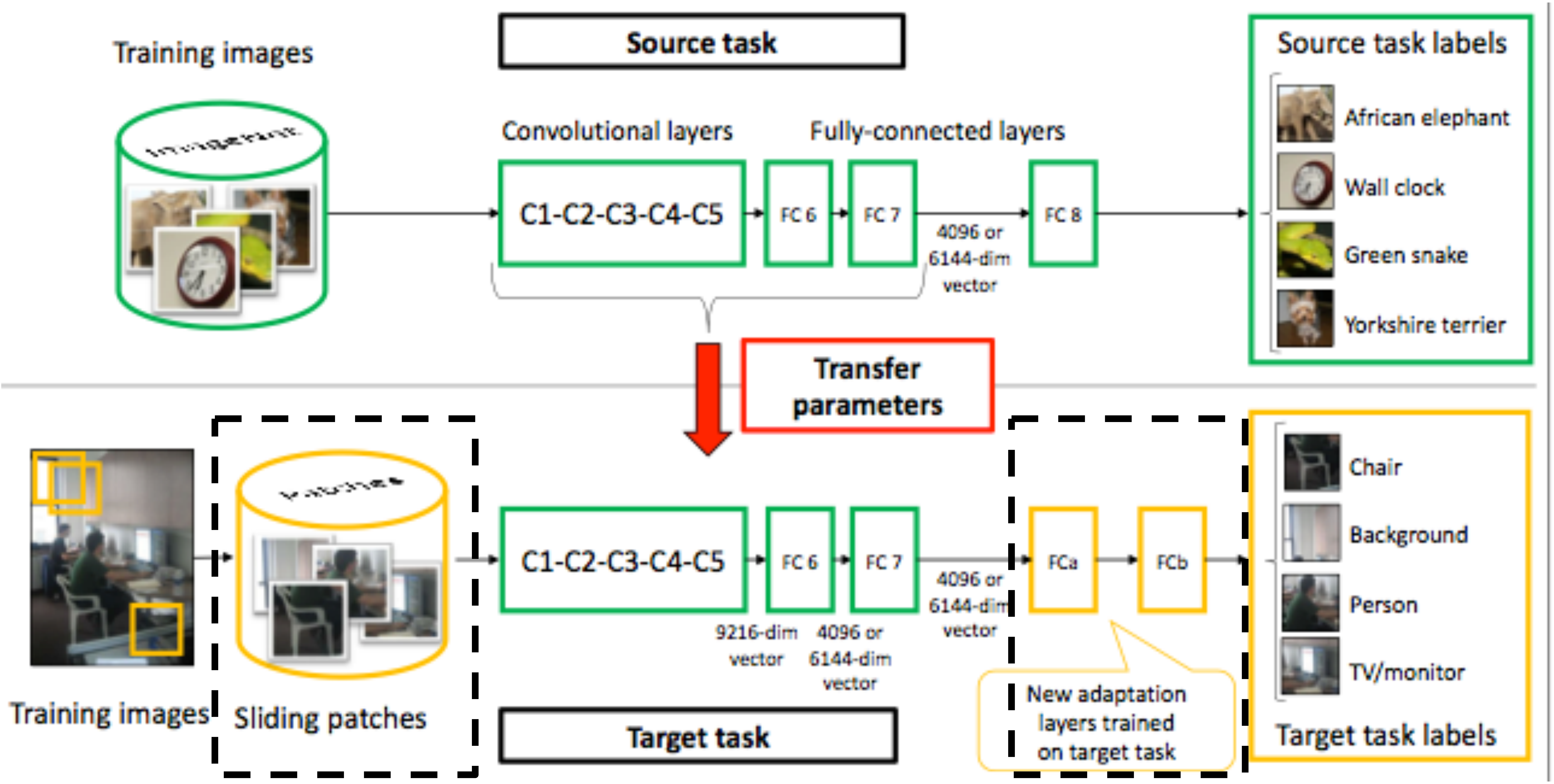
Dog

Approach



Training images
Convolutional layers
Fully-connected layers
C1-C2-C3-C4-C5
FC 6
FC 7
4096 or
6144-dim
vector
FC 8
1000 labels
African elephant
Wall clock
Green snake
Yorkshire terrier

Approach [Oquab, Bottou, Laptev, Sivic, CVPR'14]



1. Design training/test procedure using sliding windows
2. Train adaptation layers to map labels

See also [Girshick et al.'13], [Donahue et al.'13], [Sermanet et al. '14], [Zeiler and Fergus '13]
Transfer learning workshop at ICCV'13, ImageNet workshop at ICCV'13

Pre-training helps

	plane	bike	bird	boat	btl	bus	car	cat	chair	cow	table	dog	horse	moto	pers	plant	sheep	sofa	train	tv	**mAP**
NO PRETRAIN	85.2	75.0	69.4	66.2	48.8	82.1	79.5	79.8	62.4	61.9	49.8	75.9	71.4	82.7	93.1	59.1	69.7	49.3	80.0	76.7	70.9
PRE-1000C	93.5	78.4	*87.7*	80.9	57.3	85.0	81.6	*89.4*	66.9	73.8	62.0	*89.5*	83.2	87.6	*95.8*	*61.4*	79.0	54.3	88.0	78.3	78.7
PRE-1000R	93.2	77.9	83.8	80.0	55.8	82.7	79.0	84.3	66.2	71.7	59.5	83.4	81.4	84.8	95.2	59.8	74.9	52.9	83.8	75.7	76.3
PRE-1512	94.6	82.9	**88.2**	84.1	60.3	89.0	84.4	**90.7**	72.1	**86.8**	69.0	**92.1**	**93.4**	88.6	**96.1**	**64.3**	**86.6**	62.3	91.1	79.8	**82.8**

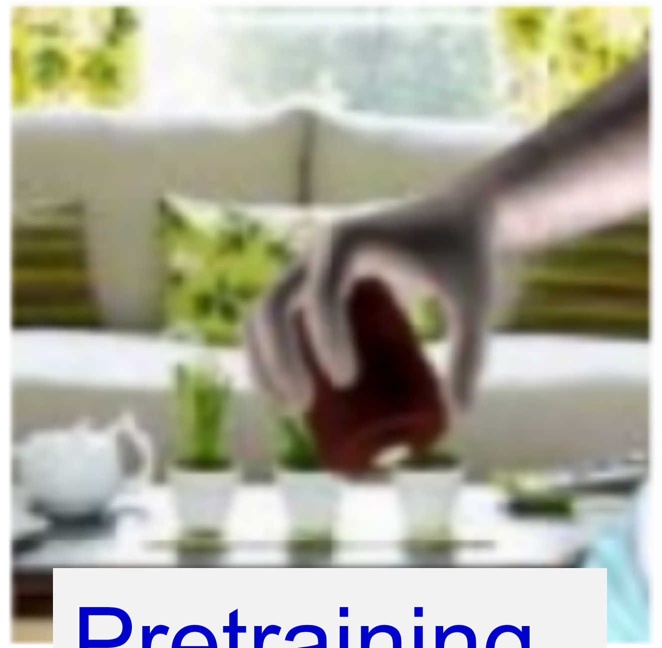
- Pascal VOC 2012 object classification

Action	jump	phon	instr	read	bike	horse	run	phot	comp	walk	**mAP**
NO PRETRAIN	43.2	30.6	50.2	25.0	76.8	80.7	75.2	22.2	37.9	55.6	49.7
PRE-1512	73.4	44.8	**74.8**	43.2	92.1	94.3	83.4	**45.7**	65.5	66.8	68.4
PRE-1512U	74.8	46.0	**75.6**	**45.3**	93.5	95.0	86.5	**49.3**	66.7	69.5	**70.2**

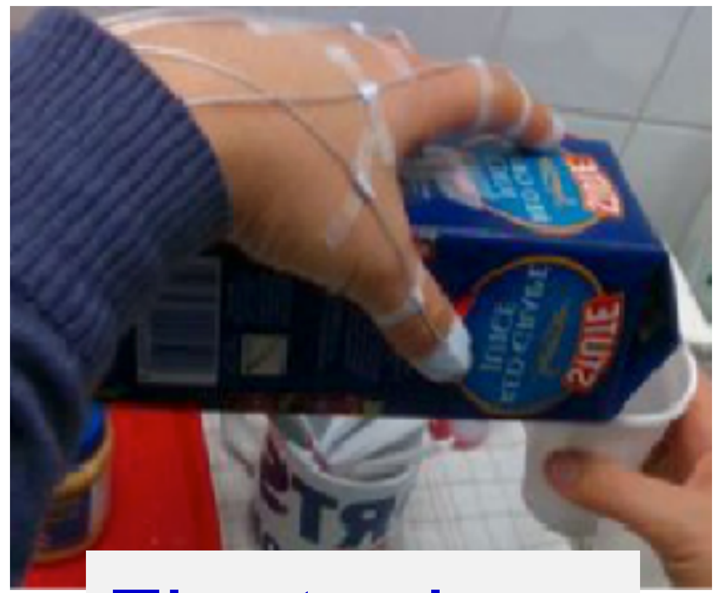
- Pascal VOC 2012 action classification

Other "pre-training" examples

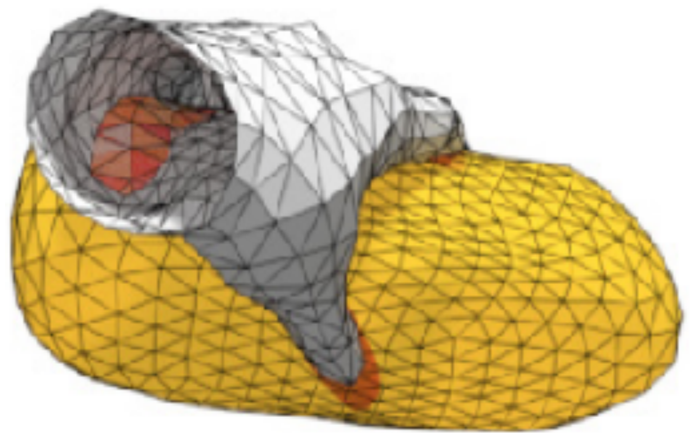
3D hand-object reconstruction



Pretraining



Finetuning



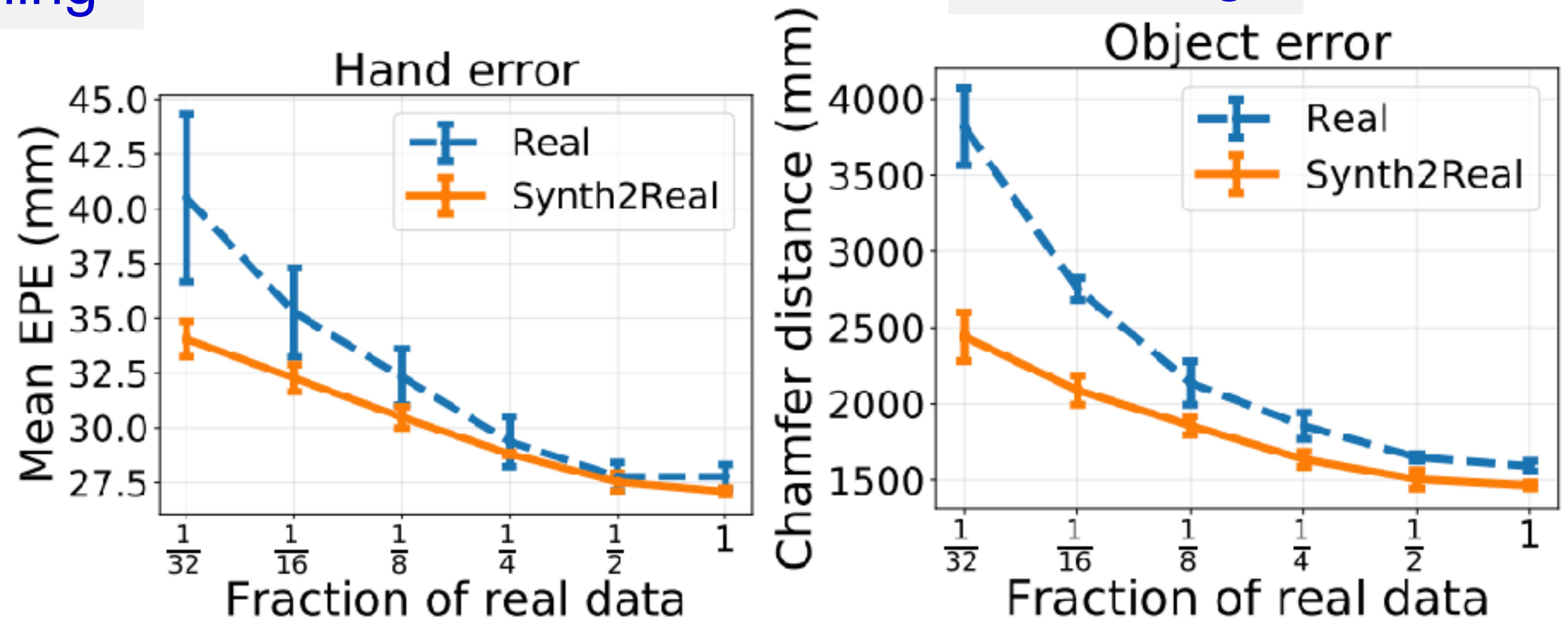
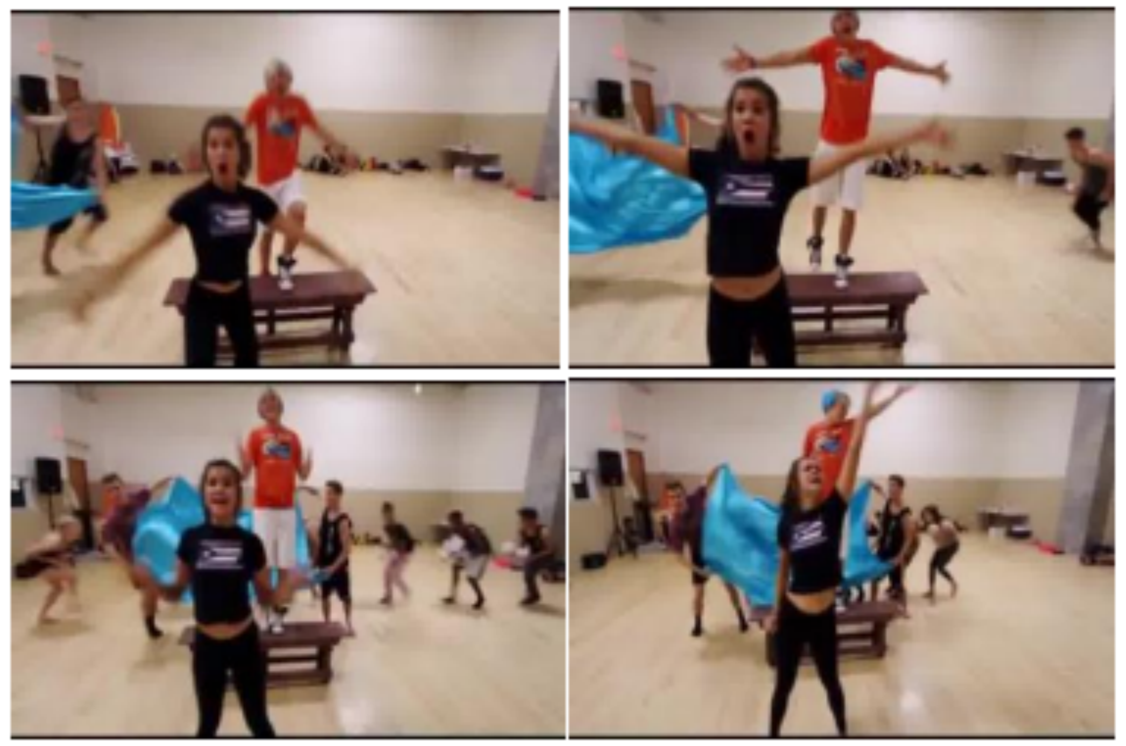


Figure 8: We compare training on FHB only (Real) and pre-training on synthetic, followed by fine-tuning on FHB (Synth2Real). As the amount of real data decreases, the benefit of pre-training increases. For both the object and the hand reconstruction, synthetic pre-training is critical in low-data regimes.

Hasson et al. **"Learning joint reconstruction of hands and manipulated objects",** CVPR 2019.

Text-to-Video Retrieval

Pretraining on millions of images & videos
Finetuning on MSRVTT with 9K training videos



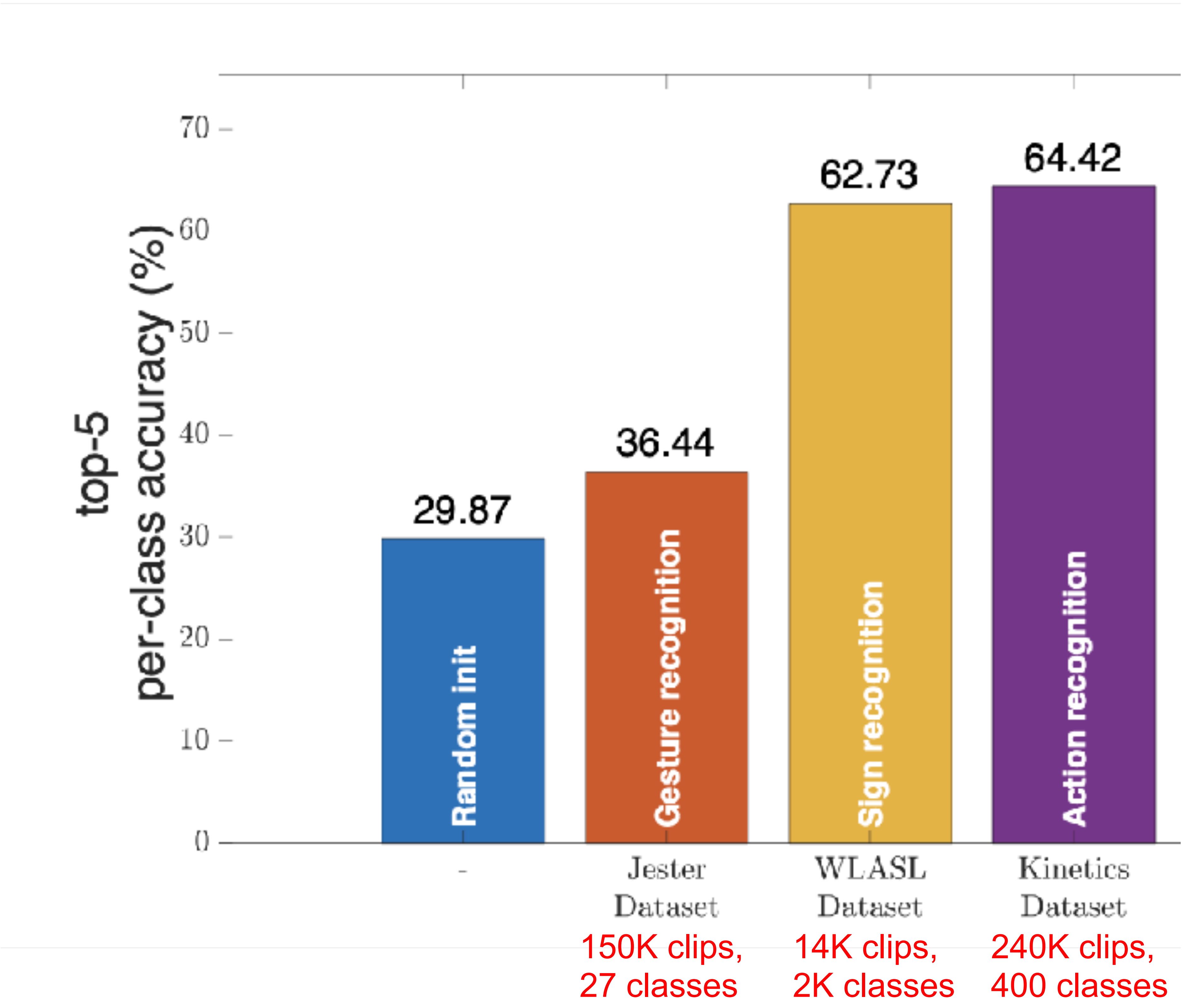
1. A man and a woman performing a musical.
2. A teenage couple perform in an amateur musical
3. Dancers are playing a routine.
4. People are dancing in a musical.
5. Some people are acting and singing for performance.

Pre-training (for 1 epoch)	#pairs	↑R@1	↑R@10	↓MedR
-	-	5.6	22.3	55
ImageNet		15.2	54.4	9.0
Noisy HowTo-17M subset	17.1M	24.1	63.9	5.0
CC3M	3.0M	24.5	62.7	5.0
WebVid2M	2.5M	26.0	64.9	5.0
CC3M + WebVid2M	**5.5M**	**27.3**	**68.1**	**4.0**

Bain et al. "**Frozen in Time: A Joint Video and Image Encoder for End-to-End Retrieval**", ICCV 2021.

Sign Language Recognition

Pretraining on various tasks on different datasets
Finetuning on 50K videos from BSL-1K sign language dataset



Albanie et al. "**BSL-1K: Scaling up co-articulated sign language recognition using mouthing cues",** ECCV 2020.

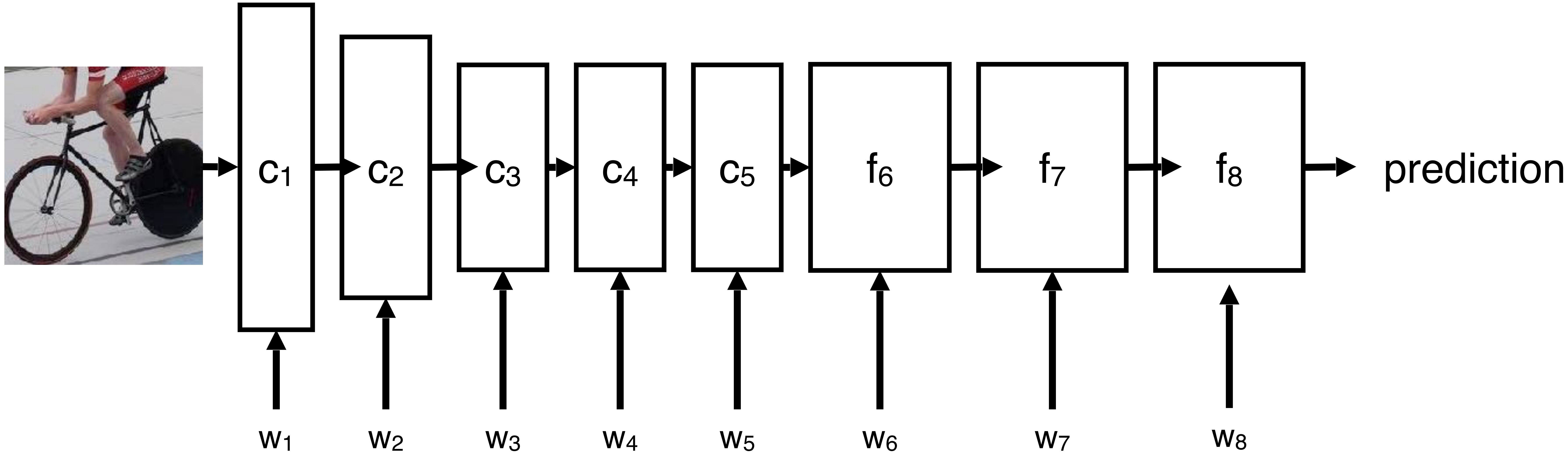
Pretraining Summary

- Common practice: Pretrain on large data, finetune on small data.
  - Remove the last class-specific layer (e.g. 1000 categories)
  - Add new layer(s) for the new task randomly initialized
  - Either "freeze" the pretrained parameters and train a simple classifier on top,
  - Or train "end-to-end" all parameters.
- Avoids overfitting
- Shortens training time
- Lots of pretrained models available online
- Task and domain-relevant pretraining is usually better

Agenda

- **1. Recap: Bag of Visual Words, Analogy with NNs**
- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs
- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures
- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers
- **5. Beyond classification**

A CNN for image classification

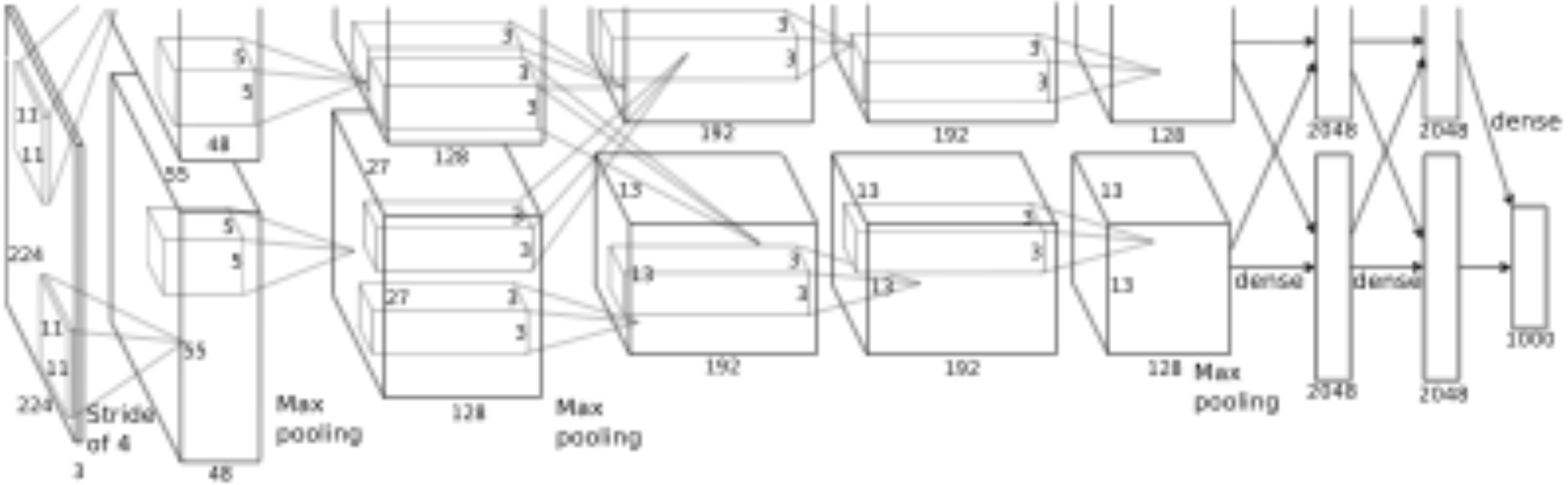


Recall: the goal of this model is to map an input image to a class prediction.

Slide: A. Vedaldi

Recall: The AlexNet model

A breakthrough in image understanding



[AlexNet by Krizhevsky et al. 2012]

Each large block represents a data tensor

Each smaller block represents a filter

The filter size and stride are shown

The number of filters can be deduced from the number of feature channels

There are two parallel streams in this network (for efficiency reasons)

How deep is deep enough?

AlexNet (2012)

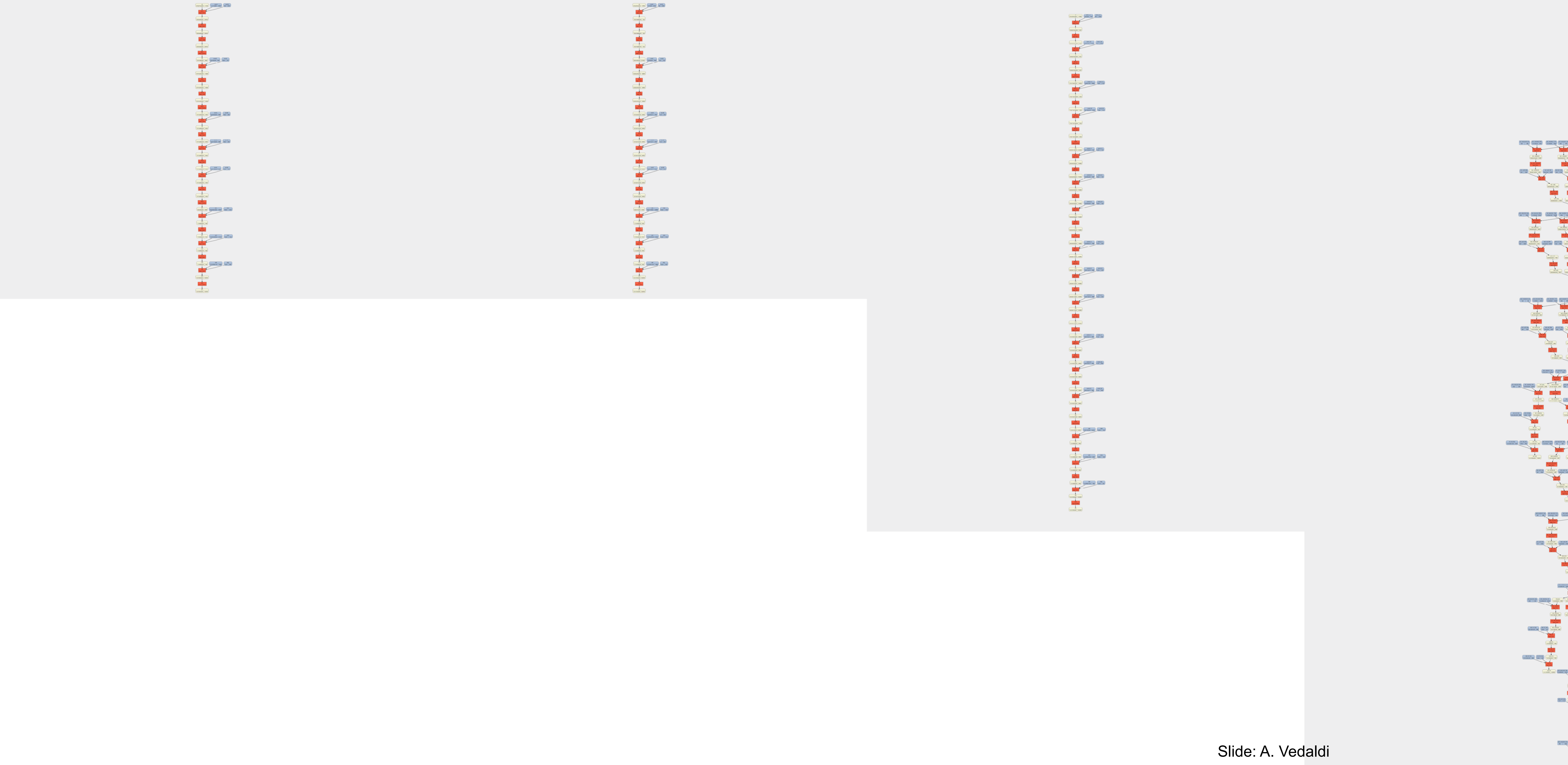
5 convolutional layers

3 fully-connected layers

Slide: A. Vedaldi

How deep is deep enough?

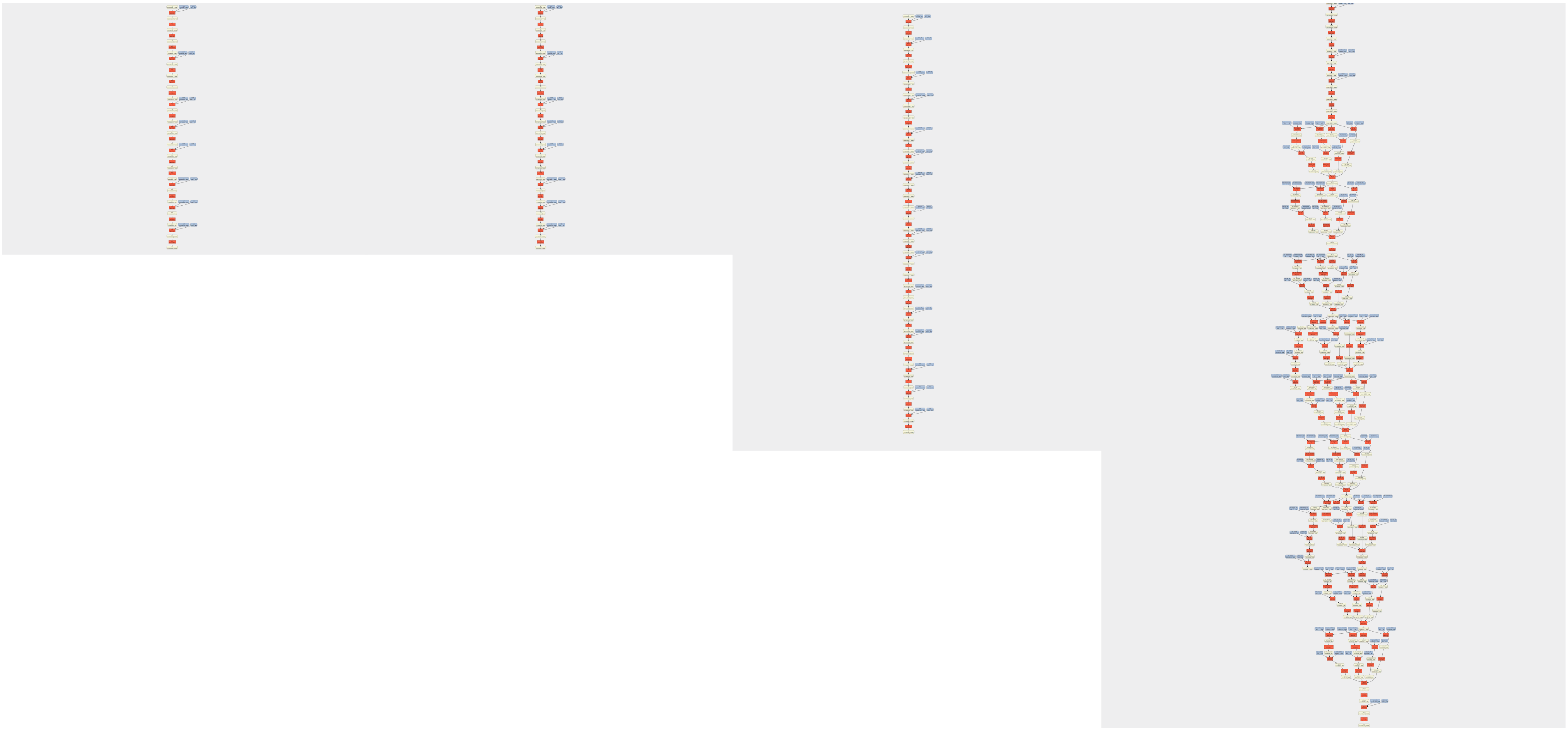
AlexNet (2012) VGG-M (2013) VGG-VD-16 (2014)



Slide: A. Vedaldi

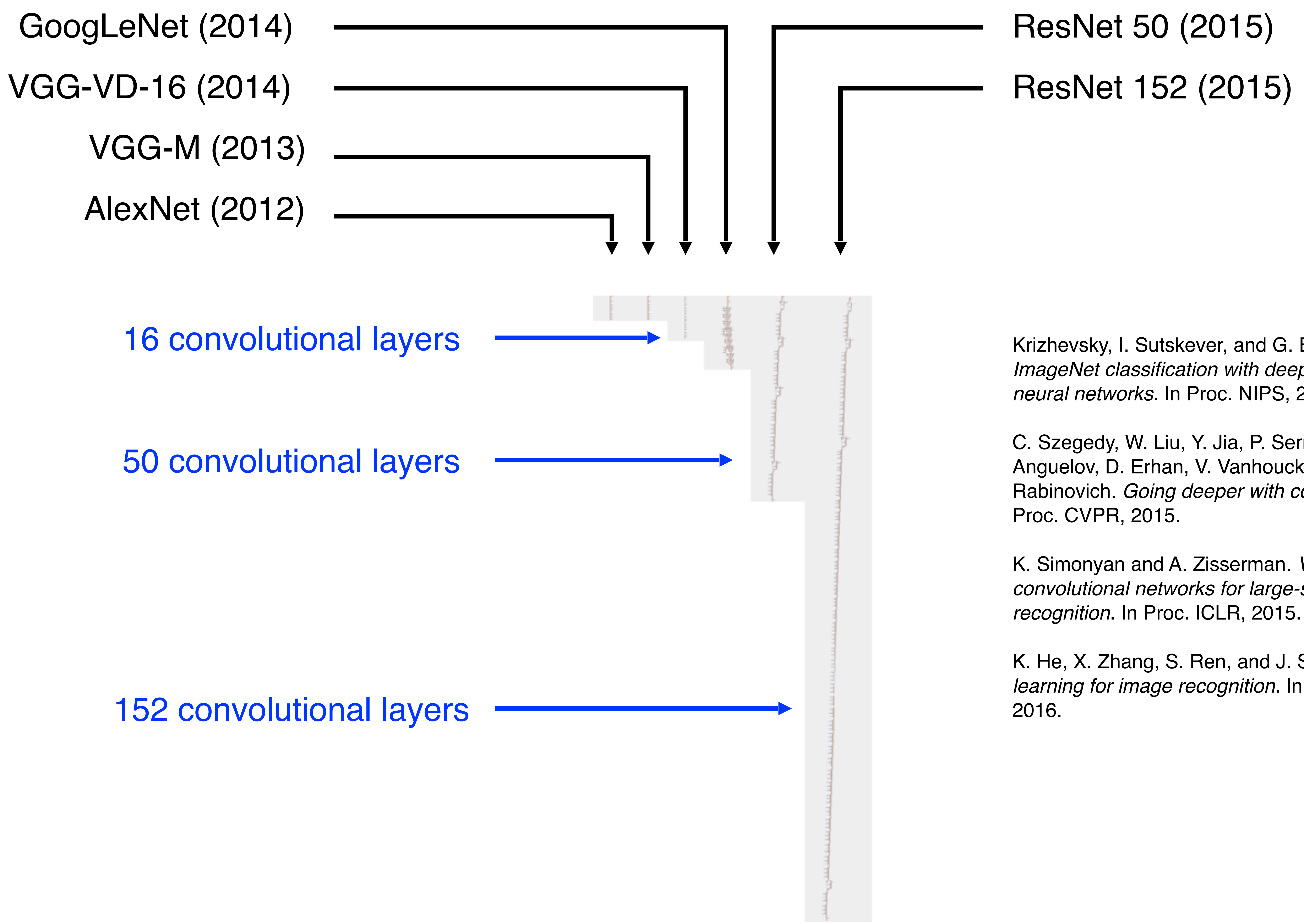
How deep is deep enough?

AlexNet (2012) VGG-M (2013) VGG-VD-16 (2014) GoogLeNet (2014)



Slide: A. Vedaldi

How deep is deep enough?



Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet classification with deep convolutional neural networks*. In Proc. NIPS, 2012.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions*. In Proc. CVPR, 2015.

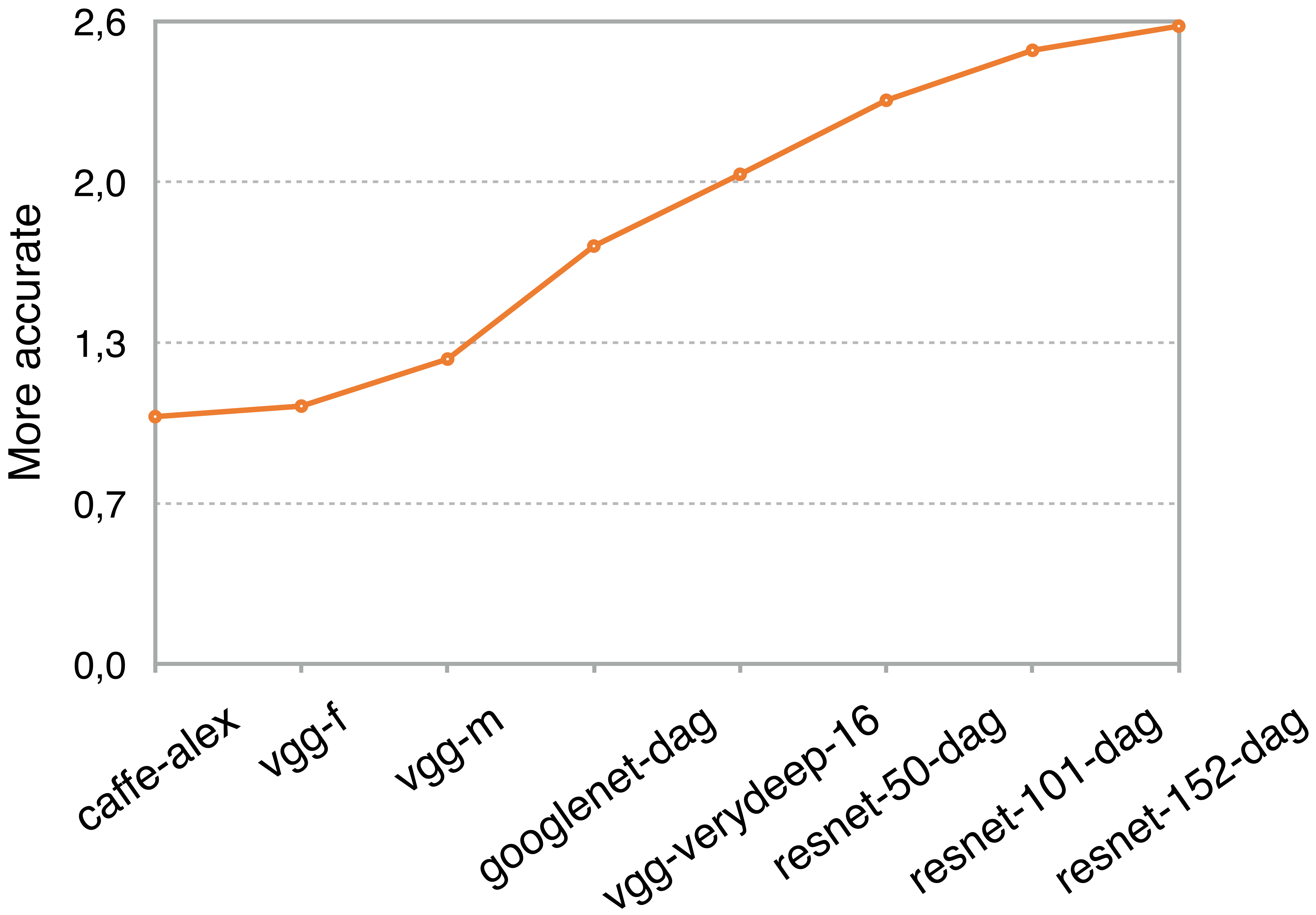
K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In Proc. ICLR, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition*. In Proc. CVPR, 2016.

Slide: A. Vedaldi

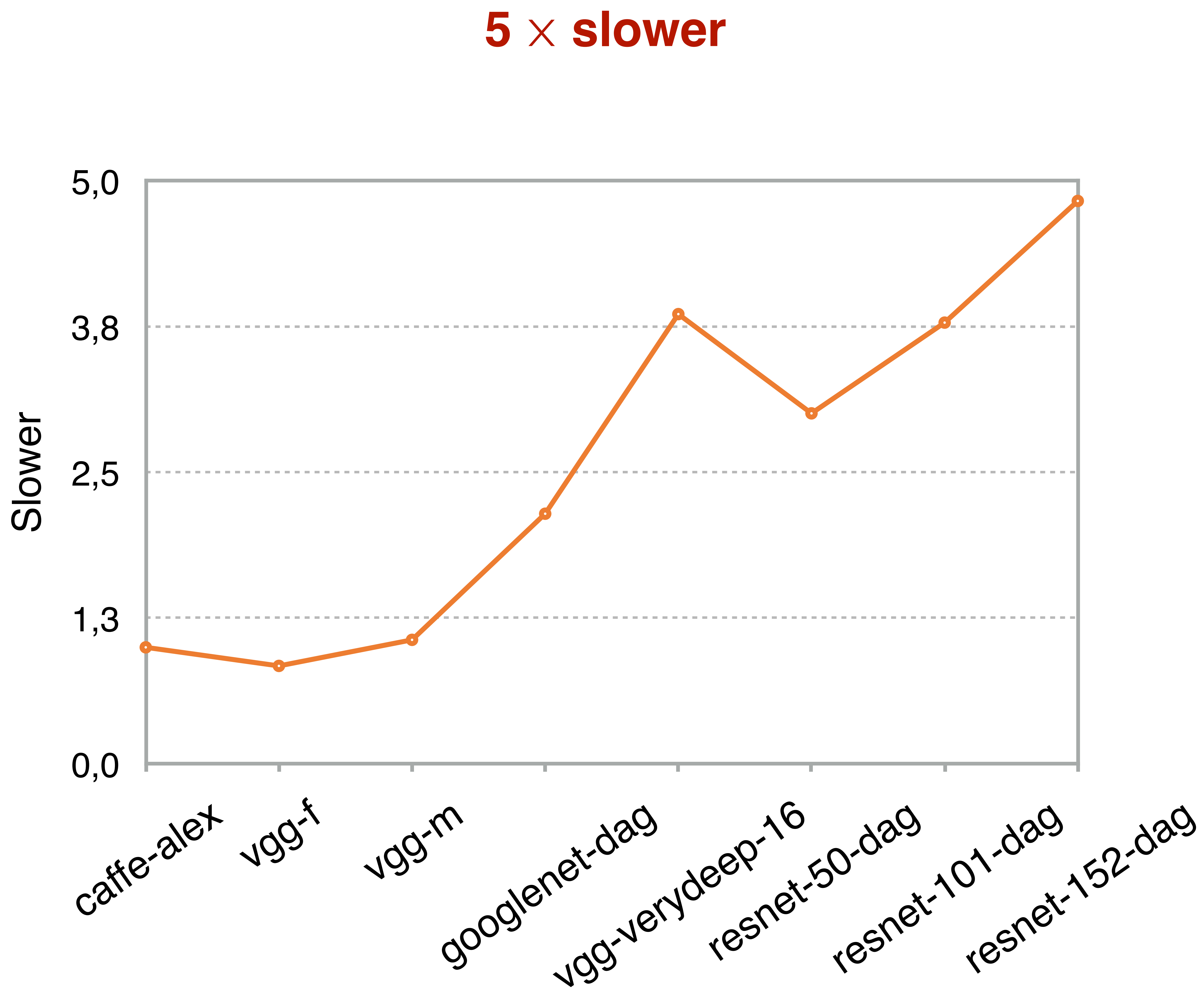
Accuracy

3 × more accurate in 3 years



Slide: A. Vedaldi

Speed



Remark: *101 ResNet* layers same size/speed as *16 VGG-VD* layers

Reason: far fewer feature channels (quadratic speed/space gain)

Moral: optimize your architecture

Slide: A. Vedaldi

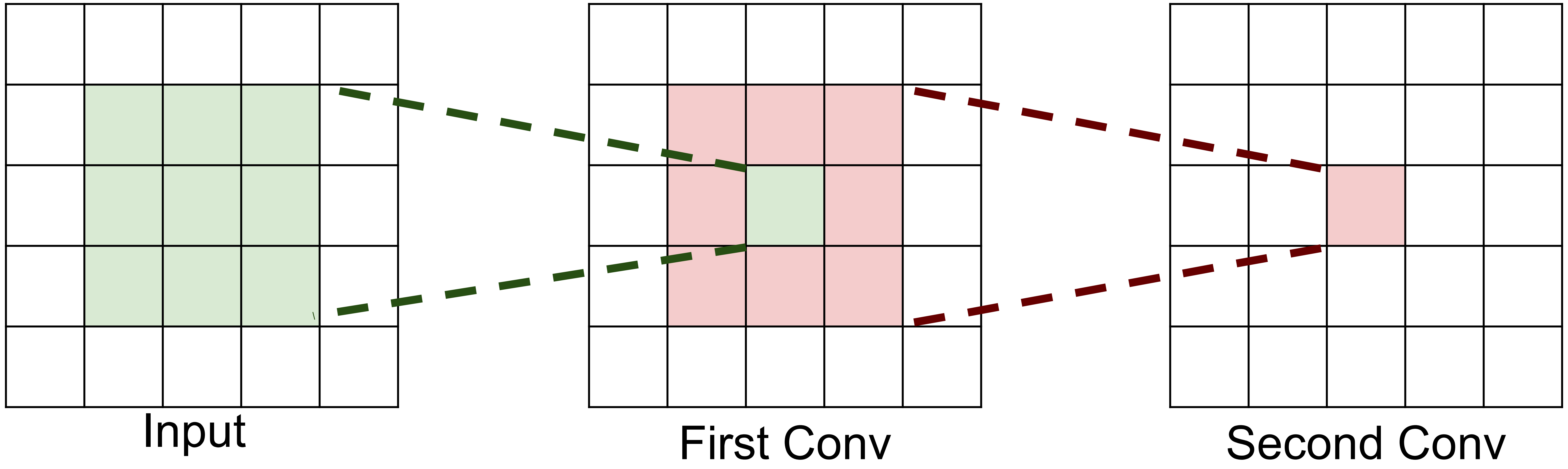
CNN architectures – notes and details

- Increased depth of recent architectures
- Number of parameters matter (How to count parameters?)
- Power of small filters, e.g. 3x3 convolutions
- ResNet architecture

The power of small filters

Suppose we stack two CONV layers with **receptive field** size 3x3

Q: What region of input does each neuron in 2^{nd} CONV see?



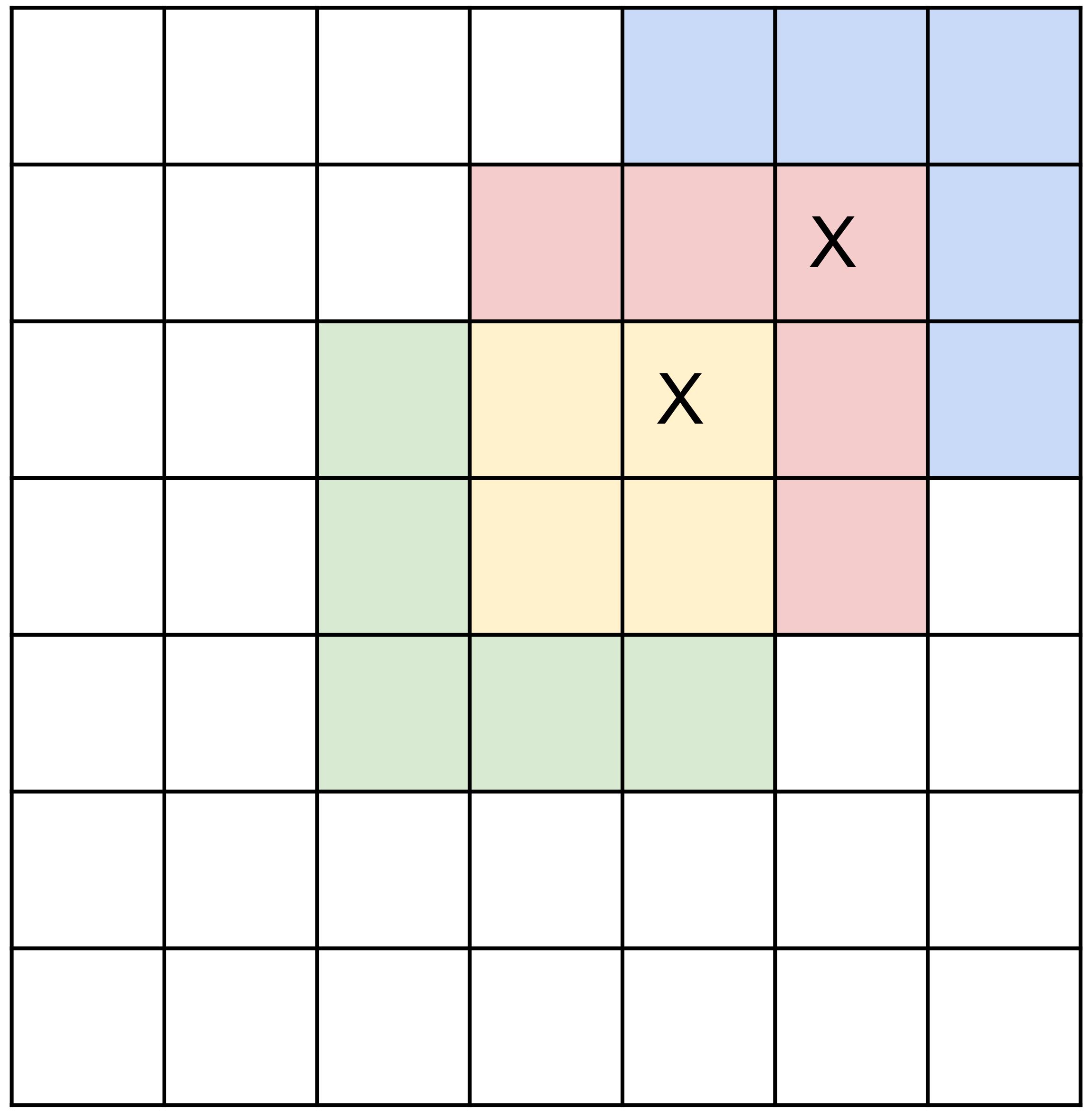
Answer: [5x5]

Slide: A. Karpathy / L. Fei Fei

The power of small filters

Suppose we stack three CONV layers with receptive filed size 3x3

Q: What region of input does each neuron in 3rd CONV see?



Answer: [7x7]

Slide: A. Karpathy / L. Fei Fei

The power of small filters

Suppose input has depth C & we want output depth C as well.

1x CONV with 7x7 filters

Number of weights:

C*(7*7*C)
= **49 C^2**

3x CONV with 3x3 filters

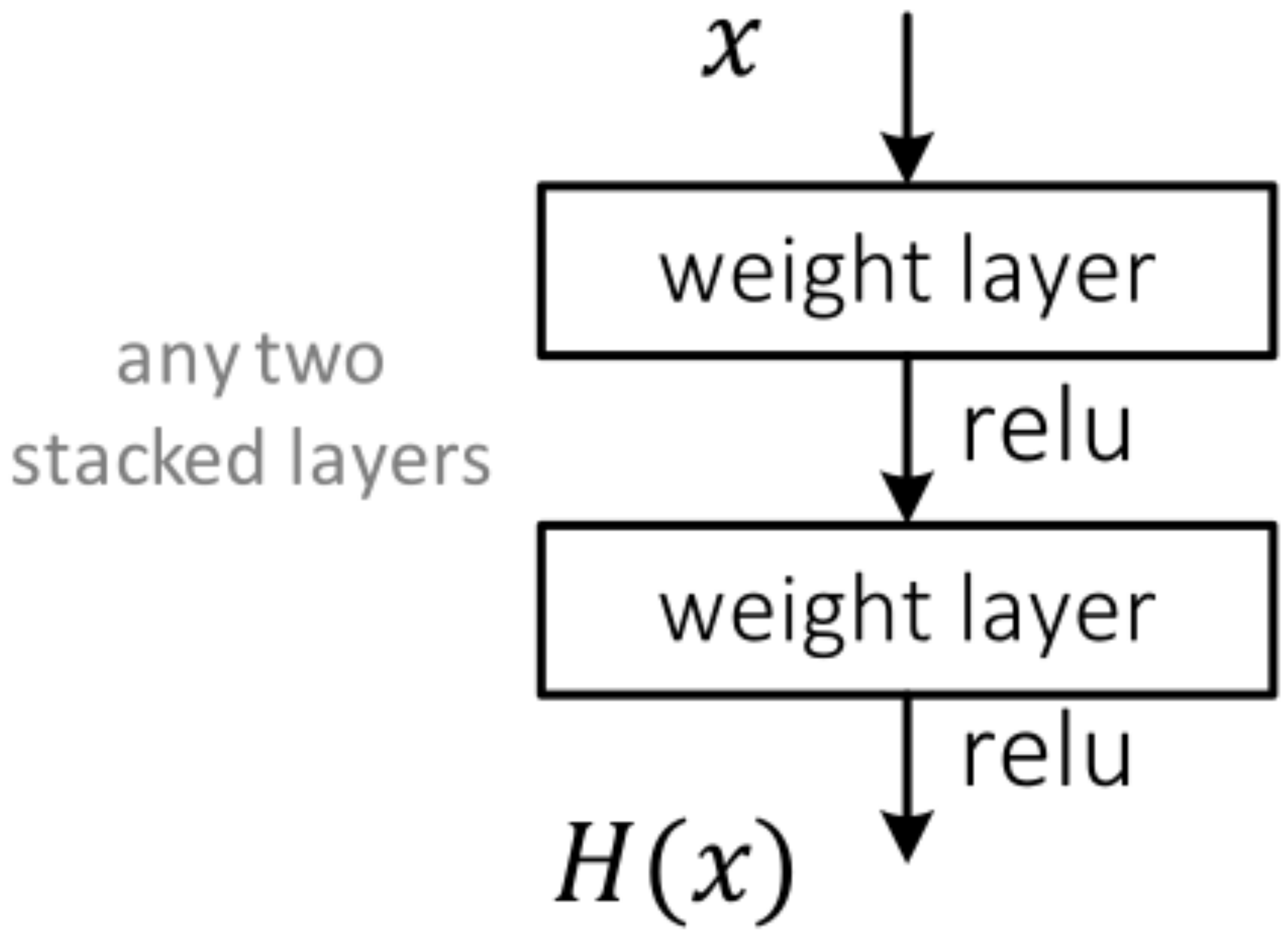
Number of weights:

C*(3*3*C) + C*(3*3*C) + C*(3*3*C)
= 3 * 9 * C^2
= **27 C^2**

Slide: A. Karpathy / L. Fei Fei

Residual networks [ResNets]

Plain net



$H(x)$ is any desired mapping,

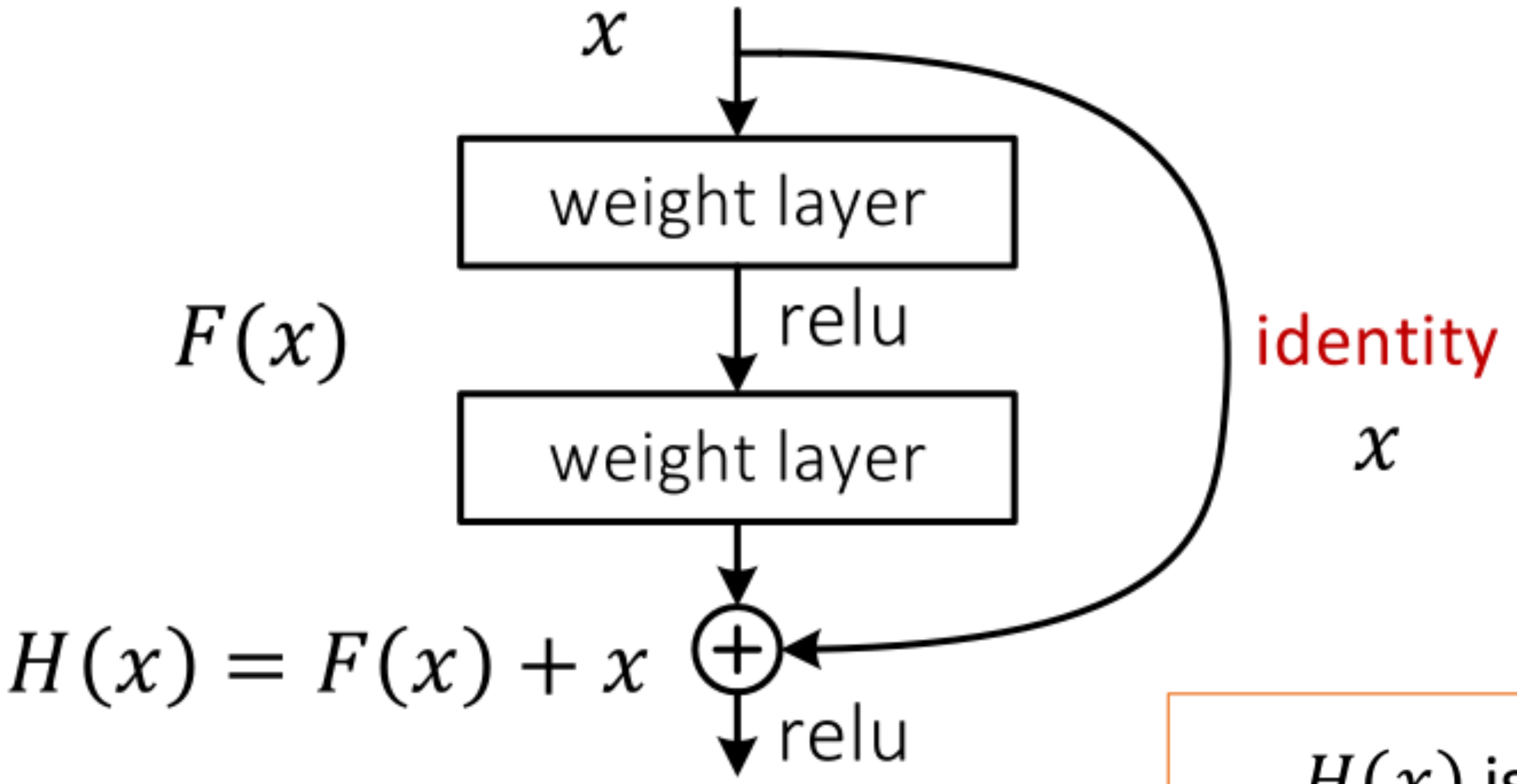
hope the 2 weight layers fit $H(x)$

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Slide: K. He

Residual networks [ResNets]

Residual net



$H(x)$ is any desired mapping,

~~hope the 2 weight layers fit $H(x)$~~

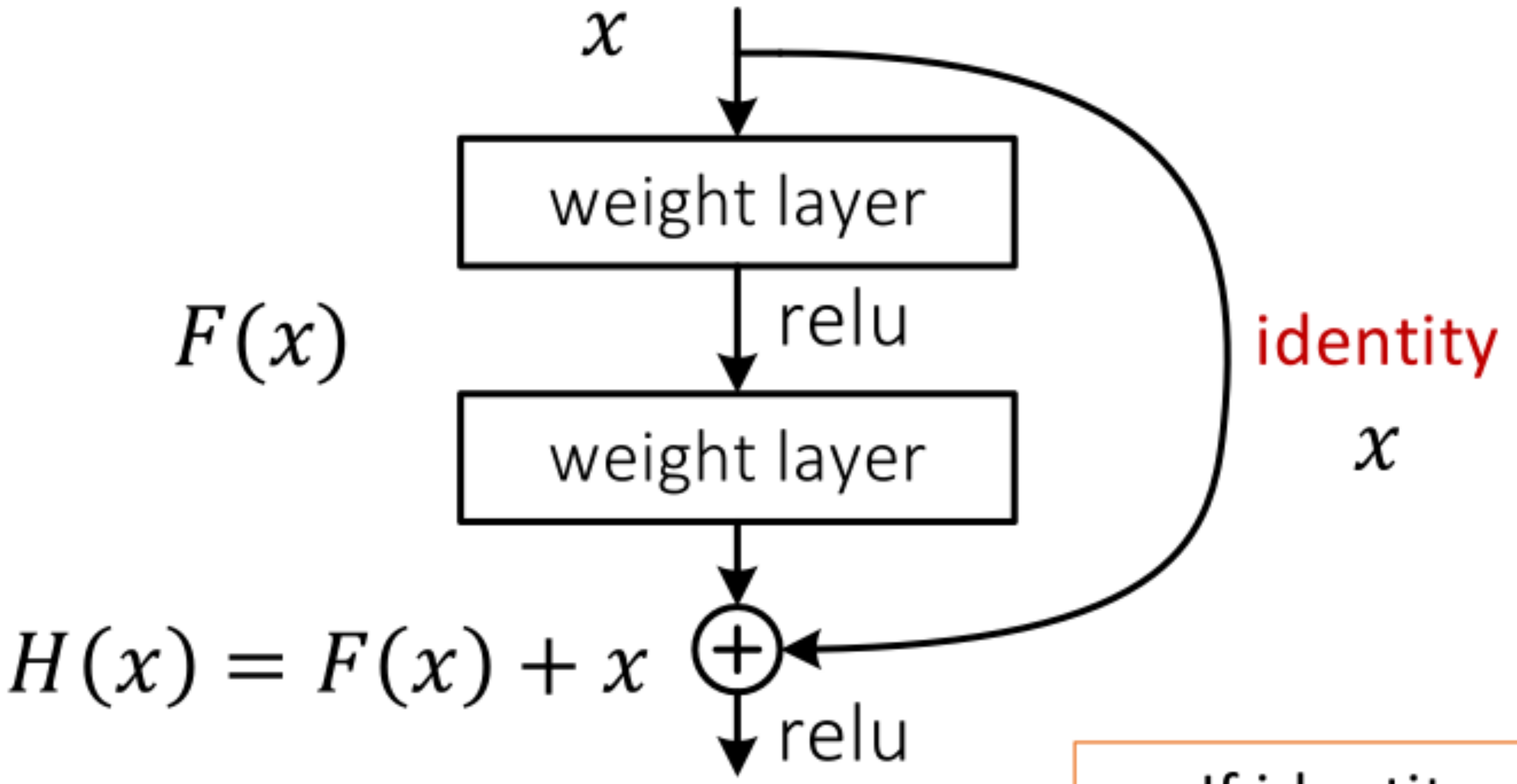
hope the 2 weight layers fit $F(x)$

let $H(x) = F(x) + x$

Slide: K. He

Residual networks [ResNets]

$F(x)$ is a residual mapping w.r.t. identity



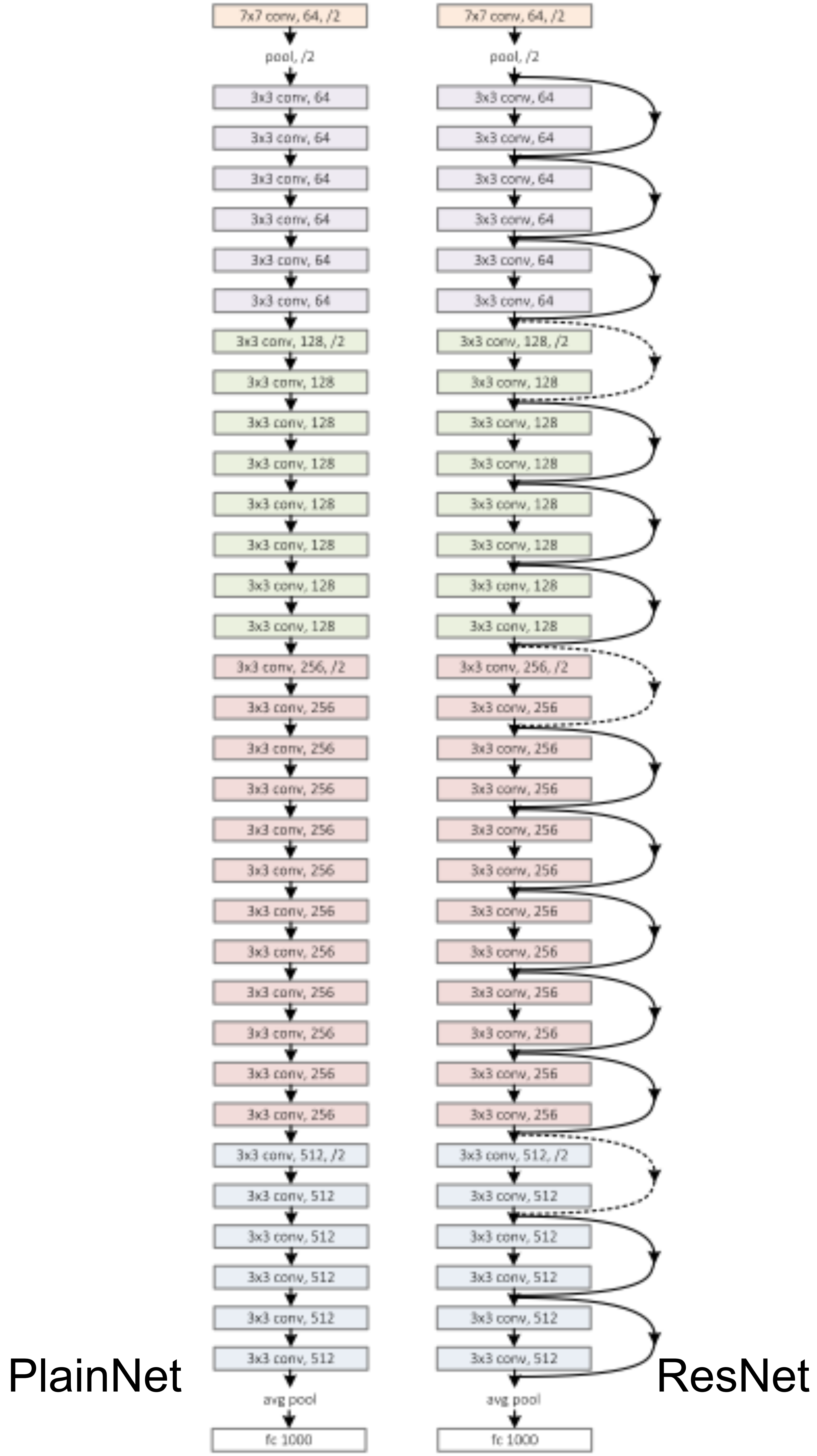
- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations

Slide: K. He

Network design

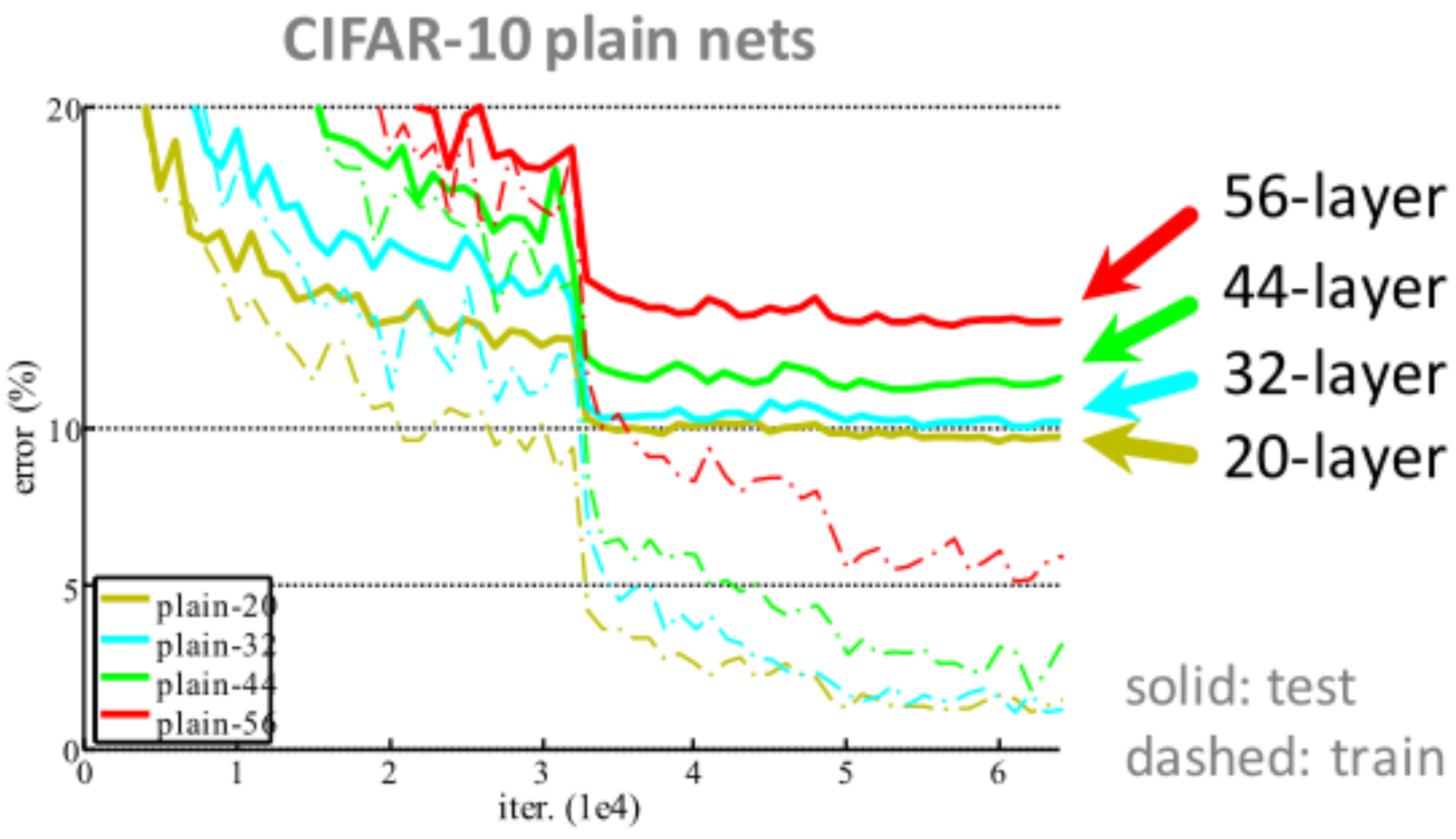
Basic design (vgg-style)

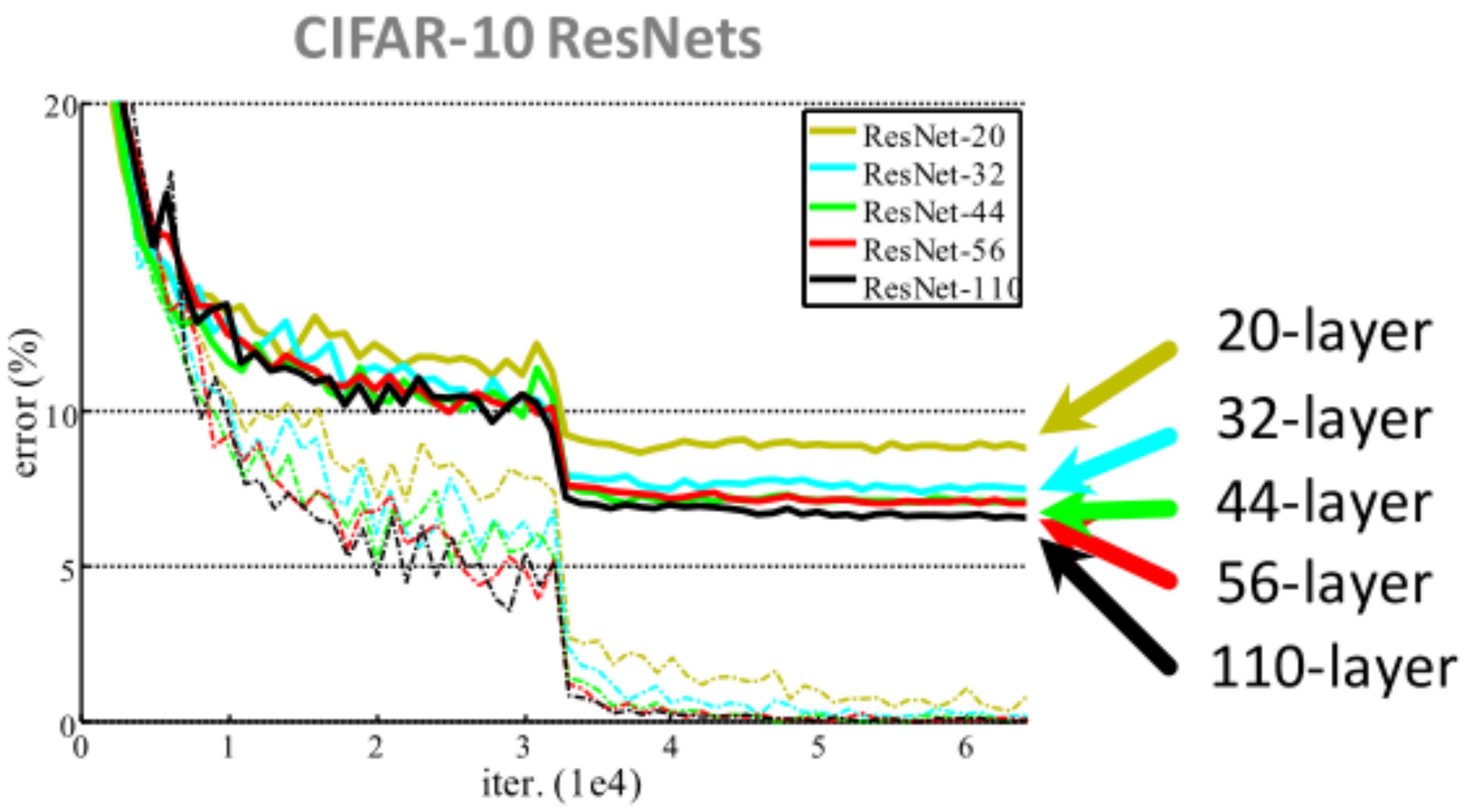
- almost all 3x3 conv
- Spatial size /2 => # filters x2
- Simple design, just deep
- No fully connected layers
- No dropout



Slide: K. He

CIFAR-10 experiments





- Deep ResNets can be trained without difficulties
- Deeper ResNets have lower training error and lower test error

Slide: K. He

Agenda

- **1. Recap: Bag of Visual Words, Analogy with NNs**

- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs

- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures

- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers

- **5. Beyond classification**

Do we need convolutions?

Attention Is All You Need

Ashish Vaswani[*] Google Brain avaswani@google.com

Noam Shazeer[*] Google Brain noam@google.com

Niki Parmar[*] Google Research nikip@google.com

Jakob Uszkoreit[*] Google Research usz@google.com

Llion Jones[*] Google Research llion@google.com

Aidan N. Gomez[*] [†] University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser[*] Google Brain lukaszkaiser@google.com

Illia Polosukhin[*] [‡] illia.polosukhin@gmail.com

NeurIPS 2017

Published as a conference paper at ICLR 2021

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*], **Xiaohua Zhai**[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby**[*,†]

[*]equal technical contribution, [†]equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

ICLR 2021

MLP-Mixer: An all-MLP Architecture for Vision

Ilya Tolstikhin[*], **Neil Houlsby**[*], **Alexander Kolesnikov**[*], **Lucas Beyer**[*], **Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, Alexey Dosovitskiy**

[*]equal contribution

Google Research, Brain Team

{tolstikhin, neilhoulsby, akolesnikov, lbeyer, xzhai, unterthiner, jessicayung[†], andstein, keysers, usz, lucic, adosovitskiy}@google.com

NeurIPS 2021

Under review as a conference paper at ICLR 2022

~~CONVOLUTIONS~~ ~~ATTENTION~~ ~~MLPS~~ PATCHES ARE ALL YOU NEED?

Anonymous authors

Paper under double-blind review

arXiv 2022

https://github.com/KentoNishi/awesome-all-you-need-papers

Recent Hype#1: Transformers

- Transformers = neural network architectures stacking "attention" layers[1]
- Initially successful for natural language processing
- Then applied to computer vision[2]. Better performance than CNNs <u>given enough data</u>.
- The hype still continues today.
- What is attention?

[1] Vaswani et al. "**Attention is all you need",** NeurIPS 2017.
[2] Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale", ICLR 2021.

Attention & Transformer

- **Basic transformer model**
- Image transformers

Attention

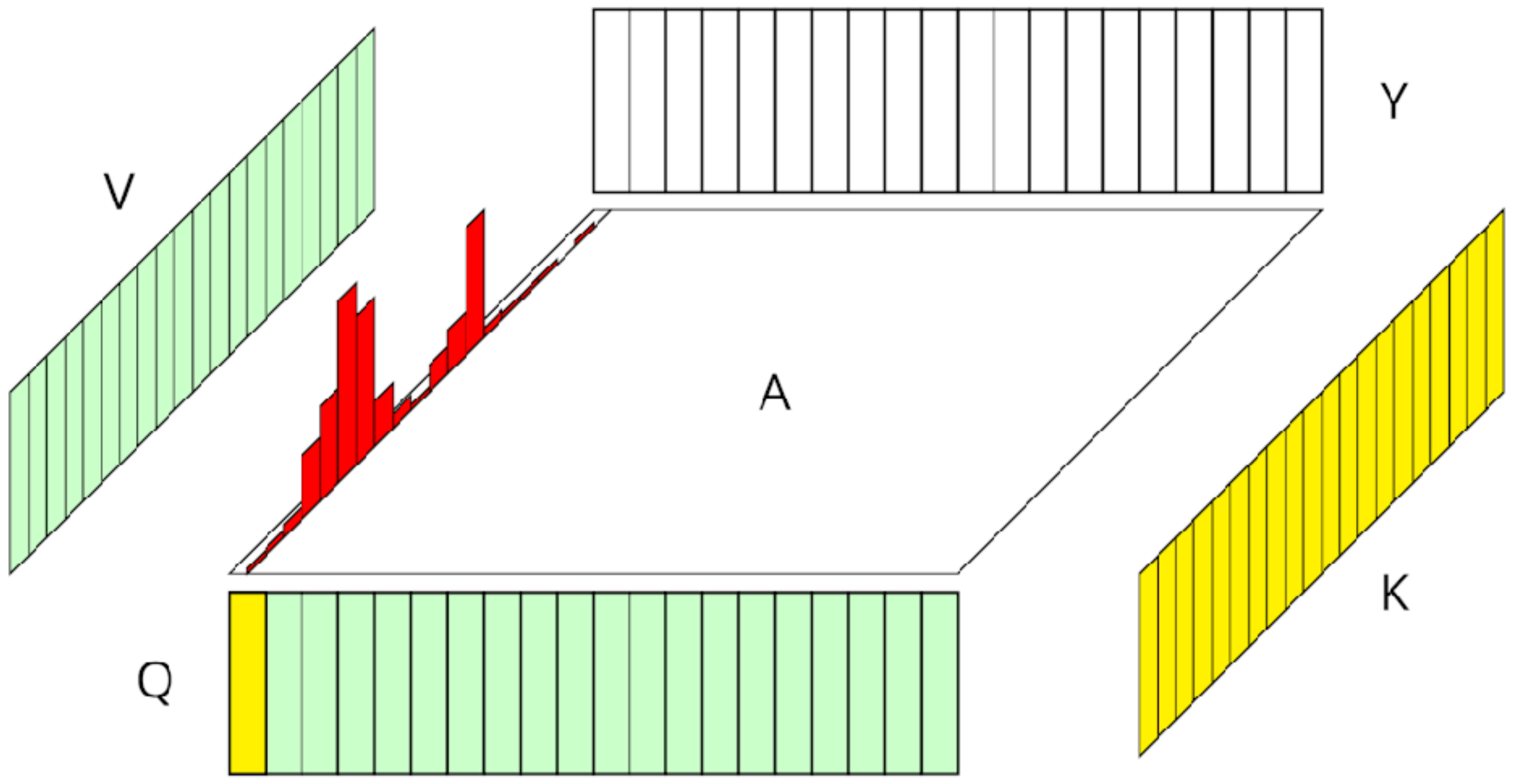
- Motivation: sequence-to-sequence models

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}})\mathbf{V}$$

$$A_{i,j} = \text{softmax}\left(\frac{Q_i \cdot K_j}{\sqrt{d}}\right) \qquad Y_i = \sum_j A_{i,j} V_j$$

Given a query sequence Q, a key sequence K, and a value sequence V, compute an attention matrix A by matching Qs to Ks, and weight V with it to get the sequence Y.

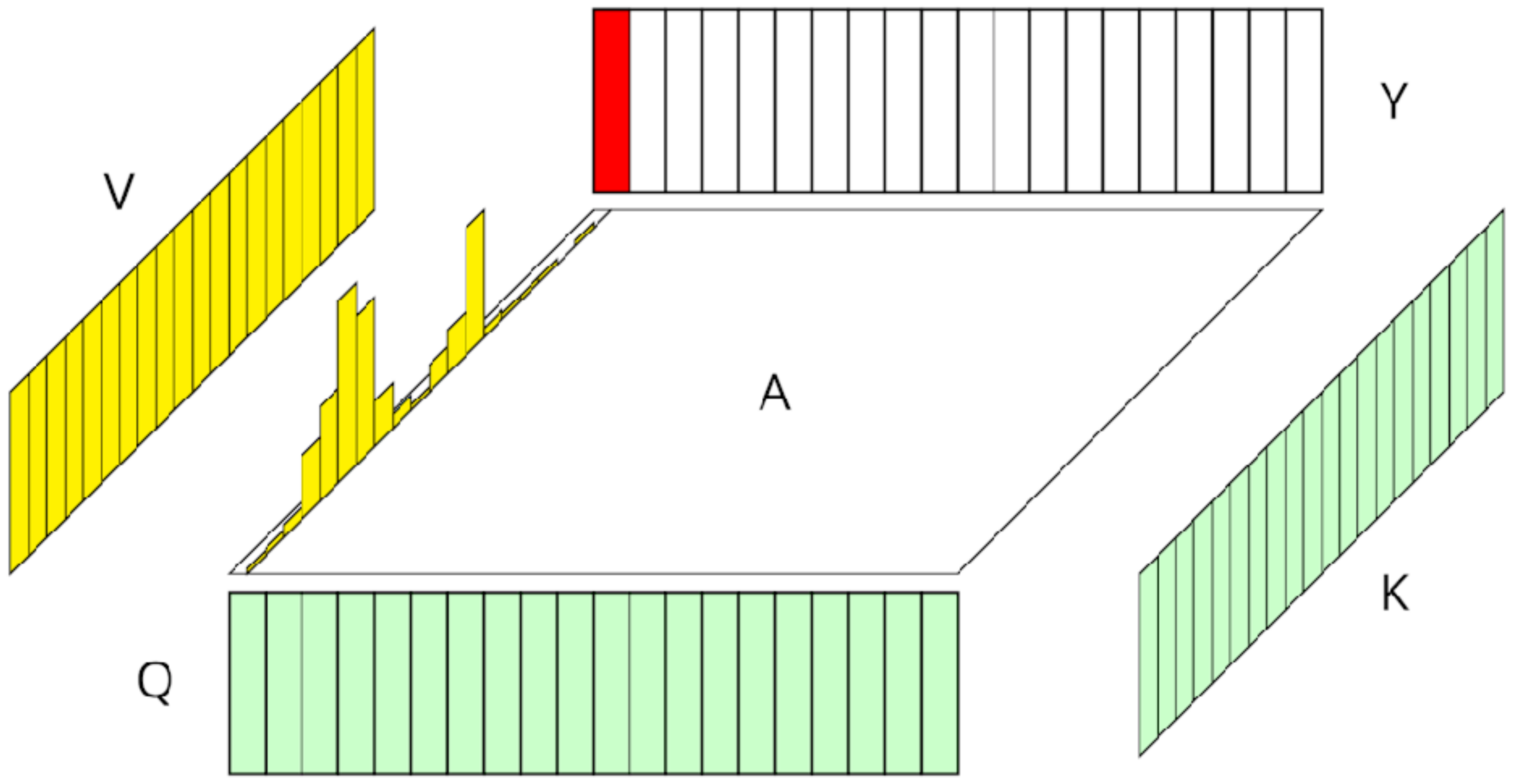
$$A_{i,j} = \text{softmax}\left(\frac{Q_i \cdot K_j}{\sqrt{d}}\right)$$



Attention mechanisms

Given a query sequence Q, a key sequence K, and a value sequence V, compute an attention matrix A by matching Qs to Ks, and weight V with it to get the sequence Y.

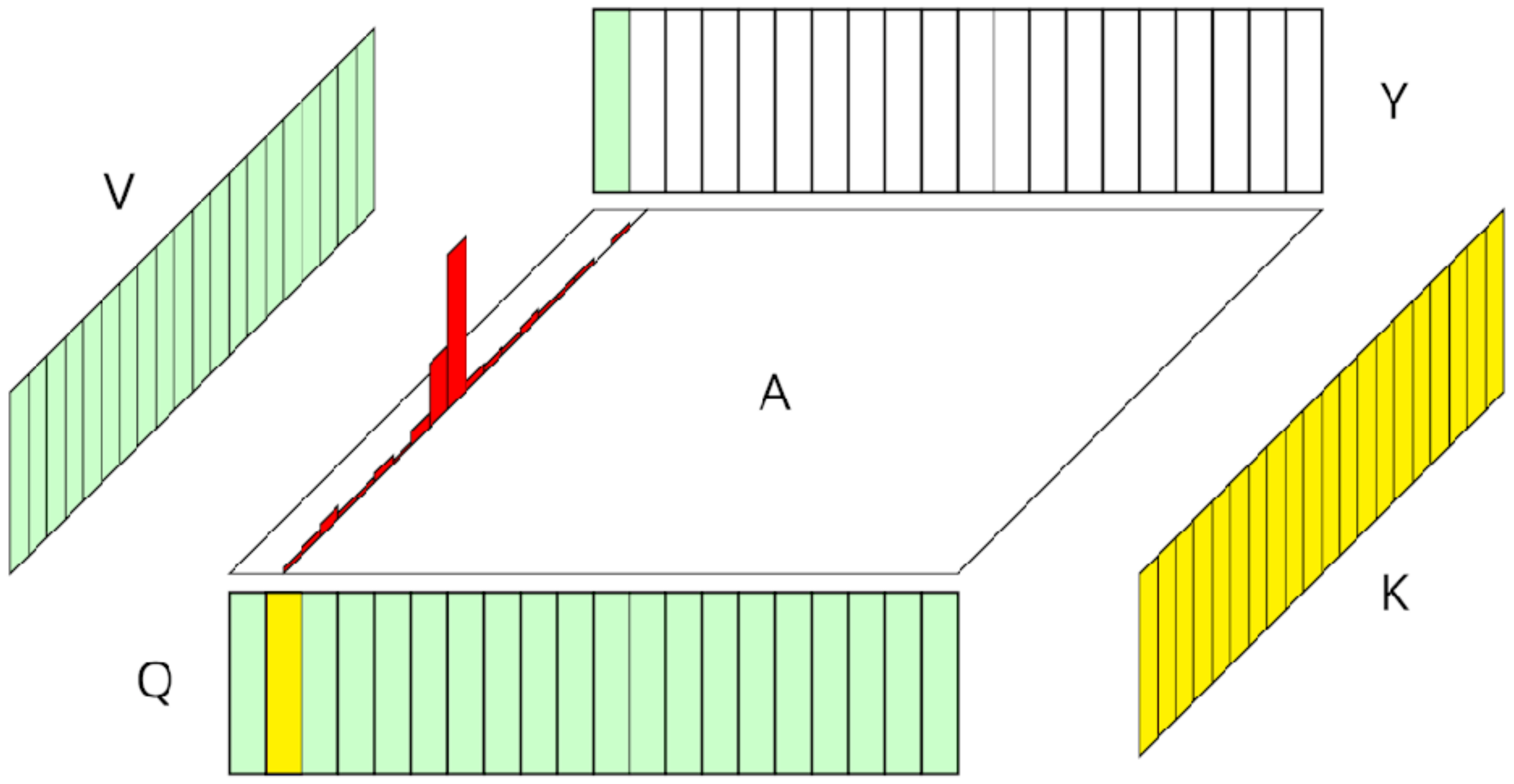
$$Y_i = \sum_j A_{i,j} V_j$$



Attention mechanisms

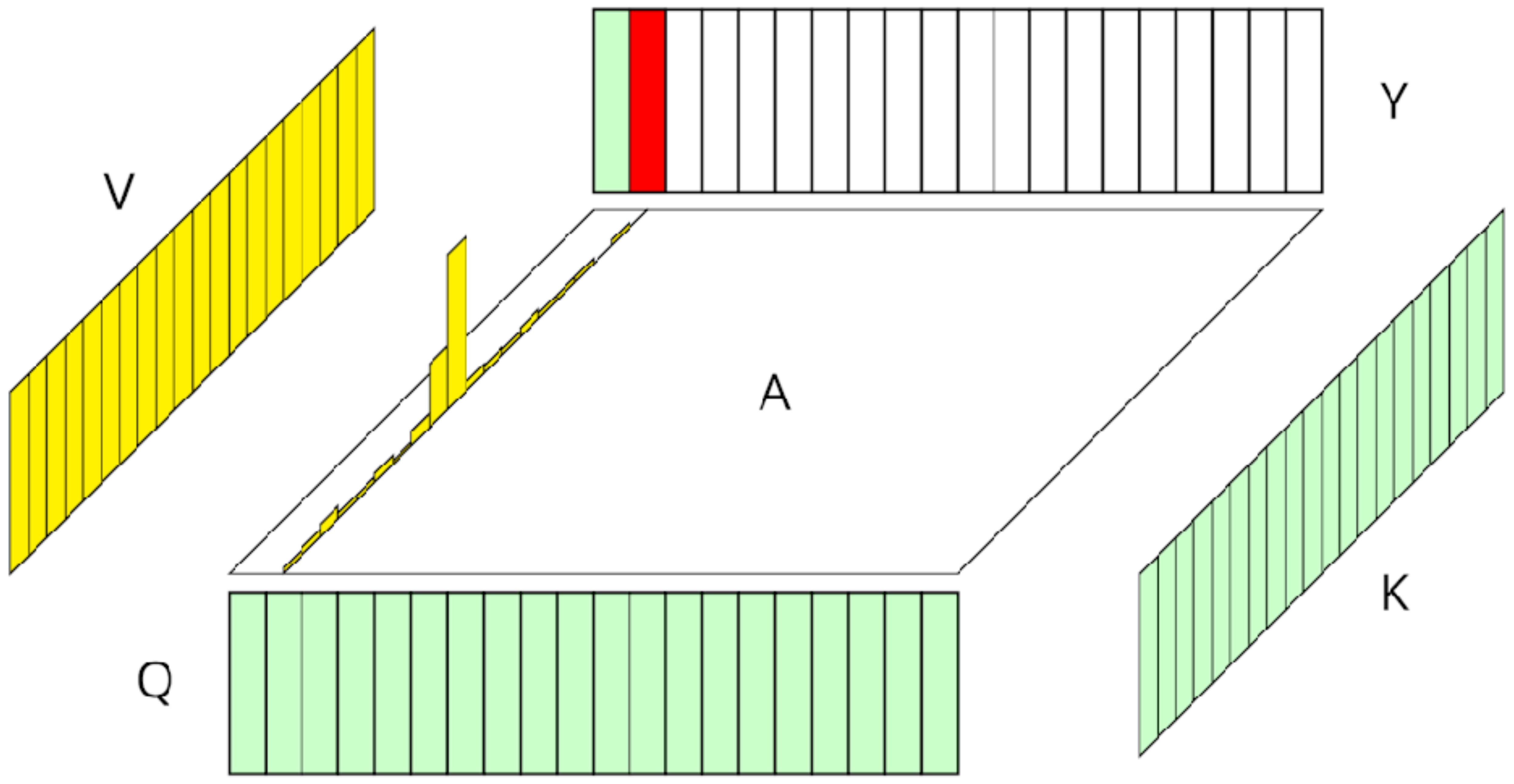
Given a query sequence Q, a key sequence K, and a value sequence V, compute an attention matrix A by matching Qs to Ks, and weight V with it to get the sequence Y.

$$A_{i,j} = \text{softmax}\left(\frac{Q_i \cdot K_j}{\sqrt{d}}\right)$$



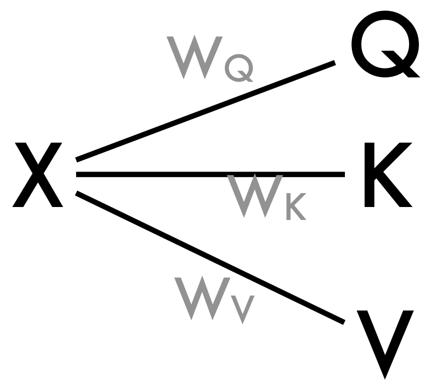
Given a query sequence Q, a key sequence K, and a value sequence V, compute an attention matrix A by matching Qs to Ks, and weight V with it to get the sequence Y.

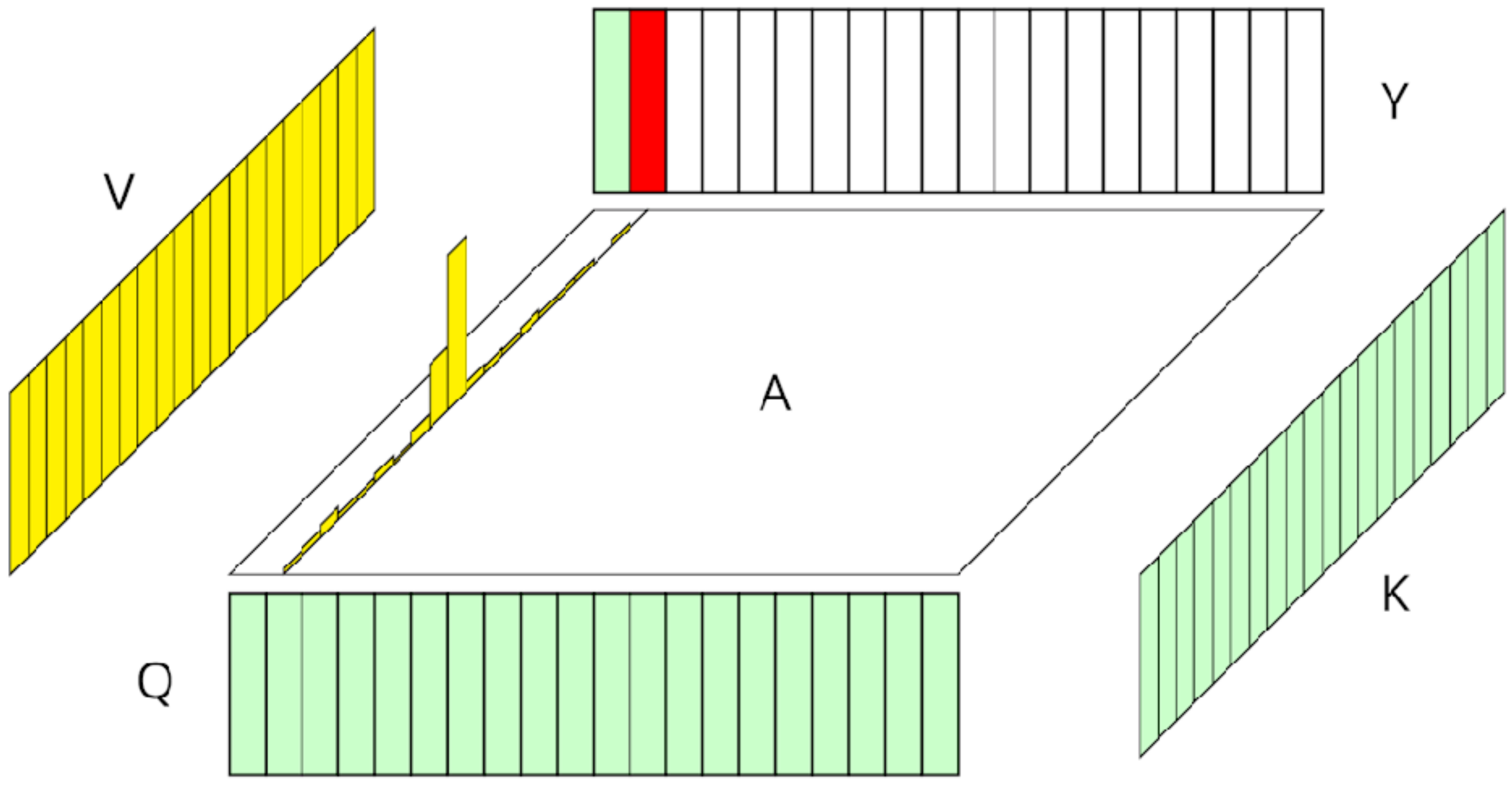
$$Y_i = \sum_j A_{i,j} V_j$$



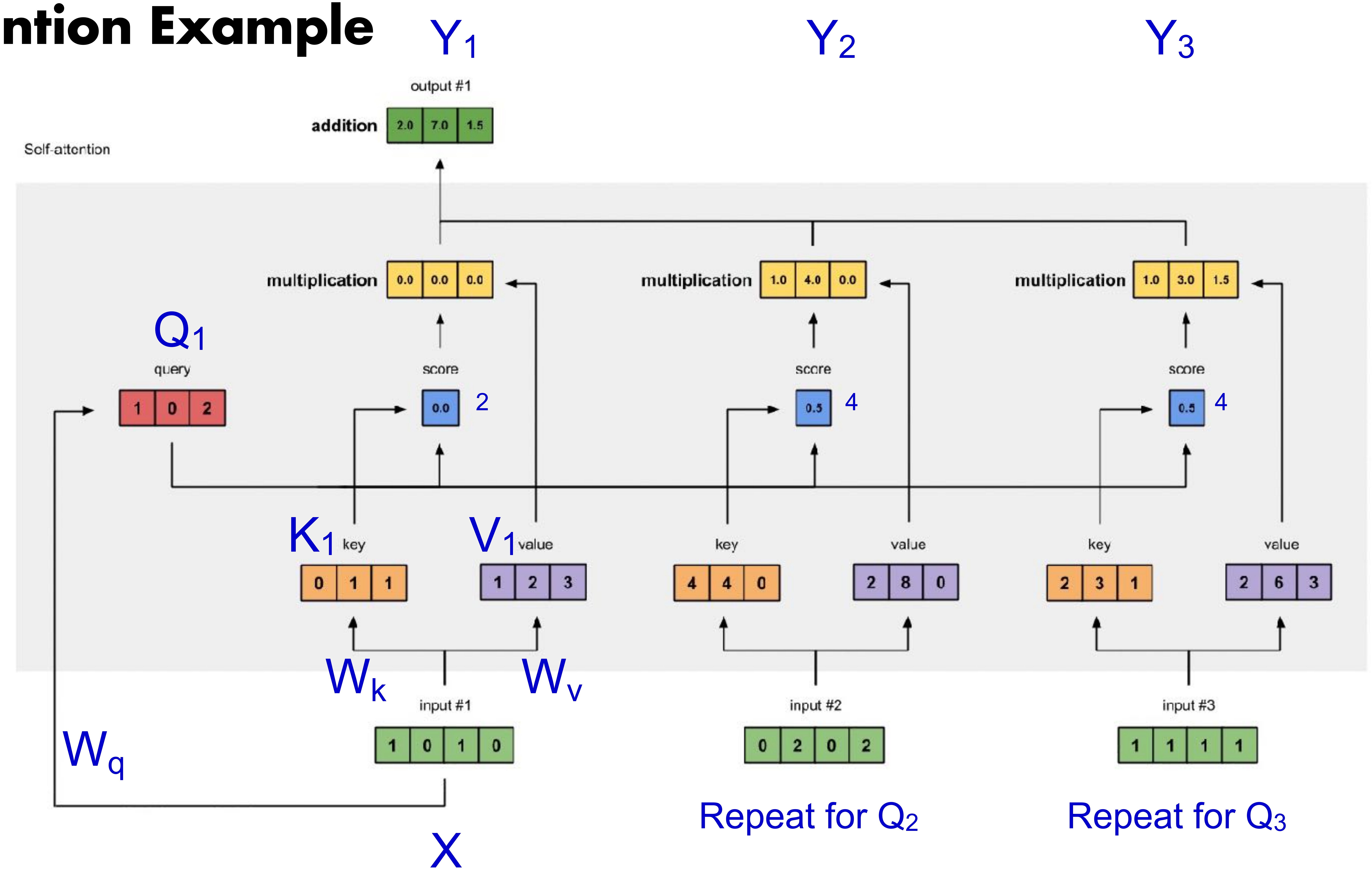
- **Q**uery and **K**ey dimensionalities are the same.
- **V**alue dimensionality may be different.
- Output dimensionality will be the same as **V**alue.

In "self-attention", (Q, K, V) obtained from the same input, linearly projected three times.





Self-Attention Example



$$A_{i,j} = \text{softmax}\left(\frac{Q_i \cdot K_j}{\sqrt{d}}\right) \qquad Y_i = \sum_j A_{i,j} V_j$$

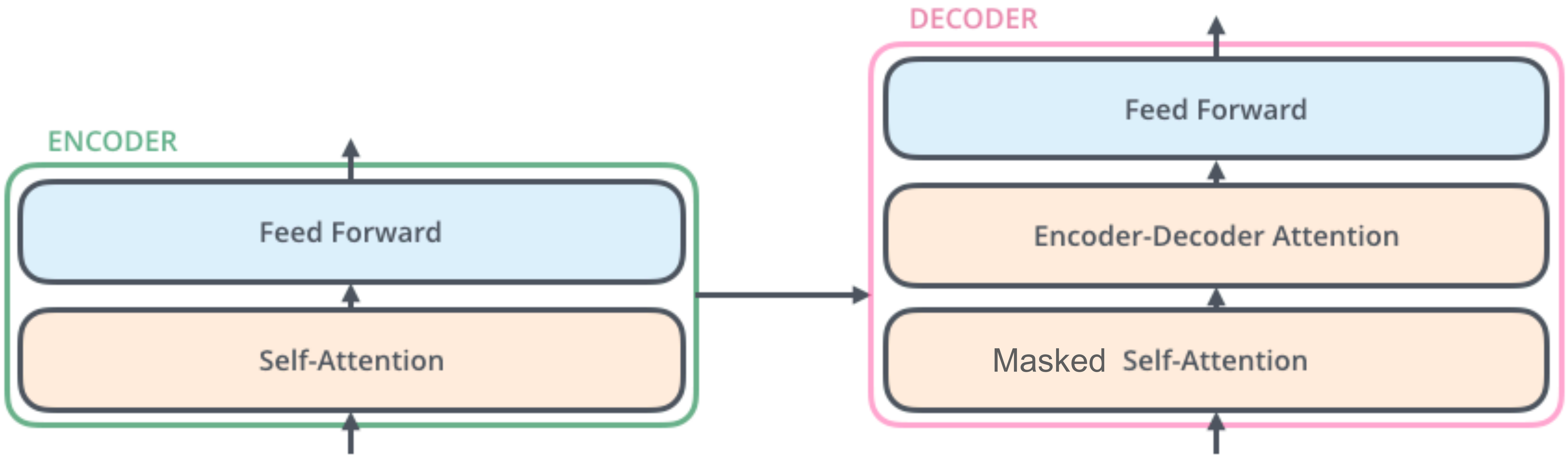
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Basic transformer model

- Sequence-to-sequence architecture using only point-wise processing and attention (no recurrent units or convolutions)

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

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



NLP application: Machine Translation

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

Key-Value-Query attention model

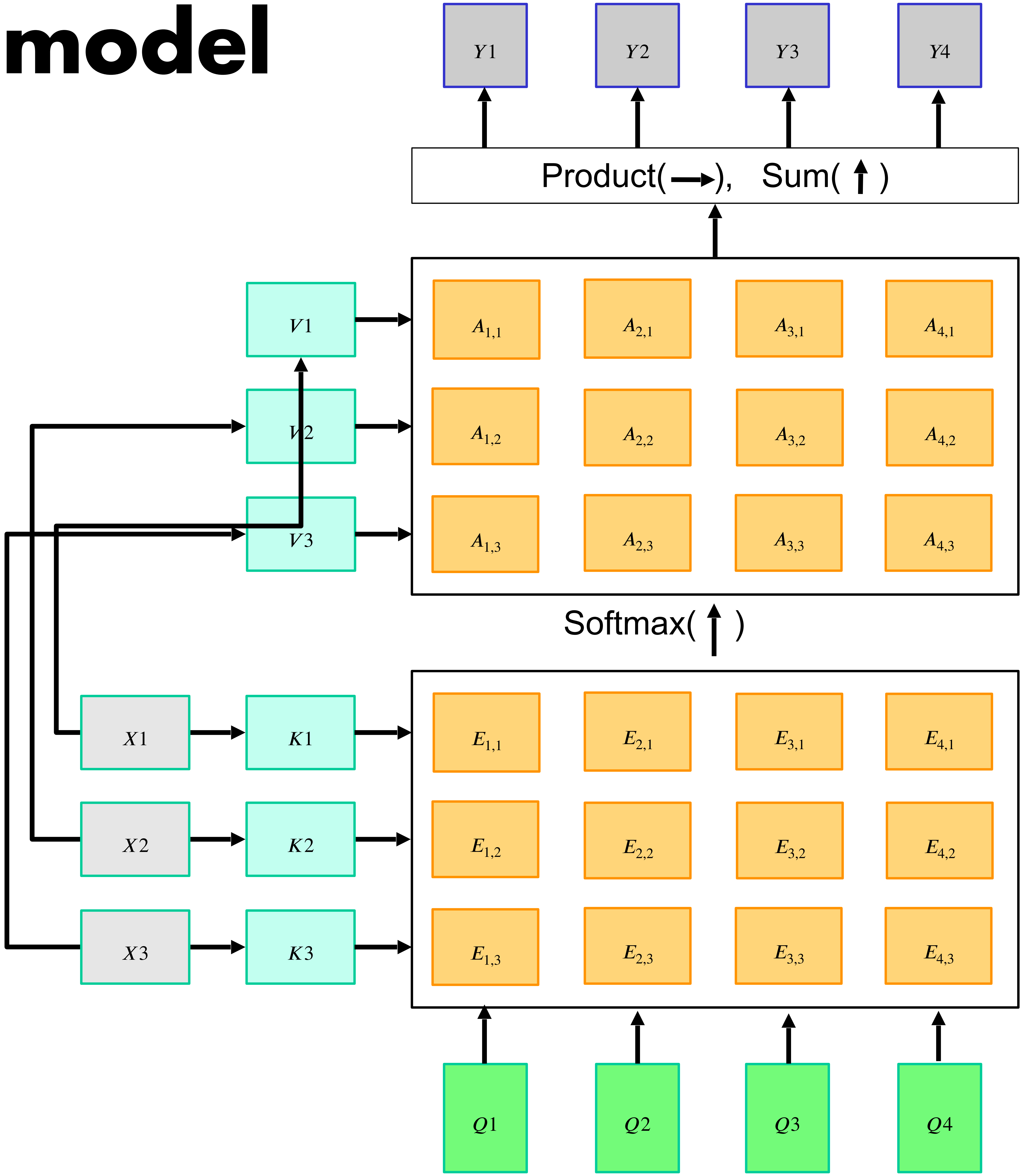
- Key vectors: $K = XW_K$
- Value Vectors: $V = XW_V$
- Query vectors
- Similarities: *scaled dot-product attention*

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

(D is the dimensionality of the keys)

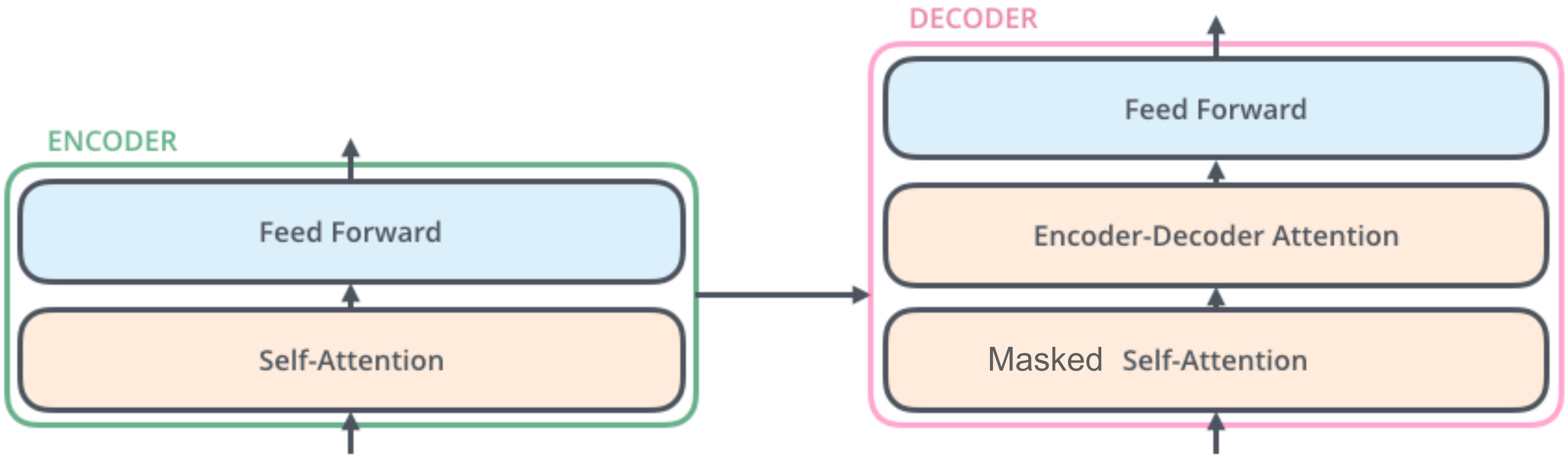
- Attn. weights: $A = \mathrm{softmax}(E, \dim = 1)$
- Output vectors:

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



Adapted from J. Johnson

Attention mechanisms



- **Encoder self-attention:** queries, keys, and values come from previous layer of encoder
- **Decoder self-attention:** values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention:** queries come from previous decoder layer, keys and values come from output of encoder

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”

Image source

Self-attention layer

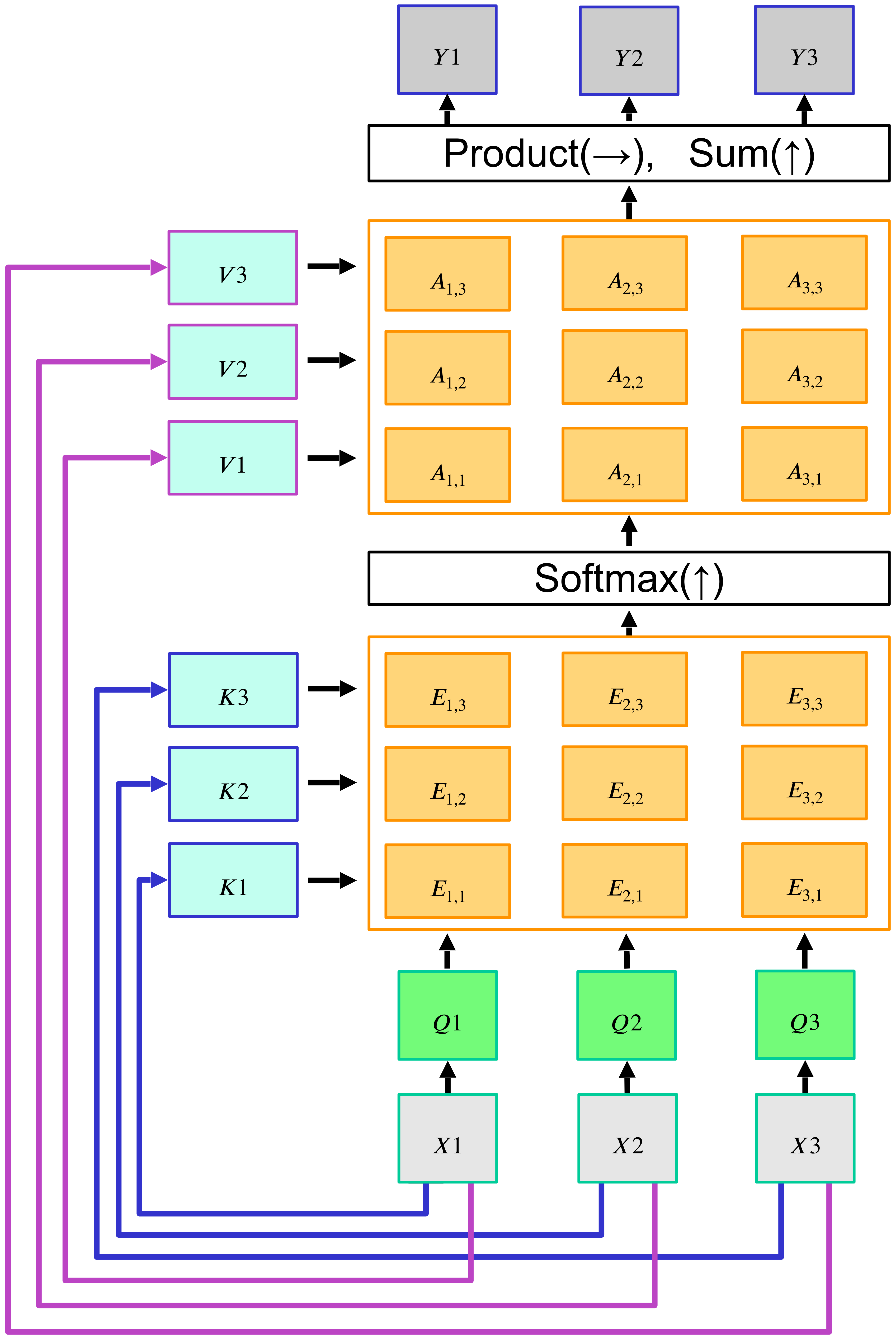
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Similarities: *scaled dot-product attention*

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

(D is the dimensionality of the keys)

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

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

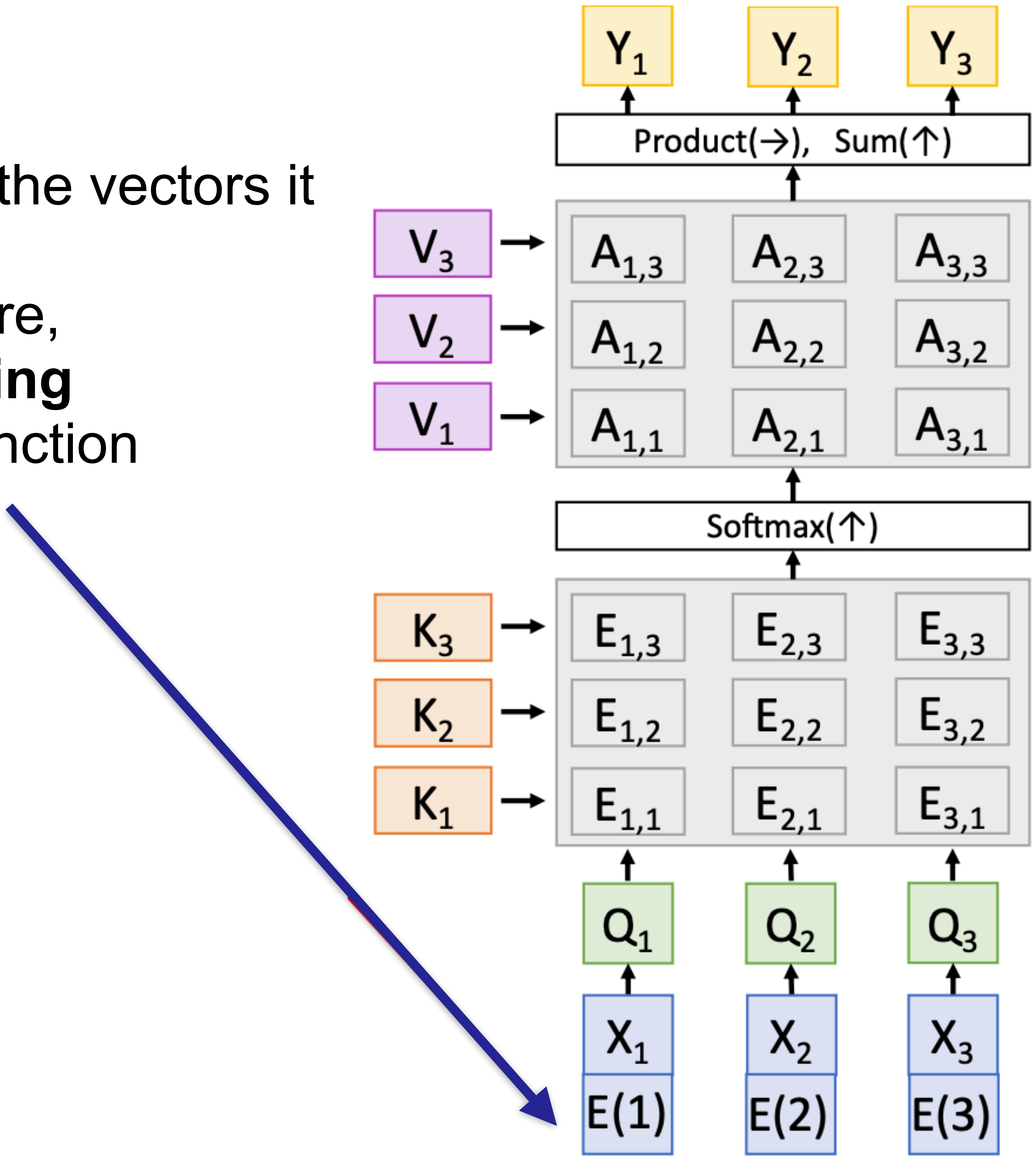


One query per input vector

Adapted from J. Johnson

Positional encoding

- Self attention doesn't "know" the order of the vectors it is processing!
- In order to make processing position-aware, concatenate input with **positional encoding**
- *E* can be learned lookup table, or fixed function

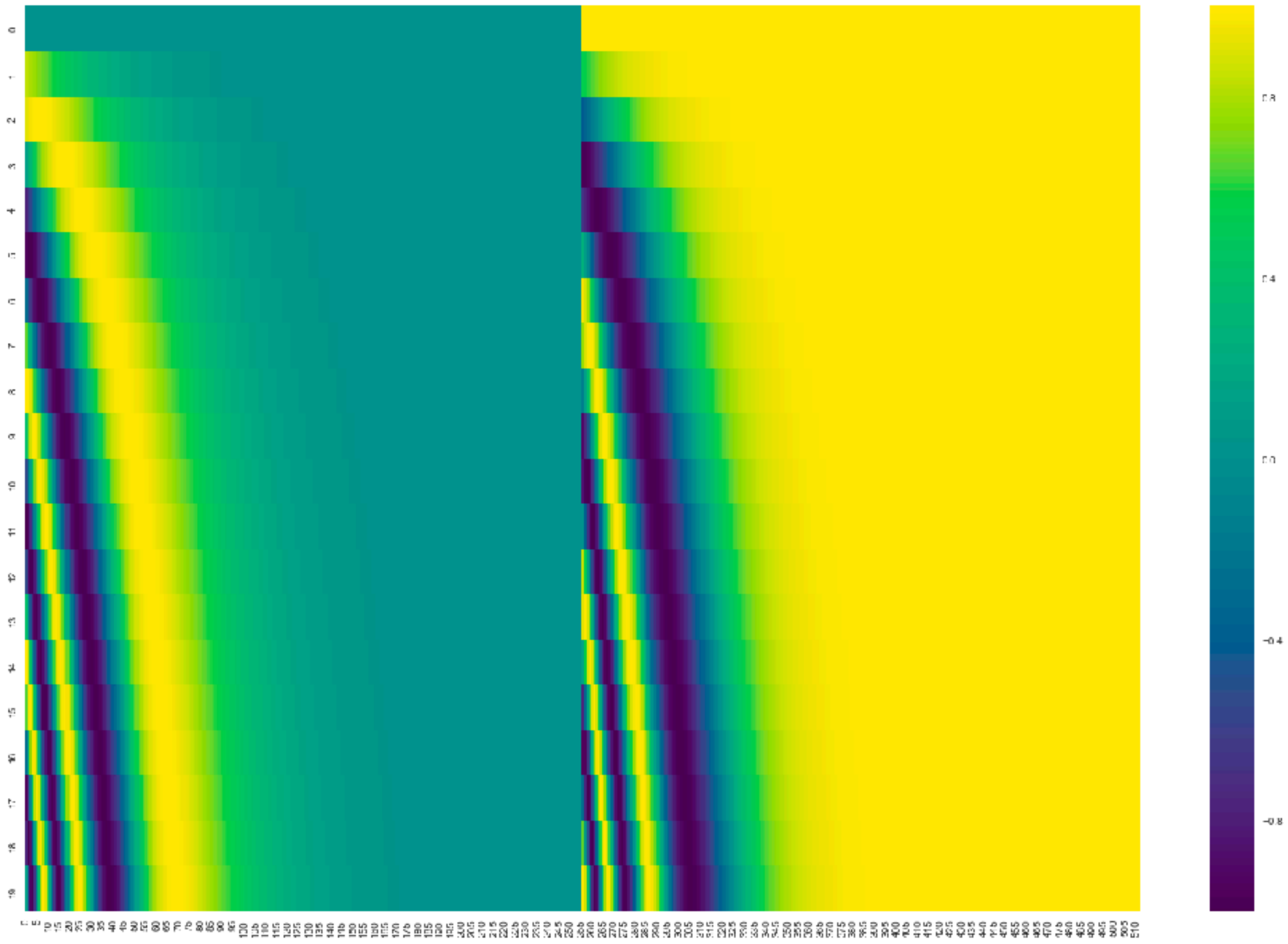


Adapted from J. Johnson

Positional encoding

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

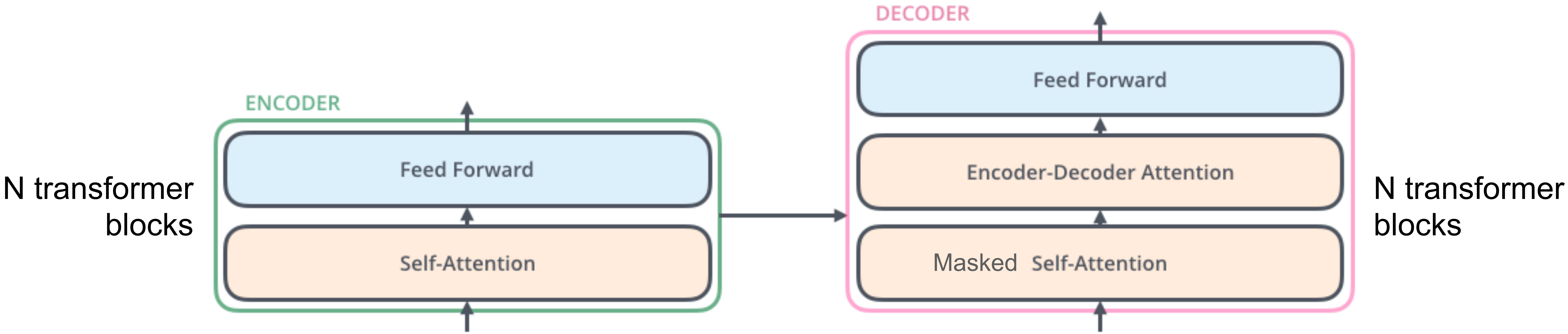
position



Embedding dimension

Image source

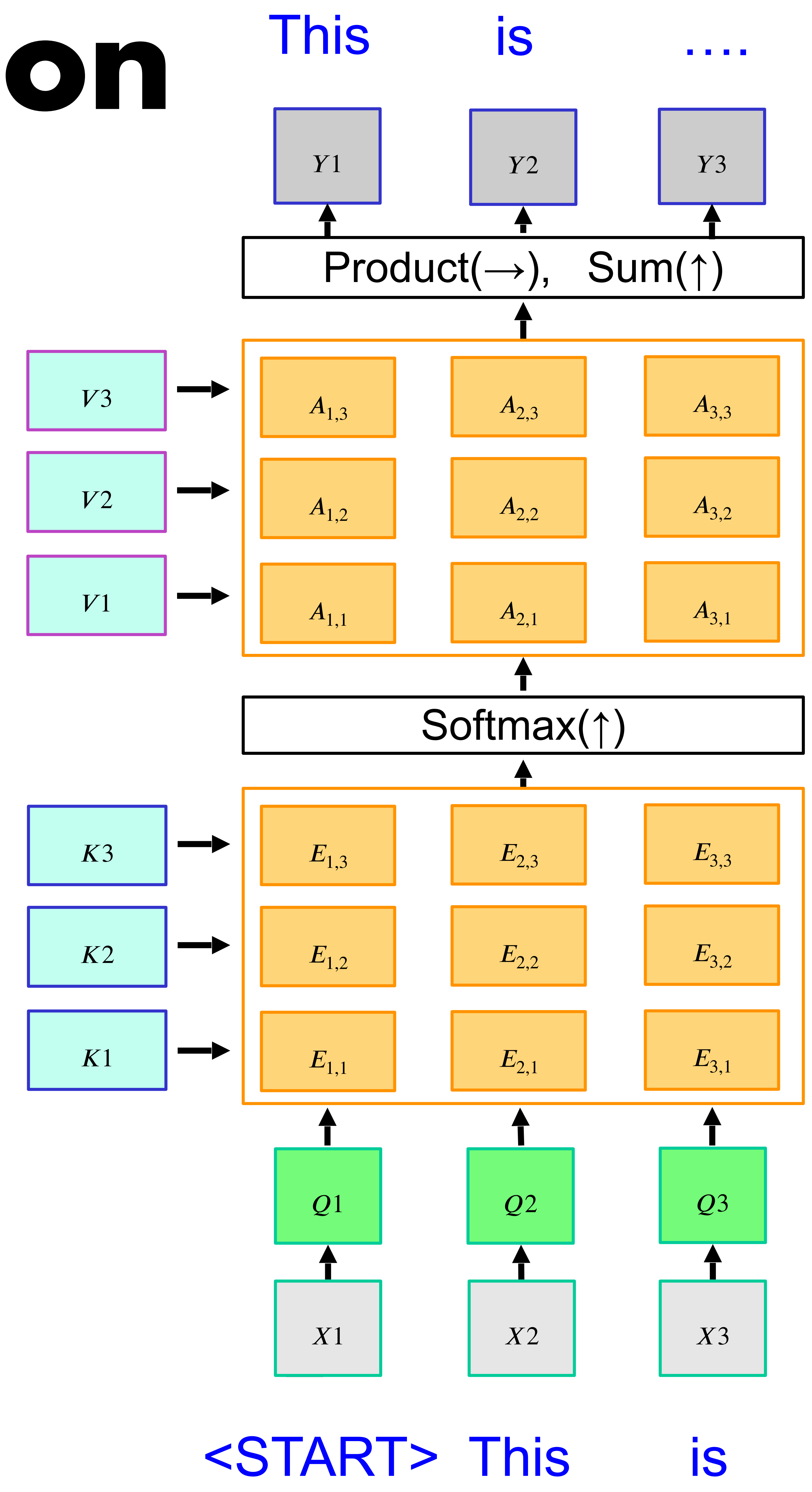
Attention mechanisms: Overview



- **Encoder self-attention:** queries, keys, and values come from previous layer of encoder
- **Decoder self-attention:** values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention:** queries come from previous decoder layer, keys and values come from output of encoder

Decoder: Masked self-attention

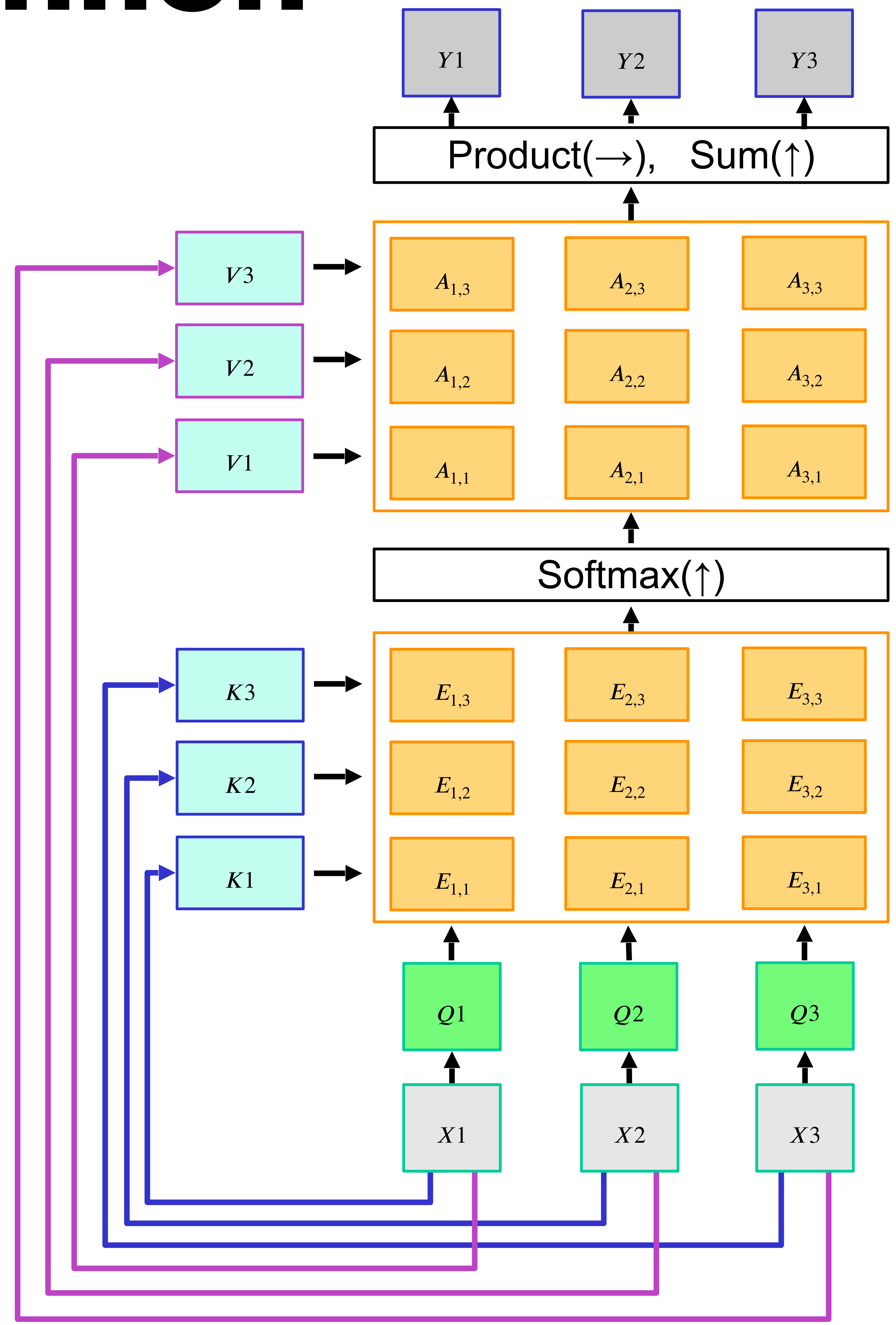
- The decoder should not "look ahead" in the output sequence



Adapted from [J. Johnson]

Decoder: Masked self-attention

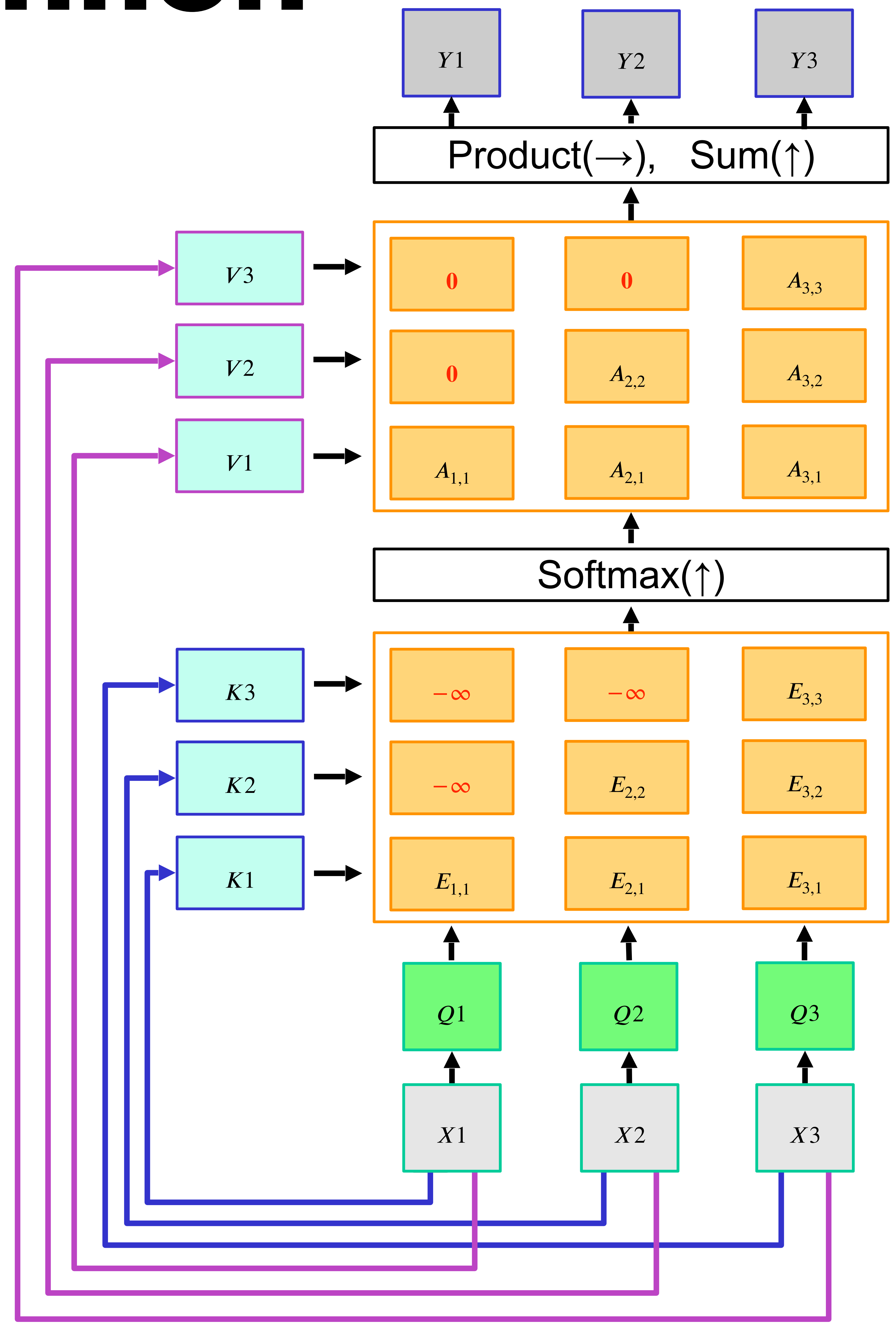
- The decoder should not "look ahead" in the output sequence



Adapted from [J. Johnson]

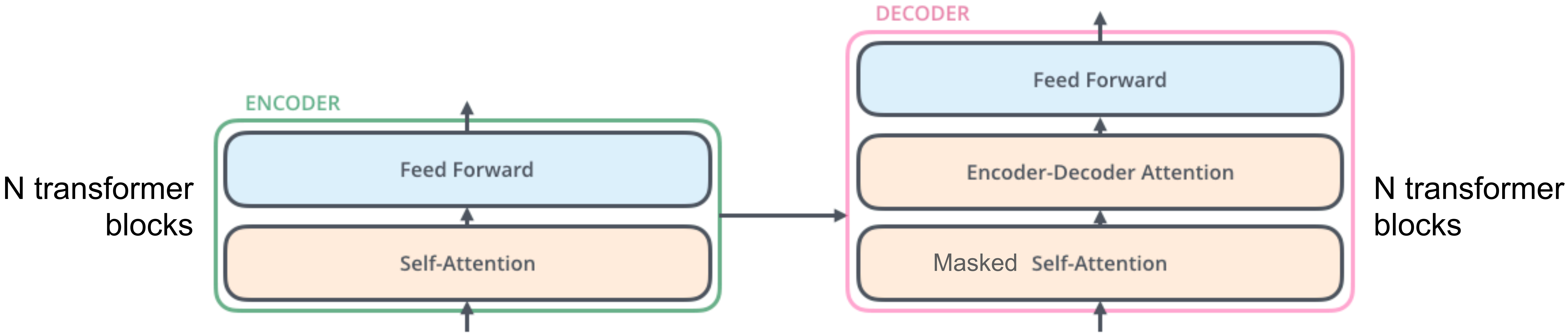
Decoder: Masked self-attention

- The decoder should not "look ahead" in the output sequence



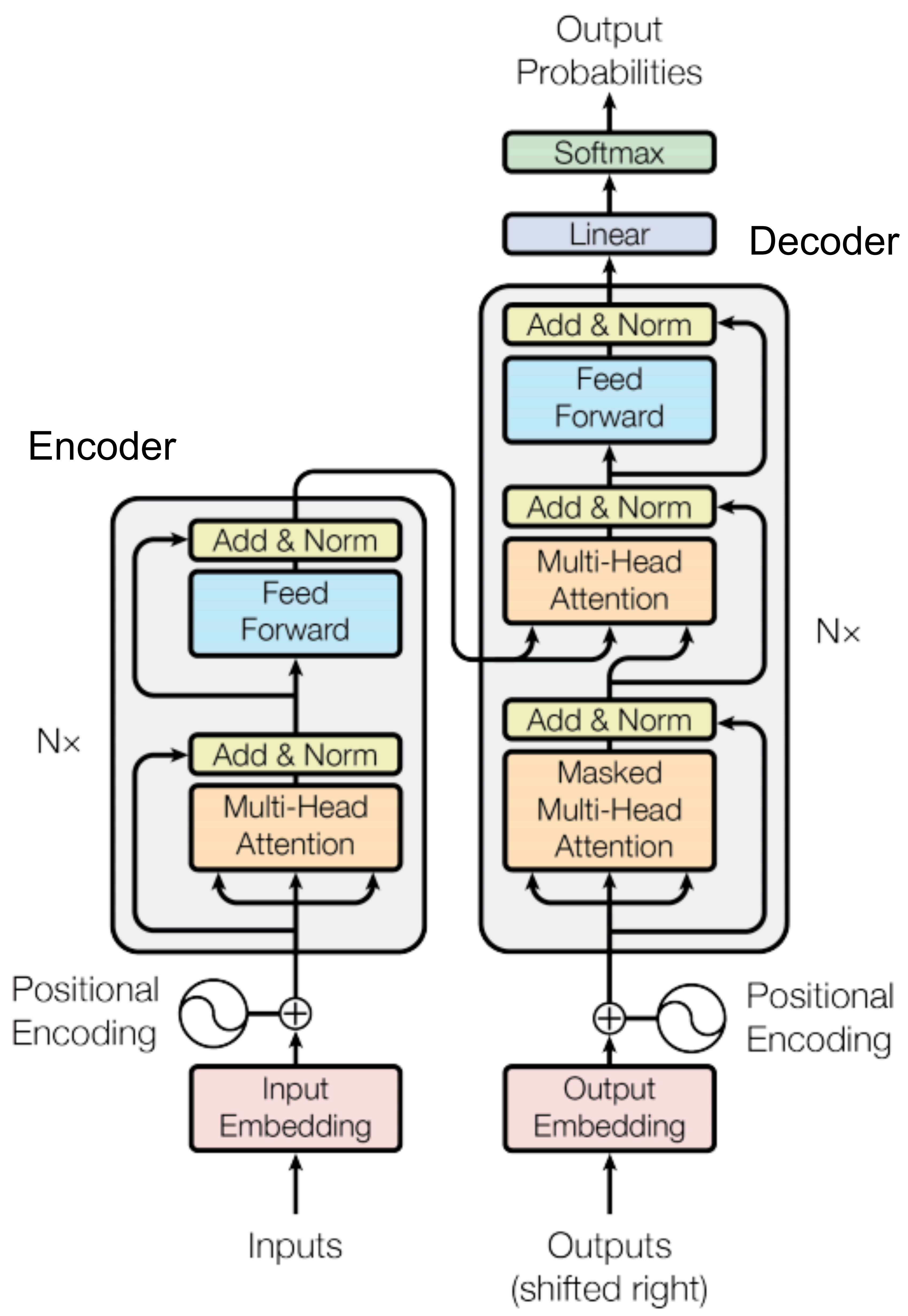
Adapted from [J. Johnson]

Attention mechanisms: Overview



- **Encoder self-attention:** queries, keys, and values come from previous layer of encoder
- **Decoder self-attention:** values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention:** queries come from previous decoder layer, keys and values come from output of encoder

Transformer architecture: Details

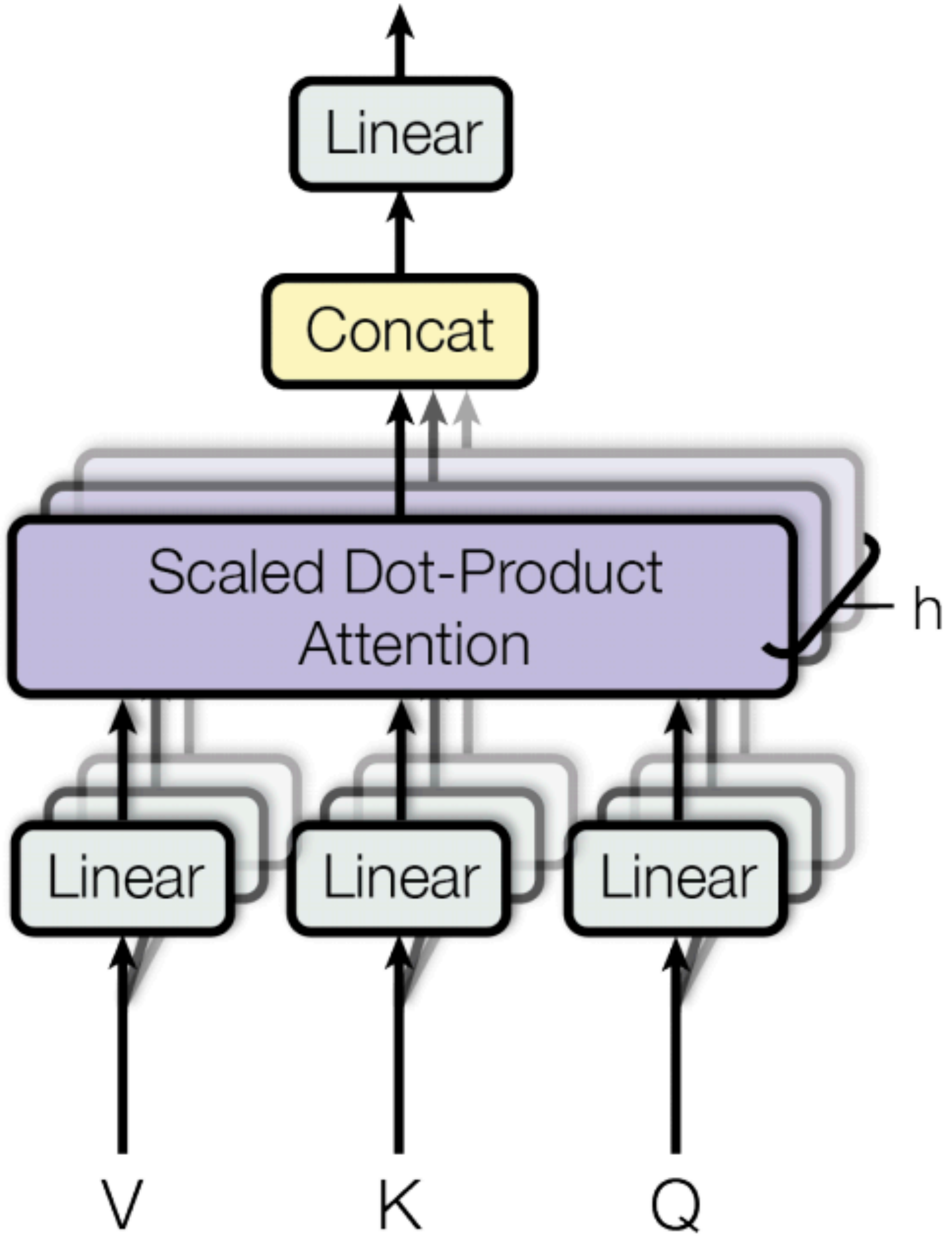


A. Vaswani et al., [Attention is all you need](), NeurIPS 2017

Multi-head attention

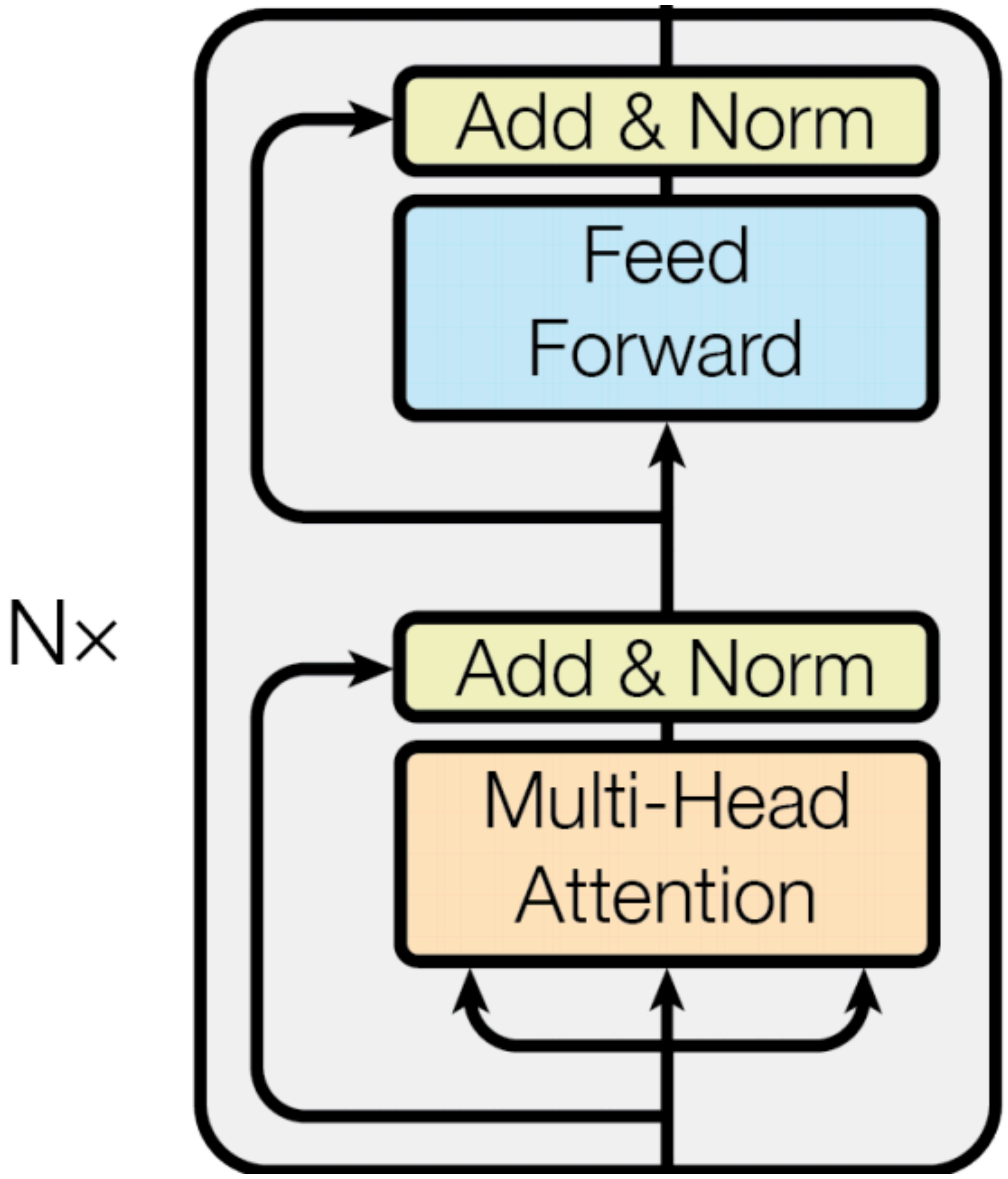
- Run h attention models in parallel on top of different linearly projected versions of Q, K, V; concatenate and linearly project the results

- Intuition: enables model to attend to different kinds of information at different positions (see visualization tool)

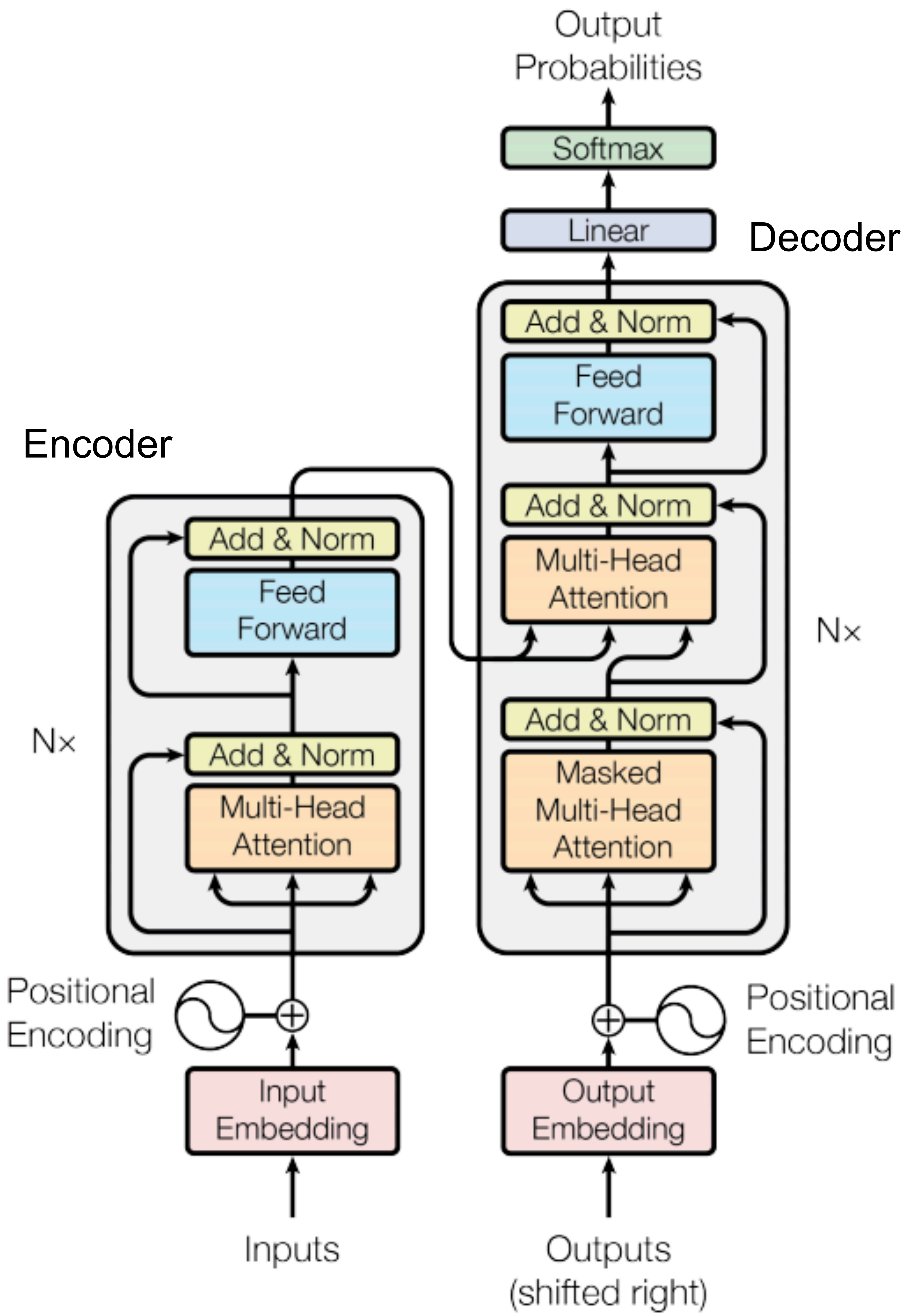


Transformer blocks

- 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!



Transformer architecture: Zooming back out



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

Transformer implementation

```python
class MultiHeadedAttention(nn.Module):
    """
    Multi-Head Attention module from "Attention is All You Need"
    Implementation modified from OpenNMT-py.
    https://github.com/OpenNMT/OpenNMT-py
    """

    def __init__(self, num_heads: int, size: int, dropout: float = 0.1):
        """
        Create a multi-headed attention layer.
        :param num_heads: the number of heads
        :param size: model size (must be divisible by num_heads)
        :param dropout: probability of dropping a unit
        """
        super(MultiHeadedAttention, self).__init__()

        assert size % num_heads == 0

        self.head_size = head_size = size // num_heads
        self.model_size = size
        self.num_heads = num_heads

        self.k_layer = nn.Linear(size, num_heads * head_size)
        self.v_layer = nn.Linear(size, num_heads * head_size)
        self.q_layer = nn.Linear(size, num_heads * head_size)

        self.output_layer = nn.Linear(size, size)
        self.softmax = nn.Softmax(dim=-1)
        self.dropout = nn.Dropout(dropout)
```

nn.Linear:
Learnable params

Transformer implementation

```python
def forward(
    self, k: Tensor, v: Tensor, q: Tensor, rel_mouth_times=None, mask: Tensor = None
):
    """
    Computes multi-headed attention.
    :param k: keys   [B, K, D] with K being the sentence length.
    :param v: values [B, K, D]
    :param q: query  [B, Q, D] with Q being the sentence length.
    :param mask: optional mask [B, 1, K]
    :return:
    """
    batch_size = k.size(0)  # B
    num_heads = self.num_heads  # H
    # project the queries (q), keys (k), and values (v)
    k = self.k_layer(k)
    v = self.v_layer(v)
    q = self.q_layer(q)

    # reshape q, k, v for our computation to [B, H, ..., D/H]
    k = k.view(batch_size, -1, num_heads, self.head_size).transpose(1, 2)
    v = v.view(batch_size, -1, num_heads, self.head_size).transpose(1, 2)
    q = q.view(batch_size, -1, num_heads, self.head_size).transpose(1, 2)

    # compute scores
    q = q / math.sqrt(self.head_size)
```

$$K = XW_K$$
$$V = XW_V$$
$$Q = XW_Q$$

Transformer implementation

```python
# [B, H, Q, K]
scores = torch.matmul(q, k.transpose(2, 3))

# apply the mask (if we have one)
# we add a dimension for the heads to it below: [B, 1, 1, K]
if mask is not None:
    scores = scores.masked_fill(~mask.unsqueeze(1), float("-inf"))

# apply attention dropout and compute context vectors.
# [B, H, Q, K]
attention_map = self.softmax(scores)
attention = self.dropout(attention_map)

# get context vector (select values with attention)
# [B, H, Q, D/H]
context = torch.matmul(attention, v)
# reshape back to [B, Q, D]
context = (
    context.transpose(1, 2)
    .contiguous()
    .view(batch_size, -1, num_heads * self.head_size)
)

# [B, Q, D]
output = self.output_layer(context)

return output, attention_map
```

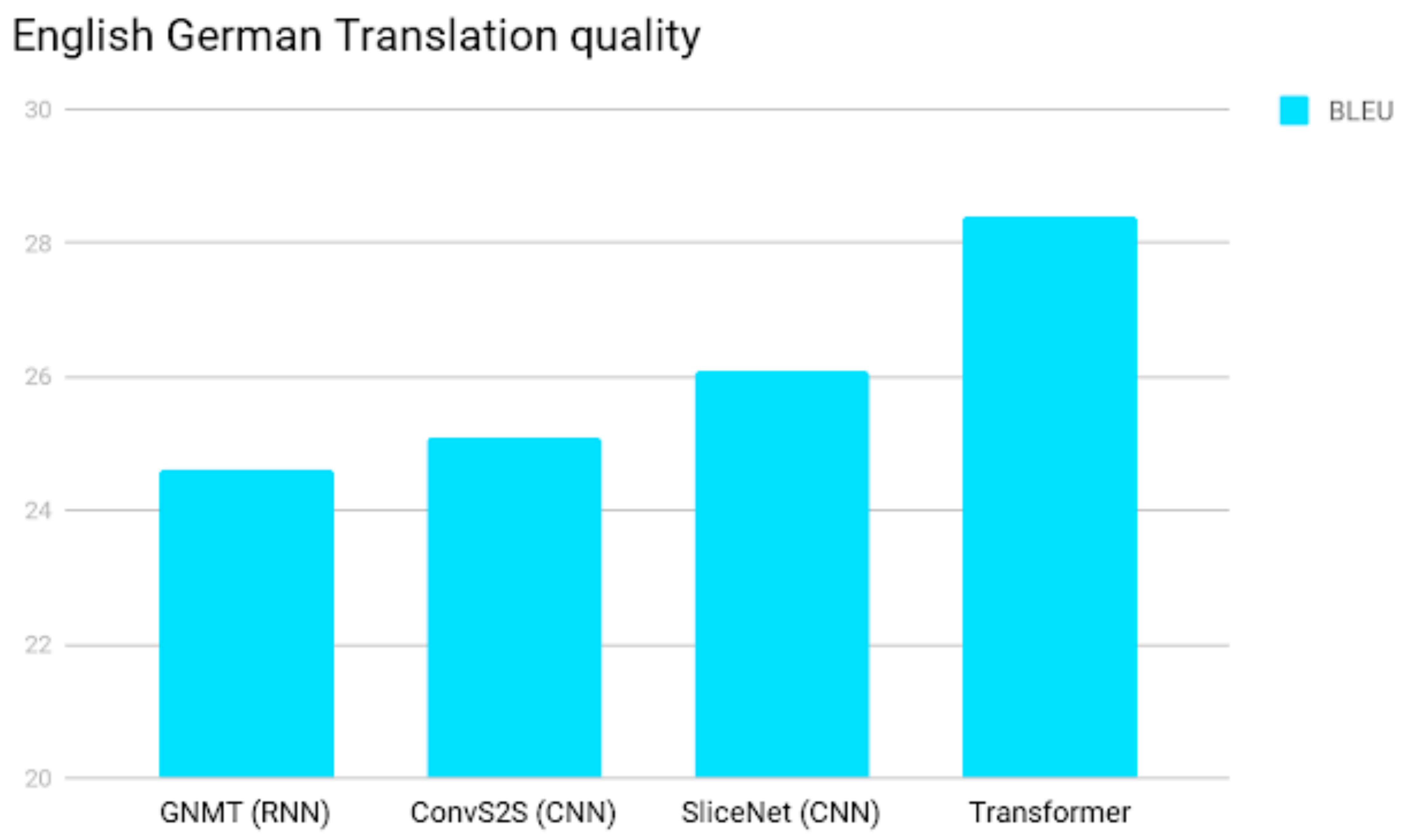
$$E = QK^T / \sqrt{D}$$

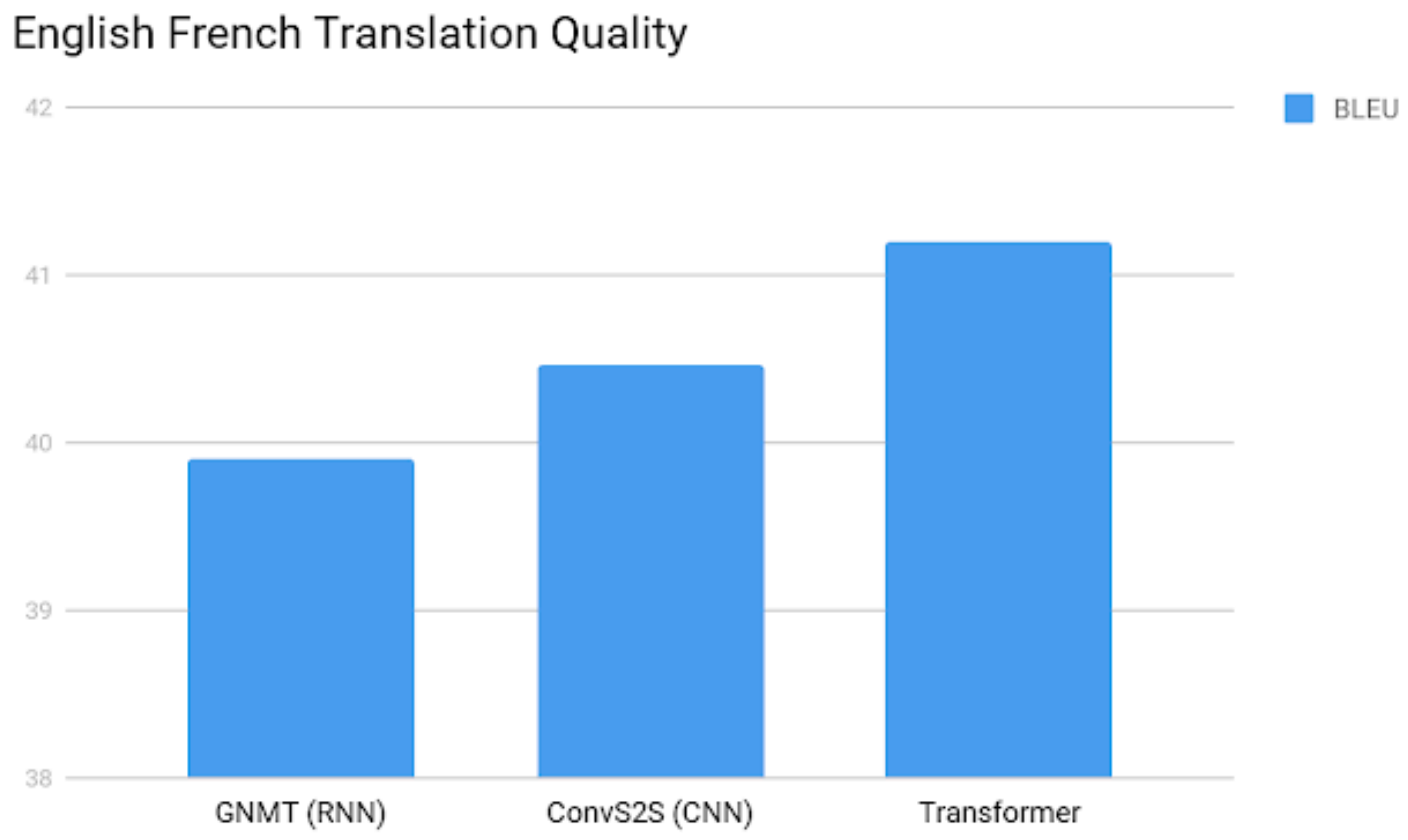
$$A = \mathrm{softmax}(E, \mathrm{dim} = 1)$$

$$Y = AV$$

```python
class TransformerEncoderLayer(nn.Module):
    """
    One Transformer encoder layer has a Multi-head attention layer plus
    a position-wise feed-forward layer.
    """

    def __init__(
        self, size: int = 0, ff_size: int = 0, num_heads: int = 0, dropout: float = 0.1
    ):
        """
        A single Transformer layer.
        :param size:
        :param ff_size:
        :param num_heads:
        :param dropout:
        """
        super(TransformerEncoderLayer, self).__init__()

        self.layer_norm = nn.LayerNorm(size, eps=1e-6)
        self.src_src_att = MultiHeadedAttention(num_heads, size, dropout=dropout)
        self.feed_forward = PositionwiseFeedForward(
            size, ff_size=ff_size, dropout=dropout
        )
        self.dropout = nn.Dropout(dropout)
        self.size = size

    # pylint: disable=arguments-differ
    def forward(self, x: Tensor, mask: Tensor) -> Tensor:
        """
        Forward pass for a single transformer encoder layer.
        First applies layer norm, then self attention,
        then dropout with residual connection (adding the input to the result),
        and then a position-wise feed-forward layer.
        :param x: layer input
        :param mask: input mask
        :return: output tensor
        """
        x_norm = self.layer_norm(x)
        h, att_map_src_src = self.src_src_att(k=x_norm, v=x_norm, q=x_norm, mask=mask)
        h = self.dropout(h) + x
        o = self.feed_forward(h)
        return o
```

Original transformer results on machine translation





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

Agenda

- **1. Recap: Bag of Visual Words, Analogy with NNs**
- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs
- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures
- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers
- **5. Beyond classification**

Attention & Transformers

- Basic transformer model
- **Image transformers**

Image transformer – Google

Self-attention only locally

- Image generation and super-resolution with 32x32 output, attention restricted to local neighborhoods

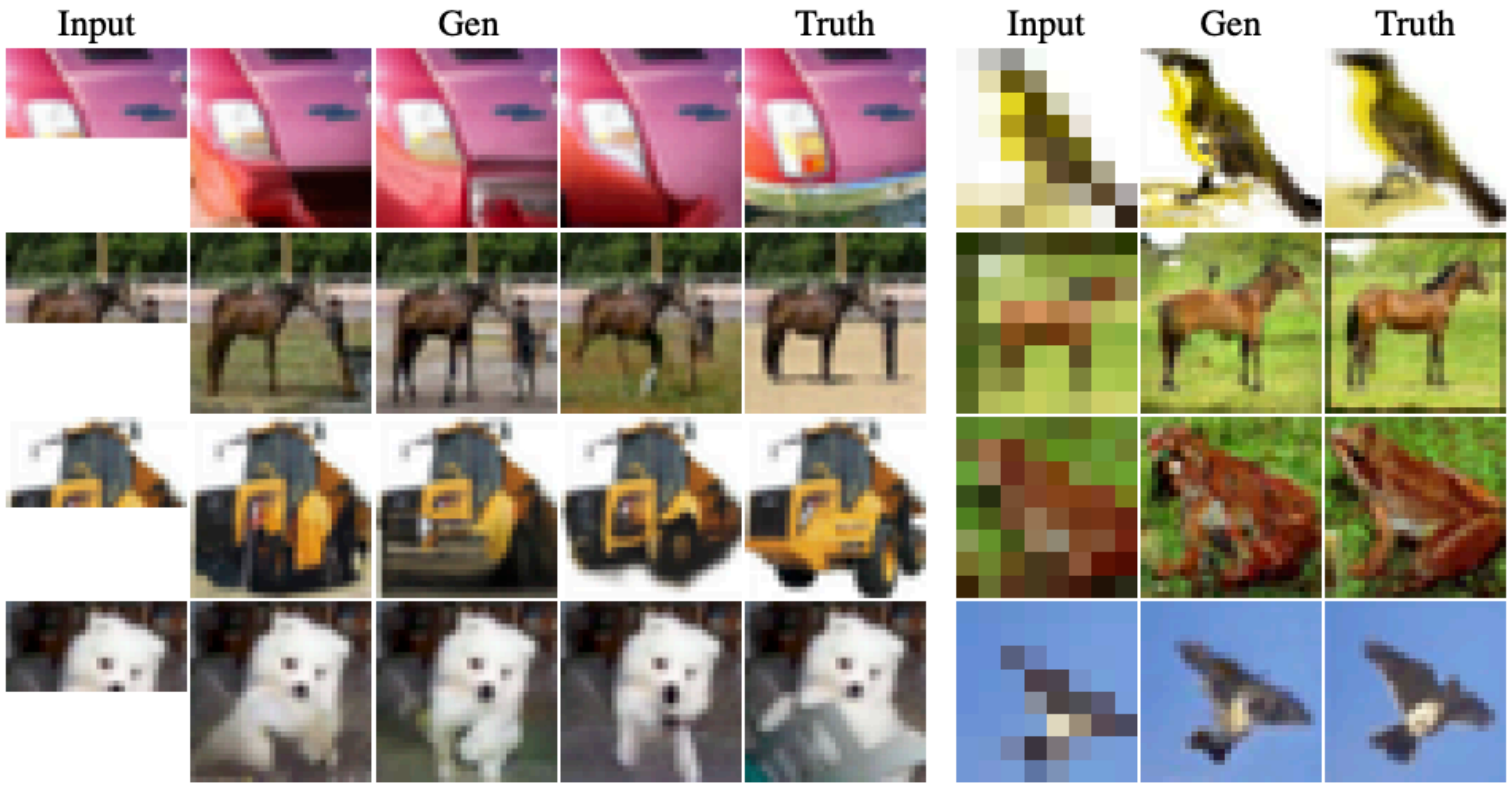


Table 2. On the left are image completions from our best conditional generation model, where we sample the second half. On the right are samples from our four-fold super-resolution model trained on CIFAR-10. Our images look realistic and plausible, show good diversity among the completion samples and observe the outputs carry surprising details for coarse inputs in super-resolution.

N. Parmar et al., [Image transformer](), ICML 2018

Sparse transformers – OpenAI

scalable approximations to global self-attention in order to be applicable to images

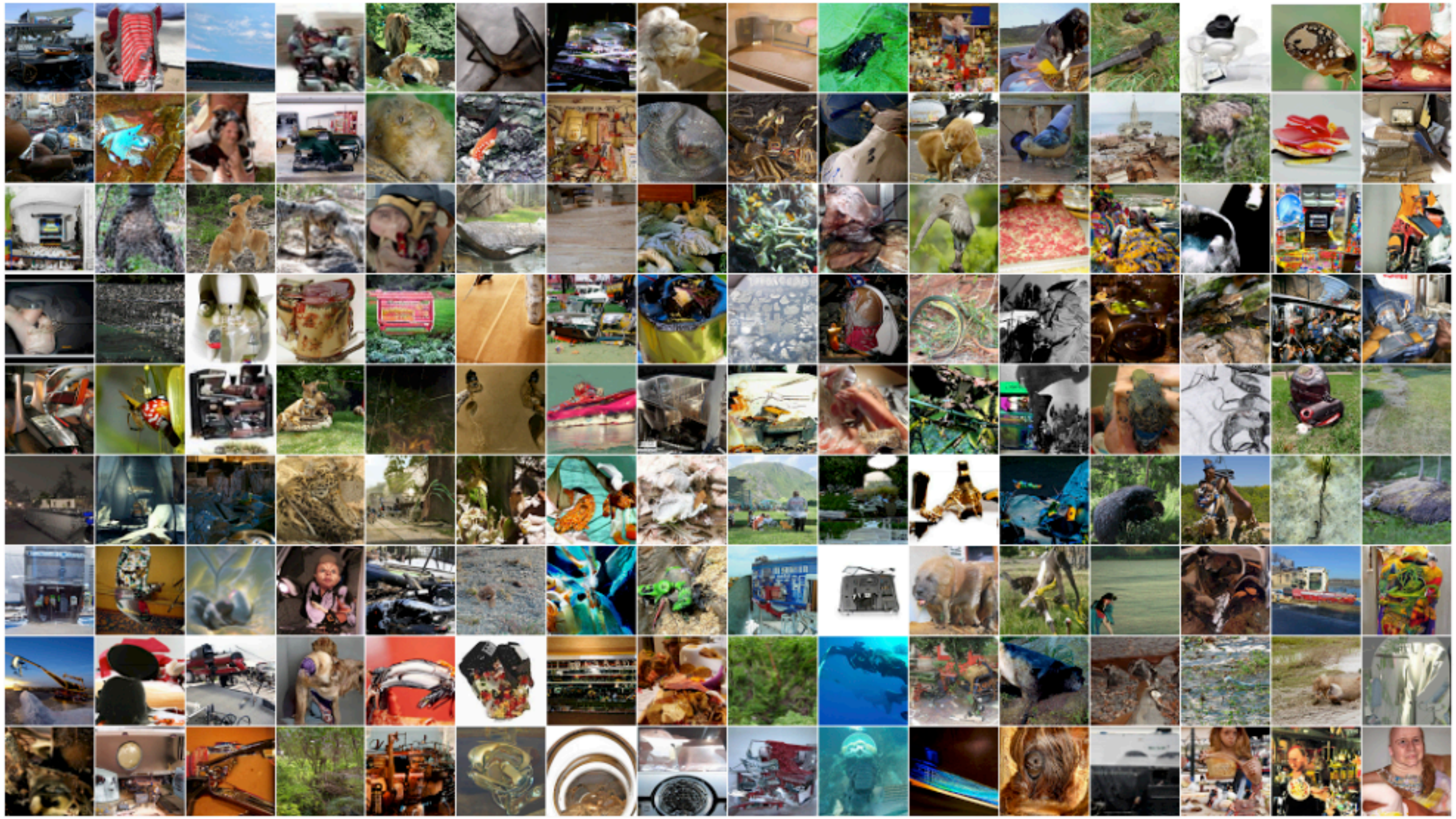
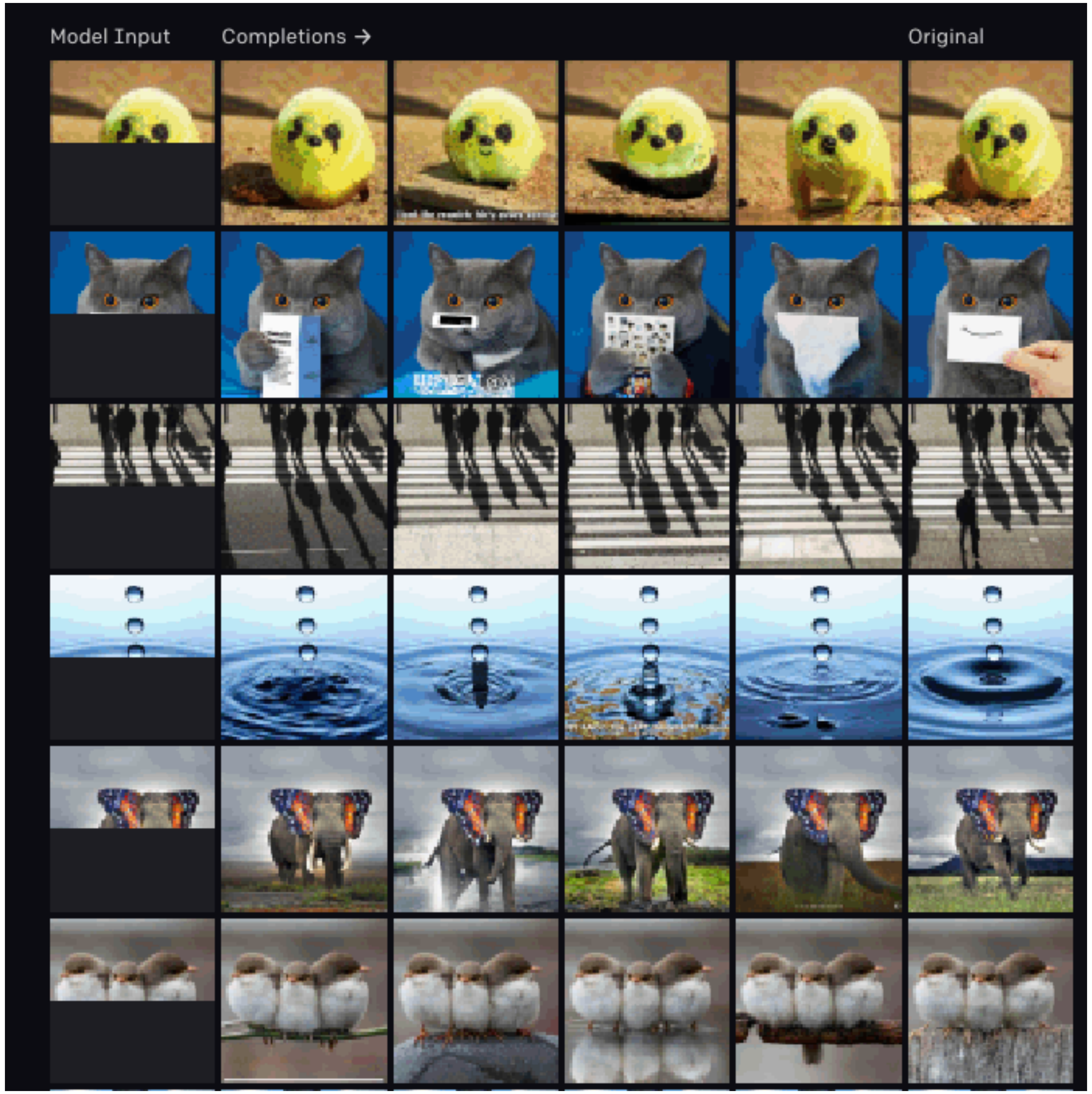


Figure 5. Unconditional samples from ImageNet 64x64, generated with an unmodified softmax temperature of 1.0. We are able to learn long-range dependencies directly from pixels without using a multi-scale architecture.

R. Child et al., [Generating Long Sequences with Sparse Transformers](), arXiv 2019

Image GPT* – OpenAI

works on reduced resolutions



https://openai.com/blog/image-gpt/

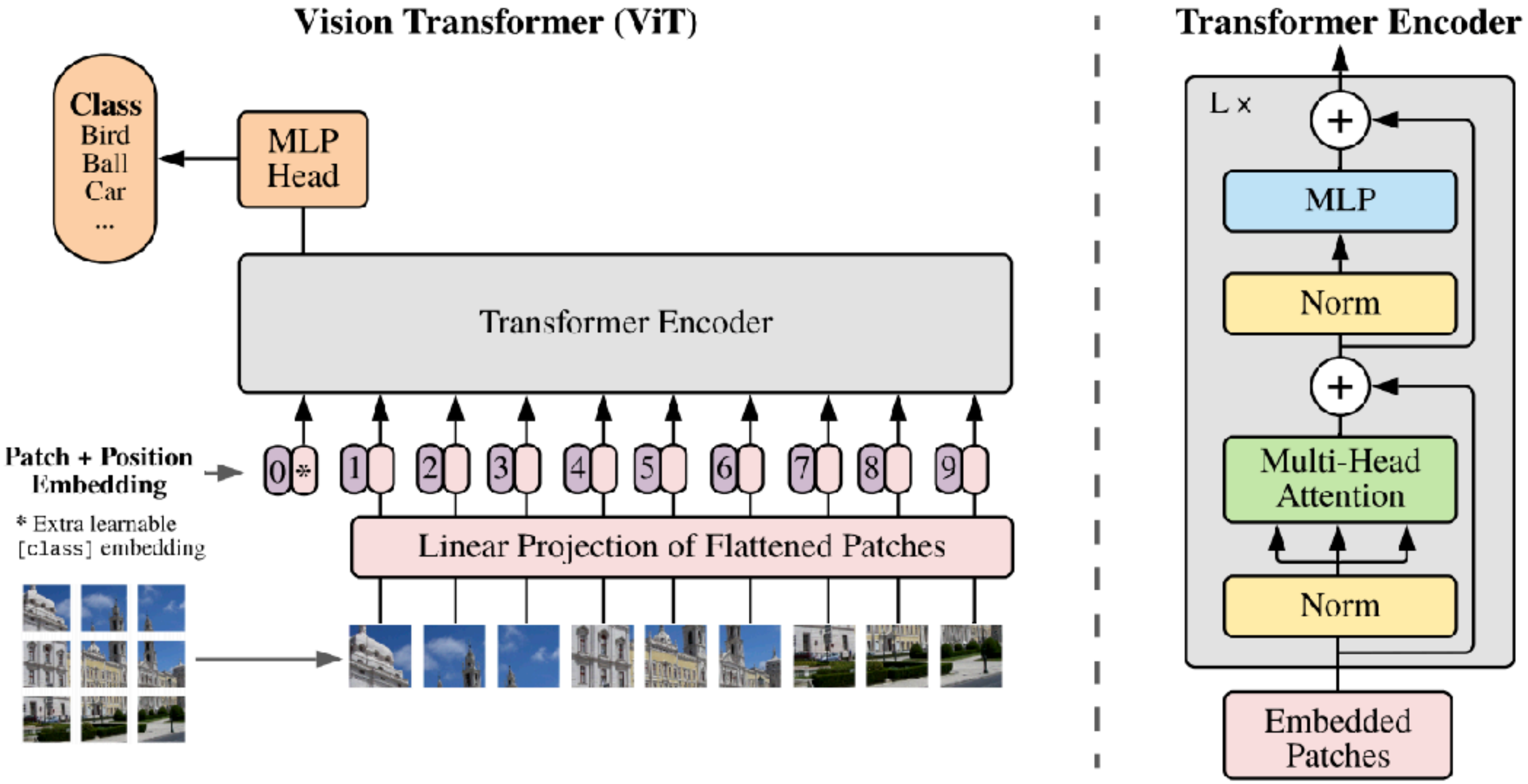
M. Chen et al., [Generative pretraining from pixels](), ICML 2020

*GPT: Generative pre-trained Transformer

Vision transformer (ViT) - Google

Full resolution

- 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

Vision transformer (ViT)

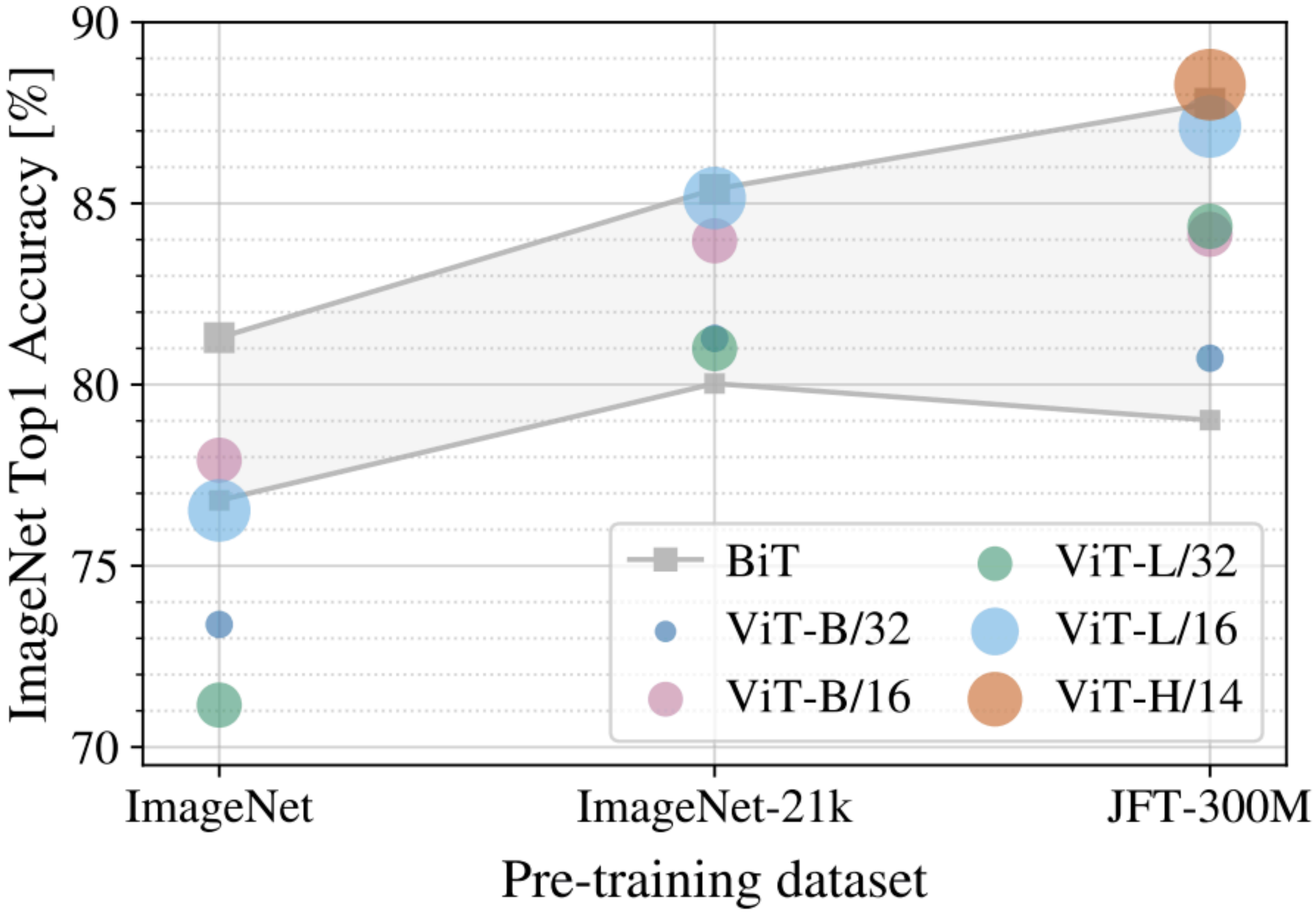


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

BiT: Big Transfer (ResNet)
ViT: Vision Transformer (Base/Large/Huge, patch size of 14x14, 16x16, or 32x32)

Internal Google dataset (not public)

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

Masked autoencoders are scalable vision learners

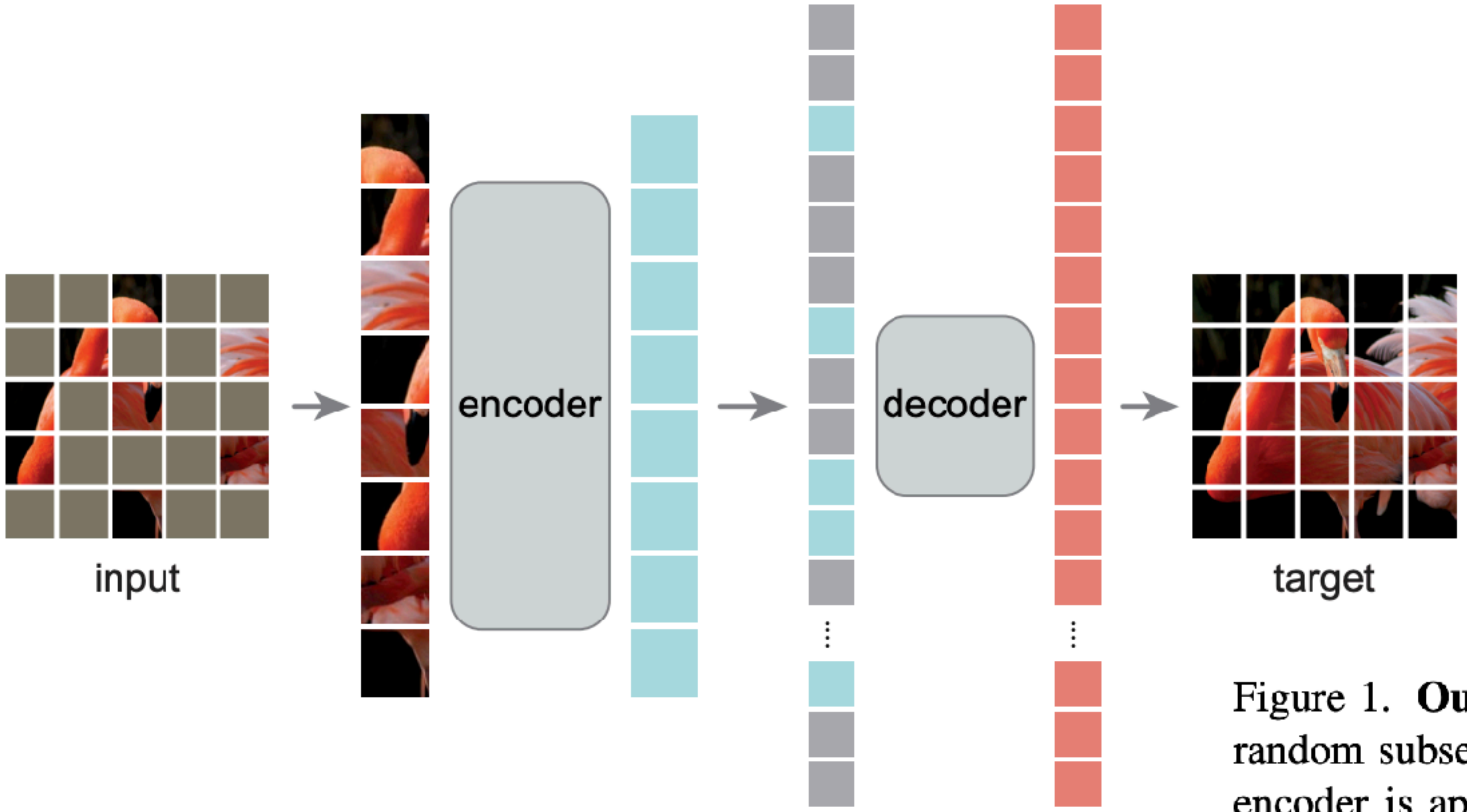


Figure 1. **Our MAE architecture**. During pre-training, a large random subset of image patches (*e.g.*, 75%) is masked out. The encoder is applied to the small subset of *visible patches*. Mask tokens are introduced *after* the encoder, and the full set of encoded patches and mask tokens is processed by a small decoder that reconstructs the original image in pixels. After pre-training, the decoder is discarded and the encoder is applied to uncorrupted images to produce representations for recognition tasks.

K. He et al. [Masked autoencoders are scalable vision learners](). arXiv 2021

Masked autoencoders are scalable vision learners

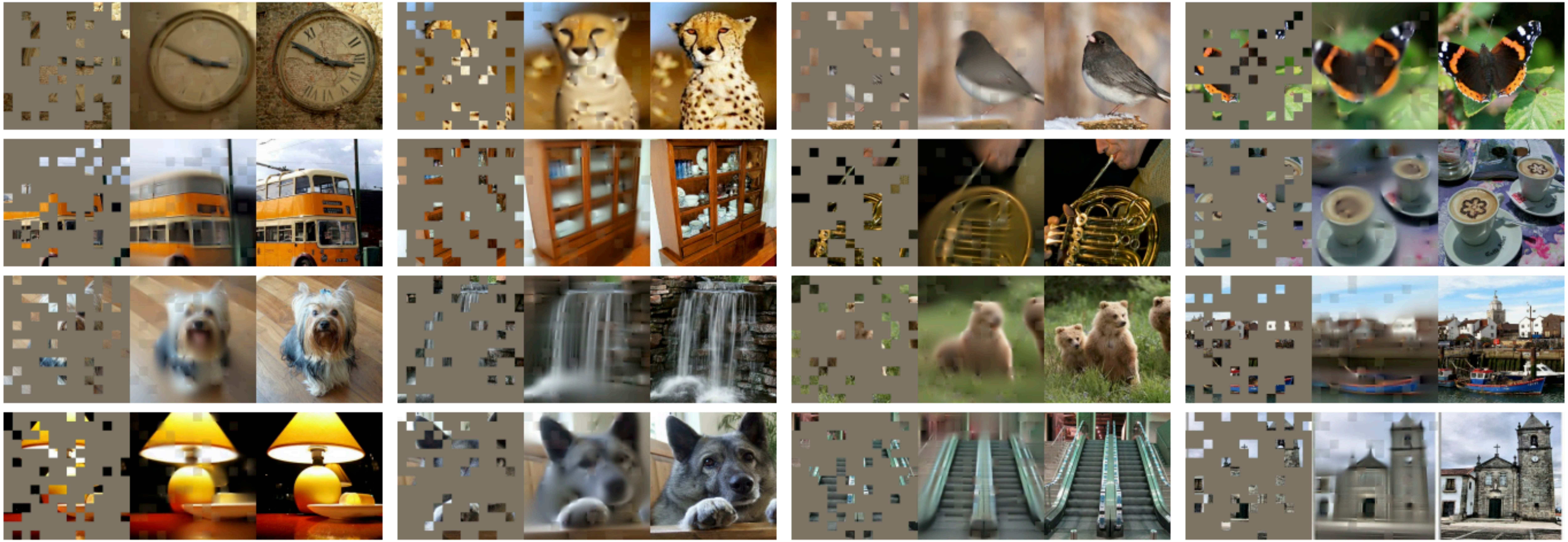
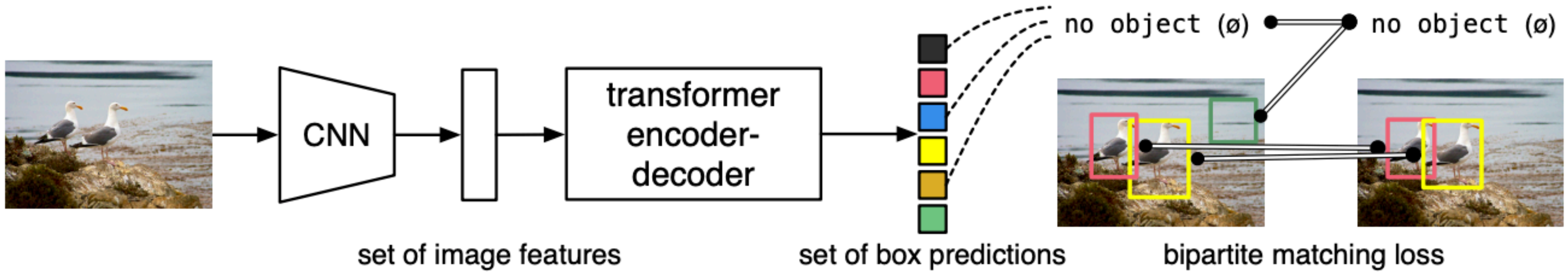


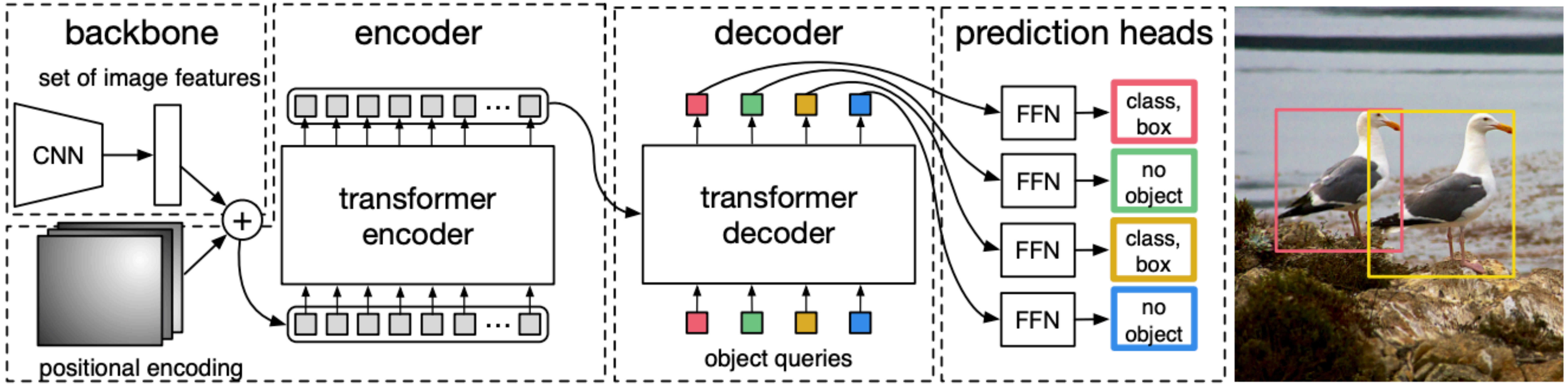
Figure 2. Example results on ImageNet *validation* images. For each triplet, we show the masked image (left), our MAE reconstruction† (middle), and the ground-truth (right). The masking ratio is 80%, leaving only 39 out of 196 patches. More examples are in the appendix. †*As no loss is computed on visible patches, the model output on visible patches is qualitatively worse. One can simply overlay the output with the visible patches to improve visual quality. We intentionally opt not to do this, so we can more comprehensively demonstrate the method's behavior.*

K. He et al. [Masked autoencoders are scalable vision learners](). arXiv 2021

Detection Transformer (DETR)

- Hybrid of CNN and transformer, aimed at standard recognition task





N. Carion et al., [End-to-end object detection with transformers](), ECCV 2020

Do we need attention?

Posted by u/L-MK 7 months ago 3 4 4

576 [R] Do You Even Need Attention? A Stack of Feed-Forward Layers Does Surprisingly Well on ImageNet

Research

https://www.reddit.com/r/MachineLearning/comments/n62qhn/r_do_you_even_need_attention_a_stack_of/

Do we need attention?

MLP-Mixer: An all-MLP Architecture for Vision

Ilya Tolstikhin*, Neil Houlsby*, Alexander Kolesnikov*, Lucas Beyer*,
Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner,
Daniel Keysers, Jakob Uszkoreit, Mario Lucic, Alexey Dosovitskiy
*equal contribution
Google Research, Brain Team
{tolstikhin, neilhoulsby, akolesnikov, lbeyer,
xzhai, unterthiner, jessicayung†, andstein,
keysers, usz, lucic, adosovitskiy}@google.com

4 May 2021

Do You Even Need Attention? A Stack of Feed-Forward Layers Does Surprisingly Well on ImageNet

Luke Melas-Kyriazi
Oxford University
lukemk@robots.ox.ac.uk

6 May 2021

Pay Attention to MLPs

Hanxiao Liu, Zihang Dai, David R. So, Quoc V. Le
Google Research, Brain Team
{hanxiaol,zihangd,davidso,qvl}@google.com

17 May 2021

Recent Hype#2: MLPs (!)

- Back to basics
- MLPs perform similar to Transformers while being more efficient
- CNNs and MLPs complexity linear with the number of input pixels, Transformers quadratic

MLP-Mixer: An all-MLP Architecture for Vision

Ilya Tolstikhin*, Neil Houlsby*, Alexander Kolesnikov*, Lucas Beyer*,
Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner,
Daniel Keysers, Jakob Uszkoreit, Mario Lucic, Alexey Dosovitskiy
*equal contribution
Google Research, Brain Team
{tolstikhin, neilhoulsby, akolesnikov, lbeyer,
xzhai, unterthiner, jessicayung†, andstein,
keysers, usz, lucic, adosovitskiy}@google.com

4 May 2021

Do You Even Need Attention? A Stack of Feed-Forward Layers Does Surprisingly Well on ImageNet

Luke Melas-Kyriazi
Oxford University
lukemk@robots.ox.ac.uk

6 May 2021

Pay Attention to MLPs

Hanxiao Liu, Zihang Dai, David R. So, Quoc V. Le
Google Research, Brain Team
{hanxiaol,zihangd,davidso,qvl}@google.com

17 May 2021

Summary: Beyond CNNs

- CNNs (convolution), Transformers (attention), MLPs (fully connected)
- There is no answer to which architecture is better.
- Often depends on the data.
- If you have infinite data, more complex can be better (e.g., MLP ~ Transformers > CNN).
- Similar performance can be obtained with more efficient models (e.g., MLP ~ Transformers)
- It is possible there will be newer/better architectures/ hypes before you graduate. Stay tuned.

Agenda

- **1. Recap: Bag of Visual Words, Analogy with NNs**

- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs

- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures

- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers

- **5. Beyond classification - preview**

The field makes progress

Beyond Classification

Computer vision tasks

Extracting meaning from visual signals*

Object recognition,
Object detection,
Pixel-level segmentation,
3D localization,
etc.

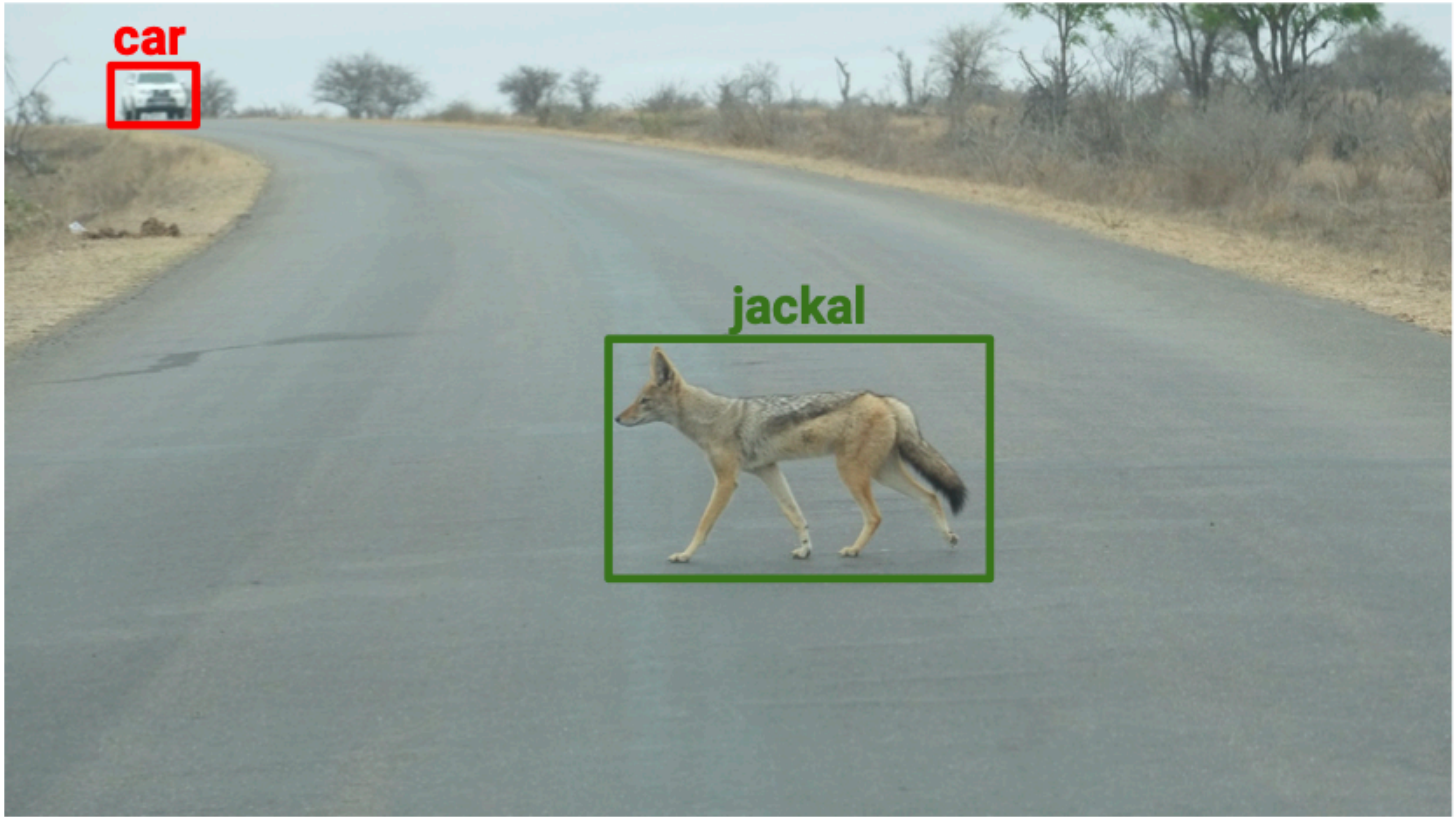
*Visual signal: Image, video, depth, 3D point cloud, MRI, scans, ...

Slide credit: Naila Murray

Example tasks

Slide credit: Naila Murray

Object recognition and localization (detection)



Slide credit: Naila Murray

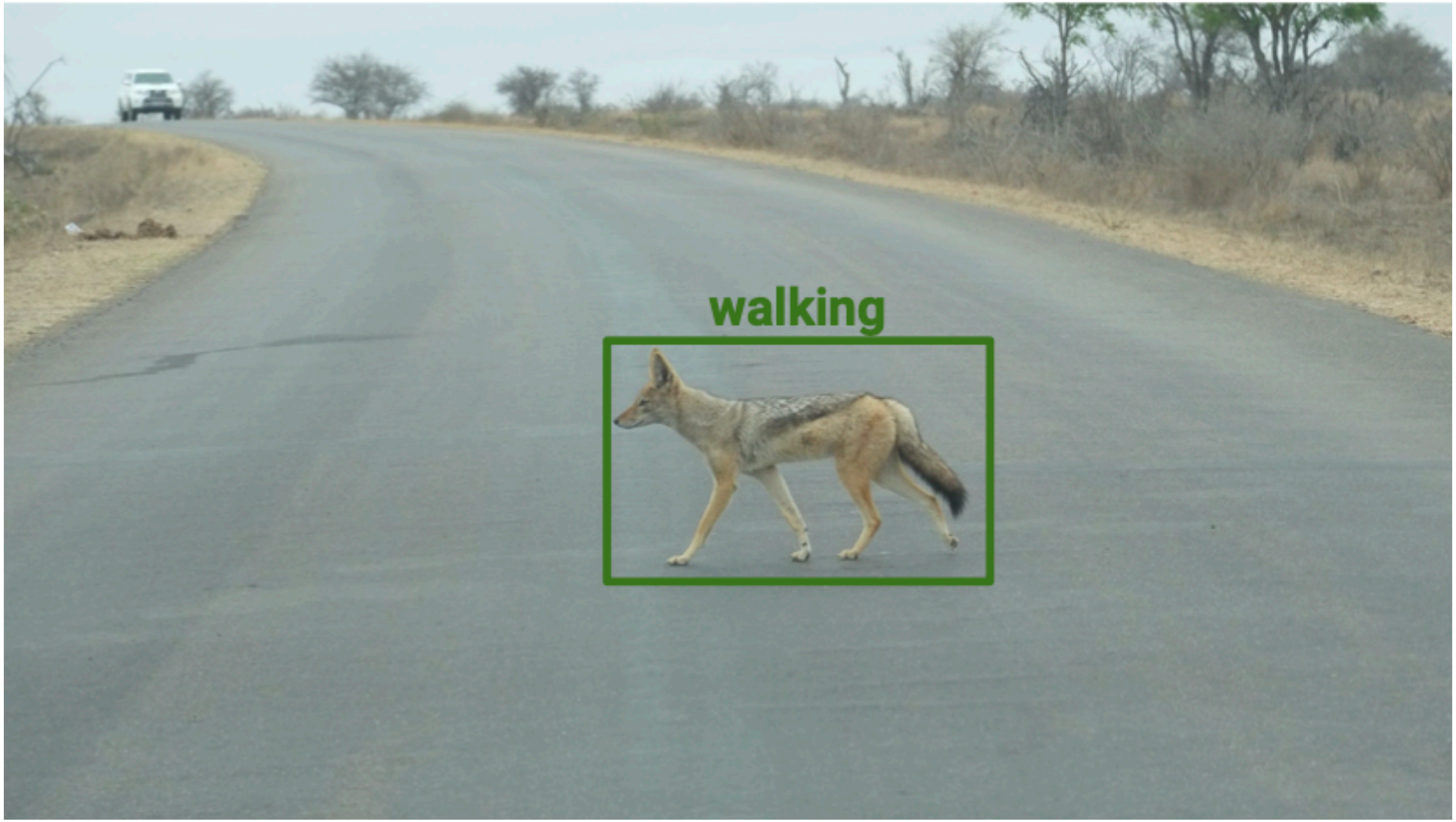
Visual question answering

Q: Is this an outdoor scene?
A: Yes

Q: What is the weather like?
A: Cloudy but dry

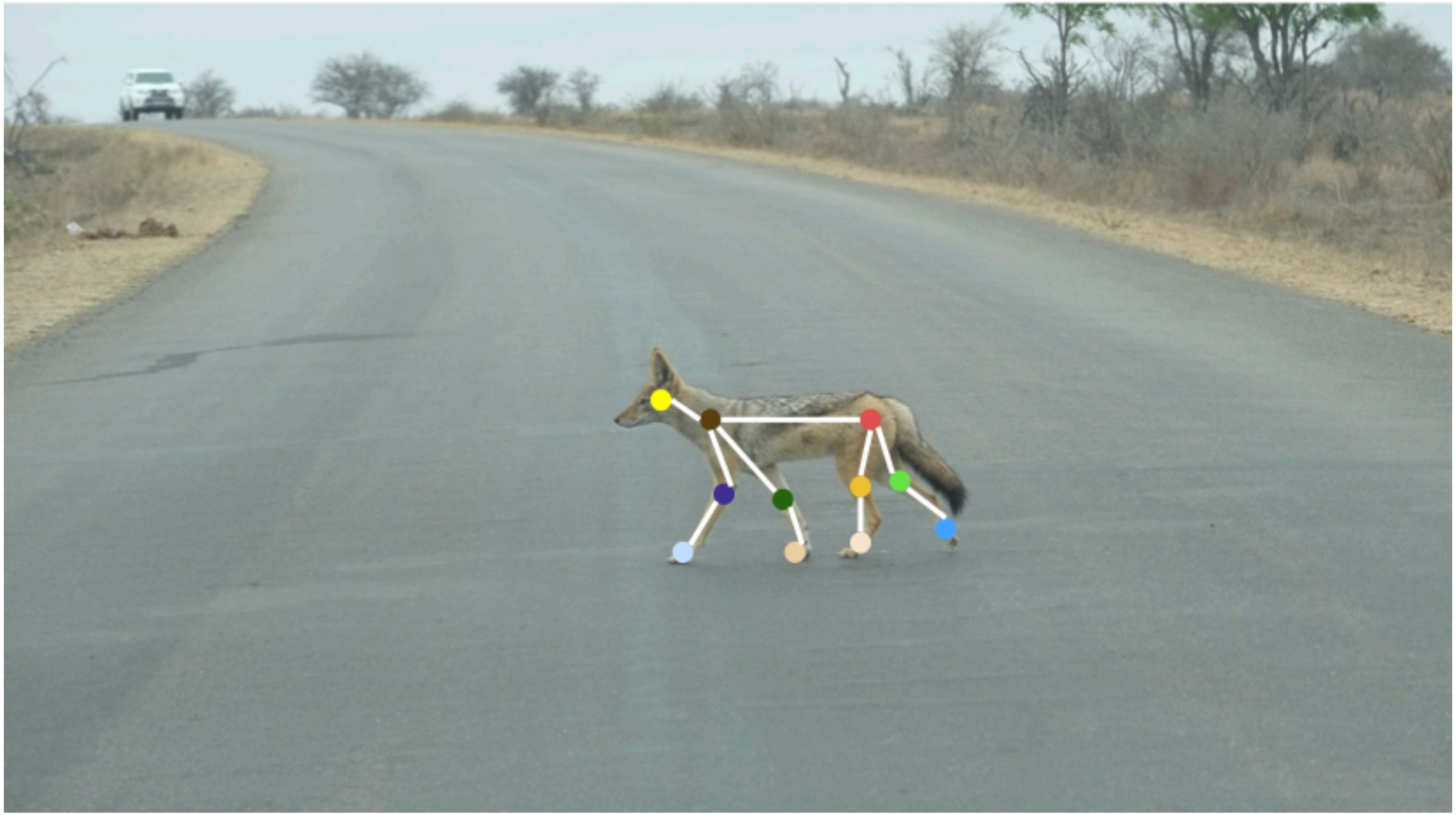
Slide credit: Naila Murray

Activity recognition



Slide credit: Naila Murray

Pose estimation



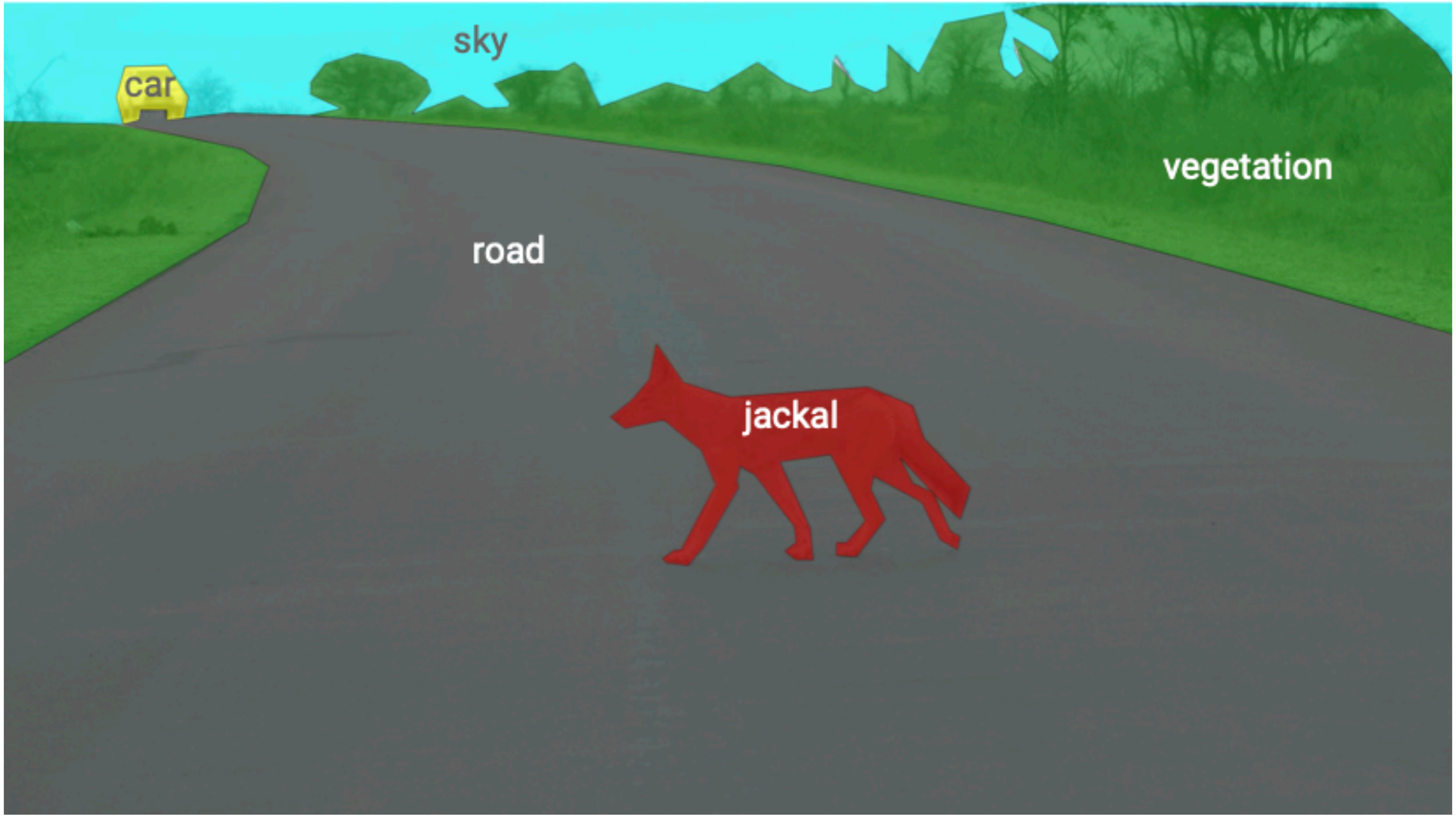
Slide credit: Naila Murray

Captioning

A jackal walking across a rural asphalt road

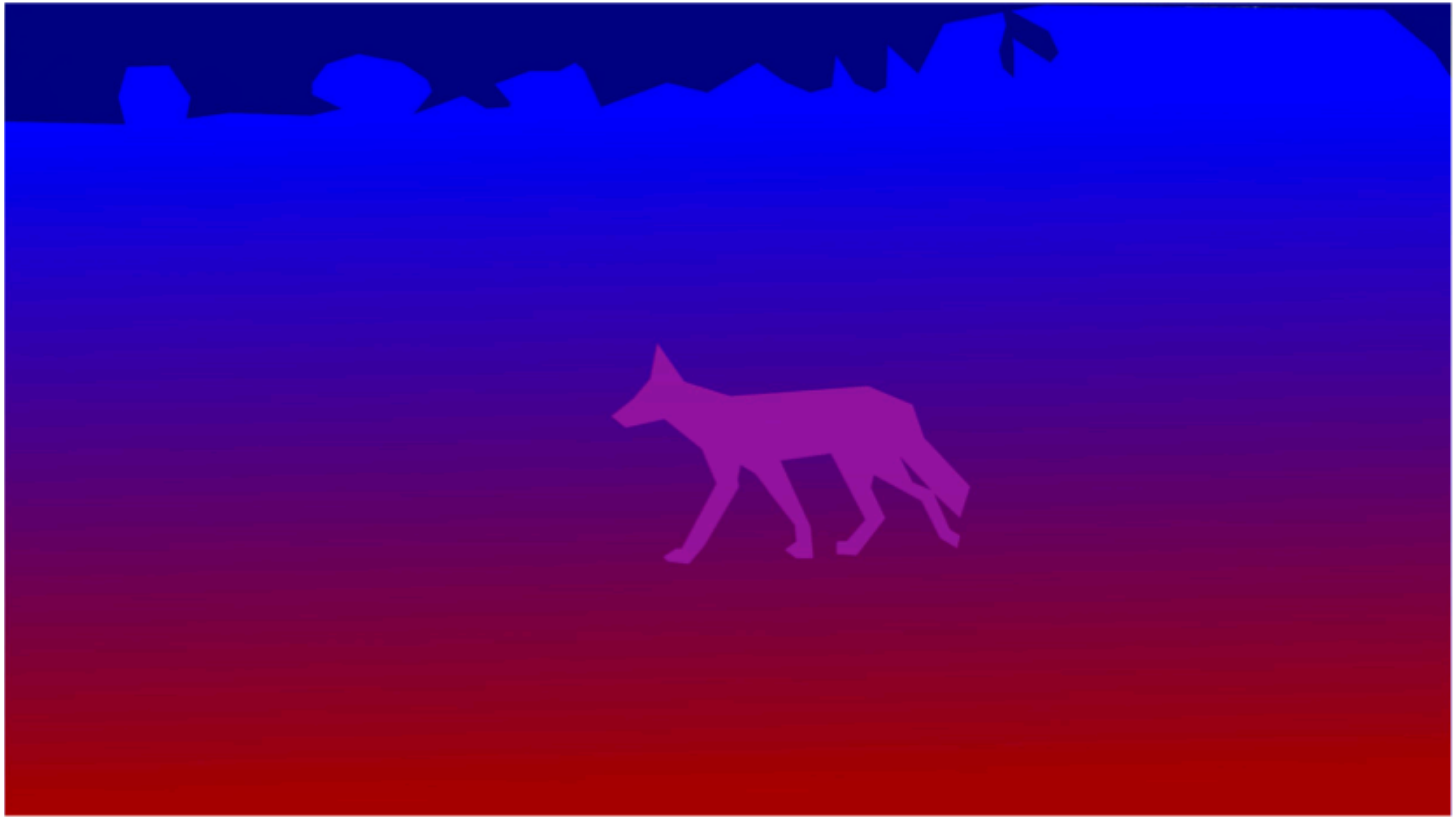
Slide credit: Naila Murray

Semantic segmentation



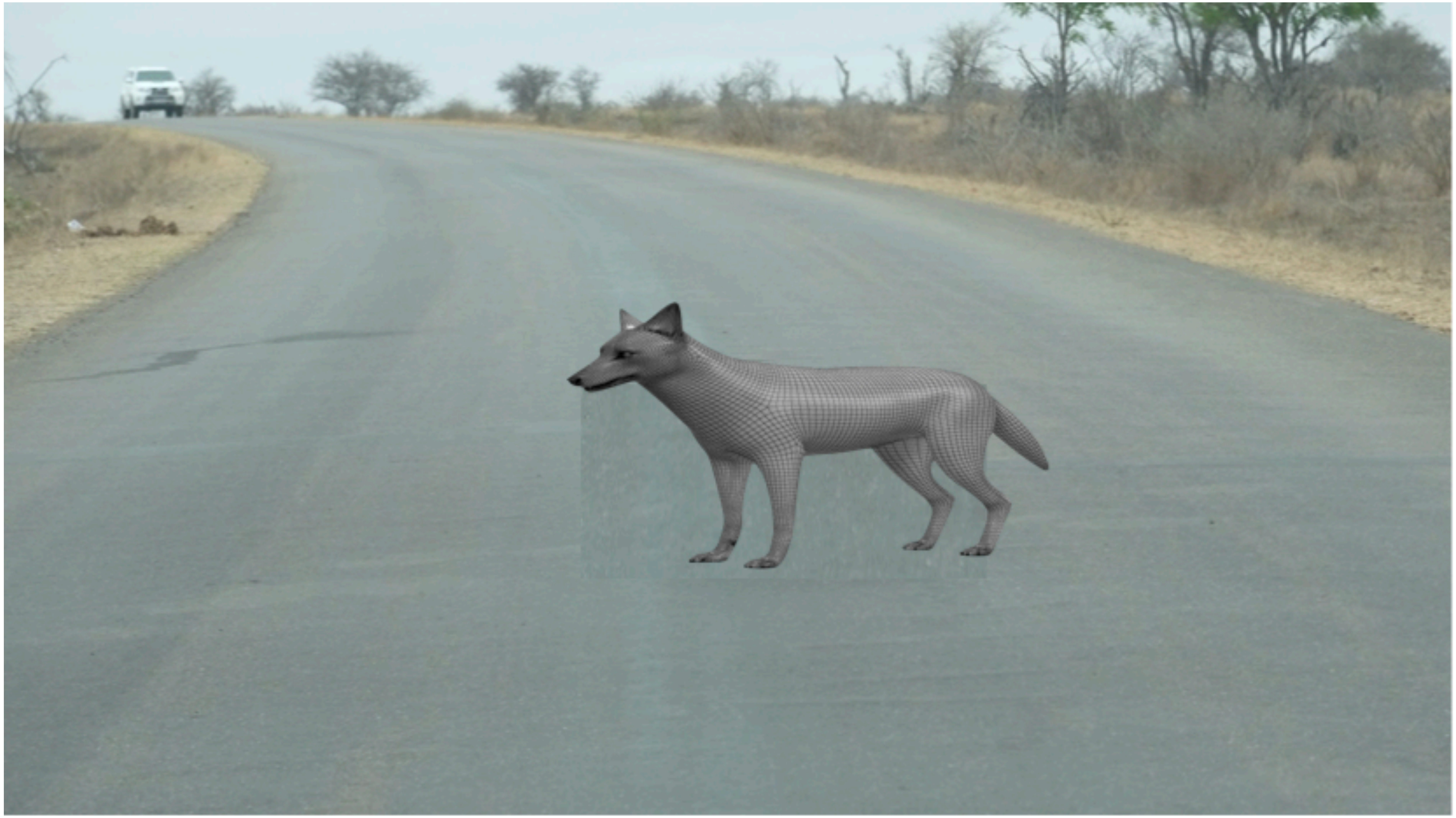
Slide credit: Naila Murray

Depth estimation



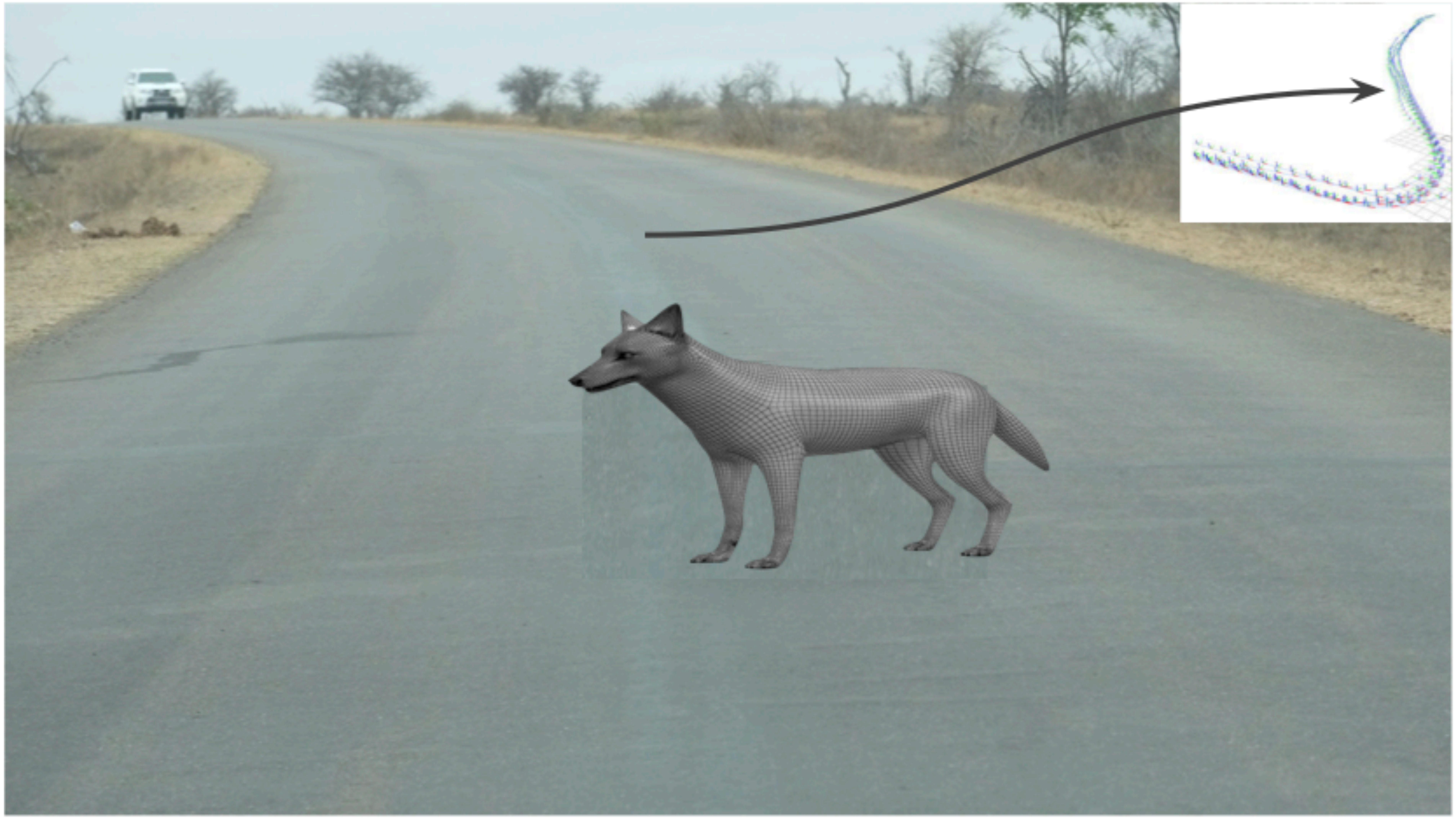
Slide credit: Naila Murray

3D shape estimation



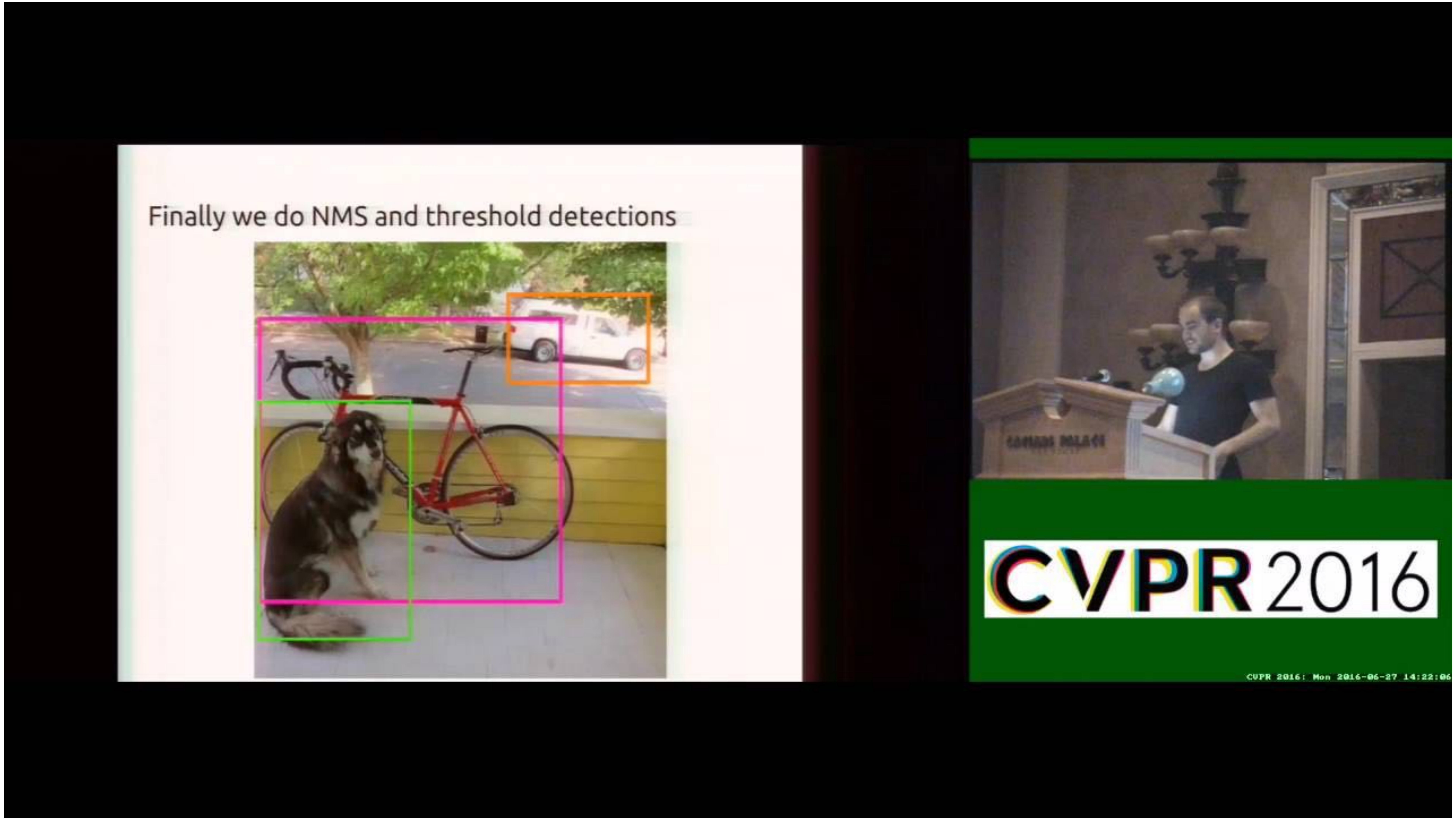
Slide credit: Naila Murray

Visual localization



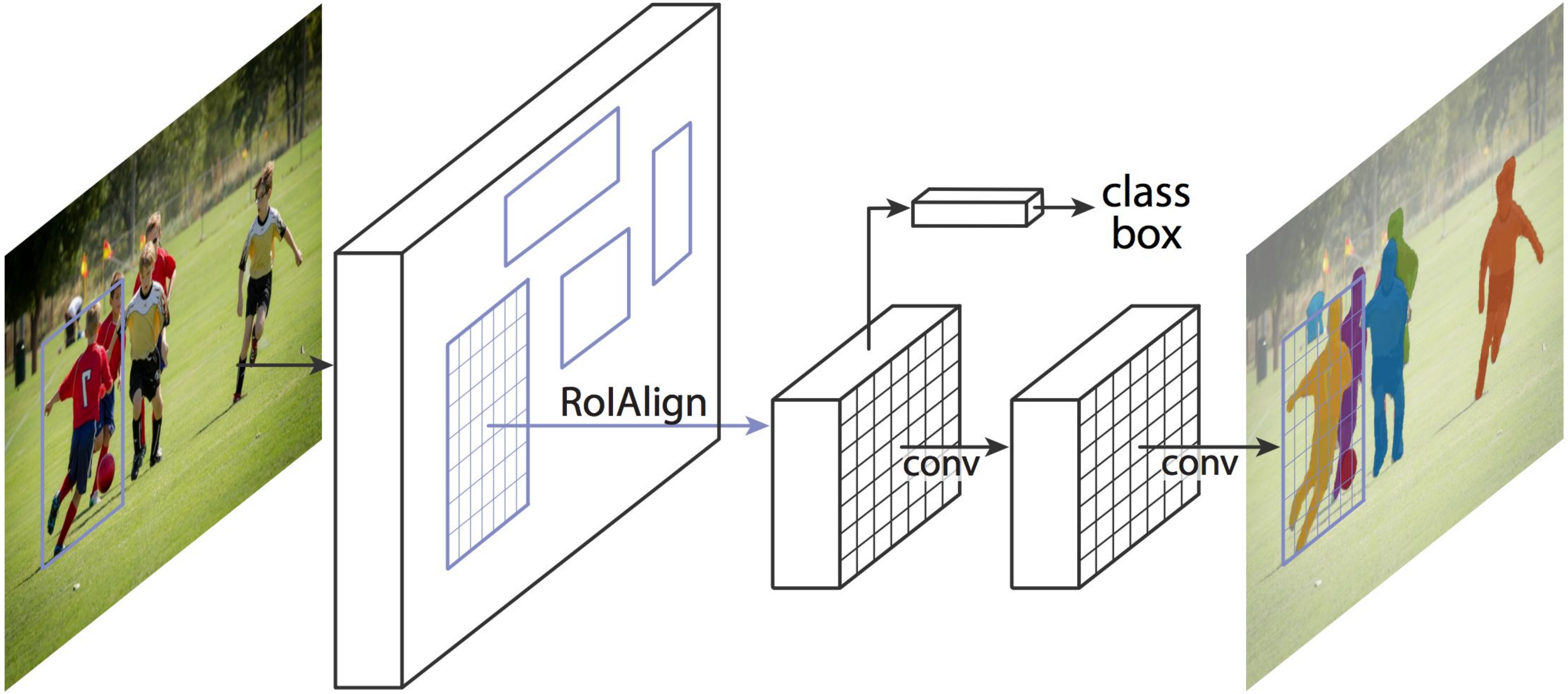
Slide credit: Naila Murray

Object detection



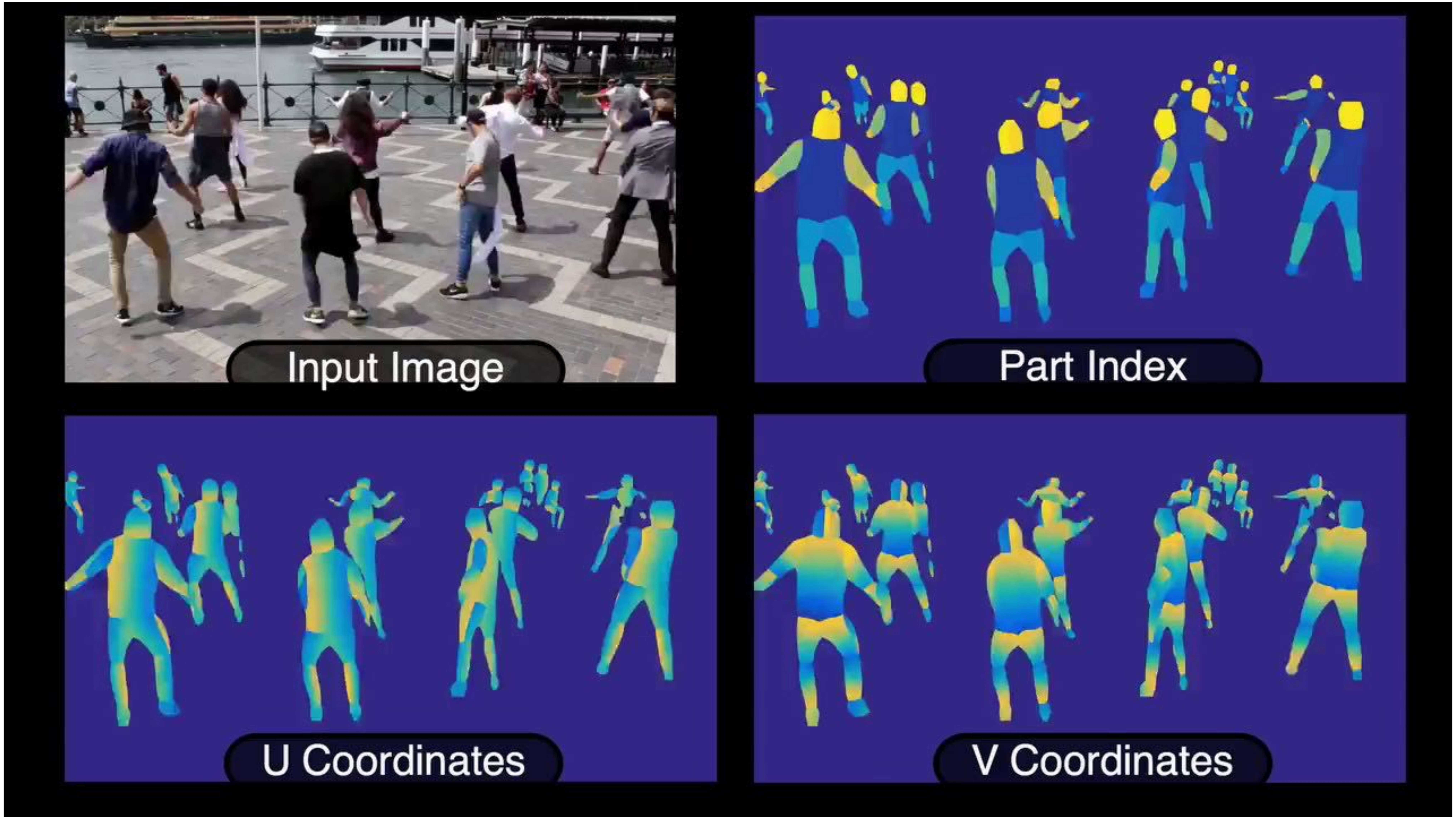
Redmon et al. YOLO, CVPR 2016

Segmentation



He et al. Mask R-CNN. ICCV 2017.

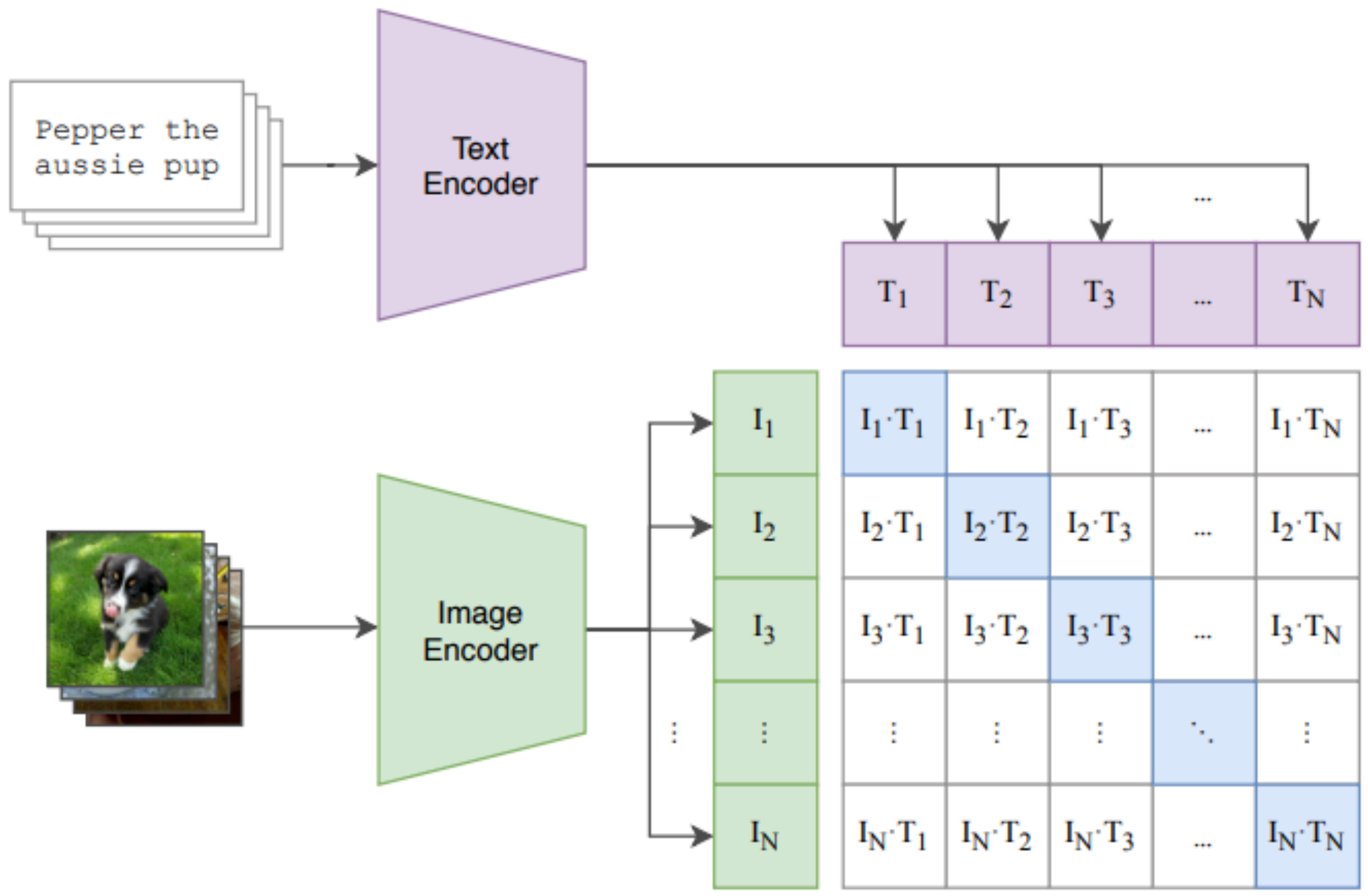
Human pose estimation



Guler et al. DensePose, CVPR 2018

Text-to-image retrieval

Contrastive Language-Image Pretraining (CLIP)



Contrastive objective: in a batch of N image-text pairs, classify each text string to the correct image and vice versa

A. Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021
https://openai.com/blog/clip/

Text-to-video retrieval

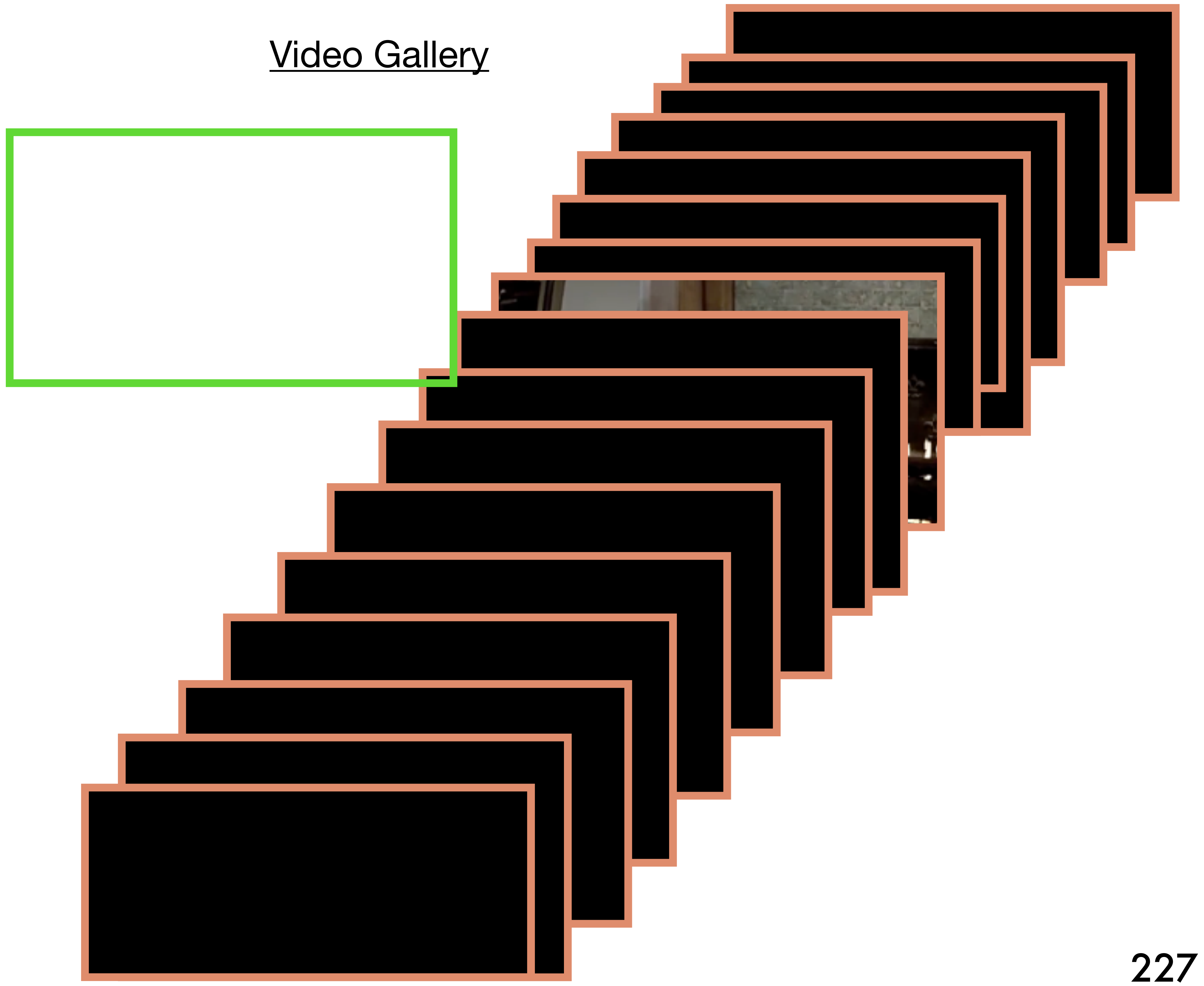
Text Query

Billy reveals the truth to Louis about the Duke’s bet which changed both their lives

Text-video retrieval model

Similarity: **0.89**

Video Gallery



VGG
UNIVERSITY OF OXFORD

❄️ Frozen in Time ⏳
🔍 Video Search Demo 🎬

e.g. empty street in nepa | Search

display: 8

Visual search of ~2.6M videos are based on research described in
Frozen in time: A joint video and image encoder for end-to-end retrieval.

[Bain, Nagrani, Varol, Zisserman, ICCV 2021]

Text-based image generation: DALL-E

(a) a tapir made of accordion. a tapir with the texture of an accordion.

(b) an illustration of a baby hedgehog in a christmas sweater walking a dog

(c) a neon sign that reads "backprop". a neon sign that reads "backprop". backprop neon sign

A. Ramesh et al., [Zero-Shot Text-to-Image Generation](https://openai.com/blog/dall-e/), ICML 2021
https://openai.com/blog/dall-e/

Summary of today

- **1. Recap: Bag of Visual Words, Analogy with NNs**

- **2. Neural networks (NNs) for computer vision:**
  - Applications
  - A brief history: from perceptron to MLPs to CNNs

- **3. Convolutional neural networks (CNNs)**
  - Standard layers
  - Recap: Training NNs
  - Visualizing CNNs
  - Pretraining & finetuning NNs
  - Typical CNN architectures

- **4. Beyond CNNs**
  - Attention & Transformer
  - Vision Transformers

- **5. Beyond classification**

Key elements of DL for CV

Initially in CV	❶ Model (i.e., architectural definition of connectivity and learnable parameters)
Next in CV	❷ Loss
These days in CV !	❸ Data
ML community	● Optimization algorithm (i.e., variations of SGD)

Still many open questions

- 3D
- Videos
- Visual perception in robotics